

ES7 and ES8 特性



爱在西元钱 (/u/74959923eaf0) [+ 关注](#)

2017.03.26 21:52* 字数 3162 阅读 55985 评论 17 喜欢 50

(/u/74959923eaf0)

最近我写了一篇博客文章,甚至做一篇关于ES6/ES2015在线课程。你猜怎么样? TC39-JavaScript最强势的监工-正在迈向ES8, 所以让我们了解下ES7 and ES8 (官方称ES2016 and ES2017), 幸运的是, 他们比ES6标准功能少好多好多, 真的, ES7只有2个新特性。

ES7 特性:

1. `Array.prototype.includes`
2. Exponentiation Operator(求幂运算)



Paste_Image.png

ES8在本文（2017年1月）之前尚未完成。但我们可以假设所有完成的提案（第4阶段）和大多数阶段3（更多的阶段在这里 (<https://link.jianshu.com?t=https%3A%2F%2Ftc39.github.io%2Fprocess-document>)和我的课程 (<https://link.jianshu.com?t=https%3A%2F%2Fnode.university%2Fp%2Fes6>)) 2017年（ES8）完成的提案：

1. `Object.values / Object.entries`
2. `String padding`(字符串填充)
3. `Object.getPrototypeOf`
4. 函数参数列表和调用中的尾逗号（Trailing commas）
5. 异步函数（Async Functions）

本文中我不会介绍stage 3的提案，但你可以在这里 (<https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Ftc39%2Fproposals%2Fblob%2Fmaster%2FREADME.md>)检查阶段1到3的建议的状态。

让我们深入了解建议和功能。

Array.prototype.includes

`Array.prototype.includes`用法都容易和简单。它是一个替代`indexOf`，开发人员用来检查数组中是否存在值，`indexOf`是一种尴尬的使用，因为它返回一个元素在数组中的位置或者-1当这样的元素不能被找到的情况下。所以它返回一个数字，而不是一个布尔值。开发人员需要实施额外的检查。在ES6，要检查是否存在值你需要做一些如下图所示小技巧，因为他们没有匹配到值，`Array.prototype.indexOf`返回-1变成了`true`（转换成`true`），但是当匹配的元素为0位置时候，该数组包含元素，却变成了`false`

```
let arr = ['react', 'angular', 'vue']

// WRONG
if (arr.indexOf('react')) { // 0 -> evaluates to false, definitely as we expected
  console.log('Can use React') // this line would never be executed
}

// Correct
if (arr.indexOf('react') !== -1) {
  console.log('Can use React')
}
```

或者使用一点点hack 位运算符 `~` 使代码更加紧凑一些，因为 `~`（位异或）对任何数字相当于 `-(a + 1)`：

```
let arr = ['react', 'angular', 'vue']

// Correct
if (~arr.indexOf('react')) {
  console.log('Can use React')
}
```

在ES7中使用 `includes` 代码如下:

```
let arr = ['react', 'angular', 'vue']

// Correct
if (arr.includes('react')) {
  console.log('Can use React')
}
```

开发者还能在字符串中使用 `includes` :

```
let str = 'React Quickly'

// Correct
if (str.toLowerCase().includes('react')) { // true
  console.log('Found "react"')
}
```

有趣的是, 许多JavaScript库已经实现 `includes` 或类似功能 `contains`

(但TC39决定不使用名称 `contains` 因为MooTools (<https://link.jianshu.com?t=https%3A%2F%2Fdiscuss.org%2Ftopic%2Fhaving-a-non-enumerable-array-prototype-contains-may-not-be-web-compatible>)) :

- jQuery: `$.inArray` (<https://link.jianshu.com?t=http%3A%2F%2Fapi.jquery.com%2Fjquery.inarray>)
- Underscore.js: `_.contains` (<https://link.jianshu.com?t=http%3A%2F%2Funderscorejs.org%2F%23contains>)
- Lodash: `_.includes` (<https://link.jianshu.com?t=https%3A%2F%2Flodash.com%2Fdocs%2F4.17.3%23includes>) (在版本3或者早期版本中是 `_.contains` 和 Underscore一样)
- CoffeeScript: `in` 操作(example (<https://link.jianshu.com?t=http%3A%2F%2Fbit.ly%2F2jGxfal>))

- Darf: `list.contains` (example (<https://link.jianshu.com?t=https%3A%2F%2Fgist.github.com%2Fanonymous%2Fb8e39109e5705a9a0ff7281c1af97195>))

除了增强了可读性语义化，实际上给开发者返回布尔值，而不是匹配的位置。

`includes` 也可以在 `NaN` (非数字)使用。最后，`includes` 第二可选参数 `fromIndex`，这对于优化是有好处的，因为它允许从特定位置开始寻找匹配。

更多例子：

```
console.log([1, 2, 3].includes(2)) // === true
console.log([1, 2, 3].includes(4)) // === false
console.log([1, 2, NaN].includes(NaN)) // === true
console.log([1, 2, -0].includes(+0)) // === true
console.log([1, 2, +0].includes(-0)) // === true
console.log(['a', 'b', 'c'].includes('a')) // === true
console.log(['a', 'b', 'c'].includes('a', 1)) // === false
```

总而言之，`includes`在一个数组或者列表中检查是否存在一个值，给任何开发人员带来简单性。

Exponentiation Operator(求幂运算)

求幂运算大多数是为开发者做一些数学计算，对于3D，VR，SVG还有数据可视化非常有用。在ES6或者早些版本，你不得不创建一个循环，创建一个递归函数或者使用

`Math.pow` ,如果你忘记了什么是指数,当你有相同数字（基数）自相相乘多次（指数）。例如，7的3次方是 $7*7*7$

所以在ES6/2015ES，你能使用 `Math.pow` 创建一个短的递归箭头函数

```
calculateExponent = (base, exponent) => base*(!exponent?1:calculateExponent(base, exponent-1))
console.log(calculateExponent(7,12) === Math.pow(7,12)) // true
console.log(calculateExponent(2,7) === Math.pow(2,7)) // true
```

现在在ES7 /ES2016，以数学向导的开发者可以使用更短的语法:

```
let a = 7 ** 12
let b = 2 ** 7
console.log(a === Math.pow(7,12)) // true
console.log(b === Math.pow(2,7)) // true
```

开发者还可以操作结果:

```
let a = 7
a **= 12
let b = 2
b **= 7
console.log(a === Math.pow(7,12)) // true
console.log(b === Math.pow(2,7)) // true
```

许多ES新特性是从其他语言（CoffeeScript-俺最爱，Ruby等）模仿而来的。你可以猜到，指数运算符在其他语言的存在形式：

- Python: $x ** y$
- CoffeeScript: $x ** y$
- F#: $x ** y$
- Ruby: $x ** y$
- Perl: $x ** y$
- Lua, Basic, MATLAB: $x ^ y$

对我个人而言没有指数运算不是什么问题。除了这次，在我15年的Javascript访谈和代码教程写作中没有写过指数运算。指数运算符是你们最需要的特性么？

Object.values/Object.entries

`Object.values` 和 `Object.entries` 是在ES2017规格中，它和 `Object.keys` 类似，返回数组类型，其序号和 `Object.keys` 序号对应。

`Object.values` , `Object.entries` 和 `Object.keys` 各自项返回是数组，相对应包括key,value或者可枚举特定对象property/attribute

在ES8 /ES2017之前，Javascript开发者需要迭代一个对象的自身属性时候不得不用 `Object.keys` , 通过迭代且使用 `obj[key]` 获取value值返回一个数组：

```
let obj = {a: 1, b: 2, c: 3}
Object.keys(obj).forEach((key, index)=>{
  console.log(key, obj[key])
})
```

而使用ES6/ES2015 中 `for/of` 稍微好点：

```
let obj = {a: 1, b: 2, c: 3}
for (let key of Object.keys(obj)) {
  console.log(key, obj[key])
}
```

你使用老方式 `for/in` (ES5)也许用的非常好。但是他会迭代所有可以枚举属性（像原型中的带名字的-see MDN (https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FStatements%2Ffor...of%23Difference_between_for...of_and_for...in)），不仅仅自己的属性，会意外的破坏那些 像 `prototype` 和 `toString` 得到意想不到的值。

`Object.values` 返回对象自身可以迭代属性值（values）为数组类型。我们最好使用 `Array.prototype.forEach` 迭代它，结合ES6的箭头函数隐形返回值：

```
let obj = {a: 1, b: 2, c: 3}
Object.values(obj).forEach(value=>console.log(value)) // 1, 2, 3
```

或者使用 `for/of`：

```
let obj = {a: 1, b: 2, c: 3}
for (let value of Object.values(obj)) {
  console.log(value)
}
// 1, 2, 3
```

·`Object.entries`·，在另一方面，将会返回对象自身可迭代属性key-value对数组（作为一个数组），他们（key-value）分别以数组存放数组中。

```
let obj = {a: 1, b: 2, c: 3}
JSON.stringify(Object.entries(obj))
"[[\"a\",1],[\"b\",2],[\"c\",3]]"
```

我们可以使用ES6/ES2015解构（需要深入了解解构请点击这篇文章 (<https://link.jianshu.com?t=https%3A%2F%2Fwebapplog.com%2Fes6>)和课程 (<https://link.jianshu.com?t=https%3A%2F%2Fnode.university%2Fp%2Fes6>)），从这嵌套数组中分别声明key和value

```
let obj = {a: 1, b: 2, c: 3}
Object.entries(obj).forEach(([key, value]) => {
  console.log(`${key} is ${value}`)
})
// a is 1, b is 2, c is 3
```

你可以猜一猜，我们同样使用ES6 `for/of`（毕竟全部都是数组）遍历 `Object.entries` 返回来的结果值。

```
let obj = {a: 1, b: 2, c: 3}
for (let [key, value] of Object.entries(obj)) {
  console.log(`${key} is ${value}`)
}
// a is 1, b is 2, c is 3
```

现在从对象中提取values和key-value pairs 变得非常容易了。 `Object.values` 和 `Object.entries` 这种方式不想之前 `Object.keys` (自身属性key+顺序相同)结合 `for/of` (ES6) 一起，我们不仅仅可以提取他们还可以迭代他们。

字符填充函数padStart 和 padEnd

`String.prototype.padStart` 和 `String.prototype.padEnd` 在javascript字符操作是一个不错的体验，帮助避免依赖而外的库。

`padStart()` 在开始部位填充，返回一个给出长度的字符串，填充物给定字符串，把字符串填充到期望的长度。从字符串的左边开始（至少大部分西方语言），一个经典例子是使用空格创建列：

```
console.log('react'.padStart(10).length)      // "      react" is 10
console.log('backbone'.padStart(10).length)    // "    backbone" is 10
```

它对于财务方面非常有用：

```
console.log('0.00'.padStart(20))
console.log('10,000.00'.padStart(20))
console.log('250,000.00'.padStart(20))
```

这结果作为一个会计总账格式非常漂亮：

```
0.00
10,000.00
250,000.00
```

第二个参数，让我们放一些其他的填充字符替代空字符串，一个字符串填充：

```
console.log('react'.padStart(10, '_'))      // "____react"
console.log('backbone'.padStart(10, '**'))   // "***backbone"
```

padEnd 顾名思义就是从字符串的尾端右边开始填充。第二个参数，你能实际上用一个任何长度的字符串。例如：

```
console.log('react'.padEnd(10, ':-'))       // "react:-:-" is 10
console.log('backbone'.padEnd(10, '**'))    // "backbone**" is 10
```

Object.getOwnPropertyDescriptor

这新的 `Object.getOwnPropertyDescriptor` 返回对象obj所有自身属性描述。这是一个多参数版本的`Object.getOwnPropertyDescriptor(obj, propName)` (https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_Objects%2FObject%2FgetOwnPropertyDescriptor)将会返回obj中propName属性的一个单独描述。

在我们日常不可变编程（immutable programming）时代中，有了这个方法很方便（记住，Javascript中对象是引用传递）在ES5中，开发者要使用 `Object.assign()` 来拷贝对象，`Object.assign()` 分配属性只有copy和定义新的属性。当我们使用更加复杂对象和类原型，这可能会出问题。

`Object.getOwnPropertyDescriptor` 允许创建真实的对象浅副本并创建子类,它通过给开发者描述符来做到这一点.在 `Object.create(prototype, object)` 放入描述符后，返回一个真正的浅拷贝

```
Object.create(
  Object.getPrototypeOf(obj),
  Object.getOwnPropertyDescriptor(obj)
)
```

或者你可以合并两个对象 `target` 和 `source` 如下：


```
Object.defineProperties(  
  target,  
  Object.getOwnPropertyDescriptors(source)  
)
```

以上是 `Object.getOwnPropertyDescriptors` 用法。但是什么是描述符(descriptor)呢？就是一个对象的描述，废话！

好吧！好吧！，让我们挖掘一下描述符一点点多信息。这里有两种描述符号类型：

1.数据描述符 (Data descriptor)

2.存取器描述符 (Accessor descriptor)

存取描述符有必须属性：`get` 或者`set`或者`get`和`set`两个就是如你所想的getter和setter函数，然后存取描述符还有可选属性 `configurable` 和 `enumerable`

```
let azatsBooks = {  
  books: ['React Quickly'],  
  get latest () {  
    let numberOfBooks = this.books.length  
    if (numberOfBooks == 0) return undefined  
    return this.books[numberOfBooks - 1]  
  }  
}
```

这个例子数据描述符 `books` 由 `Object.getOwnPropertyDescriptor(azatsBooks, 'books')` 产生结果如下：

```
Object  
  configurable: true  
  enumerable: true  
  value: Array[1]  
  writable: true  
  __proto__: Object
```

同样的，`Object.getOwnPropertyDescriptor(azatsBooks, 'latest')` 将会展现latest的描述符，这个latest (`get`) 存取器描述符展现如下：

```
Object  
  configurable: true  
  enumerable: true  
  get: latest()  
  set: undefined  
  __proto__: Object
```

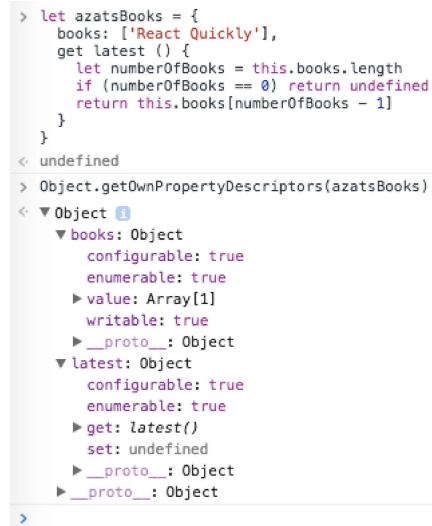
现在我们调用新方法获取所有的描述符：

```
console.log(Object.getOwnPropertyDescriptors(azatsBooks))
```

它会给出这个对象两个描述符books和latest：

```
Object
  books: Object
    configurable: true
    enumerable: true
    value: Array[1]
    writable: true
    __proto__: Object
  latest: Object
    configurable: true
    enumerable: true
    get: latest()
    set: undefined
    __proto__: Object
  __proto__: Object
```

或者你可以使用Devtools格式化一下，下面截图：



```
> let azatsBooks = {
  books: ['React Quickly'],
  get latest () {
    let numberOfBooks = this.books.length
    if (numberOfBooks == 0) return undefined
    return this.books[numberOfBooks - 1]
  }
}
< undefined
> Object.getOwnPropertyDescriptors(azatsBooks)
< ▼ Object
  ▼ books: Object
    configurable: true
    enumerable: true
    value: Array[1]
    writable: true
    __proto__: Object
  ▼ latest: Object
    configurable: true
    enumerable: true
    get: latest()
    set: undefined
    __proto__: Object
  __proto__: Object
>
```

Paste_Image.png

函数参数列表和调用中的尾逗号

尾逗号在函数定义中只是一个纯粹语法变化，在ES5中，将会非法语法，在函数参数后面应该没有逗号的：

```
var f = function(a,
  b,
  c,
  d) { // NO COMMA!
  // ...
  console.log(d)
}
f(1,2,3,'this')
```

在ES8中，这种尾逗号是没有问题的：

```
var f = function(a,
  b,
  c,
  d,
) { // COMMA? OK!
  // ...
  console.log(d)
}
f(1,2,3,'this')
```

现在，函数中尾逗号是向数组（ES3）中和字面量对象（ES5）中尾逗号看齐。

```
var arr = [1, // Length == 3
  2,
  3,
] // <--- ok
let obj = {a: 1, // Only 3 properties
  b: 2,
  c: 3,
} // <--- ok
```

更不用说他是无用友好的。

尾逗号主要有用在使用多行参数风格（典型的是那些很长的参数名），开发者终于可以忘记逗号放在第一位这种奇怪的写法。自从逗号bugs主要原因就是使用他们。而现在你可以到处使用逗号，甚至最后参数都可以。

异步函数

异步函数（或者async/await）特性操作是Promise (https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_Objects%2FPromise) 最重要的功能。所以你大概进一步阅读他们或者看一个进修视频课程来。这种想法是为了在写异步代码中简化它，因为人类大脑最讨厌这种平行非序号思维了。它只是不会演变这种方式。

对于我个人来说，我不喜欢Promise，就仅仅相比callback显得特别冗余。所以我从来没有使用过它，幸运的是，在ES8，异步函数是那么给力。开发者定义一个 `async` 函数里面不包含或者包含await 基于Promise异步操作。在这引擎之下一个异步函数返回一个Promise，无论何你在任何地方不会看到这样的一个词（注：Promise）（当然了，你非的自己使用）。

例如，在ES6中我们可以使用Promise，Axios (<https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fmzabriskie%2Faxios>)库向GraphQL服务器发送一个请求：

```
axios.get(`/q?query=${query}`)
  .then(response => response.data)
  .then(data => {
    this.props.processfetchedData(data) // Defined somewhere else
  })
  .catch(error => console.log(error))
```

任何一个Promise库都能兼容新的异步函数，我们可以使用同步try/catch做错误处理。

```
async fetchData(url) => {
  try {
    const response = await axios.get(`/q?query=${query}`)
    const data = response.data
    this.props.processfetchedData(data)
  } catch (error) {
    console.log(error)
  }
}
```

异步函数返回一个Promise，所以我们像下面可以继续执行流程：

```

async fetchData(query) => {
  try {
    const response = await axios.get(`/q?query=${query}`)
    const data = response.data
    return data
  } catch (error) {
    console.log(error)
  }
}
fetchData(query).then(data => {
  this.props.processFetchedData(data)
})

```

你可以看到这段代码在(Babel REPL (<https://link.jianshu.com?t=http%3A%2F%2Fbit.ly%2F2kjLPFg>))生效。请注意，这个例子中，Axios库被代替的，是通过模拟来做相同功能，而HTTP请求通过setTimeout代替：

```

let axios = { // mocks
  get: function(x) {
    return new Promise(resolve => {
      setTimeout(() => {
        resolve({data: x})
      }, 2000)
    })
  }
}
let query = 'mangos'
async function fetchData(query) {
  try {
    const response = await axios.get(`/q?query=${query}`)
    const data = response.data
    return data
  } catch (error) {
    console.log(error)
  }
}
fetchData(query).then(data => {
  console.log(data) // Got data 2s Later... Can use data!
})

```

有了 async/await,我们的代码执行异步看起来像执行同步一样。可以从头到尾读起来非常简单和易懂，因为出现结果顺序和函数题中从头到尾顺序一样啊！

小结

这就是或多或少ES8特性（还没有定稿），和已经定稿的ES7（已经定稿），如果你使用Babel，Traceur或者类似编译器，您可以使用这些所有功能以及更多0~3 stage功能，而不需要等浏览器实现他们。ES7和ES8将简单转换ES5兼容代码，即使是IE9都可以运

行：)

一些ES8功能需要注意了，因为他们还是处于stage3阶段，但是有可能出现ES8/ES2017中。

- 共享内存和原子 (Shared Memory and Atomics)
- SIMD.JS - SIMD APIs
- Function.prototype.toString
- 提升模板文字限制 (Lifting Template Literal Restriction)
- global
- Rest/Spread Properties
- 异步迭代 (Asynchronous Iteration)
- import()

你可以时刻关注他们的状态 进行中提案 (<https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Ftc39%2Fproposals%2Fblob%2Fmaster%2FREADME.md%23active-proposals>)和已经完成的提案 (<https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Ftc39%2Fproposals%2Fblob%2Fmaster%2Ffinished-proposals.md>)

PS:需要复习 Primose、箭头函数、let/const和其他ES6/ES2015功能，请阅读我的博客和视频。

PS2: 如果你更喜欢观看视频的话，请登录Node.University'sES6/ES2015 course

(<https://link.jianshu.com?t=https%3A%2F%2Fnode.university%2Fp%2Fes6>),还有即将来临的ES7和ES8视频课程 (<https://link.jianshu.com?t=https%3A%2F%2Fnode.university%2Fp%2Fes7-es8>)

原文地址: [https://node.university/blog/498412/es7-es8?](https://node.university/blog/498412/es7-es8?utm_source=javascriptweekly&utm_medium=email)

[utm_source=javascriptweekly&utm_medium=email](https://link.jianshu.com?t=https%3A%2F%2Fnode.university%2Fblog%2F498412%2Fes7-es8%3Futm_source%3Djavascriptweekly%26utm_medium%3Demail) (https://link.jianshu.com?t=https%3A%2F%2Fnode.university%2Fblog%2F498412%2Fes7-es8%3Futm_source%3Djavascriptweekly%26utm_medium%3Demail)

[t=https%3A%2F%2Fnode.university%2Fblog%2F498412%2Fes7-es8%3Futm_source%3Djavascriptweekly%26utm_medium%3Demail](https://link.jianshu.com?t=https%3A%2F%2Fnode.university%2Fblog%2F498412%2Fes7-es8%3Futm_source%3Djavascriptweekly%26utm_medium%3Demail))

【爱在西元前翻译】