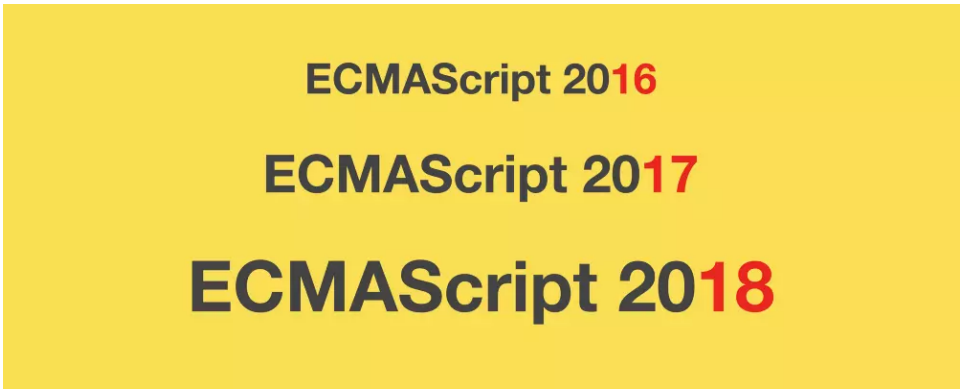
# 细解JavaScript ES7 ES8 ES9新特性

zhleven (/u/60e9b22078a5) + 关注
1.3 2018.11.28 13:58 字数6001 阅读10548 评论3 喜欢19
(/u/60e9b22078a5)

导言：ECMAScript中的演化不会停止，但是我们完全没必要害怕除了ES6这个史无前例的版本带来了海量的信息和知识点以外，之后每年一发的版本都仅仅带有少量的增量更新，一年更新的东西花半个小时就能搞懂了，完全没必要畏惧。本文将带您花大约一个小时左右的时间，迅速过一遍ES7，ES8，ES9的新特性。

想追求更好的阅读体验，请移步原文地址 (https://tuobaye.com/2018/11/27/%E7%BB%86%E8%A7%A3JavaScript-ES7-ES8-ES9-%E6%96%B0%E7%89%B9%E6%80%A7/)




es_16_17_18

题记：本文提供了一个在线PPT版本，方便您浏览细解JAVASCRIPT ES7 ES8 ES9新特性在线PPT版本 (https://tuobaye.com/demo/es7_es8_es9/)

本文的大部分内容译作作者Axel Rauschmayer博士的网站，想了解更多关于作者的信息，可以浏览探索JS：程序员的JavaScript书籍 (http://exploringjs.com/)

## 那些与ECMAScript中有关的事情

### 谁在设计的ECMAScript?

TC39（技术委员会39）。

TC39是推进JavaScript发展的委员会。其会员都是公司（其中主要是浏览器厂商）.TC39定期召开会议，会议由会员公司的代表与特邀专家出席。会议纪录都可在网上查看，可以让你对TC39如何工作有一个清晰的概念。

很有意思的是，TC39实行的是协商一致的原则：通过一项决议必须得到每一位会员（公司代表）的赞成。

### ECMAScript中的发布周期

在2015年发布的ECMAScript（ES6）新增内容很多，在ES5发布近6年（2009-11至2015-6）之后才将其标准化。两个发布版本之间时间跨度如此之大主要有两大原因：

- 比新版率先完成的特性，必须等待新版的完成才能发布。

- 那些需要花长时间完成的特性，也顶着很大的压力被纳入这一版本，因为如果推迟到下一版本发布意味着又要等很久，这种特性也会推迟新的发布版本。

因此，从ECMAScript 2016（ES7）开始，版本发布变得更加繁繁，每年发布一个新版本，这么一来新增内容也会更小。新版本将会包含每年截止时间之前完成的所有特性。



## ECMAScript中的发布流程

每个ECMAScript特性的建议将会从阶段0开始，然后经过下列几个成熟阶段。其中从一个阶段到下一个阶段必须经过TC39的批准。

1. stage-0 - Strawman：只是一个想法，可能是Babel插件。
   任何讨论，想法，改变或者还没加到提案的特性都在这个阶段。只有TC39成员可以提交。

当前的第0阶段列表可以查看这里 - > 第0阶段提案 (https://github.com/tc39/proposals/blob/master/stage-0-proposals.md)

2. 阶段1 - 提案：这值得继续。

   **什么是提案?** 一份新特性的正式建议文档。提案必须指明此建议的潜在问题，例如与其他特性之间的关联，实现难点等。

3. 阶段2 - 草案：初始规范。

   **什么是草案?** 草案是规范的第一个版本。其与最终标准中包含的特性不会有太大差别。

草案之后，原则上只接受增量修改。这个阶段开始实验如何实现，实现形式包括polyfill，实现引擎（提供草案执行本地支持），或者编译转换（例如babel）

4. 阶段-3 - 候选人：完整的规范和初始浏览器实施。

候选阶段，获得具体实现和用户的反馈。此后，只有在实现和使用过程中出现了重大问题才会修改。至少要在一个浏览器中实现，提供填充工具或者巴贝尔插件。

5. 阶段4 - 完成：将添加到下一年度版本。

已经准备就绪，该特性会出现在下个版本的ECMAScript的规范之中。

当前的阶段1-3列表可以查看这里 - > ECMAScript提案 (https://github.com/tc39/proposals)

## 已经正式发布的特性索引

| 提案 | 作者 | 冠军（S） | TC39会议记录 |
| --- | --- | --- | --- |
| `Array.prototype.includes` (https://github.com/tc39/Array.prototype.includes) | Domenic Denicola | Domenic Denicola <br /> Rick Waldron | 2015年11月 (https://github.com/rwaldron/tc notes/blob/master/es7/2015-1 17.md#arrayprototypeincludes |
| 指数运算符 (https://github.com/rwaldron/exponentiation-operator) | 里克沃尔德伦 | 里克沃尔德伦 | 2016年1月 (https://github.com/rwaldron/tc notes/blob/master/es7/2016-0 28.md#5xviii-exponentiation-o rw) |

| 提案 | 作者 | 冠军（S） | TC39会议记录 |
| --- | --- | --- | --- |
| `Object.values` / `Object.entries` (https://github.com/tc39/proposal-object-values-entries) | Jordan Harband | Jordan Harband | 2016年3月 (https://github.com/rwaldron/tc notes/blob/master/es7/2016-0 29.md#objectvalues--objecten |
| 字符串填充 (https://github.com/tc39/proposal-string-pad-start-end) | Jordan Harband | Jordan Harband <br /> Rick Waldron | 2016年5月 (https://github.com/rwaldron/tc notes/blob/master/es7/2016-0 25.md#stringprototypepadstar |
| `Object.getOwnPropertyDescriptors` (https://github.com/ljharb/proposal-object-getownpropertydescriptors) | Jordan Harband <br /> Andrea Giammarchi | Jordan Harband <br /> Andrea Giammarchi | 2016年5月 (https://github.com/rwaldron/tc notes/blob/master/es7/2016-0 25.md#objectgetownpropertyd jhd) |
| 函数参数列表和调用中的尾随逗号 (https://github.com/tc39/proposal-trailing-function-commas) | 杰夫莫里森 | 杰夫莫里森 | 2016年7月 (https://github.com/rwaldron/tc notes/blob/master/es7/2016-0 26.md#9ie-trailing-commas-in |
| 异步功能 (https://github.com/tc39/ecmascript-asyncawait) | 布莱恩特尔森 | 布莱恩特尔森 | 2016年7月 (https://github.com/rwaldron/tc notes/blob/master/es7/2016-0 28.md#10iv-async-functions) |
| 共享记忆和原子 (https://github.com/tc39/ecmascript_sharedmem) | Lars T Hansen | Lars T Hansen | 2017年1月 (https://github.com/rwaldron/tc notes/blob/master/es7/2017-0 24.md#13if-seeking-stage-4-fo sharedarraybuffer) |
| 提升模板文字限制 (https://github.com/tc39/proposal-template-literal-revision) | 添迪士尼 | 添迪士尼 | 2017年3月 (https://github.com/rwaldron/tc notes/blob/master/es8/2017-0 21.md#10ia-template-literal-up |
| `s dotAll` 正则表达式的 （） 标志 (https://github.com/tc39/proposal-regexp-dotall-flag) | Mathias Bynens | Brian Terlson <br /> Mathias Bynens | 2017年11月 (https://github.com/rwaldron/te notes/blob/master/es8/2017-1 28.md#9ie-regexp-dotall-statu |
| RegExp命名为捕获组 (https://github.com/tc39/proposal-regexp-named-groups) | Gorkem Yakin <br /> Daniel Ehrenberg | Daniel Ehrenberg <br /> Brian Terlson <br /> Mathias Bynens | 2017年11月 (https://github.com/rwaldron/tc notes/blob/master/es8/2017-1 28.md#9if-regexp-named-capt status-update) |
| 休息/传播属性 (https://github.com/tc39/proposal-object-rest-spread) | 塞巴斯蒂安马克博格 | 塞巴斯蒂安马克博格 | 2018年1月 (https://github.com/rwaldron/tc notes/blob/master/es8/2018-0 23.md#restspread-properties- 4) |
| RegExp Lookbehind断言 (https://github.com/tc39/proposal-regexp-lookbehind) | Gorkem Yakin <br />NozomuKatō<br /> Daniel Ehrenberg | Daniel Ehrenberg <br /> Mathias Bynens | 2018年1月 (https://github.com/rwaldron/tc notes/blob/master/es8/2018-0 23.md#conclusionresolution-1 |
| RegExp Unicode属性转义 (https://github.com/tc39/proposal-regexp-unicode-property-escapes) | Mathias Bynens | Brian Terlson <br /> Daniel Ehrenberg <br /> Mathias Bynens | 2018年1月 (https://github.com/rwaldron/tc notes/blob/master/es8/2018-0 24.md#conclusionresolution-1 |

| 提案 | 作者 | 冠军（S） | TC39会议记录 |
| --- | --- | --- | --- |
| `Promise.prototype.finally` (https://github.com/tc39/proposal-promise-finally) | Jordan Harband | Jordan Harband | 2018年1月 (https://github.com/rwaldron/tc notes/blob/master/es8/2018-0 24.md#conclusionresolution-2 |
| 异步迭代 (https://github.com/tc39/proposal-async-iteration) | Domenic Denicola | Domenic Denicola | 2018年1月 (https://github.com/rwaldron/tc notes/blob/master/es8/2018-0 25.md#conclusionresolution) |
| 可选 `catch` 绑定 (https://github.com/tc39/proposal-optional-catch-binding) | 迈克尔菲卡拉 | 迈克尔菲卡拉 | 2018年5月 |
| JSON超集 (https://github.com/tc39/proposal-json-superset) | 理查德吉布森 | Mark Miller <br /> Mathias Bynens | 2018年5月 |

## ES7新特性（ECMAScript 2016）


ECMAScript 2016

ES7在ES6的基础上主要添加了两项内容：

- Array.prototype.includes（）方法
- 求幂运算符（**）

### Array.prototype.includes（）方法

includes（）方法用来判断一个数组是否包含一个指定的值，根据情况，如果包含则返回true，否则返回false。

```
var array = [1, 2, 3];

console.log(array.includes(2));
// expected output: true

var pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));
// expected output: true

console.log(pets.includes('at'));
// expected output: false
```

Array.prototype.includes（）方法接收两个参数：

- 要搜索的值
- 搜索的开始索引。

当第二个参数被传入时，该方法会从索引处开始往后搜索（默认索引值为0）。若搜索值在数组中存在则返回true，否则返回false。且看下面示例：

```
['a', 'b', 'c', 'd'].includes('b')         // true
['a', 'b', 'c', 'd'].includes('b', 1)      // true
['a', 'b', 'c', 'd'].includes('b', 2)      // false
```

乍一看，包括的作用跟数组的的indexOf重叠，为什么要特意增加这么一个API呢主要区别有以下几点?

- 如果条件判断的时候包含要简单得多，而indexOf需要多写一个条件进行判断。

```
var ary = [1];
if (ary.indexOf(1) !== -1) {
    console.log("数组存在1")
}
if (ary.includes(1)) {
    console.log("数组存在1")
}
```

- 为NaN的判断。如果数组中有NaN时，你又正好需要判断数组是否有存在NaN时，这时你使用的indexOf是无法判断的，你必须使用包括这个方法。

```
var ary1 = [NaN];
console.log(ary1.indexOf(NaN))//-1
console.log(ary1.includes(NaN))//true
```

- 当数组的有空的值的时候，包括会认为空的值是未定义的，而不会的indexOf。

```
var ary1 = new Array(3);
console.log(ary1.indexOf(undefined));//-1
console.log(ary1.includes(undefined))//true
```

## 求幂运算符（**）

加/减法我们通常都是用其中缀形式，直观易懂。在ECMAScript2016中，可以我们使用 ** 来替代Math.pow。

```
4 ** 3          // 64
```

效果等同于

```
Math.pow(4,3)
```

值得一提的是，作为中缀运算符，**还支持以下操作

```
let n = 4;
n **= 3;
// 64
```

## ES8新特性（ECMAScript 2017）


ECMAScript 2017

在2017年1月的TC39会议上，ECMAScript 2017的最后一个功能“共享内存和原子”推进到第4阶段。这意味着它的功能集现已完成。

ECMAScript 2017特性一览

主要新功能：

- 异步函数（Brian Terlson）
- 共享内存和原子学（Lars T. Hansen）

次要新功能：

- Object.values / Object.entries（Jordan Harband）
- 字符串填充（Jordan Harband，Rick Waldron）
- Object.getOwnPropertyDescriptors（）（Jordan Harband，Andrea Giammarchi）
- 函数参数列表和调用中的尾逗号（Jeff Morrison）

## 异步功能

Async Functions也就是我们常说的Async / Await，相信大家对于这个概念都已经不陌生了.Async / Await是一种用于处理JS异步操作的语法糖，可以帮助我们摆脱回调地狱，编写更加优雅的代码。

通俗的理解，异步关键字的作用是告诉编译器对于标定的函数要区别对待。当编译器遇到标定的函数中的的await关键字时，要暂时停止运行，带到的await标定的函数处理完毕后，再进行相应操作。如果该函数履行了，则返回值是履行价值，否则得到的就是拒绝价值。

下面通过拿普通的承诺写法来对比，就很好理解了：

```
async function asyncFunc() {
    const result = await otherAsyncFunc();
    console.log(result);
}

// Equivalent to:
function asyncFunc() {
    return otherAsyncFunc()
    .then(result => {
        console.log(result);
    });
}
```

按顺序处理多个异步函数的时候优势更为明显：

```
async function asyncFunc() {
    const result1 = await otherAsyncFunc1();
    console.log(result1);
    const result2 = await otherAsyncFunc2();
    console.log(result2);
}

// Equivalent to:
function asyncFunc() {
    return otherAsyncFunc1()
    .then(result1 => {
        console.log(result1);
        return otherAsyncFunc2();
    })
    .then(result2 => {
        console.log(result2);
    });
}
```

并行处理多个异步函数：

```
async function asyncFunc() {
    const [result1, result2] = await Promise.all([
        otherAsyncFunc1(),
        otherAsyncFunc2(),
    ]);
    console.log(result1, result2);
}

// Equivalent to:
function asyncFunc() {
    return Promise.all([
        otherAsyncFunc1(),
        otherAsyncFunc2(),
    ])
    .then([result1, result2] => {
        console.log(result1, result2);
    });
}
```

处理错误：

```
async function asyncFunc() {
    try {
        await otherAsyncFunc();
    } catch (err) {
        console.error(err);
    }
}

// Equivalent to:
function asyncFunc() {
    return otherAsyncFunc()
    .catch(err => {
        console.error(err);
    });
}
```

Async Functions若是要展开去讲，可以占用很大段的篇幅。鉴于本文是一篇介绍性文章，再次不再进行深入。

## SharedArrayBuffer和原子能

内存管理中
的速成课程 (https://hacks.mozilla.org/2017/06/a-crash-course-in-memory-management/)
ArrayBuffers和SharedArrayBuffers的漫画介绍 (https://hacks.mozilla.org/2017/06/a-cartoon-intro-to-arraybuffers-and-sharedarraybuffers/)避免使用Atomics的 (https://hacks.mozilla.org/2017/06/avoiding-race-conditions-in-sharedarraybuffers-with-atomics/)SharedArrayBuffers中的 (https://hacks.mozilla.org/2017/06/a-cartoon-intro-to-arraybuffers-and-sharedarraybuffers/)竞争条件。 (https://hacks.mozilla.org/2017/06/avoiding-race-conditions-in-sharedarraybuffers-with-atomics/)**注**，如果之前您没有接触过ArrayBuffer相关知识的话，建议您从内存管理速成教程系列漫画解说入门，强推：

(https://hacks.mozilla.org/2017/06/avoiding-race-conditions-in-sharedarraybuffers-with-atomics/)

---

ECMAScript 2017特性SharedArrayBuffer和atomics“，由Lars T. Hansen设计。它引入了一个新的构造函数SharedArrayBuffer和具有辅助函数的命名空间对象Atomics。

在我们开始之前，让我们澄清两个相似但截然不同的术语：并行（Parallelism）和并发（Concurrency）。他们存在许多定义，我使用的定义如下

- 并行（Parallelism）（parallel并行与串行串行）：同时执行多个任务;

- 并发（Concurrency）（并发并发与顺序连续）：在重叠的时间段内（而不是一个接一个）执行几个任务。

## JS并行的历史

- JavaScript在单线程中执行。某些任务可以异步执行：浏览器通常会在单线程中运行这些任务，然后通过回调将结果重新加入到单线程中。
- 网络工作者将任务并行引入了JavaScript：这些是相对重量级的进程。每个工人都有自己的全局环境。默认情况下，不共享任何内容。工人之间的通信（或工人和主线程之间）的通信）发展：
  - 起初，你只能发送和接收字符串。
  - 然后，引入结构化克隆：可以发送和接收数据副本。结构化克隆适用于大多数数据（JSON数据，TypedArray，正则表达式，Blob对象，ImageData对象等）。它甚至可以正确处理对象之间的循环引用。但是，不能克隆错误对象，功能对象和DOM节点。
  - 可在workers之间的转移数据：当接收方获得数据时，发送方失去访问权限。
- 通过WebGL使用GPU计算（它倾向于数据并行处理）

## 共享数组缓冲区（共享阵列缓冲区）

共享阵列缓冲区是更高并发抽象的基本构建块。它们允许您在多个workers和主线程之间共享SharedArrayBuffer对象的字节（该缓冲区是共享的，用于访问字节，将其封装在一个TypedArray中）这种共享有两个好处：

你可以更快地在工作者之间共享数据。
工人之间的协调变得更简单和更快（与postMessage（）相比）。

```
// main.js
const worker = new Worker('worker.js');

// 要分享的buffer
const sharedBuffer = new SharedArrayBuffer( // (A)
    10 * Int32Array.BYTES_PER_ELEMENT); // 10 elements

// 使用Worker共用sharedBuffer
worker.postMessage({sharedBuffer}); // clone

// 仅限本地使用
const sharedArray = new Int32Array(sharedBuffer); // (B)
```

创建一个共享数组缓冲区（Shared Array Buffers）的方法与创建普通的数组缓冲区（Array Buffer）类似：通过调用构造函数，并以字节的形式指定缓冲区的大小（行A）。你与工人共享的是缓冲区（缓冲区）。对于你自己的本地使用，你通常将共享数组缓冲区封装在TypedArray中（行B）。

工人的实现如下所列。

```
// worker.js
self.addEventListener('message', function (event) {
    const {sharedBuffer} = event.data;
    const sharedArray = new Int32Array(sharedBuffer); // (A)
    // ···
});
```

## sharedArrayBuffer的API

构造函数：

- new SharedArrayBuffer（length）
  创建一个length字节的缓冲区（缓冲区）。

静态属性：

- 获取SharedArrayBuffer [Symbol.species]
  默认情况下返回此。覆盖以控制slice（）的返回。

实例属性：

- get SharedArrayBuffer.prototype.byteLength（）
  返回缓冲区（缓冲区）的字节长度。

- SharedArrayBuffer.prototype.slice（start，end）
  创建一个新的this.constructor [Symbol.species]实例，并用字节填充从（包括）开始到（不包括）结束的索引。

## Atomics：安全访问共享数据

举一个例子

```
// main.js
sharedArray[1] = 11;
sharedArray[2] = 22;
```

在单线程中，您可以重新排列这些写入操作，因为在中间没有读到任何内容。对于多线程，当你期望以特定顺序执行写入操作时，就会遇到麻烦：

```
// worker.js
while (sharedArray[2] !== 22) ;
console.log(sharedArray[1]); // 0 or 11
```

Atomics方法可以用来与其他工人进行同步。例如，以下两个操作可以让你读取和写入数据，并且不会被编译器重新排列：

- Atomics.load（ta：TypedArray，index）
- Atomics.store（ta：TypedArray，index，value：T）
-

这个想法是使用常规操作读取和写入大多数数据，而Atomics操作（load，store和其他操作）可确保读取和写入安全。通常，您将使用自定义同步机制，例如锁，其实现基于原子公司。

这是一个非常简单的例子，它总是有效的：

```
// main.js
console.log('notifying...');
Atomics.store(sharedArray, 0, 123);

// worker.js
while (Atomics.load(sharedArray, 0) !== 123) ;
console.log('notified');
```

## Atomics的API

Atomic函数的主要操作数必须是Int8Array，Uint8Array，Int16Array，Uint16Array，Int32Array或Uint32Array的一个实例。它必须包裹一个SharedArrayBuffer。

所有函数都以atomically方式进行操作。存储操作的顺序是固定的并且不能由编译器或CPU重新排序。

加载和存储

- Atomics.load（ta：TypedArray <T>，index）：T
  读取和返回ta [index]上的元素，返回数组指定位置上的值。
- Atomics.store（ta：TypedArray <T>，index，value：T）：T
  在ta [index]上写入值，并且返回值。
- Atomics.exchange（ta：TypedArray <T>，index，value：T）：T
  将ta [index]上的元素设置为value，并且返回索引索引原先的值。
- Atomics.compareExchange（ta：TypedArray <T>，index，expectedValue，replacementValue）：T
  如果ta [index]上的当前元素为expectedValue，那么使用replacementValue替换。并且返回索引索引原先（或者未改变）的值。

简单修改TypeArray元素

以下每个函数都会在给定索引处更改TypeArray元素：它将一个操作符应用于元素和参数，并将结果写回元素。它返回元素的原始值。

- Atomics.add（ta：TypedArray <T>，index，value）：T
  执行ta [index] + = value并返回ta [index]的原始值。
- Atomics.sub（ta：TypedArray <T>，index，value）：T
  执行ta [index] - = value并返回ta [index]的原始值。
- Atomics.and（ta：TypedArray <T>，index，value）：T
  执行ta [index]＆= value并返回ta [index]的原始值。
- Atomics.or（ta：TypedArray <T>，index，value）：T
  执行ta [index] | = value并返回ta [index]的原始值。
- Atomics.xor（ta：TypedArray <T>，index，value）：T
  执行ta [index] ^ = value并返回ta [index]的原始值。

等待和唤醒

- Atomics.wait（ta：Int32Array，index，value，timeout = Number.POSITIVE_INFINITY）:('not-equal'|'ok'|'timed-out'）
  如果ta [index]的当前值不是值，则返回'不等于'。否则继续等待，直到我们通过Atomics.wake（）唤醒或直到等待超时。在前一种情况下，返回'ok'。在后一种情况下，返回'timed-out'.timeout以毫秒为单位。记住此函数执行的操作：“如果ta [index]为值，那么继续等待”。
- Atomics.wake（ta：Int32Array，index，count）
  唤醒等待在ta [index]上的计数工人。

## Object.values和Object.entries

Object.values（）方法返回一个给定对象自己的所有可枚举属性值的数组，值的顺序与使用for ... in循环的顺序相同（区别在于for-in循环枚举原型链中的属性）。

obj参数是需要待操作的对象。可以是一个对象，或者一个数组（是一个带有数字下标的对象，[10,20,30] - > {0：10,1：20,2：30}）。

```
const obj = { x: 'xxx', y: 1 };
Object.values(obj); // ['xxx', 1]

const obj = ['e', 's', '8']; // 相当于 { 0: 'e', 1: 's', 2: '8' };
Object.values(obj); // ['e', 's', '8']

// 当我们使用数字键值时，返回的是数字排序
// 根据键值排序
const obj = { 10: 'xxx', 1: 'yyy', 3: 'zzz' };
Object.values(obj); // ['yyy', 'zzz', 'xxx']

Object.values('es8'); // ['e', 's', '8']
```

Object.entries方法返回一个给定对象自身可遍历属性[key，value]的数组，排序规则和Object.values一样。这个方法的声明比较琐碎：

```
const obj = { x: 'xxx', y: 1 };
Object.entries(obj); // [['x', 'xxx'], ['y', 1]]

const obj = ['e', 's', '8'];
Object.entries(obj); // [['0', 'e'], ['1', 's'], ['2', '8']]

const obj = { 10: 'xxx', 1: 'yyy', 3: 'zzz' };
Object.entries(obj); // [['1', 'yyy'], ['3', 'zzz'], ['10': 'xxx']]

Object.entries('es8'); // [['0', 'e'], ['1', 's'], ['2', '8']]
```

## 字符串填充

为String对象增加了2个函数：padStart和padEnd。

像它们名字那样，这几个函数的主要目的就是填补字符串的首部和尾部，为了使得到的结果字符串的长度能达到给定的长度。你可以通过特定的字符，或者字符串，或者默认的空格填充它下面是函数的声明：

```
str.padStart(targetLength [, padString])
str.padEnd(targetLength [, padString])
```

这些函数的第一个参数是targetLength（目标长度），这个是结果字符串的长度。第二个参数是可选的padString（填充字符），一个用于填充到源字符串的字符串。默认值是空格。

```
'es8'.padStart(2);          // 'es8'
'es8'.padStart(5);          // '  es8'
'es8'.padStart(6, 'woof');  // 'wooes8'
'es8'.padStart(14, 'wow');  // 'wowwowwowwoes8'
'es8'.padStart(7, '0');     // '0000es8'

'es8'.padEnd(2);            // 'es8'
'es8'.padEnd(5);            // 'es8  '
'es8'.padEnd(6, 'woof');    // 'es8woo'
'es8'.padEnd(14, 'wow');    // 'es8wowwowwowwo'
'es8'.padEnd(7, '6');       // 'es86666'
```

## Object.getOwnPropertyDescriptors

getOwnPropertyDescriptors方法返回指定对象所有自身属性的描述对象。属性描述对象是直接在对象上定义的，而不是继承于对象的原型.ES2017加入这个函数的主要动机在于方便将一个对象深度拷贝贝给另一个对象，同时可以将的getter / setter拷贝声明如下：

```
Object.getOwnPropertyDescriptors(obj)
```

obj是待操作对象。返回的描述对象键值有：configurable，enumerable，writable，get，set和value。

```
const obj = {
  get es7() { return 777; },
  get es8() { return 888; }
};
Object.getOwnPropertyDescriptor(obj);
// {
//   es7: {
//     configurable: true,
//     enumerable: true,
//     get: function es7(){}, //the getter function
//     set: undefined
//   },
//   es8: {
//     configurable: true,
//     enumerable: true,
//     get: function es8(){}, //the getter function
//     set: undefined
//   }
// }
```

## 结尾逗号

结尾逗号用代码展示非常明了：

```
// 参数定义时
function foo(
    param1,
    param2,
) {}

// 函数调用时
foo(
    'abc',
    'def',
);

// 对象中
let obj = {
    first: 'Jane',
    last: 'Doe',
};

// 数组中
let arr = [
    'red',
    'green',
    'blue',
];
```

这个改动有什么好处呢?

- 首先，重新排列项目更简单，因为如果最后一项更改其位置，则不必添加和删除逗号。
- 其次，它可以帮助版本控制系统跟踪实际发生的变化例如，从：

```
[
    'foo'
]
```

修改为

```
[
    'foo',
    'bar'
]
```

导致线条 '富' 和线条 '酒吧' 被标记为已更改，即使唯一真正的变化是后一条线被添加。

# ES9新特性（ECMAScript 2018）



ECMAScript 2018

ES9的新特性索引如下：

主要新功能：

- 异步迭代（Domenic Denicola，Kevin Smith）
- Rest / Spread属性（SebastianMarkbåge）

新的正则表达式功能：

- RegExp命名为捕获组（Gorkem Yakin，Daniel Ehrenberg）
- RegExp Unicode Property Escapes（Mathias Bynens）
- RegExp Lookbehind断言（Gorkem Yakin，NozomuKatō，Daniel Ehrenberg）
- 正则表达式的s（dotAll）标志（Mathias Bynens）

其他新功能：

- Promise.prototype.finally（）（Jordan Harband）
- 模板字符串修改（Tim Disney）

## 异步迭代

### 首先来回顾一下同步迭代器：

ES6引入了同步迭代器，其工作原理如下：

- 可迭代的：一个对象，表示可以通过Symbol.iterator方法进行迭代。
- Iterator：通过调用iterable [Symbol.iterator]（）返回的对象。它将每个迭代元素包装在一个对象中，并通过其next（）方法一次返回一个。
- IteratorResult：返回的对象next（）。属性值包含一个迭代的元素，属性done是真后最后一个元素。

示例：

```
const iterable = ['a', 'b'];
const iterator = iterable[Symbol.iterator]();
iterator.next()
// { value: 'a', done: false }
iterator.next()
// { value: 'b', done: false }
iterator.next()
// { value: undefined, done: true }
```

### 异步迭代器

先前的迭代方式是同步的，并不适用于异步数据源例如，在以下代码中，readLinesFromFile（）无法通过同步迭代传递其异步数据。：

```
for (const line of readLinesFromFile(fileName)) {
    console.log(line);
}
```

异步迭代器和常规迭代器的工作方式非常相似，但是异步迭代器涉及承诺：

```
async function example() {
  // 普通迭代器:
  const iterator = createNumberIterator();
  iterator.next(); // Object {value: 1, done: false}
  iterator.next(); // Object {value: 2, done: false}
  iterator.next(); // Object {value: 3, done: false}
  iterator.next(); // Object {value: undefined, done: true}

  // 异步迭代器:
  const asyncIterator = createAsyncNumberIterator();
  const p = asyncIterator.next(); // Promise
  await p;// Object {value: 1, done: false}
  await asyncIterator.next(); // Object {value: 2, done: false}
  await asyncIterator.next(); // Object {value: 3, done: false}
  await asyncIterator.next(); // Object {value: undefined, done: true}
}
```

异步迭代器对象的下一个（）方法返回了一个无极，解析后的值跟普通的迭代器类似。用法： `iterator.next().then(({ value, done })=> {//{value: ‘some val’, done: false}}`

```
const promises = [
    new Promise(resolve => resolve(1)),
    new Promise(resolve => resolve(2)),
    new Promise(resolve => resolve(3)),
];

async function test() {
    for await (const p of promises) {
        console.log(p);
    }
}
test(); //1 ,2 3
```

## Rest / Spread属性

这个就是我们通常所说的休息参数和扩展运算符，这项特性在ES6中已经引入，但是ES6中的作用对象仅限于数组：

```
restParam(1, 2, 3, 4, 5);

function restParam(p1, p2, ...p3) {
  // p1 = 1
  // p2 = 2
  // p3 = [3, 4, 5]
}

const values = [99, 100, -1, 48, 16];
console.log( Math.max(...values) ); // 100
```

在ES9中，为对象提供了像数组一样的休息参数和扩展运算符：

```
const obj = {
  a: 1,
  b: 2,
  c: 3
}
const { a, ...param } = obj;
  console.log(a)     //1
  console.log(param) //{b: 2, c: 3}

function foo({a, ...param}) {
  console.log(a);    //1
  console.log(param) //{b: 2, c: 3}
}
```

## 正则表达式命名捕获组

### 编号的捕获组

```
//正则表达式命名捕获组
const RE_DATE = /([0-9]{4})-([0-9]{2})-([0-9]{2})/;

const matchObj = RE_DATE.exec('1999-12-31');
const year = matchObj[1]; // 1999
const month = matchObj[2]; // 12
const day = matchObj[3]; // 31
```

通过数字引用捕获组有几个缺点：

- 找到捕获组的数量是一件麻烦事：必须使用括号。
- 如果要了解组的用途，则需要查看正则表达式。
- 如果更改捕获组的顺序，则还必须更改匹配代码。

### 命名的捕获组

ES9中可以通过名称来识别捕获组：`(?<year>[0-9]{4})`

在这里，我们用名称标记了前一个捕获组一年。该名称必须是合法的JavaScript的标识符（认为变量名称或属性名称）。匹配后，您可以通过访问捕获的字符串matchObj.groups.year来访问。

让我们重写前面的代码：

```
const RE_DATE = /(?<year>[0-9]{4})-(?<month>[0-9]{2})-(?<day>[0-9]{2})/;

const matchObj = RE_DATE.exec('1999-12-31');
const year = matchObj.groups.year; // 1999
const month = matchObj.groups.month; // 12
const day = matchObj.groups.day; // 31

// 使用解构语法更为简便
const {groups: {day, year}} = RE_DATE.exec('1999-12-31');
console.log(year); // 1999
console.log(day); // 31
```

可以发现，命名捕获组有以下优点：

- 找到捕获组的“ID”更容易。
- 匹配代码变为自描述性的，因为捕获组的ID描述了正在捕获的内容。
- 如果更改捕获组的顺序，则无需更改匹配代码。
- 捕获组的名称也使正则表达式更容易理解，因为您可以直接看到每个组的用途。

## 正则表达式Unicode转义

该特性允许您使用 `\p{}` 通过提及大括号内的Unicode字符属性来匹配字符，在正则表达式中使用标记 `u` （unicode）设置。

```
/^\p{White_Space}+$/u.test('\t \n\r')
// true
/^\p{Script=Greek}+$/u.test('μετά')
// true
```

## 新方法匹配中文字符

由于在Unicode里面，中文字符对应的Unicode脚本是**汉**，于是我们就可以用这个reg来匹配中文：

```
/\p{Script=Han}/u
```

我们这样就可以不用记忆繁琐汉语中类似的不好记的 `/[\u4e00-\u9fa5]/` 了，况且这个表达式已经有些年头了，说实话，后来又新增的属性为汉族的字符并不在这个范围内，因此这个有年头REG并不一定好使。

我随便从网上找了一个Unicode8.0添加的中文字符“钅仑”，我测了一下两种REG的兼容性：

```
oldReg=/[\u4e00-\u9fa5]/
newReg=/\p{Script=Han}/u

oldReg.test('abc')
// false
newReg.test('abc')
// false

oldReg.test('地平线')
// true
newReg.test('地平线')
// true

oldReg.test('钅仑')
// false
newReg.test('钅仑')
// true
```

http://www.unicode.org/charts/PDF/U4E00.pdf
(http://www.unicode.org/charts/PDF/U4E00.pdf)

可以参考一下这个PDF，是统一的汉字全集，从524页9FA6至526页（最后一页）用旧匹配方式都无法生效。

统一

## 一些对于统一的科普

- 名称：唯一名称，由大写字母，数字，连字符和空格组成例如：
  - 答：姓名=拉丁文大写字母A.
  - 😀：名称= GRINNING FACE
- General_Category：对字符进行分类例如：
  - X：General_Category = Lowercase_Letter
  - $：General_Category = Currency_Symbol
- White_Space：用于标记不可见的间距字符，例如空格，制表符和换行符例如：
  - \ T：White_Space = True
  - π：White_Space = False
- 年龄：引入字符的统一标准版本例如：欧元符号€在统一标准的2.1版中添加。
  - €：年龄= 2.1
- 脚本：是一个或多个书写系统使用的字符集合。
  - 有些脚本支持多种写入系统。例如，拉丁文脚本支持英语，法语，德语，拉丁语等书写系统。
  - 某些语言可以用多个脚本支持的多个备用写入系统编写。例如，土耳其语在20世纪初转换为拉丁文字之前使用了阿拉伯文字。
  - 例子：
    - α：剧本=希腊语
    - Д：脚本=西里尔语

## 正则表达式的统一属性转义

- 匹配其属性的道具具有值的所有字符值：

```
\p{prop=value}
```

- 匹配所有没有属性的道具值的字符值：

```
\P{prop=value}
```

- 匹配二进制属性bin_prop为真正的所有字符：

```
\p{bin_prop}
```

- 匹配二进制属性bin_prop为假的所有字符：

```
\P{bin_prop}
```

- 匹配空格：

```
/^\p{White_Space}+$/u.test('\t \n\r')
//true
```

匹配字母：

```
/^\p{Letter}+$/u.test('πüé')
//true
```

匹配希腊字母：

```
/^\p{Script=Greek}+$/u.test('μετά')
//true
```

匹配拉丁字母：

```
/^\p{Script=Latin}+$/u.test('Grüße')
//true
```

## 正则表达式反向断言

先来看下正则表达式先行断言是什么：

如获取货币的符号

```
const noReLookahead = /\D(\d+)/,
      reLookahead = /\D(?=\d+)/,
      match1 = noReLookahead.exec('$123.45'),
      match2 = reLookahead.exec('$123.45');
console.log(match1[0]); // $123
console.log(match2[0]); // $
```

在ES9中可以允许反向断言：

```
const reLookahead = /(?<=\D)[\d\.]+/;
      match = reLookahead.exec('$123.45');
console.log(match[0]); // 123.45
```

使用?<=进行反向断言，可以使用反向断言获取货币的价格，而忽略货币符号。

## 正则表达式DOTALL模式

正则表达式中点匹配除回车外的任何单字符，标记小号改变这种行为，允许行终止符的出现，例如：

```
/hello.world/.test('hello\nworld');  // false
/hello.world/s.test('hello\nworld'); // true
```

## Promise.prototype.finally ()

。这个基本没什么好讲的，看名字就能看懂了其用法如下：

```
promise
  .then(result => {···})
  .catch(error => {···})
  .finally(() => {···});
```

最终的回调总会被执行。

## 模板字符串修改

ES2018移除对ECMAScript在带标签的模版字符串中转义序列的语法限制。
之前，\ u开始一个unicode转义，\ x开始一个十六进制转义，\后跟一个数字开始一个八进制转义。这使得创建特定的字符串变得不可能，例如Windows文件路径C：\ uuu \ xxx \ 111。

要取消转义序列的语法限制，可在模板字符串之前使用标记函数String.raw：

```
`\u{54}`
// "T"
String.raw`\u{54}`
// "\u{54}"
```

## 尾声

ECMAScript中的演化不会停止，但是我们完全没必要害怕。除了ES6这个史无前例的版本带来了海量的信息和知识点以外，之后每年一发的版本都仅仅带有少量的增量更新，一年更新的东西花半个小时就能搞懂了，完全没必要畏惧。

保持饥饿。保持愚蠢。





zhleven (/u/60e9b22078a5)

写了33632字，被33人关注，获得了100个喜欢

(/u/60e9b22078a5)

＋关注

前端

登录 (/sign后发表评论source=desktop&utm_medium=not-signed-in-comment-form

**3条评论** 只看作者 按时间倒序 按时间正序

伯纳乌的追风少年 (/u/b28d7175a04d)
2楼 · 2019.02.01 15:42
(/u/b28d7175a04d)
求不要更新了，老子学不动了

2人赞 回复

精神NASA人 (/u/6a1136fc52ae)： 是啊，我他妈现在语言学多了，脑子都快凌乱了
2019.06.30 19:35 回复

沙雕先生 (/u/9e9c9a16ef6c)： 求求你们别学了,我跟不上....
2019.07.09 12:13 回复

添加新评论



被以下专题收入，发现更多相似内容

拓跋的前端客栈 (/c/e4401921c63f?utm_source=desktop&utm_medium=notes-included-collection)

让前端飞 (/c/c261fa3879d6?utm_source=desktop&utm_medium=notes-included-collection)

前端开发那些事 (/c/0020d95b7928?utm_source=desktop&utm_medium=notes-included-collection)

ES6 (/c/dbbc438c574b?utm_source=desktop&utm_medium=notes-included-collection)

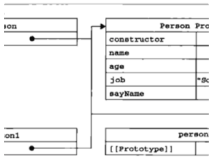
前端开发笔记 (/c/1499d63b1185?utm_source=desktop&utm_medium=notes-included-collection)

ES6 (/c/1d497c5825e2?utm_source=desktop&utm_medium=notes-included-collection)

JS笔记 (/c/14f2d75312cd?utm_source=desktop&utm_medium=notes-included-collection)

展开更多

(/p/8e9a0299abeb?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
**JavaScript的高级程序设计 - 笔记 (/p/8e9a0299abeb?utm_campaign=ma…**

第3章基本概念3.1语法3.2关键字和保留字3.3变量3.4数据类型5种简单数据类型：Undefined，Null，Boolean，Number，String 1种复杂数据类型：Object（本质上是由一组无序的名值对组成的）3.4.1 typeo…

RickCole (/u/79a382ee2560?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

---

### ES6笔记 (/p/1897185f8f23?utm_campaign=maleskine&utm_content=n…

ES6 http://es6.ruanyifeng.com目录1让和const命令2变量的解构赋值3字符串的扩展4正则的扩展5数值的扩展6函数的扩展7数组的扩展8对象的扩展9符号10集和地图数据结构1 ...

常青1890年 (/u/78f40c91f815?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

---

### JavaScript的程序 (/p/22c4737e8272?utm_campaign=maleskine&utm_c…

第2章基本语法2.1概述基本句法和变量语句JavaScript程序的执行单位为行（line），也就是一行一行地执行。一般情况下，每一行就是一个语句。语句（statement）是为了完成某种任务而进行的操作，比如下面…

泰迪学长_ (/u/e0d5c1d81e8b?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

---

### 第5章引用类型 (/p/7b8e95e5f116?utm_campaign=maleskine&utm_cont…

第5章引用类型（返回首页）本章内容使用对象创建并操作数组理解基本的JavaScript类型使用基本类型和基本包装类型用类型的值（对象）是引用类型的 - 个实例。在ECMAScript中，引用类型是 - 种数据结构，1…

千机楼 (/u/6558a1eafb6a?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

---

### 【七月未央】20170802学习力6期践行D79 (/p/2b377de1ff69?utm_campai…

【七月未央】20170802学习力6期践行D79践行：1。“手指谣”。复习了小鸽子，小汽车，学说话和最爱，新增了一首“两只小鸡”。在读小鸽子时，自己可以接出“咕咕咕”，还不断向我示意阳台，提醒我鸽子是在阳台…

七妈_haiyan (/u/ebce85625071?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

---

### java的正则表达式 (/p/7c40f3e2d80a?utm_campaign=maleskine&utm_c…

Java的正则表达式语法

cengel (/u/1c87c086674e?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

---

### 控制器数据显示，但不能拖动或者点击上面的控件 (/p/c8a5e1af6b03?utm_…

1.原因控制器已经释放，查看还存在

通哥 (/u/c9eb96b5b8cb?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

---

### 一年又一年 (/p/93e07f314b98?utm_campaign=maleskine&utm_content…

1吃了腊八饭，离过年还有二十一天半。这句老话在儿时每年的腊八节上，母亲总会念叨两遍。虽然当时有些小，但意思还是懂的，知道离年越来越近了。那时的自己不太会看日历，隔两天，就会问母亲，还有几天…

梦里看海 (/u/324ee531f285?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)