

ES6、ES7、ES8学习指南



CrazyCodeBoy (/u/ca3943a4172a) [+ 关注](#)

0.7 2018.09.15 15:25* 字数 3719 阅读 1700 评论 2 喜欢 17 阅读 1700 评论 2 喜欢 17

(/u/ca3943a4172a)

期待已久的新课上线啦！解锁React Native开发新姿势，一网打尽React Native最新与最热技术，点我Get!!! (<https://coding.imooc.com/class/304.html>)

概述

ES全称ECMAScript，ECMAScript是ECMA制定的标准化脚本语言。目前JavaScript使用的ECMAScript版本为ECMAScript-262 (<https://www.ecma-international.org/ecma-262/>)。

ECMAScript 标准建立在一些原有的技术上，最为著名的是 JavaScript (网景) 和 JScript (微软)。它最初由网景的 Brendan Eich 发明，第一次出现是在网景的 Navigator 2.0 浏览器上。Netscape 2.0 以及微软 Internet Explorer 3.0 后序的所有浏览器上都有它的身影。

ECMAScript版本	发布时间	新增特性
ECMAScript 2009(ES5)	2009年11月	扩展了Object、Array、Function的功能等
ECMAScript 2015(ES6)	2015年6月	类，模块化，箭头函数，函数参数默认值等
ECMAScript 2016(ES7)	2016年3月	includes，指数操作符
ECMAScript 2017(ES8)	2017年6月	sync/await，Object.values()，Object.entries()，String padding等

了解这些特性，不仅能使我们的编码更加的符合规范，而且能提高我们Coding的效率。

ES6的特性

ES6的特性比较多，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很大，所以ES6中的特性比较多。

在这里列举几个常用的：

- 类
- 模块化
- 箭头函数
- 函数参数默认值
- 模板字符串
- 解构赋值
- 延展操作符
- 对象属性简写
- Promise
- Let与Const

1.类（class）

对熟悉Java， object-c， c#等纯面向对象语言的开发者来说，都会对class有一种特殊的情怀。ES6 引入了class（类），让JavaScript的面向对象编程变得更加简单和易于理解。

```
class Animal {
  // 构造函数，实例化的时候将会被调用，如果不指定，那么会有一个不带参数的默认构造函数。
  constructor(name,color) {
    this.name = name;
    this.color = color;
  }
  // toString 是原型对象上的属性
  toString() {
    console.log('name:' + this.name + ',color:' + this.color);
  }
}

var animal = new Animal('dog','white');//实例化Animal
animal.toString();

console.log(animal.hasOwnProperty('name')); //true
console.log(animal.hasOwnProperty('toString')); // false
console.log(animal.__proto__.hasOwnProperty('toString')); // true

class Cat extends Animal {
  constructor(action) {
    // 子类必须要在constructor中指定super 函数，否则在新建实例的时候会报错。
    // 如果没有置顶constructor,默认带super函数的constructor将会被添加、
    super('cat','white');
    this.action = action;
  }
  toString() {
    console.log(super.toString());
  }
}

var cat = new Cat('catch')
cat.toString();

// 实例cat 是 Cat 和 Animal 的实例，和Es5完全一致。
console.log(cat instanceof Cat); // true
console.log(cat instanceof Animal); // true
```

2.模块化(Module)

ES5不支持原生的模块化，在ES6中模块作为重要的组成部分被添加进来。模块的功能主要由 export 和 import 组成。每一个模块都有自己单独的作用域，模块之间的相互调用关系是通过 export 来规定模块对外暴露的接口，通过import来引用其它模块提供的接口。同时还为模块创造了命名空间，防止函数的命名冲突。

导出(export)

ES6允许在一个模块中使用export来导出多个变量或函数。

导出变量

```
//test.js
export var name = 'Rainbow'
```

心得：ES6不仅支持变量的导出，也支持常量的导出。 export const sqrt = Math.sqrt;//导出常量

ES6将一个文件视为一个模块，上面的模块通过 export 向外输出了一个变量。一个模块也可以同时往外面输出多个变量。

```
//test.js
var name = 'Rainbow';
var age = '24';
export {name, age};
```

导出函数

```
// myModule.js
export function myModule(someArg) {
  return someArg;
}
```

导入(import)

定义好模块的输出以后就可以在另外一个模块通过import引用。

```
import {myModule} from 'myModule';// main.js
import {name,age} from 'test';// test.js
```

心得:一条import 语句可以同时导入默认函数和其它变量。 import defaultMethod, { otherMethod } from 'xxx.js';

3.箭头（Arrow）函数

这是ES6中最令人激动的特性之一。=> 不只是关键字function的简写，它还带来了其它好处。箭头函数与包围它的代码共享同一个 this ,能帮你很好的解决this的指向问题。有经验的JavaScript开发者都熟悉诸如 var self = this; 或 var that = this 这种引用外围this的模式。但借助 => ，就不需要这种模式了。

箭头函数的结构

箭头函数的箭头=>之前是一个空括号、单个的参数名、或用括号括起的多个参数名，而箭头之后可以是一个表达式（作为函数的返回值），或者是用花括号括起的函数体（需要自行通过return来返回值，否则返回的是undefined）。

```
// 箭头函数的例子
()=>1
v=>v+1
(a,b)=>a+b
()=>{
    alert("foo");
}
e=>{
    if (e == 0){
        return 0;
    }
    return 1000/e;
}
```

心得：不论是箭头函数还是bind，每次被执行都返回的是一个新的函数引用，因此如果你还需要函数的引用去做一些别的事情（譬如卸载监听器），那么你必须自己保存这个引用。

卸载监听器时的陷阱

错误的做法

```
class PauseMenu extends React.Component{
  componentWillMount(){
    AppStateIOS.addListener('change', this.onAppPaused.bind(this))
  };
  componentWillUnmount(){
    AppStateIOS.removeListener('change', this.onAppPaused.bind(this));
  }
  onAppPaused(event){
  }
}
```

正确的做法

```

class PauseMenu extends React.Component{
  constructor(props){
    super(props);
    this._onAppPaused = this.onAppPaused.bind(this);
  }
  componentWillMount(){
    AppStateIOS.addListener('change', this._onAppPaused);
  }
  componentWillUnmount(){
    AppStateIOS.removeListener('change', this._onAppPaused);
  }
  onAppPaused(event){
  }
}

```

除上述的做法外，我们还可以这样做：

```

class PauseMenu extends React.Component{
  componentWillMount(){
    AppStateIOS.addListener('change', this.onAppPaused);
  }
  componentWillUnmount(){
    AppStateIOS.removeListener('change', this.onAppPaused);
  }
  onAppPaused = (event) => {
    //把函数直接作为一个arrow function的属性来定义，初始化的时候就绑定好了this指针
  }
}

```

需要注意的是：不论是bind还是箭头函数，每次被执行都返回的是一个新的函数引用，因此如果你还需要函数的引用去做一些别的事情（譬如卸载监听器），那么你必须自己保存这个引用。

4.函数参数默认值 (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/default_parameters)

ES6支持在定义函数的时候为其设置默认值：

```

function foo(height = 50, color = 'red')
{
  // ...
}

```

不使用默认值：

```

function foo(height, color)
{
  var height = height || 50;
  var color = color || 'red';
  //...
}

```

这样写一般没问题，但当 参数的布尔值为false 时，就会有问题了。比如，我们这样调用foo函数：

```
foo(0, '')
```

因为 0的布尔值为false ， 这样height的取值将是50。同理color的取值为‘red’。

所以说， 函数参数默认值 不仅能是代码变得更加简洁而且能规避一些问题。

5.模板字符串

ES6支持 模板字符串 ， 使得字符串的拼接更加的简洁、直观。

不使用模板字符串：

```
var name = 'Your name is ' + first + ' ' + last + '.'
```

使用模板字符串：

```
var name = `Your name is ${first} ${last}.\`
```

在ES6中通过 \${} 就可以完成字符串的拼接， 只需要将变量放在大括号之中。

6.解构赋值 (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

解构赋值语法是JavaScript的一种表达式， 可以方便的从数组或者对象中快速提取值赋给定义的变量。

获取数组中的值

从数组中获取值并赋值到变量中， 变量的顺序与数组中对象顺序对应。

```
var foo = ["one", "two", "three", "four"];

var [one, two, three] = foo;
console.log(one); // "one"
console.log(two); // "two"
console.log(three); // "three"

//如果你要忽略某些值，你可以按照下面的写法获取你想要的值
var [first, , , last] = foo;
console.log(first); // "one"
console.log(last); // "four"

//你也可以这样写
var a, b; //先声明变量

[a, b] = [1, 2];
console.log(a); // 1
console.log(b); // 2
```

如果没有从数组中的获取到值，你可以为变量设置一个默认值。

```
var a, b;

[a=5, b=7] = [1];
console.log(a); // 1
console.log(b); // 7
```

通过解构赋值可以方便的交换两个变量的值。

```
var a = 1;
var b = 3;

[a, b] = [b, a];
console.log(a); // 3
console.log(b); // 1
```

获取对象中的值

```
const student = {
  name: 'Ming',
  age: '18',
  city: 'Shanghai'
};

const {name, age, city} = student;
console.log(name); // "Ming"
console.log(age); // "18"
console.log(city); // "Shanghai"
```

7. 延展操作符(Spread operator)

延展操作符... 可以在函数调用/数组构造时，将数组表达式或者string在语法层面展开；还可以在构造对象时，将对象表达式按key-value的方式展开。

语法

函数调用：

```
myFunction(...iterableObj);
```

数组构造或字符串：

```
[...iterableObj, '4', ...'hello', 6];
```

构造对象时,进行克隆或者属性拷贝（ECMAScript 2018规范新增特性）：

```
let objClone = { ...obj };
```

应用场景

在函数调用时使用延展操作符

```
function sum(x, y, z) {  
  return x + y + z;  
}  
const numbers = [1, 2, 3];  
  
//不使用延展操作符  
console.log(sum.apply(null, numbers));  
  
//使用延展操作符  
console.log(sum(...numbers));// 6
```

构造数组

没有展开语法的时候，只能组合使用 push，splice，concat 等方法，来将已有数组元素变成新数组的一部分。有了展开语法, 构造新数组会变得更简单、更优雅：

```
const stuendts = ['Jine','Tom'];  
const persons = ['Tony',... stuendts,'Aaron','Anna'];  
console.log(persons)// ["Tony", "Jine", "Tom", "Aaron", "Anna"]
```

和参数列表的展开类似，... 在构造数组时, 可以在任意位置多次使用。

数组拷贝


```
var arr = [1, 2, 3];
var arr2 = [...arr]; // 等同于 arr.slice()
arr2.push(4);
console.log(arr2)//[1, 2, 3, 4]
```

展开语法和 Object.assign() 行为一致, 执行的都是浅拷贝(只遍历一层)。

连接多个数组

```
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];
var arr3 = [...arr1, ...arr2];// 将 arr2 中所有元素附加到 arr1 后面并返回
//等同于
var arr4 = arr1.concat(arr2);
```

在ECMAScript 2018中延展操作符增加了对对象的支持

```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var clonedObj = { ...obj1 };
// 克隆后的对象: { foo: "bar", x: 42 }

var mergedObj = { ...obj1, ...obj2 };
// 合并后的对象: { foo: "baz", x: 42, y: 13 }
```

在React中的应用

通常我们在封装一个组件时, 会对外公开一些 props 用于实现功能。大部分情况下在外部使用都应显示的传递 props 。但是当传递大量的props时, 会非常繁琐, 这时我们可以使用 ... (延展操作符, 用于取出参数对象的所有可遍历属性) 来进行传递。

一般情况下我们应该这样写

```
<CustomComponent name ='Jine' age ={21} />
```

使用 ... , 等同于上面的写法

```
const params = {
  name: 'Jine',
  age: 21
}

<CustomComponent {...params} />
```

配合解构赋值避免传入一些不需要的参数

```
var params = {
  name: '123',
  title: '456',
  type: 'aaa'
}

var { type, ...other } = params;

<CustomComponent type='normal' number={2} {...other} />
//等同于
<CustomComponent type='normal' number={2} name='123' title='456' />
```

8.对象属性简写

在ES6中允许我们在设置一个对象的属性时候不指定属性名。

不使用ES6

```
const name='Ming',age='18',city='Shanghai';

const student = {
  name:name,
  age:age,
  city:city
};
console.log(student); //{name: "Ming", age: "18", city: "Shanghai"}
```

对象中必须包含属性和值，显得非常冗余。

使用ES6

```
const name='Ming',age='18',city='Shanghai';

const student = {
  name,
  age,
  city
};
console.log(student); //{name: "Ming", age: "18", city: "Shanghai"}
```

对象中直接写变量，非常简洁。

9.Promise (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide/Using_promises)

Promise 是异步编程的一种解决方案，比传统的解决方案callback更加的优雅。它最早由社区提出和实现的，ES6 将其写进了语言标准，统一了用法，原生提供了Promise对象。

不使用ES6

嵌套两个setTimeout回调函数：

```
setTimeout(function()  
{  
  console.log('Hello'); // 1秒后输出"Hello"  
  setTimeout(function()  
  {  
    console.log('Hi'); // 2秒后输出"Hi"  
  }, 1000);  
, 1000);
```

使用ES6

```
var waitSecond = new Promise(function(resolve, reject)  
{  
  setTimeout(resolve, 1000);  
});  
  
waitSecond  
  .then(function()  
  {  
    console.log("Hello"); // 1秒后输出"Hello"  
    return waitSecond;  
  })  
  .then(function()  
  {  
    console.log("Hi"); // 2秒后输出"Hi"  
  });
```

上面的的代码使用两个then来进行异步编程串行化，避免了回调地狱：

10.支持let与const

在之前JS是没有块级作用域的，const与let填补了这方便的空白，const与let都是块级作用域。

使用var定义的变量为函数级作用域：

```
{  
  var a = 10;  
}  
  
console.log(a); // 输出10
```

使用let与const定义的变量为块级作用域：

```
{  
  let a = 10;  
}  
  
console.log(a); //-1 or Error“ReferenceError: a is not defined”
```

ES7的特性

在ES6之后，ES的发布频率更加频繁，基本每年一次，所以自ES6之后，每个新版本的特性的数量就比较少。

- includes()
- 指数操作符

1. Array.prototype.includes()

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

includes 函数与 indexOf 函数很相似，下面两个表达式是等价的：

```
arr.includes(x)
arr.indexOf(x) >= 0
```

接下来我们来判断数字中是否包含某个元素：

在ES7之前的做法

使用 indexOf() 验证数组中是否存在某个元素，这时需要根据返回值是否为-1来判断：

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

if (arr.indexOf('react') !== -1)
{
    console.log('react存在');
}
```

使用ES7的includes()

使用includes()验证数组中是否存在某个元素，这样更加直观简单：

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

if (arr.includes('react'))
{
    console.log('react存在');
}
```

2.指数操作符

在ES7中引入了指数运算符 **，** 具有与 Math.pow(..) 等效的计算结果。

不使用指数操作符

使用自定义的递归函数`calculateExponent`或者`Math.pow()`进行指数运算：

```
function calculateExponent(base, exponent)
{
  if (exponent === 1)
  {
    return base;
  }
  else
  {
    return base * calculateExponent(base, exponent - 1);
  }
}

console.log(calculateExponent(2, 10)); // 输出1024
console.log(Math.pow(2, 10)); // 输出1024
```

使用指数操作符

使用指数运算符`**`，就像`+`、`-`等操作符一样：

```
console.log(2**10); // 输出1024
```

ES8的特性

- `async/await`
- `Object.values()`
- `Object.entries()`
- String padding
- 函数参数列表结尾允许逗号
- `Object.getOwnPropertyDescriptors()`

浏览器兼容性 (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/values#Browser_compatibility)

1.async/await

在ES8中加入了对 `async/await` 的支持，也就我们所说的 异步函数，这是一个很实用的功能。 `async/await` 将我们从头痛的回调地狱中解脱出来了，使整个代码看起来很简洁。

使用 `async/await` 与不使用 `async/await` 的差别：

```

login(userName) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('1001');
    }, 600);
  });
}

getData(userId) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (userId === '1001') {
        resolve('Success');
      } else {
        reject('Fail');
      }
    }, 600);
  });
}

// 不使用async/await ES7
doLogin(userName) {
  this.login(userName)
    .then(this.getData)
    .then(result => {
      console.log(result)
    })
}

// 使用async/await ES8
async doLogin2(userName) {
  const userId=await this.login(userName);
  const result=await this.getData(userId);
}

this.doLogin()// Success
this.doLogin2()// Success

```

async/await的几种应用场景

接下来我们来看一下 async/await 的几种应用场景。

获取异步函数的返回值

异步函数本身会返回一个 Promise，所以我们可以通过 then 来获取异步函数的返回值。

```

async function charCountAdd(data1, data2) {
  const d1=await charCount(data1);
  const d2=await charCount(data2);
  return d1+d2;
}
charCountAdd('Hello','Hi').then(console.log);//通过then获取异步函数的返回值。
function charCount(data) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(data.length);
    }, 1000);
  });
}

```

对于上述的例子，我们调用 `await` 两次，每次都是等待1秒一共是2秒，效率比较低，而且两次 `await` 的调用并没有依赖关系，那能不能让其并发执行呢，答案是可以的，接下来我们通过 `Promise.all` 来实现 `await` 的并发调用。

```
async function charCountAdd(data1, data2) {
  const [d1,d2]=await Promise.all([charCount(data1),charCount(data2)]);
  return d1+d2;
}
charCountAdd('Hello','Hi').then(console.log);
function charCount(data) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(data.length);
    }, 1000);
  });
}
```

通过上述代码我们实现了两次 `charCount` 的并发调用，`Promise.all` 接受的是一个数组，它可以将数组中的promise对象并发执行；

async/await的几种错误处理方式

第一种：捕捉整个async/await函数的错误

```
async function charCountAdd(data1, data2) {
  const d1=await charCount(data1);
  const d2=await charCount(data2);
  return d1+d2;
}
charCountAdd('Hello','Hi')
  .then(console.log)
  .catch(console.log);//捕捉整个async/await函数的错误
...
```

这种方式可以捕捉整个 `charCountAdd` 运行过程中出现的错误，错误可能是由 `charCountAdd` 本身产生的，也可能是由对 `data1` 的计算中或 `data2` 的计算中产生的。

第二种：捕捉单个的await表达式的错误

```
async function charCountAdd(data1, data2) {
  const d1=await charCount(data1)
    .catch(e=>console.log('d1 is null'));
  const d2=await charCount(data2)
    .catch(e=>console.log('d2 is null'));
  return d1+d2;
}
charCountAdd('Hello','Hi').then(console.log);
```

通过这种方式可以捕捉每一个 `await` 表达式的错误，如果既要捕捉每一个 `await` 表达式的错误，又要捕捉整个 `charCountAdd` 函数的错误，可以在调用 `charCountAdd` 的时候加个 `catch`。

```
...
charCountAdd('Hello','Hi')
  .then(console.log)
  .catch(console.log);//捕捉整个async/await函数的错误
...
```

第三种：同时捕捉多个的`await`表达式的错误

```
async function charCountAdd(data1, data2) {
  let d1,d2;
  try {
    d1=await charCount(data1);
    d2=await charCount(data2);
  }catch (e){
    console.log('d1 is null');
  }
  return d1+d2;
}
charCountAdd('Hello','Hi')
  .then(console.log);

function charCount(data) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(data.length);
    }, 1000);
  });
}
```

2.Object.values()

`Object.values()` 是一个与 `Object.keys()` 类似的新函数，但返回的是 `Object` 自身属性的所有值，不包括继承的值。

假设我们要遍历如下对象 `obj` 的所有值：

```
const obj = {a: 1, b: 2, c: 3};
```

不使用 `Object.values()` :ES7

```
const vals=Object.keys(obj).map(key=>obj[key]);
console.log(vals);//[1, 2, 3]
```

使用 `Object.values()` :ES8

```
const values=Object.values(obj1);
console.log(values);//[1, 2, 3]
```


从上述代码中可以看出 `Object.values()` 为我们省去了遍历key，并根据这些key获取value的步骤。

3.Object.entries

`Object.entries()` 函数返回一个给定对象自身可枚举属性的键值对的数组。

接下来我们来遍历上文中的 `obj` 对象的所有属性的key和value：

不使用`Object.entries()` :ES7

```
Object.keys(obj).forEach(key=>{
  console.log('key:'+key+' value:'+obj[key]);
})
//key:a value:1
//key:b value:2
//key:c value:3
```

使用`Object.entries()` :ES8

```
for(let [key,value] of Object.entries(obj1)){
  console.log(`key: ${key} value:${value}`)
}
//key:a value:1
//key:b value:2
//key:c value:3
```

4.String padding

在ES8中String新增了两个实例函数 `String.prototype.padStart` 和 `String.prototype.padEnd`，允许将空字符串或其他字符串添加到原始字符串的开头或结尾。

`String.padStart(targetLength,[padString])`

- `targetLength`:当前字符串需要填充到的目标长度。如果这个数值小于当前字符串的长度，则返回当前字符串本身。
- `padString`:(可选)填充字符串。如果字符串太长，使填充后的字符串长度超过了目标长度，则只保留最左侧的部分，其他部分会被截断，此参数的缺省值为 ""。

```
console.log('0.0'.padStart(4,'10')) //10.0
console.log('0.0'.padStart(20))//          0.00
```

`String.padEnd(targetLength,padString])`

- `targetLength`:当前字符串需要填充到的目标长度。如果这个数值小于当前字符串的长度，则返回当前字符串本身。
- `padString`:(可选) 填充字符串。如果字符串太长，使填充后的字符串长度超过了目标长度，则只保留最左侧的部分，其他部分会被截断，此参数的缺省值为 ""；

```
console.log('0.0'.padEnd(4,'0')) //0.00
console.log('0.0'.padEnd(10,'0'))//0.000000000
```

4.函数参数列表结尾允许逗号

这是一个不痛不痒的更新，主要作用是方便使用git进行多人协作开发时修改同一个函数减少不必要的行变更。

不使用ES8
<pre>//程序员A var f = function(a, b) { ... } //程序员B var f = function(a, b, //变更行 c //变更行) { ... } //程序员C var f = function(a, b, c, //变更行 d //变更行) { ... }</pre>
使用ES8

```
//程序员A
var f = function(a,
    b,
    ) {
    ...
}

//程序员B
var f = function(a,
    b,
    c,    //变更行
    ) {
    ...
}

//程序员C
var f = function(a,
    b,
    c,
    d,    //变更行
    ) {
    ...
}
```

5.Object.getOwnPropertyDescriptors()

Object.getOwnPropertyDescriptors() 函数用来获取一个对象的所有自身属性的描述符,如果没有任何自身属性, 则返回空对象。

函数原型:

Object.getOwnPropertyDescriptors(obj)

返回 obj 对象的所有自身属性的描述符, 如果没有任何自身属性, 则返回空对象。

```
const obj2 = {
    name: 'Jine',
    get age() { return '18' }
};
Object.getOwnPropertyDescriptors(obj2)
// {
//   age: {
//     configurable: true,
//     enumerable: true,
//     get: function age(){}, //the getter function
//     set: undefined
//   },
//   name: {
//     configurable: true,
//     enumerable: true,
//     value:"Jine",
//     writable:true
//   }
// }
```

总结


技术更替的车轮一直在前进中，JavaScript的规范和标准也在不断的制定和完善。你会发现ECMAScript 新版的很多特性已经是Typescript，浏览器或其他polyfills的一部分，就拿ES8的 `async/await` 来说，它是2017年6月被纳入ECMAScript的，但我在2016年的时候就已经开始使用这个特性了，这些特性通常由ECMAScript议员提交，然后会出现在在未来的某个ECMAScript版本中。

参考

- MDN (<https://developer.mozilla.org/>)
- ECMAScript® 2016 (<https://www.ecma-international.org/ecma-262/7.0/>)
- ECMAScript® 2016 语言标准 (<https://github.com/w3c-html-ig-zh/es7>)
- ECMAScript® 2017 (<https://www.ecma-international.org/ecma-262/8.0/>)

小礼物走一走，来简书关注我

赞赏支持

 前端技术 (/nb/29474128)

[举报文章](#) © 著作权归作者所有



CrazyCodeBoy (/u/ca3943a4172a)

写了 134605 字，被 2018 人关注，获得了 6709 个喜欢
(/u/ca3943a4172a)写了 134605 字，被 2018 人关注，获得了 6709 个喜欢

+ 关注

专注于移动开发，分享知识，传播快乐 技术博客：<http://www.devio.org> 我的开源：<https://github.com/>...

喜欢 | 17







更多分享




后发表评论

(/signin?source=desktop&utm_medium=not-signed-in-com

2条评论

只看作者

[按时间倒序](#) [按时间正序](#)



陈Sir的流水账 (/u/e38b7eab23ae)

3楼 · 2019.02.11 19:17
(/u/e38b7eab23ae)

很好，很清晰。

赞  回复



哈哈咯_de3f (/u/580070377e86)

2楼 · 2018.10.29 10:10

(/u/580070377e86)

感谢分享!

赞 回复

被以下专题收入，发现更多相似内容



工具癖 (/c/2mvgxp?utm_source=desktop&utm_medium=notes-included-collection)



React N... (/c/35f193da15ed?utm_source=desktop&utm_medium=notes-included-collection)

推荐阅读

更多精彩内容 > (/)

ES6笔记 (/p/1897185f8f23?utm_campaign=maleskine&utm_content=...

ES6 <http://es6.ruanyifeng.com> 目录 1 let 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 Symbol 10 Set 和 Map 数据结构 1...



常青1890 (/u/78f40c91f815?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio

ES6~ES7/ES8/发生的变化 (/p/0a3f6af5a6c8?utm_campaign=maleski...

ES7新特性 ES7在ES6的基础上添加了三项内容：求幂运算符（[^]）、Array.prototype.includes()方法、函数作用域中严格模式的变更。Array.prototype.includes()方法 includes()的作用，是查找一个值在不在数



吴佳浩 (/u/c817dc83befd?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio

ES6的新特性 (/p/3a3d4e94f007?utm_campaign=maleskine&utm_cont...

一、ES6简介 历时将近6年的时间来制定的新 ECMAScript 标准 ECMAScript 6（亦称 ECMAScript Harmony，简称 ES6）终于在 2015 年 6 月正式发布。自从上一个标准版本 ES5 在 2009 年发布以后，



荣嬷嬷King、 (/u/21d6f5b55dc9?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio

你不可不知的ES6 (/p/ed72aaccfac?utm_campaign=maleskine&utm_c...

本文为阮一峰大神的《ECMAScript 6 入门》的个人版提纯！ babel babel负责将JS高级语法转义，使之能被各种浏览器所执行。其使用步骤如下： 编写配置文件.babelrc此文件存于根目录，基本格式如下： 选定



Devildi已被占用 (/u/ecc18a48cb30?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio

Java面试宝典Beta5.0 (/p/fb7d48083e5e?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

pdf下载地址：Java面试宝典 第一章内容介绍 20 第二章JavaSE基础 21 一、Java面向对象 21 1. 面向对象都有哪些特性以及你对这些特性的理解 21 2. 访问权限修饰符public、private、protected, 以及不写（默



王震阳 (/u/773a782d9d83?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation

吉列剃须刀质量培训与考核虚拟现实系统世峰数字sufencg.com (/p/c245e0000ea9?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

随着经济的高速发展，科技技术的日新月异，我国制造业开始进行大规模改革，传统的手工制作已经不能满足当下的市场需求，取而代之的是智能机器时代的来临。大型制造业企业开始引进智能机器生产线，但



sufenmd (/u/e66e4e485da9?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation

(/p/1fc8cb586f0a?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation

关于教育 (/p/1fc8cb586f0a?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

转发了一篇关于比较中西方教育的文章，却引起了朋友圈朋友的抓狂和质疑：“这个文章读了到底要表达什么观点呢？是要提倡苦读还是不提倡呢？如果是，那后面的部分不是打脸吗？如果不是，前半部分去说这



筱荀 (/u/a863fffe9c6b?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation

酒后 (/p/2c2bf2899062?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

吃饭、喝酒，以及 我曾经爱过你 默默无闻，不知孤独 与卑微地爱着你 爱情啊，你瞧它 多像一次残留在人世间的 毫无指望的神迹 这条决绝的单向街上 垂落的松针穿透了无数支离破碎 不再想起你的时候 多少泪眼



野孤蝉 (/u/ad601efd8457?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation

成人礼 (/p/38e2ed000ea9?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

来自老师的馈赠



Banana01 (/u/3732a4a3de5e?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation

(/p/a033c5457894?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation

陈益峰：万达广场生意火爆，风水窍门很多 (/p/a033c5457894?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

文/陈益峰 近日受商户邀请，去好几个万达广场看过风水，多数万达的商场生意不错，详细揣摩其中的风水门道，心得很大。其实商业地产内有很多研究风水的大师，包含王健林先生，他们不断的总结归纳人流、



利科甲 (/u/4ef3be8385f0?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation

