Es6学习手记

**作　者　：韩　　涛**

**2017年3月10日**



[第一章 Let和const命令 7](#_Toc1663258760)

[1. Let命令 7](#_Toc533224956)

[2. 块级作用域 7](#_Toc924068517)

[3. Const命令 7](#_Toc372236891)

[4. 顶层对象的属性 7](#_Toc2124121781)

[5. Global对象 7](#_Toc644030470)

[第二章 变量的解构和赋值 7](#_Toc1887083640)

[1. 数组的解构赋值 7](#_Toc700844249)

[2. 对象的解构赋值 7](#_Toc1547273321)

[3. 字符串的解构赋值 7](#_Toc475913180)

[4. 数值和布尔值的解构赋值 7](#_Toc228618852)

[5. 函数参数的解构赋值 7](#_Toc26900994)

[6. 圆括号问题 7](#_Toc845603743)

[7. 用途 7](#_Toc410672931)

[第三章 字符串的扩展 8](#_Toc1849988820)

[1. 字符的Unicode表示法 8](#_Toc1665951354)

[2. codePointAt() 8](#_Toc496762586)

[3. String.fromCodePoint() 8](#_Toc527913350)

[4. 字符串的遍历器接口 8](#_Toc281641675)

[5. at() 8](#_Toc1398521171)

[6. normalize() 8](#_Toc1137997069)

[7. includes(), startsWith(), endsWith() 8](#_Toc199042239)

[8. repeat() 8](#_Toc1589039558)

[9. padStart()，padEnd() 8](#_Toc1914937435)

[10. 模板字符串 8](#_Toc1369274691)

[11. 实例：模板编译 8](#_Toc1055433160)

[12. 标签模板 8](#_Toc537712215)

[13. String.raw() 8](#_Toc1779769876)

[14. 模板字符串的限制 8](#_Toc1729594067)

[第四章 正则的扩展 9](#_Toc787615832)

[1. RegExp构造函数 9](#_Toc638999800)

[2. 字符串的正则方法 9](#_Toc1245369179)

[3. u修饰符 9](#_Toc1320840788)

[4. y 修饰符 9](#_Toc1563068317)

[5. sticky属性 9](#_Toc1617606070)

[6. flags属性 9](#_Toc1297478921)

[7. RegExp.escape() 9](#_Toc59615139)

[8. s 修饰符：dotAll 模式 9](#_Toc1357206063)

[9. 后行断言 9](#_Toc1998323170)

[10. Unicode属性类 9](#_Toc1606888460)

[第五章 数值的扩展 9](#_Toc1833119243)

[1. 二进制和八进制表示法 10](#_Toc79458374)

[2. Number.isFinite(), Number.isNaN() 10](#_Toc1633789454)

[3. Number.parseInt(), Number.parseFloat() 10](#_Toc531239338)

[4. Number.isInteger() 10](#_Toc490131306)

[5. Number.EPSILON 10](#_Toc1336294627)

[6. 安全整数和Number.isSafeInteger() 10](#_Toc49707044)

[7. Math对象的扩展 10](#_Toc986893892)

[8. Math.signbit() 10](#_Toc1864207977)

[9. 指数运算符 10](#_Toc331348719)

[第六章 数组的扩展 10](#_Toc237931415)

[1. Array.from() 11](#_Toc854721399)

[2. Array.of() 11](#_Toc530390958)

[3. 数组实例的copyWithin() 11](#_Toc1826970974)

[4. 数组实例的find()和findIndex() 11](#_Toc622175186)

[5. 数组实例的fill() 11](#_Toc1899665650)

[6. 数组实例的entries()，keys()和values() 11](#_Toc734920486)

[7. 数组实例的includes() 11](#_Toc1159887401)

[8. 数组的空位 11](#_Toc1531951878)

[第七章 函数的扩展 11](#_Toc317030905)

[1. 函数参数的默认值 12](#_Toc1947503233)

[2. rest参数 12](#_Toc23468030)

[3. 扩展运算符 12](#_Toc1562400085)

[4. 严格模式 12](#_Toc1120860373)

[5. name 属性 12](#_Toc1586536347)

[6. 箭头函数 12](#_Toc1032522507)

[7. 绑定 this 12](#_Toc270855647)

[8. 尾调用优化 12](#_Toc1646151486)

[9. 函数参数的尾逗号 12](#_Toc242244922)

[第八章 对象的扩展 12](#_Toc121695169)

[1. 属性的简洁表示法 13](#_Toc1105556298)

[2. 属性名表达式 13](#_Toc2075364165)

[3. 方法的 name 属性 13](#_Toc201153544)

[4. Object.is() 13](#_Toc591862104)

[5. Object.assign() 13](#_Toc459119856)

[6. 属性的可枚举性 13](#_Toc691284850)

[7. 属性的遍历 13](#_Toc1928156731)

[8. \_\_proto\_\_Object.setPrototypeOf()，Object.getPrototypeOf() 13](#_Toc508826900)

[9. Object.key(),Object.values(),Object.entries() 13](#_Toc1678178742)

[10. 对象的扩展运算符 13](#_Toc1644881061)

[11. Object.getOwnPropertyDescriptors() 13](#_Toc840175620)

[12. Null 传导运算符 13](#_Toc1916110158)

[第九章 Symbol 13](#_Toc352118812)

[1. 概述 14](#_Toc1370566578)

[2. 作为属性名的Symbol 14](#_Toc1595597484)

[3. 实例：消除魔术字符串 14](#_Toc974293998)

[4. 属性名的遍历 14](#_Toc1122748580)

[5. Symbol.for()，Symbol.keyFor() 14](#_Toc183034322)

[6. 实例：模块的 Singleton 模式 14](#_Toc2134181399)

[7. 内置的Symbol值 14](#_Toc507216810)

[第十章 Set和map数据解构 14](#_Toc500065228)

[1. Set 14](#_Toc1934200985)

[2. WeakSet 14](#_Toc530684840)

[3. Map 14](#_Toc2062465313)

[4. WeakMap 14](#_Toc907577710)

[第十一章 Proxy 14](#_Toc2117221187)

[1. 概述 15](#_Toc947504172)

[2. Proxy 实例的方法 15](#_Toc1178433357)

[3. Proxy.revocable() 15](#_Toc1615889025)

[4. this 问题 15](#_Toc1189749095)

[5. 实例：Web 服务的客户端 15](#_Toc1300128527)

[第十二章 Reflect 15](#_Toc573961675)

[1. 概述 15](#_Toc1117629612)

[2. 静态方法 15](#_Toc1501282071)

[3. 实例：使用 Proxy 实现观察者模式 15](#_Toc1165823780)

[第十三章 Promise对象 15](#_Toc1576749468)

[1. Promise 的含义 16](#_Toc45083273)

[2. 基本用法 16](#_Toc946496863)

[3. Promise.prototype.then() 16](#_Toc2085576369)

[4. Promise.prototype.catch() 16](#_Toc1723262015)

[5. Promise.all() 16](#_Toc443894276)

[6. Promise.race() 16](#_Toc778268341)

[7. Promise.resolve() 16](#_Toc1491888525)

[8. Promise.reject() 16](#_Toc796013088)

[9. 两个有用的附加方法 16](#_Toc1351271)

[10. 应用 16](#_Toc940002361)

[11. Promise.try() 16](#_Toc1770307086)

[第十四章 Iterator和for...of循环 16](#_Toc1124099852)

[1. Iterator（遍历器）的概念 17](#_Toc1123036684)

[2. 数据结构的默认Iterator接口 17](#_Toc1757004838)

[3. 调用Iterator接口的场合 17](#_Toc1631316662)

[4. 字符串的Iterator接口 17](#_Toc1623101912)

[5. Iterator接口与Generator函数 17](#_Toc1543722175)

[6. 遍历器对象的return()，throw() 17](#_Toc14517855)

[7. for...of循环 17](#_Toc1538083577)

[第十五章 Generator函数的语法 17](#_Toc303816237)

[1. 简介 18](#_Toc2131739042)

[2. next方法的参数 18](#_Toc338104101)

[3. for...of循环 18](#_Toc1482249595)

[4. Generator.prototype.throw() 18](#_Toc1600144420)

[5. Generator.prototype.return() 18](#_Toc1527853196)

[6. yield\* 语句 18](#_Toc634894474)

[7. 作为对象属性的Generator函数 18](#_Toc26622447)

[8. Generator函数的this 18](#_Toc497999161)

[9. 含义 18](#_Toc2136176545)

[10. 应用 18](#_Toc1192446227)

[第十六章 Generator函数的异步应用 18](#_Toc2074748629)

[1. 传统方法 19](#_Toc33776170)

[2. 基本概念 19](#_Toc2138943091)

[3. Generator 函数 19](#_Toc2012841350)

[4. Thunk 函数 19](#_Toc1757038185)

[5. co 模块 19](#_Toc435353719)

[第十七章 Async函数 19](#_Toc643626043)

[1. 含义 19](#_Toc1101443063)

[2. 用法 19](#_Toc1231366808)

[3. 语法 19](#_Toc644977315)

[4. async 函数的实现原理 19](#_Toc2041445424)

[5. 与其他异步处理方法的比较 19](#_Toc854190246)

[6. 实例：按顺序完成异步操作 19](#_Toc1769077167)

[7. 异步遍历器 19](#_Toc1016998460)

[第十八章 Class 19](#_Toc463711436)

[1. Class基本语法 20](#_Toc1252910181)

[2. Class的继承 20](#_Toc492616724)

[3. 原生构造函数的继承 20](#_Toc2007433611)

[4. Class的取值函数（getter）和存值函数（setter） 20](#_Toc1267428036)

[5. Class 的 Generator 方法 20](#_Toc2030700301)

[6. Class 的静态方法 20](#_Toc163766201)

[7. Class的静态属性和实例属性 20](#_Toc1251683431)

[8. 类的私有属性 20](#_Toc221320755)

[9. new.target属性 20](#_Toc1646015796)

[10. Mixin模式的实现 20](#_Toc704344203)

[第十九章 Decorator 20](#_Toc1749173951)

[1. 类的修饰 21](#_Toc133426622)

[2. 方法的修饰 21](#_Toc730966650)

[3. 为什么修饰器不能用于函数？ 21](#_Toc99689464)

[4. core-decorators.js 21](#_Toc122119519)

[5. 使用修饰器实现自动发布事件 21](#_Toc1923412878)

[6. Mixin 21](#_Toc26954446)

[7. Trait 21](#_Toc155895689)

[8. Babel转码器的支持 21](#_Toc1914872321)

[第二十章 Module的语法 21](#_Toc2039795796)

[1. 概述 22](#_Toc1912933874)

[2. 严格模式 22](#_Toc202742392)

[3. export 命令 22](#_Toc535938192)

[4. import 命令 22](#_Toc866893289)

[5. 模块的整体加载 22](#_Toc1434109200)

[6. export default 命令 22](#_Toc1180915507)

[7. export 与 import 的复合写法 22](#_Toc760855066)

[8. 模块的继承 22](#_Toc140815799)

[9. 跨模块常量 22](#_Toc802509026)

[10. import() 22](#_Toc1777853526)

[第二十一章 Module的加载实现 22](#_Toc604527235)

[1. 浏览器加载 23](#_Toc2055419207)

[2. ES6 模块与 CommonJS 模块的差异 23](#_Toc122986603)

[3. Node 加载 23](#_Toc464477199)

[4. 循环加载 23](#_Toc1175363596)

[5. ES6模块的转码 23](#_Toc6203256)

[第二十二章 编程风格 23](#_Toc628243400)

[1. 块级作用域 23](#_Toc279563379)

[2. 字符串 24](#_Toc227524011)

[3. 解构赋值 24](#_Toc126775548)

[4. 对象 24](#_Toc983907582)

[5. 数组 24](#_Toc1976697963)

[6. 函数 24](#_Toc260202170)

[7. Map结构 24](#_Toc1714874232)

[8. Class 24](#_Toc2076387427)

[9. 模块 24](#_Toc382321689)

[10. ESLint的使用 24](#_Toc1490803462)

# Let和const命令

## Let命令

### 基本用法

Let命令，是es６新特性，用来声明变量，但是变量的作用域仅限于let命令所在的块内。





以前for循环体内很难调用循环变量，现在使用let就能很方便解决。

for循环还有一个特点就是循环体与循环语句的作用域不是一个作用域



### 不存在变量提升

Let不存在变量提升，因此只能在let声明语句之后才能调用。



### 暂时性死区

只要块级作用域内存在let命令，它所声明的变量就绑定这个区域，不再受外部影响，es6明确规定，如果区块中存在let和const命令，这个区块对这些命令声明的变量从一开始就形成封闭作用域。凡是在声明之前调用变量就会报错。



### 不允许重复声明变量





## Const命令

### 基本用法

Const声明一个只读的常量。一旦声明，常量的值就不能改变，const声明的变量不能改变，这就意味着，const声明的变量，就必须立即初始化，不能留到以后赋值。



### 基本性质

与let类似：

块级作用域

变量不提升

暂时性死区

不能重复定义

**本质**

Const实际上保证的并不是变量的值不得改变的是变量指向的内存地址，对于简单类型的数据（数值、字符串、布尔值），值就是保存在变量指向的那个地址，因此等同于常量。但是对于符合类型的数据（数组、对象），变量指向的内存地址，保存的只是一个指针，const只能保证这个指针不变，至于指针指向的数据结构是可变的。



## 顶层对象的属性

Es6以前全局变量是和顶层对象的属性挂钩的，这样会导致意外的事情发生，但是es6的let和const就解决了这个问题



Es6为了和以前版本兼容，var和function 声明的全局变量仍然与顶层对象属性相挂钩，但是let、const声明的全局变量不属于顶层对象的属性。



## Global对象

顶层对象：浏览器：window

Web worker: self

Node:global

为了同意标准es6同意把顶层对象定位global.

# 变量的解构和赋值

## [数组的解构赋值](http://es6.ruanyifeng.com/" \l "docs/destructuring#数组的解构赋值)

### 基本用法

Es6允许按照一定模式，从数组和对象中提取值，对变量进行赋值，这被称为解构。



在本质上，这种写法属于‘模式匹配’，只要等号两边的模式相同，左边的变量就会被赋予对应的值。





**不完全解构**





### 默认值

解构赋值允许默认值

## [对象的解构赋值](http://es6.ruanyifeng.com/" \l "docs/destructuring#对象的解构赋值)

解构赋值不仅可以用于数组，还可以用于对象



和数组有一个重要的不同，数组元素是按次序排序的，变量的取值有他的位置决定；而对象的属性没有次序，变量必须与属性同名，才能正确的值。

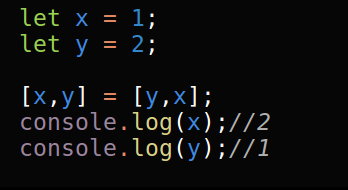


## [字符串的解构赋值](http://es6.ruanyifeng.com/" \l "docs/destructuring#字符串的解构赋值)



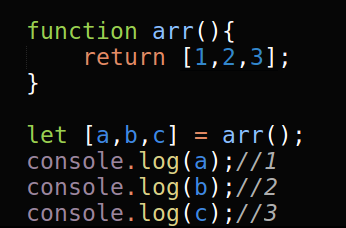
## [用途](http://es6.ruanyifeng.com/" \l "docs/destructuring#用途)

### 交换变量的值

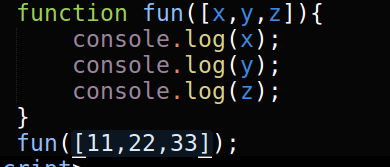


### 从函数返回多个值

函数只能返回一个值，如果向返回多个值需要将他们放在数组或者对象里面，有了解构赋值，就可以直接返回多个值



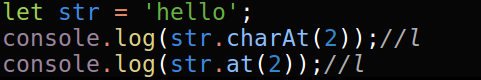
**函数的参数**



# 字符串的扩展

## [at()](http://es6.ruanyifeng.com/" \l "docs/string#at())

ES5提供charAt();函数用于返回字符串指定位置的字符，ES6同样提供了at()函数，功能与charAt一样不过是，是别的字符更加广泛。



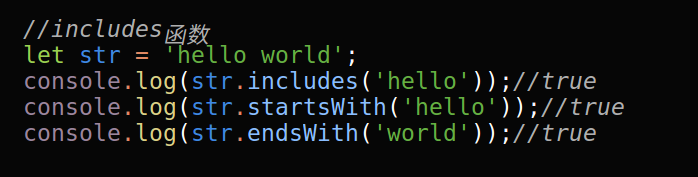
## [includes(), startsWith(), endsWith()](http://es6.ruanyifeng.com/" \l "docs/string#includes(), startsWith(), endsWith())

传统js只有indexOf()方法，可以确定一个字符串是否包含在另一个字符串中，ES6又提供了三种方法

includes(); 返回布尔值，表示是否找到参数字符串

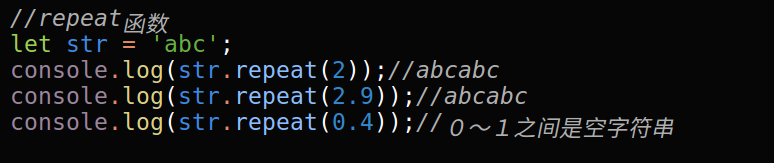
startsWith(); 返回布尔值，表示参数字符串是否在源字符串头部

endsWith();返回布尔值，表示参数字符串是否在源字符串尾部．



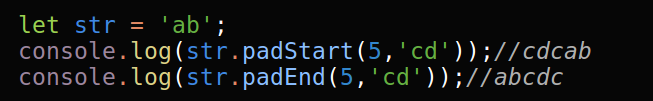
## [repeat()](http://es6.ruanyifeng.com/" \l "docs/string#repeat())

repeat()方法返回一个新的字符串，表示将源字符串重复Ｎ次



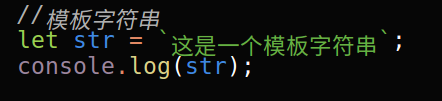
## [padStart()，padEnd()](http://es6.ruanyifeng.com/" \l "docs/string#padStart()，padEnd())

ES2017引入了字符串补全长度的功能，padStart()是从头部补全，padEnd()是从尾部补全



## [模板字符串](http://es6.ruanyifeng.com/" \l "docs/string#模板字符串)

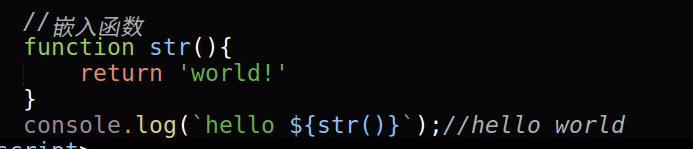
模板字符串（template string）是增强版字符串，用反引号(`)标识。他可以当做普通字符串使用，也可以用来定义多行字符串，或者在字符串中嵌入变量。



模板字符串可以嵌入，变量、表达式、函数，但是必须放在${}的大括号中。



嵌入函数



# 正则的扩展

## [RegExp构造函数](http://es6.ruanyifeng.com/" \l "docs/regex#RegExp构造函数)

## [字符串的正则方法](http://es6.ruanyifeng.com/" \l "docs/regex#字符串的正则方法)

## [u修饰符](http://es6.ruanyifeng.com/" \l "docs/regex#u修饰符)

## [y 修饰符](http://es6.ruanyifeng.com/" \l "docs/regex#y 修饰符)

## [sticky属性](http://es6.ruanyifeng.com/" \l "docs/regex#sticky属性)

## [flags属性](http://es6.ruanyifeng.com/" \l "docs/regex#flags属性)

## [RegExp.escape()](http://es6.ruanyifeng.com/" \l "docs/regex#RegExp.escape())

## [s 修饰符：dotAll 模式](http://es6.ruanyifeng.com/" \l "docs/regex#s 修饰符：dotAll 模式)

## [后行断言](http://es6.ruanyifeng.com/" \l "docs/regex#后行断言)

## [Unicode属性类](http://es6.ruanyifeng.com/" \l "docs/regex#Unicode属性类)

# 数值的扩展

## [二进制和八进制表示法](http://es6.ruanyifeng.com/" \l "docs/number#二进制和八进制表示法)

## [Number.isFinite(), Number.isNaN()](http://es6.ruanyifeng.com/" \l "docs/number#Number.isFinite(), Number.isNaN())

## [Number.parseInt(), Number.parseFloat()](http://es6.ruanyifeng.com/" \l "docs/number#Number.parseInt(), Number.parseFloat())

## [Number.isInteger()](http://es6.ruanyifeng.com/" \l "docs/number#Number.isInteger())

## [Number.EPSILON](http://es6.ruanyifeng.com/" \l "docs/number#Number.EPSILON)

## [安全整数和Number.isSafeInteger()](http://es6.ruanyifeng.com/" \l "docs/number#安全整数和Number.isSafeInteger())

## [Math对象的扩展](http://es6.ruanyifeng.com/" \l "docs/number#Math对象的扩展)

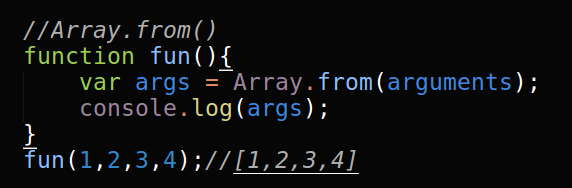
## [Math.signbit()](http://es6.ruanyifeng.com/" \l "docs/number#Math.signbit())

## [指数运算符](http://es6.ruanyifeng.com/" \l "docs/number#指数运算符)

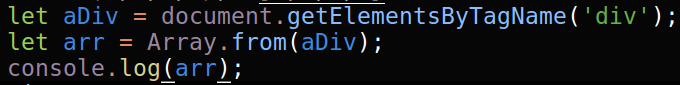
# 数组的扩展

## [Array.from()](http://es6.ruanyifeng.com/" \l "docs/array#Array.from())

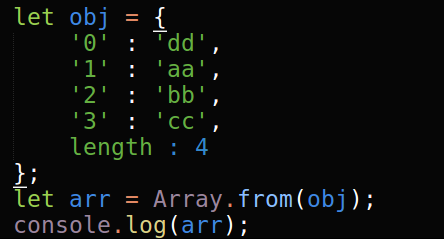
Array.from()的作用是将类似数组的对象和可遍历的对象转为真正的数组。实际应用中函数内部的arguments对象，就可以转化为真正数组。



DOM操作返回的Nodelist集合也可以转换为数组

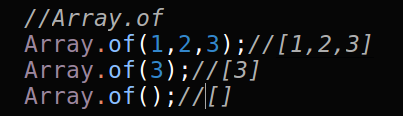


对于类似数组的对象，只要有length属性就可以使用Array.from转化为数组。

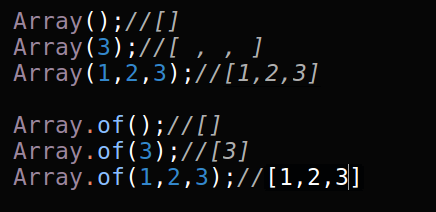


## [Array.of()](http://es6.ruanyifeng.com/" \l "docs/array#Array.of())

Array.of方法用于将一组值转化为数组

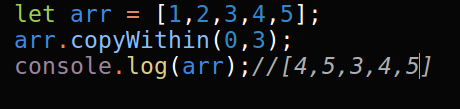


与Array()声明数组不同，Array()会根据参数的个数不同返回不同的结果，而Array.of()总是返回参数组成的数组



## [数组实例的copyWithin()](http://es6.ruanyifeng.com/" \l "docs/array#数组实例的copyWithin())

数组实例的copyWithin方法，在当前数组内部，将指定位置的成员赋值到其他位置（会覆盖原有成员），然后返回当前数组。也就是能改变当前数组



copyWithin(target,start,end)函数有三个参数：

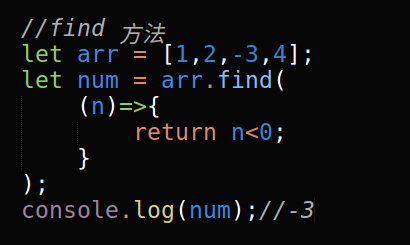
target(必须):从开始位置替换数据

start(可选)：从该位置开始读取数据

end(可选)：到该位置前停止读取数据

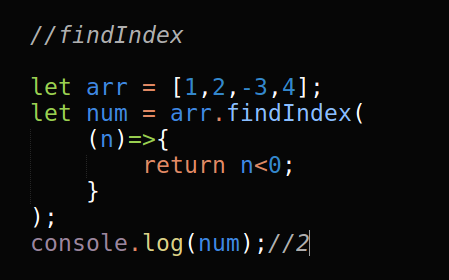
## [数组实例的find()和findIndex()](http://es6.ruanyifeng.com/" \l "docs/array#数组实例的find()和findIndex())

Find方法用于找出一个符合条件的数组成员。他的参数是一个回调函数，所有数组成员依次执行该回调函数，直到找出第一个返回值为true的成员，然后返回改成员。如果没有符合条件的成员返回undefined。



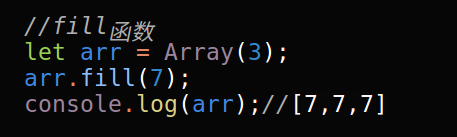
find方法的回调函数的参数可以接受三个，分别是，当前值、当前位置、当前数组本身。

findIndex 方法和find方法基本一致，只是返回的结果不同他返回的结果是符合条件的成员的位置。

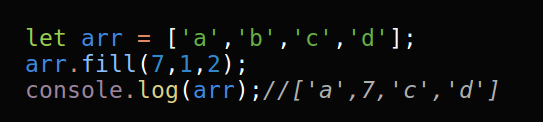


## [数组实例的fill()](http://es6.ruanyifeng.com/" \l "docs/array#数组实例的fill())

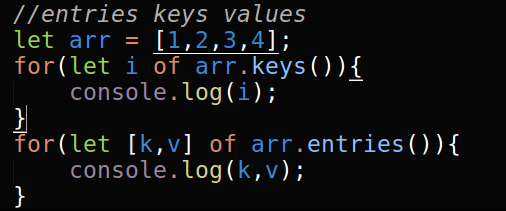
使用指定值，填充数组



fill函数还可以接收第二个和第三个参数，分别表示填充的起始位置，和结束位置。

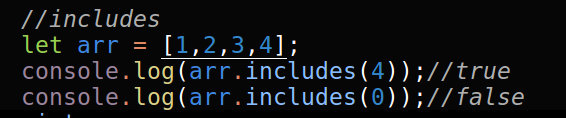


## [数组实例的entries()，keys()和values()](http://es6.ruanyifeng.com/" \l "docs/array#数组实例的entries()，keys()和values())



## [数组实例的includes()](http://es6.ruanyifeng.com/" \l "docs/array#数组实例的includes())

用于检查数组中是否含有指定成员。



# 函数的扩展

## [函数参数的默认值](http://es6.ruanyifeng.com/" \l "docs/function#函数参数的默认值)

## [rest参数](http://es6.ruanyifeng.com/" \l "docs/function#rest参数)

## [扩展运算符](http://es6.ruanyifeng.com/" \l "docs/function#扩展运算符)

## [严格模式](http://es6.ruanyifeng.com/" \l "docs/function#严格模式)

## [name 属性](http://es6.ruanyifeng.com/" \l "docs/function#name 属性)

## [箭头函数](http://es6.ruanyifeng.com/" \l "docs/function#箭头函数)

## [绑定 this](http://es6.ruanyifeng.com/" \l "docs/function#绑定 this)

## [尾调用优化](http://es6.ruanyifeng.com/" \l "docs/function#尾调用优化)

## [函数参数的尾逗号](http://es6.ruanyifeng.com/" \l "docs/function#函数参数的尾逗号)

# 对象的扩展

## [属性的简洁表示法](http://es6.ruanyifeng.com/" \l "docs/object#属性的简洁表示法)

## [属性名表达式](http://es6.ruanyifeng.com/" \l "docs/object#属性名表达式)

## [方法的 name 属性](http://es6.ruanyifeng.com/" \l "docs/object#方法的 name 属性)

## [Object.is()](http://es6.ruanyifeng.com/" \l "docs/object#Object.is())

## [Object.assign()](http://es6.ruanyifeng.com/" \l "docs/object#Object.assign())

## [属性的可枚举性](http://es6.ruanyifeng.com/" \l "docs/object#属性的可枚举性)

## [属性的遍历](http://es6.ruanyifeng.com/" \l "docs/object#属性的遍历)

## [\_\_proto\_\_　Object.setPrototypeOf()，Object.getPrototypeOf()](http://es6.ruanyifeng.com/" \l "docs/object#__proto__属性，Object.setPrototypeOf()，Object.getPrototypeOf())

## [Object.key(),Object.values(),Object.entries()](http://es6.ruanyifeng.com/" \l "docs/object#Object.keys()，Object.values()，Object.entries())

## [对象的扩展运算符](http://es6.ruanyifeng.com/" \l "docs/object#对象的扩展运算符)

## [Object.getOwnPropertyDescriptors()](http://es6.ruanyifeng.com/" \l "docs/object#Object.getOwnPropertyDescriptors())

## [Null 传导运算符](http://es6.ruanyifeng.com/" \l "docs/object#Null 传导运算符)

# Symbol

## [概述](http://es6.ruanyifeng.com/" \l "docs/symbol#概述)

## [作为属性名的Symbol](http://es6.ruanyifeng.com/" \l "docs/symbol#作为属性名的Symbol)

## [实例：消除魔术字符串](http://es6.ruanyifeng.com/" \l "docs/symbol#实例：消除魔术字符串)

## [属性名的遍历](http://es6.ruanyifeng.com/" \l "docs/symbol#属性名的遍历)

## [Symbol.for()，Symbol.keyFor()](http://es6.ruanyifeng.com/" \l "docs/symbol#Symbol.for()，Symbol.keyFor())

## [实例：模块的 Singleton 模式](http://es6.ruanyifeng.com/" \l "docs/symbol#实例：模块的 Singleton 模式)

## [内置的Symbol值](http://es6.ruanyifeng.com/" \l "docs/symbol#内置的Symbol值)

# Set和map数据解构

## [Set](http://es6.ruanyifeng.com/" \l "docs/set-map#Set)

## [WeakSet](http://es6.ruanyifeng.com/" \l "docs/set-map#WeakSet)

## [Map](http://es6.ruanyifeng.com/" \l "docs/set-map#Map)

## [WeakMap](http://es6.ruanyifeng.com/" \l "docs/set-map#WeakMap)

# Proxy

## [概述](http://es6.ruanyifeng.com/" \l "docs/proxy#概述)

## [Proxy 实例的方法](http://es6.ruanyifeng.com/" \l "docs/proxy#Proxy 实例的方法)

## [Proxy.revocable()](http://es6.ruanyifeng.com/" \l "docs/proxy#Proxy.revocable())

## [this 问题](http://es6.ruanyifeng.com/" \l "docs/proxy#this 问题)

## [实例：Web 服务的客户端](http://es6.ruanyifeng.com/" \l "docs/proxy#实例：Web 服务的客户端)

# Reflect

## [概述](http://es6.ruanyifeng.com/" \l "docs/reflect#概述)

## [静态方法](http://es6.ruanyifeng.com/" \l "docs/reflect#静态方法)

## [实例：使用 Proxy 实现观察者模式](http://es6.ruanyifeng.com/" \l "docs/reflect#实例：使用 Proxy 实现观察者模式)

# Promise对象

## [Promise 的含义](http://es6.ruanyifeng.com/" \l "docs/promise#Promise 的含义)

## [基本用法](http://es6.ruanyifeng.com/" \l "docs/promise#基本用法)

## [Promise.prototype.then()](http://es6.ruanyifeng.com/" \l "docs/promise#Promise.prototype.then())

## [Promise.prototype.catch()](http://es6.ruanyifeng.com/" \l "docs/promise#Promise.prototype.catch())

## [Promise.all()](http://es6.ruanyifeng.com/" \l "docs/promise#Promise.all())

## [Promise.race()](http://es6.ruanyifeng.com/" \l "docs/promise#Promise.race())

## [Promise.resolve()](http://es6.ruanyifeng.com/" \l "docs/promise#Promise.resolve())

## [Promise.reject()](http://es6.ruanyifeng.com/" \l "docs/promise#Promise.reject())

## [两个有用的附加方法](http://es6.ruanyifeng.com/" \l "docs/promise#两个有用的附加方法)

## [应用](http://es6.ruanyifeng.com/" \l "docs/promise#应用)

## [Promise.try()](http://es6.ruanyifeng.com/" \l "docs/promise#Promise.try())

# Iterator和for...of循环

## [Iterator（遍历器）的概念](http://es6.ruanyifeng.com/" \l "docs/iterator#Iterator（遍历器）的概念)

## [数据结构的默认Iterator接口](http://es6.ruanyifeng.com/" \l "docs/iterator#数据结构的默认Iterator接口)

## [调用Iterator接口的场合](http://es6.ruanyifeng.com/" \l "docs/iterator#调用Iterator接口的场合)

## [字符串的Iterator接口](http://es6.ruanyifeng.com/" \l "docs/iterator#字符串的Iterator接口)

## [Iterator接口与Generator函数](http://es6.ruanyifeng.com/" \l "docs/iterator#Iterator接口与Generator函数)

## [遍历器对象的return()，throw()](http://es6.ruanyifeng.com/" \l "docs/iterator#遍历器对象的return()，throw())

## [for...of循环](http://es6.ruanyifeng.com/" \l "docs/iterator#for...of循环)

# Generator函数的语法

## [简介](http://es6.ruanyifeng.com/" \l "docs/generator#简介)

## [next方法的参数](http://es6.ruanyifeng.com/" \l "docs/generator#next方法的参数)

## [for...of循环](http://es6.ruanyifeng.com/" \l "docs/generator#for...of循环)

## [Generator.prototype.throw()](http://es6.ruanyifeng.com/" \l "docs/generator#Generator.prototype.throw())

## [Generator.prototype.return()](http://es6.ruanyifeng.com/" \l "docs/generator#Generator.prototype.return())

## [yield\* 语句](http://es6.ruanyifeng.com/" \l "docs/generator#yield* 语句)

## [作为对象属性的Generator函数](http://es6.ruanyifeng.com/" \l "docs/generator#作为对象属性的Generator函数)

## [Generator函数的this](http://es6.ruanyifeng.com/" \l "docs/generator#Generator函数的this)

## [含义](http://es6.ruanyifeng.com/" \l "docs/generator#含义)

## [应用](http://es6.ruanyifeng.com/" \l "docs/generator#应用)

# Generator函数的异步应用

## [传统方法](http://es6.ruanyifeng.com/" \l "docs/generator-async#传统方法)

## [基本概念](http://es6.ruanyifeng.com/" \l "docs/generator-async#基本概念)

## [Generator 函数](http://es6.ruanyifeng.com/" \l "docs/generator-async#Generator 函数)

## [Thunk 函数](http://es6.ruanyifeng.com/" \l "docs/generator-async#Thunk 函数)

## [co 模块](http://es6.ruanyifeng.com/" \l "docs/generator-async#co 模块)

# Async函数

## [含义](http://es6.ruanyifeng.com/" \l "docs/async#含义)

## [用法](http://es6.ruanyifeng.com/" \l "docs/async#用法)

## [语法](http://es6.ruanyifeng.com/" \l "docs/async#语法)

## [async 函数的实现原理](http://es6.ruanyifeng.com/" \l "docs/async#async 函数的实现原理)

## [与其他异步处理方法的比较](http://es6.ruanyifeng.com/" \l "docs/async#与其他异步处理方法的比较)

## [实例：按顺序完成异步操作](http://es6.ruanyifeng.com/" \l "docs/async#实例：按顺序完成异步操作)

## [异步遍历器](http://es6.ruanyifeng.com/" \l "docs/async#异步遍历器)

# Class

## [Class基本语法](http://es6.ruanyifeng.com/" \l "docs/class#Class基本语法)

## [Class的继承](http://es6.ruanyifeng.com/" \l "docs/class#Class的继承)

## [原生构造函数的继承](http://es6.ruanyifeng.com/" \l "docs/class#原生构造函数的继承)

## [Class的取值函数（getter）和存值函数（setter）](http://es6.ruanyifeng.com/" \l "docs/class#Class的取值函数（getter）和存值函数（setter）)

## [Class 的 Generator 方法](http://es6.ruanyifeng.com/" \l "docs/class#Class 的 Generator 方法)

## [Class 的静态方法](http://es6.ruanyifeng.com/" \l "docs/class#Class 的静态方法)

## [Class的静态属性和实例属性](http://es6.ruanyifeng.com/" \l "docs/class#Class的静态属性和实例属性)

## [类的私有属性](http://es6.ruanyifeng.com/" \l "docs/class#类的私有属性)

## [new.target属性](http://es6.ruanyifeng.com/" \l "docs/class#new.target属性)

## [Mixin模式的实现](http://es6.ruanyifeng.com/" \l "docs/class#Mixin模式的实现)

# Decorator

## [类的修饰](http://es6.ruanyifeng.com/" \l "docs/decorator#类的修饰)

## [方法的修饰](http://es6.ruanyifeng.com/" \l "docs/decorator#方法的修饰)

## [为什么修饰器不能用于函数？](http://es6.ruanyifeng.com/" \l "docs/decorator#为什么修饰器不能用于函数？)

## [core-decorators.js](http://es6.ruanyifeng.com/" \l "docs/decorator#core-decorators.js)

## [使用修饰器实现自动发布事件](http://es6.ruanyifeng.com/" \l "docs/decorator#使用修饰器实现自动发布事件)

## [Mixin](http://es6.ruanyifeng.com/" \l "docs/decorator#Mixin)

## [Trait](http://es6.ruanyifeng.com/" \l "docs/decorator#Trait)

## [Babel转码器的支持](http://es6.ruanyifeng.com/" \l "docs/decorator#Babel转码器的支持)

# Module的语法

## [概述](http://es6.ruanyifeng.com/" \l "docs/module#概述)

## [严格模式](http://es6.ruanyifeng.com/" \l "docs/module#严格模式)

## [export 命令](http://es6.ruanyifeng.com/" \l "docs/module#export 命令)

## [import 命令](http://es6.ruanyifeng.com/" \l "docs/module#import 命令)

## [模块的整体加载](http://es6.ruanyifeng.com/" \l "docs/module#模块的整体加载)

## [export default 命令](http://es6.ruanyifeng.com/" \l "docs/module#export default 命令)

## [export 与 import 的复合写法](http://es6.ruanyifeng.com/" \l "docs/module#export 与 import 的复合写法)

## [模块的继承](http://es6.ruanyifeng.com/" \l "docs/module#模块的继承)

## [跨模块常量](http://es6.ruanyifeng.com/" \l "docs/module#跨模块常量)

## [import()](http://es6.ruanyifeng.com/" \l "docs/module#import())

# Module的加载实现

## [浏览器加载](http://es6.ruanyifeng.com/" \l "docs/module-loader#浏览器加载)

## [ES6 模块与 CommonJS 模块的差异](http://es6.ruanyifeng.com/" \l "docs/module-loader#ES6 模块与 CommonJS 模块的差异)

## [Node 加载](http://es6.ruanyifeng.com/" \l "docs/module-loader#Node 加载)

## [循环加载](http://es6.ruanyifeng.com/" \l "docs/module-loader#循环加载)

## [ES6模块的转码](http://es6.ruanyifeng.com/" \l "docs/module-loader#ES6模块的转码)

# 编程风格

## [块级作用域](http://es6.ruanyifeng.com/" \l "docs/style#块级作用域)

## [字符串](http://es6.ruanyifeng.com/" \l "docs/style#字符串)

## [解构赋值](http://es6.ruanyifeng.com/" \l "docs/style#解构赋值)

## [对象](http://es6.ruanyifeng.com/" \l "docs/style#对象)

## [数组](http://es6.ruanyifeng.com/" \l "docs/style#数组)

## [函数](http://es6.ruanyifeng.com/" \l "docs/style#函数)

## [Map结构](http://es6.ruanyifeng.com/" \l "docs/style#Map结构)

## [Class](http://es6.ruanyifeng.com/" \l "docs/style#Class)

## [模块](http://es6.ruanyifeng.com/" \l "docs/style#模块)

## [ESLint的使用](http://es6.ruanyifeng.com/" \l "docs/style#ESLint的使用)