# [JS 对象之扩展、密封及冻结三大特性](https://segmentfault.com/a/1190000003894119)

扩展特性

Object.isExtensible 判断对象是否可扩展

Object.preventExtensions 将对象变为不可扩展

密封特性

Object.isSealed 判断对象是否密封

Object.seal 将一个对象密封

冻结特性

Object.isFrozen 判断一个对象是否冻结

Object.freeze 冻结一个对象

获取计算后的样式属性(IE9以上支持，只读不可写)

getComputedStyle(document.getElementByClassName(‘…’))[‘top’]

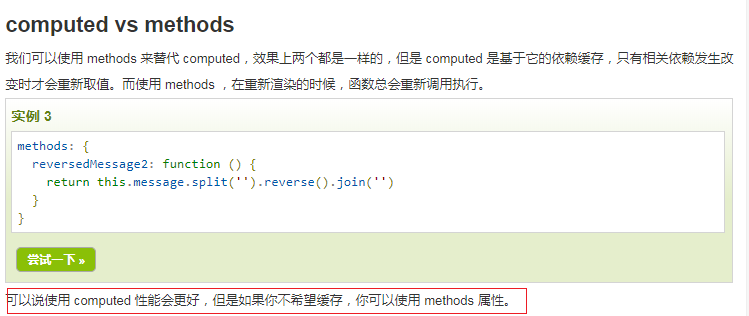
# 当一个 form 元素中只有一个输入框时，在该输入框中按下回车应提交该表单。如果希望阻止这一默认行为，可以在 标签上添加 @submit.native.prevent

# VUE

## Computed

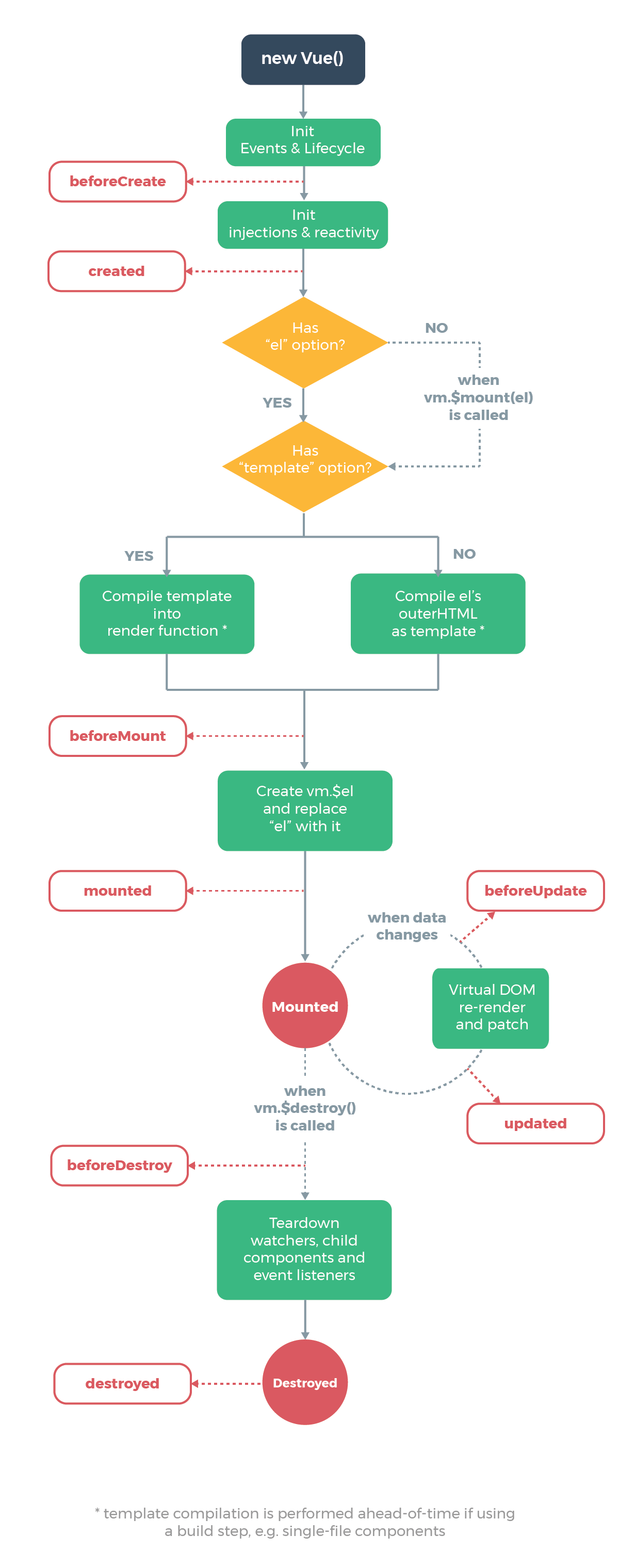
假设我们有一个性能开销比较大的的计算属性 **A**，它需要遍历一个巨大的数组并做大量的计算。然后我们可能有其他的计算属性依赖于 **A** 。如果没有缓存，我们将不可避免的多次执行 **A** 的 getter！如果你不希望有缓存，请用方法来替代。

## Computed与methods





## VUE生命周期



## 注意事项

由于 JavaScript 的限制，Vue 不能检测以下变动的数组：

1. 当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue
2. 当你修改数组的长度时，例如：vm.items.length = newLength

为了解决第一类问题，以下两种方式都可以实现和 vm.items[indexOfItem] = newValue 相同的效果，同时也将触发状态更新：

|  |
| --- |
| // Vue.set Vue.set(example1.items, indexOfItem, newValue) |

|  |
| --- |
| // Array.prototype.splice example1.items.splice(indexOfItem, 1, newValue) |

为了解决第二类问题，你可以使用 splice：

|  |
| --- |
| example1.items.splice(newLength) |

## $nexTick

Vue实现响应式并不是数据发生变化之后DOM立即变化，而是按照一定的策略进行DOM更新。

作用：DOM更新循环之后的延迟回调，当我们改变了数据之后，DOM的渲染需要时间，然而我们希望去操作DOM元素，就需要等渲染完成后再去操作，就需要用到nextTick

原理：Vue内部分三种情况来实现nextTick

       1、Promise：当浏览器支持promise时使用

       2、MutationObserver: 也是在浏览器支持时使用，这个API是用来监视DOM变动，DOM的任何变动比如节点的增删改及文本的变动，这个API都可以得到通知，可以理解成是一个事件，但是跟事件有一个本质不同：事件是同步的，也就是说DOM变动会立刻触发相应事件，

           而MutationObserver是异步触发，要等到当前DOM操作都结束后才触发，这样是为了应对DOM变动频繁

       3、当以上两种方法都不支持时，用setTimeout(handler,0)

## Keep-alive注意事项

当引入keep-alive时，页面第一次进入，钩子的触发顺序created -> mounted -> activated,退出时触发deactivated。当再次进入（前进或后退）时，只触发activated。

## 路由懒加载

当打包构建应用时，Javascript 包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

https://router.vuejs.org/zh/guide/advanced/lazy-loading.html#%E6%8A%8A%E7%BB%84%E4%BB%B6%E6%8C%89%E7%BB%84%E5%88%86%E5%9D%97

# 正则表达式

// ip (127.0.0.1)

ip: /^(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9]|0)\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]{1}[0-9]{1}|[0-9])$/,

// port (0～65535)

port: /^([0-9]|[1-9]\d{1,3}|[1-5]\d{4}|6[0-5]{2}[0-3][0-5])$/,

// email

email: /(^\w+((-\w+)|(\.\w+))\*@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)\*\.[A-Za-z0-9]+$)|(^$)/,

// mac地址

macAddr: /^([A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2}$/,

// 数字

number: /^\d\*$/,

// 自然数 0 1 2 3 4

naturalNum: /^(0|[1-9]\d\*)$/,

// 正整数 1 2 3 4

positiveNum: /^[1-9]\d\*$/,

// 1～32个字符；不能包含 ' / \ : \* ? " < > | 等特殊字符

name: /^[^'\/\\:\\*\?"<>\|]\*$/,

// 1-32个字符，只能用字母、数字、汉字、小数点、下划线、连接符、括号 「设备型号」

deviceType: /^[a-zA-Z0-9\.\-\_\(\)\u4E00-\u9FA5]\*$/,

// 字母、数字、汉字、小数点、下划线、连接符 [设备编码]

deviceCode: /^[a-zA-Z0-9\.\-\_\u4E00-\u9FA5#()\s]\*$/,

// 设备登录用户名 1～32个字符，只能用字母、数字、汉字、小数点、下划线、连接符。

deviceUserName: /^[a-zA-Z0-9\.\-\_\u4E00-\u9FA5]\*$/,

// 字母、数字的组合 「设备编号1-32位」

numLetter: /^[\da-zA-Z]\*$/,

// 字母、数字、汉字的组合 「工号1-32」「入场播报」「出场播报」「放行播报」

numLetterAndCN: /^[\da-zA-Z\u4E00-\u9FA5]\*$/,

// 字母、汉字的组合 「职位1-128」

numAndCN: /^[a-zA-Z\u4E00-\u9FA5]\*$/,

// 字母、数字和除:\"之外的特殊字符 「设备账号1-32位」

deviceAccount: /^[a-zA-Z0-9~`!@#\$%^&\*()\_+\-=\[\];',<.>/|?]{1,32}$/,

// 车牌号

licenseNumber: /^.{1,16}$/,

// 手机号

phoneNum: /^\d{1,11}$/,

// 姓名 1-128个数字、字母、汉字、间隔号（·）和空格

personName: /^[\da-zA-Z\u4E00-\u9FA5· ]{1,128}$/,

// 证件号码

IDNumber: /^[0-9a-zA-Z]{0,20}$/,

// 身份证号 18位，前17位为数字，尾号为数字或大写字母X。

identityCard: /^\d{17}[\d|X]$/,

// 密码为8-16位字符，至少由大写字母、小写字母、数字、特殊字符任意两种组成。

password: /^(?![A-Z]+$)(?![a-z]+$)(?!\d+$)(?![\W\_]+$)\S{8,16}$/,

// 卡号 8-18个字符，只能用数字和大写字母。

cardNum: /^[0-9A-Z]{8,18}$/,

// 4-8位数字 「卡密码」「胁迫密码」「超级密码」

simplePassword: /^\d{4,8}$/,

// UUID 32位大写英文字母、数字、连接符。

uuid: /^[a-zA-Z\d]{8}-[a-zA-Z\d]{4}-[a-zA-Z\d]{4}-[a-zA-Z\d]{4}-[a-zA-Z\d]{12}$/

# 对输入框绑定键盘事件

<input v-model="dataInput" v-on:keyup.13="addDataItem()" ></input>

# 英文单词换行问题

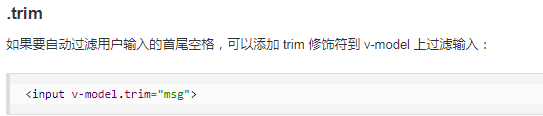
word-break: keep-all;

word-wrap: break-word;

white-space: pre-wrap;

# 事件修饰符





# 前端axios拦截器

<https://www.cnblogs.com/guoxianglei/p/7084506.html>

# 网页标题更新

router.afterEach(function() {  
if (router.app.$route.title != '')  
document.title = $router.app.$route.title;  
console.log(router.app.$route.title)  
});

# git 强制覆盖本地代码

git fetch --all

git reset --hard origin/master

git pull

单条命令

git fetch --all && git reset --hard origin/master && git pull

# 使用jsencrypt实现参数的前端加密

# Async与await

1. async表示一个异步函数，而await只能在这个函数里面使用。
2. await表示在这里等待await后面的操作执行完毕，再执行下一句代码
3. await后面紧跟着的最好是一个耗时的操作或者是一个异步操作（非耗时的操作也可以，但是就失去了await的意义）

# VueX

## 核心（store仓库）

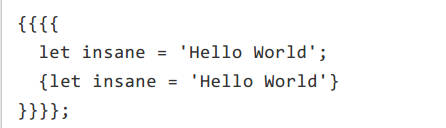
State

单一状态树

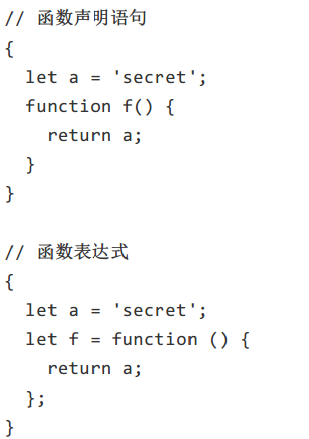
# ES6

## let与const

let类似于var,只是let有其单独的作用域，let只在它所在的代码块有效，即在let所在的大括号范围内（一个块级作用域）有效，且let不会发生变量提升，在它之前该变量是不存在的。Let不允许在相同的作用域内重复声明同一个变量，但可以在内层作用域定义外层作用域的同名变量。

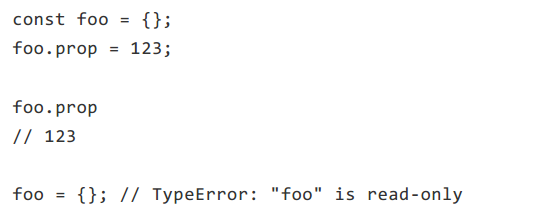


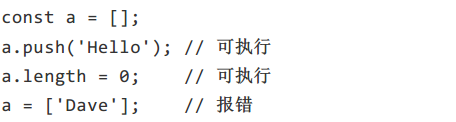
考虑到环境导致的行为差异太大，应该避免在块级作用域内声明函数。如果确实需要，也应该写成函数表达式，而不是函数声明语句。



Const声明一个只读的常量。一旦声明，常量的值就不能改变。与let相同，const只在声明所在的块级作用域内有效，也不存在变量提升，也不能重复声明。

对于复合类型的变量，变量名不指向数据，而是指向数据所在的地址。Const只是保证变量名指向的地址不变，并不保证改地址的数据不变，所以将一个对象声明为常量必须非常小心。



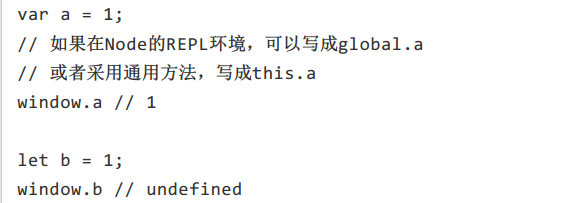


如果真的想将对象冻结，应该使用Object.freeze方法。

Const foo = Object.freeze({});

ES5只有两种声明变量的方法var和function，ES6除了let和const还有import和Class,所以ES6一共有6种声明变量的方法。

ES6中Var命令和function命令声明的全局变量，依旧是全局对象的属性，let、const、class命令声明的全局变量，不属于全局对象的属性。

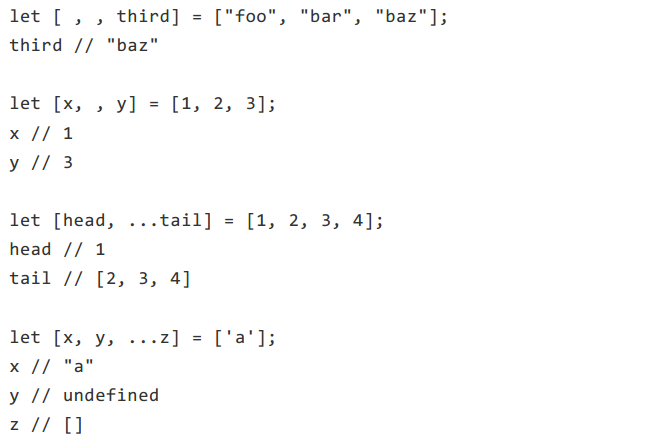


## 数组的解构赋值

ES6允许按照一定的模式，从数组和对象中提取值，对变量进行赋值，这被称为解构。

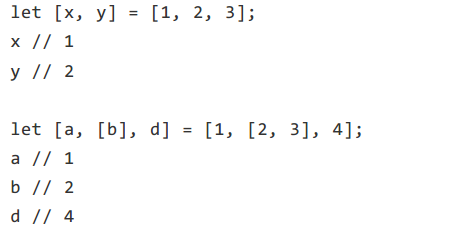
Var [a,b,c] = [1,2,3];即

Var a = 1,b = 2,c = 3;

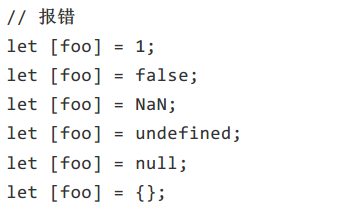


如果解构不成功，变量的值就等于undefined.

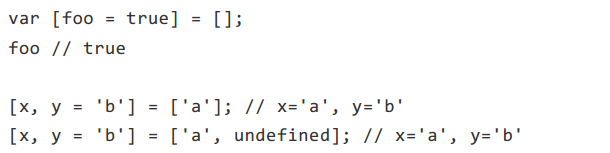
另一种情况是不完全解构，即等号左边的模式，只匹配一部分的等号右边的数组。这种情况下，解构依然可以成功。



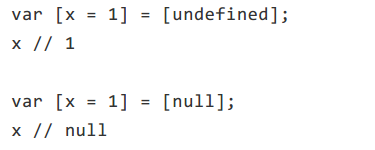
如果等号右边不是数组，那么将会报错。



解构允许指定默认值。

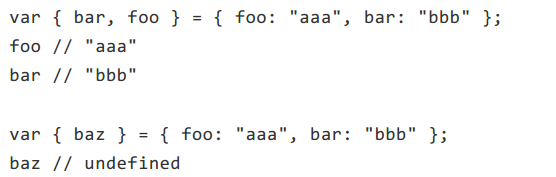


注意：ES6内部使用严格相等运算符（===）判断一个位置是否有值。所以，如果一个数组成员不严格等于undefined默认值是不会生效的。



## 对象的解构赋值

对象的解构与数组有一个重要的不同。数组是按照次序排列的，变量的取值由它的位置决定；而对象的顺序没有次序，变量必须与属性同名，才能取到正确的值。





注意：采用这种写法时，变量的声明和赋值是一体的。对于let和const来说，变量不能重新声明，所以一旦赋值的变量以前声明过就会报错。

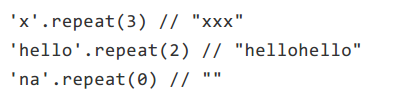
## 新增方法

### includes(),startsWith(),endsWith()



### repeat()

该方法返回一个新的字符串，表示将原字符串重复n次。



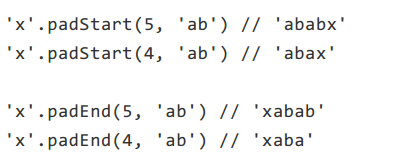
参数如果是小数将会被取整。

如果是负数或Infinity,会报错。

如果是字符串会先转化为数字。

### padStart(),padEnd()

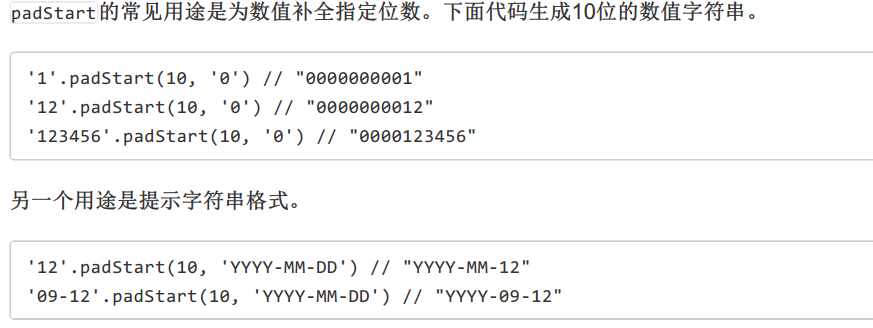
ES7推出了字符串补全长度的功能。如果某个字符串不够指定长度，会在头部或尾部补全。



指定两个参数，第一个参数用来指定字符串最小的长度，第二个参数是用来补全的字符串。

如果用来补全的字符串与原字符串两者的长度之和超过了指定的最小长度，则会截取超出位数的补全字符串。

如果省略第二个参数，则会用空格补全长度。



## 模板

### 模板字符串

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

模板字符串中嵌入变量，需要将变量名写在${ }之中。

大括号内可以放任意的js表达式，可以进行运算，以及引用对象属性。

模板字符串中还可以调用函数。

let [x,y] = [1,2];

`${x} + ${y} = ${x + y}`

//” 1 + 2 = 3”

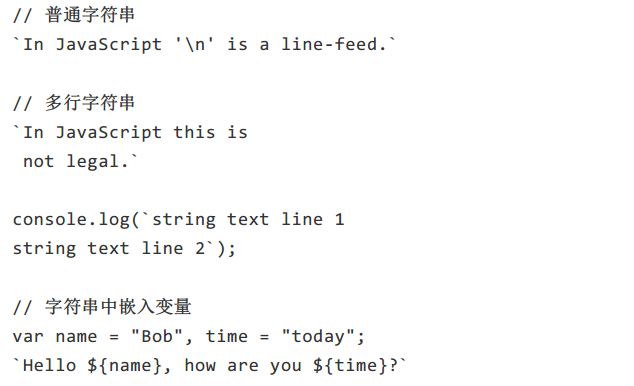
let fn = function (){

return ‘hello world’;

}

` ${ fn() } `

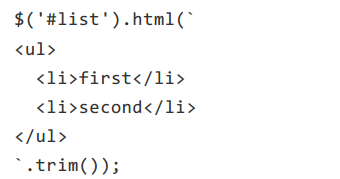
// “hello world”



如果在模板字符串中要使用反引号，则前面要用反斜杠转义。

如果使用模板字符串表示多行字符串，所有的空格和缩进都会保留在输出中。

如果不想要这个换行，可以使用trim方法消除它。



### 标签模板

一个重要的应用，就是过滤HTML字符串，防止用户输入恶意内容。

另一个应用，就是多语言转换（国际化处理）。

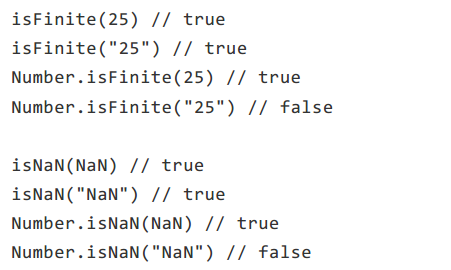
## 数值的扩展

### Number.isFinite( ), Number.isNaN( )

前者是用来检查一个数值是否为有限的。

后者用来检查一个值是否为NaN。

与传统的全局方法isFinite( )和isNaN( )区别在于，传统方法先调用Number()将非数值的值转化为数值，再进行判断，而这两个新方法只对数值有效，非数值一律返回false.



### Number.parseInt( ),Number.parseFloat( )

Es6将全局方法移植到Number对象上面，行为完全保持不变，目的是逐步减少全局性方法，使语言逐步模块化。

### Number.isInteger( )

判断一个值是否为整数。要注意的是，在js内部，整数和浮点数是同样的储存方法，所以3和3.0被视为同一个值。

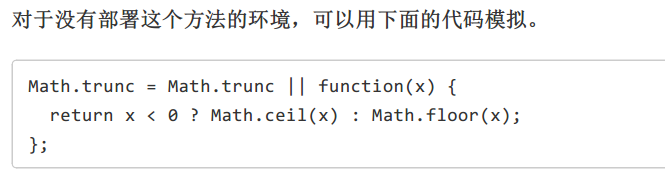
### Math对象的扩展

#### Math.trunc( )

去除一个数的小数部分，返回整数部分。

对于非数值，Math.trunc内部使用Number方法将其先转化为数值。

对于空值和无法截取整数的值，返回NaN。



#### Math.sign( )

判断一个数到底是正数、负数还是零。它会返回五种值：

正数返回+1

负数返回-1

0返回0

-0返回-0

其他值返回NaN

#### Math.cbrt( )

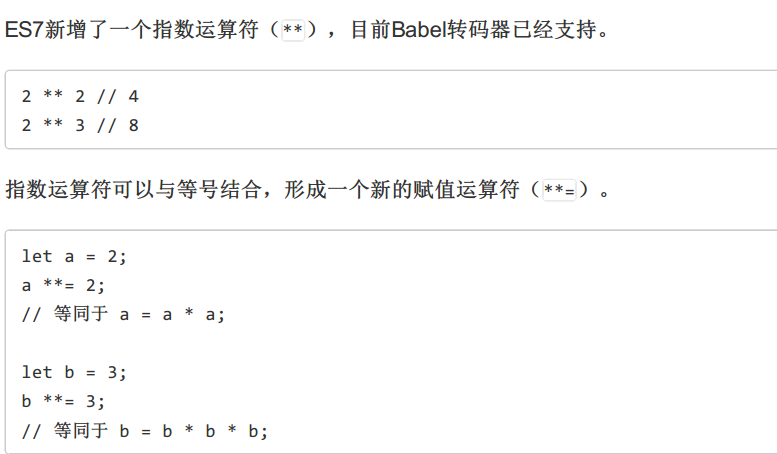
计算一个数的立方根。

对于非数值，Math.cbrt方法内部也是先使用Number方法将其转化为数值。

#### Math.hypot( )

返回所有参数的平方和的平方根。

### 指数运算符



## 数组的扩展

### Array.from( )

将两类对象转化为真正的数组：类似数组的对象和可遍历的对象。

Let arrayLike = {

‘0’:’a’,

‘1’:’b’,

‘2’:’c’,

length:3

};

//es5写法

Var arr1 = [ ].slice.call(arrayLike);//[‘a’,’b’,’c’]

//es6的写法

Let arr2 = Array.from(arrayLike);// [‘a’,’b’,’c’]

Array.from还可以接受第二个参数，作用类似于数组的map方法，用来对每个元素进行处理，将处理后的值放入返回的数组。

### Array.of( )

将一组值转换为数组。

Array.of(3,4,5)//[3,4,5]

Array.of总是返回参数值组成的数组，如果没有参数就返回一个空数组。

### copyWithin( )

在当前数组内部，将指定位置的成员复制到其他位置（会覆盖原有成员），然后返回当前数组。也就是说，该方法会修改当前数组。

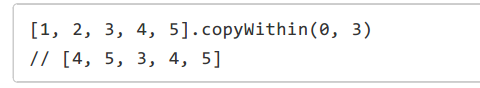
它接受三个参数：

Target(必须)：从该位置开始替换数据。

Start(可选)：从该位置开始读取数据，默认为0，如果为负值，表示倒数。

End(可选)：到该位置前停止读取数据，默认等于数组长度。如果为负值，表示倒数。

这三个参数应该都是数值，如果不是，会自动转化为数值。



### find( )和findIndex( )

find用于找出第一个符合条件的数组成员。它的参数是一个回调函数，所有的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，然后返回该成员。如果没有符合条件的成员，则返回undefined。

[1,2,3,4].find((n) => n>0) //1

findIndex方法的用法与find非常类似，返回第一个符合条件的数组成员的位置，如果所有成员都不符合条件，则返回-1。

这两个方法都可以接受第二个参数，用来绑定回调函数的this对象。

另外，这两个方法都可以发现NaN,弥补了数组的IndexOf方法的不足。

### fill( )

使用定值，填充一个数组。

[‘a’,’b’,’c’].fill(7) //[7,7,7]

new Array(3).fill(7) //[7,7,7]

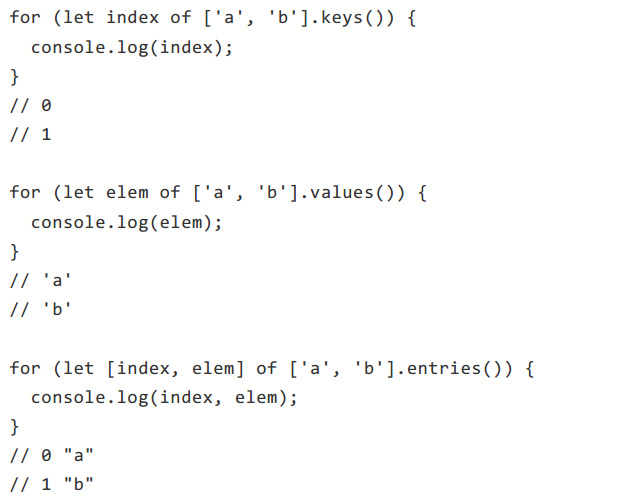
初始化空数组非常方便。数组中已有的元素，会被全部抹去。

Fill方法还可以接受第二个和第三个参数，用于指定填充的起始位置和结束位置。

[‘a’,’b’,’c’].fill(7,1,2) //[‘a’,7,’c’] 从1号位置开始，向原数组填充7，到2号位之前结束。

### keys(),values(),entries()

用于遍历数组。它们都返回一个遍历器对象，可以用for…of循环进行遍历，唯一的区别是keys()是对键名的遍历，values()是对键值的遍历，entries()是对键值对的遍历。



如果不使用for…of循环，可以手动调用遍历器对象的next方法进行遍历。

### Includes()

该方法返回一个布尔值，表示某个数组是否包含定的值，与字符串的includes方法类似。属于es7

该方法的第二个参数表示搜索的起始位置，默认为0。如果第二个参数为负数，则表示倒数的位置，如果这时它大于数组的长度，则会重置为0开始。

[1,2,3].includes(3,3) //false

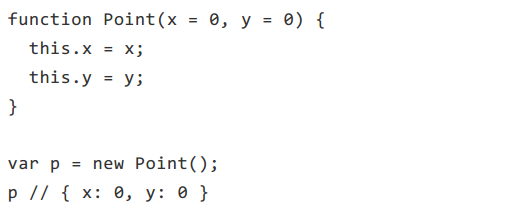
[1,2,3].includes(3,-1) //true

## 函数的扩展

### 函数参数的默认值

#### 基本用法

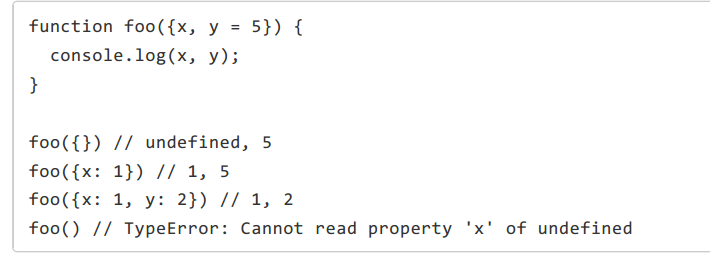
Es6允许为函数的参数设置默认值，即直接写在参数定义的后面。



参数变量是默认声明的，所以不能用let或const再次声明，否则会报错。

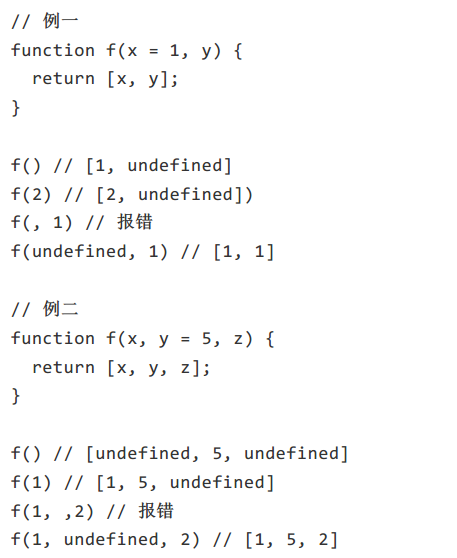
#### 与解构赋值默认值结合使用

只有当函数的参数是一个对象时，变量才会通过解构赋值而生成。



### 参数默认值的位置

通常情况下，定义了默认值的参数，该参数应该是函数的尾参数。这样比较容易看出来到底省略了哪些参数。如果非尾部的参数设置默认值，实际上这个参数是没办法省略的。



所以在函数调用时填入参数时，如果要使用该位置的默认参数时，要用undefined来占位，避免不必要的报错。（应该可以不用的参数都用undefined填上更保险）

### 函数的length属性

指定了默认值之后，函数的length属性将返回没有设置默认值的参数个数，也就是说，如果指定了默认值之后，length属性失真。Length属性的含义是，该函数预期传入的参数的个数。某个参数指定默认值之后，预期传入的参数个数就不包括这个参数了。同理，rest参数也不会计入length属性。

(function (a){}).length //1

(function (a = 0){}).length //0

(function (a,b,c = 1){}).length //2

(function (…args){}).length //0

如果设置默认值的参数不是尾参数，那么length属性也不再计算后面的参数了

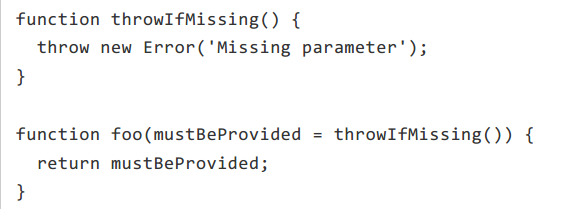
(function (a,b=1,c,d){}).length //1

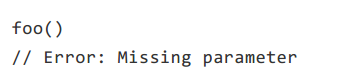
### 作用域

如果参数的默认值是一个变量，则该变量所处的作用域，与其他变量的作用域规则是一样的，即先是当前函数的作用域，然后才是全局作用域。

### 应用

利用参数默认值，可以指定某一个参数不得省略，如果省略就抛出错误。





另外可以将一个参数默认值设置为undefined，表明这个参数是可以省略的。

### rest参数

rest参数（形式为”…变量名”），用于获取函数的多余参数。Rest参数搭配的变量是一个数组，该变量将多余的参数放入数组中。Rest参数的变量代表一个数组，所以数组特有的方法都可以用于这个变量。

【注意】rest参数之后不能再有其他参数（即只能是最后一个参数），否则会报错。函数的length属性不包括rest参数。

### 扩展运算符

Spread是三个点（…）。好比是rest参数的逆运算，将一个数组转化为用逗号分隔的参数序列。

Console.log(…[1,2,3]) //1 2 3

该运算符主要用于函数调用。

#### 扩展运算符的应用

1. 合并数组：[…arr1,…arr2,…arr3]
2. 与解构赋值结合：

const [first,…rest] = [1,2,3,4,5]

first //1 rest//[2,3,4,5]

如果将扩展运算符用于数组赋值，只能放在参数的最后一位，否则会报错。

Const […rags,last] = [1,2,3,4,5]

//报错

1. 字符串

扩展运算符还可以将字符串转为真正的数组。

[…’hello’] //[‘h’,’e’,’l’,’l’,’o’]

该写法可以识别32位的unicode字符。

1. 实现了Iterator接口的对象，都可以用扩展运算符转化为真正的数组。

es6中有三类结构生来就具有Iterator接口：数组、类数组对象、Map和Set结构。

1. Generator函数运行后，返回一个遍历器对象，因此也可以使用扩展运算符。

### 箭头函数

箭头函数使用注意点：

1. 函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象，是固定的。实际原因是箭头函数根本没有自己的this，导致内部this就是外层代码块的this，正是因为它没有this，所以也就不能用作构造函数。所以当然也就不能用call(),apply(),bind()这些方法去改变this的指向。
2. 不可以当做构造函数，也就是说，不可以用new命令，否则会抛出一个错误。
3. 不可以使用yield命令，因此箭头函数不能用作generator函数。

## 对象的扩展

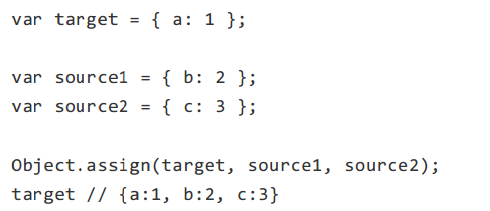
### Object.is()

用来比较两个值是否严格相等，与es5的严格比较运算符（===）的行为基本一致。

不同之处只有两个：一个是+0不等于-0，二是NaN等于自身。

### Object.assign()

用于对象的合并，将源对象的所有可枚举属性，复制到目标对象。



该方法的第一个参数是目标对象，后面的参数都是源对象。

【注意】如果目标对象与源对象有同名属性，或多个源对象有同名属性，则后面的属性会覆盖前面的属性。

如果只有一个参数，会直接返回该参数。

如果该参数不是对象，则会先转成对象，然后返回。

由于undefined和null无法转成对象，所以如果它们作为参数就会报错。