# [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)

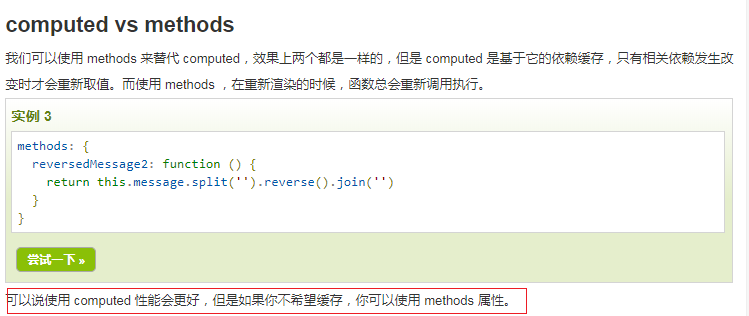
getComputedStyle(document.getElementByClassName(‘…’))[‘top’]

# VUE

## Computed

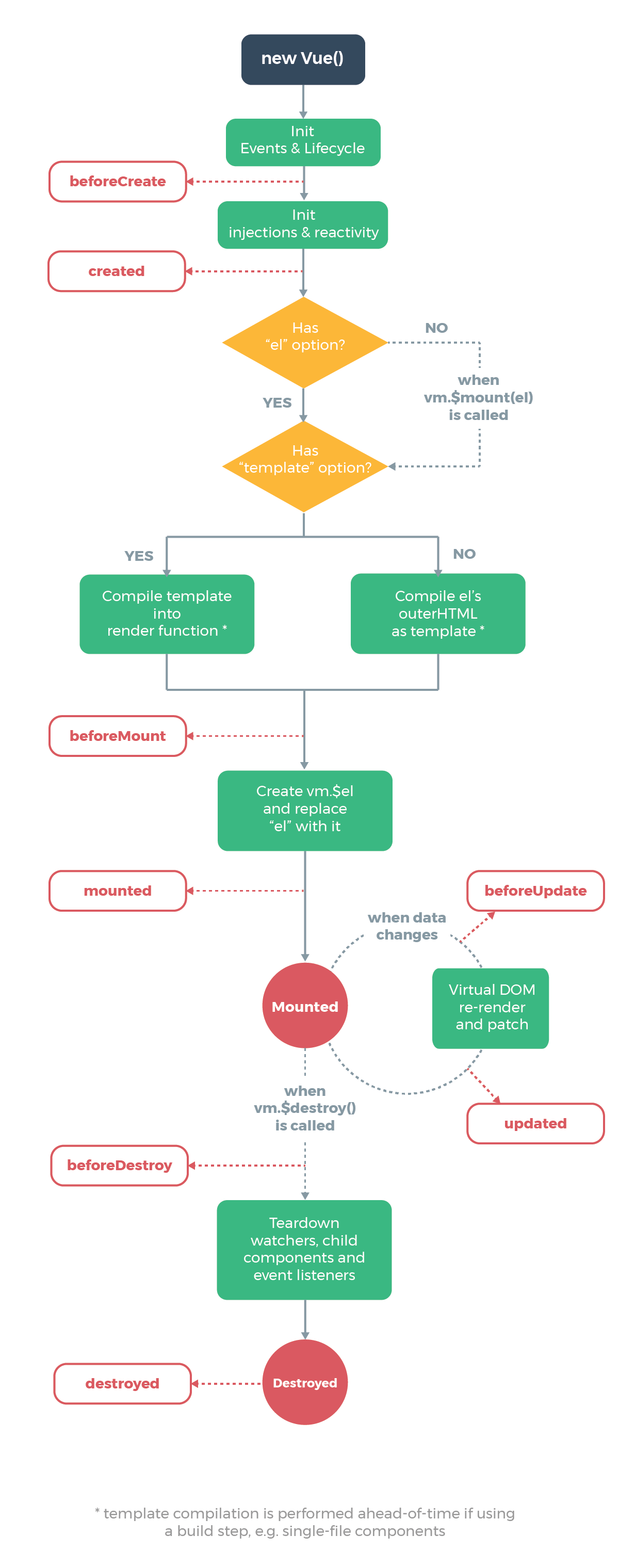
假设我们有一个性能开销比较大的的计算属性 **A**，它需要遍历一个巨大的数组并做大量的计算。然后我们可能有其他的计算属性依赖于 **A** 。如果没有缓存，我们将不可避免的多次执行 **A** 的 getter！如果你不希望有缓存，请用方法来替代。

## 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) |

# 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

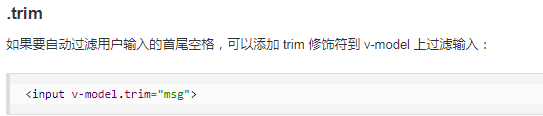
word-break: keep-all;

word-wrap: break-word;

white-space: pre-wrap;

# 事件修饰符





# VueX

## 核心（store仓库）

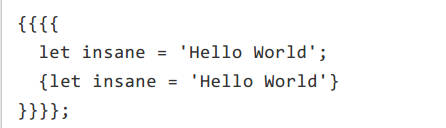
State

单一状态树

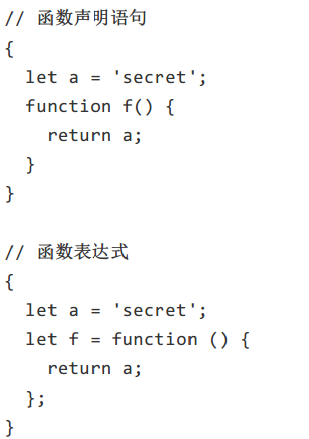
# ES6

## 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。

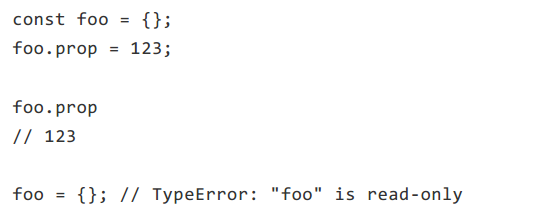


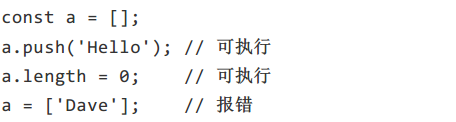
考虑到环境导致的行为差异太大，应该避免在块级作用域内声明函数。如果确实需要，也应该写成函数表达式，而不是函数声明语句。



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。



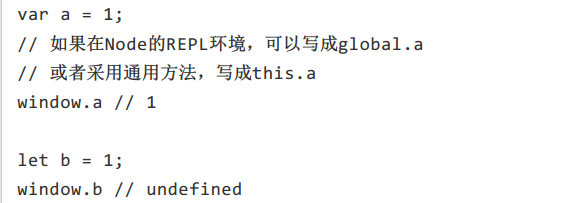


如果真的想将对象冻结，应该使用Object.freeze方法。

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。

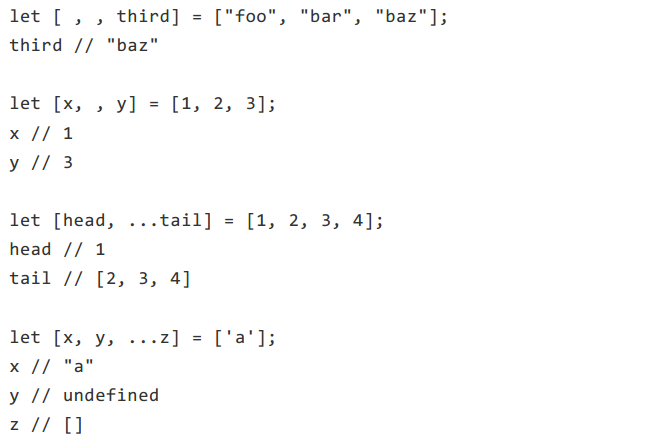


## 数组的解构赋值

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。

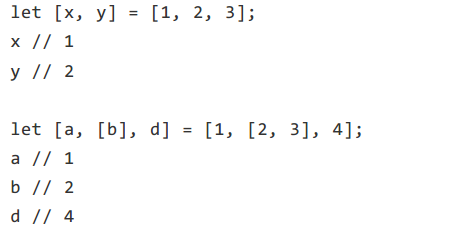
Var [a,b,c] = [1,2,3];即

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

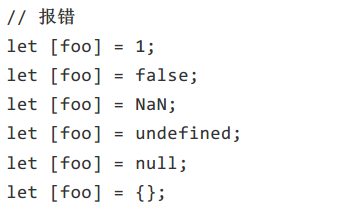


如果解构不成功，变量的值就等于undefined.

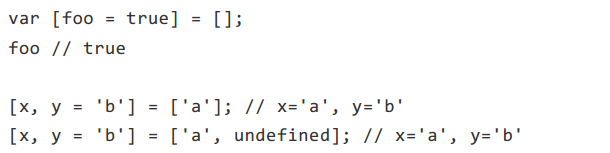
另一种情况是不完全解构，即等号左边的模式，只匹配一部分的等号右边的数组。这种情况下，解构依然可以成功。



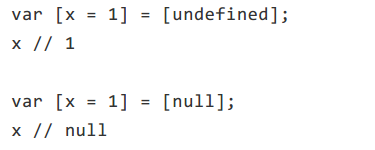
如果等号右边不是数组，那么将会报错。



解构允许指定默认值。

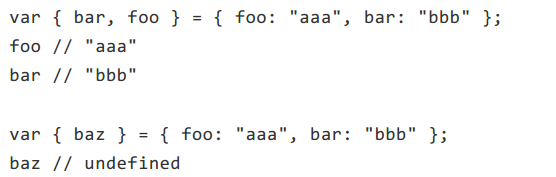


注意：ES6内部使用严格相等运算符（===）判断一个位置是否有值。所以，如果一个数组成员不严格等于undefined默认值是不会生效的。



## 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。

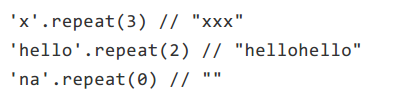
## 新增方法

### 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。



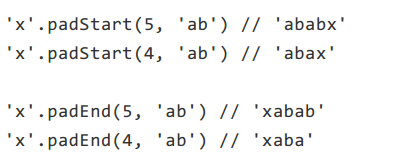
参数如果是小数将会被取整。

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()

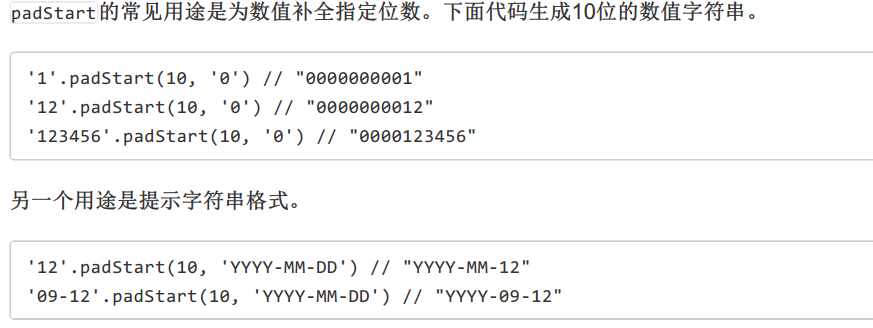
ES7推出了字符串补全长度的功能。如果某个字符串不够指定长度，会在头部或尾部补全。



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”

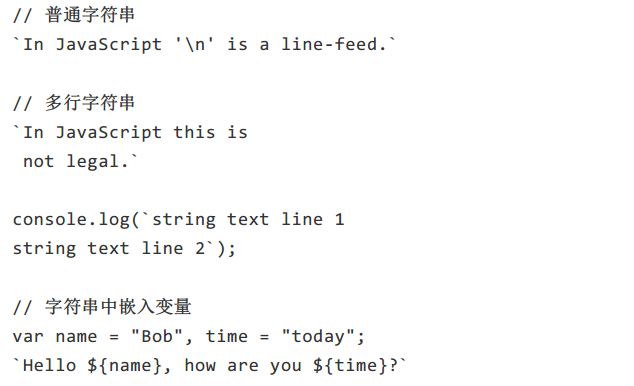
let fn = function (){

return ‘hello world’;

}

` ${ fn() } `

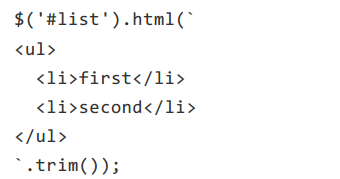
// “hello world”



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）。

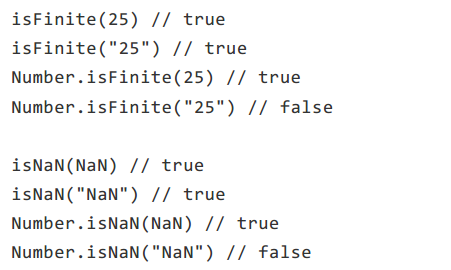
## 数值的扩展

### 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。

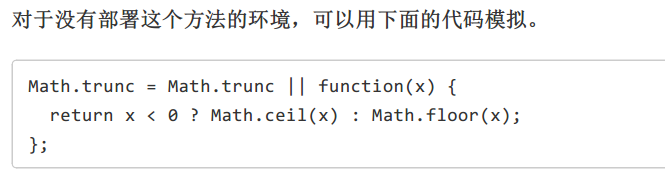
### Math对象的扩展

#### 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( )

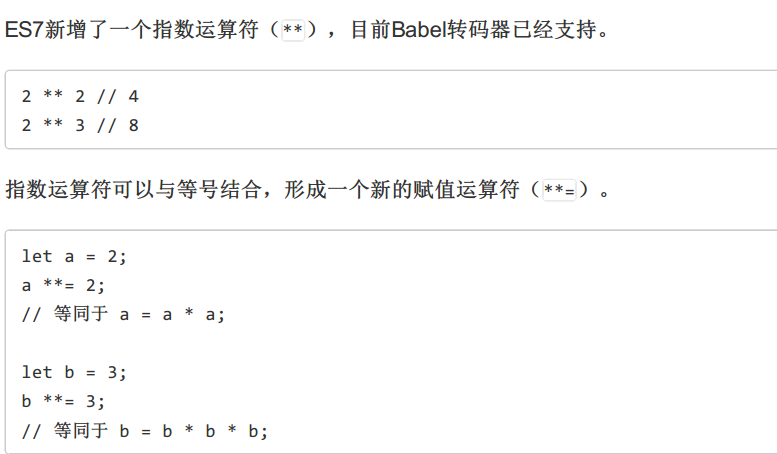
计算一个数的立方根。

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。

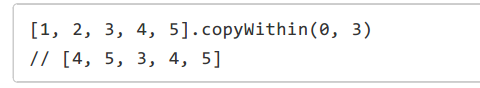
它接受三个参数：

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( )