目录

[location 2](#_Toc492495298)

[search 3](#_Toc492495299)

[host 3](#_Toc492495300)

[Date 4](#_Toc492495301)

[时间 4](#_Toc492495302)

[Date() 4](#_Toc492495303)

[getTime 4](#_Toc492495304)

[getTimezoneOffset 5](#_Toc492495305)

[年 5](#_Toc492495306)

[getFullYear 5](#_Toc492495307)

[月 5](#_Toc492495308)

[getMonth 5](#_Toc492495309)

[日 5](#_Toc492495310)

[getDay 5](#_Toc492495311)

[getDate 6](#_Toc492495312)

[天以下 6](#_Toc492495313)

[getHours 6](#_Toc492495314)

[getMinutes 6](#_Toc492495315)

[getSeconds 6](#_Toc492495316)

[getMillSeconds 6](#_Toc492495317)

[Date方法 6](#_Toc492495318)

[parse 6](#_Toc492495319)

[UTC() 6](#_Toc492495320)

[valueOf 6](#_Toc492495321)

[转换 7](#_Toc492495322)

[toDateString 7](#_Toc492495323)

[toString() 7](#_Toc492495324)

[toJSON 7](#_Toc492495325)

[toTimeString() 7](#_Toc492495326)

[Math 8](#_Toc492495327)

[random 8](#_Toc492495328)

[String 9](#_Toc492495329)

[获取 9](#_Toc492495330)

[charAt 9](#_Toc492495331)

[charCodeAt 9](#_Toc492495332)

[indexOf 9](#_Toc492495333)

[lastIndexOf 9](#_Toc492495334)

[substr 9](#_Toc492495335)

[substring 9](#_Toc492495336)

[操作 9](#_Toc492495337)

[concat 9](#_Toc492495338)

[clice 9](#_Toc492495339)

[String方法 9](#_Toc492495340)

[fromCharCode 10](#_Toc492495341)

[正则 10](#_Toc492495342)

[match 10](#_Toc492495343)

[replace 10](#_Toc492495344)

[search 10](#_Toc492495345)

[转换 10](#_Toc492495346)

[split 10](#_Toc492495347)

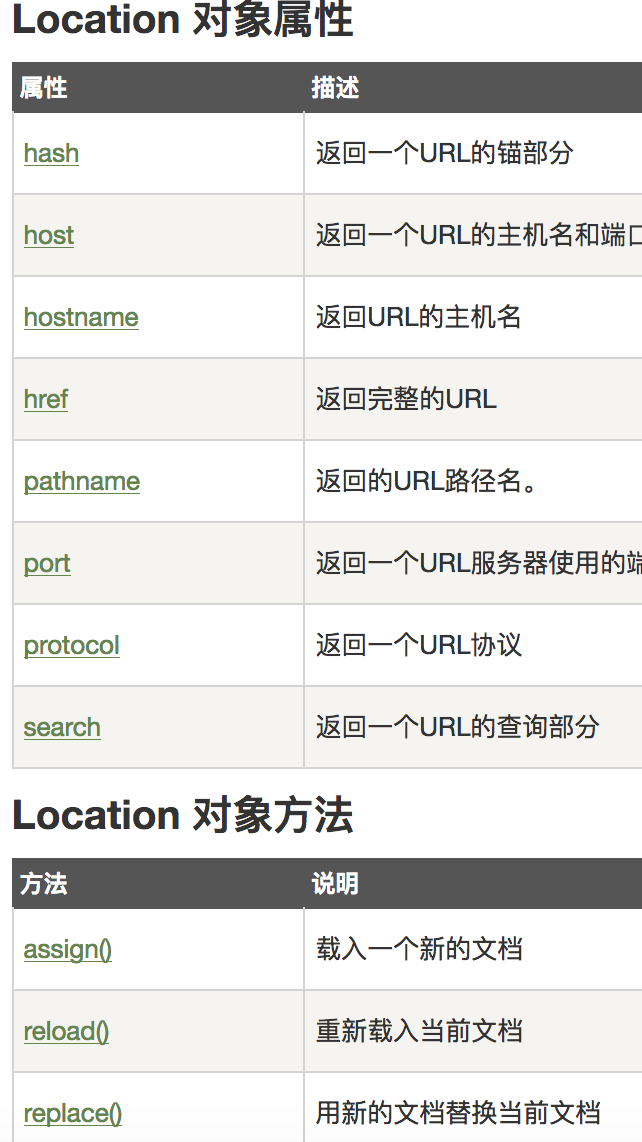
[toLowerCase 10](#_Toc492495348)

[toUpperCase 10](#_Toc492495349)

[trim 10](#_Toc492495350)

# location

属性也可以赋值

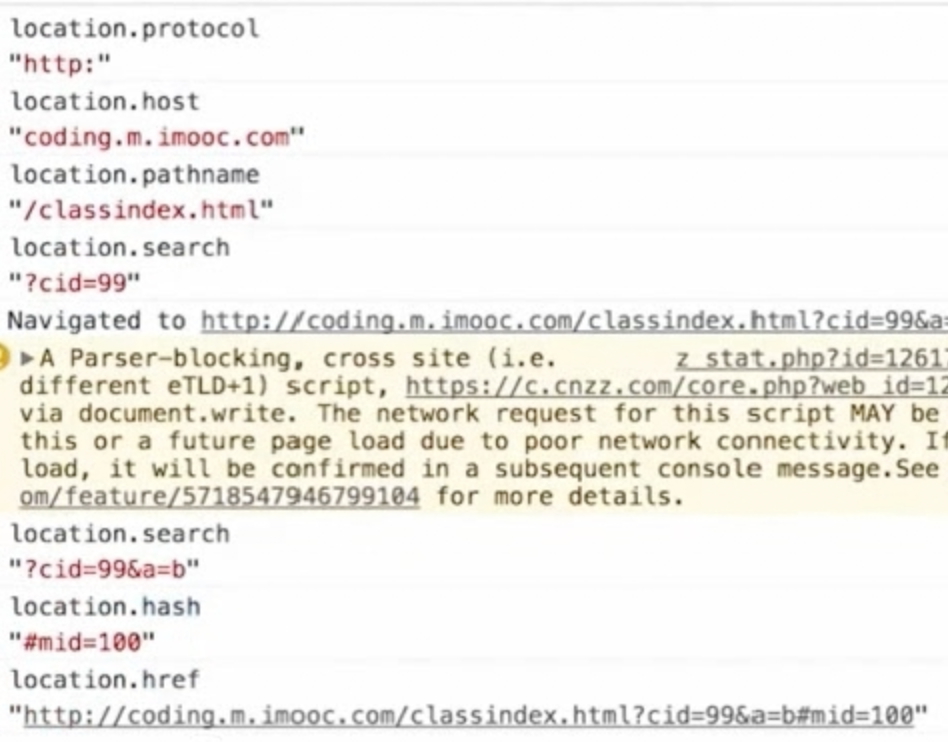


## search

包括?

## host

ip+端口，或者域名



# Date

UTC表示世界时，比中国晚8个小时。中国20点，世界时12点。

实例时间为2017.05.20

get都有对应的set。

## 操作

### 时间相减

得到相差毫秒值。

## 时间

### Date()

### getTime

返回 1970 年 1 月 1 日至今的毫秒数。

### getTimezoneOffset

-480

返回本地时间与格林威治标准时间 (GMT) 的分钟差。。

## 年

### getFullYear

四位数字

## 月

### getMonth

从0开始

## 日

### getDay

从周末开始，为0

### getDate

## 天以下

### getHours

### getMinutes

### getSeconds

### getMillSeconds

## Date方法

### parse

返回1970年1月1日午夜到指定日期（字符串）的毫秒数。

### UTC()

根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数。

### valueOf

根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数

var d = Date.parse("March 21, 2012");

## 转换

默认都是本地时间，加local格式有区别。

### toDateString

把 Date 对象的日期部分转换为字符串。

"Tue Sep 05 2017"

toLocaleDateString()

"2017/9/5"

### toString()

"Tue Sep 05 2017 20:54:39 GMT+0800 (CST)"

toUTCString

toLocaleString()

"2017/9/5 下午8:54:39"

toISOString

"2017-09-05T12:54:39.015Z"

### toJSON

世界时

"2017-09-05T12:54:39.015Z"

### toTimeString()

"20:54:39 GMT+0800 (CST)"

toLocaleTimeString()

"下午8:54:39"

# Math

## random

0到1之间的随机数，但位数是不确定的。

# String

## 获取

### charAt

### charCodeAt

### indexOf

### lastIndexOf

### substr/substring/slice

substr(起始位置，数量（省略代表到结尾）)

slice/substring(起始位置，结束位置(省略到结尾))

slice可以是负数，substring不可以

## 操作

### concat

## String方法

### fromCharCode

## 正则

### match

有g标志则返回一个数组，只有一个参数。没匹配到返回null。

不会有子表达式，区分exec。

### replace

str.replace(regexp|substr, newSubstr|function)

字符串

var newstr = str.replace(re, '$2, $1');

函数

function replacer(match, p1, p2, p3, offset, string) {

// p1 is nondigits, p2 digits, and p3 non-alphanumerics

return [p1, p2, p3].join(' - ');

}// 和正则对象一致。

可以用对象去匹配

function formatDate(t,str){

var obj = {

yyyy:t.getFullYear(),

yy:(""+ t.getFullYear()).slice(-2),

M:t.getMonth()+1,

MM:("0"+ (t.getMonth()+1)).slice(-2),

d:t.getDate(),

dd:("0" + t.getDate()).slice(-2),

H:t.getHours(),

HH:("0" + t.getHours()).slice(-2),

h:t.getHours() % 12,

hh:("0"+t.getHours() % 12).slice(-2),

m:t.getMinutes(),

mm:("0" + t.getMinutes()).slice(-2),

s:t.getSeconds(),

ss:("0" + t.getSeconds()).slice(-2),

w:['日', '一', '二', '三', '四', '五', '六'][t.getDay()]

};

return str.replace(/([a-z]+)/ig,function($1){return obj[$1]});

}

### search

查找与正则表达式相匹配的值。

匹配到返回起始匹配位置，没匹配到返回-1

## 转换

### split

string.split(separator正则,limit份数)份数可以省略，默认最大数量

### toLowerCase

### toUpperCase

### trim

# Array

## 遍历方法

返回boolean every some

无返回值 forEach

返回一个值 find findIndex（索引位置）（都是返回找到的第一个）reduce recudeRight

返回部分数组filter

返回整个数组map sort

### reducer/reducerRight

**reduce** reduceRight(function（total,currentValue,currentIndex,arr）,initalValue)(多了一个属性total)

|  |  |
| --- | --- |
| *total* | 必需。*初始值*, 或者计算结束后的返回值。 |
| *currentValue* | 必需。当前元素 |
| *currentIndex* | 可选。当前元素的索引 |
| *arr* | 可选。当前元素所属的数组对象。 |

### fill

引用类型将会填充同一个引用.arr[0] === arr[1] true

### sort

var scores = [1, 10, 21, 2];

scores.sort();

// [1, 10, 2, 21]

## 查

### indexOf/lastIndexOf

如果在数组中没找到字符串则返回 -1。

### slice

截取时不包括最后一个，slice(0,0)返回一个空数组！！！，String也是一样。

## 增

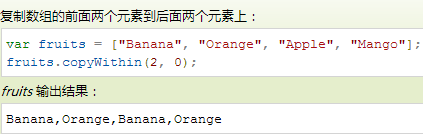
### concat

Array.prototype.concat(a.slice(0,1),a.slice(2))

### copyWithin

从数组的指定位置拷贝元素到数组的另一个指定位置中。返回这个数组（即原数组变了）

|  |  |
| --- | --- |
| *target* | 必需。复制到指定目标索引位置。 |
| *start* | 必需。元素复制的起始位置。 |
| *end* | 可选。停止复制的索引位置 (默认为 *array*.length) |



### push/unshift

## 删

### pop/shift

### splice

**splice**(index,howmany,item1,.....,itemX) (起始位置,数量（省略表示删到最后，0表示不删），添加的新元素（不是另一个数组）)

在原数组上操作。不改变原数组，考虑[...slice(0,index),...slice(index)]

## 改

## 转换

### toString

### join

### reverse

## 与String共同方法

indexOf lastIndexOf concat **slice**

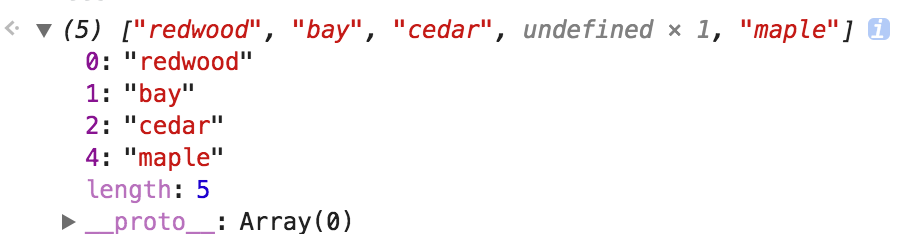
## 高级

### delete

将对应那一项变成undefined。

var trees = new Array("redwood", "bay", "cedar", "oak", "maple");

delete trees[3];





### [Object,Object]

将对象作为属性，这个属性会成为Object

# 正则

遍历有g标志会记住其中的位置，没有g会每次从头开始，陷入死循环。

## test

返回true,false

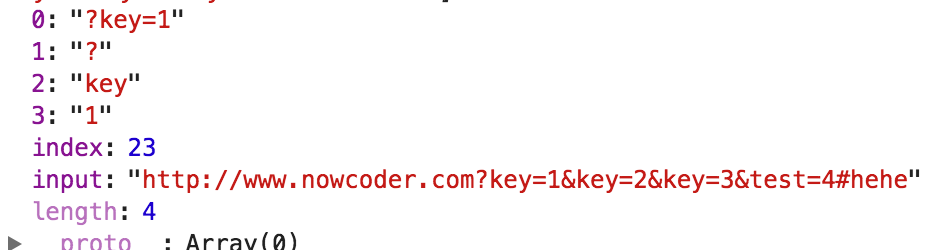
### 邮箱

/^(\w+)(\.\w+)\*@(\w+)(\.\w+)\*.(\w+)$/i.test(sEmail)

## exec

js

返回一个类数组。包括整个匹配到的和子串。



## compile

RegExpObject.compile(regexp,modifier(gim))

var patt=/man/g;  
patt=/(wo)?man/g;  
patt.compile(patt);

## 技巧

### 开头匹配与向前匹配结合

(?!^)： 只在开头才进行负向前匹配。

/(?!^)\-(\w)(\w+)/g ：

### ^

^前面不跟任何东西，表示只在开头时匹配。

/^a/.exec('aeae')

["a", index: 0, input: "aeae"]

# Number

## 转换

### toString

可选参数规定表示数字的基数，使 2 ~ 36 之间的整数。若省略该参数，则使用基数 10。但是要注意，如果该参数是 10 以外的其他值，则 ECMAScript 标准允许实现返回任意值。

* 2 - 数字以二进制值显示
* 8 - 数字以八进制值显示
* 16 - 数字以十六进制值显示

## 总结

/不是整除，整除需要全局函数parseInt转换。

# 全局函数

parseInt