# ECMAScript（ES）

ECMASCript是一种语言标准，javascript是对ES标准的一种实现。

最新版本的ECMAScript6标准（ES6）在2015年6月发布

# strict模式

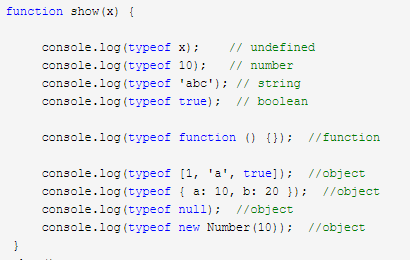


# NaN和Infinity

表示not a number，当无法计算结果时用NaN表示

Infinity表示无限大，数值超过了JavaScript的Number所能表示的最大值时，就表示为Infinity

# typeof



# instanceof和typeof区别

在js中，判断一个变量的类型常常会用typeof运算符，在使用typeof运算符时采用引用类型存储值会出现一个问题，无论引用的是什么类型的对象，他都返回object，这时就需要用到instance来检测某个对象是不是另一个对象的实例。

**instanceof 运算符用来测试一个对象在其原型链中是否存在一个构造函数的 prototype 属性。**

# ==和===（NaN比较）

==会自动转换数据类型再比较

===不会自动转换数据类型，如果数据类型不一致，返回false

false==0//true

false===0//false

由于JavaScript这个设计缺陷，不要使用==比较，始终坚持使用===比较

NaN这个特殊的Number与所有其他值都不相等，包括他自己

唯一能判断NaN的方法时通过isNaN()函数

isNaN(NaN)//true

# 浮点数比较

参考：

<https://blog.csdn.net/gaoshuang5678/article/details/50554131>

浮点数在运算过程中会产生误差，因为计算机无法精确表示无限循环小数，比较两个浮点数是否相等，只能计算他们 之差的绝对值，看是否小雨某个阈值

1/3===(1-2/3);//false

Math.abs(1/3-(1-2/3))<0.00000001//true

# null和undefined

null表示一个空值，undefined表示值未定义。大多数情况下，应该用null，undefined仅仅在判断函数参数是否传递的情况下有用。

# 数组

可包含任意数据类型

[1,2,”3”,true]或者new Array(1,2,3…)实现

# 对象

js对象由一组键-值组成的无序集合，示例：

var person = {

name: 'Bob',

age: 20,

tags: ['js', 'web', 'mobile'],

city: 'Beijing',

hasCar: true,

zipcode: null

};

# ES6新增-多行字符串、模板字符串

## 多行字符串

由于\n写起来比较费事，ES6新增标准表示多行字符串，使用反引号

反引号在数字键1的左边

## 模板字符串

示例：

var name=’小明’;

var age=20;

var message=`你好，${name},今年${age}岁`;//你好，小明，今年20岁

# 字符串操作

var s=’hello world’

长度获取：s.length

获取指定位置字符：

1. s[1],索引超出长度时返回undefined
2. 对s[index] 不会报错，也没有任何效果们不会改变s的值

toUppdeCase()

toLowerCase()

indexOf()

substring()

# Array操作

对Array的length复制会导致Array大小的变化，不足的补undefined

对Array超出索引的访问会导致Array大小的变化，不足的补undefined

1. indexOf() 搜索元素位置
2. slice() 截取Array部分元素，返回一个新的Array（不传递参数时，从头截取到尾，我们可以用此方式复制一个Array）
3. push和pop

向Array的末尾添加若干元素，pop把Array的最后一个元素删除掉,返回删除掉的元素,为空时调用返回undefined

arr.push(‘A’,’B’)

1. unshift和shift

unshift向Array头部添加若干元素，返回长度

shift把第一个元素删除

1. sort排序
2. reverse

把Array元素掉个个，反转1,2,3编程3,2,1

1. splice

修改Array的万能方法，可以从指定的索引开始删除若干元素，然后从该位置添加若干元素

1. concat连接Array

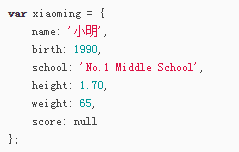
不会修改当前Array而是返回一个新Array

1. join用指定字符串连接每个元素并返回

var arr=[‘a’,’b’,’c’]

arr.join(‘-’) 返回a-b-c

# 对象操作

示例：

正常访问xiaoming.name

当属姓名包含特殊的字符时，需要用’’括起来，访问时也必须xiaoming[‘属性名’]访问

访问不存在的属性返回undefined

//添加删除属性

xiaoming.newpro=19;

delete xiaoming.newpro;

//检查是否拥有某一属性

‘name’ in xiaoming;//true

//判断属性时，属性也有可能是继承的

‘toString’ in xiaoming;//true,toString定义在object对象中，所有的对象最终都会在原型链上指向object，要判断属性是自身拥有而不是继承的可以用hasOwnProperty(‘属性名’)判断

# if(a) 是true或者false？

js把null、undefined、0、NaN和空字符串’’视为false，其他一概视为true

# for(::)表示无限循环，需要break来退出

# for…in

for(var key in o){}

他可以把一个对象所有属性依次循环出来，可以使用hasOwnProperty()判断是否是继承属性

也可以遍历数组

请注意，for ... in对Array的循环得到的是String而不是Number。

# Map和Set（ES6新增）

js {}对象键必须是字符串，实际Number或者其他数据类型作为键也是很合理的，英雌ES6为了解决这个问题引入新的数据类型Map

var m=new Map();

var s=new Set();

1. 初始化

var m=new Map([‘Mi’,95],[‘Bob’,75],[‘Tracy’,85])

或者初始化一个空的

var m=new Map();

## Map操作

var m=new Map();

m.set(‘A’,67) //添加新的key-value

m.has(‘A’)//判断是否存在keyA

m.get(‘A’) //获取A key对应的值

m.delete(‘A’)//获取A key

m.get(‘A’)//不存在时返回undefined

key是唯一的，对一个key重复放入数据，后面的值会把前面的冲掉

## Set操作

set是一组key的集合，不存在value，key不重复

var s1=new Set();

var s2=new Set([1,2,3]); //key重复时被自动过滤

s.add(4);

s.delete(3)

# iterable类型（ES6新增）

Array可以采用下标循环，Map和Set遍历时无法使用下标，因此为了统一集合类型，ES6标准引入iterable类型，Array、Map和Set都属于iterable类型。

for…of循环（ES6引入的新语法）

var a=[1,2,3];

for(var x of a){}

遍历Map时，x 值为 key=value

## for in和for of区别

for in遍历对象的key？？

Array实际是一个对象，每个元素的索引被视为一个属性,初始化后手动添加属性后，in遍历会将添加的属性也遍历到

var a=[‘A’,’B’,’C’]

1. name=’Hello’

for(var x in a){

//0,1,2,name

}

for of

var a=[‘A’,’B’,’C’];

1. name=”Hello”

for(var x of a){

//A,B,C

}

## forEach遍历(ES5.1标准引入)

var a=[‘A’,’B’,’C’]

a.forEach(function (element, index, array) {

// element: 指向当前元素的值

// index: 指向当前索引

// array: 指向Array对象本身

console.log(element + ', index = ' + index);

});

A, index = 0  
B, index = 1  
C, index = 2

Set与Array类似，但Set没有索引，因此回调函数的前两个参数都是元素本身

Map的回调函数参数依次为value、key和map本身

JavaScript的函数调用不要求参数必须一致，回调可以省略不感兴趣的参数

# 函数

不renturn的话函数返回undefined

调用时可以传入更多或者更少的参数

## 函数也是一个对象

可以这么定义：

var abs=function(x)

{

….

};

## arguments关键字

只能在函数内部起作用的关键字

function foo(x) {

console.log('x = ' + x); // 10

for (var i=0; i<arguments.length; i++) {

console.log('arg ' + i + ' = ' + arguments[i]); // 10, 20, 30

}

}

foo(10, 20, 30);

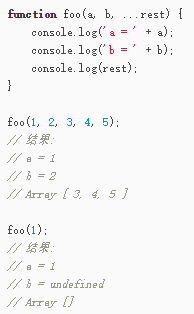
输出

x = 10  
arg 0 = 10  
arg 1 = 20  
arg 2 = 30

## rest参数（ES6标准）-获取已定义参数之外的参数

由于JavaScript函数允许接收任意个参数，于是我们就不得不用arguments来获取所有参数：因此es6引入rest参数

rest参数只能写在最后，前面用…标识



## return语句-分行时

js引擎有一个在行末自动添加分号的机制，注意下面的情况

return{name:’foo’}; //正常

return

{name:’foo’}; //会变成return;{name:’foo’} 相当于返回undefined

正确的多行写法是：

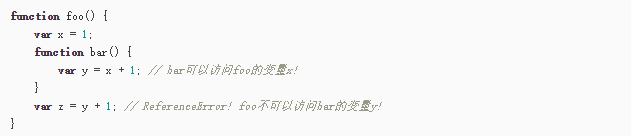
return {

name:’foo’

}

## 函数可以嵌套

内部函数可以访问外部函数定义的变量

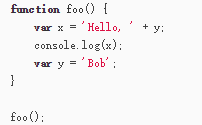


## 变量提升

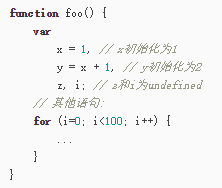
js会扫描整个函数体的语句，把所有申明的变量提升到函数顶部

但是这种提升只会提升申明，不会提升赋值

因此下方不会报错，会输出Hello undefined



由于JavaScript的这一怪异的“特性”，我们在函数内部定义变量时，请严格遵守“在函数内部首先申明所有变量”这一规则。最常见的做法是用一个var申明函数内部用到的所有变量：



## 全局作用域

js默认有一个全局对象window，全局作用域的变量实际上被绑定到window的一个属性



alert函数实际也是window的一个变量

## 防止全局冲突：

把自己的所有变量和函数全部绑定到一个全局变量中，示例：

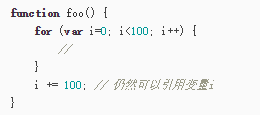
var MYAPP={};

MYAPP.name=’wj’

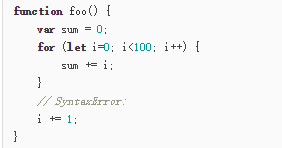
……

## 局部作用域-let申明（ES6新增）

如下示例，i的作用域不仅限于for循环内



ES6标准引入let关键字，可以申明块级作用域的变量



## 申明常量-const（ES6新增）

使用const来定义常量（不可修改）；const和let都具有块级作用域

const PI=3.14

## 解析赋值-对一组变量赋值（ES6新增）

var [x, y, z] = ['hello', 'JavaScript', 'ES6'];

x，y，z被分别赋值

对数组赋值时，多个变量要用[…]括起来，数组有嵌套也可以赋值（嵌套位置要一致）

**let** [x, [y, z]] = ['hello', ['JavaScript', 'ES6']];

还可以对对象进行解构赋值

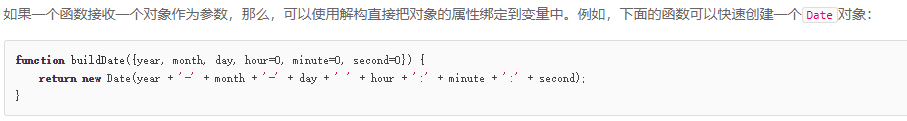
参考：

<https://www.liaoxuefeng.com/wiki/001434446689867b27157e896e74d51a89c25cc8b43bdb3000/0014344993159773a464f34e1724700a6d5dd9e235ceb7c000>

使用场景

交换变量：

获取页面的域名和路径：



# 方法

绑定到对象上的函数称为方法，和普通函数没啥区别，在内部可以使用this关键字，始终指向当前对象，可以访问对象中的属性。

## 在对象中绑定函数

var xiaoming={

name:’小明’,

age:function(){….}

}

## apply（call）函数，功能类似

### 指定函数中this关键字

函数中this，在一个独立函数中（非对象中）有时是window或者undefined，当需要控制this的指向时，可以用函数本身的apply方法：

参数1：需要绑定的this变量

参数2：是一个Array，函数本身的参数

call函数与apply类似，call是将参数按顺序传入



### 装饰器（可以修改系统函数-感觉是重点）

apply可以动态改变函数的行为

示例：需要统计调用了多少次parseInt()

可以更换系统函数，实现如下

var count=0;

var oldParseInt=parseInt;

window.parseInt=function()

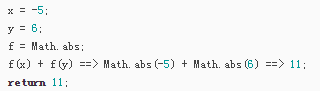
{

count+1;

return oldParseInt.apply(null,arguments);//调用原函数

}

# 高阶函数（函数可作为另一函数的参数传入）



## map（定义在Array中）

map中的函数实际传入了三个参数：

currentValue，index，array

可以对Array中元素执行同一函数并返回

示例：

var arr=[1,2,3,4,5]

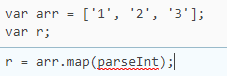
//想得到所有元素平方的一个数组

function pow(x){return x\*x}

var results=arr.map(pow);

//results即为[1,4,9,16,25]

疑问：？？？？？



返回



parseInt接收了map传入的三个参数，获取了前两个作为自己的参数，即value和index

所有此处返回的值为parseInt(‘1’,0)//1;parseInt(‘2’,1)//NaN;parseInt(‘3’,2)//NaN

因为parseInt实际调用时应该有两个参数，第一个为被解析的字符串，第二个为要解析的数字的基数，介于2-36之间。省略时，以10为基础来解析，小于2或者大于36时，返回NaN

当参数 *radix* 的值为 0，或没有设置该参数时，parseInt() 会根据 *string* 来判断数字的基数。

## reduce(定义在Array中)

reduce中的函数实际传入了四个参数：

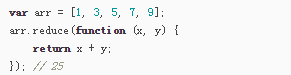
Accumulator，currentValue，index，array

Accumulator为上次函数计算结果，第一次时为arr[0]

reduce把一个函数作用在这个Array,这个函数有两个参数，先调用数组前两个元素作为参数，结果值加上第三个元素作为下一组参数，以此类推调用下去

[x1, x2, x3, x4].reduce(f) = f(f(f(x1, x2), x3), x4)

示例：求和



## filter（定义在Array中）

三个参数

element，index，selfArr

可以将某些元素过滤掉，返回剩下的函数

接收一个函数，将传入的函数依次作用于每个元素，然后根据函数返回是true还是false决定保留还是丢弃钙元素

## sort（定义在Array中）

正常排序时，sort方法会把所有元素转为String之后再进行排序，会出错

sort也是一个高阶函数，可以接收一个比较函数来进行自定义的排序

x<y返回-1 x=y返回0 x>y返回1

# 闭包

高阶函数除了可以接收函数作为参数外，还可以把函数作为结果值返回。

参考：

<https://www.liaoxuefeng.com/wiki/001434446689867b27157e896e74d51a89c25cc8b43bdb3000/00143449934543461c9d5dfeeb848f5b72bd012e1113d15000>

<http://www.cnblogs.com/wangfupeng1988/p/3977924.html>

闭包就是携带状态的函数，并且它的状态可以完全对外隐藏起来。

# 箭头函数（ES6新增）

x=>{return x\*x}

(x,y)=>{return x\*y;}

返回对象时

x=>{foo:x}//error

应该将对象括起来

x=>({foo:x})

箭头内部函数的this是词法作用域，由上下文确定



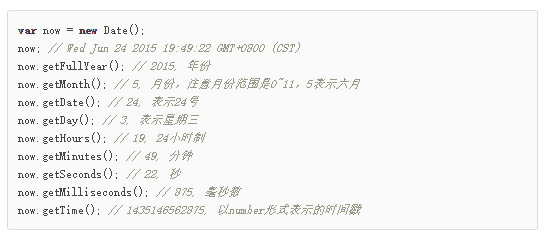
# generator（生成器 ES6新增）

generator有function\*定义，除了return语句，还可以yield返回多次。

# 标准对象

## Date

表示日期和时间



注意：

时间是从浏览器本机获取的

js的月份是从0-11表示的

### 创建指定日期

var d=new Date(2015,1,22,20,15,30,123); //返回date对象

var d=Date.parse(“201`5-06-24T19:49:22 875+8:00”)//返回一个时间戳，是一个长整形

var d=new Date(时间戳)//时间戳转Date对象

### 时区

var d=new Date(时间戳)

d.toLocalString()//本地时间

d.toUTCString //UTC时间，与本地时间相差8小时

### 时间戳

时间戳表示1970年1月1号零时整的GMT时区开始的那一刻到现在的毫秒数。

获取时间戳

new Date().getTime()

## RegExp

new RegExp(*pattern*, *attributes*);

参数 *pattern* 是一个字符串，指定了正则表达式的模式或其他正则表达式。

参数 *attributes* 是一个可选的字符串，包含属性 "g"、"i" 和 "m"，分别用于指定全局匹配、区分大小写的匹配和多行匹配。ECMAScript 标准化之前，不支持 m 属性。如果 *pattern* 是正则表达式，而不是字符串，则必须省略该参数。

正则表达式

正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言来给字符串定义一个规则，凡是符合规则的字符串，我们就认为它“匹配”了，否则，该字符串就是不合法的。

1. \d匹配一个数字
2. \w匹配一个字母或数字
3. .可以匹配任意字符
4. \*表示任意个字符（包括0个）
5. +表示至少一个字符
6. ？表示0个或1个字符
7. {n}表示n个字符，{n,m}表示n-m个字符 示例：\d{3}表示3个数字
8. \s可以匹配一个空格（包括Tab等空白符）,\s+表示至少有一个空格
9. ^表示行的开头 ^\d表示必须以数字开头
10. $表示行的结束 \d$表示必须以数字结束

要做更精确的匹配，可以用[]表示范围：

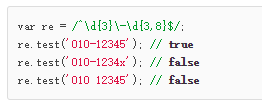


### 创建正则表达式

var rel=/ABC\-001/ ；//直接写出正则表达式，通过/正则表达式/

var rel2=new RegExp(‘ABC\\-001’);//使用new RegExp创建

RegExp对象的test方法用域测试给定的字符串是否符合条件。



### 通过正则表达式切分字符串

正常切分，会把空格也算一个字符



可以通过split传入正则表达式筛除空格



可以分割多个空格



加入其它字符;



### 分组

正则表达式可以提取子串，用()表示的就是表示要提取的分组



返回一个Arr，第一个为原始字符串，之后的为要提取的分组

失败时返回null

### 贪婪匹配

默认贪婪匹配，尽可能匹配更多的字符

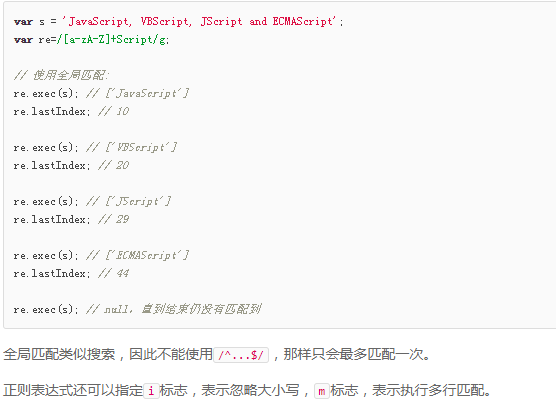


\d+将所有数字都匹配了，可以加个?让\d+采用非贪婪匹配



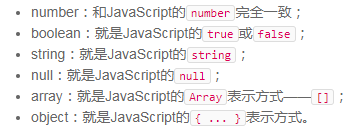
### 全局搜索（（查找所有匹配而非在找到第一个匹配后停止））

全局匹配可以多次执行exec()方法来搜索一个匹配的字符串。当我们指定g标志后，每次运行exec()，正则表达式本身会更新lastIndex属性，表示上次匹配到的最后索引：



## JSON

json中数据类型



### 序列化（转字符串）

json对象转字符串

JSON.stringify(jsonobj)

JSON,stringift(jsonobj,null,’ ’)

第二个参数用于控制如何筛选对象的键值

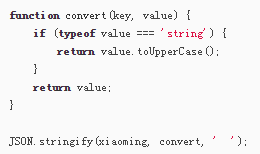
示例：

只输出指定键值（name和skills）

JSON.stringify(jsonobj,[‘name’,’skills’,’ ’])

示例2：

对键对应属性值进行处理，第二个参数传入函数



可以给json对象一个toJson方法，该方法返回的值即为stringify序列化之后的值

### 反序列化（字符串转json对象）

JSON.parse(‘’)

JSON.parse(‘’,func)//第二个参数可以传入一个函数，用来解析转换后的属性

function (key, value) {

if (key === 'name') {

return value + '同学';

}

return value;

}

## 面向对象编程

# 浏览器

## 浏览器分类：

1. IE6-11 从IE10开始支持ES6标准
2. Chrome 基于webkit内核浏览器，内置v8js引擎
3. Safari Apple Mac系统知道基于webkit内核浏览器
4. Firefox Mozila颜值Gecko内核和js引擎OdinMonkey
5. 移动设备ios和Android分别是safari和chrome

## 浏览器对象

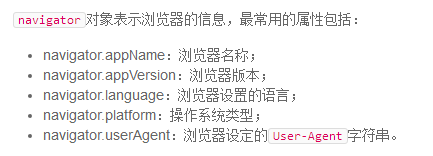
### window

window对象全局作用域，而且表示浏览器窗口

innerWidth和innerHeight 获取浏览器窗口内部宽度和高度（随窗口大小调整而改变）

outerWidth和outerHeight 获取浏览器窗口整个大小（包括工具栏等等）

### navigator



### screen

表示屏幕信息

常用属性：

宽高以像素为单位

screen.width

screen.height

screen.colorDepth: 返回颜色位数，如8、16、24

### location（URL信息）



location.assign(‘url’)//加载一个新页面

location.reload()//重新加载当前页

### document

document对象表示当前页面，document就是整个DOM树的根节点

document.title可以动态改变

document.cookie属性，获取当前页面的Cookie

由于引入的第三方js可能存在恶意代码读取网站用户的信息，服务器可以在设置Cookie时使用httpOnly，设定了httpOnly的Cookie将不能被JavaScript读取（这个行为由浏览器实现），为了确保安全，服务器端在设置Cookie时，应尽量使用httpOnly

### history

保存浏览器历史记录

history.back()或history.forward() 后退或者前进(尽量不要使用)

# DOM操作

var test=document.getElementById(‘’);

//获取所有直属子节点

var cs=test.children;

//获取父节点

self.parentElement

//获取第一个 最后一个

test.firstElementChild

test.lastElementChild

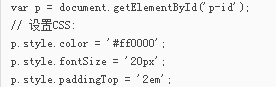
严格地讲，我们这里的DOM节点是指Element，但是DOM节点实际上是Node，在HTML中，Node包括Element、Comment、CDATA\_SECTION等很多种，以及根节点Document类型，但是，绝大多数时候我们只关心Element，也就是实际控制页面结构的Node，其他类型的Node忽略即可。根节点Document已经自动绑定为全局变量document。

## 更新DOM节点

修改innerHTML（注意对字符编码避免XSS攻击）

修改innerText或textContent（嵌入的script节本不会被作为脚本执行）

修改css



## 插入DOM

节点为空时，可以直接设置innerHTML插入节点

appendChild()添加到父节点的最后一个节点

createElement创建DOM节点

insertBefore将子节点插入到指定的位置

parentElement.insertBefore(newElement, referenceElement);

没有现成的insertAfter方法

### 可以创建style节点



可在网站控制台执行，调整网站样式

## 删除DOM

parent.removeChild(self)

注意到删除后的节点虽然不在文档树中了，但其实它还在内存中，可以随时再次被添加到别的位置。

当你遍历一个父节点的子节点并进行删除操作时，要注意，children属性是一个只读属性，并且它在子节点变化时会实时更新。（remove之后，children 长度会减少）

## 表单操作

可以直接调用value获取用户输入的值

value应用于text、password、hidden、select（对于单选框和复选框，value永远返回的HTML预设的值，应该用checked判断，为true或者false）

HTML5新增了大量标准控件，常用的包括date、datetime、datetime-local、color等，它们都使用<input>标签：

form本身的onsubmit事件，可在form提交时做修改或者验证（返回true或者false决定是否提交）

没有name属性的input数据不会被提交

## 文件

可以上传文件的唯一控件就是<input type="file">

当包含file时，表单的enctype必须指定为multipart/form-data,method必须为post，浏览器才能正确编码并以multipart/form-data格式发送表单的数据

出于安全考虑，浏览器只允许用户点击file来选择本地文件，用js设置值获取触发click事件是没有任何效果的，用户上传某个文件后，js也无法获取该文件的真实路径

（可以用<label for=’fileid’>）点击该label时，触发选择文件点击事件，是合法有效的

file.value=” ”//非真实路径

var file=fileInput.files[0]



### HTML5 新增File API允许js读取文件内容

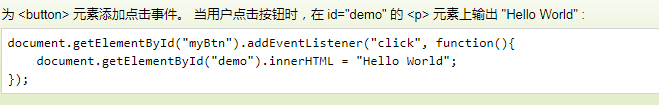
查资料

## addEventListener() （IE8以及更早不支持）



element.addEventListener(event, function, useCapture)

dom添加事件监听

**提示：** 使用 [removeEventListener()](http://www.runoob.com/jsref/met-element-removeeventlistener.html) 方法来移除 addEventListener() 方法添加的事件句柄。

# AJAX

var request=new XMLHttpRequest()

对于低版本的IE，使用ActiveObject请求

默认情况下，JavaScript在发送AJAX请求时，URL的域名必须和当前页面完全一致。

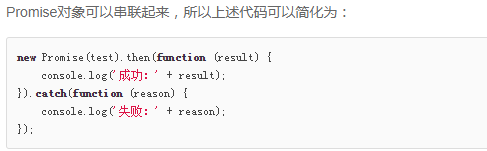
完全一致的意思是，域名要相同（www.example.com和example.com不同），协议要相同（http和https不同），端口号要相同（默认是:80端口，它和:8080就不同）。有的浏览器口子松一点，允许端口不同，大多数浏览器都会严格遵守这个限制。

1. 可以用flash请求外域的URL
2. 在同源域名下架设一个代理服务器来转发，js负责把请求发到这个代理服务器，有代理服务器转发
3. JSONP，只能用GET请求（利用了浏览器用于跨域引用js资源）
4. HTML5 CORS（必须要对象返回正确的响应才可正确访问）

# Promise对象（ES6新增）

承诺将来会执行的对象在JavaScript中称为Promise对象





# Canvas组件

绘制图形

绘制文本（带阴影）

HTML5新增，绘制图表、动画

**var** ctx = canvas.getContext('2d');

拿到一个CanvasRenderingContext2D对象，所有绘图都需要通过这个对象

需要绘制3D时

# JQuery

$.fn.jquery //jquery版本

jQuery.noConflict()方法可解决$与其他库定义的$冲突问题

## $符号

是jQuery的简写

$===jQuery //true

$本质上就是一个函数

## 选择器

选择器获取的是jQuery对象

如果拿到一个DOM对象，可以调用$(obj)把他变成jQuery对象

返回多个时：



### 基本选择器

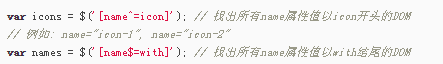
$(‘#id’)

$(‘p’)

$(‘.classname.classname2’)//两个之间没有空格

$(‘[name=email]’); $(‘[type=password]’)//按属性查找

//按属性查找还可以使用前缀或后缀查找



### 组合查找

//input中name=email的

$(‘input[name=email]’)

//tr中class=red的

$(‘tr.red’)

### 多项选择器

就是把多个选择器用,组合起来一起选

$(‘p,div’)

$(‘p.red,p.green’)

### 层级选择器

先定位父节点 在定位子节点，中间用空格分开



可以多层选择器

$(‘form.test p input’)

### 子选择器

与层级选择器类似，但是限定了必须是直属子节点

$(‘ul.lang>li.lang-javascript’)

### 过滤器

过滤器一般不单独使用，通常附加在选择器上，能更精确的定位元素



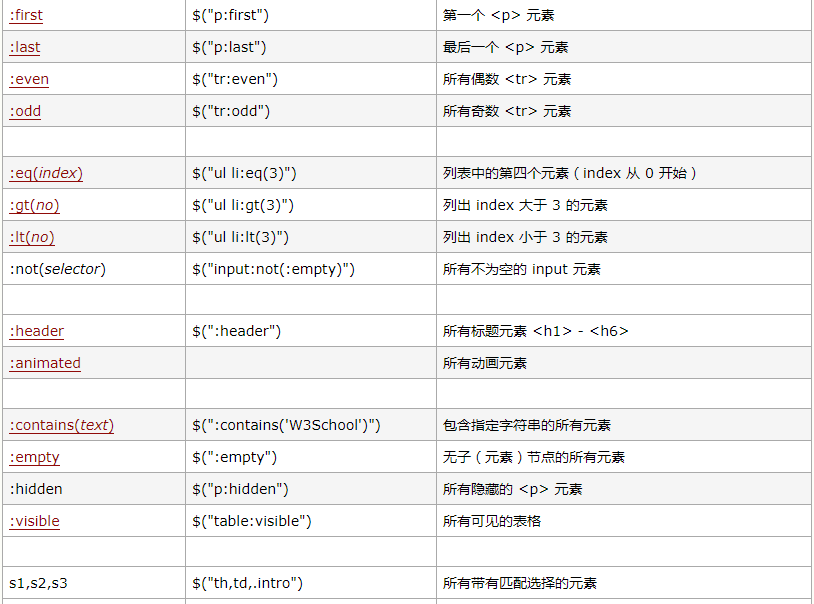
表单元素过滤器：

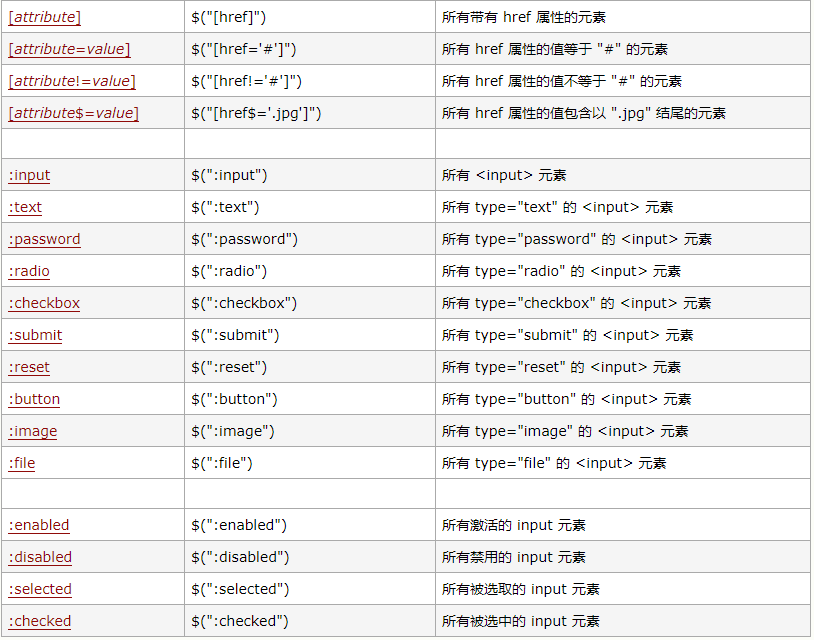


### 其他选择器

$(‘div:visible’)

$(‘div:hidden’)





### 查找和过滤

选择器查到元素后，可以在用find()或者parent()继续向下或向上查找

参数也是一个选择器

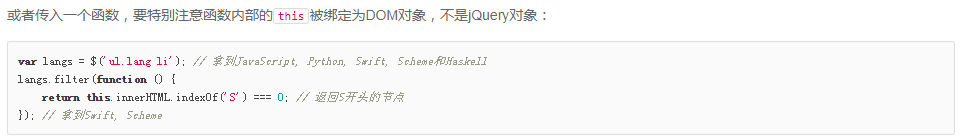
对于同一层级的节点，可以调用next()和prev()方法

过滤

filter() 传入选择器条件



或者传入一个函数



map()





## 操作DOM

1. text()和html()

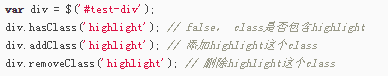
无参数调用text()是获取文本，传入参数就变成设置文本，html()也是类似操作

1. css() //直接作用于dom的style属性，具有最高优先级

$('#test-css li.dy>span').css('background-color', '#ffd351').css('color', 'red');



1. 修改class



1. 显示隐藏hide() show()
2. 获取或设置宽高 width() height() 加参数说明设置
3. attr()和removeAttr()

attr(‘attrname’,’attrvalue’)//没有第二个参数时说明是获取属性值

prop方法和attr类似，对于属性没有值的处理不同



1. 表单val()获取或设置表单元素的值

## 修改DOM

### 插入

如果要插入的节点已在DOM中，会先移除，再插入到新位置

append()添加到最后

prepend()添加到最前面

同级 before() after()

1. 传入html字符串
2. 传入原始DOM元素
3. 传入jquery对象
4. 传入函数



### 删除

获取到节点之后

$(‘’).remove()

## 事件

### 绑定

on()方法用来绑定一个事件

var a=$(“#d”);

a.on(‘click’,function(){

});

或者

a.click(function(){

})

### document.ready

$(function(){})和$(document).ready(function())

效果一样

可以定义多次，会依次执行

### 取消绑定

off



off('click')一次性移除已绑定的click事件的所有处理函数。

## 动画

show/hide

slideUp/slideDown

fadeIn/fadeOut

show(3000) show(‘fast’)

hide(3000) hide(‘fast’)

自定义动画

obj.animate({

opacity:0.25,

width:’256px’,

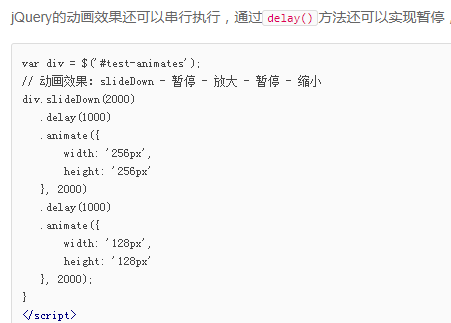
height:’256px’

},3000,function(){

//动画执行完成事件

})

动画可以串行执行



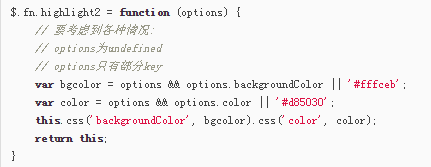
## 扩展jQuery

### 编写jQuery插件

扩展highlight1，所有jquery对象可以调用



带参数的



extend

另一种方法是使用jQuery提供的辅助方法$.extend(target, obj1, obj2, ...)，它把多个object对象的属性合并到第一个target对象中，遇到同名属性，总是使用靠后的对象的值，也就是越往后优先级越高：

# 错误处理

try ... catch ... finally

错误类型

JavaScript有一个标准的Error对象表示错误，还有从Error派生的TypeError、ReferenceError等错误对象。我们在处理错误时，可以通过catch(e)捕获的变量e访问错误对象

抛出错误

程序也可以主动抛出一个错误，让执行流程直接跳转到catch块。抛出错误使用throw语句。



如果在一个函数内部发生了错误，它自身没有捕获，错误就会被抛到外层调用函数，如果外层函数也没有捕获，该错误会一直沿着函数调用链向上抛出，直到被JavaScript引擎捕获，代码终止执行。

异步代码中无法捕获异常

涉及到异步代码，无法在调用时捕获，原因就是在捕获的当时，回调函数并未执行