**JavaScript**

**1.HTML**定义了网页的内容

**2.CSS**描述了网页的布局

**3.JavaScript**网页的行为

**用法:**

HTML 中的(JavaScript)脚本语句必须位于 <script> 与 </script> 标签之间。

内部：<body>文本中的 JavaScript 语句会在页面加载时执行。浏览器将直接执行。

通常把代码放入 <head> 部分中，就可以在事件发生时调用该函数。

外部： 外部 JavaScript 文件的文件扩展名是 .js

如需使用外部文件：<body> < script src="myScript.js" > </script> </body>

**输出：**

**window.alert( )：** 弹出警告框。

**document.write( )** ：将内容直接写到 HTML 文档中。

例：document.write(Date());

如果在文档已完成加载后执行 document.write，整个 HTML 页面将被覆盖。

**innerHTML：** 写入到 HTML 元素。

例：用过ID更改元素内容 document.getElementById("demo").innerHTML="段落已修改。"

**document.getElementById("demo")** 是用 id 属性来查找 HTML 元素

**console.log( )** ：控制台查看变量内容（F12开启调试模式，Console为调试窗口）。

例：console.log(5); 窗口将显示输出 5

**JavaScript 变量:** 使用关键字 **var** 来定义变量，如果没有通过var申明，就自动被申明为全局变量

例： var length; length = 6;

# **JavaScript 数据类型：** JavaScript 拥有动态类型

字符串： var carname="Volvo XC60";

数字： var x1=34.00;

NaN表示不是一个数字 例：0 / 0; Infinity表示无限大 例：2 / 0;

Boolean： var x=true;

数组： var cars=new Array(); cars[0]="Saab";

或： var cars=new Array("Saab","Volvo","BMW");

或： var cars=["Saab","Volvo","BMW"];

对象： 对象由花括号分隔。在括号内部，对象的属性以名称和值对的形式 (name : value) 来定义。

例：var person={firstname:"John", lastname:"Doe", id:5566};

对象寻址：person.lastname; 或 person["lastname"]; 结果都为Doe

清空变量：cars=null;

声明变量类型：使用关键词 "new"  例：var carname=new String;

# **JavaScript 对象：** 对象也是一个变量，但对象可以包含多个键值组（多个变量）。

JavaScript 中的所有事物都是对象：String、数值、Array、Date、函数...

此外，JavaScript 允许自定义对象。

对象拥有**属性**(键称为对象的属性名)和**方法**

例：var person={firstname:"John", lastname:"Doe", id:5566};

访问对象属性：person.lastname; 或 person["lastname"];

访问不存在的属性不报错，而是返回undefined

访问对象方法：person.fullName(); （不加括号返回函数表达式）

## 创建 JavaScript 对象

## 直接创建实例

var person=new Object();  
 person.firstname="John";  
 person.lastname="Doe";

或：person={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"};

## 使用对象构造器

定义一个构造器: function person(firstname,lastname){

this.firstname=firstname;

this.lastname=lastname; }

创建对象实例：用关键字new来调用这个函数，并返回一个对象

myFather=new person("John","Doe");

myMother=new person("Tom","Sunny");

写了new，它就变成了一个构造函数，它绑定的this指向新创建的对象，并默认返回this

## 向构造器添加属性和方法：object.prototype.name=value；

## 例：person.prototype.salary=320;

## 

**检测对象是否拥有某一属性：**

1. 用in操作符 ‘firstName’ in person // 返回true

如果in判断一个属性存在，这个属性不一定是person的，它可能是它继承得到的

1. 用hasOwnProperty( )方法可以判断一个属性是否是对象自身拥有的

person.hasOwnProperty(‘name’) //true

**包装对象**：类似JAVA里边的int和Integer的关系

例：var n = new Number(123); // 123,生成了新的包装类型

var s = new String('str'); // 'str',生成了新的包装类型

他们的类型已经变为**object**了

typeof new Number(123); // 'object'

new Number(123) === 123; // false

**注意**：如果不使用new，例Number（）函数会将任何数据转换为其类型。

var n = Number('123'); // 123，相当于parseInt()或parseFloat()

typeof n; // 'number'

# **Number 对象**

在JavaScript中，数字不分为整数类型和浮点型类型，所有的数字都是浮点型类型。

JavaScript 数字为十进制显示，可以使用 toString( ) 方法 输出16进制、8进制、2进制。

var myNumber=128;

myNumber.toString( );   // 返回 128  
myNumber.toString(16);   // 返回 80  
**注意**：直接转换要加两个点或加括号 123..toString() 或 (123).toString()

## Infinity - 无穷大： 当数字运算结果超过了JavaScript所能表示的数字上限（溢出），结 果为一个特殊的无穷大（infinity）值，在JavaScript中以Infinity表示。

## 例：var x = 2/0; // x = infinity

## NaN - 非数字值：可以使用 isNaN( ) 全局函数来判断一个值是否是 NaN 值。

## 例：isNaN(x); // 返回 true

## isNaN(y); // 返回 false

isNaN(Infinity) ； //返回 false

## 数字可以是数字或者对象：

## var x = 123; typeof(x) // 返回 Number var y = new Number(123); typeof(y) // 返回 Object (x === y) // 为 false，因为 x 是一个数字，y 是一个对象

# **String 对象**

## 使用位置（索引）可以访问字符串中任何的字符：

## var carname="Volvo XC60"; var character=carname[7];

## 可以在字符串中使用转义字符使用引号：

var answer='It's alright';

使用长度属性**length**来计算字符串的长度： txt.length

## 连接字符串：

## 1可以用+号连接

2．使用模板字符串，替换字符串内部的变量

var name = '小明';

var age = 20;

alert(`你好, ${name}, 你今年${age}岁了!`); （`）反引号在键盘的ESC下方

## 在字符串中查找字符串：indexOf( ) 来定位字符串中某一个指定的字符首次出现的位置

var n=str.indexOf("welcome"); 如果没找到对应的字符函数返回-1

lastIndexOf( ) 方法在字符串末尾开始查找字符串出现的位置。

## 内容匹配： ****match( )****函数用来查找字符串中特定的字符，并且如果找到的话，则返回这个字符。

例：var str="Hello world!";  
 document.write(str.match("world")); //返回world

**括号内也可放置正则表达式，返回匹配数组。**

## 替换内容： ****replace( )**** 方法在字符串中用某些字符替换另一些字符。

例： var str="Please visit Microsoft!"  
 var n=str.replace("Microsoft","w3cschool");

### **返回指定索引区间的子串：substring（）**

例：var s = 'hello, world'

s.substring(0, 5); // 从索引0开始到5（不包括5），返回'hello'

s.substring(7); // 从索引7开始到结束，返回'world'

## 字符串大小写转换：字符串大小写转换使用函数 ****toUpperCase( )**** / ****toLowerCase( )****:

例：var txt="Hello World!";       // String  
 var txt1=txt.toUpperCase();   // txt1 文本会转换为大写

## 字符串转为数组：字符串使用****split( )****函数转为数组:

例：txt="a,b,c,d,e"   // String  
 txt.split(",");   // 使用逗号分隔  
 txt.split(" ");   // 使用空格分隔

txt.split(/\s+/); //使用若干字符分隔

## 特殊字符：使用反斜线（\）插入特殊符号

## 注意：字符串是不可变的 例：var s = 'Test';

s[0] = 'X';

alert(s); // s仍然为'Test'

# **Date（日期） 对象 :** 用于处理日期和时间.

[getFullYear( )](http://www.runoob.com/try/try.php?filename=tryjsref_getfullyear) ：返回年份。 同理getMonth( ) , getDate( ) , getDay( ) , getHour( )

[getTime( )](http://www.runoob.com/try/try.php?filename=tryjsref_gettime) ：返回从 1970 年 1 月 1 日至今的毫秒数。

[setFullYear( 年，月，日)](http://www.runoob.com/try/try.php?filename=tryjsref_setfullyear2)：设置具体的日期。

[toUTCString( )](http://www.runoob.com/try/try.php?filename=tryjsref_toutcstring)：将当日的日期（根据 UTC）转换为字符串。

[getDay( )](http://www.runoob.com/try/try.php?filename=tryjsref_date_weekday)：显示星期，而不仅仅是数字。weekday[d.getDay()]返回星期，weekday数组需自己创建

**有四种方式初始化日期:**

new Date() // 当前日期和时间  
 new Date(milliseconds) //返回从 1970 年 1 月 1 日至今的毫秒数  
 new Date(dateString)  
 new Date(year, month, day, hours, minutes, seconds, milliseconds)

**设置日期:**

例：设置了一个特定日期(2010 年 1 月 14 日)：new Date( ).setFullYear(2010,0,14);

设置为 5 天后的日期：new Date().setDate(myDate.getDate()+5);

**两个日期比较:** 直接对两个对象实例进行比较就可以。

**时区**：Date.now()返回1970年1月1日零时整的GMT时区开始的那一刻，到现在的毫秒数。

var d = new Date(返回的毫秒数); //返回UTC时间

d.toLocaleString(); // 转换成当地时间

d.toUTCString(); //转换成UTC时间

# **Array（数组） 对象**

**创建一个数组：**

1常规方式:

var myCars=new Array();   
 myCars[0]="Saab";         
 myCars[1]="Volvo";  
 myCars[2]="BMW";

2简洁方式:

var myCars=new Array("Saab","Volvo","BMW");

3字面:

var myCars=["Saab","Volvo","BMW"];

**访问数组：** myCars[0];

# **修改数组：** myCars[0]="Opel";

数组中包含对象元素、函数、数组

## 数组方法和属性：

## var x=myCars.length             // myCars 中元素的数量 var y=myCars.indexOf("Volvo")   // "Volvo" 值的索引值

## var z = myCars.slice(0,2) //返回=["Saab","Volvo"]，不给参数等于复制数组

## 

## push和pop

push( )向Array的末尾添加若干元素，pop( )则把Array的最后一个元素删除掉

例：myCas.push(‘a’,’f’); //返回myCars新长度

myCars.pop( ) //返回数组最后一个元素

## unshift和shift

unshift( )方法往Array的头部添加若干元素，shift( )方法则把Array的第一个元素删掉

例：myCas.unshift(‘a’,’f’); //返回myCars新长度

myCars.shift( ) //返回数组第一个元素

## sort：对当前Array进行排序 例：myCars.sort( )

**reverse：** 反转数字 例: myCars.reverse( )

**splice :** 可以从指定的索引开始删除若干元素，然后再从该位置添加若干元素

例：从索引2开始删除3个元素,然后再添加两个元素:

arr.splice(2, 3, 'Google', 'Facebook'); // 返回删除的元素的数组

**concat :** 把当前的Array和另一个Array连接起来，并返回一个新的Array

注意： concat()方法并没有修改当前Array，而是返回了一个新的Array。

**Join ：**把Array的每个元素用指定的字符串连接起来，然后返回连接后的字符串

例：var arr = ['A', 'B', 'C', 1, 2, 3];

arr.join('-'); // 'A-B-C-1-2-3'

## 创建新方法： 例：fruits.myUcase( );

需要：Array.prototype.myUcase=function(){ 。。。}

**多维数组**：var arr = [[1, 2, 3], [400, 500, 600], '-'];

# **Boolean（布尔） 对象**

## 创建 Boolean 对象： var myBoolean=new Boolean( );

如果布尔对象无初始值或者其值为:0/-0/null/""/false/undefined/NaN对象的值为 false。

如：var b=new Boolean(""); 则b值为false

# **Math（算数） 对象:** 执行常见的算数任务。

# Math.[round( )](http://www.runoob.com/try/try.php?filename=tryjsref_round) [四舍五入及取整](http://blog.sina.com.cn/s/blog_5106481f01009f56.html)

### Math.[random( )](http://www.runoob.com/try/try.php?filename=tryjsref_random) 0 到 1 之间的随机数。

Math.[max( )](http://www.runoob.com/try/try.php?filename=tryjsref_max) 返回两个给定的数中的较大的数。

Math.[min( )](http://www.runoob.com/try/try.php?filename=tryjsref_min) 返回两个给定的数中的较小的数。

常量使用方法: Math.E / Math.PI /Math.SQRT2

# **JavaScript 函数：**使用关键词 function：

function functionname(var1,var2)  
 { 执行代码

（return x;）  
 }

**函数表达式可以存储在变量中：**var x = function (a, b) {return a \* b};  
 var z = x(4, 3);

## Function( ) 构造函数: 函数同样可以通过内置的 JavaScript 函数构造器（Function()）定义。

## 按照完整语法需要在函数体末尾加一个;，表示赋值语句结束。

## 例：var myFunction = new Function("a", "b", "return a \* b"); var x = myFunction(4, 3); （x此时为12）

## 函数提升（Hoisting）：函数可以在声明之前调用

## 例：myFunction(5); function myFunction(y) { return y \* y; }

## 自调用：如果表达式后面紧跟 ( ) ，则会自动调用。

## 例：(function () {var x = "Hello!!"; })( );

## 函数是对象：****typeof****操作符判断函数类型将返回 "function"，但是函数描述为一个对象更加准确。

## JavaScript 函数有**属性**和**方法**。

## arguments.length 属性返回函数调用过程接收到的参数个数： return arguments.length;

## toString() 方法将函数整体作为一个字符串返回: return myFunction.toString();

# **默认参数：** 参数会默认设置为：undefined

y = y || 0; 如果y已经定义 ， y || 返回 y, 因为 y 是 true, 否则返回 0

### **rest参数:** 获取除了已定义参数之外的参数, 多余的参数以数组形式交给变量rest

形式：**function** foo(a, b, ...rest) { }

## Arguments 对象： 函数内部的arguments 对象，它包含了函数调用的传入参数数组。

## 可以通过遍历arguments数组访问所有参数。

# **函数调用：** 函数有 4 种调用方式。

**this：**在Javascript中，this指向函数执行时的当前对象。

1. a.b( ), b中的this指向a。

2. b( ), b中的this指向全局对象，也就是windows。

1. **作为函数调用** function myFunction(a, b) { return a \* b; }

myFunction(10, 2); 或 window.myFunction(10, 2); //返回 20

在浏览器中的页面对象是浏览器窗口(window 对象)。以上函数会自动变为 window 对象的函数。

所以myFunction() 和 window.myFunction() 是一样的

1. **作为对象中的方法调用：**JavaScript 中你可以将函数定义为对象的方法。

var myObject = { firstName:"John", lastName: "Doe",

fullName: function () { return this.firstName + " " + this.lastName; }}; myObject.fullName(); // 返回 "John Doe"

**fullName** 函数绑定在对象上，叫做方法。 **myObject**对象是函数的所有者。

return this; // 返回 [object Object] (所有者对象)

**3**. **使用构造函数调用函数**：如果函数调用前使用了 **new** 关键字, 则是调用了构造函数。

这看起来就像创建了新的函数，但实际上 JavaScript 函数是创建了新对象。

function myFunction(arg1, arg2) { this.firstName = arg1;

this.lastName = arg2; }

var x = new myFunction("John","Doe");

x.firstName; // 返回 "John"

构造函数的调用会创建一个新的对象。新对象会继承构造函数的属性和方法。

## 使用函数方法调用函数 ：在 JavaScript 中, 函数是对象。JavaScript 函数有它的属性和方法。

## ****call( )**** 和 ****apply( )**** 是预定义的函数方法。 两个方法可用于调用函数，第一个参数必须是对象本身。

例： function myFunction(a, b) { return a \* b; }

myObject = myFunction.call(myObject, 10, 2); // 返回 20

myArray = [10, 2];

myObject = myFunction.apply(myObject, myArray); // 返回 20

两个方法都使用了对象本身作为第一个参数。

两者的区别在于第二个参数：call直接按顺序传入参数， apply传入的是二个参数数组。

在 严格模式(strict mode)下, 在调用函数时第一个参数会成为 **this** 的值， 即使该参数不是一个对象。

在 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

通过 call() 或 apply() 方法你可以设置 this 的值, 且作为已存在对象的新方法调用。

## 

# **JavaScript 作用域：**

局部变量：只能在函数内部访问。函数执行完后局部变量会自动销毁。

全局变量：定义在函数外。 网页中所有脚本和函数均可使用。

如果变量在函数内没有声明（没有使用 var 关键字），该变量为全局变量。

例： function myFunction() { carName = "Volvo"; }

全局变量是 window 对象: 所有数据变量都属于 window 对象。

上例carName 可以使用  window.carName

定义的全局变量，或者函数，可以覆盖 window 对象的变量或者函数。  
 局部变量，包括 window 对象可以覆盖全局变量和函数

**let :** 可以申明一个块级作用域的变量, 比如只在for循环中起作用。

**const：**定义常量。

# **JavaScript 内嵌函数：** 嵌套函数可以访问上一层的函数变量。

例：内嵌函数 **plus( )** 可以访问父函数（add）的 **counter**变量

function add() { var counter = 0;  
    function plus() {counter += 1;}  
    plus();

}

**闭包(Closure)**：当一个函数返回了一个函数后，其内部的局部变量还被新函数引用

var add = (function () { var counter = 0;  
    return function () {return counter += 1;} } )( );  
  
 add(); add(); add(); // counter值为 3

变量 **add** 指定了函数自我调用的返回值（为一个函数）。

自我调用函数只执行一次。设置计数器为 0。并返回新函数。

add变量成为一个函数。它可以访问函数上一层作用域的计数器。

这个叫作 JavaScript **闭包。**它使得函数拥有私有变量。

计数器受匿名函数的作用域保护，只能通过 add 方法修改。

**高阶函数：**一个函数接受的参数为另外一个函数。

例：add(-5, 6, Math.abs);

1. **map( )**：定义在array中的方法，传入自己的函数，返回新的array。

例： function pow（x）{ return x\*x ；}

Var arr=[1，2，3，4，5，6，7，8，9] ;

arr.map(pow); // [1, 4, 9, 16, 25, 36, 49, 64, 81]

1. **reduce( )**：定义在array中的方法，传入自己的函数，这个函数必须接受两个参数， reduce（）会将array前两个元素送进函数，返回的结果加下一个元素再送进函数，依此类推。

例：对数组进行求和。

var arr = [1, 3, 5, 7, 9];

arr.reduce(function (x, y) {

return x + y; }); // 25

1. **filter( ) :** 传入的函数依次作用于每个元素，然后根据函数决定保留还是丢弃元素。

例：只保留数组的奇数。

var arr = [1, 2, 4, 5, 6, 9, 10, 15];

arr.filter(function (x) {

return x % 2 !== 0; }); // [1, 5, 9, 15]

1. **sort( ):** 两个元素x和y，如果x < y返回-1，如果x == y返回0，如果x > y返回1

sort方法为array的高阶函数。

例：按数字大小排序

var arr = [10, 20, 1, 2];

arr.sort(function (x, y) {

if (x < y) return -1;

if (x > y) return 1;

return 0;

}); // [1, 2, 10, 20]

# **箭头函数（Arrow Function）**：定义用的就是一个箭头

1. 用法： x => x \* x

相当于：function (x) {

return x \* x;

}

1. 包含多条语句，不能省略 {….}

x => {

if (x > 0) return x \* x;

else return - x \* x;

}

如果参数不止一个，需要用（ ） 例：(x, y) => x \* x + y \* y

如果返回对象： x => ({ foo: x })

this在箭头函数中总是指向外层调用者。

**generator（生成器）：**generator由function\*定义，除了return语句，还可以用yield

返回多次。

例：function\* foo(x) { yield x + 1;

yield x + 2;

return x + 3; }

foo(5)仅仅是创建了一个generator对象，调用generator对象有两个方法：

1. 不断地调用generator对象的**next()**方法

例：var f = fib(5);

f.next(); // {value: 0, done: false}

f.next(); // {value: 1, done: false}

f.next(); // {value: 1, done: true}

next()方法会执行generator的代码，每次遇到yield x;就返回一个对象

{value: x, done: true/false}，然后“暂停”。返回的value就是yield的返回值，

done表示这个generator是否已经执行结束了。

如果done为true，则value就是return的返回值。表示对象执行完毕。

1. 用**for ... of**循环迭代generator对象，这种方式不需要我们自己判断done

例 ：for (var x of foo(5)) {

console.log(x); //依次输出6，7，8

}

# **JavaScript 事件：**事件触发时 JavaScript 可以执行一些代码。

HTML 事件的实例：

1. HTML 页面完成加载
2. HTML input 字段改变时
3. HTML 按钮被点击

常见的HTML事件的列表:

onchange HTML 元素改变

onclick 用户点击 HTML 元素

onmouseover 用户在一个HTML元素上移动鼠标

onmouseout 用户从一个HTML元素上移开鼠标

onkeydown 用户按下键盘按键

onload 浏览器已完成页面的加载

例：<button onclick='getElementById("demo").innerHTML=Date()'>11</button>

如果想修改元素本身：(使用 **this**.innerHTML) 🡺 onclick="this.innerHTML=Date()"

## JavaScript 字符串

## var carname = "Volvo XC60";

可以使用索引位置来访问字符串中的每个字符：var character = carname[7];

字符串长度：var sln = carname.length;

转义字符：使用反斜杠 (\) 来转义。 如 \' 表示单引号

字符串也可以是对象：  **var firstName = new String("John")**

var x = "John"; typeof x // 返回 String  
 var y = new String("John"); typeof y // 返回 Object

(x === y) //结果为 false，因为 x 是字符串，y 是对象

**(=== 为绝对相等，即数据类型与值都必须相等。)** (!==为值和类型至少有一个不等)

## For/In 循环

## JavaScript把null、undefined、0、NaN和空字符串''视为false

## var person={fname:"John",lname:"Doe",age:25};

## for (var x in person) 🡺 对象键(数组索引)：x ，值：person[x]

## Map和Set

## Map: 是一组键值对的结构。

创建方法： 1. var m = new Map([['Michael', 95], ['Bob', 75], ['Tracy', 85]]);

m.get('Michael'); // 95

2. var m = new Map(); // 空Map

m.set('Adam', 67); // 添加新的key-value

m.has('Adam'); // 是否存在key 'Adam': true

m.get('Adam'); // 67

m.delete('Adam'); // 删除key 'Adam'

m.get('Adam'); // undefined

**Set：**是一组没有重复的key的集合

创建方法：1. var s = new Set([1, 2, 3,'3']);

2. var s1 = new Set();

s.add(4); //添加

s.delete(4); //删除

#### **iterable**: Array、Map和Set都属于iterable类型。

具有iterable类型的集合可以通过新的for ... of循环来遍历。

数组： var a = ['A', 'B', 'C'];

Set： var s = new Set(['A', 'B', 'C']);

Map： var m = new Map([[1, 'x'], [2, 'y'], [3, 'z']]);

for (var x of a) { alert(x); }

**for ... of和for ... in循环区别:**

for ... in : 遍历是对象的属性名称。数组实际上也是一个对象，它的索引被视为一个属性。

for … of : 只循环集合本身的元素。

**forEach方法：**它接收一个函数，每次迭代就自动回调该函数。

数组： a.forEach(function (element, index, array) { …… });

Map： m.forEach(function (value, key, map) { …… });

Set： Set没有索引，因此回调函数的前两个参数都是元素本身：

s.forEach(function (element, sameElement, set) { …… });

如果对某些参数不感兴趣，可以忽略它们。

## JavaScript 标签: 在语句之前加上冒号 例： label: statements

## break 和 continue 语句仅仅是能够跳出代码块的语句。 例：break labelname;

## continue labelname;

continue 语句（带有或不带标签引用）只能用在循环中。

break 语句（不带标签引用），只能用在循环或 switch 中。

通过标签引用，break 语句可用于跳出任何 JavaScript 代码块：

## JavaScript typeof, null, 和 undefined

## typeof :检测变量的数据类型。 例：typeof [1,2,3,4]   // 返回 object

（在JavaScript中，数组是一种特殊的对象类型。 因此 typeof [1,2,3,4] 返回 object。）

返回类型有：number、string、boolean、function和undefined

## Null：表示一个空对象引用。用 typeof 检测 null 返回是object。

## 例：var person = null;   // 值为 null(空), 但类型为object

## undefined ：没有设置值的变量。

## 例：var person = undefined; // 值为 undefined, 类型是undefined

## 

## Undefined 和 Null 的区别：

## typeof undefined             // undefined typeof null                  // object null === undefined           // false null == undefined            // true

**JavaScript 类型转换**

5 种不同的数据类型：

String / number / Boolean / object / function

3 种对象类型（**typeof** 都是 返回 Object）：

Object / Date / Array

2不包含任何值的数据类型：

Null / undefined

constructor 属性返回变量或对象的构造函数。

例："john".constructor 返回function String() { [native code] }

可以使用 constructor 属性来判断对象是否为数组 (利用字符串是否包含 "Array"字符)

myArray.constructor.toString().indexOf("Array") > -1; 若为数组则返回true

**类型转换：**使用 JavaScript 函数 或通过 JavaScript 自身自动转换

## 数字转换为字符串：全局方法 **String()** 可以将数字转换为字符串。 例：String(123)

Number 方法 **toString()** 也是有同样的效果。 例：(123).toString()

## 布尔值转换为字符串：全局方法 **String()**  例：String(false)  // 返回 "false"

Boolean 方法 **toString() 例：**false.toString()

## 日期转换为字符串：全局方法 **String()**  例：String(Date())

**toString()**  例：Date().toString()

## 字符串转换为数字： 全局方法 ****Number()****  例：Number("3.14")    // 返回 3.14

## 空字符串转换为 0。 其他的字符串会转换为 NaN (不是个数字)。

## 一元运算符 +：将变量转换为数字 例：var y = "5"; var x = + y;  // x 是一个数字

var y = "John"; var x = + y;  // x 值为NaN

## 布尔值转换为数字：全局方法 **Number()** ：Number(false) //返回0 Number(true) //返回1

## 日期转换为数字：全局方法 **Number()**  例：d = new Date(); Number(d)

日期方法 **getTime()**  例：d = new Date(); d.getTime()

## 自动转换类型

5 + null    // 返回 5         null 转换为 0  
 "5" + null  // 返回"5null"   null 转换为 "null"  
 "5" + 1     // 返回 "51"      1 转换为 "1"    
 "5" - 1     // 返回 4         "5" 转换为 5

## 自动转换为字符串：自动调用变量的 toString() 方法

document.getElementById("demo").innerHTML = myVar;  
 // if myVar = {name:"Fjohn"}  // toString 转换为 "[object Object]"  
 // if myVar = [1,2,3,4]       // toString 转换为 "1,2,3,4"  
 // if myVar = new Date()  // toString 转换为 "Fri Jul 18 2014 09:08:55 GMT

数字和布尔值也经常相互转换：

// if myVar = 123             // toString 转换为 "123"  
 // if myVar = true            // toString 转换为 "true"  
 // if myVar = false           // toString 转换为 "false"

**JavaScript 正则表达式（Regular Expression）**

语法： /正则表达式主体/修饰符(可选)

JavaScript 中，正则表达式通常用于两个字符串方法 : search() 和 replace()。

**search( ) ：**用于检索字符串中的子字符串，并返回子串的起始位置。

例：搜索 "Runoob" 字符串，且不区分大小写：（**i**  是一个**修饰符** (搜索不区分大小写））

var str = "Visit Runoob!";

var n = str.search(/Runoob/i); 结果为：6

search( ) 也可使用字符串作为参数：var n = str.search("Runoob");结果为：6

**replace( ) ：**用于在字符串中用一些字符（串）替换另一些字符（串）。

例：将字符串中的 Microsoft 替换为 Runoob :

var txt = str.replace(/microsoft/i,"Runoob");

## replace( ) 也可使用字符串： var txt = str.replace("Microsoft","Runoob");

## 修饰符****：****可以在全局搜索中不区分大小写修饰符-描述

i：执行对大小写不敏感的匹配。

g：执行全局匹配（查找所有匹配后停止）。可以多次执行exec()方法来搜索一个匹配的字符串

m：执行多行匹配。

## 正则表达式模式

方括号用于查找某个范围内的字符：

（1）[abc] 查找方括号之间的任何字符。

（2）[0-9] 查找任何从 0 至 9 的数字。

（3）(x|y) 查找x或y。

元字符是拥有特殊含义的字符：

（1）\d 查找数字。 [\D](http://www.runoob.com/jsref/jsref-regexp-digit-non.html) 查找非数字字符

（2）\s 查找空白字符。 [\S](http://www.runoob.com/jsref/jsref-regexp-whitespace-non.html) 查找非空白字符。

（3）\b 匹配单词边界。 [\B](http://www.runoob.com/jsref/jsref-regexp-begin-not.html) 匹配非单词边界。

（4）\uxxxx 查找以十六进制数 xxxx 规定的 Unicode 字符。

（5）转义字符\ 例： \- 匹配 ‘-’

量词:

（1）n+ 匹配包含至少一个 *n* 的字符串。

（2）n\* 匹配包含任意个（包含0个）*n* 的字符串。

（3）n? 匹配包含零个或一个 *n* 的字符串。

（4）n{n} 匹配包含n个n 的字符串。

（5）n{n,m} 匹配包含n-m个n 的字符串。

（6）^n 匹配开头为n 的字符串。

（7）n$ 匹配结束为n 的字符串。 ^n$则只匹配n

（8） . 匹配任意字符

例：\d{3}\s+\d{3,8} 表示 匹配3个数字+一个空格+3-8个数字

**使用 RegExp 对象**

**创建RegExp对象：**

当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）。

1.创建对象: new RegExp('正则表达式'，修饰符 ) 例：var re = new RegExp("\\w+");

2.直接写：/正则表达式/修饰符 例：var re = /\w+/;

**注意**：第一种方法表达转义字符时使用\\,此时表示一个\。

## test( )：检测一个字符串是否与正则表达式匹配（返回 true或false。）

例：/e/.test("The best things in life are free!")

字符串中含有 "e"，所以该实例输出为：true

## exec( )：提取字符串中与正则表达式匹配的子字符串数组。

返回一个存放匹配结果的数组。如果未找到匹配，则返回值为 null。如果有( )表示提取的分组

例：/e/.exec("The best things in life are free!");

**JavaScript 错误**

**try**  测试代码块的错误。

**catch**  处理错误。

JavaScript 语句 **try** 和 **catch** 是成对出现的。

语法：try {

//在这里运行代码

} catch(err) {

//在这里处理错误

}

**throw**  创建自定义错误

语法：throw exception

例：try { var x=document.getElementById("demo").value;

if(x < 5) throw "太小"; }

catch(err) { var y=document.getElementById("mess");

y.innerHTML="错误：" + err + "。"; } }

如果输入2，则抛出错误：太小。

**JavaScript 调试**

很多浏览器都内置了调试工具，一般是按下 F12 键，并在调试菜单中选择 "Console" 。

## console.log( ) ：在调试窗口上打印 JavaScript 值。 例：console.log(10);

**debugger关键字** ：用于停止执行 JavaScript，并调用调试函数。

例：<script> var x = 15 \* 5;

debugger;

document.getElementById("demo").innerHTML = x;

</script> （开启 debugger ，代码在第三行前停止执行。）

**JavaScript 变量提升（hoisting）**

JavaScript 中，函数及变量的声明都将被提升到函数的最顶部。

JavaScript 中，变量可以在使用后声明，也就是变量可以先使用再声明。

## 初始化不会提升：只有变量声明（var x）会被提升，初始化（x=5）不会。

例： var x ; //声明 x

var x = 5; //初始化 x

**JavaScript 严格模式 (strict mode)：**严格模式下你不能使用未声明的变量

## **"use strict" 指令：**指定代码在严格条件下执行。（写在第一行）

例："use strict";  
 x = 3.14;      //**调试模式会报错** (x 未定义)

在函数内部声明"use strict"; 表示局部作用域 (只在函数内使用严格模式)

# **JavaScript 表单**: **HTML 表单验证可以通过 JavaScript 来完成。**

# **获取html表单中的值:**

# 第一种：<input type="text" value="123" name="name1"/> document.forms[表单name属性值][name1].value 第二种：<input type="text" value="123" id="id1"/> document.getElementById("id1").value

**表单验证**

# 必填（或必选）项目: 如果为空 需要返回false，表示数据有问题。

# var x=document.forms["myForm"]["fname"].value;

# if (x==null || x=="") { alert("姓必须填写"); return false; }

## E-mail 验证: 输入的数据必须包含 @ 符号和点号(.)，同时，@ 不可以是邮件地址的首字符，并且 @ 之后需有至少一个点号。

var x=document.forms["myForm"]["email"].value;

var atpos=x.indexOf("@");

var dotpos=x.lastIndexOf(".");

if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)

{ alert("不是一个有效的 e-mail 地址"); return false; }

**JavaScript 验证 API：**

# **约束验证 DOM 方法：**

checkValidity( )：如果 input 元素中的数据是合法的返回 true，否则返回 false。

setCustomValidity( )：设置 input 元素的 validationMessage 属性，用于自定义错误提示信息 的方法。

# **约束验证 DOM 属性：**

validity：布尔属性值，返回 input 输入值是否合法

validationMessage：浏览器错误提示信息

willValidate：指定 input 是否需要验证

例：<input id="id1" type="number" min="100" max="300" required>

var inpObj = document.getElementById("id1");

if (inpObj.checkValidity() == false) {

document.getElementById("demo").innerHTML = inpObj.validationMessage; }

**validity：**包含一系列关于数据的属性:

rangeOverflow ：为 true, 如果元素的值大于设置的最大值。

rangeUnderflow ：为true, 如果元素的值小于它的最小值。

tooLong ：为 true, 如果元素的值超过了 maxLength 属性设置的长度。

valid ：为 true，如果元素的值是合法的。

例：var inpObj = document.getElementById("id1");

if (inpObj.validity.rangeUnderflow) { txt = "输入的值太小了";}

**JavaScript JSON （** **JavaScript Object Notation）**

JSON 是用于存储和传输数据的格式。通常用于服务端向网页传递数据 。

JSON 是一种轻量级的数据交换格式。

JSON是独立的语言 ：使用 JavaScript 语法，但是 JSON 格式仅仅是一个文本。  
 文本可以被任何编程语言读取及作为数据格式传递。

**JSON 语法规则**：1.数据为 键:值 对。键和字符串必须用双引号"" 例："name":"Runoob"

2.数据由逗号分隔。

3.大括号保存对象。 例：{"name":"Runoob", "url":"www.runoob.com"}

4.方括号保存数组。 例：对象 "sites" 是一个数组，包含了三个对象。

"sites":[{"name": "Runoob", "url":"www.runoob.com"},

{"name": "Google", "url":"www.google.com"},

{"name": "Taobao", "url":"www.taobao.com"}]

## JSON 字符串转换为 JavaScript 对象

通常我们从服务器中读取 JSON 数据，并在网页中显示数据。

1. 简单起见创建 JavaScript 字符串，字符串为 JSON 格式的数据：

var text = '{ "sites" : [' +

'{ "name":"Runoob" , "url":"www.runoob.com" },' +

'{ "name":"Google" , "url":"www.google.com" },' +

'{ "name":"Taobao" , "url":"www.taobao.com" } ]}';

1. 使用 JavaScript 内置函数 JSON.parse() 将字符串转换为 JavaScript 对象

var obj = JSON.parse(text);

1. 在页面中使用新的 JavaScript 对象

例：document.write = obj.sites[1].name + " " + obj.sites[1].url;

结果：Google [www.google.com](http://www.google.com)

## 相关函数

[**JSON.parse( )**](http://www.runoob.com/js/javascript-json-parse.html)： 将 JSON 字符串转换为 JavaScript 对象。

[**JSON.stringify(obj,keys,’ ‘ )**](http://www.runoob.com/js/javascript-json-stringify.html)：将 JavaScript 值转换为 JSON 字符串。 第二个参数用于筛选对象的键值，可以传入Array。 第三个参数按缩进输出

可以给对象内部定义一个**toJSON( )**的方法，直接返回JSON应该序列化的数据：

例：toJSON: function ( ) {

return {

'Name': this.name, // 只输出name，并且改变了key名

};

**Javascript:void(0)**

javascript:void(0) 中最关键的是 void 关键字，该操作符要计算括号内表达式但是不返回值。

语法格式如下：<head>

<script type="text/javascript">

void func( ) 或者：void(func( ))

javascript:void func( ) javascript:void(func( ))

</script>

</head>

例：创建了一个超级链接，当用户点击以后不会发生任何事：

<a href="javascript:void(0)">单击此处什么也不会发生</a>

在用户点击链接后显示警告信息：

<a href="javascript:void(alert('Warning!!!'))">点我!</a>

参数 a 将返回 undefined （结果：a = undefined b = 5 c = 7）:

function getValue(){  
   var a,b,c;  
   a = void ( b = 5, c = 7 );  
   document.write('a = ' + a + ' b = ' + b +' c = ' + c ); }

## href="#"与href="javascript:void(0)"的区别

**#** 包含了一个位置信息，默认的锚是**#top** 也就是网页的上端。而javascript:void(0), 仅仅表示

一个死链接。在页面很长的时候会使用 **#** 来定位页面的具体位置，格式为：**# + id**。如果要定

义一个死链接请使用 javascript:void(0) 。

例：<a href="#pos">点我定位到指定位置!</a>

<p id="pos">尾部定位点</p>

## HTML 载入外部 JavaScript 文件

<script src="myscript.js">

## 使用 JavaScript 访问 HTML 元素

var obj = getElementById("Demo")

**面向对象编程**

javaScript不区分类和实例的概念，而是通过原型（prototype）来实现面向对象编程。

Obj1.**\_\_proto\_\_** = obj2 obj1拥有obj2的属性和方法。

它没有“Class”的概念，所有对象都是实例，继承关系不过是把一个对象的原型指向另一个对象而已。

**注意**：在编写JavaScript代码时，不要直接用obj.\_\_proto\_\_去改变一个对象的原型。

**Object.create( )**方法 可以传入一个原型对象(相当于父类)，并创建一个基于该原型的新对象，但是新对象不存在自己的属性和方法，都是继承下来的。

用法： 原型对象 var student = { 。。。。。。。} ；

var xiaoming = Object.create（student）；

## 创建对象：JavaScript对每个创建的对象都会设置一个原型，指向它的原型对象。

## 当我们用obj.xxx访问一个对象的属性时，JavaScript引擎先在当前对象上查找该属性，如果 没有找到，就到其原型对象上找，如果还没有找到，就一直上溯到Object.prototype对象，最后，如果还没有找到，就只能返回undefined。

## 例：创建一个Array对象： var arr = [1, 2, 3];

## 其原型链是： arr ----> Array.prototype ----> Object.prototype ----> null

## Array.prototype定义了indexOf( )、shift( )等方法，因此所有的Array对象可以直接调用这 些方法。

## 例：创建一个函数时：function foo( ) { return 0; }

## 函数的原型链是： foo ----> Function.prototype ----> Object.prototype ----> null

## 由于Function.prototype定义了apply()等方法，因此，所有函数都可以调用apply()方法。

## 对象的原型链：xiaoming ----> Student.prototype ----> Object.prototype ----> null

#### **原型继承:??**

#### **class继承 :** 用class关键字来编写Student，可以这样写：

class Student {

constructor(name) { this.name = name; }

hello( ) { alert('Hello, ' + this.name + '!'); }

}

class的定义包含了构造函数constructor和定义在原型对象上的函数hello( )（注意没有 function关键字）.

创建一个Student对象：var xiaoming = new Student('小明');

继承通过extends来实现:

class PrimaryStudent extends Student {

constructor(name, grade) {

super(name); // 用super调用父类的构造方法!

this.grade = grade;

}

myGrade( ) { alert('I am at grade ' + this.grade); }

}

# JavaScript HTML DOM

通过 HTML DOM， JavaScript 可访问和操作 HTML 文档的所有元素。

当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model）。

HTML DOM 模型被构造为对象的树：

JS 能够改变页面中的所有 HTML 元素/HTML 属性/CSS 样式，对页面中的所有事件做出反应。

# **DOM Nodes（节点）**



整个文档是一个文档节点

每个 HTML 元素是元素节点

HTML 元素内的文本是文本节点

每个 HTML 属性是属性节点

注释是注释节点

**节点父、子和同胞**

父节点拥有子节点。同级的子节点被称为同胞（兄弟或姐妹）。

在节点树中，顶端节点被称为根（root）.document对象就是整个DOM树的根节点。

每个节点都有父节点、除了根（它没有父节点）

一个节点可拥有任意数量的子节点。

同胞是拥有相同父节点的节点

# **HTML DOM 方法和属性**

常用的 HTML DOM 方法：

getElementById(id) - 获取带有指定 id 的节点（元素）

appendChild(node) - 插入新的子节点（元素）到最后一项

removeChild(node) - 删除子节点（元素）

一些常用的 HTML DOM 属性：

innerHTML - 节点（元素）的文本值

parentNode - 节点（元素）的父节点

childNodes - 节点（元素）的子节点（返回NodeList，类似数组）

attributes - 节点（元素）的属性节点

nodeValue -规定节点的值。元素节点是 undefined 或 null，文本节点是文本本身，属性节点是属性值

# nodeType -返回节点的类型。元素-1，属性-2，文本-3，注释-8，文档-9。

# **DOM HTML**

 DOM 允许 JavaScript 改变 HTML 元素的内容。

**查找 HTML 元素**

1.通过 id 找到 HTML 元素

var x=document.getElementById("intro");

如果未找到该元素，则 x 将包含 null。

2.通过标签名找到 HTML 元素**（返回NodeList，类似数组）**

例 : 查找 id="main" 的元素，然后查找 id="main" 元素中的所有 <p> 元素

var x=document.getElementById("main");

var y=x.getElementsByTagName("p");

3.通过类(class)名找到 HTML 元素**（返回NodeList，类似数组）**

var x=document.getElementsByClassName("intro");

4.通过 querySelector( )和querySelectorAll( )

querySelector( ) 可以使用 id, 类, 类型, 属性, 属性值等参数来选取元素。

例：获取ID为q1的节点：var q1 = document.querySelector('#q1');

获取q1节点内所有符合条件的节点：q1.querySelectorAll('div.highlighted > p');

**改变 HTML 输出流**

在 JavaScript 中，document.write( ) 可用于直接向 HTML 输出流写内容。

绝对不要在文档加载完成之后使用 document.write( )。这会覆盖该文档。

**改变 HTML 内容**

1.最简单的方法时使用 innerHTML 属性。

例 : 改变<p id="p1">元素的内容 : document.getElementById("p1").innerHTML="新文本";

2. 修改innerText或textContent属性

document.getElementById('p-id') .innerText = '<script>alert("Hi")</script>';

**改变 HTML 属性**

document.getElementById(id).attribute= 新属性值

例 : 改变<img id="image" src="smiley.gif">元素的 src 属性:

document.getElementById("image").src="landscape.jpg";

# [**DOM CSS**](http://www.runoob.com/js/js-htmldom-css.html)

DOM 允许 JavaScript 改变 HTML 元素的样式。

**改变 HTML 样式**

document.getElementById(id).style.property= 新样式

例 : 改变 <p> 元素的样式：document.getElementById("p2").style.color="blue";

# **DOM 事件**

DOM 使 JavaScript 有能力对 HTML 事件做出反应。

**对事件做出反应**

例：当用户在 <h1> 元素上点击时，会改变其内容：

<h1 onclick="this.innerHTML='Ooops!'">点击文本!</h1>

或：function changetext(id) {id.innerHTML="Ooops!"; }

<h1 onclick="changetext(this)">点击文本! </h1>

**HTML 事件属性**

HTML 元素有事件属性。

例：向 button 元素分配 onclick 事件：

<button onclick="displayDate()">点这里</button>

**使用 HTML DOM 为元素添加事件**

例：document.getElementById("myBtn").onclick=function(){displayDate()};

function displayDate( ){

document.getElementById("demo").innerHTML=Date(); }

displayDate 函数被分配给 id="myBtn" 的 HTML 元素的onclick属性。

按钮点击时Javascript函数将会被执行。

**onload 和 onunload 事件**

onload 和 onunload 事件会在用户进入或离开页面时被触发.

onload 事件可用于检测访问者的浏览器类型和浏览器版本，并基于这些信息来加载网页的正确版本。

**onchange 事件**

onchange 事件常结合对输入字段的验证来使用。

当用户改变输入字段的内容时，会产生onchange事件。

**onmouseover 和 onmouseout 事件**

onmouseover 和 onmouseout 事件可用于在用户的鼠标移至 HTML 元素上方或移出元素时触发函数。

**onmousedown、onmouseup 以及 onclick 事件**

onmousedown, onmouseup 以及 onclick 构成了鼠标点击事件的所有部分。

首先当点击鼠标按钮时，会触发 onmousedown 事件，当释放鼠标按钮时，会触发 onmouseup 事件，最后，当完成鼠标点击时，会触发 onclick 事件。

# **DOM EventListener**

**addEventListener( ) 方法:**

用于向指定元素添加事件句柄。

添加的事件句柄不会覆盖已存在的事件句柄。

可以向一个元素添加多个事件句柄。

可以向同个元素添加多个同类型的事件句柄，如：两个 "click" 事件。

可以向任何 DOM 对象添加事件监听，不仅仅是 HTML 元素。如： window 对象。

使用 addEventListener() 方法时, JavaScript 从 HTML 标记中分离开来，可读性更强， 在没有控制HTML标记时也可以添加事件监听。

可以使用 removeEventListener() 方法来移除事件的监听。

## 语法：element.addEventListener(event, function, useCapture);

第一个参数是事件的类型 (不要使用 "on" 前缀，如 "click" 或 "mousedown").

第二个参数是事件触发后调用的函数。

第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。

## 例：当用户点击元素时弹出 "Hello World!"

*element*.addEventListener("click", function(){ alert("Hello World!"); });

可以使用函数名，来引用外部函数:

element.addEventListener("click", myFunction);

**向同一个元素中添加多个事件句柄：**

element.addEventListener("click", myFunction);  
 element.addEventListener("click", mySecondFunction);

**向 Window 对象添加事件句柄：**

例：当用户重置窗口大小时添加事件监听：

window.addEventListener("resize", myFunction );

**传递参数**

使用"匿名函数"调用带参数的函数：

element.addEventListener("click", function(){ myFunction(p1, p2); });

**事件冒泡或事件捕获**

事件传递有两种方式：冒泡与捕获。

如果你将 <p> 元素插入到 <div> 元素中

在 冒泡中，内部元素的事件会先被触发，然后再触发外部元素，即： <p> 元素的点击事件先触发，然后会触发 <div> 元素的点击事件。

在 捕获中，外部元素的事件会先被触发，然后才会触发内部元素的事件，即： <div> 元素的点击事件先触发 ，然后再触发 <p> 元素的点击事件。

addEventListener()方法可以指定 "useCapture" 参数来设置传递类型：

addEventListener(event, function, useCapture);

默认值为 false, 即冒泡，当值为 true 时, 事件使用捕获。

**removeEventListener( ) 方法**

removeEventListener() 方法移除由 addEventListener() 方法添加的事件句柄:

例：element.removeEventListener("mousemove", myFunction);

# **DOM 元素(Nodes节点)**

## 创建新的 HTML 元素 - createElement( )

## 例：创建新的<p> 元素：

var para=document.createElement("p");

如需向 <p> 元素添加文本，首先创建文本节点：

var node=document.createTextNode("这是一个新段落。");

然后向 <p> 元素追加这个文本节点：

para.appendChild(node);

最后向一个已有的元素追加这个新元素。

找到一个已有的元素：

var element=document.getElementById("div1");

以下代码在已存在的元素后添加新元素：

element.appendChild(para);

## 加在某子元素之前- insertBefore( )

例：document.getElementById(父元素ID).insertBefore(新元素,某个子元素);

## 删除已有的 HTML 元素

**DOM 需要清楚您需要删除的元素，以及它的父元素。**

## 例：删除<div> 元素里的一个<p>元素：

找到 id="div1" 的元素：

var parent=document.getElementById("div1");

找到 id="p1" 的 <p> 元素：

var child=document.getElementById("p1");

从父元素中删除子元素：

parent.removeChild(child);

另外一种方法：子元素使用parentNode 属性来找到父元素：

var child=document.getElementById("p1");  
 child.parentNode.removeChild(child);

注意：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。

## 替换 HTML 元素 -replaceChild( )

## document.getElementById(父元素ID).replaceChild(新元素,某个子元素)

# **HTML DOM 导航**

## HTML DOM 节点列表：

getElementsByTagName( ) 方法返回节点列表。节点列表是一个节点数组。

例：访问第二个 <p> var x=document.getElementsByTagName("p");

y=x[1];

## HTML DOM 节点列表长度： x.length

## 导航节点关系：

## 使用节点属性：parentNode、firstChild 以及 lastChild ，children（返回数组）在文档结构中进行导航。

## 例 ：document.getElementById("intro").firstChild

## DOM 根节点

## 两个特殊的属性，可以访问全部文档：

document.documentElement - 全部文档<HTML>

document.body - 文档的主体<body>

例：显示<body>内部内容 document.body.innerHTML

## childNodes 和 nodeValue

## 可以使用 childNodes 和 nodeValue 属性来获取元素的内容。

## 例：document.getElementById("intro").childNodes[0].nodeValue

## Window - 浏 览 器 对 象 模 型 (BOM)

# 浏览器对象模型（**B**rowser **O**bject **M**odel (BOM)）使 JavaScript 有能力与浏览器"对话"。

## Window 对象

所有浏览器都支持 window 对象。它表示浏览器窗口。

所有 JavaScript 全局对象、函数以及变量均自动成为 window 对象的成员。

全局变量是 window 对象的属性。

全局函数是 window 对象的方法。

甚至 HTML DOM 的 **document 也是 window 对象的属性之一：**

window.document.getElementById("header");

与此相同：document.getElementById("header");

## Window 尺寸：确定浏览器的大小

window.innerHeight 浏览器窗口的内部高度

window.innerWidth 浏览器窗口的内部宽度

## 其他 Window 方法

window.open( ) 打开新窗口

window.close( ) 关闭当前窗口

window.moveTo( ) 移动当前窗口

window.resizeTo( ) 调整当前窗口的尺寸

**Window.Screen对象 :** 包含有关用户屏幕的信息。

**window.screen**对象在编写时可以不使用 window 这个前缀。

一些属性：

screen.availWidth 可用屏幕宽度, 以像素计，减去界面特性，比如窗口任务栏。

screen.availHeight 可用屏幕高度，以像素计，减去界面特性，比如窗口任务栏。

# **window.location 对象 :** 获得当前页面的地址(URL),并把浏览器重定向到新的页面。

一些属性:

location.hostname 返回 web 主机的域名

location.pathname 返回当前页面的路径和文件名

location.port 返回 web 主机的端口 （80 或 443）

location.protocol 返回所使用的 web 协议（http:// 或 https://）

location.assign( )加载新的文档。例：location.assign("http://www.w3cschool.cc")

**window.history 对象：**包含浏览器的历史记录。

history.back( ) 与在浏览器点击后退按钮相同，加载历史列表中的**前一个** URL。

history.forward( ) 与在浏览器点击按钮向前相同，加载历史列表中的**下一个** URL。

## window.navigator 对象：包含有关访问者浏览器的信息。

浏览器代号: navigator.appCodeName

浏览器名称: navigator.appName

浏览器版本: navigator.appVersion

启用Cookies: navigator.cookieEnabled

硬件平台: navigator.platform

用户代理: navigator.userAgent

用户代理语言: navigator.systemLanguage

## JavaScript 弹窗

**警告框 语法：**window.alert("sometext"); 可以直接使用**alert( )**方法

### **确认框 语法：**window.confirm("sometext"); 可以直接使用**confirm ( )**方法

当点击 "确认"时返回 true， 如果点击 "取消"则返回 false。如var r=confirm("按下按钮")

**提示框 语法：**window.confirm("sometext" ,"*defaultvalue*");

当提示框出现后，用户需要输入某个值，然后点击确认或取消按钮才能继续操纵。

如果用户点击确认，那么返回值为输入的值。如果用户点击取消，那么返回值为 null

例： var person=prompt("请输入你的名字","Harry Potter"); 默认为哈里波特

**换行** 弹窗使用 反斜杠 + "n"(\n) 来设置换行。 例：alert("Hello\nHow are you?");：

## JavaScript 计时事件

JavaScript 一个设定的时间间隔之后来执行代码称之为计时事件

**setInterval( )** 间隔指定的毫秒数不停地执行指定的代码。

方法：window.setInterval("javascript function",*毫秒数*);

例：每三秒弹出一次 setInterval(function(){alert("Hello")},3000);

**clearInterval( )**：用于停止 setInterval( ) 方法执行的函数代码。 例：clearInterval(Variable)

**setTimeout( )** 暂停指定的毫秒数后执行指定的代码

方法：window.setTimeout("*javascript 函数*",*毫秒数*);

**例：** 等待3秒，然后弹出 "Hello": setTimeout(function(){alert("Hello")},3000);

**clearTimeout( )**  用于停止执行setTimeout( )方法的函数代码。

**Note:** setInterval( ) 和 setTimeout( ) 是 HTML DOM Window对象的两个方法

## JavaScript Cookie

Cookie 是一些数据, 存储于电脑上的文本文件中。

当用户访问 web 页面时，服务器发送一个cookie给浏览器。用户的名字会记录在 cookie 中。

在用户下一次访问该页面时，浏览器会在请求头附上这个Cookie。服务器可以根据 cookie 区分用户。

Cookie 以Key-Value对形式存储： username=John Doe

当浏览器从服务器上请求 web 页面时， 属于该页面的 cookie 会被添加到该请求中。服务端通过这种方式来获取用户的信息。

JavaScript 可以使用 **document.cookie** 属性来创建 、读取、及删除 cookie。

**创建 cookie** ：document.cookie="username=John Doe";

还可以为 cookie 添加一个过期时间（以 UTC 或 GMT 时间）。默认情况下，cookie 在浏览器关闭时删除： 例 ： document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT";

可以使用 path 参数告诉浏览器 cookie 的路径。默认情况下，cookie 属于当前页面。

例： document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";

**读取 Cookie:** 读取到当前页面的Cookie

使用以下代码来读取 cookie：var x = document.cookie; 以字符串的方式返回所有的 cookie。

类型格式： cookie1=value; cookie2=value; cookie3=value;

**修改 Cookie**

在 JavaScript 中，修改 cookie 类似于创建 cookie，如下所示：

document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";

旧的 cookie 将被覆盖。

**删除 Cookie**

只需要设置 expires 参数为以前的时间即可，设置为 Thu, 01 Jan 1970 00:00:00 GMT:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 GMT";

注意，删除时不必指定 cookie 的值。

## JavaScript 库

JavaScript 常用库 - jQuery、Prototype、MooTools。

**JavaScript 框架（库）**

JavaScript 库常被称为 **JavaScript 框架**。所有这些框架都提供针对常见 JavaScript 任务的函数，包括动画、DOM 操作以及 Ajax 处理。使得 JavaScript 编程更容易。

**jQuery**

jQuery 是目前最受欢迎的 JavaScript 框架。

它使用 CSS 选择器来访问和操作网页上的 HTML 元素（DOM 对象）。

jQuery 同时提供 companion UI（用户界面）和插件。

**Prototype**

**Prototype**提供用于执行常见 web 任务的简单 API。

**API** 是应用程序编程接口（Application Programming Interface）的缩写。它是包含属性和方法的库，用于操作 HTML DOM。

Prototype 通过提供类和继承，实现了对 JavaScript 的增强。

**MooTools**

**MooTools** 也是一个框架，提供了可使常见的 JavaScript 编程更为简单的 API。

MooTools 也含有一些轻量级的效果和动画函数。

**使用框架**

在使用 JavaScript 框架之前，首先对框架进行测试是明智的。

JavaScript 框架很容易进行测试。通常只需从网页中引用一个库文件。

## JavaScript - jQuery

**引用 jQuery**

如需测试 JavaScript 库，需要在网页中引用它。引用某个库，请使用 <script> 标签。 src 设置为库的 URL

例： <script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js">  
 </script>

**jQuery 描述**

主要的 jQuery 函数是 $( ) 函数（jQuery 函数）。如果向该函数传递 DOM 对象，它会返回 jQuery 对象，带有向其添加的 jQuery 功能。

jQuery 允许通过 CSS 选择器来选取元素。

例，处理窗口加载事件：

## JavaScript 方式： function myFunction( ) {      var obj=document.getElementById("h01");      obj.innerHTML="Hello jQuery";

## } onload=myFunction;

## jQuery 方式：function myFunction( ) {      $("#h01").html("Hello jQuery");

## } $(document).ready(myFunction);

代码的最后一行，HTML DOM 文档对象被传递到 jQuery ：$(document)。

当您向 jQuery 传递 DOM 对象时，jQuery 会返回以 HTML DOM 对象包装的 jQuery 对象。

jQuery 函数会返回新的 jQuery 对象，其中的 ready() 是一个方法。

由于在 JavaScript 中函数就是变量，因此可以把 myFunction 作为变量传递给 jQuery 的 ready 方法。

jQuery 返回 jQuery 对象，与已传递的 DOM 对象不同。  
jQuery 对象拥有的属性和方法，与 DOM 对象的不同。  
不能在 jQuery 对象上使用 HTML DOM 的属性和方法。

## JavaScript - Prototype

## 引用 Prototype

如需测试 JavaScript 库，需要在网页中引用它。引用某个库，请使用 <script> 标签。 src 设置为库的 URL

例： <script src="http://apps.bdimg.com/libs/prototype/2.1.1/ prototype.js">  
 </script>

## Prototype 描述

Prototype（原型） 提供的函数可使 HTML DOM 编程更容易。

与 jQuery 类似，Prototype 也有自己的 $( ) 函数。

$() 函数接受 HTML DOM 元素的 id 值（或 DOM 元素），并会向 DOM 对象添加新的功能。

与 jQuery 不同，Prototype 没有用以取代 window.onload( ) 的 ready() 方法。相反，Prototype 会向浏览器及 HTML DOM 添加扩展。

例： 处理窗口加载事件：

## JavaScript 方式： function myFunction( ){ var obj=document.getElementById("h01"); obj.innerHTML="Hello Prototype"; } onload=myFunction;

## Prototype 方式：function myFunction( ){ $("h01").insert("Hello Prototype!"); } Event.observe(window,"load",myFunction);

Event.observe() 接受三个参数：

希望处理的 HTML DOM 或 BOM（浏览器对象模型）对象

希望处理的事件

希望调用的函数