# JavaScript教程

## JavaScript简介

### 什么是JavaScript：

**JavaScript 是世界上最流行的编程语言。**

**这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。**

JavaScript 是脚本语言

* JavaScript 是一种轻量级的编程语言。
* JavaScript 是可插入 HTML 页面的编程代码。
* JavaScript 插入 HTML 页面后，他不需要在一个语言环境下运 行，而只需要支持它的浏览器即可。它的主要目的是，验证发往服务器端的数据、增加 Web 互动、加强用户体验度等。可由所有的现代浏览器执行。
* JavaScript 很容易学习。JavaScript前端 Java后端——二者无关。

### JavaScript历史

**引子**

大概在 1992 年，有一家公司 Nombas 开发一种叫做 C--(C-minus-minus,简称 Cmm)的嵌 入式脚本语言。后应觉得名字比较晦气，最终改名为 ScripEase。而这种可以嵌入网页中的 脚本的理念将成为因特网的一块重要基石。

**诞生**

1995 年，当时工作在 Netscape(网景)公司的布兰登(BrendanEich)为解决类似于“向服务 器提交数据之前验证”的问题。在 NetscapeNavigator2.0 与 Sun 公司联手开发一个称之为 LiveScript 的脚本语言。为了营销便利，之后更名为 JavaScript(目的是在 Java 这课大树下好 乘凉)。

**邪恶的后来者**

因为 JavaScript1.0 如此成功，所以微软也决定进军浏览器，发布了 IE3.0 并搭载了一 个 JavaScript 的克隆版，叫做 JScript（这样命名是为了避免与 Netscape 潜在的许可纠纷）， 并且也提供了自己的 VBScript。

**标准的重要**

在微软进入后，有 3 种不同的 JavaScript 版本同时存在：NetscapeNavigator 3.0 中的 JavaScript、IE 中的 JScript 以及 CEnvi 中的 ScriptEase。与 C 和其他编程语言不同的是， JavaScript 并没有一个标准来统一其语法或特性，而这 3 种不同的版本恰恰突出了这个问题。 随着业界担心的增加，这个语言标准化显然已经势在必行。

**ECMA**

1997 年，JavaScript1.1 作为一个草案提交给欧洲计算机制造商协会（ECMA）。第 39 技术委员会（TC39）被委派来“标准化一个通用、跨平台、中立于厂商的脚本语言的语法 和语义”（http://www.ecma-international.org/memento/TC39.htm）。由来自 Netscape、Sun、微 软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的 TC39 锤炼出了 ECMA-262，该标准定义了叫做 ECMAScript 的全新脚本语言。

**灵敏的微软、迟钝的网景**

虽然网景开发了 JavaScript 并首先提交给 ECMA 标准化，但因计划改写整个浏览器引擎 的缘故，网景晚了整整一年才推出“完全遵循 ECMA 规范”的 JavaScript1.3。而微软早在一 年前就推出了“完全遵循 ECMA 规范”的 IE4.0。这导致一个直接恶果：JScript 成为 JavaScript 语言的事实标准。

**标准的发展**

在接下来的几年里，国际标准化组织及国际电工委员会（ISO/IEC）也采纳 ECMAScript 作为标准（ISO/IEC-16262）。从此，Web 浏览器就开始努力（虽然有着不同程度的成功和失 败）将 ECMAScript 作为 JavaScript 实现的基础。

**山寨打败原创**

JScript 成为 JavaScript 语言的事实标准，加上 Windows 绑定着 IE 浏览器，几乎占据全 部市场份额，因此，1999 年之后，所有的网页都是基于 JScript 来开发的。而 JavaScript1.x变成可怜的兼容者。

**网景的没落与火狐的崛起**

网景在微软强大的攻势下，1998 年全面溃败。但，星星之火可以燎原。网景的高管DaveHyatt（戴夫 海厄特）同年成立 Mozilla 项目中 Firefox(火狐浏览器)在支持 JavaScript 方面无可比拟，在后来的时间里一步步蚕食 IE 的市场，成为全球第二大浏览器。

**谷歌的野心**

GoogleChrome，又称 Google 浏览器，是一个由 Google（谷歌）公司开发的开放原始码 网页浏览器。他以简洁的页面，极速的浏览，一举成为全球第三大浏览器。随着移动互联网 的普及，嵌有 Android 系统的平板电脑和智能手机，在浏览器这块将大有作为。

**苹果的战略**

Safari 浏览器是苹果公司各种产品的默认浏览器，在苹果的一体机(iMac)、笔记本(Mac)、 MP4(ipod)、iphone(智能手机)、ipad(平板电脑)，并且在 windows 和 Linux 平台都有相应版 本。目前市场份额全球第四，但随着苹果的产品不断的深入人心，具有称霸之势。

**幸存者**

Opera 的全球市场份额第五，2%左右。它的背后没有财力雄厚的大公司，但它从“浏览 器大战”存活下来的，有着非常大的潜力。

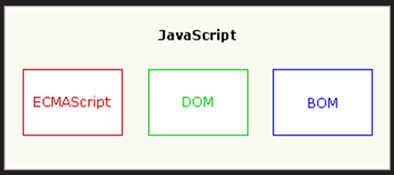
**JavaScript核心**

虽然 JavaScript 和 ECMAScript 通常被人们用来表达相同的含义，但 JavaScript 的含义 却比ECMA-262中规定的要多得多。一个完整的JavaScript应该由下列三个不同的部分组成。

1.核心(ECMAScript)

2.文档对象模型(DOM)

3.浏览器对象模型(BOM)



**ECMAScript 介绍**

由 ECMAScript-262 定义的 ECMAScript 与 Web 浏览器没有依赖关系。ECMAScript 定 义的只是这门语言的基础，而在此基础之上可以构建更完善的脚本语言。我们常见的 Web 浏览器只是 ECMAScript 实现可能的宿主环境之一。 既然他不依赖于 Web 浏览器，那么他还在哪些环境中寄宿呢？比如：ActionScript、 ScriptEase 等。而他的组成部分有：语法、类型、语句、关键字、保留字、操作符、对象等。

**ECMAScript 版本**

ECMAScript 目前有四个版本，1、2、3、4、5 版本，这里不再进行详细探讨。有兴趣 的同学，可以搜索查阅。

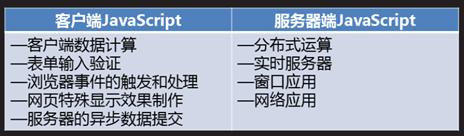
**Web浏览器对ECMAScript 的支持**

到了 2008 年，五大主流浏览器(IE、Firefox、Safari、Chrome、Opera)全部做到了与 ECMA-262 兼容。其中，只有 Firefox 力求做到与该标准的第 4 版兼容。以下是支持表。



### JavaScript特点：

* 代码可使用任何文本编辑工具编写，语法类似于c和java
* 无需编译，由javaScript引擎解释执行
* 弱类型语言
* 基于对象
* Javascript的用途



## JavaScript用法

**HTML 中的脚本必须位于 <script> 与 </script> 标签之间。**

**脚本可被放置在 HTML 页面的 <body> 和 <head> 部分中。**

### 1）<script> 标签

如需在 HTML 页面中插入 JavaScript，请使用 **<script>** 标签。

<script> 和 </script> 会告诉 JavaScript 在何处开始和结束。

<script> 和 </script> 之间的代码行包含了 JavaScript：

如：

<script>

alert("My First JavaScript");

</script>

### 2） <body> 中的 JavaScript

在本例中，JavaScript 会在页面加载时向 HTML 的 <body> 写文本：

<!DOCTYPE html>

<html>

<body>

<p>

JavaScript 能够直接写入 HTML 输出流中：

</p>

<script>

document.write("<h1>This is a heading</h1>");

document.write("<p>This is a paragraph.</p>");

</script>

<p>

您只能在HTML 输出流中使用 <strong>document.write</strong>。

如果您在文档已加载后使用它（比如在函数中），会覆盖整个文档。

</p>

</body>

</html>

### 3） <head> 中的 JavaScript 函数

**内部脚本块**

在本例中，我们把一个 JavaScript 函数放置到 HTML 页面的 <head> 部分。

该函数会在点击按钮时被调用：

<!DOCTYPE html>

<html>

<head>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML="My First JavaScript Function";

}

</script>

</head>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">点击这里</button>

</body>

</html>

**外部脚本块**

将javascript代码写入一个单独的文件，并保存为后缀名为.js文件

-为纯文本文件

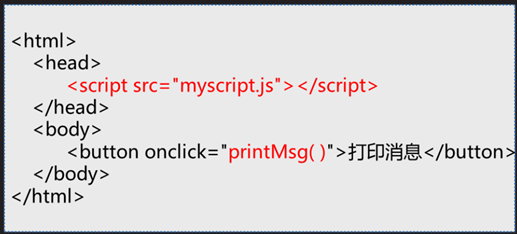
-文件中，不需要包含<script>标签，直接书写js代码



HTML页面的head中引用外部的 . js文件

—在<head>中添加<script>标签

—设置<script>标签的 src 属性，以指定js 文件的url



## JavaScript语句

**JavaScript 语句**

JavaScript 语句向浏览器发出的命令。语句的作用是告诉浏览器该做什么。

下面的 JavaScript 语句向 id="demo" 的 HTML 元素输出文本 "Hello World"：

document.getElementById("demo").innerHTML="Hello World";

**JavaScript 代码**

JavaScript 代码（或者只有 JavaScript）是 JavaScript 语句的序列。

浏览器会按照编写顺序来执行每条语句。

本例将操作两个 HTML 元素

<!DOCTYPE html>

<html>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph.</p>

<div id="myDIV">A DIV.</div>

<script>

document.getElementById("demo").innerHTML="Hello World";

document.getElementById("myDIV").innerHTML="How are you?";

</script>

</body>

</html>

**JavaScript 代码块**

JavaScript 语句通过代码块的形式进行组合。

块由左花括号开始，由右花括号结束。

块的作用是使语句序列一起执行。

JavaScript 函数是将语句组合在块中的典型例子。

下面的例子将运行可操作两个 HTML 元素的函数：

<!DOCTYPE html>

<html>

<body>

<h1>My Web Page</h1>

<p id="myPar">I am a paragraph.</p>

<div id="myDiv">I am a div.</div>

<p>

<button type="button" onclick="myFunction()">点击这里</button>

</p>

<script>

function myFunction()

{

document.getElementById("myPar").innerHTML="Hello World";

document.getElementById("myDiv").innerHTML="How are you?";

}

</script>

<p>当您点击上面的按钮时，两个元素会改变。</p>

</body>

</html>

## JavaScript语法

编写javascript代码

* 语句——会被javascript引擎解释执行的代码

由表达式、关键字、运算符组成；

————大小写敏感

——使用分号或者换行结束

* 注释——不会被javascript引擎解释执行的代码

——单行注释以 // 开头。

——多行注释以 /\* 开始，以 \*/ 结尾。

// 输出标题：

document.getElementById("myH1").innerHTML="Welcome to my Homepage";// 输出段落：

document.getElementById("myP").innerHTML="This is my first paragraph.";

/\*

下面的这些代码会输出

一个标题和一个段落

并将代表主页的开始

\*/

1、大小写敏感

JavaScript 对大小写是敏感的，包括变量、函数名和操作符都是区分大小写的。例如：text 和 Text 表示两种不同的变量。

当编写 JavaScript 语句时，请留意是否关闭大小写切换键。

函数 getElementById 与 getElementbyID 是不同的。

同样，变量 myVariable 与 MyVariable 也是不同的。

<script>

Function playMusic(){

//…… 大小写必须一致

}

</script>

<button onclick=”playMusic()”>播放音乐</button>

2、语句结束必须加分号；

分号用于分隔 JavaScript 语句。

通常我们在每条可执行的语句结尾添加分号。

使用分号的另一用处是在一行中编写多条语句。

提示：您也可能看到不带有分号的案例。

在 JavaScript 中，用分号来结束语句是可选的。

3、标识符 所谓标识符，就是指变量、函数、属性的名字，或者函数的参数。

1．第一字符只能以下划线\_、美元$、字母（大、小）开头

2．其他字符可以是字母、下划线、美元符号或数字。

3．不能和JavaScript中已经使用的名字相同（关键字）、未来的名字（保留字）

4、见名知义 studentName studentAge

建议第一个单词小写，第二个单词大写

## Javascript输出

操作 HTML 元素

如需从 JavaScript 访问某个 HTML 元素，您可以使document.getElementById(id) 方法。请使用 "id" 属性来标识 HTML 元素：

例子：

通过指定的 id 来访问 HTML 元素，并改变其内容：

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p id="demo">My First Paragraph.</p>

<script>

document.getElementById("demo").innerHTML="MyFirst JavaScript";

</script>

</body>

</html>

JavaScript 由 web 浏览器来执行。在这种情况下，浏览器将访问 id="demo" 的 HTML 元素，并把它的内容（innerHTML）替换为 "My First JavaScript"。

写到文档输出

下面的例子直接把 <p> 元素写到 HTML 文档输出中：

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<script>

document.write("<p>My First JavaScript</p>");

</script>

</body>

</html>

警告

请使用 document.write() 仅仅向文档输出写内容。

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

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p>My First Paragraph.</p>

<button onclick="myFunction()">点击这里</button>

<script>

function myFunction()

{

document.write("糟糕！文档消失了。");

}

</script>

</body>

</html>

## JavaScript变量

**什么是变量**

* 变量是存储信息的容器
* 在代数中，我们使用字母（比如x）来保存值（比如2）。

X=2

Y=3

Sum=x+y

* 通过上面三行语句，能够算出sum的值为5
* 在javascript中，这些字母被称为变量

**变量的声明**

* 定义变量时要使用 var 操作符（var 是关键）后面跟一个变量名（变量名是标识符）。

Var box;

alert(box);

* 使用” =”为变量赋值 初始化（赋值）： var变量=值；

var box= '李易峰';

var bookPrice = 25.5;

* 没有初始化的变量自动取值为undefined

Var publishDate;

alert(publisDate);

* 在变量声明时初始化

可以在定义变量时立即进行初始化

Var age = 10;

Var gender = ‘男’;

Var email = ‘tom@163.com’;

Var isOnSale = true;

* 变量也可以先声明，再赋值

Var birthday;

Var price;

Birthday = ‘2006-10-22’;

Price = 35.5;

Alert(price);

* 可以对变量中的值进行存取，操作

获取变量的值——get操作

Var userPwd = ‘123456’;

Document.write(userPwd);

Var anotherPwd = userPwd; //创建新变量，赋值为已有变量的值

重新设置变量的值——set操作

Var oldPwd = ‘123’;

oldPwd = ‘456’;

oldPwd = anotherPwd; //这条语句对于变量anotherPwd是Get操作；对于变量oldPwd是set操作

* 变量的操作必须与类型匹配

虽然javascript是弱类型语言，语法上允许变量先后赋值为任意有效值；但项目开发中，与变量的赋值和操作也需要注意是否满意实际业务逻辑

Var studentCout = 35;

studentCount = 35.5;

//studentCount（学生数量），在实际业务中，应该是整数，不应该是赋值为浮点数

Var bookName = ‘西游记’;

Aler(bookName>3);

//bookName(书名)，在实际业务中应该是字符串，不应该与一个整数比较大小

* 所谓变量，就是可以初始化后可以再次改变的量。Javascript属于

(松散类型) 的语言，可以同时改变不同类型的量。

Var boxString= '李易峰';

boxString=100;

alert(boxString);

* var声明的变量不指定具体的类型，赋值给什么类型的数据，变量就为什么类型。

var a4 = 10.2

a4 = "tom"

a4 = "false"

最终a4的值为逻辑类型的false

赋值会覆盖之前的数据。

取值只要直接写a4

重复的使用 var 声明一个变量，只不过是一个赋值操作，并不会报错。

Var box= '李易峰';

Var box= 'Lee';

* 一条语句中声明多个变量

可以在一条语句中声明多个变量，变量名使用“，”分隔

Var name1,name2,name3;

Var age1,age2=30;

**总结步骤：**

**1、声明。Var XXX**

**2、存放或者取出数据**

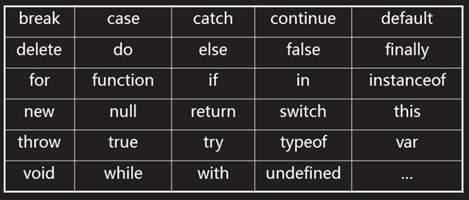
* 变量的作用：存放和取出数据的容器。
* 变量的声明需要包括：变量的名字、类型、初始值。
* Var 变量的名字=初始值;
* Var 变量名字;
* 声明好的变量后期赋值会覆盖之前的数据。

**3、JavaScript 变量命名**

命名需要符合标示符语法要求，不允许使用语言关键字和保留字作为变量名

* + - * 变量必须以字母开头，不能以数字开头
      * 变量也能以 $ 和 \_ 符号开头（不过我们不推荐
      * 变量名称对大小写敏感（y 和 Y 是不同的变量）
      * 提示：JavaScript 语句和 JavaScript 变量都对大小写敏感。

标准关键字



预保留的关键字



**4、未经过初始化的变量**

变量定义之后，在使用前从未赋值，值为undifined

Var username; //定义变量，但未被赋值

Console.log(username); //直接使用未被赋值的变量

变量未被定义过，而被直接使用，属于语法错误。

**常量的声明和使用**

**常量，是值不能改变的量**

**使用const关键字声明一个常量**

**习惯上，常量名使用纯大小形式**

Const PI = 3.1415926;

Const WITHDRAW\_LIMIT = 2000;

**有问题？？？**

var PI = 3.141592653;

PI = 3.3 //错误，但在JavaScript中不报错。

JavaScript没有严格意义上的常量，只能把变量名全部大写来表示为常量。提示自己和别人的作用。

案例1：定义一个变量存放100，在定义一个变量为50，求两个变量的和！

运算符：+ - \* / == “=”是赋值 “==”是等于

案例2：定义2个变量，第一个存放true，第二个存放false，交换两个变量的值。

案例3：已知圆的半径为5，求这个圆的周长和面积，分别在消息框中显示！

答案在下面。

练习：

买一辆汽车，汽车为10万元，首付20%，剩下的80%余款按月偿还，每个月偿还5000元，每个月会产生尾款0.001的利息。

1、 求最终买车的花多少钱？

2、将每个月的余款同利息输出到页面上。

## JavaScript数据类型

**Ja****vaScript数据类型：**

**例子：张三，年龄 25，性别 男**

* Name：张三 stevejobs 字符类型 A-Z a-z 0-9 特殊符号#$% 汉字
* Age：25 数字类型 正数、负数、整数、小数
* Sex：男 0 1 逻辑类型 真ture、假false 布尔值

**字符类型：**表示一系列的文本字符数据

* + - 字符‘a’‘c’‘0’‘$’
    - 字符串 “abc”“123”“a12”“This is JavaScript”
    - JavaScript的字符类型中没有区分字符与字符串，全部以字符串对待。可以写单引号，也可以写双引号。
    - 特殊字符需要转义符 转义符，如：\n , \r ,\’ , \”

Var msg=”欢迎来到\n \”javascript世界\” ”

**number类型：**

* 整数 100 23 -100
* 小数——在其他语言中叫做浮点型（小数 float浮点型 double双精度）
* Javascript中没有区分整数与小数，全部以小数对待 10=10.0

**逻辑类型：(boolean类型)**

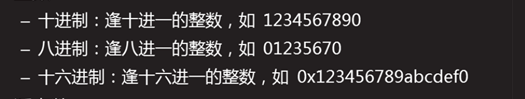
* ture 代表1
* false 代表0

为什么要出现逻辑类型？程序运行时在电路板中运行，如显卡，开关需要逻辑类型。（现实中也有逻辑类型：等没等到车；考没考上清华；登录没登录网站，登录可以看信息，没登录不能看信息。）

**延伸：数字类型进制**

* 二进制（计算机语言，没有二进制就没有电脑，0,1

与西方的文化相通，所以注定由外国人发明中国人有二进制思维：八卦 ）



//十六进制 0x或者 0X

var a = 0xC8;

alert(a); //显示出来的是十进制，转换为十进制就是 200

//八进制 0开头 0-7

var b = 02760;

alert(b); //显示出来也是十进制，值1520

**隐式转换**

Javascript属于松散类型的程序语言，变量在声明时不需要指定数据类型，变量由赋值操作确定数据类型

数字+字符串：数字转换为字符串

数字+布尔值： true转换为1，false转换为0

字符串+布尔值：布尔值转换为字符串true或false

布尔值+布尔值：布尔值转换为数值1或0

var str = "zhangling";

var num = 10;

var boot = true;

var boot2 = false;

document.write(str+num+"\n");

document.write(num+boot+"\n");

document.write(str+boot+"\n");

**Typoof()函数**可用于查询变量的当前类型

返回string/number/Boolean/object/function/undefined

例如 typeof(‘hello+0’),返回“string“

**数据类型转换函数**

Tostring() 转换成字符串，所有数据类型均可转换为string类型

parseInt()解析出一个string或number的整数部分，如果没有可以转换的部分，则返回NaN(not a Number)

parseFloat()解析出一个string的浮点数部分，如果没有可用转换的部分，则返回NaN

Number()把一个string解析为number,如果包含非法字符，则返回NaN

**isNaN函数**

isNaN() 用于判断其参数是否为一个“非数字（NaN）“值，如果把NaN与任何值（包括其自身）相比得到的结果均是false,所以要判断某个值是否是NaN,不能使用==或===运算符。这种情况下，只能使用isNaN()函数。通常用于检测类型转换函数的运算结果，以判断他们表示的是否是合法的数字。

Douument.write(isNaN(‘123.4a’));

Douument.write(parseInt (‘123.4a’));

Douument.write(parseFloat (‘123.4a’));

Douument.write(Number (‘123.4a’));

**Null**  语义：“存在该数据，但当前还不存在确定的值“

Null在程序中代表“无值“或者”无对象“

可以通过给一个变量赋值null来清除变量的内容

Var president = null;

Var emp = new Object();

Emp.manage = null;

Undefined 语义：“不存在该数据“，声明了变量但从未赋值，或者对象属性不存在

Var msg;

Document.write(msg); //undefined

Var emp = new object();

Document.write(emp.isOnSale); //undefined

## 8、运算符和表达式

### 1、算数运算符： 算术运算符用于执行变量与/或值之间的算术运算。

给定 y=5，下面的表格解释了这些算术运算符：

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **描述** | **例子** | **结果** |
| + | 加 | x=y+2 | x=7 |
| - | 减 | x=y-2 | x=3 |
| \* | 乘 | x=y\*2 | x=10 |
| / | 除 | x=y/2 | x=2.5 |
| % | 求余数 (保留整数) | x=y%2 | x=1 |
| ++ | 累加 | x=++y | x=6 |
| -- | 递减 | x=--y | x=4 |

### 2、JavaScript 赋值运算符

赋值运算符用于给 JavaScript 变量赋值。

给定 x=10 和 y=5，下面的表格解释了赋值运算符：

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **例子** | **等价于** | **结果** |
| = | x=y |  | x=5 |
| += | x+=y | x=x+y | x=15 |
| -= | x-=y | x=x-y | x=5 |
| \*= | x\*=y | x=x\*y | x=50 |
| /= | x/=y | x=x/y | x=2 |
| %= | x%=y | x=x%y | x=0 |

### 3、比较运算符：在逻辑语句中使用，以测定变量或值是否相等。

给定 x=5，下面的表格解释了比较运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **例子** |
| == | 等于 | x==8 为 false |
| === | 全等（值和类型） | x===5 为 true；x==="5" 为 false |
| != | 不等于 | x!=8 为 true |
| > | 大于 | x>8 为 false |
| < | 小于 | x<8 为 true |
| >= | 大于或等于 | x>=8 为 false |
| <= | 小于或等于 | x<=8 为 true |

**如何使用**

可以在条件语句中使用比较运算符对值进行比较，然后根据结果来采取行动：

if (age<18) document.write("Too young");

### 4、逻辑运算符

逻辑运算符用于测定变量或值之间的逻辑。

给定 x=6 以及 y=3，下表解释了逻辑运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **例子** |
| && | and | (x < 10 && y > 1) 为 true |
| || | or | (x==5 || y==5) 为 false |
| ! | not | !(x==y) 为 true |

解释：

||或者

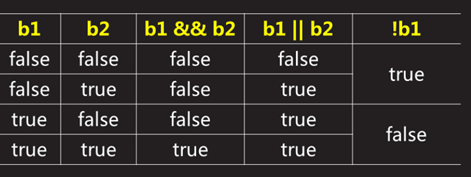
表达式1 || 表达式2

只要有一个是真，结果就是真 并联电路

&& 并且

表达式1 && 表达式2

两个都为真，结果才是真 串联电路



### 5、条件运算符 （三目运算符）

表达式1？表达式2：表达式3

表达式1一应会返回一个布尔值

表达式2、3可以是字符串或数字或逻辑

表达式1为真，取2的值，否则取3的值

JavaScript 还包含了基于某些条件对变量进行赋值的条件运算符。

语法

variablename=(条件语句)?value1:value2

例子：

Var age = 20;

Var msg = age>18? “成年人”：”未成年人”；

Var a =10;

Var b =20;

Var max = a>b? a:b;

如果变量 visitor 中的值是 "PRES"，则向变量 greeting 赋值 "Dear President "，否则赋值 "Dear"。

### 6、用于字符串的 + 运算符

+ 运算符用于把文本值或字符串变量加起来（连接起来）。

如需把两个或多个字符串变量连接起来，请使用 + 运算符。

txt1="What a very";

txt2="nice day";

txt3=txt1+txt2;

在以上语句执行后，变量 txt3 包含的值是 "What a verynice day"。

要想在两个字符串之间增加空格，需要把空格插入一个字符串之中：

txt1="What a very ";

txt2="nice day";

txt3=txt1+txt2;

或者把空格插入表达式中：

txt1="What a very";

txt2="nice day";

txt3=txt1+" "+txt2;

在以上语句执行后，变量 txt3 包含的值是：

"What a very nice day"

**对字符串和数字进行加法运算**

请看这些例子：

<html>

<body>

<script type="text/javascript">

x=5+5;

document.write(x);

document.write("<br />");

x="5"+"5";

document.write(x);

document.write("<br />");

x=5+"5";

document.write(x);

document.write("<br />");

x="5"+5;

document.write(x);

document.write("<br />");

</script>

<h3>规则是：</h3>

<p><strong>如果把数字与字符串相加，结果将成为字符串。</strong></p>

</body>

</html>

## 数组Array 对象

**数组（Array）**

Array 类型是 ECMAScript 最常用的类型。而且 ECMAScript 中 的 Array 类型和其他语言中的数组有着很大的区别。虽然数组都是有序排列，但 ECMAScript 中的数组每个元素可以保存任何类型。ECMAScript 中数组的大小也是可以调整的。

创建 Array 类型有两种方式：第一种是 new 运算符，第二种是字面量。

**之前讲过变量**： var a=100; var b=50; 存储一个数字——盒子

**数组**：连续的存储单元——柜子。数组的长度，存多少个？

语法：

var 数组名= new Array（长度）;

**1、使用 new 关键字创建数组**

Var arr = new Array(); //创建了一个数组

Var arr = new Array(10); //创建一个包含 10 个元素的数组

Var arr = new Array('李易峰',28,'教师','盐城'); //创建一个数组并分配好了元素

获取数组里面的值，通过下标：（index）门牌号--标示数组的每一个存储区域。

下标数字：从0开始，往上递增1

第一种写法：

var arr = new Array(5);

arr[0]="stevejobs";

arr[1]=18;

arr[2]=true; //逻辑值

arr[3]="tom";

arr[4]="1351351356678";

Arr[6]=11; //错误，没有下标6，报错：下标越界。

//alert(arr[6]);

第二种写法：

//声明方式，直接将数据存放到数组中

var arr2 = new Array(

"stevejobs",18,true,"tom","1351351356678");

//alert(arr2[3]);

循环将数组中所有的内容输出：

for(i=0;i<arr2.length;i++){

alert(arr[i]);

}

Length属性：表示数组的长度。

使用 length 属性获取数组元素量

alert(box.length) //获取元素个数

box.length= 10; //强制元素个数

box[box.length]='JS 技术'; //通过 length 给数组增加一个元素

push()方法可以接收任意数量的参数，把它们逐个添加到数组的末尾，并返回修改后数 组的长度。而 pop()方法则从数组末尾移除最后一个元素，减少数组的 length 值，然后返回 移除的元素。

var box=['李易峰', 28,'计算机编程']; //字面量声明

alert(box.push('盐城')); //数组末尾添加一个元素，并且返回长度

alert(box); //查看数组

box.pop(); //移除数组末尾元素，并返回移除的元素

alert(box); //查看元素

自定义对象：

JSON方法

对象：1、属性（变量） 2、方法（函数）

Var 变量 = {name：“张三”,age：20,sex：“男”};

var a = {

"name":"张三",

"age":12,

"getNameLength":function(){

return this.name.length;

}

}

alert(a.getNameLength());

创建一个复杂的数组

var box=[

{ //第一个元素是一个对象

name:'李易峰',

age:28,

run:function(){

return 'run 了';

}

},

['马云','李彦宏',new Object()],//第二个元素是数组

'江苏', //第三个元素是字符串

25+25, //第四个元素是数值

new Array(1,2,3) //第五个元素是数组

];

alert(box);

练习：定义一个数组，存储10个学生的成绩，

* 1. 通过操作数组，求出10个学生中的最高分和最低成绩。
  2. 通过操作数组，求出所有学生的平均成绩。

var arr = new Array(

29.32,432.432,65.78,98.65,23.4,56.876,765.45,678.43,4556.76,43.2

);

var min = arr[0];

var max = arr[0];

var add = 0;

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

if(arr[i]<=min){

min=arr[i];

}

if(arr[i]>=max){

max=arr[i];

}

add+=arr[i];

}

var a = add/arr.length;

document.writeln(min+"<br/>");

document.writeln(max+"<br/>");

document.writeln(a);

-----------------------------------------------------------------------

## JavaScript函数

课程概要：

了解函数

定义函数

函数调用

带参数的函数

带返回值的函数

定义函数:将完成某一特定功能的代码集合起来，可以重复使用的代 码块。

优点:

– 使程序更加简洁

– 逻辑性更调理

– 调用更方便

– 维护更加容易

### ****9.1定义函数（方法、过程）的功能****

* 使用关键字function定义一个函数，
* 函数名的定义与标示符一致，大小写敏感
* 函数就是包裹在花括号中的代码块
* 函数定义语法：

function 函数名()

{

这里是要执行的代码

}

当调用该函数时，会执行函数内的代码。

例如：

<script>

function myFunction()

{

alert("Hello World!");

}

</script>

<button onclick="myFunction()">点击这里</button>

可以在某事件发生时直接调用函数（比如当用户点击按钮时），并且可由 JavaScript 在任何位置进行调用。

提示：JavaScript 对大小写敏感。关键词 function 必须是小写的，并且必须以与函数名称相同的大小写来调用函数。

### ****9.2函数的调用****

定义好之后不能自动执行，需要调用

调用方式

1）在<script>标签内调用

2）在html文件中调用

<p>点击这个按钮，来调用带参数的函数。</p>

<button onclick="myFunction()">点击这里</button>

<script>

function myFunction()

{

var a = 10;

var b = 10;

var sum = a + b;

alert(sum);

}

/\*myFunction();\*/ /\*在<script>标签内调用\*/

练习：

将输出五行五列的表格代码，封装成方法。调用一次就输出一个5行5列的表格。

### 9.3定义参数和返回值

在调用函数时，您可以向其传递值，这些值被称为参数。

这些参数可以在函数中使用。您可以发送任意多的参数，由逗号 (,) 分隔：

myFunction(argument1,argument2)

当您声明函数时，请把参数作为变量来声明：

function myFunction(var1,var2)

{

这里是要执行的代码

}

变量和参数必须以一致的顺序出现。第一个变量就是第一个被传递的参数的给定的值，以此类推。

实例：

<!DOCTYPE html>

<html>

<body>

<p>点击这个按钮，来调用带参数的函数。</p>

<button onclick="myFunction('Bill Gates','CEO')">点击这里</button>

<script>

function myFunction(name,job)

{

alert("Welcome " + name + ", the " + job);

}

</script>

</body>

</html>

上面的函数会当按钮被点击时提示 "Welcome Bill Gates, the CEO"。

函数很灵活，您可以使用不同的参数来调用该函数，这样就会给出不同的消息：

<!DOCTYPE html>

<html>

<body>

<p>请点击其中的一个按钮，来调用带参数的函数。</p>

<button onclick="myFunction('Harry Potter','Wizard')">点击这里</button>

<button onclick="myFunction('Bob','Builder')">点击这里</button>

<script>

function myFunction(name,job)

{

alert("Welcome " + name + ", the " + job);

}

</script>

</body>

</html>

根据您点击的不同的按钮，上面的例子会提示 "Welcome Harry Potter, the Wizard" 或 "Welcome Bob, the Builder"。

**带有返回值的函数**

有时，我们会希望函数将值返回调用它的地方。通过使用 return 语句就可以实现。

在使用 return 语句时，函数会停止执行，并返回指定的值。

语法

function myFunction()

{

var x=5;

return x;

}

上面的函数会返回值 5。

var myVar=myFunction();

myVar 变量的值是 5，也就是函数 "myFunction()" 所返回的值。

即使不把它保存为变量，您也可以使用返回值：

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

"demo" 元素的 innerHTML 将成为 5，也就是函数 "myFunction()" 所返回的值。

您可以使返回值基于传递到函数中的参数：

实例：

计算两个数字的乘积，并返回结果：

<!DOCTYPE html>

<html>

<body>

<p>本例调用的函数会执行一个计算，然后返回结果：</p>

<p id="demo"></p>

<script>

function myFunction(a,b)

{

return a\*b;

}

document.getElementById("demo").innerHTML=myFunction(4,3);

</script>

</body>

</html>

"demo" 元素的 innerHTML 将是：

12

注释：整个 JavaScript 并不会停止执行，仅仅是函数。JavaScript 将继续执行代码，从调用函数的地方。

函数调用将被返回值取代：

带参数练习：

定义一个方法，传递2个整数到该方法，在方法内传递大小，将大的数输出。

**return返回值的练习:**

练习：定义一个函数，传递两个参数，第一个参数为类型type，第二个参数为半径r。

1、传递参数类型为1，半径为5，返回圆的面积。

2、传递参数类型为2，半径为5，返回圆的周长。

-----------------------------------------------------------------------

作业：定义一个函数，传递2个整数参数，分别为number1和number2.

在函数中将number2和number1之间所有的能整除3的数，全部输出。

\*注意：如果number1大于number2，需要输出的是number1到number2之间的数。

### 9.4 函数全部变量和局部变量

**变量作用范围：**

全局（函数外面声明的变量）：哪里都可以使用，从声明的那一行开始到程序结束，都可以用。函数内部也可以调用。

局部（函数里面声明的变量）：只能在函数内部使用，函数外找不到。函数内可以调用全局变量。

var a =100; //全局

a=200;

alert(a); //弹出全局变量a

function fun(){

var b = 100;

a=50;

alert(b); //b ? //弹出局部变量b

}

fun(); //调用函数

function fun2(){

var b=20;

a+=100;

alert(b); //b ?

}

fun2();

alert(b); //错误，不能够在函数外 访问局部变量

----------------------------------------

## 11、JavaScript对象

课程概要：  
 什么是对象

JavaScript 中的所有事物都是对象：

**基本对象**：字符串、数字、boolean

**自定义对象**：

var temp = new Object();

temp.username=”xxx”;

temp.password=”xxx”;

范例：

|  |
| --- |
| <script>  var person = new Object();  person.username="wangmazi";  person.password="nima";  alert(person.username);  </script> |

var x = function (){

this.username=”xxx”;

this.pwd=”xxx”;

}

范例：

|  |
| --- |
| //采用构造函数的形式，没有参数的形式  /\*var x = function(){  this.username = "gugu";  this.pwd="guoer";    }  var xx = new x();  alert(xx.username);\*/ |

|  |
| --- |
| //采用有参数的构造函数的形式  var x1 = function(a1,b1){  this.username=a1;  this.password=b1;  }  var x11 = new x1(2,2);  alert(x11.username+"\t"+x11.password); |

var xx = {username:”xxx”,password:”xxx”}

获取属性值的方式：对象的名字.对象的属性名字，例如：

temp.username

x.username;

xx.username

范例：json格式的形式（后面讲ajax的时候的重点格式）

|  |
| --- |
| var x12 = {username:"aaa",password:"bbb"};  alert(x12.username); |

引用/内置对象：数组、函数、还有日期、string、math，最后三个是js里面的常用类，我们在后面讲解。

Var array = new Array(“a”,”b”,”c”,”……”);

Var x = new Array();x[0]=”1”;

## Array 对象属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| **[constructor](http://www.w3school.com.cn/jsref/jsref_constructor_array.asp)** | 返回对创建此对象的数组函数的引用。 |
| **[length](http://www.w3school.com.cn/jsref/jsref_length_array.asp)** | 设置或返回数组中元素的数目。 |
| **[prototype](http://www.w3school.com.cn/jsref/jsref_prototype_array.asp)** | 使您有能力向对象添加属性和方法。 |

## Array 对象方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| **[concat()](http://www.w3school.com.cn/jsref/jsref_concat_array.asp)** | 连接两个或更多的数组，并返回结果。 |
| **[join()](http://www.w3school.com.cn/jsref/jsref_join.asp)** | 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。 |
| **[pop()](http://www.w3school.com.cn/jsref/jsref_pop.asp)** | 删除并返回数组的最后一个元素 |
| **[push()](http://www.w3school.com.cn/jsref/jsref_push.asp)** | 向数组的末尾添加一个或更多元素，并返回新的长度。 |
| **[reverse()](http://www.w3school.com.cn/jsref/jsref_reverse.asp)** | 颠倒数组中元素的顺序。 |
| **[shift()](http://www.w3school.com.cn/jsref/jsref_shift.asp)** | 删除并返回数组的第一个元素 |
| **[slice()](http://www.w3school.com.cn/jsref/jsref_slice_array.asp)** | 从某个已有的数组返回选定的元素 |
| **[sort()](http://www.w3school.com.cn/jsref/jsref_sort.asp)** | 对数组的元素进行排序 |
| **[splice()](http://www.w3school.com.cn/jsref/jsref_splice.asp)** | 删除元素，并向数组添加新元素。 |
| **[toSource()](http://www.w3school.com.cn/jsref/jsref_tosource_array.asp)** | 返回该对象的源代码。 |
| **[toString()](http://www.w3school.com.cn/jsref/jsref_toString_array.asp)** | 把数组转换为字符串，并返回结果。 |
| **[toLocaleString()](http://www.w3school.com.cn/jsref/jsref_toLocaleString_array.asp)** | 把数组转换为本地数组，并返回结果。 |
| **[unshift()](http://www.w3school.com.cn/jsref/jsref_unshift.asp)** | 向数组的开头添加一个或更多元素，并返回新的长度。 |
| **[valueOf()](http://www.w3school.com.cn/jsref/jsref_valueof_array.asp)** | 返回数组对象的原始值 |

范例：

|  |
| --- |
| var x11 = new x1(2,2);  alert(x11.username+"\t"+x11.password);\*/    /\*var x12 = {username:"aaa",password:"bbb"};  alert(x12.username);\*/  //var arr = new Array("b","a","b","c","d");  //alert(arr.length)  //alert(arr.valueOf());  //alert(arr.sort());    //var arr = new Array("你2","你11","我你","我你他","你");  //alert(arr.length)  //alert(arr.valueOf());  //alert(arr.sort());      //reverse()    var arr = new Array("b","a","b","c","d");  alert(arr.reverse()); |

对象是拥有属性和方法的数据

Javascript允许自定义对象

# **12、JavaScript事件**

JavaScript 事件是由访问 Web 页面的用户引起的一系列操作，例如：用户点击。当用户执行某些操作的时候，再去执行一系列代码。

JavaScript用户——>触发——>事件驱动——>调用函数

* 1. **事件介绍**

事件一般是用于浏览器和用户操作进行交互。最早是 IE 和 NetscapeNavigator 中出现， 作为分担服务器端运算负载的一种手段。直到几乎所有的浏览器都支持事件处理。而 DOM2 级规范开始尝试以一种复合逻辑的方式标准化 DOM 事件。IE9、Firefox、Opera、Safari 和 Chrome 全都已经实现了“DOM2 级事件”模块的核心部分。IE8 之前浏览器仍然使用其专 有事件模型。 JavaScript 有三种事件模型：内联模型、脚本模型和 DOM2 模型。

* 1. **内联模型**

这种模型是最传统接单的一种处理事件的方法。在内联模型中，事件处理函数是 HTML 标签的一个属性，用于处理指定事件。虽然内联在早期使用较多，但它是和 HTML 混写的， 并没有与 HTML 分离。

//在 HTML 中把事件处理函数作为属性执行 JS 代码

<input type="button" value="按钮" onclick="alert('Lee');"/> //注意单双引号

//在 HTML 中把事件处理函数作为属性执行 JS 函数

<input type="button" value="按钮"onclick="box();" /> //执行 JS 的函数

PS：函数不得放到 window.onload 里面，这样就看不见了。

* 1. **脚本模型**

由于内联模型违反了 HTML 与 JavaScript 代码层次分离的原则。为了解决这个问题，我 们可以在 JavaScript 中处理事件。这种处理方式就是脚本模型。

Var input=document.getElementsByTagName('input')[0]; //得到 input 对象

input.onclick=function(){ //匿名函数执行

alert('Lee');

};

PS：通过匿名函数，可以直接触发对应的代码。也可以通过指定的函数名赋值的方式，来执行函数(赋值的函数名不要跟着括号)。

input.onclick=box; //把函数名赋值给事件处理函数 区别调用函数

//把函数绑定到input.onclick单击事件上，点击是执行box这个函数，而不是绑定时去调用box函数

input.onclick=box（）；//box（）函数的返回值赋给input.onclick

* 1. **事件处理函数**

JavaScript 可以处理的事件类型为：鼠标事件、键盘事件、HTML 事件。

|  |  |  |
| --- | --- | --- |
| 事件处理函数 | 影响的元素 | 何时发生 |
| 鼠标事件： |  |  |
| onclick | 链接、按钮、表单对象、图像映射区域 | 当用户单击对象时 |
| ondblclick | 链接、按钮、表单对象 | 当用户双击对象时 |
| onmousedown |  | 鼠标某个键被按下还未弹起时 |
| onmouseup |  | 某个鼠标按键被松开 |
| onmouseover | 链接 | 鼠标移动到某元素上 |
| onmouseout | 链接 | 鼠标从某元素上移开 |
| onmousemove |  | 鼠标被移动 |
| 键盘事件： |  |  |
| onkeydown | 文档、图像、链接、表单 | 当按键被按下时 |
| onkeypress | 文档、图像、链接、表单 | 当按键被按下然后松开时 |
| onkeyup | 文档、图像、链接、表单 | 当按键被松开时 |
| HTML|表单|事件： |  |  |
| onload | 主题、框架集、图像 | 加载页面时 |
| onunload | 主体、框架集 | 退出页面时 |
| onselect | 表单元素 | 当选择一个表单对象时 |
| onchange | 输入框，选择框和文本区域 | **当改变一个元素的值且失去焦点时** |
| onfocus | 窗口、框架、所有表单对象 | 当单击鼠标或者将鼠标移动聚焦到窗口或框架时 |
| onblur | 窗口、框架、所有表单对象 | 当焦点从对象上移开时 |
| onsubmit | 表单 | 当发送表格到服务器时 |
| onreset | 表单复位按钮 | 单击表单的 reset按钮 |
| onresize | 窗口 | 当选择一个表单对象时 |
| onscroll |  | 当用户滚动带滚动条的元素时触发 |
| onabort | 图像 | 当图像加载被中断时 |
| ondragdrop | 窗口 | 当用户将一个对象拖放到浏览器窗口时 |
| onError | 脚本 | 当脚本中发生语法错误时 |
| onmove | 窗口 | 当浏览器窗口移动时 |

PS：所有的事件处理函数都会都有两个部分组成，on+ 事件名称，例如 click 事件的事件处理函数就是：onclick。

范例：

Onload范例：

第一种方法：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script>  function sdxl(){  alert("d");  }    </script>  </head>  <body onload="sdxl()">  </body>  </html> |

第二种方法：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  <script>  /\*function sdxl(){  alert("d");  }\*/  window.onload=function(){  alert("aaa");  }  </script>  </head>  <body>  </body>  </html> |

**获取键值：**

event.keycode 键盘值   
events.button==0 鼠标左键   
events.button==1 鼠标中键  
events.button==2 鼠标右键   
鼠标的键值在IE与其他浏览器中有所不同，查看详情：

<http://www.jb51.net/article/26760.htm>

获取键盘键值：

document.onkeydown = function (e){

alert(e.keyCode);

}

获取鼠标键值：

document.onmousedown = function (e){

var e = window.event || e;

var v = e.button;

if(v==0){

alert("左");

}else if(v==2){

alert("右");

}else if(v==1){

alert("中间");

}

}

---------------------------------------------------------------

---------------------------------------------------------------------

Onchange 内容改变 //改变文本弹出提示

Onfocus 获得焦点 //提示：请输入您的用户名

Onblur 失去焦点 //提示：密码框不能为空，确认密码

Onload 页面加载事件 //写在body标签里

Onunload 页面撤销事件

Onkeydowm 按下键盘某个键事件

Onkeypress 按住键盘某个键

Onkeyup 松开键盘某个键

---------------------------------------------------------------------

对于每一个事件，它都有自己的触发范围和方式，如果超出了触发范围和方式，事件处 理将失效。

**通过获取id得到对象和对象的属性：**

document.getElementById(“id名称”);

document.getElementById(“id名称”).value;

**通过传递this获取对象和对象的属性：**

function demo(o){

o.value;

}

<button onclick = “demo（this）“></button>

demo(this.value) this是一个对象，就是当前这个标记。一个标记作为一个对象。

.是一个引用符 .属性

========================================================

练习：

<script type="text/javascript">

function fun(){

var a = document.getElementById("txt");

a.value+="$";

if(a.value.length>5){

a.value="$";

}

}

</script>

<input type="button" onclick="fun()" value="$" id="txt"/>

按钮颜色变换案例：1、点击替换两个图片。2、点击替换三个图片

按钮颜色变换案例：

var b=true;

function XXXX(){

if(b){

XXXX.src=""; //元素的属性改变

b=false;

}else{

XXX.src="";

b=true;

}

}

练习\_鼠标事件:

下拉菜单

练习\_键盘事件:

台球移动

onkeypress 和 onkeydown 是有区别，下面将讲解 onkeypress 与 onkeydown 事件的区别。   
onkeypress 事件在用户按下并放开任何字母数字键时发生。但是系统按钮（例如：箭头键、功能键）无法得到识别。   
onkeydown 事件在用户按下任何键盘键（包括系统按钮）时发生。   
**具体区别**：   
1. 一个放开一个没有放开，onkeydown 先于 onkeypress 发生。   
2.onkeypress 无法系统按钮。   
2.onkeydown 捕获的 keyCode 不区分字母大小，而 onkeypress 区分。   
  
onkeypress是在用户按下并放开任何字母数字键时发生。系统按钮（例如，箭头键和功能键）无法得到识别。   
onkeyup 是在用户放开任何先前按下的键盘键时发生。   
onkeydown 是在用户按下任何键盘键（包括系统按钮，如箭头键和功能键）时发生。

练习\_表单事件:

验证表单(注册页面)

## 13、流程控制语句

学习要点：

1.语句的定义

2.if 语句

3.switch 语句

4.do...while 语句

5.while 语句

6.for 语句

8.break 和 continue 语句

9.with 语句

## 13.1 条件语句

通常在写代码时，您总是需要为不同的决定来执行不同的动作。您可以在代码中使用条件语句来完成该任务。在 JavaScript 中，我们可使用以下条件语句：

* **if 语句** - 只有当指定条件为 true 时，使用该语句来执行代码
* **if...else 语句** - 当条件为 true 时执行代码，当条件为 false 时执行其他代码
* **if...else if....else 语句**- 使用该语句来选择多个代码块之一来执行
* **switch 语句** - 使用该语句来选择多个代码块之一来执行

### If 语句

只有当指定条件为 true 时，该语句才会执行代码。

语法:

语句 0；

if (逻辑表达式)  
  {  
  语句1； //当条件为 true 时执行的代码

语句 2；  
  }

语句3；

1. 执行语句0；
2. 判断逻辑表达式的值；

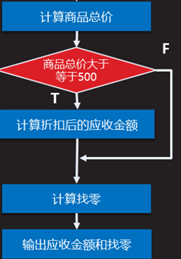
若值为true,则执行if语句块中的语句；

若值为flase,则不执行if语句块中的语句；

3.执行语句3；

请使用小写的 **if**。使用大写字母（IF）会生成 JavaScript 错误！

案例：购买商品是总价满500时候打7折，不满500时不打折。



<!DOCTYPE html>

<html>

<body>

<p id="demo"></p>

<script>

var x=400;

var y;

if(x>500){

y=0.7\*x;

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

}

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

</script>

</body>

</html>

### If...else 语句

请使用 if....else 语句在条件为 true 时执行代码，在条件为 false 时执行其他代码。

语法:

if (条件)  
  {  
  当条件为 true 时执行的代码  
  }  
else

{

 当条件不为 true 时执行的代码  
  }

**实例**：学生成绩60分以上考试及格，60分一下考试不及格；

<!DOCTYPE html>

<html>

<body>

<input type="text" id="demo" value="" placeholder="请输入成绩"/>

<input type="button" value="点击" onclick="tj()">

<script>

function tj(){

var grade=document.getElementById("demo").value;

if(grade>60){

alert("你的成绩是："+grade+"结果是：及格");

}else{

alert("考试不合格");

}

}

</script>

</body>

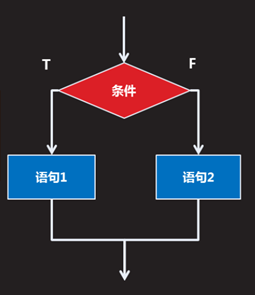
</html>

### If...else if...else 语句

使用 if....else if...else 语句来选择多个代码块之一来执行。

语法

if (条件1)  
  {  
  当条件 1 为 true 时执行的代码  
  }  
else if (条件2)  
  {  
 当条件 2 为 true 时执行的代码  
  }  
else  
  {  
  当条件 1 和 条件 2 都不为 true 时执行的代码  
  }



.if (条件表达式) { 语句;} else if (条件表达式) {语句;} ... else { 语句;} var box = 100;

if (box >= 100) { //如果满足条件，不会执行下面任何分支 alert('甲');

} else if (box >= 90) { alert('乙');

} else if (box >= 80) { alert('丙');

} else if (box >= 70) { alert('丁');

} else if (box >= 60) {

alert('及格');

} else { //如果以上都不满足，则输出不及格 alert('不及格');

}

## 13.2 **switch语句**

请使用 switch 语句来选择要执行的多个代码块之一。

**switch 语法**

switch(n)  
{  
case 1:  
  执行代码块 1  
break;  
case 2:  
  执行代码块 2  
break;  
default:  
 n 与 case 1 和 case 2 不同时执行的代码  
}

工作原理：首先设置表达式 *n*（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 **break** 来阻止代码自动地向下一个 case 运行。

<!DOCTYPE html>

<html>

<body>

<p>点击下面的按钮来显示今天是周几：</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo"></p>

<script>

function myFunction()

{

var x;

var d=new Date().getDay();

switch (d)

{

case 0:

x="Today it's Sunday";

break;

case 1:

x="Today it's Monday";

break;

case 2:

x="Today it's Tuesday";

break;

case 3:

x="Today it's Wednesday";

break;

case 4:

x="Today it's Thursday";

break;

case 5:

x="Today it's Friday";

break;

case 6:

x="Today it's Saturday";

break;

}

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

}

</script>

</body>

</html>

**default 关键词**

请使用 default 关键词来规定匹配不存在时做的事情：

<html>

<body>

<p>点击下面的按钮，会显示出基于今日日期的消息：</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo"></p>

<script>

function myFunction()

{

var x;

var d=new Date().getDay();

switch (d)

{

case 6:

x="Today it's Saturday";

break;

case 0:

x="Today it's Sunday";

break;

default:

x="Looking forward to the Weekend";

}

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

}

</script>

</body>

</html>

## 13.3 **for循环**

            如果您希望一遍又一遍地运行相同的代码，并且每次的值都不同，那么使用循环是很方便的。

for循环语法：

for (语句 1; 语句 2; 语句 3)  
  {  
  被执行的代码块  
  }

语句 1 （代码块）开始前执行 starts.

语句 2 定义运行循环（代码块）的条件

语句 3 在循环（代码块）已被执行之后执行

实例：

<!DOCTYPE html>

<html>

<body>

<p>点击按钮循环代码5次。</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo"></p>

<script>

function myFunction()

{

var x="";

for (var i=0;i<5;i++)

{

x=x + "The number is " + i + "<br>";

}

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

}

</script>

</body>

</html>

For循环实现九九乘法表

for(var i=1;i<=9;i++){

// alert(i);

for(var j=1;j<=i;j++){

document.write(j+"\*"+i+"="+i\*j+"\t");

}

document.write("<br/>");

}

## 13.4 while循环

只要指定条件为 true，循环就可以一直执行代码块。

while 循环会在指定条件为真时循环执行代码块。

语法

while (条件)  
  {  
  需要执行的代码  
  }

实例

本例中的循环将继续运行，只要变量 i 小于 5：

<!DOCTYPE html>

<html>

<body>

<p>点击下面的按钮，只要 i 小于 5 就一直循环代码块。</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo"></p>

<script>

function myFunction()

{

var x="",i=0;

while (i<5)

{

x=x + "The number is " + i + "<br>";

i++;

}

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

}

</script>

</body>

</html>

## 13.5 do/while 循环

do/while 循环是 while 循环的变体。该循环会在检查条件是否为真之前执行一次代码块，然后如果条件为真的话，就会重复这个循环。

**语法**

do  
  {  
  需要执行的代码  
  }  
while (条件);

**实例:**

下面的例子使用 do/while 循环。该循环至少会执行一次，即使条件为 false 它也会执行一次，因为代码块会在条件被测试前执行：

<!DOCTYPE html>

<html>

<body>

<p>点击下面的按钮，只要 i 小于 5 就一直循环代码块。</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo"></p>

<script>

function myFunction()

{

var x="",i=0;

do

{

x=x + "The number is " + i + "<br>";

i++;

}

while (i<5)

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

}

</script>

</body>

</html>

## 13.6 比较 for 和 while

如果您已经阅读了前面那一章关于 for 循环的内容，您会发现 while 循环与 for 循环很像。

案例：**从1加到100**

<!--实现1到100相加-->

<!--<script>

var y=0;

for(var i=1;i<=100;i++){

y+=i;

}

document.write(y);

</script>-->

<!--实现1到100相加-->

<script>

var i =0;

var y =0;

while(i<=100){

y+=i;

i++;

}

document.write(y);

</script>

## 14、JavaScript break和continue语句

break 语句用于跳出循环。

continue 用于跳过循环中的一个迭代。

### Break 语句

我们已经在本教程之前的章节中见到过 break 语句。它用于跳出 switch() 语句。

break 语句可用于跳出循环。

break 语句跳出循环后，会继续执行该循环之后的代码（如果有的话）：

<!DOCTYPE html>

<html>

<body>

<p>点击按钮，测试带有 break 语句的循环。</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo"></p>

<script>

function myFunction()

{

var x="",i=0;

for (i=0;i<10;i++)

{

if (i==3)

{

break;

}

x=x + "The number is " + i + "<br>";

}

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

}

</script>

</body>

</html>

### Continue 语句

**continue 语句**中断循环中的迭代，如果出现了指定的条件，然后继续循环中的下一个迭代。 该例子跳过了值 3：

<!DOCTYPE html>

<html>

<body>

<p>点击下面的按钮来执行循环，该循环会跳过 i=3 的步进。</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo"></p>

<script>

function myFunction()

{

var x="",i=0;

for (i=0;i<10;i++)

{

if (i==3)

{

continue;

}

x=x + "The number is " + i + "<br>";

}

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

}

</script>

</body>

</html>

**JavaScript 标签**

正如您在 switch 语句那一章中看到的，可以对 JavaScript 语句进行标记。

如需标记 JavaScript 语句，请在语句之前加上冒号：

label:  
statements

break 和 continue 语句仅仅是能够跳出代码块的语句。

语法:

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：

<!DOCTYPE html>

<html>

<body>

<script>

cars=["BMW","Volvo","Saab","Ford"];

list:

{

document.write(cars[0] + "<br>");

document.write(cars[1] + "<br>");

document.write(cars[2] + "<br>");

break list;

document.write(cars[3] + "<br>");

document.write(cars[4] + "<br>");

document.write(cars[5] + "<br>");

}

</script>

</body>

</html>

## 15、JavaScript typeof

### typeof 操作符

你可以使用 typeof 操作符来检测变量的数据类型。

<!DOCTYPE html>

<html>

<body>

<p> typeof 操作符返回变量或表达式的类型。</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

typeof "john" + "<br>" +

typeof 3.14 + "<br>" +

typeof false + "<br>" +

typeof [1,2,3,4] + "<br>" +

typeof {name:'john', age:34};

</script>

</body>

</html>

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。

### Null

在 JavaScript 中 null 表示 "什么都没有"。

null是一个只有一个值的特殊类型。表示一个空对象引用。

用 typeof 检测 null 返回是object。

<!DOCTYPE html>

<html>

<body>

<p>对象可以通过设置为 <b>null</b> 来清空。</p>

<p id="demo"></p>

<script>

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

var person = null;

document.getElementById("demo").innerHTML = typeof person;

</script>

</body>

</html>

你可以设置为 undefined 来清空对象:

<!DOCTYPE html>

<html>

<body>

<p>对象可以设置为 <b>undefined</b> 来清空。</p>

<p id="demo"></p>

<script>

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

var person = undefined;

document.getElementById("demo").innerHTML = typeof person;

</script>

</body>

</html>

### Undefined

在 JavaScript 中, **undefined** 是一个没有设置值的变量。

**typeof** 一个没有值的变量会返回 **undefined**。

<!DOCTYPE html>

<html>

<body>

<p>变量的值如果不存在则该变量值为 <b>undefined</b>。</p>

<p id="demo"></p>

<script>

var person;

document.getElementById("demo").innerHTML =

person + "<br>" + typeof person;

</script>

</body>

</html>

任何变量都可以通过设置值为 **undefined** 来清空。 类型为 **undefined**.

<!DOCTYPE html>

<html>

<body>

<p>变量可以通过设置 <b>undefined</b> 来清空。</p>

<p id="demo"></p>

<script>

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

var person = undefined;

document.getElementById("demo").innerHTML =

person + "<br>" + typeof person;

</script>

</body>

</html>

## 16、JavaScript 类型转换

### JavaScript 类型转换

Number() 转换为数字， String() 转换为字符串， Boolean() 转化为布尔值。

### JavaScript 数据类型

在 JavaScript 中有 5 中不同的数据类型：

* string
* number
* boolean
* object
* function

3 种对象类型：

* Object
* Date
* Array

2 个不包含任何值的数据类型：

* null
* undefined

### typeof 操作符

你可以使用 **typeof** 操作符来查看 JavaScript 变量的数据类型。

<!DOCTYPE html>

<html>

<body>

<p> typeof 操作符返回变量、对象、函数、表达式的类型。</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

typeof "john" + "<br>" +

typeof 3.14 + "<br>" +

typeof NaN + "<br>" +

typeof false + "<br>" +

typeof [1,2,3,4] + "<br>" +

typeof {name:'john', age:34} + "<br>" +

typeof new Date() + "<br>" +

typeof function () {} + "<br>" +

typeof myCar + "<br>" +

typeof null;

</script>

</body>

</html>

请注意：

* NaN 的数据类型是 number
* 数组(Array)的数据类型是 object
* 日期(Date)的数据类型为 object
* null 的数据类型是 object
* 未定义变量的数据类型为 undefined

如果对象是 JavaScript Array 或 JavaScript Date ，我们就无法通过 **typeof** 来判断他们的类型，因为都是 返回 Object。

### JavaScript 类型转换

JavaScript 变量可以转换为新变量或其他数据类型：

* 通过使用 JavaScript 函数
* 通过 JavaScript 自身自动转换

### 将数字转换为字符串

全局方法 **String()** 可以将数字转换为字符串。

该方法可用于任何类型的数字，字母，变量，表达式：

<!DOCTYPE html>

<html>

<body>

<p> String() 方法可以将数字转换为字符串。</p>

<p id="demo"></p>

<script>

var x = 123;

document.getElementById("demo").innerHTML =

String(x) + "<br>" +

String(123) + "<br>" +

String(100 + 23);

</script>

</body>

</html>

Number 方法 **toString()** 也是有同样的效果。

<!DOCTYPE html>

<html>

<body>

<p>toString() 方法将数字转换为字符串。</p>

<p id="demo"></p>

<script>

var x = 123;

document.getElementById("demo").innerHTML =

x.toString() + "<br>" +

(123).toString() + "<br>" +

(100 + 23).toString();

</script>

</body>

</html>

在 [Number 方法](http://www.runoob.com/jsref/jsref-obj-number.html) 章节中，你可以找到更多数字转换为字符串的方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| toExponential() | 把对象的值转换为指数计数法。 |
| toFixed() | 把数字转换为字符串，结果的小数点后有指定位数的数字。 |
| toPrecision() | 把数字格式化为指定的长度。 |

### 将布尔值转换为字符串

全局方法 **String()** 可以将布尔值转换为字符串。

String(false)        // 返回 "false"  
 String(true)         // 返回 "true"

Boolean 方法 **toString()** 也有相同的效果。

false.toString()     // 返回 "false"  
 true.toString()      // 返回 "true"

### 将日期转换为字符串

全局方法 **String()** 可以将日期转换为字符串。

String(Date())      // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

Date 方法 **toString()** 也有相同的效果。

Date().toString()   // 返回 Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

在 [Date 方法](http://www.runoob.com/jsref/jsref-obj-date.html) 章节中，你可以查看更多关于日期转换为字符串的函数：

|  |  |
| --- | --- |
| **方法** | **描述** |
| getDate() | 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 |
| getDay() | 从 Date 对象返回一周中的某一天 (0 ~ 6)。 |
| getFullYear() | 从 Date 对象以四位数字返回年份。 |
| getHours() | 返回 Date 对象的小时 (0 ~ 23)。 |
| getMilliseconds() | 返回 Date 对象的毫秒(0 ~ 999)。 |
| getMinutes() | 返回 Date 对象的分钟 (0 ~ 59)。 |
| getMonth() | 从 Date 对象返回月份 (0 ~ 11)。 |
| getSeconds() | 返回 Date 对象的秒数 (0 ~ 59)。 |
| getTime() | 返回 1970 年 1 月 1 日至今的毫秒数。 |

### 将字符串转换为数字

全局方法 **Number()** 可以将字符串转换为数字。

字符串包含数字(如 "3.14") 转换为数字 (如 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)。

Number("3.14")    // 返回 3.14  
Number(" ")       // 返回 0   
Number("")        // 返回 0  
Number("99 88")   // 返回 NaN

在 [Number 方法](http://www.runoob.com/jsref/jsref-obj-number.html) 章节中，你可以查看到更多关于字符串转为数字的方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| parseFloat() | 解析一个字符串，并返回一个浮点数。 |
| parseInt() | 解析一个字符串，并返回一个整数。 |

### 将布尔值转换为数字

全局方法 **Number()** 可将布尔值转换为数字。

Number(false)     // 返回 0  
Number(true)      // 返回 1

### 将日期转换为数字

全局方法 **Number()** 可将日期转换为数字。

d = new Date();  
Number(d)          // 返回 1404568027739

日期方法 **getTime()** 也有相同的效果。

d = new Date();  
d.getTime()        // 返回 1404568027739

### 自动转换类型 Type Conversion

当 JavaScript 尝试操作一个 "错误" 的数据类型时，会自动转换为 "正确" 的数据类型。

以下输出结果不是你所期望的：

5 + null    // 返回 5         because null is converted to 0  
"5" + null  // 返回"5null"   because null is converted to "null"  
"5" + 1     // 返回 "51"      because 1 is converted to "1"    
"5" - 1     // 返回 4         because "5" is converted to 5

### 自动转换为字符串

当你尝试输出一个对象或一个变量时 JavaScript 会自动调用变量的 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+0200"

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"

## 19、JavaScript 保留关键字

Javascript 的保留关键字不可以用作变量、标签或者函数名。有些保留关键字是作为 Javascript 以后扩展使用。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | arguments | boolean | break | byte |
| case | catch | char | class\* | const |
| continue | debugger | default | delete | do |
| double | else | enum\* | eval | export\* |
| extends\* | false | final | finally | float |
| for | function | goto | if | implements |
| import\* | in | instanceof | int | interface |
| let | long | native | new | null |
| package | private | protected | public | return |
| short | static | super\* | switch | synchronized |
| this | throw | throws | transient | true |
| try | typeof | var | void | volatile |
| while | with | yield |  |  |

## 20、JavaScript 代码规范

变量名推荐使用驼峰法来命名(**camelCase**)

通常运算符 ( = + - \* / ) 前后需要添加空格

通常使用 4 个空格符号来缩进代码块

简单语句的通用规则:

* 一条语句通常以符号作为结束符。

复杂语句的通用规则:

* 将左花括号放在第一行的结尾。
* 左花括号前添加一空格。
* 将右花括号独立放在一行。
* 不要以分号结束一个复杂的声明。

函数:

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

循环:

for (i = 0; i < 5; i++) {  
    x += i;  
}

条件语句:

if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

对象规则

对象定义的规则:

* 将左花括号与类名放在同一行。
* 冒号与属性值间有个空格。
* 字符串使用双引号，数字不需要。
* 最后一个属性-值对后面不要添加逗号。
* 将右花括号独立放在一行，并以符号作为结束符号。

**实例:**

var person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};

短的对象代码可以直接写成一行:

**实例:**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

**每行代码字符小于 80**

为了便于阅读每行字符建议小于数 80 个。

如果一个 JavaScript 语句超过了 80 个字符，建议在 运算符或者逗号后换行。

**实例:**

document.getElementById("demo").innerHTML =  
    "Hello Runoob.";

**命名规则**

一般很多代码语言的命名规则都是类似的，例如:

* 变量和函数为驼峰法（ **camelCase**）
* 全局变量为大写 (**UPPERCASE** )
* 常量 (如 PI) 为大写 (**UPPERCASE** )

变量命名你是否使用这几种规则： **hyp-hens**, **camelCase**, 或 **under\_scores** ?

**HTML 和 CSS 的横杆(-)字符:**

HTML5 属性可以以 data- (如：data-quantity, data-price) 作为前缀。

CSS 使用 - 来连接属性名 (font-size)。

|  |  |
| --- | --- |
| **IMG_256** | - 通常在 JavaScript 中被认为是减法，所以不允许使用。 |

**下划线:**

很多程序员比较喜欢使用下划线(如：date\_of\_birth), 特别是在 SQL 数据库中。

PHP 语言通常都使用下划线。

**帕斯卡拼写法(PascalCase):**

帕斯卡拼写法(PascalCase) 在 C 语言中语言较多。

驼峰法：

JavaScript 中通常推荐使用驼峰法，jQuery 及其他 JavaScript 库都使用驼峰法。

|  |  |
| --- | --- |
| **IMG_257** | 变量名不要以 $ 作为开始标记，会与很多 JavaScript 库冲突。 |

HTML 载入外部 JavaScript 文件

使用简洁的格式载入 JavaScript 文件 ( type 属性不是必须的):

<script src="myscript.js">

**使用 JavaScript 访问 HTML 元素**

一个糟糕的 HTML 格式可能会导致 JavaScript 执行错误。

以下两个 JavaScript 语句会输出不同结果:

实例

var obj = getElementById("Demo")  
  
var obj = getElementById("demo")

**文件扩展名**

HTML 文件后缀可以是 **.html** (或r **.htm**)。

CSS 文件后缀是 **.css** 。

JavaScript 文件后缀是 **.js** 。

## 使用小写文件名

大多 Web 服务器 (Apache, Unix) 对大小写敏感： london.jpg 不能通过 London.jpg 访问。

其他 Web 服务器 (Microsoft, IIS) 对大小写不敏感： london.jpg 可以通过 London.jpg 或 london.jpg 访问。

你必须保持统一的风格，我们建议统一使用小写的文件名。

# JavaScript html Dom

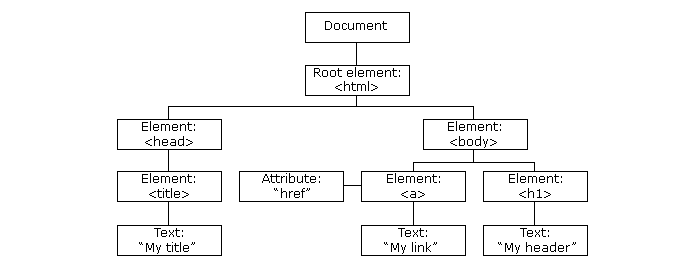
# DOM简介

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

## HTML DOM (文档对象模型)

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）。

**H**TML DOM** 模型被构造为**对象**的树**：



通过可编程的对象模型，JavaScript 获得了足够的能力来创建动态的 HTML。

* JavaScript 能够改变页面中的所有 HTML 元素
* JavaScript 能够改变页面中的所有 HTML 属性
* JavaScript 能够改变页面中的所有 CSS 样式
* JavaScript 能够对页面中的所有事件做出反应

## 查找 HTML 元素

通常，通过 JavaScript，您需要操作 HTML 元素。

为了做到这件事情，您必须首先找到该元素。有四种方法来做这件事：

* 通过 id 找到 HTML 元素

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

* 通过标签名找到 HTML 元素

|  |
| --- |
| var x=document.getElementById("main"); var y=x.getElementsByTagName("p"); |

* 通过元素名找到 HTML 元素

|  |
| --- |
| var x=document.getElementByName("main"); |

* 通过类名找到 HTML 元素

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

## HTML DOM 教程

在本教程接下来的篇幅中，您将学到：

* 如何改变 HTML 元素的内容 (innerHTML)
* 如何改变 HTML 元素的样式 (CSS)
* 如何对 HTML DOM 事件对出反应
* 如何添加或删除 HTML 元素

# DOM HTML

HTML DOM 允许 JavaScript 改变 HTML 元素的内容。

## 改变 HTML 输出流

JavaScript 能够创建动态的 HTML 内容：

**今天的日期是： Wed Jul 27 2016 22:34:05 GMT+0800 (中国标准时间)**

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

|  |
| --- |
| <!DOCTYPE html> <html> <body> <script> document.write(Date()); </script> </body> </html> |

## 改变 HTML 内容

修改 HTML 内容的最简单的方法时使用 innerHTML 属性。

如需改变 HTML 元素的内容（这里值得是HTML代码），请使用这个语法：

document.getElementById(*id*).innerHTML=*新的 HTML*

如需改变 HTML 元素的内容（这里指的是文本内容），请使用这个语法：

document.getElementById(*id*).innerText=*新的文本*

本例改变了 <p>元素的内容：

|  |
| --- |
| <html> <body> <p id="p1">Hello World!</p> <script> document.getElementById("p1").innerHTML="新文本!"; </script> </body> </html> |

本例改变了 <h1> 元素的内容：

|  |
| --- |
| <!DOCTYPE html> <html> <body> <h1 id="header">Old Header</h1> <script> var element=document.getElementById("header"); element.innerHTML="新标题"; </script> </body> </html> |

解释：

* 上面的 HTML 文档含有 id="header" 的 <h1> 元素
* 我们使用 HTML DOM 来获得 id="header" 的元素
* JavaScript 更改此元素的内容 (innerHTML)

## 改变 HTML 属性

如需改变 HTML 元素的属性，请使用这个语法：

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

本例改变了 <img> 元素的 src 属性：

|  |
| --- |
| <!DOCTYPE html> <html> <body> <img id="image" src="smiley.gif"> <script> document.getElementById("image").src="landscape.jpg"; </script> </body> </html> |

解释：

* 上面的 HTML 文档含有 id="image" 的 <img> 元素
* 我们使用 HTML DOM 来获得 id="image" 的元素
* JavaScript 更改此元素的属性（把 "smiley.gif" 改为 "landscape.jpg"）

# DOM CSS

HTML DOM 允许 JavaScript 改变 HTML 元素的样式。

## 改变 HTML 样式

如需改变 HTML 元素的样式，请使用这个语法：

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

下面的例子会改变 <p> 元素的样式：

|  |
| --- |
| <html> <body> <p id="p2">Hello World!</p> <script> document.getElementById("p2").style.color="blue"; </script> <p>以上段落通过脚本修改。</p> </body> </html> |

## 使用事件

HTML DOM 允许我们通过触发事件来执行代码。

比如以下事件：

* 元素被点击。
* 页面加载完成。
* 输入框被修改。
* ……

在接下来的章节，你会学到更多关于事件的知识。

本例改变了 id="id1" 的 HTML 元素的样式，当用户点击按钮时：

|  |
| --- |
| <!DOCTYPE html> <html> <body> <h1 id="id1">我的标题 1</h1> <button type="button"  onclick="document.getElementById('id1').style.color='red'"> 点我!</button> </body> </html> |

# DOM事件

HTML DOM 使 JavaScript 有能力对 HTML 事件做出反应。

## 对事件做出反应

我们可以在事件发生时执行 JavaScript，比如当用户在 HTML 元素上点击时。

如需在用户点击某个元素时执行代码，请向一个 HTML 事件属性添加 JavaScript 代码：

onclick=*JavaScript*

HTML 事件的例子：

* 当用户点击鼠标时
* 当网页已加载时
* 当图像已加载时
* 当鼠标移动到元素上时
* 当输入字段被改变时
* 当提交 HTML 表单时
* 当用户触发按键时

在本例中，当用户在 <h1> 元素上点击时，会改变其内容：

|  |
| --- |
| <!DOCTYPE html> <html> <body> <h1 onclick="this.innerHTML='Ooops!'">点击文本!</h1> </body> </html> |

本例从事件处理器调用一个函数：

|  |
| --- |
| <!DOCTYPE html> <html> <head> <script> function changetext(id) {     id.innerHTML="Ooops!"; } </script> </head> <body> <h1 onclick="changetext(this)">点击文本!</h1> </body> </html> |

## HTML 事件属性

如需向 HTML 元素分配 事件，您可以使用事件属性。

向 button 元素分配 onclick 事件：

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

在上面的例子中，名为 displayDate 的函数将在按钮被点击时执行。

## 使用 HTML DOM 来分配事件

HTML DOM 允许您使用 JavaScript 来向 HTML 元素分配事件：

向 button 元素分配 onclick 事件：

<script>  
document.getElementById("myBtn").onclick=function(){displayDate()};  
</script>

在上面的例子中，名为 displayDate 的函数被分配给 id=myButn" 的 HTML 元素。

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。

onload 和 onunload 事件可用于处理 cookie。

|  |
| --- |
| <body onload="checkCookies()"> |

## onchange 事件

onchange 事件常结合对输入字段的验证来使用。

下面是一个如何使用 onchange 的例子。当用户改变输入字段的内容时，会调用 upperCase() 函数。

|  |
| --- |
| <input type="text" id="fname" onchange="upperCase()"> |

## 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元素

创建新的 HTML 元素

## 创建新的 HTML 元素

如需向 HTML DOM 添加新元素，您必须首先创建该元素（元素节点），然后向一个已存在的元素追加该元素。

|  |
| --- |
| <div id="div1"> <p id="p1">这是一个段落。</p> <p id="p2">这是另一个段落。</p> </div>  <script> var para=document.createElement("p"); var node=document.createTextNode("这是一个新段落。"); para.appendChild(node);  var element=document.getElementById("div1"); element.appendChild(para); </script> |

## 例子解析：

这段代码创建新的<p> 元素：

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

如需向 <p> 元素添加文本，您必须首先创建文本节点。这段代码创建了一个文本节点：

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

然后您必须向 <p> 元素追加这个文本节点：

para.appendChild(node);

最后您必须向一个已有的元素追加这个新元素。

这段代码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);

## 删除已有的 HTML 元素

以下代码演示了如何删除元素：

|  |
| --- |
| <div id="div1"> <p id="p1">这是一个段落。</p> <p id="p2">这是另一个段落。</p> </div>  <script> var parent=document.getElementById("div1"); var child=document.getElementById("p1"); parent.removeChild(child); </script> |

## 实例解析

这个 HTML 文档含有拥有两个子节点（两个 <p> 元素）的 <div> 元素：

<div id="div1">  
<p id="p1">这是一个段落。</p>  
<p id="p2">这是另一个段落。</p>  
</div>

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);

如果能够在不引用父元素的情况下删除某个元素，就太好了。  
不过很遗憾。DOM 需要清楚您需要删除的元素，以及它的父元素。

这是常用的解决方案：找到您希望删除的子元素，然后使用其 parentNode 属性来找到父元素：

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

学完本节你应该掌握的内容：

* 如何改变 HTML 元素的内容 (innerHTML)
* 如何改变 HTML 元素的样式 (CSS)
* 如何对 HTML DOM 事件作出反应
* 如何添加或删除 HTML 元素

综合例子：

练习1:

列表切换tabs

练习2:

定义一个文本框，输入链接的名字：

在定义一个文本框，输入链接的网址。

再定义一个按钮，点击按钮，获得上面文本框输入的信息，动态创建一个 <a>链接在页面上。

<a href='XXXXXXXX'>链接</a>

a createElement(“a”)

a变量.setAttribute("href",文本框动态获得);

链接 createTextNode(“链接”)

练习3:

消失的星星

练习4：

轮播图

练习5：

贪吃蛇

## 附 录

1. **Document对象方法**

|  |  |
| --- | --- |
| Close() | //关闭到窗口的输出流 |
| createAttribute(“attName”) | //创建属性结点 |
| setAttribute("href",文本框动态获得); |  |
| createComment(“connentText”) | //创建一个注释结点 |
| createElement(“tagName”) | //创建标签元素对象 |
| createTextNode(“text”) | //创建文本 |
| getElementById(“id”); |  |
| getElementByName(“name”) |  |

-------------------------------------------------------------------

1. **JS中关于clientWidth offsetWidth scrollWidth 等的含义**

|  |  |
| --- | --- |
| 网页可见区域宽： | document.body.clientWidth; |
| 网页可见区域高： | document.body.clientHeight; |
| 网页可见区域宽： | document.body.offsetWidth   (包括边线的宽); |
| 网页可见区域高： | document.body.offsetHeight  (包括边线的宽); |
| 网页正文全文宽： | document.body.scrollWidth; |
| 网页正文全文高： | document.body.scrollHeight; |
| 网页被卷去的高： | document.body.scrollTop; |
| 网页被卷去的左： | document.body.scrollLeft; |
| 网页正文部分上： | window.screenTop; |
| 网页正文部分左： | window.screenLeft; |
| 屏幕分辨率的高： | window.screen.height; |
| 屏幕分辨率的宽： | window.screen.width; |
| 屏幕可用工作区高度： | window.screen.availHeight; |
| 屏幕可用工作区宽度： | window.screen.availWidth; |

1. **JavaScript属性**

|  |  |  |
| --- | --- | --- |
| **属性** | **属性值** | **类型** |
| **accessKey** | 属性值：单个字符的字符串 | 类型:可读写 |
| **all[]** | 属性值：嵌套元素对象的数组。获得所有文档中的HTML元素，包括html、head、title、body元素  例：获得表单中的元素  <form>标记name为Demoform  var arr = document.Demoform.all;  for(v in arr){  alert(arr[v]);  } | 类型:只读 |
| **attributes[]** | 属性值：attribute对象引用数组  获得元素的所有属性  var arr = document.getElementById("btn").attributes;  for(v in arr){  alert(arr[v].name);  } | 类型：只读 |
| **behaviorUrns[]** | 属性值:数组 | 类型:只读 |
| **canHaveChildren** | 属性值:Boolean  判断一个特定的元素是否可以包含一个特子元素 | 类型:只读 |
| **canHaveHTML** | 属性值:Boolean  判断一个特定的元素是否可以接收HTML。如div br | 类型：只读 |
| **childNodes[]** | 属性值：结点对象数组  返回当前对象中包含的结点对象的数组。 | 类型：只读 |
| **children[]** | 属性值：元素对象数组  和childNodes区别：不包含文本结点 | 类型：只读 |
| **className** | 属性值:字符串  获得元素class属性的值 | 类型：可读可写 |
| **clientHeight和clientWidth** | 属性值：整型  显示元素内容的像素高度和宽度。Body显示窗口内高和内宽 | 类型：只读 |
| **clientLeft和clientTop** | 属性值：整型  返回元素周围边框的厚度 | 类型：只读 |
| **contentEditable** | 属性值：Boolean  设置一个元素是否是可以编辑的。true可以编辑，false不可以 | 类型：可读可写 |
| **currentStyle** | 属性值：style对象  返回一个元素的style对象，style.width或style.height等 | 类型：只读 |
| **dir** | 属性值：”ltr”或者”rtl”  设置元素文本从左到右显示还是从右到左 | 类型：可读可写 |
| **disabled** | 属性值：Boolean  设置一个元素是否是激活的 true不可以 | 类型：Boolean |
| **document** | 属性值：document对象  包含对象的文档引用 | 类型：只读 |
| **filters[]** | 属性值：数组  获得当前元素设置的滤镜对象 | 类型：只读 |
| **firstChild和lastChild** | 属性值：节点对象引用  表示某个元素内部的首结点和尾结点 | 类型：只读 |
| **height和width** | 属性值：整型或百分比字符串  表示特定元素的高度和宽度 | 类型：可读可写 |
| **hideFocus** | 属性值：Boolean  设置元素隐藏焦点。true隐藏焦点 | 类型：可读可写 |
| **id** | 属性值：字符串  返回元素的id属性的标识符 | 类型：只读 |
| **innerHtml和innerText** | 属性值：字符串  设置元素的包含内容，分别是HTML标记式文本和Text纯文本 | 类型：可读可写 |
| **isContentEditable** | 属性值：Boolean  检查特定的元素是否是可以编辑的 | 类型：只读 |
| **isDisabled** | 属性值:Boolean  检查一个特定元素对象是否为不可用的 | 类型：只读 |
| **isMultiLine** | 属性值:Boolean  检查一个特点元素对象是否设置为不可用的 | 类型：只读 |
| **isTextEdit** | 属性值:Boolean  检查一个对象是否可以拥有其内容创建的TextRange对象 | 类型：只读 |
| **lang** | 属性值：ISO语言代码字符串  显示元素文本内容的语言系统 | 类型：可读可写 |
| **language** | 属性值：字符串  设置或获得标记的脚本语言名称 | 类型：可读可写 |
| **length** | 属性值：整数  返回对象集合或数组中项目的数目，可作为for循环条件 | 类型：可读可写 |
| **localName、namespaceURL和prefix** | 属性值：字符串  应用于xml文档 | 类型：只读 |
| **nextSibling和previousSibling** | 属性值：对象引用  获得元素的兄弟元素（前一个或后一个） | 类型：只读 |
| **nodeName** | 属性值：字符串  获得节点的名称 | 类型：只读 |
| **nodeType** | 属性值：整数  获得节点类型的序号 Element 1 attribute 2 Text 3  document.getElementById("show  time").firstChild.nodeType | 类型：只读 |
| **nodeValue** | 属性值：数值、字符串或空  获得文本结点的值、如果是属性节点，获得属性的值 | 类型：可读可写 |
| **offsetHeight和offsetWidth** | 属性值：整型  获得原高度和宽度，区别clientHeight，标记没有高度按照body的值 | 类型：只读 |
| **offsetLeft和offsetTop** | 属性值：整型  获得元素的像素坐标 | 类型：只读 |
| **offsetParent** | 属性值：对象引用  返回作为当前元素位置上下文的对象的引用 | 类型：只读 |
| **outerHTML和outerText** | 属性值:字符串  输出HTML或者Text | 类型：可读可写 |
| **parentElement** | 属性值：对象  返回父元素 | 类型：只读 |
| **parentNode** | 属性值：对象  返回父结点 | 类型：只读 |
| **parentTextEdit** | 属性值：对象  返回能够生成文本范围的对象容器 | 类型：只读 |
| **readyState** | 属性值：字符串  返回元素的载入状态  complete4 //完全载入  interactive3 //不能完全载入，但可交换  loaded2 //数据载入，对象开始工作  loading1 //数据正载入  uninitialized0 //对象还没有开始载入 | 类型：只读 |
| recordNumber | 属性值：整型或null  返回元素在数据绑定中的重复值 | 类型：只读 |
| runtimeStyle | 属性值：style  返回IE中的默认元素样式，不包括文档中的样式赋值 | 类型：只读 |
| scopeName | 属性值：字符串  返回xml应用中的命名空间 | 类型：只读 |
| scrollHeight和scrollWidth | 属性值：整数  返回对的实际像素尺寸 | 类型：只读 |
| scrollLeft和scrollTop | 属性值：整数  返回元素的滚动条水平和垂直滚动了多远 | 类型：只读 |
| sourceIndex | 属性值：整型  返回document.all中对象的索引号，从0开始 | 类型：只读 |
| Style | 属性值：style对象引用  返回可读可写的style对象(区别currentStyle) | 类型：可读可写 |
| tabIndex | 属性值：整数  元素在tab键位中的顺序 | 类型：可读可写 |
| tagName | 属性值：字符串  返回元素的标签名 | 类型：只读 |
| title | 属性值：字符串  返回元素的title属性值 | 类型：可读可写 |
| uniqueID | 属性值：字符串  获得网页生成的元素ID符 | 类型：只读 |

appendChild(elementObject)

appendChild(newNodeObject,oldNodeObject)

返回值：结点对象引用

方法1：插入元素或文本到当前元素的最后一个子结点

方法2：新结点替换现有的子结点对象

applyElement(elementObject[,type])

插入新元素作为当前对象的父元素或者子元素,第二参数”Outside”父”Inside”子

blur()和focus()

blur从元素中移去焦点 focus获得焦点

clearAttributes()

从元素中删除出name和id属性外的所有属性

click()

完成等同click事件的动作

cloneNode(deepBoolean)

克隆一个对象的副本，参数：包含所有子结点(true)，仅自身(fasle)

Contains(elementObjectReference)

返回值:Boolean

判断一个元素是否嵌套在另一个元素中

getAttribute()/removeAttribute()/setAttribute()

返回/删除/添加对象特定属性的值

getElementByTagName(“tagName”);

返回值：对象数组

以name的形式查找所有子元素

insertAdjacentHTML(“location”,”HTMLtext”)

insertAdjacentText(“location”,”text”)

replacetAdjacentText(“location”,”text”)

当前元素的一个位置“插入或替换”HTML或直接文本

beforeBegin 当前元素标记之前 afterEnd当前标记之后

beforeEnd 所有嵌入内容之后 afterBegin 嵌入内容之前

insertBefore(newChildNOdeObject[,referenceChildNode])

将新的结点插入现有元素，省略第二个参数，则插入到结点最后，否则插入指定结点前面

removeNode(remoceChildrenFlag) 删除指定结点

replaceNode(“newNOdeObject”) 新结点替换当前结点

swapNode(otherNodeObject) 在新元素层次内交换两个结点的位置

tags(“tagName”) 返回值：元素对象数组

对象集合（all,forms,elements）的过滤器,document.all.tags(“div”);获得所有div

## 17、JavaScript 正则表达式

正则表达式（英语：Regular Expression，在代码中常简写为regex、regexp或RE）使用单个字符串来描述、匹配一系列符合某个句法规则的字符串搜索模式。

搜索模式可用于文本搜索和文本替换。

### 什么是正则表达式？

正则表达式是由一个字符序列形成的搜索模式。

当你在文本中搜索数据时，你可以用搜索模式来描述你要查询的内容。

正则表达式可以是一个简单的字符，或一个更复杂的模式。

正则表达式可用于所有文本搜索和文本替换的操作。

### 正则表达式模式

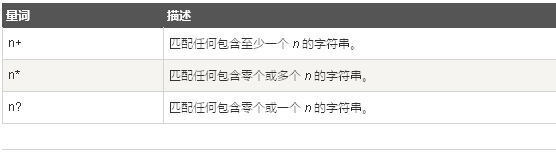
方括号用于查找某个范围内的字符：



元字符是拥有特殊含义的字符：



量词:



### 使用 RegExp 对象

在 JavaScript 中，RegExp 对象是一个预定义了属性和方法的正则表达式对象。

### 使用 test()

test() 方法是一个正则表达式方法。

test() 方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。

以下实例用于搜索字符串中的字符 "e"：

<!DOCTYPE html>

<html>

<body>

<script>

var patt1=new RegExp("e");

document.write(patt1.test("The best things in life are free"));

</script>

</body>

</html>

### 使用 exec()

exec() 方法是一个正则表达式方法。

exec() 方法用于检索字符串中的正则表达式的匹配。

该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。

以下实例用于搜索字符串中的字母 "e":

<!DOCTYPE html>

<html>

<body>

<script>

var patt1=new RegExp("e");

document.write(patt1.exec("The best things in life are free"));

</script>

</body>

</html>

## 18、JavaScript表单验证

JavaScript 可用来在数据被送往服务器前对 HTML 表单中的这些输入数据进行验证。

表单数据经常需要使用 JavaScript 来验证其正确性：

* 验证表单数据是否为空？
* 验证输入是否是一个正确的email地址？
* 验证日期是否输入正确？
* 验证表单输入内容是否为数字型？

必填（或必选）项目

下面的函数用来检查用户是否已填写表单中的必填（或必选）项目。假如必填或必选项为空，那么警告框会弹出，并且函数的返回值为 false，否则函数的返回值则为 true（意味着数据没有问题）：

function validateForm()  
{  
var x=document.forms["myForm"]["fname"].value;  
if (x==null || x=="")  
  {  
  alert("First name must be filled out");  
  return false;  
  }  
}

以上函数在 form 表单提交时被调用:

<!DOCTYPE html>

<html>

<head>

<script>

function validateForm()

{

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

if (x==null || x=="")

{

alert("姓必须填写");

return false;

}

}

</script>

</head>

<body>

<form name="myForm" action="demo-form.php" onsubmit="return validateForm()" method="post">

姓: <input type="text" name="fname">

<input type="submit" value="提交">

</form>

</body>

</html>

E-mail 验证

下面的函数检查输入的数据是否符合电子邮件地址的基本语法。

意思就是说，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：

<!DOCTYPE html>

<html>

<head>

<script>

function validateForm()

{

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("Not a valid e-mail address");

return false;

}

}

</script>

</head>

<body>

<form name="myForm" action="demo-form.php" onsubmit="return validateForm();" method="post">

Email: <input type="text" name="email">

<input type="submit" value="Submit">

</form>

</body>

</html>

# JavaScript高级教程

## JavaScript对象

## JavaScript Number对象

## JavaScript String

## JavaScript Date

## JavaScript Array

## JavaScript Boolean

## JavaScript Math(算数)

## JavaScript Window

## JavaScript Window Screen

## JavaScript Window Location

## JavaScript Window History

## JavaScript Navigator

## JavaScript 弹窗

## JavaScript 计时事件

## JavaScript Cookies

# 第二章JavaScript浏览器BOM

# JavaScript Window

浏览器对象模型 (BOM) 使 JavaScript 有能力与浏览器"对话"。

## 浏览器对象模型 (BOM)

浏览器对象模型（**B**rowser **O**bject **M**odel (BOM)）尚无正式标准。

由于现代浏览器已经（几乎）实现了 JavaScript 交互性方面的相同方法和属性，因此常被认为是 BOM 的方法和属性。

## 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

有三种方法能够确定浏览器窗口的尺寸（浏览器的视口，不包括工具栏和滚动条）。

对于Internet Explorer、Chrome、Firefox、Opera 以及 Safari：

* window.innerHeight - 浏览器窗口的内部高度
* window.innerWidth - 浏览器窗口的内部宽度

对于 Internet Explorer 8、7、6、5：

* document.documentElement.clientHeight
* document.documentElement.clientWidth

或者：

* document.body.clientHeight
* document.body.clientWidth

|  |
| --- |
| 实用的 JavaScript 方案（涵盖所有浏览器）：  var w=window.innerWidth  || document.documentElement.clientWidth  || document.body.clientWidth;  var h=window.innerHeight  || document.documentElement.clientHeight  || document.body.clientHeight; |

该例显示浏览器窗口的高度和宽度：（不包括工具栏/滚动条）

其他window方法

一些其他方法：

* 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

# JavaScript Window Screen

window.screen 对象包含有关用户屏幕的信息。

Window Screen

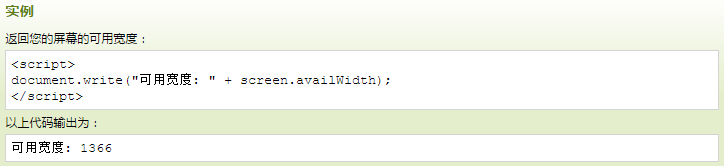
window.screen对象在编写时可以不使用 window 这个前缀。

一些属性：

* screen.availWidth - 可用的屏幕宽度
* screen.availHeight - 可用的屏幕高度

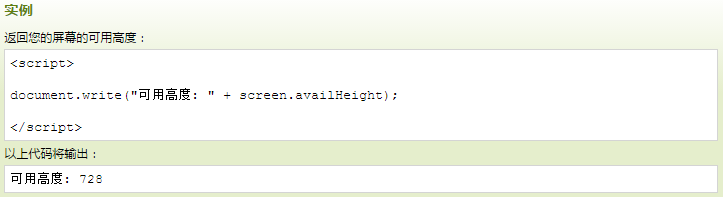
Window Screen 可用宽度

screen.availWidth 属性返回访问者屏幕的宽度，以像素计，减去界面特性，比如窗口任务栏。



Window Screen 可用高度

screen.availHeight 属性返回访问者屏幕的高度，以像素计，减去界面特性，比如窗口任务栏。



# JavaScript Window Location

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

## Window Location

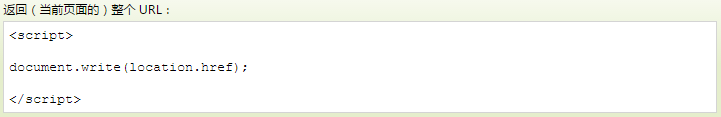
**window.location** 对象在编写时可不使用 window 这个前缀。 一些例子：

一些实例:

* 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://）>

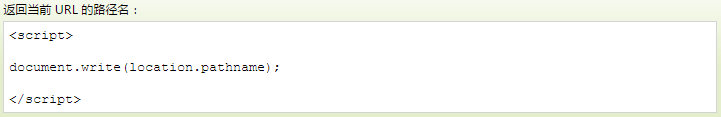
## Window Location Href

location.href 属性返回当前页面的 URL。



## Window Location Pathname

location.pathname 属性返回 URL 的路径名。



## Window Location Assign

location.assign() 方法加载新的文档。

加载一个新的文档：

|  |
| --- |
| <html> <head> <script> function newDoc(){   window.location.assign("http://www.sxsram.com")  } </script> </head> <body> <input type="button" value="Load new document" onclick="newDoc()"> </body> </html> |

# JavaScript Window History

window.history 对象包含浏览器的历史。

## Window History

**window.history**对象在编写时可不使用 window 这个前缀。

为了保护用户隐私，对 JavaScript 访问该对象的方法做出了限制。

一些方法：

* history.back() - 与在浏览器点击后退按钮相同
* history.forward() - 与在浏览器中点击按钮向前相同

## Window History Back

history.back() 方法加载历史列表中的前一个 URL。

这与在浏览器中点击后退按钮是相同的：

|  |
| --- |
| <html> <head> <script> function goBack(){   window.history.back() } </script> </head> <body> <input type="button" value="Back" onclick="goBack()"> </body> </html> |

## Window History Forward

history forward() 方法加载历史列表中的下一个 URL。

这与在浏览器中点击前进按钮是相同的：

在页面上创建一个向前的按钮：

|  |
| --- |
| <html> <head> <script> function goForward(){   window.history.forward() } </script> </head> <body> <input type="button" value="Forward" onclick="goForward()"> </body> </html> |

# JavaScript Window Navigator

window.navigator 对象包含有关访问者浏览器的信息。

## Window Navigator

**window.navigator** 对象在编写时可不使用 window 这个前缀。

|  |
| --- |
| <div id="example"></div> <script> txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>"; txt+= "<p>Browser Name: " + navigator.appName + "</p>"; txt+= "<p>Browser Version: " + navigator.appVersion + "</p>"; txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>"; txt+= "<p>Platform: " + navigator.platform + "</p>"; txt+= "<p>User-agent header: " + navigator.userAgent + "</p>"; txt+= "<p>User-agent language: " + navigator.systemLanguage + "</p>"; document.getElementById("example").innerHTML=txt; </script> |

警告：

来自 navigator 对象的信息具有误导性，不应该被用于检测浏览器版本，这是因为：

* navigator 数据可被浏览器使用者更改
* 一些浏览器对测试站点会识别错误
* 浏览器无法报告晚于浏览器发布的新操作系统

浏览器检测：

由于 navigator 可误导浏览器检测，使用对象检测可用来嗅探不同的浏览器。

由于不同的浏览器支持不同的对象，您可以使用对象来检测浏览器。例如，由于只有 Opera 支持属性 "window.opera"，您可以据此识别出 Opera。

例子：if (window.opera) {...some action...}

# JavaScript弹窗

可以在 JavaScript 中创建三种消息框：警告框、确认框、提示框。

## 警告框

警告框经常用于确保用户可以得到某些信息。

当警告框出现后，用户需要点击确定按钮才能继续进行操作。

### 语法

window.alert("*sometext*");

**window.alert()** 方法可以不带上window对象，直接使用**alert()**方法。

|  |
| --- |
| <!DOCTYPE html> <html> <head> <script> function myFunction(){     alert("你好，我是一个警告框！"); } </script> </head> <body> <input type="button" onclick="myFunction()" value="显示警告框"> </body> </html> |

## 确认框

确认框通常用于验证是否接受用户操作。

当确认卡弹出时，用户可以点击 "确认" 或者 "取消" 来确定用户操作。

当你点击 "确认", 确认框返回 true， 如果点击 "取消", 确认框返回 false。

### 语法

window.confirm("*sometext*");

**window.confirm()** 方法可以不带上window对象，直接使用**confirm()**方法。

|  |
| --- |
| var r=confirm("按下按钮"); if (r==true){     x="你按下了\"确定\"按钮!"; }else{     x="你按下了\"取消\"按钮!"; } |

## 提示框

提示框经常用于提示用户在进入页面前输入某个值。

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。

### 语法

window.prompt("*sometext*","*defaultvalue*");

**window.prompt()** 方法可以不带上window对象，直接使用**prompt()**方法。

|  |
| --- |
| var person=prompt("请输入你的名字","Harry Potter"); if (person!=null && person!=""){     x="你好 " + person + "! 今天感觉如何?";     document.getElementById("demo").innerHTML=x; } |

## 换行

弹窗使用 反斜杠 + "n"(\n) 来设置换行。

|  |
| --- |
| alert("Hello\nHow are you?"); |

# JavaScript计时事件

## JavaScript 计时事件

通过使用 JavaScript，我们有能力作到在一个设定的时间间隔之后来执行代码，而不是在函数被调用后立即执行。我们称之为计时事件。

在 JavaScritp 中使用计时事件是很容易的，两个关键方法是:

* setInterval() - 间隔指定的毫秒数不停地执行指定的代码。
* setTimeout() - 暂停指定的毫秒数后执行指定的代码

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

## setInterval() 方法

setInterval() 间隔指定的毫秒数不停地执行指定的代码

### 语法

window.setInterval("*javascript function*",*milliseconds*);

**window.setInterval()** 方法可以不使用window前缀，直接使用函数**setInterval()**。

setInterval() 第一个参数是函数（function）。

第二个参数间隔的毫秒数

**注意:** 1000 毫秒是一秒。

每三显示秒弹出 "hello" ：

|  |
| --- |
| setInterval(function(){alert("Hello")},3000); |

实例展示了如何使用 setInterval() 方法，但是每三秒弹出一次对用户体验并不好。

以下实例将显示当前时间。 setInterval() 方法设置每秒钟执行一次代码，就是手表一样。

显示当前时间：

|  |
| --- |
| var myVar=setInterval(function(){myTimer()},1000); function myTimer(){ var d=new Date(); var t=d.toLocaleTimeString(); document.getElementById("demo").innerHTML=t; } |

## 如何停止执行?

clearInterval() 方法用于停止 setInterval() 方法执行的函数代码。

### 语法

window.clearInterval(intervalVariable)

**window.clearInterval()** 方法可以不使用window前缀，直接使用函数**clearInterval()**。

要使用 clearInterval() 方法, 在创建计时方法时你必须使用全局变量：

myVar=setInterval("*javascript function*",*milliseconds*);

然后你可以使用clearInterval() 方法来停止执行。

以下例子,我们添加了 "Stop time" 按钮：

|  |
| --- |
| <p id="demo"></p> <button onclick="myStopFunction()">Stop time</button>  <script> var myVar=setInterval(function(){myTimer()},1000); function myTimer(){ var d=new Date(); var t=d.toLocaleTimeString(); document.getElementById("demo").innerHTML=t; } function myStopFunction(){ clearInterval(myVar); } </script> |

## setTimeout() 方法

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

### 语法

setTimeout() 方法会返回某个值。在上面的语句中，值被储存在名为 t 的变量中。假如你希望取消这个 setTimeout()，你可以使用这个变量名来指定它。

setTimeout() 的第一个参数是含有 JavaScript 语句的字符串。这个语句可能诸如 "alert('5 seconds!')"，或者对函数的调用，诸如 alertMsg()"。

第二个参数指示从当前起多少毫秒后执行第一个参数。

提示：1000 毫秒等于一秒。

等待3秒，然后弹出 "Hello":

|  |
| --- |
| setTimeout(function(){alert("Hello")},3000); |

## 如何停止执行?

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

### 语法

window.clearTimeout(timeoutVariable)

**window.clearTimeout()** 方法可以不使用window 前缀。

要使用clearTimeout() 方法, 你必须在创建超时方法中（setTimeout）使用全局变量:

myVar=setTimeout("*javascript function*",*milliseconds*);

如果函数还未被执行，你可以使用 clearTimeout() 方法来停止执行函数代码。

以下是同一个实例, 但是添加了 "Stop the alert" 按钮:

|  |
| --- |
| var myVar; function myFunction(){ myVar=setTimeout(function(){alert("Hello")},3000); } function myStopFunction(){ clearTimeout(myVar); } |

# JavaScript Cookie

Cookies 用于存储 web 页面的用户信息。

## 什么是 Cookies？

Cookies 是一些数据, 存储于你电脑上的文本文件中。

当 web 服务器向浏览器发送 web 页面时，在连接关闭后，服务端不会记录用户的信息。

Cookies 的作用就是用于解决 "如何记录客户端的用户信息":

* 当用户访问 web 页面时，他的名字可以记录在 cookie 中。
* 在用户下一次访问该页面时，可以在 cookie 中读取用户访问记录。

Cookies 以名/值对形式存储，如下所示:

username=John Doe

当浏览器从服务器上请求 web 页面时， 属于该页面的 cookies 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。

## 使用 JavaScript 创建Cookie

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

JavaScript 中，创建 cookie 如下所示：

document.cookie="username=John Doe";

您还可以为 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：

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

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

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

## 使用 JavaScript 读取 Cookie

在 JavaScript 中, 可以使用以下代码来读取 cookies：

var x = document.cookie;

document.cookie 将以字符串的方式返回所有的 cookies，类型格式： cookie1=value; cookie2=value; cookie3=value;

## 使用 JavaScript 修改 Cookie

在 JavaScript 中，修改 cookies 类似于创建 cookies，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。

## 使用 JavaScript 删除 Cookie

删除 cookie 非常简单。您只需要设置 expires 参数为以前的时间即可，如下所示，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，当您删除时不必指定 cookie 的值。

## Cookie 字符串

document.cookie 属性看起来像一个普通的文本字符串，其实它不是。

即使您在 document.cookie 中写入一个完整的 cookie 字符串, 当您重新读取该 cookie 信息时，cookie 信息是以名/值对的形式展示的。

如果您设置了新的 cookie，旧的 cookie 不会被覆盖。 新 cookie 将添加到 document.cookie 中，所以如果您重新读取document.cookie，您将获得如下所示的数据：

cookie1=value; cookie2=value;

如果您需要查找一个指定 cookie 值，您必须创建一个JavaScript 函数在 cookie 字符串中查找 cookie 值。

## JavaScript Cookie 实例

在以下实例中，我们将创建 cookie 来存储访问者名称。

首先，访问者访问 web 页面, 他将被要求填写自己的名字。该名字会存储在 cookie 中。

访问者下一次访问页面时，他会看到一个欢迎的消息。

在这个实例中我们会创建 3 个 JavaScript 函数:

* 设置 cookie 值的函数
* 获取 cookie 值的函数
* 检测 cookie 值的函数

## 设置 cookie 值的函数

首先，我们创建一个函数用于存储访问者的名字：

function setCookie(cname,cvalue,exdays){  
var d = new Date();  
d.setTime(d.getTime()+(exdays\*24\*60\*60\*1000));  
var expires = "expires="+d.toGMTString();  
document.cookie = cname + "=" + cvalue + "; " + expires;  
}

**函数解析：**

以上的函数参数中，cookie 的名称为 cname，cookie 的值为 cvalue，并设置了 cookie 的过期时间 expires。

该函数设置了 cookie 名、cookie 值、cookie过期时间。

## 获取 cookie 值的函数

然后，我们创建一个函数用户返回指定 cookie 的值：

function getCookie(cname){  
var name = cname + "=";  
var ca = document.cookie.split(';');  
for(var i=0; i<ca.length; i++){  
  var c = ca[i].trim();  
  if (c.indexOf(name)==0) return c.substring(name.length,c.length);  
  }  
return "";  
}

**函数解析：**

cookie 名的参数为 cname。

创建一个文本变量用于检索指定 cookie :cname + "="。

使用分号来分割 document.cookie 字符串，并将分割后的字符串数组赋值给 ca (ca = document.cookie.split(';'))。

循环 ca 数组 (i=0;i<ca.length;i++)，然后读取数组中的每个值，并去除前后空格 (c=ca[i].trim())。

如果找到 cookie(c.indexOf(name) == 0)，返回 cookie 的值 (c.substring(name.length,c.length)。

如果没有找到 cookie, 返回 ""。

最后，我们可以创建一个检测 cookie 是否创建的函数。

如果设置了 cookie，将显示一个问候信息。

如果没有设置 cookie，将会显示一个弹窗用于询问访问者的名字，并调用 setCookie 函数将访问者的名字存储 365 天：

function checkCookie(){  
var username=getCookie("username");  
if (username!=""){  
  alert("Welcome again " + username);  
 }  
else{  
  username = prompt("Please enter your name:","");  
  if (username!="" && username!=null){  
    setCookie("username",username,365);  
    }  
  }  
}

## 检测 cookie 值的函数

最后，我们可以创建一个检测 cookie 是否创建的函数。

如果设置了 cookie，将显示一个问候信息。

如果没有设置 cookie，将会显示一个弹窗用于询问访问者的名字，并调用 setCookie 函数将访问者的名字存储 365 天：

function checkCookie(){  
var username=getCookie("username");  
if (username!=""){  
  alert("Welcome again " + username);  
}else{  
  username = prompt("Please enter your name:","");  
  if (username!="" && username!=null){  
    setCookie("username",username,365);  
  }  
}  
}

## 完整实例

|  |
| --- |
| function setCookie(cname,cvalue,exdays){ var d = new Date(); d.setTime(d.getTime()+(exdays\*24\*60\*60\*1000)); var expires = "expires="+d.toGMTString(); document.cookie = cname + "=" + cvalue + "; " + expires; } function getCookie(cname){ var name = cname + "="; var ca = document.cookie.split(';'); for(var i=0; i<ca.length; i++) {   var c = ca[i].trim();   if (c.indexOf(name)==0) return c.substring(name.length,c.length); } return ""; } function checkCookie(){ var user=getCookie("username"); if (user!=""){   alert("Welcome again " + user); } else{   user = prompt("Please enter your name:","");   if (user!="" && user!=null){     setCookie("username",user,365);     }   } } |

# JavaScript库

# JavaScript库

JavaScript 库 - jQuery、Prototype、MooTools。

JavaScript 高级程序设计（特别是对浏览器差异的复杂处理），通常很困难也很耗时。

为了应对这些调整，许多的 **JavaScript (helper)** 库应运而生。

这些 JavaScript 库常被称为 **JavaScript 框架**。

在本教程中，我们将了解到一些广受欢迎的 JavaScript 框架：

* Jquery
* Prototype
* mooTools

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。

在本教程中，您将学习到如何开始使用它们，来使得 JavaScript 编程更容易、更安全且更有乐趣。

**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。

许多大公司在网站上使用 jQuery：

* + Google
  + Microsoft
  + IBM
  + Netflix

**Prototype**

**Prototype** 是一种库，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。

**其他框架**

下面是其他一些在上面未涉及的框架：

**YUI** - Yahoo! User Interface Framework，涵盖大量函数的大型库，从简单的 JavaScript 功能到完整的 internet widget。

**Ext JS** - 可定制的 widget，用于构建富因特网应用程序（rich Internet applications）。

**Dojo** - 用于 DOM 操作、事件、widget 等的工具包。

**script.aculo.us** - 开源的 JavaScript 框架，针对可视效果和界面行为。

**UIZE** - Widget、AJAX、DOM、模板等等。

**CDN内容分发网络**

您总是希望网页可以尽可能地快。您希望页面的容量尽可能地小，同时您希望浏览器尽可能多地进行缓存。

如果许多不同的网站使用相同的 JavaScript 框架，那么把框架库存放在一个通用的位置供每个网页分享就变得很有意义了。

CDN (Content Delivery Network) 解决了这个问题。CDN 是包含可分享代码库的服务器网络。

Google 为一系列 JavaScript 库提供了免费的 CDN，包括：

* jQuery
* Prototype
* MooTools
* Dojo
* Yahoo！YUI

但是由于 Google 在中国经常被GFW（防火长城，英文名称Great Firewall of China，简写为Great Firewall，缩写GFW）屏蔽，造成访问不稳定。

如需在您的网页中使用 JavaScript 框架库，只需在 <script> 标签中引用该库即可：

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

**使用框架**

在您决定为网页使用 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。您无需在计算机上安装它们，同时也没有安装程序。

通常您只需从网页中引用一个库文件。

# JavaScript测试jQuery

测试 JavaScript 框架库 - jQuery

**引用jQuery**

如需测试 JavaScript 库，您需要在网页中引用它。

为了引用某个库，请使用 <script> 标签，其 src 属性设置为库的 URL：

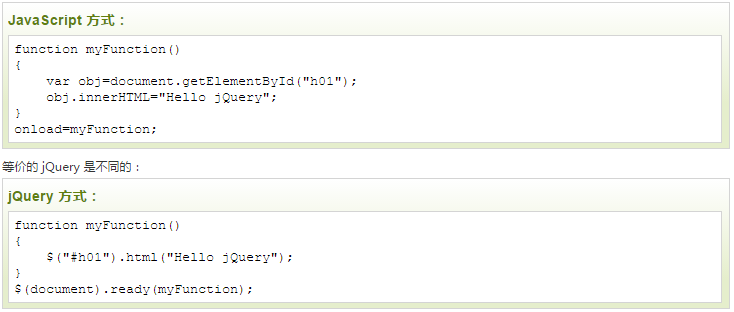
|  |
| --- |
| <!DOCTYPE html> <html> <head> <script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js"> </script> </head> <body> </body> </html> |

**jQuery描述**

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

jQuery 允许您通过 CSS 选择器来选取元素。

在 JavaScript 中，您可以分配一个函数以处理窗口加载事件：



上面代码的最后一行，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。 您不能在 jQuery 对象上使用 HTML DOM 的属性和方法。 |

**测试jQuery**

|  |
| --- |
| <!DOCTYPE html> <html> <head> <script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js"> </script> <script> function myFunction(){     $("#h01").html("Hello jQuery") } $(document).ready(myFunction); </script> </head> <body> <h1 id="h01"></h1> </body> </html> |

|  |
| --- |
| <!DOCTYPE html> <html> <head> <script src="http://apps.bdimg.com/libs/jquery/2.1.1/jquery.min.js"> </script> <script> function myFunction(){     $("#h01").attr("style","color:red").html("Hello jQuery") } $(document).ready(myFunction); </script> </head> <body> <h1 id="h01"></h1> </body> </html> |

正如您在上面的例子中看到的，jQuery 允许链接（链式语法）。

链接（Chaining）是一种在同一对象上执行多个任务的便捷方法。