**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。

**<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:

<script>  
alert("我的第一个 JavaScript");  
</script>

您无需理解上面的代码。只需明白，浏览器会解释并执行位于 <script> 和 </script>之间的 JavaScript 代码

|  |  |
| --- | --- |
| **lamp** | 那些老旧的实例可能会在 <script> 标签中使用 type="text/javascript"。现在已经不必这样做了。JavaScript 是所有现代浏览器以及 HTML5 中的默认脚本语言。 |

**<body> 中的 JavaScript**

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

**实例**

<!DOCTYPE html>  
<html>  
<body>  
.  
.  
<script>  
document.write("<h1>这是一个标题</h1>");  
document.write("<p>这是一个段落</p>");  
</script>  
.  
.  
</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_intro_document_write)

**JavaScript 函数和事件**

上面例子中的 JavaScript 语句，会在页面加载时执行。

通常，我们需要在某个事件发生时执行代码，比如当用户点击按钮时。

如果我们把 JavaScript 代码放入函数中，就可以在事件发生时调用该函数。

您将在稍后的章节学到更多有关 JavaScript 函数和事件的知识。

**在 <head> 或者 <body> 的JavaScript**

您可以在 HTML 文档中放入不限数量的脚本。

脚本可位于 HTML 的 <body> 或 <head> 部分中，或者同时存在于两个部分中。

通常的做法是把函数放入 <head> 部分中，或者放在页面底部。这样就可以把它们安置到同一处位置，不会干扰页面的内容。

**<head> 中的 JavaScript 函数**

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：

**实例**

<!DOCTYPE html>  
<html>

<head>  
<script>  
function myFunction()  
{  
document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";  
}  
</script>  
</head>

<body>

<h1>我的 Web 页面</h1>

<p id="demo">一个段落</p>

<button type="button" onclick="myFunction()">尝试一下</button>

</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_whereto_head)

**<body> 中的 JavaScript 函数**

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

该函数会在点击按钮时被调用：

**实例**

<!DOCTYPE html>  
<html>  
<body>

<h1>我的 Web 页面</h1>

<p id="demo">一个段落</p>

<button type="button" onclick="myFunction()">尝试一下</button>

<script>  
function myFunction()  
{  
document.getElementById("demo").innerHTML="我的第一个 JavaScript 函数";  
}  
</script>

</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_whereto_body)

**外部的 JavaScript**

也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。

外部 JavaScript 文件的文件扩展名是 .js。

如需使用外部文件，请在 <script> 标签的 "src" 属性中设置该 .js 文件：

**实例**

<!DOCTYPE html>  
<html>  
<body>  
<script src="myScript.js"></script>  
</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_externalexample)

你可以将脚本放置于 <head> 或者 <body>中 实际运行效果与您在 <script> 标签中编写脚本完全一致。

|  |  |
| --- | --- |
| **lamp** | 外部脚本不能包含 <script> 标签。 |

**JavaScript 输出**

JavaScript 没有任何打印或者输出的函数。

在 HTML 中， JavaScript 通常用于操作 HTML 元素。

**操作 HTML 元素**

如需从 JavaScript 访问某个 HTML 元素，您可以使用 document.getElementById(*id*) 方法。

请使用 "id" 属性来标识 HTML 元素，并 innerHTML 来获取或插入元素内容：

**实例**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>我的第一个 Web 页面</h1>  
  
<p id="demo">我的第一个段落</p>  
  
<script>  
document.getElementById("demo").innerHTML = "段落已修改。";  
</script>  
  
</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_dom)

以上 JavaScript 语句（在 <script> 标签中）可以在 web 浏览器中执行：

**document.getElementById("demo")** 是使用 id 属性来查找 HTML 元素的 JavaScript 代码 。

**innerHTML = "Paragraph changed."** 是用于修改元素的 HTML 内容(innerHTML)的 JavaScript 代码。

**在本教程中**

在大多数情况下，在本教程中，我们将使用上面描述的方法来输出：

下面的例子直接把 id="demo" 的 <p> 元素写到 HTML 文档输出中：

**写到 HTML 文档**

出于测试目的，您可以将JavaScript直接写在HTML 文档中：

**实例**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>我的第一个 Web 页面</h1>  
  
<p>我的第一个段落。</p>  
  
<script>  
document.write(Date());  
</script>  
  
</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_write)

|  |  |
| --- | --- |
| **Note** | 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。 |

**实例**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>我的第一个 Web 页面</h1>  
  
<p>我的第一个段落。</p>  
  
<button onclick="myFunction()">点我</button>  
  
<script>  
function myFunction() {  
    document.write(Date());  
}  
</script>  
  
</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_write_over)

**写到控制台**

如果您的浏览器支持调试，你可以使用 **console.log()** 方法在浏览器中显示 JavaScript 值。

浏览器中使用 F12 来启用调试模式， 在调试窗口中点击 "Console" 菜单。

**实例**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>我的第一个 Web 页面</h1>  
  
<script>  
a = 5;  
b = 6;  
c = a + b;  
console.log(c);  
</script>  
  
</body>  
</html>

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_console)

**您知道吗?**

|  |  |
| --- | --- |
| **Note** | 程序中调试是测试，查找及减少bug(错误)的过程。 |

**JavaScript 语法**

JavaScript 是一个程序语言。语法规则定义了语言结构。

**JavaScript 语法**

JavaScript 是一个脚本语言。

它是一个轻量级，但功能强大的编程语言。

**JavaScript 字面量**

在编程语言中，一个字面量是一个常量，如 3.14。

**数字（Number）字面量** 可以是整数或者是小数，或者是科学计数(e)。

3.14  
  
1001  
  
123e5

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_syntax_numbers)

**字符串（String）字面量** 可以使用单引号或双引号 be written with double or single quotes:

"John Doe"  
  
'John Doe'

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_syntax_strings)

**表达式字面量** 用于计算：

5 + 6  
  
5 \* 10

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_syntax_expressions)

**数组（Array）字面量** 定义一个数组：

[40, 100, 1, 5, 25, 10]

**对象（Object）字面量** 定义一个对象：

{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}

**函数（Function）字面量** 定义一个函数：

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

**JavaScript 变量**

在编程语言中，变量用于存储数据值。

JavaScript 使用关键字 **var** 来定义变量， 使用等号来为变量赋值：

var x, length  
  
x = 5  
  
length = 6

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_syntax_variables)

变量可以通过变量名访问。在指令式语言中，变量通常是可变的。字面量是一个恒定的值。

|  |  |
| --- | --- |
| **Note** | 变量是一个**名称**。字面量是一个**值**。 |

**JavaScript 操作符**

JavaScript使用 **算术运算符** 来计算值:

(5 + 6) \* 10

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_syntax_operators)

JavaScript使用**赋值运算符**给变量赋值：

x = 5  
y = 6  
z = (x + y) \* 10

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_syntax_assign)

JavaScript语言有多种类型的运算符：/p>

|  |  |  |
| --- | --- | --- |
| **Type** | **实例** | **描述** |
| 赋值，算术和位运算符 | =  +  -  \*  / | 在 JS 运算符中描述 |
| 条件，比较及逻辑运算符 | ==  != <  > | 在 JS 比较运算符中描述 |

**JavaScript 语句**

在 HTML 中，JavaScript 语句向浏览器发出的命令。

语句是用分号分隔：

x = 5 + 6;  
y = x \* 10;

**JavaScript 关键词**

JavaScript 语句通常于**关键词**为开头。 **var** 关键词告诉浏览器创建一个新的变量：

var x = 5 + 6;  
var y = x \* 10;

**JavaScript 标识符**

和其他任何编程语言一样，JavaScript 保留了一些标识符为自己所用。

JavaScript 同样保留了一些关键字，这些关键字在当前的语言版本中并没有使用，但在以后 JavaScript 扩展中会用到。

JavaScript 标识符必须以字母、下划线（\_）或美元符（$）开始。

后续的字符可以是字母、数字、下划线或美元符（数字是不允许作为首字符出现的，以便 JavaScript 可以轻易区分开标识符和数字）。

以下是 JavaScript 中最​​重要的保留字（按字母顺序）：

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

**JavaScript 注释**

不是所有的 JavaScript 语句都是"命令"。双斜杠 **//** 后的内容将会被浏览器忽略：

// 我不会执行

**JavaScript 数据类型**

JavaScript 有多种数据类型：数字，字符串，数组，对象等等：

var length = 16;                                  // Number 通过数字字面量赋值   
var points = x \* 10;                              // Number 通过表达式字面量赋值  
var lastName = "Johnson";                         // String 通过字符串字面量赋值  
var cars = ["Saab", "Volvo", "BMW"];              // Array  通过数组字面量赋值  
var person = {firstName:"John", lastName:"Doe"};  // Object 通过对象字面量赋值

**数据类型的概念**

编程语言中，数据类型是一个非常重要的内容。

为了可以操作变量，了解数据类型的概念非常重要。

如果没有使用数据类型，以下实例将无法执行：

16 + "Volvo"

16 加上 "Volvo" 是如何计算呢? 以上会产生一个错误还是输出以下结果呢？

"16Volvo"

你可以在浏览器尝试执行以上代码查看效果。

在接下来的章节中你将学到更多关于数据类型的知识。

**JavaScript 函数**

JavaScript 语句可以写在函数内，函数可以重复引用：

**引用一个函数** = 调用函数(执行函数内的语句)。

function myFunction(a, b) {  
    return a \* b;                                // 返回 a 乘于 b 的结果  
}

**JavaScript 对大小写敏感。**

JavaScript 对大小写是敏感的。

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。

**JavaScript 字符集**

JavaScript 使用 Unicode 字符集。

Unicode 覆盖了所有的字符，包含标点等字符。

如需进一步了解，请学习我们的 [完整 Unicode 参考手册](http://www.w3cschool.cc/charsets/ref-html-utf8.html)。

**您知道吗?**

|  |  |
| --- | --- |
| **Note** | JavaScript 中，常见的是驼峰法的命名规则，如 lastName (而不是lastname)。 |

**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。

**JavaScript 语句**

JavaScript 语句是发给浏览器的命令。

这些命令的作用是告诉浏览器要做的事情。

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

**实例**

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

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_statement)

**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。

Writing:

a = 5;  
b = 6;  
c = a + b;

Is the same as writing:

a = 5; b = 6; c = a + b;

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_semicolon)

|  |  |
| --- | --- |
| **lamp** | 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。 |

**JavaScript 代码**

JavaScript 代码是 JavaScript 语句的序列。

浏览器按照编写顺序依次执行每条语句。

本例向网页输出一个标题和两个段落：

**实例**

document.getElementById("demo").innerHTML="你好 Dolly";  
document.getElementById("myDIV").innerHTML="你最近怎么样?";

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_statements)

**JavaScript 代码块**

JavaScript 可以分批地组合起来。

代码块以左花括号开始，以右花括号结束。

代码块的作用是一并地执行语句序列。

本例向网页输出一个标题和两个段落：

**实例**

function myFunction()  
{  
document.getElementById("demo").innerHTML="你好Dolly";  
document.getElementById("myDIV").innerHTML="你最近怎么样?";  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_blocks)

您将在稍后的章节学到更多有关函数的知识。

**JavaScript 语句标识符**

JavaScript 语句通常以一个 **语句标识符** 为开始，并执行该语句。

语句标识符是保留关键字不能作为变量名使用。

下表列出了 JavaScript 语句标识符 (关键字) ：

|  |  |
| --- | --- |
| **语句** | **描述** |
| break | 用于跳出循环。 |
| catch | 语句块，在 try 语句块执行出错时执行 catch 语句块。 |
| continue | 跳过循环中的一个迭代。 |
| do ... while | 执行一个语句块，在条件语句为 true 时继续执行该语句块。 |
| for | 在条件语句为 true 时，可以将代码块执行指定的次数。 |
| for ... in | 用于遍历数组或者对象的属性（对数组或者对象的属性进行循环操作）。 |
| function | 定义一个函数 |
| if ... else | 用于基于不同的条件来执行不同的动作。 |
| return | 退出函数 |
| switch | 用于基于不同的条件来执行不同的动作。 |
| throw | 抛出（生成）错误 。 |
| try | 实现错误处理，与 catch 一同使用。 |
| var | 声明一个变量。 |
| while | 当条件语句为 true 时，执行语句块。 |

**JavaScript 对大小写敏感。**

JavaScript 对大小写是敏感的。

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。

**空格**

JavaScript 会忽略多余的空格。您可以向脚本添加空格，来提高其可读性。下面的两行代码是等效的：

var person="Hege";  
var person = "Hege";

**对代码行进行折行**

您可以在文本字符串中使用反斜杠对代码行进行换行。下面的例子会正确地显示：

document.write("你好 \  
世界!");

不过，您不能像这样折行：

document.write \   
("你好世界!");

**您知道吗？**

提示：JavaScript 是脚本语言。浏览器会在读取代码时，逐行地执行脚本代码。而对于传统编程来说，会在执行前对所有代码进行编译。

**JavaScript 注释**

JavaScript 注释可用于提高代码的可读性。

**JavaScript 注释**

JavaScript 不会执行注释。

我们可以添加注释来对 JavaScript 进行解释，或者提高代码的可读性。

单行注释以 // 开头。

本例用单行注释来解释代码：

**实例**

// 输出标题：  
document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
// 输出段落：  
document.getElementById("myP").innerHTML="这是我的第一个段落。";

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comments1)

**JavaScript 多行注释**

多行注释以 /\* 开始，以 \*/ 结尾。

下面的例子使用多行注释来解释代码：

**实例**

/\*  
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  
\*/  
document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
document.getElementById("myP").innerHTML="这是我的第一个段落。";

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comments2)

**使用注释来阻止执行**

在下面的例子中，注释用于阻止其中一条代码行的执行（可用于调试）：

**实例**

//document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
document.getElementById("myP").innerHTML="这是我的第一个段落。";

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comments3)

在下面的例子中，注释用于阻止代码块的执行（可用于调试）：

**实例**

/\*  
document.getElementById("myH1").innerHTML="欢迎来到我的主页";  
document.getElementById("myP").innerHTML="这是我的第一个段落。";  
\*/

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comments4)

**在行末使用注释**

在下面的例子中，我们把注释放到代码行的结尾处：

**实例**

var x=5;    // 声明 x 并把 5 赋值给它  
var y=x+2;  // 声明 y 并把 x+2 赋值给它

# JavaScript 变量

变量是用于存储信息的"容器"。

## 实例

var x=5;  
var y=6;  
var z=x+y;

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_data1)

## 就像代数那样

x=5  
y=6  
z=x+y

在代数中，我们使用字母（比如 x）来保存值（比如 5）。

通过上面的表达式 z=x+y，我们能够计算出 z 的值为 11。

在 JavaScript 中，这些字母被称为变量。

|  |  |
| --- | --- |
| **lamp** | 您可以把变量看做存储数据的容器。 |

## JavaScript 变量

与代数一样，JavaScript 变量可用于存放值（比如 x=5）和表达式（比如 z=x+y）。

变量可以使用短名称（比如 x 和 y），也可以使用描述性更好的名称（比如 age, sum, totalvolume）。

* 变量必须以字母开头
* 变量也能以 $ 和 \_ 符号开头（不过我们不推荐这么做）
* 变量名称对大小写敏感（y 和 Y 是不同的变量）

|  |  |
| --- | --- |
| **lamp** | JavaScript 语句和 JavaScript 变量都对大小写敏感。 |

## JavaScript 数据类型

JavaScript 变量还能保存其他数据类型，比如文本值 (name="Bill Gates")。

在 JavaScript 中，类似 "Bill Gates" 这样一条文本被称为字符串。

JavaScript 变量有很多种类型，但是现在，我们只关注数字和字符串。

当您向变量分配文本值时，应该用双引号或单引号包围这个值。

当您向变量赋的值是数值时，不要使用引号。如果您用引号包围数值，该值会被作为文本来处理。

## 实例

var pi=3.14;  
var person="John Doe";  
var answer='Yes I am!';

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_data2)

## 声明（创建） JavaScript 变量

在 JavaScript 中创建变量通常称为"声明"变量。

我们使用 var 关键词来声明变量：

var carname;

变量声明之后，该变量是空的（它没有值）。

如需向变量赋值，请使用等号：

carname="Volvo";

不过，您也可以在声明变量时对其赋值：

var carname="Volvo";

在下面的例子中，我们创建了名为 carname 的变量，并向其赋值 "Volvo"，然后把它放入 id="demo" 的 HTML 段落中：

## 实例

<p id="demo"></p>  
var carname="Volvo";  
document.getElementById("demo").innerHTML=carname;

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_variables1)

|  |  |
| --- | --- |
| **lamp** | 一个好的编程习惯是，在代码开始处，统一对需要的变量进行声明。 |

## 一条语句，多个变量

您可以在一条语句中声明很多变量。该语句以 var 开头，并使用逗号分隔变量即可：

var lastname="Doe", age=30, job="carpenter";

声明也可横跨多行：

var lastname="Doe",  
age=30,  
job="carpenter";

## Value = undefined

在计算机程序中，经常会声明无值的变量。未使用值来声明的变量，其值实际上是 undefined。

在执行过以下语句后，变量 carname 的值将是 undefined：

var carname;

## 重新声明 JavaScript 变量

如果重新声明 JavaScript 变量，该变量的值不会丢失：

在以下两条语句执行后，变量 carname 的值依然是 "Volvo"：

var carname="Volvo";   
var carname;

## JavaScript 算数

您可以通过 JavaScript 变量来做算数，使用的是 = 和 + 这类运算符：

## 实例

y=5;  
x=y+2;

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_add)

您将在本教程稍后的章节学到更多有关 JavaScript 运算符的知识。

# JavaScript 数据类型

字符串（String）、数字(Number)、布尔(Boolean)、数组(Array)、对象(Object)、空（Null）、未定义（Undefined）。

## JavaScript 拥有动态类型

JavaScript 拥有动态类型。这意味着相同的变量可用作不同的类型：

## 实例

var x;               // x 为 undefined  
var x = 5;           // 现在 x 为数字  
var x = "John";      // 现在 x 为字符串

## JavaScript 字符串

字符串是存储字符（比如 "Bill Gates"）的变量。

字符串可以是引号中的任意文本。您可以使用单引号或双引号：

## 实例

var carname="Volvo XC60";  
var carname='Volvo XC60';

您可以在字符串中使用引号，只要不匹配包围字符串的引号即可：

## 实例

var answer="It's alright";  
var answer="He is called 'Johnny'";  
var answer='He is called "Johnny"';

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_datatypes_string)

您将在本教程的高级部分学到更多关于字符串的知识。

## JavaScript 数字

JavaScript 只有一种数字类型。数字可以带小数点，也可以不带：

## 实例

var x1=34.00;      //使用小数点来写  
var x2=34;         // //不使用小数点来写

极大或极小的数字可以通过科学（指数）计数法来书写：

## 实例

var y=123e5;      // 12300000  
var z=123e-5;     // 0.00123

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_numbers)

您将在本教程的高级部分学到更多关于数字的知识。

## JavaScript 布尔

布尔（逻辑）只能有两个值：true 或 false。

var x=true;  
var y=false;

布尔常用在条件测试中。您将在本教程稍后的章节中学到更多关于条件测试的知识。

## JavaScript 数组

下面的代码创建名为 cars 的数组：

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

或者 (condensed array):

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

或者 (literal array):

## 实例

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

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_datatypes_array)

数组下标是基于零的，所以第一个项目是 [0]，第二个是 [1]，以此类推。

您将在本教程稍后的章节中学到更多关于数组的知识。

## JavaScript 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。属性由逗号分隔：

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

上面例子中的对象 (person) 有三个属性：firstname、lastname 以及 id。

空格和折行无关紧要。声明可横跨多行：

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

对象属性有两种寻址方式：

## 实例

name=person.lastname;  
name=person["lastname"];

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_datatypes_object)

您将在本教程稍后的章节中学到更多关于对象的知识。

## Undefined 和 Null

Undefined 这个值表示变量不含有值。

可以通过将变量的值设置为 null 来清空变量。

## 实例

cars=null;  
person=null;

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_undefined)

## 声明变量类型

当您声明新变量时，可以使用关键词 "new" 来声明其类型：

var carname=new String;  
var x=      new Number;  
var y=      new Boolean;  
var cars=   new Array;  
var person= new Object;

|  |  |
| --- | --- |
| **lamp** | JavaScript 变量均为对象。当您声明一个变量时，就创建了一个新的对象。 |

**JavaScript 对象**

JavaScript 对象是拥有属性和方法的数据。

**真实生活中的对象，属性和方法**

真实生活中，一辆汽车是一个对象。

对象有它的属性，如重量和颜色等，方法有启动停止等:

|  |  |  |
| --- | --- | --- |
| **对象** | **属性** | **方法** |
| http://www.w3cschool.cc/images/objectExplained.gif | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()   car.stop() |

所有汽车都有这些属性，但是每款车的属性都不尽相同。

所有汽车都拥有这些方法，但是它们被执行的时间都不尽相同。

**JavaScript 对象**

在 JavaScript中，几乎所有的事物都是对象。

|  |  |
| --- | --- |
| **Note** | 在 JavaScript 中，对象是非常重要的，当你理解了对象，就可以了解 JavaScript 。 |

你已经学习了 JavaScript 变量的赋值。

以下代码为变量 **car** 设置值为 "Fiat" :

var car = "Fiat";

对象也是一个变量，但对象可以包含多个值（多个变量）。

var car = {type:"Fiat", model:500, color:"white"};

在以上实例中，3 个值 ("Fiat", 500, "white") 赋予变量 car。

在以上实例中，3 个变量 (type, model, color) 赋予变量 car。

|  |  |
| --- | --- |
| **Note** | JavaScript 对象是变量的容器。 |

**对象定义**

你可以使用字符来定义和创建 JavaScript 对象:

**实例**

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

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_object_create_1)

定义 JavaScript 对象可以跨越多行，空格跟换行不是必须的：

**实例**

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

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_object_create_2)

**对象属性**

可以说 "JavaScript 对象是变量的容器"。

但是，我们通常认为 "JavaScript 对象是键值对的容器"。

键值对通常写法为 **name : value** (键与值以冒号分割)。

键值对在 JavaScript 对象通常称为 **对象属性**。

|  |  |
| --- | --- |
| **Note** | JavaScript 对象是属性变量的容器。 |

对象键值对的写法类似于：

* PHP 中的关联数组
* Python 中的字典
* C 语言中的哈希表
* Java 中的哈希映射
* Ruby 和 Perl 中的哈希表

**访问对象属性**

你可以通过两种方式访问对象属性:

**实例 1**

person.lastName;

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_object_properties_1)

**实例 2**

person["lastName"];

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_object_properties_2)

**对象方法**

对象的方法定义了一个函数，并作为对象的属性存储。

对象方法通过添加 () 调用 (作为一个函数)。

该实例访问了 person 对象的 fullName() 方法:

**实例**

name = person.fullName();

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_object_method)

如果你要访问 person 对象的 fullName 属性，它将作为一个定义函数的字符串返回：

**实例**

name = person.fullName;

[**尝试一下 »**](http://www.w3cschool.cc/try/tryit.php?filename=tryjs_object_function)

|  |  |
| --- | --- |
| **Note** | JavaScript 对象是属性和方法的容器。 |

在随后的教程中你将学习到更多关于函数，属性和方法的知识。

**访问对象方法**

你可以使用以下语法创建对象方法：

*methodName : function() { code lines }*

你可以使用以下语法访问对象方法：

*objectName.methodName()*

通常 fullName() 是作为 person 对象的一个方法， fullName 是作为一个属性。

有多种方式可以创建，使用和修改 JavaScript 对象。

同样也有多种方式用来创建，使用和修改属性和方法。

# JavaScript 函数

函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。

## 实例

<!DOCTYPE html>  
<html>  
<head>  
<script>  
function myFunction()  
{  
alert("Hello World!");  
}  
</script>  
</head>  
  
<body>  
<button onclick="myFunction()">Try it</button>  
</body>  
</html>

## JavaScript 函数语法

函数就是包裹在花括号中的代码块，前面使用了关键词 function：

function *functionname*()  
{  
*执行代码*  
}

当调用该函数时，会执行函数内的代码。

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。

|  |  |
| --- | --- |
| **lamp** | 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。 |

## 调用带参数的函数

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*)  
{  
*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

<button onclick="myFunction('Harry Potter','Wizard')">Try it</button>  
  
<script>  
function myFunction(name,job)  
{  
alert("Welcome " + name + ", the " + job);  
}  
</script>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_function2)

上面的函数在按钮被点击时会提示 "Welcome Harry Potter, the Wizard"。

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：

## 实例

<button onclick="myFunction('Harry Potter','Wizard')">Try it</button>  
<button onclick="myFunction('Bob','Builder')">Try it</button>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_function3)

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。

**注意：** 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：

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：

function myFunction(a,b)  
{  
return a\*b;  
}  
  
document.getElementById("demo").innerHTML=myFunction(4,3);

"demo" 元素的 innerHTML 将是：

12

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_function_return)

在您仅仅希望退出函数时 ，也可使用 return 语句。返回值是可选的：

function myFunction(a,b)  
{  
if (a>b)  
  {  
  return;  
  }  
x=a+b  
}

如果 a 大于 b，则上面的代码将退出函数，并不会计算 a 和 b 的总和。

## 局部 JavaScript 变量

在 JavaScript 函数内部声明的变量（使用 var）是局部变量，所以只能在函数内部访问它。（该变量的作用域是局部的）。

您可以在不同的函数中使用名称相同的局部变量，因为只有声明过该变量的函数才能识别出该变量。

只要函数运行完毕，本地变量就会被删除。

## 全局 JavaScript 变量

在函数外声明的变量是全局变量，网页上的所有脚本和函数都能访问它。

## JavaScript 变量的生存期

JavaScript 变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量会在页面关闭后被删除。

## 向未声明的 JavaScript 变量分配值

如果您把值赋给尚未声明的变量，该变量将被自动作为全局变量声明。

这条语句：

carname="Volvo";

将声明一个全局变量 carname，即使它在函数内执行。

# JavaScript 作用域

作用域可访问变量的集合。

## JavaScript 作用域

在 JavaScript 中, 对象和函数同样也是变量。

**在 JavaScript 中, 作用域为可访问变量，对象，函数的集合。**

JavaScript 函数作用域: 作用域在函数内修改。

## JavaScript 局部作用域

变量在函数内声明，变量为局部作用域。

局部变量：只能在函数内部访问。

## 实例

// 此处不能调用 carName 变量  
  
function myFunction() {  
    var carName = "Volvo";  
  
    // 函数内可调用 carName 变量  
  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_scope_local)

因为局部变量只作用于函数内，所以不同的函数可以使用相同名称的变量。

局部变量在函数开始执行时创建，函数执行完后局部变量会自动销毁。

## JavaScript 全局变量

变量在函数外定义，即为全局变量。

全局变量有 **全局作用域**: 网页中所有脚本和函数均可使用。

## 实例

var carName = " Volvo";  
  
// 此处可调用 carName 变量  
  
function myFunction() {  
  
    // 函数内可调用 carName 变量   
  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_scope_global)

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。

以下实例中 carName 在函数内，但是为全局变量。

## 实例

// 此处可调用 carName 变量  
  
function myFunction() {  
    carName = "Volvo";  
  
    // 此处可调用 carName 变量  
  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_local_global)

## JavaScript 变量生命周期

JavaScript 变量生命周期在它声明时初始化。

局部变量在函数执行完毕后销毁。

全局变量在页面关闭后销毁。

## 函数参数

函数参数只在函数内起作用，是局部变量。

## HTML 中的全局变量

在 HTML 中, 全局变量是 window 对象: 所有数据变量都属于 window 对象。

## 实例

//此处可使用 window.carName  
  
function myFunction() {  
    carName = "Volvo";  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_scope_window)

## 你知道吗?

|  |  |
| --- | --- |
| **Note** | 你的全局变量，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。 |

**JavaScript 事件**

HTML 事件是发生在 HTML 元素上的事情。

当在 HTML 页面中使用 JavaScript 时， JavaScript 可以触发这些事件。

**HTML 事件**

HTML 事件可以是浏览器行为，也可以是用户行为。

以下是 HTML 事件的实例：

* HTML 页面完成加载
* HTML input 字段改变时
* HTML 按钮被点击

通常，当事件发生时，你可以做些事情。

在事件触发时 JavaScript 可以执行一些代码。

HTML 元素中可以添加事件属性，使用 JavaScript 代码来添加 HTML 元素。

单引号:

<*some-HTML-element* *some-event*=**'*some JavaScript*'**>

双引号:

<*some-HTML-element* *some-event*=**"*some JavaScript*"**>

在以下实例中，按钮元素中添加了 onclick 属性 (并加上代码):

**实例**

<button onclick='getElementById("demo").innerHTML=Date()'>The time is?</button>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_event_onclick1)

以上实例中，JavaScript 代码将修改 id="demo" 元素的内容。

在下一个实例中，代码将修改自身元素的内容 (使用 **this**.innerHTML):

**实例**

<button onclick="this.innerHTML=Date()">The time is?</button>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_event_onclick)

|  |  |
| --- | --- |
| **Note** | JavaScript代码通常是几行代码。比较常见的是通过事件属性来调用： |

**实例**

<button onclick="displayDate()">The time is?</button>

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_events1)

**常见的HTML事件**

下面是一些常见的HTML事件的列表:

|  |  |
| --- | --- |
| **事件** | **描述** |
| 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 |

更多事件列表: [JavaScript 参考手册 - HTML DOM 事件](http://www.w3cschool.cc/jsref/dom-obj-event.html)。

**JavaScript 可以做什么?**

事件可以用于处理表单验证，用户输入，用户行为及浏览器动作:

* 页面加载时触发事件
* 页面关闭时触发事件
* 用户点击按钮执行动作
* 验证用户输入内容的合法性
* 等等 ...

可以使用多种方法来执行 JavaScript 事件代码：

* HTML 事件属性可以直接执行 JavaScript 代码
* HTML 事件属性可以调用 JavaScript 函数
* 你可以为 HTML 元素指定自己的事件处理程序
* 你可以阻止事件的发生。
* 等等 ...

|  |  |
| --- | --- |
| **Note** | 在 HTML DOM 章节中你将会学到更多关于事件及事件处理程序的知识。 |

# JavaScript 字符串

JavaScript 字符串用于存储和处理文本。

## JavaScript 字符串

字符串可以存储一系列字符，如 "John Doe"。

字符串可以是插入到引号中的任何字符。你可以使用单引号或双引号：

## 实例

var carname = "Volvo XC60";  
var carname = 'Volvo XC60';

你可以使用索引位置来访问字符串中的每个字符：

## 实例

var character = carname[7];

字符串的索引从 0 开始，这意味着第一个字符索引值为 [0],第二个为 [1], 以此类推。

你可以在字符串中使用引号，字符串中的引号不要与字符串的引号相同:

## 实例

var answer = "It's alright";  
var answer = "He is called 'Johnny'";  
var answer = 'He is called "Johnny"';

你也可以在字符串添加转义字符来使用引号：

## 实例

var answer = 'It's alright';  
var answer = "He is called "Johnny"";

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_datatypes_string)

## 字符串长度

可以使用内置属性 **length** 来计算字符串的长度：

## 实例

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
var sln = txt.length;

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_string_length)

## 特殊字符

在 JavaScript 中，字符串写在单引号或双引号来中。

因为这样，以下实例 JavaScript 无法解析：x

 "We are the so-called "Vikings" from the north."

字符串 "We are the so-called " 被截断。

如何解决以上的问题呢？可以使用反斜杠 (\) 来转义 "Vikings" 字符串中的双引号，如下:

 "We are the so-called \"Vikings\" from the north."

 反斜杠是一个**转义字符**。 转义字符将特殊字符转换为字符串字符：

转义字符 (\) 可以用于转义撇号，换行，引号，等其他特殊字符。

下表中列举了在字符串中可以使用转义字符转义的特殊字符：

|  |  |
| --- | --- |
| **代码** | **输出** |
| \' | 单引号 |
| \" | 双引号 |
| \\ | 反斜杠 |
| \n | 换行 |
| \r | 回车 |
| \t | tab(制表符) |
| \b | 退格符 |
| \f | 换页符 |

## 字符串可以是对象

通常， JavaScript 字符串是原始值，可以使用字符创建： **var firstName = "John"**

但我们也可以使用 new 关键字将字符串定义为一个对象： **var firstName = new String("John")**

## 实例

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

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_object_string_type)

|  |  |
| --- | --- |
| **Note** | 不要创建 String 对象。它会拖慢执行速度，并可能产生其他副作用： |

## 实例

var x = "John";                
var y = new String("John");  
(x === y) // is false because x is a string and y is an object.

## 字符串属性和方法

原始值字符串，如 "John", 没有属性和方法(因为他们不是对象)。

原始值可以使用 JavaScript 的属性和方法，因为 JavaScript 在执行方法和属性时可以把原始值当作对象。

**字符串方法我们将在下一章节中介绍。**

## 字符串属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| constructor | 返回创建字符串属性属性的函数 |
| length | 返回字符串的长度 |
| prototype | 允许您向对象添加属性和方法 |

## 字符串方法

|  |  |
| --- | --- |
| **Method** | **描述** |
| charAt() | 返回指定索引位置的字符 |
| charCodeAt() | 返回指定索引位置字符的 Unicode 值 |
| concat() | 连接两个或多个字符串，返回连接后的字符串 |
| fromCharCode() | 将字符转换为 Unicode 值 |
| indexOf() | 返回字符串中检索指定字符第一次出现的位置 |
| lastIndexOf() | 返回字符串中检索指定字符最后一次出现的位置 |
| localeCompare() | 用本地特定的顺序来比较两个字符串 |
| match() | 找到一个或多个正则表达式的匹配 |
| replace() | 替换与正则表达式匹配的子串 |
| search() | 检索与正则表达式相匹配的值 |
| slice() | 提取字符串的片断，并在新的字符串中返回被提取的部分 |
| split() | 把字符串分割为子字符串数组 |
| substr() | 从起始索引号提取字符串中指定数目的字符 |
| substring() | 提取字符串中两个指定的索引号之间的字符 |
| toLocaleLowerCase() | 根据主机的语言环境把字符串转换为小写，只有几种语言（如土耳其语）具有地方特有的大小写映射 |
| toLocaleUpperCase() | 根据主机的语言环境把字符串转换为大写，只有几种语言（如土耳其语）具有地方特有的大小写映射 |
| toLowerCase() | 把字符串转换为小写 |
| toString() | 返回字符串对象值 |
| toUpperCase() | 把字符串转换为大写 |
| trim() | 移除字符串首尾空白 |
| valueOf() | 返回某个字符串对象的原始值 |

# JavaScript 运算符

**运算符 = 用于赋值。**

**运算符 + 用于加值。**

运算符 = 用于给 JavaScript 变量赋值。

算术运算符 + 用于把值加起来。

## 实例

指定变量值，并将值相加：

y=5;  
z=2;  
x=y+z;

在以上语句执行后，x 的值是：

7

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_operators1)

## JavaScript 算术运算符

<="" p="" style="color: rgb(0, 0, 0); font-family: 'Microsoft Yahei', 'Helvetica Neue', Helvetica, Arial, sans-serif; font-size: 12px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255);"><p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** | **x 运算结果** | **y 运算结果** | **在线实例** |
| + | 加法 | x=y+2 | 7 | 5 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_add) |
| - | 减法 | x=y-2 | 3 | 5 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_sub) |
| \* | 乘法 | x=y\*2 | 10 | 5 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_mult) |
| / | 除法 | x=y/2 | 2.5 | 5 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_div) |
| % | 取模（余数） | x=y%2 | 1 | 5 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_mod) |
| ++ | 自增 | x=++y | 6 | 6 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_incr) |
| x=y++ | 5 | 6 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_incr2) |
| -- | 自减 | x=--y | 4 | 4 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_decr) |
| x=y-- | 5 | 4 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_decr2) |

## 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** | **等同于** | **运算结果** | **在线实例** |
| = | x=y |  | x=5 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_equal) |
| += | x+=y | x=x+y | x=15 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_plusequal) |
| -= | x-=y | x=x-y | x=5 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_minequal) |
| \*= | x\*=y | x=x\*y | x=50 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_multequal) |
| /= | x/=y | x=x/y | x=2 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_divequal) |
| %= | x%=y | x=x%y | x=0 | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_oper_modequal) |

## 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。**

## 实例

**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;**

**txt3 运算结果如下:**

**What a verynice day**

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_operators2)

**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：**

## 实例

**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**

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_operators3)

**或者把空格插入表达式中：:**

## 实例

**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**

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_operators4)

## 对字符串和数字进行加法运算

**两个数字相加，返回数字相加的和，如果数字与字符串相加，返回字符串，如下实例：**

## 实例

**x=5+5;  
y="5"+5;  
z="Hello"+5;**

**x,y, 和 z 输出结果为:**

**10  
55  
Hello5**

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_operators5)

**规则:如果把数字与字符串相加，结果将成为字符串！**

# JavaScript 比较 和 逻辑运算符

比较和逻辑运算符用于测试 true 或者 false。

## 比较运算符

比较运算符在逻辑语句中使用，以测定变量或值是否相等。

<p给定**x=5，下面的表格解释了比较运算符：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **运算符** | **描述** | **比较** | **返回值** | **实例** |
| == | 等于 | x==8 | false | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison1) |
| x==5 | true | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison2) |
| === | 绝对等于 (值和类型均相等) | x==="5" | false | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison3) |
| x===5 | true | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison4) |
| != | 不等于 | x!=8 | true | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison5) |
| !== | 绝对不等于 (值和类型均不相等) | x!=="5" | true | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison6) |
| x!==5 | false | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison7) |
| > | 大于 | x>8 | false | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison8) |
| < | 小于 | x<8 | true | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison9) |
| >= | 大于或等于 | x>=8 | false | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison10) |
| <= | 小于或等于 | x<=8 | true | [**实例 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison11) |

## 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) x="Too young";

您将在本教程的下一节中学习更多有关条件语句的知识。

## 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 |

## 条件运算符

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=(condition)?value1:value2

### 例子

## 实例

如果变量 age 中的值小于 18，则向变量 voteable 赋值 "年龄太小"，否则赋值 "年龄已达到"。

voteable=(age<18)?"年龄太小":"年龄已达到";

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_comparison)

# JavaScript If...Else 语句

条件语句用于基于不同的条件来执行不同的动作。

## 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

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

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

## 实例

当时间小于 20:00 时，生成问候 "Good day"：

if (time<20)  
  {  
  x="Good day";  
  }

<p**x 的结果是：**

**Good day**

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_ifthen)**</p**

请注意，在这个语法中，没有 ..else..。您已经告诉浏览器只有在指定条件为 true 时才执行代码。

## 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 (*condition*)  
  {  
*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

当时间小于 20:00 时，生成问候 "Good day"，否则生成问候 "Good evening"。

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

**x**的结果是：

Good day

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_ifthenelse)

## 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 (*condition1*)  
  {  
*当条件 1 为 true 时执行的代码*  
  }  
else if (*condition2*)  
  {  
*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*  
  }

## 实例

如果时间小于 10:00，则生成问候 "Good morning"，如果时间大于 10:00 小于 20:00，则生成问候 "Good day"，否则生成问候 "Good evening"：

if (time<10)  
  {  
  x="Good morning";  
  }  
else if (time>=10 && time<20)  
  {  
  x="Good day";  
  }  
else  
  {  
  x="Good evening";  
  }

**x**的结果是：

Good day

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_elseif)

# JavaScript switch 语句

switch 语句用于基于不同的条件来执行不同的动作。

## JavaScript switch 语句

请使用 switch 语句来选择要执行的多个代码块之一。

### 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。

## 实例

显示今天的星期名称。请注意 Sunday=0, Monday=1, Tuesday=2, 等等：

var day=new Date().getDay();  
switch (day)  
{  
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;  
}

x 的运行结果：

Today it's Friday

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_switch)

## default 关键词

请使用 default 关键词来规定匹配不存在时做的事情：

## 实例

如果今天不是星期六或星期日，则会输出默认的消息：

var day=new Date().getDay();  
switch (day)  
{  
case 6:  
  x="Today it's Saturday";  
  break;  
case 0:  
  x="Today it's Sunday";  
  break;  
default:  
  x="Looking forward to the Weekend";  
}

x 的运行结果：

Looking forward to the Weekend

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_switch2)

# JavaScript for 循环

循环可以将代码块执行指定的次数。

## JavaScript 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。

我们可以这样输出数组的值：

## 一般写法：

document.write(cars[0] + "<br>");   
document.write(cars[1] + "<br>");   
document.write(cars[2] + "<br>");   
document.write(cars[3] + "<br>");   
document.write(cars[4] + "<br>");   
document.write(cars[5] + "<br>");

## 使用for循环

for (var i=0;i<cars.length;i++)  
{   
document.write(cars[i] + "<br>");  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_loop_for)

## 不同类型的循环

JavaScript 支持不同类型的循环：

* **for** - 循环代码块一定的次数
* **for/in** - 循环遍历对象的属性
* **while** - 当指定的条件为 true 时循环指定的代码块
* **do/while** - 同样当指定的条件为 true 时循环指定的代码块

## For 循环

for 循环是您在希望创建循环时常会用到的工具。

下面是 for 循环的语法：

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

for (var i=0; i<5; i++)  
  {  
  x=x + "The number is " + i + "<br>";  
  }

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_loop_for_ex)

从上面的例子中，您可以看到：

Statement 1 在循环开始之前设置变量 (var i=0)。

Statement 2 定义循环运行的条件（i 必须小于 5）。

Statement 3 在每次代码块已被执行后增加一个值 (i++)。

## 语句 1

通常我们会使用语句 1 初始化循环中所用的变量 (var i=0)。

语句 1 是可选的，也就是说不使用语句 1 也可以。

您可以在语句 1 中初始化任意（或者多个）值：

## 实例:

for (var i=0,len=cars.length; i<len; i++)  
{   
document.write(cars[i] + "<br>");  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_loop_for_om1)

同时您还可以省略语句 1（比如在循环开始前已经设置了值时）：

## 实例:

var i=2,len=cars.length;  
for (; i<len; i++)  
{   
document.write(cars[i] + "<br>");  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_loop_for_om2)

## 语句 2

通常语句 2 用于评估初始变量的条件。

语句 2 同样是可选的。

如果语句 2 返回 true，则循环再次开始，如果返回 false，则循环将结束。

|  |  |
| --- | --- |
| **lamp** | 如果您省略了语句 2，那么必须在循环内提供 break。否则循环就无法停下来。这样有可能令浏览器崩溃。请在本教程稍后的章节阅读有关 break 的内容。 |

## 语句 3

通常语句 3 会增加初始变量的值。

语句 3 也是可选的。

语句 3 有多种用法。增量可以是负数 (i--)，或者更大 (i=i+15)。

语句 3 也可以省略（比如当循环内部有相应的代码时）：

## 实例:

var i=0,len=cars.length;  
for (; i<len; )  
{   
document.write(cars[i] + "<br>");  
i++;  
}

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_loop_for_om3)

## For/In 循环

JavaScript for/in 语句循环遍历对象的属性：

## 实例

var person={fname:"John",lname:"Doe",age:25};   
  
for (x in person)  
  {  
  txt=txt + person[x];  
  }

[**尝试一下 »**](http://www.w3cschool.cc/try/try.php?filename=tryjs_object_for_in)

您将在有关 JavaScript 对象的章节学到更多有关 for / in 循环的知识。

## While 循环

我们将在下一章为您讲解 while 循环和 do/while 循环。