

javascript 简介

JavaScript 简介

JavaScript 是互联网上最流行的脚本语言，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，可由所有的现代浏览器执行。

JavaScript 很容易学习。

您将学到什么

下面是您将在本教程中学到的主要内容。

JavaScript：直接写入 HTML 输出流

实例

```
document.write("<h1>这是一个标题</h1>");  
document.write("<p>这是一个段落。</p>");
```



您只能在 HTML 输出中使用 `document.write`。如果您在文档加载完成后使用该方法，会覆盖整个文档。

JavaScript：对事件的反应

实例

```
<button type="button" onclick="alert('欢迎!')">点我!</button>
```

alert() 函数在 JavaScript 中并不常用，但它对于代码测试非常方便。

[onclick 事件](#)只是您即将在本教程中学到的众多事件之一。

JavaScript：改变 HTML 内容

使用 JavaScript 来处理 HTML 内容是非常强大的功能。

实例

```
x=document.getElementById("demo") //查找元素
x.innerHTML="Hello JavaScript";    //改变内容
```

您会经常看到 **document.getElementById("some id")**。这个方法是 HTML DOM 中定义的。

DOM (**D**ocument **O**bject **M**odel) (文档对象模型) 是用于访问 HTML 元素。

您将在本教程的多个章节中学到有关 HTML DOM 的知识。

JavaScript：改变 HTML 图像

本例会动态地改变 HTML 图像的来源 (src)：

点亮灯泡



“点亮灯泡” 实现代码详解：

```
<script>function changeImage(){
    element=document.getElementById('myimage')
    if (element.src.match("bulbon"))
    {
        element.src="/images/pic_bulboff.gif";
    }
    else
    {
        element.src="/images/pic_bulbon.gif";
    }
}</script>
```

以上实例中代码 `element.src.match("bulbon")` 的作用意思是：

检索 `` 里面 `src` 属性的值有没有包含 `bulbon` 这个字符串，如果存在字符串 `bulbon`，图片 `src` 更新为 `bulboff.gif`，若匹配不到 `bulbon` 字符串，`src` 则更新为 `bulbon.gif`

JavaScript 能够改变任意 HTML 元素的大多数属性，而不仅仅是图片。

JavaScript：改变 HTML 样式

改变 HTML 元素的样式，属于改变 HTML 属性的变种。

实例

```
x=document.getElementById("demo") //找到元素
x.style.color="#ff0000";           //改变样式
```

JavaScript: 验证输入

JavaScript 常用于验证用户的输入。

实例

```
if isNaN(x) {alert("不是数字");}
```

以上实例只是普通的验证，如果要在生产环境中使用，需要严格判断，如果输入的空格，或者连续空格 `isNaN` 是判别不出来的。

您知道吗？



JavaScript 与 Java 是两种完全不同的语言，无论在概念上还是设计上。

Java（由 Sun 发明）是更复杂的编程语言。

ECMA-262 是 JavaScript 标准的官方名称。

JavaScript 由 Brendan Eich 发明。它于 1995 年出现在 Netscape 中（该浏览器已停止更新），并于 1997 年被 ECMA（一个标准协会）采纳。

JavaScript 用法

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



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>
```

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>
```

<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>

```

提示： 我们把 JavaScript 放到了页面代码的底部，这样就可以确保在 <p> 元素创建之后再执行脚本。

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>

```

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。



外部脚本不能包含 <script> 标签。

JavaScript 输出

JavaScript 输出

JavaScript 没有任何打印或者输出的函数。

JavaScript 可以通过不同的方式来输出数据：

- 使用 `window.alert()` 弹出警告框。
 - 使用 `document.write()` 方法将内容写到 HTML 文档中。
 - 使用 `innerHTML` 写入到 HTML 元素。
 - 使用 `console.log()` 写入到浏览器的控制台。
-

使用 `window.alert()`

你可以弹出警告框来显示数据：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1>我的第一个页面</h1>
<p>我的第一个段落。</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

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>
```

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。该方法是 [HTML DOM](#) 中定义的。

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

使用 **document.write()** 方法将内容写到 HTML 文档中。此功能可用于写入文本和 HTML。

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>
```

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>
```

写到控制台

如果您的浏览器支持调试，您可以使用 **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>
```

提示： console.log() 方法能够让你看到你在页面中的输出内容，让你更容易调试 javascript；与 alert 相比，console 不会打断你页面的操作，console 里面的内容非常丰富，你可以在控制台输入 console。

JavaScript 语法

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
```

字符串 (String) 字面量 可以使用单引号或双引号：

```
"John Doe"
'John Doe'
```

表达式字面量 用于计算：

```
5 + 6
5 * 10
```

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
```

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。



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
```

JavaScript 使用**赋值运算符**给变量赋值:

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

JavaScript 语言有多种类型的运算符:

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。浏览器通过 JavaScript 语句明白要执行什么操作。

语句是用分号分隔：

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

JavaScript 关键词

JavaScript 语句通常以**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 乘以 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，请注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。

提示：在本站的编程实战中，你可以通过练习[理解 JavaScript 变量的大小写敏感性](#)。

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](#)。

您知道吗？



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



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 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。

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 "Hello Dolly"：

实例

```
document.getElementById("demo").innerHTML = "你好 Dolly.";
```

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;
```



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?";
```

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?";
}
```

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。

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 \ Lenovo!");
```

不过，您不能像这样折行：

```
document.write \ ("你好 Lenovo!");
```

JavaScript 语句练习

[JavaScript 注释语句操作](#)

练习如何在 JavaScript 中对语句进行注释，被注释的代码块在 JavaScript 之中是不会被运行的。

JavaScript 注释

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。";
```

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。";
```

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。";
```

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。";
*/
```

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
```

相关练习

《JavaScript 编程实战》：[JavaScript 注释语句操作](#)

JavaScript 变量

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;
```

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。



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）



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!';
```

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;
```

提示：你可以通过本站的 JavaScript 编程实战部分来练习如何[声明 JavaScript 变量](#)。



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;
```

您将在本教程稍后的章节学到更多有关 JavaScript 运算符的知识。

您可以在 [JavaScript 编程实战](#)部分中了解 JavaScript 算数。

JavaScript 变量学习图

JavaScript 数据类型

2018-12-28 16:41 更新

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"';
```

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
```

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"];
```

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"];
```

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;
```

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;
```



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。

提示：JavaScript 具有隐含的全局概念，意味着你不声明的任何变量都会成为一个全局对象属性。

JavaScript 数据类型学习脑图：

JavaScript 函数

JavaScript 函数

在 JavaScript 中，函数即对象，可以随意地被程序操控，函数可以嵌套在其他函数中定义，这样可以访问它们被定义时所处的作用域中的任何变量。

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。



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。

提示： `function` 中的花括号是必需的，即使函数体内只包含一条语句，仍然必须使用花括号将其括起来。

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>
```

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>
```

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

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。

局部变量比同名全局变量的优先级高，所以局部变量会隐藏同名的全局变量。

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

JavaScript 作用域

作用域是可访问变量的集合。

在 JavaScript 中，能够定义全局作用域或者局部作用域。

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
```

```
}
```

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  
}
```

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  
}
```

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";  
}
```

你知道吗？



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。



在 ES6 中，提供了 let 关键字和 const 关键字。

let 的声明方式与 var 相同，用 let 来代替 var 来声明变量，就可以把变量限制在当前代码块中。

使用 const 声明的是常量，其值一旦被设定便不可被更改。

JavaScript 事件

2018-12-29 17:50 更新

JavaScript 事件

事件是可以被 JavaScript 侦测到的行为。

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。

HTML 网页中的每个元素都可以产生某些可以触发 JavaScript 函数的事件。

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>
```

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>
```



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>
```

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](#)。

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 ...



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

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\"";
```

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;
```

提示：你可以通过本站的 JavaScript 编程实战部分来进一步了解 [JavaScript 获取字符串长度](#) 的操作。

特殊字符

在 JavaScript 中，字符串写在单引号或双引号中。

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
```



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()	将指定的 Unicode 值转换为字符串
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

相关文章

JavaScript 标准参考手册：[JavaScript String 对象](#)

JavaScript 运算符

JavaScript 运算符

本节将介绍 JavaScript 中的三种类型的运算符：算术运算符、赋值运算符以及字符串连接运算符。

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

JavaScript 算术运算符

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	x 运算结果	y 运算结果	在线实例
+	加法	x=y+2	7	5	实例 »
-	减法	x=y-2	3	5	实例 »
*	乘法	x=y*2	10	5	实例 »
/	除法	x=y/2	2.5	5	实例 »
%	取模（余数）	x=y%2	1	5	实例 »
++	自增	x=++y	6	6	实例 »
		x=y++	5	6	实例 »
--	自减	x=--y	4	4	实例 »

		x=y--	5	4	实例 »
--	--	-------	---	---	----------------------

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 »
+=	x+=y	x=x+y	x=15	实例 »
-=	x-=y	x=x-y	x=5	实例 »
=	x=y	x=x*y	x=50	实例 »
/=	x/=y	x=x/y	x=2	实例 »
%=	x%=y	x=x%y	x=0	实例 »

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
```

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
```

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
```

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
```

规则:如果把数字与字符串相加，结果将成为字符串！

提示：为了熟悉 JavaScript 运算符的使用，你可以参考本站的 [JavaScript 编程实战!](#)

JavaScript 比较和逻辑运算符

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。

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 »
		x==5	true	实例 »
===	绝对等于（值和类型均相等）	x=="5"	false	实例 »
		x==5	true	实例 »
!=	不等于	x!=8	true	实例 »
!==	不绝对等于（值和类型有一个不相等，或两个都不相等）	x!="5"	true	实例 »
		x!==5	false	实例 »
>	大于	x>8	false	实例 »
<	小于	x<8	true	实例 »
>=	大于或等于	x>=8	false	实例 »
<=	小于或等于	x<=8	true	实例 »

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

提示：JavaScript 逻辑运算符的优先级是：！、&&、||。

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";
```

JavaScript if…else 语句

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
 - **JavaScript 三目运算** - 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";
}
```

x 的结果是：

```
Good day
```

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
```

提示：在本站的编程实战中，你可以练习如何[使用 JavaScript 的 if 语句](#)！

Javascript 三目运算（三元运算） 语句

请使用 `(condition) ? ture-doing : else-doing;` 语句在条件为 `true` 时执行代码，在条件为 `false` 时执行其他代码。

实例

```
5 > 3 ? alert("5 大于 3") : alert("5 小于 3");
```

注意：if...else 与三目运算这两者的区别，总结为一句话：三目运算有返回值，if else 没有返回值

例子 1:

```
var n=1;if(n>1){
    n=0;
}else{
    n++;
}console.log(n);
#输出结果: 2
var n=1;
n = n>1?0 : n++;console.log(n);
#输出结果为: 1
```

例子 2:

```
var n=1;if(n>1){
    n=0;
}else{
    ++n;
}console.log(n);
#输出结果: 2
var n=1;
n = n>1?0 : ++n; console.log(n);
#输出结果为: 2
```

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 morning

提示: 在使用 if、else if 语句的时候需要注意它们中代码的执行顺序, 具体练习请参考本站的 [《JavaScript 编程实战》](#)!

JavaScript switch 语句

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。你可以在 JavaScript 编程实战中了解怎么[使用 switch 语句进行多项选择](#)。

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
}
```

代码解释：

- 计算一次 switch 表达式
- 把表达式的值与每个 case 的值进行对比
- 如果存在匹配，则执行关联代码

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:

break 关键词

如果 JavaScript 遇到 **break** 关键词，它会跳出 switch 代码块。

此举将停止代码块中更多代码的执行以及 **case** 测试。

如果找到匹配，并完成任务，则随机中断执行 (break)。无需更多测试。

break 能够节省大量执行时间，因为它会“忽略” switch 代码块中的其他代码的执行。

不必中断 switch 代码块中的最后一个 **case**。代码块在此处会自然结束。

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:

默认的 case 不必是 switch 代码块中最后一个 case:

实例

```
switch (new Date().getDay()) {  
    default:  
        text = "期待周末! ";  
        break;  
    case 6:  
        text = "今天是周六";  
        break;  
    case 0:  
        text = "今天是周日";  
}
```

如果 default 不是 switch 代码块中最后一个 case, 请记得用 break 结束默认 case。

常见的代码块

有时您会需要不同的 case 来使用相同的代码。

在本例中, case 4 和 5 分享相同的代码块, 而 0 和 6 分享另一段代码块:

实例

```
switch (new Date().getDay()) {  
    case 4:  
    case 5:  
        text = "周末快到了: ) ";  
        break;  
    case 0:  
    case 6:  
        text = "今天是周末~";  
        break;
```

```
default:
text = "期待周末! ";
}
```

Switching 的细节

如果多种 case 匹配一个 case 值，则选择第一个 case。

如果未找到匹配的 case，程序将继续使用默认 label。

如果未找到默认 label，程序将继续 switch 后的语句。

严格的比较

Switch case 使用严格比较 (===)。

值必须与要匹配的类型相同。

只有操作数属于同一类型时，严格比较才能为 true。

在这个例子中，x 将不匹配：

实例

```
var x = "0";
switch (x) {
case 0:
text = "Off";
break;
case 1:
text = "On";
break;
default:
text = "No value found";
}
```

JavaScript for 循环

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>");
}
```

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>";
}
```

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>");
}
```

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>");
}
```

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。



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++;
}
```

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];
}
```

提示: 在 JavaScript 中, for in 循环不仅可以遍历对象的属性, 还可以遍历数组。

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。

JavaScript while 循环

JavaScript while 循环

JavaScript while 循环的目的是为了反复执行语句或代码块。
只要指定条件为 true，循环就可以一直执行代码块。

while 循环

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：

实例

```
while (i<5)
{
    x=x + "The number is " + i + "<br>";
    i++;
}
```

提示：在本站 JavaScript 编程实战部分，您可以通过练习来实现[使用 while 语句循环迭代](#)。



如果您忘记增加条件中所用变量的值，该循环永远不会结束。这可能导致浏览器崩溃。

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：

实例

```
do
{
    x=x + "The number is " + i + "<br>";
    i++;
}
while (i<5);
```

别忘记增加条件中所用变量的值，否则循环永远不会结束！

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。

本例中的循环使用 **for 循环**来显示 cars 数组中的所有值：

实例

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
for (;cars[i];)
{
    document.write(cars[i] + "<br>");
    i++;
}
```

本例中的循环使用 **while 循环**来显示 cars 数组中的所有值：

实例

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
while (cars[i])
{
    document.write(cars[i] + "<br>");
    i++;
}
```

JavaScript Break 和 Continue 语句

2018-01-03 15:02 更新

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。

continue 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）：

实例

```
for (i=0;i<10;i++)
{
    if (i==3)
    {
        break;
    }
    x=x + "The number is " + i + "<br>";
}
```

由于这个 if 语句只有一行代码，所以可以省略花括号：

```
for (i=0;i<10;i++)
{
    if (i==3) break;
}
```

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

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：

实例

```
for (i=0;i<=10;i++)
{
  if (i==3) continue;
  x=x + "The number is " + i + "<br>";
}
```

注意：由于 break 语句的作用是跳出代码块，所以 break 可以使用于循环和 switch 等；而 continue 语句的作用是进入下一个迭代，所以 continue 只能用于循环的代码块。

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：

实例

```
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>");
}
```

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 种不同的数据类型：

- 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。

实例

```
typeof "John"           // 返回 string
typeof 3.14             // 返回 number
typeof NaN              // 返回 number
typeof false            // 返回 boolean
typeof [ 1,2,3,4]        // 返回 object
typeof {name: 'John', age:34} // 返回 object
typeof new Date()        // 返回 object
typeof function () {}    // 返回 function
typeof myCar             // 返回 undefined (if myCar is not declared)
typeof null             // 返回 object
```

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。

constructor 属性

constructor 属性返回所有 JavaScript 变量的构造函数。

实例

```
"John".constructor      // 返回函数 String() { [native code] }
(3.14).constructor      // 返回函数 Number() { [native code] }
false.constructor       // 返回函数 Boolean() { [native code] }
[1,2, 3,4].constructor  // 返回函数 Array() { [native code] }
{name:'John', age:34}.constructor // 返回函数 Object() { [native code] }
new Date().constructor   // 返回函数 Date() { [native code] }
function() {}.constructor // 返回函数 Function(){ [native code] }
```

你可以使用 `constructor` 属性来查看对象是否为数组 (包含字符串 "Array"):

实例

```
function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}
```

你可以使用 `constructor` 属性来查看是对象是否为日期 (包含字符串 "Date"):

实例

```
function isDate(myDate) {  
    return myDate.constructor.toString().indexOf("Date") > -1;  
}
```

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:

实例


```
String(x)      // 将变量 x 转换为字符串并返回
String(123)    // 将数字 123 转换为字符串并返回
String(100+ 23) // 将数字表达式转换为字符串并返回
```

Number 方法 **toString()** 也是有同样的效果。

实例

```
x.toString()
(123).toString()
(100 + 23).toString()
```

在 [Number 方法](#) 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
<code>toExponential()</code>	把对象的值转换为指数计数法。
<code>toFixed()</code>	把数字转换为字符串，结果的小数点后有指定位数的数字。
<code>toPrecision()</code>	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。

实例

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

在 [Date 方法](#) 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](#) 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
<code>parseFloat()</code>	解析一个字符串，并返回一个浮点数。
<code>parseInt()</code>	解析一个字符串，并返回一个整数。

一元运算符 +

Operator + 可用于将变量转换为数字：

实例

```
var y = "5";      // y 是一个字符串
var x = + y;      // x 是一个数字
```

如果变量不能转换，它仍然会是一个数字，但值为 NaN (不是一个数字)：

实例

```
var y = "John";   // y 是一个字符串
var x = + y;      // x 是一个数字 (NaN)
```

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"
```

null

在 JavaScript 中 null 表示 "什么都没有", 是一个只有一个值的特殊类型, 表示一个空对象引用。

当设置为 "null" 时, 可以用来清空对象:

```
var person = null; // 值为 null(空), 但类型为对象
```

提示: 你可以使用 typeof 检测 null 返回是 object。

undefined

在 JavaScript 中 undefined 是一个没有设置值的变量。

如果一个变量没有设置值的话, 就会返回 undefined:

```
var person; // 值为 undefined(空), 类型是 undefined
```

JavaScript 正则表达式

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。

语法

```
/pattern/modifiers;
```

实例:

```
var patt = /Lenovo/i
```

实例解析：

/Lenovo/i 是一个正则表达式。

Lenovo 是一个模式 (用于检索)。

i 是一个修饰符 (搜索不区分大小写)。

使用字符串方法

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

search() 方法 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串，并返回子字符串的起始位置。

replace() 方法 用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子字符串。

search() 方法使用正则表达式

实例

使用正则表达式搜索 "Lenovo" 字符串，且不区分大小写：

```
var str = "Visit Lenovo";  
var n = str.search(/Lenovo/i);
```

输出结果为：

```
6
```

search() 方法使用字符串

search 方法可使用字符串作为参数。字符串参数会转换为正则表达式：

实例

检索字符串中 "Lenovo" 的子字符串：

```
var str = "Visit Lenovo!";  
var n = str.search("Lenovo");
```

replace() 方法使用正则表达式

实例

使用正则表达式且不区分大小写将字符串中的 Microsoft 替换为 Lenovo :

```
var str = "Visit Microsoft!";  
var res = str.replace(/microsoft/i, "Lenovo");
```

结果输出为:

Visit Lenovo!

replace() 方法使用字符串

replace() 方法将接收字符串作为参数:

```
var str = "Visit Microsoft!";  
var res = str.replace("Microsoft", "Lenovo");
```

你注意到了吗?



正则表达式参数可用在以上方法中（替代字符串参数）。正则表达式使得搜索功能更加强大(如实例中不区分大小写)。

正则表达式修饰符

修饰符 可以在全局搜索中不区分大小写:

修饰符	描述
i	执行对大小写不敏感的匹配。
g	执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。
m	执行多行匹配。

正则表达式模式

方括号用于查找某个范围内的字符:

表达式	描述
[abc]	查找方括号之间的任何字符。
[0-9]	查找任何从 0 至 9 的数字。
(x y)	查找任何以 分隔的选项。

元字符是拥有特殊含义的字符:

元字符	描述
\d	查找数字。
\s	查找空白字符。
\b	匹配单词边界。
\uxxxx	查找以十六进制数 xxxx 规定的 Unicode 字符。

量词:

量词	描述
----	----

<code>n+</code>	匹配任何包含至少一个 <code>n</code> 的字符串。
<code>n*</code>	匹配任何包含零个或多个 <code>n</code> 的字符串。
<code>n?</code>	匹配任何包含零个或一个 <code>n</code> 的字符串。

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":

实例

```
var patt = /e/;
patt.test("The best things in life are free!");
```

字符串中含有 "e", 所以该实例输出为:

```
true
```

你可以不用设置正则表达式的变量, 以上两行代码可以合并为一行:

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

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":

Example 1

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

字符串中含有 "e"，所以该实例输出为:

```
e
```

使用 compile()

compile() 方法用于改变 RegExp。

compile() 既可以改变检索模式，也可以添加或删除第二个参数。

```
var patt1=new RegExp("e"); document.write(patt1.test("The best things in life are free"));  
patt1.compile("d"); document.write(patt1.test("The best things in life are free"));
```

由于字符串中存在 "e"，而没有 "d"，以上代码的输出是:

```
truefalse
```

完整的 RegExp 参考手册

完整的 RegExp 对象参考手册，请参考我们的 [JavaScript RegExp 参考手册](#)。

该参考手册包含了所有 RegExp 对象的方法和属性。

如果你想知道有哪些 [js 常用的正则表达式](#)，请参考 js 实战手册。

JavaScript 正则表达式在线测试工具

[JavaScript 正则表达式在线测试工具](#)



JavaScript 错误处理 Throw、Try 和 Catch

JavaScript 错误 - throw、try 和 catch

try 语句测试代码块的错误。

catch 语句处理错误。

throw 语句创建自定义错误。

JavaScript 错误

当 JavaScript 引擎执行 JavaScript 代码时，会发生各种错误：

可能是语法错误，通常是程序员造成的编码错误或错别字。

可能是拼写错误或语言中缺少的功能（可能由于浏览器差异）。

可能是由于来自服务器或用户的错误输出而导致的错误。

当然，也可能是由于许多其他不可预知的因素。

JavaScript 抛出（throw）错误

当错误发生时，当事情出问题时，JavaScript 引擎通常会停止，并生成一个错误消息。

描述这种情况的技术术语是：JavaScript 将**抛出**一个错误。

JavaScript try 和 catch

try 语句允许我们定义在执行时进行错误测试的代码块。

catch 语句允许我们定义当 try 代码块发生错误时，所执行的代码块。

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

语法

```
try
{
    //在这里运行代码
}
catch(err)
{
    //在这里处理错误
}
```

实例

在下面的例子中，我们故意在 try 块的代码中写了一个错字。

catch 块会捕捉到 try 块中的错误，并执行代码来处理它。

实例

```
<!DOCTYPE html>
<html>
<head>
<script>
var txt="";
function message()
{
try
{
    adddler("Welcome guest!");
}
catch(err)
{
    txt="本页有一个错误。\\n\\n";
    txt+="错误描述: " + err.message + "\\n\\n";
    txt+="点击确定继续。\\n\\n";
    alert(txt);
}
}
</script>
</head>

<body>
```

```
<input type="button" value="查看消息" onclick="message()">
</body>

</html>
```

Throw 语句

throw 语句允许我们创建自定义错误。

正确的技术术语是：创建或**抛出异常**（exception）。

如果把 throw 与 try 和 catch 一起使用，那么您能够控制程序流，并生成自定义的错误消息。

语法

```
throw exception
```

异常可以是 JavaScript 字符串、数字、逻辑值或对象。

实例

本例检测输入变量的值。如果值是错误的，会抛出一个异常（错误）。catch 会捕捉到这个错误，并显示一段自定义的错误消息：

实例

```
<script>
function myFunction()
{
  try
  {
    var x=document.getElementById("demo").value;
    if(x=="")    throw "empty";
    if(isNaN(x)) throw "not a number";
    if(x>10)    throw "too high";
    if(x<5)     throw "too low";
  }
  catch(err)
  {
    var y=document.getElementById("mess");
    y.innerHTML="Error: " + err + ".";
  }
}
</script>
```

```
<h1>My First JavaScript</h1>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="mess"></p>
```

JavaScript 调试

JavaScript 调试

在编写 JavaScript 时，如果没有调试工具将是一件很痛苦的事情。

JavaScript 调试

没有调试工具是很难去编写 JavaScript 程序的。

你的代码可能包含语法错误，逻辑错误，如果没有调试工具，这些错误比较难于发现。

通常，如果 JavaScript 出现错误，是不会有提示信息，这样你就无法找到代码错误的位置。



通常，你在编写一个新的 JavaScript 代码过程中都会发生错误。

JavaScript 调试工具

在程序代码中寻找错误叫做代码调试。

调试很难，但幸运的是，很多浏览器都内置了调试工具。

内置的调试工具可以开始或关闭，严重的错误信息会发送给用户。

有了调试工具，我们就可以设置断点 (代码停止执行的位置)，且可以在代码执行时检测变量。

浏览器启用调试工具一般是按下 F12 键，并在调试菜单中选择 "Console" 。

console.log() 方法

如果浏览器支持调试，你可以使用 console.log() 方法在调试窗口上打印 JavaScript 值：

实例

```
<!DOCTYPE html>
<html>
```

```
<body>

<h1>My First Web Page</h1>

<script>
a = 5;
b = 6;
c = a + b;
console.log(c);
</script>

</body>
</html>
```

设置断点

在调试窗口中，你可以设置 JavaScript 代码的断点。

在每个断点上，都会停止执行 JavaScript 代码，以便于我们检查 JavaScript 变量的值。

在检查完毕后，可以重新执行代码（如播放按钮）。

debugger 关键字

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

这个关键字与在调试工具中设置断点的效果是一样的。

如果没有调试可用，debugger 语句将无法工作。

开启 debugger，代码在第三行前停止执行。

实例

```
var x = 15 * 5;
debugger;
document.getElementById("demo").innerHTML = x;
```

主要浏览器的调试工具

通常，浏览器启用调试工具一般是按下 F12 键，并在调试菜单中选择 "Console" 。

各浏览器的步骤如下：

Chrome 浏览器

- 打开浏览器。
- 在菜单中选择工具。
- 在工具中选择开发者工具。
- 最后，选择 Console。

Firefox 浏览器

- 打开浏览器。
- 访问页面：
<http://www.getfirebug.com>。
- 按照说明：
安装 Firebug。

Internet Explorer 浏览器。

- 打开浏览器。
- 在菜单中选择工具。
- 在工具中选择开发者工具。
- 最后，选择 Console。

Opera

- 打开浏览器。
- Opera 的内置调试工具为 Dragonfly，详细说明可访问页面：
<http://www.opera.com/dragonfly/>。

Safari

- 打开浏览器。
- 访问页面：
<http://extentions.apple.com>。

- 按说明操作：
install Firebug Lite。

JavaScript 表单验证

JavaScript 表单验证

JavaScript 表单验证

JavaScript 可用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：

实例

```
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm()" method="post">
First name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

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。

也就是说，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：

```
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;
}
}
```

下面是连同 HTML 表单的完整代码：

实例

```
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm();" method="post">
Email: <input type="text" name="email">
<input type="submit" value="Submit">
</form>
```

数字验证

下面的函数检查输入的数据是否是 1~10 之间的数字。假如输入的不为数字或不是，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）：

实例

```
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm();" method="post">
```

```

<strong>请输入 1 到 10 之间的数字: </strong>

<input id="number">

<button type="button" onclick="myFunction()">提交</button>

</form>

```

JavaScript 保留关键字

JavaScript 保留关键字

保留关键字在意思上表达成为将来的关键字而保留的单词。

在 JavaScript 中，一些标识符是保留关键字，不能用作变量名或函数名。

JavaScript 标准

所有的现代浏览器完全支持 [ECMAScript 3](#) (ES3, JavaScript 的第三版, 从 1999 年开始)。

ECMAScript 4 (ES4) 未通过。

ECMAScript 5 (ES5, 2009 年发布)，是 JavaScript 最新的官方版本。

随着时间的推移，我们开始看到，所有的现代浏览器已经完全支持 ES5。

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		

* 标记的关键字是 ECMAScript5 中新添加的。

JavaScript 对象、属性和方法

您也应该避免使用 JavaScript 内置的对象、属性和方法的名称作为 Javascript 的变量或函数名：

Array	Date	eval	function	hasOwnProperty
Infinity	isFinite	isNaN	isPrototypeOf	length
Math	NaN	name	Number	Object
prototype	String	toString	undefined	valueOf

Java 保留关键字

JavaScript 经常与 Java 一起使用。您应该避免使用一些 Java 对象和属性作为 JavaScript 标识符：

getClass	java	JavaArray	javaClass	JavaObject	JavaPackage
----------	------	-----------	-----------	------------	-------------

Windows 保留关键字

JavaScript 可以在 HTML 外部使用。它可在许多其他应用程序中作为编程语言使用。

在 HTML 中，您必须（为了可移植性，您也应该这么做）避免使用 HTML 和 Windows 对象和属性的名称作为 Javascript 的变量及函数名：

alert	all	anchor	anchors	area
assign	blur	button	checkbox	clearInterval
clearTimeout	clientInformation	close	closed	confirm
constructor	crypto	decodeURI	decodeURIComponent	defaultStatus
document	element	elements	embed	embeds
encodeURIComponent	encodeURIComponent	escape	event	fileUpload
focus	form	forms	frame	innerHeight
innerWidth	layer	layers	link	location
mimeType	navigate	navigator	frames	frameRate
hidden	history	image	images	offscreenBuffering
open	opener	option	outerHeight	outerWidth
packages	pageXOffset	pageYOffset	parent	parseFloat
parseInt	password	pkcs11	plugin	prompt
propertyIsEnum	radio	reset	screenX	screenY
scroll	secure	select	self	setInterval
setTimeout	status	submit	taint	text
textarea	top	unescape	untaint	window

HTML 事件句柄

除此之外，您还应该避免使用 HTML 事件句柄的名称作为 Javascript 的变量及函数名。

实例：

onblur	onclick	onerror	onfocus
onkeydown	onkeypress	onkeyup	onmouseover
onload	onmouseup	onmousedown	onsubmit

注意：在 JavaScript 中关键字不能用作变量名或者函数名，否则可能会得到错误消息，例如 `"'Identifier Expected"`（应该有标识符、期望标识符）。

非标准 JavaScript

除了保留关键字，在 JavaScript 实现中也有一些非标准的关键字。

一个实例是 **const** 关键字，用于定义变量。一些 JavaScript 引擎把 const 当作 var 的同义词。另一些引擎则把 const 当作只读变量的定义。

Const 是 JavaScript 的扩展。JavaScript 引擎支持它用在 Firefox 和 Chrome 中。但是它并不是 JavaScript 标准 ES3 或 ES5 的组成部分。**建议：不要使用它。**

JavaScript JSON

JavaScript JSON

JSON 是用于存储和传输数据的格式。

JSON 通常用于服务端向网页传递数据。

什么是 JSON?

- JSON 英文全称 **JavaScript Object Notation**
- JSON 是一种轻量级的数据交换格式。
- JSON 是独立的语言 *
- JSON 易于理解。



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

JSON 实例

以下 JSON 语法定义了 employees 对象: 3 条员工记录 (对象) 的数组:

JSON Example

```
{ "employees": [
  { "firstName": "John", "lastName": "Doe" },
  { "firstName": "Anna", "lastName": "Smith" },
  { "firstName": "Peter", "lastName": "Jones" }
]}
```

JSON 格式化后为 JavaScript 对象

JSON 格式在语法上与创建 JavaScript 对象代码是相同的。

由于它们很相似，所以 JavaScript 程序可以很容易的将 JSON 数据转换为 JavaScript 对象。

JSON 语法规则

- 数据为 键/值 对。
- 数据由逗号分隔。
- 大括号保存对象
- 方括号保存数组

JSON 数据 - 一个名称对应一个值

JSON 数据格式为 键/值 对，就像 JavaScript 对象属性。

键/值对包括字段名称 (在双引号中)，后面一个冒号，然后是值：

```
"firstName": "John"
```

JSON 对象

JSON 对象保存在大括号内。

就像在 JavaScript 中, 对象可以保存多个 键/值 对:

```
{ "firstName": "John", "lastName": "Doe" }
```

JSON 数组

JSON 数组保存在中括号内。

就像在 JavaScript 中, 数组可以包含对象:

```
"employees": [
  { "firstName": "John", "lastName": "Doe" },
  { "firstName": "Anna", "lastName": "Smith" },
  { "firstName": "Peter", "lastName": "Jones" }
]
```

在以上实例中, 对象 "employees" 是一个数组。包含了三个对象。

每个为个对象为员工的记录 (姓和名) 。

JSON 字符串转换为 JavaScript 对象

通常我们从服务器中读取 JSON 数据, 并在网页中显示数据。

简单起见, 我们网页中直接设置 JSON 字符串 (你还可以阅读我们的 [JSON 教程](#)):

首先, 创建 JavaScript 字符串, 字符串为 JSON 格式的数据:

```
var text = '{ "employees" : [' +
'{ "firstName": "John" , "lastName": "Doe" },' +
'{ "firstName": "Anna" , "lastName": "Smith" },' +
'{ "firstName": "Peter" , "lastName": "Jones" } ]}';
```

然后, 使用 JavaScript 内置函数 `JSON.parse()` 将字符串转换为 JavaScript 对象:

```
var obj = JSON.parse(text);
```

最后, 在你的页面中使用新的 JavaScript 对象:

实例

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

<script>
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```


javascript:void(0) 含义

javascript:void(0) 含义

我们经常会使用到 javascript:void(0) 这样的代码，那么在 JavaScript 中 javascript:void(0) 代表的是什么意思呢？

javascript:void(0) 中最关键的是 void 关键字，void 是 JavaScript 中非常重要的关键字，该操作符指定要计算一个表达式但是不返回值。

下面的代码创建了一个超级链接，当用户点击以后不会发生任何事。

实例

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

当用户链接时，void(0) 计算为 0，但 Javascript 上没有任何效果。

以下实例中，在用户点击链接后显示警告信息：

实例

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<a href="javascript:void(alert('Warning!!!'))">点我!</a>
</body>
```

以下实例中参数 a 将返回 undefined：

实例

```
<head>
<script type="text/javascript">
<!--
function getValue(){
  var a,b,c;
  a = void ( b = 5, c = 7 );
}
```

```
document.write('a = ' + a + ' b = ' + b + ' c = ' + c );
}
//-->
</script>
</head>
```

href="#"与href="javascript:void(0)"的区别

包含了一个位置信息，默认的锚是#top 也就是网页的上端。

而javascript:void(0), 仅仅表示一个死链接。

在页面很长的时候会使用 # 来定位页面的具体位置，格式为：# + id。

如果你要定义一个死链接请使用 javascript:void(0) 。

实例

```
<a href="javascript:void(0);">点我没有反应的!</a>
<a href="#pos">点我定位到指定位置!</a>
<br><br> <p id="pos">尾部定位点</p>
```

JavaScript 代码规范

JavaScript 代码规范

所有的 JavaScript 项目适用同一种规范。

JavaScript 代码规范

代码规范通常包括以下几个方面:

- 变量和函数的命名规则
- 空格，缩进，注释的使用规则。
- 其他常用规范.....

规范的代码可以更易于阅读与维护。

代码规范一般在开发前规定，可以跟你的团队成员来协商设置。

变量名

变量名推荐使用驼峰法来命名(**camelCase**):

```
firstName = "John";
lastName = "Doe";

price = 19.90;
tax = 0.20;

fullPrice = price + (price * tax);
```

在 JavaScript 中定义变量名时，还应该注意以下事项:

- 变量名应该区分大小写，允许包含字母、数字、美元符号(\$)和下划线，但第一个字符不允许是数字，不允许包含空格和其他标点符号；
- 变量命名长度应该尽可能的短，并抓住要点，尽量在变量名中体现出值的类型；
- 变量名的命名应该是有意义的；
- 变量名不能为 JavaScript 中的关键词、保留字全名；
- 变量命名方法常见的有匈牙利命名法、驼峰命名法和帕斯卡命名法。

空格与运算符

通常运算符 (= + - * /) 前后需要添加空格:

实例:

```
var x = y + z;var values = ["Volvo", "Saab", "Fiat"];
```

代码缩进

通常使用 4 个空格符号来缩进代码块:

函数:

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

注意:不推荐使用 TAB 键来缩进，因为不同编辑器 TAB 键的解析不一样。

语句规则

简单语句的通用规则:

- 一条语句通常以分号作为结束符。

实例:

```
var values = ["Volvo", "Saab", "Fiat"];
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

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，并以分号作为结束符号。

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：

实例：

```
document.getElementById("demo").innerHTML =  
  "Hello Lenovo.";
```

注意：在函数声明、函数表达式、函数调用、对象创建、数组创建、for 语句等场景中，不允许在 , 或 ; 前换行。

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 的横杠(-)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)。

注意:- 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。

注意: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")
```

HTML 与 JavaScript 尽量使用相同的命名规则。

[访问 HTML\(5\) 代码规范。](#)

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。

学完本教程之后, Lenovo 推荐您进行实战练习来巩固自己的知识: [javascript 实战](#)

JavaScript 格式化整理工具

[在线 JavaScript 格式化整理工具](#)



JavaScript 函数定义

JavaScript 函数定义

JavaScript 使用关键字 **function** 定义函数。
函数可以通过声明定义，也可以是一个表达式。

函数声明

在之前的教程中，你已经了解了函数声明的语法：

```
function functionName(parameters) {  
    执行的代码  
}
```

函数声明后不会立即执行，会在我们需要的时候调用到。

实例

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




分号是用来分隔可执行 JavaScript 语句。
由于函数声明不是一个可执行语句，所以不以分号结束。

函数表达式

JavaScript 函数可以通过一个表达式定义。

函数表达式可以存储在变量中：

实例

```
var x = function (a, b) {return a * b};
```

在函数表达式存储在变量后，变量也可作为一个函数使用：

实例

```
var x = function (a, b) {return a * b};  
var z = x(4, 3);
```

以上函数实际上是一个 **匿名函数** (函数没有名称)。

函数存储在变量中，不需要函数名称，通常通过变量名来调用。



上述函数以分号结尾，因为它是一个执行语句。

Function() 构造函数

在以上实例中，我们了解到函数通过关键字 **function** 定义。

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

实例

```
var myFunction = new Function("a", "b", "return a * b");  
var x = myFunction(4, 3);
```

实际上，你不必使用构造函数。上面实例可以写成：

实例

```
var myFunction = function (a, b) {return a * b}  
var x = myFunction(4, 3);
```



在 JavaScript 中，很多时候，你需要避免使用 **new** 关键字。

函数提升 (Hoisting)

在之前的教程中我们已经了解了 "hoisting(提升)"。

提升 (Hoisting) 是 JavaScript 默认将当前作用域提升到前面去的的行为。

提升 (Hoisting) 应用在变量的声明与函数的声明。

因此，函数可以在声明之前调用：

```
myFunction(5);  
function myFunction(y) {  
    return y * y;  
}
```

使用表达式定义函数时无法提升。

自调用函数

函数表达式可以 "自调用"。

自调用表达式会自动调用。

如果表达式后面紧跟 `()`，则会自动调用。

`Y` 不能自调用声明的函数。

通过添加括号，来说明它是一个函数表达式：

实例

```
(function () {  
    var x = "Hello!!";    // 我将调用自己  
})();
```

以上函数实际上是一个 **匿名自我调用的函数** (没有函数名)。

函数可作为一个值使用

JavaScript 函数作为一个值使用：

实例

```
function myFunction(a, b) {  
    return a * b;  
}  
  
var x = myFunction(4, 3);
```

JavaScript 函数可作为表达式使用：

实例

```
function myFunction(a, b) {  
    return a * b;  
}  
  
var x = myFunction(4, 3) * 2;
```

函数是对象

在 JavaScript 中使用 **typeof** 操作符判断函数类型将返回 "function" 。

但, JavaScript 函数描述为一个对象更加准确。

JavaScript 函数有 **属性** 和 **方法**。

arguments.length 属性返回函数调用过程接收到的参数个数:

实例

```
function myFunction(a, b) {  
    return arguments.length;  
}
```

toString() 方法将函数作为一个字符串返回:

实例

```
function myFunction(a, b) {  
    return a * b;  
}  
  
var txt = myFunction.toString();
```



函数定义作为对象的属性, 称之为对象方法。
函数如果用于创建新的对象, 称之为对象的构造函数。

JavaScript 函数练习

JavaScript 函数定义

在 JavaScript 中把代码的重复部分抽取出来，放到一个函数（functions）中。

JavaScript 定义带参数函数

JavaScript 函数的参数 parameters 充当占位符(也叫形参)的作用，参数可以为一个或多个。调用一个函数时所传入的参数为实参，实参决定着形参真正的值。

JavaScript 函数参数

JavaScript 函数参数

JavaScript 函数对参数的值(arguments)没有进行任何的检查。

JavaScript 函数参数与大多数其他语言的函数参数的区别在于：它不会关注有多少个参数被传递，不关注传递的参数的数据类型。

函数显式参数与隐藏参数(arguments)

在先前的教程中，我们已经学习了函数的显式参数：

```
functionName(parameter1, parameter2, parameter3) {  
    code to be executed  
}
```

函数显式参数在函数定义时列出。

函数隐藏参数(arguments)在函数调用时传递给函数真正的值。

参数规则

JavaScript 函数定义时参数没有指定数据类型。

JavaScript 函数对隐藏参数(arguments)没有进行检测。

JavaScript 函数对隐藏参数(arguments)的个数没有进行检测。

默认参数

如果函数在调用时缺少参数，参数会默认设置为：**undefined**

有时这是可以接受的，但是建议最好为参数设置一个默认值：

实例

```
function myFunction(x, y) {  
  if (y === undefined) {  
    y = 0;  
  }  
}
```

或者，更简单的方式：

实例

```
function myFunction(x, y) {  
  y = y || 0;  
}
```



如果 y 已经定义， $y || 0$ 返回 y ，因为 y 是 `true`，否则返回 `0`，因为 `undefined` 为 `false`。

如果函数调用时设置了过多的参数，参数将无法被引用，因为无法找到对应的参数名。只能使用 `arguments` 对象来调用。

Arguments 对象

JavaScript 函数有个内置的对象 `arguments` 对象。

`argument` 对象包含了函数调用的参数数组。

通过这种方式你可以很方便的找到最后一个参数的值：

实例

```
x = findMax(1, 123, 500, 115, 44, 88);
```

```
function findMax() {  
  var i, max = arguments[0];  
  
  if(arguments.length < 2) return max;
```

```

    for (i = 0; i < arguments.length; i++) {
        if (arguments[i] > max) {
            max = arguments[i];
        }
    }
    return max;
}

```

或者创建一个函数用来统计所有数值的和：

实例

```

x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {
    var i, sum = 0;
    for (i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    return sum;
}

```

通过值传递参数

在函数中调用的参数是函数的参数。

如果函数修改参数的值，将不会修改参数的初始值（在函数外定义）。

总结：JavaScript 函数传值只是将参数的值传入函数，函数会另外配置内存保存参数值，所以并不会改变原参数的值。

实例

```

var x = 1; // 通过值传递参数 function myFunction(x) {
    x++; // 修改参数 x 的值，将不会修改在函数外定义的变量 x
}

```

```
        console.log(x);
    }
    myFunction(x); // 2console.log(x); // 1
```

通过对象传递参数

在 JavaScript 中，可以引用对象的值。
因此我们在函数内部修改对象的属性就会修改其初始的值。
修改对象属性可作用于函数外部（全局变量）。

实例

```
var obj = {x:1}; // 通过对象传递参数
function myFunction(obj) {
    obj.x++; // 修改参数对象 obj.x 的值，函数外定义的 obj 也将被修改
    console.log(obj.x);
}
myFunction(obj); // 2console.log(obj.x); // 2
```

提示： 你可以在本站的 [JavaScript 编程实战](#) 中练习使用 JavaScript 函数参数！


JavaScript 函数调用

JavaScript 函数调用

JavaScript 函数有 4 种调用方式。
每种方式的不同在于 **this** 的初始化。

this 关键字

一般而言，在 Javascript 中，this 指向函数执行时的当前对象。



注意 **this** 是保留关键字，你不能修改 **this** 的值。

提示： 本站的《深入理解 JavaScript》中的 this 关键字一节你可以学到更多与 JavaScript this 关键字相关的内容！

调用 JavaScript 函数

在之前的章节中我们已经学会了如何创建函数。

函数中的代码在函数被调用后执行。

作为一个函数调用

实例

```
function myFunction(a, b) {  
    return a * b;  
}  
myFunction(10, 2);           // myFunction(10, 2) 返回 20
```

以上函数不属于任何对象。但是在 JavaScript 中它始终是默认的全局对象。

在 HTML 中默认的全局对象是 HTML 页面本身，所以函数是属于 HTML 页面。

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

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

实例

```
function myFunction(a, b) {  
    return a * b;  
}  
window.myFunction(10, 2);    // window.myFunction(10, 2) 返回 20
```



这是调用 JavaScript 函数常用的方法，但不是良好的编程习惯
全局变量，方法或函数容易造成命名冲突的 bug。

全局对象

当函数没有被自身的对象调用时，**this** 的值就会变成全局对象。

在 web 浏览器中全局对象是浏览器窗口（window 对象）。

该实例返回 **this** 的值是 window 对象：

实例

```
function myFunction() {
    return this;
}
myFunction();           // 返回 window 对象
```



函数作为全局对象调用，会使 **this** 的值成为全局对象。使用 **window** 对象作为一个变量容易造成程序崩溃。

函数作为方法调用

在 JavaScript 中你可以将函数定义为对象的方法。

以下实例创建了一个对象 (**myObject**)，对象有两个属性 (**firstName** 和 **lastName**)，及一个方法 (**fullName**)：

实例

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

fullName 方法是一个函数。函数属于对象。**myObject** 是函数的所有者。

this 对象，拥有 JavaScript 代码。实例中 **this** 的值为 **myObject** 对象。

测试一下！修改 **fullName** 方法并返回 **this** 值：

实例

```
var myObject = {
    firstName: "John",
    lastName: "Doe",
    fullName: function () {
```

```

        return this;
    }
}
myObject.fullName();           // 返回 [object Object] (所有者对象)

```



函数作为对象方法调用，会使得 **this** 的值成为对象本身。

使用构造函数调用函数

如果函数调用前使用了 **new** 关键字, 则是调用了构造函数。

这看起来就像创建了新的函数, 但实际上 JavaScript 函数是重新创建的对象:

实例

```

// 构造函数:
function myFunction(arg1, arg2) {
    this.firstName = arg1;
    this.lastName  = arg2;
}

// This creates a new object
var x = new myFunction("John","Doe");
x.firstName;           // 返回 "John"

```

构造函数的调用会创建一个新的对象。新对象会继承构造函数的属性和方法。



构造函数中 **this** 关键字没有任何的值。
this 的值在函数调用时实例化对象 (new object) 时创建。

作为函数方法调用函数

在 JavaScript 中, 函数是对象。JavaScript 函数有它的属性和方法。

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

实例

```
function myFunction(a, b) {
    return a * b;
}
myFunction.call(myObject, 10, 2);    // 返回 20
```


实例

```
function myFunction(a, b) {
    return a * b;
}
myArray = [10,2];
myFunction.apply(myObject, myArray);    // 返回 20
```

两个方法都使用了对象本身作为第一个参数。两者的区别在于第二个参数：apply 传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而 call 则作为 call 的参数传入（从第二个参数开始）。

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

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



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

JavaScript 闭包

JavaScript 闭包

JavaScript 变量可以是局部变量或全局变量。

私有变量可以用到闭包。

全局变量

函数可以访问函数内部定义的变量，如：

实例

```
function myFunction() {
    var a = 4;
```

```
    return a * a;
}
```

函数也可以访问函数外部定义的变量，如：

实例

```
var a = 4;
function myFunction() {
    return a * a;
}
```

后面一个实例中，**a** 是一个 **全局** 变量。

在 web 页面中全局变量属于 window 对象。

全局变量可应用于页面上的所有脚本。

在第一个实例中，**a** 是一个 **局部** 变量。

局部变量只能用于定义它函数内部。对于其他的函数或脚本代码是不可用的。

全局和局部变量即便名称相同，它们也是两个不同的变量。修改其中一个，不会影响另一个的值。



变量声明是如果不使用 **var** 关键字，那么它就是一个全局变量，即便它在函数内定义。

提示：请在 JavaScript 编程实战中练习如何[定义 JavaScript 函数的全局变量](#)！

变量生命周期

全局变量的作用域是全局性的，即在整个 JavaScript 程序中，全局变量处处都在。

而在函数内部声明的变量，只在函数内部起作用。这些变量是局部变量，作用域是局部性的；函数的参数也是局部性的，只在函数内部起作用。

计数器困境

设想下如果你想统计一些数值，且该计数器在所有函数中都是可用的。

你可以使用全局变量，函数设置计数器递增：

实例

```
var counter = 0;

function add() {
    counter += 1;
}

add();
add();
add();

// 计数器现在为 3
```

计数器数值在执行 add() 函数时发生变化。

但问题来了，页面上的任何脚本都能改变计数器，即便没有调用 add() 函数。

如果我在函数内声明计数器，如果没有调用函数将无法修改计数器的值：

实例

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

add();
add();
add();

// 本意是想输出 3，但事与愿违，输出的都是 1 ！
```

以上代码将无法正确输出，每次我调用 add() 函数，计数器都会设置为 1。

JavaScript 内嵌函数可以解决该问题。

JavaScript 内嵌函数

所有函数都能访问全局变量。

实际上，在 JavaScript 中，所有函数都能访问它们上一层的作用域。

JavaScript 支持嵌套函数。嵌套函数可以访问上一层的函数变量。

该实例中，内嵌函数 **plus()** 可以访问父函数的 **counter** 变量：

实例

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

如果我们能在外部访问 **plus()** 函数，这样就能解决计数器的困境。

我们同样需要确保 **counter = 0** 只执行一次。

我们需要闭包。

JavaScript 闭包

还记得函数自我调用吗？该函数会做什么？

实例

```
var add = (function () {  
    var counter = 0;  
    return function () {return counter += 1;}  
})();  
  
add();  
add();  
add();  
  
// 计数器为 3
```

实例解析

变量 `add` 指定了函数自我调用的返回字值。

自我调用函数只执行一次。设置计数器为 0。并返回函数表达式。

`add` 变量可以作为一个函数使用。非常棒的部分是它可以访问函数上一层作用域的计数器。

这个叫作 JavaScript **闭包**。它使得函数拥有私有变量变成可能。

计数器受匿名函数的作用域保护，只能通过 `add` 方法修改。

HTML DOM

JavaScript HTML DOM

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

HTML DOM（文档对象模型）

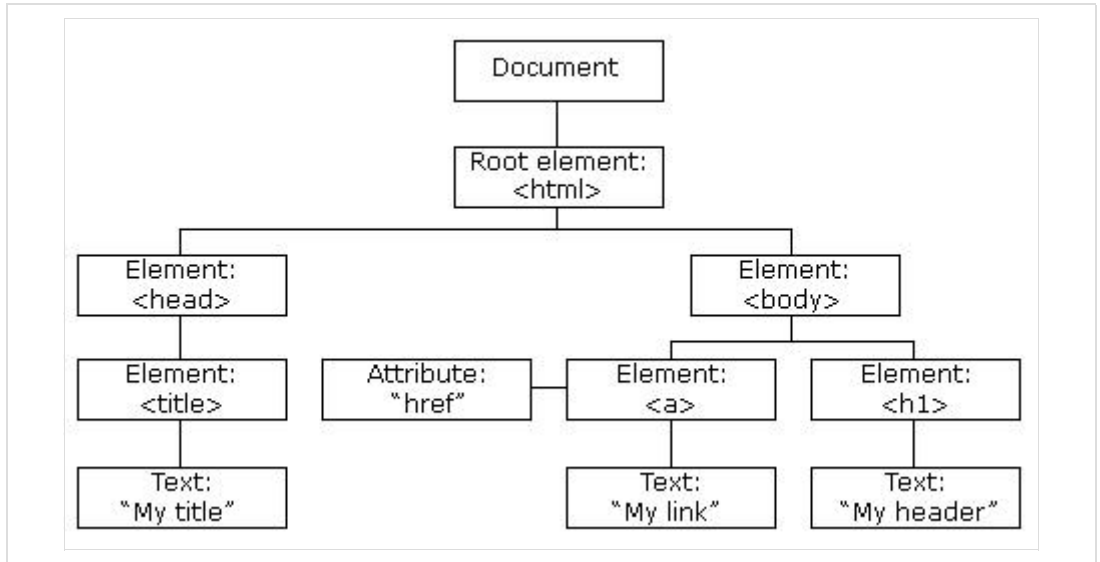
当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model）。

HTML DOM 定义了用于 HTML 的一系列标准的对象，以及访问和处理 HTML 文档的标准方法。通过 DOM，你可以访问所有的 HTML 元素，连同它们所包含的文本和属性。

HTML DOM 独立于平台和编程语言。它可被任何编程语言诸如 Java、JavaScript 和 VBScript 使用。

HTML DOM 模型被构造为**对象**的树：

HTML DOM 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。有三种方法来做这件事：

- 通过 id 找到 HTML 元素
- 通过标签名找到 HTML 元素
- 通过类名找到 HTML 元素

通过 id 查找 HTML 元素

在 DOM 中查找 HTML 元素的最简单的方法，是通过使用元素的 id。

本例查找 id="intro" 元素：

实例

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

如果找到该元素，则该方法将以对象（在 x 中）的形式返回该元素。

如果未找到该元素，则 x 将包含 null。

通过标签名查找 HTML 元素

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

实例

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

通过类名找到 HTML 元素

本例通过 **getElementsByClassName** 函数来查找 class="intro" 的元素：

实例

```
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 事件作出反应
- 如何添加或删除 HTML 元素

HTML DOM 改变 HTML 内容

JavaScript HTML DOM - 改变 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：

Thu Feb 25 2016 19:25:10 GMT+0800

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

实例

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```



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

改变 HTML 内容

修改 HTML 内容的最简单的方法是使用 [innerHTML 属性](#)。

如需改变 HTML 元素的内容，请使用这个语法：

```
document.getElementById(id).innerHTML=new HTML
```

本例改变了 <p>元素的内容：

实例

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML="New text!";
</script>

</body>
</html>
```

本例改变了 <h1> 元素的内容：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element=document.getElementById("header");
element.innerHTML="New Header";
</script>

</body>
</html>
```

实例讲解：

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=new value
```

本例改变了 元素的 src 属性：

实例

```
<!DOCTYPE html>
<html>
<body>



<script>
document.getElementById("image").src="landscape.jpg";
</script>

</body>
</html>
```

实例讲解：

- 上面的 HTML 文档含有 id="image" 的 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"）
-

HTML DOM 改变 CSS

JavaScript HTML DOM - 改变 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=new style
```

下面的例子会改变 <p> 元素的样式：

实例

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color="blue";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
```

本例改变了 id="id1" 的 HTML 元素的样式，当用户点击按钮时：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color='red'">
Click Me!</button>

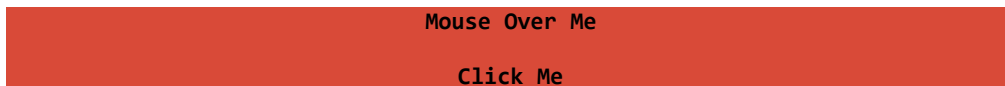
</body>
</html>
```

HTML DOM 事件

JavaScript HTML DOM 事件

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

实例



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：

实例

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Oops!'">点击文本!</h1>
</body>
</html>
```

本例从事件处理器调用一个函数：

实例

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id)
{
id.innerHTML="Oops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</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。

实例

向 button 元素分配 onclick 事件：

```
<button onclick="displayDate()">点我</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：

实例

向 button 元素分配 onclick 事件:

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

在上面的例子中, 名为 displayDate 的函数被分配给 id=myBtn 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。

实例

```
<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。

实例

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

onmouseover 和 onmouseout 事件

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

实例

一个简单的 onmouseover-onmouseout 实例：

Mouse Over Me

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。

实例

一个简单的 onmousedown-onmouseup 实例：

Thank You

HTML DOM EventListener

JavaScript HTML DOM EventListener

addEventListener() 方法

实例

当用户点击按钮时触发监听事件：

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

`addEventListener()` 方法用于向指定元素添加事件句柄。

`addEventListener()` 方法添加的事件句柄不会覆盖已存在的事件句柄。

你可以向一个元素添加多个事件句柄。

你可以向同个元素添加多个同类型的事件句柄，如：两个 "click" 事件。

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

`addEventListener()` 方法可以更简单的控制事件（冒泡与捕获）。

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

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

语法

```
element.addEventListener(event, function, useCapture);
```

第一个参数是事件的类型 (如 "click" 或 "mousedown").

第二个参数是事件触发后调用的函数。

第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。



注意:不要使用 "on" 前缀。例如，使用 "click", 而不是使用 "onclick"。

向原元素添加事件句柄

实例

当用户点击元素时弹出 "Hello World!" :

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

你可以使用函数名，来引用外部函数：

实例

当用户点击元素时弹出 "Hello World!"：

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

function myFunction() {
    alert ("Hello World!");
}
```

向同一个元素中添加多个事件句柄

addEventListener() 方法允许向同个元素添加多个事件，且不会覆盖已存在的事件：

实例

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

你可以向同个元素添加不同类型的事件：

实例

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

提示：你可以在本站的 [HTML DOM addEventListener\(\) 方法](#)部分获得更多有关 addEventListener()方法的信息！

向 Window 对象添加事件句柄

`addEventListener()` 方法允许你在 HTML DOM 对象添加事件监听，HTML DOM 对象如：HTML 元素, HTML 文档, window 对象。或者其他支出的事件对象如: XMLHttpRequest 对象。

实例

当用户重置窗口大小时添加事件监听:

```
window.addEventListener("resize", function(){
    document.getElementById("demo").innerHTML = sometext;
});
```

传递参数

当传递参数值时，使用"匿名函数"调用带参数的函数：

实例

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

事件冒泡或事件捕获？

事件传递有两种方式：冒泡与捕获。

事件传递定义了元素事件触发的顺序。如果你将 <p> 元素插入到 <div> 元素中，用户点击 <p> 元素，哪个元素的 "click" 事件先被触发呢？

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

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

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

```
addEventListener(event, function, useCapture);
```

默认值为 false, 即冒泡传递, 当值为 true 时, 事件使用捕获传递。

实例

```
document.getElementById("myDiv").addEventListener("click", myFunction, true);
```

提示: DOM 事件流同时支持两种事件模型: 捕获型事件和冒泡型事件, 捕获型事件首先发生。捕获型事件和冒泡型事件会触及 DOM 中的所有对象, 从 document 对象开始, 也在 document 对象结束。

removeEventListener() 方法

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

实例

```
element.removeEventListener("mousemove", myFunction);
```

浏览器支持

表格中的数字表示支持该方法的第一个浏览器的版本号。

方法					
addEventListener()	1.0	9.0	1.0	1.0	7.0
removeEventListener()	1.0	9.0	1.0	1.0	7.0

注意: IE 8 及更早 IE 版本, Opera 7.0 及其更早版本不支持 addEventListener() 和 removeEventListener() 方法。但是, 对于这类浏览器版本可以使用 detachEvent() 方法来移除事件句柄:

```
element.attachEvent(event, function);
element.detachEvent(event, function);
```

实例

跨浏览器解决方法:

```
var x = document.getElementById("myBtn");
if (x.addEventListener) { // 所有主流浏览器，除了 IE 8 及更早版本
    x.addEventListener("click", myFunction);
} else if (x.attachEvent) { // IE 8 及更早版本
    x.attachEvent("onclick", myFunction);
}
```

HTML DOM 事件对象参考手册

所有 HTML DOM 事件，可以查看我们完整的 [HTML DOM Event 对象参考手册](#)。

HTML DOM 元素

JavaScript HTML DOM 元素(节点)

创建新的 HTML 元素

在文档对象模型 (DOM) 中，每个节点都是一个对象。DOM 节点有三个重要的属性，分别是：

1. nodeName : 节点的名称
 2. nodeValue : 节点的值
 3. nodeType : 节点的类型
-

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。

实例

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para=document.createElement("p");
var node=document.createTextNode("This is new.");
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("This is a new paragraph.");
```

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

以下代码将已有的元素删除：

实例

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</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">This is a paragraph.</p>
<p id="p2">This is another paragraph.</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);
```

HTML DOM 教程

在我们的 JavaScript 教程的 HTML DOM 部分，您已经学到了：

- 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

如果您希望学到更多有关使用 JavaScript 访问 HTML DOM 的知识，请访问我们完整的 [HTML DOM 教程](#)。

JavaScript 对象

JavaScript 对象

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数...

此外，JavaScript 允许自定义对象。

所有事物都是对象

JavaScript 提供多个内建对象，比如 String、Date、Array 等等。对象只是带有属性和方法的特殊数据类型。

- 布尔型可以是一个对象。
- 数字型可以是一个对象。
- 字符串也可以是一个对象
- 日期是一个对象
- 数学和正则表达式也是对象

- 数组是一个对象
- 甚至函数也可以是对象

JavaScript 对象

对象只是一种特殊的数据。对象拥有**属性**和**方法**。

访问对象的属性

属性是与对象相关的值。

访问对象属性的语法是：

```
objectName.propertyName
```

这个例子使用了 String 对象的 length 属性来获得字符串的长度：

```
var message="Hello World!";  
var x=message.length;
```

在以上代码执行后，x 的值将是：

```
12
```

访问对象的方法

方法是能够在对象上执行的动作。

您可以通过以下语法来调用方法：

```
objectName.methodName()
```

这个例子使用了 String 对象的 toUpperCase() 方法来将文本转换为大写：

```
var message="Hello world!";  
var x=message.toUpperCase();
```

在以上代码执行后，x 的值将是：

```
HELLO WORLD!
```

创建 JavaScript 对象

通过 JavaScript，您能够定义并创建自己的对象。

创建新对象有两种不同的方法：

- 定义并创建对象的实例
- 使用函数来定义对象，然后创建新的对象实例

创建直接的实例

这个例子创建了对象的一个新实例，并向其添加了四个属性：

实例

```
person=new Object();
person.firstname="John";
person.lastname="Doe";
person.age=50;
person.eyecolor="blue";
```

替代语法（使用对象 literals）：

实例

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

提示：

你可以在本站的 JavaScript 编程实战中练习使用

[JavaScript 对象操作！](#)

使用对象构造器

本例使用函数来构造对象：

实例

```
function person(firstname,lastname,age,eyecolor)
{
    this.firstname=firstname;
    this.lastname=lastname;
    this.age=age;
    this.eyecolor=eyecolor;
}
```

在 JavaScript 中，this 通常指向的是我们正在执行的函数本身，或者是指向该函数所属的对象（运行时）

创建 JavaScript 对象实例

一旦您有了对象构造器，就可以创建新的对象实例，就像这样：

```
var myFather=new person("John","Doe",50,"blue");
var myMother=new person("Sally","Rally",48,"green");
```

把属性添加到 JavaScript 对象

您可以通过为对象赋值，向已有对象添加新属性：

假设 personObj 已存在 - 您可以为其添加这些新属性：firstname、lastname、age 以及 eyecolor：

```
person.firstname="John";
person.lastname="Doe";
person.age=30;
person.eyecolor="blue";

x=person.firstname;
```

在以上代码执行后，x 的值将是：

```
John
```

把方法添加到 JavaScript 对象

方法只不过是附加在对象上的函数。

在构造器函数内部定义对象的方法：

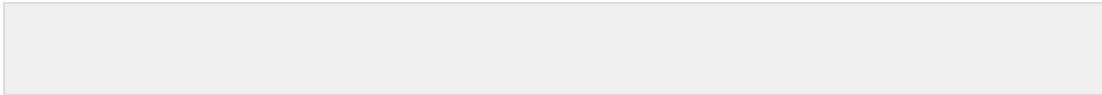
```
function person(firstname,lastname,age,eyecolor)
{
  this.firstname=firstname;
  this.lastname=lastname;
  this.age=age;
  this.eyecolor=eyecolor;

  this.changeName=changeName;
  function changeName(name)
  {
    this.lastname=name;
  }
}
```

changeName() 函数 name 的值赋给 person 的 lastname 属性。

现在您可以试一下：

```
myMother.changeName("Doe");
```



JavaScript 类

JavaScript 是面向对象的语言，但 JavaScript 不使用类。
在 JavaScript 中，不会创建类，也不会通过类来创建对象（就像在其他面向对象的语言中那样）。
JavaScript 基于 prototype，而不是基于类的。

JavaScript for...in 循环

JavaScript for...in 语句循环遍历对象的属性。

语法

```
for (variable in object)
{
  code to be executed
}
```

注意：for...in 循环中的代码块将针对每个属性执行一次。

实例

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];
}
```

JavaScript Number 对象

JavaScript Number 对象

JavaScript 的 Number 对象是经过封装的能让你处理数字值的对象。

JavaScript 的 Number 对象由 Number() 构造器创建。

JavaScript 只有一种数字类型。

可以使用也可以不使用小数点来书写数字。

JavaScript 数字

JavaScript 数字可以使用也可以不使用小数点来书写：

实例

```
var pi=3.14;    // 使用小数点
var x=34;       // 不使用小数点
```

极大或极小的数字可通过科学（指数）计数法来写：

实例

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

所有 JavaScript 数字均为 64 位

JavaScript 不是类型语言。与许多其他编程语言不同，JavaScript 不定义不同类型的数字，比如整数、短、长、浮点等等。在 JavaScript 中，数字不分为整数类型和浮点型类型，所有的数字都是由 浮点型类型。JavaScript 采用 IEEE754 标准定义的 64 位浮点格式表示数字，它能表示最大值为 $\pm 1.7976931348623157 \times 10^{308}$ ，最小值为 $\pm 5 \times 10^{-324}$

值 (aka Fraction/Mantissa)	指数	Sign
52 bits (0 - 51)	11 bits (50 - 62)	1 bit (63)

精度

整数（不使用小数点或指数计数法）最多为 15 位。
小数的最大位数是 17，但是浮点运算并不总是 100% 准确：

实例

```
var x = 0.2+0.1; // result will be 0.30000000000000004
```

八进制和十六进制

如果前缀为 0，则 JavaScript 会把数值常量解释为八进制数，如果前缀为 0 和 "x"，则解释为十六进制数。

实例

```
var y = 0377;
var z = 0xFF;
```



绝不要在数字前面写零，除非您需要进行八进制转换。

默认情况下，JavaScript 数字为十进制显示。

但是你可以使用 `toString()` 方法 输出 16 进制、8 进制、2 进制。

实例

```
var myNumber=128;
myNumber.toString(16); // returns 80
myNumber.toString(8);  // returns 200
myNumber.toString(2);  // returns 10000000
```

无穷大 (Infinity)

当数字运算结果超过了 JavaScript 所能表示的数字上限 (溢出)，结果为一个特殊的无穷大 (infinity) 值，在 JavaScript 中以 `Infinity` 表示。同样地，当负数的值超过了 JavaScript 所能表示的负数范围，结果为负无穷大，在 JavaScript 中以 `-Infinity` 表示。无穷大值的行为特性和我们所期望的是一致的：基于它们的加、减、乘和除运算结果还是无穷大 (当然还保留它们的正负号)。

实例

```
myNumber=2;
while (myNumber!=Infinity)
{
  myNumber=myNumber*myNumber; // Calculate until Infinity
}
```

除以 0 也产生了无限:

实例

```
var x = 2/0;
var y = -2/0;
```

NaN - 非数字值

NaN 属性是代表非数字值的特殊值。该属性用于指示某个值不是数字。可以把 Number 对象设置为该值，来指示其不是数字值。

你可以使用 isNaN() 全局函数来判断一个值是否是 NaN 值。

实例

```
var x = 1000 / "Apple";
isNaN(x); // returns true
var y = 100 / "1000";
isNaN(y); // returns false
```

除以 0 是无穷大，无穷大是一个数字:

实例

```
var x = 1000 / 0;
isNaN(x); // returns false
```

提示: 在 JavaScript 中，如果参数无法被转换为数字，则返回 NaN。

数字可以是数字或者对象

数字可以私有数据进行初始化，就像 x = 123;

JavaScript 数字对象初始化数据， var y = new Number(123);

实例

```
var x = 123;
var y = new Number(123);
typeof(x) // returns Number
typeof(y) // returns Object
```

实例

```
var x = 123;  
var y = new Number(123);  
(x === y) // is false because x is a number and y is an object.
```

数字属性

- MAX_VALUE
 - MIN_VALUE
 - NEGATIVE_INFINITY
 - POSITIVE_INFINITY
 - NaN
 - prototype
 - constructor
-

数字方法

- toExponential()
- toFixed()
- toPrecision()
- toString()
- valueOf()

JavaScript 字符串 (String) 对象

JavaScript 字符串 (String) 对象

String 对象用于处理已有的字符块。

JavaScript 字符串

一个字符串用于存储一系列字符就像 "John Doe".

一个字符串可以使用单引号或双引号：

实例

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

你使用位置（索引）可以访问字符串中任何的字符：

实例

```
var character=carname[7];
```

字符串的索引从零开始, 所以字符串第一字符为 [0],第二个字符为 [1], 等等。

你可以在字符串中使用引号，如下实例：

实例

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

或者你可以在字符串中使用转义字符 (\) 使用引号：

实例

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

字符串（String）

字符串（String）使用长度属性 **length** 来计算字符串的长度：

实例

```
var txt="Hello World!";
document.write(txt.length);

var txt="ABCDEFGHJKLMNOPQRSTUVWXYZ";
document.write(txt.length);
```

JavaScript 获取字符串的长度：通过在字符串变量或字符串后面写上 .length 来获得变量中 string（字符串）值的长度。

在字符串中查找字符串

字符串使用 indexOf() 来定位字符串中某一个指定的字符首次出现的位置：

实例

```
var str="Hello world, welcome to the universe.";
var n=str.indexOf("welcome");
```

如果没找到对应的字符函数返回-1

lastIndexOf() 方法在字符串末尾开始查找字符串出现的位置。

内容匹配

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

实例

```
var str="Hello world!";
document.write(str.match("world") + "<br>");
document.write(str.match("World") + "<br>");
document.write(str.match("world!"));
```

替换内容

replace() 方法在字符串中用某些字符替换另一些字符。

实例

```
str="Please visit Microsoft!"  
var n=str.replace("Microsoft","Lenovo");
```

字符串大小写转换

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

实例

```
var txt="Hello World!";      // String  
var txt1=txt.toUpperCase();  // txt1 is txt converted to upper  
var txt2=txt.toLowerCase();  // txt2 is txt converted to lower
```

字符串转为数组

字符串使用 **string>split()**函数转为数组:

实例

```
txt="a,b,c,d,e"      // String  
txt.split(",");      // Split on commas  
txt.split(" ");      // Split on spaces  
txt.split("|");      // Split on pipe
```

特殊字符

Javascript 中可以使用反斜线 (\) 插入特殊符号，如：撇号,引号等其他特殊符号。

查看如下 JavaScript 代码:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

在 JavaScript 中，字符串的开始和停止使用单引号或双引号。这意味着，上面的字符串将被切成： We are the so-called
解决以上的问题可以使用反斜线来转义引号：

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript 将输出正确的文本字符串： We are the so-called "Vikings" from the north.

下表列出其他特殊字符，可以使用反斜线转义特殊字符：

代码	输出
\'	单引号
\"	双引号
\\	斜杆
\n	换行
\r	回车
\t	tab
\b	空格
\f	换页

字符串属性和方法

属性:

- length

- prototype
- constructor

方法:

- charAt()
- charCodeAt()
- concat()
- fromCharCode()
- indexOf()
- lastIndexOf()
- match()
- replace()
- search()
- slice()
- split()
- substr()
- substring()
- toLowerCase()
- toUpperCase()
- valueOf()

JavaScript Date（日期）对象

JavaScript Date（日期） 对象

日期对象用于处理日期和时间。

完整的 Date 对象参考手册

我们提供 JavaScript Date 对象参考手册，其中包括所有可用于日期对象的属性和方法。[JavaScript Date 对象参考手册](#)。该手册包含了对每个属性和方法的详细描述以及相关实例。

创建日期

Date 对象用于处理日期和时间。

可以通过 new 关键词来定义 Date 对象。以下代码定义了名为 myDate 的 Date 对象：

有四种方式初始化日期：

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

上面的参数大多数都是可选的，在不指定的情况下，默认参数是 0。

从 1970 年 1 月 1 日通用一天计算为 86,400,000 毫秒

实例化一个日期的一些例子：

```
var today = new Date()
var d1 = new Date("October 13, 1975 11:13:00")
var d2 = new Date(79,5,24)
var d3 = new Date(79,5,24,11,33,0)
```

设置日期

通过使用针对日期对象的方法，我们可以很容易地对日期进行操作。

在下面的例子中，我们为日期对象设置了一个特定的日期 (2010 年 1 月 14 日)：

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

在下面的例子中，我们将日期对象设置为 5 天后的日期：

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

注意：如果增加天数会改变月份或者年份，那么日期对象会自动完成这种转换。

两个日期比较

日期对象也可用于比较两个日期。

下面的代码将当前日期与 2100 年 1 月 14 日做了比较：

```
var x=new Date();
x.setFullYear(2100,0,14);
var today = new Date();
if (x>today)
```

```
{
  alert("Today is before 14th January 2100");
}
else
{
  alert("Today is after 14th January 2100");
}
```

JavaScript Array（数组）对象

JavaScript Array（数组） 对象

数组对象的作用是：使用单独的变量名来存储一系列的值。

参数

参数 `size` 是期望的数组元素个数。返回的数组，`length` 字段将被设为 `size` 的值。

参数 `element ...; elementn` 是参数列表。当使用这些参数来调用构造函数 `Array()` 时，新创建的数组的元素就会被初始化为这些值。它的 `length` 字段也会被设置为参数的个数。

返回值

返回新创建并被初始化了的数组。

如果调用构造函数 `Array()` 时没有使用参数，那么返回的数组为空，`length` 字段为 0。

当调用构造函数时只传递给它一个数字参数，该构造函数将返回具有指定个数、元素为 `undefined` 的数组。

当其他参数调用 `Array()` 时，该构造函数将用参数指定的值**初始化数组**。

当把构造函数作为函数调用，不使用 `new 运算符` 时，它的行为与使用 `new 运算符` 调用它时的行为完全一样。

Array 对象属性

属性	描述
constructor	返回对创建此对象的数组函数的引用。
length	设置或返回数组中元素的数目。
prototype	使您有能力向对象添加属性和方法。

在线实例

创建数组, 为其赋值:

实例

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

页面底部你可以找到更多的实例。

提示: 在 JavaScript 数组中, 第一个数组元素的索引值为 0, 第二个索引值为 1, 依次类推。

什么是数组?

数组对象是使用单独的变量名来存储一系列的值。

如果你有一组数据 (例如: 车名字), 存在单独变量如下所示:

```
var car1="Saab";
var car2="Volvo";
var car3="BMW";
```

然而, 如果你想从中找出某一辆车? 并且不是 3 辆, 而是 300 辆呢? 这将不是一件容易的事!

最好的方法就是用数组。

数组可以用一个变量名存储所有的值, 并且可以用变量名访问任何一个值。

数组中的每个元素都有自己的 ID, 以便它可以很容易地被访问到。

创建一个数组

创建一个数组, 有三种方法。

下面的代码定义了一个名为 myCars 的数组对象:

1: 常规方式:

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

2: 简洁方式:

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

```
3: 字面:
var myCars=["Saab","Volvo","BMW"];
```

提示: 你可以通过本站的 JavaScript 编程实战来练习如何[创建 JavaScript 数组](#)!

访问数组

通过指定数组名以及索引号码，你可以访问某个特定的元素。

以下实例可以访问 myCars 数组的第一个值:

```
var name=myCars[0];
```

以下实例修改了数组 myCars 的第一个元素:

```
myCars[0]="Opel";
```



[0] 是数组的第一个元素。[1] 是数组的第二个元素。

在一个数组中你可以有不同的对象

所有的 JavaScript 变量都是对象。数组元素是对象。函数是对象。

因此，你可以在数组中有不同的变量类型。

你可以在一个数组中包含对象元素、函数、数组:

```
myArray[0]=Date.now;
myArray[1]=myFunction;
myArray[2]=myCars;
```

数组方法和属性

使用数组对象预定义属性和方法:

```
var x=myCars.length      // the number of elements in myCars
var y=myCars.indexOf("Volvo") // the index position of "Volvo"
```

创建新方法

原型是 JavaScript 全局构造函数。它可以构建新 Javascript 对象的属性和方法。

实例：创建一个新的方法。

```
Array.prototype.ucase=function()
{
  for (i=0;i<this.length;i++)
  {this[i]=this[i].toUpperCase();}
}
```

JavaScript Boolean（布尔）对象

JavaScript Boolean（布尔） 对象

Boolean（布尔）对象用于将非布尔值转换为布尔值（true 或者 false）。

Boolean（布尔）对象是三种包装对象：Number、String 和 Boolean 中最简单的一种，它没有大量的实例属性和方法。

创建 Boolean 对象

Boolean 对象代表两个值："true" 或者 "false"

下面的代码定义了一个名为 myBoolean 的布尔对象：

```
var myBoolean=new Boolean();
```

如果布尔对象无初始值或者其值为：

- 0
- -0
- null
- ""
- false
- undefined
- NaN

那么对象的值为 false。否则，其值为 true（即使当自变量为字符串 "false" 时）！

JavaScript Math（算数）对象

JavaScript Math（算数） 对象

Math（算数）对象的作用是：执行常见的算数任务。

Math 对象

Math（算数）对象的作用是：执行普通的算数任务。

Math 对象提供多种算数值类型和函数。无需在使用这个对象之前对它进行定义。

使用 Math 的属性/方法的语法：

```
var x=Math.PI;  
var y=Math.sqrt(16);
```

注意： Math 对象无需在使用这个对象之前对它进行定义。

提示： Math 对象不能使用 new 关键字创建对象实例。直接用 “对象名.成员” 的格式来访问其属性或者方法。

算数值

JavaScript 提供 8 种可被 Math 对象访问的算数值：

你可以参考如下 Javascript 常量使用方法：

```
Math.E  
Math.PI  
Math.SQRT2  
Math.SQRT1_2  
Math.LN2  
Math.LN10  
Math.LOG2E  
Math.LOG10E
```

算数方法

除了可被 Math 对象访问的算数值以外，还有几个函数（方法）可以使用。

下面的例子使用了 Math 对象的 round 方法对一个数进行四舍五入。

```
document.write(Math.round(4.7));
```

上面的代码输出为：

5

下面的例子使用了 Math 对象的 random() 方法来返回一个介于 0 和 1 之间的随机数：

```
document.write(Math.random());
```

上面的代码输出为：

0.6581708136621066

你也可以在 JavaScript 编程实战中练习[使用 random\(\)生成随机小数](#)。

下面的例子使用了 Math 对象的 floor() 方法和 random() 来返回一个介于 0 和 11 之间的随机数：

```
document.write(Math.floor(Math.random()*11));
```

上面的代码输出为：

8

JavaScript RegExp 对象

JavaScript RegExp 对象

RegExp：是正则表达式（regular expression）的简写。

RegExp 对象用于规定在文本中检索的内容。

完整 RegExp 对象参考手册

请查看我们的 [JavaScript RegExp 对象的参考手册](#)，其中提供了可以与字符串对象一同使用的所有的属性和方法。

这个手册包含的关于每个属性和方法的用法的详细描述和实例。

什么是 RegExp？

正则表达式描述了字符的模式对象。

当您检索某个文本时，可以使用一种模式来描述要检索的内容。RegExp 就是这种模式。

简单的模式可以是一个单独的字符。

更复杂的模式包括了更多的字符，并可用于解析、格式检查、替换等等。

您可以规定字符串中的检索位置，以及要检索的字符类型，等等。

语法

```
var patt=new RegExp(pattern,modifiers);  
or more simply:  
var patt=/pattern/modifiers;
```

- 模式描述了一个表达式模型。
- 修饰符描述了检索是否是全局，区分大小写等。

RegExp 修饰符

修饰符用于执行不区分大小写和全文的搜索。

i - 修饰符是用来执行不区分大小写的匹配。

g - 修饰符是用于执行全文的搜索（而不是在找到第一个就停止查找,而是找到所有的匹配）。

实例 1

在字符串中不区分大小写找"Lenovo"

```
var str="Visit Lenovo";  
var patt1=/Lenovo/i;
```

以下标记的文本是获得的匹配的表达式：

Lenovo

实例 2

全文查找 "is"

```
var str="Is this all there is?";  
var patt1=/is/g;
```

以下标记的文本是获得的匹配的表达式：

Is this all there is?

实例 3

全文查找和不区分大小写搜索 "is"

```
var str="Is this all there is?";  
var patt1=/is/gi;
```

以下 标记的文本是获得的匹配的表达式:

Is this all there is?

test()

The test()方法搜索字符串指定的值, 根据结果并返回真或假。

下面的示例是从字符串中搜索字符 "e" :

实例

```
var patt1=new RegExp("e");  
document.write(patt1.test("The best things in life are free"));
```

由于该字符串中存在字母 "e", 以上代码的输出将是:

true

exec()

exec() 方法检索字符串中的指定值。返回值是被找到的值。如果没有发现匹配, 则返回 null。

下面的示例是从字符串中搜索字符 "e" :

实例 1

```
var patt1=new RegExp("e");  
document.write(patt1.exec("The best things in life are free"));
```

由于该字符串中存在字母 "e"，以上代码的输出将是：

e