



JavaScript

轻松学习

快速掌握 JavaScript 编程思想

极客学院出版

前言

JavaScript 是一个轻量级的，面向对象的解释编程语言，允许我们交互的建成其他静态 HTML 网页。

[ECMA-262 Specification](#) 定义了核心 JavaScript 语言的一个标准版本。

- JavaScript 是一种轻量级，解释性编程语言。
- 为了创建以网络为中心的应用程序而设计。
- 补充和集成了 Java
- 补充和集成了 HTML
- 开放和跨平台

适用人群

本教程是为初中级学者准备的，可以帮助他们理解并掌握 JavaScript 的基本语法、面向对象的设计思想以及一个高级特性。

学习前提

学习 JavaScript 的前提是你要对 HTML、CSS 有一定的了解。

目录

前言.	1
第 1 章 JavaScript 基础.	4
概述	5
语法	8
启用	11
位置结构	13
变量	16
运算符	20
If...Else.	24
Switch Case.	27
While 循环	30
For 循环	33
For...in	35
循环控制	37
函数	42
事件	45
Cookies.	49
页面重定向	55
对话框	58
Void 关键字.	60
页面打印	62
第 2 章 JavaScript 对象.	64
对象概述	65
数字对象	71

	布尔对象	73
	字符串对象	74
	数组对象	76
	日期对象	78
	算数对象	81
	正则表达式	83
	文档对象模型	87
第 3 章	JavaScript 高级	89
	错误 & 异常处理	90
	表单有效性验证	96
	动画	99
	多媒体	103
	调试	106
	图像映射	110
	浏览器兼容性	112
第 4 章	JavaScript 帮助文档	114
	快速指南	115
	JavaScript 内置函数	130



JavaScript 基础



概述

什么是 JavaScript ？

JavaScript 开始的时候是叫 LiveScript，但是 Netscape 改变了这个名字，可能是因为由 Java 而产生的兴奋使它改为了 JavaScript。JavaScript 在 1995 年的 Netscape 2.0 中以 *LiveScript* 的名字第一次出现。

JavaScript 是一个轻量级的，面向对象的解释编程语言，允许我们交互的建成其他静态 HTML 网页。

这种语言的通用核心已经被内嵌到 Netscape，Internet Explorer 和其他网络浏览器中。

[ECMA-262 Specification](#) 定义了核心 JavaScript 语言的一个标准版本。

JavaScript：

- JavaScript 是一种轻量级，解释性编程语言。
- 为了创建以网络为中心的应用程序而设计。
- 补充和集成了 Java
- 补充和集成了 HTML
- 开放和跨平台

客户端 JavaScript

客户端 JavaScript 是语言中最常见的形式。脚本应包括在或由 HTML 文件中引用的代码，以通过浏览器解释。

这意味着一个网页不再需要是静态 HTML，但可以包含与用户交互的程序，控制浏览器，和动态产生 HTML 内容。

在 JavaScript 客户端的机制中，拥有比传统的 CGI 服务器端脚本诸多优点。例如，你可能使用 JavaScript 去检查用户在窗体域中是否输入了有效的电子邮件地址。

JavaScript 的代码在用户提交表单时被执行，而且只有当所有的输入都有效时才会被提交给网络服务器。

JavaScript 可以用来捕获用户启动的事件，如单击按钮，链接导航和其他用户显式或隐式启动的操作。

JavaScript 的优点

使用 JavaScript 的优点有：

- **更少的服务器交互：**你可以在发送网页关闭到服务器之前验证用户输入。这样节省了服务器的通信，这意味着您的服务器上的负载更少。
- **即时反馈给访问者：**他们不再需要等待重新加载页面，来看看他们是否忘了输入东西。
- **增加互动性：**你可以创建反应界面，当用户将鼠标悬停在他们上面或者通过键盘激活他们时。
- **丰富的接口：**你可以使 JavaScript 包括诸如拖放组件和滑块条的项目来给你的网站访客一个丰富的接口。

使用 JavaScript 的限制

我们不能把 JavaScript 看做一个完全成熟的编程语言。它缺少下列重要特征：

- 客户端 JavaScript 不允许读取或写入文件。这是出于安全的原因。
- JavaScript 不能用于网络应用程序，因为没有可用的这种支持。
- JavaScript 没有任何多线程或多进程功能。

再次，JavaScript 是一种轻量级的，解释性编程语言，它允许你交互性建成另外的静态 HTML 页面。

JavaScript 开发工具

JavaScript 的优势之一是，它并不需要昂贵的开发工具。你可以用一个简单的文本编辑器如记事本开始。

因为它是一个网页浏览器上下文中的解释性语言，你甚至都不需要购买一个编辑器。

为了使我们的生活更简单，各个厂商都提供了非常好的 JavaScript 编辑工具。它们中的少部分在这里列出：

- **Microsoft FrontPage：**微软开发了一个非常流行的 HTML 编辑器称为 FrontPage。FrontPage 还为网页开发者提供若干 JavaScript 工具，以协助建立一个交互式网站。
- **Macromedia Dreamweaver MX：**Macromedia Dreamweaver MX 在专业网站开发人群中是一个非常流行的 HTML 和 JavaScript 编辑器。它提供了一些便利的预制的 JavaScript 组件，这些组件与数据库集成的很好，而且符合新的标准比如 XHTML 和 XML。
- **Macromedia Homesite 5：**它提供了一个很受欢迎的 HTML 和 JavaScript 编辑器，这种编辑器用来管理他们自己的网站正好。

■ 当今 JavaScript 在哪里？

ECMAScript 第 4 版标准将会是第一次更新，它将会在四年内发布。JavaScript 2.0 符合 ECMAScript 标准的第 4 版本，两者之间的区别是非常小的。

JavaScript 2.0 的具体说明可以在以下网站找到：<http://www.ecmascript.org/>

现在，Netscape' s JavaScript and Microsoft' s JScript 符合 ECMAScript 标准，尽管每种语言仍然支持不是标准中的功能。

语法

一个 JavaScript 包括那些在 HTML 中放置在 `<script> ... </script>` 标签内的 JavaScript 语句。

你可以把包含你的 JavaScript 的 `<script>` 标签放置在你的网页的任何地方，但是保存在 `<head>` 标签内是它的首选方式。

`<script>` 标签作为一个脚本，提醒浏览器程序开始解释在这些标签之间的所有的文本。所以你的 JavaScript 的简单语法将会像下列一样。

```
<script ...>
    JavaScript 代码
</script>
```

脚本标签有两个重要属性：

- **语言：**该属性制定你使用的脚本语言。通常情况下，它的值将会是 `javascript`。尽管最近的 HTML 版本（包括 XHTML，它的继任者）不再使用这个属性。
- **类型：**该属性是现在被推荐来指示所使用的脚本语言，它的值应被设置为 `"text/javascript"`。

所以你的 JavaScript 的片段应该是像这样：

```
<script language="javascript" type="text/javascript">
    JavaScript 代码
</script>
```

你的第一个 JavaScript 脚本

让我们来写出课上的例子来打印出 “Hello World”。

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write("Hello World!")
//-->
</script>
</body>
</html>
```

我们增加了一个可选的 HTML 注释，围绕着我们的 JavaScript 代码。这是为了在一个不支持 JavaScript 的浏览器中节省我们的代码。注释以 “`//-->`” 结尾。这里 “`//`” 标志着 JavaScript 中的注释，所以我们增加它来阻止一个浏览器把 HTML 的注释的结尾作为 JavaScript 代码的一部分来阅读。

另外，我们调用一个函数 `document.write`，它将一个字符串写进我们的 HTML 文档。这个函数可以被用来书写正文、HTML 或者两个一起。所以上面的代码会显示下面的结果。

```
Hello World!
```

空格和换行

JavaScript 忽略出现在 JavaScript 中的空格，制表符和换行符。

因为你可以在你的程序中自由的使用空格，制表符，换行符，所以你可以自由的用一个整洁的，一致的方法格式化和缩进你的程序，来使得代码易于阅读和理解。

分号是可选的

在 JavaScript 中简单语句通常后面跟着一个分号，正如 C、C++ 和 Java 中一样。然而，JavaScript 允许你忽略这个分号，如果你的每个陈述都放在一个单独的行。例如，下面的代码就可以不写分号。

```
<script language="javascript" type="text/javascript">
<!--
    var1 = 10
    var2 = 20
//-->
</script>
```

但是，当像下面这样书写一行时，就需要分号了。

```
<script language="javascript" type="text/javascript">
<!--
    var1 = 10; var2 = 20;
//-->
</script>
```

注意：使用分号是一个非常好的编程习惯。

区分大小写

JavaScript 是一种区分大小写的语言。这意味着语言的关键字，变量，函数名，以及任何其他标识符必须使用一致的大小写字母类型。

所以标识符 *Time*，*TIme* 和 *TIME* 在 JavaScript 中有不同的含义。

注意：当你在 JavaScript 中写变量和函数名中应该特别注意。

JavaScript 中的注释

JavaScript 支持 C 形式和 C++ 形式的注释，即：

- 在 `//` 之间的任何文本和最后一行都被视为是注释，都被 JavaScript 所忽略。
- 在字母 `/*` 和 `*/` 之间的任何文本都被视为注释。它可以是多行。
- JavaScript 还可以识别 HTML 注释的开始语句，所以可以写成 `//-->`。

例子

```
<script language="javascript" type="text/javascript">
<!--
// This is a comment. It is similar to comments in C++
/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
//-->
</script>
```

启用

目前，几乎所有浏览器均支持 JavaScript。但是，一般情况下需要用户手动的启动或禁用浏览器对 JavaScript 的支持。

本教程将指导用户了解在浏览器（IE、Firefox 和 Opera 浏览器）中启用和禁用 JavaScript 支持的流程。

IE 浏览器

如下为在 IE 浏览器中启用或禁用 JavaScript 的大致流程：

1. 在IE浏览器的菜单栏中找到“工具”选项，然后点击工具选项后选择“Internet 选项”。
2. 在上个步骤后出现的对话框中选择“安全”标签。
3. 选择“自定义级别”按钮。
4. 向下滚动下滑条找到“脚本”选项。
5. 在“活动脚本”子选项下选择“启动”选项。
6. 最后，点击“确定”按钮，并退出。

如果需要禁用 IE 浏览器对 JavaScript 功能的支持，只需要在上述补流程的第 5 个步骤，选择”禁用“选项即可。

火狐浏览器（Firefox）

如下为在火狐浏览器中启用或禁用 JavaScript 的大致流程：

1. 从火狐浏览器的菜单栏中找到“工具”选项，然后选择“选项”。
2. 从出现的对话框中选择“内容”选项。
3. 选择”启用 JavaScript“选项。
4. 最后，点击”确定“按钮，并退出。

如果需要禁用火狐浏览器对 JavaScript 功能的支持，只需要在上述补流程的第 5 个步骤，选择”禁用JavaScript“选项即可。

Opera 浏览器

如下为在 Opera 浏览器中启用或禁用 JavaScript 的大致流程：

1. 从 Opera 浏览器的菜单栏中找到“工具”选项，然后选择“偏好”。
2. 在出现的对话框中选择“高级”选项。
3. 从列表中选择“内容”。
4. 选择“启动 JavaScript”。
5. 最后，点击“确定”按钮，并退出。

如果需要禁用 Opera 浏览器对 JavaScript 功能的支持，只需要在上述流程的第 4 个步骤，选择“禁用 JavaScript”选项即可。

为不支持 JavaScript 的浏览器给出警告

在使用 JavaScript 的过程中可以通过 `<noscript>` 标记来显示警告信息。

可以按照下面的方式添加 `noscript` 代码。

```
<html>
<body>

<script language="javascript" type="text/javascript">
<!--
    document.write("Hello World!")
//-->
</script>

<noscript>
    Sorry...JavaScript is needed to go ahead.
</noscript>
</body>
</html>
```

如果用户的浏览器不支持或者没有开启 JavaScript 功能，`</noscript>` 标记的信息就会显示在用户的屏幕上。

位置结构

JavaScript 脚本可以很灵活的写在几乎 HTML 网页的任何地方。

但是，在 HTML 文件中编写的 JavaScript 脚本只可以放置在如下部分中：

1. HTML 网页的 `<head>...</head>` 里。
2. HTML 网页的 `<body>...</body>` 里。
3. HTML 网页的 `<head>...</head>` 和 `<body>...</body>` 里。
4. 外部文件里, 并且引用在 `<head>...</head>` 中。

如下章节，我们将了解如何在上述 HTML 文件的不同的地方编写 JavaScript 脚本。

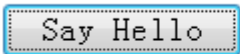
在 `<head>...</head>` 之间编写JavaScript 脚本

如果你希望在某个事件中编写脚本，比如，当用户点击一个按钮时触发一个事件。

你可以按照如下方式将脚本编写在 `<head>...</head>` 结构中

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

上述例子将产生如下效果：



在 `<body>...</body>` 之间编写 JavaScript 脚本

如果你需要一段脚本来实现页面加载后将信息内容显示在页面中的功能。

这段脚本需要编写在HTML文件的 `<body>...</body>` 部分。

这种情况下，你不需要定义任何 JavaScript 函数。

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<p>This is web page body </p>
</body>
</html>
```

上述例子将产生如下效果：

```
Advertisements

Hello World
This is web page body
```

在 `<body>` 与 `<head>` 中编写JavaScript脚本

你也可以同时将脚本编写在 `<body>` 与 `<head>` 中。

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
```

```

<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>

```

上述例子将产生如下效果:

Hello World

Say Hello

在外部文件中编写 JavaScript 脚本

当你在工作中开始广泛使用 JavaScript 后,你会发现在多 HTML 页面中重用相同的 JavaScript 是一个不错的选择。

这样,你可以不用维护多个 HTML 文件中相同的代码。

Script 标签提供了在外部文件中编写 JavaScript 脚本并引用在 HTML 文件中的功能。

下面的例子将展示如何使用 script 标签将外部 JavaScript 脚本文件引用在 HTML 文件中。

```

<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.....
</body>
</html>

```

为了实现上述功能,需要将所有的 JavaScript 源代码编写到以 “.js” 为格式后缀名的外部文本文件中,然后按照上面的方式引用入 HTML 文件中。

例如,你可以将下面的内容编写到“文件名.js”文件中,然后在 HTML 文件中引入该外部脚本文件后,你可以使用 sayHello 函数。

```

function sayHello() {
    alert("Hello World")
}

```


变量

JavaScript 数据类型

可编程语言的基本特征之一就是对常用数据类型的支持。这些数据类型可以用可编程语言来表示和操作。

按照JavaScript的语法规范，它允许有如下三类基础数据类型：

1. 数值类型：比如 123, 120.50 等。
2. 字符串类型：比如 “This text string”。
3. 布尔类型：比如 true or false。

JavaScript也支持另外两个常用类型：null 和 undefined，这两个类型均仅限定一个单一的值。

除了上述的基础数据类型，JavaScript 也支持符合数据类型，我们称之为“对象”。我们会在其他章节中学习“对象”的具体内容。

注意：JAVA语言并区分整数类型与浮点类型。JavaScript 中的数值均使用浮点值来表示。同时，按照 IEEE754 标准，JavaScript 用64位浮点格式来表示数。

JavaScript变量

和其他可编程语言相同，JavaScript 也有“变量”的概念。“变量”可以认为是有名字的容器。你可以将数据置于这些容器中，然后通过容器的名称就可以知道数据的类型。

值得注意的是，在 JavaScript 编程过程中，必须先声明一个变量，这个变量才能被使用。

此外，变量是通过 “var” 来声明的，例子如下：

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

使用 “var” 关键词也可以同时声明多个变量。例子如下：

```
<script type="text/javascript">
<!--
var money, name;
//-->
</script>
```

变量初始化就是在一个变量中存储一个值。

变量初始化可以在创建变量或者再需要使用变量时进行。

比如，需要创建一个名叫 money 的变量，然后赋值 2000.50。

或者直接在初始化的时候对变量进行赋值：

例子如下：

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

注意：“var”关键字仅能用于变量的声明或初始化。同一个 HTML 网页中，同一变量名称不能声明多次。

JavaScript 是一种无类型语言。这就是说，JavaScript 变量可以存储任何类型的值。与其他语言不同的是，我们不需要在变量声明阶段告诉变量其要存储的数据类型是什么。

变量中存储的数据类型在程序执行阶段可以被改变，这些操作都是对编程人员透明的。

JavaScript 变量作用域

一个变量的作用域就是该变量定义后在程序中的作用范围。JavaScript 变量有两个变量作用域。

1. 全局变量：全局变量具有全部整体范围的作用域，这意味着它可以在 JavaScript 代码任何地方定义。
2. 局部变量：局部变量仅在定义它的函数体内可以访问到。函数参数对于函数来说就是局部变量。

在函数体内部，局部变量可以与全局变量同名，此时是局部变量在起作用。如果局部变量或者函数参数与全局变量同名，那么此时全部变量会被隐藏到，并不会起作用。

例子如下：

```
<script type="text/javascript">
<!--
```

```

var myVar = "global"; // Declare a global variable
function checkscope() {
    var myVar = "local"; // Declare a local variable
    document.write(myVar);
}
//-->
</script>

```

该例子产生如下结果：

Local

JavaScript 变量名称

JavaScript 中变量的命名规则如下：

1. 不能使用 JavaScript 中的保留关键字来命名变量。这些保留关键字会在下一节中介绍。比如 `break` 或 `boolean`，这些命名变量是无效的。
2. JavaScript 变量名称不能以数字（0-9）开头，只能以字母或下划线来命名变量。比如 `123test` 就是无效的变量名。但是，`_123` 就是有效的变量名。
3. JavaScript 变量名称对大小写敏感。比如，`Name` 和 `name` 是两个不同的变量。

JavaScript 保留的关键字

下面是 JavaScript 中的保留关键字。他们不能用来命名 JavaScript 中的变量、行数、方法、循环标签或任何对象名称。

<code>abstract</code>	<code>else</code>	<code>instanceof</code>	<code>switch</code>
<code>boolean</code>	<code>enum</code>	<code>int</code>	<code>synchronized</code>
<code>break</code>	<code>export</code>	<code>interface</code>	<code>this</code>
<code>byte</code>	<code>extends</code>	<code>long</code>	<code>throw</code>
<code>case</code>	<code>FALSE</code>	<code>native</code>	<code>throws</code>
<code>catch</code>	<code>final</code>	<code>new</code>	<code>transient</code>
<code>char</code>	<code>finally</code>	<code>null</code>	<code>TRUE</code>
<code>class</code>	<code>float</code>	<code>package</code>	<code>try</code>
<code>const</code>	<code>for</code>	<code>private</code>	<code>typeof</code>
<code>continue</code>	<code>function</code>	<code>protected</code>	<code>var</code>
<code>debugger</code>	<code>goto</code>	<code>public</code>	<code>void</code>
<code>default</code>	<code>if</code>	<code>return</code>	<code>volatile</code>

delete	implements	short	while
do	import	static	with
double	in	super	

运算符

什么是运算符？

用表达式 `4 + 5` 能很快得到 4 和 5 这两个数的和为 9。这里 4 和 5 被称为运算数，+ 叫做运算符。JavaScript 语言支持以下类型的运算符：

- 算术运算符
- 比较运算符
- 逻辑（或关系）运算符
- 赋值运算符
- 条件（或三元）运算符

让我们依次看一下这些运算符。

算数运算符

JavaScript语言支持以下算术运算符：

给定 `A=10`, `B=20`，下面的表格解释了这些算术运算符：

运算符	描述	例子
<code>+</code>	两个运算数相加	<code>A + B = 30</code>
<code>-</code>	第一个运算数减去第二个运算数	<code>A - B = -10</code>
<code>*</code>	运算数相乘	<code>A * B = 200</code>
<code>/</code>	分子除以分母	<code>B / A = 2</code>
<code>%</code>	模数运算符，整除后的余数	<code>B % A = 0</code>
<code>++</code>	增量运算符，整数值逐次加1	<code>A++ = 11</code>
<code>--</code>	减量运算符，整数值逐次减1	<code>A-- = 9</code>

注意：`+` 运算符不仅可以用于数字相加，还可以用于把文本值或字符串变量加起来（连接起来）。例如，`"a" + 10 = "a10"`。

比较运算符

JavaScript语言支持以下比较运算符：

给定 A=10, B=20，下面的表格解释了这些比较运算符：

运算符	描述	例子
==	检查两个运算数的值是否相等，如果是，则结果为true	A == B 为false
!=	检查两个运算数的值是否相等，如果不相等，则结果为true	A != B 为true
>	检查左边运算数是否大于右边运算数，如果是，则结果为true	A > B 为false
<	检查左边运算数是否小于右边运算数，如果是，则结果为true	A < B 为true
>=	检查左边运算数是否大于或者等于右边运算数，如果是，则结果为true	A >= B 为false
<=	检查左边运算数是否小于或者等于运算数，如果是，则结果为true	A <= B 为true

逻辑运算符

JavaScript语言支持以下逻辑运算符：

给定 A=10, B=20，下面的表格解释了这些逻辑运算符：

运算符	描述	例子
&&	称为逻辑与运算符。如果两个运算数都非零，则结果为true。	A && B 为true
	称为逻辑或运算符。如果两个运算数中任何一个非零，则结果为true。	A B 为 true
!	称为逻辑非运算符。用于改变运算数的逻辑状态。如果逻辑状态为true，则通过逻辑非运算符可以使逻辑状态变为false	!(A && B) 为 false

按位运算符

JavaScript语言支持以下逻辑运算符：

给定 A=2, B=3，下面的表格解释了这些逻辑运算符

运算符	描述	例子
-----	----	----

&	称为按位与运算符。它对整型参数的每一个二进制位进行布尔与操作。	A & B = 2 .
	称为按位或运算符。它对整型参数的每一个二进制位进行布尔或操作。	A B = 3.
^	称为按位异或运算符。它对整型参数的每一个二进制位进行布尔异或操作。异或运算是指第一个参数或者第二个参数为true，并且不包括两个参数都为true的情况，则结果为true。	(A ^ B) = 1.
~	称为按位非运算符。它是一个单运算符，对运算数的所有二进制位进行取反操作。	~B = -4 .
<<	称为按位左移运算符。它把第一个运算数的所有二进制位向左移动第二个运算数指定的位数，而新的二进制位补0。将一个数向左移动一个二进制位相当于将该数乘以2，向左移动两个二进制位相当于将该数乘以4，以此类推。	A << 1 = 4.
>>	称为按位右移运算符。它把第一个运算数的所有二进制位向右移动第二个运算数指定的位数。为了保持运算结果的符号不变，左边二进制位补0或1取决于原参数的符号位。如果第一个运算数是正的，运算结果最高位补0；如果第一个运算数是负的，运算结果最高位补1。将一个数向右移动一位相当于将该数乘以2，向右移动两位相当于将该数乘以4，以此类推。	A >> 1 = 1.
>>>	称为0补最高位无符号右移运算符。这个运算符与>>运算符相像，除了位移后左边总是补0。	A >>> = 1.

赋值运算符

JavaScript语言支持以下赋值运算符：

运算符	描述	例子
=	简单赋值运算符，将右边运算数的值赋给左边运算数	C = A + B 将A+B的值赋给C
+=	加等赋值运算符，将右边运算符与左边运算符相加并将运算结果赋给左边运算数	C += A 相当于 C = C + A
-=	减等赋值运算符，将左边运算数减去右边运算数并将运算结果赋给左边运算数	C -= A 相当于C = C - A
*=	乘等赋值运算符，将右边运算数乘以左边运算数并将运算结果赋给左边运算数	C *= A 相当于C = C * A
/=	除等赋值运算符， 将左边运算数除以右边运算数并将运算结果赋值给左边运算数	C /= A 相当于 C = C / A

%=	模等赋值运算符，用两个运算数做取模运算并将运算结果赋值给左边运算数	C %= A 相当于 C = C % A
----	-----------------------------------	----------------------

注意：同样的逻辑可以应用到位运算符，因此就有<<=，>>=，>>>=，&=，|= 以及 ^=。

其他运算符

条件运算符（?:）

有一种运算符叫条件运算符。首先判断一个表达式是真或假，然后根据判断结果执行两个给定指令中的一个。条件运算符语法如下：

运算符	描述	例子
?:	条件表达式	如果条件为真 ? X值 : Y值

typeof 运算符

typeof是一个置于单个参数之前的一元运算符，这个参数可以是任何类型的。它的值是一个表示运算数的类型的字符串。

typeof运算符可以判断“数值”，“字符串”，“布尔”类型，看运算数是一个数字，字符串还是布尔值，并且根据判断结果返回true或者false。

下表是typeof运算符的返回值：

类型	Typeof返回字符串
数值	"number"
字符串	"string"
布尔	"boolean"
对象	"object"
函数	"function"
未定义	"undefined"
空	"object"

If...Else

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。因此你需要利用条件语句来使你的程序做出正确决定并且执行正确的动作。

JavaScript支持的条件语句用于基于不同的条件来执行不同的动作。下面我们看一下if...else语句。

JavaScript支持如下形式的if...else语句：

- if statement
- if...else statement
- if...else if... statement.

if 语句

if语句是基本的控制语句，能使JavaScript做出决定并且按条件执行语句。

语法

```
if (expression){  
    Statement(s) to be executed if expression is true  
}
```

这里JavaScript *expression*是需要判断的。如果返回值是`true`，执行给定的 *statement(s)*。如果表达式的值是`false`，不会执行任何语句。多数情况下，你可能会用比较运算符来做决定。

例子

```
<script type="text/javascript">  
<!--  
var age = 20;  
if( age > 18 ){  
    document.write("<b>Qualifies for driving</b>");  
}  
//-->  
</script>
```

程序运行结果如下：

Qualifies for driving

if...else 语句

if...else 语句是另一种控制语句，它能使JavaScript选择多个代码块之一来执行。

语法

```
if (expression){
    Statement(s) to be executed if expression is true
}else{
    Statement(s) to be executed if expression is false
}
```

这里JavaScript *expression*是需要判断的。如果返回值是`true`，执行if代码块给定的`statement(s)`。如果表达式的值是`false`，则执行else代码块给定的`statement(s)`。

例子

```
<script type="text/javascript">
<!--
var age = 15;
if( age > 18 ){
    document.write("<b>Qualifies for driving</b>");
}else{
    document.write("<b>Does not qualify for driving</b>");
}
//-->
</script>
```

程序运行结果如下：

Does not qualify for driving

if...else if... 语句

if...else if... 语句是一种推进形式的控制语句，它能使JavaScript选择多个代码块之一来执行。

语法

```
if (expression 1){
    Statement(s) to be executed if expression 1 is true
}else if (expression 2){
    Statement(s) to be executed if expression 2 is true
}else if (expression 3){
    Statement(s) to be executed if expression 3 is true
}else{
    Statement(s) to be executed if no expression is true
}
```

这段代码没什么特别的地方。它就是一系列 *if* 语句，只是每个 *if* 语句是前一个 *else* 语句的一部分。语句根据正确的条件被执行，如果不满足任何一个条件则执行 *else* 代码块。

例子

```
<script type="text/javascript">
<!--
var book = "maths";
if( book == "history" ){
    document.write("<b>History Book</b>");
}else if( book == "maths" ){
    document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
    document.write("<b>Economics Book</b>");
}else{
    document.write("<b>Unknown Book</b>");
}
//-->
</script>
```

程序运行结果如下：

Maths Book

Switch Case

你可以像前面章节那样用多个 `if...else if` 语句来执行多个代码块。然而，这不是最佳解决方案，尤其是当所有代码块的执行依赖于单个变量值时。

从 JavaScript 1.2 开始，你可以使用一个 `switch` 语句来处理上面提到的问题，而且这样做的效率远高于重复使用 `if...else if` 语句。

语法

`switch` 语句的基本语法是给定一个判断表达式以及若干不同语句，根据表达式的值来执行这些语句。编译器检查每个 `case` 是否与表达式的值相匹配。如果没有与值相匹配的，则执行缺省条件。

```
switch (expression)
{
    case condition 1: statement(s)
    break;
    case condition 2: statement(s)
    break;
    ...
    case condition n: statement(s)
    break;
    default: statement(s)
}
```

`break` 语句用于在特殊 `case` 的最后终止程序。如果省略掉 `break`，编译器将继续执行下面每个 `case` 里的语句。

我们将在循环控制那一章节里继续讨论 `break` 语句。

例子

下面这个例子演示了一个基本的 `while` 循环：

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
```

```

    case 'A': document.write("Good job<br />");
break;
    case 'B': document.write("Pretty good<br />");
break;
    case 'C': document.write("Passed<br />");
break;
    case 'D': document.write("Not so good<br />");
break;
    case 'F': document.write("Failed<br />");
break;
    default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>

```

程序运行结果如下：

```

Entering switch block
Good job
Exiting switch block

```

例子

看一下如果没用break语句的情况：

```

<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
    case 'A': document.write("Good job<br />");
    case 'B': document.write("Pretty good<br />");
    case 'C': document.write("Passed<br />");
    case 'D': document.write("Not so good<br />");
    case 'F': document.write("Failed<br />");
    default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>

```

程序运行结果如下：

```
Entering switch block  
Good job  
Pretty good  
Passed  
Not so good  
Failed  
Unknown grade  
Exiting switch block
```

While 循环

循环语句就是在满足一定条件的情况下反复执行某一个操作。循环语句可以有效减少程序的行数。

JavaScript 支持所有必要的循环语句，以适用于编程过程中的所有情况。

While 循环语句

While 循环是 *JavaScript* 中最基本的循环模式，下边将加以介绍。

语法如下

```
while(expression) {  
    statement  
}
```

对于 **while** 循环，当条件表达式 *expression* 的返回值为真时，则执行 “{}” 中的语句，当执行完 “{}” 中的语句后，重新判断 *expression* 的返回值，知道表达式返回值的结果为假时，退出循环。

例子

下面的例子说明了一个基本的 *while* 循环：

```
<script type="text/javascript">  
<!--var count = 0;  
document.write("Starting Loop"+"<br />");  
while(count < 10) {  
    document.write("Current Count : " + count + "<br />");  
    count++;  
}  
document.write("Loop stopped!");  
//-->  
</script>
```

运行结果如下：

```
Starting Loop  
Current Count : 0  
Current Count : 1
```

```

Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!

```

do...While 循环语句

do...while 循环和 **while** 循环非常相似，它们之间的区别是 *while* 语句为先判断条件是否成立在执行循环体，而 *do...while* 循环语句则先执行一次循环后，再判断条件是否成立。也就是说即使判断条件不成立，*do...while* 循环语句中“{}”中的程序段至少要被执行一次。

语法如下

```

do{
    statement
}while(expression);

```

注意 **do...while** 语句在结尾处多了一个分号（;）。

例子

下面编写一个 **do...while** 循环的例子：

```

<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
do{
    document.write("Current Count : " + count + "<br />");
    count++;
}while (count < 0);
document.write("Loop stopped!");
//-->
</script>

```

运行结果如下：


```
Starting Loop  
Current Count : 0  
Loop stopped!
```

For 循环

我们已经学习了几种不同的 `while` 循环，这一章我们来学习另一种更普及的循环 `for` 循环。

for 循环语句

`for` 循环是一种最简洁的循环模式，包括三个重要部分：

- *initialize* : 初始化表达式, 初始化计数器一个初始值, 在循环开始前计算初始状态。
- *test condition* : 判断条件表达式, 判断给定的状态是否为真。如果条件为真, 则执行循环体 “{}” 中的代码, 否则跳出循环。
- *iteration statement* : 循环操作表达式, 改变循环条件, 修改计数器的值。

可以将这三个部分放在同一行, 用分号隔开。

语法如下

```
for(initialize;test condition;iteration statement)
{
    statement;
}
```

例子

下面的例子说明了一个基本的 `for` 循环：

```
<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++){
    document.write("Current Count : " + count );
    document.write("<br />");
}
document.write("Loop stopped!");
//-->
</script>
```

下面就是本实例的运行结果，可见这和 **while** 循环的运行结果一样：

```
Starting Loop  
Current Count : 0  
Current Count : 1  
Current Count : 2  
Current Count : 3  
Current Count : 4  
Current Count : 5  
Current Count : 6  
Current Count : 7  
Current Count : 8  
Current Count : 9  
Loop stopped!
```

For...in

JavaScript 还支持另外一种循环模式，即 `for...in` 循环。这一种类型的循环将对象属性作为参数变量来实现循环。

对象的概念现在还没有讨论，对于 `for...in` 循环，用起来可能会觉得不舒服。但是一旦开始了解 JavaScript 中的对象概念，会发现 `for...in` 循环是非常有用的。

语法如下

```
for (variablename in object){  
    statement  
}
```

在每次迭代中将一个对象的属性赋值给变量，这个循环会持续到这个对象的所有属性都枚举完。

例子

下面的例子用于打印出一个 Web 浏览器导航对象的属性：

```
<script type="text/javascript">  
<!--  
var aProperty;  
document.write("Navigator Object Properties<br /> ");  
for (aProperty in navigator)  
{  
    document.write(aProperty);  
    document.write("<br />");  
}  
document.write("Exiting from the loop!");  
//-->  
</script>
```

运行结果：

```
Navigator Object Properties  
appCodeName  
appName  
appMinorVersion  
cpuClass
```

```
platform  
plugins  
opsProfile  
userProfile  
systemLanguage  
userLanguage  
appVersion  
userAgent  
onLine  
cookieEnabled  
mimeTypes  
Exiting from the loop!
```

循环控制

JavaScript 提供您完全控制来处理你的 *loops* 循环体和 *switch* 语句。可能会有这样一种情形, 你需要在还没有到达循环体底部的时候跳出 *loop* 循环体。也可能存在这样的情况, 当你想跳过一个代码块的一部分, 想要开始下一个迭代。

为处理所有这些情况, *JavaScript* 提供了 *break* 和 *continue* 语句。这些语句用于立即跳出来任何循环或开始下一个迭代循环。

Break 语句

Break 语句, 简要介绍了 switch 语句, 用于早期退出循环, 打破封闭的花括号。

举例

这个例子演示了在一个 while 循环体内使用 break 语句。注意是怎样在 x 等于 5 之前跳出循环体到达大括号下面的 `write(. .)` 语句。

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 20)
{
  if (x == 5){
    break; // breaks out of loop completely
  }
  x = x + 1;
  document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

产生的结果如下:

```
Entering the loop
2
3
```

```
4
5
Exiting the loop!
```

我们已经见过在switch语句里面使用break语句。

Continue 语句

Continue语句告诉解释器立即开始下一次迭代的循环和跳过剩余的代码块。

当遇到continue语句, 程序流将立即循环检查表达式, 如果条件保持真那么下个迭代开始, 否则控制跳出循环体。

举例

这个例子展示了 *continue* 语句在 *while* 循环语句的使用。当变量x达到5时, *continue* 语句用于跳过 *print* 语句。

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 10)
{
    x = x + 1;
    if (x == 5){
        continue; // skip rest of the loop body
    }
    document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

产生结果如下:

```
Entering the loop
2
3
4
6
7
8
9
```

10

Exiting the loop!

使用标签来控制流

从 *JavaScript 1.2* ,开始, 一个标签可以被用于 *break* , *continue* 语句去更精确地控制流。

一个标签仅仅是一个标识符后跟一个冒号, 应用于声明或代码块。我们将看到两个不同的例子来理解标签 *break* 和 *continue* 。

注意: *continue* 或 *break* 语句及其标签的名字之间不允许有换行符。当然标签名称和其后的的循环体之间也不应当有任何其他语句。

例 1

```
<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop:  // This is the label name
for (vari = 0; i< 5; i++)
{
document.write("Outerloop: " + i + "<br />");
innerloop:
for (var j = 0; j < 5; j++)
{
if (j > 3) break ; // Quit the innermost loop
if (i == 2) break innerloop; // Do the same thing
if (i == 4) break outerloop; // Quit the outer loop
document.write("Innerloop: " + j + " <br />");
}
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

将会产生如下的结果:

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
```



```

Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!

```

例 2

```

<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop:  // This is the label name
for (vari = 0; i< 3; i++)
{
document.write("Outerloop: " + i + "<br />");
for (var j = 0; j < 5; j++)
{
if (j == 3){
continueouterloop;
}
document.write("Innerloop: " + j + "<br />");
}
}
document.write("Exiting the loop!<br /> ");
//-->
</script>

```

结果如下：

```

Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1

```

```
Innerloop: 2  
Outerloop: 2  
Innerloop: 0  
Innerloop: 1  
Innerloop: 2  
Exiting the loop!
```

函数

函数是一组可重用的代码，你可以在你程序的任何地方调用他。这使你不需要一遍又一遍地写相同的代码。这将帮助程序员编写模块代码。你可以把大项目在许多小和易于管理的功能。你可以把你的大型程序分成许多的小型且易于管理的函数。

像任何其他高级编程语言，*JavaScript* 还支持使用函数编写模块代码所需的所有特性。

你一定在之前的章节见过 *alert()* 和 *write()* 函数。虽然我们将一次又一次的使用这些函数，但是他们已经被一次性的写在核心 *JavaScript* 。

JavaScript 使我们能够编写自己的函数。这一节将解释如何用 *JavaScript* 编写自己的函数。

函数定义

我们使用一个函数之前, 我们需要定义该函数。在 *JavaScript* 中最常见的定义一个函数的方式是使用函数关键字，紧随其后的是一个独特的函数名，参数列表(也可能是空的)，和一个被花括号包围的语句块。这里显示的基本语法：

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
    statements
}
//-->
</script>
```

例子

一个不需要参数的简单的函数定义如下：

```
<script type="text/javascript">
<!--
function sayHello()
{
    alert("Hello there");
}
```

```
//-->
</script>
```

调用函数

在脚本中调用某个函数之后, 你会需要简单的编写的函数的名称如下:

```
<script type="text/javascript">
<!--
sayHello();
//-->
</script>
```

函数参数

到目前为止, 我们已经看到了没有参数的函数, 但这些函数里面都有一个功能就是去传递不同的参数。这些被传递的参数可以在函数内被捕获且可以在这些函数上进行任何操作。在一个函数内可以把多个参数用逗号隔开。

例子

让我们在sayhello函数上做一些修改。这次会有两个参数。

```
<script type="text/javascript">
<!--
function sayHello(name, age)
{
    alert( name + " is " + age + " years old.");
}
//-->
</script>
```

注意: 我们使用+运算符连接字符串和数字。JavaScript 不介意将数字添加到字符串里。

现在, 我们可以调用如下这个函数:

```
<script type="text/javascript">
<!--
sayHello('Zara', 7 );
//-->
</script>
```

return 语句

一个 *JavaScript* 函数可以有一个可选的返回语句。这是必需的，如果你想从一个函数返回一个值。这个语句应该是函数里的最后一个语句。

例如你可以在一个函数内输入两个数字，然后你可以从函数返回调用他们相乘的结果。

例子

这个函数把两个参数连接起来并在调用程序中返回结果：

```
<script type="text/javascript">
<!--
function concatenate(first, last)
{
    var full;

    full = first + last;
    return full;
}
//-->
</script>
```

现在我们可以像下面这样调用这个函数：

```
<script type="text/javascript">
<!--
    var result;
    result = concatenate('Zara', 'Ali');
    alert(result );
//-->
</script>
```

事件

事件是什么？

JavaScript 与 *HTML* 的交互是通过事件来处理的，当用户或浏览器操纵页面时。事件就会发生。

当页面加载时, 这是一个事件。当用户单击一个按钮, 点击, 是一个事件。另一个例子的事件就像按任何键, 关闭窗口, 调整窗口等。

开发人员可以使用这些事件来执行 *JavaScript* 编码的响应, 如导致按钮关闭窗口, 信息显示给用户, 数据验证, 和任何其他类型的反应的发生。

事件都是文档对象模型 (*DOM*) 三级的一部分, 并且每个 *HTML* 元素有一定的事件可以触发 *JavaScript* 代码。

请通过这个小教程为更好地理解 *HTML* 事件参考。这里我们将看到一些例子来理解事件和 *JavaScript* 之间的关系:

onclick 事件类型

这是最常用的事件类型, 当用户点击鼠标左按钮。你可以把你的验证、警告等反对这个事件类型。

例子

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

它将产生以下结果, 然后当你点击你好按钮 *onclick* 事件将发生, 将触发 *sayHello()* 函数。

```

<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>

```

onsubmit 事件类型

另一个最重要的是 *onsubmit* 事件类型。这个事件发生在您尝试提交一个表单。所以你可以用此事件类型进行表单验证。

下面一个简单简单的例子显示其用法。我们在这里调用 *validate()* *函数之前提交表单数据到网络服务器。如果 **validate()* 函数返回 *true* 表单将提交否则不会提交数据。

例子

```

<html>
<head>
<script type="text/javascript">
<!--
function validation() {
    all validation goes here
    .....
    return either true or false
}
//-->
</script>
</head>
<body>
<form method="POST" action="t.cgi" onsubmit="return validate()">
.....
<input type="submit" value="Submit" />
</form>

```

```
</body>
</html>
```

onmouseover 和 onmouseout

这两个事件类型将会帮助你创建好良好的图像效果和文本事件。*onmouseover* 事件发生时, 当你把你的鼠标在任何元素上时, *onmouseover* 事件发生。当你把鼠标从该元素移开时, *onmouseout* 事件发生。

例子

下面的例子显示了一个部位如何反应当我们把鼠标在这个部位上:

```
<html>
<head>
<script type="text/javascript">
<!--
function over() {
    alert("Mouse Over");
}
function out() {
    alert("Mouse Out");
}
//-->
</script>
</head>
<body>
<div onmouseover="over()" onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

你可以改变不同的图像使用这两种事件类型或您可以创建帮助气框, 来帮助你的用户。

HTML 4 标准事件

标准的 *HTML 4* 事件被列在这里, 供您参考。执行以下脚本显示一个 Javascript 函数。

Event	Value	Description
onchange	script	Script runs when the element changes
onsubmit	script	Script runs when the form is submitted

onreset	script	Script runs when the form is reset
onselect	script	Script runs when the element is selected
onblur	script	Script runs when the element loses focus
onfocus	script	Script runs when the element gets focus
onkeydown	script	Script runs when key is pressed
onkeypress	script	Script runs when key is pressed and released
onkeyup	script	Script runs when key is released
onclick	script	Script runs when a mouse click
ondblclick	script	Script runs when a mouse double-click
onmousedown	script	Script runs when mouse button is pressed
onmousemove	script	Script runs when mouse pointer moves
onmouseout	script	Script runs when mouse pointer moves out of an element
onmouseover	script	Script runs when mouse pointer moves over an element
onmouseup	script	Script runs when mouse button is released

Cookies

Cookies 是什么？

Web 浏览器与服务器之间利用 HTTP 协议进行通信，而且 HTTP 是一种无状态的协议类型。但是对于商业网站，它通常需要维护一些与客户端在不同网页交互的信息。例如，一个用户在很多网页中填写信息后完成了注册，此时应该如何维护(或者缓存)众多网页中用户的信息。

在很多情况下，利用 Cookies 是最有效的方式，通过这种方式可以记录和跟踪一些用户的偏好，购物车，工资，和其他的信息，而利用这些信息可以更好的了解用户经历或者进行网站统计。

它是如何的起作用的？

服务器以 Cookies 的形式发送一些数据给访问者的浏览器。浏览器可以选择接收这些 Cookies。如果它被接收了，它就会被以纯文本的形式存储在访问者的硬盘中。接着，当访问者访问网站中的另外一个网页时，那些被缓存的 Cookies 会被发送到服务器进行检索。一旦服务器在服务端检索到该 Cookies 信息，服务器就会知道访问者本地已经缓存了的数据。

Cookies 是纯文本数据，记录了 5 个可变长度的字段：

- **Expires:** 这个字段记录 Cookies 有效时间长度。如果这个字段为空，该 Cookies 将会在用户关闭浏览器时过期，即该 Cookies 的数据不再可用。
- **Domain:** 这个字段记录网站名。
- **Path:** 这个字段记录设置 Cookies 的目录或者网页的路径。如果你想在任何目录或网页里面等够检索到 Cookies 数据这个字段可以被设置为空。
- **Secure:** 如果这个字段包含“secure”这个单词，那么 Cookies 仅仅只能被安全服务器进行检索。如果这个字段为空，就没有前面的限制。
- **Name=Value:** Cookies 以键值对的形式进行赋值和检索。

Cookies 最初是为 CGI 编程提供的，并且 Cookies 数据在网页浏览器和服务器之间是自动的传输的，因此服务器端的 CGI 脚本能够读写存储在客户端的 Cookies。

JavaScript 通过使用 Document 对象的 Cookies 属性同样可以操作 Cookies。JavaScript 可以读、创建、修改、和删除 Cookie，或者那些应用于当前网页的 Cookies。

Cookies 的存储

创建 Cookie 最简单的方式就是给 `document.cookie` 对象赋值一个字符串值，它的语法如下：

语法

```
document.cookie = "key1=value;key2=value2;expires=date";
```

这里的 `expires` 属性字段是可选的。如果你给它提供一个有效的日期或者时间值，Cookie 将会在你给定的日期或时间达到时过期，并且这之后 Cookie 的属性值都会变的不可访问。

注意：Cookie 的值不包括分号，逗号或者空格。因此，在存储 Cookie 之前，你可能需要利用 JavaScript 提供的 `escape()` 函数来对其值进行转义。如果你按照那样做的话，当你读取 Cookie 的值时，你就需要利用相应的 `unescape()` 函数。

例子

下面是一个将用户的名称记录在 Cookie 的例子。

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    if( document.myform.customer.value == "" ){
        alert("Enter some value!");
        return;
    }

    cookievalue= escape(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
```

```
<input type="button" value="Set Cookie" onclick="WriteCookie();" />
</form>
</body>
</html>
```

上面代码执行效果如下。当在文本框中输入文字并且点击了“Set Cookie”按钮就可以设置 Cookie。

现在你的机器中存在一个名称为 name 的 Cookie。你可以利用多个 key = value 键值对，他们之间用逗号分开，这样来设置多个 Cookie 信息。

读取 Cookies

读取 Cookie 就像写它一样简单，因为 document.cookie 对象的值就是 Cookie 的属性值。因此你可以利用这个字符串在任何时候对 Cookie 进行访问。

document.cookie 字符串会保存一系列用分号分开的 name = value 键值对，这里的 name 就是一个 Cookie 名称，value 是它的值。

你可以利用 *split()* 函数将字符串分解成如下形式的 key 和 value：

例子

下面的是读取前面一节设置的 Cookie 信息。

```
<html>
<head>
<script type="text/javascript">
<!--
function ReadCookie()
{
    var allcookies = document.cookie;
    alert("All Cookies : " + allcookies );

    // Get all the cookies pairs in an array
    cookiearray = allcookies.split(';');

    // Now take key value pair out of this array
    for(var i=0; i<cookiearray.length; i++){
        name = cookiearray[i].split('=')[0];
        value = cookiearray[i].split('=')[1];
        alert("Key is : " + name + " and Value is : " + value);
    }
}
```

```

}
//-->
</script>
</head>
<body>
<form name="myform" action="">
<input type="button" value="Get Cookie" onclick="ReadCookie()"/>
</form>
</body>
</html>

```

注意：

这里的 `length` 是 `Array` 类型的一个方法，该方法返回一个数组的长度。我们会在一个单独的章节再讨论数组类型。在那个时候请再好好理解这个方法。

上面的代码会有如下的效果。当你按下 “Get Cookie” 按钮，你就会看到你在上一节设置的 Cookie 信息。

注意：

在你的机器上可能已经存在一些其他的 Cookie。因此上面的代码会显示存储在你的机器上所有的 Cookies 信息。

设置 Cookies 有效日期

你可以设置有效日期并且将该有效日期与 Cookie 进行绑定，从而可以延长 Cookie 的生存时间超过当前浏览器的会话。这个可以为 `expires` 属性赋值一个日期或者时间来实现。

例子

下面的代码说明怎样设置让 Cookie 在一个月之后失效：

```

<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    var now = new Date();
    now.setMonth( now.getMonth() + 1 );
    cookievalue = escape(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    document.cookie = "expires=" + now.toUTCString() + ";";
    alert("Setting Cookies : " + "name=" + cookievalue );
}

```

```

}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>

```

删除一个 Cookie

有时你想要删除一个 Cookie，从而下次尝试读取 Cookie 信息时会返回一个空值。为了实现这个，你可以设置 Cookie 的有效生存时间为过去的某个时间的即可。

例子

下面的代码说明如何将一个 Cookie 的有效时间设置为过去的一个月来删除 Cookie。

```

<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    var now = new Date();
    now.setMonth( now.getMonth() - 1 );
    cookievalue = escape( document.myform.customer.value ) + ";";
    document.cookie="name=" + cookievalue;
    document.cookie = "expires=" + now.toUTCString() + ";";
    alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>

```

```
</body>  
</html>
```

注意：

除了设置日期你还可以利用 `setTime()` 方法来实现设置有效时间。

页面重定向

什么是页面重定向？

当你点击一个 URL 会跳转到页面 X，但是在一个页面内部点击会直接跳转到另外一个页面 Y，这里能够跳转的原因是因为页面重定向。这个是与 JavaScript 页面刷新是有区别的。

这里有许多原因可以解释为什么想要从原始页面进行重定向。例举了如下几个原因：

- 你不喜欢你现在的域名，并且你想要使用一个新的域名。有时你想将你的所有的访问者转向到你的新的网站。在这种情况下，你可以继续维护你旧的域名，同时增加单独的一页用来进行重定向，这样你的所有旧域名的访问者就可以转到新的域名。
- 你已经基于浏览器的版本构建了各种网页或者他们的名称在不同的国家不同，你可以客户端网页让用户重定向到合适的网页，而不是在服务器端进行网页的跳转。
- 搜索引擎可能已经对你的网页建立了索引。但是当你网站迁移到另外一个域名时，你不想丢失通过搜索引擎访问你网站的用户。此时你可以使用客户端网页重定向。但是请记住，不要利用这种方式欺骗搜索引擎，否则你的网站会被禁止访问。

网页重定向如何工作的？

例子 1

利用 JavaScript 在客户端进行重定向是非常简单的。为了重定向你网站的访问者，你仅仅只需要在网页代码的头部中添加一行代码，如下：

```
<head>
<script type="text/javascript">
<!--
    window.location="http://www.newlocation.com";
//-->
</script>
</head>
```


例子 2

在重定向到一个新的网页之前，你可以给访问者显示一些合适的提示信息。虽然这样可能稍微需要一点的额外加载时间。下面是一个简单的例子来实现那个功能：

```
<head>
<script type="text/javascript">
<!--
function Redirect()
{
    window.location="http://www.newlocation.com";
}

document.write("You will be redirected to main page in 10 sec.");
setTimeout('Redirect()', 10000);
//-->
</script>
</head>
```

这里的 `setTimeout()` 是 JavaScript 内置的函数，它可以在给定的时间之后执行另外一个函数。

例子 3

下面是一个基于用户的浏览器跳重定向网页到不同的网页的例子：

```
<head>
<script type="text/javascript">
<!--
var browsername=navigator.appName;
if( browsername == "Netscape" )
{
    window.location="http://www.location.com/ns.htm";
}
else if ( browsername == "Microsoft Internet Explorer")
{
    window.location="http://www.location.com/ie.htm";
}
else
{
    window.location="http://www.location.com/other.htm";
}
//-->
```

```
</script>  
</head>
```

对话框

JavaScript 支持三种重要的对话框类型。这些对话框可以用来弹出警告，或者根据用户的输入来得到确定的信息，或者得到用户输入的某一类型。

警告对话框

警告对话框是最常用的，它通常被用来给用户提示一些警告信息。比如，某个输入区域需要用户输入一些文本信息，但是用户并没有输入任何信息，那么为了使用户输入有效的信息，你可以利用警告对话框来提示警告信息，如下：

```
<head>
<script type="text/javascript">
<!--
    alert("Warning Message");
//-->
</script>
</head>
```

除了这个作用外，警告对话框也可以提示一些友好的信息。警告对话框仅仅值提供一个“OK”按钮供选择来继续执行。

确认对话框

确认对话框是最常用来获取用户对任何选项的赞成的观点。确认对话框会显示两个按钮：Ok 和 Cancel。

如果用户点击了 OK 按钮，窗口函数 `confirm()` 的返回值为 `true`。如果用户点击了 Cancel 按钮，`confirm()` 函数返回值为 `false`。你可以像如下的方式使用确认对话框：

```
<head>
<script type="text/javascript">
<!--
    var retVal = confirm("Do you want to continue ?");
    if( retVal == true ){
        alert("User wants to continue!");
        return true;
    }else{
        alert("User does not want to continue!");
        return false;
    }
//-->
</script>
</head>
```

```
    }  
    //-->  
</script>  
</head>
```

提示对话框

当你想弹出一个文本框，并且得到用户的输入数据，提示框就可以实现这个功能。因此，这个框可以与用户进行交互。用户需要填写信息，然后点击 Ok 按钮。

这种对话框通过调用 `prompt()` 函数来显示，给函数有两个形式参数 (i) 你想在框中显示的文本标签 (ii) 一个默认的显示在框中的字符串。

这种对话框提供两个按钮：OK 和 Cancel。如果用户点击 OK 按钮，窗口函数 `prompt()` 将会返回文本框中输入的值。如果用户点击 Cancel 按钮，窗口函数 `prompt()` 的返回值为 `null`。

你可以使用如下的方式来实现提示对话框：

```
<head>  
<script type="text/javascript">  
<!--  
    var retVal = prompt("Enter your name : ", "your name here");  
    alert("You have entered : " + retVal );  
    //-->  
</script>  
</head>
```

Void 关键字

JavaScript 中 `void` 是一个重要的关键字。它可以用作一个一元运算符，此时它会出现在一个操作数之前，这个操作数可以是任意类型的。

这个操作符指定要计算一个表达式但是不返回值。它的语法可能是下列之一：

```
<head>
<script type="text/javascript">
<!--
void func()
javascript:void func()

or:

void(func())
javascript:void(func())
//-->
</script>
</head>
```

例子 1

这个运算符最常用在客户端的 `javascript: URL` 中，在 URL 中可以写带有副作用的表达式，而 `void` 则让浏览器不必显示这个表达式的计算结果。

这里的 `alert('Warning!!!')` 表达式被执行了，但是它不会在当前文档处装入任何内容：

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

例子 2

另外的一个例子，下面的超级链接并不会做任何事情，因为在 JavaScript 中表达式 “0” 没有任何作用。这里的表达式 “0” 已被计算，但是它并没有在当前文档处装入任何内容：

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<a href="javascript:void(0)">Click me!</a>
</body>
```

例子 3

`void` 的另一种用法是有意的生成 *undefined* 值，如下所示：

```
<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>
```

页面打印

很多时候你会想在你 web 页面上添加一个按钮来用实际的打印机打印当前页面的内容。

JavaScript 能使用 `window` 对象的打印函数来帮你实现这个功能。

当JavaScript的打印方法 `window.print()` 执行后，就会打印当前的 web 页面。

你可以使用 `onclick` 事件直接调用这个函数，如下所示：

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<form>
<input type="button" value="Print" onclick="window.print()" />
</form>
</body>
```

这段代码会产生如下的所示的按钮，它能让你打印当前的页面。试着点击一下：

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<form>
<input type="button" value="Print" onclick="window.print()" />
</form>
</body>
```

虽然这能够满足你将页面打印出来的要求，但并不推荐这种方法将页面提供给打印设备。一个友好的打印页面，仅仅是打印一个包含文本的页面，而不包括图片，图形或者广告。

你可以采用以下方法之一来使一个页面友好的打印：

- 拷贝一份页面，并且删去不想要的文本和图形，然后从原始页面链接到能友好打印的页面。

- 如果你不想额外的拷贝一份页面，那你可以使用合适的注释来标记可以打印的文本。例如，`<!--PRINT STARTS HERE -->..... <!--PRINT ENDS HERE -->`。然后你可以使用 PERL 或者其他任何语言的脚本在后台对最终可打印的文本进行净化并展示出来。我们网站使用的是同样的方法来提供给网站访问者的打印设备打印。

怎样打印一个页面

如果某个页面并没有提供如上所示的打印工具，那你可以使用浏览器的标准工具栏来打印出 web 页面。操作流程如下所示：

文件→打印→点击确定按钮。



2

JavaScript 对象



对象概述

JavaScript 是一种面向对象编程语言（OOP）。一种语言如果可以为开发者提供四种基本功能就可以被称为面向对象：

- **封装**：把相关信息，无论数据或方法都存储在一个对象中的能力。
- **聚合**：将一个对象存储在另一个对象中的能力。
- **继承**：一个类依赖另一个类（或一些类）中的一些性质和方法的能力。
- **多态**：写一个函数或方法，这个函数或方法可以以各种不同形式工作的能力。

对象由属性组成。如果一个属性包含一个函数，它被认为是这个对象的一个方法，否则这个属性被认为成一个属性。

对象的属性

对象属性可以是任何三个基本数据类型，或抽象数据类型中的任何一个，如另一个对象。对象属性通常是在对象的方法内部使用的变量，也可以是在整个页面中使用的全局变量。

将属性添加到对象的语法是：

```
objectName.objectProperty = propertyValue;
```

示例

以下是一个简单的例子，介绍了如何使用文档对象的“title”属性来获得一个文档标题：

```
var str = document.title;
```

对象的方法

方法是函数让对象做某事或让某事作用在这个对象上。一个函数和一个方法之间，除了函数是声明的一个独立的单元，而方法是附加到某个对象上并且可以使用 `this` 关键字引用方法，除此之外几乎没有差别。

方法从将对象的内容显示到屏幕上到对一组本地属性和参数执行复杂的数学运算都很有用。

示例

以下是一个简单的例子，介绍了怎样使用文档对象的 `write()` 方法在文档中写任何内容：

```
document.write("This is test");
```

用户定义的对象

所有用户定义的对象和对象中的内置对象都是一个被称为 `Object` 的对象的后代。

`new` 运算符

`new` 运算符用于创建对象的实例。若要创建一个对象，`new` 运算符后紧接着是构造函数方法。

在以下示例中，构造函数方法是 `Object()`，`Array()`，和 `Date()`。这些函数是内置的 JavaScript 函数。

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

`Object()` 构造函数

构造函数是一个可以创建和初始化对象的函数。JavaScript 提供了名为 `Object()` 的一个特殊的构造函数来生成对象。`Object()` 构造函数将返回值赋给一个变量。

示例 1

此示例演示如何创建一个对象：

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
var book = new Object(); // Create the object
    book.subject = "Perl"; // Assign properties to the object
    book.author  = "Mohtashim";
</script>
</head>
```

```

<body>
<script type="text/javascript">
    document.write("Book name is : " + book.subject + "<br>");
    document.write("Book author is : " + book.author + "<br>");
</script>
</body>
</html>

```

示例 2

此示例演示了如何用用户定义的函数来创建一个对象。这里的 `this` 关键字用于引用被传递给函数的对象。

```

<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
function book(title, author){
    this.title = title;
    this.author = author;
}
</script>
</head>
<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
</script>
</body>
</html>

```

为对象定义的方法

前面的示例演示了如何用构造函数创建对象和分配属性。但是我们需要通过给对象分配方法来完成一个对象的定义。

示例

这里是一个简单的示例，说明了如何与一个对象一起添加一个函数。

```

<html>
<head>

```

```

<title>User-defined objects</title>
<script type="text/javascript">

// Define a function which will work as a method
function addPrice(amount){
    this.price = amount;
}

function book(title, author){
    this.title = title;
    this.author = author;
    this.addPrice = addPrice; // Assign that method as property.
}

</script>
</head>
<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    myBook.addPrice(100);
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
    document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>

```

with 关键字

with 关键字被用来作为用于引用一个对象的属性或方法的一种速记。

对象被指定成 with 关键字的参数，进而成为后面语句块的默认对象。这个对象的下的属性和方法可以不指定对象名而直接使用。

语法

```

with (object){
    properties used without the object name and dot
}

```

示例

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">

// Define a function which will work as a method
function addPrice(amount){
with(this){
    price = amount;
}
}

function book(title, author){
    this.title = title;
    this.author = author;
    this.price = 0;
    this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    myBook.addPrice(100);
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
    document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>
```

JavaScript Native 对象

JavaScript 有几个内置或 native 对象。这些对象可以在任何地方被访问，并且在任何浏览器中运行的任何操作系统的工作方式相同。

这里是所有重要的 JavaScript native 对象的列表：

- [JavaScript Number Object](#)
- [JavaScript Boolean Object](#)

- [JavaScript String Object](#)
- [JavaScript Array Object](#)
- [JavaScript Date Object](#)
- [JavaScript Math Object](#)
- [JavaScript RegExp Object](#)

数字对象

`Number` 对象表示数值日期，整数或浮点数。一般情况下，你不需要担心 `Number` 对象，因为浏览器自动将数字文本转换为数字类的实例。

语法

创建一个 `Number` 对象：

```
var val = new Number(number);
```

如果该参数不能转换为数字，它将返回为 `NaN` (Not-a-Number)。

数字属性

这里有每个属性和它的描述列表。

属性	描述
MAX_VALUE	最大的可能值在 JavaScript 中的数量可以有 1.7976931348623157E+308
MIN_VALUE	最小的可能值在 JavaScript 中的数量可以有 5E-324
NaN	等价于一个值不是一个数字。
NEGATIVE_INFINITY	比 MIN-VALUE 小的值。
POSITIVE_INFINITY	比 MAX-VALUE 大的值。
prototype	数字对象的静态属性。使用原型对象的属性来给当前文档中的数字对象分配新的属性和方法。

数字方法

数字对象只包含每个对象定义的一部分默认方法。

方法	描述
constructor()	返回创建此对象的实例的函数。默认这是数字对象。
toExponential()	将一个数字强制以指数表示法显示，即使这个数字在 JavaScript 通常规定使用标准符号表示的范围之内。
toFixed()	格式一个数为小数点右边有特定位数的小数。

<code>toLocaleString()</code>	返回当前数字的字符串值版本的格式可能根据浏览器的区域设置不同而发生变化。
<code>toPrecision()</code>	定义了总共有多少有多少为来显示一个数（包括小数点左边和右边的数）
<code>toString()</code>	返回数的值的字符串表示形式。
<code>valueOf()</code>	返回数的值。

布尔对象

Boolean 对象表示两个值，不是“真”就是“假”。

语法

创建一个 Boolean 对象：

```
var val = new Boolean(value);
```

如果省略 `value` 参数或参数是 `0`，`-0`，空，假，`NaN`，未定义，或者是空字符串（`""`），这个对象初始值为假。

布尔属性

这里列出了每个属性和它们的描述。

属性	描述
<code>constructor</code>	返回创建这个对象的布尔函数的一个引用。
<code>prototype</code>	原型属性允许您添加对象的属性和方法。

布尔方法

这里列出了每个方法和它们的描述。

方法	描述
<code>toSource()</code>	返回一个包含布尔对象来源的一个字符串；你可以使用这个字符串来创建一个等效的对象。
<code>toString()</code>	根据对象的值来返回“真”或者“假”。
<code>valueOf()</code>	返回布尔对象的原始值。

字符串对象

`String` 对象通过大量的辅助方法来操作一系列字符的组合（即字符串），这些方法隐藏了 JavaScript 字符串原始数据类型。

因为 JavaScript 可以实现原始字符串数组和字符串对象之间的自动转换，你可以调用字符串对象的任何一个辅助方法作用于原始字符串数据。

语法

创建一个 `String` 对象：

```
var val = new String(string);
```

参数 `string` 是正确编码的字符序列。

String 属性

下边列出了 `String` 的各个属性及对应的属性描述。

属性	描述
<code>constructor</code>	对创建该对象的函数的引用
<code>length</code>	字符串的长度
<code>prototype</code>	允许向对象添加属性和方法

String 对象方法

下边列出了 `String` 的一系列方法及对应的描述。

方法	描述
<code>charAt()</code>	返回在指定位置的字符
<code>charCodeAt()</code>	返回在指定的位置的字符的 Unicode 编码
<code>concat()</code>	连接字符串
<code>indexOf()</code>	检索字符串
<code>lastIndexOf()</code>	从后向前检索字符串
<code>localeCompare()</code>	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()	把字符串转换为小写
toLocaleUpperCase()	把字符串转换为大写
toLowerCase()	把字符串转换为小写
toString()	返回字符串
toUpperCase()	把字符串转换为大写
valueOf()	返回某个字符串对象的原始值

String 的 HTML 基本类型包装器

下边列出一系列方法，这些方法返回一个封装在适当的 HTML 标记中的字符串副本。

方法	描述
author()	创建一个 HTML 锚作为一个超文本的目标
big()	创建一个字符串用大号字体显示，就像使用 <big> 标签的效果
blink()	创建一个字符串闪动显示，就像使用 <blink> 标签的效果
bold()	创建一个字符串加粗显示，就像使用 标签的效果
fixed()	创建一个字符串以打字机文本显示，就像使用 <tt> 标签的效果
fontcolor()	创建一个字符串使用指定的颜色显示，就像使用 标签的效果
fontsize()	创建一个字符串使用指定的尺寸显示，就像使用 标签的效果
italics()	创建一个字符串使用斜体显示，就像使用 <i> 标签的效果
link()	创建一个 HTML 超链接，用来请求另一个 URL
small()	创建一个字符串使用小字号显示，就像使用 <small> 标签的效果
strike()	创建一个字符串使用删除线显示，就像使用 <strike> 标签的效果
sub()	创建一个字符串显示为下标，就像使用 <sub> 标签的效果
sup()	创建一个字符串显示为上标，就像使用 <sup> 标签的效果

数组对象

Array 对象用于在单个的变量中存储多个值。

语法

创建一个 Array 对象：

```
var fruits = new Array("apple", "orange", "mango");
```

数组的参数可以是一组字符串或整数。当你为数组构造函数指定一个数值参数时，数组的初始长度就被确定了。数组允许的最大长度是 4,294,967,295。

你可以通过简单赋值来创建一个数组，如下所示：

```
var fruits = ["apple", "orange", "mango"];
```

可以通过序列号（下标）来访问和设置数组内元素的值，如下所示：

- fruits[0] 是第一个元素
- fruits[1] 是第二个元素
- fruits[2] 是第三个元素

数组属性

下边列出了数组的各个属性及对应的属性描述。

属性	描述
constructor	返回对创建该对象的函数的引用
index	从零开始检索匹配的字符串
input	只见于通过正则表达式创建的数组
length	设置或返回数组中元素的数目
prototype	允许向对象添加属性和方法

Array 对象方法

下边列出了数组的一系列方法及对应的描述。

方法	描述
<code>concat()</code>	连接两个或更多的数组，并返回结果
<code>every()</code>	对数组元素应用指定的函数进行判断，当且仅当所有返回值为 <code>true</code> ，返回 <code>true</code> ，否则返回 <code>false</code>
<code>filter()</code>	创建一个新数组，数组中的元素是原数组中满足过滤函数返回值为空的元素
<code>forEach()</code>	从头到尾遍历数组，为每个元素调用制定的函数
<code>indexOf()</code>	从头到尾检索，返回给定元素在数组中的索引
<code>join()</code>	把数组的所有元素放入一个字符串。元素通过制定的分隔符进行分割
<code>lastIndexOf()</code>	从尾到头检索，返回给定元素在数组中的索引
<code>map()</code>	创建一个新数组，用来存储原数组中每个元素调用指定函数的返回值
<code>pop()</code>	删除并返回数组的最后一个元素
<code>push()</code>	向数组的末尾添加一个或更多元素，并返回新的长度。
<code>reduce()</code>	同时对数组中的两个值应用一个函数，使减少到一个单一值（从头到尾）
<code>reduceRight()</code>	同时对数组中的两个值应用一个函数，使减少到一个单一值（从尾到头）
<code>reverse()</code>	颠倒数组中元素的顺序
<code>shift()</code>	删除并返回数组的第一个元素
<code>slice()</code>	从某个已有的数组返回选定的元素
<code>some()</code>	对数组元素应用指定的函数进行判断，只有有一个返回值为 <code>true</code> ，返回 <code>true</code> ，否则返回 <code>false</code>
<code>toString()</code>	返回该对象的源代码
<code>sort()</code>	将数组中的元素进行排序
<code>splice()</code>	在数组中插入或删除元素
<code>toString()</code>	把数组转换为字符串，并返回结果
<code>unshift()</code>	将一个或多个元素添加到数组的前面，并返回新数组的长度。

日期对象

Date 对象是 JavaScript 语言的一个内置数据类型。Date 对象用 `new Date()` 创建，如下所示。

Date 对象一旦被创建，就可以使用许多方法来操作它。大多数方法只允许获取并设置对象的年、月、日、小时、分钟、秒、和毫秒字段，可以使用当地时间或世界标准时间（UTC，GMT）时间。

ECMAScript 标准要求的 Date 对象能够代表任何日期和时间，在1/1/1970之前或之后的 1 亿天内精确到毫秒。这是一个正负 273785 年的变化范围，所以 JavaScript 能够表示直到 275755 年的日期和时间。

语法

Date() 构造函数有几种不同的形式：

```
new Date()  
new Date(milliseconds)  
new Date(datestring)  
new Date(year, month, date[, hour, minute, second, millisecond])
```

注意： 方括号内的参数是可选的。

下面是参数描述：

- **No Argument:** 不带参数，Date() 构造函数创建一个日期对象，设置为当前日期和时间。
- **milliseconds:** 当传递一个数字作为参数，它作为表示日期中毫秒的内部数字，作为 `getTime()` 方法的返回值中的毫秒。例如，通过传递参数 5000 创建一个日期对象，代表 1/1/1970 午夜过去 5 秒钟。
- **datestring:** 当传递一个字符串作为参数，它必须是一个日期形式的字符串，即可以被 `Date.parse()` 方法接收的格式。
- **7 argument:** 对于上面给出的最后一种形式的构造函数，下面是每个参数的描述：
 1. **year:** 整数，表示年。为了兼容性(为了避免Y2K问题)，应该完整地指定年；使用 1998而不是 98。
 2. **month:** 整数，表示月。从 0（表示一月）开始到 11（表示十二月）。
 3. **date:** 整数，表示一个月的某一天。
 4. **hour:** 整数，表示一天的某一个小时（24小时制）。
 5. **minute:** 整数，表示时间计数的分钟片段。
 6. **second:** 整数，表示时间计数的秒片段。

7. **milliseconds**: 整数，表示时间计数的毫秒片段。

Date属性

下边列出了日期的各个属性及对应的属性描述。

属性	描述
constructor	返回对创建该对象的函数的引用
prototype	允许向对象添加属性和方法

Date方法

下边列出了日期的一系列方法及对应的描述。

方法	描述
Date()	返回当日的日期和时间
getDate()	根据本地时从Date对象返回一个月中的某一天 (1 ~ 31)
getDay()	根据本地时从Date对象返回一周中的某一天 (1 ~ 6)
getFullYear()	根据本地时从 Date 对象以四位数字返回年份
getHours()	根据本地时返回 Date 对象的小时 (0 ~ 23)
getMilliseconds()	根据本地时返回 Date 对象的毫秒 (0 ~ 999)
getMinutes()	根据本地时返回 Date 对象的分钟 (0 ~ 59)
getMonth()	根据本地时从Date对象返回月份 (1 ~ 11)
getSeconds()	根据本地时返回 Date 对象的秒数 (0 ~ 59)
getTime()	根据本地时返回 1970 年 1 月 1 日至今的毫秒数
getTimezoneOffset()	返回本地时间与格林威治标准时间 (GMT) 的分钟差
getUTCDate()	根据世界时从 Date 对象返回月中的一天 (1 ~ 31)
getUTCDay()	根据世界时从 Date 对象返回周中的一天 (0 ~ 6)
getUTCFullYear()	根据世界时从 Date 对象返回四位数的年份
getUTCHours()	根据世界时返回 Date 对象的小时 (0 ~ 23)
getUTCMilliseconds()	根据世界时返回 Date 对象的毫秒 (0 ~ 999)
getUTCMinutes()	根据世界时返回 Date 对象的分钟 (0 ~ 59)
getUTCMonth()	根据世界时从 Date 对象返回月份 (0 ~ 11)
getUTCSeconds()	根据世界时返回 Date 对象的秒钟 (0 ~ 59)
getYear()	弃用, 返回在指定的日期根据当地时间。使用getFullYear()代替。
setDate()	根据本地时设置 Date 对象中月的某一天 (1 ~ 31)

setFullYear()	根据本地时设置 Date 对象中的年份（四位数字）
setHours()	根据本地时设置 Date 对象中的小时（0 ~ 23）
setMilliseconds()	根据本地时设置 Date 对象中的毫秒（0 ~ 999）
setMinutes()	根据本地时设置 Date 对象中的分钟（0 ~ 59）
setMonth()	根据本地时设置 Date 对象中月份（0 ~ 11）
setSeconds()	根据本地时设置 Date 对象中的秒钟（0 ~ 59）
setTime()	根据本地时以毫秒设置 Date 对象
setUTCDate()	根据世界时设置 Date 对象中月份的一天（1 ~ 31）
setUTCFullYear()	根据世界时设置 Date 对象中的年份（四位数字）
setUTCHours()	根据世界时设置 Date 对象中的小时（0 ~ 23）
setUTCMilliseconds()	根据世界时设置 Date 对象中的毫秒（0 ~ 999）
setUTCMinutes()	根据世界时设置 Date 对象中的分钟（0 ~ 59）
setUTCMonth()	根据世界时设置 Date 对象中的月份（0 ~ 11）
setUTCSeconds()	根据世界时设置 Date 对象中的秒钟（0 ~ 59）
setYear()	弃用, 设置为指定的日期根据当地时间。使用 setFullYear() 代替。
toString()	把 Date 对象的日期部分转换为字符串
toGMTString()	弃用, 将日期转换为一个字符串, 使用互联网格林尼治时间约定。使用 toUTCString() 代替。
toLocaleDateString()	根据本地时间格式, 把 Date 对象的日期部分转换为字符串
toLocaleFormat()	将日期转换为一个字符串, 使用格式化字符串。
toLocaleString()	根据本地时间格式, 把 Date 对象转换为字符串
toLocaleTimeString()	根据本地时间格式, 把 Date 对象的时间部分转换为字符串
toSource()	返回一个字符串代表一个等价的日期对象的源码, 您可以使用这个值来创建一个新的对象
toString()	把 Date 对象转换为字符串
getTimeString()	把 Date 对象的时间部分转换为字符串
toUTCString()	根据世界时, 把 Date 对象转换为字符串
valueOf()	返回 Date 对象的原始值

Date 静态方法

方法	描述
Date.parse()	返回 1970 年 1 月 1 日午夜到指定日期（字符串）的毫秒数
Date.UTC()	根据世界时返回 1970 年 1 月 1 日到指定日期的毫秒数

算数对象

Math 对象提供针对数学常量的属性、方法和功能。

不同于其他的全局对象，**Math** 不是一个构造函数。**Math** 的所有属性和方法都是静态的，无需创建它，通过把 **Math** 作为对象使用就可以调用其所有属性和方法。

因此，可以定义常量 **pi** 为 **Math.PI**，也可以调用 **sin** 函数 **Math.sin(x)**，其中 *x* 是方法的参数。

语法

这是 **Math** 中调用属性和方法的简单语法。

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

Math属性

这是 **Math** 的各个属性及对应的属性描述的列表。

属性	描述
E	返回算术常量 e，即自然对数的底数（约等于 2.718）
LN2	返回 2 的自然对数（约等于 0.693）
LN10	返回 10 的自然对数（约等于 2.302）
LOG2E	返回以 2 为底的对数（约等于 1.414）
LOG10E	返回以 10 为底的对数（约等于 0.434）
PI	返回圆周率（约等于 3.14159）
SQRT1_2	返回 2 的平方根的倒数（约等于 0.707）
SQRT2	返回2的平方根（约等于 1.414）

Math方法

这是 **Math** 的各个方法及对应的属性描述的列表。

属性	描述
abs()	返回数的绝对值
acos()	返回数的反余弦值

<code>asin()</code>	返回数的反正弦值
<code>atan()</code>	以介于 $-\pi/2$ 与 $\pi/2$ 弧度之间的数值来返回 x 的反正切值
<code>atan2()</code>	返回从 x 轴到点 (x, y) 的角度（介于 $-\pi/2$ 与 $\pi/2$ 弧度之间）
<code>ceil()</code>	对数进行上舍入
<code>cos()</code>	返回数的余弦
<code>exp()</code>	返回 e 的指数
<code>floor()</code>	对数进行下舍入
<code>log()</code>	返回数的自然对数（底为 e ）
<code>max()</code>	返回 x 和 y 中的最高值
<code>min()</code>	返回 x 和 y 中的最低值
<code>pow()</code>	返回 x 的 y 次幂
<code>random()</code>	返回 $0 \sim 1$ 之间的随机数
<code>round()</code>	把数四舍五入为最接近的整数
<code>sin()</code>	返回数的正弦
<code>sqrt()</code>	返回数的平方根
<code>tan()</code>	返回角的正切
<code>toSource()</code>	返回该对象的源代码

正则表达式

正则表达式是一个对象，这个对象描述一种字符模式。

JavaScript `RegExp` 类代表正则表达式，字符串和 `RegExp` 都定义了方法，在方法中使用正则表达式来执行文本中强大的模式匹配和搜索替换功能。

语法

正则表达式可以被 `RegExp()` 构造函数定义，如下所示：

```
var pattern = new RegExp(pattern, attributes);  
or simply  
var patter = /pattern/attributes;
```

参数描述：

- **pattern**：是一个字符串，指定了正则表达式的模式或其他正则表达式。
- **attributes**：是一个可选的字符串，包含属性 “g”、“i” 和 “m”，分别用于指定全局匹配、区分大小写的匹配和多行匹配。

方括号

方括号 (`[]`) 用于正则表达式的上下文中时有特殊意义，用来查找一系列字符。

表达	描述
<code>[...]</code>	查找方括号之间的任何字符
<code>[^...]</code>	查找任何不在方括号之间的字符
<code>[0-9]</code>	查找任何从 0 至 9 的数字
<code>[a-z]</code>	查找任何小写 a 到小写 z 的字符
<code>[A-Z]</code>	查找任何大写 A 到大写 Z 的字符
<code>[a-Z]</code>	查找任何小写 a 到大写 Z 的字符

上面所示的范围为一般情况；还可以使用范围 `(0-3)` 匹配任何从 0 到 3 的十进制数字，或范围 `(b-v)` 来匹配任何从小写 b 到小写 v 的字符。

量词

方括号括起来的字符序列或单个字符出现的频率或位置可以用一个特殊的符号来表示。每个特殊字符都有一个特定的含义。+、*、? 和 \$ 符号都遵循一个字符序列模式。

表达	描述
p+	匹配任何包含至少一个 p 的字符串
p*	匹配任何包含零个或多个 p 的字符串
p?	匹配任何包含零个或一个 p 的字符串
p{N}	匹配包含 N 个 p 的序列字符串
p{2,3}	匹配包含 2 或 3 个 p 的序列的字符串
p{2,}	匹配包含至少 2 个 p 的序列的字符串
p\$	匹配任何结尾为 p 的字符串
^p	匹配任何开头为 p 的字符串

例子

下面的例子会帮助理清字符匹配的概念。

表达	描述
[^a-zA-Z]	匹配任何不包含从 a 到 z 和从 A 到 Z 中任何字符的字符串
p.p	匹配任何以一个 p 开始、其次是任意字符、紧随其后的是另一个 p 的字符串
^. {2}\$	匹配任何包含两个字符的字符串
(.*)	匹配任何封闭在 和 内的字符串
p(hp)*	匹配任何包含一个 p、紧随其后的零个或多个 hp 序列的字符串

原义字符

字符	描述
Alphanumeric	它自己
\0	查找 NUL 字符 (\u0000)
\t	查找制表符 (\u0009)
\n	查找换行符 (\u000A)
\v	查找垂直制表符 (\u000B)
\f	查找换页符 (\u000C)
\r	查找回车符 (\u000D)

\xnn	指定的以十六进制数 nn 表示的拉丁字符;例如 \x0A 和 \n 表示的一样
\uxxxx	查找以十六进制数 xxxx 规定的 Unicode 字符，例如 \u0009 和 \t 表示的一样
\cX	控制字符 ^X；例如 \cJ 相当于换行符 \n

元字符

元字符：在一个字母字符之前加上一个反斜杠，使这个组合具有特殊的含义。

例如, 您可以使用 ' \d ' 元字符搜索大量资金数额: /([\d]+)000/, 这里 \d 将寻找任何数值字符的字符串。

下面是元字符的列表，使用 PERL 风格的正则表达式表达。

字符	描述
.	单个字符
\s	空白字符（空格、制表符、换行符）
\S	非空白字符
\d	数字字符（0-9）
\D	非数字字符
\w	单词字符（a-z, A-Z, 0-9, _）
\W	非单词字符
[\b]	一个文字退格(特殊情况)
[aeiou]	匹配一个在给定集合内的字符
[^aeiou]	匹配一个不在给定集合内的字符
[foo bar baz]	匹配任何指定的备选方案

修饰

几个可用的 regexp 修饰符，它能使你的工作更容易，比如大小写敏感、搜索多个行等。

字符	描述
i	执行对大小写不敏感的匹配
m	执行多行匹配
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）

RegExp属性

这是 RegExp 的各个属性及对应的属性描述的列表。

属性	描述
constructor	指定创建一个对象原型的函数
global	RegExp 对象是否具有标志 g
ignoreCase	RegExp 对象是否具有标志 i
lastIndex	一个整数，标示开始下一次匹配的字符位置
multiline	RegExp 对象是否具有标志 m
source	正则表达式的源文本

RegExp方法

这是 RegExp 的各个方法及对应的属性描述的列表。

方法	描述
exec()	检索字符串中指定的值。返回找到的值，并确定其位置
test()	检索字符串中指定的值。返回 true 或 false
toSource	返回一个对象字面值代表指定的对象;您可以使用这个值来创建一个新的对象。
toString()	返回一个代表指定对象的字符串。

文档对象模型

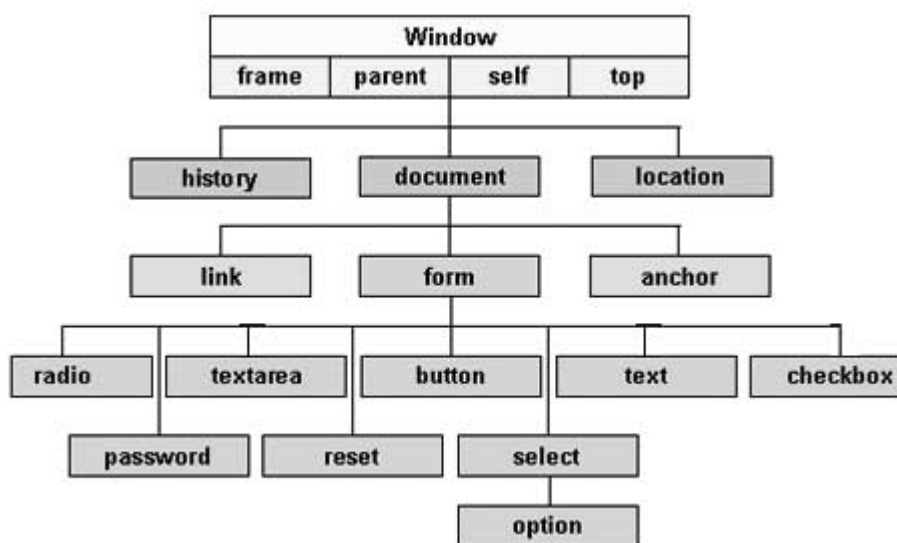
在浏览器窗口中的每个网页都可以看作一个对象。

文档对象就代表了在浏览器窗口中显示的那个 HTML 文档。文档对象有很多属性表示其他的一些对象，通过对这些属性的操作，我们就可以访问或者修改文档的内容。

对文档对象的访问和修改的方式被称为**文档对象模型**，或者称为**DOM**。这些对象是按照继承关系组织在一起的。这个继承关系结构用来将网页文档对象联系在一起。

- **窗口对象**：继承关系中的最顶层。它是继承结构中对顶层的元素。
- **文档对象**：每一个被加载到窗口中的 HTML 对象变成了文档对象。文档对象中包含了网页的内容。
- **表单对象**：任何以 `<form>...</form>` 标签圈起来的内容是表单对象。
- **表单控制元素**：这个表单对象包含表单中定义的所有元素，例如文本框，按钮，单选按钮和多选按钮。

如下是一些重要对象的继承结构图：



有几个文档对象是已经实现了的。下面的部分详细的介绍这些文档对象，并且介绍你如何利用这些对象访问和修改文档内容。

- **传统的文档对象模型**：这个模型是在 JavaScript 语言早期版本中引入的。它能够很好的被所有的浏览器支持，但是仅仅只被允许访问文档某些确定的关键区域，比如，表单，表单元素和图像。
- **W3C 文档对象模型**：这种文档对象模型允许访问和修改所有的文档内容，而且它是被万维网组织标准化的。这种模型基本上被所有的现代浏览器支持。

- IE4 文档对象模型：这种文档对象模型是由微软的 IE 浏览器的第四版本而引入的。IE 5和之后的浏览器版本能够支持 W3C 文档对象模型的大多数特性。

文档对象的兼容性

如果你想写个脚本，当 W3C 文档对象模型可用的时候，利用这种模型，而当 IE 4文档对象模型有效的时候，利用 IE 4文档对象模型，因此，你可以利用兼容性测试的方法，这种方法首先检测已经存在的方法或者属性，从而去决定浏览器是否能够兼容你想要的文档对象模型。例如：

```
if (document.getElementById) {  
  
    // If the W3C method exists, use it  
  
}  
  
else if (document.all) {  
  
    // If the all[] array exists, use it  
  
}  
  
else {  
  
    // Otherwise use the legacy DOM  
  
}
```



3

JavaScript 高级



错误 & 异常处理

程序中存在三种错误：（a）语法错误（b）运行期错误（c）逻辑错误：

语法错误

语法错误同样也被称为解析错误，对于传统的编程语言，该错误出现先编译的时候，对于 JavaScript 该错误出现在解释时期。例如，下面的代码会引起语法错误，因为在一行中缺少一个圆括号：

```
<script type="text/javascript">
<!--
window.print(
//-->
</script>
```

当在 JavaScript 中出现了语法错误的时候，仅仅只是在同一个包含该语法错误的进程才会受到影响，其他的进程中的代码执行不会受到影响，尽管他们依赖的代码中包含错误。

运行期错误

运行期错误也被称为异常，通常在编译或者解释之后运行时会出现。例如，下面的代码会造成运行期错误，因为它试图调用一个不存在的方法：

```
<script type="text/javascript">
<!--
window.printme();
//-->
</script>
```

异常出现时会影响进程创建时的正常执行，但是允许对于其他的 JavaScript 进程则可以继续正常执行。

逻辑错误

逻辑错误是最难被追踪的错误类型。这些错误并不是语法或者运行时错误造成的，而是由于你在程序运行的逻辑上出现错误，从而导致你的脚本程序并不能得到你想要的结果。

你并不能捕获这些错误，因为它取决于你的业务逻辑需求，你想要在你的程序中实现怎样的逻辑处理。

try...catch...finally 语句

JavaScript 的最新版本中添加了异常处理的功能。它实现了 try...catch...finally 结构和 throw 操作用来处理异常。你可以捕获程序员和运行期产生的异常，但是对于用户不能捕获 JavaScript 的语法错误。下面是 try...catch...finally 语法块：

```
<script type="text/javascript">
<!--
try {
    // Code to run
    [break;]
} catch ( e ) {
    // Code to run if an exception occurs
    [break;]
}[ finally {
    // Code that is always executed regardless of
    // an exception occurring
}]
//-->
</script>
```

try 语句块后面必须跟着一个 catch 语句块或者一个 finally 语句块(或者同时包含他俩的一个语句块)。当在 try 语句块内部产生了一个异常，这个异常就会被赋值给 e，接着 catch 语句块被执行。而可选的 finally 语句块在 try/catch 之后一定会被执行。

例子

如下是一个例子，我们尝试调用一个不存在的方法，这个会导致异常的产生。我们首先看下没有 try...catch 语句时运行情况：

```
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    alert("Value of variable a is : " + a );

}
```

```
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>
```

接下来利用 `try...catch` 语句尝试去捕获程序的异常，并且给用户提示一个友好的消息。如果你不想让用户看见这个错误，你也可以不让这个消息产生。

```
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    try {
        alert("Value of variable a is : " + a );
    } catch ( e ) {
        alert("Error: " + e.description );
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>
```

你还可以使用 `finally` 语句，它会在 `try/catch` 语句之后必定执行。如下例子所示：

```
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
```

```

{
    var a = 100;
    try {
        alert("Value of variable a is : " + a );
    }catch ( e ) {
        alert("Error: " + e.description );
    }finally {
        alert("Finally block will always execute!" );
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>

```

throw 语句

你可以使用 **throw** 语句产生一个内置的异常或者你自己定制的异常。之后这些异常可以被捕获，而且捕获后你可以采取合适的操作。

下面的是一个例子，展示如何使用 **throw** 语句。

```

<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;
    var b = 0;

    try{
        if ( b == 0 ){
            throw( "Divide by zero error." );
        }else{
            var c = a / b;
        }
    }
    }catch ( e ) {

```

```

        alert("Error: " + e );
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>

```

你可以在一个函数里面利用字符串，整型，布尔类型或者一个对象引起异常，接着你可以在同一个方法里面捕获这个异常，或者在其他的函数里面利用 `try...catch` 语句块捕获异常。

onerror() 方法

onerror 事件句柄是 JavaScript 中添加的第一个为了方便错误处理的特性。无论任何时候在网页中产生了异常，窗口对象就会触发 **error** 事件。例如：

```

<html>
<head>
<script type="text/javascript">
<!--
window.onerror = function () {
    alert("An error occurred.");
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>

```

onerror 事件句柄提供了三个信息用来准确的表示错误的特性：

- 错误消息。 浏览器显示给定错误的相关信息。

- URL。 错误出现的文件。
- 行数。 在指定的 URL 中造成错误的行数。

如下是一个例子显示如何得到上面的那些信息。

```
<html>
<head>
<script type="text/javascript">
<!--
window.onerror = function (msg, url, line) {
    alert("Message : " + msg );
    alert("url : " + url );
    alert("Line number : " + line );
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc();" />
</form>
</body>
</html>
```

你可以用任何你认为更好的方式来显示得到的信息。

你可以使用 **onerror** 方法来显示一个错误消息，以免在加载图片时出现任何的问题：

```

```

如果程序中产生错误，你可以在很多的 HTML 标签中使用 **onerror** 来显示合适的消息。

表单有效性验证

当用户输完了所有必要的的数据并且点击了提交按钮，接着就会进行表单的有效性进行验证，表单验证通常发生在服务器端。如果用户输入的一些数据存在错误，或者没有输入数据，服务器就必须返回给客户端所有的数据，并且要求用户重新提交正确的数据。这个过程确实是处理比较费时，而且给服务器造成很大压力。

JavaScript 提供了一种方式在客户端提交数据到服务器之前验证表单数据的有效性。通常利用两个函数来验证表单数据的有效性。

- 基本有效性验证： 首先，表单必须确保所有需要输入的数据的区域有数据。这个需要循环的对表单中需要输入数据的区域进行检验数据是否为空。
- 数据格式验证： 其次，被输入的数据必须要检查它的格式和值是否正确。这就需要更多的逻辑测试。

我们举个例子来说明有效性验证的过程。如下是个简单的例子：

```
<html>
<head>
<title>Form Validation</title>
<script type="text/javascript">
<!--
// Form validation code will come here.
//-->
</script>
</head>
<body>
<form action="/cgi-bin/test.cgi" name="myForm"
      onsubmit="return(validate());">
<table cellspacing="2" cellpadding="2" border="1">
<tr>
<td align="right">Name</td>
<td><input type="text" name="Name" /></td>
</tr>
<tr>
<td align="right">EMail</td>
<td><input type="text" name="EMail" /></td>
</tr>
<tr>
<td align="right">Zip Code</td>
<td><input type="text" name="Zip" /></td>
</tr>
<tr>
```

```

<td align="right">Country</td>
<td>
<select name="Country">
<option value="-1" selected>[choose yours]</option>
<option value="1">USA</option>
<option value="2">UK</option>
<option value="3">INDIA</option>
</select>
</td>
</tr>
<tr>
<td align="right"></td>
<td><input type="submit" value="Submit" /></td>
</tr>
</table>
</form>
</body>
</html>

```

基本表单验证

首先我们会展示如何进行基本的表单有效性验证。上面的代码中我们调用了 `validate()` 函数验证数据有效性，当事件 `onsubmit` 发生的时候。下面是对 `validate()` 函数的实现。

```

<script type="text/javascript">
<!--
// Form validation code will come here.
function validate()
{

    if( document.myForm.Name.value == "" )
    {
        alert( "Please provide your name!" );
        document.myForm.Name.focus() ;
        return false;
    }

    if( document.myForm.Email.value == "" )
    {
        alert( "Please provide your Email!" );
        document.myForm.Email.focus() ;
        return false;
    }

    if( document.myForm.Zip.value == "" ||
        isNaN( document.myForm.Zip.value ) ||

```

```

        document.myForm.Zip.value.length != 5 )
    {
        alert( "Please provide a zip in the format #####." );
        document.myForm.Zip.focus() ;
        return false;
    }
    if( document.myForm.Country.value == "-1" )
    {
        alert( "Please provide your country!" );
        return false;
    }
    return( true );
}
//-->
</script>

```

数据格式有效性验证

接下来我们将会演示在将数据提交到服务器端之前我们如何验证数据的有效性。

这个例子演示如何验证用户输入的邮箱地址的有效性，因为输入的邮箱格式中必须包含 @ 符号和一个点号(.)。并且，符号 @ 不能作为作为邮箱地址的第一个字符，在 @ 符号之后和点号之前至少要有一个字符：

```

<script type="text/javascript">
<!--
function validateEmail()
{

    var emailID = document.myForm.Email.value;
    atpos = emailID.indexOf("@");
    dotpos = emailID.lastIndexOf(".");
    if (atpos < 1 || ( dotpos - atpos < 2 ))
    {
        alert("Please enter correct email ID")
        document.myForm.Email.focus() ;
        return false;
    }
    return( true );
}
//-->
</script>

```

动画

你可以利用 JavaScript 创造一些复杂的运动，包括下面但不限于下面的：

- 烟花式
- 淡出效果
- 旋转进入或者旋转推出
- 整个页面进入或者整个页面出去
- 对象移动

这个教程将讲解如何利用 JavaScript 创建一些基本的运动。

JavaScript 可以用来移动一些文档对象模型元素（``，`<div>` 或者其他 HTML 元素）在页面附近，这个是通过一些逻辑等式或者函数来决定的。

JavaScript 提供如下的几种比较常用的函数来进行动画编程。

- `setTimeout(function, duration)`：这个函数在规定的时间 *duration* 到达时，会调用参数中的函数 *function*。
- `setInterval(function, duration)`：这个方法调用了之后会清除所有 `setTimeout()` 函数设定的计时器。

JavaScript 能够设置一系列文档模型对象的属性值，包括该对象在屏幕中的位置。你可以通过设置 *top* 和 *left* 等对象的属性值，从而让该对象放在屏幕中任何位置。如下是该语法的一个简单例子：

```
// Set distance from left edge of the screen.  
object.style.left = distance in pixels or points;  
  
or  
  
// Set distance from top edge of the screen.  
object.style.top = distance in pixels or points;
```

手动动画

让我们利用文档对象模型的对象的属性值实现一个简单的动画，JavaScript 的函数如下：

```
<html>  
<head>  
<title>JavaScript Animation</title>
```

```

<script type="text/javascript">
<!--
var imgObj = null;
function init() {
    imgObj = document.getElementById('myImage');
    imgObj.style.position= 'relative';
    imgObj.style.left = '0px';
}
function moveRight() {
    imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
}
window.onload =init;
//-->
</script>
</head>
<body>
<form>

<p>Click button below to move the image to right</p>
<input type="button" value="Click Me" onclick="moveRight();" />
</form>
</body>
</html>

```

如下是对上面例子的解释：

- 我们利用 JavaScript 的函数 `getElementById()` 得到了文档对象模型对象，接着把它赋值给一个全局变量 `imgObj`。
- 我们定义了一个初始化函数 `init()` 用来初始化 `imgObj` 对象，初始化时设置了该对象的 `position` 和 `left` 属性的值。
- 初始化函数在网页窗口加载的时候被调用。
- 最后，我们调用 `moveRight()` 函数增加该对象到左边边界的距离为 10 个像素点。你也可以设置该对象到左边的距离为负值。

自动动画

在上面的例子中我们已经看到如何让一张图片在每次点击之后向右移动。我们可以通过利用 JavaScript 中的函数 `setTimeout()` 让它自动执行这个操作：

```

<html>
<head>

```

```

<title>JavaScript Animation</title>
<script type="text/javascript">
<!--
var imgObj = null;
var animate ;
function init() {
imgObj = document.getElementById('myImage');
imgObj.style.position= 'relative';
imgObj.style.left = '0px';
}
function moveRight() {
imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
animate = setTimeout(moveRight,20); // call moveRight in 20msec
}
function stop() {
clearTimeout(animate);
imgObj.style.left = '0px';
}
window.onload =init;
//-->
</script>
</head>
<body>
<form>

<p>Click the buttons below to handle animation</p>
<input type="button" value="Start" onclick="moveRight();" />
<input type="button" value="Stop" onclick="stop();" />
</form>
</body>
</html>

```

这里我们增加了一些新的知识：

- `moveRight()` 函数通过调用 `setTimeout()` 函数设置 `imgObj` 对象的位置。
- 我们添加了一个新的函数 `stop()`，它的作用是用来清除 `setTimeout()` 函数设置的计时器和让文档对象处于它的初始位置。

伴随鼠标事件的翻转

如下是一个简单的例子演示由鼠标事件引起的图片翻转：

```

<html>
<head>
<title>Rollover with a Mouse Events</title>
<script type="text/javascript">
<!--
if(document.images){
    var image1 = new Image();      // Preload an image
    image1.src = "/images/html.gif";
    var image2 = new Image();      // Preload second image
    image2.src = "/images/http.gif";
}
//-->
</script>
</head>
<body>
<p>Move your mouse over the image to see the result</p>
<a href="#" onMouseOver="document.myImage.src=image2.src;"
    onMouseOut="document.myImage.src=image1.src;">
    
</a>
</body>
</html>

```

让我们来看看这里的不同点：

- 在加载这个网页的时候，if 语句是否存在图片对象。如果没有有效的图片对象，那么下面的语句块不会被执行。
- Image() 构造方法创建和预加载一个新的图片对象称为 image1。
- src 属性值被赋值为外部图片文件的路径名称，即是 /images/html.gif。
- 同样的方式，我们创建了 image2 对象，并且给它赋值为 /images/http.gif。
- 符号 # 禁用链接，这样浏览器中点击该链接就不会发生跳转。这个链接表示的是一个图片。
- onMouseOver 事件句柄在用户的鼠标移动到链接上时会被触发，同样，用户鼠标离开图片链接时会触发 onMouseOut 事件句柄。
- 当鼠标移动到图片上时，HTTP 上的图片就会从第一个变到第二个。用鼠标离开图片时，最开始的那副图片就会被显示。
- 当鼠标从图片链接上离开时，初始的 html.gif 图片会再次出现在屏幕上面。

多媒体

JavaScript 导航对象包含的子对象称为插件。这个对象是一个数组，每个插件会在浏览器上安装一个条目。导航器和插件对象只被网景，Firefox 和 Mozilla 支持。

下面是一个示例，列出了所有安装浏览器上的插件：

```
<html>
<head>
<title>List of Plug-Ins</title>
</head>
<body>
<table border="1">
<tr>
<th>Plug-in Name</th>
<th>Filename</th>
<th>Description</th>
</tr>
<script language="JavaScript" type="text/javascript">
for (i=0; i<navigator.plugins.length; i++) {
    document.write("<tr><td>");
    document.write(navigator.plugins[i].name);
    document.write("</td><td>");
    document.write(navigator.plugins[i].filename);
    document.write("</td><td>");
    document.write(navigator.plugins[i].description);
    document.write("</td></tr>");
}
</script>
</table>
</body>
</html>
```

检查插件

每个插件在数组中都有一个入口。每个入口有以下属性：

- 名字：是插件的名称。
- 文件名：是加载安装插件的可执行文件。
- 描述：是对于插件的描述，由开发人员提供。

- **mimetype**: 是有一个入口的数组, 这个入口是被插件支持的 MIME 类型的入口。

您可以使用这些属性在脚本中找到已安装的插件, 然后使用 JavaScript 可以适当运行的多媒体文件如下:

```
<html>
<head>
<title>Using Plug-Ins</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
media = navigator.mimeTypes["video/quicktime"];
if (media){
    document.write("<embed src='quick.mov' height=100 width=100>");
}
else{
    document.write("<img src='quick.gif' height=100 width=100>");
}
</script>
</body>
</html>
```

注意: 我们使用 HTML `<embed>` 标记中嵌入一个多媒体文件。

多媒体控制

让我们举一个真实的例子, 它在几乎所有的浏览器里面都有效:

```
<html>
<head>
<title>Using Embedded Object</title>
<script type="text/javascript">
<!--
function play()
{
    document.demo.Play();
}
function stop()
{
    if (document.demo.IsPlaying()){
        document.demo.StopPlay();
    }
}
function rewind()
```

```
{
    if (document.demo.IsPlaying()){
document.demo.StopPlay();
    }
    document.demo.Rewind();
}
//-->
</script>
</head>
<body>
<embed id="demo" name="demo"
src="http://www.amrood.com/games/kumite.swf"
width="318" height="300" play="false" loop="false"
pluginspage="http://www.macromedia.com/go/getflashplayer"
swliveconnect="true">
</embed>
<form name="form" id="form" action="#" method="get">
<input type="button" value="Start" onclick="play();" />
<input type="button" value="Stop" onclick="stop();" />
<input type="button" value="Rewind" onclick="rewind();" />
</form>
</body>
</html>
```

调试

当你写程序的时候很有可能出现错误，这在脚本中被称为一个bug。

发现和修复 bug 的过程叫做调试，是一个正常的开发过程的一部分。本节讨论的工具和技术，可以帮助您进行任务调试。

IE 浏览器里面的错误信息

跟踪错误的最基本的方法是在您的浏览器中打开错误信息。默认情况下，当一个错误发生在页面，Internet Explorer 在状态栏显示错误图标：

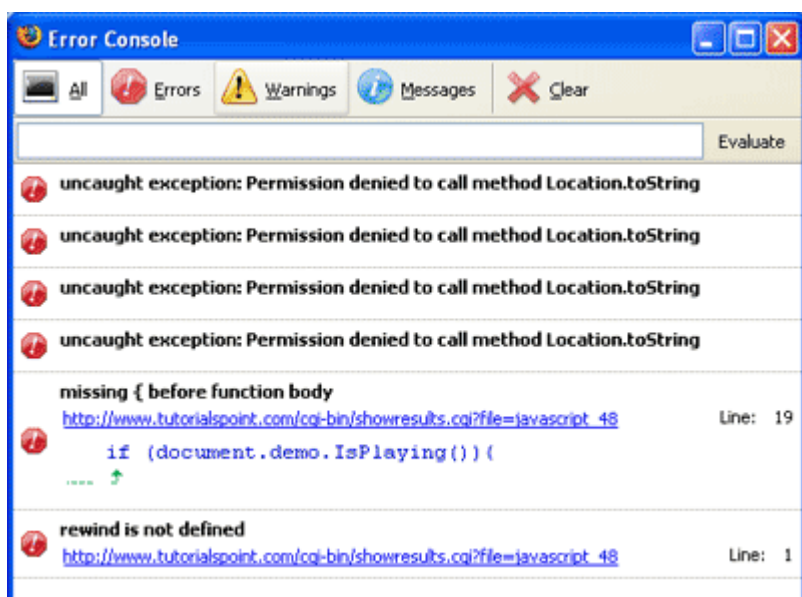


图片 3.1 image1

双击这个图标将您带到一个对话框，这个对话框将会显示发生的特定的错误信息。

因为这个图标很容易忽视，当发生错误时，Internet Explorer 提供了选项自动显示错误对话框。

要启用这个选项，选择 Tools --> Internet Options --> Advanced tab。最后通过检查显示一个关于每一个脚本错误框的通知，选项如下所示：

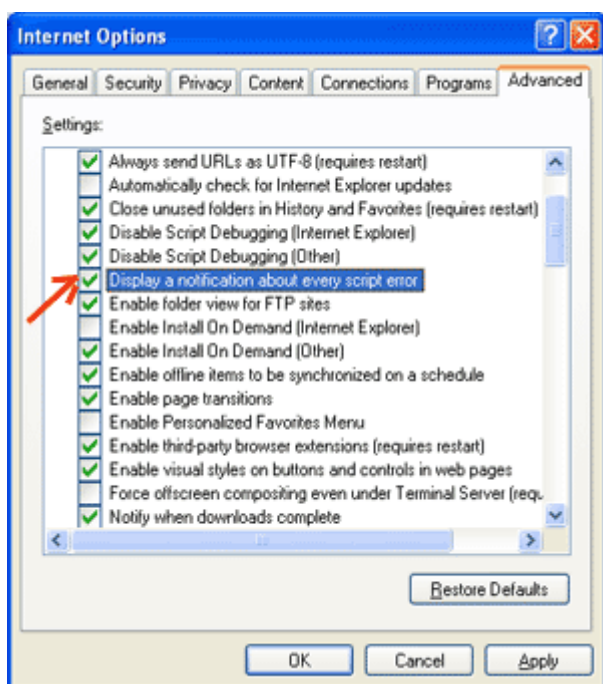


图片 3.2 image2

在 Mozilla 或 Firefox 里的错误消息

火狐 Netscape 和 Mozilla 等浏览器将错误消息发送到一个叫做 JavaScript 控制台或错误控制台的特殊窗口。查看控制台，选择 Tools-->Error Console or Web Development。

不幸的是，因为这些浏览器在一个错误发生时，没有给出视觉的标示，你必须保持打开控制台，才能看您的脚本执行的错误。



图片 3.3 image3

错误通知

错误通知，出现在控制台或通过 IE 浏览器对话框里的错误是语法和运行时错误的结果。这些错误通知会显示错误所在的行号当错误发生时。

如果您使用的是 Firefox 浏览器，那么你可以点击在错误控制台脚本中可以获得的错误从而到达有错误的那一行。

如何调试脚本

有多种方法来调试 JavaScript

用一个 JavaScript 验证器

检查您的 JavaScript 代码出现的奇怪的错误的一种方法是通程序来运行它，检查它以确保它是有效的。这是官方语言的语法规则。这些程序被称为验证解析器，也可以简称为验证器，通常是存在于商业HTML和JavaScript编辑器。

最方便的 JavaScript 验证器是 Douglas Crockford's JavaScript Lint, Douglas Crockford's JavaScript Lint 可以免费在线使用。

简单地访问这个网页,将 JavaScript 代码(只有 JavaScript)粘贴到文本区域,并单击 jslint 按钮。这个程序将通过您的 JavaScript 代码解析,确保任何变量和函数定义遵循正确的语法。它还将检查 JavaScript 语句,比如 if 和 while语句,以确保他们也遵循正确的格式

添加调试代码到您的程序

您可以在你的程序中使用 `alert()` 或 `document.write()` 方法去调试你的代码。例如,你应该这样写:

```
var debugging = true;
var whichImage = "widget";
if( debugging )
    alert( "Calls swapImage() with argument: " + whichImage );
var swapStatus = swapImage( whichImage );
if( debugging )
    alert( "Exits swapImage() with swapStatus=" + swapStatus );
```

当他们出现的时候,通过检测程序内容和`alert()`s的命令,你可以非常容易的确定你的程序是否有错误。

使用一个JavaScript调试器

调试器是一个应用程序,该应用程序确保各种各样的脚本在程序员的控制下执行。脚本调试器通过一个界面提供细粒度的控制的状态,允许您检查和设置值以及控制流执行。

一旦一个脚本被加载到一个调试器,它可以每次运行一行或指示停止在某个断点。一旦停止执行,程序员可以检查脚本和它的状态变量,以确定是否有地方出了问题。你也可以观察变量的变化值。

最新版本的 Mozilla JavaScript 调试器(代号为完 Venkman) Mozilla 和 Netscape 浏览器可以下载 <http://www.hacksrus.com/~ginda/venkman>

对开发人员有用的技巧

这里有几个技巧，您可以使用在你的脚本中用于减少错误的数量，同时使调试过程更容易一些。

- 记住要使用大量的注释。注释使您能够解释为什么你以这样的方式书写脚本和解释特别困难的部分代码。
- 总是使用缩进，使代码易于阅读。缩进语句也使你更容易匹配开始和结束标记，花括号和其他 HTML 和脚本元素。
- 编写模块代码。只要有可能，用函数的形式来组织你的语句。函数使你能够阻止相关语句，并以最小的努力测试和重用的部分代码以。
- 命名变量和函数的方法要一致。尝试使用足够长的时间有意义的名称，描述的内容变量或函数的目的。
- 命名变量和函数时使用一致的语法。换句话说，让他们所有的小写或大写；如果你喜欢 Camel-Back 符号，一致地使用它。
- 以模块化的方式测试长脚本。换句话说，不要试图在编写完成整个脚本之后再测试它的任何部分。写一段，使它能够工作之后，再添加代码的下一部分。
- 使用描述性的变量和函数名称和避免使用单个字符的名称。
- 注意你的引号。记住，引号用于对字符串和这两个引号必须相同的风格(单引号或双)。
- 主义你的等于号。你不应该一个 `=` 用于比较的目的。
- 声明变量显式地使用 `var` 关键字。

图像映射

您可以使用 JavaScript 来创建客户端的图像映射。usemap 启用客户端图像映射的属性定义的 标记和特殊的 <map> 和 <area> 扩展标签。

一般情况下，用 <map> 将形成映射的图像插入到页面，此外它带有一个额外的属性称为 usemap。usemap 属性的值是 <map> element 上的 name 属性的值。

<map> 元素实际上创建的地图图片，通常遵循后直接 元素。它为 <area /> element 充当一个容器，<area /> element 用于定义可点击的热点。<map> element 只有一个属性，即名称属性，用于标识映射。这就是为什么 元素知道使用哪个 <map> element。

<area> 元素指定坐标定义边界的形状和每一个可点击的热点。

当鼠标移动到图像的不同部分时，以下结合 imagemap 和 JavaScript 在一个文本框里面产生一个消息。

```
<html>
<head>
<title>Using JavaScript Image Map</title>
<script type="text/javascript">
<!--
function showTutorial(name){
    document.myform.stage.value = name
}
//-->
</script>
</head>
<body>
<form name="myform">
    <input type="text" name="stage" size="20" />
</form>
<!-- Create Mappings -->

<map name="tutorials">
    <area shape="poly"
coords="74, 0, 113, 29, 98, 72, 52, 72, 38, 27"
href="/perl/index.htm" alt="Perl Tutorial"
target="_self"
onMouseOver="showTutorial('perl')'"
onMouseOut="showTutorial('')"/>
    <area shape="rect"
```

```
coords="22, 83, 126, 125"
href="/html/index.htm" alt="HTML Tutorial"
target="_self"
onMouseOver="showTutorial('html')"
onMouseOut="showTutorial('')"/>
    <area shape="circle"
coords="73, 168, 32"
href="/php/index.htm" alt="PHP Tutorial"
    target="_self"
onMouseOver="showTutorial('php')"
onMouseOut="showTutorial('')"/>
</map>
</body>
</html>
```


浏览器兼容性

重要的是要了解不同浏览器之间的差异，以处理每个预计会出现的问题。所以重要的是要知道哪个浏览器运行在您的 Web 页面。

获得目前运行在 Web 页面的浏览器的信息，使用内置的 `navigator` 对象。

导航属性

有几个导航相关属性，您可以使用您的 Web 页面。下面是一个列表的名称和描述：

属性	描述
appName	这个属性是一个包含浏览器 code name 的字符串，比如 Netscape 是 Netscape 的 code name，Microsoft Internet Explorer 是 Internet Explorer 的code name。
Appversion	这个属性是一个字符串，其中包含浏览器的版本以及其他有用的信息，比如它的语言和兼容性。
language	这个属性包含两个字母的缩写表示这种语言，使用这种方式的浏览器只有 Netscape。
mimTypes[]	这个属性是一个数组，其中包含所有客户端支持的 MIME 类型。只有 Netscape。
platform[]	这个属性是一个字符串，其中包含浏览器编译的平台。“Win32” 32 位 Windows 操作系统。
plugins[]	这个属性是一个数组，其中包含的所有插件已经安装在客户机上。只有 Netscape 公司。
userAgent[]	这个属性是一个字符串，其中包含浏览器的代码名称和浏览器版本。这个值被发送到原始服务器用于识别客户端。

导航方法

有几个 Navigator-specific 方法。这里是一个与其相关的列表的：

方法	描述
javaEnabled()	这个方法确定是否启用了 JavaScript 客户端。如果启用了 JavaScript，那么该方法将返回 true，否则返回 false。
plugins.refresh	这个方法使新安装的插件可用，并且用所有新插件的名称去填充插件数组。 Netscape 公司 only。
preference(name, value)	这种方法允许标记脚本去获取和设置一些 Netscape 的偏好。如果省略第二个参数，那么该方法将返回的值指定的偏好；否则，使用系统默认的值。 Netscape 公司 only。

taintEnabled()	这个方法返回 true, 如果启用了数据污染, 否则, 则返回 false。
----------------	--

浏览器检测

有一个简单的 JavaScript 可以用来发现浏览器的名称, 其后相应的 HTML 页面可以被提供给用户。

```
<html>
<head>
<title>Browser Detection Example</title>
</head>
<body>
<script type="text/javascript">
<!--
var userAgent = navigator.userAgent;
var opera = (userAgent.indexOf('Opera') != -1);
var ie = (userAgent.indexOf('MSIE') != -1);
var gecko = (userAgent.indexOf('Gecko') != -1);
var netscape= (userAgent.indexOf('Mozilla') != -1);
var version = navigator.appVersion;

if (opera){
    document.write("Opera based browser");
    // Keep your opera specific URL here.
}else if (gecko){
    document.write("Mozilla based browser");
    // Keep your gecko specific URL here.
}else if (ie){
    document.write("IE based browser");
    // Keep your IE specific URL here.
}else if (netscape){
    document.write("Netscape based browser");
    // Keep your Netscape specific URL here.
}else{
    document.write("Unknown browser");
}

// You can include version to along with any above condition.
document.write("<br /> Browser version info : " + version );
//-->
</script>
</body>
</html>
```



4

JavaScript 帮助文档



快速指南

JavaScript 是什么？

JavaScript 具有如下特征：

- 轻量级的解释型（代码不需要经过预编译）可编程语言。
- 用于网络应用开发的脚本语言。
- 可以与 Java、HTML 语言互补集成。
- 开放且跨平台。

JavaScript 语法

一段 JavaScript 脚本由包含在网页页面中 `<script>... </script>` 标签内的 JavaScript 语句组成。

编程人员可以随意将由 `<script>` 标签包含着的 JavaScript 脚本置于网页的任意位置，但是通常都会放在 `<head>` 标签内。

`<script>` 标记用来让浏览器明白这个标签之间的语句需要作为脚本来解释。所以，JavaScript 脚本可以简单的按照如下形势来表示：

```
<script ...>
  JavaScript code
</script>
```

`<script>` 标签有两个很重要的属性：

- 语言（language）：这个属性指定你正在使用什么脚本语言。一般来说，指的都是 JavaScript。尽管最近版本的 HTML (XHTML 及后续版本) 会逐步不再使用这个属性。
- 类型（type）：该属性用于表明脚本语言类别的。通常应该设置为 “*text/javascript*”。

所以 JavaScript 片段是如下形式：

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

尝试写第一个 JavaScript 脚本

我们试着利用 JavaScript 脚本写一个打印 “Hello word” 的功能。

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write("Hello World!")
//-->
</script>
</body>
</html>
```

上面的脚本会显示如下结果：

```
Hello World!
```

空格和换行符

JavaScript编译器会忽略掉脚本中的所有空白、缩进符、换行符。

因此，程序员可以在脚本中自用使用空格、缩进和换行符，这样程序员就可以随意缩进格式化程序格式，以便代码更易于阅读和理解。

分号是可选的

与 C、C++ 和 Java一样，通常 JavaScript 语句以分号结尾。但是，JavaScript 允许在每行只有一句脚本的情况下省略分号。比如，下方的脚本就是可以省略分号的：

```
<script language="javascript" type="text/javascript">
<!--
    var1 = 10
    var2 = 20
//-->
</script>
```

但是一行中存在多个脚本语句时，分号是不能省略的，比如：

```
<script language="javascript" type="text/javascript">
<!--
    var1 = 10; var2 = 20;
//-->
</script>
```

注意：使用分号是一个比较实用的编程习惯

区分大小写

JavaScript 是一门大小写敏感的语言。这意味着关键词、变量、函数名及其他任何标识符在输入时都要保持一样的书写格式。

所以，*Time*，*TIme* 和 *TIME* 在 JavaScript 中表示不同的意思。

注意：在编写 JavaScript 脚本时一定要注意变量和函数名的书写。

注释

JavaScript 同样支持 C 和 C++ 那样的注释方式：

- 任何以 “//” 开头的行均被认为是注释内容而被编译器忽略掉。
- 任何以 “/” 开头且以 “/” 结尾的内容均被视为注释内容，且这样的内容可以是多行的。
- JavaScript 也识别 HTML 格式的注释开始标识 “<!--” 不会被 JavaScript 识别，应该书写成 “//-->”。

JavaScript 脚本应置于 HTML 文件的何处

在 HTML 文件中何处书写 JavaScript 脚本是非常灵活的。但是，通常情况下都会将脚本书写在 HTML 文件中的如下位置内：

- `<head>...</head>` 内
- `<body>...</body>` 内
- `<body>...</body>` 和 `<head>...</head>` 同时书写
- 以外部文件的方式包含在 `<head>...</head>` 内

数据类型

JavaScript中可以容纳下述三种数据类型：

- 数值类型：比如123, 120.50等
- 字符串类型：比如 “This text string”
- 布尔类型：比如true or false

JavaScript也支持另外两个常用类型：null和undefined，这两个类型均仅限定一个单一的值。

JavaScript变量

和其他可编程语言相同，JavaScript也有“变量”的概念。“变量”可以认为是有名字的容器。你可以将数据置于这些容器中，然后通过容器的名称就可以知道数据的类型。

值得注意的是，在JavaScript编程过程中，必须先声明一个变量，这个变量才能被使用。

此外，变量是通过“var”来声明的，例子如下：

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

JavaScript变量作用域

一个变量的作用域就是该变量定义后在程序中的作用范围。JavaScript变量有两个变量作用域。

1. 全局变量：全局变量具有全部整体范围的作用域，这意味着它可以再JavaScript代码任何地方定义。
2. 局部变量：局部变量仅在定义它的函数体内可以访问到。函数参数对于函数来说就是局部变量。

JavaScript变量名称

JavaScript中变量的命名规则如下：

- 不能使用JavaScript中的保留关键字来命名变量。这些保留关键字会在下一节中介绍。比如break 或 boolean，这些命名变量是无效的。
- JavaScript变量名称不能以数字（0-9）开头，只能以字母或下划线来命名变量。比如123test就是无效的变量名。但是，_123就是有效的变量名。
- JavaScript变量名称对大小写敏感。比如，Name和name是两个不同的变量。

JavaScript保留的关键字

下面是JavaScript中的保留关键字。他们不能用来命名JavaScript中的变量、行数、方法、循环标签或任何对象名称。

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

算数运算符

JavaScript语言支持以下算术运算符

给定 A=10, B=20，下面的表格解释了这些算术运算符：

运算符	描述	例子
+	两个运算数相加	A + B = 30
-	第一个运算数减去第二个运算数	A - B = -10
*	运算数相乘	A * B = 200

/	分子除以分母	B / A = 2
%	模数运算符，整除后的余数	B % A = 0
++	增量运算符，整数值逐次加1	A++ = 11
--	减量运算符，整数值逐次减1	A-- = 9

比较运算符

JavaScript语言支持以下比较运算符

给定 A=10, B=20，下面的表格解释了这些比较运算符：

运算符	描述	例子
==	检查两个运算数的值是否相等，如果是，则结果为true	A == B 为false
!=	检查两个运算数的值是否相等，如果不相等，则结果为true	A != B 为true
>	检查左边运算数是否大于右边运算数，如果是，则结果为true	A > B 为false
<	检查左边运算数是否小于右边运算数，如果是，则结果为true	A < B 为true
>=	检查左边运算数是否大于或者等于右边运算数，如果是，则结果为true	A >= B 为false
<=	检查左边运算数是否小于或者等于运算数，如果是，则结果为true	A <= B 为true

逻辑运算符

JavaScript语言支持以下逻辑运算符

给定 A=10, B=20，下面的表格解释了这些逻辑运算符

运算符	描述	例子
&&	称为逻辑与运算符。如果两个运算数都非零，则结果为true。	A && B 为true
	称为逻辑或运算符。如果两个运算数中任何一个非零，则结果为true。	A B 为true
!	称为逻辑非运算符。用于改变运算数的逻辑状态。如果逻辑状态为true，则通过逻辑非运算符可以使逻辑状态变为false	!(A && B) 为false

按位运算符

JavaScript语言支持以下逻辑运算符

给定 A=2, B=3，下面的表格解释了这些逻辑运算符

运算符	描述	例子
&	称为按位与运算符。它对整型参数的每一个二进制位进行布尔与操作。	A & B = 2 .
	称为按位或运算符。它对整型参数的每一个二进制位进行布尔或操作。	A B = 3.
^	称为按位异或运算符。它对整型参数的每一个二进制位进行布尔异或操作。异或运算是指第一个参数或者第二个参数为true，并且不包括两个参数都为true的情况，则结果为true。	(A ^ B) = 1.
~	称为按位非运算符。它是一个单运算符，对运算数的所有二进制位进行取反操作。	~B = -4 .
<<	称为按位左移运算符。它把第一个运算数的所有二进制位向左移动第二个运算数指定的位数，而新的二进制位补0。将一个数向左移动一个二进制位相当于将该数乘以2，向左移动两个二进制位相当于将该数乘以4，以此类推。	A << 1 = 4.
>>	称为按位右移运算符。它把第一个运算数的所有二进制位向右移动第二个运算数指定的位数。为了保持运算结果的符号不变，左边二进制位补0或1取决于原参数的符号位。如果第一个运算数是正的，运算结果最高位补0；如果第一个运算数是负的，运算结果最高位补1。将一个数向右移动一位相当于将该数乘以2，向右移动两位相当于将该数乘以4，以此类推。	A >> 1 = 1.
>>>	称为0补最高位无符号右移运算符。这个运算符与>>运算符相像，除了位移后左边总是补0.	A >>> = 1.

赋值运算符

JavaScript语言支持以下赋值运算符

运算符	描述	例子
=	简单赋值运算符，将右边运算数的值赋给左边运算数	C = A + B 将A+B的值赋给C
+=	加等赋值运算符，将右边运算符与左边运算符相加并将运算结果赋给左边运算数	C += A 相当于 C = C + A
-=	减等赋值运算符，将左边运算数减去右边运算数并将运算结果赋给左边运算数	C -= A 相当于C = C - A
*=	乘等赋值运算符，将右边运算数乘以左边运算数并将运算结果赋给左边运算数	C *= A 相当于C = C * A
/=	除等赋值运算符， 将左边运算数除以右边运算数并将运算结果赋值给左边运算数	C /= A 相当于 C = C / A

%=	模等赋值运算符，用两个运算数做取模运算并将运算结果赋值给左边运算数	C %= A 相当于 C = C % A
----	-----------------------------------	----------------------

其他运算符

条件运算符

有一种运算符叫条件运算符。首先判断一个表达式是真或假，然后根据判断结果执行两个给定指令中的一个。条件运算符语法如下

运算符	描述	例子
? :	条件表达式	如果条件为真 ? X值 : Y值

typeof 运算符

typeof 是一个置于单个参数之前的一元运算符，这个参数可以是任何类型的。它的值是一个表示运算数的类型的字符串。

typeof 运算符可以判断“数值”，“字符串”，“布尔”类型，看运算数是一个数字，字符串还是布尔值，并且根据判断结果返回true或者false。

if 语句

if语句是基本的控制语句，能使JavaScript做出决定并且按条件执行语句。

```
if (expression){
    Statement(s) to be executed if expression is true
}
```

if...else 语句

if...else 语句是另一种控制语句，它能使JavaScript选择多个代码块之一来执行。

```
if (expression){
    Statement(s) to be executed if expression is true
}else{
    Statement(s) to be executed if expression is false
}
```

if...else if... 语句

if...else if... 语句是一种推进形式的控制语句，它能使 JavaScript 选择多个代码块之一来执行。

```
if (expression 1){
    Statement(s) to be executed if expression 1 is true
}else if (expression 2){
    Statement(s) to be executed if expression 2 is true
}else if (expression 3){
    Statement(s) to be executed if expression 3 is true
}else{
    Statement(s) to be executed if no expression is true
}
```

switch-case 语句

语句的基本语法是给定一个判断表达式以及若干不同语句，根据表达式的值来执行这些语句。编译器检查每个case是否与表达式的值相匹配。如果没有与值相匹配的，则执行default缺省条件。

```
switch (expression)
{
    case condition 1: statement(s)
break;
    case condition 2: statement(s)
break;
    ...
    case condition n: statement(s)
break;
    default: statement(s)
}
```

while 循环

循环是 JavaScript 中最基本的循环模式，下边将加以介绍。

```
while(expression){
    statement
}
```

do-while 循环语句

`do...while` 循环和 `while` 循环非常相似，它们之间的区别是 `while` 语句为先判断条件是否成立在执行循环体，而 `do...while` 循环语句则先执行一次循环后，再判断条件是否成立。也就是说即使判断条件不成立，`do...while` 循环语句中 “{}” 中的程序段至少要被执行一次。

```
do{  
    statement  
}while(expression);
```

for 循环

循环是一种最简洁的循环模式，包括三个重要部分

- initialize : 初始化表达式, 初始化计数器一个初始值，在循环开始前计算初始状态。
- test condition : 判断条件表达式，判断给定的状态是否为真。如果条件为真，则执行循环体 “{}” 中的代码，否则跳出循环。
- iteration statement : 循环操作表达式，改变循环条件，修改计数器的值。

可以将这三个部分放在同一行，用分号隔开。

```
for(initialize;test condition;iteration statement)  
{  
    statement;  
}
```

for in 语句

```
for (variablename in object){  
    statement  
}
```

在每次迭代中将一个对象的属性赋值给变量，这个循环会持续到这个对象的所有属性都枚举完。

break 语句

Break语句用于提前跳出循环，打破封闭的循环体。

continue 语句

Continue语句告诉解释器立即开始下一次迭代的循环和跳过剩余的代码块。

当遇到continue语句, 程序流将立即循环检查表达式, 如果条件保持真那么下个迭代开始, 否则控制跳出循环体。

函数定义

使用一个函数之前, 我们需要定义该函数。在 *JavaScript* 中最常见的定义一个函数的方式是使用函数关键字, 紧随其后的是一个独特的函数名, 参数列表(也可能是空的), 和一个被花括号包围的语句块。这里显示的基本语法

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
    statements
}
//-->
</script>
```

调用函数

在脚本中调用某个函数之后, 你会需要简单的编写的函数的名称如下

```
<script type="text/javascript">

<!--
sayHello();
//-->
</script>
```

异常处理

异常通常使用 *try/catch/finally* 结构来进行处理。

```
<script type="text/javascript">
<!--
try {
    statementsToTry
```

```

    } catch ( e ) {
        catchStatements
    } finally {
        finallyStatements
    }
//-->
</script>

```

try部分后面必须紧随catch后者finally部分（同时或者其一）。当在catch部分中发生异常事件后，异常会被存储于e变量中，然后catch部分会被执行。而finally部分会在try/catch之后必然执行。

警告对话框

警告对话框是最常用的，它通常被用来给用户提示一些警告信息。比如，某个输入区域需要用户输入一些文本信息，但是用户并没有输入任何信息，那么为了使用户输入有效的信息，你可以利用警告对话框来提示警告信息，如下：

```

<head>
<script type="text/javascript">
<!--
    alert("Warning Message");
//-->

</script>
</head>

```

确认对话框

确认对话框是最常用来获取用户对任何选项的赞成的观点。确认对话框会显示两个按钮：**Ok** 和 **Cancel**。你可以像如下的方式使用确认对话框：

```

<head>
<script type="text/javascript">
<!--
    var retVal = confirm("Do you want to continue ?");
    if( retVal == true ){
        alert("User wants to continue!");
        return true;
    }else{
        alert("User does not want to continue!");
        return false;
    }
}

```

```
//-->
</script>
</head>
```

提示对话框

你可以使用如下的方式来实现提示对话框

```
<head>
<script type="text/javascript">
<!--
    var retVal = prompt("Enter your name : ", "your name here");
    alert("You have entered : " + retVal );
//-->
</script>
</head>
```

页面重定向

利用 JavaScript 在客户端进行重定向是非常简单的。为了重定向你的网站，你仅仅只需要在网页代码的头部中添加一行代码，如下

```
<head>
<script type="text/javascript">
<!--
    var retVal = prompt("Enter your name : ", "your name here");
    alert("You have entered : " + retVal );
//-->
</script>
</head>
```

Void关键字

JavaScript 中 void 是一个重要的关键字。它可以用作一个一元运算符，此时它会出现在一个操作数之前，这个操作数可以是任意类型的。

这个操作符指定要计算一个表达式但是不返回值。它的语法可能是下列之一

```
<head>
<script type="text/javascript">
<!--
```



```

void func()
javascript:void func()

or:

void(func())
javascript:void(func())
//-->
</script>
</head>

```

页面打印

JavaScript 能使用 window 对象的打印函数 **print** 来帮你实现这个功能。

当 JavaScript 的打印方法 **window.print()** 执行后，就会打印当前的 web 页面。

你可以使用 onclick 事件直接调用这个函数，如下所示

```

<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<form>
<input type="button" value="Print" onclick="window.print()" />
</form>
</body>

```

Cookies 的存储

创建 Cookie 最简单的方式就是给 document.cookie 对象赋值一个字符串值，它的语法如下

```
document.cookie = "key1=value;key2=value2;expires=date";
```

Cookies 的读取

读取 Cookie 就像写它一样简单，因为 document.cookie 对象的值就是 Cookie 的属性值。因此你可以利用这个字符串在任何时候对 Cookie 进行访问。

`document.cookie` 字符串会保存一系列用分号分开的 `name = value` 键值对，这里的 `name` 就是一个 Cookie 名称，`value` 是它的值。

JavaScript 内置函数

数值方法

数值对象仅包含了几个任何对象均定义的默认方法

方法	描述
constructor()	返回创建该对象实例的函数。默认是数值对象。
toExponential()	强制将数值以指数形式显示。
toFixed()	可把 Number 四舍五入为指定小数位数的数字。
toLocaleString()	以字符串的形式返回当前对象的值。该字符串适用于宿主环境的当前区域设置。
toPrecision()	定义显示一个数多少位数（包括位小数的左和右）
toString()	返回该数值的字符串格式
valueOf()	返回数值

布尔方法

如下为相关方法及描述列表：

方法	描述
toSource()	返回一个包含布尔对象的源字符串;可以使用这个字符串创建一个等价的对象。
toString()	按照布尔结果返回“true”或“fales”。
valueOf()	返回布尔对象的原始值。

字符串方法

如下为相关方法及描述列表：

方法	描述
charAt()	返回指定位置的字符。
charCodeAt()	返回指定位置字符的数值。
concat()	返回布尔对象的原始值。
indexOf()	返回匹配子字符串第一次出现的位置，如果不存在就返回-1。
lastIndexOf()	返回匹配子字符串最后一次出现的位置，如果不存在就返回-1。
localeCompare()	比较两个字符串, 并返回以数字形式表示的比较结果。
length()	返回字符串的长度。

match()	用于匹配正则表达式。
replace()	通过与正则表达式找到子串位置，并替换为新指定的字符串。
search()	执行与一个正则表达式进行的搜索。
slice()	提取并返回一个子串。
split()	将字符串分割成多个子串，并存储进字符串数组。
substr()	返回字符串中指定位置，指定长度的子串。
toLocaleLowerCase()	大写字符转为小写，同时尊重当前语言环境。
toLocaleUpperCase()	小写字符转为大写，同时尊重当前语言环境。
toLowerCase()	大写字符转为小写。
toString()	返回表示该对象的一个字符串。
toUpperCase()	小写字符转为大写。
valueOf()	返回指定对象的原始数值。

HTML字符串格式化工具

方法	描述
anchor()	创建一个HTML锚作为一个超文本的目标。
big()	创建一个以“大”字体表示的字符串，好比置于< big >标签中一样。
blink()	创建一个闪烁的字符串，好比置于< blink >标签中一样。
bold()	创建一个粗体显示的字符串，好比置于< b >标签中一样。
fixed()	创建一个打字机字体显示的字符串，好比置于< tt >标签中一样。
fontcolor()	创建一个特定字体颜色显示的字符串，好比置于< font color="color" >标签中一样。
fontsize()	创建一个特定字体大小显示的字符串，好比置于< font size="size" >标签中一样。
italics()	创建一个斜体显示的字符串，好比置于< i >标签中一样。
link()	创建HTML超级链接。
small()	创建一个小小字体显示的字符串，好比置于< small >标签中一样。
strike()	创建一个加了删除线显示的字符串，好比置于< strike >标签中一样。
sub()	以下标的方式显示，好比置于< sub >标签中一样。
sup()	以上标的方式显示，好比置于< sup >标签中一样。

数组方法

如下为相关方法及描述的列表：

方法	描述
concat()	返回两个数据经过联接后的数组。
every()	如何数组内的元素均满足某测试函数，那么就返回true。

filter()	原来的数组中能过通过过滤器的元素组成一个新的数组返回。
forEach()	调用一个函数来处理数组中的每个元素。
indexOf()	返回与指定元素相匹配的第一个位置，如果不存在就返回-1
join()	连接数组中所有的元素，返回一个字符串
lastIndexOf()	返回与指定元素相匹配的最后一个位置，如果不存在就返回-1。
map()	调用一个函数处理数组中的每一个元素，将生成的结果组成一个新的数组，并返回
pop()	返回数组中的最后一个元素，并删除。
push()	在数组的最后增加一个元素，并返回新数组的长度
reduce()	对数组中的所有元素（从左到右）调用指定的回调函数。 该回调函数的返回值为累积结果，并且此返回值在下一次调用该回调函数时作为参数提供。
reduceRight()	对数组中的所有元素（从右到左）调用指定的回调函数。 该回调函数的返回值为累积结果，并且此返回值在下一次调用该回调函数时作为参数提供。
reverse()	反转数组元素的顺序——第一个成为最后一个, 最后成为第一。
shift()	删除数组的第一个元素并返回。
slice()	提取一段数组并返回一个新的数组
some()	, 如果存在一个元素满足所提供的测试函数，就返回true。
toString()	代表一个对象的源代码。
sort()	对数组中的元素排序。
splice()	增删数组中的元素。
toString()	返回一个表示数组及其元素的字符串。
unshift()	在数组的首部添加新的元素，并且返回新数组的长度

时期方法

如下为相关方法及描述列表：

方法	描述
Date()	返回今天的日期及时间。
getDate()	按照本地模式返回指定日期是哪日。
getDay()	按照本地模式返回指定日期是周几。
getFullYear()	按照本地模式返回指定日期是哪一年。
getMilliseconds()	按照本地模式返回指定日期是几毫秒。
getMinutes()	按照本地模式返回指定日期是几分。

<code>getMonth()</code>	按照本地模式返回指定日期的月份。
<code>getSeconds()</code>	按照本地模式返回指定日期是几秒。
<code>getTime()</code>	按照本地模式当前的格林威治时间。
<code>getTimezoneOffset()</code>	以分钟为单位返回时间偏差。
<code>getUTCDate()</code>	按照世界统一时间返回指定日期是几号。
<code>getUTCDay()</code>	按照世界统一时间返回指定日期是周几。
<code>getUTCFullYear()</code>	按照世界统一时间返回指定日的年份。
<code>getUTCHours()</code>	按照世界统一时间返回指定日期是几时。
<code>getUTCMilliseconds()</code>	按照世界统一时间返回指定日期的毫秒数。
<code>getUTCMinutes()</code>	按照世界统一时间返回指定日期的分钟数。
<code>getUTCMonth()</code>	按照世界统一时间返回指定日期的月份。
<code>getUTCSeconds()</code>	按照世界统一时间返回指定日期的秒数。
<code>setDate()</code>	按照本地模式设置日期。
<code>setFullYear()</code>	按照本地模式设置年份。
<code>setHours()</code>	按照本地模式设置小时。
<code>setMilliseconds()</code>	按照本地模式设置毫秒数。
<code>setMinutes()</code>	按照本地模式设置分钟数。
<code>setMonth()</code>	按照本地模式设置月份。
<code>setSeconds()</code>	按照本地模式设置秒数。
<code>setTime()</code>	按照格林威治格式设置毫秒数。
<code>setUTCDate()</code>	按照世界统一时间设置日期。
<code>setUTCFullYear()</code>	按照世界统一时间设置年份。
<code>setUTCHours()</code>	按照世界统一时间设置小时数。
<code>setUTCMilliseconds()</code>	按照世界统一时间设置毫秒数。
<code>setUTCMinutes()</code>	按照世界统一时间设置分钟数。
<code>setUTCMonth()</code>	按照世界统一时间设置月份。
<code>setUTCSeconds()</code>	按照世界统一时间设置秒数。
<code>toString()</code>	返回日期的字符串。
<code>toLocaleDateString()</code>	按照本地模式，返回日期的字符串。
<code>toLocaleFormat()</code>	使用格式字符串，将日期转换为一个字符串。
<code>toLocaleString()</code>	使用当前语言环境的约定将日期转换为一个字符串。
<code>toLocaleTimeString()</code>	返回日期的“时间”部分作为一个字符串, 使用当前语言环境的约定。
<code>toSource()</code>	返回一个字符串代表一个等价的日期对象的来源, 您可以使用这个值来创建一个新的对象。
<code>toString()</code>	返回一个字符串代表指定的日期对象。

toString()	返回日期的“时间”部分以字符串形式。
toUTCString()	使用通用时间约定，将日期转换为一个字符串。
valueOf()	返回日期对象的原始值。

日期静态方法

如下为相关方法及描述列表：

方法	描述
Date.parse()	解析并返回日期和时间的字符串表示的内部毫秒表示日期。
Date.UTC()	返回指定的毫秒表示UTC日期和时间。

数学方法

如下为相关方法及描述列表：

方法	描述
abs()	返回数值的绝对值。
acos()	返回一个数值的arccos值。
asin()	返回一个数值的arcsin值。
atan()	返回一个数值的arctan值。
ceil()	返回大于或等于整数最小的一个数字。
cos()	返回一个数值的cos值。
exp()	返回指数。
floor()	返回小于等于一个数的最大数。
log()	返回一个数值以e为底的对数。
max()	返回最大值。
min()	返回最小值。
pow()	返回以e为底的幂。
random()	返回0和1之间的一个伪随机数。
round()	返回四舍五入后的值。
sin()	返回sin值。
sqrt()	返回一个整数的平方根。
tan()	返回一个数值的tan值。
toSource()	返回字符串“Manth”。

正则表达式方法

如下为相关方法及描述列表：

方法	描述
<code>exec()</code>	执行一个字符串的搜索匹配。
<code>test()</code>	测试匹配的字符串参数。
<code>toSource()</code>	返回一个对象文字代表指定的对象；您可以使用这个值来创建一个新的对象。
<code>toString()</code>	返回一个字符串代表指定的对象。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/javascript/>