前言

JavaScript 参考教程

本教程为未接触过 JavaScript 的读者提供了比较完善的初级知识,但只限于初级知识:所有与动态网页密切相关的 JavaScript 在本教程中都未提及,包括动态定位、动画、让文档接收更多事件(document.captureEvent())等;所有在 IE 及 Netscape 中有不同的 JavaScript 都尽少提及。

本教程在结构上设计的比较像一个参考(reference),有参考的规划性,但又有教程的性质,所以我把它叫做“参考教程”。

本教程不面向连 HTML 都没有学好的读者。本教程中会多次提到有关 HTML 的内容,并不会对这些 HTML 的知识做讲解。

未接触过 JavaScript 的读者看完此教程应该对 JavaScript 有比较深的掌握。

接触过 JavaScript 的读者可以当此教程为参考来用。

本教程对 JavaScript 的基础有比较详细的讲述,对如对象、事件之类比较深入的知识则讲述得比较浅易,型如参考。如果读者有一定的理解能力和消化能力,相信在掌握了基础部分后,在学习进阶的章节时不会遇到什么困难。本教程所用的参考式结构,使教程看起来很统一——在讲述语句的时候,对语句以外的任何知识只字不提;在讲述对象的时候,又对对象以外的任何知识只字不提。如果你想快速学会 JavaScript,你可能要掌握“跳章学习”的方法。

左边可以看到本教程整个结构树。可以看到共分 5 章,前两章是基础知识,第三第四章是进阶知识——对象,第五章是针对两个对象——框架和 Cookie 进行详细的讲解,作为第三第四章中内容的补充。

理解能力不好或者依赖性强的读者可能学到 JavaScript 的核心——对象化编程时会觉得力不从心，因为它们不习惯这种参考式的文章；急于求成的读者可能对本教程非常统一的参考式结构觉得厌恶，因为他们必须学到第三第四章才可以做一个小小的 JavaScript。

本教程缺乏例子。读者要有很强的“悟性”和很愿意自己去实践，才可以很好的掌握 JavaScript。

本教程的作者 Micro.只此一人，伟大吗？除了我，还有两本 JavaScript 的书，一本是 JS 1.1 教程，但是本教程编了不久书被别人借去了；另一本是很浅的参考，如果照搬，大家都不会明白说了什么。还有两个浏览器，用来实践；还有英文版的 JavaScript 参考、教程，来自 MSDN 和 Netscape 网。还有一个脑，两只手，一台电脑……

JavaScript 有什么特点

JavaScript 使网页增加互动性。JavaScript 使有规律地重复的 HTML 文段简化，减少下载时间。JavaScript 能及时响应用户的操作，对提交表单做即时的检查，无需浪费时间交由 CGI 验证。JavaScript 的特点是无穷无尽的，只要你有创意。

Java 与 JavaScript 有什么不同

很多人看到 Java 和 JavaScript 都有“Java”四个字，就以为它们是同一样东西，连我自己当初也是这样。其实它们是完完全全不同的两种东西。Java，全称应该是 Java Applet，是嵌在网页中，而又有自己独立的运行窗口的小程序。Java Applet 是预先编译好的，一个 Applet 文件（.class）用 Notepad 打开阅读，根本不能理解。Java Applet 的功能很强大，可以访问 http、ftp 等协

议，甚至可以在电脑上种病毒（已有先例了）。相比之下，JavaScript 的能力就比较小了。JavaScript 是一种“脚本”（“Script”），它直接把代码写到 HTML 文档中，浏览器读取它们的时候才进行编译、执行，所以能查看 HTML 源文件就能查看 JavaScript 源代码。JavaScript 没有独立的运行窗口，浏览器当前窗口就是它的运行窗口。它们的相同点，我想只有同是以 Java 作编程语言一点了。

开发 JavaScript 该用什么软件

一个 JavaScript 程序其实是一个文档，一个文本文件。它是嵌入到 HTML 文档中的。所以，任何可以编写 HTML 文档的软件都可以用来开发 JavaScript。在此我推荐大家用 FrontPage 2000 附带的 Microsoft 脚本编辑器（在 FrontPage 菜单 | 工具 | 宏 | Microsoft 脚本编辑器）。它是个像 Visual Basic / C++ 一样的程序开发器，能对正在输入的语句作出简要提示。配合 FrontPage 2000，使工作量大大减少。

JavaScript 语言的基础

在什么地方插入 JavaScript

JavaScript 可以出现在 HTML 的任意地方。使用标记 <script>…</script>，你可以在 HTML 文档的任意地方插入 JavaScript，甚至在<HTML>之前插入也不成问题。不过如果要在声明框架的网页（框架网页）中插入，就一定要在<frameset>之前插入，否则不会运行。

基本格式

<script>
<!--
...
(JavaScript 代码)
...
//-->
</script>

第二行和第四行的作用，是让不懂<script>标记的浏览器忽略 JavaScript 代码。一般可以省略，因为现在想找不懂 Script 的浏览器，恐怕就连博物馆里也没有了。第四行前边的双反斜杠“//”是 JavaScript 里的注释标号，以后将学到。

另外一种插入 JavaScript 的方法，是把 JavaScript 代码写到另一个文件当中（此文件通常应该用“.js”作扩展名），然后用格式为“<script src="javascript.js"></script>”的标记把它嵌入到文档中。注意，一定要用“</script>”标记。

参考 <script>标记还有一个属性：language（缩写 lang），说明脚本使用的语言。对于 JavaScript，请用“language="JavaScript""”。

参考 相对于<script>标记，还有一个<server>标记。<server>标记所包含的，是服务器端（Server Side）的脚本。本教程只讨论客户器端（Client Side）的 JavaScript，也就是用<script>标记包含的脚本。

如果想在浏览器的“地址”栏中执行 JavaScript 语句，用这样的格式：

javascript:<JavaScript 语句>

这样的格式也可以用在连接中：

<a href="javascript:<JavaScript 语句>">...

JavaScript 基本语法

每一句 JavaScript 都有类似于以下的格式：

<语句>;

其中分号“;”是 JavaScript 语言作为一个语句结束的标识符。虽然现在很多浏览器都允许用回车充当结束符号，培养用分号作结束的习惯仍然是很好的。

语句块 语句块是用大括号“{ }”括起来的一个或 n 个语句。在大括号里边是几个语句，但是在大括号外边，语句块是被当作一个语句的。语句块是可以嵌套的，也就是说，一个语句块里边可以再包含一个或多个语句块。

JavaScript 中的变量

什么是变量 从字面上看，变量是可变的量；从编程角度讲，变量是用于存储某种/某些数值的存储器。所储存的值，可以是数字、字符或其他的一些东西。

变量的命名 变量的命名有以下要求：

只包含字母、数字和/或下划线；

要以字母开头；

不能太长（其实有什么人喜欢使用又长又臭的名字呢？）；

不能与 JavaScript 保留字（Key Words，Reserved Words，数量繁多，不能一一列出；凡是可以用来做 JavaScript 命令的字都是保留字）重复。

而且，变量是区分大小写的，例如，variable 和 Variable 是两个不同的变量。不仅如此，大部分命令和“对象”（请参阅“对象化编程”章）都是区分大小写的。

提示 给变量命名，最好避免用单个字母“a”“b”“c”等，而改用能清楚表达该变量在程序中的作用的词语。这样，不仅别人能更容易的了解你的程序，而且你在以后要修改程序的时候，也很快会记得该变量的作用。变量名一般用小写，如果是由多个单词组成的，那么第一个单词用小写，其他单词的第一个字母用大写。例如：myVariable 和 myAnotherVariable。这样做仅仅是为了美观和易读，因为 JavaScript 一些命令（以后将用更具体的方法阐述“命令”一词）都是用这种方法命名的：indexOf；charAt 等等。

变量需要声明 没有声明的变量不能使用，否则会出错：“未定义”。声明变量可以用：

var 〈变量〉 [= 〈值〉];

var 我们接触的第一个关键字（即保留字）。这个关键字用作声明变量。最简单的声明方法就是“var 〈变量〉;”，这将为〈变量〉准备内存，给它赋初始值“null”。如果加上“= 〈值〉”，则给〈变量〉赋予自定的初始值〈值〉。

数据类型 变量可以用的数据类型有：

整型 只能储存整数。可以是正整数、0、负整数，可以是十进制、八进制、十六进制。八进制数的表示方法是在数字前加“0”，如“0123”表示八进制数“123”。十六进制则是加“0x”：“0xEF”表示十六进制数“EF”。

浮点型 即“实型”，能储存小数。有资料显示，某些平台对浮点型变量的支持不稳定。没有需要就不要用浮点型。

字符串型 是用引号“″″”、“’’”包起来的零个至多个字符。用单引号还是

双引号由你决定。跟语文一样，用哪个引号开始就用哪个结束，而且单双引号可嵌套使用：'这里是"JavaScript 教程"。' 不过跟语文不同的是，JavaScript 中引号的嵌套只能有一层。如果想再多嵌一些，你需要转义字符：

转义字符 由于一些字符在屏幕上不能显示，或者 JavaScript 语法上已经有了特殊用途，在要用这些字符时，就要使用“转义字符”。转义字符用斜杠“\”开头：\' 单引号、\" 双引号、\n 换行符、\r 回车（以上只列出常用的转义字符）。于是，使用转义字符，就可以做到引号多重嵌套：'Micro 说："这里是\"JavaScript 教程\"。" '

布尔型 常用于判断，只有两个值可选：true（表“真”）和 false（表“假”）。true 和 false 是 JavaScript 的保留字。它们属于“常数”。

对象 关于对象，在“对象化编程”一章将详细讲到。

由于 JavaScript 对数据类型的要求不严格，一般来说，声明变量的时候不需要声明类型。而且就算声明了类型，在过程中还可以给变量赋予其他类型的值。声明类型可以用赋予初始值的方法做到：

```
var aString = '';
```

这将把 aString 定义为具有空值的字符串型变量。

```
var anInteger = 0;
```

这将把 anInteger 定义为值为 0 的整型。

变量的赋值 一个变量声明后，可以在任何时候对其赋值。赋值的语法是：

〈变量〉 = 〈表达式〉;

其中“=”叫“赋值符”，它的作用是把右边的值赋给左边的变量。下一节将讨论到表达式。

JavaScript 常数 有下列几个：

null 一个特殊的空值。当变量未定义，或者定义之后没有对其进行任何赋值操作，它的值就是“null”。企图返回一个不存在的对象时也会出现 null 值。

NaN “Not a Number”。出现这个数值比较少见，以至于我们可以不理它。当运算无法返回正确的数值时，就会返回“NaN”值。NaN 值非常特殊，因为它“不是数字”，所以任何数跟它都不相等，甚至 NaN 本身也不等于 NaN 。

true 布尔值“真”。用通俗的说法，“对”。

false 布尔值“假”。用通俗的说法，“错”。

在 Math 对象中还有一系列数学常数。这将在讨论“对象化编程”时谈到。

表达式与运算符

表达式 与数学中的定义相似，表达式是指具有一定的值的、用运算符把常数和变量连接起来的代数式。一个表达式可以只包含一个常数或一个变量。运算符可以是四则运算符、关系运算符、位运算符、逻辑运算符、复合运算符。下表将这些运算符从高优先级到低优先级排列：

括号	(x) [x]	中括号只用于指明**数组**的**下标**
求反、自加、	-x	返回 x 的相反数
自减	!x	返回与 x (布尔值)相反的布尔值
	x++	x 值加 1，但仍返回原来的 x 值
	x--	x 值减 1，但仍返回原来的 x 值

	++x	x 值加 1，返回后来的 x 值
	--x	x 值减 1，返回后来的 x 值
	x*y	返回 x 乘以 y 的值
乘、除	x/y	返回 x 除以 y 的值
	x%y	返回 x 与 y 的模（x 除以 y 的余数）
	x+y	返回 x 加 y 的值
加、减	x-y	返回 x 减 y 的值
关系运算	x<y x<=y x>=y x>y	当符合条件时返回 true 值，否则返回 false 值
	x==y	当 x 等于 y 时返回 true 值，否则返回 false 值
等于、不等于	x!=y	当 x 不等于 y 时返回 true 值，否则返回 false 值
位与	x&y	当两个数位同时为 1 时，返回的数据的当前数位为 1，其他情况都为 0
位异或	x^y	两个数位中有且只有一个为 0 时，返回 0，否则返回 1
位或	x|y	两个数位中只要有一个为 1，则返回 1；当两个数位都为零时才返回零

位运算符通常会被当作逻辑运算符来使用。它的实际运算情况是：把两个操作数（即 x 和 y）化成二进制数，对每个数位执行以上所列工作，然后返回得到的新二进制数。由于“真”值在电脑内部（通常）是全部数位都是 1 的二进制数，而“假”值则是全部是 0 的二进制数，所以位运算符也可以充当逻辑运算符。

逻辑与	x&&y	当 x 和 y 同时为 true 时返回 true，否则返回 false
逻辑或	x||y	当 x 和 y 任意一个为 true 时返回 true，当两者同时为 false 时返回 false

逻辑与/或有时候被称为“快速与/或”。这是因为当第一操作数（x）已经可以决定结果，它们将不去理会 y 的值。例如，false && y，因为 x == false，不管 y 的值是什么，结果始终是 false，于是本表达式立即返回 false，而不论 y 是多少，甚至 y 可以导致出错，程序也可以照样运行下去。

条件	c?x:y	当条件 c 为 true 时返回 x 的值（执行 x 语句），否则返回 y 的值（执行 y 语句）
赋值、复合运算	x=y	把 y 的值赋给 x，返回所赋的值
	x+=y x-=y x*=y x/=y x%=y	x 与 y 相加/减/乘/除/求余，所得结果赋给 x，并返回 x 赋值后

注意 所有与四则运算有关的运算符都不能作用在字符串型变量上。字符串可以使用 +、+= 作为连接两个字符串之用。

提示 请密切注意运算的优先级。编程时如果不记得运算符的优先级，可以使用括号( )。例如：(a == 0)||(b == 0)。

一些用来赋值的表达式，由于有返回的值，可以加以利用。例如，用以下语句：a = b = c = 10，可以一次对三个变量赋值。

语句

下面将开始讨论 JavaScript 基本编程命令，或者叫“语句”。

注释

像其他所有语言一样，JavaScript 的注释在运行时也是被忽略的。注释只给程序员提供消息。

JavaScript 注释有两种：单行注释和多行注释。单行注释用双反斜杠“//”表示。当一行代码有“//”，那么，“//”后面的部分将被忽略。而多行注释是用“/*”和“*/”括起来的一行到多行文字。程序执行到“/*”处，将忽略以后的所有文字，直到出现“*/”为止。

提示 如果你的程序需要草稿，或者需要让别人阅读，注释能帮上大忙。养成写注释的习惯，能节省你和其他程序员的宝贵时间，使他们不用花费多余的时间琢磨你的程序。在程序调试的时候，有时需要把一段代码换成另一段，或者暂时不要一段代码。这时最忌用 Delete 键，如果想要回那段代码怎么办？最好还是用注释，把暂时不要的代码“隐”去，到确定方法以后再删除也不迟。

if 语句

if（〈条件〉）〈语句 1〉[else 〈语句 2〉];

本语句有点象条件表达式“?:”：当〈条件〉为真时执行〈语句 1〉，否则，如果 else 部分存在的话，就执行〈语句 2〉。与“?:”不同的是，if 只是一条语句，

不会返回数值。<条件>是布尔值，必须用小括号括起来；<语句 1>和<语句 2>都只能是一个语句，欲使用多条语句，请用语句块。

注意 请看下例：

```
if (a == 1)
  if (b == 0) alert(a+b);
else
  alert(a-b);
```

本代码企图用缩进的方法说明 else 是对应 if (a == 1) 的，但是实际上，由于 else 与 if (b == 0) 最相近，本代码不能按作者的想法运行。正确的代码是

```
if (a == 1) {
  if (b == 0) alert(a+b);
} else {
  alert(a-b);
}
```

提示 一行代码太长，或者涉及到比较复杂的嵌套，可以考虑用多行文本，如上例，if (a == 1) 后面没有立即写上语句，而是换一行再继续写。浏览器不会混淆的，当它们读完一行，发现是一句未完成语句，它们会继续往下读。使用缩进也是很好的习惯，当一些语句与上面的一两句语句有从属关系时，使用缩进能使程序更易读，方便程序员进行编写或修改工作。

循环体

for (<变量>=<初始值>; <循环条件>; <变量累加方法>) <语句>;

本语句的作用是重复执行<语句>，直到<循环条件>为 false 为止。它是这样运作的：首先给<变量>赋<初始值>，然后*判断<循环条件>（应该是一个关于<变量>的条件表达式）是否成立，如果成立就执行<语句>，然后按<变量累加方法>对<变量>作累加，回到“*”处重复，如果不成立就退出循环。这叫做“for 循环”。下面看看例子。

```
for (i = 1; i < 10; i++) document.write(i);
```

本语句先给 i 赋初始值 1，然后执行 document.write(i)语句（作用是在文档中写 i 的值，请参越“对象化编程”一章）；重复时 i++，也就是把 i 加 1；循环直到 i<10 不满足，也就是 i>=10 时结束。结果是在文档中输出了“123456789”。

和 if 语句一样，<语句>只能是一行语句，如果想用多条语句，你需要用语句块。

与其他语言不同，JavaScript 的 for 循环没有规定循环变量每次循环一定要加一或减一，<变量累加方法>可以是任意的赋值表达式，如 i+=3、i*=2、i-=j 等都成立。

提示 适当的使用 for 循环，能使 HTML 文档中大量的有规律重复的部分简化，也就是用 for 循环重复写一些 HTML 代码，达到提高网页下载速度的目的。不过请在 Netscape 中重复进行严格测试，保证通过了才好把网页传上去。作者曾试过多次因为用 for 循环向文档重复写 HTML 代码而导致 Netscape“猝死”。IE 中绝对没有这种事情发生，如果你的网也是只给 IE 看的，用多多的 for 也没问题。

除了 for 循环，JavaScript 还提供 while 循环。

while (<循环条件>) <语句>;

比 for 循环简单，while 循环的作用是当满足<循环条件>时执行<语句>。while 循环的累加性质没有 for 循环强。<语句>也只能是一条语句，但是一般情况下都使用语句块，因为除了要重复执行某些语句之外，还需要一些能变动<循环条件>所涉及的变量的值的语句，否则一但踏入此循环，就会因为条件总是满足而一直困在循环里面，不能出来。这种情况，我们习惯称之为“死循环”。死循环会弄停当时正在运行的代码、正在下载的文档，和占用很大的内存，很可能造成死机，应该尽最大的努力避免。

break 和 continue

有时候在循环体内，需要立即跳出循环或跳过循环体内其余代码而进行下一次循环。break 和 continue 帮了我们大忙。

break;

本语句放在循环体内，作用是立即跳出循环。

continue;

本语句放在循环体内，作用是中止本次循环，并执行下一次循环。如果循环的条件已经不符合，就跳出循环。

例

```
for (i = 1; i < 10; i++) {
  if (i == 3 || i == 5 || i == 8) continue;
  document.write(i);
}
```

输出：124679。

switch 语句

如果要把某些数据分类，例如，要把学生的成绩按优、良、中、差分类，我们可能会用 if 语句：

```
if (score >= 0 && score < 60) {
  result = 'fail';
} else if (score < 80) {
  result = 'pass';
} else if (score < 90) {
  result = 'good';
} else if (score <= 100) {
  result = 'excellent';
} else {
  result = 'error';
}
```

看起来没有问题，但使用太多的 if 语句的话，程序看起来有点乱。switch 语句就是解决这种问题的最好方法。

```
switch (e) {
  case r1: （注意：冒号）
```

```
    ...
    [break;]
  case r2:
    ...
    [break;]
  ...
  [default:
    ...]
}
```

这一大段的作用是:计算 e 的值(e 为表达式),然后跟下边“case”后的 r1、r2……比较,当找到一个相等于 e 的值时,就执行该“case”后的语句,直到遇到 break 语句或 switch 段落结束(“}”)。如果没有一个值与 e 匹配,那么就执行“default:”后边的语句,如果没有 default 块,switch 语句结束。

上边的 if 段用 switch 改写就是:

```
switch (parseInt(score / 10)) {
  case 0:
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
    result = 'fail';
    break;
  case 6:
  case 7:
    result = 'pass';
    break;
```

```
  case 8:
    result = 'good';
    break;
  case 9:
    result = 'excellent';
    break;
  default:
    if (score == 100)
      result = 'excellent';
    else
      result = 'error';
}
```

其中 parseInt()方法是以后会介绍的，作用是取整。最后 default 段用的 if 语句，是为了不把 100 分当错误论（parseInt(100 / 10) == 10）。

对象的基本知识

04-13

JavaScript 是使用“对象化编程”的，或者叫“面向对象编程”的。所谓“对象化编程”，意思是把 JavaScript 能涉及的范围划分成大大小小的对象，对象下面还继续划分对象直至非常详细为止，所有的编程都以对象为出发点，基于对象。小到一个变量，大到网页文档、窗口甚至屏幕，都是对象。这一章将“面向对象”讲述 JavaScript 的运行情况。

对象的基本知识

对象是可以从 JavaScript“势力范围”中划分出来的一小块，可以是一段文字、

一幅图片、一个表单（Form）等等。每个对象有它自己的**属性**、**方法**和**事件**。对象的**属性**是反映该对象某些特定的性质的，例如：字符串的长度、图像的长宽、文字框（Textbox）里的文字等等；对象的**方法**能对该对象做一些事情，例如，表单的“提交”(Submit)，窗口的“滚动”(Scrolling)等等；而对象的**事件**就能响应发生在对象上的事情，例如提交表单产生表单的“提交事件”，点击连接产生的“点击事件”。不是所有的对象都有以上三个性质，有些没有事件，有些只有属性。引用对象的任一“性质”用“<对象名>.<性质名>”这种方法。

基本对象

现在我们要复习以上学过的内容了——把一些数据类型用对象的角度重新学习一下。

Number “数字”对象。这个对象用得很少，作者就一次也没有见过。不过属于“Number”的对象，也就是“变量”就多了。

属性

MAX_VALUE 用法：Number.MAX_VALUE；返回“最大值”。

MIN_VALUE 用法：Number.MIN_VALUE；返回“最小值”。

NaN 用法：Number.NaN 或 NaN；返回“NaN”。“NaN”（不是数值）在很早就介绍过了。

NEGATIVE_INFINITY 用法:Number.NEGATIVE_INFINITY;返回:负无穷大,比“最小值”还小的值。

POSITIVE_INFINITY 用法:Number.POSITIVE_INFINITY;返回:正无穷大,比“最大值”还大的值。

方法

toString() 用法：<数值变量>.toString()；返回：字符串形式的数值。如：若 a == 123；则 a.toString() == '123'。

String 字符串对象。声明一个字符串对象最简单、快捷、有效、常用的方法就是直接赋值。

属性

length 用法：<字符串对象>.length；返回该字符串的长度。

方法

charAt() 用法：<字符串对象>.charAt(<位置>)；返回该字符串位于第<位置>位的单个字符。注意 :字符串中的一个字符是第 0 位的 ,第二个才是第 1 位的，最后一个字符是第 length - 1 位的。

charCodeAt() 用法：<字符串对象>.charCodeAt(<位置>)；返回该字符串位于第<位置>位的单个字符的 ASCII 码。

fromCharCode() 用法：String.fromCharCode(a, b, c...)；返回一个字符串，该字符串每个字符的 ASCII 码由 a, b, c... 等来确定。

indexOf() 用法：<字符串对象>.indexOf(<另一个字符串对象>[, <起始位置>])；该方法从<字符串对象>中查找<另一个字符串对象>（如果给出<起始位置>就忽略之前的位置），如果找到了，就返回它的位置，没有找到就返回“-1”。所有的“位置”都是从零开始的。

lastIndexOf() 用法：<字符串对象>.lastIndexOf(<另一个字符串对象>[, <起始位置>])；跟 indexOf() 相似，不过是从后边开始找。

split() 用法：<字符串对象>.split(<分隔符字符>)；返回一个数组，该数组是

从<字符串对象>中分离开来的，<分隔符字符>决定了分离的地方，它本身不会包含在所返回的数组中。例如：'1&2&345&678'.split('&')返回数组:1,2,345,678。关于数组，我们等一下就讨论。

substring() 用法：<字符串对象>.substring(<始>[,<终>])；返回原字符串的子字符串，该字符串是原字符串从<始>位置到<终>位置的前一位置的一段。<终> - <始> = 返回字符串的长度（length）。如果没有指定<终>或指定得超过字符串长度，则子字符串从<始>位置一直取到原字符串尾。如果所指定的位置不能返回字符串，则返回空字符串。

substr() 用法：<字符串对象>.substr(<始>[,<长>])；返回原字符串的子字符串，该字符串是原字符串从<始>位置开始，长度为<长>的一段。如果没有指定<长>或指定得超过字符串长度，则子字符串从<始>位置一直取到原字符串尾。如果所指定的位置不能返回字符串，则返回空字符串。

toLowerCase() 用法：<字符串对象>.toLowerCase()；返回把原字符串所有大写字母都变成小写的字符串。

toUpperCase() 用法：<字符串对象>.toUpperCase()；返回把原字符串所有小写字母都变成大写的字符串。

Array 数组对象。数组对象是一个对象的集合，里边的对象可以是不同类型的。数组的每一个成员对象都有一个“下标”，用来表示它在数组中的位置（既然是“位置”，就也是从零开始的啦）。

数组的定义方法：

var <数组名> = new Array();

这样就定义了一个空数组。以后要添加数组元素，就用：

<数组名>[<下标>] = ...;

注意这里的方括号不是“可以省略”的意思,数组的下标表示方法就是用方括号括起来。

如果想在定义数组的时候直接初始化数据，请用：

var <数组名> = new Array(<元素 1>, <元素 2>, <元素 3>...);

例如，var myArray = new Array(1, 4.5, 'Hi'); 定义了一个数组 myArray，里边的元素是：myArray[0] == 1; myArray[1] == 4.5; myArray[2] == 'Hi'。

但是，如果元素列表中只有一个元素，而这个元素又是一个正整数的话，这将定义一个包含<正整数>个空元素的数组。

注意：JavaScript 只有一维数组！千万不要用“Array(3,4)”这种愚蠢的方法来定义 4 x 5 的二维数组，或者用“myArray[2,3]”这种方法来返回“二维数组”中的元素。任意“myArray[...,3]”这种形式的调用其实只返回了“myArray[3]”。要使用多维数组，请用这种虚拟法：

var myArray = new Array(new Array(), new Array(), new Array(), ...);

其实这是一个一维数组，里边的每一个元素又是一个数组。调用这个“二维数组”的元素时：myArray[2][3] = ...;

属性

length 用法：<数组对象>.length；返回：数组的长度，即数组里有多少个元素。它等于数组里最后一个元素的下标加一。所以，想添加一个元素，只需要：myArray[myArray.length] = ...。

方法

join() 用法：<数组对象>.join(<分隔符>)；返回一个字符串，该字符串把数组

中的各个元素串起来，用<分隔符>置于元素与元素之间。这个方法不影响数组原本的内容。

reverse() 用法：<数组对象>.reverse()；使数组中的元素顺序反过来。如果对数组[1, 2, 3]使用这个方法，它将使数组变成：[3, 2, 1]。

slice() 用法：<数组对象>.slice(<始>[,<终>])；返回一个数组，该数组是原数组的子集，始于<始>，终于<终>。如果不给出<终>，则子集一直取到原数组的结尾。

sort() 用法：<数组对象>.sort([<方法函数>])；使数组中的元素按照一定的顺序排列。如果不指定<方法函数>，则按字母顺序排列。在这种情况下，80 是比 9 排得前的。如果指定<方法函数>，则按<方法函数>所指定的排序方法排序。<方法函数>比较难讲述，这里只将一些有用的<方法函数>介绍给大家。

按升序排列数字：

```
function sortMethod(a, b) {
    return a - b;
}

myArray.sort(sortMethod);
```

按降序排列数字：把上面的“a - b”该成“b - a”。

有关函数，请看下面。

Math “数学”对象，提供对数据的数学计算。下面所提到的属性和方法，不再详细说明“用法”，大家在使用的时候记住用“Math.<名>”这种格式。

属性

E 返回常数 e (2.718281828...)。

LN2 返回 2 的自然对数 (ln 2)。

LN10 返回 10 的自然对数 (ln 10)。

LOG2E 返回以 2 为低的 e 的对数 (log2e)。

LOG10E 返回以 10 为低的 e 的对数 (log10e)。

PI 返回 π (3.1415926535...)。

SQRT1_2 返回 1/2 的平方根。

SQRT2 返回 2 的平方根。

方法

abs(x) 返回 x 的绝对值。

acos(x) 返回 x 的反余弦值(余弦值等于 x 的角度),用弧度表示。

asin(x) 返回 x 的反正弦值。

atan(x) 返回 x 的反正切值。

atan2(x, y) 返回复平面内点(x, y)对应的复数的幅角,用弧度表示,其值在 -π 到 π 之间。

ceil(x) 返回大于等于 x 的最小整数。

cos(x) 返回 x 的余弦。

exp(x) 返回 e 的 x 次幂 (ex)。

floor(x) 返回小于等于 x 的最大整数。

log(x) 返回 x 的自然对数 (ln x)。

max(a, b) 返回 a, b 中较大的数。

min(a, b) 返回 a, b 中较小的数。

pow(n, m) 返回 n 的 m 次幂 (nm)。

random() 返回大于 0 小于 1 的一个随机数。

round(x) 返回 x 四舍五入后的值。

sin(x) 返回 x 的正弦。

sqrt(x) 返回 x 的平方根。

tan(x) 返回 x 的正切。

Date 日期对象。这个对象可以储存任意一个日期，从 0001 年到 9999 年，并且可以精确到毫秒数(1/1000 秒)。在内部，日期对象是一个整数，它是从 1970 年 1 月 1 日零时正开始计算到日期对象所指的日期的毫秒数。如果所指日期比 1970 年早，则它是一个负数。所有日期时间，如果不指定时区，都采用“UTC”（世界时）时区，它与“GMT”（格林威治时间）在数值上是一样的。

定义一个日期对象：

```
var d = new Date;
```

这个方法使 d 成为日期对象，并且已有初始值：当前时间。如果要自定初始值，可以用：

```
var d = new Date(99, 10, 1);      //99 年 10 月 1 日
var d = new Date('Oct 1, 1999'); //99 年 10 月 1 日
```

等等方法。最好的方法就是用下面介绍的“方法”来严格的定义时间。

方法

以下有很多“g/set[UTC]XXX”这样的方法，它表示既有“getXXX”方法，又有“setXXX”方法。“get”是获得某个数值，而“set”是设定某个数值。如果带有“UTC”字母，则表示获得/设定的数值是基于 UTC 时间的，没有则表示基于本地时间或浏览期默认时间的。

如无说明，方法的使用格式为：“<对象>.<方法>”，下同。

g/set[UTC]FullYear() 返回/设置年份，用四位数表示。如果使用“x.set[UTC]FullYear(99)”，则年份被设定为 0099 年。

g/set[UTC]Year()返回/设置年份，用两位数表示。设定的时候浏览器自动加上“19”开头，故使用“x.set[UTC]Year(00)”把年份设定为 1900 年。

g/set[UTC]Month()返回/设置月份。

g/set[UTC]Date()返回/设置日期。

g/set[UTC]Day()返回/设置星期，0 表示星期天。

g/set[UTC]Hours()返回/设置小时数，24 小时制。

g/set[UTC]Minutes()返回/设置分钟数。

g/set[UTC]Seconds()返回/设置秒钟数。

g/set[UTC]Milliseconds()返回/设置毫秒数。

g/setTime() 返回/设置时间 ,该时间就是日期对象的内部处理方法 :从 1970 年 1 月 1 日零时正开始计算到日期对象所指的日期的毫秒数。如果要使某日期对象所指的时间推迟 1 小时，就用：“x.setTime(x.getTime() + 60 * 60 * 1000);”（一小时 60 分，一分 60 秒，一秒 1000 毫秒）。

getTimezoneOffset() 返回日期对象采用的时区与格林威治时间所差的分钟数。在格林威治东方的市区，该值为负，例如：中国时区(GMT+0800)返回“-480”。

toString() 返回一个字符串，描述日期对象所指的日期。这个字符串的格式类似于：“Fri Jul 21 15:43:46 UTC+0800 2000”。

toLocaleString() 返回一个字符串，描述日期对象所指的日期，用本地时间表示格式。如：“2000-07-21 15:43:46”。

toGMTString() 返回一个字符串，描述日期对象所指的日期，用 GMT 格式。

toUTCString() 返回一个字符串，描述日期对象所指的日期，用 UTC 格式。

parse() 用法：Date.parse(<日期对象>)；返回该日期对象的内部表达方式。

全局对象

全局对象从不现形，它可以说是虚拟出来的，目的在于把全局函数“对象化”。在 Microsoft JScript 语言参考中，它叫做“Global 对象”，但是引用它的方法和属性从来不用“Global.xxx”（况且这样做会出错），而直接用“xxx”。

属性

NaN 一早就说过了。

方法

eval() 把括号内的字符串当作标准语句或表达式来运行。

isFinite() 如果括号内的数字是“有限”的（介于 Number.MIN_VALUE 和 Number.MAX_VALUE 之间）就返回 true；否则返回 false。

isNaN() 如果括号内的值是“NaN”则返回 true 否则返回 false。

parseInt() 返回把括号内的内容转换成整数之后的值。如果括号内是字符串，则字符串开头的数字部分被转换成整数，如果以字母开头，则返回“NaN”。

parseFloat() 返回把括号内的字符串转换成浮点数之后的值，字符串开头的数字部分被转换成浮点数，如果以字母开头，则返回“NaN”。

toString() 用法：<对象>.toString()；把对象转换成字符串。如果在括号中指定一个数值，则转换过程中所有数值转换成特定进制。

escape() 返回括号中的字符串经过编码后的新字符串。该编码应用于 URL，也

就是把空格写成“%20”这种格式。“+”不被编码，如果要“+”也被编码，请用：escape('...', 1)。

unescape() 是 escape() 的反过程。解编括号中字符串成为一般字符串。

函数函数的定义

所谓“函数”，是有返回值的对象或对象的方法。

函数的种类

常见的函数有：构造函数，如 Array()，能构造一个数组；全局函数，即全局对象里的方法；自定义函数；等等。

自定义函数

定义函数用以下语句：

```
function 函数名([参数集]) {
    ...
    [return[ <值>];]
    ...
}
```

其中，用在 function 之后和函数结尾的大括号是不能省去的，就算整个函数只有一句。

函数名与变量名有一样的起名规定，也就是只包含字母数字下划线、字母排头、不能与保留字重复等。

参数集可有可无，但括号就一定要有。

参数 是函数外部向函数内部传递信息的桥梁，例如，想叫一个函数返回 3 的立方，你就要让函数知道“3”这个数值，这时候就要有一个变量来接收数值，这

种变量叫做参数。

参数集是一个或多个用逗号分隔开来的参数的集合，如：a, b, c。

函数的内部有一至多行语句，这些语句并不会立即执行，而只当有其它程序调用它时才执行。这些语句中可能包含“return”语句。在执行一个函数的时候，碰到 return 语句，函数立刻停止执行，并返回到调用它的程序中。如果“return”后带有<值>，则退出函数的同时返回该值。

在函数的内部，参数可以直接当作变量来使用，并可以用 var 语句来新建一些变量，但是这些变量都不能被函数外部的过程调用。要使函数内部的信息能被外部调用，要么使用“return”返回值，要么使用**全局变量**。

全局变量 在 Script 的“根部”（非函数内部）的“var”语句所定义的变量就是全局变量，它能在整个过程的任意地方被调用、更改。

例

```
function addAll(a, b, c) {
    return a + b + c;
}

var total = addAll(3, 4, 5);
```

这个例子建立了一个叫“addAll”的函数，它有 3 个参数：a, b, c，作用是返回三个数相加的结果。在函数外部，利用“var total = addAll(3, 4, 5);”接收函数的返回值。

更多的时候，函数是没有返回值的，这种函数在一些比较强调严格的语言中是叫做“过程”的，例如 Basic 类语言的“Sub”、Pascal 语言的“procedure”。

属性

arguments 一个数组，反映外部程序调用函数时指定的参数。用法：直接在函数内部调用“arguments”。

JavaScript 中的 Navigator 浏览器对象

navigator 浏览器对象，包含了正在使用的 Navigator 的版本信息。反映了当前使用的浏览器的资料。JavaScript 客户端运行时刻引擎自动创建 navigator 对象。

更详细的信息可以去查 msdn 或者 Navigator 2.0 以后的说明文档，这里我们之做个简单的说明

包括一下几大属性：

- appCodeName 返回浏览器的“码名”(?)，流行的 IE 和 NN 都返回 'Mozilla'。

  下面的例子显示了 appCodeName 属性的值：

  ```
  document.write("navigator.appCodeName 的值是" +
  navigator.appCodeName)
  ```

- appName 返回浏览器名。IE 返回 'Microsoft Internet Explorer',NN 返回 'Netscape'。

  下面的例子显示了 appName 属性的值：

```
document.write("navigator.appName 的值是 " + navigator.appName)
```

- appVersion 返回浏览器版本，包括了大版本号、小版本号、语言、操作平台等信息。
- language 语言
- mimeType 以数组表示所支持的 MIME 类型
- platform 返回浏览器的操作平台，对于 Windows 9x 上的浏览器，返回 'Win32'（大小写可能有差异）。
- userAgent 返回以上全部信息。例如，IE5.01 返回 'Mozilla/4.0 (compatible; MSIE 5.01; Windows 98)'。
- plugins 以数组表示已安装的外挂程序
- javaEnabled() 返回一个布尔值，代表当前浏览器允许不允许 Java。

检测浏览器的版本、所支持的 MIME 类型、已安装的外挂程序（plug-in）。该对象包含两个子对象：外挂对象、MIME 类型对象。

例如：

```
<Script>
with (document) {
     write ("你的浏览器信息：<OL>");
     write ("<LI>代码："+navigator.appCodeName);
     write ("<LI>名称："+navigator.appName);
     write ("<LI>版本："+navigator.appVersion);
     write ("<LI>语言："+navigator.language);
```

```
      write ("<LI>编译平台："+navigator.platform);
      write ("<LI>用户表头："+navigator.userAgent);
}
</Script>
```

例如：

```
<Script>
if (document.all) {
    document.write("你的浏览器是：MSIE");
}    else {
    document.write("你的浏览器是：Navigator");
}
</Script>
```

JavaScript 中的 Screen 屏幕对象

这是基本 JavaScript 的屏幕对象

- screen 屏幕对象 反映了当前用户的屏幕设置。
- width 返回屏幕的宽度（像素数）。
- height 返回屏幕的高度。
- availWidth 返回屏幕的可用宽度（除去了一些不自动隐藏的类似任务栏的东西所占用的宽度）。
- availHeight 返回屏幕的可用高度。

- colorDepth 返回当前颜色设置所用的位数 - 1：黑白；8：256 色；16：增强色；24/32：真彩色

下面是英文的 Navigator 浏览器的屏幕对象：

- availHeight:minus permanent or semipermanent user interface features displayed by the operating system:such as the Taskbar on Windows.
- availWidth:Specifies the width of the screen, in pixels, minus permanent or semipermanent user interface:features displayed by the operating system, such as the Taskbar on Windows.
- colorDepth:The bit depth of the color palette, if one is in use; otherwise, the value is derived from screen.pixelDepth.
- height:Display screen height.
- pixelDepth:Display screen color resolution (bits per pixel).
- width:Display screen width.

JavaScript 中的 Window 窗口对象

JavaScript 中的 Window 窗口对象

他是 JavaScript 中最大的对象，它描述的是一个浏览器窗口。一般要引用它的属性和方法时，不需要用“window.xxx”这种形式，而直接使用“xxx”。一个框架页面也是一个窗口。

Window 窗口对象有如下属性：

- name 窗口的名称，由打开它的连接(<a target="...">)或框架页(<frame name="...">)或某一个窗口调用的 open() 方法（见下）决定。一般我们不会用这个属性。
- status 指窗口下方的“状态栏”所显示的内容。通过对 status 赋值，可以改变状态栏的显示。
- opener 用法：window.opener；返回打开本窗口的窗口对象。注意：返回的是一个窗口对象。如果窗口不是由其他窗口打开的，在 Netscape 中这个属性返回 null；在 IE 中返回“未定义”（undefined）。undefined 在一定程度上等于 null。注意：undefined 不是 JavaScript 常数，如果你企图使用“undefined”，那就真的返回“未定义”了。
- self 指窗口本身，它返回的对象跟 window 对象是一模一样的。最常用的是“self.close()”，放在<a>标记中：“关闭窗口”。
- parent 返回窗口所属的框架页对象。
- top 返回占据整个浏览器窗口的最顶端的框架页对象。
- history 历史对象，见下。
- location 地址对象，见下。
- document 文档对象，见下。

Window 窗口对象有如下方法：

第一个方法是 open() 打开一个窗口。

用法：

open(<URL 字符串>, <窗口名称字符串>, <参数字符串>);

说明：

- <URL 字符串>：描述所打开的窗口打开哪一个网页。如果留空（''），则不打开任意网页。
- <窗口名称字符串>：描述被打开的窗口的名称（window.name），可以使用'_top'、'_blank'等内建名称。这里的名称跟“<a href="..." target="...">”里的“target”属性是一样的。
- <参数字符串>:描述被打开的窗口的样貌。如果只需要打开一个普通窗口，该字符串留空(''),如果要指定样貌,就在字符串里写上一到多个参数，参数之间用逗号隔开。

例：打开一个 400 x 100 的干净的窗口：

```
open('','_blank','width=400,height=100,menubar=no,toolbar=no,
location=no,directories=no,status=no, scrollbars=yes,resizable=yes')
```

open()的参数

- top=# 窗口顶部离开屏幕顶部的像素数
- left=# 窗口左端离开屏幕左端的像素数
- width=# 窗口的宽度
- height=# 窗口的高度
- menubar=... 窗口有没有菜单，取值 yes 或 no
- toolbar=... 窗口有没有工具条，取值 yes 或 no
- location=... 窗口有没有地址栏，取值 yes 或 no
- directories=... 窗口有没有连接区，取值 yes 或 no
- scrollbars=... 窗口有没有滚动条，取值 yes 或 no
- status=... 窗口有没有状态栏，取值 yes 或 no
- resizable=... 窗口给不给调整大小，取值 yes 或 no

注意：open() 方法有返回值，返回的就是它打开的窗口对象。比如

```
var newWindow = open('','_blank');
```

这样把一个新窗口赋值到“newWindow”变量中，以后通过“newWindow”变量就可以控制窗口了。

close() 关闭一个已打开的窗口。

用法：

```
window.close()
```

或

```
self.close()
```

主要作用是关闭本窗口；

<窗口对象>.close()：关闭指定的窗口。注意如果该窗口有状态栏，调用该方法后浏览器会警告：“网页正在试图关闭窗口，是否关闭？”然后等待用户选择是否；如果没有状态栏，调用该方法将直接关闭窗口。

另外 Window 窗口对象还有如下方法

- blur() 使焦点从窗口移走，窗口变为“非活动窗口”。
- focus() 是窗口获得焦点，变为“活动窗口”。不过在 Windows 98，该方法只能使窗口的标题栏和任务栏上的相应按钮闪烁，提示用户该窗口正在试图获得焦点。
- scrollTo() 用法：[<窗口对象>.]scrollTo(x, y)；使窗口滚动，使文档从左上角数起的(x, y)点滚动到窗口的左上角。
- scrollBy() 用法：[<窗口对象>.]scrollBy(deltaX, deltaY)；使窗口向右滚动 deltaX 像素，向下滚动 deltaY 像素。如果取负值，则向相反的方向滚动。
- resizeTo() 用法：[<窗口对象>.]resizeTo(width, height)；使窗口调整大小到宽 width 像素，高 height 像素。
- resizeBy() 用法：[<窗口对象>.]resizeBy(deltaWidth, deltaHeight)；使窗口调整大小，宽增大 deltaWidth 像素，高增大 deltaHeight 像素。如果取负值，则减少。

- alert() 用法：alert(<字符串>)；弹出一个只包含“确定”按钮的对话框，显示<字符串>的内容，整个文档的读取、Script 的运行都会暂停，直到用户按下“确定”。
- confirm() 用法 :confirm(<字符串>) ;弹出一个包含“确定”和“取消”按钮的对话框，显示<字符串>的内容，要求用户做出选择，整个文档的读取、Script 的运行都会暂停。如果用户按下“确定”，则返回 true 值，如果按下“取消”，则返回 false 值。
- prompt() 用法：prompt(<字符串>[, <初始值>])；弹出一个包含“确认”“取消”和一个文本框的对话框，显示<字符串>的内容，要求用户在文本框输入一些数据，整个文档的读取、Script 的运行都会暂停。如果用户按下“确认”，则返回文本框里已有的内容，如果用户按下“取消”，则返回 null 值。如果指定<初始值>，则文本框里会有默认值。

Window 窗口对象有如下事件：

window.onload;发生在文档全部下载完毕的时候。全部下载完毕意味着不但 HTML 文件，而且包含的图片，插件，控件，小程序等全部内容都下载完毕。本事件是 window 的事件，但是在 HTML 中指定事件处理程序的时候，我们是把它写在<body>标记中的。

window.onunload;发生在用户退出文档（或者关闭窗口，或者到另一个页面去）的时候。与 onload 一样，要写在 HTML 中就写到<body>标记里。

- window.onresize;发生在窗口被调整大小的时候。
- window.onblur;发生在窗口失去焦点的时候。
- window.onfocus;发生在窗口得到焦点的时候。
- window.onerror;发生在错误发生的时候。它的事件处理程序通常就叫做“错误处理程序”(Error Handler)，用来处理错误。上边已经介绍过，要忽略一切错误，就使用：

```
function ignoreError() {
  return true;
}
window.onerror = ignoreError;
```

JavaScript 中的 History 历史对象

JavaScript 中的 History 历史对象包含了用户已浏览的 URL 的信息，是指历史对象指浏览器的浏览历史。鉴于安全性的需要，该对象收到很多限制，现在只剩下下列属性和方法。History 历史对象有 length 这个属性，列出历史的项数。JavaScript 所能管到的历史被限制在用浏览器的“前进”“后退”键可以去到的范围。本属性返回的是“前进”和“后退”两个按键之下包含的地址数的和。

History 历史对象并有以下方法

- back() 后退，跟按下“后退”键是等效的。

- forward() 前进，跟按下“前进”键是等效的。
- go() 用法：history.go(x)；在历史的范围内去到指定的一个地址。如果 x < 0，则后退 x 个地址，如果 x > 0，则前进 x 个地址，如果 x == 0，则刷新现在打开的网页。history.go(0) 跟 location.reload() 是等效的。

JavaScript 中的 Location 地址对象

location 地址对象 它描述的是某一个窗口对象所打开的地址。要表示当前窗口的地址，只需要使用“location”就行了；若要表示某一个窗口的地址，就使用“<窗口对象>.location”。先前写了一片用 window.location.href 实现刷新另个框架页面 ，特此我看了一下 locaiton 的详细用法，对此有点改进，具体如下：

注意：属于不同协议或不同主机的两个地址之间不能互相引用对方的 location 对象，这是出于安全性的需要。例如，当前窗口打开的是“www.a.com”下面的某一页，另外一个窗口（对象名为：bWindow）打开的是“www.b.com”的网页。如果在当前窗口使用“bWindow.location”，就会出错：“没有权限”。这个错误是不能用错误处理程序（Event Handler，参阅 onerror 事件)来接收处理的。

第一、简单介绍一下 location 属性、用法以及相关示例：

Location

包含了关于当前 URL 的信息。location 对象描述了与一个给定的 Window 对象

关联的完整 URL。location 对象的每个属性都描述了 URL 的不同特性。

通常情况下 ,一个 URL 会有下面的格式 :协议//主机:端口/路径名称#哈希标识?搜索条件

例如 :http://www.ijavascript.cn/jiaocheng/index.html#topic1?x=7&y=2 这些部分是满足下列需求的：

- “协议”是 URL 的起始部分，直到包含到第一个冒号。
- “主机”描述了主机和域名，或者一个网络主机的 IP 地址。
- “端口”描述了服务器用于通讯的通讯端口。
- 路径名称描述了 URL 的路径方面的信息。
- “哈希标识”描述了 URL 中的锚名称，包括哈希掩码(#)。此属性只应用于 HTTP 的 URL。
- “搜索条件”描述了该 URL 中的任何查询信息，包括问号。此属性只应用于 HTTP 的 URL。“搜索条件”字符串包含变量和值的配对；每对之间由一个“&”连接。

属性概览

- protocol 返回地址的协议，取值为 'http:','https:','file:' 等等。

- hostname 返回地址的主机名，例如，一个“http://www.microsoft.com/china/”的地址，location.hostname == 'www.microsoft.com'。
- port 返回地址的端口号，一般 http 的端口号是 '80'。
- host 返回主机名和端口号，如：'www.a.com:8080'。
- pathname 返回路径名，如“http://www.a.com/b/c.html”，location.pathname == 'b/c.html'。
- hash 返回“#”以及以后的内容，如“http://www.a.com/b/c.html#chapter4”，location.hash == '#chapter4'；如果地址里没有“#”，则返回空字符串。
- search 返回“?”以及以后的内容，如“http://www.a.com/b/c.asp?selection=3&jumpto=4”,location.search =='?selection=3&jumpto=4'；如果地址里没有“?”，则返回空字符串。
- href 返回以上全部内容，也就是说，返回整个地址。在浏览器的地址栏上怎么显示它就怎么返回。如果想一个窗口对象打开某地址，可以使用“location.href = '...'”，也可以直接用“location = '...'”来达到此目的。

方法概览

- reload() 相当于按浏览器上的“刷新”(IE)或“Reload”(Netscape)键。
- replace() 打开一个 URL，并取代历史对象中当前位置的地址。用这个方法打开一个 URL 后，按下浏览器的“后退”键将不能返回到刚才的页面。

二、location 之页面跳转 js 如下：

```
//简单跳转
function gotoPage(url) {
// eg. var url =
"newsview.html?catalogid="+catalogID+"&pageid="+pageid;
window.location = url;
}
// 对 location 用法的升级，为单个页面传递参数
function goto_catalog(iCat) {
if(iCat<=0) {
top.location = "../index.aspx"; // top 出去
} else {
window.location = "../newsCat.aspx?catid="+iCat;
}
}
// 对指定框架进行跳转页面，二种方法皆可用
function goto_iframe(url) {
parent.mainFrame.location = "../index.aspx"; //
// parent.document.getElementById("mainFrame").src =
```

"../index.aspx";// use dom to change page // 同时我增加了 dom 的写法

}

// 对指定框架进行跳转页面，因为

parent.iframename.location="../index.aspx"; 方法不能实行，主要是

"parent.iframename" 中的 iframename 在 js 中被默认为节点,而不能把传递过来的参数转换过来,所以用 dom 实现了该传递二个参数的框架跳转页面,希望那位仁兄不吝赐教！

```
function goto_iframe(iframename,url)  {
parent.document.getElementById(iframename).src = "../index.aspx";//
use dom to change page by iframeName
//}
```

// 回到首页

```
function gohome() {
top.location = "/index.aspx";
}
```

JavaScript 中的 Document 文档对象

Document 文档对象是 JavaScript 中 window 和 frames 对象的一个属性,是显示于窗口或框架内的一个文档。描述当前窗口或指定窗口对象的文档。它包含了文档从<head>到</body>的内容。

用法：document （当前窗口） 或 <窗口对象>.document （指定窗口）

属性：

- document.title //设置文档标题等价于 HTML 的<title>标签
- document.bgColor //设置页面背景色
- document.fgColor //设置前景色(文本颜色)
- document.linkColor //未点击过的链接颜色
- document.alinkColor //激活链接(焦点在此链接上)的颜色
- document.vlinkColor //已点击过的链接颜色
- document.URL //设置 URL 属性从而在同一窗口打开另一网页
- document.fileCreatedDate //文件建立日期，只读属性
- document.fileModifiedDate //文件修改日期，只读属性
- document.fileSize //文件大小，只读属性
- document.cookie //设置和读出 cookie
- document.charset //设置字符集 简体中文:gb2312
- cookie 关于 cookie 请参看“使用框架和 Cookies”一章。
- lastModified 当前文档的最后修改日期，是一个 Date 对象。
- referrer 如果当前文档是通过点击连接打开的，则 referrer 返回原来的 URL。
- title 指<head>标记里用<title>...</title>定义的文字。在 Netscape 里本属性不接受赋值。
- fgColor 指<body>标记的 text 属性所表示的文本颜色。
- bgColor 指<body>标记的 bgcolor 属性所表示的背景颜色。
- linkColor 指<body>标记的 link 属性所表示的连接颜色。

- alinkColor 指<body>标记的 alink 属性所表示的活动连接颜色。
- vlinkColor 指<body>标记的 vlink 属性所表示的已访问连接颜色。

方法：

- open() 打开文档以便 JavaScript 能向文档的当前位置（指插入 JavaScript 的位置）写入数据。通常不需要用这个方法，在需要的时候 JavaScript 自动调用。
- write(); writeln() 向文档写入数据，所写入的会当成标准文档 HTML 来处理。writeln() 与 write() 的不同点在于，writeln() 在写入数据以后会加一个换行。这个换行只是在 HTML 中换行，具体情况能不能够是显示出来的文字换行，要看插入 JavaScript 的位置而定。如在<pre>标记中插入，这个换行也会体现在文档中。
- clear() 清空当前文档。
- close() 关闭文档，停止写入数据。如果用了 write[ln]() 或 clear() 方法，就一定要用 close() 方法来保证所做的更改能够显示出来。如果文档还没有完全读取，也就是说，JavaScript 是插在文档中的，那就不必使用该方法。

现在我们已经拥有足够的知识来做以下这个很多网站都有的弹出式更新通知了。

```
<script language="JavaScript">
var whatsNew = open('','_blank','top=50,left=50,width=200,height=300,'
+
                'menubar=no,toolbar=no,directories=no,location=no,' +
                'status=no,resizable=no,scrollbars=yes');
whatsNew.document.write('<center><b>更新通知</b></center>');
whatsNew.document.write('<p>最后更新日期：00.08.01');
whatsNew.document.write('<p>00.08.01：增加了“我的最爱”栏目。');
whatsNew.document.write('<p align="right">' +
                '<a href="javascript:self.close()">关闭窗口</a>');
whatsNew.document.close();
</script>
```

当然也可以先写好一个 HTML 文件，在 open() 方法中直接 load 这个文件。

JavaScript 中的其他对象

anchors[]; links[]; Link 连接对象

用法：document.anchors[[x]]; document.links[[x]]; <anchorId>; <linkId>

document.anchors 是一个数组，包含了文档中所有锚标记（包含 name 属性的<a>标记），按照在文档中的次序，从 0 开始给每个锚标记定义了一个下标。

document.links 也是一个数组，包含了文档中所有连接标记（包含 href 属性的<a>标记和<map>标记段里的<area>标记），按照在文档中的次序，从 0 开始给每个连接标记定义了一个下标。

如果一个<a>标记既有 name 属性，又有 href 属性，则它既是一个 Anchor

对象，又是一个 Link 对象。

在 IE 中，如果在<a>标记中添加“id="..."”属性，则这个<a>对象被赋予一个标识（ID），调用这个对象的时候只需要使用“<id>”就行了。很多文档部件都可以用这个方法来赋予 ID，但要注意不能有两个 ID 相同。

anchors 和 links 作为数组，有数组的属性和方法。单个 Anchor 对象没有属性；单个 Link 对象的属性见下。

属性

protocol; hostname; port; host; pathname; hash; search; href 与 location 对象相同。

target 返回/指定连接的目标窗口（字符串），与<a>标记里的 target 属性是一样的。

事件

onclick; onmouseover; onmouseout; onmousedown; onmouseup

applets[] Java **小程序对象** 它是一个数组，包含了文档中所有的 Applet 对象（Java 小程序）。作为一个数组，有数组的属性和方法。关于单个 Applet 对象的属性和方法，我引用一句话：“Applet 对象继承了 Java 小程序的所有公共属性和方法。”（英文原句：The Applet object inherits all public properties of the Java applet./The Applet object inherits all public methodss of the Java applet.）因为本人很厌恶 Java 小程序，所以对它的什么“公共”“私有”的问题不感兴趣，也就没有探讨了。

embeds[] **插件对象** 它是一个数组，包含了文档中所有的插件（<embed>标记）。因为每个插件的不同，每个 Embed 对象也有不同的属性和方法。

forms[]; Form **表单对象** document.forms[] 是一个数组，包含了文档中所有的表单(<form>)。要引用单个表单，可以用 document.forms[x]，但是一般来说，人们都会这样做：在<form>标记中加上“name="..."”属性，那么直接用“document.<表单名>”就可以引用了。

Form **对象的属性**

name 返回表单的名称，也就是<form name="...">属性。

action 返回/设定表单的提交地址，也就是<form action="...">属性。

method 返回/设定表单的提交方法，也就是<form method="...">属性。

target 返回/设定表单提交后返回的窗口，也就是<form target="...">属性。

encoding 返回/设定表单提交内容的编码方式，也就是<form enctype="...">属性。

length 返回该表单所含元素的数目。

方法

reset() 重置表单。这与按下“重置”按钮是一样的。

submit() 提交表单。这与按下“提交”按钮是一样的。

事件

onreset; onsubmit

以下从“Button”到“Textarea”都是表单的元素对象。

Button **按钮对象** 由“<input type="button">”指定。引用一个 Button 对象，可以使用“<文档对象>.<表单对象>.<按钮名称>”。<按钮名称>指在<input>标记中的“name="..."”属性的值。引用任意表单元素都可以用这种方法。

属性

name 返回/设定用<input name="...">指定的元素名称。

value 返回/设定用<input value="...">指定的元素的值。

form 返回包含本元素的表单对象。

方法

blur() 从对象中移走焦点。

focus() 让对象获得焦点。

click() 模拟鼠标点击该对象。

事件

onclick; onmousedown; onmouseup

Checkbox **复选框对象** 由“<input type="checkbox">”指定。

属性

name 返回/设定用<input name="...">指定的元素名称。

value 返回/设定用<input value="...">指定的元素的值。

form 返回包含本元素的表单对象。

checked 返回/设定该复选框对象是否被选中。这是一个布尔值。

defaultChecked 返回/设定该复选框对象默认是否被选中。这是一个布尔值。

方法

blur() 从对象中移走焦点。

focus() 让对象获得焦点。

click() 模拟鼠标点击该对象。

事件

onclick

elements[]; Element **表单元素对象** <表单对象>.elements 是一个数组，包含了该表单所有的对象。一般我们不用该数组，而直接引用各个具体的对象。

Hidden **隐藏对象** 由“<input type="hidden">”指定。

属性

name 返回/设定用<input name="...">指定的元素名称。

value 返回/设定用<input value="...">指定的元素的值。

form 返回包含本元素的表单对象。

Password **密码输入区对象** 由“<input type="password">”指定。

属性

name 返回/设定用<input name="...">指定的元素名称。

value 返回/设定密码输入区当前的值。

defaultValue 返回用<input value="...">指定的默认值。

form 返回包含本元素的表单对象。

方法

blur() 从对象中移走焦点。

focus() 让对象获得焦点。

select() 选中密码输入区里全部文本。

事件

onchange

Radio **单选域对象** 由“<input type="radio">”指定。一组 Radio 对象有共同的名称（name 属性），这样的话，document.formName.radioName 就成了一个数组。要访问单个 Radio 对象就要用：document.formName.radioName[x]。

单个 Radio 对象的属性

name 返回/设定用<input name="...">指定的元素名称。

value 返回/设定用<input value="...">指定的元素的值。

form 返回包含本元素的表单对象。

checked 返回/设定该单选域对象是否被选中。这是一个布尔值。

defaultChecked 返回/设定该对象默认是否被选中。这是一个布尔值。

方法

blur() 从对象中移走焦点。

focus() 让对象获得焦点。

click() 模拟鼠标点击该对象。

事件

onclick

Reset **重置按钮对象** 由“<input type="reset">”指定。因为 Reset 也是按钮，所以也有 Button 对象的属性和方法。至于“onclick”事件，一般用 Form 对象的 onreset 代替。

Select **选择区（下拉菜单、列表）对象** 由“<select>”指定。

属性

name 返回/设定用<input name="...">指定的元素名称。

length 返回 Select 对象下选项的数目。

selectedIndex 返回被选中的选项的下标。这个下标就是在 options[] 数组中该选项的位置。如果 Select 对象允许多项选择，则返回第一个被选中的选项的下标。

form 返回包含本元素的表单对象。

方法

blur() 从对象中移走焦点。

focus() 让对象获得焦点。

事件

onchange

options[]; Option **选择项对象** options[] 是一个数组，包含了在同一个 Select 对象下的 Option 对象。Option 对象由“<select>”下的“<options>”指定。

options[] **数组的属性**

length; selectedIndex 与所属 Select 对象的同名属性相同。

单个 Option 对象的属性

text 返回/指定 Option 对象所显示的文本

value 返回/指定 Option 对象的值，与<options value="...">一致。

index 返回该 Option 对象的下标。对此并没有什么好说，因为要指定特定的一个 Option 对象，都要先知道该对象的下标。这个属性好像没有什么用。

selected 返回/指定该对象是否被选中。通过指定 true 或者 false，可以动态的改变选中项。

defaultSelected 返回该对象默认是否被选中。true / false。

Submit 提交按钮对象 由“<input type="submit">指定。因为 Submit 也是按钮，所以也有 Button 对象的属性和方法。至于“onclick”事件，一般用 Form 对象的 onsubmit 代替。

Text 文本框对象 由“<input type="text">”指定。Password 对象也是 Text 对象的一种，所以 Password 对象所有的属性、方法和事件，Text 对象都有。

Textarea 多行文本输入区对象 由“<textarea>”指定。Textarea 对象所有的属性、方法和事件和 Text 对象相同，也就是跟 Password 对象一样。

images[]; Image 图片对象 document.images[] 是一个数组，包含了文档中所有的图片（<img>）。要引用单个图片，可以用 document.images[x]。如果某图片包含“name”属性，也就是用“<img name="...">”这种格式定义了一幅图片，就可以使用“document.images['...']”这种方法来引用图片。在 IE 中，如果某图片包含 ID 属性，也就是用“<img id="...">”这种格式定义了一幅图片，就可以直接使用“<imageID>”来引用图片。

单个 Image 对象的属性

name; src; lowsrc; width; height; vspace; hspace; border 这些属性跟<img>标记里的同名属性是一样的。在 Netscape 里，除了 src 属性，其它属性（几乎全部）都不能改的，即使改了，在文档中也不能显示出效果来。这些属性最有用的就是 src 了，通过对 src 属性赋值，可以实时的更改图片。

事件

onclick

不显示在文档中的 Image 对象

不显示在文档中的 Image 对象是用 var 语句定义的：

```
var myImage = new Image(); 或
var myImage = new Image(<图片地址字符串>);
```

然后就可以像一般 Image 对象一样对待 myImage 变量了。不过既然它不显示在文档中，以下属性：lowsrc, width, height, vspace, hspace, border 就没有什么用途了。一般这种对象只有一个用：预读图片(preload)。因为当对对象的 src 属性赋值的时候，整个文档的读取、JavaScript 的运行都暂停，让浏览器专心的读取图片。预读图片以后，浏览器的缓存里就有了图片的 Copy，到真正要把图片放到文档中的时候，图片就可以立刻显示了。现在的网页中经常会有一些图像连接，当鼠标指向它的时候，图像换成另外一幅图像，它们都是先预读图像的。

预读图像的 JavaScript 例子

```
var imagePreload = new Image();
imagePreload.src = '001.gif';
imagePreload.src = '002.gif';
imagePreload.src = '003.gif';
```

以上例子适合预读少量图片。

```
function imagePreload() {
  var imgPreload = new Image();
  for (i = 0; i < arguments.length; i++) {
    imgPreload.src = arguments[i];
  }
}
imagePreload('001.gif', '002.gif', '003.gif', '004.gif', '005.gif');
```

以上例子适合预读大量图片。

JavaScript 中的事件处理

事件处理概述

事件处理是对象化编程的一个很重要的环节，没有了事件处理，程序就会变得很死，缺乏灵活性。事件处理的过程可以这样表示：发生事件 - 启动事件处理程序 - 事件处理程序作出反应。其中，要使事件处理程序能够启动，必须先告诉对象，如果发生了什么事情，要启动什么处理程序，否则这个流程就不能进行下去。事件的处理程序可以是任意 JavaScript 语句，但是我们一般用特定的自定义函数（function）来处理事情。

指定事件处理程序有三种方法：

方法一 直接在 HTML 标记中指定。这种方法是用得最普遍的。方法是：

<标记 ... ... 事件="事件处理程序" [事件="事件处理程序" ...]>

让我们来看看例子：

<body ... onload="alert('网页读取完成，请慢慢欣赏！')" onunload="alert('再见！')">

这样的定义<body>标记，能使文档读取完毕的时候弹出一个对话框，写着“网页读取完成，请慢慢欣赏”；在用户退出文档（或者关闭窗口，或者到另一个页面去）的时候弹出“再见”。

方法二 编写特定对象特定事件的 JavaScript。这种方法用得比较少，但是在某些场合还是很好用的。方法是：

<script language="JavaScript" for="对象" event="事件">

...

(事件处理程序代码)

...

</script>

<script language="JavaScript" for="window" event="onload">

alert('网页读取完成，请慢慢欣赏！');

</script>

方法三 在 JavaScript 中说明。方法：

<事件主角 - 对象>.<事件> = <事件处理程序>;

用这种方法要注意的是，“事件处理程序”是真正的代码，而不是字符串形式的代码。如果事件处理程序是一个自定义函数，如无使用参数的需要，就不要加“()”。

```
function ignoreError() {
  return true;
}
window.onerror = ignoreError; // 没有使用“()”
```

这个例子将 ignoreError() 函数定义为 window 对象的 onerror 事件的处理程序。它的效果是忽略该 window 对象下任何错误（由引用不允许访问的 location 对象产生的“没有权限”错误是不能忽略的）。

事件详解

- onblur 事件 发生在窗口失去焦点的时候。应用于：window 对象
- onchange 事件 发生在文本输入区的内容被更改，然后焦点从文本输入区移走之后。捕捉此事件主要用于实时检测输入的有效性，或者立刻改变文

档内容。应用于 :Password 对象 ;Select 对象 ;Text 对象 ;Textarea 对象

- onclick 事件 发生在对象被单击的时候。单击是指鼠标停留在对象上，按下鼠标键，没有移动鼠标而放开鼠标键这一个完整的过程。一个普通按钮对象（Button）通常会有 onclick 事件处理程序，因为这种对象根本不能从用户那里得到任何信息，没有 onclick 事件处理程序就等于废柴。按钮上添加 onclick 事件处理程序，可以模拟“另一个提交按钮”，方法是：在事件处理程序中更改表单的 action, target, encoding, method 等一个或几个属性，然后调用表单的 submit() 方法。在 Link 对象的 onclick 事件处理程序中返回 false 值（return false），能阻止浏览器打开此连接。即，如果有一个这样的连接 :Go!，那么无论用户怎样点击，都不会去到 www.a.com 网站，除非用户禁止浏览器运行 JavaScript。应用于 :Button 对象；Checkbox 对象；Image 对象；Link 对象；Radio 对象；Reset 对象；Submit 对象
- onerror 事件 发生在错误发生的时候。它的事件处理程序通常就叫做“错误处理程序”(Error Handler)，用来处理错误。上边已经介绍过，要忽略一切错误，就使用：

```
function ignoreError() {
  return true;
}
window.onerror = ignoreError;
```

应用于：window 对象

- onfocus 事件 发生在窗口得到焦点的时候。应用于：window 对象
- onload 事件 发生在文档全部下载完毕的时候。全部下载完毕意味着不但 HTML 文件，而且包含的图片，插件，控件，小程序等全部内容都下载完毕。本事件是 window 的事件 ,但是在 HTML 中指定事件处理程序的时候，我们是把它写在<body>标记中的。应用于：window 对象
- onmousedown 事件 发生在用户把鼠标放在对象上按下鼠标键的时候。参考 onmouseup 事件。应用于：Button 对象；Link 对象
- onmouseout 事件 发生在鼠标离开对象的时候。参考 onmouseover 事件。应用于：Link 对象
- onmouseover 事件 发生在鼠标进入对象范围的时候。这个事件和 onmouseout 事件，再加上图片的预读，就可以做到当鼠标移到图像连接上，图像更改的效果了。有时我们看到，在指向一个连接时，状态栏上不显示地址，而显示其它的资料，看起来这些资料是可以随时更改的。它们是这样做出来的：

  <a href="..." onmouseover="window.status='Click Me Please!'; return true;" onmouseout="window.status=''; return true;">

  应用于：Link 对象

- onmouseup 事件 发生在用户把鼠标放在对象上鼠标键被按下的情况下，放开鼠标键的时候。如果按下鼠标键的时候，鼠标并不在放开鼠标的对象上，则本事件不会发生。应用于：Button 对象；Link 对象

- onreset 事件 发生在表单的“重置”按钮被单击(按下并放开)的时候。通过在事件处理程序中返回 false 值(return false)可以阻止表单重置。应用于：Form 对象
- onresize 事件 发生在窗口被调整大小的时候。应用于：window 对象
- onsubmit 事件 发生在表单的“提交”按钮被单击(按下并放开)的时候。可以使用该事件来验证表单的有效性。通过在事件处理程序中返回 false 值(return false)可以阻止表单提交。应用于：Form 对象
- onunload 事件 发生在用户退出文档(或者关闭窗口,或者到另一个页面去)的时候。与 onload 一样，要写在 HTML 中就写到<body>标记里。有的 Web Masters 用这个方法来弹出“调查表单”，以“强迫”来者填写；有的就弹出广告窗口，唆使来者点击连接。我觉得这种“onunload=″open...″”的方法很不好,有时甚至会因为弹出太多窗口而导致资源缺乏。有什么对来者说就应该在网页上说完,不对吗？ 应用于：window 对象

JavaScript 中的对象化编程

关于**对象化编**程的语句 现在我们有实力学习以下关于对象化编程，但其实属于上一章的内容了。

with **语句** 为一个或一组语句指定默认对象。

用法：

```
with (<对象>) <语句>;
```

with 语句通常用来缩短特定情形下必须写的代码量。在下面的例子中，请注意 Math 的重复使用：

```
x = Math.cos(3 * Math.PI) + Math.sin(Math.LN10);
y = Math.tan(14 * Math.E);
```

当使用 with 语句时，代码变得更短且更易读：

```
with (Math) {
  x = cos(3 * PI) + sin(LN10);
  y = tan(14 * E);
}
```

this 对象 返回“当前”对象。在不同的地方，this 代表不同的对象。如果在 JavaScript 的“主程序”中（不在任何 function 中，不在任何事件处理程序中）使用 this，它就代表 window 对象；如果在 with 语句块中使用 this，它就代表 with 所指定的对象；如果在事件处理程序中使用 this，它就代表发生事件的对象。

一个常用的 this 用法：

```
<script>
...
function check(formObj) {
  ...
}
...
```

```
</script>

<body ...>
...
<form ...>
...
<input type="text" ... onchange="check(this.form)">
...
</form>
...
</body>
```

这个用法常用于立刻检测表单输入的有效性。

自定义构造函数 我们已经知道，Array()，Image()等构造函数能让我们构造一个变量。其实我们自己也可以写自己的构造函数。自定义构造函数也是用 function。在 function 里边用 this 来定义属性。

```
function <构造函数名> [(<参数>)] {
  ...
  this.<属性名> = <初始值>;
  ...
}
```

然后，用 new 构造函数关键字来构造变量：

```
var <变量名> = new <构造函数名>[(<参数>)];
```

构造变量以后，<变量名>成为一个对象，它有它自己的属性——用 this 在 function 里设定的属性。

以下是一个从网上找到的搜集浏览器详细资料的自定义构造函数的例子：

```
function Is() {
  var agent = navigator.userAgent.toLowerCase();
  this.major = parseInt(navigator.appVersion);  //主版本号
  this.minor = parseFloat(navigator.appVersion);//全版本号
  this.ns = ((agent.indexOf('mozilla')!=-1) &&
             ((agent.indexOf('spoofer')==-1) && //是否 Netscape
              (agent.indexOf('compatible') == -1)));
  this.ns2 = (this.ns && (this.major == 3));    //是否 Netscape 2
  this.ns3 = (this.ns && (this.major == 3));    //是否 Netscape 3
  this.ns4b = (this.ns && (this.minor < 4.04)); //是否 Netscape 4 低版本
  this.ns4 = (this.ns && (this.major >= 4));    //是否 Netscape 4 高版本
  this.ie = (agent.indexOf("msie") != -1);      //是否 IE
  this.ie3 = (this.ie && (this.major == 2));    //是否 IE 3
  this.ie4 = (this.ie && (this.major >= 4));    //是否 IE 4
  this.op3 = (agent.indexOf("opera") != -1);    //是否 Opera 3
  this.win = (agent.indexOf("win")!=-1);        //是否 Windows 版本
  this.mac = (agent.indexOf("mac")!=-1);        //是否 Macintosh 版本
  this.unix = (agent.indexOf("x11")!=-1);       //是否 Unix 版本
}
```

```
var is = new Is();
```

这个构造函数非常完整的搜集了浏览器的信息。我们看到它为对象定义了很多个属性：major, minor, ns, ie, win, mac 等等。它们的意思见上面的注释。把 is 变量定义为 Is() 对象后，用 if (is.ns) 这种格式就可以很方便的知道浏览器的信息了。我们也可以从这个构造函数中看到，它也可以使用一般的 JavaScript 语句（上例中为 var 语句）。

让我们再来看一个使用参数的构造函数：

```
function myFriend(theName, gender, theAge, birthOn, theJob) {
  this.name = theName;
  this.isMale = (gender.toLowerCase == 'male');
  this.age = theAge;
  this.birthday = new Date(birthOn);
  this.job = theJob
}

var Stephen = new myFriend('Stephen', 'Male', 18, 'Dec 22, 1982',
'Student');
```

从这个构造函数我们不但看到了参数的用法，还看到了不同的属性用不同的数据型是可以的（上例五个属性分别为：字符串，布尔值，数字，日期，字符串），还看到了构造函数里也可以用构造函数来“构造”属性。如果用了足够的“保护措施”来避免无限循环，更可以用构造函数自身来构造自己的属性。

JavaScript 框架编程

使用 JavaScript 框架

在讲述 window 对象的时候，我们提到过，一个框架内的网页也是 window 对象，也就是说，Frame 对象也是 window 对象。用最容易理解的话说，每一个 HTML 文件占用一个 window 对象，包括定义框架的网页(“框架网页”)。在 IE 里用“<iframe>”标记在文档中插入的框架也是 window 对象，但是用“包含网页”的方法（在 HTML 中显示为“<!--webbot bot="include" ...-->”）读取的 HTML 就不占用独自的 window 对象。每一个框架都是包含它的页的 window 对象的一个子对象（不知道应该叫“属性”不该），要引用它，可以用以下几种方法之一：

```
window.frames[x]
window.frames['frameName']
window.frameName
```

其中，x 指的是该 window 对象中指定的第几个框架，与其它数组一样，x 也是从零开始的。frameName 指的是该框架的名字，跟<frame>里的“name”属性一样。

如果使用 window.frameName 指定的 window 对象又是一个框架网页，那么引用它的框架的方法：window.frameName.subFrameName。以此类推。

要注意的时，无论在何处，引用“window”对象所返回的，都是“当前”window 对象。如果要访问其它 window 对象，就要用到 parent 和 top 属性。parent 指的是“父级”window 对象，也就是包含当前 window 对象的框架网页；top 指的是窗口最顶端的 window 对象。

使用框架还要密切留意你的 JavaScript 中定义的全局变量和自定义函数。它们都有它们的所属——所在的 window 对象。要引用其它框架中的全局变量或自定义函数，都要用“窗口对象.框架对象[.框架对象…].全局变量或自定义函数”这种很烦的方法。

以上这个问题在建立连接时经常会被忽略 :如果在<head>中定义了一个默认目标窗口（<base target="...">），在<a href="javascript:...">中，要知道输入的 JavaScript 语句是在默认目标窗口中运行的，必要时加一些“parent”“top”属性。

框架编程概述

一个 HTML 页面可以有一个或多个子框架，这些子框架以<iframe>来标记，用来显示一个独立的 HTML 页面。这里所讲的框架编程包括框架的自我控制以及框架之间的互相访问，例如从一个框架中引用另一个框架中的 JavaScript 变量、调用其他框架内的函数、控制另一个框架中表单的行为等。

框架间的互相引用

一个页面中的所有框架以集合的形式作为 window 对象的属性提供，例如：window.frames 就表示该页面内所有框架的集合，这和表单对象、链接对象、图片对象等是类似的，不同的是，这些集合是 document 的属性。因此，要引用一个子框架，可以使用如下语法：

window.frames["frameName"];

window.frames.frameName

window.frames[index]

其中，window 字样也可以用 self 代替或省略，假设 frameName 为页面中第一个框架，则以下的写法是等价的：

self.frames["frameName"]

self.frames[0]

frames[0]

frameName

每个框架都对应一个 HTML 页面，所以这个框架也是一个独立的浏览器窗口，它具有窗口的所有性质，所谓对框架的引用也就是对 window 对象的引用。有了这个 window 对象，就可以很方便地对其中的页面进行操作，例如使用 window.document 对象向页面写入数据、使用 window.location 属性来改变框架内的页面等。

下面分别介绍不同层次框架间的互相引用：

1．父框架到子框架的引用

知道了上述原理，从父框架引用子框架变的非常容易，即：

window.frames["frameName"];

这样就引用了页面内名为 frameName 的子框架。如果要引用子框架内的子框架，根据引用的框架实际就是 window 对象的性质，可以这样实现：

window.frames["frameName"].frames["frameName2"];

这样就引用到了二级子框架，以此类推，可以实现多层框架的引用。

2．子框架到父框架的引用

每个 window 对象都有一个 parent 属性，表示它的父框架。如果该框架已经是顶层框架，则 window.parent 还表示该框架本身。

3．兄弟框架间的引用

如果两个框架同为一个框架的子框架，它们称为兄弟框架，可以通过父框架来实现互相引用，例如一个页面包括 2 个子框架：

```
<frameset rows="50%,50%">
      <frame src="1.html" />
      <frame src="2.html" />
</frameset>
```

在 frame1 中可以使用如下语句来引用 frame2：

```
self.parent.frames["frame2"];
```

4．不同层次框架间的互相引用

框架的层次是针对顶层框架而言的。当层次不同时，只要知道自己所在的层次以及另一个框架所在的层次和名字，利用框架引用的 window 对象性质，可以很容易地实现互相访问，例如：

```
self.parent.frames["childName"].frames["targetFrameName"];
```

5．对顶层框架的引用

和 parent 属性类似，window 对象还有一个 top 属性。它表示对顶层框架的引用，这可以用来判断一个框架自身是否为顶层框架，例如：

```
//判断本框架是否为顶层框架
if(self==top){
        //dosomething
}
```

改变框架的载入页面

对框架的引用就是对 window 对象的引用，利用 window 对象的 location 属性，可以改变框架的导航，例如：

```
window.frames[0].location="1.html";
```

这就将页面中第一个框架的页面重定向到 1.html，利用这个性质，甚至可以使用一条链接来更新多个框架。

```
<frameset rows="50%,50%">
      <frame src="1.html" />
      <frame src="2.html" />
</frameset>
<!--somecode-->
<a href="frame1.location='3.html;frame2.location='4.html'" >link</a>
<!--somecode-->
```

引用其他框架内的 JavaScript 变量和函数

在介绍引用其他框架内 JavaScript 变量和函数的技术之前，先来看以下代码：

```
<script language="JavaScript" type="text/javascript">
<!--
function hello(){
       alert("hello,ajax!");
}
```

```
window.hello();
//-->
</script>
```

如果运行了这段代码，会弹出“hello,ajax!”的窗口，这正是执行 hello()函数的结果。那为什么 hello()变成了 window 对象的方法呢？因为在一个页面内定义的所有全局变量和全局函数都是作为 window 对象的成员。例如：

```
var a=1;
alert(window.a);
```

就会弹出对话框显示为 1。同样的原理，在不同框架之间共享变量和函数，就是要通过 window 对象来调用。

例如：一个商品浏览页面由两个子框架组成，左侧表示商品分类的链接；当用户单击分类链接时，右侧显示相应的商品列表；用户可以单击商品旁的【购买】链接将商品加入购物车。

在这个例子中，可以利用左侧导航页面来存储用户希望购买的商品，因为当用户单击导航链接时，变化的是另外一个页面，即商品展示页面，而导航页面本身是不变的，因此其中的 JavaScript 变量不会丢失，可以用来存储全局数据。其实现原理如下：

假设左侧页面为 link.html，右侧页面为 show.html，页面结构如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title> New Document </title>
</head>
<frameset cols="20%,80%">
       <frame src="link.html" />
```

```
        <frame src="show.html" />
</frameset>
</html>
```

在 show.html 中展示的商品旁边可以加入这样一条语句：

```
<a href="void(0)" >加入购物车</a>
```

其中 link 表示导航框架，在 link.html 页面中定义了 arrOrders 数组来存储商品的 id，函数 addToOrders()用来响应商品旁边【购买】链接的单击事件，它接收的参数 id 表示商品的 id，例子中是一个 id 为 32068 的商品：

```
<script language="JavaScript" type="text/javascript">
<!--
var arrOrders=new Array();
function addToOrders(id){
        arrOrders.push(id);
}
//-->
</script>
```

这样，在结帐页面或是购物车浏览页面就可以用 arrOrders 来获取所有准备购买的商品。

框架可以使一个页面划分为功能独立的多个模块，每个模块之间彼此独立，但又可以通过 window 对象的引用来建立联系，是 Web 开发中的一个重要机制。在 Ajax 开发中，还可以利用隐藏框架实现各种技巧，在后面介绍 Ajax 实例编程时可以发现，无刷新上传文件以及解决 Ajax 的前进后退问题都会用到这种技术。

JavaScript 的 Cookies

使用 Cookies　　我们已经知道，在 document 对象中有一个 cookie 属性。但是 Cookie 又是什么？“某些 Web 站点在您的硬盘上用很小的文本文件存储了一些信息，这些文件就称为 Cookie。”——MSIE 帮助。一般来说，Cookies 是 CGI 或类似，比 HTML 高级的文件、程序等创建的，但是 JavaScript 也提供了对 Cookies 的很全面的访问权利。

在继续之前，我们先要学一学 **Cookie 的基本知识。**

每个 Cookie 都是这样的：<cookie 名>=<值>

<cookie 名>的限制与 JavaScript 的命名限制大同小异，少了“不能用 JavaScript 关键字”，多了“只能用可以用在 URL 编码中的字符”。后者比较难懂，但是只要你只用字母和数字命名，就完全没有问题了。<值>的要求也是“只能用可以用在 URL 编码中的字符”。

每个 Cookie 都有失效日期，一旦电脑的时钟过了失效日期，这个 Cookie 就会被删掉。我们不能直接删掉一个 Cookie，但是可以用设定失效日期早于现在时刻的方法来间接删掉它。

每个网页，或者说每个站点，都有它自己的 Cookies，这些 Cookies 只能由这个站点下的网页来访问，来自其他站点或同一站点下未经授权的区域的网页，是不能访问的。每一“组”Cookies 有规定的总大小（大约 2KB 每“组”），一超过最大总大小，则最早失效的 Cookie 先被删除，来让新的 Cookie“安家”。

现在我们来学习**使用 document.cookie 属性。**

如果直接使用 document.cookie 属性，或者说，用某种方法，例如给变量赋值，来获得 document.cookie 的值，我们就可以知道在现在的文档中有多少

个 Cookies，每个 Cookies 的名字，和它的值。例如，在某文档中添加“document.write(document.cookie)”，结果显示：

name=kevin; email=kevin@kevin.com; lastvisited=index.html

这意味着，文档包含 3 个 Cookies：name, email 和 lastvisited，它们的值分别是 kevin, kevin@kevin.com 和 index.html。可以看到，两个 Cookies 之间是用分号和空格隔开的，于是我们可以用 cookieString.split(';') 方法得到每个 Cookie 分开的一个数组(先用 var cookieString = document.cookie)。

设定一个 Cookie 的方法是对 document.cookie 赋值。与其它情况下的赋值不同，向 document.cookie 赋值不会删除掉原有的 Cookies，而只会增添 Cookies 或更改原有 Cookie。赋值的格式：

```
document.cookie = 'cookieName=' + escape('cookieValue')
    + ';expires=' + expirationDateObj.toGMTString();
```

是不是看到头晕了呢？以上不是粗体字的地方是要照抄不误的，粗体字是要按实际情况做出改动的。cookieName 表示 Cookie 的名称，cookieValue 表示 Cookie 的值，expirationDateObj 表示储存着失效日期的日期对象名，如果不需要指定失效日期，则不需要第二行。不指定失效日期，则浏览器默认是在关闭浏览器（也就是关闭所有窗口）之后过期。

看到了上面的一些下划线了么？这些是应该注意的地方。

首先 escape() 方法：为什么一定要用？因为 Cookie 的值的要求是“只能用可以用在 URL 编码中的字符”。我们知道“escape()”方法是把字符串按 URL 编码方法来编码的，那我们只需要用一个“escape()”方法来处理输出到 Cookie 的值，用“unescape()”来处理从 Cookie 接收过来的值就万无一失了。

而且这两个方法的最常用途就是处理 Cookies。其实设定一个 Cookie 只是“document.cookie = 'cookieName=cookieValue'”这么简单，但是为了避免在 cookieValue 中出现 URL 里不准出现的字符，还是用一个 escape() 好。

然后“expires”前面的分号：注意到就行了。是分号而不是其他。

最后 toGMTString() 方法 :设定 Cookie 的时效日期都是用 GMT 格式的时间的，其它格式的时间是没有作用的。

现在我们来实战一下。设定一个“name=rose”的 Cookie，在 3 个月后过期。

```
var expires = new Date();
expires.setTime(expires.getTime() + 3 * 30 * 24 * 60 * 60 * 1000);
/*   三个月 x 一个月当作 30 天 x 一天 24 小时
   x 一小时 60 分 x 一分 60 秒 x 一秒 1000 毫秒 */
document.cookie = 'name=rose;expires=' + expires.toGMTString();
```

为什么没有用 escape() 方法？这是因为我们知道 rose 是一个合法的 URL 编码字符串，也就是说，'rose' == escape('rose')。一般来说，如果设定 Cookie 时不用 escape()，那获取 Cookie 时也不用 unescape()。

再来一次：编写一个函数，作用是查找指定 Cookie 的值。

```
function getCookie(cookieName) {
  var cookieString = document.cookie;
  var start = cookieString.indexOf(cookieName + '=');
  // 加上等号的原因是避免在某些 Cookie 的值里有
  // 与 cookieName 一样的字符串。
  if (start == -1) // 找不到
```

```
    return null;
  start += cookieName.length + 1;
  var end = cookieString.indexOf(';', start);
  if (end == -1) return unescape(cookieString.substring(start));
  return unescape(cookieString.substring(start, end));
}
```

这个函数用到了字符串对象的一些方法，如果你不记得了（你是不是这般没记性啊），请快去查查。这个函数所有的 if 语句都没有带上 else，这是因为如果条件成立，程序运行的都是 return 语句，在函数里碰上 return，就会终止运行，所以不加 else 也没问题。该函数在找到 Cookie 时，就会返回 Cookie 的值，否则返回“null”。

现在我们要删除刚才设定的 name=rose Cookie。

```
var expires = new Date();
expires.setTime(expires.getTime() - 1);
document.cookie = 'name=rose;expires=' + expires.toGMTString();
```

可以看到，只需要把失效日期改成比现在日期早一点（这里是早 1 毫秒），再用同样的方法设定 Cookie，就可以删掉 Cookie 了。