第 1 章　JavaScript 语言概述

JavaScript 是目前 Web 应用程序开发者使用最为广泛的客户端脚本编程语言，它不仅可用来开发交互式的 Web 页面，更重要的是它将 HTML、XML 和 Java applet、flash 等功能强大的 Web 对象有机结合起来，使开发人员能快捷生成 Internet 或 Intranet 上使用的分布式应用程序。另外由于 Windows 对其最为完善的支持并提供二次开发的接口来访问操作系统各组件并实施相应的管理功能，JavaScript 成为继.bat(批处理文件)以来 Windows 系统里使用最为广泛的脚本语言。

1.1　JavaScript 是什么

应用程序开发者在学习一门新语言之前，兴趣肯定聚焦在诸如“它是什么”、“它能做什么”等问题而不是“如何开发”等问题上面。同样，学习 JavaScript 脚本，首先来揭开 JavaScript 脚本的面纱：“JavaScript 是什么？”

1.1.1　JavaScript 简史

二十世纪 90 年代中期，大部分因特网用户使用 28.8kbit/s 的 Modem 连接到网络进行网上冲浪，为解决网页功能简单的问题，HTML 文档已经变得越来越复杂和庞大，更让用户痛苦的是，为验证一个表单的有效性，客户端必须与服务器端进行多次的数据交互。难以想象这样的情景：当用户填完表单单击鼠标提交后，经过漫长的几十秒等待，服务器端返回的不是“提交成功”的喜悦却是“某某字段必须为阿拉伯数字，请单击按钮返回上一页面重新填写表单！”的错误提示！当时业界已经开始考虑开发一种客户端脚本语言来处理诸如验证表单合法性等简单而实用的问题。

1995 年 Netscape 公司和 Sun 公司联合开发出 JavaScript 脚本语言，并在其 Netscape Navigator 2 中实现了 JavaScript 脚本规范的第一个版本即 JavaScript 1.0 版，不久就显示了其强大的生机和发展潜力。由于当时 Netscape Navigator 主宰着 Web 浏览器市场，而 Microsoft 的 IE 则扮演追赶者的角色，为了跟上 Netscape 步伐，Microsoft 在其 Internet Explorer 3 中以 JScript 为名发布了一个 JavaScript 的克隆版本 JScript 1.0。

1997 年，为了避免无序竞争，同时解决 JavaScript 几个版本语法、特性等方面的混乱，JavaScript 1.1 作为草案提交给 ECMA（欧洲计算机厂商协会），并由 Netscape、Sun、Microsoft、Borland 及其它一些对脚本语言比较感兴趣的公司组成的 TC39（第 39 技术委员会：以下简称 TC39）协商并推出了 ECMA-262 规范版本，其定义了以 JavaScript 为蓝本、全新的 ECMAScript 脚本语言。

ECMA-262 标准 Edition 1 删除了 JavaScript 1.1 中与浏览器相关的部分，同时要求对象是平台无关的并且支持 Unicode 标准。

在接下来的几年，ISO/IEC（估计标准化组织/国际电工委员会）采纳 ECMAScript 作为 Web 脚本语言标准（ISO/IEC-16262）。从此，ECMAScript 作为 JavaScript 脚本的基础开始得到越来越多的浏览器厂商在不同程度上支持。

为了与ISO/IEC-16262 标准严格一致，ECMA-262 标准发布Edition2，此版本并没有添加、

更改和删除内容。ECMA-262 标准Edition 3 提供了对字符串处理、错误定义和数值输出等方面的更新，同时增加了对try…catch异常处理、正则表达式、新的控制语句等方面的完美支持，它标志着ECMAScript成为一门真正的编程语言，以ECMAScript为核心的JavaScript脚本语言得到了迅猛的发展。ECMA-262 标准Edition 4 正在制定过程中，可能明确的类的定义方法和命名空间等概念。表 1.1 是ECMA-262 标准四个版本之间的异同及浏览器支持情况。

表 1.1 ECMA-262 标准各版本间异同及浏览器支持情况

ECMA版本	特性	浏览器支持
Edition 1	删除了JavaScript 1.1中与浏览器相关的部分，同时要求对象是平台无关的并且支持Unicode标准	Netscape Navigators 4(.06版)、Internet Explorer 5
Edition 2	提供与ISO/IEC-16262标准的严格一致	Opera 6.0-7.1
Edition 3	提供了对字符串处理、错误定义和数值输出等方面的更新，同时增加了对try…catch异常处理、正则表达式、新的控制语句等方面的完美支持	Internet Explorer 5.5+、Netscape Navigators 6.0+、Opera 7.2+、Safari 1.0+
Edition 4*	可能明确的类的定义方法和命名空间等概念	未知（此版本正在制订过程中）

1999 年 6 月 ECMA 发布 ECMA-290 标准，其主要添加用 ECMAScript 来开发可复用组件的内容。

2005 年 12 月 ECMA 发布 ECMA-357 标准（ISO/IEC 22537）出台，主要增加对扩展标记语言 XML 的有效支持。

注意：JavaScript 脚本也能进行服务器端应用程序的开发，但相对于客户端的功能和应用范围而言，一般仍将其作为一门客户端脚本语言对待，后面有专门章节讲述服务器端 JavaScript 脚本。

对 JavaScript 历史的了解有助于开发者迅速掌握这门语言，同时也能加深对 JavaScript 语言潜力的理解。下面介绍其语言特点。

1.1.2 JavaScript 有何特点

JavaScript 是一种基于对象和事件驱动并具有相对安全性的客户端脚本语言，主要用于创建具有交互性较强的动态页面。主要具有如下特点：

- 基于对象：JavaScript 是基于对象的脚本编程语言，能通过 DOM（文档结构模型）及自身提供的对象及操作方法来实现所需的功能。
- 事件驱动：JavaScript 采用事件驱动方式，能响应键盘事件、鼠标事件及浏览器窗口事件等，并执行指定的操作。
- 解释性语言：JavaScript 是一种解释性脚本语言，无需专门编译器编译，而是在嵌入 JavaScript 脚本的 HTML 文档载入时被浏览器逐行地解释，大量节省客户端与服务器端进行数据交互的时间。
- 实时性：JavaScript 事件处理是实时的，无须经服务器就可以直接对客户端的事件做出响应，并用处理结果实时更新目标页面。
- 动态性：JavaScript 提供简单高效的语言流程，灵活处理对象的各种方法和属性，同时及时响应文档页面事件，实现页面的交互性和动态性。
- 跨平台：JavaScript 脚本的正确运行依赖于浏览器，而与具体的操作系统无关。只要客户端装有支持 JavaScript 脚本的浏览器，JavaScript 脚本运行结果就能正确反映在客户端浏览器平台上。
- 开发使用简单：JavaScript 基本结构类似 C 语言，采用小程序段的方式编程，并提供了简易的开发平台和便捷的开发流程，就可以嵌入到 HTML 文档中供浏览器解释执行。同时 JavaScript 的变量类型是弱类型，使用不严格。

- 相对安全性：JavaScript 是客户端脚本，通过浏览器解释执行。它不允许访问本地的硬盘，并且不能将数据存入到服务器上，不允许对网络文档进行修改和删除，只能通过浏览器实现信息浏览或动态交互，从而有效地防止数据的丢失。

综上所述，JavaScript 是一种有较强生命力和发展潜力的脚本描述语言，它可以被直接嵌入到 HTML 文档中，供浏览器解释执行，直接响应客户端事件如验证数据表单合法性，并调用相应的处理方法，迅速返回处理结果并更新页面，实现 Web 交互性和动态的要求，同时将大部分的工作交给客户端处理，将 Web 服务器的资源消耗降到最低。

> 注意：之所以说相对安全性，是因为 JavaScript 代码嵌入到 HTML 页面中，在客户端浏览该页面过程中，浏览器自动解释执行该代码，且不需要用户的任何操作，给用户带来额外的执行恶意代码的风险。

1.1.3 JavaScript 能做什么

JavaScript 脚本语言由于其效率高、功能强大等特点，在表单数据合法性验证、网页特效、交互式菜单、动态页面、数值计算等方面获得广泛的应用，甚至出现了完全使用 JavaScript 编写的基于 Web 浏览器的类 Unix 操作系统 JS/UIX 和无需安装即可使用的中文输入法程序 JustInput，可见 JavaScript 脚本编程能力不容小觑！下面仅介绍 JavaScript 常用功能。

> 注意：JS/UIX（系统测试：http://www.masswerk.at/jsuix/，命令手册：http://www.masswerk.at/ jsuix/man.txt，说明文档：http://www.masswerk.at/jsuix/jsuix-documentation.txt）；JustInput（官方网站 http://justinput.com/）

1. 表单数据合法性验证

使用 JavaScript 脚本语言能有效验证客户端提交的表单上数据的合法性，如数据合法则执行下一步操作，否则返回错误提示信息，如图 1.1 所示。

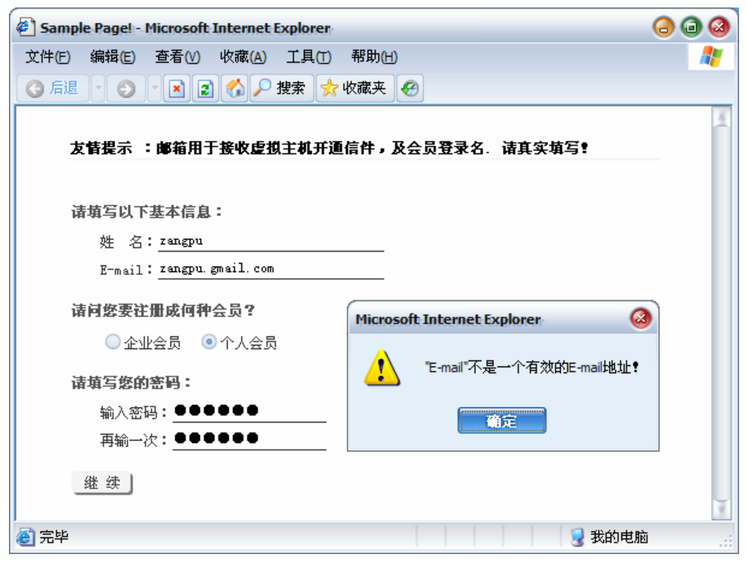

图 1.1 应用之一：表单数据合法性验证

2. 网页特效

使用 JavaScript 脚本语言，结合 DOM 和 CSS 能创建绚丽多彩的网页特效，如火焰状闪烁文字、文字环绕光标旋转等。火焰状闪烁文字效果如图 1.2 所示。

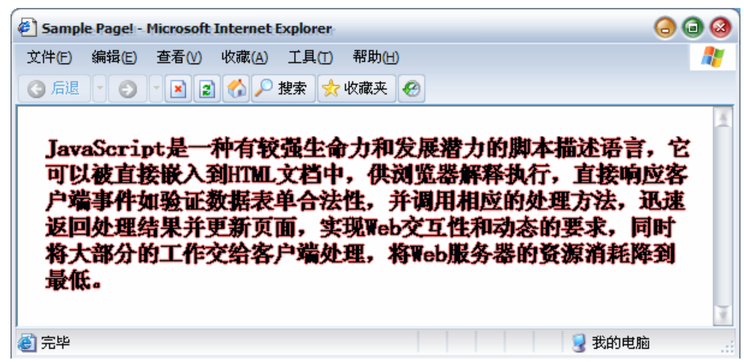

图 1.2 应用之二：火焰状闪烁文字特效

3. 交互式菜单

使用 JavaScript 脚本可以创建具有动态效果的交互式菜单，完全可以与 falsh 制作的页面导航菜单相媲美。如图 1.3 所示，鼠标在文档内任何位置单击，在其周围出现如下图所示的导航菜单。

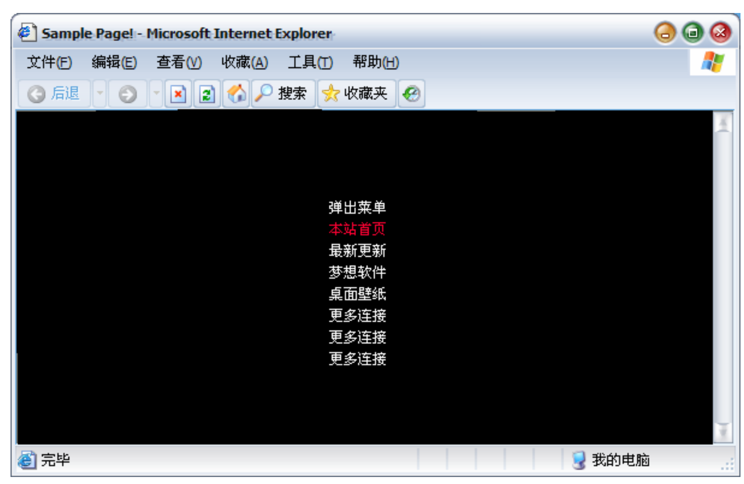

图 1.3 应用之三：动态的交互式菜单

4. 动态页面

使用 JavaScript 脚本可以对 Web 页面的所有元素对象进行访问并使用对象的方法访问并修改其属性实现动态页面效果，其典型应用如网页版俄罗斯方块、扑克牌游戏等。如图 1.4 所示为网页版俄罗斯方块游戏。

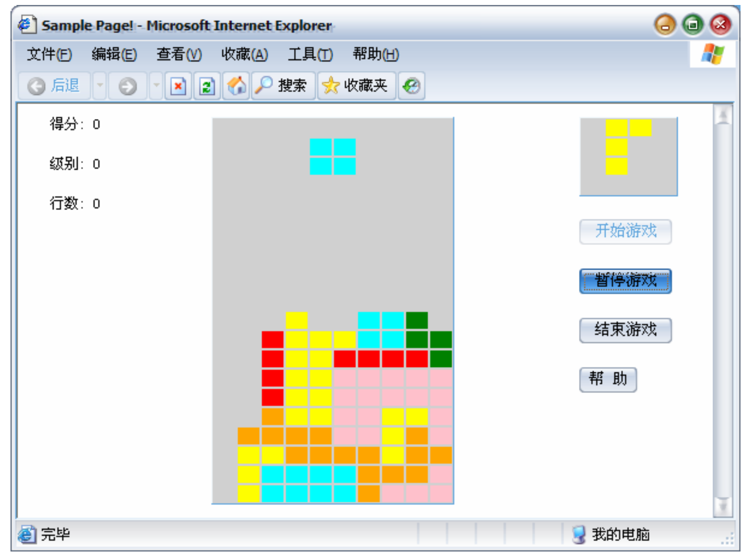

图 1.4 应用之四：使用 JavaScript 脚本的网页版俄罗斯方块游戏

5. 数值计算

JavaScript 脚本将数据类型作为对象，并提供丰富的操作方法使得 JavaScript 用于数值计算。如图 1.5 所示为用 JavaScript 脚本编写的计算器。

图 1.5 应用之五：使用 JavaScript 脚本编写的网页版计算器

JavaScript 脚本的应用远非如此，Web 应用程序开发者能将其与 XML 有机结合，并嵌入 Java applet 和 flash 等小插件，就能实现功能强大并集可视性、动态性和交互性于一体的 HTML 网页，吸引更多的客户来浏览该网站。

使用 DOM 所定义的文档结构，JavaScript 可用于多框架的 HTML 页面中框架之间的数据交互。同时 Windows 提供给 JavaScript 特有的二次编程接口，客户端可以通过编写非常短小的 JavaScript 脚本文件（.js 格式），通过内嵌的解释执行平台 WSH（Windows Script Host：Windows 脚本宿主，以下简称 WSH）解释来实现高效的文件系统管理。

注意：1、任何一种语言都是伟大的，都可以做很多事情，包括很不可思议的事情，但有一些是有意义

的，另一些是没有意义的，只是语言的侧重点不同而已。

1.1.4 JavaScript 的未来如何

自 ECMA-262 标准以来，JavaScript 及其派生语言如 Flash MX 中的 ActionScript、微软的 JScript 等在很多不同的编程环境中得到了大量的应用，同时 TC39 一直积极促进 JavaScript 新标准的出台。2005 年 12 月 ECMAScript for XML (E4X) Specification 作为 ECMA-357 标准（ISO/IEC 22537）出台，主要增加对扩展标记语言 XML 的有效支持：

- 在 ECMAScript 中定义了 XML 的语法和语义，同时将 XML 的数据类型添加进 ECMAScript 类型库中；
- 专门为 XML 扩展、修订和增加了少数操作符（operators），如搜索（searching）、过滤（filtering）等，同时增加对 XML 名字空间（namespaces）等的支持。

ECMA-357 标准是 JavaScript 发展史上的变革点，显示出对传统 ECMAScript 从根本上的改变，采用一种操作简单而功能强大的方式来支持 XML。

ECMAScript 4 作为下一个事实版本（在 IE 7 和 Mozilla 上已部分实现其功能），已作为提案最先由 Netscape 提出，接着 Microsoft 也将自己的修改意见提交给 TC39。由于 TC39 各成员的观点存在较大的分歧，主要是不能很好统一有关 JavaScript 未来发展方向的意见，该标准到本书截稿时还未获通过。

从 IE 5.5 版本发布开始，Microsoft 就没有更新过它基于浏览器的 JavaScript 实现策略，但在.NET Framework 中包含了 JScript.NET 作为 ECMAScript 4 的实现，它不能被浏览器解释执行，而只能通过特有编译器编译后作为独立的应用程序来使用。

令人意想不到的是，一直特立独行的苹果（Apple）电脑在其操作系统 MacOS X Tiger 版本中支持名为 DashBoard 的新型开发平台，它使用 JavaScript 脚本来创建轻量级应用程序，并能在 MacOS 桌面环境中运行。

JavaScript 作为一门语言依然在发展，虽然发展方向不太确定，其逐渐走出 Web 世界进入桌面应用领域也只是一种可能，但可以肯定的是，迎接 JavaScript 脚本语言的将是十分美好的前景。

注意：有关ECMAScript for XML (E4X) Specification(即ECMA-357 标准)的具体内容请参见其官方文档：
www.ecma-international.org/publications/files/ECMA-ST/ECMA-357.pdf

1.2 JavaScript 编程起步

JavaScript 脚本已经成为 Web 应用程序开发的一门炙手可热的语言，成为客户端脚本的首选。网络上充斥着形态各异的 JavaScript 脚本实现不同的功能，但用户也许并不了解 JavaScript 脚本是如何被浏览器中解释执行，更不知如何开始编写自己的 JavaScript 脚本来实现自己想要实现的效果。本节将一步步带领读者踏入 JavaScript 脚本语言编程的大门。

1.2.1 “Hello World!”程序

像学习 C、Java 等其他语言一样，先来看看使用 JavaScript 脚本语言编写的“Hello World!”程序：

```
//源程序 1.1
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
  <title>Sample Page!</title>
</head>
<body>
<br>
<center>
<script language="javascript 1.2" type="text/javascript">
   document.write("Hello World!");
</script>
</center>
</body>
</html>
```

将上述代码保存为.html(或.html)文件并双击打开，系统调用默认浏览器解释执行，结果如图 1.6 所示。

图 1.6 JavaScript 编写的“Hello World！”程序

JavaScript 脚本编程一般分为如下步骤：

- 选择 JavaScript 语言编辑器编辑脚本代码；
- 嵌入该 JavaScript 脚本代码到 HTML 文档中；
- 选择支持 JavaScript 的浏览器浏览该 HTML 文档；
- 如果错误则检查并修正源代码，重新浏览，此过程重复直至代码正确为止；
- 处理不支持 JavaScript 脚本的情况。

由于 JavaScript 脚本代码是嵌入到 HTML 文档中被浏览器解释执行，所以开发 JavaScript 脚本代码并不需要特殊的编程环境，只需要普通的文本编辑器和支持 JavaScript 脚本的浏览器即可。那么如何选择 JavaScript 脚本编辑器呢？

1.2.2 选择 JavaScript 脚本编辑器

编写 JavaScript 脚本代码可以选择普通的文本编辑器，如 Windows Notepad、UltraEdit 等，只要所选编辑器能将所编辑的代码最终保存为 HTML 文档类型(.htm、.html 等)即可。

虽然 Dreamweaver、Microsoft FrontPage 等专业网页开发工具套件内部集成了 JavaScript

脚本的开发环境，但笔者依然建议JavaScript脚本程序开发者在起步阶段使用最基本的文本编辑器如NotePad、UltraEdit等进行开发，以便把注意力放在JavaScript脚本语言而不是开发环境上。

同时，如果脚本代码出现错误，可用该编辑器打开源文件（.html、.html 或.js）修改后保存，并重新使用浏览器浏览，重复此过程直到没有错误出现为止。

> 注意：.js为JavaScript纯脚本代码文件的保存格式，该格式在通过<script>标记的src属性引入JavaScript脚本代码的方式中使用，用于嵌入外部脚本文件*.js。

1.2.3 引入JavaScript脚本代码到HTML文档中

将JavaScript脚本嵌入到HTML文档中有4种标准方法：

- 代码包含于<script>和</script>标记对，然后嵌入到HTML文档中；
- 通过<script>标记的src属性链接外部的JavaScript脚本文件；
- 通过JavaScript伪URL地址引入；
- 通过HTML文档事件处理程序引入。

下面分别介绍JavaScript脚本的几种标准引入方法：

1. 通过<script>与</script>标记对引入

在源程序 1.1 的代码中除了<script>与</script>标记对之间的内容外，都是最基本的HTML代码，可见<script>和</script>标记对将JavaScript脚本代码封装并嵌入到HTML文档中：

```
document.write("Hello World!");
```

浏览器载入嵌有JavaScript脚本的HTML文档时，能自动识别JavaScript脚本代码起始标记<script>和结束标记</script>，并将其间的代码按照解释JavaScript脚本代码的方法加以解释，然后将解释结果返回HTML文档并在浏览器窗口显示。

> 注意：所谓标记对，就是必须成对出现的标记，否则其间的脚本代码不能被浏览器解释执行。

来看看下面的代码：

```
<script language="javascript 1.2" type="text/javascript">
    document.write("Hello World!");
</script>
```

首先，<script>和</script>标记对将JavaScript脚本代码封装，同时告诉浏览器其间的代码为JavaScript脚本代码，然后调用document文档对象的write( )方法写字符串到HTML文档中。

下面重点介绍<script>标记的几个属性：

- language 属性：用于指定封装代码的脚本语言及版本，有的浏览器还支持 perl、VBScript 等，所有脚本浏览器都支持 JavaScript(当然，非常老的版本除外)，同时language属性默认值也为JavaScript；
- type属性：指定<script>和</script>标记对之间插入的脚本代码类型；
- src 属性：用于将外部的脚本文件内容嵌入到当前文档中，一般在较新版本的浏览器中使用，使用JavaScript脚本编写的外部脚本文件必须使用.js为扩展名，同时在<script>和</script>标记对中不包含任何内容，如下：

```
<script language="JavaScript 1.2" type="text/javascript" src="Sample.js">
</script>
```

> 注意：W3C HTML标准中不推荐使用language语法，要标记所使用的脚本类型，应设置<script>的type

属性为对应脚本的 MIME 类型（JavaScript 的 MIME 类型为“text/javascript”）。但在该属性中可设定所使用脚本的版本，有利于根据浏览器支持的脚本版本来编写有针对性的脚本代码。

下面讨论<script>标记的 src 属性如何引入 JavaScript 脚本代码。

2. 通过<script>标记的 src 属性引入

改写源程序 1.1 的代码并保存为 test.html：

```
//源程序 1.2
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
  <title>Sample Page!</title>
</head>
<body>
<script language="javascript 1.2" type="text/javascript" src="1.js">
</script>
</body>
</html>
```

同时在文本编辑器中编辑如下代码并将其保存为 1.js：

```
document.write("Hello World!");
```

将 test.html 和 1.js 文件放置于同一目录，双击运行 test.html，结果如图 1.6 所示。

可见通过外部引入 JavaScript 脚本文件的方式，能实现同样的功能。同时具有如下优点：

- 将脚本程序同现有页面的逻辑结构及浏览器结果分离。通过外部脚本，可以轻易实现多个页面共用完成同一功能的脚本文件，以便通过更新一个脚本文件内容达到批量更新的目的；
- 浏览器可以实现对目标脚本文件的高速缓存，避免额外的由于引用同样功能的脚本代码而导致下载时间的增加。

与 C 语言使用外部头文件（.h 文件等）相似，引入 JavaScript 脚本代码时使用外部脚本文件的方式符合结构化编程思想，但也有不利的一面，主要表现在如下几点：

- 不是所有支持 JavaScript 脚本的浏览器都支持外部脚本，如 Netscape 2 和 Internet Explorer 3 及以下版本都不支持外部脚本。
- 外部脚本文件功能过于复杂或其他原因导致的加载时间过长有可能导致页面事件得不到处理或者得不到正确的处理，程序员必须谨慎使用并确保脚本加载完成后其中的函数才被页面事件调用，否则浏览器报错。

综上所述，引入外部 JavaScript 脚本文件的方法是效果与风险并存，开发者应权衡优缺点以决定是将脚本代码嵌入到目标 HTML 文档中还是通过引用外部脚本文件的方式来实现相同的功能。

> 注意：一般来讲，将实现通用功能的 JavaScript 脚本代码作为外部脚本文件引用，而实现特有功能的 JavaScript 代码则直接嵌入到 HTML 文档中的<head>与</head>标记对之间提前载入以及时、正确响应页面事件。

下面介绍一种短小高效的脚本代码嵌入方式：伪 URL 引入。

3. 通过 JavaScript 伪 URL 引入

在多数支持 JavaScript 脚本的浏览器中，可以通过 JavaScript 伪 URL 地址调用语句来引入 JavaScript 脚本代码。伪 URL 地址的一般格式如下：

```
JavaScript:alert("Hello World!")
```

一般以“javascript:”开始，后面紧跟要执行的操作。下面的代码演示如何使用伪 URL 地址来引入 JavaScript 代码：

```
//源程序 1.3
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
  <title>Sample Page!</title>
</head>
<body>
<br>
<center>
<p>伪 URL 地址引入 JavaScript 脚本代码实例：</p>
<form name="MyForm">
  <input type=text name="MyText" value="鼠标点击"
        onclick="javascript:alert('鼠标已点击文本框!')">
</form>
</center>
</body>
</html>
```

鼠标点击文本框，弹出警示框如图 1.7 所示。

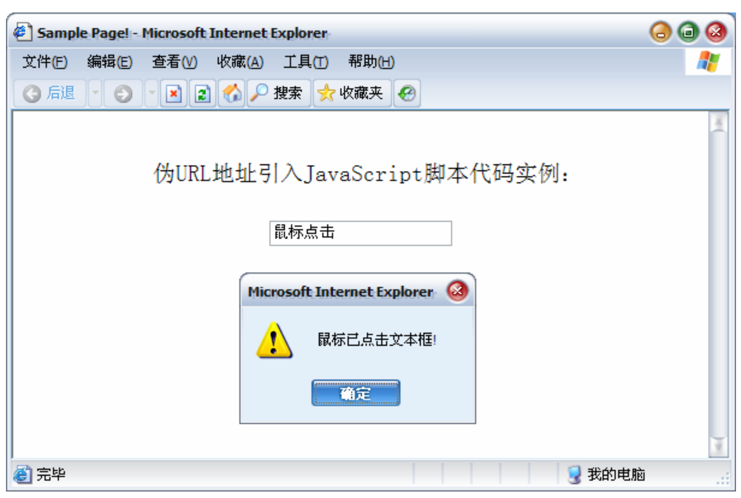

图 1.7 伪 URL 地址引入 JavaScript 脚本代码实例

伪 URL 地址可用于文档中任何地方，并触发任意数量的 JavaScript 函数或对象固有的方法。由于这种方式代码短小精悍，同时效果颇佳，在表单数据合法性验证譬如某个字段是否符合日期格式等方面应用非常广泛。

4. 通过 HTML 文档事件处理程序引入

在开发 Web 应用程序的过程中，开发者可以给 HTML 文档中设定不同的事件处理器，通常是设置某 HTML 元素的属性来引用一个脚本（可以是一个简单的动作或者函数），属性一般以 on 开头，如鼠标移动 onmousemove( )等。

下面的程序演示如何使用 JavaScript 脚本对按钮单击事件进行响应：

```
//源程序 1.4
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
```

```
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="javascript" type="text/javascript">
function ClickMe()
{
   alert("鼠标已单击按钮");
}
</script>
</head>
<body>
<br>
<center>
<p>通过文档事件处理程序引入 JavaScript 脚本代码实例：</p>
<form name="MyForm">
   <input type=button name="MyButton" value="鼠标单击" onclick="ClickMe()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“鼠标单击”按钮，出现如图 1.8 所示的警告框。

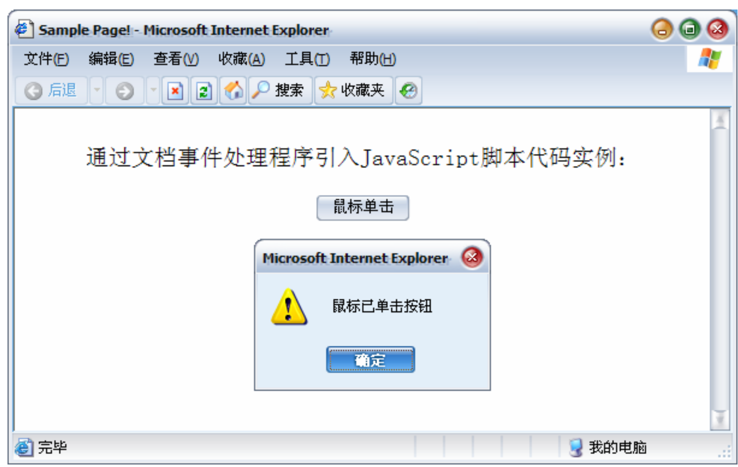

图 1.8 通过文档事件处理程序引入 JavaScript 脚本实例

遗憾的是，许多版本较低的浏览器，不能够从众多 HTML 标记中识别出事件处理器，即使支持，支持的程度也不同，对事件的处理方法差别也很大。但是大部分浏览器都能理解 HTML 标记的核心事件，如 onclick、ondbclick、onkeydown、onkeypress、onkeyup、onmousedown、onmousemove、onmouseover、onmouseout 等鼠标和键盘触发事件。

知道了如何引入 JavaScript 脚本代码，下面介绍在 HTML 中嵌入 JavaScript 脚本代码的位置。

1.2.4 嵌入 JavaScript 脚本代码的位置

JavaScript 脚本代码可放在 HTML 文档任何需要的位置。一般来说，可以在<head>与</head>标记对、<body>与</body>标记对之间按需要放置 JavaScript 脚本代码。

1. <head>与</head>标记对之间放置

放置在<head>与</head>标记对之间的 JavaScript 脚本代码一般用于提前载入以响应用户的动作，一般不影响 HTML 文档的浏览器显示内容。如下是其基本文档结构：

```
//源程序 1.5
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>文档标题</title>
<script language="javascript" type="text/javascript">
  //脚本语句…
</script>
</head>
<body>
</body>
</html>
```

2. <body>与</body>标记对之间放置

如果需要在页面载入时运行 JavaScript 脚本生成网页内容，应将脚本代码放置在<body>与</body>标记对之间，可根据需要编写多个独立的脚本代码段并与 HTML 代码结合在一起。如下是其基本文档结构：

```
//源程序 1.6
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>文档标题</title>
</head>
<body>
<script language="javascript" type="text/javascript">
  //脚本语句…
</script>
  //HTML 语句
<script language="javascript" type="text/javascript">
  //脚本语句…
</script>
</body>
</html>
```

3. 在两个标记对之间混合放置

如果既需要提前载入脚本代码以响应用户的事件，又需要在页面载入时使用脚本生成页面内容，可以综合以上两种方式。如下是其基本文档结构：

```
//源程序 1.7
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>文档标题</title>
<script language="javascript" type="text/javascript">
```

```
  //脚本语句…
</script>
</head>
<body>
<script language="javascript" type="text/javascript">
  //脚本语句…
</script>
</body>
</html>
```

在 HTML 文档中何种位置嵌入 JavaScript 脚本代码应由其实际功能需求来决定。嵌入脚本的 HTML 文档编辑完成，下一步选择合适的浏览器。

1.2.5 选择合适的浏览器

JavaScript 脚本在客户端由浏览器解释执行并将结果更新目标页面，由于各浏览器厂商对 JavaScript 版本的支持不尽相同，浏览器的版本也对 JavaScript 脚本的支持有很大影响，所以编写代码时一定要考虑合适的浏览器之间的兼容性，重点在于编写符合 JavaScript 标准的代码以适应目标浏览器。下面是浏览器版本与其支持的 JavaScript 版本之间的关系，如表 1.2 所示：

表 1.2 浏览器版本与其支持的JavaScript版本之间的关系表

浏览器版本	JavaScript版本
Netscape Navigator 2.x	1.0
Netscape Navigator 3.x	1.1
Netscape Navigator 4.0-4.05	1.2
Netscape Navigator 4.06-4.08，4.5x，4.6x，4.7x	1.3
Netscape Navigator 6.0+	1.5
Internet Explorer 3.0	JScript 1.0
Internet Explorer 4.0	JScript 3.0
Internet Explorer 5.0	JScript 5.0
Internet Explorer 5.5	JScript 5.5
Internet Explorer 6.0+	JScript 5.6

在各大浏览器厂商中，基于 Mozilla 的浏览器（包括 Netscape Navigator 6+）对 JavaScript 标准的支持最好，其实现了 JavaScript 1.5 且只修改了其中很少的语言特性。

即使 ECMA 出台 ECMA-262、ECMA-290、ECMA-357 等标准意在消除 JavaScript 各个不同版本之间的差异性，JavaScript 的应用依然受到很大的挑战。本书将在后面的章节中结合 DOM（文档结构模型）针对 JavaScript 各种规范版本之间差异进行重点讨论以真正解决脚本代码的浏览器兼容问题。

可通过如下的代码检查当前浏览器的版本信息：

```
//源程序 1.8
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page！</title>
<script language="javascript" type="text/javascript">
function PrintVersion()
{
  var msg="";
```

```
   msg+="浏览器名称:"+navigator.appName+"\n";
   msg+="浏览器版本:"+navigator.appVersion+"\n";
   msg+="浏览器代码:"+navigator.appCodeName+"\n";
   alert(msg);
}
</script>
</head>
<body>
<br>
<center>
<p>鼠标单击按钮显示当前浏览器的版本信息</p>
<form name="MyForm">
   <input type=button name="MyButton" value="鼠标单击" onclick="PrintVersion()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“鼠标单击”按钮，弹出警告框如图 1.9 所示。

图 1.9 获取当前浏览器的版本信息

在确定浏览器的版本信息后，可以根据浏览器类型编写有针对性的脚本，同时可在其源程序中加入针对不同浏览器版本的脚本代码，根据浏览器的类型返回相应结果给浏览器，从而克服客户端浏览器对 JavaScript 脚本支持程度不同的问题。

1.2.6 处理不支持 JavaScript 脚本的情况

客户端浏览器不支持当前 JavaScript 脚本存在如下三种可能：

- 客户端浏览器不支持任何 JavaScript 脚本；
- 客户端浏览器支持的 JavaScript 脚本版本与该脚本代码使用的版本所支持的对象、属性或方法不同；
- 客户端为了安全起见，已经将浏览器对 JavaScript 脚本的支持设置为禁止。

以上三种情况总结起来，就是浏览器对当前脚本不能解释出正确的结果，在编写脚本代码时如不进行相关处理，用户使用该浏览器浏览带有该脚本的文档时将出现警告框。可以通过以下两种方法解决：

1. 使用<!--和-->标记对直接屏蔽法

该方法使用<!--和-->标记对将 JavaScript 代码进行封装，告诉浏览器如果它不支持该脚本就直接跳过，如果支持脚本代码则自动跳过该标记对，达到如果浏览器不支持脚本代码则将其隐藏的目的。如下代码结构：

```
<script language="javascript" type="text/javascript">
<!--
  //此处添加脚本代码
-->
</script>
```

注意：上述方法并没有实现 JavaScript 脚本代码的真正隐藏，因为浏览器同样下载了该脚本，并将其作为源代码使用，只是在解释的时候忽略<!--和-->标记对之间的代码。

2. 使用<noscript>和</noscript>标记对给出提示信息

该方法在浏览器不支持该脚本代码或者浏览器对 JavaScript 脚本的支持已设置为禁止的情况下，忽略<script>和</script>标记对之间脚本代码，返回<noscript>和</noscript>标记对中预设的页面提示信息；如果支持该脚本代码则解释执行<script>和</script>标记对之间脚本代码，而忽略<noscript>和</noscript>标记对之间预设的页面提示信息。这种方法较之第一种方法更人性化。如下代码结构：

```
<script language="javascript" type="text/javascript">
  //脚本代码
</script>
<noscript>
  //提示信息
</noscript>
```

目前，客户端浏览器版本很少有不支持 JavaScript 脚本的情况，但其禁用 JavaScript 脚本的情况很常见，在编写代码的应充分考虑不支持 JavaScript 脚本的情况并采取相应的代码编写策略。

1.3 JavaScript 的实现基础

前面已经描述过，ECMAScript 是 JavaScript 脚本的基石，但并不是使用 JavaScript 脚本开发过程中应唯一特别值得关注的部分。实际上，一个完整的 JavaScript 脚本实现应包含如下三部分：

- ECMAScript 核心：为不同的宿主环境提供核心的脚本能力；
- DOM（文档对象模型）：规定了访问 HTML 和 XML 的应用程序接口；
- BOM（浏览器对象模型）：提供了独立于内容而在浏览器窗口之间进行交互的对象和方法。

下面分别介绍这三个部分：

1.3.1 ECMAScript

ECMAScript 规定了能适应于各种宿主环境的脚本核心语法规则，关于 ECMAScript 语言，ECMA-262 标准描述如下：

“ECMAScript 可以为不同种类的宿主环境提供核心的脚本编程能力，因此核心的脚本

语言是与任何特定的宿主环境分开进行规定的……”

ECMAScript 并不附属于任何浏览器，事实上，Web 浏览器只是其中一种宿主环境，但并不唯一。在其发展史上有很多宿主环境，如 Microsoft 的 WSH、Micromedia 的 ActionScript、Nombas 的 ScriptBase 和 Yahool!的 Widget 引擎等都可以容纳 ECMAScript 实现。

ECMAScript 仅仅是个描述，定义了脚本语言所有的对象、属性和方法，其主要描述了如下内容：

- 语法
- 数据类型
- 关键字
- 保留字
- 运算符
- 对象
- 语句

支持 ECMA 标准的浏览器都提供自己的 ECMAScript 脚本接口，并按照需要扩展其内容如对象、属性和方法等。

ECMAScript 标准定义了 JavaScript 脚本中最为核心的内容，是 JavaScript 脚本的骨架，有了骨架，就可以在其上进行扩展，典型的扩展如 DOM（文档对象模型）和 BOM（浏览器对象模型）等。

1.3.2 DOM

DOM 定义了 JavaScript 可以进行操作的文档的各个功能部件的接口，提供访问文档各个功能部件（如 document、form、textarea 等）的途径以及操作方法。

在浏览器载入文档（HTML 或 XML）时，根据其支持的 DOM 规范级别将整个文档规划成由节点层级构成的节点树，文档中每个部分都是一个节点，然后依据节点树层级同时根据目标节点的某个属性搜索到目标节点后，调用节点的相关处理方法进行处理，完成定位到处理的整个过程。先看下面简单的 HTML 代码：

```
//源程序 1.9
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
  <title> First Page!</title>
</head>
<body>
<h1>Test!</h1>
<!--NOTE!-->
<p>Welcome to<em> DOM </em>World! </p>
<ul>
  <li>Newer</li>
</ul>
</body>
</html>
```

浏览器载入该文档后，根据 DOM 中定义的结构，将其以图 1.10 所示的节点树形式表示出来（灰色表示本本节点）。

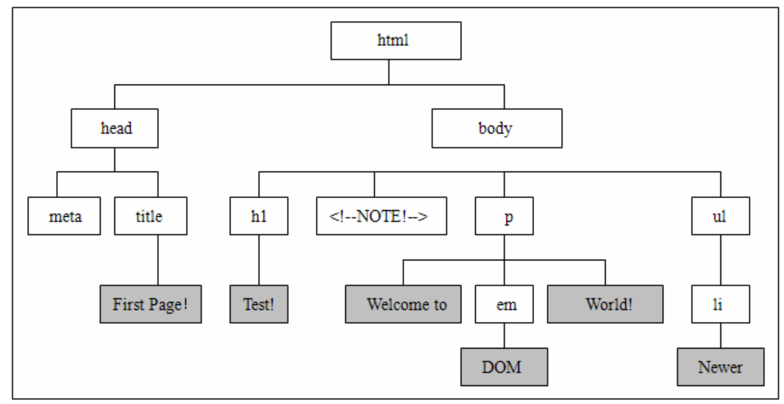

图 1.10 DOM 载入实例中文档后形成的节点树层级

关于 DOM 的具体内容在本书“文档对象模型（DOM）”一章将作详细的介绍。

1.3.3 BOM

BOM 定义了 JavaScript 可以进行操作的浏览器的各个功能部件的接口，提供访问文档各个功能部件（如窗口本身、屏幕功能部件、浏览历史记录等）的途径以及操作方法。遗憾的是，BOM 只是 JavaScript 脚本实现的一部分，没有任何相关的标准，每种浏览器都有自己的 BOM 实现，这可以说是 BOM 的软肋所在。

通常情况下浏览器特定的 JavaScript 扩展都被看作 BOM 的一部分，主要包括：

- 关闭、移动浏览器及调整浏览器窗口大小；
- 弹出新的浏览器窗口；
- 提供浏览器详细信息的定位对象；
- 提供载入到浏览器窗口的文档详细信息的定位对象；
- 提供用户屏幕分辨率详细信息的屏幕对象；
- 提供对 cookie 的支持；
- 加入 ActiveXObject 类扩展 BOM，通过 JavaScript 实例化 ActiveX 对象。

BOM 有一些事实上的标准，如窗口对象、导航对象等，但每种浏览器都为这些对象定义或扩展了属性及方法。

在后面的章节中，将详细介绍 BOM 模型中的相关对象。

1.4 客户端脚本与服务器端脚本

最早实现动态网页的技术是 CGI（Common Gateway Interface，通用网关接口）技术，，它可根据用户的 HTTP 请求数据动态从 Web 服务器返回请求的页面。客户与服务器端的一次握手过程如图 1.11 所示。

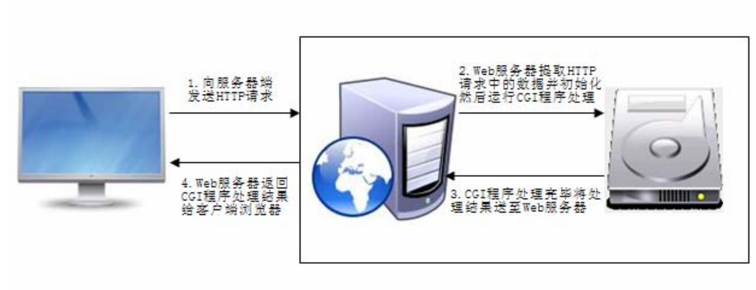

图 1.11 CGI 动态网页技术中的页面请求处理过程

当用户从 Web 页面提交 HTML 请求数据后，Web 浏览器发送用户的请求到 Web 服务器上，服务器运行 CGI 程序，后者提取 HTTP 请求数据中的内容初始化设置，同时交互服务器端的数据库，然后将运行结果返回 Web 服务器，Web 服务器根据用户请求的地址将结果返回该地址的浏览器。从整个过程来讲，CGI 程序运行在服务器端，同时需要与数据库交换数据，这需要开发者拥有相当的技巧，同时拥有服务器端网站开发工具，程序的编写、调试和维护过程十分复杂。

同时，由于整个处理过程全部在服务器端处理，无疑是服务器处理能力的一大硬伤，而且客户端页面的反应速度不容乐观。基于此，客户端脚本语言应运而生，它可直接嵌入到 HTML 页面中，及时响应用户的事件，大大提高页面反应速度。

脚本分为客户端脚本和服务器端脚本，其主要区别如表 1.3 所示：

表 1.3 客户端脚本与服务器端脚本的区别

脚本类型	运行环境	优缺点	主要语言
客户端 脚本	客户端 浏览器	当用户通过客户端浏览器发送HTTP请求时，Web服务器将HTML文档部分和脚本部分返回客户端浏览器，在客户端浏览器中解释执行并及时更新页面，脚本处理工作全部在客户端浏览器完成，减轻服务器负荷，同时增加页面的反应速度，但浏览器差异性导致的页面差异问题不容忽视	JavaScript、 JScript、 VBScript等
服务器端 脚本	Web 服务器	当用户通过客户端浏览器发送HTTP请求时，Web服务器运行脚本，并将运行结果与Web页面的HTML部分结合返回至客户端浏览器，脚本处理工作全部在服务器端完成，增加了服务器的负荷，同时客户端反应速度慢，但减少了由于浏览器差异带来的运行结果差异，提高页面的稳定性。	PHP、 JSP、 ASP、 Perl、 LiveWire等

注意：有关 HTTP 请求、TCP\IP 协议、Web 服务器、CGI 技术等请参阅相关文档。

客户端脚本与服务器端脚本各有其优缺点，在不同需求层次上得到了广泛的应用。JavaScript 作为一种客户端脚本，在页面反应速度、减轻服务器负荷等方面效果非常明显，但由于浏览器对其支持的程度不同导致的页面差异性问题也不容小觑。

下面几节来阐明几个容易混淆的概念，如 JavaScript 与 JScript、VBScript 背景的区别、JavaScript 与 Java、Java applet 概念的不同等。

1.5 JavaScript 与 JScript、 VBScript

JavaScript 由 Netscape 公司和 Sun 公司联合开发，并在其 Netscape Navigator 2 上首先实现了该语言的 JavaScript 1.0 版，主要应用于客户端 Web 应用程序开发，由于及时推出了相关标准，以及语言本身使用简单、实现功能强大的优点，受到 Web 应用程序开发者的追捧，

并陆续推出其 1.1，1.2，1.3，1.4 和 1.5 版。

为了应对 JavaScript 脚本强劲的发展势头，Microsoft 在其 Internet Explorer 3 里推出了 JavaScript 1.0 的克隆版本 JScript 1.0 来抢占客户端脚本市场。在后来的版本中 JScript 逐渐被 WSH 和 ASP（Active Server Pages：活动服务器页面，以下简称 ASP）所支持，并实现了动态脚本技术。JScript 的最新版本是基于尚未定稿的 ECMAScript4.0 版规范的 JScript .NET，其可在微软的.Net 环境下编译，然后生成.net 框架内的应用程序。其保持了与 JScript 以前版本的完全向后兼容性，同时包含了强大的新功能并提供了对公共语言运行库和.NET Framework 的访问方法。

VBScript（Microsoft Visual Basic Scripting Edition）是程序开发语言 Visual Basic 家族的最新成员，它将灵活的脚本应用于更广泛的领域，包括 Microsoft Internet Explorer 中的 Web 客户端脚本和 Microsoft Internet Information Server 中的 Web 服务器端脚本。VBScript 也是Microsoft推出的产品，开始主要定位于客户端脚本，由于动态页面技术的快速发展，VBScript走向服务器端，与ASP、IIS（Internet Information Server：互联网信息服务）紧密结合，有力促进动态页面技术的发展。

同时，Microsoft 的 JScript 和 VBScript 脚本应用在服务器端，执行相应的管理权限，同时 Microsft 提供其访问系统组建的 API，使之与系统紧密结合，如访问本地数据库，并将结果返回客户端浏览器等。

这三种脚本语言各有各的产生背景，同时其侧重点也不大相同。

1.6 JavaScript 与 Java、Java applet

JavaScript 和 Java 虽然名字都带有 Java，但它们是两种不同的语言，也可以说是两种互不相干的语言：前者是一种基于对象的脚本语言，可以嵌在网页代码里实现交互及控制功能，而后者是一种面向对象的编程语言，可用在桌面应用程序、Internet 服务器、中间件、嵌入式设备以及其他众多环境。其主要区别如下：

- 开发公司不同：JavaScript 是 Netscape 公司的产品，其目的是为了扩展 Netscape Navigator 功能，而开发的一种可以嵌入 Web 页面中的基于对象和事件驱动的解释性语言；Java 是 Sun 公司推出的新一代面向对象的程序设计语言，特别适合于 Internet 应用程序开发。
- 语言类型不同：JavaScript 是基于对象和事件驱动的脚本编程语言，本身提供了非常丰富的内部对象供设计人员使用；Java 是面向对象的编程语言，即使是开发简单的程序，也必须设计对象。
- 执行方式不同：JavaScript 是一种解释性编程语言，其源代码在发往客户端执行之前不需经过编译，而是将文本格式的字符代码发送给客户，由浏览器解释执行；Java 的源代码在传递到客户端执行之前，必须经过编译，因而客户端上必须具有相应平台上的仿真器或解释器，它可以通过编译器或解释器实现独立于某个特定的平台编译代码的束缚。
- 代码格式不同：JavaScript 的代码是一种文本字符格式，可以直接嵌入 HTML 文档中，并且可动态装载；Java 是一种与 HTML 无关的格式，必须将其通过专门编译器编译为 Java applet，其代码以字节代码的形式保存在独立的文档中，然后在 HTML 中通过引用外部插件的方式进行装载，
- 变量类型不同：JavaScript 采用弱类型变量，即变量在使用前不需作特别声明，而是在浏览器解释运行时该代码时才检查其数据类型；Java 采用强类型变量，即所有

变量在通过编译器编译之前必须作专门声明，否则报错。

- 嵌入方式不同：JavaScript 使用 <script>和</script>标记对来标识其脚本代码并将其嵌入到 HTML 文档中；Java 程序通过专门编译器编译后保存为单独的 Java applet 文件，并通过使用<applet> ... </applet>标记对来标识该插件。
- 联编方式不同：JavaScript 采用动态联编，即其对象引用在浏览器解释运行时进行检查，如不经编译则就无法实现对象引用的检查；Java 采用静态联编，即 Java 的对象引用必须在编译时进行，以使编译器能够实现强类型检查。

经过以上几个方面的比较，读者应该能清醒认识 JavaScript 和 Java 是没有任何联系的两门语言。下面讨论 Java applet。

Java applet 是用 Java 语言编写的、有特定用途的应用程序，其直接嵌入到 HTML 页面中，由支持 Java 的浏览器解释执行并发挥其特定功能，大大提高 Web 页面的交互和动态执行能力，包含 applet 应用程序的页面被称为 Java-powered 页。

当用户访问这样的网页时，如果客户端浏览器支持 Java 且没有将 Java 支持选项设置为禁止，则 applet 被下载到用户的计算机上执行，且执行速度不受网络带宽的限制，用户可以更好地欣赏网页上 applet 产生的各种效果。

与其他应用程序不同，applet 应用程序必须通过<applet>和</applet>标记对将自己内嵌到 HTML 页面中，当支持 Java 的客户端浏览器遇到该标记对时，立即下载该 applet 并在本地计算机上执行。执行的过程中它可从目标页面中获得相应的参数，并产生相应的功能，与 Web 页面进行交互，实现页面的动态效果。

在 HTML 页面中嵌入 applet，至少需获得该 applet 的以下信息：

- 字节码文件名：编译后的 Java 文件，以.class 为后缀；
- 字节码文件的地址：相对地址和绝对地址均可；
- 显示参数设定：一些需要设定的参数如 width、height 等。

嵌入 applet 应用程序使页面更加富有生气，增加页面的交互能力，改进页面的定态效果，同时，嵌入 applet 应用程序并不影响 HTML 页面中的其他元素。在本书将有专门的章节来讲述网页中 Java applet 的应用。

1.7 本章小结

本章主要介绍了 JavaScript 脚本的发展历史、使用特点、功能和未来，同时带领读者开始编写自己的“Hello World!”程序，兼顾 JavaScript 及浏览器的版本差异性提出相应的编程策略；讲述了 JavaScript 脚本语言的实现基础，阐明了几个比较易混淆的脚本术语，如 JavaScript 与 JScript、VBScript 背景的区别，以及 JavaScript 与 Java、Java applet 概念的异同点等，力图给读者一个比较全面、直观的印象。

第 2 章　JavaScript 语言基础

JavaScript 脚本语言作为一门功能强大、使用范围较广的程序语言，其语言基础包括数据类型、变量、运算符、函数以及核心语句等内容。本章主要介绍 JavaScript 脚本语言的基础知识，带领读者初步领会 JavaScript 脚本语言的精妙之处，并为后续章节的深入学习打下坚实的基础。

本章涉及到对象的相关知识，在本书后续章节将对其进行适当的分类和详细的论述，如读者理解有困难，可自行跳过，待学习了对象的基本概念和相关知识后再进行深入理解。

2.1 编程准备

在正式介绍 Javascript 脚本语言之前，先介绍使用 JavaScript 脚本进行编程需要首先了解的知识，包括编程术语、大小写敏感性、空白字符以及分号等内容，以及脚本编程过程中需遵守的一些约定，以编写合法的 JavaScript 脚本程序。

2.1.1 编程术语

首先我们来学习一下 Javascript 程序语言的基本术语，这些术语将贯穿 JavaScript 脚本编程的每个阶段，汇总如表 2.1 所示：

表 2.1　Javascript脚本编程基本术语

项目	简要说明	举例
Token(语言符号)	Javascrip脚本语言中最小的词汇单元，是一个字符序列	6，“I am a boy”，所有的标识符和关键字
Literal(常量)	拥有固定值的表达式	6，“I am a boy”，[1，2，3]
Identifer(标识符)	变量、函数、对象等的名称	num，alert，yourSex
Operator(运算符)	执行赋值、数学运算、比较等的符号	=，+，%，>
Expression(表达式)	标识符、运算符等组合起来的一个语句，返回该语句执行特定运算后的值	x+1，(num+1)/5

Statement(语句)	达到某个特定目的的强制性命令，脚本程序由多个语句构成	var num=5；
		function sum(x,y){
		　result=x+y;
		　return(result);
		}

Keyword(关键字)	作为脚本语言一部分的字符串，不能用作标识符使用	if，for，var，function
Reserved(保留字)	有可能作为脚本语言一部分的字符串，但并不严格限制其不能作为标识符	const，short，long

2.1.2 脚本执行顺序

JavaScript 脚本解释器将按照程序代码出现的顺序来解释程序语句，因此可以将函数定义和变量声明放在<head>和</head>之间，此时与函数体相关的操作不会被立即执行。

2.1.3 大小写敏感

JavaScript 脚本程序对大小写敏感，相同的字母，大小写不同，代表的意义也不同，如变量名 name、Name 和 NAME 代表三个不同的变量名。在 JavaScript 脚本程序中，变量名、函数名、运算符、关键字、对象属性等都是对大小写敏感的。同时，所有的关键字、内建函数以及对象属性等的大小写都是固定的，甚至混合大小写，因此在编写 JavaScript 脚本程序时，要确保输入正确，否则不能达到编写程序的目的。

2.1.4 空白字符

空白字符包括空格、制表符和换行符等，在编写脚本代码时占据一定的空间，但脚本被浏览器解释执行时无任何作用。脚本程序员经常使用空格作为空白字符，JavaScript 脚本解释器是忽略任何多余空格的。考察如下赋值语句：

```
s   =   s   +   5  ;
```

以及代码：

```
s=s+5;
```

上述代码的运行结果相同，浏览器解释执行第一个赋值语句时忽略了其中的空格。值得注意的是，浏览器解释执行脚本代码时，并非语句中所有的空格均被忽略掉。考察如下变量声明：

```
x=typeof y;
x=typeofy;
```

上面这两行代码代表的意义是不同的。第一行是将运算符 typeof 作用在变量 y 上，并将结果赋值给变量 x；而第二行是直接将变量 typeofy 的值赋给了 x，两行代码的意义完全不同。

在编写 JavaScript 脚本代码时经常使用一些多余的空格来增强脚本代码的可读性，并有助于专业的 JavaScript 脚本程序员（或者非专业人员）查看代码结构，同时有利于脚本代码的日后维护。

> 注意：在字符串中，空格不被忽略，而作为字符串的一部分显示出来，在编写 JavaScript 脚本代码时，经常需添加适当的空格使脚本代码层次明晰，方便相关人员查看和维护。

2.1.5 分号

在编写脚本语句时，用分号作为当前语句的结束符，例如：

```
var x=25;
var y=16;
var z=x+y;
```

当然，也可将多个语句写在同一行中，例如：

```
var x=25;var y=16;var z=x+y;
```

值得注意的是，为养成良好的编程习惯，尽量不要将多个语句写在一行中，避免降低脚本代码的可读性。

另外，语句分行后，作为语句结束符的分号可省略。例如可改写上述语句如下：

```
var x=25
var y=16
var z=x+y
```

代码运行结果相同，如将多个语句写在同一行中，则语句之间的分号不可省略。

2.1.6 块

在定义函数时，使用大括号“{}”将函数体封装起来，例如：

```
function muti(m,n)
{
  var result=m*n;
  return result;
}
```

在使用循环语句时，使用大括号“{}”将循环体封装起来，例如：

```
if(age<18)
{
   alert("对不起，您的年龄小于 18 岁，您无权浏览此网页");
}
```

从本质上讲，使用大括号“{}”将某段代码封装起来后，构成“块”的概念，JavaScript 脚本代码中的块，即为实现特定功能的多句（也可为空或一句）脚本代码构成的整体。

2.2 数值类型

2.2.1 整型和浮点数值

JavaScript 允许使用整数类型和浮点类型两种数值，其中整数类型包含正整数、0 和负整数；而浮点数则可以是包含小数点的实数，也可以是用科学计数法表示的实数，例如：

```
var age = 32;           //整数型
var num = 32.18;        //包含小数点的浮点型
var num = 3.7E-2;       //科学计数法表示的浮点型
```

2.2.2 八进制和十六进制

在整数类型的数值中，数制可使用十进制、八进制以及十六进制，例如：

```
var age = 32;               //十进制
var num = 010;              //八进制
var num = C33;              //十六进制
```

2.3 变量

几乎任何一种程序语言都会引入变量（variable），包括变量标识符、变量申明和变量作用域等内容。JavaScript 脚本语言中也将涉及到变量，其主要作用是存取数据以及提供存放信息的容器。在实际脚本开发过程中，变量为开发者与脚本程序交互的主要工具。下面分别介绍变量标识符、变量申明和变量作用域等内容。

2.3.1 变量标识符

与 C++、Java 等高级程序语言使用多个变量标识符不同，JavaScript 脚本语言使用关键字 var 作为其唯一的变量标识符，其用法为在关键字 var 后面加上变量名。例如：

```
var age;
var MyData;
```

2.3.2 变量申明

在 JavaScript 脚本语言中，声明变量的过程相当简单，例如通过下面的代码声明名为 age 的变量：

```
var age;
```

JavaScript 脚本语言允许开发者不首先声明变量就直接使用，而在变量赋值时自动申明该变量。一般来说，为培养良好的编程习惯，同时为了使程序结构更加清晰易懂，建议在使用变量前对变量进行申明。

变量赋值和变量声明可以同时进行，例如下面的代码声明名为 age 的变量，同时给该变量赋初值 25：

```
var age = 25;
```

当然，可在一句 JavaScript 脚本代码中同时声明两个以上的变量，例如：

```
var age , name;
```

同时初始化两个以上的变量也是允许的，例如：

```
var age = 35 , name = “tom”;
```

在编写 JavaScript 脚本代码时，养成良好的变量命名习惯相当重要。规范的变量命名，不仅有助于脚本代码的输入和阅读，也有助于脚本编程错误的排除。一般情况下，应尽量使用单词组合来描述变量的含义，并可在单词间添加下划线，或者第一个单词头字母小写而后续单词首字母大写。

> 注意：JavaScript 脚本语言中变量名的命名需遵循一定的规则，允许包含字母、数字、下划线和美元符号，而空格和标点符号都是不允许出现在变量名中，同时不允许出现中文变量名，且大小写敏感。

2.3.3 变量作用域

要讨论变量的作用域，首先要清楚全局变量和局部变量的联系和区别：

- 全局变量：可以在脚本中的任何位置被调用，全局变量的作用域是当前文档中整个脚本区域。
- 局部变量：只能在此变量声明语句所属的函数内部使用，局部变量的作用域仅为该函数体。

声明变量时，要根据编程的目的决定将变量声明为全局变量还是局部变量。一般而言，保存全局信息（如表格的原始大小、下拉框包含选项对应的字符串数组等）的变量需声明为全局变量，而保存临时信息（如待输出的格式字符串、数学运算中间变量等）的变量则声明为局部变量。

考察如下代码：

```
//源程序 2.1
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
```

```
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var total=100;
function add(num)
{
  var total=2*num;
  alert("\n 局部变量 ：\n\ntotal ="+total+"\n");
  return true;
}
-->
</script>
</head>
<body onload="add(total)">
<center>
<form>
  <input type=button value="局部和全局变量测试"
        onclick="javascript:alert('\n 全局变量 ：\n\ntotal ='+total+'\n');">
</form>
</center>
</body>
</html>
```

浏览器载入上述代码后，弹出警告框显示局部变量 total 的值，其结果如图 2.1 所示。

图 2.1 局部变量

在上述警告框中单击“确定”后，关闭该警告框。单击“局部和全局变量测试”按钮，弹出警告框显示全局变量 total 的值，如图 2.2 所示。

图 2.2 全局变量

代码载入后，add(num)函数响应 body 元素对象的 onload 事件处理程序，输出其函数体中定义的局部变量 total 的值（整数 200）。单击“局部和全局变量测试”按钮，触发其 onclick 事件处理程序运行与其关联的 JavaScript 代码输出全局变量 total 的值（整数 100）：

```
javascript:alert('\n 全局变量 : \n\ntotal ='+total+'\n');
```

由上述结果可以看出，全局变量可作用在当前文档的 JavaScript 脚本区域，而局部变量仅存在于其所属的函数体内。实际应用中，应根据全局变量和局部变量的作用范围恰当定义变量，并尽量避免全局变量与局部变量同名，否则容易出现不易发现的变量调用错误。同时注意应对代码中引入的任何变量均进行声明。

2.4 弱类型

JavaScript 脚本语言像其他程序语言一样，其变量都有数据类型，具体数据类型将在下一节中介绍。高级程序语言如 C++、Java 等为强类型语言，与此不同的是，JavaScript 脚本语言是弱类型语言，在变量声明时不需显式地指定其数据类型，变量的数据类型将根据变量的具体内容推导出来，且根据变量内容的改变而自动更改，而强类型语在变量声明时必须显式地指定其数据类型。

变量声明时不需显式指定其数据类型既是 JavaScript 脚本语言的优点也是缺点，优点是编写脚本代码时不需要指明数据类型，使变量声明过程简单明了；缺点就是有可能造成因微妙的拼写不当而引起致命的错误。

考察如下代码：

```
//源程序 2.2
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//弱类型测试函数
function Test()
{
  var msg="\n 弱类型语言测试 : \n\n";
  msg+="'600''*5 = "+('600'*6)+"\n";
  msg+="'600''-5 = "+('600'-5)+"\n";
  msg+="'600''/5 = "+('600'/5)+"\n";
  msg+="'600''+5 = "+('600'+5)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value="弱类型测试" onclick="Test()">
</form>
</center>
```

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

程序运行后，在原始页面单击“弱类型测试”按钮，弹出警告框如图 2.3 所示。

图 2.3　弱类型语言测试

由上图中前三个表达式运算结果可知，JavaScript 脚本在解释执行时自动将字符型数据转换为数值型数据，而最后一个结果由于加号“+”的特殊性导致运算结果不同，是将数值型数据转换为字符型数据。运算符“+”有两个作用：

- 作为数学运算的加和运算符
- 作为字符型数据的连接符

由于加号“+”作为后者使用时优先级较高，故实例中表达式“'600'+5”的结果为字符串“6005”，而不是整数 605。

2.5 基本数据类型

在实现预定功能的程序代码中，一般需定义变量来存储数据（作为初始值、中间值、最终值或函数参数等）。变量包含多种类型，JavaScript 脚本语言支持的基本数据类型包括 Number 型、String 型、Boolean 型、Undefined 型、Null 型和 Function 型，分别对应于不同的存储空间，汇总如表 2.2 所示：

表 2.2　六种基本数据类型

类型	举例	简要说明
Number	45 , -34 , 32.13 , 3.7E-2	数值型数据
String	"name" , 'Tom'	字符型数据，需加双引号或单引号
Boolean	true , flase	布尔型数据，不加引号，表示逻辑真或假
Undefined		
Null	null	表示空值
Function		表示函数

2.5.1 Number 型

Number 型数据即为数值型数据，包括整数型和浮点型，整数型数制可以使用十进制、八进制以及十六进制标识，而浮点型为包含小数点的实数，且可用科学计数法来表示。一般来说，Number 型数据为不在括号内的数字，例如：

```
var myDataA=8;
var myDataB=6.3;
```

上述代码分别定义值为整数 8 的 Number 型变量 myDataA 和值为浮点数 6.3 的 Number 型变量 myDataB。

2.5.2 String 型

String 型数据表示字符型数据。JavaScript 不区分单个字符和字符串，任何字符或字符串都可以用双引号或单引号引起来。例如下列语句中定义的 String 型变量 nameA 和 nameB 包含相同的内容：

```
var nameA = "Tom";
var nameB = 'Tom';
```

如果字符串本身含有双引号，则应使用单引号将字符串括起来；若字符串本身含有单引号，则应使用双引号将字符串引起来。一般来说，在编写脚本过程中，双引号或单引号的选择在整个 JavaScript 脚本代码中应尽量保持一致，以养成好的编程习惯。

2.5.3 Boolean 型

Boolean 型数据表示的是布尔型数据，取值为 ture 或 false，分别表示逻辑真和假，且任何时刻都只能使用两种状态中的一种，不能同时出现。例如下列语句分别定义 Boolean 变量 bChooseA 和 bChooseB，并分别赋予初值 true 和 false：

```
var bChooseA = true;
var bChooseB = false;
```

值得注意的是，Boolean 型变量赋值时，不能在 true 或 false 外面加引号，例如：

```
var happyA = true;
var happyB = “true”;
```

上述语句分别定义初始值为 true 的 Boolean 型变量 happyA 和初始值为字符串“true”的 String 型变量 happyB。

2.5.4 Undefined 型

Undefined 型即为未定义类型，用于不存在或者没有被赋初始值的变量或对象的属性，如下列语句定义变量 name 为 Undefined 型：

```
var name;
```

定义 Undefined 型变量后，可在后续的脚本代码中对其进行赋值操作，从而自动获得由其值决定的数据类型。

2.5.5 Null 型

Null 型数据表示空值，作用是表明数据空缺的值，一般在设定已存在的变量（或对象的属性）为空时较为常用。区分 Undefined 型和 Null 型数据比较麻烦，一般将 Undefined 型和 Null 型等同对待。

2.5.6 Function 型

Function 型表示函数，可以通过 new 操作符和构造函数 Function()来动态创建所需功能的函数，并为其添加函数体。例如：

```
var myFuntion = new Function()
```

```
{
   staments;
};
```

JavaScript 脚本语言除了支持上述六种基本数据类型外，也支持组合类型，如数组 Array 和对象 Object 等，下面介绍组合类型。

2.6 组合类型

JavaScript 脚本支持的组合类型比基本数据类型更为复杂，包括数组 Array 型和对象 Object 型。本节将简要介绍上述组合类型的基本概念及其用法，在本书后续章节将进行专门论述。

2.6.1 Array 型

Array 型即为数组，数组是包含基本和组合数据的序列。在 JavaScript 脚本语言中，每一种数据类型对应一种对象，数组本质上即为 Array 对象。考察如下定义：

```
var score = [56,34,23,76,45];
```

上述语句创建数组 score，中括号“[]”内的成员为数组元素。由于 JavaScript 是弱类型语言，因此不要求目标数组中各元素的数据类型均相同，例如：

```
var score = [56,34,”23”,76,”45”];
```

由于数组本质上为 Array 对象，则可用运算符 new 来创建新的数组，例如：

```
var score=new Array(56,34,”23”,76,”45”);
```

访问数组中特定元素可通过该元素的索引位置 index 来实现，如下列语句声明变量 m 返回数组 score 中第四个元素：

```
var m = score [3];
```

数组作为 Array 对象，具有最重要的属性 length，用来保存该数组的长度，考察如下的测试代码：

```
源程序 2.3
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 数组的 length 属性 : \n\n";
var myArray=new Array("Tom","Jerry","Lily","Hanks");
//响应按钮的 onclick 事件处理程序
function Test()
{
   GetInfo(myArray);
   msg+="\n 操作语句 :\nmyArray.length=3\n\n";
   myArray.length=3;
   GetInfo(myArray);
   alert(msg);
}
```

```
//输出数组内容
function GetInfo(tempArray)
{
  var myLength=tempArray.length;
  msg+="数组长度 ：\n"+myLength+"\n";
  msg+="数组内容 ：\n";
  for(var i=0;i<myLength;i++)
  {
    msg+="myArray[ "+i+" ] ="+tempArray[i]+"\n";
  }
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value="数组测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“数组测试”按钮，弹出警告框如图 2.4 所示。

图 2.4　数组的 length 属性

值得注意的是，数组的 length 属性为可读可写属性，作为可写属性时，若新的属性值小于原始值时，将调整数组的长度为新的属性值，数组中其余元素将删除。

2.6.2 Object 型

对象为可包含基本和组合数据的组合类型，且对象的成员作为对象的属性，对象的成员函数作为对象的方法。在 JavaScript 脚本语言中，可通过在对象后面加句点“.”并加上对象属性（或方法）的名称来访问对象的属性（或方法），例如：

```
document.bgColor
document.write(“Welcome to JavaScript World!”);
```

考察如下的测试代码：

```
//源程序 2.4
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function Test()        //响应按钮的 onclick 事件处理程序
{
  var msg="\n 对象属性和方法的引用 : \n\n";
  msg+="引用语句 :\nvar myColor=document.bgColor\n";
  msg+="返回结果 :\n"+document.bgColor+"\n";
  msg+="引用语句 :\nwindow.close()\n";
  msg+="返回结果 :\nClose The Window!\n";
  alert(msg);
  window.close();
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
  <input type=button value="数组测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“数组测试”按钮，弹出警告框如图 2.5 所示。

图 2.5　访问对象的属性和方法

单击“确定”按钮，将弹出如图 2.6 所示的警告框提示用户浏览器正试图关闭当前文档页面。

图 2.6　响应 window.close()方法

2.7 运算符

编写 JavaScript 脚本代码过程中，对目标数据进行运算操作需用到运算符。JavaScript 脚本语言支持的运算符包括：赋值运算符、基本数学运算符、位运算符、位移运算符、高级赋值语句、自加和自减、比较运算符、逻辑运算符、逗号运算符、空运算符、?...:…运算符、对象运算符以及 typedof 运算符等，下面分别予以介绍。

2.7.1 赋值运算符

JavaScript 脚本语言的赋值运算符包含“=”、“+=”、“—=”、“*=”、“/=”、“%=”、“&=”、“^=”等，汇总如表 2.3 所示：

表 2.3　赋值运算符

运算符	举例	简要说明
=	m=n	将运算符右边变量的值赋给左边变量
+=	m+=n	将运算符两侧变量的值相加并将结果赋给左边变量
-=	m-=n	将运算符两侧变量的值相减并将结果赋给左边变量
*=	m*=n	将运算符两侧变量的值相乘并将结果赋给左边变量
/=	m/=n	将运算符两侧变量的值相除并将整除的结果赋给左边变量
%=	m%=n	将运算符两侧变量的值相除并将余数赋给左边变量
&=	m&=n	将运算符两侧变量的值进行按位与操作并将结果赋值给左边变量
^=	m^=n	将运算符两侧变量的值进行按位或操作并将结果赋值给左边变量
<<=	m<<=n	将运算符左边变量的值左移由右边变量的值指定的位数，并将操作的结果赋予左边变量
>>=	m>>=n	将运算符左边变量的值右移由右边变量的值指定的位数，并将操作的结果赋予左边变量
>>>=	m>>>=n	将运算符左边变量的值逻辑右移由右边变量的值指定的位数，并将操作的结果赋给左边变量

赋值运算符是编写 JavaScript 脚本代码时最为常用的操作，读者应熟练掌握各个运算符的功能，避免混淆其具体作用。

考察如下的测试程序：

```
//源程序 2.5
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var lValue=48;                //设定初始值
var rValue=3;
function Test()               //响应按钮的 onclick 事件处理程序
{
  var msg="\n 赋值运算符操作 :  \n\n";
  msg+="原始数值 :\nlValue="+lValue+"rValue="+rValue+"\n\n";
  msg+="操作语句及返回结果 :\n\n";
  lValue=rValue;
  msg+="语句 : lValue=rValue         结果 : lValue="+lValue+"rValue="+rValue+"\n";
```

```
    lValue+=rValue;
    msg+="语句 ：lValue+=rValue        结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    lValue-=rValue;
    msg+="语句 ：lValue-=rValue        结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    lValue*=rValue;
    msg+="语句 ：lValue*=rValue        结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    lValue/=rValue;
    msg+="语句 ：lValue/=rValue        结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    lValue%=rValue;
    msg+="语句 ：lValue%=rValue        结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    lValue=13;
    lValue&=rValue;
    msg+="语句 ：lValue&=rValue        结果 ：lValue="+lValue+" rValue="+rValue+"\n";
    lValue^=rValue;
    msg+="语句 ：lValue^=rValue        结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    lValue<<=rValue;
    msg+="语句 ：lValue<<=rValue       结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    lValue>>=rValue;
    msg+="语句 ：lValue>>=rValue       结果 ：lValue="+lValue+" rValue="+rValue+"\n";
    lValue>>>=rValue;
    msg+="语句 ：lValue>>>=rValue      结果 ：lValue="+lValue+"rValue="+rValue+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<hr>
<form>
    <input type=button value="运算符测试" onclick="Test()">
</form>
</body>
</html>
```

程序运行后，在原始页面单击“运算符测试”按钮，弹出警告框如图 2.7 所示。

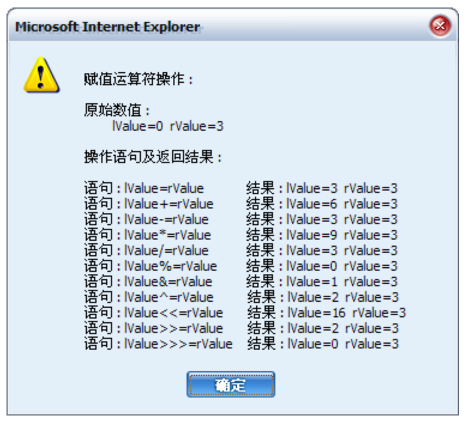

图 2.6 赋值运算符

由上述结果可知，JavaScript 脚本语言的运算符在参与数值运算时，其右侧的变量将保持不变。从本质上讲，运算符右侧的变量作为运算的参数而存在，脚本解释器执行指定的操作后，将运算结果作为返回值赋予运算符左侧的变量。

2.7.2 基本数学运算符

JavaScript 脚本语言中基本的数学运算包括加、减、乘、除以及取余等，其对应的数学运算符分别为“+”、“-”、“*”、“/”和“%”等，如表 2.4 所示：

表 2.4　　基本数学运算符

基本数学运算符	举例	简要说明
+	m=5+5	将两个数据相加，并将结果返回操作符左侧的变量
-	m=9-4	将两个数据相减，并将结果返回操作符左侧的变量
*	m=3*4	将两个数据相乘，并将结果返回操作符左侧的变量
/	m=20/5	将两个数据相除，并将结果返回操作符左侧的变量
%	m=14%3	求两个数据相除的余数，并将结果返回操作符左侧的变量

考察如下测试代码：

```
//源程序 2.6
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var lValue=25;                  //设定初始值
var rValue=4;
function Test()                 //响应按钮的 onclick 事件处理程序
{
  var tempData=0;
  var msg="\n 基本数学运算符 : \n\n";
  msg+="原始数值 :\n\ntempData="+tempData+"lValue="+lValue+" rValue="+rValue+"\n\n";
  msg+="操作语句及返回结果 :\n\n";
  tempData=lValue+rValue;
  msg+="语句 : tempData=lValue+rValue      结果 : tempData="+tempData+"\n";
  tempData=lValue-rValue;
  msg+="语句 : tempData=lValue-rValue       结果 : tempData="+tempData+"\n";
  tempData=lValue*rValue;
  msg+="语句 : tempData=lValue*rValue       结果 : tempData="+tempData+"\n";
  tempData=lValue/rValue;
  msg+="语句 : tempData=lValue/rValue       结果 : tempData="+tempData+"\n";
  tempData=lValue%rValue;
  msg+="语句 : tempData=lValue%rValue      结果 : tempData="+tempData+"\n";
  alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
  <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
```

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

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.8 所示。

图 2.8　基本数学运算符

2.7.3 位运算符

JavaScript 脚本语言支持的基本位运算符包括：“&”、“|”、“^”和“~”等。脚本代码执行位运算时先将操作数转换为二进制数，操作完成后将返回值转换为十进制。位运算符的作用如表 2.5 所示:

表 2.5 位运算符

位运算符	举例	简要说明
&	9&4	按位与，若两数据对应位都是1，则该位为1，否则为0
^	9^4	按位异或，若两数据对应位相反，则该位为1，否则为0
|	9|4	按位或，若两数据对应位都是0，则该位为0，否则为1
~	~4	按位非，若数据对应位为0，则该位为1，否则为0

位运算符在进行数据处理、逻辑判断等方面使用较为广泛，恰当应用位运算符，可节省大量脚本代码。

考察如下测试代码：

```
//源程序 2.7
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var lValue=6;                  //二进制值 0000 0110b
var rValue=36;                 //二进制值 0010 0100b
function Test()                //响应按钮的 onclick 事件处理程序
{
  var tempData=0;
  var msg="\n 基本位运算符 ：  \n\n";
  msg+="原始数值 :\n\ntempData="+tempData+" lValue="+lValue+" rValue="+rValue+"\n\n";
  msg+="操作语句及返回结果 :\n\n";
  tempData=lValue&rValue;
  msg+="语句 : tempData=lValue&rValue    结果 : tempData="+tempData+"\n";
```

```
    tempData=lValue^rValue;
    msg+="语句 : tempData=lValue^rValue      结果 : tempData="+tempData+"\n";
    tempData=lValue|rValue;
    msg+="语句 : tempData=lValue|rValue      结果 : tempData="+tempData+" \n";
    tempData=~lValue;
    msg+="语句 : tempData=~lValue            结果 : tempData="+tempData+"\n";
    alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
    <input type=button value="位运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.9 所示。

图 2.9 位运算符

原始操作数分别为二进制 0000 0110b 和 0010 0100b，执行位与、位异或、位或和位非操作后，其结果分别为二进制 0000 0100b、0010 0010b、0010 0110b 和 1000 0111b，对应的十进制结果分别为 4、34、38 和-7。

2.7.4 位移运算符

位移运算符用于将目标数据往指定方向移动指定的位数。JavaScript 脚本语言支持“<<”、“>>”和“>>>”等位移运算符，其具体作用如见表 2.6：

表 2.6 位移运算符

运算符	举例	简要说明
>>	9>>2	算术右移，将左侧数据的二进制值向左移动由右侧数值表示的位数，右边空位补0
<<	9<<2	算术左移，将左侧数据的二进制值向右移动由右侧数值表示的位数，忽略被移出的位
>>>	9>>>2	逻辑右移，将左边数据表示的二进制值向右移动由右边数值表示的位数，忽略被移出的位，左侧空位补0

位移运算符在逻辑控制、数值处理等方面应用较为广泛，恰当应用位移运算符，可节省大量脚本代码。

考察如下测试代码：

```
//源程序 2.8
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var targetValue=189;          //目标数据二进制值 1011 1101b
var iPos=2;                    //目标数据移动的位数
function Test()               //响应按钮的 onclick 事件处理程序
{
  var tempData=0;
  var msg="\n 位移运算符 :   \n\n";
  msg+="原始数值:\n\ntempData="+tempData+"targetValue="+targetValue+"iPos="+iPos+"\n\n";
  msg+="操作语句及返回结果 :\n\n";
  tempData=targetValue>>iPos;
  msg+="语句 : tempData=targetValue>>iPos      结果 : tempData="+tempData+"\n";
  tempData=targetValue<<iPos;
  msg+="语句 : tempData=targetValue<<iPos      结果 : tempData="+tempData+"\n";
  tempData=targetValue>>>iPos;
  msg+="语句 : tempData=targetValue>>>iPos     结果 : tempData="+tempData+"\n";
  alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
  <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.10 所示。

图 2.10　位移运算符

目标数据的二进制值 1011 1101b，执行算术右移两位、算术左移两位和逻辑右移两位后，

其结果分别为二进制 0010 1111b、10 1110 1100b 和 0010 1111b，分别对应于十进制值 47、756 和 47。

2.7.5 自加和自减

自加运算符为“++”和自减运算符为“--”分别将操作数加 1 或减 1。值得注意的是，自加和自减运算符放置在操作数的前面和后面含义不同。运算符写在变量名前面，则返回值为自加或自减前的值；而写在后面，则返回值为自加或自减后的值。

考察如下测试代码：

```
//源程序 2.9
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var targetValue=9;                //原始数据
function Test()                   //响应按钮的 onclick 事件处理程序
{
   var tempData=0;
   var msg="\自加和自减运算符 ：  \n\n";
   msg+="原始数值 :\n\ntempData="+tempData+"targetValue="+targetValue+"\n\n";
   msg+="操作语句及返回结果 :\n\n";
   tempData=targetValue++;
   msg+="语句 : tempData=targetValue++      结果 : tempData="
          +tempData+"   targetValue="+targetValue+"\n";
   tempData=++targetValue;
   msg+="语句 : tempData=++targetValue      结果 : tempData="
          +tempData+"   targetValue="+targetValue+"\n";
   tempData=targetValue--;
   msg+="语句 : tempData=targetValue--      结果 : tempData="
          +tempData+"   targetValue="+targetValue+"\n";
   tempData=--targetValue;
   msg+="语句 : tempData=--targetValue      结果 : tempData="
          +tempData+"   targetValue="+targetValue+"\n";
   alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
   <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.11 所示。

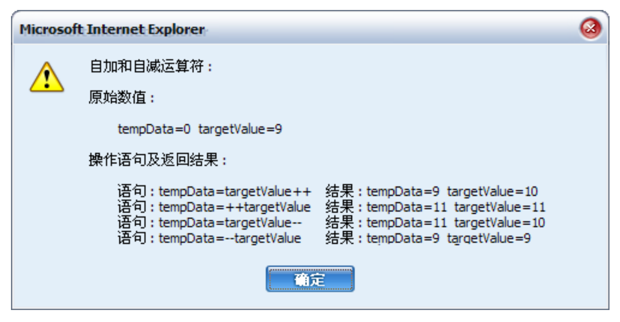

图 2.11　自加和自减运算符

由程序运行结果可以看出：

- 若自加（或自减）运算符放置在操作数之后，执行该自加（或自减）操作时，先将操作数的值赋值给运算符前面的变量，然后操作数自加（或自减）；
- 若自加（或自减）运算符放置在操作数之前，执行该自加（或自减）操作时，操作数先进行自加（或自减），然后将操作数的值赋值给运算符前面的变量。

2.7.6 比较运算符

JavaScript 脚本语言中用于比较两个数据的运算符称为比较运算符，包括“= =”、“!=”、“>”、“<”、“<=”、“>=”等，其具体作用见表 2.7。

表 2.7　比较运算符

运算符	举例	作用
==	num==8	相等，若两数据相等，则返回布尔值true，否则返回false
!=	num!=8	不相等，若两数据不相等，则返回布尔值true，否则返回false
>	num>8	大于，若左边数据大于右边数据，则返回布尔值true，否则返回false
<	num<8	小于，若左边数据小于右边数据，则返回布尔值true，否则返回false
>=	num>=8	大于或等于，若左边数据大于或等于右边数据，则返回布尔值true，否则返回false
<=	num<=8	小于或等于，若左边数据小于或等于右边数据，则返回布尔值true，否则返回false

比较运算符主要用于数值判断及流程控制等方面，考察如下的测试代码。

```
//源程序 2.10
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//响应按钮的 onclick 事件处理程序
function Test()
{
  var myAge=prompt("请输入您的年龄(数值)：",25);
  var msg="\n 年龄测试 ：\n\n";
  msg+="年龄 ："+myAge+" 岁\n";
  if(myAge<18)
    msg+="结果 ： 您处于青少年时期! \n";
  if(myAge>=18&&myAge<30)
```

```
        msg+="结果 ：您处于青年时期! \n";
    if(myAge>=30&&myAge<55)
        msg+="结果 ：您处于中年时期! \n";
    if(myAge>=55)
        msg+="结果 ：您处于老年时期! \n";
    alert(msg);
}
-->
</script>
</head>
<body bgColor="green">
<center>
<form>
    <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出提示框提示用户输入相关信息，如图 2.12 所示。

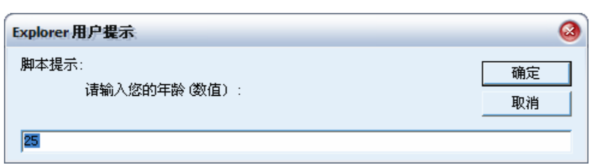

图 2.12　提示用户输入相关信息

在上述提示框输入相关信息（如年龄 35）后，单击“确定”按钮，弹出警告框如图 2.13 所示。

图 2.13 根据用户输入进行特定操作

可以看出，脚本代码采集用户输入的数值，然后通过比较运算符进行判定，再做出相对应的操作，实现了程序流程的有效控制。

> 注意：比较运算符 “==” 与赋值运算符 “=” 截然不同，前者用于比较运算符前后的两个数据，主要用语数值比较和流程控制；后者用于将运算符后面的变量的值赋予运算符前面的变量，主要用于变量赋值。

2.7.7 逻辑运算符

JavaScript 脚本语言的逻辑运算符包括 “&&”、“||” 和 “!” 等，用于两个逻辑型数据

之间的操作，返回值的数据类型为布尔型。逻辑运算符的功能如表 2.8 所示：

表 2.8　逻辑运算符

运算符	举例	作用
&&	num<5&&num>2	逻辑与，如果符号两边的操作数为真，则返回true，否则返回false
||	num<5||num>2	逻辑或，如果符号两边的操作数为假，则返回false，否则返回true
!	!num<5	逻辑非，如果符号右边的操作数为真，则返回false，否则返回true

逻辑运算符一般与比较运算符捆绑使用，用以引入多个控制的条件，以控制 JavaScript 脚本代码的流向。

2.7.8 逗号运算符

编写 JavaScript 脚本代码时，可使用逗号“,”将多个语句连在一起，浏览器载入该代码时，将其作为一个完整的语句来调用，但语句的返回值是最右边的语句。

考察如下的测试代码：

```
//源程序 2.11
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//响应按钮的 onclick 事件处理程序
function Test()
{
  var dataA,dataB,dataC,dataD;
  dataA=(dataB=1,dataC=2,dataD=3);
  var msg="\n 逗号运算符测试 ：\n\n";
  msg+="赋值语句 ：\ndataA=(dataB=1,dataC=2,dataD=3);\n";
  msg+="赋值结果 ：\n";
  msg+="dataA = "+dataA+"\n";
  msg+="dataB = "+dataB+"\n";
  msg+="dataC = "+dataC+"\n";
  msg+="dataD = "+dataD+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.14 所示。

图 2.14　逗号“,”运算符

由运行结果可知，使用长语句赋值时，返回值为赋值语句最右边变量的值，为养成良好的编程习惯，建议不是用该方法。逗号“,”一般用于在函数定义和调用时分隔多个参数，例如：

```
function sum(a,b,c){
  statements
}
```

2.7.9 空运算符

空运算符对应的关键字为“void”，其作用是定义一个表达式，但该表达式并不返回任何值。修改源程序 2.11 中变量赋值语句为：

```
dataA=(dataB=1,dataC=2,dataD=3);
```

保存代码后，使用浏览器载入，在原始页面单击“运算符测试”按钮，弹出警告框如图 2.15 所示。

图 2.15　空运算符

由程序运行结果可知，使用空运算符 void 后，变量 dataA 定义为 undefined 型，并不返回任何值。

2.7.10 ?...:运算符

在 JavaScript 脚本语言中，“?...:”运算符用于创建条件分支。在动作较为简单的情况下，较之 if…else 语句更加简便，其语法结构如下：

```
(condition)?statementA:statementB;
```

载入上述语句后，首先判断条件 condition，若结果为真则执行语句 statementA，否则执行语句 statementB。值得注意的是，由于 JavaScript 脚本解释器将分号“;”作为语句的结束符，statementA 和 statementB 语句均必须为单个脚本代码，若使用多个语句会报错，例如下

列代码浏览器解释执行时得不到正确的结果：

```
(condition)?statementA:statementB;ststementC;
```

考察如下简单的分支语句：

```
var age= prompt("请输入您的年龄(数值)：",25);
var contentA="\n 系统提示 ：\n 对不起，您未满 18 岁，不能浏览该网站！\n";
var contentB="\n 系统提示 ：\n 点击"确定"按钮，注册网上商城开始欢乐之旅！"
if(age<18)
{
  alert(contentA);
}
else{
  alert(contentB);
}
```

程序运行后，单击原始页面中“测试”按钮，弹出提示框提示用户输入年龄，并根据输入值决定后续操作。例如在提示框中输入整数 17，然后单击“确定”按钮，则弹出警告框如图 2.16 所示：

图 2.16 输入值为 12

若在提示框中输入整数 24，然后单击“确定”按钮，则弹出警告框如图 2.17 所示：

图 2.17 输入值为 24

上述语句中的条件分支语句完全可由“?...:”运算符简单表述：

```
(age<18)?alert(contentA):alert(contentB);
```

可以看出，使用“?...:”运算符进行简单的条件分支，语法简单明了，但若要实现较为复杂的条件分支，推荐使用 if…else 语句或者 switch 语句。

2.7.11 对象运算符

JavaScript 脚本语言主要支持四种对象运算符，包括点号运算符、new 运算符、delete 运算符以及（）运算符等。

对象包含属性和方法，点号运算符用来访问对象的属性和方法。其用法是将对象名称与对象的属性（或方法）用点号隔开，例如：

```
var myColor=document.bgColor;
window.alert(msg);
```

语句一使用变量 myColor 返回 Document 对象的 bgColor 属性，语句二调用 Window 对象的 alert()方法输出提示信息。当然，也可使用双引号“[]”来访问对象的属性，改写上述语句：

```
var myColor=document[" bgColor "];
```

new 运算符用来创建新的对象，例如创建一个新的数组对象，可以写成：

```
var exam = new Array (43,76,34 89,90);
```

new 运算符可以创建程序员自定义的对象，以可以创建 JavaScript 内建对象的实例。下列函数创建 Date 对象，并调用 Window 对象的 alert()方法输出当前时间信息：

```
function createDate()
{
  var myDate=new Date();
  var msg="\n 当前时间 : \n\n";
  msg+="          "+myDate+"           \n";
  alert(msg)
}
```

上述函数被调用后，弹出警告框显示当前时间信息，如图 2.18 所示。

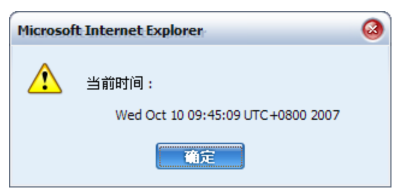

图 2.18　new 运算符

delete 运算符主要用于删除数组的特定元素，也可用来删除对象的属性、方法等。考察如下的测试代码：

```
//源程序 2.12
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\ndelete 运算符测试 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
  var myClassmate=new Array("JHX","LJY","QZY","HZF");
  getInfo(myClassmate);
  delete myClassmate[2];
  msg+="\n 执行语句 : \n delete myClassmate[2];\n\n";
  getInfo(myClassmate);
  alert(msg);
}
//获取当前数组信息
function getInfo(iArray)
{
  var myLength=iArray.length;
```

```
  msg+="数组长度 ：\n "+myLength+"\n";
  msg+="数组元素 ：\n";
  for(i=0;i<myLength;i++)
  {
    msg+="iArray[ "+i+" ]="+iArray[i]+"\n";
  }
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“运算符测试”按钮，将弹出警告框如图 2.19 所示。

图 2.19　delete 运算符

由上图可知，执行“delete myClassmate[2];”语句后，数组元素 myClassmate[2]被定义为 undefined 类型。

“()”运算符用来调用对象的方法，例如：

```
window.alert(msg);
```

上述四种对象运算符，在后续章节将进行更为深入的介绍。

2.7.12 typeof 运算符

typeof 运算符用于表明操作数的数据类型，返回数值类型为一个字符串。在 JavaScript 脚本语言中，其使用格式如下：

```
var myString=typeof(data);
```

考察如下实例：

```
//源程序 2.13
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\ntypeof 运算符测试 ：\n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
  var myData;
  msg+="语句 ：var myData;     类型 ："+typeof(myData)+" \n";
  myData=5;
  msg+="语句 ：myData=5;       类型 ："+typeof(myData)+"\n";
  myData="5";
  msg+="语句 ：myData="5";     类型 ："+typeof(myData)+"\n";
  myData=true;
  msg+="语句 ：myData=true;   类型 ："+typeof(myData)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value="运算符测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，出现如图 2.20 所示页面：

图 2.20　typeof 运算符

可以看出，使用关键字 var 定义变量时，若不指定其初始值，则变量的数据类型默认为 undefined。同时，若在程序执行过程中，变量被赋予其他隐性包含特定数据类型的数值时，其数据类型也随之发生更改。

2.7.13 运算符优先级

JavaScript 脚本编程中，运算表达式中可能含有多个运算符，同其他程序语言一样，这些运算符也是有处理的先后顺序的，运算符的优先级如表 2.9 所示。

表 2.9　运算符优先级

运算符优先级	运算符	简要说明
1	( )	
	[ ]	
2	!	逻辑非
	~	按位非
	—	取负
	+ +	自加
	——	自减
	typeof	表明数据类型
3	*	乘
	/	除
	%	取余
4	+	
	—	
5	<<	按位移
	>	
	>>	
6	<	比较运算符
	>	
	<=	
	>=	
7	= =	
	!=	
8	&	按位与
9	^	按位异或
10	|	按位或
11	&&	逻辑与
12	||	逻辑或
13	?	条件表达式
14	=	赋值运算符
	+=	
	—=	
	*=	
	/=	
	%=	
	<<=	
	>=	
	>>=	
	&=	
	^=	
	|=	
15	,	参数分隔

进行表达式求值时，先执行优先级脚高的运算符，再执行优先级较低的运算符；若优先级相同则按照从左至右的顺序执行。构造特定运算功能的表达式时，应根据上述表格中列举的运算符优先级合理安排。

2.8 核心语句

前面小节讲述了 JavaScript 脚本语言数据结构方面的基础知识，包括基本数据类型、运算符、运算符优先级等，本节将重点介绍 JavaScript 脚本的核心语句。

在 JavaScript 脚本语言中，语句的基本格式为：

<statement>;

分号为语句结束标志符，为养成良好的编程习惯，在编程中应使用分号。值得注意的是，JavaScript 脚本支持符号匹配，如双引号、单引号等。若分号嵌套在上述匹配符号内，脚本解释器搜索匹配的符号：

- 若存在匹配符，则将其中的分号作为普通符号而不是作为语句结束符对待。例如：

```
var msg="语句 : var myData;";
```

- 若不存在匹配符，则提示脚本出现语法错误。例如：

```
var msg="语句 : var myData;
```

基本语句构成代码段，下面介绍 JavaScript 脚本代码的基本处理流程

2.8.1 基本处理流程

基本处理流程就是对数据结构的处理流程，在 JavaScript 里，基本的处理流程包含三种结构，即顺序结构、选择结构和循环结构。

顺序结构即按照语句出现的先后顺序依次执行，为 JavaScript 脚本程序中最基本的结构，如图 2.21 所示。

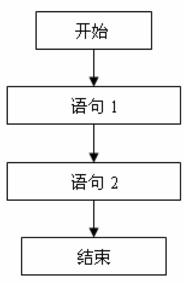

图 2.21 顺序结构

选择结构即按照给定的逻辑条件来决定执行顺序，可以分为单向选择、双向选择和多向选择。但无论是单向还是多向选择，程序在执行过程中都只能执行其中一条分支。单向选择和双向选择结构如图 2.22 所示。

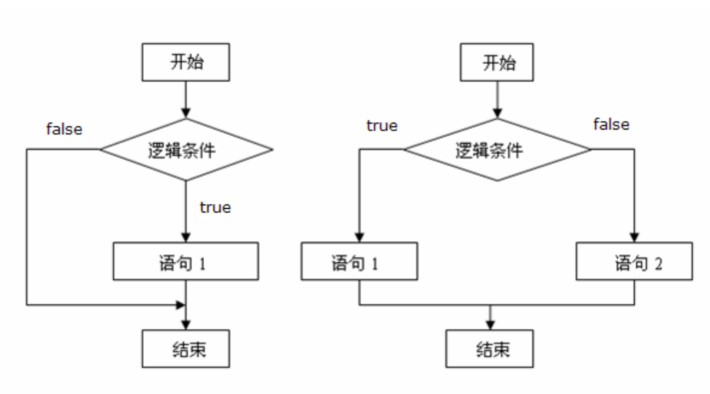

图 2.22 单向和双向选择结构

循环结构即根据代码的逻辑条件来判断是否重复执行某一段程序。若逻辑条件为 true, 则重复执行，即进入循环，否则结束循环。循环结构可分为条件循环和计数循环，如图 2.23 所示。

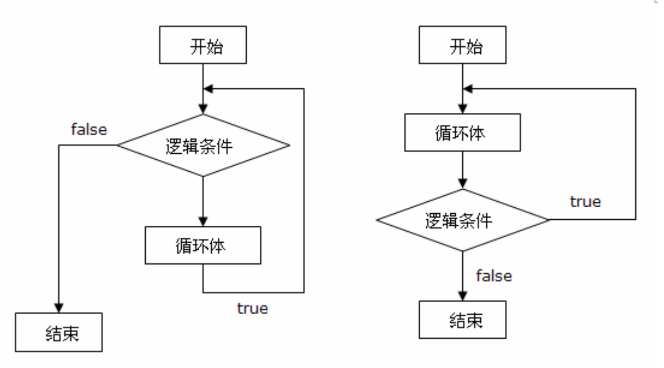

图 2.23 循环结构

一般而言，在 JavaScript 脚本语言中，程序总体是按照顺序结构执行的，而在顺序结构中可以包含选择结构和循环结构。

2.8.2 if 条件假设语句

if 条件假设语句是比较简单的一种选择结构语句，若给定的逻辑条件表达式为真，则执行一组给定的语句。其基本结构如下：

```
if(conditions)
{
 statements;
}
```

逻辑条件表达式 conditions 必须放在小括号里，且仅当该表达式为真时，执行大括号内包含的语句，否则将跳过该条件语句而执行其下的语句。大括号内的语句可为一个或多个，当仅有一个语句时，大括号可以省略。但一般而言，为养成良好的编程习惯，同时增强程序代码的结构化和可读性，建议使用大括号将指定执行的语句括起来。

if 后面可增加 else 进行扩展，即组成 if…else 语句，其基本结构如下：

```
if(conditions)
{
  statement1;
}
else
{
  statement2;
}
```

当逻辑条件表达式 conditions 运算结果为真时，执行 statement1 语句（或语句块），否则执行 statement2 语句（或语句块）。

if(或 if…else)结构可以嵌套使用来表示所示条件的一种层次结构关系。值得注意的是，嵌套时应重点考虑各逻辑条件表达式所表示的范围。

2.8.3 switch 流程控制语句

在 if 条件假设语句中，逻辑条件只能有一个，如果有多个条件，可以使用嵌套的 if 语句来解决，但此种方法会增加程序的复杂度，并降低程序的可读性。若使用 switch 流程控制语句就可完美地解决此问题，其基本结构如下：

```
switch (a)
{
  case a1:
    statement 1;
    [break;]
  case a2:
    statement 2;
    [break];
  ……
  default:
    [statement n;]
}
```

其中 a 是数值型或字符型数据，将 a 的值与 a1、a2、……比较，若 a 与其中某个值相等时，执行相应数据后面的语句，且当遇到关键字 break 时，程序跳出 statement n 语句，并重新进行比较；若找不到与 a 相等的值，则执行关键字 default 下面的语句。

考察如下的测试代码：

```
//源程序 2.14
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\nswitch 流程控制语句 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
  var year=window. prompt("请输入您的军龄(整数值,0 表示未参军) : ",25);
  var army;
  switch(year)
  {
    case 0:
      army="平民";
      break;
    case 1:
      army="列兵";
      break;
    case 2:
      army="上等兵";
      break;
    case 3:
    case 4:
    case 5:
      army="一级士官";
      break;
    case 6:
    case 7:
    case 8:
      army="二级士官";
      break;
```

```
    default:
      if (year>8)
        army="中高级士官";
  }
  msg+="军龄 ："+year+"年\n";
  msg+="结论 ："+army+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<form>
  <input type=button value="测试" onclick="Test()">
</form>
</body>
</html>
```

程序运行后，在原始页面中单击“测试”按钮，将弹出提示框提示用户输入相关信息，例如输入 12，单击“确定”按钮提交，弹出警告框如图 2.24 所示。

图 2.24　switch 流程控制语句

2.8.4 for 循环语句

for 循环语句是循环结构语句，按照指定的循环次数，循环执行循环体内语句（或语句块），其基本结构如下：

```
for(initial condition; test condition; alter condition)
{
   statements;
}
```

循环控制代码（即小括号内代码）内各参数的含义如下：

- initial condition 表示循环变量初始值；
- test condition 为控制循环结束与否的条件表达式，程序每执行完一次循环体内语句（或语句块），均要计算该表达式是否为真，若结果为真，则继续运行下一次循环体内语句（或语句块）；若结果为假，则跳出循环体。
- alter condition 指循环变量更新的方式，程序每执行完一次循环体内语句（或语句块），均需要更新循环变量。

上述循环控制参数之间使用分号“;”间隔开来，考察如下的测试函数：

```
function Test()
{
  var iArray=new Array("JHX","QZY","LJY","HZF");
  var iLength=iArray.length;
  var msg="\nfor 循环语句测试 :\n\n";
```

```
    msg+="数组长度 ：\n "+iLength+"\n";
    msg+="数组元素 ：\n";
    for(var i=0;i<iLength;i++)
    {
        msg+="iArray["+i+"] ="+iArray[i]+" \n";
    }
    alert(msg);
}
```

上述函数被调用后，弹出警告框如图 2.25 所示。

图 2.25　for 循环语句举例

2.8.5 while 和 do-while 循环语句

while 语句与 if 语句相似，均有条件地控制语句（或语句块）的执行，其语言结构基本相同:

```
while(conditions)
{
  statements;
}
```

while 语句与 if 语句的不同之处在于：在 if 条件假设语句中，若逻辑条件表达式为真，则运行 statements 语句（或语句块），且仅运行一次；while 循环语句则是在逻辑条件表达式为真的情况下，反复执行循环体内包含的语句（或语句块）。

注意：while 语句的循环变量的赋值语句在循环体前，循环变量更新则放在循环体内；for 循环语句的循环变量赋值和更新语句都在 for 后面的小括号中，在编程中应注意二者的区别。

改写 Test()函数代码如下，程序运行结果不变：

```
function Test()
{
    var iCount=0;
    var iArray=new Array("JHX","QZY","LJY","HZF");
    var iLength=iArray.length;
    var msg="\nfor 循环语句测试 :\n\n";
    msg+="数组长度 ：\n "+iLength+"\n";
    msg+="数组元素 ：\n";
    while(iCount<iLength)
    {
        msg+="iArray["+iCount+"] ="+iArray[iCount]+" \n";
        iCount+=1;
    }
    alert(msg);
```

```
}
```

在某些情况下，while 循环大括号内的 statements 语句（或语句块）可能一次也不被执行，因为对逻辑条件表达式的运算在执行 statements 语句（或语句块）之前。若逻辑条件表达式运算结果为假，则程序直接跳过循环而一次也不执行 statements 语句（或语句块）。

若希望至少执行一次 statements 语句（或语句块），可改用 do…while 语句，其基本语法结构如下：

```
do {
  statements;
}while(condition);
```

改写 Test()函数代码如下，程序运行结果不变：

```
function Test()
{
  var iCount=0;
  var iArray=new Array("JHX","QZY","LJY","HZF");
  var iLength=iArray.length;
  var msg="\nfor 循环语句测试 :\n\n";
  msg+="数组长度 : \n "+iLength+"\n";
  msg+="数组元素 : \n";
  do{
    msg+="iArray["+iCount+"] ="+iArray[iCount]+" \n";
    iCount+=1;
  }while(iCount<iLength);
  alert(msg);
}
```

for、while、do…while 三种循环语句具有基本相同的功能，在实际编程过程中，应根据实际需要和本着使程序简单易懂的原则来选择到底使用哪种循环语句。

2.8.6 使用 break 和 continue 进行循环控制

在循环语句中，某些情况下需要跳出循环或者跳过循环体内剩余语句，而直接执行下一次循环，此时需要通过 break 和 continue 语句来实现。break 语句的作用是立即跳出循环，continue 语句的作用是停止正在进行的循环，而直接进入下一次循环。

考察如下测试代码：

```
//源程序 2.15
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 使用 break 和 continue 控制循环 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
  var n=-1;
  var iArray=new Array("YSQ","JHX","QZY","LJY","HZF","XGM","LJY","LHZ");
  var iLength=iArray.length;
```

```
    msg+="数组长度 : \n "+iLength+"\n";
    msg+="数组元素 : \n";
    while(n<iLength)
    {
      n+=1;
      if(n==3)
        continue;
      if(n==6)
        break;
      msg+="iArray["+n+"] = "+iArray[n]+"\n";
    }
    alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
    <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面中单击“测试”按钮，弹出警告框如图 2.26 所示。

图 2.26 break 和 continue 语句

从上图的结果可以看出：

- 当 n=3 时，跳出当前循环而直接进行下一个循环，故 iArray[3]不进行显示；
- 当 n=6 时，直接跳出 while 循环，不再执行余下的循环，故 iArray[5]之后的数组元素不进行显示。

在编写 JavaScript 脚本过程中，应根据实际需要来选择使用哪一种循环语句，并确保在使用了循环控制语句 continue 和 break 后，循环不出现任何差错。

2.8.7 with 对象操作语句

在编写 JavaScript 脚本过程中，经常需引用同一对象的多个属性或方法，正常的对象属性或方法的引用途径能达到既定的目的，但代码显得尤为复杂。JavaScript 脚本语言提供 with 操作语句来简化对象属性和方法的引用过程，其语法结构如下：

```
with (objct)
```

```
{
    statements;
}
```

例如下列连续引用 document 对象的 write()方法的语句：

```
document.write("Welcome to China");
document.write("Welcome to Beijing");
document.write("Welcome to Shanghai");
```

可以使用 with 语句简化为：

```
with(document)
{
    write("Welcome to China");
    write("Welcome to Beijing");
    write("Welcome to Shanghai");
}
```

在脚本代码中适当使用 with 语句可使脚本代码简明易懂，避免不必要的重复输入。若脚本代码中涉及到多个对象，不推荐使用 with 语句，避免造成属性或方法引用的混乱。

2.8.8 使用 for…in 进行对象循环

使用 for…in 循环语句可以对指定对象的属性和方法进行遍历，其语法结构如下：

```
for (变量名 in 对象名)
{
    statements;
}
```

考察如下测试代码：

```
源程序 2.16
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\nfor...in 对象循环语句遍历对象 :                    \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var i=0;
   msg+="Window 对象支持的属性和方法 : \n";
   for(num in window)
   {
      msg+=num+"             ";
      i+=1;
      if((i%5)==0)
         msg+="\n";
   }
   alert(msg);
}
-->
</script>
```

```
</head>
<body>
<form>
  <input type=button value="测试" onclick="Test()">
</form>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.27 所示。

图 2.27　for…in 对象循环语句

2.8.9 含标签的语句

经常在循环标志前加上标签文本来引用该循环，其使用方法是标识符后面加冒号“:”。在使用 break 和 continue 语句配合使用控制循环语句时，可使用 break 或 continue 加上标识符的形式使循环跳转到指定的位置。

考察如下的测试代码：

```
//源程序 2.17
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 使用标签控制循环语句 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
  msg+="循环流程 : \n";
  outer:
  for(m=1;m<4;m++)
  {
    msg+="外循环  第"+m+"次\n";
    for(n=1;n<4;n++)
    {
      if(n==2)
        break outer;
      msg+="内循环  第"+n+"次\n";
    }
```

```
   }
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.28 所示。

图 2.28　使用标签控制循环

若不加标签 outer，而直接使用 break 语句跳出循环，其运行结果如图 2.29 所示。

图 2.29　不使用标签 outer

比较上述两个实例可知：

- 若增加标签 outer，则执行到 break outer 语句时跳出整个 while 循环；
- 若直接使用 break 语句仅跳出 break 所在的 for 循环。

由此可见，标签对循环控制非常有效。若配合 break 和 continue 语句使用，可精确控制循环的走向，在实际编写脚本代码过程中应根据需要选择添加标签与否。

2.9 函数

JavaScript 脚本语言允许开发者通过编写函数的方式组合一些可重复使用的脚本代码块，增加了脚本代码的结构化和模块化。函数是通过参数接口进行数据传递，以实现特定的

功能。本小节将重点介绍函数的基本概念、组成、全局函数与局部函数、作为对象的函数以及递归函数等知识，让读者从头开始，学习如何编写执行效率高、代码利用率高，且易于查看和维护的函数。

2.9.1 函数的基本组成

函数由函数定义和函数调用两部分组成，应首先定义函数，然后再进行调用，以养成良好的编程习惯。

函数的定义应使用关键字 function，其语法规则如下：

```
function funcName ([parameters])
{
   statements;
   [return 表达式;]
}
```

函数的各部分含义如下：

- funcName 为函数名，函数名可由开发者自行定义，与变量的命名规则基本相同；
- parameters 为函数的参数，在调用目标函数时，需将实际数据传递给参数列表以完成函数特定的功能。参数列表中可定义一个或多个参数，各参数之间加逗号“,”分隔开来，当然，参数列表也可为空；
- statements 是函数体，规定了函数的功能，本质上相当于一个脚本程序；
- return 指定函数的返回值，为可选参数。

自定义函数一般放置在 HTML 文档的<head>和</head>标记对之间。除了自定义函数外，JavaScript 脚本语言提供大量的内建函数，无需开发者定义即可直接调用，例如 window 对象的 alert()方法即为 JavaScript 脚本语言支持的内建函数。

函数定义过程结束后，可在文档中任意位置调用该函数。引用目标函数时，只需在函数名后加上小括号即可。若目标函数需引入参数，则需在小括号内添加传递参数。如果函数有返回值，可将最终结果赋值给一个自定义的变量并用关键字 return 返回。

考察如下测试代码：

```
//源程序 2.18
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 函数调用实例 : \n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var i=10;
   var j=15;
   var temp=sum(i,j);
   msg+="函数参数 : \n";
   msg+="参数 1:     i="+i+"\n";
   msg+="参数 2:     j="+j+"\n";
   msg+="调用语句 : \n";
```

```
    msg+="var temp=sum(i,j); \n";
    msg+="返回结果 : \n";
    msg+=""+i+"+"+j+" = "+temp+"\n";
    alert(msg);
}
//计算两个数的加和
function sum(data1,data2)
{
    var tempData=data1+data2;
    return tempData;
}
-->
</script>
</head>
<body>
<center>
<form>
    <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.30 所示。

图 2.30　函数调用

上述代码中，定义了实现两数加和的函数 sum(data1,data2)及响应“测试”按钮 onclick 事件处理程序的 Test()函数，并在后者内部调用了 window 对象的内建函数 alert()，实现了函数的相互引用。

如果函数中引用的外部函数较多或函数的功能很复杂，势必导致函数代码过长而降低脚本代码可读性，违反了开发者使用函数实现特定功能的初衷。因此，在编写函数时，应尽量保持函数功能的单一性，使脚本代码结构清晰、简单易懂。

2.9.2 全局函数与局部函数

JavaScript 脚本语言提供了很多全局（内建）函数，在脚本编程过程中可直接调用，在此介绍四种简单的全局函数：parseInt()、parseFloat()、escape()和 unescape()。

parseInt()函数的作用是将字符串转换为整数，parseFloat()函数的作用是将字符串转换为浮点数；escape()函数的作用是将一些特殊字符转换成 ASCII 码，而 unescape()函数的作用是将 ASCII 码转换成字符。

考察如下测试代码：

```
//源程序 2.19
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 全局函数调用实例 ：\n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
  var string1="30121";
  var string2="34.12";
  var string3="Money*#100";
  var temp1,temp2,temp3,temp4;
  msg+="原始变量 ：\n";
  msg+="string1 = "+string1+"类型 ："+typeof(string1)+"\n";
  msg+="string2 = "+string2+"类型 ："+typeof(string2)+"\n";
  msg+="string3 = "+string3+"类型 ："+typeof(string3)+"\n";
  msg+="执行语句与结果:\n";
  temp1=parseInt(string1);
  temp2=parseInt(string2);
  msg+="语句 ：parseInt(string1)   结果 ：string1="+temp1+"类型 ："+typeof(temp1)+"\n";
  msg+="语句 ：parseInt(string2)   结果 ：string1="+temp2+"类型 ："+typeof(temp2)+"\n";
  temp1=parseFloat(string1);
  temp2=parseFloat(string2);
  msg+="语句 ：parseFloat(string1)   结果 ：string1="+temp1+"类型 ："+typeof(temp1)+"\n";
  msg+="语句 ：parseFloat(string2)   结果 ：string1="+temp2+"类型 ："+typeof(temp2)+"\n";
  temp3=escape(string3);
  msg+="语句 ：temp3=escape(string3)      结果 ：temp3="+temp3+"类型 ："+typeof(temp3)+"\n";
  temp4=unescape(temp3);
  msg+="语句 ：temp4=unescape(temp3)   结果 ：temp4="+temp4+"类型 ："+typeof(temp4)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.31 所示。

图 2.31 全局函数

由程序运行结果可知上述全局函数的具体作用，当然 JavaScript 脚本语言还支持很多其他的全局函数，在编程中适当使用它们可大大提高编程效率。

与全局函数相对应的函数是局部函数，即定义在某特定函数内部，并仅能在其内使用的函数。

考察如下测试代码：

```
//源程序 2.20
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 局部函数调用实例 : \n\n";
//响应按钮的 onclick 事件处理程序
function multi(m,n)
{
  var result;
  function inner(m)
  {
    if (m%2!=0)
      return 0;
    else
      return 1;
  }
  result=inner(m)*n;
  msg+="输入参数 : \n";
  msg+="m = "+m+"\n            n = "+n+"\n";
  msg+="乘积结果 : \n";
  msg+="result = "+result+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value="测试" onclick="multi(4,3)">
```

```
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.32 所示。

图 2.32 局部函数

函数 muti()内部定义了局部函数 inner()，判断变量 m 是否为偶数，如果是偶数则返回 1，否则返回 0。根据调用语句 muti(4,3)，m=4 为偶数，故局部函数 inner()返回值为 1，函数 muti()的返回值为 3。

注意：通过上述方式定义的函数为局部函数，函数的作用域为自所属的框架函数，任何处于框架函数外部对局部函数的引用均为不合法。

2.9.3 作为对象的函数

JavaScript 脚本语言中所有的数据类型、数组等均可作为对象对待，函数也不例外。可以使用 new 操作符和 Function 对象的构造函数 Function()来生成指定规则的函数，其基本语法如下：

```
var funcName = new Function (arguments,statements;);
```

值得注意的是，上述的构造函数 Function()首字母必须为大写，同时函数的参数列表与操作代码之间使用逗号隔开。考察如下测试代码：

```
//源程序 2.21
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 使用构造函数 Function 构造函数 : \n\n";
//使用 new 操作符和 Function()构造函数生成函数 newFunc()
var newFunc=new Function("result","alert(msg+'      '+result)");
//响应按钮的 onclick 事件处理程序
function Test()
{
   msg+="生成语句: \n";
   msg+="var newFunc=new Function(''result'',''alert(msg+result)''); \n"
   msg+="调用语句: \n";
   msg+="newFunc(''Welcome to JavaScript World!'');\n";
```

```
    msg+="返回结果: \n";
    newFunc("Welcome to JavaScript World!");
}
-->
</script>
</head>
<body>
<center>
<form>
    <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.33 所示。

图 2.33　作为对象的函数

通过 new 操作符和 Function()构造函数定义函数对象时，并没有给函数赋予名称，而是定义函数后直接将其赋值给变量 newFunc，并通过 newFunc 进行访问，与通常的函数定义不同。

> 注意：在定义函数对象时，参数列表可以为空，也可有一个或多个参数，使用变量引用该函数时，应将函数执行所需要的参数传递给函数体。

作为对象的函数最重要的性质即为它可以创建静态变量，给函数增加实例属性，使得函数在被调用之间也能发挥作用。考察如下测试代码：

```
//源程序 2.22
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 作为对象的函数创建静态变量 : \n\n";
function sum(x,y)
{
    sum.result=sum.result+x+y;
    return(sum.result);
}
sum.result=0;
//响应按钮的 onclick 事件处理程序
```

```
function Test()
{
   var tempData;
   msg+="调用语句及返回结果: \n";
   tempData=sum(2,3);
   msg+="语句 ：tempData=sum(2,3);";
   msg+="结果 ：tempData = "+tempData+"      sum.result = "+sum.result+"\n";
   tempData=sum(4,5);
   msg+="语句 ：tempData=sum(4,5);";
   msg+="结果 ：tempData = "+tempData+"      sum.result = "+sum.result+"\n";
   tempData=sum(6,7);
   msg+="语句 ：tempData=sum(6,7);";
   msg+="结果 ：tempData = "+tempData+"      sum.result = "+sum.result+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.34 所示。

图 2.34　创建静态变量

由上述结果可以看出，作为对象的函数使用静态变量后，可以用来保存其运行的环境参数如中间值等数据。

2.9.4 函数递归调用

函数的递归调用即函数在定义时调用自身，考察如下实例代码：

```
//源程序 2.35
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
```

```
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 函数的递归调用 ：\n\n";
//响应按钮的 onclick 事件处理程序
function Test()
{
   var result;
   msg+="调用语句 ：\n";
   msg+="result = sum(6);\n";
   msg+="调用步骤 ：\n";
   result=sum(6);
   msg+="计算结果 ：\n";
   msg+="result = "+result+"\n";
   alert(msg);
}
//计算当前步骤加和值
function sum(m)
{
   if(m==0)
      return 0;
   else
   {
      msg+="语句 ：result = " +m+ "+sum(" +(m-1)+"); \n";
      result=m+sum(m-1);
   }
   return result;
}
-->
</script>
</head>
<body>
<center>
<form>
   <input type=button value="测试" onclick="Test()">
</form>
</center>
</body>
</html>
```

程序运行后，在原始页面单击“测试”按钮，弹出警告框如图 2.35 所示。

图 2.35　函数递归调用

函数递归调用能使代码显得紧凑、简练，但也存在执行效率并低、容易出错、资源耗费

较多等问题，推荐在递归调用次数较少的情况下使用该方法，其余情况尽量使用其余方法来代替。

2.9.5 语言注释语句

在 JavaScript 脚本代码中，可加如一些提示性的语句，以便提示代码的用途及跟踪程序执行的流程，并增加程序的可读性，同时有利于代码的后期维护。上述提示性语句称作语言注释语句，JavaScript 脚本解释器并不执行语言解释语句。

一般使用双反斜杠“//”作为每行解释语句的开头，例如：

```
// 程序解释语句
```

值得注意的是，必须在每行注释语句前均加上双反斜杠“//”。例如：

```
// 程序解释语句 1
// 程序解释语句 2
```

下面的语句为错误的注释语句：

```
// 程序解释语句 1
   程序解释语句 2
```

上述语句在中第二行不被脚本解释器作为解释语句，而作为普通的代码对待，即由脚本解释器解释执行。

对于多行的解释语句，可以在解释语句开头加上“/*”，末尾加上*/，不须在每行开头加上双反斜杠“//”。例如：

```
/* 程序解释语句 1
    程序解释语句 2
    程序解释语句 3 */
```

JavaScript 脚本语言中，还允许使用 HTML 风格的语言注释，即用“<!--”来代替双反斜杠“//”。与 HTML 文档注释不同的是，HTML 允许“<!--”跨越多行进行注释，而 JavaScript 脚本里必须在每行注释前加上“<!--”。另外，JavaScript 脚本里不需要以“-->”来结束注释语句，而 HTML 中则必须用“-->”来结束注释语句。

为养成良好的编程习惯，最好不用 HTML 风格的语言注释语句，而使用双反斜杠“//”来加入单行注释语句，用“/*”和“*/”加入多行注释语句。

2.9.6 函数应用注意事项

最后介绍一下在使用函数过程中应特别予以注意的几个问题，以帮助读者更好、更准确确地使用函数，并养成良好的编程习惯。具体表现在如下几点：

- 定义函数的位置：如果函数代码较为复杂，函数之间相互调用较多，应将所有函数的定义部分放在 HTML 文档的<head>和</head>标记对之间，既可保证所有的函数在调用之前均已定义，又可使开发者后期的维护工作更为简便；
- 函数的命名：函数的命名原则与变量的命名原则相同，但尽量不要将函数和变量取同一个名字。如因实际情况需要将函数和变量定义相近的名字，也应给函数加上可以清楚辨认的字符（如前缀 func 等）以示区别；
- 函数返回值：在函数定义代码结束时，应使用 return 语句返回，即使函数不需要返回任何值；
- 变量的作用域：区分函数中使用的变量是全局变量还是局部变量，避免调用过程中出现难以检查的错误；

- 函数注释：在编写脚本代码时，应在适当的地方给代码的特定行添加注释语句，例如将函数的参数数量、数据类型、返回值、功能等注释清楚，既方便开发者对程序的后期维护，也方便其他人阅读和使用该函数，便于模块化编程；
- 函数参数传递：由于 JavaScript 是弱类型语言，使用变量时并不检查其数据类型，导致一个潜在的威胁，即开发者调用函数时，传递给函数的参数数量或数据类型不满足要求而导致错误的出现。在函数调用时，应仔细检查传递给目标函数的参数变量的数量和数据类型。

其中第五点尤为值得特别关注，因由其导致的错误非常难于检测。考察如下两数乘法的测试代码：

```
function muti(x,y)
{
  if(muti.arguments.length==2)
    return (x*y);
}
```

以上代码检查了输入参数的数量，但未检查操作数的数据类型，例如输入字符串“num1”和“num2”，并调用 multi(num1,num2)时，浏览器报错。修改函数 multi()如下：

```
function muti(x,y)
{
  if(muti.arguments.length==2)
  {
    if( (typeof(x)!="number" || (typeof(y)!="number"))
        return errorNum;
    else
        return (x*y);
    }
return;
}
```

这个函数既检查了参数的数量，又检查了参数的数据类型，避免了我们在调用时传递了错误的参数数量或数据类型。

以上简要讨论了在使用函数时应注意的问题，应该说编写一个好的函数，一个好的脚本程序是不容易的，需要读者朋友在以后的编程实践中去摸索，去总结，养成良好的编程习惯。

2.10 本章小结

本章介绍了 JavaScript 脚本语言的基本语法知识，包括数据类型、变量、运算符、核心语句以及函数等相关内容。其中，数据类型和运算符较为简单，通过本章的学习相信读者可以完全掌握；变量、核心语句和函数等知识，本章只作简要的介绍，在后续章节将加大介绍的力度。

本章还涉及到小部分与对象相关的知识，在后续章节中将进行深入的讲解。JavaScript 脚本是基于事件的程序开发语言，下一章将重点介绍“JavaScript 事件处理”的相关知识。

第 3 章 JavaScript 事件处理

用户可以通过多种方式与浏览器中的页面进行交互，而事件是交互的桥梁。Web 应用程序开发人员通过 JavaScript 脚本内置的和自定义的事件处理器来响应用户的动作，就可以开发出更具交互性、动态性的页面。

本章主要介绍 JavaScript 脚本中的事件处理的概念、方法，列出了 JavaScript 预定义的事件处理器，并且介绍了如何编写用户自定义的事件处理函数以及如何将它们与页面中用户的动作相关联，以得到预期的交互性能。同时讲述了 IE4 和 NN4 对基本事件模型的扩展，以及 DOM2 标准事件模型的架构等。

3.1 什么是事件

广义上讲，JavaScript 脚本中的事件是指用户载入目标页面直到该页面被关闭期间浏览器的动作及该页面对用户操作的响应。事件的复杂程度大不相同，简单的如鼠标的移动、当前页面的关闭、键盘的输入等，复杂的如拖曳页面图片或单击浮动菜单等。

事件处理器是与特定的文本和特定的事件相联系的 JavaScript 脚本代码，当该文本发生改变或者事件被触发时，浏览器执行该代码并进行相应的处理操作，而响应某个事件而进行的处理过程称为事件处理。

下面就是简单的事件触发和处理过程，如图 3.1 所示。

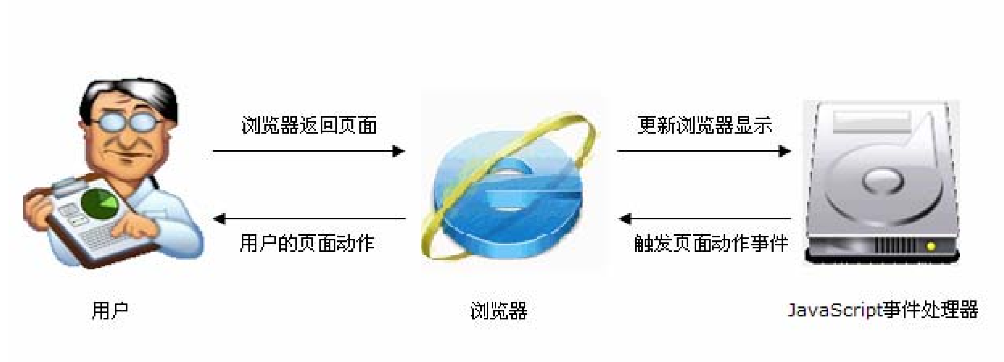

图 3.1 基本的用户动作触发事件示意图

如前所述，JavaScript 脚本中的事件并不限于用户的页面动作如 MouseMove、Click 等，还包括浏览器的状态改变，如在绝大多数浏览器获得支持的 Load、Resize 事件等。Load 事件在浏览器载入文档时触发，如果某事件（如启动定时器、提前加载图片等）要在文档载入时触发，一般都在<body>标记里面加入类似于“onload="MyFunction()"”的语句；Resize 事件则在用户改变了浏览器窗口的大小时触发，当用户改变窗口大小时，有时需改变文档页面的内容布局，使其以恰当、友好的方式显示给用户。

浏览器响应用户的动作，如鼠标单击按钮、链接等，并通过默认的系统事件与该动作相关联，但用户可以编写自己的脚本，来改变该动作的默认事件处理器。举个简单的例子，模拟用户单击页面链接的例子，该事件产生的默认操作是浏览器载入链接的 href 属性对应的

URL 地址所代表的页面，但利用 JavaScript 脚本可很容易编写另外的事件处理器来响应该单击鼠标的动作。考察如下代码：

```
<a name=MyA href="http://www.baidu.com/"
onclick="javascript:this.href='http://www.sina.com/'">MyLinker</a>
```

鼠标单击页面中名为“MyLinker”的文本链接，其默认操作是浏览器载入该链接的 href 属性对应的 URL 地址（本例中为“http://www.baidu.com/”）所代表的页面，但程序员编写了自定义的事件处理器即：

```
onclick="javascript:this.href='http://www.sina.com/'
```

通过该 JavaScript 脚本代码，上述事件处理器取代了浏览器默认的事件处理器，并将页面引导至 URL 地址为“http://www.sina.com/”指向的页面。

现代事件模型中引入 Event 对象，它包含其它对象使用的常量和方法的集合。当事件发生后，产生临时的 Event 对象实例，并附加当前事件的信息如鼠标定位、事件类型等，然后传递给相关的事件处理器进行处理。事件处理完毕后，该临时 Event 对象实例所占据的内存空间被清理出来，浏览器等待其他事件的出现并进行处理。如果短时间内发生的事件较多，浏览器按事件按发生的顺序将这些事件排序，然后按照该顺序依次执行。

事件发生的场合很多，包括浏览器本身的状态改变和页面中的按钮、链接、图片、层等。同时根据 DOM 模型，文本也可以作为对象，并响应相关动作，如鼠标双击、，文本被选择等。当然，事件的处理方法甚至于结果同浏览器环境有很大的关系，但总的来说，浏览器的版本越新，所支持的事件处理器就越多，支持也就越完善。基于此，在编写 JavaScript 脚本时，要充分考虑浏览器的兼容性，以编写适合大多数浏览器的安全脚本。

3.2 HTML 文档事件

HTML 文档事件包括用户载入目标页面直到该页面被关闭期间浏览器的动作及该页面对用户操作的响应，主要分为浏览器事件和 HTML 元素事件两大类。在了解这两类事件之前，先来了解事件捆绑的概念。

3.2.1 事件捆绑

HTML 文档将元素的常用事件（如 onclick、onmouseover 等）当作属性捆绑在 HTML 元素上，当该元素的特定事件发生时，对应于此特定事件的事件处理器就被执行，并将处理结果返回给浏览器。事件捆绑导致特定的代码放置在其所处对象的事件处理器中。考察如下代码：

```
<a href="http://www.baidu.com/" onclick="javascript:alert('You have Clicked the link!')">MyLinker
</a>
```

上述代码为“MyLinker”文本链接定义了一个 Click 事件的处理器，返回警告框“You have Clicked the link!”。

同样，也可将该事件处理器独立出来，编成单独的函数来实现同样的功能。将下列代码加入文档的<body>和</body>标记对之间：

```
<a href="http://www.baidu.com/" onclick="MyClick()">MyLinker</a>
```

自定义的函数 MyClick()实现如下：

```
function MyClick()
{
   alert("You have Clicked the link!");
```

```
}
```

鼠标单击“MyLinker”链接后，浏览器调用自定义的 Click 事件处理器，并将结果（警告框“You have Clicked the link!”）返回给浏览器。由事件处理器的实现形式来看，<a>标记的 onclick 事件与其 href 属性地位均等，实现了 HTML 中的事件捆绑策略。

3.2.2 浏览器事件

浏览器事件指载入文档直到该文档被关闭期间的浏览器事件，如浏览器载入文档事件 onload、关闭该文档事件 onunload、浏览器失去焦点事件 onblur、获得焦点事件 onfocus 等。先考察如下的代码：

```
//源程序 3.1
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script type="text/javascript">
<!--
window.onload = function ()
{
  var msg="\nwindow.load 事件 : \n\n";
  msg+="           浏览器载入了文档!";
  alert(msg);
}
window.onfocus = function ()
{
  var msg="\nwindow.onfocus 事件 :   \n\n";
  msg+="           浏览器取得了焦点!";
  alert(msg);
}
window.onblur = function ()
{
  var msg="\nwindow.onblur 事件 :   \n\n";
  msg+="           浏览器失去了焦点!";
  alert(msg);
}
window.onscroll = function ()
{
  var msg="\nwindow.onscroll 事件 :   \n\n";
  msg+="           用户拖动了滚动条!";
  alert(msg);
}
window.onresize = function ()
{
  var msg="\nwindow.onresize 事件 :   \n\n";
  msg+="           用户改变了窗口尺寸!";
  alert(msg);
}
//-->
</script>
```

```
</head>
<body>
<br>
  <p>载入文档:</p>
  <p>取得焦点:</p>
  <p>失去焦点:</p>
  <p>拖动滚动条:</p>
  <p>变换尺寸:</p>
</body>
</html>
```

将上述源程序保存为*.html（或*.htm）文档，双击该文档后系统调用默认的浏览器进行浏览。

当载入该文档时，触发 window.load 事件，弹出警告框如图 3.2 所示。

图 3.2 载入文档时触发 window.load 事件

当把焦点给该文档页面时，触发 window.onfocus 事件，弹出警告框如图 3.3 所示。

图 3.3 文档取得焦点时触发 window.onfocus 事件

当该页面失去焦点时，触发 window.blur 事件，弹出警告框如图 3.4 所示。

图 3.4 文档失去焦点时触发 window.onblur 事件

当用户拖动滚动条时，触发 window.onscroll 事件，弹出警告框如图 3.5 所示。

图 3.5 拖动滚动条，触发 window.onscroll 事件

当用户改变文档页面大小时，触发 window.onresize 事件，弹出警告框如图 3.6 所示。

图 3.6 改变文档页面大小，触发 window.onresize 事件

浏览器事件一般用于处理窗口定位、设置定时器或者根据用户喜好设定页面层次和内容等场合，在页面的交互性、动态性方面使用较为广泛。

注意：Netscape Navigator 4 支持 window.onmove 事件，该事件在当前浏览器窗口被用户移动时触发，主要用于窗口的定位方面。Internet Explorer 不支持 window.onmove 事件。

3.2.3 HTML 元素事件

页面载入后，用户与页面的交互主要指发生在如按钮、链接、表单、图片等 HTML 元素上的用户动作以及该页面对此动作所作出的响应。如简单的鼠标单击按钮事件，元素为 button，事件为 click，事件处理器为 onclick()。只要了解了该事件的相关信息，程序员就可以编写此接口的事件处理程序，也称事件处理器，以完成诸如表单验证、文本框内容选择等功能。

HTML 文档中元素对应的事件因元素类型而异，表 3.1 按 HTML4 标记类型和字母顺序列出了当前通用版本浏览器上支持的事件。其中的标记代表标记对，如<A>代表<A>和</A>标记对，其余类似。

表 3.1 通用浏览器上定义的事件

标记类型	标记列表	事件触发模型	简要说明

链接	<A>	onclick	鼠标单击链接
		ondbclick	鼠标双击链接
		onmouseDown	鼠标在链接的位置按下
		onmMouseOut	鼠标移出链接所在的位置
		onmouseOver	鼠标经过链接所在的位置
		onmouseUp	鼠标在链接的位置放开
		onKeyDown	键被按下
		onkeyPress	按下并放开该键
		onkeyUp	键被松开
图片	<IMG>	onerror	加载图片出现错误时触发
		onload	图片加载时触发
		onkeyDown	键被按下
		onkeyPress	按下并放开该键
		onkeyUp	键被松开
区域	<AREA>	ondbClick	双击该图形映射区域
		onmouseOut	鼠标从该图形映射区域内移动到该区域之外
		onmouseOver	鼠标从该图形映射区域外移动到区域之内
文档主体	<BODY>	onblur	文档正文失去焦点
		onclick	在文档正文中单击鼠标
		ondbClick	在文档正文中双击单击鼠标
		onkeyDown	在文档正文中键被按下
		onkeyPress	在文档正文中按下并放开该键
		onkeyUp	在文档正文中键被松开
		onmouseDown	在文档正文中鼠标按下
		onmouseUp	在文档正文中鼠标松开
帧、帧组	<FRAME> <FRAMESET>	onblur	当前窗口失去焦点
		onerror	装入窗口时发生错误
		onfocus	当前窗口获得焦点
		onload	载入窗口时触发
		onresize	窗口尺寸改变
		onunload	用户关闭当前窗口
窗体	<FORM>	onreset	窗体复位
		onsubmit	提交窗体里的表单
按钮	<INPUT TYPE= "button">	onblur	按钮失去焦点
		onclick	鼠标在按钮响应范围单击
		onfocus	按钮获得焦点
		onmouseDown	鼠标在按钮响应范围按下
		onmouseUp	鼠标在按钮响应范围按下后弹起
复选框 单选框	<INPUT TYPE= "checkbox"> or "radio"	onblur	复选框（或单选框）失去焦点
		onclick	鼠标单击复选框（或单选框）
		onfocus	复选框（或单选框）得到焦点
复位按钮 提交按钮	<INPUT TYPE= "reset"> or "submit"	onblur	复位（或确认）按钮失去焦点
		onclick	鼠标单击复位（或确认）按钮
		onfocus	复位（或确认）按钮得到焦点
口令字段	<INPUT TYPE= "password">	onblur	口令字段失去当前输入焦点
		onfocus	口令字段得到当前输入焦点
文本字段	<INPUT TYPE= "text">	onblur	文本框失去当前输入焦点
		onchange	文本框内容发生改变并且失去当前输入焦点
		onfocus	文本框得到当前输入焦点
		onselect	选择文本框中的文本
文本区	<TEXTAREA>	onblur	文本区失去当前输入焦点
		onchange	文本区内容发生改变并且失去当前输入焦点
		onfocus	文本区得到当前输入焦点
		onkeyDown	在文本区中键被按下
		onkeyPress	在文本区中按下并放开该键
		onkeyUp	在文本区中键被松开
		onselect	选择文本区中的文本

选项	<SELECT>	onblur	选择元素失去当前输入焦点
		onchange	选择元素内容发生改变且失去当前输入焦点
		onfocus	选择元素得到当前输入焦点

上表总结了 JavaScript 定义的通用浏览器事件，HTML 文档中事件捆绑特性决定了脚本程序员可以将这些事件当作目标的属性，在使用过程中只需修改其属性值即可。考察如下文本框各事件的测试代码：

```
//源程序 3.2
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyBlur()
{
  var msg="\n 文本框 onblur()事件 : \n\n";
  msg+="             文本框失去了当前输入焦点!";
  alert(msg);
}
function MyFocus()
{
  var msg="\n 文本框 onfocus()事件 : \n\n";
  msg+="             文本框获得了当前输入焦点!";
  alert(msg);
}
function MyChange()
{
  var msg="\n 文本框 onchange()事件 : \n\n";
  msg+="             文本框的内容发生了改变!";
  alert(msg);
}
function MySelect()
{
  var msg="\n 文本框 onselect()事件 : \n\n";
  msg+="             选择了文本框中的某段文本!";
  alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form name=MyForm>
  <input type=text name=MyText size=40
  value="Welcome to JavaScript world!"
  onblur="MyBlur()"
  onfocus="MyFocus()"
  onchange="MyChange()"
  onselect="MySelect()">
</form>
</center>
```

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

程序运行后，根据用户的页面动作触发不同的事件处理器（即对应的函数）。

鼠标点击文本框外的其他文档区域后，文本框失去当前输入焦点，触发 MyBlur()函数，返回警告框如图 3.7 所示。

图 3.7 文本框失去当前输入焦点

鼠标点击文本框后，文本框获得当前输入焦点，触发 MyFocus()函数，返回警告框如图 3.8 所示。

图 3.8 文本框获得当前输入焦点

修改文本框的文本后，鼠标在文本框外文档中任意位置点击，触发 MyBlur()函数的同时，触发 MyChange()函数，返回警告框如图 3.9 所示。

图 3.9 改变文本框内容并将焦点移出文本框

在文本框获得焦点后，用鼠标选择某段文本，触发函数 MySelect()函数，返回警告框如图 3.10 所示。

图 3.10 选择文本框中某段文本

HTML 元素事件在表单提交、在线办公、防止网站文章被复制、禁止下载网页中图片等方面应用十分广泛，主要是能有效识别用户的动作并做出相应的反应，如返回警告框、执行 window.close()方法关闭页面等操作。

通用浏览器上实现的诸多事件基本涵盖了页面中用户的动作，但随着 Web 技术的深入发展，出于友好、保密、版权等方面的考虑，通用浏览器上实现的事件已经不能满足 JavaScript 脚本开发人员的需求，各大浏览器厂商都更新了自己的事件模型，扩展了自身支持的事件类型，其中又以 IE 扩展的事件最为完备。

3.2.4 IE 扩展的事件

IE 浏览器从 IE4 版开始在更新其版本号的同时，逐步引入了一些事件用于捕捉更复杂的动作（如监视剪贴板等），主要用于文本、数据源操作等方面。表 3.2 列出了当前 IE 浏览器中广泛使用的一些事件：

表 3.2 IE浏览器的扩展事件

标记类型	标记列表	事件触发模型	简要说明
链接、图片、区域、地址、表单及其他文本属性标记	<A>、<ADDRESS>、<AREA>、<FORM>、<IMG>及其他属性标记*	onbeforecopy	在选择的内容复制和放在系统剪贴板之前触发
		onbeforecut	在选择的内容剪贴和放在系统剪贴板之前触发
		onbeforepaste	在选择的内容粘贴到文本之前触发
		oncopy	当选择的内容从文本复制到剪贴板时触发
		oncut	当选择的内容从文本中剪贴并添加到剪贴板时触发
		ondragstart	当用户拖动高亮选择时触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发*
		onpaste	当选择的内容粘贴到文本时触发
图片	<IMG>	onabort	由用户中断加载图片或者类似效果时触发
		onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		onreadystatechange	对象就绪状态发生改变时触发
		onresize	图片大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
		onscroll	当滚动标记重新定位时触发
Applet 文档主体	<APPLET>、<BODY>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforecut	在选择的内容剪贴和放在系统剪贴板之前触发
		onbeforepaste	在选择的内容粘贴到文本之前触发
		onbeforeupdate	在数据从标记转移到数据提供者之前触发
		oncut	当选择的内容从文本中剪贴并添加到剪贴板时触发
		ondragstart	当用户拖动高亮的选择时触发
		onpaste	当选择的内容粘贴到文本时触发

		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
		onscroll	当滚动标记重新定位时触发
Applet	<APPLET>、<OBJECT>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		ondataavailable	当数据从不同步传输数据的数据源到达时触发
		ondatasetchanged	当来自数据源初始数据可用或发生改变时触发
		ondatasetcomplete	指明数据源中所有数据为可用
		ondragstart	当用户拖动高亮的选择时触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发
		onreadystatechange	对象就绪状态发生改变时触发
		onresize	对象大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
文档主体帧组	<BODY>、<FRAMESET>	onafterprint	在用户打印当前或先前的文本时触发
		onbeforeprint	在用户打印当前或先前的文本之前触发
		onbeforeunload	在文本从窗口卸载之前触发
		ondragdrop	用户将一个对象拖动到浏览器窗口试图加载时触发
		ondragstart	当用户拖动高亮的选择时触发
		onerror	当文本的加载或者脚本执行出现错误时触发
		onmove	当用户移动窗口时触发
		onreadystatechange	对象就绪状态发生改变时触发
		onresize	对象大小发生改变时触发
按钮	Button	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforecut	在选择的内容剪贴和放在系统剪贴板之前触发
		onbeforepaste	在选择的内容粘贴到文本之前触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		oncut	当选择的内容从文本中剪贴并添加到剪贴板时触发
		ondragstart	当用户拖动高亮的选择时触发
		onpaste	当选择的内容粘贴到文本时触发
		onresize	对象大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
文本区	<TEXTAREA>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发*
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发
		onscroll	当滚动标记重新定位时触发
选项	<SELECT>	onafterupdate	在数据从标记转移到数据提供者完成后触发
		onbeforeupdate	当数据从标记转移到数据提供者之前触发
		ondragstart	当用户拖动高亮选择时触发
		onerrorupdate	数据传输由onbeforeupdate事件处理器取消时触发*
		onresize	对象大小发生改变时触发
		onrowenter	指明绑定的数据行已经发生改变，新数据可用
		onrowexit	在绑定数据源控制改变当前行之前触发

注意：（1）本表的*号依次表示：其他属性标记为<b>、<big>、<blockquote>、<caption>、<center>、<cite>、<code>、<dd>、<dfn>、<dir>、<div>、<dl>、<dt>、<em>、<h1>--<h6>、<i>、<li>、<listing>、<menu>、<ol>、<p>、<plaintext>、<pre>、<s>、<samp>、<small>、<span>、<strike>、<strong>、<sub>、<sup>、<td>、<textarea>、<th>、<tr>、<tt>、<u>、<ul>等；<AREA>、<ADDRESS>、<IMG>等文本属性标记无 onerrorupdate 事件。（2）如出现一个标记两个项目的情况，为两个项目共有的事件。（3）上述事件大部分在 IE4 中获得支持，在 IE5、IE5.5 中逐步完善，IE6+得到完美地支持。此 IE 包括以 IE 为内核的浏览器，不单纯指 Microsoft Internet Explorer。

表 3.2 中列举的事件为 HTML4 规则定义之外、Microsoft 为扩展其 IE 浏览器中动作捕

获能力而增加的 HTML 元素事件。由于 IE（包括 IE 核心）浏览器有着无与伦比的技术优势和覆盖范围，在编制 JavaScript 脚本程序的过程中上述事件原型得到了广泛的应用。

3.3 JavaScript 如何处理事件

尽管 HTML 事件属性可以将事件处理器绑定为文本的一部分，但其代码一般较为短小，功能较弱，只适用于只需做简单的数据验证、返回相关提示信息等场合。相比较对而言，使用 JavaScript 脚本可以更为方便处理各种事件，特别是 Internet Explorer、Netscape Navigator 等浏览器厂商在其第 4 代浏览器中推出更为先进的事件模型后，使用 JavaScript 脚本处理事件显得顺理成章。

JavaScript 脚本处理事件主要可通过匿名函数、显式声明、手工触发等方式进行，这几种方法在隔离 HTML 文本结构与逻辑关系的程度方面略为不同。

3.3.1 匿名函数

匿名函数的方式即使用 Function 对象（第 6 章详细叙述）构造匿名的函数，并将其方法复制给事件，此时该匿名的函数成为该事件的事件处理器。考察如下的代码：

```
//源程序 3.3
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body>
<center>
<br>
<p>单击“事件测试”按钮，通过匿名函数处理事件</p>
<form name=MyForm id=MyForm>
  <input type=button name=MyButton id=MyButton value="事件测试">
</form>
<script language="JavaScript" type="text/javascript">
<!--
document.MyForm.MyButton.onclick=new Function()
{
    alert("Your Have clicked me!");
}
-->
</script>
</center>
</body>
</html>
```

程序运行结果如图 3.11 所示。

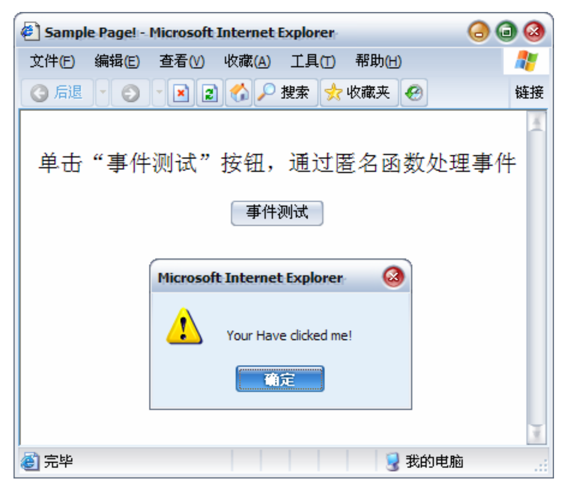

图 3.11 通过匿名函数处理事件

关键代码：

```
document.MyForm.MyButton.onclick=new Function()
{
    alert("Your Have clicked me!");
}
```

此句将名为 MyButton 的 button 元素的 click 动作的事件处理器设置为新生成的 Function 对象的匿名实例，即该匿名函数。鼠标单击该按钮后，响应“单击”事件，返回警告框。

3.3.2 显式声明

当然，设置事件处理器时，也可不使用以上匿名函数，而是将该事件的处理器设置为已经存在的函数。

考察如下图片翻转的实例：

```
//源程序 3.4
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyImageA()
{
    document.all.MyPic.src="2.jpg";
}
function MyImageB()
{
    document.all.MyPic.src="1.jpg";
}
-->
</script>
</head>
<body>
```

```
<center>
<p>在图片内外移动鼠标，图片轮换</p>
<img name="MyPic" id="MyPic" src="1.jpg" width=300 height=200></img>
<script language="JavaScript" type="text/javascript">
<!--
document.all.MyPic.onmouseover=MyImageA;
document.all.MyPic.onmouseout=MyImageB;
-->
</script>
</center>
</body>
</html>
```

程序运行结果如图 3.12 所示。

图 3.12 文档载入或鼠标移离图片时，显示默认图片“1.jpg”

当鼠标移动到图片区域时，图片发生变化，即显示图片“2.jpg”，如图 3.13 所示。

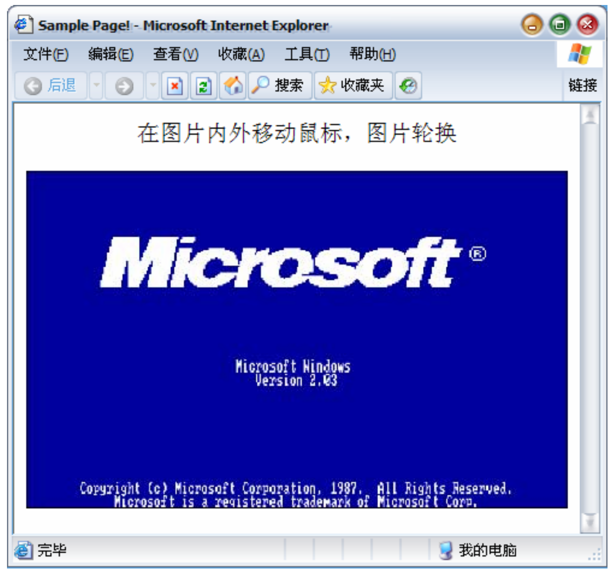

图 3.13 当鼠标移动到图片区域时显示图片“2.jpg”

当鼠标移离图片区域时，显示默认图片“1.jpg”，实现了图片的翻转。可将此法扩展为多幅新闻图片定时轮流播放的广告模式。

首先在<head>和</head>标记对之间嵌套 JavaScript 脚本定义两个函数：

```
function MyImageA()
{
    …
}
function MyImageB()
{
    …
}
```

然后通过 JavaScript 脚本代码将<IMG>标记元素的 mouseover 事件的处理器设置为已定义的函数 MyImageA()，将其 mouseout 事件的处理器设置为已定义的函数 MyImageB()：

```
document.all.MyPic.onmouseover=MyImageA;
document.all.MyPic.onmouseout=MyImageB;
```

由以上调用过程可以看出，通过显式声明的方式定义事件的处理器代码紧凑、可读性强，且对显式声明的函数没有任何限制，还可将该函数作为其他事件的处理器。较之匿名函数的方式实用。

3.3.3 手工触发

手工触发事件的原理相当简单，就是通过其他元素的方法来触发一个事件而不需要通过用户的动作来触发该事件。在源程序 3.4 的</script>和</center>标记之间插入如下代码：

```
<form name="MyForm" id="MyForm">
  <input type=button name=MyButton id=MyButton value="测试"
  onclick="document.all.MyPic.onmouseover();"
  onblur="document.all.MyPic.onmouseout();">
<form>
```

保存文件，程序运行后显示默认图片“1.pg”；单击“测试”按钮，将显示图片“2.jpg”，如图 3.13 所示；按钮失去焦点后，图片发生变化，显示图片“1.pg”，如图 3.12 所示。

如果某个对象的事件有其默认的处理器，此时再设置该事件的处理器时，将有可能出现意外的情况出现。考察如下的代码：

```
//源程序 3.5
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var msg="默认提交与其他提交方式返回不同的结果：\n\n";
  msg+="          点击"测试"按钮,直接提交表单.\n";
  msg+="          点击"确认"按钮,触发 onsubmit()方法,然后提交表单.\n";
  alert(msg);
```

```
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm1 id=MyForm1 onsubmit="MyTest()" method=post action="target.asp">
  <input type=button value="测试" onclick="document.all.MyForm1.submit();">
  <input type=submit value="确认">
</center>
</body>
</html>
```

程序运行后，单击“测试”按钮，触发表单的提交事件，并直接将表单提交给目标页面“target.asp”；单击表单默认触发提交事件的“确认”按钮，将弹出如图 3.14 所示的警告框，单击“确认”按钮后，将表单提交给目标页面“target.asp”。

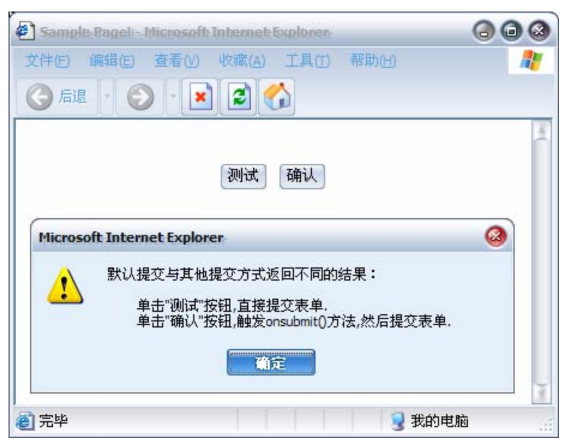

图 3.14 单击“确认”按钮后弹出警告框

由上面提交表单的例子可以看出，当事件在事实上已包含导致事件发生的方法时，该方法不会调用有问题的事件处理器，而会导致与该方法对应的行为立即发生。

> 注意：使用 JavaScript 脚本设置事件处理器时要分外小心，因为 JavaScript 事件处理器是大小写敏感的。设置目标对象中并不存在的事件的处理器将会给对象添加一个新的属性，而调用目标对象中并不存在的属性一般将导致页面运行错误。

3.4 事件处理器的返回值

事件通过给发送消息的方式来触发事件处理器对用户的动作做出相关响应来达到交互的目的，但此交互一般只是单方面的交互，即事件发送消息给事件处理器的过程，而不包括事件处理器将处理结果返回给事件的过程。事实上，事件处理器能将结果返回给事件，并由此影响事件的默认行为。考察如下代码：

```
//源程序 3.6
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
function ch_input()
{
  var msg="\n 系统提示信息 ：\n\n";
  if(document.all.MyForm.name.value=='')
  {
    msg+="           您输入的用户名为空,请重新填写并确认! \n";
    alert(msg);
    document.all.MyForm.name.focus();
    return false;
  }
  if(document.all.MyForm.password.value=='')
  {
    msg+="           您输入的密码为空,请重新填写并确认! \n";
    alert(msg);
    document.all.MyForm.password.focus()
    return false;
  }
}
</script>
</head>
<body>
<center>
<form name=MyForm method="post" action="target.asp"   onsubmit=return(ch_input())>
  用户:<input name="name" size="18" ><br>
  密码:<input type="password" name="password" size="19"><br>
  <input type="submit" value="提交" name="B1" >
  <input type="reset" value="重置" name="B2" ></p></form>
</center>
</body>
</html>
```

程序运行后，若表单的“用户”字段为空，鼠标单击“提交”按钮后，将弹出“错误”警告框如图 3.15 所示。

图 3.15 “用户”字段为空时弹出“错误”警告框

在该警告框中单击“确定”按钮，浏览器返回原始页面，并将“用户”字段设置为当前的输入焦点。

若表单的“用户”字段非空，则继续判断“密码”字段。同样，若表单的“密码”字段为空，鼠标单击“提交”按钮后，将弹出“错误”警告框如图 3.16 所示。

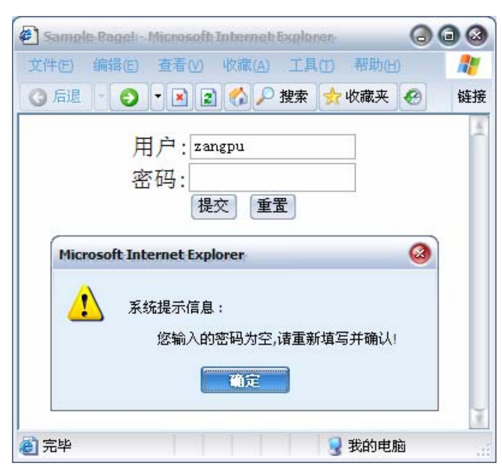

图 3.16 “密码”字段为空时弹出“错误”警告框

在该警告框中单击“确定”按钮，浏览器返回原始页面，并将“密码用户”字段设置为当前的输入焦点。

该段代码中，当客户端单击页面默认的表单 submit 按钮即“提交”按钮后，使用下列语句触发表单 submit 事件的处理器：

```
onsubmit=return(ch_input())
```

事件处理器所对应的函数 ch_input()根据页面表单各字段的情况判断：

```
if(document.all.MyForm.name.value=='')
{
  msg+="         您输入的用户名为空,请重新填写并确认! \n";
  alert(msg);
  document.all.MyForm.name.focus();
  return false;
}
```

判断各字段时，如果该字段为空，则弹出警告框，用户单击“确认”按钮后，浏览器返回设置该字段为当前的输入焦点，并将布尔值 false 作为结果返回给 submit 事件；如果各字段都不为空，则将布尔值 true 作为结果返回给 submit 事件。

submit 事件接受返回的结果，如果返回的结果为 true，浏览器将页面跳转到目标页面（本例中为 target.asp）；如果返回的结果为 false，则浏览器取消加载目标页面的动作，而继续将浏览器焦点定位在原始页面中。

在浏览器事件和 HTML 元素事件中，事件处理器的返回值将直接影响其下一步的动作。现将常用的事件返回值与其导致的行为之间关系列表如下：

表 3.3 常见事件与其返回值之间的关系

事件	返回值	对象	简要说明
Click	False	submit	取消表单提交
		reset	不重置表单
		radio、checkbox	单选框、复选框未被选择
		link	浏览器不执行页面跳转操作
DragDrop*	false	body、frameset	取消对象拖放

KeyDown	False	applet、font等	取消随后的KeyPress事件
KeyPress	False	applet、font等	取消KeyPress事件
MouseDown	False	applet、font等	取消鼠标的默认操作，如link的开始链接等
MouseOver	true	applet、font等	将窗口的status属性的变化在浏览器中表现出来
MouseOver	False	applet、font等	忽略窗口的status属性变化，直到MouseOut事件出现
Submit	False	submit	取消表单提交

注意：表 3.3 列举的是常见事件接受事件处理器返回非默认值时发生的动作，各个事件对应的对象请参见表 3.1 和表 3.2。

在实际应用中，经常在表单提交给服务器之前对其进行确认以检查通常的拼写错误或者非法的数据，极大减轻了服务器负担并充分保证其安全性，同时通过检查，提高用户提交数据的准确性。

3.5 事件处理器设置的灵活性

由于 HTML 将事件看成对象的属性，可以通过给该属性赋值的方式来改变事件的处理器，这给使用 JavaScript 脚本来设置事件处理器带来了很大的灵活性。考察如下的实例：

```
//源程序 3.7
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//设置事件处理器 MyHandlerA
function MyHandlerA()
{
  var msg="提示信息 : \n\n";
  msg+="          1、触发该按钮的 Click 事件的处理器 MyHandlerA()!\n";
  msg+="          2、改变该按钮的 Click 事件的处理器为 MyHandlerB()!\n";
  alert(msg);
  //修改按钮 value 属性
  document.all.MyForm.MyButton.value="测试按钮 : 触发事件处理器 B";
  //修改按钮的 Click 事件的处理器为 MyHandlerB
  document.all.MyForm.MyButton.onclick=MyHandlerB;
}
//设置事件处理器 MyHandlerB
function MyHandlerB()
{
  var msg="提示信息 : \n\n";
  msg+="          1、触发该按钮的 Click 事件的处理器 MyHandlerB()!\n";
  msg+="          2、改变该按钮的 Click 事件的处理器为 MyHandlerA()!\n";
  alert(msg);
  document.all.MyForm.MyButton.value="测试按钮 : 触发事件处理器 A";
  document.all.MyForm.MyButton.onclick=MyHandlerA;
}
-->
</script>
```

```
</head>
<body>
<center>
<form name="MyForm">
  <input type="button" name="MyButton" id="MyButton" value="测试按钮 ：触发事件处理器 A">
  <br>
</form>
</center>
<script language="JavaScript" type="text/javascript">
<!--
//设置按钮的 Click 事件的初始处理器为 MyHandlerA
document.all.MyForm.MyButton.onclick=MyHandlerA;
-->
</script>
</body>
</html>
```

程序运行后，单击“测试按钮 ：触发事件处理器 A”按钮，弹出警告框如图 3.17 所示。

图 3.17 第一次单击按钮，弹出警告框

在上述警告框中单击“确定”按钮后，返回原始页面，更改按钮的 value 属性为“测试按钮 ：触发事件处理器 B”。继续单击该按钮后，弹出警告框如图 3.18 所示。

图 3.18 第二次单击按钮，弹出警告框

在上述警告框中单击“确定”按钮后，返回原始页面，更改按钮的 value 属性为“测试按钮 ：触发事件处理器 B”，继续操作可以发现过程循环。

由程序结果可见，主要过程分为 4 步：

（1）文档载入后，通过属性赋值的方式将按钮的 Click 事件默认的事件处理器设置为 MyHandlerA：

```
document.all.MyForm.MyButton.onclick=MyHandlerA;
```

（2）单击按钮后，触发 Click 事件当前的事件处理器 MyhandlerA，后者返回提示信息并将按钮的 value 属性更改，同时将其 Click 事件当前的事件处理器设置为 MyhandlerB：

```
document.all.MyForm.MyButton.value="测试按钮 ：触发事件处理器 A";
document.all.MyForm.MyButton.onclick=MyHandlerA;
```

（3）在提示页面单击“确定”按钮返回原始页面后，再次单击按钮，触发 Click 事件当前的事件处理器 MyhandlerB，后者返回提示信息并将按钮的 value 属性更改，同时将其 Click 事件当前的事件处理器设置为 MyhandlerA：

```
document.all.MyForm.MyButton.value="测试按钮 ：触发事件处理器 B";
document.all.MyForm.MyButton.onclick=MyHandlerB;
```

（4）在提示页面单击“确定”按钮返回原始页面后，返回步骤（2）。

在 JavaScript 脚本中根据复杂的客户端环境及时更改事件的处理器，可大大提高了页面的交互能力。

值得注意的是，给对象的事件属性赋值为事件处理函数时，后者要省略函数后面的括号，且对象和函数要在显式赋值语句之前定义。

3.6 现代事件模型与 Event 对象

在 Internet Explorer 4（IE4，下同）和 Netscape Navigator 4（NN4，下同）浏览器版本发布之前，事件一直遵循基本事件模型标准，但后者存在着如下的诸多缺陷：

- 基本事件模型只支持最基本的事件，事件的类型和数目很有限，同时处理事件的方式也非常有局限性；
- 在基本事件模型中，某事件发生后触发事件处理器进行相关操作的过程中缺少必要的一些信息，如事件发生的位置、发生事件的对象等，事件处理器只能很被动地响应事件；
- 在基本事件模型中，没有任何途径解决对象继承关系中不同层次事件处理器之间的交互，对象的继承关系并没有反映到事件处理器的相互关系上。

基于此，在第 4 代浏览器版本中，各大浏览器厂商都扩展了其基本事件模型，添加了新的事件及处理方式。它们与基本事件模型之间最大的区别在于新的事件模型增加了 Event 对象，该对象将事件发生的快照内容如事件发生的位置、触发的按键等传递给事件处理器，事件处理器根据事件提供的这些信息进行相应的操作，从而极大扩展了 JavaScript 脚本的功能。

同时，新的事件模型中的事件可通过文本的继承关系进行事件传播，但各种浏览器中定义的事件流又不大不同，如 IE4 和 NN4 事件流的方向就完全相反：

- 在 IE4 中多数事件是由它们发生的地方开始向上回溯继承关系，上溯的事件在继承关系的每层都用合适的事件处理器来处理、改向或者将事件沿着关系树传递，当然，某些定义清晰的事件如表单提交、按钮接收焦点等不会沿着继承关系上溯；
- 在 NN4 中事件从顶端对象开始至末端对象结束，并使得这些对象能够更改、处理或取消该事件，在较高层次上的事件处理器比低层次的事件处理器有更多机会处理事件。

图 3.19 显示了 IE4 和 NN4 新事件模型中事件流的不同：

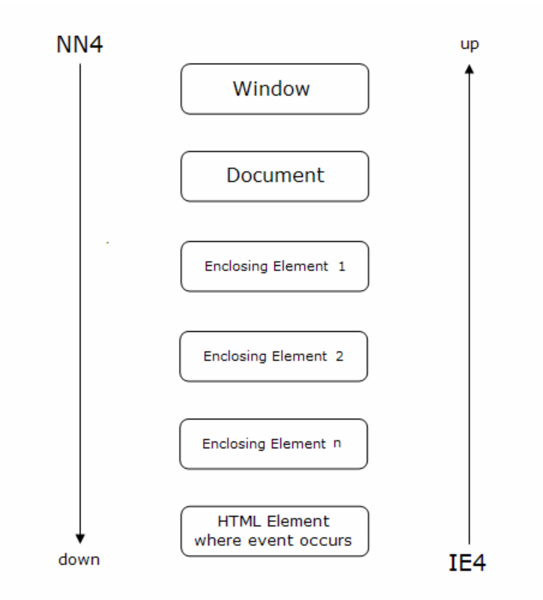

图 3.19 IE4 和 NN4 中新事件模型的事件传播方向

除了事件传播方向上的不同外，IE4 和 NN4 中定义的 Event 对象也有很大的差异，主要表现在对象实例的传递、作用范围等方面。

JavaScript1.2 版本中引入 Event 对象作为事件与事件处理器之间传递信息的桥梁，这些信息如事件发生的位置、触发事件的原始对象等通过 Event 对象的属性提供。IE4 和 NN4 通过不同途径提供 Event 对象的诸多属性供脚本开发人员调用。

3.7 IE4 中的 Event 对象

由于 IE4 中文档的每个元素都作为一个对象而存在，增加了事件发生的概率和可用的事件数目。当事件发生的时候，浏览器创建全局的 Event 对象，并使它对于合适的事件处理器可用。

3.7.1 对象属性

IE4 中的 Event 对象提供非常丰富的属性供脚本程序员调用，如事件发生的原始对象、相对位置等，其常见属性及其功能介绍如表 3.4 所示。

表 3.4 IE4 中Event对象的常见属性

属性	简要说明
altkey、crlkey、shiftkey	设置为true或者false，表示事件发生时是否按下Alt、Ctrl和Shift键
buttoon	发生事件时鼠标所按下的键（0表示无，1表示左键，2表示右键，4表示中键）
cancelBubble	设置为true或者false，表示是取消还是启用事件上溯
clientX、clientY	光标相对于事件所在Web页面的水平和垂直位置（像素）
fromElement、toElement	指向在MouseOver或MouseOut中鼠标要移开和移向的标记
keyCode	表示所按键的Unicode键盘内码
offsetX、offsetY	光标相对于事件所在容器的水平和垂直位置（像素）
reason	表示数据源对象的数据传输状态
returnValue	设置为true或false，表示事件处理器的返回值

screenX、screenY	光标相对于屏幕的水平和垂直位置（像素）
srcElement	表示发生事件的原始对象
srcFilter	指定产生onfilterchange事件的filter对象
type	指明发生事件的类型
x、y	光标相对于事件所在文档的水平和垂直位置（像素）

理解了事件与其处理器之间交互时 Event 对象携带的信息，以及其全局特性，很容易通过调用对象属性的方式根据需要获取事件发生的诸多信息。考察如下获取事件发生相对位置等信息的代码：

```
//源程序 3.8
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="Click 事件发生后创建的 Event 对象信息:\n\n";
function MyAlert()
{
  msg+="发生事件的类型  :\n";
  msg+="          type = " +event.type+ "\n\n";
  msg+="发生事件的原始对象  :\n";
  msg+="          target = " +event.srcElement+ "\n\n";
  msg+="光标相对于事件所在文档的水平和垂直位置(像素) :\n";
  msg+="          x = " +event.x+ "          y = " +event.y+ "\n\n";
  msg+="光标相对于事件所在容器的水平和垂直位置(像素) :\n";
  msg+="          x = " +event.offsetX+ "          y = " +event.offsetY+ "\n\n";
  msg+="光标相对于事件所在屏幕的水平和垂直位置(像素) :\n";
  msg+="          x = " +event.screenX+ "          y = " +event.screenY+ "\n\n";
  msg+="光标相对于事件所在 Web 页面的水平和垂直位置(像素) :          \n";
  msg+="          x = " +event.clientX+ "          y = " +event.clientY+ "\n";
  alert(msg);
}
-->
</script>
</head>
<body onclick="MyAlert();">
  <table>
    <tr>
      <td>
        <p>
          Click the
          <em>
            EM text
          </em>
          to Test!
        </p>
      <td>
    <tr>
  </table>
</body>
```

```
</html>
```

程序运行后，鼠标单击文档中文本段的任何一个位置，弹出包含当前发生事件信息的警告框如图 3.20 所示。

图 3.20 实例中 Click 事件的相关信息警告框

在 IE4 中，任何事件发生后生成的 Event 对象对该文档而言是透明的，可将其看成是全局变量使用，即在文档任何地方都不需通过传入参数的方式来调用。该变量生成于其对应的事件发生之时，且在文档被浏览器关闭时对象所占据的内存空间被释放出来。

3.7.2 事件上溯

IE4 中的大部分事件会沿着关系树上溯（Bubble，也称冒泡），在继承关系的每一层如果存在合适的事件处理器则调用，不存在则继续上溯，直至上升到关系树的顶端或者被某个层所取消，但不支持上溯的事件仅能调用当前事件发生的原始对象那个层次上可用的事件处理器。表 3.5 列出了 IE4 中常用的事件，以及该事件能否上溯、能否取消等特性。

表 3.5 IE4 中常用的事件及回溯、取消情况

事件	能否上溯	能否取消	事件	能否上溯	能否取消
abort	否	是	keypress	是	是
afterupdate	是	否	keyup	是	否
beforeunload	否	是	load	否	否
beforeupdate	是	是	mousedown	是	是
blur	否	否	mousemove	是	否
bounce	否	是	mouseout	是	否
change	否	是	mouseover	是	是
click	是	是	mouseup	是	是
dataavailable	是	否	readystatechange	否	否
datasetchanged	是	否	reset	否	是
datasetcomplete	是	否	resize	否	否
dbclick	是	是	rowenter	是	否
dragstart	是	是	rowexit	否	是
error	否	是	scroll	否	否
errorupdate	是	否	select	否	是
filterchange	否	否	selectstart	是	是
finish	否	是	start	否	否
focus	否	否	submit	否	是
help	是	是	unload	否	否
keydown	是	是			

注意：在上表中，事件对应的处理器一般为事件首字母改写为小写，然后前面加上“on”即可，如 Dbclick 事件其对应的事件处理器为 ondbclick。文档中各元素支持何种事件，请参阅表 3.1 和表 3.2。

某事件不能上溯，表明该事件只对当前发生事件的对象有效；某事件能取消表明可以通过设置其对应 Event 对象的 cancelBuuble 属性为 true 来阻止事件上溯到其上层对象。

考察如下演示 IE4（IE4 以上版本都适用）中事件上溯的代码：

```
//源程序 3.9
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="事件上溯结果:\n\n";
msg+="Click 事件开始:\n\n";
function MyAlert()
{
  msg+="Click 事件结束:\n";
  alert(msg);
}
-->
</script>
</head>
<body onclick="javascript:msg+='-->事件定位于 Body,转向下面事件\n\n';MyAlert();">
<br>
<center>
  <table onclick="javascript:msg+='-->事件定位于 Table,转向下面事件\n\n'">
    <tr onclick="javascript:msg+='-->事件定位于 Tr,转向下面事件\n\n'">
      <td onclick="javascript:msg+='-->事件定位于 Td,转向下面事件\n\n'">
        <p onclick="javascript:msg+='-->事件定位于 p,转向下面事件\n\n'">
          Click the
          //事件发生的原始对象
          <em onclick="javascript:msg+='-->事件定位于 em,转向下面事件\n\n'">
            EM text
          </em>
          to Test!
        </p>
      <td>
    <tr>
  </table>
</center>
</body>
</html>
```

程序运行后，鼠标点击页面中使用和标记对的“EM text”字符串，弹出对话框如图 3.21 所示。

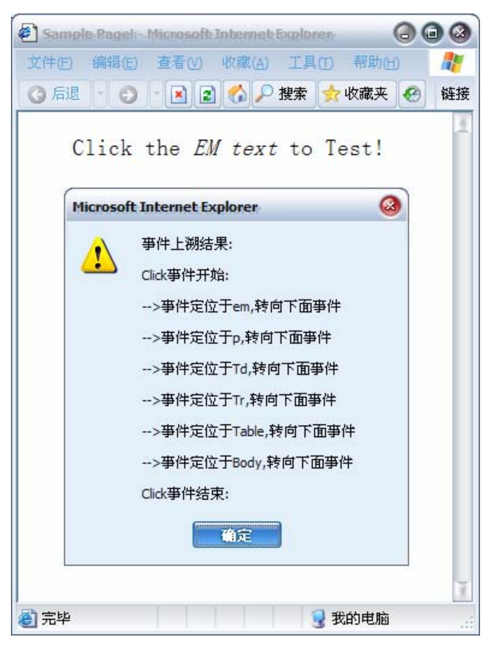

图 3.21 IE4 中事件的回溯方向实例

可以清楚看出 Click 事件由<em>标记处开始触发，然后按标记的层次顺序上溯至<p>、<td>、<tr>、<table>和<body>，直至 Document 对象，但并不上溯到 Window 对象。

3.7.3 阻止事件上溯

IE4 中的事件严格按照文档中元素对象的层次关系上溯而不管实际应用中是否需要将该事件上溯到对象关系中特定的层次。为达到在对象关系中指定层次中断事件上溯的目的，程序员可设置 Event 对象的 cancleBubble 属性为 true 来实现。

考察如下代码：

```
//源程序 3.10
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="阻止事件上溯结果:\n\n";
msg+="Click 事件开始:\n\n";
function MyAlert()
{
  msg+="Click 事件结束:\n";
  alert(msg);
}
-->
</script>
</head>
<body onclick="javascript:msg+='-->事件定位于 Body,转向下面事件\n\n'">
```

```
<center>
<table onclick="javascript:msg+='-->事件定位于 Table,转向下面事件\n\n'">
  <tr onclick="javascript:msg+='-->事件定位于 Tr,事件停止上溯\n\n';event.cancelBubble=true;">
    <td onclick="javascript:msg+='-->事件定位于 Td,转向下面事件\n\n'">
      <p onclick="javascript:msg+='-->事件定位于 p,转向下面事件\n\n'">
        Click the
        <em onclick="javascript:msg+='-->事件定位于 em,转向下面事件\n\n'">
          EM text
        </em>
        to Test!
      </p>
    <td>
  <tr>
</table>
<form>
  测试方法:<br>
      1、点击文本中斜体字符串;<br>
      2、鼠标单击"测试"按钮.<br>
  <input type=button value="测试" onclick="MyAlert()">
</form>
<center>
</body>
</html>
```

程序运行后，单击页面中的“EM text”文本段，然后单击“测试”按钮，弹出对话框如图 3.22 所示。

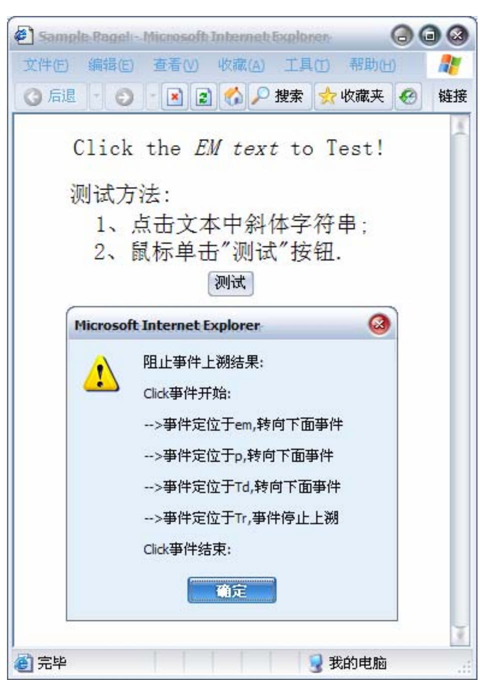

图 3.22 使用 Event 对象的 cancelBubble 属性阻止事件上溯

可以看出，Click 事件在由<em>、<p>、<td>标记对象上溯到<tr>后，上溯过程中止，而不像源程序 3.9 的运行结果那样，事件从<em>标记对象开始上溯到<body>为止。

关键语句：

```
<tr onclick="javascript:msg+='-->事件定位于 Tr,事件停止上溯\n\n';event.cancelBubble=true;">
```

当 Click 事件上溯到<tr>标记后，执行当前的事件处理器 onclick()，后者首先更新输出信息，然后设置 Event 对象的 cancelBubble 属性为 true：

```
event.cancelBubble=true;
```

程序结果显示，在对象模型某层次设置 Event 对象的 cancelBubble 属性后，事件的上溯过程被中断，达到阻止事件上溯的目的。

3.7.4 事件改向

IE4 中事件严格按照文档中元素对象的层次关系上溯的特性，事件流的方向人为很难掌握，给程序员精确控制事件的流向带来不小的难度。IE5.5 引进了了对象的 fireEvent()方法，该方法可以将当前事件传递到某个特定的对象上。语法如下：

```
object.fireEvent(arg1,arg2);
```

该方法需要给定两个参数 arg1 和 arg2，其中参数 arg1 表示目标对象的事件处理器，参数 arg2 表示当前事件。

考察如下演示事件转向的代码：

```
//源程序 3.11
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="事件转向实例:\n\n";
msg+="Click 事件开始:\n\n";
function ChangeDir()
{
  msg+="-->事件定位于 Td,准备事件转向\n\n";
  event.cancelBubble=true;
  document.body.fireEvent("onclick",event);
}
function MyAlert()
{
  msg+="Click 事件结束:\n";
  alert(msg);
}
-->
</script>
</head>
<body onclick="javascript:msg+='-->事件定位于 Body,转向下面事件\n\n'">
<center>
<table onclick="javascript:msg+='-->事件定位于 Table,转向下面事件\n\n'">
  <tr onclick="javascript:msg+='-->事件定位于 Tr,转向下面事件\n\n'">
    <td onclick="ChangeDir()">
      <p onclick="javascript:msg+='-->事件定位于 p,转向下面事件\n\n'">
        Click the
        <em onclick="javascript:msg+='-->事件定位于 em,转向下面事件\n\n'">
          EM text
```

```
            </em>
            to Test!
          </p>
        <td>
    <tr>
</table>
<form>
    测试方法:<br>
        1、点击文本中斜体字符串;<br>
        2、鼠标单击"测试"按钮.<br>
    <input type=button value="测试" onclick="MyAlert()">
</form>
<center>
</body>
</html>
```

程序运行后，点击页面中的“EM text”文本段，然后单击“测试”按钮，弹出对话框如图 3.23 所示。

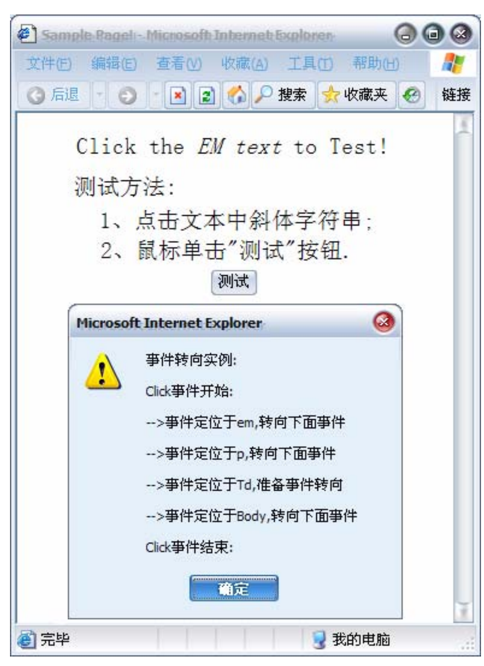

图 3.23 IE4 中事件改向实例

由程序运行结果可看出，Click 事件由<em>、<p>标记上溯到<td>标记之后，直接改向到<body>标记。关键代码：

```
<td onclick="ChangeDir()">
```

此句将<td>标记的 Click 事件的处理器设置为事件处理器函数 ChangeDir()：

```
function ChangeDir()
{
    msg+="-->事件定位于 Td,准备事件转向\n\n";
    event.cancelBubble=true;
    document.body.fireEvent("onclick",event);
}
```

该函数首先通过设置 Event 对象的 cancelBubble 属性为 true 将 Click 事件的上溯过程中止，同时通过<body>标记的 fireEvent()方法将 Click 事件改向自身，并触发<body>标记的

onclick 事件处理器更新全局变量 msg。单击“测试”按钮，输出相关提示信息。

IE4 中的事件模型为很经典的模型架构，通过其提供的诸多属性，脚本程序员很容易把握其事件的触发机制，编制出高质量的事件控制脚本。

> 注意：事件上溯、阻止事件上溯、事件改向等特性在目前的 IE 浏览器（包括以其为核心的浏览器）版本中仍然适用。

3.8 NN4 中的 Event 对象

与 IE4 相同，NN4 中发生事件时，浏览器创建 Event 对象，并将其传递给事件处理器使用，但两者创建的 Event 对象在很多方面存在很大的差异，如对象实例的传递、作用范围、事件的流向等。

3.8.1 对象属性

较之 IE4 而言，NN4 中的 Event 对象提供较为简单的属性供程序员使用，如事件发生的原始对象、相对位置等，其常见属性及其功能介绍如表 3.6 所示。

表 3.6 NN4 中Event对象的常见属性

属性	简要说明
data	包含DragDrop事件放置的对象URL字符串的数组
height、width	窗口（或帧）的高度和宽度
layerX、layerY	光标相对于事件所在层的水平和垂直位置（像素）
modifiers	与鼠标或按键相关联的组合键（ALT_MASK,CONTROL_MASK,META_MASK, SHIFT_MASK等），其中META_MASK为Macintosh机上的Command键，下同
pageX、pageY	光标相对于页面的水平和垂直位置（像素）
target	发生事件的原始对象
type	指明发生事件的类型
which	发生事件时鼠标所按下的键（1表示左，2表示中，3表示右）或所按键的ASCII值

NN4 中创建的 Event 对象并不像 IE4 中一样在文档整个范围内透明，可以当成全局变量直接调用，在触发事件处理器的过程中不需要将 Event 对象显式传递给事件处理器。在 NN4 中创建的 Event 对象只在其对应的事件处理器内可见，必须将其显式传递给事件处理器。考察如下代码：

```
//源程序 3.12
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="";
msg+="Click 事件发生后创建的 Event 对象信息:\n\n";
function MyAlert(event)
{
  msg+="发生事件的类型  :\n";
  msg+="            type = " +event.type+ "\n\n";
```

```
    msg+="发生事件的原始对象 :\n";
    msg+="              target = " +event.target+ "\n\n";
    msg+="发生事件时鼠标所按下的键(1:左,2:中,3:右) :\n";
    msg+="              which = " +event.which+ "\n\n";
    msg+="光标相对于事件所在文档的水平和垂直位置(像素) :\n";
    msg+="              x = " +event.layerX+ "              y = " +event.layerY+ "\n\n";
    msg+="光标相对于页面的水平和垂直位置(像素) :\n";
    msg+="              x = " +event.pageX+ "              y = " +event.pageY+ "\n\n";
    alert(msg);
}
-->
</script>
</head>
<body onclick="MyAlert(event);">
    <table>
        <tr>
            <td>
                <p>
                    Click the
                    <em>
                        EM text
                    </em>
                    to Test!
                </p>
            <td>
        <tr>
    </table>
</body>
</html>
```

程序运行后，鼠标单击文档中文本段的任何一个位置，弹出包含当前发生事件信息的警告框如图 3.24 所示。

图 3.24 Netscape 中通过 Event 对象返回的事件信息

如果在触发事件处理器的过程中，仍然像 IE4 中一样，不显式传递 Event 对象，则事件处理器不能正常工作。

3.8.2 事件捕获

NN4 中没有事件上溯的概念，与此功能相类似的是事件捕获。事件捕获使 window、document、layer 对象能捕获窗口、文档和层中低层次对象的事件。如 document 对象能捕获发生在文档页面中的事件如鼠标单击某文本段等，并将此事件交给该层的事件处理器处理。事件捕获对于处理事件或者在更高的层次上定义事件处理器来取代低层次的多个事件处理器是非常有用的，典型的如表单的提交等。

NN4 中的对象提供 captureEvents()方法实现事件捕获，如下列语句实现捕获 layer 中鼠标双击事件功能：

```
layer.captureEvents(Event.DBCLICK);
```

其中 Event.DBCLICK 参数为 NN4（及 NN4 以上版本）特有的关键字常量，其指明事件的类型，同时，NN4 给出了几个常用的辅助键关键字如代表 Shift 按键的 SHIFT_MASK 等。表 3.7 列出了 NN4 中常见的关键字常量：

表 3.7 NN4 中常见的关键字常量

类型	关键字			
事件型	ABORT	BLUR	CHANGE	CLICK
	DBCLICK	DRAGDROP	ERROR	FOCUS
	KEYDOWN	KEYPRESS	KEYUP	LOAD
	MOUSEDOWN	MOUSEMOVE	MOUSEUP	MOUSEOUT
	MOUSEOVER	MOVE	RESET	RESIZE
	SELECT	SUBMIT	UNLOAD	
按键型	ALT_MASK	CONTROL_MASK	META_MASK	SHIFT_MASK

要识别特定的事件类型，可通过 Event.Name 的形式调用，Name 为上表中的关键字。考察如下捕获辅助键是否按下事件的代码（仅适用于 NN4）：

```
//源程序 3.13
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//设置事件处理器
function MyHandler(MyEvent)
{
  var msg="";
  msg+="\n 键盘辅助键识别结果 : \n\n";
  if(MyEvent.modifiers&Event.ALT_MASK)
    msg+="        事件触发时按住了 ALT 键!\n";
  if(MyEvent.modifiers&Event.CONTROL_MASK)
    msg+="        事件触发时按住了 CTRL 键!\n";
  if(MyEvent.modifiers&Event.META_MASK)
    msg+="        事件触发时按住了 Command 键!\n";
  if(MyEvent.modifiers&Event.SHIFT_MASK)
    msg+="        事件触发时按住了 SHIFT 键!\n";
  alert(msg);
}
-->
```

```
</script>
</head>
<body>
<center>
  <p>按住 ALT\CTRL\SHIFT 然后点击页面，返回识别结果！</a>
</center>
<script language="JavaScript" type="text/javascript">
<!--
//文档对象捕获事件
document.captureEvents(Event.MOUSEDOWN);
document.onmousedown=MyHandler;
-->
</script>
</body>
</html>
```

程序运行后，在文档任意地方单击鼠标，如果单击的同时按住了 Shift 和 Ctrl 键，则弹出警告框如图 3.25 所示。

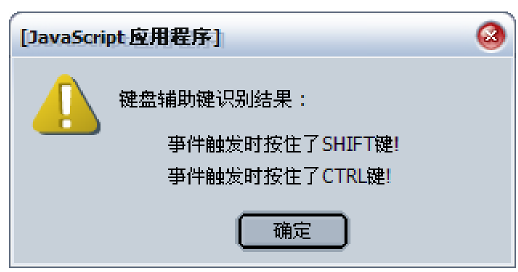

图 3.25 辅助键识别结果

关键语句：

```
document.captureEvents(Event.MOUSEDOWN);
document.onmousedown=MyHandler;
```

第一句中文档对象捕获 MouseDown 事件，第二句将 MouseDown 事件连接到其对应的事件处理器即 MyHandler()函数上面。

如果需要捕捉多个事件，各事件之间要用管道符“|”隔开。如捕获文档中所有的 Click 和 DbClick 事件可使用如下语句：

```
document.captureEvents(Event.CLICK|Event.DBCLICK);
```

3.8.3 关闭事件捕获

IE4 中通过设置 Event 对象的 cancleBubble 属性为 true 来阻止事件的上溯，而 NN4 中则通过内置的 releaseEvent()方法来将某层次上的特定事件设置为不捕获。该方法语法如下：

```
object.releaseEvent(arg);
```

其中 arg 为要设置为不捕获的目标事件列表，在表 3.7 中取值。当需要设置为不捕获的事件有很多时，各事件之间要用管道符“|”隔开。假如要将 document 层次上的所有 Click 事件和 DbClick 事件设置为不捕获，可用下列语句：

```
document.releaseEvents(Event.CLICK|Event.DBCLICK);
```

关闭事件捕获一般用于设置某种类型的事件在对象层次关系中的特定层中不需进行特别关注的场合。

3.8.4 事件传递

前面已经了解，NN4 中事件流的流向与 IE4 中正好相反，是从对象继承关系中较高层次的对象如 window、document 到较低层次的对象如标记<p>、<b>等。同时较高层次上的事件处理器往往比较低层次上的处理器触发的机会更大。

如果在实际应用中，某事件不需要在顶级对象如 document 对象层次上调用事件处理器进行处理，而只需在<table>标记的层次进行处理，此时就需要将该事件往下传递。与 IE4 中使用 fireEvent()方法改向不同，NN4 中采用 Event 对象的 routeEvent()方法将事件往下传递。该方法的语法如下：

```
routeEvent(event);
```

该方法接受一个参数 event，表示当前发生事件。该方法提供了一种在对象关系树中控制事件流向的途径，同时新的事件处理器处理完该事件后，能将处理结果返回给原始对象，后者可根据该结果更改本层的事件处理器的行为。将下列脚本加入文档的<head>和</head>标记对之间：

```
<script language="JavaScript" type="text/javascript">
<!--
function MyHandler(event)
{
  if(event.modifiers&Event.SHIFT_MASK)
    alert("鼠标单击且按住了 Shift 键");
  else
    routeEvent(event);
}
-->
</script>
```

然后将事件捕获的代码加入文档的</body>标记之前：

```
<script language="JavaScript" type="text/javascript">
<!--
window.captureEvent(Event_DBCLICK);
window.ondbclick=MyHandler;
-->
</script>
```

该实例运行后，在窗口中双击，如果此时也按下了组合键 Shift，则弹出警告框“鼠标单击且按住了 Shift 键”，否则，事件转向 window 对象所在层的下一层，即 document 对象，并将该事件通过参数的形式传到 document 对象的 DbClick 事件处理器中进行处理。

3.9 DOM 的解决之道

以 IE4 和 NN4 为代表的各大浏览器对基本事件模型的扩展都不同，给 Web 应用程序开发人员在脚本兼容性方面带来了很大的挑战，为统一标准，W3C 面向业界及时推出了一个标准的事件模型 DOM2（Document Object Model Level 2，简称 DOM2，下同），该模型在有机结合 IE4 和 NN4 事件模型的基础上，扩展了 DOM Level 1，同时加入了新的事件。DOM2 给脚本开发人员提供了一个功能十分强大的事件处理环境，并逐步获得了主流浏览器的良好支持。

注意：DOM2 最先获得支持的浏览器是 NN6，在 Microsoft 的 IE 中 DOM2 没有获得完善的支持，而是

在其最新浏览器版本中开发了类似的功能。

DOM2 提供标准的方法来访问文档中已结构化的各种对象，在 IE4 和 NN4 的基础上尽量缩小各浏览器事件模型之间的差异。并提出“事件处理器”、“UI 事件”等新概念。

3.9.1 事件流方向

在事件流的方向问题上，DOM2 综合了 IE4 和 NN4 的事件回溯和下传的做法：

- 事件下传阶段：该过程模仿 NN4 中事件流的流动过程，事件从继承关系的顶端开始搜索，直到到达目标对象为止。事件在下传过程中，可被当前的事件处理器先处理或者改向。一旦到达目标对象，该事件的处理器立即被触发。
- 事件上溯过程：事件到达目标对象之后，将模仿 IE4 中事件流的流动过程，沿着继承关系上溯至顶端，在上溯的过程中调用每一层相应的事件处理器。

事件上溯过程并不是必须的，只有当程序员在该事件下传过程中以 NN4 的方式显示捕获过此事件时，事件上溯过程才发生。

在事件下传和上溯过程中，DOM2 引入了“事件监听器”概念对事件处理器和对象进行捆绑。事件监听器本质上也是一种事件处理器，只不过它被绑定在对象继承关系的特定节点上，并可在事件生命周期的特定阶段被触发。DOM2 使用节点的 addEventListener()方法和 deleteEventListener()方法分别对事件监听器进行绑定和删除。

在事件的流动过程中，DOM2 提供节点的 dispatchEvent()方法来将当前事件改向，该方法以当前 Event 对象为参数。目标节点使用该方法后，变为新的事件节点，事件将按照正常的继承关系传递到其他的目标。

3.9.2 Event 对象

在综合 IE4 和 NN4 事件流的流动方法之外，DOM2 定义了 Event 对象的一些属性，其中常见的属性如表 3.8 所示。

表 3.8 DOM2 中Event对象的常见属性

属性	简要说明
altKey、crlKey、shiftKey、metaKey	布尔值，表示事件发生时是否按下Alt、Ctrl、Shift和Meta键
Bubbles	布尔值，指明事件是否上溯
Button	指明发生事件时鼠标所按下的键（1：左键，2：中键，3：右键）
Cancellable	指明事件是否取消
charCode	在KeyPress事件中按下的键的ASCII码
clientX、clientY	事件发生位置相对于当前Web页面的水平和垂直位置（像素）
currentTarget	事件下传和上溯过程中当前的节点
eventPhase	指明当前事件位于事件流的阶段（1：下传，2：目标，3：上溯）
keyCode	在KeyDown和KeyUp事件中所按键的ASCII码
relatedTarget	引用与事件相关的节点
screenX、screenY	光标相对于屏幕的水平和垂直位置（像素）
Target	表示发生事件的原始节点
Type	指明发生事件的类型

目标事件的属性与事件的类型紧密相关，并不是每个事件都能拥有 Event 对象的全部属性。如键盘 KeyPress 事件中就不可能拥有 button 属性，鼠标的 Click 事件就不拥有 keyCode 属性等。

3.9.3 事件类型

DOM2 基本融合了 IE4 和 NN4 中的鼠标事件、键盘事件、浏览器事件和文档事件，表 3.9 列出了 DOM2 中定义的常用事件以及能否上溯、取消等特性：

表 3.9 DOM2 中定义的基本事件及其特性

事件	能否上溯	能否取消	事件	能否上溯	能否取消
abort	是	否	mouseout	是	是
blur	否	否	mouseover	是	是
change	是	否	mouseup	是	是
click	是	是	reset	是	否
error	是	否	resize	是	否
focus	否	否	scroll	是	否
load	否	否	select	是	否
mousedown	是	是	submit	是	是
mousemove	是	否	unload	否	否

由表 3.5 和表 3.9 可以看出，DOM2 综合了 IE4 和 NN4 中种类繁多的事件，将它们融合为最基本的事件类型，如 DOM2 中的鼠标单击事件综合为 click 事件，简化了事件模型。同时事件的上溯和取消等特性也作了一定的调整。

除此之外，DOM2 还引进了“UI 事件”的概念。UI（用户界面）事件使用“DOM”作前缀，以区别于普通的事件。表 3.10 列举了几种 UI 事件及其能否上溯和取消的特性：

表 3.10 DOM2 中UI事件及其特性

事件	简要说明	能否上溯	能否取消
DOMActive	当对象激活（如鼠标单击按钮）时触发	是	是
DOMFocusIn	元素（广义的，不单指表单）获得焦点	是	否
DOMFocusOut	元素（广义的，不单指表单）失去焦点	是	否

在实际使用中，一般使用 DOM2 中定义的基本事件，而很少使用 UI 事件，其更多的是提供一种用户界面事件的模型。

上面几节简要介绍了基本事件模型、IE4 和 NN4 对基本模型的扩展以及 DOM2 标准事件模型。脚本程序员在决定使用哪个模型之前，要充分了解其目标客户使用的浏览器信息。浏览器发展的总趋势是向互相兼容的方向，也不排除短期内各浏览器厂商各自扩展标准事件模型而朝向不同方向发展的可能。总的原则是在 DOM2 标准事件模型基础上，尽量使用事件模型中兼容性较强的事件进行相关操作。

3.10 本章小结

本章主要介绍了 JavaScript 事件处理方面的相关知识，主要包括事件的类型、JavaScript 中预定义的事件处理器、如何自定义事件处理器等。同时着重讲述了 IE4 和 NN4 对基本事件模型的扩展，Event 对象在 IE4 和 NN4 中的异同点，以及在 DOM2 标准事件模型中引入的几个重要概念。

事件的基石是对象，对 JavaScript 事件处理有了基本认识之后，下章讲述 JavaScript 基于对象编程的相关知识。

第 4 章 JavaScript 基于对象编程

JavaScript 脚本是基于对象（Object-based）的编程语言，通过对象的组织层次来访问并给对象施以相应的操作方法，可大大简化 JavaScript 程序的设计，并提供直观、模块化的方式进行脚本程序开发。本章主要介绍 JavaScript 的基于对象编程、DOM 的模型层次以及有关对象的基本概念等，并引导读者创建和使用自定义的对象。

4.1 面向对象编程与基于对象编程

在软件编程术语中，存在两个类似的概念：面对对象编程（Object Oriented Programming：OPP）和基于对象编程（Object-based Programming），它们在对象创建、对象组织层次、代码封装和复用等方面存在较大的差异。

在了解它们之间差异之前，先来了解对象的概念。

4.1.1 什么是对象

对象是客观世界存在的人、事和物体等实体。现实生活中存在很多的对象，比如猫、自行车等。不难发现它们有两个共同特征：都有状态和行为。比如猫有自己的状态（名字、颜色、饥饿与否等）和行为（爬树、抓老鼠等）。自行车也有自己的状态（档位、速度等）和行为（刹车、加速、减速、改变档位等）。若以自然人为例，构造一个对象，可以用图 4.1 来表示，其中 Attribute 表示对象状态，Method 表示对象行为。

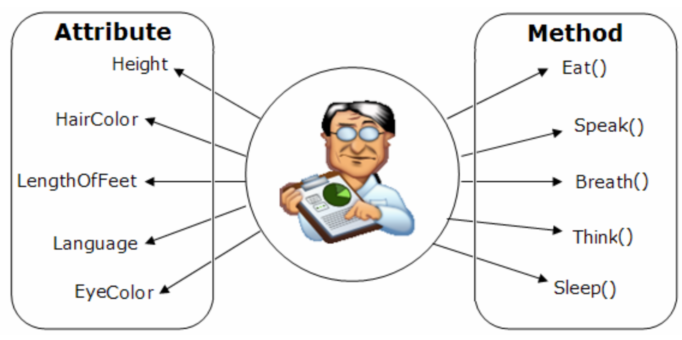

图 4.1 以自然人构造的对象

在软件世界也存在对象，可定义为相关变量和方法的软件集。主要由两部分组成：

- 一组包含各种类型数据的属性
- 允许对属性中的数据进行操作且有相关方法

以 HTML 文档中的 document 作为一个对象，如图 4.2 所示。

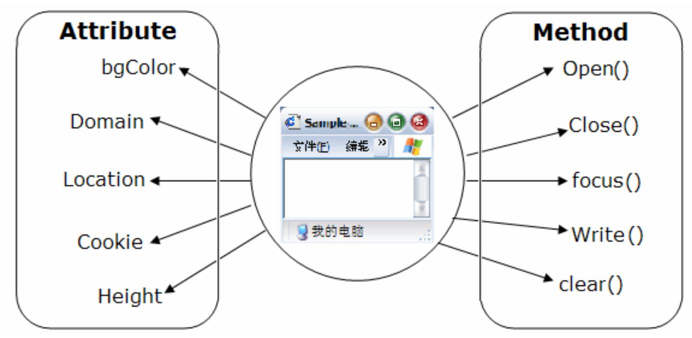

图 4.2 以 HTML 文档中的 document 构造的对象

综上所述，凡是能够提取一定度量数据并能通过某种途径对度量数据实施操作的客观存在都可以构成一个对象，且用属性来描述对象的状态，使用方法和事件来处理对象的各种行为。

- 属性：用来描述对象的状态。通过定义属性值，可以改变对象的状态。如图 4.1 中，可以定义字符串 HungryOrNot 来表示该自然人肚子的状态，HungryOrNot 成为自然人的某个属性；
- 方法：由于对象行为的复杂性，对象的某些行为可用通用的代码来处理，这些通用的代码称为方法。如图 4.1 中，可以定义方法 Eat( )来处理自然人肚子很饿的情况，Eat( )成为自然人的某个方法；
- 事件：由于对象行为的复杂性，对象的某些行为不能使用通用的代码来处理，需要用户根据实际情况爱编写处理该行为的代码，该代码称为事件。在图 4.1 中，可以定义事件 DrinkBeforeEat( )来处理自然人肚子很饿同时嘴巴很渴需要先喝水后进食的情况。

了解了什么是对象，下面来看看什么是面向对象编程。

4.1.2 面向对象编程

面向对象编程（OPP）是一种计算机编程架构，其基本原则：计算机程序由单个能够起到子程序作用的单元或对象组合而成。具有三个最基本的特点：重用性、灵活性和扩展性。这种方法将软件程序的每个元素构成对象，同时对象的类型、属性和描述对象的方法。为了实现整体操作，每个对象都能够接收信息、处理数据和向其它对象发送信息。

面向对象编程主要包含有以下重要的概念：

1．继承

允许在现存的组件基础上创建子组件，典型地说就是用类来对组件进行分组，而且还可以定义新类（子类）为现存的类（父类）的扩展，子类继承了父类的全部属性、方法和事件而不必重新定义；同时通过扩展，子类可以获得专属自己的属性、方法和事件（不影响父类的属性、方法和事件等），这样就可以将所有类拓扑成树形或网状结构。以动物“虎”类为例，拓扑成的树状结构如图 4.3 所示。

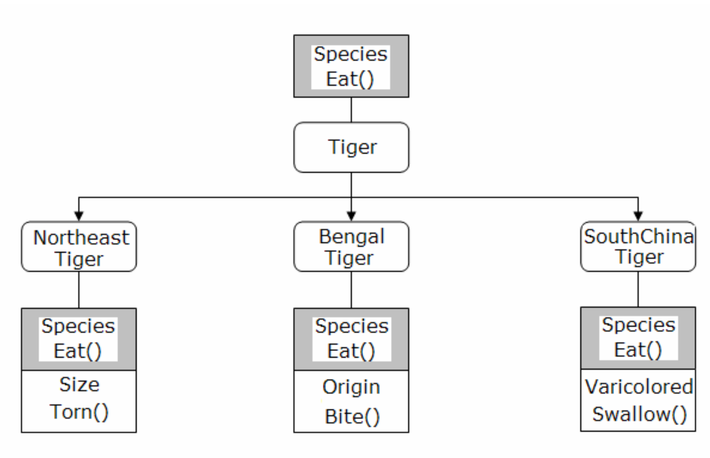

图 4.3 通过继承形成的树形结构

其中灰色框内为“虎”科共有的属性和方法，在生成子类的同时被子类继承，白色长方形框内的为经子类扩展的而特有的属性和方法，同时子类对父类的扩展并不影响父类的任何属性、方法和事件。

2. 封装

封装就是将对象的实现过程通过函数等方式封装起来，使用户只能通过对象提供的属性、方法和事件等接口去访问对象，而不需要知道对象的具体实现过程。封装允许对象运行的代码相对于调用者来说是完全独立的，调用者通过对象及相关接口参数来访问此接口。只要对象的接口不变，而只是对象的内部结构或实现方法发生了改变，程序的其他部分不用作任何处理。模拟吃饭的过程，例如有对象 Tom 及其属性 HungryOrNot、ThirstyOrNot 和方法 DrinkWater(CupNumber)、Eat(GramNumber)，下面的类 C 代码演示何为代码的封装：

```
If(Tom.HungryOrNot==YES)
{
    if(Tom.ThirstyOrNot==YES)
    {
      Tom.DrinkWater(1);
    }
    Tom.Eat(1);
}
```

在操作对象的过程中，用户并不需要知道 DrinkWater(CupNumber)、Eat(GramNumber)方法的具体实现过程，只需知道对象的接口，然后传递相应的参数即可操作对象，实现了对象的封装。

3. 多态

多态指一种对象类型定义多种实现的方案，具体实现的方案由使用的环境来决定。这样就形成了可复用的代码和标准化的程序。广义上讲，多态指一段程序能够处理多种类型对象的能力。仍然模拟吃饭的过程，例如有对象 Tom 及其属性 HungryOrNot、ThirstyOrNot 和方法 DrinkWater(CupNumber)、Drink Milk (CupNumber)、Eat(GramNumber)、FinalEat(FoodType)，下面的类 C 代码演示何为多态性：

```
::FinalEat(FoodType)
{
   if(Tom.ThirstyOrNot==YES)
   {
```

```
        if(FoodType==Rice)
          Tom.DrinkWater(1);
        else if(FoodType==Bread)
          Tom.DrinkMilk(1);
        else
            exit(1);
    }
    Tom.Eat(1);
}
```

Tom根据调用Tom.FinalEat(FoodType)使用的参数FoodType是米饭还是面包等其它东西决定下一步的动作，实现了多态性。

4.1.3 基于对象编程

定位 JavaScript 脚本为基于对象的脚本编程语言而不是面向对象的编程语言，是因为JavaScript以DOM和BOM中定义的对象模型及操作方法为基础，但又不具备面向对象编程语言所必具备的显著特征如分类、继承、封装、多态、重载等，只能通过嵌入其他面向对象编程语言如Java生成的Java applet组件等实现Web应用程序的功能。

JavaScript支持DOM和BOM提供的对象模型，用于根据其对象模型层次结构来访问目标对象的属性并施加对象以相应的操作。

在支持对象模型预定义对象的同时，JavaScript支持Web应用程序开发者定义全新的对象类型，并通过操作符 new 来生成该对象类型的实例。但不支持强制的数据类型，任何类型的对象都可以赋予任意类型的数值。

> 注意：在JavaScript语言中，之所以任何类型的对象都可以赋予任意类型的数值，是因为JavaScript为弱类型的脚本语言，即变量在使用前不需作任何声明，在浏览器解释运行其代码时才检查目标变量的数据类型。

下面简要介绍HTML文档的结构和文档对象模型（DOM）。

4.2 JavaScript 对象的生成

JavaScript 是基于对象的编程语言，除循环和关系运算符等语言构造之外，其所有的特征几乎都是按照对象的处理方法进行的。JavaScript支持的对象主要包括：

- JavaScript 核心对象：包括同基本数据类型相关的对象（如 String、Boolean、Number）、允许创建用户自定义和组合类型的对象（如Object、Array）和其他能简化 JavaScript 操作的对象（如 Math、Date、RegExp、Function）。该部分内容将在第6章重点叙述。
- 浏览器对象：包括不属于JavaScript语言本身但被绝大多数浏览器所支持的对象，如控制浏览器窗口和用户交互界面的window对象、提供客户端浏览器配置信息的Navigator对象。该部分内容将在第7章重点叙述。
- 用户自定义对象：由Web应用程序开发者用于完成特定任务而创建的自定义对象，可自由设计对象的属性、方法和事件处理程序，编程灵活性较大。该部分内容将在本章后续小节叙述。
- 文本对象：由文本域构成的对象，在DOM中定义，同时赋予很多特定的处理方法，

如 insertData( )、 appendData( )等。该部分内容将在第 5 章详细叙述。

注意：ECMA-262 标准只规定 JavaScript 语言基本构成，而 W3C DOM 规范只规定了文档对象模型的访问层次及如何在 JavaScript 脚本中实现，浏览器厂商只定义用户界面并扩展 DOM 层次，基于以上原因，上述对象分类方法可能导致重叠现象出现。

本章主要初步叙述 DOM 框架，而把重点放在如何创建和使用用户自定义的对象上。首先来了解 HTML 文档的结构。

4.2.1 HTML 文档结构

在 HTML 文档中，其标记如<body>与</body>、<p>与</p>等都是成对出现的，称为标记对。文档内容通过这些成对出现的标记对嵌入到文档中，与 JavaScript 脚本等其他代码一起构成一个完整的 HTML 文档。观察如下的简单文档：

```
//源程序 4.1
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
  <title>Frist Page!</title>
</head>
<body>
  <p>DOM</p>
</body>
</html>
```

可绘制成图 4.4 所示的 HTML 元素层次结构图。

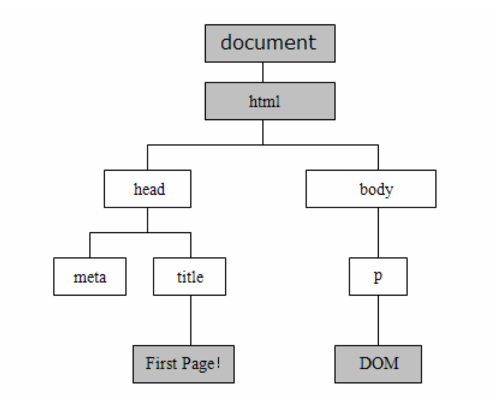

图 4.4 实例的 HTML 元素层次结构图

载入文档后，document 元素相对于该文档而言是唯一的，访问该层次结构图中任何元素都以 document 为根元素进行访问。同时<html>标记元素是<head>标记元素和<body>标记元素的父元素，同时又是 document 元素的子元素。可见如果 HTML 文档中严格使用 HTML 成对标记，则其元素是相互嵌套的，并且通过相互之间在结构图中的层次结构关系可实现相互定位，为 DOM 框架提供了理论基础。

注意：这里所说的 document 元素、<html>标记元素等，只是在单纯的 HTML 文档背景下定义的；在后面讲到的 DOM 框架中，元素又被称为对象；在浏览器载入该文档并根据 DOM 模型生成的节点

树中，元素又被称为节点。其实它们代表同一事物，只是定义的背景不同而已。

4.2.2 DOM 框架

由 HTML 文档结构可知，文档中各个元素（标记元素、文本元素等）在 HTML 元素层次结构图中都被标记为具有一定“社会”关系的成员，并可通过这种关系来访问指定的成员，那我们是不是可以设定这种结构模型的某种标准，以便实现元素访问方法的一致性呢？

DOM（文档结构模型）应运而生，其主要关注在浏览器解释 HTML 文档时如何设定各元素的这种“社会”关系及处理这种关系的方法。从实际应用的角度出发，HTML 文档根据 DOM 中定义的框架模型在浏览器解释后生成对象访问层次，而 JavaScript 脚本经常要控制其中的某个对象。DOM 基本框架如图 4.5 所示，其中灰色代表模型中的顶级对象，包括 window 对象及其下的 frames、location、document、history、navigator、screen 等对象：

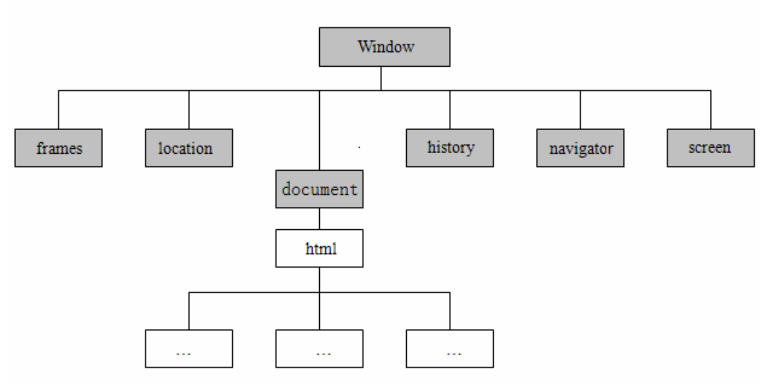

图 4.5 DOM 框架结构示意图

DOM 中几个顶级对象及其作用如表 4.1 所示。

表 4.1 DOM中的顶级对象及其作用

对象名称	作用
window	表示与当前浏览器窗口相关的最顶级对象，包含当前窗口的最小最大化、尺寸大小等信息同时具有关闭、新建窗口等方法。
frames[]	表示Window页面中的框架数组对象，每个框架都包含一个window对象
location	以URL的形式载入当前窗口，并保存正在浏览的文档的位置及其构成，如协议、主机名、端口、路径、URL的查询字符串部分等
document	包含HTML文档中的HTML标记和构成文档内容的文本的对象，客户端浏览器中每个载入的HTML文档都有一个document对象，在多框架文档中，框架集的每个成员都包含一个document对象，按照对象包含的层次进行访问
history	包含当前窗口的历史列表对象，用于跟踪窗口中曾经使用过的URL，包括其历史表的长度、历史表中上一个URL和下一个URL等信息。
navigator	包含当前浏览器的相关信息的对象，包括处理当前文档的客户端浏览器的版本号、商标等只读信息，防止脚本对客户端浏览器相关信息的恶意访问和篡改。
screen	包含当前浏览器运行的物理环境信息的对象，包含如监视器的有效像数等信息。

对 DOM 框架层次及其相关顶级对象的了解有助于更好理解浏览器载入 HTML 文档时 JavaScript 对象的生成过程。后面的“文档对象模型(DOM)”章节将详细讨论 DOM。

4.2.3 顶级对象之间的关系

我们来模拟浏览器载入某标准 HTML 的过程来阐述 window、frames[]、location 等几种常见的顶级对象之间的关系。首先参考如下包含框架集的标准 HTML 文档：

```
<html>
<head>
</head>
<body>
  <frameset>
    <frame src="a.html">
    <frame src="b.html">
  </frameset>
</body>
</html>
```

将上述代码保存为 Sample.html，鼠标双击，系统调用默认的浏览器，生成 window 对象，打开 Sample.html 文档后，生成 screen、navigator、location、history、frames[]和 document 等对象。

- window 对象包含当前浏览器窗口中的所有对象，为对象访问过程忠默认的顶级对象，如引用该对象的 alert( )方法，可将 window.alert(msg)直接改写为 alert(msg)，同样 window.document.forms[1]可改写为 document.forms[1]；
- screen 对象包含当前浏览器运行的物理环境信息，如当前屏幕分辨率；
- navigator 对象包含当前浏览器的相关信息，如浏览器版本等；
- location 对象以 URL 形式保存正在浏览的文档相关信息，如路径等；
- history 对象包含浏览器当前窗口的访问历史列表，如单击链接进入新页面，则原始页面地址列入当前窗口的访问历史列表中；
- frames[]对象包含当前 Window 页面中的框架数组成员，如实例中的两个框架，每个框架都包含一个独立的 document 对象；
- document 对象包含 HTML 文档中的 HTML 标记和构成文档内容的文本的对象，在每个单独保存的 HTML 文档中都直接包含一个 document 对象。

由上面的分析，可看出从浏览器打开文档至关闭文档期间，screen 和 navigator 对象不变（用户不改变其硬件和软件设置），且与文档无关；location 对象代表当前文档位置的相关信息，与文档地址（相对地址或绝对位置）及访问方法相关；history 对象则是当前文档页面的访问列表，与文档中链接及页面跳转情况有关；frames[]和 document 对象则是浏览器载入时根据文档结构生成的对象，决定于文档。

理解这几个顶级对象的关系有助于深入了解浏览器解释运行 HTML 文档过程中各对象的生成步骤和相互之间如何影响彼此。

4.2.4 浏览器载入文档时对象的生成

浏览器载入 HTML 文档时，根据 DOM 定义的结构模型层次，当遇到自身支持的 HTML 元素对象所对应的标记时，就按 HTML 文档载入的顺序在客户端内存中创建这些对象，并按对象创建的顺序生成对象数组，而不管 JavaScript 脚本是否真正运行这些对象。对象创建后，浏览器为这些对象提供专供 JavaScript 脚本使用的可选属性、方法和处理程序，Web 应用程序开发者通过这些属性、方法和处理程序就能动态操作 HTML 文档内容。

下面的实例说明客户端浏览器载入 HTML 文档时将按载入时对象创建的顺序生成对象

数组：

```
//源程序 4.2
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <title>Sample Page!</title>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
<h5>Test!</h5>
<!--NOTE!-->
<p>
   Welcome to
   <em>
      DOM
   </em>
   World!
</p>
<ul>
  <li>
    Newer
  </li>
</ul>
<hr>
<br>
<script language="JavaScript" type="text/javascript">
<!-- //在支持 JavaScript 脚本的浏览器将忽略该标记
  var i,origlength,msg;
  //获取生成的对象数组长度
  origlength=document.all.length;
  msg="对象数组长度: "+origlength+"\n";
  //循环输出各节点的类型和 tagName 属性值
  for(i=0;i<origlength;i++)
  {
    if(i<10)
    {
      msg+="类型:"+typeof(document.all[i])+"编号:   "_
           +i+"名称:"+document.all[i].tagName+"\n";
    }
    else
    {
      msg+="类型:"+typeof(document.all[i])+"编号:"_
           +i+"名称:"+document.all[i].tagName+"\n";
    }
  }
  //使用警示框输出信息
  window.alert(msg);
-->     //在支持 JavaScript 脚本的浏览器将忽略该标记
</script>
</body>
</html>
```

运行上面的程序，结果如图 4.6 所示。

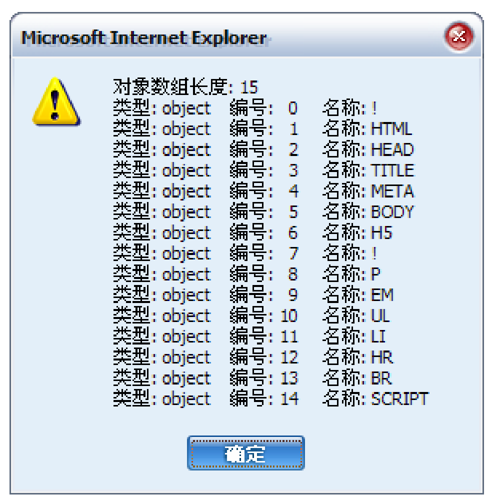

图 4.6 HTML 文档载入时生成的对象数组顺序图

分析上述代码，浏览器载入该文档时生成对象的顺序应为<! DOCTYPE>、<HTML>、<HEAD>、<TITLE>、<META>、<BODY>、<H5>、<!>、<P>、<EM>、<UL>、<LI>、< HR >、
及<SCRIPT>。图 4.6 表明两点：

- 浏览器载入文档时，根据当前浏览器支持的 DOM 规范级别生成对应于 HTML 标记的对象（object）；
- 浏览器根据其将各标记载入时的先后顺序生成对象数组的顺序。

对象生成后，浏览器将调用与对象相对应的属性、方法和事件供 JavaScript 脚本根据用户动作和页面动作进行相关处理。在本书的后续章节将详细讲解 HTML 通用元素对象，下面简要介绍 JavaScript 脚本中的核心对象。

4.3　JavaScript 核心对象

JavaScript 作为一门基于对象的编程语言，以其简单、快捷的对象操作获得 Web 应用程序开发者的首肯，而其内置的几个核心对象，则构成了 JavaScript 脚本语言的基础。主要核心对象如表 4.2 所示。

表 4.2 JavaScript核心对象

核心对象	附加说明
Array	提供一个数组模型，用来存储大量有序的类型相同或相似的数据，将同类的数据组织在一起进行相关操作。
Boolean	对应于原始逻辑数据类型，其所有属性和方法继承自Object对象。当值为真表示true，值为假则表示false。
Date	提供了操作日期和时间的方法，可以表示从微秒到年的所有时间和日期。使用Date读取日期和时间时，其结果依赖于客户端的时钟。
Function	提供构造新函数的模板，JavaScript中构造的函数是Function对象的一个实例，通过函数名实现对该对象的引用。
Math	内置的Math对象可以用来处理各种数学运算，且定义了一些常用的数学常数，如Math对象的实例的PI属性返回圆周率π的值。各种运算被定义为Math对象的内置方法，可直接调用。
Number	对应于原始数据类型的内置对象，对象的实例返回某数值类型。
Object	包含由所有 JavaScript 对象所共享的基本功能，并提供生成其它对象如Boolean等对象的模板和基本操作方法。
RegExp	表述了一个正则表达式对象，包含了由所有正则表达式对象共享的静态属性，用于指定字符或字符串的模式。

String	和原始的字符串类型相对应，包含多种方法实现字符串操作如字符串检查、抽取子串、连接两个字符串甚至将字符串标记为HTML等。

JavaScript 语言中，每种基本类型都构成了一个 JavaScript 核心对象，并由 JavaScript 提供其属性和方法，Web 应用程序开发者可以通过操作对象的方法来操作该基本类型的实例。

4.4 文档对象的引用

客户端浏览器载入 HTML 文档时，对于所有可以编码的 HTML 元素按照 DOM 规范和载入元素的顺序生成对象数组，然后进行初始化供 JavaScript 脚本使用。下面讨论 JavaScript 脚本访问文档对象的方法。

4.4.1 通过对象位置访问文档对象

浏览器载入 HTML 文档后，将根据该文档的结构和 DOM 规范生成对象数组，该对象数组中各对象之间的相对位置随着 HTML 文档的确定而确定下来，JavaScript 脚本可以通过这个确定的相对位置来访问该对象。考察如下的 HTML 文档：

```
//源程序 4.3
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <title>Sample Page!</title>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
<form>
  <input type=text value="Text in Form1">
</form>
<form>
  <input type=text value="Text1 of Form2">
  <input type=text value="Text2 of Form2">
</form>
<script language="JavaScript" type="text/javascript">
<!--
  var msg="";
  msg+="通过位置访问文档对象:\n\n";
  msg+="Form[0].element[0].value: "+document.forms[0].elements[0].value+"\n\n";
  msg+="Form[1].element[0].value: "+document.forms[1].elements[0].value+"\n\n";
  msg+="Form[1].element[1].value: "+document.forms[1].elements[1].value+"\n\n";
  //使用警示框输出信息
  window.alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.7 所示。

图 4.7 通过位置访问文档对象

浏览器载入该文档时，生成 forms[]数组，第一个 form 为 form[0]，第二个 form 为 form[1]：

```
<form>
  <input type=text value="Text in Form1">
</form>
<form>
  <input type=text value="Text1 of Form2">
  <input type=text value="Text2 of Form2">
</form>
```

则通过它们的位置进行访问的方法如下：

```
document.forms[0]
document.forms[1]
```

而 form[1]下面还有两个文本框，可通过如下方式访问：

```
document.forms[1].elements[0]
document.forms[1].elements[1]
```

则访问第二个 form 里面的第二个 text 的 value 属性可通过如下的方法：

```
document.forms[1].elements[1].value
```

此种方法简单明了，但对象的位置依赖于 HTML 文档的结构，如果文档结构改变而不改变上述的访问代码，浏览器弹出“某某对象为空或无此对象”错误信息。

4.4.2 通过对象名称访问文档对象

上述通过位置访问文档对象的方法由于对象对文档结构的依赖性太大，一旦文档结构改变，就必须改变对象访问的语句，这给脚本代码维护带来了很大的难度，可以通过给对象命名的方法来解决。

1. 通过 name 属性访问文档对象

在 HTML 4 版本之前，使用元素的 name 属性来给 HTML 文档中的元素进行标注，支持 name 属性的标记有：<form>、<input>、<button>、<select>、<text>、<textarea>、<a>、<applet>、<embeds>、<frame>、<iframe>、<img>、<object>、<map>等。考察如下的 HTML 文档：

```
//源程序 4.4
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <title>Sample Page!</title>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
```

```
//第一个表单
<form name="MyForm1">
  <input type=text name="MyTextOfForm1" value="Text in Form1">
</form>
//第二个表单
<form name="MyForm2">
  <input type=text name="MyText1OfForm2" value="Text1 of Form2">
  <input type=text name="MyText2OfForm2" value="Text2 of Form2">
</form>
<script language="JavaScript" type="text/javascript">
<!--
  var msg="";
  msg+="通过名称访问文档对象:\n\n";
  msg+="MyForm1.MyTextOfForm1.value:"+document.MyForm1.MyTextOfForm1.value+" \n\n";
  msg+="MyForm2.MyText1OfForm2.value:"+document.MyForm2.MyText1OfForm2.value+" \n\n";
  msg+="MyForm2.MyText2OfForm2.value:"+document.MyForm2.MyText2OfForm2.value+" \n\n";
  //使用警示框输出信息
  window.alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.8 所示。

图 4.8 通过 name 属性访问文档对象

程序首先设定第一个表单的 name 属性为 MyForm1，包含在它里面的文本框的 name 属性为 MyTextOfForm1，第二个表单的 name 属性为 MyForm1，包含在它里面的两个文本框的 name 属性分别为：MyText1OfForm2 和 MyText2OfForm2。则通过如下的代码访问三个文本框的 value 属性：

```
document.MyForm1.MyTextOfForm1.value
document.MyForm2.MyText1OfForm2.value
document.MyForm2.MyText2OfForm2.value
```

2. 通过 id 属性访问文档对象

在 HTML 4 版本中添加了 HTML 元素的 id 属性来定位文档对象，考察如下的代码：

```
//源程序 4.5
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
```

```
<title>Sample Page!</title>
</head>
<body>
<p id="p1">Welcome to<B> DOM </B>World! </p>
<script language="JavaScript" type="text/javascript">
<!--
//返回对象的相关信息
function nodeStatus(node)
{
  var temp="";
  if(node.nodeName!=null)
  {
    temp+="nodeName: "+node.nodeName+"\n";
  }
  else temp+="nodeName: null!\n";
  if(node.nodeType!=null)
  {
    temp+="nodeType: "+node.nodeType+"\n";
  }
  else temp+="nodeType: null\n";
  if(node.nodeValue!=null)
  {
    temp+="nodeValue: "+node.nodeValue+"\n\n";
  }
  else temp+="nodeValue: null\n\n";
  return temp;
}
//处理并输出信息
//返回 id 属性值为 p1 的元素对象
var currentElement=document.getElementById('p1');
var msg=nodeStatus(currentElement);
//返回 p1 的第一个孩子，并输出相关信息
currentElement=currentElement.firstChild;
msg+=nodeStatus(currentElement);
alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.9 所示。

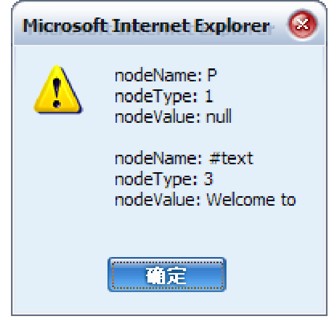

图 4.9 通过 id 属性访问文档对象

在段落语句中，设定 id 为 p1：

```
<p id="p1">Welcome to<B> DOM </B>World! </p>
```

然后通过这个 id 属性访问到 p 元素，即定位了该对象（元素对象）：

```
currentElement=document.getElementById('p1');
```

该 currentElement 即为通过 id 属性返回的对象 p，然后进行相关处理。关于元素节点的相关知识将在“文档结构模型(DOM)”章节中将详细讲解。

由于 id 属性为 HTML 4 新添加的属性，在老版本中得不到支持，而 name 属性则支持新版本和老版本，最为可靠的方法就是同时设置 name 属性和 id 属性，并将它们设置为相同的值，然后通过相同的访问方法进行访问。如有下列表单：

```
<form name="MyForm" id="MyForm">
  <input type=text name="MyText" id="MyText" value="MyTextOfMyFrom">
</form>
```

如果要访问 MyForm 中的文本框 MyText，可使用如下的方法：

```
document.MyForm.MyText
```

通过 id 属性和 name 属性访问文档对象的方法，有利于文档对象的精确定位，同时从根本上解决了“通过对象位置访问文档对象”方法过于依赖文档结构的问题。

4.4.3 通过联合数组访问文档对象

在 HTML 被浏览器解释执行的同时，同类型的元素将构成某个联合数组的元素，可通过一个整数或者字符串为索引参数，完全定位该对象。一般情况下使用 HTML 文档中分配给标记元素的 id 属性或 name 属性作为参数。

下面的代码演示如何通过联合数组访问文档对象：

```
//源程序 4.6
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <title>Sample Page!</title>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
</head>
<body>
//第一个表单
<form name="MyForm1">
  <input type=text name="MyTextOfForm1" value="Text of Form1">
</form>
//第二个表单
<form name="MyForm2">
  <input type=text name="MyText1OfForm2" value="Text1 of Form2">
  <input type=text name="MyText2OfForm2" value="Text2 of Form2">
</form>
<script language="JavaScript" type="text/javascript">
<!--
  var msg="";
  msg+="通过联合数组访问文档对象:\n\n";
  msg+="Form1.MyTextOfForm1.value:"+document.forms["MyForm1"].elements[0].value+"\n\n";
  msg+="Form2.MyText1OfForm2.value:"+document.forms["MyForm2"].elements[0].value+"\n\n";
  msg+="Form2.MyText2OfForm2.value:"+document.forms["MyForm2"].elements[1].value+"\n\n";
  //使用警示框输出信息
```

```
    window.alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 4.10 所示。

图 4.10 使用联合数组访问文档对象

在访问文档对象的实现语句 document.forms["MyForm1"].elements[0]中，也可用如下的语句代替，可实现同样的功能：

```
document.forms["MyForm1"].elements["MyTextOfForm1"]
```

注意：在 IE 中提供专门 item( )方法作为联合数组的索引，该方法将对象集中名为参数字符串的对象从对象集中取出，如上面例子中可用 document.forms.item("MyForm1")语句实现同样功能。

4.5 创建和使用自定义对象

在 JavaScript 脚本语言中，主要有 JavaScript 核心对象、浏览器对象、用户自定义对象和文本对象等，其中用户自定义对象占据举足轻重的地位。

JavaScript 作为基于对象的编程语言，其对象实例采用构造函数来创建。每一个构造函数包括一个对象原型，定义了每个对象包含的属性和方法。对象是动态的，表明对象实例的属性和方法是可以动态添加、删除或修改的。

JavaScript 脚本中创建自定义对象的方法主要有两种：通过定义对象的构造函数的方法和通过对象直接初始化的方法。

4.5.1 通过定义对象的构造函数的方法

下面的实例是通过定义对象的构造函数的方法和使用 new 操作符所生成的对象实例，先考察其代码：

```
//源程序 4.7
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
```

```
<script>
<!--
//对象的构造函数
function School(iName,iAddress,iGrade,iNumber)
{
   this.name=iName;
   this.address=iAddress;
   this.grade=iGrade;
   this.number=iNumber;
   this.information=showInformation;
}
//定义对象的方法
function showInformation()
{
   var msg="";
   msg="自定义对象实例：\n"
   msg+="\n 机构名称 ："+this.name+" \n";
   msg+="所在地址 ："+this.address +"\n";
   msg+="教育层次 ："+this.grade +" \n";
   msg+="在校人数 ："+this.number
   window.alert(msg);
}
//生成对象的实例
var ZGKJDX=new School("中国科技大学","安徽・合肥","高等学府","13400");
-->
</script>
</head>
<body>
<br>
<center>
<form>
   <input type=button value=调试对象按钮  onclick="ZGKJDX.information()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 4.11 所示。

图 4.11 通过定义对象的构造函数生成自定义对象实例

在该方法中，用户必须先定义一个对象的构造函数，然后再通过 new 关键字来创建该对象的实例。

定义对象的构造函数如下：

```
function School(iName,iAddress,iGrade,iNumber)
{
   this.name=iName;
   this.address=iAddress;
   this.grade=iGrade;
   this.number=iNumber;
   this.information=showInformation;
}
```

当调用该构造函数时，浏览器给新的对象分配内存，并隐性地将对象传递给函数。this 操作符是指向新对象引用的关键词，用于操作这个新对象。下面的句子：

```
this.name=iName;
```

该句使用作为函数参数传递过来的 iName 值在构造函数中给该对象的 name 属性赋值，该属性属于所有 School 对象，而不仅仅属于 School 对象的某个实例如上面中的 ZGKJDX。对象实例的 name 属性被定义和赋值后，可以通过如下方法访问该实例的该属性：

```
var str=ZGKJDX.name;
```

使用同样的方法继续添加其他属性 address、grade、number 等，但 information 不是对象的属性，而是对象的方法：

```
this.information=showInformation;
```

方法 information 指向的外部函数 showInformation 结构如下：

```
function showInformation()
{
   var msg="";
   msg="自定义对象实例：\n"
   msg+="\n 机构名称 ："+this.name+" \n";
   msg+="所在地址 ："+this.address +"\n";
   msg+="教育层次 ："+this.grade +" \n";
   msg+="在校人数 ："+this.number
   window.alert(msg);
}
```

同样，由于被定义为对象的方法，在外部函数中也可使用 this 操作符指向当前的对象，并通过 this.name 等访问它的某个属性。

在构建对象的某个方法时，如果代码比较简单，也可以使用非外部函数的做法，改写 School 对象的构造函数：

```
function School(iName,iAddress,iGrade,iNumber)
{
   this.name=iName;
   this.address=iAddress;
   this.grade=iGrade;
   this.number=iNumber;
   this.information=function()
                          {
                              var msg="";
                              msg="自定义对象实例：\n"
                              msg+="\n 机构名称 ："+this.name+" \n";
                              msg+="所在地址 ："+this.address +"\n";
                              msg+="教育层次 ："+this.grade +" \n";
                              msg+="在校人数 ："+this.number;
                              window.alert(msg);
                          };
}
```

删除源程序 4.7 中外部函数 showInformation，保存更改，程序运行结果与源程序 4.7 运

行结果相同。

4.5.2 通过对象直接初始化的方法

此方法通过直接初始化对象来创建自定义对象，与定义对象的构造函数方法不同的是，该方法不需要生成此对象的实例，改写源程序 4.7：

```
<script>
<!--
//直接初始化对象
var ZGKJDX={name:"中国科技大学",
            address:"安徽・合肥",
            grade:"高等学府",
            number:"13400",
            information:showInformation
           };
//定义对象的方法
function showInformation()
{
  var msg="";
  msg="自定义对象实例：\n"
  msg+="\n 机构名称 ："+this.name+" \n";
  msg+="所在地址 ："+this.address +"\n";
  msg+="教育层次 ："+this.grade +" \n";
  msg+="在校人数 ："+this.number
  window.alert(msg);
}
-->
</script>
```

程序运行结果与源程序 4.7 运行结果相同。

该方法在只需生成某个应用对象并进行相关操作的情况下使用时，代码紧凑，编程效率高，但致命的是，若要生成若干个对象的实例，就必须为生成每个实例重复相同的代码结构，而只是参数不同而已，代码的重用性比较差，不符合面向对象的编程思路，应尽量避免使用该方法创建自定义对象。

4.5.3 修改、删除对象实例的属性

JavaScript 脚本可动态添加对象实例的属性，同时，也可动态修改、删除某个对象实例的属性，更改源程序 4.7 中的 showInformation( )代码：

```
function showInformation()
{
  var msg="";
  msg="用户自定义的对象实例：\n\n"
  msg+="   机构名称 ："+this.name+" \n";
  msg+="   所在地址 ："+this.address +"\n";
  msg+="   教育层次 ："+this.grade +" \n";
  msg+="   在校人数 ："+this.number+" \n\n";
  //修改对象实例的 number 属性
  this.number=23500;
  msg+="修改对象实例的属性:\n\n"
```

```
    msg+="   机构名称 : "+this.name+" \n";
    msg+="   所在地址 : "+this.address +"\n";
    msg+="   教育层次 : "+this.grade +" \n";
    msg+="   在校人数 : "+this.number+" \n\n";
    //删除对象实例的 number 属性
    delete this.number;
    msg+="删除对象实例的属性:\n\n"
    msg+="   机构名称 : "+this.name+" \n";
    msg+="   所在地址 : "+this.address +"\n";
    msg+="   教育层次 : "+this.grade +" \n";
    msg+="   在校人数 : "+this.number+" \n\n";
    window.alert(msg);
}
```

程序运行结果如图 4.12 所示。

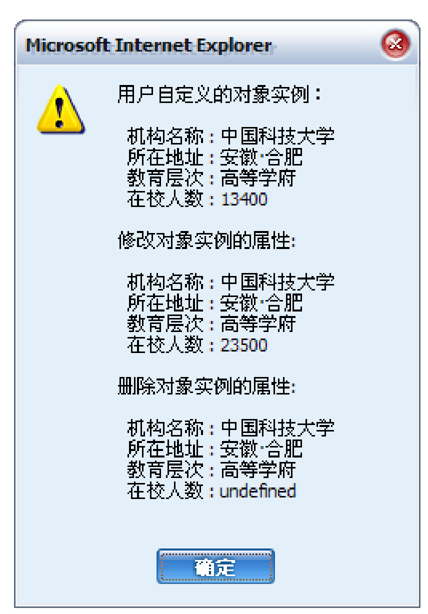

图 4.12 修改和删除对象实例的属性

从以上程序可以看出，执行 this.number=23500 语句后，对象实例的 number 属性值更改为 23500；执行 delete this.number 语句后，对象实例的 number 属性变为 undefined，同任何不存在的对象属性一样为未定义类型，但我们并不能删除对象实例本身，否则返回错误。

可见，JavaScript 动态添加、修改、删除对象实例的属性过程十分简单，之所以称之为对象实例的属性而不是对象的属性，是因为该属性只在对象的特定实例中才存在，而不能通过某种方法将某个属性赋予特定对象的所有实例。

> 注意：JavaScript 脚本中的 delete 运算符用于删除对象实例的属性，与 C++中用途不一样，C++中 delete 运算符能删除对象的实例。

4.5.4 通过原型为对象添加新属性和新方法

JavaScript 语言中所有对象都由 Object 对象派生，每个对象都有指定了其结构的原型（prototype）属性，该属性描述了该类型对象共有的代码和数据，可以通过对象的 prototype 属性为对象动态添加新属性和新方法。考察如下的代码：

```
//源程序 4.8
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script>
<!--
//对象的构造函数
function School(iName,iAddress,iGrade,iNumber)
{
  this.name=iName;
  this.address=iAddress;
  this.grade=iGrade;
  this.number=iNumber;
  this.information=showInformation;
}
//定义对象的方法
function showInformation()
{
  var msg="";
  msg="通过原型给对象添加新属性和新方法：\n\n"
  msg+="原始属性:\n";
  msg+="      机构名称 ："+this.name+" \n";
  msg+="      所在地址 ："+this.address +"\n";
  msg+="      教育层次 ："+this.grade +" \n";
  msg+="      在校人数 ："+this.number+" \n\n";
  msg+="新属性:\n";
  msg+="      占地面积 ："+this.addAttributeOfArea+" \n";
  msg+="新方法:\n";
  msg+="      方法返回 ："+this.addMethod+"\n";
  window.alert(msg);
}
function MyMethod()
{
  var AddMsg="New Method Of Object!";
  return AddMsg;
}
//生成对象的实例
var ZGKJDX=new School("中国科技大学","安徽・合肥","高等学府","13400");
School.prototype.addAttributeOfArea="3000";
School.prototype.addMethod=MyMethod();
-->
</script>
</head>
<body>
<br>
<center>
<form>
  <input type=button value="调试对象按钮" onclick="ZGKJDX.information()">
</form>
</center>
```

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

程序运行结果如图 4.13 所示。

图 4.13 通过原型给对象添加新属性和新方法

程序调用对象的 prototype 属性给对象添加新属性和新方法：

```
School.prototype.addAttributeOfArea="3000";
School.prototype.addMethod=MyMethod();
```

原型属性为对象的所有实例所共享，用户利用原型添加对象的新属性和新方法后，可通过对象引用的方法来修改。

4.5.5 自定义对象的嵌套

与面向对象编程方法相同的是，JavaScript 允许对象的嵌套使用，可以将对象的某个实例作为另外一个对象的属性来看待，考察下列代码：

```
//源程序 4.9
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script>
<!--
//对象的构造函数
//构造嵌套的对象
var SchoolData={
                    code:"0123-456-789",
                    Tel:"0551-1234567",
                    Fax:"0551-7654321"
                   };
//构造被嵌入的对象
var ZGKJDX={
                name:"中国科技大学",
```

```
                address:"安徽·合肥",
                grade:"高等学府",
                number:"13400",
                //嵌套对象 SchoolData
                data:SchoolData,
                information:showInformation
               };
//定义对象的方法
function showInformation()
{
  var msg="";
  msg="对象嵌套实例：\n\n";
  msg+="被嵌套对象直接属性值:\n"
  msg+="      机构名称 ："+this.name+"\n";
  msg+="      所在地址 ："+this.address +"\n";
  msg+="      教育层次 ："+this.grade +"\n";
  msg+="      在校人数 ："+this.number +"\n\n";
  msg+="访问嵌套对象直接属性值:\n"
  msg+="      学校代码 ："+this.data.code +"\n";
  msg+="      办公电话 ："+this.data.Tel +" \n";
  msg+="      办公传真 ："+this.data.Fax +" \n";
  window.alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form>
  <input type=button value=调试对象按钮  onclick="ZGKJDX.information()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 4.14 所示。

图 4.14 自定义对象的嵌套

首先构造对象 SchoolData 包含学校的相关联系信息，如下代码：

```
var SchoolData={
                code:"0123-456-789",
                Tel:"0551-1234567",
                Fax:"0551-7654321"
               };
```

然后构建 ZGKJDX 对象，同时嵌入 SchoolData 对象，如下代码：

```
var ZGKJDX={
            name:"中国科技大学",
            address:"安徽・合肥",
            grade:"高等学府",
            number:"13400",
            //嵌套对象 SchoolData
            data:SchoolData,
            information:showInformation
           };
```

可以看出，在构建 ZGKJDX 对象时，程序将 SchoolData 对象作为自身的某个属性 data 对应的值嵌入进去，并可通过如下的代码访问：

```
this.data.code
this.data.Tel
this.data.Fax
```

通过直接对象初始化的方法，上述代码可改写如下：

```
var ZGKJDX={
            name:"中国科技大学",
            address:"安徽・合肥",
            grade:"高等学府",
            number:"13400",
            //嵌套对象 SchoolData
            data:{
                  code:"0123-456-789",
                  Tel:"0551-1234567",
                  Fax:"0551-7654321"
                 },
            information:showInformation
           };
```

程序运行结果与源程序 4.9 相同。

下面介绍对象创建过程中内存的分配和释放问题。

4.5.6 内存的分配和释放

JavaScript 是基于对象的编程语言而不是面向对象的编程语言，缺少指针的概念，而后者在动态分配和释放内存的过程中作用巨大，那 JavaScript 中的内存如何管理呢？

创建对象的同时，浏览器自动为该对象分配内存空间，JavaScript 将新对象的引用传递给调用的构造函数，在对象清除时其占据的内存将自动回收，其实整个过程都是浏览器的功劳，JavaScript 只负责创建该对象。

浏览器中的这种内存管理机制称为“内存回收”，它动态分析程序中每个占据内存空间的数据(变量、对象等)，如果该数据对于程序标记为不可再用时，浏览器将调用内部函数将其占据的内存空间释放，实现内存的动态管理。

当然，在自定义的对象使用完后，可通过给其赋空值的方法标记对象已经使用完成：

```
ZGKJDX=null;
```

浏览器将根据此标记动态释放其占据的内存，否则将保存该对象直至当前程序再次使用它为止。

4.6 本章小结

本章主要介绍了 JavaScript 基于对象编程语言及与 C++、Java 等面向对象编程语言的异同点；初步了解了浏览器载入文档时 JavaScript 对象的产生过程及文档中对象的访问方法；重点介绍了 JavaScript 如何创建和使用自定义的对象以及程序中如何动态管理内存的问题。

JavaScript 语言使用构造函数创建新的对象实例，对象用来将该实例的属性初始化。在 JavaScript 中创建和管理对象是直线式的，并不需要对诸如内存管理（C++等面向对象编程语言概念）等方面特别注意。同时用户自定义的对象可以用于构建模块化和易于维护的脚本，但较之 JavaScript 核心对象和 DOM 规范所定义的对象而言，其功能一般较为简单，实际用途不大。下面几章将重点介绍 DOM 规范和 JavaScript 核心对象等。

第 5 章　文档对象模型(DOM)

文档对象模型（Document Object Model：DOM），最初是 W3C 为了解决浏览器混战时代不同浏览器环境之间的差别而制定的模型标准，主要是针对 IE 和 Netscape Navigator。W3C 解释为："文档对象模型（DOM）是一个能够让程序和脚本动态访问和更新文档内容、结构和样式的语言平台，提供了标准的 HTML 和 XML 对象集，并有一个标准的接口来访问并操作它们。"它使得程序员可以很快捷地访问 HTML 或 XML 页面上的标准组件，如元素、样式表、脚本等等并作相应的处理。DOM 标准推出之前，创建前端 Web 应用程序都必须使用 Java Applet 或 ActiveX 等复杂的组件，现在基于 DOM 规范，在支持 DOM 的浏览器环境中，Web 开发人员可以很快捷、安全地创建多样化、功能强大的 Web 应用程序。本章只讨论 HTML DOM。

5.1 DOM 概述

文档对象模型定义了 JavaScript 可以进行操作的浏览器，描述了文档对象的逻辑结构及各功能部件的标准接口。主要包括如下方面：

- 核心 JavaScript 语言参考（数据类型、运算符、基本语句、函数等）
- 与数据类型相关的核心对象（String、Array、Math、Date 等数据类型）
- 浏览器对象（window、location、history、navigator 等）
- 文档对象（document、images、form 等）

JavaScript 使用两种主要的对象模型：浏览器对象模型（BOM）和文档对象模型（DOM），前者提供了访问浏览器各个功能部件，如浏览器窗口本身、浏览历史等的操作方法；后者则提供了访问浏览器窗口内容，如文档、图片等各种 HTML 元素以及这些元素包含的文本的操作方法。在早期的浏览器版本中，浏览器对象模型和文档对象模型之间没有很大的区别。观察下面的简单 HTML 代码：

```
//源程序 5.1
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <meta http-equiv=content-type content="text/html; charset=gb2312">
  <title> First Page!</title>
</head>
<body>
<h1>Test!</h1>
<!--NOTE!-->
<p>Welcome to<em> DOM </em>World! </p>
<ul>
  <li>Newer</li>
</ul>
</body>
</html>
```

在 DOM 模型中，浏览器载入这个 HTML 文档时，它以树的形式对这个文档进行描述，其中各 HTML 的每个标记都作为一个对象进行相关操作，如图 5.1 所示。

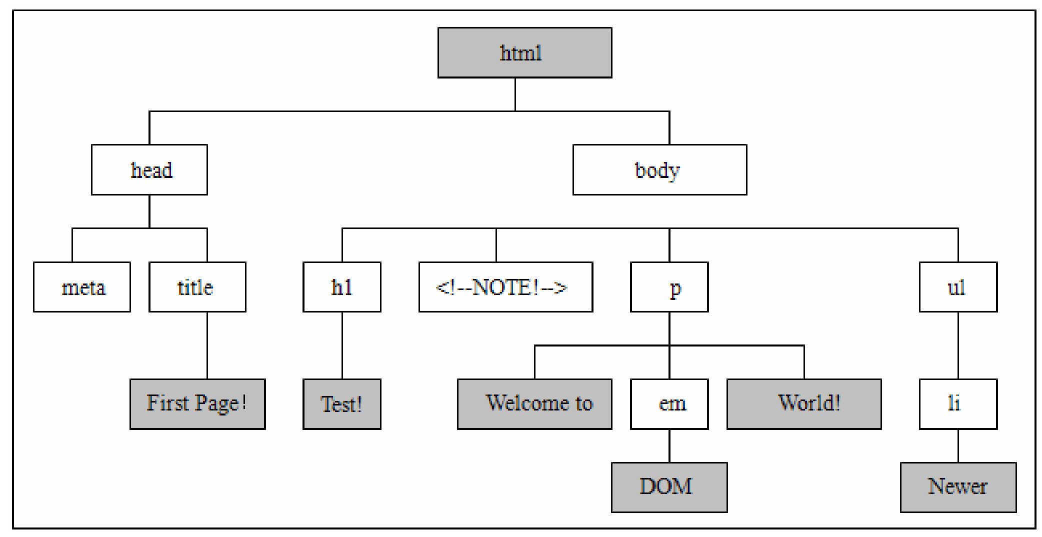

图 5.1 实例的家谱树

可以看出，html 为根元素对象，可代表整个文档，head 和 body 两个分支，位于于同一层次，为兄弟关系，存在同一父元素对象，但又有各自的子元素对象。

在支持脚本的浏览器发展过程中，出现了如下 6 种不同的文档对象模型，如表 5.1 所示：

表 5.1 文档对象模型的各个版本及浏览器支持

文档对象模型	浏览器支持
Basic Object Model（基本对象模型）	NN2，NN3，IE3/J1，IE3/J2，NN4，IE4，IE5，NN6，IE5.5，IE6，Moz1，Safari1
Basic Plus Images（基本附加图像）	NN3，IE3.01（Only for Mac），NN4，IE4，IE5，NN6，IE5.5，IE6，Moz1，Safari1
NN4 Extensions（NN4扩展）	NN4
IE4 Extensions（IE4扩展）	IE4，IE5，IE5.5，IE6(所有版本的一些功能需要Win32 OS)
IE5 Extensions（IE5扩展）	IE5，IE5.5，IE6(所有版本的一些功能需要Win32 OS)
W3C DOM（W3C文档对象模型I、II）	IE5，NN6，IE5.5，Moz1，Safari1（均为部分）

术语：IE4 表示 Internet Explorer 4，NN4 表示 Netscape Navigator 4，Moz1 表示 Mozilla1，其余类推

DOM 不同版本的存在给客户端程序员带来了很多的挑战，编写当前浏览器中最新对象模型支持的 JavaScript 脚本相对比较容易，但如果使用早期版本的浏览器访问这些网页，将会出现不支持某种属性或方法的情况。如果要使设计的网页能运行于绝大多数浏览器中，显而易见将是个难题。因此，W3C DOM 对这些问题做了一些标准化工作，新的文档对象模型继承了许多原始的对象模型，同时还提供了文档对象引用的新方法。

下面介绍在所有支持脚本的浏览器中均可实现的最低公用标准的文档对象模型：基本对象模型。

5.1.1 基本对象模型

基本对象模型提供了一个非常基础的文档对象层次结构（如图 5.2 所示），并最先受到 NN2 的脚本支持。在该模型中，window 位于对象层次的最高级，包括全部的 document 对象，同时具有其他对象所没有的属性和方法，document 就是浏览器载入的 HTML 页面，其上的链接和表单元素如按钮等交互性元素被作为有属性、方法和事件处理程序的元素对象来对待。由于功能十分有限，JavaScript 主要应用于简单的网页操作，如表单合法性验证、获

取程序最后一次修改的时间等等。

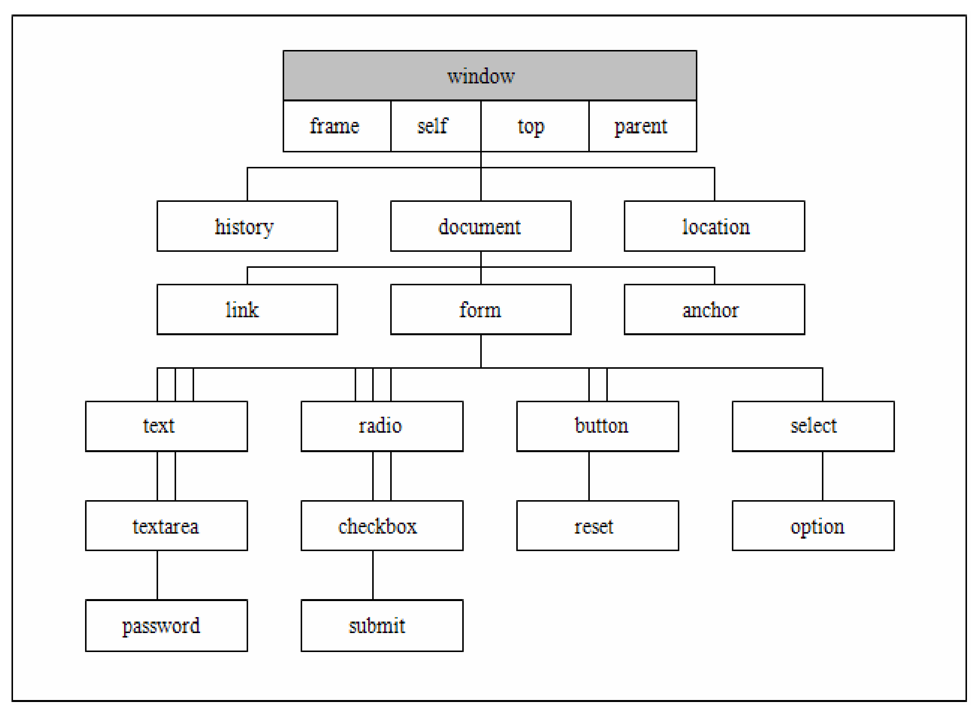

图 5.2 基本文档对象模型

IE3 及其他更高版本的浏览器实现了来自 NN2 的基本对象模型，因此，NN2 后续的浏览器版本的文档对象模型本质是相同的，只不过添加了其他的 window 对象及其操作方法，同时提供了引用原始对象的新方法，如 navigator 和 screen 对象等。

5.1.2 浏览器扩展

在各个版本浏览器中，文档对象模型都有其特殊的地方。一般来说，每发布一个新版本的浏览器，浏览器厂商都会以各种方式扩展 document 对象，新版本修订了老版本的程序错误，同时添加了对象的属性、方法及事件处理程序等，不断扩充原有的功能。

当然，从新对象模型可以更快捷地执行更多任务的技术层面上来看，每次的浏览器版本更新绝对不是一件坏事，但不同浏览器的对象模型朝着不同方向发展，却给 Web 程序员将应用程序在不同浏览器之间移植方面带来了相当的难度，导致 Web 应用程序的跨平台性较差。下面讨论文档对象模型发展过程中主要浏览器版本的对象模型，特别强调各种版本的文档对象模型的新特性以及它们和常用编程任务之间的关系。

1. Netscape Navigator 浏览器

基本对象模型最先在 NN2 种获得支持，虽然功能很有限，这也为文档对象模型的发展奠定了坚实的基础。在 NN3 中通过访问嵌入对象、Applet 应用程序、插件等，使第一个简单、类似于 DHTML 的应用程序的出现成为可能，且脚本语言能访问更多的文档属性和方法。表 5.2 中列出了 NN3 中的 document 对象新增的主要内容：

表 5.2 NN3 中document对象新增主要内容

属性	附加说明
applets[]	文本中的applets数组，用<applet>和</applet>标记

embeds[]	文本中嵌入对象的数组
images[]	文本中图像的数组，用<img>和</img>标记
plugins[]	浏览器中的插件数组
domain	包含web服务器主机名的字符串，仅能改变为更一般的主机名

NN3 中的文档对象模型如图 5.3 所示，其中灰色框内为 NN3 中 document 对象新增内容。

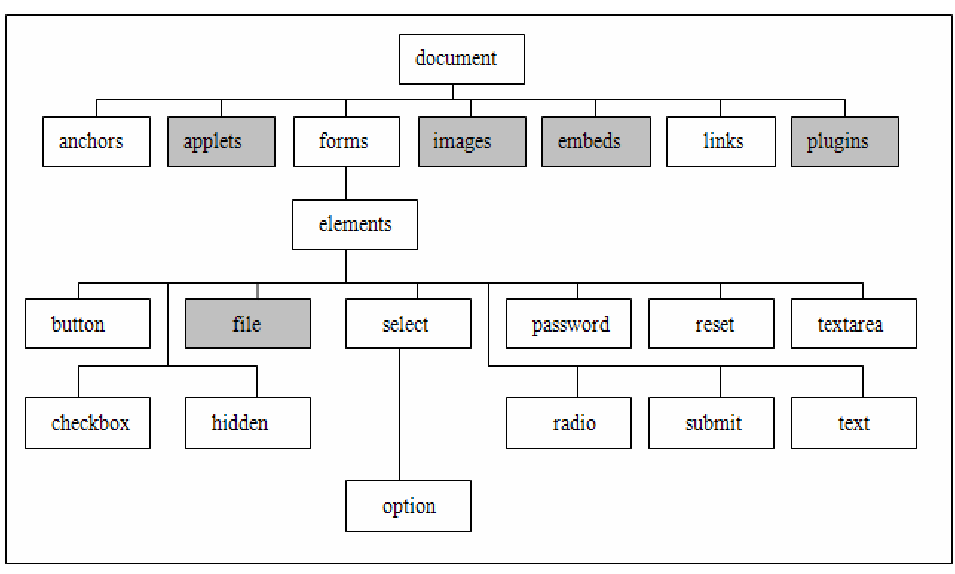

图 5.3 NN3 中 document 对象新增内容

NN3 中增加的最重要对象就是 images 对象，可通过 document.images 得到文档的一个 image 数组，然后通过一下语句进行操作：

```
document.images[n].src=…
```

images 对象的大多数属性都是只读的，而 src 可读可写，典型应用是图片翻转程序，如下代码所示：

```
//源程序 5.2
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<title>Sample</title>
</head>
<body>
<a href="#"
   onmouseover="document.images[0].src='01.jpg'"
   onmouseout="document.images[0].src='02.jpg'">
<img src="01.jpg">
</a>
</body>
</html>
```

在 NN4 之前，Web 应用程序基本不具有动态性，在 NN4 中，新增<layer>标记支持，改进了 Netscape 事件模型、style 对象及其操作方法，表 5.3 中列出了 NN3 中的 Document 对象新增的主要内容：

表 5.3 NN4 中document对象新增主要内容

属性	附加说明

classes	针对HTML标记，创建或使用带有class属性集的CSS样式
Ids	针对HTML标记，创建或使用带有id属性集的CSS样式
tags	针对任意的HTML标记，创建或使用CSS样式
layers[]	文本中的层数组，<layer>标记或<div>定位元素对象

NN4 中的文档对象模型如图 5.4 所示，其中灰色框内为 NN4 中 document 对象新增内容。

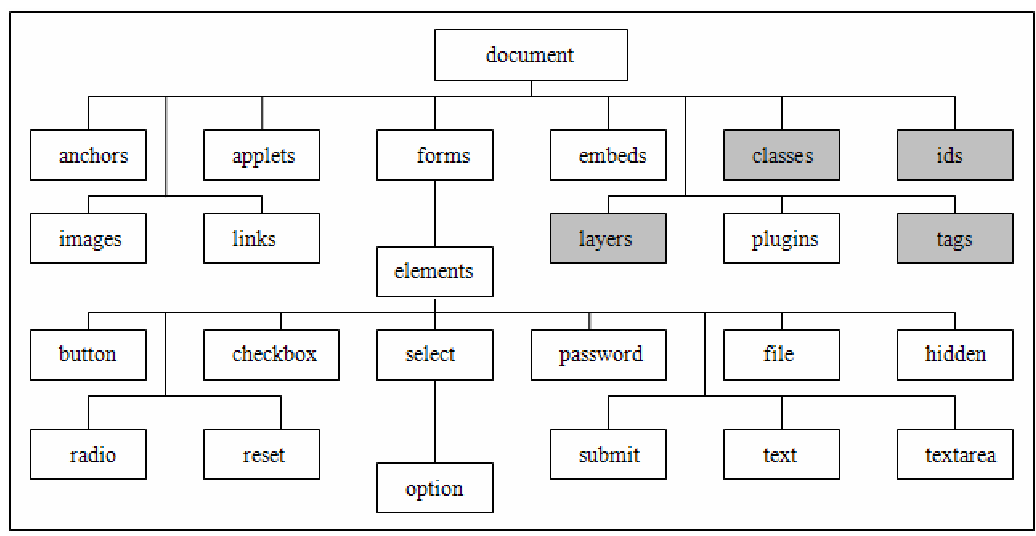

图 5.4 NN4 中 document 对象新增内容

新增对象 layer(层)是一个容器，可以容纳自己的文档，从而拥有自己的 document 对象。当然，这个文档从属于主文档。JavaScript 通过操作它的属性和方法，可以动态改变 layer 的尺寸、位置、隐藏与否等。如果有多个层，还可以更改其堆栈顺序，并且首次允许层覆盖文档中的其它元素。假如需要访问 layer 标记为“MyLayer”的层中标记为“MyPicture”的 image 对象的 src 属性，可通过如下方式访问：

```
document.MyLayer.document.MyPicture.src
```

但是在 W3C DOM 中，layer 对象没被吸收为标准对象，而是用定位 div 对象和 span 对象代替，同时赋予了新的属性、方法和事件处理程序。在 NN4 中，document.layers[]返回文档的 layers[]数组，而在其它浏览器中，则返回 undefined，这也提供了一个鉴别 NN4 浏览器的有效方法。

tags[]属性可以在全局范围内操作某个 HTML 元素对象的样式，语法如下：

```
document.tags.tagName.propertyName
```

其中，tagName 指 HTML 元素对象，propertyName 指要访问的 CSS 属性。例如：

```
<h1 class="site-name title">会员管理系统</h1>
```

如果要改变<h1></h1>之间文本对象的颜色或其它属性，可以通过以下简单的方法：

```
document.tags.h1.color="red"
```

NN6 是 Netscape 浏览器发展的里程碑，它向前兼容 DOM Level 0，也即 W3C 的 DOM 标准，并合并了早期文档对象模型中被广泛使用的特性。同时，它也部分遵循 DOM Level 1 和 DOM Level 2 标准，主要包括 W3C 对于 HTML、XML、CSS 和事件的对象模型。同时它放弃了 NN4 支持但差不多是其特有的扩展内容，如<layer>标记以及对应的 JavaScript 对象，打破了 windows 程序“向下兼容”的规律，导致很多老版本浏览器上支持的脚本在 NN6 中无效。

2. Internet Explorer 浏览器

IE3 是 IE 家族较早支持文档对象模型的浏览器，其对象模型基于 5.1.1 节的基本对象模

型，但是扩展了几个属性，如 frame[]数组等。IE3 中对象模型如图 5.5 所示。

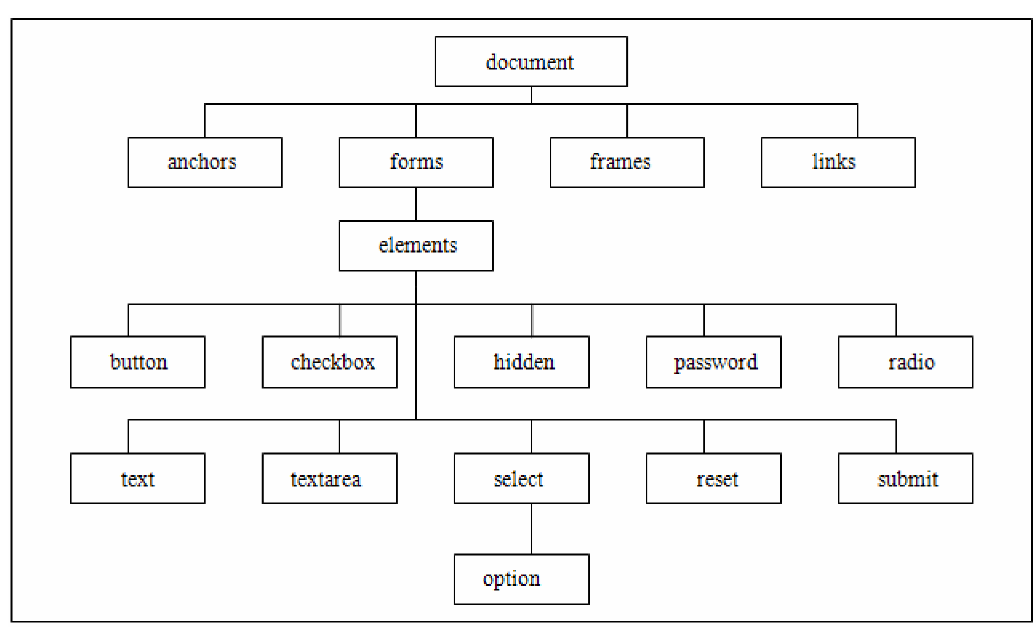

图 5.5 IE3 对象模型结构

IE4 时代，JavaScript 脚本被广泛地运用于 Web 应用程序来实现网页的动态，同时它将每个 HTML 元素都表示为对象。IE4 支持 NN2 和 IE3 的文档对象模型，同时具有许多新的、和 NN4 完全不一样的 document 对象特性，在此，Netscape Navigator 和 IE 这两种使用最为广泛的浏览器开始分道扬镳。表 5.4 列出了 IE4 中的新文本属性。

表 5.4 IE4 中的新文本属性

属性	附加说明
all[]	文档中所有HTML标记的数组
children[]	对象所有子标记数组
embeds[]	文档中嵌入对象的数组，用<embeds>标记
images[]	文档中图像对象数组，用<img>标记
scripts[]	文档的脚本数组，用<script>标记
styleSheets[]	文档中样式（style）对象数组，用<style>标记
applets[]	文档中所有applets数组，用<applets>标记

IE4 中最重要的是引入了 JavaScript 功能部件 document.all[]集合，通过它可以访问文档中的所有对象，通过其特有的检索方式，返回和索引相匹配的对象集合。document.all[]集合拆散了文档对象的层次结构，可对 HTML 文档的任意对象进行快速和简易的访问。IE4 中可通过多种方式快捷访问指定的对象或对象集合：

```
document.all[3];
document.all["name"];
document.all.item("name");
document.all.tags("p");
```

IE4 中给文本对象添加了许多非常使用的新属性和操作的方法，可见 IE4 使动态 Web 应用程序成为现实，提供了对象动态操作以及 HTML 和文本中任意插入、修改和删除等方法。见表 5.5 和表 5.6。

表 5.5 IE4 中文本对象的新属性

属性	附加说明

all[]	对象包含的所有标记的集合
children[]	对象直接派生的标记的集合
innerHTML	包含由对象的标记中包含的HTML内容的字符串，对多数HTML标记可写
innerText	包含由对象的标记中包含的文本内容的字符串，对多数HTML标记可写
outerHTML	包含标记中包含的HTML内容的字符串，对多数HTML标记可写
outerText	包含标记中包含的文本内容的字符串，对多数HTML标记可写
parentElement	对父对象的引用
className	包含对象的CSS类的字符串
style	包含对象的CSS属性的style对象
tagName	包含与对象相关的HTML标记名字的字符串

表 5.6 IE4 中文本对象的新方法

方法	附加说明
Click()	模拟鼠标单击，触发onClick()事件处理器
getAttribute()	返回所指标记的HTML属性的参数
setAttribute()	设置所指标记的HTML属性的参数
removeAttribute()	从标记中删除HTML属性的参数
insertAdjacentHTML()	在标记的前面、后面或当中插入HTML语句
insertAdjacentText()	在标记的前面、后面或当中插入文本

IE4 对象模型结构如图 5.6 所示，其中灰色框内的为 IE4 中 document 对象新增内容。

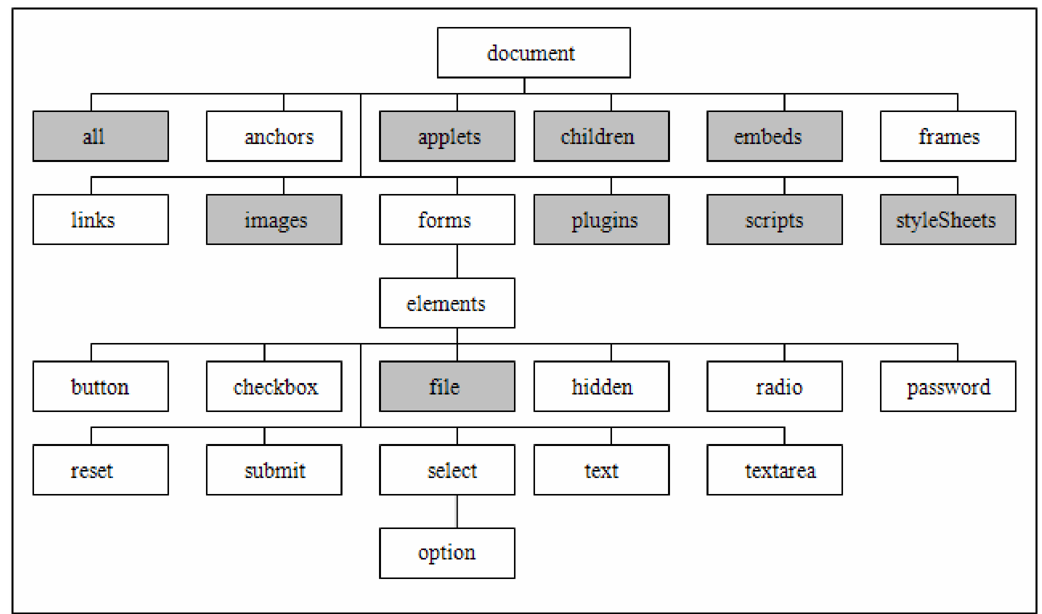

图 5.6 IE4 对象模型结构

下面一段代码综合了 IE4 中文本对象操作的新方法：

```
//源程序 5.3
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<title>Sample</title>
</head>
<body>
<br>
<h1 id="MyTest" align="center">My Test String!<h1><br>
```

```
<form name="MyTestForm" id="MyTestForm">
  <input type="button" value="左对齐" onclick="document.all['MyTest'].align='left'">
  <input type="button" value="右对其" onclick="document.all['MyTest'].align='right'">
  <input type="button" value="中间对齐" onclick="document.all['MyTest'].align='center'"><br>
  <input type="button" value="字体红色" onclick="document.all['MyTest'].style.color='red'">
  <input type="button" value="字体绿色" onclick="document.all['MyTest'].style.color='green'">
  <input type="button" value="字体黑色" onclick="document.all['MyTest'].style.color='black'"><br>
  <input type="text" name="用户内容" id="userText" size="30">
  <input type="button" value="改变字符串内容"
  onclick="document.all['MyTest'].innerText=document.MyTestForm.userText.value"><br>
</form>
</body>
</html>
```

IE5 文档对象模型中与 IE4 极其相似，但对 IE4 进行了功能扩展，增加了对象的可用属性和方法，使得它更为强大，具有更强的文档操作能力。同时，IE5 中的事件处理器数目也大大增加，达到 40 多种，从专门的鼠标和键盘事件到进行剪贴、复制的事件。IE5 中支持两种新的功能部件：DHTML 行为和 HTML 应用程序，前者允许程序员定义可被任何元素反复使用的 DHTML 成分，后者则表现得更像真正的程序而不是 Web 应用程序。

IE5.5、6、7 在继续 IE4 文档对象模型的基础上，在实现 W3C DOM 规范的同时，继续添加只能在 IE 内核浏览器中运行的功能部件，包括新的属性、方法和事件处理程序。从 IE6 开始，完全符合 CSS1 和 DOM Level 1 标准。

较之其他浏览器，IE 对 W3C DOM 标准贯彻得不是很完全，尚有许多有待完善的地方。

3. Opera、Mozilla 和其他浏览器

由于 IE 的漏洞及运行速度问题，很多工程专业的使用者采用 Opera、Mozilla 及其它的浏览器。这些浏览器一般非常严格执行 W3C 和 ECMA 标准，而不采用 IE 和 NN 的特有对象模型(一些也提供对 NN2 和 IE3 对象模型的支持)，主要致力于在 W3C 标准基础上进行开发。

在针对内容和样式的 HTML 和 CSS 的公用规范推出后，DOM 承诺建立统一的范例，用于生成不仅能和网络，而且能和离线的商业或者技术系统进行交互的文档。Netscape 承诺并已经在 NN6 中彻底贯彻了这一标准。Mozilla 具有最好的 DOM 支持，实现了完整的 DOM Level 1、几乎所有的 DOM Level 2 以及部分 DOM Level 3。在 Mozilla 之后，Opera 和 Safari 也在完全支持该标准上做出了相当的努力，极大缩小了与标准之间的差距，支持几乎所有的 DOM Level 1 和大部分 DOM Level 2。

5.1.3 W3C DOM

客户端 Web 应用程序开发人员面对的最大障碍在于 DOM 有很多不同的版本，同时在浏览器版本更替过程中，对象模型又不是统一的，如果需要在不同浏览器环境中运行该网页，将会发现对象的很多属性或方法，甚至某些对象都不起作用。W3C 文档对象模型(DOM)是一个中立的接口语言平台，为程序以及脚本动态地访问和更新文档内容，结构以及样式提供一个通用的标准。它将把整个页面(HTML 或 XML)规划成由节点分层构成的文档，页面的每个部分都是一个节点的衍生物，从而使开发者对文档的内容和结构具有空前的控制力，用 DOM API 可以轻松地删除、添加和替换指定的节点。

DOM 规范必须适应 HTML 的已知结构，同时适应 XML 文档的未知结构。DOM 的概念主要有：

- 核心 DOM：指定类属类型，将带有标记的文档看成树状结构并据此对文档进行相

关操作；

- DOM 事件：包括使用者熟悉的鼠标、键盘事件，同时包括 DOM 特有的事件，当操作文档对象模型中的各元素对象时发生。
- HTML DOM：提供用于操作 HTML 文档以及类似于 JavaScript 对象模型语法的功能部件，在核心 DOM 的基础上支持对所有 HTML 元素对象进行操作。
- XML DOM：提供用于操作 XML 文档的特殊方法，在核心 DOM 的基础上支持对 XML 元素如进程指导、名称空间、CDATA 扇区项等的操作。
- DOM CSS：提供脚本编程实现 CSS 的接口。

在 W3C DOM 规范发展历史上，主要出现了三种不同的版本，下面作简要的介绍。

5.1.4 W3C DOM 规范级别

DOM 规范是一个逐渐发展的概念，规范的发行通常与浏览器发行的时间不很一致，导致任何特定的浏览器的发行版本都只包括最近的 W3C 版本。W3C DOM 经历了三个阶段。

DOM Level 1 是 W3C 于 1998 年 10 月提出的第一个正式的 W3C DOM 规范。它由 DOM Core 和 DOM HTML 两个模块构成。前者提供了基于 XML 的文档的结构图，以方便访问和操作文档的任意部分；后者添加了一些 HTML 专用的对象和方法，从而扩展了 DOM Core。DOM Level 1 的主要目标是合理规划文档的结构。它的最大缺陷就是忽略了事件模型，其中包括 NN2 和 IE3 中最简单的事件模型。

DOM Level 2 基于 DOM Level 1 并扩展了 DOM Level 1，添加了鼠标和用户界面事件、范围、遍历（重复执行 DOM 文档的方法）、XML 命名空间、文本范围、检查文档层次的方法等新概念，并通过对象接口添加了对 CSS 的支持。同时引入几个新模块，用以处理新的接口类型，包括：

- DOM 视图——描述跟踪文档的各种视图（即 CSS 样式化之前和 CSS 样式化之后的文档）的接口；
- DOM 事件——描述事件的接口；
- DOM 样式表——描述处理基于 CSS 样式的接口；
- DOM 遍历和范围——描述遍历和操作文档树的接口。

DOM Level 3 引入了以统一的方式载入和保存文档的方法（包含在新模块 DOM Load and Save 中）以及验证文档（DOM Validation）的方法，从而进一步扩展了 W3C DOM 规范。在 DOM Level 3 中，DOM Core 被扩展为支持所有的 XML 1.0 特性，包括 XML Infoset、XPath 和 XML Base，从而改善了 DOM 对 XML 的支持。

由于 DOM Level 1 在当前浏览器版本中获得最广泛的支持，本章余下部分只针对 DOM Level 1 版本进行讨论。

> 注意：DOM 发展历史上曾出现了 DOM Level 0 的说法，其实 W3C DOM 规范并无此版本，只是 DOM 标准的一个具有试验性质的初级 DOM，主要指第四代浏览器中支持的原始 DHTML，包括对图像进行链接、显示以及在客户端进行表单的数据合法性验证。

5.2 文档对象模型的层次

文档对象模型具有层次结构，由于 JavaScript 是基于对象的编程语言，而不是面向对象的编程语言，所以在 JavaScript 编程中不必考虑类及类的实例、继承等等晦涩难懂的编程术

语，只需充分了解不同浏览器中文档对象模型的层次结构。引用对象的能力决定了代码的功能，而对象则依赖于其在文档对象模型中的层次。知道了对象在文档对象模型中所处的层次，就可以用 JavaScript 准确定位并操作该对象。

举一个很简单的例子：假如一个班的同学在上素描课，指导老师觉得 Group1 的 TOM 的画图纸 BackColor 为 LightGray 比较恰当，就说："Group1 的 TOM，将你的画图纸的 BackColor 改为 LightGray 好吗？"，TOM 听到老师的指示，把画图纸的 BackColor 改为 LightGray。在基于对象的编程方法中，"TOM"是 Object，"ChangeBackColor"为 Command，"LightGray"是 Parameters，可以通过如下的方法实现：

目标对象层次：

```
Group1.TOM
```

ChangeBackColor 可认为是 TOM 的一种方法，所以 TOM 及其方法的完整引用如下：

```
Group1.TOM.ChangeBackColor( )
```

方法需要一种参数去决定是改变成 Color 集合里的什么颜色，所以准确表示该模型的完整命令就是：

```
Group1.TOM.ChangeBackColor(Color.LightGray)
```

该实例的层次结构如图 5.7 所示。

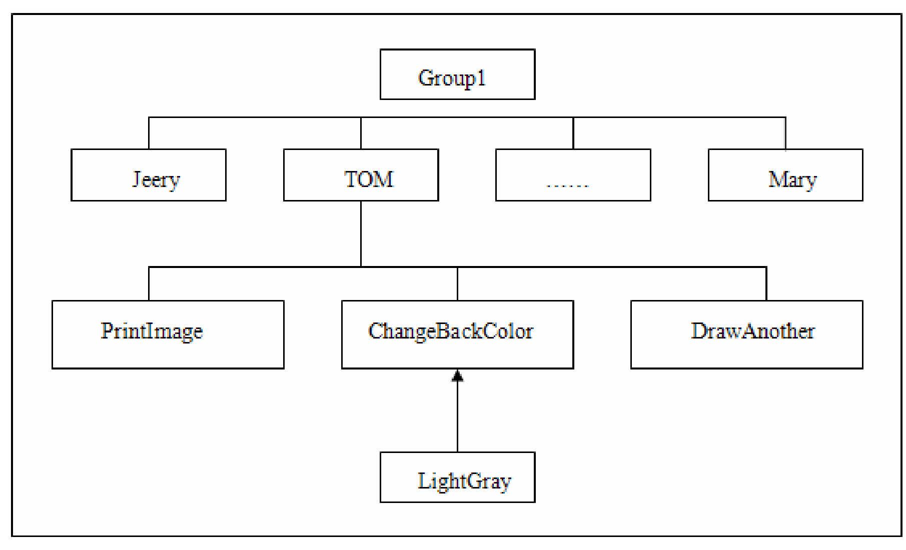

图 5.7 实例中的对象访问结构

如果引用 TOM.ChangeBackColor(Color.LightGray)来表述上述事件，而且班上其它组也有叫 TOM 的，这就很容易带来识别上的混乱。从上述意义来讲，搞清楚文档对象模型对准确定位文档对象并施加相应的操作相当重要。知道了对象所属层次，就知道了访问对象的途径，从而获得对象的操作方法。

5.3 文档对象的产生过程

在面向对象或基于对象的编程语言中，指定对象的作用域越小，对象位置的假定也就越多。对客户端 JavaScript 脚本而言，其对象一般不超过浏览器，脚本不会访问计算机硬件、操作系统、其他程序等其他超出浏览器的对象。

HTML 文档载入时，浏览器解释其代码，当遇到自身支持的 HTML 元素对象对应的标记时，就按 HTML 文档载入的顺序在内存中创建这些对象，而不管 JavaScript 脚本是否真正

运行这些对象。对象创建后，浏览器为这些对象提供专供 JavaScript 脚本使用的可选属性、方法和处理程序。通过这些属性、方法和处理程序，Web 开发人员就能动态操作 HTML 文档内容，下面代码演示如何动态改变文档的背景颜色：

```
//源程序 5.4
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<script language="javascript">
<!--
function changeBgClr(value)
{
  document.body.style.backgroundColor=value
}
//-->
</script>
</head>
<body>
<form>
  <input type=radio value=red onclick="changeBgClr(this.value)">red
  <input type=radio value=green onclick="changeBgClr(this.value)">green
  <input type=radio value=blue onclick="changeBgClr(this.value)">blue
</form>
</body>
</html>
```

其中 document.body.style.backgroundColor 语句表示访问当前 document 对象固有子对象 body 的样式子对象 style 的 backgroundColor 属性。

注意：如果创建一个多框架页面，则直到浏览器载入所有框架时，某个框架内的脚本才能与其它框架进行通信。

5.4 对象的属性和方法

DOM 将文档表示为一棵枝繁叶茂的家谱树，如果把文档元素想象成家谱树上的各个节点的话，可以用同样的记号来描述文档结构模型，在这种意义上讲，将文档看成一棵“节点树”更为准确。在充分认识这棵树之前，先来了解节点的概念。

5.4.1 何谓节点

所谓节点(node)，表示某个网络中的一个连接点，换句话说，网络是节点和连线的集合在 W3C DOM 中，每个容器、独立的元素或文本块都被看着一个节点，节点是 W3C DOM 的基本构建块。当一个容器包含另一个容器时，对应的节点之间有父子关系。同时该节点树遵循 HTML 的结构化本质，如<html>元素包含<head>、<body>，前者又包含<title>，后者包含各种块元素等。DOM 中定义了 HTML 文档中 6 种相关节点类型。所有支持 W3C DOM 的浏览器（IE5+，Moz1 和 Safari 等）都实现了前 3 种常见的类型，其中 Moz1 实现了所有类型。如表 5.7 所示：

表 5.7 DOM定义的HTML文档节点类型

节点类型数值	节点类型	附加说明	实例
1	元素(Element)	HTML标记元素	…
2	属性(Attribute)	HTML标记元素的属性	color="red"
3	文本(Text)	被HTML标记括起来的文本段	Hello World!
8	注释(Comment)	HTML注释段	<!--Comment-->
9	文档(Document)	HTML文档根文本对象	<html>
10	文档类型(DocumentType)	文档类型	<!DOCTYPE HTML PUBLIC "…">

注意：IE6 内核浏览器中属性（attribute）类型在 IE6 版本中才获得支持。

具体来讲，DOM 节点树中的节点有元素节点、文本节点和属性节点等三种不同的类型，下面具体介绍。

1. 元素节点(element node)

在 HTML 文档中，各 HTML 元素如<body>、<p>、<ul>等构成文档结构模型的一个元素对象。在节点树中，每个元素对象又构成了一个节点。元素可以包含其它的元素，例如在下面的“购物清单”代码中：

```
<ul id="purchases">
  <li>Beans</li>
  <li>Cheese</li>
  <li>Milk</li>
</ul>
```

所有的列表项元素<li>都包含在无序清单元素<ul>内部。其中节点树中<html>元素是节点树的根节点。

2. 文本节点(text node)

在节点树中，元素节点构成树的枝条，而文本则构成树的叶子。如果一份文档完全由空白元素构成，它将只有一个框架，本身并不包含什么内容。没有内容的文档是没有价值的，而绝大多数内容由文本提供。在下面语句中：

```
<p>Welcome to<em> DOM </em>World! </p>
```

包含“Welcome to” 、“DOM” 、 “World!”三个文本节点。在 HTML 中，文本节点总是包含在元素节点的内部，但并非所有的元素节点都包含或直接包含文本节点，如“购物清单”中，<ul>元素节点并不包含任何文本节点，而是包含着另外的元素节点，后者包含着文本节点，所以说，有的元素节点只是间接包含文本节点。

3. 属性节点(attribute node)

HTML 文档中的元素或多或少都有一些属性，便于准确、具体地描述相应的元素，便于进行进一步的操作，例如：

```
<h1 class="Sample">Welcome to DOM World！ </h1>
<ul id="purchases">…</ul>
```

这里 class="Sample"、id="purchases"都属于属性节点。因为所有的属性都是放在元素标签里，所以属性节点总是包含在元素节点中。

注意：并非所有的元素都包含属性，但所有的属性都被包含在元素里。

5.4.2 对象属性

属性一般定义对象的当前设置，反映对象的可见属性，如 checkbox 的选中状态，也可

能是不很明显的信息，如提交 form 的动作和方法。在 DOM 模型中，文档对象有许多初始属性，可以是一个单词、数值或者数组，来自于产生对象的 HTML 标记的属性设置。如果标记没有显式设置属性，浏览器使用默认值来给标记的属性和相应的 JavaScript 文本属性赋值。DOM 文档对象主要有如下重要属性，如表 5.8 所示：

表 5.8 文档对象的属性

节点属性	附加说明
nodeName	返回当前节点名字
nodeValue	返回当前节点的值，仅对文本节点
nodeType	返回与节点类型相对应的值，如表5.8
parentNode	引用当前节点的父节点，如果存在的话
childNodes	访问当前节点的子节点集合，如果存在的话
firstChild	对标记的子节点集合中第一个节点的引用，如果存在的话
lastChild	对标记的子节点集合中最后一个节点的引用，如果存在的话
previousSibling	对同属一个父节点的前一个兄弟节点的引用
nextSibling	对同属一个父节点的下一个兄弟节点的引用
attributes	返回当前节点（标记）属性的列表
ownerDocument	指向包含节点（标记）的HTML document对象

注意：firstchild 和 lastchild 指向当前标记的子节点集合内的第一个和最后一个子节点，但是多数情况下使用 childNodes 集合，用循环遍历子节点。如果没有子节点，则 childNodes 长度为 0。

例如如下 HTML 语句：

```
<p id="p1">Welcome to<B> DOM </B>World! </p>
```

可以用如图 5.8 的节点树表示，并标出节点之间的关系。

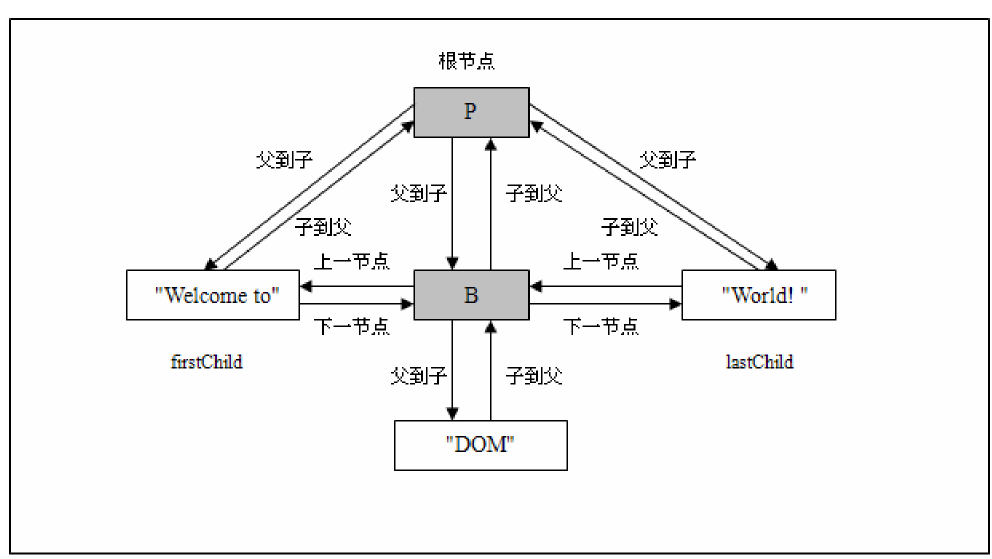

图 5.8 例句的节点树表示

下面的代码演示如何在节点树中按照节点之间的关系检索出各个节点：

```
//源程序 5.5
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
```

```
<title> First Page!</title>
</head>
<body>
<p id="p1">Welcome to<B> DOM </B>World! </p>
<script language="JavaScript" type="text/javascript">
<!--
//输出节点属性
function nodeStatus(node)
{
  var temp="";
  if(node.nodeName!=null)
  {
    temp+="nodeName: "+node.nodeName+"\n";
  }
  else temp+="nodeName: null!\n";
  if(node.nodeType!=null)
  {
    temp+="nodeType: "+node.nodeType+"\n";
  }
  else temp+="nodeType: null\n";
  if(node.nodeValue!=null)
  {
    temp+="nodeValue: "+node.nodeValue+"\n\n";
  }
  else temp+="nodeValue: null\n\n";
  return temp;
}
//处理并输出节点信息
//返回 id 属性值为 p1 的元素节点
var currentElement=document.getElementById('p1');
var msg=nodeStatus(currentElement);
//返回 p1 的第一个孩子，即文本节点“Welcome to”
currentElement=currentElement.firstChild;
msg+=nodeStatus(currentElement);
//返回文本节点“Welcome to”的下一个同父节点，即元素节点 B
currentElement=currentElement.nextSibling;
msg+=nodeStatus(currentElement);
//返回元素节点 B 的第一个孩子，即文本节点“DOM”
currentElement=currentElement.firstChild;
msg+=nodeStatus(currentElement);
//返回文本节点“DOM”的父节点，即元素节点 B
currentElement=currentElement.parentNode;
msg+=nodeStatus(currentElement);
//返回元素节点 B 的同父节点，即文本节点“Welcome to”
currentElement=currentElement.previousSibling;
msg+=nodeStatus(currentElement);
//返回文本节点“Welcome to”的父节点，即元素节点 P
currentElement=currentElement.parentNode;
msg+=nodeStatus(currentElement);
//返回元素节点 P 的最后一个孩子，即文本节点“World!”
currentElement=currentElement.lastChild;
msg+=nodeStatus(currentElement);
//输出节点属性
```

```
alert(msg);
//-->
</script>
</body>
</html>
```

运行上述代码，结果如图 5.9 所示，null 指某个节点没有对应的属性。

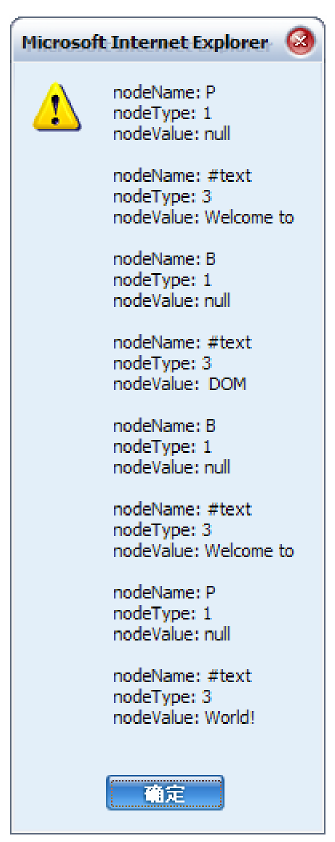

图 5.9 节点数的遍历方法实例

> 注意：遍历浏览器载入 HTML 文档形成的节点树时，可通过 document.documentElement 属性来定位根节点，即<html>标记。

在准确定位节点树中的某个节点后，就可以使用对象的方法来操作这个节点，下面介绍对象(节点)的操作方法。

5.4.3 对象方法

对象方法是脚本给该对象的命令，可以有返回值，也可没有，且不是每个对象都有方法定义。DOM 中定义了操作节点的一系列行之有效的方法，让 Web 应用程序开发者真正做到随心所欲地操作 HTML 文档中各个元素对象，先来了解 id 属性和 class 属性。

5.4.4 id 属性和 class 属性

在图 5.1 所示的家谱树中，HTML 文档载入时各元素对象都被标注成节点，同时根据浏览器载入的顺序，自动分配一个序号，可用 document.all[ ]直接访问。考察如下的实例：

```
//源程序 5.6
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title> First Page!</title>
</head>
<body>
<h5>Test!</h5>
<!--NOTE!-->
<p>Welcome to<em> DOM </em>World! </p>
<ul>
  <li>Newer</li>
</ul>
<hr>
<br>
<script language="JavaScript" type="text/javascript">
<!--
  var i,origlength;
  //获取 document.all[ ]数组的长度
  origlength=document.all.length;
  document.write('document.all.length='+origlength+"<br>");
  //循环输出各节点的 tagName 属性值
  for(i=0;i<origlength;i++)
  {
    document.write("document.all["+i+"]="+document.all[i].tagName+"<br>");
  }
//-->
</script>
</body>
</html>
```

程序运行结果如图 5.10 所示，可以看出浏览器按载入顺序为每个 HTML 元素都分配了一个序号来访问对应的元素节点。

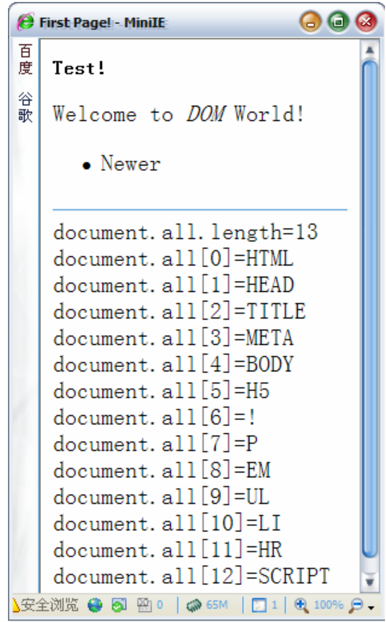

图 5.10 HTML 载入生成的 document.all[]数组

注意：如果使用 document.all.length 对循环进行检查，程序会出现死循环，因为每次输出正在检查的元素时，document.all[]元素个数都在增加。

由于不能直观看出 document.all[]数组中各元素的序号分布，甚至在文档最终完成之前根本无法获知节点树的架构，这种访问方法显然不能很方便、快捷地进行元素节点访问，现在引入 id 属性和 class 属性。

id 属性的用途是给 HTML 文档中的某个特定的元素对象加上独一无二（对当前文档而言）的标识符，便于精确访问这个元素对象。例如：

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

在 CSS 中，可以为有着特定 id 的元素对象定义一种独享的样式：

```
#p1{
  border:1px solid white;
  background-color:#333;
  color:#ccc;
  padding:1em;
}
```

每个元素对象只能有一个 id 属性值，不同的元素对象必须有不同的 id 属性值。也可利用 id 属性为包含在某给定元素里的其他元素定义样式，这样定义的样式将只作用于包含在给定元素里的指定元素。在前述的“购物清单”代码中，可通过如下方式定义
和
之间文本的样式：

```
#purchases li{
  fontweight:bold;
}
```

id 属性就是一座桥梁，连接着文档中的某个元素和 CSS 样式表中的特定样式，同时实现元素对象的相关操作。

所有的元素都有 class 属性，不同的元素可以有相同的 class 属性值，例如：

```
<p class="MyClass">The First Line</p>
<h1 class="MyClass">The Second Line</h1>
```

可以通过如下方式定义<p>和<h1>的共享样式：

```
.special{
  text-transform:uppercase;
}
```

同时，也可以通过 h1.special 方式定位第二个文本对象并改变它的样式，获取更为精确的控制。

5.4.5 getElementById( )方法

该方法返回与指定 id 属性值的元素节点相对应的对象，对应着文档里一个特定的元素节点（元素对象）。该方法是与 document 对象相关联的函数，其语法如下：

```
document.getElementById(id)
```

其中 id 为要定位的对象 id 属性值。

下面的例子演示 getElementById( )方法的使用，同时可以看出其返回一个对象(object)，而不是数值、字符串等。

```
//源程序 5.7
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
```

```
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title> First Page!</title>
</head>
<body>
<ul id="purchases">
  <li>Beans</li>
  <li>Cheese</li>
  <li>Milk</li>
</ul>
<script language="JavaScript" type="text/javascript">
<!--
document.write(typeof document.getElementById("purchases"));
//-->
</script>
</body>
</html>
```

一般来说，我们不必为 HTML 文档中的每一个元素对象都定义一个独一无二的 id 属性值，也可通过下面的 getElementByTagName( )方法准确定位文档中特定的元素。

> 注意： 1、JavaScript 对大小写敏感， getElementById 写成 GetElementById 、getelementById 等都不对。
> 2、typeof 返回数据的类型，如数值、对象、字符串等。

5.4.6 getElementByTagName( )方法

该方法返回文档里指定标签 tag 的元素对象数组，与上述的 getElementById( )方法返回对象不同，且返回的对象数组中每个元素分别对应文档里一个特定的元素节点（元素对象）。其语法如下：

```
element. getElementByTagName(tag)
```

其中 tag 为指定的标签。下面给出的例子演示该方法返回的是对象数组，而不是对象。

```
var items=document.getElementByTagName("li");
for(var i=0;i<items.length;i++)
{
  document.write(typeof item[i]);
}
```

将上述的代码替换前面购物清单<script></script>之间的语句，可以看出该方法返回对象(object)数组，长度为 3。再看下面的代码：

```
var shoplist=document.getElementById("purchases");
var items=shoplist.getElementByTagName("*")
var i=items.length;
```

以上语句运行后，items 数组将只包含 id 属性值为 purchases 的无序清单里的元素，i 返回 3，与列表项元素个数相同。

由于对象数组中定位对象需要事先知道对象对应的下标号，DOM 提供了直接通过元素对象名称进行访问的方法，即 getElementByName( )方法。

> 注意：可以用参数“*”来获取文档中所有的元素，但此时 IE6 内核的浏览器并不犯会所有的元素，必须使用 document.all[]来获取。

5.4.7 getElementByName( )方法

相对于 id 属性值，旧版本的 HTML 文档更习惯于对<form>、<select>等元素节点使用 name 属性。此时需要用到文档对象的 getElementByName( )方法来定位。该方法返回指定名称 name 的节点序列，其语法如下：

```
Document. getElementByName(name)
```

其中 name 为指定要定位的元素对象的名字，下面的代码演示其使用方法：

```
var MyList=document.getElementByName("MyTag");
var temp=" ";
for(var i=0;i<MyList.length;i++)
{
    temp+="nodeName: "+node.nodeName+"\n";
    temp+="nodeType: "+node.nodeType+"\n";
    temp+="nodeValue: "+node.nodeValue+"\n";
}
return temp;
```

在准确定位到特定元素对象后，可通过 getAttribute( )方法将它的各种属性值查询出来。

> 注意：Opera 7.5、IE 6.0 及使用 IE6 内核的浏览器在使用此方法时有很大的不同，首先，它们返回 id 为 name 的元素；其次，仅仅检查<input>和<img>元素。该方法由于浏览器的支持问题，不是很常见。

5.4.8 getAttribute( )方法

该方法返回目标对象指定属性名称的某个属性值。语法如下：

```
object.getAttribute(attribute)
```

其中 attribute 为对象指定要搜索的属性，下面的代码演示其使用方法：

```
//源程序 5.8
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body>
<p title="First Sample">This is the first Sample!</p>
<script language="JavaScript" type="text/javascript">
<!--
var objSample=document.getElementByTagName("p");
for(var i=0;i<objSample.length;i++)
{
   document.write(objSample[i].getAttribute("title"));
}
//-->
</script>
</body>
</html>
```

上述代码通过 objSample.length 控制循环，遍历整个文档的<p>标记。运行结果显示为“First Sample”。

以上从节点定位到获得其指定的属性值，都只能检索信息。下面介绍指定节点的属性值进行修改的途径：setAttribute( )方法。

5.4.9 setAttribute( )方法

该方法可以修改任意元素节点指定属性名称的某个属性值，语法如下：

```
object.setAttribute(attribute,value)
```

类似于 getAttribute( )方法，setAttribute( )方法也只能通过元素节点对象调用，但是需要传递两个参数：

- attribute：指定目标节点要修改的属性
- value：属性修改的目标值

下面的代码演示其功能：

```
//源程序 5.9
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body>
<ul id="purchases">
  <li>Beans</li>
  <li>Cheese</li>
  <li>Milk</li>
</ul>
<script language="JavaScript" type="text/javascript">
<!--
  var shoplist=document.getAttributeById("purchases");
  document.write(shoplist.getAttribute("title"));
  shoplist.setAttribute("title", "New List");
//-->
</script>
</body>
</html>
```

运行结果显示 null 和 New List，因为 id 属性值为 purchases 的 ul 元素节点的 title 属性在 shoplist.setAttribute("title","New List")代码运行之前根本不存在，所以显示 null；运行后，修改 title 属性为“New List”。这意味着至少完成了两个步骤：

（1）创建 ul 元素节点的 title 属性；

（2）设置刚创建的 title 属性值；

当然，如果 title 属性值本来就存在，运行 shoplist.setAttribute("title","New List")后，title 原来的属性值被“New List”覆盖。

注意：通过 setAttribute( )方法对文档做出的修改，将使浏览器窗口的显示效果、行为动作等发生相应的变化，这是一个动态的过程。但是这种修改并不反应到文档本身的物理内容上。这由 DOM 的工作模式决定：先加载文档静态内容，再以动态的方式对文档进行刷新，动态刷新不影响文档的静态内容。客户端用户不需要手动执行页面刷新操作就能动态刷新页面。

5.4.10 removeAttribute( )方法

该方法可以删除任意元素节点指定的属性，语法如下：

```
object.removeAttribute(name)
```

类似于 getAttribute( )和 setAttribute( )方法，removeAttribute( )方法也只能通过元素节点对象调用。其中 name 标示要删除的属性名称，例如：

```
//源程序 5.10
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function TestEvent()
{
  document.MyForm.text1.removeAttribute("disabled");
}
//-->
</script>
</head>
<body>
<form name="MyForm">
  <input type="text" name="text1" value="red" disabled>
  <input type="button" name="MyButton" value="MyTestButton" onclick="TestEvent()">
</form>
</body>
</html>
```

运行上述代码，单击“MyTestButton” 按钮之前，文本框 text1 显示为只读属性；单击“MyTestButton”按钮后，触发 TestEvent( )函数执行核心语句：

```
document.MyForm.text1.removeAttribute("disabled");
```

该语句删除 text1 的 disabled 属性，文本框变为可用状态。

5.5 附加的节点处理方法

由于文本节点具有易于操纵、对象明确等特点，DOM Level 1 提供了非常丰富的节点处理方法，如表 5.9 所示：

表 5.9 DOM中的节点处理方法

操作类型	方法原型	附加说明
生成节点	createElement(tagName)	创建由tagName指定类型的标记
	CreateTextNode(string)	创建包含字符创string的文本节点
	createAttribute(name)	针对节点创建由name指定的属性，不常用
	createComment(string)	创建由字符串string指定的文本注释
插入和添加节点	appendChild(newChild)	添加子节点newChild到目标节点上
	insertBefore(newChild,targetChild)	将新节点newChild插入目标节点targetChild之前
复制节点	cloneNode(bool)	复制节点自身，由逻辑量bool确定是否复制子节点

删除和替换节点	removeChild(childName)	删除由childName指定的节点
	replaceChild(newChild,oldChild)	用新节点newChild替换旧节点oldChild
文本节点操作	insertData(offset,string)	从由offset指定的位置插入string值
	appendData(string)	将string值插入到文本节点的末尾处
	deleteData(offset,count)	从由offset指定的位置删除count个字符
	replaceData(offset,count,string)	从由offset指定的位置用string代替count个字符
	splitText(offset)	从由offset指定的位置将文本节点分成两个文本节点，左边更新为原始节点，右边的返回到新节点
	substringData(offset,count)	返回从offset指定的位置开始的count个字符

DOM 中指定的节点处理方法，提供了 Web 应用程序开发者快捷、动态更新 HTML 页面的途径。下面通过具体实例来说明各种方法的使用。

5.5.1 生成节点

DOM 中提供的方法生成新节点的操作非常简单，语法分别如下：

```
MyElement=document.createElement("h1")
MyTextNode=document.createTextNode("My Text Node!")
MyComment=document.createComment("My Comment!")
```

下面的实例演示如何生成新节点并验证：

```
//源程序 5.11
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<script language="JavaScript" type="text/javascript">
<!--
function nodeStatus(node)
{
  var temp="";
  if(node.nodeName!=null)
  {
    temp+="nodeName: "+node.nodeName+"\n";
  }
  else temp+="nodeName: null!\n";
  if(node.nodeType!=null)
  {
    temp+="nodeType: "+node.nodeType+"\n";
  }
  else temp+="nodeType: null\n";
  if(node.nodeValue!=null)
  {
    temp+="nodeValue: "+node.nodeValue+"\n\n";
  }
  else temp+="nodeValue: null\n\n";
  return temp;
}
function MyTest( )
{
  //产生 p 元素节点和新文本节点
```

```
    var newParagraph = document.createElement("p");
    var newTextNode= document.createTextNode(document.MyForm.MyField.value);
    var msg=nodeStatus(newParagraph);
    msg+=nodeStatus(newTextNode)
    alert(msg);
    return;
}
//-->
</script>
<body>
<form name="MyForm">
    <input type="text" name="MyField" value="My Sample">
    <input type="button" value="TestButton" onclick="MyTest()">
</body>
</html>
```

上面代码运行后，单击“TestButton”按钮，触发 MyTest()函数，结果如图 5.11 所示。

图 5.11 实例生成的元素节点和文本节点

生成节点后，要将节点添加到 DOM 树中，下面介绍插入和添加节点的方法。

5.5.2 插入和添加节点

把新创建的节点插入到文档的节点树最简单的方法就是让它成为该文档某个现有节点的子节点，appendChild(newChild)作为要添加子节点的节点的方法被调用，将一个标识为 newChild 的节点添加到它的子节点的末尾。语法如下：

```
object.appendChild(newChild)
```

下面的实例演示如何在节点树中插入节点：

```
//源程序 5.12
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<script language="JavaScript" type="text/javascript">
<!--
```

```
function nodeStatus(node)
{
   var temp="";
   if(node.nodeName!=null)
   {
      temp+="nodeName: "+node.nodeName+"\n";
   }
   else temp+="nodeName: null!\n";
   if(node.nodeType!=null)
   {
      temp+="nodeType: "+node.nodeType+"\n";
   }
   else temp+="nodeType: null\n";
   if(node.nodeValue!=null)
   {
      temp+="nodeValue: "+node.nodeValue+"\n\n";
   }
   else temp+="nodeValue: null\n\n";
   return temp;
}
function MyTest( )
{
   //产生 p 元素节点和新文本节点，并将文本节点添加为 p 元素节点的最后一个子节点
   var newParagraph = document.createElement("p");
   var newTextNode= document.createTextNode(document.MyForm.MyField.value);
   newParagraph.appendChild(newTextNode);
   var msg=nodeStatus(newParagraph);
   msg+=nodeStatus(newTextNode);
   msg+=nodeStatus(newParagraph.firstChild);
   alert(msg);
   return;
}
//-->
</script>
<body>
<form name="MyForm">
   <input type="text" name="MyField" value="My Sample">
   <input type="button" value="TestButton" onclick="MyTest()">
</body>
</html>
```

程序运行结果如图 5.12 所示。

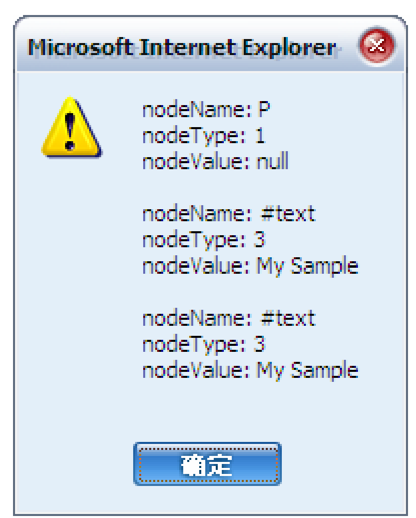

图 5.12 改写后的实例生成的元素节点和文本节点

明显看出使用 newParagraph.appendChild(newTextNode)语句后，节点 newTextNode 和节点 newParagraph.firstChild 表示同一节点，证明生成的文本节点已经添加到<p>元素节点的子节点列表中。

insertBefore(newChild,targetChild)方法将文档中一个新节点 newChild 插入到原始节点 targetChild 前面，语法如下：

```
parentElement.insertBefore(newChild,targetChild)
```

调用此方法之前，要明白三点：

- 要插入的新节点 newChild
- 目标节点 targetChild
- 这两个节点的父节点 parentElement

其中，parentElement=targetChild.parentNode，且父节点必须是元素节点。以下面的语句为例：

```
<p id="p1">Welcome to<B> DOM </B>World! </p>
```

其表示的节点树如图 5.8 所示。下面的代码演示如何在文本节点“Welcome to”之前添加一个同父文本节点“NUDT YSQ”：

```
//源程序 5.13
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title> First Page!</title>
</head>
<body>
<p id="p1">Welcome to<B> DOM </B>World! </p>
<script language="JavaScript" type="text/javascript">
<!--
function nodeStatus(node)
{
  var temp="";
  if(node.nodeName!=null)
  {
```

```
    temp+="nodeName: "+node.nodeName+"\n";
  }
  else temp+="nodeName: null!\n";
  if(node.nodeType!=null)
  {
    temp+="nodeType: "+node.nodeType+"\n";
  }
  else temp+="nodeType: null\n";
  if(node.nodeValue!=null)
  {
    temp+="nodeValue: "+node.nodeValue+"\n\n";
  }
  else temp+="nodeValue: null\n\n";
  return temp;
}
//输出节点树相关信息
//返回 id 属性值为 p1 的元素节点
var parentElement=document.getElementById('p1');
var msg="insertBefore 方法之前:\n"
msg+=nodeStatus(parentElement);
//返回 p1 的第一个孩子，即文本节点“Welcome to”
var targetElement=parentElement.firstChild;
msg+=nodeStatus(targetElement);
//返回文本节点“Welcome to”的下一个同父节点，即元素节点 B
var currentElement=targetElement.nextSibling;
msg+=nodeStatus(currentElement);
//返回元素节点 P 的最后一个孩子，即文本节点“World!”
currentElement=parentElement.lastChild;
msg+=nodeStatus(currentElement);
//生成新文本节点“NUDT YSQ”，并插入到文本节点“Welcome to”之前
var newTextNode= document.createTextNode("NUDT YSQ");
parentElement.insertBefore(newTextNode,targetElement);
msg+="insertBefore 方法之后:\n"+nodeStatus(parentElement);
//返回 p1 的第一个孩子，即文本节点“NUDT YSQ”
targetElement=parentElement.firstChild;
msg+=nodeStatus(targetElement);
//返回文本节点“Welcome to”的下一个同父节点，即元素节点“Welcome to”
var currentElement=targetElement.nextSibling;
msg+=nodeStatus(currentElement);
//返回元素节点 P 的最后一个孩子，即文本节点“World!”
currentElement=parentElement.lastChild;
msg+=nodeStatus(currentElement);
//输出节点属性
alert(msg);
//-->
</script>
</body>
</html>
```

输出信息按照父节点、第一个子节点、下一个子节点、最后一个子节点的顺序显示，程序运行结果如图 5.13 所示。

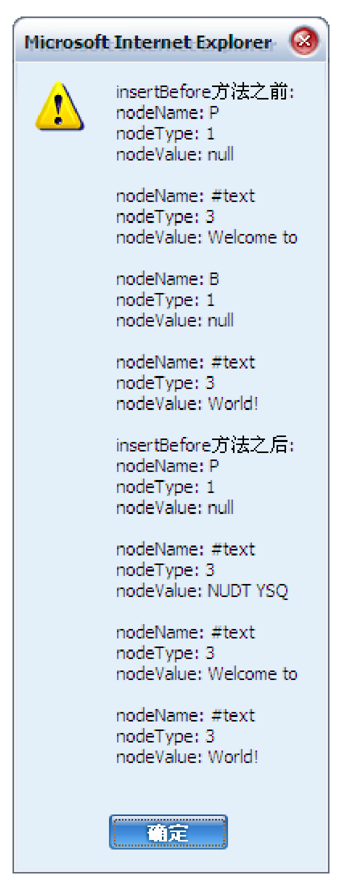

图 5.13 使用 insertBefore( )方法在目标节点之前插入新节点

可以很直观看出文本节点“Welcome to”在作为 insertBefore( )方法的目标节点后，在其前面插入文本节点“NUDT YSQ”作为<p>元素节点的第一子节点。

DOM 本身并没有提供类似 insertBefore(newChild,targetChild)方法在节点之后插入新节点方法 insertAfter(newChild,targetChild)，但是可以通过如下形式实现：

```
function insertAfter(newChild,targetChild)
{
    var parentElement=targetChild.parentNode;
    //检查目标节点是否是父节点的最后一个子节点
    //是：直接按 appendChild( )方法插入新节点
    if(parentElement.lastChild==targetChild)
    {
        parentElement.appendChild(newChild);
    }
    //不是：使用目标节点的 nextSibling 属性定位到它的下一同父节点，按 insertBefore( )方法操作
    else
        parentElement.insertBefore(newChild,targetElement.nextSibling);
}
```

将源程序 5.13 中新节点插入语句：

```
parentElement.insertBefore(newTextNode,targetElement);
```

改写为：

```
insertAfter(newTextNode,targetElement);
```

程序运行结果如图 5.14 所示。

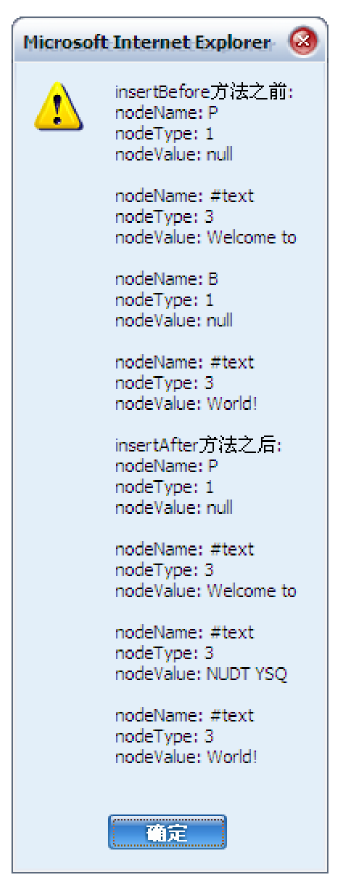

图 5.14 使用 insertAfter( )函数在目标节点之后插入新节点

可以直观看出，insertAfter( )函数实现了在目标节点后面插入同级子节点的功能，扩展了 DOM 关于节点插入和添加的方法。

5.5.3 复制节点

有时候并不需要生成或插入新的节点，而只是复制就可以达到既定的目标。DOM 提供 cloneNode( )方法来复制特定的节点，语法如下：

```
clonedNode=targetNode.cloneNode(bool)
```

其中参数 bool 为逻辑量：

- bool=1 或 true：表示复制节点自身的同时复制节点所有的子节点；
- bool=0 或 false：表示仅仅复制节点自身。

下面的实例演示使用如何复制节点并将其插入到节点树中：

```
//源程序 5.14
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title> First Page!</title>
</head>
<body>
<p id="p1">Welcome to<B> DOM </B>World! </p>
<script language="JavaScript" type="text/javascript">
```

```
<!--
function nodeStatus(node)
{
   var temp="";
   if(node.nodeName!=null)
   {
      temp+="nodeName: "+node.nodeName+"\n";
   }
   else temp+="nodeName: null!\n";
   if(node.nodeType!=null)
   {
      temp+="nodeType: "+node.nodeType+"\n";
   }
   else temp+="nodeType: null\n";
   if(node.nodeValue!=null)
   {
      temp+="nodeValue: "+node.nodeValue+"\n\n";
   }
   else temp+="nodeValue: null\n\n";
   return temp;
}
//输出节点树相关信息
//返回 id 属性值为 p1 的元素节点
var parentElement=document.getElementById('p1');
var msg="insertBefore 方法之前:\n"
msg+=nodeStatus(parentElement);
//返回 p1 的第一个孩子，即文本节点“Welcome to”
var targetElement=parentElement.firstChild;
msg+=nodeStatus(targetElement);
//返回文本节点“Welcome to”的下一个同父节点，即元素节点 B
var currentElement=targetElement.nextSibling;
msg+=nodeStatus(currentElement);
//返回元素节点 P 的最后一个孩子，即文本节点“World!”
currentElement=parentElement.lastChild;
msg+=nodeStatus(currentElement);
//通过目标节点的 cloneNode( )方法产生复制后的新节点
var ClonedNode=targetElement.cloneNode(false);
//使用父节点的 insertBefore( )方法将新节点插入到目标节点的前面
parentElement.insertBefore(ClonedNode,targetElement);
msg+="insertBefore 方法之后:\n"+nodeStatus(parentElement);
//返回 p1 的第一个孩子，即克隆的新文本节点“Welcome to”
targetElement=parentElement.firstChild;
msg+=nodeStatus(targetElement);
//返回文本节点“Welcome to”的下一个同父节点，即元素节点“Welcome to”
var currentElement=targetElement.nextSibling;
msg+=nodeStatus(currentElement);
//返回元素节点 P 的最后一个孩子，即文本节点“World!”
currentElement=parentElement.lastChild;
msg+=nodeStatus(currentElement);
//输出节点属性
alert(msg);
//-->
</script>
```

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

程序运行结果如图 5.15 所示。

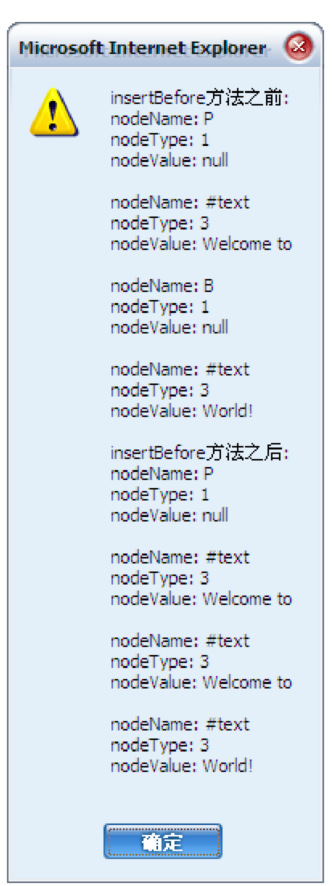

图 5.15 使用 cloneNode( )方法进行节点复制操作

> 注意：HTML 文档中，空元素不会改变文档的外观，因为浏览器经常对那些无内容的元素进行最小化。

5.5.4 删除和替换节点

可以在节点树中生成、添加、复制一个节点，当然也可以删除节点树中特定的节点。DOM 提供 removeChild()方法来进行删除操作，语法如下：

```
removeNode=object.removeChild(name)
```

参数 name 指明要删除的节点名称，该方法返回所删除的节点对象。

下面的实例演示如何使用 removeChild()方法删除节点：

```
//源程序 5.15
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title> First Page!</title>
</head>
<body>
<p id="p1">Welcome to<B> DOM </B>World! </p>
```

```
<script language="JavaScript" type="text/javascript">
<!--
function nodeStatus(node)
{
   var temp="";
   if(node.nodeName!=null)
   {
      temp+="nodeName: "+node.nodeName+"\n";
   }
   else temp+="nodeName: null!\n";
   if(node.nodeType!=null)
   {
      temp+="nodeType: "+node.nodeType+"\n";
   }
   else temp+="nodeType: null\n";
   if(node.nodeValue!=null)
   {
      temp+="nodeValue: "+node.nodeValue+"\n\n";
   }
   else temp+="nodeValue: null\n\n";
   return temp;
}
var parentElement=document.getElementById('p1');
var msg="父节点：\n"+nodeStatus(parentElement);
//返回元素节点 P 的最后一个孩子，即文本节点“World!”
currentElement=parentElement.lastChild;
msg+="删除前:lastChild:\n"+nodeStatus(currentElement);
//删除节点 P 的最后一个孩子，即文本节点“World!”，最后一个孩子变为 B
parentElement.removeChild(currentElement);
currentElement=parentElement.lastChild;
msg+="删除后:lastChild:\n"+nodeStatus(currentElement);
//输出节点属性
alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 5.16 所示。

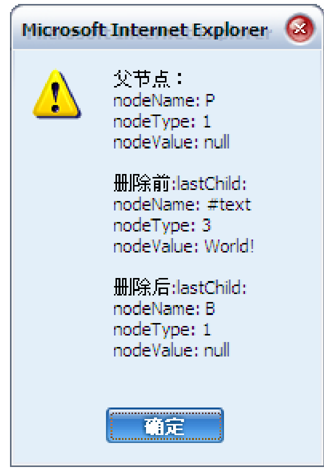

图 5.16 使用 removeChild( )方法删除子节点

DOM 中使用 replaceChild( )来替换指定的节点，语法如下：

```
object.replaceChild(newChild,oldChild)
```

其中参数：

- newChild：新添加的节点
- oldChild：被替换的目标节点

将下面的代码替换源程序 5.15 中的输出部分：

```
var parentElement=document.getElementById('p1');
var msg="父节点：\n"+nodeStatus(parentElement);
//返回元素节点 P 的最后一个孩子，即文本节点“World!”
currentElement=parentElement.lastChild;
msg+="替换前:lastChild:\n"+nodeStatus(currentElement);
//生成将用来替换的新文本节点 newTextNode
var newTextNode= document.createTextNode("NUDT YSQ");
parentElement.replaceChild(newTextNode,currentElement)
//替换文本节点“World!”为“NUDT YSQ”
currentElement=parentElement.lastChild;
msg+="替换后:lastChild:\n"+nodeStatus(currentElement);
//输出节点属性
alert(msg);
```

程序运行结果如图 5.17 所示。

图 5.17 使用 replaceChild( )方法替换指定节点

如果只是要改变文本节点的内容，只需要给该节点的 nodeValue 属性赋一个新字符串值即可，将下面的代码替换源程序 5.15 中的输出部分：：

```
var parentElement=document.getElementById('p1');
var msg="父节点:\n"+nodeStatus(parentElement);
//返回 p1 的第一个孩子，即文本节点“Welcome to”
var targetElement=parentElement.firstChild;
msg+="nodeValue 更改之前:\n"+nodeStatus(targetElement);
//将文本节点“Welcome to”赋新值“Verify String!”
targetElement.nodeValue= "Verify String!";
//返回 p1 的第一个孩子，即新文本节点“Verify String!”
msg+="nodeValue 更改之后:\n"+nodeStatus(targetElement);
//输出节点属性
```

```
alert(msg);
```

程序运行结果如图 5.18 所示。

图 5.18 修改文本节点的 nodeValue 值更改文本内容

当节点内容完全是文本时，此方法修改节点内容最为直接。如果不存在文本节点，则不起任何作用。下面专门介绍 DOM 中文本节点的特有操作方法。

> 注意：通过 createTextNode( )方法产生的文本节点没有任何内在样式，如果要改变文本节点的外观及文本，就必须修改该文本节点的父节点的 style 属性。执行样式更改和内容变化的浏览器将自动刷新此网页，以适应文本节点样式和内容的变化。

5.5.5 文本节点操作

针对文本节点，DOM 提供了多种方法来操作其文本属性，下面的例子演示各个函数的具体使用方法：

```
//源程序 5.16
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title> First Page!</title>
</head>
<body>
<p id="p1">Congratulations</p>
<script language="JavaScript" type="text/javascript">
<!--
function nodeStatus(node)
{
  var temp="";
  if(node.length!=0)
  {
    temp+="nodeLength: "+node.length+"\n";
  }
  else temp+="nodeLength: 0\n\n";
```

```
    if(node.data!="")
    {
      temp+="nodeData: "+node.data+"\n\n";
    }
    else temp+="nodeData: null\n\n";
    return temp;
}
//返回 id 属性值为 p1 的元素节点
var parentElement=document.getElementById('p1');
//返回 p1 的第一个孩子，即文本节点“Congratulations”，并输出其信息
var targetElement=parentElement.firstChild;
var msg="●原始文本节点内容:\n"+nodeStatus(targetElement);
//使用 targetElement 的 data 属性直接修改其内容
targetElement.data="Welcome you";
msg+="●使用 targetElement.data="Welcome you":\n"+nodeStatus(targetElement);
//使用 appendData( )方法在文本节点内容后面添加字符串
targetElement.appendData(' to NUDT!');
msg+="●使用 targetElement.appendData(' to NUDT!'):\n"+nodeStatus(targetElement);
//使用 insertData( )方法在文本节点内容某个位置添加字符串
targetElement.insertData(0,'YSQ ');
msg+="●使用 targetElement.insertData(0,'YSQ '):\n"+nodeStatus(targetElement);
//使用 deleteData( )方法在文本节点内容中指定位置删除指定长度的字符串
targetElement.deleteData(0,4);
msg+="●使用 targetElement.deleteData(0,4):\n"+nodeStatus(targetElement);
//使用 replaceData( )方法用指定字符串替换文本节点内容指定位置和长度的字符串
targetElement.replaceData(0,7,'JHX Let');
msg+="●使用 targetElement.replaceData(0,7,'JHX Let'):\n"+nodeStatus(targetElement);
//使用 splitText( )方法将文本节点内容从指定位置分为两部分，左边为原始节点，右边赋值给新节点
var tempNode=targetElement.splitText(8);
msg+="●使用 targetElement.splitText(8):\n";
msg+="原始文本节点: \n"+nodeStatus(targetElement);
msg+="新的文本节点: \n"+nodeStatus(tempNode);
//使用 substringData( )方法返回文本节点内容中指定位置和长度的字符串
msg+="●使用 targetElement.substringData(0,3):\n"+targetElement.substringData(0,3);
//输出文本节点属性变化过程
alert(msg);
//-->
</script>
</body>
</html>
```

程序运行结果如图 5.19 所示。

图 5.19 DOM 中关于文本节点操作的方法实例

> 注意：浏览器对这些方法的支持不太规则，在完整支持 DOM Level 1 规范的浏览器中能正常工作。

5.6 对象的事件处理程序

事件由浏览器动作如浏览器载入文档或用户动作诸如敲击键盘、滚动鼠标等触发，而事件处理程序则说明一个对象如何响应事件。在早期支持 JavaScript 脚本的浏览器中，事件处理程序是作为 HTML 标记的附加属性加以定义的，其形式如下：

```
<input type="button" name="MyButton" value="Test Event" onclick="MyEvent()">
```

下面的例子演示鼠标单击时事件的触发过程：

```
//源程序 5.17
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript">
←
//MyButtonA 的 onclick 事件触发函数
function MyEventA()
{
```

```
    alert("MyButtonA is clicked!");
}
//MyButtonB 的 onclick 事件触发函数
function MyEventB()
{
    alert("MyButtonB is clicked!");
}
//-->
</script>
</head>
<body>
<form name=”MyForm” method="post" action="OtherPage.asp">
    <input type="button" name="MyButtonA" value="Test Event A" onclick="MyEventA()">
    <input type="button" name="MyButtonB" value="Test Event B" onclick="MyEventB()">
</form>
</body>
</html>
```

结果显示，鼠标单击“MyButtonA”按钮，弹出警告框“MyButtonA is clicked!”；单击“MyButtonB”按钮，弹出警告框“MyButtonB is clicked!”，可以看出这个方法模拟了事件所表示的动作。

在 DOM 中，可以通过调用对象事件处理程序的方式来触发这个事件：

```
document.MyForm.MyButtonA.onclick()
```

同时，可以将事件处理程序作为对象的属性来触发事件，例如：

```
//源程序 5.18
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript">
<!--
//将 MyEvent( )事件处理程序赋予 MyText 的 onfocus 事件
function TestEvent()
{
    document.MyForm.MyText.onfocus=MyEvent();
}
//onfocus 事件触发结果
function MyEvent()
{
    alert("MyEvent is Trigger!");
}
//-->
</script>
</head>
<body>
<form name="MyForm">
    <input type="text" name="MyText">
    <input type="button" name="MyButton" value="Test Button" onclick="TestEvent()">
</form>
</body>
</html>
```

上面代码的核心语句 document.MyForm.MyText.onfocus=MyEvent()表示将自定义的函数 MyEvent( )赋给对象的事件处理程序 onfocus( )，即给 onfocus( )赋予了节点属性的特征。

注意：由于对事件处理程序属性做的任何变化都会随着文档的重载而消失，因此建议将事件处理程序作为脚本的一部分，像方法一样调用。

5.7 浏览器兼容性策略

由于各大浏览器厂商对 DOM 和 CSS 标准支持的程度不同，导致同样的 HTML 页面在不同浏览器环境中解析不到同样的视觉界面，同时出现了有很多的 CSS 解析 bug，如 IE 盒模型 bug、IE 浮动 3px bug 等，这就是浏览器兼容性问题。

在浏览器版本不断更替的发展过程中，主要浏览器厂商都发现他们的浏览器实现的 CSS 特性与随后发布的 DOM 和 CSS 标准有所不同。为了消除浏览器之争，打破私有代码代码的不兼容性，让 Web 标准体系里的代码在所有的浏览器上都能得到正常解析，实现 Web 应用程序的跨平台性，主要浏览器厂商如 Microsoft、Mozilla 等共同商定由 Web 应用程序开发者在 HTML 页面的<head>元素中自由选择是否添加<!DOCTYPE>标记，以确定文档是遵循旧的 quirks 方式还是标准兼容模型。

可以使用如下的方法在<head>元素中包含<!DOCTYPE >标记来实现浏览器的兼容性：

```
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0 Frameset//EN"
"http://www.w3.org/TR/REC-html140/frameset.dtd">
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html140/loose.dtd">
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD XHTML 1.0//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR//xhtml1/DTD/xhtml1-frameset.dtd">
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR//xhtml1/DTD/xhtml1-transitional.dtd">
<!DOCTYPE HTML PUBLIC    "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

由于 IE5.5 以下版本的浏览器忽略了标准兼容模型，他们不适用<!DOCTYPE>标记而用旧的 quirks 方式。在目前主流浏览器中，所有基于 Mozilla、WinIE6 和 MacIE5 及以上版本的浏览器中都实现了浏览器兼容性策略，从而使用<!DOCTYPE>标记更有可能通过不同浏览器实现相同的视觉界面。

注意：关于盒模型 bug 和移动 3px bug 原理及解决方法请参考请见 W3C 中国网站：http://www.w3cn.org/

5.8 本章小结

本章介绍了 DOM（文档对象模型），它是 JavaScript 脚本与 HTML 文档、CSS 样式表之间联系的纽带。支持 DOM 的浏览器在载入 HTML 文档时按照 DOM 规范将文档节点化形成节点树，JavaScript 通过 DOM 提供的诸如 getElementById( )、removeAttribute( )等方法，可对节点树中的任何已节点化的元素进行访问和修改属性等操作，并通过 createTextNode( )、

appendChild( )等方法迅速生成新文本节点并进行相关操作，甚至动态生成指定的 HTML 文档。

各大浏览器厂商不同程度支持 DOM 规范的推广，但都没有得到很好的支持。Web 应用程序开发者在开发普适当前主流浏览器的 Web 应用程序的时候，就必须充分了解浏览器对 DOM 的支持情况，并编写组织很有条理的 HTML 文档，因为在组织很差的 HTML 文档中执行操作 DOM 的脚本代码，会出现“不可预测的结果”（W3C）。在主流浏览器全面支持 DOM 规范之前，可使用 DOM 中比较基础但得到支持的 DOM Level 1 规范。下面几章我们将集中精力讨论 JavaScript 脚本中数据类型的对象，如 String、Array、Math 等。

第 6 章 String、Math、Array 等数据对象

JavaScript 脚本提供丰富的内置对象，包括同基本数据类型相关的对象（如 String、Boolean、 Number）、允许创建用户自定义和组合类型的对象（如 Object、Array）和其他能简化 JavaScript 操作的对象（如 Math、Date、RegExp、Function）。其中 RegExp 对象将在“正则表达式”章节进行详细的叙述，本章从实际应用出发，详细讨论其余的 JavaScript 脚本内置对象。

6.1 String 对象

String 对象是和原始字符串数据类型相对应的 JavaScript 脚本内置对象，属于 JavaScript 核心对象之一，主要提供诸多方法实现字符串检查、抽取子串、字符串连接、字符串分割等字符串相关操作。

语法如下：

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

该方法使用关键字 new 返回一个使用可选参数“string”字符串初始化的 String 对象的实例 MyString，用于后续的字符串操作。

6.1.1 如何使用 String 对象方法操作字符串

使用 String 对象的方法来操作目标对象并不操作对象本身，而只是返回包含操作结果的字符串。例如要设置改变某个字符串的值，必须要定义该字符串等于将对象实施某种操作的结果。考察如下将字符串转换为大写的程序代码：

```
//源程序 6.1
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function StringTest()
{
  var MyString=new String("Welcome to JavaScript world!");
  var msg="原始字符串 MyString:\n"
  msg+="MyString="+MyString+"\n\n";
  MyString.toUpperCase();
  msg+="运行语句:MyString.toUpperCase():\n";
  msg+="MyString="+MyString+"\n\n";
  MyString=MyString.toUpperCase();
  msg+="运行语句:MyString=MyString.toUpperCase():\n";
  msg+="MyString="+MyString+"\n\n";
```

```
    alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
    <input type=button value=调试对象按钮  onclick="StringTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.1 所示。

图 6.1 如何使用 String 对象的方法实例

调用 String 对象的方法语句 MyString.toUpperCase()运行后，并没有改变字符串 MyString 的内容，要使用 String 对象的 toUpperCase( )方法改变字符串 MyString 的内容，必须将使用 toUpperCase( )方法操作字符串的结果返回给原字符串：

```
MyString=MyString.toUpperCase();
```

通过以上语句操作字符串后，字符串的内容才真正被改变。String 对象的其他方法也具有此种特性。

> 注意：String 对象的 toLowerCase( )方法与 toUpperCase( )方法的语法相同、作用类似，不同点在于前者将目标串中所有字符转换为小写状态并返回结果给新的字符串。在表单数据验证时，如果文本域不考虑大小写，可先将其全部字符转换为小写（当然也可大写）状态再进行相关验证操作。

6.1.2 获取目标字符串长度

字符串的长度 length 作为 String 对象的唯一属性，且为只读属性，它返回目标字符串（包含字符串里面的空格）所包含的字符数。改写源程序 6.1 的 StringTest( )函数：

```
function StringTest()
{
    var MyString=new String("Welcome to JavaScript world!");
    var strLength=MyString.length;
    var msg="获取目标字符串的长度:\n\n"
    msg+="访问方法: var strLength=MyString.length\n\n";
    msg+="原始字符串  内容  :"+MyString+"\n";
```

```
  msg+="原始字符串 长度 :"+strLength+"\n\n";
  MyString="This is the New string!";
  strLength=MyString.length;
  msg+="改变内容的字符串 内容 :"+MyString+"\n";
  msg+="改变内容的字符串 长度 :"+strLength+"\n";
  alert(msg);
}
```

程序运行结果如图 6.2 所示。

图 6.2 获取目标字符串的长度

其中脚本语句:

```
strLength=MyString.length;
```

将 MyString 的 length 属性保存在变量 strLength 中，并且其值随着字符串内容的变化自动更新。

6.1.3 连接两个字符串

String 对象的 concat( )方法能将作为参数传入的字符串加入到调用该方法的字符串的末尾并将结果返回给新的字符串，语法如下:

newString=targetString.concat(anotherString);

改写源程序 6.1 的 StringTest( )函数如下:

```
function StringTest()
{
  //定义两个 String 对象的实例
  var targetString=new String("Welcome to ");
  var strToBeAdded=new String("the world!");
  //调用 String 对象的方法连接字符串
  var finalString=targetString.concat(strToBeAdded);
  //输出返回的新字符串内容
  var msg="\n 连接字符串实例:\n\n";
  msg+="当前目标字符串 :  "+targetString+"\n";
  msg+="被连接的字符串 :  "+strToBeAdded+"\n";
  msg+="连接后的字符串 :  "+finalString+"\n";
  alert(msg);
}
```

程序运行结果如图 6.3 所示。

图 6.3 字符串连接实例

连接字符串的核心语句：

```
var finalString=targetString.concat(strToBeAdded);
```

该代码运行后，将字符串 strToBeAdded 添加到字符串 targetString 的后面，并将生成的新字符串赋值给 finalString，连接过程并不改变字符串 strToBeAdded 和 targetString 的值。

JavaScript 脚本中，也可通过如下的方法实现同样的功能：

```
var finalString="Welcome to "+"the world!";
var finalString="Welcome to ".concat("the world!");
var finalString="Welcome to ".concat("the ","world!");
```

String 对象的 concat( )方法可接受任意数目的参数字符串，并按顺序将它们连接起来添加到调用该方法的字符串后面，并将结果返回给新字符串。

6.1.4 验证邮箱地址合法性

在 Web 应用程序中，经常通过邮箱来进行网站与用户之间的信息交互，如网站通过注册用户的邮箱地址给该用户传递最新资讯。在注册该网站通行证的时候，一般都需提交用户的邮箱信息，此时，必须验证邮箱地址的有效性来保证信息交互的有效进行。

String 对象的 indexOf( )方法返回通过参数传入的字符串出现的开始位置，而邮箱地址必为类似于 username@website.com 的结构，在用户提交的标记为邮箱地址的字符串中，通过 indexOf("@")和 indexOf(".")方法返回值可以判断邮箱地址的有效性。考察如下代码：

```
//源程序 6.2
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function EmailAddressTest()
{
  //获取用户输入的邮箱地址相关信息
  var EmailString=document.MyForm.MyEmail.value;
  var strLength=EmailString.length;
  var index1=EmailString.indexOf("@");
  var index2=EmailString.indexOf(".",index1);
  var msg="验证邮箱地址实例:\n\n";
```

```
  msg+="      邮箱地址 ："+EmailString+"\n";
  msg+="      验证信息 ：";
  //返回相关验证信息
  if(index1==-1||index2==-1||index2<=index1+1||index1==0||index2==strLength-1)
  {
    msg+="邮箱地址不合法!\n\n"
    msg+="不能同时满足如下条件:\n";
    msg+="      1、邮件地址中同时含有'@'和'.'字符；\n";
    msg+="      2、'@'后必须有'.'，且中间至少间隔一个字符；\n"
    msg+="      3、'@'不为第一个字符，'.'不为最后一个字符。\n"
  }
  else
  {
    msg+="邮箱地址合法!\n\n";
    msg+="能同时满足如下条件:\n";
    msg+="      1、邮件地址中同时含有'@'和'.'字符；\n";
    msg+="      2、'@'后必须有'.'，且中间至少间隔一个字符；\n"
    msg+="      3、'@'不为第一个字符，'.'不为最后一个字符。\n"
  }
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  邮箱地址:<input type=text name=MyEmail id=MyEmail value="@">
  <input type=button value=验证邮箱地址  onclick="EmailAddressTest()">
</form>
</center>
</body>
</html>
```

运行上述代码，当文本框中输入 zangpu@gmail.com 等格式合法的邮箱地址时，弹出对话框提示输入的邮箱地址合法，如图 6.4 所示。

图 6.4 验证邮箱地址：输入 zangpu@gmail.com

当文本框中输入“zangpu@gmail”、“gmail.com”、“gmail.com@ zangpu”、“@gmail.com”、“zangpu@gmail.”、“zangpu”、“zangpu@.com”等格式不合法的邮箱地址时，弹出对话框提示示输入的邮箱地址不合法，如图 6.5 所示。

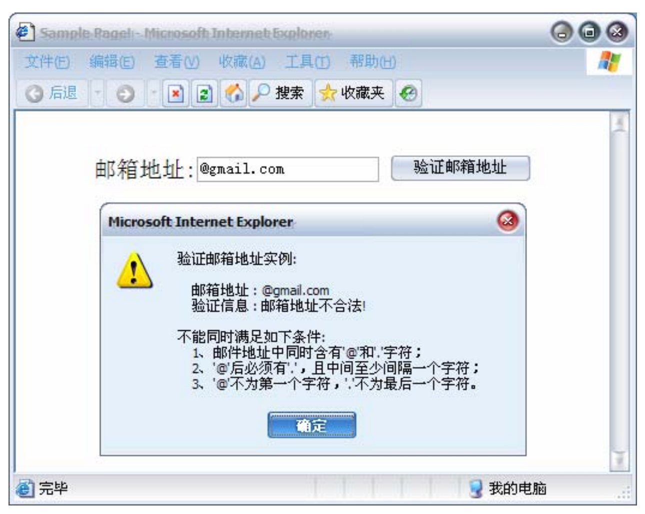

图 6.5 验证邮箱地址：输入@gmail.com

脚本代码中核心的语句：

```
var index1=EmailString.indexOf("@");
var index2=EmailString.indexOf(".",index1);
```

第一句获取目标字符串中'@'字符（也可为字符串）最先出现的位置并将结果返回 index1 变量，返回-1 表示未搜索到该字符；第二句从 index1 变量（即'@'字符后面开始）指定的位置开始搜索'.'字符最先出现的位置，并将结果返回 index2 变量，返回-1 表示未搜索到该字符，以确保在'@'字符后面存在'.'字符。判断语句：

```
if(index1==-1||index2==-1||index2<=index1+1||index1==0||index2==strLength-1)
```

该句是根据邮箱地址规范设定的条件，如果其中一项不满足，则邮箱地址不合法，反之则合法。

String 对象的 indexOf( )方法有个类似的方法，即 lastIndexOf( )方法，该方法与 indexOf( )方法不同点在于其搜索的顺序是由右向左（由后至前），与 indexOf( )方法正好相反。

注意：一般而言邮件地址的格式如下：somebody@domain_name＋后缀 ，domain_name 为域名标识符，即邮件必须要交付到的邮件目的地的域名；somebody 为该域名对应的服务器上存在的邮箱用户的 id；后缀一般则代表了该域名的性质或地区的代码。例如：.com、.edu.cn、.gov、.org、.tw 等。

6.1.5 返回指定位置的字符串

String 对象提供几种方法用于获取指定位置的字符串，且均在 JavaScript 1.0+、JScript 1.0+中获得支持。如表 6.1 所示：

表 6.1 获取指定位置的字符串

方法	语法	说明
slice( )	slice(num1,num2); slice(num)	以参数num1和num2作为开始和结束索引位置，返回目标字符串中num1和num2之间的子串。当num2为负时，从字符串结束位置向前num2个字符即为结束索引位置；当参数num2大于字符串的长度时，字符串结束索引位置为字符串末尾。若只有参数num，返回从num

		索引位置至字符串结束位置的子串。
substr( )	substr(num1,num2); substr(num)	返回字符串在指定初始位置num1、长度为num2个字符的子串。参数num1为负时，返回字符串起始位置开始、长度为num2个字符的子串；当参数num2大于字符串的长度时，字符串结束位置为字符串的末尾。使用单一参数num时，返回从该参数指定的位置到字符串结尾的字符串。
substring( )	substring(num1,num2); substring(num);	返回字符串在指定的索引位置num1和num2之间的字符。如果num2为负，返回从字符串起始位置开始的num1个字符；如果参数num1为负，将被视为0；如果参数num2大于字符串长度，将被视为string.length。使用单一参数num时返回从该参数指定的位置到字符串结尾的子串。

利用 String 对象的这三个方法，可方便地生成指定的子串，考虑下面的代码：

```
//源程序 6.3
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function StringTest()
{
  var MyString=new String("Congratulations!");
  var msg="返回指定位置字符串实例：                \n\n"
  msg+="原始字符串信息：\n";
  msg+="      内容："+MyString+"\n      长度："+MyString.length+"\n\n";
  msg+="slice()方法:\n";
  msg+="      MyString.slice(2,9) : "+MyString.slice(2,9)+"\n";
  msg+="      MyString.slice(2,-2) : "+MyString.slice(2,-2)+"\n";
  msg+="      MyString.slice(2,19) : "+MyString.slice(2,19)+"\n";
  msg+="      MyString.slice(2) : "+MyString.slice(2)+"\n\n";
  msg+="substr()方法:\n";
  msg+="      MyString.substr(2,9) : "+MyString.substr(2,9)+"\n";
  msg+="      MyString.substr(-2,9) : "+MyString.substr(-2,9)+"\n";
  msg+="      MyString.substr(2,19) : "+MyString.substr(2,19)+"\n";
  msg+="      MyString.substr(2) : "+MyString.substr(2)+"\n\n";
  msg+="substring()方法:\n";
  msg+="      MyString.substring(2,9) : "+MyString.substring(2,9)+"\n";
  msg+="      MyString.substring(2,-2) : "+MyString.substring(2,-3)+"\n";
  msg+="      MyString.substring(-2,9) : "+MyString.substring(-2,9)+"\n";
  msg+="      MyString.substring(2,19) : "+MyString.substring(2,19)+"\n";
  msg+="      MyString.substring(2) : "+MyString.substring(2)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form>
  <input type=button value=调试对象按钮  onclick="StringTest()">
</form>
</center>
```

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

程序运行结果如图 6.6 所示。

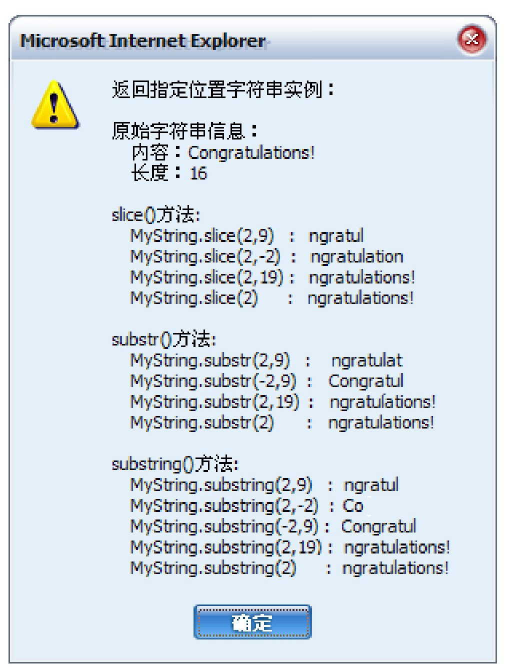

图 6.6 返回指定位置的字符串实例

在 Web 应用程序开发过程中，可以充分结合这三种方法的优缺点，实现更为复杂的字符串选取功能。

String 对象还提供 charAt(num)方法返回字符串中由参数 num 指定位置处的字符，如果 num 不是字符串中的有效索引位置则返回-1；提供 charCodeAt(num)方法返回字符串中由 num 指定位置处字符的 ISO_Latin_1 值，如果 num 不是字符串中的有效索引位置则返回-1。

6.1.6 在 URL 中定位字符串

在 HTML 页面引用、跳转中，经常需要在提交的 URL 中提取感兴趣的内容，如提交表单的用户名等。综合运用以上几个 String 对象关于字符串选取的方法，可在目标字符串中定位到指定的字符串，并能实现指定的字符串多次出现情况下的有效定位。

考察如下代码：

```
//源程序 6.4
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function URLDetect()
{
  //获取文本框内容
```

```
  var MyURL=document.MyForm.MyURL.value;
  var MyStr=document.MyForm.MyStr.value;
  //获取字符串长度
  var URLlength=MyURL.length;
  var Strlength=MyStr.length;
  var msg="";
  //判断文本框是否为空,若空,返回错误信息
  if(URLlength==0||Strlength==0)
  {
    msg+="对不起,你的输入有误，文本框不能为空，但可以为空格!";
    alert(msg);
    return;
  }
  //若不空,执行定位操作
  else
  {
    msg+="在 URL 中定位目标字符串实例：\n\n";
    msg+="原始地址 ：" + MyURL + "\n";
    msg+="目标子串 ：" + MyStr + "\n\n";
    msg+="定位结果 ：\n";
    var index=MyURL.indexOf(MyStr);
    var i=0;
    if(index==-1)
    {
      msg+="目标字符串中没有找到指定的字符串!";
    }
    else
    {
      //搜索到一个位置后,设定标记,然后继续搜索
      while(index!=-1)
      {
        i+=1;
        msg+="位置" +i+ " : " +index+ "\n";
        index=MyURL.indexOf(MyStr, index+1);
      }
    }
  }
  //输出定位信息
  alert(msg);
  return;
}
-->
</script>
</head>
<body>
<form name=MyForm>
  原始地址:
  <input type=text name=MyURL size=60><br>
  搜索子串:
  <input type=text name=MyStr size=60><br><br>
  <center>
    <input type=button value=定位指定字符串  onclick="URLDetect()">
  </center>
```

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

运行上述代码，若目标字符串中存在指定的字符串，返回如图 6.7 所示警告框。

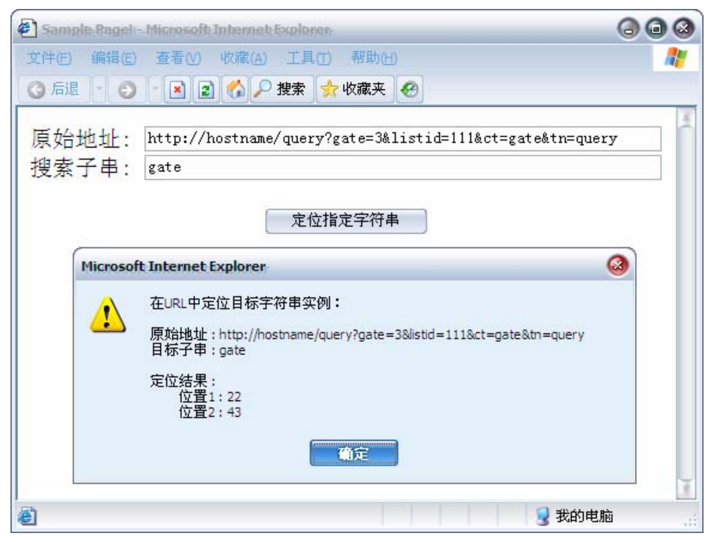

图 6.7 在 URL 中定位字符串：存在指定字符串

若目标字符串中不存在指定的字符串，则返回如图 6.8 所示警告框。

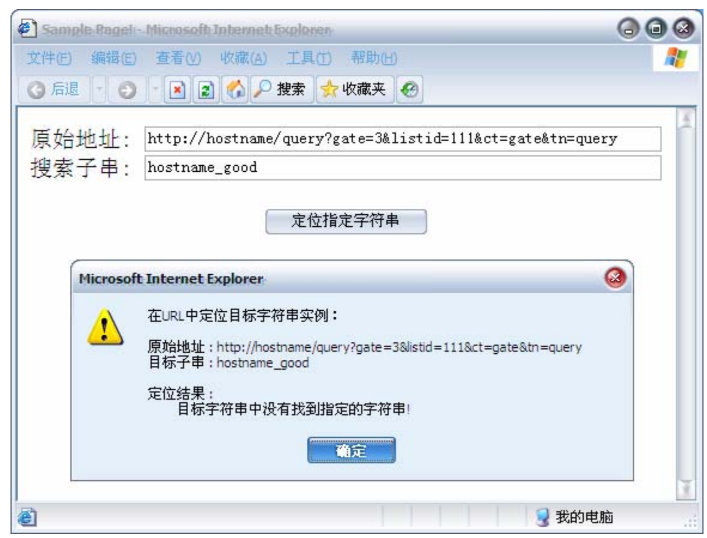

图 6.8 在 URL 中定位字符串：不存在指定字符串

本实例扩展了 String 对象的 indexOf( )方法，将其应用于指定字符串在目标 URL 字符串中存在若干次的情况，同时综合使用了 String 对象的 length 属性。

6.1.7 分隔字符串

String 对象提供 split( )方法来进行字符串的分割操作，split( )方法根据通过参数传入的规则表达式或分隔符来分隔调用此方法的字符串。split( )方法的语法如下：

```
String.split(separator,num);
```

```
String.split(separator);
String.split(regexpression,num);
```

如果传入的是一个规则表达式 regexpression，则该表达式由定义如何匹配的 pattern 和 flags 组成；如果传入的是分隔符 separator，则分隔符是一个字符串或字符，使用它将调用此方法的字符串分隔开，num 表示返回的子串数目，无此参数则默认为返回所有子串。考察如下的代码：

```
//源程序 6.5
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//显示分隔之后形成的字符串数组信息
function getMsg(arrayName,MyStr)
{
  var tempMsg="分隔模式 ："+MyStr+"            \n";
  if(arrayName==null)
  {
    tempMsg+="没有搜索到匹配模式!";
  }
  else
  {
    for(var i=0;i<arrayName.length;i++)
    {
      tempMsg+='['+i+'] : '+arrayName[i]+"\n";
    }
  }
  tempMsg+="\n";
  return tempMsg;
}
//实施分隔操作
function MySplit()
{
  var MyString=new String("Mr. R. Allen Wyke");
  var MyRegExp=/\s/g;
  var MySeparator=" ";
  var msg="分隔字符串实例:\n\n";
  msg+="原始字符串 ："+MyString+"\n";
  msg+="分隔符 ："+MySeparator+"(空格)\n";
  msg+="正则表达式 ："+MyRegExp+"\n\n";
  msg+=getMsg(MyString.split(MySeparator),"MyString.split(MySeparator)");
  msg+=getMsg(MyString.split(MySeparator,2),"MyString.split(MySeparator,2)");
  msg+=getMsg(MyString.split(MyRegExp),"MyString.split(MyRegExp)");
  msg+=getMsg(MyString.split(MyRegExp,3),"MyString.split(MyRegExp,3)");
  alert(msg);
}
-->
</script>
</head>
```

```
<body>
<center>
<form name=MyForm>
  <input type=button value=分隔指定字符串 onclick="MySplit()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.9 所示。

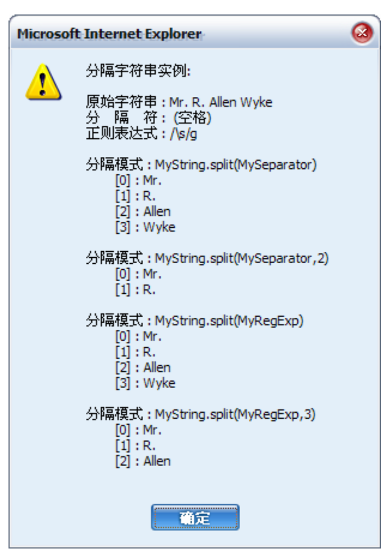

图 6.9 分隔字符串实例

核心语句：

```
MyString.split(MySeparator);
MyString.split(MySeparator,2);
MyString.split(MyRegExp);
MyString.split(MyRegExp,3);
```

代码运行后，按照 String 对象的 split( )方法，将分隔而成的子串形成的字符串数组的相关信息通过函数 getMsg(arrayName,MyStr)返回 msg 变量输出。

正则表达式 RegExp 对象的相关知识在“正则表达式”章节将详细讲述。

6.1.8 将字符串标记为 HTML 语句

客户端 JavaScript 脚本主要用于处理 HTML 文档中各元素对象，同时提供大量的方法将字符串转化为 HTML 语句，这些方法返回使用了 HTML 标记对的字符串。如下面的代码：

```
var MyString="Welcome to JavaScript world!".big();
document.write(MyString);
```

脚本运行后，相当于下面的 HTML 语句：

```
<big> Welcome to JavaScript world!</big>
```

在 HTML 文档中，标记之间可以相互嵌套，JavaScript 脚本也可通过链式方法调用实现该效果，如 HTML 语句：

```
<a url="parent.html"><big> <strike>Welcome to JavaScript world! </strike></big></a>
```

使用 JavaScript 脚本实现的代码如下：

```
var MyStr="Welcome to JavaScript world!".strike().big().link('parent.html');
```

其中调用的先后顺序对应于 HTML 语句由内向外的顺序。考察如下的代码：

```
//源程序 6.6
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
<!--
var MyString=new String("How Are You?");
document.write("                原始字符串："+MyString+"<br><hr>");
document.write("              anchor()方法："+MyString.anchor("New")+"<br>");
document.write("                 big()方法："+MyString.big()+"<br>");
document.write("               small()方法："+MyString.small()+"<br>");
document.write("                bold()方法："+MyString.bold()+"<br>");
document.write("       fontcolor('blue')方法："+MyString.fontcolor('blue')+"<br>");
document.write("     fontcolor('ff0000')方法："+MyString.fontcolor('ff0000')+"<br>");
document.write("            fontsize(5)方法："+MyString.fontsize(5)+"<br>");
document.write("          fontsize('-2')方法："+MyString.fontsize('-2')+"<br>");
document.write("              italics()方法："+MyString.italics()+"<br>");
document.write("      link('parent.html')方法："+MyString.link('parent.html')+"<br>");
document.write("               strike()方法："+MyString.strike()+"<br>");
document.write("                 sub()方法："+MyString.sub()+"<br>");
document.write("                 sup()方法："+MyString.sup()+"<br>");
-->
</script>
</body>
</html>
```

程序运行的结果如图 6.10 所示。

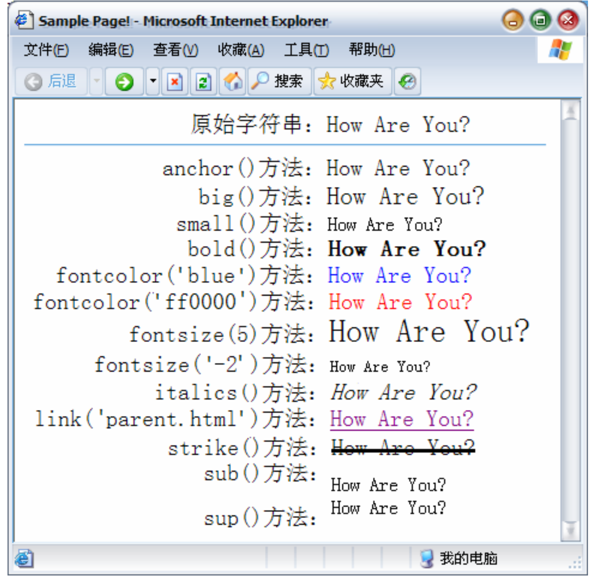

图 6.10 将字符串标记为 HTML 语句实例

在 JavaScript 脚本生成 HTML 语句诸多方法中，有些方法如 fontcolor()方法等会生成包含有 HTML 4 标准不赞成使用或已被淘汰标记如<blink>等的字符串，在 XHTML 中样式表（CSS）将逐步取代用 JavaScript 脚本生成 HTML 语句的方法。

6.1.9 常见属性和方法汇总

JavaScript 脚本的核心对象 String 提供大量的属性和方法来操作字符串。表 6.2 列出了其常用的属性、方法以及脚本版本支持情况。

表 6.2 String对象常用的属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	length	返回目标字符串的长度。详情请见6.1.2应用实例	①②
	prototype	用于给String对象增加属性和方法。	③⑥
方法	anchor(name)	创建<a>标签，并用参数nane设置其NAME属性	①②
	big( ) 、blink( )、 bold( )、fixed( )、italics()、small()、strike()、sub()、sup()	对应于方法，分别创建HTML语句中<big>、<blink>、<bold>、<tt>、<i>、<small>、<strike>、<sub>、<sup>等标签。详情请见6.1.8应用实例	①②
	link(URL)	创建<a>标签，并用参数URL指定其HREF属性。	①②
	charAt(num)	用于返回参数num指定索引位置的字符。如果参数num不是字符串中的有效索引位置则返回-1。	①②
	charCodeAt(num)	与charAt( )方法相同，但其返回ISO_Latin_1值。	①②
	concat(string2)	把参数string2传入的字符串连接到当前字符串的末尾并返回新的字符串。详见6.1.3应用实例	④⑥
	fontcolor(hexnum) fontcolor(color)	创建<font>标签并设置其color属性。详情请见6.1.8应用实例	①②
	fontsize(num) fontsize(string2)	创建<font>标签并设置size属性为数字num或由字符串string2表示的相对于<basefont>标签的增减值i。详情请见6.1.8应用实例	①②
	fromCharCode(num1,…numN) fromCharCode(keyevent.which)	返回对应于通过参数num1至numN传入的ISO_Latin_1值位置处的字符，或者传入一个键盘事件并由其which属性指定哪个键被按下。	④⑥
	indexOf(string,num) indexOf(string)	返回通过字符串传入的字符串传入的字符串string出现的位置，详情请见6.1.6应用实例。	①②
	LastIndexOf()	参数与indexOf相同，功能相似，索引方向相反。	①②
	match(regexpression)	查找目标字符串中通过参数传入的规则表达式regexpression所指定的字符串。	④⑥
	replace(regExpression,strin2)	查找目标字符串中通过参数传入的规则表达式指定的字符串，若找到匹配字符串，返回由参数字符串string2替换匹配字符串后的新字符串。	④⑥
	search(regexpression)	查找目标字符串中通过参数传入的规则表达式指定的字符串，找到配对时返回字符串的索引位置否则返回-1。	④⑥
	slice(num1,num2) slice(num)	返回目标字符串指定位置的字符串。详情请见6.1.5应用实例	①②
	split(separetor,um) split(separetor) split(regexpression,num)	根据参数传入的规则表达式regexpression或分隔符separetor来分隔目标字符串，并返回字符串数组。详情请见6.1.7应用实例	③②
	substr(num1,num2) substr(num)	返回目标字符串中指定位置的字符串，详情请见6.1.5应用实例	①②
	substring(num1,num2) substring(num)	返回目标字符串中指定位置的字符串，详情请见6.1.5应用实例	①②
	toLowerCases()	将字符串的全部字符转化为小写。	①②
	toUpperCase()	将字符串的全部字符转化为大写。	①②
	valueOf()	返回String对象的原始值。	④⑥

注意：在以上版本支持情况中，①指 JavaScript 1.0+； ②指 JScript 1.0+； ③指 JavaScript 1.1+； ④指 JavaScript 1.2+； ⑤指 JavaScript 1.3 +；⑥指 JScript 3.0+。此项设定下同。

在 JavaScript 脚本程序编写过程中，String 对象是最为常见的处理目标，用于存储较短的数据。JavaScript 语言提供了丰富的属性和方法支持，方便 Web 应用程序开发者灵活地操纵 String 对象的实例。

6.2 Math 对象

Math 对象是 JavaScript 核心对象之一，拥有一系列的属性和方法，能够进行比基本算术运算更为复杂的运算。但 Math 对象所有的属性和方法都是静态的，并不能生成对象的实例，但能直接访问它的属性和方法。例如可直接访问 Math 对象的 PI 属性和 abs(num)方法：

```
var MyPI=Math.PI;
var MyAbs=Math.abs(-5);
```

需要注意的是，JavaScript 脚本中浮点运算精确度不高，常导致计算结果产生微小误差从而导致最终结果的致命错误。例如：

```
alert(Math.sin(Math.PI));
```

代码运行结果如图 6.11 所示。

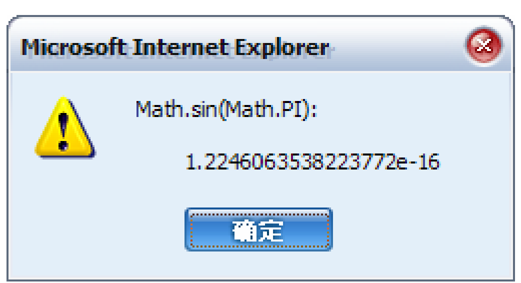

图 6.11 调用 Math.sin(Math.PI)方法的返回结果

可见，JavaScript 脚本中 Math.sin(Math.PI)返回的结果与理论上的 0 非常接近，但微小的误差足以导致精确计算的失败。

6.2.1 基本数学运算

Math 对象提供丰富的方法用于数学运算，特别是三角函数方面的方法。由于三角函数的参数使用弧度制，要在参数上乘以 $\pi/180$。考察如下的代码：

```
//源程序 6.7
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
```

```
{
   var msg="Math 对象基本数学运算实例:                              \n\n";
   msg+="绝对值  Math.abs(-1)="+Math.abs(-1)+"\n";
   msg+="平方根  Math.sqrt(9)="+Math.sqrt(9)+"\n";
   msg+="反余弦  Math.acos(-1)="+Math.acos(-1)+"\n";
   msg+="反正弦  Math.asin(-1)="+Math.asin(-1)+"\n";
   msg+="反正切  Math.atan(-1)="+Math.atan(-1)+"\n";
   msg+="反正切(方位角) Math.atan2(1,2)="+Math.atan2(1,2)+"\n";
   msg+="正弦值  Math.sin(45*Math.PI/180)="+Math.sin(45*Math.PI/180)+"\n";
   msg+="余弦值  Math.cos(45*Math.PI/180)="+Math.cos(45*Math.PI/180)+"\n";
   msg+="正切值  Math.tan(45*Math.PI/180)="+Math.tan(45*Math.PI/180)+"\n";
   msg+="大于一个数的最小整数  Math.ceil(1.2)="+Math.ceil(1.2)+"\n";
   msg+="小于一个数的最小整数  Math.floor(1.2)="+Math.floor(1.2)+"\n";
   msg+="欧拉常数的次幂  Math.exp(2)="+Math.exp(2)+"\n";
   msg+="数的自然对数  Math.log(3)="+Math.log(3)+"\n";
   msg+="两数中的最大值  Math.max(1,2)="+Math.max(1,2)+"\n";
   msg+="两数中的最小值  Math.min(1,2)="+Math.min(1,2)+"\n";
   msg+="数的次方  Math.pow(3,4)="+Math.pow(3,4)+"\n";
   msg+="最接近的整数  Math.round(2.1)="+Math.round(2.1)+"\n";
   msg+="最接近的整数  Math.round(2.6)="+Math.round(2.6)+"\n";
   msg+="最接近的整数  Math.round(2.1)="+Math.round(2.1)+"\n";
   msg+="将数值转换为字符串  Math.toString(123456)="+Math.toString(123456)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=数学运算  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.12 所示。

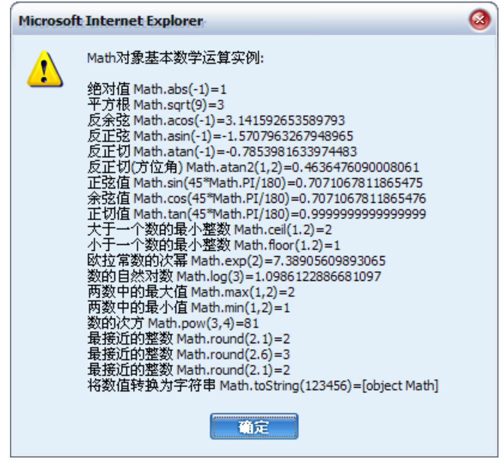

图 6.12 Math 对象的方法中基本数学运算

可见，Math 对象提供很多的数学方法用于基本运算，这些基本能满足 Web 应用程序的要求，但在实际应用中要充分考虑程序的精度要求，并在满足精度的情况下对获得的数据进行适当的截尾。

6.2.2 任意范围随机数发生器

在 JavaScript 脚本中，可使用 Math 对象的 random()方法生成 0 到 1 之间的随机数，考察下面任意范围的随机数发生器代码：

```
//源程序 6.8
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var m=document.MyForm.MyM.value;
  var n=document.MyForm.MyN.value;
  var msg="m 到 n 之间的随机数产生实例:\n\n";
  msg+="随机数范围设定:\n";
  msg+="下限:"+m+"\n";
  msg+="上限:"+n+"\n\n"
  if(m==n)
  {
     msg+="错误提示信息:\n"
     msg+="上限与下限相等,请返回重新输入!";
  }
  else
  {
    msg+="随机数产生结果:\n"
    for(var i=0;i<10;i++)
    {
      //产生 0-1 之间随机数，并通过系数变换到 m-n 之间
      msg+="第  "+(i+1)+"  个: "+(Math.random()*(n-m)+m)+"\n";
    }
  }
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  随机数产生范围下限 : <input type=text name=MyM size=30 value=1><br>
  随机数产生范围上限 : <input type=text name=MyN size=30 value=10><br><br>
  <input type=button value=数学运算  onclick="MyTest()">
```

```
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.13 所示。

图 6.13 任意范围的随机数发生器

程序中关键代码：

```
Math.random()*(n-m)+m;
```

首先产生 0 和 1 之间随机数，然后通过系数变换，将其限定在 m 和 n(n>m)之间的随机数，并可通过更改文本框内容的形式，产生任意范围的随机数。

6.2.3 访问其基本属性

Math 对象拥有很多基本属性，如圆周率 Math.PI、Math.SQRT2、Math.log10E 等，表示数学运算中经常使用的常量。考察下面的代码：

```
//源程序 6.9
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var msg="";
  msg+="Math 对象基本属性: \n\n";
  msg+="欧拉常数 Math.E = "+Math.E+"\n\n";
  msg+="圆周率 Math.PI = "+Math.PI+"\n\n";
  msg+="10 的自然对数 Math.LN10 = "+Math.LN10+"\n\n";
  msg+="2 的自然对数 Math.LN2 = "+Math.LN2+"\n\n";
  msg+="E 为底 10 的对数 Math.LOG10E = "+Math.LOG10E+"\n\n";
  msg+="E 为底 2 的对数 Math.LOG2E = "+Math.LOG2E+"\n\n";
  msg+="0.5 的平方根 Math.SQRT1_2 = "+Math.SQRT1_2+"\n\n";
  msg+="2 的平方根 Math.SQRT2 = "+Math.SQRT2+"\n";
```

```
    alert(msg);
}
-->
</script>
</head>
<body>
<center>
<form name=MyForm>
    <input type=button value=数学运算  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.14 所示。

图 6.14 Math 对象的基本属性

6.2.4 使用 with 声明简化表达式

在 Math 对象进行数学计算时，常涉及到它的很多种属性或方法，如果每次都使用 Math.abs()、Math.PI 方式调用的话，代码复杂度高。可以使用 with 申明来简化调用的代码。考察如下 JavaScript 脚本：

```
//源程序 6.10
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
    var msg="使用 with 声明简化表达式实例:\n\n";
    //使用 with 声明简化其表达式
    with(Math)
    {
        var a=document.MyForm.Mya.value;
        var b=document.MyForm.Myb.value;
```

```
        var AngleC=document.MyForm.MyC.value;
        var AngleA=atan(a/b);
        var AngleB=atan(b/a);
        var powerC=pow(a,2)+pow(b,2)+2*a*b*cos(AngleC*PI/180);
        msg+="运用余弦定理计算:\n";
        msg+="角 A="+AngleA+"\n";
        msg+="角 B="+AngleB+"\n";
        msg+="边 c="+sqrt(powerC)+"\n";
    }
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    边 a<input type=text name=Mya value=3><br>
    边 b<input type=text name=Myb value=4><br>
    角 C<input type=text name=MyC value=90><br>
    <input type=button value=简化数学运算  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.15 所示。

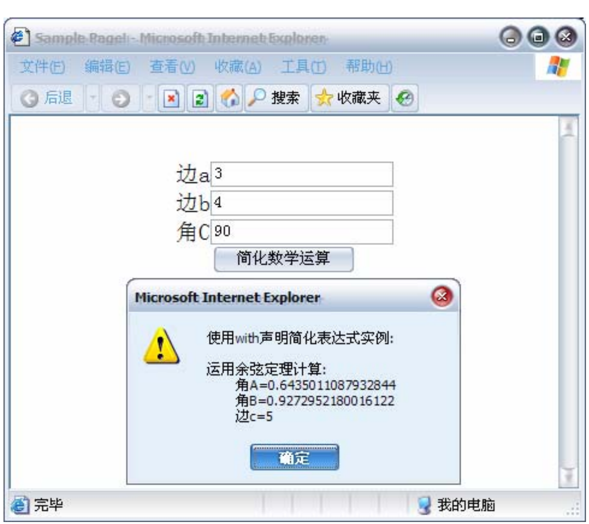

图 6.15 使用 with 声明简化表达式

在 JavaScript 脚本代码中声明 with 后，在 with 作用范围内的 Math 对象的属性和方法都可以直接使用，而不需要使用 Math 对象引用的方式调用。

6.2.5 常见属性汇总

Math 对象的常见属性汇总情况见表 6.3 所示：

表 6.3 Math对象的常见属性列表

属性	说明	脚本版本支持	浏览器版本支持
Math.E	返回欧拉常数e的值	①②	①⑤⑦
Math.LN2	返回2的自然对数	①②	①⑤⑦
Math.LN10	返回10的自然对数	①②	①⑤⑦
Math.LOG2E	返回e为底2的对数	①②	①⑤⑦
Math.LOG10E	返回e为底10的对数	①②	①⑤⑦
Math.PI	返回圆周率PI的值	①②	①⑤⑦
Math.SQRT1_2	返回0.5的平方根	①②	①⑤⑦
Math.SQRT2	返回2的平方根	①②	①⑤⑦

注意：在浏览器版本支持中，①指 Navgator 2+； ②指 Navgator 3+； ③指 Navgator 4+； ④指 Navgator 4.06+； ⑤指 Internet Explorer 3 +；⑥指 Internet Explorer 4 +；⑦指 Opera 3+。此项设定下同。

6.2.6 常见方法汇总

Math 对象的常见方法汇总情况见表 6.4 所示：

表 6.4 Math对象的常见方法列表

方法	说明	脚本版本支持	浏览器支持
Math.abs(num)	返回num的绝对值	①②	①⑤⑦
Math.acos(num)	返回num的反余弦	①②	①⑤⑦
Math.asin(num)	返回num的反正弦	①②	①⑤⑦
Math.atan(num)	返回num的反正切	①②①	①⑤⑦
Math.atan2(num1, num2)	返回一个除法表示式的反正切值	①⑥	①⑥
Math.ceil(num)	返回大于等于一个数的最小整数	①②①	①⑤⑦
Math.cos(num)	返回num的余弦值	①②	①⑤⑦
Math.exp(num)	返回底为欧拉常数e的num次方	①②	①⑤⑦
Math.floor(num)	返回小于等于一个数的最大整数	①②	①⑤⑦
Math.log(num)	返回以欧拉常数e为底num的自然对数	①②	①⑤⑦
Math.max(num1, num2)	返回num1和num2中较大的一个数	①②	①⑤⑦
Math.min(num1, num2)	返回num1和num2中较大的一个数	①②	①⑤⑦
Math.pow(num1, num2)	返回num1的num2次方	①②	①⑤⑦
Math.random()	返回0至1间的随机数	③②	①⑤⑦
Math.round(num)	返回最接近num的整数	①②	①⑤⑦
Math.sin(num)	返回num的正弦值	①②	①⑤⑦
Math.sqrt(num)	返回num的平方根	①②	①⑤⑦
Math.tan(num)	返回num的正切值	①②	①⑤⑦
Math.toSource(object)	返回Math对象object的拷贝	⑤	④
Math.toString()	返回表示Math对象的字符串	①⑥	①⑥

Math 对象提供大量的属性和方法实现 JavaScript 脚本中的数学运算，但由于其为静态对象，不能创建对象的实例，更不能动态添加属性和方法，导致其使用范围较窄。下面介绍功能完善，且扩展方便的 Array 对象。

6.3 Array 对象

数组是包含基本和组合数据类型的有序序列，在 JavaScript 脚本语言中实际指 Array 对象。数组可用构造函数 Array( )产生，主要有三种构造方法：

```
Var MyArray=new Array();
var MyArray =new Array(4);
var MyArray =new Array(arg1,arg2,...,argN);
```

第一句声明一个空数组并将其存放在以 MyArray 命名的空间里，可用数组对象的方法动态添加数组元素；第二句声明长度为 4 的空数组，JavaScript 脚本中支持最大的数组长度为 4294967295；第三句声明一个长度为 N 的数组，并用参数 arg1、arg2、...、argN 直接初始化数组元素，该方法在实际应用中最为广泛。

在 JavaScript 脚本版本更新过程中，渐渐支持使用数组字面值来声明数组的方法。与上面构造方法相对应，出现了如下的数组构造形式：

```
var MyArray=[];
var MyArray =[,,,];
var MyArray =[arg1,arg2,...,argN];
```

该构造方法在 JavaScript 1.2+版本中首先获得支持。其中第二种方式中表明数组长度为 4，并且数组元素未被指定，浏览器解释时，把其看成拥有 4 个未指定初始值的元素的数组。将其扩展如下：

```
var MyArray =[234,,24,,56,,,,3,4];
```

该数组构造方式构造一个长度为 10、某些位置指定初始值、其他位置未指定初始值的数组 MyArray，MyArray 又被称为稀疏数组，可通过给指定位置赋值的方法来修改该数组。

Array 对象提供较多的属性和方法来访问、操作目标 Array 对象实例，如增加、修改数组元素等。

6.3.1 创建数组并访问其特定位置元素

JavaScript 脚本中，使用 new 操作符来创建新数组，并可通过数组元素的下标实现对任意元素的访问。考察如下代码：

```
//源程序 6.11
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var MyArray=new Array("TOM","Allen","Lily","Jack");
  var strLength=MyArray.length;
  var msg="创建数组并通过下标访问实例:\n\n";
  msg+="创建目标数组语法:\n"
  msg+="new Array('TOM','Allen','Lily','Jack');\n\n";
  msg+="目标数组信息:\n";
  msg+="MyArray.length = "+ strLength +"\n";
```

```
  for(var i=0;i<strLength;i++)
  {
     msg+="MyArray["+i+"] = "+MyArray[i]+"\n";
  }
  msg+="MyArray[5] = "+MyArray[5]+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.16 所示。

图 6.16 创建数组并通过下标访问其元素实例

数组元素下标从 0 开始顺序递增，可通过数组元素的下标实现对它的访问：

```
var data=MyArray[i];
```

但访问数组中未被定义的元素时将返回未定义的值，如下列代码：

```
var data=MyArray[5];
```

运行后，data 返回 undefined。同样，使用稀疏数组时，访问未被定义的元素也将返回未定义的值 undefined。

6.3.2 数组中元素的顺序问题

Array 对象提供相关相关方法实现数组中元素的顺序操作，如颠倒元素顺序、按 Web 应用程序开发者制定的规则进行排列等，主要有 Array 对象的 reverse()和 sort()方法。

reverse()方法将按照数组的索引号的顺序将数组中元素完全颠倒，语法如下：

```
arrayName.reverse();
```

sort()方法较之 reverse()方法复杂，它基于某种顺序重新排列数组的元素，语法如下：

```
arrayName.sort();
arrayName.sort(function);
```

第一种调用方式不指定排列顺序，JavaScript 脚本将数组元素转化为字符串，然后按照

字母顺序进行排序。

第二种调用方式由参数 function 指定排序算法，该算法需遵循如下的规则：

- 算法必须接受两个可以比较的参数 a 和 b，即 function(a,b)；
- 算法必须返回一个值以表示两个参数之间的关系；
- 若参数 a 在参数 b 之前出现，函数返回小于零的值；
- 若参数 a 在参数 b 之后出现，函数返回大于零的值；
- 若参数 a 等于 b，则返回零。

考察如下的代码：

```
//源程序 6.12
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//输出数组信息
function getMsg(arrayName)
{
  var arrayLength=arrayName.length;
  var tempMsg="";
  for(var i=0;i<arrayLength;i++)
  {
    tempMsg+="            MyArray["+i+"]="+arrayName[i]+"\n";
  }
  return tempMsg;
}
//根据排序算法规则构造排序算法 MyFunction(arg1,arg2)
function MyFunction(arg1,arg2)
{
  var dataReturn;
  if(arg1.length<arg2.length)
    dataReturn=-1;
  else if(arg1.length>arg2.length)
    dataReturn=1;
  else   dataReturn=0;
  return dataReturn;
}
function MyTest()
{
  var MyArray=new Array("First","Second","Third","Forth");
  var msg="数组元素的顺序操作实例:\n\n";
  msg+="原始数组:\n"+getMsg(MyArray)+"\n";
  msg+="前后颠倒:\n"+getMsg(MyArray.reverse())+"\n";
  msg+="字母顺序:\n"+getMsg(MyArray.sort())+"\n";
  msg+="排序算法:\n"+getMsg(MyArray.sort(MyFunction))+"\n";
  alert(msg);
}
-->
</script>
</head>
```

```
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.17 所示。

图 6.17 数组元素排序实例

在上述程序中，构造了排序算法 MyFunction(arg1,arg2)，并根据其返回值决定数组中各元素之间的相对顺序。

6.3.3 模拟堆栈和队列操作的方法

为实现数组的动态操作，从 JavaScript 1.2+和 JScript 5.5+开始，Array 对象提供了诸如 pop()、push()、unshift()、shift()等方法来动态添加和删除数组元素。先来了解两个抽象的数据类型：

- 堆栈（LIFO）：用于以“后进先出”的顺序存储数据的结构。在读取堆栈的时候，最后存入的数据最先被读取出来；
- 队列（FIFO）：用于以“先进先出”的顺序储存数据的结构。在读取队列的时候，最先存入的数据最先被读取出来。

其中 pop()方法模拟堆栈的“压栈”动作，将数组中最后一个元素删除，并将该元素作为操作的结果返回，同时更新数组的 length 属性；push()方法模拟堆栈的“出栈”动作，将以参数传入的元素按参数顺序添加到数组的尾部，并将插入的元素作为操作的结果返回，同时更新数组的 length 属性。

unshift()、shift()方法与 pop()、push()方法相对，都是删除和添加数组元素，仅仅是删除和添加目标的位置不同，前者与后者相反方向，即从数组的第一个元素开始操作，后面的元素将分别向前和向后移动，数组的 length 属性自动更新。

考察如下的代码：

```
//源程序 6.13
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//输出数组信息
function getMsg(arrayName)
{
  var arrayLength=arrayName.length;
  var tempMsg="              长度 ："+arrayLength+"\n";
  for(var i=0;i<arrayLength;i++)
  {
    tempMsg+="              MyArray["+i+"]="+arrayName[i];
  }
  return tempMsg;
}
function MyTest()
{
  var MyArray=new Array("First","Second","Third","Forth");
  var msg="添加和删除数组元素实例:\n\n";
  msg+="原始数组:\n"+getMsg(MyArray)+"\n\n";
  var arrayPop=MyArray.pop();
  msg+="使用 pop()方法:\n"+getMsg(MyArray)+"\n";
  var arrayPush=MyArray.push("Forth");
  msg+="使用 push('Forth')方法:\n"+getMsg(MyArray)+"\n";
  var arrayShift=MyArray.shift();
  msg+="使用 shift()方法:\n"+getMsg(MyArray)+"\n";
  var arrayUnshift=MyArray.unshift("First");
  msg+="使用 unshift('First')方法:\n"+getMsg(MyArray)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.18 所示。

图 6.18 模拟堆栈和队列进行操作的方法实例

Array 对象提供的模拟堆栈和队列进行数组元素添加、删除的方法，使用非常简单。它致命的缺陷就是只能操作数组的头部或末端的数组元素，不能进行任意位置数组元素的添加和删除操作，下面介绍可以在数组中任意位置执行该操作的 splice()方法。

注意：Array 对象的 pop()、push()、unshift()、shift()等添加和删除数组元素的方法，虽模拟了堆栈和队列的基本动作，但并不能因此而将数组看成堆栈或队列。

6.3.4 使用 splice()方法添加和删除数组元素

Array 对象的 splice()方法提供一种在数组任意位置添加、删除数组元素的方法。语法如下：

```
MyArray.splice(start,delete,arg3,…,argN);
```

参数说明如下：

- 当参数 delete 为 0 时，不执行任何删除操作；
- 当参数 delete 非 0 时，在调用此方法的数组中删除下标从 start 到 shart+delete 的数组元素，其后的数组元素的下标均减小 delete；
- 如果在参数 delete 之后还有参数，在执行删除操作之后，这些参数将作为新元素添加到数组中由 start 指定的开始位置，原数组该位置之后的元素往后顺移。

考察如下的代码：

```
//源程序 6.14
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//返回数组信息
function getMsg(arrayName)
{
  var arrayLength=arrayName.length;
  var tempMsg="      长度 ："+arrayLength+"\n";
  for(var i=0;i<arrayLength;i++)
  {
```

```
        tempMsg+="          MyArray["+i+"]="+arrayName[i];
    }
    tempMsg+="               ";
    return tempMsg;
}
//执行相关操作
function MyTest()
{
    var MyArray=new Array("First","Second","Third","Forth");
    var msg="添加和删除数组元素实例:\n\n";
    msg+="原始数组:\n"+getMsg(MyArray)+"\n\n";
    MyArray.splice(1,0);
    msg+="使用 splice(1,0)方法:\n"+getMsg(MyArray)+"\n";
    MyArray.splice(1,1);
    msg+="使用 splice(1,1)方法:\n"+getMsg(MyArray)+"\n";
    MyArray.splice(1,1,"New1","New2");
    msg+="使用 splice(1,1,"New1","New2")方法:\n"+getMsg(MyArray)+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.19 所示。

图 6.19 使用 splice()方法增加和删除数组元素实例

其中核心语句：

```
MyArray.splice(1,0);
MyArray.splice(1,1);
MyArray.splice(1,1,"New1","New2");
```

第一句中参数 delete 为 0，不执行任何操作，MyArray 数组保持不变：

```
MyArray=["First","Second","Third","Forth"];
```

第二句参数 delete 不为 0（=1），执行删除下标为 start（=1）到 start+delete（=2）之间的数组元素，即 MyString[1]=Second，其后的数组元素往前挪动 delete（=1）位，此时 MyArray 数组变为：

```
MyArray=["First","Third","Forth"];
```

第三句在继续执行一次第二句的删除操作（删除 MyString[1]=Third）基础上，将以参数传入的“New1”和“New2”元素作为数组元素插入到 start（=1）指定的位置，原位置上的数组元素顺移，相当于执行两个步骤：

```
MyArray=["First","Forth"];
MyArray=["First","New1","New2","Forth"];
```

> 注意：Array 对象的 splice()方法在 Navigator 4 中存在一个缺陷，当删除数组中指定的元素时，返回的不是删除了指定元素的数组而是该指定的元素，同时，如果数组中没有元素被删除，返回的不是空数组而是 null。

6.3.5 修改 length 属性更改数组

Array 对象的 length 属性保存目标数组的长度：

```
var strLength=MyArray.length;
```

Array 对象的 length 属性检索的是数组末尾的下一个可及（未被填充）的位置的索引值，即使前面有些索引没被使用，length 属性也返回最后一个元素后面第一个可及位置的索引值。考察下面代码：

```
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var MyArray=new Array();
  MyArray[10]="Welcome!";
  var arrayLength=MyArray.length;
  var msg="数组的 length 属性实例:\n\n";
  msg+=" MyArray.length = "+arrayLength +"\n";
  alert(msg);
}
-->
</script>
```

程序运行结果如图 6.20 所示。

图 6.20　Array 对象的 length 属性实例

同时，当脚本动态添加、删除数组元素时，数组的 length 属性会自动更新。在循环访问数组元素的过程中，应十分注意控制循环的变量的变化情况。

length 属性可读可写，在 JavaScript 脚本中可通过修改数组的 length 属性来更改数组的

内容，如通过减小数组的 length 属性，改变数组所含的元素，即凡是下标在新 length-1 后的数组元素将被删除。考察如下代码：

```
//源程序 6.15
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//返回数组信息
function getMsg(arrayName)
{
  var arrayLength=arrayName.length;
  var tempMsg="      长度 ："+arrayLength+"\n";
  for(var i=0;i<arrayLength;i++)
  {
    tempMsg+="        MyArray["+i+"]="+arrayName[i];
  }
  tempMsg+="                    ";
  return tempMsg;
}
//执行相关操作
function MyTest()
{
  var MyArray=new Array("First","Second","Third","Forth");
  var msg="修改 length 属性更改数组:\n\n";
  msg+="原始数组:\n"+getMsg(MyArray)+"\n\n";
  MyArray.length=3;
  msg+="使用 MyArray.length=3 语句 ：\n"+getMsg(MyArray)+"\n";
  MyArray.length=4;
  msg+="使用 MyArray.length=4 语句 ：\n"+getMsg(MyArray)+"\n";
  MyArray[3]="Fifth";
  msg+="使用 MyArray[3]='Fifth'语句直接赋值 ：\n"+getMsg(MyArray)+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.21 所示。

图 6.21 修改 length 属性更改数组实例

在使用 MyArray.length=3 语句后，数组长度变为 3，直接删除数组元素 MyArray[3]；在使用 MyArray.length=4 语句后，数组长度变变为 4，在数组末端添加元素 MyArray[3]，且为未定义类型；在使用 MyArray[3]="Fifth"语句直接给 MyArray[3]赋值“Fifth”后，数组：

MyArray=["First","Second","Third","Fifth"];

MyArray.length=4;

更改 Array 对象的 length 属性后，任何包含数据的索引只要大于 length-1，将立即被设定为未定义类型。

6.3.6 调用 Array 对象的方法生成字符串

在 Web 应用程序开发过程中，常常需要将数组元素按某种形式转化为字符串，如需将存放用户名的数组中各个元素转换为字符串并赋值给各用户等。先考察如下代码：

```
//源程序 6.16
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//返回数组信息
function getMsg(arrayName)
{
  var arrayLength=arrayName.length;
  var tempMsg="";
  for(var i=0;i<arrayLength;i++)
  {
    tempMsg+="              MyArray["+i+"]="+arrayName[i]+"\n";
  }
  return tempMsg;
}
//执行相关操作
function MyTest()
{
  var MyArray=new Array("First","Second","Third","Forth");
  var msg="调用 Array 对象的方法生成字符串实例: \n\n";
```

```
    msg+="原始数组 ：\n"+getMsg(MyArray)+"\n";
    var tempStr="";
    tempStr=MyArray.join();
    msg+="1、调用 MyArray.join()方法返回字符串 ：\n"+tempStr+"\n\n";
    tempStr=MyArray.join("-");
    msg+="2、调用 MyArray.join("-")方法返回字符串 ：\n"+tempStr+"\n\n";
    msg+="3、调用 MyArray.toString()方法返回字符串 ：\n"+MyArray+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.22 所示。

图 6.22 调用 Array 对象的方法生成字符串实例

Array 对象的 join()方法由两种调用方式：

```
MyArray.join();
MyArray.join(string);
```

join()方法将数组中所有元素转化为字符串，并将这些串由逗号隔开合并成一个字符串作为方法的结果返回。如果调用时给定参数 string，就将 string 作为在结果字符串中分开由各个数组元素形成的字符串的分隔符。

toString()方法返回一个包含数组中所有元素，且元素之间以逗号隔开的字符串。该方法在将数组作为字符串使用时强制使用，且不需显性申明此方法的调用。

在 Navigator 3+浏览器获得支持的还有一种操作数组并返回字符串的方法，即 Array 对象的 toSource()方法：

```
MyArray.toSource();
```

该方法返回一个字符串表示该 Array 对象的源定义，其包含数组中所有元素，元素之间用逗号隔开，整个字符串用方括号“[]”括起表示它是一个数组。

注意：Array 对象的 toSource()方法在 IE 等浏览器环境中没有获得很好的支持，但属于 ECMAScript 2.0+

内置的标准方法。

6.3.7 连接两个数组

Array 对象提供 concat()方法将以参数传入的数组连接到目标数组的后面，并将结果返回新数组，从而实现数组的连接。concat()方法的语法如下：

```
var  myNewArray=MyArray.concat(arg1,arg2,…,argN);
```

该方法将按照参数的顺序将它们添加到目标数组 MyArray 的后面，并将最终的结果返回新数组 myNewArray。考察如下代码：

```
//源程序 6.17
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//输出数组信息
function getMsg(arrayName)
{
  var arrayLength=arrayName.length;
  var tempMsg="        长度 ："+arrayLength+"\n";
  for(var i=0;i<arrayLength;i++)
  {
    tempMsg+="        ["+i+"]="+arrayName[i];
    if((i+1)%3==0)
      tempMsg+="\n";
  }
  return tempMsg;
}
//执行相关操作
function MyTest()
{
  var MyArray=new Array("First","Second","Third");
  var ArrayToAdd1=new Array("Forth","Fifth");
  var ArrayToAdd2=new Array("Sixth");
  var msg="使用 concat()方法连接数组实例:\n\n";
  msg+="目标数组:\n"+getMsg(MyArray)+"\n";
  msg+="参数数组 ArrayToAdd1:\n"+getMsg(ArrayToAdd1)+"\n";
  msg+="参数数组 ArrayToAdd2:\n"+getMsg(ArrayToAdd2)+"\n\n";
  var myNewArray=MyArray.concat(ArrayToAdd1,ArrayToAdd2);
  msg+="使用 concat()方法产生新数组:\n"+getMsg(myNewArray)+"\n";
  msg+="目标数组 MyArray:\n"+getMsg(MyArray)+"\n";
  msg+="参数数组 ArrayToAdd1:\n"+getMsg(ArrayToAdd1)+"\n";
  msg+="参数数组 ArrayToAdd2:\n"+getMsg(ArrayToAdd2)+"\n";
  alert(msg);
}
-->
</script>
</head>
```

```
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=数组测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.23 所示。

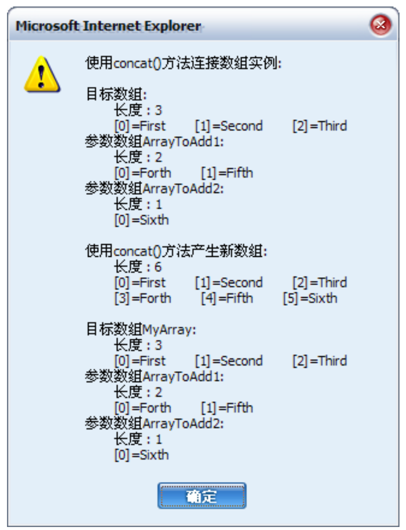

图 6.23 使用 concat()方法连接数组实例

由使用 concat()方法后目标数组和参数数组的内容不变可知，concat()方法并不修改数组本身，而是将操作结果返回给新数组。

6.3.8 常见属性和方法汇总

JavaScript 脚本的核心对象 Array 提供大量的属性和方法来操作数组。表 6.5 列出了其常用的属性、方法以及脚本版本支持情况。

表 6.5 Array对象常用的属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	Array.length	返回数组的长度，为可读可写属性	③⑥
	Array.prototype	用来给Array对象添加属性和方法	③⑥
方法	Array.concat(arg1,arg2,…argN)	将参数中的元素添加到目标数组后面并将结果返回到新数组	④⑥
	Array.join() Array.join(string)	将数组中所有元素转化为字符串，并把这些字符串连接成一个字符串，若有参数string，则表示使用string作为分开各个数组元素的分隔符	③⑥
	Array.pop()	删除数组末尾的元素并将数组length属性值减1	④
	Array.push(arg1,arg2,…,argN)	把参数中的元素按顺序添加到数组的末尾	④
	Array.reverse()	按照数组的索引号将数组元素的顺序完全颠倒	③⑥
	Array.shift()	删除数组的第一个元素并将该元素作为操作的结果返回。删除后所有剩下的元素将下移1位	④
	Array.slice(start)	返回包含参数start和stop之间的数组元素的新数	④⑥

	Array.slice(start,stop)	组，若无stop参数，则默认stop为数组的末尾	
	Array.sort() Array.sort(function)	基于一种顺序重新排列数组的元素。若有参数，则它表示一定的排序算法。详情见6.3.2实例	③⑥
	Array.splice(start,delete,arg3,…,argN)	按参数start和delete的具体值添加、删除数组元素。详情请见6.3.4应用实例	④
	Array.toSource()	返回一个表示Array对象源定义的字符串	④⑥
	Array.toString()	返回一个包含数组中所有元素的字符串，并用逗号隔开各个数组元素	③⑥

JavaScript 核心对象 Array 为我们提供了访问和操作数组的途径，使 JavaScript 脚本程序开发人员很方便、快捷操作数组这种储存数据序列的复合类型。

6.4 Date 对象

在 Web 应用中，经常碰到需要处理时间和日期的情况。JavaScript 脚本内置了核心对象 Date，该对象可以表示从毫秒到年的所有时间和日期，并提供了一系列操作时间和日期的方法。在深入了解 Array 对象前，首先了解两个有关时间标准的概念：

- GMT：格林尼治标准时间的英文缩写，指位于伦敦郊区的皇家格林尼治天文台的标准时间，该地点的经线被定义为本初子午线。理论上来说，格林尼治标准时间的正午是指当太阳横穿格林尼治子午线时的时间。
- UTC：世界协调时间的英文缩写，是由国际无线电咨询委员会规定和推荐,并由国际时间局(BIH)负责保持的以秒为基础的时间标度，相当于本初子午线上的平均太阳时。由于地球自转速度不规则，目前采用由原子钟授时的 UTC 作为时间标准。

在大多数情况下，可以假定 GMT 时间和 UTC 时间一致，电脑的时钟严格按照 GMT 时间运行。

JavaScript 脚本中采用 UNIX 系统存储时间的人工方式，即以毫秒数存储内部日期。同时，脚本在读取当前日期和时间时，依赖于客户端电脑的时钟，如果客户端电脑时钟有误，将造成一定的问题。

> 注意：为方便表述，将 GMT 时间 1970 年 1 月 1 日 0 点定义为 GMT 标准零点，下同。

6.4.1 生成日期对象的实例

Date 对象的构造函数通过可选的参数，可生成表示过去、现在和将来的 Date 对象。其构造方式有四种，分别如下：

```
var MyDate=new Date();
var MyDate=new Date(milliseconds);
var MyDate=new Date(string);
var MyDate=new Date(year,month,day,hours,minutes,seconds,milliseconds);
```

第一句生成一个空的 Date 对象实例 MyDate，可在后续操作中通过 Date 对象提供的诸多方法来设定其时间，如果不设定则代表客户端当前日期；在第二句的构造函数中传入唯一参数 milliseconds，表示构造与GMT标准零点相距 milliseconds 毫秒的 Date 对象实例 MyDate；第三句构造一个用参数 string 指定的 Date 对象实例 MyDate，其中 string 为表示期望日期的字符串，符合特定的格式；第四句通过具体的日期属性，如 year、month 等构造指定的 Date 对象实例 MyDate。

考察如下的代码：

```
//源程序 6.18
```

```
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var msg="生成日期对象实例 ：  \n\n";
  var MyDate4=new Date(2007,8,1,19,8,20,100);
  var MyDate1=new Date();
  MyDate1=MyDate4;
  var MyDate2=new Date(MyDate1.getTime());
  var MyDate3=new Date(MyDate1.toString());
  msg+="MyDate1   :   "+MyDate1.toString()+"\n";
  msg+="MyDate2   :   "+MyDate2.toString()+"\n";
  msg+="MyDate3   :   "+MyDate3.toString()+"\n";
  msg+="MyDate4   :   "+MyDate3.toString()+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=日期测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.24 所示。

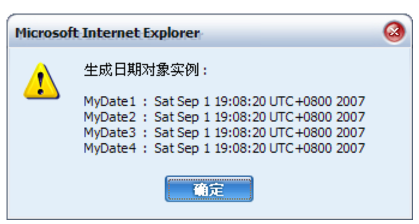

图 6.24 生成 Date 对象实例

该程序分为几步：

（1）通过第四种构造方式构造代表 2007 年 9 月 1 日 17 点 8 分 20 秒 100 毫秒的 Date 对象实例 MyDate4；

（2）通过第一种构造方式构造空的 Date 对象实例 MyDate1，并通过对象拷贝的方法，将 MyDate4 复制到 MyDate1；

（3）通过 Date 对象的 getTime()方法返回 MyDate4 表示的时间与 GMT 标准零点之间的毫秒数，然后将此毫秒数作为参数通过第二种构造方式构造 Date 对象实例 MyDate2；

（4）通过 Date 对象的 toString()方法返回表示 Date 对象实例 MyDate4 代表的时间的字符串，然后将此字符串作为参数通过第三种构造方式构造 Date 对象实例 MyDate3。

通过程序结果可知，上述四种构造 Date 对象实例的方式都能构造同样的日期对象。但上述的四种构造方法都是以当地时间创建新日期，JavaScript 提供了 UTC()方法按世界时间创建日期，语法如下：

```
date.UTC(year,month,day,hours,minutes,seconds,millseconds);
```

UTC()方法的参数意义与上述的第四种方法完全相同，只不过构建的对象基于 UTC 世界时间而已，如下的代码显示两者的不同之处：

```
var MyDate=new Date();
msg+="本地时间："+MyDate.toString()+"\n";
msg+="世界标准时间："+MyDate.toUTCString();
document.write(msg);
```

上述代码的运行后，返回：

本地时间：Fri Aug 3 21:21:43 UTC+0800 2007

世界标准时间：Fri, 3 Aug 2007 13:21:43 UTC

可以看出两者之间的差异，本地时间（东 8 区：北京时间）与 UTC 世界标准时间之间相差 8 小时，且输出字符串格式不同。

注意：欧美时间制中，星期及月份数都从 0 开始计数。如星期中第 0 天为 Sunday，第 7 天为 Saturday；月份中的第 0 月为 January，第 11 月为 December。但月的天数从 1 开始计数。

6.4.2 如何提取日期各字段

Date 对象以目标日期与 GMT 标准零点之间的毫秒数来储存该日期，给脚本程序员操作 Date 对象带来一定的难度。为解决这个难题，JavaScript 提供大量的方法而不是通过直接设置或读取属性的方式来设置和提取日期各字段，这些方法将毫秒数转化为对用户友好的格式。下面的程序以中文方式在文本框中动态显示系统时间：

```
//源程序 6.19
<! DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body onLoad="StartClock()">
<br><br>
<script language="JavaScript">
<!--
var timerID = null;
var timerRunning = false;
//获取日期各字段并赋值
function MyTimer()
{
  //生成 Date 对象实例，同时通过其方法提取日期的各字段
  var nowTime = new Date();
  var iyear = nowTime.getYear();
  var imonth = nowTime.getMonth();
  var iweek = nowTime.getDay();
```

```
   var idate = nowTime.getDate();
   var ihours= nowTime.getHours();
   var iminutes = nowTime.getMinutes();
   var iseconds = nowTime.getSeconds();
   //通过函数返回月份和日期的英文形式
   var Month = MyMonth(imonth);
   var Week = MyWeek(iweek);
   //设定输出格式
   var myDate = ((idate<10) ? "0" : "") + idate;
   var AM_PM = (ihours>= 12) ? "P.M." : "A.M.";
   var tempHours = (ihours>= 12) ? (ihours-12) : ihours;
   var Hours = ((tempHours < 10) ? "0" : "") + tempHours;
   var Minutes = ((iminutes < 10) ? ":0" : ":") + iminutes;
   var Seconds = ((iseconds < 10) ? ":0" : ":") + iseconds;
   //设定输出字符串
   var iTime = (Week+ "    " +Month+ "    " +myDate+ "," +iyear+ "    " +Hours+Minutes+Seconds+ " "
+AM_PM);
   document.MyForm.MyText.value=iTime;
   //设定定时器及时更新文本框内容
   timerID=setTimeout("MyTimer()",1000);
   timerRunning = true;
}
//转换月份，0 对应一月份，依此类推
function MyMonth(month)
{
   var strMonth;
   if(month==0) strMonth = "January";
   if(month==1) strMonth = "February";
   if(month==2) strMonth = "March";
   if(month==3) strMonth = "April";
   if(month==4) strMonth = "May";
   if(month==5) strMonth = "June";
   if(month==6) strMonth = "July";
   if(month==7) strMonth = "August";
   if(month==8) strMonth = "September";
   if(month==9) strMonth = "October";
   if(month==10) strMonth = "November";
   if(month==11) strMonth = "December";
   return strMonth;
}
//转换星期，0 对应星期日，依此类推
function MyWeek(week)
{
   var strWeek;
   if(week==0) strWeek = "Sunday";
   if(week==1) strWeek = "Monday";
   if(week==2) strWeek = "Tuesday";
   if(week==3) strWeek = "Wednesday";
   if(week==4) strWeek = "Thursday";
   if(week==5) strWeek = "Friday";
   if(week==6) strWeek = "Saturday";
   return strWeek;
}
```

```
//文档载入的同时启动定时器
function StartClock()
{
  if(timerRunning)
    clearTimeout(timerID);
  timerRunning = false;
  MyTimer();
}
-->
</script>
<center>
<form name="MyForm">
  <input type=text name="MyText" size=40>
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.25 所示，并根据客户端时钟及时更新文本框内容。

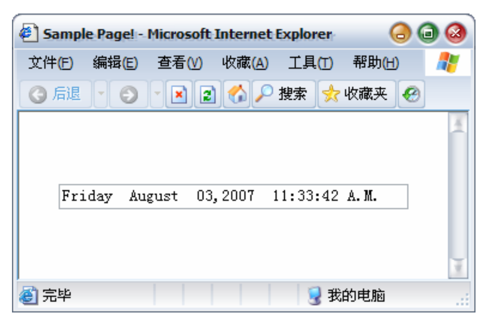

图 6.25 提取日期各字段实例

上述代码主要包括如下内容：

（1）MyTimer()函数：该函数首先构造空的 Date 对象实例 nowTime，用于保存当前系统的日期信息，然后通过 Date 对象的各种提取日期中信息的方法，获得如年、月、日、时、分、秒等信息，并更改输出格式；启动定时器 timerID 以及时更新 Date 对象实例 nowTime；

（2）MyWeek(week)函数：该函数将以数值参数传入的星期转化为英文表示的星期，并把结果以字符串的形式返回；

（3）MyMonth(month)函数：该函数将以数值参数传入的月份转化为英文表示的月份，并把结果以字符串的形式返回；

（4）StartClock()函数：该函数用于在文档载入时响应其 onload()事件，并设置初始状态，并将主动权交给 MyTimer()函数。

提取日期各字段的关键代码如下：

```
var nowTime = new Date();
var iyear = nowTime.getYear();
var imonth = nowTime.getMonth();
var iweek = nowTime.getDay();
var idate = nowTime.getDate();
var ihours= nowTime.getHours();
var iminutes = nowTime.getMinutes();
var iseconds = nowTime.getSeconds();
```

上述代码依次为构造用于保存当前日期的空对象和获取年、月、星期、日、小时、分、

秒等日期字段，并分别用变量保存个各字段信息，用于后续处理。

上述日期都是客户端日期，Date 对象也提供了基于 UTC 世界标准时间提取目标日期中各字段的诸多方法，如 getUTCDay()、getUTCSeconds()等。这些方法的使用过程与实例中的相同，只不过操作的基础不是客户端日期，而是 UTC 世界标准时间。

理解了如何从现有 Date 对象实例中提取日期各字段的问题后，下面了解设置日期中的各字段的方法。

6.4.3 如何设置日期各字段

在实际应用中，通常需要在原有的日期基础上得到新的日期，如电影中的“二十年后”等。Date 对象提供一系列的操作日期的方法。考察如下代码：

```
//源程序 6.20
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
   var MyDate=new Date(2007,8,1,19,8,20,100);
   var msg="设置日期各字段实例 : \n\n";
   msg+="原始日期   :   "+MyDate.toString()+"\n\n";
   var temp="";
   temp=MyDate.setFullYear(1997);
   msg+="setFullYear(1997)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setYear(98);
   msg+="setYear(98)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setYear(1999);
   msg+="setYear(1999)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setMonth(3);
   msg+="setMonth(3)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setDate(21);
   msg+="setDate(21)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setHours(11);
   msg+="setHours(11)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setMinutes(45);
   msg+="setMinutes(45)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   temp=MyDate.setSeconds(59);
   msg+="setSeconds(59)方法-->返回 : " +temp+ "ms       MyDate:"+MyDate.toString()+"\n";
   MyDate.setTime(21234234);
   msg+="setTime(21234234)方法-->返回 : MyDate:"+MyDate.toString()+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
```

```
<br>
<center>
<form name=MyForm>
  <input type=button value=日期测试 onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.26 所示。

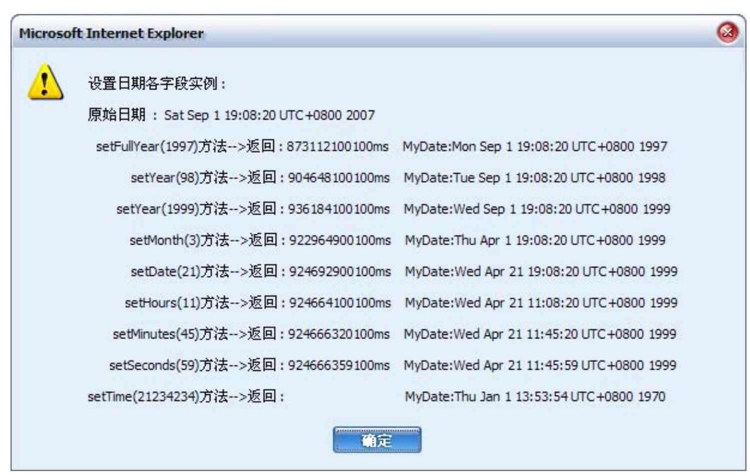

图 6.26 设置日期各字段的方法

使用 Date 对象的方法设置目标日期的指定字段之后，JavaScript 脚本更改目标日期的内容，同时将该新日期与 GMT 标准零点之间相距的毫秒数作为操作的结果返回。

注意：Date 对象的 setYear()方法可以接受 2 位或者 4 位的数字作为参数，其中 2 位的参数加上 1900 的结果作为设置的年份，但这种方法会带来二义性，一般应使用 4 位数值型参数。为解决此问题，Date 对象另外提供了 setFullYear()方法，该方法通过传入 4 位数值型参数实现同样的功能。

上述使用的都是本地时间，在 JavaScript 脚本中也可使用 UTC 标准世界时间作为操作的标准，同样存在诸如 setUTCDate()、setUTCMonth()等诸多的方法。

6.4.4 将日期转化为字符串

Date 对象提供如 toGMTString()、toLocalString()等方法将日期转换为字符串，而不需要开发人员编写专门的函数实现该功能。考察如下代码：

```
//源程序 6.21
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
```

```
    var MyDate=new Date();
    var msg="日期转化为字符串实例 : \n\n";
    msg+="本地日期 toString()   :   "+MyDate.toString()+"\n";
    msg+="本地日期 toLocalString()   :   "+MyDate.toLocaleString()+"\n";
    msg+="GMT 世界时间 toGMTString()   :   "+MyDate.toGMTString()+"\n";
    msg+="UTC 世界时间 toUTCString()   :   "+MyDate.toUTCString()+"\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=日期测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.27 所示。

图 6.27 日期转化为字符串实例

从程序结果可以看出，toString()和 toLocaleString()方法返回表示客户端日期和时间的字符串，但格式大不相同。实际上，toLocaleString()方法返回字符串的格式由客户设置的日期和时间格式决定，而 toString()方法返回的字符串遵循以下格式：

```
Fri Aug 3 22:49:00 UTC+0800 2007
```

由于目前 UTC 已经取代 GMT 作为新的世界时间标准，后面两种将日期转化为字符串的方法 toGMTString()和 toUTCString()返回的字符串格式、内容均相同。

同样，Date 对象提供了 parse()方法来将特定格式的字符串转化为毫秒数(目标日期与 GMT 标准零点的间隔)，后者可根据前面讲述的生成日期对象的第二种方法来生成表示该日期的 Date 对象，parse()方法的语法如下：

```
date.parse(date);
```

此方法与参数指定的对象而不是对象中的日期相联系，唯一的参数 date 应是使用 Date 对象的 toGMTString()方法生成的字符串格式：

```
Fri,3 August 2007 14:49:00 UTC
```

如果作为参数传入的表示日期的字符串不被 parse()方法认可，则 date.parse()方法返回 NaN 值。考察如下代码：

```
var MyDate=new Date();
var msg="字符串转化为时间实例 : \n\n";
msg+="转化为 GMT 世界时间的字符串 : \n"+MyDate.toGMTString()+"\n\n";
```

```
msg+="Date.parse(s1)方法返回毫秒数 : \n"+Date.parse(MyDate.toGMTString())+"ms\n\n";
var newDate=new Date(Date.parse(MyDate.toGMTString()));
msg+="通过返回的毫秒数生成的日期 : \n"+newDate.toString()+"\n\n";
var str="Friday,2002";
msg+="传入字符串 str : \n            "+str+"\n\n";
msg+="Date.parse(str)方法返回 : \n            "+Date.parse(str)+"\n\n";
alert(msg);
```

程序运行结果如图 6.28 所示。

图 6.28 字符串转化为时间实例

在老版本的浏览器中，toUTCString()方法和 toGMTString()方法返回的字符串不同，parse()方法只能识别 toGMTString()方法返回的字符串（也可接受缺失所有或部分时间或时区部分的字符串）。由于 UTC 世界时间取代 GMT 世界时间（实际上两者在某种意义上等同）成为世界时间标准，目前上述两种方法产生的字符串（或其子串）都可作为 parse()方法的参数传入以实现生成新的日期对象等功能。

6.4.5 常见属性和方法汇总

Date 对象提供成熟的操作日期和时间的诸多方法，方便脚本开发过程中程序员简单、快捷操纵日期和时间。表 6.6 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.6 Date对象常用的属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	prototype	允许在Date对象中增加新的属性和方法	③⑥
方法	getdate()	返回月中的某一天	①②
	getday()	返回星期中的某一天（星期几）	①②
	getFullyear()	返回用4位数表示的当地时间的年	④⑥
	getHours()	返回小时	①②
	getMillseconds()	返回毫秒	④⑥
	getMinutes()	返回分钟	①②
	getMonth()	返回月份	①②
	getseconds()	返回秒	①②
	getTime()	返回以毫秒表示的日期和时间	①②
	getTimezoneoffset()	返回以GMT为基准的时区偏差，以分计算	①②
	getUTCDate()	返回转换成世界时间的月中某一天	④⑥
	getUTCDay()	返回转换成世界时间的星期中的某一天（星期几）	④⑥
	getUTCFullyear()	返回转换成世界时间的4位数表示的当地时间的年	④⑥
	getUTCHours()	返回转换成世界时间的小时	④⑥

	getUTCMillseconds()	返回转换成世界时间的毫秒	④⑥
	getUTCSeconds()	返回转换成世界时间的分钟	④⑥
	getYear()	返回转换成世界时间的月份	①②
	parse()	返回转换成世界时间的秒	①②
	setDate()	设置月中的某一天	③⑥
	setFullyear()	按以参数传入的4位数设置年	④⑥
	setHours()	设置小时	①②
	setMillseconds()	设置毫秒	④⑥
	setMinutes()	设置分钟	①②
	setMonth()	设置月份	①②
	setseconds()	设置秒	①②
	setTime()	从一个表示日期和时间的毫秒数来设置日期和时间	①②
	setUTCDate()	按世界时间设置月中的某一天	④⑥
	setUTCFullyear()	按世界时间按以参数传入的4位数设置年	④⑥
	setUTCHours()	按世界时间设置小时	①②
	setUTCMillseconds()	按世界时间设置毫秒	④⑥
	setUTCMinute()	按世界时间设置分钟	④⑥
	setUTCMonth()	按世界时间设置月份	④⑥
	setUTCSeconds()	按世界时间设置秒	④⑥
	setYear()	以2位数或4位数来设置年	①②
	toGMTString()	返回表示GMT世界时间的日期和时间的字符串	①②
	toLocaleString()	返回表示当地时间的日期和时间的字符串	①②
	toSource()	返回Date对象的源代码	①⑥
	toString()	返回表示当地时间的日期和时间的字符串	①②
	toUTCString()	返回表示UTC世界时间的日期和时间的字符串	④⑥
	toUTC()	将世界时间的日期和时间转换位毫秒	①②

Date 对象提供大量的方法来方便脚本程序开发人员快捷操作日期对象，但在早期的浏览器及其版本（特别是 Netscape Navigator）中的支持较差，经常出现莫名其妙的错误。但在 Netscape 4+和 Internet Explorer 4+后逐步得到较为完善的支持，有效减少了错误，还可以处理 GMT 标准零点之前或之后成千上百年的日期，足以解决大部分的问题。

6.5 Number 对象

Number 对象对应于原始数值类型和提供数值常数的对象，可通过为 Number 对象的构造函数指定参数值的方式来创建一个 Number 对象的实例。其中的参数符合 IEEE 754-1985 标准，采用双精度浮点数格式表示，占用 64 字节，允许浮点量级不大于$\pm1.7976\times10^{308}$同时不小于$\pm2.2250*10^{-308}$。

6.5.1 创建 Number 对象的实例

创建 Number 对象的实例

语法如下：

```
var MyData=new Number();
var MyData=new Number(value);
```

第一句构造一个空的 Number 对象实例 MyData；第二句构造一个 Number 对象的实例 MyData，同时用通过参数传入的参数 value 初始化。考察如下代码：

```
//源程序 6.22
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var MyData1=new Number();
  var MyData2=new Number(312);
  var msg="构造 Number 对象的实例 ：\n\n";
  msg+="构造方法 ：\n";
  msg+="        语句 1 : var MyData1=new Number();\n";
  msg+="        语句 2 : var MyData1=new Number(312);          \n";
  msg+="构造结果 ：\n";
  msg+="        MyData1  :  "+MyData1+"\n";
  msg+="        MyData2  :  "+MyData2+"\n";
  MyData1=1200;
  MyData2=2400;
  msg+="更改内容方法 ：\n";
  msg+="        语句 1 : MyData1=1200;\n";
  msg+="        语句 2 : MyData2=2400;\n";
  msg+="更改内容结果 ：\n";
  msg+="        MyData1  :  "+MyData1+"\n";
  msg+="        MyData2  :  "+MyData2+"\n";
  alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
  <input type=button value=日期测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.29 所示。

图 6.29 构造 Number 对象实例

通过第一种方法构造空的 Number 对象实例后，JavaScript 语言给其赋默认值 0；第二种方法中构造 Number 对象实例，并用参数 312 初始化。在后续操作中，都可通过直接赋值的方式更改其内容。

6.5.2 将 Number 对象转化为字符串

使用上述的方法构造 Number 对象的实例后，可调用 Number 对象的 toString()方法将其转化为字符串。考察如下代码：

```
var MyData=new Number(312);
var str=MyData.toString();
var msg="Number 兑现实例转化为字符串 ：\n\n";
msg+="转化方法 ：\n";
msg+="            语句 ：var str=MyData.toString();\n";
msg+="转化结果 ：\n";
msg+="            MyData1   :   "+str+"\n";
msg+="            str 类型   :   "+typeof(str)+"\n";
alert(msg);
```

程序运行结果如图 6.30 所示。

图 6.30 Number 对象实例转化为字符串

事实上，这种转换必要性不大，因为在需要转换的时候，JavaScript 会自动将该实例转换为对应的字符串。

6.5.3 通过 prototype 属性添加属性和方法

在实际应用中，经常要为同一种数据类型定义一种临时、通用的方法，如 Date 对象的 setYear()方法可以接受 2 位和 4 位数字来修改年份，方法识别传入的参数，如果是 2 位数字，则自动加上 1900，然后将结果返回构造函数，以便正确生成目标年份。可以想象在其中存在一个函数 add1900：

```
function add1900(value)
{
  var reNum;
  var errorMsg="error!";
  if(value.isNumber==1)
  {
    if(value.length==2)
       reNum=value+1900;
    else
       reNum=value;
    return reNum;
```

```
   }
   else
      return errorMsg;
}
```

Number 对象除了 toString()方法外，不支持任何方法进行数值运算，但开发者能够通过其 prototype 属性来扩充其属性和方法。下面的代码演示如何给 Number 对象添加新的方法 newFunc()并将其指向其具体实现函数，具体功能是返回 Number 对象的实例代表的数值的正弦值并加 1，然后将结果返回：

```
//源程序 6.23
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MySin(num)
{
   var result=Math.sin(num*Math.PI/180)+1;
   return result;
}
function MyTest()
{
   var MyData=new Number();
   Number.prototype.newFunc=MySin;
   var ivalue=document.MyForm.MyText.value;
   var tempData=MyData.newFunc(ivalue);
   var msg="使用 prototype 属性添加方法实例 : \n\n";
   msg+="添加方法语句 : \n";
   msg+="              Number.prototype.newFunc=MySin;\n";
   msg+="触发函数 : MySin()\n";
   msg+="              结果 : sin(" +ivalue+"•)+1="+MyData.newFunc(ivalue)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<p>文本框数值代表目标角度,单击按钮得到其正弦值</p>
<form name=MyForm>
   <input type=text name=MyText size=20>
   <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.31 所示。

图 6.31 通过 prototype 属性添加对象的方法实例

语句：

```
Number.prototype.newFunc=MySin;
```

通过访问 Number 对象的 prototype 属性添加 newFunc()方法，并将该方法的具体实现指向自定义的函数 MySin()；然后通过调用该方法实现相应的功能。通过 Number 对象的 prototype 属性给 Number 对象添加属性的过程更简单，只需设定新属性的名称并直接赋值即可。

注意：使用 Number 对象的 prototype 属性给 Number 对象添加的属性和方法作用范围仅限于当前代码范围，超出该代码范围新的属性和方法将失效，此特性也适用于其他 JavaScript 内置核心对象。

6.5.4 常见属性和方法汇总

Number 对象为 JavaScript 核心对象中表示数值类型的对象，拥有较少的属性和方法，表 6.7 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.7 Number对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	MAX_VALUE	指定脚本支持的最大值	③⑦
	MIN_VALUE	指定脚本支持的最小值	③⑦
	NaN	为Not a Number的简写，表示一个不等于任何数的值	③⑦
	NEGTTIVE_INFINITY	表示负无穷大的特殊值	③⑦
	POSITIVE_INFINITY	表示正无穷大的特殊值	③⑦
	prototype	允许在Number对象中增加新的属性和方法	③⑦
方法	toSource()	返回表示当前Number对象实例的字符串	⑤
	toString()	得到当前Number对象实例的字符串表示	③⑦
	valueOf()	得到一个Number对象实例的原始值	③⑥

注意：在脚本版本支持一栏，⑦指 JScript 1.0+，其余代号如前述。

Number 对象为 JavaScript 脚本开发人员提供了一系列常用于数值判断的常量，同时可通过其 prototype 属性扩展 Number 对象的属性和方法。

6.6 Boolean 对象

Boolean 对象是对应于原始逻辑数据类型的内置对象，它具有原始的 Boolean 值，只有 true 和 false 两个状态，在 JavaScript 脚本中，1 代表 true 状态，0 代表 false 状态。

6.6.1 创建 Boolean 对象的实例

Boolean 对象的实例可通过使用 Boolean 对象的构造函数、new 操作符或 Boolean()函数来创建：

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

第一句通过 Boolean 对象的构造函数创建对象的实例 MyBool，并用 Boolean 对象的默认值 false 将其初始化；第二句通过 Boolean 对象的构造函数创建对象的实例 MyBool，并用以参数传入的 value 值将其初始化；第三句使用 Boolean()函数创建 Boolean 对象的实例，并用以参数传入的 value 值将其初始化。

下面的代码用不同的方式创建 Boolean 对象的实例，并使用 typeof 操作符返回其类型：

```
//源程序 6.24
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  var MyBoolA=new Boolean();
  var MyBoolB=new Boolean(false);
  var MyBoolC=new Boolean("false");
  var MyBool=Boolean(false);
  var msg="创建 Boolean 对象实例 ：\n\n";
  msg+="生成语句 ：\n";
  msg+="          var MyBoolA=new Boolean();\n";
  msg+="返回结果 ：\n";
  msg+="          MyBoolA = " +MyBoolA+ "        类型 ："+typeof(MyBoolA)+"\n\n";
  msg+="生成语句 ：\n";
  msg+="          var MyBoolB=new Boolean(false);\n";
  msg+="返回结果 ：\n";
  msg+="          MyBoolB = " +MyBoolB+ "        类型 ："+typeof(MyBoolB)+"\n\n";
  msg+="生成语句 ：\n";
  msg+="          var MyBoolC=new Boolean(''false'');\n";
  msg+="返回结果 ：\n";
  msg+="          MyBoolC = " +MyBoolC+ "        类型 ："+typeof(MyBoolC)+"\n\n";
  msg+="生成语句 ：\n";
  msg+="          var MyBool=new Boolean(false);\n";
  msg+="返回结果 ：\n";
```

```
    msg+="         MyBool   =   " +MyBool+ "        类型 : "+typeof(MyBool)+"\n\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.32 所示。

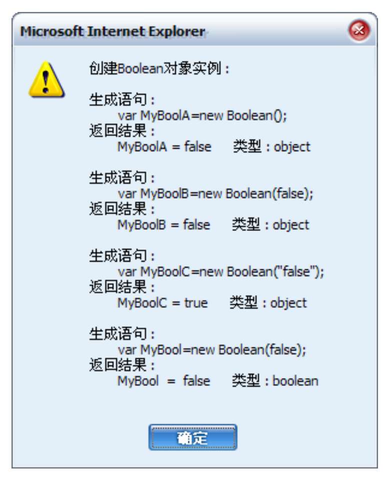

图 6.32 创建 Boolean 对象的实例

以下两点值得特别注意：

- 在第三种构造方式中，首先判断字符串“false”是否为 null，结果返回 true，并将其作为参数通过 Boolean 构造函数创建对象，故其返回 MyBoolC=true；
- 在第四种构造方式中，生成的 MyBool 仅仅为一个包含 Boolean 值的变量，其类型与前面三种不同，为 boolean 而不是 object。

Boolean 对象构造完成后，可通过直接对实例赋值的方式修改其内容。在实际构造过程中，要灵活运用这几种构造的方法，并理解其间的不同点和相似之处。

注意：在创建 Boolean 对象实例过程中，如果传入的参数为 null、NaN、""或者 0 将自动变成 false，其余的将变成 true。

6.6.2 将 Boolean 对象转化为字符串

由于 Boolean 对象继承自 Object 对象，后者为其提供 toString()方法将其代表的状态转化为字符串“true”或“false”，进行后续操作。考察如下代码：

```
//源程序 6.25
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
```

```
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
   var MyBool=new Boolean(false);
   var msg="转换 Boolean 对象为字符串实例 : \n\n";
   msg+="生成语句 : \n";
   msg+="          var MyBool=new Boolean(); \n";
   msg+="返回结果 : \n";
   msg+="          MyBool = " +MyBool+ "          类型 : "+typeof(MyBool)+"\n\n";
   var MyStr=MyBool.toString();
   msg+="转换语句 : \n";
   msg+="          var MyStr=MyBool.toString();          \n";
   msg+="转换结果 : \n";
   msg+="          MyStr = " +MyStr+ "          类型 : "+typeof(MyStr)+"\n";
   alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
   <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.33 所示。

图 6.33 将 Boolean 对象转化为字符串实例

可以明显看出，使用 toString()方法，返回的 MyStr="false"为字符串。

在 Boolean 对象中，还有一种方法将 Boolean 对象转为字符串：toSource()方法，该方法返回一个表示对象创建代码的字符串，并应括号括起。如：

```
var MyBool=new Boolean(true);
var str=MyBool.toSource();
```

```
document.write("str="+str+"<br>type:"+typeof(str));
```

代码执行后，返回字符串：

```
str=(new Boolean(true))
type:string
```

可见，toSource()方法返回的是用括号括起来的表示 Boolean 对象的字符串，与 toString()方法返回的字符串存在根本的不同。

6.6.3 常见属性和方法汇总

Boolean 对象为 JavaScript 脚本语言的封装对象，表示原始的逻辑状态 true 和 false，表 6.8 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.8 Boolean对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	prototype	允许在Boolean对象中增加新的属性和方法	③⑥
方法	toSource()	返回表示当前Boolean对象实例创建代码的字符串	⑤⑥
	toString()	返回当前Boolean对象实例的字符串（"true"或"false"）	③⑥
	valueOf()	得到一个Boolean对象实例的原始Boolean值	③⑥

在实际应用中，常通过 prototype 属性扩展 Boolean 对象的属性和方法，开发者要十分注意几种创建 Boolean 对象实例的方式的相似点和不同之处。

6.7 Function 对象

JavaScript 核心对象 Funtion 为构造函数的对象，由于开发者一般直接定义函数而不是通过使用 Function 对象创建实例的方式来生成函数，对于实际编程而言，Function 对象很少涉及到，但正确地理解它有助于开发者加深对 javaScript 脚本中函数概念的理解。

6.7.1 两个概念：Function 与 function

简而言之，Function 是对象而 function 是函数。实际上，在 JavaScript 中声明一个函数本质上为创建 Function 对象的一个实例，而函数名则为实例名。先看如下的函数：

```
function sayHello(username)
{
  alert("Hello "+name);
}
```

输入参数“NUDT!”，返回警告框如图 6.34 所示。

图 6.34 sayHello()函数的返回警告框

如果通过创建 Function 对象的实例的方式来实现该功能，代码如下：

```
var sayHello = new Function("name","alert('Hello '+name)");
```

在该方式中,第一个参数是函数 sayHello()的参数，第二个参数是函数 sayHello()的函数体。定义之后，可通过调用 sayHello("NUDT!")的方式获得上述的结果。

通过两种构造方式的对比，可以看出所谓的函数只不过是 Function 对象的一个实例，而函数名为实例的名称。

既然函数名为实例的名称，那么就可以将函数名作为变量来使用。考察如下的代码：

```
function sayHello()
{
  alert("Hello");
}
function sayBye()
{
  alert("Bye");
}
sayHello = sayBye;
```

上述代码运行后，再次调用 sayHello()函数，返回的是“Bye”而不是“Hello”。

6.7.2 使用 Function 对象构造函数

在 JavaScript 中，构造函数常用如下的两种方法：

- 函数的原始构造方法：

```
function functionName([argname1 [, ...[, argnameN]]])
{
   body
}
```

- 创建 Function 对象实例的方法：

```
functionName = new Function( [arg1, [... argN,]],body );
```

其中 functionName 是创建的目标函数名称，为必选项；arg1，...，argN 是函数接收的参数列表，为可选项。函数接收的参数列表；body 是函数体，包含调用此函数时执行的代码，为可选项。

举个执行两个数加法的程序，使用第一种构造方法：

```
function add(x, y)
{
   return(x + y);
}
```

如果采用创建 Function 对象实例的方式实现同样的功能，如下代码：

```
var add = new Function("x", "y", "return(x+y)");
```

在这两种情况下，都通过 add(4, 5)的方式调用目标函数。第二种构造方式适用于参数较少、函数代码比较简单的情形，而第一种方式代码层次感较强，且对代码的复杂程度和参数多少并无特别的规定。

鉴于以上原因，脚本开发人员主要使用第二种构造方法来构造函数，但理解 Function 对象对开发者深入理解函数的本质有很大的帮助。

6.7.3 常见属性和方法汇总

Function 对象在脚本编程中使用不是很广泛，表 6.9 列出了其常用的属性、方法以及脚

本版本支持情况：

表 6.9 Function对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	arguments	包含传给函数的参数，只能在函数内部使用	③⑦
	arity	表示一个函数期望接收的参数数目	④
	caller	用来访问调用当前正在执行的函数的函数	③⑦
	prototype	允许在Function对象中增加新的属性和方法	③⑦
方法	apply()	将一个Function对象的方法使用在其他Function对象上	⑤⑦
	call()	该方法允许当前对象调用另外一个Function对象的方法	
	toSource()	允许创建一个Function对象的拷贝	⑤⑦
	toString()	将定义函数的JavaScript源代码转换为字符串并将其作为调用此方法的结果返回	③⑥

在实际应用中，经常使用 apply()和 call()方法将一个 Function 对象的方法使用在其他对象上，实现脚本代码的重用。

6.8 Object 对象

所有的 JavaScript 对象都继承自 Object 对象，后者为前者提供基本的属性（如 prototype 属性等）和方法（如 toString()方法等）。而前者也在这些属性和方法基础上进行扩展，以支持特定的某些操作。

6.8.1 创建 Object 对象的实例

Object 对象的实例构造方法如下：

```
var MyObject=new Object(string);
```

上述语句构造 object 对象的实例 MyObject，同时用以参数传入的 string 初始化对象实例，该实例能继承 object 对象提供的几个方法进行相关处理。参数 string 为要转为对象的数字、布尔值或字符串，此参数可选，若无此参数，则构建一个未定义属性的新对象。

JavaScript 脚本支持另外一种构造 Object 对象实例的方法：

```
var MyObject={name1:value1,name2:value2,...,nameN:valueN};
```

该方法构造一个新对象，并使用指定的 name1，name2，…，nameN 指定其属性列表，使用 value1，value2，…，valueN 初始化该属性列表。

考察如下代码：

```
//源程序 6.26
<! DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function MyTest()
{
  //第一种构造方法
  var myObject1=new Object();
```

```
    //为新对象添加属性
    myObject1.name="小李";
    myObject1.gender="女";
    myObject1.age=28;
    var msg="创建 Object 对象实例 ：\n\n";
    msg+="创建语句 ：\n";
    msg+="            var myObject1=new Object();\n";
    msg+="            myObject1.name="小李";\n";
    msg+="            myObject1.gender="女";\n";
    msg+="            myObject1.age=28;\n";
    msg+="创建结果 ：\n";
    msg+="            类型 ：" +typeof(myObject1)+ "\n";
    msg+="            属性 name：" +myObject1.name+ "\n";
    msg+="            属性 gender：" +myObject1.gender+ "\n";
    msg+="            属性 age：" +myObject1.age+ "\n\n";
    //第二种构造方法
    var myObject2={name:"小王",gender:"男",age:32};
    msg+="创建语句 ：\n";
    msg+="            var myObject2={name:"小王",gender:"男",age:32};          \n";
    msg+="创建结果 ：\n";
    msg+="            类型 ：" +typeof(myObject2)+ "\n";
    msg+="            属性 name：" +myObject2.name+ "\n";
    msg+="            属性 gender：" +myObject2.gender+ "\n";
    msg+="            属性 age：" +myObject2.age+ "\n\n";
    alert(msg);
}
-->
</script>
</head>
<body>
<br>
<center>
<form name=MyForm>
    <input type=button value=测试  onclick="MyTest()">
</form>
</center>
</body>
</html>
```

程序运行结果如图 6.35 所示。

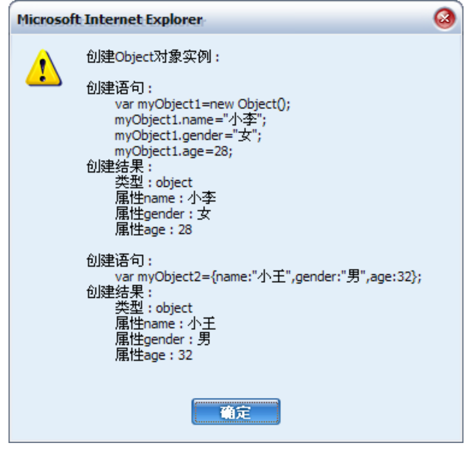

图 6.35 Object 对象两种不同的构造方式实例

由程序结果可见，这两种构造 Object 对象实例的方法得到相同的结果，相比较而言，第一种方法结构清晰、层次感强，而第二种方法代码简单、编程效率高。

6.8.2 常见属性和方法列表

通过从 Object 对象继承产生后，String、Math、Array 等对象获得了 Object 对象所有的属性和方法，同时扩充了之只属于自身的属性和方法，以对特定的目标进行处理。表 6.10 列出了其常用的属性、方法以及脚本版本支持情况：

表 6.10 Object对象常见属性、方法汇总

类型	项目及语法	简要说明	版本支持*
属性	constructor	指定对象的构造函数	③⑥
	prototype	允许在Object对象中增加新的属性和方法	③⑥
方法	eval()	通过当前对象执行一个表示JavaScript脚本代码的字符串	③⑥
	toSource()	返回创建当前对象的源代码	④
	toString()	返回表示对象的字符串	③⑥
	valueOf()	返回目标对象的值	③⑥

在 JavaScript 脚本编程实践中，由于 Object 对象的属性和方法都比较少，直接使用 Object 对象进行相关操作的情况很少，一般情况下使用由其派生的 Array、Math 等其他核心对象或者使用程序员自己定义、由 Object 对象派生的对象，并为其添加特定的属性和方法的方式来实现既定的功能。

6.9 本章小结

JavaScript 语言的核心对象由于其构成了脚本编程的基础，显得尤为重要。本章主要介绍了除 RegExp 对象之外的其余 JavaScript 核心对象，论述了其创建策略、常见操作并与实际需求相结合，列举了很多常用的操作实例，如邮箱地址验证、任意范围随机数发生器等。

很多核心对象与 JavaScript 支持的各种数据类型相关，程序员经常使用与数组、字符串等复杂数据类型相关的内置变量的属性和方法，但大多数程序员会忽略“基本类型也是对象”的事实。理解它们之间的联系可以使读者成为优秀的 JavaScript 脚本程序员，而不是普通的使用者。

前几章了解了 JavaScript 脚本的概念、语法、事件处理和内置的核心数据对象等基础知识，下一章将全面进入奇妙的文档结构模型（DOM）世界。

第 7 章　Window 及相关顶级对象

在“JavaScript 基于对象编程”一章中，读者基本理解了 Window 相关顶级对象及它们之间的关系。在“文档结构模型(DOM)”一章中，继续加深了这种认识并从对象模型层次关系的角度重点分析了对象的产生过程。本章将从实际应用的角度出发，讨论 Window、Frames、Navigator、Screen、History、Location、Document 等相关顶级对象的属性、语法及如何创建、使用等问题。通过本章的学习，读者应能使用 JavaScript 脚本生成并管理浏览器基本框架，并且熟悉框架之间的通信方法。

7.1 顶级对象模型参考

在 DOM 架构中，Window、Frames、Navigator 等顶级对象产生于浏览器载入文档至关闭文档期间的不同阶段，并起着互不相同且不可代替的作用，如 Window 对象在启动浏览器载入文档的同时生成，与当前浏览器窗口相关，包含窗口的最小最大化、尺寸大小等属性，同时具有关闭窗口、创建新窗口等方法；而 Location 对象以 URL 的形式载入当前窗口，并保存正在浏览的文档的位置及其构成信息，如协议、主机名、端口、路径、URL 的查询字符串部分等，顶级模型的结构如图 7.1 所示。

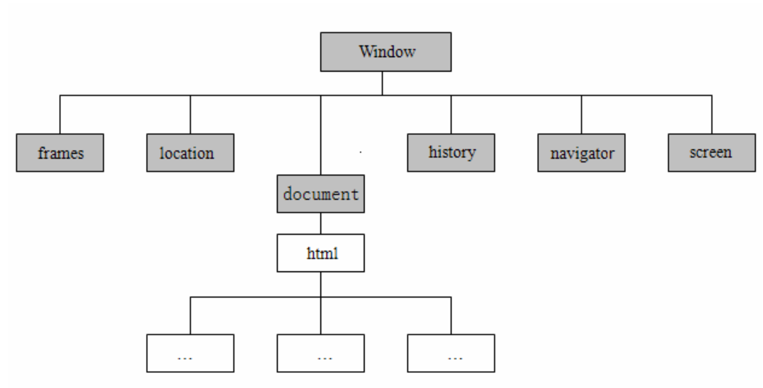

图 7.1 顶级对象模型层次结构

可见，Window 对象在层次中的最上层，而 Document 对象处于顶级对象的最底层。一般说来，Frames 对象在 Window 对象的下层，但当目前文档包含框架集时，该框架集中的每个框架都包含单独的 Window 对象；每个 Window 对象都直接包含一个（或者间接包含多个）Document 对象。首先来了解 Window 对象。

7.2 Window 对象

简而言之，Window 对象为浏览器窗口对象，为文档提供一个显示的容器。当浏览器载入目标文档时，打开浏览器窗口的同时，创建 Window 对象的实例，Web 应用程序开发者可通过 JavaScript 脚本引用该实例，从而进行诸如获取窗口信息、设置浏览器窗口状态或者新

建浏览器窗口等操作。同时，Window 对象提供一些方法产生图形用户界面中用于客户与页面进行交互的对话框（模式或者非模式），并能通过脚本获取其返回值然后决定浏览器后续行为。

由于 Window 对象是顶级对象模型中的最高级对象，对当前浏览器的属性和方法，以及当前文档中的任何元素的操作都默认以 Window 对象为起始点，并按照对象的继承顺序进行访问和相关操作，所以在访问这些目标时，可将引用 Window 对象的代码省略掉，如在需要给客户以警告信息的场合调用 Window 对象的 alert()方法产生警告框，直接使用 alert(targetStr)语句，而不需要使用 window.alert(targetStr)的方法。但在框架集或者父子窗口通信时，须明确指明要发送消息的窗口名称。

7.2.1 交互式对话框

使用 Window 对象产生用于客户与页面交互的对话框主要有三种：警告框、确认框和提示框等，这三种对话框使用 Window 对象的不同方法产生，功能和应用场合也不大相同。

1. 警告框

警告框使用 Window 对象的 alert()方法产生，用于将浏览器或文档的警告信息（也可能不是恶意的警告）传递给客户。该方法产生一个带有短字符串消息和“确定”按钮的模式对话框，且单击“确定”按钮后对话框不返回任何结果给父窗口。此方法的语法如下：

```
window.alert(Str);
alert(Str);
```

其中参数可以是已定义变量、文本字符串或者是表达式等。当参数传入时，将参数的类型强制转换为字符串然后输出：

```
var MyName="YSQ";
var iNum=1+1;
alert("\nHello " +MyName+ ":\n MyResult: 1+1=" +iNum+ "\n");
```

上述代码运行后，弹出警告框如图 7.2 所示。

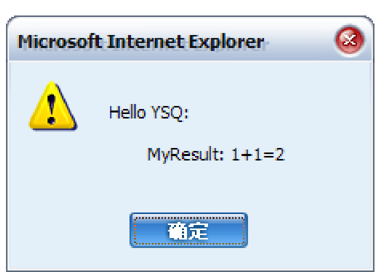

图 7.2 警告框实例

> 注意：模式对话框由父窗口创建，并使父窗口无效直到该对话框被关闭之后父窗口才能被用户激活，其内部拥有消息循环；非模式对话框由父窗口创建，对话框创建后立即返回，并与父窗口共用一个消息循环，在对话框被关闭之前，父窗口能被激活进行各种操作。

2. 确认框

确认框使用 Window 对象的 conform()方法产生，用于将浏览器或文档的信息（如表单提交前的确认等）传递给客户。该方法产生一个带有短字符串消息和“确定”、“取消”按钮的模式对话框，提示客户选择单击其中一个按钮表示同意该字符串消息与否，“确定”按钮表示同意，“取消”按钮表示不同意，并将客户的单击结果返回。此方法的语法如下：

```
answer=window.confirm(Str);
answer=confirm(Str);
```

其中参数可以是已定义变量、文本字符串或者是表达式等。当参数传入时，将参数的类型强制转换为字符串作为需要确认的问题输出。该方法返回一个布尔值表示消息确认的结果，true 表示客户同意了该消息，而 false 表示客户不同意该消息或确认框被客户直接关闭。考察下面代码：

```
var MyName="YSQ";
var iNum=1+1;
var answer=confirm("\nHello " +MyName+ ":\n MyResult: 1+1=" +iNum+ " ?\n");
if(answer==true)
  alert("\n 客户确认信息 ：\n\n"+"          算术运算 1+1=2 成立!");
else
  alert("\n 客户确认信息 ：\n\n"+"          算术运算 1+1=2 不成立!");
```

程序运行后，弹出确认框如图 7.3 所示。

图 7.3 确认框实例

若单击“确定”按钮，将弹出警告框如图 7.4 所示。

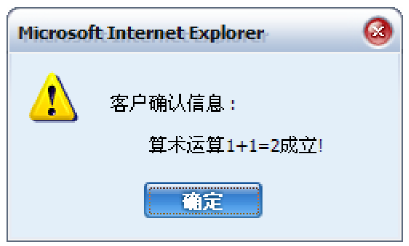

图 7.4 单击“确定”按钮，弹出警告框

若单击“取消”按钮或直接关闭该确认框，将弹出警告框如图 7.5 所示。

图 7.5 单击“取消”按钮或直接关闭确认框，弹出警告框

值得注意的是，确认框中事先设定的问题的提问方法有可能严重影响这个确认框的实用性，程序员在编制程序时，一定要将问题的返回值代表的客户意图与下一步的动作联系在一起，避免对结果进行完全相反的处理。

3. 提示框

提示框使用 Window 对象的 prompt()方法产生，用于收集客户关于特定问题而反馈的信息，该方法产生一个带有短字符串消息的问题和“确定”、“取消”按钮的模式对话框，提示客户输入上述问题的答案并选择单击其中一个按钮表示确定还是取消该提示框。如果客户单击了“确定”按钮则将该答案返回，若单击了“取消”按钮或者直接关闭则返回 null 值。此方法的语法如下：

```
returnStr=window.prompt(targetQuestion,defaultString);
returnStr=prompt(targetquestion,default string);
```

该方法通过参数 targetQuestion 传入一个字符串，代表需要客户回答的问题。通过参数 defaultString 传入一个默认的字符串，该参数一般可设定为空。当客户填入问题的答案并单击“确定”按钮后，该答案作为 prompt()方法的返回值赋值给 returnStr；当客户单击“取消”按钮时，prompt()方法返回 null。考察如下代码：

```
var answer=prompt("算术运算题目 ：1+1 = ?");
if(answer==2)
  alert("\n 算术运算结果 ：\n\n"+"恭喜您,你的答案正确! ");
else if(answer==null)
  alert("\n 算术运算结果 ：\n\n"+"对不起,您还没作答! ");
else
  alert("\n 算术运算结果 ：\n\n"+"对不起,您的答案错误! ");
```

程序运行后，弹出提示框如图 7.6 所示。

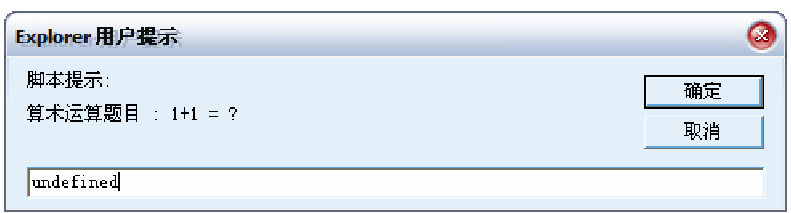

图 7.6 提示框实例

如果在上述提示框填入正确结果“2”，并单击“确定”按钮，弹出警告框如图 7.7 所示。

图 7.7 输入正确结果时弹出警告框

如果在上述提示框输入错误的答案，并单击“确定”按钮，弹出警告框如图 7.8 所示。

图 7.8 输入错误结果时弹出警告框

如果在上述提示框中单击“取消”按钮或直接关闭，弹出警告框如图 7.9 所示。

图 7.9 单击“取消”按钮或直接关闭时弹出警告框

使用 prompt()方法生成提示框返回客户的答案时，应注意考察提示框的返回值，然后采取进一步的动作。

4. 实例：学生信息采集系统

综合以上三种客户与浏览器交互的方法，可编制一个学生信息采集系统，该系统实现学生信息录入功能，并在数据合法性的检验方面进行了充分的考虑。考察系统中采集学生信息并验证数据合法性的页面代码：

```
//源程序 7.1
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//检查姓名长度是否合法
function CheckName(str)
{
  var nbool;
  var strLength=str.length;
  if(strLength>3&&strLength<9)
    nbool=true;
  else
    nbool=false;
  return nbool;
}
//检查性别是否合法
function CheckSex(sex)
{
  var nbool;
  if(sex=="male"||sex=="female")
    nbool=true;
  else
    nbool=false;
  return nbool;
}
//检查学号格式是否正确
```

```
function CheckNumber(num)
{
   var nbool;
   var numLength=num.length;
   var index=num.indexOf("2001");
   if(numLength!=8||index==-1)
      nbool=false;
   else
      nbool=true;
   return nbool;
}
//检查邮箱格式是否正确
function CheckEmail(EmailAddr)
{
   var nbool;
   var strLength=EmailAddr.length;
   var index1=EmailAddr.indexOf("@");
   var index2=EmailAddr.indexOf(".",index1);
   if(index1==-1||index2==-1||index2<=index1+1||index1==0||index2==strLength-1)
      nbool=false;
   else
      nbool=true;
   return nbool;
}
//操作函数
function MyMain( )
{
   var MyName=prompt("姓名 ：(4 到 8 个字符)");
   var nName=CheckName(MyName);
   while(nName==false)
   {
      var NameMsg="姓名字段验证 ：\n\n";
      NameMsg+="结果 ： 格式错误.\n";
      NameMsg+="格式 ： 长度必须为 4 到 8 个字符.\n";
      NameMsg+="处理 ： 单击“确定”按钮返回修改.\n";
      alert(NameMsg);
      MyName=prompt("姓名 ：(4 到 8 个字符)");
      nName=CheckName(MyName);
   }
   var MySex=prompt("性别 ：(male 或 female)","male");
   var nSex=CheckSex(MySex);
   while(nSex==false)
   {
      var SexMsg="性别字段验证 ：\n\n";
      SexMsg+="结果 ： 格式错误.\n";
      SexMsg+="格式 ： 必须为''male''或''famale''.\n";
      SexMsg+="处理 ： 单击“确定”按钮返回修改.\n";
      alert(SexMsg);
      MySex=prompt("性别 ：(male 或 female)","male");
      nSex=CheckSex(MySex);
   }
   var MyNum=prompt("学号 ：(形如 2001****)","2001");
   var nNum=CheckNumber(MyNum);
```

```
  while(nNum==false)
  {
    var NumMsg="学号字段验证 ：\n\n";
    NumMsg+="结果 ： 格式错误.\n";
    NumMsg+="格式 ： 必须形如"2001****"格式. \n";
    NumMsg+="处理 ： 单击“确定”按钮返回修改.\n";
    alert(NumMsg);
    MyNum=prompt("学号 ：(形如 2001****)","2001");
    nNum=CheckNumber(MyNum);
  }
  var MyEmail=prompt("邮箱 ：(形如 zangpu@gmail.com)","@");
  var nEmail=CheckEmail(MyEmail);
  while(nEmail==false)
  {
    var EmailMsg="邮箱字段验证 ：\n\n";
    EmailMsg+="结果 ： 格式错误.\n";
    EmailMsg+="格式 ：1、邮件地址中同时含有'@'和'.'字符.\n";
    EmailMsg+="          2、'@'后必须有'.'，且中间至少间隔一个字符. \n";
    EmailMsg+="          3、'@'不为第一个字符，'.'不为最后一个字符.\n";
    EmailMsg+="处理 ： 单击“确定”按钮返回修改.\n";
    alert(EmailMsg);
    MyEmail=prompt("邮箱 ：","@");
    nEmail=CheckEmail(MyEmail);
  }
  var msg="\n 学生信息 ：\n\n"
  msg+="姓名 ："+MyName+"\n";
  msg+="性别 ："+MySex+"\n";
  msg+="学号 ："+MyNum+"\n";
  msg+="邮箱 ："+MyEmail+"\n";
  alert(msg);
}
//-->
</script>
</head>
<body>
<form name="MyForm">
  <input type="button" value="测试" onclick="MyMain()">
</body>
</html>
```

程序运行后，单击页面中的“测试”按钮，弹出“姓名”提示框提示学生输入姓名，如图 7.10 所示。

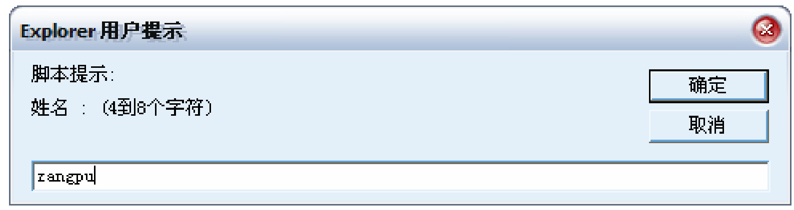

图 7.10 “姓名”提示框

当学生输入正确格式的姓名之后，弹出“性别”提示框提示学生输入性别，如图 7.11 所示。

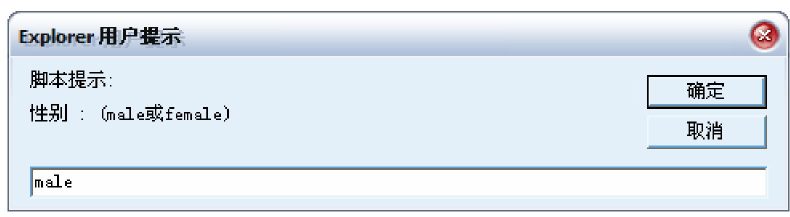

图 7.11 “性别”提示框

当学生输入正确格式的性别之后，弹出“学号”提示框提示学生输入学号，如图 7.12 所示。

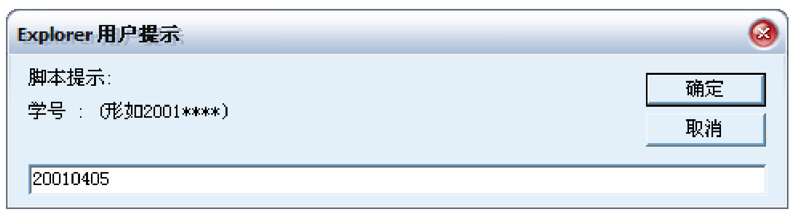

图 7.12 “学号”提示框

当学生输入正确格式的学号之后，弹出“邮箱”提示框提示学生输入邮箱，如图 7.13 所示。

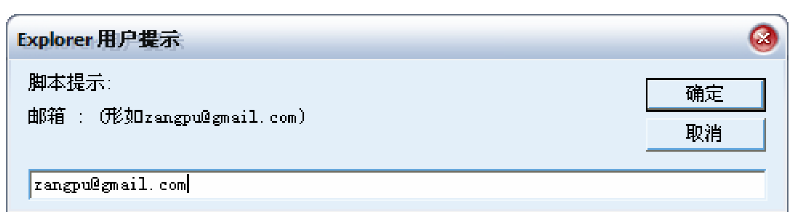

图 7.13 “邮箱”提示框

当学生完成前几项信息输入后，将弹出“学生信息”提示框，该框汇总了前面学生输入的信息，如图 7.14 所示。

图 7.14 “学生信息”提示框

上述的过程演示了如何在页面中采集客户的信息并保存下来，实际应用中，还应该加入判断客户输入信息是否符合要求的代码。本系统使用函数的形式判断，如果符合特定格式，则进入下一个收集信息步骤；若不符合该特定格式，则返回收集当前项信息的提示框继续执行下去。

如果“姓名”字段格式不满足要求，则弹出警告框提示学生输入正确的当前项信息，如图 7.15 所示。

图 7.15 “姓名”字段不正确时，弹出警告框

单击“确定”按钮后，返回“姓名”字段信息输入提示框。上述步骤反复，直到“姓名”字段正确后进入下一步。

如果“性别”字段格式不满足要求，则弹出警告框提示学生输入正确的当前项信息，如图 7.16 所示。

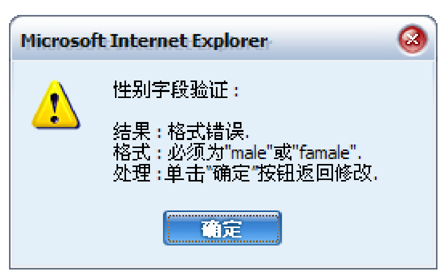

图 7.16 “性别”字段不正确时，弹出警告框

单击 i“确定”按钮后，返回“性别”字段信息输入提示框。上述步骤反复，直到“性别”字段正确后进入下一步。

如果“学号”字段格式不满足要求，则弹出警告框提示学生输入正确的当前项信息，如图 7.17 所示。

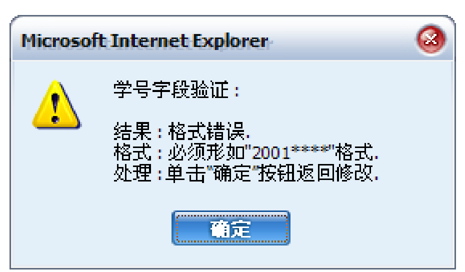

图 7.17 “学号”字段不正确时，弹出警告框

单击“确定”按钮后，返回“学号”字段信息输入提示框。上述步骤反复，直到“学号”字段正确后进入下一步。

如果“邮箱”字段格式不满足要求，则弹出警告框提示学生输入正确的当前项信息，如图 7.18 所示。

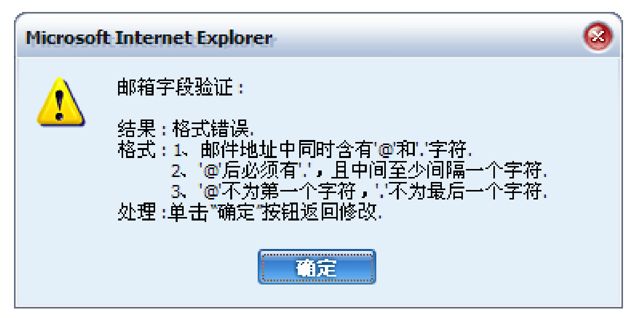

图 7.18 “邮箱”字段不正确时，返回警告框

上面的学生信息采集系统融合了采集、验证等功能，可以将其适当扩展，如将采集的数据提交给目标页面，或者直接录入数据库备用等。

Window 对象提供几种客户与页面交互，并通过对话框采集客户信息的途径。通过综合使用这几种方法，可实现页面的交互性、动态性等要求。

7.2.2 设定时间间隔

Window 对象提供 setInterval()方法用于设定时间间隔，用于按照某个指定的时间间隔去周期触发某个事件，典型的应用如动态状态栏、动态显示当前时间等，该方法的语法如下：

```
TimerID=window.setTimeout(targetProcess,itime);
TimerID=setTimeout(targetProcess,itime);
```

其中参数 targetProcess 指目标事件，参数 itime 指间隔的时间，以毫秒(ms)为单位。设定时间间隔的操作完成后，返回该时间间隔的引用变量 TimerID。

同时，Windows 对象提供 clearInterval()方法用于清除该间隔定时器使目标事件的周期触发失效，该方法语法如下：

```
window.clearInterval(TimerID);
```

该方法接受唯一的参数 TimerID，指明要清除的间隔时间引用变量名。考察如下设定和停止动态状态栏的代码：

```
//源程序 7.2
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var TimerID;
var dir=1;
var str_num=0;
//用于动态显示的目标字符串
var str="Welcome To JavaScript World!";
//设定动态显示的状态栏信息
function startStatus()
{
  var str_space="";
  str_num=str_num+1*dir;
```

```
    if(str_num>30 || str_num<0)
    {
       dir=-1*dir;
    }
    for(var i=0;i<str_num;i++)
    {
       str_space+=" ";
    }
    window.status=str_space+str;
}
//状态栏滚动开始
function MyStart()
{
   TimerID=setInterval("startStatus();",100);
}
//状态栏滚动结束，并更新状态栏
function MyStop()
{
   clearInterval(TimerID);
   window.status="The Moving Status have been stopped!";
}
//-->
</script>
</head>
<body onload="window.status='Original Status!'">
<br>
<center>
<p>单击对应的按钮，实现动态状态栏的滚动与停止!</p>
<form name="MyForm">
   <input type="button" value="开始状态栏滚动" onclick="MyStart()"><br>
   <input type="button" value="停止状态栏滚动" onclick="MyStop()"><br>
</form>
</center>
</body>
</html>
```

代码运行后，出现如图 7.19 所示的页面：

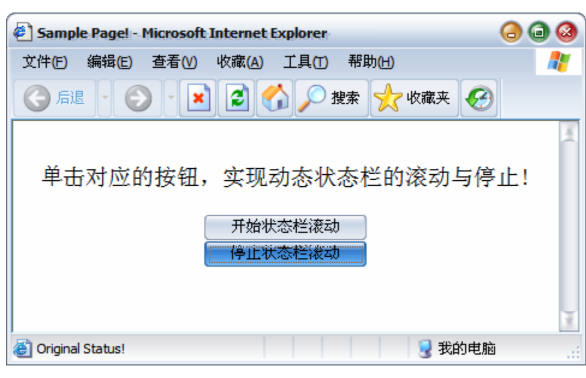

图 7.19 载入页面时出现原始界面

此时的状态栏显示为“Original Status!”，单击“开始状态栏滚动”按钮后，状态栏显示为“Welcome To JavaScript World!”，并按照 setInterval()方法设定的时间间隔左右滚动，如

图 7.20 所示。

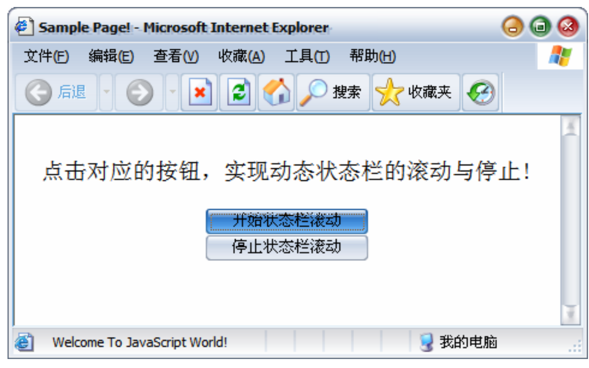

图 7.20 状态栏信息左右滚动显示

单击“停止状态栏滚动”按钮后，状态栏信息滚动显示的效果停止，状态栏显示“The Moving Status have been stopped!”，如图 7.21 所示。

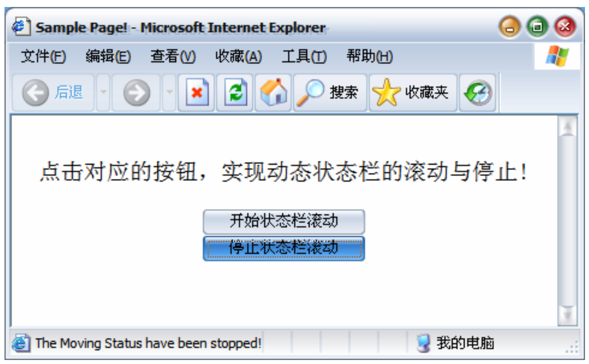

图 7.21 状态栏滚动停止

该实例演示了 Window 对象的 setInterval()和 clearInterval()方法的使用情况，如果要更改滚动的速度，只需修改 TimerID=setInterval("startStatus();",100)语句里面的间隔时间“100ms”即可实现。

7.2.3 事件超时控制

Window 对象提供 setTimeout()方法用于设置某事件的超时，即在设定的时间到来时触发某指定的事件，该方法的实际应用有警告框的显示时间和状态栏的跑马灯效果、打字效果等。其语法如下：

```
var timer=window.setTimeout(targetProcess,itime);
var timer=setTimeout(targetProcess,itime);
```

参数 targetProcess 表示设定超时的目标事件，参数 itime 表示设定的超时时间，以毫秒(ms)为单位，返回值 timer 为该事件超时的引用变量名。

同时，Window 对象提供 clearTimeout()方法来清除通过参数传入的事件超时操作。该语法如下：

```
clearTimeout(timer);
```

该方法接受唯一的参数 timer 指定要清除的事件超时引用变量名，方法执行后将该事件

超时设置为失效。考察如下演示状态栏打字效果的代码：

```
//源程序 7.3
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var seq=0;
var TimerID;
//超时时间
var interval = 120;
//设定要实现状态栏打字效果的字符串
var MyStatus = "Welcome to JavaScript World!";
var strLength= MyStatus.length;
//设定状态栏的显示内容
function MyScroll()
{
  window.status=MyStatus.substring(0, seq+1);
  seq++;
  if(seq>= strLength)
  {
    seq=0;
    window.status='';
    //设定触发事件的时间间隔
    TimerID=setTimeout("MyScroll();",interval);
  }
  else
    TimerID=setTimeout("MyScroll();",interval);
}
//停止该超时操作
function MyStopScroll()
{
  clearTimeout(TimerID);
  window.status="The Changing Status have been stopped!";
}
//-->
</script>
</head>
<body onload="window.status='Original Status!'">
<br>
<center>
<p>单击对应的按钮，实现状态栏打字效果的显示与停止!</p>
<form name="MyForm">
  <input type="button" value="开始状态栏滚动" onclick="MyScroll()"><br>
  <input type="button" value="停止状态栏滚动" onclick="MyStopScroll()"><br>
</form>
</center>
</body>
</html>
```

程序运行后，出现如图 7.19 所示页面。在此页面单击“开始状态栏滚动”按钮后，状

态栏中状态信息从空白逐字增加直到指定字符串结束，然后清空为空白并重复此过程，如图7.22 所示。

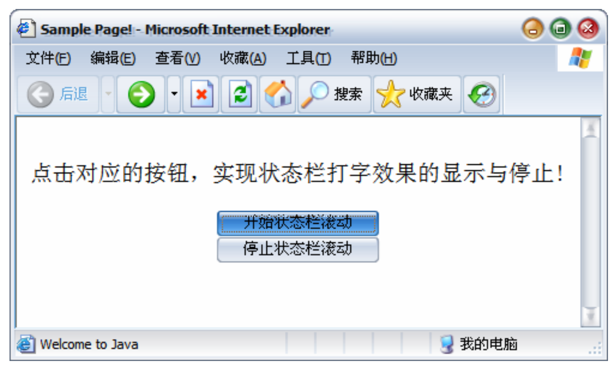

图 7.22 状态栏的打字效果

单击"停止状态栏滚动"按钮后，状态栏的打字效果停止，并显示状态信息"The Changing Status have been stopped!"，如图 7.23 所示。

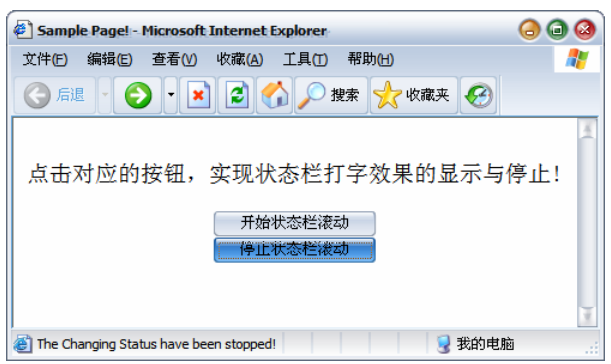

图 7.23 状态栏的打字效果失效

该实例演示了 Window 对象的 setTimeout()和 clearTimeout()方法的使用情况，如果要更改打字效果中字符出现的速度，只需修改 TimerID=setTimeout("MyScroll();",interval)语句里面的间隔时间 interval 即可实现。

7.2.4 创建和管理新窗口

Window 对象提供完整的方法用于创建新窗口并在父窗口与字窗口之间进行通信。一般来说，主要使用其 open()方法创建新浏览器窗口，新窗口可以包含已存在的 HTML 文档或者完全由该方法创建的新文档，其语法如下：

```
var newWindow=window.open(targetURL,pageName, options,repalce);
var newWindow=open(targetURL,pageName, options,repalce);
```

其中参数：

- targetURL：指定要打开的目标文档地址；
- pageName：设定该页面的引用名称；
- options：指定该窗口的属性，如页面大小、有否工具条等。

其中 options 包含一组用逗号隔开的可选属性对，用以指明该窗口所具备的各种属性，

其属性及对应的取值如表 7.1 所示。

表 7.1 options参数可包含的属性

属性	取值	简要说明
directories	yes/no	目标窗口是否具有目录按钮
height	integer	目标窗口的高度
left	interger	目标窗口与屏幕最左边的距离
location	yes/no	目标窗口是否具有地址栏
menubar	yes/no	目标窗口是否具有菜单栏
resizable	yes/no	目标窗口是否允许改变大小
scrollbars	yes/no	目标窗口是否具有滚动条
status	yes/no	目标窗口是否具有状态栏
toolbar	yes/no	目标窗口是否具有工具栏
top	integer	目标窗口与屏幕最顶端的距离
width	integer	目标窗口的宽度

注意：left、height、top、width 属性的取值为整数，为像素值。其余取值为 yes/no，分别表示目标具有或不具有某种属性。在当前浏览器版本中，可用 1 代替 yes，用 0 代替 no。

窗口建立后，可通过新窗口的 document 对象的 write()方法往该窗口写入内容，可以是纯粹的字符串，也可是 HTML 格式的字符串，后者将被浏览器解释之后再显示。

操作完成后，可通过 Window 对象的 close()方法来关闭该窗口，close()方法的语法如下：

```
windowName.close();
```

使用 close()方法关闭某窗口之前，一定要核实该窗口是否已经定义、是否已经定义，如果目标窗口未定义或已经被关闭，则 close()方法返回错误信息。

考察如下的综合实例，该实例演示了如何新建浏览器窗口及往该窗口写入内容的方法：

```
//源程序 7.4
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var ExistWindow;
var NewWindow;
var strOptions="";
//产生窗口属性字符串
function CreateStr()
{
  var CheckLength=document.MyForm.elements.length;
  for(var i=0;i<CheckLength-1;i++)
  {
    if((document.MyForm.elements[i].type=="checkbox")&&
       (document.MyForm.elements[i].checked))
      strOptions+=document.MyForm.elements[i].name+"=yes,";
  }
  strOptions+="height="+document.MyForm.height.value+",";
  strOptions+="width="+document.MyForm.width.value+",";
  strOptions+="top="+document.MyForm.top.value+",";
  strOptions+="left="+document.MyForm.left.value;
}
```

```
//新建已存在的窗口
function CreateExistWindow()
{
  strOptions="";
  CreateStr();
  var MyURL=document.MyForm.MyURL.value;
  var MyName=document.MyForm.MyName.value;
  ExistWindow=window.open(MyURL,MyName,strOptions);
  if(!window.ExistWindow)
    alert("The target window is not exist!");
}
//新建空白窗口并写入预设的字符串
function WriteNewWindow()
{
  strOptions="";
  CreateStr();
  var MyURL='';
  var MyName=document.MyForm.MyName.value;
  NewWindow=window.open(MyURL,MyName,strOptions);
  if(window.NewWindow)
  {
    NewWindow.document.write(document.MyForm.MyContents.value);
    NewWindow.focus();
  }
  else
    alert("The window to be written is not exist!");
}
//关闭当前子窗口
function CloseWindow()
{
  if(window.ExistWindow)
    ExistWindow.close();
  else if(window.NewWindow)
    NewWindow.close();
  else
    alert("The window to be closed is not exist!");
}
//-->
</script>
</head>
<body>
<center>
设置新窗口属性,然后单击“确定”按钮产生新窗口!
<hr>
<form name="MyForm">
  窗口地址 ：<input type="text" name="MyURL" id="MyURL" value="temp.html"><br>
  窗口名称 ：<input type="text" name="MyName" id="MyName" value="MySamplePage"><br>
  窗口高度 ：<input type="text" name="height" id="height" maxlength=4 value=290><br>
  窗口宽度 ：<input type="text" name="width" id="width" maxlength=4 value=324><br>
  左边距离 ：<input type="text" name="top" id="top" maxlength=4 value=100><br>
  顶端距离 ：<input type="text" name="left" id="left" maxlength=4 value=100><br>
  窗口内容 ：<textarea name="MyContents" id="MyContents" rows=4 cols=19>
             Contents Write to the new blank window!</textarea><br>
```

```
    滚动条    ：<input type="checkbox" name="scrollbar" id="scrollbar"><br>
    大小改变  ：<input type="checkbox" name="resizable" id="resizable"><br>
    目录按钮  ：<input type="checkbox" name="directories" id="directories"><br>
    地址栏    ：<input type="checkbox" name="location" id="location"><br>
    菜单栏    ：<input type="checkbox" name="menubar" id="menubar"><br><hr>
    <input type="button" value="新建浏览器窗口" onclick="CreateExistWindow()">
    <input type="button" value="写内容进空白窗口" onclick="WriteNewWindow()">
    <input type="button" value="关闭当前子窗口" onclick="CloseExistWindow()">
</form>
</center>
</body>
</html>
```

程序运行后，出现如图 7.24 所示的页面。

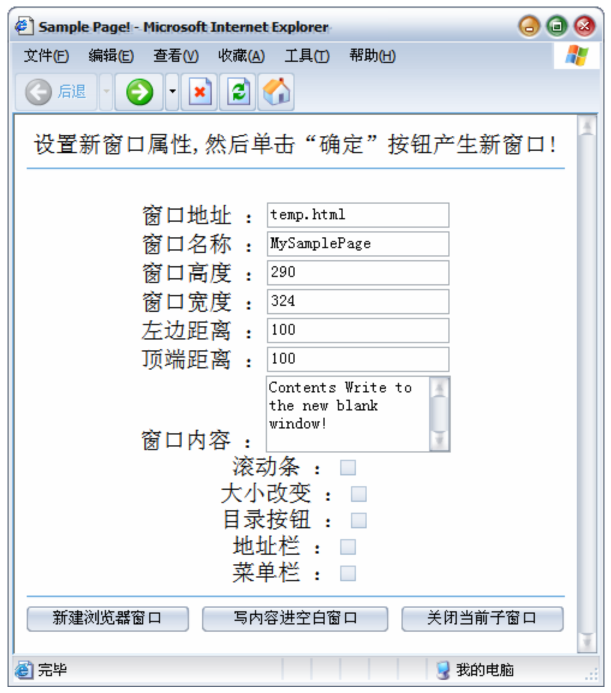

图 7.24 程序运行后的原始页面

在原始页面选中五个 checkbox 单选框，单击“新建浏览器窗口”按钮，浏览器将按照表单中设定的属性弹出当前目录下 HTML 文档“temp.html”的页面，如图 7.25 所示。

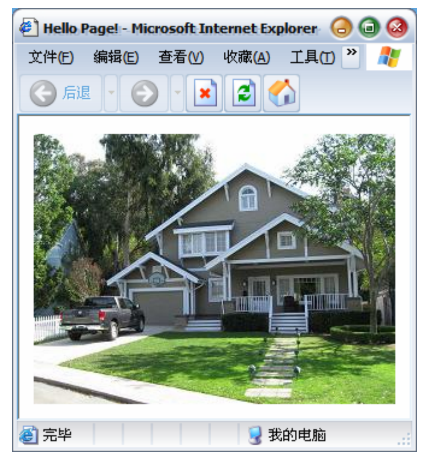

图 7.25 按照设定的属性加载已存在的 HTML 文档

在原始页面选中五个 checkbox 单选框，在原始页面单击“写内容进空白页面”按钮，浏览器将按照表单中设定的属性生成新浏览器窗口，并将文本域“MyContents”的内容“Contents Write to the new blank window!”写进该页面，如图 7.26 所示。

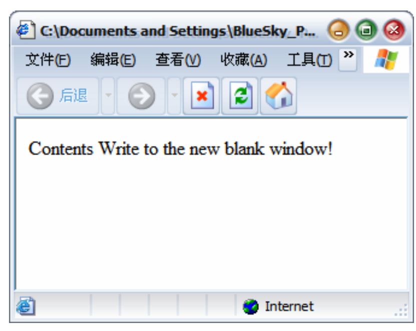

图 7.26 生成新浏览器页面并写入指定的内容

该新建的 HTML 页面拥有写入的字符串，不包含任何格式信息。查看源文件，仅包含“Contents Write to the new blank window!”字符串。

当然，往新建的 HTML 窗口动态写入 HTML 语句也是可行的。查看如下代码：

```
var embedHtml="<html><head><title>Sample Page!</title></head>";
embedHtml+="<body>Contents Write to the new blank window!</body></html>";
NewWindow.document.write(embedHtml);
```

将上述代码替换源程序 7.4 中的下列语句：

```
NewWindow.document.write(document.MyForm.MyContents.value);
```

保存后，运行该程序，单击“写内容进空白页面”按钮，弹出新页面，与源程序 7.4 一样，但查看源程序，如图 7.27 所示。

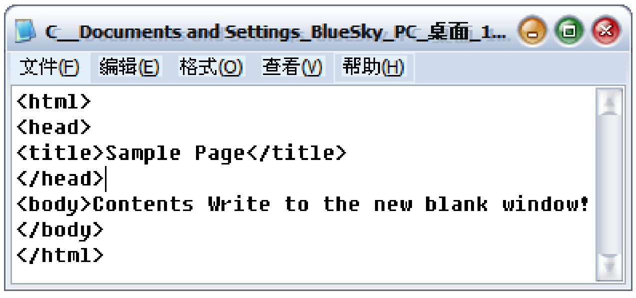

图 7.27 将 HTML 语句写入新窗口

通过上述实例，读者可以适当扩展，按照需要给新窗口适当的属性值，然后根据新窗口的引用名称动态写入 HTML 语句生成交互性较强的页面。

7.2.5 常见属性和方法汇总

Window 对象提供诸多属性和方法用于浏览器窗口操作，如获取和设置当前窗口信息、创建浏览器窗口等。但由于各大浏览器厂商在继承 DOM 标准的基础上各自扩展了 Window 对象，而且浏览器的版本对 Window 对象的支持程度也不一样。出于兼容性考虑，表 7.2 列出了 Internet Explorer（简称 IE，下同）和 Netscape Navigator（简称 NN，下同）浏览器平

台通用的 Window 对象常见属性和方法。

表 7.2 Window对象常见属性和方法汇总

类型	项目	简要说明
属性	closed	表示窗口是否已被关闭
	defaultStatus	窗口底部默认的状态栏信息
	document	窗口中当前文档对象
	frames	包含窗口中所有Frame对象的数组
	history	包含窗口历史URL清单的History对象
	location	包含与Window对象相关联的URL地址的对象
	name	当前窗口的标识
	opener	表示打开窗口的Window对象
	parent	与包含某个窗口的父窗口含义相同
	self	与当前窗口的含义相同
	status	窗口底部的状态栏信息
	top	指一组嵌套窗口的最上层浏览器窗口
方法	alert()	显示提示信息对话框
	blur()	使当前窗口失去焦点
	clearInterval(TimerID)	使由参数TimerID指定的间隔定时器失效
	clearTimeout(TimerID)	使由参数TimerID指定的超时设置失效
	close()	关闭当前窗口
	conform(text)	显示确认对话框，text为确认内容
	focus()	使当前窗口获得焦点
	moveBy(deltaX,deltaY)	将浏览器窗口移动到由参数deltaX和deltaY(像素)指定相对距离的位置
	moveTo(x,y)	将浏览器移动到由参数x和y(像素)指定的位置
	open(URL,Name,Options)	按照Options指定的属性打开新窗口并创建Window对象
	prompt(text[, str])	显示提示对话框，text为问题，str为默认答案（可选参数）
	resizeBy(deltaX,deltaY)	将浏览器窗口大小按照参数deltaX和deltaY(像素)指定的相对像素改变
	resizeTo(x,y)	将浏览器窗口的大小按照参数参数x和y(像素)指定的值进行设定
	scroll(hori,Verti)	将目标文档移动到浏览器窗口中由参数hori和Verti指定的位置（NN3+）
	scrollBy(deltaX,deltaY)	在浏览器窗口中将文档移动由deltaX和deltaY指定相对距离的位置
	scrollTo(x,y)	在浏览器窗口中将文档移动到由x和y指定的位置
	setInterval(expression, milliseconds, [arguments])	通过由参数milliseconds指定的时间间隔重复触发由参数expression指定的表达式求值或函数调用，可选参数arguments为供函数调用的参数列表，以逗号为分隔符
	setTimeout(expression, milliseconds, [arguments])	通过由参数milliseconds指定的超时时间触发由参数expression指定的表达式求值或函数调用，可选参数arguments为供函数调用的参数列表，以逗号为分隔符

Window 对象为 Web 应用开发者提供了丰富的属性和操作浏览器窗口及事件的方法，通过 Window 对象，可以访问对象关系层次中处于其下层的对象如 Navigator 对象等。下面讨论与浏览器紧密相关的 Navigator 对象。

7.3 Navigator 对象

由于浏览器及其版本对 JavaScript 脚本的支持情况不同，出于脚本代码兼容性考虑，经常需要获取客户端浏览器相关信息，以根据其浏览器具体情况编制不同的脚本代码。

Navigator 对象最初由 Netscape 浏览器引入，并在其 NN2 中获得支持。Microsoft 在其 IE3 上引入 Navigator 对象，但只支持其部分属性和方法。由于 Navigator 对象为程序员提供了十分有效的浏览器相关信息而得到较为广泛的应用，Microsoft 在其 IE4 中引入 Navigator 对象的克隆版本即 clientInformation 对象并在 IE4 后续版本中得到更为完善的支持，该对象的所有属性和方法与 Navigator 对象完全相同。不同的是，clientInformation 对象仅适用于 IE

浏览器，而 Navigator 对象则适用于所有浏览器，当然也包括 IE 浏览器。

同 Window 对象一样，Navigator 对象为浏览器对象模型中的顶级对象，而不是作为其他对象的属性而存在。相比较 Window 对象而言，Navigator 对象与浏览器及其版本的关联程度更紧密，对编写代码兼容性较强的应用程序贡献更大，但 Navigator 对象的属性多为只读，且提供的操作方法也较少。

7.3.1 获取浏览器信息

在编写跨平台 JavaScript 脚本时，须事先获取客户端浏览器的相关信息，然后作对应的操作，此方法可解决脚本代码在各浏览器中的兼容性问题。下面的代码演示了如何获取客户端浏览器的相关信息并作相应的处理：

```
//源程序 7.5
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//根据客户端浏览器信息确定脚本的走向
function GetInfo()
{
  //判断浏览器类型，若浏览器为 NN，则转向 displayNNInfo()
  if(navigator.appName=="Netscape")
    displayNNInfo();
  //若浏览器为 IE,则转向 displayIEInfo()
  else if(navigator.appName=="Microsoft Internet Explorer")
    displayIEInfo();
  //否则,输出警告信息
  else
    alert("对不起,当前浏览器类型不支持!");
}
//如果客户端浏览器为 NN,则触发此函数
function displayNNInfo()
{
  var msg="\nNN 浏览器信息 ：\n\n"+"检测结果 ：\n\n"+"通用属性 ：\n";
  msg+="----appCodeName : "+navigator.appCodeName+"\n";
  msg+="----appName : "+navigator.appName+"\n";
  msg+="----appVersion : "+navigator.appVersion+"\n";
  msg+="----cookieEnabled : "+navigator.cookieEnabled+"\n";
  msg+="----mimeTypes.length : "+navigator.mimeTypes.length+"\n";
  msg+="----plugins.length : "+navigator.plugins.length+"\n";
  msg+="----platform : "+navigator.platform+"\n";
  mag+="----userAgent : "+navigator.userAgent+"\n";
  msg+="扩展属性 ：\n";
  msg+="----language : "+navigator.language+"\n";
  alert(msg);
}
//如果客户端浏览器为 IE,则触发此函数
```

```
function displayIEInfo()
{
   var msg="\nIE 浏览器信息 ：\n\n"+"检测结果 ：\n\n"+"通用属性 ：\n";
   msg+="----appCodeName : "+navigator.appCodeName+"\n";
   msg+="----appName : "+navigator.appName+"\n";
   msg+="----appVersion : "+navigator.appVersion+"\n";
   msg+="----cookieEnabled : "+navigator.cookieEnabled+"\n";
   msg+="----mimeTypes.length : "+navigator.mimeTypes.length+"\n";
   msg+="----plugins.length : "+navigator.plugins.length+"\n";
   msg+="----platform : "+navigator.platform+"\n";
   msg+="----userAgent : "+navigator.userAgent+"\n";
   msg+="扩展属性 ：\n";
   msg+="----appMinorVersion : "+navigator.appMinorVersion+"\n";
   msg+="----cpuClass : "+navigator.cpuClass+"\n";
   msg+="----language : "+navigator.language+"\n";
   msg+="----browserLanguage : "+navigator.browserLanguage+"\n";
   msg+="----userLanguage : "+navigator.userLanguage+"\n";
   msg+="----systemLanguage : "+navigator.systemLanguage+"\n";
   msg+="----onLine : "+navigator.onLine+"\n";
   msg+="----userProfile : "+navigator.userProfile+"\n";
   alert(msg);
}
//-->
</script>
</head>
<body>
<hr>
<center>
<form name="MyForm">
   <input type="button" value="测试" onclick="GetInfo()">
</form>
</center>
</body>
</html>
```

程序运行后，单击页面中的“测试”按钮，如果客户端浏览器为 IE4+，则弹出警告框如图 7.28 所示。

图 7.28 当客户端浏览器为 IE4+时弹出警告框

如果客户端浏览器为 NN2+，则弹出警告框如图 7.29 所示。

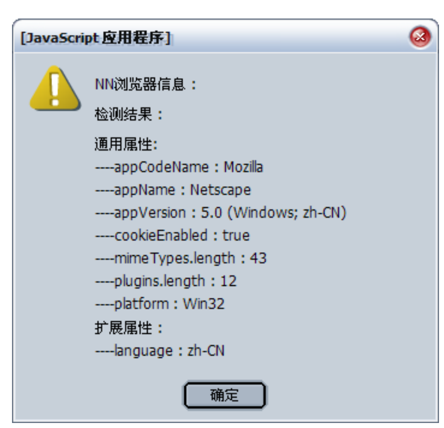

图 7.29 当客户端浏览器为 NN2+时弹出提示框

浏览器载入页面后，客户单击“测试”按钮，触发 GetInfo()函数，该函数判断当前浏览器的类型：若其 appName 属性为“Netscape”，则转向 displayNNInfo()函数输出当前浏览器相关信息；若其 appName 属性为“Microsoft Internet Explorer”，则转向 displayIEInfo()函数输出当前浏览器相关信息。

注意：上述程序的调试平台为 IE6 和 NN7，较低浏览器版本可能不支持某些属性，有关 Navigator 对象属性的版本支持将在后面列表详述。

7.3.2 常见方法和属性汇总

Navigator 对象拥有的属性和方法随浏览器版本的更新而不断增加，总的来说，除了早期的 Navogator 对象的基本属性和方法之外，大多数新增的属性和方法都与浏览器版本相关。表 7.3 列出了 Navigator 常见的属性、方法及浏览器支持情况。

表 7.3 Navigator常见的属性和方法

类型	项目	简要说明	浏览器支持
属性	appCodeName	返回包含每个浏览器的客户类，代表浏览器代码号	NN2+、IE3+
	appName	返回浏览器官方名称，如IE的Microsoft Internet Explorer等	NN2+、IE3+
	appVersion	返回浏览器的版本号	NN2+、IE3+
	cookieEnabled	标记浏览器的cookie功能是否已开启	NN6+、IE4+
	MimeTypes	保存MIME类型信息的数组	NN4+、IE4+
	Plugins	保存网页中插件程序的数组	NN3+、IE4+
	platform	保存操作系统的类型	NN4+、IE4+
	userAgent	保存从客户端向服务器发送的HTTP协议用户代理头的值	NN2+、IE3+
	appMinorVersion	返回浏览器的次版本号	IE4+
	cpuClass	返回运行浏览器的中央处理器种类	IE4+
	browserLanguage	返回程序本地语言版本的标识符	IE4+
	userLanguage	返回当前操作系统使用的自然语言	IE4+
	systemLanguage	返回操作系统默认使用的语言	IE4+
	online	标记浏览器确定脱机浏览的状态，联机状态下该属性为true	IE4+
	userProfile	返回用户的档案设置	IE4+
	language	返回浏览器应用程序的语言代码	NN4+
方法	javaEnabled()	返回浏览器是否禁止Java的标志位	NN3+，IE4+
	taintEnabled()	返回浏览器支持数据感染安全特性的标志位	NN3+、IE4+
	preference()	允许标识的脚本获取并设置某些 Navigator 的首选项信息	NN4+

Navigator 对象的属性和方法在实际调用过程中，除了理解其基本含义外，还需了解以下几项内容：

- appName 属性返回浏览器应用程序的官方名称，IE 浏览器的官方名称为“Microsoft Internet Explorer”，NN 浏览器的官方名称为“Netscape”，而有些浏览器则通过 Navigator 对象的扩展方法来检测其官方名称，如 Opera 浏览器的 isOpera()、Safari 浏览器的 isSafari()等。
- appVersion 属性返回当前浏览器的版本号，一般情况下可通过 parseInt()和 parseFloat()方法提取其中的数值再进行相关比较，但此数值更多的是表现浏览器版本的继承特性而不是真正的版本号，如 IE6 的 appVersion 属性在提取数值后，返回 4 而不是 6。
- platform 属性返回操作系统的类型。Win32 代表 Window XP、Win98 代表 Windows 98、WinNT 代表 Windows NT、Win16 代表 Window3.x、Mac68k 代表 Mac(680x0 CPU)、MscPPC 代表 Mac(PowerPC CPU)、SunOS 代表 Saloris 等。
- plugins 属性在 IE4+上获得支持，但返回一个空数组，表示不包含任何 IE 中不存在的对象。

Navigator 对象从本质上说是在顶级对象模型中与浏览器类型及其版本紧密相关的顶级对象，其属性和方法随着浏览器类型、版本及系统设置的完成而确定下来，多为只读的属性和方法。Navigator 对象又包括两个作为其属性的对象，分别为 mimeType 对象和 plugin 对象，该部分内容在后续章节详细叙述。下面介绍与浏览器物理相关的 Screen 对象。

7.4 Screen 对象

Screen 对象最初由 NN4 引入，该对象提供了客户端用户屏幕的相关信息，如屏幕尺寸、颜色深度等。如同 Navogator 对象由 NN2 引入后 Micorsoft 在其 IE3 中引入 Navigator 对象的克隆版本 clientInformation 对象一样，在 NN4 中 Screen 对象引入后，Microsoft 在其 IE4 中定义了新的 Screen 对象，其属性和方法与 NN4 中定义的完全相同，不同点在于 IE4 中的 Screen 对象作为 Window 对象的属性而存在，而 NN4 中 Screen 对象和 Window 对象同为顶级对象模型的成员。

Screen 对象的属性可用 screen.property 的方式调用，在 IE4+中还可以使用如下的语法：

```
[window.]navigator.screen.property
```

Screen 对象基本的属性包括 height、width 和 colorDepth 等，但各浏览器厂商都对其进行了一定的扩展，如 NN4 扩展了 availLeft 和 availTop、pixelDepth 等属性，用于返回屏幕可用区域的初始像素位置坐标（像素）；IE4 扩展了 bufferDepth 属性，用于打开 offscreen 缓冲并控制缓冲的颜色深度等。除此之外，NN4 和 IE4 共同扩展了 availHeight 和 availWidth 等属性，前两个表示客户端屏幕的可用尺寸（像素），而后者返回客户端的“显示”控制面板中设置的颜色位数。

7.4.1 获取客户端屏幕信息

在 Web 应用程序中，为某种特殊目的如固定文档窗口相对于屏幕尺寸的比例、根据显示器的颜色位数选择需要加载的目标图片等都需要首先获得屏幕的相关信息。Screen 对象提供了 height 和 width 属性用于获取客户屏幕的高度和宽度信息，如分辨率为 1024*768 的显示器，调用这两个属性后分别返回 1024 和 768。但并不是所有的屏幕区域都可以用来显示

文档窗口，如任务栏等都占有一定的区域。为此，Screen 对象提供了 availHeight 和 availWidth 属性来返回客户端屏幕的可用显示区域。一般来说，Windows 操作系统的任务栏默认在屏幕的底部，也可以被拖动到屏幕的两侧或者顶部。假定屏幕的分辨率为 1024*768，当任务栏在屏幕的底部或者顶部时，其占据的屏幕区域大小为 1024*30（像素）；当任务栏被拖动到屏幕两侧时，其占据的屏幕区域大小为 60*768。

考察如下针对不同浏览器平台获取客户端屏幕相关信息的代码：

```
//源程序 7.6
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//根据客户端浏览器信息确定脚本的走向
function CheckOS()
{
  //判断浏览器类型
  //若浏览器为 NN,则转向 displayNNInfo()
  if(navigator.appName=="Netscape")
    GetWindowNN();
  //若浏览器为 IE,则转向 displayIEInfo()
  else if(navigator.appName=="Microsoft Internet Explorer")
    GetWindowIE();
  //否则,输出警告信息
  else
    alert("对不起,当前浏览器类型不支持!");
}
//如果客户端浏览器为 NN,则触发此函数
function GetWindowNN()
{
  var msg="\n 屏幕信息(NN4+) : \n\n";
  msg+="通用属性 : \n";
  msg+="----availHeight : "+screen.availHeight+"\n";
  msg+="----availWidth : "+screen.availWidth+"\n";
  msg+="----Height : "+screen.height+"\n";
  msg+="----Width : "+screen.width+"\n";
  msg+="----colorDepth : "+screen.colorDepth+"\n\n";
  msg+="扩展属性 : \n";
  msg+="----availLeft : "+screen.availLeft+"\n";
  msg+="----availTop : "+screen.availTop+"\n";
  msg+="----pixelDepth : "+screen.pixelDepth+"\n";
  alert(msg);
}
//如果客户端浏览器为 IE,则触发此函数
function GetWindowIE()
{
  var msg="\n 屏幕信息(IE4+) : \n\n";
  msg+="通用属性 : \n";
  msg+="----availHeight : "+screen.availHeight+"          \n";
  msg+="----availWidth : "+screen.availWidth+"\n";
```

```
    msg+="----Height : "+screen.height+"\n";
    msg+="----Width : "+screen.width+"\n";
    msg+="----colorDepth : "+screen.colorDepth+"\n\n";
    msg+="扩展属性 : \n";
    msg+="----bufferDepth : "+screen.bufferDepth+"\n";
    alert(msg);
}
//-->
</script>
</head>
<body>
<hr>
<center>
<form name="MyForm">
    <input type="button" value="测试" onclick="CheckOS()">
</form>
</center>
</body>
</html>
```

保存文档，若使用 IE 浏览器打开，当任务栏在屏幕底端或顶端时，在目标页面上鼠标鼠标单击“测试”按钮，将弹出警告框如图 7.30 所示。

图 7.30 IE 中当任务栏在底端或顶部时的屏幕信息

当任务栏在屏幕两侧时，在目标页面上鼠标单击“测试”按钮，将弹出警告框如图 7.31 所示。

图 7.31 IE 中当任务栏在两侧时的屏幕信息

若使用 NN 浏览器打开，无论任务栏在屏幕的哪个位置，在目标页面上鼠标单击“测试”按钮，将弹出警告框如图 7.32 所示。

图 7.32 NN 中获取屏幕的相关信息

可以发现，不管任务栏在何种位置，其 availHeight 和 availWidth 属性都返回默认状态栏在底部时可用的区域高度和宽度。

获取客户端屏幕的相关信息后，就可以通过 JavaScript 脚本来进行感兴趣的操作，如窗口的尺寸、位置及文档中图片的大小等。典型应用如全屏页面、保持图片与文档页面大小比例等。

7.4.2 定位窗口到指定位置

通过 Screen 对象的属性获得屏幕的相关信息后，结合 Window 对象有关窗口移动、更改尺寸的属性，可准确定位目标窗口。实际应用中如跟随鼠标移动的窗口、拥有固定位置的窗口等。考察如下控制窗口位置的代码：

```
//源程序 7.7
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//设定窗口水平和垂直位置的参数(像素)
var ix;
var iy;
//设定窗口宽度和高度的参数(像素)
var iwidth;
var iheight;
//设定窗口移动的方向和速度的参数
```

```
var xDirection=7;
var yDirection=4;
//设定与窗口移动相关联的间隔时间计时器
var TimerID;
//初始化窗口大小和位置
function InitWindow()
{
  ix=200;
  iy=200;
  iwidth=480;
  iheight=320;
  window.moveTo(ix,iy);
  window.resizeTo(iwidth,iheight);
}
//将浏览器窗口居中放置
function CenterWindow()
{
  var ix=(screen.width-iwidth)/2;
  var iy=(screen.height-iheight)/2;
  window.moveTo(ix,iy);
}
//设定间隔时间计时器
function StartMove()
{
  //此处的 100 代表改变的速度,即 100ms 改变一次位置
  TimerID=setInterval("MoveWindow();",100);
}
//控制页面移动
function MoveWindow()
{
  //如果目标页面水平方向在屏幕之内,则不改变水平移动方向
  if((ix+iwidth<screen.width)&&(ix>0))
  {
    ix=ix+xDirection;
  }
  //如果目标页面水平方向在屏幕之外,则改变水平移动方向
  if((ix+iwidth>=screen.width)||(ix<=0))
  {
    xDirection=-xDirection;
    ix=ix+xDirection;
  }
  //如果目标页面垂直方向在屏幕之内,则不改变垂直移动方向
  if((iy+iheight<screen.height-30)&&(iy>0))
  {
    iy=iy+yDirection;
  }
  //如果目标页面垂直方向在屏幕之外,则改变垂直移动方向
  if((iy+iheight>=screen.height-30)||(iy<=0))
  {
    yDirection=-yDirection;
    iy=iy+yDirection;
  }
  //移动窗口至指定的位置
```

```
   window.moveTo(ix,iy);
}
//停止窗口的滚动,回到原始位置
function StopMove()
{
   clearInterval(TimerID);
   CenterWindow();
}
//全屏化浏览器窗口
function FullWindow()
{
   //判断浏览器类型
   //若浏览器为 NN
   if(navigator.appName=="Netscape")
   {
      ix=screen.availLeft;
      iy=screen.availTop;
      iwidth=screen.availWidth;
      iheight=screen.availHeight;
      window.moveTo(ix,iy);
      window.resizeTo(iwidth,iheight);
   }
   //若浏览器为 IE
   else if(navigator.appName=="Microsoft Internet Explorer")
   {
      ix=0;
      iy=0;
      iwidth=screen.Width;
      iheight=screen.Height;
      window.moveTo(ix,iy);
      window.resizeTo(iwidth,iheight);
   }
   //否则,输出警告信息
   else
      alert("对不起,当前浏览器类型不支持!");
}
//-->
</script>
</head>
<body>
<center>
<P>顺序单击如下按钮,实现窗口准确定位<p>
<form name="MyForm">
   <input type="button" value="初始化浏览器窗口" onclick="InitWindow()"><br>
   <input type="button" value="将浏览器窗口居中" onclick="CenterWindow()"><br>
   <input type="button" value="开始目标窗口移动" onclick="StartMove()"><br>
   <input type="button" value="停止目标窗口移动" onclick="StopMove()"><br>
   <input type="button" value="全屏化浏览器窗口" onclick="FullWindow()"><br>
</form>
</center>
</body>
</html>
```

程序运行后，原始页面如图 7.33 所示。

图 7.33 使用 Screen 对象的属性控制窗口位置

程序主要分为如下几个步骤：

- 在页面中单击“初始化浏览器窗口”按钮，按照 InitWindow()函数设定的参数值初始化目标窗口。通过 Window 对象的 moveTo()方法将窗口移动到(200,200)位置，并通过其 resizeTo()方法改变目标窗口大小为 480*320，单位均为像素值；
- 单击“将浏览器窗口居中”按钮，JavaScript 脚本通过 Screen 对象的 width 和 height 属性及窗口的宽度和高度计算窗口居中时其左上顶点的坐标，通过 Window 对象的 moveTo()方法将目标窗口居中；
- 单击“开始目标窗口移动”按钮，触发 StartMove()函数启动间隔时间计时器，该计时器连接到函数 MoveWindow()，后者计算目标窗口的当前位置设定移动的方向和下一个位置，并通过 Window 对象的 moveTo()方法更新目标窗口的位置。此过程在间隔时间计时器的控制下循环进行，实现窗口的移动；
- 单击“停止目标窗口移动”按钮，取消控制窗口移动的间隔时间计时器，并通过 CenterWindow()函数将窗口居中放置；
- 单击“全屏化浏览器窗口”按钮，触发 FullWindow()函数，后者通过检查客户端浏览器的类型，并调用其支持的属性，通过 Window 对象的 moveTo()和 resizeTo()方法将目标窗口最大化。

上述代码演示了如何精确控制指定窗口的位置，在窗口移动过程中，可通过缩短间隔计时器时间并减小像素的每次偏移 xDirection 和 yDirection 的初始值来达到使目标窗口移动流畅的目的。

7.4.3 常见属性和方法汇总

Screen 对象提供较少但非常有效的属性用于获取客户端屏幕的相关信息并据此作出相应的动作。表 7.4 列出了 Screen 对象常见的属性及浏览器支持情况。

表 7.4 Screen对象常见属性和方法汇总

属性	简要说明	浏览器支持
availHeight	返回客户端屏幕分辨率中可用的高度（像素）	NN4+、IE4+
availWIdth	返回客户端屏幕分辨率中可用的宽度（像素）	NN4+、IE4+
height	返回客户端屏幕分辨率中的高度（像素）	NN4+、IE4+
width	返回客户端屏幕分辨率中的宽度（像素）	NN4+、IE4+
colorDepth	返回客户端“显示”控制面板中设置的颜色位数	NN4+、IE4+
availLeft	返回客户端屏幕的可用区域最左边初始像素位置（像素）	NN4+
availTop	返回客户端屏幕的可用区域最顶端初始像素位置（像素）	NN4+
pixelDepth	返回客户端“显示”控制面板中设置的颜色位数	NN4+

bufferDepth	返回标记offscreen缓冲是否打开和缓冲的颜色深度，默认值为0	IE4+

Screen 对象保存了客户端屏幕的相关信息，与文档本身相关程度较弱。下面介绍在顶级对象模型中与浏览器浏览网页后保存已访问页面和所在位置相关信息的 History 对象和 Location 对象。

7.5 History 对象

在顶级对象模型中，History 对象处于 Window 对象的下一个层次，主要用于跟踪浏览器最近访问的历史 URL 地址列表，但除了 NN4+中使用签名脚本并得到用户许可的情况之外，该历史 URL 地址列表并不能由 JavaScript 脚本显示读出，而只能通过调用 History 对象的方法模仿浏览器的动作来实现访问页面之间的漫游。

7.5.1 使用 back()和 forward()方法进行站点导航

History 对象提供 back()、forward()和 go()方法来实现站点页面的导航。back()和 forward()方法实现的功能分别与浏览器工具栏中“后退”和“前进”导航按钮相同，而 go()方法则可接受合法参数，并将浏览器定位到由参数指定的历史页面。这三种方法触发脚本检测浏览器的历史 URL 地址记录，然后将浏览器定位到目标页面，整个过程与文档无关，但在 IE 和 NN 浏览器中这两种方法又存在着不同点。

在 IE3+中，back()和 forward()方法模拟工具栏的物理行为，并不局限于框架集中特定的框架。如果要确保框架集中特定框架的跨浏览器的行为，而不是跨浏览器版本的行为，则须将 history.back()和 history.forward()方法的引用传递给父窗口。

在 NN4 中，History 对象的 back()和 forward()方法遵循其基本功能，同时又变成一对 Window 对象的方法 window.back()和 window.forward()。除此之外，History 对象不局限于框架集中使用，当框架集中使用 parent.frameName.history.forward()到达框架历史 URL 地址记录的尾部时，此方法无效。

站点导航是 back()和 forward()方法应用最为广泛的场合，可以想象在没有工具栏或菜单栏的页面（如用户注册进程中间页面等）中设置导航按钮的必要性。考察站点页面导航的实例，先看简单的框架集文档“index.html”的代码：

```
//源程序 7.8
<html>
<head>
  <title>Sample Page!</title>
</head>
<frameset cols="50%,50%">
  <frame name="MainBoard" src="left_main.html">
  <frame name="Display" src="01.html">
</frameset>
</html>
```

其中 HTML 文档“01.html”与下面将要引用的“02.html”、“03.html”、“04.html”文档类似，只列出“01.html”文档的代码：

```
//源程序 7.9
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
```

```
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>测试文档 01</title>
</head>
<body>
<center>
<p>测试文档 01</p>
</center>
<p>测试文档具体内容</p>
</body>
</html>
```

页面中左框架文档“left_main.html”的代码如下：

```
//源程序 7.10
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body>
<center>
<p><b>控制框架页面</b></p>
<p>
  <a href="01.html" target="Display">演示文档 01</a><br>
  <a href="02.html" target="Display">演示文档 02</a><br>
  <a href="03.html" target="Display">演示文档 03</a><br>
  <a href="04.html" target="Display">演示文档 04</a><br>
</p>
<form name="MyForm">
  <p><b>NN4+独有方法 ：</b></p>
  window.back() : <input type=button name="MyBack" value="返回"
                  onclick="window.back()"><br>
  window.forward() : <input type=button name="MyForward" value="前进"
                  onclick="window.forward()"><br>
  <p><b>NN4+和 IE3+公共方法 ：</b></p>
  parent.Display.history.back() : <input type=button name="MyBack2" value="返回"
                  onclick="parent.Display.history.back()"><br>
  parent.Display.history.forward() : <input type=button name="MyForward2" value="前进"
                  onclick="parent.Display.history.forward()"><br>
  <p><b>访问当前框架历史记录 ：</b></p>
  history.back() : <input type=button name="MyBack3" value="返回"
                  onclick="history.back()"><br>
  history.forward() : <input type=button name="MyForward3" value="前进"
                  onclick="history.forward()"><br>
  <p><b>访问父页面历史记录 ：</b></p>
  parent.history.back() : <input type=button name="MyBack4" value="返回"
                  onclick="parent.history.back()"><br>
  parent.history.forward() : <input type=button name="MyForward4" value="前进"
                  onclick="parent.history.forward()"><br>
</form>
</center>
</body>
```

```
</html>
```

保存后运行“index.html”文档，显示如图 7.34 所示的页面。

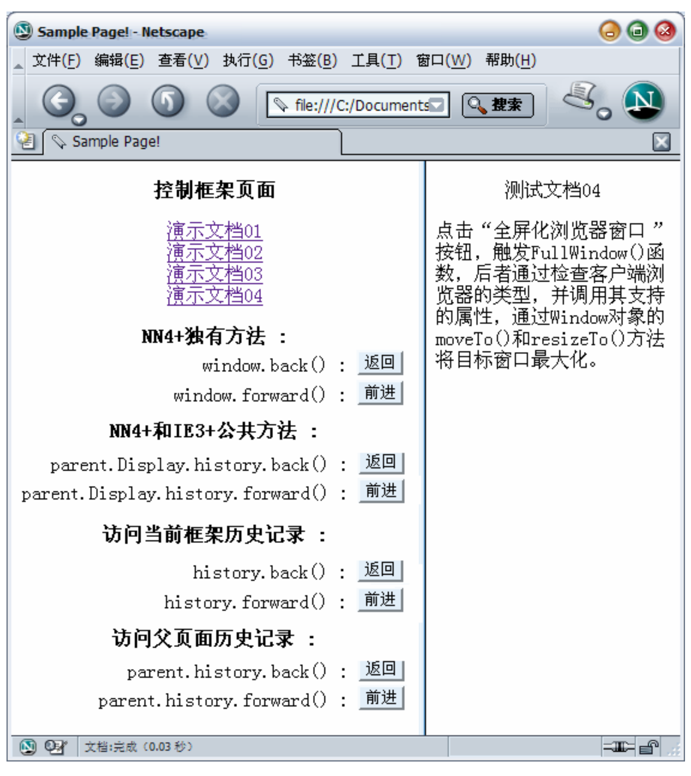

图 7.34 使用 back()和 forward()方法进行站点页面导航

除非要在导航到另外一地址之前需要进行一些附加操作，否则只需将 back()和 forward()方法作为参数传递给按钮定义的事件处理属性即可：

```
<input type=button name="MyButton" value="Forward" onclick="history.forward()">
```

出于兼容性考虑，一般通过框架对象的 parent 属性回溯到父文档中调用 back()和 forward()方法进行导航。

值得注意的是，History 对象的 back()和 forward()方法只能通过目标窗口或框架的历史 URL 地址记录列表分别向后和向前延伸，两者互为平衡。这两种方法有个显著的缺点，就是只能实现历史 URL 地址列表的顺序访问，而不能实现有选择的访问。为此，History 对象引入了 go()方法实现历史 URL 地址列表的随机访问。

7.5.2 使用 go()方法进行站点导航

History 对象提供另外一种站点导航的方法即 history.go(index|URLString)，该方法可接受两种形式的参数：

- 参数 index 传入导航目标页面与当前页面之间的相对位置，正整数值表示向前，负整数值表示向后。
- 参数 URLString 表示历史 URL 列表中目标页面的 URL，要使 history.go(URLString)方法有效，则 URLString 必须存在于历史 URL 列表中。

更改上节中的“left_main.html”文档，使用 history.go()方法进行站点页面导航：

```
//源程序 7.11
```

```
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function NavByNum()
{
  var num=document.MyForm.MyOffset.value;
  if(num>-10&&num<10)
    window.history.go(num);
  else
    alert("The input data is error!");
}
function NavByURL()
{
  var str=document.MyForm.MyURL.value;
  if(str)
    window.history.go(str);
  else
    alert("The input data is error!");
}
//-->
</script>
</head>
<body>
<center>
<p><b>控制框架页面</b></p>
<p>
  <a href="01.html" target="Display">演示文档 01</a><br>
  <a href="02.html" target="Display">演示文档 02</a><br>
  <a href="03.html" target="Display">演示文档 03</a><br>
  <a href="04.html" target="Display">演示文档 04</a><br>
  <a href="05.html" target="Display">演示文档 05</a><br>
  <a href="06.html" target="Display">演示文档 06</a><br>
  <a href="07.html" target="Display">演示文档 07</a><br>
  <a href="08.html" target="Display">演示文档 08</a><br>
  <a href="09.html" target="Display">演示文档 09</a><br>
  <a href="10.html" target="Display">演示文档 10</a><br>
</p>
<form name="MyForm">
  目标页面偏移：<input type="text" name="MyOffset" id="MyOffset" value=1><br>
  <input type="button" name="MyButton1" value="使用偏移量导航"
  onclick="NavByNum()"><br>
  历史页面 URL：<input type="text" name="MyURL" id="MyURL"><br>
  <input type="button" name="MyButton2" value="使用历史页面 URL 导航"
  onclick="NavByURL()"><br>
</form>
</center>
</body>
</html>
```

该程序将文本框数值或URL地址作为参数调用History对象的go()方法实现站点页面的导航，其中“01.html”、“02.html”、…、“10.html”与源程序 7.9 中“01.html”文档结构类似，内容基本相同。主框架集文档“index.html”与源程序7.8相同。

运行“index.html”文档，显示如图7.35所示页面。

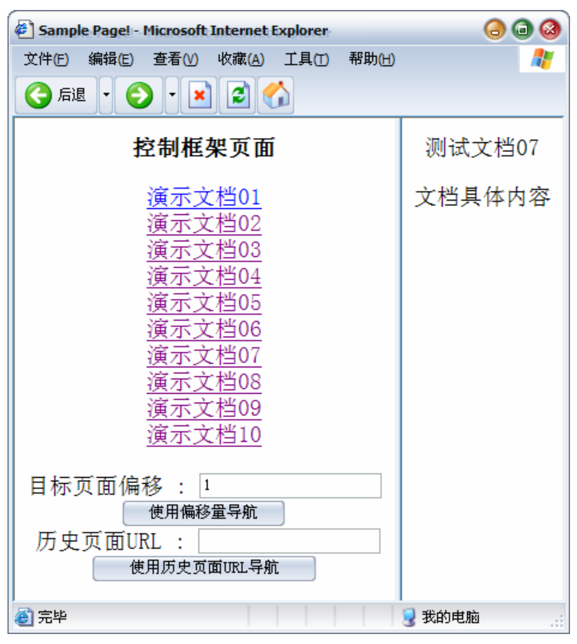

图7.35 使用go()方法进行站点页面导航

History 对象的 go()方法可传入参数 0 并设置合适的间隔时间计时器来实现文档页面重载。同时，history.go(-1)等同于 history.back()，history.go(1)等同于 history.forward()。

值得注意的是，go()方法在 IE 浏览器较老版本中获得的支持欠佳，具体表现在如下几个方面：

- IE3 中在 go()方法传入非 0 值，其结果相当于传入参数-1，即返回上一页面（如果存在的话）。同时，go()方法不接受字符串类型的历史 URL 地址；
- IE4 中，匹配字符串必须是 URL 的一部分，而不是文档标题的一部分。同时，传入参数 0 进行页面重载时，浏览器直接请求服务器返回重载页面，而不是从文档缓存中重载。

实际应用中，由于历史 URL 地址列表对用户而言一般为不可见的，所以其相对位置不确定，很难使用除-1、1 和 0 之外的参数调用 go()方法进行准确的站点页面导航。

7.5.3 常见属性和方法汇总

History对象在处理历史URL地址列表并进行站点页面导航方面有着广泛的应用，表7.5列出了其常见的属性、方法以及浏览器支持情况。

表 7.5 History对象常见属性和方法汇总

类型	项目	简要说明	浏览器支持
属性	length	保存历史URL地址列表的长度信息	NN2+、IE3+
	current	在具有签名脚本的网页中指向历史URL列表的当前项	NN4+
	next	在具有签名脚本的网页中指向历史URL列表当前项的前一项	NN4+

	previous	在具有签名脚本的网页中指向历史URL列表当前项的下一项	NN4+
方法	back()	浏览器载入历史URL地址列表的当前URL的前一个URL	NN2+、IE3+
	forward()	浏览器载入历史URL地址列表的当前URL的下一个URL	NN2+、IE3+
	go(num|str)	浏览器载入历史URL列表中由参数num指定相对位置的URL地址对应的页面、或由参数str指定其URL地址对应的页面。	NN2+、IE3+

总的来说，JavaScript 脚本很难使用 History 对象提供的方法来管理历史 URL 地址列表或者明确当前 URL 在此列表中的相对位置，使得脚本在站点页面导航时精确定位相当困难，如每个方向能导航多远、指定偏移相对位置将跳转到哪个 URL 等。

理解了保存浏览器访问历史 URL 地址信息的 History 对象，下面介绍与浏览器当前文档 URL 信息相关的 Location 对象。

7.6 Location 对象

Location 对象在顶级对象模型中处于 Window 对象的下一个层次，用于保存浏览器当前打开的窗口或框架的 URL 信息。如果窗口含有框架集，则浏览器的 Location 对象保存其父窗口的 URL 信息，同时每个框架都有与之相关联的 URL 信息。在深入了解 Location 对象之前，先简单介绍 URL 的概念。

7.6.1 统一资源定位器（URL）

URL（Uniform Resource Locator：统一资源定位器，以下简称 URL）是 Internet 上用来描述信息资源的字符串，主要用在各种 WWW 客户程序和服务器程序上。采用 URL 可以用一种统一的格式来描述各种信息资源，包括文件、服务器地址和目录等。

URL 常见格式如下：

```
protocol://hostname[:port]/[path][?search][#hash]
```

参数的意义如下：

- protocol：指访问 Internet 资源和服务的网络协议。常见的协议有 Http、Ftp、File、Telnet、Gopher 等；
- hostname：指要访问的资源和服务所在的主机对应的域名，由 DNS 负责解析。例如 www.baidu.com、www.lenovo.com 等；
- port：指网络协议所使用的 TCP 端口号，此参数可选，并且在服务器端可自由设置。如 Http 协议常使用 80 端口等；
- path：指要访问的资源和服务相对于主机的路径，此参数可选。假设目标页面“query.cgi”相对于主机 hostname 的位置为/MyWeb/htdocs/，访问该页面的网络协议为 Http，则通过 http://hostname/MyWeb/htdocs/query.cgi 访问；
- search：指 URL 中传递的查询字符串，该字符串通过环境变量 QUERY_STRING 传递给 CGI 程序，并使用问号（?）与 CGI 程序相连，若有多项查询目标，则使用加号（+）连接，此参数可选。例如要在“query.cgi”中查询 name、number 和 code 信息，可通过语句 http://hostname/MyWeb/htdocs/query.cgi?name+number+code 实现；
- hash：表示指定的文件偏移量，包括散列号（#）和该文件偏移量相关的位置点名称，此参数可选。例如要创建与位置点“MyPart”相关联的文件部分的链接，可在链接的 URL 后添加“#MyPart”。

URL 是 Location 对象与目标文档之间联系的纽带。Location 对象提供的方法可通过传入的 URL 将文档装入浏览器，并通过其属性保存 URL 的各项信息，如网络协议、主机名、

端口号等。

注意：search 代表的产旬字符串有多种形式，其形式与搜索引擎相关。如常见的名-值对应格式，用等号（=）分开名字与对应的值，多个名值之间使用连接符（&）连接。上述的搜索字符串可表示为：?name=zzangpu&num=200104&code=014104。在实际应用中常将查询字符串使用 escape()函数转换位 URL 适用的格式。

7.6.2 Location 对象属性与 URL 的对应

浏览器载入目标页面后，Location 对象的诸多属性保存了该页面 URL 的所有信息，其属性、方法及浏览器支持情况如表 7.6 所示。

表 7.6 Location对象的属性列表

类型	项目	简要说明	浏览器支持
属性	hash	保存URL的散列参数部分，将浏览器引导到文档中锚点	NN2+、IE3+
	host	保存URL的主机名和端口部分	NN2+、IE3+
	hostname	保存URL的主机名	NN2+、IE3+
	href	保存完整的URL	NN2+、IE3+
	pathname	保存URL完整的路径部分	NN2+、IE3+
	port	保存URL的端口部分	NN2+、IE3+
	protocol	保存URL的协议部分，包括协议之后的冒号	NN2+、IE3+
	search	保存URL的查询字符串部分	NN2+、IE3+
方法	assign(URL)	将以参数传入的URL赋予Location对象或其href属性	NN2+、IE3+
	reload(Boolean)	重载（刷新）当前页面	NN3+、IE4+
	replace(URL)	载入以参数URL传入的地址对应的文档	NN3+、IE4+

在 URL 载入后，其各个部分将分别由 Location 对象的各个属性保存起来。考察如下典型网页的 URL 地址实例：

```
http://www.webname.com:80/MyWeb/htdocs/query.cgi?name+num+code#MyPart3
```

浏览器载入该 URL 对应的页面时创建 Window 对象，并立即创建 Location 对象，且将 URL 的各个部分作为其属性值保存起来，如表 7.7 所示。

表 7.7 创建Location对象后各属性与其值的对应表

属性	取值
hash	#MyPart3
host	www.webname.com:80
hostname	www.webname.com
href	http://www.webname.com:80/MyWeb/htdocs/query.cgi?name+num+code#MyPart3
pathname	/MyWeb/htdocs/query.cgi
port	80
protocol	http:
search	?name+num+code

在使用 Location 对象的属性获得了 URL 地址的各个部分之后，可重新对属性赋值，实现页面跳转、锚点间转移、指定搜索串等功能。考察如下使用 Location 对象的 hash 属性进行锚点间跳转的实例代码：

```
//源程序 7.12
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
```

```
<script language="JavaScript" type="text/javascript">
<!--
var index=0;
var AnchorArray=new Array("StartAnchor","FirstAnchor","SecondAnchor",
                          "ThirdAnchor","ForthAnchor");
function GoAnchorNext()
{
  window.location.hash=AnchorArray[index];
  if(index<5)
  {
    index=index+1;
  }
  else
    index=0;
  return;
}
//-->
</script>
</head>
<body>
<center>
<p><b>锚点间跳转实例</b></p>
<p><a id="StartAnchor">Start Anchor</a></p>
<p><a id="FirstAnchor">First Anchor</a></p>
<p><a id="SecondAnchor">Second Anchor</a></p>
<p><a id="ThirdAnchor">Third Anchor</a></p>
<p><a id="ForthAnchor">Forth Anchor</a></p>
<form name="MyForm">
  <input type="button" name="MyButton" value="使用 hash 进行锚点转换"
  onclick="GoAnchorNext()"><br>
</form>
</center>
</body>
</html>
```

程序运行结果如图 7.36 所示。

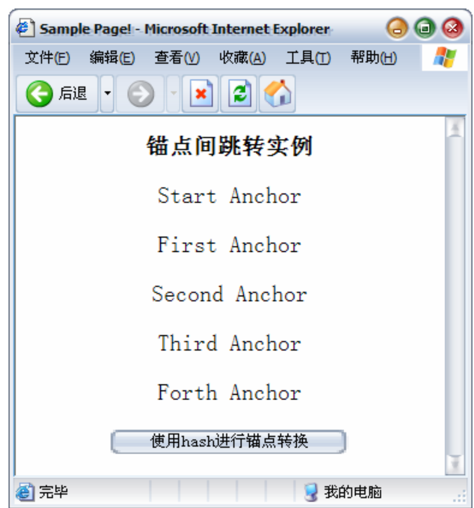

图 7.36 使用 Location 对象的 hash 属性进行锚点间跳转

单击页面中的“使用 hash 进行锚点转换”按钮，锚点在五个锚点间循环移动。主要使用的语句为：

```
window.location.hash=AnchorArray[index];
```

同样，可以使用 Location 对象的 href 属性将浏览器页面导航到任意的 URL，方法如下：

```
window.location.href=newURL;
```

在框架集中，Location 对象的引用需遵循一定的原则，考察如下的包含框架集的简单网页代码：

```
//源程序 7.13
<html>
<head>
  <title>Sample Page!</title>
</head>
<frameset cols="50%,50%">
  <frame name="Frame01" src="01.html">
  <frame name="Frame02" src="02.html">
</frameset>
</html>
```

文档载入后，将产生几个 Location 对象，具体来说：

- window.location：显示运行包含此脚本的文档的框架 URL；
- parent.location：表示框架集的父窗口的 URL 信息；
- parent.frames[0].location：表示框架集中第一个可见框架的 URL 信息；
- parent.frames[1].location：表示框架集中第二个可见框架的 URL 信息；
- parent.otherFrameName.location：同一框架集中另一个框架的 URL 信息。

可见，对于框架集中某一个可见框架而言，访问其 URL 信息，需先将引用指向其父窗口，然后通过层次关系来调用。

7.6.3 使用 reload()方法重载页面

Location 对象的 reload()方法容易与浏览器工具栏中的 Reload/Refresh（重载/刷新）按钮发生混淆，但前者比后者可实现的重载方式要灵活得多。总体来讲，reload()方法可实现的重载方式主要有三种：

- 强制从服务器重载：每次刷新页面时，浏览器都默认请求 Web 服务器返回当前 URL 对应的页面给客户端，此法使用 true 作为参数；
- 启动会话时从服务器重载：如果 Web 服务器上的文档较之缓冲区内存放的文档新，或者缓冲区内根本没有这个文档，则从 Web 服务器重载当前 URL 对应的页面；
- 强制从缓冲区重载：每次刷新页面时，浏览器都默认请求浏览器从缓冲区载入当前 URL 对应的页面。如果缓冲区没有此页面，则从 Web 服务器重载。

前面已经讲述过，使用 History 对象的 go(0)方法也可实现页面的重载。从本质上讲，主要体现了两种不同性质的重载。

history.go(0)方法重载页面时，在缓冲区取得文档，并保持页面中许多对象的状态，仅改变其全局变量值及一些可设置但不可见的属性（如 hidden 输入对象的值等），该方法与浏览器工具栏内的 Reload/Refresh（重载/刷新）按钮功能相同。

location.reload(URL)方法重载页面时，不管是从 Web 服务器还是从缓冲区重载，目标页面内的所有对象都自动更新。

考察如下实现页面不同重载方式的代码：

```
//源程序 7.14
```

```
<!DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function softReload()
{
  history.go(0);
}
function hardReload()
{
  location.reload(true);
}
//-->
</script>
</head>
<body>
<center>
<p>学籍注册程序</p>
<form name="MyForm">
  <p>姓名 ：<input type="text" name="MyText" id="MyText" value="ZangPu"></p>
  <p>性别 ：<input type="radio" name="MyRadio" id="MyRadio" value=1 checked>Male
            <input type="radio" name="MyRadio" id="MyRadio" value=2>Female</p>
  <p>年级 ：<select name="MyClass" id="MyClass">
              <option selected>Class 1</option>
              <option>Class 2</option>
              <option>Class 3</option>
              <option>Class 4</option>
            </select>
  </p>
  <p><input type="button" name="Soft" value="使用 history.go(0)方法重载"
      onclick="softReload()"></p>
  <p><input type="button" name="Hard" value="使用 location.reload()方法重载"
      onclick="hardReload()"></p>
</form>
</center>
</body>
</html>
```

程序运行后，出现如图 7.37 所示的页面。

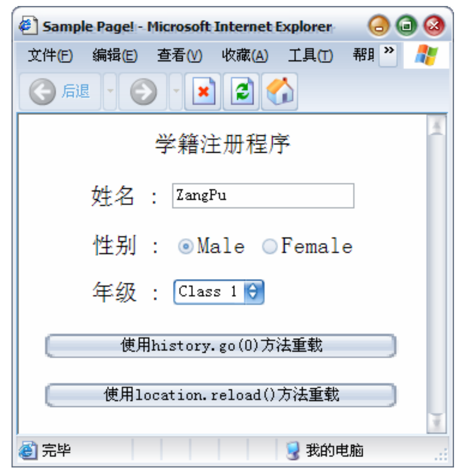

图 7.37 程序运行后的原始页面

在原始页面表单中更改“姓名”栏为“YSQ”、“性别”栏为“Female”、“年级”栏为“Class 3”后，出现如图 7.38 所示页面。

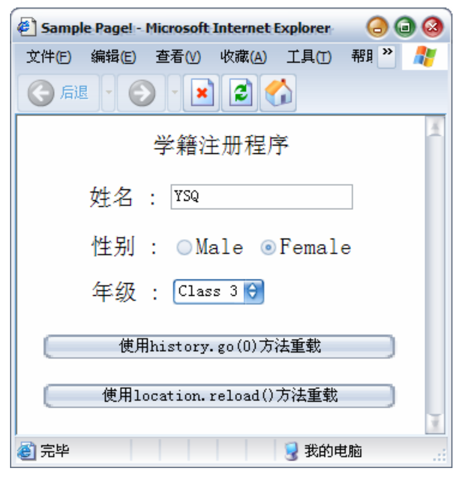

图 7.38 更改表单中各项参数

单击“使用 history.go(0)方法重载”按钮后，页面刷新但表单内容不变，结果如图 7.38 所示；单击“使用 location.reload()方法重载”按钮后，页面刷新且表单内容发生变化，结果如图 7.37 所示。

Location 对象在搜索引擎和页面的重定位、重载等方面应用比较广泛，实际使用过程中应综合使用字符串处理的有关方法如 substring()、escape()和 unescape()等方法分析目标 URL 地址以得到有用的信息。下面介绍与 Window 对象紧密相连的 Frame 对象。

7.7 Frame 对象

框架在 Web 应用程序设计中存在着很大的争议，某些场合使用框架能简化设计步骤，如多页面导航实例中，可在一个固定的框架内添加导航控件如图片、按钮等，而在另外的框架中显示导航的结果，这样客户端随时都可通过该导航控件控制其它框架的显示内容而不需

刷新整个文档。

浏览器载入含有框架的文档时自动创建 Frame 对象，并允许脚本通过调用 Frame 对象的属性和方法来控制页面中的框架。在 IE 和 NN 浏览器中，一般将 Frame 对象实现为 Window 对象，但前者不支持 close()方法关闭框架自身，并且 Frame 对象拥有其独有的属性和方法用于框架操作。在介绍 Frame 对象之前，先来了解框架集文档中对象的结构层次。

7.7.1 框架集文档中对象的结构

对于单个、无框架的文档，对象模型以 Window 对象开始，并将其作为对象模型层次的起始点和默认的对象而存在，实际使用过程中可忽略对该对象的引用。

对于含有多个框架的框架集文档，该框架集文档作为父窗口，且此时浏览器的标题栏显示的是框架集文档的标题。考察如下最简单的框架集文档代码：

```
//源程序 7.15
<html>
<head>
  <title>Sample Page!</title>
</head>
<frameset cols="50%,50%">
  <frame name="Frame01" id="Frame01" src="01.html">
  <frame name="Frame02" id="Frame02" src="02.html">
</frameset>
</html>
```

每个<Frame>标记都可以加载一个新的文档（当然也可引入新的框架集文档形成更为复杂的框架集文档）而产生各自的 document 对象，并可通过对象之间的层次关系进行访问操作。父子对象之间通过 parent 对象进行联系，且存在 top 对象指向所有框架唯一共有的顶层窗口。图 7.39 显示了上述框架集文档的结构：

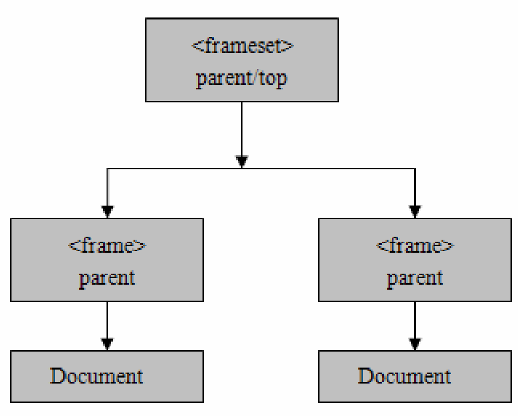

图 7.39 简单的框架集文档模型

在对象模型中构造一个对象的引用，首先要获得目标对象的位置信息，然后应了解用何种方法来实现该引用。在框架集文档中对框架的引用路径主要有三种：

- 子到父
- 父到子
- 子对子

在子到父的引用路径中，可使用 parent 关键字实现；父到子的引用可直接使用对象模型

层次；而子到子的访问则需要通过 top 关键字引用其共有的父对象，然后通过该父对象实现对另一框架的访问。综合如下：

```
this.parent;
parent.frameName;
top.otherFrameName.document;
```

Frame 对象的属性和方法受<frame>标记的控制，可在此标记内设定该框架的相关信息，如框架是否有滚动条、边框的颜色等。一般而言，在<frame>标记内应设置其 ID 属性（或 name 属性）以实现对象的有效引用。在上述最简单的框架集文档中，可通过如下方法实现对框架 Frame02 的 frameBorder 属性的访问（假设操作焦点在 Frame01 框架中）：

```
parent.document.all.Frame02.frameBorder;
parent.document.getElementById("Frame02").frameBorder;
```

在嵌套的框架集文档中，可根据对象模型层次从顶层的 Window 对象开始引用并到达最终的 Frame 对象，然后通过其属性和方法来操作该框架中载入的文档。

> 注意：如果框架集中某个框架载入了另外的框架集文档，则该框架的 top 属性属于另一个框架集文档中定义的框架集。如果总是把 top 设定为父对象，则该引用将不可实现。实际应用中可通过判断顶级文档是否为其 top 或 parent 属性载入，并根据结果实施页面重定位的方法来禁止框架中载入新的框架集文档，进而解决对象引用失效的问题。

7.7.2 控制指定的框架

Frame 对象提供诸多的属性和方法用于控制制定的框架，如更改框架是否能改变大小的标志、是否显示框架的滚动条等。

NN6+浏览器中实现的 Frame 对象提供 contentDocument 属性来引用与当前框架载入的文档相关的 document 对象，从而获取框架中其它有用的信息。IE5.5+和 NN7 浏览器实现的 Frame 对象提供 contentWindow 属性来引用与当前框架产生的窗口相关的 Window 对象，从而根据文档对象模型来获取 document 对象及其他信息。

例如某框架集文档包含两个框架分别为“FrameName1”和“FrameName2”，而 targetWin 是与“FrameName2”框架相关的 Window 对象，则在“FrameName1”框架所载入的文档中可通过下面句子实现对 targetWin 的引用：

```
var targetWin=parent.document.getElementById("FrameName1").contentWindow;
```

考察如下使用 Frame 对象的属性和方法控制框架的实例。首先考虑包含左右两个框架并设定了相关参数的框架集文档：

```
//源程序 7.16
<html>
<head>
  <title>Sample Page!</title>
</head>
<frameset name="MyFrameset" border=3 borderColor="black" cols="60%,40%">
  <frame name="Control" src="leftmain.html">
  <frame name="Display" frameBorder="yes" borderColor="#c0c0c0" src="target.html">
</frameset>
</html>
```

其中右侧框架中载入的“target.html”文档为“学籍注册程序”页面，包含文本框、单选框和下拉框等。其程序代码如下：

```
//源程序 7.17
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
```

```
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
</head>
<body>
<center>
<p>学籍注册程序</p>
<form name="MyForm">
  <p>姓名 ：<input type="text" name="MyName" id="MyName" value="ZangPu"></p>
  <p>性别 ：<input type="radio" name="MySex" id="MySex" value="男" checked>男
            <input type="radio" name="MySex" id="MySex" value="女">女</p>
  <p>年级 ：<select name="MyClass" id="MyClass">
              <option value="class 1" selected>Class 1</option>
              <option value="class 2" >Class 2</option>
              <option value="class 3" >Class 3</option>
              <option value="class 4" >Class 4</option>
            </select>
  </p>
  <p>
  <input type="button" name="MySubmit" value="确认"
     onclick="MySubmitFunc()">
  <input type="button" name="MyCancle" value="取消"
     onclick="MyCancleFunc()">
  </p>
</form>
</center>
</body>
</html>
```

其中 MySubmitFunc()、MyCancleFunc()事件处理程序为演示程序，读者可自行扩充以实现更为复杂的功能。框架集左侧框架载入的“leftmain.html”文档的代码如下：

```
//源程序 7.18
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//设置右侧框架信息
function SetInfo()
{
  //通过右侧框架对象的 name 属性定位该框架
  var targetFrame=parent.document.getElementById(parent.document.frames[1].name);
  //获取左侧框架中的"框架名称"文本框对象，并修改右侧框架对象的 name 属性
  var leftFrameName=parent.document.frames[0].document.all.MyForm.FrameName;
  if(leftFrameName.value=="")
  {
    alert("右侧框架名称信息不能为空!");
    leftFrameName.focus();
  }
```

```
    else
    {
        targetFrame.name=leftFrameName.value;
    }
    //获取左侧框架中的"边框显示"单选框对象,并设置右侧框架对象的 frameBorder 属性
    var leftFrameRadio=parent.document.frames[0].document.all.MyForm.MyRadio;
    for(i=0;i<leftFrameRadio.length;i++)
    {
        if(leftFrameRadio[i].checked)
            targetFrame.frameBorder=leftFrameRadio[i].value;
    }
    //获取左侧框架中的"边框颜色"下拉框对象,并设置右侧框架对象的 borderColor 属性
    var leftFrameColor=parent.document.frames[0].document.all.MyForm.MyColor;
    for(i=0;i<leftFrameColor.length;i++)
    {
        if(leftFrameColor[i].selected)
            targetFrame.borderColor=leftFrameColor[i].value;
    }
    //获取左侧框架中"姓名字段"文本框对象的 value 值,并修改右侧框架中"姓名"字段 value 属性
    var iMyName=parent.document.frames[0].document.all.MyForm.MyNameLeft.value;
    parent.document.frames[1].document.all.MyForm.MyName.value=iMyName;
    //获取左侧框架中"性别字段"的选择结果,并修改右侧框架中"性别"字段选择结果
    var iCheck=parent.document.frames[0].document.all.MyForm.MySexLeft;
    for(i=0;i<iCheck.length;i++)
    {
        if(iCheck[i].checked)
            parent.document.frames[1].document.all.MyForm.MySex[i].checked=true;
    }
    //获取左侧框架中"年级字段"的选择结果,并修改右侧框架中"年级"字段选择结果
    var iSelect=
    parent.document.frames[0].document.all.MyForm.MyClassLeft.options.selectedIndex;
    parent.document.frames[1].document.all.MyForm.MyClass.selectedIndex=iSelect;
}
//获取右侧框架信息
function AlertInfo()
{
    var nameTemp;
    var sexTemp;
    var classTemp;
    //获取右侧框架对象
    var targetFrame=parent.document.getElementById(parent.document.frames[1].name);
    //获取右侧框架中"姓名"文本框
    var iName=parent.document.frames[1].document.all.MyForm.MyName;
    nameTemp=iName.value;
    //获取右侧框架中"性别"单选框,并确定选中状态
    var iRadio=parent.document.frames[1].document.all.MyForm.MySex;
    for(i=0;i<iRadio.length;i++)
    {
        if(iRadio[i].checked)
            sexTemp=iRadio[i].value;
    }
    //获取右侧框架中"年级"下拉框,并确定选中状态
    var iClass=parent.document.frames[1].document.all.MyForm.MyClass;
```

```
  for(i=0;i<iClass.length;i++)
  {
    if(iClass[i].selected)
      classTemp=iClass[i].value;
  }
  //输出相关信息
  var msg="\n 右侧框架信息 :                                    \n\n";
  msg+="        框架名称 : "+targetFrame.name+"\n";
  msg+="        边框显示 : "+targetFrame.frameBorder+"\n";
  msg+="        边框颜色 : "+targetFrame.borderColor+"\n";
  msg+="        姓名字段 : "+nameTemp+"\n";
  msg+="        性别字段 : "+sexTemp+"\n";
  msg+="        年级字段 : "+classTemp+"\n";
  alert(msg);
}
//-->
</script>
</head>
<body>
<center>
<p>设置右侧框架信息</p>
<form name="MyForm">
  <p>框架名称 : <input type="text" name="FrameName" id="FrameName" value="Right"></p>
  <p>边框显示 : <input type="radio" name="MyRadio" id="MyRadio" value="yes" checked>是
                <input type="radio" name="MyRadio" id="MyRadio" value="no">否</p>
  <p>边框颜色 : <select name="MyColor" id="MyColor">
                  <option value="black" selected>黑色</option>
                  <option value="red">红色</option>
                  <option value="blue">蓝色</option>
                  <option value="green">绿色</option>
                </select>
  </p>
  //设置右侧框架的元素对象信息
  <p>姓名字段 : <input type="text" name="MyNameLeft" id="MyNameLeft" value="ZangPu"></p>
  <p>性别字段 : <input type="radio" name="MySexLeft" id="MySexLeft" value="男" checked>男
                <input type="radio" name="MySexLeft" id="MySexLeft" value="女">女</p>
  <p>年级字段 : <select name="MyClassLeft" id="MyClassLeft">
                  <option value="Class 1" selected>Class 1</option>
                  <option value="Class 2" >Class 2</option>
                  <option value="Class 3" >Class 3</option>
                  <option value="Class 4" >Class 4</option>
                </select>
  <p>
  <p><input type="button" name="setInfo" value="提交设置" onclick="SetInfo()"></p>
</form>
<hr>
<p>显示右侧框架信息</p>
<form>
  <p><input type="button" name="alertInfo" value="获取信息" onclick="AlertInfo()"></p>
</form>
</center>
</body>
</html>
```

程序运行后，出现如图 7.40 所示的页面。

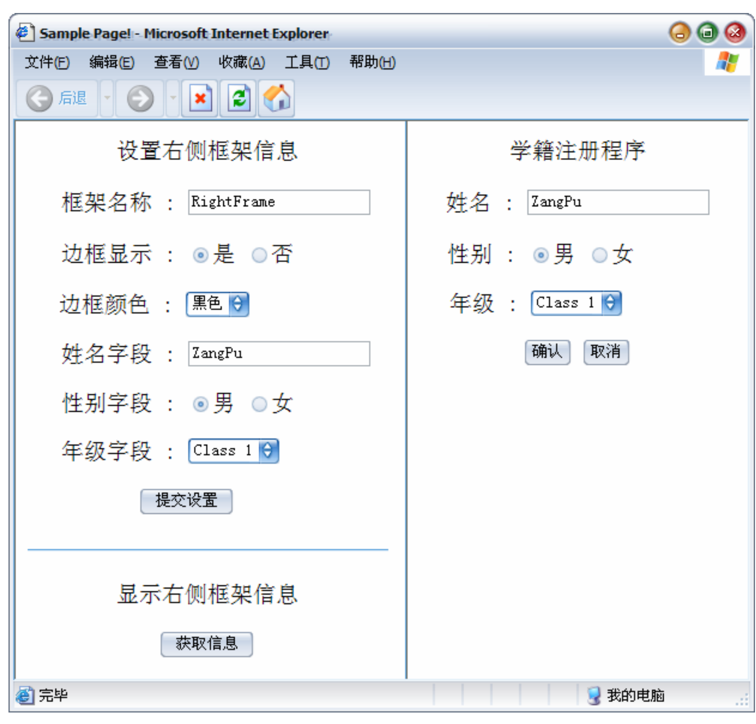

图 7.40 程序运行后的原始页面

此时单击“获取信息”按钮，则弹出包含右侧框架及其对应得 Frame 对象相关信息的警告框，如图 7.41 所示。

图 7.41 页面载入时根据页面初始值设定的右侧框架信息

在原始页面更改右侧框架及 Frame 对象的相关信息，将 Frame 对象的 name 属性改为“RightFrame”，frameBorder 属性改为“no”，borderColor 属性改为“red”，“姓名字段”文本框改为“YSQ”，“性别字段”单选框改为“女”，“年级字段”下拉框改为“class 3”。然后单击“提交设置”按钮后更新右侧框架及其对应的 Frame 对象相关信息。

单击“获取信息”按钮，弹出包含右侧框架及对应的 Frame 对象相关信息的警告框，如图 7.42 所示。

图 7.42 更改相关设置后更新右侧框架及 Frame 对象相关信息

不同的浏览器通常都会在文档内容和框架之间添加空白区域以便文档载入时其内容能自动插入到框架中。Frame 对象提供属性 marginHeight(上边界和下边界)、marginWidth(左边界和右边界)来表示该空白区域的大小。

值得注意的是，Frame 对象的属性如 frameBorder、marginHeight 等都可读可写，但框架集加载后，改变这些属性的取值能够改变其具体内容，并不能改变框架中文档的外观，而需通过 document 对象调用其对应的属性和方法来修改。

7.7.3 常见属性和方法汇总

Frame 对象提供的属性和方法较少，主要用于设置框架的标记（如 name、src 等属性）、改变框架的外观（如 borderColor、scrolling 等属性）等，表 7.8 列出了其常见的属性及浏览器版本支持情况。

表 7.8 Frame对象常见的属性汇总

属性	简要说明	浏览器支持
allowTransparency	标记框架的背景是否透明，为Boolean值	IE6+
borderColor	设置框架边框的颜色，为三组16进制值或颜色字符串	IE4+
contentDocument	对框架载入的目标文档对应的document对象的引用	NN6+
contentWindow	对与框架对应的窗口相关的Window对象的引用	NN7+、IE5.5+
frameBorder	设置框架是否包含边框，为字符串“yes”或“no”	NN6+、IE4+
height	保存框架的高度信息（像素）	IE4+
marginHeight	保存框架与所载入文档之间空白区域的高度（像素）	NN6+、IE6+
marginWidth	保存框架与所载入文档之间空白区域的宽度（像素）	NN6+、IE6+
name	返回与框架相关的标识符，用于框架的引用	NN6+、IE4+
noResize	表示框架是否能改变大小的状态标识符	NN6+、IE6+
scrolling	表示框架打开和关闭滚动条的状态标识符	NN6+、IE6+
src	框架载入的文档对应的URL地址	NN6+、IE6+
width	保存框架的宽度信息（像素）	IE4+

注意：如果某属性的取值为 Boolean 类型，则在调用该属性时可使用 1 代替 true、0 代替 false 作为其取值或返回的结果，下同。

框架（Frame）一般包含在框架集（Frameset）中，并且一个框架集一般包含多个框架。下面简要介绍与框架集相关的 Frameset 对象和 iframe 元素对象。

7.7.4 Frameset 对象

Frameset 对象主要用于处理框架与框架之间的关系，如框架之间边框的厚度（像素）、颜色及框架集的大小等。浏览器载入包含框架集的文档时，生成 Frameset 对象，表 7.9 列出了其常见的属性及浏览器版本支持情况。

表 7.9 Frameset对象常见属性

属性	简要说明	浏览器支持
border	保存框架集中各框架之间的边框厚度信息（像素）	IE4+
borderColor	保存框架边框的颜色，为三组16进制值或颜色字符串	IE4+
cols	保存框架集cols信息的字符串，以百分比或星号引入	NN6+、IE4+
frameBorder	设置框架是否包含边框，为字符串“yes”或“no”	IE4+
frameSpacing	保存框架集中各框架之间的间距（像素）	IE4+
rows	保存框架集rows信息的字符串，以百分比或星号引入	NN6+、IE4+

在框架集文档中，如果某属性未被指定，则该属性为空，浏览器一般以该属性的默认值来定义框架集，如框架集文档定义中 frameSpacing 属性未被指定时，该框架集中框架之间的间距为默认值 2 像素。

一般而言，在框架集中访问 Frameset 对象的属性可通过如下方法：

```
（IE4+）          document.all.FramesetID.property
（IE5+/W3C）      document.getElementById(FramesetID).property
```

在框架集的某个框架内，可通过如下方法访问该 Frameset 对象：

```
（IE4+）          parent.document.all.FramesetID.property
（IE5+/W3C）      parent.document.getElementById(FramesetID).property
```

考察如下的框架集文档代码：

```
//源程序 7.19
<!DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
  <title>Sample Page!</title>
</head>
<frameset name="FramesetOuter" id="FramesetOuter" border=5 frameBorder="yes"
          borderColor="green" frameSpacing=10 rows="290,*" cols="60%,100,100">
  <frame name="Control"  scrolling="no" src="main.html">
  <frame name="Control"  scrolling="no" src="target1.html">
  <frame name="Control"  scrolling="no" src="target2.html">
  <frame name="Control"  scrolling="no" src="target3.html">
  <frame name="Control"  scrolling="no" src="target4.html">
  <frame name="Control"  scrolling="no" src="target5.html">
</frameset>
</html>
```

该框架集包含六个框架，其中“target1.html”、“target2.html”、…、“target5.html”文档格式基本相同，文档“target1.html”的代码如下：

```
//源程序 7.20
<!DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
```

```
</head>
<body>
<center>
<br>
<p>学籍注册</p>
步骤之一
</center>
</body>
</html>
```

框架一载入的文档“main.html”实现对 Frameset 对象各属性的访问，同时可根据文档中文本框、单选框和下拉框对应的值更新该对象的各项属性，同时更新框架集文档视图。其代码如下：

```
//源程序 7.21
<!DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\n 框架集信息 : \n\n";
msg+="结果(修改前) : \n";
//修改指定框架集各项属性并输出相关信息
function SetInfo()
{
  AlertInfo();
  var iFrameset=parent.document.getElementById("FramesetOuter");
  //设置边框显示与否
  var iFrameBorder=parent.document.frames[0].document.all.MyForm.MyFrameBorder;
  for(i=0;i<iFrameBorder.length;i++)
  {
    if(iFrameBorder[i].checked)
      iFrameset.frameBorder=iFrameBorder[i].value;
  }
  //设置边框厚度
  var iBorder=parent.document.frames[0].document.all.MyForm.MyBorder.value;
  if(iBorder==""||parseInt(iBorder)==NaN||parseInt(iBorder)<=0)
  {
    alert("\n 错误提示信息 : \n\n''边框厚度''字段所填数据不合法!\n");
  }
  else
    iFrameset.border=iBorder;
  //设置边框颜色
  var iBorderColor=
  parent.document.frames[0].document.all.MyForm.MyBorderColor.options.selectedIndex;
  iFrameset.BorderColor=
  parent.document.frames[0].document.all.MyForm.MyBorderColor.options[iBorderColor].value;
  //设置框架间距
  var iFrameSpacing=parent.document.frames[0].document.all.MyForm.MyFrameSpacing.value;
  if(iFrameSpacing==""||parseInt(iFrameSpacing)==NaN||parseInt(iFrameSpacing)<=0)
  {
    alert("\n 错误提示信息 : \n\n''框架间距''字段所填数据不合法!\n");
```

```
    }
    else
      iFrameset.frameSpacing=iFrameSpacing;
    //设置行高信息
    iFrameset.rows=parent.document.frames[0].document.all.MyForm.MyRows.value;
    iFrameset.cols=parent.document.frames[0].document.all.MyForm.MyCols.value;
    msg+="结果(修改后) : \n";
    AlertInfo();
    alert(msg);
    return;
}
//获取指定框架集信息
function AlertInfo()
{
    var iFrameset=parent.document.getElementById("FramesetOuter");
    msg+="          边框显示 ：" +iFrameset.frameBorder+ "\n";
    msg+="          边框厚度 ：" +iFrameset.border+ "\n";
    msg+="          边框颜色 ：" +iFrameset.borderColor+ "\n";
    msg+="          框架间距 ：" +iFrameset.frameSpacing+ "\n";
    msg+="          行高信息 ：" +iFrameset.rows+ "\n";
    msg+="          列宽信息 ：" +iFrameset.cols+ "              \n\n";
}
//-->
</script>
</head>
<body>
<center>
<br>
设置并显示框架集信息
<form name="MyForm">
    边框显示 ：<input type="radio" name="MyFrameBorder" id="MyFrameBorder" value="yes"
              checked>是
              <input type="radio" name="MyFrameBorder" id="MyFrameBorder" value="no">否
              <br>
    边框厚度 ：<input type="text" name="MyBorder" id="MyBorder" size="11" value="5"><br>
    边框颜色 ：<select name="MyBorderColor" id="MyBorderColor">
                  <option value="black" selected>黑色</option>
                  <option value="red">红色</option>
                  <option value="blue">蓝色</option>
                  <option value="green">绿色</option>
               </select><br>
    框架间距 ：<input type="text" name="MyFrameSpacing" id="MyFrameSpacing" size="11"
              value="10"><br>
    行高信息 ：<input type="text" name="MyRows" id="MyRows" size="11" value="290,*"><br>
    列宽信息 ：<input type="text" name="MyCols" id="MyCols" size="11" value="60%,100,100"><br>
    <p><input type="button" name="setInfo" value="提交更改并获取信息" onclick="SetInfo()"></p>
</form>
</center>
</body>
</html>
```

程序运行后，出现如图 7.43 所示的页面。

图 7.43 程序运行后的原始页面

在该页面中修改表单中各项 HTML 元素对应的值，如选择“边框显示”为“否”、“框架间距”为 5，单击“提交更改并获取信息”按钮，将触发 SetInfo()函数更新 Frameset 对象各个对应的属性。浏览器根据 Frameset 对象的新属性值来刷新目标页面的视图，如图 7.44 所示。

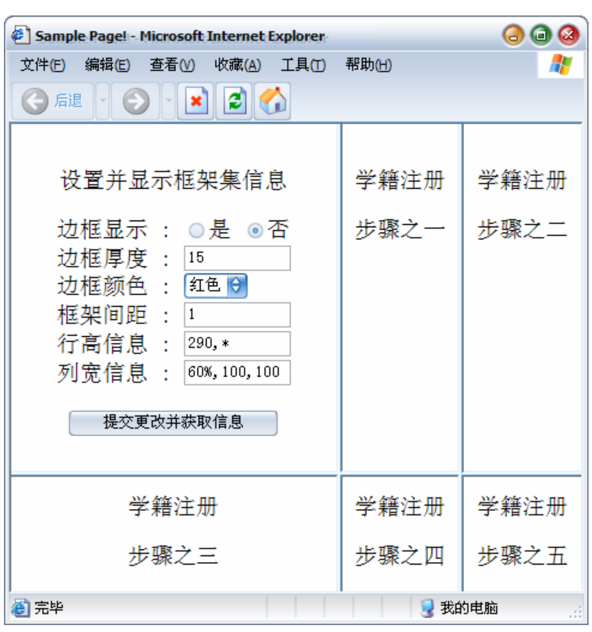

图 7.44 更改 Frameset 对象的属性后刷新视图

在更新目标页面视图的同时，将弹出包含修改前后框架集文档对应的 Frameset 对象属性值对比信息的警告框。如选择“边框显示”为“否”、“边框厚度”为 10、“框架间距”为 20，并单击“提交更改并获取信息”按钮后，浏览器更新目标页面视图并弹出警告框如图 7.45 所示。

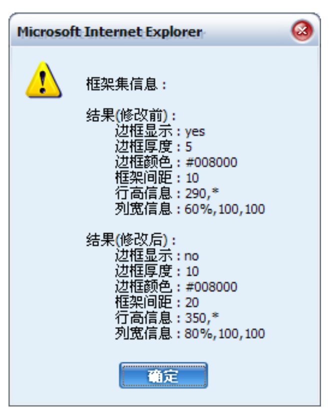

图 7.45 修改前后 Frameset 对象各属性值对比

值得注意的是，Frameset 对象的属性均为可读可写，但在框架集载入后，某些属性的更改并不反映到页面布局上来，如 frameBorder、border 属性等，其更多的是作为可读的属性而存在。

由上面的实例不难看出，框架集文档中的 Frameset 对象在框架管理方面提供快捷的途径来操作指定的项目并能实时更新目标页面。当框架集文档中所含的框架比较多或结构嵌套比较复杂的情况下，一般设定框架的 name 属性（或 id 属性），然后通过父窗口 document 对象的 getElementById(FrameID)方法准确定位，并进行相关操作。

7.7.5 iframe 元素对象

iframe 元素对象本质上是通过<iframe>和</iframe>标记对嵌入目标文档到父文档时所产生的对象，表示浮动在父窗口中的目标文档，表 7.10 列出了其常见的属性及浏览器版本支持情况。

表 7.10 iframe元素对象常见属性汇总

属性	简要说明	浏览器支持
align	根据文档中其他元素的位置对目标iframe元素进行定位	NN6+、IE4+
allowTransparency	标识iframe的背景是否透明	IE6+
contentDocument	包含对iframe框架内文档对象的引用	NN6+
contentWindow	包含对iframe框架所产生的Window对象的引用	NN7+、IE5.5+
height	保存iframe框架的高度信息（像素）	NN6+、IE4+
Hspace	保存iframe框架左右边框的页边空白（像素）	IE4+
longDesc	提供一个包含iframe元素详细描述的文档的URL	NN6+、IE6+
marginHeight	保存iframe框架上下边框与外部元素之间的间隔（像素）	NN6+、IE4+
marginWidth	保存iframe框架左右边框与外部元素之间的间隔（像素）	NN6+、IE4+
name	标识目标iframe框架的字符串	NN6+、IE4+
scrolling	标识目标iframe框架是否含有的滚动条的Boolean值	NN6+、IE4+
src	保存嵌入目标iframe框架内文档的URL	NN6+、IE4+
Vspace	保存iframe框架上下边框的页边空白（像素）	IE4+
Width	保存iframe框架的宽度信息（像素）	NN6+、IE4+

iframe 元素对象的访问方法与 Frameset 对象类似，在父文档中访问 iframe 元素对象的属性可通过如下方式：

```
（IE4+）        document.all.iframeID.property
（IE4+/NN6+）   window.frames[iframeID].property
（IE5+/W3C）    document.getElementById(iframeID).property
```

在包含 iframe 元素对象的文档内，可通过如下方式访问该 iframe 元素对象：

```
（IE4+）        parent.document.all.iramesetID.property
（IE5+/W3C）    parent.document.getElementById(FramesetID).property
```

考察如下通过 iframe 元素对象的属性操作其对应框架的代码：

```
//源程序 7.22
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\niframe 元素对象对应的框架信息 : \n\n";
msg+="结果(修改前) : \n";
//返回下拉框被选择项的序号
function CheckIndex(iSelect)
{
  var index;
  for(i=0;i<iSelect.length;i++)
  {
    if(iSelect[i].selected)
      index=i;
  }
  return index;
}
//修改指定框架集信息
function SetInfo()
{
  ChangeInfo();
  msg+="结果(修改后) : \n";
  var iFrame=document.getElementById("MyIframe");
  //对齐方式
  var iAlign=document.all.MyForm.MyAlign;
  iFrame.align=iAlign[CheckIndex(iAlign)].value;
  //背景透明
  var iAllowTransparency=document.all.MyForm.MyAllowTransparency;
  iFrame.allowTransparency=iAllowTransparency[CheckIndex(iAllowTransparency)].value;
  //框架高度
  var iHeight=document.all.MyForm.MyHeight;
  iFrame.height=iHeight[CheckIndex(iHeight)].value;
  //框架宽度
  var iWidth=document.all.MyForm.MyWidth;
  iFrame.width=iWidth[CheckIndex(iWidth)].value;
  //空白区域高度
  var iMarginHeight=document.all.MyForm.MyMarginHeight;
  iFrame.marginHeight=iMarginHeight[CheckIndex(iMarginHeight)].value;
  //空白区域宽度
  var iMarginWidth=document.all.MyForm.MyMarginWidth;
  iFrame.marginWidth=iMarginWidth[CheckIndex(iMarginWidth)].value;
  //滚动显示
  var iScrolling=document.all.MyForm.MyScrolling;
  iFrame.scrolling=iScrolling[CheckIndex(iScrolling)].value;
  //文档地址
```

```
    var iSrc=document.all.MyForm.MySrc;
    if(iSrc.value=="")
    {
       alert("''文档地址''字段不能为空!");
       iSrc.focus();
    }
    else
    {
       iFrame.src=iSrc.value;
    }
    ChangeInfo();
    //输出信息并更新页面
    alert(msg);
    return;
}
//获取框架信息
function ChangeInfo()
{
    var iFrame=document.getElementById("MyIframe");
    msg+="        框架对齐 ：" +iFrame.align+ "\n";
    msg+="        背景透明 ：" +iFrame.allowTransparency+ "\n";
    msg+="        框架高度 ：" +iFrame.height+ "\n";
    msg+="        框架宽度 ：" +iFrame.width+ "\n";
    msg+="        空白高度 ：" +iFrame.marginHeight+ "\n";
    msg+="        空白宽度 ：" +iFrame.marginWidth+ "\n";
    msg+="        水平空白 ：" +iFrame.hspace+ "\n";
    msg+="        垂直空白 ：" +iFrame.vspace+ "\n";
    msg+="        滚动显示 ：" +iFrame.scrolling+ "\n";
    msg+="        文档地址 ：" +iFrame.src+ "\n\n";
    return;
}
//-->
</script>
</head>
<body>
<table height=380 width=650 border=2 borderColor="#c0c0c0">
    <tr>
       <td height=380 width=250>
          <center>设置并显示框架集信息</center>
          <form name="MyForm">
             框架对齐 ：<select name="MyAlign" id="MyAlign">
                         <option value="absbottom">absbottom</option>
                         <option value="absmiddle">absmiddle</option>
                         <option value="baseline">baseline</option>
                         <option value="bottom">bottom</option>
                         <option value="left">left</option>
                         <option value="middle" selected>middle</option>
                         <option value="right">right</option>
                         <option value="texttop">texttop</option>
                         <option value="top">top</option>
                      </select><br>
             背景透明 ：<select name="MyAllowTransparency" id="MyAllowTransparency">
                         <option value="true" selected>true</option>
```

其中“空白高度”和“空白宽度”表示框架中的内容与边框之间的距离（像素），而“水平空白”和“垂直空白”表示框架与其外部文档之间的距离（像素）；测试文档“target.html”和“other.html”为普通的 HTML 页面。

```
                        <option value="false">false</option>
                      </select><br>
            框架高度 ：<select name="MyHeight" id="MyHeight">
                        <option value=200 selected>200pixels</option>
                        <option value=250>250pixels</option>
                        <option value=300>300pixels</option>
                        <option value=350>350pixels</option>
                      </select><br>
            框架宽度 ：<select name="MyWidth" id="MyWidth">
                        <option value=200 selected>200pixels</option>
                        <option value=250>250pixels</option>
                        <option value=300>300pixels</option>
                        <option value=350>350pixels</option>
                      </select><br>
            空白高度 ：<select name="MyMarginHeight" id="MyMarginHeight">
                        <option value=10 selected>10 pixels</option>
                        <option value=15 selected>15 pixels</option>
                        <option value=20>20 pixels</option>
                        <option value=25>25 pixels</option>
                      </select><br>
            空白宽度 ：<select name="MyMarginWidth" id="MyMarginWidth">
                        <option value=10>10 pixels</option>
                        <option value=15 selected>15 pixels</option>
                        <option value=20>20 pixels</option>
                        <option value=25>25 pixels</option>
                      </select><br>
            滚动显示 ：<select name="MyScrolling" id="MyScrolling">
                        <option value="auto">auto</option>
                        <option value="yes" selected>yes</option>
                        <option value="no">no</option>
                      </select><br>
            文档地址 ：<input type="text" name="MySrc" id="MySrc" size="8" value="other.html"><br>
            <p><input type="button" name="setinfobutton" value="提交更改并获取信息"
                onclick="SetInfo()"></p>
          </form>
        </td>
        <td height=380 width=400>
             框架集对象的属性均为可读可写,但某些属性的更改并不反映到页面视图中.
             <iframe name="MyIframe" id="MyIframe" align="absbottom" allowTransparency=true
                     height=200 width=200 frameborder=10 marginHeight=15 marginWidth=15
                     scrolling="yes"   hspace=20 vspace=10 src="target.html">
             </iframe>
             当框架集文档中所含的框架比较多时,一般需设定框架的 name 属性或 id 属性.
        </td>
      </tr>
</table>
</body>
</html>
```

其中“空白高度”和“空白宽度”表示框架中的内容与边框之间的距离（像素），而“水平空白”和“垂直空白”表示框架与其外部文档之间的距离（像素）；测试文档“target.html”和“other.html”为普通的 HTML 页面。

程序运行后，出现如图 7.46 所示的页面。

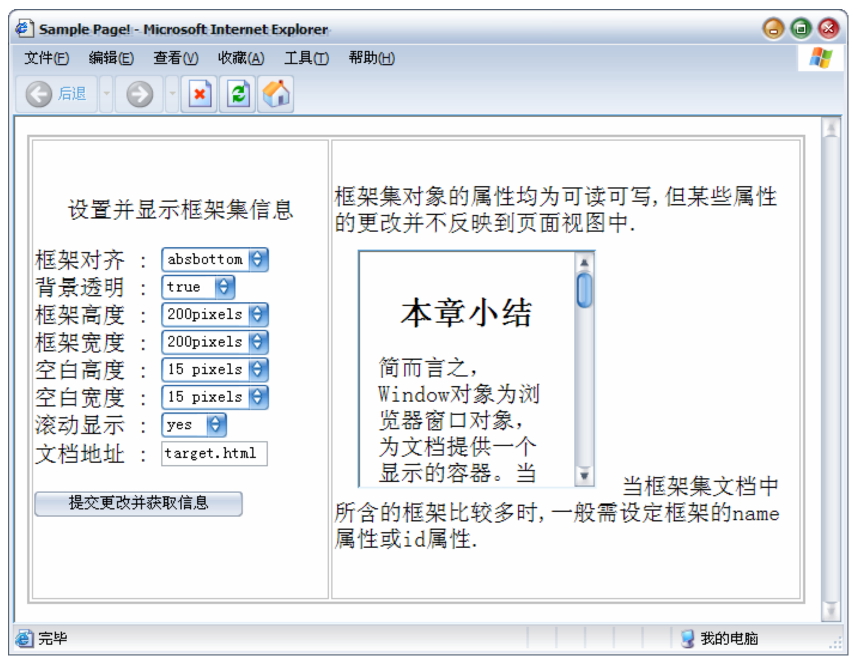

图 7.46 程序运行后出现的原始页面

在原始页面中修改各个下拉框的选项及文本框的内容后，单击“提交更改并获取信息”按钮，将弹出包含修改前后框架对应的 iframe 元素对象各属性值对比信息的警告框。如修改 “框架对齐”为“middle”、“框架高度”为 300、“框架宽度”为 350 等信息，并单击“提交更改并获取信息”按钮后，浏览器根据设置的属性值更新目标页面视图并弹出警告框如图 7.47 所示。

图 7.47 修改前后 iframe 元素对象各属性值对比

iframe 对象的 align 属性表示框架与其外内容的对齐方式，可以利用 JavaScript 脚本动态调整该属性来定位目标框架。表 7.11 列出了该属性的可能取值及简要说明：

表 7.11 iframe元素对象的align属性的可能属性值

取值	简要说明
absbottom	框架的底部与周围文本下方的虚线对齐
absmiddle	框架的中部与周围文本的top和absbottom值之间的中心点对齐
baseline	框架的滚动条按钮与周围文本的基线对齐
bottom	框架的滚动条按钮与周围文本的基线对齐（IE）
left	根据新设定的属性值刷新框架，同时使该框架与包含元素的左边缘对齐
middle	框架的垂直方向中心线与周围文本的虚垂直中心线对齐
right	根据新设定的属性值刷新框架，同时使该框架与包含元素的右边缘对齐
texttop	框架顶部与周围文本顶部最高点的虚线对齐
top	框架顶部与周围元素顶部最高点的虚线对齐

值得注意的是，除 contentDocument 和 contentWindow 属性为只读外，iframe 对象的其余属性均为可读可写，但并非所有的属性被改变后都能实时更新目标 iframe 框架，如控制滚动条显示与否的 scrolling 属性、控制框架与其外部的文档之间空白尺寸的 hspace 和 vspace 属性等，其更多的是作为一种只读属性而存在。

下面简要介绍顶级对象模型中的 Document 对象。

7.8 Document 对象

JavaScript 主要作为一门客户端脚本而存在，在与客户交互的过程中 Document 对象扮演着及其重要的角色。深入理解 Document 对象对编写跨浏览器的 Web 应用程序是 JavaScript 脚本程序员进阶的关键。

Document 对象在顶级对象模型中处于较低的层次，通俗地说，浏览器载入文档后，首先产生顶级对象模型中的 Window、Frame 等对象，且当该文档包含框架集时，一个 Window 对象下面可包含多个 Document 对象。

Document 对象及其组件相关的内容相当丰富，在“Document 对象”章节将对 Document 对象作专门的讲述。

7.9 本章小结

本章主要讲述了 Window 及相关顶级对象如 Frames、Navigator、Screen、History、Location、Document 等，并从实际应用的角度分析了其创建过程、属性、方法、操作实例等。并重点介绍了如何通过这些对象创建和管理浏览器窗口及文档中的框架。

Document 对象提供了与客户交互的平台，使用 JavaScript 脚本操作 Document 对象可以创建动态、交互性较强的页面，下一章将重点介绍 Document 对象的创建，以及如何使用 JavaScript 脚本调用其属性和方法来动态更新页面。

第 8 章 Document 对象

Document 对象在顶级对象模型中占据非常重要的地位，它可以更新正在装入和已经装入的文档，并使用 JavaScript 脚本访问其属性和方法来操作已加载文档中包含的 HTML 元素，如表单 form、单选框 radio、下拉框 checkbox 等，并将这些元素当作具有完整属性和方法的元素对象来引用。本章将重点讲述顶级对象模型中 Document 对象及与其相关的 body 元素对象的基础知识，如对象的创建、引用及与其它 HTML 元素对象之间的相互关系等。

8.1 对象模型参考

客户端浏览器载入目标 HTML 文档后，在创建顶级对象模型中其它顶级对象的同时，创建 Document 对象的实例，并将该实例指向当前的文档。当文档包含多个框架组成的框架集或者在该文档中由<iframe>和</iframe>标记对引入其它外部文档时，当前浏览器窗口就同时包含了多个 Document 对象。Web 程序开发人员根据对象之间的相对位置关系使用 JavaScript 脚本进行相关操作如对象定位、访问等。

Document 对象在文档结构模型中处于顶级层次，但较之如 Window 等其它顶级对象而言，该对象与客户端浏览器的关联程度比较小，而与所载入文档本身的关联程度较为紧密。图 8.1 从 Document 对象的角度出发，显示了它在文档对象模型的参考层次中所处的相对位置（NN4+和 IE4+文档结构模型通用）：

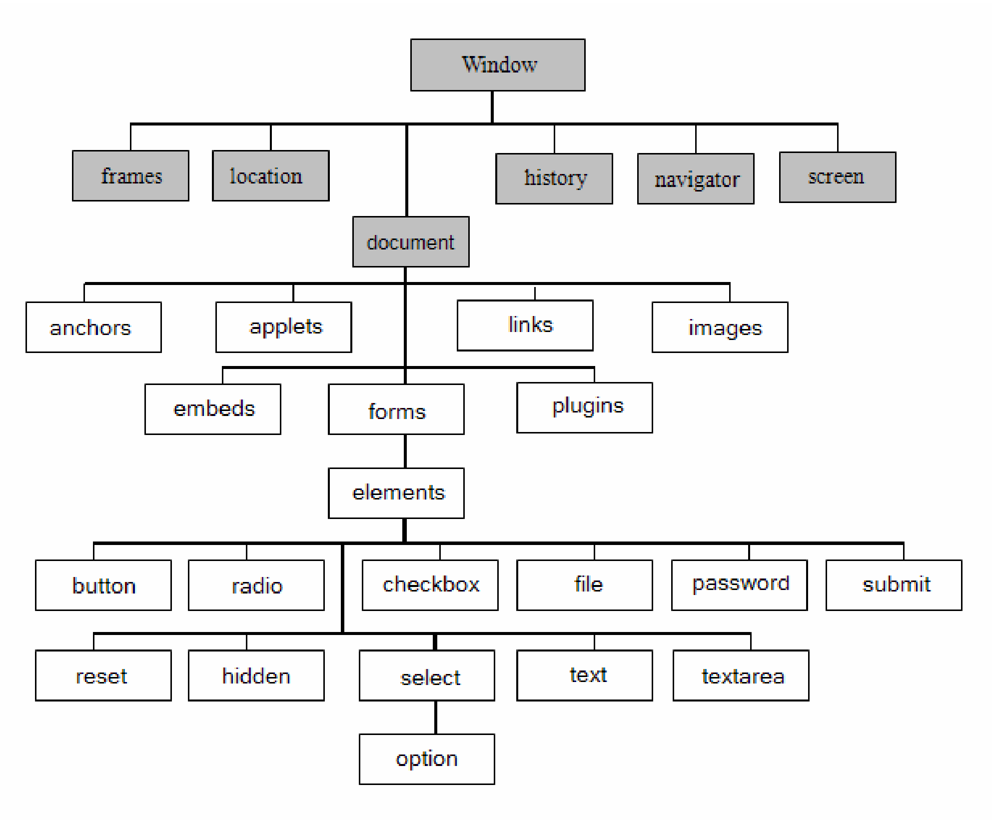

图 8.1 Document 对象模型参考

在上述的对象模型参考中，灰色表示的是DOM中的顶级对象，而Document对象所在层次之下的对象为目标文档包含的HTML元素对象。可见在文档中定位了Document对象之后，就可根据对象的层次关系操作其层次之下任意的元素对象。

注意：上面描述的对象模型中frames分别作为顶级对象和Document对象包含的元素对象而存在，因为当某文档包含框架集时，frames对象作为该文档对应的Document对象的元素对象而存在。当框架集中某个框架载入另一个文档时，该文档对应的Document对象又作为frames对象下一层次的对象而存在。

8.2 Document对象

Document对象包括当前浏览器窗口或框架内区域中的所有内容，包含文本域、按钮、单选框、复选框、下拉框、图片、链接等HTML页面可访问元素，但不包含浏览器的菜单栏、工具栏和状态栏。

Document对象提供多种方式获得HTML元素对象的引用，如在某目标文档中含有多个通过<form>和</form>标记对引入的表单，则可通过如下方式获得对该文档中forms对象数组长度信息的引用：

```
document.forms.length
document.getElementsByTagName("form").length
```

获取了对象数组信息后，就可以根据目标文档中该类型对象的相对位置定位某对象，如循环检索forms数组各表单的name属性的代码：

```
var MyForms=document.forms;
for(i=0;i<MyForms.length;i++)
{
  msg+="forms[" +i+ "].name : " +MyForms[i].name+ "\n";
}
```

代码运行后，将根据forms对象数组的长度信息遍历该数组并输出各表单name属性值。

8.2.1 获取目标文档信息

浏览器载入目标文档后，将根据文档标记的类型产生该类型的对象数组，并以标记元素载入的时间顺序进行数组下标分配。考察如下获取Document对象信息的代码，其中框架集文档“main.html”代码如下：

```
//源程序 8.1
<html>
<head>
  <title>Sample Page!</title>
</head>
<frameset name="MyFrameset" border=3 borderColor="black" cols="60%,40%">
  <frame name="Control" src="leftmain.html">
  <frame name="Display" src="target.html" frameBorder="yes" borderColor="#c0c0c0">
</frameset>
</html>
```

该框架集文档包含右框架文档“target.html”和左框架文档“leftmain.html”，其中前者为普通测试文档，而后者为包含链接、图片、插件、表单等HTML页面可见元素的文档，其代码如下：

```
//源程序 8.2
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
var msg="\nDocument 对象信息 ：\n\n";
//获取 Document 对象信息
function GetInfo()
{
  //获取父文档的 frames 对象数组
  var MyFrames=parent.document.frames;
  msg+="父文档 frames 数组 ：\n";
  msg+="长度 ："+MyFrames.length+"\n";
  for(i=0;i<MyFrames.length;i++)
  {
    msg+="frames[" +i+ "].name : " +MyFrames[i].name+ "\n";
  }
  //获取文档的 links 对象数组
  var MyLinks=document.links;
  msg+="links 数组 ：\n";
  msg+="长度 ："+MyLinks.length+"\n";
  for(i=0;i<MyLinks.length;i++)
  {
    msg+="links[" +i+ "].name : " +MyLinks[i].name+ "\n";
  }
  //获取文档的 images 对象数组
  var MyImages=document.images;
  msg+="images 数组 ：\n";
  msg+="长度 ："+MyImages.length+"\n";
  for(i=0;i<MyImages.length;i++)
  {
    msg+="images[" +i+ "].name : " +MyImages[i].name+ "\n";
  }
  //获取文档的 embeds 对象数组
  var MyEmbeds=document.embeds;
  msg+="embeds 数组 ：\n";
  msg+="长度 ："+MyEmbeds.length+"\n";
  for(i=0;i<MyEmbeds.length;i++)
  {
    msg+="embeds[" +i+ "].name : " +MyEmbeds[i].name+ "\n";
  }
  //获取文档的 plugins 对象数组
  var MyPlugins=document.plugins;
  msg+="plugins 数组 ：\n";
  msg+="长度 ："+MyPlugins.length+"\n";
  for(i=0;i<MyPlugins.length;i++)
  {
    msg+="plugins[" +i+ "].name : " +MyPlugins[i].name+ "\n";
  }
```

```
    //获取文档的 forms 对象数组
    var MyForms=document.forms;
    msg+="forms 数组 ：\n";
    msg+="长度 ："+MyForms.length+"\n";
    for(i=0;i<MyForms.length;i++)
    {
       msg+="forms[" +i+ "].name : " +MyForms[i].name+ "\n";
    }
    //输出文档信息
    alert(msg);
}
//-->
</script>
</head>
<body>
<center>
<p>获取文档 Document 对象信息</p>
</center>
<table border=1 borderColor="#c0c0c0" align=center>
    <tr>
       <td align=middle>
          链接 :<br><br>
          <a href="target1.html" target="Display" name="linka">超级链接一</a><br>
          <a href="target2.html" target="Display" name="linkb">超级链接二</a><br>
          <a href="target3.html" target="Display" name="linkc">超级链接三</a><br>
       </td>
       <td align=middle>
          图片 :<br>
          <img src="image1.JPG" name="image1" width="84" height="98" border=1></img>
          <img src="image2.JPG" name="image2" width="84" height="98" border=1></img>
       </td>
    </tr>
    <tr>
       <td align=middle>
          音乐 :<br>
          <embed name="music" src="town.mid" units="pixels" height=130 width=165>
       </td>
       <td align=middle>
          多媒体 :<br>
          <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" codebase=
               "http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab
               #version=6,0,29,0" width="150" height="150">
               <param name="movie" value="ClockFlash.swf">
               <param name="quality" value="high">
               <embed src="ClockFlash.swf" quality="high" width="150"height="150"
                    pluginspage="http://www.macromedia.com/go/getflashplayer"
                    type="application/x-shockwave-flash" >
               </embed>
          </object>
       </td>
    </tr>
    <tr>
       <td align=left>
```

```
            <center>表单一 :</center><br>
            <form name="MyForm1">
                单选框 : <input type="radio" name="MyRadio" value="yes" checked>是
                         <input type="radio" name="MyRadio" value="no">否<br>
                下拉框 : <select name="MySelect" id="MySelect">
                           <option value="black" selected>黑色</option>
                           <option value="green">绿色</option>
                         </select><br>
            </form>
        </td>
        <td align=left>
            <center>表单二 :</center><br>
            <form name="MyForm2">
                复选框 : <input type="checkbox" name="MyCheckbox"value="yes" checked><br>
                文本框 : <input type="text" name="MyText" size="11" value="Welcome!"><br>
            </form>
        </td>
    </tr>
</table>
<center>
<form name="MyForm3">
    <input type="submit" name="MySure" value="获取信息" onclick="GetInfo()">
</form>
<center>
</body>
</html>
```

程序运行后，出现如图 8.2 所示的原始页面。

图 8.2 程序运行后的原始页面

鼠标单击上述原始页面中的“获取信息”按钮后，触发 GetInfo()函数收集当前文档对应的 Document 对象相关信息，并使用警告框输出，如图 8.3 所示。

图 8.3 当前文档的 Document 对象信息

由上图可以看出，浏览器载入文档后，根据 HTML 元素载入的顺序和类型生成对象数组并按顺序分配数组下标以便 JavaScript 脚本对各元素对象进行访问。对象数组生成后，可根据数组中各元素的相对位置或其标识符（name 属性等）来引用。如在上述实例中，表单 MyForm3 载入时顺序为 3，则可通过下面的方式引用该表单的 name 属性：

```
document.forms["MyForm3"].name
document.forms[2].name
```

当然，也可使用如下方式直接进行访问：

```
document.all.MyForm3.name
document.getElementByName("MyForm3")
```

其中，getElementByName()为 W3C DOM Level 1 中定义的标准方法，目前通用的浏览器版本都基本实现了 W3C DOM Level 1 规范。同样，如果设置了标记元素的 id 属性，也可使用 getElementById()方法来代替该方法。

综上所述，Document 对象一般的处理步骤如下：

- 获取文档的指定对象类型的目标数组，如 images、forms 数组等；
- 使用目标数组的长度为参数遍历数组，根据提供的属性值检索出目标在对象数组中的位置信息；
- 使用目标对象的位置信息访问该对象的属性和方法。

在 W3C 新版本中定义了更多访问 Document 对象中标记元素对象的通用方法，同时浏览器版本的更新也对其进行了大量的扩展，本章仅讨论通用浏览器版本上的 Document 对象的相关知识。

8.2.2 设置文档颜色值

Document 对象提供了几个属性如 fgColor、bgColor 等来设置 Web 页面的显示颜色，它们一般定义在<body>标记中，在文档布局确定之前完成设置。例如：

```
<body bgColor="white" fgColor="green" linkColor="red" alinkColor="blue" vlinkColor="purple">
```

其中 bgColor 属性可通过 JavaScript 脚本动态改变。Document 对象提供有关颜色的属性

分别代表如下：

- bgColor 表示文档的背景色；
- fgColor 表示文档中文本的颜色；
- linkColor 表示文档中未访问链接的颜色；
- alinkColor 表示文档中链接被单击时出现的颜色；
- vlinkColor 表示文档中已访问链接的颜色；

考察如下设置文档中各项颜色的实例代码：

```
//源程序 8.3
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//设置文档的颜色显示
function SetColor()
{
  document.bgColor="white";
  document.fgColor="green";
  document.linkColor="red";
  document.alinkColor="blue";
  document.vlinkColor="purple";
}
//改变文档的背景色为黑色
function ChangeColorOver()
{
  document.bgColor="black";
  return;
}
//改变文档的背景色为黑色
function ChangeColorOut()
{
  document.bgColor="white";
  return;
}
//-->
</script>
</head>
<body onload="SetColor()">
<center>
<br>
<p>设置颜色</p>
<a href="target.html">链接实例</a>
<form name="MyForm3">
    <input type="submit" name="MySure" value="改变背景色"
              onmouseover="ChangeColorOver()"
              onmouseOut="ChangeColorOut()">
</form>
<center>
</body>
```

```
</html>
```

该程序使用<body>标记内的 onload()事件调用 SetColor()方法来设置文档页面各项颜色的初始值如背景色 bgColor 为颜色字符串常量“white”。程序运行后，将出现如图 8.4 所示的页面。

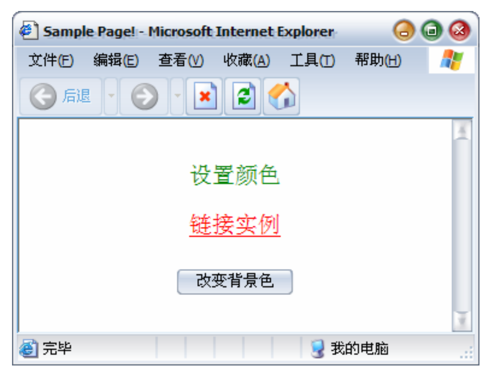

图 8.4 设置文档页面各项颜色的初始值

在上述页面中，鼠标移动到“改变背景色”按钮上时，触发 onmouseOver()事件调用 ChangeColorOver()函数来改变文档的背景颜色为黑色；当鼠标移离“改变背景色”按钮时，触发 onmouseOut()方法调用 ChangeColorOut()函数来改变文档的背景颜色为白色。

例如在页面中单击“链接实例”链接，并将鼠标移动到“改变背景色”按钮之上时，出现如图 8.5 所示的页面。

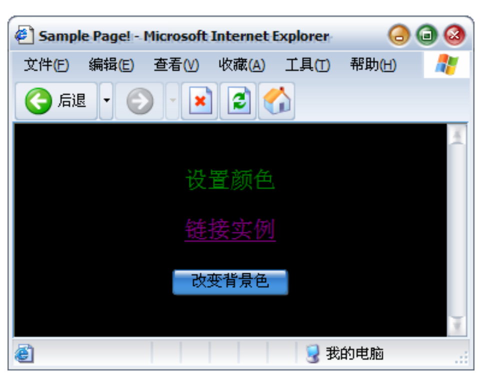

图 8.5 改变文档的背景色

在 HTML4 中，颜色有如下两种表示方式：

- 颜色字符串常量表示法：使用特定的字符串表示某种颜色，如字符串“blue”表示蓝色、“red”表示红色等。在 W3C 制定的 HTML 4.0 标准中，存在 16 个颜色字符串常量，详情请见表 8.1；
- RGB 原色表示法：RGB 是 Red、Green、Blue 三个词语的缩写，一个 RGB 颜色值由三个两位十六进制数字组成，分别代表各原色的强度。该强度为从 0 到 255 之间的整数值，如果换算成十六进制值表示，则范围从#00 到#FF。例如 RGB(255,255,255)表示白色，且用#FFFFFF 表示；RGB(0,0,0)表示黑色，且用#000000 表示。

在遵循 HTML4 规范的同时，各大浏览器厂商都扩展了在 HTML4 中预定义的颜色字符串常量，但 RGB 原色表示法可以在所有浏览器中得到正确的显示。在编写需跨浏览器环境工作的 HTML 文档时，要尽量使用 RGB 原色表示法以避免出现兼容性问题。

在 HTML4 中预定义的颜色字符串常量与 RGB 原色值之间存在一定对应关系，如表 8.1 所示。

表 8.1 颜色字符串常量与RGB原色值之间关系

字符串常量	RGB原色值	字符串常量	RGB原色值
aqua	#00ffff	navy	#000080
black	#000000	olive	#808000
blue	#0000ff	purple	#800080
fuchsia	#ff00ff	red	#ff0000
gray	#808080	silver	#c0c0c0
green	#008000	teal	#008080
lime	#00ff00	white	#ffffff
maroon	#800000	yellow	#ffff00

8.2.3 往文档写入新内容

Document 对象提供 open()用于打开新文档以准备执行写入操作，并提供 write()方法和 writeIn()方法用于写入新内容。这些动作完成后，可以使用 close()方法来关闭该文档。

open()方法的语法如下：

```
open([mimeType][,replace])
```

其中参数 mimeType 指定发送到窗口的 MIME 类型，如 text/html、image/jpeg 等。MIME 类型是在 Internet 上描述和传输多媒体数据的规范。在浏览器参数设置的辅助应用程序列表中已简单描述 MIME 类型，它是用斜杠分隔的一对数据类型。将某个 MIME 类型以参数传入 open()方法即是通知浏览器准备接收该类型的数据，接收完成后，浏览器调用其相关功能将该数据显示出来。

各浏览器支持的 MIME 类型会有所区别，一般来说，常见的 MIME 类型有超文本标记语言文本 text/html、普通文本 text/plain、RTF 文本 application/rtf、GIF 图形 image/gif、JPEG 图形 image/jpeg、au 声音文件 audio/basic、MIDI 音乐文件 audio/midi 或 audio/x-midi、RealAudio 音乐文件 audio/x-pn-realaudio、MPEG 文件 video/mpeg、AVI 文件 video/x-msvideo 等。open()方法默认（缺省）的 MIME 类型参数为 text/html，在文档载入浏览器前，使用脚本组合典型的数据类型，包括 HTML 页面上的任何图片。如果当前浏览器不支持以参数传入的特定 MIME 类型，则浏览器搜索支持该 MIME 类型的插入件。如果目标插入件存在，则浏览器装入该插入件，并在文档载入时将要写入的内容传入该插入件。open()方法接收的第二个可选参数 replace 表示待写入的目标文档是否替换浏览器历史列表中当前文档。

> 注意：在 IE3 中 open()方法不接收任何参数，IE4 中仅接收 MIME 类型为 text/html 的参数，IE5+和 NN4+支持第二个可选参数；若使用 open()方法打开文档成功，则该方法返回非 null 值，若由于某种原因打开文档失败，则返回 null 值。

使用 open()方法打开文档后，可调用 write()和 writeIn()方法往其中写入新内容并在浏览器窗口中显示出来。write()和 writeIn()方法本质上并无大的不同，唯一的区别在于后者在发送给文档的内容末尾添加换行符，下面两种表达方式等价：

```
document.write(targetStr+"<br>");
document.writeIn(targetStr);
```

实际上，一旦文档载入窗口或框架完成后，使用 write()和 writeIn()方法可在不重载或不重写整个页面的情况下改变文本节点和文本域 textarea 对象的内容。

一般情况下，脚本在当前文档收集用户输入（或动作）和浏览器环境信息，然后通过一定的算法产生表示另一个窗口（或框架）布局和内容的字符串变量，并使用 write()和 writeIn()

方法将目标字符串变量写入新窗口或者多框架窗口中的某个框架中。该种方式允许使用任意多个 document.write()和 document.writeIn()方法写入任意多个字符串变量，同时可通过一个组合字符串调用这两种方法来写入。实际中采用何种方法写入目标字符串取决于浏览器环境和脚本样式。

在组合字符串过程中，可通过加号“+”或逗号“,”将字符串连接起来，例如下列两种方式均合法：

```
document.write(targetStr+"<br>");
document.write(targetStr,"<br>");
```

考察如下将相关字符串写入框架集中指定框架的实例，演示了“学生注册程序”中如何收集学生相关信息并在目标框架显示。其中框架集文档“main.html”的代码如下：

```
//源程序 8.4
<html>
<head>
  <title>Sample Page!</title>
</head>
<frameset name="MyFrameset" border=3 borderColor="black" cols="60%,40%">
  <frame name="Control" src="leftmain.html">
  <frame name="Display" src="target.html">
</frameset>
</html>
```

其中左框架载入的文档为“leftmain.html”，该文档收集学生信息如姓名、性别、年级和学号等。其代码如下：

```
//源程序 8.5
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
function WriteContent()
{
  var msg="写入的新内容 ：<br><br>";
  //获取"姓名"字段信息
  var iName=document.MyForm.MyName;
  if(iName.value=="")
  {
    alert("''姓名''字段不能为空!");
    iName.focus();
  }
  else
  {
    msg+="姓名 ："+iName.value+"<br>";
  }
  //获取"性别"字段信息
  var iSex=document.MyForm.MySex;
  for(i=0;i<iSex.length;i++)
  {
    if(iSex[i].checked)
      msg+="性别 ："+iSex[i].value+"<br>";
```

```
    }
    //获取"班级"字段信息
    var iClass=document.MyForm.MyClass;
    for(i=0;i<iClass.length;i++)
    {
      if(iClass[i].selected)
        msg+="班级 ："+iClass[i].value+"<br>";
    }
    //获取"学号"字段信息
    var iNum=document.MyForm.MyNum;
    if(iNum.value=="")
    {
      alert("''学号''字段不能为空!");
      iNum.focus();
    }
    else
    {
      msg+="学号 ："+iNum.value+"<br>";
    }
    //写入新内容到右框架
    parent.frames[1].document.write(msg);
    //关闭文档
    parent.frames[1].document.close();
}
//-->
</script>
</head>
<body>
<center>
<br>
<p>学籍注册程序</p>
<form name="MyForm">
    <p>姓名 ：<input type="text" name="MyName" id="MyName" value="ZangPu"></p>
    <p>性别 ：<input type="radio" name="MySex" id="MySex" value="male" checked>male
              <input type="radio" name="MySex" id="MySex" value="female">female</p>
    <p>年级 ：<select name="MyClass" id="MyClass">
                 <option value="class 1" selected>Class 1</option>
                 <option value="class 2" >Class 2</option>
                 <option value="class 3" >Class 3</option>
                 <option value="class 4" >Class 4</option>
                 <option value="class 5" >Class 5</option>
                 <option value="class 6" >Class 6</option>
              </select>
    <p>学号 ：<input type="text" name="MyNum" id="MyNum" value="200104014104"></p>
    </p>
    <p>
    <input type="button" name="MySubmit" value="写新内容进右框架" onclick="WriteContent()">
    </p>
</form>
</center>
</body>
</html>
```

右框架为待写入新内容的目标框架，其载入的文档“target.html”代码为：

```
//源程序 8.6
<html>
<head>
  <title>Sample Page!</title>
</head>
<body>
  <p>待写入的目标框架</p>
</body>
</html>
```

浏览器载入“main.html”文档后，出现如图 8.6 所示的原始页面。

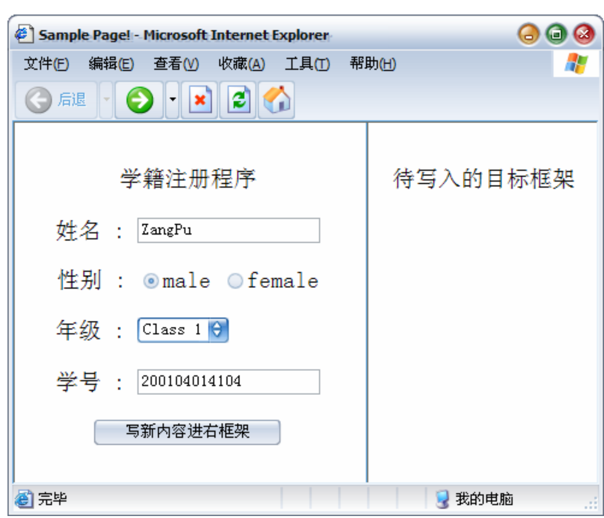

图 8.6 浏览器载入框架集文档后出现的原始页面

在上述原始页面中，单击“写新内容进右框架”按钮，触发左框架文档“leftmain.html”中 WriteContent()函数将通过当前框架文档 MyForm 表单收集的学生信息写入右框架文档中，同时更新页面，如图 8.7 所示。

图 8.7 写入新信息后的框架集文档页面

查看此时右框架所载入文档的源代码，如下所示：

```
写入的新内容 ：<br><br>姓名 ：ZangPu<br>性别 ：male<br>性别 ：class 1<br>姓名 ： 20010401
4104<br>
```

除了可以往窗口（或框架）所载入文档中写入普通字符串外，还可写入 HTML 格式的代码来生成标准的 HTML 文档。修改上述实例中的“leftmain.html”文档中 WriteContent()函数为下述形式：

```
//源程序 8.7
function WriteContent()
{
  var msg="<html><head><title>Sample Page!<\/title><\/head><body>";
  msg="写入的新内容 ：<br><br>";
  //获取"姓名"字段信息
  var iName=document.MyForm.MyName;
  if(iName.value=="")
  {
    alert("\"姓名\"字段不能为空!");
    iName.focus();
  }
  else
  {
    msg+="姓名 ："+iName.value+"<br>";
  }
  //获取"性别"字段信息
  var iSex=document.MyForm.MySex;
  for(i=0;i<iSex.length;i++)
  {
    if(iSex[i].checked)
      msg+="性别 ："+iSex[i].value+"<br>";
  }
  //获取"班级"字段信息
  var iClass=document.MyForm.MyClass;
  for(i=0;i<iClass.length;i++)
  {
    if(iClass[i].selected)
      msg+="班级 ："+iClass[i].value+"<br>";
  }
  //获取"学号"字段信息
  var iNum=document.MyForm.MyNum;
  if(iNum.value=="")
  {
    alert("\"学号\"字段不能为空!");
    iNum.focus();
  }
  else
  {
    msg+="学号 ："+iNum.value+"<br>";
  }
  msg+="<\/body><\/html>"
  //写入新内容到右框架
  parent.frames[1].document.write(msg);
  //关闭文档
  parent.frames[1].document.close();
}
```

单击左框架文档页面中“写新内容进右框架”按钮，将新内容写入右框架文档，其结果不变。但此时右框架所载入文档的源代码发生变化：

```
<html>
<head>
<title>Sample Page!</title>
</head>
<body>
写入的新内容 ：<br><br>姓名 ：ZangPu<br>性别 ：male<br>班级 ：class 1<br>学号 ：
200104014104<br>
</body>
</html>
```

可以看出，该文档为标准的 HTML 文档，但写入 HTML 标记如</head>时要注意写入的格式，如 document.write("<head></head>")语句将会出现错误，因为浏览器会将脚本标记的尾标记解释为进行写操作脚本的结束符。

可通过将尾标记分成几个部分的方式解决，可将上句改写如下：

```
document.write("<head><\/head>");
```

在使用组合字符串进行文档写操作时，容易出现难于觉察的错误，JavaScript 脚本程序初学者应尽量使用多个 write()和 writeIn()方法多次写入相关内容的方式，避免字符串相加时出现错误。

通过 Document 对象的 open()方法或者 write()方法打开一个到目标窗口或框架的输出流并往文档写入新内容的过程结束后，应使用其 close()方法来关闭该输出流。关闭输出流步骤相当重要，如果不关闭，则后续的写入操作会将新内容添加到该输出流对应的文档底部。

当 document.close()方法调用后，往指定窗口或框架添加的部分或全部数据才能正常显示，特别是 open()方法接收的 MIME 类型参数为 GIF 图形 image/gif 或 JPEG 图形 image/jpeg 时，使用 close()方法后图片才能正确显示出来。

8.2.4 常见属性和方法汇总

Document 对象提供一系列属性和方法操作目标文档，其属性分为包含文档附加信息的属性如标记文档背景色的 bgColor 等，以及文档结构模型中作为其属性的对象如表单元素对象 forms 等。表 8.2 列出了 IE 和 NN 浏览器上 Document 对象通用的属性、方法及浏览器版本支持情况。

表 8.2 Document对象常见属性和方法汇总

类型	项目	简要说明	浏览器支持
属性	alinkColor	表示<body>标记的alink属性	NN2+、IE3+
	anchor、anchors	表示文档中所有的锚点对象及形成的数组	NN2+、IE3+
	applet、applets	表示文档中所有嵌入的小程序及形成的数组	NN2+、IE3+
	area	表示文档中包含图形映射区的对象	NN2+、IE3+
	bgColor	表示<body>标记的bgColor属性值	NN2+、IE3+
	compatMode	表示在文档中DOCTYPE元素说明的兼容模式	NN7+、IE6+
	cookie	表示文档的cookie值	NN2+、IE3+
	domain	表示受当前文档信任的域名列表	NN3+、IE4+
	embeds	表示文档中所有插入件形成的数组	NN3+、IE4+
	fgColor	表示<body>标记的fgColor属性值	NN2+、IE3+
	form、forms	表示文档中所有表单对象及它们形成的数组	NN2+、IE3+
	image、images	表示文档中所有图片对象及它们形成的数组	NN3+、IE4+
	lastModified	表示文档最后的修改时间	NN2+、IE3+
	link、links	表示文档中所有链接对象及它们形成的数组	NN2+、IE3+

	linkColor	表示<body>标记的linkColor属性值	NN2+、IE3+
	plugin、plugins	表示文档中所有插入件对象及它们形成的数组	NN4+、IE4+
	referrer	为文档提供一个链接文档的URL	NN2+、IE3+
	title	表示文档的标题栏文本内容	NN2+、IE3+
	URL	表示文档的URL地址	NN3+、IE4+
	vlinkColor	表示<body>标记的vlinkColor属性值	NN2+、IE3+
方法	clear()	清除当前文档的内容	NN2+、IE3+
	close()	关闭用于创建当前文档对象的流	NN2+、IE3+
	createAttribute(attributeStr)	创建新的attribute对象并引用该对象	NN6+、IE6+
	createComment(commentStr)	创建新注释节点对象并引用该对象	NN6+、IE6+
	createElement(elementStr)	创建以参数elementStr为名称的HTML元素	NN6+、IE6+
	createTextNode(nodeStr)	创建以参数nodeStr为名称的文本节点对象	NN6+、IE5+
	exeCommand(commandStr)	执行以参数commandStr标识的命令	NN7+、IE4+
	getElementById(idStr)	根据元素的id属性引用文档中任意元素	NN6+、IE5+
	getElementByName(nameStr)	根据元素的name属性引用文档中任意元素	NN6+、IE5+
	open([mimeType][,replace])	用可选MIME类型的参数mimeType打开用于创建当前文档对象的流，repalce参数用于取代历史清单中的当前文档	NN2+、IE3+
	queryCommandEnabled(str)	显示适合调用的对象是Document还是TextRange	NN7+、IE4+
	queryCommandIndtem(str)	显示命令是否处于不确定状态	NN7+、IE4+
	queryCommandCommandState (str)	显示命令是处于完成状态（true）、正在执行状态（false）、还是不确定状态（null）	NN7+、IE4+
	queryCommandSupported(str)	显示当前浏览器是否支持指定的命令	NN7+、IE4+
	queryCommandText(str)	返回命令执行完毕以结果返回的任何文本	NN7+、IE4+
	queryCommandValue(str)	返回命令执行完毕返回的结果（如果存在）	NN7+、IE4+
	write(expr1[,expr2,…,exprn])	将表达式expr1,expr2,…,exprn写入当前文档	NN2+、IE3+
	writeIn(expr1[,expr2,…exprn])	将表达式expr1,expr2,…,exprn写入当前文档并在结尾加上换行符	NN2+、IE3+

上表中关于元素节点对象的相关内容如createTextNode()、createElement()等方法已在“文档对象模型(DOM)”章节详细叙述，此处不再累述。

Document 对象提供的属性和方法主要用于设置浏览器当前载入文档的相关信息、管理页面中已存在的标记元素对象、往目标文档添加新文本内容、产生并操作新的元素对象等方面。在包含框架集的文档中，要注意 Document 对象引用的层次关系，并通过该层次关系准确定位目标对象并调用 Document 对象的属性和方法进行相关操作。

关于 Document 对象下一层次中的对象如 forms 对象、links 对象、images 对象等将在后续的章节进行专门的叙述。下面讲述与 Document 对象紧密相连的 body 元素对象。

8.3 body 元素对象

Document 对象在 HTML 文档中并没有使用任何显式标记来描述，但 JavaScript 脚本将其作为文档设置的入口，这些设置的内容又作为 body 元素的内在属性在 HTML 中描述。根据 DOM 中的节点模型概念，可将 body 元素对象作为<body>标记的引用。首先考虑如下简单的 HTML 代码：

```
<body bgColor="red" fgColor ="blue">
<p>Contents</p>
<table>
  ...
</table>
<form name="MyForm">
  ...
</form>
```

```
</body>
```

如上述的文档背景颜色 bgColor、文本颜色 fgColor 等均可以作为 body 元素对象的属性来调用；同时，body 标记元素对象也可以作为其他标记元素对象如<table>、<form>等的 HTML 容器而存在。

body 元素对象具有 Document 对象的大部分功能，但不具有 title、URL 等属性，主要包含与文档页面相关的属性和方法，如设置文档背景图片的 background 属性、控制页面滚动条的 scroll 属性等。

8.3.1 获取 body 元素对象信息

如前所述，Document 对象提供了表示文档背景、文本、链接等颜色的属性，body 对象也提供了这几个属性的别名来访问文档中的各项与颜色相关的内容，如 body 元素对象的 aLink 属性、link 属性和 vLink 属性分别对应于 Document 对象的 alinkColor 属性、linkColor 属性和 vlinkColor 属性，而 text 属性则对应于 Document 对象的 fgColor 属性。

考察如下获取目标文档 body 元素对象信息的实例代码：

```
//源程序 8.8
<!DOCTYPE HTML PUBLIC   "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html140/strict.dtd">
<html>
<head>
<meta http-equiv=content-type content="text/html; charset=gb2312">
<title>Sample Page!</title>
<script language="JavaScript" type="text/javascript">
<!--
//获取 body 元素对象信息
function GetInfo()
{
  var msg="\n 获取 body 元素对象信息 ：\n\n";
  msg+="document.body.aLink = " +document.body.aLink+ "\n";
  msg+="document.body.bgColor = " +document.body.bgColor+ "\n";
  msg+="document.body.link = " +document.body.link+ "\n";
  msg+="document.body.text = " +document.body.text+ "\n";
  msg+="document.body.vLink = " +document.body.vLink+ "\n";
  msg+="document.body.background = " +document.body.background+ "\n";
  alert(msg);
}
//-->
</script>
</head>
<body aLink="blue" bgColor="white" link="red" text="black" vLink="purple"
        background="beijing.jpg">
<center>
<br>
<a href="#">文本链接测试</a>
<form name="MyForm">
  <p>
  <input type="button" name="MyGet" value="获取 body 元素对象信息" onclick="GetInfo()">
  </p>
</form>
</center>
```

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

上述代码在<body>标记内初始化了文档背景、文本、链接等颜色及文档的背景图片。程序运行后，出现如图 8.8 所示的原始页面。

图 8.8 显示 body 元素对象信息的原始页面

在原始页面中单击“获取 body 元素对象信息”按钮，触发“GetInfo()”函数，收集当前文档的 body 元素对象的相关信息后，弹出如图 8.9 所示的警告框。

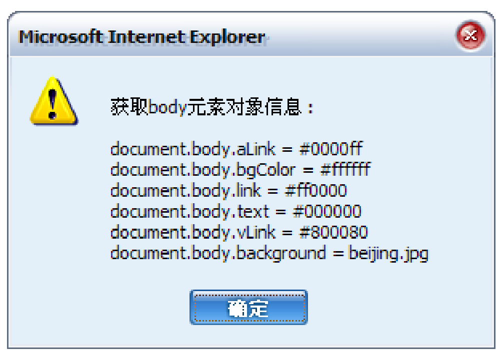

图 8.9 显示当前文档的 body 元素对象相关信息

body 元素对象中与颜色相关的属性如 bgColor、text 等与 body 元素的 HTML 属性等同，而不是与旧属性名相联系。

通过 body 元素对象的 background 属性设置文档页面背景图片后，浏览器使用目标图片覆盖背景颜色，可以通过如下脚本动态去除该背景图片并显示背景色：

```
document.body.background= "";
```

以上列举的属性在 NN6+和 IE4+浏览器上均获得完善的支持。

8.3.2 常见属性和方法汇总

body 元素对象提供的属性和方法为数不多，但提供了快捷设置文档页面的途径。表 8.3 列出了 body 元素对象常见的属性、方法和浏览器版本支持情况。

表 8.3 body元素对象常见属性和方法汇总

类型	项目	简要说明	浏览器版本
属性	aLink	表示文档中文本链接被单击后的颜色	NN6+、IE4+
	background	表示文档的背景图片	NN6+、IE4+

	bgColor	表示文档的背景色	NN6+、IE4+
	bgProperties	标识文档滚动时页面背景图片是固定还是随文档移动，可选值scroll（表示滚动）和fixed（表示固定）	IE4+
	bottomMargin	保存文档内容与浏览器窗口或框架底部的距离	IE4+
	leftMargin	保存文档内容与浏览器窗口或框架左边的距离	IE4+
	link	表示文档中未访问文本链接的颜色	NN6+、IE4+
	rightMargin	保存文档内容与浏览器窗口或框架右边的距离	IE4+
	text	表示文档中文本的颜色	NN6+、IE4+
	topMargin	保存文档内容与浏览器窗口或框架顶部的距离	IE4+
	vLink	表示文档中已访问文本链接的颜色	NN6+、IE4+
	noWrap	标识是否将文档中文本限制在窗口或框架的宽度内，参数为布尔值“true”或“false”	IE4+
	scroll	标识是否隐藏文档的滚动条，参数为布尔型字符串“yes”或“no”	IE4+
	scrollLeft	返回页面左边与水平滚动条左端之间的距离	NN7+、IE4+
	scrollTop	返回页面顶部与垂直滚动条顶部之间的距离	NN7+、IE4+
方法	createControlRange()	返回处于编辑模式下当前文档选定范围内所有控件形成的数组，常规视图下返回空数组	IE5+
	createTextRange()	返回包含body元素的HTML文本和body文本对应的初始TextRange对象	IE4+
	doScroll(ScrollAction)	模拟滚动条上的用户动作，以ScrollAction标识用户的动作，如PageDown、Left、PageUp等	IE5+

在提供上述属性和方法基础上，body 元素对象提供事件处理程序 onscroll 响应用户的动作，如鼠标移动滚动条到底部时调用 Window 对象的 scroll()方法将当前浏览器窗口移动到指定位置等。

8.4 本章小结

本章主要介绍了顶级对象模型中与文档紧密相关的 Document 对象和 body 元素对象，重点讲述了对象模型中 Document 对象和 body 元素对象的层次关系及基础知识，如对象的创建、引用及与其它 HTML 元素对象之间的相互关系等。理解了 HTML 文档中元素对象如 form、image 等都作为 Document 对象层次之下的元素对象而存在，并各自具有独特的属性和方法用于文档界面的设置。

从下章开始，将进入 DOM 层次规范中 Document 对象层次以下的元素对象如 anchor、link 等对象的基本概念的学习，并通过实例让读者熟悉其属性和方法，并深刻体会浏览器版本与它们之间的依存关系。