网页设计尖峰之旅丛书

JavaScript 实例教程

赵丰年 编著

光盘包括微软授权 Windows 2000 Server 免费软件，
本书的 Web 版和全部源代码，大量相关的共享或免费工具软件。

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http:// www.phei.com.cn

网页设计尖峰之旅丛书 JavaScript 实例教程

网站建设尖峰之旅丛书

JavaScript 实例教程

赵丰年　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书通过大量实例介绍了 JavaScript 的基础知识和实际应用，使读者可以按部就班地系统掌握 JavaScript 客户端编程技术。全书共分为 9 章，分别介绍了 JavaScript 语言基础、JavaScript 对象、JavaScript 事件处理、文档对象、窗口与浏览器、表单对象、链接与图像、DHTML 基础以及 DHTML 应用。

本书结构严谨，内容丰富，适合各层次的网页设计人员学习使用，并可以作为相关课程或培训的参考用书。

图书在版编目(CIP)数据

JavaScript 实例教程/赵丰年编著．－北京：电子工业出版社，2001.1
(网站建设尖峰之旅丛书)
ISBN 7-5053-6272-0

Ⅰ.J… Ⅱ.赵… Ⅲ.JAVA 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2000)第 80991 号

丛 书 名：网站建设尖峰之旅丛书
书　　名：JavaScript 实例教程
编 著 者：赵丰年
策　　划：何芳芳
责任编辑：吕　迈　何芳芳
特约编辑：谢　培
排版制作：电子工业出版社计算机排版室
印 刷 者：北京东光印刷厂
出版发行：电子工业出版社　URL：http://www.phei.com.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销：各地新华书店
开　　本：787×1 092　1/16　印张：21.25　字数：541 千字　附光盘：1 张
版　　次：2001 年 1 月第 1 版　2001 年 1 月第 1 次印刷
书　　号：ISBN 7-5053-6272-0 / TP·3384
印　　数：8 000 册　　定价：37.00 元(含光盘)

技术先锋　尖峰之旅

——《网页设计尖峰之旅丛书》与《网站建设尖峰之旅丛书》序

我们有幸生逢技术变革不断发生的伟大时代。几年来，信息领域里最为振奋人心的事情，便是 Internet 的迅速拓展和普及应用。Internet 作为一个蕴含巨大信息资源和人类智慧的网络空间吸引了越来越多的人。今天，Internet 的应用已经与人类生活密切相关，尤其是由信息技术推动和基于 Internet 而兴起的电子商务，已经成为人们普遍接受的一种崭新的商务方式。电子商务能够提供准确、快速、高效的商务环境，能使人们更高效、更省力、更省钱地从事社会和生产活动，代表了当今世界商务模式发展的主流方向。目前，全世界很多国家和地区都在大规模地发展电子商务，以求能为传统商务活动开创新的发展机遇，并为人们提供反应迅速、成本低廉的交易模式。

在电子商务建设中，最关键、最核心的任务是技术平台的建设，其中涉及的技术主要包括三个方面，即面向客户端的网页制作技术，面向服务器端的网站建设和 Web 数据库技术，以及实现客户端与服务器端相互关系的连接和集成技术等。为了帮助广大网页设计和网站建设的专业技术人员快速而深入地掌握这三方面的技术知识，我们组织有关大学教师和资深技术专家编写了“网页设计尖峰之旅”与“网站建设尖峰之旅”两套丛书。

这两套丛书基本上涵盖了当前电子商务建设中的主流技术与软件。其中，“网页设计尖峰之旅丛书”主要侧重于网页设计领域的流行软件和主流编程技术，如《Dreamweaver 3.0 实例教程》、《Fireworks 3.0 实例教程》、《Flash 4.0 实例教程》、《FrontPage 2000 中文版实例教程》、《VBScript 与 JScript 实例教程》、《网页制作实用素材集 I/II》、《Dreamweaver 2.0 实例教程》、《Fireworks 2.0 实例教程》等；“网站建设尖峰之旅丛书”主要侧重于网站建设、Web 数据库系统及网络系统连接与集成领域的流行软件和主流编程语言、开发技术，如《Web 数据库开发技术集成实战演练》、《ASP 实例教程》、《PHP 4 & MySQL 完全实例教程》、《Perl 5、PHP 4 与 CGI 实例教程》等。每一本新书的封底都将有《尖峰之旅丛书》最新成员的身影！

两套丛书紧紧围绕“深入、实用、精炼”的创作主题，以轻松、简练、明快的行文笔调，深入浅出地讲解各书相应的内容，语言流畅、活泼、通俗易懂，举例新颖、实用并易于操作，能使学习者在比较短的时间里快速学习到丰富的软件使用方法和开发技术，收到卓有成效的学习效果，并迅速提高自己的技术开发水平。

两套丛书的特色和风格是一致的，从总体上来讲，主要包括以下 5 个方面：

其一，立足技术前沿。两套丛书所选软件和编程语言、数据库系统等都是国内外著名软件公司的知名产品，也是当前国内应用最新、最广泛的产品。

其二，读者定位明确。丛书中“实例教程”类主要针对电子商务的初、中级专业技术人员，“实战演练”类主要针对电子商务的中、高级专业技术人员。

其三，实例讲解内容。两套丛书均注重通过大量的实例，以实际的技术问题和开发环境，讲解开发案例、编程方法、软件使用技术与操作技巧等。

其四，版式灵活醒目。两套丛书在力求文字精炼、脉络清晰的同时，注意通过大量的图表直观地说明问题，并在正文之外设计了一些特殊的段落，来讲解具有技巧性、提示性或重要性的内容。我们希望通过这种灵活醒目的版式，减小读者的阅读难度，增加学习兴致，真正让学习变成一种乐趣。

其五，随书附带光盘。为了节省读者的学习和练习时间，两套丛书的每一部书都配有光盘。光盘包括3部分内容：多媒体操作指导教程、书中所有举例及例程源代码、相应的系统软件（由出品公司授权）及常用工具软件（共享或免费）。

由于技术在不断地发展，所以我们这两套丛书也采取了不断进取、不断推陈出新的创作与出版思路。也就是说，在已推出各书的基础上，一方面我们要继续创作和出版网络技术领域中其他流行软件和主流编程语言、数据库开发技术的“实例教程”或“实战演练”；另一方面，当软件或语言版本升级时，我们将及时推出更新版本的相应书籍，以保证我们丛书“立足技术前沿”的鼎力追求！

“严谨、实用、高质量、新技术”是我们两套丛书出版中孜孜以求的目标，尽管我们精心而为，刻意而志，但书中错误和不足之处难免，所以恳请广大读者不吝批评指正，多提宝贵意见。我们一定会认真听取读者心声、意见和建议，把后续工作做好，把两套丛书出好！

巅峰风景无限好。紧跟时代潮流，勇攀技术高峰，打造技术先锋，请与我们共赴尖峰之旅！

电子工业出版社

前　言

随着 Internet 的飞速发展，我们已经深深地陷入了“网”中央。在这一“网”无尽的 Internet 海洋中，我们一方面尽力在其中遨游，一方面又渴望亲手编织这张网。在这样的潮流下，不但计算机专业人员争相涌入网页设计和编程这一热门行业，而且许多普通网民也希望通过网络来表现自己甚至挖掘商机。于是，网页设计技术几乎成为一种时尚。

虽然使用 HTML 语言或者 FrontPage、Dreamweaver 等设计工具已可以设计出丰富多采的网页，但如果要创建出功能更强大、更吸引人的网页，就还需要更强大的工具，也就是说，我们必须过渡到网页设计的第二个层次——客户端脚本编程。通过客户端脚本编程，我们可以使网页具有更强的交互性和动态特征，甚至达到与应用程序类似的效果，而 JavaScript 可以说是客户端脚本的最佳选择。一方面 JavaScript 得到绝大多数浏览器的支持，另一方面其类 C 特性也使众多具有 C 语言编程经验的用户能够更快地掌握它。

本书正是在这种背景下应运而生，以帮助读者系统而全面地掌握 JavaScript 客户端脚本编程这一实用技术。全书在逻辑上可以分为三部分：第一部分是前三章，包括“JavaScript 语言基础”、“JavaScript 对象”和“JavaScript 事件处理”，介绍了 JavaScript 语言的基础知识和如何在网页中使用该脚本语言；第二部分是中间四章，包括“文档对象”、“窗口与浏览器”、“表单对象”和“链接与图像”，介绍了各种浏览器对象，使读者可以充分利用它们的属性和方法编写出符合自己需要的网页；第三部分是最后两章，包括“DHTML 基础”、“DHTML 应用”，介绍了 JavaScript 的一个重要的应用领域——DHTML。除了正文之外，本书的两个附录：“HTML4.0 快速参考”和“CSS 属性参考”也为读者制作网页提供了很大的方便。

本书的一个典型特点是其实用性，书中的绝大多数示例都是根据实际需要编写或改编的。例如：导航列表，网页导航条，翻滚图，JavaScript 动画，动态折叠菜单等示例都可以直接应用于读者自己的网页中。书中的所有示例均在 Internet Explorer 5.0 中进行了严格的测试，读者可以放心使用。

本书的读者对象是：了解一般的 Internet 常识，具有 HTML 语言基础的网页设计和编程人员。如果读者具有 C 语言的编程基础，那么有关语言基础的部分学习起来就不会有什么障碍，否则需要详细阅读第 1 章。另外，鉴于国内大多数浏览者使用的都是 Internet Explorer，因此本书默认的目标浏览器是 IE。如果需要在多浏览器平台上使用本书中的示例，则可能重新编写和测试。

建议读者在使用本书的过程中采用边读边实践的方式，这样不但可以学得更有效率，也会学得更有乐趣。

本书由赵丰年编写，参加相关工作的还有胡长清、赵承志、赵念东、刘浩、谢小强、白峰、孙志勇、李伟、吴亚东、孙海、陈熙、李江军、刘东、曹海燕和李涛等。

由于时间仓促，书中疏漏和不妥之处在所难免，希望广大读者谅解。欢迎读者就书中的问题与作者讨论，作者的电子邮件地址是：zhaofengnian @ 263.net。同时欢迎读者访问作者的个人网站(http://zhaofengnian.yeah.net)，在该网站上可以下载本书中的所有源代码和其他一些相关参考资料。

编 著 者

2001 年 1 月

目　录

第 1 章　JavaScript 语言基础

第 1 章　JavaScript 语言基础

JavaScript 是目前最流行的 Web 脚本语言，用于开发动态的 Web 页面。作为一种通用的脚本语言，JavaScript 具有所有相关编程语言的特性，例如：通过变量保存数据，通过运算符进行计算，以及通过函数获得程序的模块化和功能的简化等等。本章介绍有关 JavaScript 作为一种编程语言的基础知识，使读者能初步掌握该语言（具有 C 语言基础的读者会发现该语言与 C 十分类似），以便为进一步的学习打下扎实的基础。

本章主要内容包括：

- 什么是 JavaScript
- 在 Web 页中使用 JavaScript
- JavaScript 变量
- JavaScript 运算符
- JavaScript 语句
- JavaScript 函数

1.1　什么是 JavaScript

JavaScript 是 WWW 上的一种功能强大的编程语言，用于开发交互式的 Web 页面。它不仅可以直接应用于 HTML 文档以获得交互式效果或其他动态效果，而且可以运行于服务器端，从而替代传统的 CGI 程序。

1.1.1　JavaScript 的基本特点

JavaScript 是一种基于对象并具有安全性的脚本语言，具有以下几个基本特点：

- 是一种脚本（script）语言

JavaScript 是一种脚本语言，它采用小程序段的方式实现编程。与其他脚本语言一样，JavaScript 是一种解释性语言，在程序运行过程中被逐行解释。

- 基于对象

JavaScript 是一种基于对象的语言，它的许多功能来自于脚本环境中对象的方法与脚本的相互作用。有关 JavaScript 的对象特性，将从第 2 章开始逐步介绍，并贯穿于全书。

- 安全性

JavaScript 是一种安全性语言（有时称为具有 Web 安全特性），它不允许访问本地的硬盘，也不允许对网络文档进行修改和删除，而只能通过浏览器实现信息浏览或动态交互。

- 跨平台性

JavaScript 的执行依赖于浏览器本身，而与操作环境无关——只要是能运行浏览器的计算机，而该浏览器又支持 JavaScript，则脚本就可正确执行。

1.1.2 JavaScript 与 JScript

对任何一门编程语言略有了解的读者可能都知道：编程语言通常分为各种不同的版本。这种不同版本的划分，要么是基于不同组织对该语言的不同实现，要么是基于同一组织实现的不同升级版本。对于 JavaScript 而言，这种版本区分当然也不例外。

JavaScript 的前身是 Netscape 公司开发的 LiveScript，也就是说，JavaScript 是由 Netscape 开发的。但随着 WWW 在世界范围内的普及，许多其他大公司也步入了这个领域，Microsoft 公司就是其中的一个。与 Internet Explorer 和 Netscape Navigator 的浏览器大战类似，Microsoft 公司在 JavaScript 的开发上也不甘步人后尘，于是独自开发了自己的 JavaScript 版本，通常称为 JScript。

Netscape 在 Navigator 3 中引入了 JavaScript1.1，在 Navigator 4 中引入了 JavaScript1.2，而在 Navigator 4.06 和 4.5 中引入了 JavaScript1.3。对于 Microsoft 公司，最初是在 Explorer 3 中引入了 JavaScript（称为 JScript，与 JavaScript1.2 兼容），后来在 Explorer 4 中升级到 JScript 3.1，而在 Explorer 5 中则升级到 JScript 5。

虽然 JavaScript 与 JScript 是由不同的公司实现的，但幸运的是对于大部分的功能实现，二者几乎没有什么差别。并且，与其他计算机领域的标准类似，JavaScript 的实现也正在走

向统一，Netscape 和 Microsoft 已将各自的脚本实现提交给了总部位于欧洲的一个标准组织 ECMA（欧洲计算机制造商协会），由该协会制定统一的标准。ECMA 于 1997 年 6 月发布了 ECMA-262 标准，并于 1998 年 6 月更新了该标准。ECMA-262 标准规定了 ECMA Script 语言，该语言结合了 Netscape 的 JavaScript 和 Microsoft 的 JScript 的特点，具有更强的适用性。JavaScript1.3 支持 ECMA Script 标准，JScript 3.1 也支持该标准。

根据以上介绍，我们可以得知：JavaScript 和 JScript 并没有本质的差别。因此本书中不区分 JavaScript 与 JScript，而是一律用 JavaScript 表示。也就是说，如果没有特殊说明，所有的内容都是同时适用于 Navigator 和 Explorer 的。虽然如此，由于在某些使用细节上 Microsoft 与 Netscape 的实现截然不同，因此有必要进行选择。鉴于我国绝大多数浏览者均使用 Internet Explorer，本书中凡是涉及二者的不同，通常都是以 Explorer 为例，同时忽略 Navigator 的实现。

1.1.3　JavaScript 应用

JavaScript 虽然与 Java 的名称类似，但二者不论在性质上还是用途上都大不相同：Java 是一种类似于 C++的高级语言，可以实现各种复杂、专业化的应用（例如，著名的电子商务平台 Enfinity 就是基于 Java 技术构建的）；而 JavaScript 则是一种脚本语言，只能实现有限的功能（例如，无法进行文件的读写）。

但是作为一种脚本语言，JavaScript 可以说是非常成功的，它在相当多的领域中得到了广泛应用。它不但可以用于编写 Web 浏览器端（或者称为客户端）脚本，实现在 Web 页面上下文中执行的程序，而且可以在服务器端用于编写可处理浏览器提交的信息并相应地更新浏览器显示的 Web 服务器程序。

1．客户端应用

JavaScript 最典型的应用就是开发客户端 Web 应用程序，也就是开发所谓的客户端脚本。与高级语言不同，客户端脚本程序通常都是解释执行的。也就是说，在执行 JavaScript 脚本之前，无须进行编译等预处理。

在最典型的客户端应用中，JavaScript 脚本程序被嵌入到 HTML 文件中，随着 HTML 文件一同下载到浏览器端。浏览器读 HTML 文件，然后解释执行并显示其中的元素。读取 HTML 文件并分辨其中的元素的过程称为语法分析或解析（parsing）。如果解析到 JavaScript

脚本，则浏览器执行其脚本语句。该过程如图 1.1 所示。

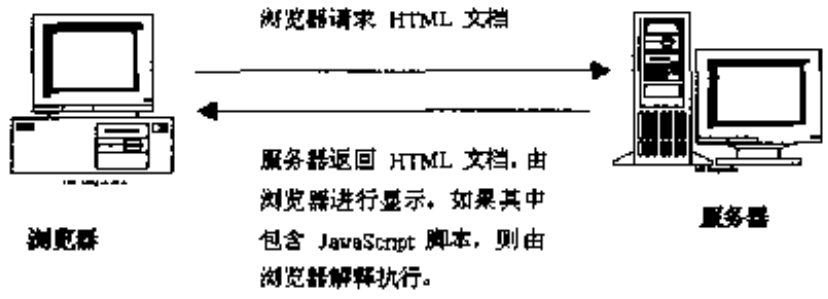


图 1.1　客户端脚本示意图

2. 服务器端应用

JavaScript 脚本不但可以运行于客户端，而且可以运行于服务器端，用于实现服务器端的某些特定功能（例如，取代传统 CGI 程序的表单处理功能）。在 Microsoft 的服务器上，典型的一种应用就是作为 ASP（Active Server Pages，活动服务器页）的实现脚本。

服务器端脚本的工作过程如下：浏览器输入URL请求；服务器调用脚本，生成从浏览器传递数据的对象，并向脚本提供这些对象；脚本进行处理（对于数据库应用则需要进行特定的数据库操作），并将数据以HTML文件的方式通过服务器返回发出请求的浏览器。该过程如图1.2所示。

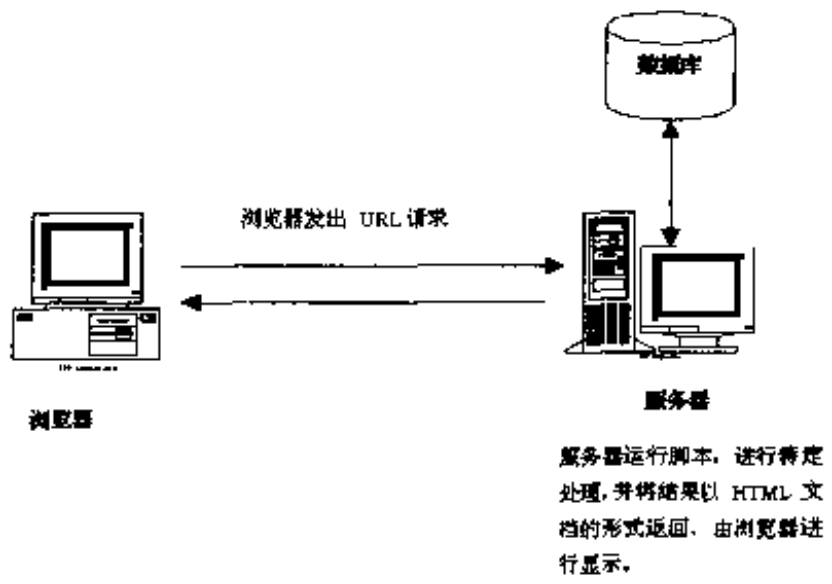


图 1.2　服务器端脚本示意图

说明： 本书的重点在于客户端应用，对于如何编写服务器端脚本程序的详细信息，请读者参考其他书籍。

1.1.4　为什么要使用 JavaScript

虽然前面已经介绍了各种 JavaScript 的特点和用途，但读者可能有疑问——为什么我

一定要学习 JavaScript，难道不能有其他的替代方式？对于有经验的网页设计者来说还会有一个疑问，既然现在 FrontPage、Dreamweaver 等网页设计软件已经可以自动生成 JavaScript 代码以使完成动态 Web 页的制作，那么还有必要学习 JavaScript 吗？

这两个问题实际上代表了大多数网页设计者的疑问，在此做以下说明：

首先，在本章一开始就已经讲到，JavaScript 是一种最流行的 Web 脚本语言，其主要原因在于它具有最广泛的兼容性。实际上，VBScript 也是一种通用的 Web 脚本语言，而且是由软件业的老大——微软开发的，但由于它只能被 IE 浏览器所支持，因此无法与 JavaScript 抗衡。至于 VBScript 以外的其他 Web 脚本语言就更是影响微弱了。试想，连一向颐指气使的微软都忙不迭地开发出自己的 JavaScript 版本——JScript，那么对于 JavaScript 作为一种 Web 脚本语言的统治地位还能有什么疑义呢？因此，如果需要学习一种 Web 脚本语言的话，JavaScript 无疑是首选。

其次，对于是否需要学习一种 Web 脚本语言的回答显然也是一个响亮的“是”。我们先来看一下 Web 页制作工具软件的变迁：早期最流行的 Web 页制作工具无疑是 FrontPage，但近年来最火的软件却变成了 Dreamweaver，而且大有将 FrontPage 逐出市场的意思。对此，最主要的原因有两条：一是 Dreamweaver 提供的面板式界面比 FrontPage 提供的对话框式界面更加适合 Web 页面设计；二是 Dreamweaver 提供了更多的动态特性，尤其是 Dreamweaver 可以与 Fireworks、Flash 等软件紧密集成，从而开发出动感十足、亮丽眩目的 Web 页面。实际上，第二个原因代表了当今 Web 页开发的一种趋势，即不但要注重内容，而且要注重表现形式。因此开发动态 Web 页已经成为网页制作的一个基本要求。如果我们看一下 Dreamweaver 等软件生成的动态 Web 页的 HTML 源代码，会发现其中充满了大量 JavaScript 代码，可见 Dreamweaver 中相当多的动态特性是借助于 JavaScript 实现的。由于 Web 吸引人之处正在于其独特性，而网页制作工具能够提供的自动功能显然是非常有限的（虽然足以应付基本的需要），因此如果要创建出个性化的动态功能，当然必须使用一种 Web 脚本语言。可见，如果要成为一个 Web 页制作高手，掌握一门 Web 脚本语言是基本的要求。

综上所述，学习 JavaScript 势在必行，它将带领我们进入更广阔的 Web 开发世界。

1.2 在 Web 页中使用 JavaScript

实现客户端 JavaScript 应用时，脚本程序既可以直接嵌入到 HTML 文件中，也可以用

链接的方式将脚本程序和 HTML 文件结合起来。

1.2.1 嵌入 JavaScript

无论使用什么脚本语言，最常用的一种方式是在网页中使用 SCRIPT 标记符将脚本语句包含起来，方法是：把脚本标记符<SCRIPT> </SCRIPT>置于网页上的 HEAD 部分或 BODY 部分。尽管可以在网页上的多个位置使用 SCRIPT 标记符，但最好还是将脚本代码放在 HEAD 部分，以确保容易维护。

1. 指定脚本语言

在 HTML 4.0 版以前，SCRIPT 标记符的 language 属性曾被用来规定标记符内的脚本语言是 JavaScript 还是其他类型（如 VBScript、TCL）。在 HTML 4.0 中，W3C 建议使用 type 属性替代 language 属性。

在实际使用时，可以同时包含 language 和 type 这两种属性，以适应不同的浏览器。由于 Web 浏览器会忽略其不懂的标记符和属性，因此当包括了 language 和 type 两个属性时，Web 浏览器会使用其懂得的属性并忽略其不懂的属性。

因此，当使用 JavaScript 编写脚本时，基本语法如下：

```
<SCRIPT Language =“JavaScript” TYPE=“text/javascript” >
 JavaScript 代码
</SCRIPT>
```

网页设计者也可以在 HEAD 标记符内放置 META 标记符，将 META 标记符的 HTTP-EQUIV 属性设置为“Content-Script-Type”，将其 CONTENT 属性设置为“text/javascript”或其他（如“text/vbscript”），则可以设置整个网页的默认脚本语言。

例如，以下语句设置整个网页的默认脚本语言是 JavaScript：

```
<META  http-equiv=“Content-Script-Type”  content=“text/javascript” >
```

2. 其他 language 属性

如果将 language 属性设置为 JavaScript，则所有支持 JavaScript 的浏览器都能处理 JavaScript 代码。但 language 属性也可以设置为下列其他数值，以便限制能处理 JavaScript 代码的浏览器：

- JavaScript1.1　只有支持 JavaScript1.1 的浏览器才能处理 JavaScript 代码，包括

Navigator 3 以上和 Explorer 4 以上。

- JavaScript1.2　只有支持 JavaScript1.2 的浏览器才能处理 JavaScript 代码，包括 Navigator 3 以上和 Explorer 4 以上。
- JavaScript1.3　只有支持 JavaScript1.3 的浏览器才能处理 JavaScript 代码，包括 Navigator 4.06 以上。
- JScript　只有支持 JScript 的浏览器才能处理 JavaScript 代码，包括 Explorer 3 以上。

3. Hello World 示例

几乎学习任何一门编程语言都要做的一个练习就是编写一个“Hello World”程序，以下是这个程序的 JavaScript 版本：

```
<HTML>
<HEAD>
  <TITLE>Hello World</TITLE>
</HEAD>
<BODY>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
  document.write("Hello World!")
</SCRIPT>
</BODY>
</HTML>
```

这个示例用到了 document 对象的 write 方法，有关内容将在以后的章节中详细说明，现在我们只需要知道 document.write("Hello World!")这条语句在浏览器中显示了“Hello World!”字样即可，效果如图 1.3 所示。

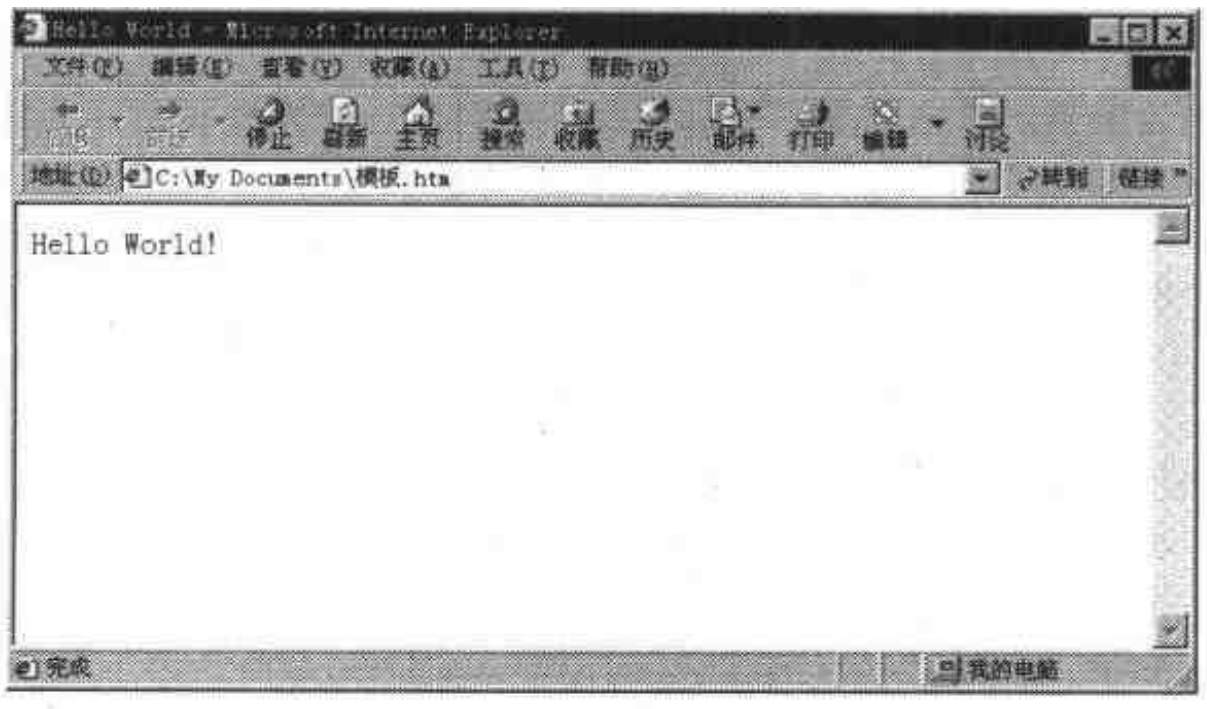


图 1.3　Hello World 程序的显示效果

注意：在 JavaScript 中，语句结束时可以不必键入分号（;），只需要用空行分隔不同的语句即可。当然，对于已经养成在语句后键入分号习惯的读者，也不必改掉这个习惯，因为 JavaScript 也可以用分号作为语句分隔符。而且，为了避免在切换编程语言时忘记了到底什么语言需要分号（C 语言中语句必须用分号结束），什么语言不需要分号，最简单的办法还是一律使用分号。

说明：本书中的所有示例均在 Internet Explorer 5 中进行了严格的测试，如果要在其他的目标浏览器中进行显示，则可能需要具体测试。

4. 响应事件示例

以下的一个示例显示了 JavaScript 应用的另外一种方式——响应事件，代码如下所示：

```
<HTML>
<HEAD>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
  function showdate()
  {  alert(Date())  }
</SCRIPT>
<TITLE>响应事件示例</TITLE>
</HEAD>
<BODY>
<FORM>
   <INPUT  TYPE="Button"  onClick="showdate();"  VALUE="显示时间">
</FORM>
</BODY>
</HTML>
```

这个示例的效果是：单击“显示时间”按钮后弹出一个对话框显示当前系统时间，如图 1.4 所示。

另外，由于事件响应脚本的语句只有一句，因此也可以采用直接嵌入的方式，如下所示：

```
<HTML>
<HEAD>
  <TITLE>直接嵌入脚本</TITLE>
</HEAD>
<BODY>
```

```
<FORM>
   <INPUT TYPE="Button" onClick="alert(Date());" VALUE="显示时间">
</FORM>
</BODY>
</HTML>
```

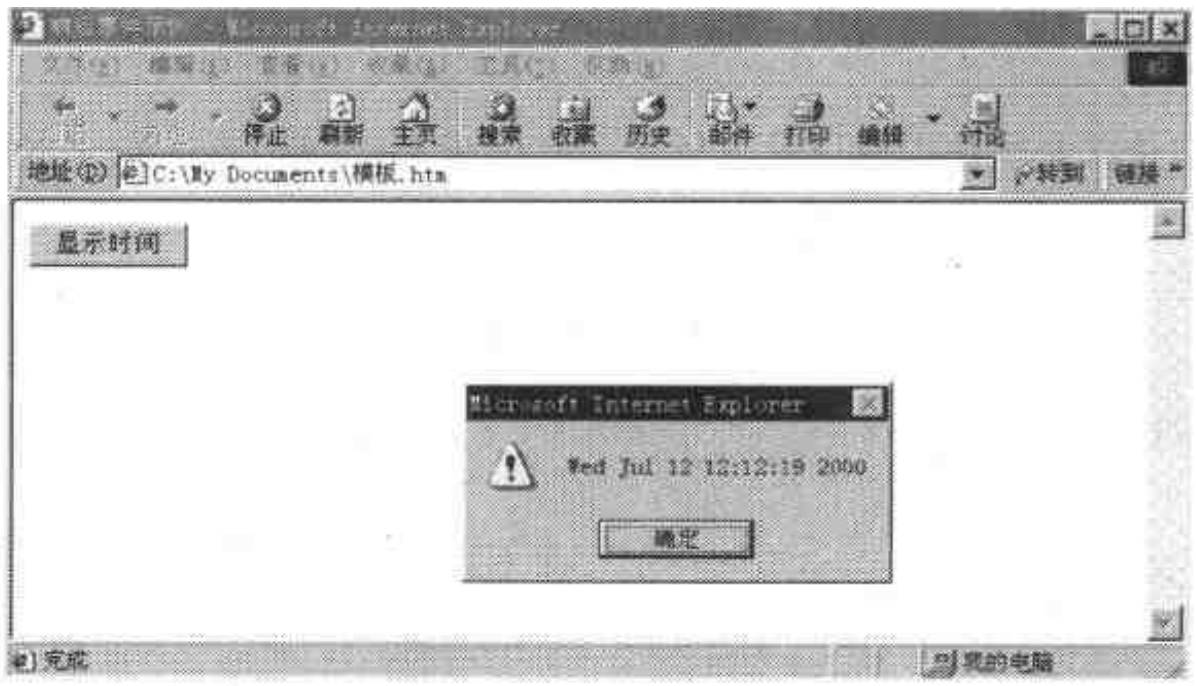


图 1.4 响应事件示例

说明：有关事件响应的详细信息，请参见本书第 3 章。

1.2.2 链接 JavaScript

如果同一段脚本可以在若干个 Web 页中使用，则没有必要在多处维护相同的冗余代码，此时可以将脚本放在单独的一个文件里，然后再从任何需要该文件的 Web 页中引用该文件。

要引用外部脚本文件，应使用 SCRIPT 标记符的 SRC 属性来指定外部脚本文件的 URL。通过使用这种方式，可以使脚本得到复用，从而降低了维护的工作量。如果使用 SCRIPT 标记符的 SRC 属性，则 Web 浏览器只使用在外部文件中的脚本，并忽略任何位于 SCRIPT 标记符之间的脚本。

例如，以下示例显示了如何链接脚本文件，该示例的效果与图 1.4 一模一样。

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript" src="test.js">
</SCRIPT>
<TITLE>JavaScript 示例</TITLE>
</HEAD>
```

```
<BODY>
<FORM>
   <INPUT  TYPE="Button"  onClick="showdate();"  VALUE="显示时间">
</FORM>
</BODY>
</HTML>
```

需要注意的是，必须在当前目录下包含 test.js 文件。可以直接在文本编辑器中键入以下内容：

```
function showdate(){alert(Date())}
```

然后将文件保存为 test.js。

1.2.3 确保兼容性

虽然大多数浏览器都支持 JavaScript，但一些低版本的浏览器（如 Internet Explorer 2）或基于字符的 Lynx 浏览器并不能识别 SCRIPT 标记，此时它们会将 SCRIPT 标记中包括的内容显示为字符串。为了避免这种情况的出现，我们可以用 HTML 的注释语句将 JavaScript 脚本语句包括起来，从而使不支持脚本的浏览器能忽略脚本内容。

另外，在使用 JavaScript 时，HTML 注释标记符的结束标记之前通常有两道斜杠（//）。这两道斜杠是 JavaScript 语言中的注释，需放置在注释标记符的前面。如果没有这两道斜杠，JavaScript 解释器会试图将 HTML 注释的结束标记符作为 JavaScript 来解释，从而有可能导致出错。

说明： JavaScript 语言的注释标记除了可以使用“//”外，还可以使用“/*”和“*/”。使用“//”时，JavaScript 解释器将把该行其后的内容都作为注释；使用“/*”和“*/”时，则将所有包括在其中的内容作为注释，不论这些内容是多行，还是只是一行中的一部分。

因此，为了确保不支持 JavaScript 的浏览器不至于把脚本语句作为字符串显示，正确的使用方法如下：

```
<SCRIPT Language = “JavaScript” TYPE= “text/javascript” >
<!--
JavaScript 代码
//-->
</SCRIPT>
```

另外，在 HTML 4.0 中，还提供了一个 NOSCRIPT 标记，用于在其中包含替换脚本的内容。对于不支持 JavaScript 的浏览器或者设置为不执行 JavaScript 的浏览器，将显示位于<NOSCRIPT>和</NOSCRIPT>标记符之间的内容。

例如，将前面的 Hello World 程序改编为具有最广泛兼容性的版本如下，这样不论在什么浏览器中都可以正确地进行显示。

```
<HTML>
<HEAD>
  <TITLE>Hello World</TITLE>
</HEAD>
<BODY>
<SCRIPT Language =“JavaScript” TYPE=“text/javascript”>
<!--
  document.write("Hello World!")
//-->
</SCRIPT>
<NOSCRIPT>
  This is a "hello world" program using JavaScript!
</NOSCRIPT>
</BODY>
</HTML>
```

1.3 JavaScript 变量

与其他编程语言一样，JavaScript 也是采用变量存储数据。所谓变量，就是程序中一个已命名的存储单元。变量的主要作用是存取数据和提供存放信息的容器。对于变量必须明确变量的命名、变量的类型以及变量的作用域。

1.3.1 变量的命名约定

JavaScript 中的变量命名与其他计算机语言非常相似，包含以下几个要点：

- 变量名必须以大写字母（A 到 Z）、小写字母（a 到 z）或下划线（_）开头，其他的字符可以用字母、下划线或数字（0 到 9）。变量名称中不能有空格、+、一等其他符号。
- 不能使用 JavaScript 中的关键字作为变量名。在 JavaScript 中定义了多个关键字，

这些关键字是 JavaScript 内部使用的，不能作为变量的名称。例如 var、int、double、true 等都不能作为变量的名称。

- 在对变量命名时，最好把变量名的意义与其代表的内容对应起来，以便能方便地区分变量的含义。例如，today 这样的变量就很容易让人明白其代表的内容。
- JavaScript 变量名是区分大小写的，因此在使用时必须确保大小写相同。不同大小写的变量，例如 sum、Sum、SUM，将被视为不同的变量。
- JavaScript 变量命名约定与 Java 类似，也就是说，对于变量名为一个单词的，则要求其为小写字母，例如 money；对于变量名由两个或两个以上的单词组成，则要求第二个和第二个以后的单词的首字母为大写，例如，theDate、theOtherDay。

1.3.2 变量的类型

与 Java 和其他一些高级语言（例如 C 语言）不同，JavaScript 并不要求指定变量中包含的数据类型，这种特性通常使 JavaScript 被称为弱类型的语言。

1. 使用变量

在 JavaScript 中，我们可以简单地用 var 来定义所有的变量，而不论将在变量中存放什么类型的数值。实际上，变量的类型由赋值语句隐含确定。例如，如果赋予变量 money 数字值 1000，则 money 可参与整型操作；如果赋予该变量字符串值"This is my money"，则它可以参与字符串操作；同样，如果赋予它逻辑值 false，则它可以支持逻辑操作。

不但如此，变量还可以先赋予一种类型的数值，然后再根据需要赋予其他类型的数值。例如，在以下示例中，变量 today 先被赋予了数字值 15，然后又将一个字符串值赋予该变量：

```
<SCRIPT>
var today=15;
today="Today is the 15th";
</SCRIPT>
```

说明： 变量可以在声明时直接赋值，如上所示。也可以声明之后再赋值，例如：

```
<SCRIPT>
var today;
today=15;
```

```
</SCRIPT>
```

另外，在 JavaScript 中也可以事先不声明一个变量而直接使用，这时 JavaScript 会自动声明该变量。不过使用这种方法常常会引起混乱，这涉及到变量作用域的问题，详细信息请参见 1.3.3 节“变量的作用域”。

JavaScript 支持的数据类型如下：

- Number（数字） 包括整数和浮点数以及 NaN（非数）值，数字用 64 位 IEEE 754 格式。
- Boolean（布尔） 包括逻辑值 true 和 false。
- String（字符串） 包括单引号或双引号中的字符串值。
- Null（空） 包括一个 null 值，定义空的或不存在的引用。
- Undefined（未定义） 包括一个 undefined 值，表示变量还没有赋值，也就是还没有被赋予任何类型。
- Object（对象） 包括各种对象类型，例如数组类型 Array，日期对象 Date 等。有关对象类型的信息，将在第 2 章中介绍。

注意：undefined 值是 ECMA Script 规范引入的，不符合该规范的浏览器不支持，包括 Navigator 4.05 之前和 Explorer 3。

2. 数值

对于数值，JavaScript 支持整数和浮点数。当数值表达式中混合使用不同类型的数值时，它透明地将一种类型转换成另一种类型。例如，整数用在浮点表达式中将被转换为浮点型。

整数在 JavaScript 中可以使用十进制、十六进制和八进制表示其值。

- 十进制 十进制数是我们日常使用的数，每位用 0~9 的数字表示，权为 10。例如，100，1234。
- 十六进制 十六进制数是以 16 为权进行进位，0~9 仍然用 0~9 表示，10~15 分别用 A、B、C、D、E、F 表示。在 JavaScript 中，十六进制数的前两位必须是 0X 或 0x。例如，0xff（相当于十进制的 255）、0X400（相当于十进制的 1024）。
- 八进制 八进制数以 8 为权进行进位，每位用 0~8 的数字表示。在 JavaScript 中，八进制数的第一位必须是 0。例如，0377（相当于十进制的 255））、02000（相

当于十进制的 1024）。

浮点数由整数部分加小数部分组成，或者由整数部分加指数部分组成。例如，以下数值都是有效的浮点数：

- -12.32
- 100.
- 5E7
- 1e-2
- -4e-4
- .75

下面的示例显示了如何在 JavaScript 中使用整数和浮点数，显示效果如图 1.5 所示。

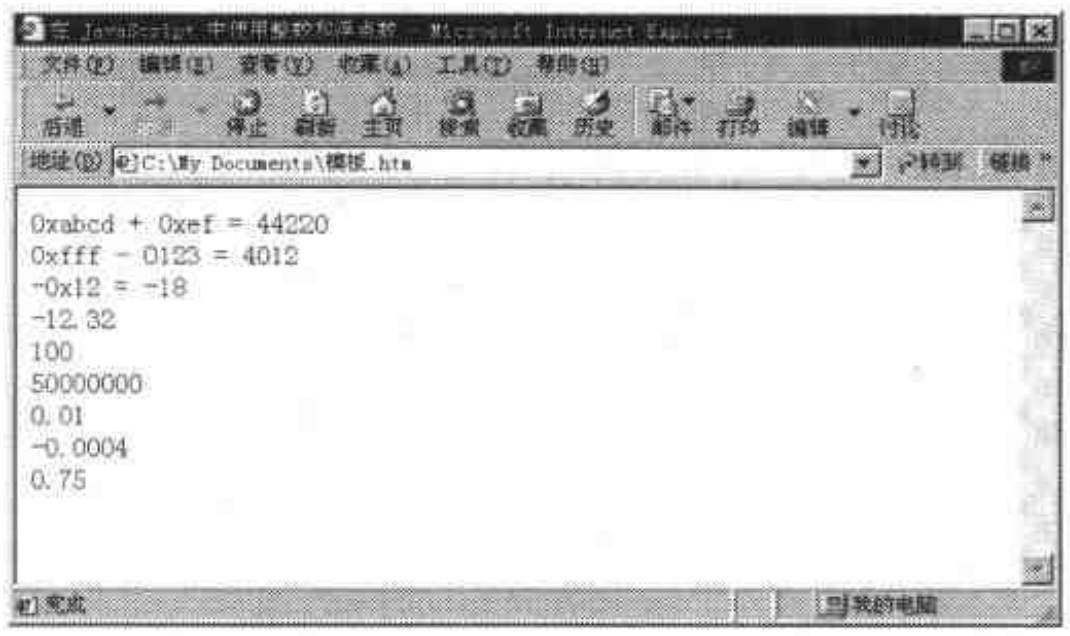


图 1.5　在 JavaScript 中使用整数和浮点数

```
<HTML>
<HEAD>
  <TITLE>在 JavaScript 中使用整数和浮点数</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
document.write("0xabcd + 0xef = ")
document.write(0xabcd + 0xef)  //注意在浏览器中所有的结果都用十进制显示
document.write("<BR>");
document.write("0xfff - 0123 = "); document.write(0xfff - 0123);
/*在同一行中，可以用分号分隔不同的 JavaScript 语句。为了节省篇幅，本书中会用到这种
.方法。*/
document.write("<BR>");
document.write("-0x12 = "); document.write(-0x12);
```

```
document.write("<BR>");
document.write(-12.32); document.write("<BR>");
document.write(100.); document.write("<BR>");
document.write(5E7); document.write("<BR>");
document.write(1e-2); document.write("<BR>");
document.write(-4e-4); document.write("<BR>")
document.write(.75)
// -->
</SCRIPT>
</BODY>
</HTML>
```

3. 布尔值

布尔（Boolean）值也称逻辑值，由 true（真）和 false（假）两个值构成。布尔值主要用来说明或代表一种状态或标志，以控制操作流程。布尔值之间可以进行逻辑操作，构成布尔表达式。有关逻辑运算符的详细信息，将在 1.4.3 节中说明。

在 JavaScript 中只能用 true 或 false 表示其状态，而不能用 1 或 0 表示其状态（但在 C++ 中可以这样表示）。但当布尔值用于数值表达式时，true 和 false 将自动转换成 1 和 0，如以下示例所示。

```
<HTML>
<HEAD>
  <TITLE>布尔值转换为数值</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<! -
document.write("true*100 + false*10 + true*1 = ")
document.write(true*100 + false*10 + true*1)
// -->
</SCRIPT>
</BODY>
</HTML>
```

此示例在浏览器中的显示效果如图 1.6 所示。

4. 字符串

字符串是指使用单引号(')或双引号(")括起来的一个或几个字符，例如："This is a book of JavaScript"、'3245'、"ewrt234234" 等。如果字符串是以双引号开头，则必须以双引号结

束，同样，单引号也必须配对出现。

图 1.6　布尔值转换为数值

与C语言一样，在 JavaScript 中也以反斜杠（\）开头定义了若干个不可显示的特殊字符，这些字符通常称为控制字符。另外，如果要在字符串中使用反斜杠，或者在单引号括起来的字符串中使用单引号以及在双引号括起来的字符串中使用双引号，则都需要使用反斜杠作为转义符。

JavaScript 中的这些特殊字符如表 1.1 所示。

表 1.1　JavaScript 中的特殊字符

字　符	含　义	字　符	含　义
\'	单引号	\r	回车
\"	双引号	\f	进纸
\\	反斜杠	\t	水平制表符
\n	新行	\b	退格

以下示例显示了如何在字符串中使用这些特殊字符，效果如图 1.7 所示。需要注意的是，浏览器并不支持某些控制字符（此时显示为空格）。为了防止所有格式控制符被当作空格，我们使用了预格式化标记<PRE>。

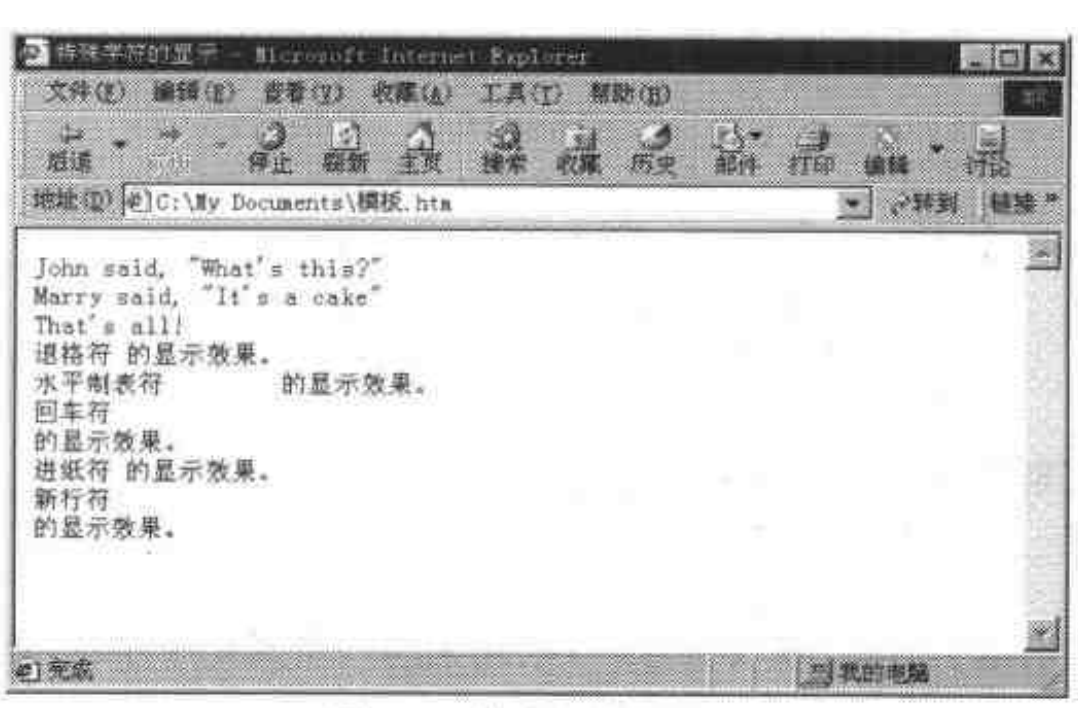


图 1.7　特殊字符的显示

```
<HTML>
<HEAD>
  <TITLE>特殊字符的显示</TITLE>
</HEAD>
<BODY>
<PRE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
document.write("John said, \"What's this?\"\n")
document.write('Marry said, "It\'s a cake"\n')
document.write('That\'s all!\n')
document.write("退格符\b 的显示效果。\n")
document.write("水平制表符\t 的显示效果。\n")
document.write("回车符\r 的显示效果。\n")
document.write("进纸符\f 的显示效果。\n")
document.write("新行符\n 的显示效果。\n")
// -->
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

5. 类型转换

在表达式中使用变量时，JavaScript 自动将一种数值类型转换为另一种。也就是说，如果表达式中使用了不同类型的变量，则 JavaScript 会自动完成使表达式有意义的类型转换。

例如，在以下示例中，执行加法运算时，数字值被转换成了字符串值；而执行乘法运算时，字符串值转换为 NaN（非数）。该示例的显示效果如图 1.8 所示。

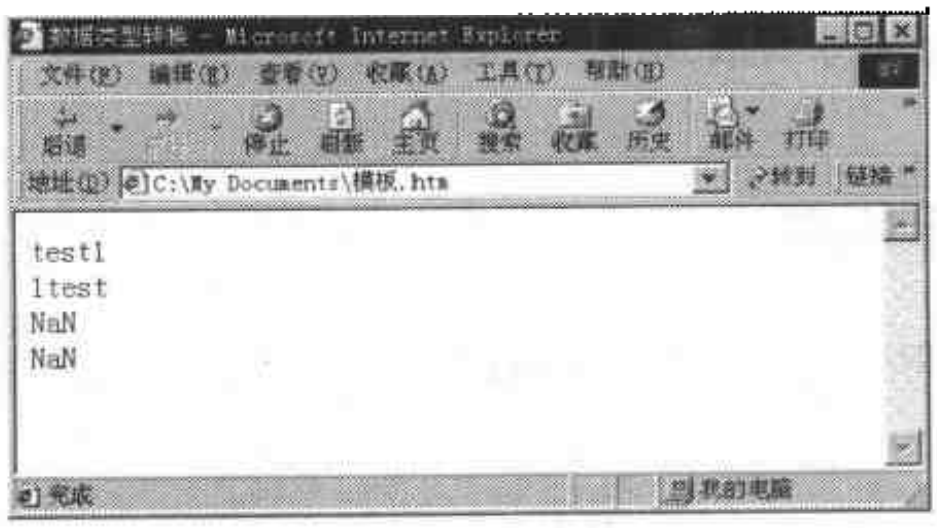


图 1.8 数据类型转换

```
<HTML>
<HEAD>
  <TITLE>数据类型转换</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var myString="test";
var num=1;
document.write(myString+num);  document.write("<BR>");
document.write(num+myString);  document.write("<BR>");
document.write(myString*num);  document.write("<BR>");
document.write(num*myString);
// -->
</SCRIPT>
</BODY>
</HTML>
```

说明： 在进行 JavaScript 表达式求值时，首先按照操作的优先顺序将表达式分解成最基本的一元或二元表达式求值，然后逐步合并。例如，表达式 a+b*c 就是先计算 b*c，然后计算加法。每个表达式根据涉及的运算符求值，如果运算符涉及不同类型的运算数，则相应运算数自动转换为对该运算符有效的类型。对于可应用于多种类型的运算符，则根据优先顺序进行转换。在 JavaScript 中，通常字符串运算优先级较高，然后依次是浮点运算、整型运算和逻辑运算。

在以上示例中，由于“+”操作符既可以进行数字运算，也可以进行字符串运算，因此按优先顺序进行字符串运算；而“*”操作符只能对数值进行操作，因此字符串值被转换为 NaN（非数），数字与 NaN 相乘，结果当然还是 NaN。

1.3.3 变量的作用域

变量的作用域是指变量在其中起作用的范围，也就是说，变量的作用域由变量在其中是“可见”的那部分程序构成。所谓“可见”，就是指可以在该范围内引用该变量。

在 JavaScript 中变量分为全局变量和局部变量。全局变量定义在所有函数体之外，其作用范围是整个脚本；而局部变量是定义在函数体之内，只对该函数是可见的，而对其他

函数则是不可见的。如果一个全局变量和一个局部变量同名，那么在该局部范围内的变量引用是指局部变量，而局部变量范围以外的变量引用则是指全局变量。如果事先没有定义就使用一个变量，那么 JavaScript 将把该变量作为全局变量。

例如，在以下示例中，函数 double 中定义了局部变量 x，同时在文档主体中未经定义地使用了 x（因此作为全局变量），而这两个变量名并没有发生冲突。此示例的显示效果如图 1.9 所示（有关函数以及程序控制语句的详细信息，请参见本章后面的小节）。

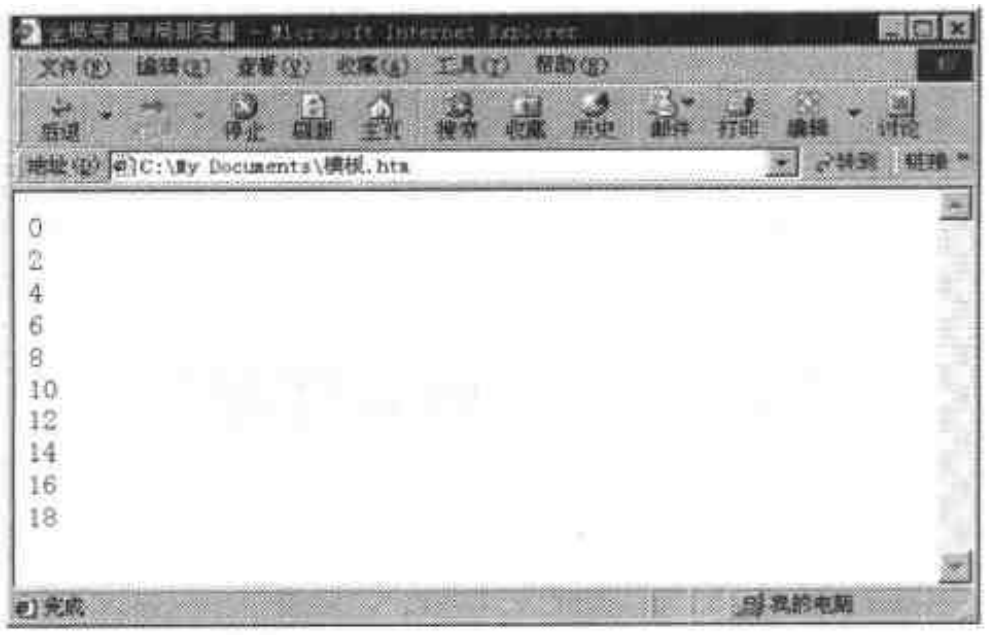


图 1.9 全局变量与局部变量

```
<HTML>
<HEAD>
<TITLE>全局变量与局部变量</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function double(y) {
var x = 2 * y
document.write(x+"<BR>")
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
for(x=0;x<10;++x)
 double(x);
// -->
</SCRIPT>
</BODY>
```

```
</HTML>
```

说明： 由于在 JavaScript 中不需要显式地定义变量，未经定义的变量自动作为全局变量，因此在使用变量时要非常小心，以免由于拼写错误或大小写错误导致严重的后果。例如，原来需要引用全局变量 today，却由于输入了 Today 而新建了一个全局变量，这显然会导致不可预知的结果。避免出现这种问题的一种办法是：对所有的变量（不论是全局变量还是局部变量）都在特定位置进行显式定义。这样，当在程序中需要引用某个变量时，就可以到变量定义处查找，从而避免了变量名拼写错误的问题。这种方法对于比较大的应用程序而言十分有用。

1.4 JavaScript 运算符

1.4.1 运算符与表达式

1. 运算符

运算符是完成操作的一系列符号，也称为操作符。运算符用于将一个或几个值变成结果值，使用运算符的值称为算子或操作数。

在 JavaScript 中包括以下 8 类运算符：

- 算术运算符
- 逻辑运算符
- 比较运算符
- 字符串运算符
- 位操作运算符
- 赋值运算符
- 条件运算符
- 其他运算符

除了条件运算符是三目运算符以外，JavaScript 中其他的运算符要么是双目运算符，要么是单目运算符。

说明： 单目、双目、三目或多目运算符也称为单元、二元、三元或多元运算符。

大多数 JavaScript 运算符都是双目运算符，即具有两个操作数的运算符，通常用以下方式进行操作：

操作数 1 运算符 操作数 2

例如，50＋40、"This"+"that" 等。

双目运算符包括：+（加）、-（减）、*（乘）、/（除）、%（取模）、|（按位或）、&(按位与)、<<（左移）、 >>（右移）、>>>（右移，零填充）等。

单目运算符是只需要一个操作数的运算符，此时运算符可能在运算符前或运算符后。单目运算符包括：-（单目减）、!（逻辑非）、~（取补）、++（递加 1）、--（递减 1）等。

2. 表达式

表达式是运算符和操作数的组合。表达式通过求值确定表达式的值，这个值是对操作数实施运算符所确定的运算后产生的结果。有些运算符将数值赋予一个变量，而另一些运算符则可以用在其他表达式中。

由于表达式是以运算符为基础的，因此表达式可以分为算术表达式、字符串表达式、赋值表达式以及逻辑表达式等等。

表达式是一个相对的概念，例如，在表达式 a = b + c * d 中，c * d、b + c * d、a = b + c * d、以至于 a、b、c、d 都可以看作是一个表达式。在计算了表达式 a = b + c * d 之后，表达式 a、表达式 b + c * d 和表达式 a = b + c * d 的值都等于 b + c * d。

1.4.2 算术运算符

JavaScript 支持的算术运算符如表 1.2 所示。

表 1.2 JavaScript 算术运算符

运 算 符	说 明
+	加
-	减或单目减
*	乘
/	除
%	取模，即计算两个整数相除的余数。例如，10%3 = 1
++	递加 1 并返回数值或返回数值后递加 1，取决于运算符的位置在操作数前还是后
--	递减 1 并返回数值或返回数值后递减 1，取决于运算符的位置在操作数前还是后

以下示例演示了 JavaScript 中算术运算符的应用，效果如图 1.10 所示。

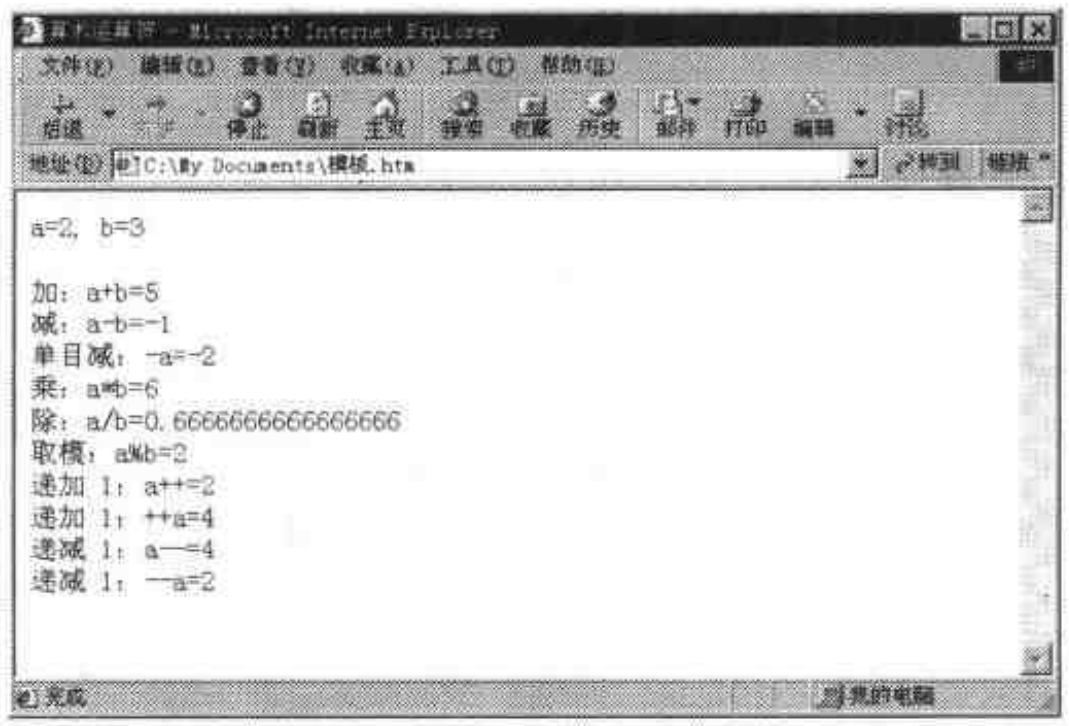


图 1.10　算术运算符

```
<HTML>
<HEAD>
  <TITLE>算术运算符</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a=2;
var b=3;
document.write("a=2, b=3<P>");
document.write("    加    :    a+b=");          document.write(a+b);
document.write("<BR>");
document.write("    减    :    a-b=");          document.write(a-b);
document.write("<BR>");
document.write("  单  目  减  :   -a=");         document.write(-a);
document.write("<BR>");
document.write("    乘    :    a*b=");          document.write(a*b);
document.write("<BR>");
document.write("    除    :    a/b=");          document.write(a/b);
document.write("<BR>");
document.write("   取   模   :   a%b=");         document.write(a%b);
document.write("<BR>");
document.write("递加 1: a++=");
document.write(a++);  /*此语句如果用 document.write(b=a++) 代替，结果一模
一样，请参见稍后的说明。*/
```

```
document.write("<BR>");
document.write("递加 1: ++a=");
document.write(++a);    /*此语句可用 document.write(b=++a) 代替。*/
document.write("<BR>");
document.write("递减 1: a--=");
document.write(a--);    /*此语句可用 document.write(b=a--) 代替。*/
document.write("<BR>");
document.write("递减 1: --a=");    /*此语句可用 document.write(b=--a) 代
替。*/
document.write(--a);
// -->
</SCRIPT>
</BODY>
</HTML>
```

说明：对于没有 C 语言基础的读者来说，图 1.10 中最后四行的结果十分令人费解。实际上，对于递加运算符++，如果它放在操作数之后，其含义是：先返回操作数的数值，然后将操作数加 1；如果它放在操作数之前，其含义是：先将操作数加 1，然后返回操作数的值。对于递减运算符，将其放在操作数之前或之后的效果与递加运算符类似。

于是，在图 1.10 中，计算 a++ 时，先将 a 的值（此时为 2，因为前面的任何运算都没有更改这个初始值）作为表达式的值返回，然后将 a 加 1（此时 a=3），因此 a++这个表达式的值就是 2，而计算了这个表达式之后 a 的值为 3；接着计算表达式++a，此对先将 a 的值加 1（3+ 1=4），然后将 a 的值返回，因此++a 这个表达式的值是 4，同时 a 的值也是 4（用于下一轮的计算）。

如果按照注释行中的说明进行替代，则计算了 b=a++之后，b=2（也就是先计算该表达式的值为 a，然后将 a 加 1），a=3；计算了 b=++a 后，b=a=4；计算了 b=a--后，b=4，a=3；最后计算 b=--a，结果是 b=a=2。

1.4.3 比较运算符与逻辑运算符

1. 比较运算符

比较运算符的基本操作过程是，首先对它的操作数进行比较，然后返回一个布尔值 true 或 false。在 JavaScript 中有 8 个比较运算符，如表 1.3 所示。

表 1.3 JavaScript 比较运算符

运 算 符	说 明
<	小于
<=	小于等于
>	大于
>=	大于等于
==	等于，此运算符先进行类型转换再测试是否相等。例如，"10"==10 的值为 true
===	严格等于，此运算符不进行类型转换直接测试是否相等。例如，"10"===10 的值为 false
!=	不等于，此运算符先进行类型转换再测试是否相等。例如，"10"!=10 的值为 false。
!==	严格不等于，此运算符不进行类型转换直接测试是否相等。例如，"10" !== 10 的值为 true

说明： 严格等于（===）和严格不等于（!==）运算符是 ECMAScript 标准中的运算符，因此只在 Navigator 4.06 以上和 IE4 以上的浏览器中支持。

以下示例演示了 JavaScript 中比较运算符的应用，效果如图 1.11 所示。

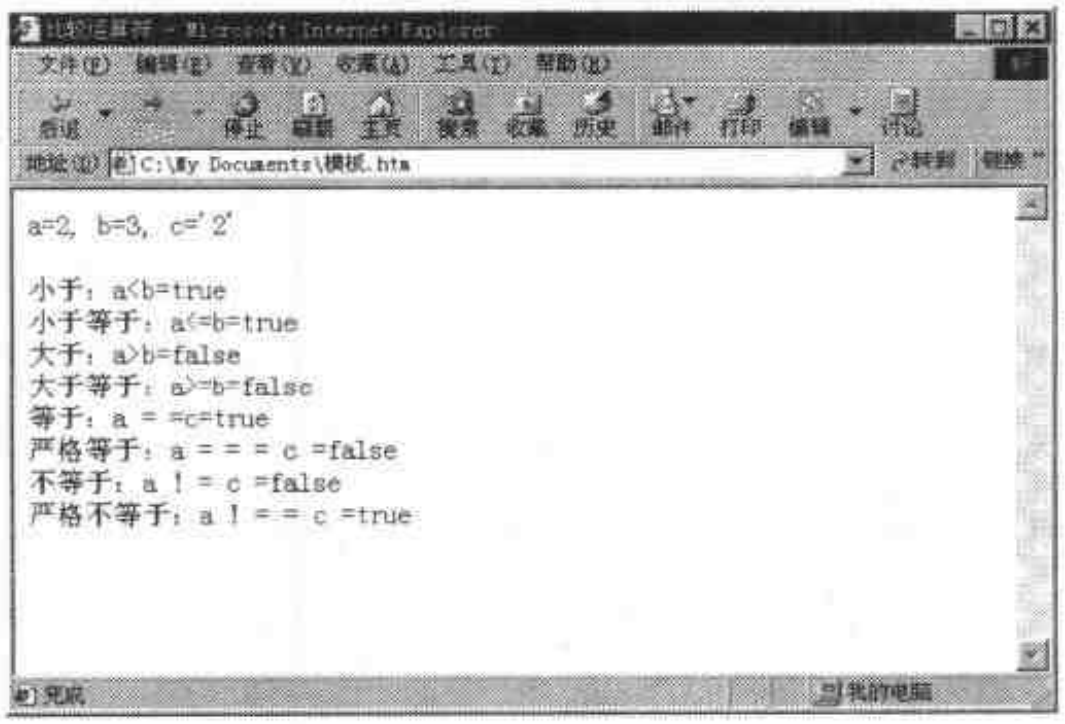


图 1.11 比较运算符

```
<HTML>
<HEAD>
  <TITLE>比较运算符</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a=2;
var b=3;
var c="2";
document.write("a=2, b=3, c='2'<P>");
document.write("小于: a&lt;b=");  /*在网页中显示 < 号这样的特殊字符需要采用
```

字符实体的形式，详细信息请参阅其他有关 HTML 的参考书。*/

```
document.write(a<b);    document.write("<BR>");
document.write("小于等于: a<=b=");  document.write(a<=b);
document.write("<BR>");
document.write("大于: a>b=");  document.write(a>b);
document.write("<BR>");
document.write("大于等于: a>=b=");  document.write(a>=b);
document.write("<BR>");
document.write("等于: a = =c=");  document.write(a == c);
document.write("<BR>");
document.write("严格等于: a = = = c =");  document.write(a === c);
document.write("<BR>");
document.write("不等于: a ! = c =");  document.write(a != c);
document.write("<BR>");
document.write("严格不等于: a ! = = c =");  document.write(a !== c);
// -->
</SCRIPT>
</BODY>
</HTML>
```

2. 逻辑运算符

JavaScript 中包括 3 个逻辑运算符，如表 1.4 所示。

表 1.4 JavaScript 逻辑运算符

运 算 符	说 明
&&	逻辑与，只有当两个操作数的值都为 true 时，a && b 的值才为 true
||	逻辑或，只要两个操作数中其中之一的值为 true，a || b 的值就为 true
!	逻辑非，!true 的值为 false，!false 的值为 true

逻辑运算符用于对逻辑操作数进行逻辑运算，而最典型的逻辑操作数就是条件表达式。以下示例显示了逻辑运算符与条件表达式共同使用的情况，效果如图 1.12 所示。

```
<HTML>
<HEAD>
  <TITLE>逻辑运算符</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a=2;
var b=3;
```

```
document.write("a=2, b=3<P>");
document.write("逻辑与: a&lt;b&&a<=b = "); document.write(a<b&&a<=b);
document.write("<BR>");
document.write("逻辑与: a&lt;b&&a>b = "); document.write(a<b&&a>b);
document.write("<BR>");
document.write("逻辑或: a&lt;b||a>b = "); document.write(a<b||a>b);
document.write("<BR>");
document.write("逻辑或: a>b||a>=b = "); document.write(a>b||a>=b);
document.write("<BR>");
document.write("逻辑非: !(a&lt;b) = "); document.write(!(a<b));
document.write("<BR>");
document.write("逻辑非: !(a>b) = "); document.write(!(a>b));
// -->
</SCRIPT>
</BODY>
</HTML>
```

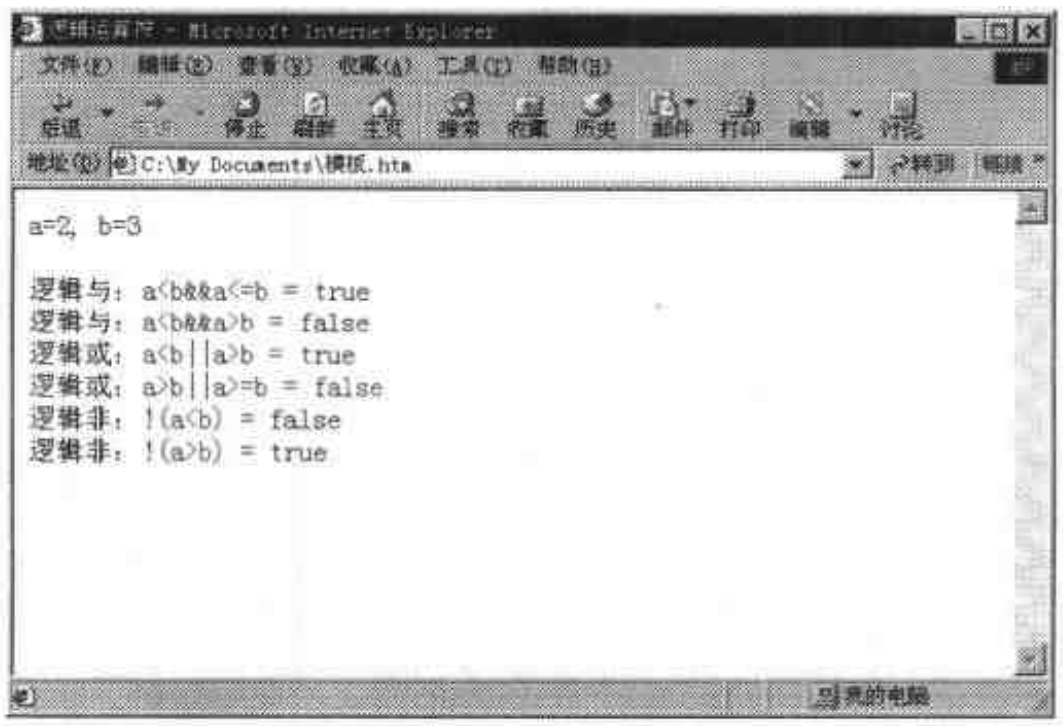


图 1.12　逻辑运算符

1.4.4　字符串运算符

在 JavaScript 中，只有一种字符串运算符——字符串加运算符（+）。该运算符与加法运算符相同，但在进行字符串操作时，结果是将两个字符串接合起来。

以下示例显示了字符串运算符的应用，效果如图 1.13 所示。

```
<HTML>
<HEAD>
  <TITLE>字符串运算符</TITLE>
```

```
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a="this";
var b="is";
var c=100;  //变量 c 是一个整数
var d=true;  //变量 d 是一个布尔值
document.write("a='this', b='is', c=100, d=true<P>");
document.write("a+b = "); document.write(a+b); document.write("<BR>");
document.write("b+a = "); document.write(b+a); document.write("<BR>");
document.write("a+c = "); document.write(a+c); document.write("<BR>");
document.write("c+a = "); document.write(c+a); document.write("<BR>");
document.write("a+d = "); document.write(a+d); document.write("<BR>");
document.write("d+a = "); document.write(d+a); document.write("<BR>");
document.write("a+b+d = "); document.write(a+b+d);
// -->
</SCRIPT>
</BODY>
</HTML>
```

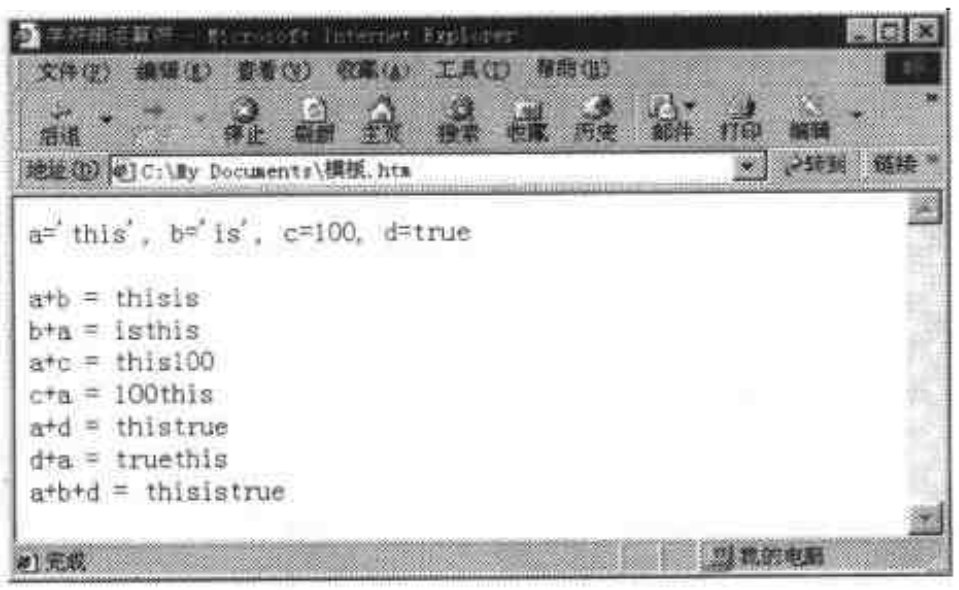


图 1.13 字符串运算符

说明：可以看出，在进行字符串运算时，数字值被转换为相应的字符串，而布尔值被转换为字符串"true"或"false"。

1.4.5 位操作运算符

位操作运算符用于对数值的位（二进制位，或者说比特位）进行操作，例如按位与、按位或、左移、右移等。在 JavaScript 中包括 6 个位运算符，如表 1.5 所示。

表 1.5 JavaScript 位运算符

运 算 符	说 明
&	按位与。两个操作数的相应位都为 1 时，该位的结果为 1，否则为 0。例如，4&7=4，因为 0100&0111=0100
|	按位或。两个操作数的相应位有一个为 1，则该位的结果为 1
^	按位异或。两个操作数的相应位不同时，该位的结果为 1
<<	左移。左移的位数由第二个操作数确定
>>	右移。右移的位数由第二个操作数确定
>>>	无符号右移

以下示例显示了位运算符的应用，效果如图 1.14 所示。

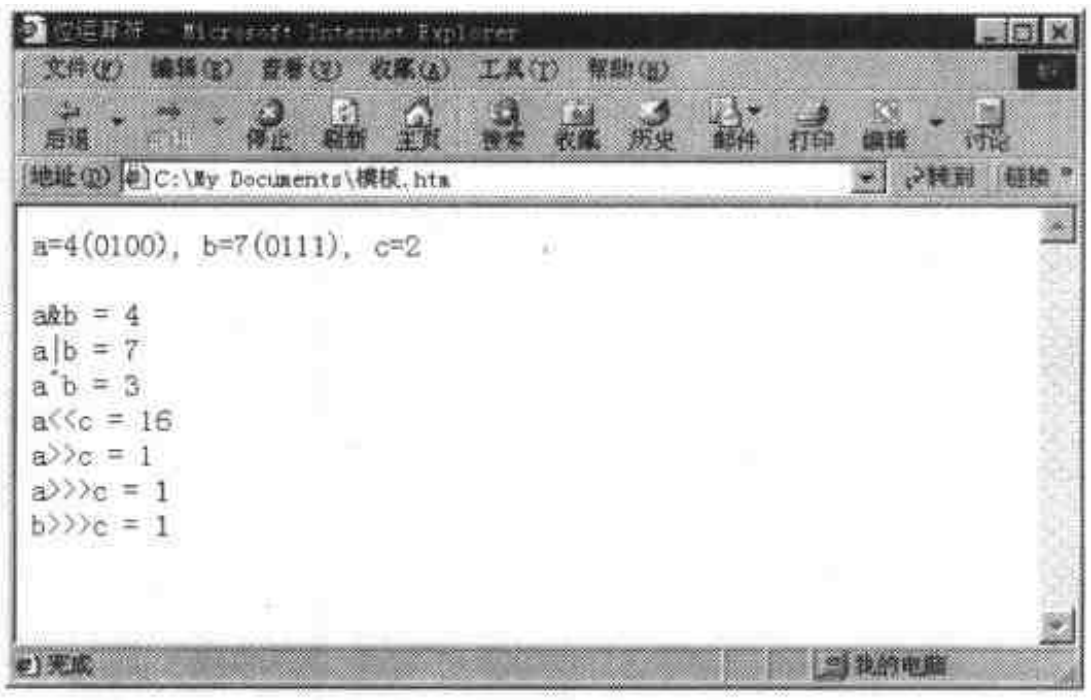


图 1.14 位运算符

```
<HTML>
<HEAD>
  <TITLE>位运算符</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a=4;
var b=7;
var c=2;
document.write("a=4(0100), b=7(0111), c=2<P>");
document.write("a&b = "); document.write(a&b); document.write("<BR>");
document.write("a|b = "); document.write(a|b); document.write("<BR>");
document.write("a^b = "); document.write(a^b); document.write("<BR>");
document.write("a&lt;&lt;c=");document.write(a<<c);
document.write("<BR>");
document.write("a>>c = "); document.write(a>>c); document.write("<BR>");
```

```
document.write("a>>>c = "); document.write(a>>>c);
document.write("<BR>");
document.write("b>>>c = "); document.write(b>>>c);
//   ->
</SCRIPT>
</BODY>
</HTML>
```

说明： 左移（<<）和右移（>>）操作常用于进行快速乘除，因为左移一位相当于源数乘以 2，而右移一位相当于源数除以 2。

1.4.6 赋值运算符

赋值运算符用于更新变量的赋值，另外一些运算符可以和赋值运算符联合使用，构成混合赋值运算符。JavaScript 中支持的各种赋值运算符如表 1.6 所示。

表 1.6 JavaScript 赋值运算符

运 算 符	说 明
=	将运算符左边的变量设置为右边表达式的值
+=	将运算符左边的变量递增右边表达式的值。例如，a+=b 相当于 a=a+b
-=	将运算符左边的变量递减右边表达式的值。例如，a-=b 相当于 a=a-b
*=	将运算符左边的变量乘以右边表达式的值。例如，a*=b 相当于 a=a*b
/=	将运算符左边的变量除以右边表达式的值。例如，a/=b 相当于 a=a/b
%=	将运算符左边的变量用右边表达式的值求模。例如，a%=b 相当于 a=a%b
&=	将运算符左边的变量与右边表达式的值按位与。例如，a&=b 相当于 a=a&b
|=	将运算符左边的变量与右边表达式的值按位或。例如，a|=b 相当于 a=a|b
^=	将运算符左边的变量与右边表达式的值按位异或。例如，a^=b 相当于 a=a^b
<<=	将运算符左边的变量左移，具体位数由右边表达式的值给出。例如，a<<=b 相当于 a=a<<b
>>=	将运算符左边的变量右移，具体位数由右边表达式的值给出。例如，a>>=b 相当于 a=a>>b
>>>=	将运算符左边的变量进行无符号右移，具体位数由右边表达式的值给出。例如，a>>>=b 相当于 a=a>>>b

以下示例显示了位运算符的应用，效果如图 1.15 所示。

```
<HTML>
<HEAD>
  <TITLE>赋值运算符</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
```

```
<!--
var a=3;
var b=2;
document.write("a=3, b=2<P>");
document.write("a+=b = "); document.write(a+=b); document.write("<BR>");
document.write("a-=b = "); document.write(a-=b); document.write("<BR>");
document.write("a*=b = "); document.write(a*=b); document.write("<BR>");
document.write("a/=b = "); document.write(a/=b); document.write("<BR>");
document.write("a%=b = "); document.write(a%=b); document.write("<BR>");
document.write("a&=b = "); document.write(a&=b); document.write("<BR>");
document.write("a|=b = "); document.write(a|=b); document.write("<BR>");
document.write("a^=b = "); document.write(a^=b); document.write("<BR>");
document.write("a|=b = "); document.write(a|=b); document.write("<BR>");
document.write("a&lt;&lt;=b = "); document.write(a<<=b);
document.write("<BR>");
document.write("a>>=b = "); document.write(a>>=b);
document.write("<BR>");
document.write("a>>>=b = "); document.write(a>>>=b);
document.write("<BR>");
// -->
</SCRIPT>
</BODY>
</HTML>
```

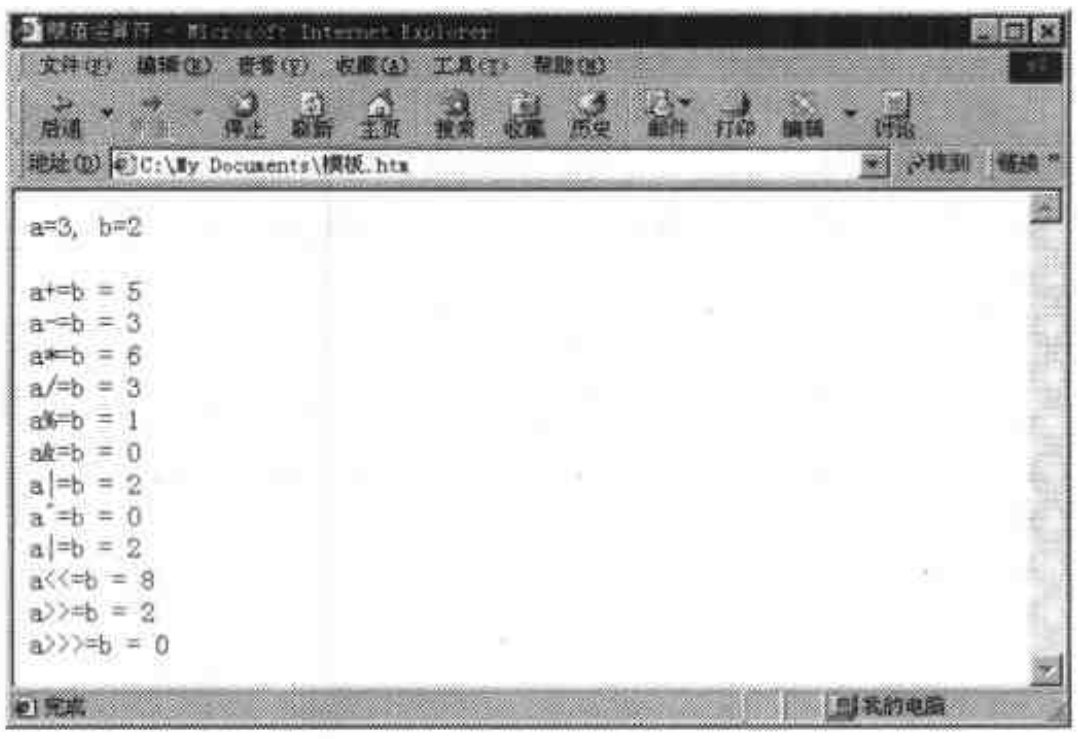


图 1.15　赋值运算符

说明： 赋值表达式的值也就是所赋的值。例如，a=(b+=c) 就相当于 a=(b=b+c)，相当于 a=b+c。在以上示例中，a 的值不断随赋值语句发生变化，而 b 的值始终不变。

1.4.7　条件运算符

JavaScript 支持一种特殊的三目运算符，称为条件运算符，其格式如下：

条件？结果 1：结果 2

该格式表示若“条件”值为真，则表达式的值为“结果 1”，否则为“结果 2”。

以下示例显示了条件运算符的应用，效果如图 1.16 所示。

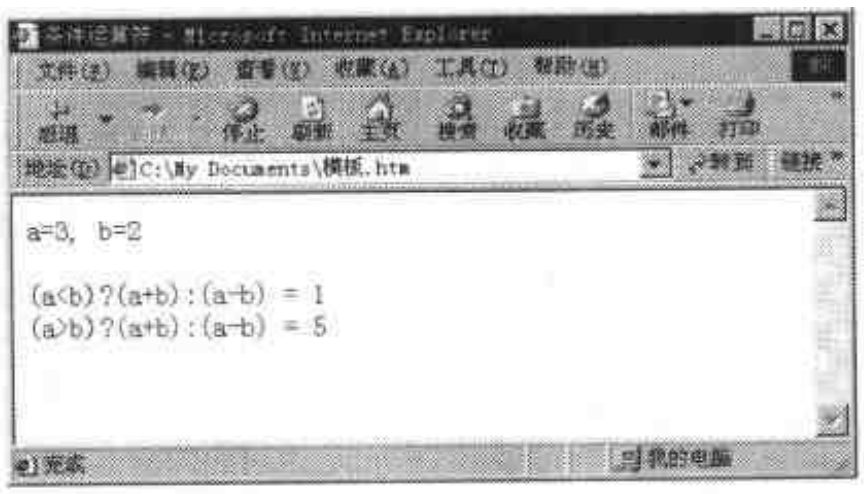


图 1.16　条件表达式

```
<HTML>
<HEAD>
  <TITLE>条件运算符</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a=3;
var b=2;
document.write("a=3, b=2<P>");
document.write("(a&lt;b)?(a+b):(a-b) = ");
document.write((a<b)?(a+b):(a-b));  /*若 a<b 则计算 a+b 并把值赋给表达式，
否则计算 a-b 并把值赋给表达式。*/
document.write("<BR>")
document.write("(a>b)?(a+b):(a-b) = ");
document.write((a>b)?(a+b):(a-b));
// -->
</SCRIPT>
</BODY>
</HTML>
```

1.4.8 其他运算符

JavaScript 中还包含其他几个特殊运算符，如表 1.7 所示。

表 1.7 其他运算符

运 算 符	说 明
.	成员选择运算符，用于引用对象的属性和方法。例如，window.status。有关对象的详细信息，将从第 2 章开始说明
[]	下标运算符，用于引用数组元素。例如，class[3]。有关数组的详细信息，请参见第 2 章
( )	函数调用运算符，用于进行函数调用。例如，myFunction()。有关函数的详细信息，请参见本章的 1.6 节
,	逗号运算符，用于将不同的值分开。例如，var today, date
delete	delete 运算符删除一个对象的属性或一个数组索引处的元素。例如，delete myArray[3]删除 myArray 数组的第 4 个元素
new	new 运算符生成一个对象的实例。例如，new myObject
typeof	typeof 运算符返回表示操作数类型的字符串值。例如，typeof true 的值为 boolean
Void	void 运算符不返回任何数值

1.4.9 运算符的优先级

运算符的优先级确定了计算复杂表达式时哪个运算优先进行。最基本的运算符优先级就是所谓的“先乘除，后加减”。

JavaScript 定义了所有运算符的优先顺序，如表 1.8 所示。

表 1.8 运算符优先顺序

优 先 顺 序	运 算 符
1	成员选择、括号、函数调用、数组下标
2	!、-（单目减）、++、--、typeof、new、void、delete
3	*、/、%
4	+、-
5	<<、>>、>>>
6	<、<=、<、>=
7	==、!=、===、!==、
8	&
9	^
10	|
11	&&
12	||
13	?:
14	=、+=、-=、*=、/=、%=、<<=、>>=、>>>=、&=、^=、|=
15	逗号运算符（,）

说明：对于优先顺序处于同一层次上的运算符，按照从左到右出现的顺序计算。

以下示例显示了运算符优先顺序的作用，效果如图 1.17 所示。

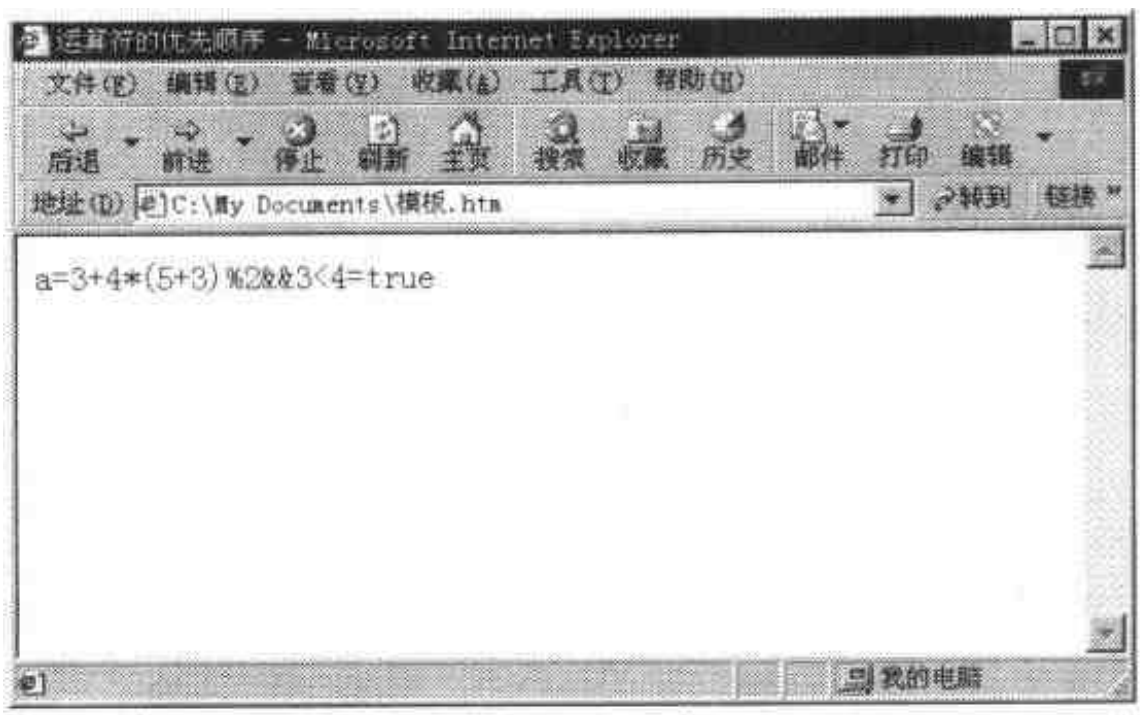


图 1.17　运算符的优先顺序

```
<HTML>
<HEAD>
  <TITLE>运算符的优先顺序</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a;
document.write("a=3+4*(5+3)%2&&3<4=");
document.write(a=3+4*(5+3)%2&&3<4);
// -->
</SCRIPT>
</BODY>
</HTML>
```

说明：以上示例中的运算顺序依次为 (5+3)、4*(5+3)、(4*(5+3))%2、3+(4*(5+3))%2、3<4、(3+(4*(5+3))%2)&&(3<4)。可以看出，对于复杂表达式运算符的优先顺序决定了表达式的值。不过，为了容易理解起见，表们最好将运算的顺序用括号明确表示，这样就不必记忆到底哪个运算符的优先级更高。例如，以上示例中的表达式如果写成 (3+4*(5+3)%2)&&(3<4)，就很容易看出这是一个逻辑表达式，其结果要么是 true 要么是 false。

1.5 JavaScript 语句

1.5.1 概述

在任何一门编程语言中，程序的逻辑都是通过编程语句来实现的。在 JavaScript 中包含完整的一组编程语句，用于实现基本的程序控制和操作功能。

表 1.9 总结了 JavaScript 中提供的语句，在后面的小节中将分别详细说明。

表 1.9 JavaScript 语句

语　句	格　式
赋值	variable=expression;
数据声明	var variable;
If	if(condition) statement;
switch	switch(expression) {case value1: statement; break; case value2: statement : break; ------ default: statement; }
while	while(condition) statement;
for	for(expression;condition;expression) statement;
do while	do {statement;} while(expression);
label	labelName: statement;
break	break;
continue	continue;
函数调用	function();
return	return value;
with	with(object) statement;
for in	for(variable in object) {statement;}

1.5.2 条件语句

条件语句可以使程序按照预先指定的条件进行判断，从而选择执行任务。在 JavaScript 中提供了 if 语句、else if 语句以及 switch 语句等三种条件语句。

1. if 语句

if 语句是最基本的条件语句，它的格式为：

```
if(expression)
  statement;
```

也就是说，如果括号里的表达式为真，则执行 statement 语句，否则就跳过该语句。如果要执行的语句只有一条，那么可以写在与 if 所在的同一行，例如：

```
if(a==1)a++;
```

如果要执行的语句有多条，则应使用大括号将这些语句括起来，例如：

```
if(a==1){a++;b++;}
```

说明：如果要在同一行中书写多个语句，语句之间应用分号分隔。

不过，为了易于阅读，建议尽量使用多行，例如：

```
if(a==1)
{a++;
 b++;}
```

2. if else 语句

如果需要在表达式为假时执行另外一个语句，则可以使用 else 关键字扩展 if 语句。if else 语句的格式为：

```
if(expression)
 statement1;
else
 statement2;
```

同样，语句 1 和语句 2 都可以是一个代码块。实际上，如果语句本身又是一个条件语句，则构成了条件语句的嵌套，如下例所示。这段代码的执行效果如图 1.18 所示。

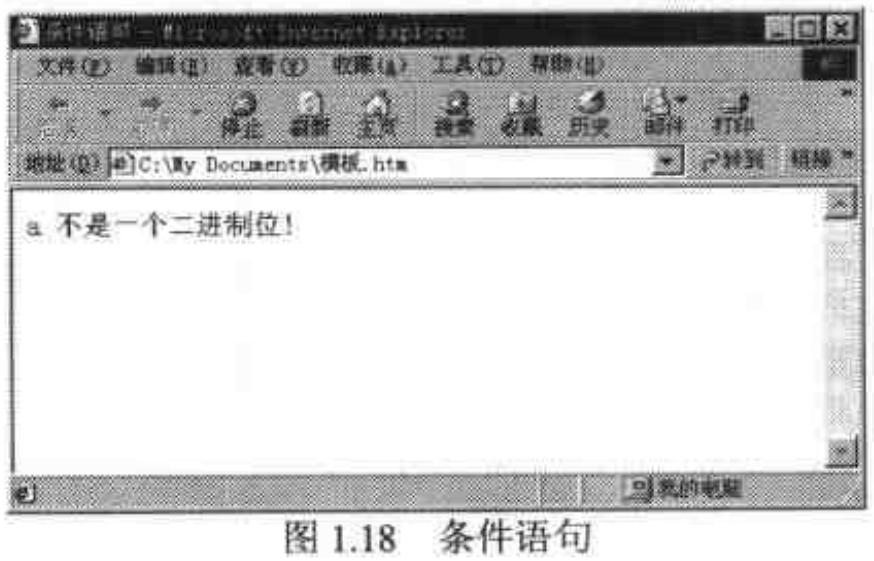


图 1.18 条件语句

```
<HTML>
<HEAD>
  <TITLE>条件语句</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a=3;
if(a==0)
 document.write("a=0");
else
{
if(a==1)
 document.write("a=1")
else
 document.write("a 不是一个二进制位!")
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

除了用条件语句的嵌套表示多种选择，还可以直接用 else if 语句获得这种效果，格式如下：

```
if(expression1)
 statement1;
else if(expression2)
 statement2;
else if(expression3)
 statement3;
-
else
 statementn;
```

该格式表示只要满足任何一个条件，则执行相应的语句，否则执行最后一条语句。例如，可以将刚才的代码段改为如下，效果与嵌套的条件语句完全一样：

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
```

```
var a=3;
if(a==0)
 document.write("a=0");
else if(a==1)
 document.write("a=1")
else
 document.write("a 不是一个二进制位!")
// -->
</SCRIPT>
```

3. switch 语句

如果需要对同一个表达式进行多次判断，那么就可以使用 switch 语句，格式如下：

```
switch(expression)
{  //注意：必须用大括号将所有 case 括起来。
case value1:
  statement1;  //注意：此处即使使用了多条语句，也不能使用大括号。
  break;  /* 注意：如果不使用 break 语句断开各个 case，则在执行（如果确实执行）
此 case 中的语句结束后会接着继续执行下一个 case 中的语句。*/
case value2:
  statement2;
  break;
……
case valueN:
  statementN;
  break;
default:
  statement;
}
```

该格式实际上相当于以下 if else 语句：

```
if(expression==value1) statement1;
 else if(expression==value2) statement2;
 -
  else if(expression==valueN) statementN;
   else statement;
```

但 switch 语句显然比 if else 语句更容易让人理解，尤其是当需要判断的条件多于 3 个时。

以下示例显示了 switch 语句的用法，显示效果如图 1.19 所示。

```
<HTML>
```

```
<HEAD>
  <TITLE>switch 语句</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var a=1;
switch (a)
{
case 0:
  document.write("a=0");
  break;
case 1:
  document.write("a=1");
  break;
default:
  document.write("a 不是一个二进制位!")
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

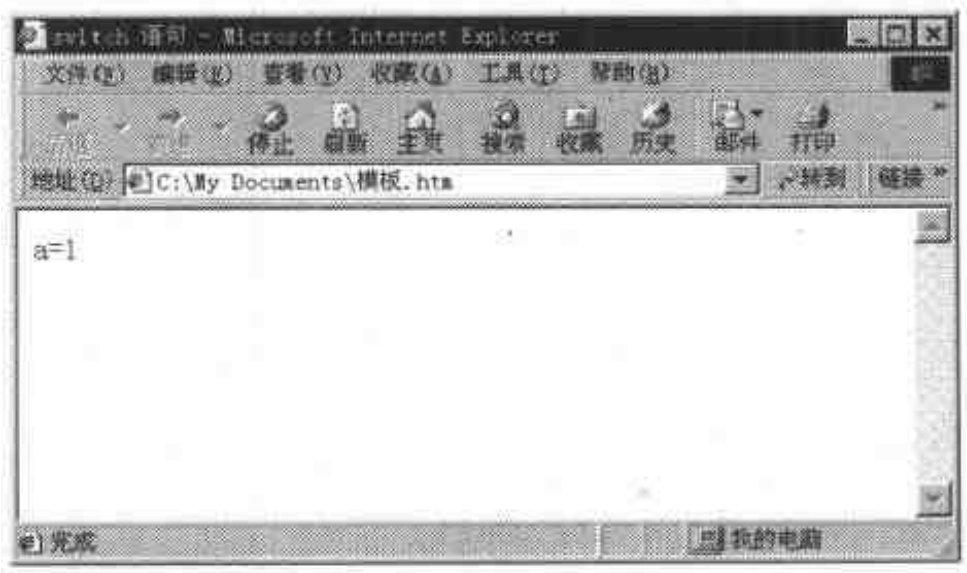


图 1.19 switch 语句

1.5.3 循环语句

循环语句用于在一定条件下重复执行某段代码。在 JavaScript 中提供了多种循环语句：for 语句、while 语句以及 do while 语句，同时还提供了 break 语句用于跳出循环，continue 语句用于终止当前循环并继续执行下一轮循环，以及 label 语句用于标记一个语句。

1. for 语句

for 语句的格式如下：

```
for (initializationStatement; condition; adjustStatement)
{
statement;
}
```

可以看出，for 语句由两部分构成：条件和循环体。循环体部分由具体的语句构成，是需要循环执行的代码。条件部分由括号括起来，分为三个部分，每个部分用分号分开。第一部分是计数器变量初始化部分；第二部分是循环判断条件，决定了循环的次数；第三部分给出了每循环一次，计数器变量应如何变化。

for 循环的执行步骤如下：

（1）执行 initializationStatement 语句，完成计数器初始化；

（2）判断条件表达式 condition 是否为 true，如果为 true，执行循环体语句，否则退出循环；

（3）执行循环体语句之后，执行 adjustStatement 语句。

（4）重复步骤（2）和（3），直到退出循环。

以下示例显示了如何在 JavaScript 中使用 for 语句，效果如图 1.20 所示。

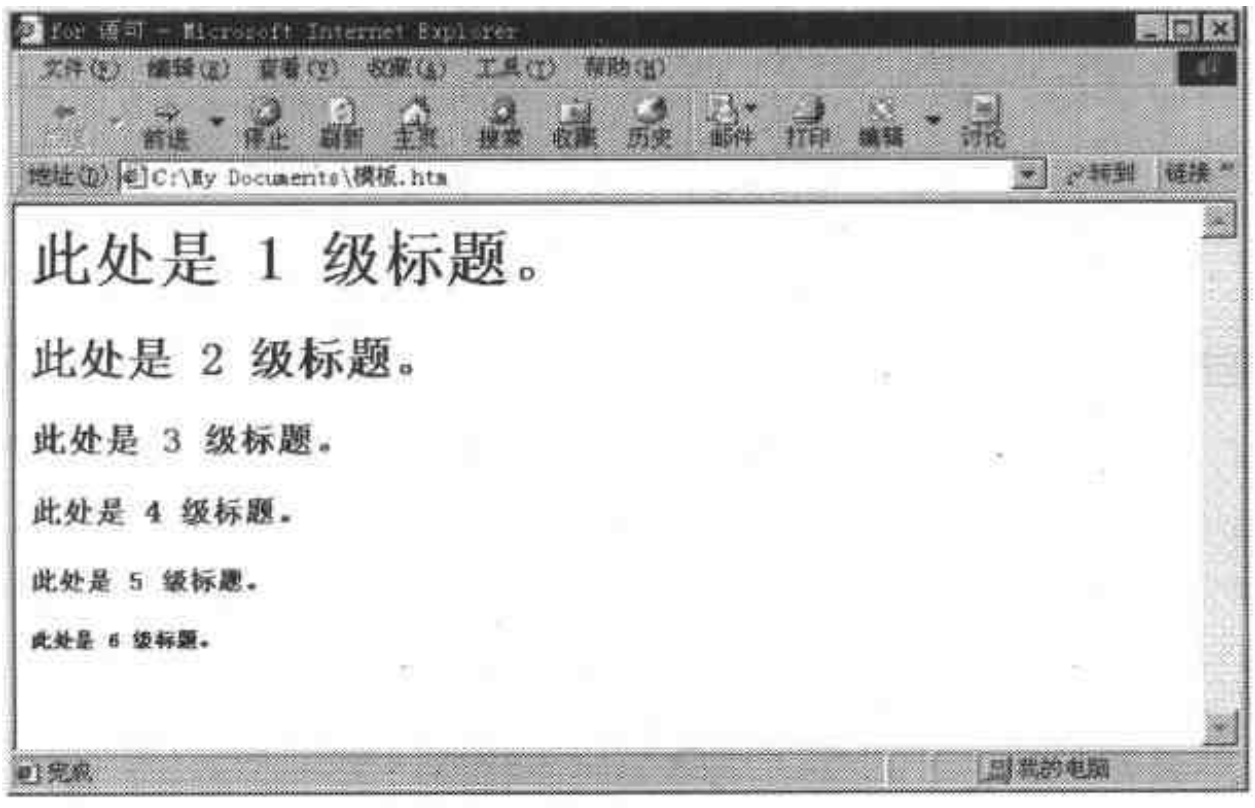


图 1.20 for 语句示例

```
<HTML>
<HEAD>
  <TITLE>for 语句</TITLE>
```

```
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
for(i=1;i<7;i++)
{   //读者可以自行按照刚才介绍的 for 循环的执行步骤来理解此循环的含义。
  document.write("<H"+i+">此处是 "+i+" 级标题。</H"+i+">");
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

实际上，for 循环语句还可以嵌套，通过嵌套实现更复杂的应用。例如，以下示例在网页中显示出了乘法表，效果如图 1.21 所示。

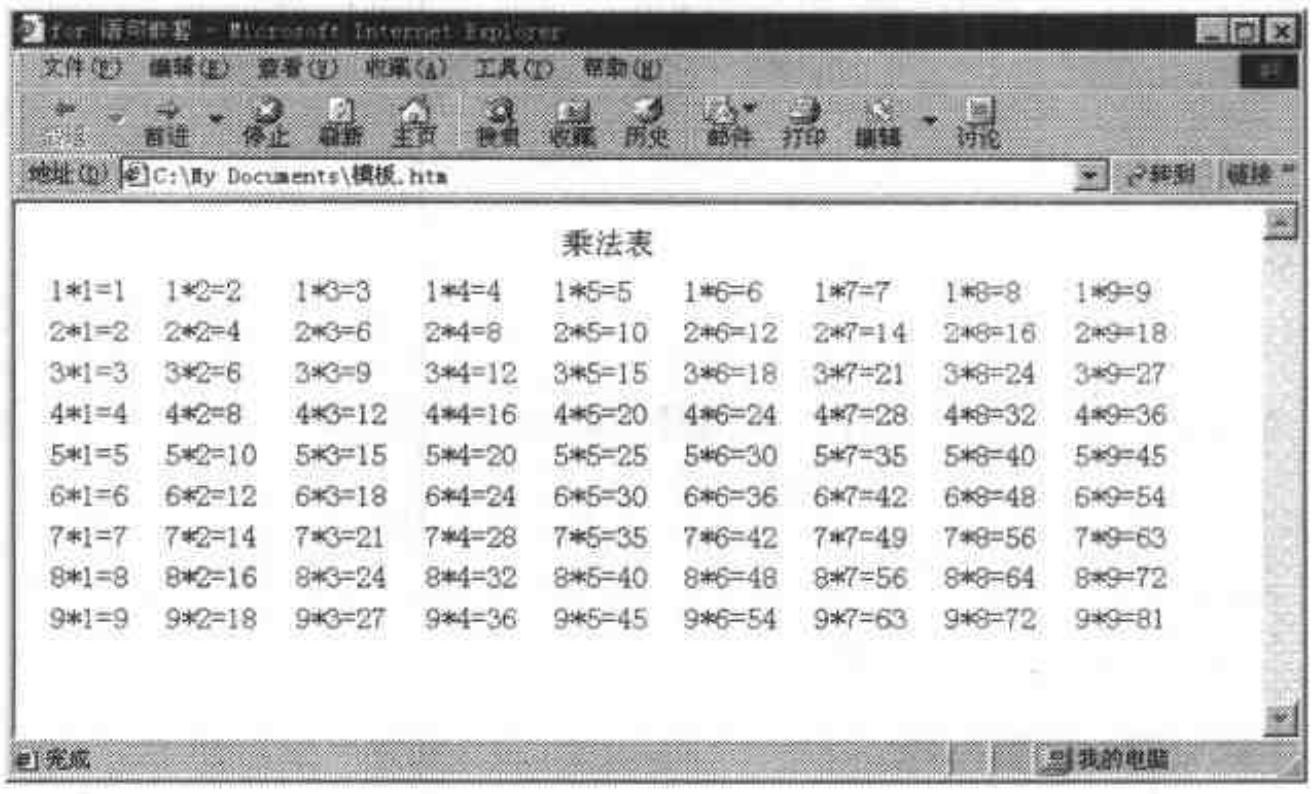


图 1.21　for 循环嵌套

```
<HTML>
<HEAD>
  <TITLE>for 语句嵌套</TITLE>
</HEAD>
<BODY>
<TABLE>  //使用表格协助布局
<CAPTION>乘法表</CAPTION>  //定义表格标题
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
```

```
for(i=1;i<10;i++)
{document.write("<tr>")  //定义表格行
 for(j=1;j<10;j++)
  {document.write("<td>");  //定义表格列
   document.write(" ");  //  是空格的字符实体
   document.write(i+"*"+j+"=");
   document.write(i*j);
   document.write(" ");
  }
document.write("<BR>")
}
// -->
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

2. while 语句

while 语句是另一种基本的循环语句，格式如下：

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

表示当表达式为真时执行循环体语句。

while 循环的执行步骤如下：

（1）计算 expression 表达式的值；

（2）如果 expression 表达式的值为真，则执行循环体，否则跳出循环；

（3）重复执行步骤（1）和（2），直到跳出循环。

以下示例显示了如何用 while 语句控制循环，效果如图 1.22 所示：

```
<HTML>
<HEAD>
  <TITLE>while 语句</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
```

```
var i=0, s=0;
while(i<=100)
{  s+=i;
   i++;
}
document.write("1+2+3+...+100 = "+s);
// -->
</SCRIPT>
</BODY>
</HTML>
```

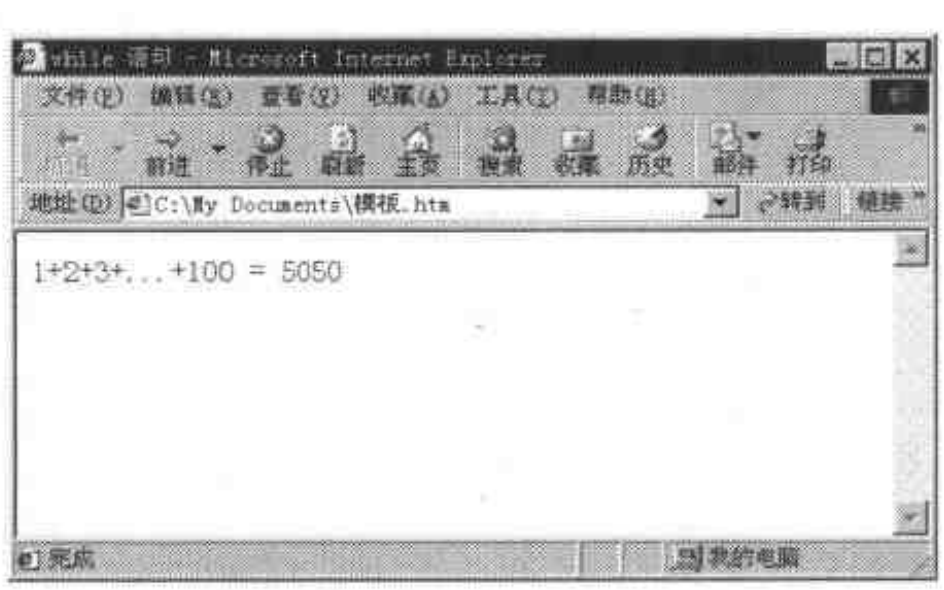


图 1.22　while 语句

3. do while 语句

do while 语句是 while 语句的变体，格式如下：

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

它的执行步骤如下：

（1）执行循环体语句。

（2）计算 expression 表达式的值。

（3）如果表达式的值为真，则执行循环体语句，否则退出循环。

（4）重复步骤（2）和（3），直到退出循环。

可见，do while 语句与 while 语句的区别是循环体语句至少执行一次。因为在 while

语句中，如果第一次表达式计算的值就为 false，则循环一次都不执行。除此之外，这两种语句并没有其他区别。

例如，可以上一个示例中的 while 循环改写为 do while 循环，如下所示：

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var i=0, s=0;
do
{  s+=i;
   i++;
}
while(i<=100)
document.write("1+2+3+...+100 = "+s);
// -->
</SCRIPT>
```

说明： 无论采用哪一种循环语句，都必须注意控制循环的结束条件，以免出现“死循环”。以下将介绍的 label、break 和 continue 语句可以进一步帮助控制循环。

4. label 语句

label 语句用于为语句添加标号。在任意语句前放上标号名称即可为该语句指定标号。例如：

```
myLabel:
a++;
```

为 a++这条语句指定了标号 myLabel。

label 语句通常用于标记一个循环、switch 或 if 语句，并且与 break 或 continue 语句联合使用。

5. break 语句

break 语句提供无条件跳出循环结构或 switch 语句的功能。在多数情况下，break 语句都是单独使用的。但有时也可以在其后面加一个语句标号，以表明跳出该标号所指定的循环，执行该循环之后的代码。

以下示例显示了 break 语句的用法，效果如图 1.23 所示。

```
<HTML>
```

```
<HEAD>
  <TITLE>break 语句</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
forLoop1:
for(i=1; i<10; i++)
{
  for(j=1; j<10; j++)
  {
  document.write("i="+i+" ");
  document.write("j="+j+"<BR>");
  if(j==3) break;    //跳出嵌套的第二层循环
  if(i==3) break forLoop1;    //跳出顶层循环
  }
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

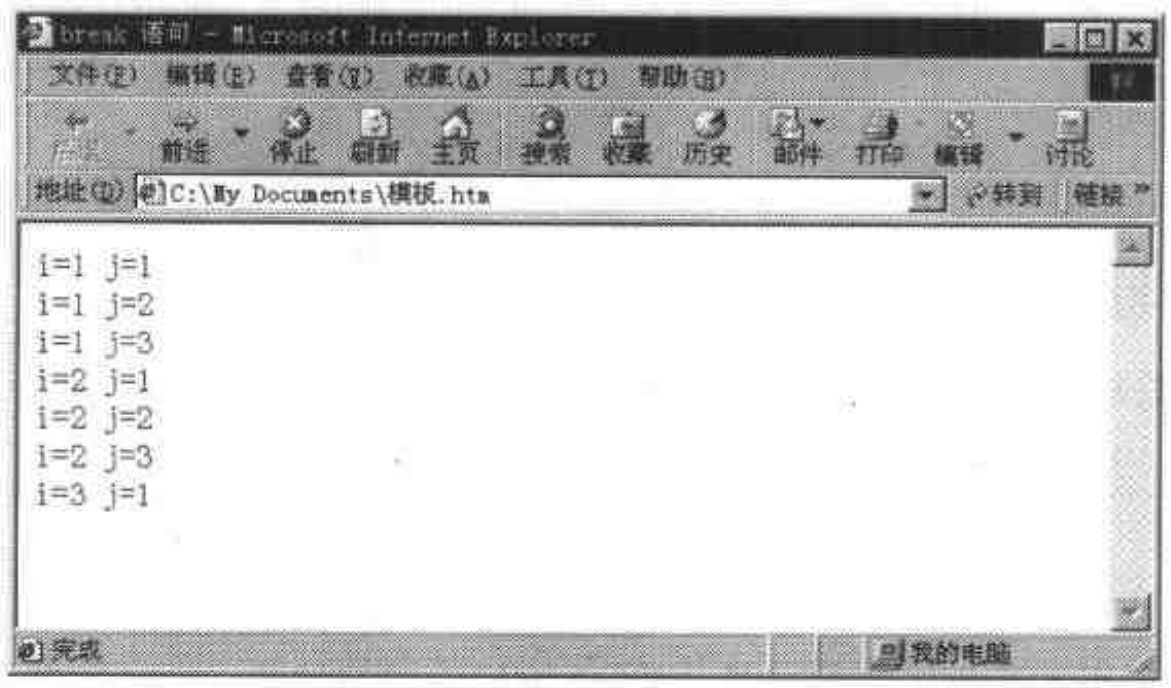


图 1.23 break 语句

6. continue 语句

与 break 语句不同，continue 语句的作用是终止当次循环，跳转到循环的开始处继续下一轮循环。同样，continue 语句既可以单独使用，也可以与语句标号一起使用。

以下示例显示了 continue 语句的用法，效果如图 1.24 所示。

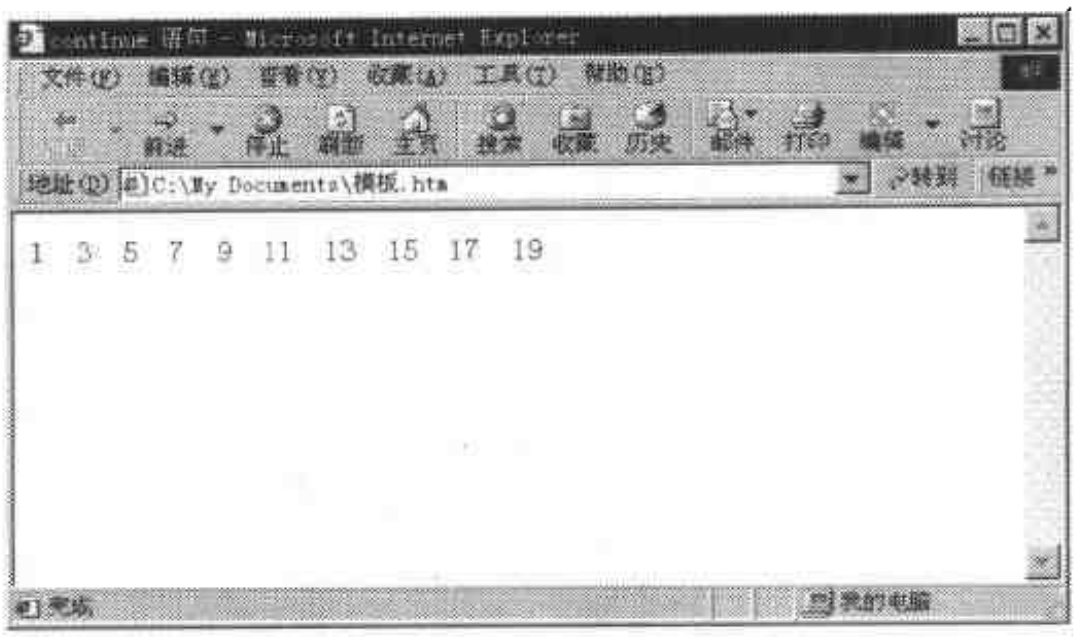


图 1.24　continue 语句

```
<HTML>
<HEAD>
  <TITLE>continue 语句</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
for(i=1; i<=20; i++)
{
if(i%2==0) continue;   //遇到偶数则跳出此次循环，进入下一轮循环
document.write(i+"  ")
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

1.5.4　其他语句

除了以上条件语句和循环语句以外，JavaScript 中还包括以下语句：

- 赋值语句　使用赋值运算符构成的语句都是赋值语句，用于更新变量的值。例如，a+=3。
- 数据声明语句　数据声明语句用于声明一个变量。例如，var i=3, j=4。
- 函数调用语句　函数调用语句用于调用函数。例如，escape("this is a test")。
- return 语句　用于返回函数调用的值。例如，return x*x。

- with 语句　用于表示默认对象。例如，with(Math){ d = 2*PI*r;}。有关信息，请参见第 2 章。
- for in 语句　用于对一个对象的所有属性进行循环，直到每个属性都访问到。例如，for (aProperty in document.Form1.myBotton) { document.write(aProperty + "<BR>"); }。有关信息，请参见第 2 章。

1.6 JavaScript 函数

1.6.1 使用函数

函数是已命名的代码块，代码块中的语句作为一个整体引用和执行。函数可以使用参数来传递数据，也可以不使用参数。函数可以使用 return 语句返回确切的值，也可以不返回任何值。

1. 定义函数

在使用函数之前，必须先定义函数。函数定义通常放在 HTML 文档头中，也可以放在其他位置。但最好放在文档头，这样就可以确保先定义后使用。

定义函数的格式如下：

```
function functionName (parameter1, parameter2 ...)
{
  statements
}
```

函数名是调用函数时引用的名称，参数是调用函数时接收传入数值的变量名。大括号中的语句是函数的执行语句，当函数被调用时执行。

以下示例显示了如何在 JavaScript 中使用函数，效果如图 1.25 所示。

```
<HTML>
<HEAD>
<TITLE>使用函数</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function Hello()
```

```
{
  document.write("Hello, ");
}
function Message(message)
{
  document.write(message);
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
  Hello();
  Message("JavaScript");
//-->
</SCRIPT>
</BODY>
</HTML>
```

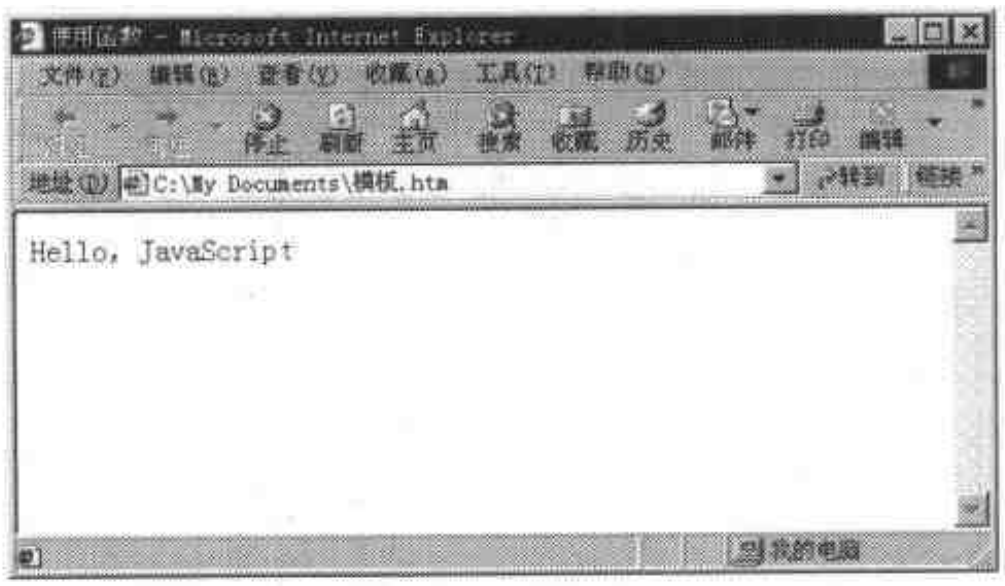


图 1.25 使用函数

2. 函数的参数

在 JavaScript 中使用函数时，既可以用函数定义时确定的参数进行调用，也可以不完全按照函数定义使用参数。无论函数如何定义，都可以用 arguments 数组来访问调用函数时所用的参数。每次调用函数时，JavaScript 都会自动生成 arguments 数组。

以下示例显示了 JavaScript 函数如何使用 arguments 数组给函数传递参数，效果如图 1.26 所示。

```
<HTML>
<HEAD>
<TITLE>使用 arguments 数组</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function displayArguments()
{
document.write("此函数使用了以下参数: <BR>")
for(i=0; i<arguments.length; i++)
{
document.write(i+"="+arguments[i]+"<BR>")
}
}
displayArguments(21,"cat",-2000,"tiger","great");
//-->
</SCRIPT>
</BODY>
</HTML>
```

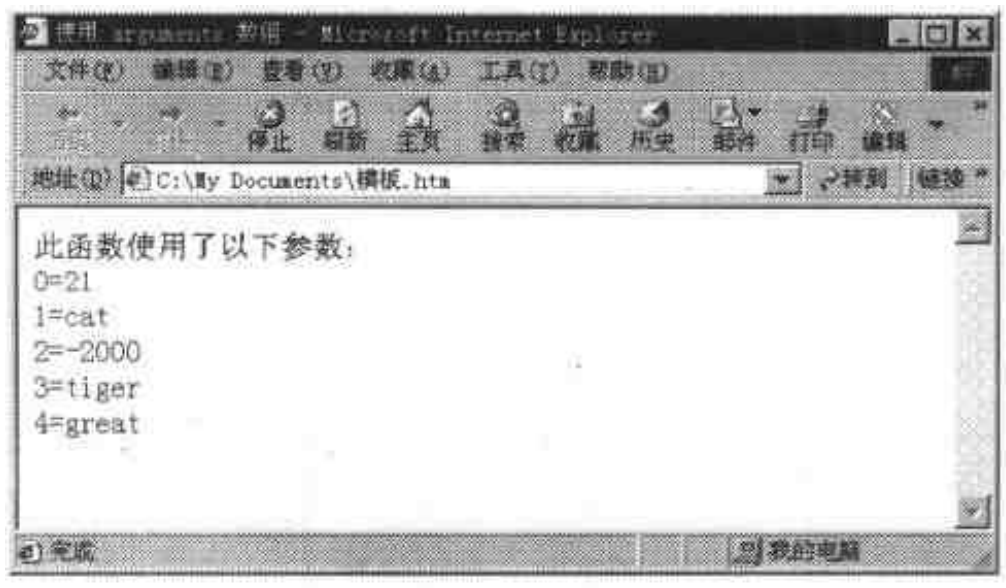


图 1.26　使用 arguments 数组

3. 函数的返回值

如果需要函数返回值，那么可以使用 return 语句，需要返回的值应放在 return 之后。如果 return 后没有指明数值或者没有使用 return 语句，则函数返回值为不确定值。

函数返回值可以直接赋予变量或用于表达式。例如，在以下示例中，函数的返回值直接用在了表达式中，效果如图 1.27 所示。

```
<HTML>
```

```
<HEAD>
<TITLE>函数的返回值</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function add(a,b)
{
  return(a+b);
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
document.write("3 + 5 = "+add(3,5));
//-->
</SCRIPT>
</BODY>
</HTML>
```

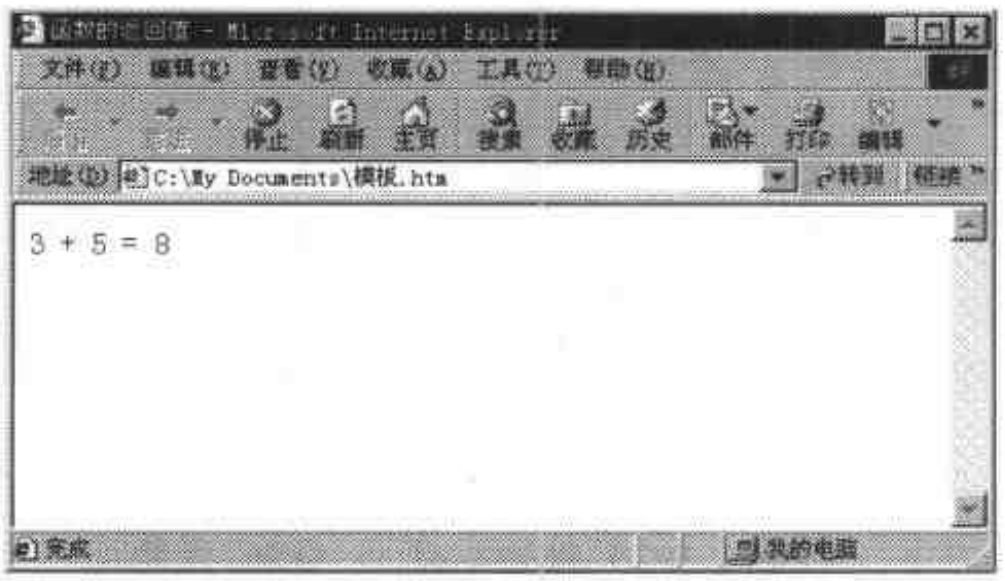


图 1.27 函数的返回值

1.6.2 JavaScript 全局函数

JavaScript 中包含以下 7 个全局函数，用于完成一些常用的功能（以后的章节中可能会用到）：escape( )、eval( )、isFinite( )、isNaN( )、parseFloat( )、parseInt( )、unescape( )。

1. escape( )

escape( ) 函数以一个 string 对象或表达式为参数并返回一个 string 对象。参数指定的字符串中的所有非字母字符被转换成以 XX%表示的等价数字，XX 是一个表示非字母字

符的十六进制数。

以下示例显示了 escape( )函数的作用，效果如图 1.28 所示。

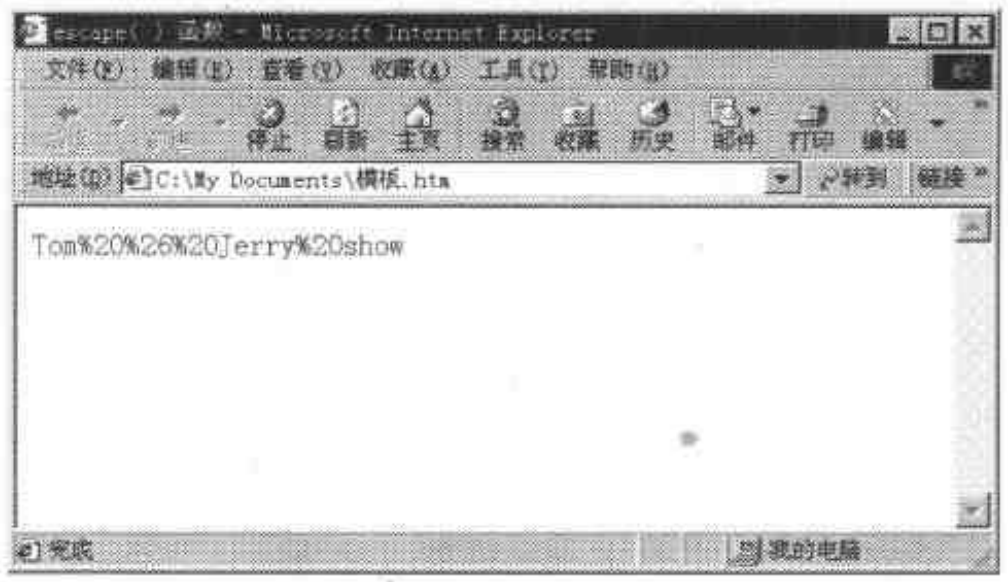


图 1.28 escape( ) 函数

```
<HTML>
<HEAD>
  <TITLE>escape( ) 函数</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var escapeString=escape("Tom & Jerry show");
document.write(escapeString);
//-->
</SCRIPT>
</BODY>
</HTML>
```

说明：在以上示例中，空格被 %20 代替，& 被 %26 代替。

2. eval( )

eval( )函数将通过参数传入的一个包含 JavaScript 语句的字符串作为一个 JavaScript 源代码执行。eval ( )返回执行 JavaScript 语句的返回值。

例如，在编写跨浏览器代码时，可以使用以下代码段：

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
styleRef=".style"
eval("document.all.['something'],"+styleRef+".visibility='visible'");
/* 以上函数调用相当于执行语句：document.all.['something'].style.visibility
```

```
='visible' */
//-->
</SCRIPT>
```

3. isFinite( )

isFinite ( )函数用于确定一个变量是否有界，如果有界则返回 true，否则返回 false。所谓有界是指表达式的值界于 MAX_VALUE 和 MIN_VALUE 之间。

以下示例显示了 isFinite( )函数的作用，效果如图 1.29 所示。

```
<HTML>
<HEAD>
  <TITLE>isFinite( ) 函数</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
document.write("isFinite(12)="+isFinite(12)+"<BR>");
document.write("isFinite(1.2)="+isFinite(1.2)+"<BR>");
document.write("isFinite('a')="+isFinite('a')+"<BR>")
document.write("isFinite(true)="+isFinite(true)+"<BR>")
document.write("isFinite(null)="+isFinite(null))
//-->
</SCRIPT>
</BODY>
</HTML>
```

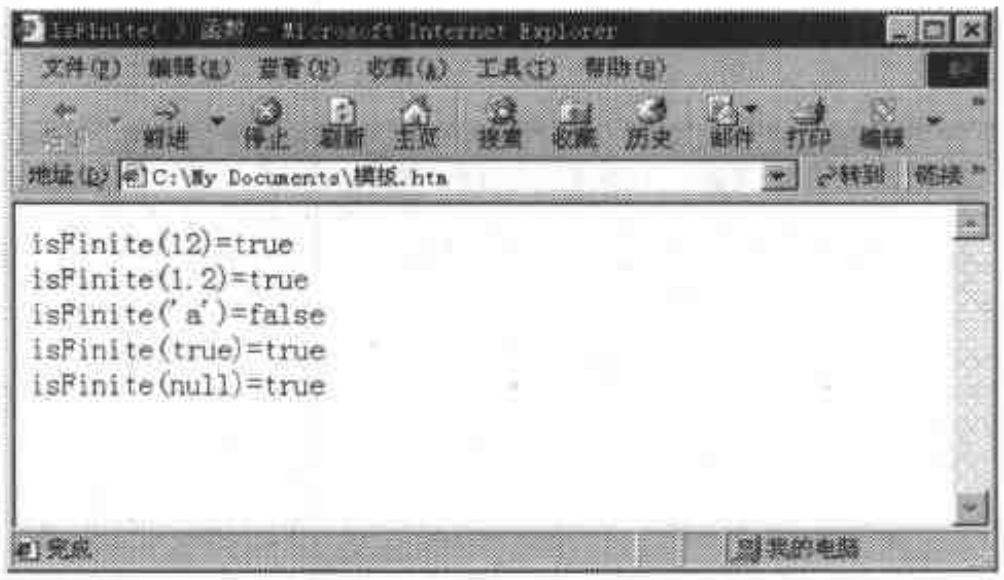


图 1.29 isFinite( ) 函数

4. isNaN( )

isNaN( )函数用于确定一个变量是否是 NaN，如果是，则返回 true，否则返回 false。NaN

代表 Not a Number，表示非数，即不是任何数。

例如，isNaN(12)返回 false，isNaN('a')返回 true，isNaN(true)返回 false（因为此时 true 被当作数字 1）。

5. parseFloat( )

parseFloat( )函数用于将字符串开头的整数或浮点数分解出来，若字符串不是以数字开头，则返回 NaN。

以下示例显示了 parseFloat( )函数的用法，效果如图 1.30 所示。

```
<HTML>
<HEAD>
  <TITLE>parseFloat( ) 函数</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
document.write("parseFloat('123.56abc')="+parseFloat("123.56abc")+"<
BR>");
document.write("parseFloat('123abc')="+parseFloat("123abc")+"<BR>");
document.write("parseFloat('abc')="+parseFloat("abc")+"<BR>")
document.write("parseFloat(true)="+parseFloat(true))
//-->
</SCRIPT>
</BODY>
</HTML>
```

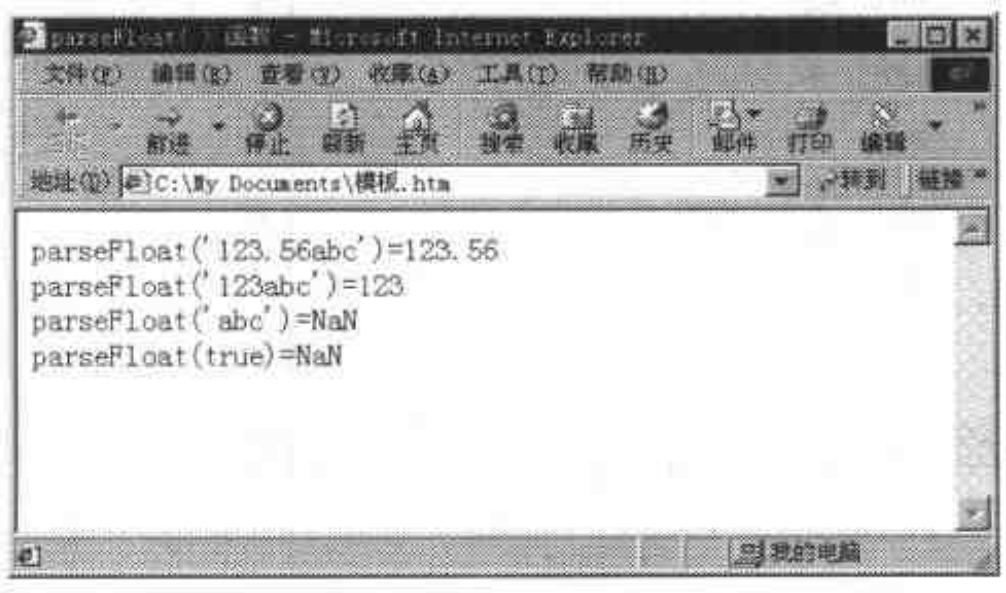


图 1.30　parseFloat( ) 函数

6. parseInt( )

parseInt( )函数与 parseFloat( )函数类似，用于将字符串开头的整数分解出来，若字符串不是以数字开头，则返回 NaN。

例如，如果将刚才的 parseFloat( )函数示例中的所有 parseFloat 都用 parseInt 代替，则结果如图 1.31 所示。

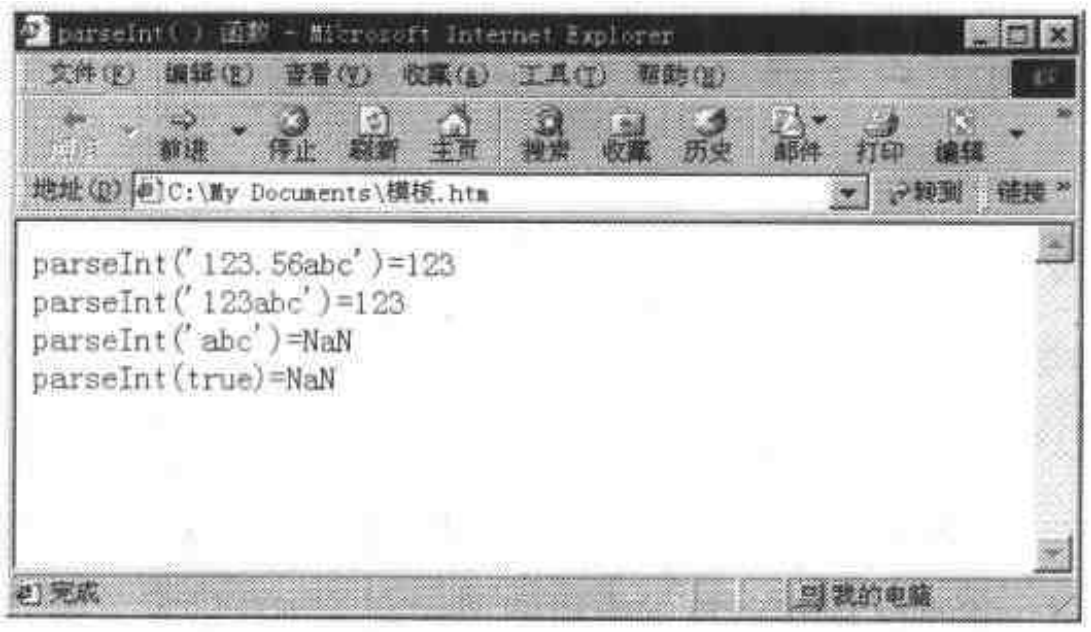


图 1.31 parseInt( )函数

7. unescape( )

unescape( )函数将参数传递来的字符串中的十六进制码转换成 ASCII 码并返回，它完成 escape( )函数的逆操作。例如，unescape("Tom%20%26%20and%20Jerry%20show") 的返回值为"Tom & Jerry show"。

第 2 章 JavaScript 对象

JavaScript 作为一种基于对象的编程语言，其中主要包含两类对象：一类是 JavaScript 内建对象，即 JavaScript 语言本身所自带的对象；另一类是浏览器对象，由客户端浏览器所支持。通过使用这些对象，网页设计者可以控制页面元素的显示以及完成各种实用功能（例如，使用 Math 对象的方法可以完成某些数学计算）。本章介绍有关 JavaScript 对象的基础知识以及各种 JavaScript 内建对象的应用细节（有关浏览器对象的详细信息，将从第 4 章开始介绍）。

本章主要内容包括：

- 什么是对象
- 使用 JavaScript 对象
- Array（数组）对象
- Boolean（布尔）对象
- Date（日期）对象
- Function（函数）对象
- Global（全局）对象
- Math（数学）对象
- Number（数字）对象
- Object 对象
- RegExp（正规表达式）对象
- String（字符串）对象

2.1 什么是对象

2.1.1 对象的属性与方法

对象就是客观世界中存在的特定实体。“人”就是一个典型的对象，“他”包含身高、体重、年龄等特性，同时又包含吃饭、睡觉、行走这些动作——“人”这个对象由这些特性和动作所规定。同样，一盏灯也是一个对象，它包含功率、亮灭状态等特性，同时又包含“开灯”、“关灯”这些动作。

在计算机世界中，也包含各种各样的对象。例如，一个 Web 页可以被看作一个对象，它包含背景颜色、前景颜色等特性，同时包含打开、关闭、写等动作。Web 页上的一个表单也可以看作一个对象，它包含表单内控件的个数、表单名称等特性，以及表单提交和表单重置等动作。

根据这些说明可以看出，对象包含两个要素：

- 用来描述对象特性的一组数据，也就是若干变量，通常称为属性；
- 用来操作对象特性的若干动作，也就是若干函数，通常称为方法。

例如，document 对象的 bgColor 属性用于描述文档的背景颜色，而使用 document 对象的 write 方法可以在文档中写特定内容。

通过访问或设置对象的属性，并且调用对象的方法，我们就可以对对象进行各种操作，从而获得需要的功能。

2.1.2 基于对象的 JavaScript

1. 面向对象技术

面向对象技术，也就是通常所说的 OO 技术（Object-Oriented Technology），近年来已经逐步成为占主导地位的编程技术。与常规的线性编程方法不同，在面向对象的编程技术中，从概念上将一组函数和变量组织成一个对象，从而将数据封装起来，达到模块化编程的目的。

面向对象技术具有一些典型的特点，包括：

- 封装性（Encapsulation） 所谓封装就是将对象的属性和方法封装到具有适当定义接口的容器中，对象通过接口提供的属性和方法与外部对象打交道。通过封装，用户可以在不知道对象实现细节的情况下正确地使用对象。

- 继承性（Inheritance） 继承性是指通过对象层次中更抽象的高层对象，可以推导出低层更具体的对象。创建低层对象类型时，子类型继承上级类型中的所有属性和方法，从而不需要重新定义这些属性和方法。子类型也可以重新定义继承的方法，或加入新的属性或方法。例如，“人”-“哺乳动物”-“动物”-“生物”就是一个典型的继承层次。
- 分类性（Classification） Java 和 C++等面向对象的语言将对象类型称为类（class），它提供了利用继承性从上级类创建子类的方法。通常都是先创建一个一般的类（例如“生物”），然后再将该类具体化（例如，创建出“动物”和“植物”两个子类），如此这般构成类的层次结构。
- 多态性（Polymorphism） 所谓多态性就是指可以为同一种方法指定多种实现方案，这些方法通过类型和可接受的参数来区分。例如，可以定义多个 print()方法，用于不同对象类型的打印，也可以定义取不同个参数的 print()方法。

2. 基于对象的 JavaScript

从严格意义上讲，JavaScript 并不是面向对象的编程语言，因为它不支持分类、继承等基本的面向对象特性。不过，JavaScript 确实是基于对象（Object-Based）的编程语言，它支持多种对象类型，并可以实际创建对象实例。

尽管 JavaScript 没有提供完全的面向对象特性，但它提供了一组特别适用于浏览器和服务器脚本的基于对象的特性。这些特性包含一组预定义浏览器对象和服务器对象，以及通过其他对象的属性和方法访问相关对象的功能。此外，JavaScript 提供了一系列内置对象，用于实现一些通用的功能。

从本章开始，我们将逐步介绍这些 JavaScript 对象特性（不包括服务器端对象）。实际上，JavaScript 编程也就是围绕着各种对象进行的，因此有关各种对象的信息正是本书的关键所在。由于本书不打算涉及服务器端编程，因此以后的所有内容都将针对客户端（也就是浏览器端）编程，而不再具体指明。

2.2 使用 JavaScript 对象

在 JavaScript 中可以操作的对象通常包括两种类型：JavaScript 内置对象和浏览器对

象。此外，用户还可以使用自定义对象。在操作这些对象时，可以使用某些特殊的运算符和语句。

2.2.1 内置对象与浏览器对象

JavaScript 内置对象包括一些常用的通用对象，例如数组对象 Array、字符串对象 String、日期对象 Date、数学对象 Math 等。通过使用这些对象，编程者可以获得最基本的一些功能。例如，可以使用 Math 对象的 PI 属性获得圆周率的数值，使用 Date 对象的 getTime 方法获得系统日期和时间。这些对象在 JavaScript 编程中使用非常广泛（类似于其他编程语言中的核心函数），我们将从 2.3 节开始依次介绍相关的具体细节。

浏览器对象是指支持 JavaScript 的浏览器在装入 Web 页面时创建出的多个 JavaScript 对象，可以通过这些对象访问 Web 页面中的各种元素，获得相应的操作效果。

表 2.1 按字母顺序列出了所有的浏览器对象及相关说明，有关的细节将从第 4 章开始介绍。这些细节是进行 JavaScript 编程的关键，因此将占据本书的大部分篇幅。

表 2.1 浏览器对象

对　象	含　义
anchor	代表当前文档中设置了 name 属性的超链接
applet	代表当前文档中的小程序
area	代表客户端图像映射中的区域
button	代表表单中的按钮
checkbox	代表表单中的复选框
document	代表当前窗口中的 HTML 文档
embed	代表当前文档中的嵌入对象
event	代表在浏览器中发生的事件
fileUpload	代表表单中的文件选择框
form	代表当前文档中的表单
frame	代表当前窗口中的框架
hidden	代表表单中的隐藏字段
history	代表浏览器访问过的 URL 历史记录
image	代表当前文档中的图形图像
link	代表当前文档中设置了 href 属性的超链接
location	代表浏览器当前显示网页的 URL
mimeType	代表浏览器支持的特定 MIME 类型信息
navigator	代表当前浏览器
option	代表表单中选项菜单的选项
password	代表表单中的口令框

续表

对　象	含　义
plugin	代表当前浏览器中的插件
radio	代表表单中的单选框
reset	代表表单中的重置按钮
screen	代表用户屏幕
select	代表表单中的选项菜单
submit	代表表单中的提交按钮
text	代表表单中的单行文本框
textarea	代表表单中的多行文本框
window	代表浏览器窗口或窗口中的框架

2.2.2 使用自定义对象

除了使用各种已经规定好的对象以外，JavaScript 还允许用户创建自己的对象。以下以一个具体实例说明如何创建和使用一个“书”对象 Book，在此过程中我们将学到一些面向对象编程技术的细节。

1. 构造函数

要定义一个新的对象，必须定义用于构造该对象的函数。此函数称为构造函数，其名称即为对象的类型名称。构造函数要完成以下两项工作：

- 为对象类型的属性赋值；
- 指出作为对象类型方法的函数。

在此示例中，我们定义对象名为 Book，它包含三个属性：title（书名）、author（作者）、publisher（出版社），以及一个方法：show_book（显示书的属性信息）。

构造函数如下：

```
function Book(title, author, publisher)
{
this.title=title;
this.author=author;
this.publisher=publisher;
this.show_book=show_book;
}
```

构造函数的参数是用来为对象属性赋值的，它们不一定要与对象属性同名，例如，该构造函数也可以写成：

```
function Book(x, y, z)
{
this.title=x;
this.auth r=y;
this.publisher=z;
this.show_book=show_book;
}
```

不过，习惯上将参数名取为对象的属性名。this 关键字表示当前对象，前三条语句的作用是用对象的参数初始化对象属性。

最后一条语句用于初始化对象的方法，它的含义是：将 show_book 这个函数指定为对象 Book 的 show_book 方法。显然，方法的名称与函数的名称也可以不同。

2. 定义方法

虽然我们已经指定了对象的方法，但对应于该方法的函数尚未定义。实际上，在编写程序时，首先应定义方法所对应的函数，然后才能指定方法。

show_book 函数定义如下，其作用只是简单地显示书的属性信息。

```
function show_book()
{
document.write("<H1>书名: "+this.title+"</H1>");
document.write("<H1>作者: "+this.author+"</H1>");
document.write("<H1>出版社: "+this.publisher+"</H1>");
}
```

3. 实例化对象

如果要使用这个新创建的对象，必须将其实例化。也就是说，必须使用构造函数构造一个具体的对象。我们使用 new 操作符构造一个对象的实例，如下所示：

```
MyBook=new Book("JavaScript 实例教程","赵丰年","电子工业出版社");
```

为了显示效果，在构造了对象实例之后，我们可以直接引用该对象的方法和属性，如下所示：

```
MyBook.show_book();
document.write("<H2>谁是"+MyBook.author+"?</H2>")
```

4. 完整代码

此实例的显示效果如图 2.1 所示，完整代码如下：

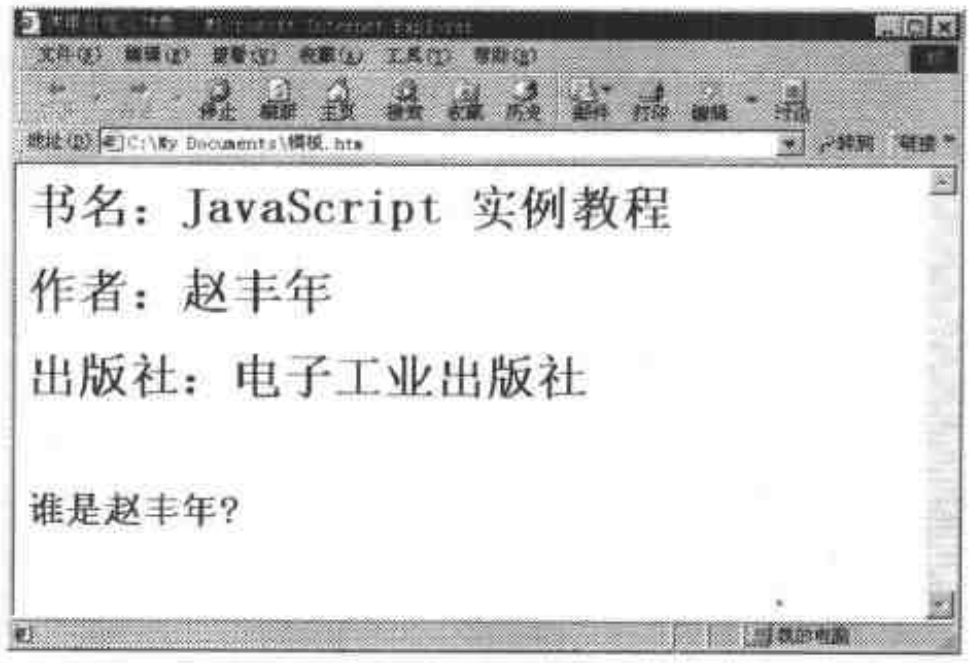


图 2.1　使用自定义对象

```
<HTML>
<HEAD>
<TITLE>使用自定义对象</TITLE>
<SCRIPT LANGUAGE ="JavaScript"TYPE="text/javascript">
<!--
function show_book()  //定义方法所对应的函数
{
document.write("<H1>书名："+this.title+"</H1>");
document.write("<H1>作者："+this.author+"</H1>");
document.write("<H1>出版社："+this.publisher+"</H1>");
}
function Book(title, author, publisher)  //构造函数
{
this.title=title;
this.author=author;
this.publisher=publisher;
this.show_book=show_book;
}
//-->
</SCRIPT>
<HEAD>
<BODY>
<SCRIPT LANGUAGE ="JavaScript"TYPE="text/javascript">
<!--
MyBook=new Book("JavaScript 实例教程","赵丰年","电子工业出版社");   //实例
化对象
```

```
MyBook.show_book();    //引用对象方法
document.write("<BR><BR><H2>谁是"+MyBook.author+"?</H2>")    //引用对象
属性
//-->
</SCRIPT>
</BODY>
</HTML>
```

2.2.3 对象运算符与语句

在 JavaScript 中分别包含两个运算符和两条语句，专门用于对象操作，它们是：new 运算符、delete 运算符、with 语句和 for in 语句。

1. new 运算符

在前面我们已经用到了 new 运算符，它的作用是实例化一个对象。例如，要创建一个新的数组对象，可以使用以下语句：

```
var myArray=new Array( );
```

2. delete 运算符

delete 运算符的作用是从对象中删除属性和方法或从数组中删除元素。例如，以下示例显示了 delete 运算符的作用，效果如图 2.2 所示。

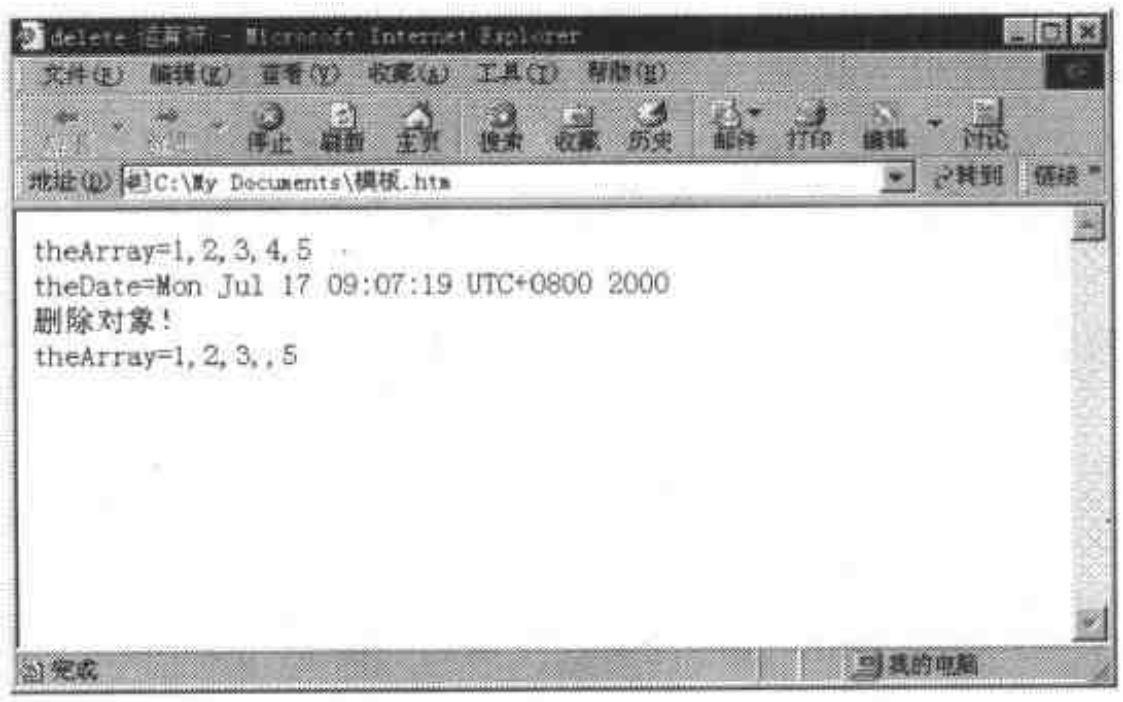


图 2.2 delete 运算符的作用

```
<HTML>
<HEAD>
  <TITLE>delete 运算符</TITLE>
```

```
<HEAD>
<BODY>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
<!--
theArray=new Array(1,2,3,4,5);    //创建一个数组对象
theDate=new Date();    //创建一个日期对象
document.write("theArray="+theArray+"<BR>");    //显示数组对象
document.write("theDate="+theDate);  //显示日期对象
document.write("<BR>删除对象! <BR>")
delete theArray[3];    //删除数组中的第 4 个元素
delete theDate;    //删除该日期对象
document.write("theArray="+theArray);    //第 4 个元素显示为空
document.write("theDate="+theDate);  /* 执行到此语句时，浏览器会显示出错信
息，说明 theDate 未定义——单击浏览器左下角的出错图标即可看到出错信息*/
-->
</SCRIPT>
</BODY>
</HTML>
```

说明： 在实际应用中，使用 delete 运算符的情况很少。

3. with 语句

如果在程序中要使用某个对象的多个属性和方法，则可以考虑使用 with 语句。将需要使用其属性和方法的对象用 with 语句包含起来，然后在代码块中就可以不必再引用该对象而直接使用它的方法和属性。

使用 with 语句的语法如下：

```
with(object)
{
  statement
}
```

例如，以下示例中通过 with 语句说明要使用的对象是 document，因此在代码块中引用前景色属性 fgColor 和背景色属性 bgColor 时不再使用完整的引用，显示效果如图 2.3 所示。

```
<HTML>
<HEAD>
  <TITLE>使用 with 语句</TITLE>
```

```
<HEAD>
<BODY>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
<!--
with(document)
{
    document.write("文档的前景色为" + fgColor + ".<BR>")
    document.write("文档的背景色为" + bgColor + ".")
}
//-->
</SCRIPT>
</BODY>
</HTML>
```

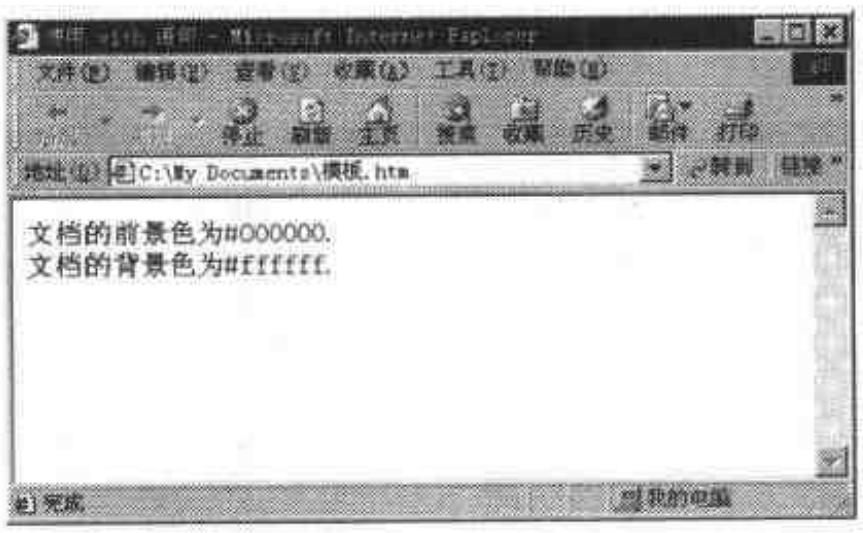


图 2.3 with 语句使用示例

4. for in 语句

for in 语句是一种特殊的循环语句，用于遍历一个对象的所有属性，对于每一个属性，循环体内的语句被执行一次。

for in 语句的语法如下：

```
for (variable in object)
{
  statement;
}
```

例如，以下示例用 for in 语句遍历了 document 对象的所有属性，并将它们显示出来，效果如图 2.4 所示。

```
<HTML>
```

```
<HEAD>
  <TITLE>使用 for in 语句</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
<!--
document.write("<H2>document 对象的属性如下：</H2>")
for(var i in document)
{  document.write(i+"<BR>")
}
//-->
</SCRIPT>
</BODY>
</HTML>
```

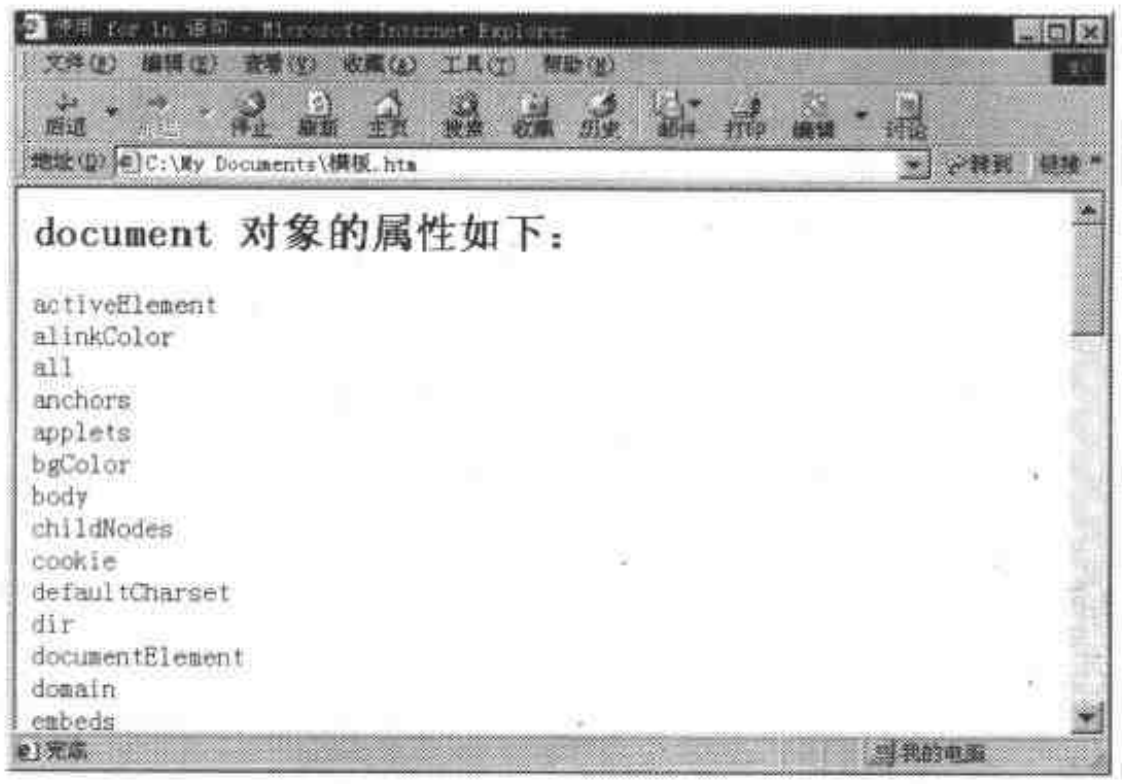


图 2.4　for in 语句使用示例

2.3　Array 对象

2.3.1　创建数组

Array 对象也就是数组对象，用于实现任何一门编程语言中最常见的一种数据结构——数组。

1. 构造函数

Array 对象的构造函数有三种，分别用不同的方式构造一个数组对象：

- var variable = new Array();
- var varialble = new Array(int);
- var variable = new (arg1,arg2,...argN)

使用第一种构造函数创建出的数组长度为 0，当具体为其指定数组元素时，JavaScript 自动延伸数组的长度。例如，可以定义数组：

```
order=new Array();
```

然后当具体为数组元素赋值时，数组自动扩充。对应于刚才的 order 数组，如果指定：

```
order[20]="test20";  //在 JavaScript 中用 [ ] 进行数组下标引用
```

则 JavaScript 自动将数组扩充为 21 个元素，前 20 个元素（order[0]~order[19]）被初始化为 null，第 21 个元素为"test20"。如果再次指定：

```
order[30]="test30";
```

则 JavaScript 自动继续将数组扩充为 31 个元素，并将 order[21]~order[29]初始化为 null，而 order[30]赋值为"test30"。

以上过程用 JavaScript 程序表示如下，效果如图 2.5 所示。

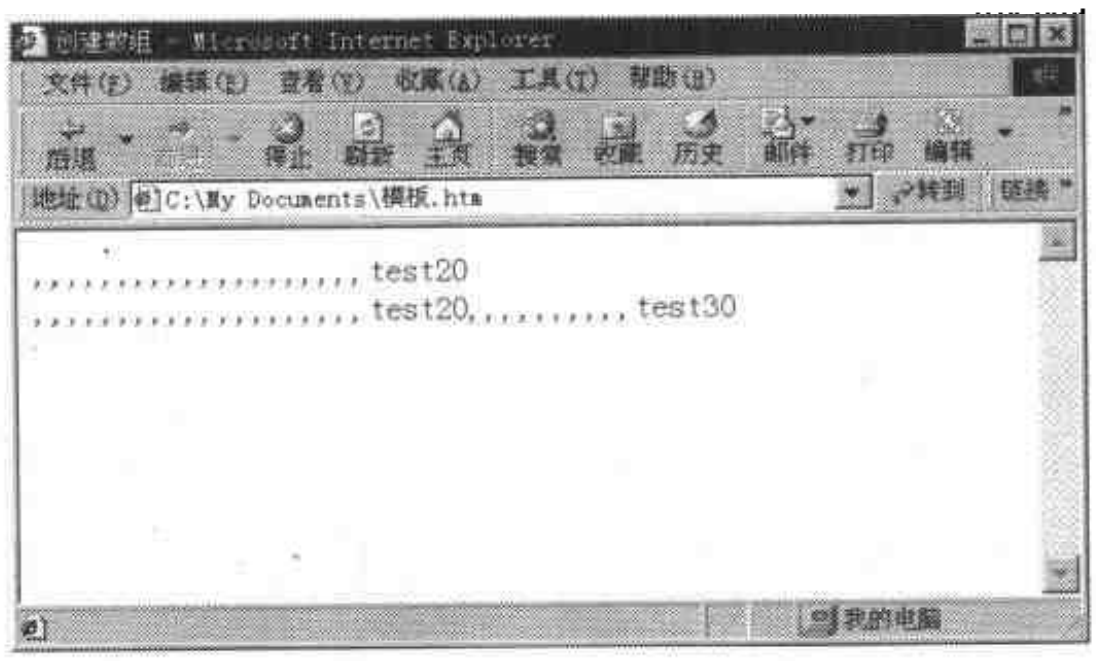


图 2.5 动态扩充数组

```
<HTML>
<HEAD>
  <TITLE>创建数组</TITLE>
<HEAD>
<BODY>
```

```
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
<!--
var order=new Array();
order[20]="test20";
document.write(order+"<BR>");
order[30]="test30";
document.write(order);
//-->
</SCRIPT>
</BODY>
</HTML>
```

说明： JavaScript 中的数组与 C 等语言一样，都是从 0 下标开始的。也就是说，数组的第一个元素是 arrayName[0]。

使用第二种构造函数时应使用数组的长度作为参数，此时创建出一个长度为 int 的数组，但并没有指定具体的元素。同样，当具体指定数组元素时，数组的长度也可以动态更改。

例如，myArray=new Array(10)创建出一个长度为 11 的数组，如果使用赋值语句 myArray[20]=20 为数组元素赋值，则数组自动扩充为长度为 21。

使用第三种构造函数时直接使用数组元素作为参数，此时创建出一个长度为 N 的数组，同时数组元素按照指定的顺序赋值。在构造函数使用数组元素作为参数时，参数之间必须使用逗号分隔开，并且不允许省略任何参数。例如，以下两种数组定义都是错误的：

```
myArray=new Array(0,,2,3,4)
myArray=new Array(0,1,2,3,)
```

而正确的定义为：

```
myArray=new Array(0,1,2,3,4)
```

除了使用以上三个构造函数定义数组以外，还可以直接用 [ ] 运算符定义数组，如下所示：

```
var myArray=[0,1,2,3,4]
```

该定义的效果与 var myArray=new Array(0,1,2,3,4)一模一样。

2. 数组元素

从前面的数组定义中已经可以看出，数组元素可以是整数，也可以是字符串。实际上，

JavaScript 并不对数组元素的值作限制，它们可以是任意类型。例如，以下数组包含各种不同类型的数据：

```
var myArray=new Array(0,1,true,null,"great");
```

该数组有 5 个元素，分别如下：

```
myArray[0]=0;
myArray[1]=1;
myArray[2]=true;
myArray[3]=null;
myArray[4]="great";
```

数组元素不但可以是其他数据类型，而且可以是其他数组或对象。例如，以下示例构造出了一个二维数组并将其元素在表格中显示，效果如图 2.6 所示。

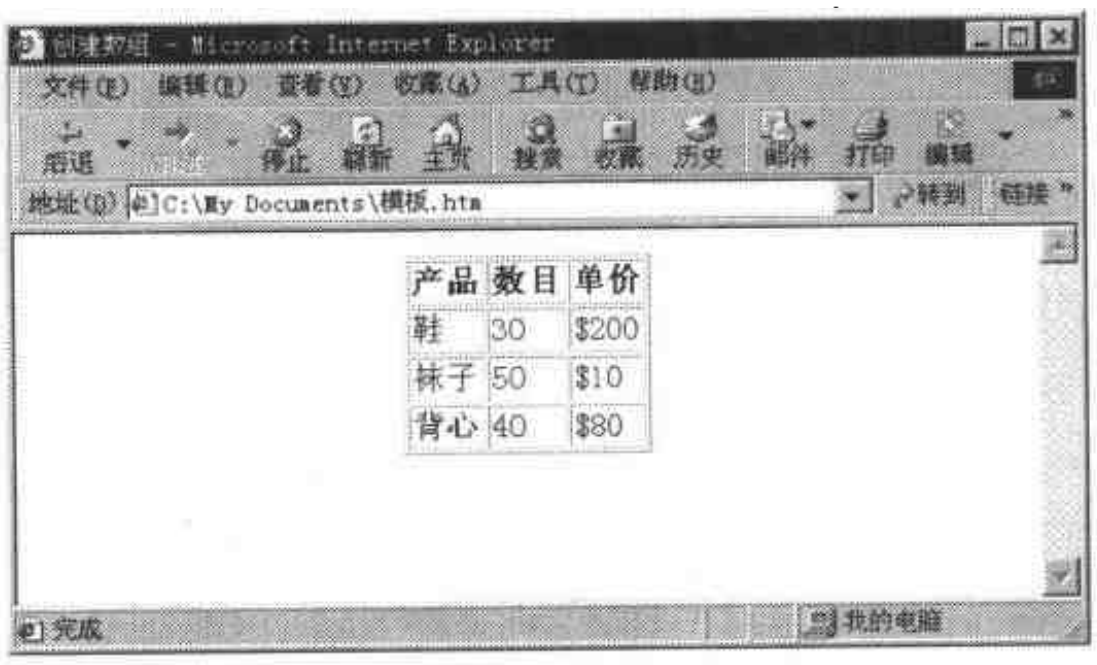


图 2.6 使用二维数组

```
<HTML>
<HEAD>
  <TITLE>创建数组</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE = "JavaScript"TYPE='text/javascript">
<!--
var order=new Array();
order[0]=new Array("鞋","30","$200");
order[1]=new Array("袜子","50","$10");
order[2]=new Array("背心","40","$80");
document.write("<table border align=center>")
```

```
document.write("<th>产品</th><th>数目</th><th>单价</th>")
for(i=0;i<order.length;i++)
{
document.write("<tr>")
for(j=0;j<order[0].length;j++)
{
  document.write("<td>"+order[i][j]+"</td>")
}
document.write("</tr>")
}
document.write("</table>")
//-->
</SCRIPT>
</BODY>
</HTML>
```

2.3.2 Array 对象的属性和方法

Array 对象的属性和方法如表 2.2 所示。

表2.2 Array对象的属性和方法

类　型	项　目	说　明
属性	length	数组中元素的个数
	prototype	用于在 Array 对象中添加新的属性和方法
方法	concat(arg1,...argN)	将参数中的元素合并到数组中，但并不改变数组原来的属性
	join(string)	将数组中的所有元素合并为一个字符串，如果指定参数，则该参数将作为字符串中分开各数组元素的分隔符
	reverse()	颠倒数组中元素的排序
	slice(start, stop)	返回数组中的一部分。start 参数表示数组的开始位置，负数可用来表示倒数的位置，例如 -2 表示倒数第 2 个元素。stop 参数表示数组的结束位置，同样也可以用负数，如果不指定 stop 参数，则新数组就会包含原数组中从 start 开始（不包含 start 所在的元素）一直到结束的数组元素
	sort(function)	对数组元素进行排序。如果不指定参数，则 JavaScript 将元素转换为字符串，然后按字母顺序排序。如果指定了参数，则该参数必须是自定义的排序函数，同时该排序函数必须遵循一定的规则。也就是说，如果指定排序函数，则它应有两个参数 arg1 和 arg2，返回值为负整数（arg1<arg2 时），0（arg1=arg2 时）或正整数（arg1>arg2 时）
	toString()	返回一个字符串，该字符串包含数组中的所有元素，各个元素间用逗号分隔
	valueOf()	返回对象的原始值

1. 示例 1

本示例说明 prototype 属性的用法（请注意注释语句），效果如图 2.7 所示。

图 2.7 使用 prototype 属性

```
<HTML>
<HEAD>
  <TITLE>使用 prototype 属性</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
<!--
function pop()  //为 Array 对象定义一个新方法
{
if(this.length)  //如果数组不为空
{
var last=this[this.length-1];
this.length=this.length-1;  //数组长度减 1，相当于删除最后一个元素
return(last);  //返回已经删除的最后一个数组元素
}
}
Array.prototype.pop=pop;  //用 prototype 属性为 Array 对象指定一个新方法 pop
var color=new Array("red","green","blue");
document.write("color 数组的初始内容："+color+"<br>");
color.pop();  //调用 pop 方法
document.write("使用 pop 方法后 color 数组的内容："+color);
//-->
</SCRIPT>
</BODY>
</HTML>
```

2. 示例 2

本示例说明各种 Array 方法的使用（注意各方法对原数组的影响），效果如图 2.8 所示。

图 2.8　Array 对象的方法示例

```
<HTML>
<HEAD>
  <TITLE>Array 对象的方法示例</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
<!--
var colorArray=new Array("red","green","blue");
document.write("初始数组内容: colorArray="+colorArray+"<p>");
colorArray.concat("yellow","purple","gray");  //
document.write("执行 concat('yellow','purple','gray')后数组的内容:
colorArray = " + colorArray + "<br>");
document.write("colorArray.concat('yellow','purple','gray')="+colorA
rray.concat("yellow","purple","gray")+"<p>")
colorArray.join(":");  //
document.write("执行 join(':')后的数组内容: colorArray="+colorArray+"
```

```
<br>")
document.write("colorArray.join( :')="+colorArray.join(":")+"<p>")
colorArray.reverse();  //————————————————————
document.write("执行 reverse()后的数组内容: colorArray="+colorArray+
"<br>")
document.write("colorArray.reverse()-"+colorArray.reverse()+"<p>')
//执行了两次 reverse()!因此又恢复了原来的顺序!
colorArray.slice(-2,-1);  //————————————————————
document.write("执行 slice(-2,-1)后的数组内容: colorArray="+colorArray+
"<br>")
document.write("colorArray.slice(-2,-1)-"+colorArray.slice(-2,
1)+"<p>")
colorArray.sort();  //————————————————————
document.write("执行 sort()后的数组内容: colorArray="+colorArray+"<p>")
document.write("colorArray.sort()="+colorArray.sort()+"<p>")
colorArray.toString();  //————————————————————
document.write("执行 toString()后的数组内容: colorArray="+colorArray+
"<br>")
document.write("colorArray.toString()-"+colorArray.toString()+"<p>")
colorArray.valueOf();  //————————————————————
document.write("执行 valueOf()后的数组内容: colorArray="+colorArray+
"<br>")
document.write("colorArray.valueOf()="+colorArray.valueOf(":")+"<p>"
)
//-->
</SCRIPT>
</BODY>
</HTML>
```

2.4 Boolean 对象

Boolean 对象可以将布尔值当作对象访问，它支持以下属性和方法：

- prototype 属性　用于在 Boolean 对象中添加新的属性和方法；
- toString()方法　返回表示 Boolean 对象的布尔值字符串（"true"或"false"）；
- valueOf()方法　返回对象的原始值。

创建 Boolean 对象的构造函数要求一个参数，该参数可以是 true 或 false，也可以是任意类型的值，但值 null、""、NaN、0 将变成 false，其他值变成 true。

以下示例显示了如何使用 Boolean 对象，显示效果如图 2.9 所示。

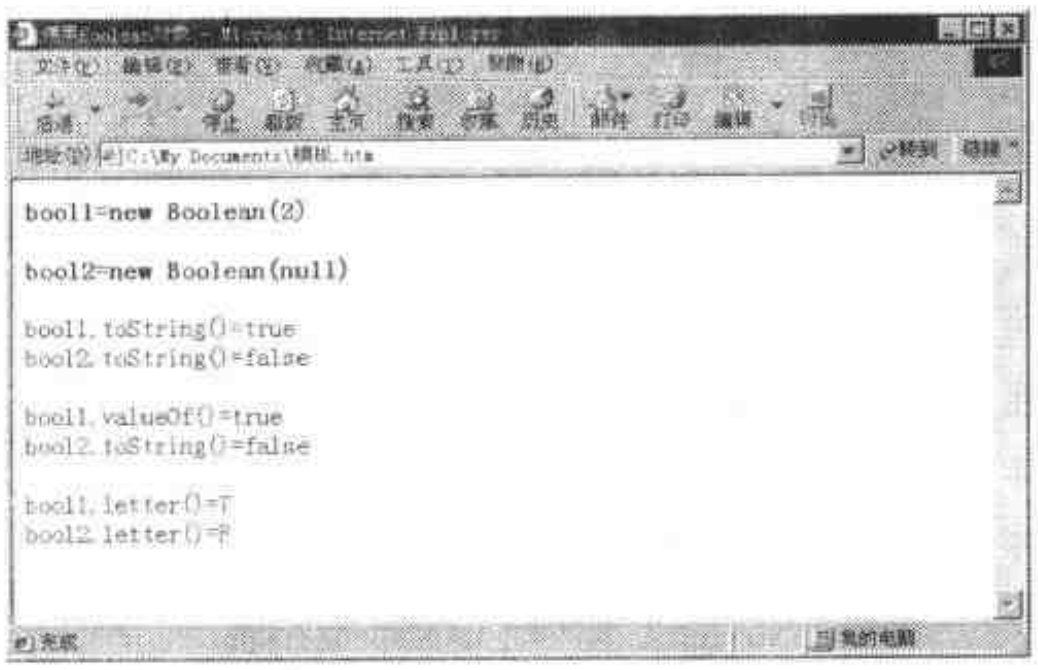


图 2.9　使用 Boolean 对象

```
<HTML>
<HEAD>
  <TITLE>使用 Boolean 对象</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript"TYPE="text/javascript">
<!--
function letter()  //自定义的新方法
{
if(this==true)
 return('T');
else
 return('F');
}

Boolean.prototype.letter=letter;  //使用 prototype 属性指定新方法

bool1=new Boolean(2);  //创建 Boolean 对象
bool2=new Boolean(null);

document.write("<H4>bool1=new Boolean(2)</H4>");
document.write("<H4>bool2=new Boolean(null)</H4>");

document.write("bool1.toString()="+bool1.toString()+"<BR>");
document.write("bool2.toString()="+bool2.toString()+"<P>");
```

```
document.write("bool1.valueOf()="+bool1.valueOf()+"<BR>");
document.write("bool2.toString()="+bool2.toString()+"<P>");

document.write("bool1.letter()="+bool1.letter()+ "<BR>");
document.write("bool2.letter()="+bool2.letter());
//-->
</SCRIPT>
</BODY>
</HTML>
```

2.5 Date 对象

2.5.1 创建日期对象

Date 对象也就是日期对象，它可以表示从年到毫秒的所有时间和日期。如果在创建 Date 对象时就给定了参数，则新对象就表示指定的日期和时间；否则新对象就被设置为当前日期。

创建日期对象可以使用以下 4 种构造函数中的一种：

- var variable=new Date()
- var variable=new Date(milliseconds)
- var variable=new Date(string)
- var variable=new Date(year, month, day, hours, minutes, seconds, milliseconds)

第一种构造函数使用当前时间和日期创建 Date 实例；第二种构造函数使用从 GMT（格林威治平均时间）时间 1970 年 1 月 1 日凌晨到期望日期和时间之间的毫秒来创建 Date 实例；第三种构造函数使用特定的表示期望日期和时间的字符串来创建 Date 实例，该字符串的格式应该与 Date 对象的 parse 方法相匹配，可以是"month day, year hours:minutes:seconds"等格式；第四种构造函数使用年、月、日、小时、分钟、秒、毫秒的形式创建 Date 实例，其中年和月是必须的参数，其他参数可选，注意在指定月份时，0 表示 1 月，依次类推，11 表示 12 月。

例如，以下示例显示了如何使用不同的构造函数创建 Date 对象，显示效果如图 2.10 所示（请注意显示的格式）。

图 2.10　创建 Date 对象

```
<HTML>
<HEAD>
  <TITLE>创建 Date 对象</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE='JavaScript'TYPE='text/javascript'>
<!--
document.write("<H3>四种不同的创建 Date 对象的方式：</H3>")

today=new Date();
document.write("Date()="+today+"<p>");

newDay1=new Date(20000);
document.write("Date(2000)="+newDay1+'<p>');

newDay2=new Date("15 Jul,2000 20:20:20")
document.write('Date("15 Jul,2000 20:20:20")='+newDay2+'<p>');

newDay3=new Date(2000,7,16,21,34)
document.write('Date(2000,7,16,21,34)='+newDay3+'<p>');
//-->
</SCRIPT>
</BODY>
</HTML>
```

2.5.2 Date 对象的属性和方法

Date 对象的属性和方法如表 2.3 所示。

表 2.3 Date 对象的属性和方法

类 型	项 目	说 明
属性	prototype	用于在 Date 对象中添加新的属性和方法
方法	getDate()	返回一个整数，表示一月中的某一天（1~31）
	getDay()	返回一个整数，表示星期中的某一天（0~6，0 表示星期日，6 表示星期六）
	getFullyear()	返回表示当地时间的年份的 4 位数
	getHours()	返回表示当前时间中的小时部分的整数（0~23）
	getMilliseconds()	返回表示当前时间中的毫秒部分的整数（0~999）
	getMinutes()	返回表示当前时间中的分钟部分的整数（0~59）
	getMonth()	返回表示当前日期中月的整数（0~11）
	getSeconds()	返回表示当前时间中的秒部分的整数（0~59）
	getTime()	返回从 GMT 时间 1970 年 1 月 1 日凌晨到当前 Date 对象指定的时间之间的毫秒数
	getTimezoneoffset()	返回以 GMT 为基准的时区偏差，以分钟计量
	getUTCDate()	返回表示转换成世界时间的月中的某一天（1~31）
	getUTCDay()	返回表示转换成世界时间的星期中的某一天（0~6，0 表示星期日，6 表示星期六）
	getUTCFullyear()	返回表示转换成世界时间的年的 4 位数
	getUTCHours()	返回用世界时间表示的时间的小时数（0~23）
	getUTCMilliseconds()	返回用世界时间表示的时间的毫秒数（0~999）
	getUTCMinutes()	返回用世界时间表示的时间的分钟数（0~59）
	getUTCMonth()	返回用世界时间表示的日期的月数（0~11）
	getUTCSeconds()	返回用世界时间表示的时间的秒数（0~59）
	getYear()	返回日期对象中的年份，用 2 位或 4 位数字表示
	parse(date)	返回以参数 Date 表示的日期和时间与 GMT 时间 1970 年 1 月 1 日凌晨之间的毫秒数。注意此方法与参数中指定的日期相联系，而不是与对象中的日期相联系。参数 Date 应使用 Date.toGMTString() 方法所写的如下格式的字符串（可以省略其中的部分信息）： Mon, 17 Jul 2000 15:30:52 UTC
	setDate(day)	将日期对象中的日期设置为参数 day，day 为一个 1~31 的整数。该方法返回在日期调整后从 GMT 时间 1970 年 1 月 1 日凌晨到 Date 对象所确定的日期和时间之间的毫秒数（注意：其他 set 开头的方法均返回相应毫秒数，后面不再重复）
	setFullyear(year)	将日期对象中的年设置为参数 year 表示的 4 位整数
	setHours(hour)	将日期对象中的小时数设置为参数 hour 所表示的 0~23 的一个整数

续表

类　型	项　目	说　明
	setMilliseconds(milliseconds)	将日期对象中的毫秒数设置为参数 milliseconds 所表示的一个 0~999 的整数
	setMinutes(minutes)	将日期对象中的分钟数设置为参数 minutes 所表示的 0~59 的一个整数
	setMonth(month)	将日期对象中的月份数设置为参数 month 所表示的一个 0~11 的整数
	setSeconds(seconds)	将日期对象中的秒数设置为参数 seconds 所表示的 0~59 的一个整数
	setTime(milliseconds)	将日期对象的时间设置为参数 milliseconds 表示的整数，参数 milliseconds 表示从 GMT 时间 1970 年 1 月 1 日凌晨到要设定时间之间的毫秒数
	setUTCDate(day)	将日期对象中的日期设置为参数 day 表示的一个 1~31 的整数（世界时间）
	setUTCFullyear(year)	将日期对象中的年份设置为参数 year 表示的 4 位整数（世界时间）
	setUTCHours(hours)	将日期对象中的小时数设置为参数 hours 所表示的 0~23 的一个整数（世界时间）
	setUTCMilliseconds(milliseconds)	将日期对象中的毫秒数设置为参数 milliseconds 所表示的一个 0~999 的整数（世界时间）
	setUTCMinutes(minutes)	将日期对象中的分钟数设置为参数 minutes 所表示的 0~59 的一个整数（世界时间）
	setUTCMonth(month)	将日期对象中的月份数设置为参数 month 所表示的一个 0~11 的整数（世界时间）
	setUTCSeconds(seconds)	将日期对象中的秒数设置为参数 seconds 所表示的 0~59 的一个整数（世界时间）
	setYear(year)	将日期对象中表示的年份设置为参数 year 指定的值，这个参数可以是 4 位或 2 位整数
	toGMTString()	返回表示日期对象的世界时间的字符串，日期在转换成字符串之前转换到 GMT 零时区
	toLocalString()	返回一个表示日期对象所表示的当地时间的字符串
	toString()	返回一个表示日期对象的字符串
	toUTCString()	返回一个表示日期对象所表示的世界时间的字符串
	UTC(year,month,day,hours,Minutes,seconds,milliseconds)	与相应的构造函数 Date 类似，不过是用世界时间创建日期。同样，year 和 month 是必要参数，其他参数是可选参数
	valueOf()	返回对象的原始值

说明： UTC 是指同一协调时间，是世界时间标准定义的时间。

以下的几个示例显示了 Date 对象的常见用法。

1. 示例 1

本示例使用 prototype 属性和 getDay( )方法为 Date 对象创建了一个新方法，用于显示

日期对象所对应星期几的字符串，代码如下：

```
<HTML>
<HEAD>
  <TITLE>创建 Date 对象的新方法</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript"TYPE="text/javascript">
<!--
function getDayString()
{var day;
switch(this.getDay())  //获得星期的数字
{
case 0:
 day="星期日";  break;
case 1:
 day="星期一";  break;
case 2:
 day="星期二";  break;
case 3:
 day="星期三";  break;
case 4:
 day="星期四";  break;
case 5:
 day="星期五";  break;
case 6:
 day="星期六";  break;
default:
 day="没有这个星期数!"
}
return(day);
}
Date.prototype.getDayString=getDayString;  //指定新方法
today=new Date();
document.write("今天是"+today.getDayString()+"<p>");
//-->
</SCRIPT>
</BODY>
</HTML>
```

显示效果如图 2.11 所示。

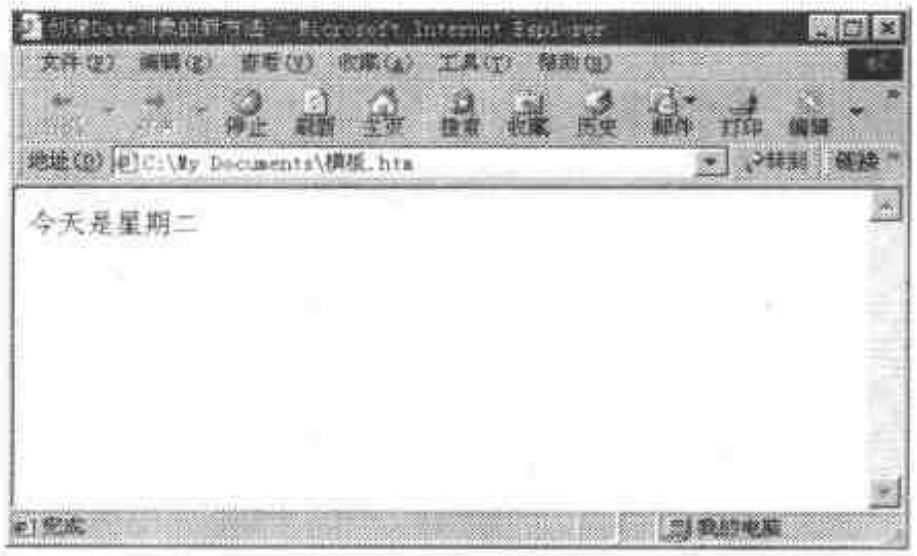


图 2.11　创建新方法

2. 示例 2

本示例在网页中用浏览者容易接受的格式显示当前的日期和时间，显示效果如图 2.12 所示，代码如下：

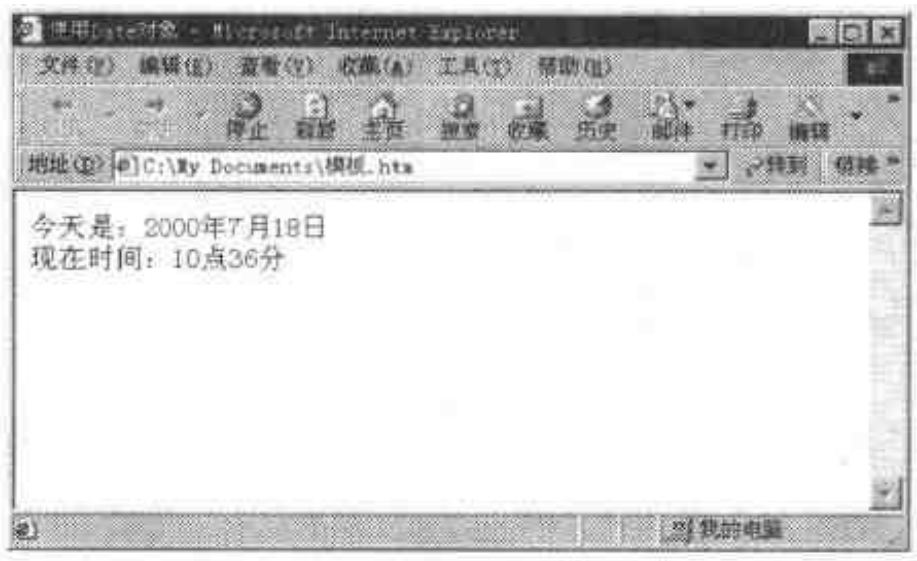


图 2.12　显示当前日期和时间

```
<HTML>
<HEAD>
  <TITLE>使用 Date 对象</TITLE>
<HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript"TYPE="text/javascript">
<!--
today=new Date();  //获取当前日期
with(today)
{
  month=getMonth()+1;    //getMonth() 方法返回的月份从 0 开始!
```

```
  document.write("今天是： "+getYear()+"年 "+month+"月 "+getDate()+"日
<BR>");
  document.write("现在时间："+getHours()+"点"+getMinutes()+"分")
}
//-->
</SCRIPT>
</BODY>
</HTML>
```

3. 示例3

在网页的顶端显示一条简单的欢迎信息是一种常见的功能，以下就是实现这一功能的代码，效果如图2.13所示。

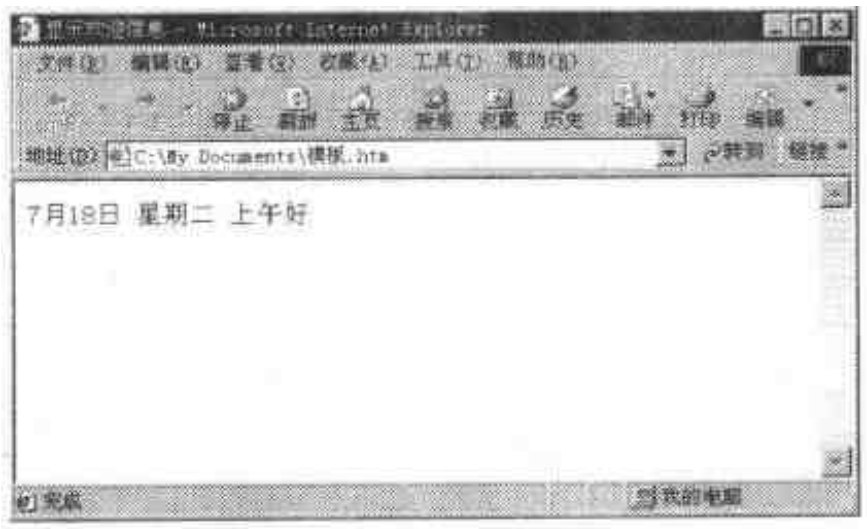


图2.13 显示欢迎信息

```
<HTML>
<HEAD>
  <TITLE>显示欢迎信息</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
<!--
myDate = new Date();  //创建一个日期对象。
myHour = myDate.getHours();  //获得当前的小时数。

    if(myHour<6)  //根据小时数显示不同的欢迎信息。
      welcomeString="凌晨好";
    else if(myHour<9)
      welcomeString="早上好";
    else if(myHour<12)
      welcomeString="上午好";
    else if(myHour<14)
```

```
      welcomeString="中午好";
    else if(myHour<17)
      welcomeString="下午好";
    else if(myHour<19)
      welcomeString="傍晚好";
    else if(myHour<22)
      welcomeString="晚上好";
    else
      welcomeString="夜里好";
arrayDay=["日" , "一" , "二" , "三" , "四" , "五" , "六" ]; /*定义一个字
符数组以显示星期数*/
document.write((myDate.getMonth()+1) + "月" + myDate.getDate() + "日
 ");
document.write("星期" + arrayDay[myDate.getDay()] +" ");
document.write(welcomeString);
  ->
</SCRIPT>
</BODY>
</HTML>
```

2.6 Function 对象

Function 对象可以将函数作为对象访问，并可以在脚本执行期间动态生成和调用函数。Funtion 对象的属性和方法如表 2.4 所示。

表2.4 Function对象的属性和方法

类　型	项　目	说　明
属性	arguments	表示函数参数的数组
	caller	表示调用当前正在执行的函数
	prototype	为对象添加新的属性和方法
方法	apply()	用于将一个对象中的方法应用到另一个对象
	call(this) call(this,arg1,arg2,...argN)	允许调用另外一个对象的方法
	toString()	用于将一个函数转换为字符串

使用 Function 对象构造函数时，应提供函数参数和函数体，格式如下：

```
variable=new Function ("arg1","arg2"..."argN","body")
```

例如，以下示例显示了如何使用 Function 对象，效果如图 2.14 所示。

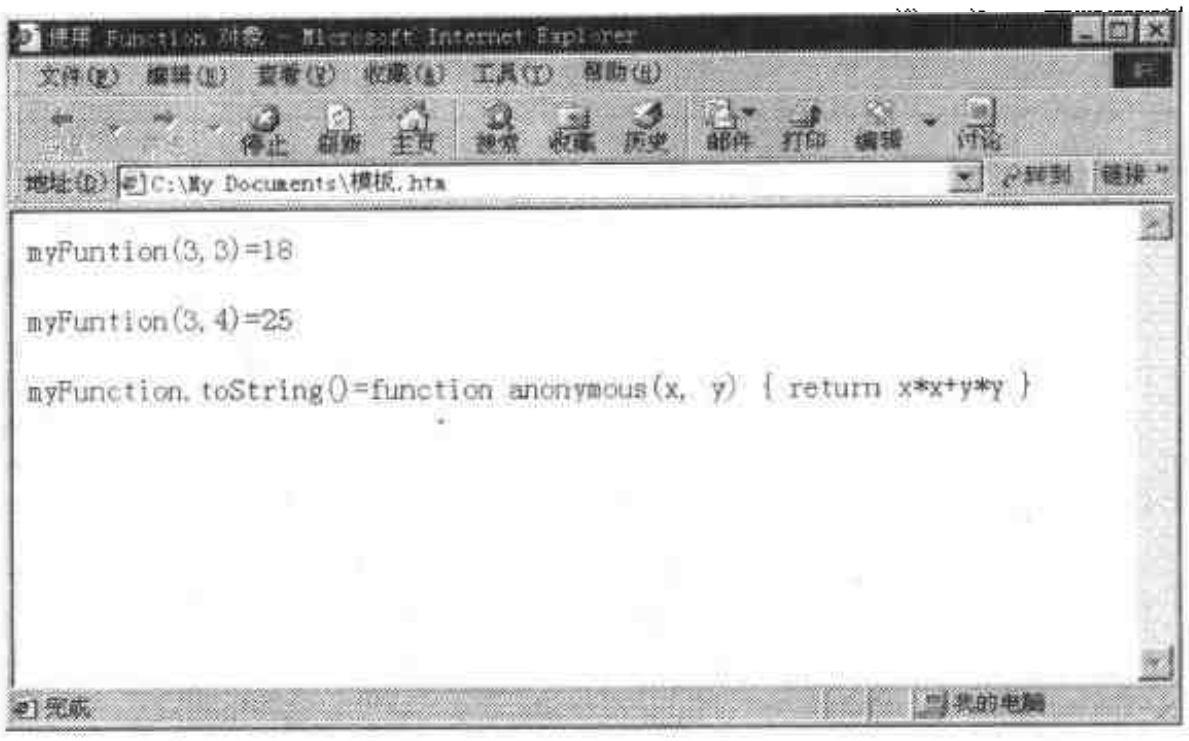


图 2.14 使用 Function 对象

```
<HTML>
<HEAD>
  <TITLE>使用 Function 对象</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
<!--
myFunction = new Function("x","y","return x*x+y*y");  //创建一个函数对象。
document.write("myFuntion(3,3)="+myFunction(3,3));
document.write("myFuntion(3,4)="+myFunction(3,4));
document.write("myFunction.toString()="+myFunction.toString());
-->
</SCRIPT>
</BODY>
</HTML>
```

2.7 Global 对象

Global 对象定义了 JavaScript 中的全局变量和函数，它的属性和方法如表 2.5 所示。

表2.5 Global对象的属性和方法

类 型	项 目	说 明
属性	Infinity	表示正无穷大的关键字
	NaN	表示一个变量不等于任何数（即非数，Not a Number）
方法	escape(string) escape(expression)	escape() 函数以一个 string 对象或表达式为参数并返回一个 string 对象。参数指定的字符串中的所有非字母字符被转换成以 XX%表示的等价数字，XX 是一个表示非字母字符的十六进制数

续表

类　型	项　目	说　明
	eval(command) eval(string)	eval() 函数将通过参数传入的一个包含 JavaScript 语句的字符串作为一个 JavaScript 源代码执行。eval() 返回执行 JavaScript 语句的返回值
	isFinite(variable)	isFinite() 函数用于确定一个变量是否有界，如果有界则返回 true，否则返回 false
	isNaN(variable)	isNaN() 函数用于确定一个变量是否是 NaN，如果是，则返回 true，否则返回 false
	parseFloat(string)	parseFloat( ) 函数用于将字符串开头的整数或浮点数分解出来，若字符串不是以数字开头，则返回 NaN
	parseInt(string,radix)	parseInt( ) 函数与 parseFloat( ) 函数类似，用于将字符串开头的整数分解出来，若字符串不是以数字开头，则返回 NaN。参数 radix 可选，用来表示字符串所表示的数的基数（二进制为 2，十六进制为 16 等）
	unescape(string)	unescape( ) 函数将参数传递来的字符串中的十六进制码转换成 ASCII 码并返回，它完成 escape( ) 函数的逆操作

Internet Explorer 和 Navigator 都实现了 Global 对象，但不能显式生成该对象，而是直接以全局变量和函数的方式访问其属性和方法。在第 1 章的 1.6.2 节“JavaScript 全局函数”中已经说明了各种全局函数的使用，以下再举一个例子说明如何使用最常用的 eval()方法，效果如图 2.15 所示，代码如下：

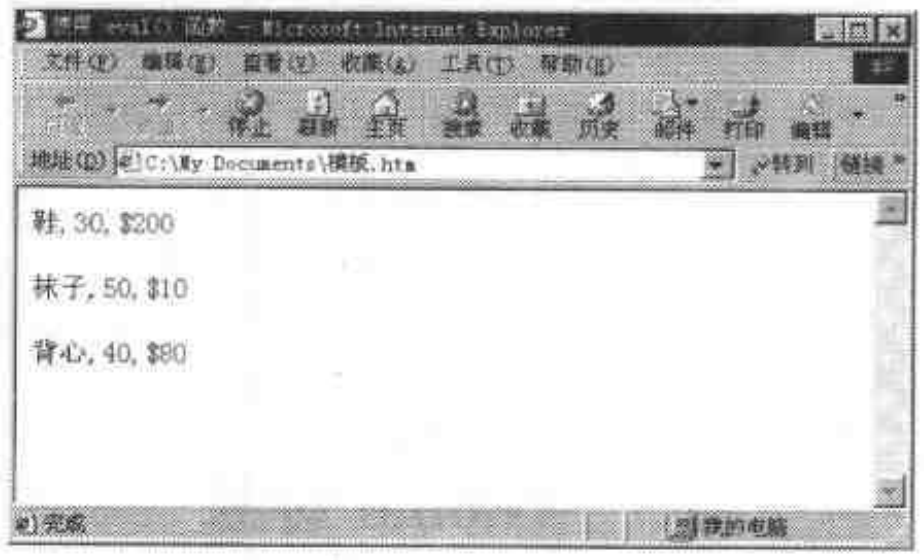


图 2.15　使用 eval()方法

```
<HTML>
<HEAD>
  <TITLE>使用 eval() 函数</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
<!--
```

```
myArray1=new Array("鞋","30","$200");
myArray2=new Array("袜子","50","$10");
myArray3=new Array("背心","40","$80");
for(i=1;i<4;i++)
  document.write(eval("myArray"+i));  //这种用法非常有用，请读者注意领会!
-->
</SCRIPT>
</BODY>
</HTML>
```

2.8 Math 对象

2.8.1 Math 对象的属性与方法

Math 对象包含用来进行数学计算的属性和方法，其属性也就是标准数学常量，其方法则构成了数学函数库。Math 对象可以在不使用构造函数的情况下使用，并且所有的属性和方法都是静态的。Math 对象的属性和方法如表 2.6 所示。

表 2.6 Math 对象的属性和方法

类 型	项 目	说 明
属性	E	欧拉常数，约为 2.718
	LN10	10 的自然对数（自然对数以欧拉常数为底），约为 2.302
	LN2	2 的自然对数，约为 0.693
	LOG10E	以 10 为底的欧拉常数 E 的对数，约为 0.434
	LOG2E	以 2 为底的欧拉常数 E 的对数，约为 1.442
	PI	圆周率常数，约为 3.14159
	SQRT1_2	0.5 的平方根，约为 0.707
	SQRT2	2 的平方根，约为 1.414
方法	abs(num)	返回参数 num 的绝对值
	acos(num)	返回参数 num 的反余弦，其值在 0 到 PI 之间，用弧度计量
	asin(num)	返回参数 num 的反正弦，其值在-PI/2 到 PI/2 之间，用弧度计量
	atan(num)	返回参数 num 的反正切，其值在-PI/2 到 PI/2 之间
	atan2(num1,num2)	返回坐标(num1,num2)对应的极坐标角度，其值在-PI 到 PI 之间
	ceil(num)	返回大于或等于参数 num 的最小整数
	cos(num)	返回参数 num 的余弦，其值在-1 到 1 之间
	exp(num)	返回欧拉常数 E 的 num 次方
	floor(num)	返回大于或等于参数 num 的最大整数
	log(num)	返回参数 num 的自然对数

续表

类　型	项　目	说　明
	max(num1,num2)	返回参数 num1 和 num2 中较大的一个
	min(num1,num2)	返回参数 num1 和 num2 中较小的一个
	pow(num1,num2)	返回 num1 的 num2 次方
	random()	返回一个 0 到 1 之间的随机数
	round(num)	返回最接近参数 num 的整数。如果该数的小数部分大于等于 0.5，则取大于它的整数，否则取小于它的整数
	sin(num)	返回参数 num 的正弦，结果在-1 到 1 之间
	sqrt(num)	返回参数 num 的平方根
	tan(num)	返回参数 num 的正切
	toString()	返回表示该对象的字符串

2.8.2　示例

1. 示例 1

以下示例列出了 Math 对象的各种属性值（直接引用而无须创建对象），也就是各种数学常数的值，效果如图 2.16 所示。

图 2.16　Math 对象的属性示例

```
<HTML>
<HEAD>
  <TITLE>使用 Math 对象的属性</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
```

```
<!--
document.write("以下是各种数学常数的值：<P>");
document.write("<TABLE>")
document.write("<TR><TD>E<TD> = "+Math.E);
document.write("<TR><TD>LN10<TD> = "+Math.LN10);
document.write("<TR><TD>LN2<TD> = "+Math.LN2);
document.write("<TR><TD>LOG10E<TD> = "+Math.LOG10E);
document.write("<TR><TD>LOG2E<TD> = "+Math.LOG2E);
document.write("<TR><TD>PI<TD> = "+Math.PI);
document.write("<TR><TD>SQRT1_2<TD> = "+Math.SQRT1_2);
document.write("<TR><TD>SQRT2<TD> = "+Math.SQRT2);
document.write("</TABLE>")
-->
</SCRIPT>
</BODY>
</HTML>
```

2. 示例 2

以下示例显示了 Math 对象各种方法的使用效果，如图 2.17 所示。

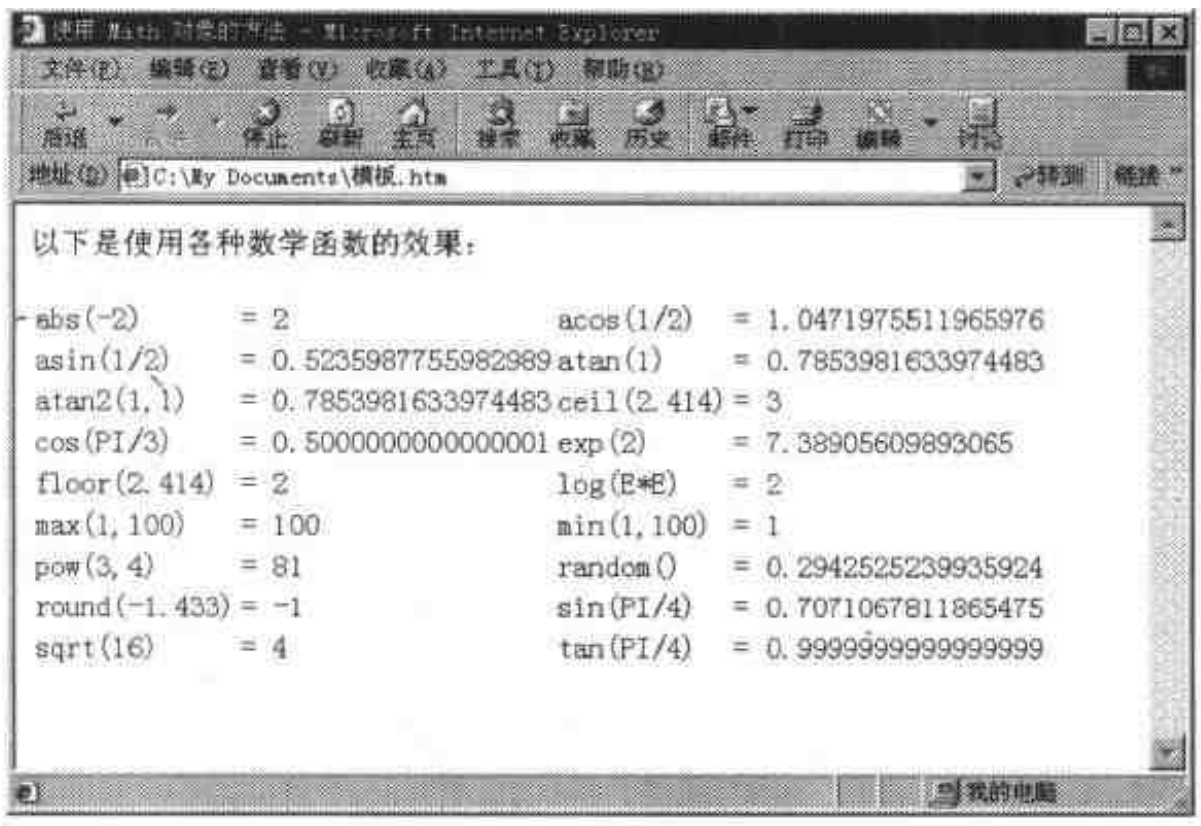


图 2.17　Math 对象的方法示例

```
<HTML>
<HEAD>
  <TITLE>使用 Math 对象的方法</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
```

```
<!--
document.write("以下是使用各种数学函数的效果：<P>");
document.write("<TABLE>")
document.write("<TR><TD>abs(-2)<TD> = "+Math.abs(-2));
document.write("<TD>acos(1/2)<TD> = "+Math.acos(1/2));
document.write("<TR><TD>asin(1/2)<TD> = "+Math.asin(1/2));
document.write("<TD>atan(1)<TD> = "+Math.atan(1));
document.write("<TR><TD>atan2(1,1)<TD> = "+Math.atan2(1,1));
document.write("<TD>ceil(2.414)<TD> = "+Math.ceil(2.414));
document.write("<TR><TD>cos(PI/3)<TD> = "+Math.cos(Math.PI/3));
document.write("<TD>exp(2)<TD> = "+Math.exp(2));
document.write("<TR><TD>floor(2.414)<TD> = "+Math.floor(2.414));
document.write("<TD>log(E*E)<TD> = "+Math.log(Math.E*Math.E));
document.write("<TR><TD>max(1,100)<TD> = "+Math.max(1,100));
document.write("<TD>min(1,100)<TD> = "+Math.min(1,100));
document.write("<TR><TD>pow(3,4)<TD> = "+Math.pow(3,4));
document.write("<TD>random()<TD> = "+Math.random());
document.write("<TR><TD>round( 1.433)<TD> = "+Math.round(-1.433));
document.write("<TD>sin(PI/4)<TD> = "+Math.sin(Math.PI/4));
document.write("<TR><TD>sqrt(16)<TD> = "+Math.sqrt(16));
document.write("<TD>tan(PI/4)<TD> = "+Math.tan(Math.PI/4));
document.write("</TABLE>")
-->
</SCRIPT>
</BODY>
</HTML>
```

2.9 Number 对象

Number 对象可以将数字作为对象访问，它通过为构造函数指定一个参数值来创建。与 Math 对象类似，Number 对象的多数属性也是固定的值，可以不用创建对象直接使用。

Number 对象的属性和方法如表 2.7 所示。

表 2.7 Number 对象的属性和方法

类　型	项　目	说　明
属性	MAX_VALUE	指定一个数的最大可能值，大约为 1.9E308
	MIN_VALUE	指定一个数的最小可能值，大约为 5E-324
	NaN	指定非数

续表

类　型	项　目	说　明
	NEGTIVE_INFINITY	指定数字为负无穷大
	POSITIVE_INFINITY	指定数字为正无穷大
	prototype	用于为对象添加属性和方法
方法	toString()	返回用字符串表示的 Number 对象
	valueOf()	返回对象的原始值

以下示例显示了如何使用 Number 对象，效果如图 2.18 所示。

图 2.18　使用 Number 对象

```
<HTML>
<HEAD>
  <TITLE>使用 Number 对象</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
<!--
//以下是 Number 对象中的属性值:
document.write("MAX_VALUE="+Number.MAX_VALUE+"<P>");
document.write("MIN_VALUE="+Number.MIN_VALUE+"<P>");
document.write("NaN="+Number.NaN+"<P>");
document.write("NEGATIVE_INFINITY="+Number.NEGATIVE_INFINITY+"<P>");
document.write("POSITIVE_INFINITY="+Number.POSITIVE_INFINITY+"<P>");

//以下是 prototype 属性的用法:
function double(num)
```

```
{
return num*2;
}
Number.prototype.mutiply2=double;
myNum=new Number();
document.write("30*2="+myNum.mutiply2(30)+"<P>");

//以下是 toString() 方法和 valueOf() 方法的用法:
myNum=new Number(100);
document.write("myNum.toString()="+myNum.toString()+"<P>");
document.write("myNum.valueOf()="+myNum.valueOf());
-->
</SCRIPT>
</BODY>
</HTML>
```

2.10 Object 对象

Object 对象是派生所有其他对象的对象，其属性和方法可以派生给所有其他对象。创建 Object 对象时，可以在构造函数中提供数字、字符串或布尔值，但一般不这样做，而是使用具体类型对象的构造函数 Number()、String()和 Boolean()等。

Object 对象的属性和方法如表 2.8 所示。

表 2.8 Object 对象的属性和方法

类　型	项　目	说　明
属性	constructor	表示对象的构造函数的名称
	prototype	用来为对象添加新的属性和方法
方法	toString()	将对象转换为用字符串表示
	valueOf()	获得指定对象的原始值

以下示例显示了如何使用 Object 对象，效果如图 2.19 所示。

```
<HTML>
<HEAD>
  <TITLE>使用 Object 对象</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
<!--
```

```
var myObj1=new Object(true);
var myObj2=new Object(20);

document.write("myObj1=new Object(true)<P>")
document.write("myObj1.constructor="+myObj1.constructor+"<br>");
document.write("myObj1.toString()="+myObj1.toString()+"<br>");
document.write("myObj1.valueOf()="+myObj1.valueOf()+"<P>")

document.write("myObj2=new Object(20)<P>")
document.write("myObj2.constructor="+myObj2.constructor+"<br>");
document.write("myObj2.toString()="+myObj2.toString()+"<br>");
document.write("myObj2.valueOf()="+myObj2.valueOf())
-->
</SCRIPT>
</BODY>
</HTML>
```

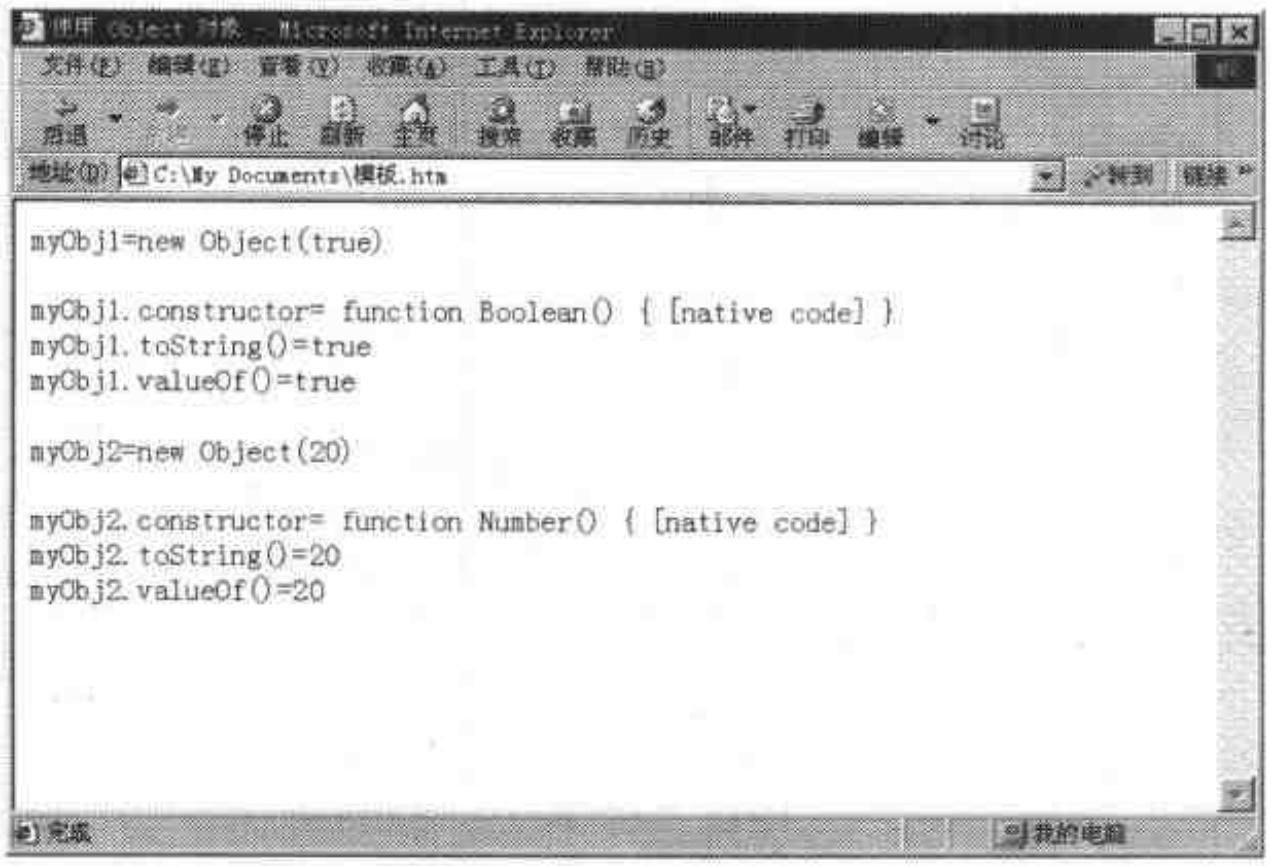


图 2.19 使用 Object 对象

2.11 RegExp 对象

RegExp 对象表示一个用来进行模式匹配的正规表达式。正规表达式是描述字符模式的字符串表达式，提供了在文本字符串中寻找模式和对文本进行查找和替换操作的强大功能。创建正规表达式时，需要使用以下格式：

```
var variable = new RegExp(pattern, flags)
```

参数 pattern 表示一个合法的正规表达式，参数 flags 可以取 g(global)和 i(ignor case)中的一个或两个。

由于正规表达式涉及比较复杂的专业概念，因此本书不打算详细说明如何使用 RegExp 对象，以下仅列出该对象的属性和方法供需要的读者参考，如表 2.9 所示。

表 2.9 RegExp 对象的属性和方法

类 型	项 目	说 明
属性	RegExp.$*	表示 multiline 属性
	RegExp.$&	表示 lastmatch 属性
	RegExp.$_	表示 input 属性
	RegExp.$`	表示 leftContext 属性
	RegExp.$'	表示 rightContext 属性
	RegExp.$+	表示 lastParen 属性
	RegExp.$1,$2,...$9	表示匹配的子字符串
	global	指定是否检查所有可能的匹配
	ignoreCase	指定在查找字符串时是否忽略大小写
	input	被进行匹配的字符串
	lastIndex	指定进行下一个匹配的开始位置
	lastMatch	最后一次匹配的字符串
	lastParen	最后被括号括住的子字符串匹配
	leftContext	表示最近一次匹配之前的子字符串
	multiline	决定模式匹配是否能在多行中进行
	rightContext	表示最近一次匹配之后的子字符串
	source	表示被用来进行模式匹配的文本
方法	compile(pattern, flags)	编译一个正规表达式对象
	exec(string)	在参数 string 指定的字符串中寻找匹配，匹配的结果通过一个数组返回
	test()	测试一个字符串是否能够被匹配，返回布尔值 true 或 false

2.12 String 对象

2.12.1 创建 String 对象

String 对象就是字符串对象。创建字符串对象时，既可以使用对象的构造函数，如下

所示：

```
myStr=new String("This is a string")
```

也可以直接将字符串值赋予变量，如下所示：

```
myStr="This is a string"
```

这两种方法是完全等价的。

字符串对象包含以下两个属性：

- **length** 表示字符串的长度；
- **prototype** 用来在字符串对象中添加属性和方法。

以下示例显示了字符串和其 length 属性的基本用法，效果如图 2.20 所示。

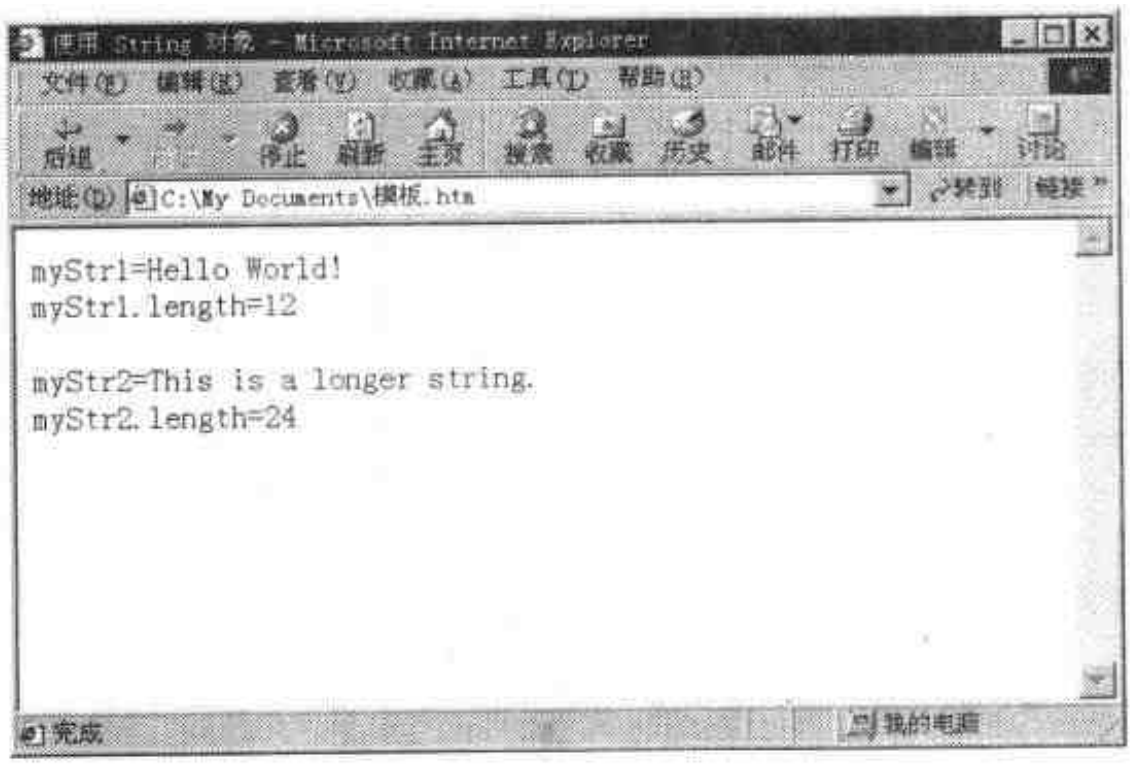


图 2.20 使用字符串和其属性

```
<HTML>
<HEAD>
  <TITLE>使用 String 对象</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
<!--
myStr1="Hello World!";
myStr2=new String("This is a longer string.")
document.write("myStr1="+myStr1+"<BR>")
document.write("myStr1.length="+myStr1.length+"<P>");
document.write("myStr2="+myStr2+"<BR>")
```

```
document.write("myStr2.length="+myStr2.length);
-->
</SCRIPT>
</BODY>
</HTML>
```

2.12.2 字符串格式设置

String 对象中包含一些用于设置字符串格式的方法，相当于为字符串添加格式设置标记符。例如，使用 bold()方法就相当于为字符串添加了<B></B>标记符。

表 2.10 列出了这些用于字符串格式设置的方法（包括字符大小写转换）。

表 2.10 String 对象的字符串格式设置方法

方 法	说 明
anchor(name)	将调用此方法的字符串转换成一个 <A> 标记符的实例，用参数 name 设置 NAME 属性。这相当于将字符串用<A NAME=name></A>进行标记
big()	将调用此方法的字符串用大体字显示，相当于为字符串添加 <BIG></BIG> 标记符
bold()	将调用此方法的字符串用黑体字显示，相当于为字符串添加 <B></B> 标记符
fixed()	将调用此方法的字符串用等宽字体显示，相当于为字符串添加 <TT></TT> 标记符
fontcolor(hexnum) fontcolor(colorname)	将调用此方法的字符串用参数所设置的颜色显示，相当于为字符串添加 <FONT color=""></FONT> 标记符。参数既可以是表示颜色的十六进制值，也可以是浏览器能够识别的颜色名称
fontsize(num) fontsize(string)	将调用此方法的字符串用参数所设置的字体大小显示，相当于为字符串添加 <FONT size=""></FONT> 标记符。参数既可以是 1 到 7 之间的一个数字，也可以是以字符串形式传入的数字。使用字符串数字时，字体大小取 BASEFONT 大小的相对倍数
italics()	将调用此方法的字符串用斜体字显示，相当于为字符串添加 <I></I> 标记符
link(URL)	将调用此方法的字符串转换成一个 <A> 标记符的实例，用参数 URL 设置 HREF 属性。这相当于将字符串用 <A HREF=URL></A> 进行标记
small()	将调用此方法的字符串用小体字显示，相当于为字符串添加 <SMALL> </SMALL> 标记符
strike()	将调用此方法的字符串用删除线格式显示，相当于为字符串添加 <STRIKE> </STRIKE> 标记符
sub()	将调用此方法的字符串用下标格式显示，相当于为字符串添加 标记符
sup()	将调用此方法的字符串用上标格式显示，相当于为字符串添加 标记符
toLowerCase()	将调用此方法的字符串中的字符转换为小写
toUpperCase()	将调用此方法的字符串中的字符转换为大写

以下示例显示了这些字符格式设置方法的作用（请注意注释语句），效果如图 2.21 所示。

```
<HTML>
<HEAD>
  <TITLE>使用 String 对象</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE."text/javascript">
<!--
myStr="Hello World!";
testStr=new String("请单击此处!");
document.write(myStr.anchor("Hi")+"<P>")  //设置一个锚点Hi
document.write("<table>")
document.write("<tr><td>myStr.big()<td> = "+myStr.big())
document.write("<tr><td>myStr.bold()<td> - "+myStr.bold())
document.write("<tr><td>myStr.fixed()<td> - "+myStr.fixed())
document.write("<tr><td>myStr.fontcolor('#00ff00')<td>                    =
"+myStr.fontcolor("#00ff00"))
document.write("<tr><td>myStr.fontcolor('red')<td>                        =
"+myStr.fontcolor("red"))
document.write("<tr><td>myStr.fontsize(5)<td> = "+myStr.fontsize(5))
document.write("<tr><td>myStr.fontsize('-2')<td>                          -
"+myStr.fontsize("-2"))
document.write("<tr><td>myStr.italics()<td> - "+myStr.italics())
document.write("<tr><td>myStr.small()<td> = "+myStr.small())
document.write("<tr><td>myStr.strike()<td> = "+myStr.strike())
document.write("<tr><td>myStr.sub()<td> = "+myStr.sub())
document.write("<tr><td>myStr.sup()<td> = "+myStr.sup())
document.write("<tr><td>myStr.toLowerCase()<td>                           =
"+myStr.toLowerCase())
document.write("<tr><td>myStr.toUpperCase()<td>                           =
"+myStr.toUpperCase())
document.write("</table><p>");
document.write(testStr.link("#Hi"));  //将字符串设置为超链接，目标为前面定
义的锚点!
-->
</SCRIPT>
</BODY>
</HTML>
```

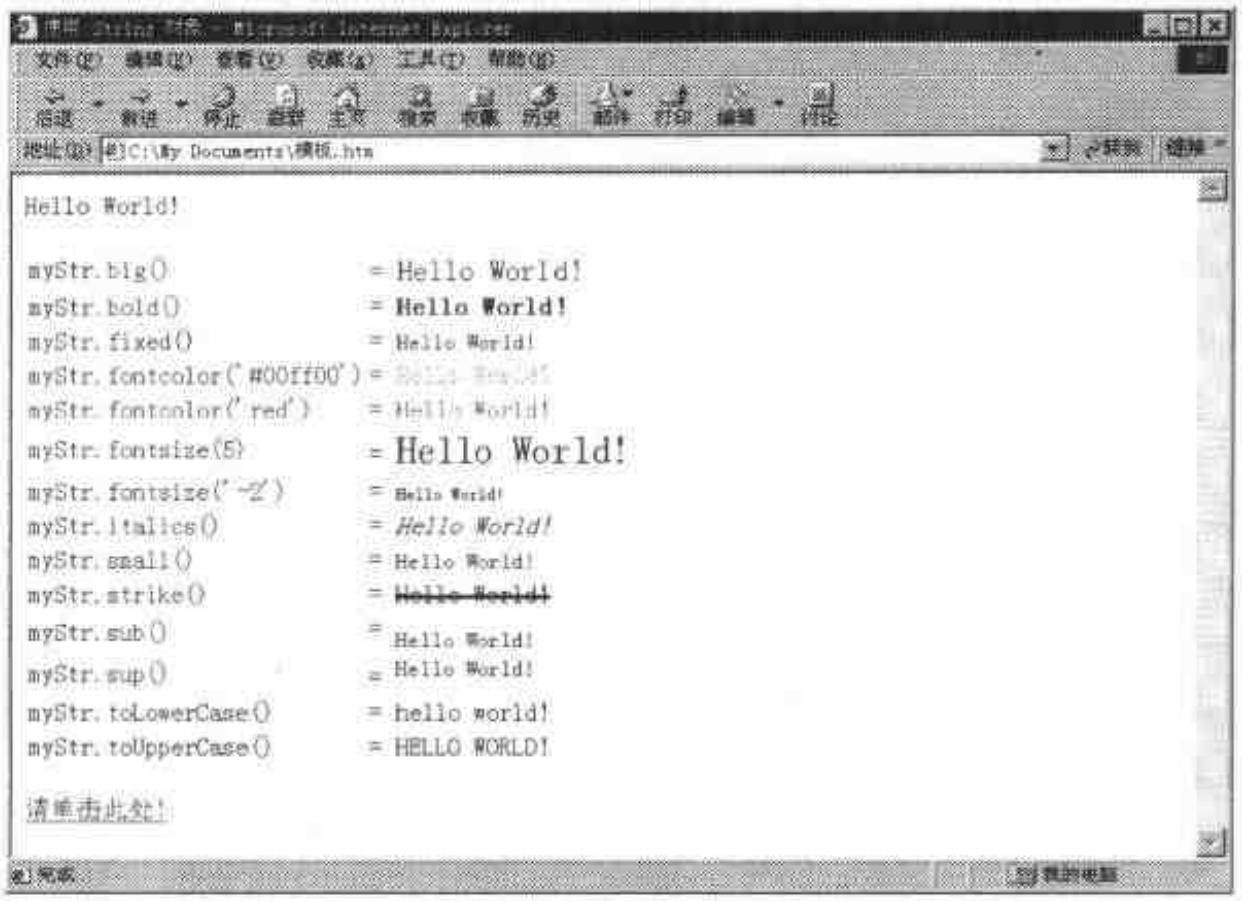


图 2.21　设置字符格式

2.12.3　通用字符串操作

除了设置字符格式的方法外，String 对象还包括一些执行通用字符串操作的方法。例如，charAt()方法用于获得指定位置上的字符。

表 2.11 列出了这些用于进行通用字符串操作的方法（包括另两个通用方法 toString()和 valueOf()）。

表 2.11　String 对象的通用字符串操作方法

方　法	说　明
charAt(num)	返回参数 num 指定索引位置处的字符。从左到右索引，开始处为 0
charCodeAt(num)	返回参数 num 指定索引位置处字符的 ISO-Latin-1 值。从左到右索引，开始处为 0
cancat(string)	将参数 string 传过来的字符串加到当前字符串的末尾，并返回新的字符串
fromCharCode(num1,num2,...numN)	返回对应于参数传入的 ISO-Latin-1 值（num1,num2,...numN）位置处的字符
indexOf(string,num) indexOf(string)	返回参数 string 在字符串中出现的初始位置。如果指定了参数 num，则表示从索引 num 处开始查找
lastIndexOf(string,num) lastIndexOf(string)	返回参数 string 在字符串中出现的初始位置。如果指定了参数 num，则表示从索引 num 处开始查找。与 indexOf 方法不同的是，此方法从后向前查找字符串
match(regexpression)	在调用此方法的字符串中查找通过参数 regexpression 传入的正规表达式所指定的字符串，返回包含在字符串中找到的所有匹配的一个数组

续表

方　法	说　明
replace(regexpression,replacestring)	在调用此方法的字符串中查找通过参数 regexpression 传入的正规表达式所指定的字符中，当在字符串中找到一个匹配时，它返回由参数 replacestring 替换匹配之后的字符串
search(regexpression)	在调用此方法的字符串中查找通过参数 regexpression 传入的正规表达式所指定的字符串，返回匹配开始在字符串中的索引，如果没有找到匹配，则返回 -1
slice(num1,num2) slice(num)	返回字符串中从索引 num1 到索引 num2 之间的字符串。当 num2 是负数时，则从字符串结束位置向前 num2 个字符的位置是返回字符串的结束位置。如果只指定一个参数，则返回从该索引到字符串结束之间的字符串
split(separator,num) split(separator) split(regexpression,num)	根据参数传入的正规表达式或分隔符来分隔调用此方法的字符串
substr(num1,num2) substr(num)	返回在字符串中索引 num1 和 num2 之间的字符串。如果只指定一个参数，则返回从该位置到字符串结尾处的字符串
substring(num1,num2) substring(num)	返回在字符串中索引 num1 和 num2 之间的字符串。当 num1 为负数时，则其被视为 0；如果参数 num2 的值大于 string.length 则其被视为 string.length；如果 num1=num2，则返回空字符串。如果只指定一个参数，则返回从该位置到字符串结尾处的字符串
toString()	如果直接调用此方法，则返回构造函数，格式如下：function String() { [native code] }。如果在字符串实例中调用此方法，则返回创建实例时的源字符串
valueOf()	返回对象的原始值，对于 String 对象，也就是返回字符串自身

以下示例显示了如何使用常用的字符串操作方法，效果如图 2.22 所示。

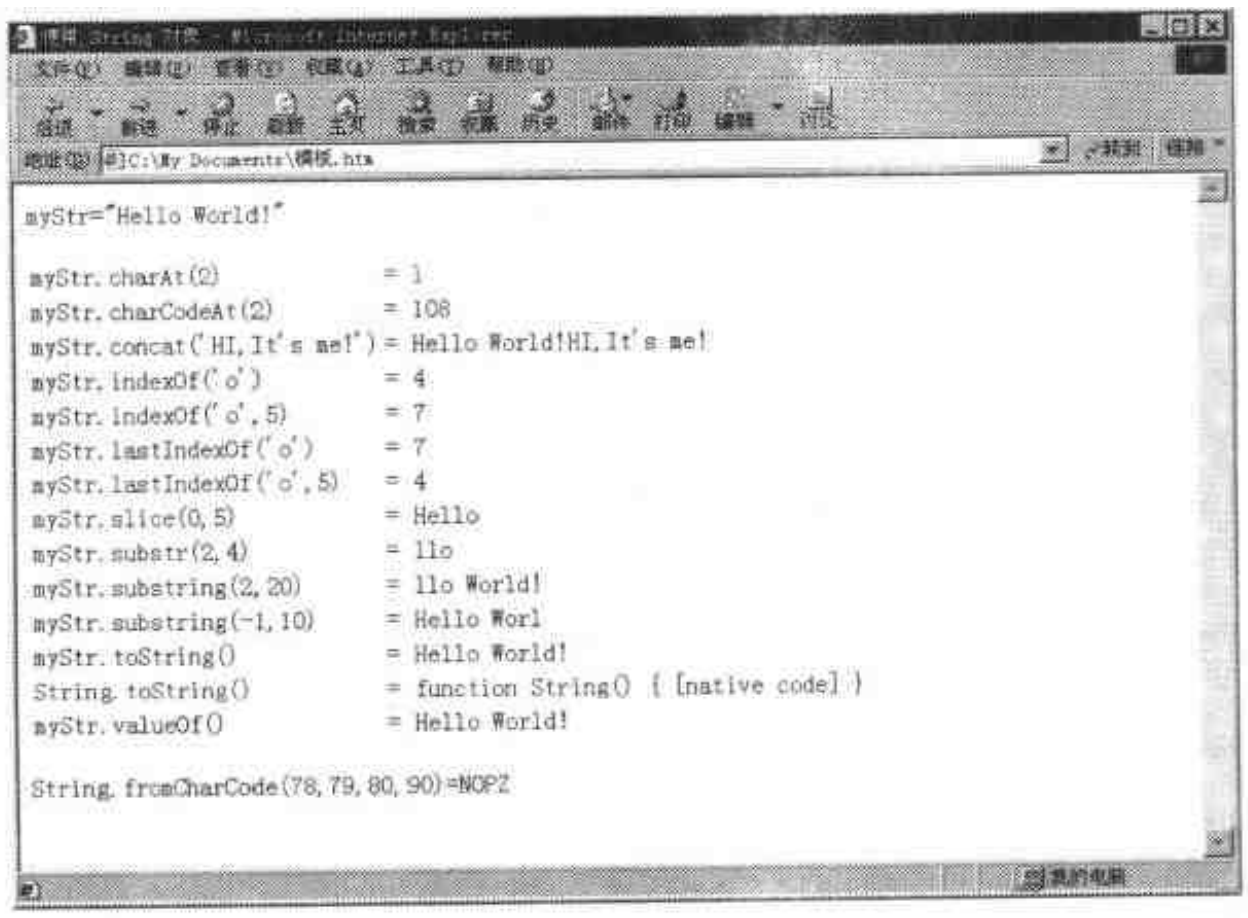


图 2.22　使用字符串操作方法

```
<HTML>
<HEAD>
```

```
  <TITLE>使用 String 对象</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUANGE="JavaScript" TYPE="text/javascript">
<!--
myStr="Hello World!";
document.write('myStr="Hello World!"<P>')
document.write("<table>")
document.write("<tr><td>myStr.charAt(2)<td> = "+myStr.charAt(2))
document.write("<tr><td>myStr.charCodeAt(2)<td> = "+myStr.charCodeAt(2))
document.write("<tr><td>myStr.concat('HI,It\'s me!')<td> = "+myStr.concat
('HI,It\'s me!'))
document.write("<tr><td>myStr.indexOf('o')<td> = "+myStr.indexOf("o"))
document.write("<tr><td>myStr.indexOf('o',5)<td> = "+myStr.indexOf
('o',5))
document.write("<tr><td>myStr.lastIndexOf('o')<td> = "+myStr.lastIndexOf
('o'))
document.write("<tr><td>myStr.lastIndexOf('o',5)<td> = "+myStr
.lastIndexOf('o',5))
document.write("<tr><td>myStr.slice(0,5)<td> = "+myStr.slice(0,5))
document.write("<tr><td>myStr.substr(2,4)<td> = "+myStr.substr(2,4))
document.write("<tr><td>myStr.substring(2,20)<td> = "+myStr.substring
(2,20))
document.write("<tr><td>myStr.substring(-1,10)<td> = "+myStr.substring
(-1,10))
document.write("<tr><td>myStr.toString()<td> = "+myStr.toString())
document.write("<tr><td>String.toString()<td> = "+String.toString())
document.write("<tr><td>myStr.valueOf()<td> = "+myStr.valueOf())
document.write("</table><p>");
var myStr2=String.fromCharCode(78,79,80,90);
document.write("String.fromCharCode(78,79,80,90)="+myStr2)
-->
</SCRIPT>
</BODY>
</HTML>
```

第3章 JavaScript 事件处理

在客户端脚本中，JavaScript 通过对事件进行响应来获得与用户的交互。例如，当用户单击一个按钮或者在某段文字上移动鼠标时，就触发了一个单击事件或鼠标移动事件，通过对这些事件的响应，可以完成特定的功能（例如，单击按钮弹出对话框，鼠标移动到文本上后文本变色等）。

本章介绍有关客户端事件处理的基础知识和实现细节，主要内容包括：

- 什么是事件
- 处理 JavaScript 事件
- event 对象
- 错误处理

3.1 什么是事件

在第 1 章中说明如何添加脚本时，我们已经接触到了如何在脚本中响应单击按钮事件（onClick 事件）的例子。实际上，客户端脚本程序要处理的内容就是操作 Web 页上的各种对象，并通过响应各种事件（如鼠标单击、鼠标移动、按下键盘按键等），来提供一种表现力丰富的、具有交互功能的用户界面。

实际上，事件（event）在此的含义就是用户与 Web 页面交互时产生的操作。当用户进行单击按钮等操作时，即产生了一个事件，需要浏览器进行处理。浏览器响应事件并进行处理的过程称为事件处理，进行这种处理的代码称为事件响应函数。

通常浏览器会默认定义一些通用的事件处理过程，以便响应那些最基本的事件。例如，单击超链接的默认响应就是装入并显示目标页面，单击表单中的提交按钮的默认响应就是将表单提交到服务器，等等。

虽然如此，要实现动态的、具有交互功能的页面，却必须自定义事件处理函数。通过自定义事件处理函数，可以让 Web 页完成指定的功能，例如：

- 当用户单击某个按钮时，完成一系列指定的操作；
- 当鼠标指针移动到超链接上时，将超链接用另外一种醒目的方式显示；
- 在提交表单内容之前，先进行特定的验证处理，以便减轻网络和服务器的负担；

等等。

3.2 处理 JavaScript 事件

在 JavaScript 中，通过为不同的事件编写事件处理函数，可以对事件进行必要的处理，从而获得需要的效果或功能。

3.2.1 事件处理属性

JavaScript 就文档、表单、图像、超链接等对象定义了若干个标准事件，同时定义了对应于特定对象的 HTML 标记符的事件处理属性，以便指定脚本处理函数。

1. JavaScript 事件

表 3.1 列出了 Internet Explorer 和 Navigator 共有的 JavaScript 事件，对应于不同对象的具体事件处理将从第 4 章开始分别介绍。

表 3.1 JavaScript 事件

HTML 元素	JavaScript 事件	说　明
所有元素	mouseMove	鼠标移动
A	click	鼠标单击超链接
	dblClick	鼠标双击超链接
	mouseDown	按下鼠标左键
	mouseUp	释放鼠标左键
	mouseOver	鼠标移入超链接
	mouseOut	鼠标移出超链接
	keyDown	按下键盘上的某键
	keyUp	释放键盘上的某键
Area	mouseOver	鼠标移入客户端图像映射区域
	mouseOut	鼠标移出客户端图像映射区域
	dblClick	鼠标双击客户端图像映射区域

续表

HTML 元素	JavaScript 事件	说　明
Body	click	在文档中单击鼠标左键
	dblClick	在文档中双击鼠标左键
	keyDown	在文档中按下键盘上的某键
	keyUp	在文档中释放键盘上的某键
	keyPress	在文档中按下并释放键盘上的某键
	mouseDown	在文档中按下鼠标左键
	mouseUp	在文档中释放鼠标左键
Body Frameset Frame	blur	窗口失去当前输入焦点
	error	装入窗口时发生错误
	focus	窗口获得当前输入焦点
	load	加载窗口
	unload	卸载窗口
	move	窗口移动
	resize	窗口缩放
	dragDrop	在窗口中拖放对象
Form	submit	提交表单
	reset	重置表单
Img	abort	放弃图像装入操作
	error	图像装入期间发生错误
	load	图像装入
	keyDown	按下键盘上的某键
	keyUp	释放键盘上的某键
	keyPress	按下并释放键盘上的某键
Input (type="text")	blur	当前文本框失去输入焦点
	focus	当前文本框获得输入焦点
	change	文本框中的内容被修改并失去输入焦点
	select	文本框中的文本被选中
Input (type="password")	blur	当前口令框失去输入焦点
	focus	当前口令框获得输入焦点
Textarea	blur	当前多行文本框失去输入焦点
	focus	当前多行文本框获得输入焦点
	change	多行文本框中的内容被修改并失去输入焦点
	select	多行文本框中的文本被选中
Input (type="checkbox")	click	单击复选框
	blur	当前复选框失去输入焦点
	focus	当前复选框获得输入焦点
Input (type="radio")	click	单击单选框
	blur	当前单选框失去输入焦点
	focus	当前单选框获得输入焦点

续表

HTML 元素	JavaScript 事件	说　明
Input (type="file")	blur	当前文件选择框失去输入焦点
	focus	当前文件选择框获得输入焦点
	change	文件选择框的文本内容被修改并失去输入焦点
Input (type="button")	click	单击按钮
	blur	当前按钮失去输入焦点
	focus	当前按钮获得输入焦点
	mouseDown	在按钮上单击鼠标左键
	mouseUp	在按钮上释放鼠标左键
Input (type="submit")	click	单击提交按钮
	blur	当前提交按钮失去输入焦点
	focus	当前提交按钮获得输入焦点
Input (type="reset")	click	单击重置按钮
	blur	当前重置按钮失去输入焦点
	focus	当前重置按钮获得输入焦点
Select	blur	当前选项菜单失去输入焦点
	focus	当前选项菜单获得输入焦点
	change	选项菜单中的选项被修改并失去输入焦点

2. 事件处理属性

在实际进行事件处理时，需要将具体的事件与事件处理函数相关联，关联的方法就是为 HTML 元素指定事件处理属性（或者说是事件响应属性）。

例如，在第 1 章中的示例里，通过为 INPUT 标记符指定 onClick 属性，则确定了事件与事件处理函数的关联，如下所示：

```
<INPUT  TYPE="Button"  onClick="showdate();"  VALUE="显示时间">
```

onClick 表示要处理的事件是 click 事件，等号后面的内容，或者说是 onClick 这个事件处理属性的值，是事件处理函数的名称。具体的事件处理函数，则用前面章节介绍的方法包含在 SCRIPT 标记符中。

以下示例进一步说明了如何指定事件与事件处理函数的关联，效果如图 3.1 所示。

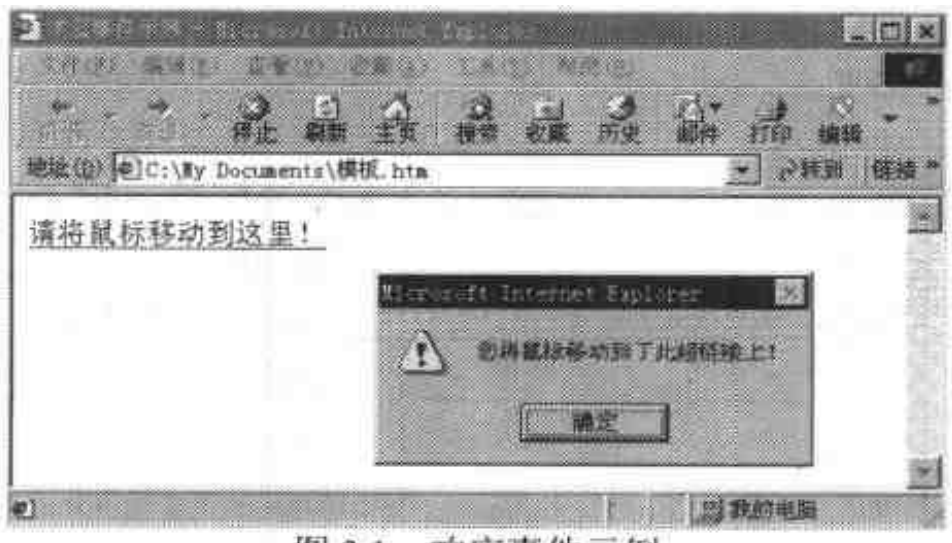


图 3.1　响应事件示例

```
<HTML>
<HEAD>
<SCRIPT Language ="JavaScript" TYPE='text/javascript'>
function eventHandler()
{  alert("您将鼠标移动到了此超链接上！")  }
</SCRIPT>
<TITLE>响应事件示例</TITLE>
</HEAD>
<BODY>
  <A href="nonexist.com.cn' onMouseOver='eventHandler()'>请将鼠标移动到
这里！</A>
</BODY>
</HTML>
```

在这个示例中，当用户将鼠标指针移动到超链接上时，导致了 mouseOver 事件的发生，根据 onMouseOver 属性的指定，调用了 eventHandler()函数，该函数的作用是显示一个警告对话框。

可以看出，对应于某事件的事件处理属性通常就是在该事件名称之前加 on。通常在 HTML 中属性是与大小写无关的，即 onMouseOver、onMouseover 与 ONMOUSEOVER 都一样可以使用。

对应于定义的 JavaScript 事件，表 3.2 列出了 HTML 元素的事件处理属性。

表 3.2 HTML 事件

事　件	功　能	适 用 于
onAbort	当用户中断图像装载时发生	与 IMG 元素一起使用
onBlur	当一个元素失去来自鼠标或键盘的焦点时发生	LABEL、INPUT、SELECT、TEXTAREA 和 BUTTON 元素
onChange	当一个元素丢失焦点，其值被改变时	与 INPUT、SELECT 和 TEXTAREA 一起使用
onClick	当一个元素被鼠标单击时发生	与绝大多数元素一起使用
onDbiclick	当一个元素被鼠标双击时发生	与绝大多数元素一起使用
onDragDrop	将对象拖放到窗口或框架时发生	与 BODY、FRAME 和 FRAMESET 元素一起使用
onError	当装入窗口、框架、图像期间出错时发生	与 BODY、FRAME、FRAMESET 和 IMG 元素一起使用
onFocus	当一个元素接收到来自鼠标或键盘的焦点时发生	LABEL、INPUT、SELECT、TEXTAREA 和 BUTTON 元素
onKeyDown	当在一个元素上方一个键被按住不放时发生	与绝大多数元素一起使用
onKeyPress	当在一个元素上方一个键被按下后和松手时发生	与绝大多数元素一起使用

续表

事　件	功　能	适　用　于
onKeyUp	当在一个元素上方一个键被松手时发生	与绝大多数元素一起使用
onLoad	当 Web 浏览器加载窗口或框架集时发生	与 BODY、FRAME 和 FRAMESET 元素一起使用
onMouseDown	当鼠标在一个元素上方被按住时发生	与绝大多数元素一起使用
onMouseMove	当鼠标在一个元素的上方移动时发生	与绝大多数元素一起使用
onMouseOut	当鼠标离开一个元素时发生	与绝大多数元素一起使用
onMouseOver	当鼠标从网页上的某处在一个元素上方经过时发生	与绝大多数元素　起使用
onMouseUp	当鼠标在　个元素上方被释放时发生	与绝大多数元素一起使用
onMove	当移动窗口或框架时发生	与 BODY、FRAME 和 FRAMESET 元素一起使用
onReset	当一个表单被重置时发生	与 FORM 元素一起使用
onResize	当调整窗口、框架的尺寸时发生	与 BODY、FRAME 和 FRAMESET 元素一起使用
onSelect	当文本被选择时发生	与 INPUT 和 TEXTAREA 一起使用
onSubmit	当一个表单被提交时发生	与 FORM 元素一起使用
onUnload	当 Web 浏览器从窗口或框架卸载一个文档时发生	与 BODY、FRAME 和 FRAMESET 元素一起使用

3.2.2　事件处理函数

在第 1 章中我们已经看到，如果事件处理函数的逻辑比较简单，也可以直接将 JavaScript 语句作为事件处理属性的值。当然，为了使事件处理代码更加模块化和易于复用，最好还是采用事件处理函数的方式。

1. 使用语句进行事件处理

当事件处理过程仅包含很简单的内容时，可以直接用 JavaScript 语句作为事件处理属性的值，而无须定义函数。

通常，事件处理属性的值都用双引号括起来，如果其中某些语句需要使用引号，则应使用单引号；如果包含引号的语句中仍然需要引号，则只能用反斜杠的方式进行引用 \'。另外，如果需要包含多条语句，则需要使用分号分隔不同的语句。这些规定都与 JavaScript 的语言规定相同。

例如，以下示例显示了使用语句进行事件处理的效果，效果如图 3.2 所示。

```
<HTML>
<HEAD>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
```

```
var count=0;  //定义一个全局变量，用于统计用户在超链接上移动鼠标的次数
</SCRIPT>
<TITLE>响应事件示例</TITLE>
</HEAD>
<BODY>
  <A href="nonexist.com.cn" onMouseOver="count++; alert('您已经将鼠标移
动到此超链接上'+count+'次了！')">请将鼠标移动到这里！</A>
</BODY>
</HTML>
```

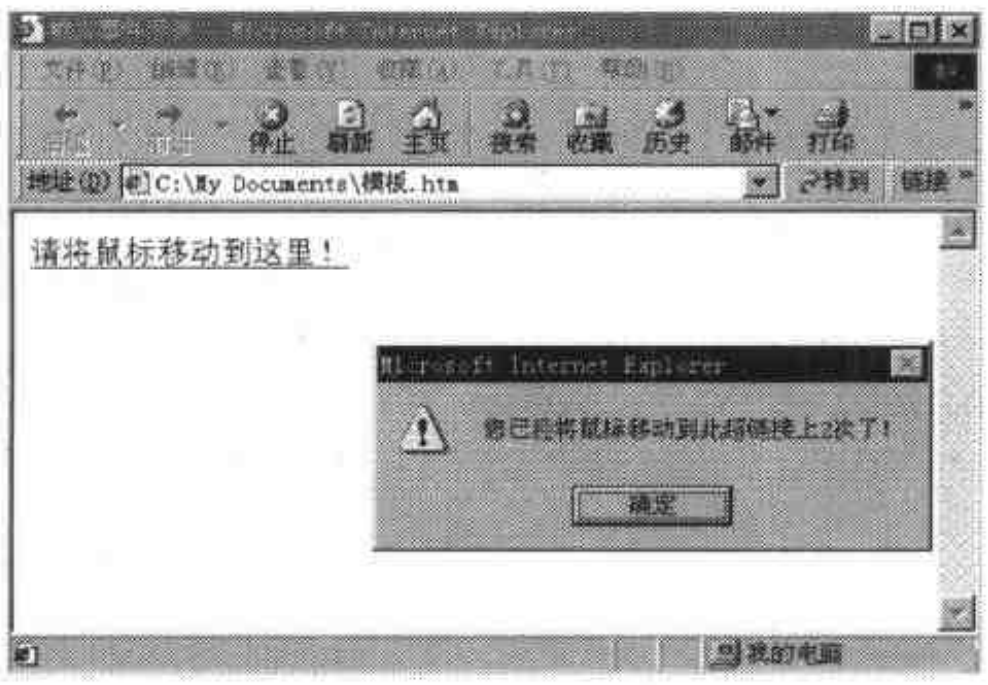


图 3.2 使用语句进行事件处理

2. 使用函数进行事件处理

虽然直接使用语句指定事件处理过程比较简洁，但显然不如使用事件处理函数那么逻辑清楚，尤其是语句超过一句时。使用函数进行事件处理，可以使事件处理代码更容易调试、更易于模块化、更易于在其他网页中反复使用。

前面的多数示例都是使用事件处理函数进行处理，以下再举一个例子，代码如下：

```
<HTML>
<HEAD>
<TITLE>强调链接</TITLE>
<SCRIPT LANGUAGE="JavaScript"TYPE="text/javascript">
<!--
function link1Over()
    {   link1.style.color="red"
       //这里用到了对象的 style 属性，有关的详细信息请参见本书第 8 章
        link1.style.fontSize=36    }
```

```
function link1Out()
    {   link1.style.color="black"
        link1.style.fontSize=16    }

function link2Over()
    {    link2.style.color="red"
         link2.style.fontSize=36    }
function link2Out()
    {   link2.style.color="black"
        link2.style.fontSize=16    }

function link3Over()
    {   link3.style.color="red"
        link3.style.fontSize=36    }
function link3Out()
    {   link3.style.color="black"
        link3.style.fontSize=16    }
-->
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<FORM NAME = form1>
<H3>移动鼠标到任意一个超链接上则可以放大显示该超链接。</H3>
<A HREF="http://www.server.com" name= link1 onMouseOver = "link1Over()"
    onMouseOut = "link1Out()">超链接 1</A>
<BR><BR>
<A HREF="http://www.server.com" name= link2 onMouseOver = "link2Over()"
    onMouseOut = "link2Out()">超链接 2</A>
<BR><BR>
<A HREF="http://www.server.com" name= link3 onMouseOver = "link3Over()"
    onMouseOut = "link3Out()">超链接 3</A>
</FORM>
</DIV>
</BODY>
</HTML>
```

在浏览器上执行以上代码后，当用户将鼠标指针移动到任意一个超链接上时，超链接的文字将放大显示，同时改变颜色，如图 3.3 所示。如果用户将鼠标指针移出超链接，则

超链接文字将变小，颜色变为黑色。

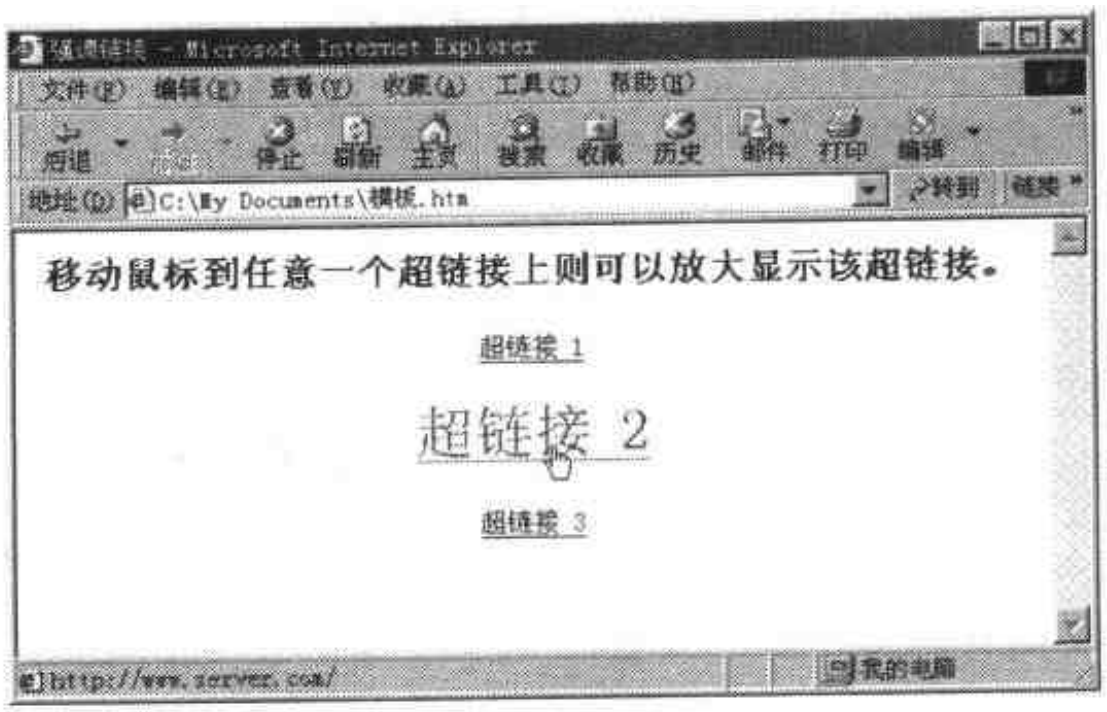


图 3.3 使用事件处理函数

3.2.3 通过对象指定事件处理函数

除了通过在 HTML 标记符中指定事件处理属性来确定事件响应函数以外，还可以用对象的方式指定事件响应函数。

例如，对应于单击按钮显示当前日期和时间的示例，将其改为用对象的方式进行响应，代码如下所示。该段代码的显示效果如图 3.4 所示，与使用 JavaScript 语句或在事件处理属性中指定值时一模一样。

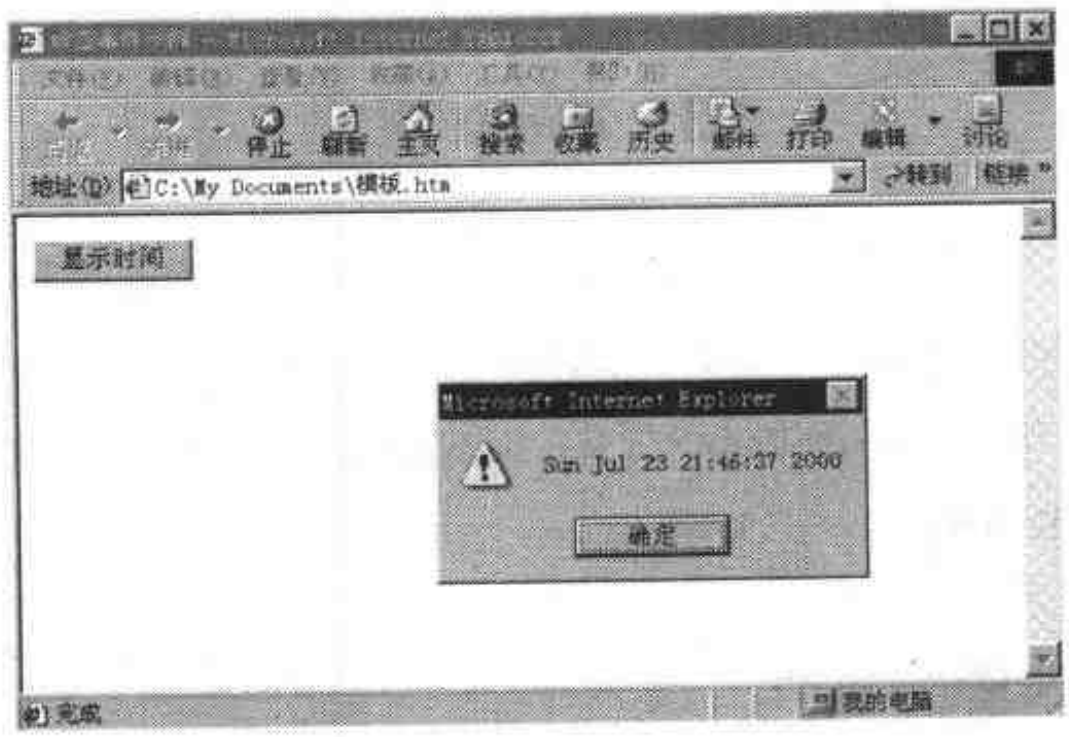


图 3.4 通过对象指定时间处理函数

```
<HTML>
<HEAD>
```

```
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
<!--
function showdate()
{ alert(Date()) }
//-->
</SCRIPT>
<TITLE>响应事件示例</TITLE>
</HEAD>
<BODY>
<FORM  NAME=form1>
   <INPUT  TYPE="Button"  NAME=button1  VALUE="显示时间">
</FORM>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
<!--
document.form1.button1.onclick=showdate;  //注意此处的函数名称后没有括号
//这种引用表单中对象的方式将在后面的章节中详细介绍
//-->
</SCRIPT>
</BODY>
</HTML>
```

实际上，这种方式与直接在 HTML 标记符中指定事件处理过程的本质都是——将特定对象上发生的事件与相应的事件处理过程联系起来，不同之处仅在于形式。

3.3 event 对象

event 对象是一种特殊的对象，它通过各种属性提供了特定的事件处理机制。对于 Internet Explorer 和 Navigator，event 对象的属性大不相同，并且传递事件的机制也各不相同（IE 采用事件浮升(event bubbling)的方式，而 Navigator 采用事件捕获(event capturing)的方式），本书仅以 IE 为例。

3.3.1 event 对象的属性

1. 属性列表

在 Internet Explorer 中，event 对象的属性如表 3.3 所示。

表 3.3 event 对象的属性

属 性	功 能
altKey	Alt 键按下时为真
ctrlKey	Ctrl 键按下时为真
shiftKey	Shift 键按下时为真
button	发生事件时所按的鼠标键（0 表示没按下任何键，1 表示按下鼠标左键，2 表示按下鼠标右键……）
cancelBubble	设置为真或假，表示取消或启用事件浮升（有关信息，请参见本章 3.3.2 节）
clientX	鼠标光标相对于事件所在窗口客户区域的水平坐标，不包括窗口修饰或滚动条
clientY	鼠标光标相对于事件所在窗口客户区域的垂直坐标，不包括窗口修饰或滚动条
keyCode	表示与所按键相关联的 Unicode 代码
offsetX	鼠标光标相对于事件所在对象（或者说容器）的水平坐标
offsetY	鼠标光标相对于事件所在对象（或者说容器）的垂直坐标
reason	表示数据源对象的数据传输状态
returnValue	表示从事件中返回的值，取值为 true 或 false
screenX	鼠标光标相对于用户屏幕的水平坐标
screenY	鼠标光标相对于用户屏幕的垂直坐标
fromElement	表示被移动的元素
srcElement	表示触发事件的对象
srcFilter	表示导致 onfilterchange 事件触发的过滤器对象（有关过滤器对象的详细信息，请参见本书第 8 章）
toElement	表示正在向其移动的那个元素
type	以字符串形式返回事件对象中的事件名称
x	鼠标光标相对于事件所在文档的水平坐标
y	鼠标光标相对于事件所在文档的垂直坐标

说明： event 对象只在事件过程中才有效，也就是说，只能在事件处理代码而不能在其他代码中使用该对象。尽管 event 对象可以使用所有属性，但某些属性只在特定事件处理代码中才有意义。例如，只有在处理 onmouseover 和 onmouseout 事件时 fromElement 和 toElement 属性才有意义。

2. 示例 1

以下示例显示了几种与坐标有关的属性的用法，代码如下：

```
<HTML>
<HEAD>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
<!--
function clientCoords()
{
var clientInfo="";
clientInfo="客户区域 X 坐标为"+window.event.clientX+"\r";
clientInfo+="客户区域 Y 坐标为"+window.event.clientY+"\r";
alert(clientInfo);
}
```

```
function screenCoords()
{
var screenInfo="";
screenInfo="屏幕区域 X 坐标为"+window.event.screenX+"\r";
screenInfo+="屏幕区域 Y 坐标为"+window.event.screenY+"\r";
alert(screenInfo);
}
//-->
</SCRIPT>
<TITLE>event 对象属性示例</TITLE>
</HEAD>
<BODY                    onmousemove="window.status='x='+window.event.x+'
y='+window.event.y">
<FORM>
<INPUT type=button onclick="clientCoords()" value="单击此处获得当前位置
的 client 坐标">
<BR><BR>
<INPUT type=button onclick="screenCoords()" value="单击此处获得当前位置
的 screen 坐标">
</FORM>
</BODY>
</HTML>
```

这段代码的效果为：当用户在浏览器窗口中移动鼠标指针时，在状态栏中显示出光标的 X 坐标和 Y 坐标；当单击“单击此处获得当前位置的 client 坐标”按钮时，将弹出一个提示框显示当前位置的客户区域坐标；当单击“单击此处获得当前位置的 screen 坐标”按钮时，将弹出一个提示框显示当前位置的屏幕区域坐标，如图 3.5 所示。

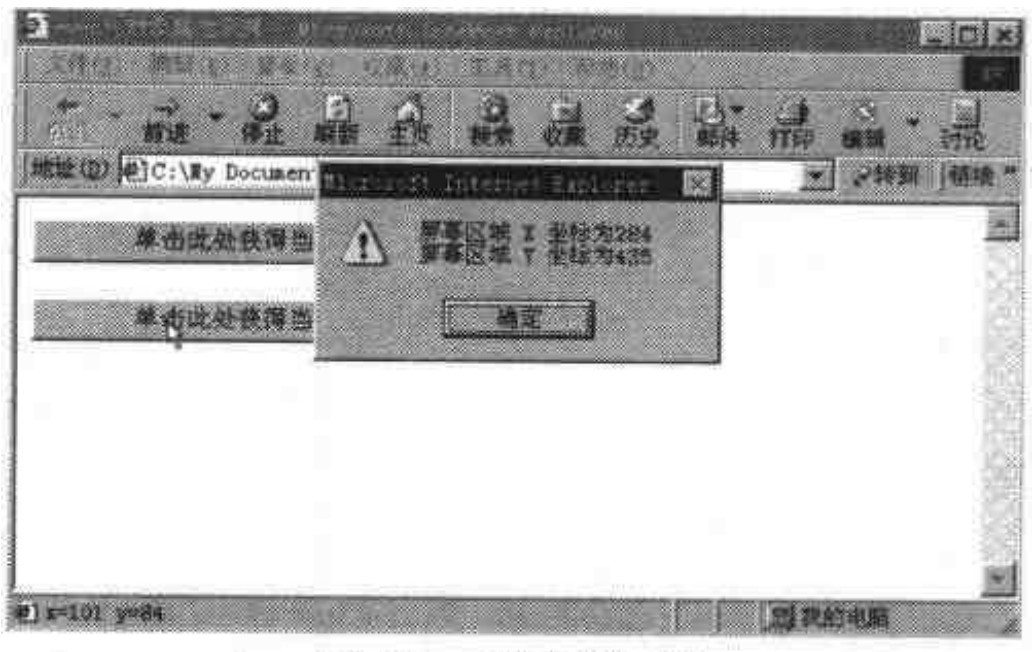


图 3.5　坐标属性示例

3. 示例 2

以下示例显示了 srcElement、shiftKey 以及 returnValue 等属性的用法，代码如下：

```
<HTML>
<HEAD>
<SCRIPT Language ="JavaScript" TYPE='text/javascript'>
<!--
function cancelLink()
{
if(window.event.srcElement.tagName=="A" && window.event.shiftKey)
  window.event.returnValue=false;
}
//-->
</SCRIPT>
<TITLE>event 对象属性示例</TITLE>
</HEAD>
<BODY onclick="cancelLink()">
<A href="noexist.com.cn">按住 shift 键再单击超链接，则无法跳转！</A>
</BODY>
</HTML>
```

在该段代码中，表达式 window.event.srcElement.tagName=="A"表示事件发生的对象是否为超链接标记符。由于 srcElement 属性的含义是触发事件的对象，因此该属性与一般 HTML 对象一样具有 tagName 等属性。在 if 语句中同时还判断 window.event.shiftKey 是否为真，即事件发生时是否按下了 shift 键。如果 if 条件判断为真，则将事件的返回值设置为 false，表示不进行事件处理。所以，当在文档中按下 shift 键的同时单击超链接时，并不进行通常的跳转操作，而是忽略该单击事件。

该段代码的显示效果如图 3.6 所示。

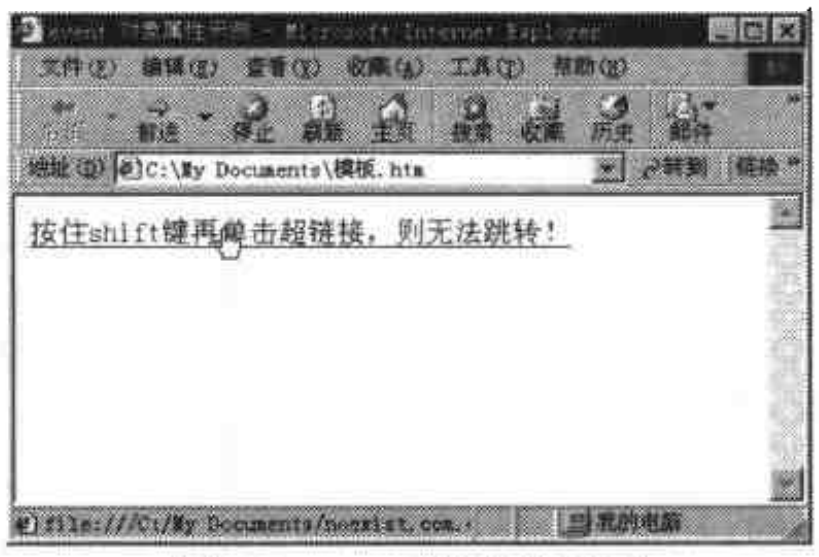


图 3.6 event 对象属性示例

4. 示例 3

以下示例更为实用，它演示了如何使用 event 对象的 keyCode 属性响应键盘事件，代码如下。

```
<HTML>
<HEAD>
<TITLE>响应键盘事件</TITLE>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
    function ShowHelp()
    {
     window.open("HelpWindow.htm")
    }  //有关 Window 对象的详细信息，请参见本书第 5 章
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<FORM>
<H2>单击“显示帮助”按钮则可以显示帮助窗口...</H2>
<BR>
<INPUT TYPE = BUTTON Value = "显示帮助" onClick = "ShowHelp()">
</FORM>
</DIV>
</BODY>
</HTML>
```

以下是帮助页面（HelpWindow.htm）的 HTML 代码：

```
<HTML>
<HEAD>
<TITLE>帮助窗口</TITLE>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
function keyPress(e)
{
    if(window.event.keyCode == 27){  //Esc 键的 Unicode 代码是 27
      window.close()
    }
}
</SCRIPT>
</HEAD>
<BODY onKeyPress = "keyPress()">
<DIV align=center>
```

```
<FORM NAME = form1>
<H2>这里是帮助窗口……</H2>
<BR>
<H3>帮助信息尚未完成，对不起……</H3>
<BR>
请按 ESC 键关闭此窗口……
</FORM>
</DIV>
</BODY>
</HTML>
```

这段代码的效果为：单击“显示帮助”按钮时将显示一个帮助窗口，此时如果按 Esc 键，则可以关闭该窗口，效果如图 3.7 所示。

图 3.7 响应键盘事件

3.3.2 事件浮升

在 3.3.1.3 的示例中，虽然事件发生在超链接标记符，但事件处理函数却定义在 BODY 标记符中，然而这并没有影响到我们期望的效果。这实际上反映了 Internet Explorer 处理事件时的一个机制，称为事件浮升（Event Bubbling）。

事件浮升的含义为：当事件发生时，它首先定向到发生事件的最底层对象，然后依次上浮，由各级对象定义的事件处理函数依次处理；如果某一级对象没有定义处理函数，则事件直接上浮。例如，在 3.3.1 的示例中，事件首先定向到 A 对象，由于该对象没有定义

事件处理函数，因此直接上浮到上一层对象 BODY，而 BODY 对象中定义了 onclick 事件处理函数，于是调用该函数处理发生在 A 对象上的 click 事件。

以下示例进一步显示了事件浮升机制的作用——当用户单击文档中的按钮时，click 事件依次由 INPUT 对象、FORM 对象和 BODY 对象中的事件处理函数进行处理。

```
<HTML>
<HEAD>
<TITLE>事件浮升示例</TITLE>
<SCRIPT LANGUAGE ="JavaScript"TYPE="text/javascript">
<!--
function buttonHandler()
{
 alert("Click 事件由 buttonHandler() 函数处理！")
}
function formHandler()
{
 alert("Click 事件由 formHandler() 函数处理！")
}
function documentHandler() {
 alert("Click 事件由 documentHandler() 函数处理！")
}
//-->
</SCRIPT>
</HEAD>
<BODY onclick="documentHandler()">
<FORM onclick="formHandler()">
  <INPUT TYPE=button onclick="buttonHandler()" VALUE="请单击此处！">
</FORM>
</BODY>
</HTML>
```

当用户单击“请单击此处”按钮时，首先显示“Click 事件由 buttonHandler() 函数处理！”提示框，如图 3.8 所示；单击“确定”按钮后，事件浮升到 FORM 对象，由 formHandler() 函数进行处理，此时显示如图 3.9 所示提示框；单击“确定”按钮后，事件进一步浮升，由 documentHandler() 处理，显示出如图 3.10 所示提示框（如果用户在按钮以外的区域单击鼠标左键，则会直接显示该提示框）。

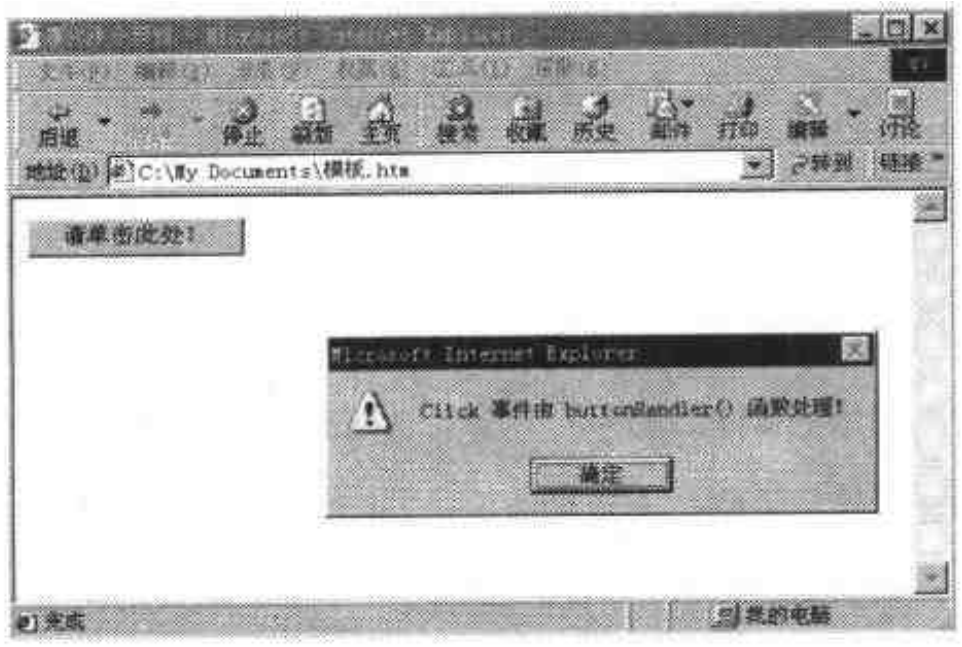


图 3.8　事件浮升示例

图 3.9　事件浮升到 FORM

图 3.10　事件进一步浮升到 BODY

在事件处理过程中，如果需要事件停止浮升，则应设置 event 对象的 cancelBubble 属性为 true，如下所示：

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

例如，在刚才的事件浮升示例中，如果将 buttonHandler()按如下方式定义，则事件仅由该处理函数处理，而不会继续浮升到上级对象。

```
function buttonHandler()
{
 alert('Click 事件由 buttonHandler() 函数处理！')
 window.event.cancelBubble=true;
}
```

说明：设置 cancelBubble 属性只影响当前事件的浮升，而不影响其他事件。

3.4 错 误 处 理

在任何一门编程语言中，错误处理都是很重要的一个方面，在 JavaScript 中也提供了在脚本执行期间处理错误的功能。用户通常可以使用 error 事件来处理与装入图形和文档相关联的错误，以及处理运行时的错误。另外，在 Internet Explorer 5 中，还可以使用几个特殊的错误处理语句进行错误处理。

3.4.1 error 事件

在 JavaScript 中，通过使用 onError 事件处理属性可以指定出错时的错误处理函数。对于一般的图像装载错误，可以象指定其他事件处理函数一样简单指定。如果 onError 事件绑定到 window 对象，则事件处理函数可以使用以下三个参数：

- sMsg　表示所发生错误的描述；
- sUrl　表示发生错误页面的 URL；
- sLine　表示发生错误的行号。

利用这些参数可以向用户提供有关的错误信息。此外，onError 事件处理函数的返回值确定是否向用户显示标准错误信息——返回 true 时不显示，返回 false 时显示。

1. 示例 1

本示例显示了当装载图像出错时的处理，代码如下：

```
<HTML>
<HEAD>
<TITLE>错误处理示例</TITLE>
<SCRIPT LANGUAGE = "JavaScript"TYPE="text/javascript">
<!--
function errorImage()
{
 alert("图像装载错误! ")
}
//-->
</SCRIPT>
</HEAD>
<BODY>
```

```
  <IMG src="nonexist.gif" onerror="errorImage()">
</BODY>
</HTML>
```

由于 IMG 标记符的 src 属性为一个不存在的图像，因此当装载图像时出错，于是调用 errorImage() 事件处理函数，显示一个提示框，效果如图 3.11 所示。

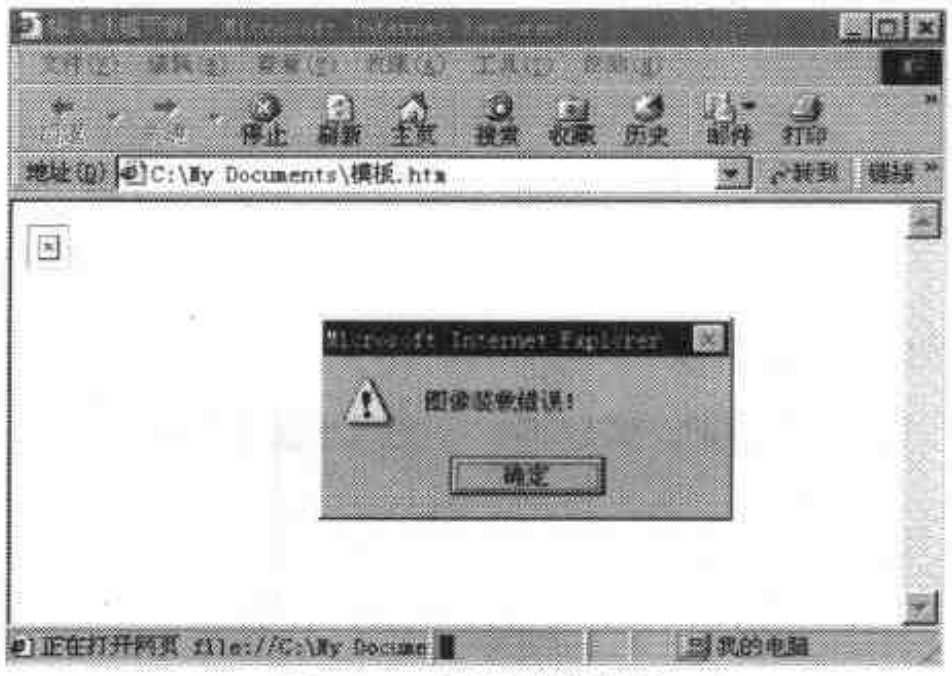


图 3.11 图像装载错误

3. 示例 2

本示例显示了如何使用错误处理函数的参数，代码如下：

```
<HTML>
<HEAD>
<TITLE>错误处理示例</TITLE>
<SCRIPT LANGUAGE ="JavaScript"TYPE="text/javascript">
<!--
window.onerror=errorHandler;  /*由于 IE 不能正确处理 BODY 标记符的 onerror
属性，因此采用此方法指定事件处理函数 */
function errorHandler(sMsg,sUrl,sLine)
{
document.write("出错信息: "+sMsg+"<P>")
document.write("出错 URL: "+sUrl+"<P>")
document.write("出错行号: "+sLine+"<P>")
return true;  //使用此语句可以确保浏览器不再显示标准的脚本出错信息
}
function launchError()
{
eval(errorCode.value);
}
```

```
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM>
  <INPUT name=errorCode value="nonexist()">
  <INPUT type=button value="创建错误" onclick=launchError()>
</FORM>
</BODY>
</HTML>
```

该段代码的显示如图 3.12 所示。

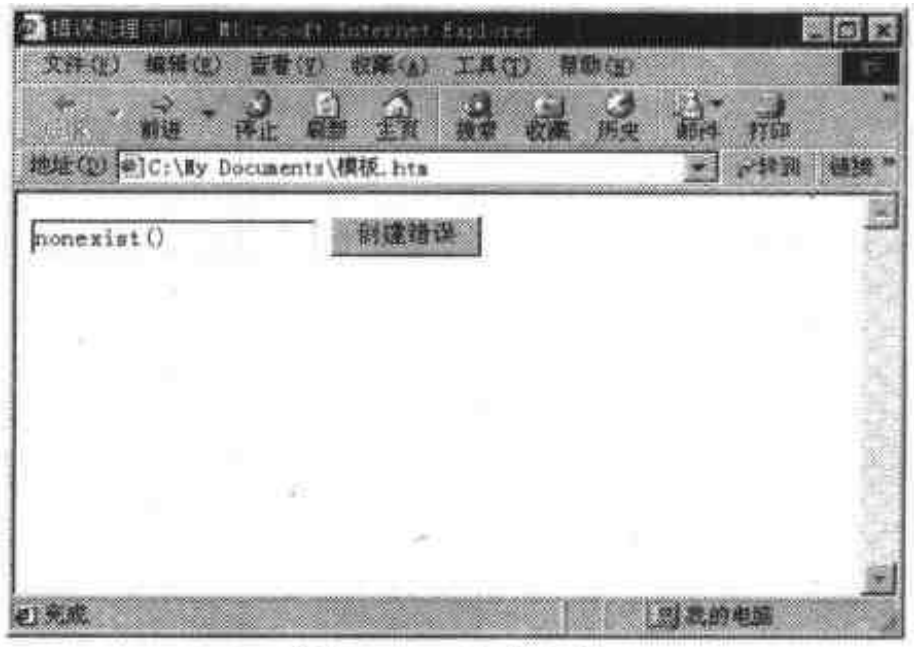


图 3.12　出错之前

当单击“创建错误”按钮时，调用 launchError()函数，该函数执行一个并不存在的 nonexist()函数，从而导致出错。出错后由 onError 事件处理函数进行处理，显示出了错误信息，并禁止浏览器显示标准错误信息，如图 3.13 所示。

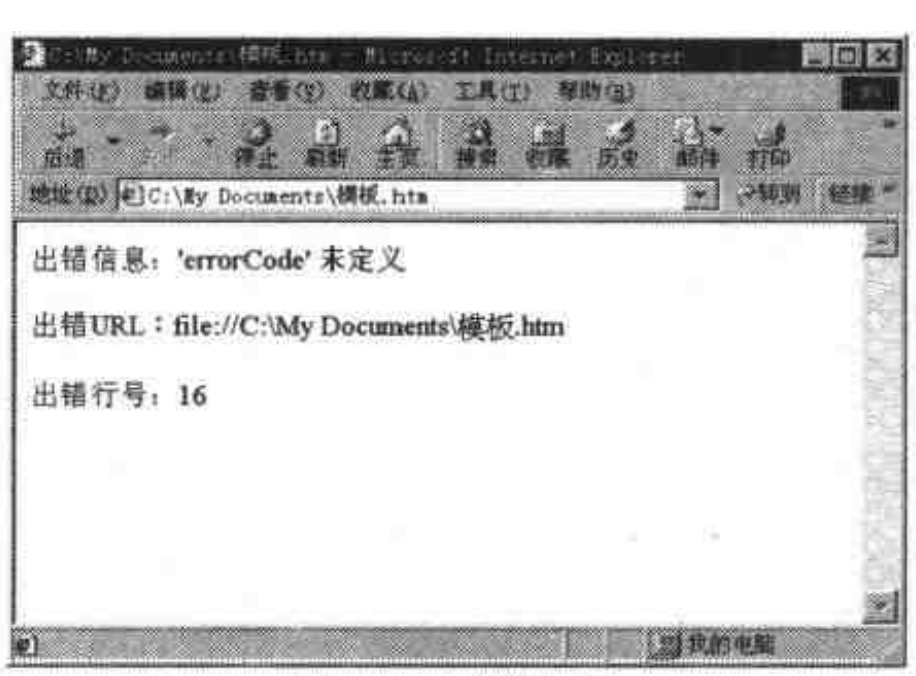


图 3.13　出错之后

3.4.2 错误处理语句

在 Internet Explorer 5 中支持 JavaScript 中的若干错误处理语句，它们是：throw、try 和 catch 语句。

1. 语法

throw 语句用于扔出异常，其语法如下：

```
throw expression;
```

其中 expression 表达式的值表示发生的错误类型，通常应使用一个字符串值，以确保代码更容易理解和调试。例如，以下语句扔出一个"输入错误"异常：

```
throw "输入错误";
```

try 和 catch 语句须结合使用，一起支持异常处理的过程，其语法如下：

```
try
{  statements;  //扔出异常
}
catch(exception)
{  statements;  //处理异常
}
```

如果在处理 try 语句所包含的语句时发生异常，则控制立即转入 catch 语句所包含的语句，并将出错信息保存在 exception 中；如果处理 try 语句所包含语句时没有发生异常，则跳过 catch 语句，控制转入 catch 语句后面的语句。

try 和 catch 语句还可以嵌套，以便实现多层异常处理。如果低层 try 语句中发生异常，则由低层 catch 语句处理；如果低层 catch 语句处理异常时又遇到异常，则把新的异常扔给高层 catch 语句进行处理。

2. 示例 1

本示例演示了基本的异常处理过程，代码如下：

```
<HTML>
<HEAD>
<TITLE>异常处理语句示例</TITLE>
<SCRIPT LANGUAGE ="JavaScript"TYPE="text/javascript">
<!--
```

```
function errorHandler(data)
{
try{
if(data=='string")
 throw "Err1"
else throw "Err2"
}catch(exception){
if(exception=="Err1")
  return("Error("+exception+"):输入内容必须是数字！")
else
  return("Error("+exception+"):输入内容必须是数字！")
}
}
function processData(form)
{
if(isNaN(parseInt(form.myText.value)))
 alert(errorHandler("string"))
else
 alert("输入正确!")
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM name=form1>
请输入一个数字：<P>
  <INPUT name="myText">
  <INPUT type=button value="数据处理" onclick="processData(this.form)">
</FORM>
</BODY>
</HTML>
```

执行该段代码时，如果在文本框中输入的内容不是以数字开头，则导致异常，效果如图 3.14 所示。

3. 示例 2

本示例将上一个示例的 errorHandler()函数改为采用嵌套 try ... catch 语句，如下所示（其他代码都与上一个示例一样，不再重复）。

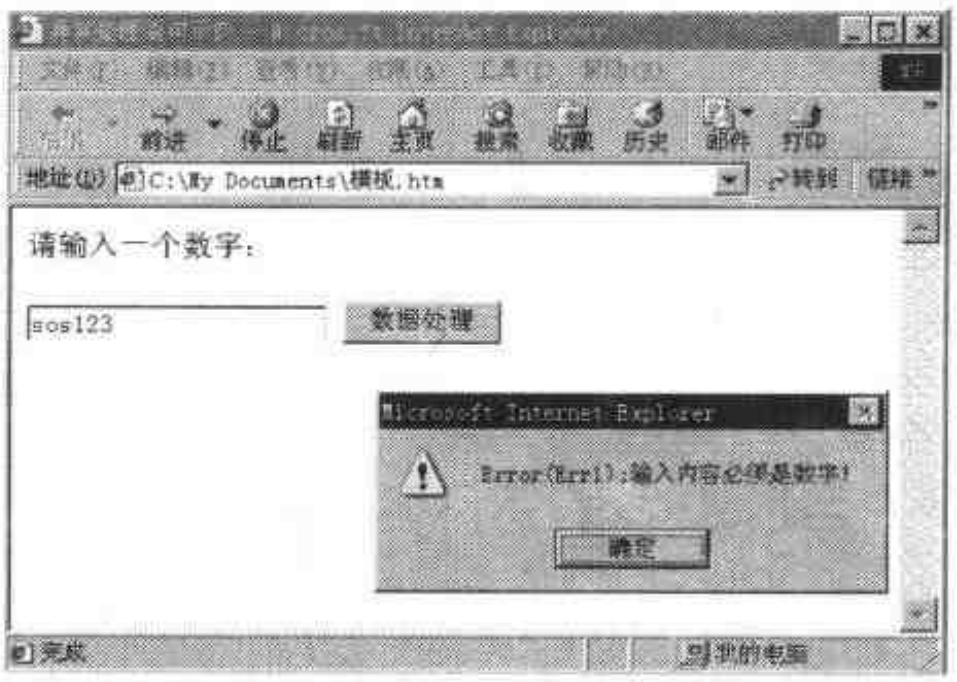


图 3.14　异常处理

```
function errorHandler(data)
{
try{
try{
if(data=="string")
  throw "Err1"
else throw "Err2"
}
catch(e)
{
if(e=="Err1")
  return("Error("+e+"):输入内容必须是数字！")
else
  throw e  //将异常扔到上一级异常处理
}
 }
catch(e)
{
  return('Error('+e+'):输入非法！')
}
}
```

使用此段代码的效果与上一个示例一样，但显示了嵌套 try ... catch 语句的用法。

第 4 章　文档对象

Document（文档）对象是一种最基本的浏览器对象，它代表当前在浏览器窗口中打开的文档。使用 document 对象可以访问页面上的各种元素，通过控制这些页面元素可以实现需要的效果或功能。

本章介绍有关浏览器中 document 对象的基础知识和实际应用，主要内容包括：

- 浏览器对象简介
- document 对象的属性
- document 对象的事件
- document 对象的方法

4.1　浏览器对象简介

在本书第 2 章中已经说明，JavaScript 中包括两种对象——内置对象和浏览器对象。内置对象包括一些常用的通用对象，例如数组对象 Array、字符串对象 String、日期对象 Date 等，这些都已经在第 2 章中介绍过；而浏览器对象是指支持 JavaScript 的浏览器在装入 Web 页面时创建出的多个 JavaScript 对象，可以通过这些对象访问 Web 页面中的各种元素，获得相应的操作效果，例如我们经常用到的 document 对象、form 对象等。

从本章开始我们将陆续介绍各种常用的浏览器对象，利用这些对象可以实现各种客户端应用。

4.1.1　文档对象模型

文档对象模型（Document Object Model，简称 DOM）是用于表示 HTML 元素以及 Web

浏览器信息的一个模型，它使脚本能够访问 Web 页上的信息，并可以访问诸如网页位置等特殊信息。通过操纵文档对象模型中对象的属性并调用其方法，可以使脚本按照一定的方式显示 Web 页并与用户的动作进行交互。

对于不同的脚本语言，通常都具有一个 DOM 的子集，以便在特定的脚本语言中实现对象模型。例如，JavaScript 语言中就有一个对象模型。对于 Internet Explorer，Microsoft 公司专门为其创建了一个对象模型。使用为浏览器创建对象模型的方式使得对象模型与语言无关，从而可以获得更强的可扩展性。

JavaScript 对象模型和 IE 对象模型非常相象，它们包含相似的对象和事件，反映了如图 4.1 所示的对象层次结构。

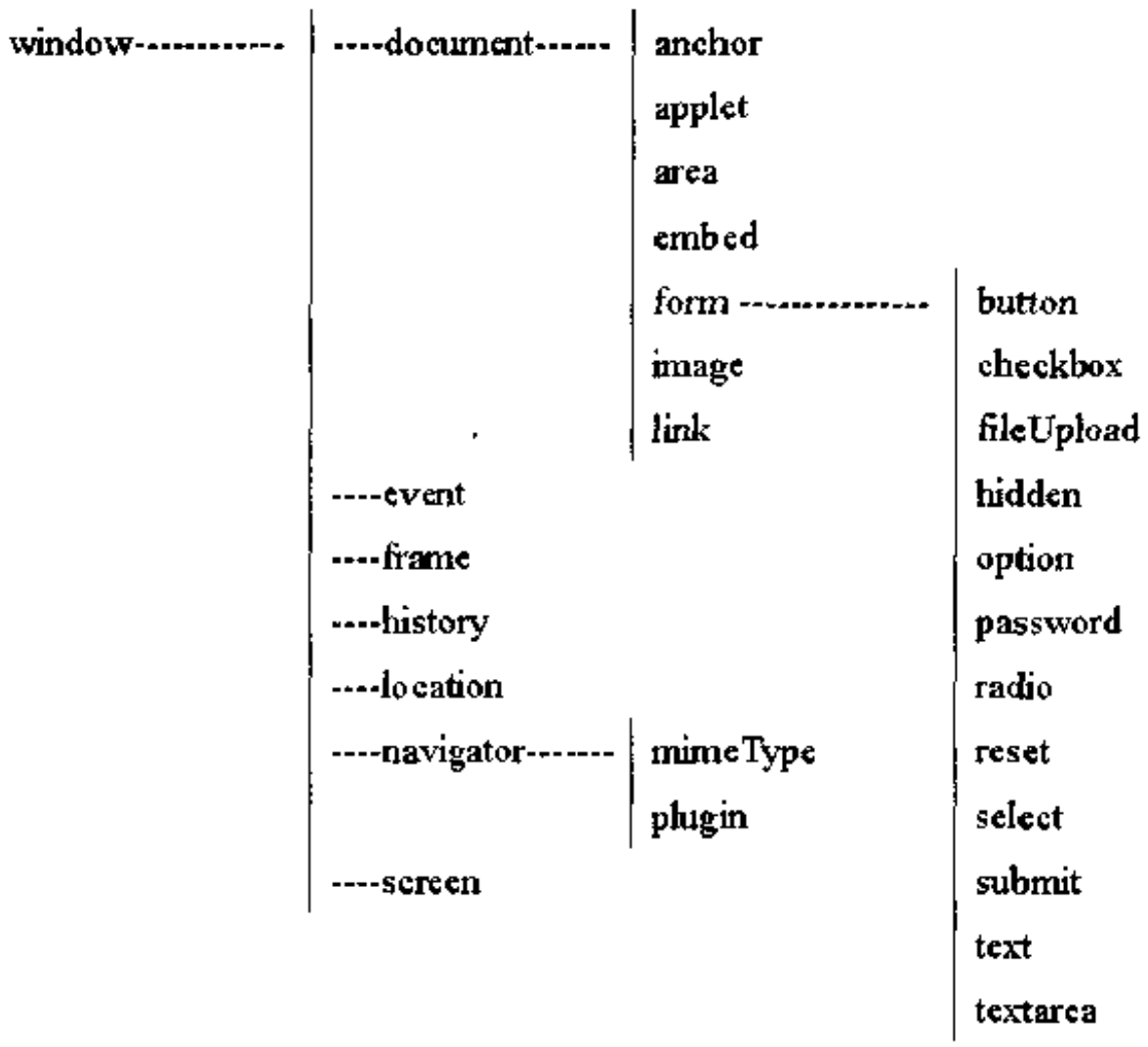


图 4.1 文档对象模型

在该层次结构中，最高层的对象是窗口对象（window），它代表当前的浏览器窗口；之下是文档（document）、事件（event）、框架（frame）、历史（history）、地址（location）、浏览器（navigator）和屏幕（screen）对象；在文档对象之下包括表单（form）、图像（image）和链接（link）等多种对象；在浏览器对象之下包括 MIME 类型对象（mimeType）和插件（plugin）对象；在表单对象之下还包括按钮（button）、复选框（checkbox）、文件选择框（fileUpload）等多种对象。

4.1.2 对象引用方法

了解了浏览器对象的层次结构之后，我们就可以用特定的方法引用这些对象，以便在脚本中正确地使用它们。

在 JavaScript 中引用对象的方式与典型的面向对象方法相同，都是根据对象的包含关系，使用成员引用操作符（.）一层一层地引用对象。例如，如果要引用 document 对象，应使用 window.document；如果要引用 location 对象，应使用 window.location。由于 window 对象是默认的最上层对象，因此引用它的子对象时，可以不使用 window.，也就是说，可以直接用 document 引用 document 对象，用 location 引用 location 对象。

当引用较低层次的对象时，一般有两种方式：使用对象索引或使用对象名称（或 ID）。例如，如果要引用文档中的第一个表单对象，则可以用 document.forms[0]来引用；如果该表单的 name 属性为 form1（或者 ID 属性为 form1），则可以用 document.forms["form1"]或直接用 document.form1 来引用该表单。同样，如果在名称为 form1 的表单中包括一个名称为 myText 的文本框，则可以用 document.form1.myText 来引用该文本框对象。

说明： 由于 name 和 ID 属性的值使用相同的名字空间，因此最好只使用其中一种，以免造成混淆。也就是说，在 FORM 标记符中，最好只指定 name 属性和 ID 属性中的一种，并且不论是名称还是 ID，都不能重复。

对应于不同的对象，通常还有一些特殊的引用方法。例如，如果要引用表单对象中包含的对象，可以使用 elements 数组；如果要引用文档对象中包含的某个标记符对象（例如 P 对象），可以使用 document 对象的 all 属性，等等。对于这些特殊的引用方法，我们将在遇到时具体说明。

4.2 document 对象的属性

document 对象代表当前浏览器窗口中的文档，使用它可以访问到文档中的所有其他对象（例如图像、表单等），因此该对象是实现各种文档功能的最基本对象。本节首先介绍 document 对象的各种属性，之后的两节将分别介绍 document 对象的事件和方法。

4.2.1 属性列表

表 4.1 列出了 document 对象的各种属性以及相应的说明。

表 4.1 document 对象的属性

属 性	说 明
alinkColor	表示活动超链接的颜色
all	表示文档中所有 HTML 标记符的数组
anchors	表示文档中所有锚的数组，锚是指带有 name 属性的 A 对象
applets	表示文档中所有 JAVA 小应用程序
bgcolor	表示文档的背景颜色
cookie	表示与文档相关的 Cookie
domain	表示提供文档的服务器域
embeds	表示文档中所有嵌入对象的数组
fgColor	表示文档的前景颜色
forms	表示文档中所有表单的数组
images	表示文档中所有图像的数组
lastModified	表示文档的最后修改日期
linkColor	表示未被访问的超链接的颜色
links	表示文档中所有超链接的数组，超链接是指带有 href 属性的 A 或 Area 对象
referrer	表示链接到当前文档的文档的 URL，也就是说，如果当前文档是从另外一个文档调用而进入的，则当前文档的 referrer 属性为该文档的 URL
title	表示文档的标题
URL	表示文档的 URL
vlinkColor	表示已被访问的超链接的颜色

说明： 虽然本书主要针对 Internet Explorer，但为了使所介绍的内容具有更大范围的兼容性，因此不论是属性还是方法，都尽量不介绍仅适用于 IE 的内容——除非必须如此（例如 all 属性）。

4.2.2 用 all 属性访问 HTML 元素

在 document 的属性中，有一个属性非常特殊和重要，它就是 all 属性。通过该属性，用户可以访问文档中的所有 HTML 元素对象。

使用 all 属性访问 HTML 元素对象有三种方法：

- 通过索引或名称直接引用；
- 使用 item() 方法；
- 使用 tags() 方法。

1. 直接引用

用户可以直接用索引或名称的方式引用文档中的任意一个元素对象。例如，如果要引

用文档中的第 5 个元素对象，则可以使用 document.all[4]；如果该对象的 name 属性为 myTag5，则也可以使用 document.all["myTag5"]进行引用。

以下示例显示了如何使用这种引用方式，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 all 引用对象</TITLE>
</HEAD>
<BODY>
<FORM name=form1>
请输入一个数字：<P>
  <INPUT name="myText" >
  <INPUT type=button value="数据处理">
</FORM>
<HR>
<H2>以下是本文档中出现的所有 HTML 标记：</H2>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<! -
for(i=0; i<document.all.length; i++)  /*由于 all 属性本身是一个数组，因此具有 length 属性*/
 document.write(document.all[i].tagName+",")  //tagName 属性表示元素的标记符名称
//-->
</SCRIPT>
</BODY>
</HTML>
```

执行该段代码的效果如图 4.2 所示。

说明：不论在文档中是否使用了以下标记符（这些标记符在 HTML 文档中可以省略），由 all 数组返回的元素中始终包含它们：HTML、HEAD、TITLE 以及 BODY。另外，与开始标记符相匹配的结束标记符不再作为单独的对象。

2. 使用 item()方法

all 对象的 item()方法提供了用 HTML 元素对象的名称或 ID 访问对象的途径。该方法的返回值为具有指定名称或 ID 的元素，如果找到多个具有相同名称或 ID 的元素，则返回一个数组。

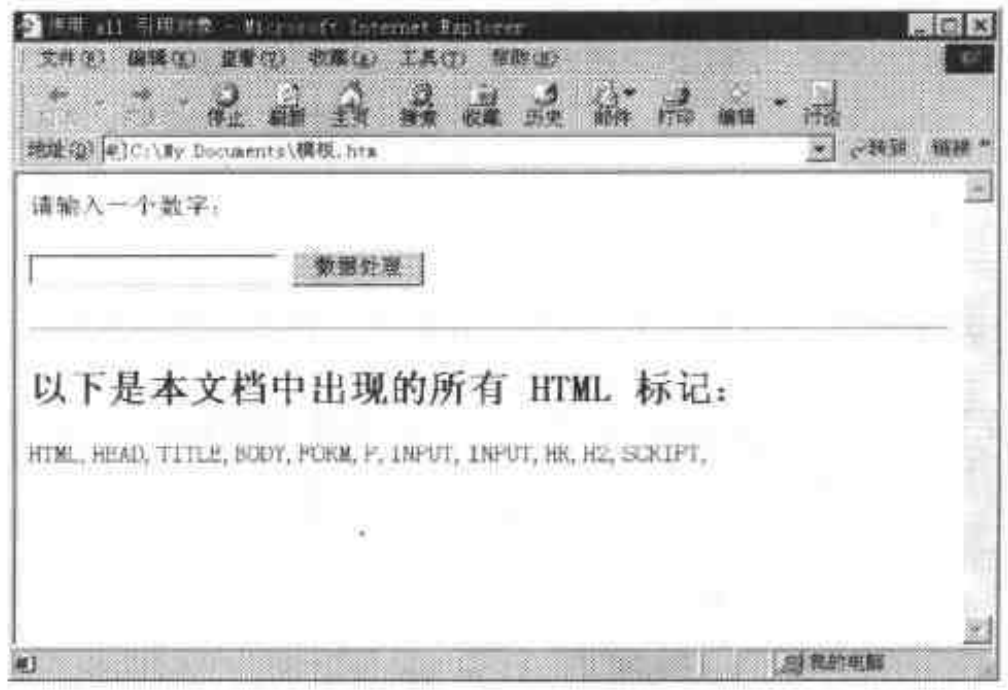


图 4.2　使用 all 属性引用对象

该方法的语法为：document.all.item("string")

以下示例使用 item()方法查找文档中所有名称或 ID 为"test"的元素对象，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 item() 方法</TITLE>
</HEAD>
<BODY>
<FORM name=form1>
请输入一个数字：<P>
  <INPUT name="test" >
  <INPUT type=button value="数据处理">
</FORM>
<HR>
<H2 ID="test">以下是本文档中名称或 ID 为 "test" 的所有 HTML 标记：</H2>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
var result=document.all.item("test");
if(result!=null)
{
if(result.length!=null)
 for(i=0; i<result.length; i++)
   document.write(result[i].tagName+",")
}
else
  document.write("没有发现名称或 ID 为 'test' 的元素！")
//-->
```

```
</SCRIPT>
</BODY>
</HTML>
```

执行该段代码的效果如图 4.3 所示。

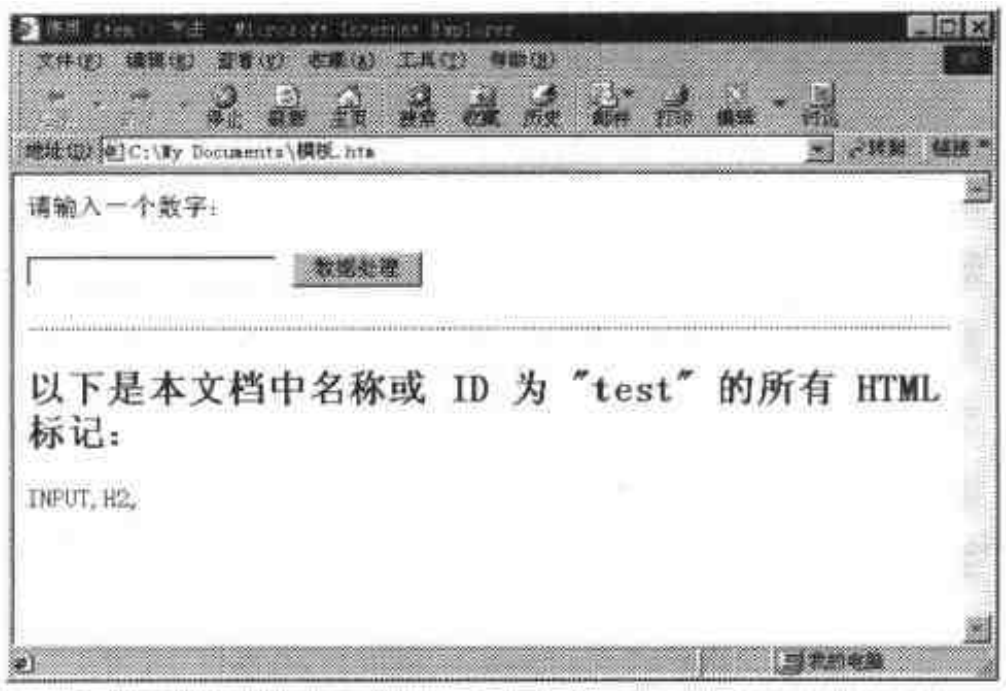


图 4.3　使用 item()方法

3. 使用 tags()方法

all 对象的 tags()方法用来返回文档中具有特定标记符类型的所有 HTML 元素对象，返回值为一个数组。

该方法的语法为：document.all.tags("tagName")

以下示例使用 tags() 方法查找文档中包含的段落数，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 tags() 方法</TITLE>
</HEAD>
<BODY>
<P>tags() 方法用来返回文档中具有特定标记符类型的所有 HTML 元素对象。</P>
<P>返回值为一个数组。</P>
<P>以下示例使用 tags() 方法查找文档中包含的段落数，代码如下。</P>
<HR><BR>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
  document.write("<H2>本文档中包含"+document.all.tags("P").length+"个段落。</H2>")
```

```
//-->
</SCRIPT>
</BODY>
</HTML>
```

执行该段代码的效果如图 4.4 所示。

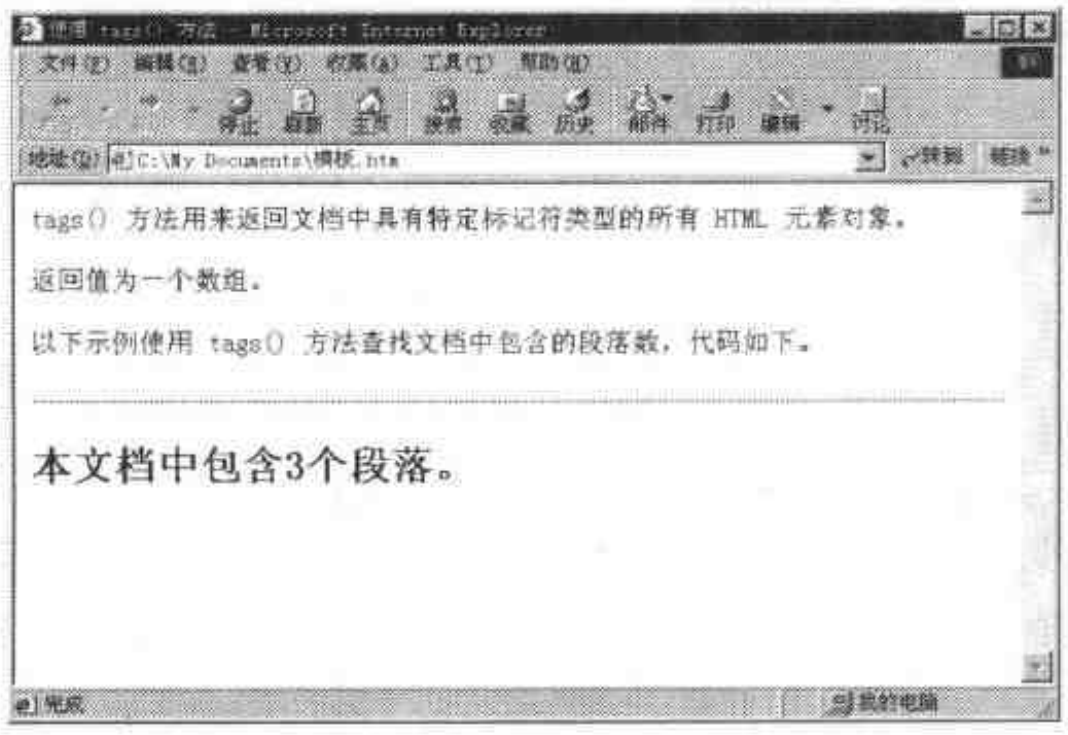


图 4.4 使用 tags()方法

4.2.3 其他属性示例

本小节中的示例显示了除 all 以外其他几个常用属性的用法。

1. 示例 1

本示例列举了文档的标题信息以及其中包含的对象信息，代码如下：

```
<HTML>
<HEAD><TITLE>显示文档信息</TITLE></HEAD>
<BODY>
<A NAME="#top"></A>
<A HREF="http://nonexist.yeah.net"><IMG SRC="./images/pic1.jpg"></A>
<IMG SRC="./images/jieba.gif">
<FORM>
 <INPUT TYPE="BUTTON" VALUE="示例按钮">
</FORM>
<A HREF="http://zhaofengnian.yeah.net">作者主页</A>
<A HREF="#top">返回页首</A>
<HR>
```

```
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
document.write("<H3>本文档的统计信息如下：</H3>")
document.write("本文档的标题为："+document.title+"<BR>")
document.write("本文档的最后修改时间为："+document.lastModified+"<BR>")
document.write("本文档中包含 <B>"+document.links.length+" </B>个超链接
<BR>")
document.write("本文档中包含 <B>"+document.anchors.length+" </B>个锚点
<BR>")
document.write("本文档中包含 <B>"+document.forms.length+" </B>个表单
<BR>")
document.write("本文档中包含 <B>"+document.images.length+" </B>个图像
<BR>")
document.write("本文档中包含 <B>"+document.applets.length+" </B>个 Java
小应用程序<BR>")
document.write("本文档中包含 <B>"+document.embeds.length+" </B>个嵌入对
象<P>")
// -->
</SCRIPT>
</BODY>
</HTML>
```

该段代码的显示效果如图 4.5 所示。

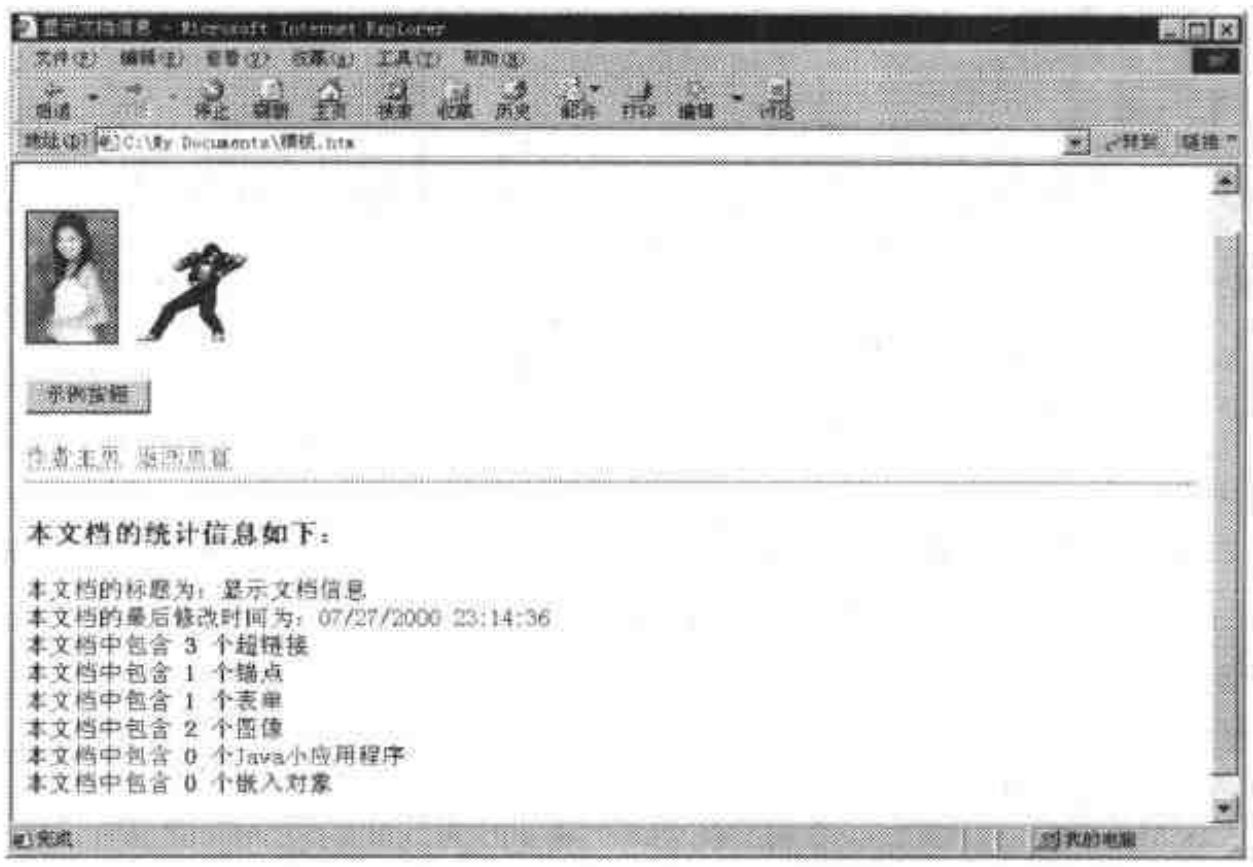


图 4.5　显示文档信息

2. 示例2

本示例显示了如何使用几个表示颜色的属性，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用与颜色相关的属性</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
function changeColor()
{
document.bgColor=document.myForm.myBGColor.value;
document.fgColor=document.myForm.myFGColor.value;
document.linkColor=document.myForm.myLinkColor.value;
document.vlinkColor=document.myForm.myVLinkColor.value;
document.alinkColor=document.myForm.myALinkColor.value;
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<H2>请在以下文本框中输入各种与页面有关的十六进制颜色值（例如：#00ffff）：</H2>
<FORM name="myForm">
<TABLE>
<TR>
  <TD>背景颜色：<TD><INPUT name="myBGColor" value="#ffffff"><P>
<TR>
  <TD>前景颜色：<TD><INPUT name="myFGColor" value="#000000"><P>
<TR>
  <TD>未访问过的超链接的颜色：<TD><INPUT name="myLinkColor"><P>
<TR>
  <TD>已访问过的超链接的颜色：<TD><INPUT name="myVLinkColor"><P>
<TR>
  <TD>活动超链接的颜色：<TD><INPUT name="myALinkColor"><P>
</TABLE>
  <INPUT TYPE="BUTTON" VALUE="单击此按钮更改文档的颜色设置！"
   onclick="changeColor()"><P>
</FORM>
<HR>
<A href="nonexist.com.cn">超链接示例</A>
</BODY>
```

```
</HTML>
```

该段代码的显示效果为：在各文本框中设置了特定的颜色后，单击指定按钮可以应用所设置的颜色；如果不指定某选项的颜色，则该颜色设置保持原来的默认值；如图 4.6 所示。

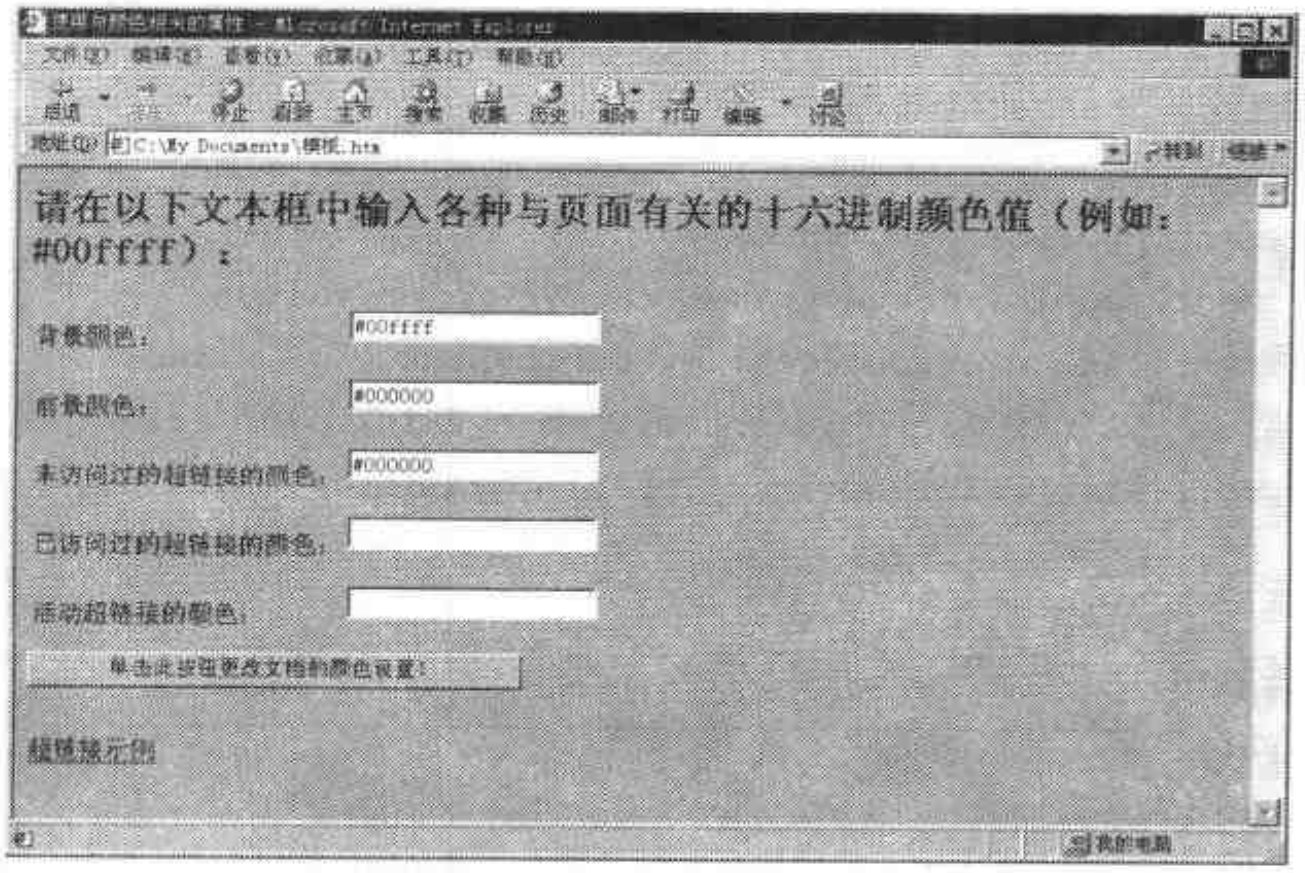


图 4.6　使用与颜色相关的属性

4.3　document 对象的事件

document 对象对应于整个网页对象，它接收发生在网页上的各种事件，同时也是事件起泡机制中的最顶层对象。在上一章中介绍 event 事件时，我们已经介绍过如何响应一些基本的事件，本节对 document 对象的事件响应问题作进一步说明。

4.3.1　处理键盘事件

document 对象可以处理 3 个基本的键盘事件 keydown、keypress 和 keyup，其中 keydown 和 keypress 都是当用户在文档中按下任意键时发生，而 keyup 是在用户释放某键时发生。

在实际应用过程中，最常用的是 keydown 事件，并且经常与 event 事件一起使用。以下示例显示了如何响应用户按下的组合键事件，代码如下：

```
<HTML>
<HEAD>
<TITLE>处理键盘事件</TITLE>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
<!--
function keyHandler()
{
var realKey=String.fromCharCode(event.keyCode);
/* String 对象的 fromCharCode 方法用于返回参数所代表的字符，即将代码转换为字符。
*/
if(event.ctrlKey && event.altKey && event.shiftKey)
  document.myForm.result.value="Ctr+Alt+Shift+"+realKey;
else if (event.ctrlKey && event.altKey)
  document.myForm.result.value="Ctr+Alt+"+realKey;
else if(event.ctrlKey && event.shiftKey)
  document.myForm.result.value="Ctrl+Shift+"+realKey;
else if(event.altKey && event.shiftKey)
  document.myForm.result.value="Alt+Shift+"+realKey;
else if(event.ctrlKey)
  document.myForm.result.value="Ctrl+"+realKey;
else if(event.shiftKey)
  document.myForm.result.value="Shift+"+realKey;
else if(event.altKey)
  document.myForm.result.value="Alt+"+realKey;
}
//-->
</SCRIPT>
</HEAD>
<BODY onkeydown="keyHandler()">
<DIV align="center">
 <H2>请按任意组合键（可以用 Ctrl、Shift 和 Alt 键）</H2>
<BR><BR>
 <FORM name=myForm>
您刚才按下了：
  <INPUT name=result value="">
 </FORM>
</DIV>
</BODY>
</HTML>
```

当执行以上代码时，用户可以先按下 Ctrl、Shift 和 Alt 键的任意组合，然后按任意其他键，则所按的组合键将在文本框中显示，如图 4.7 所示。如果所按下的组合键碰巧又是浏览器定义的快捷键，则在文本框中显示该组合键的同时还将执行浏览器的操作。例如，如果按 Alt+F 键，则会激活浏览器的“文件”菜单。

说明： 本示例显示了如何处理键盘组合键，读者可以根据自己的实际需要添加自己的处理逻辑。

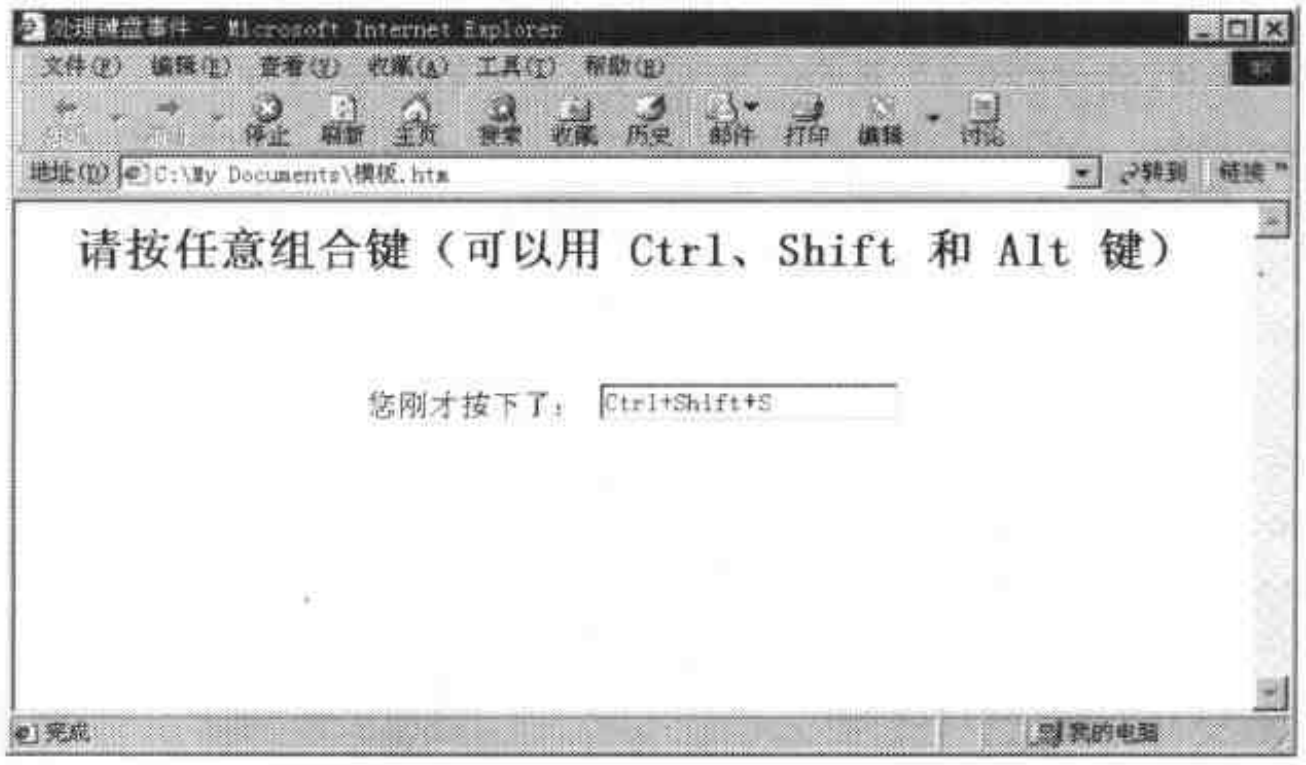


图 4.7 处理键盘事件

4.3.2 处理鼠标事件

document 对象通常处理以下四个基本的鼠标事件：click、dblclick、mousedown 和 mouseup。

以下示例显示了处理鼠标事件的基本机制，代码如下：

```
<HTML>
<HEAD>
<TITLE>处理鼠标事件</TITLE>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
<!--
function mouseDownHandler()
{
document.myForm.start.value=event.x+","+event.y;
}
```

```
function mouseUpHandler()
{
document.myForm.end.value=event.x+","+event.y;
}
//-->
</SCRIPT>
</HEAD>
<BODY onmousedown="mouseDownHandler()" onmouseup="mouseUpHandler()">
<DIV align="center">
 <H2>以下文本框中将显示用户拖放鼠标的起始点和终点：</H2>
<BR>
 <FORM name=myForm>
开始点：<INPUT name=start value=""><P>
终点：<INPUT name=end value="">
 </FORM>
</DIV>
</BODY>
</HTML>
```

这段代码的执行效果是：用户按住鼠标按钮（可以是左键或右键）不放，然后拖动到另外一个位置后释放按钮，则可以在文本框中显示鼠标拖放的起始点和终点，如图 4.8 所示；如果用户在某位置单击鼠标按钮，则起始点和终点的坐标都相同。

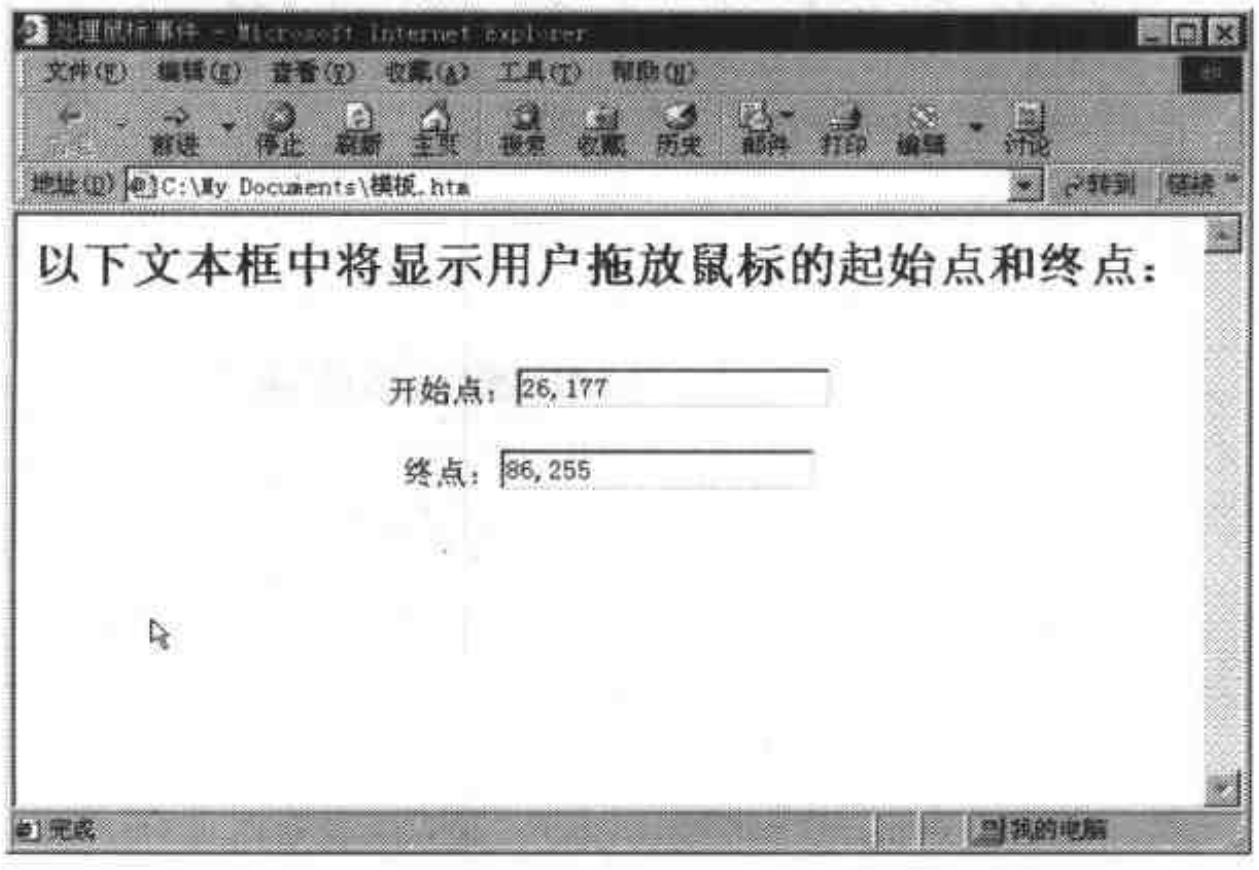


图 4.8 处理鼠标事件

4.3.3 处理加载卸载事件

对于 document 对象还包括两个常用的事件：onload 和 onunload，分别在文档装载完毕和卸载（或从当前页跳转到其他页）完毕时发生。

以下示例显示了这两个事件的作用，代码如下：

```
<HTML>
<HEAD>
  <TITLE>处理加载卸载事件</TITLE>
</HEAD>
<BODY onload="alert('欢迎光临!')"
      onunload="alert('谢谢!')">
<H2>onload 和 onunload 事件示例</H2>
</BODY>
</HTML>
```

这段代码的效果为：当进入页面时，显示“欢迎光临”对话框，如图 4.9 所示；退出网页（单击工具栏中的“前进”按钮跳转到另外一个页面）时，显示“谢谢”对话框，如图 4.10 所示。

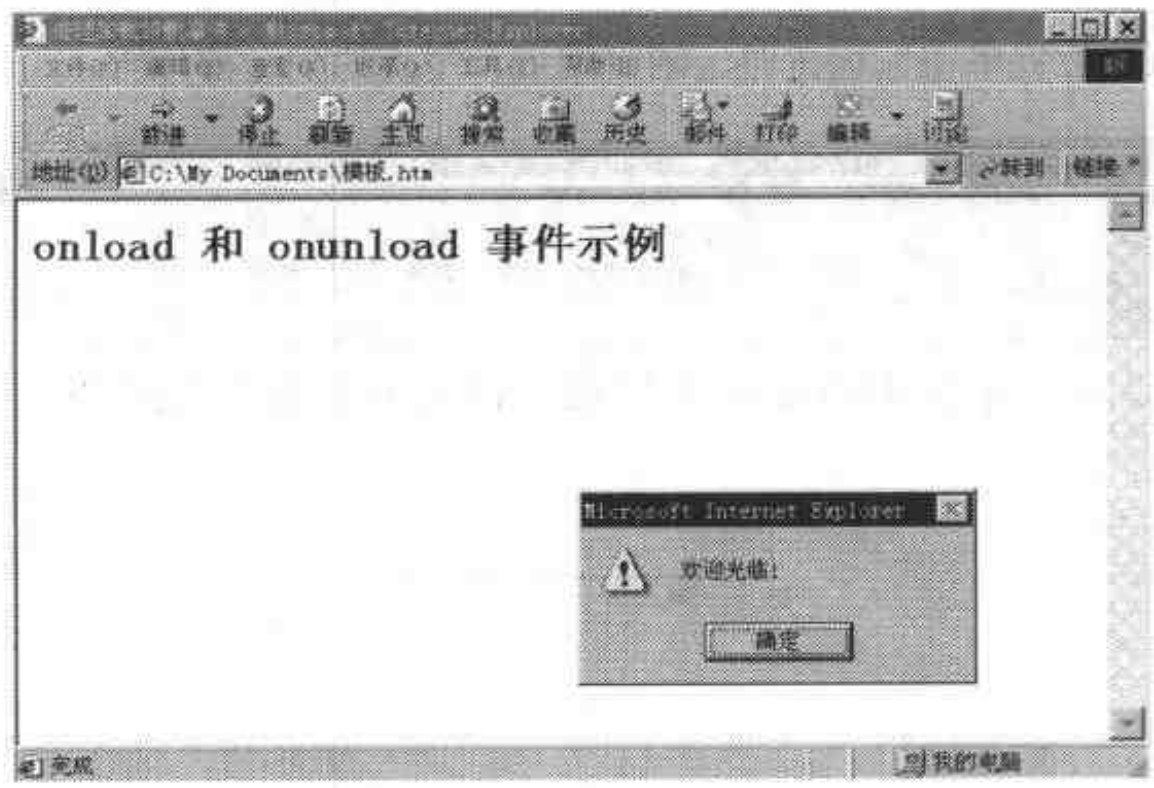


图 4.9　加载卸载事件示例

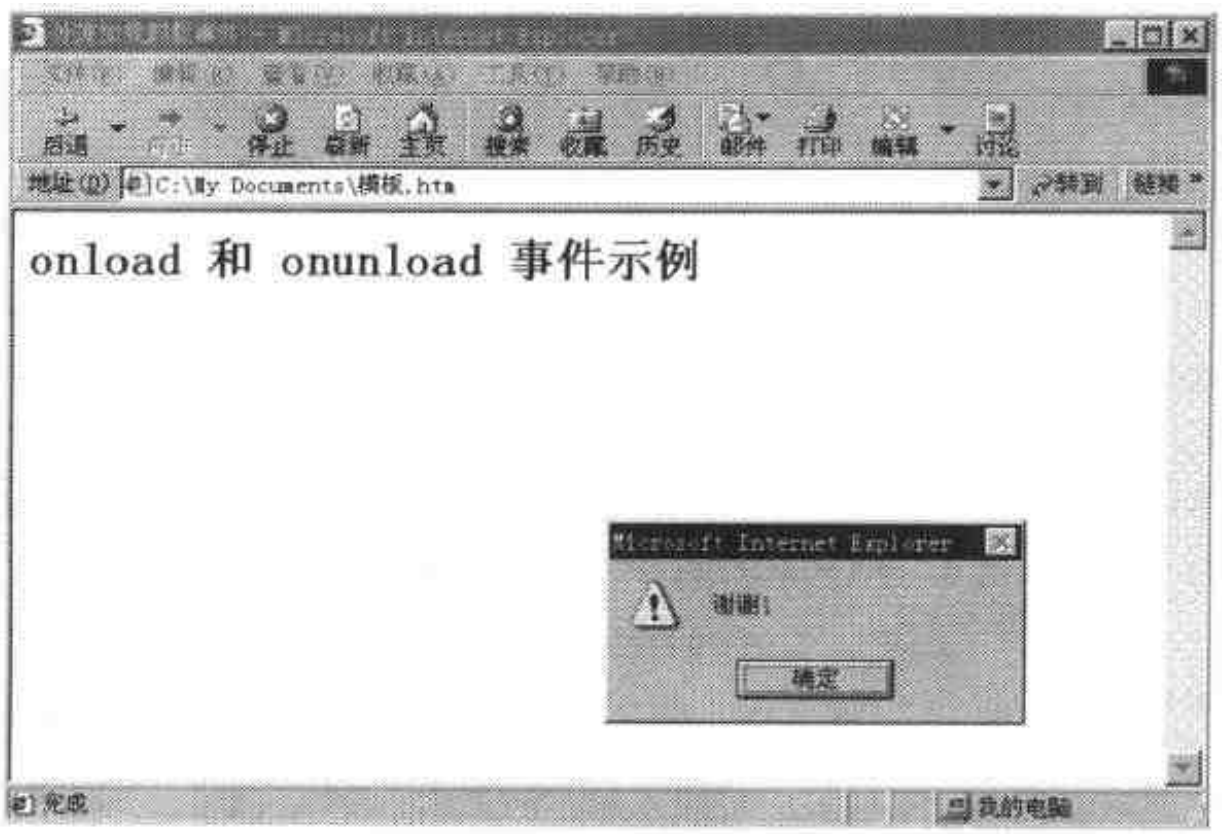


图 4.10 卸载网页时的效果

4.4 document 对象的方法

4.4.1 方法列表

表 4.2 列出了 document 对象的各种常用方法以及相应的说明。

表 4.2 document 对象的方法

方 法	说 明
close()	关闭文档的输出流。调用此方法时，在调用方法之前没有被显示到页面中的输入内容都将被显示出来。
open(mimeType)	清除当前文档并为要放置到该文档中的新数据打开一个流。该方法可以接受一个可选的参数 mimeType，以指定要写到文档中的数据的类型。该参数可以取下面的标准 mimeType 中的一种：text/html、text/plain、image/gif、image/jpeg 或 image/x-bitmap，但 IE 仅支持 text/html。
write(value1,value2...)	将用逗号分隔的参数作为字符串添加到文档中。如果参数有不是字符串，则在被添加到文档之前转换为字符串。
writeln(value1,value2...)	将用逗号分隔的参数作为字符串添加到文档中。与 write() 方法的不同之处在于：writeln() 方法在最后一个参数写入之后在文档中添加一个换行符。如果有参数不是字符串，则在被添加到文档之前转换为字符串。

说明： close() 和 open() 方法中的文档流概念是指构成所建文档的字符串序列。

4.4.2 方法示例

本示例显示了 document 对象各方法的使用，并对比了 write()方法和 writeln()方法的不

同之处，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 document 对象的方法</TITLE>
</HEAD>
<BODY>
<H3>本示例显示了 open()、close()、writeln() 和 write() 方法的应用：</H3>
<SCRIPT Language ="JavaScript" TYPE="text/javascript">
<!--
document.open()
document.writeln("<pre>Hello")
document.writeln("World</pre>")
document.write("<pre>Hello")
document.write("World</pre>")
document.close()
//-->
</SCRIPT>
</BODY>
</HTML>
```

说明：由于在 HTML 中忽略文档中的换行符，因此必须使用预格式化文本标记符 PRE 使换行符显示出来，才能看出 write() 和 writeln() 的区别。PRE 标记符的作用是：按照在文档中输入时的格式显示文档。

该示例的执行效果如图 4.11 所示。

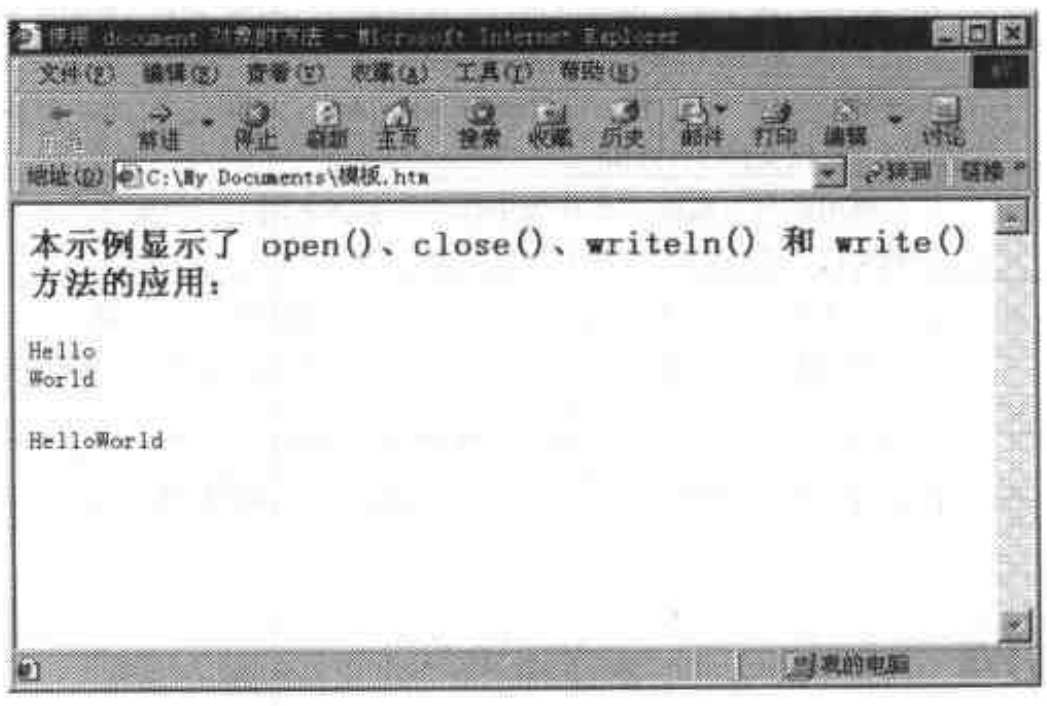


图 4.11　document 对象方法示例

第5章 窗口与浏览器

第5章 窗口与浏览器

window（窗口）对象是文档对象模型中的顶级对象，它表示当前浏览器窗口，是所有其他对象的父对象；frame 对象表示框架文档中的框架，相当于窗口中的窗口；navigator 对象表示当前浏览器，通过它可以访问有关浏览器的信息；screen 对象表示用户屏幕，提供有关屏幕参数的信息。

本章介绍 window、frame、navigator 以及 screen 这 4 个与窗口和浏览器相关的对象，主要内容包括：

- window 对象的属性
- window 对象的方法
- frame 对象
- navigator 对象
- screen 对象

5.1 window 对象的属性

window 对象包含了 document、navigator、location、history 等子对象，是浏览器对象层次中的最顶级对象，代表当前窗口。当遇到 body、frameset 或 frame 标记符时创建该对象的实例，另外，该对象的实例也可由 window.open()方法创建。

5.1.1 属性列表

表 5.1 列出了 window 对象的各种属性以及相应的说明。

表5.1 window对象的属性

属　性	说　明
closed	表明窗口是否关闭的布尔值
defaultStatus	表示窗口状态栏中的默认状态信息
document	表示窗口中显示的当前文档对象
frames	表示窗口中包含的所有 frame 对象的数组
history	表示窗口中最近访问过的 URL 列表
length	表示当前窗口中的框架个数
location	表示窗口中显示的当前 URL
name	表示窗口名称
opener	表示打开当前窗口的窗口
parent	表示当前窗口的上一级窗口（框架）
self	表示当前窗口
status	表示窗口状态栏中的临时信息
top	表示一系列嵌套窗口中的最上层窗口（框架）

5.1.2 窗口的状态信息

除了 document、frames、history、location 等代表窗口中对象的属性以外，其他属性可以分为两类：一类是表示窗口状态信息的属性，一类是作为窗口代名词的属性。

表示窗口状态信息的属性包括：closed、defaultStatus、length、name、status。以下示例显示了这些属性的用法，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 window 的属性</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
function setStatus()
{
window.defaultStatus="默认状态栏文本";
window.name="我的窗口";
}
//-->
</SCRIPT>
</HEAD>
<BODY onmousemove="window.status='当前光标位置坐标:
('+event.x+','+event.y+')'">
<FORM name=form1>
单击以下按钮设置窗口状态：<P>
```

```
  <INPUT type=button value="按钮" onClick="setStatus()">
</FORM>
<SCRIPT LANGUAGE = JavaScript TYPE="text/javascript">
 document.write("<H2>当前窗口的状态信息如下：</H2>")
 document.write("当前窗口的关闭状态为："+window.closed+"<BR>")
 document.write("当前窗口中包含的框架个数为："+window.length+"<BR>")
 document.write("当前窗口的名称为："+window.name+"<BR>")
</SCRIPT>
</BODY>
</HTML>
```

这段代码的执行效果为：单击“按钮”按钮则将窗口的名称设置为“我的窗口”（此后当前窗口的名称始终如此，除非更改），之后单击“刷新”按钮可以显示当前窗口的状态。在此过程中，窗口的状态栏中显示当前光标位置的坐标（虽然设置了 defaultStatus 属性，但由于 Statas 属性的优先级更高，因此并不显示 defaultStatus 的值），如图 5.1 所示。

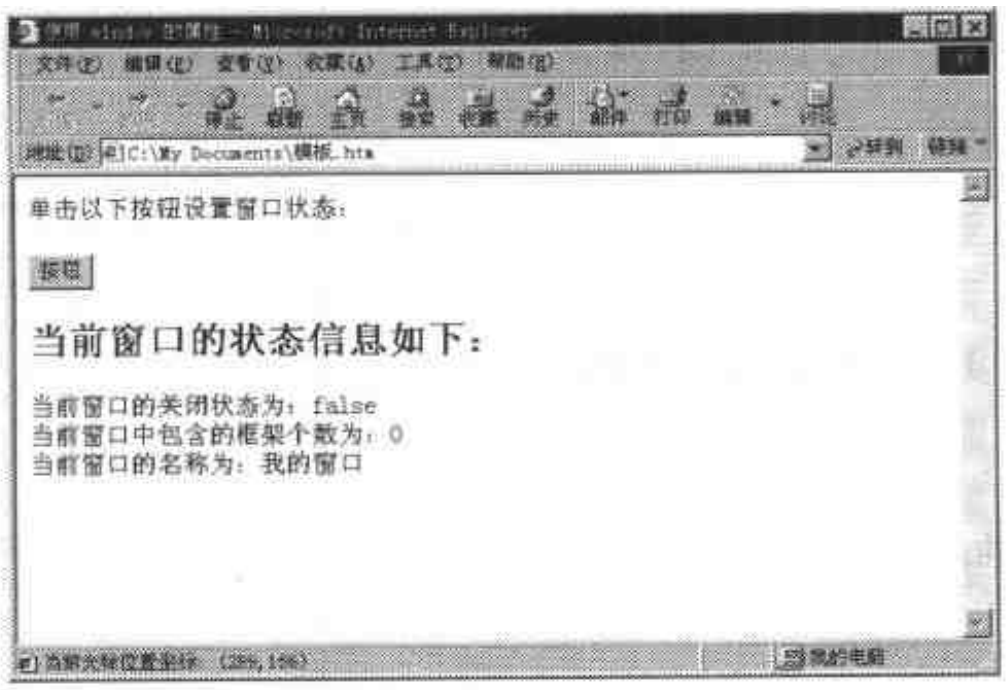


图 5.1 窗口的状态信息

5.1.3 窗口代名词

window 对象的另外几个属性用来表示各种窗口——opener、self、parent 和 top。

1. opener

opener 属性表示打开当前窗口的窗口，或者说返回当前窗口的父窗口。通过这个属性，可以使用父窗口对象中的方法和属性。

以下示例显示了 opener 属性的用法，代码如下：

```
<HTML>
<HEAD>
<TITLE>opener 属性的用法</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
function newWin()
{
  window.name="oldWin"  //设置旧窗口的名称
  window.open("newWin.htm","newWin")  //打开一个新窗口
}
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<H2>单击以下按钮显示新窗口...</H2>
<FORM name=form1>
<BR>
<INPUT TYPE = BUTTON Value = "新窗口" onClick = "newWin()">
</FORM>
</DIV>
</BODY>
</HTML>
```

以下是 newWin.htm 文件的源代码：

```
<HTML>
<HEAD>
<TITLE>新窗口</TITLE>
</HEAD>
<BODY>
<H2 align=center>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
  document.write("此窗口的父窗口是："+window.opener)
</SCRIPT>
</H2>
</BODY>
</HTML>
```

此示例的效果为：在当前窗口中显示一个按钮，如图 5.2 所示；当单击该按钮后，则新建一个窗口，其中显示了有关父窗口的名称信息，如图 5.3 所示。

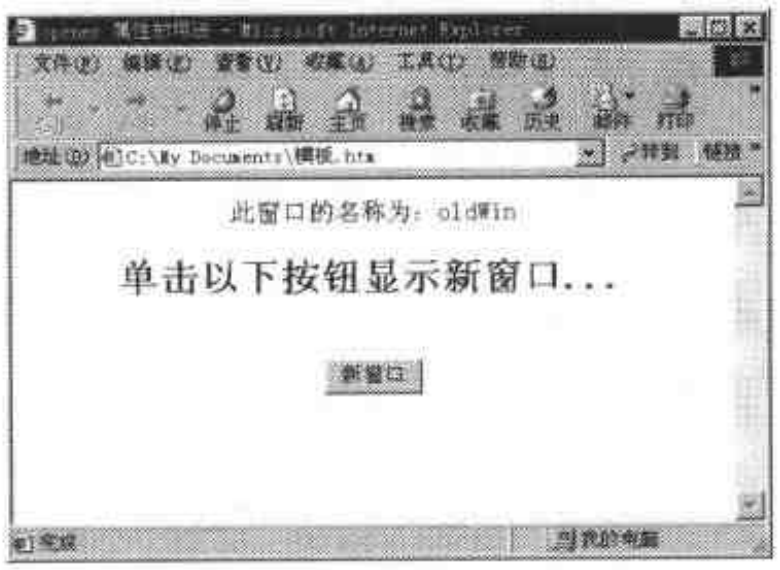


图 5.2 父窗口

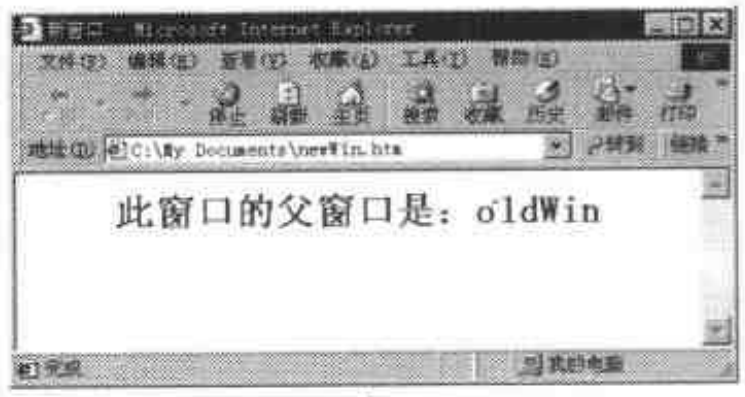


图 5.3 子窗口

2. self

self 属性代表当前窗口。利用这个属性，可以保证在多个窗口被打开的情况下，正确地调用当前窗口内的函数或属性而不会发生混乱。例如，使用 self.close()语句表示关闭当前窗口。

3. Parent 和 top

parent 属性和 top 属性都用于框架文档，分别表示当前框架的父框架和当前框架的最顶层框架。例如，在图 5.4 中，一个浏览器窗口分为上下两部分——框架 A 和框架 B，而框架 B 又分为框架 B1 和框架 B2 两个子框架；此时框架 A 和框架 B 的 parent 与它们的 top 相同，都是最顶层窗口，框架 B1 和框架 B2 的 parent 是框架 B，而所有框架的 top 都是最顶层的窗口。

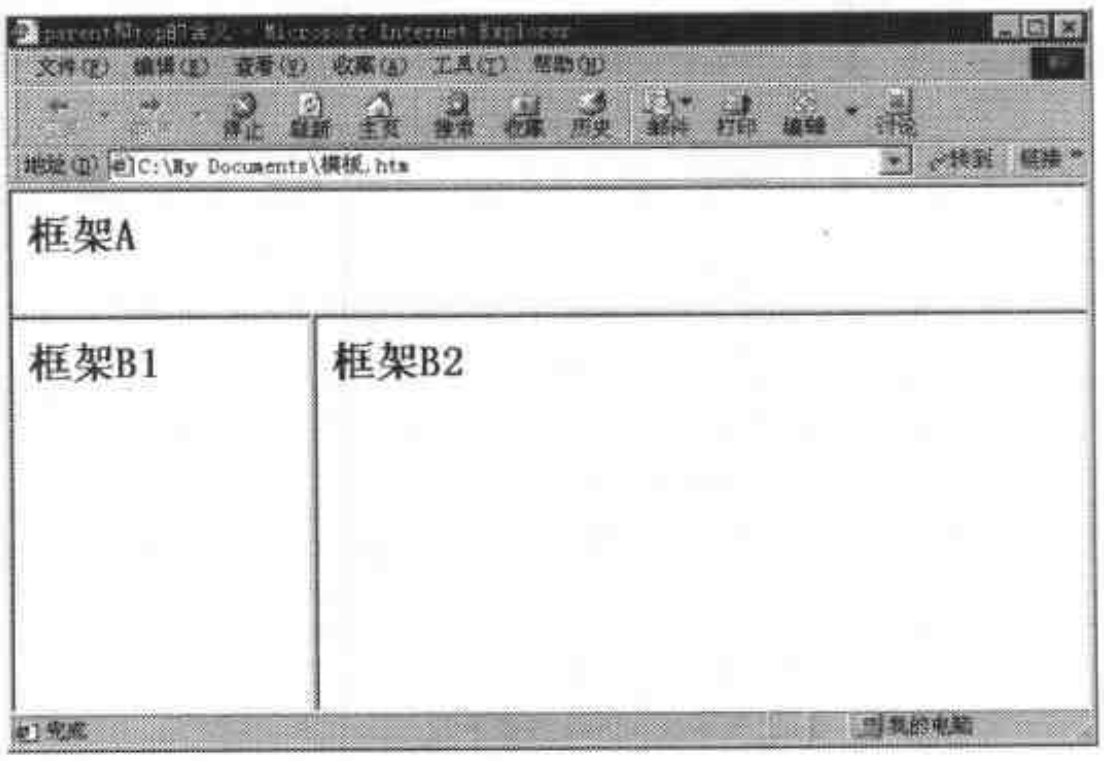


图 5.4 框架示例

5.2 window 对象的方法

window 对象提供了多种方法，用于实现与窗口有关的各种常用功能，例如，打开和关闭窗口、创建各种弹出式对话框以及进行各种定时设置等。

5.2.1 方法列表

表 5.2 列出了 window 对象的各种方法以及相应的说明。

表5.2 window对象的方法

方　法	说　明
alert(string)	显示提示信息对话框
blur()	从窗口中移出焦点
clearInterval(interval)	清除由参数传入的先前用 setInterval() 方法设置的重复操作
clearTimeout(timeout)	取消通过参数传入的先前用 setTimeout() 方法设置的延时操作
close()	关闭窗口
confirm(string)	显示确认对话框，其中包含“确定”和“取消”按钮（或 OK 和 Cancel 按钮），如果用户单击“确定”按钮，confirm()方法返回 true；如果用户单击“取消”按钮，confirm()方法返回 false
focus()	把焦点设置到窗口
moveBy(num1,num2)	按参数指定的像素数移动窗口，其中 num1 表示横向移动的像素数，num2 表示纵向移动的像素数
moveTo(num1,num2)	将窗口移动到指定位置，其中 num1 和 num2 分别表示要移动到的横坐标和纵坐标
open(pageURL,name,parameters)	创建一个新窗口实例，该窗口使用 name 参数作为窗口名，装入 pageURL 指定的页面，并按照 parameters 指定的效果显示
print()	相当于使用“文件”菜单中的“打印”命令，通知浏览器打开“打印”对话框打印当前页
prompt(string1,string2)	弹出一个要求键盘输入的提示对话框，参数 string1 的内容作为提示信息，参数 string2 的内容作为文本框中的默认文本
resizeBy(num1,num2)	按照参数指定的像素数调整窗口的大小，其中 num1 表示横向调整的像素数，num2 表示纵向调整的像素数
resizeTo(num1,num2)	将窗口调整到指定大小，其中 num1 和 num2 分别表示要调整到的横向像素数和纵向像素数
scroll(num1,num2)	按照参数指定的像素数将窗口滚动到指定位置，其中 num1 表示横向滚动的像素数，num2 表示纵向滚动的像素数
scrollBy(num1,num2)	按照参数指定的像素数滚动窗口，其中 num1 表示横向滚动的像素数，num2 表示纵向滚动的像素数
scrollTo(num1,num2)	按照参数指定的像素数将窗口滚动到指定位置，效果与 scroll() 方法完全一样

续表

方　法	说　明
setInterval(expression,milliseconds) setInterval(function,milliseconds,arg1,arg2,...argN)	按照参数 milliseconds 指定的时间间隔，循环触发通过参数传入的表达式（expression）或函数（function）。如果函数需要参数，则由 arg1,...argN 传入。可以用 clearInterval() 方法取消设置的重复操作
setTimeout(expression,milliseconds) setTimeout(function,milliseconds,arg1,arg2,...argN)	按照参数 milliseconds 指定的延时，触发通过参数传入的表达式（expression）或函数（function）。如果函数需要参数，则由 arg1,...argN 传入。可以用 clearTimeout() 方法取消设置的重复操作

5.2.2 打开和关闭窗口

当用户启动浏览器时，浏览器自动打开一个窗口，并在其中显示指定的页面，这是创建 window 对象最常见的一种方式。用户也可以在浏览器中同时打开多个窗口，以便在带宽有限的情况下同时下载和浏览多个页面，此时只要选择浏览器“文件”菜单中“新建”命令中的“窗口”子命令（或直接按“ctrl+N”组合键）即可。用户还可以自行确定在一个新窗口中打开超链接的目标页面，以便保留原来的页面，此时只要在超链接源上单击鼠标右键，然后选择“在新窗口中打开链接”命令即可，如图 5.5 所示（该网页取自 8848 网站）。

图 5.5 在新窗口中打开链接

不过，这些都是浏览器提供的功能，需要用户的具体选择。有时我们需要在网页中自定义一些需要打开新窗口的场景，例如，单击一个按钮打开一个新窗口，并在其中显示特

定内容等，这时可以使用 window 对象的 open()方法在脚本中定义相应逻辑。

open()方法的语法为：open(pageURL,name,parameters)，其中 name 参数作为窗口名，pageURL 参数表示要装入的页面，parameters 参数是一个字符串，其中包含用逗号分开的 option=value 对，用于指定窗口的显示效果。

用于指定窗口显示效果的通用选项如表 5.3 所示。

表5.3 指定窗口显示效果的选项

选　项	说　明
height	窗口高度，取值为像素数
location	确定在窗口中是否显示地址栏，取值为 yes/no 或 1/0
menubar	确定在窗口中是否显示菜单栏，取值为 yes/no 或 1/0
resizable	确定窗口是否可以调整大小，取值为 yes/no 或 1/0
status	确定在窗口中是否显示状态栏，取值为 yes/no 或 1/0
toolbar	确定在窗口中是否显示工具栏，取值为 yes/no 或 1/0
width	窗口宽度，取值为像素数。

说明：在 IE 5 中，如果不指定这些参数，则新建的窗口将使用默认大小并显示所有窗口元素；如果指定了任意的 option=value 对，则其他窗口元素都默认为不显示（请参见稍后的示例）。

使用 open()方法创建了一个 window 对象后，还可以使用 close()方法将其关闭，此时必须使用 open()方法所返回对象的 close()方法。

以下示例显示了如何使用 window 对象的 open()方法和 close()方法，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 open() 和 close() 方法</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
function openWin()
{
var newWin=open("","","menubar=1,height=200");
newWin.document.write("<FORM>")  //省略了其他的基本 HTML 标记
newWin.document.write("单击以下按钮关闭窗口：<P>")
newWin.document.write("<INPUT  TYPE=BUTTON  value=' 关 闭 '  onclick  =
window.close()> ")
newWin.document.write("</FORM>")
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<DIV align=center>
<H2>单击以下按钮显示新窗口...</H2>
<FORM name=form1>
<!--新窗口 1：打开一个默认的空白窗口-->
  <INPUT TYPE = BUTTON Value = "新窗口 1"
        onClick = window.open('','new1')>
<!--新窗口 2：打开一个默认的窗口，指定其中显示的文档-->
  <INPUT TYPE = BUTTON Value = "新窗口 2"
        onClick = window.open('./newWin.htm','myWin2')>
<!--打开一个空白窗口，设置高度和宽度（于是其他窗口元素都不显示）-->
  <INPUT TYPE = BUTTON Value = "新窗口 3"
        onClick = open('','new2','height=200,width=450')>
<!--打开一个空白窗口，并在其中写入文档内容-->
  <INPUT TYPE = BUTTON Value = "新窗口 4"
        onClick = openWin() >
</FORM>
</DIV>
</BODY>
</HTML>
```

使用此示例时，当前目录下必须包含以下 newWin.htm 文件：

```
<HTML>
<HEAD>
<TITLE>新窗口</TITLE>
</HEAD>
<BODY>
<H2>单击以下按钮关闭本窗口：</H2>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "关闭" onClick = window.close()>
</FORM>
</BODY>
</HTML>
```

此示例的显示效果如图 5.6 所示，如果单击“新窗口 1”按钮，显示如图 5.7 所示窗口；如果单击“新窗口 2”按钮，显示如图 5.8 所示窗口；如果单击“新窗口 3”按钮，显示如图 5.9 所示窗口；如果单击“新窗口 4”按钮，显示如图 5.10 所示窗口。

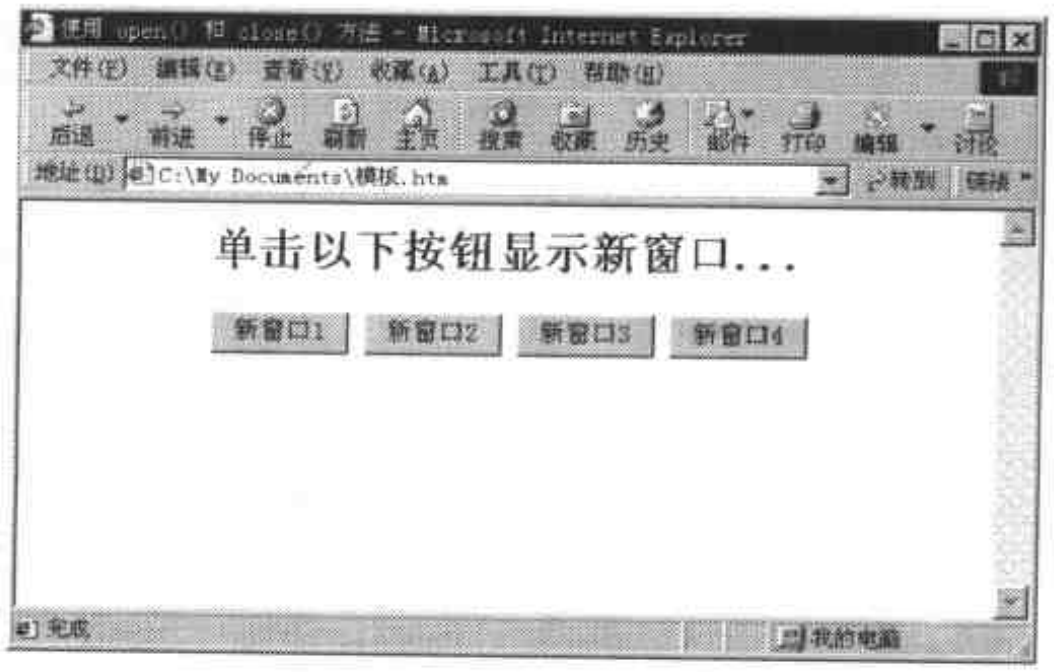


图 5.6　打开窗口示例

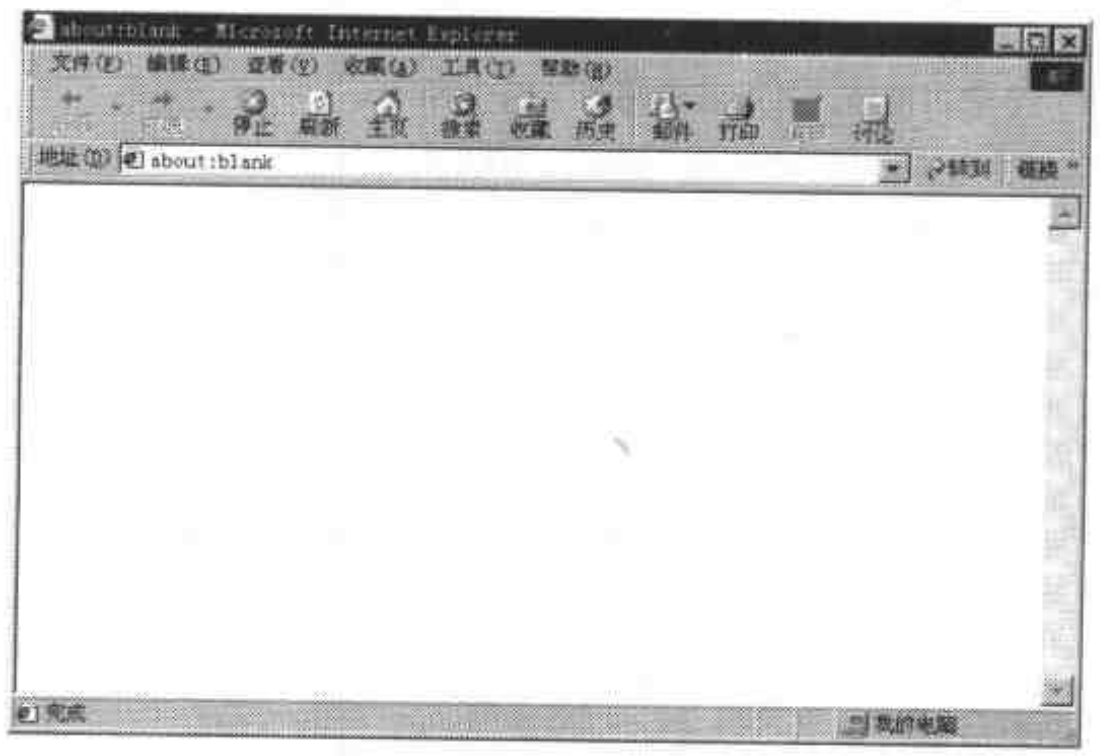


图 5.7　新窗口 1

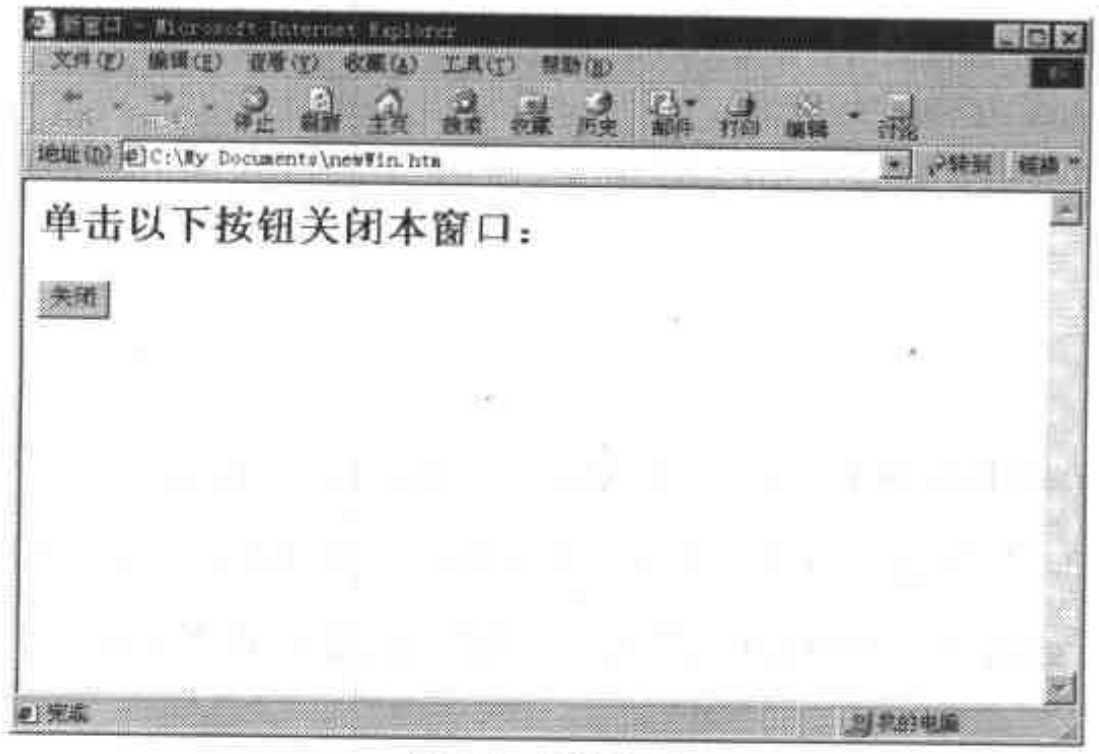


图 5.8　新窗口 2

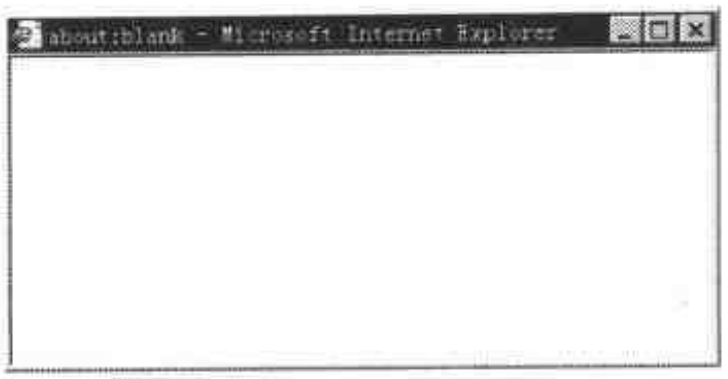


图 5.9　新窗口 3

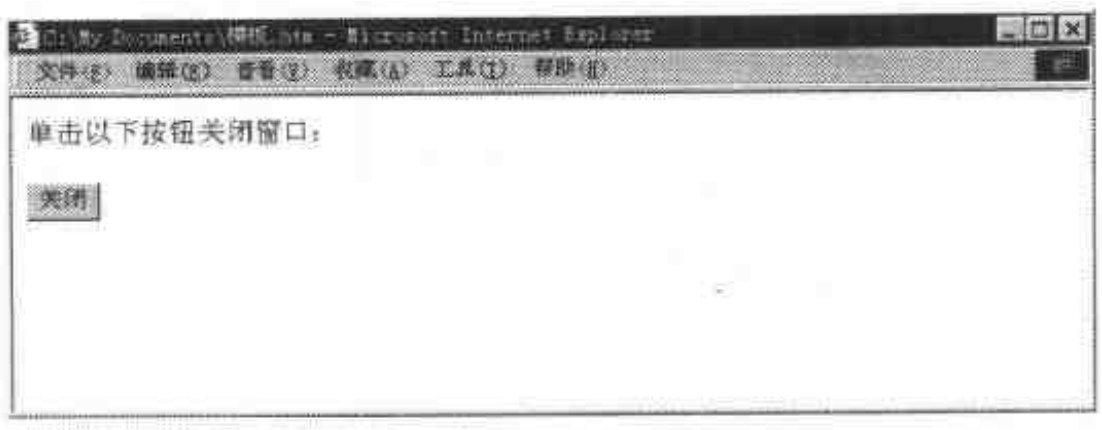


图 5.10　新窗口 4

5.2.3　使用对话框

在用户与应用程序进行交互操作时，经常使用的是对话框这种工具，在网页交互过程中也可以使用对话框。window 对象提供了三种方法：alert()、confirm() 和 prompt()，分别用于显示不同类型的对话框。

1．alert()方法

实际上，在前面的章节中我们已经多次用到了 window 对象的 alert()方法，它的功能是将参数提供的字符串显示在一个提示对话框中，该对话框中包含一个“确定”按钮（对于英文版的浏览器是“OK”按钮），单击该按钮可以关闭该按钮。

以下一个简单的示例显示了 alert()方法的使用，代码如下：

```
<HTML>
<HEAD>
<TITLE>alert() 方法</TITLE>
</HEAD>
<BODY>
<H2>单击以下按钮显示提示对话框：</H2>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "按钮" onClick = window.alert("这是一个
```

```
提示对话框")>
<!--调用 alert() 方法时，可以不使用 window.alert()，而直接使用 alert()。-->
</FORM>
</BODY>
</HTML>
```

该示例的效果如图 5.11 所示。

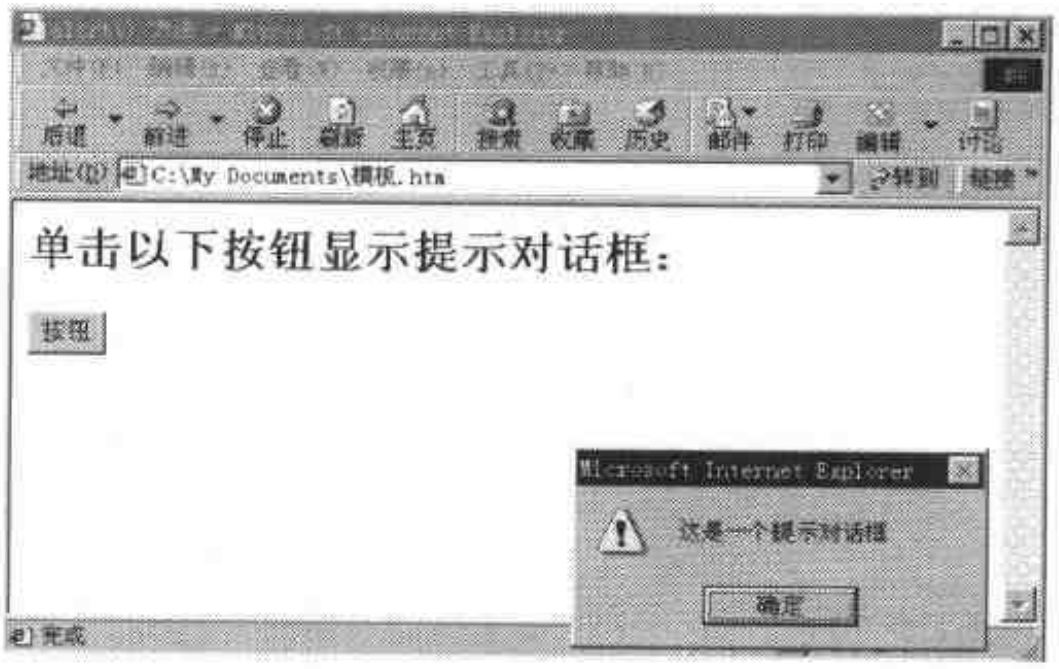


图 5.11　alert()方法示例

2. confirm()方法

使用 window 对象的 confirm()方法，可以创建出包含一个“确定”按钮和“取消”按钮（或“OK”按钮和“Cancel”按钮）的对话框，用来确认用户所作的选择。如果用户选择了“确定”按钮，则 confirm()方法返回 true；如果用户选择了“取消”按钮，则 confirm()方法返回 false。作为 confirm()方法参数的字符串，将与使用 alert()方法时一样出现在对话框中。

以下示例显示了 confirm()方法的使用，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 confirm() 方法</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
function confirmClose()
{
if(confirm("您正在关闭当前窗口，真的要如此吗？"))
  window.close()
// 调用 confirm() 方法时，可以不使用 window.confirm()，而直接使用 confirm()。
```

```
}
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<H2>单击以下按钮关闭窗口...</H2>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "关闭" onClick = confirmClose()>
</FORM>
</DIV>
</BODY>
</HTML>
```

使用该示例的效果是：当单击“关闭”按钮时，弹出一个确认对话框，如图 5.12 所示。如果用户单击“确定”按钮，则确实关闭窗口，如果用户单击“取消”按钮，则什么也不做。

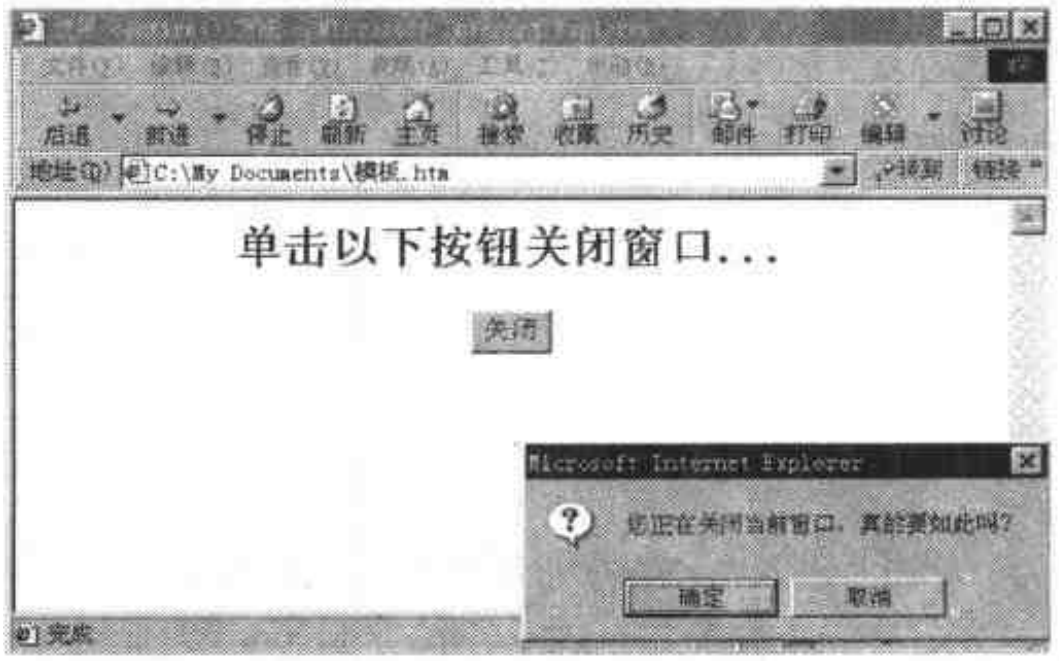


图 5.12 confirm()方法示例

3. prompt()方法

除了以上两种要求用户直接单击按钮作出选择的对话框以外，在 JavaScript 中还可以有一种要求用户进行键盘输入的对话框，实现这种对话框的方法是使用 window 对象的 prompt()方法。

prompt()方法包含两个字符串参数，第一个参数表示要在对话框中显示的信息，第二个参数表示文本框中的默认文本。在显示的对话框中包含“确定”和“取消”按钮（或“OK”和“Cancel”按钮），如果用户在该对话框中单击“确定”按钮，则该方法的返回值为用户在文本框中输入的内容或其初始值（如果用户没有输入）；如果用户单击“取消”按钮，则 prompt()方法返回 null。

以下示例显示了 prompt()方法的用法，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 prompt() 方法</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
function promptPwd()
{
while(prompt("请输入密码：","password")!="admin")
  alert("密码错误，请重新输入。");
// 调用 prompt() 方法时，可以不使用 window.prompt()，而直接使用 prompt()。
}
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<H2>单击以下按钮输入密码...</H2>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = '输入密码' onClick = promptPwd()>
</FORM>
</DIV>
</BODY>
</HTML>
```

该示例的效果为：单击“输入密码”按钮后弹出一个要求用户输入的对话框，如图 5.13 所示；输入之后单击“确定”按钮，如果输入正确（即输入“admin”）则返回窗口，否则弹出一个提示对话框要求继续输入，如图 5.14 所示。

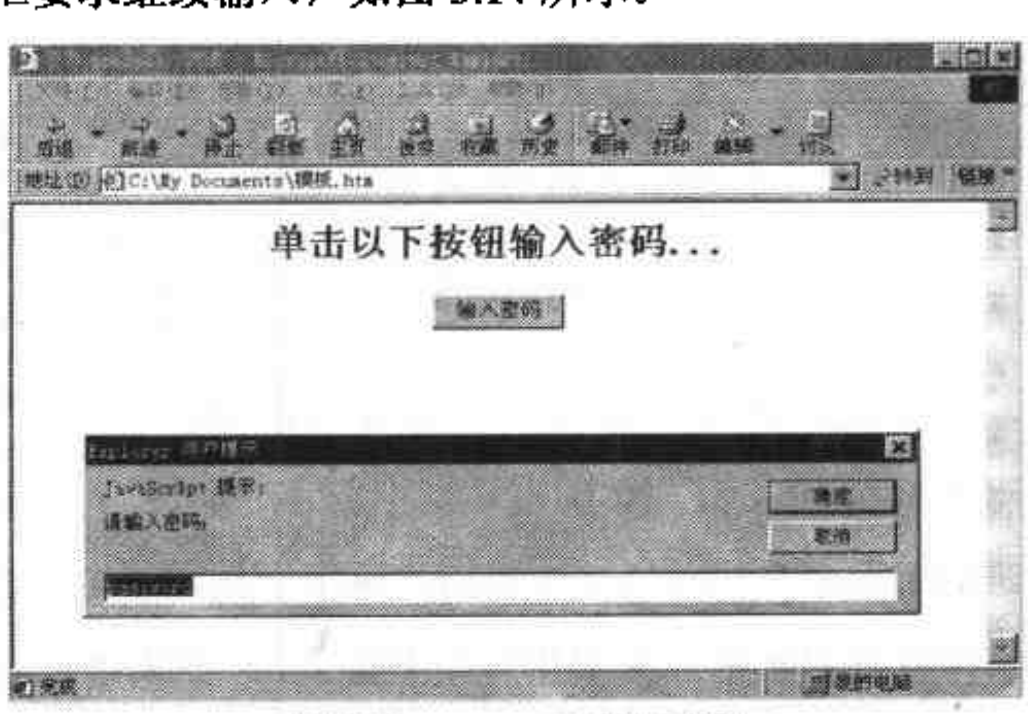


图 5.13　prompt()方法示例

图 5.14　继续提示

说明：用这种方式实现密码确认只具有最低级别的安全性，因为客户端脚本的所有内容都随着网页一起下载到浏览器端。如果要实现具有进一步安全性的密码确认，应使用服务器端脚本。

5.2.4　定时设置

在 window 对象中包含四种方法用于进行定时设置，其中 setInterval()和 clearInterval()方法用于设置和取消循环定时操作，setTimeout()和 clearInterval()方法用于设置和取消延时定时操作。

1．setInterval()和 clearInterval()方法

setInterval()方法的语法如下：

setInterval(*expression,milliseconds*)

或

setInterval(*function,milliseconds,arg1,arg2,...argN*)

其中 milliseconds 参数表示循环操作的定时时间间隔（以毫秒为单位），expression 表示定时执行的表达式，function 表示定时执行的函数，如果函数有参数，则用 arg1,...argN 表示。

该方法的含义是：每隔 milliseconds 毫秒，执行 expression 或 function 一次。由于它具有循环定时执行的特性，因此经常用于动态内容的显示（第 7 章中将介绍该方法在创建动画中的应用）。

设置了定时重复操作之后，还可以用 clearInterval()方法取消，该方法的语法为：

clearInterval(interval)

其中，interval 参数表示 setInterval()方法返回的值。

例如，以下示例显示了如何用 setInterval()和 clearInterval()方法进行定时操作，代码如下：

```
<HTML>
```

```
<HEAD>
  <TITLE>使用 setInterval() 和 clearInterval() 方法</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var myTimer="";

function startTimer()
//此函数调用 setInterval() 方法，每隔一秒钟调用一次 acquireTime() 函数
{
myTimer=window.setInterval("acquireTime()",1000);
}

function stopTimer()
{
window.clearInterval(myTimer);
}

function acquireTime()
{
today=new Date();  //获取当前日期
with(today)
{
  document.form1.timer.value=getHours()+"    点    "+getMinutes()+"    分
"+getSeconds()+"秒";
}
}
//-->
</SCRIPT>
<HEAD>
<BODY onLoad="startTimer();">  <!--加载文档时调用 startTimer() 函数，启动
定时器。-->
<DIV align=center>
<FORM name=form1>
<H2>现在时间：</H2>
  <INPUT NAME="timer">
<H2>单击以下按钮停止计时器...</H2>
  <INPUT TYPE = BUTTON Value = "停止" onClick = stopTimer()>
<H2>单击以下按钮重新启动计时器...</H2>
  <INPUT TYPE = BUTTON Value = "启动" onClick = startTimer()>
```

```
</FORM>
</DIV>
</BODY>
</HTML>
```

本示例的效果为：在文档的一个文本框中动态显示当前时间（按秒更新），如图 5.15 所示；如果用户单击“停止”按钮，则时间显示停留在单击该按钮的时间点；如果用户单击“启动”按钮，则继续动态更新文本框中显示的时间。

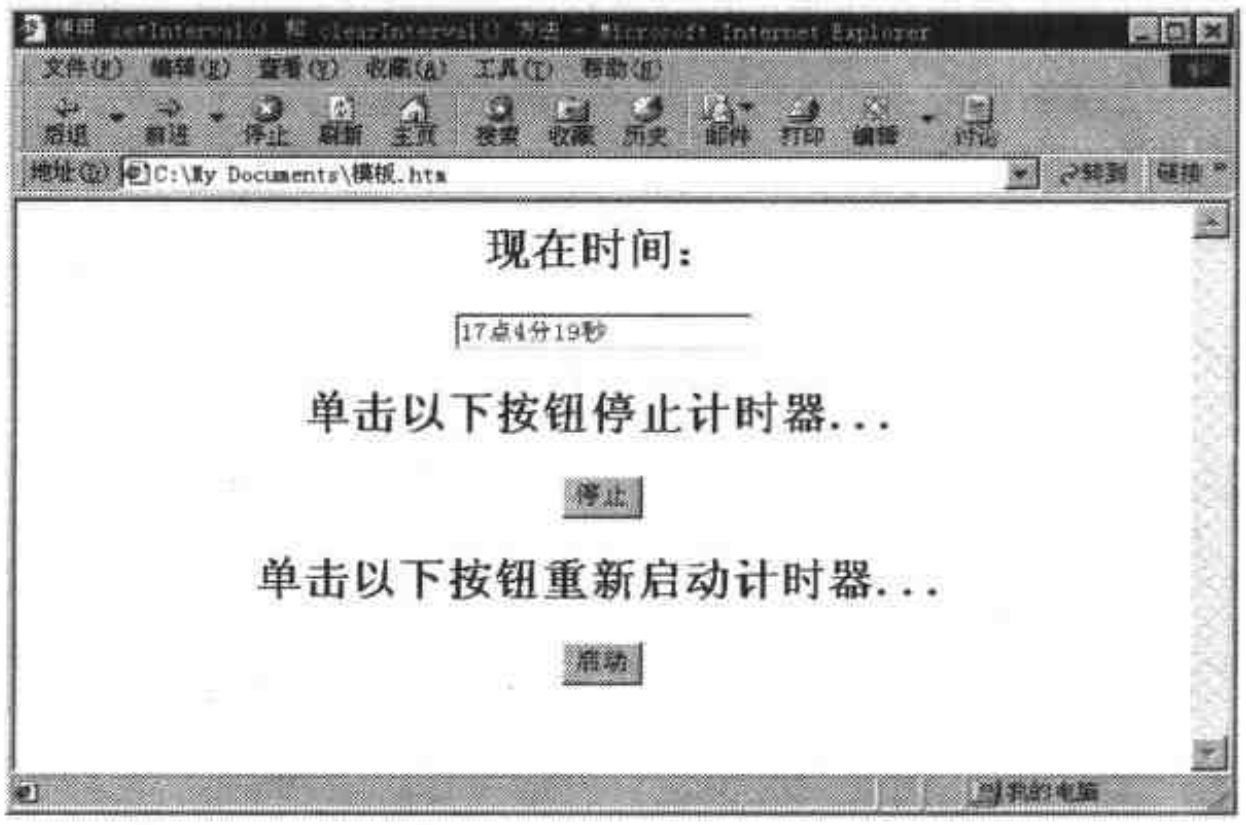


图 5.15 setInterval()方法和 clearInterval()方法示例

2. setTimeout()和 clearTimeout()方法

setTimeout()方法与 setInterval()方法类似，都是在指定时间间隔执行一个表达式或函数，不同的是 setTimeout()方法只在延时到来时执行一次，而 setInterval()方法按指定时间间隔循环执行。

setTimeout()方法的语法如下所示：

setTimeout(*expression,milliseconds*)

或

setTimeout(*function,milliseconds,arg1,arg2,...argN*)

其中 milliseconds 参数表示延时的时间间隔（以毫秒为单位），expression 表示延时到来时执行的表达式，function 表示延时到来时执行的函数，如果函数有参数，则用 arg1,...argN 表示。该方法的含义是：调用方法 milliseconds 毫秒之后，执行 expression 或 function 一次。

同样，也可以在指定时间到来之前用 clearTimeout() 方法取消设置的定时操作，该方

法的语法为：

clearTimeout(timeout)

其中，timeout 参数表示 setTimeout()方法返回的值。

例如，以下示例显示了如何用 setTimeout()和 clearTimeout()方法进行定时操作，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 setTimeout() 和 clearTimeout() 方法</TITLE>
<SCRIPT LANGUAGE='JavaScript" TYPE="text/javascript'>
<!--
var myTimer="";

function startTimer()
{
myTimer=window.setTimeout("alert('太晚了！！！')",5000);
}

function clearTimer()
{
window.clearTimeout(myTimer);
alert("已经清除了！")
}
//-->
</SCRIPT>
<HEAD>
<BODY onLoad="startTimer()">
<DIV align=center>
<FORM name=form1>
<H2>请在 5 秒之内单击以下按钮……</H2>
  <INPUT TYPE = BUTTON Value = '清除' onClick = clearTimer()>
</FORM>
</DIV>
</BODY>
</HTML>
```

本示例的效果为：如果在打开页面 5 秒之内单击“清除”按钮，则显示“已经清除了”提示框，如图 5.16 所示；否则将显示“太晚了”提示框，如图 5.17 所示。

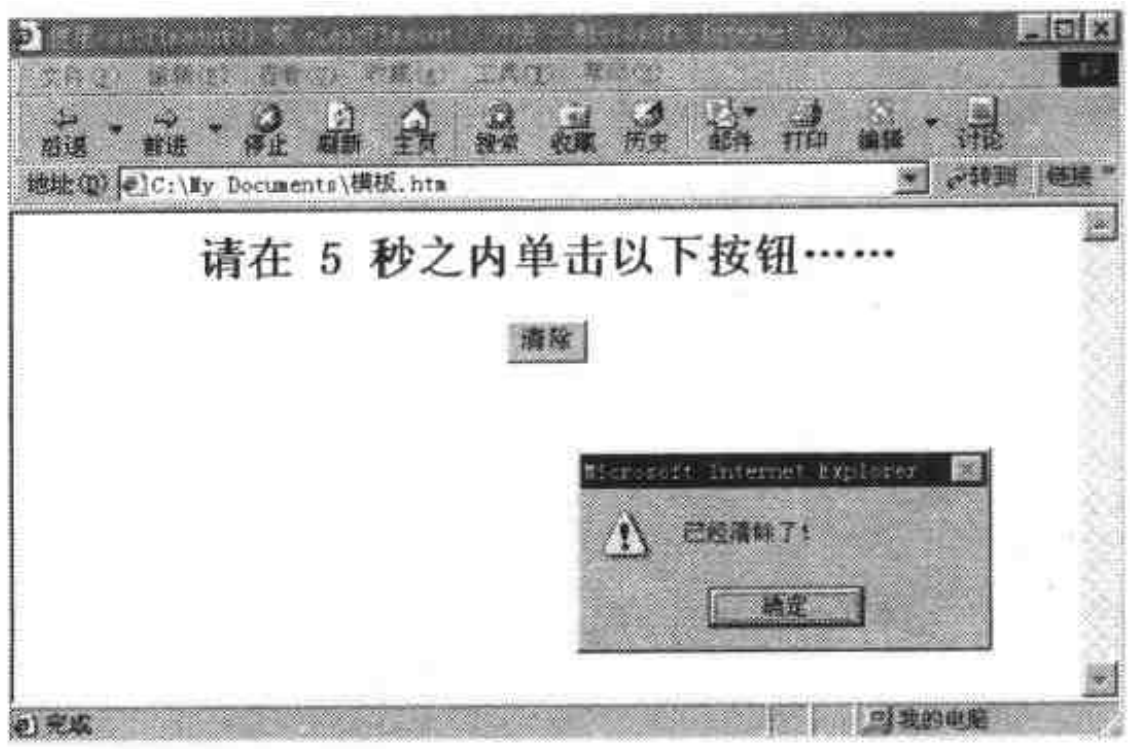


图 5.16　setTimeout()和 clearTimeout()方法示例

图 5.17　“太晚了”提示框

5.2.5　其他窗口操作

window 对象的其他方法用于进行一些常用的窗口操作，例如设置输入焦点、移动窗口、改变窗口大小等。

1．设置输入焦点

使用 window 对象的 blur()方法可以从当前窗口中移出焦点，使用 focus()方法可以使当前窗口获得焦点。注意在结合使用这两个方法时要小心，以免造成焦点不断移进移出的循环。

以下示例显示了 blur()方法和 focus()方法的应用，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 blur() 和 focus() 方法</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function newWin()
{
```

```
newWindow=open('','newWindow','height=300,width=400')
newWindow.document.write("<H3>这是子窗口！</H3>")
newWindow.document.close();
self.focus();  //本语句使焦点返回父窗口!
}

function blurWin()
{
self.blur();  //本语句使父窗口失去焦点。
newWindow.focus();  //本语句使子窗口获得焦点。
}
//-->
</SCRIPT>
<HEAD>
<BODY>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "新窗口" onClick = newWin()>
  <INPUT TYPE = BUTTON Value = "切换焦点" onClick = blurWin()>
</FORM>
</BODY>
</HTML>
```

这段代码的效果为：单击“新窗口按钮”，则新建一个窗口，但焦点仍然保持在原窗口，如图 5.18 所示；当单击“切换焦点”按钮时，焦点切换到子窗口。

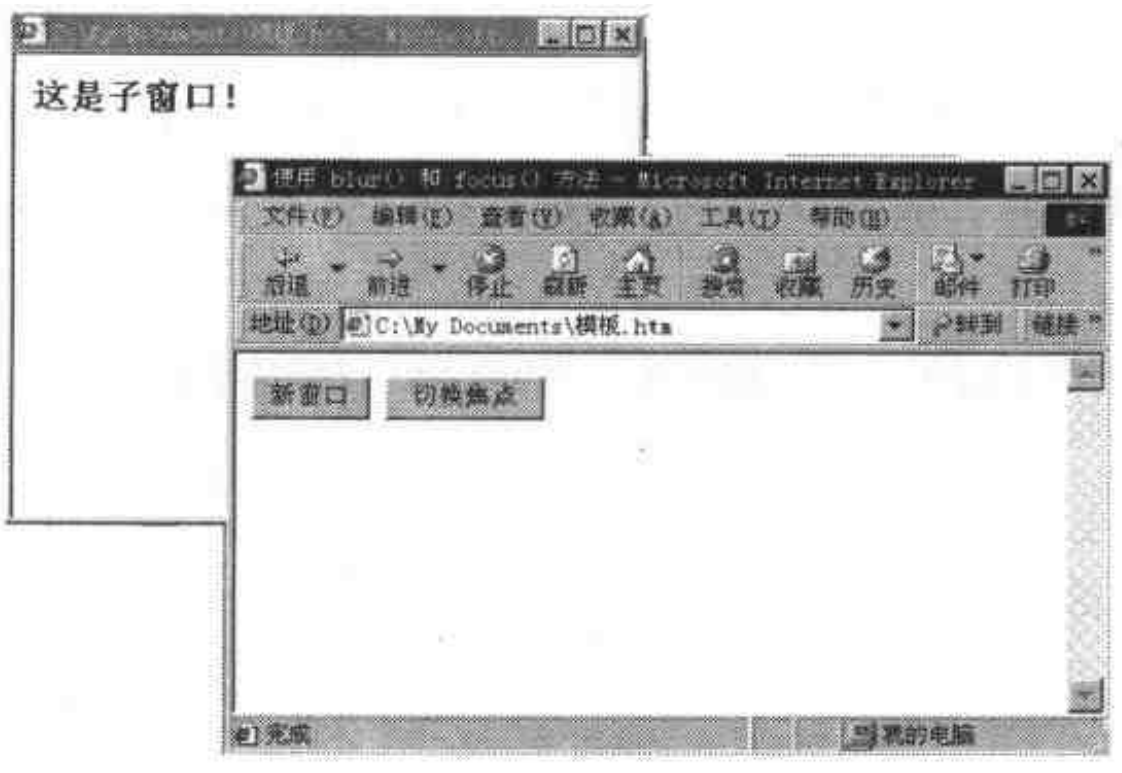


图 5.18　blur()和 focus()方法示例

实际上，对应于焦点移进移出窗口还有两个常用事件 onBlur 和 onFocus，在对这两个事件进行处理时也要注意避免造成焦点移进移出的循环。

以下示例显示了当焦点移进新窗口时，弹出一个欢迎对话框，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用焦点事件</TITLE>
<HEAD>
<BODY>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "新窗口"
         onClick
-open("newWin.htm","newWindow","height=300,width=400")>
</FORM>
</BODY>
</HTML>
```

newWin.htm 文件的代码如下：

```
<HTML>
<HEAD>
<TITLE>新窗口</TITLE>
</HEAD>
<BODY onfocus="alert('欢迎！')">
<H2>单击以下按钮关闭本窗口：</H2>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "关闭" onClick = window.close()>
</FORM>
</BODY>
</HTML>
```

本示例的效果是：当用户单击“新窗口”按钮时，新建一个窗口并在其中显示 newWin.htm 文件，此时新窗口获得焦点，因此触发 onfocus 事件，于是弹出一个欢迎对话框，如图 5.19 所示。

2. 移动窗口

使用 window 对象的 moveBy()和 moveTo()方法可以移动窗口，前者每次移动指定像素，后者移动到指定位置。这两个方法的参数都是两个像素数，moveBy()方法表示横向和纵向移动的像素数，moveTo()方法表示移动到的坐标。在使用像素数时，可以使用负数。

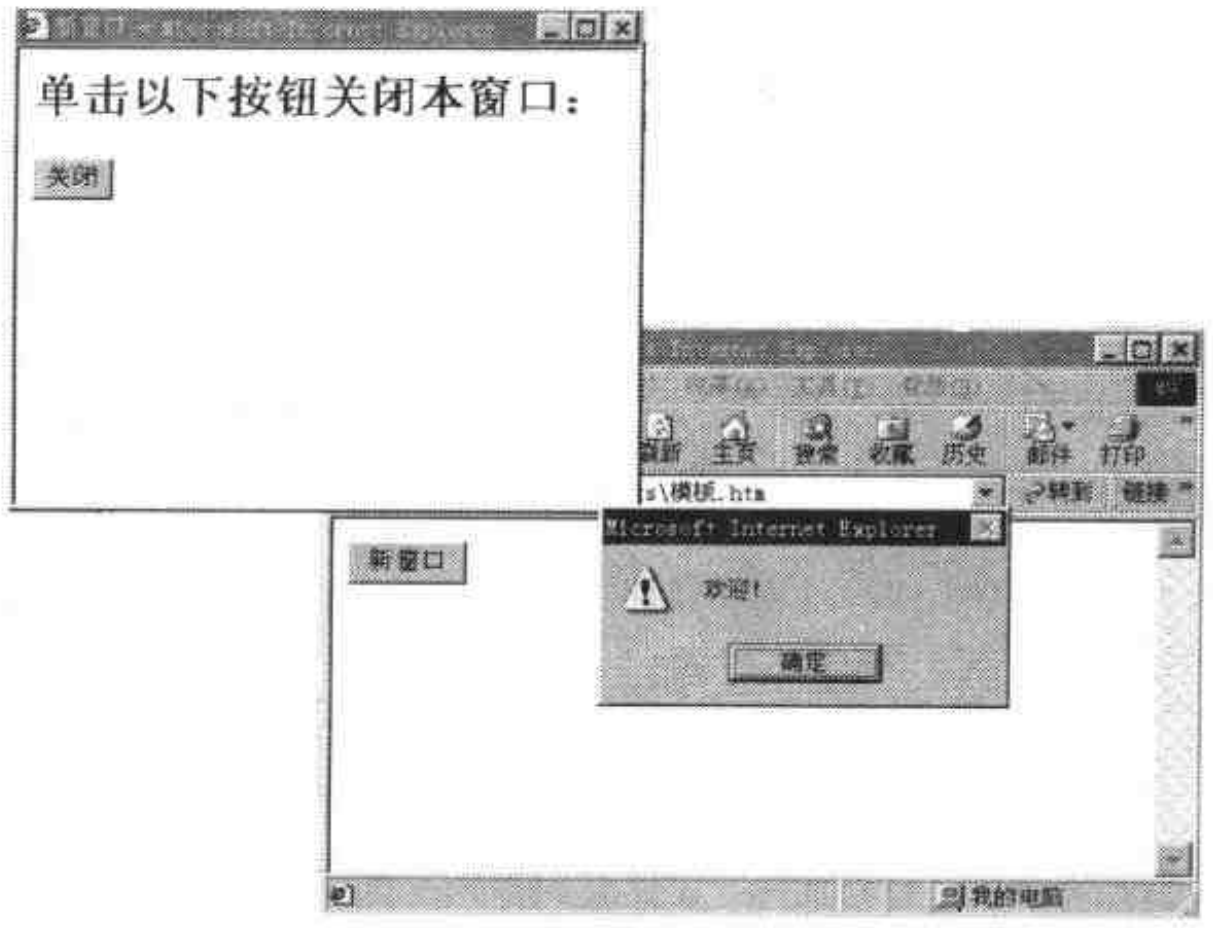


图 5.19　焦点事件示例

以下示例显示了如何用这两个方法移动窗口，代码如下：

```
<HTML>
<HEAD>
  <TITLE>moveBy() 方法和 moveTo() 方法示例</TITLE>
<HEAD>
<BODY>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "向右下移动" onClick =moveBy(10,10)>
  <INPUT TYPE = BUTTON Value = "向左上移动" onClick =moveBy(-10,-10)>
  <P>
  <INPUT TYPE = BUTTON Value = "移动到 (10,10)" onClick =moveTo(10,10)>
  <INPUT TYPE = BUTTON Value = "移动到 (-10,-10)" onClick =moveTo(-
10,-10)>
</FORM>
</BODY>
</HTML>
```

以上代码的显示效果如图 5.20 所示。

当用户单击“向右下移动”按钮时，每次在横向和纵向都移动 10 个像素；当用户单击“向左上移动”按钮时，每次在横向和纵向都反向移动 10 个像素；当用户单击“移动到(10,10)”按钮时，窗口移动到屏幕左上角的(10,10)坐标处；当用户单击“移动到(-10,-10)”

按钮时，窗口移动到屏幕左上角的(-10,-10)坐标处（此时有一部分看不到）。

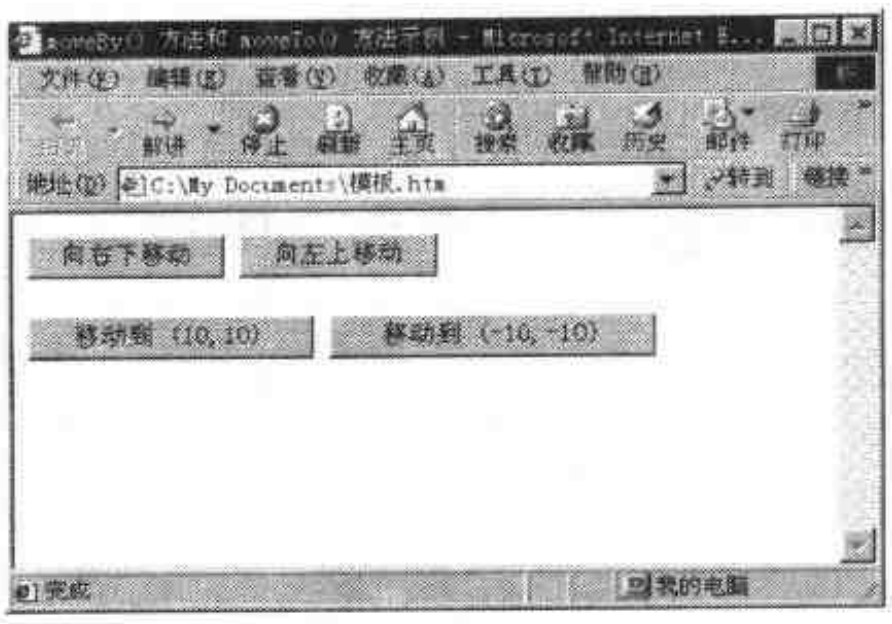


图 5.20 moveBy() 和 moveTo() 方法示例

注意：在 Internet Exploer 中不支持 onMove 事件。

3. 滚动窗口

使用 window 对象的 scroll()、scrollBy()和 scrollTo()方法可以滚动窗口，其中 scroll()方法和 scrollTo()方法的效果相同，都是滚动到指定位置，而 scrollBy()方法则是一次滚动指定像素。同样，这些方法的参数都是两个像素数。

以下示例显示了如何用这三个方法滚动窗口，代码如下：

```
<HTML>
<HEAD>
  <TITLE>滚动窗口方法示例</TITLE>
<HEAD>
<BODY>
<PRE>本行比较长本行比较长本行比较长本行比较长本行比较长本行比较长本行比较长本行比较长本行比较长本行比较长</PRE>
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "向右下滚动" onClick =scrollBy(10,10)>
  <INPUT TYPE = BUTTON Value = "向左上滚动" onClick =scrollBy(-10,-10)>
  <P>
  <INPUT TYPE = BUTTON Value = "滚动到 (10,10)" onClick =scroll(10,10)>
  <INPUT TYPE = BUTTON Value = "滚动到 (-10,-10)" onClick =scroll(-10,-10)>
</FORM>
<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本<BR>示例文本
</BODY>
```

```
</HTML>
```

这段代码的显示效果如图 5.21 所示。

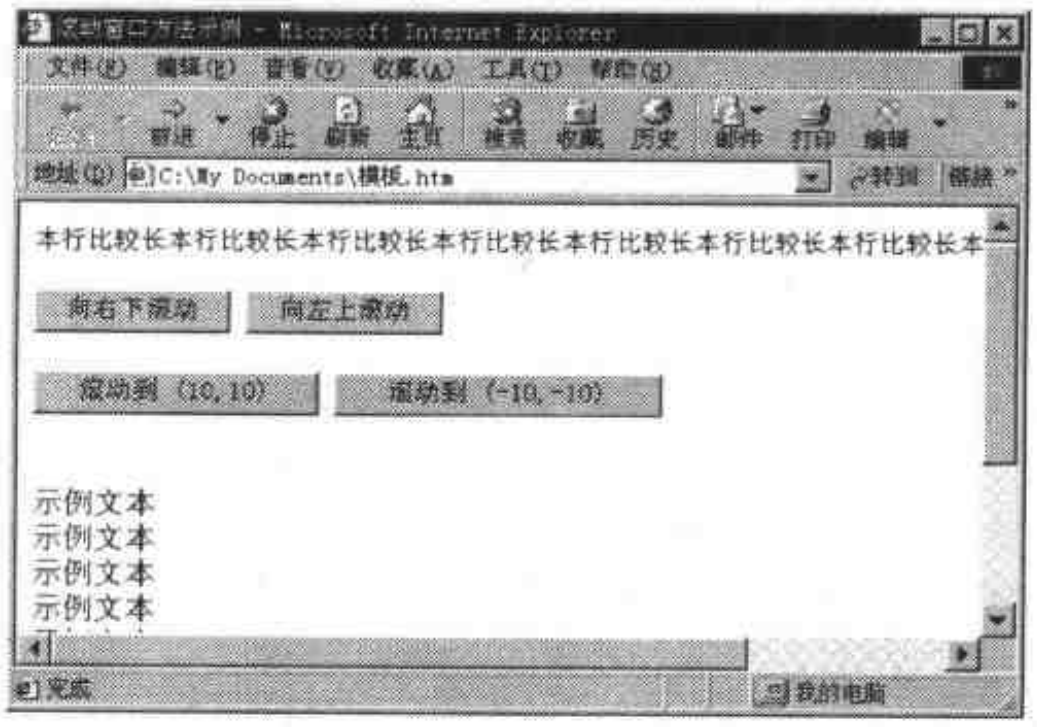


图 5.21　滚动窗口方法示例

当用户单击“向右下滚动”按钮时，滚动条每次在横向和纵向都移动 10 个像素，如图 5.22 所示。

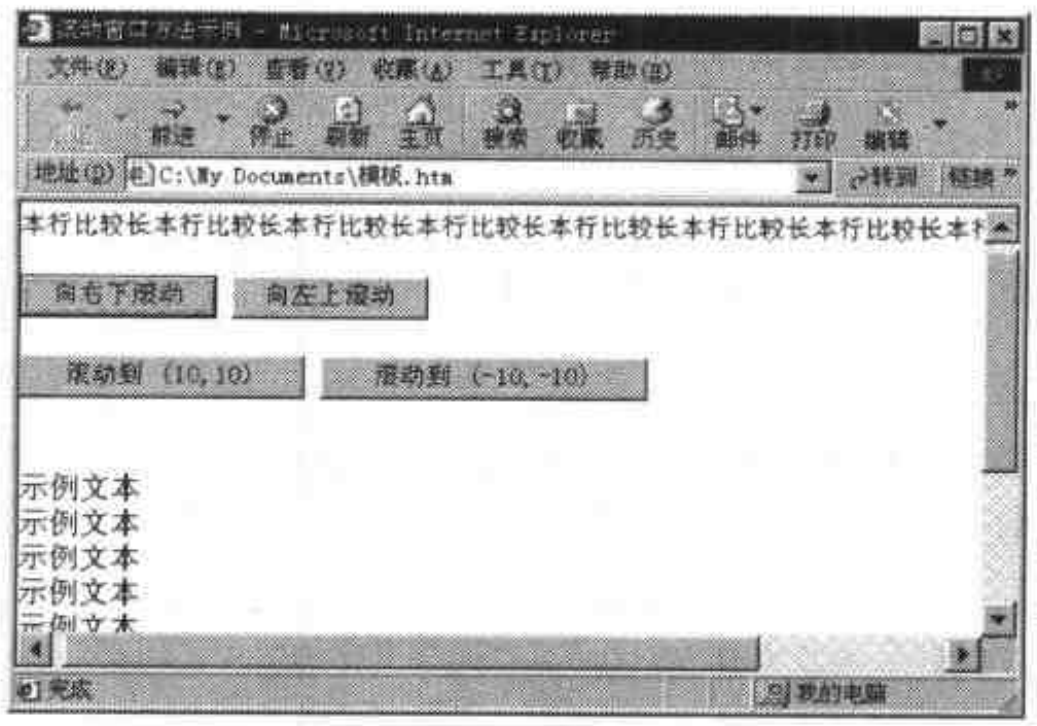


图 5.22　滚动窗口

当用户单击“向左上移动”按钮时，滚动条每次在横向和纵向都反向移动 10 个像素；当用户单击“滚动到 (10,10)”按钮时，相当于第一次单击“向右下滚动”按钮；当用户单击“移动到 (-10,-10)”按钮时，没有任何效果。

横据这个示例可以看出，只有当滚动有意义时，使用这三个方法才能产生特定的效果，否则窗口内容并不按参数指定滚动。例如，如果没有水平滚动条，则 moveBy(10,10) 的效

果与 moveBy(0,10) 一模一样；如果既没有水平滚动条，也没有垂直滚动条，则任何方法都不能使窗口内容滚动。

4. 缩放窗口

使用 window 对象的 resizeBy() 和 resizeTo() 方法可以缩放窗口（也就是更改窗口大小），前者每次缩放指定像素，后者缩放到指定大小。这两个方法的参数都是两个像素数，对于 resizeBy() 方法表示横向和纵向缩放的像素数，对于 resizeTo() 方法表示缩放到的宽高。在使用像素数时，可以使用负数。在更改窗口大小时，浏览器会触发 onResize 事件，可以对该事件进行处理。

以下示例显示了如何缩放窗口以及响应 onResize 事件，代码如下：

```
<HTML>
<HEAD>
  <TITLE>缩放窗口示例</TITLE>
<HEAD>
<BODY onResize="alert('最好别动我！！！')">
<FORM name=form1>
  <INPUT TYPE = BUTTON Value = "缩小" onClick =resizeBy(-10,-10)>
  <INPUT TYPE = BUTTON Value = "放大" onClick =moveBy(10,10)>
  <P>
  <INPUT TYPE = BUTTON Value = "缩小到 400*300" onClick
=resizeTo(400,300)>
</FORM>
</BODY>
</HTML>
```

以上代码的显示效果如图 5.23 所示。

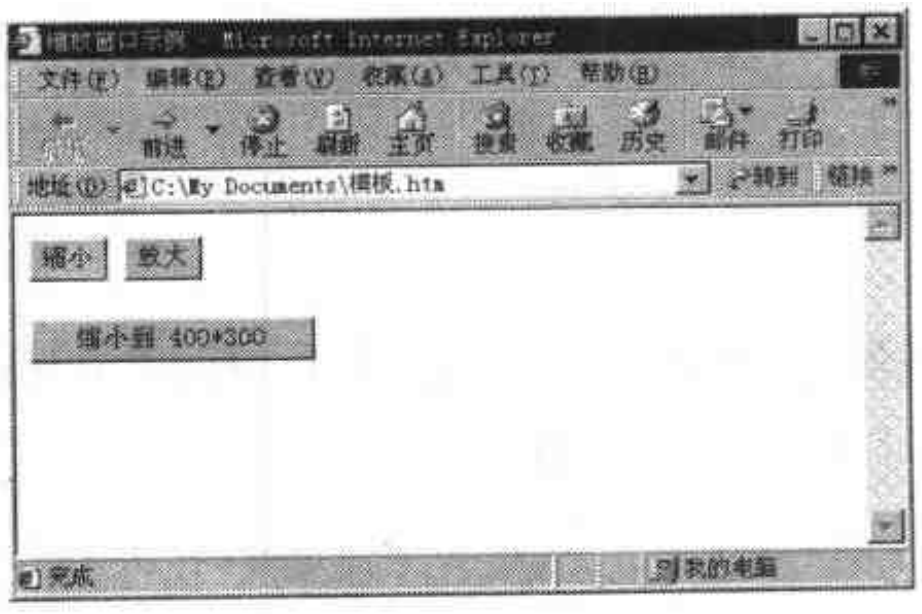


图 5.23 缩放窗口示例

当用户单击“缩小”按钮时，两次弹出“最好别动我”提示框（这是因为横向纵向缩放分别触发了 onresize 事件），然后窗口长和宽都缩小 10 个像素；当用户单击“放大”按钮时，也是两次弹出“最好别动我”提示框，然后窗口长和宽都增大 10 个像素；当用户单击“缩小到 400*300”按钮时，也是两次弹出“最好别动我”提示框，然后窗口缩小到长 400 像素、宽 300 像素，如图 5.24 所示。

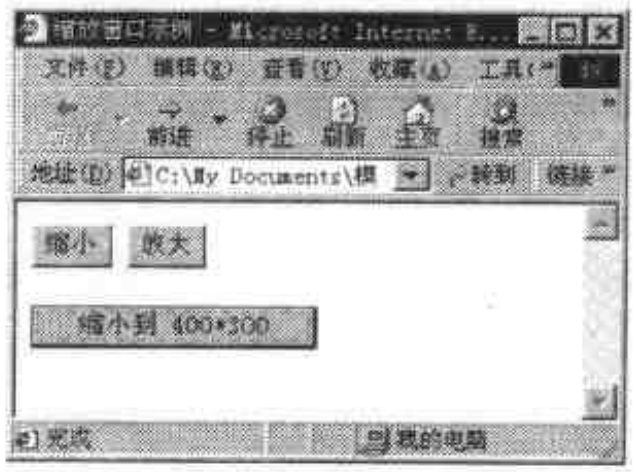


图 5.24　400*300 的窗口

5. 打印窗口内容

使用 window 对象的 print() 方法，可以使用户打开“打印”对话框，以便打印当前网页。调用该方法相当于使用“文件”菜单中的“打印”命令。

以下示例显示了 print()方法的作用，代码如下：

```
<HTML>
<HEAD>
  <TITLE>print() 方法示例</TITLE>
<HEAD>
<BODY>
<FORM name=form1>
请单击此按钮打印：<INPUT TYPE = BUTTON Value = '打印' onClick = print()>
</FORM>
</BODY>
</HTML>
```

这段代码的显示效果如图 5.25 所示，当用户单击“打印”按钮时，将弹出“打印”对话框，用户可以设置打印选项，如图 5.26 所示。

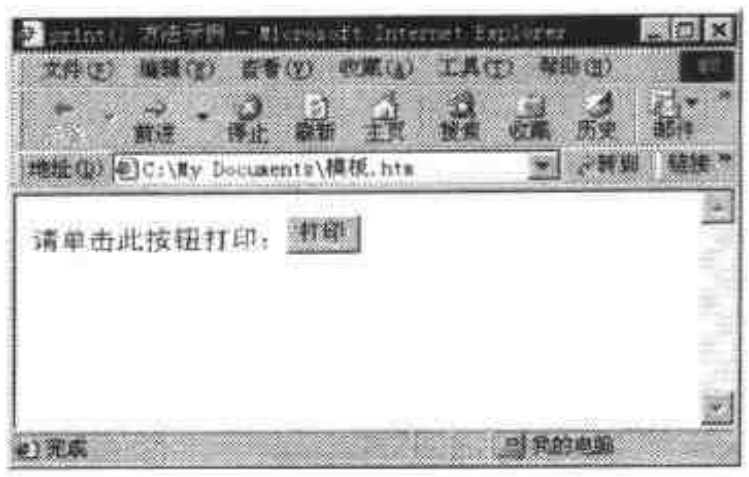


图 5.25　print() 方法示例

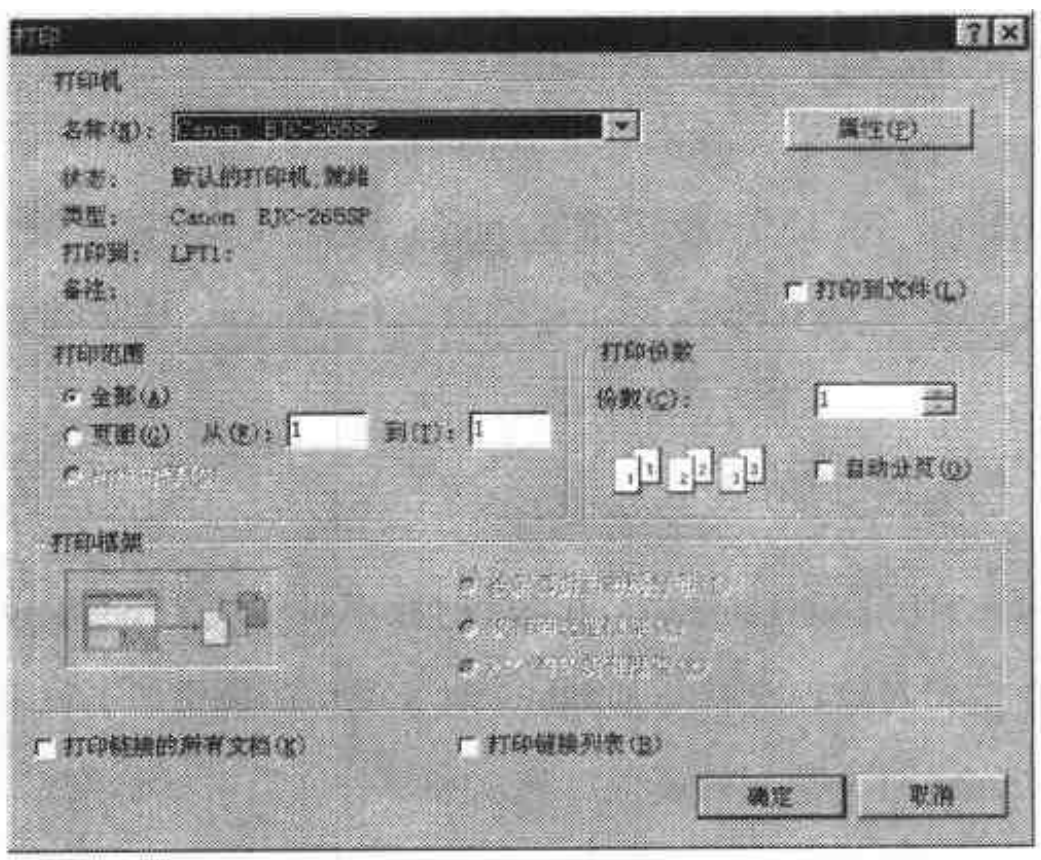


图 5.26　“打印”对话框

5.3　frame 对象

5.3.1　概述

frame（框架）对象用于表示框架文档中的框架，它是在遇到 FRAME 标记符时由浏览器创建的。在一个框架文档中，FRAMESET 标记符代替了 BODY 标记符，在其中包含一个或多个 FRAME 标记符，代表每个具体的框架。

例如，以下文档就是一个典型的框架文档：

```
<HTML>
<HEAD>
  <TITLE>框架文档示例</TITLE>
<HEAD>
```

```
<FRAMESET COLS="25%, *">
  <FRAME name=frame1 src="left.htm">
  <FRAME name-frame2 src="right.html">
<FRAMESET>
</HTML>
```

如果要访问窗口中的框架，可以用以下方法之一：

- window.frames['*framename*']
- window.frames[*index*]
- window.framename

其中，framename 是 FRAME 标记符的 name 属性所指定的框架名称，index 表示框架在整个框架文档中的索引，window 可以是任意一个 window 对象（例如，parent、top 等）。

5.3.2 属性、方法与事件

在 Internet Explorer 和 Navigator 中，frame 对象都被实现为 window 对象，因此每一个 frame 对象都是一个窗口对象，支持与 window 对象类似的属性、方法和事件。

表 5.4 列出了 frame 对象常用的属性、方法和事件（有关的详细信息请参见对 window 对象的说明）。

表5.4 frame对象的属性、方法和事件

项　目	说　明
blur()	从框架中移出焦点
clearInterval()	取消重复执行的操作
clearTimeout()	取消延时执行的操作
document	表示框架中显示的文档
focus()	将焦点移到框架
frames	表示框架中包含的所有子框架的数组
length	表示当前框架中的子框架个数
name	表示框架名称
onBlur	焦点从框架中移出时触发
onFocus	焦点移动到框架时触发
onResize	更改框架尺寸时触发
parent	表示当前框架的上一级框架
print()	显示“打印”对话框，打印当前框架中的文档内容
self	表示当前框架
setInterval()	设置重复执行操作
setTimeout()	设置延时执行操作
top	表示一系列嵌套框架中的最上层框架

5.3.3 示例

本示例显示了如何使用 frame 对象的属性、方法和事件，代码如下：

```
<HTML>
<HEAD>
  <TITLE>frame 对象示例</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function frameSum()
{
newWin=open("","","height=200,width=300");
for(i=0;i<window.frames.length;i++)
  newWin.document.write("框架#"+i+"的名称是"+window.frames[i]
.name+"<P>");
}
-->
</SCRIPT>
<HEAD>
<FRAMESET COLS="25%, *" onload="frameSum()">
  <FRAME name=frame1 src="./left.htm"
         onResize=frames[0].document.bgColor='green'>
  <FRAME name=frame2 src="./right.htm" onResize="window.frame2
.print()" >
<FRAMESET>
</HTML>
```

left.htm 文件的代码如下：

```
<HTML>左框架</HTML>  <!--省略了其他标记-->
```

left.htm 文件的代码如下：

```
<HTML>右框架</HTML>
```

本示例的效果是：当浏览器加载页面时弹出一个新窗口，其中显示了框架名称信息，如图 5.27 所示；将焦点切换回主窗口，如果更改窗口大小或拖动框架边框，则同时触发两个框架的 onResize 事件，左边框架将变为绿色背景，同时弹出“打印”对话框。

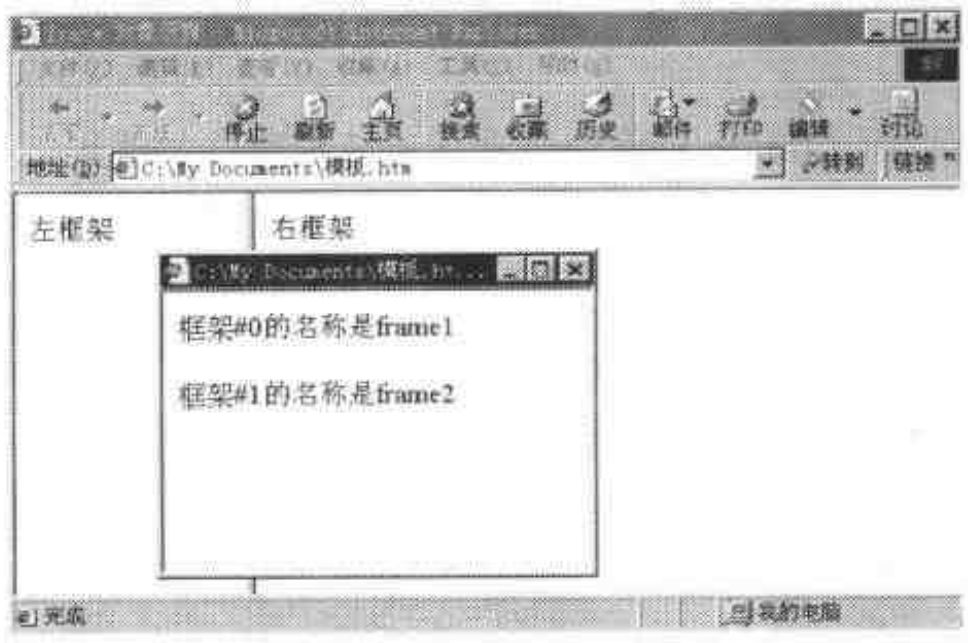


图 5.27　frame 对象示例

5.4　navigator 对象

navigator 对象就是浏览器对象，用于获得与浏览器相关的信息。通常在编写跨浏览器平台的 Web 页时，需要首先判断浏览器的相关信息，然后决定如何编写代码，此时就可以使用 navigator 对象。

navigator 对象支持以下属性和方法，如表 5.5 所示。

表5.5　navigator对象的属性、方法

项　　目	说　　明
appCodeName	表示浏览器的代码名称
appName	表示官方浏览器名称
appVersion	有关浏览器的版本信息
javaEnabled()	判断浏览器是否支持 Java，如果支持，返回 true，否则返回 false
mimeTypes	表示浏览器当前支持的所有 MIME 类型的数组
platform	表示浏览器运行的操作系统平台
plugins	表示浏览器当前安装的所有插件的数组
userAgent	表示浏览器发往服务器的 HTTP 协议中的用户代理头，实际上包含浏览器的代码名称和版本信息

以下示例显示了如何使用 navigator 对象的属性和方法获得有关的浏览器信息，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 navigator 对象</TITLE>
</HEAD>
```

```
<BODY>
<H2 align=center>显示浏览器信息……</H2>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
  document.write("浏览器代码名称："+window.navigator.appCodeName+"<BR>")
//访问 navigator 对象时可以用 window.navigator，也可以直接用 navigator。
  document.write("浏览器名称："+navigator.appName+"<BR>")
  document.write("浏览器版本号："+navigator.appVersion+"<BR>")
  document.write("是否支持 Java："+navigator.javaEnabled()+"<BR>")
  document.write("MIME 类型数："+navigator.mimeTypes.length+"<BR>")
  document.write("操作系统平台："+navigator.platform+"<BR>")
  document.write("插件数："+navigator.plugins.length+"<BR>")
  document.write("用户代理："+navigator.userAgent+"<BR>")
-->
</SCRIPT>
</BODY>
</HTML>
```

该示例的显示效果如图 5.28 所示。

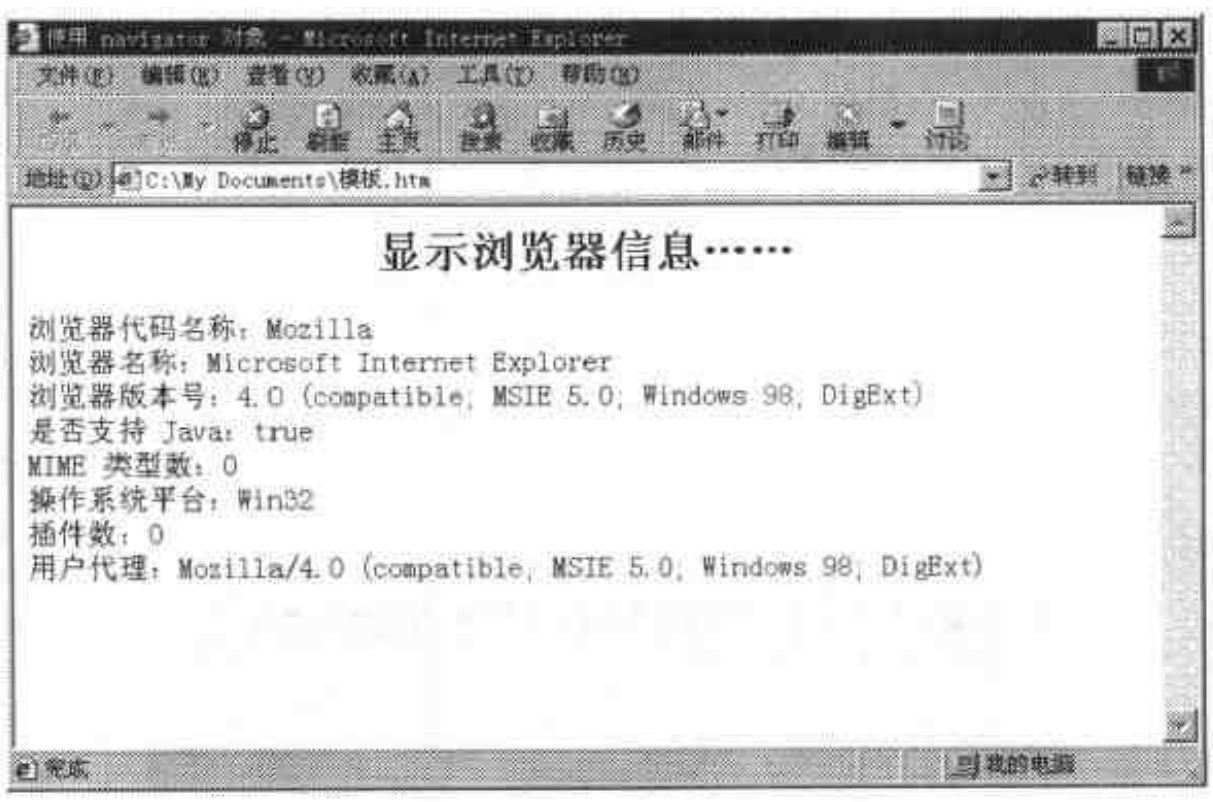


图 5.28 使用 navigator 对象

5.5 screen 对象

screen 对象表示用户屏幕，它提供了屏幕尺寸和颜色深度等信息。screen 对象支持的属性如表 5.6 所示。

表5.6 screen对象的属性

属　性	说　明
AvailHeight	表示屏幕上可用的像素高度，在 Windows 系统中指不包括任务栏的屏幕高度
AvailWidth	表示屏幕上可用的像素宽度，在 Windows 系统中指不包括任务栏的屏幕宽度。（如果任务栏水平放置，则 availHeight = height - 任务栏高度，availWidth = width；如果任务栏垂直放置，则 availWidth = weight - 任务栏宽度，availHeight = height）
colorDepth	表示屏幕的颜色深度，即用户在“显示 属性”对话框“设置”选项卡中设置的颜色位数
height	表示屏幕的像素宽度（与显示设置有关）
width	表示屏幕的像素宽度（与显示设置有关）

以下示例显示了如何用 screen 对象的属性获得有关屏幕的信息，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 screen 对象</TITLE>
</HEAD>
<BODY>
<H2 align=center>显示屏幕信息……</H2>
<SCRIPT LANGUAGE = "JavaScript" TYPE='text/javascript">
<!--
  document.write("屏幕允许高度：'+window.screen.availHeight+"<BR>")
//访问 screen 对象时可以用 window.screen，也可以直接用 screen。
  document.write("屏幕允许宽度：'+screen.availWidth+"<BR>")
  document.write("颜色深度："+screen.colorDepth+"<BR>")
  document.write("屏幕高度：'+screen.height+"<BR>")
  document.write("屏幕宽度："+screen.width+"<BR>")
-->
</SCRIPT>
</BODY>
</HTML>
```

该示例的显示效果如图 5.29 所示。

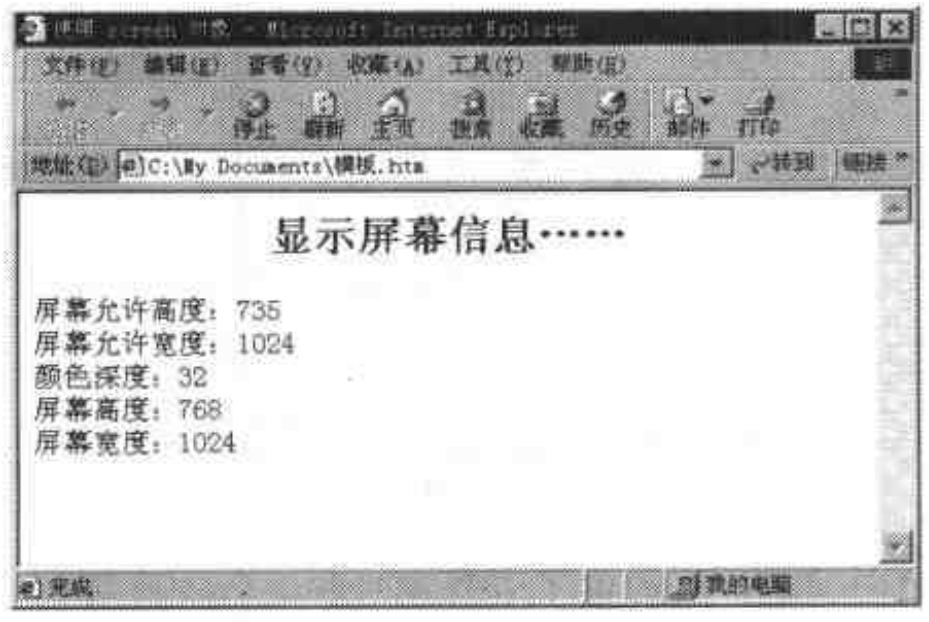


图 5.29　使用 screen 对象

第6章 表单对象

Form（表单）对象是浏览者与网页进行交互的重要工具，通过使用表单中的各种控件对象（按钮、单选框、列表框等），可以实现各种实用功能（例如，通过单击列表框中的选项实现跳转等）。本章介绍 form 对象以及各种表单控件对象，同时介绍了如何使用隐藏字段和 cookie 实现信息的保存和传递。

本章主要内容包括：

- form 对象
- 文本型表单控件
- 单选框与复选框
- 按钮对象
- 选项菜单
- 隐藏字段与 cookie

6.1 Form 对象

表单是网页中实现信息交互的一种重要手段，经常用于收集用户信息或实现在线调查等需要用户输入的情形。如果要在网页中插入表单，应使用 FORM 标记符，然后将各种表单控件添加到该<FORM>和</FORM>之间。本节首先介绍 form 对象，也就是包含其他表单控件的容器对象，接下来的几节将分别介绍各种表单控件对象。

6.1.1 属性、方法与事件

当用户在网页中添加了 FORM 标记符后，既创建了一个 form 对象。用户可以通过两种方式访问网页中的 form 对象：

- 使用 document.forms[]数组，该数组代表文档中的所有表单。使用该数组时，既可以用索引号，也可以用名称。例如，如果文档中的第一个 FORM 标记符的 name 属性为 form1，则可以通过 document.forms[0]或 document.forms['form1']来访问该 form 对象。
- 直接使用表单名称。例如，对于刚才的那个表单对象，可以通过 document.form1 来访问。

获得了 form 对象之后，即可以通过使用其属性、事件和方法来实现各种功能。

1. 属性、方法与事件列表

表 6.1 列出了 form 对象的各种属性、方法与事件以及相应的说明。

表6.1 form对象的属性、方法和事件

项　目	说　明
action	表示表单提交时执行的动作，相当于 form 标记符 的 action 属性，它通常是一个服务器端脚本程序的 URL
elements	代表表单中所有控件元素的数组，表单元素在该数组中的序号就是它在 HTML 源文件中的序号
encoding	表示表单数据的编码类型，相当于 form 标记符的 enctype 属性
length	表示表单中元素的数目
method	表示发送表单的 HTTP 方法，相当于 form 标记符的 method 属性，取值为 get 或 post
name	表示表单名称，相当于 form 标记符的 name 属性
onReset	当用户单击“重置”按钮时触发，执行相应的脚本代码
onSubmit	当用户单击“提交”按钮时触发，执行相应的脚本代码
reset()	将所有表单元素的值重新设置为它们的默认值，相当于单击表单中的“重置”按钮。
submit()	提交表单，相当于单击表单中的“提交”按钮
target	表示用来显示表单结果的目标窗口或框架，相当于 form 标记符的 target 属性，其取值可以是：_blank、_parent、_self 和 _top

2. 访问表单元素

单有 form 对象并不能构成一个有用的表单，要使表单能够起到传递信息的作用，其中必须包含各种表单元素，例如，文本框（text）、单选框（Radio）、复选框（checkbox）、按钮（button）等。

由于这些表单元素都是包含在表单中，因此是 form 对象的子对象，可以直接用名称进行访问。例如，如果一个名称为 form1 的表单中包含一个名称为 text1 的文本框，则可以用 document.form1.text1 来访问该文本框。

除了这种方式以外，还可以利用 form 对象的 elements 属性来访问表单元素。elements 属

性是一个代表表单中所有元素的数组，数组元素值的顺序与在表单中出现的顺序相同。例如，如果名称为 form1 的表单中先后包含一个名为 text1 的文本框和一个名为 button 的按钮，则可以通过 document.form1.elements[0]和 document.form1.elements[1]来访问这两个元素（当然也可以通过在数组运算符中使用元素名称来访问，如 document.form1.elements['text1']，但这样显然没有必要）。

以下示例通过 elements 数组显示出表单中所有元素的名称，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 elements 数组</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript >
<!--
function show()
{
newWin=window.open("","","height=300,width=350")
newWin.document.write("文档中共包含"+document.form1.length+"个元素，分别是: <P>");
for(i=0;i<document.form1.length;i++)
{
newWin.document.write("<UL>")
newWin.document.write("<LI>"+document.form1.elements[i].name+'</LI>'
);
newWin.document.write("</UL>")
}
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM name="form1">
  文本框 1: <INPUT name="text1"><P>
  文本框 2: <INPUT name='text2'><P>
<H3 align=center>单击以下按钮显示表单中的元素信息: </H3>
  按钮 1:<INPUT type=button name="button1" value="按钮" onClick="show()">
</FORM>
</BODY>
</HTML>
```

该示例的效果为：当单击“按钮”按钮时将在一个新窗口中显示出当前文档中表单的

所有元素，如图 6.1 所示。

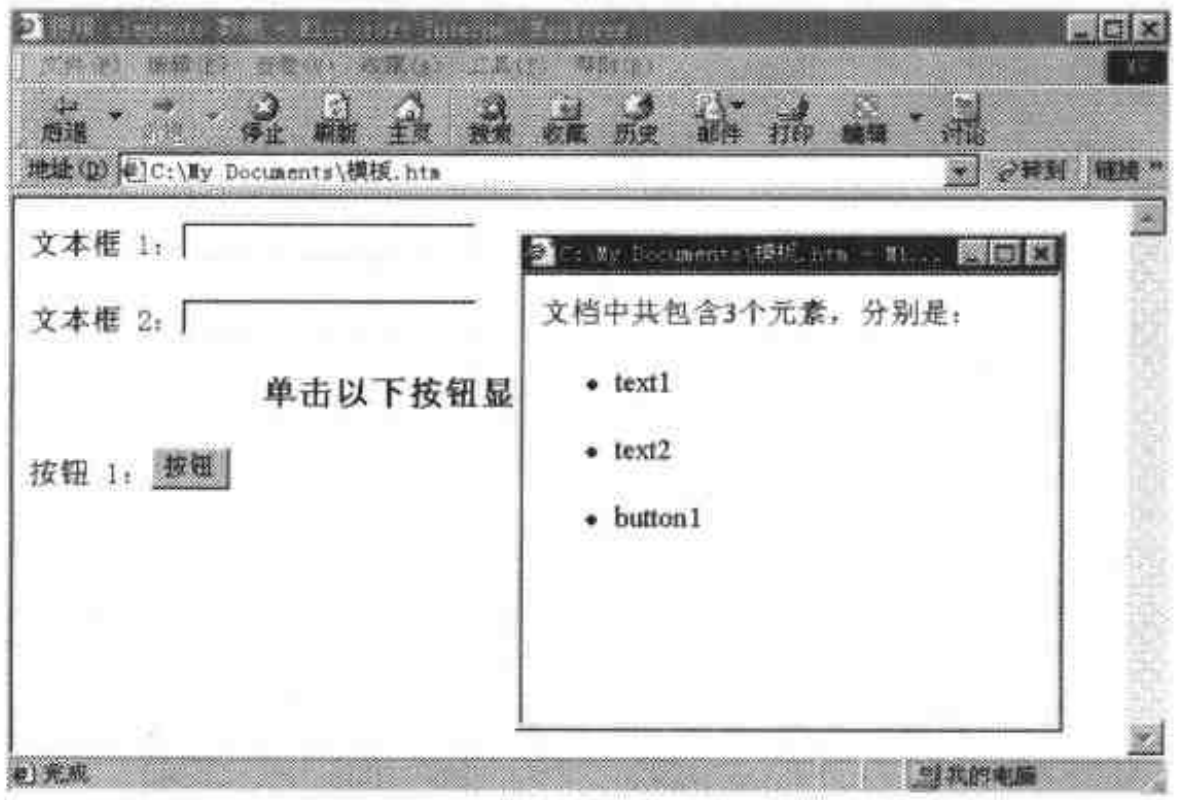


图 6.1　使用 elements 数组

3. 使用其他属性

以下示例使用 form 对象的各种属性显示了相应的表单信息，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 form 对象的属性</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
function show()
{
newWin=window.open("","","height=300,width=450")
newWin.document.write("文档中的表单信息如下：<P>");
newWin.document.write("<UL>")
newWin.document.write("<LI>action="+document.form1.action+"</LI>");
newWin.document.write("<LI>encoding="+document.form1.encoding+"</LI>
");
newWin.document.write("<LI>length="+document.form1.length+"</LI>");
newWin.document.write("<LI>method="+document.form1.method+"</LI>");
newWin.document.write("<LI>name="+document.form1.name+"</LI>");
newWin.document.write("<LI>target="+document.form1.target+"</LI>");
newWin.document.write("</UL>")
return false;  //使 onSumit 的值为 false，则不执行 action 指定的动作。
```

```
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM action="http://www.test.com/bin/process.asp"
      method=POST
      name="form1"
      target="_top"
      onReset="alert('您单击了重置按钮')"
      onSubmit="return show()">
  <H3 align=center>单击提交按钮显示有关的表单信息：</H3>
  姓名：<INPUT><BR>
  年龄：<INPUT><P>
  <INPUT type=submit name="button2" value="提交">
  <INPUT type=reset name="button3" value="重置">
</FORM>
</BODY>
</HTML>
```

该示例的效果为：如果用户单击“重置”按钮，则弹出一个提示框，之后将重置用户在表单中填写的数据；如果用户单击“提交”按钮，则弹出一个新窗口，显示与表单有关的各种信息（FORM 标记符的 encoding 属性没有设置，因此返回默认值），如图 6.2 所示。另外，由于 show() 函数返回 false，因此表单不再提交到 action 指定的 URL。

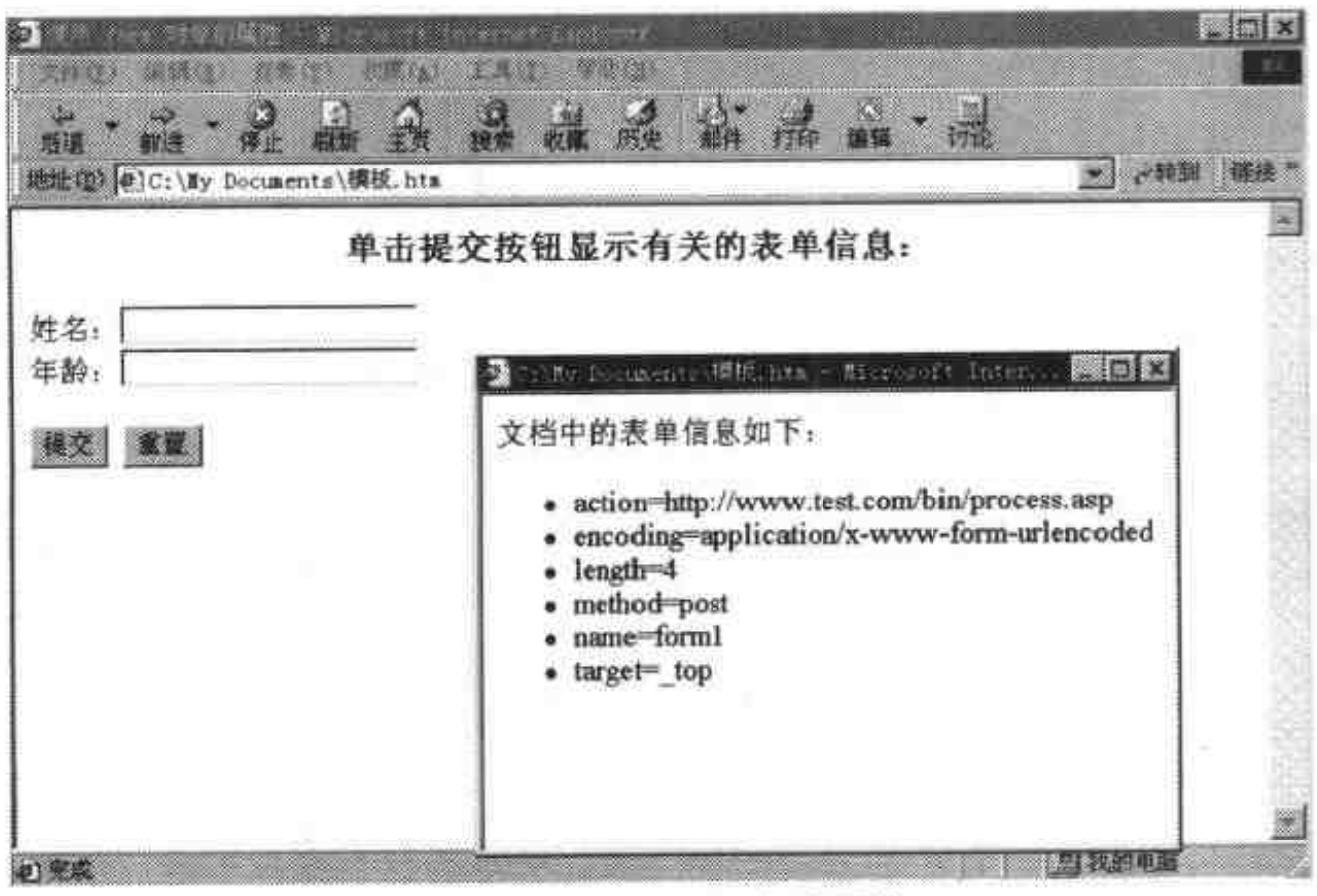


图 6.2 使用 form 对象的属性

6.1.2 表单处理

当用户单击了表单中的提交按钮之后，用户在表单中填写或选择的内容将被传送到服务器端特定的程序（由 action 属性指定，通常是 CGI 程序或 ASP 程序等）中，由该程序进行具体的处理。对表单数据的处理通常包括三个部分：数据解码（即将浏览器传递来的信息以能够使用的方式进行解释），数据操作（对解释出的数据进行操作，例如根据特定关键字查询数据库等）和返回信息（通常是将数据操作的结果以网页的形式返回给浏览器）。由于本书不涉及服务器端编程，因此请读者参阅其他书籍以了解如何在服务器端进行表单处理。

虽然最终的表单处理通常都需要服务器端的支持，但还是有一些工作可以由客户端脚本来做——最典型的一项工作就是表单验证。表单验证是指确定用户提交的表单数据是否合法。例如是否所有必须填的字段都填写了，某些字段是否按照要求（例如电子邮件地址中应包含@字符）填写了等等。如果表单验证由服务器端程序来做，则会带来相应的网络开销和服务器开销，因为每次提交都需要与服务器端程序交互。如果能将表单验证工作放到客户端来做，由于相应的脚本已经随着网页下载到了客户端，因此可以大大减少各种开销，使得只有合法的表单数据才被提交到服务器。

以下示例显示了如何进行客户端表单验证，代码如下：

```
<HTML>
<HEAD>
<TITLE>客户端表单验证</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function validator()
{
 if(document.form1.name.value=="")
{
 alert("您还没有输入姓名呢!");
 return false;
}

 if(document.form1.eMail.value=="")
{
 alert("您还没有输入电子邮件呢!");
```

```
 return false;
}

if(document.form1.idNumber.value=="")
{
 alert("您还没有输入身份证号呢!");
 return false;
}
else  //判断身份证号是否合法。
{
  var checkOK = "0123456789";
  var checkStr = document.form1.idNumber.value;
  var allValid = true;

  if(checkStr.length!=15)  //如果身份证号少于 15 位，则非法。
     allValid=false;

  for (i = 0;  i < checkStr.length;  i++)
  {  //判断身份证号是否全部由数字组成
ch = checkStr.charAt(i);
//String 对象的 charAt 方法用于获得指定位置的字符。
    for (j = 0;  j < checkOK.length;  j++)
      if (ch == checkOK.charAt(j))
        break;
    if (j == checkOK.length)
    {
      allValid = false;
      break;
    }
  }
  if (!allValid)  //如果身份证号非法，则显示为红色。
  {
    document.all.form1.idNumber.style.color="red";
    return false;
  }
  else  //如果再次输入正确，则恢复原来的颜色。
  {
    document.all.form1.idNumber.style.color="black";
  }
```

```
}
}
//  >
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<FORM action="http://www.test.com/process.cgi"
      method=POST
      NAME=form1
      onSubmit="return validator()">
<H2>个人信息</H2>
<TABLE>
  <TR>
    <TD>请输入您的姓名（必须）：
    <TD><INPUT NAME=name>
  <TR>
    <TD>请输入您的身份证号（必须）：
    <TD><INPUT NAME=idNumber>
  <TR>
    <TD>请输入您的电子邮件地址（必须）：
    <TD><INPUT NAME=eMail>
</TABLE>
<BR>
<INPUT TYPE=submit VALUE="提交">
<INPUT TYPE=reset>
</FORM>
</DIV>
</BODY>
</HTML>
```

由于在表单正式提交到服务器之前，需要 onSubmit 的值为 true（如果不设置事件处理函数，则该值默认为 true），因此可以通过为 onSubmit 事件指定处理函数来进行表单数据的验证。在本示例中，对于姓名和电子邮件地址，仅判断是否填写内容，如果没有填写则弹出对话框提示；对于身份证号，除了验证是否填写内容以外，还提供更进一步的验证——如果填写的数据不是由数字组成，或者数字少于 15 位，则将非法的身份证号用红色显示。当用户单击提交“按钮”后，即执行表单验证函数 validator()，如果有不合要求的数

据，则无法提交表单，如图 6.3 所示；直到所有的数据都合法，才能将表单数据提交到 action 属性指定的 URL。

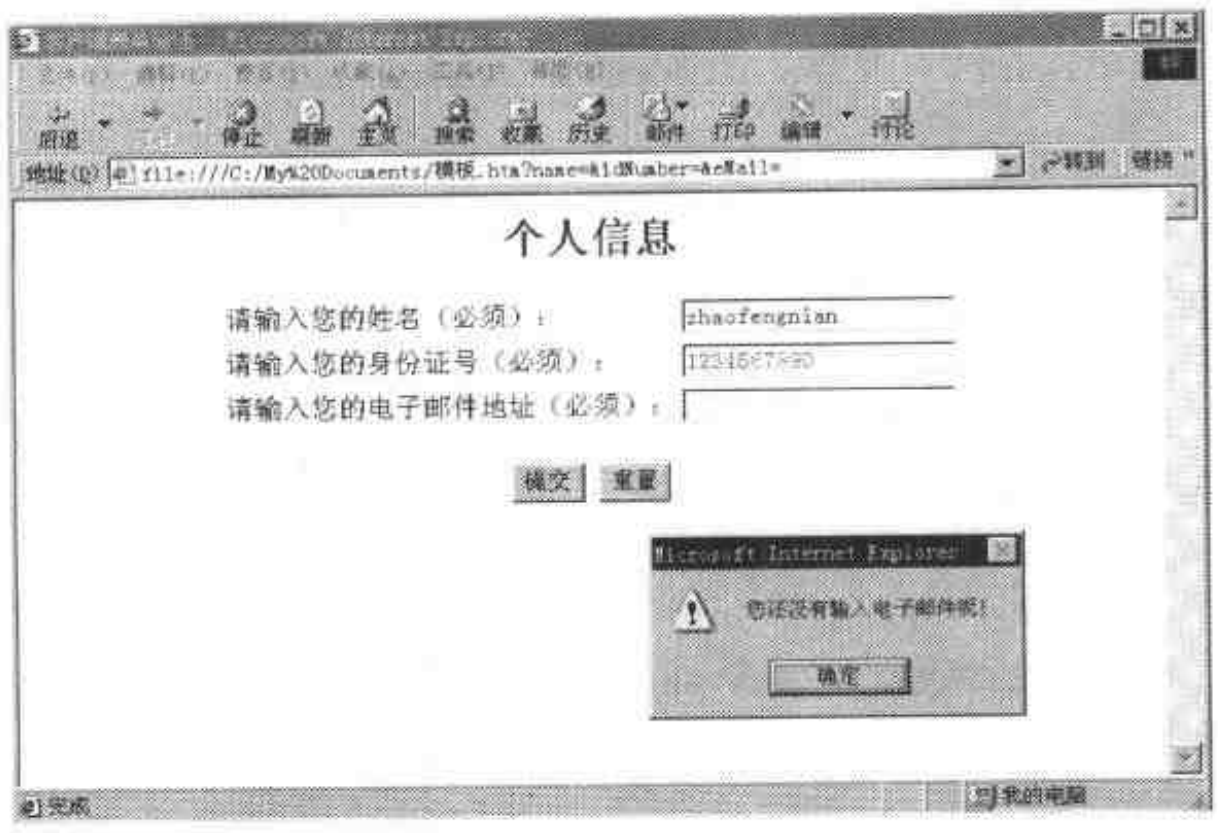


图 6.3　客户端表单验证

6.2　文本型表单控件

文本型表单控件包括：单行文本框对象 text、口令框对象 password、多行文本框对象 textarea 以及文件选择框 fileUpload。这些对象都可用于获取用户的键盘输入，因此具有相似的用法。

6.2.1　概述

文本框对象（text）与口令框对象（password）基本相同，唯一的不同之处在于：文本框中的内容以明文显示，而口令框中的内容以暗文（即 * 号）显示。当用户在网页的表单中使用 INPUT 标记符，并且 TYPE 属性为 text 或不指定该属性时，即创建出了文本框对象；当用户在网页的表单中使用 INPUT 标记符，并且 TYPE 属性为 password 时，即创建出了口令框对象。

多行文本框对象（textarea）与文本框对象类似，不同之处在于它可以输入多行文本。当用户在网页的表单中使用 TEXTAREA 标记符时，即创建出了多行文本框对象。

文件选择框（fileUpload）用于让用户选择或输入需要上传的文件，通常显示一个文本

框和一个“浏览”按钮。当用户在网页的表单中使用 INPUT 标记符，并且 TYPE 属性为 file 时，即创建出了文件选择框对象。

以下示例显示了这几种文本型表单控件的外观，代码如下：

```
<HTML>
<HEAD><TITLE>文本型表单控件</TITLE></HEAD>
<BODY>
<DIV align=center>
<FORM NAME=form1>
<H2>文本型表单控件</H2>
文本框：<INPUT value="text"><P>
口令框：<INPUT TYPE=password value="password"><P>
多行文本框：<TEXTAREA></TEXTAREA><P>
文件选择框：<INPUT TYPE=file><P><BR>
<INPUT TYPE=submit VALUE="提交">
<INPUT TYPE=reset>
</FORM>
</DIV>
</BODY>
</HTML>
```

这段代码的效果如图 6.4 所示。

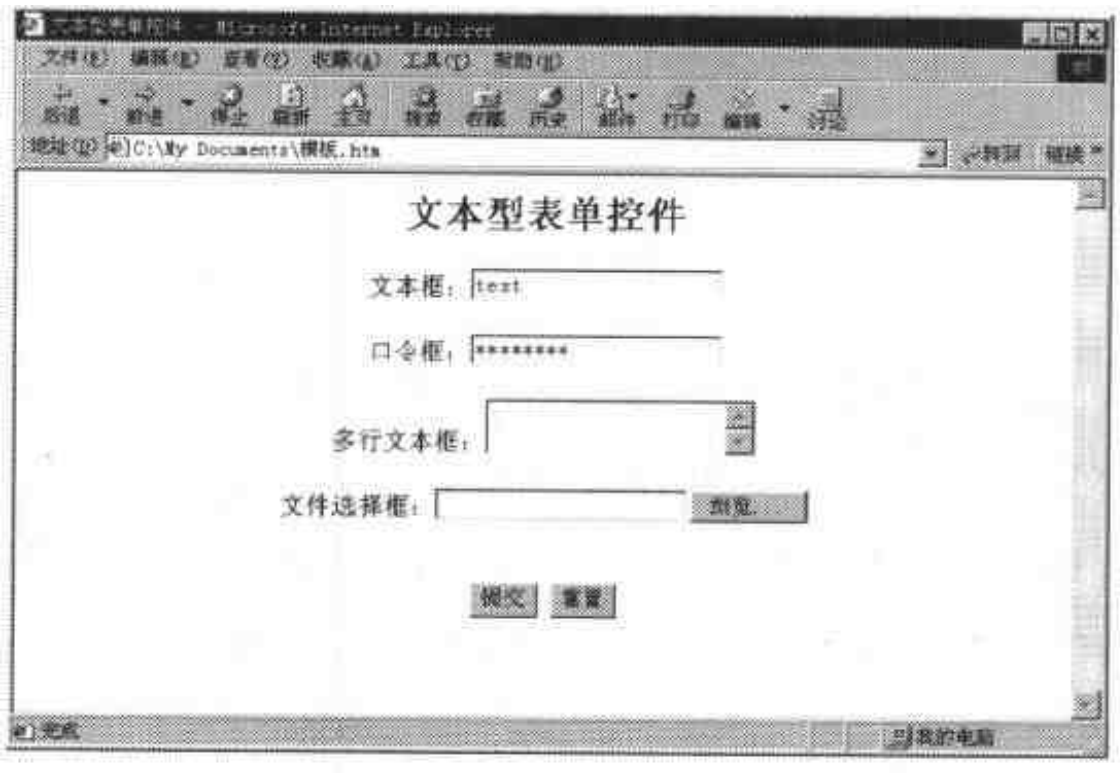


图 6.4　文本型表单控件

对于文件选择框，如果单击“浏览”按钮，则弹出一个标准的“选择文件”对话框，如图 6.5 所示，用户可以从中选择需要上传的文件。选择之后，所选文件的路径将显示在

文件选择框的文本框中，作为文件选择框的值。

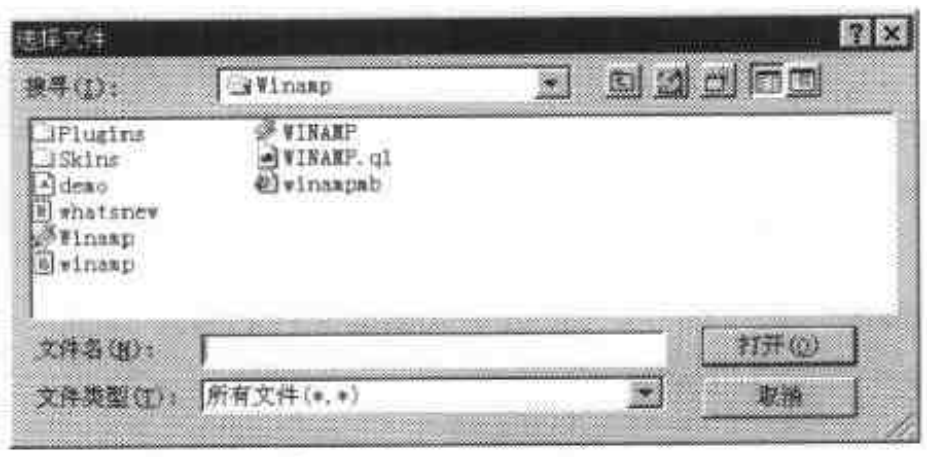


图 6.5 “选择文件”对话框

6.2.2 属性、方法和事件

这些文本型表单控件具有类似的属性、方法和事件，如表 6.2 所示。

表6.2 文本型对象的属性、方法和事件

项 目	说 明
blur()	将焦点从对象上移走
defaultValue	表示对象的默认 value 属性。注意：fileUpload 对象没有此属性
focus()	将焦点移动到对象上
form	表示对象所在的表单
name	表示对象的名称
onBlur	当从对象上移开焦点时触发
onChange	当对象的值发生变化时触发
onFocus	当对象获得焦点时触发
select()	选中对象中的文本
type	表示对象的类型。对于文本框，该值为 text；对于口令框，该值为 password；对于多行文本框，该值为 textarea；对于文件选择框，该值为 file
value	表示对象的值。对于文本框、口令框、多行文本框，该值就是在框中输入的文字；对于文件选择框，该值是用户在“选择文件”对话框中选择的文件，或在文本框中输入的内容

6.2.3 示例

本示例综合应用了文本型控件对象的属性、方法和事件，代码如下：

```
<HTML>
<HEAD>
<TITLE>文本型表单控件</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
```

```
function show()
{
myForm=document.form1.elements[0].form;
  //以上语句使用对象的 form 属性引用文档中的表单。
newWin=open("","","height=400,width=400");
newWin.document.write("<H3>表单元素信息：</H3>")
for(i=0;i<myForm.length;i++)
{
if(myForm.elements[i].type=="text")
  myType="文本框";
else if(myForm.elements[i].type=="password")
  myType="口令框";
else if(myForm.elements[i].type=="textarea")
  myType="多行文本框";
else if(myForm.elements[i].type=="file")
  myType="文件选择框";
else
  myType="其他元素";

newWin.document.write(myType+"中的值为："+myForm.elements
[i].value+"<P>")
}
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<FORM NAME=form1 onSubmit="show()">
<H2>文本型表单控件</H2>
文本框：<INPUT value="text" onfocus=this.select()> <!-- this 代表当前对
象，在此语句中相当于 document.form1.elements[0]。-->
<P>
口令框：<INPUT TYPE=password onfocus=this.select() value="password">
<P>
多行文本框：<TEXTAREA ROWS=4 onfocus=this.select()>多行文本框</TEXTAREA>
<P>
文件选择框：<INPUT TYPE=file NAME=file1 onfocus=this.select()>
<P><BR>
```

```
<INPUT TYPE=submit VALUE="提交">
<INPUT TYPE=reset>
</FORM>
</DIV>
</BODY>
</HTML>
```

该示例的效果为：单击任意一个表单元素，使其获得焦点，则可以选中当前对象中的所有文字（这是因为使用了语句 onfocus=this.focus();），如图 6.6 所示；填写完表单内容之后，单击“提交”按钮，则可以在一个新窗口中显示当前表单中各元素的值，如图 6.7 所示。

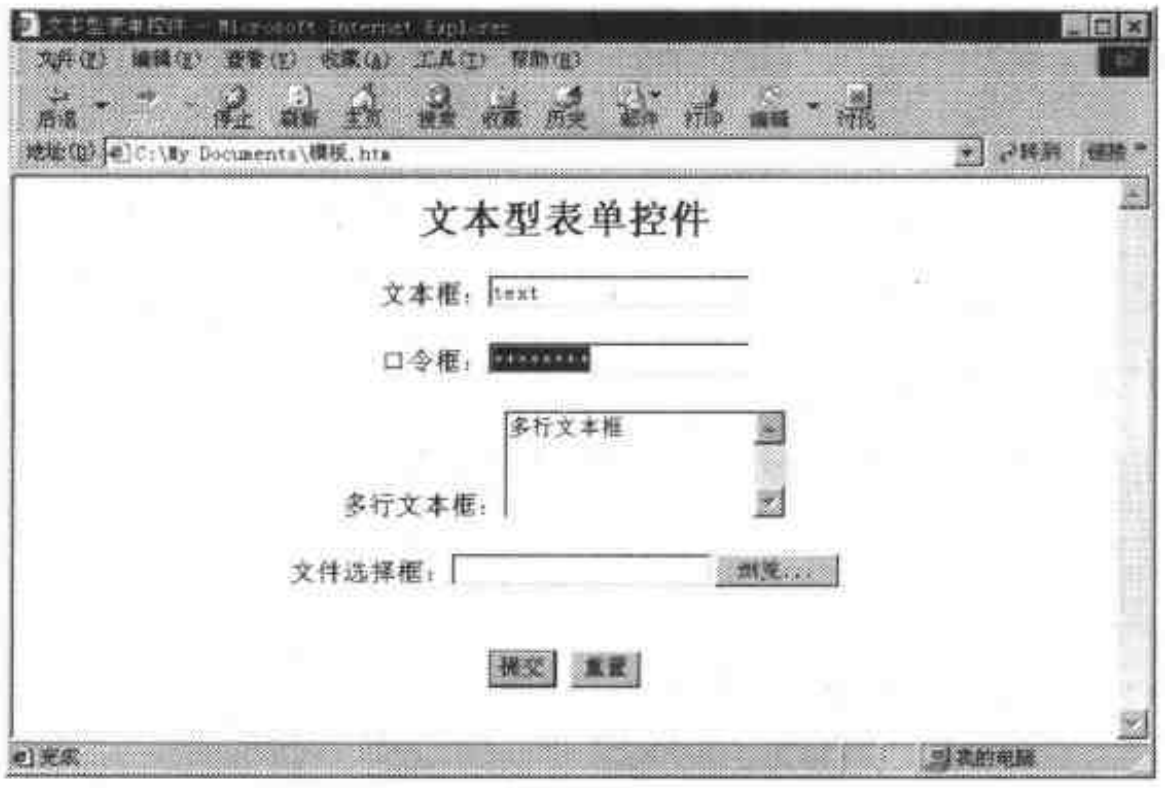


图 6.6 使用文本型表单控件

图 6.7 显示控件的值

6.3 单选框与复选框

6.3.1 概述

单选框（radio）和复选框（checkbox）都是让用户在多个选项中进行选择的控件，不同的是：在一组单选框中，只允许选中一个选项，而在一组复选框中则可以同时选中多个选项。

当用户在网页的表单中使用 INPUT 标记符，并且 TYPE 属性为 radio 时，即创建出了单选框对象；同样，当使用了 TYPE 属性为 checkbox 的 INPUT 标记符时，即创建出了复选框对象。

需要强调的一点是：在创建单选框对象时，同一组的单选框必须具有相同的 NAME 属性，也就是说，不同 NAME 属性的单选框构成不同的组。只有在同一组单选框对象中，才能获得一次只能选择一项的功能。

以下示例显示了如何创建单选框和复选框，代码如下：

```
<HTML>
<HEAD>
  <TITLE>单选框与复选框</TITLE>
</HEAD>
<BODY>
<DIV align=center>
<FORM NAME=form1>
<H2>在线调查</H2>
您感觉本网站的主页效果如何：<P>
<INPUT type=radio name=radio1 checked id=myRadio1>
  <LABEL for="myRadio1">非常好 </LABEL>
<INPUT type=radio name=radio1 id=myRadio2>
  <LABEL for="myRadio2">好 </LABEL>
<INPUT type=radio name=radio1 id=myRadio3>
  <LABEL for="myRadio3">一般</LABEL>
<P>
您感觉本网站的页面下载速度如何：<P>
<INPUT type=radio name=radio2 checked id=myRadio4>
  <LABEL for="myRadio4">快 </LABEL>
```

```
<INPUT type=radio name=radio2 id=myRadio5>
  <LABEL for="myRadio5">可以接受 </LABEL>
<INPUT type=radio name=radio2 id=myRadio6>
  <LABEL for="myRadio6">难以忍受</LABEL>
<P>
您希望本网站中出现以下什么栏目（多选）：<P>
<INPUT type=checkbox name=check1 id=myCheck1>
 ,<LABEL for="myCheck1">新闻</LABEL>
<INPUT type=checkbox name=check2 id=myCheck2>
  <LABEL for="myCheck2">娱乐</LABEL>
<INPUT type=checkbox name=check3 id=myCheck3>
  <LABEL for="myCheck3">教育</LABEL>
<P><BR>
<INPUT TYPE=submit VALUE="提交">
</FORM>
</DIV>
</BODY>
</HTML>
```

说明：在此示例中使用了 LABEL 标记符创建表单元素的标签，使用户在单击标签文字时就相当于单击了表单元素对象，从而获得更友好的用户界面。使用 LABEL 标记符时，应将其 FOR 属性的值指定为对应表单元素的 ID 属性值。

该示例实现了一个在线调查的界面部分，其中包括两组单选框和一组复选框，效果如图 6.8 所示。

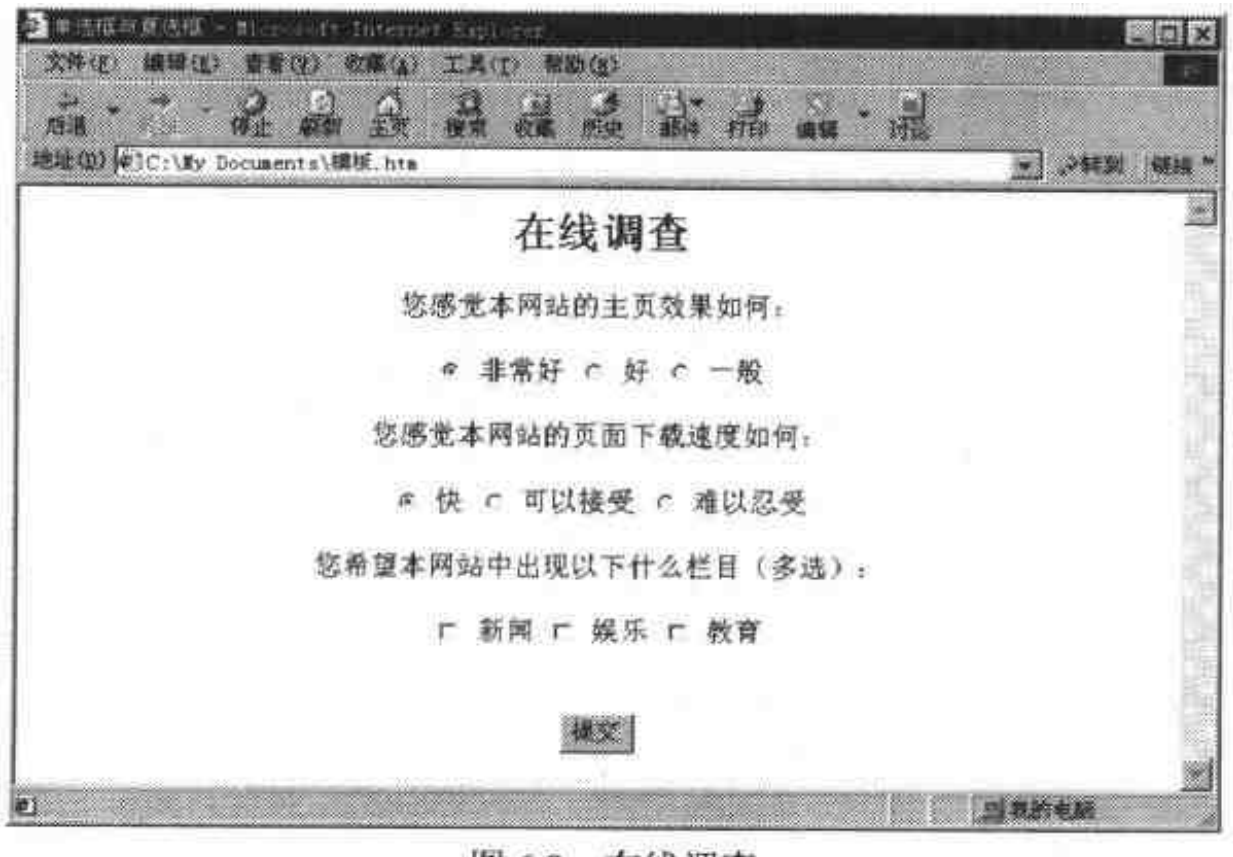


图 6.8 在线调查

6.3.2 属性、方法与事件

由于 radio 对象和 checkbox 对象都是让用户通过单击进行选择，因此它们具有相同的属性、方法和事件，如表 6.3 所示。

表6.3 radio和checkbox对象的属性、方法和事件

项　目	说　明
blur()	将焦点从对象上移走
checked	确定对象是否被选中，取值为 true 或 false
click()	相当于单击鼠标
defaultChecked	确定对象的初始状态是否为被选中，取值为 true 或 false。如果在 INPUT 标记符中使用了 CHECKED 属性，则此属性为 true，否则为 false
focus()	将焦点移动到对象上
form	表示对象所在的表单
name	表示对象的名称
onBlur	当从对象上移开焦点时触发
onClick	当单击对象时触发
onFocus	当对象获得焦点时触发
type	表示对象的类型，对应于 INPUT 标记符的 TYPE 属性。对于单选框，该值为 radio；对于复选框，该值为 checkbox
value	表示对象的值，对应于 INPUT 标记符中的 VALUE 属性

6.3.3 示例

以下示例实现了一个简单的电子商务页面，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用单选框与复选框</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function checkAll()  //此函数用于选中所有复选框。
{
with(document.form1)
{
for(i=0;i<length;i++)
if(elements[i].type=="checkbox")
  elements[i].checked=true;
}
```

```
}

function checkIt(obj)  //此函数用于选中与数量框相对应的复选框。
{
index=obj.name.charAt(3);
/* 由于数量文本框的名称都是 numx，其中 x 是数字，因此可以用这种方式获得相应复选框
的索引。charAt() 是 String 对象的方法，用于获得指定位置的字符。*/
eval("if(document.form1.check"+index+".checked==false)document.form1.
check"+index+".checked=true");
/* eval() 函数的作用是将字符串作为 JavaScript 语句执行。*/
}

function mybuy()
{
with(document.form1)
{
for(i=0;i<length;i++)
 if(elements[i].name=="card")  // 找到第一个信用卡单选框。
   break;
for(j=0;j<4;j++)
 if(elements[i++].checked==true)  // 找到选中的信用卡单选框在小组中的索引。
   break;
switch(j)
{
case 0:
  var cardStr="VISA";break;
case 1:
  cardStr="MasterCard";break;
case 2:
  cardStr="招行一卡通";break;
default:
  cardStr=myCard.value;
}
}

newWin=open("","","width=500,height=450")
newWin.document.write("您使用卡号为"+document.form1.cardNum.value+"的
"+cardStr+"购买了以下产品：<P>")
```

```
for(i=0;i<document.form1.length;i++)
 if((document.form1.elements[i].type=="checkbox")&&(document.form1.e
lements[i].checked==true))
{
  index=document.form1.elements[i].name.charAt(5);
/* 由于复选框的名称都是 checkx，其中 x 是数字，因此可以用这种方式获得数字文本框
的索引。*/
  num=eval("document.form1.num"+index+".value");
  newWin.document.write(document.form1.elements[i].value+"  
"+num+" 件<P>");
}
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME=form1>
<H2 align=center>产品列表</H2>
<TABLE>  <!-- 使用表格协助排版。 -->
<TR>
 <TD><INPUT type=checkbox name=check1 id=myCheck1 value="产品 A"><LABEL
for=myCheck1>  </LABEL>
<!--   是 HTML 特殊字符中的空格，此处使用它是为了使用户获得更好的操作感觉，
因为只需单击复选框或空格所在位置即可设置复选框状态。-->
 <TD> 数 量 ： <INPUT  size=2  name=num1  value=1  onfocus=this.select()
onchange=checkIt(this);>
<!-- 单击数量框时即选中其中数字，以便修改；如果更改了其中的数字，则自动使相应复选
框成为选中状态（如果尚未选中）。-->
 <TD>  产品 A……
<TR>
 <TD><INPUT type=checkbox name=check2 id=myCheck2 value="产品 B"><LABEL
for=myCheck2>  </LABEL>
 <TD> 数 量 ： <INPUT  size=2  name=num2  value=1  onfocus=this.select()
onchange=checkIt(this)>
 <TD>  产品 B……
<TR>
 <TD><INPUT type=checkbox name=check3 id=myCheck3 value="产品 C"><LABEL
for=myCheck3>  </LABEL>
 <TD> 数 量 ： <INPUT  size=2  name=num3  value=1  onfocus=this.select()
```

```
onchange=checkIt(this)>
 <TD>  产品 C……
<TR>
 <TD><INPUT type=button name=allCheck value="全选" onClick=
"checkAll();">
<!-- 单击“全选”按钮可以选中所有复选框。-->
<TR>
 <TD><INPUT type=reset value="重选">
</TABLE>
<H3>请选择您使用的信用卡种类：</H3>
<INPUT type=radio name=card id=card1 checked><LABEL for="card1">
VISA</LABEL>
<INPUT type=radio name=card id=card2><LABEL for="card2">MasterCard
</LABEL>
<INPUT type=radio name=card id=card3><LABEL for="card3">招行一卡通
</LABEL>
<INPUT type=radio name=card id=card4><LABEL for="card4">其他</LABEL>
<INPUT name=myCard>
<P>
```

请输入您的信用卡号：

```
<INPUT name=cardNum><P>
<DIV align=center>
  <INPUT type=submit value="购买" onClick="mybuy()">
</DIV>
</FORM>
</BODY>
</HTML>
```

此示例不但应用到了单选框和复选框对象的属性和方法，还应用到了一些我们已经学过的知识，读者可以参考注释仔细体会（尤其是 eval() 函数的用法）。此示例的显示效果如图 6.9 所示，当用户修改了表单并单击“购买”按钮后，将打开一个新窗口显示用户所作的选择，如图 6.10 所示。

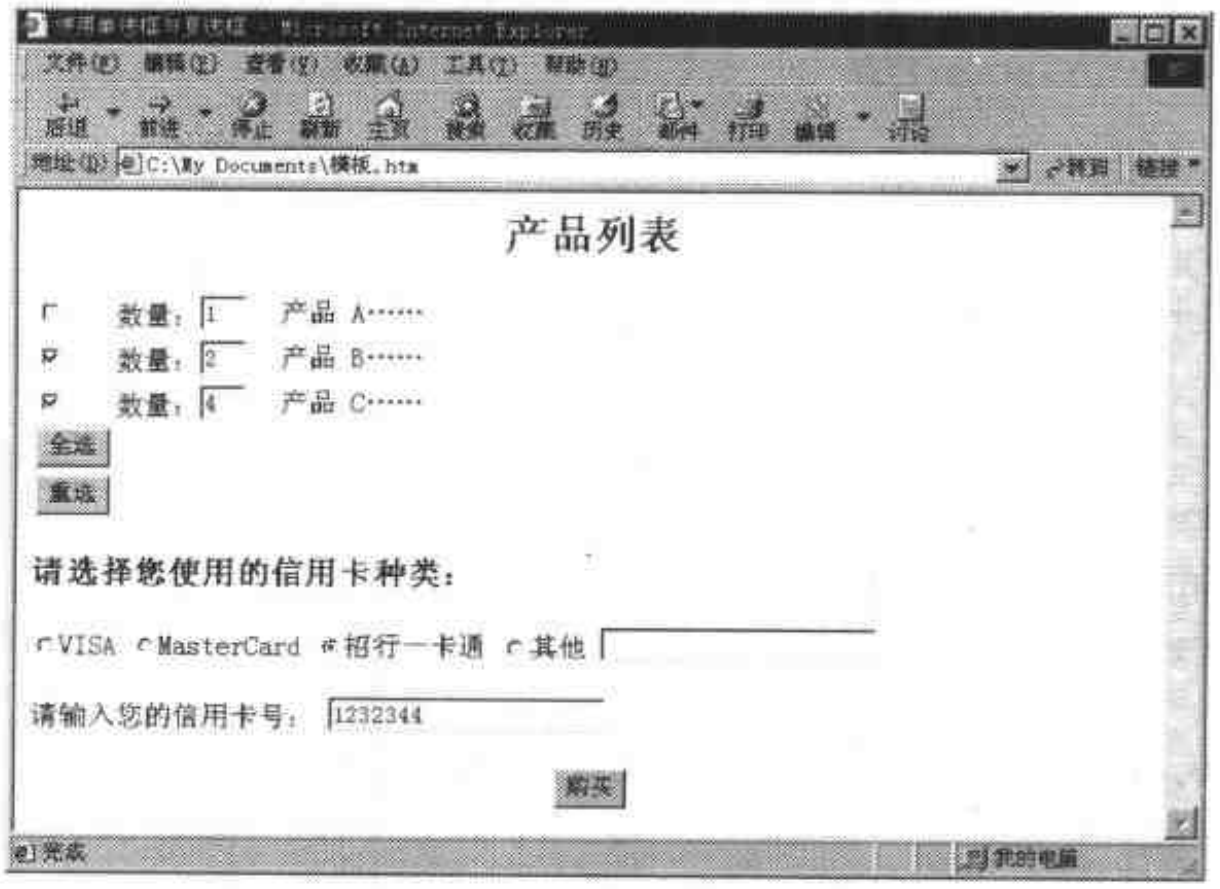


图 6.9 简单电子商务页面

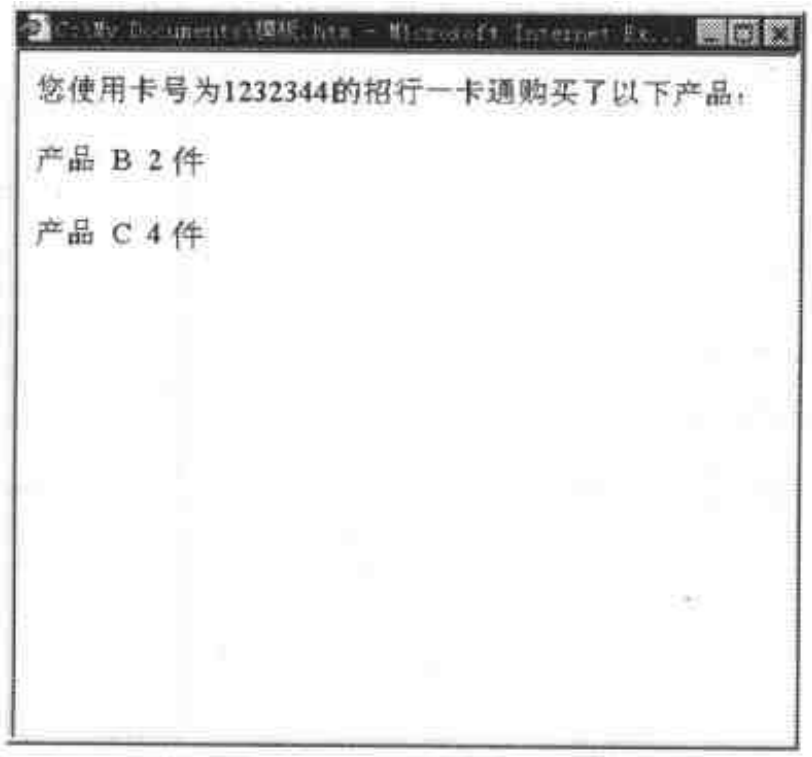


图 6.10 显示用户所作选择

6.4 按钮对象

6.4.1 概述

按钮对象包括普通按钮对象 button、提交按钮对象 submit 和重置按钮对象 reset，它们分别是在 INPUT 标记符中将 TYPE 属性设置为 button、submit 和 reset 时创建的。

1. button 对象

普通按钮对象 button 通常用于响应用户的单击事件，从而使网页具有与一般 Windows 应用程序类似的界面效果和功能。最常见的用法是：通过为按钮定义 onClick 事件响应函数，执行一段 JavaScript 程序，从而获得与用户交互的效果（实际上，我们在前面的示例中已经多次用到这种方法）。

例如，在以下示例中，当用户单击“显示”按钮时，将在“显示密码”文本框中显示出用户在“输入密码”文本框中输入的密码，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用普通按钮</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function displayPwd()
{
document.form1.myTxt.value=document.form1.myPwd.value;
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<H3>普通按钮示例</H3>
<FORM NAME=form1>
  <LABEL for=pwd1>输入密码：</LABEL>
    <INPUT name=myPwd type=password id=pwd1>
  <P>
  <LABEL for=txt1>显示密码：</LABEL>
    <INPUT name=myTxt id=txt1>
  <P>
  <INPUT type-button value-"显示" onClick-"displayPwd()">
</FORM>
</DIV>
</BODY>
</HTML>
```

这段代码的显示效果如图 6.11 所示。

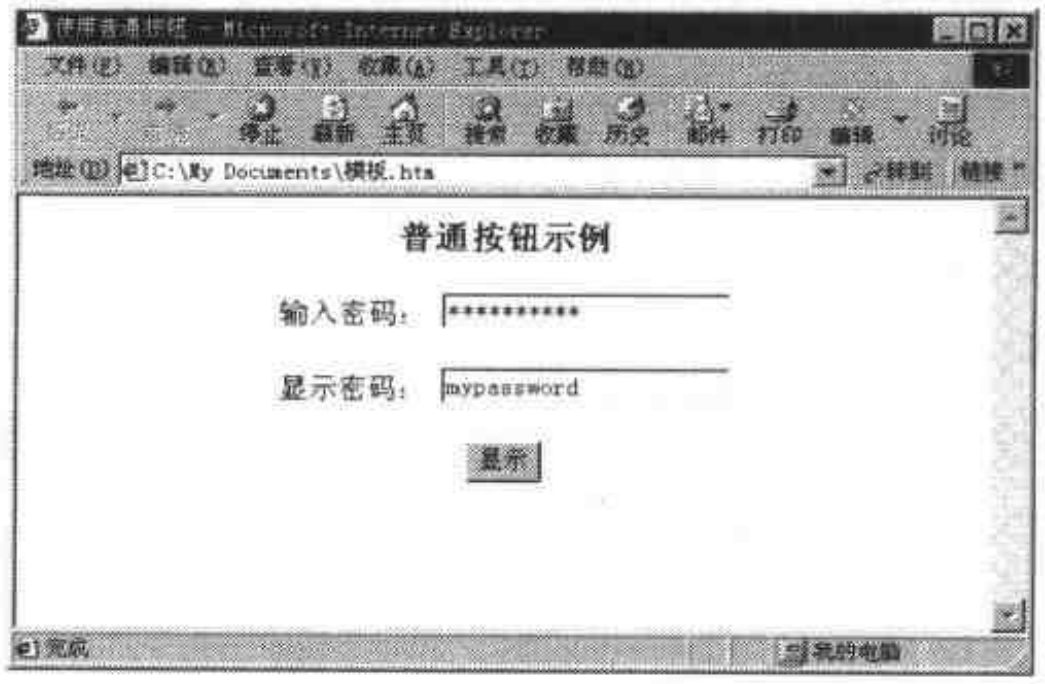


图 6.11　普通按钮示例

2. submit 对象

提交按钮 submit 是表单中的一个重要组成部分，因为只有设置了该按钮才能将表单数据提交，从而使表单具有交互功能。在实际应用过程中，提交按钮通常是必不可少的，因为不能提交的表单也就失去了信息传递的意义。

使用 submit 对象时，注意以下一些要点（在前面的一些示例中已经有所展示）：

- 在 INPUT 标记符中应将 TYPE 属性设置为 submit，并将 VALUE 属性的值设置为要在提交按钮上显示的文字。如果不设置 VALUE 属性，则浏览器使用默认文字显示提交按钮。
- 除非相应 form 对象中的 onSubmit 属性的值为 false，否则单击提交按钮的默认操作都是按照 form 标记符中的属性指定将表单数据提交。
- 可以为 submit 对象指定 onClick 事件响应函数，以便响应单击事件，但响应之后仍然要执行提交操作。
- 在相应 form 对象中指定 onSubmit 事件响应函数也可以在提交表单之前执行必要操作，例如进行表单数据有效性验证。默认情况下，FORM 标记符的 onSubmit 属性值为 true，如果将其设置为 false，则不再执行指定的表单提交操作。

例如，以下示例显示了这些要点的效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用提交按钮</TITLE>
</HEAD>
```

```
<BODY>
<DIV align=center>
<H3>提交按钮示例</H3>
<FORM NAME=form1
      action=http://www.nonexist.com/ddd.asp
      method=POST
      onsubmit="alert('响应 onSubmit 事件! '); return false">
  <LABEL for=txt1>输入姓名: </LABEL><INPUT name=myName id=txt1>
  <P>
  <INPUT type=submit onClick="alert('您单击了提交按钮! ')">  // 使用默认按
钮文字
</FORM>
</DIV>
</BODY>
</HTML>
```

这段代码的效果为：单击提交按钮后，首先响应 onClick 事件，显示提示框，如图 6.12 所示；之后响应 onSubmit 事件，显示另一个提示框，如图 6.13 所示；由于将 false 返回给 onSubmit 属性，因此不再执行 action 和 method 指定的提交操作。

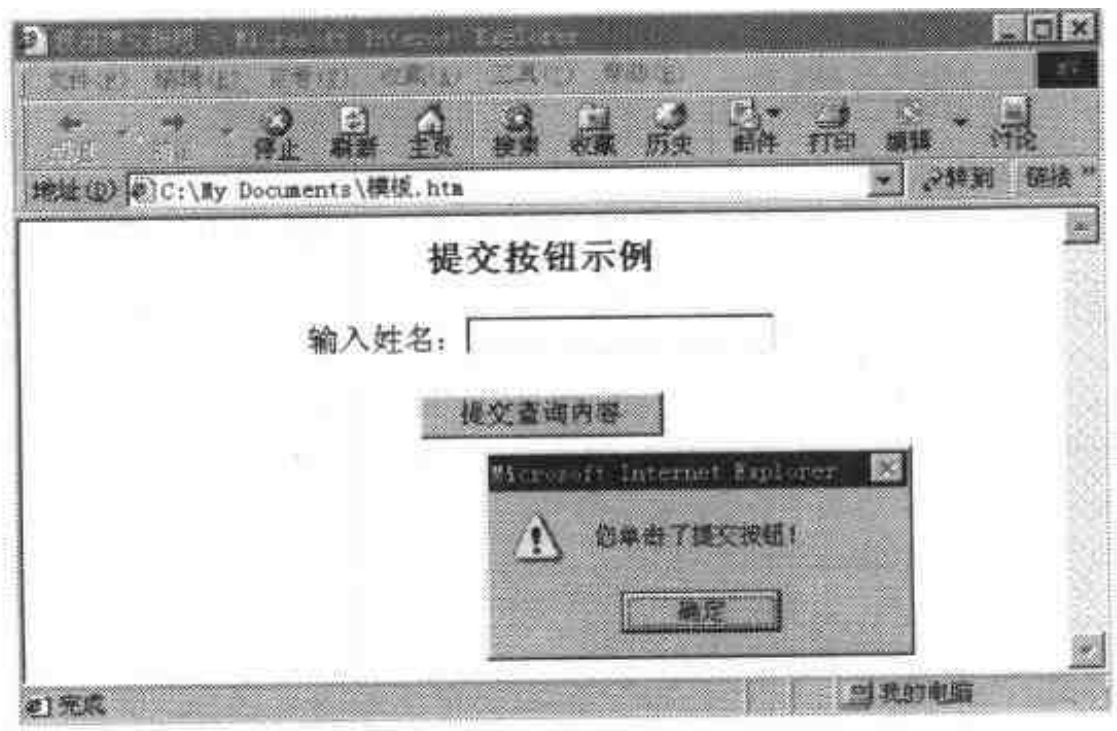


图 6.12 使用提交按钮

图 6.13 响应 onSubmit 事件

3. reset 对象

表单中的重置按钮用于将表单信息恢复为默认状态，以便用户能够重新填写。在实际应用过程中，重置按钮有时可以省略。

使用 reset 对象的要点与使用 submit 对象类似：

- 在 INPUT 标记符中应将 TYPE 属性设置为 reset，并将 VALUE 属性的值设置为要在重置按钮上显示的文字。如果不设置 VALUE 属性，则浏览器使用默认文字显示重置按钮。
- 除非相应 form 对象中的 onReset 属性的值为 false，否则单击重置按钮的默认操作都是将表单数据恢复为初始值。
- 可以为 reset 对象指定 onClick 事件响应函数，以便响应单击事件，但响应之后仍然要执行重置操作。
- 在相应 form 对象中指定 onReset 事件响应函数也可以在重置表单之前执行必要操作。默认情况下，FORM 标记符的 onReset 属性值为 true，如果将其设置为 false，则不再执行重置操作。

6.4.2 属性、方法与事件

由于 button 对象、submit 对象和 reset 对象都是让用户单击的按钮，因此它们具有相同的属性、方法和事件，如表 6.4 所示（与其他控件对象的相应功能类似，不再举例）。

表6.4 各种按钮对象的属性、方法和事件

项　目	说　明
blur()	将焦点从按钮上移走
click()	相当于单击按钮
focus()	将焦点移动到按钮上
form	表示按钮所在的表单
name	表示按钮的名称
onBlur	当从按钮上移开焦点时触发
onClick	当单击按钮时触发
onFocus	当按钮获得焦点时触发
type	表示按钮的类型，对应于 INPUT 标记符的 TYPE 属性。对于普通按钮，该值为 button；对于提交按钮，该值为 submit；对于重置按钮，该值为 reset
value	表示按钮的值，对应于 INPUT 标记符中的 VALUE 属性

6.5 选项菜单

6.5.1 概述

选项菜单使用户可以在由多个选项构成的列表或菜单中选择一个或多个选项。当用户在表单中使用 SELECT 标记符时，即创建出了一个选项菜单对象。而在选项菜单标记符内，应使用 OPTION 标记符，对应于每个具体的选项。

在 SELECT 标记符中使用 MULTIPLE 属性可以使用户选择选项菜单中的多个选项，使用 SIZE 属性可以控制显示的行数（从而获得下拉菜单或选项列表效果）。在 OPTION 标记符中使用 SELECTED 属性可以设置该选项的默认状态为选中。

以下示例显示了如何创建选项菜单，代码如下：

```
<HTML>
<HEAD>
  <TITLE>创建选项菜单</TITLE>
</HEAD>
<BODY>
<DIV align=center>
  <H3>选项菜单示例</H3>
  <FORM NAME=form1>
下面是下拉菜单：<P>
   <SELECT>  <!-- 默认的 SIZE 属性值为 1 -->
     <OPTION>选项 1
     <OPTION SELECTED>选项 2  <!-- 设置选项 2 为默认选中 -->
     <OPTION>选项 3
     <OPTION>选项 4
   </SELECT><P>
下面是选项列表：<P>
   <SELECT SIZE=4 MULTIPLE>
     <OPTION SELECTED>选项 1
     <OPTION SELECTED>选项 2
     <OPTION>选项 3
     <OPTTON>选项 4
   </SELECT>
  </FORM>
</DIV>
</BODY>
```

```
</HTML>
```

这段代码的显示效果如图 6.14 所示：第一个选项菜单实现为一个下拉菜单，单击该对象时会弹出一个列表，用户可以选择其中的选项，由于在第二个 OPTION 标记符中设置了 SELECTED 属性，因此默认选中“选项 2”（在下拉菜单中如果不为任何 OPTION 标记符设置 SELECTED 属性，则默认选中第一项）；第二个选项菜单实现为一个选项列表，由于设置了 MUTIPLE 属性，所以可以选中多项（在 Windows 系统中，要选择多项可在按住 Ctrl 键或 Shift 键时单击多个选项，前者选中多个不连续的选项，后者选中多个连续的选项），另外初始时也选中了两个选项，这是因为为相应的 OPTION 标记符设置了 SELECTED 属性（在选项列表中如果不为 OPTION 标记符设置 SELECTED 属性，则默认不选中任何选项）。

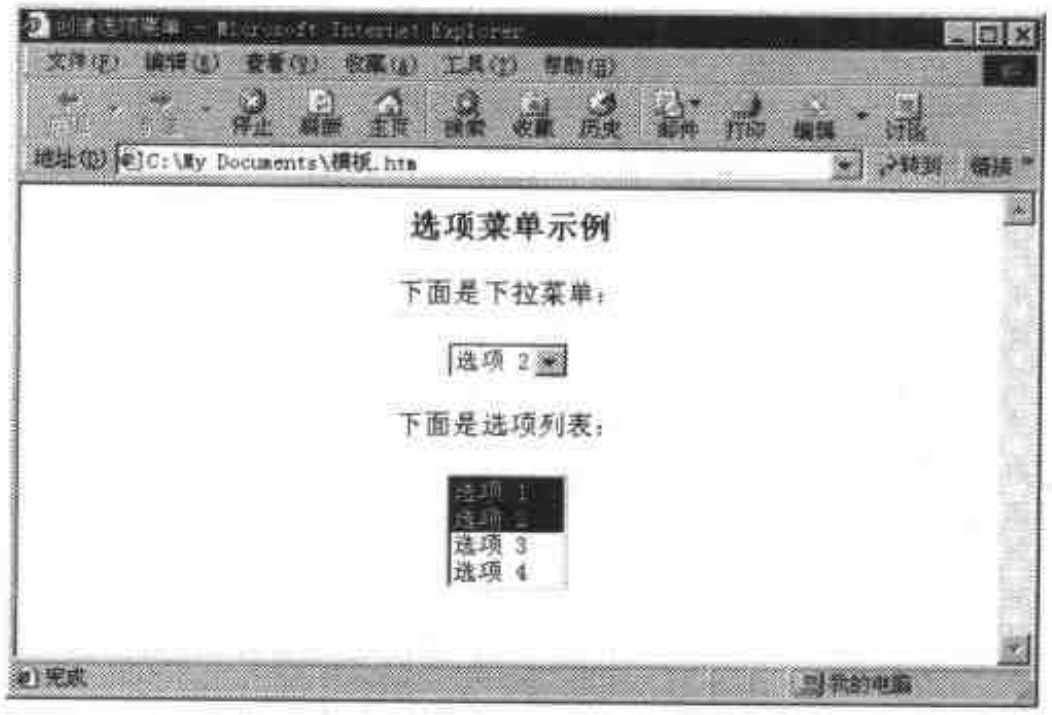


图 6.14　创建选项菜单

6.5.2　option 对象

1．创建 option 对象

当用户在 SELECT 标记符中使用了 OPTION 标记符时，即创建出了 option 对象，表示选项菜单中的一个选项。

也可以使用 option 对象的构造函数动态创建 option 对象，如以下示例所示：

```
<HTML>
<HEAD>
```

```
  <TITLE>动态创建选项菜单</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function add(myForm)
{  // 使用构造函数时，Option 的首字母必须大写，参数为选项文本和选项的值。
var option0=new Option("刘德华","star1")
var option1=new Option("王菲","star2")
var option2=new Option("张信哲","star3")
var option3=new Option("张惠妹","star4")
for(i=0;i<4;i++)  // 有关 select 对象的属性说明，请参见 6.5.3 节。
 eval("myForm.star.options[i]=option"+i)
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
  <H2>动态创建选项菜单</H2>
  <FORM>
    <SELECT name=star></SELECT><P>
    <INPUT type=button value="添加选项" onClick=add(this.form)>
    <!-- this.form 使用当前按钮对象的 form 属性引用了当前表单 -->
  </FORM>
</DIV>
</BODY>
</HTML>
```

这段代码的显示效果如图 6.15 所示，此时选项菜单中没有选项；当用户单击“添加选项”按钮后，选项菜单中添加了 4 个选项，如图 6.16 所示。

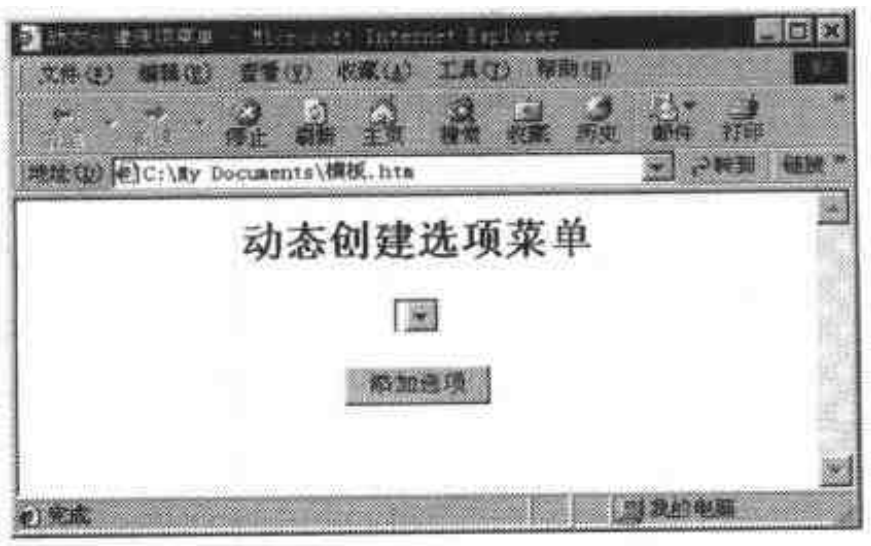


图 6.15 动态创建选项菜单

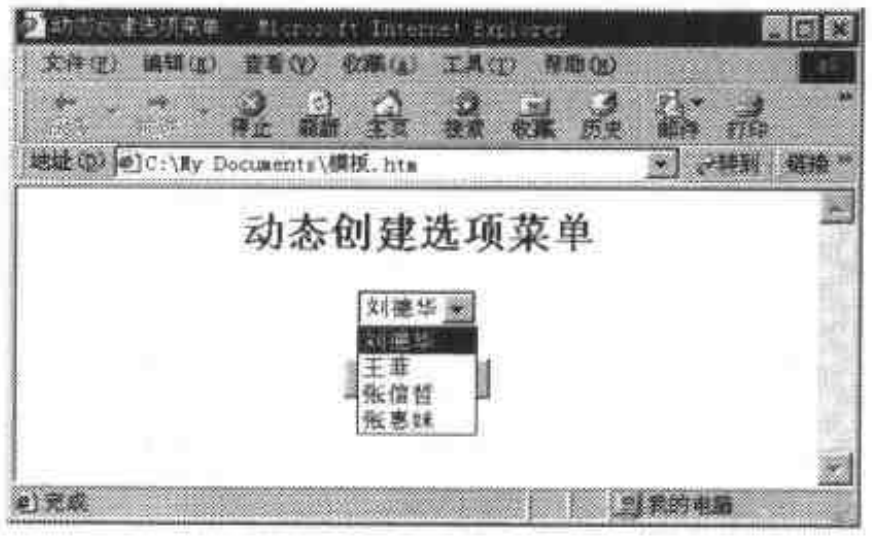


图 6.16　新添选项

2. 对象属性

option 对象包含 4 个属性（不包含方法和事件），如表 6.5 所示。

表6.5　option对象的属性

属　性	说　明
defaultSelected	表示选项默认是否被选中，取值为 true 或 false
selected	表示选项的当前选中状态，取值为 true 或 false
text	表示选项文本
value	表示选项的值，对应于 OPTION 标记符中的 VALUE 属性

以下示例显示了如何使用这些属性，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 option 对象的属性</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function display(myForm)
{
newWin=open("","","width=300,height=300")
newWin.document.write("<H3>选项菜单中的选项属性如下：</H3>")
for(i=0;i<document.form1.myList.length;i++)
 {
  if(document.form1.myList.options[i].defaultSelected==true)
   { j=i+1;
    newWin.document.write("默认选中的选项为第 "+j+" 项。<P>")
   }
  if(document.form1.myList.options[i].selected==true)
   { j=i+1;
```

```
    newWin.document.write("当前选中的选项为第 "+j+" 项；<BR>")
    newWin.document.write("其文本为' "+document.form1.myList.options
    [i].text+"'。<BR>")
    newWin.document.write("其值为' "+document.form1.myList.options
    [i].value+"'。<BR>")
   }
 }
}
//-->
</SCRIPT>
</HEAD>
<BODY>
  <H3>选项菜单示例</H3>
  <FORM NAME=form1>
   <SELECT name=myList>
     <OPTION VALUE=opt1>选项 1
     <OPTION VALUE=opt2 SELECTED>选项 2
     <OPTION VALUE=opt3>选项 3
     <OPTION VALUE=opt4>选项 4
   </SELECT><P>
  单击以下按钮显示当前选项菜单的选项信息：<P>
    <INPUT type=button value="显示信息" onclick=display(this.form)>
  </FORM>
</DIV>
</BODY>
</HTML>
```

这段代码的效果为：在选项菜单中选中了一项之后，单击“显示信息”按钮，则在一个新窗口中显示默认选项和新选中选项的信息，如图 6.17 所示。

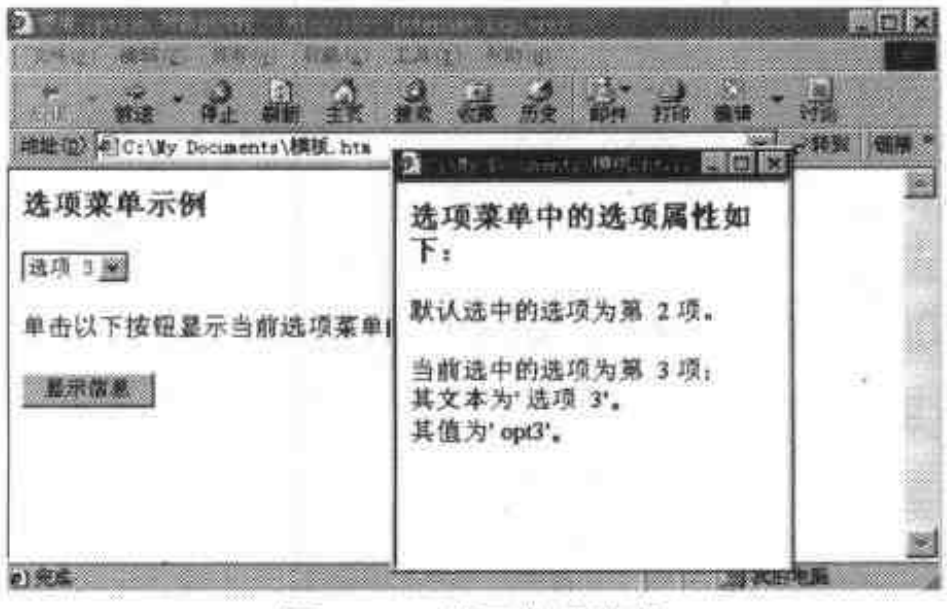


图 6.17 显示选项信息

6.5.3 select 对象

1. 属性、方法和事件

select 对象包含多个属性、方法和事件，如表 6.6 所示。

表6.6 select对象的属性、方法和事件

项　目	说　明
blur()	将焦点从选项菜单上移走
focus()	将焦点移动到选项菜单上
form	表示选项菜单所在的表单
length	表示选项菜单中选项的数目
name	表示选项菜单的名称，对应于 SELECT 标记符中的 NAME 属性
options	表示选项菜单中各选项的数组，该数组还有 length 和 selectedIndex 子属性，分别表示选项菜单中选项的个数和当前选中选项的索引（如果选中多项，则表示第一个选项的索引）
selectedIndex	表示选项菜单中被选项的索引（如果选中多项，则表示第一个选项的索引），与使用 options.selectedIndex 效果一样
onBlur	当从选项菜单上移开焦点时触发
onChange	当从选项菜单上移开焦点并且所选选项发生变化时触发
onFocus	当选项菜单获得焦点时触发
type	表示选项菜单的类型，如果为 SELECT 标记符设置了 MULTIPLE 属性，则值为 select-multiple；否则值为 select-one

2. 导航列表示例

以下示例实现了一个 Web 页导航列表，在该列表中将主页上重要的超链接组织起来，使浏览者可以在这个列表中寻找自己需要查看的网页，而不用在整个主页上查找超链接的具体位置，代码如下：

```
<HTML>
<HEAD>
<TITLE>导航列表示例</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function jump()
{
location=document.form1.navbar.options[document.form1.navbar.selecte
dIndex].value;
// 此处使用了 location 对象，有关说明请参见下一章。
}
//-->
</SCRIPT>
</HEAD>
```

```
<BODY>
<DIV align=center>
<H2>请在以下列表中选择任意选项，以便导航到需要的页面……</H2>
<FORM name=form1>
<SELECT onchange="jump()" name="navbar">
        <OPTION selected>--导航栏--</OPTION>
        <OPTION value="./商家业务中心.htm">商家业务中心</OPTION>
        <OPTION value="./超级市场.htm">超级市场</OPTION>
        <OPTION value="./商贸中心.htm">商贸中心</OPTION>
</SELECT>
</FORM>
</DIV>
</BODY>
</HTML>
```

当用户在导航列表中选择了一个选项（如图 6.18 所示）之后，当前窗口中将显示对应的页面（该网页选自 8848 网站），如图 6.19 所示。

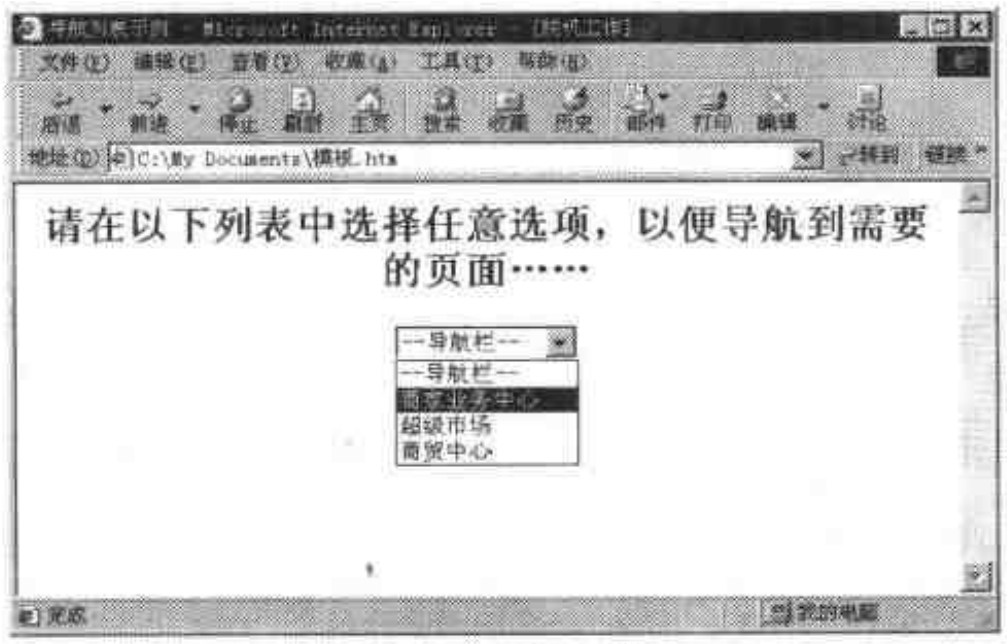


图 6.18 导航列表

图 6.19 导航到的网页

3. 导航到新窗口

如果在导航时希望保留原来打开的网页，那么可以将目标网页在一个新窗口中打开。这种选项列表常用于组织“友情链接”、“合作伙伴”等外部网页。

要实现这样的效果，只需将刚才示例中的 jump()函数更改为如下所示即可，请读者自行尝试：

```
function jump()
{window.open(document.form1.navbar.options[document.form1.navbar.sel
ectedIndex].value);
}
```

6.6 隐藏字段与 cookie

在表单中还有一种特殊的对象，叫做隐藏字段（hidden）。该对象不在表单中显示，但可用于传递信息。除了隐藏字段以外，还有一种方式用于在更广泛的范围内保存和传递信息，即使用 cookie。

本节介绍如何在客户端脚本中使用隐藏字段和 cookie。

6.6.1 使用隐藏字段

如果在表单中使用 INPUT 标记符，同时将 TYPE 属性设置为 hidden，则创建出了一个隐藏字段。隐藏字段在网页中不显示，但其值随着其他表单数据一起提交。

隐藏字段对象（hidden）具有如表 6.7 所示的属性。

表6.7 hidden对象的属性

属　性	说　明
form	表示隐藏字段所在的表单
name	表示隐藏字段的名称，对应于 INPUT 标记符中的 NAME 属性
type	表示隐藏字段的类型，取值为 hidden
value	表示隐藏字段的值，对应于 INPUT 标记符中的 VALUE 属性

隐藏字段通常用于提交隐藏的或不需要用户输入的信息。例如，在以下示例中通过隐藏字段将用户所用浏览器的类型信息保存下来，当需要时可以将其随其他数据一起提交到服务器，代码如下：

```
<HTML>
```

```
<HEAD>
<TITLE>隐藏字段示例</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function acquireInfo()
{  document.form1.info.value=navigator.appName;
}
function process()
{
var submitStr="";
submitStr+=document.form1.myName.name+"="+document.form1.myName.valu
e+" & ";
submitStr+=document.form1.eMail.name+"="+document.form1.eMail.value+
" & ";
submitStr+=document.form1.info.name+"="+document.form1.info.value;
document.form2.myInfo.value="您提交了以下数据："+submitStr;
return false;  //确保表单不提交。
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<H2>用户信息……</H2>
<FORM name=form1 onSubmit="return process()">
<TABLE>
  <TR><TD>姓名：<TD><INPUT name=myName>
  <TR><TD>电子邮件：<TD><INPUT name=eMail>
</TABLE>
  <INPUT type=hidden name=info value=""><P><BR>
  <INPUT type=submit value="提交" onClick=acquireInfo()>
</FORM>
<FORM name=form2>
 <TEXTAREA name=myInfo ROWS=5 COLS=30 value=""></TEXTAREA>
</FORM>
</DIV>
</BODY>
</HTML>
```

此示例的效果为：在表单中输入信息后单击“提交”按钮，则在多行文本框中显示出

有关要提交的表单数据的信息，如图 6.20 所示。

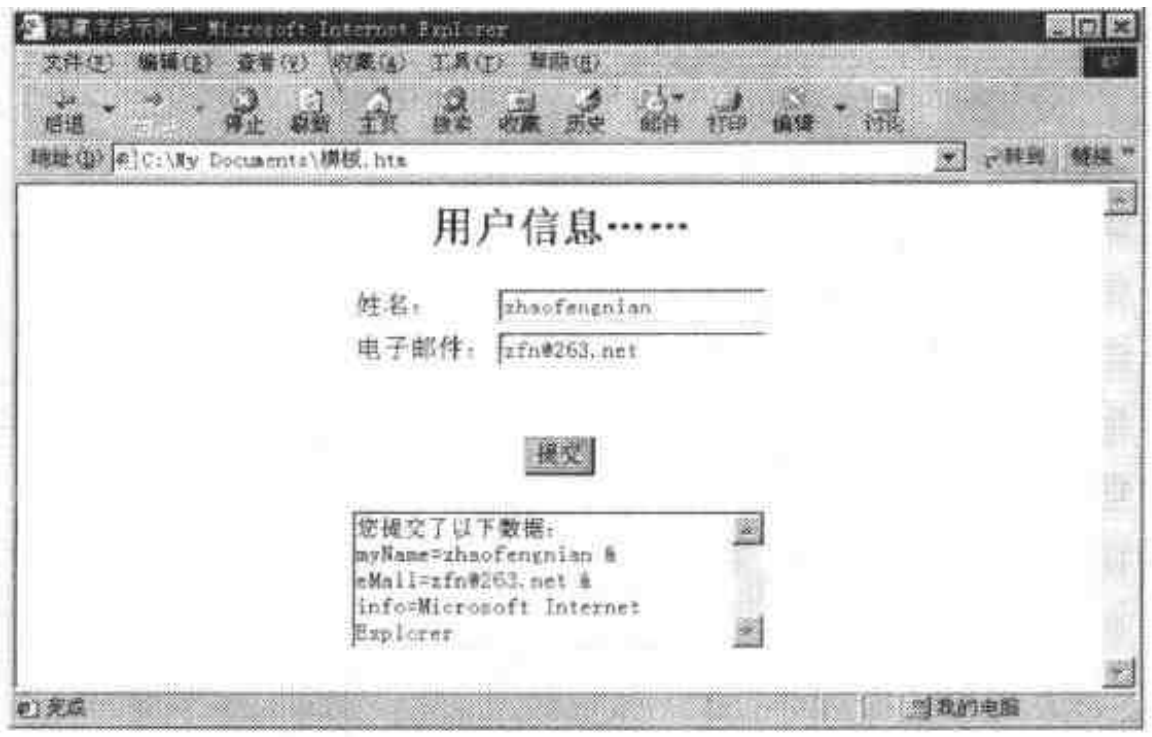


图 6.20　使用隐藏字段

隐藏字段除了具有提交隐藏信息的功能以外，还可以用于保存状态信息，以便在多个表单间共享，如以下示例所示：

```
<HTML>
<HEAD>
  <TITLE>共享数据</TITLE>
</HEAD>
<FRAMESET COLS="*,10" border=0>
  <FRAME name=cntFrm src="page1.htm">
  <FRAME name=ctrlFrm src="control.htm">
</FRAMESET>
</HTML>
```

control.htm 文件的代码如下：

```
<HTML>
<BODY>
<FORM name=ctrlForm target=_top action="http://www.nonexist
.com/process.asp">
  <INPUT name=myName type=hidden>  <!--隐藏字段用于保存信息。-->
  <INPUT name=eMail type=hidden>
</FORM>
</BODY>
</HTML>
```

page1.htm 文件的代码如下：

```
<HEAD>
  <TITLE>第一步：填写姓名</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function process1()
{
parent.frames[1].document.ctrlForm.myName.value=document.form1.name1.
value;
window.location="./page2.htm"
}
//  >
</SCRIPT>
</HEAD>
<BODY>
<FORM name=form1 action="">
  请输入姓名：<INPUT name=name1><P>
<INPUT type=button value="下一步" onClick=process1()>
</FORM>
</BODY>
</HTML>
```

page2.htm 文件的代码如下：

```
<HTML>
<HEAD>
  <TITLE>第二步：填写 eMail</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function process2()
{
parent.frames[1].document.ctrlForm.eMail.value=document.form1.eMail.
value;
location.href="page3.htm"
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM name=form1>
  请输入 eMail：<INPUT name=eMail><P>
<INPUT type=button value="下一步" onClick="process2()"
```

```
</FORM>
</BODY>
</HTML>
```

page3.htm 文件的代码如下：

```
<HTML>
<HEAD>
  <TITLE>完成</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
document.write("<H2>谢谢!</H2>")
document.write("您填写的信息为：<P>")
document.write("姓名="+parent.frames[1].document.ctrlForm.myName
.value+"<BR>")
document.write('eMail="+parent.frames[1].document.ctrlForm.eMail.val
ue)
//-->
</SCRIPT>
<FORM>
<INPUT type=button value="完成" onClick=parent.frames[1].ctrlForm
.submit()>
</BODY>
</HTML>
```

本示例的效果如下：在第一页中填写姓名，如图 6.21 所示；单击“下一步”按钮后显示第二页，在其中填写 eMail 信息，如图 6.22 所示；单击“下一步”按钮后显示第三页，其中显示了要提交的表单信息，如图 6.23 所示；单击“完成”按钮，则将表单提交。

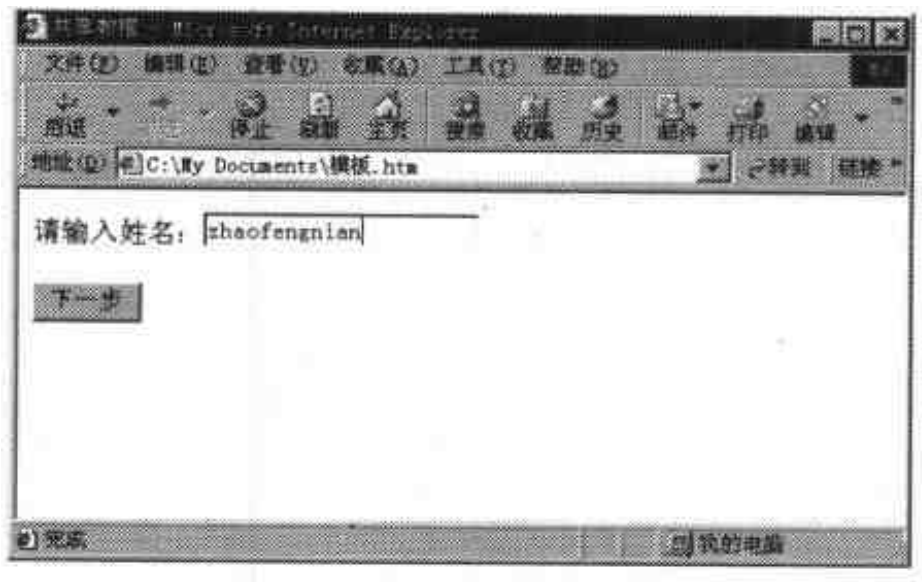


图 6.21　第一页

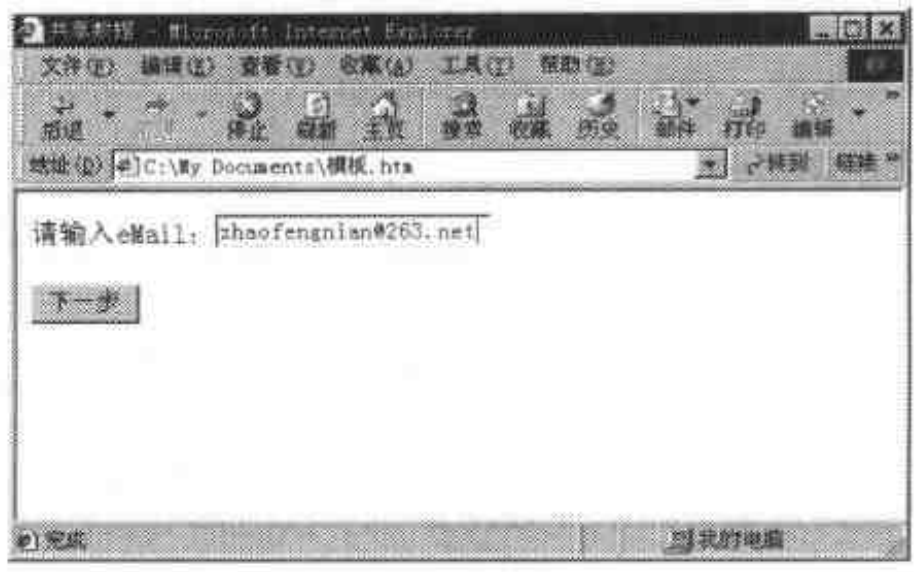


图 6.22　第二页

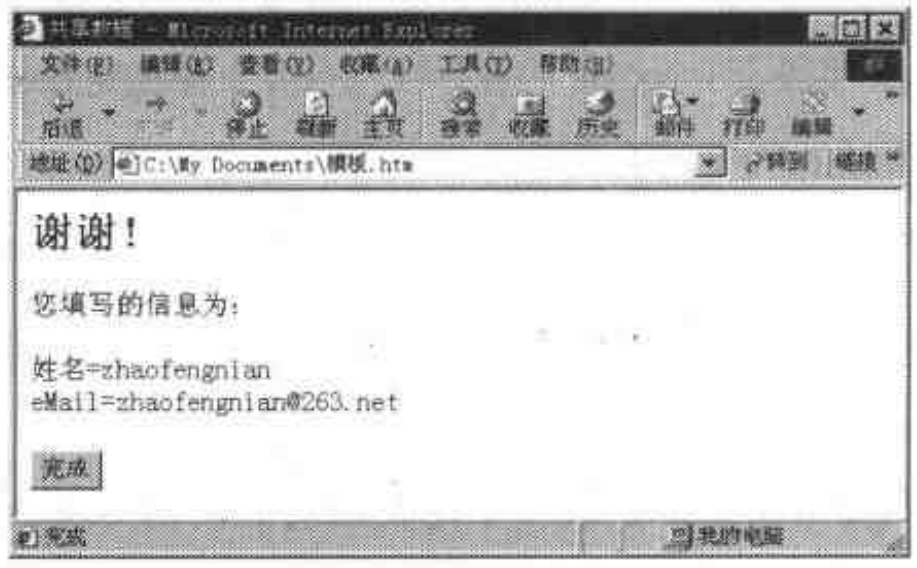


图 6.23　第三页

说明：虽然本示例实现的功能很简单，但体现了通过使用隐藏字段将多个表单的结果汇总起来发送给服务器的技术，请读者仔细体会这种用法。

6.6.2　使用 cookie

虽然隐藏字段可以用于保存一定的状态信息，但显然仅局限于比较小的范围且只能在一次浏览器会话期间保存，如果要在更大的范围（例如整个网站）或更长的时效内保存状态信息，则需要使用 cookie。

1. 什么是 cookie

cookie 是一种保存持续状态信息和其他信息的一种方式，它表示由 Web 浏览器存储起来的一小组数据，通过这些数据可以在 Web 页内共享信息。一个典型的 cookie 中存储着与某个站点相关的用户信息，以便将来连接到该站点时使用。

对于不同的浏览器，cookie 的存储位置不同。对于 Navigator，所有的 cookie 都存储在

一个名为 cookie.txt 的文件里；对于 Internet Explorer，每个 cookie 都有自己单独的文本文件，这些文件默认存储在\Windows\cookies 文件夹。

cookie 由一个或多个 NAME=VALUE 对组成，并包括几个特殊字段：expires、domain、path 和 secure。

- expires 字段　此字段表示 cookie 的过期日期。如果不指定此字段，则 cookie 在当前浏览器会话结束时过期。为了使 cookie 的持续时间变长，应将 expires 字段设置为合法的时间值（可以使用 Date 对象）。如果指定了 expires 字段，则浏览器会话结束后，cookie 被写到文本文件中；否则浏览器会话结束后，cookie 自动销毁。
- domain 字段　此字段表示 cookie 的有效域。例如，如果指定 domain=www.javascript.com，则 cookie 被该域的所有网页共享。
- path 字段　此字段确定比 domain 更进一步的有效范围，取值为文件夹名。例如，如果同时指定 domain=www.javascript.com 和 path=/examples，则表示 cookie 在 www.javascript.com/examples 范围内有效。
- secure 字段　如果指定此字段（不需要取值），则 cookie 只能在安全通信通道（HTTPS）上传递。

2. 读写 cookie

在 JavaScript 中，cookie 是 document 对象的一个属性，可以通过使用该属性来使用 cookie。

在设置 cookie 时，应为 cookie 设置一个或多个 NAME=VALUE 对，然后指定必要的属性，并将不同属性之间以分号隔开，最后将代表 cookie 的字符串赋给 document.cookie。例如，以下语句创建了一个名为 cookie1 的 cookie，并将过期日期设置为 1 天之后：

```
var expireDate=new Date();
expireDate.setTime(expireDate.gettime()+24*60*60*1000);
document.cookie="cookie1=this is my cookie.;expires="+
expireDate.toGMTString();
```

在指定 cookie 的名称和值时，可以使用任意字符串，但字符串中不允许使用分号、逗号和空格。在指定 expires 属性时，最好使用 Date 对象，以确保日期值采用了正确的格式。在刚才设置 cookie1 的过程中，使用了 Date()对象的 setTime()（将 Date 对象设置为毫秒值）、getTime()（获得日期对象的毫秒值）和 toGMTString()（将毫秒值转换为 GMT 时间字符串）

方法，以确保获得所需格式的过期时间。

设置了 cookie 之后，只有将其读取出来并进行必要的解析，才能使其发挥作用。由于我们通常只使用 cookie 的值，因此如何获取对应于指定 cookie 名称的 cookie 值就是至关重要的。

以下函数用于读取 cookie 中对应于 name 的 value，读者可以将其简单修改之后应用于自己的程序之中：

```
var myCookie=document.cookie;  //myCookie 是代表 cookie 的字符串。
function readCookie(myCookie,name)
{
var start=myCookie.indexOf(name+"=");  //找到名称 name
if(start == -1)
  alert("指定名称不存在！")
else
{
start=myCookie.indexOf("=",start)+1;  //指定值的起点
var end=myCookie.indexOf(";",start);  /* 指定值的终点（如果 cookie 的值不以分号结束，例如仅有一个 cookie 时，则此函数无法返回正确结果；请读者考虑这种情况，然后修改此函数，使其具有更广泛的适用性。）*/
var value=unescape(myCookie.substring(start,end));  //获取值
if(value==null)
  alert("没有值！")
else
  return value;
}
}
```

3. 示例 1

以下示例显示了 cookie 的基本用法，代码如下：

```
<HTML>
<HEAD>
<TITLE>cookie 示例</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function setCookie()
{  document.cookie=document.form1.myCookie.value;
   location.reload(true);  //刷新页面以清除第一个文本框中输入的信息
```

```
function readCookie()
{  document.form1.cookieValue.value=document.cookie;
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM name=form1>
  <H3>请在以下文本框中输入 cookie，然后单击"设置"按钮：</H3>
  <INPUT name=myCookie size=30><P>
  <INPUT type=button value="设置 cookie" onClick=setCookie()><P>
  <HR width=50%><P>
  <H3>单击以下按钮将在 cookie 文本框中显示出当前 cookie 的值：</H3>
  <INPUT type=button value="读 cookie" onClick=readCookie()><P>
  cookie: <INPUT name=cookieValue size=30>
</FORM>
</BODY>
</HTML>
```

此示例可以用于测试 cookie 的各种特性（请读者自行尝试）：

- 在第一个文本框中输入一个 NAME=VALUE 对，然后单击“设置 cookie”按钮，如图 6.24 所示，则所输入的 NAME=VALUE 对被加入到 cookie 中；单击“读 cookie”按钮，可以在 cookie 文本框中看到 cookie 中的内容，如图 6.25 所示。关闭浏览器，然后重新启动浏览器并打开示例页面，单击“读 cookie”按钮，cookie 文本框中不显示任何内容，这是因为没有设置 expires 属性的 cookie 在本次浏览器会话任务结束后即销毁。

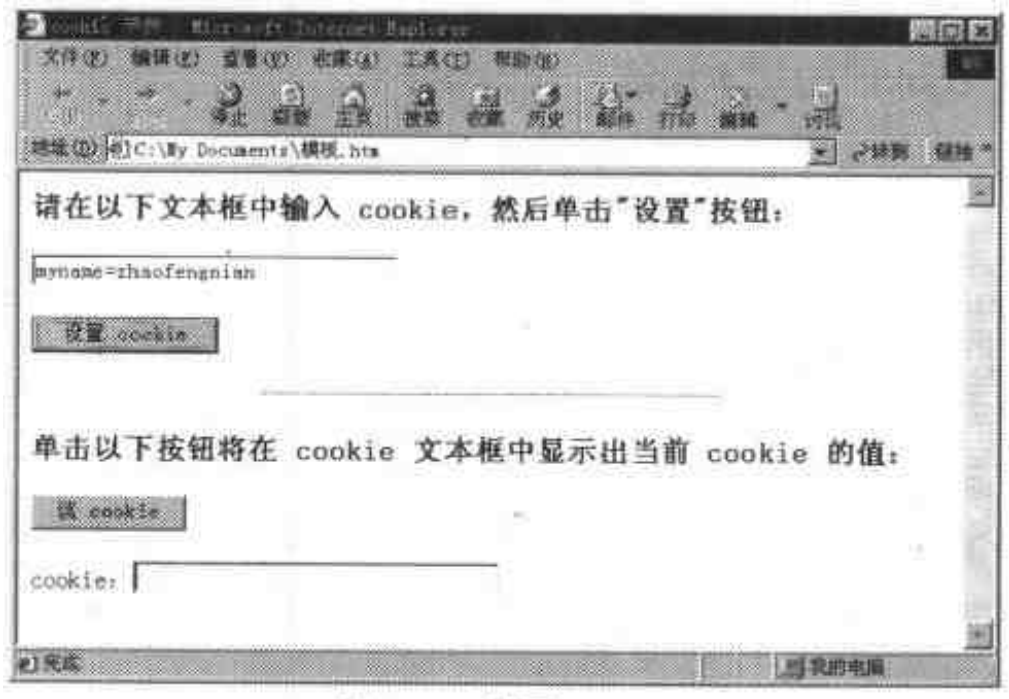


图 6.24　设置 cookie

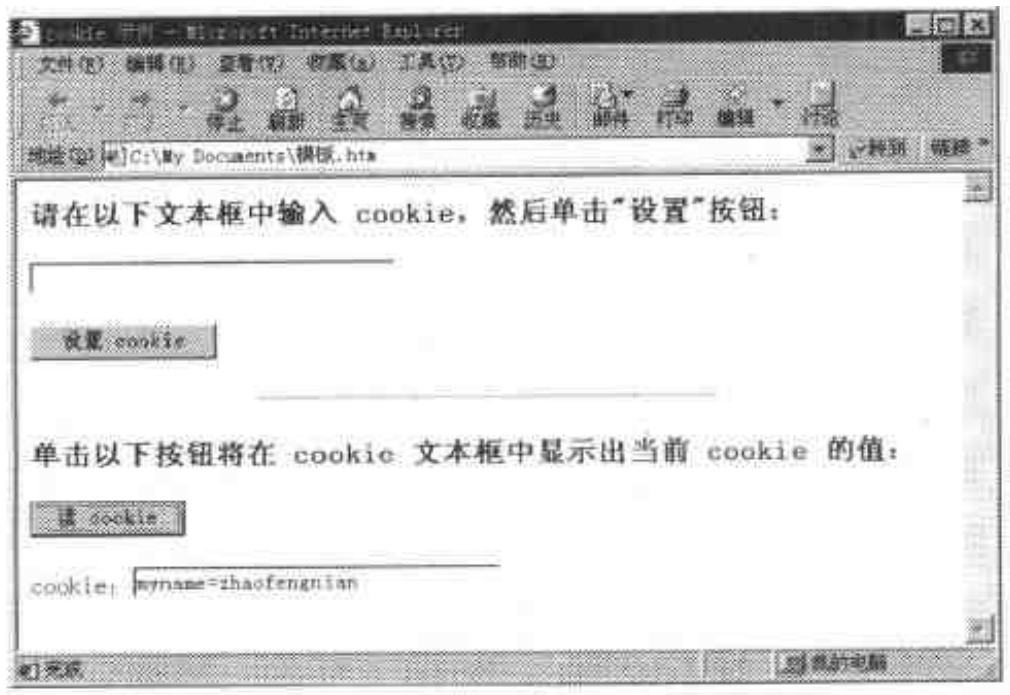


图 6.25 读取 cookie

- 在第一个文本框中输入一个 NAME=VALUE 对，然后单击“设置 cookie”按钮。多次重复以上步骤。单击“读 cookie”按钮，则在 cookie 文本框中显示出多次设置的 NAME=VALUE 对，可见写 cookie 的方式是附加式的。但如果使用了相同的 name，则后设置的 cookie 将覆盖先前的 cookie。
- 在第一个文本框中设置 cookie 时，加入 expires 属性，例如输入字符串：myname=zhaofengnian;expires='Wednesday 00-Oct-912:00:00 GMT'，注意确保 expires 的时间晚于当前时间，然后单击“设置 cookie”按钮。再在第一个文本框中输入一个没有 expires 属性的 cookie，例如输入字符串 email=zfn@263.net，然后单击“设置 cookie”按钮。单击“读 cookie”按钮，则在 cookie 文本框中仅显示 NAME=VALUE 对——"myname=zhaofengnian; email=zfn@263.net"，而不显示 expires 的值。关闭浏览器，然后重新启动浏览器并打开示例页面，则仅显示设置有 expires 属性的 cookie——"myname=zhaofengnian"。

4. 示例 2

以下示例显示了如何利用 cookie 的值传递信息，代码如下：

```
<HTML>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var myCookie=document.cookie;
var cookieHeader = "userName="
var start = myCookie.indexOf(cookieHeader)
if (start != -1)
```

```
{
  var userName = myCookie.substring(start + cookieHeader.length) /* 由
于仅有一个 cookie，因此可用此方式获得 cookie 的值。*/
  document.write("<BR><BR><H1 align=center>你好，" + userName + "!</H1>")
}
else
{
  document.write("<DIV align=center>")
  document.write("<H2>请输入您的姓名并单击确定按钮……</H2>")
  document.write('<FORM NAME = form1>")
  document.write("<INPUT NAME = userName SIZE = 40>")
  document.write("<BR><BR>")
  document.write("<INPUT TYPE = button VALUE = '确定' onClick =
setCookie()>")
  document.write("</FORM>")
  document.write("</DIV>")
}
function setCookie()
{
  var expireTime = new Date();
  expireTime.setTime(expireTime.getTime() + 30 * 24 * 60 * 60 * 1000) /*
将过期时间设置为一个月。*/
  document.cookie = "userName=" + document.form1.userName.value +
";expires=" + expireTime.toGMTString();
  location.reload(true);
}
</SCRIPT>
</HTML>
```

本示例的效果是：当用户第一次打开页面时，要求输入姓名，如图 6.26 所示；输入姓名并单击“确定”按钮后，将显示如图 6.27 所示信息；关闭浏览器，然后再次启动浏览器并打开示例页面，则直接显示图 6.27，不再进入图 6.26。

注意：使用此示例时，事先必须没有 cookie 与当前文件夹相关联，否则在欢迎页面中会将与当前文件夹关联的所有 cookie 信息都显示出来。如果要清除与当前文件夹相关联的 cookie，应打开 \windows\cookies 文件夹，找到文件名中包含示例所在文件夹的文件夹名的 cookie 文件，将其删除即可。

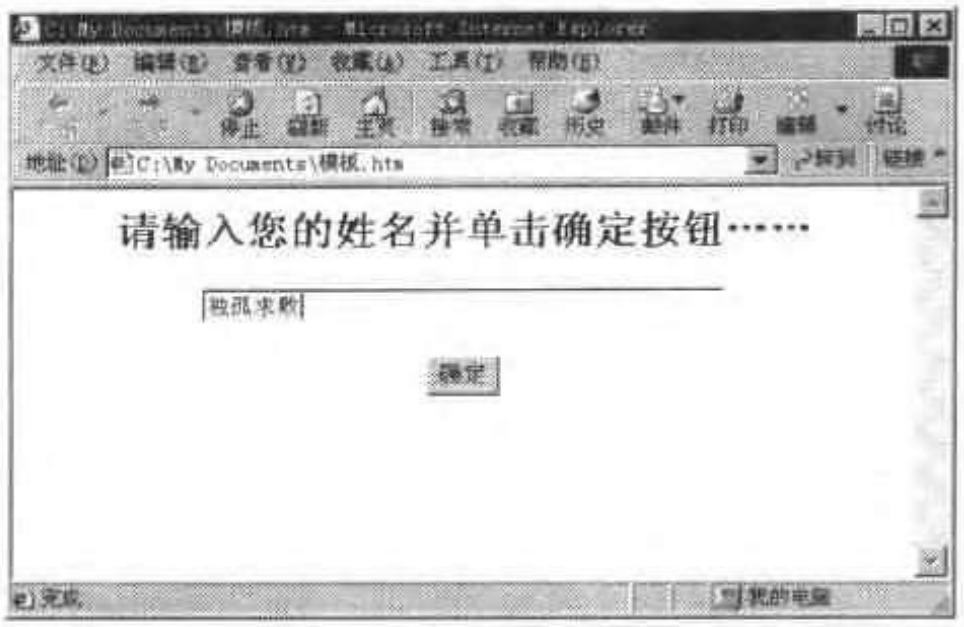


图 6.26　输入姓名

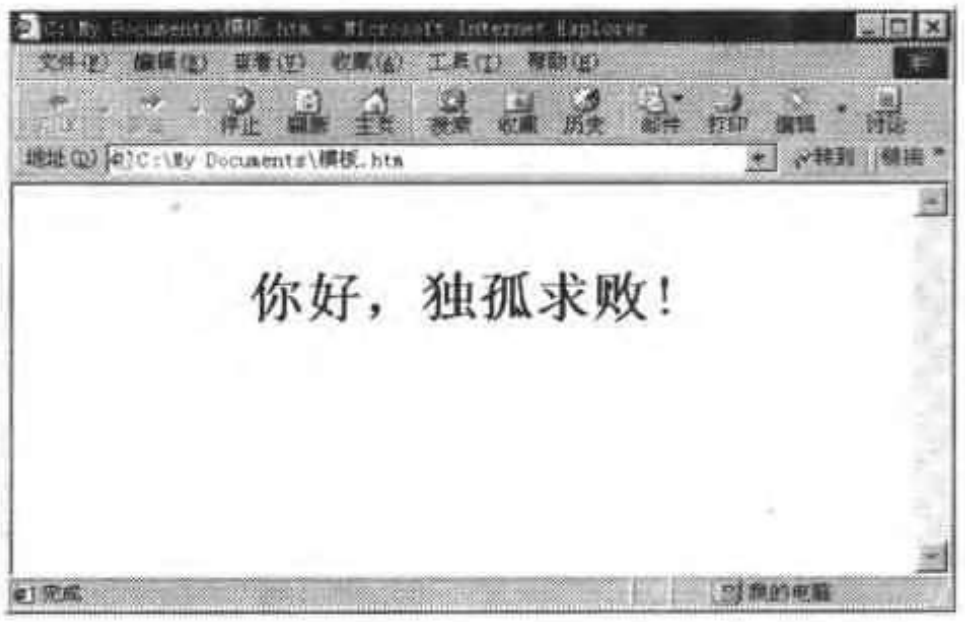


图 6.27　显示欢迎信息

第 7 章　链接与图像

超链接是网页的灵魂，无数的网页正是通过超链接连接而成一个巨大的 Web。在浏览器对象中，link、anchor、location、history、area 等对象都与超链接有关，可用于实现相关的跳转功能。图像则是使网页增光添色的重要手段，在浏览器中用 image 对象表示，通过使用该对象可以创建出各种常见的动画效果。

本章介绍浏览器对象中的各种链接对象和图像对象，主要内容包括：

- link 对象
- anchor 对象
- location 对象
- history 对象
- area 对象
- image 对象

7.1　link 对象

7.1.1　属性与事件

当用户在网页中使用了 A 标记符，并且设置了 HREF 属性，则创建出了一个 link 对象，也就是超链接对象。如果要在 JavaScript 中访问 link 对象，一般是采用 document.links[]数组，该数组包含了网页中所有具有 HREF 属性的 link 对象和具有 HREF 属性的 area 对象。

link 对象具有如表 7.1 所示的属性和事件。

表7.1 link对象的属性和事件

项 目	说 明
hash	表示 URL 中锚点的名称。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，hash 属性的值为 #top
host	表示 URL 中的主机部分。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，host 属性的值为 www.nonexist.com:80
hostname	表示 URL 中的主机名称部分。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，hostname 属性的值为 www.nonexist.com
href	表示一个链接的完整 URL。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，href 属性的值为 http://www.nonexist.com:80/examples/myPage.htm?mySearch#top
onClick	在超链接上单击鼠标时触发
onDblClick	在超链接上双击鼠标时触发
onKeyDown	在超链接上按下某键时触发
onKeyPress	在超链接上按住某键时触发
onKeyup	在超链接上松开某键时触发
onMouseDown	在超链接上按下鼠标键时触发
onMouseOut	将鼠标指针从超链接上移开时触发
onMouseOver	将鼠标指针移到超链接上时触发
onMouseUp	在超链接上松开鼠标键时触发
pathname	表示 URL 中的路径名称部分。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，pathname 属性的值为 examples/myPage.htm
port	表示 URL 中的端口部分。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，port 属性的值为 80
protocol	表示 URL 中的协议部分。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，protocol 属性的值为 http:
search	表示 URL 中的查询字符串部分。例如，对于超链接 <A href = "http://www.nonexist.com:80/examples/myPage.htm?mySearch#top">，search 属性的值为 ?mySearch
target	表示超链接结果的目标窗口，对应于 A 标记符中的 target 属性

7.1.2 示例

1. 属性示例

本示例用于说明 link 对象各属性的含义，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 link 对象的属性</TITLE>
</HEAD>
<BODY>
<H3 align=center>对于超链接 <A target=_blank href = "http://www.nonexist
.com:80/examples/myPage.htm?mySearch#top">&lt;A target=_blank href =
"http://www.nonexist.com:80/examples/myPage.htm?mySearch#top"&gt;</A
>，其各属性的取值为：</H3>
```

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
document.write("hash = "+document.links[0].hash+"<BR>")
document.write("host = "+document.links[0].host+"<BR>")
document.write("hostname = "+document.links[0].hostname+"<BR>")
document.write("href = "+document.links[0].href+"<BR>")
document.write("pathname = "+document.links[0].pathname+"<BR>")
document.write("port = "+document.links[0].port+"<BR>")
document.write("protocol = "+document.links[0].protocol+"<BR>")
document.write("search = "+document.links[0].search+"<BR>")
document.write("target = "+document.links[0].target+"<BR>")
//-->
</SCRIPT>
</BODY>
</HTML>
```

此示例的显示效果如图 7.1 所示。

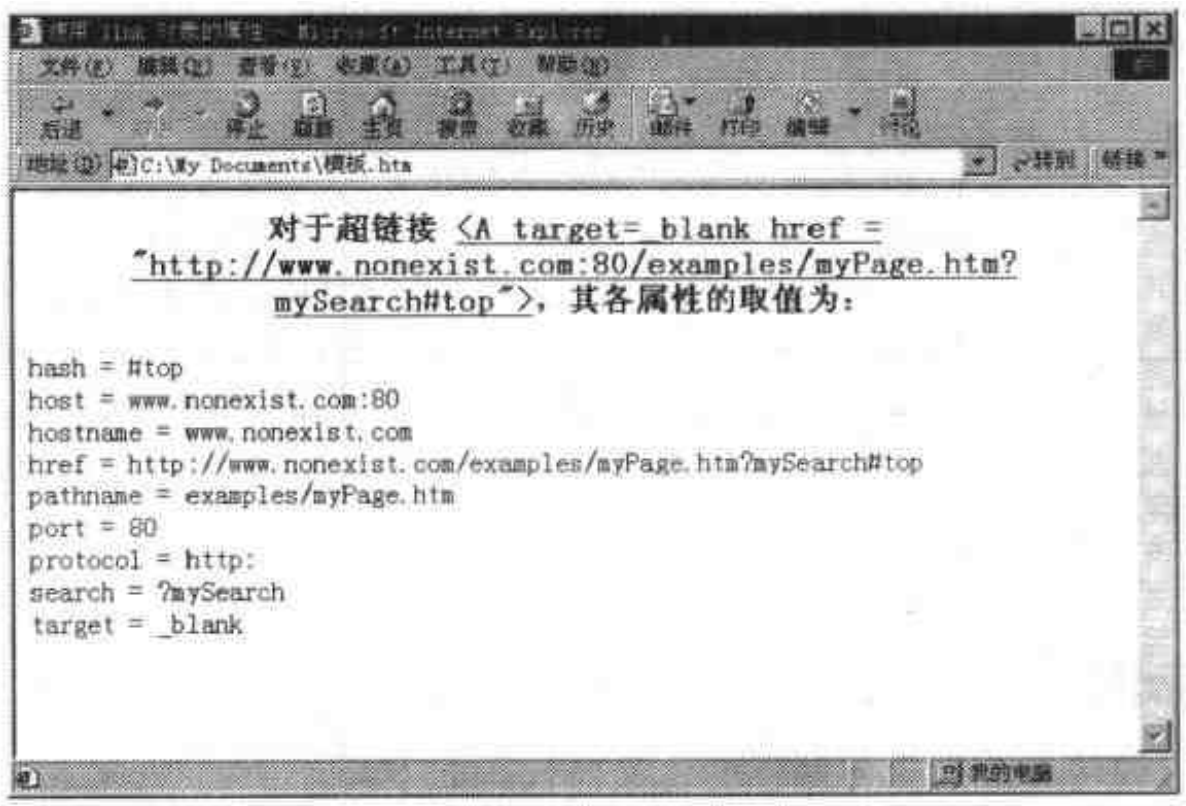


图 7.1　使用 link 对象的属性

2. 事件示例 1

在本书第 3 章的 3.2.2 小节中已经演示了如何响应 onMouseOver 和 onMouseOut 事件。以下示例用到了另外一个常用的事件：onClick，代码如下：

```
<HTML>
<HEAD>
<TITLE>onClick 事件示例</TITLE>
```

```
</HEAD>
<BODY>
<DIV align=center>
  <H2>请单击以下链接……</H2>
  <A href="./large.htm" onClick="return confirm('您将要链接到一个包含有大量图像的文件，下载速度可能较慢，是否真要链接？')">图像库</A>
</DIV>
</BODY>
</HTML>
```

此示例的效果为：单击“图像库”超链接时，弹出一个提示对话框，如图7.2所示；如果单击“取消”按钮，则不进行链接，如果单击“确定”按钮，则链接到指定页面，如图7.3所示。

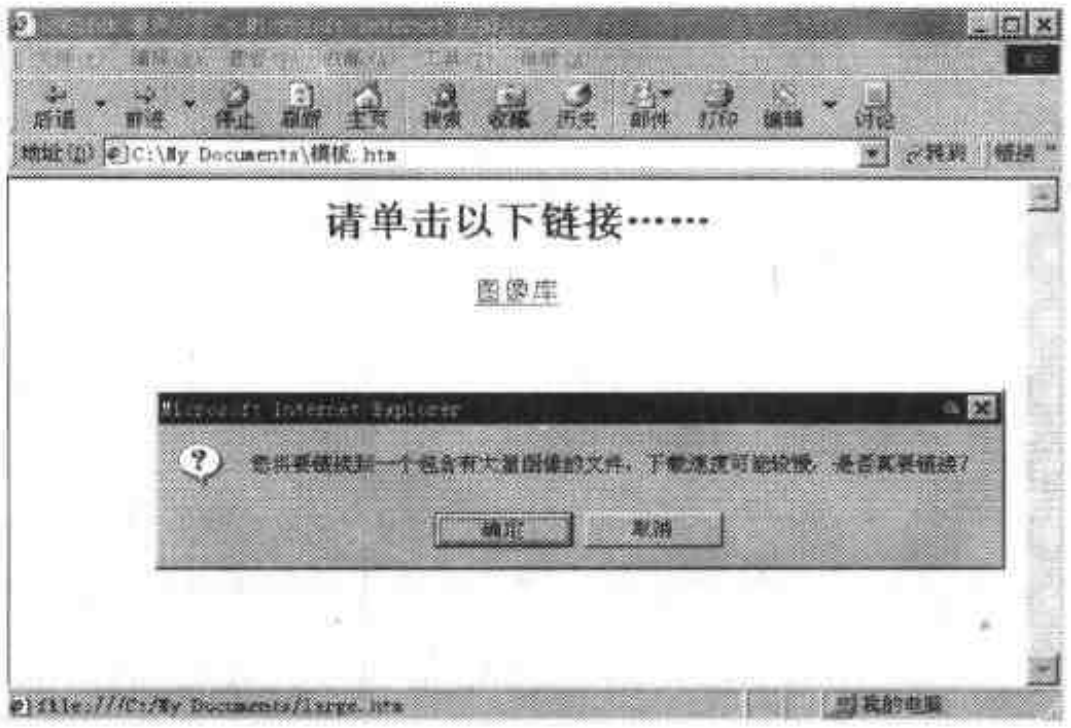


图 7.2　提醒是否链接

图 7.3　链接到指定页面

3. 事件示例 2

如果我们需要将网页制作成类似应用程序的方式，那么就应该对鼠标的双击事件 onDblClick 作出响应。不过，对于超链接而言，默认就要响应 onClick 事件，因此还来不及响应 onDblClick 事件。此时可以将 link 对象的 href 属性设置为 javascript:void(0)，如以下示例所示：

```
<HTML>
<HEAD>
<TITLE>onDblClick 事件示例</TITLE>
</HEAD>
<BODY>
<DIV align=center>
  <H2>请双击以下链接……</H2>
  <A href="javascript:void(0)" onDblClick="alert('您双击了此链接！')">示例超链接</A>
</DIV>
</BODY>
</HTML>
```

此示例的效果为：当单击超链接时不执行任何操作，当双击超链接时弹出一个提示对话框，如图 7.4 所示。

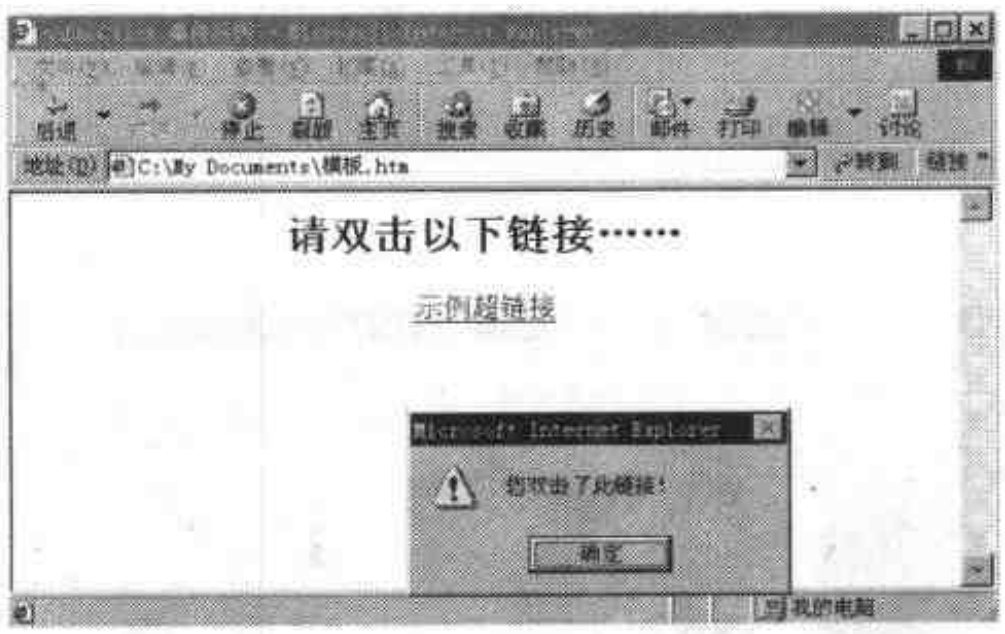


图 7.4　onDblClick 事件示例

7.2　anchor 对象

当用户在网页中使用 A 标记符，并且指定了 name 属性时，即创建出了一个 anchor 对象，也就是锚点对象。所谓锚点就是网页上的一个点，可以将其作为超链接（不仅限于本

页面内的超链接，也可以是其他页面中的超链接）的目标，从而使用户可以跳转到页面上的特定位置。

如果要在 JavaScript 中访问 anchor 对象，一般是采用 document.anchors[]数组，该数组包含了网页中所有具有 name 属性的 anchor 对象。如果一个 A 标记符同时具有 HREF 属性和 name 属性，则该对象既是 link 对象，也是 anchor 对象，它将同时出现在 document.links[]数组和 document.anchors[]数组。

anchor 对象只有一个属性 name，表示 anchor 的名称，也就是 A 标记符中 name 属性的值。

以下示例显示了 anchor 对象的用法，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 anchor 对象</TITLE>
</HEAD>
<BODY>
<A name=top>目录</A><P>
<A href=#first>第一部分</A><BR>
<A href=#second>第二部分</A><BR>
<A href=#third>第三部分</A><P>
<HR><P>
<A name=first>第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分第一部分<P>
<A href=#top>返回目录</A>
<HR>
<A name=second>第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分第二部分<P>
<A href=#top>返回目录</A>
<HR>
<A name=third>第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分第三部分<P>
<A href=#top>返回目录</A>
<P>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
document.write("<B>本文档中包含 "+document.links.length+" 个链接：
</B><P>")
document.write("<B>本文档中包含 "+document.anchors.length+" 个锚点，它们
```

```
的名称分别是：</B>")
for(i=0;i<document.anchors.length;i++)
{
  document.write(document.anchors[i].name+" ")
}
//-->
</SCRIPT>
</BODY>
</HTML>
```

本示例的显示如图 7.5 所示，它建立了一个目录，使用户可以通过单击目录中的超链接跳转到相应的部分（如果单击“第三部分”这个链接，则显示如图 7.6 所示）；同时可以通过单击“返回目录”超链接回到目录所在位置。

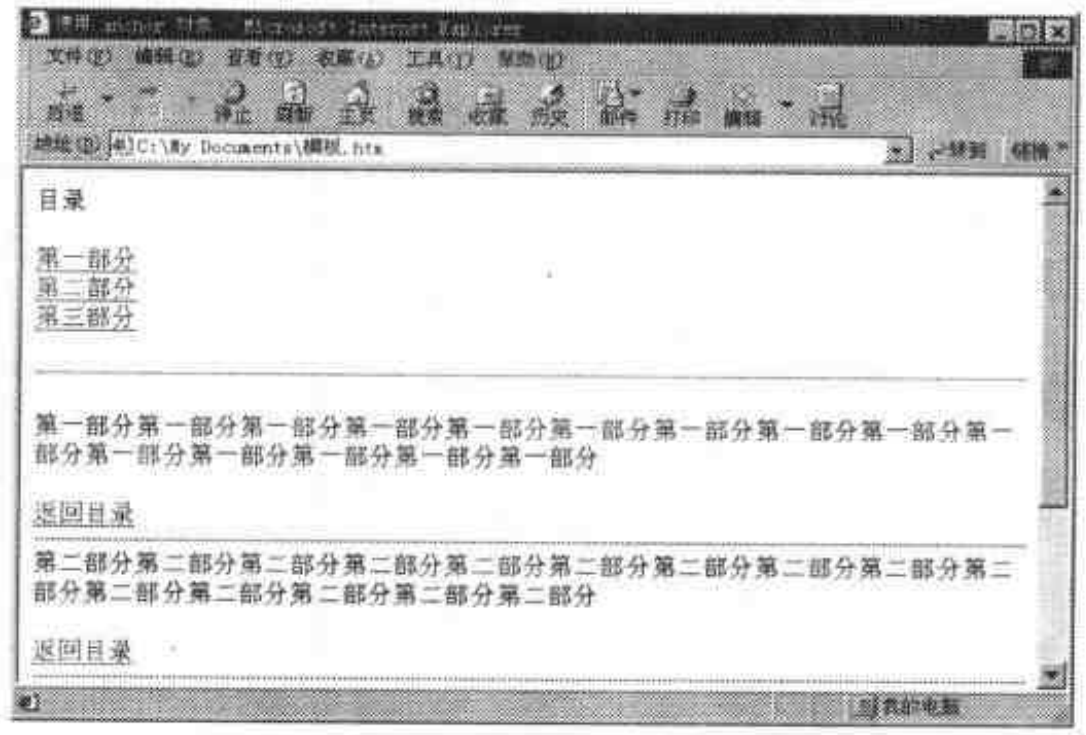


图 7.5　使用锚点

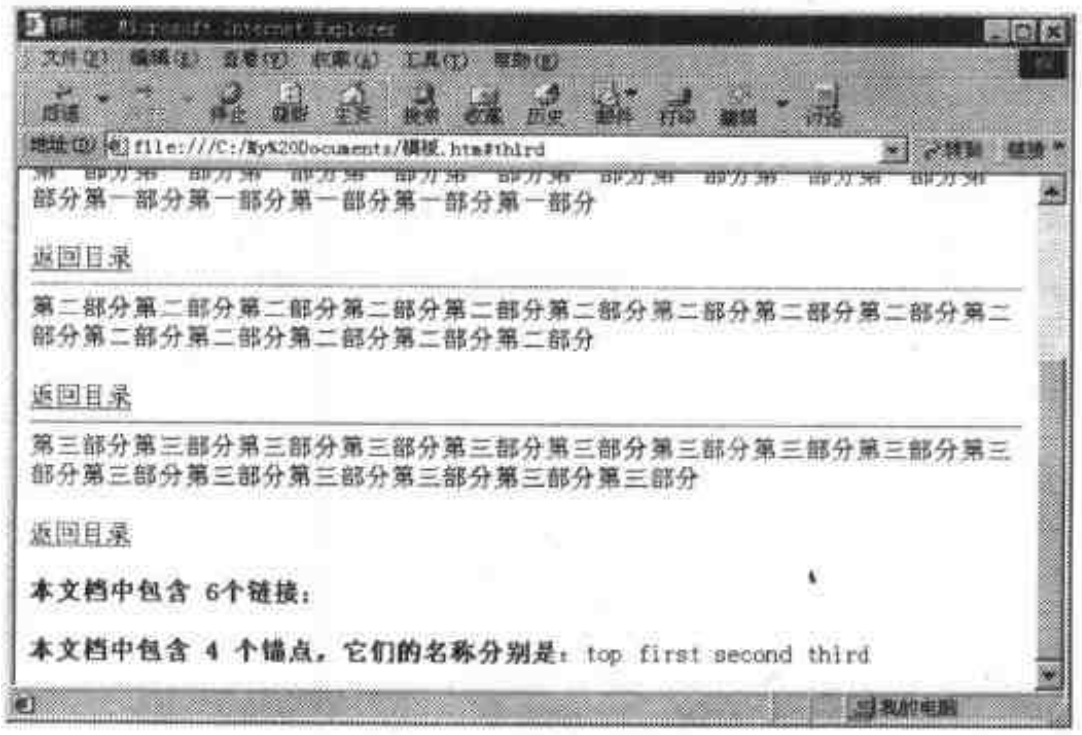


图 7.6　跳转到锚点

7.3 location 对象

7.3.1 属性与方法

location 对象是一种特殊的链接对象，它代表当前窗口中所装入文档的 URL，通常用于重新装入新的文档。

location 对象具有用于描述 URL 各个部分的属性以及两个方法，如表 7.2 所示。

表7.2 location对象的属性和方法

项　目	说　明
hash	表示 URL 中锚点的名称
host	表示 URL 中的主机部分
hostname	表示 URL 中的主机名称部分
href	表示整个 URL
pathname	表示 URL 中的路径名称部分
port	表示 URL 中的端口部分
protocol	表示 URL 中的协议部分
reload()	重新装入当前的 URL。如果使用参数 true，则表示从服务器重新装入
replace(URL)	在浏览器窗口中装入由 URL 指定的新网页，并在历史列表中代替上一个网页的位置，从而使用户不能再用“后退”按钮返回前一文档
search	表示 URL 中的查询字符串部分

7.3.2 示例

实际上，我们在前面的章节中已经多次用到 location 对象，以下再举一个例子说明该对象的用法，代码如下：

```
<HTML>
<HEAD>
<TITLE>location 对象示例</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
function jump1()
{  window.location = document.form1.URL_Textbox.value;
  // 使用 location、window.location 和 location.href 的效果都一样。
}
function jump2()
{  window.location.replace(document.form1.URL_Textbox.value);
```

```
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<H1>请输入一个 Internet 地址……</H1>
<FORM NAME = form1>
<BR>
<INPUT TYPE = TEXT NAME = "URL_Textbox" SIZE = 60>
<BR><BR>
<INPUT TYPE = BUTTON Value = "跳转" onClick = "jump1()">
<INPUT TYPE = BUTTON Value = "替换" onClick = "jump2()">
</FORM>
</DIV>
</BODY>
</HTML>
```

此示例的效果为：用户可以在文本框中输入一个 URL，如图 7.7 所示；输入 URL 之后单击“跳转”按钮则可以跳转到相应页面，如图 7.8 所示（该页面取自 www.163.com 网站），此时单击“后退”按钮，可以回到图 7.7 所示页面；如果在输入 URL 后单击“替换”按钮，也可以跳转到相应页面，但单击“后退”按钮将回退到历史列表中的前一个页面，而不能返回图 7.7 所示页面，这是因为在历史列表中，图 7.8 所示页面已经将图 7.7 所示页面替换。

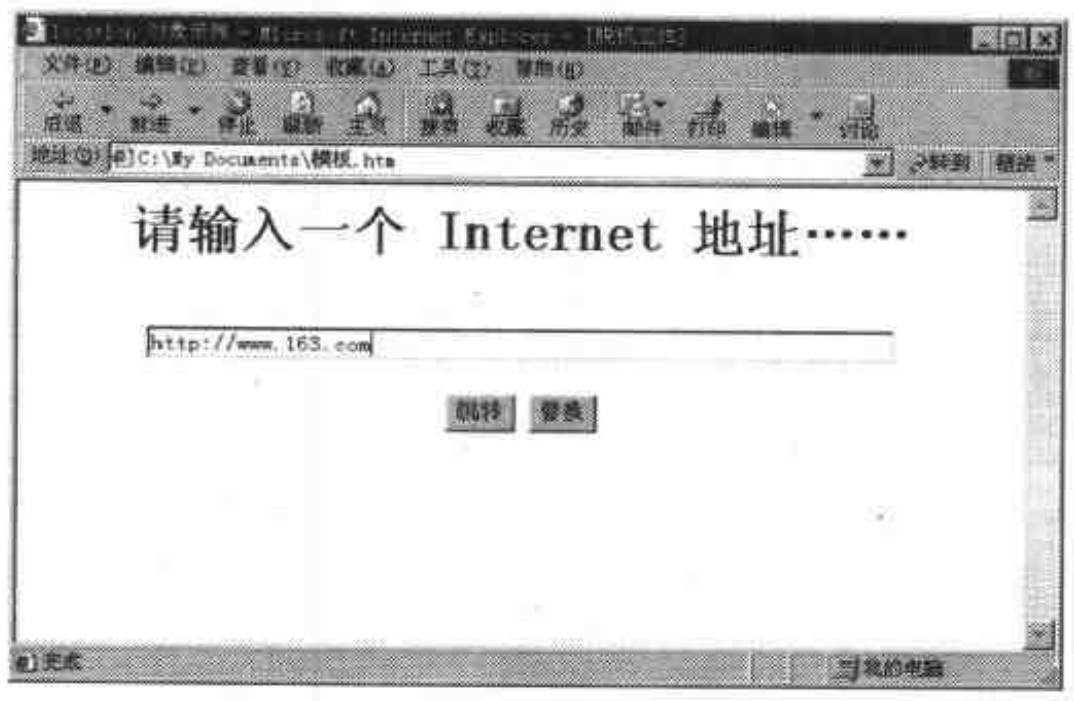


图 7.7　使用 location 对象

图 7.8　跳转到的页面

7.4　history 对象

7.4.1　属性与方法

history 对象是另外一种特殊的链接对象，它代表浏览器的历史列表，使用户可以跟踪窗口中曾经使用过的 URL。

history 对象具有如表 7.3 所示的属性和方法。

表7.3　history对象的属性和方法

项　　目	说　　明
back()	装入历史列表中的上一个页面，相当于单击浏览器上的“后退”按钮
forward()	装入历史列表中的下一个页面，相当于单击浏览器上的“前进”按钮
go(*num*)	装入历史列表中的下第 num 个页面，如果 num 是负数，则装入上第 num 个页面
length	表示历史列表的长度，即历史列表中包含 URL 的个数

7.4.2　示例

以下示例用到了所有 history 的属性与方法，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 history 对象</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
```

```
<!--
function back()
{  window.history.back()  //使用 window.history 与使用 history 效果一样。
}
function forward()
{  window.history.forward()
}
function go()
{  window.history.go(document.form1.num.value)
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<H1>History 对象示例</H1>
  单击"上一页"按钮后退一页；单击"下一页"按钮前进一页；在"跳转多页"按钮右边的文本框中输入一个整数（可以是负数），然后单击该按钮可以跳到多页之前或之后。
<P>
<DIV align=center>
<FORM name=form1>
<INPUT TYPE = BUTTON Value = "< 上一页" onClick = "back()">
<INPUT TYPE = BUTTON Value = "下一页 >" onClick = "forward()"><P>
<INPUT TYPE = BUTTON Value = "<< 跳转多页 >>" onClick = "go()">
<INPUT name=num size=2>
</FORM>
<SCRIPT LANGUAGE = JavaScript TYPE="text/javascript">
<!--
  document.write("<B>当前历史列表的长度为："+history.length+"</B>")
//-->
</SCRIPT>
</DIV>
</BODY>
</HTML>
```

此示例的显示如图 7.9 所示，具体功能可以从该图中看出，不再赘述。

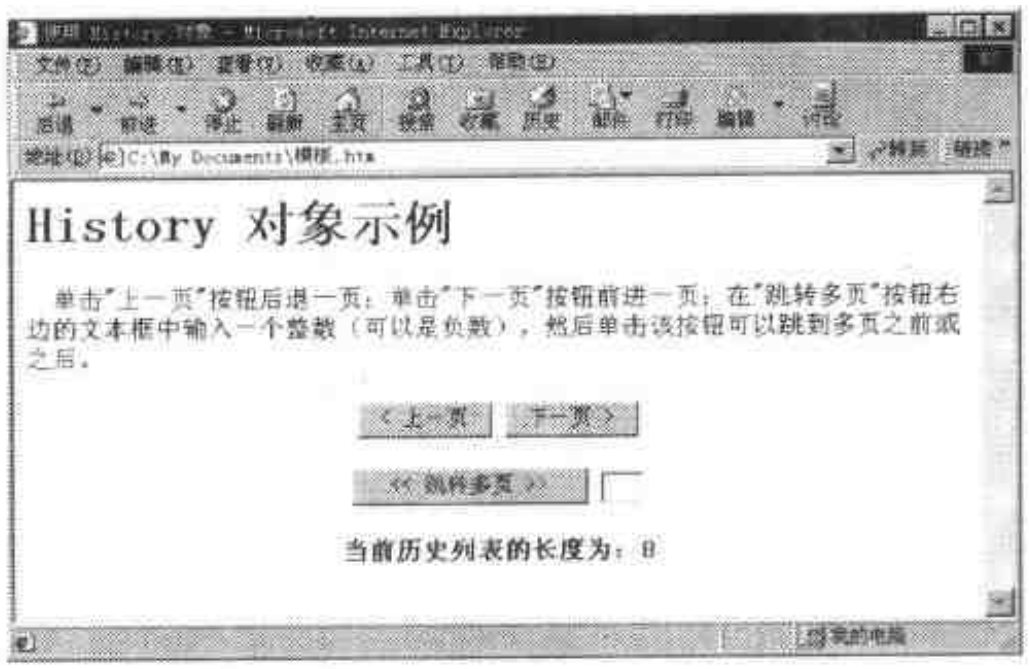


图 7.9 使用 history 对象

7.5 area 对象

7.5.1 属性与事件

area 对象是当用户在网页中使用了 AREA 标记符，并指定了 HREF 属性时创建的。它作为一种链接，可以通过 document.links[]数组访问，并且具有与 link 对象类似的属性与事件，如表 7.4 所示。

表7.4 area对象的属性和事件

项 目	说 明
hash	表示 URL 中锚点的名称
host	表示 URL 中的主机部分
hostname	表示 URL 中的主机名称部分
href	表示一个链接的完整 URL
onClick	在映射区域上单击鼠标时触发
onDblClick	在映射区域上双击鼠标时触发
onKeyDown	在映射区域上按下某键时触发
onKeyPress	在映射区域上按住某键时触发
onKeyup	在映射区域上松开某键时触发
onMouseDown	在映射区域上按下鼠标键时触发
onMouseOut	将鼠标指针从映射区域上移开时触发
onMouseOver	将鼠标指针移到映射区域上时触发
onMouseUp	在映射区域上松开鼠标键时触发
pathname	表示 URL 中的路径名称部分
port	表示 URL 中的端口部分
protocol	表示 URL 中的协议部分
search	表示 URL 中的查询字符串部分
target	表示超链接结果的目标窗口，对应于 AREA 标记符中的 target 属性

7.5.2 客户端图像映射

实际上，AREA 标记符并不是象 A 标记符那样单独使用的，它只有用在 MAP 标记符内构成客户端图像映射时才有意义。

1. 什么是客户端图像映射

所谓客户端图像映射是指网页上的一幅图像，其中包括多个区域，每个区域对应于一个超链接，单击任意区域可以跳转到相应的页面。

使用客户端图像映射应首先用 MAP 标记符定义映射区域，然后在 IMG 标记符中使用 USEMAP 属性与所定义的映射区域关联。在定义映射区域时，每个区域都对应于一个 AREA 标记，该标记具有 shape（说明映射区域的形状，取值为 rect | circle | poly，分别表示长方形、圆形和多边形）、coords（表示映射区域的坐标，根据 shape 的不同而不同）、href（表示单击映射区域时要跳转到的 URL）和 target（表示超链接的目标窗口）等常用属性。具体如何使用客户端图像映射，请参见以下示例。

2. 导航条示例

客户端图像映射的一个最常见的应用就是作为页面导航条。例如，在图 7.10 中（该页面取自 www.263.net），页面最上面的灰色导航条就是用图像映射实现的（请注意鼠标指针和状态栏）。

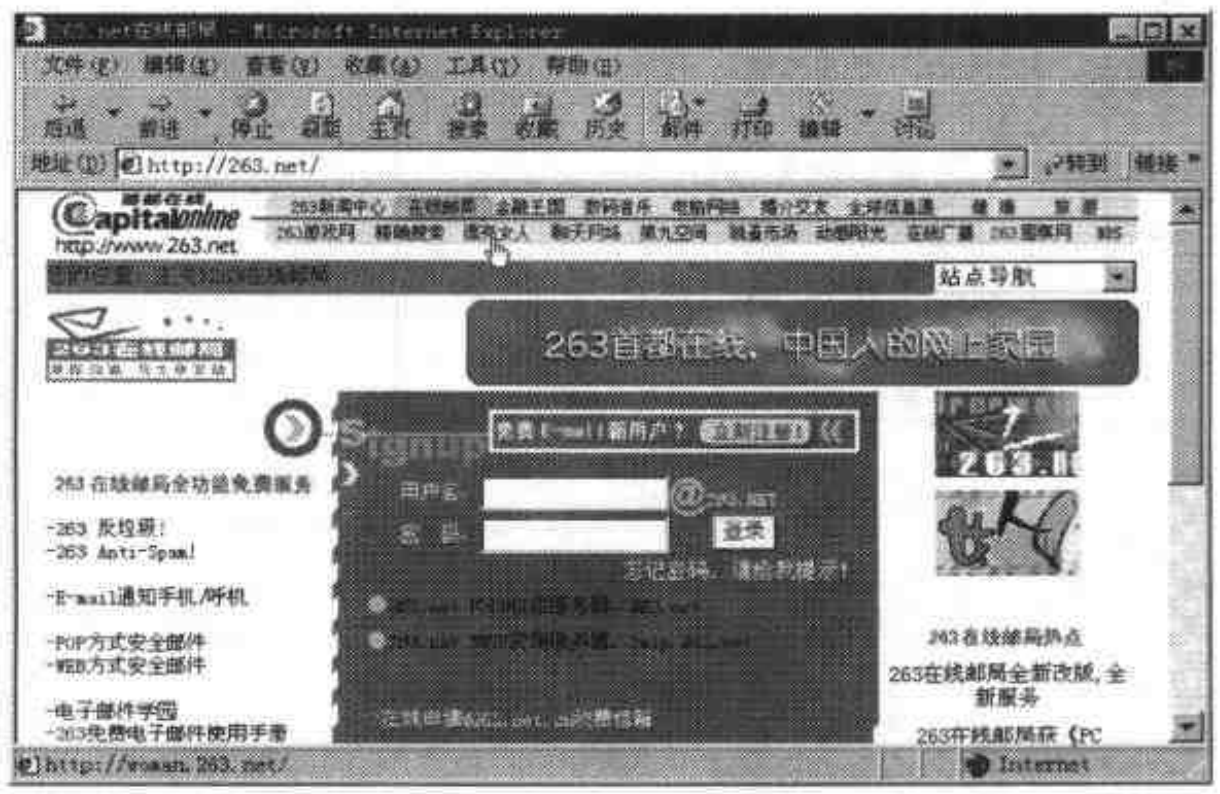


图 7.10 图像映射导航条

以下代码用于实现该导航条：

```
<HTML>
<HEAD>
  <TITLE>导航条</TITLE>
</HEAD>
<BODY>
<IMG  src="images/wizard.gif"  width="622"  height="38"  border="0"
usemap="#navBar">
  <!-- usemap 属性的值必须等于 # 号加上 MAP 中 name 属性的值 -->
<MAP name="navBar">
  <AREA shape="rect" coords="585,21,617,35" href="http://bbs.263.net">
  <AREA shape="rect" coords="18,19,88,34" href="http://games.263.net">
  <AREA shape="rect" coords="517,20,582,35" href="http://gogame.263.net">
  <AREA shape="rect" coords="210,19,267,33" href="http://chat.263.net">
  <AREA shape="rect" coords="150,19,208,34" href="http://woman.263.net">
  <AREA shape="rect" coords="454,20,509,35" href="http://real.263.net
  /enjoy/direct/music2.ram">
  <AREA shape="rect" coords="394,19,448,34" href="http://sport.263.net">
  <AREA shape="rect" coords="330,19,386,34" href="http://flea.263.net">
  <AREA shape="rect" coords="269,19,329,34" href="http://www.topcool
  .net">
  <AREA shape="rect" coords="88,19,148,34" href="http://210.78.145.252">
  <AREA shape="rect" coords="503,2,545,15" href="http://health.263.net/">
  <AREA shape="rect" coords="27,1,101,15" href="http://news.263.net">
  <AREA shape="rect" coords="420,1,486,15" href="http://pca.263.net">
  <AREA shape="rect" coords="357,2,415,17" href="http://love.263.net">
  <AREA shape="rect" coords="296,2,351,17" href="http://it.263.net">
  <AREA shape="rect" coords="231,1,290,16" href="http://music.263
  .net/">
  <AREA shape="rect" coords="172,1,228,16" href="http://stock.263
  .net">
  <AREA shape="rect" coords="555,1,602,16" href="#">
</MAP>
</BODY>
</HTML>
```

执行以上代码的效果如图 7.11 所示。

从这段代码可以看出，制作图像映射关键是要确定图像上的映射区域，也就是指定 AREA 标记符的 shape 和 coords 属性。对于简单的图像映射，可以直接使用图形图像处理

软件获取每个区域的坐标；对于比较复杂的图像映射，通常应借助于专门的图像映射编辑器（如"Map This"）或网页编辑软件（如 FrontPage2000、Dreamweaver3.0 等）。

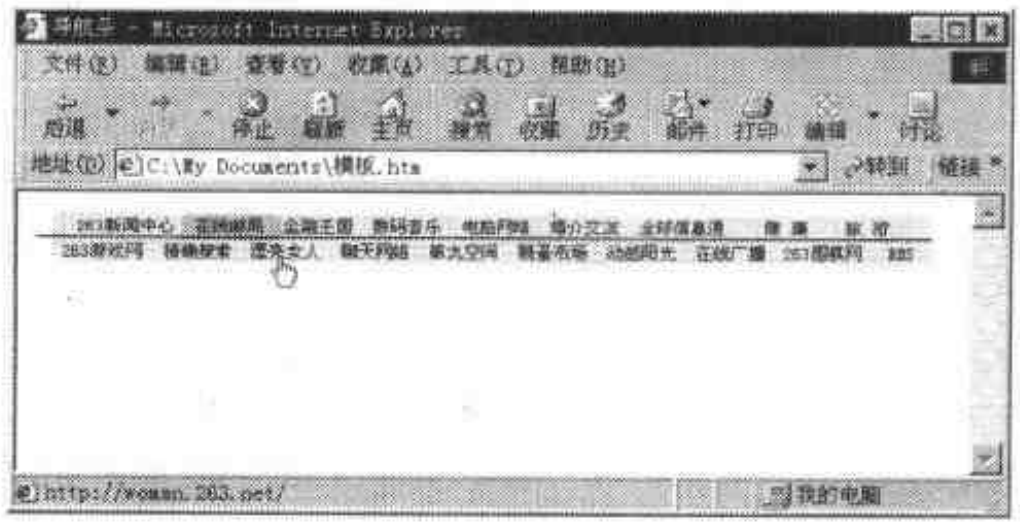


图 7.11　分离出的图像映射导航条

7.6　image 对象

7.6.1　创建 image 对象

当用户在网页中使用 IMG 标记符插入图像时，即创建出了一个 image 对象。image 对象可以通过 document.images[]数组访问，例如使用 document.images[0]即可访问页面上的第一个图像。

image 对象还可以由对象构造函数 Image() 来创建。Image() 构造函数有两个可选参数：width 和 height，分别用于指定图像的像素宽度和像素高度。如果这些参数大于或小于图像的实际尺寸，则图像将被伸缩到设定的尺寸。例如，myImage=new Image(80,64) 生成一个 80×64 的 image 对象。构造出 image 对象之后，应使用对象的 src 属性指定图像的来源，如下所示：

```
myImage = new Image();
myImage.src = "./images/myImg.gif"
```

用这种方式创建的 image 对象可以预先下载到浏览器并动态缓存到浏览器的缓冲区，当需要时再装入到页面。

注意：用 Image() 构造函数创建的 image 对象不能用 document.images[] 数组访问。

7.6.2 属性与事件

1. 属性与事件列表

image 对象的属性和事件如表 7.5 所示。

表7.5 image对象的属性和方法

项 目	说 明
border	表示图像边框的宽度（以像素为单位），对应于 IMG 标记符的 BORDER 属性
complete	表示图像是否完全装入的布尔属性。如果完成装入，此属性值为 true；如果装入失败或发生错误，此属性值为 false
height	表示图像的高度（以像素为单位），对应于 IMG 标记符的 HEIGHT 属性
hspace	表示图像在水平方向上与其他相邻对象的距离（以像素为单位），对应于 IMG 标记符的 HSPACE 属性
lowsrc	表示用于低分辨率显示器的备用图像（通常具有较低分辨率），对应于 IMG 标记符的 LOWSRC 属性。在高分辨率显示器上装入图像时，先装入此低分辨率图像，然后用实际图像替换
name	表示图像的名称，对应于 IMG 标记符的 NAME 属性
onAbort	图像装入中断时触发
onError	图像装入出错时触发
onKeyDown	在图像上按下某键时触发
onKeyPress	在图像上按住某键时触发
onKeyup	在图像上松开某键时触发
onLoad	在图像装入完成时触发
src	表示图像的 URL，对应于 IMG 标记符的 SRC 属性。动态改变此属性时可以动态载入图像
vspace	表示图像在垂直方向上与其他相邻对象的距离（以像素为单位），对应于 IMG 标记符的 VSPACE 属性
width	表示图像的宽度（以像素为单位），对应于 IMG 标记符的 WIDTH 属性

2. 示例 1

本示例在网页中显示出了 image 对象的各种属性值，代码如下：

```
<HTML>
<HEAD><TITLE>image 对象属性示例</TITLE></HEAD>
<BODY>
<DIV align=center>
  <IMG  name=MMPic  src=./images/MM.jpg  border=2  width=120  height=80
lowsrc=./images/lowsrc.gif>
</DIV>
<P><HR width=50%>以上图片的属性值如下：<P>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
```

```
<!--
  document.write("<B>border = "+document.images[0].border+"</B><BR>")
  document.write("<B>complete = "+document.images[0].complete
  +"</B><BR>")
  document.write("<B>height = "+document.images[0].height+"</B><BR>")
  document.write("<B>hspace = "+document.images[0].hspace+"</B><BR>")
  document.write("<B>lowsrc = "+document.images[0].lowsrc+"</B><BR>")
  document.write("<B>name = "+document.images[0].name+"</B><BR>")
  document.write("<B>src = "+document.images[0].src+"</B><BR>")
  document.write("<B>vspace = "+document.images[0].vspace+"</B><BR>")
  document.write("<B>width = "+document.images[0].width+"</B>")
//-->
</SCRIPT>
</BODY>
</HTML>
```

此示例的显示效果如图 7.12 所示。

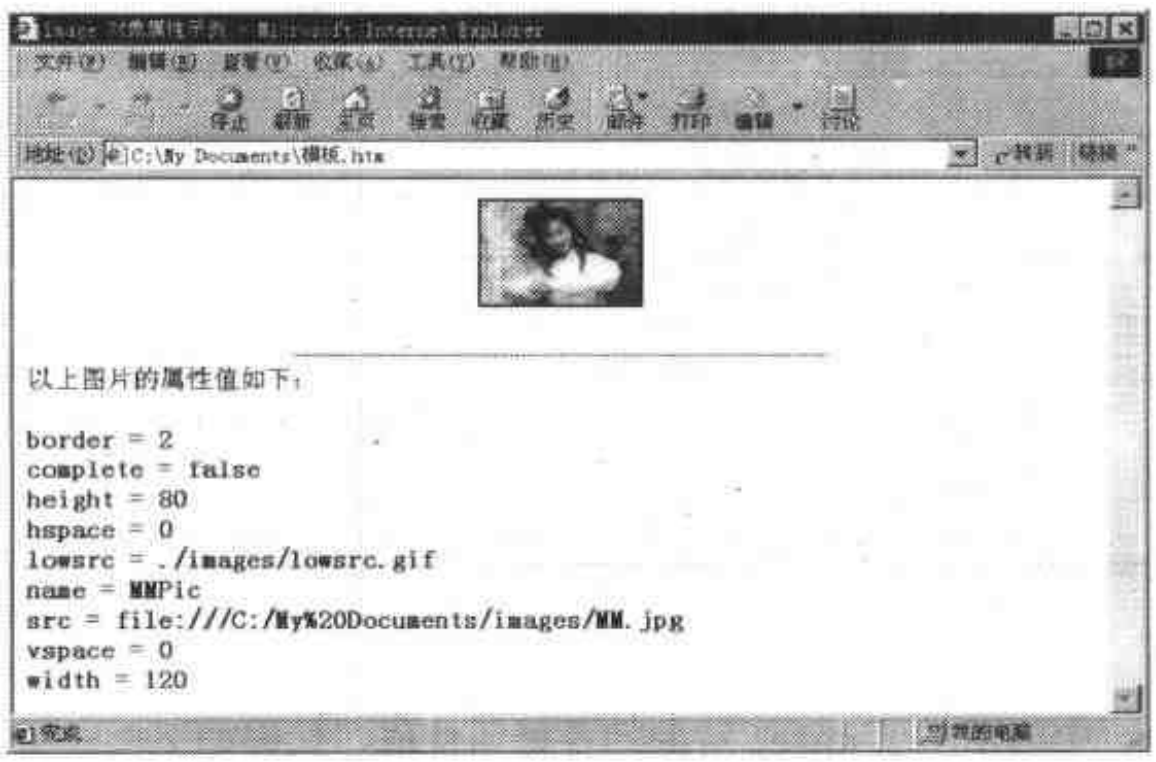


图 7.12　显示 image 对象的属性值

3. 示例 2

本示例通过动态改变 image 对象的 src 属性制作出常见的翻滚图效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>翻滚图</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
```

```
<!--
function loadImages()
{
image1=new Image(320,240);  //image1 是全局变量
image2=new Image(320,240);  //image2 是全局变量
image1.src="./images/cactus.jpg";
image2.src="./images/sun.jpg"
}

function imgOver()
{  //当鼠标指针移进时，切换到第二幅图像。
  document.images[0].src=image2.src;
}

function imgOut()
{  //当鼠标指针移出时，切换回第一幅图像。
  document.images[0].src=image1.src;
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad=loadImages()>   <!-- 将图像下载到缓冲区 -->
<DIV align=center>
<H2>请将鼠标指针移动到图像上……</H2>
<A HREF=javascript:void(0)
   onMouseOver = "imgOver()"
   onMouseOut = "imgOut()">
    <IMG NAME = img1 SRC = "./images/cactus.jpg" WIDTH = 320 HEIGHT = 240>
</A>
</DIV>
</BODY>
</HTML>
```

本示例的效果为：网页上显示一幅图像，如图 7.13 所示；当用户将鼠标指针移动到该图像上时，图像动态切换，如图 7.14 所示；当鼠标指针移出图像范围时，切换回原始图像。

图 7.13　初始状态

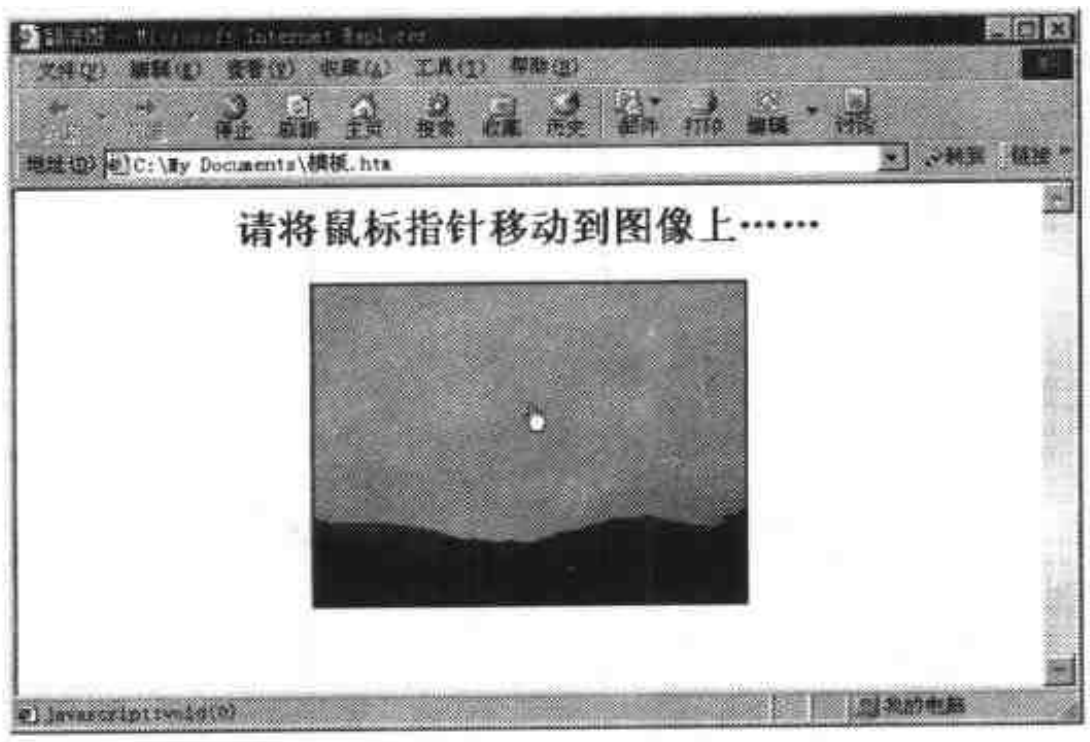


图 7.14　切换到另一幅图像

7.6.3　制作 JavaScript 动画

动画在网页上显然是一种吸引浏览者的重要手段。常见的动画制作方法有三种：一是使用 GIF 编辑软件将多幅图像组织成一幅 GIF 动画；二是使用 Java 语言编写动画 JavaApplet；三是使用 JavaScript 控制多幅图像之间的切换，生成动画效果。本节用几个示例说明如何制作 JavaScript 动画。

1．示例 1

动画的一种常见应用就是广告横幅（banner），以下示例说明如何用 JavaScript 实现动

态变换的广告效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>广告横幅示例</TITLE>
<SCRIPT LANGUAGE = "JavaScript" TYPE="text/javascript">
<!--
URLs=new      Array('http://www.8848.com",      "http://www.2911.com",
"http://www.yl.com");

function loadImages()
{
image1=new Image(360,52);
image2=new Image(460,52);
image3=new Image(460,52);
image1.src="./images/8848.jpg";
image2.src="./images/2911.jpg"
image3.src="./images/yinlian.jpg"
}

function changeAd()
{
 randomIndex=Math.round(Math.random()*URLs.length)%3    // 随机生成 0、
1、2
 document.links[0].href=URLs[randomIndex];
 randomIndex++;
 eval("imgStr=image"+randomIndex+".src");
 document.images[0].src=imgStr;
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="loadImages();setInterval('changeAd()',1500)">
<DIV align=center>
  <A href="http://www.8848.com">
    <IMG src="./images/8848.jpg">
  </A>
  <P><HR><HR>
  <H3>其他网页内容……</H3>
</DIV>
```

```
</BODY>
</HTML>
```

本示例的效果为：每隔 1.5 秒随机显示一幅广告图片，同时图片上的超链接目标随着更换，如图 7.15 和图 7.16 所示（请注意状态栏中显示的超链接目标）。

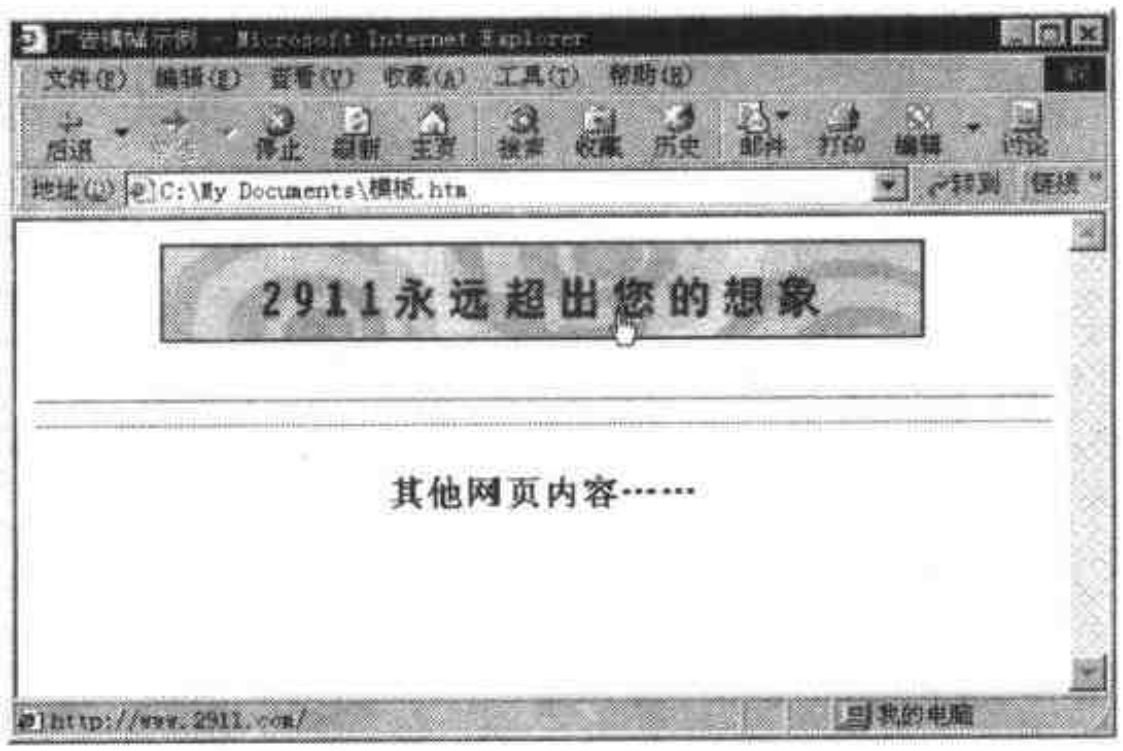


图 7.15　广告图片之一

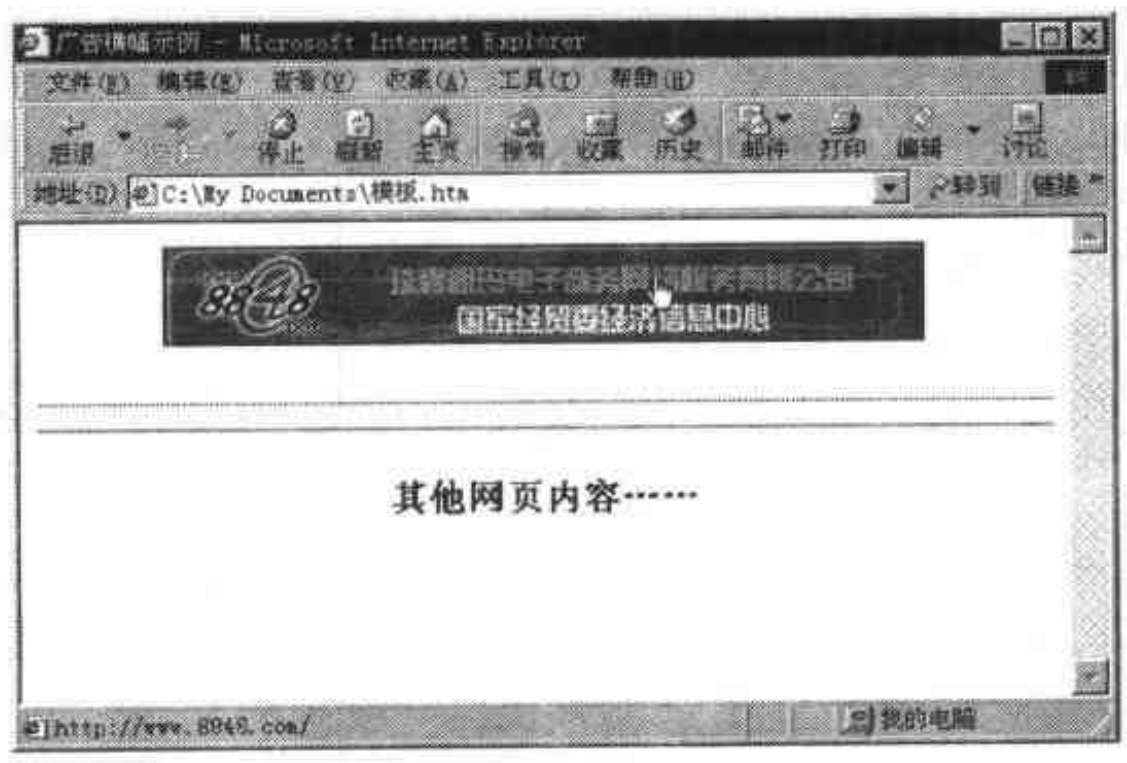


图 7.16　广告图片之二

2. 示例 2

动画的本质就是按照一定次序连续显示一系列图片，以下示例用 JavaScript 实现了这一效果，代码如下：

```
<HTML>
```

```
<HEAD>
<TITLE>JavaScript 动画</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function loadImage()
{
imageSrc=new Array(4)  //创建一个数组，存放 image 对象。
for(i=0;i<4;++i)
{
  imageSrc[i]=new Image()
  imageSrc[i].src="./images/page"+i+".gif"
}
}
nextImage=1  //nextImage 是全局变量
}

function startAnimation()  //启动动画
{
  interval=setInterval('animate()',200)
}

function animate()
{
  i=nextImage++
  nextImage%=4
  document.genie.src=imageSrc[i].src  //动态更改图片显示
}

function pauseAnimation()
{
  clearInterval(interval);  //暂停动画
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad=loadImage()>
<DIV align=center>
<IMG NAME="genie" SRC="./images/page0.gif">
<BR>
<FORM>
```

```
<INPUT TYPE='BUTTON' NAME='start' VALUE='开始锻炼' ONCLICK=
'startAnimation()'>
<INPUT TYPE='BUTTON' NAME='pause' VALUE='我要歇！' ONCLICK=
'pauseAnimation()'>
</FORM>
</DIV>
</BODY>
</HTML>
```

本示例的效果为：当单击“开始锻炼”按钮时，动画开始，如图 7.17 所示；如果单击“我要歇!”按钮，则动画暂时停止；当再次单击“开始锻炼”按钮时可以重新启动。

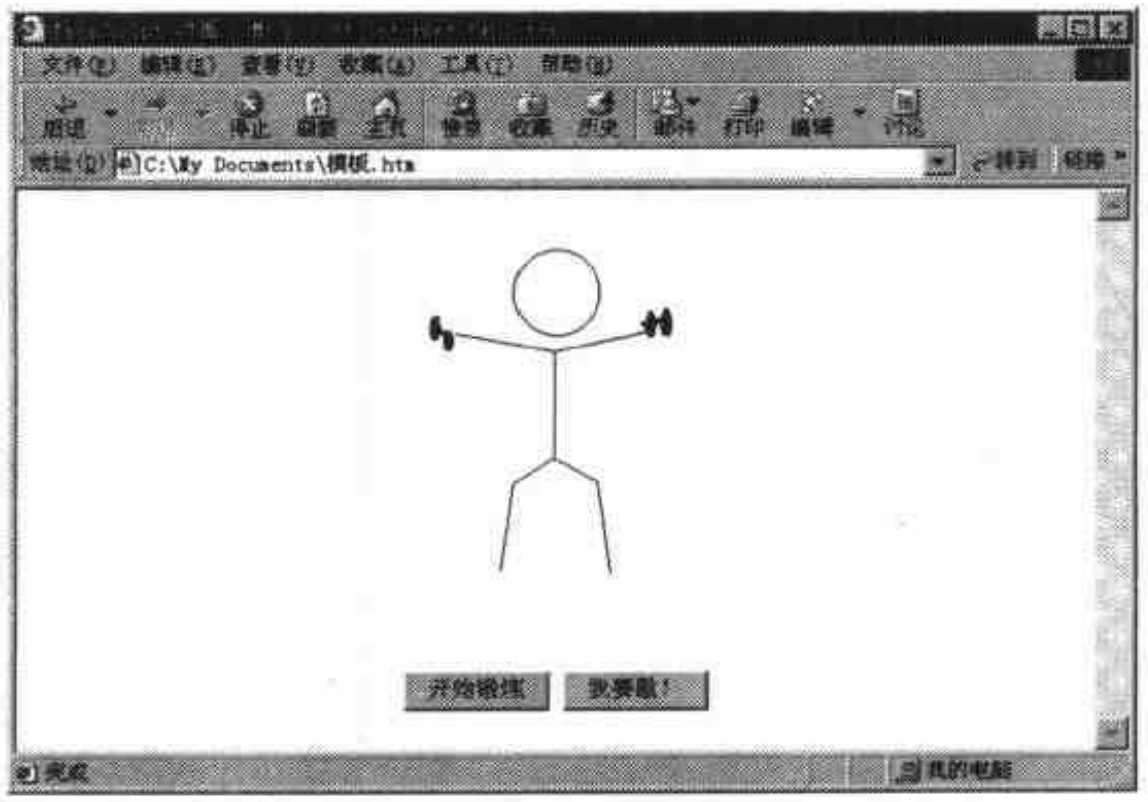


图 7.17　动画示例

实际上，本示例的动画是通过将如图 7.18 所示的 4 个原始图片连续显示获得的。如果要获得更逼真、过渡更平滑的动画效果，应该使用更多的原始图片，图片与图片之间的差异应更小，同时设置更短的时间间隔（interval）。

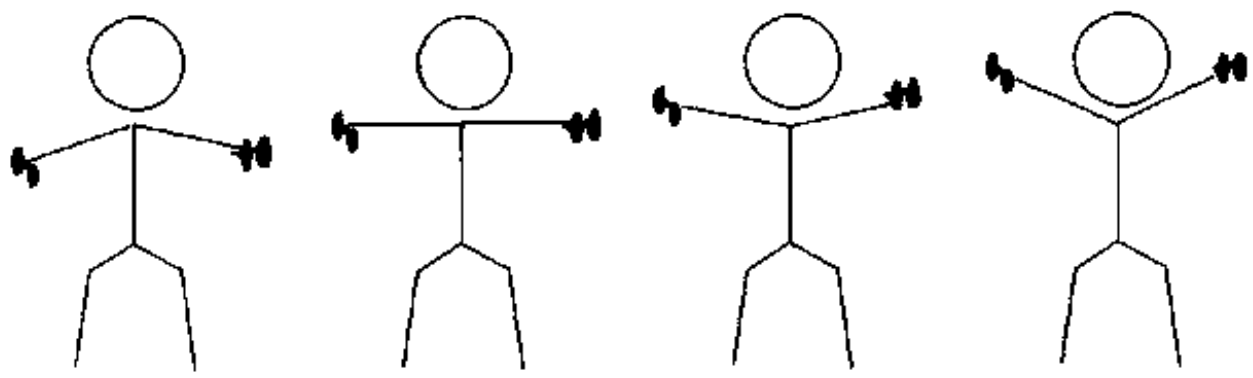


图 7.18　原始图片

3. 示例3

在 JavaScript 中使用 image 对象还可以创建出移动的效果，如以下示例所示：

```
<HTML>
<HEAD>
<TITLE>移动效果</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function loadImages()
{
 hand = new Image()
 blank = new Image()
 hand.src = "./images/hand.gif"
 blank.src = "./images/blank.gif"
 total = 5
 current = 4
}

function scrollImages()
{
 window.document.images[current].src = blank.src
 current = (current + 1) % total
 window.document.images[current].src = hand.src
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="loadImages(); setInterval('scrollImages()',400)">
<DIV align=center>
  <!--以下用空白图像作为占位符，然后通过动态更改获得移动效果-->
<IMG SRC="./images/blank.gif">
<IMG SRC="./images/blank.gif">
<IMG SRC="./images/blank.gif">
<IMG SRC="./images/blank.gif">
<IMG SRC="./images/blank.gif">
<A HREF="javascript:void(0)">
  <!--此超链接不作任何跳转-->
  <FONT size=+1 face="楷体_GB2312">快来看呀！</FONT>
</A>
<P><HR width=50%><P>
```

```
<H2>网页内容</H2>
</DIV>
</BODY>
</HTML>
```

本示例的效果如图 7.19 所示，在页面可以看到一个“手”的图案在动态移动。（由于本示例所用方法的局限，获得的动画效果有些“跳”，如果要获得更连续的移动效果，应使用动态更改对象位置的方式，具体请参见本书第 8 章 8.4.1 节中的内容）

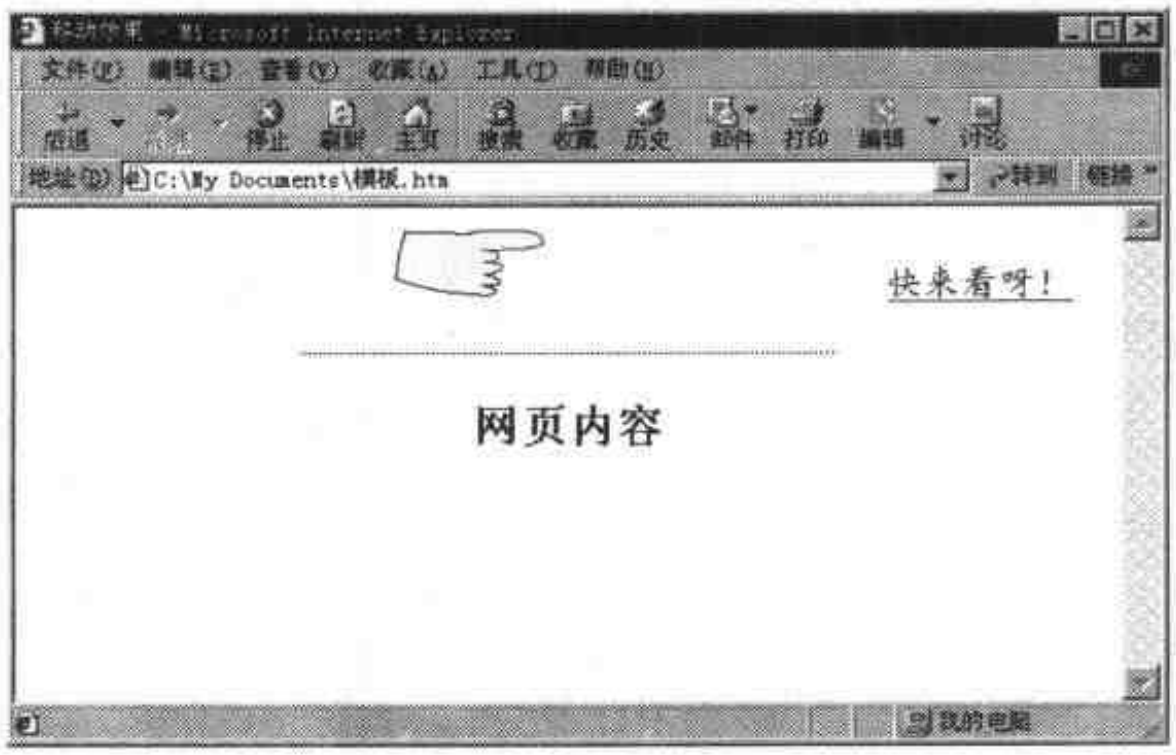


图 7.19　移动的图片

实际上，可以将本示例与上一个示例用到的方法结合起来，从而获得边移动、边产生动画的效果，请读者自行尝试。

第8章 DHTML 基础

第8章 DHTML 基础

DHTML（动态 HTML）是近年来兴起的一项新技术，用于实现具有动态特征和交互式功能的网页。DHTML 的本质就是使用脚本语言访问 HTML 元素对象和 CSS 属性，通过动态改变这些对象和属性，从而获得需要的效果。

本章介绍 DHTM 的基础知识，主要内容包括：

- DHTML 概述
- CSS 基础
- CSS 属性
- 过滤器属性

8.1 DHTML 概述

首先我们看一看微软官方站点 microsoft.com 中的一个重要界面特性，如图 8.1 所示——当浏览者将鼠标指针移动到页面导航条上时，会动态地弹出一个菜单，在该菜单中移动鼠标，所指向的菜单项变为红色显示；如果将鼠标指针移出菜单所在范围，则菜单自动隐藏；如果将鼠标指针移动到导航条上另外一个区域，则会弹出另外一个菜单。

这种效果非常类似于 Windows 应用程序的特性，即通过图形化的界面为用户提供尽可能多的功能。实际上，采用这种方式可以使同一个页面上包含更多的信息，对于 microsoft.com 这样庞大的站点来说十分有用。

要实现这种效果，单纯依靠 HTML 和 JavaScript 已经无法实现，必须采用新的技术——动态 HTML。

动态 HTML（Dynamic HTML，简称 DHTML）并不是一门新的语言，它只是 HTML、CSS 和客户端脚本的一种集成。DHTML 建筑在原有技术的基础上，可分为三个方面：

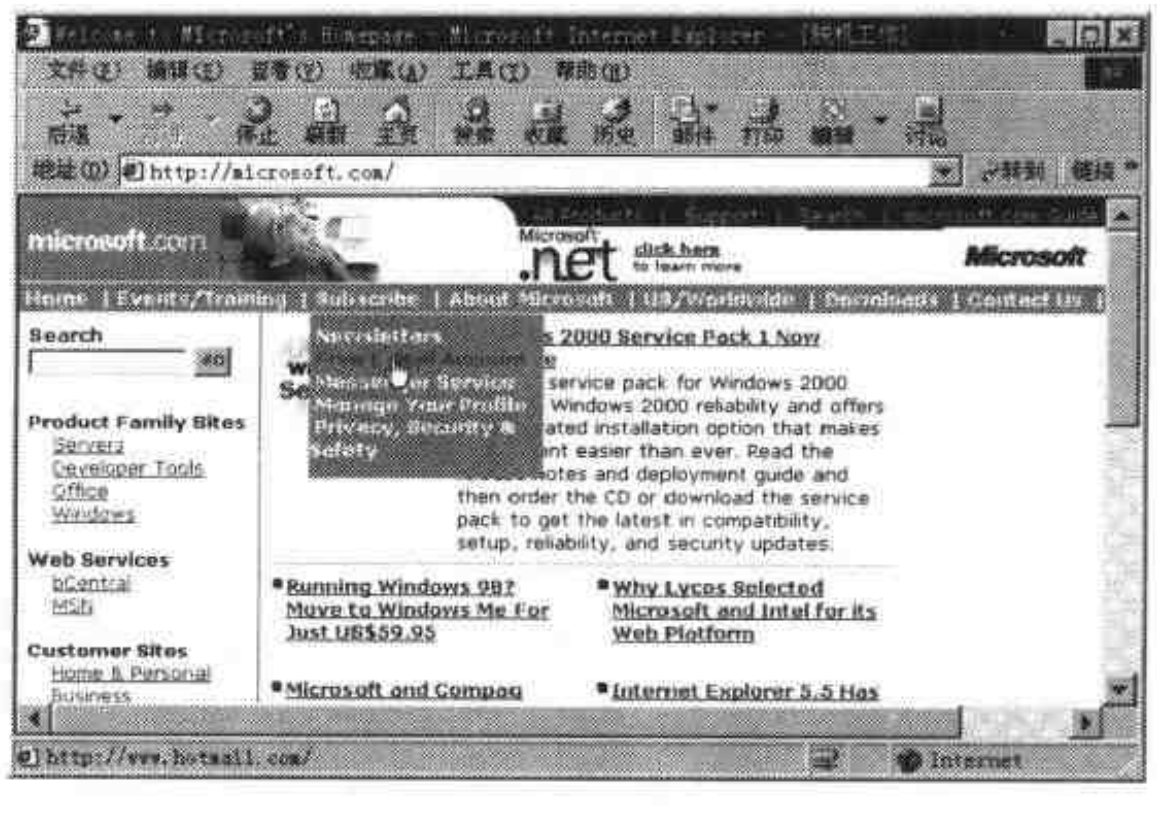


图 8.1　动态 HTML 示例

- HTML

也就是页面中的各种页面元素对象，它们是实际被动态操纵的内容。

- CSS

CSS 是 Cascading Style Sheet 的缩写，一般译为层叠样式表。CSS 是一种格式化网页的标准方式，它扩展了 HTML 的功能。CSS 属性也是动态操纵的内容，从而获得动态的格式效果。

- 客户端脚本

客户端脚本（例如 JavaScript）是 DHTML 技术的关键，它实际操纵 Web 页上的 HTML 和 CSS。

使用 DHTML 技术，可使网页设计者创建出能够与用户交互并包含动态内容的页面（例如图 8.1）。实际上，DHTML 使网页设计者可以动态操纵网页上的所有元素——甚至是在这些页面被装载以后。利用 DHTML，网页设计者可以动态地隐藏或显示内容、修改样式定义、激活元素以及为元素定位。DHTML 还可使网页设计者在 Web 页上显示外部信息，方法是将元素捆绑到外部数据源（如文件和数据库）上。

所有这些功能均可用浏览器完成而无需请求 Web 服务器，同时也无需重新装载 Web 页。这是因为一切功能都包含在 HTML 文件中，随着对网页的请求面一次性下载到浏览器端。这种方式改善了用户的操作感觉同时又缩短了下载的时间，降低了网络传输量以及 Web 服务器的负载。

注意： 虽然 DHTML 能实现很强大的功能，但由于它是一个新技术，因此只在最新的浏览器中得到支持，例如 Internet Explorer 4.0/5.0、Netscape Navigator 4.0 等。另外，对于不同的浏览器，DHTML 的实现往往是不同的，需要编写不同的代码（本书均以 IE 为例）。所以，网页设计者需要仔细地编写 DHTML，并确保彻底测试 Web 页，以便使自己的网页更专业和稳定（实际上，微软站点中使用的 DHTML 特性就可用于多种浏览器）。

8.2 CSS 基础

要使用 DHTML 技术，首先必须掌握 CSS 技术。从本节开始将简要介绍 CSS 技术的要点，为尚未掌握该技术的读者提供一个参考。如果读者已经熟练地掌握了 CSS，那么可以直接跳到下一章。

8.2.1 样式定义

HTML 的本质就是通过标记符为网页内容定义一定的格式，例如，包含在 H1 标记符中的内容以粗体和大字号显示，包含在 I 标记符中的内容以斜体显示等。然而，单纯用 HTML 标记符设置格式，所能达到的效果是非常有限的，此时就要用到 CSS 技术。

CSS（Cascading Style Sheet），也就是层叠样式表，是一种格式化网页的标准方式，它就颜色、字体、边框、定位等格式提供了多种属性，这些属性均可适用于 HTML 标记符，从而使网页的格式设置功能大大增强。

例如，以下一个简单示例显示了 CSS 的功能，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 CSS 设置格式</TITLE>
  <STYLE>
    P { text-indent:0.75cm; font-family:楷体_GB2312 }  /*定义一个样式，使 P 标记符的内容以 0.75 cm 的缩进和“楷体”字体显示。*/
  </STYLE>
</HEAD>
<BODY>
  <P>第一段文本第一段文本第一段文本第一段文本第一段文本第一段文本第一段文本第一段文本第一段文本第一段文本</P>
  <P>第二段文本第二段文本第二段文本第二段文本第二段文本第二段文本第二段文本第二段文本第二段
```

```
    文本第二段文本</P>
    </BODY>
    </HTML>
```

这段代码的显示效果如图 8.2 所示，所有包含在 P 标记符中的内容都以 0.75cm 的首行缩进和“楷体”字体显示。

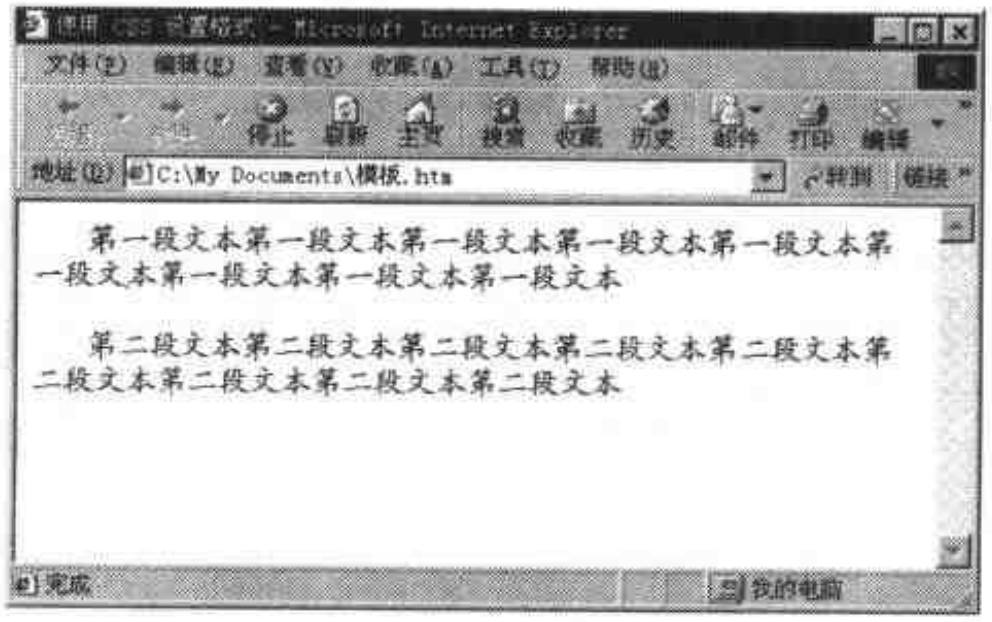


图 8.2　CSS 示例

在这个示例中，我们对 P 标记符定义了特定的 CSS 属性，使包含在该标记符中的内容都具有一致的格式。实际上，样式定义并不局限于 HTML 标记符，它的基本格式如下：

```
selector { property1:value1; property2:value2; ... ...}
```

其中 selector 表示需要应用样式的内容，property 表示由 CSS 定义的标准样式属性（请参见第 8.2.3 节），value 表示样式属性的值。

以下是几种常用的 selector 类别：

- HTML 标记符
- 具有上下文关系的 HTML 标记
- 用户定义的类（class）
- 用户定义的 ID
- 虚类

1. HTML 标记符 selector

HTML 标记符是最典型的 selector 类型，网页设计者可以为某个或某些具体的 HTML 元素应用样式声明。对于不同的标记符 selector，我们可以采用编组的方式简化样式定义。

例如，如果 3 个样式定义如下：

```
H1 {color:#ff0000}
H2 {color:#ff0000}
H3 {color:#ff0000}
```

则可以将其转换成编组样式，用逗号将不同的 selector 分开，如下所示：

```
H1,H2,H3 {color:#ff0000}
```

2. 具有上下文关系的 HTML 标记符 selector

如果需要为位于某个标记符内的标记符设置特定的样式规则，则应将 selector 指定为具有上下文关系的 HTML 标记。例如，如果只想使位于 H1 标记符内的 B 标记符具有特定的属性，则应使用以下格式：

```
H1 B{color:blue}  /* 注意 H1 和 B 之间以空格分隔 */
```

这表示只有位于 H1 标记符内的 B 元素具有指定样式，而其他 B 元素不具有该样式。实际上，这种嵌套关系可以有多层，不过通常仅用一层。

3. 用户定义的类 selector

可以使用类（Class）来为单一 HTML 标记符创建多个样式。要想将一个类包括到样式定义中，可将一个句点和一个类名称添加到 selector 后，如下所示：

```
selector.classname {property: value;  ... ...}
```

可以使用任何名称命名类，但通常应使用有具体含义的名称。例如，如果需要在网页的三处使用 H1 标记符，而且每处的文本具有不同的颜色，此时可以定义以下类样式：

```
H1.color_red{color:red}
H1.color_yellow{color:yellow}
H1.color_blue{color:blue}
```

然后在网页中需要使用该类处用 CLASS 属性引用这些类，如下所示：

```
<H1 CLASS="color_red">此标题为红色。</H1>
<H1 CLASS="color_yellow">此标题为黄色。</H1>
<H1 CLASS="color_blue">此标题为蓝色。</H1>
```

此时如果使用了 H1 标记符但没有使用相应的 CLASS 属性，则不应用所定义的样式。实际上，不仅可以为某个或某些标记符定义类，还可以定义应用于所有标记符的类，

此时直接用句点后跟类名即可，如下所示：

```
.classname{property:value; ... ...}
```

例如，可以定义一个类：

```
.red {color:red}
```

然后在需要引用该类的任意标记符内使用 class 属性，以便所有引用该类的标记符都可以采用所定义的样式。在定义了以上的 red 类后，就可以用以下方式引用它：

```
<P class="red"> 本行文字为红色。</P>
<H1 class="red"> 本标题为红色。</H1>
```

4. 用户定义的 ID selector

当网页设计者在整个网页或几个页面上有许多想以相同样式显示的标记符时，除了使用.classname 的方式定义一个类样式以外，还可以使用 ID 定义样式。

要将一个 ID 包括在样式定义中，应用一个井号（#）作为 ID 名称的前缀，如下所示：

```
#IDname{property:value ... ...}
```

定义了 ID 样式后，需要在引用该样式的标记符内使用 id 属性。例如，可以定义一个 ID 样式如下：

```
#red {color:red}
```

然后可以在若干不同的 HTML 标记符中使用该样式规则，如下所示：

```
<P id="red"> 本行文字为红色。</P>
<H1 id="red"> 本标题为红色。</H1>
```

注意： 使用.classname 和使用#IDname 这两种方式在效果上并没有区别，但最好只使用其中之一，以免造成混淆。

5. 虚类 selector

对于 A 标记符，可以用虚类的方式设置不同类型链接的显示方式。所谓不同类型链接，是指访问过的、未访问过的和激活的这三种状态的链接。

可以通过指定下列 selector 之一设置链接样式：

A:link 或 :link　当超链接没被访问过时，所设置的样式应用于超链接。

A:visited 或 :visited　当超链接已被访问过时，所设置的样式应用于超链接。

A:active 或 :active　当超链接当前为被选中状态时，所设置的样式应用于超链接。

例如，可以使用以下方式更改超链接的颜色显示：

```
:link {color:blue}
:visited {color:purple}
:active {color:gray}
```

对于虚类也可以使用编组的方法。例如，要去掉所有超链接的下划线，可以使用以下样式：

```
A:link, A:visited, A:active {text-decoration: none}
```

对于 Internet Explorer，还有一个特殊的虚类叫做 hover，表示当浏览者将鼠标悬停在超链接上时的样式。例如，以下定义使浏览者将鼠标悬停到超链接时，超链接显示为红色：

```
A:hover {color:red}   /*也可以使用 A:hover{color:red}*/
```

8.2.2 使用样式

明确了如何定义一个样式之后，我们需要在网页中具体引用该样式，才能使样式生效。在网页中引用样式的常用方法有三种：

使用 HTML 标记符的 STYLE 属性嵌套样式信息；

通过在网页 HEAD 标记符中使用 STYLE 标记符嵌套样式信息；

通过在网页 HEAD 标记符中使用 LINK 标记符链接外部的层叠样式表文件（.css 文件）。

1. 使用 STYLE 属性

在 HTML 标记符中使用 STYLE 属性可以将样式信息直接嵌入到标记符内，使相应标记符具有指定的样式。用于 STYLE 属性的值是一个或多个 CSS“属性:值”对，并由分号分开。

例如，要设置一个半英寸的左边距，字体大小为 14pt 的段落，应使用如下样式定义：

```
<P STYLE="margin-left:0.5in; font-size:14pt">
  仅有位于此标记符内的文本受样式的影响。
</P>
```

2. 使用 STYLE 标记符

最常用的引用样式的方式是在页面的 HEAD 标记符内使用 STYLE 标记符，将样式定

义放置到该标记符内。

在 STYLE 标记符内，应使用“属性:值”对为想要定义样式的各种 selector 定义样式。同时，为了使不支持 CSS 的浏览器忽略样式定义，应将样式定义用注释标记符包围起来。如果需要为样式添加注释，应将注释内容用“/*”和“*/”包围起来。

例如，以下示例使用 STYLE 标记符定义了一个 P 标记符的样式：

```
<HEAD>
<STYLE>
<! --
  P{margin-left:0.5in; font-size:14pt}  /* 在这里添加注释 */
-->
</STYLE>
</HEAD>
```

定义了该样式后，在网页中使用 P 标记符时，将自动应用相应的样式。

3. 使用 LINK 标记符

如果相同的样式要应用于多个网页，则可以使用链接样式，也就是将样式定义保存为样式表文件（.css 文件），然后将其链接到需要使用相应样式的网页中。链接样式时应在 HEAD 标记符内使用 LINK 标记符引入外部样式表文件。

例如，如果要将刚才用 STYLE 标记符定义的样式改换成链接式样式，应使用以下两步：

（1）将 HTML 页上 <STYLE> </STYLE> 标记符和其间的所有内容都剪切不来，并粘贴到另一个空白文件上，并用.css 扩展名保存该新文件（假设保存在当前目录，文件名为 linkedStyle.css）。

（2）将一个 LINK 标记符添加到 <HEAD></HEAD> 标记符之间，引用链接式样式表文件（.CSS 文件），如下所示：

```
<LINK REL="stylesheet" TYPE="text/css" HREF="./linkedStyle.css">
```

REL 属性规定了被链接文件的关系，在链接式样表文件（.css 文件）的情况下，REL 属性的值永远是 "stylesheet"；TYPE 属性规定了链接文件的 MIME 类型，它的值永远是 "text/css"；HREF 属性引用了先前创建的链接样式表文件。

链接了样式表文件之后，当网页设计者在网页中使用样式表文件中所定义样式对应的

selector 时，相应样式将自动应用。

8.3 CSS 属性

CSS 属性用于指明格式，通过给它设置具体的值可以确定特定 selector 的样式。例如，如果在样式定义中将 CSS 属性 font-family 的值指定为"楷体_GB2312"，则在应用该样式的 selector 中字体采用“楷体”。本节将介绍各种常用 CSS 属性的基本用法，在附录中将包含所有 CSS 属性的列表，读者可以在制作网页时参考。

8.3.1 CSS 属性单位

在指定 CSS 属性的值时，常常可以使用多种单位。例如，指定长度值时既可以用绝对单位，也可以用百分比单位；指定颜色值时，既可以直接用颜色名称，也可以用 #RRGGBB 方式。本节介绍 CSS 属性单位的一些规定。

1. 长度单位

表 8.1 显示了可在 CSS 中用于描述长度的单位。

表8.1 CSS长度单位

缩 写	单位类型	说 明
cm	厘米	
em	ems	当前字体中 m 字母的宽度
ex	exes	当前字体中 x 字母的高度
in	英寸	
mm	毫米	
pc	Picas	1 pica =12 点
pt	点	1 点=1/72 英寸
px	像素	

注意： 并非所有浏览器都支持以上长度单位。例如，Internet Explorer 3.x 不支持 em 和 ex 单位，Netscape Navigator 4.0 不支持 pc 单位。

2. 百分比单位

除了可以使用前面介绍的长度单位指定尺寸以外，在 CSS 中经常还可以使用百分比单位指定尺寸。例如：P{font-size:150%} 表示该段文字的大小为标准字体的 1.5 倍。

使用百分比单位的格式是：先写上“+”号或“-”号，然后紧跟一个数字，最后是百

分号“%”。如果百分比值是正，那么正号“+”可以忽略不写，也就是说，“+50%”和“50%”是等效的。符号和百分号之间的数字可以是任意值，但由于在某些环境下浏览器不能处理带小数点的百分数，因此建议不要使用小数。另外，符号、数字和百分比号之间不能有空格。

百分比值总是相对于另一个值来说的，该值可以是长度单位或是其他单位。每一个可以使用百分比值单位指定的属性同时也自定义了这个百分比值的参照值。大多数情况下，这个参照值是该元素本身的字体尺寸。

3. 颜色单位

CSS 允许网页设计者使用以下方式中的一种指定颜色：

- 颜色名　直接使用 16 种标准颜色名称（aqua 或 cyan、black、blue、fuchsia 或 magenta、gray、green、lime、maroon、navy、olive、purple、red、silver、teal、white、yellow）或浏览器支持的其他颜色名称（有关 IE 4.0/5.0 支持的颜色名称列表，可以到作者的个人网站 http://zhaofengnian.yeah.net 上下载）。
- #RRGGBB　使用两位十六进制数表示颜色中的红、绿、蓝含量。例如，红色可以用 #FF0000 表示。
- #RGB　使用一位十六进制数表示颜色中的红、绿、蓝含量，它是 #RRGGBB 方式的快捷方式。例如，颜色 #002200 可以表示为 #020；#00FFEE 可以表示为 #0FE。
- rgb(rrr, ggg, bbb)　使用十进制数表示颜色的红、绿、蓝含量，其中 rrr、ggg 和 bbb 都是 0~255 的十进制数。
- rgb(rrr%, ggg%, bbb%)　使用百分比表示颜色的红、绿、蓝含量。例如，rgb(50%, 0, 50%) 相当于 rgb(128, 0, 128)。

浏览器有时并不能兼容所有的颜色，此时，使用三位的十六进制定义或标准颜色名称可以比较“安全”地显示颜色。如果确实需要定义其他颜色，通常应使用十六进制数来定义，因为所有浏览器兼容的颜色都是基于十六进制数组的。

8.3.2　字体与文本属性

字体属性用于控制网页中文本的字符显示方式，文本属性用于控制网页中文本的段落

布局。

1. 字体属性

字体属性用于控制网页中的文本显示样式，例如控制文字的大小、粗细以及使用的字体等。CSS 中的字体属性包括 font、font-family、font-size、font-style、font-variant 和 font-weight。

font-family 属性用于确定要使用的字体列表，取值可以是字体名称，也可以是字体族名称，值之间用逗号分隔。在使用字体或字体族时，字体或字体族名称中间的空格应用破折号进行替换（例如，new century schoolbook 变为 new-century-schoolbook），或对字体或字体族加上引号（例如“new century schoolbook”）。系统中安装的字体可以通过“控制面板”中的“字体”工具来查看。第 1 级 CSS 定义了以下字体族名：cursive、fantasy、monospace、serif 和 sans-serif。字体族中通常包含多种字体，例如，serif 字体族中包含 Times 字体；monospace 字体族中包含 Courtier 字体等。

font-size 属性用于控制字体的大小，它的取值分为四种类型：绝对大小、相对大小、长度值以及百分数。该属性的默认值是 medium。当使用绝对大小类型时，可能的取值为：xx-small | x-small | small | medium | large |x-large |xx-large，表示越来越大的字体。当使用相对大小时，可能的取值为 smaller | larger，分别表示比上一级元素中的字体小一号或大一号。例如，如果在上级元素中使用了 medium 大小的字体，而子元素采用了 larger 值，则子元素的字体尺寸将是 large。当使用长度值时，可以直接指定。当使用百分比值时，表示与当前默认字体（即 medium 所代表字体的大小）的百分比。

font-style 属性确定指定元素显示的字形。font-style 属性的值包括：normal、italic 和 oblique 三种。默认值为 normal，表示普通字形；italic 和 oblique 表示斜体字形。

font-variant 属性决定了浏览器显示指定元素的字体变体。该属性可以有两个值：small-caps 和 normal。默认值为 normal，表示使用标准字体；small-caps 表示小体大写，也就是说，字体中所有小写字母看上去与大写字母一样，不过尺寸要比标准的大写字母要小一些。

font-weight 属性定义了字体的粗细值，它的取值可以是以下值中的一个：normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900，默认值为 normal，表示正常粗细，bold 表示粗体。也可以使用数值，范围为 100 至 900，对应从最细到最粗。Normal 相当于 400，bold 相当于 700。如果使用 bolder 或 lighter，则表示相对于上一级元素中的字体更粗或更细。

使用 font 属性可一次性设置前面介绍的各种字体属性（属性之间以空格分隔）。在使用 font 属性设置字体格式时，各字体属性可以省略，但如果包括相应属性，必须按以下顺序出现：font-weight、font-variant、font-style、font-size、line-height（此属性的值可以位于 font 属性中，用于指定行高，它必须在 font-size 后用斜线隔开）和 font-family。

以下示例显示了各种常用字体属性的用法，代码如下：

```
<HTML>
<HEAD>
  <TITLE>字体属性示例</TITLE>
<STYLE>
<!--
   H1 {font:bolder italic}
   .title{font-family:楷体_GB2312 }
   .author{font-family:隶书}
   .content{font-size:larger}
-->
</STYLE>
</HEAD>
<BODY>
  <H1 class=title align=center>浣溪沙</H1>
  <P class=author align=right>秦观</P>
  <P class=content>漠漠轻寒上小楼，晓阴无赖似穷秋，淡烟流水画屏幽。自在飞花轻似梦，无边丝雨细如愁，宝帘闲挂小银钩。</P>
</BODY>
</HTML>
```

此示例的显示效果如图 8.3 所示。

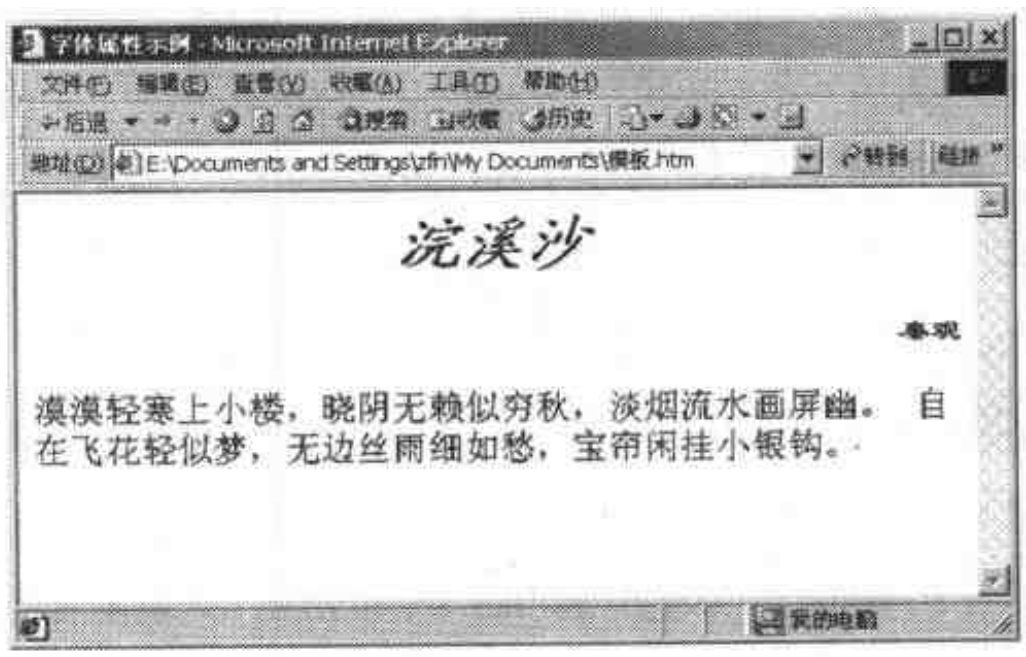


图 8.3　字体属性示例

2. 文本属性

文本属性用于控制文本的段落格式，例如设置首行缩进、段落对齐方式等。CSS 中的常用文本属性包括：line-height、letter-spacing、text-align、text-decoration、text-indent、text-transform 和 vertical-align。

letter-spacing 属性指定的值决定了字符间距(除去默认距离外)。它的取值可以是 normal 或具体的长度值，也可以是负值。默认值为 normal，表示浏览器根据最佳状态调整字符间距。也就是说，如果将 letter-spacing 设置为 0，它的效果并不与 normal 相同。

line-height 属性决定了相邻行之间的间距（或者说行高）。其取值可以是数字、长度或百分比，默认值是 normal。当以数字指定值时，行高就是当前字体高度与该数字相乘的倍数。例如，DIV{font-size:10pt; line-height:1.5}表示的行高是 15pt。如果指定具体的长度值，则行高为该值。如果用百分比指定行高，则行高为当前字体高度与该百分比相乘。

text-align 属性指定了所选元素的对齐方式，取值可以是：left | right | center | justify，分别表示左对齐、右对齐、居中对齐和两端对齐。此属性的默认值依浏览器的类型而定。使用此 CSS 属性指定对齐，类似于在 HTML 元素中使用 align 属性。

text-decoration 属性可以对特定选项的文本进行修饰，它的取值为：none | [underline | overline | line-through | blink]，默认值为 none，表示不加任何修饰。underline 表示添加下划线；overline 表示添加上划线；line-through 表示添加删除线；blink 表示添加闪烁效果。

text-indent 属性可以对特定选项的文本进行首行缩进，取值可以是长度值或百分比。此属性的默认值是 0，表示无缩进。

text-transform 属性用于转换文本，取值为：capitalize | uppercase | lowercase | none，默认值是 none。capitalize 值指示所选元素中文本的每个单词的首字母为大写；uppercase 值指示所有的文本都为大写，lowercase 值指示所有文本都以小写显示。

vertical-align 属性定义了一个元素在垂直方向上的位置，该位置是相对于它上一级元素的位置而言的。该属性的取值为：baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage>，默认值是 baseline。其中的 6 个值（baseline、middle、sub、super、text-bottom 和 text-top）表示与修饰选项的父元素竖直对齐（baseline 表示将元素的基线与上一级元素的基线对齐，如果该元素没有基线，则取它的底端与上级元素的基线对齐；middle 表示将元素的垂直中点与上级元素的垂直中点对齐；sub 和 super 分别表示按元素下标或上标位置对齐；text-top 和 text-bottom 分别表示将元素的顶端或底端与上级元素的顶端或底

端对齐），top 和 bottom 值分别表示与行的最高和最低元素竖直对齐，<percentage> 表示采用相对于该元素行高属性（line-height）的百分比。

以下示例将前面的字体属性示例进一步完善，把段落对齐格式用文本属性设置并增加了首行缩进效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>文本属性示例</TITLE>
<STYLE>
<!--
   H1 {font:bolder italic}
   .title{font-family:楷体_GB2312; text-align:center}
   .author{font-family:隶书; text-align:right}
   .content{font-size:larger; text-indent:1cm}
-->
</STYLE>
</HEAD>
<BODY>
  <H1 class=title >浣溪沙</H1>
  <P class=author>秦观</P>
  <P class=content>漠漠轻寒上小楼，晓阴无赖似穷秋，淡烟流水画屏幽。自在飞花轻似梦，无边丝雨细如愁，宝帘闲挂小银钩。</P>
</BODY>
</HTML>
```

此示例的显示效果如图 8.4 所示。

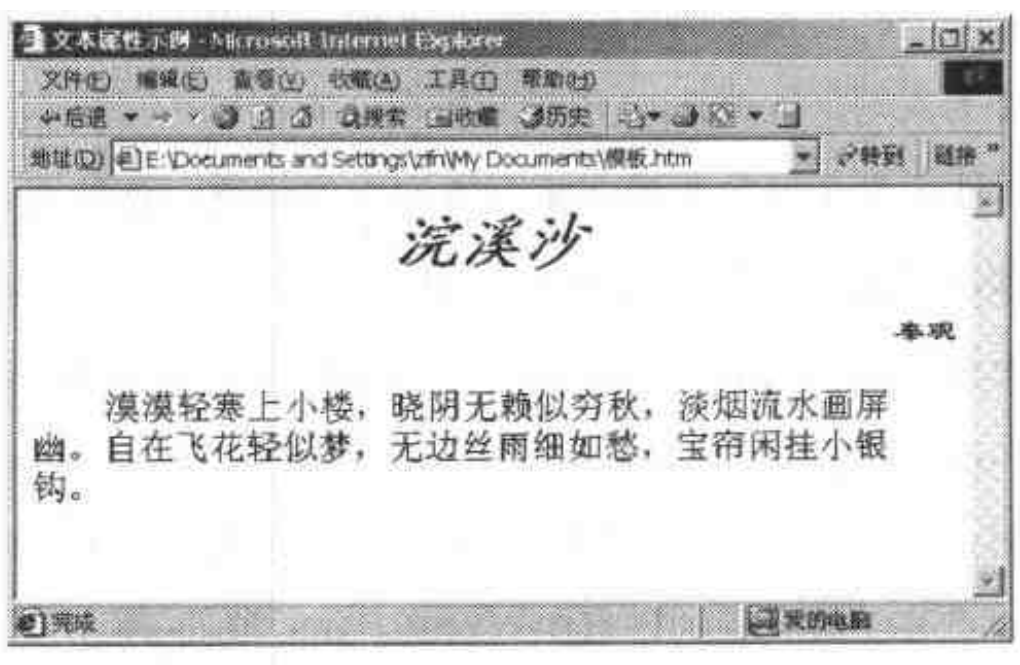


图 8.4　文本属性示例

8.3.3 颜色与背景属性

在 CSS 中，color 属性可以设置元素内文本的颜色，而各种背景属性则可以控制元素的背景颜色以及背景图案。CSS 背景属性包括：background、background-attachment、background-color、background-image、background-position 和 background-repeat。

color 属性用于控制 HTML 元素内文本的颜色，取值可以使用 8.3.1 中介绍的任意一种方式。例如，可以用以下任意一种方式为 H1 元素设置绿色的文本颜色：H1.green{color:green}、H1.green{color:#00FF00}、H1.green{color:#0F0}、H1.green{color:rgb(0,255,0)}、H1.green{color:rgb(0,100%,0)}。

background-color 属性用于设置 HTML 元素的背景颜色，取值可以是 8.3.1 中介绍的任意一种表示颜色的方式。此属性的默认值是 transparent，表示没有任何颜色（或者说是透明色），此时上级元素的背景可以在子元素中显示出来。

background-image 属性用于设置 HTML 元素的背景图案，取值为 url(imageurl) 或 none。默认值为 none，即没有背景图案。当指定图案的位置时，应包括在“url”字样后的括号中。background-attachment、background-position 和 background-repeat 属性都是当使用了 background-image 属性时才有效，它们用于控制背景图案的显示。

background-attachment 属性控制背景图像是否随内容一起滚动，取值为 scroll | fixed。默认值为 scroll，表示背景图案随着内容一起滚动；fixed 表示背景图案静止，而内容可以滚动。

background-position 属性指定了背景图案相对于关联区域左上角的位置。该属性通常是指定由空格隔开的两个值，既可以使用关键字 left | center | right 和 top | center | bottom，也可以指定百分数值，或者指定以标准单位计算的距离。例如，50%表示将背景图像放在区域的中心位置，25px 的水平值表示图像左侧距离区域左侧 25px。如果只提供了一个值而不是一对值，则相当于只指定水平位置，垂直位置自动设置为 50%。指定距离时，也可以使用负值，表示图像可超出边界。此属性的默认值是“0% 0%”，表示图像与区域左上角对齐。

background-repeat 属性表示当使用背景图案时，背景图案是否重复显示。取值可以是：repeat | repeat-x | repeat-y | no-repeat，默认值是 repeat，表示在水平方向和垂直方向都重复，即象铺地板一样将背景图案平铺；repeat-x 表示在水平方向上平铺；repeat-y 表示在垂直方

向上平铺；no-repeat 表示不平铺，即只显示一幅背景图案。

background 属性与 font 属性类似，它也是一个组合属性，可用于同时设置 background-color 、 background-image 、 background-attachment 、 background-position 、 background-repeat 等背景属性。不过，在指定 background 属性时，各属性值的位置可以是任意的。在 Internet Explorer 5.0 中，甚至 background-position 的两个值都可以分开。

以下示例将前面的文本属性示例进一步完善，为标题增加了背景图案，并为整个正文增加了背景颜色，同时设置了文本的颜色，代码如下：

```
<HTML>
<HEAD>
  <TITLE>颜色与背景属性示例</TITLE>
<STYLE>
<!--
   H1 {font:bolder italic}
   .title{ font-family:楷体_GB2312;
text-align:center;
background-image:url(./images/paper.jpg)
}
   .author{font-family:隶书;
text-align:right;
color:#00008b
}
   .content{font-size:larger; text-indent:1cm}
   BODY{background:#f0f8ff}
-->
</STYLE>
</HEAD>
<BODY>
  <H1 class=title >浣溪沙</H1>
  <P class=author>秦观</P>
  <P class=content>漠漠轻寒上小楼，晓阴无赖似穷秋，淡烟流水画屏幽。自在飞花轻似梦，无边丝雨细如愁，宝帘闲挂小银钩。</P>
</BODY>
</HTML>
```

此示例的显示效果如图 8.5 所示。

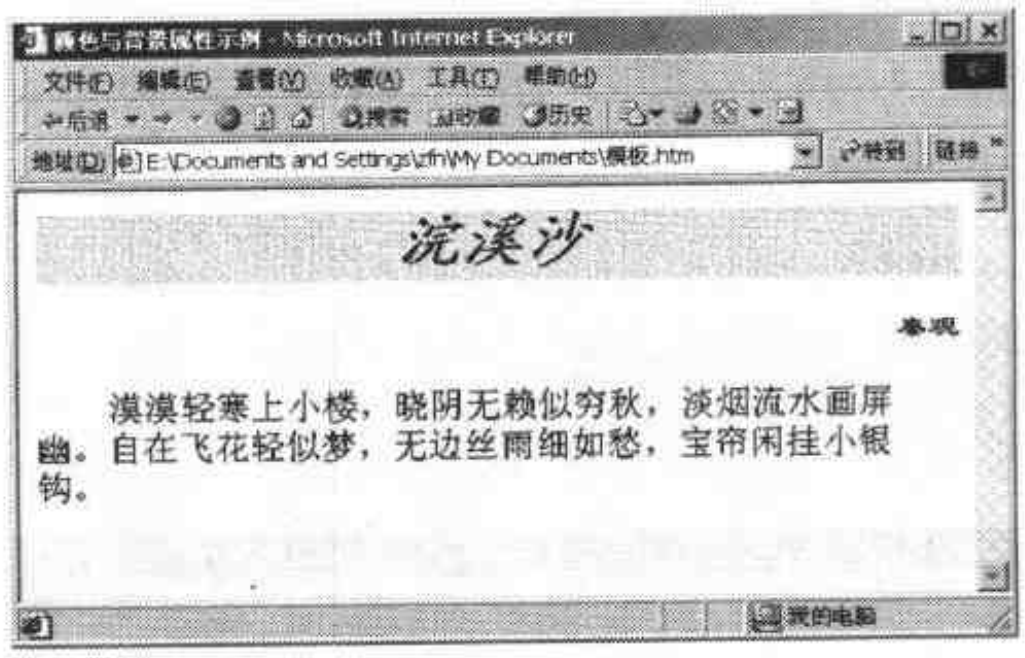


图 8.5 颜色与背景属性示例

8.3.4 布局属性

布局属性主要包括 4 类：边框（border）属性、边界（margin）属性、填充（padding）属性和浮动属性。

在任何一个 HTML 元素的周围，都包含边框、边界和填充这三种空白。最接近元素内容的是填充，接下来是边框，最外围是边界。边界区总是透明的，可以显示出背景色或背景图像；而填充总是采用标记符的背景色或背景图像；边框则可以使用自己的颜色。

例如，使用以下示例可以看出这三种空白的区别，代码如下：

```
<HTML>
<HEAD>
<TITLE>边界、边框和填充的区别</TITLE>
<STYLE>
<!--
  P{margin:0.25in;   border:0.25in   solid   black;   padding:0.25in;
background:yellow}
-->
</STYLE>
</HEAD>
<BODY>
    <P>示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。</P>
    <P>示例文字。示例文字。</P>
</BODY>
</HTML>
```

此示例的效果如图 8.6 所示，其中，文字周围的黄色空白为填充距（padding），黑色部分为边框（border），而再向外与浏览器窗口或其他内容相临的空白（以文档的背景颜色显示）则是边界（margin）。

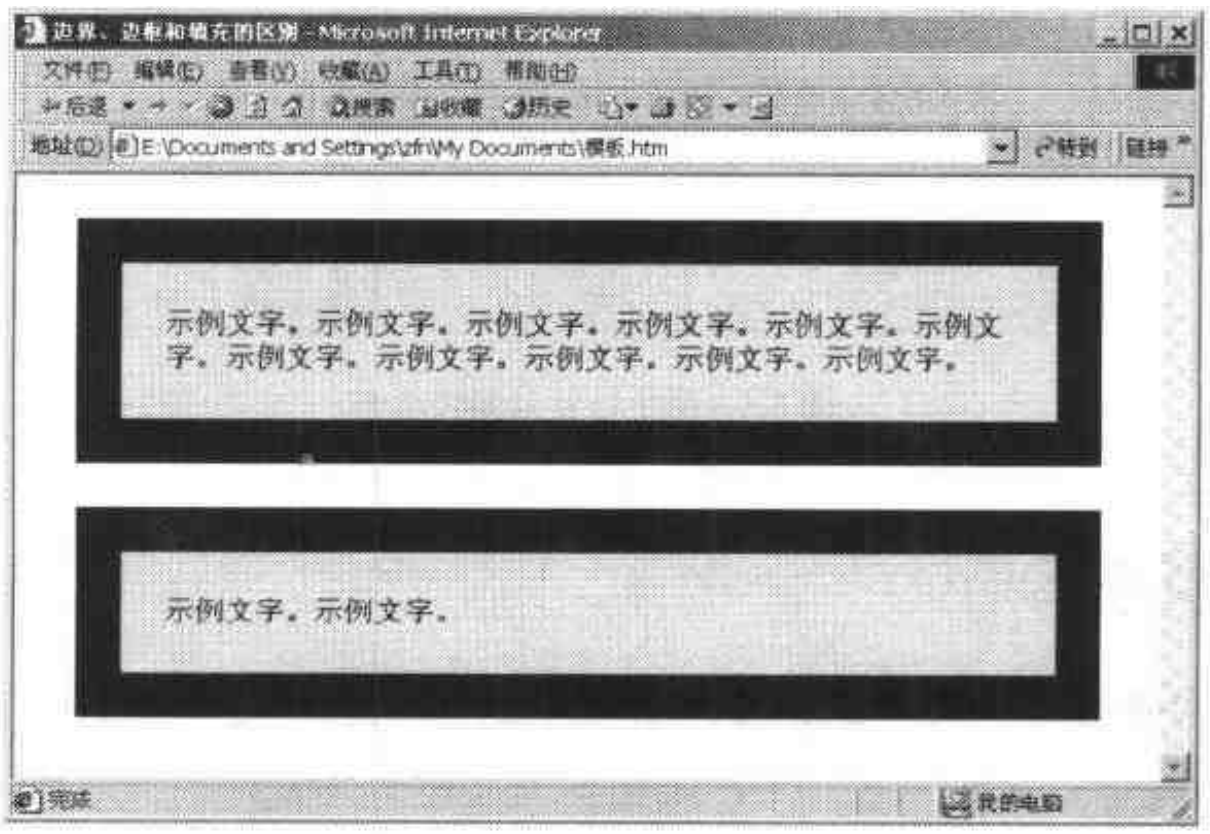


图 8.6　元素周围空白的区别

1．边框属性

CSS 边框属性包括 border、border-bottom、border-bottom-color、border-bottom-style、border-bottom-width、border-color、border-left、border-left-color、border-left-style、border-left-width、border-right、border-right-color、border-right-style、border-right-width、border-style、border-top、border-top-color、border-top-style、border-top-width 以及 border-width。

根据属性的命名可以看出，有关边框的设置包括三项：边框颜色（color）、边框样式（style）和边框宽度（width），而边框又包括四个方向：上（top）、下（bottom）、左（left）和右（right）。将边框设置和方向组合起来，则构成了多种属性。

border-bottom-color、border-left-color、border-right-color、border-top-color 属性分别用于指定下、左、右、上边框的颜色，取值可以使用各种指定颜色的方式。也可以使用 border-color 属性同时指定 4 个边框的颜色。如果分别指定，则必须按上、右、下、左的顺序指定；如果只指定了一个值，则所有边框的颜色一样；如果指定了 2 或 3 个值，则来指定颜色的边框采用相对边框的颜色值。

border-bottom-style、border-left-style、border-right-style、border-top-style 属性分别用于

设置下、左、右、上边框的样式，取值可以是：none | dotted | dashed | solid |double | groove | ridge | inset | outset，默认值是 none。none 指示无边框（即使已设置了其边框宽度）；dotted 指定边框由点线组成；dashed 指定使用划线表示边框；solid 指边框由实线组成；double 指使用双线；groove 和 ridge 利用元素的颜色属性值描出具有三维效果的边框；类似地，inset 和 outset 利用修饰元素的颜色值描出边框效果。也可以用 border-style 属性同时指定 4 个边框的样式。如果分别指定，则必须按上、右、下、左的顺序指定；如果只指定了一个值，则所有边框的样式一样；如果指定了 2 或 3 个值，则未指定样式的边框采用相对边框的样式。需要注意的是，如果浏览器（例如 IE5）不支持边框样式的属性值，则除了 none 以外的所有属性值都用 solid 代替。

border-bottom-width、border-left-width、border-right-width、border-top-width 属性分别用于设置下、左、右、上边框的宽度，取值可以是：thin | medium | thick | <length>，其中 <length> 是可以使用的长度单位数值。这 4 个属性的默认值是 medium，并且取值不能是负数。也可以用 border-width 属性同时指定 4 个边框的宽度。如果分别指定，则必须按上、右、下、左的顺序指定；如果只指定了一个值，则所有边框的宽度一样；如果指定了 2 或 3 个值，则未指定宽度的边框采用相对边框的宽度。

border-left、border-right、border-top 和 border-bottom 属性可以用来一次性指定左、右、上、下边框的宽度、样式和颜色，其取值可以是 border-width、border-color 和 border-style 属性的取值。如果没有指定某个值，则该值采用默认值。当指定宽度、样式和颜色时，并没有顺序要求。

border 属性可以用来一次性设置 4 个方向上边框的宽度、样式和颜色，它是指定元素边框各个边的简捷方式。用 border 属性指定边框时，4 个边框都具有相同的设置。同样，指定宽度、样式和颜色时，也没有顺序要求。

以下示例将前面的颜色与背景示例进一步完善，为标题和内容增加了边框设置，代码如下：

```
<HTML>
<HEAD>
  <TITLE>边框属性示例</TITLE>
<STYLE>
<!--
   H1 {font:bolder italic}
   .title{ font-family:楷体_GB2312; text-align:center;
```

```
background-image:url(./images/paper.jpg);
border-width:thin medium thick;
border-style:solid;
border-color:gray
}
   .author{font-family:隶书; text-align:right; color:#00008b}
   .content{font-size:larger; text-indent:1cm;
border:solid #5f9ea0
}
   BODY{background:#f0f8ff}
-->
</STYLE>
</HEAD>
<BODY>
  <H1 class=title >浣溪沙</H1>
  <P class=author>秦观</P>
  <P class=content>漠漠轻寒上小楼，晓阴无赖似穷秋，淡烟流水画屏幽。自在飞花轻似梦，无边丝雨细如愁，宝帘闲挂小银钩。</P>
</BODY>
</HTML>
```

此示例的显示效果如图 8.7 所示。

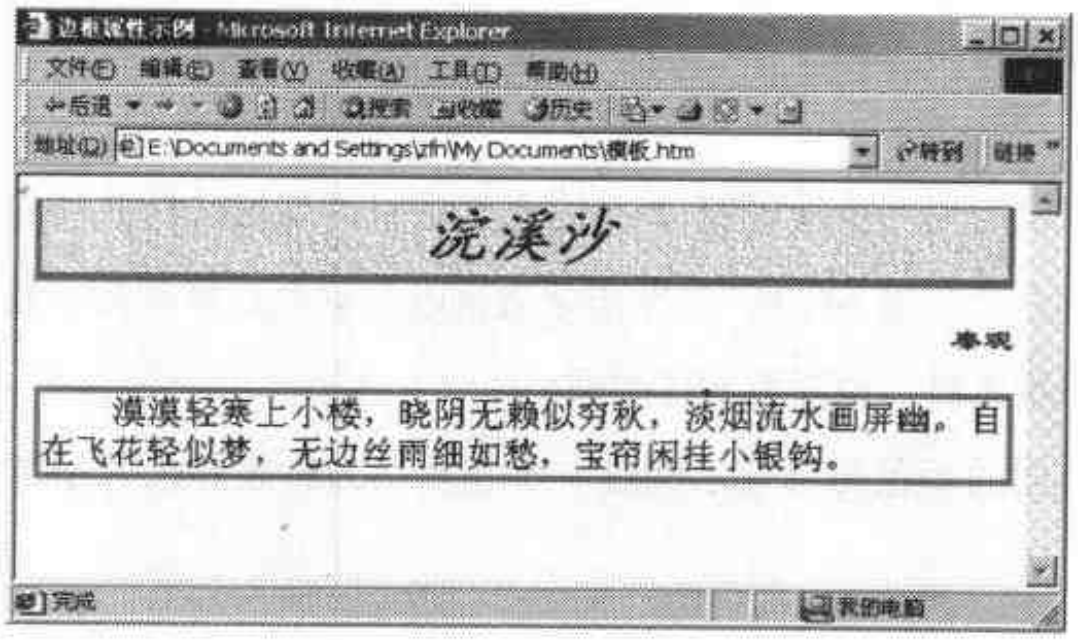


图 8.7　边框属性示例

2. 边界属性

CSS 边界属性包括 margin、margin-bottom、margin-left、margin-right 以及 margin-top。margin-left、margin-right、margin-top 和 margin-bottom 属性可以分别用来设置左、右、

上、下边界的宽度，它们的取值可以是长度、百分比或 auto。当使用百分比时，表示相对于父元素宽度的百分比。

margin 属性可以同时指定上、右、下、左（以此顺序）边界的宽度。如果只指定一个值，则四个方向都采用相同的边界宽度；如果指定了 2 或 3 个值，则没有指定边界宽度的边采用对边的边界宽度。指定边界宽度时也可以使用负值，以便获得特殊的效果。

以下示例将前面的边框属性示例进一步完善，为“标题”和“作者”文字增加了一定的边界空间，代码如下：

```
<HTML>
<HEAD>
  <TITLE>边界属性示例</TITLE>
<STYLE>
<!--
   H1 {font:bolder italic}
   .title{ font-family:楷体_GB2312; text-align:center;
        background-image:url(./images/paper.jpg);
        border-width:thin   medium   thick;border-style:solid;border-
color:gray;
        margin:20px 50px;
        }
   .author{font-family:隶书; text-align:right; color:#00008b;
          margin-right:0.75cm
          }
   .content{font-size:larger; text-indent:1cm;border:solid #5f9ea0}
   BODY{background:#f0f8ff}
-->
</STYLE>
</HEAD>
<BODY>
  <H1 class=title>浣溪沙</H1>
  <P class=author>秦观</P>
  <P class=content>漠漠轻寒上小楼，晓阴无赖似穷秋，淡烟流水画屏幽。自在飞花轻似梦，无边丝雨细如愁，宝帘闲挂小银钩。</P>
</BODY>
</HTML>
```

本示例的显示效果如图 8.8 所示。

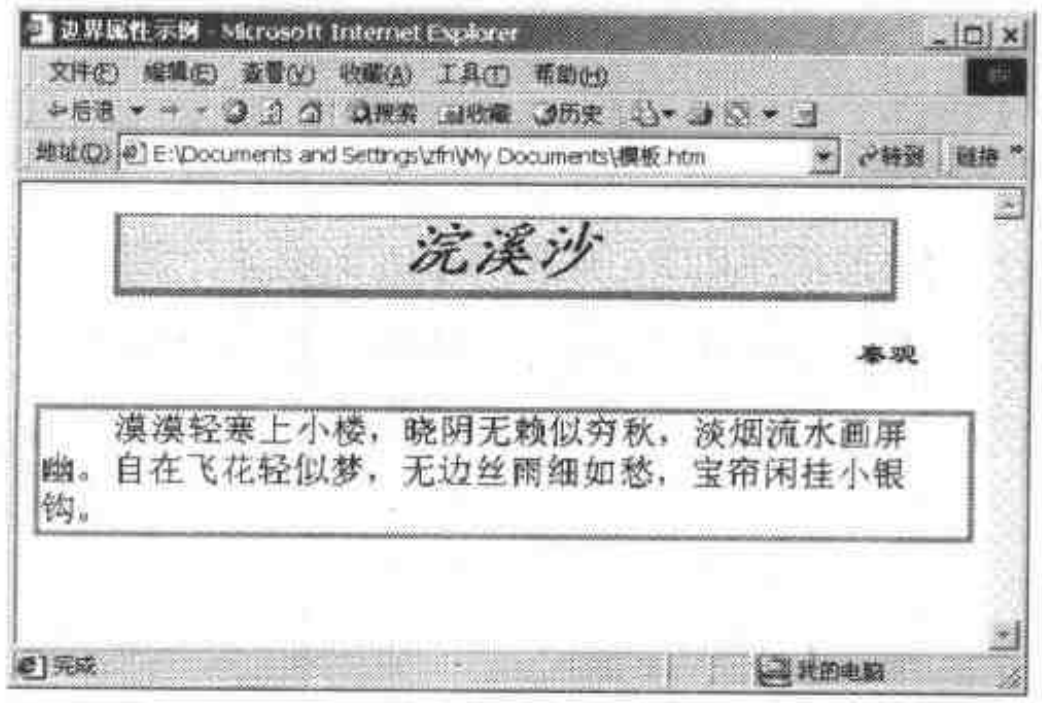


图 8.8　边界属性示例

3. 填充属性

CSS 填充属性包括 padding、padding-left、padding-right、padding-top 以及 padding-bottom。

padding-left、padding-right、padding-top 和 padding-bottom 属性这四个属性与对应的四个边界属性类似，用于设置左、右、上、下填充区的宽度，取值可以是长度和百分数，但不允许使用负值。当使用百分比时，表示相对于父元素宽度的百分比。

padding 属性用于同时指定上、右、下、左四个方向（以此顺序）填充的宽度。如果只指定一个值，则四个方向都采用相同的填充宽度；如果指定了 2 或 3 个值，则没有指定填充宽度的边采用对边的填充宽度。

以下示例将前面的边界属性示例进一步完善，为“标题”和“作者”文字增加了一定的填充空间，代码如下：

```
<HTML>
<HEAD>
  <TITLE>填充属性示例</TITLE>
<STYLE>
<!--
   H1 {font:bolder italic}
   .title{ font-family:楷体_GB2312; text-align:center;
        background-image:url(./images/paper.jpg);
        border-width:thin   medium   thick;border-style:solid;border-
color:gray;
```

```
        margin:20px 50px;
        padding:10px 20px 20px}
    .author{font-family:隶书; text-align:right; color:#00008b;
          margin-right:0.75cm}
    .content{font-size:larger; text-indent:1cm;border:solid #5f9ea0;
          padding:10px}
    BODY{background:#f0f8ff}
-->
</STYLE>
</HEAD>
<BODY>
  <H1 class=title>浣溪沙</H1>
  <P class=author>秦观</P>
  <P class=content>漠漠轻寒上小楼，晓阴无赖似穷秋，淡烟流水画屏幽。自在飞花轻似梦，无边丝雨细如愁，宝帘闲挂小银钩。</P>
</BODY>
</HTML>
```

本示例的显示效果如图 8.9 所示。

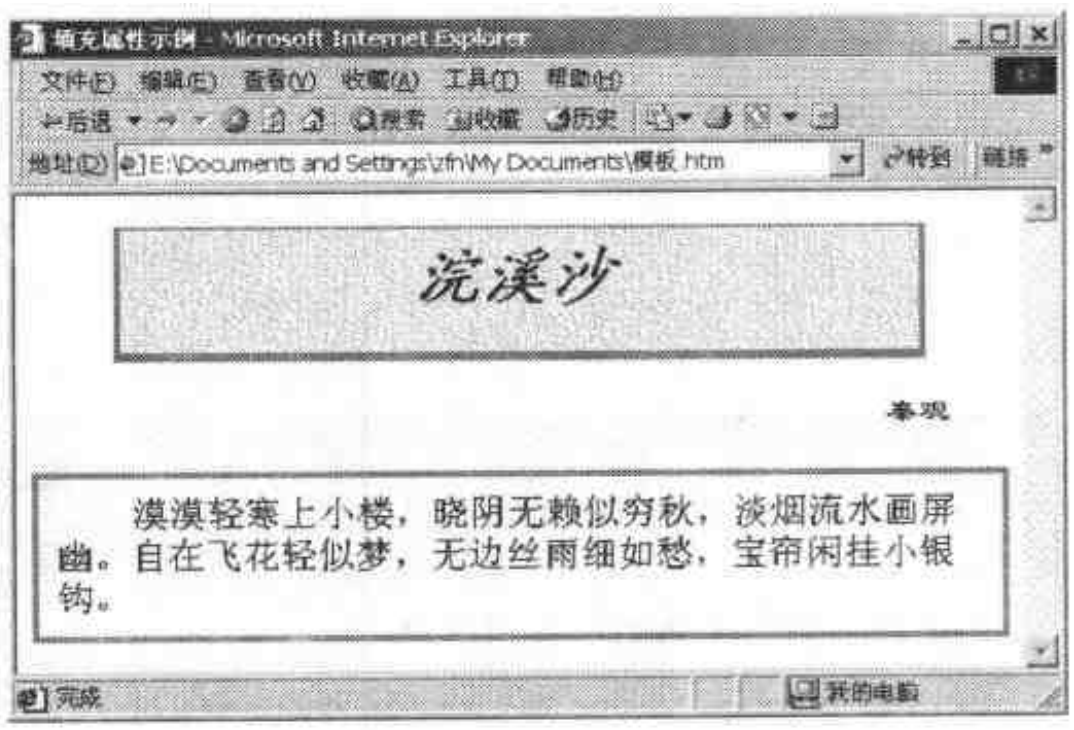


图 8.9 填充属性示例

4. 浮动属性

CSS 浮动属性包括 float 和 clear。通过 float 属性可以将元素的内容浮动到页面左边缘或右边缘，该属性的取值为：none | left | right，默认值为 none，指示元素不浮动到任一边缘。clear 属性指定了元素是否允许浮动元素在它旁边，取值可以是：none | left | right | both，默认值为 none，表示允许浮动元素在其旁边；值 left 表示跳过左边的浮动元素，right 表示

跳过右边的浮动元素，both 表示跳过所有的浮动元素。

以下示例显示了这两个浮动属性的效果，代码如下：

```
<HTML>
<HEAD>
<TITLE>浮动属性示例</TITLE>
<STYLE>
<!--
SPAN {font-family:Garamond;font-size: 45pt;
      float: left}
-->
</STYLE>
</HEAD>
<BODY>
<H2 align=center>I have a dream</H2>
<SPAN>F</SPAN>ive score years ago, a great American, in whose symbolic
shadow we stand today, signed the Emancipation Proclamation.
<P style="clear:both">This momentous decree came as a great beacon light
of hope to millions of Negro slaves who had been seared in the flames
of withering injustice. It came as the joyous daybreak to end the long
night of their captivity.</P>
</BODY>
</HTML>
```

本示例的效果如图 8.10 所示。

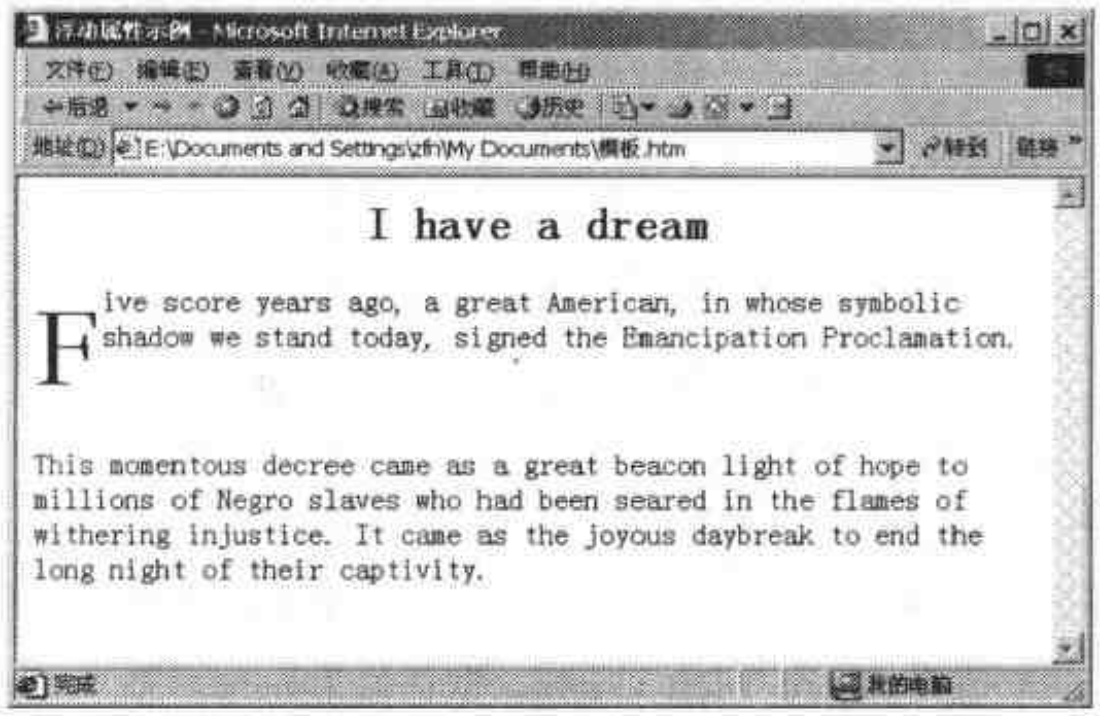


图 8.10　浮动属性示例

8.3.5 定位和显示属性

定位和显示属性用于控制页面元素的位置和显示。

1. 定位属性

CSS 定位属性包括 position、top、bottom、left、right 和 z-index。其中 bottom 和 right 属性并不常用。

position 属性用来规定元素怎样在 Web 页上定位，它的取值为：static（默认值）、relative 或 absolute。“static”表示按照 HTML 格式规则正常定位；“relative”是指某元素将定位在相对于 Web 页上前一个元素的尾端位置；“absolute”是指某元素将定位在框架或浏览器窗口本身的左上角绝对位置。

top 和 left 属性用来规定某个元素与其父或其他元素之间的距离。这两个属性按像素来设定元素位置往下或往右的距离，这里既包括与其父的左上角的（绝对定位）之间的距离，也包括相对于前一个元素尾端之间（相对定位）的距离。

使用 top 和 left 属性可能会造成元素相互堆叠在一起，此时可以使用 z-index 属性。z-index 属性用来控制元素的堆叠，值较高的元素将覆盖值较低的元素。如果使用值-1，则表示元素将置于页面默认文本的后边，这对于设置背景图案是很有用的。

以下示例展示了 CSS 定位属性的强大功能，获得了一种类似图像的效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>定位属性示例</TITLE>
<STYLE>
<!--
 .block1 {background-color:#777744;
       position:absolute; top:20px; left:30px;
       z-index:1;
       width:400
}
 .block2 {background-color:#7777aa;
     position:absolute; top:35px; left:80px;
       z index:2;
       width:450
}
 .block3 {background-color:#7777ff;
```

```
        position:absolute; top:50px; left:180px;
        z-index:3;
        width:400
}
 .title1 {color:#ffffff; font-size:66pt;
      position:absolute; top:20px; left:300px; z-index:6 }
 .title2 {color:#000000; font-size:66pt;
      position:absolute; top:23px; left:303px;
      z-index:5}
 .title3 {color:#444444; font-size:66pt;
      position:absolute; top:26px; left:306px;
      z-index:4}
 .author {color:#ff0000; font-size:12pt; letter-spacing:16;
      position:absolute; top:100px; left:30px;
      z-index:7}
 .content{color:#007fff; font-size:18pt;
      position:absolute; top:200px; left:50px;
      z-index:8; width:650}
 .innerContent{ text-indent:.75cm;text-align:justify}
-->
</STYLE>
</HEAD>
<BODY>
<DIV class="block1">  <!--定义颜色块-->
  <H4> </H4>
</DIV>
<DIV class="block2">  <!--定义颜色块-->
  <H5> </H5>
</DIV>
<DIV class="block3">  <!--定义颜色块-->
  <H6> </H6>
</DIV>
<DIV class="title1"><P>醉花阴</P></DIV>  <!--文字重叠造成阴影效果-->
<DIV class="title2"><P>醉花阴</P></DIV>
<DIV class="title3"><P">醉花阴</P></DIV>
<DIV class="author">
  <P><I>李清照</I></P>
</DIV>
<DIV class="content">
```

```
<P class="innerContent">薄雾浓云愁永昼，瑞脑消金兽。佳节又重阳，玉枕纱厨，
半夜凉初透。 东篱把酒黄昏后，有暗香盈袖。莫道不消魂？帘卷西风，人比黄花瘦。</P>
</DIV>
</BODY>
</HTML>
```

此示例的显示效果如图 8.11 所示。

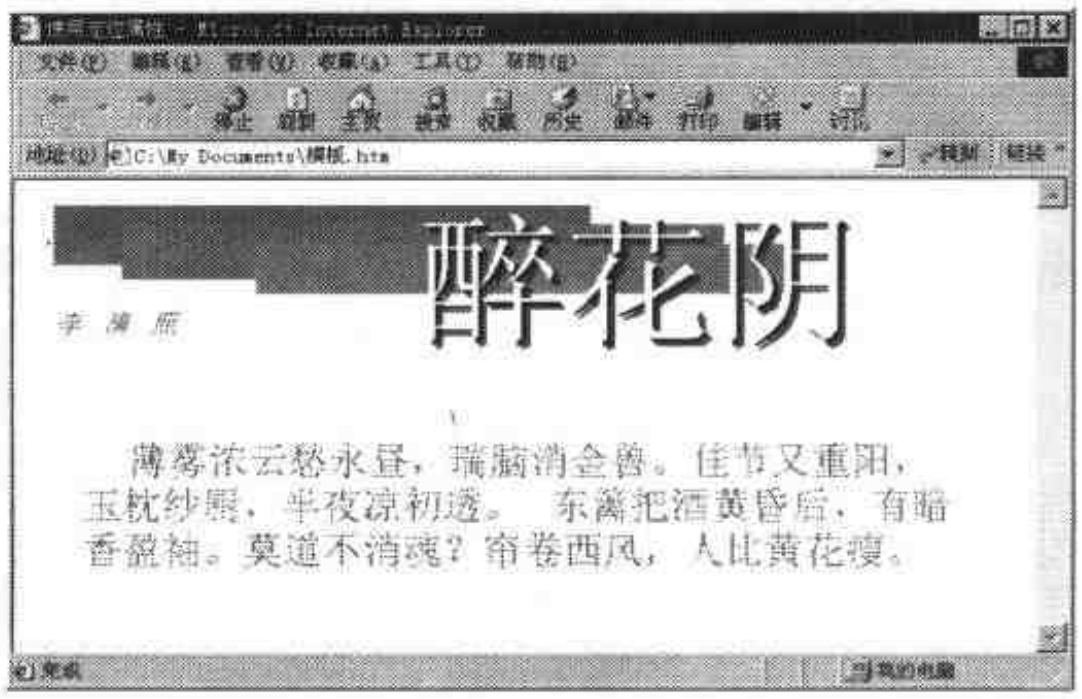


图 8.11 使用定位属性

2. 宽高和剪裁属性

在 CSS 中，可以用 width 和 height 属性控制元素的宽度和高度（在前面的定位属性示例中已经用到了 width 属性），用 clip 和 overflow 属性控制元素的剪裁。使用这些属性时，position 属性必须指定为 absolute。

width 和 height 属性的取值可以是长度值，也可以是百分比。如果说 left 和 top 属性定义了元素的位置，那么 width 和 height 属性则规定了元素所占空间的大小。

clip 属性可以确定可视定位对象的剪裁区域，取值为 rect(top right bottom left) | auto。其中 top、right、hottom 和 left 用于指定上、右、下、左 4 个方向上的剪裁长度，取值为长度值或 auto。如果任意一边使用 auto，则相当于该边没有进行剪裁。

overflow 属性用于设置当元素的内容超出它的高度和宽度限制时，浏览器如何进行处理，取值可以是 visible | hidden | scroll | auto，其中 visible 是默认值。值 visible 表示不剪裁内容，也不添加滚动条，强制显示元素内容；hidden 表示剪裁内容，即不显示超出对象尺寸的内容；scroll 表示剪裁内容，同时提供滚动条；auto 表示只有在必要时才剪裁内容并添加滚动条。

以下示例显示了这些属性的作用，代码如下：

```
<HTML>
<HEAD>
<TITLE>宽高和剪裁属性示例</TITLE>
<STYLE>
<!--
.original{ position:absolute; top:20px; left:30px }
.test1{ position:absolute; top:280px; left:30px; clip:rect(0 50px 50px
0)}
.test2{   position:absolute;   top:280px;   left:130px;   width:50;
height:50;overflow:scroll}
.test3{   position:absolute;   top:280px;   left:230px;   width:50;
height:50;overflow:hidden}
.test4{   position:absolute;   top:280px;   left:330px;   width:50;
height:50;overflow:auto}
-->
</STYLE>
</HEAD>
<BODY>
<DIV class=original>
  <IMG src="./images/paper.jpg">   <!--原始图像-->
</DIV>
<DIV class=testl>
  <IMG src="./images/paper.jpg">  <!--剪裁图像-->
</DIV>
<DIV class=test2>
  <IMG src="./images/paper.jpg">  <!--剪裁图像-->
</DIV>
<DIV class=test3>
  <IMG src="./images/paper.jpg">  <!--剪裁图像-->
</DIV>
<DIV class=test4>
  <IMG src="./images/paper.jpg">  <!--剪裁图像-->
</DIV>
</BODY>
</HTML>
```

本示例的效果如图 8.12 所示。

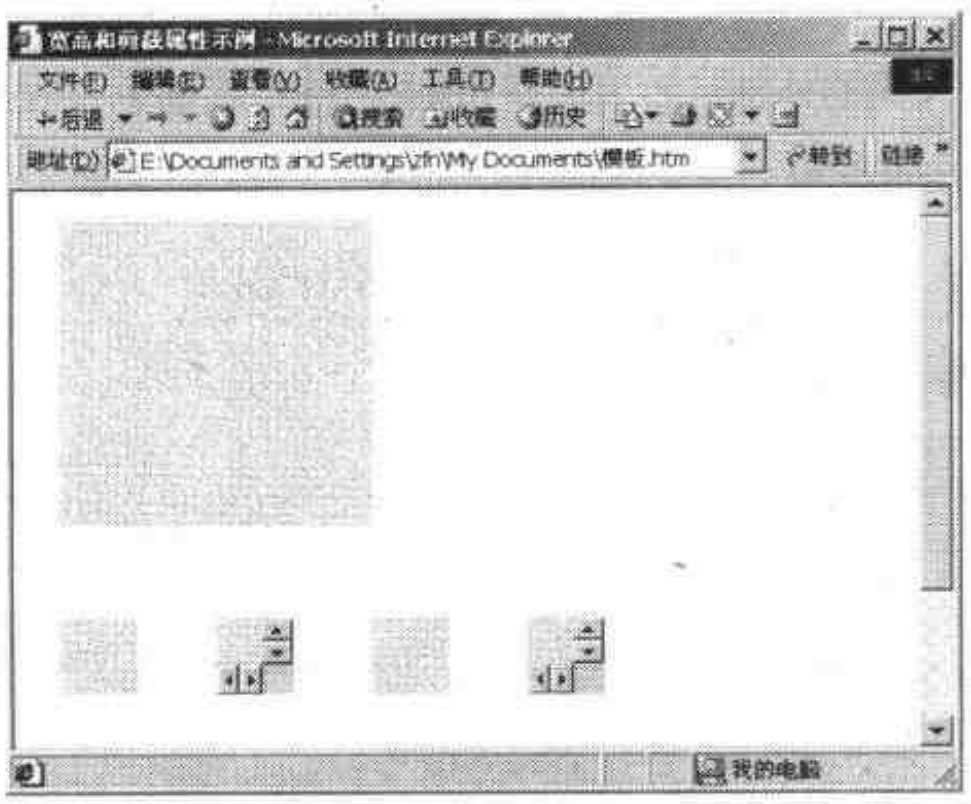


图 8.12 宽高和剪裁属性示例

3. 显示属性

在 CSS 中，有两个属性可以控制元素的显示和隐藏，即 display 属性和 visibility 属性。

display 属性确定一个元素是否应绘制在页面上，它的取值有多个，但在一般的浏览器中，只有一个 none 值可以使用。当使用 display 属性隐藏元素时，不但元素看不见，而且元素也将退出当前的页面布局层，不占用任何空间。

visibility 属性有时也被分类为定位属性，它控制定位的元素是否可见，取值包括：visible（可见）、hidden（隐藏）和 inherit（继承），默认值为 inherit。与 display 属性的不同之处在于：当隐藏元素时，仍然为元素保留原有的显示空间。

静态设置这两个属性并没有什么实际意义，但当动态更改这两个属性时，却可以获得很多实用的效果，详细信息请参见本书第 9 章。

8.3.6 列表属性

列表属性用于设置网页中列表的格式，例如可以设置图像作为项目符号。CSS 中的列表属性包括 list-style、list-style-image、list-style-position 以及 list-style-type。

list-style-image 属性使网页设计者可以指定图片作为列表项目的符号，取值为 url(imageurl) | none，默认值为 none。

list-style-position 属性可以设置列表元素标记的位置，取值可以是 inside 或 outside，默认值是 outside。该值指定了相对于列表中其它文本的位置——如果选择 outside，标记就按

规定出现在所有列表元素的外部；如果选择 inside，标记就位于列表元素的文本内部。

list-style-type 属性可以用来设置项目符号和编号的样式，取值如表 8.2 所示：

表8.2 list-style-type属性的取值

样　式	说　明
disc	默认值：实心黑点
circle	空心圆圈
square	方形黑块
decimal	十进制数（1，2，3，4 等等）
lower-roman	小写罗马数字（I，ii，iii，iv 等等）
upper-roman	大写罗马数字（I，II，III，IV，V 等等）
lower-alpha	小写字母（a、b、c、d 等等）
upper-alpha	大写字母（A、B、C、D 等等）
none	无

list-style 属性用于一次性地指定以上的 list-style-image、list-style-type 和 list-style-position 属性（不限顺序）。如果同时指定了 list-style-type 和 list-style-image 属性，则只有当浏览器不能显示图片作为项目符号时，list-style-type 才生效。

以下示例显示了列表属性的用法，代码如下：

```
<HTML>
<HEAD>
<TITLE>列表属性示例</TITLE>
<STYLE>
<!--
.UL-inside {list-style:url(./images/balloon.gif) inside}
.UL-outside{list-style:url(./images/balloon.gif)}
OL {list-style-type:upper-roman}
-->
</STYLE>
</HEAD>
<BODY>
<UL class=UL-inside> <LI>项目 1 <LI>项目 2 </UL>
<UL class=UL-outside> <LI>项目 1 <LI>项目 2 </UL>
<OL> <LI>项目 1 <LI>项目 2 </OL>
</BODY>
</HTML>
```

此示例的显示效果如图 8.13 所示。

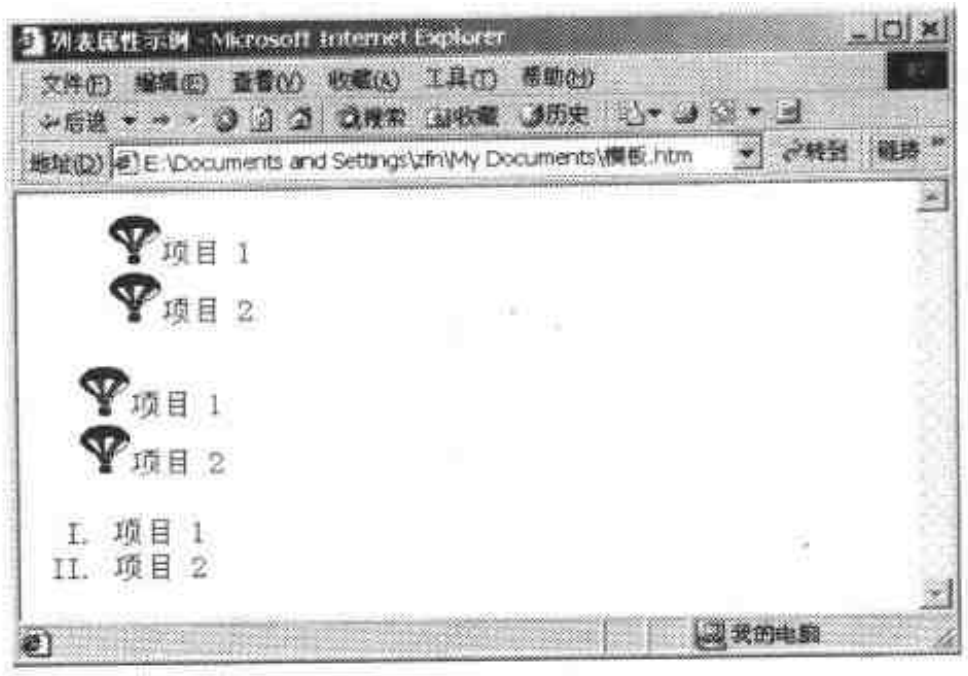


图 8.13 列表属性示例

8.3.7 鼠标属性

cursor 属性用于设置在对象上面移动的鼠标指针显示的指针类型，取值如表 8.3 所示（在第 9 章的示例中有多处用到此属性）。

表8.3 cursor属性的取值

值	含 义
auto	浏览器基于当前文本决定显示哪种指针
crosshair	简单十字形
default	随平台而定的默认指针（通常为箭头）
hand	手形
move	指示某物被移动的交叉箭头
*-resize	指示边缘被移动的箭头（*可以是 n、ne、nw、s、se、sw、e 以及 w，分别代表北、东北、西北、南、东南、西南、东以及西等方向）
text	编辑文本指针（通常为 I 形）
wait	指示程序正忙、用户需要等待的沙漏图标或监视图标
help	指示用户可以得到帮助的问号图标

8.4 过滤器属性

过滤器（filter）是 CSS 的一种扩充，它能够将特定的效果应用于文本容器、图片或其他对象。例如，可以创建阴影效果、模糊效果、翻转效果等。通过与脚本（例如 JavaScript）进行交互，应用可视化过滤器还可以创建出更复杂美观的交互式网页（有关内容请参见下一章）。

8.4.1 属性列表

过滤器效果是通过 filter 样式表属性定义的，格式如下：

```
filter:过滤器名称（参数）
```

其中的参数用于控制特定的过滤器效果。例如，如果要为 IMG 标记符定义透明度效果，可以使用以下样式定义：IMG {filter: alpha(Opacity=80)}。其中，Opacity=80 是参数指定，用于控制透明度。

也可以同时指定多个过滤器效果，此时只需将不同的过滤器用空格分隔即可。例如，以下样式定义为 IMG 标记符同时应用了透明度效果和垂直翻转效果：IMG{filter: alpha(Opacity=80)　flipV()}。

与大多数 CSS 属性不同，可视化过滤器属性只能应用于 HTML 控件元素上。所谓 HTML 控件元素是指它们在网页上定义了一个矩形空间，浏览器窗口可以显示这些空间。合法的 HTML 控件元素包括：BODY、BUTTON、DIV、IMG、INPUT、MARQUEE、SPAN、TABLE、TD、TEXTAREA、TH。

表 8.4 列出了可以在 Internet Explorer 4.0 和 Internet Explorer 5.0 中使用的各种过滤器效果和相关的说明。

表8.4　IE中的过滤器效果

过 滤 器	语　法	说　明
alpha	HTML: filter:alpha(opacity=opacity,finishOpacity=finishopacity,style=style,startX=startx,startY=starty,finishX=finishX,finishY=finishY) 脚本: object.style.filter="alpha(*properties*)"	设置透明度效果。参数 opacity 表示透明度，取值为 0~100；参数 finishopacity 用于设置渐变透明度效果的结束透明度；style 参数表示透明区域的形状特征，取值为 0、1、2、3；其他几个参数表示渐变效果的开始和结束坐标
blendTrans	HTML: filter:blendTrans(duration=duration) 脚本: object.style.filter="blendTrans(*properties*)" 方法: apply, play, stop	创建淡入或淡出效果。参数 duration 表示效果持续的时间。方法 apply 用于应用过滤器效果；方法 play 用于运行过滤器效果；方法 stop 用于终止过滤器效果
blur	HTML: filter:blur(add=true|false,direction=direction,strength=strength) 脚本: object.style.filter="blur(*properties*)"	设置模糊效果。参数 add 指定图片是否被改变成印象派的模糊效果，取值为布尔值；参数 direction 用于设置模糊效果的方向，方向为顺时针，取值为任意值，但只能显示从 0 度开始递增 45 度的 8 个方向；参数 strength 设置模糊效果所影响的像素数，值为整数

续表

过 滤 器	语 法	说 明
chroma	HTML: filter:chroma(color=color) 脚本: object.style.filter="chroma(*properties*)"	将指定颜色设置为透明。参数 color 用于指定要作为透明色的特定颜色，其取值可以是任意颜色值
dropShadow	HTML: filter:dropShadow(color=color,offX=offx,offY=offy, positive=true|false) 脚本: object.style.filter="dropShadow(*properties*)"	设置阴影效果。参数 color 表示阴影的颜色；参数 offX 和 offY 表示阴影的偏移量，取值为像素数；参数 positive 用于为透明对象指定阴影
flipH	HTML: filter:flipH 脚本: object.style.filter="flipH"	设置水平翻转效果
flipV	HTML: filter:flipV 脚本: object.style.filter="flipV"	设置垂直翻转效果
glow	HTML: filter:glow(color=color,strength=strength) 脚本: object.style.filter="glow(*properties*)"	设置发光效果。参数 color 用于设置发光效果的颜色；参数 strength 用于设置发光效果的强度，取值为 0~255 之间的整数
gray	HTML: filter:gray 脚本: object.style.filter="gray"	去除可视对象的颜色信息，使其变为灰度显示
invert	HTML: filter:invert 脚本: object.style.filter="invert"	反转可视对象的色调、饱和度和亮度，创建底片效果
light	HTML: filter:light 脚本: object.style.filter="invert" *方法*: addAmbient(iRed,iGreen,iBlue,iStrength)、 addCone(iX1,iY1,iZ1,iX2,iY2,iRed,iGreen,iBlue,iStrength,iSpread)、 addPoint(iX,iY,iZ,iRed,iGreen,iBlue,iStrength)、 changeColor(iLightNumber,iRed,iGreen,iBlue,bAbsolute)、changeStrength(iLightNumber,iStrength, bAbsolute)、clear()、moveLight(iLightNumber,iX,iY, iZ,bAbsolute)	通过使用过滤器的各种方法模拟光源在可视对象上的投影。方法 addAmbient 用于将环境光添加到对象；方法 addCone 用于将锥形光添加到对象；方法 addPoint 用于添加点光源；方法 changeColor 用于改变光的颜色；方法 changeStrength 用于改变光的强度；方法 clear 用于删除与指定过滤器关联的所有光；方法 moveLight 用于在页面上移动光效果。 iRed 表示红色值，iGreen 表示绿色值，iBlue 表示蓝色值，取值都为 0~255 的整数；iStrength 表示光强度，取值为整数；iSpread 表示光传播的角度，取值是 0~90 的整数；iLightNumber 表示光的标识号；bAbsolute 表示是否采用绝对值
mask	HTML: filter:mask(color=color) 脚本: object.style.filter="mask(*properties*)"	设置透明膜效果。参数 color 表示膜的颜色
revealTrans	HTML: filter:revealTrans(duration=duration,transition=transition) 脚本: object.style.filter="revealTrans(*properties*)" 方法: apply, play, stop	设置转换效果，显示或隐藏可视对象。参数 duration 表示效果持续的时间，单位为秒；参数 transition 表示转换的类型，取值为 0~23。方法 apply 用于应用过滤器效果；方法 play 用于运行过滤器效果；方法 stop 用于终止过滤器效果
shadow	HTML: filter:shadow(color=color,direction=direction) 脚本: object.style.filter="shadow(*properties*)"	设置阴影效果（与 dropShadow 不同）。参数 color 表示阴影的颜色；参数 direction 表示阴影的方向，取值与 blur 过滤器相同

续表

过 滤 器	语 法	说 明
wave	HTML: filter:wave(add=true|false,freq=freq,lightStrength=lightStrength,phase=phase,strength=strength) 脚本: object.style.filter="wave(*properties*)"	设置波纹效果。参数 add 表示是否按正弦波形显示；参数 freq 用于设置波形的频率；参数 lightStrength 用于设置波形的光影效果，取值为 0~100 的整数；参数 phase 用于设置波形开始时的偏移量，取值为 0~100 的整数；参数 strength 用于设置波形的振幅
xray	HTML: filter:xray 脚本: object.style.filter="xray"	设置 X 光效果

8.4.2 效果示例

以下两个示例分别显示了各种过滤器效果（blendTrans、ligh 和 revealTrans 效果通常需要使用脚本，示例请参见第 9 章）。

1. 示例 1

以下示例对图片应用了一些过滤器效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>过滤器效果之一</TITLE>
  <STYLE>
<!--
 IMG.alpha{filter:alpha(Opacity=80,Style=1);}
 IMG.chroma{filter:chroma(color=black);}
 IMG.flipH{filter:flipH;}
 IMG.flipV{filter:flipV;}
 IMG.gray{filter:gray;}
 IMG.invert{filter:invert;}
 IMG.wave{filter:wave(freq=3,lightStrength=20,phase=25;strength=20);
}
 IMG.xray{filter:xray;}
-->
</STYLE>
</HEAD>
<BODY>
  <table align="center" width=80%>  <!--使用表格辅助排版-->
    <tr>
```

```
    <td colspan="4" align="center"><IMG src="./images/construction
.gif"></td>
  </tr>
  <tr>
    <td colspan="4" align="center">原图</td>
  </tr>
  <tr>
    <td align="center"><IMG  class="alpha" src="./images/construction
.gif"></td>
    <td align="center"><IMG  class="chroma" src="./images/
construction.gif"></td>
    <td align="center"><IMG  class="flipH" src="./images/construction
.gif"></td>
    <td align="center"><IMG  class="flipV" src="./images/construction
.gif"></td>
  </tr>
  <tr>
    <td align="center">alpha 效果</td>
    <td align="center">chroma 效果</td>
    <td align="center">flipH 效果</td>
    <td align="center">flipV 效果</td>
  </tr>
  <tr>
    <td align="center"><IMG  class="gray" src="./images/construction
.gif"></td>
    <td align="center"><IMG  class="invert" src="./images/construction
.gif"></td>
    <td align="center"><IMG  class="wave" src="./images/construction
.gif"></td>
    <td align="center"><IMG  class="xray" src="./images/construction
.gif"></td>
  </tr>
  <tr>
    <td align="center">gray 效果</td>
    <td align="center">invert 效果</td>
    <td align="center">wave 效果</td>
    <td align="center">xray 效果</td>
  </tr>
</table>
```

```
</BODY>
</HTML>
```

此示例的效果如图 8.14 所示。

图 8.14　过滤器效果之一

2. 示例 2

以下示例对文本应用了一些过滤器效果，代码如下：

```
<HTML>
<HEAD>
<TITLE>过滤器效果之二</TITLE>
<STYLE>
<!--
.blur{filter:blur(strength=6,direction=135);width=800}
.dropShadow{filter:dropShadow(color=gray,offX=3,offY=3);width=800}
.glow{filter:glow(color="#ff7f00",strength=10);width=800}
.mask{filter:mask(color="#238e68");width=800}
.shadow{filter:shadow(color=gray,direction=135);width=800}
-->
</STYLE>
</HEAD>
<BODY>
<DIV align="center">
  <H1>此段文本未使用效果</H1>
  <H1 class="blur">此段文本使用了 blur 效果</H1>
```

```
    <H1 class='dropShadow'>此段文本使用了 dropShadow 效果</H1>
    <H1 class='glow'>此段文本使用了 glow 效果</H1>
    <H1 class="mask">此段文本使用了 mask 效果</H1>
    <H1 class='shadow'>此段文本使用了 shadow 效果</H1>
</DIV>
</BODY>
</HTML>
```

此示例的效果如图 8.15 所示。

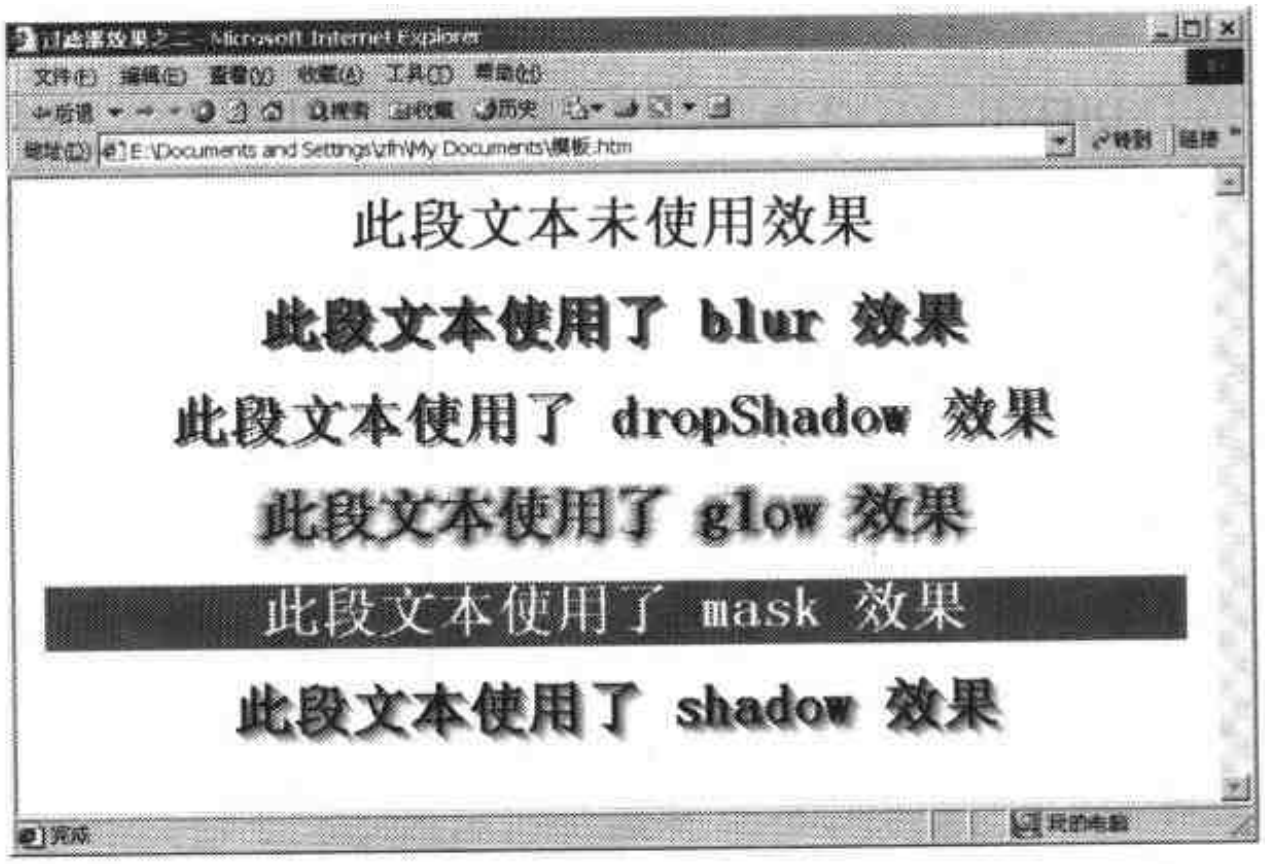


图 8.15 过滤器效果之二

第9章 DHTML 应用

上一章介绍了 DHTML 的基本概念，同时着重说明了 CSS 技术。本章将把 CSS 与 JavaScript 结合起来，真正实现 DHTML 的效果。通过本章的学习，读者将能把前面章节学习的内容具体地应用到实际的网页制作中，达到学以致用的目的。本章中提供的大量示例多数改编自各大网站的页面，因此具有很强的参考价值，读者可以直接修改应用到自己的网页中。

本章主要内容包括：

- style 对象
- 动态定位与显示
- 动态过滤器效果
- DHTML 小脚本应用

9.1 style 对象

9.1.1 概述

如果用户要在客户端脚本中访问 CSS 属性，必须首先访问到相应的 HTML 元素对象（请参考本书 4.2.2 节的内容），然后使用该对象的 style 属性访问具体的 CSS 属性。而 style 属性本身也是对象，称为 style 对象，可通过它的属性访问 CSS 属性。

style 对象的属性包括附录 2 中列出的所有 DHTML 特性。此外，style 对象还具有以下属性：pixelHeight、pixelWidth、pixelTop、pixelBottom、pixelLeft、pixcelRight、posHeight、posWidth、posTop、posBottom、posLeft、posRight。其中，以 pixel 开头的属性都表示以像

素为单位的对应位置；而以 pos 开头的属性都表示以浮点数计算的对应位置（相应的 height、width、top、bottom、left 和 right 的值都是字符串）。

通过使用 style 对象，我们可以动态地更改相应 HTML 元素的 CSS 样式，从而获得需要的效果。

9.1.2 示例

本小节使用三个示例来说明 style 对象的用法。在这三个示例中，都动态地更改了对象的样式，因此反映了基本的 DHTML 特性。

1. 示例 1

本示例实现了动态更改文字颜色和鼠标指针形状的效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>DHTML 示例</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var toggle = 0;  //全局变量

function changeColor()
{
if (toggle==0){
  document.all.item("changingText").style.color="red";
  toggle = 1;
}
else{
  document.all.item("changingText").style.color="black";
  toggle = 0;
}
}
//-->
</SCRIPT>
</HEAD>
<BODY>
  <H1    id="changingText"    onclick=changeColor()    onmouseover    =
"this.style.cursor='hand'" onmouseout="this.style.cursor='default'" >
请单击这段文字！</H1>
```

```
</BODY>
</HTML>
```

本示例的效果为：当用户将鼠标指针移动到文字上时，指针变为“手”形，如果单击文字，则文字变为红色，如图 9.1 所示；再次单击则文字恢复为黑色。

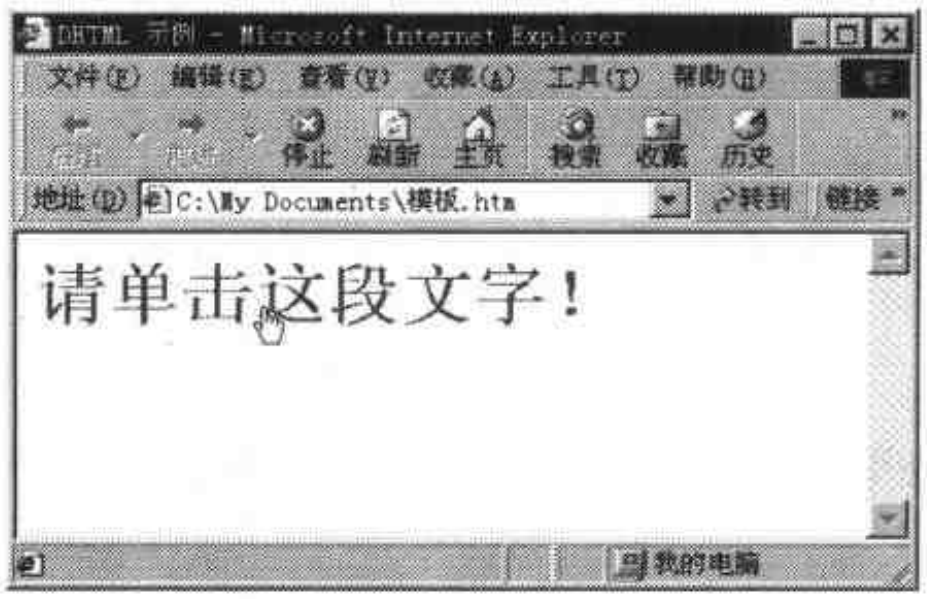


图 9.1　改变鼠标指针和文字颜色

2. 示例 2

本示例显示出一个动态时钟，并根据当前时间动态设置格式，代码如下：

```
<HTML>
<HEAD>
  <TITLE>DHTML 示例</TITLE>
<SCRIPT LANGUAGE=JavaScript TYPE='text/javascript'>
<!--
function acquireTime()
{  today=new Date();  //获取当前日期
with(today)
{
document.form1.timer.value=getHours()+"    点    "+getMinutes()+"    分
"+getSeconds()+"秒";
if(getHours()>=0 && getHours()<=7)
  document.form1.timer.style.color="red";  //更改颜色
else
  document.form1.timer.style.color="black";  //恢复颜色
}
}
//-->
```

```
</SCRIPT>
<HEAD>
<BODY onLoad="setInterval('acquireTime()',1000)">
<DIV align=center>
<FORM name=form1>
  <B>现在时间：</B><INPUT NAME="timer" SIZE=11>
</FORM>
</DIV>
</BODY>
</HTML>
```

本示例的显示效果如图 9.2 所示，如果当前时间在 0~7 点之间，则时间显示为红色。

图 9.2 动态时钟示例

3. 示例 3

本示例通过动态更改 H1 标记符的内容和样式，获得了一种动态变换的广告效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>DHTML 示例</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function advertise()
{
randomIndex=Math.round(Math.random()*3)%3    // 随机生成 0、1、2
switch(randomIndex)
{
case 0:
  document.all.ad.innerText="广告标语……"
    //也可以直接用 ad.innerText="广告标语……"
```

```
    document.all.ad.style.fontFamily="楷体_GB2312"
    document.all.ad.style.color="#8b0000"
    document.all.ad.style.backgroundColor="#f0f8ff"
    break;
  case 1:
    document.all.ad.innerText="太好了！！！"
    document.all.ad.style.fontFamily="黑体"
    document.all.ad.style.color="#deb887"
    document.all.ad.style.backgroundColor="#808080"
    break;
  default:
    document.all.ad.innerText="真没劲！！！"
    document.all.ad.style.fontFamily="隶书"
    document.all.ad.style.color="#228b22"
    document.all.ad.style.backgroundColor="#ff8c00"
  }
  }
  //-->
  </SCRIPT>
  <HEAD>
  <BODY onLoad="setInterval('advertise()',1000)">
  <DIV align="center">
    <H1 id="ad" style="width:400px" align="center"></H1>
  </DIV>
  </BODY>
  </HTML>
```

这段代码的效果是：每隔一秒钟，在 H1 标记符中随机显示文字并采用一定的样式。例如，图 9.3 和图 9.4 就是在不同时刻的两种状态。

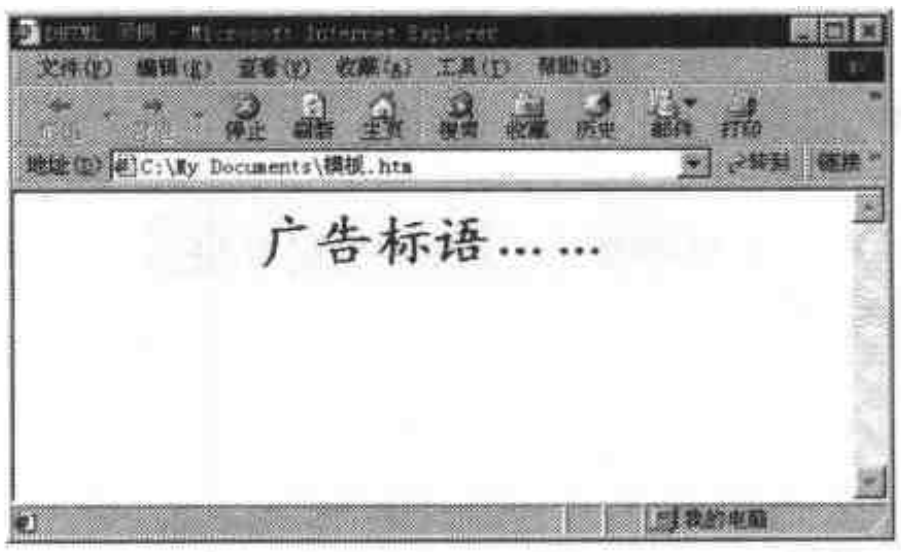


图 9.3　显示效果 1

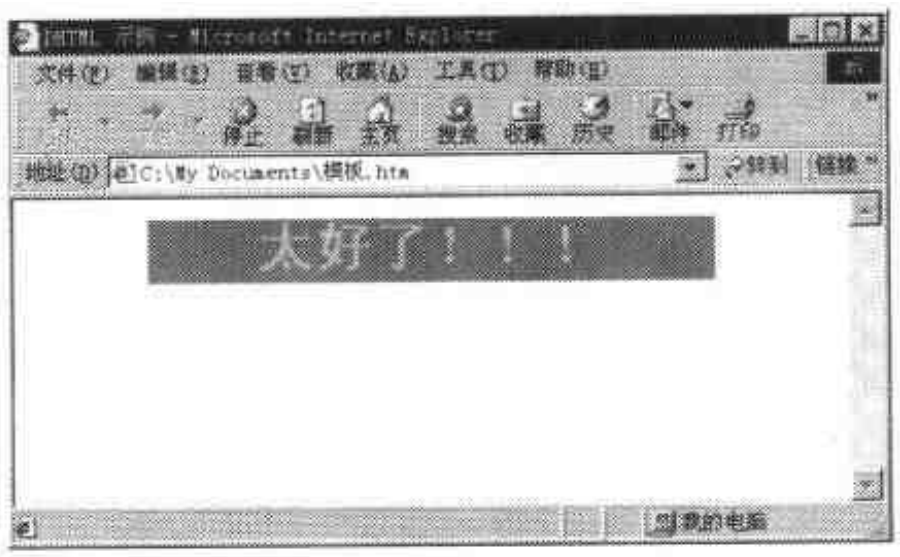


图 9.4 显示效果 2

注意：在这个示例中，用到了另外一种常用的引用 HTML 元素对象的方法，即直接使用 id 属性（而不是使用 all.item() 方法）。在 HTML 标记符中指定了 id 属性之后，就可以用 document.all.idName 或直接用 idName 来访问该元素对象，这是因为 id 代表了对象在文档中的唯一标识。

9.2 动态定位与显示

除了更改对象的格式属性来获得动态格式效果以外，在 DHTML 中最常见的用法是更改对象的定位属性和显示属性，以便使对象动态移动或动态显示和隐藏，从而获得动画或动态界面的效果。实际上，所谓的“层”技术就是通过动态更改对象的定位属性和显示属性来实现的（虽然本书并不强调该术语，但“层”确实很好地反映了动态显示的特点，本节中的很多示例都应用到了这一点）。

9.2.1 动态定位

所谓动态定位就是指在网页下载之后动态改变对象的位置，从而获得动画的效果。以下使用两个示例来说明如何应用动态定位获得动画效果。

1. 示例 1

本示例通过动态更改对象的定位属性而获得移动的效果（请与第 7 章中 7.6.3 节的示例对比），代码如下：

```
<HTML>
<HEAD>
```

```
<TITLE>移动效果</TITLE>
<STYLE>
<!--
  #scrollImg {position:absolute; top:125px; left:25px}
  #text {position:absolute; top:128px; left:500px}
-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var start-25;  //移动的起点位置
var end=380;  //移动的终点位置
var step=10;  //移动的步长，减小此数可以使动画更平滑
function changePosition()
{
if(document.all.scrollImg.style.pixelLeft<end)    // 注意使用的是
pixelLeft 而非 left
{
  document.all.scrollImg.style.pixelLeft += step;
}
else
{
  document.all.scrollImg.style.pixelLeft=start;
  document.all.scrollImg.style.pixelLeft += step;
}
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad=setInterval("changePosition()",40);>
<DIV ID="scrollImg">
   <IMG src="images/hand.gif">
</DIV>
<DIV ID="text">
  <A HREF="javascript:void(0)">
    <FONT size=+1 face="楷体_GB2312">快来看呀！</FONT>
  </A>
</DIV>
</BODY>
</HTML>
```

本示例的效果如图 9.5 所示。

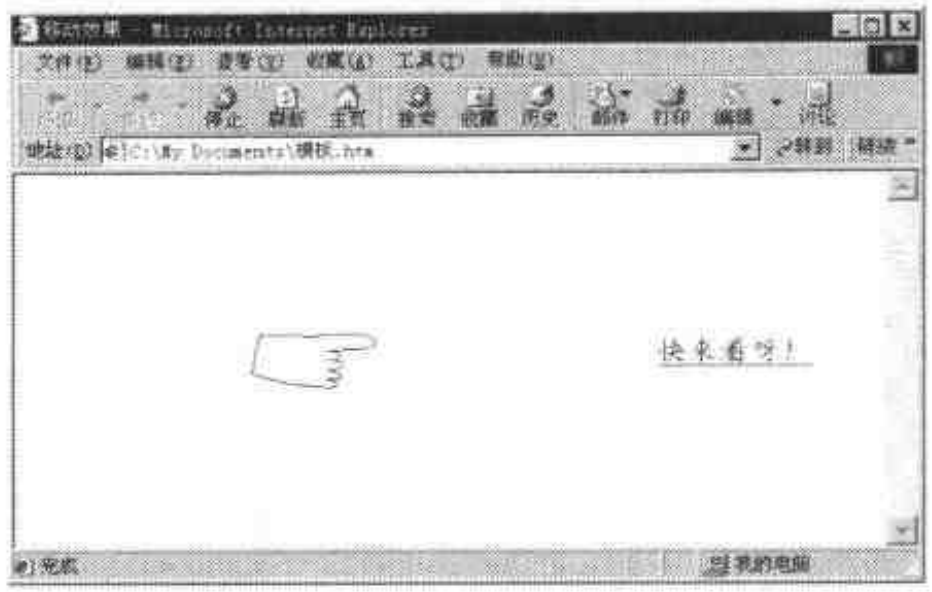


图 9.5 移动效果

2. 示例 2

本示例要复杂一些，但实现的功能更强大——它获得了一种页面动画广告的效果，另外也显示了如何编写跨浏览器平台的脚本，其代码如下：

```
<HTML>
<HEAD>
  <TITLE>前景动画</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var brOK=false;
var mie=false;  //判断是否是 IE
var aver=parseInt(navigator.appVersion.substring(0,1));
var aname=navigator.appName;

function checkbrOK()
{
if(aname.indexOf("Internet Explorer") != -1)
{  if(aver>=4)
     brOK=navigator.javaEnabled();  //浏览器版本应大于等于 4.0
   mie=true;  //浏览器是 IE
}
 if(aname.indexOf("Netscape") != -1)
{
  if(aver>=4)
    brOK=navigator.javaEnabled();  //浏览器版本应大于等于 4.0
```

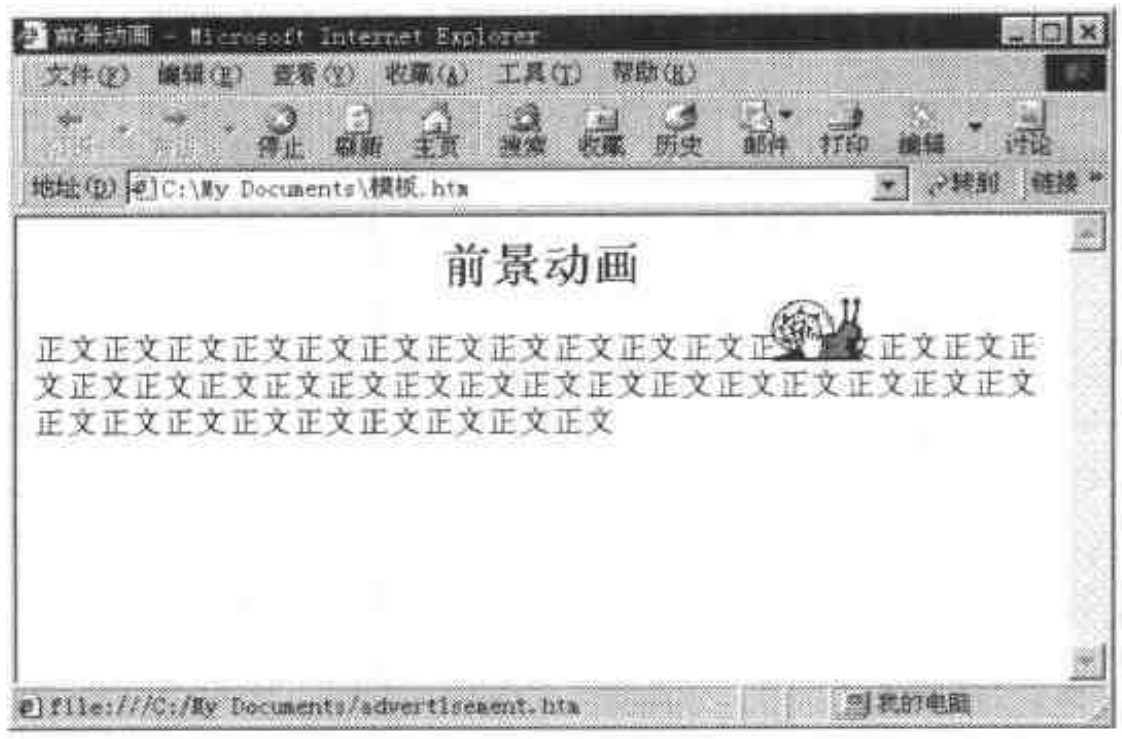
```
}
}

var vmin-2;
var vmax=5;
var vr=2;
var timer1;

function Obj(objname,width,height)  //定义一个对象 Obj
{
 this.named-objname;
 this.vx=vmin+vmax*Math.random();
 this.vy=vmin+vmax*Math.random();
 this.w=width;
 this.h=height;
 this.xx=0;
 this.yy=0;
 this.timer1=null;
}

function moveobj(objname)  //移动对象
{
 if(brOK)
  {
   eval("obj="+objname);
   if(!mie)
    {
     pageX=window.pageXOffset;
     pageW=window.innerWidth;
     pageY=window.pageYOffset;
     pageH=window.innerHeight;
    }
  else
   {
    pageX=window.document.body.scrollLeft;
    pageW=window.document.body.offsetWidth-8;
    pageY=window.document.body.scrollTop;
    pageH=window.document.body.offsetHeight;
   }
```

```
obj.xx=obj.xx+obj.vx;
obj.yy=obj.yy+obj.vy;

obj.vx+=vr*(Math.random()-0.5);
obj.vy+=vr*(Math.random()-0.5);
if(obj.vx>(vmax+vmin))  obj.vx=(vmax+vmin)*2-obj.vx;
if(obj.vx<(-vmax-vmin))  obj.vx=(-vmax-vmin)*2-obj.vx;
if(obj.vy>(vmax+vmin))  obj.vy=(vmax+vmin)*2-obj.vy;
if(obj.vy<(-vmax-vmin))  obj.vy=(-vmax-vmin)*2-obj.vy;

if(obj.xx<=pageX)
  {
   obj.xx=pageX;
   obj.vx=vmin+vmax*Math.random();
  }
if(obj.xx>=pageX+pageW-obj.w)
  {
   obj.xx=pageX+pageW-obj.w;
   obj.vx=-vmin-vmax*Math.random();
  }
if(obj.yy<=pageY)
  {
   obj.yy=pageY;
   obj.vy=vmin+vmax*Math.random();
  }
if(obj.yy>=pageY+pageH-obj.h)
  {
   obj.yy=pageY+pageH-obj.h;
   obj.vy=-vmin-vmax*Math.random();
  }

if(!mie)
   {
    eval('document.'+obj.named+'.top ='+obj.yy);
    eval('document.'+obj.named+'.left='+obj.xx);
   }
else
   {
```

```
      eval('document.all.'+obj.named+'.style.pixelLeft='+obj.xx);
      eval('document.all.'+obj.named+'.style.pixelTop ='+obj.yy);
     }
   obj.timer1=setTimeout("moveobj('"+obj.named+"')",100);
}
}

function stopme(objname)  //停止移动对象
{
if(brOK)
{
  eval("obj="+objname);
  if(obj.timer1!=null) clearTimeout(obj.timer1)
}
}
var obj1;
function pagestart()
{
checkbrOK();
obj1=new Obj("obj1",60,80);  //新建一个 Obj 对象
if(brOK)
  moveobj("obj1");
}
//-->
</SCRIPT>
<BODY onLoad="pagestart();" onUnload="if(brOK) stopme('obj1');">
<DIV ID="obj1" STYLE="position:absolute; width:110; height:76;">
  <DD><A    HREF="advertisement.htm"><IMG    SRC="./images/snail.gif"
WIDTH="64" HEIGHT="54" BORDER="0"></A></DD>
</DIV>
<H2 align=center>前景动画</H2>
<P>正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正
文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文
</P>
</BODY>
</HTML>
```

本示例的效果如图 9.6 所示——一个图片在页面范围内“漂浮”，并且该图片对应于一个超链接，单击它可以跳转到相应的页面。如果将这种效果应用到网站的主页上，则可以

将最重要的一条新闻或消息作为图片的链接目标，并用适当的图片表达该新闻或消息，从而使浏览者能够首先注意到该内容。



前景动画

正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正
文正文正文正文正文正文正文正文正文正文正文正文正文正文正文正文
正文正文正文正文正文正文正文正文正文

图 9.6　前景动画

9.2.2　显示属性

显示属性 display 和 visibility 可以实现多种特殊的效果，可以称得上是 DHTML 的精华所在。本节的第一个示例首先显示了这两种显示属性的区别，之后的几个示例分别获得了多种实用效果，可以直接应用于读者自己的网页。

1．示例 1

本示例显示了 display 属性和 visibility 属性的区别，代码如下：

```
<HTML>
<HEAD>
  <TITLE>显示和隐藏属性</TITLE>
</HEAD>
<BODY>
<H1 align=center>显示和隐藏</H1>
<TABLE>
<TR>
  <TH>使用 display 属性
  <TH>使用 visibility 属性
</TR>
<TR>
```

```
  <TD    width=50%    valign=top><P    style="background-color:#dddddd"
onClick="document.all.dspl.style.display=(document.all.dspl.style.di
splay=='none'?'':'none')">单击此处可以动态隐藏或显示信息</P>
<P id="dspl" style="display:none"><B>此信息动态隐藏或显示；此信息动态隐藏或
显示；此信息动态隐藏或显示；</B></P>
<P>已经显示的其他信息；已经显示的其他信息；已经显示的其他信息；</P>
  <TD    width=50%    valign=top><P    style="background-color:#dddddd"
onClick="document.all.vsb.style.visibility=(document.all.vsb.style.v
isibility=='hidden'?'':'hidden')">单击此处可以动态隐藏或显示信息</P>
<P id="vsb" style="visibility:hidden"><B>此信息动态隐藏或显示；此信息动态
隐藏或显示；此信息动态隐藏或显示</B></P>
<P>已经显示的其他信息；已经显示的其他信息；已经显示的其他信息；</P>
</TR>
</TABLE>
</BODY>
</HTML>
```

本示例的效果为：初始状态时如图 9.7 所示——使用 display 属性的一段信息事先没有预留空间，而使用 visibility 属性的信息则已经预留了空间；当分别单击兰色条中的文字时，原先没有显示的信息都显示出来，如图 9.8 所示——使用 display 属性的信息新分配了一段显示空间，而使用 visibility 属性的信息直接使用预留的空间。

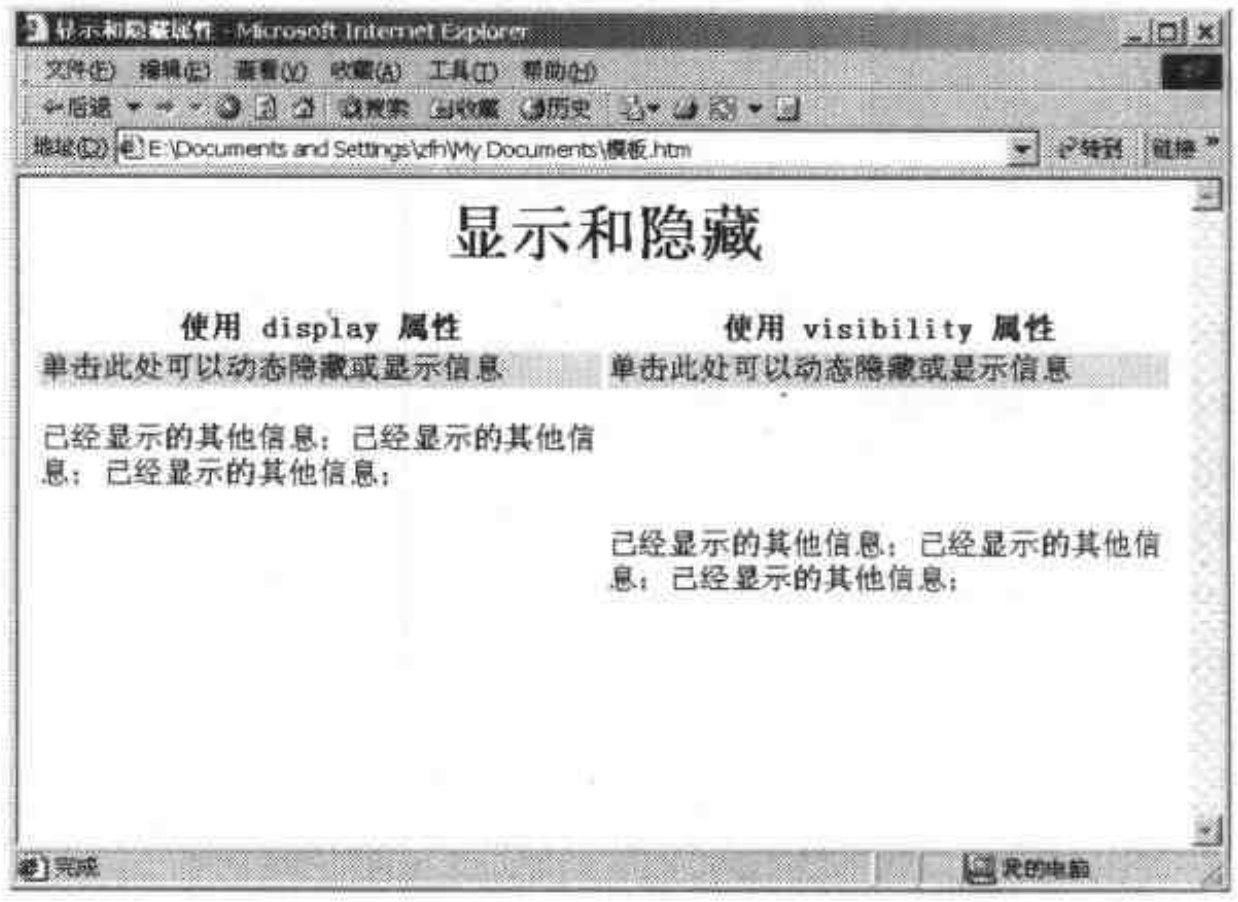


图 9.7　隐藏信息

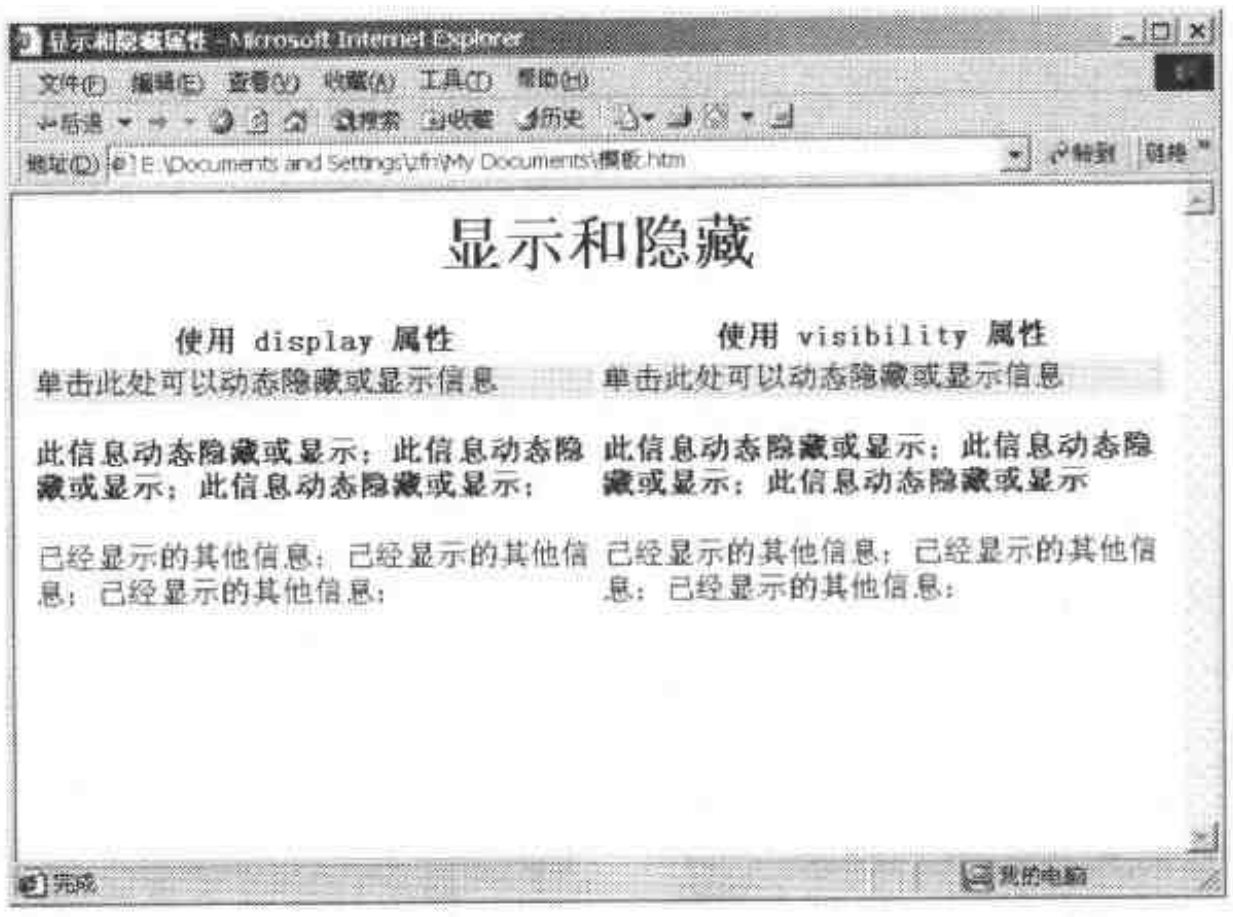


图 9.8 显示信息

2. 示例 2

本示例将 7.6.2.3 节的翻滚图效果改为用 visibility 属性来实现，代码如下：

```
<HTML>
<HEAD>
  <TITLE>翻滚图</TITLE>
</HEAD>
<BODY>
<H2 align=center>请将鼠标指针移动到图像上……</H2>
<DIV id=img1 style="position:absolute; top:100px; left:100px"
onMouseOver = "document.all.img1.style.visibility='hidden';
              document.all.img2.style.visibility='visible'"
   onMouseOut = "document.all.img2.style.visibility='hidden';
                 document.all.img1.style.visibility='visible'">
<A HREF=javascript:void(0)>
    <IMG SRC = "./images/cactus.jpg" WIDTH = 320 HEIGHT = 240>
</A>
</DIV>
<DIV id=img2
     style="position:absolute; top:100px; left:100px; visibility='hidden'"
     onMouseOver = "document.all.img1.style.visibility='hidden';
                    document.all.img2.style.visibility='visible'"
     onMouseOut = "document.all.img2.style.visibility='hidden';
```

```
                    document.all.img1.style.visibility='visible'">
<A HREF=javascript:void(0)>
    <IMG SRC = "./images/sun.jpg" WIDTH = 320 HEIGHT = 240>
</A>
</DIV>
</BODY>
</HTML>
```

本示例的效果与 7.6.2 节的示例一模一样——初始时显示第一幅图片，当用户将鼠标指针移动到图片上时，将显示另一幅图片，如图 9.9 所示，如果将鼠标指针移出图片，则又将显示第一幅图片。

图 9.9　翻滚图效果

3. 示例 3

本示例实现了一个动态折叠菜单——当浏览者单击菜单条目时，其子菜单会动态显示或隐藏，并且当鼠标指针移动到菜单条目上时，鼠标指针会变成手形，并且每个子菜单条目在鼠标指针位于其上时变为红色。以下是本示例的源代码，效果如图 9.10 所示。

```
<HTML>
<HEAD>
<TITLE>动态折叠菜单</TITLE>
</HEAD>
<STYLE>
<!--
    BODY {font-size:12pt}
```

```
    A {font size:10pt}
    .red {color:red}
    .menu{color:blue; cursor:hand}
    .indent {margin-left:0.3in}
-->
</STYLE>
<SCRIPT LANGUAGE-"JavaScript" TYPE-"text/javascript">
<!--
function menuChange() {
  var src;
  var subId;
  src = window.event.srcElement;
  if (src.className == "menu") {   //判断是否单击了某菜单项。
     subId = "sub" + src.id;
     if (document.all(subId).style.display == "none") {  /* 如果没有显示
子菜单，则显示。*/
        document.all(subId).style.display = "";
     } else {  //如果已经显示子菜单，则折叠。
        document.all(subId).style.display = "none";
     }
  }
}
//-->
</SCRIPT>
<BODY onClick="menuChange();">
<H3>单击一个菜单项则可以打开或折叠菜单……</H3>
<SPAN ID="menu1" CLASS="menu">+ 菜单项 1</SPAN>  <!-- 定义菜单 -->
<DIV ID=submenu1 STYLE="display:None">              <!   定义子菜单-->
    <DIV CLASS="indent">                            <!--定义缩进-->
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
       onmouseout = "this.className = '';">子菜单项 1</A><BR>
<!-- 对象的 className 属性用于访问定义的样式类 -->
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
       onmouseout = "this.className = '';">子菜单项 2</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
       onmouseout = "this.className = '';">子菜单项 3</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
       onmouseout = "this.className = '';">子菜单项 4</A><BR>
```

```
    </DIV>
</DIV>
<BR>
<SPAN ID="menu2" CLASS="menu">+ 菜单项 2</SPAN>
<DIV ID=submenu2 STYLE="display:None">
    <DIV CLASS="indent">
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 1</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 2</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 3</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 4</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 5</A>
    </DIV>
</DIV>
<BR>
<SPAN ID="menu3" CLASS="menu">+ 菜单项 3</SPAN>
<DIV ID=submenu3 STYLE="display:None">
    <DIV CLASS="indent">
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 1</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 2</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 3</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 4</A><BR>
    <A href="javascript:void(0)" onmouseover = "this.className = 'red'"
        onmouseout = "this.className = '';">子菜单项 5</A>
    </DIV>
</DIV>
</BODY>
</HTML>
```

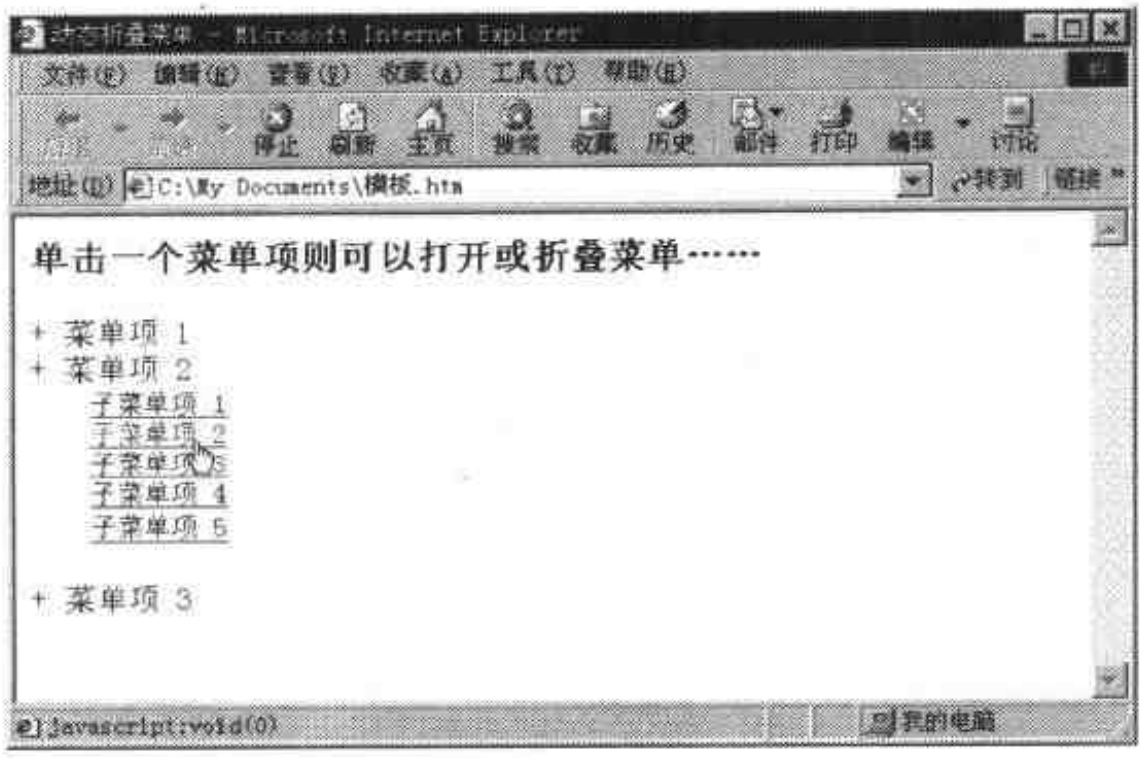


图 9.10 动态折叠菜单

4. 示例 4

本小节的示例实现了一个类似 Outlook 的界面，当浏览者单击某个按钮时，将显示该类的图标，如图 9.11 所示。

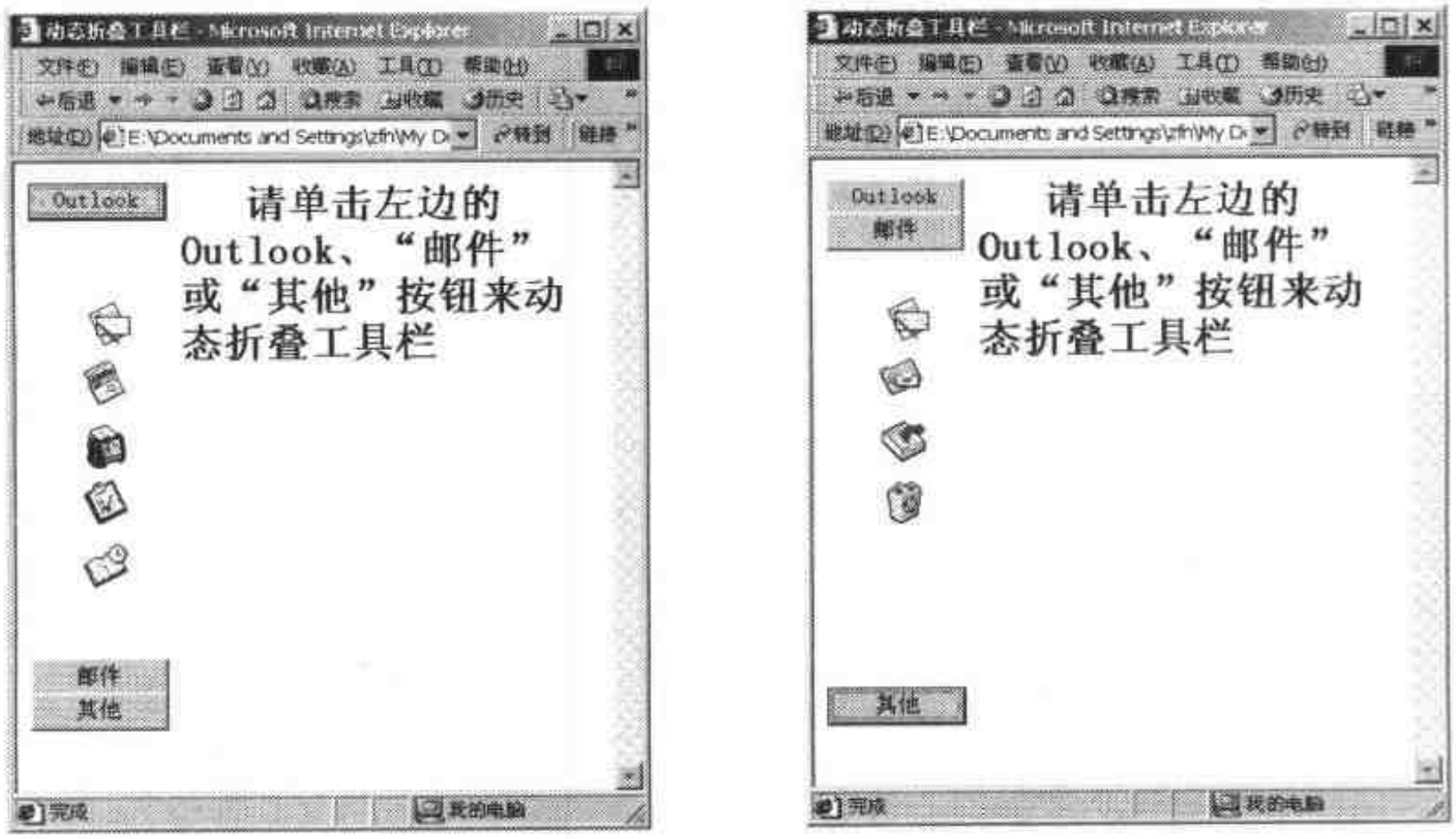


图 9.11 动态折叠工具栏

以下是实现该效果的源代码（主要是通过动态更改对象的 top 和 display 属性来实现的，请注意注释）：

```
<HTML>
<HEAD>
```

```
<TITLE>动态折叠工具栏</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<! -
function RollBar(intIndex)
    {
     OutlookBarVisible("none");  //首先使所有工具栏均不可见
     MailBarVisible("none");
     OtherBarVisible("none");
      switch(intIndex)
      {
          case 1:  //显示 Outlook 工具栏
              cmdOutlook.style.top="0";
              cmdMail.style.top="350";
              cmdOther.style.top="375";
              OutlookBarVisible("");
              break;
      case 2:
          if(cmdMail.style.top == "25px")
           {
              cmdOutlook.style.top = "0";
              cmdMail.style.top = "350";
              cmdOther.style.top = "375";
              OutlookBarVisible("");  //显示 Outlook 工具栏
           }
          else
           {
              cmdOutlook.style.top = "0";
              cmdMail.style.top = "25";
              cmdOther.style.top = "375";
              MailBarVisible("");  //显示“邮件”工具栏
           }
           break;
      case 3:
          if(cmdOther.style.top == "50px")
          {
              cmdOutlook.style.top = "0";
              cmdMail.style.top = "25";
              cmdOther.style.top = "375";
              MailBarVisible("");  //显示“邮件”工具栏
```

```
            }
             else
            {
                cmdOutlook.style.top = "0";
                cmdMail.style.top    "25";
                cmdOther.style.top = "50";
                OtherBarVisible("");  //显示"其他"工具栏
             }
             break;
    }
}

    function OutlookBarVisible(strDisplay)
       {
       imgOutlook1.style.display = strDisplay;   //为每个 Outlook 图标设
置 display 属性
        imgOutlook2.style.display = strDisplay
        imgOutlook3.style.display = strDisplay
        imgOutlook4.style.display = strDisplay
        imgOutlook5.style.display = strDisplay
       }

    function MailBarVisible(strDisplay)
       {
        imgMail1.style.display = strDisplay  //为每个"邮件"图标设置 display
属性
        imgMail2.style.display = strDisplay
        imgMail3.style.display = strDisplay
        imgMail4.style.display = strDisplay
       }

    function OtherBarVisible(strDisplay)
       {
        imgOther1.style.display = strDisplay   //为每个"其他"图标设置
display 属性
        imgOther2.style.display = strDisplay
        imgOther3.style.display = strDisplay
       }
-->
```

```
</SCRIPT>
</HEAD>
<BODY>
<DIV STYLE="position:absolute; width:100; height:400;"> <!-- 指定工具
栏区域 -->
<!-  按钮   >
<INPUT STYLE="position:absolute; top:0; width:100;"
    TYPE="BUTTON" NAME="cmdOutlook" VALUE="Outlook"
    OnClick="RollBar(1)">

<INPUT STYLE="position:absolute; top:350; width:100;"
    TYPE="BUTTON" NAME="cmdMail" VALUE="邮件"
    OnClick="RollBar(2)">

<INPUT STYLE="position:absolute; top:375; width:100;"
    TYPE="BUTTON" NAME="cmdOther" VALUE="其他"
    OnClick="RollBar(3)">

<!-- Outlook 工具栏图片  ->
<IMG SRC="./images/inbox.gif " ID="imgOutlook1"
    STYLE="position:absolute; left:32; top:80">
<IMG SRC="./images/calendar.gif" ID="imgOutlook2"
    STYLE="position:absolute; left:32; top:125">
<IMG SRC="./images/contacts.gif" ID="imgOutlook3"
    STYLE="position:absolute; left:32; top:170">
<IMG SRC="./images/tasks.gif"    ID="imgOutlook4"
    STYLE="position:absolute; left:32; top:215">
<IMG SRC="./images/journal.gif"  ID="imgOutlook5"
    STYLE="position:absolute; left:32; top:260">

<!-- "邮件"工具栏图片-->
<IMG SRC="./images/inbox.gif"  ID="imgMail1"
    STYLE="position:absolute; left:32; top:80; display:none">
<IMG SRC="./images/sentitems.gif "ID="imgMail2"
    STYLE="position:absolute; left:32; top:125; display:none">
<IMG SRC="./images/outbox.gif"  ID="imgMail3"
    STYLE="position:absolute; left:32; top:170; display:none">
<IMG SRC="./images/deleted.gif "  ID="imgMail4"
    STYLE="position:absolute; left:32; top:215; display:none">
```

```
<!-- “其他”工具栏图片 -->
<IMG SRC="./images/mycomputer.gif" ID="imgOther1"
    STYLE="position:absolute; left:32; top:80; display:none">
<IMG SRC="./images/folder.gif" ID="imgOther2"
    STYLE="position:absolute; left:32; top:125; display:none">
<IMG SRC="./images/notes.gif "     ID="imgOther3"
    STYLE="position:absolute; left:32; top:170; display:none">
</DIV>
<DIV STYLE="position:absolute; left:120; text-indent:1cm">
  <H2>请单击左边的 Outlook、“邮件”或“其他”按钮来动态折叠工具栏</H2>
</DIV>
</BODY>
</HTML>
```

9.3 动态过滤器效果

在第 8.4 节中介绍了各种过滤器效果，本节使用三个示例说明如何通过动态更改过滤器属性以及通过使用过滤器的方法来获得动态效果。

9.3.1 示例 1

本示例通过更改 alpha 过滤器的 opacity 属性和 style 属性以便获得不同的透明度效果，代码如下：

```
<HTML>
<HEAD>
  <TITLE>使用 alpha 过滤器</TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var delta=10;
var myOpacity=10, myStyle=0;  //全局变量
function changeOpacity()
{
myOpacity+=delta;
if(myOpacity>100) myOpacity%=100
document.all.alpha1.filters.alpha.opacity=myOpacity;
}
```

```
function changeStyle()
{
myStyle++;
if(myStyle>=4) myStyle%=4;
document.all.alphal.filters.alpha.style=myStyle;
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<DIV align=center>
<H2>单击以下按钮可以更改透明度效果：</H2>
<INPUT type=button value-"更改透明度" onClick=changeOpacity()>
<INPUT type=button value="更改样式" onClick=changeStyle()>
  <IMG id="alpha1" style="filter:alpha(opacity=50,style=0);width=200"
src="./images/forest.jpg">
</BODY>
</HTML>
```

注意：本示例显示了如何引用过滤器的属性——object.filters.filtername.property，即使用对象的 filters 属性引用相应的过滤器名称，然后引用过滤器的属性。

本示例的效果是：当单击“更改透明度”按钮时，图像的透明度会更改；当单击“更改样式”按钮时，图像的透明度样式会更改。例如，图 9.12 和图 9.13 就是处于两种不同状态时的显示情况。

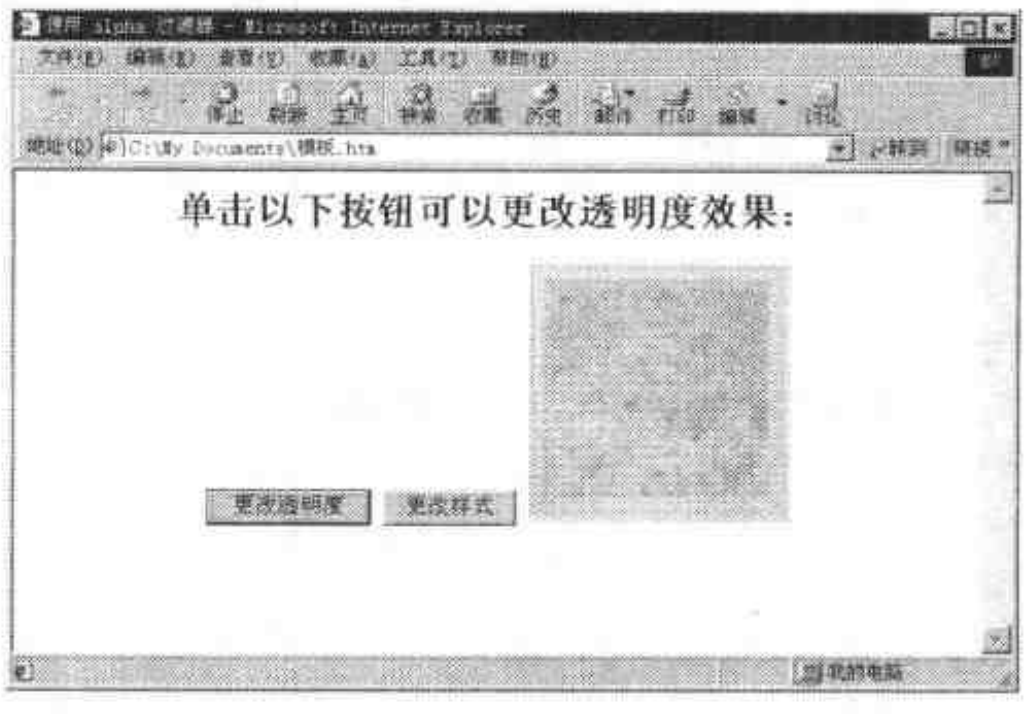


图 9.12　更改透明度

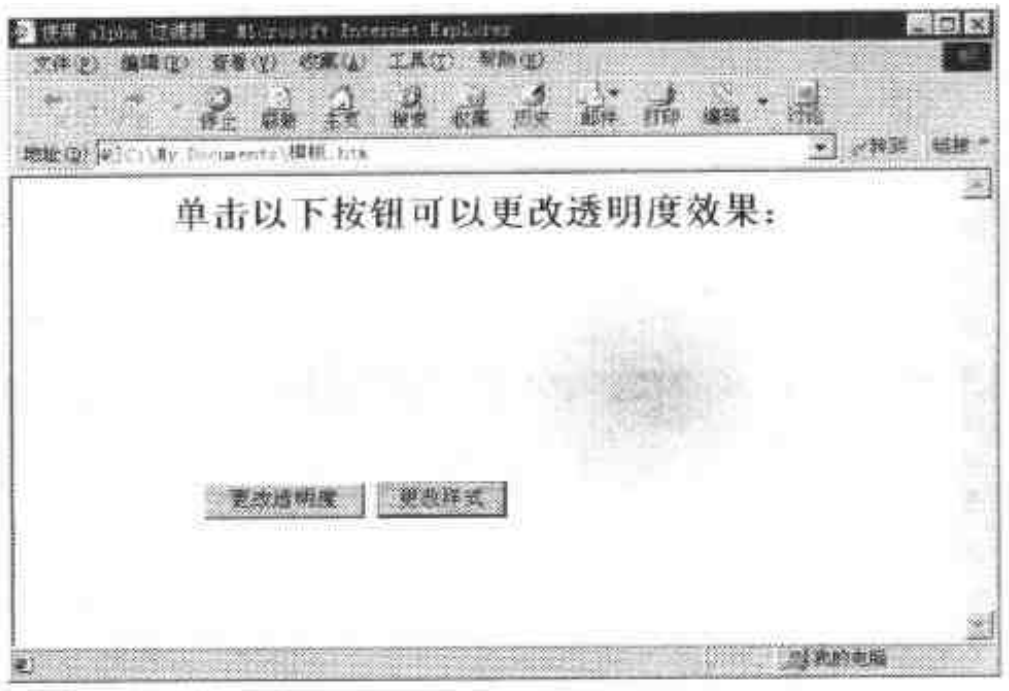


图 9.13 更改样式

9.3.2 示例 2

本示例使用 blendTrans 过滤器创建了淡入效果，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 blendTrans 过滤器</TITLE>
<SCRIPT LANGUAGE = "Javascript" TYPE="text/javascript">
function fadeIn()
{
  myDiv.style.filter="blendTrans(duration=2)";
  myDiv.filters.blendTrans.apply();
  myDiv.style.visibility='visible';  /* 此语句必须使用，否则无法创建出转换
效果。实际上，revealTrans 和 blendTrans 过滤器都是创建一种从可见到不可见（或从
不可见到可见）的过渡效果。*/
  myDiv.filters.blendTrans.play();
}
</SCRIPT>
</HEAD>
<BODY onLoad="fadeIn()">
<!--必须为 DIV 设置宽度以便应用布局…… -->
<DIV ID="myDiv" STYLE="visibility=hidden;width:600">
  <H3>当加载本网页时，所有内容都以淡入的方式逐步显示……</H3>
  <IMG src=./images/MM.jpg width=400>
</DIV>
</BODY>
```

```
</HTML>
```

本示例的效果为：当用户加载网页时，所有网页内容都以从无到有的方式显示，如图 9.14 所示。

图 9.14　淡入效果

9.3.3　示例 3

本示例显示了如何使用 light 过滤器，代码如下：

```
<HTML>
<HEAD>
<TITLE>使用 light 过滤器</TITLE>
<STYLE>
    .aFilter{ filter:light();
          width:400; }
</STYLE>
<SCRIPT LANGUAGE = "Javascript" TYPE="text/javascript">
<!--
function init()
{
  var ix2=myDiv.offsetWidth/2;
  var iy2=myDiv.offsetHeight/2;
  myDiv.filters[0].addCone(0,0,1,ix2,iy2,255,255,0,100,30);   //添加一
个点光源
}
```

```
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="init()">
<H2>请在图片上移动鼠标观看效果……</H2>
<DIV CLASS="aFilter" ID="myDiv" onmousemove=" myDiv.filters[0].moveLight
( 0, window.event.x, window.event.y, 5, 1);">
  <IMG src=./images/cactus.jpg width=400>
</DIV>
</BODY>
</HTML>
```

本示例的效果为：当用户在图片上移动鼠标指针时，光源的焦点会随之移动，产生一种聚光灯的效果，如图 9.15 和图 9.16 所示。

图 9.15 使用 light 过滤器之一

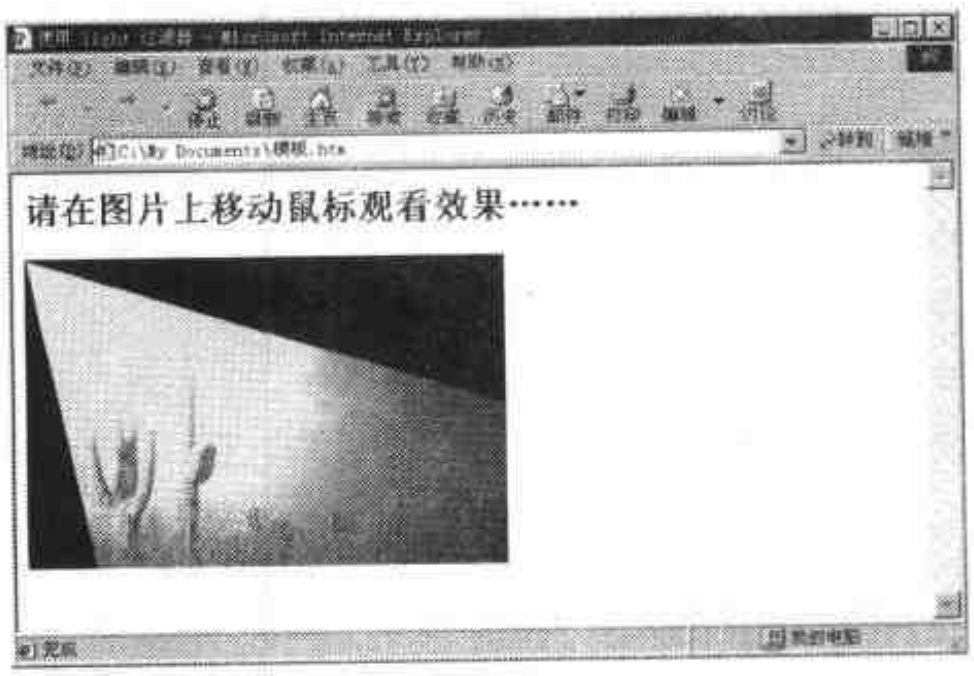


图 9.16 使用 light 过滤器之二

9.4 DHTML 小脚本应用

9.4.1 概述

DHTML 小脚本（scriptlet）是一种特殊的对象，可以直接插入到网页中以便实现特定的功能。与 ActiveX 控件或 Java Applet 等对象相比，DHTML 小脚本更容易开发和维护，并且下载更快，因为它的本质就是一个包含 DHTML 的独立页面。

作为一个对象，DHTML 小脚本需要通过 OBJECT 标记符插入到网页中。对于 DHTML 小脚本而言，有 5 个重要的属性需要指明：

- TYPE

在使用 DHTML 小脚本时，TYPE 属性必须设置为 "text/x-scriptlet"，它告诉 Internet Explorer 插入的对象应作为 DHTML 小脚本来对待。

注意： 仅 Internet Explorer 支持 DHTMLscriptlet 技术，而 NetScape Navigator 不支持。

- DATA

DATA 属性用于指定作为 DHTML 小脚本的网页的文件名，其值为标准的 URL。

- ID

ID 属性表示对象的标识，用于在脚本中引用 DHTML 小脚本。

- WIDTH 和 HEIGHT

WIDTH 和 HEIGHT 属性用于指定 DHTML 小脚本对象在网页中占据的空间，如果不指定，则对象是看不见的。

例如，在网页中可以使用以下语句引用一个 DHTML 小脚本：

```
<OBJECT
  ID="myScriptlet"
  WIDTH=200
  HEIGHT=200
  TYPE="text/x-scriptlet"
  DATA="myScriptlet.htm">
</OBJECT>
```

9.4.2 一个简单的小脚本

如果要使用 DHTML 小脚本，应该创建两个网页——一个用于包含小脚本对象，另一

个本身就是小脚本对象。本小节通过一个简单的示例来说明如何使用 DHTML 小脚本。

1. 创建小脚本

本示例中的小脚本仅有一个属性 text，为确保该属性是公用的（以便在主页面中通过脚本存取），我们在其名字前冠以 public_。以下是该小脚本的代码（保存在文件 scriptlet.htm 中）：

```
<SCRIPT LANGUAGE = "Javascript" TYPE="text/javascript">
<!--
  public_text = "";
//-->
</SCRIPT>
```

2. 创建主页面

在主页面中，我们用 OBJECT 标记符引用小脚本，代码如下：

```
<HTML>
<HEAD>
<TITLE> Hello World 小脚本示例 </TITLE>
</HEAD>
<BODY>
<OBJECT  WIDTH=0  HEIGHT=0  ID="myScriptlet"  TYPE="text/x-scriptlet"
DATA="scriptlet.htm">
</OBJECT>
<SCRIPT LANGUAGE = "Javascript" TYPE="text/javascript">
<!--
myScriptlet.text="Hello World!"  //引用小脚本对象的属性
document.write("<H2>"+myScriptlet.text+"</H2>");
-->
</SCRIPT>
</BODY>
</HTML>
```

此示例的效果如图 9.17 所示。

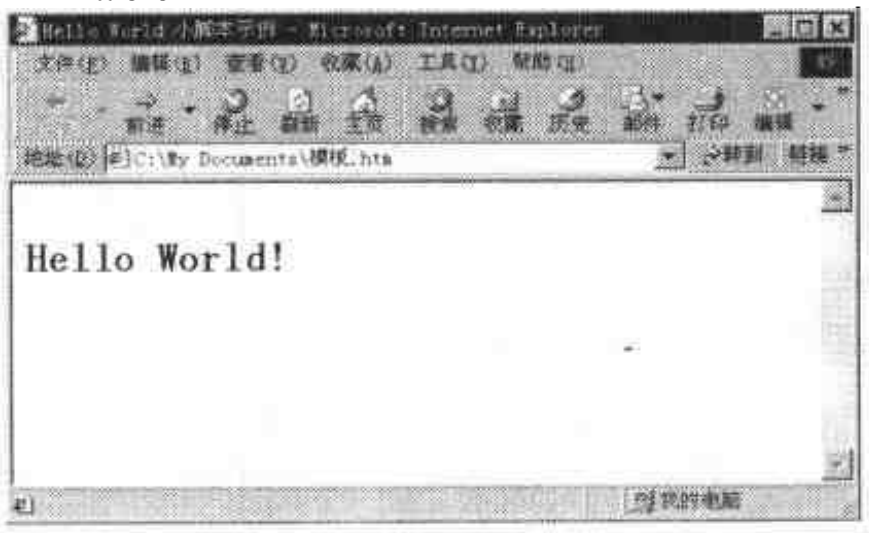


图 9.17　Hello World 小脚本示例

9.4.3 导航条示例

本小节的示例要复杂的多，需要综合应用本书中学到的很多知识，它改编自微软站点上的小脚本示例，用于实现两种不同样式的导航条。

1. 主页面的实现

以下代码是包含小脚本的主页面：

```
<HTML>
<HEAD>
<TITLE>导航条 Scriptlet</TITLE>
<SCRIPT LANGUAGE = "Javascript" TYPE="text/javascript">

function option_A()
{
    nav.style.width = "110%";
    nav.style.height = 70;
    textDiv.style.left = "40";
    textDiv.style.top = 245;
    textDiv.style.width = "90%";
    buttonA.style.left = "40";
    buttonB.style.left = "125";
    buttonA.style.top = 205;
    buttonB.style.top = 205;
    nav.setForeColor("#6A5ACD");
    nav.setBackColor("seashell");
    nav.setFont("Arial");
    nav.setFontSize("10pt");
    nav.setFontStyle("normal");
    nav.setWidth("340");
    nav.setHeight("40");
    nav.setLinks("浏览", "Browse.htm", "搜索", "Search.htm", "商店",
"Shop.htm", "购物", "Buy.htm", "主页", "Home.htm");
    nav.setOrientation("horizontal");
    nav.setHighLightColor("red");
}

function option_B()
```

```
{
    nav.style.width = "90";
    nav.style.height = 300;
    textDiv.style.left = "140";
    textDiv.style.top = 230;
    textDiv.style.width = "60%";
    buttonA.style.left = "140";
    buttonB.style.left = "140";
    buttonA.style.top = 140;
    buttonB.style.top = 175;
    nav.setForeColor("deeppink");
    nav.setBackColor("darkslateblue");
    nav.setFont("Arial");
    nav.setFontSize("10pt");
    nav.setFontStyle("normal");
    nav.setWidth("60");
    nav.setHeight("250");
    nav.setLinks("新闻","news.htm", "体育", "sports.htm", "影视",
"movie.htm","旅游", "local.htm", "广告", "classAds.htm");
    nav.setOrientation("vertical");
    nav.setHighLightColor("white");
}
</SCRIPT>
</HEAD>
<BODY TOPMARGIN=20 LEFTMARGIN="40" BGCOLOR="#FFFFFF" LINK="#000066"
VLINK="#666666" TEXT="#000000">
<H2><FONT FACE="楷体_gb2312">导航条</FONT> Scriptlet</H2>
<FONT FACE="隶书" SIZE=2>
  <A HREF="#" ONCLICK="self.close();return false">关闭窗口</A>
</FONT>
<P>
<HR>
<P>
<FONT FACE="黑体" SIZE=2>
<BR>
<OBJECT ID="nav"
        STYLE="position:absolute;
        width:80%;
        height:100;
```

```
        top:120;
        left:0;"
       type="text/x-scriptlet"
        data="navBar.htm">
</OBJECT>
<BUTTON  ID=buttonA  onclick="option_A()"  STYLE="position:absolute;
top:120; left:90%">样式 A</BUTTON>
<BUTTON  ID=buttonB  onclick="option_B()"  STYLE="position:absolute;
top:155; left:90%">样式 B</BUTTON>
<DIV ID=textDiv STYLE="position:absolute; top:230; left:40; width:90%">
<P><B>相关文本</B>
<P>此文本应相对于导航条的位置移动。在样式 A中，这些文本位于导航条下方；在样式 B中，
这些文本位于导航条右边。
<P>
</DIV>
</FONT>
</BODY>
</HTML>
```

2. 导航条对象的实现

以下代码是实现的 DHTML 小脚本页面（navBar.htm）：

```
<HTML>
<HEAD>
<TITLE>导航条 Scriptlet</TITLE>
<SCRIPT LANGUAGE = "Javascript" TYPE="text/javascript">
<!--

var     public_link1href,     public_link2href,     public_link3href,
public_link4href, public_link5href;
var public_docspans;
var public_divWidth, public_divHeight;
var highLightColor;

function public_setForeColor(foreColor)
{
    theDiv.style.color = foreColor;
}

function public_setBackColor(backColor)
```

```
{
    theDiv.style.backgroundColor = backColor;
}

function public_setFont(newFont)
{
    theDiv.style.fontFamily = newFont;
}

function public_setFontSize(fSize)
{
    theDiv.style.fontSize = fSize;
}

function public_setFontStyle(newStyle)
{
    theDiv.style.fontStyle = newStyle;
}

function public_setWidth(newWidth)
{
    public_divWidth = newWidth;
    theDiv.style.width = newWidth;
}

function public_setHeight(newHeight)
{
    public_divHeight = newHeight;
    theDiv.style.height = newHeight;
}

function public_setHighLightColor(col)
{
    highLightColor = col;
}

function public_setLinks(text1, href1, text2, href2, text3, href3, text4,
href4, text5, href5)
{
```

```
        link1.innerText = text1;
        link2.innerText = text2;
        link3.innerText = text3;
        link4.innerText = text4;
        link5.innerText = text5;
        public_link1href = href1;
        public_link2href = href2;
        public_link3href = href3;
        public_link4href = href4;
        public_link5href = href5;
    }

    function public_setOrientation(orient)
    {
        if (orient == "horizontal")
        {
            public_docspans = document.all.tags("span");
            for (i = 0; i<public_docspans.length; i++)
            {
                public_docspans(i).style.width = "18%";
                public_docspans(i).style.height = "90%";
                public_docspans(i).style.left                    =
((18*i+5)/100)*public_divWidth;
                public_docspans(i).style.textAlign = "center";
                public_docspans(i).style.top = "55%";
            }
        }
        else
        {
            public_docspans = document.all.tags("span");
            for (i = 0; i<public_docspans.length; i++)
            {
                public_docspans(i).style.width = "90%";
                public_docspans(i).style.height = "18%";
                public_docspans(i).style.left = "5%";
                public_docspans(i).style.textAlign = "center";
                public_docspans(i).style.top                     =
((18*i+5)/100)*public_divHeight;
            }
```

```
    }
}

function sethref()
{
    if (window.event.srcElement.id == "link1")
    {
        alert(public_link1href);
    }
    if (window.event.srcElement.id == "link2")
    {
        alert(public_link2href);
    }
    if (window.event.srcElement.id == "link3")
    {
        alert(public_link3href);
    }
    if (window.event.srcElement.id == "link4")
    {
        alert(public_link4href);
    }
    if (window.event.srcElement.id == "link5")
    {
        alert(public_link5href);
    }
}

var defaultColor;
var defaultStyle;

function highLight()
{
    if(window.event.srcElement.id != "theDiv")
    {
        defaultColor = window.event.srcElement.style.color;
        defaultStyle = window.event.srcElement.style.fontStyle;

        window.event.srcElement.style.color = highLightColor;
        window.event.srcElement.style.fontStyle = "italic";
```

```
        }
    }

    function highLightOff()
    {
        if(window.event.srcElement.id != "theDiv")
        {
            window.event.srcElement.style.color = defaultColor;
            window.event.srcElement.style.fontStyle = defaultStyle;
        }
    }
    // -->
    </SCRIPT>
    </HEAD>
    <BODY leftmargin=20 topmargin=20>
    <DIV ID=theDiv style="position:absolute; width:0; height:0"
    onclick="sethref()" onmouseover="highLight()"
    onmouseout="highLightOff()">
        <SPAN ID=link1 STYLE="position:absolute; top:4%; left:10%;
    width:80%; height:20%; cursor:hand; text-align:center; font
    weight:bold"></SPAN>
        <SPAN ID=link2 STYLE="position:absolute; top:22%; left:10%;
    width:80%; height:18%; cursor:hand; text-align:center; font
    weight:bold"></SPAN>
        <SPAN ID=link3 STYLE="position:absolute; top:42%; left:10%;
    width:80%; height:18%; cursor:hand; text-align:center; font
    weight:bold"></SPAN>
        <SPAN ID=link4 STYLE="position:absolute; top:62%; left:10%;
    width:80%; height:18%; cursor:hand; text-align:center; font-
    weight:bold"></SPAN>
        <SPAN ID=link5 STYLE="position:absolute; top:84%; left:10%;
    width:80%; height:16%; cursor:hand; text-align:center; font-
    weight:bold"></SPAN>
    </DIV>
    </BODY>
    </HTML>
```

本示例的显示效果为：当初次显示主页面时，如图 9.18 所示；如果单击“样式 A”按钮，则显示如图 8.19 所示；如果单击“样式 B”按钮，则显示如图 9.20 所示。

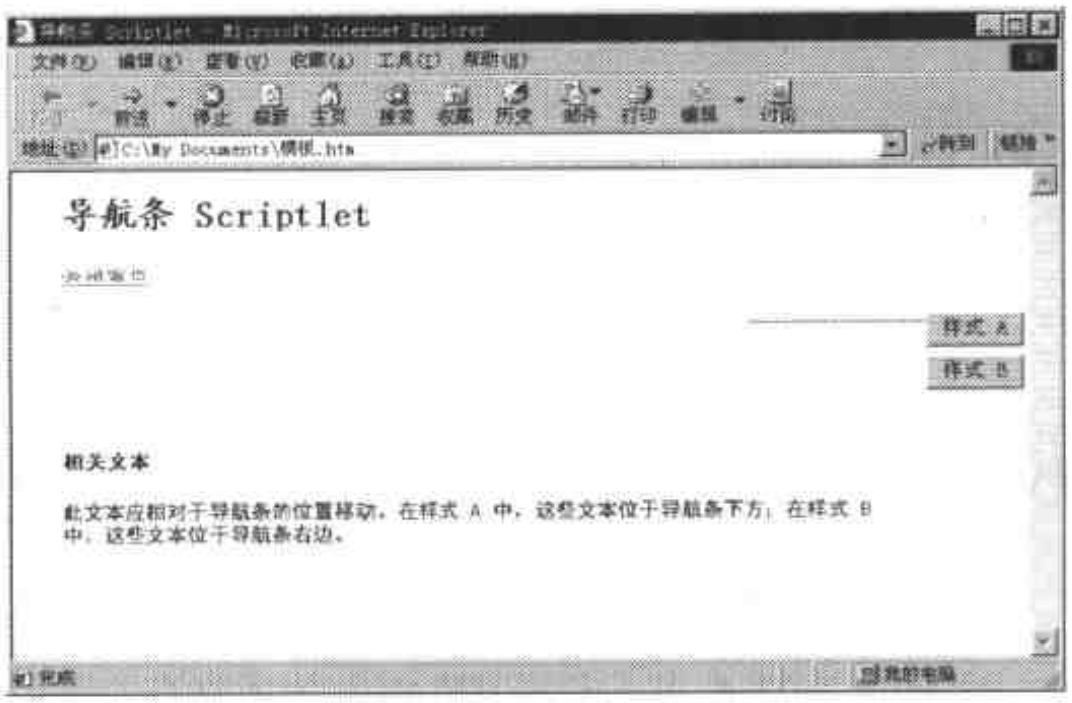


图 9.18 初始页面

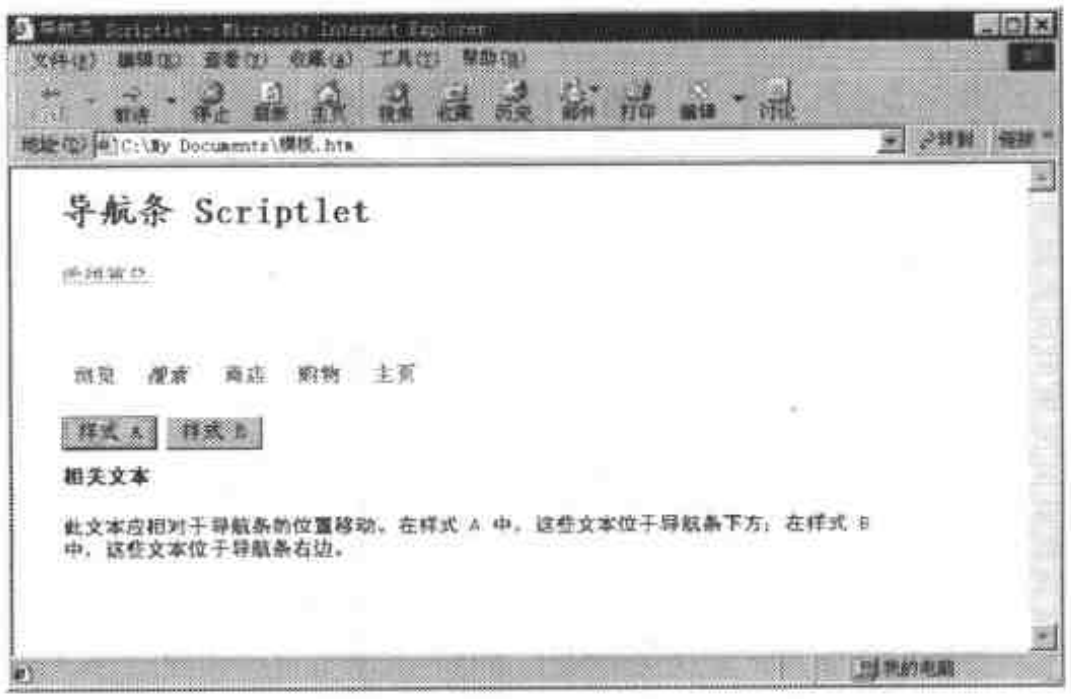


图 9.19 样式 A 页面

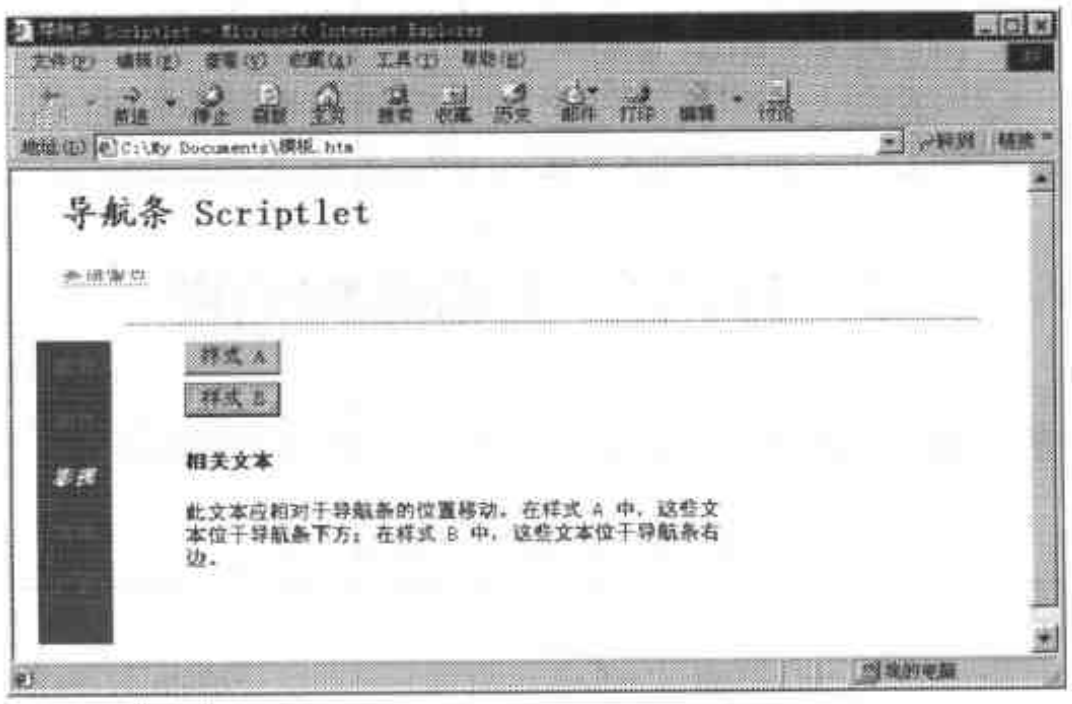


图 9.20 样式 B 页面

附录 1　HTML4.0 快速参考

在本附录的 9 个表格中，首先列举了多数 HTML 元素所具有的通用属性，然后按照类别列举了 HTML4.0 的所有元素和常用属性，以供读者参考。

1. 通 用 属 性

附表 1.1 列出了多数 HTML 元素所具有的通用属性。

附表1.1　通 用 属 性

通 用 属 性	说　　明
ID	ID 属性为文档中的元素指定了一个独一无二的身份标识，用于样式表和脚本引用。在定义 ID 属性时，必须注意此属性值由英文字母开头，后面可以跟任意字母（大写 A 到 Z 和小写 a 到 z）、数字（从 0 到 9）、连字符（-）、下划线（_）、冒号（:）以及点号（.）。另外需要注意的是，ID 属性与 NAME 属性使用相同的名称空间，因此不能在同一个文档中为 ID 和 NAME 属性定义相同的名称，以防止发生混乱
class	Class 属性定义了特定标记符的类，用于样式表和脚本引用
style	style 属性用于为一个单独的标记符指定样式
title	Title 属性与 TITLE 标记符不同（TITLE 标记符在文档中只能出现一次），它可以为文档中任意多个标记符指定参考标题信息。通常浏览器将参考标题信息以即时提示（tooltip，也叫做工具栏提示）的方式显示出来，以便浏览者查看

2. HTML 文档结构元素

附表 1.2 列出了所有的 HTML 文档结构元素以及它们的常用属性。

附表1.2　HTML文档结构元素

语　　法	常 用 属 性	说　　明
<HTML> </HTML>	无	开始标记符和结束标记符都可以省略。HTML 标记符说明此文档是一个 HTML 文档

续表

语　法	常用属性	说　明
<HEAD> </HEAD>	无	开始标记符和结束标记符都可以省略。HEAD元素包含文档的头部信息，如标题、关键字、说明和样式表等
<BODY> </BODY>	BACKGROUND=URL（文档的背景图像） BGCOLOR=Color（文档的背景色） TEXT=Color（文档中文本的颜色） LINK=Color（文档中链接的颜色） VLINK=Color（文档中已被访问过的链接的颜色） ALINK=Color（文档中活动链接的颜色） 通用属性	开始标记符和结束标记符都可以省略。BODY元素中包含文档体，也就是文档的正文。对于非框架文档，BODY位于HEAD之后；对于框架文档，如果包含NOFRAMES标记符，则BODY必须位于该标记符内，否则不能包含BODY标记符
<TITLE> </TITLE>	无	TITLE标记符位于HEAD标记符内，它包含的内容是文档的标题。TITLE标记符中包含的内容将在浏览器的标题栏中显示
<META>	NAME=name（名字） HTTP-EQUIV=Name（HTTP相应标题名） CONTENT=CDATA（相关数据）	META标记符中包含了网页的元数据信息，诸如文档关键字、作者信息等。文档的HEAD标记符内可以包含任意数量的<META>元素
<DIV> </DIV>	ALIGN=[left | center | right | justify]（水平对齐方式） 通用属性	DIV标记符用于包含行内元素（也称为字符级元素或文本级元素）和块级元素，以便定义一个块。通常该元素与class和ID等属性联合使用，以便在样式表中为某一块内容定义样式
<SPAN> </SPAN>	通用属性	SPAN标记符与DIV标记符类似，但通常用于包含行内元素
<H1>.. </H1> …… <H6>...</H6>	ALIGN=[left | center | right | justify]（水平对齐方式） 通用属性	H1~H6元素用于定义从1级到6级标题，可以使用align属性设置标题的对齐方式
<ADDRESS> </ADDRESS>	通用属性	此标记符用于提供联系信息，通常用斜体字显示其中的内容

3. 文本元素

附表1.3列出了所有的HTML文本元素以及它们的常用属性。

附表1.3　HTML文本元素

语　法	常用属性	说　明
<ABBR> </ABBR>	通用属性	ABBR元素用来标记缩写，通常与title属性一起使用。例如，<ABBR title = "Structured Query Language">SQL</ABBR>，则当浏览者将鼠标移动到SQL字样上时，将显示即时提示“Structured Query Language”

续表

语　法	常用属性	说　明
<ACRONYM> </ACRONYM>	通用属性	ACRONYM 元素被用来标记首字母缩略词，与 ABBR 元素类似，它常常与 title 属性一起使用
<BLOCKQUOTE> </BLOCKQUOTE>	CITE＝URL（引用源） 通用属性	BLOCKQUOTE 元素定义了一个块引用，其中可以包含块级元素，表示 <BLOCKQUOTE></BLOCKQUOTE> 中包含的内容是引自 cite 属性所指定的源（例如，"http://www.microsoft.com"）
 	CLEAR＝[left | all | right | none]（清除浮动对象） 通用属性	 标记符用于强行中断当前行，多个 标记符可以创建多个空行
<CITE> </CITE>	通用属性	CITE 元素用以标记引用内容，诸如杂志报纸的标题等。浏览器一般将 <CITE></CITE> 中的内容显示为斜体字
<CODE> </CODE>	通用属性	CODE 元素用于标记文档中的代码，通常浏览器将 <CODE></CODE> 中的内容显示为等宽字体
<DEL> </DEL>	CITE = URL（包含删除原因信息的 URL） DATETIME = Datetime（删除时间） 通用属性	DEL 元素用来标记文档中已删除的内容，可以用 title 属性给出简单的删除原因。通常浏览器将包含在 <DEL></DEL> 中的文字添加上删除线。为确保在多数浏览器中都可以有删除线效果，也可以结合使用 STRIKE 或 S 元素
<DFN> </DFN>	通用属性	DFN 元素用于指定一个定义，在浏览器中通常用斜体字显示
<EM> </EM>	通用属性	EM 元素用于对其中包含的内容进行强调，通常浏览器用斜体字显示 <EM></EM> 中包含的内容
<HR>	ALIGN＝[left | center | right]（指定水平对齐方式） NOSHADE（实线） SIZE＝Pixels（线宽） WIDTH＝Length（线长） 通用属性	HR 元素用于在网页中添加一条水平线
<INS> </INS>	CITE = URL（说明插入原因信息所在的 URL） DATETIME = Datetime（插入时间） 通用属性	INS 元素用于包含被插入的内容。在 IE5 中以下划线显示包含在 <INS></INS> 中的文字。用户也可以自定义样式表，以便指定特定的显示格式
<KBD> </KBD>	通用属性	KBD 元素用于包含键盘录入的文字，在浏览器中通常以等宽字体显示
<P>…</P>	ALIGN＝［left | center | right | justify］（设置水平对齐方式） 通用属性	结束标记符可以省略，但使用样式表时请使用结束标记符。P 元素用于在网页中分段
<PRE> </PRE>	WIDTH＝Number（宽度） 通用属性	PRE 元素用于包含预先格式化的文本。也就是说，包含在 <PRE></PRE> 中的内容将以所设置的格式显示
<Q> </Q>	CITE = URL（引用源） 通用属性	Q 元素用于表示短的行内引用。如果需要表示更长的引用，应使用 BLOCKQUOTE 元素

续表

<SAMP> </SAMP>	通用属性	SAMP 元素标记了网页中的输出样本，如程序的输出。通常浏览器将 <SAMP> </SAMP> 中的文字以等宽字体显示
<STRONG> </STRONG>	通用属性	STRONG 元素用于对包含在其中的内容进行强调，浏览器通常用粗体字显示包含在 <STRONG></STRONG> 中的内容
_{...}	通用属性	SUB 元素用于定义下标
^{...}	通用属性	SUP 元素用于定义上标
<VAR> </VAR>	通用属性	VAR 元素用以标记变量或程序参数。浏览器通常用斜体字显示包含在 <VAR> </VAR> 中的文字

4. 字体样式元素

附表 1.4 列出了所有的 HTML 字体样式元素以及它们的常用属性。

附表1.4 HTML字体样式元素

语　法	常用属性	说　明
<B>...</B>	通用属性	B 元素可以使文本以粗体形式出现
<BASEFONT>	SIZE＝CDATA（指定默认字体大小，范围为 1~7，默认值是 3） COLOR＝Color（指定默认字体颜色） FACE＝CDATA（指定默认字体） ID＝ID（唯一的 ID）	BASEFONT 元素允许作者规定基本字体的大小、颜色和“字体”。但由于样式表的出现，在 HTML4 中它是已过时的用法
<BIG>.. </BIG>	通用属性	BIG 元素规定文本以大字体显示。
<FONT> </FONT>	SIZE＝CDATA（字体大小调整） COLOR＝Color（字体颜色调整） FACE＝CDATA（字体样式调整） 通用属性	FONT 元素用于设置所包含字体的大小、颜色和“字体”。由于样式表单的出现，FONT 元素在 HTML4 中属已过时的用法
<I>...</I>	通用属性	I 元素规定文本以斜体显示
<S>...</S>	通用属性	S 元素规定文本以包含删除线的方式显示，效果与 STRIKE 元素相同
<SMALL> </SMALL>	通用属性	SMALL 元素规定文本以小字体显示
<STRIKE> </STRIKE>	通用属性	STRIKE 元素规定文本显示时加删除线，效果与 S 元素相同
<TT>...</TT>	通用属性	TT 元素规定文本以电报文字体或等宽字体显示
<U>...</U>	通用属性	U 元素规定文本显示时加下划线

5. 列表元素

附表 1.5 列出了所有的 HTML 列表元素以及它们的常用属性。

附表1.5 HTML列表元素

语　法	常用属性	说　明
<UL> </UL>	TYPE＝[disc|square|circle]（编号样式） COMPACT（紧凑显示） 通用属性	UL 元素定义了一个无序列表，其中包含一个或多个 LI 元素来定义实际的列表项
<OL> </OL>	TYPE＝[1|a|A|i|I]（编号方式） START＝Number（起始数） COMPACT（紧凑显示） 通用属性	OL 元素定义了一个有序列表。OL 元素中包含一个或多个 LI 元素来定义实际的列表项
<LI> </LI>	TYPE＝[disc|square|circle|1|a|A|i|I]（列表项标记样式） VALUE＝Number（序列号） 通用属性	结束标记可以省略，但使用样式表时应使用结束标记。LI 元素定义了一个列表项
<DL> </DL>	COMPACT（紧凑显示） 通用属性	DL 元素定义了一个定义列表。定义列表中的条目是通过使用 DT 元素和 DD 元素创建的。DT 元素给出了术语名，而 DD 元素给出了术语的定义
<DT> </DT>	通用属性	结束标记可以省略，但使用样式表时应使用结束标记。DT 元素在定义列表中定义了一个术语
<DD> </DD>	通用属性	结束标记可以省略，但使用样式表时应使用结束标记。DD 元素在定义列表中为一个术语提供定义数据
<DIR> </DIR>	COMPACT（紧凑显示） 通用属性	DIR 元素定义了一个目录列表，其中包含一个或多个定义实际列表项的 LI 元素。此时 LI 元素中不可包含块级元素。在 HTML 4.0 中，DIR 元素已被 UL 元素取代
<MENU> </MENU>	COMPACT（紧凑显示） 通用属性	MENU 元素定义了一个菜单列表，其中包含一个或多个 LI 元素来定义实际菜单项。此元素在 HTML4.0 中属过时的用法

6. 表格元素

附表 1.6 列出了所有的 HTML 表格元素以及它们的常用属性。

附表1.6 HTML表格元素

语　法	常用属性	说　明
<TABLE> </TABLE>	WIDTH＝Length（表宽） BORDER＝Pixels（边框宽度） FRAME＝[void | above | below | hsides | lhs | rhs | vsides | box | border]（外边框） RULES＝ [none | groups | rows | cols | all]（表格框线） CELLSPACING＝Length（单元格间距） CELLPADDING＝Length（单元格填充距） ALIGN＝［left | center | right］（表格对齐） BGCOLOR＝Color（表格背景色） 通用属性	TABLE 元素用于定义表格，所有表格中的内容都应包含在<TABLE> 和 </TABLE> 中
<CAPTION> </CAPTION>	ALIGN＝［top | bottom | left | right］（对齐方式） 通用属性	CAPTION 元素定义了表格的标题，使用时 CAPTION 标记符必须放在表格最开头（即<TABLE> 之后）
<THEAD> </THEAD>	ALIGN＝［left | center | right | justify | char］（组中单元格的水平对齐方式） VALIGN＝［top | middle | bottom | baseline］（组中单元格的垂直对齐方式） 通用属性	THEAD 元素定义了表格的表头，一个表格中最多可含有一个 THEAD 标记符。目前多数浏览器还不支持 THEAD 标记符
<TFOOT> </TFOOT>	ALIGN＝［left | center | right | justify | char］（组中单元格的水平对齐方式） VALIGN＝［top | middle | bottom | baseline］（组中单元格的垂直对齐方式） 通用属性	TFOOT 元素定义了表格的脚注行，一个表格中最多可含有一个 TFOOT 标记符。目前多数浏览器还不支持 TFOOT 标记符
<TBODY> </TBODY>	ALIGN＝［left | center | right | justify | char］（组中单元格的水平对齐方式） VALIGN＝［top | middle | bottom | baseline］（组中单元格的垂直对齐方式） 通用属性	TBODY 在表格中定义了一组数据行，表格中至少有一个 TBODY 标记符
<COLGROUP> </COLGROUP>	SPAN = Number（组的列数） WIDTH = MultiLength（每列宽度） ALIGN＝［left | center | right | justify | char］（组中单元格的水平对齐方式） VALIGN＝［top | middle | bottom | baseline］（组中单元格的垂直对齐方式） 通用属性	COLGROUP 元素定义了一个表格中的列组。使用列组时，COLGROUP 元素必须放在可选的 CAPTION 元素之后，且在可选的 THEAD 元素之前
<COL>	SPAN = Number（列数） WIDTH = MultiLength（列宽度） ALIGN＝［left | center | right | justify | char］（列单元格的水平对齐方式） VALIGN＝［top | middle | bottom | baseline］（列单元格的垂直对齐方式） 通用属性	COL 元素定义了一个表格列的属性

续表

语　法	常用属性	说　明
<TR> </TR>	ALIGN＝[left | center | right | justify | char]（组中单元格的水平对齐方式） VALIGN＝[top | middle | bottom | baseline]（组中单元格的垂直对齐方式） BGCOLOR = Color（背景色） 通用属性	TR 元素定义了一个表格行。TR 标记符包含 <TH> 和 <TD> 标记符，<TH> 和 <TD> 标记符中又包含了表格的实际数据
<TH> </TH>	ROWSPAN＝Number（单元格所占的行数） COLSPAN＝Number（单元格所占的列数） ALIGN＝[left | center | right | justify | char]（单元格的水平对齐方式） VALIGN＝[top | middle | bottom | baseline]（单元格的垂直对齐方式） WIDTH＝Pixels（单元格宽） HEIGHT＝Pixels（单元格高） NOWRAP（单元格内不换行） BGCOLOR＝Color（单元格背景色） 通用属性	TH 元素定义了表格中的一个标题单元格，其中的内容通常以黑体显示。TH 标记符位于 TR 标记符内
<TD> </TD>	ROWSPAN＝Number（单元格所占的行数） COLSPAN＝Number（单元格所占的列数） ALIGN＝[left | center | right | justify | char]（单元格的水平对齐方式） VALIGN＝[top | middle | bottom | baseline]（单元格的垂直对齐方式） WIDTH＝Pixels（单元格宽） HEIGHT＝Pixels（单元格高） BGCOLOR＝Color（单元格背景色） 通用属性	TD 元素定义了表格中的一个数据单元格。TD 标记符位于 TR 标记符内

7. 框架元素

附表 1.7 列出了所有的 HTML 框架元素以及它们的常用属性。

附表1.7　HTML框架元素

语　法	常用属性	说　明
<FRAMESET> </FRAMESET>	ROWS＝MultiLengths（设置横向框架） COLS＝MultiLengths（设置纵向框架） 通用属性	FRAMESET 元素是一个框架容器，它将窗口分成长方形的子区域，即框架。FRAMESET 标记符中包含一个或多个 <FRAMESET> 或 <FRAME> 标记符，并可能含有一个可选的 <NOFRAMES> 标记符

续表

语　法	常用属性	说　明
<FRAME>	NAME＝CDATA（框架名） SRC＝URL（框架的初始页面） FRAMEBORDER＝［1	0］（设置是否显示框架边框） MARGINWIDTH＝Pixels（边距宽度） MARGINHEIGHT＝Pixels（边距高度） NORESIZE（禁止修改框架尺寸） SCROLLING＝［yes	no	auto］（设置是否显示滚动条） 通用属性	FRAME 元素定义了一个框架，即一个框架集文档（FRAMESET）中的长方形空间。FRAME 标记符必须包含在 FRAMESET 标记符中
<NOFRAMES> </NOFRAMES>	通用属性	NOFRAMES 元素中包含了框架不能被显示时的替换内容。NOFRAMES 元素通常在 Frameset 文档中使用，它在浏览器不支持框架或框架被禁用时，提供相应的替换内容。NOFRAMES 标记符必须位于 FRAMESET 标记符之间
<IFRAME> </IFRAME>	SRC＝URL（框架内容网页的 URL） NAME＝CDATA（框架名） WIDTH＝Length（框架宽度） HEIGHT＝Length（框架高度） ALIGN＝［top	middle	bottom	left	right］（框架对齐方式） FRAMEBORDER＝［1	0］（设置是否显示框架边框） MARGINWIDTH＝Pixels（边距宽） MARGINHEIGHT＝Pixels（边距高） SCROLLING＝［yes	no	auto］（是否显示滚动条） 通用属性	IFRAME 元素定义了一个页内框架，可以在其中显示 HTML 页面。包含在 <IFRAME> 和 </IFRAME> 中的内容只有当浏览器不支持框架时才显示

8. 表单元素

附表 1.8 列出了所有的 HTML 框架元素以及它们的常用属性。

附表1.8　HTML框架元素

语　法	常用属性	说　明
<FORM> </FORM>	ACTION＝URL（处理表单结果的脚本的位置） METHOD＝［get	post］（发送表单的 HTTP 方法） TARGET＝FrameTarget（显示表单内容的框架） 通用属性	FORM 元素定义了一个交互式表单

续表

语　法	常用属性	说　明
<INPUT>	TYPE＝［text	password	checkbox	radio	submit	reset	file	hidden	image	button］（控件类型） NAME＝CDATA（控件的名称） VALUE＝CDATA（控件的值） CHECKED（设置单选框或复选框的初始选中状态） SIZE＝CDATA（文本框的宽度，以字符数为单位） MAXLENGTH＝Number（最大文本输入字符数） SRC＝URL（图像源） ALT＝CDATA（图像的替换文本） USEMAP＝URL（客户端图像映射） ALIGN＝［top	middle	bottom	left	right］（表单元素的对齐方式） ACCESSKEY＝Character（快捷键） TABINDEX＝Number（在 tab 键遍历次序中的位置） 通用属性	INPUT 元素定义了一个用于用户输入的表单控件，通常位于 FORM 标记符内
<BUTTON> </BUTTON>	NAME＝CDATA（控件的名称） VALUE＝CDATA（控件的值） TYPE＝［submit	reset	button］（按钮类型） ACCESSKEY＝Character（快捷键） TABINDEX＝Number（在 tab 键遍历次序中的位置） 通用属性	BUTTON 元素定义了一个按钮，可以是提交、重置或普通按钮。虽然也可以用 INPUT 元素创建按钮，但用 BUTTON 元素创建的按钮通常具有更强的表现力
<SELECT> </SELECT>	NAME＝CDATA（控件的名称） MULTIPLE（控制是否可以选择多个选项） SIZE＝Number（显示出的菜单框行数） TABINDEX＝Number（在 tab 键遍历次序中的位置） 通用属性	SELECT 元素定义了一个选项菜单，其中包含若干个 OPTGROUP 或 OPTION 元素来为用户提供选项
<OPTGROUP> </OPTGROUP>	LABEL＝Text（组标签） 通用属性	OPTGROUP 元素定义了一个 SELECT 菜单内的选项组，其中至少包含一个 OPTION 元素来定义实际的选项。多数浏览器并不支持 OPTGROUP 元素
<OPTION> </OPTION>	VALUE＝CDATA（选项值） SELECTED（初始选择值） LABEL＝Text（选项标签） 通用属性	OPTION 元素定义了 SELECT 菜单中的菜单选项
<TEXTAREA> </TEXTAREA>	NAME＝CDATA（控件的名称） ROWS＝Number（多行文本框的行数） COLS＝Number（多行文本框的列数） ACCESSKEY＝Character（快捷键） TABINDEX＝Number（在 tab 键遍历次序中的位置） 通用属性	TEXTAREA 元素定义了一个多行文本框控件

续表

语　法	常用属性	说　明
<ISINDEX>	PROMPT（提示信息） 通用属性	ISINDEX 元素定义了一个单行文本输入框
<LABEL> </LABEL>	FOR=IDREF（相关表单控件的 ID） ACCESSKEY=Character（快捷键） 通用属性	LABEL 元素将一个表单控件和一个标签联系起来
<FIELDSET> </FIELDSET>	通用属性	FIELDSET 元素定义了一个表单控件组。在 FIELDSET 标记符中应包含作为控件组成员的各表单控件，并需要使用 LEGEND 标记符创建一个控件组标签
<LEGEND> </LEGEND>	ACCESSKEY=Character（快捷键） ALIGN=［top | bottom | left | right］（标签文字相对于控件组的对齐方式） 通用属性	LEGEND 元素定义了一个控件组的标签，且必须立即出现在 <FIELDSET> 标记符之后

9. 其他元素

附表 1.9 列出了其他不属于以上任意组的所有的 HTML 元素以及它们的常用属性。

附表1.9　其他HTML元素

语　法	常用属性	说　明
<A> </A>	HREF=URL（链接的目标文件位置） NAME=CDATA（已命名的链接目标） TARGET=FrameTarget（显示链接的目标框架） ACCESSKEY=Charater（快捷键） TABINDEX=Number（tab 键遍历次序中的位置） 通用属性	A 元素定义了一个超链接（使用 href 属性时）或者一个超链接的目的位置（使用 name 属性时）。当定义超链接时，位于 <A> 和 </A> 之间的内容成为超链接的源，浏览者可以单击超链接源跳转到超链接目标
<APPLET> </APPLET>	CODE=CDATA（类文件名称或路径） CODEBASE=URL（类文件的基础 URL） WIDTH=Length（小程序在网页中所占的宽度） HEIGHT=Length（小程序在网页中所占的高度） ARCHIVE=URL-LIST（存档文件所在的位置列表） OBJECT=CDATA（序列化的小程序） NAME=CDATA（小程序实例的名称，用于小程序间通信） ALT=Text（替换文本） ALIGN=［top | middle | bottom | left | right］（小程序在页面的对齐方式） HSPACE=Pixels（小程序对象左右的空白距离） VSPACE=Pixels（小程序对象上下的空白距离） 通用属性	APPLET 元素用来嵌入一个 Java 小程序（Applet）。在 HTML 4.0 中，建议使用 OBJECT 元素代替 APPLET 元素。使用 APPLET 标记符时，可以用 PARAM 标记符指定运行时参数

续表

语　法	常用属性	说　明
<AREA>	SHAPE＝[rect | circle | poly | default]（客户端图像映射中映射区域的形状） COORDS＝Coords（客户端图像映射中映射区域的坐标） HREF＝URL（链接的目标文件位置） TARGET＝FrameTarget（显示链接的目标框架） NOHREF（不包含链接） ALT＝Text（替换文本） 通用属性	AREA 元素定义了一个在客户端图像映射中的图形区域。AREA 标记符位于 MAP 标记符内
<BASE>	HREF＝URL（默认 URL 基准） TARGET＝FrameTarget（默认目标框架）	BASE 元素定义了文档的默认 URL 基准和默认目标框架。一个文档中最多有一个 BASE 标记符，而且如果使用则必须位于 HEAD 标记符内
<BDO> </BDO>	DIR＝[ltr | rtl]（文本的方向） LANG＝LanguageCode（文本的语言） 通用属性	BDO 元素覆盖了所包含文本的双向算法。BDO 元素用于设置多语言文本的显示方向，在一般的网页中并不常用。
<CENTER> </CETNER>	通用属性	CENTER 元素定义了一个居中对齐的块。
<IMG>	SRC＝URL（图像源的位置） ALT＝Text（替换文本） WIDTH＝Length（图像宽度） HEIGHT＝Length（图像高度） USEMAP＝URL（客户端图像映射的映射说明，对应于 MAP 元素指定的内容） ISMAP（指示使用服务器端图像映射） ALIGN＝top | middle | bottom | left | right（图像对齐方式） BORDER＝Length（图像边框的宽度） HSPACE＝Pixels（图像左右的空白距离） VSPACE＝Pixels（图像上下的空白距离） 通用属性	IMG 元素定义了一个行内图像
<LINK>	HREF＝URL（链接资源的 URL） TARGET＝FrameTarget（显示链接的目标框架） 通用属性	LINK 元素定义了文档的关联关系。LINK 标记符应包含在 HEAD 标记符内，并且可以有多个
<MAP> </MAP>	NAME＝CDATA（图像映射的名称） 通用属性	MAP 元素用于定义图像映射的区域信息。MAP 的 NAME 属性通常用作 IMG 或 OBJECT 标记符的 USEMAP 属性的值。MAP 标记符内包含多个 AREA 标记符，用于定义图像上可单击的区域

续表

语　法	常用属性	说　明
<NOSCRIPT> </NOSCRIPT>	通用属性	NOSCRIPT 元素为不执行客户端程序的浏览器提供了替代的显示内容。NOSCRIPT 标记符应紧跟在它所提供替换内容的 SCRIPT 标记符后。只有当浏览器不支持客户端程序时，才显示 <NOSCRIPT></NOSCRIPT> 中的内容
<OBJECT> </OBJECT>	DATA＝URL（对象数据的位置） CLASSID＝URL（实现位置） ARCHIVE＝CDATA（存档文件） CODEBASE＝URL（CLASSID、DATA、ARCHIVE 的基准 URL） WIDTH＝Length（对象宽度） HEIGHT＝Length（对象高度） NAME＝CDATA（如果对象在表单中提交则定义其名称） TYPE＝ContentType（对象内容类型） CODETYPE＝ContentType（代码内容类型） STANDBY＝Text（装载时显示的信息） TABINDEX＝NUMBER（在 tab 键遍历顺序中的位置） ALIGN＝［top | middle | bottom | left | right］（对象对齐方式） BORDER＝Length（对象边框宽度） HSPACE＝Pixels（对象左右的空白距离） VSPACE＝Pixels（对象上下的空白距离） 通用属性	OBJECT 元素在网页中定义了一个对象。这个对象可以是图像、Java 小程序、ActiveX 控件、多媒体对象等各种对象。使用 OBJECT 定义对象时，还可以用 PARAM 标记符为对象指定运行时参数。在 HTML4.0 中，建议用通用的 OBJECT 元素取代更为特殊的 IMG、APPLET 等元素。不过，使用 OBJECT 代替所有其他对象元素（如 IMG、APPLET 等）的用法目前还没有得到多数浏览器的支持。 为确保浏览器的支持，通常使用嵌套的 OBJECT 元素包含多个对象，以便当浏览器无法显示外层对象时，依次尝试显示内层对象
<PARAM>	NAME＝CDATA（参数名称） VALUE＝CDATA（参数值） VALUETYPE＝［data | ref | object］（值的类型） TYPE＝ContentType（当 valuetype＝ref 时指定值的内容类型） ID＝ID（元素的 ID）	PARAM 元素指定了对象在运行时需要的一系列值。在 OBJECT 或 APPLET 标记符中可以以任意顺序包含任意数量的 PARAM 标记符。在使用 PARAM 指定参数时，对象必须能识别所指定的参数名和值
<SCRIPT> </SCRIPT>	TYPE＝ContentType（编程语言的内容类型） LANGUAGE＝CDATA（编程语言名） SRC＝URL（外部程序位置） CHARSET＝Charset（外部程序的字符编码） DEFER（设置此布尔属性时，表示告知浏览器脚本并不产生任何文档内容（例如，在 JavaScript 中没有 "document.write" 语句），从而使浏览器可以继续解释 HTML 文件的内容并进行显示）	SCRIPT 元素在文档中包含一段客户端脚本程序。客户端脚本程序能使文档更好地对客户端的事件作出反应。例如，一段程序可以在用户发送所填写的表单之前先检查用户填写的内容，并立即通知用户填写错误。SCRIPT 标记符可以位于文档中的任何位置，但通常位于 HEAD 标记符内，以便于维护
<STYLE> </STYLE>	TYPE＝ContentType（样式语言的类型） TITLE＝Text（样式表的名字）	STYLE 元素用于在文档中嵌入样式表。文档的 HEAD 标记符中可以包含任意数量的 STYLE 标记符。对于层叠样式表（CSS），TYPE 属性的值是 "text/css"

附录 2 CSS 属性参考

附表 2.1 按字母顺序列出了所有的 CSS 属性和对应的 DHTML 特性（所谓 DHTML 特性是指在客户端脚本中引用该 CSS 属性时使用的名称），并对常用的属性作了较详细的说明，供读者在设计网页时参考。

附表2.1 CSS属性参考

CSS 属性	DHTML 特性	说 明
background	background	提供多个背景属性的组合。例如：BODY{background: no-repeat yellow left center url(./image/background.jpg)}
background-attachment	backgroundAttachment	指定特定 HTML 元素背景图案是否与内容一起滚动，取值为 scroll | fixed。默认值 scroll 表示背景图案随着内容一起滚动；fixed 表示背景图案静止，而内容可以滚动。例如：BODY{background-attachment:fixed}
background-color	backgroundColor	指定特定 HTML 元素的背景颜色，取值为 transparent 或具体颜色值。例如，H1{background-color:gray}
background-image	backgroundImage	指定特定 HTML 元素的背景图案，取值为 none | url(imageurl)（在指定其他有关背景图案的属性时，应先指定此属性）。例如，H1{background-image:url(./MM.jpg)}
background-position	backgroundPosition	指定特定 HTML 元素背景图案的位置，取值为由空格隔开的两个值，既可以使用关键字 left | center | right 和 top | center | bottom，也可以指定百分数值，或者指定以标准单位计算的距离。例如：BODY{ background-position:left center}
background-repeat	backgroundRepeat	指定特定 HTML 元素背景图案是否重复，取值为 repeat | repeat-x | repeat-y | no-repeat
border	border	一次性指定四个框线的宽度、样式和颜色。例如，H1{border:thin solid blue}
border-bottom	borderBottom	指定边框底边的宽度、样式和颜色。例如，H1{border-top:thin solid blue}
border-bottom-color	borderBottomColor	指定边框底边的颜色，取值可以是任意颜色值
border-bottom-style	borderBottomStyle	指定边框底边的样式，取值可以是：none | dotted | dashed | solid |double | groove | ridge | inset | outset，默认值是 none
border-bottom-width	borderBottomWidth	指定边框底边的宽度，取值可以是：thin | medium | thick 或长度值

续表

CSS 属性	DHTML 特性	说　明
border-color	borderColor	设置边框的颜色，取值可以是任意颜色值。可以指定多个值，按上、右、下、左的顺序指定边框的颜色。如果指定了 2 或 3 个值，则未指定颜色的边框采用相对边框的颜色值
border-left	borderLeft	指定边框左边的宽度、样式和颜色。例如，H1{border-left:thin solid blue}
border-left-color	borderLeftColor	指定边框左边的颜色，取值可以是任意颜色值
border-left-style	borderLeftStyle	指定边框左边的样式，取值可以是：none | dotted | dashed | solid |double | groove | ridge | inset | outset，默认值是 none
border-left-width	borderLeftWidth	指定边框左边的宽度，取值可以是：thin | medium | thick 或长度值
border-right	borderRight	指定边框右边的宽度、样式和颜色。例如，H1{border-right:thin solid blue}
border-right-color	borderRightColor	指定边框右边的颜色，取值可以是任意颜色值
border-right-style	borderRightStyle	指定边框右边的样式，取值可以是：none | dotted | dashed | solid |double | groove | ridge | inset | outset，默认值是 none
border-right-width	borderRightWidth	指定边框右边的宽度，取值可以是：thin | medium | thick 或长度值
border-style	borderStyle	设置边框的样式，取值可以是：none | dotted | dashed | solid |double | groove | ridge | inset | outset，默认值是 none，当取多个值时，也按上、右、下、左的顺序为四个边框设置不同的样式；如果指定了 2 或 3 个值，则未指定样式的边框采用相对边框的样式值
border-top	borderTop	指定边框顶边的宽度、样式和颜色。例如，H1{border-top:thin solid blue}
border-top-color	borderTopColor	指定边框顶边的颜色，取值可以是任意颜色值
border-top-style	borderTopStyle	指定边框顶边的样式，取值可以是：none | dotted | dashed | solid |double | groove | ridge | inset | outset，默认值是 none
border-top-width	borderTopWidth	指定边框顶边的宽度，取值可以是：thin | medium | thick 或长度值
border-width	borderWidth	此属性是设置 border-top-width、border-right-width、border-bottom-width、和 border-left-width 的快捷方式（以此给定顺序，即上、右、下、左的顺序）。如果只给定一个值，则它应用于所有四个边框线。如果给定 2 或 3 个值，则未指定的边框采用与其相对边框的设置
bottom	bottom	确定元素下方向的位置
clear	clear	指定元素是否允许浮动元素在它旁边，取值可以是：none | left | right | both，默认值为 none
clip	clip	控制元素的剪切
color	color	指定特定 HTML 元素内文本的显示颜色，取值为任何合法的颜色值。例如，以下定义都是正确的：H1.red {color:red}、H1.red {color:#F00}、H1.red {color:#FF0000}、H1.red {color:rgb(255,0,0)}、H1.red {color:rgb(100%,0,0)}

续表

CSS 属性	DHTML 特性	说　明
cursor	cursor	控制鼠标指针的样式，取值可以是 auto | crosshair | default | hand | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize | text | wait | help。例如，P{cursor:hand}
display	display	确定元素是否应绘制在页面上，取值可以是 block | inline | list-item | none
float	styleFloat	指定元素在何处浮动，取值为：none | left | right，默认值为 none
font	font	提供多个字体属性的组合，按以下顺序定义：font-weight,font-variant,font-style,font-size,line-height，属性可以省略。例如，P{font:italic 200% serif}
font-family	fontFamily	描绘要使用的“字体”优先级序列，取值可以是字体名称，也可以是字体族名称，值之间用逗号分隔。例如：BODY{font-family:gill,helvetica,sans-serif}
font-size	fontSize	指定所用字体的大小，取值为 xx-small | x-small | small | medium | large lx-large | xx-large | smaller | larger，或者是具体的长度值或百分比
font-style	fontStyle	指定选定字体的样式，取值为 normal | italic | oblique，后两者表示斜体。例如：H4 {font-style: italic}
font-variant	fontVariant	选择正常或小写变体，取值为 normal | small-caps
font-weight	fontWeight	指定所用字体的粗细，取值为 normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
height	height	指定元素的高度
left	left	确定元素左方向的位置
letter-spacing	letterSpacing	指定文本字母间的间距，取值为 normal 或具体的长度值。例如：P{letter-spacing:1mm}
line-height	lineHeight	指定目标选项内文本的行高，取值可以是数字、长度或百分比，默认值是 normal。例如，DIV{line-height:1.5}
link	link	控制链接的颜色
list-style	listStyle	一次性指定列表项目标记的图片、类型和位置。例如，UL{list-style:url(./image/ball.gif) disc inside}
list-style-image	listStyleImage	为列表指定图片作为项目标记，取值可以是 none | url(imageurl)。例如，UL{list-style-image:url(./target.gif)}
list-style-position	listStylePosition	指定列表项目标记的位置，取值可以是 inside 或 outside，默认值是 outside
list-style-type	listStyleType	指定列表项目标记的类型，取值可以是 disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none。例如，OL{list-style-type:lower-roman}
margin	margin	指定边界宽度的简捷方式，可同时指定上、右、下、左（以此顺序）边界的宽度。如果只指定一个值，则四个方向都采用相同的边界宽度；如果指定了 2 或 3 个值，则没有指定边界宽度的边采用对边的边界宽度。例如，P{margin: 20px,0px}

续表

CSS 属性	DHTML 特性	说　明
margin-bottom	marginBottom	设置底端边界的宽度，取值可以是长度、百分比或 auto
margin-left	marginLeft	设置左端边界的宽度，取值可以是长度、百分比或 auto
margin-right	marginRight	设置右端边界的宽度，取值可以是长度、百分比或 auto
margin-top	marginTop	设置顶端边界的宽度，取值可以是长度、百分比或 auto
overflow	overflow	控制元素内容越界时的处理
padding	paddingTop	指定填充的简捷方式，可同时指定上、右、下、左四个方向（以此顺序）填充的宽度。如果只指定一个值，则四个方向都采用相同的填充宽度；如果指定了 2 或 3 个值，则没有指定填充宽度的边采用对边的填充宽度。例如，P{padding:0.25in, 0.1in}
padding-bottom	width	设置底端填充，取值可以是长度和百分数，但不允许使用负值
padding-left	paddingLeft	设置左端填充，取值可以是长度和百分数，但不允许使用负值
padding-right	paddingRight	设置右端填充，取值可以是长度和百分数，但不允许使用负值
padding-top	paddingTop	设置顶端填充，取值可以是长度和百分数，但不允许使用负值
page-break-after	pageBreakAfter	设置在元素后是否出现页分隔符
page-break-before	pageBreakBefore	设置在元素前是否出现页分隔符
position	position	确定元素的定位方式，取值可以是 static | relative | absolute，默认值为 static
right	right	确定元素右方向的位置
text-align	textAlign	指定目标选项内文本的对齐方式，取值是：left | right | center | justify。例如：P{text-align:justify}
text-decoration	textDecoration	对文本施加修饰效果，取值为 none | underline | overline | line-through | blink。例如，:link,:visited,:active{text-decoration:none}
text-indent	textIndent	按指定数值缩进文本，取值可以是长度值或百分比，默认值是 0。例如，P{text-indent:.74cm}
text-transform	textTransform	对文本进行大小写转换，取值为：capitalize | uppercase | lowercase | none，默认值是 none。capitalize 值指示所选元素中文本的每个单词的首字母为大写；uppercase 值指示所有的文本都为大写，lowercase 值指示所有文本都以小写显示。例如，P{text-transform:uppercase}
top	top	确定元素上方向的位置
unicode-bidi	unicodeBidi	确定与双向运算法则相关的嵌入层
vertical-align	verticalAlign	确定垂直方向上的对齐，取值为 baseline | sub | super | top | text-top | middle | bottom | text-bottom 或百分比。例如，IMG{vertical-align:text-top}
visibility	visibility	确定定位的元素是否可见，取值可以是 inherit | visible | hidden，默认值是 inherit

续表

CSS 属性	DHTML 特性	说　明
width	width	指定元素的宽度
word-spacing	wordSpacing	指定单词之间的额外间距，取值为 normal 或长度值
z-index	zIndex	控制元素的堆叠，取值为整数，也可以是负数，数值越大，越在上层

参 考 文 献

1 [美]R.Allen Wyke 等著. 深入学习：JavaScript 开发与实例. 陈建春等译. 北京：电子工业出版社，2000

2 [美]Microsoft Corporation 著.动态 HTML 参考和开发应用大全. 北京华中兴业科技发展公司译. 北京：人民邮电出版社，2000

3 [美]Steven Holzner 著. JavaScript 使用详解. 丁利剑等译. 北京：机械工业出版社，1999

4 赵十年著.最新 HTML4.0 网页制作教程. 北京：机械工业出版社，2000

[General Information]
书名=JavaScript实例教程
作者=
页数=327
SS号=10331518
出版日期=