北京科海培训中心

网络程序员入门

裘民民 戴小林 罗　畅　编著

北京科海集团公司　　出品

2001.10

内 容 提 要

《新概念 HTML+JavaScript+ASP 三合一教程》是科海集团公司出品的新概念系列教程之一。本书深入浅出地介绍了当前网页编程的 3 种最基本的利器，包括 HTML、JavaScript、ASP。全书依此分为相对独立的 3 部分，共 24 章。每一部分均从基础入手，涵盖了基本应用的各个方面，辅以切合实际的实例分析，力求与应用紧密结合。

本书还配有多媒体教学光盘，通过程序代码的演示过程和语音讲解，读者能够更加直观地学习到编程的技巧，领略到编程的乐趣，且更易于读者接受书本中的知识。

本书内容丰富，覆盖面广，结构合理。适合于网页编程人员作为入门学习手册，也可作为高手的参考资料，特别适合高等院校及培训班用作培训教材。

书　　名：新概念 HTML+JavaScript+ASP 三合一教程
作　　者：裘民民　戴小林　罗　畅
责 任 编 辑：张雁芳
出　　品：北京科海集团公司
印 刷 者：北京门头沟胶印厂
发　　行：新华书店总店北京科技发行所
开　　本：787×1092　1/16　　印张：27.25　　字数：643 千字
版　　次：2001 年 10 月第 1 版　　2001 年 10 月第 1 次印刷
印　　数：0001~5000
盘　　号：ISBN 7-89998-025-9
定　　价：42.00 元（1 张多媒体光盘）

前　言

在当前的网页编程中，HTML、JavaScript、ASP 是应用最广、最基础的利器。

HTML，即超文本标记语言，是一种用于编写超文本文档的脚本语言。自从 1990 年首次应用于网页编辑后，HTML 迅速崛起成为网页编辑的主流语言。

JavaScript 是 WWW 上一种功能强大的编程语言。用 Sun 公司的话来说，“JavaScript 是一种易于使用的对象描述语言，它是为了创建真正的联机应用程序而设计的，无论在客户端还是服务器端，该种应用程序都将对象和资源链接在一起。

JavaScript 的前身称为 LiveScript，它是 Netscape 公司为了进一步扩充其浏览器的功能而开发的一种可以嵌入 Web 页面中的脚本语言。后来，Sun 公司开发的 Java 语言的流行促使 Netscape 公司重新设计了 LiveScript，并改名为 JavaScript。之所以叫 JavaScript，原因在于 JavaScript 作为一种嵌入 HTML 文档、基于对象的脚本设计语言，其语法和 Java 语言极为相似。

ASP（Active Server Pages）是指一种服务器端的脚本环境。ASP 页就是包含 HTML 标记、文本以及脚本命令的文件。ASP 页可以调用 COM（Component Object Model，组件对象模型）组件执行任务，如连接到数据库或执行商业计算等。

WWW 网络发展到今天，已经进入了一个比较成熟的时代，少了许多喧闹和泡沫，多了冷静和理智，应该说现在正是一个发展酝酿时期。作为一名网络技术人员，现在最重要的就是打好自己的基础，充实自己的知识。为了能让读者获得更系统和更全面的知识，我们编著了这本《新概念 HTML+JavaScript+ASP 三合一教程》，全书分为 3 个部分，分别讲述了 HTML、JavaScript、ASP 等 3 种基本的网络开发技术，是一本引导读者进入绚丽的网络开发殿堂的书籍。本书采用通俗易懂的语言描述了这 3 种基本开发技术，阅读本书不要求读者具备其他网络开发技术基础，适合作为大中院校培训教材。

参与本书写作的人员还有胡苏、赖晓铭、王琪、王萍等。由于时间仓促，书中难免有疏漏之处，欢迎读者批评指正。

作者

2001 年 10 月

目　录

第 1 部分　HTML

第 2 部分 JavaScript

第 3 部分 ASP

第 1 部分　HTML

第 1 章　Internet 与 World Wide Web

本章主要介绍 Internet 的一些基本知识。读者可以从中了解 Internet 的产生、发展和应用现状，了解 WWW 的基本概念，同时对 HTML 也有一个初步的认识。

1.1　Internet 的发展与主要服务

计算机的诞生给人类的生活方式带来了巨大的变革，现代生活已越来越离不开计算机。在现今的网络时代，大家只要一进入 Internet，那无穷无尽的内容就让我们目不暇接，而那些精彩的界面更会给我们带来美的享受。

1.1.1　Internet 的起源与发展

Internet 起源于 60 年代后期美国国防部创建的 ARPANET（Advanced Research Projects Agency Network，美国国防部高级研究计划署网络）。ARPANET 是冷战时期的产物，美国军方建立它的目的是要把分布在各地的不同计算机系统通过各种介质连接起来，在由于战争等原因导致部分网络出现故障时，其余部分能够自动配置，而不影响彼此之间的通信，从而避免造成整个网络的崩溃。

从 1975 年开始，ARPANET 完成其试验阶段而正式投入运行，它最初采用“主机－主机”协议进行通信，主要用于军事部门的信息传递。在 ARPANET 运行期间，美国国防部高级研究计划署资助加州大学伯克利分校的研究人员开发了一种新的网络通信协议，即现在被广泛使用的 TCP/IP 协议。该协议在 1983 年被采纳为美国军方标准。

进入 80 年代初期，TCP/IP 协议获准用于非军事领域。伯克利分校将 TCP/IP 协议融入其 UNIX 操作系统（即 BSD UNIX），并将该操作系统在各大学和研究机构分发，使它获得了广泛的应用。自此，ARPANET 从单纯的军用信息网逐渐发展成为今天的 Internet，美国国防部也开始重建自己的专用网络——MILNET。由于 TCP/IP 协议是作为一种开发技术加以公布的，许多公司可以自由使用它开发 TCP/IP 应用软件，从而有力地推动了 Internet 的发展。

在 Internet 的发展过程中，美国国家科学基金会（National Science Foundation，简称 NSF）所建立的国家科学基金网（NSFNET）功不可没。在 80 年代后期，NSF 在美国建立

了 5 大超级计算中心，各大学可以就近连接到这些超级计算中心，并且使各个超级计算中心彼此之间互联起来，这就形成了美国国家科学基金网——NSFNET。NSFNET 采用 IP 通信协议，该网建成后很快就取代 ARPANET 而成为 Internet 的主干网。

随着各大学和科研机构的加入，Internet 获得了快速的发展，同时也大大丰富了网上的信息资源。随着网上信息的增加，用户从 Internet 这个信息的“汪洋大海”中查找他们所需要的信息也变得更加困难。为了便于检索，各种 Internet 服务（如 Gopher、FTP 和 WWW 等）被相继开发，这使 Internet 真正发展成为以信息为目的的计算机网络。由于这一时期的 Interne 主要用于教育和科研机构的非商业应用，网络安全没有受到足够的重视，这给后来的 Internet 商业应用留下了安全隐患。

到了 90 年代初，一些有远见的公司已经意识到 Internet 的商业价值，于是纷纷加入 Internet，通过它开展产品宣传、技术支持、用户培训、产品销售等工作。在这一时期，网络规模呈爆炸性增长态势，使 Internet 真正发展成一个国际性的互联网络。

在我国，中国科学技术网（CSTNET）于 1994 年 4 月，首次实现了与 Internet 的直接连接。从 1994 年 5 月开始，中国科学院计算机网络信息中心即着手开展国家级的 Internet 信息中心的工作，开始为中国 Internet 用户提供域名注册服务。通过在网络上设立 WWW、Gopher、FTP、Whois、News、Mail 等多种网络服务器，为中国 Internet 用户提供目录数据库服务和信息服务。

近几年来，Internet 在我国发展十分迅速。目前，已先后建成了中国科学技术网（CSTNET）、中国公用计算机互联网（CHINANET）、中国教育和科研计算机网（CERNET）、中国金桥信息网（CHINAGBN）等。为了适应我国 Internet 发展的需要，更好地为我国的 Internet 用户提供服务，原国务院信息化工作领导小组办公室委托中国科学院在中国科学院计算机网络信息中心组建了中国互联网络信息中心（CNNIC），由它行使国家互联网络信息中心的职责，为用户提供域名注册、IP 地址分配、自制系统号分配、反向域名登记等服务。这标志着 Internet 在我国的发展、运行和服务走入了更加有序、完善和规范的轨道。

1.1.2 TCP/IP 网络体系结构

TCP/IP 协议是 Internet 上通用的标准协议，用于解决一个网络中的不同操作系统之间的互连问题。TCP 协议和 IP 协议总是成对出现，其实它们是两个完全独立的协议。

IP（Internet Protocol，网间协议）协议执行的操作可映射到 OSI（Open System Interconnection，开放系统互连）模型的“网络层”。在这一层上，软件将信息封装为数据包，然后将这些数据包传送到其他主机或网络。在 IP 协议层，这些数据包相互间是独立的。

IP 协议既不保证所有这些数据包都能够到达目的地，也不保证任意两个数据包会经过相通的路由而到达。因为接收到的这些数据包的顺序与其发送时的顺序可能不同，所以 IP 协议无法实现上述两个保证。

TCP（Transmission Control Protocol，传输控制协议）协议执行的操作可映射到 OSI 模型的“传输层”，它的作用是将数据进行格式化并传递给 IP 层进行发送。TCP 协议对从 IP 层接收到的数据包进行记录并校验，从而确保其被安全无误地接收。TCP 协议传送数据时，会设置一个定时器，当远程计算机接收到一个数据包时，会送回一条确认消息给发送者。如果在定时器设置的时间范围内没有收到这条确认消息，那么该数据包就会重新发送。

TCP/IP 协议从功能和概念上描述网络，是一个庞大的、具有层次结构的计算机网络协议体系。这套协议是一个开放的通信标准，它允许不同类型网络上的节点计算机之间的通信，这正好符合 Internet 互联性的要求。目前 TCP/IP 协议已经在 Internet 上得到了广泛的应用和发展。

TCP/IP 协议不是仅仅由 TCP 协议和 IP 协议组成，而是由一系列通信协议所组成的协议族，由这些协议来共同控制各主机与网络之间的数据交换。在协议的网络连接层中，使用 TCP/IP 协议规定了网络中的计算机之间互相通信的规则，在网络层中使用的是 IP 协议，在传输层中则可以使用 TCP 协议和 UDP（User Datagram Protocol，用户数据报协议）协议。下面分别介绍一下 TCP/IP 协议族中的一些协议：

- IP 协议：即网间协议。如前所述，它主要负责基本的网络连接，能够确保数据发送准确无误。
- TCP 协议：即传输控制协议。如前所述，它能够保证数据传输的完整性，保证网络中计算机之间数据流的可靠性，同时还可以检查错误和序列编号。
- SMTP 协议：即简单邮件传递协议（Simple Mail Transfer Protocol）。它主要用于 Internet 上的电子邮件传送，是网络中的一个标准协议，许多使用这个协议的通信软件都可以自动地收发电子邮件，并对邮件传送过程中出现的错误做出相应的处理。
- FTP：即文件传输协议（File Transmission Protocol）。它主要用于在 Internet 上的两台计算机之间进行文件传送。这些文件可以是包括 ASCII 码二进制文件在内的多种格式的文件。
- Gopher 协议：一个应用层的通信协议。它以客户机/服务器（Client/Server）的方式供用户检索 Internet 上的数据信息，是网络上使用频率最高的检索系统。
- HTTP 协议：即超文本传输协议（Hyper Text Transfer Protocol）。HTTP 协议也是应用层的通信协议，它利用 TCP 协议在 Internet 上传输文本信息。
- Telnet 协议：即远程登录协议。Telnet 是一种应用层的通信协议，它可以使 Internet 上的一台计算机仿真一个终端，并与网络上的任何一台主机相连。本地计算机作为网络上的远程计算机的一个虚拟终端，可以使用远程计算机中的数据库和其他有关服务。

1.1.3　IP 地址和域名系统

为了实现 Internet 上各个主机之间的通信，每个主机都必须用一个地址来标识自己。为了保证其有效性，该地址在 Internet 上必须是唯一的。在 Internet 世界中有两种主要的地址标识系统：IP 地址和域名系统。IP 地址是基于数字的名字，但实际上是逻辑地址；域名系统很像英语，类似于地址，但实际上是名字。

IP 地址实际上就是通过数字来表示一台主机在 Internet 中的位置，它由两部分组成：网络号（Net ID）和主机号（Host ID）。网络号用于表示网络部分，主机号用于表示主机部分。为了避免在地址分配中产生冲突，整个 Internet 的地址由网络信息中心（NIC）统一进行分配。

IP 地址可以分为 5 类，见表 1.1。

表 1.1 IP 地址格式及分类

IP 地址格式	IP 地址分类
二进制 0+7 位网络号+24 位主机号	A 类
二进制 10+14 位网络号+16 位主机号	B 类
二进制 110+21 位网络号+8 位主机号	C 类
二进制 1110+28 位多点播送地址	D 类
二进制 11110+27 位保留地址	E 类

A 类地址适用于网络数量少而网络内主机数量较多的网络。

B 类地址适用于中等规模的网络。

C 类地址适用于主机数量少的小规模的网络。

D 类地址主要用于多点播送。

E 类地址目前暂时保留，以供今后使用。

如果通过拨号网络并作为终端用户进入 Internet，就不必拥有一个 IP 地址。当然，通过拨号所连接的计算机需要有它自己的 IP 地址。如果通过拨号建立网络连接，就需要一个 IP 地址，但这一地址是在拨号期间由系统自动建立的，当下一次拨号进入网络时，也许会得到另外一个地址。只有一直连接在 Internet 上的计算机才有自己的永久 IP 地址。

由于 IP 地址是一长串数字（如 169.254.34.148），不符合人们的使用习惯，也难以记忆，因此又引入了域名的概念。将整个 Internet 划分成为不同的域，各域按照组织或行政管理层次把名字空间划分成树状结构，然后给每个节点（也就是域）分配一个标识，一个节点的域名就是从根到该节点间所有域标识的组合，这些标识从右至左排列，相互之间用点“.”隔开。顶级域名的划分有按组织分类（如 com 代表商业组织，edu 代表教育机构，org 代表政府机构等）和按国家/地区分类（如 cn 代表中国，hk 代表香港，tw 代表台湾，us 代表美国等）两种。例如：www.sohu.com，其中，com 为分配给商业企业类的顶级域名，sohu.com 是搜狐公司的注册域名。www.sohu.com 域名标识搜狐公司的 WWW 服务器主机。

1.1.4 Internet 的应用服务

Internet 发展过程中，许多单位和组织根据需要不断开发出各种服务项目。目前常用的 Internet 服务项目包括域名解析服务（DNS）、电子邮件服务（Email）、文件传输服务（FTP）、远程登录服务（Telnet）、新闻服务（News）、文档查询服务（Archive）、Gopher 服务和 WWW 服务，等等。

下面简单介绍一下这些服务。

1. 域名解析服务

使用域名便于人们记忆，在网络通信时系统所使用的标识仍然是节点的 IP 地址，这就经常需要将 IP 地址转换成域名，或反过来将域名转换成 IP 地址，这两个过程称作域名的解析和反向解析。最早的域名解析工作是通过在每个主机上配置一个域名数据库文件（文

件名称通常为 hosts，在 Windows NT 系统中，存放在\winnt\system32\drivers\etc 目录下），在该文件里定义了网络中各节点的域名与其 IP 地址之间的对应关系。当网络通信需要进行域名解析时，系统从 hosts 文件中便可查找到目的节点的 IP 地址。在 ARPANET 建立初期，由于网络规模很小，网络中的主机数量有限，所以使用这种方法很容易实现域名解析。

随着网络规模的增大，网络节点数迅速增加，在每增加或删除一个节点时就需要修改所有主机中的 hosts 文件，这给域名的管理和维护工作带来很大困难，再采用这种方法显然是不合适的。为了克服 hosts 文件的这一弊端，于是设计了一种新的域名解析方案——DNS（Domain Name System，域名系统）。

DNS 的设计思想是将域名解析工作由 hosts 数据库的集中管理方式转换为分布式管理，即在不同的域中建立域名服务器，每一个域名服务器在其上一级域名服务器中进行注册，它负责其辖区内各节点的域名解析工作。如果用户要解析的域名不能从它所在域的域名服务器中解析出来，那么该域名服务器就将这一请求提交给根域名服务器，然后再逐级下传给指定的子域域名服务器进行解析，最后将解析后的结果返回给客户端解析器。

DNS 的核心是一个在域名服务器上运行的操作系统守护进程（在 Windows NT 中它作为服务或应用程序启动）和域名数据库文件。守护进程响应用户的解析请求，域名数据库文件记录了域名服务器所管辖区域内各节点的 IP 地址，其作用与 hosts 文件相同。当在一个域中增删网络节点时，只需更改该子域域名服务器中的数据库文件即可。

目前，大多数的 UNIX 操作系统（如 SGI IRIX、SUN Solaries 和 BSD UNIX 等）都具有 DNS 功能，Windows NT 操作系统从 4.0 版本开始也增加了域名服务功能。

2. 电子邮件服务

电子邮件简称 Email，它是 Internet 中使用最早、最广泛的一种服务，它具有简单、快捷、费用低廉等优点。所以，所有的 ISP（Internet Services Provider，Internet 服务提供商）和 Intranet 系统都将电子邮件作为一项基本服务提供给用户。

Email 的工作过程类似于传统的邮政系统，电子邮件服务器相当于邮局，用户的电子邮件账户相当于用户地址，服务器为用户分配的存储空间相当于用户信箱。用户要发送电子邮件时，首先通过网络或拨号等方式与邮件服务器建立连接，然后通过客户端邮件收发软件（如 Microsoft Internet Mail、Outlook、Foxmail、Eudora、Netscape Messenger 等）将编写好的邮件发送给电子邮件服务器，服务器再按一定的路由将邮件传递给接收者。电子邮件服务器常使用存储转发方式处理用户邮件，在用户未提取邮件之前它将邮件一直保存在用户信箱中。电子邮件除了能够实现传统意义上的邮件收发功能外，它还可以与 FTP 等 Internet 服务联合使用，为用户提供更广泛的服务。

3. 文件传输服务

文件传输服务使用 TCP/IP 协议族中的 FTP 协议进行工作，所以常叫做 FTP 服务，它是目前使用最普遍的一种文件传输方式。用户在使用 FTP 服务传输文件时，首先需要登录到对方主机上，然后就可以在二者之间传输文件。与 FTP 服务器联机后，用户还可以远程执行 FTP 命令，浏览对方主机的文件目录结构，设置文件的传输格式（以文本格式或二进制格式传输）等。

FTP 软件有字符界面和图形界面两种，后来又开发出了具有断点续传功能的 FTP 软件，

这种软件可以在线路中断再次恢复连接后，接着上次中断的地方继续传输，这对于在网络上传输大型文件非常有用。常用的 FTP 软件有 CuteFTP、LeapFTP、LeechFTP、WS_FTP 等等。

4. 远程登录服务

远程登录服务使用 Telnet 协议使用户登录到远程主机。成功登录后，本地计算机作为远程主机终端，用户通过它访问远程主机资源。使用远程登录服务时，用户必须是远程主机的合法用户。

5. 新闻服务

Internet 的新闻服务相当于一个大的电子公告板，它实际上是一个论坛，用户可以通过它阅读新闻服务器中的消息，也可以向新闻服务器发送新闻。新闻服务器的主要信息来源是 Usenet 网络。为便于用户检索，Usenet 将各种信息进行逐级分类，第 1 级分为以下 7 类：

- comp 类：计算机技术。
- misc 类：杂类，包括广告等。
- news 类：关于 Usenet 自身的一些新闻，如管理、通告等。
- rec 类：有关体育、艺术、爱好等娱乐方面的话题。
- sci 类：涉及自然科学方面的新闻。
- soc 类：关于文化、妇女等社会问题。
- talk 类：各种话题的一般讨论和闲聊。

6. 文档查询服务

文档查询服务（Archive）的功能是帮助用户在 Internet 上的 FTP 服务器中查找用户所需的文件。为了实现文档查询服务功能，文档查询服务器定期读取 Internet 网上 FTP 服务器中的文件信息，然后产生文件信息数据库，供用户检索使用。文档查询服务器可提供联机查询和电子邮件方式查询两种服务。

7. Gopher 服务

Gopher 的原意是指美国明尼苏达州特产的一种穴居哺乳动物花栗鼠，Internet 中的 Gopher 服务以此动物的名称命名也正指明了该服务的发源地——明尼苏达大学。Gopher 软件于 1991 年 4 月开发成功，最初是为了满足明尼苏达大学的学生查询信息的需要而设计的。

Gopher 服务与 FTP 服务的不同之处在于它提供了一种交互式的菜单界面。用户与 Gopher 服务器连接后，可以在菜单的引导下访问自己感兴趣的信息资源。因为 Gopher 服务的操作十分简便，所以在 Web 技术出现以前，它是 Internet 上主要的信息检索方法。

8. WWW 服务

WWW（World Wide Web）服务是 Internet 信息服务的核心，使用它既可以检索纯文本信息和多媒体信息，又能够以交互方式操作数据库，是目前 Internet 上使用最广泛的信息服务。下一节将详细介绍 WWW 服务及其技术。

1.2　World Wild Web

World Wild Web 是从理论到实践的典范，其成功有着深厚的基础。这一节将介绍一些关于 WWW 的基本概念和 WWW 的浏览器。

1.2.1　什么是 WWW

World Wide Web，国内一般翻译为万维网，其缩写为 WWW。它集中了全球网络上的大量信息，并且使用简单的前端和服务器协议来实现这一服务。目前，计算机网络中包含了许多多媒体信息，在这种情况下，WWW 正好提供了一种统一的方式来获取各种类型的信息，如访问数据库等。

WWW 的历史十分短，但是其发展却十分迅速。在 1990 年 11 月，Next 开发了最初的 WWW 模型，同年诞生了 Next Step 浏览器，它能够访问超文本和 Internet 新闻组，到 1994 年底，全世界就已经有超过一万台的 WWW 服务器。WWW 之所以迅速增长，首先是因为 Internet 的迅速发展。目前 Internet 已经发展成为名副其实的全球性网络，在这种情况下就必然需要一种能链接网上所有资源的应用系统。其次，越来越多的用户开始使用 Internet，商业部门也很快接受了多媒体概念，同时，随着超媒体和多媒体软件的开发，在 WWW 中引入了超链接技术，这些超链接使不掌握多媒体技术的用户也可以在不同的文本和媒体资源之间建立链接，撰写自己的媒体文件，在浏览 WWW 时通过激活嵌入的超链接（Hyperlink）来进行浏览。

WWW 由客户机（Client）、服务器（Server）和 HTTP 协议组成，它以 Client/Server 方式工作。实际的工作过程就是客户机向服务器发送一个请求并从服务器上得到一个响应，服务器负责管理信息并对来自客户机的请求作出应答。客户机与服务器都是用 HTTP 协议来传送信息，而信息的基本单位就是网页。当选择一个超链接时，WWW 服务器就会把超链接所附的地址读出来，然后向相应的服务器发送一个请求，要求相应的文件，最后服务器对此作出响应，将超文本文件传输过来。

WWW 的服务范围包括学术交流与研究、电子邮件、网络应用、可视会议、多媒体教学和视频节目等。

1.2.2　WWW 浏览器

人们将在 Internet 中浏览各 Web 站点称为网上“冲浪”。近年来网上“冲浪”已成为一项时髦的休闲娱乐活动。的确，面对 WWW 所提供的丰富多彩的内容，又有谁不为之心动，不想在 Internet 这浩瀚的海洋中“冲冲浪”呢？可是不要忘了，“冲浪”是离不开“冲浪板”的，而网上“冲浪”所需的“冲浪板”就是 WWW 浏览器（Browser），简称 Web 浏览器。

浏览器这一名词对于广大计算机爱好者来说可能并不陌生。浏览器其实就是一个可用来寻找和显示各种 Web 页的应用软件，它能够对 HTML 语言所创建的文档作出解释，并最终将那些分散在各 Web 站点上的信息资源展现在我们的眼前。

浏览器的历史并不算久远，最早的 Web 浏览器是 Mosaic 浏览器，它是由 NCSA（National Center for Supercomputing Applications）于 1993 年开发的。最初它作为免费软件向全世界

进行公开发布，但在 1994 年，NCSA 将浏览器的商业开发交给了 Spyglass 公司，现在该公司将浏览器命名为 Spyglass Mosaic。Mosaic 浏览器对于 WWW 在全球范围的迅速普及起到过极其重要的作用，并为浏览器今后的发展奠定了基础。

在浏览器历史中另一个重要的里程碑是 1994 年由 James H.Clark 和 Marc Andreessen 共同创建了 Netscape Communications Corporation（简称 Netscape，网景公司）。Marc Andreessen 原是 Mosaic 浏览器的开发人员，在成立了网景公司后，推出了自己的浏览器产品，这就是著名的 Netscape Navigator 浏览器（简称 Navigator）。由于 Navigator 的优越性能以及它采用了免费发布的新举措，所以很快就占领了浏览器市场的大部分份额，出现了一统浏览器市场的局面。Microsoft 公司在看到 Internet 这一具有广阔潜力的市场之后，也开始涉足浏览器市场。尽管在开发浏览器方面它比 Netscape 公司要晚，但是，Microsoft 公司凭借自己雄厚的技术、人力和物力优势很快就开发出了自己的浏览器产品，这就是另一个著名的浏览器 Internet Explorer（简称 IE）。于是，Navigator 与 IE 之间展开了一场浏览器大战。

开始时，由于 IE 在性能上与 Navigator 存在着一定的差距，所以，在竞争中处于下风。可是，随着其版本的不断更新，这两种浏览器在性能上已逐渐变得旗鼓相当，且各具特色。Navigator 所占据的市场份额正日益减小，这两种浏览器的竞争变得越来越激烈。Navigator 和 IE 已经成为当前最为流行的两种浏览器。

上面所介绍的 Mosaic、Navigator 和 IE 这 3 种浏览器都是基于图形界面的浏览器，它们既能显示普通的文本信息，也能显示图形、声音等各种多媒体信息。目前大多数用户所使用的浏览器都是基于图形界面的浏览器。另外还有一种基于字符界面的浏览器，例如 Lynx。如果使用这种基于字符界面的浏览器，那么在浏览 Web 页时只能看到其中的文本信息，而 Web 页中的其他诸如图像、声音等多媒体信息则将使用[image]、[sound]等标记进行代替。在使用上，这种浏览器与基于图形界面的浏览器大致相同。

一个 Web 浏览器应具备以下功能：

- 可以以适当的格式去请求 Internet 上的各种信息资源。
- 能够接收并显示来自服务器的各种信息，包括普通文本图像以及声音等多媒体信息。
- 能够保存所浏览过的文档以及 URL 地址以供以后再进行使用。
- 能够有效地管理内存以减少不必要的下载。
- 具有可扩充能力，即能够利用其他第三方应用程序和插件来扩充自己的性能。
- 可自行定制工作环境，如设置显示字体、颜色、下载方式等。
- 能够提供诸如书签（Bookmark）、主页（Homepage）和搜索引擎等访问服务。
- 能够提供安全可靠的通信，包括加密和解密。

当我们在网上查询信息时，经常会遇到许多不同种类的文件格式。在这些文件格式中，有些格式可以使用浏览器进行直接处理，而有一些则需要其他第三方软件来进行处理。了解一些网上常见文件格式对于我们浏览各种信息是有帮助的。下面介绍 Internet 上的一些常见文件格式：

- .html（.htm）：超文本文档格式，是 Internet 上一种最常见的文件格式，在浏览器

中可以直接浏览。

- .jpeg：一种图像文件格式，在浏览器中可以直接浏览。
- .gif：一种图像文件格式，在浏览器中可以直接浏览。
- .tif（.tiff）：一种图像文件格式，可用 PolyView 等软件进行浏览。
- .au：一种声音格式（在 Sun 工作站上使用），Navigator 中的 LiveAudio 插件可用来播放此格式的声音文件。
- .wav：一种声音格式（在 Windows 中使用），Windows 中的“媒体播放器”可播放此格式的声音文件。
- .mp3：一种声音格式，可用 Winamp 等软件播放此格式的声音文件。
- .aif（.aiff）：一种声音格式（在 Macintosh 中使用）。
- .avi：一种视频格式，Windows 中的“媒体播放器”可播放此格式的视频文件。
- .rm：一种视频格式，可用 RealPlayer 等软件播放此格式的视频文件。
- .mov（.moov）：一种视频格式，可用 QuickTime 软件播放此格式的视频文件。
- .mpeg（.mpg）：一种视频格式，可用超级解霸软件播放此格式的视频文件。
- .zip：一种压缩文件格式，可用 WinZip 软件对其进行压缩和解压缩。
- .arj：一种压缩文件格式，可用 Arj 软件对其进行压缩和解压缩，用 WinZip 进行浏览和解压缩。
- .gz：UNIX 中的一种压缩文件格式，可用 WinZip 和 gzip 软件对其进行处理。
- .z：UNIX 中的一种压缩文件格式，可用 WinZip 软件对其进行处理。
- .tar：UNIX 中的一种归档格式，可用 tar 命令进行处理。tgz 和 taz 是以 gz 和 z 方式进行压缩的.tar 文件。
- .txt：文本文件，在浏览器中可以直接进行浏览。
- .ps：Postscript 文件，是一种格式化的文本文件，可用 GhostViewer 软件读取。
- .pdf：Adobe Acrobat 公司的一种文件格式，可用 Adobe Acrobat Reader 进行阅读。
- .bin： Macbinary II 编码文件，可用 Stuffit Expander 将其还原为可用的 Macintosh 文件。
- .hqx：一种编码文件，通过 BinHex 将 Macintosh 文件编码为 7 位文本文件以进行传输。大多数 Mac 文件以.hqx 的形式出现，可用 Stuffit Expander 对其进行处理。
- .uu（.uue）：uuencode 编码文件，在 UNIX 中经常用 uuencode 命令将二进制文件进行编码以进行传输。可用 UU Undo、WinCode 或 WinZip 对其进行处理。

1.3　HTML 简介

HTML 的全名是 Hyper Text Markup Language（超文本标记语言），可以把它理解为一种用来生成活动文档浏览效果的代码系统。HTML 的最大特点是提供了强大的超链接（Hyperlink）功能，它增强了文档之间的关联性，使得浏览者能够准确、快速地从一个文档跳转至另一个文档，从而极大地方便了文档的检索。

HTML 的使用非常简单。它有其自身的标准，或者说语法规范。与其他各种规范一样，

必须要有一个组织来负责规范的制定工作。

1994 年，W3C（万维网联盟）在麻省理工学院（MIT）成立，负责管理 Web 标准的发展，其中就包括制定 HTML 标准。访问 W3C 的网站 http://www.w3.org 可以获取最新的 HTML 标准。

由于 HTML 标准是完全开放的，所以任何人、任何浏览器都可以使用和拓展这一标准。如今两个主要浏览器 Navigator 和 IE 的竞争，就是这一标准开放性的体现。这种竞争从某种意义上说，促进了 HTML 标准的发展。

随着 WWW 用户的不断增加，对网络的期望和要求也变得越来越高。在 HTML 3.2 的基础上，HTML 4.0 规范应运而生，并立即成为浏览器生产厂家新的竞争砝码。IE 4.0 和 Navigator 4.0 及其更高版本基本上都支持 HTML 4.0 规范。与以前不同的是，这两个浏览器都还各自支持一些 HTML 4.0 中所没有的特性。虽然这些特性的确会为 WWW 浏览带来更新、更妙的效果享受，但也宣告了 HTML 标准分化的开始。这一分化的直接后果就是导致 HTML 最明显的跨平台优势开始逐步丧失，即在不同的操作系统平台上，甚至在同一平台的不同浏览器上，同一个 HTML 文档的浏览效果也可能会有明显的不同。或许今后发布的更新的 HTML 规范能完全兼容这些拓展了的特性。

1.4 小 结

本章主要介绍了有关 Internet 和 WWW 的一些基础知识，如 Internet 协议以及所提供的服务等。并且介绍了浏览器的有关知识以及 Internet 中的一些常见文件格式。最后对 HTML 作了简单的介绍，从下一章开始将详细介绍 HTML。

第 2 章　HTML 的基本概念

HTML，即超文本标记语言，是一种用于编写超文本文档的脚本语言。自从 1990 年首次应用于网页编辑后，HTML 迅速崛起成为网页编辑的主流语言。几乎所有的网页都是由 HTML 或以其他脚本语言嵌入 HTML 中编写的。HTML 具有平台无关性，无论用户使用何种操作系统（如 Windows、DOS、Unix 等等），只要有相应的浏览器程序，就可以运行 HTML 文档。

本章主要讨论 HTML 的基本概念。

2.1　HTML 文档的特点

Web 上的信息是以页面（Page）的形式组织起来的，Web 页面由 HTML 描述。Web 页面与 HTML 文档实际上是同一事物的两个不同侧面：人们通常把用 HTML 编写的文件称为 HTML 文档，而把 HTML 文档在 Web 浏览器中的表现形式称为 Web 页面。

Web 页面中有一种称为主页（Homepage）的特殊页面。主页通常是用户通过 Web 浏览器访问某一 Web 站点（或称为网点）时看到的第 1 个页面，用户通过主页可以访问该站点的其他页面。因此主页一般是关于某一组织、机构或个人的基本信息的页面。

HTML 文档是一种纯文本文档，但是与普通的纯文本文档相比，HTML 文档具有下述特点：

- HTML 文档中使用各种标记来标注文档的不同逻辑单元，如标题、段落、列表和表格等，因此 HTML 文档具有特定的逻辑结构，是一种结构化的文本文档。
- HTML 文档中一般会有许多指向其他 HTML 文档以及指向图像、动画、音频和视频等多媒体信息的超链接，因此 HTML 文档是一种超文本文档。
- HTML 作为 Web 上通用的信息描述语言，把分布广泛的不同类型的信息资源连接在一起，为各种平台提供了一个公开的标准接口，因此 HTML 文档可以应用在各种不同的平台之上，即具有平台无关性。
- 简单、易维护。HTML 易学易用，创作 HTML 文档的过程非常简单，这也是 Web 迅速发展的一个重要原因。HTML 版本的开发采取向后兼容的方式，新的版本一般是原有版本的超集，从而使 HTML 文档的十分易于维护。

HTML 文档是一种结构化的文档，这就意味着在编写 HTML 文档之前，应该先编排好其格式，这样可以使文档的可读性更好，更易于维护和扩充。如果只是想编制一个 Web 页面，那么这样做的重要性也许还不那么明显，然而如果要建立与维护一个由许多 Web 页面组成的 Web 站点，那么这样做会使工作更加容易。

HTML 文档有两个主要组成部分：head 和 body，这一点与电子邮件的消息非常相似。本书中将 head 和 body 分别译为“页首”和“正文”。HTML 文档的页首部分称为页首节（Head Section）。页首节一般很短，主要用于提供有关 HTML 文档本身的一些信息。HTML 文档的正文部分称为正文节（Body Section），它是 HTML 文档的主体，定义了 Web 页面的内容。出现在页首节中的 HTML 单元称为页首单元。相应地，出现在正文节中的单元则称为正文单元。

除了页首和正文以外，HTML 文档还有一个序言（Prologue）。序言是 HTML 文档开头的第 1 行文本，用来向 Web 浏览器说明本文档所采用的 HTML 版本。序言是可选的，许多 HTML 文档都没有序言，而且大多数 Web 浏览器也不理睬 HTML 文档的序言，因为 HTML 文档的版本信息已经编码在 Web 服务器所提供的 MIME 内容类型中了。尽管如此，在 HTML 文档中提供序言是一种良好的习惯，因此建议读者在编写自己的 HTML 文档时不要忘了加上序言。

HTML 文档的页首节含有关于 HTML 文档本身的一些信息，页首节中的信息不会显示在 Web 浏览器的文档窗口内。页首节的 head 单元是可选的。如果把所有关于 HTML 文档的信息都放在文档的前部，而把所有正文标记放在文档的后部，那么，即使略去 head 单元，HTML 语法分析器也会很容易判断出文档的页首在哪里结束，正文从哪里开始。

页首节中的 title 单元是必需的，其内容（即文档的标题）将出现在 Web 浏览器窗口的标题栏中，同时还会在书签文件（Bookmark）和搜索引擎（Search Engine）的结果列表中用到。因此，文档标题应该能够概括文档的内容，诸如“主页”和“索引”一类的词就不是好的标题，因为它们说明不了什么问题。页首节中的其他几个单元是可选的，如果只想编写一个 Web 页面，那么它们都可以省略。然而如果要创建一个由许多页面组成的 Web 站点的话，那么 meta、link 和 base 这 3 个单元还是十分有用的。

设计 HTML 文档的结构化内容本身可以说是一门艺术，在开始时可以使用基本结构建立起文档的框架，接着添加段落、列表和表格等单元来组成文档的内容。

HTML 文档通常并不是孤立的，它往往是一个 Web 站点的 HTML 文档集合中的一个成员。对于一个 Web 站点来说，所有的 HTML 文档应该具有一致的风格。

2.2 HTML 的基本语法

HTML 的主要语法是单元和标记。单元是符合 DTD（Document Type Definition，文档类型定义）的文档组成部分，如 title（文档标题）、img（图像）和 table（表格）等等。单元名不区分大小写。HTML 用标记来规定单元的属性和它在文档中的位置。标记分单独出现的标记和成对出现的标记两种。大多数标记是成对出现的，由首标记和尾标记组成。首标记的格式为<单元名>，尾标记的格式为</单元名>。成对标记用于规定单元所含的范围，如<title>和</title>标记用来界定标题单元的范围，也就是说，<title>和</title>之间的部分是该 HTML 文档的标题。单独标记的格式为<单元名>，它的作用是在相应的位置插入单元。如
标记表示在该标记所在位置插入一个换行符。

在介绍 HTML 文档的基本结构之前，首先让我们来认识一下组成 HTML 文档的 3 种

主要标记。

1．<html>标记

<html>标记是文档标识符，它是成对出现的，首标记<html>和尾标记</html>分别位于文档的最前面和最后面，明确地表示文档是以 HTML 编写的。该标记不带任何属性。

事实上，我们现在常用的 Web 浏览器（Navigator 和 IE）都可以自动识别 HTML 文档，并不要求有<html>标记，也不对该标记做任何操作。但是，为了提高文档的适用性，使编写的 HTML 文档能适应日新月异的 Web 浏览器，还是应该养成使用这个标记的习惯。

2．<head>标记

<head>标记用来规定该文档的标题（出现在 Web 浏览器窗口的标题栏中）和文档的一些属性。HTML 文档的标记是可以嵌套的，即在一对标记中可以嵌入另一对子标记，用来规定母标记所含范围的属性或其中某一部分内容。嵌套在<head>标记中使用的主要有<title>标记和<isindex>标记。这些内容将在 2.4 节作详细介绍。

3．<body>标记

<body>标记是成对出现的。在<body>和</body>之间的内容将显示在 Web 浏览器窗口的用户区内，它是 HTML 文档的主体部分。在<body>标记中可以规定整个文档的一些基本属性：

- bgcolor：指定 HTML 文档的背景色。
- text：指定 HTML 文档中文字的颜色。
- link：指定 HTML 文档中待连接的超链接对象的颜色。
- alink：指定 HTML 文档中正处于连接中的超链接对象的颜色。
- vlink：指定 HTML 文档中已连接的超链接对象的颜色。
- background：指定 HTML 文档的背景文件。

在指定对象的颜色时，可以使用该颜色的代码或直接使用该颜色对应的英文单词，例如，如果指定 HTML 文档的背景色为绿色，可以表示为：<body bgcolor="Green">，也可以表示为<body bgcolor="#0080000">。为了便于记忆，建议直接用相应的英文单词指定颜色。body 属性的参数值可以是表 2.1 中的 16 种颜色中的任何一种。

表 2.1　文档主体部分的可用颜色

名称	颜色	名称	颜色
Black	黑色	Red	红色
Lime	石灰色	Maroon	栗色
Gray	灰色	Silver	银白色
Navy	海军蓝	Olive	橄榄绿
Purple	紫色	Yellow	黄色
Aqua	浅蓝绿	Blue	蓝色
Green	绿色	Fuchsia	紫红色
White	白色	Teal	暗蓝绿

一个 HTML 文档的基本结构如下：

```
<html>
</head>
<title>文档标题</title>
</head>
<body>文档的主体部分</body>
</html>
```

其中<html>和</html>分别表示文档的开始和结束，<head>与</head>之间的部分是 HTML 文档的文档头部分，用以说明文档的标题和整个文档的一些公共属性。<body>与</body>之间的部分是 HTML 文档的主体部分。我们下节所介绍的标记，如不加特别说明，均是嵌套在这一对标记中使用的。

2.3 HTML 中的标记

如前所述，一个 HTML 文档包含两个主要的节：head 和 body。每个节都有其允许出现的单元，而每个单元本身也都具有相应的内容模式和上下文条件。一个单元的内容模式指的是哪些单元允许出现在该单元的内部，而上下文条件指的则是该单元允许出现在什么地方。弄清每个单元的内容模式和上下文条件是十分重要的。例如，title 单元的内容模式为 plain text（普通文本），那么在 title 单元中就不允许使用其他标记，否则就是错误的。同时，title 单元的上下文条件是 head，那么 title 单元只能出现在和 head 单元之中，而不允许出现在其他单元的内部，否则也是错误的。

需要说明的是，单元与标记是两个不同的概念，把单元当成标记是不正确的。例如在一个 HTML 文档中，head 单元（即正文节）总是存在的，而不管该文档中是否提供了 head 单元的首标记<head>和尾标记</head>。然而由于大多数情况下，单元由标记标注，而标记则用来标注单元，因此单元和标记通常密不可分。

HTML 共有 67 种标记，相应地也就有 67 种单元，它们可以分成 3 个大的类别，即文档单元、页首单元和正文单元。

文档单元只有以下 3 个，如上一节所述，它们构成了一个 HTML 文档的基本结构。

- html：HTML 文档单元
- head：文档页首节单元
- body：文档正文节单元

页首单元有以下 7 个：

- title：文档标题单元
- isindex：可搜索的索引文档单元
- meta：元信息单元
- link：链接单元

- base：基点单元
- script：脚本单元
- style：风格单元

需要说明的是，head 单元（即页首节）中只能出现上述 7 种标记，如果其他标记或普通文本出现在页首节中，则 Web 浏览器将自动假定页首节已经结束，并且开始显示 HTML 文档的正文内容。

正文单元有 56 个，可以分成两组不同的类型：字块级（Block Level）单元和文本级（Text Level）单元。字块级单元用于形成 HTML 文档的结构，而文本级单元用于组成某个字块的内容。文本级单元也称为字符级（Character Level）单元。

HTML 文档的正文节中可以包含多个字块级单元。如果正文节中出现了普通文本，则 Web 浏览器将假设该文本位于一个段落（p 单元）的内部。

字块级单元共有 23 个，以下为 6 个级别的标题单元：

- h1：1 级标题单元
- h2：2 级标题单元
- h3：3 级标题单元
- h4：4 级标题单元
- h5：5 级标题单元
- h6：6 级标题单元

以下为 8 个列表单元：

- ul：无序列表单元
- ol：有序列表单元
- dir：目录列表单元
- menu：菜单项列表单元
- li：列表项单元
- dl：定义列表单元
- dt：术语单元
- dd：定义单元

其他还有：

- p：段落单元
- pre：预排格式文本单元
- blockquote：块引用单元
- address：地址信息单元
- div：分区单元
- hr：水平标尺线单元
- form：表单单元
- table：表格单元

文本级单元用来标注出现在字块级单元内部的文本。某些字块级单元排斥某些文本级单元。同时，某些文本级单元也只能出现在某些特定的字块级单元内部。严格来说，文本级单元只能出现在字块级单元内部，但是 HTML 有很强的容错性，因此如果某个文本级单元直接出现在正文节中，则 HTML 将自动假设该文本级单元位于某个段落（p 单元）内。

文本级单元共有 34 个，分成 5 组。

第 1 组为以下 9 个字体样式单元：

- tt：电传打字机字体单元
- i：斜体单元
- b：粗体单元
- u：下划线单元
- strike：删除线单元
- big：大字体单元
- small：小字体单元
- sub：下标单元
- sup：上标单元

第 2 组为以下 8 个短语单元：

- em：强调单元
- strong：加重强调单元
- dfn：定义单元
- code：代码段单元
- samp：样例单元
- kbd：键盘输入单元
- var：变量单元
- cite：引证单元

第 3 组为以下 4 个表单字段单元：

- input：输入字段单元
- select：选项列表单元
- option：选项单元
- text area：文本输入窗口单元

第 4 组为以下 4 个表格内容单元：

- caption：表格标题单元
- tr：表行单元
- th：表头单元
- td：表数据单元

第 5 组为以下 9 个特殊文本级单元：

- a：**锚点单元**
- basefont：**默认字体大小单元**
- font：**字体单元**
- img：**内联图像单元**
- applet：Java **小程序单元**
- param：Java **小程序的参数单元**
- br：**行分隔符单元**
- map：**客户端图像映射单元**
- area：**客户端图像映射的热点区域单元**

2.4　页 首 单 元

在 HTML **文档的页首节（也称为** head **节）中，只能包含本节中将要介绍的** 7 **种单元（相应地有** 7 **种标记）。如果除了这** 7 **种单元以外，有其他单元或普通文本在** head **节内出现，则浏览器将假定** head **节在此处已经结束，并且开始显示** body **节（正文节）。**

1．title 单元

title **单元用来设置** HTML **文档的标题，每个** HTML **文档只能有一个标题。**title **是** head **节中惟一必须具有的单元，**HTML **文档只要有** head **节，就必须要有** title **单元。**HTML **文档的标题通常显示在浏览器窗口的标题栏中，此外，文档标题还将用作书签文件中的记录项和搜索引擎结果中的标题。**title **单元的语法规则为：**

一般形式：<title>…</title>
属性：None
内容模式：Plain text
上下文：head

从上面的语法规则可以看出，title **单元中只能含有普通文本（包括字符实体），不能含有其他标记。**

最好为文档选择一个在脱离了上下文时仍能表明其内容的标题，这样，当标题用在书签文件中或处于搜索引擎的结果中时，用户也可以很容易地了解该文档可能会有什么内容。

当一个 HTML **文档没有标题时，浏览器通常会在标题栏中显示该文档的** URL**。不过，为每个** HTML **文档定义一个有意义的标题是一种良好的习惯。**

下面是一个包含 title **单元的实例。**

```
<html>
<head>
<title>This is a test TITLE text.</title>
</head>
<body>
HTML is no real language such as C++ or Pascal,it is just a system for
```

```
describing documents.A WWW browser interpret the HTML- code and display
it.
</body>
</html>
```

浏览结果如图 2.1 所示。

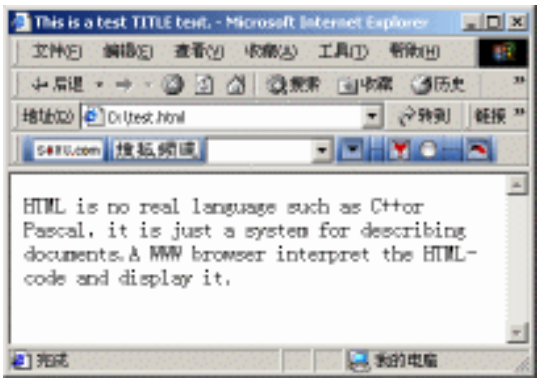

图 2.1 使用 title 单元

2．isindex 单元

isindex 单元用于表示该 HTML 文档是一个索引文档。索引文档提供了一种与用户进行交互的机制。当浏览器遇到<isindex>标记时，将在 Web 页面上提供一个文本输入框，用户可以在该输入框中输入查询关键字，浏览器会负责将查询关键字发送给 Web 服务器，由服务器执行检索，然后返回结果。isindex 单元的语法规则为：

一般形式：<isindex>
属性：prompt=*string*
内容模式：None
上下文：head

isindex 单元有一个可选的属性 prompt，它用来为文本输入框设置一个提示字符串。如果不设置 prompt 属性，那么浏览器将使用默认的提示字符串。

下面是一个使用 isindex 单元的实例。

```
<html>
<head>
<title>这是一个“ISINDEX”标记的范例程序。</title>
<isindex>
</head>
<body>
HTML is a special version of SGML (is used by big companies forexchange
of data) focused on Hypertext.HTML code is written in ASCll.This is a
advantages,because ASCII can be read by about any platform,thus making the
WWW usable for any platform as longas viewer programs,the browsers exist.
```

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

浏览结果如图 2.2 所示。

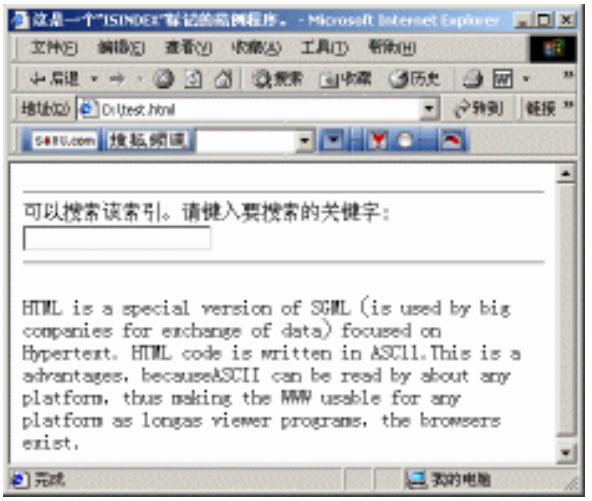

图 2.2　isindex 单元测试文档

图 2.2 中的“可以搜索该索引。请键入要搜索的关键字：”是 IE 给加上的默认提示字符串。浏览者在空白文本输入框内输入要查询的关键字并按回车键确认后，光标将移至该关键字或包含该关键字的词处。此时，与关键字相匹配的字符呈亮蓝色。

由于不同的浏览器使用的默认提示字符串往往并不相同，而且经常与用户要输入的内容毫不相干，因此最好是设置 prompt 属性。例如，可以在上面的实例中加入如下内容：

```
<isindex prompt="请输入您所感兴趣的内容：">
```

则浏览结果如图 2.3 所示。

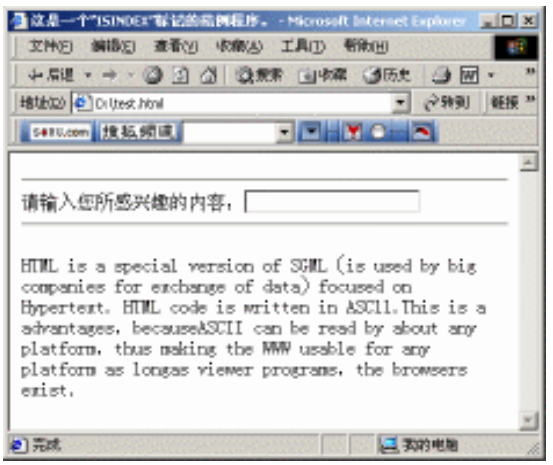

图 2.3　使用 prompt 属性

提示：用户在文本输入框中可以输入的字符个数没有限制。

isindex 单元在 Web 页面中引入的交互式查询机制通常称为基于文档的查询（Document-Based Query），这种交互式查询是在 HTML 引入表单（Form）之前使用的，其交互功能比较有限，只能在 Web 页面中插入一个文本输入框；而表单则可以提供各种类型的输入手段，如文本输入框、下拉式列表、按钮等，因此，用表单代替 isindex 单元将使交互式查询更加灵活。

isindex 单元实际上是由 Web 服务器插入到 Web 页面中的临时 HTML 代码。仅仅在 HTML 文档中人为地插入 isindex 单元并不能使文档成为可以进行交互式查询的索引文档。

3．base 单元

base 单元用来指定一个基点（Base）URL，该 URL 将作为以后所有相对 URL 的起点。用户在阅读 HTML 文档时如果脱离了原来的上下文环境，base 单元就能发挥作用。假设某一个 HTML 文档中某个图像或超链接的 URL 使用的是相对 URL，则当用户将该 HTML 文档下载到自己的计算机上，并以文件的形式保存起来，事后再阅读已存储在本地机器上的该文档时，原 Web 页面中的图像将无法显示，并且超链接也无法激活，因为虽然该 HTML 文档下载到本地机器上了，但是被链接的图像和其他页面并未下载到本地机器上，因而就无法访问。然而，如果在 HTML 文档中提供了 base 单元，则相对 URL 就仍然有效。

base 单元的语法规则为：

一般形式：<base href=*URL*>
属性：href=*URL*
内容模式：None
上下文：head

base 单元有一个必须提供的属性 href，其取值必须是一个绝对 URL，用来表示文档的实际地址。例如：

```
<base href="http://www.acme.com/intro.html">
```

用来表示 HTML 文档中所有相对 URL 都以该 base 单元指定的地址，即 www.acme.com 上的 Web 根目录为基准。于是单元

```
<img src="icons/logo.gif">
```

中的图像的相对 URL 所对应的绝对 URL 为 http://www.acme.com/icons/logo.gif，也就是说该单元等价于：

```
<img src="http://www.acme.com/icons/logo.gif">
```

4．style 单元

style 单元可用于提供“内联的（Inline）”的风格信息。style 单元的语法规则为：

一般形式：<style>…</style>
属性：type=*string*

内容模式：Plain text，but should be valid style markup
上下文：head

目前大多数 Web 浏览器都允许用户定制页面的显示形式。例如，在 IE 中，执行“工具”|“Internet 选项”命令，将弹出如图 2.4 所示的对话框。

图 2.4　“Internet 选项”对话框

单击“字体”按钮，从中可以为许多 HTML 单元指定字型和字体。这样做有其灵活的一面，但也有不利的一面，即 HTML 文档的作者将无法控制自己的文档的页面浏览效果。因此，在 HTML 中引入了风格表。

在 HTML 文档中，可以使用两种方式说明风格表，一种是利用 link 标记指向外部定义风格表的文件，例如：

```
<link rel="Stylesheet" href="mystylesheet.dsssl">
```

另一种是在 style 标记中直接定义当前文档的风格说明，例如：

```
<style type="text/css">
H3{font-size:125%;
font-weight:bold;
color:green;
font-family:Arial,Helvetica,Time;
margin-left:15%;
margin-right:25%;
}
</style>
```

这段代码通知浏览器将 3 级标题显示为绿色的粗体字，而且标题的左侧缩进 15%，右侧缩进 25%，标题采用的字体则按 Arial，Helvetica，Times 的优先顺序选择。

5．script 单元

script 单元用于在 Web 页面中嵌入客户端脚本（Client-Side Script），客户端脚本也称为内联脚本（Inline Script）。script 单元的语法规则为：

一般形式：<script>…</script>
属性：None
内容模式：Plain text，but should be a valid script
上下文：head

script 单元中应该含有合法的脚本。目前并不要求 Web 浏览器使用 script 单元中包含的信息，不支持脚本的浏览器可以将 script 单元中的内容隐藏起来。

如果需要在 HTML 文档中使用脚本，则应该将脚本的内容放到注释中，这样可以避免在不支持脚本的 Web 浏览器上可能出现的问题。

6．link 单元

link 单元用来指明本 HTML 文档与其他 HTML 文档或资源之间的关系，利用 link 单元可以定义一个 Web 站点的结构。link 的中文含义为“链接”。

link 单元的语法规则为：

一般形式：<link rel=*string* href=*URL*>
属性：rel=*string*，rev=*string*，href=*URL*，title=*string*
内容模式：None
上下文：head

link 单元有如下 4 个属性：

- href：指出被链接资源的 URL。
- rel：定义本文档与其他文档或资源的正向关系。
- rev：定义本文档与其他文档或资源的反向关系。具有“rev=关系”的从文档 A 到文档 B 的链接，与具有“rel=关系”的从文档 B 到文档 A 的链接，表达的关系是相同的。
- title：用来给被链接的资源推荐一个标题。

link 单元有如下 3 种常用的使用方法：

（1）用于表示 HTML 文档的作者。

在 link 单元中可以给出作者的 Email 地址，也可以给出作者的 Web 主页地址，此时 rev="made"。例如：

```
<link rev="made" href="mail to:Connolly@w3.org">
```

给出的是 Email 地址，而

```
<link rev="made" href="http://www.w3.org/hypertext/WWW/People/Connolly">
```

给出的则是 Web 主页地址。

有些 Web 浏览器可以用这种方法使用户与 HTML 文档的作者取得联系。

（2）用来指出用于当前 HTML 文档的外部风格表的地址。

例如：

```
<link rel="stylesheet" href="mystylesheet.dsssl">
```

（3）用来指示浏览器自动生成一个导航用的工具栏。

该工具栏中可以包含指向前驱文档和后继文档的按钮，还可以包含指向起始页、帮助页面、目录和索引页面等的按钮。这样生成的工具栏按钮不会随文本的上下移动而滚动。下面给出的是对应各种按钮的 rel 属性取值，对于 rel 的每个可能的取值，href 都既可以指向绝对 URL，也可以指向相对 URL。例如：

```
rel="home"
```

指向本 Web 站点的主页（Homepage），或称为起始页。

```
rel="toc"
```

指向本 Web 站点的目录（Table of Cntents，缩写为 TOC），或一览（Overview）。

```
rel="index"
```

指向本 Web 站点的索引（Index）。索引与目录有所不同，例如，索引可以是按字母顺序排列的，而目录则总是按内容排列的。

```
rel="glossary"
```

指向本 Web 站点的术语词汇表（Glossary of Terms）。

```
rel="copyright"
```

指向本 Web 站点所涉及信息及信息的版权（Copyright）信息页面。

```
rel="up"
```

指向在逻辑上位于当前文档之上的文档。

```
rel="next"
```

指向在文档系列中相对于当前文档的后一个（Next）文档。

```
rel="previous"
```

指向在文档系列中相对于当前文档的前一个（Previous）文档。

```
rel="help"
```

指向关于本 Web 站点的帮助（Help）信息。对于一个结构复杂的 Web 站点来说，提供帮助信息是十分必要的。例如，可以定义下列 link 单元：

```
<link rel="home" href="homepage.html">
<link rel="next" href="person.html">
<link rel="previous" href="department.html">
<link rel="toc" href="toc.html">
```

这些 link 单元依次定义了工具栏上的 Home，Next，Previous 和 TOC 按钮。

7．meta 单元

meta 单元用来描述关于 HTML 文档的元信息（Meta Information）。所谓元信息，就是关于 HTML 文档自身的信息，例如文档的作者、失效日期、关键字列表等，这些信息可以被搜索引擎（Search Engine）、索引程序（Indexer）或其他程序所使用。meta 单元还可以用来为 HTML 文档指定 HTTP 报头（Header）。

meta 单元的语法规则为：

一般形式：<meta>
属性：http-equiv=*string*，name=*string*，content=*string*
内容模式：None
上下文：head

meta 单元使用名称/取值对来描述 HTML 文档的元信息，其中的 name 属性或 http-equiv 属性用来指定元信息的名称，content 属性用来指定元信息的取值。

当 HTML 文档通过 HTTP 协议被检索时，http-equiv 属性具有特殊的意义。HTTP 服务器将使用由 http-equiv 属性所指定的名称来创建 RFC 822 风格的 HTTP 报头，同时将 content 属性的内容作为报头的取值。例如：

```
<meta http-equiv="Expires" content="Tue,20 Aug 2000 14:22:12 GMT">
<meta http-equiv="Keywords" content="HTML">
```

当 HTML 文档被请求时，服务器将生成如下所示的 HTTP 应答报头：

```
Expires:Tue,20 Aug 2000 14:22:12 GMT
Keywords:HTML
```

并非所有的服务器均使用 meta 单元中的信息生成报头，有些 Web 浏览器就会像对待报头一样处理 meta 单元中的信息。

meta 单元有如下所示的一些常见的使用方式：

（1）meta 单元的 lang 属性可与 meta 一起使用来指定 content 属性值的语言。这允许语音系统根据发音规格来应用语言。例如，作者的姓名用法与声明：

```
<meta name="Author" lang="fr" content="Arnaud Le Hors">
```

（2）一些用户代理器提供在几秒钟后通过 meta 来刷新当前页面，或用其他页来代替。如下所示：

```
<meta name="refresh" content="3,http://www.acme.com/intro.html">
```

其中数字指定了延时秒数，在 URL 被调入后计时开始。这个结构通常被用来向用户展示一个飞快的介绍。不过，由于某些用户代理器不提供这个结构，应当在介绍页中包含这个内容来引导用户绕开它（所以还不作为保留的介绍页的组成部分）。

（3）meta 和 HTTP 头

http-equiv 特性可以用在 name 特性的位置上，并且在文档通过 HTTP 协议恢复时有其特殊的重要性。HTTP 服务器可以使用 http-equiv 属性指定的属性名称在 HTTP 响应中生成一个[RFC2068]提示。

下列的实例是 meta 声明：

```
<meta http-equiv="Expires" content="Tue,20 Aug 1996 14:25:27 GMT">
```

HTTP 头表现为：

```
Expires:Tue,20 Aug 1996 14:25:27 GMT
```

这可以让缓冲来决定何时取得一份关联文档的刷新版本。

（4）meta 和搜索引擎

meta 的通常用处是指定搜索引擎用来提高搜索质量的关键词。当以数个 meta 单元提供文档语言从属信息时，搜索引擎会使用 lang 属性来过滤并通过用户的语言优先参照来显示搜索结果。例如：

```
<meta name="keywords" lang="en" content="vacation,Greece,sunshine">
<meta name="keywords" lang="fr" content="vacances,Gr&egrave;ce,soleil">
```

搜索引擎的效力也可以通过使用 link 单元来指定转换工作方式获得提高。

（5）meta 和 PICS

下面这个实例说明了如何使用 meta 声明来包含一个 PICS 1.1 标签。

```
<head>
<meta http-equiv="PICS-Label" content="(PICS-1.1 "http://www.gcf.org/
  v2.5" labels on "1994.11.05T08:15-0500" until "1995.12.31T23:59-0000"
  for "http://w3.org/PICS/Overview.html" ratings (suds 0.5 density
  0 color/hue 1))">
<title>title goes here</title>
</head>
<body>the body</body>
```

2.5　小 结

本章主要介绍了 HTML 的基本概念和 head 节的 7 个单元的基本使用方法。难点在于对 meta 单元的使用方法的掌握。

下一章将介绍 HTML 中的超链接。

第 3 章　HTML 中的超链接

HTML 文档与一般的文本文件的主要区别之一，在于它不仅可以表达标题、段落和列表等普通文本文件所拥有的单元，而且还可以表达超链接（Hyperlink）。实际上，正是由于超链接才使得 HTML 成为了一种超文本、超媒体系统。使用 Web 浏览器可以通过超链接在 Web 上漫游，访问其他 Web 页面，访问音频、视频等外部多媒体信息，甚至还可以访问 FTP 和电子邮件等其他 Internet 资源。

3.1　创建超链接

HTML 最显著的优点就在于它支持文件的超链接，利用超链接，用户可以很方便地在不同文件以及同一文件的各个段落之间跳转。

HTML 中的链接包括两部分：锚标记和目标点。锚标记就是链接的源点，当鼠标被移到锚标记处时会变成小手状。此时，用户通过单击鼠标就可以到达链接的目标点。

HTML 是通过链接标记来实现超链接的。链接标记是成对标记，首标记<a>和尾标记之间的内容就是锚标记。链接标记有一个不可缺省的属性 href，用于指定链接目标点的位置。HTML 支持的超链接主要有 3 种：不同文件之间的跳转、跳转到标记位置、链接地图。

3.1.1　不同文件之间的跳转

用户可以通过单击锚标记从当前文件直接跳转至目标文件。如前所述，首标记<a>和尾标记之间的内容就是锚标记，一般是对目标文件的简要说明。此时，href 属性的参数值是目标文件的 URL。例如：

```
<html>
<head>
<title>文件链接范例程序</title>
</head>
<boby>
<p>
<hr>
</p>
<a href="apple. html">有关苹果的说明</a>
<p>
<hr>
</p>
```

下面是相应的图像锚标记：

```
<a href="apple.html"><img src="apple.gif"></a>
<hr>
</hody>
</html>
```

在 IE 中的浏览效果如图 3.1 所示。

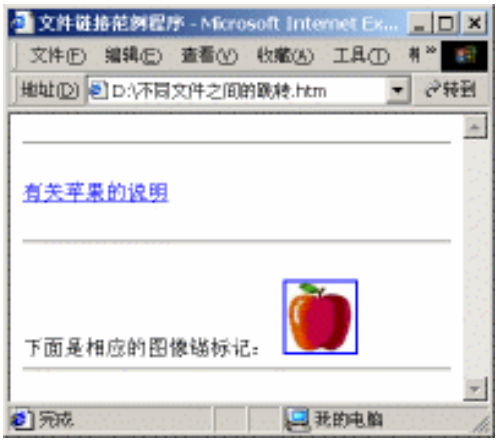

图 3.1　锚标记浏览效果图

如果用户单击文字锚标记“有关苹果的说明”或单击苹果图像，浏览器就会载入目标文件 apple.html。锚标记文字和普通文字在外观上有很大的不同，通常情况下它被显示为带下划线的亮蓝色文字。

3.1.2　跳转到标记位置

浏览器程序在载入目标文件后将自动从文件的开头部分开始显示。然而，用户需要的部分往往并不在文件的开头。如果不加以标记，用户将花费大量的时间和精力在文件（特别是一些长文件）中进行查找。为了便于浏览，可以利用链接标记的 name 属性在目标位置添加记号，然后在锚标记中进行引用。

HTML 中标识记号的格式为：

```
<a name=记号名>目标点</a>
```

引用记号的格式为：

```
<a href=URL#记号名>锚标记内容</a>。
```

例如，假定源文件（即锚标记所在文件）的文件名为 index.html，其内容为：

```
<html>
<head>
<title>超链接的源文档</title>
```

```
</head>
<body>
<p>
<hr>
欢迎来到 INDEX 主页，请选择您想浏览的部分
<br>
<p>
<a href="internet.html#ftp">文件传输</a>
<br>
<p>
<a href="internet.html#telnet＃telnet">远程登录</a>
<br>
<p>
<a href="internet.html＃email">电子邮件</a>
<br>
<p>
<a href="internet.html#bbs">电子公告权系统</a>
<br>
</body>
</html>
```

链接的目标文件与源文件在同一子目录下，文件名为 internet.html**，其内容如下：**

```
<html>
<head>
<title>超链接的目标文件</title>
</head>
<body>
Internet 提供的主要服务
<br>
Internet 提供的服务主要有 Email，FTP，BBS，Telnet 和 WWW 等 5 种。
<br>
<p>
<a name="email">电子邮件</a>
```

可以简单地理解成为通过 Internet 发送的邮件。不过这种邮件必须是存储在计算机内的文件或程序。和普通邮件一样，发送电子邮件时必须有邮件地址，它的格式为：接收用户名@目标主机名。电子邮件具有速度快，价格低廉等优点，是 Internet 上应用极为广泛的一项服务。

```
<br>
<p>
<a name="ftp">文件传输</a>
```

用于登录远程主机调用所需要的文件并将它传到自己的计算机上。FTP 的使用格式为：FTP 目标主机的 IP 地址（或域名）。当使用 FTP 命令登录目标主机后，还要输入用户名和密码进行身份验证。非注册用户可以使用通用用户名 Anonymous（匿名者）并用 Guest 或自己的 E-mail 地址作为密码。Internet 上的大多数主机都提供文件共享服务。但是给予匿名用户的权限要低于注册用户。

```
<br>
```

```
<p>
```

用户可以在电子公告板上发布信息和进行讨论。这有些类似我们平常使用的留言板。当一个用户在公告板上发布某条新闻或询问某个问题后，其他进入该公告板系统的用户就会看到这些信息并做出回应。登录 BBS 需要特定的程序，常用的如 Nettem，用户只需要在列表中输入想登录的 BBS 的 IP 地址（或域名）及相应的端口号即可。

```
<br>
<p>
```

计算机用户可以通过远程登录将本地计算机连至远程计算机上来使用远程计算机的程序和数据。Telnet 的命令格式为：Telnet 远程主机的 IP 地址（或域名）。当用户想中断与远程计算机的连接时，可以使用 quit 或 close 命令退出。

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

在 IE 中的浏览效果如图 3.2 所示。

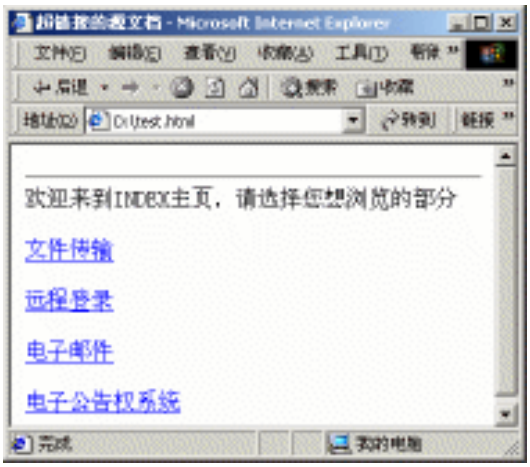

图 3.2　源文件浏览效果图

当用户单击“文件传输”超链接后，浏览器将载入目标文件 internet.html，并从标记位置所在行开始显示，而不像原来那样从文件的开头部分开始显示。单击后浏览效果如图 3.3 所示。

图 3.3　跳转到标记位置的浏览效果

3.1.3 链接地图

链接地图是一种很形象的超链接方式。它将链接源点的图像划分为若干个区域，每个区域都对应着自己的目标文件。链接地图的语法与其他两种超链接方式有很大的不同，下面来看一个实例。

```
<html>
<head>
<title>链接地图的实例</title>
<body>
<center>
<p>
<img src="map.jpg" ismap usemap="#SS">
<map name="SS">
  <area shape="rect" coords="100.76 162.123" href="畜牧场.jpg">
  <area shape="rect" coords="276.80.339.124" href="田园风光.jpg">
  <area shape="rect" coords="104.200.163.244" href="sunset.html">
  <area shape="rect" coords="262.195.351.240" href="采摘果实.jpg">
  <are> shape="circle" coords="223.169.253.169" href="ml.jpg">
</map>
</body>
</html>
```

在定义链接地图时，首先要为链接源点的图像添加 ismap 属性，将图像定义为地图。然后必须使用 usemap 属性说明该图像是遵循哪个地图定义的。usemap 属性的参数值为：

#地图名

HTML 通过<map>标记和<area>标记来定义地图。地图标记<map>有一个不可缺少的属性 name，它的参数值是地图的名字。区域的划分是通过在<map>标记中嵌套使用<area>标记来实现的。<area>标记有 3 个属性：shape、coords 和 href，分别用以规定该区域的形状、范围和对应的目标文件。

在本例中<area shape="rect" coords="100.76.162.123" href="畜牧场.jpg">语句定义了一个矩形区域，矩形的左上角坐标为(100，76)，右下角坐标为(162，123)。在该区域内任何位置单击鼠标都可以链接到目标文件——畜牧场.jpg。<area shape="circle" coords="223.169.253.169" href="ml.jpg ">语句定义了一个圆形区域，其中(223，169)为圆心坐标，(253，169)是圆上任意一点的坐标。当用户将鼠标移到定义的链接区域内时，鼠标就会变成手的形状，此时，用户单击鼠标就可以链接到相应的目标文件。

3.2 超链接与图像

<img>为图像链接标记，主要用于在网页中插入图片。其一般参数设定如下例所示：

```
<img src="logo.gif" width=100 height=100 hspace=5 vspace=5 border=2
  align="top" alt="Logo of PenPals Garden" lowsrc="pre_logo.gif">
```

其中：

- src：**指定图片来源。接受**.gif、.jpg **及**.png **格式，前两者通行已久，后者从** 96 **年开始发展，未来将取代前两者。若图片文件与该**.html **文件同处一目录则只写上文件名称，否则必须加上正确的路径，相对及绝对路径皆可。**
- width **和** heigh：**设定图片的宽度及高度，一般以像素作单位。通常只设为图片的真实大小以免失真，若要改变图片大小最好事先使用图像编辑工具对图像进行处理。**
- hspace **和** vspace：**设置图片边沿空白，以免文字或其他图片过于贴近。**hspace **用于设置图片左右的空间，**vspace **则用于设定图片上下的空间，以像素为单位。**
- borde：**指定图片边框厚度。**
- align：**调整图片旁边文字的位置。可以控制文字出现在图片的偏上方、中间、底端、左右等位置，可选值有：**top、middle、bottom、left、right，**默认值为** bottom。Navigator **还支持** texttop、baseline、absmiddle、absbottom。**其中：**
  - texttop：**表示图片沿文字顶线对齐。**
  - baseline：**表示图片沿目前文字行底线对齐。**
  - absmiddle：**表示图片沿目前文字行绝对中央对齐。**
  - absbottom：**表示图片沿目前文字行绝对底部（绝对底部意指它考虑到如** y、g、q **等字的下缘）对齐。**
- alt：**用以描述该图形的文字。若用户使用纯文本浏览器，由于不支持图片，这些文字便会代替图片而被显示。对于支持图片显示的浏览器，当鼠标移至图片上，这些文字也会同时显示出来。**
- lowsrc：**设定先显示分辨率低的图片。若所加入的是一张很大的图片，下载时间很长，图片会以低的分辨率显示。**

下面举 5 **个实例来说明图像链接的使用方法。**

实例 1：

```
<html>
<head>
<title>普通插入</title>
</head>
<body>
<img src="girl.gif" width=100 height=112 border=0 alt="beautiful girl">
普通插入
</body>
</html>
```

浏览效果如图 3.4 所示。

图 3.4 插入普通图片

实例 2：

```
<html>
<head>
<title>设定上下左右空白位置</title>
</head>
<body>
<img src="girl.gif" width=100 height=112 border=0 alt="beautiful"
  hspace=10 vspace=20">设定上下左右空白位置
</body>
</html>
```

浏览效果如图 3.5 所示。

图 3.5 设定上下左右空白位置

实例 3：

```
<html>
<head>
<title>设定字画中间对正，边框厚度为 4。</title>
</head>
<body>
<img src="girl.gif" width=100 height=112 border=4 alt="beautiful"
  hspace=10 vspace=20">设定字画中间对正，边框厚度为 4。
</body>
</html>
```

浏览效果如图 3.6 所示。

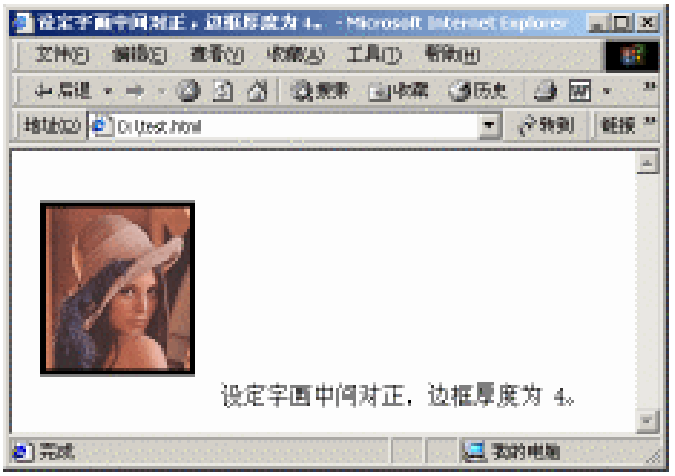

图 3.6　设置边框厚度

实例 4：

```
<html>
<head>
<title>设定图片靠右</title>
</head>
<body>
<img src="girl.gif" width=100 height=112 alt="beautiful lady"
  align="right" border=0>设定图片靠右。
</body>
</html>
```

浏览效果如图 3.7 所示。

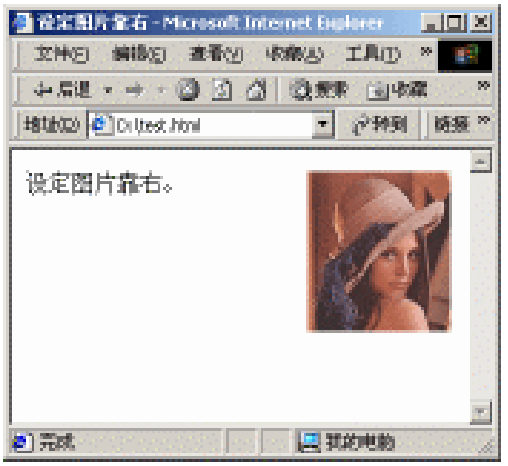

图 3.7 设定图片靠右

实例 5：

```
<html>
<head>
<title>放大了的图片</title>
</head>
<body>
<img src="girl.gif" width=200 height=220 alt="I’m not beautiful
  right now" border=0>放大了的图片
</body>
</html>
```

浏览效果如图 3.8 所示。

图 3.8 将图片放大

3.3　小　结

本章讲解了超链接和图像链接，其中图像链接是重点内容。一个图文并茂、吸引浏览者的 HTML 文件才是一个出色的文件。

下一章将介绍 HTML 的文本格式。

第 4 章　HTML 的文本格式

使用 HTML 来编写网页，无非是要让浏览者通过页面去了解一些事件或掌握一些知识，因此文本内容就是必不可少的。为了使 HTML 的编写者更好地表现和发挥，HTML 中提供了很多专门针对文本内容的标记。通过学习本章所介绍的一系列标记，一定能创作出更美观、更吸引人的网页。

4.1　换 行 标 记

HTML 允许用户对文本的输出进行换行控制。换行标记包括 和
，使用它们可以分别对文本进行分段和行分隔。

4.1.1　标记

为了有层次地表达自己的想法，写文章时通常会进行分段，HTML 文件同样如此。但是，在 HTML 文件中分行和分段并非想象中的那么容易。

在理论上，浏览者应该能看到所有的图形和字体。实际上，大多数人都使用浏览器的默认设置来浏览文件，因此，一个好的编辑器必须能在用户改变浏览器的默认设置时考虑和预见到所有可能发生的情况。这就意味着作为浏览者的计算机系统设置的一部分，浏览器软件必须能决定文本的回绕格式。源文件中的行格式符将被浏览器忽略，而根据它所显示的屏幕来重新格式化显示。此时必须使用文件的布局单元来保留所需要的布局控制。

将文本分段的正确方法是使用 标记。该标记的语法规则为：

一般形式： …
属性：None

在一个段落开头处放置一个段起始符 时，浏览器就知道分段了。段终止符 则是可选的，因为它被随之而来的下一个段起始符隐含表示出来了。然而，在文件的段尾加上终止符 可以使文件适合于并不遵循 HTML 4.0 标准的浏览器。HTML 1.0 将段标识符作为外壳标识，所有根据此标准生成的文件都必须在段起始符和终止符之间输入文本。

下面这段程序就是在编辑时使用了段标识符。

```
<html>
<head>
<title>使用段起始符</title>
</head>
<body>
<p>对爱情，几乎可以说人人都有经历都有体验。但人人的经历，人人的体验，都不尽
```

相同。即使有过几次爱情经历的人，他的每一次爱情体验，都是不一样的。我想，正是这样的不重复、不类同，才使爱情成为生活中一个永久的话题，能让人没完没了地絮说、咀嚼。
究竟为什么?!
也许，就因为爱情这份“礼物”，谁都获得过，可能都难以长久地拥有。或者说，爱情这件事，谁都有体会，但谁也说不清楚。就像大伏天吹来的一阵阵凉风，你浑身感觉到舒爽、惬意、愉悦，而仅仅是感觉。爱情，大概类似风，模糊的，不确定的，非常有限，又非常的无限。西蒙·波娃曾用风趣的语调描绘过爱情：“人们为什么会堕入情网?没有比这更复杂的了：因为这是冬天，因为这是夏天，因为劳累过度，因为闲极无聊，因为软弱，因为刚强，因为需要安全，因为喜欢冒险，因为绝望，因为希望，因为有人不爱你，因为有人爱你……”。这位著名女作家把爱情概括得这样玄妙，又论述得这样简洁、精辟。
真的，爱情对于任何人都是一个谜。
</body>
</html>

浏览效果如图 4.1 所示。可以看到浏览器在段落之间加入了一个空行间隔。

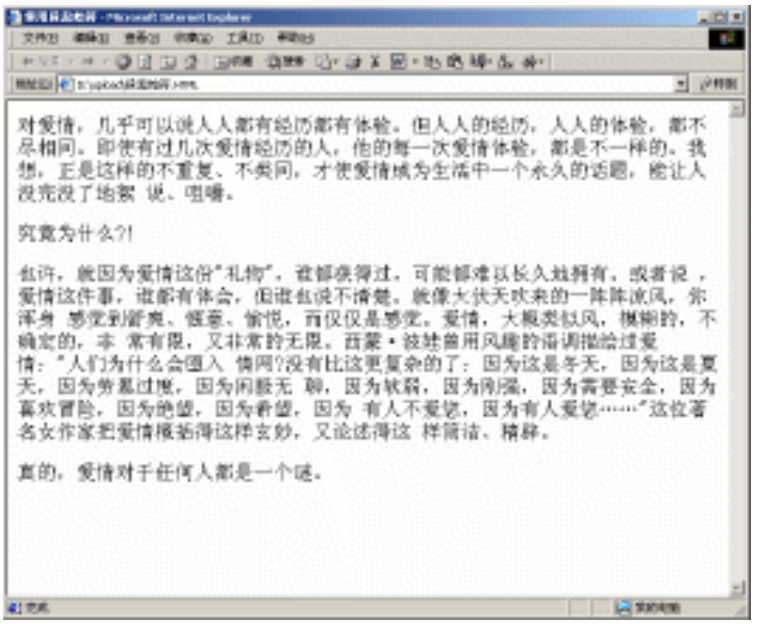

图 4.1　使用段标识符

注意：在一些 HTML 文档中，也许会看到重复使用段起始符来产生附加的空白行。其实 HTML 并不支持这种格式，大多数的浏览器都会忽略第 1 个标识符 后的所有 标识符，正如它们忽略行格式符一样。

4.1.2　
标记

HTML 文档是在浏览器中而不是在源文件中进行文本的格式化操作的，这样使它具有

设备的独立性。但是，如果想要在文本中作一个行分隔时该怎么办呢？行分隔符
实现了这个功能。该标记的语法规则为：

一般形式：
属性：clear＝left|right|all

行分隔符
可以用来终止一个文本行。这就迫使浏览器不考虑当前行的位置而开始新的一行。不同于段分隔符，行分隔符并不在文本中生成两个空格行，因为行分隔符不是一个外壳标记，它没有终止标记。

下面这段程序显示的是一首古诗，它使用了行分隔符。

```
<html>
<head>
<title>行分隔符的使用</title>
</head>
<body>
<p><h2>锄禾</h2></p>
<p>
锄禾日当午<br>
汗滴禾下土<br>
谁知盘中餐<br>
粒粒皆辛苦<br>
</p>
</body>
</html>
```

浏览效果如图 4.2 所示。

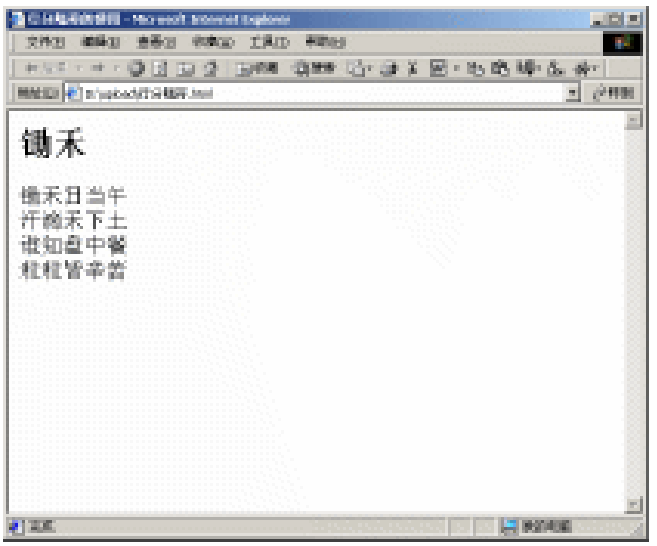

图 4.2 使用行分隔符

注意：可以使用多个行分隔符在文件中生成额外的空白符。但有些浏览器将多个行分隔符认成是一个行分隔符。

在使用行分隔符时要小心，如果浏览器中一行已经被回绕了，则行分隔符可能在下一行一两个单词之后才显示出来。在屏幕能显示的文本长度和数目多于浏览器窗口时，就有可能出现这种情况。

请看下面这段程序，其浏览效果如图 4.3 **所示。**

```
<html>
<head>
<title>行分隔符的使用</title>
</head>
<body>
也许，就因为爱情这份“礼物”，谁都获得过，可能都难以长久地拥有。或者说，爱情这件事，
谁都有体会，但谁也说不清楚。就像大伏天吹来的一阵阵<br>
凉风，你浑身感觉到舒爽、惬意、愉悦，而仅仅是感觉。爱情，大概类似风，模糊的，不确
定的，非常有限，又非常的无限。西蒙·波娃曾用风趣的语调<br>
描绘过爱情：“人们为什么会堕入情网?没有比这更复杂的了：因为这是冬天，因为这是夏天，
因为劳累过度，因为闲极无聊，因为软弱，因为刚强，因为<br>
需要安全，因为喜欢冒险，因为绝望，因为希望，因为有人不爱你，因为有人爱你……”这
位著名女作家把爱情概括得这样玄妙，又论述得这样简洁、精辟。<br>
</body>
</html>
```

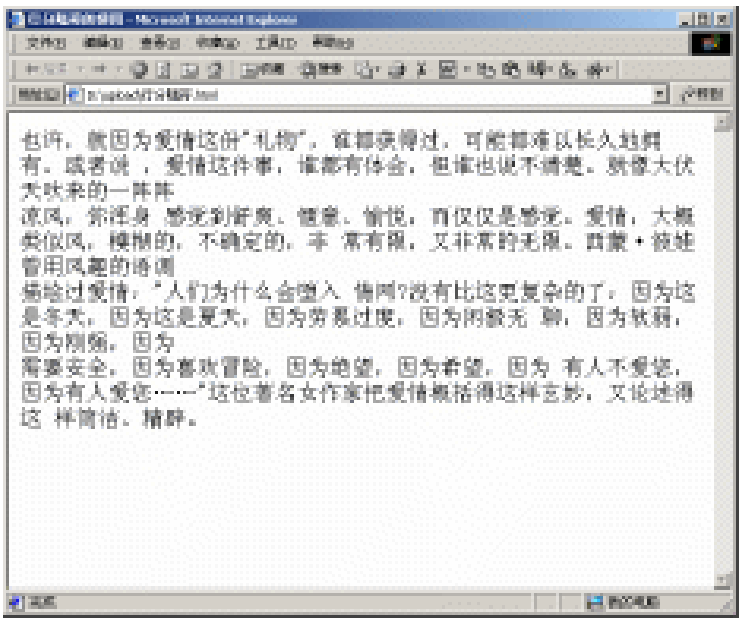

图 4.3　**任意使用行分隔符的结果**

行分隔符
的 clear **属性可以被用来定位附属于浮动图像的文本。该属性能被设置成**

left、right 或 all，它们被用来终止当前行而从左边、右边或图像的任意一边开始新的一行。

下面这段程序是一个使用 clear 属性的示例，其浏览效果如图 4.4 所示。

```
<html>
<head>
<title>换行符的 clear 属性的使用</title>
</head>
<body>
<img src="d:\upload\1.jpg" height=64 width=64 align=left>
漂亮的齿轮<br><br clear=all>
数数看<br>
共有五个齿轮。<br>
呵呵。 。 。<br>
</body>
</html>
```

图 4.4 使用
标记的 clear 属性

下面这段程序也是使用 clear 属性的示例，其浏览效果如图 4.5 所示。读者可以将它与图 4.4 进行一下比较。

```
<html>
<head>
<title>换行符的 clear 属性的使用</title>
</head>
<body>
<img src="D:\upload\1.jpg" height=64 width=64 align=left>
  漂亮的齿轮<br>
  数数看<br>
  共有五个齿轮。<br>
  呵呵。 。 。<br>
<br clear=all>
```

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

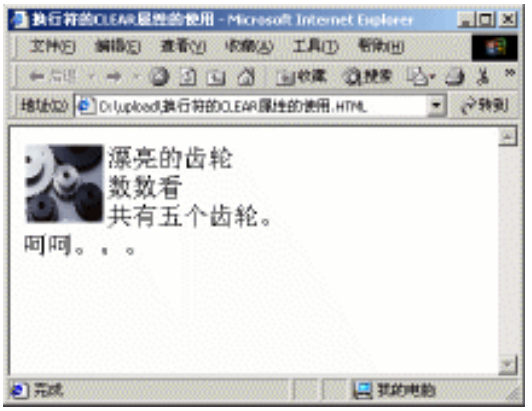

图 4.5　clear 属性的使用效果

4.2　字 符 样 式

字符样式标记是 HTML 用于改变文本中某个字符的显示形式的标记，可分为逻辑样式标记和物理样式标记。每个字符样式标记都由首标记与尾标记组成，作用于两个符号之间的文本。字符样式标记也可以嵌套使用。

下面分别对逻辑样式标记和物理样式标记进行介绍。

4.2.1　逻辑样式标记

有些作者喜欢指定文本的类型，让浏览器自己确定最后的浏览效果。这种方式将更多的自主权交给文档，作者却少了对文档的精确控制。逻辑标记用来标识文本使用的方式。它与用于段落或标题的普通标记相似，并不说明如何对文本格式化，而是说明如何在文档中使用这些文本。逻辑样式标记共有 8 种，下面对它们分别进行介绍。

1．**标记**

该标记的语法规则为：

一般形式：…

属性：None

标记指示用某种方式强调字符，这些字符将以某种突出的方式被显示。在图形界面浏览器中，通常以斜体显示。

2．**标记**

该标记的语法规则为：

一般形式：…

属性：None

标记将更强地强调所含文本，通常以黑体显示。

3．<code>标记

该标记的语法规则为：

一般形式：<code>…</code>

属性：None

<code>标记用于显示计算机代码，通常是固定宽度字体。不过要记住<和&等字符可翻译为 HTML 标记，因此要用<或&等替换。

4．<samp>标记

该标记的语法规则为：

一般形式：<samp>…</samp>

属性：None

<samp>标记用于程序实例输出，通常和<code>标记一样采用固定宽度字体。

5．<kdb>标记

该标记的语法规则为：

一般形式：<kdb>…</kdb>

属性：None

<kdb>标记用于用户的键盘输入。通常也像<code>标记一样采用固定宽度字体。IE 用斜体，Navigator 则使用当前字符样式。

注意：在没有先对<、&等特殊字符进行检查之前，不要轻易将程序代码作为 code、samp 或 kbd 块处理。

6．<var>标记

该标记的语法规则为：

一般形式：<var>…</var>

属性：None

<var>标记代表函数或过程中的变量或参量。Navigator 中显示为斜体，IE 中则显示为默认字体。

7．<dfn>标记

该标记的语法规则为：

一般形式：<dfn>…</dfn>

属性：None

<dfn>标记用于处理函数或过程中的变量或参量。Navigator 中显示为斜体，IE 中则显

示为默认字体。

8．<cite>标记

该标记的语法规则为：

一般形式：<cite>…</cite>

属性：None

<cite>标记标明它所包含的文本是引用语或参考，通常以斜体显示。

所有这些逻辑字符样式的单元都要求有首标记和尾标记，也可嵌套成复合样式，并可与物理字符样式（请参看 4.2.2 节）一起使用。

下面这段程序是一个逻辑字符样式的示例，其浏览效果如图 4.6 所示。

```
<html>
<head>
<title>逻辑样式标记示例</title>
</head>
<body>
<p><em>em 用于强调所含文本</em></p>
<p><strong>strong 用于更强地强调所含文本</strong></p>
<p><code>code 用于显示计算机代码</code><br></p>
<p><samp>samp 用于程序实例输出</samp><br></p>
<p><kbd>kbd 用于用户的键盘输入</kbd><br></p>
<p><var>var 代表函数或过程中的变量或参量</var><br></p>
<p><dfn>dfn 用于处理术语的定义</dfn><br></p>
<p><cite>cite 用于表明它所包含的文本是引用语或参量</cite><br></p>
</body>
</html>
```

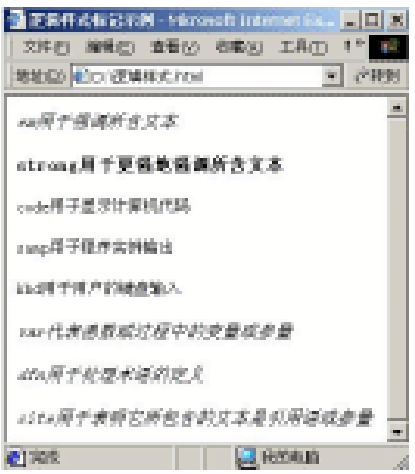

图 4.6　逻辑字符样式的浏览效果

另外，还有一种逻辑样式标记，它是作为用户自定义的字符样式的定义符。该标记的语法规则为：

一般形式：…
属性：class，id，style

标记用来支持级联式样式表。默认方式是不改变它所包含的文本。

4.2.2 物理样式标记

物理样式标记定义了字体类型或可应用于所包含文本的字符样式。它们几乎在任何地方都可使用，只有一些例外，如 title 和 submit 按钮的标签以及在一个 pre 段落内（那些用于改变字体大小的单元）除外。字符样式可以嵌套成复合样式如粗-斜体，带下划线的固定宽度字体、大字体、绿色字、粗体、斜体和含删除线的（Strike-Through）字体等等。下面分别对它们进行介绍。

1．标记
该标记的语法规则为：

一般形式：…
属性：None

标记指示浏览器为它所包含的文本使用当前字体的粗体样式。其结果可以被样式表属性所覆盖，这是由于样式表为包含物理字符样式在内的所有 HTML 标记增加了 class、id 和 style 属性。

2．标记
该标记的语法规则为：

一般形式：…
属性：None

标记指示浏览器为它所包含的文本使用斜体样式。

3．<tt>标记
该标记的语法规则为：

一般形式：<tt>…</tt>
属性：None

<u>标记指示浏览器为它所包含的文本使用一种单空（固定宽度或“电传”）字体。

4．<u>标记
该标记的语法规则为：

一般形式：<u>…</u>
属性：None

<u>标记指示浏览器为它所包含的文本加下划线。

5．<sub>标记

该标记的语法规则为：

一般形式：_…

属性：None

<sub>标记指示浏览器为它所包含的文本使用下标。

6．<sup>标记

该标记的语法规则为：

一般形式：[…]

属性：None

<sup>标记指示浏览器为它所包含的文本使用上标。

7．<big>标记

该标记的语法规则为：

一般形式：<big>…</big>

属性：None

<big>标记指示浏览器以比当前字体尺寸大 1 号的方式来显示文本，是 7 种可能尺寸中的一种，所对应的实际尺寸由浏览器决定。

8．<small>标记

该标记的语法规则为：

一般形式：<small>…</small>

属性：None

<small>标记指示浏览器以比当前字体尺寸小 1 号的方式显示文本，是 7 种可能尺寸中的一种，所对应的实际尺寸由浏览器决定。

9．<strike>标记

该标记的语法规则为：

一般形式：<strike>…</strike>

属性：None

<strike>标记指示浏览器画一条删除线穿过它所包含的文本。

10．<s>标记

该标记的语法规则为：

一般形式：<s>…</s>

属性：None

<s>标记指示浏览器画一条删除线穿过它所包含的文本。它为<strike>的简略方式。

11．<blink>标记

该标记的语法规则为：

一般形式：<blink>…</blink>

属性：None

<blink>标记仅有 Navigator 支持，将使它所包含的文本不断闪烁。许多用户觉得这有点令人烦，所以尽量（或是完全）将它应用在有较高优先级的警告及引人注意的场合。

下面这段程序是一个物理字符样式的示例，其浏览效果如图 4.7 所示。

```
<html>
<head>
<title>物理字符样式</title>
</head>
<body>
<b>粗体样式</b><br>
<i>斜体样式</i><br>
<tt>使用单空</tt><br>
<u>使用下划线</u><br>
x<sub>0</sub>=x<sub>1</sub>+x<sub>2</sub><br>
y<sup>0</sup>=y<sup>1</sup>+y<sup>2</sup><br>
<small>字体尺寸小 1 号</small><br>
<big>字体尺寸大 1 号</big><br>
<strike>穿过文本的水平线</strike><br>
<b><i>粗斜体样式</i></b><br>
<big><tt>大 1 号单空</tt></big><br>
<small><i>小 1 号斜体</i></small><br>
</body>
</html>
```

图 4.7 物理字符样式的浏览效果

4.3　预格式化文本

通常情况下，HTML 文档中的文本是基于 HTML 标记进行格式化的，文本中的任何空白字符，如空格、制表符、换行等，不论数目多少，都视为一个空格来显示。然而有时候，需要浏览器原封不动地显示预先编排好格式的一段文本，如计算机的源程序、ASCII 字符图形等，这时就可以使用<pre>标记。<pre>标记用来包含一段预先排好格式的文本，Web 浏览器将把其首标记<pre>与尾标记</pre>之间的文本按照原代码中的格式原封不动地再现出来。

<pre>标记的语法规则如下：

一般形式<pre>…</pre>
属性：None

浏览器使用等宽字体（或称为等间距字体）来显示预格式化文本。此外，与一般段落有所不同的是，对于预格式化文本来说，自动换行是被禁止的，以便保证预先编排好的格式不被破坏。

在<pre>和</pre>中不允许含有改变字体大小的标记，如<small>、<font>等，也不允许含有图像。水平制表符（简称制表符）是一个 ASCII 字符，其 ASCII 代码为十进制数 9，它使其后的一个字符移到下一个水平制表位上显示（默认时一行中每隔 8 个字符为一个水平制表位）。<pre>和</pre>中可以使用水平制表符，但是 Web 浏览器中水平制表位的设置一般是可以改变的，不同的用户往往有不同的偏好，这样可能会使其中的水平制表符不能按预期的位置显示，而导致预排格式被破坏，因此建议最好不要这样做。在<pre>和</pre>中应该使用相应数目的空格来代表水平制表符，以保证其中的文本总可以按照原样再现。

注意：尾标记</pre>是必需的。在 HTML 3.2 中<pre>标记具有 width 属性，但在 HTML 4.0 中已被取消。

下面的实例用<pre>标记写了一首小诗，如图 4.8 所示显示了该程序的页面浏览效果。如图 4.9 所示显示了不使用<pre>标记时的效果，读者可将两图进行比较。

```
<html>
<head>
<title>示例－预格式化文本标记</title>
</head>
<body>
<pre>
在年轻的时候
如果你爱上了一个人
请你　请你一定要温柔地对待他

不管你们相爱的时间有多长或多短
```

若你们能始终温柔地相待
那么　所有的时刻都将是一种无瑕的美丽

若不得不分离
也要好好地说声再见
也要在心里存着感谢
感谢他给了你一份记忆
</pre>
</body>
</html>

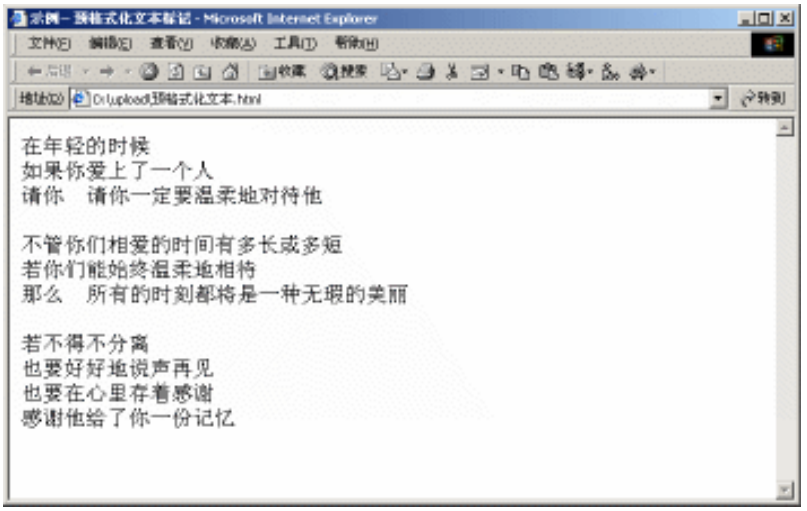

图 4.8　预格式化文本的使用效果

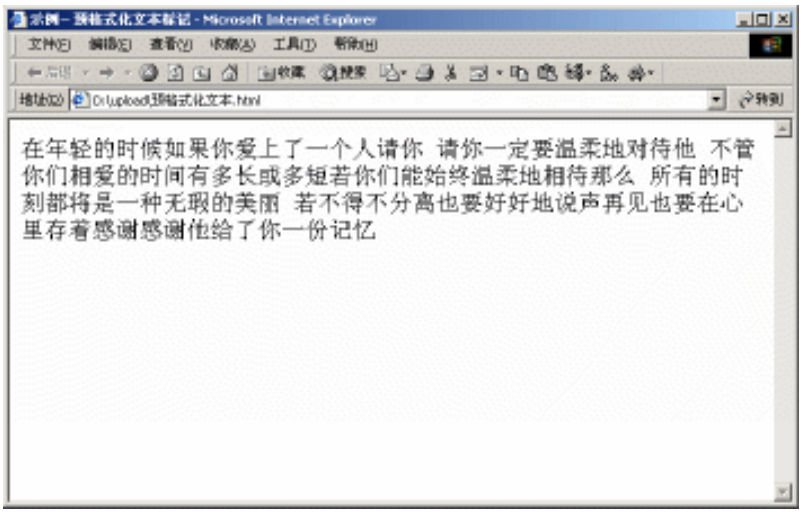

图 4.9　不使用预格式化文本的使用效果

注意：段落标记<p>不允许出现在 pre 单元内部，如果出现的话，浏览器一般会将其看作两个换行。

因为<pre>和</pre>内部允许有其他 HTML 标记出现，所以一般不能不加改变地将一段 ASCII 文本添加上<pre>和</pre>后直接插入到 HTML 文本之中。如果这段文本中存在>、<和&等字符，必须事先将它转换成为相应的字符实体。例如：

```
<html>
<head>
<title>示例－预格式化文本标记</title>
</head>
<body>
<pre>
float x;
if(x&gt;0) return 1;
else return 0;
</pre>
</body>
</html>
```

注意：其中的字符实体“ > ”代替了>字符。

本例浏览效果如图 4.10 所示。

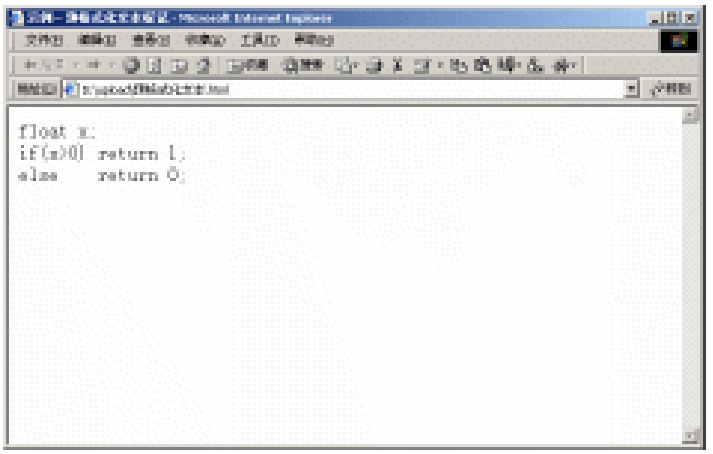

图 4.10　预格式化文本的浏览效果

4.4　水平尺寸标记

为了便于 HTML 文本的阅读，在 HTML 中制定了水平尺寸标记（又称水平标尺线）将 HTML 文本在逻辑上分隔成不同主题的分区。<hr>标记用来在 Web 页面上画出一条水平标尺线。hr 是 Horizontal Rule（水平标尺线）的缩写。<hr>标记的语法规则如下：

一般形式：<hr>
属性：size=*n*，**如：**<hr size=10>；
width=*n*|*p*%，**如：**<hr width=50>，<hr width=50%>；
align=left|center|right，**如：**<hr align=left>；
color=**十六进制** RGB **数码**|black|olive|teal|red|blue|maroon|navy|gray|lime|fuchsia|white|green|purple|silver|yellow|aqua；
noshade，**如：**<hr noshade>

<hr>**标记只有首标记**<hr>，**没有尾标记。与** HTML 3.2 **相比，在** HTML 4.0 **中，**<hr>**标记增加了一个** color **属性。**<hr>**标记的各项属性说明见表** 4.1。

表 4.1 <hr>**标记的各项属性说明**

属性	**说明**
size	**用来设置水平标尺线的粗细，以像素数为单位。该属性只能取整数值，默认值为** 2
width	**用来设置水平标尺线在水平方向上的宽度，即水平标尺线的长度。其取值可以用像素表示的绝对宽度，也可以使用以百分数表示的相对宽度**
align	**用来设置水平标尺线的对齐方式，也就是从何处开始画水平标尺线。**left **为从左画起，**center **为居中，**right **为从右画起，默认时标尺线居中。该属性只有在设置了比浏览器窗口宽度小的** width **属性时才有实际意义**
color	**用来设置水平标尺线的颜色。**HTML **既允许用户使用十六进制** RGB **数码，也允许使用英文预定义色彩**
noshade	**一个名称记号，用于取消水平标尺线的阴影，即只画出一条简单的实线，没有三维效果。默认情况下，水平标尺线是带有阴影的直线，具有三维立体效果，然而在定制背景图案和背景颜色的** Web **页面上使用带有阴影的水平标尺线有时效果并不理想，因此该属性有时还是十分有用的**

注意：在使用 width **属性时，建议不要使用绝对宽度，因为事先无法知道用户在浏览** Web **页面时浏览器窗口的宽度，所以应该尽量使用相对宽度。**

下面的实例给出了一些不同风格的水平标尺线，其浏览效果如图 4.11 **所示。**

```
<html>
<head>
<title>示例－标尺线</title>
</head>
<body>
<p>默认的标尺线
<hr>
<p>宽度 50%，左对齐的标尺线
<hr align=left width=50%>
<p>宽度 150 个像素，居中对齐的标尺线
<hr align=centre width＝150>
```

```
<p>宽度 75%，右对齐的标尺线
<hr align=right width=75%>
<p>粗细 5 个像素，宽度 50%，左对齐的标尺线
<hr align=left size=5 width=50%>
<p>粗细 5 个像素，宽度 50%，右对齐，无三维效果的标尺线
<hr align=right size=5 width=50% noshade>
</body>
</html>
```

图 4.11　水平标尺线的使用

使用水平标尺线的目的是为了让 HTML 文本更便于浏览，但是如果水平标尺线使用得过多，以致于在标尺线之间只有很少的文本时，页面浏览效果其实并不美观，这一点是需要注意的。不采用<hr>标记，而采用与标尺线的形状相似的图像代替水平线，有时也同样可以创建出美观的 Web 页面。关于如何在 Web 页面中插入图像，请参见本书第 3 章。

4.5　地 址 标 记

地址标记<address>用于 Web 页面上类似于签名的实体，一般用来提供 HTML 文本作者本人的地址，以便于阅读文本者能与作者取得联系，提供反馈或者了解更加详细的信息。<address>标记提供的一般是 Email 地址，当然也可以提供一般地址、联系人、日期、版权信息以及其他警告信息等。<address>标记的语法规则为：

一般形式：<address>…</address>
属性：None

<address>标记的首标记<address>和尾标记</address>都是必需的，此外，<address>和</address>中的内容只能是段落、普通文本和文本级单元。浏览器在显示<address>和

</address>中的内容时，通常在其前后各加上一个换行，显示时的字体一般为斜体，沿左边界对齐，并且相对左边界稍有缩进。

**注意：如果地址信息超过一行，那么一定要在除最后一行以外的每一行的末尾加上
标记，以确保地址信息的正确显示。**

下面是地址标记的应用实例，其浏览效果如图 4.12 所示。

```
<html>
<head>
<title>示例－地址标记</title>
</head>
<body>
欢迎来到 263 在线
<hr>
263 在线__中国人的网上家园<br>
Tel:010-85299696__Fax:010-65615263<br>
倾听热线:263online@263.net.cn <br>
HTTP://WWW.263.net/<br>
263 逍遥行 Tel:010-64262631   <br>
263 数据港 Tel:010-64255100   <br>
倾听热线:263@263.net.cn <br>
c)2000-2005 版权所有 京 ICP 证 000009 号<br>
</address>
</body>
</html>
```

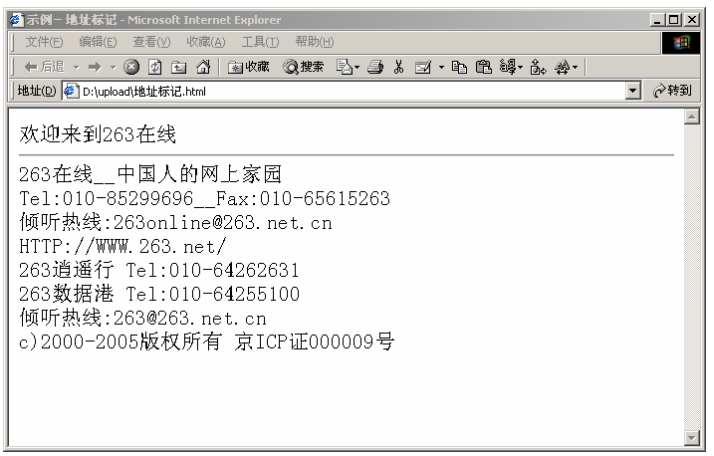

图 4.12 使用地址标记

4.6　文本对齐方式

文本对齐是安排文本块（如标题或段落）的一种能力，其目的是调整文本相对于页边左对齐（为默认值）、右对齐或中心对齐。文本对齐方式从 HTML 3.2 开始提供。HTML 3.2 及其更高版本提供了文本和单元对齐方式的属性，这些属性已经不同程度地结合到各种浏览器之中。

4.6.1　单个单元对齐

为对齐一个单个标题或段落，在 HTML 单元中，可使用 align 属性，它的取值为 left、right 或 center。left 表示单元为左对齐，right 表示单元为右对齐，center 表示单元为中对齐。

注意：align 并非一个标记，它只是一个属性。

下面这段程序是一个单个单元对齐的示例，其浏览效果如图 4.13 所示。

```
<html>
<head>
<title>单个单元对齐</title>
</head>
<body>
<h1 align=center>单个单元对齐示例</h1>
<p align=center>请比较效果</p>
<h3 align=left><a href="a.htm">左对齐</h3>
<h3 align=center><a href="b.htm">中心对齐</h3>
<h3 align=right><a href="c.htm">右对齐</h3>
</body>
</html>
```

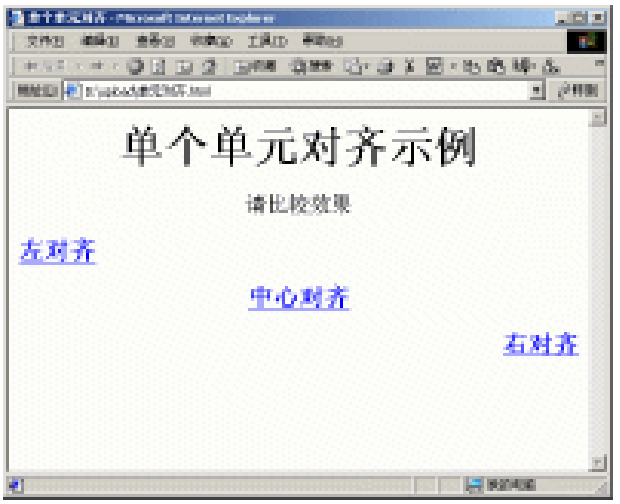

图 4.13　单个单元对齐的浏览效果

4.6.2 单元块对齐

单元块对齐使用 标记，其语法规则为：

一般形式： …
属性：align=left|center|right

与单个单元的对齐方式不一样， 标记用于围住任何种类的 HTML 标记的一块，它影响在 和 标记内的所有标记和文本。与 align 属性相比， 标记有以下两个优点：

- 标记仅需使用一次，而无需在几个不同的标记中重复地包括 align 属性。
- 标记可用于任何对齐，如标题、段落、引号、图像和表格等；而 align 属性只能用于有限个标记。

对于对齐 HTML 代码块，以 和 标记围住那个代码块，然后，在首标记 中包含 align 属性。正如在其他标记中一样，align 可以有 left、right 和 center 等值。

下面这段程序是 标记的使用示例，它的浏览效果如图 4.14 所示。

```
<html>
<head>
<title>单元块对齐</title>
</head>
<body>
<div align=center>
<h1>单元块对齐示例</h1>
<p>请比较效果</p>
</div>
<div align=left>
<h3><a href="i.htm">我是谁？</h3>
<h3><a href="he.htm">他是谁？</h3>
</div>
<div align=right>
<h3><a href="we.htm">我们是谁？</h3>
<h3><a href="they.htm">他们是谁？</h3>
</div>
</body>
</html>
```

比较图 4.13 和图 4.14 可以发现，对于所有在 和 标记之间的内容，HTML 将根据 align 属性值来对齐。如果单个 align 属性出现在 内的标题和段落中，这些值将使全局 设置无效。也就是说，单个出现的 align 属性比 中的 align 属性优先级高。

注意： 标记本身不是一个段落类型，在 和 标记内，仍需要正规的单元标记，如<p>，<h1>，<ul>和<blockquote>等。

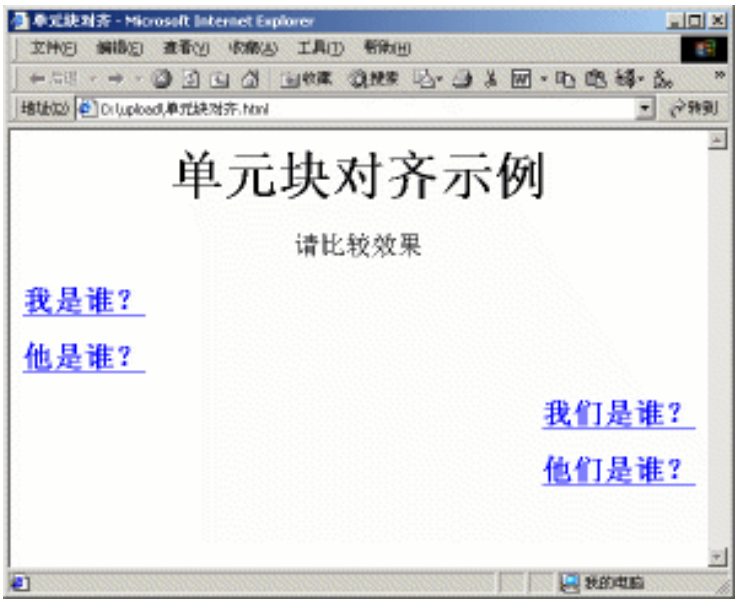

图 4.14　单元块对齐示例

4.7　居中对齐分区

标记是 HTML 最早的 Navigator 扩展之一，它用来指示 Web 浏览器将一大块文本居中显示。 标记比 标记获得的支持还要广泛。

标记的语法规则如下：

一般形式： …

属性：None

… 与 … 是完全等价的。center 是字块级单元而不是文本级单元，使用 标记将导致浏览器显示一个新的段落，所以它不能用在一个段落或其他字块级单元内部以试图将其中的某些文件居中显示。

下面这段程序同时使用了 标记和 标记，其浏览效果如图 4.15 所示。读者可对两者进行相互比较。

```
<html>
<head>
<title>居中对齐分区</title>
</head>
<body>
<div align=left>
<h2>静夜思</h2>
<h4>李白</h4>
```

```
床前明月光<br>
疑是地上霜<br>
举头望明月<br>
低头思故乡
</div>
<hr>
<center>
<h2>静夜思</h2>
<h4>李白</h4>
床前明月光<br>
疑是地上霜<br>
举头望明月<br>
低头思故乡
</center>
</body>
</html>
```

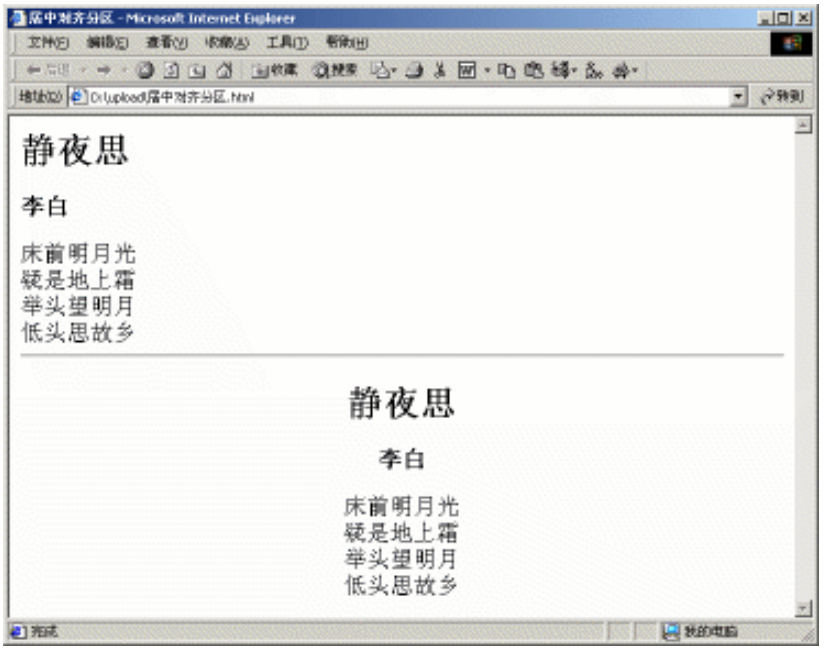

图 4.15 居中对齐分区

4.8 字体大小和颜色

HTML **中有两个用于设置字体的标记：**<basefont>**和****。**<basefont>**标记用于设置默认字体大小，而****标记将参考由**<basefont>**标记设置的默认字体大小值来设置相对字体大小。**

<basefont>标记的语法规则为：

一般形式：<basefont>
属性：size=*n*

标记的语法规则为：

一般形式：…
属性：size=*n*；
color=#RRGGBB；
face=***字体名***

注意：basefont 单元没有尾标记，并且内容为空。

<basefont>标记只有一个 size 属性，其取值为一个整数，范围是 1~7。默认的字体大小一般只作用于普通文本和预格式化文本，不会影响标题的字体大小。

font 单元用来改变文本的字体大小，除此之外，font 单元还可以改变字体的颜色。font 单元的首标记和尾标记都是必需的，font 单元所改变的是出现在首标记和尾标记之间的文本（即 font 单元的内容）的字体大小和颜色。

<font>标记的 size 属性用于设置字体的大小。该属性有两种用法，一种是设置绝对字体大小，其取值为从 1~7 的整数。

下面这段程序给出了用 size 设置绝对字体大小的示例，其浏览效果如图 4.16 所示。

```
<html>
<head>
<title>绝对字体大小</title>
</head>
<body>
<p><h1 align=center>绝对字体大小的示例</h1></p>
<br>
<p><font size=7>今天天气真好！</font></p>
<p><font size=6>今天天气真好！</font></p>
<p><font size=5>今天天气真好！</font></p>
<p><font size=4>今天天气真好！</font></p>
<p><font size=3>今天天气真好！</font></p>
<p><font size=2>今天天气真好！</font></p>
<p><font size=1>今天天气真好！</font></p>
</body>
</html>
```

size 属性的另一种用法是用一个带正负号的整数来设置一个相对字体大小，相对字体大小基于<basefont>标记所设置的默认字体大小。

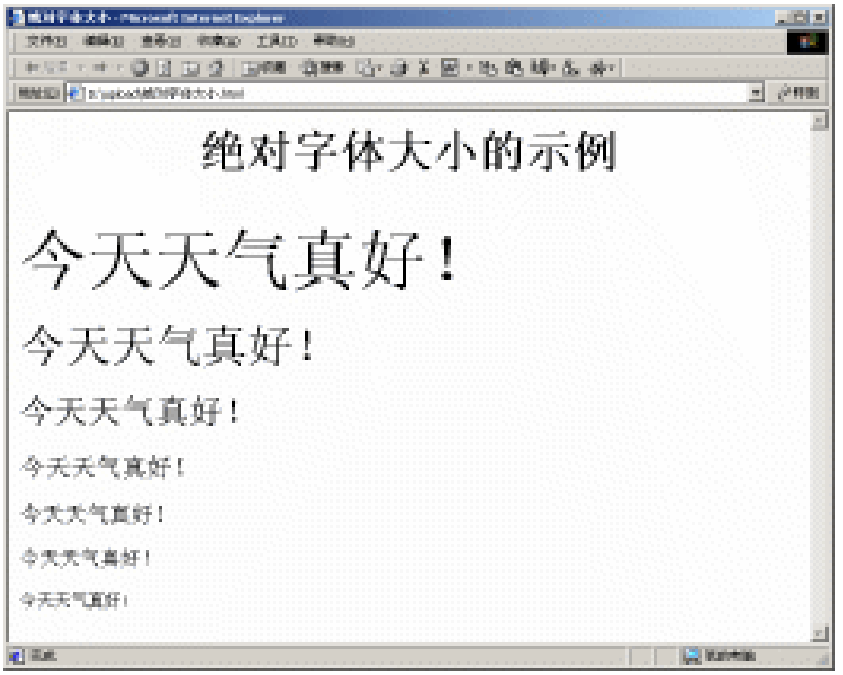

图 4.16 **设置绝对字体大小**

下面这段程序给出了相对字体大小的使用方法，其浏览效果如图 4.17 **所示。**

```
<html>
<head>
<title>相对字体大小</title>
</head>
<body>
<p><h1 align=center>相对字体大小的示例</h1></p>
<br>
<p><basefont size=3>basefont size=3<br>
<font size=-3>今天天气真好！</font><br>
<font size=-2>今天天气真好！</font><br>
<font size=-1>今天天气真好！</font><br>
<font size=+1>今天天气真好！</font><br>
<font size=+2>今天天气真好！</font><br>
<font size=+3>今天天气真好！</font></p>
<hr>
<p><basefont size=6>basefont size=6<br>
<font size=-3>今天天气真好！</font><br>
<font size=-2>今天天气真好！</font><br>
<font size=-1>今天天气真好！</font><br>
<font size=+1>今天天气真好！</font><br>
<font size=+2>今天天气真好！</font>
<font size=+3>今天天气真好！</font></p>
```

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

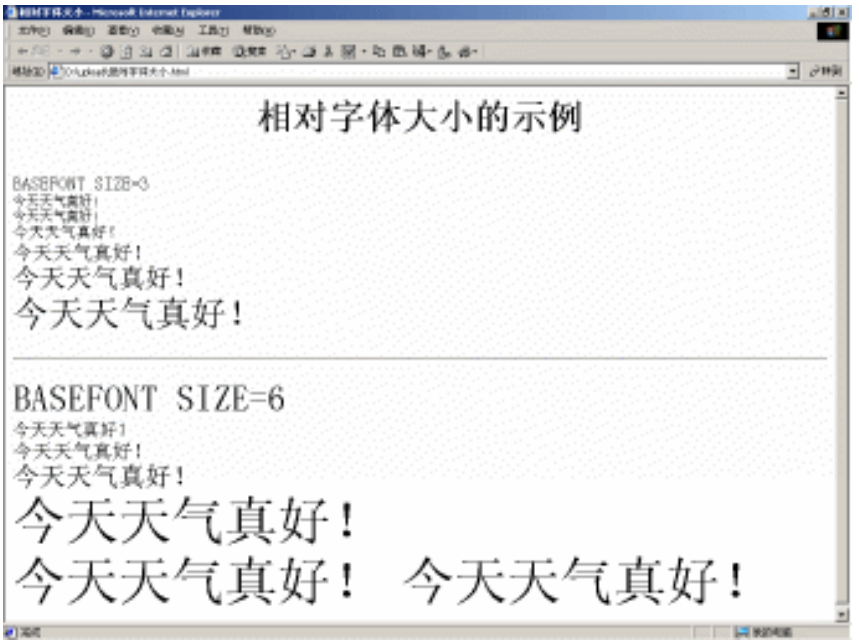

图 4.17　设置相对字体大小

从图 4.16 和图 4.17 可以看出，实际能显示的字体大小只有 7 个等级。如果由 basefont 单元设置的默认字体大小和由 font 单元设置的相对字体大小相加得到的结果小于 1，则按照绝对字体大小为 1 显示文本；若结果大于 7，则按照绝对字体大小为 7 显示文本。

编制 HTML 文本时，使用<font>标记应该只是为了增强某种浏览效果，而不要用它去模仿 HTML 文档的各级标题。索引器一类的程序可以利用 HTML 文档的 6 级标题生成文档的大纲，但是<font>标记则不能为索引器提供相应的信息。

<font>标记的 color 属性用来设置文本的颜色。#RRGGBB 是以十六进制形式表示的颜色代码。颜色表示为红-绿-蓝形式，其中的每一成分都是一个 00 到 FF 之间的十六进制数，以两位数字表示。颜色代码的第 1 个字符是#。例如，#FF0000 为红色，#00FF00 为绿色，#0000FF 为蓝色。

<font>标记的另外一个属性是 face，该属性可用来设置字型。face 属性的取值是一个按顺序排列的字型名称列表，各字型名称之间以逗号分隔。Web 浏览器将按照 face 属性所指定的次序搜索系统中已安装的字型库，并以最先找到的字型显示文本。

下面这段程序给出了几种常用的字型，其浏览效果如图 4.18 所示。

```
<html>
<head>
<title>字体浏览效果</title>
</head>
```

```
<body>
<p><font face=宋体>宋体</font></p>
<p><font face=隶书>隶书</font></p>
<p><font face=楷体>楷体</font></p>
<p><font face=仿宋>仿宋</font></p>
<p><font face=方正舒体>方正舒体</font></p>
<p><font face=华文新魏>华文新魏</font></p>
</body>
</html>
```

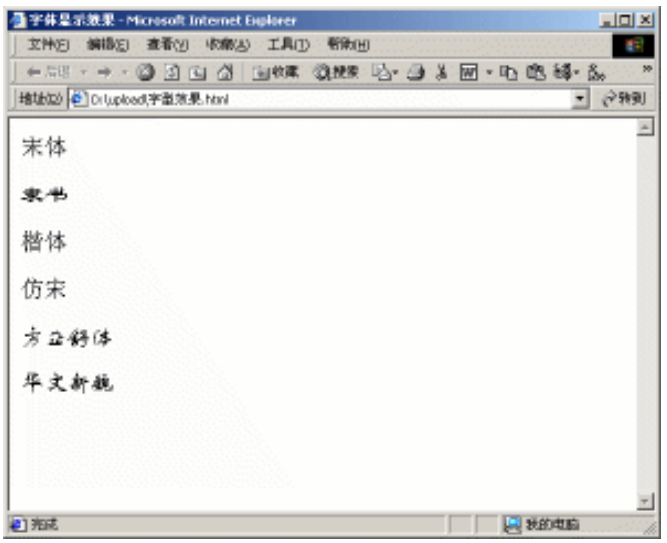

图 4.18 字型效果

4.9 列表格式标记

HTML 4.0 **允许创建无序列表（或称强调式列表、有序列表、数字式列表），以及含有左缩进部分但不含数字和强调符的定义式列表。列表可以是嵌套的，也可以在表格单元中显示。**

4.9.1 无序列表

标记用于创建无序列表，其语法规则如下：

一般形式：
…
属性：type=disc|circle|square**；**
compact

而<li>**标记（列表项标记）制定了列表内的单个入口，可以包含其他的块级单元（包括表格和其他列表），标题及地址单元除外。**

下面这段程序显示了一个简单的无序列表，其浏览效果如图 4.19 **所示。**

```
<html>
<head>
<title>一个简单的无序列表</title>
</head>
<body>
一个简单的无序列表示例：
<ul>
  <li>列表单元一
  <li>列表单元二
  <li>列表单元三
</ul>
</body>
</html>
```

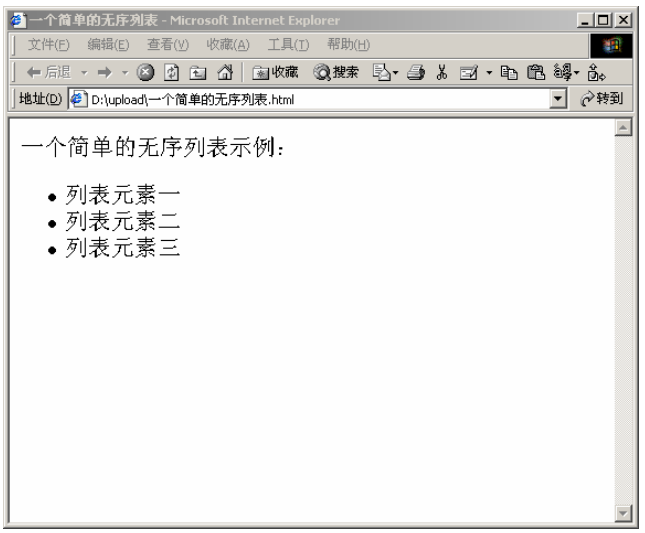

图 4.19　一个简单的无序列表

标记有两个可选属性：type **和** compact**。**

type **属性指定使用的强调符类型。**type **可取的合法值有** disc**、**circle **和** square**。**disc **指示浏览器强调符为实圆形，它通常是非嵌套列表的默认值；**circle **指示浏览器强调符为空圆形，它通常是嵌套列表的默认值；**square **指示浏览器强调符为实或空的方形，具体取决于浏览器。**

compact **属性用于表明列表应比通常的占用更少的空间，但并没有受到广泛支持。**

下面这段程序给出了一个复杂点的无序列表示例，其浏览效果如图 4.20 **所示。**

```
<html>
<head>
<title>嵌套的无序列表</title>
</head>
<body>
```

```
<h2>无序列表</h2>
<ul type=disc>
  <li>实圆形一
    <ul type=circle>
      <li>空圆形一
        <ul type=square>
          <li>方形一
          <li>方形二
          <li>方形三
        </ul>
      <li>空圆形二
    </ul>
  <li>实圆形二
    <ul type=circle>
      <li>空圆形三
        <ul type=square>
          <li>方形一
          <li>方形二
          <li>方形三
        </ul>
       <li>空圆形四
        </ul>
        </ul>
 </body>
 </html>
```

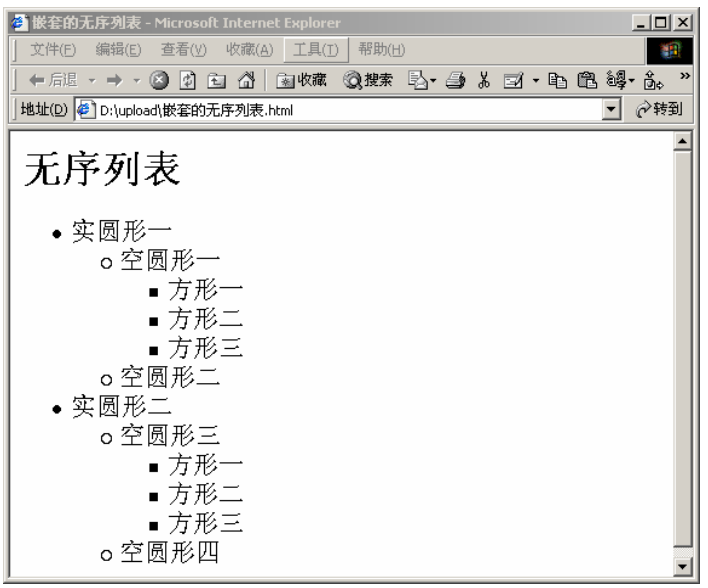

图 4.20 嵌套的无序列表

注意：嵌套式列表是缩排的。

位于 ul **单元内的** li **单元的语法规则如下：**

一般形式：
…
属性：type=disc|circle|square；
compact;
type

li **单元入口提供了不同的属性，这取决于它们位于哪种类型的列表内。无序列表仅支持列表单元中的** type **属性。严格地讲，**li **单元是一个容器单元，但多数情况下尾标记都被忽略。**

type **属性指定了特定列表单元的强调符类型，与** ul **单元本身的** type **属性允许值（**disc、circle **和** square**）完全相同。**

4.9.2　有序列表

标记用于创建有序列表，其语法规则如下：

一般形式：
…
属性：type=1|a|a|i|i|；start；compact

（列表项）标记指定列表中单个入口。
下面这段程序给出了一个简单的有序列表示例，它的浏览效果如图 4.21 **所示。**

图 4.21　**一个简单的有序列表**

```
<html>
<head>
<title>一个简单的有序列表</title>
</head>
<body>
一个简单的有序列表
<ol>
  <li>列表单元一
  <li>列表单元二
  <li>列表单元三
</ol>
</body>
</html>
```

ol 单元有 3 个可选属性：type、start 和 compact。

type 属性指定使用的数字序列样式。type 的合法值可取 1（阿拉伯数字，默认值为 1、2、3，等等）、a（大写字母，默认值为 a，b，c，等等）、a（小写字母，默认值为 a，b，c，等等）、Ⅰ（大写罗马数字，默认值为Ⅰ、Ⅱ、Ⅲ、Ⅳ，等等）、i（小写罗马字母，默认值为 i，ii，iii，iv，等等）。

start 是一个用来指明数序序列起始值的数字。可以使用 type 中定义的任意值。没有在每个数字前显示一个前缀的选项。

<ol compact>用以表明生成的列表应比通常的列表紧凑（例如，在比较小的空间中），但它并非受到广泛支持。

下面这段程序给出了一个数序序列不从 1 开始的有序列表的示例，其浏览效果如图 4.22 所示。读者可将它与图 4.21 作对比。

```
<html>
<head>
<title>一个简单的有序列表</title>
</head>
<body>
一个简单的有序列表
<ol start=2>
  <li>列表单元一
  <li>列表单元二
  <li>列表单元三
</ol>
</body>
</html>
```

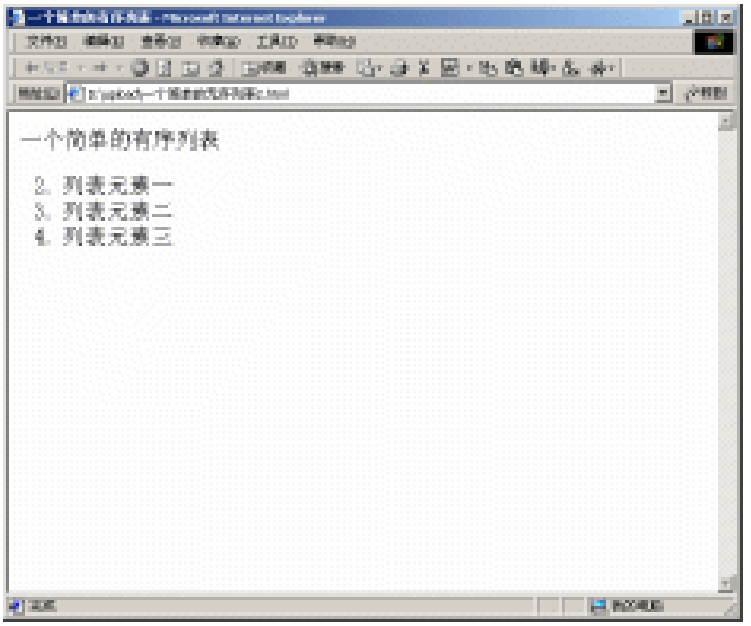

图 4.22　数序起始值不为 1 的有序列表

有序列表还可以进行嵌套，下面这段程序给出了有序列表的一个复杂点的示例。其浏览效果如图 4.23 所示。

```
<html>
<head>
<title>嵌套的有序列表</title>
</head>
<body>
<h2>有序列表</h2>
<ol type=I>
  <li>大写罗马数字
  <li>大写字母
    <ol type=A>
      <li>阿拉伯数字一
        <ol type=1>
          <li>小写字母一
            <ol type=a>
              <li>小写罗马数字一
                <ol type=i>
                  <li>type
                  <li>start
                  <li>compact
                </ol>
                  <li>小写罗马数字二
                </ol>
```

```
                <li>小写字母二
                <li>小写字母三
              </ol>
            <li>阿拉伯数字二
          </ol>
        <li>小写字母
          <ol type=a>
            <li>type
            <li>start
            <li>compact
          </ol>
        <li>小写罗马字母
      <ol type=i>
        <li>type
        <li>start
        <li>compact
      </ol>
        <li>阿拉伯数字
          <ol type=1>
            <li>type
            <li>start
            <li>compact
            </ol>
</ol>
</body>
</html>
```

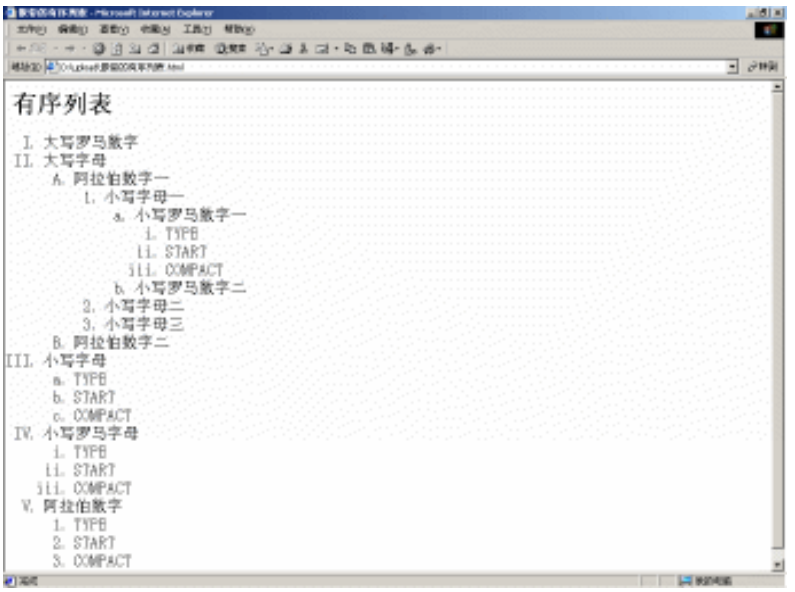

图 4.23 嵌套的有序列表

位于 ol 单元内的 li 单元的语法规则如下：

一般形式：
…
属性：value，type

li 单元支持不同的属性，这取决于它们位于哪类列表中。数字式列表支持列表项的 value 和 type 属性。严格地讲，li 单元是一种容器单元，但多数情况下尾标记都被忽略。

value 属性用来为列表的某个入口指定一个确切的值，对 ol 单元而言可用来代替 start 属性，以取得非连续性的数字序列。

列表项也可有一个 type 属性，与在 ol 单元中的潜在值相同，平常很少使用，除非一个列表内要进行数字样式切换。

4.9.3　定义列表

用于定义列表的标记有<dl>、<dt>和<dd>。
<dl>标记的语法规则为：

一般形式：<dl>…</dl>
属性：compact

<dt>标记的语法规则为：

一般形式：<dt>…</dt>
属性：None

<dd>标记的语法规则为：

一般形式：<dd>…</dd>
属性：None

dl 单元创建含缩排格式和非缩排格式项的列表，各项不含数字和强调符。标准的使用方法是，对于定义式表项（dt），则含有正常的空白边区；对于定义式描述（dd），则含有缩进的左空白边区。dl 单元有一个可选的 compact 属性，但很少使用并且未被广泛支持。严格地讲，dt 单元和 dd 单元均是容器单元，但通常尾标记都被忽略。

下面这段程序给出了一个简单的定义列表的示例，其浏览效果如图 4.24 所示。

```
<html>
<head>
<title>定义列表</title>
</head>
<body>
<h2>一个简单的定义列表</h2>
<dl>
<dt>First
<dd>Yesterday is Monday
```

```
<dt>Second
<dd>Today is Tuesday.
<dt>Third
<dd>Tomorrow is Wednesday.
</dl>
</body>
</html>
```

图 4.24 一个简单的定义列表

dd **单元允许包含除标题和地址以外的块级单元。一个** dd **单元可以出现在一个无相应的** dt **的** dl **列表中，并且是惟一一个能正常产生左缩进空白边区和标准右空白边区的块级单元，从而，含** dd **单元的** dl **列表有时用于创建缩进的段落。然而，并不是必须要用这种方式让浏览器产生缩进的定义式列表，况且创建者通常无法控制用哪种浏览器访问他们的文档。因此对于非** Intranet **应用而言，这样做并不是绝对安全的。不过，这还是要比通常（但却是误导的）使用一个“不可见的”透明** GIF **作为间隔要安全得多，因为面向文本的浏览器（例如** Lynx**）以及不能装载图像的浏览器还相当普遍。样式表是解决此类问题的理想方案，但在缺少它时，带有一个空的固定宽度的首列，且左对齐的无边线表格可能是最佳尝试，当然这要比** dl **的装载明显慢得多。**

4.9.4 菜单项列表和目录列表

用于创建菜单项列表的标记为<menu>**。这种列表类型** HTML **和** Web **浏览器都支持，这里的区别主要在于** HTML **标记。大多数浏览器对**<menu>**的默认显示同无序列表的字体和样式非常类似。如果根据** Web **浏览器的特点为菜单段落选择一个不同的屏幕格式，该标记将更有价值。**<menu>**标记在** HTML **以及其客户软件的将来版本中作用可能更大，它使得浏览器和其他应用程序能够在文档中区分菜单选择。**

<menu>**标记的语法规则为：**

一般形式：<menu>…</menu>
属性：None

同前面讲述的列表类似，菜单列表提供行的起始和结束中断，并且在菜单内部能够包

括其他 HTML 单元。在这种列表中，锚单元是最有可能使用的 HTML 单元，通常用它来链接菜单列表以及其他文档资源或 Internet 应用程序。

注意：不要仅仅只因为 HTML 给这些类型取不同的名字就限制对它们的使用，请试着去使用每一种列表形式，看哪一种效果最好，然后再决定如何使用。

目前，大多数 Web 浏览器对<menu>的实现与无序列表之间并没有外观上的差异。Navigator 对菜单列表和无序列表的显示是一样的，但 IE 除了对后者省略标记之外其他都一样。

注意：所有类型的列表都能包含超文本链接，这同除简单文本之外的任何 HTML 单元一样。而且，菜单项可以包括指向其他文档或 Internet 资源的超文本链接。

下面这段程序演示了<menu>标记的典型用法，其浏览效果如图 4.25 所示。

```
<html>
<head>
<title>一个典型的菜单列表使用</title>
</head>
<body>
一个典型的菜单列表使用
<menu>
  <li><a href="my.htm">My photo<a>
  <li><a href="his.htm">His photo<a>
  <li><a href="her.htm">Her photo<a>
</menu>
</body>
</html>
```

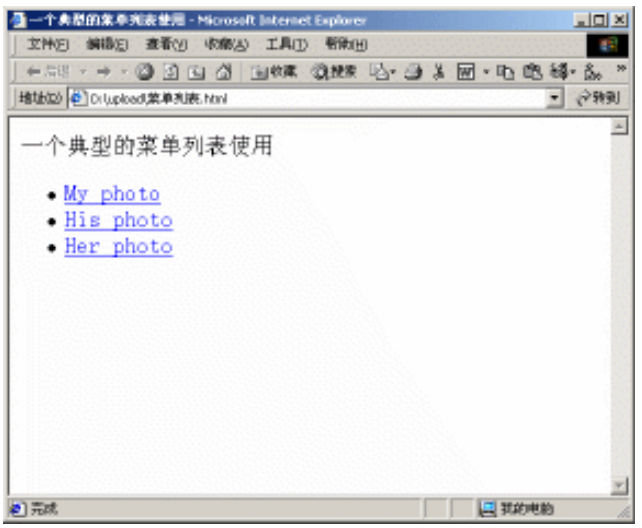

图 4.25　使用菜单列表

目录列表的标记为<dir>，它的功能与<menu>标记的功能非常类似。<dir>提供对某节文本进行 HTML 标识，这节文本潜在的作用比它实际的作用更大。<dir>标记的默认设置同无序列表。随着浏览器和其他应用程序开始支持<dir>，这将变得更常用。

<dir>标记的语法规则为：

一般形式：<dir>…</dir>

属性：None

<dir>的未来使用限制每个项目在 24 个字符之内，并且按行显示项目（类似于 UNIX 或 DOS 中使用/W 参数的文件目录），当前的浏览器并不支持这种解释。dir 单元不准备包括其他的 HTML 单元，尽管浏览器能对它们进行正确解释。

下面这段程序演示了<dir>的一个典型用法，其浏览效果如图 4.26 所示。

```
<html>
<head>
<title>一个典型的目录列表</title>
</head>
<body>
<h2>一个典型的目录列表</h2>
<dir>
  <lh><em>责任教师</em>
  <li>王老师
  <li>张老师
  <li>陈老师
  <li>李老师
</dir>
</body>
</html>
```

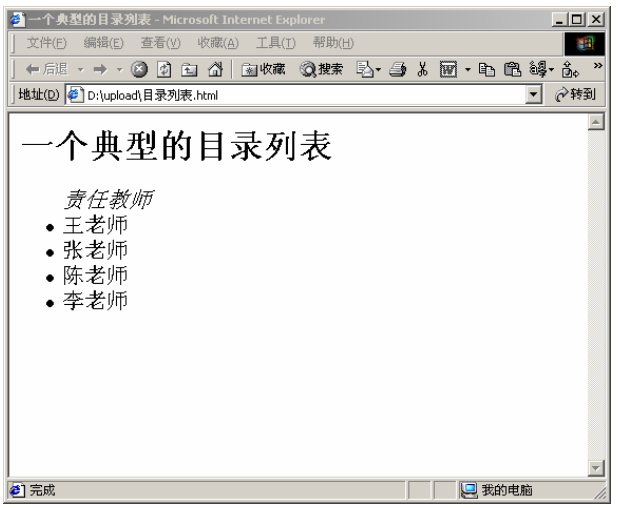

图 4.26 使用目录列表

默认情况下，浏览器不支持<dir>标记的任意惟一显示属性。正如菜单列表一样，Navigator 和 IE 仅仅只同无序列表一样解释目录列表（IE 没有标记）。

4.10　专 用 字 符

就绝大部分情况而言，字符实体的存在，使得用户可以包含标准 ASCII 字符集之外的专用字符。但有几个例外，少数字符在 HTML 本身有特定的含义。用户也必须使用这些字符的实体。

请看下面这样一个 HTML 文档行：

```
<p><code>if(x<o) return 1;</code><p>
```

它看起来完全正确，没有任何错误。但是，不幸的是，HTML 并不能显示所写的这行。这是为什么呢？问题出在<（小于）字符上。对于一个 HTML 浏览器来说，<字符意味着一个标记的开始。由于在这里的上下文中<字符实际上不是一个标记的开始，浏览器可能会被迷惑。同样，对于>（大于）字符也有类似的问题存在，因为在浏览器中它意味着 HTML 标记的结束，而对&和"也有同样的问题，因为它意味着换码字符的开始。对上面代码行要写出正确的 HTML，应用下面代码行替代：

```
<p><code>if (x&lt;o) return 1;</code><p>
```

HTML 对这些字符中的每一个提供了名字换码。<用<替代，>用>替代，&用&替代，"用&quot 替代。

注意：不要忘记最后的“ ; ”符号。

重引号换码是难以理解的一种。从技术上而言，如果要在文本中包含重引号，就必须用换码序列而不要键入重引号字符。这样的技术保证产生正确的 HTML 文档。但是，值得注意的是，当在 HTML 中照字义地键入重引号字符时，没有任何浏览器有在显示重引号字符方面的问题，也没有 HTML 文档像上述那样使用它。所以对绝大部分用户来说，可以安全地在自己的 HTML 文档中使用老式的”号，而不是使用换码符。

下面这段程序是专用字符的使用示例，其浏览效果如图 4.27 所示。

```
<html>
<head>
<title>专用字符的使用</title>
</head>
<body>
<h2>专用字符的使用</h2>
<p><code>
if(x&gt;0) return a;<br>
else if(x&lt;0) return b;<br>
else return c;<br>
```

```
</code></p>
</body>
</html>
```

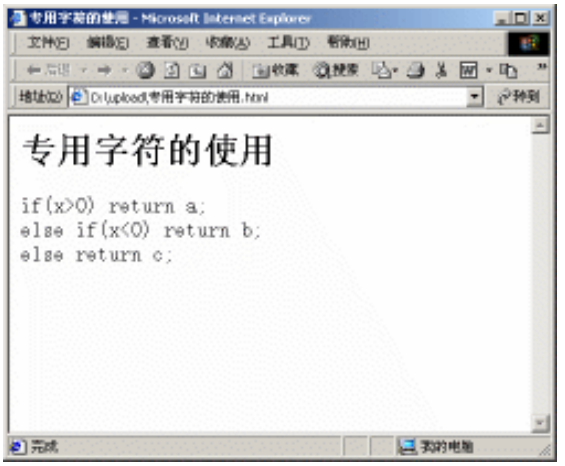

图 4.27 专用字符的使用

4.11 块 引 用

块引用用来从其他作品中引用大块的文本。在 HTML 语言中，使用 blockquote 单元标注块引用。其语法规则为：

一般形式：<blockquote>…</blockquote>
属性：None

块引用通常以缩进的形式显示出来。此外，有些浏览器会使用斜体字显示块引用，有些则在显示块引用时还能加上 Email 的标准引用记号>。

在引用他人作品时，请不要忘记提供作者以及版权等信息。

下面是一个块引用的实例，它的浏览效果如图 4.28 所示。

```
<html>
<head>
<title>块引用</title>
</head>
<body>
<h2>块引用示例</h2>
<hr>
结婚彷佛被围困的城堡。<br>
城外的人想冲进去，城里的人想逃出来。<br>
<p align=right>摘自 钱钟书《围城》
</body>
```

```
</html>
```

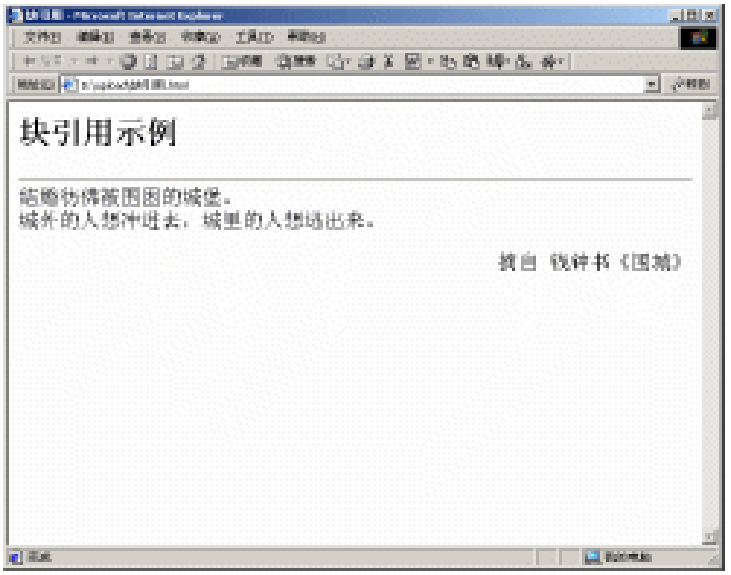

图 4.28　**块引用示例**

4.12　小 结

HTML **的核心就是各种标记的使用。本章介绍了各种文本格式的** HTML **标记，并给出了许多相应的示例。各种标记的灵活运用、组合运用是本章的难点。现在读者可以停下来建立真正可展示的** Web **页面了。但后面还有更多有趣的东西，现在还不要放下这本书。**

第 5 章　HTML 中的表格

表格是一种高级 HTML 结构，此结构允许用户把文本、图像和其他 HTML 内容放进含有行和列的长方形格子里。表格可以有边框，也可以没有边框。使用表格可以美化文档。在 HTML 中，表格不仅用于真正需要表格的场合，还可以用来设置分栏和对齐等。本章将介绍一些有关表格的知识。

5.1　创 建 表 格

为了在 HTML 中创建表格，需要先定义表格的各部分以及各项的摆放方式，然后为这些部分加进表格代码，再使用对齐方式、边框和给单元格上色来细化表格。在 HTML 中，表格中可包含非常广泛的内容，如段落、列表、标题、表单、数字和预格式化文本等。本节将介绍使用表头、数据和标题创建基本的表格。

5.1.1　表格的基本语法

表格的开始是标题（可以省略），后面是一个或多个表行。表格的每一行由多个表元（Cell，也称单元格）组成，浏览器将它们分成名称表元和数据表元。默认情况下，名称表元居中显示，数据表元居左显示。

在 HTML 中使用<table>标记来创建表格。其语法规则为：

一般形式：<table>…</table>
属性：border＝*n*，如：<table border=1>

<table>标记的 border 属性用来设置表格的边框。设置 border=0 表示边框无宽度，并且不显示。当用户需要按布局的意图使用表格时，可以使用无边框的表格；border=1 表示创建的表格的边框宽度为 1 个像素单位；border=2 表示创建的表格的边框宽度为 2 个像素单位，其他属性值类推。border 属性的默认值为 1。

除了必须用<table>标记定义表格外，还需要其他一些标记来定义表格的各个部分。

1．<tr>标记
该标记的语法规则为：

一般形式：<tr>…</tr>
属性：align=left|center|right；
　　valign=top|middle|bottom|baseline

2．<th>标记

该标记的语法规则为：

一般形式：<th>…</th>

属性：align=left|center|right；

valign=top|middle|bottom|baseline

3．<td>标记

该标记的语法规则为：

一般形式：<td>…</td>

属性：align=left|center|right；

valgn=top|middle|bottom|baseline

表格在 HTML 中是一行一行地指定的，每行包含该行所有单元的定义。所以，定义一个表格，首先要定义首行（也称顶行），再依次定义每个单元，然后再定义第 2 行和它的单元等等。HTML 中，表格的列是根据每行有多少单元数自动计算生成的。

<tr>标记用来定义表行（Table Row）。每一个表格行都用首标记<tr>和尾标记</tr>来表示。当然</tr>可以省略，但为了提高可读性和移植性，建议不要省略尾标记</tr>。<tr>的属性有 align 和 valign 两种，在 5.2 节中将详细介绍这两个属性的使用方法。

<th>标记用来定义表头（Table Head）单元。每个表头都用首标记<th>和尾标记</th>来表示。类似于</tr>标记，</th>也可省，但有时不使用尾标记，表格可能会失效，建议不要省略尾标记</th>。

<td>标记用来定义表格数据（Table Data）单元。它的用法与<th>标记类似。

用户可以想要多少行就要多少行，并根据列的需要确定每行有多少单元数，但必须确保每行有相同数量的单元，以使列能排齐。

下面这个示例定义了一个最简单的表格——一个只有一行的表格。

```
<html>
<head>
<title>只有一行的表格</title>
</head>
<body>
<table  border>
<tr>
  <th>表头单元</th>
  <td>数据单元</td>
  <td>数据单元</td>
  <td>数据单元</td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.1 所示。

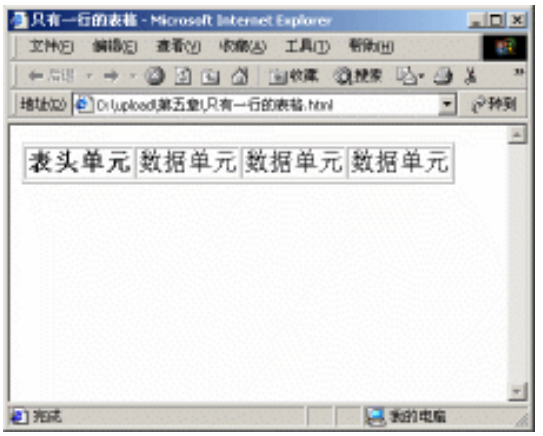

图 5.1 一个只有一行的表格

从图 5.1 可以看出，表头单元一般与数据单元有不同的显示方法，如用粗黑体字体显示。

如果表头是沿着表格顶部边界放置的，则定义表头的<th>标记应写在第 1 行内。下面这段程序给出了表头沿着表格顶部放置的一个表格。

```
<html>
<head>
<title>表头置于顶部的表格</title>
</head>
<body>
<p>表头置于顶部</p>
<table border>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
<p>表头置于左端</p>
<table border>
<tr>
  <th>语文</th>
```

```
  <td>80</td>
</tr>
<tr>
  <th>数学</th>
  <td>95</td>
</tr>
<tr>
  <th>英语</th>
  <td>90</td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.2 所示。

图 5.2　置于顶部的表格

如果单元内不想填入任何内容怎么办？这很容易，只要定义一个空表栏就行了。定义空表栏就是定义一个有<th>标记的单元或定义一个有<td>标记的单元而不在其中填入内容。有时，这类空表栏好像并不存在。如果要强制显示一个空表栏的话，则在一个本身没有其他文本的单元中增加一个换行符即可。

下面这段程序给出了两个空表栏的示例。

```
<html>
<head>
<title>具有空表栏的表格</title>
</head>
<body>
<p>空表栏</p>
<table border>
<tr>
```

```
  <th>语文</th>
  <th></th>
  <th>英语</th>
</tr>
<tr>
  <td></td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
<p>真正强制的空表栏</p>
<table border>
<tr>
  <th><br></th>
  <th><br></th>
  <th><br></th>
</tr>
<tr>
  <td><br></td>
  <td><br></td>
  <td><br></td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.3 所示。

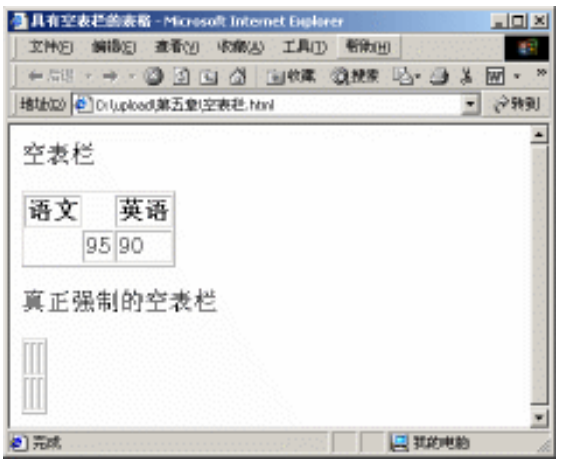

图 5.3 空表栏

5.1.2 设置表格标题

表格的标题用来告诉浏览者表格的主题。可以使用<caption>标记来设置表格的标题。

表格的标题是可省的，如果不想要标题，可以不加标题标记。<caption>标记的语法规则为：

一般形式：<caption>...</caption>
属性：align=left|center|right；
　　valign=top|bottom

align 属性用来设定标题在水平方向的对齐位置，它可以取 3 个值：left、center 和 right，默认值为 center。valign 属性用来设定标题是在表格的顶部还是底部，它可以取两个值：top 和 bottom，默认值为 top。

注意：一般来说，除非用户有一个很短的表格，否则应该把标题放在默认位置上（在表格顶部的中心）以便浏览者首先看到标题并知道表格主题是什么，而不是等阅读完后才看到标题。

下面这段程序分别给出了标题在默认位置、表格下方和居左的 3 种情况。

```
<html>
<head>
<title>标题的使用</title>
</head>
<body>
<h2>表格标题的默认位置</h2>
<table border>
<caption>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
<hr>
<h2>表格标题置于表格下方</h2>
<table border>
<caption valign=bottom>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
```

```
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
<hr>
<h2>表格标题置于表格的左端</h2>
<table border>
<caption align=left>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.4 **所示。**

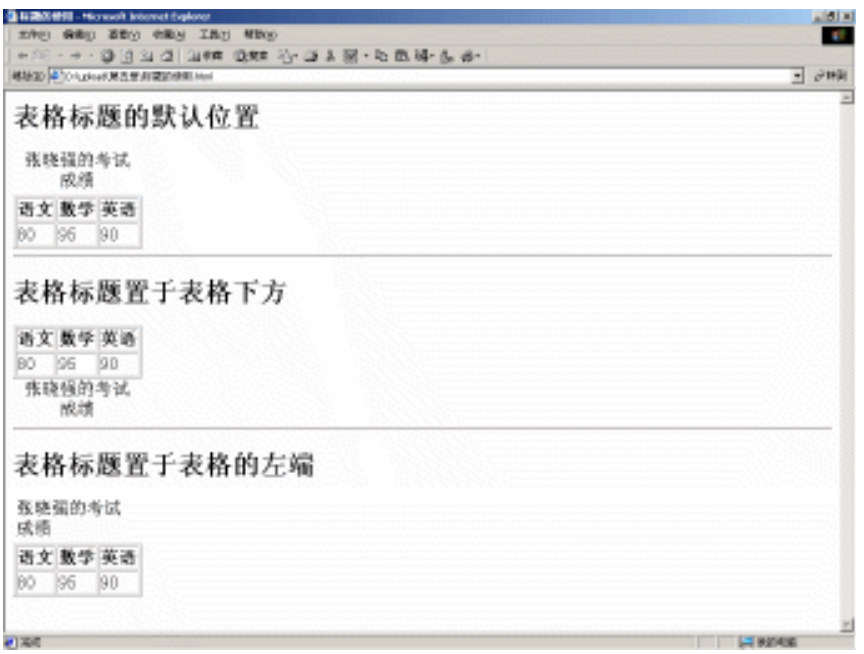

图 5.4 **标题的使用**

5.2　表格的对齐与布局

一旦在基本表格中安排好行、表头和数据的位置后，就可以开始细化表格了。对齐和布局包括两个方面，即表格内文字相对于表格和表格相对于页面的对齐与布局。这又包括水平方向和垂直方向两个方面。

5.2.1　表格内文字的对齐

表格内的文字相对于表格线的对齐可以整行设置，也可以只设置一个单元。水平方向对齐用 align 属性，垂直方向对齐用 valign 属性。

水平对齐是定义单元内数据是按单元左边缘对齐（Left），单元右边缘对齐（Right）还是居中对齐（Center）的。

垂直对齐是定义单元内数据在垂直方向的对齐，就是说，数据按单元的顶部对齐（Top），单元底部对齐（Bottom），或按单元中垂直方向的中间对齐（Middle）。

它们的使用格式如下：

```
<tr align=left|center|right>
<th align=left|center|right>
<td align=left|center|right>
<tr valign=top|bottom|middle>
<tr valign=top|bottom|middle>
<tr valign=top|bottom|middle>
```

表头单元的默认位置是水平和垂直方向居中，数据单元的默认位置是在垂直方向居中但与左边界对齐。

设置整行对齐方式时，只要把 align 或 valign 属性加进该行的<tr>标记中。设置单个单元对齐时，则在该单元的<td>或<th>标记中增加 align 或 valign 属性。

下面这段程序给出了表格内文字对齐的示例。

```
<html>
<head>
<title>表格内文字的对齐和布局</title>
</head>
<body>
<h2>表格内文字的对齐和布局</h2>
<table border width=300 height=100>
<caption>张晓强初一的期末成绩</caption>
<tr align=left valign=bottom>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
```

```
<tr align=right valign=top>
  <td>90</td>
  <td>98</td>
  <td>95</td>
</tr>
</table>
<hr>
<table border width=300 height=100>
<caption>张晓强初一的期末成绩</caption>
<tr>
  <th align=right  valign=top>语文</th>
  <th align=center valign=middle>数学</th>
  <th align=left valign=bottom>英语</th>
</tr>
<tr>
  <td align=right valign=middle>90</td>
  <td align=left valign=top>98</td>
  <td align=center valign=bottom>95</td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.5 所示。

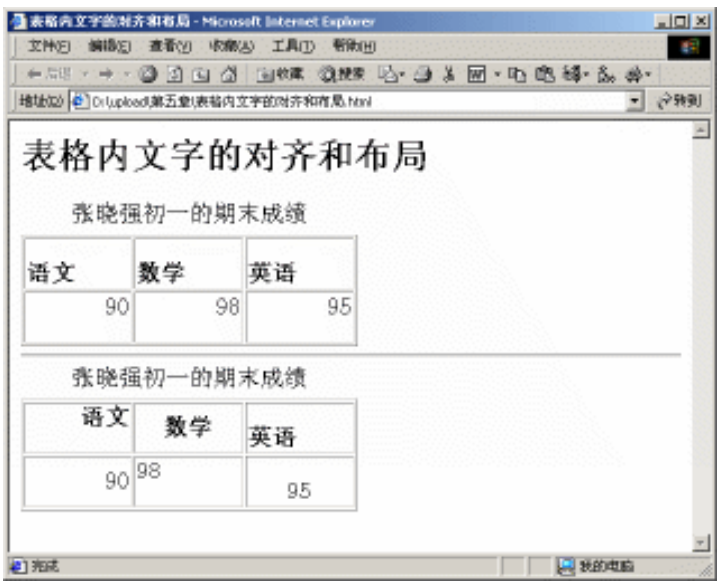

图 5.5 表格内文字的对齐和布局

5.2.2　表格在页面中的对齐与布局

通过 align 属性，表格整体可以在页面中水平居中、居左或居右。它的使用格式如下：

```
<table align=left|center right>
```

在系统默认时，表格不管在任一文本之上还是之下，都是沿着页面的左边一行显示。但使用 align 属性，可使表格按页面的左边或是右边对齐，并对文本按与图像同样的方式绕排。align 属性的取值为 left 时，表格按页面左边缘对齐，该表格后的所有文本则放在表格与页面右边缘之间。align 属性的取值为 center 时，表格居于页面中间，该表格后的所有文本则按页面左边缘对齐，但居于表格下方。align 属性的取值为 right 时，表格按页面右边缘对齐，该表格后的所有文本则放在表格与页面左边缘之间。

下面这段程序给出了表格在页面中对齐的示例。

```
<html>
<head>
<title>表格对齐</title>
</head>
<body>
<table align=left border>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>60</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
张晓明的数学成绩很好<br>
语文成绩还有待提高。<br>
英语成绩也不错<br>
<hr>
<table align=centerborder>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>60</td>
  <td>95</td>
```

```
  <td>90</td>
</tr>
</table>
张晓明的数学成绩很好<br>
语文成绩还有待提高。<br>
英语成绩也不错<br>
<hr>
<table align=right border>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>60</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
张晓明的数学成绩很好<br>
语文成绩还有待提高。<br>
英语成绩也不错<br>
</body>
</html>
```

浏览效果如图 5.6 所示。

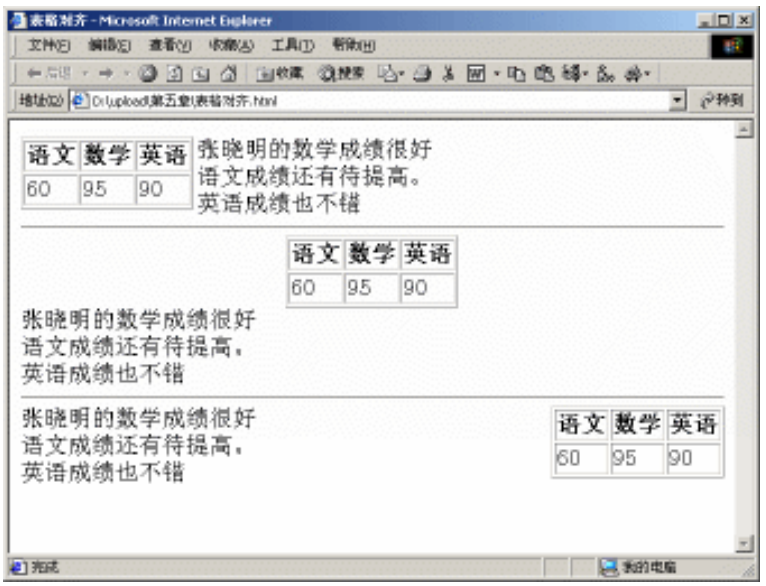

图 5.6 表格对齐

除了设定表格的位置以外，还可以设定表格与表格附近的文字之间的空距。hspace 控制水平间距，vspace 控制垂直间距。它们的使用方法如下：

```
<talbe vspace=number hspace=number>
```

下面为一个示例程序。

```
</html>
<head>
<title>表格与文字间的空距</title>
</head>
<body>
<table align=left border vspace=20 hspace=30>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>60</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
张晓明的数学成绩很好<br>
语文成绩还有待提高。<br>
英语成绩也不错<br>
</body>
</html>
```

浏览效果如图 5.7 所示。

图 5.7　设置表格与文字间的空距

注意：只有 Navigator 支持表格的 vspace 属性和 hspace 属性，其效果在其他浏

览器中不能显示，包括 IE。

5.3 跨越多行和多列的单元

以上所创建的表格都是规则的表格，每个单元只有一个数据或偶然有些空表栏。其实，HTML 的表格也可以是不规则表格，即一个单元跨越多行或者多列。跨行可以在<th>标记中使用 colspan 属性来实现，跨列可以在<td>标记中使用 rowspan 属性来实现，它们的属性值即为所跨的行数和列数。两者的使用方法分别如下：

```
<th colspan=number>
<td rowspan=number>
```

下面这段程序给出了一个跨越多行多列的表格示例。

```
<html>
<head>
<title>跨越多行多列的表格</title>
</head>
<body>
<table border>
<tr>
  <th colspan=4>张晓明的成绩<th>
</tr>
<tr>
  <th rowspan=2>第 1 学期</th>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>60</td>
  <td>95</td>
  <td>90</td>
</tr>
<tr>
  <th rowspan=2>第 2 学期</th>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>85</td>
  <td>75</td>
  <td>80</td>
</tr>
```

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

浏览效果如图 5.8 所示。

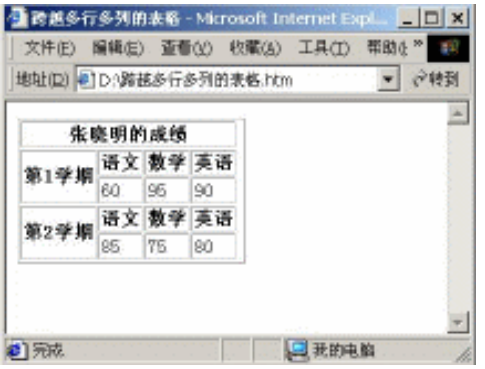

图 5.8　跨越多行多列的表格

5.4　定义表格的宽度和高度

前面创建的所有表格都是依赖于浏览器来指定其表格宽度和列的宽度及高度的。但是，有时也会出现根据需要自行控制表格和列的宽度及高度的情况。本节将介绍如何定义表格和列的宽度及高度。

在<table>标记中使用 width 属性来定义页面中表格的宽度，使用 height 属性来定义页面中表格的高度。这两个属性既能以像素为单位来定义其绝对值，也可以使用当前屏幕宽度的一个百分比（如 50%）来定义相对宽度和高度。

width 属性和 height 属性的值指定为相对值比指定为绝对值好，因为用户不可能知道浏览器窗口的宽度。使用百分数总是可以在任何宽度下对表格重新进行格式化，而使用指定像素的宽度就可能会引起运行时表格溢出页面之外。

在<th>或<td>标记中使用 width 属性和 height 属性可以定义单个单元的宽度和高度。与设定表格宽度一样，单元的 width 属性和 height 属性也有精确的像素宽度和百分比。

下面这段程序给出了定义表格宽度和高度的示例。

```
<html>
<head>
<title>定义表格的宽度和高度</title>
</head>
<body>
<h2>定义整个表格的宽度和高度</h2>
<table border width=75% height=150>
<caption>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
```

```
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
<hr>
<h2>定义单个单元的宽度和高度</h2>
<table border>
<caption>张晓强的考试成绩</caption>
<tr>
  <th width=200 height=150>语文</th>
  <th width=50%>数学</th>
  <th width=300>英语</th>
</tr>
<tr>
  <td height=30%>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.9 所示。

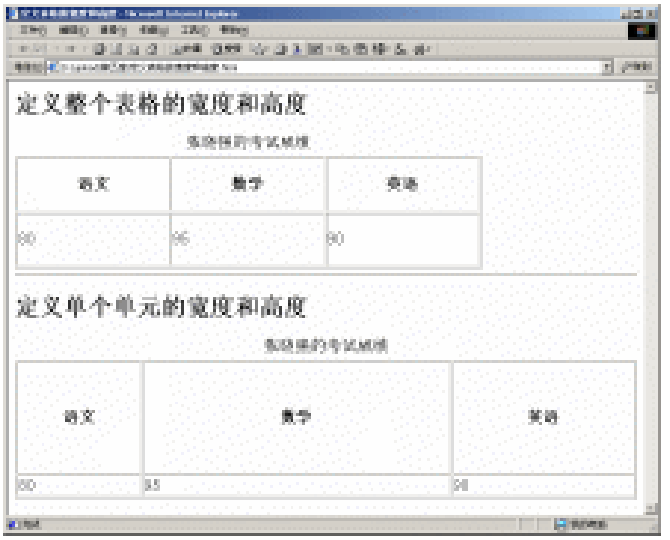

图 5.9 定义表格的宽度和高度

5.5　表格的其他特性

前 4 节已经将表格的大部分特性介绍完毕，至此已经能够制作出一个很完善的表格了。本节将介绍表格的其他一些特性，包括单元间隙、单元内部空白和表格颜色等，以便创建出更加多姿多彩的表格。

5.5.1　表格单元间隙

单元间隙是指两个表格单元之间的距离。它可以通过 cellspacing 属性来定义，如下所示：

```
<table border cellspacing=number>
```

cellspacing 属性的取值为一个整数，默认值为 2。

下面这段程序给出了单元间隙与 5.1 节中介绍的表格边框宽度的区别。

```
<html>
<head>
<title>表格单元间隙</title>
</head>
<body>
<h2>定义表格的单元间隙</h2>
<table border cellspacing=20>
<caption>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
<hr>
<h2>定义表格的边框宽度</h2>
<table border=10>
<caption>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
```

```
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
</body>
</html>
```

浏览结果如图 5.10 所示。

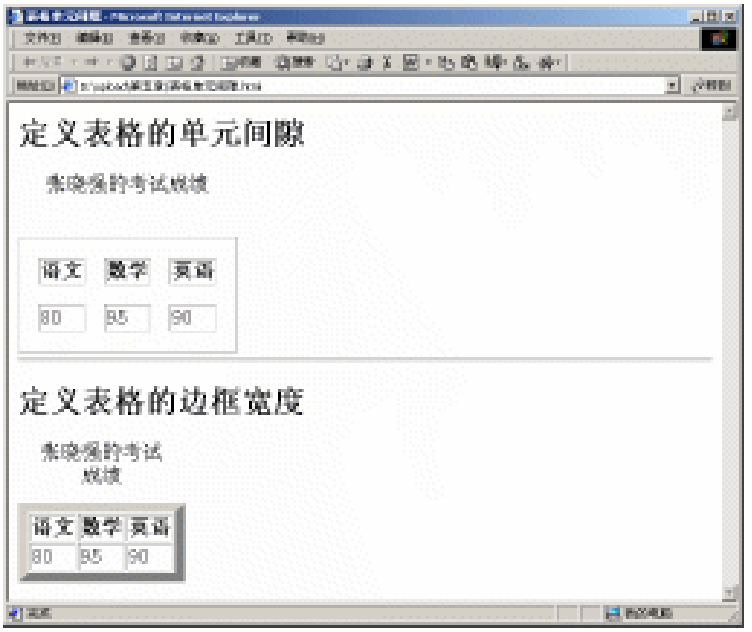

图 5.10 定义表格的单元间隙

5.5.2 表格单元边距

单元边距是指单元边框与单元内容之间空白的数值，它的默认大小为一个像素。可以用 cellpadding 属性设置单元边距的大小，如下所示：

```
<table border cellpadding=number>
```

下面这段程序给出了一个单元边距为 20 的表格示例。

```
<html>
<head>
<title>单元边距</title>
</head>
<body>
<h2>单元边距为 20 的表格</h2>
<table border cellpadding=20>
<caption>张晓强的考试成绩</caption>
```

```
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
<hr>
<h2>单元边距为默认值的表格</h2>
<table border>
<caption>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.11 所示。

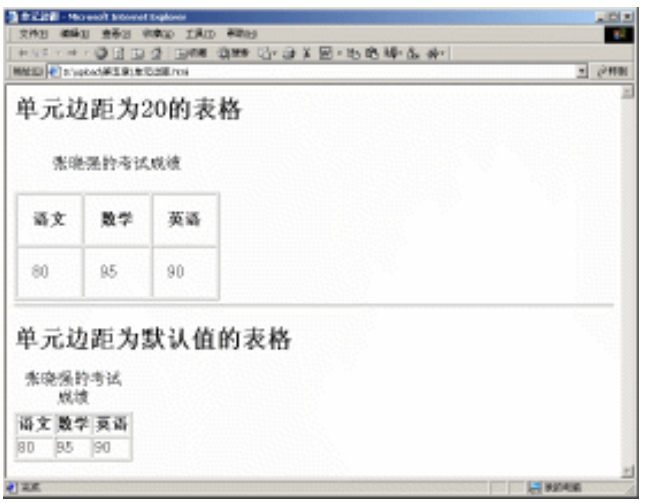

图 5.11　设置单元边距

5.5.3 表格颜色

表格的颜色，包括表格单元的背景色和背景图像、边框的颜色和亮度等，都可以自行设置。

表格单元的背景色彩由 bgcolor 属性指定，它的值既可以是预定义的颜色名（BLACK，SILVER，GRAY，WHITE，MAROON，RED，PURPLE，FUCHSIA，GREEN，LIME，OLIVE，YELLOW，NAVY，BLUE，TEAL，AQUA），也可以是十六进制的 RGB 颜色值。bgcolor 属性在<table>、<tr>、<th>和<td>标记中都可以使用。该属性的使用方法如下：

```
<table bgcolor=color>
<th bgcolor=#RRGGBB>
```

表格单元还可以将图像指定为背景。指定背景图像用 background 属性，它的属性值由标准的 URL 给定。background 属性可以在<table>、<th>和<td>标记中使用，它的使用方法如下：

```
<table background=URL>
<th background=URL>
```

利用<table>标记的 bordercolor 属性，还可以设置整个表格的边框线的颜色。它的值既可以是预定义的颜色名，也可以是十六进制的 RGB 颜色值。它的使用方法如下：

```
<table bordercolor=color>
<table bordercolor=#RRGGBB>
```

下面这段程序给出了一个设置表格背景色彩、背景图像以及边框线颜色的示例。

```
<html>
<head>
<title>表格的颜色</title>
</head>
<body>
<h2>表格单元的背景色和背景图像</h2>
<table border cellpadding=20>
<caption>张晓强的考试成绩</caption>
<tr bgcolor=#FFAA00>
  <th background=sky.jpg>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td bgcolor=#00AAFF>80</td>
  <td bgcolor=#00FF00>95</td>
  <td bgcolor=#FFFFF>90</td>
</tr>
</table>
<hr>
```

```
<h2>表格边框线的颜色</h2>
<table border=10 cellspacing=10 bordercolor="GREEN">
<caption>张晓强的考试成绩</caption>
<tr>
  <th>语文</th>
  <th>数学</th>
  <th>英语</th>
</tr>
<tr>
  <td>80</td>
  <td>95</td>
  <td>90</td>
</tr>
</table>
</body>
</html>
```

浏览效果如图 5.12 所示。

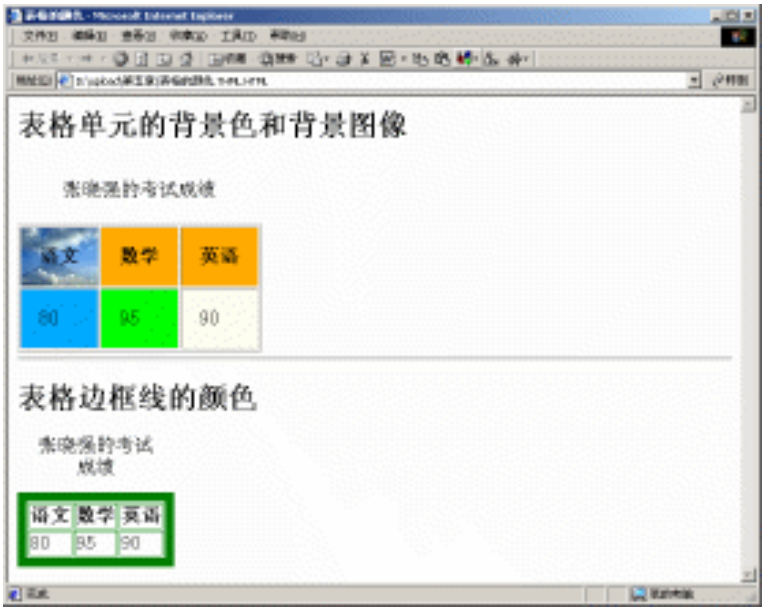

图 5.12　设置表格的颜色

由于表格本身就是具有三维效果的，因此可以利用表格的这个功能制作出具有三维效果的图片。下面这段程序利用表格实现阴影效果，它的浏览效果如图 5.13 所示。

```
<html>
<head>
<title>阴影效果</title>
</head>
<body>
```

```
<table cellpadding=0 cellspacing=0 bordercolordark="GRAY">
<tr>
  <td colspan=2 rowspan=2>
  <img src="family.jpg" height=220 width=300></td>
  <td height=8 width=10></td>
</tr>
<tr>
  <td height=212 bgcolor="GRAY"></td>
</tr>
<tr>
  <td height=8 width=10></td>
  <td bgcolor="GRAY" width=290></td>
  <td bgcolor="GRAY"></td>
</tr>
</table>
</body>
</html>
```

图 5.13 利用表格制作阴影效果

5.6 小 结

本章介绍了关于表格的所有知识。表格允许按行、按列安排信息，以便浏览者能很快从表格中得到所需要的信息。本章从表格操作中，介绍了关于表头和数据、标题、定义行和单元，在单元里对齐和创建跨越多行和多列的单元。最后介绍了表格的颜色，并利用表格功能制作出了具有三维效果的照片。要注意的是，创建一个美观大方的表格并非一蹴而就，通常需要有好几步工作。在正式创建一个表格之前，需要画出表格和行列摆放的草图，使表格具有更好的结构。

下一章将介绍有关框架的知识。

第 6 章　HTML 中的框架

从 HTML 3.2 版本开始支持框架（Frame）功能。可将浏览器的显示窗口分割为多个部分，每一部分可保持相对独立的操作，每一部分称为一个框架，所有的框架称为一个框架集（Frameset）。在 HTML 文档中首先要定义一个框架集，然后逐个定义框架，并应保证支持框架功能的浏览器能够正常浏览。

合理地使用框架可以使 HTML 文档组织更好、更易导航并且看起来更吸引人。

6.1　框架的基本概念

HTML 中的框架在许多方面与表格很相似。然而和表格不同，框架不仅组织数据，还组织浏览器的显示。本节将介绍如何创建一个框架、框架的嵌套以及<noframess>标记的使用，使读者对框架有个最基本的认识。

6.1.1　创建框架

框架扩展了 Web 页面的排版布局和链接能力，使用户能够在同一窗口中浏览不同的内容。每个框架具有以下特性：

- 可载入独立的 URL，与其他框架无关。
- 可赋予一个名字，以便其他 URL 作为目标进行调用。
- 可根据用户浏览器的窗口大小自动调整尺寸，并能选择是否允许用户手动调节框架大小。

框架可以用于下列场合：

- 页面设计者希望将某些信息一直保持为可见和不改变。例如，可将标题图示、工具栏等内容放入单独的框架中，这样在浏览时，此框架一直保持可见和内容不变，而不必重画。
- 使目录更加清晰。常见的作法是在窗口左侧设置一个目录框架，其中的每一项链接的内容显示到右侧的框架中。
- 用查询框架来包含 HTML 表单，以便提供数据库查询，再用结果框架接收每一查询的结果。

在 HTML 中使用<frameset>标记来创建框架。<frameset>标记也许是迄今为止 HTML 中最独特的标记了。它在页面中完全取代<body>标记，并且在使用框架时，不能只在页面的一部分加入框架。<frameset>标记的语法规则为：

一般形式：<frameset>…</frameset>

属性：cols n%= *n*%,n%= *n*%，**如：**< frameset cols n%=50%,n%=50%>；

rows n%＝*n*%,n%=*n*%，**如：**< frameset rows n%=30%,n%=30%,n%=50%>

<frameset>**标记的** cols **属性规定按纵向将窗口划分为若干框架，**rows **属性规定按横向将窗口划分成若干个框架。它们的属性值可以按窗口尺寸的百分比给定，通常几个百分比之和应该等于** 100%**。也可以按框架尺寸的实际像素值给定，两种情况下都可以只给定希望严格限制的值，而将其余值以星号（*）表示。**

<frame>**标记用于**<frameset>**标记内，它决定在某一特定页面上实际显示的内容。该标记的语法规则为：**

一般形式：<frame>

属性：src=*URL*，**如：**<frame src="a.html">

每个<frame>**标记都是一个空标记，这与在** HTML **列表中加入**<li>**标记不同。在**<frameset>**标记内部，**<frame>**标记只是简单地确定与**<frameset>**定义的特定框架有关的** URL **或文件名。**src **属性用来通知在该框架中应该加载哪个** URL**。**

下面这段程序给出了一个由 3 **个纵向排列的框架组成的窗口。**

```
<html>
<head>
<title>纵向排列的 3 个框架</title>
</head>
<frameset cols=30%,20%,50%>
  <frame src="a.html">
  <frame src="b.html">
  <frame src="c.html">
</frameset>
</html>
```

浏览效果如图 6.1 **所示。**

下面这个程序给出了一个由 3 **个横向排列的框架组成的窗口。**

```
<html>
<head>
<title>横向排列的 3 个框架</title>
</head>
<frameset rows=30%,20%,50%>
  <frame src="a.html">
  <frame src="b.html">
  <frame src="c.html">
</frameset>
</html>
```

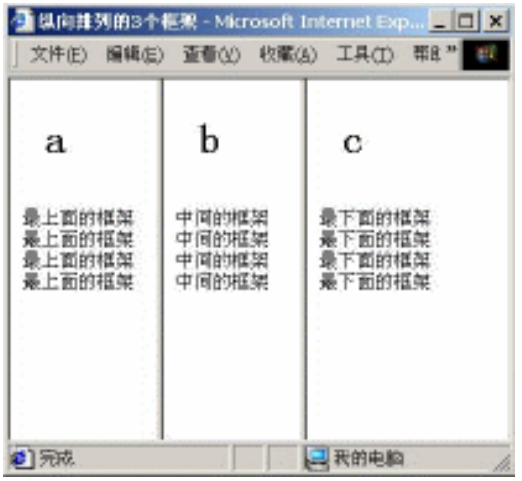

图 6.1　纵向排列的 3 个框架

浏览效果如图 6.2 所示。

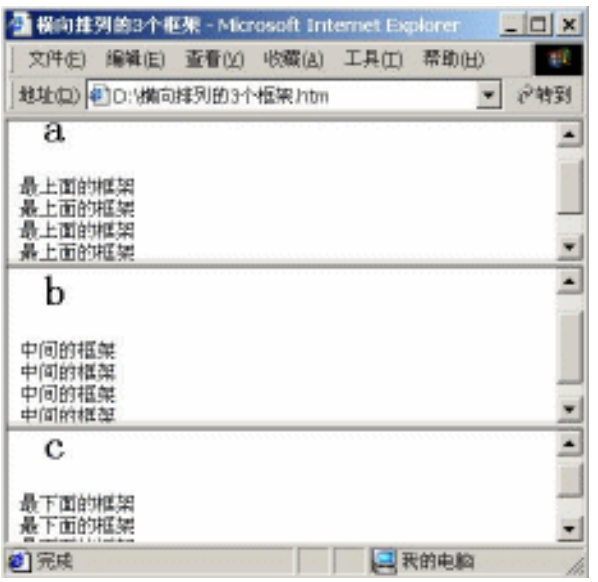

图 6.2　横向排列的 3 个框架

6.1.2　框架嵌套

框架还可以进行嵌套处理，即定义多个框架集，从而将窗口先按纵向划分后再按横向划分，或者按横向划分后再按纵向划分。从理论上说，设计很多层框架集是可能的，不过实际上框架不宜划分得过细，因为太小的框架无法清楚地显示信息。

下面这段程序将窗口纵向一分为二后，再将左右两侧的框架一分为二。

```
<html>
<head>
```

```
<title>框架的嵌套</title>
</head>
<frameset cols=30%,*>
<frameset rows=50%,*>
  <frame src="a.html">
  <frame src="b.html">
</frameset>
<frameset rows=25%,*>
  <frame src="c.html">
  <frame src="d.html">
</frameset>
</frameset>
</html>
```

浏览效果如图 6.3 所示。

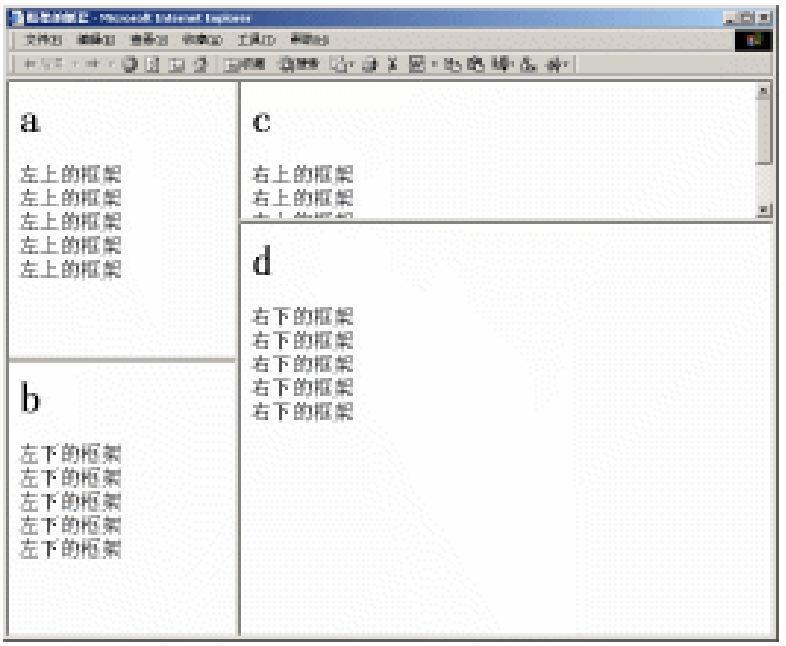

图 6.3 框架的嵌套

在<frameset>标记中规定框架的大小后，浏览时可通过拖动鼠标来改变框架的大小，如图 6.4 所示为通过拖动鼠标改变后的窗框效果，读者可以将其与图 6.3 相比较。

但有时不希望用户改变框架大小，影响浏览效果。如果不允许改变各框架大小，应在<frame>标记中使用 noresize 属性。它的语法规则为：

一般形式：<frame>

属性：noresize=None

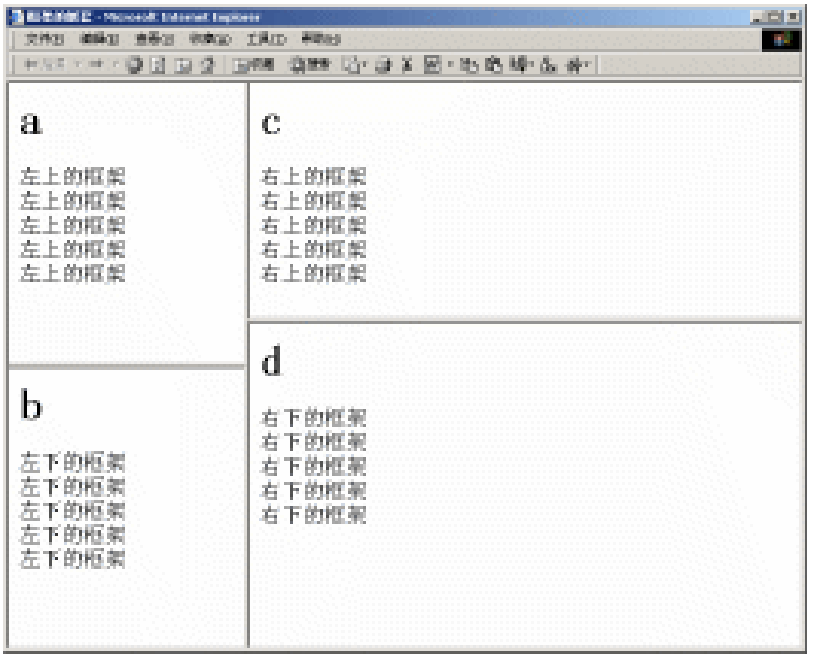

图 6.4　鼠标拖动后的框架嵌套

6.1.3　使用<noframes>标记

<noframes>**标记用来包含不能支持框架规范浏览器的** HTML **标记。具有框架功能的浏览器忽略**<noframes>**标记内的文字和** HTML **标记。而所有其他浏览器则通常忽略它们不能识别的另外一些框架标记，仅显示**<noframes>**标记内的文本。该标记的语法规则为：**

一般形式：<noframes>...</noframes>
属性：None

下面这段程序给出了一个使用<noframes>**标记的实例，它在支持和不支持框架规范的浏览器中都能显示。**

```
<html>
<head>
<title><noframes>标记的使用！</title>
</head>
<frameset cols=30%,*>
<frameset rows=50%,*>
  <frame src="a.html">
  <frame src="b.html">
</frameset>
<frameset rows=25%,*>
  <frame noresize src="c.html">
  <frame src="d.html">
```

```
</frameset>
<noframes>
<p>
<h1><noframes>标记的使用！</h2>
<hr>
该段文字用于不支持框架功能的浏览器。<br>
如果你能看到该段文字，则您的浏览器不支持框架功能。<br>
Internet Explorer 和 Netscape Navigator 浏览器都支持框架。<br>
请选择试用，你将看到不同的效果，谢谢。<br>
</p>
</noframes>
</frameset>
</html>
```

浏览效果如图 6.5 所示。

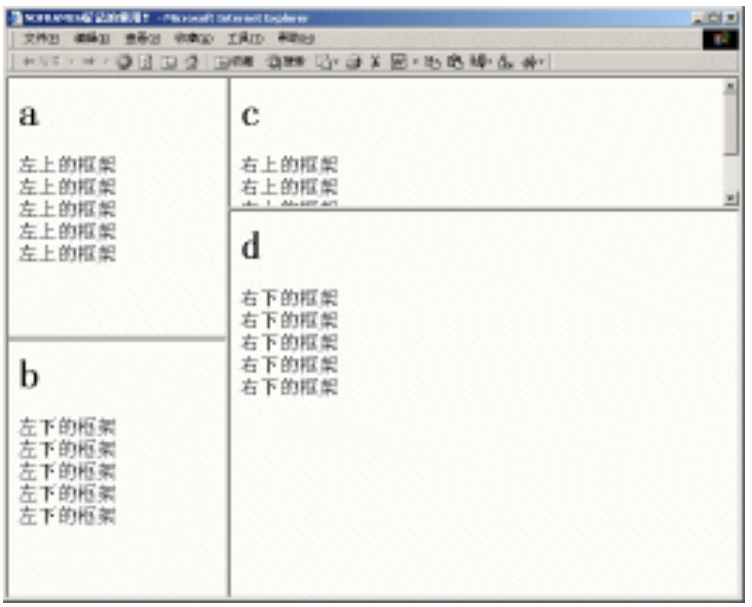

图 6.5 使用<noframes>标记在 IE 中的效果

6.2 框 架 设 置

在 HTML 中可以对框架的外观进行设置，以便产生更加美观的视觉效果。本节将介绍一些最基本的框架外观的设置。

6.2.1 框架边框设置

在一个框架集中，它的各个框架默认情况下是边框的。但也可以通过设置<frame>标记的 frameborder 属性的值来设置边框的有无。其语法规则为：

一般形式：<frame>
属性：frameborder=yes|no **或** 1|0

下面这个程序给出了对边框进行设置的示例。

```
<html>
<head>
<title>边框的设置</title>
</head>
<frameset cols=30%,*>
<frameset rows=50%,*>
  <frame src="a.html">
  <frame src="b.html">
</frameset>
<frameset rows=25%,*>
  <frame src="c.html" frameborder=NO>
  <frame src="d.html" frameborder=0>
</frameset>
</frameset>
</html>
```

浏览效果如图 6.6 **所示，读者可以将其与图** 6.3 **相比较。**

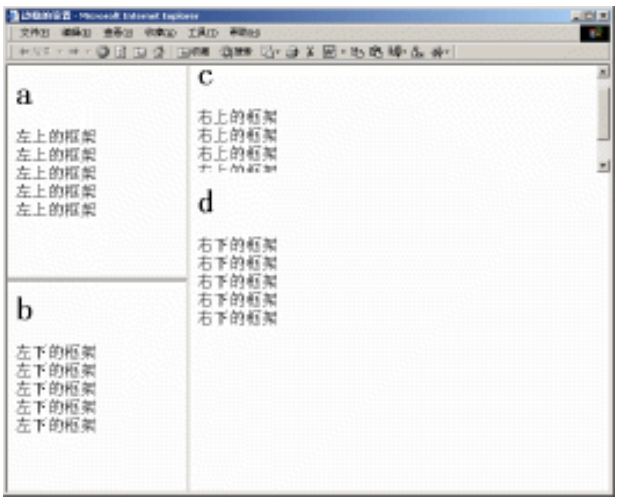

图 6.6　**设置框架的边框**

6.2.2　框架边空设置

一个框架实际上是一个独立的页面。可以通过设置<frame>**标记的** marginwidth **属性和** marginheight **属性来改变边空的大小。其语法规则为：**

一般形式：<frame>
属性：marginwidth=*n*；

marginheight=*n*

下面这个程序给出了一个示例，它在同一个文档中显示了两个框架，右边的框架设置了边空的大小，而左边的则使用了默认值。

```
<html>
<head>
<title>设置框架边空</title>
</head>
<frameset cols=50%,*>
  <frame src="a.html">
  <frame src="a.html" marginwidth=70 marginheight=60>
</frameset>
</html>
```

浏览效果如图 6.7 **所示。**

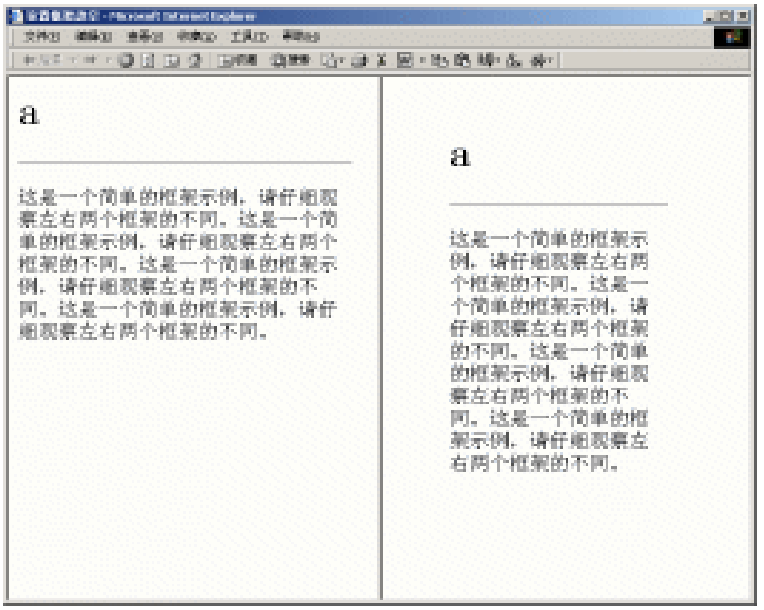

图 6.7 **设置框架的边空**

6.2.3 框架间距设置

框架间距是指框架间的空白区域，它可以通过设置<frameset>**标记的** framespacing **属性来改变，其语法规则为：**

一般形式：<frameset>
属性：framespacing=*n*

下面这段程序给出了一个设置框架间距的示例，它的 3 **个框架的间距被设置为** 50**。**

```
<html>
<head>
<title>设置框架间距</title>
</head>
<frameset framespacing=50 cols=30%,*>
<frameset rows=50%,*>
  <frame src="a.html">
  <frame src="b.html">
</frameset>
<frameset rows="25%,*">
  <frame src="c.html">
  <frame src="d.html">
</frameset>
</frameset>
</html>
```

浏览效果如图 6.8 所示。读者可以将其与图 6.4 相比较。

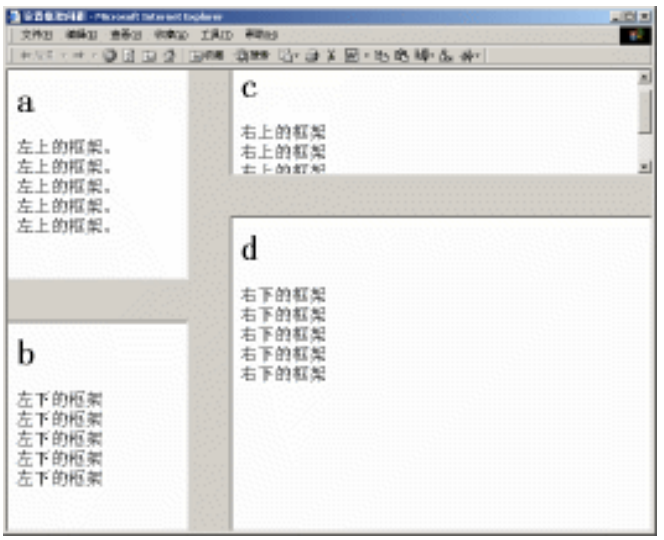

图 6.8　设置框架间距

6.2.4　框架滚动条设置

默认情况下，当框架不足以显示整个页面时自动出现滚动条，反之滚动条自动消失。可以通过设置<frame>标记的 scrolling 属性来使滚动条一直显示或不显示。其语法规则为：

一般形式：<frame>
属性：scrolling=yes|no|auto

scrolling 属性的默认值为 auto。
下面这个程序给出了一个设置滚动条的示例。

```
<html>
<head>
<title>设置滚动条</title>
</head>
<frameset cols=30%,*>
<frameset rows=50%,*>
  <frame scrolling=auto src="a.html">
  <frame src="b.html">
</frameset>
<frameset rows=25%,*>
  <frame scrolling=no src="c.html">
  <frame scrolling=yes src="d.html">
</frameset>
</frameset>
</html>
```

浏览效果如图 6.9 所示。读者可以将其与图 6.3 相比较。

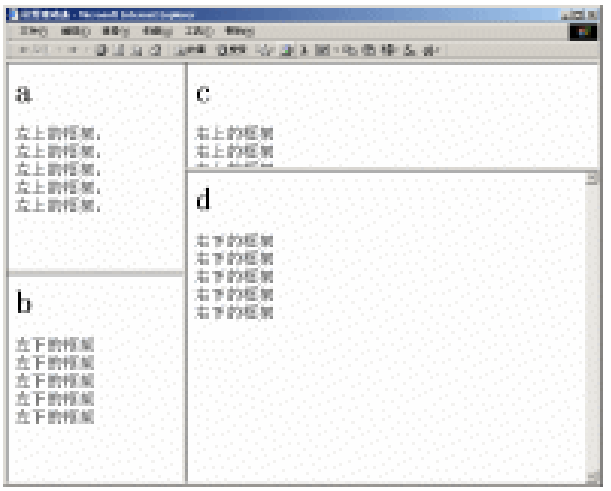

图 6.9 设置滚动条

6.2.5 框架边框颜色设置

在 HTML 4.0 中，框架边框的颜色是可以改变的，可以通过设置<frameset>标记的 Bordercolor 属性来改变框架边框的颜色。颜色可以用十六进制的 RGB 值给定，也可以用预定义颜色名给定。其语法规则为：

一般形式：<frameset>

属性：bordercolor =*color*

下面这个程序给出了一个具有蓝色边框的框架窗口示例。

```
<html>
```

```
<head>
<title>设置边框颜色</title>
</head>
<frameset cols=30%,* bordercolor=#0000FF framespacing=20>
<frameset rows=50%,*>
  <frame src="a.html">
  <frame src="b.html">
</frameset>
<frameset rows=25%,*>
  <frame src="c.html">
  <frame src="d.html">
</frameset>
</frameset>
</html>
```

浏览效果如图 6.10 所示。

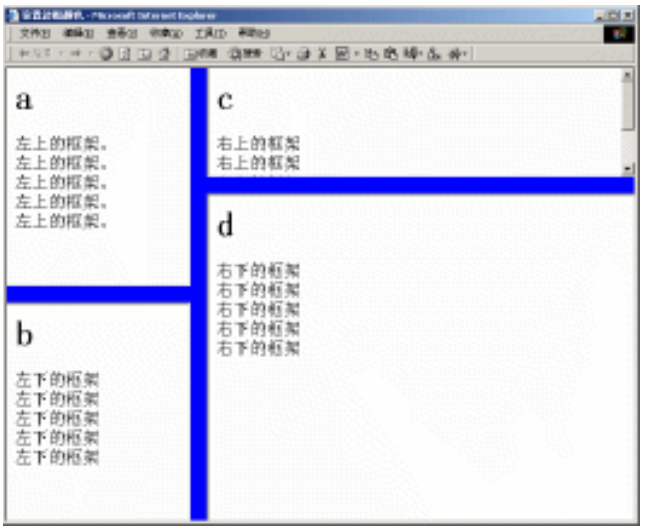

图 6.10　设置边框颜色

6.3　指定框架窗口的目标

迄今为止，框架已经具有加载彼此相互独立的 URL 的能力，这样就可以在同一个浏览窗口中显示两个或多个不同的 HTML 页面。这有什么好处呢？在很多情况下，它可以用于同时提供多个文档。

例如，以现在掌握的知识，就可以用框架在已建立的每个 HTML 页面的顶部加一个按钮条图像映像。不过这有一点乏味——因为每个链接都必须指向一个新的将加载该按钮条的文档。

但是如果只想把按钮条放在顶部框架窗口中，并且在另一个框架中加载一个新文档，那怎么办呢？学习了目标（Targeting）的知识，这个问题就迎刃而解了。

6.3.1 使用 name 属性

首先需要给框架的窗口命名，至少必须给想要改变的窗口命名。这可以通过设置<frame>标记的 name 属性来完成。其语法规则为：

一般形式：<frame>
属性：src=*URL*；
　　name=*window_name*

这个形式并不陌生，因为它与<a name>链接的 name 属性的用法有点类似。一旦框架窗口获得了惟一的名字，就可以从其他框架窗口中直接访问它了。

虽然可以给框架起各种各样的名字，但是 HTML 规定名字不能以下划线（_）开头，否则这个名字将被忽略。

name 属性的使用示例请参照下一节。

6.3.2 指定框架窗口的目标

成功地给框架命名之后，就可以用一个超文本链接指向目标名。在典型的<a>锚标记中使用 target 属性，就可以完成这个任务。其语法规则为：

一般形式：<a>...
属性：href=*new_URL*；
　　target=*window_name*

其中 *new_URL* 是一个希望出现在框架窗口中的新文档，而 *window_name* 就是用<frame>标记的 name 属性指明的框架窗口的名字。

下面这个示例给出了一个带两个框架的文档，在屏幕左边的框架中放置了一个目录，在右边的框架中放置各章节的具体内容。完成这项工作，需要 3 个不同的程序段，并且 3 者都应该是完整的 Web 文档。

程序 1——HTML 语言入门与精通.html

```
<html>
<head>
<title>HTML 语言入门与精通</title>
</head>
<frameset cols=20%,*>
  <frame src="目录.HTML"  noresize>
  <frame src="HTML 语言的基本概念.HTML" name="大窗口">
</frameset>
</html>
```

程序 2——目录.html

```
<html>
<head>
<title>目录</title>
</head>
<body>
<h1>目录</h1>
<hr>
<ul>
  <li><a href="HTML 语言的基本概念.html" target="大窗口">
    HTML 语言的基本概念</a>
  <li><a href="HTML 中的超级链接.html" target="大窗口">
    HTML 中的超级链接</a>
  <li><a href="HTML 文本格式.html" target="大窗口">
    HTML 文本格式</a>
  <li><a href="HTML 中的表格.html" target="大窗口">
    HTML 中的表格</a>
  <li><a href="HTML 中的框架.html" target="大窗口">
    HTML 中的框架</a>
  <li><a href="HTML 表单.html" target="大窗口">
    HTML 表单</a>
</ul>
</body>
</html>
```

程序 3——HTML 语言的基本概念.html

```
<html>
<head>
<title>HTML 语言的基本概念</title>
</head>
<body>
<h1 align=center>HTML 语言</h1>
<h2 align=center>——入门与精通</h2>
<hr>
<h2>HTML 语言的基本概念</h2>
<ui>
  <li><a href="HTML 基本语法.html" target="大窗口">
    HTML 基本语法</a>
  <li><a href="标记符基本属性.html" target="大窗口">
    标记符基本属性</a>
  <li><a href="HTML 标记.html" target="大窗口">
    HTML 标记</a>
  <li><a href="首部标记.html" target="大窗口">
    首部标记</a>
```

```
    <li><a href="正文标记符.html" target="大窗口">
      正文标记符</a>
    <li><a href="显示特殊字符.html" target="大窗口">
      显示特殊字符</a>
    <li><a href="添加注释.html" target="大窗口">
      添加注释</a>
    <li><a href="小结.html" target="大窗口">
      小结</a>
</ui>
</body>
</html>
```

本例的浏览效果如图 6.11 所示。

图 6.11 指定框架窗口的目标

6.4 小 结

本章介绍了框架的一些基本知识，如建立一个框架、框架的嵌套、设置框架的边框等。最后介绍了如何指定框架窗口的目标，并给出了一个较复杂的综合程序示例。

通过学习本章，将对框架有一个最基本的认识。但要熟练地使用框架，创建出更加完美的页面，还需要多实践、多学习。

下一章介绍 HTML 表单与 CGI 脚本。

第 7 章　HTML 表单与 CGI 脚本

HTML 表单用于收集信息和反馈。就像 HTML 提供了许多输出信息的机制一样，使用 HTML 表单可以允许信息的输入。这些表单可以收集形式自由的文本信息，得到 yes 或者 no 的答案，也可以从一组选项中得到答案。

表单可以在不同的场合使用。它们可以完成简单的事情，例如在会议前收集雇员已经阅读了要求阅读的文档的情况。表单输入可以创建并维护一个很重要的对特定的话题的讨论组。当与安全的传输方法相结合时，表单还可以用来在 Web 上做生意。在许多其他的应用中，也可以使用表单。

7.1　表 单 概 述

HTML 从 2.0 版本开始包括表单（Form），使 HTML 文档能够接受用户输入和对用户输入进行反馈。将输入字段放入段落和预格式化文本中创建表单，提供了与用户交互的可能，也是动态 HTML 技术的基础。

表单的作用是从用户方面收集信息，当用户填好表单上所需信息并将表单交付处理后，服务器就可以得到表单中包含的信息，并经 CGI（Common Gateway Interface，公共网关接口）程序进行处理。

显示表单的 HTML 标记相当简单。标记<form>的使用表示在文档中产生表单。该标记的语法规则为：

一般形式：<form>…</form>
属性：action=*URL*；
　　method=get|post

在 HTML 中，有 3 类用于创建域的表单：<textarea>、<select>和<input>。在<form>和</form>标记之间可以放置任意数量的这些标记。下面是对每个标记的简短描述：

- <textarea>：这个标记定义了一个文本输入框，用户可以在其中输入多行文本。
- <select>：这个标记允许用户在滚动框或者弹出菜单中，从多个选项中选择。
- <option>：这个标记提供了<select>标记中的选项。
- <input>：这个标记提供了其他所有类型的输入形式：单行文本、单选按钮、复选框和提交或者清除表单的按钮。

每一个可变的字段都由<input>、<textarea>或<option>标记来定义，并且必须包括 name 属性来标识其值。

<form>标记最重要的两个属性是 action 和 method。

action 属性是最重要的 form 属性之一，它指定服务器对表单执行的动作。action 指定了负责处理表单数据的程序（程序的 URL）。method 则说明应该如何处理来自表单的数据，最常用的属性值为 get 和 post。

表单除在开始定义其 method 和 action 属性外，通常在结束前会使用 type=submit 和 type=reset 设计“提交”（“寄送”）和“重新设置”两个按钮。这两个按钮上显示的文字可用 value 属性指定，不指定时其默认值分别为 Submit Query 和 Reset。所以，表单的基本格式如下：

```
<form action=URL method=get|post>
...
...
<input type=submit>
<input type=reset>
</form>
```

表单中用<input>标记来收集数据。<input>标记决定了表单提供给用户的输入形式，它支持 type、name 和其他属性。示例如下：

```
<input type="type" name="name">
  "type"=text,password,checkbox,radio,image,hidden,submit,reset
  "name"=cgi 程序所需的符号名。
```

7.2 文本输入框

当预计用户的输入文本超过一行时，可使用文本输入框。文本输入框的标记为<textarea>，该标记的语法规则为：

一般形式：<textarea>…</textarea>

属性：name=*text_area_ name*；

rows=*n*；

cols=*n*

name 属性用来规定文本输入框的名字，它的值由用户自己定义。rows 属性用来定义文本输入框的行数。cols 属性用来定义文本输入框的列数。

下面这段程序定义了一个名为“我的第 1 个表单”文本输入框，它有 5 行 40 列。

```
<html>
<head>
<title>我的第一个表单</title>
</head>
<body>
<h2>我的第一个表单</h2>
```

```
<form method=post action="/cgi-bin/comment_script">
<textarea name="我的第 1 个表单" rows=5 cols=40>
在此输入你要写的话。
</textarea>
<p>
<input type=submit>
<input type=reset>
</form>
</body>
</html>
```

浏览效果如图 7.1 所示。

图 7.1　使用文本输入框

在这个实例里，表单给服务器上的 cgi-bin 目录中的 comment-script 脚本发送用户输入的全部内容，并使用 post 方法来发送。

提示：对于文本输入框中较长的行，可以设置文本是否进行换行（Word Wrapping）。文本换行由 wrap 属性规定。wrap=off 时不换行。wrap=soft 是软换行，即显示时换行，但发送时是加上不换行，俗称“软回车”。wrap=hard 是硬换行，即插入回车字符，俗称“硬回车”。wrap 属性的默认值为 off，即不换行。

7.3　<input>标记的使用

表单最重要的特性是它可接受用户的输入。<input>标记则用于接收用户的输入信息。本节将向读者详细介绍<input>标记的使用方法。

注意：<input>标记只有首标记，没有尾标记，即没有</input>标记。

7.3.1 输入文本

text 是<input>标记的 type 属性最常取的一个属性值，它建立一个单行的文本输入框。当属性值为 text 时，还可以使用<input>标记的 3 个可选属性。第 1 个可选属性为 size，它定义输入区域分配的显示空间大小，默认值由浏览器决定。第 2 个可选属性为 maxlength，它限定用户所能输入的字符数，默认值为无限。第 3 个可选属性为 value，它提供了输入区域的初始值。size 的值一般比 maxlength 的值大，但也可以小一些，当 maxlength 的值比 size 的值大的时候，浏览器将设置滚动数据。

下面这段程序给出了一个文本输入的示例。

```
<html>
<head>
<title>文本输入</title>
</head>
<body>
<h2>一个文本输入的示例</h2>
<form action="/cgi-bin/comment_script" method=post>
您的姓名：<input type=text name="姓名" size=10><br>
工作单位：<input type=text name="工作单位" size=40 maxlength=50><br>
家庭住址：<input type=text name="家庭住址" size=40 maxlength=50><br>
您的主页：<input type=text name="主页" value="http://" size=40><p>
<input type=submit value="提交">
<input type=reset value="重设"></p>
</form>
</body>
</html>
```

浏览效果如图 7.2 所示。

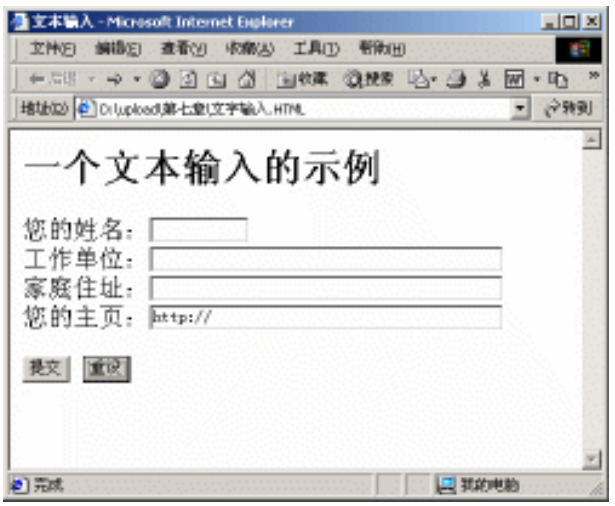

图 7.2 输入文本

7.3.2　输入密码

当要设定一个密码输入时，可将 type **的属性值取为** password**。除了在输入字母时仅显示小圆点或（由浏览器选择）类似的图形以便进行保密外，**password **选项和** text **选项几乎一样。**

下面这段程序给出了一个密码输入的示例。

```
<html>
<head>
<title>密码输入</title>
</head>
<body>
<h2>一个密码输入的示例</h2>
<form action="/cgi-bin/comment_script" method=post>
用户名：<input type=text name="姓名" size=10>
<font face="黑体">@microsoft.com</font><br>
密  码：<input type=password name="密码" size=10
maxlength=15><br>
<p>
<input type=submit value="确定">
<input type=reset value="取消"></p>
</form>
</body>
</html>
```

浏览效果如图 7.3 **所示。**

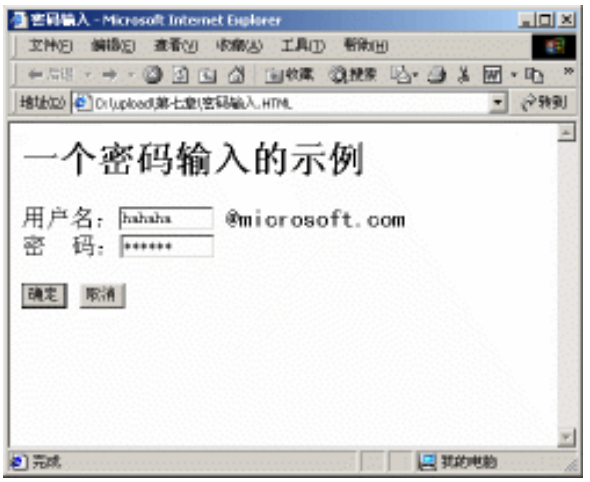

图 7.3　**输入密码**

注意：系统保存的仍然是用户所输入的文本，而不是小圆点或类似的符号。

7.3.3 复选框和单选按钮

<input>标记支持的另外两个 type 属性是复选框和单选按钮，它们相应的 type 属性值分别为 checkbox 和 radio。

复选框提供多项选择，即在所提供的选项中可选择多项；单选按钮提供单项选择，即在提供的选项中只能选择一项。在形式上，复选框为方框形，选中时方框内出现对号符；单选框为圆圈形，选中时圆圈内填一实心圆点。

使用 checked 属性还可以确定复选框是否已被选中（如果需要的话，它必须是专由用户选择的）。

下面这段程序给出了一个使用复选框和单选按钮的示例。

```
<html>
<head>
<title>复选框和单选按钮</title>
</head>
<body>
<h2>复选框和单选按钮的示例</h2>
<form action="/cgi-bin/comment_script" method=post>
<p><input type=checkbox name="语种" checked>汉语</p>
<p><input type=checkbox name="语种">英语</p>
<p><input type=checkbox name="语种">法语</p>
<p><input type=checkbox name="语种">俄语</p>
<p>
<input type=submit value="确定">
<input type=reset value="取消">
</p>
</form>
<hr>
<form action="/cgi-bin/comment_script" method=post>
<p><input type=radio name="语种" checked>汉语</p>
<p><input type=radio name="语种">英语</p>
<p><input type=radio name="语种">法语</p>
<p><input type=radio name="语种">俄语</p>
<p>
<input type=submit value="确定">
<input type=reset value="取消">
</p>
</form>
</body>
</html>
```

浏览效果如图 7.4 所示。

图 7.4　使用复选框和单选按钮

7.3.4　隐藏表单组件

有时由于某种需要，可能希望将表单中的一个或多个组件隐藏起来。<input>标记的 type 属性可以取值 hidden 来实现这种功能。当 type=hidden 时，<input>标记还需要两个属性：name 和 value。

注意：这里的隐藏并非真的隐藏，因为用户可以直接选择“查看源文件”来看隐藏域的值。但从程序员观点来看这是比较有用的。例如，在一个大的 Web 站点上，隐藏值可能告诉多用途脚本（在许多表单中）哪个特定表单要发送数据，这样该脚本就知道怎样处理这个数据。

下面这段程序给出了一个隐藏表单组件的示例。

```
<html>
<head>
<title>隐藏表单组件</title>
</head>
<body>
<h2>隐藏表单组件的示例</h2>
<form action="/cgi-bin/comment_script" method=post>
<p>
<input type=hidden name=隐藏 value="哈哈，你还能看到我吗？">
从图中看不到属性 value 的值了。<br>
它被隐藏起来了。<br>
<br>
```

```
<input type=submit value="确定">
<input type=reset value="取消">
</p>
</body>
</html>
```

浏览效果如图 7.5 所示。

图 7.5 隐藏表单组件

7.4 创建列表框

HTML 的<select>标记用于建立不同类型的列表框，包括下拉式列表和滚动式列表两种。列表框用<select>和<option>两种标记实现。其基本格式如下：

```
<select name="name" size="size" multiple>
<option selected value="value">选项文本
<option value="value">选项文本
...
</select>
```

<select>标记用来定义列表框。它支持 name、size 和 multiple 这 3 个属性。name 属性用于指定<select>标记的名字。size 属性用于定义列表框的大小，即一次可以显示的列表项的数目，如果 size=1（默认值），列表框显示为下拉列表。mutiple 属性用于允许用户进行多项选择（使用 Ctrl 键或 Shift 键配合鼠标进行多项选择）。

<option>标记定义列表框的各选项。它可以使用属性 selected，表示预先已选定。还可以使用属性 value，用来指定用户选择某选项后的返回值。

注意：当使用 mutiple 属性时，列表显示为滚动式列表。

下面这段程序设计了 3 个列表框。一个是下拉式列表，只允许用户进行单项选择；另

两个是滚动式列表，允许用户进行多项选择，且最后一个列表预选了两个选项。

```
<html>
<head>
<title>列表框</title>
</head>
<body>
<h2>使用列表框</h2>
<form action=/cgi-bin/comment_script  method=post>
<select name="语种">
<option>汉语
<option value="第 1 外语">英语
<option>法语
<option>俄语
</select>
<p>
<input type=submit value="确定">
<input type=reset value="取消">
</p>
</form>
<hr>
<form action=/cgi-bin/comment_script method=post>
<select name="语种" multiple>
<option selected>汉语
<option value="第 1 外语">英语
<option>法语
<option>俄语
</select>
<p>
<input type=submit value="确定">
<input type=reset value="取消">
</p>
</form>
<hr>
<form action=/cgi-bin/comment_script method=post>
<select name="语种" size=3 multiple>
<option selected>汉语
<option value="第 1 外语" selected>英语
<option>法语
<option>俄语
</select>
<p>
<input type=submit value="确定">
```

```
<input type=reset value="取消">
</p>
</form>
</body>
</html>
```

浏览效果如图 7.6 所示。

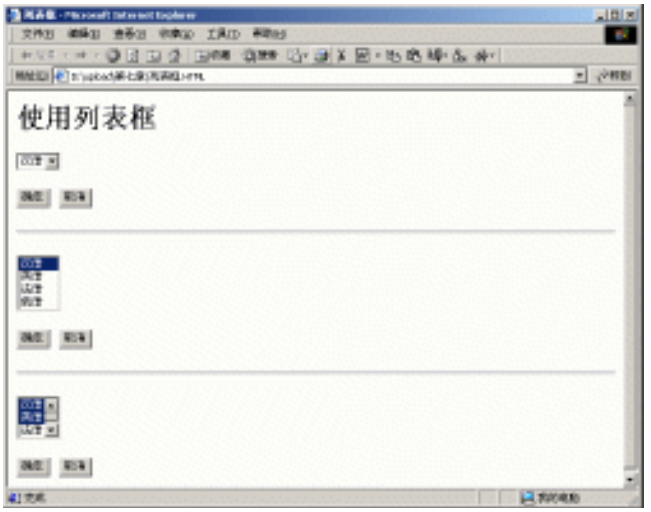

图 7.6 使用列表框

7.5 表 单 设 计

表单设计的思想中心是要使用户理解和有足够的优势，使他们能有耐性来填写完表单。如果给用户的鼓励越少，他们就越不可能去尝试填写表单。一个清楚而短小的表单可能会比冗长而混乱的表单更能吸引用户。

本节将介绍如何利用已经学过的一些 HTML 标记来格式化一个表单。

7.5.1 使用<pre>标记

到目前为止，在建立表单时最恼人的一个问题是在向下延续页面时不能排齐文本输入框的各个域。例如，在实例里使用用户名域和密码域时，它们总显得有点参差不齐。

使用<pre>标记可以很好地解决这个问题。因为在两个标记之间的任何内容都要使用空格和回车，而<pre>标记能够识别回车和空格。

下面这个程序给出了不使用<pre>标记和使用<pre>标记的作比较的示例。

```
<html>
<head>
<title>使用<pre>标记</title>
</head>
```

```
<body>
<h2>使用<pre>标记</h2>
<form action="/cgi-bin/comment_script" method=post>
用户名：<input type=text name="姓名" size=10>@microsoft.com<br>
密码：<input type=password name="密码" size=10 maxlength=15><br>
<p>
<input type=submit value="确定">
<input type=reset value="取消">
</p>
</form>
<hr>
<form action="/cgi-bin/comment_script" method=post>
<pre>
用户名：<input type=text name="姓名" size=10>@microsoft.com
密码： <input type=password name="密码" size=10 maxlength=15>
</pre>
<p>
<input type=submit value="确定">
<input type=reset value="取消">
</p>
</form>
</body>
</html>
```

浏览效果如图 7.7 所示。

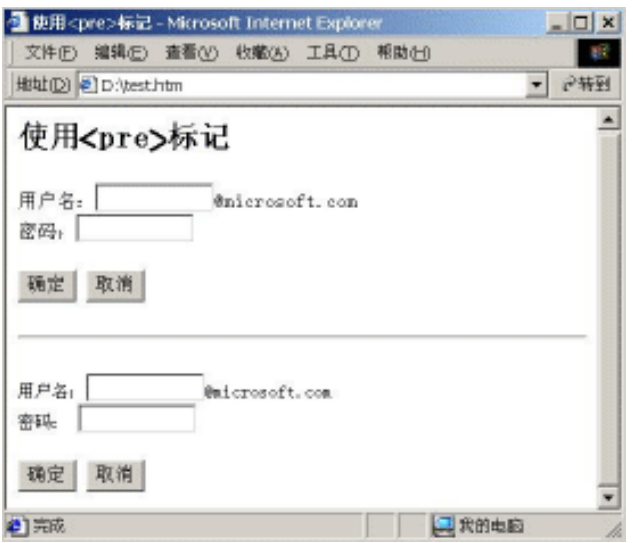

图 7.7　使用<pre>标记

注意：要在框名和文本输入框之间留出空格。应使用空格，而不应用制表符。

7.5.2 使用列表标记

表单设计的一个小技巧就是使用列表标记——特别是<ol>、<ul>和<dl>——来建立表单组织。几乎任何一种表单对象都可以是列表的一部分。建立缩进的列表或大纲格式有助于使通讯变得更好一点，而且还可以使表单外观显得更好些。另外，有时由于某种原因需要使表单对象带编号，而有序列表的<li>标记会出现序号，所以就可以简单地把表单对象叠加起来使用，从而建立一个带编号的表单。

下面这段程序给出了使用列表标记的表单的示例。

```
<html>
<head>
<title>使用列表标记的表单</title>
</head>
<body>
<h2>使用列表标记的表单</h2>
<hr>
<form action=/cgi-bin/comment_script method=post>
  <dl>
    <dt><h3>请选择语种：</h3>
    <dd><input type=radio name="语种" checked>汉语<p>
    <dd><input type=radio name="语种" value="第 1 外语">英语<p>
    <dd><input type=radio name="语种">法语<p>
    <dd><input type=radio name="语种">俄语<p>
  </dl>
</form>
<hr>
<form action=/cgi-bin/comment_script  method=post>
<h3>请输入您曾经阅读过的名著：</h3>
<ol>
  <li><input type=text name="名著" size=20>
  <li><input type=text name="名著" size=20>
  <li><input type=text name="名著" size=20>
  <li><input type=text name="名著" size=20>
</ol>
</form>
</body>
</html>
```

浏览效果如图 7.8 所示。

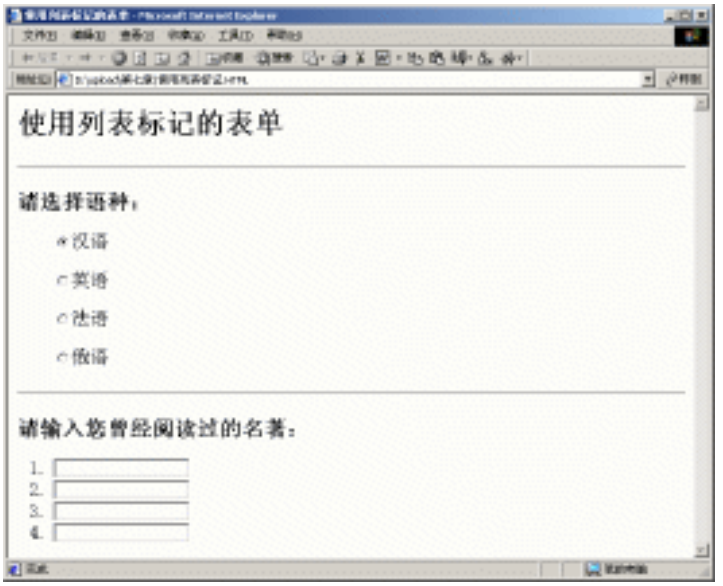

图 7.8　使用列表标记的表单

7.6　小 结

HTML 表单是给 Web 站点增加交互功能的强有力的工具。它们可用于发出信息、作出响应、确定成员资格甚至接受来自用户的产品定单，也可以作为数据检索的界面。本章介绍了建立表单的几个标记和基本属性。表单的基本元素是<form>标记本身以及许多不同类型的表单对象。其中包括<input>、<select>和<textarea>。<textarea>用于建立一个形式相对自由的文本输入框，供用户输入注释、消息和其他反馈。<input>允许用户进行多种不同的交互作用，其中包括单行文本输入框、单选按钮、复选框以及发送表单数据的特殊按钮。<select>通过提供访问列表，允许控制用户的响应方式，可以是下拉式列表或滚动式列表，使用户能够进行单项选择或多项选择。

下一章将介绍如何在 HTML 中使用多媒体对象。

第 8 章　多媒体对象的嵌入

一般来说，多媒体实视觉和声音效果，可以是图片、文字和声音的组合。本章将主要介绍如何在 HTML **文档中嵌入多媒体对象。**

8.1　嵌入多媒体文本

<embed>**标记用以插入各种多媒体文本，格式可以是** MIDI、WAV、AIFF、AU **等，其参数设定很多。该标记的基本语法如下：**

```
<embed src=#> #=URL
```

例如:

```
<embed src="your.mid" autostart=true loop=true hidden=true>
```

其中：

- src：**设定** MIDI **文件及路径，可以是相对或绝对路径。**
- autostart：**是否在音乐文件传完之后，就自动播放音乐。**true **表示是，**false **表示否（默认值）。**
- loop：**是否自动反复播放。**true **表示是，**false **表示否。**
- hidden：**是否完全隐藏控制画面，**true **表示是，**no **表示否（默认值）。**
- starttime：**设定歌曲开始播放的时间。如** starttime="00:30"**表示从歌曲的第** 30 **秒处开始播放。**
- volume：**设定音量的大小，数值在** 0 **到** 100 **之间。**
- width **和** high：**整数，设定控制画面的宽度和高度（当** hidden=no **时）。**
- align：**设定控制画面和旁边文字的对齐方式，其值可以是** top，bottom，center，baseline，left，right，texttop，middle，absmiddle **或** absbottom。
- controls：**设定控制画面的外观。有以下几个选项：**
  - console：**一般正常的面板。**
  - smallconsole：**较小的面板。**
  - playbutton：**只显示播放按钮。**
  - pausebutton：**只显示暂停按钮。**
  - stopbutton：**只显示停止按钮。**
  - volumelever：**只显示音量调整按钮。**

默认值为 console。

8.2 背 景 音 乐

<bgsound>标记用来插入背景音乐，但只适用于 IE，其参数设定不多。
<bgsound>标记的基本格式如下例所示：

```
<bgsound src=url autostart=true loop=infinite>
```

其中：

- src：设定 MIDI 文件及其路径，可以是相对或绝对路径。
- autostart：表示是否在音乐读入完之后自动播放音乐。true 表示是，false 表示否（默认值）。
- loop：是否自动反复播放。如 loop=2 表示播放两次，loop=infinite 表示重复播放无限多次。

8.3 视 频 播 放

播放视频用到<img>标记，该标记实现了几种常见的视频播放控制。

8.3.1 插入视频剪辑

其基本格式如下：

```
<img src="url.gif" dynsrc="url.avi">
```

其中：

用 url.avi 这一 AVI（Video for MS Windows）文件来播放视频。
用 url.gif 这一 GIF 图象作为视频的封面，即在浏览器尚未完全读入 AVI 文件时，先在 AVI 播放区域显示该图象。例如：

```
<img src="sample-s.gif" dynsrc="sample-s.avi">
```

8.3.2 控制何时开始播放

其基本格式如下：

```
<img start=#> #=fileopen,mouseover
```

默认值是#=fileopen，即在链接到含本标记的页面时开始播放 AVI 文件。
mouseover 是指把鼠标移到 AVI 文件播放区域之上时才开始播放。
也可以两者同时设置。例如：

```
<img start=fileopen,mouseover>
```

另外，用鼠标在 AVI 文件播放区域单击一下也将令浏览器开始播放该 AVI 文件。例如：

```
<img src="sample-s.gif" dynsrc="sample-s.avi" start=mouseover>
```

8.3.3 添加控制条

添加控制条的基本格式为：

```
<img controls>
```

实现的主要功能是在视频窗口下附加 Windows 的 AVI 文件播放控制条。例如：

```
<img src="sample-s.gif" dynsrc="sample-s.avi" controls>
```

8.3.4 循环播放

控制循环播放的基本格式为：

```
<img loop=#>
```

例如：

```
<img src="sample-s.gif" dynsrc="sample-s.avi" loop=3>
```

如果 loop=infinite 将循环播放视频文件。

8.3.5 延时

实现延时的格式为：

```
<img loopdelay=#> #=毫秒数
```

例如：

```
<img src="sample-s.gif" dynsrc="sample-s.avi" loop=3 loopdelay=250>
```

8.4 小 结

本章简单介绍了如何在 HTML 文档中嵌入多媒体对象。读者只需参照本章所给出的实例，很容易就能掌握。

下一章将介绍关于 HTML 的一些补充知识。

第 9 章　HTML 的其他知识

本章将介绍关于 HTML 的其他一些补充知识，包括动态文件、走马灯以及<body>标记的一些属性。

9.1　动 态 文 件

所谓动态文件是指不需要用鼠标（或其他点击设备）单击任何超链接或者下达任何命令，文件本身会自动产生动态效果。本节将介绍两种技巧：动态排版和动态链接。

9.1.1　content-type 动态排版

content-type 动态排版的基本格式如下：

```
<meta http-equiv="content-type" content="text/html;charset=GB2312">
```

其中<meta>标记是处于<head>与</head>之间的标记，在上面的动态排版文件中，它分为两部分，一部分是 http-equiv="content-type"，另一部分是 content="text/html;charset=GB2312"。其中 charset=GB2312 表示动态排版最终使用简体字符显示。

下面这个程序段定义了一个名为“动态排版”的动态文件。

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=GB2312">
<title>动态排版</title>
</head>
<body>
昨夜月圆了，大概又十五了吧。淡淡的月光柔柔地亲吻着我的脸，一种淡淡的温柔，一点淡淡的寂寞。荷塘的月色很美，可就是有那么一点凄凉，尤其对于现在的我。多少个月圆之夜了，静静地看着朦胧的月光，体味着那份难得的寂寞的美丽。
</body>
</html>
```

将该程序段保存为文件 1.html，并用 IE 打开，如图 9.1 所示。可以随意改变浏览器的视窗大小，此时会发现文字随着视窗大小的改变而改变其行列数，如图 9.2 所示。

某些浏览器（例如 IE 5.5）本身就具有自动进行文字排版的功能，这样 content-type 动态排版就不太有用。但是在本身不具有自动进行文字排版功能的浏览器中，content-type 动态排版就非常有用了。例如在 Navigator 中，其本身的自动排列功能是依据文件中

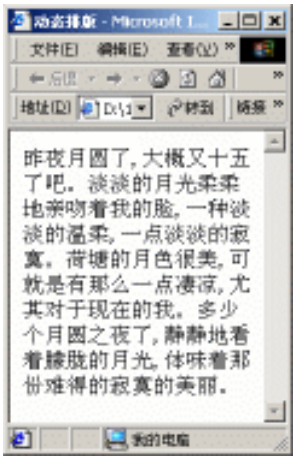

图 9.1 在 IE 中打开文件

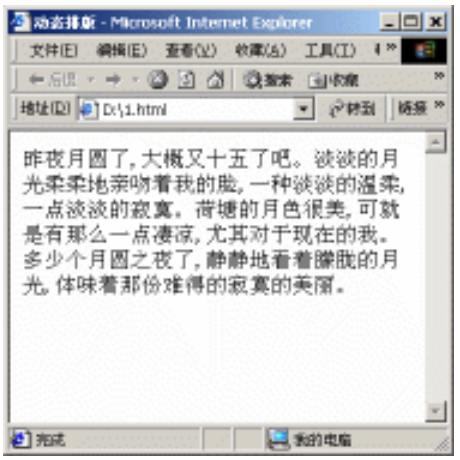

图 9.2 随 IE 视窗的改变进行动态排版

的空白字符的地址及窗口大小进行调整的，而动态排版是按照每个字来进行排列的。但是当 Navigator 的视窗缩小到一定程度时，content-type 动态排版将会失去作用。

将下面的程序段保存为文件 2.html，并用 Navigator 打开，如图 9.3 所示。

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=US">
<title>content-type</title>
</head>
<body>
I realize that admission into your school is highly competitive,
but I know that I am highly qualified,eager and prepared to meet
all of the challenges I will undoubtedly be presented with please
open your doors to me so I can open the doors of the International
Human Resource Management to China.
</body>
</html>
```

调整浏览器的视窗大小，文字会自动进行排版，如图 9.4 所示。但是当视窗缩小到一定程度时，动态排版将不起作用，如图 9.5 所示。

9.1.2 refresh 动态链接

refresh 动态链接的基本格式如下：

```
<meta http-equiv="refresh" content="时间（秒）;url=文件名或者网址">
```

这里，<meta>标记也分为两部分，一部分是 http-equiv="refresh"，用来指定动态链接或者自动更新到别的文件或者网址，另一部分是 content="*时间（秒）*;url=*文件名或者网址*"，用来指定自动更新时间和链接的目标文件。

图 9.3　在 Navigator 中打开文件

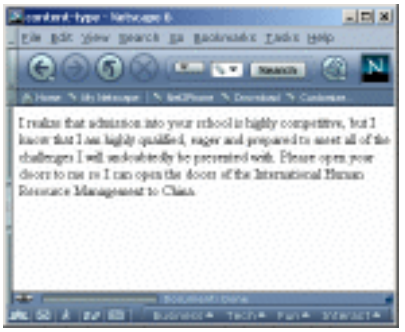

图 9.4　随 Navigator 视窗的改变进行动态排版

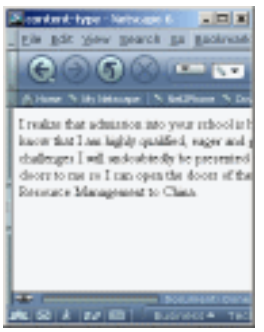

图 9.5　动态排版不起作用

下面这个程序段定义了一个名为“动态链接”的动态文件。

```
<html>
<head>
<meta http-equiv="refresh" content="5;url=4.html">
<title>动态链接</title>
</head>
<body>
<h3>5 秒钟后自动打开 4.html。</h3><br>
</body>
</html>
```

将该程序段保存为文件 3.html。文件 4.html 的程序段如下：

```
</html>
<head>
<title>动态链接测试</title>
</head>
<body>
<h3>您已经自动连接到 4.html<br>
请单击<a href=3.html>
back</a>
```

```
回到 3.html<br></h3>
</body>
</html>
```

打开文件 3.html，**如图** 9.6 **所示。**5 **秒钟后，自动打开** 4.html，**如图** 9.7 **所示。**

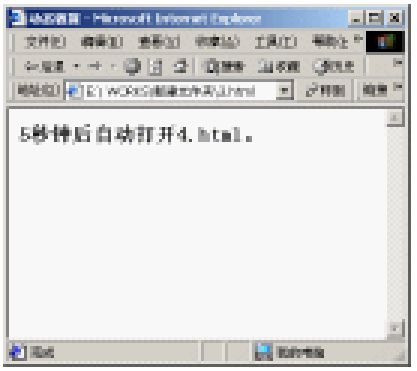

图 9.6 **打开文件** 3.html **后给出提示**

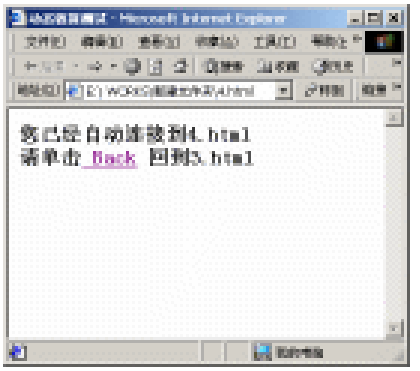

图 9.7 **自动进行动态链接**

9.2 其 他 标 记

本节将介绍两个比较特殊的标记，它们并非 HTML 4.0 **正式支持的标记。内容包括走马灯及**<body>**标记的其他属性。**

9.2.1 走马灯

走马灯为一排会自动移动的字，实现走马灯的效果需要用到<marquee>**标记，只有** IE **支持走马灯。走马灯的格式如下：**

```
<marquee scrolldelay=n1 scrollamount=n2 loop=n3 height=n4 width=n5
  hspace=n6 vapace=n7 behavior=n8 direction=n9 bgcolor=n10 align=
  n11>走马灯内容
</marquee>
```

走马灯各属性介绍见表 9.1。

表 9.1 **走马灯各属性及其说明**

属性	说明
scrolldelay	设置文字移动的速度，单位为千分之一秒
scrollamount	设置单位时间内走马灯文字移动的水平距离，单位为像素。设置较大的值，就会产生跳动的效果
loop	设置重复显示的次数，若 loop=-1 或 infinite 表示重复无限次
height	设置走马灯的高度，单位为像素
width	设置走马灯的宽度，单位为像素

（续表）

属性	说明
hspace	设置走马灯的左右边界，单位为像素
vapace	设置走马灯的上下边界，单位为像素
behavior	设置走马灯的移动方式，共有 3 种可供选择：scroll（同一方向）、slide（滑动）和 alternate（先向一个方向移动，再向另一个方向移动）
direction	设置走马灯的移动方向。共有两种可供选择：right（从右向左）和 left（从左向右）
bgcolor	设置走马灯的背景颜色，例如 bgcolor="#FF0000"代表红色背景
align	设置走马灯旁边的文字对齐走马灯的位置，共有 3 种可供选择：top（上对齐）、middle（中间对齐）和 bottom（底部对齐）

下面这个程序段定义了一个名为“走马灯”的走马灯文件。

```
<html>
<head>
<title>走马灯</title>
</head>
<body>
<font size=+2>
<marquee scrolldelay=10 scrollamount=5 loop=100 height=40 width=400
  hspace=30 vapace=10 behavior=alternate direction=right
  bgcolor="#FFFFFF" align=bottom>走马灯◎◎走马灯
</marquee>
</font>
走马灯为一排会自动移动的字，只有 IE 支持走马灯。<p>
</body>
</html>
```

将该程序段保存为文件 5.html，浏览效果如图 9.8、图 9.9 所示。

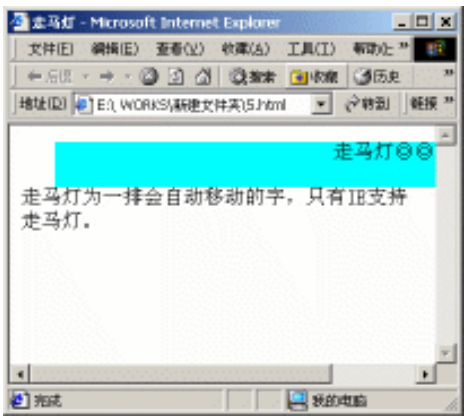

图 9.8　走马灯（a）

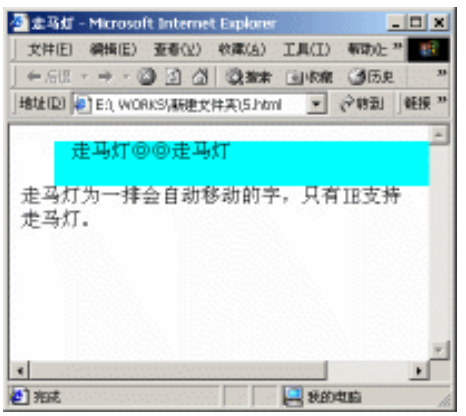

图 9.9　走马灯（b）

9.2.2 <body>标记的一些属性

下面介绍一下<body>标记的 bgproperties、leftmargin 和 topmargin 这 3 个属性。不过只有 IE 支持这 3 个属性，Navigator 不支持。

1. bgproperties 属性

bgproperties 属性可以固定网页的背景图像，其格式如下：

```
<body bgproperties=fixed>
```

下面为一个示例程序段。

```
<html>
<head>
<title>固定网页的背景</title>
</head>
<body background="1back.gif" bgproperties=fixed>
<basefront size=3>
<img src="1.gif" align=right width=300>
我的绿色漫过房顶<p>
漫过旗帜<p>
漫过呐喊<p>
我的绿色摇曳呼啸<p>
猎人的马车滚过雷霆<p>
我的绿色在田野赤足奔跑<p>
<p>
我的姐姐带着我<p>
遇见雨水 这是许多个奇迹<p>
它们说秋天会收获<p>
我说：“再见 我们要继续奔跑”<p>
我们还会遇见秋天和冬天 我知道<p>
但是 绿色你好你不要忧伤<p>
我们都是孩子 我们都是精灵<p>
我们不要忧伤 我们只要奔跑<p>
我们不要蹲下系鞋带<p>
我们会摔交 我们要拥抱<p>
我们还要飞奔<p>
我们还要遇见所有绿色的奇迹 我的兔子<p>
原来你是一阵风<p>
</body>
</html>
```

将该程序段保存为文件 6.html，浏览效果如图 9.10、图 9.11 所示。

图 9.10　使用 bgproperties 属性固定背景图（a）

图 9.11　使用 bgproperties 属性固定背景图（b）

其中 1back.gif 为背景图像，移动滚动条，可以看到只有前景的图像和文字在动，背景图像不动。

2. leftmargin 属性

这个属性用于设置显示文件内容时文件左端与浏览器左端的距离，单位为像素。

3. topmargin 属性

这个属性用于设置显示文件内容时文件上端与浏览器上端的距离，单位为像素。

以下为一个使用 leftmargin 和 topmargin 属性的示例程序段。

```
<html>
<head>
<title>leftmargin 和 topmargin</title>
</head>
<body background="1back.gif" bgproperties=fixed leftmargin=50
  topmargin=50>
<basefront size=3>
<img src="1.gif" align=right width=300>
我的绿色漫过房顶<p>
漫过旗帜<p>
漫过呐喊<p>
我的绿色摇曳呼啸<p>
猎人的马车滚过雷霆<p>
我的绿色在田野赤足奔跑<p>
<p>
我的姐姐带着我<p>
遇见雨水 这是许多个奇迹<p>
它们说秋天会收获<p>
我说：“再见 我们要继续奔跑 ” <p>
我们还会遇见秋天和冬天 我知道<p>
但是 绿色你好你不要忧伤<p>
```

```
我们都是孩子 我们都是精灵<p>
我们不要忧伤 我们只要奔跑<p>
我们不要蹲下系鞋带<p>
我们会摔交 我们要拥抱<p>
我们还要飞奔<p>
我们还要遇见所有绿色的奇迹 我的兔子<p>
原来你是一阵风<p>
</body>
</html>
```

将该程序段保存为文件 7.html**，浏览效果如图** 9.12 **所示。**

图 9.12 **使用** leftmargin **和** topmargin **属性**

9.3 小 结

作为本书 HTML **部分的结束，本章介绍了** HTML **的一些很有趣的补充知识。由于篇幅有限，只是介绍了** HTML **的基础知识，还有很多** HTML **的高级应用需要读者不断地学习。当然，对于初级读者，学习完本书介绍的** HTML **部分的内容，足以制作出漂亮的个人网页了。**

第 2 部分　JavaScript

第 10 章　JavaScript 的基础知识

本章将介绍 JavaScript 的一些基础知识，包括 JavaScript 的基本概念、工作原理和开发工具等。

10.1　JavaScript 简介

JavaScript 是 WWW 上一种功能强大的编程语言。用 Sun 公司的话来说，“JavaScript 是一种易于使用的对象描述语言，它是为了创建真正的联机应用程序而设计的，无论在客户端还是服务器端，该种应用程序都将对象和资源链接在一起。HTML 主页作者以及企业应用程序开发人员可以使用 JavaScript 动态描述在客户机或服务器上运行的对象的行为。”

10.1.1　什么是 JavaScript

JavaScript 的前身称为 LiveScript，它是 Netscape 公司为了进一步扩充其浏览器的功能而开发的一种可以嵌入 Web 页面中的脚本语言。后来，Sun 公司开发的 Java 语言的流行促使 Netscape 公司重新设计了 LiveScript，并改名为 JavaScript。之所以叫 JavaScript，原因在于 JavaScript 作为一种嵌入 HTML 文档、基于对象的脚本设计语言，其语法和 Java 语言极为相似。

JavaScript 为 Web 开发人员提供了极大的灵活性和控制手段。用户专用的内容、增强的可视化显示以及与浏览器插件的无缝集成使得 JavaScript 成为一种非常优秀的 Web 粘合剂，它可以将一个 Web 节点中的不同组成部分捆绑成为一个结合紧密的信息源。

10.1.2　为什么选用 JavaScript

首先，JavaScript 是一种解释性的语言，就是说，并不需要对 JavaScript 程序进行预编译来产生可执行代码。客户端浏览器在接收到嵌有 JavaScript 程序的 HTML 文档时，由一个内置于浏览器中的 JavaScript 语言解释器将源代码动态地处理成可执行代码。因此，从开发者的角度来看，JavaScript 语言比其他编译型语言更加简单易用。

其次，随着访问 WWW 的人越来越多，应该寻找办法减少客户端对服务器系统的处理请求，以免服务器系统负荷过重。例如一个在线购物系统，在用户将数据提交给服务器之

前，有必要对其填写的表单进行验证，如是否填写了姓名、付款方式等，这种验证应该在客户端完成。采用客户端的 JavaScript 脚本语言，就可以有效地解决这一问题。

此外，利用 JavaScript 语言可以很方便地操纵各种浏览器对象，从而使网页的界面更加友好。

10.1.3 JavaScript 和 Java

虽然 JavaScript 和 Java 很相像，但两者还是存在着很大的差异。Java 可以用于设计独立的应用程序，同时还可以用于创建称为 Applet 的小应用程序。另外，现在很多浏览器只支持 Java 而不支持 JavaScript。从目前的趋势来看，支持 JavaScript 的浏览器将越来越多。

Java 应用程序是编译运行的，而 JavaScript 脚本则是解释运行的。两者的开发工具不一样，而且使用这两种语言的用户也有很大的区别。JavaScript 实际上是为非程序员设计的，这使得 JavaScript 易于使用且不需要用户懂得太多的详细知识，如变量类型的声明等。

JavaScript 和 Java 之间的主要区别可以概括如下：

- JavaScript 是基于对象的，它有自己的内置对象；而 Java 则是面向对象的，并且对象必须在类中创建。
- JavaScript 的代码以字符的形式嵌入 HTML 文档中；而 Java 小应用程序则是由文档引用的，其代码以字节代码的形式保存在另一个独立的文档中。
- JavaScript 使用隐式数据类型，即变量可以不声明其类型，所以一个用于表示字符串的变量也可以用于表示数字；而 Java 则采用显式数据类型，即在使用变量前必须声明变量，且一个变量只能表示一种类型的数据。
- JavaScript 采用动态联编，即对象的引用只有在运行时才检查；而 Java 则是采用静态联编，即在程序中使用的对象在编译时已经存在。

10.2 JavaScript 的工作原理

如前所述，JavaScript 是一种解释性语言，它不需要预先进行编译，而是由浏览器内置的 JavaScript 语言解释器解释执行的。客户端浏览器在接收到嵌有 JavaScript 程序的 HTML 文档后，由解释器对 JavaScript 程序进行解释，然后将结果显示在浏览器中。Navigator 和 IE 的较新版本都可以很好地支持 JavaScript 语言。

在 Netscape 公司发布了 JavaScript 之后，Microsoft 公司基本独立地开发了自己的 JavaScript 语言实现版本，即 JScript。JScript 兼容了 JavaScript 的全部语法。同时，JScript 还结合 IE 浏览器，增加了许多新的功能。

> 提示：由于 JScript 和 JavaScript 很大程度上的兼容性，本书中绝大多数的 JavaScript 程序也可以改成 JScript 程序。

为了理解 JavaScript 的工作原理，下面举一个用 JavaScript 程序在网页上显示一段文字的例子，代码如下：

```
<html>
<head>
<title>
理解 JavaScript 的工作原理
</title>
</head>
<body>
<script language="JavaScript">
<!--
document.write("这是用 JavaScript 程序输出的句子。");  //输出语句
-->
</script>
</body>
</html>
```

将上面的代码保存为文件 simpleEx.html，就可以用 Web 浏览器来查看了。双击该文件的图标，或者在浏览器的地址栏中输入该文件的地址，可以看到如图 10.1 所示的结果。

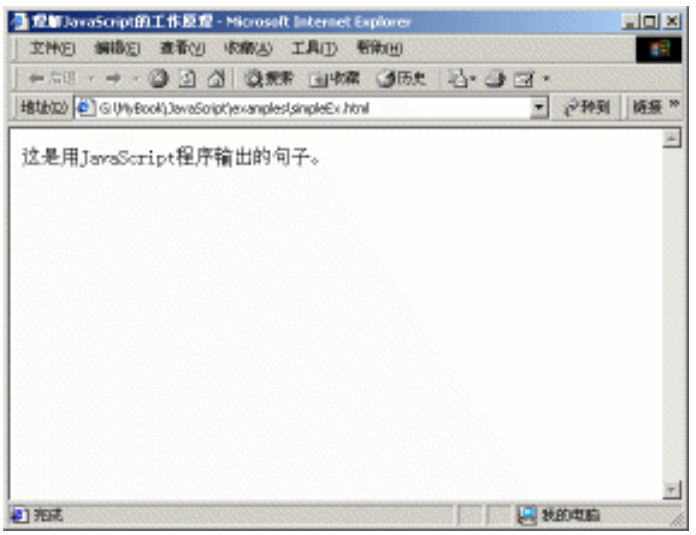

图 10.1　理解 JavaScript 的工作原理

在 HTML 文档中包含 JavaScript 程序，通常是使用<script>标记。当 Web 浏览器遇到<script>标记时，就认为下面的文本是客户端脚本程序，直到遇到相应的</script>为止。language 属性表明该程序是 JavaScript 或是 VBScript 脚本。

在这个例子中，读者可能会发现 JavaScript 脚本被包含在一对<!--…-->符号中。这对符号是 HTML 中的注释符号。由于不认识 JavaScript 的 Web 浏览器会将 JavaScript 代码看作是 Web 页面内容的一部分，因此有必要将代码包含在这样一对符号中，这样做至少可以使不认识 JavaScript 的 Web 浏览器正确地显示 Web 页。

和 HTML 不同的是，JavaScript 的注释语句不是用<!--…-->符号，而是和 C++语言一样，

用//符号。//符号可以放在程序的任何位置，任何以其开头的 JavaScript 代码都被看作注释语句。

有时候为了源代码的保密和共享，并不是将整个 JavaScript 程序都包含在 HTML 文档中，而是将 JavaScript 程序单独存为以.js 为后缀的文本文件，然后利用<script>标记的 src 属性将程序代码包含进来。

下面举一个利用 JavaScript 程序弹出一个对话框的例子来说明。JavaScript 代码保存在文件 1.js 中，在 HTML 文档中，只需用 src 属性给出.js 文件的位置就可以了，代码如下：

```
<html>
<head>
<title>
src 属性
</title>
</head>
<body>
<script language="JavaScript" src="1.js">
</script>
</body>
</html>
```

其中，文件 1.js 的内容如下：

```
window.alert("欢迎光临本站");
```

这个例子的执行结果如图 10.2 所示。

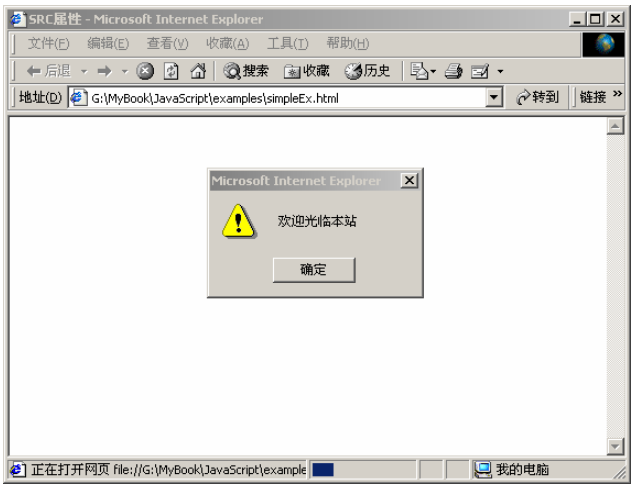

图 10.2 使用 src 属性包含 JavaScript 程序

注意：JavaScript 脚本程序是区分大小写的。例如本例的 window.alert 不能写成 Window.alert。

10.3　JavaScript 用于网页开发

JavaScript 既可以用来进行客户端的开发，也可以作为服务器端的开发语言。客户端和服务器端的 JavaScript 在语法上是相同的，但它们被设计成用来完成不同的任务。

10.3.1　客户端 JavaScript

上一节中的两个例子就是用客户端 JavaScript 开发的。在客户端和服务器端都可以运行脚本，并且很容易区分运行客户端脚本和运行服务器端脚本这两个完全不同的操作。客户端脚本是在客户机上运行的脚本程序，其运行环境是客户端的 Web 浏览器；而服务器端脚本是在服务器计算机上运行的脚本程序，其运行环境是 Web 服务器软件。

有时候不得不面对这样一个问题：到底在何处运行脚本更好，是在客户端还是在服务器端？有关这个问题的回答对于每一种脚本而言是不同的，并且也依赖于一些基本的因素。迄今为止所编写的大部分程序可能都有这样一个隐含的假设，即程序的每一行代码都在同一台计算机上运行。如果要在两台不同的计算机上运行脚本，那么 Web 应用程序就不能在单一的处理器上实现。无疑多个处理器同时完成程序的任务可以带来更高的效率。

那么，是什么因素决定哪一台计算机运行特定的脚本呢？其中一个主要的原则是：如果脚本能够在客户端运行，就应该让客户端完成。这条原则是基于这样的一些考虑：一方面，无论是客户端计算机还是服务器，都要为在 Web 服务器上执行的计算付出代价。另一方面，如果在客户端计算机上完成计算任务，客户端和服务器端都不必花费什么。基于这一经济上的考虑，每一个能够下载到客户端计算机上完成的任务都应该这样做。

第 2 个原则是，如果一个任务只能在客户端完成，那么它就必须采用客户端脚本来完成。这条原则是显而易见的。一般说来只能在客户端完成的脚本是指那些使用了仅仅在客户端浏览器中存在的内置对象的脚本。比如，用来控制用户所查看的窗口或者文档的对象。其他的脚本可能在客户端和服务器端都能够运行，那么，就可遵循这一原则来决定到底在何处运行脚本。

10.3.2　服务器端 JavaScript

使用服务器端脚本来完成某个任务的考虑首先涉及到如何使 Web 应用程序尽可能地迅速响应。一定要记住，在客户机和服务器计算机之间的 Internet 连接通常是一条很窄的通道。如果采用客户端脚本来完成某个任务意味着使这个通道充满了大量的数据，而仅仅是节省了 Web 服务器的一点点计算时间，那么 Web 应用程序的响应速度将会令人不可忍受，客户也不会感到满意。所以，数据离 Web 服务器越近就工作得越好。在 Web 服务器和客户浏览器之间保持低的数据流量，可以提高 Web 应用程序的可用性和响应速度。例如，当需要从一个大的数据库中收集信息时，尽管使用客户端脚本也可以操作一个数据库并且收集信息，但是使用服务器端脚本更好，因为只有那些关键性的有用的信息才通过 HTTP 协议传送给客户端。

第 2 个原则是，如果一个任务只能在服务器端完成，那么就必须采用服务器端脚本来完成。如果一个任务需要用到内置对象，那么该任务只能在服务器端完成，因为内置对象

只在服务器端存在。

仅仅存在于服务器环境中的内置对象，比如说 Session 对象，是一个可以在其中存入有关某个用户会话（User Session）的信息的数据对象。只要仍然在使用 Web 应用程序中的某个页面，就可以永久地保留这些信息。例如，可以用 Session 对象为用户提供一个类似于“购物小车”的特性。如果 Web 应用程序提供给客户端一个购物的场所，用 Session 对象保存在服务器上，在某个时候，客户端就可以获得一个特殊的页面，在其中列出所选择的所有对象。

使用服务器端脚本也有安全性方面的考虑。有时候可能不想向公众公开脚本代码，如果在 HTML 文档中嵌入客户端脚本，那么所有访问者都可以看到，而使用服务器端脚本则不然。所以可以将那些需要保密的脚本作为服务器端脚本，存放在 Web 服务器上。

另外，有时候在脚本的代码中可能包含了有关 Web 应用程序的关键性的信息，这些信息如果被别人窃取，就有可能对 Web 站点进行攻击。例如，如果信用卡数据库的计算机名字和数据库名字泄露，那么数据库服务器就成了外来攻击的一个明显的目标，那是非常危险的。请记住，任何客户端脚本都能够被别人查看和分析，而服务器端脚本在这一点上是足够安全的。

考虑在何处运行脚本的最后一个因素是任务的困难程度。通常客户端系统比起 Web 服务器来，是小而且慢得多的系统。所以，在使用客户端脚本的时候，要考虑到这个任务是否能够在某一可以忍受的时间内由客户端系统来完成。例如，最好不要让客户端系统来完成要使用复杂的人工智能技术才能完成的任务，因为：

- 这个任务很可能会使客户端计算机的运行速度变得很慢，并对客户端不能响应。
- 用来完成一项复杂任务的脚本很可能包含了大量的代码，这意味着大量的额外的文本代码要通过 Internet 来传送到客户端。

服务器端的 JavaScript 程序是与 Web 服务器密切相关的。下面以当前流行的 ASP（Active Server Pages）服务器 IIS 5.0 为例，介绍 JavaScript 程序在服务器端的运用。由于 VBScript 就是 IIS 服务器默认的脚本语言，因此如果要使用 JavaScript 脚本，需要设置 IIS 5.0 的脚本语言为 JavaScript（当然还有其他办法）。下面是一个在服务器端利用 JavaScript 程序输出一段话的例子。

```
<html>
<head>
<title>
服务器端 JavaScript
</title>
</head>
<body>
<script language="JavaScript" RunAt="Server">
for(i=0;i<3;i++)
  response.write("服务器端 JavaScript<br>");
</script>
```

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

在这个例子中，RunAt 属性表明该脚本是在服务器端执行的。for 语句是 JavaScript 语言中的循环语句，而 response 是 ASP 服务器端的内置对象之一。

将上面的代码保存为文本文件 server.asp，放在服务器的可执行目录下。其执行结果如图 10.3 所示。

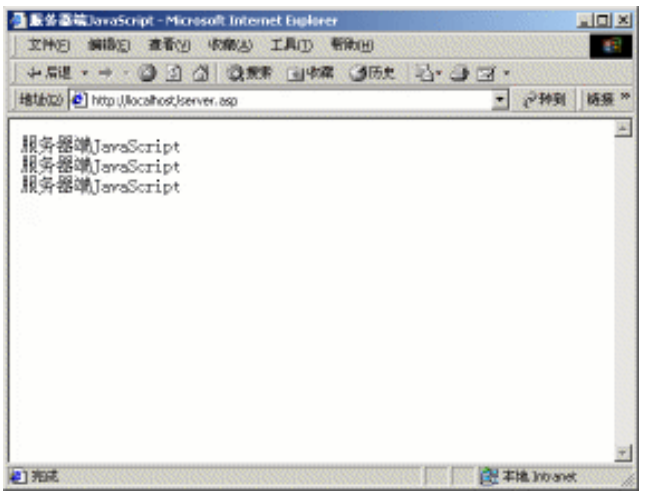

图 10.3　在服务器端使用 JavaScript

注意：关于 ASP 的更多内容，读者可以参考本书后面的有关章节或其他相关书籍。由于客户端 JavaScript 和服务器端 JavaScript 在语法上是相同的，因此后面的章节中均采用客户端 JavaScript 语言。

10.4　JavaScript 的开发工具

JavaScript 作为一种典型的脚本语言，其语法相对比较简单，程序执行的时候采用解释方式，对用户计算机的配置要求不高。也正是由于这个原因，目前为止还没有专门的 JavaScript 开发工具可以使用。但由于 JavaScript 的易用性，即使没有专门的开发工具，也可以制作出精美的网页。

由于 JavaScript 程序是文本，因此任何一个文本编辑器都可以作为 JavaScript 的开发工具，如 NotePad、UltraEdit。如图 1.4 所示是用 UltraEdit 开发 JavaScript 脚本的例子。

此外，由于 JavaScript 脚本程序往往作为网页设计的一个组件嵌套在网页中，因此一些功能较全的网页开发软件也提供一定功能的 JavaScript 脚本程序编辑功能，如 Microsoft FrontPage 98 和 MacroMedia Dreamweaver 等。

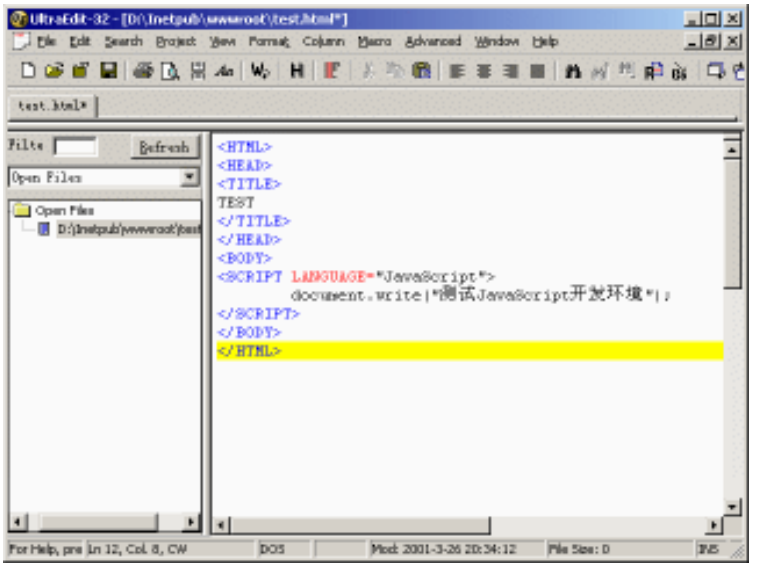

图 10.4 用 UltraEdit 开发 JavaScript 脚本

在 Microsoft 公司的 Office 2000 中，提供了一个开发网页的集成环境：Microsoft 脚本编辑器。在脚本编辑器中，可以插入 VBScript 或 JavaScript 脚本，并且提供了强大的调试功能。如图 10.5 所示是 Microsoft 脚本编辑器的界面。

在脚本编辑器中，可以跟踪程序的执行，也可以设置断点，是一个不可多得的 JavaScript 开发环境。

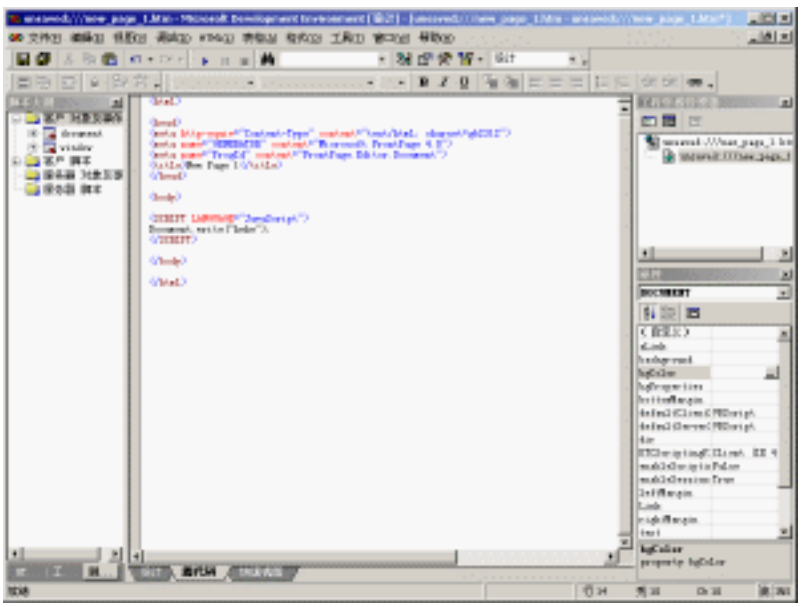

图 10.5 Microsoft 脚本编辑器

10.5　小 结

本章介绍了 JavaScript 语言的基本知识、工作原理和开发工具。读者在学习完本章后，可以对 JavaScript 语言有一个初步的认识，并能编写简单的 JavaScript 脚本程序。

下一章将系统地学习 JavaScript 的基本语法。

第 11 章　JavaScript 的基本语法

作为一种 Web 开发语言，JavaScript 既有一般编程语言的共同点，也有其独特之处。

本章将介绍 JavaScript 语言的基本语法，包括 JavaScript 的程序结构、数据类型、变量和表达式、操作符和流程控制等。

11.1　JavaScript 程序的结构

JavaScript 是基于 WWW 的面向行为模型的。Web 页的对象，如一个按钮或复选框，都可以用于触发行为或事件。当发生了某个事件时，会执行某段相应的 JavaScript 程序代码，通常是个 JavaScript 函数。而这个函数是由各种操作计算、检查或修改某个 Web 页的内容，或操作各种响应的任务的语句组成。

JavaSript 程序的元素可以通常分为以下 5 大类：

- 变量和常量
- 操作变量和值的表达式
- 改变语句操作的控制函数
- 执行一个语句块的函数
- 将相关数据组织在一起的对象和数组

JavaScript 语言元素的这种分类与大部分编程语言非常相似。JavaScript 还提供了许多可以大大简化程序组织的面向对象的编程结构。此时，JavaScript 与 C 或 Java 有点相似。

如下所示为一个典型的嵌有 JavaScript 脚本程序的超文本文件。

```
<html>
<head>
<title>
test
</title>
</head>
<body>
<script language="JavaScript">
function myAlert( ){
  var mystring="这是 JavaScript 定义的变量";    //定义变量
  window.alert(mystring);
}
</script>
```

```
<input type="button" value="执行" onClick="myAlert( )">
</body>
</html>
```

这个例子中，在 JavaScript 脚本中定义了一个函数 myAlert。在网页中有一个按钮对象，当单击该按钮时，执行相应的 JavaScript 函数 myAlert，结果如图 11.1 所示。

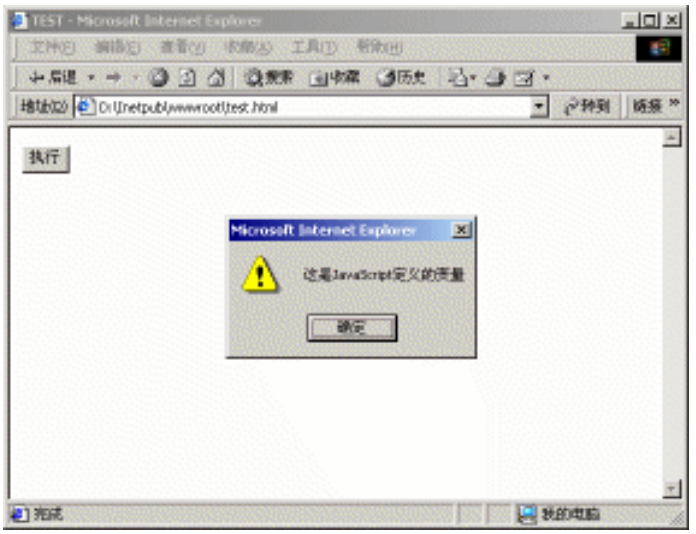

图 11.1　理解 JavaScript 的程序结构

提示：JavaScript 脚本程序可以放在超文本文件中的任何位置。可以在<head>...</head>中，也可以在<body>...</body>中，甚至可以在<html>...</html>外面。

11.2　JavaScript 的数据类型

一般来说，数据类型是学习任何语言的起点。在有些语言中，在声明变量时就必须同时声明该变量的数据类型，以便让计算机知道如何为该变量分配内存空间，而在 JavaScript 中却并非如此。

在 JavaScript 中，可以使用以下 5 种数据类型：

1. 数字

JavaScript 在语言实现的时候并没有把整数和实数严格分开，两者在程序中可以自由地转换。在 JavaScript 中还有一个特别的数字量 NaN，用以指示一个变量或者函数的返回值是否为一个数字。

2. 逻辑值

逻辑值只有两个常量——true 和 false。

3. 字符串

字符串类型是由单引号或双引号界定的一串字符。

4. undefined 类型

undefined 类型专门用来指明一个已经创建但还没有赋初值的变量。

5. 对象

对象是 JavaScript 中的重要组成部分。可以把对象看成一个已经命名的容器，可以容纳数据和对这些数据进行的操作。

可以看到，与其他的编程语言相比，JavaScript 具有简单的数据类型。在声明一个变量时，不必把它声明成一个固定的数据类型。变量的类型是根据它的当前值而改变的。如果在表达式中变量的当前类型和表达式的要求不一致，则 JavaScript 会自动完成相应的转换。

提示：所有的变量转换总是以表达式最左边的变量类型为准，其他出现在表达式中的变量均按照这个类型进行转换和运算。

以下程序段将一个数据类型强制转换为字符类型。

```
var theFrom=1;
var theTo=10;
var doWhat="Count from";
doWhat+=theFrom+"to"+theTo+".";
```

执行该代码后，变量 doWhat 的值为"Count from 1 to 10."。其中的数值数据被转换为字符串格式。

当被转换字符中包括数值类型和字符串类型时，一般系统会自动将数值类型转换成字符串类型，然后再进行相应的运算，如下面的程序段所示。

```
var nowWhat=0;
nowWhat+=1+"10";
```

在本例中，"10"是一个字符串，而+=运算符起连接作用。执行该代码后，nowWhat 变量的值为"0110"。该程序的执行步骤为：

（1）查看 1 和"10"的类型。"10"为一个字符串，1 为数值类型，因此该数被强制转换为字符串"1"。

（2）由于+运算符两边的值都是字符串，因此执行字符串连接操作，其结果为"110"。

（3）查看+=两边的值的类型。nowWhat 包含一个数，而"110"是字符串，因此将数转换为一个字符串。

（4）由于现在+=运算符两边都是字符串，因此执行字符串连接操作，其结果为"0110"。

（5）将结果存放到 nowWhat 中。

当需要将字符串类型转换成数值类型时，可以使用转换函数 parseInt 和 parseFloat。parseInt 为强制转换成整型函数，parseFloat 为强制转换成浮点型函数。以下为一个使用函

数 parseInt 进行转换的例子：

```
var nowThen=0;
nowThen+=1+parseInt("10");  //本例中+=执行加法
```

执行该代码后，字符串"10"被转换成数值 10，所以变量 nowThen 此时为整数 11。

注意：不推荐在程序中不断改变一个变量的类型。这样做的后果可能是在一段时间后连自己都看不懂原来的程序。提倡的方法是一个变量在初始化之后，其类型保持不变。

11.3　JavaScript 的变量

变量是一种使用方便的占位符，用于引用计算机内存地址，该地址可以存储脚本程序运行时可更改的程序信息。例如，可以创建一个名为 ClickCount 的变量来存储客户端单击 Web 页面上某个对象的次数。使用变量并不需要了解变量在计算机内存中的地址，只要通过变量名引用变量就可以查看或更改变量的值。

11.3.1　声明变量

尽管在 JavaScript 中不必事先声明变量，但在使用一个变量之前用 var 语句来声明变量是良好的编程习惯之一。例如：

```
var myVar;
```

虽然变量是由用户定义的，但 JavaScript 还是对变量的名字作出了如下一些限制：

- 变量名必须由字母或下划线“_”开头。第 1 个字符不能是数字或者其他非字母表字符。
- 变量不能包含空格。
- JavaScript 是区分大小写的。例如，A 和 a 是不同的名字。JavaScript 的内置语句都是小写的。
- 不能使用保留字作为变量名。保留字是已被 JavaScript 用作特定目的的字符。例如用户不能将 if 或 for 等作为变量名，因为这些字符是 JavaScript 的保留字。
- 变量在被声明的作用域内必须惟一。

例如，下面的变量声明语句都是正确的。

```
var text="This is a string";  //字符串变量
var myScore=90;  //整数变量
var myBool=true;  //布尔型变量
var avScore=85.213  //浮点型变量
```

注意：有时候声明一个变量的时候并不想给它赋初值，这时候可以给它赋一个特殊的常量 null。null 常量的使用非常灵活，作为数字时，它等效于 0，而作为字

符串时，它又等同于一个空字符串。

除了使用 var 语句声明变量之外，也可以直接用赋值的方法来定义变量。例如：

```
myScore=90;
myText="This is a string";
```

总之，JavaScript 语言不太注重变量的数据类型，所有变量的类型可以在运行的时候动态改变。但是变量的声明和初值的设定是一个良好的编程习惯。为了避免调试 JavaScript 脚本程序过程中出现不必要的麻烦，建议读者对使用的任何变量都事先用 var 语句声明，并尽可能赋予初值。

11.3.2 使用变量

在声明变量之后，可以对变量进行赋值等操作。同时，在使用变量的时候，要注意变量的作用域和数据范围。

1. 变量的作用域与存活期

变量的作用域由声明它的位置决定。如果在函数中声明变量，则只有该函数中的代码可以访问或更改变量值，此时变量具有局部作用域并被称为局部变量。如果在函数之外声明变量，则该变量可以被脚本（Script）中所有函数所识别，称为 Script 级变量（全局变量），具有 Script 级作用域。

变量存在的时间称为存活期。Script 级变量的存活期从被声明的一刻起，直到脚本运行结束。对于局部变量，其存活期仅是该过程运行的时间，该过程结束后，变量随之消失。在执行过程时，局部变量是理想的临时存储空间。可以在不同过程中使用同名的局部变量，这是因为每个局部变量只被声明它的过程所识别。

声明变量时，局部变量和全局变量可以有相同的名称，改变其中一个的值并不会改变另一个的值。如果没有声明变量，则可能不小心改变了一个全局变量的值。例如，以下脚本命令返回值为 1，虽然有两个名为 Y 的变量。

```
<script language="JavaScript">
var Y=1;
SetLocalVariable();
document.write(Y);
function SetLocalVariable( ){
  var Y=2;
}
</script>
```

以下的脚本没有显式地声明变量，将返回 2。当过程调用将 Y 设置为 2 时，脚本引擎认为该过程是要修改全局变量。

```
<script language="JavaScript">
Y=1;
SetLocalVariable( );
```

```
document.write(Y);
function SetLocalVariable( ){
  Y=2;
}
</script>
```

2．给变量赋值

前面已经讲过，在函数体外声明一个变量是可以不使用 var **语句的。只要给变量赋一个初值就可以代替声明这个变量，**JavaScript **解释器可以自动知道这个变量的类型以及目前的数值。尽管如此，还是推荐使用** var **来声明每一个用到的变量。**

给变量赋值时，就是创建这样形式的表达式：变量在表达式的左边，要赋的值在表达式的右边。例如：

```
var str="This is a string.";
str="The string has been changed.";
```

11.4　JavaScript 的表达式

一个表达式就是由任何常量、变量和操作符连接而组成的式子，这个式子可以惟一地得出一个值。这个值可以是前面提到的任何一种数据类型。

JavaScript **的表达式是以分号（;）结束的，表示在程序执行下条语句之前，当前语句的开始与分号之间的内容必须执行完毕。这一点和传统的** C/C++**语言是一样的，不同的是，如果在** C++**语句的末尾忘记了分号，编译程序时会出错，**JavaScript **则不会这样。**JavaScript **程序可以书写得很松散，如果忘记写分号，编译系统会默认这行语句末尾有一个分号。尽管忘记分号不会出现编译错误，但是这决不代表可以废弃分号不用。风格良好的程序总是在一行的末尾带有分号（当然，有些条件语句和循环语句除外）。**

在一条语句后面我们常常加上一些注释语句，以便别人容易读懂程序。JavaScript **提供了丰富的注释符号，包括有** C **语言中使用的/*…*/，**C++**中的//和** HTML **的<!--…-->等。和在** C **和** C++**语言中一样，在/*…*/中放置注释可以跨越多行，而//可以把注释放在//与当前行的末尾之间。**

值得一提的是 JavaScript **中的条件表达式，它可以根据不同的条件得到两个不同结果中的一个。它的形式如下：**

条件? **表达式** 1: **表达式** 2;

其中，条件是一个逻辑表达式，它必须为 true **或** false**。如果是** true**，那么表达式的结果为表达式** 1**；如果是** false**，那么表达式的结果为表达式** 2**。例如：**

```
var a=4;
var b=3;
var str=(a>b)? "a 比 b 大" : "b 比 a 大";
```

最后结果为 str="a **比** b **大**"**。**

11.5 JavaScript 的运算符

JavaScript 的运算符和 C 语言中的运算符几乎完全一致。按照运算符处理的对象的不同类型，可以把 JavaScript 的运算符分为赋值运算符、算术运算符、逻辑运算符、位运算符和字符串运算符等。

11.5.1 赋值运算符

与其他所有的程序语言一样，JavaScript 提供赋值运算功能，可以把数值存入变量。JavaScript 中使用等号（=）来表示赋值操作，下面的例子可以用来说明赋值运算的基本格式：

```
anInteger=3;
```

除了这种一对一的赋值外，还可以进行一连串的赋值，即多个变量同时被赋予相同的值。如下例所示，例中变量 x、y 和 z 都被赋值为 6。

```
x=y=z=6;
```

表达式中的一连串赋值运算的顺序为从右到左。所以在上例中首先把变量 z 赋值为 6，然后变量 y 被赋值为变量 z 中存储的数值 6，最后变量 x 被赋值为变量 y 中储存的数值 6。最终结果是 3 个变量都被赋值为 6。

在 JavaScript 中，赋值运算符（=）作为一种运算符，与加号（+）和减号（-）一样可以用于表达式，因此可以把变量的赋值和表达式的计算合成一步运算。例如：

```
y=(x=3)+4;
```

在这步运算中，变量 x 被赋值为 3，同时将 3 与 4 的和赋予变量 y。最终表达式运算完成后，变量 y 的值为 7。这步运算完成了 3 件事：首先给变量 x 赋值，然后作加法计算，最后给变量 y 赋值。这些特点使得 JavaScript 语言应用起来灵活而简便。

除了上述基本的赋值运算符外，JavaScript 还提供了多种高级赋值运算符，从而扩展了赋值运算的功能。所谓高级赋值运算符就是把基本赋值运算符与其他运算符结合起来而构成的运算符。表 11.1 列出了高级赋值运算符的例子及说明。

表 11.1 高级赋值运算符

运算符	例子	说明
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

（续表）

运算符	例子	说明
<<=	x<<=y	x=x<<y
>>=	x>>=y	x=x>>y
>>>=	x>>>=y	x=x>>>y
&=	x&=y	x=x&y
|=	x|=y	x=x|y
^=	x^=y	x=x^y

11.5.2　算术运算符

与其他程序语言一样，JavaScript **也提供了许多算术运算符，可以进行普通的加法、减法以及不太常见的取模和增量运算等。**

注意：所有普通算术运算符在使用时都要把字符串转换为数字。如果一个字符串不能转为数字，将返回 NaN**（不是一个数字）。**

JavaScript **的运算符和数学函数功能十分强大。这不仅因为它的内置运算符功能强大，也因为** JavaScript **的数学对象（**Math Object**）提供了一系列的高级数学运算。读者可以查找相关的手册得到数学对象的所有内部函数列表。**

1．加号

毋庸质疑，加号（+**）是最普通、应用最广泛的数学运算符。如果加号两边都是数字，计算结果就是各个数值的和。如果两边都是字符串，计算结果就是将两个字符串连起来。如下面这行代码：**

```
var resultOfAdd=34+12;
resultOfAdd="a"+"com";
```

其运行结果将变量 resultOfAdd **赋值为字符串**"acom"**。**

2．减号

减法是从减号（-**）左边的数中减去减号右边的数值。如果减号的某一侧是字符串，系统将先把字符串转换为数字，再作减法运算。例如下面这行代码：**

```
var resultOfSub=25-102;
```

其运行结果是将变量 resultOfSub **赋值为**-77**。**

3．乘号

JavaScript **的乘法与其他程序语言一样，也是把乘号（*****）两边的数相乘得到计算结果。如果乘号两边都是字符串，计算结果是把两个字符串连接起来。如果只有一边是字符串，系统将先把字符串转换为数字，再作乘法运算。例如下面这行代码：**

```
var resultOfMult=5*7；
```

其运行结果是将变量 resultOfMult 赋值为 35。

4．除号

除法运算虽然简单，却很容易出错。必须明确哪个是数，哪个是被除数。从计算式的左边读到右边，应该是除号（/）左边的数除以右边的数。如果除号某一边是字符串，系统将先把字符串转换为数字，再作除法运算。例如下面这行代码：

```
var resultOfDiv=42/7;
```

其运行结果是将变量 resultOfDiv 赋值为 6。

5．取模运算符

尽管取模运算符（%）不如其他运算符那样使用频繁，但它的出现还是令人非常兴奋。因为它是数学运算的一个进步。这种运算以除法开始，同样是运算符左边的数除以运算符右边的数，不同的是计算结果不再是商而是余数。如果运算符某一边是字符串，系统将先把字符串转换为数字，再作取模运算。例如下面这行代码：

```
var resultOfMbd=26%3;
```

其运行结果是将变量 resultOfMod 赋值为 2。

6．前增量运算符

前增量运算符（++）把程序员经常使用的两步非常普通的运算合成一步，使代码简练、易读，尤其是在循环语句中使用非常方便。前增量运算符位于变量的前边（即左边），计算结果以该变量的数值加 1 开始，返回这个新值，为其他表达式所用。如果变量的值是字符串，系统将先把字符串转换为数字，再作增量运算。例如下面两行代码：

```
var price=5;
var pricePlusShipping=(++price)+3;
```

其运行结果是将变量 price 的值由 5 增加为 6，并将变量 pricePlusShipping 赋值为 9。

7．后增量运算符

后增量运算虽然与前增量运算的运算符相同，都是++，但是它们的计算过程有很大区别。首先后增量运算符位于变量的后边（即右边），其次，后增量运算也是以该变量的数值加 1 开始。与前增量运算不同，后增量运算返回增量之前的初始值。如果变量的值是字符串，系统将先把字符串转换为数字，再作增量运算。例如下面两行代码：

```
var price=5;
var pricePlusShipping=(price++)+3;
```

其运行结果是将变量 price 的值由 5 增加为 6，并将变量 pricePlusShipping 赋值为 8 而不是 9。

8．前减量运算符

前减量运算符（--）与前增量运算符非常相似，也是位于变量的前边（即左边），执行

顺序也非常相似。有一个关键的不同之处：计算结果是该变量的数值减去 1。如果变量的值是字符串，系统将先把字符串转换为数字，再作减量运算。例如下面两行代码：

```
var price=20;
var priceMnusDiscount=(--price)-6；
```

其运行结果是将变量 price 的值由 20 减为 19，并将变量 priceMnusDiscount 赋值为 13。

9．后减量运算符

后减量运算符（--）与后增量运算符非常相似，也是位于变量的后边（即右边），执行顺序也非常相似。不同之处在于：后减量运算的计算结果是该变量的数值减 1。如果变量的值是字符串，系统将先把字符串转换为数字，再作减量运算。例如下面两行代码：

```
var price=20;
var priceMnusDiscount=(price--)-6;
```

其运行结果是将变量 price 的值由 20 减为 19，并将变量 priceMnusDiscount 赋值为 14，而不是 13。

注意：尽管 Modem 传输数据的速度飞速增长，使得下载时间不再是问题，但是尽量精简程序代码而又不损失程序清晰度仍是程序员应该注意的问题，而使用前/后增量、减量运算符取代传统的加法、减法表达式（例如 x=x+1）就是一种很好的方法。

10．一维取反运算符

在数学运算中，当一个数的值需要由正值变为负值或由负值变为正值时，就用到了一维取反运算符（-）。需要注意的是对某变量进行取反运算后，该变量的内容并不改变，只是返回的值取了负号。与其他运算符一样，如果变量的值是字符串，系统将先把字符串转换为数字，再作取返运算。例如下面两行代码：

```
var aNumber=67;
var resultOfNeg=-aNumber;
```

其运行结果是将变量 resultOfNeg 赋值为-67。

11.5.3　逻辑运算符

JavaScript 提供了 3 种逻辑运算符。如果没有这 3 种逻辑运算符，将非常复杂而冗长，乍一看程序员似乎都感觉对它们已经很熟悉了，但只有深入挖掘才能真正掌握逻辑运算，否则将会导致意想不到的错误，而且很难查找和修改。所以请仔细阅读下面各逻辑运算符的说明。

注意：JavaScript 把“真（true）”定义为 0（zero）、""（空字符串）、null、不确定数值以及“假（false）”以外的任何数。

1．逻辑与运算符

JavaScript 的逻辑与运算的执行过程为：首先判断运算符&&左边的表达式是真还是假。如果是假，则逻辑与运算到此结束，不再判断右边的表达式。如果是真，则继续判断右边表达式的真假，以决定逻辑与运算的最后结果。不论哪一种情况，逻辑与运算的最终结果都是逻辑计算的实际结果。

2．逻辑或运算符

与逻辑与运算一样，理解 JavaScript 实际如何执行逻辑或操作很重要。它首先判断运算符||左边的表达式是真还是假。如果是真，则逻辑或运算到此结束，不再判断右边的表达式；如果是假，则继续判断右边表达式的真假。不论哪一种情况，逻辑或运算的最终结果也是逻辑计算的实际结果。

3．逻辑非运算符

逻辑非运算符（!）并不像比较运算符那么复杂，其计算结果就是表达式的逻辑值的相反数。即如果表达式为真，则逻辑非运算结果为假，如果表达式为假，则逻辑非运算结果为真。如果表达式的判断结果不是逻辑数值，则先把它转换为真或假，再作逻辑非运算。

注意：逻辑运算符&&和||得出的运算结果并非都是“真”或“假”。这是因为它们的最终结果都是后判断的表达式的逻辑计算结果。所以在 Navigator 2.0 和 Navigator 3.0 中对运算符&&和||作了一些修改，不再仅返回判断表达式的结果，而是返回已转移为逻辑值的判断结果。

11.5.4 位运算符

JavaScript 中的位运算只处理 32 位整数运算，对于非 32 位字节的整数，则先把它转换为 32 位字节，再作逻辑运算。位运算符只处理每一二进制位上的 0 或 1。虽然位运算在 JavaScript 程序中不是经常使用，但在许多情况下位运算是必不可少的。

1．按位与运算符

按位与运算符（&）把&符号两边的整数当作 32 位字节数字。用本章前面所讨论的逻辑与运算符（&&）分别比较运算符两边整数的 32 位中的对应位，逻辑与运算得到的 32 位二进制结果转化为整数值，作为按位与运算的返回值。

2．按位或运算符

按位或运算符（｜）把|符号两边的整数当作 32 位字节数字。用本章前面所讨论的逻辑或运算符（||）分别比较运算符两边整数的 32 位中的对应位，按位或运算的 32 位二进制结果转化为整数值，作为按位或运算的返回值。

3．按位异或运算符

按位异或运算符（^）把^两边的整数当作 32 位字节数字。按位异或运算与按位或运算不同，它使用的是逻辑或运算的特殊版本——异或运算对两边整数的某一位字节进行比较。

注意：逻辑异或运算与逻辑或运算不同，逻辑异或运算符两边的表达式只有一个为真，计算结果才是真，如果两边的表达式都为真或都为假，则计算结果就是假。

4．按位取反运算符

按位取反运算符（~）比按位与、按位或以及按位异或运算符简单。按位取反运算中的整数以 32 位字节的形式存在。按位取反运算把整数的某位字节由 0 变为 1 或由 1 变为 0，这样处理以后的 32 位字节再转换为整数，就是按位取反的运算结果。

5．左移位运算符

左移位运算符为<<，在左移位运算中，<<左边的整数以 32 位数字的形式存在。左移位运算把<<左边整数的所有字节向左移，左移的位数由<<右边的数值决定，左移后留出的右边空位用 0 填补。由于只允许 32 位字节，左移超出的字节将丢失。这样处理以后的 32 位字节再转换为整数，就是左移位运算的运算结果。

6．带符号右移位运算符

带符号右移位运算符为>>，在左移位运算符类似。在带符号右移位运算中，>>左边的整数以 32 位数字的形式存在。右移位运算把>>左边整数的所有字节向右移，右移的位数由>>右边的数值决定。如果>>左边的数是正数，右移后留出的左边空位用 1 填补；如果>>左边的数是负数，右移后留出的左边空位用 0 填补。由于只允许 32 位字节，右边超出的字节将丢失。这样处理后的 32 位字节再转换为整数，就是带符号右移位运算的运算的结果。

7．右移后 0 填充运算符

右移后 0 填充运算符为>>>。右移后 0 填充运算与带符号右移位运算的执行过程很相似，不同之处为 32 位字节的整数右移后留出的左边空位都用 0 填补，而不考虑这个整数的正负。

11.5.5　比较运算符

JavaScript 不仅提供大多数语言中包括的普通比较运算符，还包括一些新的运算符。

1．相等运算符

相等运算符（= =）比较= =符号两边表达式的值，如果值相等，则运算结果为真，如果不相等，则运算结果为假。JavaScript 中的相等运算先把所有表达式转换为同一类型，比如= =符号左边的表达式是数字，右边的表达式是字符串，就先把字符串转换为数字，再作相等运算。

知道了在相等运算前作类型转换还不够，还需要知道如何运作，这样才能在调试程序检查错误时节省时间。类型转换遵从下列规则：

- 比较之前，先把 true 转换为 1，false 转换为 0。
- 如果= =符号两边的操作数都是非数字（NaN），则相等运算的结果为假。
- null 与不确定数（undefined）相等。
- null 与不确定数（undefined）不等于 0（zero）、""（空字符串）或假（false）。
- 如果比较字符串和数字，则先把字符串转换为数字，再作相等运算。
- 如果比较字符串和对象，则先把对象转换为字符串，再作相等运算。
- 如果比较对象和数字，则先把对象转换为数字，再作相等运算。

2．不相等运算符

不相等运算符（!=）比较!=符号两边表达式的值，如果值不相等，则运算结果为真，如果相等，则运算结果为假。与相等运算的情况一样，先把所有表达式转换为同一类型，再作不相等的比较。

3．大于运算符

大于运算符（>）比较>符号两边表达式的值，如果左边的值大于右边的值，则运算结果为真，如果左边的值小于或等于右边的值，则运算结果为假。如果有一边是字符串，则先把该字符串转换为数字，再作比较。

4．小于运算符

小于运算符（<）比较<符号两边表达式的值，如果左边的值小于右边的值，则运算结果为真，如果左边的值大于或等于右边的值，则运算结果为假。如果有一边是字符串，则先把该字符串转换为数字，再作比较。

5．大于等于运算符

大于等于运算符（>=）比较>=符号两边表达式的值，如果左边的值大于或等于右边的值，则运算结果为真，如果左边的值小于右边的值，则运算结果为假。如果有一边是字符串，则先把该字符串转换为数字，再作比较。

6．小于等于运算

小于等于运算符（<=）比较<=符号两边表达式的值，如果左边的值小于或等于右边的值，则运算结果为真，如果左边的值大于右边的值，则运算结果为假。如果有一边是字符串，则先把该字符串转换为数字，再作比较。

7．同一性运算符

同一性运算符（= = =）比较= = =符号两边表达式的值，如果值相等，则运算结果为真，如果不相等，则运算结果为假。在进行同一性运算前不作类型转换。

8．不同一性运算符

不同一性运算符（!= =）比较!= =符号两边表达式的值，如果值不相等，则运算结果为真，如果相等，则运算结果为假。在进行不同一性运算前不作类型转换。

注意：同一性运算和不同一性运算仅在 JavaScript 1.3 以后的版本中存在。

11.5.6 其他运算符

除了上面的运算符之外，JavaScript 还提供了以下几个运算符：

1．delete 运算符

这个运算符的作用是从一个对象中移去一个属性。如果对象中指定的属性存在，使用这个运算符就可以删除这个属性，如果这个属性并不存在，则返回 false 值。

由于 JavaScript 把数组也看作对象，因此使用该运算符也可以从一个数组中删除一个元素。

2．typeof 运算符

使用这个运算符可以确定一个表达式的类型。这个运算符的返回值是以一个字符串来表示的，可能的返回值为：

- "number"：要确定的表达式是一个数值类型的表达式。
- "string"：要确定的表达式是一个字符串类型的表达式。
- "boolean"：要确定的表达式是一个逻辑值表达式。
- "object"：要确定的表达式是一个对象类型。
- "function"：要确定的表达式是一个函数类型。
- "undefined"：无法确定表达式的类型。

下面的例子说明了 typeof 运算符的使用方法。

```
<html>
<head>
<title>
typeof 运算符的返回值
</title>
</head>
<body>
<script language="JavaScript">
document.writeln("\'这是字符串\'是"+typeof('这是字符串')+"类型<br>");
document.writeln("2*3 是"+typeof(2*3)+"类型<br>");
document.writeln("Math 是"+typeof(Math)+"类型<br>");
</script>
</body>
</html>
```

这个例子的执行结果如图 11.2 所示。

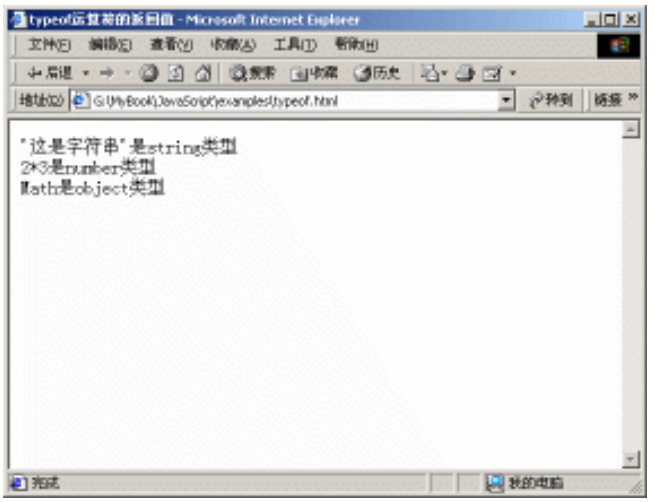

图 11.2　使用 typeof 运算符判断表达式的类型

3．= = =运算符

= = =为恒等于运算符，它在比较两个值时，只有当这两个值完全一样时，才返回 true 值。所谓“ 完全一样 ”，是指不经过 JavaScript 自动类型转换而保持相等。

4．!==运算符

与= = =运算符类似，如果在没有类型转换的情况下，接受比较的两个值不同，则返回 true 值。

下面的例子说明了= =和= = =的区别。

```
<html>
<head>
<title>
===运算符
</title>
</head>
<body>
<script language="JavaScript">
var a=3;
var b="3.0";
if(a==b)
  document.write("a==b<br>");
else
  document.write("a!=b<br>");
if(a===b)
  document.write("a===b<br>");
else
  document.write("a!==b<br>");
</script>
</body>
</html>
```

这个例子的执行结果如图 11.3 所示。

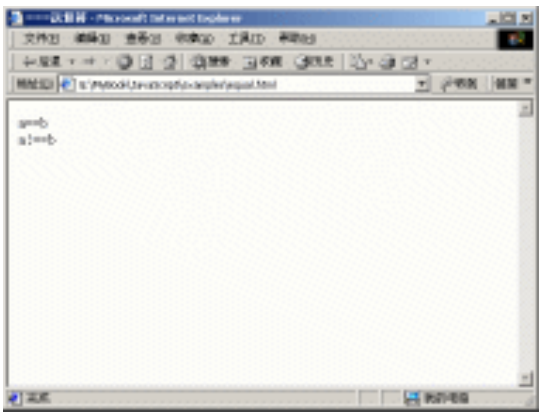

图 11.3 比较= =运算符和= = =运算符的区别

11.5.7　运算符的优先级

在实际的 JavaScript 程序表达式中，运算符的执行是按照一定的顺序进行的。这样的执行顺序称为运算符的优先级。优先级高的运算符先进行运算。如果需要优先级比较低的运算符先计算，可以使用括号，因为括号的优先级最高。

表 11.2 按照优先级从最高到最低的顺序对运算符进行了排列。

表 11.2　运算符的优先级

运算符	描述
[ ]，( )	字段访问、数组下标以及函数调用
++，--，-，~ ，!，typeof，new，void，delete	一元运算、返回数据类型、对象创建、未定义值
*，/，%	乘法、除法、取模
+，-，+	加法、减法、字符串连接
<<，>>，>>>	移位
<，<=，>，>=	小于、小于等于、大于、大于等于
= =，!=，= = =，!= =	等于、不等于、恒等、不恒等
&	按位与
^	按位异或
|	按位或
&&	逻辑与
||	逻辑或
?:	条件
=	赋值、运算赋值
,	多重求值

11.6　JavaScript 的程序流程控制

在许多情况下，都需要对程序的流程进行控制。例如，如果需要不断地执行一条语句，就要用到循环语句，如果需要对条件进行判断，就要用到条件语句。实际上，条件语句和循环语句是使用频率最高的语句。无论是进行数学运算，还是进行身份验证，都需要用到它们。

对程序流程进行控制总是离不开对条件的测试。JavaScript 对所有条件的测试结果都是布尔值。

11.6.1　if 语句

if 语句用来判定给定的条件是否满足，根据判定的结果（true 或 false）决定执行的语句。if 语句有以下几种形式：

1．if(*expression*) *statement*;

if 语句的这种用法是最简单，也是最常用的。如果括号里的表达式（*expression*）为真，则执行 *statement* 语句，否则就跳过这个语句。*statement* 语句可以与 if 表达式写在一行，但写在两行更容易阅读。例如：

```
if(x>y)
  document.write("x 比 y 大");
```

如果 *statement* 语句不止一条则必须使用大括号（{}）把这些语句括起来，表示同属于 if 语句的 *statement* 语句。例如：

```
if(a>b){
  bigger=a;
  document.write("a 比 b 大");
}
```

2．if(*expression*) *statement1*;else *statement2*;

这种形式和上面一种很相似，只是用 else 关键字扩展了 if 语句，提供了对表达式判断的另一种选择。如果 *expression* 返回真（true），则执行语句 *statement1*；否则如果 *expression* 返回假（false），则执行语句 *statement2*。例如：

```
if(score>=60)
  pass=1;
else
  pass=0;
```

3．if(*expression1*) *statement1*;else if(*expression2*) *statement2*;…
else if(*expressionm*) *statementm*; else *statementn*;

这是最复杂的 if 语句，else if 语句可以代替嵌套的 if…else 结构，使程序清晰明了。每个 else if 短语都跟在 statement 之后，短语的数量不限。最后用关键字 else 提供当所有条件都为假时的选择。例如：

```
if(score>90)
  grade="A";
else if(score>80)
  grade="B";
else if(score>60)
  grade="C";
else
  grade="D";
```

注意：if 语句里面的 *expression* 表达式一般为逻辑表达式或者关系表达式。系统对表达式的值进行判断，若为 0，则按“假（false）”处理，若为非 0，则按“真（true）”处理。但是表达式的类型不限于逻辑表达式，可以是任意的数值

类型（包括整数、字符串等）。例如，下面的语句都是正确的。

```
if(1)
  document.write("it is true");
if(0)
  document.write("it is false");
if('A')
  document.write("it is true");
```

下面是一个用 if 语句显示当前时间信息的例子。

```
<html>
<head>
<title>
显示日期
</title>
</head>
<body>
<script language="JavaScript">
<!--
var thisday=new Date( );
var toYear=thisday.getYear( );
var toMonth=thisday.getMonth( )+1;
var toDate=thisday.getDate( );
var toHour=thisday.getHours( );
document.write("<br>今天是"+toYear+"年"+"<br>"+toMonth+"月"+
  toDate+"日");
document.write("  ");
if(thisday.getDay( )==1) document.write("星期一 ");
if(thisday.getDay( )==2) document.write("星期二 ");
if(thisday.getDay( )==3) document.write("星期三 ");
if(thisday.getDay( )==4) document.write("星期四 ");
if(thisday.getDay( )==5) document.write("星期五 ");
if(thisday.getDay( )==6) document.write("星期六 ");
if(thisday.getDay( )==0) document.write("星期日 ");
var str="<br>";
if(toHour<=6)
  str+="现在是凌晨，还不睡觉？";
else if(toHour<12)
  str+="上午好";
else if(toHour<18)
  str+="下午好";
else
  str+="晚上好";
```

```
document.writeln(str);
-->
</script>
</body>
</html>
```

在这个例子中，if 语句的 3 种形式都用到了。其中，thisday 是一个创建的 Date 内置对象（内置对象将在第 13 章中介绍）。这个例子的执行结果如图 11.4 所示。

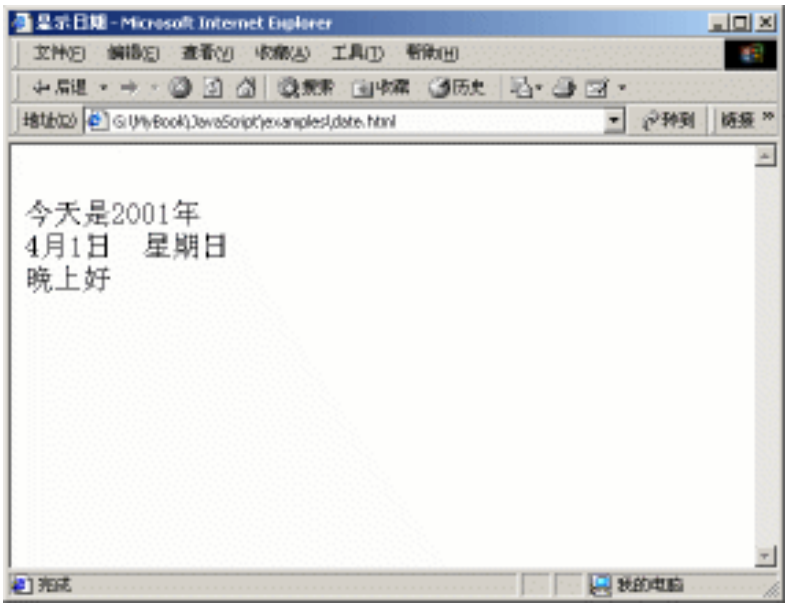

图 11.4 使用 if 语句显示时间信息

提示：if 语句支持嵌套格式，就是说在一个 if 语句中又包含一个或多个 if 语句。

11.6.2 switch 语句

switch 是多分支选择语句。if 语句只有两个分支可供选择，而实际问题中常常需要用到多分支选择。例如人口可以分为老年、中年、少年、儿童；颜色更是有红、橙、黄、绿、青、蓝、紫。当然这些都可以用嵌套的 if 语句来处理，但是如果分支太多，嵌套的 if 语句层数就多，使得程序冗长而且可读性降低。而 switch 语句可以直接处理多分支选择，其格式为：

```
switch(expression){
  case 常量表达式 1:
    statement1;
    break;
  case 常量表达式 2:
    statement2;
    break;
```

```
  default:
    statement3;
}
```

switch 后面括弧内的表达式（*expression*），可以是数值表达式也可以是字符表达式。当表达式的值与其中一个 case 后面的常量表达式的值相等时，就执行此 case 后面的语句，若所有的 case 中的常量表达式的值都没有与表达式的值匹配的，就执行 default 后面的语句。关键字 break 用于断开各个 case，防止发生偶然错误。

注意：每一个 case 的常量表达式的值必须互不相同，否则会出现互相矛盾的情况。

下面通过例子说明 switch 语句的用法。

```
<html>
<head>
<title>
利用 Switch 语句判断车的颜色
</title>
</head>
<body>
<script language="JavaScript">
<!--
var color="green";  //声明变量
switch(color){  //根据变量显示车的颜色
  case "red":
    document.write("车子是红色的");
    break;
  case "blue":
    document.write("车子是蓝色的");
    break;
  case "green":
    document.write("车子是绿色的");
    break;
  defult:
    document.write("未知的颜色");
}
-->
</script>
</body>
</html>
```

这个例子的执行结果如图 11.5 所示。

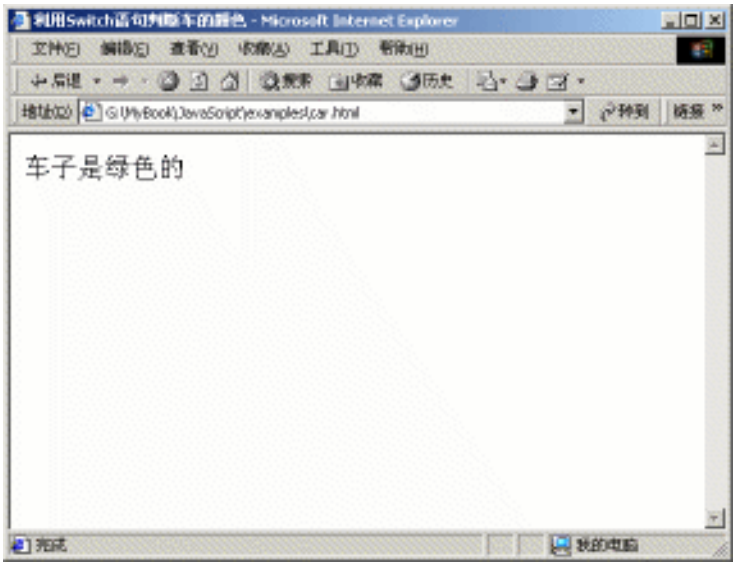

图 11.5 利用 Switch 语句判断车的颜色

11.6.3 for 语句

几乎所有实用程序都包含循环，例如求若干个数之和、利用迭代法求方程的根等。而 for 循环语句是最为灵活的，不仅可以用于循环次数已经确定的情况，而且可以用于循环次数不确定而只给出循环结束条件的情况。

for 循环语句由两部分组成：条件和循环体。条件部分决定循环的次数，循环体是需要循环执行的任务。它的一般形式如下：

```
for(expression1;expression2;expression3){
  statement;
}
```

它的执行过程如下：

（1）求解 *expression1*。

（2）求解 *expression2*，如果其值为真（true），则执行 for 语句中指定的语句 *statement*，然后执行后面的第 3 步。若其值为假（false），则结束循环，转到第（5）步。

（3）若 *expression2* 为真，在执行指定的语句后，求解 *expression3*。

（4）转到第（2）步继续执行。

（5）执行 for 语句下面的一个语句。

for 语句最简单的形式如下：

```
for(循环变量赋初值;循环条件;循环变量增值){
  语句;
}
```

例如：

```
for(i=1;i<100;i++){
  sum=sum+i;
}
```

注意：事实上，for 语句的 3 个表达式都可以不要。但是如果省略 *expression1*，则应该在 for 语句之前给循环变量赋初值；如果省略 *expression3*，则应该保证循环能够正常结束；如果省略 *expression2*，则不判断循环条件，循环会无休止地进行下去。

下面是一个用 for 循环打印九九乘法表的例子。

```
<html>
<head>
<title>
九九乘法表
</title>
</head>
<body>
<script language="JavaScript">
<!--
var i=1;  //行计数器
var j=1;  //列计数器
var str="";  //输出的字符串
for(;i<=9;i++){
  for(j=1;j<=i;j++){
    str=j+"X"+i+"="+i*j+"  ";
    document.write(str);
  }
  document.write("<br>");
}
-->
</script>
</body>
</html>
```

这个例子的执行结果如图 11.6 所示。

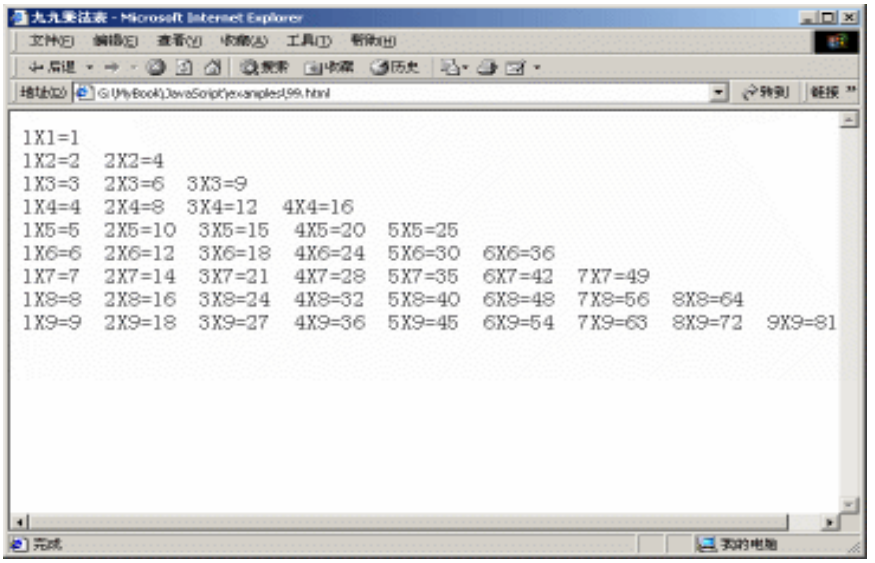

图 11.6 用 for 循环打印九九乘法表

11.6.4 for...in 语句

for...in 语句为 JavaScript 所特有的一种循环结构，用于访问 JavaScript 对象所有可列出的属性，包括数组的所有元素，因为它们的存储方式与 JavaScript 对象的属性的存储方式相同。可以用 for...in 循环语句对一个对象的所有属性进行循环，直到每一个属性都访问到。for...in 语句对于 JavaScript 对象不可列出的属性无法处理。

for...in 语句的格式为：

```
for(变量 in 对象/数组){
  语句;
}
```

在语句执行之前，将括号中的关键字 in 的右边指定的对象/数组的一个属性的名称赋给关键字 in 左边的变量，循环体中将用到该变量，这个过程将持续到所有的属性都被访问到。由于这个过程是由 JavaScript 解释器内部决定的，因此没有办法指定循环的顺序。

下面是一个显示 Date 对象所有属性的例子。

```
<html>
<head>
<title>
显示 document 对象所有属性
</title>
</head>
<body>
<script language="JavaScript">
var result="";
```

```
var i;
for(i in document)
  result+="document."+i+"="+document[i]+"<br>";
document.write(result);
</script>
</body>
</html>
```

这个例子的执行结果如图 11.7 所示。

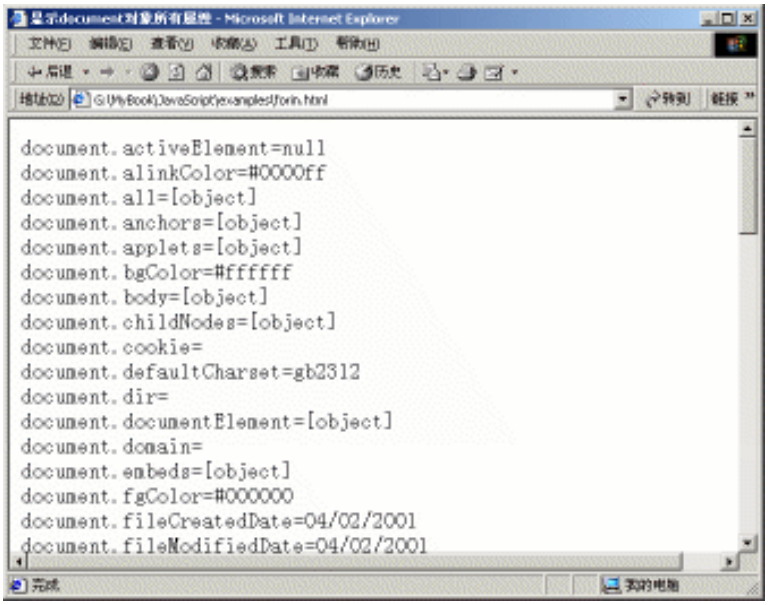

图 11.7　用 for...in 语句显示对象的所有属性

11.6.5　while 语句

while **语句是另一种循环语句，它的功能和** for **循环语句相同。**while **语句所控制的循环不断地测试一个条件，如果条件始终成立，则这个循环会一直持续下去，直到条件不再成立为止。**

while **语句的格式如下：**

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

因为 while **语句在每次循环之前先判断表达式，所以很有可能发生第** 1 **次判断表达式就为假则不执行循环体而跳出循环的情况。**

下面通过一个例子来说明 while **语句的使用方法。**

```
<html>
```

```
<head>
<title>
while 循环语句
</title>
</head>
<body>
<script language="JavaScript">
<!--
var counter=0;
var myNum=10;
while(myNum>0){
  myNum--;
  counter++;
}
window.alert("循环执行了"+counter+"次");
-->
</script>
</body>
</html>
```

显然，在循环体中，myNum **变量一直递减，直到条件不满足（**myNum=0**）才退出循环，如此循环执行了** 10 **次。这个例子的执行结果如图** 11.8 **所示。**

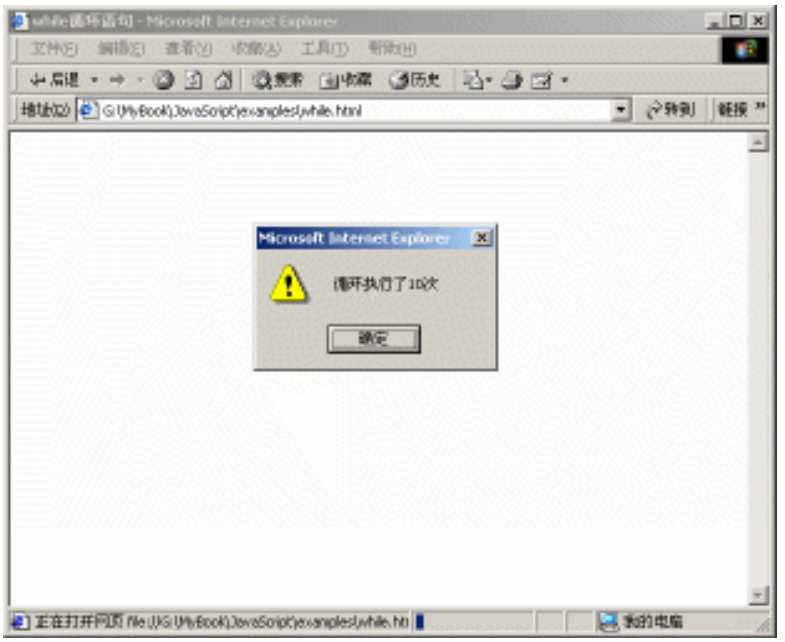

图 11.8 **使用** while **循环语句**

11.6.6　do...while 语句

do...while 语句是 while 语句的简单变体，二者之间除了语法格式不同之外，惟一的不同之处在于 do...while 语句先执行一次循环体，再作条件表达式的第 1 次判断。

do...while 语句的格式如下：

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

do…while 语句的运行过程是先执行一次循环体，然后作条件表达式的第 1 次判断，如果值为真则继续循环，如果值为假则跳出循环。语句循环体中必须包含改变表达式变量值的计算，从而保证不产生死循环。注意在最右边的括号后有分号。

下面的例子说明了 while 语句和 do...while 语句的区别。

```
<html>
<head>
<title>
do...while 循环语句
</title>
</head>
<body>
<script language="JavaScript">
<!--
var counter=0;
var myNum=10;
do{
  myNum--;
  counter++;
}
while(myNum>10);
window.alert("循环执行了"+counter+"次");
-->
</script>
</body>
</html>
```

可以看出，循环语句的条件（myNum>10）并不满足，但是循环体内的语句仍然被执行了一遍，其执行结果如图 11.9 所示。

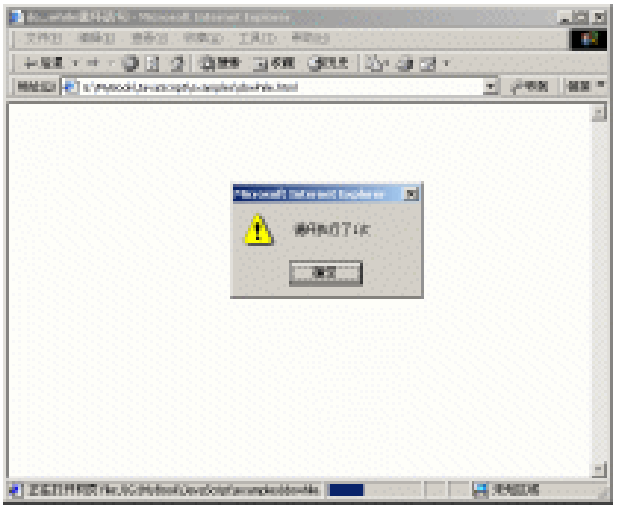

图 11.9 使用 do...while 语句

11.6.7 break 语句和 continue 语句

前面已经介绍过，用 break 语句可以使流程跳出 switch 结构，继续执行 switch 结构下面的语句。实际上，break 语句还可以用来从循环体内跳出，即提前结束循环，接着执行循环下面的语句。

下面是一个循环求和的例子，如果和大于 100，则中止循环。

```
<html>
<head>
<title>
break 语句
</title>
</head>
<body>
<script language="JavaScript">
<!--
var i=0;
var sum=0;
for(i=0;i<100;i++){
  sum+=i;
  if(sum>100){
    sum-=i;  //和已经大于 100，因此要把刚才加上的数减掉
    break;
  }
}
document.write("sum=1+2+3+...的结果为"+sum);
-->
```

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

这个例子的执行结果如图 11.10 所示。

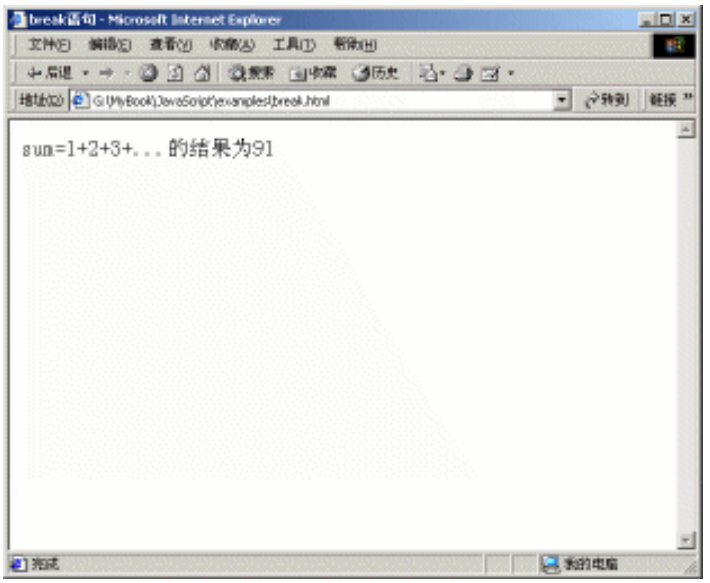

图 11.10　利用 break 语句中止求和循环

提示：break 语句不能用于循环语句和 switch 语句之外的任何其他语句中。

遇到循环嵌套时，break 语句只能跳出最内层的循环，这给程序设计带来了不便。幸运的是，我们可以利用语句标号来实现灵活的跳转。

语句标号是写在语句之前的一个标记，并在后面加上冒号（:）和语句分开。下面是一个利用语句标号和 break 语句跳出嵌套循环的例子。

```
<html>
<head>
<title>
break 语句
</title>
</head>
<body>
<script language="JavaScript">
<!--
var i=0;  //计数器 1
var j=0;  //计数器 2
var sum=0;
top:for(i=0;i<10;i++){
```

```
for(j=0;j<10;j++){
  sum+=i+j;
  if(sum>100){
    sum-=i+j;
    break top;
    }
  }
}
document.write("计数器 1 的值为"+i+"<br>");
document.write("计数器 2 的值为"+j+"<br>");
document.write("sum 值为"+sum+"<br>");
if(i<10)
  document.write("循环未结束，被 break 中止");
-->
</script>
</body>
</html>
```

在这个例子中，当外层循环还未结束时，sum 的值已经满足停止循环的条件，此时 break 语句直接中止最外层循环，其执行结果如图 11.11 所示。

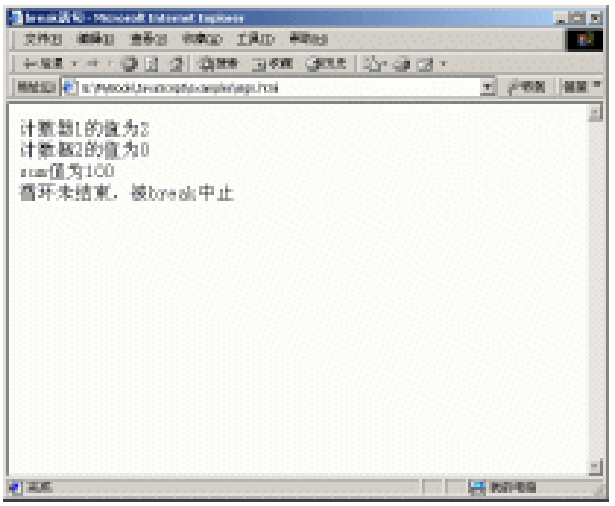

图 11.11 利用 break 语句中止最外层循环

与 break 语句不同，continue 语句只结束本次循环，而不是中止整个循环的执行。break 则是结束循环，不再进行条件判断。与 break 相同的是，在大多数情况下，continue 语句独占一行，但有时在 continue 后加一个语句标记，表示程序终止当前的一次循环，跳到标记指定的循环的开始处继续执行。

下面通过一个例子来说明 continue 语句的使用方法。在这个例子中，列出了 100 以内的所有不能被 3 整除的数。

```
<html>
<head>
<title>
0~100 内不能被 3 整除的数
</title>
</head>
<body>
<script language="JavaScript">
<!--
var i;
var primeNum=0;  //满足条件的数的个数
var str="";
for(i=0;i<=100;i++){
  if(i%3==0)
    continue;
    primeNum++;  //个数加一
if(primeNum%10==1)
  str+="<br>";  //每行列 10 个
  str+=i+"\t";
}
document.write(str);
document.write("<br>共"+primeNum+"个不能 3 整除的数");
-->
</script>
</body>
</html>
```

这个例子的执行结果如图 11.12 所示。

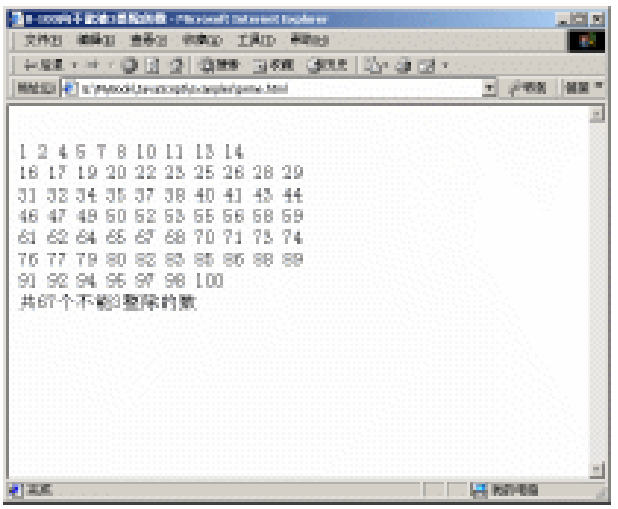

图 11.12　用 continue 语句列出 0~100 内不能被 3 整除的数

11.7 对象操作语句

JavaScript 中还提供了几条特殊的对象操作语句，使用它们可以很方便地操作对象。

1．with 语句

with 语句提供了一种简单清晰的方法来表达对象以及它的属性和方法之间的关系。简单地说，在 with 语句的作用范围之内，凡是没有指出对象的所有属性和方法操作，均指的是默认的对象，这个默认对象在 with 语句的开头给出。

with 语句的语法如下：

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

with 语句通常用来缩短特定情形下必须写的代码量。如下面的例子所示，在不使用 with 语句的情况下，不得不这样写：

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

如果这样的程序很长，将花费很多的时间写 Math 这样重复的代码。而使用 with 语句可以大大简化这个过程，使得程序精简易读，如下所示：

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

上面的语句中没有指出 cos、sin 函数的对象，则默认是 Math 对象。

2．new 操作符

可以使用 new 操作符来构造指定对象的一个实例。

new 操作符的语法如下：

```
实例名称=new  对象名称(参数);
```

new 操作符执行下面的任务：

（1）创建一个没有成员的对象。
（2）为该对象调用构造函数，传递一个指针给其作为 this 指针。
（3）构造函数根据传递给它的参数初始化该对象。

例如：

```
my_object=new Object;
```

```
my_array=new Array("ABC");
```

3．delete 操作符

同 new 操作符相反，delete 操作符可以删除一个对象的实例。简单来说，只要使用如下的语句：

```
delete 实例名称;
```

就可以完成删除对象实例的任务。

4．this 运算符

this 运算符总是指向当前的对象，同 C++语言中的 this 运算符功能很相似。

11.8　小 结

本章介绍了 JavaScript 的基本语法，包括 JavaScript 的程序结构、数据类型、变量和表达式、操作符和流程控制等。理解和掌握这些知识，是实现程序完整功能的前提。

第 12 章　JavaScript 的函数

函数是 JavaScript 中很重要的概念和工具，几乎是所有结构化和模块化编程的基础。本章将引导读者学习和掌握 JavaScript 语言的函数概念，使用 JavaScript 的系统函数，并能声明和自定义 JavaScript 函数。

12.1　函数的定义

函数是一个延迟动作集合的定义。函数可由事件处理程序或者 JavaScript 中的其他语句调用。JavaScript 的函数不仅能够执行各种操作，还可以返回各种结果。

有过其他语言编程经验的读者会发现，JavaScript 的函数与其他语言的子程序有相似之处。但是不像有些语言，区分过程（执行动作）和函数（执行动作并返回值），JavaScript 只有函数这一种形式。JavaScript 支持两种函数，即在语言中内置的函数以及用户自己创建的函数。

JavaScript 函数声明的标准形式如下：

```
function functionname([parameter1]…[,parameterN]){
  statements;
}
```

函数的命名的规则与命名 HTML 元素和变量一样。函数命应该是函数功能的简洁描述，这样在编程时才会一目了然。函数名区分大小写，开头必须是字母或下划线（_）。定义函数并不表示执行函数的指令，只有在程序中调用该函数时才会执行函数的指令。

另外，要尽量保证一个函数处理的内容不要太多，因为大的函数的维护和调试都会有很大困难，将大函数分解为多个小的更紧密的函数要更好一些。

12.2　函数的参数

函数定义和调用时常常需要一些参数。一个参数是指调用函数时所传递给函数的某个变量。函数的参数写在函数名后的括号中，例如：functionname(*parameter1*,*parameter2*,…)。用户也可能常常会看到 functionname( )这样的函数调用形式，这就意味着调用这个函数时不考虑它的参数问题。

参数提供了这样一种机制：通过函数调用，从一个语句传值给另一个语句。当一个函数接收到一个参数时，它将给这个参数值分配在函数的括号内对应的参数变量。例如：

```
function Hello(x,y){
```

```
  alert("Hello,"+x);
}
Hello("Mike","Marry");
```

该段程序定义了 Hello 函数之后，在后面调用这个函数，并用两个字符串作为参数。函数将自动分配这些参数给 x 和 y 这两个变量。因此在 alert 执行前，变量 x 被赋值为"Mike"，而变量 y 被赋值为"Marry"。

提示：与在 JavaScript 中定义的其他变量不同，函数参数没有用 var 关键字来说明，当函数被调用时它们才被初始化。

12.3　函数的返回值

当调用一个函数结束时，一般需要函数返回一个值到该函数的调用处，也就是从当前调用的函数退出，并从那个函数返回一个值。

一般使用 return [*expression*]来返回函数值，*expression* 参数是要从函数返回的值。如果省略该语句，则该函数不返回值。使用该语句可以终止一个函数的执行，并返回 *expression* 的值。如果省略 *expression* 参数，或者函数内没有 return 语句，那么当前调用的函数则返回一个 undefine 值。

下面这段程序说明了 return 语句的用法。

```
function myFunction(parameter1,parameter2){
  var temp;
  temp=parameter1*parameter2;
  return(temp);
}
```

12.4　创 建 函 数

JavaScript 用自定义的函数来代替其他语言中的过程、子程序和函数。函数的正式定义语法在前面已经介绍过了。但是从 Navigator 3.0（IE 3.0）开始，也可以用 new Function( ) 的形式创建函数，创建方法如下：

```
var function=new Function(["argname1"],…[,"argnameN"]."statement1;
  …[statementN]");
```

这是另一种创建函数的方法，在文档调入后需要创建函数时这个方法非常有用。函数的所有成分都出现在这个定义中。每个参数名都以字符串的形式给出，用逗号分隔开。最后的参数是一些语句，无论函数是否被调用，都要执行这些语句。每一条语句用分号分隔开，把整个语句序列放入引号中。例如：

```
var newFunction=new Function("width","height","var sx=screen.availWidth;
```

```
var sy=screen.availHeight;return(sx>=width&&sy>=height)");
```

本例中，newFunction 函数有两个参数，函数体中定义了两个局部变量 sx 和 sy，如果参数比局部变量小则返回布尔值“真”。而在传统方式下，这个函数应该如下定义：

```
function newFunction(width,height){
  var sx=screen.availWidth;
  var sy=screen.availHeight;
  return(sx>=width&&sy>=height);
}
```

12.5 嵌 套 函 数

JavaScript 提供了一种把函数嵌套在另一个函数内部的方式。在前面介绍的脚本程序中，所有的函数定义都是全局性的，所有的函数都可以被其他脚本访问。使用嵌套函数可以把一个函数封装在另一个函数内部，成为它的私有函数，而且可以在多个全局函数之内嵌套同名函数。例如：

```
function outFirst( ){
  statements;
  function inFirst( ){
    statements;
  }
  statements;
  }
function outSecond( ){
  statements;
  function inFirst( ){
    statements;
  }
  function inSecond( ) {
    statements;
  }
  statements;
}
```

提示：一个嵌套函数只能由包含它的函数中的语句访问。而且在嵌套函数定义中，外部函数的所有变量（包括参数）能在内部函数中使用，但外部函数不能访问内部函数中的变量。

12.6　递 归 函 数

函数可以自己调用自己，这称为递归函数。JavaScript 中一个递归函数的实例如下：

```
function factorial(n){
  if(n>0)
    return n*(factorial(n-1));
  else
    return 1;
}
```

在该程序的第 3 行，语句调用了它自己，把 *n* 的下一个值作为参数。当函数执行时，一层一层地深入下去，JavaScript 监视中间值并计算出最终结果。

警告：对每个递归函数都要小心地测试，特别要保证递归次数有一个上限。在上一个程序中是初始值 *n*，如果没有标明这个上限值，函数就会无限制地计算下去，会耗用内存，严重的甚至会引起程序崩溃。

12.7　系 统 函 数

在面向对象的编程语言中，函数一般是作为对象的方法定义的。有些函数由于其应用很广泛，可以作为独立的函数定义。而还有些函数根本就无法归属于任何一个对象，这些函数是 JavaScript 固有的，并且没有任何对象相关性，这就是 JavaScript 的系统函数。用户可以在 JavaScript 代码的任何位置调用这些函数。JavaScript 系统函数不能使用点（.）操作符来获取某个对象的实例，它们与使用 function 关键字创建的函数属于同一个层次。

JavaScript 中的系统函数有：escape，unescape，eval，parseInt，parseFloat，isNaN，toString，String 等。

下面就介绍一些这些函数。

1. escape(*charstring*[,1])和 unescape(*charstring*)

函数 escape 和 unescape 用于 HTML 使用的转义符（Escape Code）。目前，有两种主要转义码：ASCII（美国标准信息交换码）码和 EBCDI（扩充二进制编码的十进制交换码）码。JavaScript 使用的是基于 ISO Latin-1 字符集的 ASCII 码。通常人们很少注意转义码，因为它们一般很少用。转义码可以很好地进行格式化和显示文本。

JavaScript 使用百分号（%）加数字来表示 ASCII 码。这种编码方式把一个字符变成等值的十六进制形式，前面加一个%号。例如，与（&）操作符的 ASCII 码为 38，则它的编码就应该是%26。

所有的字符，包括制表符和回车符，都可以用这种方式编码成简单的字符串发给接收端再解码重建。当多行文本存储在数据库中时，必须存储成一个字符串，这时也可以用这

种方式处理多行文本。为了把一个字符串编码，需要使用 escape 函数，它返回一个编码了的字符串。例如：

```
var theCode=escape("Hello Mike");
//结果为："Hello%20Mike"
```

并不是所有的非字母字符都能用 escape 函数编码。一个例外就是＋号，它被 URL 用来分隔等检索串的各个部分。如果必须对＋号进行编码，要为函数提供可选的第 2 个参数，则+号被编码成它的十六进制形式：2B。例如：

```
var a=escape("Adding 1+1");
//结果为："Adding%201+1"
var a=escape("Adding 1+1",1)
//结果为："Adding%202%2B2"
```

escape 和 unescape 是两个功能相反的函数。把编码了的串解码要用到 unescape 函数。这个函数返回一个串，并转换所有 URL 编码的字符串。下面是对 unescape 函数的调用：

```
unescape("%26")
unescape("F")
```

2. eval(*string*)

eval 函数接收一个字符串形式的表达式，将其字符串参数作为一个 JavaScript 表达式进行赋值并返回这个值。作为参数的表达式可以采用任何合法的 JavaScript 操作符和常数。

如果传递给 eval 函数的参数中包含 JavaScript 命令，这些命令也可以被执行，就像这些命令是 JavaScript 程序的一部分一样。

eval 函数可以赋值给任何 JavaScript 表达式，无论表达式的形式如何。例如，如果变量 x 的值为 10，则下面的两个表达式都会给 y 和 z 赋值 155。

```
y=(x*14)-(x/2)+20;
z=eval("(x*14)-(x/2)+20");
```

利用这个函数可以很轻松地编写一个计算器程序，其代码如下：

```
<title>
eval 函数功能演示
</title>
<script language="JavaScript">
function Calculation(obj) {
  obj.result.value=eval(obj.expr.value);
}
</script>
<form name="evalform">
在这里输入需要计算的表达式：
<input type=text name="expr" size=20>
```

```
<br>
您所输入表达式的计算结果是：
<input type=text name="result" size=20>
<br>
<input type="button" value="计算" onClick="Calculation(this.form)">
<form>
```

该程序执行时的界面如图 12.1 所示。

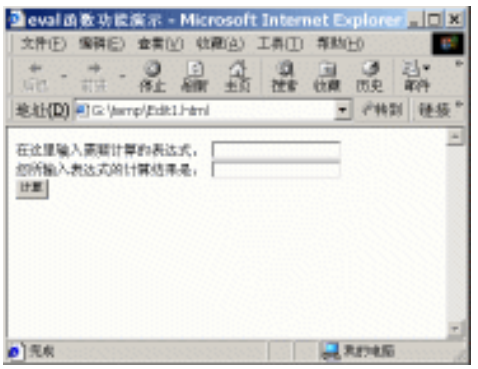

图 12.1　演示 eval 函数的功能

3. parseFloat(*numstring*)和 parseInt(*numstring*[,*radix*])

parseFloat 函数试图从一个字符串中提取一个浮点值。如果在字符串中存在除了数字、符号、小数点和指数符号之外的字符，parseFloat 函数就停止转换，返回已有的结果。如果第 1 个字符不能转换，该函数就返回 NaN，指示字符串中并不存在数字。

parseInt 函数试图从一个字符串中提取一个整数。parseInt 函数也可以附加一个可选的整数参数 *radix*。可以决定转换的进制，可以返回 *radix* 进制的一个整数，可以从字符串中提取二进制、八进制和十六进制的整数。如果不给出该参数，则由 JavaScript 自行决定，但是这样会带来一些问题。如在字符串的第 1 个字符是 0 的情况下，JavaScript 会将其视为八进制数，所以 parseInt("010")的结果是十进制的 8。如果在字符串中存在除了数字、符号、小数点和指数符号之外的字符，parseFloat 函数就停止转换，返回已有的结果。如果第 1 个字符不能转换，该函数就返回 NaN，指示字符串中并不存在数字。

当使用 parseInt 函数时，如果需要十进制数，则一定要指明。可选择的进制范围为 2~36。例如，把一个二进制数转换为十进制数：

```
var n=parseInt("011",2);
//结果为：3
```

4. isNaN(*numvalue*)

当数据来自文本框或者其他字符串源时，经常希望能够确保它是数字串。如果不是一个数字，可能会导致脚本出错。在运算之前可以用 isNaN 函数来判断一个值是否是数字、运算符号、小数点、指数符号以及括号。这个函数最常用的用途就是测试 parseInt 和

parseFloat 的结果。如果传递给这两个函数的参数不能转换成数字，则它们返回 NaN。例如：

```
function Calculation(form){
  var inputValue=parseInt(form.entry.value);
  if(isNaN(inputValue))
    alert("Your input must be numbers.");
  else
    statements for calculation;
}
```

5. String(*objecOrValue*)和 toString([*radix*])

每个 JavaScript 语言对象和文档都有 toString 方法。这个函数把对象的内容表达成一个有意义的字符串。表 12.1 列出了对某些核心语言对象类型进行 toString 转换后的结果。

表 12.1 对象经 toString 转换后的结果

对象类型	结果
String	同一字符串
Number	字符串表示
Boolean	true 或 false
Array	用逗号隔开的数组内容表（逗号后无空格）
Function	函数定义的字符串描述

toString 函数的参数可以从 2 取到 16，可以把数字转换成不同基数的字符串表示。例如，下面的程序计算并且画出了一个 0~15 范围内的十进制、十六进制和二进制转换表。

```
<html>
<head>
<title>
Number conversion Table
</title>
</head>
<body>
<b>
Using toString() to convert to other number bases:
</b>
<hr>
<table border=1>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Binary</th>
</tr>
<script language="JavaScript">
```

```
var content=" ";
for (var i=0;i<=15;i++){
  content+="<tr>";
  content+="<td>"+i.toString(10)+"</td>";
  content+="<td>"+i.toString(16)+"</td>";
  content+="<td>"+i.toString(2)+ "</td></tr>";
}
document.write(content);
</script>
</table>
</body>
</html>
```

该程序的运行结果如图 12.2 **所示。**

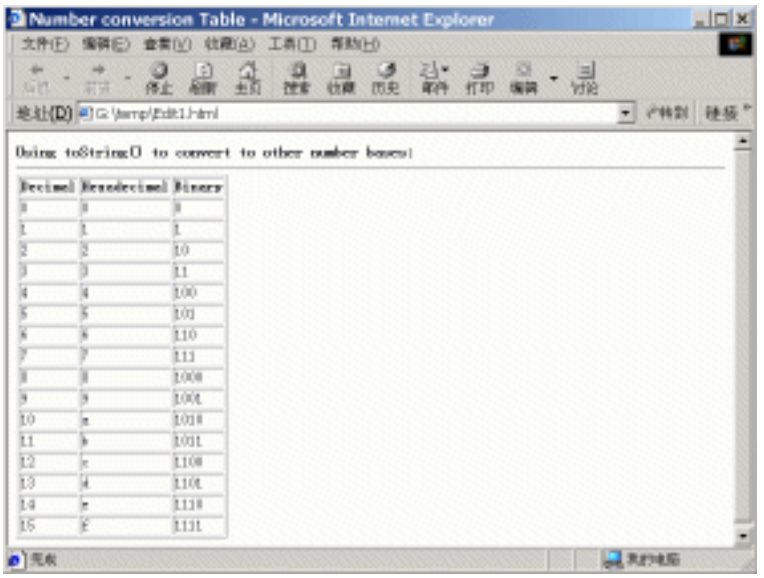

图 12.2　**演示** toString **函数的功能**

从 Navigator 4.0 **后新添了** String **函数，该函数的功能和** toString **函数相同。**String **的参数是一个对象名，它将对象转换成一个字符串。当这个对象是一个** Date **对象时，**String **返回一个代表这个日期的字符串。例如：**

```
<script language="JavaScript">
d=new Date(973404000000);
document.write(String(d)+"<br>");
</script>
```

返回结果为"Sun Nov 5 14:00:00 UTC+0800 2000"**。**

12.8 小 结

学习和使用函数是学好 JavaScript 的基础。本章主要介绍了函数的概念、用法和怎样创建自己的函数。并且结合 JavaScript 的函数库介绍了 JavaScript 的库函数的类别及用法。读者在了解了有关函数的概念后，可以自己通过函数手册进一步学习 JavaScript 的库函数。这样就可以一步一步地熟练掌握 JavaScript 函数的用法。

下一章我们将学习 JavaScript 的内置对象，这些对象提供编程的几种最常用的功能。

第 13 章　JavaScript 的内置对象

JavaScript 自带了一些内置的对象，这些对象提供编程的几种最常用的功能，例如数学运算、时间处理和字符串处理等。JavaScript 的内置对象有：String，Date，Event，Math，Array，RegExp 等。本章就来介绍一下这些内置对象。

13.1　JavaScript 中对象的基本概念

在学习 JavaScript 的面向对象编程之前，有必要先学习 JavaScript 对象的一些基本概念。下面将介绍面向对象编程中的几个非常关键的内容。

13.1.1　定义对象

在 JavaScript 中如何定义对象及创建其实例呢？实际上，JavaScript 中的对象是通过定义一类特殊的函数来创建的。

用户先可以定义自己的对象，然后通过创建构造函数在 JavaScript 中构造它们的实例。构造函数定义了构成一个对象的属性和方法。这与面向对象的程序设计中的类所做的非常相似。使用构造函数能够创建一个对象的多个实例。

构造函数的定义方法如下：

```
function functionName(parameters){
  Object Definition;
}
```

例如，可以定义 car 对象，它仅有一个属性，如下所示：

```
function car( ){
  this.owner=" ";
}
```

这里为 car 对象定义了一个没有参数的构造函数，在这个例子中，并没有传递参数给构造函数，owner 属性的初始值被赋为空串。

为使用对象，需要创建它的一个实例，例如，可以使用 new 关键字来创建一个实例，如下所示：

```
var car1=new car( );
```

然后可以引用 owner 属性并赋给它一个值，如下所示：

```
car1.owner="Mr. Li";
```

13.1.2 用文字表达式创建对象

除了可以使用构造函数和 new 语句创建对象实例外,还可以使用文字表达式创建对象,其语法如下所示:

```
objectName={propertyl:valuel,property2:value2,...,propertyn:valueN}
```

其中 *objectName* 是新对象名,*propertyl* 到 *propertN* 为标识符,可以为名字,数字或字符串,*valueN* 是一个分配给 *propertyN* 的表达式值。

可以给构造函数传递参数,从而在实例化一个对象时给属性赋值,例如:

```
function car(name){
  this.owner=name;
}
```

那么下面就可以用文字表达式来创建对象,并赋值给属性 owner。例如,在上节中创建的对象实例可以使用下面的语句来创建:

```
var car1=new car("Mr. Li");
```

13.1.3 方法和 this 语句

JavaScript 的一个强大功能是创建有函数属性的对象。读者已经知道这些函数被称为方法。在对象中使用方法,还有许多与对象使用有关的好处。我们在前面已经看到了用于创建对象的关键字 this,它还可以用于当前对象的方法。

现在先扩展构造函数来定义 car 对象的所有属性,如下所示:

```
function car(name){
  this.owner=name;
  this.maker="none";
  this.model="none";
  this.color="none";
  this.year="none";
}
```

给函数增加方法的过程分为两步:首先创建一个函数来定义方法,然后将这个方法加入构造函数中。

例如,要定义一个 carInfo 方法,那么先定义一个函数来指明该方法是如何工作的,如下所示:

```
function carInfo( ){
  writeln("Owner's name:"+this.name);
  writeln("Manufacturer:"+this.maker);
  writeln("Model:"+this.model);
  writeln("Color:"+this.color);
  writeln("Year:"+this.year);
```

```
}
```

提示：函数 carInfo 使用 this 关键字来引用与函数相关联的特定对象。

下一步就是将这个函数连接为 car 对象的一个方法，这可以通过在该对象的构造函数中增加下面一行来实现：

```
this.printInfo=carInfo;
```

这行语句指出 printInfo 方法由函数 carInfo 定义。用同样的方法可以增加其它的方法，然后把它们加入到构造函数中。

下面是前面讲到的函数和方法应用的完整例子——car 对象。

```
<html>
<head>
<title>
定义方法函数的演示
</title>
</head>
<body>
<script language="JavaScript">
function car(name){
  this.owner=name;
  this.maker="none";
  this.model="none";
  this.color="none";
  this.year="none";
  this.printInfo=carInfo;
}
function carInfo( ){
  document.writeln("Owner's name:"+this.owner+"<br>");
  document.writeln("Manufacturer:"+this.maker+"<br>");
  document.writeln("Model:"+this.model+"<br>");
  document.writeln("Color:"+this.color+"<br>");
  document.writeln("Year:"+this.year+"<br>");
}
var car1=new car("Mike");
car1.maker="Honda";
car1.model="Accord";
car1.color="White";
car1.year="1994";
car1.printInfo( );
</script>
</body>
</html>
```

该段程序将会产生如图 13.1 所示的输出结果。

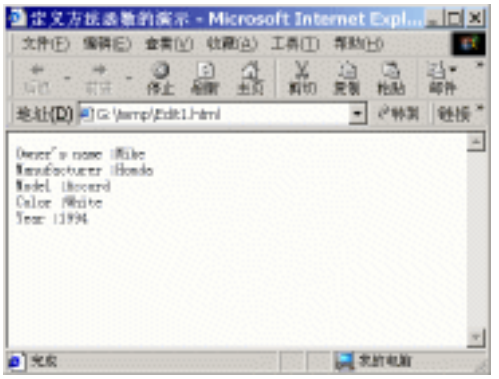

图 13.1 car 对象实例程序的输出结果

13.2 Array 对象

数组（Array）是一个有相同类型的有序数据项的数据集合。很多编程语言都支持数组特性。数组是把很多相同类型的数据组织在一起的有效手段。

13.2.1 定义 Array 对象

在数组里，可以通过下标来访问不同的数据。例如，a[1]指的是数组 a 的第 2 个元素。

提示：C 和 Java 语言的数组下标是从 0 开始，也有一些语言的数组下标从 1 开始。在 JavaScript 中，数组下标从 0 开始。

在 Navigator 以及 IE 中都可以用如下的方法定义一个数组：

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

当然，如果知道要使用的数组的大小，也可以显式地指定数组的元素个数。例如：

```
myArray=new Array(7);
```

定义数组还可以使用如下一种更加方便的形式：

```
myArray=new Array("sun","Mon","Tue","wed","Thu","Fri","Sat");
```

这段代码在定义一个数组的同时，也给这个数组的每个元素赋上了初值。这时的数组一共有 7 个元素——数组的大小是按照给出的初值数目自动调整的。

提示：当 Array 对象只有一个参数值时，如果这个参数是整数，JavaScript 就以这个参数值定义数组的大小，如果这个参数不是整数，JavaScript 就认为这是数组的第 1 个元素的初值。

JavaScript 的数组实现是非常灵活的。如果在定义数组的时候没有指定它的大小，那么根据程序的需要它可以自动地伸缩。可以在数组中储存数字、字符串或其他的对象。此外，同一个数组中的元素也不必是同一个数据类型的，甚至可以让数组的一个元素同时也是一个数组，这样就形成了高维数组。例如：

```
test=new Array(1,2,new Array("Mon","Tue","Wed"));
```

那么就有：

```
test[0]=1
test[1]=2
test[2][0]="Mon"
test[2][1]="Tue"
test[2][2]="Wed"
```

提示：Array 对象的 length 属性记录了当前数组中元素的个数。例如对于上面的 test 数组，语句 test.length 就会返回数值 3。由于 JavaScript 数组中的元素可能不是连续存放的，所以这个属性值可能并不反映实际元素的个数。只有当数组中的元素连续存放，中间没有空格的时候，这个 length 属性值才和数组的实际大小相符合。

13.2.2　使用 Array 对象

在 JavaScript 中，每个对象都是其属性的数组，而每个数组也是一个对象。例如，下面创建的对象就是一个有以下 4 个元素的数组。

```
myhouse[0]=5
myhouse[1]="Colonial";
myhouse[2]=1999；
myhouse[3]=ture;
```

这种形式的对象表达的优点在于可以顺序访问属性值，而不是使用各自的名字访问。例如，如果知道 house 对象有 4 个成员属性，那么可以使用以下的函数显示所有的属性值。

```
function showhouse(somehouse){
  for(var item=0;iter<4;item++){
    document.write("<br>Property "+item+"is"+somehouse[item]);
  }
documetn.write("<br>");
}
```

这个函数必须使用一个实例调用，而不是对象调用。这个程序还有一个缺陷：如果对象实例的属性不是 4 个而是 3 个或 5 个，则这个函数显示的信息将缺少属性甚至出错。解决这个问题的方法之一是使用 length 属性，它提供对象中的属性个数。

13.2.3 Array 对象的方法

Array 对象可以使用的方法如下：

1. join 方法

该方法可以把一个数组中所有的数据以一个字符串的形式表达出来，这样在网页中就可以方便地显示和阅读了。

join 方法的语法为：

```
arrayobj.join(separator)
```

其中参数 *separator* 用于指出每一个数据之间使用的分隔符号，如果省略这个参数，则默认的分隔符号是一个空字符串，意味着每一个数据之间没有分隔。例如：

```
var myarray=new Array("My","teacher","is",Mr.","Wang");
var mystring=myarray.join(",");
```

那么字符串 mystring 的值就是"My, teacher,is,Mr,Wang"。

2. reverse 方法

该方法不带参数，它返回一个元素顺序被反转的 Array 对象。reverse 方法将一个 Array 对象中的元素位置进行反转，在执行过程中，这个方法并不会创建一个新的 Array 对象。如果数组是不连续的，reverse 方法将在数组中创建元素以便填充数组中的间隔。这样所创建的全部元素的值都是 undefined。例如：

```
function ReverseDemo( ){
  var a,b;
  a=new Array(0,1,2,3,4);
  b=a.reverse( );
  return(b);
}
```

调用该函数后，数组中元素的顺序将是 4，3，2，1，0。

3. sort 方法

该方法返回一个已经进行了元素排序的 Array 对象。其语法为：

```
arrayobj.sort(sortFunction)
```

其中参数 *sortfunction* 是用来确定元素顺序的函数的名称，如果这个参数被省略，那么元素将按照 ASCII 码字符顺序进行升序排列。

sort 方法将 Array 对象进行适当的排序，在执行过程中并不会创建新的 Array 对象。如果为 *sortfunction* 参数提供了一个函数，那么该函数必须返回下列值之一：

- 负值：如果所传递的第 1 个参数比第 2 个参数小。
- 0：如果两个参数相等。

- 正值：如果第 1 个参数比第 2 个参数大。

下面这个例子说明了 sort 方法的用法。

```
function SortDemo( ){
  var a,b;
  a=new Array("X","y","d","Z","v","m","r");
  b=a.sort();
  return(b);
}
```

调用该函数后，数组 b 中元素的顺序将是 X，Z，d，m，r，v，y。

4. pop 方法

该方法不带参数，它从一个数组中删除最后一个元素并返回这个元素。这个方法改变了数组的长度。

5. push 方法

使用这个方法可以在数组最后加入一个元素。如果读者学习过数据结构，那么对这两个方法就不会感到陌生。push 的语法为：

```
arrayobj.push(Newelement)
```

6. concat 方法

该方法返回一个新数组，这个新数组是由两个数组组合而成的。它的语法为：

```
array1.concat(array2)
```

其中：

- *array1*：一个与 *array2* 进行连接的 Array 对象。
- *array2*：一个连接到 *array1* 末尾的 Array 对象。

concat 方法返回一个 Array 对象，它是 *array1* 和 *array2* 的连接。如果一个对象引用被从 *array1* 或 *array2* 复制到结果中，那么结果中的对象引用仍然指向同一个对象。对该对象的改变将同时反映在两个数组中。

下面这个例子说明了 concat 方法的用法。

```
function ConcatArrayDemo( ){
  var a,b,c;
  a=new Array(0,1,2,3,4);
  b=new Array(5,6,7,8,9);
  c=a.concat(b);
  return©;
}
```

调用该函数后数组 c 中的元素将是 0，1，2，3，4，5，6，7，8，9。

7. slice 方法

该方法返回一个数组的一段。它的语法为：

```
arrayobj.slice(start,[end])
```

其中：

- *arrayobj*：一个 Array 对象。
- *start*：*arrayobj* 中所指定部分的从 0 开始计算的开始元素的下标。
- *end*：可选参数。*arrayobj* 中所指定部分的从 0 开始计算的结束元素的下标。slice 方法返回一个 Array 对象，其中包含了 *arrayobj* 的指定部分。slice 方法一直复制到 *end* 所指定的元素，但是不包括该元素。如果 *end* 值为负，那就说明 *end* 是从 *arrayobj* 结尾计算的偏移量。此外，它不是从 0 开始计数的。如果省略了这个值，那么 slice 方法将一直复制到 *arrayobj* 的结尾。

在下面这个例子中，除了最后一个元素之外，myArray 中所有的元素都被复制到 newArray 中。

```
newArray=myArray.slice(0,-1)
```

如果一个对象引用被从 *arrayobj* 复制到结果中，那么结果中的该对象引用仍然指向同一个对象。对该对象的改变将被同时反映在两个数组上。

13.3 String 对象

String（字符串对象）对象也是 JavaScript 的内置对象。一个字符串是由一串字符组成的。例如：Netscape。这里的分隔符号可以是单引号，也可以是双引号，在 JavaScript 中两者的作用都是分隔字符。String 对象同 JavaScript 语言本身结合得如此紧密，以至于在变量定义时甚至可以不使用传统的 new 操作符。

13.3.1 创建 String 对象

String 对象可用于处理或格式化文本字符串以及确定和定位字符串中的子字符串。String 对象可用字符串文字显式创建。用这种方法创建的 String 对象（指以标准字符串形式）与用 new 运算符创建的 String 对象处理上不同。所有字符串文字共享公用的全局字符串对象。所以，如果为字符串文字添加属性，则它对所有标准字符串对象都是可用的。例如：

```
var alpha,beta;
alpha="这是一个字符串";
beta="这也是一个字符串";
alpha.test=10;
```

在本例中，为 beta 和所有将来的字符串定义属性 test。然而，在下面的例子中，被添加属性的处理略有不同：

```
var gamma,delta;
gamma=new String("这是一个字符串");
delta=new String("这是也一个字符串");
gamma.test=10;
```

在这种情况下，不为 delta 定义 test 属性。每个用 new String 创建的 String 对象有其自己的一组成员。这是对 String 对象和字符串文字的处理不同的惟一情况。

13.3.2　String 对象的方法

下面介绍 String 对象的几个方法。

1. anchor 方法

该方法在对象中的指定文本两端放置一个有 name 属性的 HTML 锚点。它的语法为：

```
stringobj.anchor(anchorstring)
"String Literal".anchor(anchorstring)
```

其中 *anchorstring* 参数是想要放置 HTML 锚点的 name 属性的文本。

可以调用 anchor 方法来在 String 对象外创建一个命名的锚点，下面示例说明了用 anchor 方法是如何实现的。

```
var strVar="This is an anchor";
strVar=strVar.anchor("Anchor1");
```

执行完最后一条语句后，strVar 的值与下面的程序行等效。

```
<a name="Anchor1">This is an anchor</a>
```

2. charAt 方法

该方法返回指定索引位置处的字符。它的基本语法为：

```
stringObj.charAt(index)
"String Literal".charAt(index)
```

其中参数 *index* 是想得到的字符的基于 0 的索引。有效值是 0 与字符串长度减 1 之间的值。

charAt 方法返回一个字符值，该字符位于指定索引位置。字符串中的第 1 个字符的索引为 0，第 2 个的索引为 1 等等。如果参数 *index* 超出有效范围的索引值则返回 undefined。

下面的示例说明了 charAt 方法的用法。

```
function charAtTest(n){
  var str="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  var s;
  s=str.charAt(n-1);
  return(s);
}
```

如果 *n* 的值为 9，那么 s 的返回值就是字符"I"。

3. charCodeAt 方法

该方法返回指定字符的 Unicode 编码。它的基本语法为：

```
stringobj.charCodeAt(index)
```

其中：

- *stringobj*：一个 String 对象或文字。
- *index*：指定字符基于 0 的索引。

如果在指定 *index* 处没有字符，则返回 NaN。

下面的示例演示了 charCodeAt 方法的用法。

```
function charCodeAtTest(n){
  var str="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  var s;
  s=str.charCodeAt(n-1);
  return(s);
}
```

如果参数 *n* 的值为 9，那么 s 的返回值就是 73。

4. fromCharCode 方法

该方法从一些 Unicode 字符值中返回一个字符串。它的语法为：

```
stringobj.fromCharCode(code1,code2,...,coden)
```

其中各个参数是要转换为字符串的 Unicode 字符值序列。在调用 fromCharCode 前不必创建 String 对象。

在下面的例子中，test 包含字符串"plain"。

```
var test=stringobj.fromCharCode(112,108,97,105,110);
```

5. indexOf 方法

该方法返回 String 对象内第 1 次出现子字符串的字符位置。它的基本语法为：

```
stringobj.indexOf(substring,startindex)
```

其中：

- *substring*：要在 String 对象中查找的子字符串。
- *startindex*：可选的整数值，指出在 String 对象内开始查找的索引。如果省略，则从字符串的开始处查找。

indexOf 方法返回一个整数值，指出 String 对象内子字符串的开始位置。如果没有找到子字符串，则返回-1。如果 *startindex* 是负数，则 *startindex* 被当作 0。如果它比最大的字符

位置索引还大，则它被当作可能的最大索引。

下面的示例演示了 indexOf 方法的用法。

```
function IndexDemo(str2){
  var str1="BABEBIBOBUBABEBIBOBU"
  var s=str1.indexOf(str2);
  return(s);
}
```

如果 str2 的值为"BOB"，那么 s 的返回值就是 6。

6. lastIndexOf 方法

该方法返回 String 对象中子字符串最后出现的位置。它的语法为：

```
stringobj.lastIndexOf(substring,startindex)
```

其中：

- *substring*：要在 String 对象内查找的子字符串。
- *startindex*：可选的整数值，指出在 String 对象内进行查找的开始索引位置。如果省略，则从字符串的末尾开始查找。

lastIndexOf 方法返回一个整数值，指出 String 对象内子字符串的开始位置。如果没有找到子字符串，则返回-1。如果 *startindex* 是负数，则 *startindex* 被当作 0。如果它比最大字符位置索引还大，则它被当作最大的可能索引。

它从右向左执行查找。否则，该方法和 indexOf 相同。

7. match 方法

该方法使用正则表达式对象对字符串进行查找，并将结果作为数组返回。它的语法为：

```
stringobj.match(rgExp)
```

其中：

- *stringobj*：对其进行查找的 String 对象或文字。
- *rgExp*：在查找中使用的正则表达式。

match 方法与 exec 方法有些相似，将返回一个数组。该数组的元素 0 包含最后匹配的字符，元素 1...n 包含与正则表达式中任何用插入符分开的子字符串匹配的内容。

该方法将更新 RegExp 对象的内容，有关 RegExp 对象的知识将在后面讲到。

下面的示例演示了 match 方法的用法。

```
function MatchDemo( ){
  var r,re;
  var s="The quick brown fox jumped over the lazy yellow dog.";
  re=/fox/i;
  r=s.match(re);
```

```
  return(r);
}
```

调用函数 MatchDemo，**那么返回** r **的值就是**"fox"。

8. replace **方法**

该方法返回根据正则表达式进行文字替换后的字符串的复制结果。它的语法为：

```
stringobj.replace(rgExp,replaceText)
```

其中：

- *stringobj*：**要执行该替换的** String **对象或文字。该对象不会被** replace **方法修改。**
- *rgExp*：**描述要查找的内容的一个正则表达式对象。**
- *replaceText*：**一个** String **对象或文字，对于** *stringobj* **中每个匹配** *rgExp* **的位置，都用该对象所包含的文字加以替换。**

replace **方法的结果是一个完成了所有替换的** String **对象的复制。该方法将更新** RegExp **对象的内容。**

下面的示例演示了 replace **方法的用法。**

```
function ReplaceDemo(){
  var r,re;
  var s="The quick brown fox jumped over the lazy yellow dog.";
  re=/fox/i;
  r=s.replace(re,"pig");
  return(r);
}
```

调用函数 ReplaceDemo，**那么返回** r **的值就是**"The quick brown pig jumped over the lazy yellow dog."。

9. search **方法**

该方法返回与正则表达式查找内容匹配的第 1 **个子字符串的位置。它的语法为：**

```
stringobj.search(rgExp)
```

其中：

- *stringobj*：**进行查找的** String **对象或文字。**
- *rgExp*：**包含要查找的模式的正则表达式对象。**

search **方法指明是否存在相应的匹配。如果找到一个匹配，**search **方法将返回一个整数值，指明这个匹配距离字符串开始的偏移位置。如果没有找到匹配，则返回**-1。**要获得进一步的信息，请使用** match **方法。**

下面的示例演示了 search **方法的用法。**

```
function SearchDemo( ){
  var r,re;
  var s="The quick brown fox jumped over the lazy yellow dog.";
  re=/fox/i;
  r=s.search(re);
  return(r);
}
```

调用函数 SearchDemo，**那么返回** r **的值是** 16。

10. slice **方法**

该方法返回字符串的片段。它的语法为：

```
stringobj.slice(start,[end])
```

其中：

- *stringobj*：**一个** String **对象或文字。**
- *start*：**下标以** 0 **开始的** *stringobj* **指定部分的起始索引。**
- *end*：**下标以** 0 **起始的** *stringobj* **指定部分的结束索引。该参数可选。**

slice **方法返回一个包含** *stringobj* **指定部分的** String **对象。如果** *end* **是负数，就表示从** *stringobj* **结尾开始起计算的一个正偏移量。另外，在此意义下，它不是从** 0 **开始起计算的。例如，如果** *end*=1 **则表示提取到字符串的结尾。如果省略** *end*，**就一直提取到** *stringobj* **的结尾。**

在下面的示例中，slice **方法的两种用法将返回相同的值。第** 2 **个示例中的**-1 **指向** str1 **中的最后一个字符，并作为提取操作的结束位置。**

```
str1.slice(0)
str2.slice(0,-1)
```

11. toLowerCase **方法**

该方法返回一个字符串，该字符串中的字母被转换为小写字母。它的语法为：

```
stringobj.toLowerCase( )
```

toLowerCase **方法对非字母字符不会产生影响。下面的示例演示了** toLowerCase **方法的用法。**

```
var strVar="This is a STRING object";
strVar=str.toLowerCase( );
```

在执行上一条语句后，strVar **的值为**"this is a string object"。

12. toUpperCase **方法**

该方法返回一个字符串，该字符串中的所有字母都被转化为大写字母。它的语法为：

```
stringobj.toUpperCase( )
```

toUpperCase 方法对非字母字符会产生影响，这一点与 toLowerCase 方法恰好相反。

提示：为继承 C 语言的传统，这里函数的位置参数和所有的位置返回值均从 0 开始计算。

除了上面的 String 方法之外，还有用于 Web 网页外观处理的 String 方法。所有可以处理字符串在 Web 页面中的外观的 String 对象的内部方法可以参考表 13.1。

表 13.1 用于 Web 网页外观处理的 String 方法

方法名称	功能	相当于
Mystring.big( )	加大字符串显示的字号	<big>Mystring</big>
Mystrin.blink( )	使字符串闪烁	<blink>Mystring</blink>
Mystring.bold( )	使字符串以黑体显示	Mystring
Mystring.fixed( )	使字符串显示成等宽字	<tt>Mystring</tt>
Mystring.fontcolor(*color*)	指定字符串显示的颜色	<fontcolor="green">Mystring
Mystring.fontsize(*fontsize*)	指定字符串的大小	Mystring
Mystring.italics( )	使字符串以斜体显示	Mystring
Mystring.small( )	减小字符串显示的字号	Mystring
Mystrin.strike( )	显示删除线	<strike>Mystring</strike>
Mystring.sub( )	使字符串以下标显示	_{Mystring}
Mystring.sup( )	使字符串以上标显示	^{Mystring}

提示：并不是所有的 HTML 标记都有对应的方法可以处理。如果没有相应的方法，可以直接在字符串中嵌入 HTML 标记。

另外，由于 HTML 使用" "来分隔一个字符串或者类似的值，所以如果在一个 HTML 标记中必须使用 JavaScript 语句，那么就必须用' '来分隔 JavaScript 自己的字符串。例如：

```
<input type="button" value="press Me" onClick="Myfunc("astring")">
```

13.4 Date 对象

JavaScript 内置的 Date 对象可以用来处理所有有关日期与时间的操作。Date 对象保存以毫秒为单位表示的特定时间段。如果某个参数的值大于其范围或为负数，则存储的其他值将作相应的调整。例如，如果指定时间段为 150 秒，那么 JavaScript 将该数字重新定义为 2 分 30 秒。如果参数的值为 NaN，则表示该对象不代表特定的时间段。如果未向 Date 对象传递参数，它将被初始化为当前时间（UTC）。在能够使用该对象前必须为其赋值。

Date 对象能够表示的日期范围约等于 1970 年 1 月 1 日前 285 年与此日期后 616 年。Date 对象具有两个不创建 Date 对象就可以调用的静态方法：parse 和 UTC。

13.4.1　创建 Date 对象

Date 对象采用如下形式之一来创建：

```
var newDateObj=new Date();
var newDateObj=new Date(dateVal);
var newDateObj=new Date(year,month,date[,hours[,minutes[,seconds
  [,ms]]]]);
```

其中：

- *dateVal*：如果是数字值，*dateVal* 表示指定日期与 1970 年 1 月 1 日午夜间全球标准时间之间的毫秒数。如果是字符串，则 *dateVal* 按照 parse 方法中的规则进行解析。*dateVal* 参数也可以是从某些 ActiveX 对象返回的 VT_DATE 值。
- *year*：完整的年份，比如，1976（而不是 76）。
- *month*：表示月份，是从 0 到 11 之间的整数（1 月至 12 月）。
- *date*：表示日期，是从 1 到 31 之间的整数。
- *hours*：该参数可选，如果提供了 *minutes* 则必须给出。表示小时，是从 0 到 23 的整数（午夜到 11pm）。
- *minutes*：该参数可选，如果提供了 *seconds* 则必须给出。表示分钟，是从 0 到 59 的整数。
- *seconds*：该参数可选，如果提供了 *millisecondms* 则必须给出。表示秒钟，是从 0 到 59 的整数。
- *ms*：该参数可选，表示毫秒，是从 0 到 999 的整数。

如果采用第 1 种形式创建日期对象的实例，这个实例就自动存储了当前的日期和时间。采用第 2 种形式创建 Date 对象，这个日期就自动存储在新创建的对象实例中。

同样也可以采取一种最直接的方式——显式地指定日期的年份、月份、日期和小时等信息。采用这种形式的时候，年份、月份和日期是必须指定的，而其他的时间信息则可以省略。

同 String 对象不同的是，Date 对象定义了静态方法。所谓的静态方法，指的是所有的 Date 对象的实例共同拥有同样的方法，这样的方法同每一个具体的实例没有关系。Date 对象拥有的两个静态方法是：

- UTC：Date.UTC(*parameters*)。
- parse：Date.parse(*parameters*)。

parse 方法可以方便地将一个以字符串表达的时间或日期转换成时间或日期的内部表示。例如：

```
AfineDate=new Date( )；
AfineDate.setTime(Date.parse("March 17,1999"))；
```

这样，Date 对象的实例 AfineDate 就存放了 1999 年 3 月 17 日的信息。

在时间的处理上，JavaScript 同 Java 语言非常相似。两种语言均在内部存储了自从 1970 年 1 月 1 日开始到指定的时间之间的差值。这样的做法是从 UNIX 系统中继承下来的。1970 年 1 月 1 日距 UNIX 系统在学术和商业领域的广泛范围应用不远，一直被认为是 UNIX 系统的“创世纪”。

13.4.2 Date 对象的方法

下面介绍 Date 对象可以使用的方法。

1. getDate 方法

该方法返回 Date 对象中用本地时间表示的一个月中的日期值。它的语法为：

```
dateobj.getDate( )
```

getDate 方法的返回值是一个处于 1 到 31 之间的整数，它代表了相应的 Date 对象中的日期值。与之相对应的设置 Date 对象中用本地时间表示的数字日期的方法是 setDate。

下面这个例子演示了 getDate 方法的用法。

```
function DateDemo( ){
  var d,s;
  s="今天日期是: ";
  d=new Date();
  s+=(d.getMonth()+1)+"/";
  s+=d.getDate()+"/";
  s+=d.getYear();
  return(s);
}
```

调用函数 DateDemo，那么返回 s 的值是"今天日期是: 4/3/2001"。

2. getDay 方法

该方法返回 Date 对象中用本地时间表示的一周中的日期值。它的语法为：

```
dateobj.getDay( )
```

getDay 方法所返回的值是一个处于 0 到 6 之间的整数，它代表了一周中的某一天，返回值与一周中日期的对应关系如下：

- 0：星期天
- 1：星期一
- 2：星期二
- 3：星期三
- 4：星期四
- 5：星期五

下面这个例子演示了 getDay 方法的用法。

```
function DateDemo( ){
  var d,day,x,s="今天是: ";
  var x=new Array("星期日","星期一","星期二");
  var x=x.concat("星期三","星期四","星期五");
  var x=x.concat("星期六");
  d=new Date( );
  day=d.getDay( );
  return(s+=x[day]);
}
```

如果当前是星期二的话，调用函数 DataDemo s 的返回值将是"今天是: 星期二"。

3. getHours 方法

该方法返回 Date 对象中用本地时间表示的小时值。它的语法为：

```
dateobj.getHours( )
```

getHours 方法返回一个处于 0 到 23 之间的整数,这个值表示从午夜开始计算的小时数。在下面两种情况下此方法的返回值是 0：时间在 1:00:00am 之前，或者在创建 Date 对象的时候没有将时间保存在该对象中。而要确定究竟是哪种情况，惟一的方法就是进一步检查分钟和秒钟值是否也是 0。如果这两个值都是 0，那就几乎可以确定是由于没有将时间值保存到 Date 对象中。

与 getHours 方法相对应的用于设置 Date 对象中用本地时间表示的小时值的方法是 setHours。

4. getMinutes 方法

该方法返回 Date 对象中用本地时间表示的分钟值。它的语法为：

```
dateobj.getMinutes( )
```

getMinutes 方法返回一个处于 0 到 59 之间的整数，返回值就等于保存在 Date 对象中的分钟值。在两种情况下返回值为 0：一种情况是在时钟整点之后经过的时间少于一分钟。另外一种情况是在创建 Date 对象的时候没有将时间值保存到该对象中。而要确定究竟是哪种情况，惟一的方法是同时检查小时和秒钟值是否也是 0。如果这两个值都是 0，那就几乎可以确定是由于没有将时间值保存到该 Date 对象中。

与 getMinutes 方法相对应的用于设置 Date 对象中用本地时间表示的分钟值的方法是 setMinutes。

5. getSeconds 方法

该方法返回 Date 对象中用本地时间表示的秒数。它的语法为：

```
dateobj.getSeconds( )
```

getSeconds 方法返回一个处于 0 到 59 之间的整数，它表示了相应的 Date 对象中的秒钟值。在两种情况下返回值为 0：第 1 种情况是在当前的一分钟中所经过的时间少于一秒。

另外一种情况是在创建 Date 对象时没有将时间值保存到该对象中。而为了确定究竟属于哪种情况，惟一的方法是同时检查小时和分钟值是否也都是 0。如果这两个值都是 0，那就几乎可以确定是由于没有将时间值保存到 Date 对象中。

与 getSeconds 方法相对应的用于设置 Date 对象中用本地时间表示的秒钟值的方法是 setSeconds。

6. getMonth 方法

该方法返回 Date 对象中用本地时间表示的月份值。它的语法为：

```
dateobj.getMonth( )
```

getMonth 方法返回一个处于 0 到 11 之间的整数，它代表 Date 对象中的月份值。这个整数并不等于按照惯例来表示月份的数字，而是要比按惯例表示的值小 1。如果一个 Date 对象中保存的时间值是"Jan 5,1996 08:47:00"，那么 getMonth 方法就会返回 0。

与 getMonth 方法相对应的用于设置 Date 对象中用本地时间表示的月份值是 setMonth。

7. getTime 方法

该方法返回 Date 对象中的时间值。它的语法为：

```
dateobj.getTime( )
```

getTime 方法返回一个整数值，这个整数代表了从 1970 年 1 月 1 日开始计算到 Date 对象中的时间之间的毫秒数。日期的范围大约是 1970 年 1 月 1 日午夜的前 285 年到后 616 年。负数代表 1970 年之前的日期。

在进行各种日期和时间换算的时候，可以定义一些变量来表示一天、一个小时或一分钟中包含的毫秒数。这样做通常是很有帮助的，例如：

```
var MinMilli=1000*60;
var HrMilli=MinMilli*60;
var DyMilli=HrMilli*24;
```

与 getTime 方法相对应的用于设置 Date 对象的日期和时间值的方法是 setTime。

8. getTimezoneOffset 方法

该方法返回用分钟表示的主计算机上的时间和全球标准时间（UTC）之间的差别。它的语法为：

```
dateobj.getTimezoneOffset( )
```

getTimezoneOffset 方法返回一个整数值，这个整数代表了当前计算机上的时间和 UTC 之间相差的分钟数。这些值和执行脚本的计算机相关。如果这个方法被一个服务器脚本调用，那么返回值就和服务器相关。而如果这个方法被一个客户机脚本调用，那么返回值就根据客户机上的时间来确定。

如果所在位置的时间落后于 UTC（比如 Pacific Daylight Time），那么这个值就是正值，而如果所在位置的时间超前于 UTC（比如 Japan），那么这个值就是负值。

例如，假设在 12 月 1 日，一台位于 Los Angeles 的客户机与一台位于 New York City 的

服务器进行联络。如果在客户机上执行，getTimezoneOffset 方法将返回 480，而如果在服务器上执行此方法将返回 300。

9. getFullYear 方法

该方法返回 Date 对象中用本地时间表示的年份值。它的语法为：

```
dateobj.getFullYear( )
```

getFullYear 方法以绝对数字的形式返回年份值。例如，1976 年的返回值就是 1976。这样可以避免在处理 20 世纪末的日期的时候发生问题，即“前年虫”问题。

与 getFullYear 方法相对应的用于设置 Date 对象中用本地时间表示的年份值的方法是 setFullYear。

10. toGMTString 方法

该方法返回一个日期，该日期用格林威治标准时间（GMT）表示并已被转换为字符串。它的语法为：

```
dateobj.toGMTString( )
```

toGMTString 方法已经过时，之所以仍然提供这个方法，只是为了提供向后的兼容性。推荐改用 toUTCString 方法。

toGMTString 方法返回一个 String 对象，此对象中包含了按照 GMT 惯例进行格式化的日期。返回值的格式为：05 Jan 1996 00:00:00 GMT。

下面这个例子演示了 toGMTString 方法的用法。

```
function toGMTStrDemo( ){
  var d,s;
  d=new Date( );
  s="Current setting is ";
  s+=d.toGMTString( );
  return(s);
}
```

调用函数 toGMTStrDemo，s 的返回值调用是"Current setting is Wed, 4 Apr 2001 14:17:35 UTC"。

11. toLocaleString 方法

该方法返回一个日期，该日期使用当前国别并已被转换为字符串。它的语法为：

```
dateobj.toLocaleString( )
```

toLocaleString 方法返回一个 String 对象，这个对象中包含了用当前国别的默认格式表示的日期。返回值的格式要根据当前的国别来确定。例如，同样是 1 月 5 日，在美国，toLocaleString 可能会返回"01/05/96 00:00:00"，而在欧洲，其返回值则可能是"05/01/96 00:00:00"，因为欧洲的惯例是将日期放在月份前面。

除了上述方法之外，还有用于设置 UTC 时间和获取 UTC 时间的方法，见表 13.2。

表 13.2 JavaScript 中关于 UTC 时间的处理方法

方法种类	方法名称	方法描述
获得时间	GetMilliseconds	返回当前的毫秒数
	GetUTCDate	返回以格林威治标准时间表达的日期
	GetUTCDay	返回以格林威治标准时间表达的日子
	GetUTCFullYear	返回以格林威治标准时间表达的完整年份
	GetUTCHours	返回以格林威治标准时间表达的小时
	GetUTCMilliseconds	返回以格林威治标准时间表达的毫秒
	GetUTCMnutes	返回以格林威治标准时间表达的分钟
	GetUTCSeconds	返回以格林威治标准时间表达的秒数
设置时间	SetMiliseconds	设置毫秒
	SetUTCDate	设置以格林威治标准时间表达的日期
	SetUTCDay	设置以格林威治标准时间表达的日子
	SetUTCFullYear	设置以格林威治标准时间表达的完整年份
	SetUTCHours	设置以格林威治标准时间表达的小时
	SetUTCMilliseconds	设置以格林威治标准时间表达的毫秒
	SetUTCMinutes	设置以格林威治标准时间表达的分钟
	SetUTCMonths	设置以格林威治标准时间表达的月份
	SetUTCSeconds	设置以格林威治标准时间表达的秒数

下面介绍一个综合使用 Date 对象的程序，它显示的是一个数字时钟。该程序的完整代码如下：

```
<html>
<head>
<title>
时间对象使用演示
</title>
</head>
<body>
<script language="JavaScript">
Stamp=new Date( );
document.write('<font size:"9pt" face="Arial"><b>'+Stamp.getYear( )
+"年"+(Stamp.getMonth( )+1)+"月"+Stamp.getDate( )+"日"+" "+'</b>
</font>');
var Hours;
var Mins;
var Time;
Hours=Stamp.getHours( );
```

```
if(Hours>=12)
  Time="下午";
else
  Time="上午";
if(Hours>12)
  Hours-=12;
if(Hours==0)
  Hours=12;
Mins=Stamp.getMinutes( );
if(Mins<10)
  Mins="0"+Mins;
document.write('<font size:"9pt" face="Arial"><b>'+Hours
+":"+Mins+Time+'</b></font>');
</script>
</body>
</html>
```

该程序的执行结果如图 13.2 **所示。**

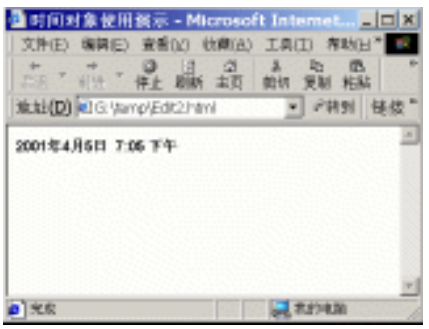

图 13.2　**用** Date **对象制作数字时钟**

13.5　Math **对象**

内置的 Math **对象可以用来进行各种数学运算。其中既定义了一些常用的数学常数，例如** PI=3.1415926**，也定义了很多实现从三角到代数各种运算的方法。**

13.5.1　使用 Math 对象

Math **对象被称为一个静态对象，它从来不用于构造实例，因为其中的方法都是一些大家普遍使用的数学函数，其中的属性都是一些公认的常数值。各种数学运算被定义为** Math **对象的内置方法。**

使用 Math **对象可以采用如下的语法：**

```
Math[.{property|method}]
```

Math 对象不能用 new 运算符创建，如果试图这样做则会给出错误。该对象在装载脚本引擎时由该引擎创建，其所有方法和属性在脚本中总是可用。

提示：Math 对象的所有方法采用的参数均是浮点型。而且，在三角函数里，输入的是弧度值。

13.5.2 Math 对象的属性

Math 对象拥有许多属性，这些属性可以看成是已经定义好的、能够任意使用的一些数学常数。Math 对象的属性见表 13.3。

表 13.3 Math 对象的属性

属性	功能	语法	数值
E	返回 Euler 常数	Math.E	2.718
LN10	返回 10 的自然对数	Math.LN10	2.302
LN2	返回 2 的自然对数	Math.LN2	0.693
LOG2E	返回以 2 为底 e（自然对数的底）的对数	objName.LOG2E	1.442
LOG10E	返回以 10 为底 e（自然对数的底）的对数	objName.LOG10E	0.434
PI	返回圆的周长与其直径的比值	Math.PI	3.14159
SQRTI_2	返回 2 的平方根分之一	Math.SQRT1_2	0.707
SQRT2	返回 2 的平方根	Math.SQRT2	1.414

提示：JavaScript 语句是区分大小写的。例如使用 Pi 值时一定要使用 Math.PI 的写法。对于方法也是如此。

13.5.3 Math 对象的方法

Math 对象拥有的方法见表 13.4。

表 13.4 Math 对象的方法

方法	功能	语法
sin	返回正弦值	Math.sin(*number*)
cos	返回余弦值	Math.cos(*number*)
tan	返回正切值	Math.tan(*number*)
asin	返回反正弦值	Math.asin(*number*)
acos	返回反余弦值	Math.acos(*number*)
atan	返回反正切值	Math.atan(*number*)
exp	返回 e（自然对数的底）的幂	Math.exp(*number*)
log	返回自然对数	Math.log(*number*)
pow	返回底表达式的指定次幂	Math.pow(*base*,*exponent*)
sqrt	返回平方根	Math.sqrt(*number*)

（续表）

方法	功能	语法
ceil	返回不小于参数值的最小整数	Math.ceil(*number*)
floor	返回不大于参数值的最大整数	Math.floor(*number*)
round	返回四舍五入的取整值	Math.round(*number*)
abs	返回参数值的绝对值	Math.abs(*number*)
random	返回一个随机数	Math.random( )
max	返回两个参数中的较大者	retVal=Math.max(*number1*,*number2*)
min	返回两个参数中的较小者	retVal=Math.min(*number1*,*number2*)

13.6　正则表达式对象和 RegExp 对象

JavaScript 新增正则表达式对象（Regular Expression Object），可以用来提供正则表达式的信息。正则表达式原来是一个 UNIX 操作系统中的概念，该表达式总是和一个字符串相联系。它可以用于搜索这个字符串中一个特定的字符组合。这个特定的字符组合以及相关的选项，还有所有待搜索的信息都存放在这个正则表达式对象中，可以使用它所提供的各种方法和属性来得到搜索的信息。

13.6.1　正则表达式对象和 RegExp 对象的语法

正则表达式对象保存用于查找字符串中的字符组合时的模式。创建正则表达式对象后，或者它被传递给字符串方法，或者字符串被传递给一个正则表达式方法。有关最近进行查找的信息被保存在 RegExp 对象中。

正则表达式对象的语法为：

```
var regularexpression=/pattern/[switch]
```

或者为：

```
var regularexpression=new RegExp("pattern",["switch"])
```

其中：

- *pattern*：要使用的正则表达式模式。如果使用第 1 种语法，用/字符分隔模式。如果用第 2 种语法，用引号将模式引起来。
- *switch*：该参数可选，如果使用第 2 种语法，要用引号将 *switch* 引起来。可选的开关选项有：i（忽略大小写）、g（全文查找出现的所有 *pattern*）、gi（全文查找，忽略大小写）。

当预先知道要查找的字符串时用第 1 种语法为好。当要查找的字符串经常变动或不知道时用第 2 种语法，比如由用户输入得到的字符串。

提示：*pattern* 参数在使用前被编译为内部格式。对第 1 种语法来说，*pattern* 在

脚本被装载时被编译。对第 2 种语法来说，在使用 *pattern* 前或调用 compile 方法时被编译。

正则表达式中还包含一些控制命令字符，见表 13.5。

表 13.5 正则表达式控制命令符

字符	描述
\	标记下一个字符是特殊字符或文字。例如，"n"和字符"n"匹配。"\n"则和换行字符匹配。序列"\\"和"\"匹配，而"\("则和"("匹配
^	匹配输入的开头
$	匹配输入的末尾
*	匹配前一个字符 0 或多次。例如，"zo*"与"z"或"zoo"匹配
+	匹配前一个字符一次或多次。例如，"zo+"与"zoo"匹配，但和"z"不匹配
?	匹配前一个字符 0 或一次。例如，"a?ve?"和"never"中的"ve"匹配
.	匹配除换行字符外的任何单个字符
(*pattern*)	匹配 *pattern* 并记住该匹配。匹配上的子字符串，可以使用 Item[0]...[n]来从生成的 Matches 集合中取回。要匹配圆括号字符( )，则需使用"\("或"\)"
x|y	匹配 x 或 y。例如，"z|food"和"z"或"food"匹配。"(z|f)ood"匹配"zoo"或"food"
{*n*}	*n* 是非负整数。共匹配 *n* 次。例如，"o{2}"和"Bob"中的"o"不匹配，但和"foooood"中的前两个"o"匹配
{*n*,}	*n* 是一个非负整数。至少匹配 *n* 次。例如，"o{2,}"和"Bob"中的"o"不匹配，但和"foooood"中的所有 o 匹配。"o{1,}"与"o+"等效。"o{0,}"和"o*"等效
{*n*,*m*}	*m* 和 *n* 是非负整数。至少匹配 *n* 次而至多匹配 *m* 次。例如，"o{1,3}"和"fooooood"中的前 3 个"o"匹配。"o{0,1}"和"o?"等效
[*xyz*]	字符集合。匹配括号内的任一字符。例如，"[abc]"和"plain"中的"a"匹配
[^*xyz*]	否定字符集合。匹配非括号内的任何字符。例如，"[^abc]"和"plain"中的"p"匹配
[*a*-*z*]	字符范围。和指定范围内的任一字符匹配。例如，"[a-z]"匹配"a"到"z"范围内的任一小写的字母表字符
[^*m*-*z*]	否定字符范围。匹配不在指定范围内的任何字符。例如，"[m-z]"匹配不在"m"到"z"范围内的任何字符
\b	匹配字的边界，也就是说，在字和空格之间的位置。例如，"er\b" 和"never"中的"er"匹配，但和"verb"中的"er"不匹配
\B	匹配非字边界。例如，"ea*r\B"和"never early"中的"ear"匹配
\d	匹配数字字符。等价于[0-9]
\D	匹配非数字字符。等价于[^0-9]
\f	匹配换页字符
\n	匹配换行字符
\r	匹配回车符字符
\s	匹配任何空白，包括空格、制表、换页等。与"[\f\n\r\t\v]"等效

（续表）

字符	描述
\S	匹配任何非空白字符。与"[^ \f\n\r\t\v]"等效
\t	匹配制表字符
\v	匹配垂直制表符
\w	匹配包括下划线在内的任何字符。与"[A-Za-z0-9_]"等效
\W	匹配任何非字符。与"[^A-Za-z0-9_]"等效
num	匹配 *num*，其中 *num* 是一个正整数。返回记住的匹配的引用。例如，"(.)\1"匹配两个连续的同一字符
n	匹配 *n*，其中 *n* 是八进制换码值。八进制换码值必须是 1、2 或 3 位长。例如，"\11"和"\011"都匹配制表字符。"\0011"和"\001"&"1"是等效的。八进制换码值必须不超过 256。如果超过了，则只有前两位组成表达式。允许在正则表达式中使用 ASCII 码
\x*n*	匹配 *n*，其中 *n* 是十六进制换码值。十六进制换码值必须正好是两位长。例如，"\x41"与"A"匹配。"\x041"和"\x04"&"1"是等效的。允许在正则表达式中使用 ASCII 码

RegExp 对象用于保存用于正则表达式模式查找的信息。它的语法为：

```
RegExp.propertyname
```

其中参数 *propertyname* 是 RegExp 对象的一个属性。

RegExp 对象不能直接创建，但始终可以使用。在完成成功的正则表达式查找之前其属性的值一直为 undefined。

下面的例子演示了 RegExp 对象的用法。

```
function matchDemo( ){
  var s;
  var re=new RegExp("d(b+)(d)","ig");
  var str="cdbBdbsbdbdz";
  var arr=re.exec(str);
  s="$1 保存:"+RegExp.$1+"<br>";
  s+="$2 保存:"+RegExp.$2+"<br>";
  s+="$3 保存:"+RegExp.$3;
  return(s);
}
```

提示：String 对象和 RegExp 对象都可以处理正则表达式。在 String 对象中可以使用 search 或者 match 方法，但是在 String 对象中存放的是正则查找的目标字符串，而 RegExp 对象中存放的是模式字符串。

13.6.2　RegExp 对象的属性

在 JavaScript 的内部实现中，与正则表达式相关联的字符串，也就是正则表达式的模式，不是采用普通的字符串形式，而是采用一种内部格式表达的。同一般的对象不同的是：

只有一个成功的正则表达式搜索完成后，这个对象的属性才有一定的值。在这样的搜索之前，这个对象可以提供一些属性给其它的对象使用。下面逐一介绍一下这些属性。

1. $1...$9 属性

该属性可以用来返回 9 个在模式匹配期间找到的、最近保存的部分，具有只读属性。其语法为：

```
RegExp.$n
```

其中参数 *n* 是 1 到 9 之间的数。

无论何时产生一个成功的带插入语的匹配，$1...$9 属性的值都会被修改，但是只有最近的 9 个可以被保存起来。

2. index 属性

该属性可以用来返回字符位置，它是查找字符串中第 1 个成功匹配的开始位置。它的语法为：

```
RegExp.index
```

index 属性是基于 0 的。不论何时产生一个成功匹配，它的值都会被修改。

3. input 属性

该属性可以用来返回执行查找的字符串，具有只读属性。它的语法为：

```
RegExp.input
```

任何时候改变了被查找的字符串，input 属性的值都会被修改。

下面的例子演示了 input 属性的用法。

```
function inputDemo(){
  var s;
  var re=new RegExp("d(b+)(d)","ig");
  var str="cdbBdbsbdbdz";
  var arr=re.exec(str);
  s="The string used for the match was "+RegExp.input;
  return(s);
}
```

调用函数 inputDemo，s 的返回值就是"The string used for the match was cdbBdbsbdbdz"。

4. lastIndex 属性

该属性可以用来返回字符位置，它是被查找字符串中最后一次成功匹配的开始位置。其语法为：

```
RegExp.lastIndex
```

lastIndex 属性是基于 0 的，也就是说，第 1 个字符的索引是 0。不论何时产生一个成

功匹配，它的值都会被修改。

lastIndex 属性被 RegExp 对象的 exec 和 test 方法，以及 String 对象的 match、replace 和 split 方法修改。

下面的规则应用于 lastIndex 的值：

- 如果还没有匹配，则 lastIndex 被设置为-1。
- 如果 lastIndex 比字符串的长度大，则 test 和 exec 方法失败，并且 lastIndex 被设置为-1。
- 如果 lastIndex 等于字符串的长度，且模式与空字符串匹配，则正则表达式匹配。否则，匹配失败并且 lastIndex 被重置为-1。
- 除此以外，lastIndex 被设置为紧接最近的匹配的下一个位置。

5. source 属性

该属性可以用来返回正则表达式模式的文本的复本，具有只读属性。它的语法为：

```
RegExp.source
```

下面的例子演示了 source 属性的用法。

```
function SourceDemo(re,s){
  var s1;
  //测试字符串中是否存在正则表达式
  if(re.test(s))
    s1="contains";
  else
    s1="does not contain";
  //获得正则表达式自己的文本
  return(s+s1+re.source);
}
```

13.6.3　正则表达式的方法

正则表达式可以使用的方法如下：

1. compile 方法

该方法把一个正则表达式编译为内部格式。它的语法为：

```
RegExp.compile(pattern)
```

其中：

- *RegExp*：正则表达式对象，可以是变量名或文字。
- *pattern*：字符串表达式，它包含要被编译的正则表达式模式。

compile 方法把 *pattern* 转换为内部格式，从而执行得更快。例如，这使得可以在循环中更有效地使用正则表达式。

下面的例子演示了 compile 方法的用法。

```
function CompileDemo( ){
  var s="AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPp";
  var r=new RegExp("[A-Z]","g");//只为大写字母创建正则表达式
  var a=s.match(r)  //查找匹配
  document.write(a);
  r.compile("[a-z]","g");  //只为小写字母编译正则表达式
  var a=s.match(r)  //查找匹配
  document.write(a);
}
```

调用函数 CompileDemo，a 的输出为"A、B、C、D、E、F、G、H、I、J、K、L、M、N、O、Pa、b、c、d、e、f、g、h、I、j、k、l、m、n、o、p"。

2. exec 方法

该方法在指定字符串中进行查找来获得匹配。它的语法为：

```
RegExp.exec(str)
```

其中：

- *RegExp*：正则表达式对象。可以是变量名或文字。
- *str*：要执行查找的字符串。

exec 方法查找的结果被放在一个数组中。如果 exec 方法没有找到匹配，则它返回 null。如果它找到一个或多个匹配，则 exec 方法返回一个数组，并且更新 RegExp 对象，来反映查找结果。

下面的例子演示了 exec 方法的用法。

```
function ExecDemo( ){
  var s="AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPp";
  var r=new RegExp("g","i");
  var a=r.exec(s);
  document.write(a);
  r.compile("g");
  var a=r.exec(s);
  document.write(a);
}
```

则用函数 ExecDemo，则 a 的输出为"Gg"。

3. test 方法

该方法返回一个 Boolean 值，它指出在被查找的字符串中是否存在模式。它的语法为：

```
RegExp.test(str)
```

其中：

- *RegExp*：正则表达式对象。可以是变量名或文字。
- *str*：要在其上测试查找的字符串。

test 方法检查在字符串中是否存在一个模式，如果存在则返回 true，否则就返回 false。RegExp 对象不由 test 方法来修改。
下面的例子演示了 test 方法的用法。

```
function TestDemo(re,s){
  var s1;
  //检查字符串是否存在正则表达式
  if(re.test(s))
    s1="contains";
  else
    s1="does not contain";
  //获得正则表达式自己的文本
  return(s+s1+re.source);
}
```

13.7　Error 对象

Error 对象用于保存有关错误的信息，它在 JavaScript 中也是常用的。

13.7.1　创建 Error 对象

创建 Error 对象有 3 种方法，其语法分别为：

```
var newErrorObj=new Error( )
var newErrorObj=new Error(number)
var newErrorObj=new Error(number,description)
```

其中：

- *number*：与错误相联的数字值。如果省略则为 0。
- *description*：描述错误的简短字符串。如果省略则为空字符串。

每当产生运行时错误，就产生 Error 对象的一个实例以描述错误。该实例有两个固有属性：保存错误的描述（description 属性）和错误号（number 属性）。

Error 对象也可以用如上所示的语法显式创建，或用 throw 语句抛掉。在两种情况下，都可以添加选择的任何属性，以扩展 Error 对象的能力。

典型地，在 try...catch 语句中创建的局部变量引用隐式创建的 Error 对象。因此，可以按选择的任何方法使用错误号和描述。

下面的例子演示了隐式创建 Error 对象的方法。

```
try  //产生错误
x=y
catch(e){  //创建局部变量 e
  response.write(e)  //打印"[object Error]"
  response.write(e.number&0xFFFF)  //打印 5009
  response.write(e.description)     //打印"'y' is undefined"
}
```

13.7.2 Error 对象的属性

Error 对象的属性只有两个，它们分别是：

1. description 属性

该属性返回或设置与特定错误相联系的描述字符串。它的语法为：

ErrorObj.description[=*stringexpression*]

其中：

- *ErrorObj*：对象的任意实例。
- *stringexpression*：包含错误描述的字符串表达式。

description 属性包含与特定错误相联系的错误信息字符串。使用包含在这个属性中的值，来警告用户发生了一个不能或不想处理的错误。

2. number 属性

该属性返回或设置与特定错误相联系的数字值。Error 对象的默认属性是 number。它的语法为：

ErrorObj.number[=*errornumber*]

其中：

- *ErrorObj*：任意 Error 对象的实例。
- *errornumber*：表示一个错误的整数。

提示：错误号是一个 32 位的值。其中高 16 位字是设备代码，而低 16 位字才是真正的错误代码。

13.8 浏览器对象

由于 JavaScript 语言是在网页中使用的，而网页又必须通过浏览器来实现浏览，所以 JavaScript、网页以及浏览器这三者就构成了密不可分的关系。如同浏览器的设计不能不考虑支持 JavaScript 一样，在 JavaScript 的实现中也必须考虑到对浏览器的操作。下面就向读者介绍 JavaScript 中的浏览器对象。

13.8.1　Netscape Navigator 的对象

本节描述在 Netscape Navigtor 浏览器中使用的对象，并提供这些对象的可在 JavaScript 编程中直接访问的各种有用的属性和方法。

其中 navigator 对象提供关于整个浏览器环境的信息。浏览器对象 navigator 中有用的属性包括：

- appName：字符串形式的浏览器名称。在使用 Navigator 时，appName 的值为“Netscape”；在使用 Internet Explorer 时，appName 的值为“MSIE”。
- appVersion：浏览器的版本号。
- appCodeName：用字符串表示的当前浏览器的代码名字。对于 Navigator 的所有版本，这个值都是“Mozilla”。
- userAgent：浏览器的完整的用户代理标识。
- mimeTypes：在浏览器中可以使用的 MIME 类型信息，其中每一条 MIME 类型由这个数组中的一个 Plugin 对象类型的元素表示。

在 Navigator 4.0 版本中又新增加了一些属性：

- Language：浏览器当前的语言设定。一般这个属性的值是两个字符，例如对于大多数的英语国家是 en，有时这个属性有 5 个字符，例如 zh_CN 表示在大陆使用的中文。这是一个只读属性。
- platform：当前浏览器适用的平台名称。在不同平台下使用的浏览器之间存在一些细微的差别。这也是一个只读属性。

另外，navigator 还拥有一个 JavaEnabled 方法，用于指出在该浏览器中是否可以使用 Java 语言。

13.8.2　Internet Explorer 的对象

虽然 Internet Explorer 浏览器对象和 Netscape Navigator 浏览器对象非常相似，但是仍然在一些差别。如果在用 JavaScript 编程时不注意，就有可能导致一些错误。

下面讨论 IE 中的浏览器对象。在这里仅讨论两种浏览器处理方式不同的部分。

在 IE 中实现了大部分的 Navigator 的对象，但是有以下这些区别：

- IE 没有 Length 属性。
- IE 中的 navigate 方法可以在当前的窗口中打开一个新的 URL。而在 Navigator 中没有这个方法。
- IE 没有实现 status 和 defaultStatus 属性。
- IE 不支持活动链接的颜色设置。在 Navigator 中，当用户单击一个超链接时，这个链接成为活动链接，颜色可以由 JavaScript 中的 document.alinkColor 来设定，而 IE 会忽略这个颜色设定。
- 在 IE 中，pathname 属性不返回路径前的斜线（/）；而 Navigator 则返回这个符号。

除了具有 Navigator 提供的各种属性和方法以外，IE 还额外提供了几种新的属性和方法，例如 enabled 属性可以用来判定一个界面的对象是否有效。

IE 同 Navigator 浏览器最重要的不同体现在浏览器对象 navigator 上，这个对象的 appCodeName，appName，appVersion 和 userAgent 属性值对这两种浏览器来讲其实完全不同。在这些属性中，特别重要的是 appName 属性，可以利用它来判断当前浏览器的类型。在这两个浏览器实现的 JavaScript 语言集合之间还存在某些差异的情况下，在 JavaScript 程序中就要特别注意这些差别。

13.8.3 window 对象

每一个打开的浏览器窗口都对应着 JavaScript 程序中的一个 window 对象。浏览器对象中其他大部分对象都从 window 对象继承而来，所以一般在 JavaScript 程序中可以隐式地引用 window 对象。在这种情况下，浏览器中的 JavaScript 解释程序总是自动地在这样的引用前面加上 window 对象。

window 对象有以下的方法：

- open(*URL,WindowName,parameterList*)：创建一个新的浏览器窗口，并在新窗口中载入一个指定的 URL 地址。
- close( )：方法关闭一个浏览器窗口。
- alert(*text*)：弹出一个信息框。
- confirm(*text*)：弹出一个确认框。
- prompt(*text,Defaulttext*)：弹出一个提示框。
- setTimeout(*expression,time*)：定时设置，在一定时间后自动执行 *expression* 描述的代码，使用 *time* 设置时间，单位是毫秒。
- clearTimeout(*timer*)：取消以前的定时设置。
- setInterval(*expression,time,*[*args*])：设定一个时间间隔，可以反复地自动执行 *expression* 描述的代码，使用 *time* 设置时间，单位是毫秒。
- clearInterval(*timer*)：取消 setInterval 设置的定时。
- moveBy(*horiz,vert*)：将窗口移动指定的位移量，其中水平位移量为 *horiz*，垂直的位移量是 *vert*。
- moveTo(*x,y*)：将窗口放置在指定的坐标处。
- resizeBy(*horiz,vert*)：按照给定的位置量重新设定窗口的大小，其中水平位移量为 *horiz*，垂直位移量是 *vert*。
- resizeTo(x,y)：将窗口设定为指定的大小。
- scrollBy(*horiz,vert*)：按照给定的位移量滚动窗口，其中水平位移量为 *horiz*，垂直位移量是 *vert*。
- scrollTo(*x,y*)：将窗口滚动到指定的位置。
- find([*string*][,true|false][,true|false])：让浏览器在一个网页中查找一个字符串。
- back( )：指示浏览器载入历史记录中的上一个 URL 地址，相当于浏览器工具栏中的后退按钮。

- forward( )：指示浏览器载入历史记录中的下一个 URL 地址，相当于浏览器工具栏中的前进按钮。
- home( )：指示浏览器载入预先设定的主页页面，相当于浏览器工具栏中的主页按钮。
- stop( )：指示浏览器停止网页的装载，相当于浏览器工具栏中的停止按钮。
- print( )：指示浏览器打印当前的网页，相当于浏览器工具栏中的打印按钮。

浏览器对象 window 中还有以下同整个网页结构和信息密切相关的属性：

- frames：一个 frame 对象的数组，记录当前窗口中所有帧的信息。
- staus：浏览器的状态行信息。
- defaultsatus：浏览器默认的状态行信息。
- location：当前窗口的 URL 信息。
- history：当前窗口的历史记录，可以在网页导航中发挥作用。
- closed：一个指出窗口目前是否关闭的逻辑值。
- prent：当前窗口的父窗口。
- self：当前文档对应的窗口。
- top：一系列浏览器上层的浏览器窗口。
- window：当前窗口。
- locationbar：浏览器的地址栏。设定这个属性的 visible 属性值为 true，可以在浏览器中显示地址栏，设定这个 visible 属性值为 false，可以在浏览器中隐藏地址栏。
- menubar：浏览器的菜单栏，使用方法同 locationbar。
- personalbar：浏览器中的目录栏，使用方法同 locationbar。
- scrollbar：浏览器中的滚动条，使用方法同 locationbar。
- statusbar：浏览器中的状态栏，使用方法同 locationbar。
- toolbar：浏览器中的工具栏，使用方法同 locationbar。

下面的程序演示了 window 对象的使用方法。

```
<html>
<head>
<title>
窗口对象程序演示
</title>
</head>
<body>
<script language="JavaScript">
function winopen( ){
  var targeturl="http://www.263.net"
  newwin=window.open("","","scrollbars")
  if(document.all){
    newwin.moveTo(0,0)
```

```
    newwin.resizeTo(screen.width,screen.height)
  }
  newwin.location=targeturl
}
</script>
<form>
<input type="button" onClick="winopen( )" value="263 首都在线"
name="button">
</form>
</body>
</html>
```

该程序的运行结果如图 13.3 所示。

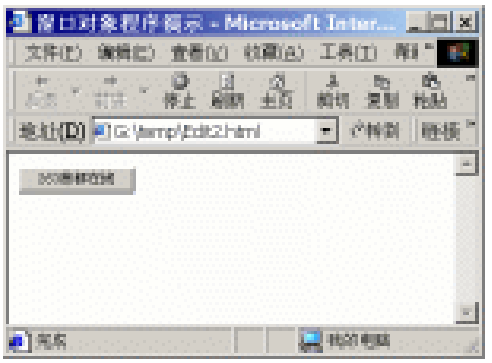

图 13.3 window 对象程序演示

在浏览器中用鼠标单击图中的按钮，就可以使用全屏的方式连接到首都在线网站 http://www.263.net**。**

13.9 小 结

本章详细介绍了 JavaScript 中对象的基本概念和各个内置对象的使用方法。通过本章的学习，读者能够学会如何创建一个对象，而且能够清晰地了解各个对象的属性以及它们的各个方法的使用。掌握这些对象的使用对于 JavaScript 编程来说是至关重要的。了解了对象的创建和使用方法之后，那么对于 JavaScript 的掌握就又上了一个新台阶。

下一章将介绍 JavaScript 的事件处理机制。

第 14 章　JavaScript 的事件处理

JavaScript 与过程编程语言的一个主要不同之处是：过程编程语言按预先规定的顺序执行指令，通常是从第 1 条指令开始一直执行到最后。而 JavaScript 是一种事件驱动的脚本语言。依靠事件处理机制，JavaScript 使用户不仅能接受 Web 页中的信息，而且还可以根据自己的需要作出一定的反应。这样的反应可以被 Web 页中的 JavaScript 脚本程序所接收，并由预定义的事件处理函数来处理。

14.1　事件处理的概念

事件是浏览器响应用户交互操作的一种机制。浏览器本身已有一套已经设计好的用于响应各种事件的方法，但有时我们需要开发自己的应用程序，希望有一种机制可以按照一定的逻辑自主处理各种用户事件。JavaScript 的事件处理机制就可以改变浏览器响应用户操作的标准方法，这样就可以开发出交互性更强、更易使用的 Web 页面。

事件定义了用户与 Web 页面交互时产生的各种操作。例如，单击一个超链接或按钮时，就产生一个事件，告诉浏览器发生了需要进行处理的单击（Click）操作。浏览器在程序运行的大部分时间都等待交互事件的发生，并在事件发生时自动调用相应的事件处理函数，完成事件处理过程。

事件不仅可以在用户与 Web 页面交互过程中产生，浏览器自己的一些动作也可能产生事件。例如，当浏览器载入一个网页的时候，就会产生一个 Load 事件。如果 JavaScript 脚本程序中定义了这个事件的处理函数，浏览器就可以在装入网页的时候自动地执行这个函数。

浏览器为了响应某个事件而进行的处理过程称为事件处理。如前所述，JavaScript 的一个特点就是可以在客户端实现完整的数据处理和验证工作，大量的数据可以在客户端进行处理，从而节省了网络的传输开销。对这些数据的处理，实际上就是对用户“提交数据”这样一个事件的处理。

为了更好地理解事件的概念和事件的处理方法，这里先介绍几个名词。

1．焦点

焦点是一个与程序界面设计有关的概念。例如，对于表单来说，用户使用键盘输入的文本会被自动地发送给表单中的某个界面对象。此时我们称这个界面对象得到了焦点。

2．切换焦点

可以通过鼠标单击一个界面对象从而使这个对象得到焦点，也可以使用键盘上的 Tab 键使界面对象得到焦点。按钮对象在得到焦点后在按钮上会有一个虚框出现，而文本输入框对象在得到焦点后会有一个闪烁的光标，提示用户输入内容。

3．失去焦点

随着一个界面对象得到焦点，那么原来拥有焦点的界面对象就失去了焦点。

4. 句柄

句柄是界面对象的一个属性，用来存储特定的事件处理函数的信息。每当一个事件发生时，JavaScript 解释器就会自动查找界面对象中相应的事件句柄，调用注册在上面的事件处理函数。

一般句柄的形式总是在事件的名称前面加上前缀 on，例如，与 Click 事件相对应的句柄就是 onClick。

在 HTML 中可以采用如下的语法来定义一个句柄（其实句柄就是一个对象的属性）：

```
eventHandler=JavaScript 语句
```

这里的 JavaScript 语句既可以是一个或一串的语句，也可以是一个函数调用。但推荐采用函数调用的形式，因为通常事件处理不是几个语句所能完全解决的，而太长的语句序列会大大影响源文件的可读性，加重浏览器的负担，甚至导致浏览器崩溃。

提示：尽可能地利用函数的形式来定义所有的事件句柄。

例如，目前有一个 JavaScript 函数 Calculate，可以完成一定的计算任务。为了启动这个计算函数，可以在页面上设计一个按钮，单击这个按钮后就可以开始计算。为了实现这样的功能，在 HTML 中使用如下的语句：

```
<input type="button" value="Calculate" onClick="Calculate( )">
```

JavaScript 事件会在 3 个层次上发生——整个 Web 页文档层次、文档中的单个表单层次和文档的某个表格的某个对象层次。同时，这 3 个层次上的任何特定对象都可能导致多个事件。例如，文本项可以根据其操作的不同产生 4 个不同的事件。

14.2 基于浏览器的事件处理

每当新的页面被加载或者旧的页面退出时，就会发生浏览器事件：Load 事件和 Unload 事件。Load 事件产生于新页面加载时，Unload 事件产生于旧页面退出时。此外，还有拖放事件（DragDrop）和提交事件（Submit）。

14.2.1 Load 事件

Load 事件发生在浏览器完成一个窗口或一组帧的装载之后。onLoad 句柄在 Load 事件发生后由 JavaScript 自动调用执行。这个句柄可以在其他所有的 JavaScript 程序和网页之前被执行，可以用来：

- 完成网页中所使用的数据的初始化。
- 弹出一个提示窗口，显示版权或欢迎信息。
- 弹出密码认证窗口，防止非授权的访问。

注意：网页开始显示时并不触发 Load 事件，只有当所有对象（包含图像、applets 等）被加载完成后才触发 Load 事件。理解这一点非常重要。

通常在<body>标记中使用 onLoad 句柄的方法如下：

```
<body onLoad="window.alert('欢迎来到我的主页！')">
```

这样就可以使浏览器载入网页时就执行定义的函数。

同样，当 onLoad 句柄附在<frameset>标记中时，在整个 frameset 被加载后将调用它。要将 onLoad 句柄包含在<frameset>标记中，可采用如下形式：

```
<frameset rows="*,*" onLoad="myFunc("frameset loaded");">
  <frame>
  <frame>
</frameset>
```

下面是一个页面加载时显示欢迎信息的例子。

```
<html>
<head>
<title>
onLoad 事件
</title>
</head>
<body onLoad="welcome( );">
<script language="JavaScript">
function welcome( ){
  window.alert("欢迎来到 vacancy 的主页！");
}
</script>
</body>
</html>
```

当浏览器载入这个页面时，会显示如图 14.1 所示的对话框。

图 14.1　利用 onLoad 事件显示欢迎信息

14.2.2　Unload 事件

Unload 事件发生在用户在浏览器的地址栏中输入一个新的 URL，或者使用浏览器工具

栏中的导航按钮，从而使浏览器试图载入一个新的网页时。在浏览器载入一个新的网页之前，会自动触发一个 Unload 事件，通知原有网页中的 JavaScript 脚本程序。值得注意的是，当用户想在另外的一个新窗口中载入页面时，并不触发浏览器的 Unload 事件。

onUnload 句柄可以完成：

- 在网页退出浏览器前执行必要的清理任务。
- 检查用户的响应情况。例如可以检测用户是否没有提交有关信息，并给出相应的提示。

onUnload 句柄同 onLoad 句柄在事件处理功能上正好相反。使用 onLoad 句柄可以初始化一个网页，而使用 onUnload 句柄则可以结束网页。

onUnload 句柄包含在<body>标记中的情形与 onLoad 句柄相同，如下例所示：

```
<body onLoad="welcome( );" onUnLoad="bye( );">
```

像 onLoad 句柄一样，onUnload 句柄也可以和<frameset>标记相关联，只有在整个 frameset 退出时才会调用 onUnload 句柄。

14.2.3 Submit 事件

Submit 事件在完成信息的输入，准备将信息“提交”给服务器处理时发生。onSubmit 句柄在 Submit 事件发生时由 JavaScript 自动调用执行。

onSubmit 句柄通常在<form>标记中声明。表单中通常有一个提交按钮，单击这个按钮就会在表单上触发一个 Submit 事件，从而自动调用与该事件相对应的句柄。

注意：一个提交按钮同样会产生一个 Click 事件，当这个事件处理完成后（如果有相应的事件句柄），才调用 Submit 句柄。

在 onSubmit 句柄中可以实现最后的数据验证工作。如果所有的数据经验证可以通过，就可以通过返回一个 true 值让 JavaScript 提交表单，把数据发送到服务器。如果有任何一项没有通过，就必须返回一个 false 值，禁止表单发送数据，并给出相关的提示信息，让用户重新输入数据。

下面是一个利用 Submit 事件进行身份验证的例子。

```
<html>
<head>
<title>
onSubmit 事件
</title>
</head>
<body>
<script language="JScript">
<!---
function form1_onSubmit( ){
  var input_name=document.form1.control1.value;
```

```
  if(input_name==""){
    window.alert("必须填写账号");
    document.form1.control1.focus( );
    return false;
}
var checkOk="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
var allValid=true;
for(var i=0;i<input_name.length;i++){
  var ch=input_name.charAt(i);
  var s=checkOk.indexOf(ch);
  if(s==-1)
    allValid=false;
  }
  if(!allValid){
    window.alert("账号必须是字母组成");
    return false;
  }
}
--->
</script>
<form name=form1 onSubmit="form1_onSubmit( )">
<p>
请输入账号
<input name=control1>
<p>
请输入密码
<input name=control2 type=password>
<p>
<input name=b1 type =submit value="提交">
<input id=reset1 name=reset1 type=reset value="重置">
</form>
</body>
</html>
```

这个例子创建一个验证客户身份的页面，该页面中包含了两个单行的文本框，分别让客户输入账号和密码，另外有“提交”和“重置”两个按钮。单击“提交”按钮时，脚本程序按照以下的规则进行客户身份验证：必须在文本框中输入内容，并且只允许输入字母。如果在单击“提交”按钮时，在账号文本框中输入了数字或者其他不符合规定的字符，将会显示出一个消息框，提示输入错误并要求重新输入。

这个例子的执行结果如图 14.2 所示。

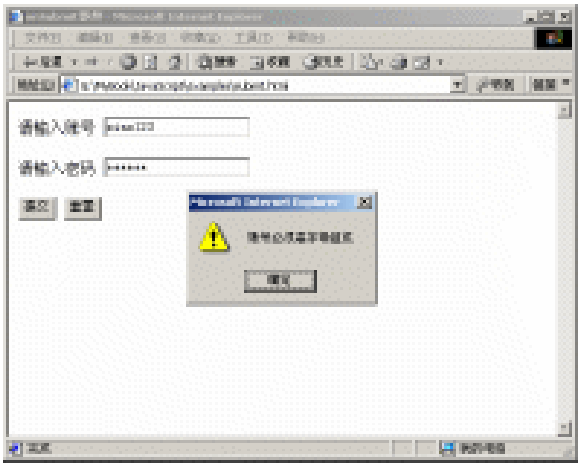

图 14.2 利用 Submit 事件验证客户身份

14.3 基于窗体的事件处理

在客户端 JavaScript 中，与事件相关联的最有用的功能之一是它截获用户操作的能力，这些操作是用户在填充窗体时产生的。

大多数与窗体相关联的 HTML 界面对象支持一个或多个事件处理程序，这依据界面对象的用途而定。表 14.1 列出了所有的界面对象以及它们支持的事件处理程序。

表 14.1 基于窗体的事件处理程序

界面对象	支持的事件处理程序（句柄）
按钮	onClick
复选框	onClick
单选按钮	onClick
列表框	onBlur，onChange，onFocus
文本输入框	onBlur，onChange，onFocus，onSelect
文本区	onBlur，onChange，onFocus，onSelect
复位按钮	onClick
提交按钮	onClick
窗体	onSubmit

14.3.1 Focus 事件

在一个选择框、文本输入框得到焦点的时候发生的事件称为 Focus 事件。onFocus 句柄在 Focus 事件发生后由 JavaScript 自动调用执行。通过鼠标单击对象或者使用 Tab 键都可以使一个区域得到焦点。当然，一个对象得到焦点的同时，另一个对象也就失去了焦点。

对一个文本输入框声明一个 onFocus 句柄的方法如下：

```
<input type="text" size=2 name="COUNTRY" onFocus="SetCountry(this)">
```

下面的例子演示了 onFocus **句柄的使用方法。**

```
<html>
<head>
<title>演示 Focus 事件</title>
<script language="JavaScript">
already=0;
function callAnswer(form){
  if(already==0){
    already=1;
    alert("您是第一次光顾？");
  }
  else
    alert ("老客人了，自己随便看吧。");
}
</script>
</head>
<body>
<form>
<p>这个例子演示了 onFocus 句柄的使用</p>
<p><b>A Nice Day, isn't it? 首先欢迎来到我们的大市场。</b><br>
<p><b>不知道您要点什么?</b><br>
<select name="answer" onFocus="callAnswer(this.FORM)">
<option>一点蔬菜
<option>上好的羊肉
<option>随便看看
</select>
</form>
</body>
</html>
```

当选择列表框时，会弹出一个对话框，告诉这个列表框的作用。在程序里采用了一个小技巧，使得用户在第 1 **次选择和以后选择时，对话框给出不同的提示信息。如图** 14.3 **以及图** 14.4 **所示分别显示了第** 1 **次单击列表框以及多次单击列表框时所给出的不同提示信息。**

14.3.2　Blur 事件

Blur **事件与** Focus **事件的触发条件正好相反。每当用户退出** select、text **或者** textarea **区域时，将触发** Blur **事件，表示该区域失去焦点。**

onBlur **句柄的主要用途之一就是在一个区域的输入完成以后确认其内容。下面的例子让用户依次输入** 4 **门考试课的成绩，如果输入不符合，则给出提示。**

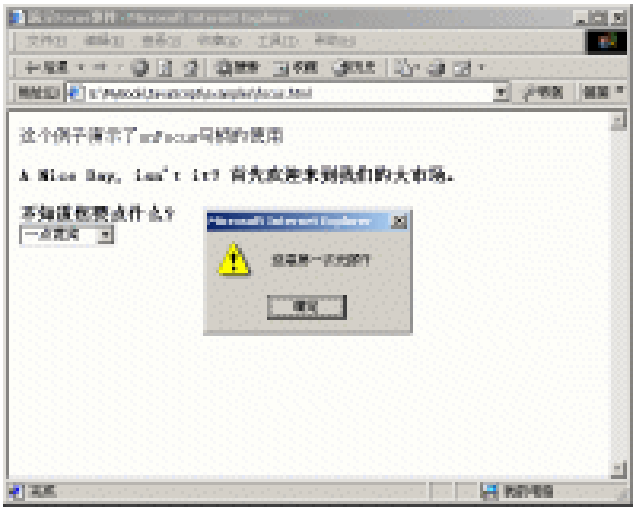

图 14.3　第 1 次访问时的提示信息

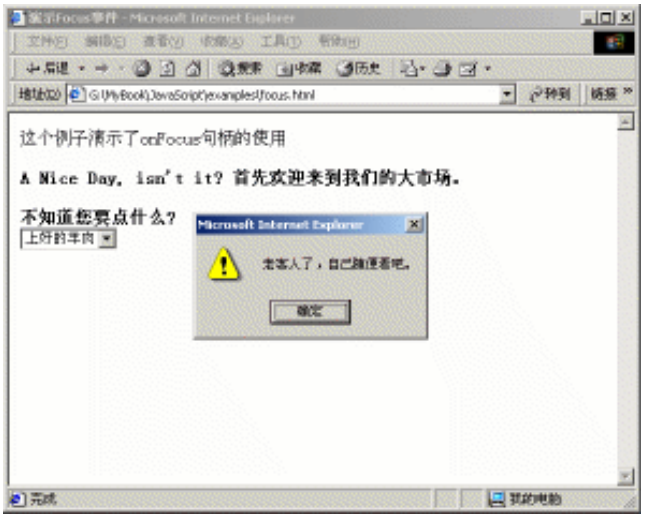

图 14.4　多次访问后的提示信息

```
<html>
<head>
<title>演示 Blur 事件</title>
<script language="JavaScript">
function check(objectdata){
  if(objectdata.value<0||objectdata.value>100){
    window.alert("该项输入不符，请重新输入");
    objectdata.focus( );
  }
}
</script>
```

```
</head>
<body>
成绩录入表单
<br>
<form name="f">
语文：
<input type="text" size=5 name="yuwen"
  onBlur="check(document.f.yuwen)">
<br>
数学：
<input type="text" size=5 name="shuxue"
  onBlur="check(document.f.shuxue)">
<br>
英语：
<input type="text" size=5 name="yingyu"
  onBlur="check(document.f.yingyu)">
</form>
</body>
</html>
```

当用户输入完一项成绩，然后将焦点切换到下一个文本输入框时，程序就会对前一项输入的数据进行验证。这个例子的执行结果如图 14.5 所示。

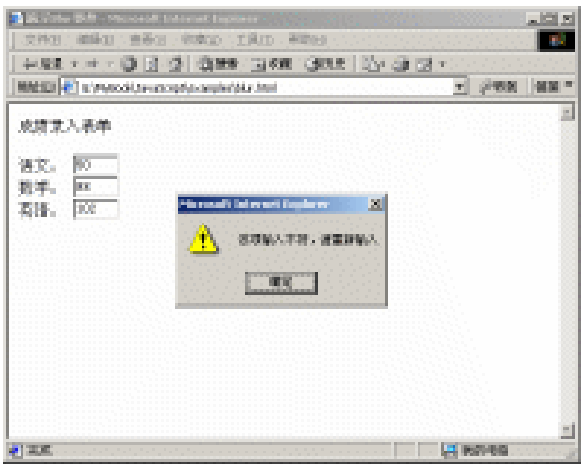

图 14.5　用 Blur 事件进行数据验证

14.3.3　Change 事件

当一个用户退出一个 select、text 或者 textarea 区域时，如果这些域的内容已经改变，将会触发 Change 事件。通过把任何类型为 text 的<input>标记、<select>标记与 onChange 句柄关联在一起，就可以捕获 Change 事件。像 onBlur 事件处理程序一样，onChange 句柄的主要用途是在输入完成以后确认其内容。在下面的例子中，每当用户退出 Email 域并且

域的内容被改变时，就会执行 onChange 句柄。

```
<input type="text" sixe="20" name="Email" onChange="if(this.value
==’’){alert(’请输入 Email’)}">
```

如果用户在没有输入一个值的情况下从输入域中将焦点移开，这个句柄会产生如图 14.6 所示的效果。

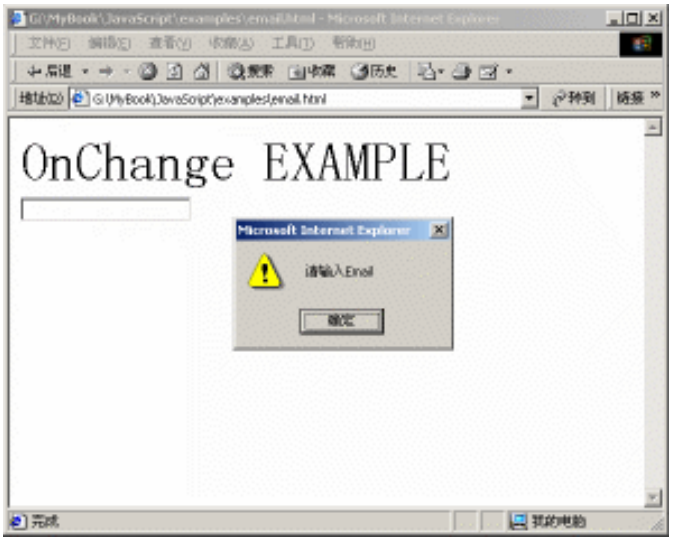

图 14.6 处理 Change 事件

14.3.4 Select 事件

当用户选取或加亮在 text 或 textarea 区域中的任何文本时，就会触发 Select 事件。为了把 onSelect 句柄包含在<textarea>标记中，应使用如下所示代码：

```
<textarea name="textitem" rows=6 cols=55 onSelect="selectText( )">
选择该区域
内的文本
会产生 Select
事件！
</textarea>
```

每当用户选取 textitem 区域中的任何文本时，由 onSelect 句柄指定的 selectText 函数将会被调用。

14.3.5 Move 事件

当用户手工或者脚本程序移动一个窗口或者一个帧的时候触发 Move 事件。这个事件发生后由 JavaScript 自动调用 onMove 句柄。这个句柄适用于窗口以及帧。

传递给 onMove 句柄的 Event 参数中：

- type：指示当前发生的是一个 Move 事件。
- x 和 y：指示当前窗口或者帧完成移动后在屏幕上的位置。

14.3.6　Resize 事件

用户手工或者脚本程序移动一个窗口或者一个帧的时候发生 Resize 事件。这个事件发生后由 JavaScript 自动调用 onResize 句柄。这个句柄适用于浏览器对象 document 以及帧。

传递给 Resize 事件处理函数的 Event 参数中：

- type：指示当前发生的是一个 Resize 事件。
- x 和 y：指示当前窗口或者帧的水平和垂直尺寸的实际大小，以像素为单位。

14.4　基于鼠标的事件处理

处理鼠标的事件很多，限于篇幅的关系，这里只介绍常用的 MouseDown，MouseMove，MouseUp，MouseOver，MouseOut，Click，Blur 以及 Focus 事件。

当鼠标事件发生的时候，JavaScript 解释器会自动把事件信息填充到一个 Event 对象的实例中，作为一个参数传送给鼠标事件处理函数。鼠标事件处理函数得到的 Event 对象有如下一些共同的属性：

- type：一个字符串，指示这个事件类型。对于鼠标事件，可能的属性值有 MouseDown、MouseUp 和 Click 等。
- layerX：指示当事件发生的时候鼠标相对于当前层的水平位置，以像素为单位。
- layerY：指示当事件发生的时候鼠标相对于当前层的垂直位置，以像素为单位。
- pageX：指示当事件发生的时候鼠标相对于当前页的水平位置，以像素为单位。
- pageY：指示当事件发生的时候鼠标相对于当前页的垂直位置，以像素为单位。
- screenX：指示当事件发生的时候鼠标相对于当前屏幕左上角的水平位置，以像素为单位。
- screenY：指示当事件发生的时候鼠标相对于当前屏幕左上角的垂直位置，以像素为单位。
- which：指示鼠标按下的键的信息。1 表示按下的是鼠标的左键，3 表示按下的是鼠标的右键。
- Modifiers：指示随着鼠标按键同时按下的键盘修饰键的种类。这个属性可以取的值有 ALT_MAS，CONTROL_MASK，SHIFT_MASK 和 META_MASK。

应该注意的是，上面的属性是为 Navigator 设计的，有些属性在 IE 中不能使用。但是 IE 中也有相应的属性。可以通过下面的例子查看 IE 中 Event 对象的所有属性。

```
<html>
<head>
<title>
Event 对象
```

```
</title>
</head>
<body onMouseDown="evt( )">
<script language="JavaScript">
function evt( ){
  for(i in event)
    document.write(i+"<br>");
}
</script>
</body>
</html>
```

这个例子的执行结果如图 14.7 **所示。**

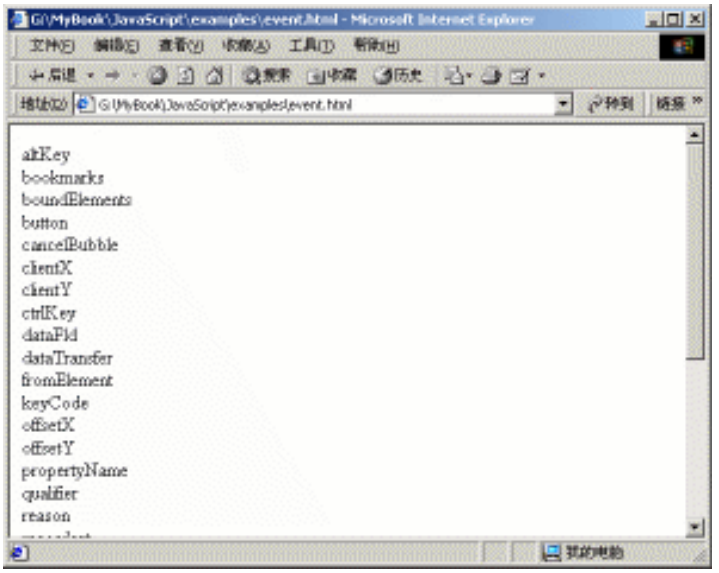

图 14.7 **显示** IE **中** Event **对象的所有属性**

在 JavaScript **程序中，可以使用下面的方法来判定伴随一个事件的发生是否有一个键盘修饰键被按下。**

```
if (e.Modifiers&Event.ALT_MASK)
  alert("Alt key pressed!!");
```

其中，e **是一个传送到事件处理函数中的** Event **对象。**

14.4.1 MouseDown 事件

当按下鼠标上一个键的时候发生 MouseDown **事件，在这个事件发生后由** JavaScript **自动调用** onMouseDown **句柄。这个句柄适用于网页、普通按钮以及超链接。**

如果 onMouseDown **句柄返回** false **值，同鼠标操作密切相关的其他操作，例如拖放、**

选定文本以及激活超链接等都无效，因为这些操作首先都必须触发这个事件。JavaScript 如果发现一个句柄的返回值为 false，就中止事件的继续处理。

下面的例子中，只要用户在页面内按下鼠标的一个键，就会显示相应的信息。

```
<html>
<head>
<title>
鼠标事件
</title>
</head>
<body onMouseDown="mousedw( )">
<script language="JavaScript">
function mousedw( ){
  document.writeln("发生了鼠标事件"+event.type+"<br>");
  document.writeln("鼠标的位置 x="+event.x+"<br>");
  document.writeln("鼠标的位置 y="+event.y+"<br>");
  return true;
}
</script>
</body>
</html>
```

这个例子的执行结果如图 14.8 所示。

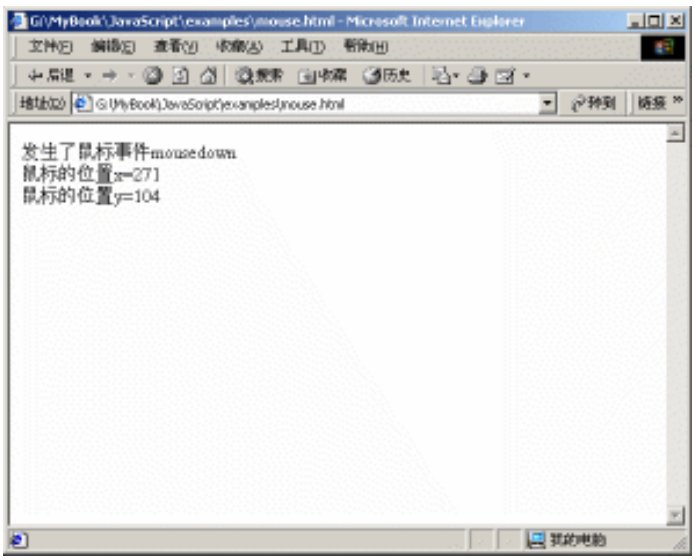

图 14.8　处理 MouseDown 事件

14.4.2 MouseMove 事件

移动鼠标的时候触发 MouseMove 事件，在这个事件发生后由 JavaScript 自动调用 onMouseMove 句柄。

MouseMove 事件不从属于任何界面对象。只有当一个对象（浏览器对象 window 或者 document）要求捕获事件的时候，这个事件才会在每一次鼠标移动时触发。

下面的例子演示了 MouseMove 事件的使用方法。当鼠标移动时，在状态栏中动态显示鼠标的位置。

```
<html>
<head>
<title>
鼠标移动
</title>
</head>
<body onMouseMove="mousemv( )">
<script language="JavaScript">
function mousemv( ){
  x=event.x;
  y=event.y;
  window.status="当前鼠标位置：x="+x+"   y="+y;
}
</script>
</body>
</html>
```

这个例子的执行结果如图 14.9 所示。

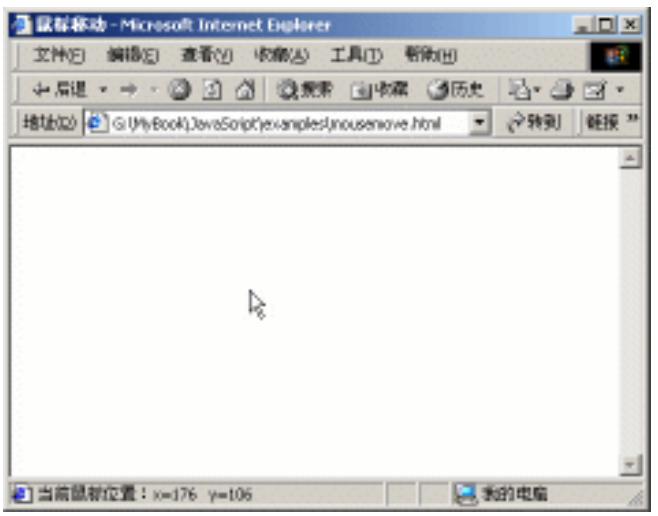

图 14.9 动态显示鼠标的位置

14.4.3　MouseUp 事件

释放鼠标上一个键的时候触发 MouseUp 事件，在这个事件发生后由 JavaScript 自动调用 onMouseUp 句柄。这个句柄适用于普通按钮、网页以及超链接。

同 MouseDown 事件一样，如果 onMouseUp 句柄的返回值为 false，同鼠标操作密切相关的其他操作，例如拖放、选定文本以及激活超链接等都无效，因为这些操作必定要触发 MouseUp 事件。JavaScript 如果发现一个句柄的返回值为 false，就中止事件的继续处理。

14.4.4　MouseOver 事件

当鼠标扫过一个界面对象时触发 MouseOver 事件，在 MouseOver 事件发生时由 JavaScript 自动调用执行 onMouseOver 句柄。这个句柄适用于区域、层以及超链接。

14.4.5　MouseOut 事件

当鼠标扫过并脱离一个界面对象时触发 MouseOut 事件。onMouseOut 句柄在 MouseOut 事件生时由 JavaScript 自动调用执行。这个句柄适用于区域、层以及超链接。

如果在 JavaScript 脚本程序中使用了 onMouseOut 句柄，就意味着浏览者每次扫过一个超链接时都要被动地接受一些信息，而这对于某些用户来说是不可容忍的。所以在使用这个句柄时一定要有创意，给出的信息一定要有吸引力，否则用户一定会厌烦这样的网页。

利用 MouseOver 和 MouseOut 事件可以制作出具有动感的图像按钮，即当鼠标在按钮上时显示一幅图片，而当鼠标不在按钮上时则显示另一幅图片。如下面的例子所示：

```
<html>
<head>
<title>
显示不同的图片按钮
</title>
</head>
<body>
<img name="pic" src="button0_friends.gif"
  onMouseOver="document.pic.src='button1_friends.gif'"
  onMouseOut="document.pic.src='button0_friends.gif'">
</body>
</html>
```

当鼠标不在图片上时，显示的效果如图 14.10 所示。而如果将鼠标移到图片之上，则显示效果会改变，如图 14.11 所示。

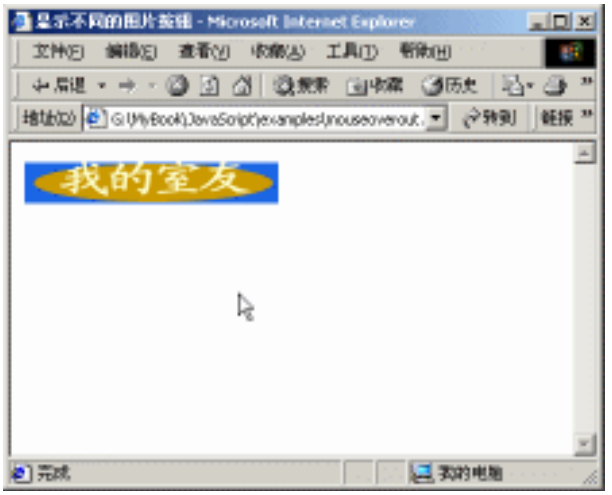

图 14.10 鼠标不在图片上时的显示效果

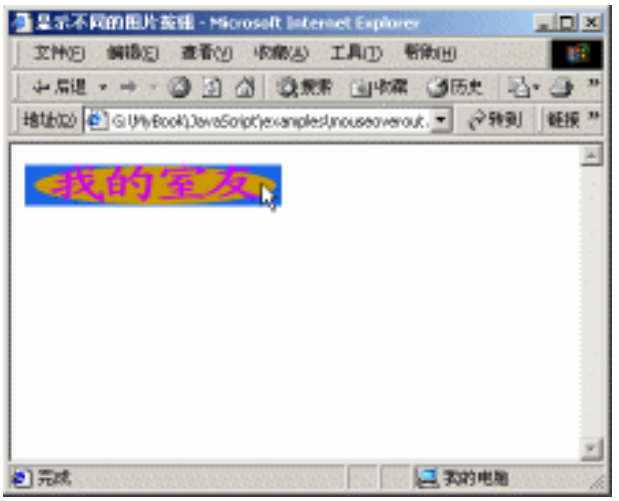

图 14.11 鼠标在图片上时的显示效果

14.4.6 Click 事件

当一个表单中的某个对象被单击时可以触发 Click 事件。onClick 句柄在 Click 事件触发时由 JavaScript 自动调用执行。onClick 句柄适用于普通按钮、复位按钮、提交按钮、单选按钮、复选框以及超链接。onClick 句柄得到的 Event 对象拥有前面提到的一切公共属性。

注意：通过给一个 onClick 句柄返回 false 值，将会取消这个单击动作，就好像没有单击过一样，这个特性同 MouseDown 以及 MouseUp 事件的句柄是一样的。

鼠标是用户浏览网页时最常用的，甚至比键盘还常用，因此设计出好的鼠标效果是使一个网页吸引人的关键。下面是一个使文字跟随鼠标移动的例子，当鼠标移动时，文字显得非常有动感。

```
<style type="text/css">
  spanstyle{
    position:absolute;
    visibility:visible;
    top:-50px;
    font-size:9pt;
    color:#000000;
    font-weight:bold;
}</style>
<script>
var x,y
var step=20
var flag=0
var message="文字在跟随着鼠标跳舞哦！"
message=message.split("")
var xpos=new Array( )
for(i=0;i<=message.length-1;i++)
  xpos[i]=-50
    var ypos=new Array( )
    for(i=0;i<=message.length-1;i++)
      ypos[i]=-50
  function handlerMM(e){
    x=(document.layers)?e.pageX:document.body.scrollLeft+event.clientX
    y=(document.layers)?e.pageY:document.body.scrollTop+event.clientY
    flag=1
}
function makesnake( ){
if(flag==1&&document.all){
  for(i=message.length-1;i>=1;i--){
    xpos[i]=xpos[i-1]+step
    ypos[i]=ypos[i-1]
  }
  xpos[0]=x+step
  ypos[0]=y
  for(i=0;i<message.length-1;i++){
  var thisspan=eval("span"+(i)+".style")
  thisspan.posLeft=xpos[i]
  thisspan.posTop=ypos[i]
}
}
else if(flag==1&&document.layers){
    for(i=message.length-1;i>=1;i--){
      xpos[i]=xpos[i-1]+step
```

```
      ypos[i]=ypos[i-1]
    }
    xpos[0]=x+step
    ypos[0]=y
    for(i=0;i<message.length-1;i++){
    var thisspan=eval("document.span"+i)
    thisspan.left=xpos[i]
    thisspan.top=ypos[i]
  }
}
  var timer=setTimeout("makesnake( )",30)
}
</script>
<body onLoad="makesnake( )" >
<script>
for(i=0;i<=message.length-1;i++){
  document.write("<span id='span"+i+"'class='spanstyle'>")
  document.write(message[i])
  document.write("</span>")
}
if(document.layers)
  document.captureEvents(Event.MOUSEMOVE);
  document.onmousemove=handlerMM;
</script>
```

这个例子的执行结果如图 14.12 所示。

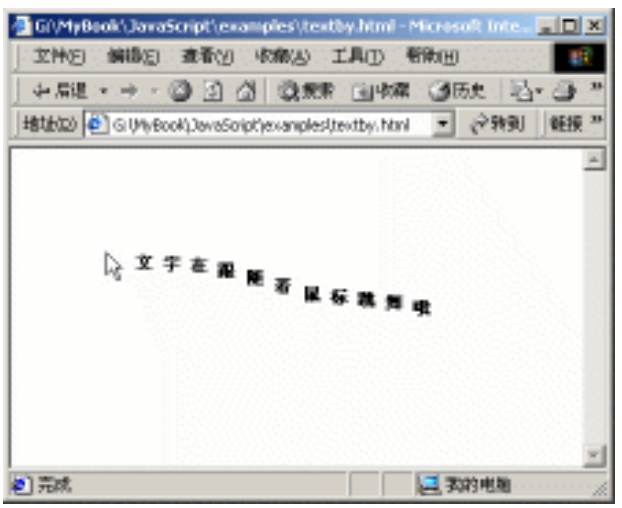

图 14.12 文字跟随鼠标移动

14.5　基于键盘的事件处理

键盘事件和鼠标事件很相似，常用的键盘事件有 KeyDown、KeyPress 和 KeyUp。JavaScript 解释器传送给键盘事件处理函数的 Event 对象有一些共同的属性。

14.5.1　KeyDown 事件

在键盘上按下一个键的时候触发 KeyDown 事件，在这个事件触发后由 JavaScript 自动调用 onKeyDown 句柄。这个句柄适用于浏览器对象 document、图像、超链接以及文本区域。

提示：KeyDown 事件总是在 KeyPress 事件之前触发，如果 onKeyDown 句柄的返回值为 false，就不会触发随后的 KeyPress 事件。

14.5.2　KeyPress 事件

当按下键盘上一个键的时候触发 KeyPress 事件，在这个事件触发后由 JavaScript 自动调用 onKeyPress 句柄。这个句柄适用于游览器对象 document、图像、超链接以及文本区域。

只有当 JavaScript 中 onKeyDown 句柄的返回值为 true 的时候，KeyPress 事件才可能紧接着触发，而且如果用户一直按着相同的键，就可能不断地触发这个事件。

14.5.3　KeyUp 事件

在键盘上释放一个键的时候触发 KeyUp 事件，在这个事件触发后由 JavaScript 自动调用 onKeyUp 句柄。这个句柄适用于浏览器对象 document、图像、超链接以及文本区域。

下面举一个简单的例子说明键盘事件的使用。在这个例子中，如果用户按下了加号（+）号或者减号（-）号，就会显示相应的信息。

```
<html>
<head>
<title>
key
</title>
</head>
<body onKeyDown="evt( )">
<script language="JavaScript">
function evt( ){
  if(event.keyCode==187)  //按下了+号
    document.f.str.value="你按下了+号";
  if(event.keyCode==189)  //按下了-号
    document.f.str.value="你按下了-号";
  return;
}
```

```
</script>
<form name="f">
<input type="text" name="str" size=20>
</form>
</body>
</html>
```

当按下＋号时，文本输入框中就显示“你按下了＋号”，如图 14.13 **所示。**

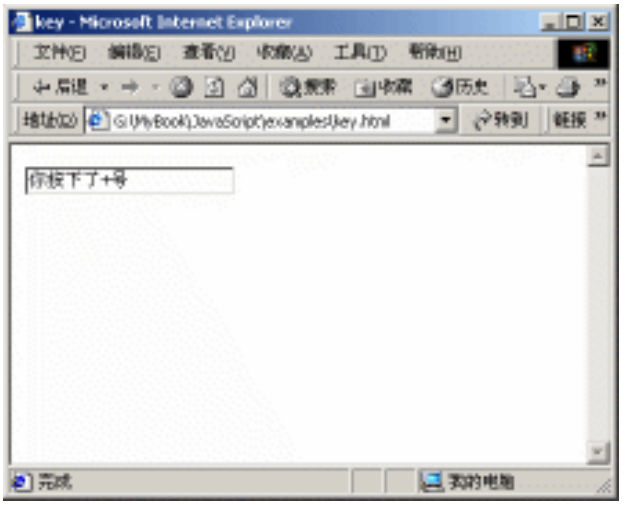

图 14.13 **按键时显示相应的提示信息**

14.6 小 结

本章介绍了 JavaScript **中事件的概念和事件句柄的使用。由于** JavaScript **是一种事件驱动的语言，因此学会常用的事件处理方法是掌握** JavaScript **的关键。**

第 15 章　JavaScript 的高级使用技巧

介绍完 JavaScript 的语法、对象、事件处理机制等内容后，本章将向读者介绍 JavaScript 的高级使用技巧，包括 frame 对象、Cookie 和 AcitveX 等。通过本章的学习，读者可以将 JavaScript 的编程技巧提高到一个新的层次。

15.1　frame 对象

frame（帧）是 HTML 中引入的一个新概念。有了帧以后，我们就可以将浏览器划分为好几个矩形区域，每个矩形区域就叫做 frame。在网页中使用帧，可以实现对网页的一部分进行更新，而其余部分仍然保持不变。

15.1.1　使用 frame

frame 其实是单独的窗口，它对应于单独的窗口对象，有自己的 location、history 和 document 属性。

在 HTML 中一般用<frameset>和<frame>标记来生成帧。还有一个标记<noframes>来向不支持帧的浏览器提供解决方案。

创建 frame 对象的语法如下：

```
<frameset rows="  ">
  <frame name=" " src=" ">
</frameset>
```

其中各个标记后面的属性值可选。

<frameset>标记具有如下的属性：

- rows：指明两个帧在垂直方向上的位置，同样用占据整个浏览器窗口尺寸的百分数来表示。
- cols：指明两个帧在水平方向上的位置，用占据整个浏览器窗口尺寸的百分数来表示。
- border：设定边界的粗细。当这个属性设置为 0 时，意味着帧没有边界。

提示：在 rows 和 cols 中设置时可以采用*号，它的意思是不具体指明帧的大小，而让它填满浏览器窗口的剩余空间。

<frame>标记拥有如下的属性：

- name：帧的名称。

- src：载入的网页文件。
- marginwidth：边框的宽度。
- scrolling：可以设为 yes 或 no。如果设置为 yes，则表示允许在适当的时候滚动帧。
- noresize：指出不能改变帧的大小。

提示：<noframes>和</noframes>中的内容将被 Navigator 等支持 frame 对象的浏览器所忽略。

下面的程序创建了含有 4 个帧的页面布局。

```
<html>
<head>
<title></title>
</head>
<frameset rows="300,*">
  <frame name="a" src=" ">
  <frameset cols="33%,33%,33%">
    <frame name="b" src=" ">
    <frame name="c" src=" ">
    <frame name="d" src=" ">
  </frameset>
</frameset>
</html>
```

帧对象同样也是一个浏览器对象。浏览器对象的最顶层是 window 对象，其他对象都是从 window 对象继承而来。HTML 中将帧的 name 属性的值定义为浏览器对象 windows 的属性名称。如果在 HTML 中没有定义 name 属性，也可以通过浏览器对象中的帧数组来访问一个帧。例如在上面的程序中可以使用：

```
window.a
```

或者

```
windows.frames[0]
```

来访问同样的一个帧。
如果帧中包含有界面对象，可以通过下面的形式来访问这些界面对象：

```
a.button.value
```

或者

```
window.a.button.value
```

其中浏览器对象是可以省略的。从这里可以看出，帧在网页中的位置相当于一个窗口，而且在 JavaScript 中的使用方法也非常类似于一个窗口。

15.1.2　frame 对象的属性和方法

frame 对象的属性主要有：

- name：帧的名称。
- parent：当前帧的父窗口。
- self：当前帧本身。

帧在网页中的位置基本同 window 对象类似，所以可以大致列出 frame 对象的各种方法：

- blur( )：使当前帧失去焦点位置。
- focus( )：使当前帧得到焦点。
- setInterval(*expression*,*time*,[*args*])：定义一个定时器，每隔一定时间自动执行指定的操作。
- clearInterval(*timer*)：清除 setInterval 方法的定时器设置。
- setTimeout(*expression*,*time*)：定义一个定时器，每隔一定时间进行一次操作。
- clearTimeout(*timer*)：清除 setTimeout 方法的定时器设置。

使用 frame 对象的属性和方法只能改变当前帧的内容，而不能改变网页中其他帧的内容。有时候可能需要从帧中访问帧的父窗口的 window 对象，这时使用 window 对象的各种方法，就可以让浏览器重新载入一个不含帧的网页。frame 对象的属性 parent 就是为了这个目的而设置的，在帧中可以使用这个属性载入一个新的网页，在当前帧中可以使用

```
parent.location.href="http://……"
```

如果在访问一个帧 frame1 的同时想访问另一个帧 frame2，那么可以按如下所示的方式来访问：

```
parent.frame2.document.write("……")
```

15.1.3　frame 的应用实例

下面通过一个具体的实例来说明如何在 JavaScript 脚本中利用 frame 对象在 Web 中实现更为复杂的信息交互。该例子在多个窗口中实现窗体信息的动态交互。程序首先在浏览器窗口中制作 3 个用于交互的窗口，每个窗口实现不同信息的动态交互。

本示例由 4 个文档组成，分别介绍如下：

（1）example.html 为主调用文档。它首先将窗口划分为具有两行的窗体，然后再将第 2 行的窗体划分为具有两列的窗体。其内容如下：

```
<html>
<head>
</head>
<frameset rows="20%,90%">
  <frame src="example_1.html">
<frameset cols="40%,50%">
```

```
  <frame src="example_2.html">
  <frame src="example_3.html">
</frameset>
</frameset>
</html>
```

（2）example_1.html 构成第 1 个框架，主要作用是显示标题文档。其内容如下：

```
<html>
<head>
</head>
<h2>使用框架实现 WEB 交互</h2>
</html>
```

（3）example_2.html 构成第 2 个框架，主要作用是实现交互。其内容如下：

```
<html>
<head>
</head>
<body>
<form name="example_1">
请选择学校：<br>
<select name="select1" multiple>
  <option>a 中学
  <option>b 中学
  <option>c 中学
  <option>d 中学
  <option>e 中学
  <option>f 中学
  <option>g 中学
  <option>h 中学
</select>
<br>
<hr>
<input type="submit" name="" value="确定">
<input type="reset" name="" value="重置">
</form>
</body>
</html>
```

（4）example_3.html 构成第 3 个框架，主要作用也是实现交互。其内容如下：

```
<html>
<head>
</head>
```

```
<body>
<form name="example_2">
请输入用户名：
<input type="text" name="text1" value="" size=20>
<br>
<hr>
请选择：
<input type="checkbox" name="checkbox1" value="qb">全部信息
<input type="checkbox" name="checkbox2" value="bf">部分信息<br>
<hr>
<input type="submit" name="" value="确定">
<input type="reset" name="" value="重置">
<br>
</form>
<script language="JavaScript">
document.example_2.elements[0].value="黎明";
document.example_2.elements[1].checked=true;
document.example_2.elements[2].checked=true;
</script>
</body>
</html>
```

在浏览器中的结果如图 15.1 所示。

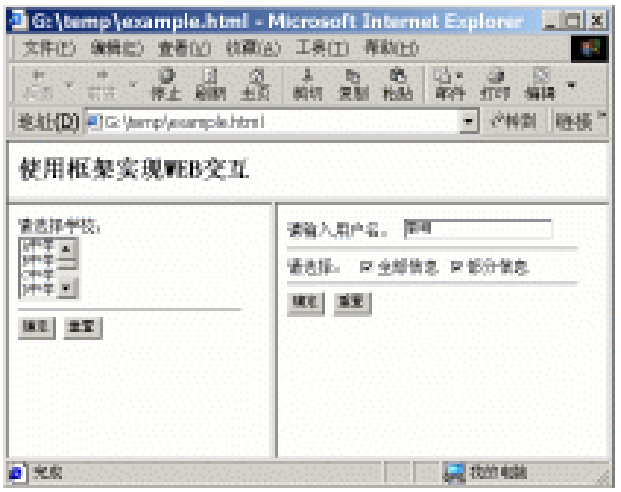

图 15.1　frame 实例在浏览器中的结果

15.2 Cookie 的使用

Cookie 提供了在浏览器中维持客户端状态信息的方法。Cookie 在标准的 HTTP 和 CGI-BIN 环境下工作，但它并不是 JavaScript 所独有的。

15.2.1 Cookie 简介

Cookie 是在 Web 中用于存储客户系统信息的对象。所有的信息都以每行一个 Cookie 的形式存放在客户端的一个名为 cookie.txt 的文件里。Cookie 在 HTTP 头标（客户和服务器用来标识自身的分组）中在客户机与服务器之间传输。

Cookie 由某个 Web 页在客户机上进行设置。比如，某个 Web 页已在一个用户的计算机上设置了一个 Cookie，其中存储的信息是该用户的身份号（随机赋予该用户的唯一标识），当该用户的浏览器连接该 Web 站点时，站点要求浏览器将 Cookie 送回，他的身份号就通过 Cookie 传递给该网页所在的 Web 服务器。服务器上的一个 CGI 程序查找服务器端的一个文件以确定关于该用户的预设内容。

当某个服务器在客户机上设置 Cookie 后，请注意如果要让 Cookie 信息确实写入文件，必须关闭浏览器。在浏览器未关闭之前，任何新的或变化的 Cookie 都存放在内存中。

15.2.2 Cookie 的属性

Cookie 常用的属性有 name，value，expires，domain 和 secure。这些属性存放了某个 Cookie 的作用范围及实际数据。

1. name 属性

这是每一个 Cookie 都具有的属性，是该 Cookie 的名字。name 属性是一个不含分号、逗号和空格的字符串，其命名方式与变量命名相同。

2. value 属性

value 也是每一个 Cookie 都具有的属性，是 Cookie 的值。value 属性是实际存放于 Cookie 中的信息。它是由任何字符构成的字符串。

3. expires 属性

expires 是 Cookie 的过期时间。没有设置 expires 属性的 Cookie 在用户断开连接后过期，但在用户关闭浏览器之前该 Cookie 依然存在。

Cookie 有一个过期时间并不等于它会被从 cookie.txt 文件中删除。在它的位置被用来存放另一个 Cookie 前，它依然存在。过期的 Cookie 只是不被送往要求使用它的服务器。

expire 属性的形式如下：

```
Wdy,DD-Mon-YY HH:MM:SS GMT
```

expires 属性是可选的，所有日期都按格林威治标准时间存储。

4. domain 属性

domain 用于设置某个 Cookie 的 Web 页所在的计算机的域名。这样，由一个站点创建的 Cookie 不会影响到另一个站点上的程序。对于较高层的域名如.com、.edu 或.mil，域名中至少有两个分隔符（.）。而对于较低层的域名如.cn、.uk 或.ca，域名中至少有 3 个分隔符。domain 属性被自动设置为网页所在站点的基本域名。

例如，如果网页位于 http://www.dscga.com/～user，则该网页创建的 Cookie 默认情况下对域 dscga.com 有效。如果希望自己的 Cookie 只应用于服务器 www3.dscya.com，那么就必须在设置 Cookie 的时候指定。

只有拥有域名的站点才能为那个域名设置 Cookie。

5. path 属性

一个 Cookie 可以被指定为只针对一个站点的某一层次。如果一个 Web 站点要区分已注册的和未注册的用户，就可以为已经注册的用户设置 Cookie，当注册过的用户访问该站点时，他就可以访问只对注册用户有效的页面。

path 属性是可选项，如果没有指定 path，其默认值为设置 Cookie 的页面的路径。

6. secure 属性

secure 是一个布尔值（true 或 false），它的默认值为 false。如果它被设为 true，这个 Cookie 将只被浏览器认为是安全的服务器所记住。

15.2.3　Cookie 的限制

一个站点能为一个单独的用户最多设置 20 个 Cookie。如果一个站点有两个服务器（比如 www.dscga.com 和 www3.dscga.com），但是没有指定域名，则 Cookie 的域名默认为 dscga.com。如果指定了确切的服务器地址，则每个站点可以设置 20 个 Cookie，而不是总共 20 个。

不仅每个服务器能设置的 Cookie 的数目是有限的，而且每个客户机最多只能存储 300 个 Cookie。如果一个客户机已有 300 个 Cookie，并且一个站点在它上面又设置了一个新 Cookie，那么，先前存在的某一个 Cookie 将被删除。

每个 Cookie 也有自身的大小限制。Cookie 不得超过 4KB（4096 bytes），其中包括名字和其他信息。

15.2.4　JavaScript 和 Cookie

现在读者已经了解有关 Cookie 的一些基本概念了，让我们更进一步讨论 Cookie。可以用 JavaScript 来很方便地编写函数用来创建、读取和删除 Cookie。下面就逐一介绍这些函数。

1. 创建 Cookie

使用 Cookie 首先要创建一个 Cookie。下面给出的 SetCookie 函数将完成这一功能。程序如下：

```
function  SetCookie(name,value){
var argv=SetCookie.arguments;
```

```
var argc=SetCookie.arguments.length;
var expires=(argc>2)?argv[2]:null;
var path=(argc>3)?argv[3]:null;
var domain=(argc>4)?argv[4]:null;
var secure=(argc>5)?argv[5]:false;
document.Cookie=name+"="+escape(value)+((expires==null)?"":(";
  expires="+expires.toGMTString( ))+((path==null)?"":(";path="
  +path))+((domain==null)?"":(";domain="+domain))+((secure==
  true)?";secure":"");
}
```

SetCookie 函数只要求传递被设置的 Cookie 的名字和值，但如果必要的话可以设置其他 4 个参数而不必改变这个函数。可选的参数必须按正确的次序使用。如果不想设置某个参数，必须设置一个空串。

如果创建的一个 Cookie 需要指定 secure 属性的值，但不想设置 expires、path 或 domain，就可以如下所示调用 SetCookie：

```
SetCookie("MyNewCookie","MyValue","","","tyue);
```

2. 读取 Cookie

下面给出的函数 GetCookie 用来读取一个 Cookie。当一个 Cookie 的请求被客户机收到时，客户机查找它的 cookie.txt 文件以进行匹配。GetCookie 函数首先匹配这个 Cookie 的名字。如果有多个同名的 Cookie，再匹配路径。函数完成匹配后返回这个 Cookie 的值。如果客户机中没有这个 Cookie，或者路径不匹配，该函数返回一个 NULL。

```
function GetCookie(name){
  var arg=name+"=";
  var alen=arg.length;
  var clen=document.cookie.length;
  var i=0;
  while(i<clen){
    var j=i+alen;
    if(document.cookie.substring(i,j)==arg)
      return GetCookieVal(j);
    i=document.cookie.indexOf("",i)+1;
    if(i==0)
      break;
  }
  return null;
}
```

函数中使用了 GetCookieVal 函数，它是 GetCookie 函数的补充。GetCookieVal 将 cookie.txt 文件分解为片断，并从一批 Cookie 中读出正确的 Cookie。该函数的实现如下：

```
function GetCookieVal(offset){
  var endstr=document.cookie.indexOf(";",offset);
  if(endstr==-1)  //没有指定其他元素
    endstr=document.cookie.length;
  return  unescape(document.cookie.substring(offset,endstr));
}
```

3. 删除 Cookie

删除一个 Cookie 很简单，只需将它的过期时间设为 NULL 即可。当用户断开连接后，这个 Cookie 就过期了（没有过期时间的 Cookie 将在浏览器关闭时被删除）。

下面的函数 DeleteCookie 演示了如何删除一个 Cookie。

```
function DeleteCookie(name){
  var exp=new Date();
  exp.setTime(exp.getTime( )-1);  //将 exp 设为已过期的时间
  var cval=GetCookie(name);
  document.cookie=name+"="+cval+";
  expires="+exp.toGMTString( );
}
```

15.2.5　Cookie 的应用实例

Cookie 能够存储与某个独立的 Web 页面相关的特定用户的信息，这使得 Cookie 具有强大的功能。以下举出了 Cookie 的一些用途：

- 指出一个用户已对一个网页进行访问的次数。
- 提醒用户自上次看这个网页后，该网页已发生变化。
- 在联机订购系统中使用 Cookie 来记住一个用户想要买些什么，即可以把 Cookie 看成是一个“购物筐”。用户可能选好了物品后由于某种原因离开了联机订购系统，当他下一次再连接该系统时，他会发现它的“购物筐”依然有上次选好的物品。
- 记住一个用户想要看网页的有页框还是无页框的版本。

下面我们实现前两个功能。

（1）记住一个用户对一个网页进行访问的次数。

这个例子可以指出一个用户访问网页的次数。它通过使用一个 Cookie 来存储连接双方的值来完成。代码如下：

```
<html>
<head>
<script language="JavaScript">
<!--HIDE FROM NONJAVASCRIPT BROWSERS
function GetCookieVal(offset)  //函数 GetCookieVal 的声明
function GetCookie(name)  //函数 GetCookie 的声明
function SetCookie(name,value)  //函数 SetCookie 的声明
function DeleteCookie(name)  //函数 DeleteCookie 的声明
```

```
var expdate=new Date( );
var numvisits;
expdate.setTime(expdate.getTime( )+(5*24*60*60*1000));
if(!(numvisits=GetCookie("numvisits")))
  numvisits=0;
numvisits++;
SetCookie("numvisits",numvisits,expdate);
document.write("Thank you for visiting us.<br>");
document.write("You have loaded this page
<font size=5>"+numvisits+"</font>times<br>");
//end hide->
</script>
</html>
```

提示：注意语句 if(!(numvisits=GetCookie("numvisits"))) numvisits=0**。如果想获取一个** Cookie **的值，而这个** Cookie **又不存在，解释器将返回一个“**numvisits is undefined**”的错误提示。可以使用** not **（**!**）操作符来消除这个问题。**

（2）提醒用户网页已经发生了变化。

这个例子使用一个 Cookie **来确定自从上次访问后一个页面是否已经改变。代码如下：**

```
<html>
<head>
</head>
<script language="JavaScript">
<!--hide from nonjavascript browsers
function GetCookieVal(offset)  //函数 GetCookieVal 的声明
function GetCookie(name)  //函数 GetCookie 的声明
function SetCookie(name,value)  //函数 SetCookie 的声明
function DeleteCookie(name)  //函数 DeleteCookie 的声明
var cookie-data=new Date( );
(document.lastModified);
var expdate=new Date( );
expdate.setTime(expdate.getTime( )+(5*24*60*60*1000));
document.write("This page last updated on:"+document.lastModified);
document.write("<br>");
if(!(cookie-data==GetCookie("cookie-date"))){
  SetCookie("cookie-date",cookie-date,expdate);
  document.write("<font color='red'>this page has changed since your
    last visit!</font><br>");
}
//end hide->
</script>
```

```
</html>
```

程序中检查 Cookie 是否存在的方法与前面一个例子相同。但在这里，将 Cookie 的值设为一个日期而不是一个整数，并把位于 Cookie 中的值 document.lastModified 与前一页面的 doument.lastModified 值相比较。如果它们相同，页面就不发生变化，否则，这个 Cookie 的值将被重新设置，并且浏览器将提醒用户页面已发生变化。

15.3　声 音 处 理

JavaScript 本身没有处理声音的功能。处理声音看上去需要很多函数调用或者复杂的程序。但是在 JavaScript 中载入和回放一段声音是非常容易的事情。下面就向读者介绍如何在 JavaScript 中实现处理声音。

Navigator 具有一个 LiveAudio 插件，使用它可以在浏览器中自由地使用声音。这个插件提供了一些 JavaScript 接口。通过这些接口，就可以在 JavaScript 脚本中完成对声音的一定操作。

LiveAudio 插件提供给 JavaScript 使用的控制声音的方法主要有：

- play(*loop*,*URL*)：回放一段声音。这段声音由 *URL* 参数指出，如果 *loop* 参数是 false，声音只回放一遍就停止了，如果这个参数是 true，声音就反复回放。
- pause( )：暂停一段声音的回放。
- stop( )：完全停止声音的回放。
- stopAll( )：停止所有的声音回放。
- setvol(*vol*)：设定音量，参数 *vol* 表示所要设定的音量占最大音量的百分数。
- fade_to(*vol*)：设定回放的渐弱效果，参数 *vol* 表示效果结束时的音量占最大音量的百分数。
- fade_from_to(*vol1*,*vol2*)：同样设定回放的渐弱效果，参数 *vol1* 和 *vol2* 表示效果开始和结束时的音量。
- start_at_beginning( )：从声音的最开始回放。

不过比较常用的方法是使用<embed>标记（关于该标记的具体用法，请读者参考 8.1 节）。下面是利用<embed>编写的一段随机播放音乐的程序。

```
<html>
<head>
<title>
随机播放音乐
</title>
</head>
<body>
<script language="JavaScript">
var sound1="song1.mid";
```

```
var sound2="song2.mid";
var sound3="song3.mid";
var sound4="song4.mid";
var sound5="song5.mid";
var sound6="song6.mid";
var sound7="song7.mid";
var sound8="song8.mid";
var sound9="song9.mid";
var sound10="song10.mid";
var x=Math.round(Math.random( )*9);
if(x==0) x=sound1;
else if(x==1) x=sound2;
else if(x==2) x=sound3;
else if(x==3) x=sound4;
else if(x==4) x=sound5;
else if(x==5) x=sound6;
else if(x==6) x=sound7;
else if(x==7) x=sound8;
else if(x==8) x=sound9;
else x=sound10;
if(navigator.appName=="Microsoft Internet Explorer")
  document.write('<bgsound src='+'"'+x+'"'+' loop="infinite">');
else
  document.write('<embed src='+'"'+x+'"'+'hidden="true" border="0"
    width="20" height="20" autostart="true" loop="true">');
</script>
</body>
</html>
```

上面的例子实现了随机选择一个音乐文件进行播放，而且是无限循环播放。
下面通过一个实例来介绍如何使用下拉列表框来选择播放的音乐。代码如下：

```
<html>
<head>
<title>
选择播放音乐
</title>
</head>
<body>
<script language="JavaScript">
function PlaySong(SongURL){
PopUp=window.open(SongURL,"Crescendo","toolbar=no,location=no,
  directories=no,status=no,scrollbars=no,resizeable=no,
  copyhistory=no,width=200,height=30")
```

```
}
</script>
<form name="midiform">
<select name="list">
  <option value="song1.mid" [selected]>midi1_name
  <option value="song2.mid">midi2_name
  <option value="song3.mid">midi3_name
</select>
<p><input type=button value="PLAY MIDI" onClick="PlaySong
(midiform.list.options[midiform.list.selectedIndex].value)">
</form>
</body>
</html>
```

程序的执行界面如图 15.2 所示。

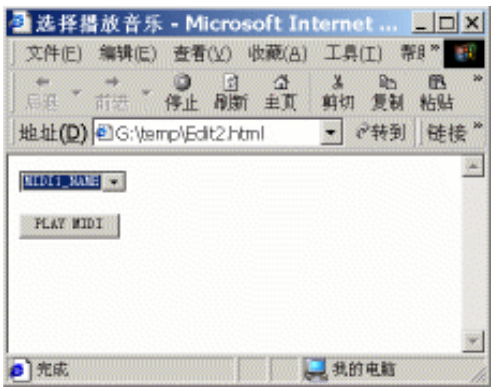

图 15.2　从下拉列表中选择要播放的音乐

在上面的程序中，从下拉列表中选择一个音乐文件，然后单击 PLAY MIDI 按钮就可以播放该音乐了。

15.4　控 制 图 形

一个漂亮的动态网页没有图像是不可想象的，而这离不开对动态图像的操作、设置各种动画效果等等。本节将向读者介绍如何使用 JavaScript 来处理图像。

JavaScript 只是一种脚本语言，它主要用于扩充程序的原有功能，所以不能指望 JavaScript 的功能太强大。但是 JavaScript 语言毕竟能够在 HTML 中操纵图像，利用 JavaScript 脚本，可以创建出非常独特的网页动态效果。

JavaScript 最新实现的一个特性就是动态图形，即可以动态改变网页中显示的图形。这实际上是通过 images 数组来实现操纵图形的。images 数组是浏览器对象 document 的一个属性，在程序中可以通过如下的语法来访问：

```
document.images[index]
```

或者

```
document.imageName
```

数组中的元素都是 Image **对象的实例，每个实例用于存储网页中一幅特定图形的信息。** Image **对象常用的属性有：**

- border：**同**<img>**标记中的** border **属性相一致，指示图形是否有边框。**
- complete：**指示浏览器是否已经完成这幅图形的装载，是一个逻辑标志。**
- height：**当前图形的高度，以像素为单位，只读，程序中不能修改。**
- width：**当前图形的宽度，以像素为单位，只读，程序中不能修改。**
- hspace：**当前图形到页面的水平距离，只读，程序中不能修改。**
- vspace：**当前图形到页面的垂直距离，只读，程序中不能修改。**
- name：**当前图形的名称，在**<img>**标记的** name **属性中设置。**
- src：**当前图形的** URL**。**
- onLoad：**当浏览器载入一个图形时调用。**
- onAbort：**当用户退出网页时调用。**
- onError：**当浏览器载入一个图形发生错误时调用。**

下面的程序演示了<img>**标记的用法，它实现的是禁止在页面上使用右键下载图片的功能。程序代码如下：**

```
<html>
<head>
<title>
禁止下载图片
</title>
</head>
<body>
<a
href="javascript:void(0)" onMouseOver="alert('对不起,图片不能随便下载!')
  "><img src="picture.gif" align="center" border="0" width="300"
  height="250">
</a>
</body>
</html>
```

该程序的执行效果如图 15.3 **所示。**

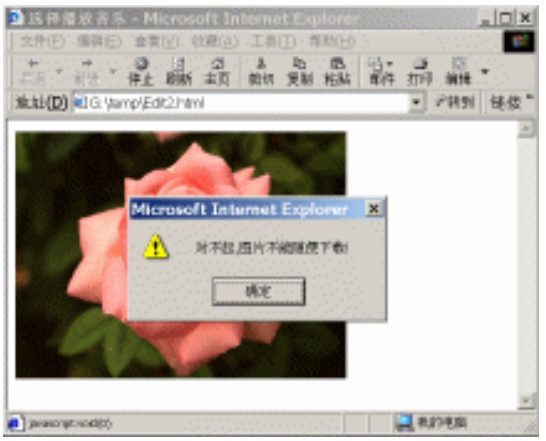

图 15.3　禁止下载图片

在 JavaScript 中使用图形，只要创建一个 image 对象，指定对象的 src 属性，浏览器就可以自动地从服务器上下载图形到本地计算机中。例如：

```
Image1=new Image( );
Image1.src="picture.gif";
```

下面介绍一个使用 Image 对象的实例，代码如下：

```
<html>
<head>
<title>
图片响应鼠标变换
</title>
</head>
<body bgcolor="#FEF4D9" onLoad="swapPic( )">
<script language="JavaScript">
var rand1=0;
var useRand= 0;
images=new Array;
images[1]=new Image( );
images[1].src="hh002.jpg";
images[2]=new Image( );
images[2].src="hh003.jpg";
images[3]=new Image( );
images[3].src="hh004.jpg";
images[4]=new Image( );
images[4].src="hh005.jpg";
function swapPic( ){
  var imgnum=images.length-1;
  do{
```

```
    var randnum=Math.random( );
    rand1=Math.round((imgnum-1)*randnum)+1;
  }while(rand1==useRand);
  useRand=rand1;
  document.randimg.src=images[useRand].src;
}
</script>
<a
onClick="swapPic( );"><img name="randimg" src="hh002.jpg">
</a>
<br>
<font face="verdana" size="8">
请点击图片
</font>
</body>
</html>
```

该程序的执行结果如图 15.4 所示。

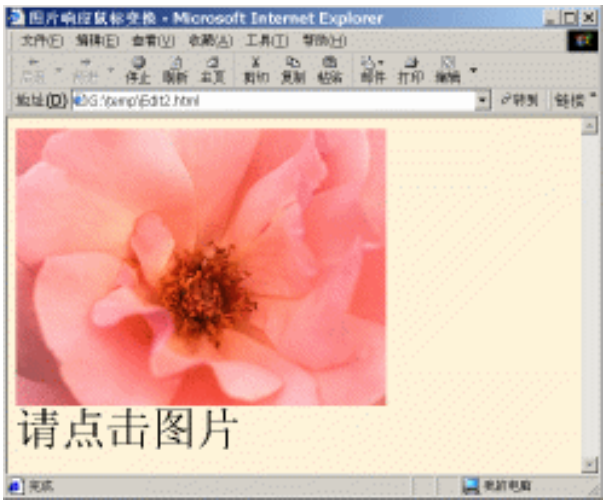

图 15.4 图片响应鼠标而变化

在图片上单击鼠标左键，就可以实现在 4 个图片之间随机变换了。

15.5 ActiveX 与 JavaScript

ActiveX 对象是 Microsoft 在 OLE（对象链接和嵌入）技术的基础上，结合 Internet 发展而成的，它是一种可以从 Web 服务器下载的特殊类型。当浏览器发现 ActiveX 对象时，它自动从硬盘上装入对应的 ActiveX 文档服务程序，并且服务程序接管了整个浏览器窗口，用以显示文档的内容，用户可以查看文档的内容，也可以编辑文档，但不能上载回服务器。

ActiveX 不像 CGI，在 CGI 中人们可以用任何语言编写一个脚本，只要保证脚本的输出能由 Web 服务器处理即可。它也不象 Java，人们用 Java 编写应用程序，然后由 Java 虚拟机来编译信息。ActiveX 是 Microsoft 为了使开发人员能够把计算机桌面环境与构成 Internet 及其大量资源的环境集成起来，同时保护在 Windows 中现有的开发投资。ActiveX 包括对两个现有 Microsoft 技术的一系列扩充和增强，这两个技术是：WIN32 API 和组件对象模型（COM）。OLE 应用是基于它构成的。

ActiveX 是工具的集合，它允许人们在其他应用，如字处理器、电子表格或 Java Applet 中加入组件并与它们进行交互，以使自己的 Web 页面更动态化。

ActiveX 是 Microsoft 的 Internet 整体战略的一个部分，该战略意在帮助 Web 开发者开发 Internet 工具，为 Web 站点开发动态的、丰富的内容。ActiveX 基于 OLE 2.0（两者都是按照 COM 标准推出的），它允许在 Web 文档中集成控件，从而能建立适于 Web 的较小的应用。

所谓 ActiveX 组件是指一些可执行的代码，比如一个 EXE、DLL 或 OCX 文件，它们在提供对象时遵循 ActiveX 的规范。通过 ActiveX 技术，程序员就能够将这些可复用的软件组装到应用程序或者服务程序中去了。

ActiveX 包括 3 类主要组件，用户可以利用它们使自己的 Web 页面更有趣味性。这 3 个组件是 ActiveX 控件、ActiveX 文档和 ActiveX 脚本。

下面将分别对每个组件作详细介绍，并说明如何使用它们。

15.5.1　ActiveX 控件

ActiveX 控件是用户可以通过<object>标记很容易加入自己的 HTML 文档中的活动对象。任何熟悉编程的人，特别是那些熟悉 OLE 2.0 规范的人都可以开发 ActiveX 控件。另外，IE 也附带有几个控件，用户无需编程即可立刻使用。下面即是这些控件中的一部分：

- Marquee：允许用户建立空页面，其中可装载 URL。该页面既可横向滚动，也可纵向滚动。
- Chart：允许用户建立不同风格的图表。例如，利用销量数据，可以建立条形图、拼图甚至股价图。
- Menu：允许用户建立下拉式菜单，只需单击一下即可激活。Menu 控件很像大部分 Windows 应用程序中的屏幕顶部菜单。
- Pop-Up Menu：允许用户建立弹出式菜单，不同菜单项均可单击。
- Preloader（预装载器）：装载一个指定的 URL 并将该 URL 放在缓存中，可供访问站点的人快速访问。

如前所述，通过使用<object>标记，用户可以在自己的 HTML 文档中使用各种 ActiveX 控件（包括 IE 附带的控件）。每个控件都有一些参数允许用户管理它们的使用或显示方式。

例如，在页面中插入一个 Calendar 控件的代码如下：

```
<html>
<head>
<title>Calendar 控件</title>
```

```
</head>
<body>
<object classid="clsid:8e27c92b-1264-101c-8a2f-040224009c02"
  id="calendar1" width="288" height="192">
  <param name="_version" value="524288">
  <param name="_extentx" value="7620">
  <param name="_extenty" value="5080">
  <param name="_stockprops" value="1">
  <param name="backcolor" value="-2147483633">
  <param name="year" value="2001">
  <param name="month" value="4">
  <param name="day" value="7">
  <param name="daylength" value="0">
  <param name="monthlength" value="0">
  <param name="dayfontcolor" value="0">
  <param name="firstday" value="1">
  <param name="gridcelleffect" value="1">
  <param name="gridfontcolor" value="10485760">
  <param name="gridlinescolor" value="-2147483632">
  <param name="showdateselectors" value="-1">
  <param name="showdays" value="-1">
  <param name="showhorizontalgrid" value="-1">
  <param name="showtitle" value="-1">
  <param name="showverticalgrid" value="-1">
  <param name="titlefontcolor" value="10485760">
  <param name="valueisnull" value="0">
</object>
</body>
</html>
```

该程序的执行结果如图 15.5 所示。

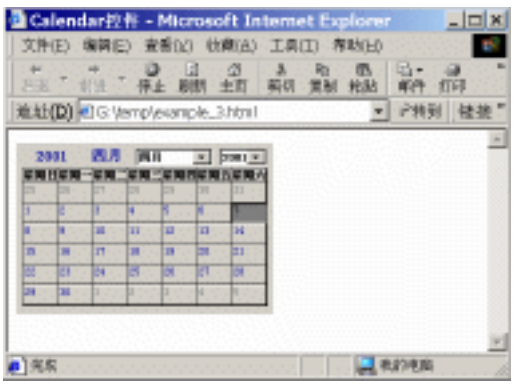

图 15.5 在页面中插入 Calendar 控件

<object>标记包含许多属性，可以用来从 HTML 文档中访问 ActiveX 控件。下面简单列出了<object>标记的属性以及它们的使用方法。

- align：对象的对齐方式。它可取值 baseline、cente、left、middle、right、textbottom、textmiddle 或 texttop。
- border：对象的边界宽度，以像素为单位。
- classid：对象的标识。
- codebase：对象的代码基。
- codetype：Internet 媒体类型。
- data：控件所用数据的位置。
- declare：声明一个对象。
- height：对象的高度。
- hspace：水平边距。
- name：对象名称。
- shapes：对象有定型的超级链接。
- standby：装载对象时需要显示的信息。
- type：数据的 Internet 媒体类型。
- usemap：与对象一起使用的图像地图。
- vspace：垂直边距。
- width：对象的宽度。

在<object>标记中还可包含其他标记，最重要的一种是<param>标记，它允许用户指定传给 ActiveX 组件的参数。参数可以是从组件的高度和宽度到组件显示给访问者的任何信息。以下是<param>标记可以使用的属性：

- name：特性的名字。
- value：在 name 中标识的特性的值。
- valuetype：可以是 ref object 或 data。
- type：当 valuetype=ref 时，在 value 属性中引用的项的 Internet 媒体类型。

15.5.2　ActiveX 文档

ActiveX 文档是指可以不经任何修改而在 Web 浏览器中浏览其内容的文档。在 ActiveX 技术出现之前，在 Web 中显示一个 Word 文档的唯一方式是利用宏功能存储 Word 文档，然后去掉特定 Word 文档的类型定义并插入适当的 HTML 代码。

支持 ActiveX 的浏览器可以不对文档作任何修改就能装载支持 ActiveX 的文档。Word 和 Excel 文档是在支持 ActiveX 的 Web 浏览器中可以浏览的两种文档。浏览器装载文档的同时也装载了 Word 本身。Word 和 Web 浏览器两者的所有菜单项和图标均被显示。

无论何时，当 Word 或其他 ActiveX 文档被装载进支持 ActiveX 的浏览器时，属于该 ActiveX 文档的所有功能对浏览该文档的人来说都可用，包括编辑该文档。但是用户所作的修改不能保存在服务器上，只能另存于本地硬盘后才可以使用。

15.5.3 ActiveX 脚本

使用 ActiveX 控件的脚本最初是用 VBScript 建立的。现在几乎可以用任何一种语言访问 ActiveX 控件，包括 Perl 和 C++。编写 ActiveX 脚本的目的就是允许用户控制 ActiveX 控件的行为。例如，如果用户使用了 ActiveX Menu 控件，通过编写脚本，用户就可以控制当选择由 ActiveX 菜单控件创建的菜单的某项时所发生的事件。控件本身仅仅简单地显示菜单列表，用户编写的脚本必须让该菜单完成某些工作。

VBScript 并不是唯一能建立使用 ActiveX 控件的活动文档的脚本语言。JavaScript 的设计使得它也能够使用 ActiveX 控件。

利用 JavaScript 和 ActiveX 控件可以建立动态 Web 页面。下面是利用 JavaScript 控制 Adobe Acrobat 的 ActiveX 插件的程序，其代码如下：

```
<html>
<head>
</head>
<body>
<p align="center">
<object classid="clsid:CA8A9780-280D-11CF-A24D-444553540000"
  id="Pdf1" width="800" height="500">
  <param name="_Version" value="65539">
  <param name="_ExtentX" value="21167">
  <param name="_ExtentY" value="13229">
  <param name="_StockProps" value="0">
  <param name="SRC" value>
</object>
<form name="form1">
<select name="list">
  <option value="01.pdf" [selected]>01
  <option value="02.pdf">02
  <option value="03.pdf">03
  <option value="04.pdf">04
</select>
<input type="button" value="选择文件" onClick="Change(
  form1.list.options[form1.list.selectedIndex].value)">
</form>
<script language="JavaScript">
function Change(pdffile){
  Pdf1.SRC=pdffile;
}
</script>
</body>
</html>
```

该程序的结果如图 15.6 所示。

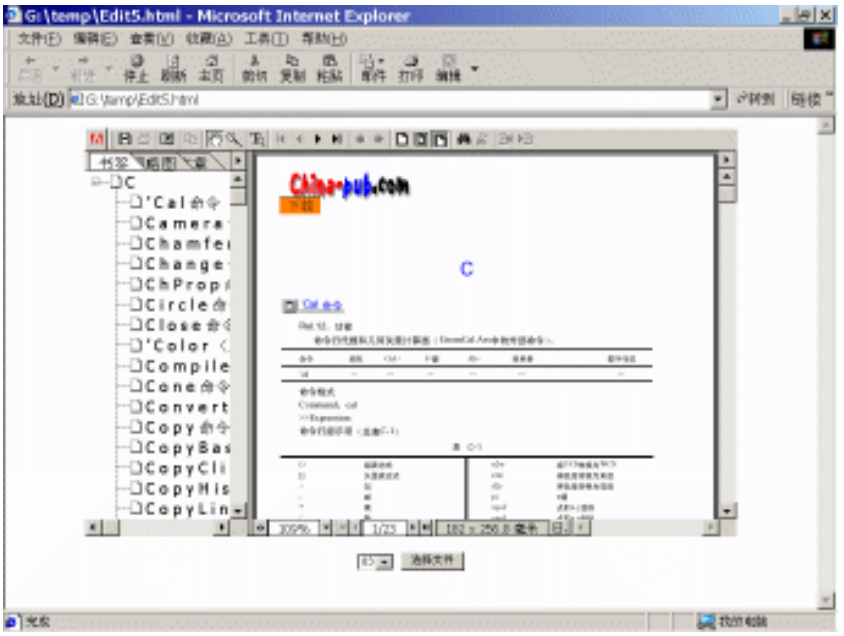

图 15.6　用 JavaScript 控制插件

从下拉列表中选择需要显示的 PDF 文件，然后单击“选择文件”按钮，就可以在 IE 中打开 PDF 格式的文件。

15.6　小 结

本章介绍的是 JavaScript 的高级内容，包括 frame，Cookie，ActiveX 和控制图像、处理声音等技术，它们是在读者掌握了 JavaScript 的基本内容之后的升华。读者在学习完本章之后对 JavaScript 的技术就已经非常了解了。当然，要使用这些高级内容，需要熟练掌握 JavaScript 的基本语法和内容以后才能够做到。

下一章将介绍两个综合使用 JavaScript 的程序范例。

第 16 章　JavaScript 创作实例

在学习完 JavaScript 的大部分功能以后，读者可能想亲自动手编写一些 JavaScript 程序了。为此本章将给出两个 JavaScript 脚本程序作为范例，并加以详细解释。这两个范例程序基本涵盖了 JavaScript 应用的各个方面。

16.1　计算 24 点程序

相信几乎所有的人都玩过用 4 张纸牌计算 24 点的游戏。游戏的规则是：对由 4 张纸牌表示的数字进行加、减、乘、除 4 种运算，得到一个值为 24 的表达式。这个游戏可以让人开动脑筋，但是有时候冥思苦想却也算不出来，很令人苦恼，甚至还会碰到无解的情况。用 JavaScript 就可以很方便地解决这一难题。

首先设计算法。假定 4 张牌的数值分别为 a，b，c，d。将 a，b，c，d 组成一个数组，并将+、−、*、/这 4 个运算符号组成符号数组。利用循环语句，可以将这 4 个数和 4 个符号组成多个表达式序列。另外，还必须考虑括号。根据排列组合的思想，括号的使用只有以下 6 种可能：

```
a+b+c+d
(a+b+c)*d
a*(b+c)+d
a+b/(c+d)
a*(b+c+d)
(a+b)*(c-d)
```

下面就是这个例子的源文件。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=GB_2312-80">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<title>
教你算 24 点
</title>
</head>
<body>
<p align=center>
<font size=9>
<marquee height="30" width="50%">教你算 24 点</marquee>
```

```
</font>
</p>
<form name="form1">
<p align=center>
<input type="text" size=12 name="card1">
<input type="text" size=12 name="card2">
<input type="text" size=12 name="card3">
<input type="text" size=12 name="card4">
</p>
<p align=center>
<input type="button" name="ok" value="点击运算"
  onClick="JavaScript:go( )">
</p>
<p align=center>
<input type="text" size=20 name="result">
</p>
<p align=center>
<input type="text" size=20 name="res"
  value="这栏只有得到 24 时才有">
</p>
</form>
<script language="JavaScript">
<!--
//init 函数用来生成+、-、*、/这 4 个符号数组
function init( ){
  this.length=init.arguments.length;
  for(var n=0;n<this.length;n++)
    this[n]=init.arguments[n];
}
//init2 函数用来生成给定长度的数组对象
function init2( ){
  this.length=init2.arguments.length;
  for(var n=0;n<this.length;n++)
    this[n]='';
}
//当单击“运算”按钮时，执行 go 函数
function go(  ){
  var result=false;  //变量 result 代表是否能得到结果
  var synal=new init('+','-','*','/');  //定义符号数组
  var card=new init2(5);  //定义长度为 5 的数值数组
  var str=new init2(7);     //定义长度为 7 的字符串数组
  card[1]=form1.card1.value;
  card[2]=form1.card2.value;
  card[3]=form1.card3.value;
  card[4]=form1.card4.value;  //获取表单输入
  //测试输入是否有效
```

```
for(var i=1;i<=4;i++){
  if(isNaN(parseInt(card[i]))){
    alert("输入错误，请重试");
    return false;
  }
}
first:for(var i=1;i<=4;i++){
  for(var j=1;j<=4;j++){
    if(j==i)
      continue;
    for(var k=1;k<=4;k++){
      if((k==i)||(k==j))
        continue;
      for(var l=1;l<=4;l++){
        if((l==i)||(l==j)||(l==k))
          continue;
        for(var ii=0;ii<=3;ii++){  //对 4 张牌排序
          for(var jj=0;jj<=3;jj++){
            for(var kk=0;kk<=3;kk++){   //对 4 个符号排序
              //可能的 6 种结果
              str[1]=""+card[i]+synal[ii]+card[j]+synal[jj]+
                card[k]+synal[kk]+card[l];
              str[2]="("+card[i]+synal[ii]+card[j]+synal[jj]+
                card[k]+")"+synal[kk]+card[l];
              str[3]=""+card[i]+synal[ii]+"("+card[j]+synal[jj]+
                card[k]+")"+synal[kk]+card[l];
              str[4]=""+card[i]+synal[ii]+card[j]+synal[jj]+
                "("+card[k]+synal[kk]+card[l]+")";
              str[5]=""+card[i]+synal[ii]+"("+card[j]+synal[jj]+
                card[k]+synal[kk]+card[l]+")";
              str[6]="("+card[i]+synal[ii]+card[j]+")"+synal[jj]+
                "("+card[k]+synal[kk]+card[l]+")";
              //对 6 种结果分别计算，如果得到 24，则退出
              for(var m=1;m<=6;m++){
                if(eval(str[m])==24){
                  document.form1.result.value="算出来了，呵呵";
                  document.form1.res.value=""+str[m];
                  result=true;
                  break first;
                }
              }
            }
          }
        }
      }
    }
```

```
      }
    }
  }
if(result==false){  //如果得不到 24
  document.form1.result.value="我算不出来!";
  document.form1.res.value="这栏只有得到 24 时才有";
}
}
//-->
</script>
<script language="JavaScript">
<!--
//状态栏跑马灯
var msg="欢迎使用 Rake 的烂程序！";
var interval=100;
var spacelen=120;
var space10=" ";
var seq=0;
function Scroll( ){
  len=msg.length;
  window.status=msg.substring(0,seq+1);
  seq++;
  if(seq>=len){
    seq=spacelen;
    window.setTimeout("Scroll2( );",interval);
  }
  else
    window.setTimeout("Scroll( );",interval);
}
function Scroll2( ){
  var out="";
  for(i=1;i<=spacelen/space10.length;i++)
    out+=space10;
    out=out+msg;
    len=out.length;
    window.status=out.substring(seq,len);
    seq++;
    if(seq>=len) seq=0;
    window.setTimeout("Scroll2( );",interval);
}
Scroll( );
//-->
</script>
</body>
</html>
```

这个例子的执行结果如图 16.1 所示。

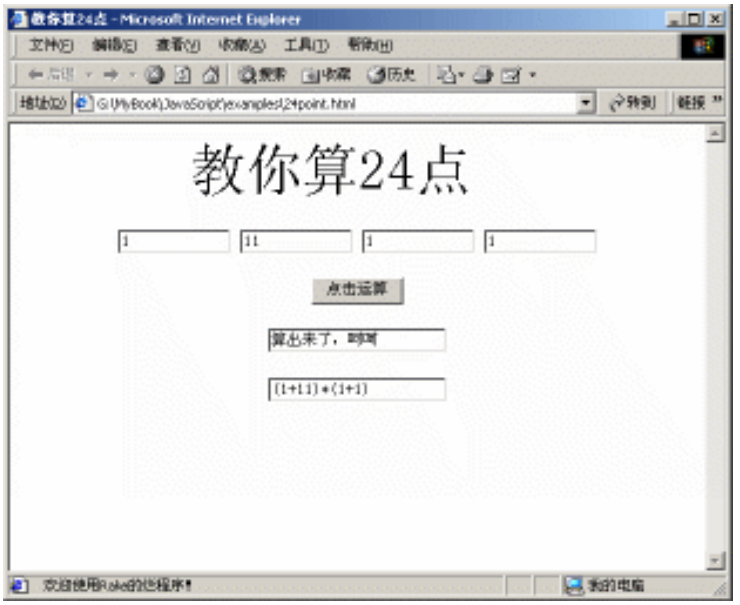

图 16.1 显示计算结果

在程序中还对表单输入的数据进行了验证。如果输入的不是数字，而是字母，那么程序会提示输入有误，如图 16.2 所示。

图 16.2 提示输入错误

16.2　漂亮的荧光字

街上的荧光灯大家见过不少了，有没有想过用 JavaScript 也可以制作出漂亮的荧光字呢？事实上，计算机上的文字显示都是通过点阵实现的。我们可以对文字自行编码，然后将点阵中的每一个点都用特定的图像来表示，这样就可以得到漂亮的荧光字的效果了。

限于篇幅，这里只考虑对英文字母和阿拉伯数字的编码。由于它们的结构简单，我们可以统一采用 5×3 的编码格式。即任何一个字母和阿拉伯数字我们都可以用一个 5 行 3 列的点阵来表示。例如，对字母 H 的编码为：

```
*     *
*     *
*   *  *
*     *
*     *
```

首先准备两幅用来表示点的图片。图片不能太大，5×5 像素就可以了。一幅图片的内容是一个带颜色的点，用来代表上面编码中的*号，文件名为 on.gif。另一幅图片为空，用来表示相应的位置没有编码，文件名为 off.gif。在程序中，我们建立一个 5×3 的图片序列，然后只要设置相应的图片为 on.gif 还是 off.gif 就可以显示出文字了。

下面是这个例子的源代码。

```
<html>
<head>
<title>
荧光字
</title>
</head>
<body onLoad="JavaScript:start( )">
<script language="JavaScript">
var sign;  //定义要显示的字符
var height=5;  //显示字符的高度，5 行
var letters=new letterArray( );  //字符编码
var on=new Image(10,10);  //对应的两幅图片
var off=new Image(10,10);
//设置图片的 URL
on.src="on.gif";
off.src="off.gif";
//用户输入需要显示的字符
function start( ){
  sign=window.prompt("请输入需要显示的字符");
  drawBlank( );  //绘制背景
  drawletter(sign);  //绘制字符
```

```
}
function drawletter(temp){  //绘制字符函数
  for(var x=0;x<3;++x){
    for(var y=0;y<5;++y){
      setLight(letters[temp][y].charAt(x)=="*",x,y);//设置相应位置亮
    }
  }
}
function setLight(state,x,y){  //设置指定位置亮还是暗
  if(state)
    document.images[3*y+x].src=on.src
  else
    document.images[3*y+x].src=off.src
}
function drawBlank( ){  //绘制背景
  var gt=unescape("%3e")  //gt 代表大于号
  document.write('<table border=2 cellpadding=0'+gt+'<tr'+gt+'<td
    bgcolor align="center" valign="center"'+gt)
  for(var y=0;y<height;++y){  //显示背景图片
    for(var x=0;x<3;++x)
      document.write('<img src="'+off.src+'" height=10 width=10'+gt)
    document.write('<br'+gt)
  }
  document.write('</td'+gt+'</tr'+gt+'</table'+gt)
}
function letterArray( ){
  this.a=new Array(height)
  this.a[0]=" * "
  this.a[1]="* *"
  this.a[2]="***"
  this.a[3]="* *"
  this.a[4]="* *"
  this.b=new Array(height)
  this.b[0]="** "
  this.b[1]="* *"
  this.b[2]="** "
  this.b[3]="* *"
  this.b[4]="** "
  this.c=new Array(height)
  this.c[0]="***"
  this.c[1]="*  "
  this.c[2]="*  "
  this.c[3]="*  "
  this.c[4]="***"
  this.d=new Array(height)
```

```
this.d[0]="** "
this.d[1]="* *"
this.d[2]="* *"
this.d[3]="* *"
this.d[4]="** "
this.e=new Array(height)
this.e[0]="*** "
this.e[1]="*  "
this.e[2]="***"
this.e[3]="*  "
this.e[4]="***"
this.f=new Array(height)
this.f[0]="***"
this.f[1]="*  "
this.f[2]="***"
this.f[3]="*  "
this.f[4]="*  "
this.g=new Array(height)
this.g[0]="***"
this.g[1]="*  "
this.g[2]="***"
this.g[3]="* *"
this.g[4]="***"
this.h=new Array(height)
this.h[0]="* *"
this.h[1]="* *"
this.h[2]="***"
this.h[3]="* *"
this.h[4]="* *"
this.i=new Array(height)
this.i[0]="***"
this.i[1]=" * "
this.i[2]=" * "
this.i[3]=" * "
this.i[4]="***"
this.j=new Array(height)
this.j[0]="  *"
this.j[1]="  *"
this.j[2]="  *"
this.j[3]="* *"
this.j[4]="***"
this.k=new Array(height)
this.k[0]="* *"
this.k[1]="* *"
this.k[2]="** "
```

```
this.k[3]="* *"
this.k[4]="* *"
this.l=new Array(height)
this.l[0]="*  "
this.l[1]="*  "
this.l[2]="*  "
this.l[3]="*  "
this.l[4]="***"
this.m=new Array(height)
this.m[0]="* *"
this.m[1]="***"
this.m[2]="***"
this.m[3]="* *"
this.m[4]="* *"
this.n=new Array(height)
this.n[0]="* *"
this.n[1]="***"
this.n[2]="***"
this.n[3]="***"
this.n[4]="* *"
this.o=new Array(height)
this.o[0]="***"
this.o[1]="* *"
this.o[2]="* *"
this.o[3]="* *"
this.o[4]="***"
this.p=new Array(height)
this.p[0]="** "
this.p[1]="* *"
this.p[2]="** "
this.p[3]="*  "
this.p[4]="*  "
this.q=new Array(height)
this.q[0]="***"
this.q[1]="* *"
this.q[2]="* *"
this.q[3]="***"
this.q[4]="***"
this.r=new Array(height)
this.r[0]="** "
this.r[1]="* *"
this.r[2]="** "
this.r[3]="* *"
this.r[4]="* *"
this.s=new Array(height)
```

```
this.s[0]="***"
this.s[1]="*  "
this.s[2]="***"
this.s[3]="  *"
this.s[4]="***"
this.t=new Array(height)
this.t[0]="***"
this.t[1]=" * "
this.t[2]=" * "
this.t[3]=" * "
this.t[4]=" * "
this.u=new Array(height)
this.u[0]="* *"
this.u[1]="* *"
this.u[2]="* *"
this.u[3]="* *"
this.u[4]="***"
this.v=new Array(height)
this.v[0]="* *"
this.v[1]="* *"
this.v[2]="* *"
this.v[3]="* *"
this.v[4]=" * "
this.w=new Array(height)
this.w[0]="* *"
this.w[1]="* *"
this.w[2]="***"
this.w[3]="***"
this.w[4]="***"
this.x=new Array(height)
this.x[0]="* *"
this.x[1]="* *"
this.x[2]=" * "
this.x[3]="* *"
this.x[4]="* *"
this.y=new Array(height)
this.y[0]="* *"
this.y[1]="* *"
this.y[2]="***"
this.y[3]=" * "
this.y[4]=" * "
this.z=new Array(height)
this.z[0]="***"
this.z[1]="  *"
this.z[2]=" * "
```

```
this.z[3]="*  "
this.z[4]="***"
this['0']=new Array(height)
this['0'][0]=" * "
this['0'][1]="* *"
this['0'][2]="* *"
this['0'][3]="* *"
this['0'][4]=" * "
this['1']=new Array(height)
this['1'][0]=" * "
this['1'][1]=" * "
this['1'][2]=" * "
this['1'][3]=" * "
this['1'][4]=" * "
this['2']=new Array(height)
this['2'][0]="***"
this['2'][1]="  *"
this['2'][2]="***"
this['2'][3]="*  "
this['2'][4]="***"
this['3']=new Array(height)
this['3'][0]="***"
this['3'][1]="  *"
this['3'][2]="***"
this['3'][3]="  *"
this['3'][4]="***"
this['4']=new Array(height)
this['4'][0]="* *"
this['4'][1]="* *"
this['4'][2]="***"
this['4'][3]="  *"
this['4'][4]="  *"
this['5']=new Array(height)
this['5'][0]="***"
this['5'][1]="*  "
this['5'][2]="***"
this['5'][3]="  *"
this['5'][4]="** "
this['6']=new Array(height)
this['6'][0]="** "
this['6'][1]="*  "
this['6'][2]="***"
this['6'][3]="* *"
this['6'][4]="***"
this['7']=new Array(height)
```

```
  this['7'][0]="***"
  this['7'][1]="  *"
  this['7'][2]=" * "
  this['7'][3]="*  "
  this['7'][4]="*  "
  this['8'] = new Array(height)
  this['8'][0]="***"
  this['8'][1]="* *"
  this['8'][2]="***"
  this['8'][3]="* *"
  this['8'][4]="***"
  this['9']=new Array(height)
  this['9'][0]="***"
  this['9'][1]="* *"
  this['9'][2]="***"
  this['9'][3]="  *"
  this['9'][4]="***"
}
</script>
</body>
</html>
```

程序开始执行时提示用户输入需要显示的字符（只能是字母或数字），如图 16.3 **所示。**

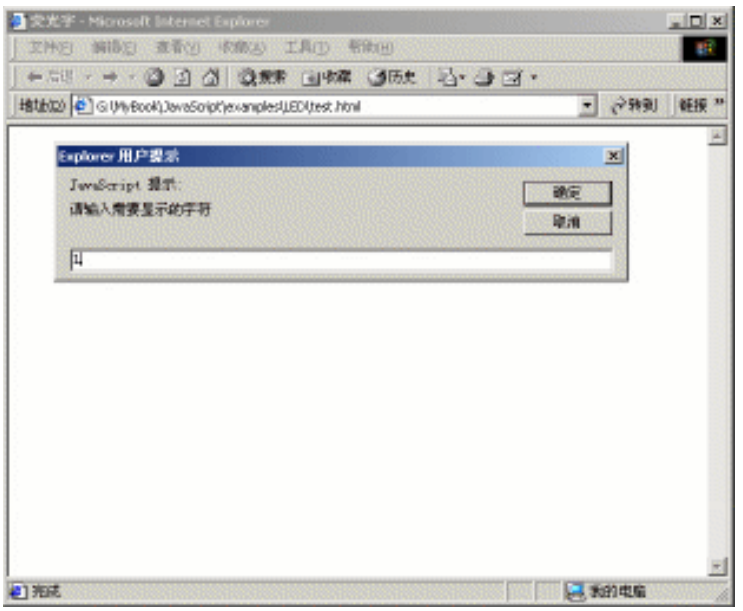

图 16.3　**提示输入需要显示的字符**

单击“确定”按钮，显示结果如图 16.4 **所示。**

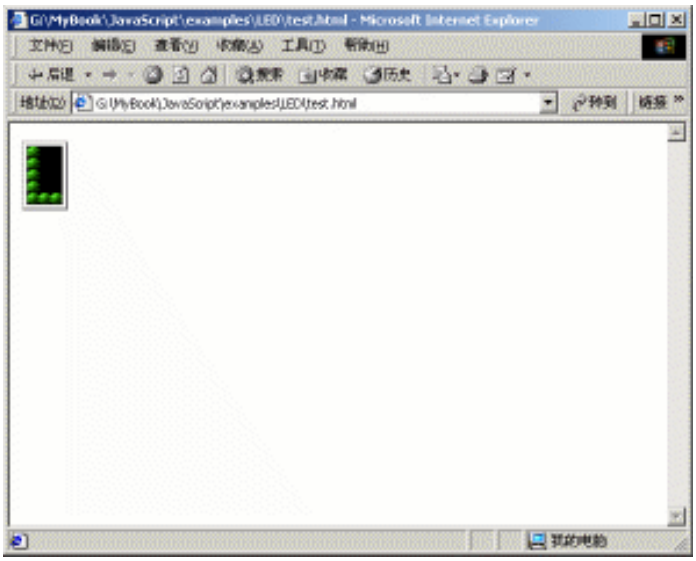

图 16.4 字母 L 的显示结果

16.3 小 结

本章介绍了两个 JavaScript 创作实例，读者可以从中体会 JavaScript 在网页设计中的应用范围，并了解使用 JavaScript 究竟可以完成什么样的功能。

第 3 部分　ASP

第 17 章　ASP 概述

ASP 使生成 Web 动态内容及构造功能强大的 Web 应用程序的工作变得十分简单。本章将主要介绍 ASP 的含义及其基本功能、ASP 的使用时机、ASP 运行环境配置以及 ASP 的开发工具等，使读者对 ASP 有一个基本了解。

17.1　ASP 简介

所谓 ASP（Active Server Pages）是指一种服务器端的脚本环境，而 ASP 页就是包含 HTML 标记、文本以及脚本命令的文件。ASP 页可以调用 COM（Component Object Model，组件对象模型）组件执行任务，如连接到数据库或执行商业计算等。利用 ASP，用户可以创建动态 Web 页，向 Web 页添加交互式内容，或者编译完整的 Web 应用程序，这些应用程序使用 Web 页作为客户界面。

17.1.1　ASP 的含义

ASP（Active Server Pages）从字面上理解是动态网页服务，它包含以下 3 方面的含义：

（1）ASP 利用 ActiveX 组件来实现一定的功能。

所谓 ActiveX 组件是指存在于 Web 服务器端的动态链接库（.dll）或可执行文件（.exe）中的、可以用来实现某些特定功能的模块。组件提供了一个或多个对象，这些对象都有自己的方法和属性。在 Web 服务器上可以利用组件来快速、方便地建立自己的 Web 应用。在 ASP 页中可以使用 ASP 的内置组件和第三方组件，也可以根据自己的需要来创建自己的组件。

（2）ASP 是服务器端的编程技术，运行在服务器端。

这是 ASP 的一个优点。ASP 对客户端的环境没有要求，在服务器端，无需考虑客户端浏览器是否支持 ASP 所使用的编程语言，这样既可以达到“瘦客户端”的目的，又便于自己控制。

下面的例子演示了运行在服务器端的 ASP 和运行在客户端的 JavaScript 的不同之处。

```
<html>
<title>测试</title>
```

```
<script language="JavaScript">
<!--
var today=new Date( );
var day=new Array("星期日","星期一","星期二","星期三","星期四",
  "星期五","星期六");
document.write("<font color=##999999 style='font-size:20pt;
  font-family:宋体'>"客户端的时间是：",
  today.getYear( ),"年",
  today.getMonth( )+1,"月",
  today.getDate( ),"日  ",
  today.getHours( ),"小时  ",
  today.getMinutes( ),"分钟  ",
  today.getSeconds( ),"秒  ",
  day[today.getDay( )+1],
  "</font>");
-->
</script>
<br>
<%
Response.Write "<font color="RED" style='font-size:20pt;font-family:
  宋体'>"&"服务器端的时间是："&date( )&chr(32)&time( )&"</font>"
%>
</html>
```

把这段代码放在 Web 服务器上运行，可以看到如图 17.1 所示的结果。

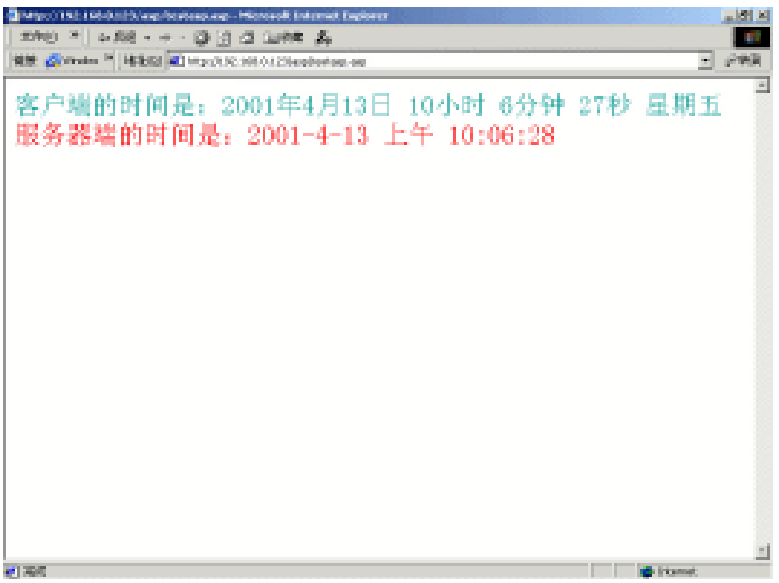

图 17.1 程序运行结果

注意：上面显示的时间分别是在服务器和客户端即时取得的时间。本书在讲述 ASP 的有关内容时所涉及的客户端指浏览器端，服务器端指运行 ASP 程序的 Web 服务器。有关客户端和服务器端的详细概念请参考其他相关书籍。

上面例子中，从<script>到</script>的代码是 JavaScript 脚本语言，其功能是取得客户端的时间（JavaScript 运行在客户端）；从<%到%>是 ASP 程序，功能也是取得系统时间。由于 ASP 运行在服务器端，因而取得的时间为服务器端时间。现在将客户端时间改变为 2000 年，再运行程序，则浏览结果如图 17.2 所示。

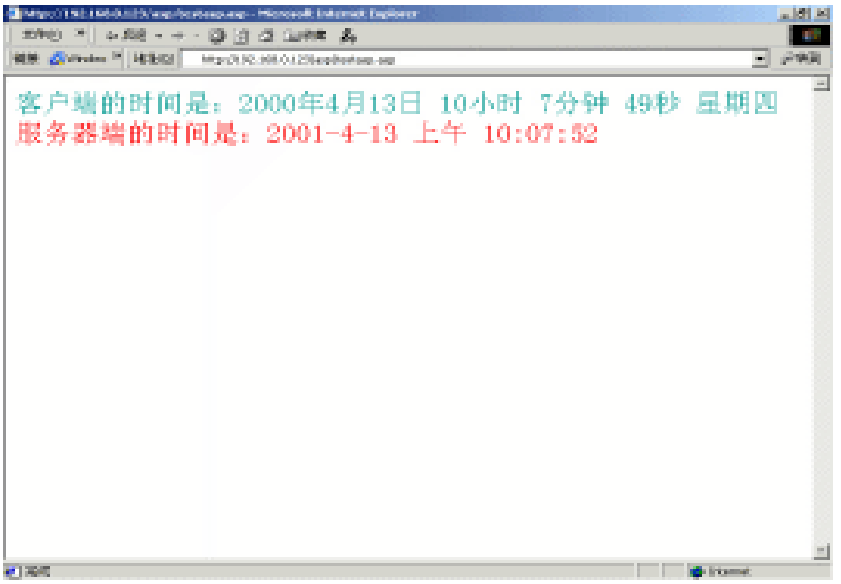

图 17.2　改变客户端时间后的结果

可以看到，此时客户端的时间变成了 2000 年，而用 ASP 显示的时间依然是 2001 年。这样，不管客户端如何调整时间，都不会影响服务器端的程序的运行结果。

（3）ASP 返回标准的 HTML 页面，可以在常用的浏览器中显示。

ASP 通过 IIS（Internet Information Server，Internet 信息服务）中的一个 asp.dll 文件解析成 HTML 代码。浏览者查看页面源文件时，看到的是 ASP 生成的 HTML 代码，而不是 ASP 程序代码，这样就可以防止别人看到源代码。ASP 运行的具体流程如图 17.3 所示。

注意：图 17. 3 只是一个简要的流程图，实际上 IIS 对一个 HTTP 请求的回应所做的动作要复杂得多。

由此可以看出，ASP 是在 IIS 下开发 Web 应用的一种简单、方便、安全的编程工具。

17.1.2　ASP 的基本功能和使用时机

ASP 的基本功能有：

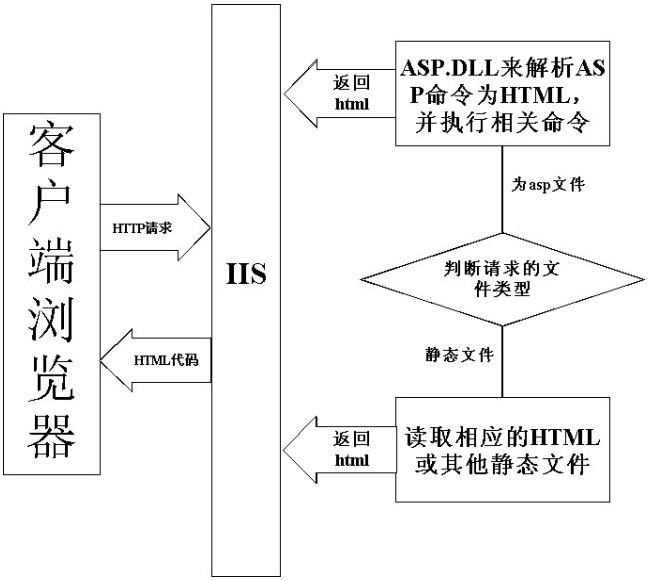

图 17.3 ASP 运行流程图

- 能够提供动态网页服务，这是 ASP 最基本的功能。ASP 可以动态地更新网站，使网站看起来更加生动活泼，使用户得到动态服务，使网站的交互性更强。由于 ASP 返回动态生成的静态（HTML）页面，所以，可以为用户定制个性化的服务，使不同的用户得到不同的服务。
- 可以利用各种组件实现各种强大的功能，并方便地创建 Web 应用程序。例如，借助 ASP 内置的 Ad Rotator 组件可以轻松实现广告发布管理。借助 AspMail（第三方的邮件组件）组件可以很方便地发送 Email。
- 利用 FileSystemObject（文件系统对象）可以方便地管理主机上的文件，进行文件的浏览、建立、删除、复制等操作。
- 具有强大的访问操作数据库的功能。利用 ADO 的支持，ASP 对数据库的操作非常得心应手。甚至可以像使用本地数据库那样，管理远程主机上的数据库，对表格、记录进行各种操作。
- 通过 ADSI（Active Directory Server Interface，动态目录服务接口）或者 WSH（Windows Scripting Host，Windows 脚本语言）可以对 Windows NT 实现远程管理，包括创建虚拟目录、设置目录权限、创建 Windows NT 用户等。
- 由于 ASP 本身是 CGI（Common Gateway Interface，通用网关接口）的一种，因此利用 ASP 可以实现利用 CGI 所能实现的一切功能，如计数器、留言簿、公告板、聊天室等，并且更加方便简单。

了解了 ASP 的功能后，通常还要在满足下面的条件时，才会考虑使用 ASP：

- 满足 ASP 的运行平台。由于 ASP 是用 IIS（在 Windows 98 中为 PWS）来解析的，因此最好运行在 Windows NT 或 Windows 98 平台上，尽管现在利用其他软件可以实现 ASP 跨平台使用，但在性能上会有所降低。

- 需要动态服务。如果需要在网站上添加一些动态服务并为不同级别的客户定制不同的服务时，可以用 ASP 轻松实现。
- 需要灵活的数据库管理。ASP 操作数据库的功能非常强大，因而当需要在网上建立 MIS（管理信息系统）系统（例如基于 Web 的 ERP 系统）时，可以使用 ASP。
- 需要提供安全的电子商务解决方案。ASP 结合数据库可以方便地建立企业电子商务应用解决方案，轻松实现网上商城。同时 CE 证书系统又可以提供强大的安全保障。

17.2　ASP 的运行环境

如前所述，ASP 通过 IIS 或 PWS（Person Web Server，个人 Web 服务器）来解析，通过一个 HTTP 请求的运行流程可以清楚地看出 ASP 对运行环境的要求。ASP 只有运行在 Microsoft 的操作平台上才可以发挥其最佳性能，因此，这里将只介绍在 Microsoft 的操作平台上配置 ASP 运行环境的方法。

17.2.1　Windows 98 操作系统

在 Windows 98 中，ASP 由 PWS 来解析，因此需要安装 PWS。在 Windows 98 的安装盘的 add-on 目录中有 PWS，找到这个目录并执行它的安装文件 setup.exe，按照它的提示一步一步安装即可。

安装完成后启动 PWS，在它的操作界面中单击“选择高级属性”按钮进行属性设置。这里主要是设置虚拟目录。选择要存放 ASP 文件的目录作为虚拟目录，为虚拟目录命名，并选中目录的执行权限复选项。这样放在已被设置为虚拟目录中的 ASP 文件就可以执行了。

关于 PWS 的详细设置，读者可以查看 PWS 的联机帮助或参考其他有关 PWS 的书籍。

17.2.2　Windows NT 4.0 操作系统

在 Windows NT 中执行 ASP 需要 IIS 的解析，因此需要安装 IIS。在 Windows NT 4.0 中集成了 IIS 2.0，若要升级 IIS，可以安装 Microsoft 的服务包——Windows NT Option Pack 4.0，在 Microsoft 的站点（Http://www.microsoft.com）中提供免费下载。安装完成后 IIS 将自动升级到了 4.0 版本。

关于 IIS 的详细设置，读者可以查看 IIS 的联机帮助或参考其他 IIS 的书籍。

17.2.3　Windows 2000 操作系统

在 Windows 2000 中集成了 IIS 5.0，可以直接配置 IIS 来建立自己的个人站点。IIS 5.0 中的 ASP 版本已经升级到了 3.0，新版本加入了一些很有用的特性，同时 IIS 的安全性也有所提高，因此推荐使用 Windows 2000 作为 ASP 的开发平台。

在 Windows 2000 中配置 ASP 运行环境的主要步骤如下：

（1）在 Windows 2000 桌面中，执行“开始”|“程序”|“管理工具”|“Internet 服务管理器”命令，启动 IIS。

（2）在如图 17.4 所示的 IIS 的操作界面中进行各项设置。右击“默认 Web 站点”节点，在弹出的快捷菜单中执行“新建”命令，分别创建自己的 Web 站点和虚拟目录。

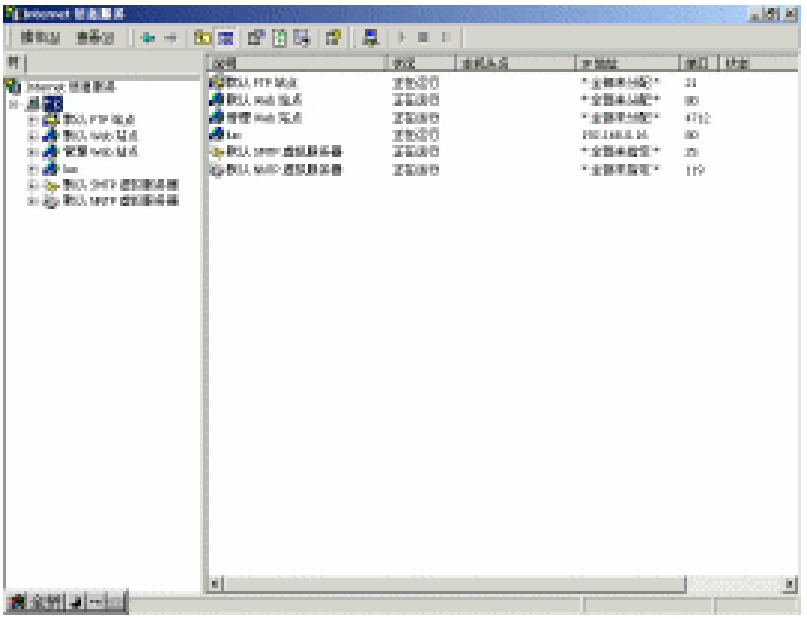

图 17.4 Windows 2000 的 IIS 的操作界面

（3）右击“默认 Web 站点”节点或者自己创建的 Web 站点，从弹出的快捷菜单中选择“属性”命令，打开“Luo 属性”（Luo 是作者的 Web 站点的名字）对话框，如图 17.5 所示，从中设置站点的主目录、安全性等属性。

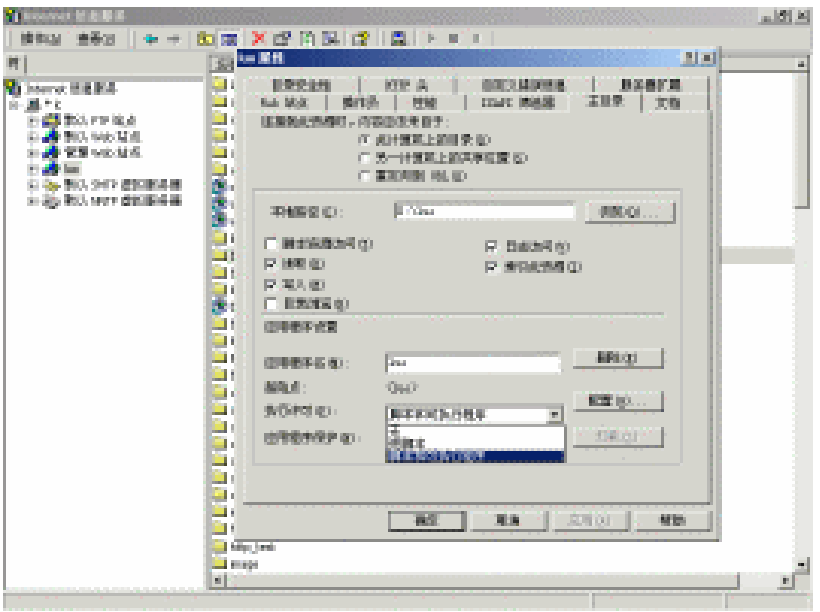

图 17.5 Windows 2000 设置站点的界面

（4）右击站点的目录，在弹出的快捷菜单中执行“属性”命令，在弹出的对话框中设置虚拟目录的各项属性，如图 17.6 所示（注意，图 17.4 为站点属性设置，而图 17.6 为目录属性设置）。

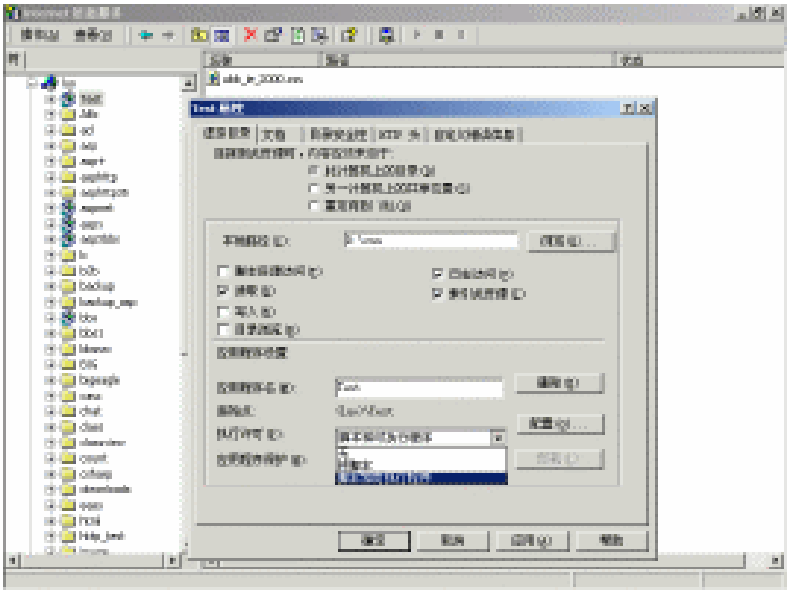

图 17.6　Windows 2000 的 IIS 中设置虚拟目录的界面

在进行完以上各个步骤后，就可以用浏览器浏览放在 Web 站点里的.asp 文件了。关于 IIS 的详细设置请参考 IIS 的联机帮助文档。

提示：可以将图 17.5 所示的“执行许可”下拉列表设置为“纯脚本”，这样整个站点就有了执行 ASP 文件的功能，包括它的各级子目录。同样必须选中执行纯脚本的选项，以便使这个虚拟目录有执行 ASP 的权限。这里不选择“脚本和可执行程序”这个选项是考虑到程序的安全性，因为允许用户在网站上执行可执行文件是十分危险的行为，这往往会给网站带来致命的攻击。

17.3　ASP 开发工具

应该说 ASP 对开发工具并没有很高的要求，因为.asp 文件是纯文本文件，因而用任何一种文本编辑工具都可以进行 ASP 的开发，例如记事本等。读者可以根据自己的爱好选择自己习惯的开发工具。目前流行的用于 ASP 开发的工具主要有：

1．Visual InterDev

这是 Microsoft 公司发布的官方开发工具。如果对 VB 或其他 Microsoft 的 Visual 系列开发工具熟悉的话，应该很习惯 Visual InterDev 的风格。它的主要特点有：

· 带有一个代码库，可调用预先设计好的 DTC（Design Time Control，设计时间控

件），非常简单地实现表单处理、数据库操作等复杂的功能。

- 可以对 ASP 代码进行颜色识别，自动完成代码，智能地列出对象的所有属性和方法。如输入 Response 之后，再输入“.”，Visual InterDev 会自动把 Response 的方法和属性列举出来以供选择，这对用户来说非常方便。
- 内置数据库管理工具，可以直接进行数据库查询、修改操作。
- 内置 RS（Remote Script），支持服务器、客户机间代码调用。
- 具有群体开发协作管理功能，支持多人同时开发一个网站。
- 内置其他组件。

但是，Visual InterDev 对 HTML 编辑的支持不太理想，在设计页面效果方面难以得心应手，这是其缺点。

2．HomeSite

如果要开发的网站不大，ASP 程序不是很庞大的话，可以使用 Allaire 公司的 HomeSite 软件来编辑 ASP 程序。作为一个 HTML 代码编辑器，HomeSite 对 HTML 的支持可以说是登峰造极。它具有代码颜色识别、自动完成、提示帮助等功能，可以使 HTML 编辑变得非常轻松。

在 HomeSite 4.0 及以后的版本中，提供了对 ASP 的简单支持，并设有一个 ASP 工具栏，上面有 ASP 常用的符号。HomeSite 也可以自动使 ASP 的几个常用词高亮显示，以帮助阅读代码。

3．其他工具

还有很多其他的编辑工具可以编辑 ASP 程序。只要是一种代码编辑器，可以进行简单编辑工作的软件均可，如 UltraEdit、HotDog 等等，可以根据个人喜好选用。

> 提示：可以根据不同的开发要求来选择不同的开发工具。当要开发的 ASP 程序比较大时，可以使用提示功能较强的 Visual InterDev，如果需要进行一些界面的设计，则可以使用 HomeSite 等类似 HTML 代码编辑器的工具来开发，但最好不要使用“所见即所得”的编辑器来编辑 ASP 程序，比如 FrontPage 98 和低版本的 Dreamweaver，因为它们都会对 ASP 代码产生错误识别，引起混乱。

17.4 脚 本 语 言

ASP 本身并不是一种开发语言，它需要借助脚本语言来实现其功能。脚本语言用来创建功能有限的“脚本”程序，以便在 Web 服务器或浏览器上执行 Web 站点的功能。与其他较复杂的编程语言不同，脚本语言是可以“解释”的，指令语句由中间程序（即命令解释程序）执行。解释过程降低了执行效率，但脚本语言简单易学并提供了强大的功能。脚本可以嵌入 HTML 页中，用来格式化内容，也可以用来实现包含高级商业逻辑的 COM 组件。

Web 开发人员可以在同一个 ASP 文件中使用多种脚本语言。由于脚本在服务器端读取和处理，因此，请求 ASP 文件的客户端浏览器并不需要支持脚本。另外，使用脚本语言，

必须在 Web 服务器上安装适合于该脚本语言的脚本引擎。

所谓“脚本引擎”就是一种处理以特定语言编写的命令的程序。ASP 提供了 Microsoft Visual Basic Scripting Edition（VBScript）和 Microsoft JScript 两种脚本引擎。也可以安装和使用其他脚本语言的引擎，如 REXX、Perl 或 Python 等。

17.4.1　VBScript 与 JScript

VBScript 只不过是 Visual Basic 的一个子集，因此，对于 Visual Basic 程序员，可以立即开始使用。尽管 JScript 与 Java 或 C 没有直接关系，但对于 Java、C 或 C++程序员，可能会发现 JScript 的语法并不陌生。

对于熟悉其他脚本语言（如 REXX、Perl 或 Python）的开发人员，可以获取并安装相应的脚本引擎，以便使用自己已经熟悉的语言。ASP 是一种 ActiveX 脚本，要使用脚本语言，必须安装一种脚本引擎，而且该脚本引擎需要符合 ActiveX 脚本标准，并作为 ActiveX 组件驻留在 Web 服务器上。

1. 设置基本脚本语言

ASP 基本脚本语言就是一种用于处理<%和%>分隔符内的命令的语言。默认的脚本语言为 VBScript。可以使用任何脚本语言作为基本脚本语言，但必须安装了该语言的脚本引擎。既可以逐页设置基本脚本语言，也可以一次性设置 ASP 应用程序中所有页的基本脚本语言。

2. 设置应用程序的语言

要对应用程序中的所有页设置基本脚本语言，只需在 IIS 单元的“应用程序选项”选项卡上设置“默认 ASP 语言”，如图 17.7 所示。

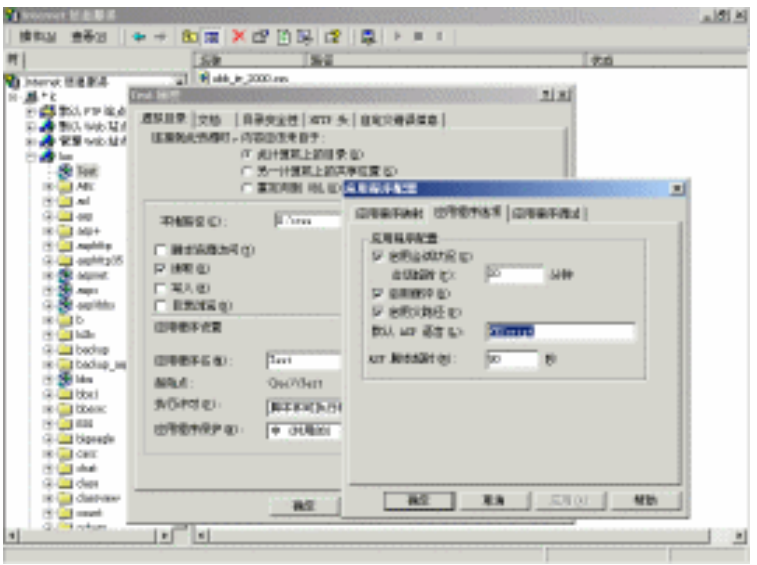

图 17.7　在 Windows 2000 的 IIS 中设置应用程序的脚本语言

3. 设置单独页的语言

要对单独页设置基本脚本语言，只需在.asp 文件开始处添加<%@ language %>。该命令的语法为：

```
<%@ language=ScriptingLanguage %>
```

其中，*ScriptingLanguage* 就是要对特定页设置的基本脚本语言，例如 VBScript。单独页的设置将覆盖对应用程序中所有页的全局设置，也就是说在单独页中如果设置了脚本语言，则应用程序中对脚本语言的设置将无效。

注意：要使用不支持 Object.Method 语法的语言作为基本脚本语言，必须首先创建 LanguageEngines 注册表键。

4. 在服务器上使用 VBScript 和 JScript

在服务器上使用 VBScript 和 ASP 时，将禁用两个 VBScript 特性。首先，因为使用 ASP 编写的脚本在服务器上执行，所以不支持用来提供用户界面元素的 VBScript 语句 InputBox 和 MsgBox。另外，不要在服务器端脚本中使用 VBScript 函数 CreateObject 和 GetObject。而应使用 Server.CreateObject，以便 ASP 可以跟踪对象例程。

关于所有 VBScript 和 JScript 操作符、函数、语句、对象、属性和方法的列表与说明，请参阅“VBScript 语言参考”和“JScript 语言参考”。可以从 http://msdn.microsoft.com-/scripting 站点上找到这些资料。

VBScript 和 JScript 最明显的区别是：前者不区分大小写，而后者区分大小写。例如，使用 Request 或 request 都可以引用 ASP Request 对象。同时也因为不区分大小写，所以无法通过大小写来区分变量名。例如，无法创建两个单独的变量 Color 和 color。

而在脚本中使用 JScript 关键字时，输入的关键字必须与参照页中的关键字完全一样。例如，如果使用 date 而不是 Date 就会导致错误。

17.4.2 第 1 个 ASP 程序

前面几节已经介绍了 ASP 的一些准备工作，下面开始写第 1 个 ASP 程序。

将下面的代码拷贝到记事本中，并命名为 test.asp，放到有 ASP 执行权限的目录中。用浏览器运行该程序，就可以看到如图 17.8 所示的结果。

```
<%@ language="VBScript" %>
<%
option explicit
Dim i
For i=1 To 10
  Response.Write "hello world!"
Next
%>
```

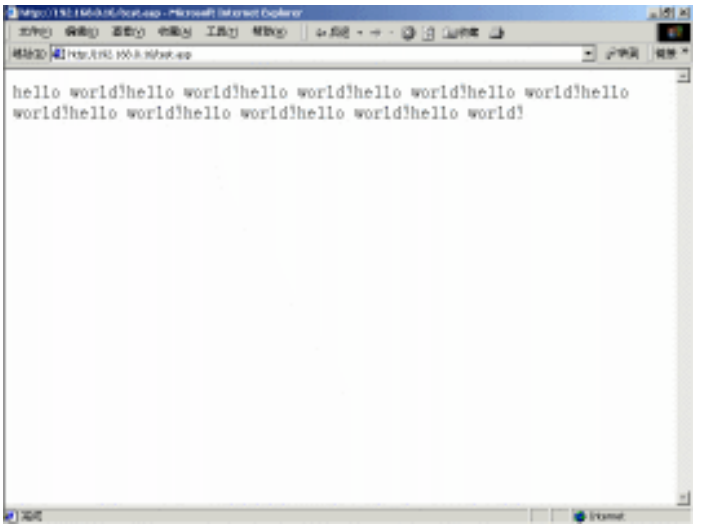

图 17.8　程序运行结果

可以看到 Web 页面上显示出了 10 个“ hello word !”字样。

在以上程序中，第 1 句代码指定脚本语言为 VBScript，第 3 句代码指定程序中必须显式声明变量，而后面第 4 句定义一个变量（可参考第 18 章“ VBScript 简介 ”），第 5 句和第 7 句是一个循环语句，第 6 句才是真正意义上的 ASP 语句，它利用一个 ASP 的内置对象在网页上显示“ hello word! ”字样（关于 ASP 的内置对象将在第 19 章讲述）。

17.5　小 结

本章介绍了关于 ASP 含义、基本功能、运行环境等一些基本知识，读者此时应该对 ASP 有了一个大概的了解，这些是开始正式学习 ASP 编程前的一些基本的准备工作。

由于 ASP 是要靠脚本语言来解释的，而且又经常要和 HTML 嵌套使用，因此要求读者必须具备一些 HTML 的知识，并且精通至少一门脚本语言。关于 HTML 的知识，读者可以参考前面的部分。

下一章将介绍 VBScript 脚本语言，为 ASP 编程打下坚实的基础。

第 18 章　VBScript 简介

除了 JavaScript 外，VBScript 也是 ASP 中使用很广泛的一门脚本语言，其语法规则和 VB 大致相同。作为 VB 的一个子集，它继承了 VB 语言的简单风格，如果读者有 VB 开发经验，那么使用 VBScript 就会感到非常得心应手。

18.1　VBScript 的数据类型和编码约定

同其他编程语言一样，VBScript 不仅有自己的数据类型，也有自己的编码约定。

18.1.1　VBScript 的数据类型

VBScript 只有一种真正意义上的数据类型：Variant。它是一种特殊的数据类型，根据其使用方式，可以包含不同类别的信息。

一个 Variant 类型的变量可以自动与赋给它的值的类型保持一致。当需要用一个变量代表一个数字时，只要把一个数字分配给该变量。而需要一个字符串时，只要将字符串分配给该变量，并用引号加以标记。除了数字和字符串以外，Variant 还包括其他的数值信息类型，称为子类型。表 18.1 列出了变量子类型和它们的取值范围。

表 18.1　Variant 变量子类型和取值范围

子类型	取值范围
Empty	未初始化的 Variant。对于数值变量，值为 0。对于字符串变量，值为 0 长度字符串（""）
Null	不包含任何有效数据的 Variant
Boolean	True 或 False
Byte	0 到 255 之间的整数
Integer	-32,768 到 32,767 之间的整数
Currency	-922,337,203,685,477.5808 到 922,337,203,685,477.5807
Long	-2,147,483,648 到 2,147,483,647 之间的整数
Single	单精度浮点数，负数范围从-3.402823E38 到-1.401298E-45，正数范围从 1.401298E-45 到 3.402823E38
Double	双精度浮点数，负数范围从-1.79769313486232E308 到-4.94065645841247E-324，正数范围从 4.94065645841247E-324 到 1.79769313486232E308
Date (Time)	表示日期的数字，日期范围从公元 100 年 1 月 1 日到公元 9999 年 12 月 31 日
String	变长字符串，最大长度可为 20 亿个字符
Object	对象
Error	错误号

另外，Variant 变量可以通过转换函数来改变数据的子类型。表 18.2 列出了各个转换函数及其功能。

表 18.2　转换函数及其功能

转换函数	功能
Asc	返回字符串的第 1 个字母所对应的 ANSI 字符代码
CBool	返回表达式，此表达式已被转换为 Boolean 子类型的 Variant
CByte	返回表达式，此表达式已被转换为 Byte 子类型的 Variant
CCur	返回表达式，此表达式已被转换为 Currency 子类型的 Variant
CDate	返回表达式，此表达式已被转换为 Date 子类型的 Variant
CDbl	返回表达式，此表达式已被转换为 Double 子类型的 Variant
Chr	返回与指定的 ANSI 字符代码相对应的字符
CInt	返回表达式，此表达式已被转换为 Integer 子类型的 Variant
CLng	返回表达式，此表达式已被转换为 Long 子类型的 Variant
CSng	返回表达式，该表达式已被转换为 Single 子类型的 Variant
CStr	返回表达式，该表达式已被转换为 String 子类型的 Variant
Hex	返回表示十六进制数字值的字符串
Oct	返回表示八进制数字值的字符串

例如：

```
Dim MyNum
MyNum=Asc("a")  '返回 97
MyNum=Asc("A")  '返回 65
```

18.1.2　VBScrip 的编码约定

使用一致的编码约定不仅可以使编码格式标准化，而且代码易于阅读和理解。

VBScript 的编码约定包含以下 3 大方面的内容：

- 对象、变量和过程的命名约定
- 注释约定
- 文本格式和缩进方面的约定

1. 常数命名约定

VBScript 的早期版本不允许创建用户自定义常数。如果要使用常数，则常数以变量的方式实现，且字母全部大写以和其他变量区分。例如：

```
MYSTRING_LIST
```

当然，也可以用 Const 语句创建真正的常数。允许大小写字母混合，并以 con 作为常数名的前缀。例如：

```
Const conMyString="good"
```

2. 变量命名约定

为了提高易读性和一致性，VBScript 使用了表 18.3 所列的变量命名约定。

表 18.3 VBScript 的变量命名约定

子类型	前缀	示例
Boolean	bln	blnFound
Byte	byt	bytRasterData
Date，Time	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFirstName

3. 对象命名约定

表 18.4 列出了 VBScript 中推荐的对象命名约定。

表 18.4 VBScript 的对象命名约定

对象类型	前缀	示例
3D 面板	pnl	pnlGroup
动画按钮	ani	aniMailBox
复选框	chk	chkReadOnly
组合框、下拉列表框	cbo	cboEnglish
命令按钮	cmd	cmdExit
公共对话框	dlg	dlgFileOpen
框架	fra	fraLanguage
水平滚动条	hsb	hsbVolume
图像	img	imgIcon
标签	lbl	lblHelpMessage
直线	lin	linVertical
列表输入框	lst	lstPolicyCodes
旋钮	spn	spnPages
文本框	txt	txtLastName
垂直滚动条	vsb	vsbRate
滑块	sld	sldScale

4. 注释约定

在 VBScript 中，对代码进行注释使得程序更容易阅读和理解。注释以 Rem 语句或撇

号开始，在一行的末尾结束，可以把注释与其他 Visual Basic 语句写在同一行中。例如：

```
<%
Rem This is a Example
For I=1 To 100  '开始一个循环
  MyString=MyString+1  '将字符串 MyString 并上 1
Next  '循环结束
%>
```

5. 文本格式和缩进方面的约定

以下为 VBScript 文档中推荐的几点：

- 标准嵌套块应缩进 4 个空格。
- 过程的概述注释应缩进 1 个空格。
- 概述注释后的最高层语句应缩进 4 个空格，每一层嵌套块再缩进 4 个空格。

当然读者可不必拘泥于这些约定，但在设计一个较大的程序时应当特别注意：文本格式和缩进的使用可以使代码格式化，并且应该反映程序的逻辑结构和嵌套关系。

18.2　VBScript 的变量

如 11.3 节所述，变量是计算机内存中已命名的存储位置，其中包含了数字或字符串等数据。变量包含的信息被称为变量的值。变量使用户便于理解脚本操作的名称，为用户提供了一种存储、检索和操作数据的途径。VBScript 中所有变量的数据类型都是 Variant。

18.2.1　变量命名规则

变量命名必须遵循 VBScript 的标准命名规则，即：

- 第 1 个字符必须是字母。
- 不能包含嵌入的句点。
- 长度不能超过 255 个字符。
- 在被声明的作用域内必须唯一。
- 变量的作用域由声明它的位置决定。如果在过程中声明变量，则只有该过程中的代码可以访问或更改变量值，此时变量具有局部作用域并被称为过程级变量。如果在过程之外声明变量，则该变量可以被脚本（Script）中的所有过程识别，称为 Script 级变量，具有 Script 级作用域。
- 变量存在的时间称为存活期。Script 级变量的存活期从被它声明的一刻起，直到脚本运行结束。对于过程级变量，其存活期仅是该过程运行的时间，该过程结束后，变量随之消失。在执行过程时，局部变量是理想的临时存储空间。可以在不同过程中使用同名的局部变量，这是因为每个局部变量只被声明它的过程识别。

18.2.2 声明变量

变量的声明分为显式和隐式两种。

显式声明是通过使用 Dim 语句、Public 语句或 Private 语句在脚本中实现的。例如：

```
Dim DegreesFahrenheit
```

如果要同时声明多个变量时，必须使用逗号将各个变量分隔开。例如：

```
Dim Top,Bottom,Left,Right
```

隐式声明是通过直接在脚本中使用变量名这一方式来声明变量的。这通常不是一个好习惯，因为这样有时会由于变量名被拼错而导致在运行 Script 时出现意外的结果。因此，最好使用 Option Explicit 语句显式声明所有变量，并将其作为 Script 的第 1 条语句。

18.2.3 给变量赋值

给变量赋值时，变量在表达式的左边，要赋的值在表达式的右边。例如：

```
A=100
```

多数情况下，只需为声明的变量赋一个值。只包含一个值的变量称为标量变量。有时候，将多个相关值赋给一个变量更为方便，因此可以创建包含一系列值的变量，称为数组变量。数组变量和标量变量是以相同的方式声明的，唯一的区别是声明数组变量时变量名后面带有括号。例如，Dim A(10)就表示声明了一个包含 11 个元素的一维数组。虽然括号中显示的数字是 10，但由于在 VBScript 中所有数组都是基于 0 的，所以这个数组实际上包含 11 个元素。在基于 0 的数组中，数组元素的数目总是括号中显示的数目加 1。这种数组被称为固定大小的数组。

在数组中利用索引为数组的每个元素赋值。例如，以下语句按索引号从 0 到 10，将数据赋给数组的元素。

```
A(0)=256
A(1)=324
A(2)=100
...
A(10)=55
```

同样，使用索引可以检索到指定索引号的数组元素。例如：

```
SomeVariable=A(8)
```

数组并不仅限于一维。数组的维数最大可以为 60（尽管大多数人不能理解超过 3 或 4 的维数）。声明多维数组时，用逗号分隔括号中每个表示数组大小的数字。在下例中，MyTable 变量是一个有 6 行和 11 列的二维数组。

```
Dim MyTable(5,10)
```

在二维数组中，括号中第 1 个数字表示行的数目，第 2 个数字表示列的数目。

也可以声明动态数组，即在运行 Script 时大小发生变化的数组。对数组的最初声明使用 Dim 语句或 ReDim 语句。但是对于动态数组，括号中不包含任何数字。例如：

```
Dim MyArray( )
ReDim AnotherArray( )
```

要使用动态数组，必须随后使用 ReDim 语句来确定其维数和每一维的大小。在下例中，ReDim 语句将动态数组的初始大小设置为 25，而后面的 ReDim 语句将数组的大小重新调整为 30，同时使用 Preserve 关键字在重新调整大小时保留数组的内容。

```
ReDim MyArray(25)
...
ReDim Preserve MyArray(30)
```

重新调整动态数组大小的次数是没有任何限制的，但是应注意：将数组的大小调小时，将会丢失被删除元素的数据。

18.3　VBScript 的函数与过程

跟其他编程语言一样，VBScript 提供了大量的函数（Function）和过程（Sub）以供用户方便地调用。用户也可定义自己的函数与过程来完成特定的功能。

18.3.1　过程

过程通过 Sub 和 End Sub 语句将一组 VBScript 脚本语句包含起来，它执行操作但不返回值。用户可以在过程中使用参数。如果过程无任何参数，则必须在 Sub 语句中包括空括号。例如：

```
<%
Sub ListDate( )
  Dim NowDate
  NowDate=Date  'Date 函数返回当前系统日期
  Response.Write  "当前时间为："&NowDate&"。"
End Sub
Call ListDate( )  '返回值为："当前时间为：XXXX-XX-XX"
%>
```

本例中，ListDate 过程通过调用 Date 系统函数得到当前的系统时间，然后使用 Call 函数调用 ListDate 过程得到返回值。

18.3.2　函数

与过程相似，函数通过 Function 和 End Function 语句将一组 VBScript 脚本语句包含起来。函数也可以使用参数。如果函数无任何参数，则 Function 语句必须包含空括号。函数

和过程最大的不同点是：函数通过函数名返回一个值，这个值是在函数的语句中赋给函数名的，它的数据类型总是为 Variant 类型。例如：

```
<%
Function ReturnMax(NumA,NumB)
  If NumA>=NumB Then
    ReturnMax=NumA
  Else
  ReturnMax=NumB
  End If
End Function
MyNum=ReturnMax(4,5)  '取得值大者
Response.Write "The Max Number is "&MyNum
%>
```

本例实现了由两个不同的数取最大值。通过 If…Then…Else 语句从函数 ReturnMax 的两个参数 *NumA* 和 *NumB* 中取得值较大者，将其结果返回给变量 MyNum。

18.3.3 过程和函数的调用

过程通过 Call 语句来调用，当然，用户也可以直接调用过程，如果其后有参数则直接写出而不需要括号。其形式如下：

```
Sub Proc(para1,para2,...)
...
End Sub
Call Proc(para1,para2,...)
```

或者

```
Proc para1,para2,...
```

调用函数时，函数名必须用在变量赋值语句的右端或表达式中。其形式如下：

```
Function Func(para1,para2,...)
...
End Function
Temp=Func(para1,para2,...)
```

或者

```
Func(para1,para2,...)
```

18.4 VBScript 的条件语句和循环语句

对于任何一门编程语言而言，要对程序的流程进行控制就要用到条件语句和循环语句。

当然 VBScript 也不例外。

18.4.1　条件语句

在 VBScript 中，用户可以使用以下两种条件语句：

- If…Then…Else 语句
- Select Case 语句

If...Then...Else 语句首先计算条件是否为 True 或 False，然后根据计算结果指定要运行的语句。通常，条件是使用比较运算符对值或变量进行比较的表达式。

先来看一个简单的例程。

```
<%
If Num>1000 Then Character="很大的数字"
%>
```

这个例子中，当数字大于 1000 时，把值“ 很大的数字 ”赋给 Character。需要注意的是，如果要运行的代码是多行的话，必须使用语句 End If 来指定分支代码块的结束。例如：

```
<%
If Num>1000 Then
  Character="很大的数字"
  Color="Red"
End If
%>
```

本例表示当数字大于 1000 时，把值“ 很大的数字 ”赋给 Character，把“ Red ”赋给 Color。

上面两个例子说明了条件只为 True 情况下的使用方法，但当条件中 True 和 False 同时存在时，就可以使用完整的 If...Then...Else 语句来实现：条件为 True 时运行某一语句块，条件为 False 时运行另一语句块。例如：

```
<%
If Num>1000 Then
  Character="很大的数字"
Else
  Character="小数字"
End If
%>
```

另外还可以通过添加 ElseIf 语句来对 If...Then...Else 语句进行扩充，使用可以控制基于多种可能的程序流程。例如：

```
<%
If Num>1000 Then
```

```
  Character="很大的数字"
ElseIf Num>100 Then
  Character="比较小的数字"
Else
  Character="很小的数字"
End If
%>
```

此外，进行嵌套也是一种解决多分支情况的方法。例如：

```
<%
If Num>1000 Then
  If Num>5000 Then
    Character="巨大的数字"
  ElseIf Num>2000 Then
    Character="极大的数字"
  Else
    Character="比较大的数字"
  End IF
Else
  Character="小数字"
End If
%>
```

在上面的例子中加入一条给 Num **赋值的语句：**

```
Num=5000
```

然后将 Character **显示出来：**

```
Response.Write Character
```

根据 Num **的不断变化，**Character **也做了相应的改变。**

在 If...Then...Else **语句中，通过使用多个** ElseIf **子句对程序流程进行控制经常会使程序变得很繁琐。所以在多个条件中进行选择的更好方法是使用** Select Case **语句。**

Select Case **结构在其开始处使用一个只计算一次的简单测试表达式。表达式的结果将与结构中每个** Case **的值比较。如果匹配，则执行与该** Case **关联的语句块。例如：**

```
Select Case VisitorName
  Case "Rose"
    SayWords="Welcoming! Rose"
  Case "Jack"
    SayWords="Welcoming! Jack"
  Case "Brown"
    SayWords="Welcoming! Brown"
  Case "John″
```

```
    SayWords="Welcoming! John"
  Case Else
    SayWrods="Welcoming! Everyone"
End Select
```

其中 Case Else **语句是可选的。如果省去表达式，并且所有的** Case **条件都没有匹配，则不执行任何语句。**

18.4.2　循环语句

循环语句的作用就是重复执行一段程序代码。循环可分为 3 **类：第** 1 **类在条件变为** False **之前重复执行语句，第** 2 **类在条件变为** True **之前重复执行语句，第** 3 **类按照指定的次数重复执行语句。**

VBScript **中可以使用的循环语句有：**

- Do…Loop**：当（或直到）条件为** True **时循环。**
- While...Wend**：当条件为** True **时循环。**
- For...Next**：指定循环次数，使用计数器重复运行语句。**
- For Each…Next**：对于集合中的每项或数组中的每个元素，重复执行一组语句。**

1. Do…Loop 语句

Do…Loop **语句可以多次（次数不定）运行语句块，当条件为** True **时或条件变为** True **之前，重复执行语句块。**

用 While **关键字检查** Do…Loop **语句重复执行的条件，当条件为** True **时则重复执行。有两种检查方式：在进入循环之前检查条件，或者在循环至少运行完一次之后检查条件。**

下面来看以下两个例程。

```
<%
Dim I,Total
I=10
Total=0
Do While I>5
  Total=Total+1
  I=I-1
Loop
Response.Write "Total is "&Total  '结果显示为："Total is 5"
%>

<%
Dim I,Total
I=10
Total=0
Do
  Total=Total+1
```

```
  I=I-1
Loop While I>5
Response.Write "Total is "&Total  '结果显示为："Total is 5"
%>
```

这时，并没有看出两种循环的区别。但是当把上两例中 I 的值设为 5 时，再运行程序，就会发现第 1 个例子中显示的结果为："Total is 0"，而第 2 个例子显示的结果为："Total is 1"。也就是说第 1 个例程当执行到 Do…Loop 循环语句时，先对 While 条件进行了判断并返回结果 False，所以跳过 Do…Loop 循环直接执行了下面的语句，返回了"Total is 0"的结果；而在第 2 个例程中，当运行到 Do…Loop 循环语句时，程序并没有立即对条件进行判断，而是首先执行了 Total=Total+1 和 I=I–1 这两条语句，然后再对 While 条件进行判断并返回 False，跳出循环，执行下面的语句。在这里，程序已经运行了 Total=Total+1 一遍，所以返回了"Total is 1"的结果。

对于 Do…Loop 循环，用关键字 Until 来检查重复执行的条件，条件为 True 则重复执行。同样有两种检查方式：在进入循环之前检查条件，或者在循环至少运行完一次之后检查条件。只要条件为 False，就会进行循环。

这里同样用两个例程来说明。

```
<%
Dim I,Total
I=10
Total=0
Do Until I=5
  Total=Total+1
  I=I-1
Loop
Response.Write "Total is "&Total  '结果显示为："Total is 5"
%>

<%
Dim I,Total
I=10
Total=0
Do
  Total=Total+1
  I=I-1
Loop Until I=5
Response.Write "Total is "&Total  '结果显示为："Total is 5"
%>
```

另外有一个关键字要提一下：Exit Do，它的作用是退出 Do…Loop 循环。

当程序在一些特殊的情况下要退出循环（如避免进入死循环）时，可以在 If...Then...Else 语句的 True 语句块中使用 Exit Do 语句。如果条件为 False，循环将照常运行。例如：

```
<%
Dim I,Total
I=10
Total=0
Do Until I=5
  Total=Total+1
  I=I-1
  If I<5 Then Exit Do
Loop
Response.Write "Total is "&Total  '结果显示为："Total is 5"
%>
```

2. While…Wend 语句

While…Wend 当其条件的值为 True 时，重复执行。

下面用一个例程来说明。

```
<%
Dim I,Total
I=10
Total=0
While I>0
  Total=Total+1
  I=I-1
Wend
Response.Write "Total is "&Total  '结果显示为：Total is 10
%>
```

While…Wend 语句是为那些熟悉其用法的用户提供的。由于 While…Wend 循环语句缺少灵活性，所以建议最好使用 Do…Loop 循环语句。

3. For...Next 语句

当程序需要指定循环运行的次数时，就用到了 For…Next 语句。在循环中使用计数器来控制次数，每循环一次计数器增加或减少。例如：

```
<%
Dim I,Sum
Sum=0
For I=1 to 100
  Sum=Sum+I
Next
Response.Write "Total Number is "&Sum  ’结果为 5050
%>
```

如果需要对计数器每次的增减量进行控制，就用到关键字 Step。如果计数器变量是递减的，可将 Step 设为负值。此时计数器变量的终止值必须小于起始值。例如：

```
<%
Dim I,Sum
Sum=0
For I=100 to 1 Step -2
  Sum=Sum+I
Next
Response.Write "Total Number is "&Sum  '结果为 2550
%>
```

在 For…Next 循环中，Exit For 语句用于在计数器达到其终止值之前退出 For…Next 语句。与 Do…Loop 语句相似，为了避免某些特殊情况的出现而直接退出循环（例如在发生错误时），可以在 If...Then...Else 语句的 True 语句块中使用 Exit For 语句。如果条件为 False，循环将照常运行。

4. For Each...Next 语句

For Each...Next 循环与 For...Next 循环类似。For Each...Next 不是将语句运行指定的次数，而是对于数组中的每个元素或对象集合中的每一项重复执行一组语句。For Each…Next 适用于对数组或集合的元素进行循环。其语法如下：

```
For Each element In group
  [statements]
  [Exit For]
  [statements]
Next [element]
```

下面用一个例程来说明。

```
<%
Dim A
A=Array(10,20,30)
For Each I In A
  Response.Write I&"<br>"
  If I=30 then Exit For
Next
%>
```

至此，已经完成了对脚本语言 VBScript 的基本知识的学习，但是如果想要熟练运用 VBScript，还要通过不断实践来提高自己的编程水平。

18.5 小 结

本章使读者对 VBScript 有了一个大致的了解，至此就已经掌握了开发 ASP 的准备知识，从下一章开始将正式介绍 ASP 程序开发。

第 19 章　ASP 的内置对象

ASP 提供了一些内置对象，这些对象使用户更容易收集通过浏览器请求发送的信息、响应浏览器以及存储用户信息。这些内置对象可以说是 ASP 的基础，很多功能都是通过内置对象来实现的。本章将分别介绍这些内置对象。

19.1　Request 和 Response 对象

Request 和 Response 对象是 ASP 最常用的两个对象，它们用来进行访问和管理在浏览器（客户端）和 Web 服务器之间请求与响应中产生的信息。

19.1.1　Request 对象

Request 对象主要用来接收客户端浏览器向服务器通过 HTTP 请求发送的内容。可以使用 Request 对象访问任何用 HTTP 请求传递的信息，包括从 HTML 表格用 post 方法或 get 方法传递的参数、Cookie 和用户认证。Request 对象使用户能够访问发送给服务器的二进制数据，如上载的文件。

首先介绍 Request 对象集合。多数 ASP 内置对象提供集合。“集合”是类似于数组的数据结构，可存储字符串、数值、对象和其他值。与数组不同的是，集合可以根据所获得或存储的项目自动伸缩。随着集合发生变化，项目的位置也将变动。要访问集合中的项目，可以通过项目惟一的字符串关键字、项目在集合中的索引（位置）或迭代集合中的所有项目来实现。

Request 对象提供了 5 个集合，可以用来访问客户端对 Web 服务器请求的各类信息。

1. ClientCertificate 集合

表示客户证书，当浏览器访问 Web 页时，表明客户身份证明字段的数值集合。客户端浏览器必须支持 SSL 3.0 或 PCT1 协议。这里需做两步工作：第 1 步，Web 服务器必须启动用户端认证选项；第 2 步，对客户端浏览器进行相应设置，这样该方法才会生效，否则传回 Empty 值。

这个对象用得并不多，这里就不详细介绍了。

2. Form 集合

接受 post 方法传递过来的 HTTP 请求的表单元素的值的集合。

注意：要区分这里的 Form 和 HTML 中的 Form，这里的是指 Request 的一个集合，而 HTML 中的 Form 是指表单。

使用 Form 传送信息一般有 3 种方式：由 HTML 网页内的表单信息给其他 ASP 页；由

ASP 页内的 Form 传送信息给另外一个 ASP 页；ASP 页内的信息传送给自身。这里的 Request.Form 是接受 method=post 的表单传递的值。

Form 的语法如下：

```
Request.Form(element)[(index)|.Count]
```

其中：

- *element*：指定集合中要检索的表单元素的名称。
- *index*：可选参数，使用该参数可以访问某参数的多个值中的一个。它可以是从 1 到 Request.Form(*parameter*).Count 之间的任意整数。
- Count：属性，代表集合中元素的个数。

以下为一段示例代码。

```
<%
For I=1 To Request.Form("user").Count
  Response.Write Request.Form("user")(I)
Next
%>
```

这段代码循环读取名称为 user 的元素，其中 Count 是 Form 的系统属性，用于计算同名元素的个数，如果该元素不存在，则其值为 0。如果没有指明要读取第几个同名元素，则系统会把全部同名元素的值都读取出来，并用“，”作间隔。例如：

```
Request.Form("user")=abc,bcd,cde
```

3. QueryString 集合

依附于用户请求的 URL 后面的名称/数值对，或者作为请求提交且 method 属性为 get（或者省略其属性）的，或表单中所有 HTML 控件单元的值的集合。

QueryString 的语法如下：

```
Request.QueryString(variable)[(index)|.Count]
```

其中：

- *variable*：变量，表示传递过来的元素名称。
- *index*：可选参数，使用该参数可以访问某参数中多个值中的一个。它可以是从 1 到 Request.QueryString(*variable*).Count 之间的任意整数。
- Count：属性，代表集合中元素的个数。

同样可以列出 QueryString 集合的所有元素的值，示例代码如下：

```
<%
For I=1 To Request.QueryString("user").Count
  Response.Write Request.QueryString("user")(I)
Next
```

```
%>
```

注意：除了用 Form **集合传输资料外，还可通过在超链接后接“？”的方式传输信息，如**:<A href="***.asp?user=abc">**，这时可通过** Request.QueryString("user") **读取传输的信息。如果出现多个重名的参数，如**:

```
<A href="***.asp?user=abc&user=bcd&user=cde">
```

第 1 **次：**Resquest.QueryString("user")=abc

第 2 **次** Resquest.QueryString("user")=bcd

第 3 **次：**Resquest.QueryString("user")=cde

4. ServerVariables **集合**

接收的随同客户端请求发出的 HTTP **报头值，以及** Web **服务器的几种环境变量的值的集合。**

ServerVariables **的语法如下：**

```
ServerVariables(server_environment_variable)
```

其中参数 *server_environment_variable* **是指服务器环境变量。**

Web/Browser **传输协议是** HTTP，HTTP **的报头会有一些客户端的信息，例如客户机的** IP **地址、浏览器的语言系统等。这时可通过** Request.ServerVariables("***")**获取相关信息，如** Request.ServerVariables("Accept_Language")**可获取客户端浏览器的语系。表** 19.1 **列出了一些常用的服务器环境变量。**

表 19.1　**常用的服务器环境变量**

服务器环境变量名	描述
SERVER_NAME	服务器的机器名称或 IP 地址
SERVER_PORT	服务器正在运行的端口号
REQUEST_METHOD	发出请求的方法（get/post/head）
SCRIPT_NAME	程序被调用的路径，如：cgi-bin/a.pl
REMOTE_HOST	发出请求的远端机器（Client）的名称
ALL_HTTP	客户端发送的所有 HTTP 标题文件
REMOTE_ADDR	发出请求的远端机器（Client）的 IP 地址
REMOTE_IDENT	发出请求的使用者名称(如是拨号上网,则为用户 ID),当 NCSA IdentityCheck 为 enabled，而且 Client 机器支持 RFC 931 时，该变量有效
CONTENT_TYPE	数据的 MIME 类型，如：text/html
CONTENT_LENGTH	客户端发出内容的长度
HTTP_USER_AGENT	客户端发出请求的浏览器类型
HTTP_REFERER	在读取 CGI 程序之前，客户端所指的文本 URL
HTTP_ACCEPT	客户端可以接受的 MIME 类型列表
LOCAL_ADDR	返回接受请求的服务器地址。如果在绑定多个 IP 地址的多宿主机器上查找请求所使用的地址时，这个变量非常重要

运行以下 ASP 程序，可以在 Web 页面上一次性列出所有的服务器环境变量，如图 19.1 所示。

```
<html>
<title>列出所有服务器环境变量</title>
<%
Dim name
For Each name In Request.ServerVariables
  Response.Write "<b>◆"&name&"</b>"&chr(32)
Next
%>
</html>
```

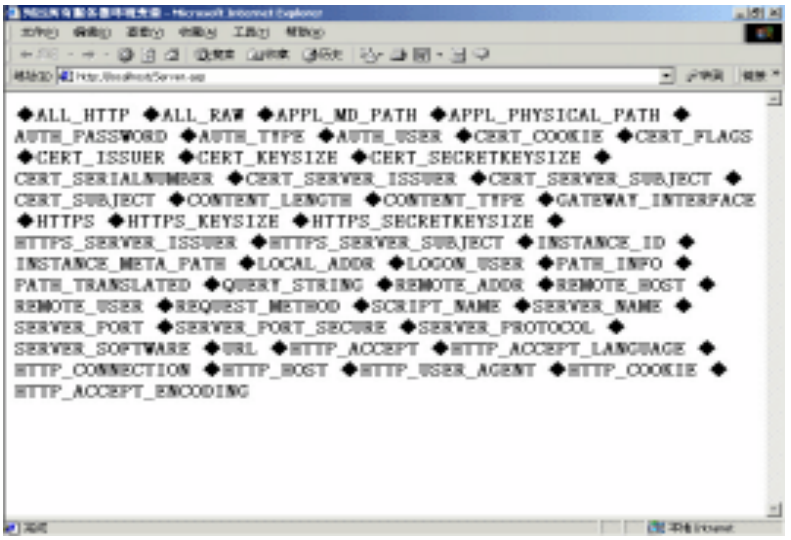

图 19.1 列出所有服务器环境变量

5. Cookies 集合

在 HTTP 协议下，服务器或脚本用来保留客户机信息的一种方法。

Cookies 是通过 Web 服务器存储在用户浏览器中的小文本文件。Cookies 包含有关用户的信息，如标识号、密码、用户如何在 Web 站点选购以及用户访问该站点的次数等。Web 站点可以在用户连接到服务器的任何时候访问 Cookie 信息。

Cookies 的语法如下：

```
Request.Cookies(cookie)[(key)|.attribute]
```

其中：

- *cookie*：要访问的 Cookie 的名称。
- *key*：Cookies 集合中的子键，可选。例如：

```
Request.Cookies("User")("Name")
```

```
Request.Cookies("User")("Pwd")
```

- *attribute*：Cookie 自身的特殊信息，可选。例如，HasKeys 指定 Cookie 是否包含关键字。

```
<%= Request.Cookies("myCookie").HasKeys %>
```

在客户端，Cookies 记录了客户端浏览器的很多信息，可通过 Request.Cookies(*cookie*)命令获取它的值，也可通过 Response.Cookies(*cookie*)=*value* 在客户端记录一些信息，以便控制访问者。设置多重 Cookies 的方法为 Response.Cookies(*cookie1*)(*cookie2*)= *value*。

注意：要保存信息到 Cookies 或从客户端 Cookies 读取信息，客户端必须支持并允许操作 Cookies。

下面介绍 Request 对象的方法和属性。

Request 对象的主要方法为 BinaryRead(*count*)，当数据作为 Post 请求的一部分发往服务器时，从客户请求中获得 *count* 字节的数据，返回一个 Variant 数组（或者 SafeArray）。

注意：在同一 ASP 文件中不能同时使用 Request 对象的 Form 集合和 BinaryRead 方法，如果 ASP 代码已经引用了 Form 集合，就不能用 BinaryRead 方法。同样，如果用了 BinaryRead 方法，就不能访问 Form 集合。

Request 对象的主要属性为 TotalBytes，它返回由客户端发出的请求的整个字节数量。以下为示例代码：

```
<%
Dim vntPostedData,lngCount
lngCount=Request.TotalBytes
vntPostedData=Request.BinaryRead(lngCount)
Response.Write IngCount&chr(32)&vntPostedData
%>
```

上面程序先获得客户端发送的请求的字节数量，然后再用 BinaryRead 方法来处理这个数值，返回一个新的数值。如果运行上面的程序，读者会发现 TotalBytes 要大于等于 BinaryRead(*count*)。

19.1.2　Response 对象

与 Request 用来获取客户端 HTTP 信息相反，Response 对象用来控制发送给用户的信息，包括直接发送信息给浏览器、重定向浏览器到另一个 URL 或设置 Cookie 的值。

Response 对象只有一个集合，那就是 Cookies。

这里的 Cookies 是从 Web 服务器向客户端设置 Cookie 的值，其功能和 Request.Cookies 刚好相反。如果客户端 Cookies 已经存在，就会替换原来的值，如果还没有创建，则会创建并赋值。

Cookies 的语法如下：

```
Response.Cookies(cookie)[(key)|.attribute]=value
```

其中：

- *cookie*：Cookie **的名称。**
- *key*：Cookies **集合中的子键，可选。例如：**

```
Reponse.Cookies("User")("Name")
Reponse.Cookies("User")("Pwd")
```

- *attribute*：**设置** Cookie **自身的特殊信息，可选。例如设置** Cookie **的应用域、路径及其有效期。**

设置 Cookies **的作用域：**

```
Response.Cookies(item-name).domain=domain-url
```

设置 Cookies **的作用路径：**

```
Response.Cookies(item-name).path=virtual-path
```

设置 Cookies **的过期时间：**

```
Response.Cookies(item-name).expires=#date#
```

创建一个单值的 Cookies：

```
Response.Cookies(item-name)=item-value
```

创建一个多值的 Cookies：

```
Response.Cookies(item-name)(sub-item-name)=sub-item-value
```

注意 通常,只有当客户端对创建 Cookie **的目录中的页面提出请求时,才将** Cookie **随请求发住服务器。通过指定** path **属性，可以指定站点中何处这个** Cookie **是合法的，并且这个** Cookie **将随请求发送。如果** Cookie **随对整个站点的页面请求发送，则设置** path **为**/**。**

如果 Expires **属性没有设置，那么当关闭当前的浏览器实例时，**Cookie **将被自动消除。**

Response **对象的属性主要有：**

1. Buffer **属性**

表明由一个 ASP **页所创建的输出是否一直存放在** IIS **缓冲区，直到当前页面的所有服务器脚本处理完毕或** Flush、End **方法被调用。在任何输出（包括** HTTP **报头信息）送住** IIS **之前这个属性必须设置。因此在**.asp **文件中，这个设置应该在**<%@ language=…%>**语句后面的第** 1 **行。该属性可读可写，为布尔型。**

Buffer **的语法如下：**

```
Response.Buffer=True|False
```

注意：除了 ASP 3.0 中默认设置缓冲（Buffer）为开（True）外，早期版本中均默认为关（False）。

2. CacheControl 属性

用来设置是否允许代理服务器缓存页面。该属性可读可写，为字符型。

CacheControl **的语法如下：**

```
Response.CacheControl[=Cache_Control_Header]
```

Cache_Control_Header **为** Public **表示允许代理服务器缓存页面，如为** Private **则禁止代理服务器缓存的发生。**

3．Charset 属性

在由服务器为每个响应创建的 HTTP Content-Type **报头中附上所用的字符集名称（例如：**ISO-LATIN-7**）。该属性可读可写，为字符型。**

Charset **的语法如下：**

```
Response.Charset(CharsetName)
```

例如：

```
<% Response.Charset="ISO-LATIN-7" %>
```

4. ContentType 属性

指明响应的 HTTP **内容类型，是标准的** MIME **类型（例如** text/xml **或者** Image/gif**）。默认情况下表示使用** MIME **类型** text/html**。内容类型告诉浏览器所期望内容的类型。该属性可读可写，为字符型。**

ContentType **的语法如下：**

```
Response.ContentType[=ContentType]
```

5. Expires 属性

指明页面的以分钟计算的有效时间长度。假如用户请求其有效期满之前的相同页面，将直接读取显示缓冲中的内容，这个有效期间过后，页面将不再保留在私有（用户）或公用（代理服务器）缓冲中。该属性可读可写，为数字型。

注意：可以设置 Response.Expires=0**，使缓存的页面立即过期。这是一个较实用的属性，当客户通过** ASP **的登录页面进入** Web **站点后，应该利用该属性使登录页面立即过期，以确保安全。**

Expires **的语法如下：**

```
Response.Expires[=number]
```

6. ExpiresAbsolute 属性

指明当一个页面过期和不再有效时的绝对日期和时间。该属性可读可写，为时间日期型。在未到期之前，若用户返回到该页，缓存中的该页面就显示。如果未指定时间，该页在指定日期的 0 点整到期。如果未指定日期，则该主页在脚本运行当天的指定时间到期。

ExpiresAbsolute 的语法如下：

```
Response.ExpiresAbsolute[=[date][time]]
```

7. IsClientConnected 属性

用来返回客户端是否仍然保持连接或正在下载页面的状态标志。在当前的页面执行完毕之前，假如一个客户转移到别一个页面，可用这个标志来中止处理（使用 End 方法）。该属性只读，为布尔型。

IsClientConnected 的语法如下：

```
Response.IsClientConnected
```

8. PICS 属性

用来创建一个 PICS 报头定义页面内容中的词汇等级。其语法如下：

```
Response.PICS(PICSLabel)
```

9. Status 属性

指明发回客户端的响应的 HTTP 报头中表明错误或页面处理是否成功的状态值和信息。其语法如下：

```
Response.Status=StatusDescription
```

例如：

```
<% Response.Status="401 Unauthorized" %>
```

Response 提供的方法主要有：

1．AddHeader 方法

AddHeader 的语法如下：

```
Response.AddHeader name,value
```

其中：

- *name*：HTTP 报头的名称。
- *value*：属性值。

通过 *name* 和 *value*，可以创建一个定制的 HTTP 报头，并增加到响应之中。不能替换现有的相同名称的报头。一旦已经增加了一个报头就不能将其删除。

例如：

```
<% Response.AddHeader "WARNING","Error Message Text" %>
```

注意：这个方法必须在任何页面内容被发往客户端前使用。

2．AppendToLog 方法

当使用 W3C Extended Log File Format **文件格式时，对于用户请求的** Web **服务器的日志文件增加一个条目。至少要求在包含页面的站点的“扩展属性”属性页中选择“**URL **资源”。在** Windows 2000 **中的设置步骤为：**

（1）在 IIS **控制面板中右键单击要设置的站点，在弹出的菜单中执行“属性”命令，出现如图** 19.2 **所示对话框。**

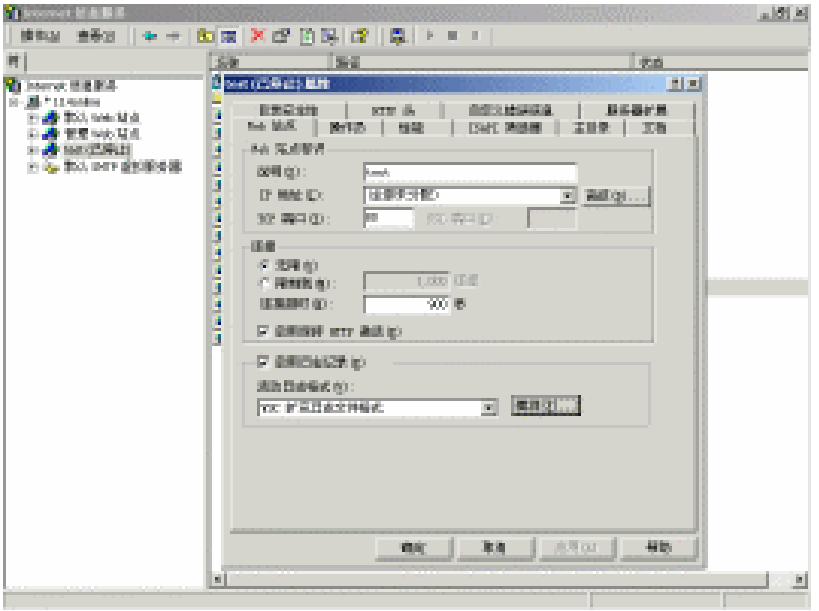

图 19.2　**设置站点属性对话框**

（2）单击“属性”按钮，在出现的对话框中选择“扩展属性”属性页，然后选中“URL **资源”，如图** 19.3 **所示。**

AppendToLog **的语法如下：**

```
Response.AppendToLog string
```

例如：

```
<% Response.AddToLog "My custom log message" %>
```

3．BinaryWrite 方法

在当前的 HTTP **输出流中写入** Variant **类型的** SafeArray**，而不经过任何字符转换。该方法对于写入非字符串的信息，例如定制的应用程序请求的二进制数据或组成图像文件的二进制字节，是非常有用的。**

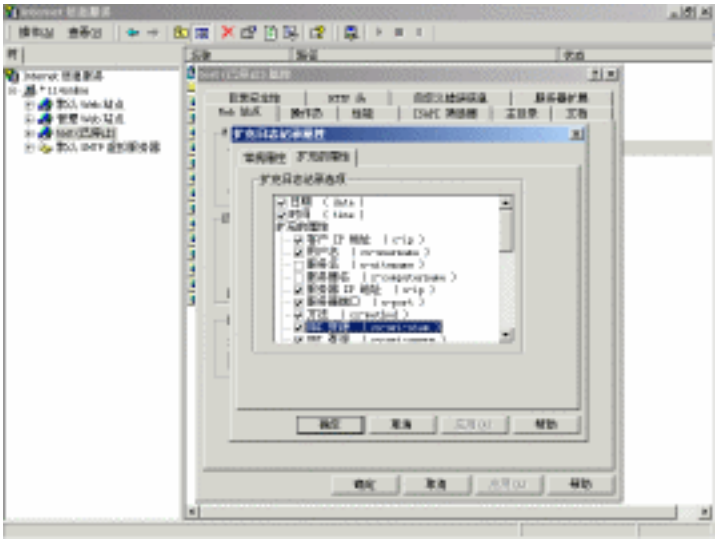

图 19.3 设置扩充日志记录属性

BinaryWrite 的语法如下：

```
Response.BinaryWrite data
```

4．Clear 方法

可以用 Clear 方法清除缓冲区中的所有 HTML 输出。但 Clear 方法只清除响应正文而不清除响应标题。可以用该方法处理错误情况。但是如果没有将 Buffer 属性设置为 True，则该方法将导致运行时错误。

Clear 的语法如下：

```
Response.Clear
```

5．End 方法

End 方法使 Web 服务器停止处理脚本并返回当前结果。文件中剩余的内容将不被处理。如果 Buffer 属性已设置为 True，则调用 End 将缓冲输出。

End 的语法如下：

```
Response.End
```

6．Flush 方法

Flush 方法立即发送缓冲区中的输出，可以用来发送较大页面的部分内容给个别的用户。如果没有将 Buffer 属性设置为 True，则该方法将导致运行时错误。

Flush 的语法如下：

```
Response.Flush
```

7．Redirect 方法

Redirect 方法使浏览器立即重定向到程序指定的 URL，发送一个 302 Object Moved 的

HTTP 报头。一旦使用了 Redirect 方法，任何在页中显式设置的响应正文内容都将被忽略。然而，此方法不向客户端发送该页设置的其他 HTTP 标题，将产生一个将重定向 URL 作为链接包含的自动响应正文。

Redirect 的语法如下：

```
Response.Redirect URL
```

例如：

```
<% Response.Redirect "http://www.microsoft.com" %>
```

7．Write 方法

将指定的字符串写到当前的 HTTP 输出。其语法如下：

```
Response.Write variant
```

例如：

```
<% Response.Write "hello world!" %>
```

注意：在 ASP 中，Response.Write 和<% =variant%>的效果一样，<%= variant%>会被解释成 Response.Write *variant*。例如上面的<% Response.Write ″hello world!″ %>就相当于<% =world%>（world=″hello world!″）。

上面介绍完了 Response 和 Request 的所有的属性、集合和方法。这两个对象是 ASP 的两个最基本的对象。无论用户何时从 Web 网站请求和载入一个网页或资源，这种会话就会进行。这意味着 Request 对象能够提供对用户请求的全部内容的访问，同时 Response 对象允许创建和修改服务器发回的响应。

19.2　Application 和 Session 对象

Application 和 Session 对象也是 ASP 中的两个重要对象。可以利用 Application 对象使给定应用程序的所有用户共享数据，使用 Session 对象保存特定的用户会话所需的信息。

19.2.1　Application 对象

在同一虚拟目录及其子目录下的所有 ASP 文件构成了 ASP 应用程序。我们不但可以使用 Application 对象在给定的应用程序的所有用户之间共享信息，并在服务器运行期间持久地保存数据。而且，Application 对象还有控制访问应用层数据的方法，以及可用于在应用程序启动和停止时触发过程的事件。

Application 对象提供了 Contents 和 StaticObjects 两个集合。

1. Contents 集合

没有使用<object>标记定义的存储于 Application 对象中的所有变量（及它们的值）的一个集合，包括 Variant 数组和 Variant 类型对象实例的引用。

Contents 的语法如下：

```
Application.Contents(Key)
```

其中参数 *key* 为 Application 的键。

下面的代码列出了 Contents 集合中所有的键。

```
<%
Dim Key
For Each Key in Application.Contents
  Response.Write Key+"="+Application(Key)+"<br>"
Next
%>
```

2. StaticObjects 集合

使用<object>标记定义的存储于 Application 对象中的所有变量（及它们的值）的一个集合。

StaticObjects 的语法如下：

```
Application.StaticObjects(Key)
```

下面代码列出了 StaticObjects 集合中所有的键。

```
<%
Dim strKey
For Each strKey In Application.StaticObjects
  Response.Write strKey&"=<i>(object)</i><br>"
Next
%>
```

下面再来看 Application 提供的方法。

Application 对象的方法允许删除全局应用程序空间中的值，控制在该空间内对变量的并发访问。它一共提供了 4 个方法。

1. Remove 方法

从 Application.Content 集合中删除一个名为 *variable_name* 的变量。

Remove 的语法如下：

```
Application.Contents.Remove(name|index)
```

以下为示例代码：

```
<%
Application("strFirst")=("First thing")
Application("strSecond")=("Second thing")
Application.Contents.Remove(strFirst)
Application.Contents.Remove(1)
```

```
%>
```

注意：在运行期间不能从 Application.StaticObjects **集合中删除变量。**

2. RemoveAll **方法**

从 Application.Content **集合中删除所有变量。**

RemoveAll **的语法如下：**

```
Application.Contents.RemoveAll( )
```

3. Lock **方法**

锁定 Application **对象，使得只有当前的** ASP **页面对内容能够进行访问。**

Lock **方法阻止其他客户修改存储在** Application **对象中的变量，以确保在同一时刻仅有一个客户可修改和存取** Application **变量。如果用户没有明确调用** Unlock **方法，则服务器将在** ASP **文件结束或超时后即解除对** Application **对象的锁定。**

Lock **的语法如下：**

```
Application.Lock
```

下面用一个例子来演示 Lock **方法的用法：**

```
<%
'------------------------------------------------------------
'目的：创建一个简单的计数器。
'      演示 Application 的 Lock 方法的用法。
'      将访问的次数和最后访问的时间存放在一个 Application 对象里面。
'------------------------------------------------------------
'初始化计数器
If Application("NumVisits")="" Then
  Application("NumVisits")=0
End If
'锁定 Application
Application.Lock
'将计数器累加并记录最后访问时间
Application("NumVisits")=Application("NumVisits")+1
Application("datLastVisited")=Now( )
'解开 Application
Application.Unlock
Response.Write Application("NumVisits")&"——"_
  &Application("datLastVisited")
%>
```

上面的代码创建了一个简单的网站计数程序。程序先判断存放计数的 NumVisits **有没有创建，如果没有就初始化计数器：**Application("NumVisits")=0 **对象，然后锁定** Application**，将计数器累加，并将最后访问的时间存放在变量** datLastVisited **里。**

4. Unlock 方法

和 Lock 方法相反，Unlock 方法允许其他客户修改 Application 对象的属性。

Unlock 的语法如下：

```
Application.Unlock
```

Application 还提供了两个事件，分别在其启动和结束时触发。

1. Application_OnStart 事件

当 ASP 启动时触发。可以利用该事件初始化变量、创建对象或运行其他代码。

Application_OnStart 事件在首次创建新的会话（即 Session_OnStart 事件）之前发生。当 Web 服务器启动并允许对应用程序所包含的文件进行请求时同样触发 Application_OnStart 事件。Application_OnStart 事件的处理过程必须写在 global.asa 文件之中。

Application_OnStart 的语法如下：

```
<script language=ScriptLanguage RunAt="Server">
Sub Application_OnStart
. . .
End Sub
</script>
```

在 Sub 到 End Sub 中可以写一些程序，比如联接数据库等等，上面的计数器的例子也可以放到这个里面，这样就成了一个真正的计数器。

2. Application_OnEnd 事件

当 ASP 应用程序结束时触发。在最后一个用户会话已经结束并且该会话的 OnEnd 事件中的所有代码已经执行之后发生。其结束时，应用程序中存在的所有变量被清除。

Application_OnEnd 事件的处理过程也必须写在 global.asa 文件之中。

Application_OnEnd 的语法如下：

```
<script language=ScriptLanguage RunAt="Server">
Sub Application_OnEnd
. . .
End Sub
</script>
```

global.asa 文件用于存储有关 IIS 应用程序的如结构初始化信息和已给定应用程序范围的对象，是一个可选文件。用户可以在该文件中指定事件脚本，并声明具有会话和应用程序作用域的对象。该文件的内容不是用来给浏览者显示的，而是用来存储事件信息和由应用程序全局使用的对象。该文件的名称必须是 global.asa，且必须存放在应用程序的根目录中。每个应用程序只能有一个 global.asa 文件。

在 global.asa 文件中，如果包含的脚本没有用<script>标记封装，或定义的对象没有会话或应用程序作用域，则服务器将返回错误。可以用任何支持脚本的语言编写 global.asa

文件中包含的脚本。如果多个事件使用同一种脚本语言，就可以将它们组织在一组<script>标记中。

在 global.asa 文件中声明的过程只能从一个或多个与 Application_OnStart，Application_OnEnd，Session_OnStart 和 Session_OnEnd 事件相关的脚本中调用。在基于 ASP 的应用程序的 ASP 页中，它们是不可用的。如果要在应用程序之间共享过程，可以在单独的文件中声明这些过程，然后使用服务器端包容（SSI）语句将该文件包含在调用该过程的 ASP 程序中。通常，包含文件的扩展名应为.inc。

下面是一个标准的 global.asa 文件。

```
<script language="VBScript" RunAt="Server">
'Session_OnStart  当客户首次运行 ASP 应用程序中的任何一个页面时运行
'Session_OnEnd  当一个客户的会话超时或退出应用程序时运行
'Application_OnStart  当任何客户首次访问该应用程序的首页时运行
'Application_OnEnd  当该站点的 Web 服务器关闭时运行
</script>
```

使用 Application 必须注意以下两点：

（1）不能在 Application 对象中存储 ASP 内置对象。例如，下面的每一行都返回一个错误。

```
<%
Set Application("var1")=Session
Set Application("var2")=Request
Set Application("var3")=Response
Set Application("var4")=Server
Set Application("var5")=Application
Set Application("var6")=ObjectContext
%>
```

（2）若将一个数组存储在 Application 对象中，请不要直接更改存储在数组中的元素。例如，下列的脚本将无法运行。

```
<% Application("StoredArray")(3)="new value" %>
```

这是因为 Application 对象是作为集合被实现的。数组元素 StoredArray(3)未获得新的赋值，而此值包含在 Application 对象中，并将覆盖此位置以前存储的任何信息。如果要对存储在 Application 对象中的数组更新，先要保存该数组的一个副本，然后对副本数组进行操作，完成以后再重新赋值给 Application 对象。这样就可以使所做的任何改动存储下来。

下面以一个例子来演示其用法。

```
<%
'-----------------------------------------------
'目的：  演示如何更新 Application 对象中的数组。
'-----------------------------------------------
```

```
'定义并给数组赋值
dim MyArray()
Redim MyArray(5)
MyArray(0)="hello"
MyArray(1)="some other string"
'将数组存储在 Application 对象中
Application.Lock
Application("StoredArray")=MyArray
Application.Unlock
'将保存在 Application 对象中的数组保存副本
LocalArray=Application("StoredArray")
'更新副本数组
LocalArray(1)="there"
Response.Write LocalArray(0)&LocalArray(1)
'更新 Application 对象
Application.Lock
Application("StoredArray")=LocalArray
Application.Unlock
%>
```

本例中就将 Application 对象间接实现了更新。

Session 是与 Application 对象具有相似作用的另一个非常实用的 ASP 内置对象。存储在 Session 对象中的变量不会清除，而用户在应用程序中访问页面时，这些变量始终存在。当用户请求来自应用程序的 Web 页时，如果该用户还没有会话，则 Web 服务器将自动创建一个 Session 对象。当会话过期或被放弃后，服务器将终止该会话。

通过向客户端程序发送惟一的 Cookie，可以管理服务器上的 Session 对象。当用户第 1 次请求 ASP 应用程序中的某个页面时，ASP 要检查 HTTP 头信息，查看报文中是否有名为 ASPSESSIONID 的 Cookie 发送过来，如果有，则服务器会启动新的会话，并为该会话生成一个全局唯一的值，再把这个值作为新 ASPSESSIONID Cookie 的值发送给客户端，正是通过使用这种 Cookie，可以访问存储在服务器上的属于客户端程序的信息。Session 对象最常见的作用就是存储用户的首选项。例如，如果用户指明不喜欢查看图形，就可以将该信息存储在 Session 对象中。另外 Session 对象还经常被用在鉴别客户身份的程序中。要注意的是，会话状态仅在支持 Cookie 的浏览器中保留，如果客户关闭了 Cookie 选项，Session 也就不能发挥作用了。

19.2.2 Session 对象

可以使用 Session 对象存储特定的用户会话所需的信息。当用户在应用程序的页之间跳转时，存储在 Session 对象中的变量不会清除。而当用户在应用程序中访问页时，这些变量始终存在。也可以显式地结束一个会话和设置空闲会话的超时期限。

Session 对象同样提供了两个集合。

1. Contents 集合

存储于某个特定 Session 对象中的所有变量及其值的一个集合，并且这些变量和值没有使用<object>标记进行定义，包括 Variant 数组和 Variant 类型对象实例的引用。

Contents 的语法如下：

```
Session.Contents(Key)
```

其中参数 *key* 为 Session 的键。

下面的代码列出了 Contents 集合中所有的键。

```
<%@ language="VBScript" %>
<%
Dim sessitem
For Each sessitem In Session.Contents
  If IsObject(Session.Contents(sessitem)) Then
    Response.Write(sessitem&":Session 对象无法显示."&"<br>")
  Else If IsArray(Session.Contents(sessitem)) Then
    Response.Write "Array named"&Session.Content(sessitem)&"<ol>"
    For Each objArray In Session.Contents(sessitem)
      Response.Write "<li>"&_
      Session.Contents(sessitem)(objArray)&"<br>"
    Next
    Response.write "</ol>"
  Else
    Response.write(sessitem&":"&Session.Contents(sessitem)&"<br>")
  End If
  End If
Next
%>
```

以上代码利用 For Each….In…列出集合中的所有元素，其中
和
都是一些用于排版的 HTML 标记。

2. StaticObjects 集合

通过使用<object>标记定义的、存储于某个 Session 对象中的所有变量的一个集合。其语法如下：

```
Session.StaticObjects(Key)
```

下面的代码列出了 StaticObjects 集合中所有的键。

```
<%
Dim objProp
For Each objProp In Session.StaticObjects
  If IsObject(Session.StaticObjects(objProp)) Then
```

```
    Response.Write(objProp&":Session 无法显示。"&"<br>")
  Else
    Response.Write(objprop&":"&Session.StaticObjects(objprop)&"<br>")
  End If
Next
%>
```

以上代码同样利用 For Each….In…列出集合中的所有元素。

Session 对象提供的属性有：

1. CodePage 属性

定义用于在浏览器中显示页内容的代码页（Code Page）。代码页是字符集的数字值，不同的语言和场所可能使用不同的代码页。例如，ANSI 代码页 1252 用于美国英语和大多数欧洲语言，代码页 932 用于日文字。

该属性为可读可写的整型。其语法如下：

```
Session.CodePage(=Codepage)
```

2. LCID 属性

定义发送给浏览器的页面地区标识（LCID）。LCID 是惟一地标识地区的一个国际标准的缩写，例如，2057 定义当前地区的货币符号是 £。LCID 也可用于 FormatCurrency 等函数中，只要其中有一个可选的 LCID 参数。LCID 也可在 ASP 处理指令<%…%>中设置，并优先于会话的 LCID 属性中的设置。

该属性为可读可写的整型。其语法如下：

```
Session.LCID(=LCID)
```

例如：

```
<%
Session.LCID=2057
Dim curNumb
curNumb=FormatCurrency(125)
Response.Write(curNumb)
%>
```

3. SessionID 属性

用于返回某个会话的会话标识符，创建会话时，该标识符由服务器产生。SessionID 只在父 Application 对象的生存期内是惟一的，因此当一个新的应用程序启动时可以重新使用。在很多情况下 SessionID 可以用于 Web 页面注册统计。

该属性为只读的长整型。其语法如下：

```
Session.SessionID
```

4. Timeout 属性

用于为会话定义以分钟为单位的超时周期。如果用户在超时周期内没有进行刷新或请求一个网页，则该会话结束。在各网页中根据需要可以修改该属性。默认值是 10 分钟。在访问率高的站点上该时间应更短。

该属性为可读可写的整型。其语法如下：

```
Session.Timeout[=nMinutes]
```

Session 提供了以下 3 个方法来对 Session 对象进行操作：

1. Abandon 方法

删除所有存储在 Session 对象中的对象并释放这些对象的源。如果未明确地调用 Abandon 方法，一旦会话超时，服务器将删除这些对象。

使用 Abandon 方法，当网页的执行完成时，结束当前用户会话并撤消当前 Session 对象。但即使在调用该方法以后，仍可访问该页中的当前会话的变量。当用户请求下一个页面时，将启动一个新的会话，并建立一个新的 Session 对象（如果存在的话）。

Abandon 的语法如下：

```
Session.Abandon
```

2. Remove 方法

从 Session.Content 集合中删除指定变量。其语法如下：

```
Session.Contents.Remove(Item|Index)
```

例如：

```
<%
Session("myName")=" "
Session.Collection.Remove("myName")
%>
```

3. RemoveAll 方法

从 Session.Content 集合中删除指定变量。其语法如下：

```
Session.Contents.RemoveAll
```

注意：在运行期间不能从 Session. StaticObjects 集合中删除变量。

和 Application 相同，Session 对象也提供了两个事件。

1. Session_OnStart 事件

在服务器创建新会话时发生。服务器在执行请求的页之前先处理该脚本。该事件是设置会话期变量的最佳时机，因为在访问任何页之前都会先设置它们。

尽管在 Session_OnStart 事件包含 Redirect 或 End 方法调用的情况下 Session 对象仍会保持，然而服务器将停止处理 global.asa 文件并触发 Session_OnStart 事件的文件中的脚本。

为了确保用户在打开某个特定的 Web 页时始终启动一个会话，就可以在 Session_OnStart 事件中调用 Redirect 方法。当用户进入应用程序时，服务器将为用户创建一个会话并处理 Session_OnStart 事件脚本。可以将脚本包含在该事件中以便检查用户打开的页是不是启动页，如果不是，就指示用户调用 Response.Redirect 方法启动网页。程序如下:

```
<script RunAT="Server" language="VBScript">
Sub Session_OnStart
  startPage="/MyApp/StartHere.asp"
  currentPage=Request.ServerVariables("SCRIPT_NAME")
  If strcomp(currentPage,startPage,1) Then
    Response.Redirect(startPage)
  End If
End Sub
</script>
```

注意：上述程序只能在支持 Cookie 的浏览器中运行。因为不支持 Cookie 的浏览器不能返回 SessionID cookie，所以，每当用户请求 Web 页时，服务器都会创建一个新会话。这样，对于每个请求，服务器都将处理 Session_OnStart 事件脚本并将用户重定向到启动页中。

2. Session_OnEnd 事件

在会话被放弃或超时时发生。其形式如下：

```
<script language=ScriptLanguage RunAT="Server">
Sub Session_OnEnd
. . .
End Sub
</script>
```

关于使用 Session 对象需要注意的事项和 Application 对象相似，读者可以参考前面关于 Application 的说明。

会话可以通过以下 3 种方式启动：

- 新用户请求的 URL 标识应用程序中的.asp 文件，并且该应用程序的 global.asa 文件包含 Session_OnStart 事件过程。
- 用户保留有 Session 对象中的值。
- 只要服务器收到的请求中不包含有效的 SessionID cookie，就将自动启动新会话。
- 用户请求应用程序中的.asp 文件，并且应用程序的 global.asa 文件使用<object>标记在会话间实例化对象。

如果用户在指定时间内没有请求或刷新应用程序中的任何页，会话将自动结束。会话过期的默认值是 20 分钟。

可以通过在 IIS 中设置“应用程序选项”属性页中的“会话超时”属性改变应用程序

的默认超时限制设置，如图 19.4 所示。

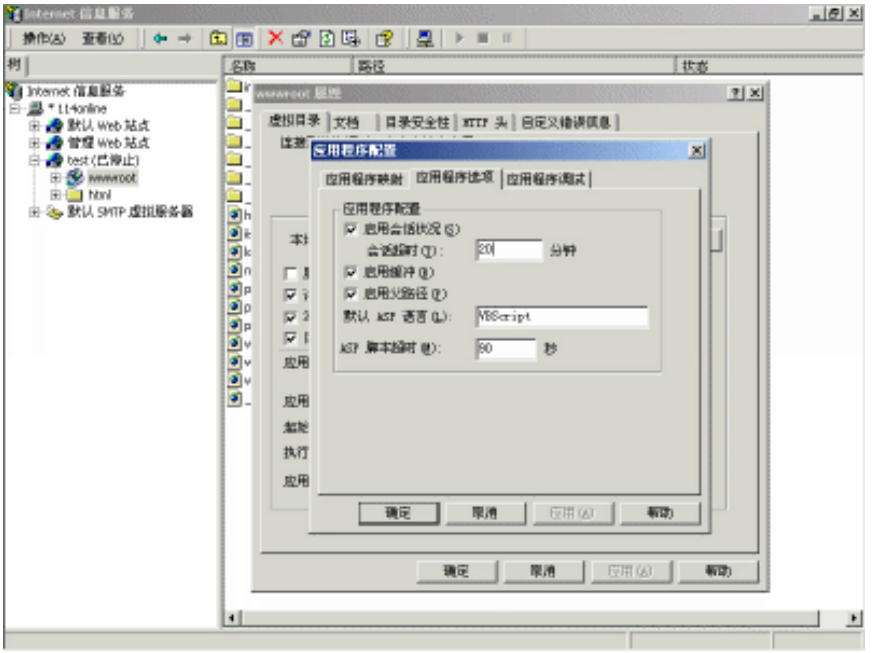

图 19.4　设置应用程序会话过期时间的窗口

应依据 Web 应用程序的要求和服务器的内存空间来设置此值。例如，如果希望浏览 Web 应用程序的用户在每一页仅停留几分钟，就应该缩短会话的默认超时值。过长的会话超时值将导致打开的会话过多而耗尽服务器的内存资源。对于一个特定的会话，如果想设置一个小于默认超时值的超时值，可以设置 Session 对象的 Timeout 属性。例如，下面这行脚本将超时值设置为 5 分钟。

```
<% Session.Timeout=5 %>
```

当然也可以设置一个大于默认设置的超时值，Session.对象的 Timeout 属性决定超时值。还可以通过 Session 对象的 Abandon 方法显式结束一个会话。例如，在表格中提供一个“退出”按钮，将按钮的 Action 参数设置为包含下列命令的.asp 文件的 URL。

```
<% Session.Abandon %>
```

19.3　Server 对象

Server 对象是 ASP 专门用来处理服务器端特定任务的内置对象，主要处理与服务器的环境和处理活动有关的任务。利用 Server 对象提供的方法可以以服务器特定的方法格式化数据、管理其他网页的执行、管理外部对象和组件的执行以及处理错误。

19.3.1 Server 对象的属性

Server 对象的主要属性为 ScriptTimeout，该属性设置或返回页面的脚本在服务器退出执行和报告一个错误之前可以执行的时间（秒数）。达到该值后将自动停止页面的执行，并从内存中删除包含可能进入死循环的错误的页面或者是那些长时间等待其他资源的网页。这会防止服务器因存在错误的页面而过载。对于运行时间较长的页面需要增大这个值。

可以在程序中直接指定 ScriptTimeout 的值，例如：

```
<%Server.ScriptTimeout=100 %>
```

也可以在 IIS 的控制面板中设置整个应用程序的脚本超时时间，如图 19.4 所示（和设置会话过期时间在同一窗口）

19.3.2 Server 对象的方法

Server 对象提供的方法有：

1．CreateObject 方法

CreateObject 的语法如下：

```
Server.CreateObject(ProgID)
```

该方法创建由参数 *ProgID* 标识的对象（一个组件、应用程序或脚本对象）的一个实例，返回可以在代码中使用的一个引用。可以用于一个虚拟应用程序创建会话层或应用程序层范围内的对象。该对象可以用其 ClassID 来标识，如{clsid: BD96C556-65A3…37A9}或一个 ProgID 串来标识，如“ ADODB.Connection ”。

默认情况下，由 Server.CreateObject 创建的对象具有页作用域。这就是说，在当前 ASP 页处理完成之后，服务器将自动破坏这些对象。如果要创建有会话或应用程序作用域的对象，可以使用<object>标记并设置 Session 或 Application 的 scope 属性，也可以在对话及应用程序变量中存储该对象。例如：

```
<% Set Session("ad")=Server.CreateObject("MSWC.AdRotator")%>
```

注意：不能创建与内置对象同名的对象实例。例如，下列脚本将返回错误。

```
<% Set Response=Server.CreateObject("Response") %>
```

2. HTMLEncode 方法

HTMLEncode 的语法如下：

```
HTMLEncode(string)
```

HTMLEncode 对特定的字符串进行 HTML 编码。HTMLEncode 方法返回一个字符串，该串是输入值 *string* 的拷贝，但去掉了所有非法的 HTML 字符，如<、>、&和双引号，并转换为等价的 HTML 条目，即<，>，&和"等。

这个方法在需要输出 HTML 源代码时十分有用，比如要在页面上显示
标记，如果

直接用 Response.Write "
"，那么该标记在页面中会被自动解释成一个软回车符，看不到字符。这时使用 Response.Write Server.HTMLEncode ("
")，就会在页面上显示
标记。而且有时直接输出 HTML 源代码会由于 HTML 代码被浏览器解释了，从而把页面搞乱，达不到预期效果，这时候也可以使用 Server.HTMLEncode。

3. URLEncode 方法

URLEncode 的语法如下：

```
URLEncode(string)
```

URLEncode 根据 URL 规则对字符串进行正确编码。当字符串数据以 URL 的形式传递到服务器时，在字符串中不允许出现空格，也不允许出现特殊字符。URLEncode 方法返回一个字符串，该串是输入值 *string* 的拷贝，但是在 URL 中无效的所有字符，如?、&和空格，都转换为等价的 URL 条目，即%3F、%26 和+。

4. MapPath 方法

MapPath 的语法如下：

```
MapPath(URL)
```

MapPath 返回在参数 *URL* 中指定的文件或资源的完整物理路径和文件名。

URL 指定要映射物理目录的相对或虚拟路径。若 *URL* 以一个正斜杠（/）或反斜杠（\）开始，则 MapPath 方法返回路径时将 *URL* 视为完整的虚拟路径。若 *URL* 不是以斜杠开始，则 MapPath 方法返回同.asp 文件中已有的路径的相对路径。

假如 C:\Inetpub\wwwroot 目录被设置为服务器的宿主目录。下列示例使用服务器变量 PATH_INFO 映射当前文件的物理路径。

```
<%= Server.Mappath(Request.ServerVariables("/")) %>
```

其输出为：

```
c:\inetpub\wwwroot
```

由于下列示例中的路径参数不是以斜杠开始的，所以它们被相对映射到当前目录，此处是目录 C:\Inetpub\wwwroot\asp。

```
<%= Server.Mappath("data.txt") %>
<%= Server.Mappath("asp/data.txt") %>
```

其输出为：

```
c:\inetpub\wwwroot\asp\data.txt
c:\inetpub\wwwroot\asp\asp\data.txt
```

注意：MapPath 方法不检查返回的路径是否正确或在服务器上是否存在。

19.4 小 结

本章学习了 ASP 的几个内置对象，可以说 ASP 程序是建立在这些对象的基础上的，基本上，以后的每个 ASP 程序都会用到这其中的一个或几个对象。ASP 还有一个内置对象 ObjectContext，可以使用它提交或撤消由 ASP 脚本初始化的事务。由于 ObjectContext 对象实际中使用比较少，因此这里就没有作详细介绍，如果读者有兴趣，可以参考相关书籍。

下一章将学习激动人心的 ActiveX 组件。

第 20 章　ActiveX 组件

在前面的有关章节中已经提到过 ActiveX，本章再结合 ASP 来深入讲解 ActiveX 组件的有关知识。

所谓 ActiveX 是指允许开发人员创建 Web 交互式内容的 Microsoft 技术的综合术语。ActiveX 是一套独立于语言的相互作用技术，允许采用各种语言编写的软件组件在网络环境中一起工作。ActiveX 的核心技术是 COM（组件对象模型）及 DCOM（分布式组件对象模型）。这些技术已经授权给 The Open Group 标准组织，正在多种平台上实现。

而 ActiveX 组件是指合并了 ActiveX 技术的可重用的软件组件。这些组件可用来向 Web 页、桌面应用程序以及软件开发工具添加特殊功能，如动画或弹出式菜单。ActiveX 控件可以采用各种编程语言编写，包括 C、C++、Visual Basic 和 Java 等。

ASP 可以通过访问 ActiveX 组件实现强大的功能，本章将学习 ActiveX 组件在 ASP 中的使用方法、详细介绍一些 ASP 可安装组件和第三方组件的功能和用法，最后还要学习利用 VB 来创建自己的 ActiveX 组件。关于一些特殊的 ActiveX 组件，例如 ADO 和 FileSystemObject，由于其十分重要，后面将用专门的章节来介绍。

20.1　ActiveX 组件的创建和使用方法

ActiveX 组件功能十分强大，而它们在 ASP 中的使用却十分简便。在 ASP 中应用 ActiveX 组件的方法主要有两种：

- 使用 HTML 中的<object>标记创建对象。
- 使用 ASP 中的 Server.CreateObject 方法来创建对象实例。

下面以广告组件 Ad Rotator 的创建为例来说明。

使用第 1 种方法的 HTML 代码示例如下：

```
<object RunAT="Server" ID=MyAds ProgID="MSWC.AdRotator"></object>
```

或者

```
<object RunAT="Server" ID=MyAds ClassID="Clsid:1621F7C0-60AC-11CF
  -9427-444553540000"></object>
```

而采用第 2 种方法创建组件的代码应该为：

```
<% Set MyAds=Server.CreateObject("MSWC.AdRotator") %>
```

提示：第 1 种方法是用 HTML 代码直接指定组件运行在服务器端（RunAT="Server"），

指定组件的注册名（ProgID）或注册号码（ClassID），同时为将在脚本语言中使用的变量名提供 ID 属性组(ID=MyAds)。第 2 种方法是利用 ASP 的内置对象 Server 来创建对象，将组件以一个变量命名（MyAds），以便 ASP 引用，关于 Server 对象请参考本书第 19 章。

而组件的使用和它在 VB 等其他环境中一样，也是直接使用组件的方法和属性。当将组件以变量命名后，就可以直接使用

变量名.方法|属性

调用组件。具体请参考下一节。

20.2 ASP 中常用的 ActiveX 组件

ActiveX 组件可以完成很强大的功能，而 ASP 中已经自带了一些组件以完成一些强大的功能。当安装 ASP 运行环境后，就可以使用这些组件了，比如广告组件 Ad Rotator、文件系统访问组件 File Access 和数据库访问组件 Database Access 等。

表 20.1 列出来了 ASP 中可安装的常用组件。

表 20.1 ASP 中可安装的常用组件

组件	功能描述
Ad Rotator	创建一个 Ad Rotator 对象。该对象可按指定计划在同一页上自动轮换显示广告
Browser Capabilities	创建一个 BrowserType 对象。该对象决定访问 Web 站点的每个浏览器的性能、类型及版本
Database Access	提供用 ADO 对数据库的访问
Content Linking	创建一个 NextLink 对象。该对象可生成 Web 页内容列表，并像书一样将各页顺续连接
File Access	提供文件的输入输出访问
Collaboration Data Objects for NTS	可以快速、简便地在 Web 页上添加收发邮件功能。该组件只适用于 Internet Information Server for Windows NT® Server
MyInfo	创建一个 MyInfo 对象。该对象追踪个人信息，例如站点管理员的姓名、地址及显示选择
Counters	创建一个 Counters 对象。该对象可以创建、保存、增加或检索任意数量的独立计数器
Content Rotator	自动翻转 Web 页中的 HTML 内容字符串
Page Counter	记录并显示 Web 页被打开的次数

下面将介绍几个常用组件的用法。

20.2.1　Ad Rotator 组件

Ad Rotator 组件可以使网页按指定计划在同一页上自动轮换显示广告。
首先要创建 Ad Rotator 组件，如下所示。具体方法已经介绍，这里就不再重复。

```
<% Set MyAds=Server.CreateObject("MSWC.AdRotator") %>
```

组件方法 GetAdvertisement（广告计划文件）用来获得将要显示在 Web 页上的广告，示例如下：

```
<% =MyXAds.GetAdvertisement("/ads/adrot.txt") %>
```

注意：指定广告计划文件只能指定它的虚拟路径，而不能直接指定物理路径，比如 adrot.txt 在 C:\Inetpub\wwwroot\Ads\Adrot.txt，而网站发布主目录是“/”，即 C:\Inetpub\wwwroot，那么只能指定路径为虚拟路径，即/ads/adrot.txt，而不能用物理路径，即 C:\Inetpub\wwwroot\Ads\Adrot.txt。

利用 GetAdvertisement 可以产生 HTML 代码输出。
adrot.txt 就是广告计划文件，它包括与要显示的图像文件的地点有关的信息以及每个图像的不同属性。其语法如下：

```
[REDIRECT URL]
[WIDTH numWidth]
[HEIGHT numHeight]
[BORDER numBorder]
*
adURL
adHomePageURL
Text
Impressions
```

其中：

- *URL*：广告所联接的地址，单击广告后到达该地址。该地址可以是完整路径，如 Http://sina.com.cn/adredir.asp，也可以是网站的虚拟路径，如/script/adredir.asp。
- *numWidth*：广告图片框的宽度，单位是像素，默认为 440 像素。
- *numHeight*：广告图片框的宽度，单位是像素，默认为 60 像素。
- *numBorder*：以像素为单位指定广告四周超链接的边框宽度。如果指定为 0 则表示没有边框，默认是 1 个像素宽度的边框。
- *：这是将要进行具体广告设计的分隔符，*前面的部分是对整个广告的图片大小、重定向文件的位置、广告边框等的设定，而*后面部分是每个具体广告的设定，包括每个广告图片的地址、主页地址、权重等等。
- *adURL*：广告图片的地址。
- *adHomePageURL*：广告主页地址，如果该广告没有主页地址的话，就用一条横线

代替。

- *Text*：当把鼠标移到广告图片，或当浏览器不支持该广告图片格式，或者浏览器的功能被关闭的时候出现的动态提示文字。
- *Impressions*：表示广告的权重，是从 0 到 10000 的数字。数字越大则表示该广告出现的几率越大。例如 3 个广告的权重分别为 20、30 和 50，则在此网页被访问 100 次内，第 1 个广告将出现 20 次，第 2 个广告将出现 30 次，第 3 个广告则出现 50 次，也就是说它们出现的几率分别为 20/(20+30+50)=20%、30/(20+30+50)）=30% 和 50/(20+30+50)=50%。

下面是一份具体的广告计划文件：

```
REDIRECT /adr/adredir.asp
WIDTH 440
HEIGHT 60
BORDER 1
*
/adr/images/sina.gif
http://www.sina.com.cn/
这是新浪网的图标！
10
/adr/images/sohu.gif
-
这是搜狐的图标！
20
/adr/images/yahoo.gif
http://www.yahoo.com.cn/
这是 yahoo 的中文站点图标！
30
/adr/images/114online.gif
http://www.114online.com/
这是 114 在线的图标!!!
40
```

将上面的计划文件保存为 adr.txt，并放在 C:\Inetpub\wwwroot\Adr\下（这里默认主页发布主目录为 C:\Inetpub\wwwroot）。

建立一个 adrot.asp 文件，其内容为：

```
<% Set MyAds=Server.CreateObject("MSWC.AdRotator") %>
<% =MyAds.GetAdvertisement("/adr/adr.txt") %>
```

将文件保存在 C:\Inetpub\wwwroot\Adr\下。

再建立一个重定向广告文件 adredir.asp，其内容为：

```
<% Response.Redirect(Request.Querystring("url")) %>
```

用浏览器查看 adrot.asp，可以看到如图 20.1 所示的效果。

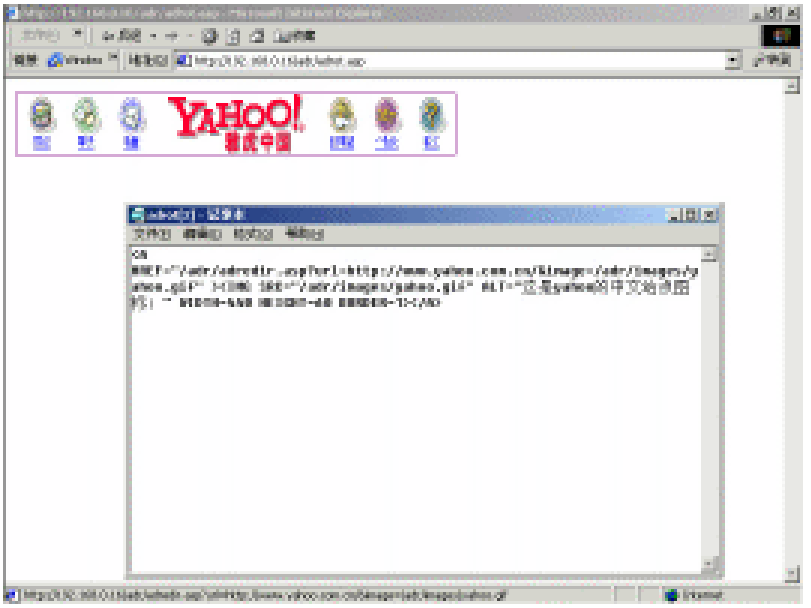

图 20.1　使用 Ad Rotator 组件的页面效果

提示：图 20. 1 中的记事本是用来查看 HTML 源代码的，在页面上应该是没有的，这里是为了更好地让读者看到 Ad Rotator 组件动态生成了 HTML 代码输出。

在图 20.1 中可以看到，Ad Rotator 组件动态生成了以下 HTML 代码并输出以显示广告图片和生成相关的连接。

```
<a
href="/adr/adredir.asp?url=http://www.yahoo.com.cn/&image=/adr/images/
  yahoo.gIf" ><img src="/adr/images/yahoo.gIf" alt="这是 yahoo 的中文站点
  图标！" width=440 height=60 border=1>
</a>
```

可以很清楚地看出，Ad Rotator 组件是以动态生成并输出 HTML 代码来实现广告的发布管理的。

同时如果刷新浏览器 10 次，可以很明显地看出，图片 sina.gif 出现 1 次，sohu.gif 出现了两次，yahoo.gif 出现了 3 次，114online.gif 出现了 4 次。

使用 AdRotator 组件可以直接通过对象属性而不是计划文件中的设置来直接控制某些广告特性，其可用属性如下：

- Border：指定广告边框的大小。
- Clickable：指定广告是否为超链接。
- TargetFrame：指定显示广告的框架的名称。

例如：

```
<%
Set ad=Server.CreateObject("MSWC.AdRotator")
ad.Border=0
ad.Clickable=true
ad.TargetFrame=AdFrame
ad.GetAdvertisement("/ads/adrot.txt")
%>
```

这样，借助 Ad Rotator 组件，就可以很方便地建立一个自己的广告管理系统。还可以在重定向文件中加入一个计数器，比如在 adredir.asp 的第 1 行加入

```
<% Counters.Increment(request.querystring("url")) %>
```

注意：这里用到了 Counters 组件，因此要先创建这个组件对象，关于这个组件的详细用法请参考 20.2.3 小节。

这样就可以得到一份网站广告的统计报告，就可以来管理发布广告和了解网站浏览者的兴趣了，而这些都是很有商业价值的信息。

20.2.2 Browser Capabilities 组件

Browser Capabilities 组件创建一个 BrowserType（浏览器类型）对象用以向脚本描述客户端浏览器的能力。一旦一个浏览器连接到 Web 服务器，浏览器会自动将一串用户代理 HTTP 报头（User Agent HTTP Header）传送到服务器。该报头为一个 ASCII 码字符串，用以识别该浏览器及其版本号。BrowserType 对象将该报头与 browsercap.ini 文件中的条目进行比较。如果找到匹配的条目，BrowserType 对象假设该浏览器具备 browsercap.ini（一个用以描述浏览器属性的文本文件，该文件必须和 Brwosercap.dll 文件处于同一目录下）文件中所描述的属性。如果对象没有在 browsercap.ini 文件中找到与报头匹配的条目，BrowserType 对象假设该浏览器具备默认浏览器的属性。如果没有匹配条目并且 browsercap.ini 文件没有定义默认浏览器的各项设置，则 BrowserType 对象将该浏览器的各项属性值赋为 Unkown。通过升级 browsercap.ini 文件，可以方便地把浏览器属性或者对新浏览器的描述加入到这个组件（稍候介绍如何升级 browsercap.ini 文件）。

创建 BrowserType 对象的方法如下例所示：

```
<% Set MyBrowserTypeObj=Server.CreateObject("MSWC.BrowserType") %>
```

以下例子给出如何用 Browser Capabilities 组件取得当前浏览器的一些属性。

```
<html>
<% Set bc=Server.CreateObject("MSWC.BrowserType") %>
<table width="75%" border="1">
<tr>
  <td>Browser</td>
```

```
  <td><% Response.Write bc.browser %></td>
</tr>
<tr>
  <td>Version</td>
  <td><% Response.Write bc.version %></td>
</tr>
<tr>
  <td>Frames</td>
  <td>
  <%
  If (bc.frames=TRUE) Then
    Response.Write "TRUE"
  Else
    Response.Write "FALSE"
  End If
  %>
  </td>
</tr>
<tr>
  <td height="19">Tables</td>
  <td height="19">
  <%
  If (bc.tables=TRUE) Then
    Response.Write "TRUE"
  Else
    Response.Write "FALSE"
  End If
  %>
  </td>
</tr>
<tr>
  <td>BackgroundSounds</td>
  <td>
  <%
  If (bc.BackgroundSounds=TRUE) Then
    Response.Write "TRUE"
  Else
    Response.Write "FALSE"
  End If
  %>
  </td>
</tr>
<tr>
```

```
  <td>VBScript</td>
  <td>
  <%
  If (bc.vbscript=TRUE) Then
    Response.Write "TRUE"
  Else
    Response.Write "FALSE"
  End If
  %>
  </td>
</tr>
<tr>
  <td>JScript</td>
  <td>
  <%
  If (bc.javascript=TRUE) Then
    Response.Write "TRUE"
  Else
    Response.Write "FALSE"
  End If
  %>
  </td>
</tr>
</table>
</html>
```

运行上面的程序，可以看到如图 20.2 所示的效果。

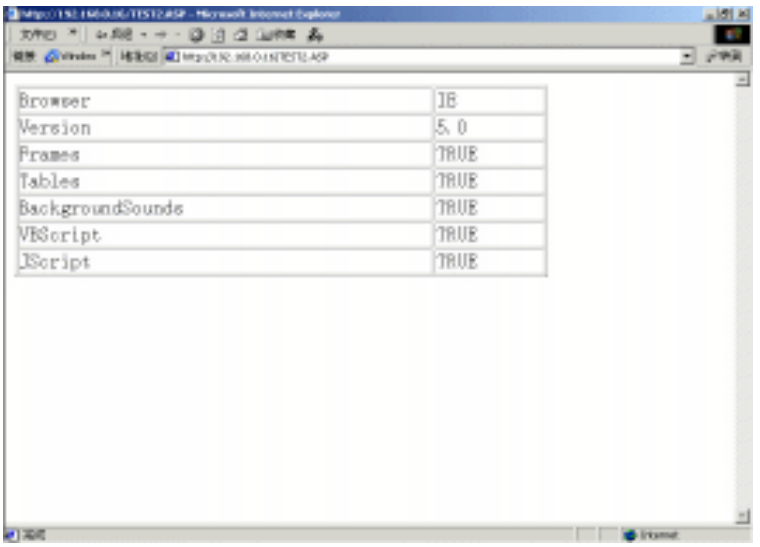

图 20.2 当前浏览器的一些属性列表

下面将介绍 browscap.ini 文件以及如何升级 browscap.ini 文件。

可以在 browscap.ini 文件中描述任意多个浏览器的属性。也可以设置一套默认属性，当浏览器发送的报头无法和 browscap.ini 中的任何条目匹配时， BrowerType 对象会假设该浏览器具备默认的属性。

每个浏览器定义由一个 HTTP 报头、一些属性名以及相关属性值构成。关于各种浏览器的 HTTP 报头的定义可参考[link]http://www.w3.org/[/link]网站中的内容。

browscap.ini 文件的语法如下：

```
[;comments]
[HTTPUserAgentHeader]
[parent=browserDefinition]
[property1=value1]
...
[propertyN=valueN]
[Default Browser Capability Settings]
[defaultProperty1=defaultValue1]
...
[defaultPropertyN=defaultValueN]
```

其中：

- *comments*：注释，以；开头的任意多行。
- *HTTPUserAgentHeader*：HTTP 用户代理报头，类似于格式[Mozilla/2.0 (compatible; MSIE 3.0;* Windows 95)]，其中可以使用通配符*。
- *browserDefinition*：用以定义父类的 HTTP 报头。
- *propertyN*：浏览器的属性名，可从属性列表中选取所需的属性。
- *valueN*：浏览器的属性值。
- *defaultPropertyN*：默认浏览器的属性名，可选属性同 *propertyN*。
- *defaultValueN*：默认浏览器的属性值。

以下为两个 browscap.ini 文件的例子。

```
;;ie 4.0
[IE 4.0]
browser=IE
Version=4.0
frames=TRUE
tables=TRUE
cookies=TRUE
backgroundsounds=TRUE
vbscript=TRUE
javascript=TRUE
javaapplets=True
```

```
ActiveXControls=TRUE
beta=False

;;ie 4.01
[Mozilla/2.0 (compatible; MSIE 4.01*; Windows 95)]
parent=IE 4.0
version=4.01
minorver=01
platform=Win95
; Default Browser
[Default Browser Capability Settings]
browser=Default
frames=FALSE
tables=TRUE
cookies=FALSE
backgroundsounds=FALSE
vbscript=FALSE
javascript=FALSE
```

作为参考，以下给出浏览器的属性列表。

- ActiveXControls：**指定浏览器是否支持** ActiveX **控件。**
- Backgroundsounds：**指定浏览器是否支持背景音乐。**
- Beta：**指定浏览器是否为测试版。**
- Browser：**指定浏览器的名字。**
- Cdf：**指定浏览器是否支持** Web **发布的频道解释定义（**Channel Definition Format**）。**
- Cookies：**指定浏览器是否支持** Cookies**。**
- Frames：**指定浏览器是否支持帧显示。**
- Javaapplets：**指定浏览器是否支持** Java applets**。**
- Javascript：**指定浏览器是否支持** Java Script**。**
- Platform：**指定浏览器运行所需的操作系统。**
- Tables：**指定浏览器是否支持表格。**
- Vbscript：**指定浏览器是否支持** VBScript**。**
- Version：**指定浏览器的版本。**

注意：以上解释内容为“ 指定浏览器是否支持 ”开头的属性，其属性值应为 True/False**，其余为字符串。**

20.2.3 Counters 组件

在介绍 Ad Rotator **组件时提到了** Counters **组件，这个组件用来创建、存储、递增和检索每个计数器的值。为一个网站做一个计数器是十分有必要的，因为** Web **的流量是考察一个网站最直观的数值，而** Counters **组件能用于支持任何种类数据的统计。**

Counters 组件通过自己所带的方法创建一个计数器，并且将所有计数器的值存放在一个名为 counters.txt 的文本文件中，该文本文件存在于 counters.dll 所在的目录。

该组件具有以下方法：

- Get(*counter_name*)：获得计数器 *counter_name* 的当前值，如果该计数器先前没有创建，则会自动创建并置初始值为 0 并返回 0。
- Remove(*counter_name*)：删除计数器 *counter_name*。
- Set(*counter_name,value*)：把指定计数器的值设置成参数 *value* 所提供的整数值，如果此计数器先前没有创建，则先创建并设定为指定值。
- Increment(*counter_name*)：增加指定计数器的当前值，如果此计数器先前没有创建，则首先创建并设置为 1。

使用 Counters 组件首先要创建一个组件对象，这里同样可以采用如下所示的两种方法：

```
<object ID="objCounters" RunAT="Server" scope="Application" ProgID=
  "MSWC.Counters">
```

或者

```
<%Set objCounters=Server.Createobject("MSWC.Counters")%>
```

使用 Counters 组件的方法很简单，下面举一个网站调查的例子进行说明。

先创建一个要调查的页面，然后来统计页面中各个选项分别被选中的数值。保存下面的代码为文件 count.html——要调查的页面。

```
<html>
<title>一个小的调查</title>
<form name="form1" method="post" action="count.asp">
<p align="center">调查：您认为 <b>ASP</b> 有用吗？</p>
<p align="center">
<input type="radio" name="radiobutton" value="radiobutton">
  很有用<br>
</p>
<p align="center">
<input type="radio" name="radiobutton" value="radiobutton">
  还行</p>
<p align="center">
<input type="radio" name="radiobutton" value="radiobutton">
  一点用处都没有</p>
<p align="center">
<input type="submit" name="Submit" value="投票">
</p>
</form>
</html>
```

调查提交后交给 count.asp 来处理，该文件应该完成根据用户的选择来分别给不同的计数器累加，然后再取得计数器的结果显示出来，这里就要用到 Get 方法和 Increment 方法。

将下面的代码保存到有 ASP 运行权限的目录。

```
<%
'------------------------------------------------------------------------
'目的：处理用户提交的选项，在相应的计数器上累加，以得到调查的结果中并显示调查结果。
'------------------------------------------------------------------------
'创建 Counters 组件对象
Set objCounters=Server.Createobject("MSWC.Counters")
'根据用户的选择分别计数
Select Case Request.Form("radiobutton")
  '利用组件的 Increment 方法计数
  Case 1
    objCounters.Increment("选第 1 个选项的人数")
  Case 2
    objCounters.Increment("选第 2 个选项的人数")
  Case 3
    objCounters.Increment("选第 3 个选项的人数")
End Select
'利用 Get 方法显示调查结果
Response.Write "<font style=font-size:20pt;font-family:宋体>
  认为 ASP 很有用的有：<font color=red>"
Response.Write objCounters.Get("选第 1 个选项的人数")
Response.Write "</font>个人<br>"
Response.Write "认为 ASP 还行的有：
  <font color=red style=font-size:20pt;font-family:宋体> "
Response.Write objCounters.Get("选第 2 个选项的人数")
Response.Write "</font>个人<br>"
Response.Write "认为 ASP 没有一点用的有：<font color="RED"
  style=font-size:20pt;font-family:宋体>"
Response.Write objCounters.Get("选第 3 个选项的人数")
Response.Write "</font>个人</font><br>"
%>
```

在 count.html 上选择并提交后可以看到如图 20.3 所示的效果。

通过图 20.3 可以很清楚地看出用户选择的结果，这样利用 Counters 组件就可以很轻松地创建属于自己的网站小调查了。当然读者还可以对上面的程序做一些完善，比如规定一个用户只能投一次票等等。

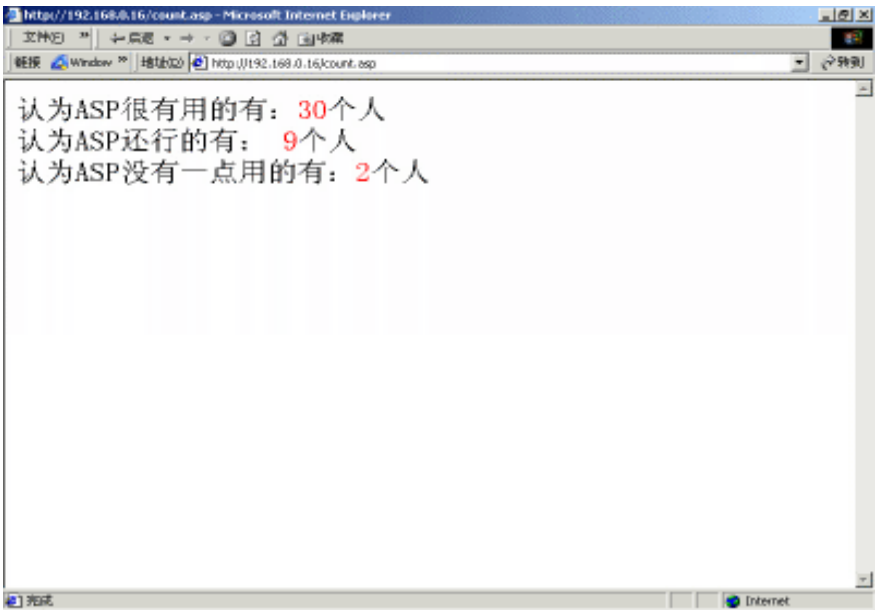

图 20.3　调查显示结果

20.2.4　CDONTS 组件

CDONTS 是 CDO（Collaboration Data Object，数据协作对象）的一个子集。

此组件只对 Windows NT 和 Windows 2000 系列有用，对 Windows 9X 无效。利用 CDONTS 组件可以实现不借助其他第三方组件方便地发送 Email，甚至还可以结合 Microsoft Exchange Server 实现一个 Email Web Server（邮件 Web 服务器）。

使用 CDONTS 组件必须配置好其运行环境。

首先必须安装了 CDO For NTS。如果默认安装 Window NT Server 或者 Windows 2000 的话就已经安装了 CDONTS，如果没有的话可以到 Microsoft 的官方站点免费下载（Http://www.mirosoft.com）。

其次要安装 SMTP Service。首先确认有没有安装 SMTP Service，这可以从 IIS 的管理面板中看到。打开 IIS 控制面板，如果安装了 SMTP Service 就可以看到一个像信封样的图标。如果没有安装 SMTP Service，可以运行 Windows 2000（或者 Windows NT）的安装程序添加 SMTP Service 组件。

安装配置 SMTP Service 组件的具体操作如下：

（1）进入操作系统的控制面版，选择“添加/删除程序”。出现对话框后单击“添加/删除 Windows 组件”按钮，再选择组件就会出现如图 20.4 所示的界面。

（2）在对话框中选择“Internet 信息服务（IIS）”，然后单击“详细信息”按钮，将出现如图 20.5 所示的窗口。

（3）在对话框中选中 SMTP Service，单击“确定”按钮，再根据提示一步一步地安装即可。在其中会要求指定 MailRoot 目录的位置。

（4）安装完成后就可以在 IIS 的控制菜单中看到 SMTP Service 的图标，如果没有启动

的话，可以在 SMTP Service **图标上单击右健然后在出现的菜单中执行“ 启动 ”命令。**

图 20.4 Windows **组件安装向导的界面**

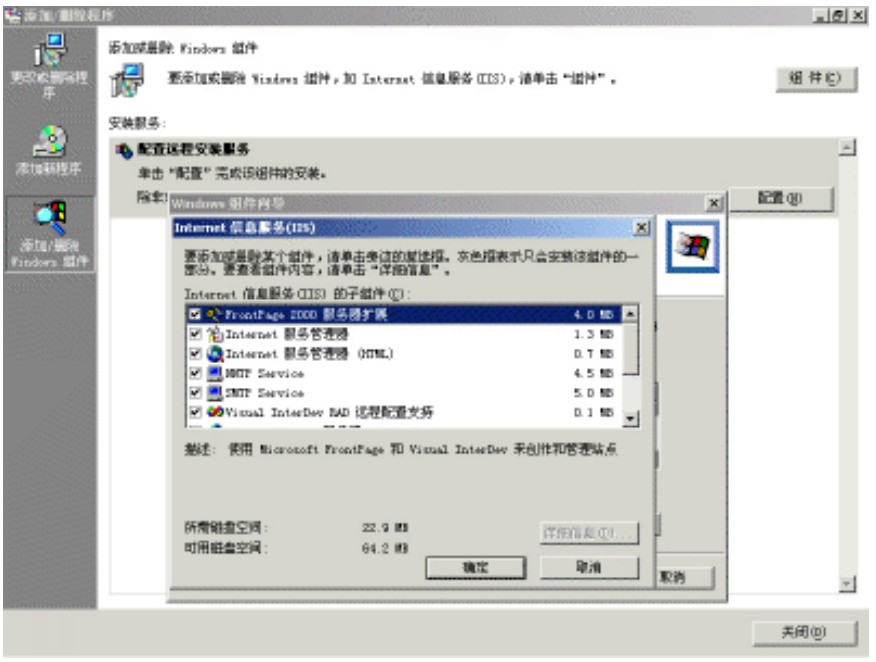

图 20.5 **安装** IIS **的各项组件窗口**

（5**）打开** IIS**，展开默认** SMTP **虚拟服务器，用右健单击“ 域 ”，在出现的菜单中执行**

“新建”命令，再选择“域”，然后按照提示新建一个域。

（6）右健单击“默认 SMTP 虚拟服务器”，在出现的菜单中执行“属性”命令，就会出现设置 SMTP 虚拟服务器的界面，如图 20.6 所示。

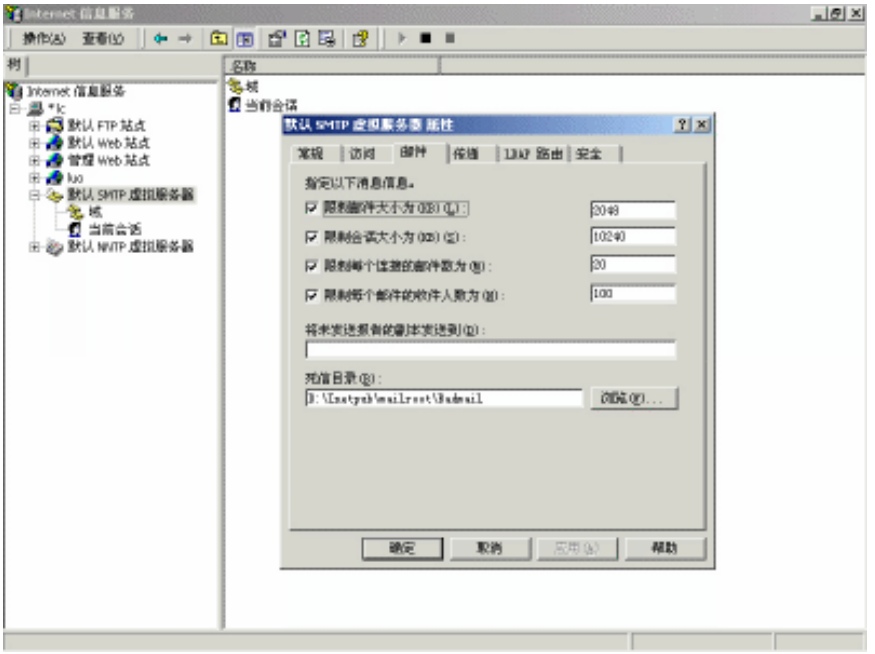

图 20.6　SMTP 虚拟服务器设置窗口

（7）在图 20.6 所示的界面中设置 SMTP 虚拟服务器的名称、邮件大小限制、死信的存放目录等。

到此，就成功配置了 CDONTS 的运行环境。

注意：上面的关于 SMTP Service 的设置都是在 Windows 2000 中进行的，在 Windows NT 中的设置和 Windows 2000 相似，读者可以参照以上介绍的步骤进行设置。

表 20.2 列出了 CDONTS 组件的方法和属性。

表 20.2　CDONTS 组件的方法和属性

对象	支持版本	属性	方法
AddressEntry	verion1.2	Address，Application，Class，Name，Parent，Session，Type	无
Attachment	version1.2	Application，Class，ContentBase，ContentID，ContentLocation，Name，Parent，Session，Source，Type	Delete，WriteToFile，ReadFromFile
Attachments collection	version1.2	Application，Class，Count，Item，Parent，Session	Add，Delete

（续表）

对象	支持版本	属性	方法
Folder	version1.2	Application，Class，Messages，Name，Parent，Session	无
Message	version1.2	Application，Attachments，Class，ContentBase，ContentID，HTMLText，Importance，MessageFormat，Parent，Recipients，Sender，Session，Size，Subject，Text，TimeReceived，TimeSent，ContentLocation	Delete，Send
Messages collection	version1.2	Application，Class，Count，Item，Parent，Session	Add，Delete，GetPrevious GetFirst，GetLast，GetNext,
NewMail	version1.2	Bcc，Body，BodyFormat，Cc，ContentBase，ContentLocation，From，Importance，MailFormat，Subject，To，Value，Version	AttachFile，AttachURL，Send，SetLocaleIDs
Recipient	version1.2	Address，Application，Class，Name，Parent，Session，Type	Delete
Recipients collection	version1.2	Application，Class，Count，Item，Parent，Session	Add，Delete
Session	version1.2	Application，Class，Inbox, MessageFormat, Name, Outbox, Parent, Session, Version	GetDefaultFolder，LogonSMTP，Logoff，SetLocaleIDs

这里将介绍最常用的 NewMail **对象。**

NewMail **对象允许在** Web **页上添加一个简单的邮件发送功能。**

首先要创建 NewMail **对象，创建方法很简单，如下例所示：**

```
<%
Dim objNewMail
set objNewMail=Server.CreateObject("CDONTS.NewMail")
%>
```

NewMail **对象的属性有：**

- From**：指明发信人。其语法为：**

```
objMail.From=strMan
```

- To**：指明收信人。其语法为：**

```
objMail.To=strMan
```

- Body：Email 的内容。其语法为：

```
objMail.Body=strBody
```

- Subject：Email 的主题。其语法为：

```
objMail.Subject=strSubject
```

- Bcc 和 Cc：Cc 是指 Email 发送同时抄送给的地址，Bcc 是指 Email 发送同时密件抄送给的地址。Bcc 行上的任何收件人都得不到其他收件人的信息，而其他人也得不到 Bcc 行上收件人的信息，这样可以达到保护收件人信息的效果。To、Bcc、Cc 可以一次指定多个用户地址，中间用分号隔开就可以了。例如：

```
<%
objMail.To="aa@sina.com";"bb@yeah.net";"cc@china.com"
%>
```

- Importance：表示邮件的等级。一共有 0（低）、1（正常）和 2（高）等 3 个等级。其语法为：

```
objMail.Importance=intNum
```

  由于不是所有的邮件服务器都支持级别加速，所以邮件等级设置可能不会对邮件发送的速度起任何作用。

- BodyFormat：指定 Email 的格式。其语法为：

```
objMail.BodyFormat=conValue
```

  其中 *conValue* 可能的取值为：

  - CdoBodyFormatHTML：以 HTML 格式发送邮件。
  - CdoBodyFormatTEXT：以文本格式发送邮件。

- MailFormat：指定邮件格式是普通文本还是 MIME 格式。如果为 MIME 格式，则指定 MailFormat 为 CdoBodyFormatMIME，如果为普通文本则指定 MailFormat 为 CdoBodyFormatTEXT。
- ContentBase 和 ContentLocation：这两个属性指定了包含在消息中的 URL 属性。ContentBase 对包含在消息中的 URL 指定一个基本路径，ContentLocation 消息主体内的所有 URL 作为相对路径来解释。
- Version：返回 CDONTS 组件的版本。

NewMail 对象的方法有：

- AttachFile：给消息添加附件。
- Send：发送信件。

下面将介绍如何创建邮件发送程序，让读者可以更好地了解 CDONTS 组件的各项属性和方法的用法。

首先要建立一个发送邮件的界面，代码如下：

```
<html>
<title>发送邮件的表单</title>
<div align="center">
<h1>发送电子邮件界面</h1>
<form name="form1" method="post" action="sendmail.asp">
<table width="75%" border="1">
<tr>
  <td width="16%">收件人：</td>
  <td width="84%">
  <input type="text" name="ToAddress">
  </td>
</tr>
<tr>
  <td width="16%">抄送：</td>
  <td width="84%">
  <input type="text" name="ccAddress">
  </td>
</tr>
<tr>
  <td width="16%">密件抄送：</td>
  <td width="84%">
  <input type="text" name="BccAddress">
  </td>
</tr>
<tr>
  <td width="16%">发件人：</td>
  <td width="84%">
  <input type="text" name="FromAddress">
  </td>
</tr>
<tr>
  <td width="16%">邮件等级：</td>
  <td width="84%">
  <select name="intImport">
  <option value="0">低</option>
  <option value="1">正常</option>
```

```
  <option value="2">高</option>
  </select>
  </td>
</tr>
<tr>
  <td width="16%">邮件格式：</td>
  <td width="84%">
  <select name="conBodyFormat">
  <option value="CdoBodyFormatTEXT">普通文本</option>
  <option value="CdoBodyFormatHTML">HTML 格式</option>
  </select>
  <select name="conEmailFormat">
  <option value="CdoBodyFormatTEXT">普通文本</option>
  <option value="CdoBodyFormatMIME">MIME 格式</option>
  </select>
  </td>
</tr>
<tr>
  <td width="16%">邮件主题：</td>
  <td width="84%">
  <input type="text" name="Subject">
  </td>
</tr>
<tr>
  <td width="16%">邮件内容：</td>
  <td rowspan="2">
  <textarea name="Body"></textarea>
  <input type="submit" name="Submit" value="发送">
  </td>
</tr>
<tr>
  <td width="16%"> </td>
</tr>
</table>
</form>
<p> </p>
</div>
</html>
```

保存上面代码为一个.html **文件，用** Web **浏览器打开看到如图** 20.7 **所示的界面。**

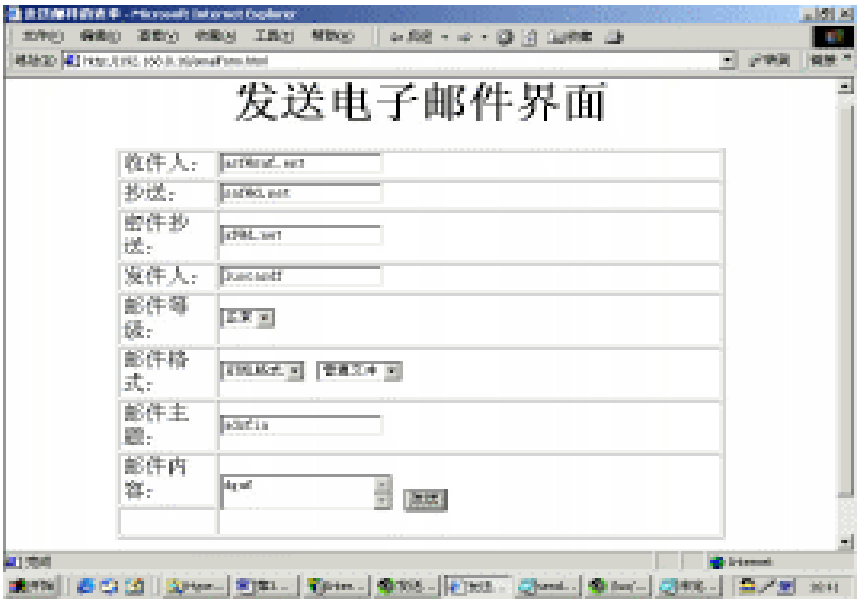

图 20.7 发送电子邮件的表单界面

在上面表单中填入要发送 Email 的详细信息，包括发件人、收件人、主题等，把这些信息发送到一个 ASP 文件中处理。将下面的代码保存为 sendmail.asp 并和上面的.html 文件存放在同一个目录下。

```
<%
Dim Body,FromAddress,ToAddress,Subject,BccAddress,CcAddress
Body=Request.Form("Body")
FromAddress=Request.Form("FromAddress")
ToAddress=Request.Form("ToAddress")
Subject=Request.Form("Subject")
BccAddress=Request.Form("BccAddress")
CcAddress=Request.Form("CcAddress")
dim objCDOMail
Set objCDOMail=Server.CreateObject("CDONTS.NewMail")
objCDOMail.From=FromAddress  '收信人的 Email
objCDOMail.To=ToAddress  '收信人的 Email
objCDOMail.Cc=CcAddress  '抄送人的地址
objCDOMail.Bcc=BccAddress  '密件抄送人的地址
objCDOMail.Subject=Subject  '邮件主题
objCDOMail.MailFormat=Request.Form("conMailFormat")
objCDOMail.BodyFormat=Request.Form("conBodyFormat")
objCDOMail.Body=Body
objCDOMail.Send
Set objCDOMail=Nothing
```

```
Response.Write "邮件成功发送！"
%>
```

这个程序创建了一个 NewMail 对象，然后给它的各项属性赋值，再利用 Send 方法将信件发送到指定的信箱。

这样，利用 CDONTS 的一个对象，建立了一个简单但十分实用的 Email 发送程序。但这只是组件的一个很简单的应用，如果结合 Microsoft Exchange Server 甚至可以实现自己创建一个 Email Web Server。这个限于篇幅就不在这里介绍了。

20.3　第三方组件

这里的第三方组件是指由其他人开发的实现某些特定用途的 ASP 组件，很多情况下可以直接使用第 3 方组件来实现一些功能。Internet 上有很多免费的 ASP 组件，比如文件上传组件、发送 Email 的组件等。

这里将介绍应用比较广泛的 Jmail 组件，使用该组件可以在 ASP 程序里面发送电子邮件，而不需要使用一个诸如 Eudora/exchange/outlook 之类的邮件客户端。Jmail 组件发送邮件速度快，功能强大，而且可以免费下载。

下载地址：http://download.dimac.net/jmail/jmail.exe

使用环境：

Web Server （IIS 4/PWS 4）

Microsoft Windows NT or Microsoft Windows 95 with Winsock 2.0

Jmail 版本 3.60 具有以下特性：

- Attachments：附件。
- Return Receipt：发信回执。
- Detailed Logging Capabilities：详细日志能力。
- Priority Settings：优先级设定。
- MIME with BASE64, UUEncode and Quoted-Printable Encoding：邮件编码设置。
- Queued mailings without separate NT service：邮件队列发送，无需单独的 NT 服务。
- Unlimited Redundant Servers：没有限制的多余 Server（SMTP 服务器）
- Blind Carbon Copy（BCC）：密送。
- Carbon Copy（CC）：抄送。
- Urgent flagging：紧急标志。
- US ASCII and ISO-8859-1
- UUEncoding
- X-Headers and Custom Headers
- MS Mail and Exchange Priority Headers.
- Internal MX Lookup with MX Priority and Redundancy

Jmail 的日志功能允许查看发送邮件时候发生的事情，如果 Mail Server 错误设置的话，

这是一种发现问题所在的好方法。

JMail 已经通过了下列 Y2K 测试：

2000-01-01 Bug - OK
2000-02-29 Bug - OK
2000-04-01 Bug - OK（Found in MS VCRT）

Jmail 使用 16 位精度表示年，所以日期范围限制在-32767~32768。
创建一个 Jmail 组件的方法如下例所示：

```
<% Set JMail=Server.CreateObject("JMail.SMTPMail") %>
```

Jmail 组件版本 3.0 具有以下属性：

- Body：String，信件体，正文，使用 AppendText 追加内容。
  如：JMail.Body="Hello world"。
- Charset：String，默认为"US-ASCII"。
  如：JMail.Charset="US-ASCII"。
- ContentTransferEncoding：String，指定内容传送时的编码方式，默认是"Quoted-Printable"。
  如：JMail.ContentTransferEncoding="base64"。
- ContentType：String，信件的类型，缺省是"text/plain"，但是可以设置为其他想要的类型。如果以 HTML 格式发送邮件，改为"text/html"即可。
  如：JMail.ContentType="text/html"。
- DeferredDelivery：Date，设置延期发送。如果邮件服务器支持的话，消息到了这个时间才会发送。
- Encoding：String，这个属性可以用来改变附件编码方式（默认是"base64"），可以选择使用的是"base64"、"uuencode"或者"quoted-printable"。
  如：JMail.Encoding="base64"。
- ErrorCode：Integer，如果 JMail.Silent 设置为 true，则 ErrorCode 包含的是错误代码。
  如：Response.Write(JMail.ErrorCode)。
- ErrorMessage：String，如果 JMail.Silent 设置为 true，包含的是错误信息。
  如：Response.Write(JMail.ErrorMessage)。
- ErrorSource：String，如果 JMail.Silent 设置为 true，ErrorCode 包含的是错误源。
  如：Response.Write(JMail.ErrorSource)。
- ISOEncodeHeaders：Boolean，是否将信头编码成 ISO-8859-1 字符集，默认是 true。
  如：JMail.ISOEncodeHeaders=false。
- Lazysend：Boolean，指定 Jmail 是否一直等到 Email 发送了然后返回，或者缓冲这条消息然后在后台发送。然而如果设置了这个属性，就不能控制错误信息。

注意：如果选择了这个选项，属性 ServerAddress 将无效，Lazysend 将通过 dsn

查询决定邮件服务器，在一些设置里面，这可能有问题。如：JMail.LazySend=true。

- Log：String，如果 Logging 属性设置为 true，表示 Jmail 创建的日志。
  如：Response.Write(JMail.Log)。
- Logging：Boolean，是否使用日志。
  如：JMail.Logging=true。
- MimeVersion：String，指定 MIME，默认是"1.0"。
  如：JMail.MimeVersion="1.0"。
- Priority：Integer，优先级，范围为 1~5，其中：
  1：高优先级，有些邮件程序称之为紧急。
  2：也是高优先级。
  3：普通优先级。
  4：低优先级。
  5：最低的优先级。

  如：JMail.Priority=3。
- Recipients：String，只读属性，返回所有收件人。
  如：Response.Write(""+JMail.Recipients+"")。
- ReplyTo：String，指定一个可选的回信地址。
  如：JMail.ReplyTo= "president@dimac.net"。
- ReturnReceipt：Boolean，指定是否发件人需要一个回复收据，默认是 false。
  如：JMail.ReturnReceipt=true。
- Sender：String，指定发件人的邮件地址。
  如：JMail.Sender="batman@dimac.net"。
- SenderName：String，指定发件人的姓名。
  如：JMail.SenderName="Bat man"。
- ServerAddress：String，指定邮件服务器的地址。可以指定多个服务器，用分号隔开。可以指定端口号。如果 ServerAddress 保持空白，Jmail 会尝试连接远程邮件服务器，然后直接发送到服务器上去。
  如：JMail.ServerAddress="mail.mydom.net; mail2.mydom.net:2500"。
- Silent：Boolean，如果设置为 true，Jmail 不会出现例外错误。JMail.execute( )会根据操作结果返回 true 或 false。
  如：JMail.Silent=true。
- SimpleLayout：Boolean，如果设置为 true，将减少 Jmail 产生的 header 信头。
  如：JMail.SimpleLayout=true。
- Subject：String，设定消息的标题。
  如：JMail.Subject="Dimac rocks big time!"。

Jmail 组件版本 3.0 具有以下方法：

- AddAttachment(*FileName*,[*ContentType*])：（v3.0）添加自定义附件（文件）。
  如：JMail.AddAttachment("C:\\autoexec.bat")。
- AddCustomAttachment(*FileName,Data*)：添加自定义附件。
  如：JMail.AddCustomAttachment("readme.txt","Contents of file")。
- AddHeader(*XHeader*,*Value*)：添加用户定义的 X-header 到 Message。
  如：JMail.AddHeader("Originating-IP","193.15.14.623")。
- AddNativeHeader(*Header*,*Value*)：添加信头。
  如：JMail.AddNativeHeader("MTA-Settings","route")。
- AddRecipient(*Email_Address*)：增加收件人。
  如：JMail.AddRecipient("info@dimac.net")。
- AddRecipientBCC(*Email_Address*)：增加密件收件人。
  如：JMail.AddRecipientBCC("someone@somedomain.net")。
- AddRecipientCC(*Email_Address*)：增加抄送收件人。
  如：JMail.AddRecipientCC("someone@somedomain.net")。
- AddRecipientEx(*Email_Address,Name*)：增加一个带名字的收件人。
  如：JMail.AddRecipientEx("info@dimac.net","Dimac INFO")。
- AddURLAttachment(*bstrURL*,*bstrAttachAs*,[*bstrAuth*])。下载并添加一个来自 *bstrURL* 的附件，其中第 2 个参数 *AttachAs* 用来指定信件收到后的文件名，第 3 个可选参数用来进行 WWW-鉴定。
  如：JMail.AddURLAttachment("http://download.sina.com/jmail.exe","jmail.exe")。
- AppendBodyFromFile(*FileName*)：将一个文件内容追加到正文后。
  如：JMail.AppendBodyFromFile("C:\mytext.txt")。
- AppendText(*Text*)：追加信件的正文内容。
  如：JMail.AppendText("Text appended to message Body")。
- ClearAttachments( )：清除附件列表。
- ClearCustomHeaders( )：清除所有自定义的信头。
- ClearRecipients( )：清除收件人列表。
- Close( )：强制 Jmail 关闭缓冲的与邮件服务器的连接。
- Execute( )：Boolean，执行邮件的发送。
- ExtractEmailAddressesFromURL(*bstrURL*,[*bstrAuth*]) ：从一个 URL 下载和添加 Email 地址。
  如：JMail.ExtractEmailAddressesFromURL("http://duplo.org/List.asp")。
- GetMessageBodyFromURL(*bstrURL*,[*bstrAuth*]) ：清除 Message 的正文，并用 *bstrURL* 的内容替换。ContenType 会自动设置成 *bstrURL* 的 ContenType。第 2 个参数（login and password）是可选的。
  如：JMail.GetMessageBodyFromURL("http://duplo.org/","login:password")。
- LogCustomMessage(*Message*)：将用户自定义消息加入 Jmail 日志。只有属性 Logging 设置为 true 时，这项功能才能起作用。
  如：JMail.LogCustomMessage("Hello world")。

以下为使用 Jmail 组件的例子。

```
<html>
<head>
<title>Confirmation </title>
<body>
<%
Set JMail=Server.CreateObject("JMail.SMTPMail")
'This is my local SMTP server
JMail.ServerAddress="mail.yourdomain.com:25"
'This is me....
JMail.Sender="myemail@mydomain.net"
JMail.Subject="Here you go..."
'Get the recipients mailbox from a form (note the lack
'of a equal sign).
JMail.AddRecipient "mum@any.com"
JMail.AddRecipient "dad@some.com"
'The body property is both read and write.
'If you want to append text to the body you can
'use JMail.Body = JMail.Body & "Hello world!"
'or you can use JMail.AppendText "Hello World!"
'which in many cases is easier to use.
JMail.Body="Here you go. Your request has been approved"&_
"and the program is attached to this message"
'1 - highest priority (Urgent)
'3 - normal
'5 - lowest
JMail.Priority=1
JMail.AddHeader "Originating-IP",
  Request.ServerVariables("REMOTE_ADDR")
'Must make sure that IUSR_???? has access to the following files.
JMail.AppendBodyFromFile "e:\mail\standard_footer.txt"
JMail.AddAttachment "e:\products\MyProduct.exe"
'Send it...
JMail.Execute
%>
<center>
An Email has been sent to your mailbox(<% =request.form("email") %>).
</center>
</body>
</html>
```

注意：以上内容参考 Jmail 的站点，读者可以从 http://www.dimac.net 站点获得关于 Jmail 组件的详细信息。

其他比较出名的第三方组件还有 ASPMail 组件、SendMail 组件等等，这里就不一一介绍了。使用别人的组件虽然省事，但是很多时候还需要自己动手来做组件，满足自己特殊的需要。下一节将学习如何创建自己的组件。

20.4 用 VB 创建 ASP 组件

关于使用 ASP 组件的优越性，读者应该不会有什么异议了。尽管 ASP 提供了一些基本的组件，而且还有一些免费的第三方组件，但这些都是照顾到了许多网站应用的共同之处，如果自己网站有某些特殊的需要话，就要自己来动手做组件了利用组件，可以使网站层次有一个本质的提高。下面将介绍使用 VB 来创建 ASP 组件，注意这里说的组件都是服务器端的组件。

> 提示：服务器端的组件有别于客户端的组件，客户端的组件是通过网络传输，依靠 HTML 来起作用，而且只在 IE 中有用。但是服务器端的组件是运行在服务器端，它在服务器上执行各种操作。因此，所有的浏览器都能享用，它依靠的是服务器而不是浏览器。

对系统和软件的要求：

- 满足 ASP 运行环境的系统配置：IIS 3.0 以上+32 位 Windows 操作系统。
- Microsoft Visual Basic 5.0 或以上版本。

本节所示程序调试环境为：Windows 2000 Server 中文版+IIS 5.0+Microsoft Visual Basic 6.0 中文版。

操作步骤：

（1）启动 VB，在出现的“新建工程”对话框中选择 ActiveX DLL 图标，单击“打开”按钮，进入 VB 编辑程序界面。

（2）VB 会提供一个默认的工程名（project1）和类名（class1）。为了方便使用这里要将这两个名字都改掉。在 VB 右边的对象属性窗口中将工程名和属性名分别改为 Myproject 和 Myclass。

在 VB 中使用 ASP 必须引用一个类库（Microsoft Active Server Pages Object Library），首先确认拥有 Microsoft Active Server Pages Object Library。从 VB 操作菜单中选择“工程”，然后在其中选择“引用”，此时会出现引用的一个操作对话框，在对话框中选中 Microsoft Active Server Pages Object Library，如图 20.8 所示。

为了在 VB 类中使用 ASP 的方法，必须先在类中声明这些方法，可以在类中写一个 OnStartPage 子函数。例如：

```
Public Sub OnStartPage(PassedScriptingContext As ScriptingContext)
  Set MyScriptingContext=PassedScriptingContext
End Sub
```

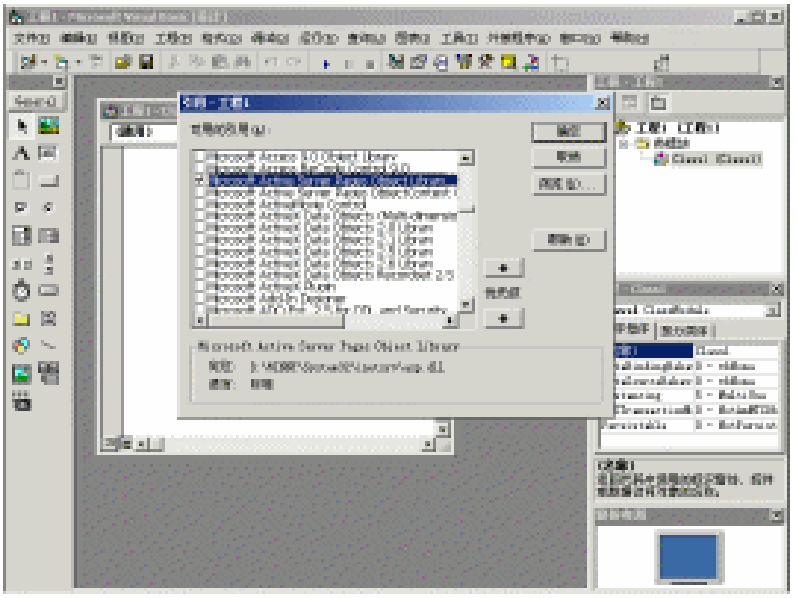

图 20.8　VB 的引用对象窗口

现在，无论什么时候用户访问一个带有本组件的 ASP 文件，IIS 就会把 ScriptingContext 传送给我们的对象供我们使用。这个 ScriptingContext 包括了全部的 ASP 方法和属性。实现上，这使得我们有能力访问所有 ASP 的对象。例如：

```
Public Sub OnStartPage(PassedScriptingContext As ScriptingContext)
  Set MyScriptingContext=PassedScriptingContext
  Set MyApplication=MyScriptingContext.Application
  Set MyRequest=MyScriptingContext.Request
  Set MyResponse=MyScriptingContext.Response
  Set MyServer=MyScriptingContext.Server
  Set MySession=MyScriptingContext.Session
End Sub
```

以后就能用在 VB 中用 MyApplication 来代替 ASP 中的 Application，同理可以代替 Request，Server…，不过还是要在 OnStartPage 之前来申明这些变量。例如：

```
Private MyScriptingContext As ScriptingContext
Private MyApplication As Application
Private MyRequest As Request
Private MyResponse As Response
Private MyServer As Server
Private MySession As Session
```

现在变量就能像标准的 ASP 对象那样使用了。比如，经常在 ASP 中用 Request.Form 来收集提交表单的数据。

在 ASP 中的实现：

```
<%
Dim strMyName
strMyName=Request.Form("userName")
Response.Write("My name is "&MyName)
%>
```

在 VB 中的实现：

```
Dim strMyName As String
strMyName=MyRequest.Form("userName")
MyResponse.Write("My name is "&MyName)
```

通过使用 MyResponse 来代替 Response，能够使用所有 Response 的方法。另一件需要注意的事是，必须在建立的类中写上 OnEndPage 子函数，这与 OnStartPage 是相反的。OnStartPage 是创建对象，而 OnEndPage 是消毁对象。

```
Public Sub OnEndPage( )
  Set MyScriptingContext=Nothing
  Set MyApplication=Nothing
  Set MyRequest=Nothing
  Set MyResponse=Nothing
  Set MyServer=Nothing
  Set MySession=Nothing
End Sub
```

注意：使用 Dim MyResponse As Response 其实对于写过组件的人来说并不是很好，因为如果要将该程序拿到 ASP 程序中进行调试，尤其是将已经调试通过的 ASP 程序做成组件时，ASP 程序里用的是标准的组件名。默认的组件名为 Response，Request 等，引文中在前面加上了 My，则如果程序中用到了这句，则也要进行相应的改变，一则这会带来较大的工作量，二则对于程序的可读性并没有什么改进。因此，最好的方法是如下定义：

```
Dim Context As ObjectContext
Dim Server As Server
Dim Request As Request
Dim Session As Session
Dim Response As Response
```

当然，后面还要有初始化：

```
Private Sub Class_Initialize( )
  Set Context=GetObjectContext( )
  Set Server=Context("Server")
  Set Request=Context("Request")
```

```
  Set Response=Context("Response")
  Set Session=Context("Session")
End Sub
```

这样声明的话就无需再更改对象名了。

现在有了自己的工程（MyProject）和类名（MyClass），以后就可以在 ASP 代码中使用它们的名字来引用这个组件。如下所示：

```
<% Set ObjReference=Server.CreateObject("ProjectName.ClassName") %>
```

对于上面建立的工程的引用就是：

```
<% Set ObjReference=Server.CreateObject("MyProject.MyClass")%>
```

引用了工程后就可以直接使用组件中创建的方法和属性。比如用 VB 在类中创建一个函数，它的作用是向 Web 中输出 1 行字符“ hello word ”。

例如，在 VB 中添加一个函数，代码如下：

```
Public Sub SayHello( )
  MyResponse.Write("Hello World")
End Sub
```

注意：上面声明的代码都是必需的，上面提到的操作也要做。

现在一个小型的组件编写完成，剩下的工作就是编译这个组件，在“ 工程 ”菜单中保存它，在出现的对话框中为组件取名，然后就在菜单中执行 make myProject.dll 命令，将其编译成 DLL 文件。一个组件就真正完成了。

注意：编译组件时必须先停掉 IIS 或者先把 PWS 关掉，关闭 IIS 可以在命令行的模式下，输入 net stop iisadmin/y，然后再编译此组件。否则 VB 就会提示哪些组件在使用中。编译完成后再启动 IIS 或者启动 PWS，启动 IIS 的方法为在命令行的模式下，输入 net start w3svc。

如果成功编译，先必须注册该组件，具体方法为进入命令行的模式，进入组件存放的目录，输入命令：regsvr32 *projectName*，注销该组件的方法为：regsvr32 *projectName* –u。

对于上面的组件就是：

```
regsvr32 myProject.dll
```

编译并注册成功后，就可以在 ASP 中使用这个组件，具体方法为：

```
<%
Set ObjReference=Server.CreateObject("myProject.myClass")
ObjReference.SayHello
%>
```

（上面代码必须在.asp 文件中）。

运行后即可看到结果。在 Web 页面显示出“Hello World”字样。

上面已经介绍了创建一个组件的步骤，下面就给一个 ASP 组件的例子给读者参考。这是一个通过 Web 创建 Microsoft Exchange Server 邮箱的组件，此组件有一个类模块和一个模块。

模块中源码如下：

```
Public strName As String
Public strServer As String
Public strDomain As String
Public strSite As String
Public strOrg As String
Public strUsername As String
Public strPassword As String
Public Sub Main( )
  '-------------------------------------------------------
  'SD 安全对象
  '要求注册 ADSSECURITY.DLL)
  '-------------------------------------------------------
  Set sid=CreateObject("ADsSID")
  Set sec=CreateObject("ADsSecurity")
  '-------------------------------------------------------
  '重置 ADSI Interfaces
  '-------------------------------------------------------
  Dim sd As IADsSecurityDescriptor
  Dim dacl As IADsAccessControlList
  Dim ace As New AccessControlEntry
  '-------------------------------------------------------
  '如果没有在工程中加入 ADSI2.5 安全类库
  '则必须定义下面的常量
  '-------------------------------------------------------
  Const ADS_SID_HEXSTRING=1
  Const ADS_SID_WINNT_PATH=5
  Const ADS_RIGHT_EXCH_MODIFY_USER_ATT=&H2
  Const ADS_RIGHT_EXCH_MAIL_SEND_AS=&H8
  Const ADS_RIGHT_EXCH_MAIL_RECEIVE_AS=&H10
  '-------------------------------------------------------
  '-----创建 Email 邮箱-------------------
  '-------------------------------------------------------
  '----Server,Organd Site information
  server=strServer
  Org=strOrg
  Site=strSite
  domain=strDomain
```

```
UserName=strUsername
password=strPassword
'设定邮箱参数
strDisplayName=strName
strFirstName=strName
strLastName=strName
strAlias=strName
strMTA="cn=Microsoft MTA,cn=" & server & ",cn=Servers,
 cn=Configuration,ou="&Site&",o="&Org
strMDB="cn=Microsoft Private MDB,cn="&server&",
 cn=Servers,cn=Configuration,ou="&Site&",o="&Org
strSMTPAddr=strName&"@"&server&".com"
'为每一个邮箱创建一个 NT 用户
Set dom=GetObject("WinNT://"&domain)
Set usr=dom.create("user",UserName)
usr.SetInfo
usr.SetPassword password
'设定 ADS 地址
ADsPath="LDAP://"+server
ADsPath=ADsPath+"/cn=Recipients,OU="
ADsPath=ADsPath+Site
ADsPath=ADsPath+",O="
ADsPath=ADsPath+Org
Set objCont=GetObject(ADsPath)
'创建新邮箱
Set mailBox=objCont.Create("organizationalPerson","cn="&strAlias)
mailBox.Put "mailPreferenceOption",0
mailBox.Put "givenName",strFirstName
mailBox.Put "sn",strLastName
mailBox.Put "cn",strDisplayName
mailBox.Put "uid",strAlias
mailBox.Put "Home-MTA",strMTA
mailBox.Put "Home-MDB",strMDB
mailBox.Put "mail",strSMTPAddr
mailBox.Put "MAPI-Recipient",True
mailBox.Put "rfc822Mailbox",strSMTPAddr
'-----------------------------------------------------'
'绑定邮箱和 NT 用户
'(要求注册 ADSSECURITY.DLL)
'-----------------------------------------------------'
sid.SetAs ADS_SID_WINNT_PATH,"WinNT://"&domain&"/"&UserName&",user"
sidHex=sid.GetAs(ADS_SID_HEXSTRING)
mailBox.Put "Assoc-NT-Account",sidHex
```

```
  '将属性缓存存放到服务器
  mailBox.SetInfo
  '-------------------------------------------------------'
  '设定邮箱安全
  '允许用户更改自己的属性和发送、接收 Email
  '-------------------------------------------------------'
  Set sd=sec.GetSecurityDescriptor(mailBox.ADsPath)
  Set dacl=sd.DiscretionaryAcl
  ace.Trustee=domain&"\"&strAlias
  ace.AccessMask=ADS_RIGHT_EXCH_MODIFY_USER_ATT Or
    ADS_RIGHT_EXCH_MAIL_SEND_AS Or ADS_RIGHT_EXCH_MAIL_RECEIVE_AS
  ace.AceType=ADS_ACETYPE_ACCESS_ALLOWED
  dacl.AddAce ace
  sd.DiscretionaryAcl=dacl
  sec.SetSecurityDescriptor sd
End Sub
```

注意：上面的组件的运行环境为 Windows 2000+Microsoft Exchange Server 5.5，需要 ADSI 2.0 以上支持(可以从 http://www.microsoft.com 下载)。

类模块的作用为调用模块和传递参数，其源码如下：

```
Public Sub crea(strname2 As String,strserver2 As String,strdomain2
  As String,strsite2 As String,strorg2 As String,strusername2 As
  String,strpassword2 As String)
  strName=strname2
  strServer=strserver2
  strDomain=strdomain2
  strSite=strsite2
  strOrg=strorg2
  strUsername=strusername2
  strPassword=strpassword2
  Main
End Sub
```

注意：上面的代码仅供参考，请勿用于商业用途！！

20.5 构建健壮的服务器端组件

在服务器上安装了 IIS，就可以发挥 ASP 的优势了，ASP 利用 ActiveX 组件来为网络应用完成所有种类的工作。尽管可以在 HTML 和有 ASP 页面的 IIS 里面使用许多 ActiveX 组件，但服务器端组件也不是运行在一台服务器上的普通组件，它在运行时不会提示与需要特别关照的产品服务器有关的任何信息。用户无法做任何事情去改变其对服务器性能、

安全和稳定性的影响。对服务器端组件的不恰当选择可能会导致一些问题，包括速度的明显下降、安全漏洞或者其他更恶劣的问题。

客户端组件在用户计算机上执行。客户端组件包括绝大多数流行的组件，如标签、文本框、命令按钮等。这些组件可以通过<object>标记和（或）HTML 对象语法来包含在客户端 HTML 代码中。

多数的有用的客户端组件会提供特定种类的用户界面。记住，使用客户端组件就意味着真实组件已经被传到客户计算机上。通常的做法是把组件下载到客户计算机上。当然，用户不得不等待下载过程，而且客户计算机必须被配置为允许下载。

与此形成对比的是，服务器端组件在服务器计算机上执行。服务器端组件也为用户做一些工作，但却是在服务器上运行的。必须认识到这个差异并且相应地编制代码。服务器端组件为整个应用程序封装了一些逻辑或功能。

当一个用户使用应用程序时，他将不会真正看到服务器端的组件。这些组件大多数都可以通过需要使用组件的 ASP 脚本中的<object>标记来包含。同样可以通过服务器端脚本的 CreateObject 语法来包含服务器端组件。

这里向大家推荐 7 个关键步骤，可以帮助创建稳定和安全的服务器端组件，可以很优雅地缩放并且维持性能。在创建一个服务器端网络应用时，需要把稳定、安全和性能放在首要位置。

（1）服务器端组件不应该具有 GUI（图形用户界面）。

因为服务器端组件是在服务器上运行的，网络应用的用户是看不到可能弹出的任何对话的。组件需要能够同脚本和其他组件进行交流，却不需同用户交流。应该避免所有的消息框和其他任何图形的用户界面单元。必须开发利用返回结果来同状态和其他模块信息进行交流的代码。如果出现问题，不要抛出一个错误消息或者使用一个消息框，可以返回一个状态变量。需要做的最后一件事是锁定忙碌服务器等待 OK 按钮被按下。

（2）服务器端组件不得被传递引用或者传递引用给对象。

普遍的做法是把控制作为一个参数传递给其他过程或组件。这包括其他对象的引用，比如 RecordSets。尽管如此，向网络中的组件传递引用可能导致速度明显下降，使一个繁忙的服务器更加缓慢，网络应用程序在响应用户需求时也表现得更慢。

（3）服务器端组件应该尽可能地少含方法和属性。

每一个方法或属性的调用都需要大量处理。因此，一个编写得好的服务器端组件应该几乎不含明显的方法和属性。组件含有的那些方法和属性会带来更多的参数。具有很多参数的调用越少，性能就越好，尤其是网络应用程序需要支持多用户时。这个技巧和许多开发人员的经验是相反的。尽量少使用带有许多参数的调用也会带来另外一些问题，使得编码和调试更加困难，但是速度上的改进是与付出的努力相当的。

（4）服务器端组件必须实现恰当的线程模型。

利用单线程组件可能导致服务器限制一个线程的会话，这将带来速度的明显下降。应该选择 VB 的公寓模型线程选项并且努力避免单线程组件。但是，VB 不能创建这种线程模型，可以在 Visual C++里的具有线程选择范围的组件。这一点也表明 VB 不是一种很合适这项工作的开发语言。

（5）服务器端组件应该使用早期绑定。

如果服务器应用程序要扩容，这显得特别重要。早期绑定的对象在编译时就拥有引用解析，可以节省不少执行时间。

（6）服务器端组件不能使用在应用程序或会话作用域的声明中。

请注意控件是如何被限定作用域的。作用域描述了如何创建一个组件的实例，这对服务端组件的成功起着关键作用。正如本书先前所讨论的那样，存在 3 种级别的作用域：页面级，会话级和应用程序级。页面级作用域对象可以用页面本身的 HTML 和 ASP 脚本代码来创建。页面级作用域组件的最佳性能可以通过使用公寓线程来得到。而对应用程序级和会话级作用域组件来说，可以通过使用 ATL 组件的双线程模型来得到。同作用域结合的线程模型也影响服务器的安全性能。例如，一个利用应用程序级作用域的公寓线程组件在系统安全环境里运行，但并不是当前用户的安全环境。这对那些具有安全意识的人来说可能是一个问题。

（7）为了速度，服务器端组件应该是进程中组件，为了稳定，则应该是进程外组件。

有两种方式去创建 COM（OLE）服务器：进程中和进程外。在 VB 里，用.exe 或.dll 扩展名去编译服务器。具有.dll 扩展名的 OLE 服务器被称为进程中服务器，而具有.exe 扩展名的被叫做进程外服务器。进程外意味着组件作为一个独立的过程在运行，而且与调用它的应用程序不共享地址空间。运行进程外组件会导致性能的降低，因为 Windows 不得不在两个或多个应用程序之间来回移动数据。进程中意味着组件在调用它的应用程序的地址空间里运行。在过程之间交流无需中间物，这使性能显著提高。进程中组件的负面影响是如果组件失败，那调用它的应用程序也会失败。

服务器端组件使得创建一流的解决方案成为可能。利用 IIS，可能还得用上 MTS，可以基于 Windows NT 的强有力的处理能力创建高性能的可升级的网络应用程序。

20.6 小 结

本章学习了 ActiveX 组件的含义、作用和使用方法。正是由于 ActiveX 组件的存在，才使 ASP 可以实现强大的功能，可以扩展出很多用途。可以说 ActiveX 是 ASP 的核心。

本章介绍了一些 ASP 可安装组件和一个第三方组件 Jmail 的用法，其实组件的内容远远不止如此，这里不过是让读者稍微领略一下组件的风采而已，如果读者想进一步了解组件，可以参考专门的书籍。

下一章将介绍另一个十分重要的 ASP 可安装组件——文件系统组件 FileSystemObject。

第 21 章　File Access 组件对象

File Access 组件对象是 ASP 中一个十分重要的对象，提供对计算机文件系统的访问。它提供对文件对象的操作，一般称为 FileSystemObject（FSO）对象，借助它可以实现 Web 对文件系统的操作。由于它包含的内容多而且作用十分的重要，所以这里单独列出一章来介绍这个对象。

其实 FSO 是 Scripting 运行类库的一个对象，这里把它归属为常用的 ASP 可安装组件是为了便于介绍。

FSO 对象模式所包含的对象和集合见表 21.1。

表 21.1　FileSystemObject（FSO）对象所包含的对象和集合。

对象/集合	描述
FileSystemObject	主对象。包含用来创建、删除和获得有关信息，以及通常用来操作驱动器、文件夹和文件的方法和属性。和该对象相关联的许多方法，与其他 FSO 对象中的方法完全相似，它们是为了方便才被提供的
Drive	对象。包含用来收集信息的方法和属性，这些信息是关于连接在系统上的驱动器的，如驱动器的共享名和它有多少可用空间。请注意，Drive 并非必须是硬盘，也可以是 CD-ROM 驱动器，RAM 磁盘等等。并非必须把驱动器实物地连接到系统上；它也可以通过网络在逻辑上被连接起来
Drives	集合。提供驱动器的列表，这些驱动器实物地或在逻辑上与系统相连接。Drives 集合包括所有驱动器，与类型无关。要可移动的媒体驱动器在该集合中显现，不必把媒体插入到驱动器中
File	对象。包含用来创建、删除或移动文件的方法和属性。也用来向系统询问文件名、路径和多种其他属性
Files	集合。提供包含在文件夹内的所有文件的列表
Folder	对象。包含用来创建、删除或移动文件夹的方法和属性。也用来向系统询问文件夹名、路径和多种其他属性
Folders	集合。提供在 Folder 内的所有文件夹的列表
TextStream	对象。用来读写文本文件

下面各节来分别介绍 FSO 的具体用法。

21.1　FileSystemObject 对象

利用 FileSystemObject 对象编程首先要创建 FileSystemObject 对象，创建的方法如下所

示：

```
<% Dim fso%>
<% Set fso=CreateObject("Scripting.FileSystemObject") %>
```

其中 Scripting 是类型库的名字，而 FileSystemObject 则是想要创建的对象的名字。可以看出 FSO 是 Scripting 类库的一个对象，在实际要用时才创建。

FileSystemObject 对象的方法有：

1．CreateTextFile 方法

该方法创建指定文件并返回 TextStream 对象，该对象可用于读或写创建的文件。其语法如下：

```
object.CreateTextFile(Filename[,Overwrite[,Unicode]])
```

其中：

- *object*：FileSystemObject 或 Folder 对象的名称。
- *Filename*：要创建的文件，必须为字符串。
- *Overwrite*：创建模式，指明是否可以覆盖现有文件。如果为 True 表示覆盖文件；如果为 False 表示不覆盖文件，默认值为 False。该值为布尔类型。
- *Unicode*：指明是否以 Unicode 或 ASCII 文件格式创建文件。为 True 时表示以 Unicode 文件格式创建文件，为 False 时表示以 ASCII 码文件格式创建文件。如果省略此部分，则假定创建 ASCII 码文件。该值为布尔类型。

以下为示例代码：

```
<%
Dim objFs
Dim ObjFile
Set objFs=CreateObject("Scripting.FileSystemObject")
Set objFile=objFs.CreateTextFile("C:\test.txt",True)
objFile.Close
%>
```

这段代码创建了一个 FileSystemObject 对象，然后利用该对象在 C:\创建一个 test.txt 文件。读者运行上面的代码后就可以看到 C:\多了这么一个文件。

注意：如果程序报错，那很可能是要创建文件的目录没有写的权限，因此要给这个目录开放写的权限。关于 FS0 权限的问题在后面会讲到。

2．DeleteFile 方法

该方法删除指定的文件。其语法如下：

```
object.DeleteFile Filespec[,Force]
```

其中：

- *object*：FileSystemObject 对象的名称。
- *Filespec*：要删除的文件名。*Filespec* 在路径的最后一个组成部分中可包含通配符。
- *Force*：Boolean 值。如果要删除只读文件，则该值为 True。否则为 False（默认）。

注意：如果没有找到匹配文件，则会出现错误。在删除文件前，可以用 FileExists 方法来判断文件是否存在。DeleteFile 方法在遇到出现的第 1 个错误时停止。该方法不会撤消错误发生前所作的任何更改。

以下为示例代码：

```
<%
Dim objFs
Dim ObjFile
Set objFs=CreateObject("Scripting.FileSystemObject")
Set objFile=objFs.DeleteFile("C:\test.txt",True)
objFile.Close
%>
```

这段代码创建了一个 FileSystemObject 对象，然后利用该对象删除了 C 盘上的一个 test.txt 文件。读者运行上面的代码后就可以看到上一个例子中创建的文件已经被删除了，同样这个方法也有权限的问题。

3．FileExists 方法

如果指定的文件存在返回 True，否则返回 False。该方法的语法如下：

```
object.FileExists(Filespec)
```

其中：

- *object*：FileSystemObject 对象的名称。
- *Filespec*：文件名，表示要确定是否存在的文件。如果文件不在当前文件夹中，则必须提供完整的路径名（绝对路径或相对路径）。

以下为示例代码：

```
<%
Dim objFs
Set objFs=CreateObject("Scripting.FileSystemObject")
Response.Write objFs.FileExists("C:\test.txt")
%>
```

4．GetFile 方法

该方法返回与指定路径中某文件对应的 File 对象。其语法如下：

```
object.GetFile(Filespec)
```

其中：

- *object*：FileSystemObjectd **对象的名称。**
- *Filespec*：**指定文件的路径（绝对路径或相对路径）。**

注意：如果文件不存在，程序会报错。

5．GetFileName **方法**

返回指定路径文件的文件名称。该方法的语法如下：

```
object.GetFileName(Pathspec)
```

其中：

- *object*：FileSystemObject **对象的名称。**
- *Pathspec*：**指定文件的路径（绝对路径或相对路径）。**

注意：如果指定路径错误或文件不存在，则返回空字符串。

6．CopyFile **方法**

复制指定一个或多个文件到指定目录。该方法的语法如下：

```
object.CopyFile Source,Destination[,Overwrite]
```

其中：

- *object*：FileSystemObject **对象的名称。**
- *Source*：**表示指定文件的字符串。要复制一个或多个文件时，文件名中可以有通配符。**
- *Destination*：**表示目标位置的字符串，从** *Source* **复制文件到该位置。不允许用通配符。**
- *Overwrite*：**可选。**Boolean **值表明是否覆盖现有文件。如果是** True**，则覆盖文件。如果是** False**，则不覆盖现有文件，默认值是** True**。注意，无论** *Overwrite* **设置为何值，只要设置** *Destination* **为只读属性，**CopyFile **操作就无法完成。**

以下为示例代码：

```
<%
Dim objFs
Set objFs=CreateObject("Scripting.FileSystemObject")
objFs.CopyFile "C:\test\*.doc","C:\temp\"
%>
```

这段代码将 C:\test **目录下所有以**.doc **为后缀的文件复制到** C:\temp **目录下。**

注意：通配符仅能在 *Source* **参数的路径最后一个组成部分使用。例如** objFs.CopyFile ″C:*est\test.doc″,″C:\temp\″ **的语法是错误的。还有如果**

Destination 是已经存在的文件，当 *overwrite* 为 False 时会出现错误。否则，复制 *source* 覆盖现有文件。

7．MoveFile 方法

该方法将一个或多个文件从某位置移动到另一位置。其语法如下：

```
object.MoveFile Source,Destination
```

其中：

- *object*：FileSystemObject 对象的名称。
- *Source*：要移动的文件的路径。*Source* 参数字符串仅可在路径的最后一个组成部分中用通配符。
- *Destination*：指定路径，表示要将文件移动到该目标位置。*Destination* 参数不能包含通配符。

以下为示例代码：

```
<%
Dim objFs
Set objFs=CreateObject("Scripting.FileSystemObject")
objFs.MoveFile "C:\test\*.doc","C:\temp\"
%>
```

8．CreateFolder 方法

该方法用来创建指定目录。其语法如下：

```
object.CreateFolder(Foldername)
```

以下为示例代码：

```
<%
Dim objFs
Set objFs=CreateObject("Scripting.FileSystemObject")
objFs.CreateFolder("C:\testfolder")
%>
```

这段代码在 C:\创建了一个名为 testfolder 的目录。

注意：如果要创建的目录已经存在，程序会出错。所以在实际使用中要先用 FolderExists 方法来判断一下要创建的文件夹是否已经存在了。

9．DeleteFolder 方法

该方法用来删除指定的文件夹和其中的内容。其语法如下：

```
object.DeleteFolder Folderspec[,Force]
```

其中：

- *object*：FileSystemObject 对象的名称。
- *Folderspec*：要删除的文件夹名称。*Folderspec* 在路径的最后一个组成部分中可包含通配符。
- *Force*：Boolean 值，可选。如果要删除只读文件夹，则该值为 True。否则为 False（默认）。

以下为示例代码：

```
<%
Dim objFs
Set objFs=CreateObject("Scripting.FileSystemObject")
objFs.DeleteFolder("C:\tes*",True)
%>
```

上面代码删除了 C:\中所有以 tes 开头的文件夹。

注意：DeleteFolder 方法不能判断文件夹中是否包含内容。无论文件夹是否包含内容，都将删除该文件夹。如果未找到匹配文件夹，则会出现错误。DeleteFolder 方法在遇到出现的第 1 个错误时停止。该方法不会撤消错误发生前所作的任何更改。

10．FolderExists 方法

如果指定的文件夹存在，则返回 True。否则返回 False。FolderExists 方法的语法和用法和 FileExists 差不多。

11．GetFolder 方法

该方法用来返回与指定的路径中某文件夹相应的 Folder 对象。

12．CopyFolder 方法

该方法用来将文件夹从某位置复制到另一位置。

13．MoveFolder 方法

该方法用来将一个或多个文件夹从某位置移动到另一位置。

14．DriveExists 方法

如果指定的驱动器存在，该方法返回 True，否则返回 False。

15．GetBaseName 方法

该方法用来返回字符串，该字符串包含路径最后一个组成部分的基本名，无扩展名。

16．GetDrive 方法

该方法用来返回与指定的路径中驱动器相对应的 Drive 对象。

17．GetDriveName 方法

该方法用来返回包含指定路径中驱动器名的字符串。如果指定的驱动器不存在，则返

回空串。

18．GetExtensionName 方法

该方法用来返回一个字符串，该字符串包含路径最后一个组成部分的扩展名。

19．GetExtensionName 方法

该方法用来返回一个字符串，该字符串包含指定的路径中最后一个组成部分的父文件夹。

20．GetExtensionName 方法

该方法用来返回指定的特殊文件夹。其语法如下：

```
object.GetSpecialFolder(Folderspec)
```

其中 *Folderspec* 为要返回的特殊文件夹的名称，可以是“设置”部分列出的任何常数，可为下列值之一：

- WindowsFolder：0，即 Windows 文件夹，包含 Windows 操作系统安装的文件。
- SystemFolder：1，即 System 文件夹，包含库、字体和设备驱动程序文件。
- TemporaryFolder：2，即 Temp 文件夹，用于保存临时文件。可以在 TMP 环境变量中找到该文件夹的路径。

21．GetTempName 方法

该方法返回随机生成的临时文件或文件夹的名称，用于执行要求临时文件或文件夹的操作。

FileSystemObject 只有一个属性为 Drives，它表示服务器上所有驱动器的集合。其语法如下：

```
object.Drives
```

以下为示例代码：

```
<%
'------------------------------------------------------------
'目的：列出服务器中所有驱动器盘符。
'------------------------------------------------------------
Dim objFs
Dim thing
Set objFs=CreateObject("Scripting.FileSystemObject")
For Each thing In objfs.drives
  Response.Write thing
Next
%>
```

这段代码先取得服务器上所有驱动器的集合，然后用一个 For Each…Next 循环列出集合中的元素的名称即驱动器盘符。

提示：这里仅列出了盘符，结合 Drives 对象还可以列出更详细的信息，请读者参考下节的 Drives 对象。

21.2 Drive 对象

Drive 对象提供对磁盘驱动器或网络共享资源的访问，它没有任何方法。它的属性有：

1．AvailableSpace 属性

该属性表示指定的驱动器或网络共享对于用户的可用空间大小。

2．DriveLetter 属性

该属性表示本地驱动器或网络共享的驱动器号，只读。

3．DriveType 属性

该属性表示一个描述指定驱动器的类型的数字值。各个数字代表的意思分别为：

0：**未知驱动器**
1：**可移动驱动器**
2：**固定驱动器**
3：**网络驱动器**
4：CD-ROM
5：RAM **磁盘**

4．FileSystem 属性

该属性表示指定的驱动器所使用的文件系统的类型。可用的类型包括 FAT、NTFS **和** CDFS。

5．FreeSpace 属性

该属性表示指定的驱动器或网络共享对于用户的可用空间大小，只读。

注意：FreeSpace **属性与** AvailableSpace **属性的值基本相同。对于支持限额的计算机系统来说，这两个属性的值有所差异。如果有限额的话，**AvailableSpace **最大为规定的限额。**

6．IsReady 属性

如果指定的驱动器就绪，其值 True**，否则为** False。

7．Path 属性

该属性表示指定文件、文件夹或驱动器的路径。

注意：对于驱动器而言，其路径不包括根目录，比如 C **驱动器返回的就是** C:**而不是** C:\。

8．RootFolder 属性

该属性表示指定驱动器的根文件夹是一个 Folder 对象，只读。

9．SerialNumber 属性

该属性表示一个十进制序列号，用于惟一标识一个磁盘卷。

10．ShareName 属性

该属性表示指定的驱动器的网络共享名。

11．TotalSize 属性

该属性表示驱动器或网络共享的总字节数。

12．VolumeName 属性

该属性表示指定驱动器的卷标，可读写。其语法如下：

```
object.VolumeName[=newname]
```

其中 *newname* 可选。如果提供此参数，则 *newname* 为指定的 *object* 的新名称。

下面用一个例子来说明这些属性的使用方法

```
<%
'-------------------------------------------------------
'目的：列出驱动器的详细信息
'-------------------------------------------------------
Dim fso,drv,str,drvPath,strType
drvPath="d:"
Set fso=CreateObject("Scripting.FileSystemObject")
Set drv=fso.GetDrive(fso.GetDriveName(drvPath))
Select Case drv.DriveType
  Case 0:strType="未知驱动器"
  Case 1:strType="可移动驱动器"
  Case 2:strType="固定驱动器"
  Case 3:strType="网络驱动器"
  Case 4:strType="CD-ROM"
  Case 5:strType="RAM 磁盘"
End Select
str="Drive"&UCase(drvPath)&"-"
str=str&drv.VolumeName&"-"&strType&"<br>"
str=str&vbCrLf&"序列号："&drv.SerialNumber&"<br>"
If drv.IsReady Then
  str=str&vbCrLf&"驱动器已就绪。"&"<br>"
Else
  str=str&vbCrLf&"驱动器未就绪。"&"<br>"
End If
str=str&"文件系统"&"-"&drv.FileSystem&"<br>"
```

```
str=str&"总共空间:"&FormatNumber(drv.TotalSize/1024,0)
str=str&"KB"&"<br>"
str=str&"空白空间:"&FormatNumber(drv.FreeSpace/1024,0)
str=str&" KB" & "<br>"
str=str&"可用空间:"&FormatNumber(drv.AvailableSpace/1024,0)
str=str&"KB"&"<br>"
If drv.ShareName<>"" Then
  str=str&"网络共享名:"&drv.ShareName&"<br>"
Else
  str=str&"不是网络驱动器<br>"
End If
Response.Write str
%>
```

这段代码很简单，这里就不再多说明了，读者可以参照上面的例子使用 Drive 对象的属性。

21.3 File 对象及其属性集合

File 对象提供对文件所有属性的访问，利用文件对象可以实现对文件的操作。
File 对象所提供的方法有：

1．Copy 方法

该方法用来将指定的文件或文件夹从某位置复制到另一位置。其语法如下：

```
object.Copy Destination[,Overwrite]
```

其中：

- *object*：File 或 Folder 对象的名称。
- *Destination*：复制文件或文件夹的目标位置。不允许使用通配符。
- *Overwrite*：Boolean 值，可选。如果覆盖现有文件或文件夹，则该值为 True（默认）；否则为 False。

注意 对 File 或 Folder 应用 Copy 方法的结果与使用 FileSystemObject.CopyFile 或 FileSystemObject.CopyFolder 方法执行的操作完全相同。在 FileSystemObject.CopyFile 或 FileSystemObject.CopyFolder 中，使用 object 引用文件或文件夹，并将文件或文件夹作为参数传递给 FileSystemObject.CopyFile 或 FileSystemObject.CopyFolder。然而，应该注意的是，FileSystemObject.CopyFile 或 FileSystemObject.CopyFolder 方法可以复制多个文件或文件夹。

2．Delete 方法

该方法用来删除指定的文件或文件夹。其语法如下：

```
object.Delete [Force]
```

其中：

- *object*：File 或 Folder 对象的名称。
- *Force*： Boolean 值，可选。如果要删除的文件或文件夹的属性设置为只读属性，则该值为 True，否则为 False（默认）。

3．Move 方法

该方法用来将指定的文件或文件夹从某位置移动到另一位置。其语法如下：

```
object.Move Destination
```

其中：

- *object*：File 或 Folder 对象的名称。
- *Destination*：目标位置。表示要将文件或文件夹移动到该位置。不允许使用通配符。

注意：使用 Move 方法一次只能移动一个文件或文件夹。

4．OpenAsTextStream 方法

该方法用来打开指定的文件并返回一个 TextStream 对象，此对象用于对文件进行读、写或追加操作。其语法如下：

```
object.OpenAsTextStream([Iomode,[Format]])
```

其中：

- *object*：File 对象的名称。
- *Iomode*：可选。输入/输出模式，是 3 个常数之一：ForReading、ForWriting 或 ForAppending。其中：
  - ForReading：1，表示以只读模式打开文件。不能对此文件进行写操作。
  - ForWriting：2，表示以可读写模式打开文件。如果已存在同名的文件，则覆盖旧的文件。
  - ForAppending：8，表示打开文件并在文件末尾进行写操作。
- *Format*：可选。3 个 Tristate 值之一，指出以何种格式打开文件。忽略此参数，则文件以 ASCII 码格式打开。其中：
  - TristateUseDefault：2 表示以系统默认格式打开文件。
  - TristateTrue：-1，表示以 Unicode 格式打开文件。
  - TristateFalse：0，表示以 ASCII 码格式打开文件。

OpenAsTextStream 方法可提供与 FileSystemObject 对象的 OpenTextFile 方法相同的功能。另外，使用 OpenAsTextStream 方法可对文件进行写操作。关于 OpenAsTextStream 对象，读者可以参考本章后面介绍的 Stream 对象部分。

Files 集合是指由指定文件夹中所有 File 对象（包括隐藏文件和系统文件）所组成的集合。

以下为示例代码：

```
<%
'------------------------------------------------------
'利用 Files 属性集合列出指定文件夹中的所有文件。
'------------------------------------------------------
Dim fso,file,thing,str,strFolder
strFolder="c:\temp"
Set fso=CreateObject("Scripting.FileSystemObject")
Set file=fso.GetFolder(strFolder)
For Each thing in file.files
  str=str&thing.name
  str=str&"<br>"
Next
Response.Write str
%>
```

这段代码首先创建一个 FileSystemObject 组件对象（第 7 行），再获得指定文件夹的 Files 属性集合（第 8 行），然后利用一个循环取得集合中的所有元素的名称并赋值给一个变量（第 9、10、11、12 行），最后在页面上显示出这个变量，即显示出文件夹中所有文件的名称（第 13 行）。

21.4 Folder 对象和 Folders 集合

Folder 对象提供对文件夹所有属性的访问，利用 Folder 对象可以实现对文件夹的操作。Folder 对象所提供的方法有：

1．Copy 方法

该方法用来将指定的文件或文件夹从某位置复制到另一位置。其语法如下：

```
object.Copy Destination[,Overwrite]
```

其中：

- *object*：File 或 Folder 对象的名称。
- *Destination*：复制文件或文件夹的目标位置。不允许使用通配符。
- *Overwrite*：Boolean 值，可选。如果覆盖现有文件或文件夹，则其该值为 True（默认），否则为 False。

2．Delete 方法

该方法用来删除指定的文件或文件夹。其语法如下：

```
object.Delete [Force]
```

其中：

- *object*：File 或 Folder 对象的名称。
- *Force*： Boolean 值，可选。如果要删除的文件或文件夹的属性设置为只读属性，则值为 True，否则为 False（默认）。

3．Move 方法

该方法用来将指定的文件或文件夹从某位置移动到另一位置。其语法如下：

```
object.Move Destination
```

其中：

- *object*：File 或 Folder 对象的名称。
- *Destination*：目标位置。表示要将文件或文件夹移动到该位置。不允许使用通配符。

4．CreateTextFile 方法

该方法用来创建指定文件并返回 TextStream 对象，该对象可用于读或写创建的文件。其语法如下：

```
object.CreateTextFile(Filename[,Overwrite[,Unicode]])
```

其中：

- *object*：FileSystemObject 或 Folder 对象的名称。
- *Filename*：字符串表达式，指明要创建的文件。
- *Overwrite*：Boolean 值，可选。指明是否可以覆盖现有文件。如果可覆盖文件，该值为 True；如果不能覆盖文件，则该值为 False。如果省略该值，则不能覆盖现有文件。
- *Unicode*：Boolean 值，可选。指明是否以 Unicode 或 ASCII 码文件格式创建文件。如果以 Unicode 文件格式创建文件，则该值为 True，如果以 ASCII 码文件格式创建文件，则该值为 False。如果省略此部分，则假定创建 ASCII 码文件。

Folder 对象提供的属性有：

1．Attributes 属性

该属性表示文件或文件夹的属性，可读写或只读（与属性有关）。其语法如下：

```
object.Attributes[=Newattributes]
```

其中：

- *object*：File **或** Folder **对象的名称。**
- *Newattributes*：**可选。如果指定此参数，则** *Newattributes* **为指定的** *object* **的属性的新值。其可能的取值为：**
  - Normal：0，**表示为普通文件。没有设置任何属性。**
  - ReadOnly：1，**表示为只读文件。可读写。**
  - Hidden：2，**表示为隐藏文件。可读写。**
  - System：4，**表示为系统文件。可读写。**
  - Volume：8，**表示为磁盘驱动器卷标。只读。**
  - Directory：16，**表示为文件夹或目录。只读。**
  - Archive：32，**表示为上次备份后已更改的文件。可读写。**
  - Alias：64，**表示为链接或快捷方式。只读。**
  - Compressed：128，**表示为压缩文件。只读。**

2．DateCreated **属性**

该属性表示指定的文件或文件夹的创建日期和时间。只读。

3．DateLastAccessed **属性**

该属性表示指定的文件或文件夹的最后访问日期和时间。只读。

4．DateLastModified **属性**

该属性表示指定的文件或文件夹的最后修改日期和时间。只读。

5．Drive **属性**

该属性表示指定的文件或文件夹所在的驱动器的驱动器号。只读。

6．Name **属性**

该属性表示指定的文件或文件夹的名称。可读写。

7．ParentFolder **属性**

该属性表示指定文件或文件夹的父文件夹。只读。

8．Path **属性**

该属性表示指定文件、文件夹或驱动器的路径。

9．ShortName **属性**

该属性表示按照早期 8.3 **文件命名约定转换的短文件名。**

10．ShortPath **属性**

该属性表示按照 8.3 **命名约定转换的短路径名。**

11．Size **属性**

对于文件，该属性表示指定文件的字节数，对于文件夹，表示该文件夹中所有文件和子文件夹的字节数。

下面用一个例子来说明这些属性的使用方法，代码如下：

```
<%
'-------------------------------------------------------
'目的：列出文件夹 D:\Inetpub\wwwroot 的详细信息。
'-------------------------------------------------------
Dim fso,strFolder,str,strFolderPath,strType
strFolderPath="d:\Inetpub\wwwroot"
Set fso=CreateObject("Scripting.FileSystemObject")
Set strFolder=fso.GetFolder(strFolderPath)
str="文件夹"&UCase(strFolderPath)&"-"
str=str&strFolder.Name&"<br>"
str=str&"短文件名"&strFolder.ShortName&"<br>"
str=str&"短路径为"&strFolder.ShortPath&"<br>"
str=str&"位于驱动器"&UCase(strFolder.Drive)&"<br>"
str=str&"其父文件夹为"&UCase(strFolder.ParentFolder)&"<br>"
str=str&"创建时间"&"-"&strFolder.DateCreated&"<br>"
str=str&"最后访问时间:"&strFolder.DateLastAccessed&"<br>"
str=str&"最后修改时间:"&strFolder.DateLastModified&"<br>"
str=str&"大小为："&FormatNumber(strFolder.Size/1024,0)
str=str&"KB"&"<br>"
Response.Write str
%>
```

这段代码也都简单，这里就不再多说明了，读者可以参照上面的例子使用 Folder 对象的属性。

Folders 集合是指包含在一个 Folder 对象中的所有 Folder 对象的集合。
以下为示例代码：

```
<%
'-------------------------------------------------------------
'利用 Folders 集合列出文件夹 D:\Inetpub\wwwroot 中所有子文件夹。
'-------------------------------------------------------------
Dim fso,folder,thing,str,strFolder
strFolder="d:\Inetpub\wwwroot"
Set fso=CreateObject("Scripting.FileSystemObject")
Set folder=fso.GetFolder(strFolder)
For Each thing in folder.SubFolders
  str=str&thing.name
  str=str&"<br>"
Next
Response.Write str
%>
```

这段代码首先创建一个 FileSystemObject 组件对象（第 7 行），再获得指定文件夹的 Folders 集合（第 8 行），然后利用一个循环取得集合中的所有元素的名称并赋值给一个变量（第 9，10，11，12 行），最后在页面上显示出这个变量，即显示出文件夹中所有子文件夹的名称（第 13 行）。

21.5 TextStream 对象

这是 FileSystemObject 最重要的一个对象，对文件内容的操作全部由这个对象来实现。TextStream 对象的语法为：

```
TextStream.{property|method}
```

注意：*property* 和 *method* 参数可以是任何与 TextStream 对象相关联的属性和方法。请注意在实际使用时，TextStream 对象由代表从 FileSystemObject 返回的 TextStream 对象的变量取代。

取得 TextStream 对象很简单，首先要创建一个 FileSystemObject 对象，再通过 FileSystemObject 来得到 TextStream 对象。示例代码如下：

```
<%
dim fso
dim objtext
Set fso=CreateObject("Scripting.FileSystemObject")
Set objText=fso.CreateTextFile("c:\test.txt",True)
%>
```

这段代码就创建了一个名称为 objText 的 TextStream 对象。

下面来介绍 TextStream 对象的属性和方法。

TextStream 对象的基本方法有：

1．Close 方法

该方法用来关闭打开的 TextStream 文件。

2．Read 方法

该方法用来从 TextStream 文件中读入指定数目的字符并返回结果字符串。其语法如下：

```
object.Read(characters)
```

其中：

- *object*：TextStream 对象的名称。
- *Characters*：要从文件读的字符数目。

以下为示例代码：

```
<%
dim fso
dim objtext
dim str
Set fso=CreateObject("Scripting.FileSystemObject")
Set objText=fso.CreateTextFile("c:\test.txt",True)
str=objText.Read(3)
Response.Write str
%>
```

这段代码读出文本文件 C:\test.txt **的头** 3 **个字符并显示出来。**

3．ReadAll **方法**

该方法用来读入全部 TextStream **文件并返回结果字符串。其语法如下：**

```
object.ReadAll
```

把上面例子的 str=objText.Read(3)**改成** str=objText.ReadAll **就读取了整个文本文件的内容并显示出来。**

注意：这个方法在读取大文本的时候十分浪费资源，建议慎用，可以用下面介绍的 ReadLine **来代替。**

4．ReadLine **方法**

该方法用来从 TextStream **文件中读入一整行字符(直到下一行,但不包括下一行字符)，并返回结果字符串。其语法如下：**

```
object.ReadLine
```

5．Skip **方法**

该方法用来在读取 TextStream **文件时跳过指定数目的字符。**

6．SkipLine **方法**

该方法用来在读取 TextStream **文件跳到下一行。其语法如下：**

```
object.SkipLine
```

注意：跳过一行意味着读并放弃本行所有字符并包括下一新行字符内容。如果文件不是以只读方式打开则会出现错误。

7．Write **方法**

该方法用来向 TextStream **文件写入指定字符串。其语法如下：**

```
object.Write(string)
```

其中 *string* **为要写入文件的文本。**

注意：指定的字符串被写入文件中时，字符串之间没有插入空格或字符。使用 WriteLine 方法写入新行字符或以新行字符结束的字符串。

8．WriteLine 方法

该方法用来向 TextStream 文件写入指定字符串，并加入换行符。其语法如下：

```
object.Write(string)
```

其中 *string* 为要写入文件的文本。

从下面例子可以看出 Write 和 WriteLine 的两种方法用法和不同之处。

```
<%
'-----------------------------------------------------
'目的：演示 Write 和 WriteLine 的用法并比较它们的不同。
'-----------------------------------------------------
dim fso
dim objtext
dim str
Set fso=CreateObject("Scripting.FileSystemObject")
Set objText=fso.CreateTextFile("c:\test.txt", True)
objText.Write("这是用 Write 方法写入的")
objText.Write("这也是用 Write 方法写入的")
objText.WriteLine("这是用 WriteLine 方法写入的")
objText.WriteLine("这也是用 WriteLine 方法写入的")
str=objText.ReadAll
ojbText.Close
Response.Write str
%>
```

这段代码分别用 Write 和 WriteLine 方法各向文本文件 C:\test.txt 写入了两次字符串。如果打开该文本文件可以很清楚地看到，两次用 Write 方法写入的字符串并没有换行，而两次用 WriteLine 方法写入两次的字符串换了行。

9．WriteBlankLines 方法

该方法用来向 TextStream 文件写入指定的空白行。其语法如下：

```
object.WriteBlankLines(lines)
```

其中 *lines* 表示要向文件写入的新行的字符数目。

TextStream 对象的属性主要有：

1．AtEndOfLine 属性

该方法用来判断 TextStream 文件的文件指针是否指向行末标记，是其值为 True，否则为 False。

其语法如下：

```
object.AtEndOfLine
```

注意：AtEndOfLine **属性仅应用于以只读方式打开的** TextStream **文件，否则会出现错误。**

2．AtEndOfStream 属性

该属性用来判断在 TextStream **文件中文件指针是否指向文件末，是则其值为** True**；否则为** False**。**

其语法如下：

```
object.AtEndOfStream
```

注意：TextStream **属性仅应用于以只读方式打开的** TextStream **文件，否则会出现错误。**

上面提到过，由于利用 ReadAll **方法读取大文本的时候会很浪费内存资源，实际使用的时候可以利用** ReadLine **方法来代替，其实实现这个目的需要和** AtEndOfStream **方法结合起来。**

以下为示例代码：

```
<%
'-------------------------------------------------------
'目的：用 AtEndOfStream 和 ReadLine 方法代替 ReadAll 方法。
'-------------------------------------------------------
Dim fso,objText,retstring
Set fso=CreateObject("Scripting.FileSystemObject")
Set ojbText=fs.OpenTextFile("c:\test.txt",ForReading,False)
Do While objText.AtEndOfStream<>True
  retstring=objText.ReadLine
  objText.SkipLine
Loop
Response.Write retstring
ojbText.Close
%>
```

3．Column 属性

该属性为只读属性，表示 TextStream **文件中当前字符位置的列号。其语法如下：**

```
object.Column
```

注意：在写入新行字符后而未写其他字符前，Column **等于** 1**。**

4．Line 属性

该属性为只读属性，表示 TextStream **文件中当前字符位置的行号。其语法如下：**

```
object.Line
```

注意：文件刚刚打开并在写入任何信息前，Line **等于** 1**。**

利用 FileSystemObject 对象可以实现很强大的功能，比如做网站发布系统、网站内容管理等等。下面将介绍利用 FileSystemObject 做一个简单的搜索引擎。

首先要创建一个搜索的页面，代码如下：

```
<html>
<title>搜索页面</title>
<Form action="search.asp" method="post">
  <input type="text" name="search">
  <input type="submit" value="提交">
</Form>
</html>
```

先要获得要搜索的字符串，代码如下：

```
<%
Dim strSearch
StrSearc  Request Form("Search")
%>
```

然后介绍如何在一个目录中按关键字搜索。

先要创建一个 FileSystemObject 对象，如下所示：

```
<%
Dim fso
Set fso=Server.CreateObject("Scripting.FileSystemObject")
%>
```

再获得要搜索目录的物理路径，在上一章中已经学习了如何获得一个网站目录的物理目录（利用 Server.MapPath("/html")）。这里假设要搜索的目录就是网站根目录下面的 html 目录。

然后获得一个 Folder 对象，如下所示：

```
<%
Dim objFolder
Set objFolder=objFSO.GetFolder(Server.MapPath("/html"))
%>
```

再在该 Folder 对象集合中查找所有文件，与有搜索的字符串作比较，看是否符合条件，如果有的话就列出文件的名称，并计数。代码如下：

```
<%
Dim objFile
Dim strContent
Dim intCount
IntCoun=0  '初始化计数器
For Each objFile In objFolder.Files
```

```
  '以只读方式打开文件
  Set objText=fsoOpenTextFile(objFile.Path,1)
  '读取文件内容并存放在一个变量中
  strContent=objText.ReadAll
  '比较文件中是否存在要搜索的字符串，如果存在就列出文件的名称
  If InStr(strContent,strSearch) Then
    Response.Write objFile.Name&"<br>"
    IntCount=intCount+1
  End If
  objText.Close
Next
%>
```

需要搜索目录中所有子目录的内容，这里使用一个循环来实现，即在每一个子目录中都作上面的步骤。代码如下：

```
<%
For Each objSubFolder In objFolder.SubFolders
……
Next
%>
```

把搜索的步骤写成一个函数，这样每次只需要在子目录中执行这个函数即可。完整的程序代码如下：

```
<%
'---------------------------------------------------------------------
'目的：在目录中搜索文件，看是否存在指定的字符串，并列出符合条件的文件的名称和数目。
'---------------------------------------------------------------------
Dim fso
Dim objFolder
Dim strSearch
Dim intCount
'获得搜索字符串
strSearch=Request.Form("Search")
Response.Write "要搜索的字符串是："&strSearch&"<br>"
'初始化计数器
intCount=0
'创建 FileSystemObject 和 Folder 对象
Set fso=Server.CreateObject("Scripting.FileSystemObject")
Set objFolder=fso.GetFolder(Server.MapPath("/html"))
'搜索本目录
funSearch objFolder
'利用循环在各个子目录中执行搜索函数
For Each objSubFolder In objFolder.SubFolders
  funSearch objSubFolder
```

```
Next
Response.Write "共找到"&intCount
'创建函数
Function funSearch(objSubFolder)
  Dim objFile
  Dim objText
  Dim strContent
  '利用循环取得当前搜索目录中的所有文件
  For Each objFile In objSubFolder.Files
    '以只读方式打开文件
    Set objText=fso.OpenTextFile(objFile.Path,1)
    '读取当前文件的内容并存放于一个变量
    strContent=objText.ReadAll
    '再在该 Folder 对象集合中查找所有文件,
    '与有搜索的字符串作比较,看是否符合条件,
    '如果有的话就列出文件的名称,并计数
    If InStr(strContent,strSearch) Then
      Response.Write objFile.Name &"——"
      Response.Write objSubFolder.path&"<br>"
      IntCount=intCount+1
    End If
    objText.Close
  Next
End Function
%>
```

将两个文件分别保存为 searchmainform.html 和 search.asp,在 searchmainform.html 中输入要搜索的字符串并提交,可以得到如图 21.1 所示的结果。

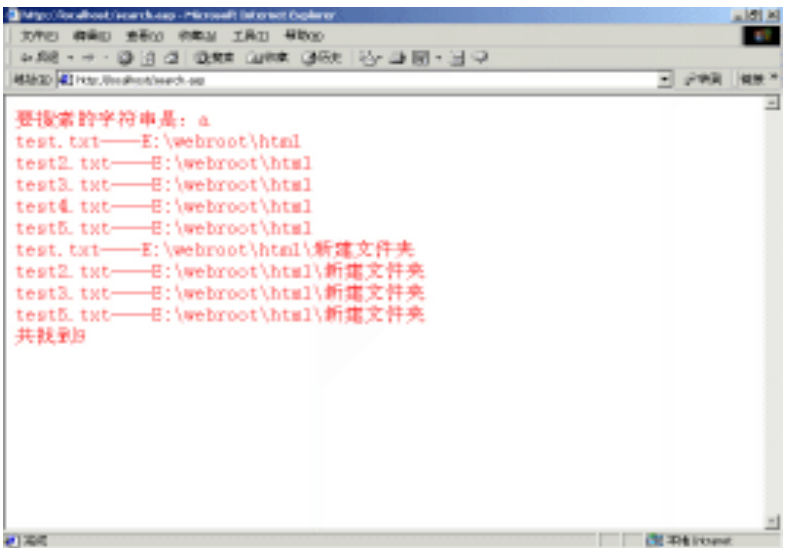

图 21.1 搜索得到的结果画面

其实，这就是一个简单的搜索引擎，使用起来十分方便。这种方法不需要再安装其他任何的组件，实现起来也很简单，缺点是速度比较慢，如果是大流量的搜索是不适合的，随着文件和子目录数量的增加，执行搜索所花费的时间也将增加。如果需要实现繁重的搜索工作，可以用专门的组件来实现，比如 Microsoft 的索引服务器（Index Server）。

21.6　FileSystemObject 的权限设置与安全

运行 FileSystemObject 需要一定的权限。FSO 以创建它的用户帐户权限运行，换言之，如果有人从 Internet 上访问 Web 页面，那么这个 Internet 帐户就创建 FSO。如果以 Administrator 的身份登录计算机，并且登录页面，那么 Administrator 帐户就创建了 FSO。这是非常重要的，因为一定的帐户拥有一定的权限，并且 FSO 需要一些权限从而能完整的执行功能。

要使用 FSO 就必须开放要操作文件的权限，这个可以 IIS 中进行设置，也可以直接设置文件的属性。也可以设置执行文件的帐户，指定专门的帐户来执行该程序，然后开放帐户的权限。

正是由于 FSO 功能强大，同时就带来了安全问题，因为 FileSystemObject 可以操作文件，如果别人可以利用这个组件，就会有可能带来恶意的攻击，而且破坏力的很大的。因此必须做好安全防范的工作。主要措施为删掉 FSO 组件或者在注册表中将 FSO 组件改名。

有关安全性问题，本书第 24 章将有专门的论述。

21.7　小 结

本章介绍了 FileSystemObject 组件的用途、它的各个对象属性的作用和使用方法，同时还学习了使用 FSO 来做一个小型的搜索引擎。

可以看出 FSO 是一个十分重要、功能十分强大的组件，通过它可以实现用 ASP 来对文件、目录的操作，很轻松地实现远程网站目录管理。同时正是它作用强大，因此一旦被用意不良的人利用，就会对网站带来致命的攻击。因此在不需要使用此组件的时候，可以将它屏蔽，或者改名。

下一章将讲解网站数据库，它是 ASP 最大的应用领域。

第 22 章　网站数据库

简单地说，一个网站数据库就是指用户在客户端利用浏览器作为输入界面，输入所需的数据，然后浏览器将这些数据返回给网站。而网站再对这些数据进行处理，例如将数据存入数据库，或者对数据库进行查询操作等。最后网站再将执行的结果返回给浏览器，通过浏览器将结果显示给用户。

网站数据库让 WWW 有了一个继续发展的空间，而且也带来了商机。例如，网上定票、资料查询、网上下定单等，而这些都需要网站与数据库的紧密结合。

本章将介绍网站数据库的实现方式，重点介绍网站数据库的基础 ADO 和 SQL。

22.1　SQL 基础

SQL（Structured Query Language，结构化查询语言）语言是数据库的标准语言。在 ASP 编程中，无论何时要访问一个数据库，都要使用 SQL 语言。因此，掌握好 SQL 对 ASP 编程非常重要。

本章中是以操作 Microsoft SQL Server 自带的数据库为样例的，当然也可以用 SQL 操作许多其他类型的数据库。

1．SQL 的特点

在学习 SQL 的细节之前，需要理解它的两大特点。其中一个特点容易掌握，而另一个特点掌握起来有点困难。

第 1 个特点是，所有 SQL 数据库中的数据都存储在表中，一个表由行和列组成。例如：

```
Name               EmailAddress
.............................................................
Bill Gates         billg@microsoft.com
president Clinton  president@whitehouse.com
Stephen Walther    swalther@somewhere.com
```

这个表有两列（列也称为字段、域）：Name 和 EmailAddress。有 3 行，每一行包含一组数据。一行中的数据组合在一起称为一条记录。一个数据表可以有几十个记录，也可以有几千甚至几十亿个记录。

数据库很有可能包含几十个表，所有存储在数据库中的信息都被存储在这些表中。当考虑怎样把信息存储在数据库中时，就是考虑怎样把它们存储在表中。

SQL 的第 2 个特点有些难于掌握。这种语言被设计为不允许按照某种特定的顺序来取出记录，因为这样做会降低 SQL Server 读取记录的效率。使用 SQL，只能按查询条件来读取记录。

当考虑如何从表中取出记录时，自然会想到按记录的位置读取它们。例如，也许读者会尝试通过一个循环逐个记录地扫描来选出特定的记录。在使用 SQL 时，必须训练自己不要有这种思路。

假如想选出所有的名字是 Bill Gates 的记录，如果使用传统的编程语言，也许会构造一个循环，逐个查看表中的记录，看名字域是否是 Bill Gates。

这种选择记录的方法虽然是可行的，但是效率不高。使用 SQL，只要设置查询条件为“选择所有名字域等于 Bill Gates 的记录”，SQL 会确定实现查询的最佳方法，并选出所有符合条件的记录。

假如想取出表中的前 10 个记录，使用传统的编程语言，可以通过一个循环取出前 10 个记录。但使用标准的 SQL 查询，这是不可能实现的。从 SQL 的角度来说，在一个表中不存在“前 10 个记录”这样的概念。

其实，SQL 的这个特点不仅不是个限制，反而是其长处。因为 SQL 不根据位置来读取记录，且可快速读取记录。

综上所述，SQL 有两个特点：所有数据存储在表中；从 SQL 的角度来说，表中的记录没有顺序。

2．使用 SQL 从表中读取记录

SQL 的主要功能之一是实现数据库查询。如果读者熟悉 Internet 搜索引擎，那么对查询已经非常熟悉了。在 Internet 上，搜索引擎使得我们能够查询到满足特定条件的信息。

例如，如果想找到有 ASP 信息的全部站点，可以连接到 Yahoo 网站并执行一个对表达式 Active Sever Pages 的搜索。我们会收到一个列表，表中包括所有其描述中包含 Active Sever Pages 的站点。

多数 Internet 搜索引擎允许逻辑查询。在逻辑查询中，可以包括特殊的运算符如 AND、OR 和 NOT，以选择特定的记录。例如，可以用 AND 来限制查询结果。如果执行一个对表达式 Active Server Pages AND SQL 的搜索，将得到其描述中同时包含 Active Server Pages 和 SQL 的记录。

如果需要扩展查询的结果，可以使用逻辑操作符 OR。例如，如果执行一个对表达式 Active Server Pages OR SQL 的搜索，所收到的列表中将包括所有其描述中同时包含两个表达式或其中任何一个表达式的站点。

如果想从搜索结果中排除特定的站点，可以使用 NOT。例如，查询 Active Server Pages AND NOT SQL 将返回一个列表，列表中的站点包含 Active Server Pages，但不包含 SQL。当必须排除特定的记录时，可以使用 NOT。

用 SQL 执行的查询与用 Internet 搜索引擎执行的搜索非常相似。当执行一个 SQL 查询时，通过使用包括逻辑运算符的查询条件，可以得到一个记录列表。此时查询结果是来自一个或多个表。

SQL 查询的语法非常简单。假设有一个名为 email_table 的表，包含名字和地址两个字段，要得到 Bill Gates 的 Email 地址（保存在 email 字段中），可以使用下面的查询：

```
SELECT email from email_table WHERE name="Bill Gates"
```

当这个查询执行时，就从名为 email_table 的表中读取 Bill Gates 的 Email 地址。这个简单的语句包括 3 部分：

第 1 部分指名要选取的列。在此例中，只有 email 列被选取。当执行时，只显示 email 列的值 billg@microsoft.com。

第 2 部分指明要从哪个（些）表中查询数据。在此例中，要查询的表名为 email_table。

最后，SELECT 语句的 WHERE 子句指明要选择满足什么条件的记录。在此例中，查询条件为只有 name 列的值为 Bill Gates 的记录才被选取。

Bill Gates 很有可能拥有不止一个 Email 地址。如果表中包含 Bill Gates 的多个 Email 地址，用上述的 SELECT 语句可以读取他所有的 Email 地址。SELECT 语句从表中取出所有 name 字段值为 Bill Gates 的记录的 email 字段的值。

前面说过，查询条件中可以包含逻辑运算符。假如想读取 Bill Gates 或 Clinton 总统的所有 Email 地址，可以使用下面的查询语句：

```
SELECT email FROM email_table WHERE name="Bill Gates"
  OR name="president Clinton"
```

此例中的查询条件比前一个复杂了一点。这个语句从表 email_table 中选出所有 name 列为 Bill Gates 或 Clinton 的记录。如果表中含有 Bill Gates 或 Clinton 的多个地址，所有的地址都被读取。

SELECT 语句的结构看起来很直观。如果从一个表中选择一组记录，SQL SELECT 语句的形式如下：

```
SELECT  特定的列 FROM 一个表 WHERE  某些列满足一个特定的条件。
```

3．操作多个表

到现在为止，只尝试了用一条 SQL 查询语句从一个表中取出数据。也可以用一个 SELECT 语句同时从多个表中取出数据，只需在 SELECT 语句的 FROM 从句中列出要从中取出数据的表名称即可，如下所示：

```
SELECT au_lname,title FROM authors,titles
```

这个 SELECT 语句执行时，同时从表 authors 和表 titles 中取出数据。从表 authors 中取出所有的作者名字，从表 titles 中取出所有的书名。在 ISQL/w 程序中执行这个查询后会发现，查询结果中作者的名字并没有和他们所著的书相匹配，而是出现了作者名字和书名的所有可能的组合，这也许不是所希望见到的。

问题在于没有指明这两个表之间的关系。以上的 SELECT 语句没有通过任何方式告诉 SQL 如何把表和表关联在一起。由于不知道如何关联两个表，服务器只能简单地返回取自两个表中的记录的所有可能组合。

要从两个表中选出有意义的记录组合，需要通过建立两表中字段的关系来关联两个表。要做到这一点的途径之一是创建第 3 个表，专门用来描述另外两个表的字段之间的关系。

表 authors 有一个名为 au_id 的字段，该字段包含有每个作者的唯一标识。表 titles 有一个名为 title_id 的字段，包含每个书名的惟一标识。如果能在字段 au_id 和字段 title_id 之间

建立一个关系，就可以关联这两个表。样例数据库 pubs 中有一个名为 titleauthor 的表，正是用来完成这个工作的。表中的每个记录包括两个字段，用来把表 titles 和表 authors 关联在一起。下面的 SELECT 语句使用了这 3 个表以得到正确的结果。

```
SELECT au_name,title FROM authors,titles,titleauthor
  WHERE authors.au_id=titleauthor.au_id AND
  titles.title_id=titleauthor.title_id
```

当这个 SELECT 语句执行时，每个作者都将与正确的书名相匹配。表 titleauthor 指明了表 authors 和表 titles 的关系，它通过包含分别来自两个表的一个字段实现这一点。第 3 个表的惟一目的是在另外两个表的字段之间建立关系。它本身不包含任何附加数据。

注意在这个例子中字段名是如何书写的。为了区别表 authors 和表 titles 中相同的字段名 au_id，每个字段名前面都加上了表名前缀和一个句号。名为 author.au_id 的字段属于表 authors，名为 titleauthor.au_id 的字段属于表 titleauthor，两者不会混淆。

通过使用第 3 个表，可以在两个表的字段之间建立各种类型的关系。例如，一个作者也许写了许多不同的书，或者一本书也许由许多不同的作者共同完成。当两个表的字段之间有这种“多对多”的关系时，需要使用第 3 个表来指明这种关系。

多数情况下，两个表之间的关系并不复杂。比如需要指明表 titles 和表 publishers 之间的关系，因为一个书名不可能与多个出版商相匹配，不需要通过第 3 个表来指明这两个表之间的关系。要指明表 titles 和表 publishers 之间的关系，只要让这两个表有一个公共的字段就可以了。在数据库 pubs 中，表 titles 和表 publishers 都有一个名为 pub_id 的字段。如果想得到书名及其出版商的一个列表，可以使用如下的语句：

```
SELECT title,pub_name FROM titles,publishers WHERE
  titles.pub_id=publishers.pub_id
```

当然，如果一本书是由两个出版商联合出版的，那么需要第 3 个表来代表这种关系。

通常，当预先知道两个表的字段间存在“多对多”关系时，就使用第 3 个表来关联这两个表。反之，如果两个表的字段间只有“一对一”或“一对多”关系，则使用公共字段来关联它们。

4．操作字段

通常，从一个表中取出字段值时，该值与创建该表时所定义的字段名联系在一起。如果从表 authors 中选择所有的作者名字，所有的值将会与字段名 au_lname 相联系。但是在某些情况下，需要对字段名进行操作。在 SELECT 语句中，可以在默认字段名后面仅跟一个新名字来取代它。例如，可以用一个更直观易读的名字 Author Last Name 来代替字段名 au_lname，如下所示：

```
SELECT au_lname "Author Last Name" FROM authors
```

当这个 SELECT 语句执行时，来自字段 au_lname 的值会与 Author Last Name 相联系。查询结果可能是这样的：

```
Author Last Name
……
White
Green
Carson
O'Leary
Straight
…
(23 row(s) affected)
```

注意：字段标题不再是 au_lname**，而是被** Author Last Name **所取代。**

也可以操作从一个表返回的字段值。例如，如果要把表 titles **中的所有书的价格加倍，可以使用下面的** SELECT **语句：**

```
SELECT price*2 FROM titles
```

当这个查询执行时，每本书的价格从表中取出时都会加倍。但是，通过这种途径操作字段不会改变存储在表中的书价。对字段的运算只会影响 SELECT **语句的输出，而不会影响表中的数据。为了同时显示书的原始价格和涨价后的新价格，可以使用下面的查询：**

```
SELECT price "Original price",price*2 "New price" FROM titles
```

当数据从表 titles **中取出时，原始价格显示在标题** Original price **下面，加倍后的价格显示在标题** New price **下面。结果可能是这样的：**

```
original price new price
……
39.98
11.95
23.90
5.98
39.98
…
(18 row(s) affected)
```

可以使用大多数标准的数学运算符来操作字段值，如加（+）、减（-）、乘（*）和除（/）。也可以一次对多个字段进行运算，例如：

```
SELECT price*ytd_sales "total revenue" FROM titles
```

在这个例子中，通过把价格与销售量相乘，计算出了每种书的总销售额。这个 SELECT **语句的结果将是这样的：**

```
total revenue
……
```

```
81,859,05
46,318,20
55,978,78
81,859,05
40,619,68
…
(18 row(s) affected)
```

最后，还可以使用连接运算符（它看起来像个加号）来连接两个字符型字段，例如：

```
SELECT au_fname+" "+au_lname "author name" FROM authors
```

在这个例子中，把字段 au_fname 和字段 au_lname 粘贴在一起，中间用一个逗号隔开，并把查询结果的标题指定为 author name。这个语句的执行结果将是这样的：

```
author names
……
Johnson White
Marjorie Green
Cheryl Carson
Michael O'Leary
Dean Straight
…
(23 row(s) affected)
```

可以看到，SQL 提供了对查询结果的许多控制。在 ASP 编程中，使用 SQL 来操作查询结果几乎总是比使用有同样作用的脚本效率更高。

5．排序查询结果

如前所述，SQL 表没有内在的顺序。例如，从一个表中取第 2 个记录是没有意义的。从 SQL 的角度看来，没有一个记录在任何其他记录之前。然而，可以操纵一个 SQL 查询结果的顺序。默认情况下，当记录从表中取出时，记录不以特定的顺序出现。例如，当从表 authors 中取出字段 au_lname 时，查询结果显示将是这样的：

```
au_lname
……
White
Green
Carson
O'Leary
Straight
…
(23 row(s) affected)
```

看一列没有特定顺序的名字是很不方便的。如果把这些名字按字母顺序排列，读起来

就会容易得多。通过使用 ORDER BY 子句，可以强制一个查询结果按升序排列，如下所示：

```
SELECT au_lname FROM authors ORDER BY au_lname
```

当这个 SELECT 语句执行时，作者名字的显示将按字母顺序排列。ORDER BY 子句将作者名字按升序排列。

也可以同时对多个列使用 ORDER BY 子句。例如，如果想同时按升序显示字段 au_lname 和字段 au_fname，需要对两个字段都进行排序，如下所示：

```
SELECT au_lname,au_fname FROM authors ORDER BY au_lname,au_fname
```

这个查询首先把结果按 au_lname 字段进行排序，然后按字段 au_fname 排序。记录将按如下的顺序取出：

```
au_lname    au_fname
……
Bennet      Abraham
Ringer      Albert
Ringer      Anne
Smith       Meander
…
(23 row(s) affected)
```

注意有两个作者有相同的名字 Ringer。名为 Albert Ringer 的作者出现在名为 Anne Ringer 的作者之前，这是因为姓 Albert 按字母顺序应排在姓 Anne 之前。

如果想把查询结果按相反的顺序排列，可以使用关键字 DESC。关键字 DESC 把查询结果按降序排列，如下例所示：

```
SELECT au_lname,au_fname FROM authors
  WHERE au_lname="Ringer" ORDER BY au_lname,au_fname DESC
```

这个查询从表 authors 中取出所有名字为 Ringer 的作者的记录。ORDER BY 子句根据作者的名字和姓将查询结果按降序排列。查询结果将是这样的：

```
au_lname au_fname
……
Ringer Anne
Ringer Albert
(2 row(s) affectec)
```

注意在这个表中，姓 Anne 出现在姓 Albert 之前，作者名字按降序显示。也可以按数值型字段对一个查询结果进行排序。例如，如果想按降序取出所有书的价格，可以使用如下的 SQL 查询：

```
SELECT price FROM titles ORDER BY price DESC
```

这个 SELECT 语句从表中取出所有书的价格，显示结果时，价格低的书先显示，价格高的书后显示。

警告：不是特别需要，不要对查询结果进行排序，因为服务器完成这项工作需要额外的开销。这意味着带有 ORDER BY 子句的 SELECT 语句执行起来比一般的 SELECT 语句花的时间长。

6．读取出互不相同的记录

一个表有可能在同一列中有重复的值。例如，数据库 pubs 的表 authors 中有两个作者的名字是 Ringer。如果从这个表中取出所有的名字，名字 Ringer 将会显示两次。

在特定情况下，可能只有兴趣从一个表中取出互不相同的值。如果一个字段有重复的值，也许希望每个值只被选取一次，可以使用关键字 DISTINCT 来做到这一点，如下例所示：

```
SELCET DISTINCT au_lname FROM authors WHERE au_lname="Ringer"
```

当这个 SELECT 语句执行时，只返回一条记录。通过在 SELECT 语句中包含关键字 DISTINCT，可以删除所有重复的值。例如，假设有一个关于新闻组信息发布的表，要想取出所有曾在这个新闻组中发布信息的人的名字，那么可以使用关键字 DISTINCT。每个用户的名字只取一次——尽管有的用户发布了不止一条信息。

警告：如同 ORDER BY 子句一样，强制服务器返回互不相同的值也会增加运行开销，服务器不得不花费一些时间来完成这项工作。因此，除非有特别的必要，尽量不要使用关键字 DISTINCT。

7．插入数据

向表中添加一个新记录，要使用 SQL INSERT 语句。如下例所示：

```
INSERT mytable (mycolumn) VALUES ("some data")
```

这个语句把字符串"some data"插入表 mytable 的 mycolumn 字段中。将要被插入数据的字段的名字在第 1 个括号中指定，实际的数据在第 2 个括号中给出。

INSERT 语句的完整语法如下：

```
INSERT [INTO] {table_name|view_name} [(column_list)] {DEFAULT VALUES
  |values_list|select_statement}
```

如果一个表有多个字段，通过把字段名和字段值用逗号隔开，可以向所有的字段中插入数据。假设表 mytable 有 3 个字段：first_column、second_column 和 third_column。下面的 INSERT 语句添加了一条 3 个字段都有值的完整记录：

```
INSERT mytable (first_column,second_column,third_column)
  VALUES ("some data","some more data","yet more data")
```

注意：可以使用 INSERT 语句向文本型字段中插入数据。但是，如果需要输入很长

的字符串，应该使用 WRITETEXT 语句。这部分内容已超出了本书的讨论范围。要了解更多的信息，请参考 Microsoft SQL Server 的文档。

如果在 INSERT 语句中只指定两个字段和数据会怎么样呢？换句话说，向表中插入一条新记录，但有一个字段没有提供数据。在这种情况下，有下面的 4 种可能：

- 如果该字段有一个默认值，该值会被使用。例如，假设插入新记录时没有给字段 third_column 提供数据，而这个字段有一个默认值"some value"。在这种情况下，当新记录建立时会插入值"some value"。
- 如果该字段可以接受空值，而且没有默认值，则会被插入空值。
- 如果该字段不能接受空值，而且没有默认值，就会出现错误。会收到错误信息：The column in table mytable may not be null.
- 如果该字段是一个标识字段，那么它会自动产生一个新值。当向一个有标识字段的表中插入新记录时，只要忽略该字段，标识字段就会给自己赋一个新值。

提示：向一个有标识字段的表中插入新记录后，可以用 SQL 变量@@identity 来访问新记录的标识。字段的值。考虑如下的 SQL 语句：

```
INSERT mytable (first_column) VALUES("some value")
INSERT anothertable(another_first,another_second) VALUES(@@identity,"some value")
```

如果表 mytable 有一个标识字段，该字段的值会被插入表 anothertable 的 another_first 字段。这是因为变量@@identity 总是保存最后一次插入标识字段的值。字段 another_first 应该与字段 first_column 有相同的数据类型。但是，字段 another_first 不能是应该标识的字段。Another_first 字段用来保存字段 first_column 的值。

8．删除记录

要从表中删除一个或多个记录，需要使用 SQL DELETE 语句。可以给 DELETE 语句提供 WHERE 子句。WHERE 子句用来选择要删除的记录。例如，下面的这个 DELETE 语句只删除字段 first_column 的值等于"Delete Me"的记录。

```
DELETE mytable WHERE first_column="Delete Me"
```

DELETE 语句的完整语法如下：

```
DELETE [FROM] {table_name|view_name} [WHERE clause]
```

在 SQL SELECT 语句中可以使用的任何条件都可以在 DELECT 语句的 WHERE 子句中使用。例如，下面的这条 DELETE 语句只删除那些 first_column 字段的值为"goodbye"或 second_column 字段的值为"so long"的记录。

```
DELETE mytable WHERE first_column="goodby" OR second_column="so long"
```

如果不给 DELETE 语句提供 WHERE 子句，则表中的所有记录都将被删除。不应该有

这种想法。如果想删除表中的所有记录，应使用 TRUNCATE TABLE 语句。

注意：当使用 TRUNCATE TABLE 语句时，记录的删除是不作记录的。这意味着 TRUNCATE TABLE 要比 DELETE 快得多。

9．更新记录

要修改表中已经存在的一条或多条记录，应使用 SQL UPDATE 语句。同 DELETE 语句一样，UPDATE 语句可以使用 WHERE 子句来选择更新特定的记录，如下例所示：

```
UPDATE mytable SET first_column="Updated!" WHERE second_column="Update Me!"
```

这个 UPDATE 语句更新所有 second_column 字段的值为"Update Me!"的记录。对所有被选中的记录，字段 first_column 的值被置为"Updated!"。

下面是 UPDATE 语句的完整语法：

```
UPDATE {table_name|view_name} SET [{table_name|view_name}]
  {column_list|variable_list|variable_and_column_list}
  [,{column_list2|variable_list2|variable_and_column_list2}…
  [,{column_listN|variable_listN|variable_and_column_listN}]]
  [WHERE clause]
```

注意：可以对文本型字段使用 UPDATE 语句。但是，如果需要更新很长的字符串，应使用 UPDATETEXT 语句。要了解更多的信息，请参考 Microsoft SQL Server 的文档。

如果不提供 WHERE 子句，则表中的所有记录都将被更新。例如，如果想把表 titles 中的所有书的价格加倍，可以使用如下的 UPDATE 语句：

```
UPDATE mytable SET price=price*2
```

也可以同时更新多个字段。例如，下面的 UPDATE 语句同时更新 first_column、second_column 和 third_column 这 3 个字段。

```
UPDATE mytable SET first_column="Updated!" Second_column="Updated!"
  Third_column="Updated!" WHERE first_column="Update Me!"
```

提示：SQL 忽略语句中多余的空格。可以把 SQL 语句写成任何易读的格式。

10．用 SELECT 语句创建记录和表

读者也许已经注意到，INSERT 语句与 DELETE 语句及 UPDATE 语句有一点不同：它一次只操作一个记录。然而，有一个方法可以使 INSERT 语句一次添加多个记录。要做到这一点，需要把 INSERT 语句与 SELECT 语句结合起来，如下例所示：

```
INSERT mytable (first_column,second_column) SELECT
  another_first,another_second
  FROM anothertable WHERE another_first="Copy Me!"
```

该语句从表 anothertable 拷贝记录到表 mytable。表 anothertable 中只有字段 another_first 的值为"Copy Me！"的记录才被拷贝。

当为一个表中的记录建立备份时，这种形式的 INSERT 语句是非常有用的。在删除一个表中的记录之前，可以先用这种方法把它们拷贝到另一个表中。

如果需要拷贝整个表，可以使用 SELECT INTO 语句。例如，下面的语句创建了一个名为 newtable 的新表，该表包含表 mytable 的所有数据。

```
SELECT * INTO newtable FROM mytable
```

也可以指定特定的字段来创建这个新表。要做到这一点，只需在字段列表中指定想要拷贝的字段。另外，可以使用 WHERE 子句来筛选拷贝到新表中的记录。下面的例子只拷贝字段 second_columnd 的值等于"Copy Me!"的记录的 first_column 字段。

```
SELECT first_column INTO newtable FROM mytable
  WHERE second_column="Copy Me!"
```

使用 SQL 修改已经建立的表是很困难的。例如，如果向一个表中添加了一个字段，没有容易的办法再将它删除。另外，如果在创建一个表时不小心把一个字段的数据类型给弄错了，将没有办法改变它。但是，使用 SELECT 语句，可以解决这两个问题。

例如，假设想从一个表中删除一个字段，使用 SELECT INTO 语句，可以创建该表的一个拷贝，但不包含要删除的字段。这样既删除了该字段，又保留了不想删除的数据。

如果想改变一个字段的数据类型，可以创建一个包含正确数据类型字段的新表。创建好该表后，就可以结合使用 UPDATE 语句和 SELECT 语句，把原来表中的所有数据拷贝到新表中。通过这种方法，既可以修改表的结构，又能保存原有的数据。

11．使用集合函数

到现在为止，只学习了如何根据特定的条件从表中取出一条或多条记录。但是，有时候需要对一个表中的记录进行数据统计。例如，统计存储在表中的一次民意测验的投票结果，或者想知道一个访问者在站点上平均花费了多少时间。要对表中的任何类型的数据进行统计，都需要使用集合函数。

Microsoft SQL 支持 5 种类型的集合函数。可以统计记录数目、平均值、最小值、最大值或者求和。当使用一个集合函数时，它只返回一个数，该数值代表这几个统计值之一。

注意：要在 ASP 页中使用集合函数的返回值，需要给该值起一个名字。要做到这一点，可以在 SELECT 语句中，在集合函数后面紧跟一个字段名，如下例所示：

```
SELECT AVG(vote) "the_average" FROM opinion
```

在这个例子中，vote 的平均值被命名为 the_average。现在就可以在 ASP 页的数据库方法中使用这个名字了。

（1）统计字段值的数目。

函数 COUNT 也许是最有用的集合函数。可以用这个函数来统计一个表中某个字段的值的数目，如下例所示：

```
SELECT COUNT(au_lname) FROM authors
```

这个例子计算表 authors 中名字（last name）的数目。如果相同的名字出现了不止一次，该名字将会被计算多次。如果想知道名字为某个特定值的作者有多少个，可以使用 WHERE 子句，如下所示：

```
SELECT COUNT(au_lname) FROM authors WHERE au_lname="Ringer"
```

这个例子返回名字为 Ringer 的作者的数目。如果这个名字在表 authors 中出现了两次，则此函数的返回值是 2。

假如想知道有不同名字的作者的数目，可以通过使用关键字 DISTINCT 来得到该数目，如下例所示：

```
SELECT COUNT(DISTINCT au_lname) FROM authors
```

如果名字 Ringer 出现了不止一次，它将只被计算一次。关键字 DISTINCT 决定了只有互不相同的值才被计算。

通常，当使用 COUNT 函数时，字段中的空值将被忽略。一般来说，这正是所希望的。但是，如果仅仅想知道表中记录的数目，那么需要计算表中所有的记录——不管它是否包含空值，如下例所示：

```
SELECT COUNT(*) FROM authors
```

注意函数 COUNT 没有指定任何字段。这个语句计算表中所有记录的数目，包括有空值的记录。因此，不需要指定要被计算的特定字段。

函数 COUNT 在很多情况下是有用的。例如，假设有一个表保存了对站点的质量进行民意调查的结果。这个表有一个名为 vote 的字段，该字段的值要么是 0，要么是 1。0 表示反对票，1 表示赞成票。要统计赞成票的数量，可以使用下面的 SELECT 语句：

```
SELECT COUNT(vote) FROM opinion_table WHERE vote=1
```

（2）计算字段的平均值。

使用函数 COUNT 可以统计一个字段中有多少个值，但有时需要计算这些值的平均值。使用函数 AVG，可以返回一个字段中所有值的平均值。

假如要对站点进行一次较为复杂的民意调查，访问者可以在 1 到 10 之间投票，表示他们喜欢站点的程度。投票结果保存在名为 vote 的 INT 型字段中。要计算用户投票的平均值，需要使用函数 AVG，如下所示：

```
SELECT AVG(vote) FROM opinion
```

这个 SELECT 语句的返回值代表用户对站点的平均喜欢程度。函数 AVG 只能对数值型字段使用。这个函数在计算平均值时也忽略空值。

（3）计算字段值的和。

假设站点被用来出售卡片，已经运行了两个月，是该计算赚了多少钱的时候了。假设

有一个名为 orders 的表用来记录所有访问者的定购信息。要计算所有定购量的总和，可以使用函数 SUM，如下例所示：

```
SELECT SUM(purchase_amount) FROM orders
```

函数 SUM 的返回值代表字段 purchase_amount 中所有值的平均值。字段 purchase_amount 的数据类型也许是 MONEY 型，但也可以对其他数值型字段使用函数 SUM。

（4）返回最大值或最小值。

再一次假设有一个表用来保存对站点进行民意调查的结果。访问者可以选择从 1 到 10 的值来表示他们对站点的评价。如果想知道访问者对站点的最高评价，可以使用如下的语句：

```
SELECT MAX(vote) FROM opinion
```

也许希望有人对站点给予了很高的评价。通过函数 MAX，可以知道一个数值型字段的所有值中的最大值。如果有人对站点投了数字 10，函数 MAX 将返回 10。

另一方面，假如想知道访问者对站点的的最低评价，可以使用函数 MIN，如下例所示：

```
SELECT MIN(vote) FROM opinion
```

函数 MIN 返回一个字段的所有值中的最小值。如果字段是空的，函数 MIN 返回空值。

12．通过匹配一定范围的值来读取出数据

假设有一个表用来保存对站点进行民意调查的结果。现在想向所有对站点的评价在 7 到 10 之间的访问者发送书面感谢信。要得到这些人的名字，可以使用如下的 SELECT 语句：

```
SELECT username FROM opinion WHERE vote>6 and vote<11
```

使用下面的 SELECT 语句也可以得到同样的结果：

```
SELECT username FROM opinion WHERE vote BETWEEN 7 AND 10
```

这个 SELECT 语句与上一个语句是等价的。使用哪一种语句是编程风格的问题，但使用表达式 BETWEEN 的语句更易读。

现在假设只想取出对站点投了 1 或者 10 的访问者的名字。要从表 opinion 中取出这些名字，可以使用如下的 SELECT 语句：

```
SELECT username FROM opinion WHERE vote=1 or vote=10
```

但是，存在一种等价的方式。使用如下的 SELECT 可以得到相同的结果：

```
SELECT username FROM opinion WHERE vote IN (1,10)
```

这条 SELECT 语句只取出 vote 的值等于括号中的值之一的记录，注意表达式 IN 的使用。

也可以使用 IN 来匹配字符数据。例如，假设只想取出 Bill Gates 或 Clinton 的投票值。可以使用如下的 SELECT 语句：

```
SELECT vote FROM opinion WHERE username
  IN ("Bill Gates","President Clinton")
```

最后，可以在使用 BETWEEN 或 IN 的同时使用表达式 NOT。例如，要取出那些投票值不在 7 到 10 之间的人的名字，可以使用如下的 SELECT 语句：

```
SELECT username FROM opinion WHERE vote NOT BETWEEN 7 and 10
```

要选取那些某个字段的值不在一列值之中的记录，可以同时使用 NOT 和 IN，如下例所示：

```
SELECT vote FROM opinion WHERE username
  NOT IN ("Bill Gates","President Clinton")
```

不是必须在 SQL 语句中使用 BETWEEN 或 IN，但是，要使查询更接近自然语言，这两个表达式是很有帮助的。

13．转换数据

SQL Server 足够强大，可以在需要的时候把大部分数值从一种类型转换为另一种类型。例如，要比较 SMALLINT 类型和 INT 类型数据的大小，不需要进行显式的类型转换，SQL Server 会为完成这项工作。但是，当想在字符型数据和其他类型的数据之间进行转换时，却需要自己进行转换操作。例如，假设想从一个 MONEY 类型的字段中取出所有的值，并在结果后面加上字符串"US Dollars"。需要使用函数 CONVERT，如下例所示：

```
SELECT CONVERT(CHAR(8),price)+"US Dollars" FROM orders
```

函数 CONVERT 带有两个变量。第 1 个变量指定了数据类型和长度。第 2 个变量指定了要进行转换的字段。在这个例子中，字段 price 被转换成长度为 8 个字符的 CHAR 类型的字段。字段 price 要被转换成字符类型才可以在它后面连接上字符串"US Dollars"。

当向 BIT，DATETIME，INT，或者 NUMERIC 类型字段添加字符串时，需要进行同样的转换操作。例如，下面的语句在一个 SELECT 语句的查询结果中加入字符串"The vote is"，该 SELECT 语句返回一个 BIT 型字段的值。

```
SELECT "The vote is"+CONVERT(CHAR(1),vote) FROM opinion
```

下面是这个语句的结果示例：

```
The vote is 1
The vote is 1
The vote is 0
(3 row(s) affected)
```

如果不进行显式的转换，会收到如下的错误信息：

```
Implicit conversion from datatype "varchar" to "bit" is not allowec.
Use the CONVERT function to run this query.
```

14．操作字符串数据

SQL Server 有许多函数和表达式，可以用来对字符串进行各种操作，包括各种各样的模式匹配和字符转换。在这一节小中，将学习如何使用最重要的字符串处理函数和表达式。

（1）匹配通配符。

假设想建立一个与 Yahoo 功能相似的 Internet 目录，可以建立一个表用来保存一系列的站点名称，统一资源定位器（URL）、描述和类别，并允许访问者通过在 HTML 表单中输入关键字来检索这些内容。

假如有一个访问者想从这个目录中得到其描述中包含关键字“ trading card ”的站点的列表。要取出正确的站点列表，也许试图使用这样的查询：

```
SELECT site_name FROM site_directory WHERE site_desc="trading card"
```

这个查询可以工作。但是，它只能返回那些其描述中只有 trading card 这个字符串的站点。例如，一个描述为 We have the greatest collection of trading cards in the world!的站点不会被返回。

要把一个字符串与另一个字符串的一部分相匹配，需要使用通配符。使用通配符和关键字 LIKE 来实现模式匹配。下面的语句使用通配符和关键字 LIKE 重写了上面的查询，以返回所有正确站点的名字：

```
SELECT SITE_name FROM site_directory
  WHERE site_desc LIKE "%trading cark%"
```

在这个例子中，所有其描述中包含表达式 trading card 的站点都被返回。描述为 We have the greatest collection of trading cards in the world!的站点也被返回。当然，如果一个站点的描述中包含 I am trading cardboard boxes online，该站点的名字也被返回。

注意本例中百分号（%）的使用。百分号是通配符之一。它代表 0 个或多个字符。通过把 trading card 包含在百分号中，所有其中嵌有字符串 trading card 的字符串都被匹配。

现在，假设站点目录变得太大而不能在一页中完全显示，决定把目录分成两部分。在第 1 页，显示所有首字母在 A 到 M 之间的站点。在第 2 页，显示所有首字母在 N 到 Z 之间的站点。要得到第 1 页的站点列表，可以使用如下的 SQL 语句：

```
SELECT site_name FROM site_directory WHERE site_name LIKE "[A-M]%"
```

这个例子使用了表达式[A-M]，只取出那些首字母在 A 到 M 之间的站点。中括号（[]）用来匹配处在指定范围内的单个字符。要得到第 2 页中显示的站点，应使用如下语句：

```
SELECT site_name FROM site_directory WHERE site_name LIKE "[N-Z]%"
```

在这个例子中，括号中的表达式代表任何处在 N 到 Z 之间的单个字符。

假设站点目录变得更大了，现在需要把目录分成更多页。如果想显示那些以 A、B 或

C 开头的站点，可以用下面的查询来实现：

```
SELECT site_name FROM site_directory WHERE site_name LIKE "[ABC]%"
```

在这个例子中，括号中的表达式不再指定一个范围，而是给出了一些字符。以这些字符中的任一个开头的所有站点都将被返回。

通过在括号内的表达式中同时包含一个范围和一些指定的字符，可以把这两种方法结合起来。例如，用下面的这个查询，可以取出那些首字母在 C 到 F 之间，或者以字母 Y 开头的所有站点：

```
SELECT site_name FROM site_directory WHERE site_name LIKE "[C-FY]%"
```

在这个例子中，名字为 Collegescape 和 Yahoo 的站点会被选取，而名字为 Magicw3 的站点则不会被选取。

也可以使用脱字符（^）来排除特定的字符。例如，要得到那些名字不以 Y 开头的站点，可以使用如下的查询：

```
SELECT site_name FROM site_directory WHERE site_name LIKE "[^Y]%"
```

对给定的字符或字符范围均可以使用脱字符。

最后，通过使用下划线字符（_），可以匹配任何单个字符。例如，下面这个查询返回每一个其名字的第 2 个字符为任何字母的站点：

```
SELECT site_name FROM site_directory WHERE site_name LIKE "M_crosoft"
```

这个例子既返回名为 Microsoft 的站点，也返回名为 Macrosoft 的站点。但是，名字为 Moocrosoft 的站点则不被返回。与通配符%不同，下划线只代表单个字符。

注意：如果想匹配百分号或下划线字符本身，需要把它们括在方括号中。如果想匹配连字符（-），应把它指定为方括号中的第 1 个字符。如果想匹配方括号，应把它们也括在方括号中。例如，下面的语句返回所有其描述中包含百分号的站点：

```
SELECT site_name FROM site_directory WHERE site_desc LIKE "%[%]%"
```

（2）匹配发音。

Microsoft SQL 有两个允许按照发音来匹配字符串的函数。函数 SOUNDEX 给一个字符串分配一个音标码，函数 DIFFERENCE 按照发音比较两个字符串。当不知道一个名字的确切拼写，但多少知道一点它的发音时，使用这两个函数将有助于取出该记录。

例如，如果建立了一个 Internet 目录，也许想增加一个选项，允许访问者按照站点名的发音来搜索站点，而不是按名字的拼写。考虑如下的语句：

```
SELECT site_name FROM site_directory
  WHERE DIFFERENCE(site_name,'Microsoft'>3
```

这个语句使用函数 DEFFERENCE 来取得其名字的发音与 Microsoft 非常相似的站点。函数 DIFFERENCE 返回一个 0 到 4 之间的数字。如果该函数返回 4，表示发音非常相近，

如果该函数返回 0，说明这两个字符串的发音相差很大。

例如，上面的语句将返回站点名 Microsoft 和 Macrosoft。这两个名字的发音与 Microsoft 都很相似。如果把上一语句中的大于 3 改为大于 2，那么名为 Zicrosoft 和 Megasoft 的站点也将被返回。最后，如果只需要等级差别大于 1 即可，则名为 Picosoft 和 Minisoft 的站点也将被匹配。

要深入了解函数 DIFFERENCE 是如何工作的，可以用函数 SOUNDEX 来返回函数 DIFFERENCE 所使用的音标码，如下例所示：

```
SELECT site_name "site name",SOUNDEX(site_name) "sounds like"
```

这个语句选取字段 site_name 的所有数据及其音标码。下面是这个查询的结果：

```
site name sounds like
……
Yahoo Y000
Mahoo M000
Microsoft M262
Macrosoft M262
Minisoft M521
Microshoft M262
Zicrosoft Z262
Zaposoft Z121
Millisoft M421
Nanosoft N521
Megasoft M221
Picosoft P221
(12 row(s) affected)
```

如果仔细看一下音标码，会注意到音标码的第 1 个字母与字段值的第 1 个字母相同。例如，Yahoo 和 Mahoo 的音标码只有第 1 个字母不同。还可以发现 Microsoft 和 Macrosoft 的音标码完全相同。

函数 DIFFERENDE 比较两个字符串的第 1 个字母和所有的辅音字母。该函数忽略任何元音字母（包括 y），除非一个元音字母是一个字符串的第 1 个字母。

不幸的是，使用 SOUNDEX 和 DIFFERENCE 有一个缺陷：WHERE 子句中包含这两个函数的查询执行起来效果不好。因此，应该小心使用这两个函数。

（3）删除空格。

有两个函数：TTRIM 和 LTRIM，可以用来从字符串中删除空格。函数 LTRIM 删除一个字符串前面的所有空格；函数 RTRIM 去除一个字符串尾部的所有空格。以下为一个使用函数 RTRIM 的例子：

```
SELECT RTRIM(site_name) FROM site_directory
```

在这个例子中，如果任何一个站点的名字尾部有多余的空格，多余的空格将从查询结果中删去。

可以嵌套使用这两个函数，把一个字符串前后的空格同时删去，如下例所示：

```
SELECT LTRIM(RTRIM(site_name) FROM site_directory
```

在从 CHAR 类型字段中剪掉多余的空格时，这两个函数非常有用。记住，如果把一个字符串保存在 CHAR 类型的字段中，该字符串会被追加多余的空格，以匹配该字段的长度。用这两个函数，可以去掉无用的空格，从而解决这个问题。

关于 SQL 就简单介绍到这里。网站数据库的基础是数据库，但如何实现 Web 和数据库的交互则必须用到 ADO 对象，下一节将介绍 ADO 对象。

22.2　ADO 基础

ADO（ActiveX Data Objects）是一项容易使用并且可扩展的、用于将数据库访问添加到 Web 页的技术。可以通过 ADO 访问存储在服务器端的数据库或其他表格化数据结构中的信息。可以使用 ADO 去编写紧凑简明的脚本以便连接到 ODBC（Open Database Connectivity）兼容的数据库和 OLE DB 兼容的数据源。ADO 是对当前 Microsoft 所支持的数据库进行操作的最有效、最简单直接的方法，它是一种功能强大的数据访问编程模式，从而使得大部分数据源可编程的属性得以直接扩展到 ASP 页面上。

简单来说，一个对 Web 数据库的 HTTP 请求回应的流程如下：用户的浏览器向 Web 服务器发出读取 ASP 文件的要求，Web 服务器就执行 ASP 文件。如果有存取数据的操作，则通过 ADO，而 ADO 再通过 ODBC 来存取数据库。最后，Web 服务器再将结果返回给用户的浏览器，从而显示结果。整个流程如图 22.1 所示。

可以看出对 Web 数据库操作要通过 ADO 和 ODBC，这里首先介绍如何创建 DSN 文件。

1．创建 ODBC DSN 文件

在创建数据库脚本之前，必须提供一条使 ADO 定位、标识和与数据库通讯的途径。数据库驱动程序使用 DSN（Data Source Name，数据源名称）定位和标识特定的 ODBC 兼容数据库，将信息从 Web 应用程序传递给数据库。典型情况下，DSN 包含数据库配置、用户安全性和定位信息。

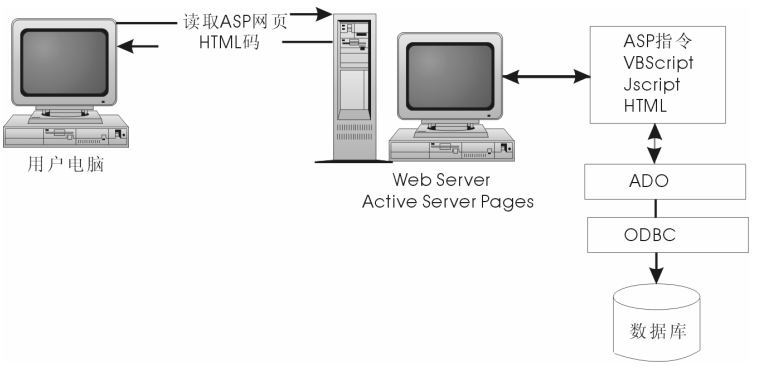

图 22.1　对 Web 数据库的 HTTP 请求回应的流程

通过 ODBC，可以选择希望创建的 DSN 的类型：用户、系统或文件 DSN。用户和系统 DSN 存储在 Windows NT 注册表中。系统 DSN 允许所有的用户登录到特定的服务器上去访问数据库，而用户 DSN 使用适当的安全身份证明限制特定用户到数据库的连接。文件 DSN 用于从文本文件中获取表格，提供了对多用户的访问，并且通过复制 DSN 文件，可以轻易地从一个服务器转移到另一个服务器。

创建文件 DSN 文件的步骤如下：

（1）在 Windows 的“开始”菜单打开“控制面板”。

（2）双击 ODBC 图标，然后选择“文件 DSN”属性页。

（3）单击“添加”按钮，选择数据库驱动程序。

（4）单击“下一步”按钮，按照后面的指示配置适用于数据库软件的文件 DSN，如图 22.2 所示。

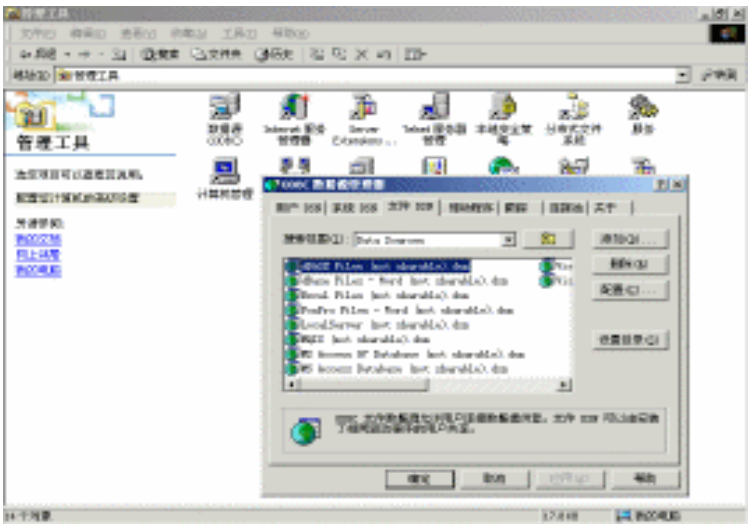

图 22.2 配置 ODBC 数据源窗口

2．配置 Microsoft Access 数据库的文件 DSN

配置 Microsoft Access 数据库的文件 DSN 的步骤如下：

（1）“创建新数据源”对话框中，从列表框选择“Microsoft Access Driver”。

（2）单击“下一步”按钮。输入 DSN 文件名，然后单击“下一步”按钮。

（3）单击“完成”按钮创建数据源。

（4）在“ODBC Microsoft Access 97 安装程序”对话框中，单击“选择”按钮。选择 Microsoft Access 数据库文件（*.mdb）。

（5）单击“确定”按钮完成配置。

注意：由于性能和可靠性的原因，推荐使用“客户-服务器数据库引擎”配置由这样一种由 Web 应用程序驱动的数据，这些 Web 应用程序必须满足 10 个以上的用户的同时访问。尽管 ASP 可以使用任何 ODBC 兼容的数据库，但它是为使用客户-服

务器数据库而设计的，而且经过了严格的测试，这些数据库包括 Microsoft SQL Server、Oracle 等。

ASP 支持共享文件数据库（如 Microsoft Access 或 Microsoft FoxPro）作为有效的数据源。尽管在 ASP 文档中的一些示例使用共享文件数据库，但建议只将此类数据库引擎用于开发或有限的配置方案。共享文件数据库可能无法很好地适用于可满足高需求、高质量的 Web 应用程序的客户-服务器数据库。

3．配置 SQL Server 数据库的文件 DSN

注意：如果数据库驻留在远程服务器上，请与服务器管理员联系，获取附加的配置信息；下面的过程使用 SQL Server 的 ODBC 默认的设置，它可能不适用于某些硬件配置。

（1）在“创建新数据源”对话框中，从列表框中选择“SQL Server”，然后单击“下一步”按钮。

（2）输入 DSN 文件的名称，然后单击“下一步”按钮。

（3）单击“完成”按钮创建数据源。

（4）输入运行 SQL 服务程序的服务器的名称、登录 ID 和密码。

（5）在“创建 SQL Server 的新数据源”对话框中，在“服务器”列表框中输入包含 SQL Server 数据库的服务器的名称，然后单击“下一步”按钮。

（6）选择验证登录 ID 的方式。

（7）如果要选择 SQL 服务器验证，请输入一个登录 ID 和密码，然后单击“下一步”按钮。

（8）在“创建 SQL Server 的新数据源”对话框中，设置默认数据库、存储过程设置的驱动程序和 ANSI 标识，然后单击“下一步”按钮（要获取详细信息，请单击“帮助”按钮）。

（9）在对话框（同样名为“创建 SQL Server 的新数据源”）中，选择一种字符转换方法，然后单击“下一步”按钮（要获取详细信息，请单击“帮助”按钮）。

（10）在下一个对话框（同样名为“创建 SQL Server 的新数据源”）中，选择登录设置。

注意：典型情况下，只能使用日志来调试数据库访问问题。

（11）在“ODBC Microsoft SQL Server 安装程序”对话框中，单击“测试数据源”按钮。如果 DSN 正确创建，“测试结果”对话框将指出测试成功完成。

4．SQL Server 连接和安全信息

如果正在开发用于连接远程 SQL Server 数据库的 ASP 数据库应用程序，应考虑以下问题：

- 连接方案。可以选择 TCP/IP 套接字和命名管道的方法访问远程的 SQL Server 数据库。当使用命名管道时，因为在建立连接之前，数据库用户必须被 Windows NT

确认，所以对只有适当的 SQL Server 访问身份而在该计算机上没有 Windows NT 用户帐号的用户可能会被拒绝访问命名管道。作为一种替代方案，使用 TCP/IP 套接字的连接可直接连接到数据库服务器，而不必通过使用命名管道的中间计算机。因为使用 TCP/IP 套接字可直接连接到数据库服务器，通过 SQL Server 的确认，用户就可以获得访问权，而不必通过 Windows NT 的确认。

注意：在连接到远程数据库时使用 TCP/IP 套接字可提高性能。

- 安全性。如果使用 SQL Server 的集成或混合安全特性，并且 SQL Server 数据库位于远程服务器上，则不能使用 Windows NT 请求/响应的确认。也就是说，不能将 Windows NT 请求/响应身份证转发到远程计算机上，而只能使用基本身份验证，它根据用户提供的用户名和口令进行。

有关这一主题的详细信息，请参阅 http://www.microsoft.com/sqlsupport/上的 Microsoft SQL Server 技术支持主页。

5．配置 Oracle 数据库文件 DSN

首先要确保 Oracle 软件被正确地安装在要创建 DSN 的计算机上。详细信息，请与服务器管理员联系或参阅数据库软件文档。

配置 Oracle 数据库文件 DSN 的步骤如下：

（1）在“创建新数据源”对话框中，从列表框中选择“Microsoft ODBC for Oracle”，然后单击“下一步”按钮。

（2）输入 DSN 文件的名称，然后单击“下一步”按钮。

（3）单击“完成”按钮创建数据源。

（4）输入用户名、密码和服务器名，然后单击“确定”按钮。

注意：DSN 文件用.dsn 扩展名，位于\Programs\Common Files\ODBC\Data Sources 目录中。有关创建 DSN 文件的详细信息，请访问 Microsoft ODBC Web 站点 http://microsoft.com/odbc/。

6．连接数据库

访问数据库信息的第 1 步是和数据库源建立连接。ADO 提供了 Connection 对象，可以使用该对象建立和管理应用程序与 ODBC 数据库之间的连接。Connection 对象具有各种属性和方法，可以使用它们打开和关闭数据库连接，并且发出查询请求来更新信息。

要建立数据库连接，首先应创建 Connection 对象的实例。例如，下面的脚本创建 Connection 对象，接着打开数据库连接。

```
<%
'创建一个连接对象
Set objCon=Server.CreateObject("ADODB.Connection")
'Open a connection,the string refers to the DSN
objCon.Open "FILEDSN=MyDatabase.dsn"
%>
```

注意：无论在等号（=）之前还是之后，DSN 字符串都不能包含空格。

在这种情况下，Connection 对象的 Open 方法引用基于 DSN 的文件，其中包含关于数据库的位置和配置信息。也可以不引用 DSN，直接显式引用供应程序、数据源、用户 ID 和密码。有关建立连接的可选方法的详细信息，请参阅 Microsoft ActiveX Data Objects（ADO）。

7．用 Connection 对象执行查询

用 Connection 对象的 Execute 方法，可以查询数据库源并检索结果。SQL 有许多命令可用来检索和更新信息。

在下面的示例中，脚本使用 Connection 对象的 Execute 方法向 SQL INSERT 语句所指定的表格中发出查询，该语句将数据插入特定的数据库表格。在下面的示例中，脚本将名称 Jose Lugo 插入名为 Customers 的数据库表中。

```
<%
'Define file based DSN
strDSN="FILEDSN=MyDatabase.dsn"
'Instantiate the Connection object and open a database connection
Set cn=Server.CreateObject("ADODB.Connection")
cn.Open strDSN
'Define SQL SELECT statement
strSQL="INSERT INTO Customers (FirstName,LastName) VALUES ('Jose','Lugo')"
'Use the Execute method to issue a SQL query to database
cn.Execute(strSQL)
%>
```

除了 SQL INSERT 以外，也可以使用 SQL UPDATE 和 DELETE 语句更改和删除数据库信息。

用 SQL UPDATE 可以改变数据库表中各项目值。例如，下面的脚本使用 UPDATE 语句将 Customers 表中每个 LastName 字段中包含姓 Smith 记录的 FirstName 字段的值更改为 Jeff。

```
<%
Set cn=Server.CreateObject("ADODB.Connection")
cn.Open "FILEDSN=MyDatabase.dsn"
cn.Execute "UPDATE Customers SET FirstName='Jeff' WHERE LastName='Smith'"
%>
```

要想从数据库表中删除特定的记录，可使用 SQL DELETE 语句。例如，下面的脚本从 Customers 表中删除了所有姓 Smith 的行。

```
<%
Set cn=Server.CreateObject("ADODB.Connection")
cn.Open "FILEDSN=MyDatabase.dsn"
```

```
cn.Execute "DELETE FROM Customers WHERE LastName='Smith'"
%>
```

注意 在使用 SQL DELETE 语句时，必须谨慎从事。当使用不带 WHERE 子句的 DELETE 语句时，它将删除表中的所有行。一定要包含 SQL WHERE 子句来指定要删除的确切行。

8．使用 Recordset 对象处理结果

尽管 Connection 对象简化了连接数据库和查询任务，但仍有许多不足。确切地说，检索和显示数据库信息的 Connection 对象不能用于创建脚本，必须确切知道要对数据库作出的更改，然后才能使用查询实现更改。

为此，ADO 提供了 Recordset 对象，用于检索数据、检查结果、更改数据库。正如它的名称所暗示的那样，Recordset 对象有许多可以使用的特性，可以根据查询的限制条件，检索并且显示一组数据库行，即记录。Recordset 对象保持查询返回的记录的位置，允许一次一项逐步扫描结果。

根据 Recordset 对象的指针类型属性的设置，可以滚动和更新记录。数据库指针可以在一组记录中定位到特定的项。指针还可以用于检索和检查记录，然后在这些记录的基础上执行操作。Recordset 对象有一些属性，可用于精确地控制指针的行为，提高检查和更新结果的能力。例如，可以使用 CursorType 和 CursorLocation 属性设置指针的类型，将结果返回给客户端应用程序（结果通常保留在数据库服务器上）并显示其他用户对数据库的最后一次更改。有关配置 Recordset 对象指针的信息，请参阅 Microsoft ActiveX Data Objects（ADO）。

9．检索记录

一个成功的数据库应用程序使用 Connection 对象建立连接并使用 Recordset 对象处理返回的数据。通过“协调”这两个对象的特定功能，可以开发出几乎可以执行任何数据处理任务的数据库应用程序。例如，下面的服务器端脚本使用 Recordset 对象执行 SQL SELECT 语句。SELECT 语句检索一组基于查询条件的信息。查询也包含 SQL WHERE 子句，用来缩小查询的范围。此例中，WHERE 子句将查询结果限制为 Customers 数据库表中所有包含姓 Smith 的记录。

```
<%
'Establish a connection with data source
strDSN="FILEDSN=MyDatabase.dsn"
Set cn=Server.CreateObject("ADODB.Connection")
cn.Open strDSN
'Instantiate a Recordset object
Set rsCustomers=Server.CreateObject("ADODB.Recordset")
'Open a recordset using the Open method
'and use the connection established by the Connection object
strSQL="SELECT FirstName, LastName FROM Customers WHERE LastName='Smith' "
rsCustomers.Open strSQL,cn
```

```
'Cycle through record set and display the results
'and increment record position with MoveNext method
Set objFirstName=rsCustomers("FirstName")
Set objLastName=rsCustomers("LastName")
Do Until rsCustomers.EOF
Response.Write objFirstName&" "&objLastName&"<BR>"
rsCustomers.MoveNext
Loop
%>
```

注意，在前面的例子中，用来建立数据库连接的 Connection **对象和** Recordset **对象使用该连接从数据库中检索结果。当需要精确地设置和数据库建立连接所采用的方式时，这个方法是非常有用的。例如，如果需要在连接尝试失败之前指定等待的时间，则需要使用** Connection **对象来设置属性。但是，如果仅仅想使用** ADO **默认的连接属性建立连接，则应该使用** Recordset **对象的** Open **方法去建立链接，如下例所示：**

```
<%
strDSN="FILEDSN=MyDatabase.dsn"
strSQL="SELECT FirstName,LastName FROM Customers WHERE LastName='Smith'"
Set rsCustomers=Server.CreateObject("ADODB.Recordset")
'Open a connection using the Open method
'and use the connection established by the Connection object
rsCustomers.Open strSQL,strDSN
'Cycle through the record set, display the results,
'and increment record position with MoveNext method
Set objFirstName=rsCustomers("FirstName")
Set objLastName=rsCustomers("LastName")
Do Until rsCustomers.EOF
Response.Write objFirstName&" "&objLastName&"<BR>"
rsCustomers.MoveNext
Loop
%>
```

当使用 Recordset **对象的** Open **方法建立一个连接时，必须使用** Connection **对象来保证链接的安全。详细信息请参阅** Microsoft ActiveX Data Objects（ADO）**。**

10．用 Command 对象改善查询

通过 ADO Command **对象，可以像用** Connection **对象和** Recordset **对象那样执行查询，惟一的不同在于，用** Command **对象可以在数据库源上准备、编译查询并且反复使用一组不同的值来发出查询。这种方式的编译查询的优点是可以最大程度地减少向现有查询重复发出修改请求所需的时间。另外，还可以在执行之前通过查询的可变部分的选项使** SQL **查询保持局部未定义。**

Command **对象的** Parameter **集合使不必在每次发出查询时重新建立查询。例如，如果**

需要有规律地更新基于库存清单的 Web 系统中的供应和价格信息，可以用下面的方法预先定义查询：

```
<%
'Open a connection using Connection object Command object
'does not have an Open method for establishing a connection
strDSN="FILEDSN=MyDatabase.dsn"
Set cn=Server.CreateObject("ADODB.Connection")
cn.Open strDSN
'Instantiate Command object; use ActiveConnection property to attach
'connection to Command object
Set cm=Server.CreateObject("ADODB.Command")
Set cm.ActiveConnection=cn
'Define SQL query
cm.CommandText="INSERT INTO Inventory (Material,Quantity) VALUES (?,?)"
'Save a prepared (or pre-compiled) version of the query specified in
CommandText
'property before a Command object's first execution.
cm.Prepared=True
'Define query parameter configuration information
cm.Parameters.Append cm.CreateParameter("material_type",200,,255)
cm.Parameters.Append cm.CreateParameter("quantity",200,,255)
'Define and execute first insert
cm("material_type")="light bulbs"
cm("quantity")="40"
cm.Execute
'Define and execute second insert
cm("material_type")="fuses"
cm("quantity")="600"
cm.Execute
%>
```

检查上面的例子，可以注意到，脚本用不同的数值重复构建和发出一个 SQL 查询，而没有重新定义和发送查询到数据库源。用 Command 对象编译查询也可避免 SQL 查询引起的合并字符串和表格变量问题。特别是，通过使用 Command 对象的 Parameter 集合可以避免与定义字符串、日期、时间变量的类型有关的问题。例如，下面的包含“'”的 SQL 查询值可能导致查询失败：

```
strSQL="INSERT INTO Customers (FirstName,LastName) VALUES ('Robert','O'Hara')"
```

注意：姓 O'Hara 中包含一个“'”，它与在 SQL VALUES 关键字中用来表示数据的“'”冲突。通过将查询数值作为 Command 对象参数绑定，可以避免此类问题。

11．结合 HTML 表格对数据库进行访问

包含 HTML 表格的 Web 页可使用户远程查询数据库并且检索特定的信息。用 ADO 可以创建非常简单的脚本来收集用户表格信息、创建自定义的数据库查询以及将信息返回给用户。使用 ASP Request 对象，可以检索输入到 HTML 表格的信息并将这些信息合并到 SQL 语句中。例如，下面的脚本模块将 HTML 表格提供的信息插入表格中。此脚本用 Request 对象的 Form 集合收集用户信息。

```
<%
'Open a connection using Connection object. The Command object
'does not have an Open method for establishing a connection
strDSN="FILEDSN=MyDatabase.dsn"
Set cn=Server.CreateObject("ADODB.Connection")
cn.Open strDSN
'Instantiate Command object
'and use ActiveConnection property to attach
'connection to Command object
Set cm=Server.CreateObject("ADODB.Command")
Set cm.ActiveConnection=cn
'Define SQL query
cm.CommandText="INSERT INTO MySeedsTable (Type) VALUES (?)"
'Define query parameter configuration information
cm.Parameters.Append cm.CreateParameter("type",200,,255 )
'Define and execute insert
cm("type")=Request("SeedType")
cm.Execute
%>
```

有关表格和使用 ASP Request 对象的详细信息，请参阅本书相关章节。

12．管理数据库连接

设计一个能经得起考验的 Web 数据库应用程序（例如为几千个客户服务的联机购物应用程序）的最大挑战，在于如何合理地管理数据库连接。打开并且保持数据库连接，即使在没有信息传输时，也会严重耗费数据库服务器的资源并且可能会导致连接性问题。设计良好的 Web 数据库应用程序将回收数据库连接并能够补偿由于网络堵塞而造成的延迟。

13．使连接超时

活动的突然增长可能使数据库服务器变得十分笨拙，大量增加建立数据库连接的时间。结果是，过长的连接延时将降低数据库的性能。

利用 Connection 对象的 ConnectionTimeout 属性，可以限制放弃连接尝试并发出错误消息之前应用程序等待的时间。例如，下面的脚本设置 ConnectionTimeout 属性，在取消连接尝试之前等待 20 秒。

```
<%
```

```
Set cn=Server.CreateObject("ADODB.Connection")
cn.ConnectionTimeout=20
cn.Open "FILEDSN=MyDatabase.dsn"
%>
```

ConnectionTimeout 属性的默认值是 30 秒。

注意：在将 ConnectionTimeout 属性合并到数据库应用程序之前，一定要确保连接提供程序和数据源支持该属性。

14．共享连接

经常建立和中断数据库连接的 Web 数据库应用程序可能会降低数据库服务器的性能。ASP 支持用 ODBC 3.5 的共享特性有效地管理连接。连接共享维持打开的数据库连接并管理不同的用户共享该连接，以维持其性能和减少空闲的连接数。对每一个连接请求，连接池首先确定池中是否存在空闲的连接。如果存在，连接池返回连接而不是建立到数据库的新连接。

如果希望将 ODBC 驱动程序加入到连接共享中，则必须配置数据库驱动程序并在 Windows NT 注册表中设置驱动程序的 CPTimeout 属性。当 ODBC 断开连接时，连接被存入连接池中，而不是被断开。CPTimeout 属性决定在连接池中的连接保留的时间长度。如果在连接池中连接保留的时间比 CPTimeout 设置的时间长，则连接将被关闭并且从连接池中删除。CPTimeout 的默认值是 60 秒。

可以通过创建如下设置的注册表键来有选择地设置 CPTimeout 的属性，从而启用特定 ODBC 数据库驱动程序的连接池。

```
\HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\driver-name\CPTimeout
  =timeout
(REG_SZ,units are in seconds)
```

例如，下面的键将 SQL Server 驱动程序的连接池的超时设置定为 180 秒（3 分钟）。

```
\HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\SQL Server\CPTimeout
  =180
```

注意：默认情况下，将 CPTimeout 设置为 60 秒，Web 服务器将激活 SQL Server 的连接池。

15．使用跨页连接

尽管可以通过存储 ASP 的 Application 对象的连接重复使用跨页连接，但是，始终使连接保持打开是不必要的，也没有充分利用连接池的优点。如果有许多用户需要连接到同一个 ASP 数据库应用程序，一个好方法就是，将跨页连接字符串置于 ASP 的 Application 对象中，重复使用数据库连接。例如，可以在 global.asa 文件的 Application_OnStart 事件过程中指定连接字符串，如下面的脚本所示：

```
Application.lock
```

```
Application("ConnectionString")="FILEDSN=MyDatabase.dsn"
Application.unlock
```

然后，在每一个访问数据库的 ASP **文件中写入：**

```
<object RunAT="Server" ID=cn ProgID="ADODB.Connection"></object>
```

要想创建连接对象的实例，请使用以下脚本：

```
cn.Open Application("ConnectionString")
```

对于打开的连接，可以在页尾写入以下脚本，关闭连接:

```
cn.Close
```

在单个用户需要重复使用跨页连接的情况下，使用 Session **对象连接比使用** Application **对象更好。**

16．关闭连接

要想更好地使用连接池，就应尽快地关闭数据库连接。默认情况下，当脚本执行完后，连接将被终止。当不再需要连接时将其关闭，就可以减少对数据库服务器的要求，并使其他用户能够使用该连接。可以使用 Connection **对象的** Close **方法终止** Connection **对象和数据库之间的连接。例如，下面的脚本打开连接，然后将其关闭：**

```
<% strDSN="FILEDSN=MyDatabase.dsn"
Set cn=Server.CreateObject("ADODB.Connection")
cn.Open
cn.Close
%>
```

17．Recordset **对象的常用属性**

Recordset **对象具有以下常用属性：**

- AbsolutePage：**目前数据记录在页数中的位置，位于第几页。**
- AbsolutePosition：**目前数据记录在项数中的位置，位于第几页。**
- ActiveConnection：Recordset **所属的** Connection **对象。**
- BOF：**表明已到了数据记录的开头。**
- Bookmark：**书签记号。目的是为了记录目前数据记录的位置，以便将来在程序中也能快速地回到这个位置的数据记录。**
- CacheSize：**暂存于内存的数据记录数量。**
- CursorType：Recordset **的游标模式。**
- EOF：**与** BOF **相反，指数据记录的结尾。**
- PageCount：**数据记录共有多少页。**
- PageSize：**数据记录每页的记录数量。**
- RecordCount：**数据记录所有的记录数量。**

- Source：设定数据来源。
- Status：目前数据记录的状态。

18．Recordset 对象的常用方法

Recordset 对象具有以下常用方法：

- Move：将指针移动到当前数据库的指定记录的位置。
- MoveFirst：将指针移到第 1 条数据记录。
- MoveLast：将指针移到当前数据库的最后一条数据记录。
- MovePrevious：将指针移到当前数据库的上一条数据记录。
- MoveNext：将指针移到当前数据库的下一条数据记录。
- AddNew：在数据库里增加一条新的数据记录。
- Close：关闭已打开的对象。
- Delete：删除目前的数据记录。
- NextRecordset：消除目前的 Recordset 对象，并且执行下一条指令。该方法返回一个记录集。其语法如下：

```
Set recordset2=recordset1.NextRecordset(RecordsAffected)
```

- Open：打开指针（Cursor）。其语法如下：

```
recordset.Open Source,ActiveConnection,CursorType,LockType,Options
```

其中：

- *Source*：可选，计算 Command 对象的变量名、SQL 语句、表名、存储过程调用或持久 Recordset 文件名。
- *ActiveConnection*：可选。计算有效的 Connection 对象变的量名，包含 *ConnectionString* 参数的字符串。
- *CursorType*：可选。CursorTypeEnum 值，确定提供者打开 Recordset 时应该使用的游标类型。可为下列常量之一：
  - AdOpenForwardOnly:（默认值）打开仅向前类型游标。
  - AdOpenKeyset：打开键集类型游标。
  - AdOpenDynamic：打开动态类型游标。
  - AdOpenStatic：打开静态类型游标。
- *LockType*：可选。确定提供者打开 Recordset 时应该使用的锁定（并发）类型的 LockTypeEnum 值，可为下列常量之一：
  - AdLockReadOnly：（默认值）只读，不能改变数据。
  - AdLockPessimistic：保守式锁定（逐个）。提供者完成确保成功编辑记录所需的工作，通常通过在编辑时立即锁定数据源的记录。
  - AdLockOptimistic：开放式锁定（逐个）。提供者使用开放式锁定，只在调用 Update 方法时才锁定记录。

- AdLockBatchOptimistic：开放式批更新。用于批更新模式（与立即更新模式相对）。

- *Options*：可选。长整型值，用于指示提供者如何计算 *Source* 参数（如果它代表的不是 Command 对象），或从以前保存 Recordset 的文件中恢复 Recordset。可为下列常量之一；

  - AdCmdText：指示提供者将 *Source* 作为命令的文本定义来计算。
  - AdCmdTable：指示 ADO 生成 SQL 查询以便从在 *Source* 中命名的表中返回所有行。
  - AdCmdTableDirect：指示提供者更改从在 *Source* 中命名的表中返回所有行。
  - AdCmdStoredProc：指示提供者将 *Source* 视为存储过程。
  - AdCmdUnknown：指示 *Source* 中的命令类型为未知。
  - AdCmdFile：指示从在 *Source* 中命名的文件中恢复保留（保存的）Recordset。
  - AdAsyncExecute：指示异步执行 *Source*。
  - AdAsyncFetch：指示在提取 Initial Fetch Size 属性中指定的初始值后，应该异步提取所有剩余的行。如果所需的行尚未提取，主线程将被堵塞，直到行重新可用。
  - AdAsyncFetchNonBlocking：指示主线程在提取期间从未堵塞。如果所请求的行尚未提取，当前行自动移到文件末尾。

- Requery：重新执行查询，并且更新数据记录的数据。其语法如下：

```
recordset.Requery Options
```

- Resync：与数据库做同步操作，从数据库更新 Recordset 的数据记录。其语法如下：

```
recordset.Resync AffectRecords,ResyncValues
```

- Supports：设定是否提供其他特别的功能。
- Update：将变动的数据更新到数据库里。其语法如下：

```
recordset.Update Fields,Values
```

- UpdateBatch：将变动的批处理数据更新到数据库里。其语法如下：

```
recordset.UpdateBatch AffectRecords
```

其中 *AffectRecords* 表示受影响的记录数

22.3 小 结

网站数据库可以说是 ASP 最大的应用领域，它的内容重要而繁多。本章只是介绍了一些网站数据库的基础知识，更多的内容需要读者在实际的工作过程中去学习和体会。

第 23 章　ASP 的使用技巧

本章介绍两个比较实用的 ASP 使用技巧。读者可以借鉴一下这两个技巧，也可以根据自己的实际需要来选择不同的方法。

23.1　出错时显示详细信息

在 Windows 2000 和 IIS 5.0 默认的情况下，系统不会显示 ASP 错误的详细信息，而只显示错误号，如“错误 ID800xxxxx”，这使得程序调试很不方便。有人采取在 IIS 中设置 500-100 错误指向默认，这样做也不会显示详细的错误信息。实际上这可能是微软出于安全考虑而在 500-100.asp 中没有显示详细出错信息。其实，只要把 500-100.asp 稍加修改就可以了。下面是经作者修改后的文件，只要把它存为 500-100.asp，然后覆盖原来的这个文件就可以了。注意文件中两行横线之间就是修改的部分。

```
<%@ language="VBScript" %>
<%
Option Explicit
Const lngMaxFormBytes=200
Dim objASPError,blnErrorWritten,strServername,strServerIP,strRemoteIP
Dim strMethod,lngPos,datNow,strQueryString,strURL
If Response.Buffer Then
  Response.Clear
  Response.Status="500 Internal Server Error"
  Response.ContentType="text/html"
  Response.Expires=0
End If
Set objASPError=Server.GetLastError
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html dir=ltr>
<head>
<style>
a:link          {font:9pt 宋体;color:FF0000}
a:visited       {font:9pt 宋体;color:#4e4e4e}
</style>
<meta name="robots" content="noindex">
<title>本页无法显示</title>
```

```
<meta http-equiv="content-type" content="text-html; charset=GB2312">
<meta name="ms.locale" content="zh-cn">
</head>
<script>
function Homepage( ){
  <!--
  //in real bits, urls get returned to our script like this:
  //res://shdocvw.dll/http_404.htm#http://www.DocURL.com/bar.htm
  //For testing use
  //DocURL="res://shdocvw.dll/http_404.htm#https://www.microsoft.com
  //bar/.htm"
  DocURL=document.URL;
  //this is where the http or https will be, as found by searching for
  //://, but skipping the res://
  protocolIndex=DocURL.indexOf("://",4);
  //this finds the ending slash for the domain server
  serverIndex=DocURL.indexOf("/",protocolIndex+3);
  //for the href, we need a valid URL to the domain. We search for the
  //# symbol to find the begining of the true URL, and add 1 to skip
  //it - this is the BeginURL value. We use serverIndex as the end
  //marker.
  //urlresult=DocURL.substring(protocolIndex-4,serverIndex);
  BeginURL=DocURL.indexOf("#",1)+1;
  urlresult=DocURL.substring(BeginURL,serverIndex);
  //for display, we need to skip after http://, and go to the next
  //slash
  displayresult=DocURL.substring(protocolIndex+3,serverIndex);
  document.write('<a hbref="'+urlresult+'">'+displayresult+"</a>");
}
//-->
</script>
<body bgcolor="FFFFFF">
<table width=410 cellpadding=3 cellspacing=5>
<tr>
<td align=left valign=middle width=360>
<h1 style="COLOR:000000;FONT:9pt 宋体">
<!--Problem-->本页无法显示</h1>
</td>
</tr>
<tr>
<td width=400 colspan=2>
<font style="COLOR:000000;FONT:9pt 宋体">试图访问的网页出现问题，无法显示。
</font></td>
```

```
</tr>
<tr>
<td width=400 colspan=2>
<font style="COLOR:000000;FONT:9pt 宋体">
<hr color="#C0C0C0" noshade>
<p>请尝试以下方法：</p>
<ul>
<li id="instructionsText1">单击
<a href="javascript:location.reload( )">
刷新</a>按钮或者稍候再试。<br>
</li>
<li>打开
<script>
<!--
if(!((window.navigator.userAgent.indexOf("MSIE")>0)&&
   (window.navigator.appVersion.charAt(0)=="2"))){
   Homepage( );
}
//-->
</script>
主页，然后查找与所需信息相关的链接。</li>
</ul>
<h2 style="FONT:9pt 宋体;color:000000">HTTP 500.100-内部服务器错误-ASP 错误
<br>
Internet 信息服务</h2>
<hr color="#C0C0C0" noshade>
<p>技术信息（适用于支持人员）</p>
<ul>
<li>错误类型：<br>
<%
Dim bakCodepage
bakCodepage=Session.CodePage
Session.Codepage=936
Response.Write Server.HTMLEncode(objASPError.Category)
If objASPError.ASPCode>"" Then
  Response.Write Server.HTMLEncode(","&objASPError.ASPCode)
Response.Write
Server.HTMLEncode("(0x"&Hex(objASPError.Number)&")")&"<br>"
If objASPError.ASPDescription>"" Then
  Response.WriteServer.HTMLEncode(objASPError.ASPDescription)&"<br>"
blnErrorWritten=False
'Only show the Source if it is available and the request is from the
'same machine as IIS
```

```
If objASPError.Source>"" Then
  strServername=LCase(Request.ServerVariables("SERVER_NAME"))
strServerIP=Request.ServerVariables("LOCAL_ADDR")
strRemoteIP=Request.ServerVariables("REMOTE_ADDR")
If (strServername="localhost" Or strServerIP=strRemoteIP)
  And objASPError.File<>"?" Then
  Response.Write Server.HTMLEncode(objASPError.File)
If objASPError.Line>0 Then
  Response.Write ",第"&objASPError.Line&"行"
If objASPError.Column>0 Then
  Response.Write ",第"&objASPError.Column&"列"
  Response.Write "<br>"
  Response.Write "<font style=""COLOR:000000; FONT:9pt 宋体""><b>"
  Response.Write Server.HTMLEncode(objASPError.Source)&"<br>"
  If objASPError.Column>0 Then
    Response.Write String((objASPError.Column-1),"-")&"^<br>"
    Response.Write "</b></font>"
    blnErrorWritten=True
  End If
End If
If Not blnErrorWritten And objASPError.File<>"?" Then
  Response.Write "<b>"
Response.Write Server.HTMLEncode(objASPError.File)
If objASPError.Line>0 Then
  Response.Write Server.HTMLEncode(",第"&objASPError.Line&"行")
  If objASPError.Column>0 Then
  Response.Write ",第"&objASPError.Column&"列"
  '--------------------------------------------------------------------
  '以下内容为笔者所添加
  '日期:2000/5/10
  dim l_strAspDescription
  l_strAspDescription="错误原因:"+objASPError.Description( )
  If l_strAspDescription<>"" Then
    Response.Write("<p>"+l_strAspDescription+"</p>")
  End If
  l_strAspDescription="详细描述:"+objASPError.ASPDescription( )
  If l_strAspDescription<>"" Then
    Response.Write("<p>"+l_strAspDescription+"</p>")
  End If
  '--------------------------------------------------------------------
  Response.Write "</b><br>"
End If
%>
```

```
</li>
<p>
<li>浏览器类型：<br>
<%= Request.ServerVariables("HTTP_USER_AGENT") %>
</li>
<p>
<li>页：<br>
<%
strMethod=Request.ServerVariables("REQUEST_METHOD")
Response.Write strMethod&" "
If strMethod="POST" Then
  Response.Write Request.TotalBytes&" bytes to "
End If
Response.Write Request.ServerVariables("SCRIPT_NAME")
lngPos=InStr(Request.QueryString,"|")
If lngPos>1 Then
  Response.Write "?"&Left(Request.QueryString,(lngPos-1))
End If
Response.Write "</li>"
If strMethod="POST" Then
  Response.Write "<p><li>POST 数据:<br>"
  If Request.TotalBytes>lngMaxFormBytes Then
    Response.Write
    Server.HTMLEncode(Left(Request.Form,lngMaxFormBytes))&" . . ."
  Else
  Response.Write Server.HTMLEncode(Request.Form)
End If
Response.Write "</li>"
End If
%>
<p>
<li>时间：<br>
<%
datNow=Now( )
Response.Write
Server.HTMLEncode(FormatDateTime(datNow,1)&","&FormatDateTime(datNow,3))
Session.CodePage=bakCodepage
%>
</li>
</p>
<p>
<li>详细信息：<br>
<% strQueryString="prd=iis&sbp=&pver=5.0&ID=500;100&cat="&_
```

```
  Server.URLEncode(objASPError.Category)&_
  "&os=&over=&hrd=&Opt1="&Server.URLEncode(objASPError.ASPCode)&"&_
    Opt2="&Server.URLEncode(objASPError.Number)&_
    "&Opt3="&Server.URLEncode(objASPError.Description)
strURL="http://www.microsoft.com/ContentRedirect.asp?"&strQueryString
%>
<a href="<% =strURL %>">Microsoft 支持</a>
</li>
</p>
</font></td>
</tr>
</table>
</body>
</html>
```

23.2 打印测试、有效性检查及错误处理

一些 ASP 的初学者为了测试一个值到处用 Response.Write 来打印，而要看页面效果时再删除这些语句或加上注释，在正式版本出来以前要如此反复多次。而有些人为了减少麻烦，干脆全当它是正确的，不做输出测试，这样极易出现各种各样的问题。最常见的是如果要生成一条 SQL 语句，需要使用变量，如果不做打印测试，很难做到一次正确。实际上只要打印出这条语句，看一下语法是否正确就行了。好的编程习惯应该是在自己没有很大把握的情况下把生成的语句或变量值打印出来，但这样做费时又费力。

在 C 语言里可以使用_DEBUG 这样的测试开关来控制 Debug 版本和 Release 版本，但 ASP 中并没有类似#define 这样的语句，那么是不是就没有办法了呢？其实完全可以仿照 C 语言的这种做法，那就是在 global.asa 文件里定义一个 Application 变量来控制，象下面这个例子所示：

在 global.asp 里加上：

```
Application("DEBUG")=1
```

然后做这么一个过程：

```
'------------------------------------------------------
'Name:          PRINT
'Argument:       a_strPrint:    打印字符串
'Return:
'Description:    打印（仅在 DEBUG 状态下运行）
'------------------------------------------------------
Sub PRINT(a_strPrint)
  If Application("DEBUG")=1 Then
    Response.Write("<p align=center>"+a_strPrint+"</p>")
```

```
End Sub
```

这个过程的功能就是当测试开关打开时（Application("DEBUG")=1）打印，而当测试开关关闭（Application("DEBUG")=0）后就不会有打印输出了。这样在程序调试期间，可以打开测试开关，以观察变量的值，而当要看页面效果或发布 Release 版本时就可以关闭测试开关，这样所有的测试输出就不会出现在页面上。

以上谈到的是变量的输出测试，下面谈谈正确性检测问题。经常见到很多人把从页面提交过来或从数据库取出的值想都不想就用，根本不做正确性检测，那怎么能保证这些值的正确性呢？比如有一个变量为 input，提交后它的值应该是一个包含数字的字符串，但如果用户的输入包含其他字符，若不做正确性检测，那当用 CInt 或 CLng 函数转换时就会发生错误，导致整个程序崩溃。另外一种情况是这样，当从数据库中取值或诸如此类的操作，应该是不会发生问题，但如果出现数据库出错等问题，那么用户也只能见到一个诸如“ odbc 错误 ”等等的提示信息，对于一个成熟的商品程序来说，这点是很不好的，其实现在包括国内很多知名站点也出现这种问题。所以应该养成这样一个习惯，那就是任何可能出问题的变量、参数在使用之前都应该做正确性检测，并且对数据库操作后应当判断是否成功。这是就又出现一个版本问题，如果是 Debug 版本则应显示出错信息以备修改，而 Release 版本则应该引导到一个统一的页面，如“ 本站暂时出现未知故障，请稍候再来 ”等等。原则上永远不要给用户一个系统出错信息页面。要实现上述功能，请看以下几个函数和过程。

```
'-------------------------------------------------------
'Name:          ASSERT
'Argument:      a_blnConditon:  断言条件
'               a_FunctionName: 调用函数
'               a_ErrorString:  错误描述
'Return:
'Description:   断言
'History:       Create by Bigeagle
'-------------------------------------------------------
Sub ASSERT(a_blnConditon,a_FunctionName,a_ErrorString)
  If Application("DEBUG")=0 Then
    If a_blnConditon<>TRUE Then
      Response.Redirect("../include/bigerror.asp")
    End If
    Else If a_blnConditon<>TRUE Then
      call print("断言错误:在<font color=red>"+a_FunctionName+"_
        </font>出现："+a_ErrorString)
      Response.End
    End If
  End If
End Sub
```

这个过程的作用是检测变量或参数有效性，如果条件 *a_blnCondition*<>TRUE，那么如

果测试开关打开，则显示错误信息；如果测试开关关闭，则重定向到错误处理页面 bigerror.asp。

```
'-------------------------------------------------------
'Name:          CheckError
'Argument:
'Return:
'Description:   检查错误
'-------------------------------------------------------
Function CheckError( )
  Dim intErrNumber
  intErrNumber=Err.Number  '保存错误代码，因为在 ERROR 中将执行 Err.Clear
  If intErrNumber<>0 Then
    Call ERROR(-1,"")  'Err 错误的错误码为-1
  CheckError=intErrNumber
End Function

'-------------------------------------------------------
'Name:      ERROR
'Argument:  a_intErrCode:错误码（-1 时表示是系统错误，即 Err.Number<>0）
'           a_strErrText:错误描述
'Return:
'Description:   错误处理
'-------------------------------------------------------
Sub ERROR(a_intErrCode,a_strErrText)
  Dim strMsg
  Dim strLogMsg
  '如果是 Err 错误，则一定执行错误页
  If a_intErrCode=-1 Then
    strMsg=strMsg+"*******************************"+"<br>"
    strMsg=strMsg+"错误时间："+CStr(Now())       +"<br>"
    strMsg=strMsg+"错误类型：Err 错误"             +"<br>"
    strMsg=strMsg+"错误号  ："+CStr(Err.Number)   +"<br>"
    strMsg=strMsg+"错误源  ："+Err.Source          +"<br>"
    strMsg=strMsg+"错误描述："+Err.Description   +"<br>"
    strMsg=strMsg+"*******************************"+"<br>"
    strLogMsg=strLogMsg+"*************"+Chr(13)+Chr(10)
    strLogMsg=strLogMsg+"错误时间："+CStr(Now( ))+Chr(13)+Chr(10)
    strLogMsg=strLogMsg+"错误类型：Err 错误"+Chr(13)+Chr(10)
    strLogMsg=strLogMsg+"错误号  ："+CStr(Err.Number)+Chr(13)+Chr(10)
    strLogMsg=strLogMsg+"错误源  ："+Err.Source+Chr(13)+Chr(10)
    strLogMsg=strLogMsg+"错误描述："+Err.Description+Chr(13)+Chr(10)
    strLogMsg=strLogMsg+"*****************"+Chr(13)+Chr(10)
```

```
  '清空 Err
  Err.Clear
  '在 Redirect 之前写日志
  WriteLog(strLogMsg)
  strMsg=Server.UrlEncode(strMsg)
  '如果是调试则显示错误代码
  If Application("DEBUG")=1 Then
    Response.Redirect("../include/error.asp?"+"ErrText="&strMsg&"")
  Else
    Response.Redirect("../include/error.asp")
  End If
Else  '如果是程序错误，则写日志
  strMsg=strMsg+"************************************"+"<br>"
  strMsg=strMsg+"错误时间："+CStr(Now( ))+"<br>"
  strLogMsg=strLogMsg+"**********************"+Chr(13)+Chr(10)
  strLogMsg=strLogMsg+"错误时间："+CStr(Now( ))+Chr(13)+Chr(10)
  Dim strWhichErr
  Select Case a_intErrCode
    Case 99001  '程序错误从 99001 开始
      strWhichErr="断言错误"
    Case 99002
      strWhichErr="Case Else 错误"
    Case Else
      strWhichErr="未知的错误"
  End Select
  strMsg=strMsg+"错误类型："+strWhichErr+"<br>"
  If a_strErrText<>"" Then
    strMsg=strMsg+"错误描述："+a_strErrText+"<br>"
  End If
  strMsg=strMsg+"************************************"+"<br>"
  strLogMsg=strLogMsg+"错误类型："+strWhichErr+Chr(13)+Chr(10)
  If a_strErrText<>"" Then
    strLogMsg=strLogMsg+"错误描述："+a_strErrText+Chr(13)+Chr(10)
  End If
  strLogMsg=strLogMsg+"**********************"+Chr(13)+Chr(10)
  '写日志
  WriteLog(strLogMsg)
  '如果是调试状态则指向错误页
  If Application("DEBUG")=1 Then
    strMsg=Server.UrlEncode(strMsg)
    Response.Redirect("http://server1/4biz/include/error.asp?"+_
      "ErrText="&strMsg&"")
  End If
```

```
  End If
End Sub

'-------------------------------------------------------
'Name:          WriteLog
'Argument:      a_strMsg:日志内容
'Return:
'Description:   错误处理
'-------------------------------------------------------
Function WriteLog(a_strMsg)
  Dim strFileName
  If Application("g_strLogFileName")="" Then
    strFileName="c:\AspLog.txt"
  Else
    strFileName=Application("g_strLogFileName")
  End If
  Dim objFSO
  Dim objFile
  Set objFSO=CreateObject("Scripting.FileSystemObject")
  '只有一个人写日志
  Application.Lock
  Set objFile=objFSO.OpenTextFile(strFileName,8,True,0)
  objFile.Write(a_strMsg)
  objFile.Close
  Application.UnLock
  Set objFile=Nothing
  Set objFSO=Nothing
End Function
```

以上几个函数及过程的作用是处理数据库错误，注意使用时在 global.asp **里加上** on eroor resume **语句，以使错误处理程序能够运行。其中可以显示错误代码、写日志文件，详细内容就不做解释了，请读者自己研究一下源代码。**

23.3 小 结

本章介绍了两个小技巧：

- **有关** Windows 2000、IIS5 **中** ASP **详细出错信息的显示。**
- ASP **编程中的打印测试、有效性检查及错误处理。**

总之，读者应养成良好的编程习惯，不断积累自己的编程经验。

第 24 章　ASP 安全

ASP 功能强大，但同时也带来了很多安全性问题。网站的安全就是网站的生命线，假如网站被人恶意入侵，盗取重要的资料或删除网站程序，那将对网站带来致命的打击。本章将列出一些常见的 ASP 安全漏洞和解决方法，供读者参考。

24.1　常见的 ASP 漏洞及解决方法

这里的安全性其实并不完全是指 ASP 本身的安全性，而是包括其使用的操作平台、运行环境所带来的安全性问题，因此，可以把 ASP 的安全性分为两个方面来描述：ASP 程序自身的安全性（程序安全）和其操作平台 Windows NT（Windows 2000）以及 IIS 的安全性（系统安全）。下面将分别介绍常见的这两类安全漏洞和解决方法。

24.1.1　系统安全漏洞

这类漏洞的出现主要是由于 Windows NT（Windows 2000）和 IIS 系统本身存在漏洞，使入侵者可以利用这些漏洞控制机器、获得本应该是保密的信息，或者对系统造成破坏，目前被发现的这类漏洞主要有以下几种：

（1）IIS 4.0/5.0 超长文件名请求存在漏洞。

这是由于映射.htr 文件的.dll 引起的。如果在一个已知的文件名后加 230 个“ %20 ”再加个“ .htr ”，会使安装有 IIS 4.0/5.0 的系统泄漏该文件的内容。

影响版本：

Microsoft Internet Information Server 4.0

Microsoft Internet Information Server 5.0

解决方法：

对于这个漏洞，Microsoft 公司已经提供了专门的补丁给用户免费下载，以修补这个漏洞。下载地址：

Microsoft IIS 5.0：

http://download.microsoft.com/download/win2000platform/Patch/Q249599/NT5/EN-US/Q249599_W2K_SP1_X86_en.exe

Microsoft IIS 4.0：

http://download.microsoft.com/download/iis40/Patch/Q260838/NT4ALPHA/EN-US/ismpst4i.exe

（2）用 IIS 4.0/5.0 特殊数据格式的 URL 请求远程 DOS 攻击。

如果在安装有 IIS 4.0/5.0 的 Web 服务器上请求一个具有特殊数据格式的 URL，会使受攻击的 Web 服务器的响应速度变慢，甚至会使其暂时停止响应。

影响版本：

Microsoft Internet Information Server 4.0

Microsoft Internet Information Server 5.0

解决方法：

下载相应的补丁进行修补即可。下载地址：

Internet Information Server 4.0：

http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20906

Internet Information Server 5.0：

http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20904

（3）MDAC——执行本地命令漏洞。

IIS 的 MDAC 组件存在一个漏洞，可以导致攻击者远程执行目标系统的命令。核心问题在于 RDSDatafactory，默认情况下，它允许远程命令发送到 IIS 服务器中，这个命令会以设备用户的身份运行，默认情况下是 SYSTEM 用户。

影响版本：

Microsoft Internet Information Server 4.0

Microsoft Internet Information Server 5.0

解决方法：

删除 IIS 中相应的目录。

（4）存在一些暴力破解从而威胁.htr 程序。

IIS 4.0 中包含一个严重漏洞，就是允许远程用户攻击 Web 服务器上的用户帐号。IIS 4.0 安装的时候建立了一个虚拟目录/iisadmpwd，这个目录包含多个.htr 文件，允许匿名用户访问这些文件。如果这些文件恰好没有规定只限制在本地地址（127.0.0.1）的话，请求这些文件时就会弹出对话框让访问者通过 Web 来修改用户的帐号和密码，这样攻击就可以暴力破解用户帐号和密码。

影响版本：

Microsoft Internet Information Server 4.0

解决方法：

删除 IIS 中相应的目录。

（5）利用 Translate:f 漏洞获取 ASP 源代码。

问题存在于 Office 2000 和 FrontPage 2000 Server Extensions 中的 WebDAV 中，当有人请求一个 ASP/ASA 或者其他任意脚本的时候，在 HTTP GET 中加上 Translate:f 后缀，并在请求的文件后面加一个/就会显示文件的源代码。

影响版本：

Microsoft Internet Information Server 4.0

Microsoft Internet Information Server 5.0

解决方法：

升级 Window 2000 SP1。

24.1.2　ASP 程序安全漏洞

这类漏洞是由于在编写 ASP 程序时，没有注意处理一些细节，或者在编程思路上的误差或考虑不够全面而引起的安全性问题。实际中主要有以下几种情况：

（1）没有处理 SQL 语句而造成的安全性问题。

由于某些特殊字符对 SQL 语句会产生影响，因此，如果没有对该类字符处理的话，就会引起安全问题。这些字符主要是单引号。

首先，如果在用户验证的时候，直接输入单引号，在 ASP 中直接执行 SQL 语句时，程序会出错，而如果没有对出错页面处理的话，就会暴露程序的物理路径。而这往往是被入侵的第 1 步。

其次，利用单引号构造一些特殊语句，可以绕开用户身份验证。

一般来说，在验证用户是否合法时，会用如下的语句：

```
SqlStr=SELECT * FROM TableName WHERE username='"&Request("username")&"'
  and password='"&Request("password")&"'"
```

然后检测得到的记录集是否为空，如果为空则用户非法，如果不为空则用户是合法的。

这段程序单从技术角度看规范的，但从安全角度看就存在一个非常大的漏洞。因为攻击者可以构造一个特殊的用户口令和用户名，从而进入受保护的系统内部。例如：

```
username='aa' or username<>'aa'
password='aa' or password<>'aa'
```

相应地，在浏览器端的用户名框内写入：

```
aa' or username<>'aa
```

口令框内写入：

```
aa' or password<>'aa
```

这样就可以成功地骗过系统而进入。注意这两个字符串两头是没有单引号的。

处理方法：

一般情况下，可以检测用户输入的字符，限制用户输入非法字符。也可以对用户输入的字符作相应的处理，比如可以将单引号替换成两个单引号。同时对数据库操作尽可能地使用存储过程来处理。

（2）身份验证不够全面使非法用户进入本来无权进入的页面。

一般在采取会员制的网站中，很多页面都希望用户先通过会员身份验证后再进入，如果直接输入 URL 的话应该加以拒绝。通常的处理是构造一个 Session 对象，用户登录成功后将该 Session 赋值，然后，在每个页面开始前检测该 Session 的值是否符合要求。但这样做存在一个漏洞，攻击者可以在本地构造一个相应的 Session 后，再连接至该页面，从而绕过验证。

处理方法：

构造一个没有特殊意义的 Session 对象，让攻击者猜不到，同时在此页面对前一个页面

加以限制，也就是说限制用户只能从相应的页面连接过来，防止攻击者从本地连接。

（3）FileSystemObject 组件带来的安全性问题。

FileSystemObject 组件功能强大，但带来的杀伤力也十分可怕。如果在系统中使用 FileSystemObject 组件对文件操作的话，必然要给该文件赋予相应的操作权限，也就是说攻击者同样可以使用 FileSystemObject 组件来对文件操作。假如一个提供 ASP 空间的网站同样提供 FileSystemObject 组件支持的话，攻击者就可以上传 ASP 文件，利用 FileSystemObject 组件来浏览甚至删除整个网站目录文件，这可以说是致命的攻击。

处理方法：

删除该组件，或者将该组件改名。

（4）没有及时关闭、释放相应资源带来的安全性问题。

其实这并不算安全方面的内容，但是很多初学者会犯此类错误。如果每次建立数据库连接的时候，没有及时关闭或释放所连接的相应资源，那么当刷新该页面时，实际上又建立了一个新的数据库连接，而一个数据库连接要消耗服务器不少的资源。这样，如果用户再不断刷新页面，或者访问量大的时候，很容易耗尽服务器资源，导致 IIS 当机。

处理方法：

养成良好的编程习惯。

24.2 ASP 安全建议

由于 ASP 安全隐患主要来自两个方面，所以，这里对 ASP 安全的建议也从两个方面来讲述。

24.2.1 系统安全建议

系统安全建议有两点：

（1）安装时的“略”就是安全，够用。去掉一些多余的东西。

主要包括以下几点：

- 没有必要的话不要安装 IIS 的 FTP 服务，功能不好，还容易出错。而且 FTP 默认传输密码的过程是明文传送，漏洞很大。
- 安装时选择站点目录时，建议不要用默认目录 C:\inetpub，最好安装到非系统盘上，考虑自建目录，如 D:\IISWEB。这样即使 IIS 被突破，也能尽可能地保护好系统文件。
- 不要安装 HTML 的远程管理。HTML 的远程管理在 Wndows NT 4.0 中还能用的上，但漏洞比较大，而且比较危险。端口号虽然是随机的，但很容易被人扫描到，从而留下隐患。
- 多余的服务，如 NNTP，如果不做新闻组，就不要安装。SMTP，如果有更好的邮件服务，也不安装。
- 索引服务器（Index Server）也存在着很多的漏洞。索引不但消耗相当多的服务器

资源，而且有返回上级目录的安全漏洞。如果需要做搜索引擎的话，可以采用 SQL Server 的全文搜索服务。

（2）有目的地进行服务器安全性配置。

主要包括以下几点：

- 启动 IIS，删除多余的目录，如\iissamples、\IIShelp 和\msadc\等，它们都是作为例子或帮助，只剩下新建的虚拟目录即可。
- 共享权限的修改。在 NT 下执行“开始”|“程序”|“管理工具”|“系统策略编辑器”命令，然后执行“文件”|“打开注册表”命令进行修改，对于 Windows NT 网络，把其中勾去掉。Windows 2000 下可以写个 net share c$ /delete 的.bat 文件，放到机器的启动任务里。
- 对服务器端口和 IP 进行配置。通过配置 TCP/IP 过滤，可以禁止很多服务，但可以减少许多不必要的麻烦。执行“控制面板”|“网络”|“协议”|“TCP/IP 协议”|“属性”|“高级”|“启用安全机制”|“配置”命令进行配置，如图 24.1 所示（详细步骤可以参考 Windows 的帮助）。一般可以配置 TCP 端口为 80 和 443（SSL 的端口）；不允许 UDP 端口；IP 协议 6。这是一个典型的安全配置，推荐使用。

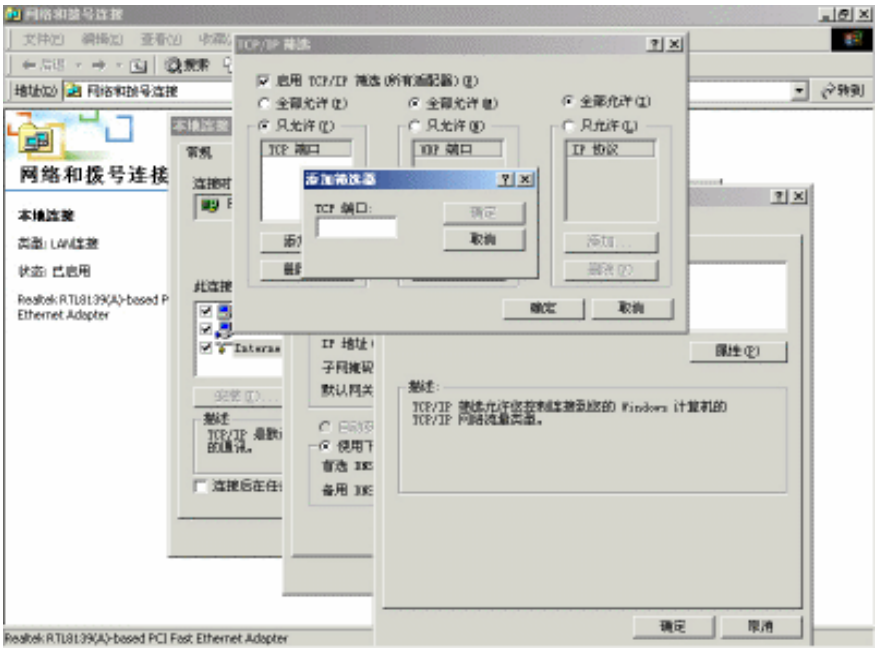

图 24.1　Windows 2000 中设定 IP 限制的界面

- 配置合适的脚本映射。大部分 ASP 源代码泄漏都是由于不安全或有错误的脚本映射导致的。而它们中的大多数可能用不到。
  - *.htr：Web 应用程序的一种。同.hta 一样，功能很强大，但安全性较低。大多数 ASP 程序员和 NT 网管不需要，可以把它的对应选项去掉。否则，任何人

都可以通过 Web 来进行非法操作，甚至格式化掉硬盘。

- *.hta：一种.html 格式的 Application，功能比较强大，可以通过它来访问 NT 的很多操作，比如可以通过 Web 来重新设置密码，在 ASP 中建一个 NT 用户也是可能的。但大多数的工作如果不通过 Web 最好。*.hta 在 Web 中很少用到，可以删掉。
- *.idc：这是个比较老的数据库连接方法了，现在大多都直接用.asp 文件而不用.idc 文件，所以也可以删掉。
- *.printer：打印机文件，同样删掉。
- *.htw、*.ida 和*.idq：这些都是索引文件，也可以去掉。

删除映射文件的方法很简单：启动 IIS，右键单击站点名称，执行“属性”命令，在弹出的对话框中单击计算机 MIME 映射的编辑按钮，出现如图 24.2 所示的界面。按照提示删除相应的映射即可。

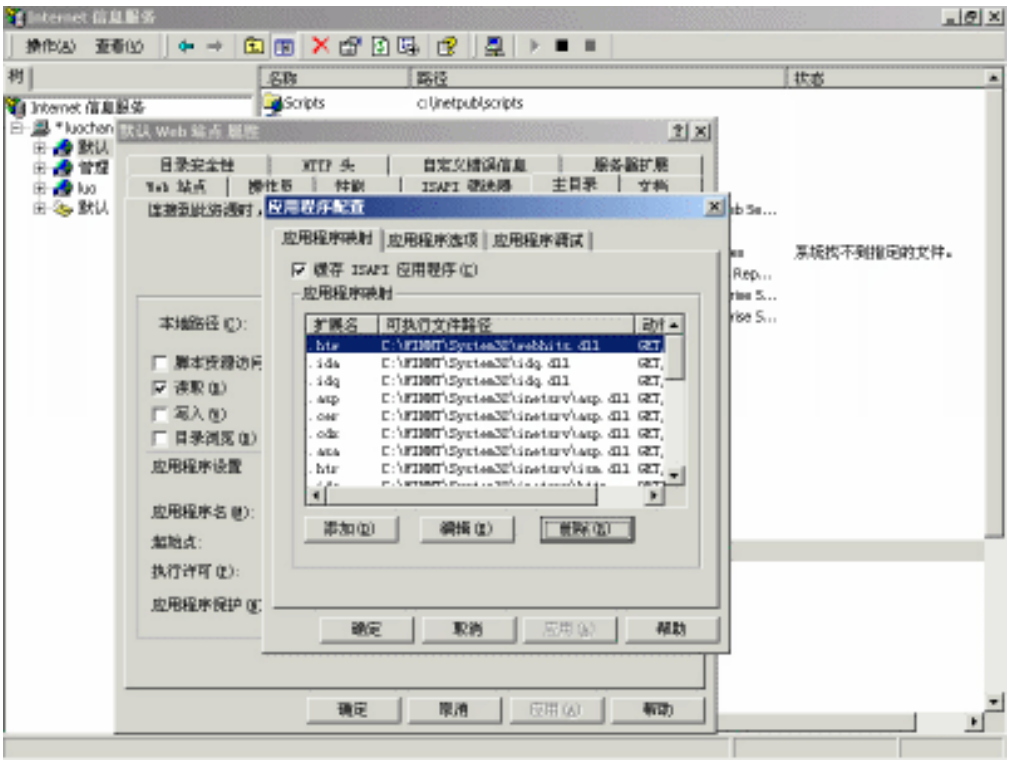

图 24.2 IIS 5.0 中删除映射文件的操作界面

24.2.2 ASP 程序安全建议

在 ASP 程序安全方面的建议主要是为了避免源代码泄漏以及拒绝无权限的用户进入网站。主要应该注意的有：

（1）保护数据库连接字符串。

毫无疑问，数据库连接字符串是十分重要的信息，可以利用 IIS 的权限机制来保护连接字符串。

首先建立连接字符串，并建立一个单独的*.inc 文件（注意，是*.inc 文件而不是*.asp 文件），把连接字符串用变量复制进来。如下所示：

```
connstr="Provider=SQLOLEDB.1;Password=passw;..."
```

然后建立一个文件夹 include，放在根目录里。每一个文件用如下所示的办法打开连接：

```
<!-- #include file="include\*.inc" -->
set conn=Server.CreateObject("adodb.connection")
conn.open connstr
```

最后在 IIS 里把 include 文件夹用拒绝读的方法保护起来。这样可以照常打开连接，但是即使攻击者看到 ASP 的源代码，也看不到连接字符串；即使看到了包含文件的路径及名称，也无法下载或用 IE 打开。这样就可以保护连接字符串了。

（2）进行严格的权限验证。

可以要求每个试图访问被限制的 ASP 内容的用户必须有有效的 Windows NT 帐号的用户名和密码。每当用户试图访问被限制的内容时，Web 服务器将进行身份验证，即确认用户身份，以检查用户是否拥有有效的 Windows NT 帐号。

（3）及时释放相关的资源。

24.3　小 结

本章介绍了一些常见的 ASP 漏洞及相应的解决方案，并提出了一些提高 ASP 安全性的建议。但实际上，并没有什么绝对的安全。如果想真正使自己的网络安全，必须注意网站每一个细小的方面，及时获取关于网络安全方面的信息，并不断对自己的网站进行改进。