北京科海培训中心

新概念 VBScript 教程

赵彩花　编著

北京科海集团公司　　出品

2001.10

内 容 提 要

本书从介绍网页、网站等使用 VBScript 编程必需的基本概念入手，循序渐进地讲述了 VBScript 的基础知识，包括数据类型、语法、语句、过程和函数等。同时还讲解了 VBScript 在实际中的高级应用，描述了如何使用浏览器中的内置对象，如何使用窗体、控件和常见的事件，以及在编程中如何捕获和处理错误。本书除了介绍 VBScript 的知识外，还介绍了如何通过将 VBScript 和 ASP 结合，编制出真正的客户端/服务器模式的程序。在本书的最后，精心制作了 8 个实例，通过对各种实例的详细讲解，希望能使读者在实例的制作过程中全方位地掌握 VBScript 的应用，并可以自己做出各种实例效果。

与本书配套的多媒体光盘采用了最新的流媒体技术，以类似于课堂教学的方式详细地讲解了 VBScript 的知识，有助于读者更迅速地掌握 VBScript 这门应用技术，并融会贯通。

本书简明实用，通俗易学，实例丰富，是学习 VBScritp 的一本非常好的教材。不仅适合初、中级用户自学，还可以作为大、中专及相关培训班的教材。

书　　名：新概念 VBScript 教程
作　　者：赵彩花
责 任 编 辑 ：王金柱
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：21.25　　字数：492 千字
版　　次：2001 年 10 月第 1 版　　2001 年 10 月第 1 次印刷
印　　数：0000~5000
盘　　号：ISBN 0-00--
定　　价：元（0 张多媒体光盘）

前　言

随着 Internet 的飞速发展，几乎每天都有新技术在世界的某个角落出现。如果能够把新技术放置于 Internet 之上，无疑将使公司、企业、单位的知名度大大提高，从而占领更广泛的市场，增强竞争力。因此网上浏览、主页制作成了人们津津乐道的热门话题。随着 FrontPage、Dreamweaver 等软件的出台，制作网页的工作变得越来越容易和大众化。然而在人们要求越来越高的今天，HTML 语言显现了它的缺点，那就是不能动态更新主页的内容，缺乏实时、动态、可视及交互性。为了适应这种要求，出现了很多编程的脚本语言，如 Microsoft 的 VBScript、Netscape 的 JavaScript 等。这些脚本语言与 HTML 语言的无缝连接，大大弥补了 HTML 的不足，实现了网页的动态交互。

Microsoft 的 VBScript 脚本语言继承了 Microsoft 的一贯风格，和其他 Microsoft 软件产品有很大的相同之处，而且由于它简单易学，成为人们编写动态网页的首选。

本书共分 11 章。

第 1 章从网页、网站等基本概念入手，介绍了学习 VBScript 应该必备的一些基础知识。

第 2 章~第 5 章讲述了 VBScript 的基础知识，包括数据类型、语法、语句、过程和函数等。

第 6 章~第 9 章讲述了 VBScript 在实际中的高级应用，描述了如何使用 VBScript 的内置对象和集合、控件和事件以及浏览器对象的使用，还讲述了在编程中如何捕获和处理错误等。

第 10 章讲述了如何将 VBScript 和 ASP 结合，以使服务器端响应客户端的输入，从而编制出真正的客户端/服务器模式的程序。

第 11 章精心制作了 8 个实例，旨在通过对各个实例的详细讲解，使读者体会 VBScript 的编程技巧和各种功能的使用方法，能够全方位地掌握 VBScript 的应用。通过实例，举一反三，并可以自己做出各种网页效果。

本书的内容编排本着循序渐进的原则，从基本知识入手，逐步深化，每一章都附有练习题，以使读者巩固学过的知识。

虽然作者长期从事网站及网页的开发工作，对 VBScript 有较深的理解，但要完成这样一本 VBScript 教程仍觉所知甚少。在本书的写作过程中，得到了许多网上不知名的朋友和作者的同事们的帮助，在此对这些 VBScript 高手们表示深深的感谢和敬意。由于作者水平所限，书中疏漏和不足之处，敬请读者批评指正。

作　者

2001 年 10 月

目　录

第 1 章　VBScript 入门

为了方便读者学习 VBScript，我们先做一些准备工作，了解几个相关概念，之后再简单介绍一下 HTML 的基本知识。

1.1　相关概念与 HTML 简介

如果你是一个 WWW 页面制作的新手，有几个概念需要掌握。

1.1.1　Web、网页和站点

1．什么是 Web

全球广域网（World Wide Wed）的简称是 WWW，也叫 Web。它是 90 年代初开发成功的，开发者是瑞士日内瓦“欧洲粒子实验室（CERN）”的 Time Berners-Lee 和他的同事们。

最初，Web 只是作为超文本文档提供服务的一种途径，1993 年 2 月，美国“国家超级计算机应用中心”发行了一个针对 Unix 系统的图形浏览器——Mosaic，它集成了图形用户界面，能直接观察 Web 中的图形，同时可以支持其他媒体类型，并能方便地浏览文本、声音、图像等多媒体信息。图形浏览器的出现，使 Web 得到了社会各界的注意，也是 Web 逐渐流行起来的原因之一。Web 的快速发展，也使 Internet 的信息量呈爆炸性的上升，这也使 Web 成为 Internet 上最主要的服务项目。

Web 站点的基本单元是 Web 页面。Web 上的信息存放在 Web 页面上，一个 Web 页面是文本、图像、声音、图形、视频等形式的信息集合。Web 页面是用 HTML（超文本标识语言）写成的文档，浏览器能够识别和解释 HTML 文档，并显示这些文档。

Web 中有“文档”和“连接”两个概念，网上的文档是按统一资源定位器 URL（Uniform Resoure Locator）寻址的。URL 完整地描述了 Internet 超文本文档的地址，该地址可以是本地磁盘，也可以是 Internet 上的站点。一个典型的 URL 地址如下：

http://www.e-yao.net/commerce/login.htm

其中，“ http ”代表用于检索文档的协议，即超文本传输协议，“ // ”后面表示的是 Internet 有效主机名，“ / ”之后紧跟的是用户要查找的文档的路径名和文件名。这样上述的 URL 地址的含义是：利用 http 协议，在 Internet 宿主机 www.e-yao.net 上的 commerce 目录下，查找文档 login.htm.。

Web 页面分为两种：静态页面和动态页面。静态页面的内容是固定不变的，而动态页面的内容在每次访问时生成，可以响应用户从浏览器上的输入。页面可以由用户定义，页面的内容可随不同用户而改变，也可随时间的不同或其他因素而改变。

一组相关的 Web 页面组成了一个 Web 站点，这些网页通常用超链接连接起来，存储在 Web 服务器某个子目录下，一个 Web 站点包括的网页可以是几个也可以是上千个，进入站点的第一个网页称为主页，通常主页中包含了必要的内容和索引信息。

2．Web 的几个重要概念

统一资源定位器 URL

URL 是互联网上资源的地址。Internet 的最大优势是信息资源丰富，但是各种各样、无边无际的信息会给用户带来查找和使用上的困难，而 URL 提供一种统一格式的 Internet 信息资源地址的表示方法，它将 Internet 提供的各种类型的服务统一编址，从而可以使用户方便地通过 URL 查找需要的资源。

URL 的格式如下：

协议：//域名／文件路径和文件名

URL 描述了 Web 浏览器对 Internet 信息资源所在的计算机主机名（域名）以及信息资源所在的文件路径和文件名。

（1）协议

协议主要有下面几种：

- HTTP：提供 WWW 服务，传输协议是 HTTP。
- FTP：提供文件传输服务，传输协议是 FTP。
- Telnet：提供远程登录服务，传输协议是 Telnet。
- Gopher：提供 Gopher 服务。
- Mailto：提供 E-mail 服务（电子邮件服务），传输协议是 SMTP。
- News：提供新闻服务，传输协议是 NMTP。

（2）域名

域名是指信息资源所在的网络或服务器，它可以唯一地确定 Internet 上每一台计算机的地址。

域名一般是指用多级域名方式表示的公司、企业或单位的网址。URL 的域名是按照四级域名的方式来分类的。域名的书写格式为：大地址在后，小地址在前，各级域名之间用“.”隔开，各级域名不区别大小写，而且不限长度。例如：http://www.sina.com.cn。

（3）文件路径和文件名

URL 最后的一部分是信息资源在服务器上的路径和文件名。有时，有的 URL 中没有文件路径，这表示该地址指向该域名代表的 Web 服务器的缺省主页的文档。

URL 中也可以使用 IP 地址做网络地址，如 202.96.3.223，这时 URL 就表示为 http://202.96.3.223

超文本传输协议 HTTP

HTTP 是客户和服务器通讯使用的下层协议。HTTP 是分布、协作的超媒体信息系统的应用层协议。该协议是一般的、无国籍的面向对象的协议。HTTP 的功能是键入并协商数

据的表示，允许建立独立于正在传输的数据的系统。通俗地说，就是超文本传输协议规定了浏览器在运行 HTML 文档时所遵循的规则和要进行的操作。它使得浏览器在运行超文本文档时有了统一的规则和标准。

1.1.2　HTML 简介

HTML 的全称是 Hyper Text Markup Language，即超文本标示语言。它是网页制作的基本语言，它决定了网页以什么样的形式出现在用户面前，所有的网页都是由 HTML 语言或者是将其他的脚本语言嵌入到 HTML 语言中编写而成的。HTML 不是一种程序语言，而是一种结构语言，它具有跨平台性。

1．HTML 的基本结构

一个 HTML 文件，一般由下列几部分组成：序、文件头和主体。

- 序：指文件的注释部分，以“ <! ”开头，以“ > ”结尾，可以放在文件中的任何地方。
- 文件头 :由头标记<HEAD>和尾标记</HEAD>构成，标题写在<TITLE>和</TITLE>之间。
- 主体：文件的主体部分放在头标记<BODY>和尾标记</BODY>中间，用于显示正文。

标记<HTML>与</HTML>表示一个 HTML 文件的开始和结束。下面我们看一个最基本的例子。

【例 1.1】　仅有标题和一行文字的 HTML 页面

代码如下：

```
<HTML>
<HEAD>
<TITLE>我的标题</TITLE>
</HEAD>
<BODY>
这个文档只有一句话
</BODY>
</HTML>
```

HTML 作为一种描述性语言，由它所写的文件只是一种纯文本文件。它可以使用 Unix 的 vi、DOS 的 Edit 编辑器、汉字系统下的文件编辑器 WPS、Word，也可以用专用编辑器如 HTML Webedit 的标记编辑器、Jpad 等提供的编辑工具生成 HTML 文件。当然，我们也可以直接在记事本里编写。

打开记事本，将【例 1.1】中的代码写入之后，将其保存为 1_1.htm。在浏览器如 Internet Explorer 下运行，结果如图 1.1 所示。

这是一个最简单的例子，它的输出只有一句话，但是向我们说明了一个 HTML 文件的基本结构。

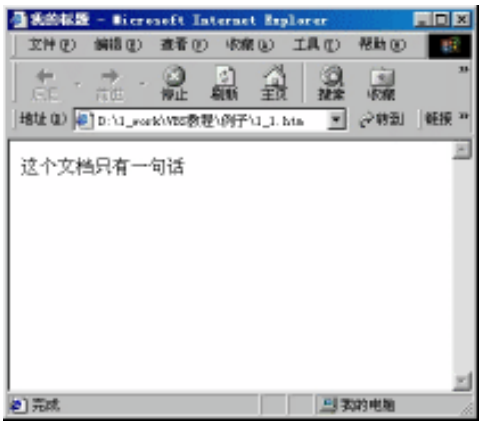

图 1.1 一个简单的 HTML 文档

2．HTML 的基本标记

元素是文档的组成部分，如 TITLE（文档的标题）、IMG（图像）、TABLE（表格）等等。元素名不分大小写。标记用来规定元素的属性和它在文档中出现的位置，用它将 HTML 文档划分成不同的逻辑部分，如段落、标题和表格等。

大多数标记都是成对出现的，也有单独出现的。一般的，首标记的格式为<元素名>，尾标记的格式为</元素名>。成对的标记用于规定元素所包含的范围，如<TITLE>与</TITLE>标记用来界定标题元素的范围，也就是说<TITLE>和</TITLE>之间包含的部分是该 HTML 文档的标题。单独标记的格式为<元素名>，它的作用是在相应的位置插入元素，如<HR>标记表示在该标记所在的位置插入一个水平基准线。

下面我们将分类介绍 HTML 的常用标记。

结构定义

（1）<HTML>标记：用来表示一个文件的起始，结束标记为</HTML>。

（2）<HEAD>标记：文档头部，用来设置初始化文档信息。此种信息包括<TITLE>、<META NAME>和其他的文档管理标注，结束标记为</HEAD>。

（3）<TITLE>标记：告诉浏览者你的页面名字，页面的题目通常显示在浏览者所阅读的页面的浏览器的标题栏中。<TITLE>标记是 HTML 页面中的<HEAD>标记中的一部分，结束标记为<TITLE>。

（4）<BODY>标记：在<BODY>和</BODY>标记中包含了 HTML 文档的全部内容，这些内容将被浏览器格式化后显示出来。显示的风格由包围文本的标签决定。

<BODY>标记中包括一些设定颜色的属性：

- text 属性：控制 Web 页中所有的文本（不包括链接）的颜色。
- link 属性：控制 Web 页中标准的、未被访问过的链接的颜色。
- vlink 属性：控制已访问过的链接的颜色。
- alink 属性：控制当一个链接在鼠标按下还没有释放时的颜色。
- background 属性：设置页面的背景图片。
- bgcolor 属性：设置页面的背景颜色。

格式控制

（1）<P>：指明一个段落的起止。

（2）<H1>：第一层头标题。

（3）<H2>：第二层头标题。

（4）<H3>：第三层头标题。

（5）<H4>：第四层头标题。

（6）<H5>：第五层头标题。

（7）<H6>：第六层头标题。

**（8）
：开始新的一行。**

（9）<HR>：插入一水平分隔线，它没有相应的结束标记，也没有关联文本，只是在页面上生成一条水平线。该标记有 4 个属性，如表 1.1 所示。

表 1.1　<HR>标记的属性

属性	说明
size	以像素为单位定义水平线的宽度
width	指出水平线的宽度
align	指出水平线的对齐方式
noshade	使浏览器以简单的黑线但无三维投影效果的方式来显示水平线

（10）<OL>：有序列表。

（11）<UL>：无序列表。

（12）<LI>：在列表中另起一行。

（13）<PRE>：按源文件格式显示。因为在多数情况下，HTML 文件中的文本是基于 HTML 标记进行格式化的，文本中任何额外的空白字符（空格、制表符和回车符）都将被浏览器忽略。但是，使用预格式化文本标记<PRE>…</PRE>，包含在其中的空白字符都可以出现在最后的屏幕输出中。

字体的逻辑风格控制

逻辑风格标记用来标识文本使用的方式，说明高亮文本是如何使用的，而不是如何显示它。

（1）<EM>：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，通常是斜体。

（2）<STRONG>：比<EM>强调的程度更深。通常表现为黑体。

（3）<CODE>：用代码形式格式化文本。

（4）<SAMP>：以样本程序输出方式格式化文本。

（5）<KBD>：用户将输入的文本。

（6）<VAR>：变量名或将被填入实际值的实体名，常以斜体或下划线方式显示。

（7）<DFN>：定义标记，它用于高亮化将被或刚被定义过的词。

（8）<CITE>：一段简短的引语，使之高亮化。

字体的物理风格控制

（1）<B>：黑体正文。

（2）<BIG>：以当前字体中的大字体显示正文。

（3）<I>：斜体正文。

（4）<S>：在文字中间加一条删除线。

（5）<SMALL>：用比当前字体小的字体显示正文。

（6）<TT>：以定宽字体显示正文。

（7）<U>：在文字下端加一条下划线。

（8）<SUB>：以其中的字体作下标显示。

（9）<SUP>：以其中的字体作上标显示。

字体和字号标记

<FONT>**标记用于控制字符的显示，**<FONT>**标记有几个重要的属性：**

（1）size **属性：用来指明字体的显示字号，**size **的取值范围是从** 1～7 **依次增大，其中** 3 **是默认值。也可以使用相对值来指定字号的大小，相对值是针对** 3 **的相对，使用“** + **”和“** - **”的方法来指定字号大小，相对值的取值范围是从**-3 **到**+4**。**

（2）face **属性：用来指明文本的实际显示字体。它要求客户机中必须包括指定的这种字体。**face **把一套字体名称作为它的值，并用逗号分隔，当浏览器解释含有** FACE **的页时，浏览器将逐个搜索系统字体得以匹配。如果** face **属性中指定的值中第一种字体找不到，就找第二种，直到找到一种实际已经安装了的字体为止。如果都找不到，就用缺省的字体代替。**

（3）color **属性：用来指明文字的颜色。可以使用两种方法设置文字的颜色，一种是用颜色的十六进制的** RGB **值。例如：**

```
<FONT color=#000099>
```

另一种方法是使用颜色的名字，所有的计算机使用的颜色名字都是一样的。要想了解详细的信息请参阅附录 A**。**

地址标记

地址标记<ADDRESS>**用于** Web **页面上类似于签名的实体。它常写在每个** Web **页面的最底行，用于显示** Web **页面的作者、联系人、日期、版权信息以及其他警告信息等。用**<ADDRESS>**标记对每个** Web **页面进行“签名”，可以使读者了解更多的信息。地址通常包含一条水平线**<HR>**和用于换行的换行符**
，下面是一个使用地址标记例子的源代码，其运行效果如图 1.2 **所示。**

```
<HTML>
<HEAD>
<TITLE>使用地址标记</TITLE>
</HEAD>
<BODY>
<P>地址标记应用实例</P>
```

```
<P>
<HR>
<ADDRESS>
中关村在线<BR>
中国.北京.海淀区.海淀南路 19 号<BR>
HTTP://www.zol.com.cn<BR>
1999-2001&copy;版权所有<BR>
</ADDRESS>
</BODY>
</HTML>
```

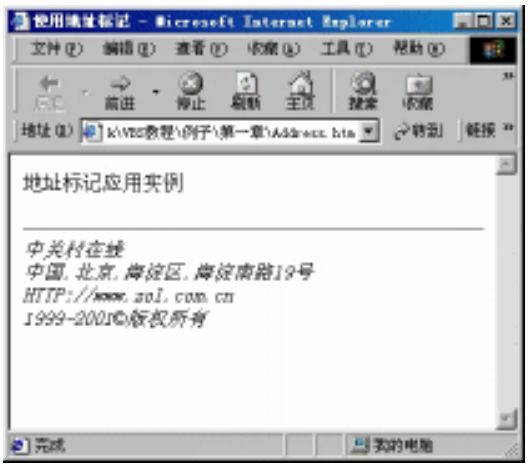

图 1.2　使用地址标记<ADDRESS>的效果

文档中的链接

链接是 HTML 的基本功能，它能够使你跳转到 Web 或你的本地网络站点上不同的文档，而且还能够跳转到同一个页面的不同部分。

链接是靠定义链接文件的 URL 实现的。我们可以使用<A>标记的“ HREF ”属性达到这个目的。如我们想要链接到 Microsoft 的网站上，就可以使用下面的语句：

```
<A href=HTTP://www.microsoft.com> 我要链接到 Microsoft</A>
```

此外，还可以链接到文档的指定位置，这时就要在该指定位置设定一个特殊标志，通常我们把这个标记叫做书签，也称锚标记。它是用 NAME 取代 HREF 属性，NAME 属性是一个命名书签的关键字。下面是定义一个书签的表示格式：

```
<A name="bookmark">这里定义了一个书签</A>
```

定义了书签之后，再在链接文本中设置链接标记：

```
<A href="书签所在的页面#书签名称 bookmark">跳转到书签 bookmark</A>
```

这样当你在浏览器中用鼠标单击跳转到书签显示的文本时，该链接将链接到你要链接的书签处。下面我们举一个例子来说明。

【例 1.2】 在 HTML 中使用书签

```
<HTML>
<HEAD>
<TITLE>链接实例</TITLE>
</HEAD>
<BODY>
<P align="center"><B>链接实例</B></P>
<P><FONT size="2">在本章，我们将学习以下内容：</FONT></P>
<UL>
   <LI>< FONT size="2">Web 相关知识</ FONT ></LI>
   <LI>< FONT size="2">常用概念</ FONT ></LI>
   <LI>< FONT size="2"><A href="#html">HTML 基本知识</A></ FONT ></LI>
   <LI>< FONT size="2">VBScript 简介</ FONT ></LI>
</UL>
<P>< FONT size="2">.....</FONT></P>
<P>< FONT size="2"><A name="html"></A><B>HTML 基础</B></ FONT >
</P>
<P>< FONT size="2">    HTML 的全称是 Hyper Text Markup Language，即超文本标示语言。它是网页制作的基本语言，决定了网页以什么样的形式出现在用户面前。HTML 不是一种程序语言，而是一种结构语言，它具有跨平台性。</FONT></P>
<P>  </P>
<P>  </P>
<P>  </P>
</BODY>
</HTML>
```

当用户单击“ HTML 基本知识 ”这句话时，就链接到书签名为“ HTML ”且写着“ HTML 基本知识 ” 的这句话上，并将这句话显示在窗口的第一行。如图 1.3 和图 1.4 所示。

图 1.3 使用书签示例——初始页面

图 1.4　跳转到书签页面

图形

要将一幅图形插入到 HTML 文件中，使用<IMG>标记。<IMG>标记有两个重要的属性：

（1）src 属性：包含了要显示的图形文件的路径。

（2）alt 属性：当浏览器正在装载要显示的图形时，可以用 alt 属性指明图形的作用。如果读者使用的是非图形浏览器（或者读者在 Internet 中关闭了图形显示选项），使用 alt 属性可以指出该图形原本是用来做什么用的。在这种情况下，如果不使用 alt 属性，则浏览器上的页面将会变得一团糟。

例如：

```
<IMG src="welcome.gif" alt="欢迎您来到我的主页">
```

图 1.5、图 1.6 和图 1.7 分别表示了正常情况下、关闭图形显示但使用 Alt 属性和关闭图形显示且没有使用 Alt 属性的 3 种情况。从这几个例图中可以看出，由于使用了 Alt 属性，当浏览器不显示图形时，将会显示一行替代文字，这将会给用户的浏览带来方便。因此我们在做网页的时候，必须想到每一种可能的情况。

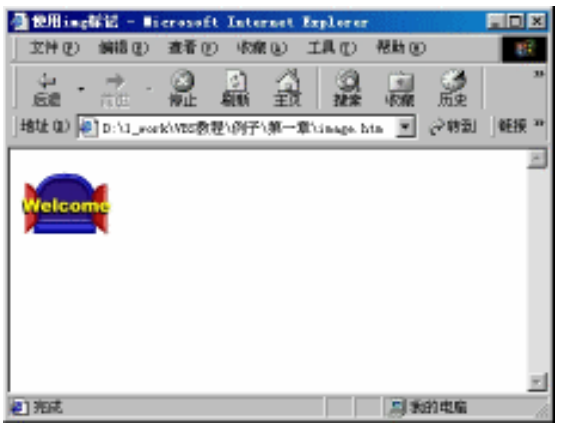

图 1.5　正常情况下的图形显示

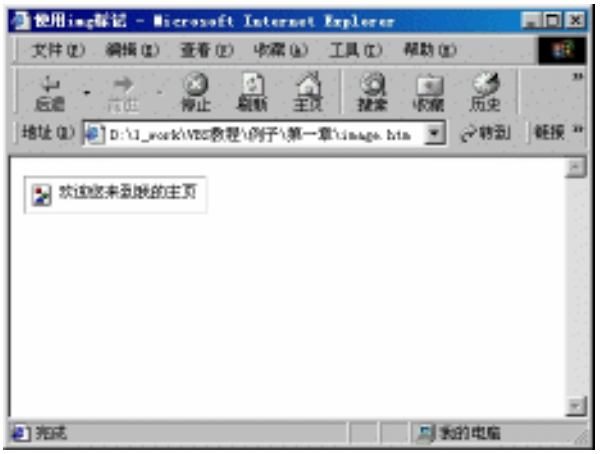

图 1.6　用户关闭了图形显示功能但在<IMG>标记中使用了 alt 属性

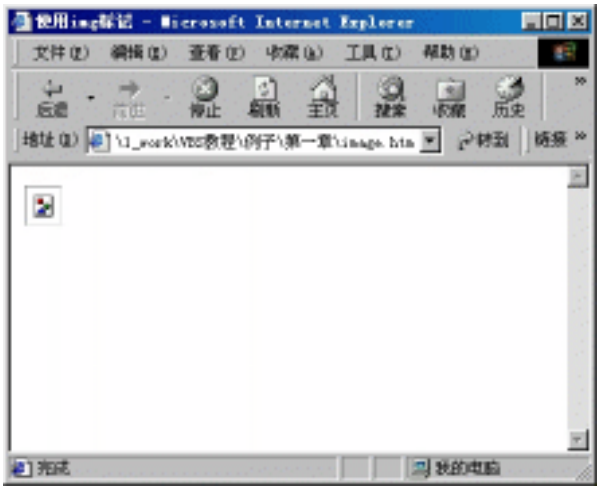

图 1.7　用户关闭了图形显示功能且<IMG>标记中没有使用 alt 属性

对齐标记

要对齐单独的一个标题或一个段落，在该 HTML 标记后使用 align 属性。align **有以下 3 个取值：**

（1）left：**左对齐**
（2）right：**右对齐**
（3）center：**中间对齐**

表格

表格（TABLE**）是组织数据的一种十分有效的方式，使用表格可以更加清晰明了的表达信息，但是在使用 HTML 进行手工制表的时候就不那么容易了，因为一个表格包含了几个部分，而不象其他的标记一样好掌握。下面我们先看一个实例，然后通过它的源代码来看一下表格都有哪些标记。**

【例 1.3】　在 HTML 中使用表格

```
<HTML>
<HEAD>
<TITLE>在 HTML 中使用表格</TITLE>
</HEAD>
<BODY>
<TABLE border="1" width="75%" >
<CAPTION>
学习成绩单（小明）
</CAPTION>
  <TR>
    <TH> </TH>
    <TH>高等数学</TH>
    <TH>大学英语</TH>
    <TH>体育</TH>
  </TR>
  <TR>
    <TH>第一学期</TH>
    <TD>85</TD>
    <TD>80</TD>
    <TD>75</TD>
  </TR>
  <TR>
    <TH>第二学期</TH>
    <TD>80</TD>
    <TD>92</TD>
    <TD>80</TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

这个例子的运行效果如图 1.8 所示。

从上例我们可以看出，一个表格的基本标记包括：

（1）<TABLE>：用于标注表格。该标记有 5 个可选的属性：

- align 属性：控制表格本身在页面中的对齐方式，表格本身有 3 种对齐方式，即 left（左对齐）、right（右对齐）、center（中间对齐），默认值是左对齐。
- width 属性：用来设置表格宽度，这个属性的取值可以是以像素为单位的绝对宽度，也可以是百分比（相对于浏览器窗口宽度的百分比）。
- border 属性：设置表格边框的宽度，默认情况下表格是没有框线的。实际上，border 属性只是设置了表格外边框的阴影宽度，而不是表格框线的宽度。

图 1.8 在 HTML 中使用表格

- cellspacing 属性：设置框线的宽度，单位是像素。
- cellpadding 属性：设置表元内容与框线之间的间距，单位是像素。

（2）<CAPTION>：用来标注表格的标题。该标记要紧随在<TABLE>之后，放在第一个 TR 标记之前。

（3）<TR>：在 HTML 中，表格的构造是按行进行的，先定义第一行，然后定义第二行、第三行，依此类推。每个表行都是由 TR 进行封装的。

（4）<TD>和<TH>：用来表示一个表行中的各个单元，这两个标记内部几乎可以包含所有的 HTML 标记，甚至还可以包含嵌套的表格。<TD>标记和<TH>标记的区别在于<TD>标记表示的是普通表元，即表数据，通常是左对齐的；而<TH>标记表示的表元是表头，表头通常以粗体字居中显示。

其实，表格的标记还有很多有用的属性，可以美化表格，使表格多样化，表现形式也不拘一格，在这里就不一一细述了，有兴趣的读者可以参考相关的书籍。

特殊字符集

在 HTML 语言中，还可以使用一些特殊字符，这些特殊字符及含义如表 1.2 所示。

表 1.2 特殊字符及含义

特殊字符	含义
&lt	表示文档内的“ < ”号
&gt	表示文档内的“ > ”号
&amp	表示文档内的“ & ”号
&qout	表示文档内的双引号“ ” ”
	一个非换行空格
&copy	版权符号
&trade	商标符号

以上仅仅列出了部分常用的 HTML 语言标记，由于不是本书的重点内容，不作更详细的讲解。实际上，还有许许多多丰富的标记可以控制网页的外观，有兴趣的读者可以参考相关的参考书目。

3．HTML 中的表单

表单主要用于与用户的交互，表单允许用户输入数据，并将这些数据提交 Web 服务器处理。可以使用标记<FORM></FORM>对来定义表单。具体用法如下：

```
…
<BODY>
…
<FORM action="form_hdl.asp" method="post">
…
</FORM>
…
</BODY>
…
```

其中，属性“ action= ”的值，如 action="form_hdl.asp"，表示这个表单提交后将由 form_hdl.asp 去处理；属性 method 表明了用户输入的数据将以什么样的方式传给服务器，可能的值为 POST 和 GET，默认值为 GET。当使用 GET 方法时，数据将与 URL 一起传送，故其长度受到一定的限制，而 POST 无此局限性。

在一个表单中，可以包括很多的内置控件，这些控件通常是用<INPUT>标记来引入的。通过设置该标记的 type 属性的值来指明所创建控件的类型。常用的控件有下列几种：

（1）文本输入框

文本输入框有两种：其一是单行文本输入框，引入格式为：

```
<INPUT type="text" name="txtname" value="defualt value">
```

属性 type="text"表明这个控件是单行文本输入框，name 属性为该控件起一个名字；value 属性设置了该控件的初始值。

二是文本区域<TEXTAREA>，这是一个可输入多行多列文件的输入区域，引入格式为：

```
< TEXTAREA rows=3 cols=80>default value</textarea>
```

属性 rows 和 cols 分别表示这个多行文本框的行和列，与单行文本框不同的是，它的默认值放在标记<TEXTAREA>和</TEXTAREA>中间。

（2）按钮

按钮有 3 种不同的类型：

- 提交按钮：type=submit
- 重置按钮：type=reset
- 普通按钮：type=button

一般来说，一个表单中都至少包含一个 submit 按钮和一个 reset 按钮。

（3）单选框

单选框也使用< INPUT >标记创建，属性 type 的值为 radio。例如：

```
<INPUT type=radio name=radio1 value="yes" checked>是
<INPUT type=radio name=radio1 value="no" checked>否
```

单选框通常成组出现，同组的单选框的 name 属性值必须相同，任何时刻，一组单选框中只能有一个被选中（checked）。

（4）复选框

复选框用< INPUT >标记创建，属性 type 的值为 checkbox。例如：

```
<INPUT type=checkbox name=checkbox1 value="BeiJing" checked>北京
<INPUT type= checkbox name= checkbox1 value="ShangHai" checked>上海
```

同单选框类似，复选框也通常成组出现，同组的复选框具有相同的 name 值，只是任何时刻都可以有多个复选框被选中。

（5）下拉列表

下拉列表（或叫列表框）用<SELECT></SELECT>标记创建，供选择的项出现在标记对之中，<OPTION>标记表示列表框的选项。例如：

```
<SELECT name=select1 size=1>
    <OPTION selected>程序员</OPTION>
    <OPTION >高级程序员</OPTION>
    <OPTION >系统分析员</OPTION>
</SELECT>
```

name 属性的值是列表框对象的名字；size 属性的值指明在页面上可显示选择项的个数；用 selected 属性设置缺省的选择，用户选中的项可用 value 的返回值得到。

下面我们举一个简单的例子来加深对上述内容的理解。

【例 1.4】 一个包含表单的页面

```
<HTML>
<HEAD>
<TITLE>包含表单的网页</TITLE>
</HEAD>
<BODY>
<FONT size=2>
<FORM action="info_hdl.asp" method="post">
<P>姓名：<INPUT name=name></P>
<P>性别：<INPUT name =sex type=radio selected>男
        <INPUT name =sex type =radio>女</P>
<P>从事行业：<SELECT name =career>
        <OPTION selected>计算机</OPTION>
```

```
            <OPTION >金融</OPTION>
            <OPTION >个体</OPTION>
            <OPTION >其他</OPTION>
            </SELECT></P>
<P>爱好：<INPUT name =interest type =checkbox>体育运动
            <INPUT name =interest type =checkbox>旅游
            <INPUT name =interest type =checkbox>音乐
            <INPUT name =interest type =checkbox>阅读</P>
<P><INPUT name =submit1 type =submit value=提交>
    <INPUT id=reset1 name =reset1 type =reset value=重置>
</P>
</FORM>
</FONT>
</BODY>
</HTML>
```

运行结果如图 1.9 **所示。**

图 1.9　**在** HTML **中使用表单**

1.2　**认识** VBScript

1.2.1　**什么是** VBScript

VBScript **用来控制为** WWW **制作的** HTML **页面中的内容和对象，并与** Internet **相配。** VBScript **不能生成应用程序，它只能生成动态的** HTML**。当然，如果你的页面看上去并且运行起来像应用程序，那是再好不过了。最重要的是当你在** Web **页面中使用** VBScript **时，应该使页面富有活力和交互性。**

VBScript **的全称是** Microsoft Visual Basic Scripting Edition**，它是程序开发语言** Visual

Basic 家族的成员之一，它将灵活的 Script 应用于更广泛的领域，包括 Microsoft Internet Explorer 中的 Web 客户机 Script 和 Microsoft Information Serer 中的 Web 服务器 Script。

VBScript 与 Basic 语言有密切关系，VBScript 语言是 Visual Basic 的一个子集，由微软公司推出，如果你是一个 Visual Basic 程序员，VBScript 可轻松掌握。即使你没有学过 Visual Basic，只要学会 VBScript，就能够使用 Visual Basic 语言进行程序设计。

VBScript 是一种脚本语言。这意味着这种语言比 Basic 语言的成熟版本容易使用又难于使用。编制简单的程序时，脚本语言是容易使用的。脚本语言的句法比较简单，但是，简单的句法也使开发大的应用程序变得很困难。

利用 VBScript，用户可以很方便地制作出各式各样具有强大功能的主页 利用 Microsoft 公司的 ActiveX 技术，用户可以在主页中插入功能各异的 ActiveX 控件，并利用 VBScript 语言对它们进行编程；利用 VBScript 语言结合 ASP 技术，用户可以开发现在最流行的 Web 数据库。总之，拥有了 VBScript，不仅将会使用户的主页变得更加丰富多彩，也会使主页增添了数据库管理和网上交互的能力。

1．客户端和服务器端编程语言

VBScript 既可以作为客户端编程语言，也可以作为服务器端编程语言。客户端编程语言是可以由浏览器解释执行的语言。Java 和 Jscript/JavaScript 也可以作为客户端编程语言。当一个以这些语言中的任意一种编制的程序被下载到一个兼容的浏览器中时，浏览器将自动执行该程序。

客户端编程语言的优点是浏览器完成了所有的工作，这可以减轻服务器的负担。而且客户端程序运行起来比服务器端程序快得多。当一个浏览器的用户执行了一个操作时，不必通过网络对其作出响应，客户端程序就可以作出响应。

但是，一般来说，可以解释 VBScript 脚本的浏览器只有 Microsoft Internet Explorer，而 Netscape Navigator 将忽略 VBScript 脚本。这意味着目前没有充分的理由把 VBScript 作为客户端编程语言。你不应该花费时间来编写只有少数浏览器可以解释的程序。

VBScript 也可以作为服务器端编程语言。服务器端编程语言是在服务器上执行的语言。服务器为一个站点提供文件，而浏览器接收这些文件。服务器端编程语言执行站点主机上的所有操作，所有的功能要你自己来实现。

用 VBScript 作为服务器端编程语言的好处是 VBScript 不受浏览器的限制。VBScript 脚本在网页通过网络传送给浏览器之前被执行，Web 浏览器收到的只是标准的 HTML 文件。

2．VBScript 的特性

（1）VBScript 程序可针对若干个对象

对象可以是控件，也可以是代表浏览器窗口或文档等抽象的事物。

每个对象都有若干特定的属性和方法。对象的属性是用来描述对象的状态的，其中有些属性是只读的，有的则是可以在程序中修改的。对象的方法可以改变对象的状态，也可以控制对象。对象的属性及方法的使用是相同的，如 Document 对象的 bgcolor 属性是用来设定页面背景颜色的，写为 document.bgclor = vbBlack；单行文本框有设置焦点的方法，写为：对象名.focus。

（2）VBScript 采用事件驱动的形式

所谓事件就是用户对对象进行一些操作。如单击按钮将触发按钮对象的 OnClick 事件。每个对象都会有一套事件的集合，如果发生了其中的某个事件，则 VBScript 将执行与该事件相关的程序代码。事件比起其他的程序设计模式有许多优势，最大的益处就是系统始终由用户来控制，而不是由程序来控制，这样程序执行起来就具有很大的灵活性。此外，事件驱动的程序结构还具有以下特点：

- 各种对象事件的发生决定了程序执行的顺序。当用户对屏幕上的不同对象进行操作时，将执行不同的事件处理程序代码段。事件驱动程序是由各种用户操作激发的事件或各种用户操作引发的事件来决定其当前所作的工作的。
- 事件驱动程序是等待事件发生的，在等待的过程中并不占用系统资源。
- 事件处理过程简化了程序的设计。如果没有事件驱动，可能程序要经常去监测每一个输入，这都要程序的设计者来自己设计和完成。
- 事件总是针对某一个对象的，这样当事件发生时，程序就可以准确地转到相应对象的事件处理程序段中。

1.2.2　如何使用 VBScript

可以在 HTML 文件中直接嵌入 VBScript 脚本。这使你能够扩展 HTML，使它不仅仅是一种页面格式语言。带有 VBScript 脚本的网页在每次下载到浏览器时都可以是不同的，而且可以对用户的操作作出反应。

在页面中加入 VBScript 代码非常简单，使用 SCRIPT 标记将 VBScript 代码添加到 HTML 页面中，VBScript 代码写在成对的 <SCRIPT> 标记之间。例如，以下代码为一个将日期转化为特定格式（XXXX 年 XX 月 XX 日）的过程的例子。

【例 1.5】

```
<SCRIPT language="VBScript">
<!--
Function Format_Date(Dt)
Format_Date = Year(Dt) & "年" & Month(Dt) & "月" & Day(Dt) &"日"
End Function
-->
</SCRIPT>
```

因为以上示例是一个通用函数（不依赖于任何窗体控件），所以可以将其包含在页面的 HEAD 部分：

```
<HTML>
<HEAD>
<TITLE>显示日期</TITLE>
<SCRIPT language="VBScript">
<!--
Function Format_Date(Dt)
Format_Date = Year(Dt) & "年" & Month(Dt) & "月" & Day(Dt) &"日"
```

```
End Function
-->
</SCRIPT>
</HEAD>
<BODY>
...
```

VBScript 代码的开始和结束部分都有<SCRIPT>标记。language 属性用于指定所使用的 Script 语言。由于浏览器能够使用多种 Script 语言，所以必须在此指定所使用的 Script 语言，通常客户端的默认脚本语言是 JavaScript，服务器端默认的脚本语言是 VBScript。如果你不想使用默认的脚本语言，而想使用 VBScript 语言，则必须指定 SCRIPT 的属性 language="VBScript"。

提示：脚本代码块必须以</SCRIPT>标记结尾，如果忘了加上这个标记，脚本可能不会被执行，而且不会提示错误信息，这种情况下，你可能不知道发生了什么问题。因为如果你的脚本块比较大，那么这种错误就很难被发现，这样也会浪费大量的时间。

在上面的例子中，我们还使用了注释标记对“ <!-- ”和“ --> ”，这样做的作用是为了防止当用户使用的浏览器不能识别<SCRIPT>标记时破坏页面的效果。许多早期版本的浏览器，例如 Navigator 2.0 以前版本的浏览器就不支持脚本语言。如果浏览器不能支持脚本语言，那么语句代码将作为页面的文本直接显示在页面上，这将极大地破坏整个页面的布局，所以我们需要运用注释标签来解决这个问题。在脚本语句代码外部加上一个注释标签后，对于不能支持脚本的浏览器来说，脚本语句是包含于注释之中的，所以不会显示到页面上，而<SCRIPT>标记和</SCRIPT>标记是不能识别的标签，将会被忽略，在这种情况下，不会对页面的布局产生破坏作用。而支持脚本语言的浏览器却能正确识别位于注释中的脚本语句。

SCRIPT 块可以出现在 HTML 页面的任何地方（BODY 或 HEAD 部分之中）。然而最好将所有的一般目标 SCRIPT 代码放在 HEAD 部分中，以使所有 SCRIPT 代码集中放置。这样可以确保在 BODY 部分调用代码之前所有 SCRIPT 代码都被读取并解码。

上述规则的一个值得注意的例外情况是，在窗体中提供内部代码以响应窗体中对象的事件。例如，以下示例在窗体中嵌入 SCRIPT 代码以响应窗体中按钮的单击事件。

【例 1.6】 一个有 VBScript 的包含表单的 HTML 文件

```
<HTML>
<HEAD>
<TITLE>一个简单的例子</TITLE>
</HEAD>
<BODY>
<FORM NAME="Form1">
<INPUT TYPE="Button" NAME="cmdOK" VALUE="现在什么时间？">
<SCRIPT language="VBScript">
```

```
<--
Sub cmdOK_OnClick()
    MsgBox now
End Sub
-->
</SCRIPT>
</FORM>
</BODY>
</HTML>
```

代码的运行结果是弹出一个对话框，显示当前的日期和时间，如图 1.10 所示。

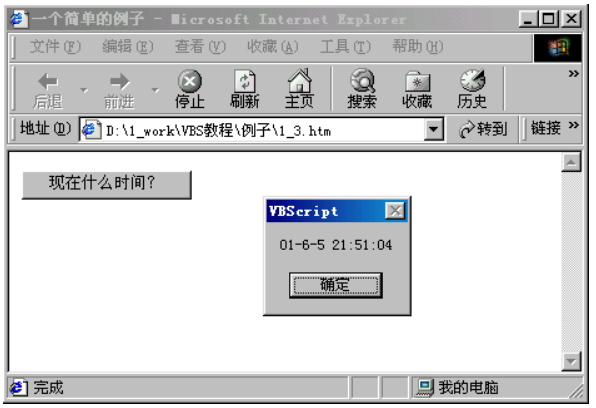

图 1.10　在 HTML 中使用 VBScript

如我们在前面提到的，当用户单击“现在什么时间？”按钮时，将激发按钮 cmdOK 的 OnClick 事件，在它的 OnClick 事件中，我们编写了简单的代码，弹出对话框显示当前的日期和时间。

由于 VBScript 非常灵活，程序设计者可以不必单独为每一个事件生成一个事件处理过程，只要把事件处理的过程当作某元素标记中的一个属性就可以了。例如，一个上述按钮 cmdOK 的单击事件的处理程序段，可以作为该按钮的一个属性。因此，【例 1.6】的代码也可以写成：

```
<HTML>
<HEAD>
<TITLE>一个简单的例子</TITLE>
</HEAD>
<BODY>
<FORM name="Form1">
<INPUT type="Button" name="cmdOK" value="现在什么时间？"
        OnClick='Msgbox Now' Language="VBScript">
</FORM>
</BODY>
```

```
</HTML>
```

上面的例子中，任何地方都没有<SCRIPT>标记。这段程序包含了一个属性，这个属性本身就是一个语句。当激活了按钮 cmdOK 的 OnClick 事件时，程序将执行单引号之间所有的内容。但是这样书写的代码有两个不利之处：首先，它会导致非常长的 HTML 语句，如果在 HTML 的“ < > ”标记之间的语句太长，将会使阅读变得非常困难；此外，这样的代码也不符合模块化的规则，但是，当只有一句语句时，这样使用是比较方便的。

还有一种添加 VBScript 代码的方法，是使用 for/event 属性，我们将【例 1.5】改写为：

```
<HTML>
<HEAD>
<SCRIPT language="VBScript" for = "cmdOK" event="OnClick">
<!--
MsgBox Now
-->
</SCRIPT>
<TITLE>一个简单的例子</TITLE>
</HEAD>
<BODY>
<FORM name="Form1">
<INPUT type="Button" name="cmdOK" value="现在什么时间？">
</FORM>
</BODY>
</HTML>
```

for 和 event 属性分别定义了与这段脚本程序相联系的对象和事件。

当事件处理过程包含许多条语句时，每条语句都放于单引号中，并且语句与语句之间需要用冒号隔开。例如：

```
Onclick='Msgbox "OK!!!":Msgbox "Hello!!!" '
```

要注意的一点是，无论采用哪一种添加 VBScript 代码的方法，都要使用 language 属性来指定所使用的脚本语言。如果在 HTML 文件中没有指定脚本语言的类型，浏览器会默认脚本语言为 JavaScript。在 HTML 文件中一旦指定了脚本语言的类型，则不必再次指定。浏览器将默认为 HTML 文件中最近一次所使用的脚本语言。

大多数 SCRIPT 代码在 Sub 或 Function 过程中，仅在其他代码要调用它时执行。然而，也可以将 VBScript 代码放在过程之外、SCRIPT 块之中。这类代码仅在 HTML 页面加载时执行一次。这样就可以在加载 Web 页面时初始化数据或动态地改变页面的外观。

1.2.3 VBScript 和其他程序语言的差异

我们讲 VBScript 与其他语言的差异，一般是针对 JavaScript 而言的，由于 VBScript 是 Microsoft 公司在风靡全球的 Visual Basic 语言的基础上开发出来的，所以它的语法和风格等方面与 Visual Basic 在很大程度上有相通之处。对于习惯了在 Window 下编程的人来说，

学习 VBScript 是轻而易举的，随着 ASP 技术的流行，VBScript 的应用前景也被看好。

美中不足的是，VBScript 对开发平台有一定的限制，它开发出来的程序并不是所有的浏览器都能支持，比如 Netscape Navigator 就不支持它，而 JavaScript 却具有跨平台性，不受此限制，因此 JavaScript 编写的程序具有很好的可移植性。好在国内的用户绝大多数都在自己的计算机上装有 Internet Explorer，同时，由于它的简单易学，它依然成为众多编程者的首选。

1.3 小　结

在这一章中，我们介绍了一些与 VBScript 密切相关的知识，包括一些相关的概念，如 Web、网页和网站等，本章的重点是 HTML 的基础知识和 VBScript 的基本概念。通过本章的学习，希望读者能够使用 HTML 语言编写简单的网页，并从整体上对 VBScript 有一个认识。从下一章开始我们将重点转入 VBScript，讲述它的基本知识。

练习题

1．HTML 的全称是什么？一个 HTML 文件包括哪几部分，分别代表什么含义？
2．如何在 HTML 中使用表单？可以在表单中引入哪些控件？
3．VBScript 语言实现的动态功能相对于过去的 HTML 静态页面有哪些进步？
4．在 HTML 中如何引入 VBScript 脚本？
5．根据自己的认识，写出 VBScript 的特点。

第 2 章　VBScript 基础

从这一章开始，我们将正式进入 VBScript 的领域。本章中，我们将主要学习以下内容：

- VBScript 的数据类型和 Variant 子类型
- 变量
- 常量
- 运算符和表达式

2.1　数 据 类 型

虽然在 Visual Basic 中有多种数据类型，但在 VBScript 只有一种数据类型，称为 Variant。Variant 是一种特殊的数据类型，它可以根据使用方式的不同而包含不同的信息。因为 Variant 是 VBScript 中唯一的数据类型，所以它也是 VBScript 中所有函数的返回值的唯一数据类型。

虽然这种结构很简单，但 Variant 同样可以包含数字或字符串信息。当 Variant 用于数字上下文中时作为数字处理，当用于字符串上下文中时作为字符串处理。这就是说，如果使用看起来像是数字的数据，则 VBScript 会假定其为数字并以适用于数字的方式处理。与此类似，如果使用的数据只可能是字符串，则 VBScript 将按字符串处理。也可以将数字包含在双引号（" "）中使其成为字符串。

除简单数字或字符串以外，Variant 可以进一步区分数值信息的特定含义。例如使用数值信息表示日期或时间。此类数据在与其他日期或时间数据一起使用时，结果也总是表示为日期或时间。它还可以表示其他各种不同大小的数值数据，例如，可以表示小到 Boolean 值，大到浮点数。数值信息类型是多种多样的，Variant 包含的数值信息类型称为子类型。

2.2　Variant 子类型

大多数情况下，可将所需的数据放进 Variant 中，而 Variant 也会按照最适用于其包含的数据的方式进行操作。

Variant 包含的数据子类型有下列几种：

1．空（Empty）

未初始化的 Variant。对于数值变量，值为 0；对于字符串变量，值为零长度字符串（""）。

2．无效类型（Null）

不包含任何有效数据的 Variant。

3．布尔型（Boolean）

包含 True 或 False。Boolean 类型在内存中占据很小的空间。

4．字节型（Byte）

包含 0 到 255 之间的整数。在内存中也只占很小的空间。

5．整型（Integer）

在内存中占据两个字节，包含－32,768 到 32,767 之间的整数。存储更多的字节将使变量可以取相当大范围的值。

6．货币类型 Currency

－922,337,203,685,477.5808 到 922,337,203,685,477.5807。

7．长整数（Long）

占据 4 个字节，包含－2,147,483,648 到 2,147,483,647 之间的整数。

8．单精度浮点类型（Single）

包含单精度浮点数。精度指的是分配给变量的小数部分的字节数。单精度数分配了 2 个字节的小数部分。Single 占据 4 个字节，数值范围是：

负数：从－3.402823E38 到－1.401298E－45。

正数：从 1.401298E－45 到 3.402823E38。

9．双精度浮点类型（Double）

包含双精度浮点数，占据 8 个字节，其中 4 个字节用来存储小数部分。Double 子类型，已经相当精确了，所以常用来为高等数学和科学计算服务。数值的范围是：

负数：从－1.79769313486232E308 到－4.94065645841247E－324

正数：从 4.94065645841247E－324 到 1.79769313486232E308。

10．日期类型（Date（Time））

包含表示日期的数字，日期范围从公元 100 年 1 月 1 日到公元 9999 年 12 月 31 日。

11．字符串类型（String）

包含变长字符串，最大长度可为 20 亿个字符。变量的大小取决于字符串的长度。

12．对象类型（Object）

包括了一个 OLE Automation 对象名。对象可以通过这个变量来操纵。

13．错误编号类型（Error）

包含了一个错误号。可以使用这个产生的错误号来对这个错误进行解释。

注意：对于许多人来说，无效类型（Null）不太好理解，它不像其他几种类型那样直观，一定不要把无效类型和数字类型中的 0 或者字符串类型中的空字符串混淆起来。Null 代表什么也没有，代表的是变量中没有存储有效的数据，而且也谈不上它是属于哪一种类型的。数据类型的 0 代表的是一个数值，空字符串代表的是字符串的长度为 0，这两个值都是有效的。

如果我们需要查看变量的数据子类型，可以使用 VarType 函数获取。例如，我们定义了一个变量 a，并且将其赋值为 5，那么我们可以使用以下语句获取其类型：

```
VarType(a)
```

这个函数执行之后将返回 2，表明这个变量的子类型是整数类型。在表 2.1 中列出了使用 VarType 函数获取的各种数据子类型变量的返回值。

表 2.1 VarType 函数获取的数据子类型与返回值对照表

数据子类型	返回值
Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6
Date(Time)	7
String	8
Object	9
Error	10
Boolean	11
Variant	12
Dataobject	13
Byte	17
Array	8192

另外，可以使用转换函数来转换数据的子类型。这一点我们将在第 5 章“内部函数”中详细讲解。

2.3 变 量

2.3.1 什么是变量

其值可以改变的量称为变量。一个变量应该有一个名字，在内存中占据一定的存储单元。使用脚本语言时，变量是其中最基本的元素，脚本执行过程中，往往需要一个单元将信息存储起来，变量就是这样的一个命名的存储单元，存储在这个单元中的数据就是变量的值。请注意变量名和变量值是两个不同的概念。

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运行时可更改的程序信息。例如，在一个网上购书的网页中可以创建一个名为 BookCount 的变

量来存储用户购买某本图书的数量，这个变量的值是存储在计算机内存中的，在使用的过程中，我们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 BookCount 引用变量就可以查看或更改变量的值。因为在 VBScript 中只有一个基本数据类型，即 Variant，因此所有变量的数据类型都是 Variant。

2.3.2　声明变量

使用变量前，一般要先声明变量。

声明变量也就是定义变量，在使用变量以前，不一定非要声明，在 VBScript 中，不声明而直接使用变量，称为对变量的隐式声明。但是我们并不赞成使用这种方法，最好使用 Option Explicit 语句显式声明所有变量。

1．Option Explicit

Option Explicit 语句强制要求显式声明脚本中使用到的所有变量，在使用了这条语句之后，所有将要用到的变量都必须在使用前进行声明。注意 Option Explicit 语句必须放在 HTML 文本和脚本命令之前，也就是它必须作为页面的起始语句之一。

使用 Option Explicit 显式地声明变量是一个良好的编程习惯，它对于我们调试程序有很大的帮助。因为在编写程序的过程中，我们很有可能拼错了变量的名称，而产生不可预期的执行结果，并且这种错误往往是难以发现的。因此我们建议读者在 Script 的开头都放上 Option Explicit 语句。

下面举例说明如何使用 Option Explicit 语句。

```
Option Explicit              '强制显示声明变量
Dim MyVarName                '声明变量
MyIntName = 10               '未声明变量产生错误
MyVarName = 10               '声明变量不产生错误
```

2．Dim 和 ReDim

我们一般使用 Dim 语句来声明变量。语法如下：

```
Dim varname[([subscripts])]
```

其中，varname 是变量的名称，在声明数组的时候，subscripts 代表的是数组的上界，例如：

```
Dim MyStr(10)
```

对于这个数组，数组的上界是 10，而下界默认是 0，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。这种数组被称为静态数组。

在数组中使用索引为数组的每个元素赋值。从 0 到 10，将数据赋给数组的元素，如下所示：

```
MyStr (0) = 100
MyStr (1) = 200
```

```
MyStr (2) = 300
. . .
MyStr (10) = 501
```

与此类似，使用索引可以检索到所需的数组元素的数据。例如：

```
. . .
SomeVariable = MyStr (2)
. . .
```

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，theTable 变量是一个有 5 行和 6 列的二维数组：

```
Dim theTable(4, 5)
```

在二维数组中，括号中第一个数字表示行的数目，第二个数字表示列的数目。

也可以声明动态数组，即在运行脚本时才确定数组的大小。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 theArray()
ReDim AnotherArray()
```

要使用动态数组，必须随后使用 ReDim 确定维数和每一维的大小。在下例中，ReDim 将动态数组的初始大小设置为 10，而后面的 ReDim 语句将数组的大小重新调整为 15，同时使用 Preserve 关键字在重新调整大小时保留数组的内容。

```
ReDim theArray(10)
. . .
ReDim Preserve theArray(15)
```

重新调整动态数组大小的次数是没有任何限制的，将数组的大小调小时，将会丢失被删除元素的数据。

当声明多个变量时，使用逗号分隔变量。例如：

```
Dim var1,var2,var3
```

3．Public 语句

使用 Dim 语句可以在脚本的过程中声明变量，也可以在过程外声明变量，在过程中声明的变量称为过程级变量，在过程外声明的变量称为脚本级变量，过程级变量只能应用于过程中，脚本级变量可以应用于脚本中所有的过程。关于过程，我们将会在第 4 章中详细介绍。

Public 语句用来声明脚本级的变量，运用 Public 声明的变量可以运用于所有项目的全部脚本中，其语法如下：

```
Public varname[([subscripts])]
```

其中，varname 代表的是变量名称，subscripts 代表的是数组的上界。和 Dim 语句一样，Public 语句也能够声明动态数组和静态数组，以及多维数组。

4．Private 语句

Private 语句是和 Public 语句相对而言的，Private 语句也只能声明脚本级变量，运用 Private 语句声明的变量只能在声明该变量的脚本中使用（Pubic 是用于所有项目）。其语法如下：

```
Private varname[([subscripts])]
```

其中，varname 代表的是变量名称，subscripts 代表的是数组的上界。和 Dim 语句一样，Private 语句也能够声明动态数组和静态数组，以及多维数组。

2.3.3　变量的命名规则

变量命名必须遵循 VBScript 的标准命名规则。具体规则内容如下：

（1）第一个字符必须是字母，如 FristName 是合法的变量名称，而_FristName、1FristName 是非法的变量名称。注意和其他语言的区别，在 VBScript 中，变量名是不能以下划线“ _ ”开头的。

（2）不能包含嵌入的句点，如 Frist.Name 是非法的变量名称。

（3）长度不能超过 255 个字符。

（4）在定义的有效范围内必须唯一。

变量的作用域是由变量声明的位置决定的，如果是在过程中声明的变量，则变量的作用域是整个过程；如果是在过程外声明的变量，则变量的作用域是整个脚本。也就是说，如果在相同的作用域内出现了相同名称的两个或者多个变量，将会出错。不过在两个过程中，将变量命名为相同的名称是可以的。例如，在两个不同的过程中，我们都需要用到一个临时的变量，那么可以将这个临时变量都命名为 temp。

虽然在为变量起名字的时候，没有强制性的要求，但最好为变量起一个有意义的名字，我们强烈建议读者这么做，养成一个良好的编程习惯，这将提高脚本的可读性，方便开发人员进行脚本的编写和调试。

2.3.4　给变量赋值

可以创建如下形式的表达式来给变量赋值，其中变量在表达式左边，要赋的值在表达式右边。例如：

```
aNum = 100
```

2.3.5　变量的作用域与生命周期

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中所有过程所识别，称为 Script 级变量，具有脚本级作用域。

变量存在的时间称为生命周期。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。

2.4 常 量

2.4.1 什么是常量

在程序运行过程中，其值不能被改变的量称为常量。常量是具有一定含义的名称，用于代替数字或字符串，其值从不改变。VBScript 定义了许多内部常量，在脚本中可以引用。

2.4.2 创建常量

可以使用 Const 语句在 VBScript 中创建用户自定义常量。使用 Const 语句可以创建名称具有一定含义的字符串型或数值型常量，并给它们赋原义值。例如：

```
Const MyName = "Zhao caihua"
Const MyAge = 24
```

请注意字符串文字包含在两个引号（" "）之间。这是区分字符串型常量和数值型常量的最明显的方法。日期文字和时间文字包含在两个井号（#）之间。例如：

```
Const StartDate = #10-6-2001#
```

最好采用一个命名方案以区分常量和变量，这样可以避免在运行脚本时对常量重新赋值。例如，可以使用“ vb ”或“ con ”作常量名的前缀，或将常量名的所有字母大写。将常量和变量区分开可以在开发复杂的脚本时避免混乱。

2.4.3 VBScript 中的内部常量

内部常量指的是由应用程序提供的常量。由于不能禁用内部常量，因此不能以与内部常量相同的名称创建用户自定义常量。表 2.2 给出了 VBScript 内部常量的分类列表。详细信息请参见附录 A。

表 2.2 VBScript 常量分类

语言元素	描述
颜色常量	颜色常量列表
比较常量	用于比较操作的常量列表
日期和时间常量	定义一周中日期以及用于日期和时间操作的常量列表
日期格式常量	用于格式化日期和时间的常量的列表
Locale ID (LCID) 表	区域 ID 及其相关值的列表
杂项常量	不属于任何类别的常量的列表

（续表）

语言元素	描述
MsgBox 常量	与 MsgBox 函数一起使用的常量列表
字符串常量	字符串常量列表
Tristate 常量	可在程序的任何地方使用，表示说明值
VarType 常量	定义 Variant 子类型的常量的列表

2.5　运算符和表达式

2.5.1　运算符

VBScript 有一套完整的运算符，包括算术运算符、比较运算符、连接运算符和逻辑运算符。如表 2.3 所示。

表 2.3　VBScript 的运算符

分类	描述	符号
算术运算符	求幂	^
	负号	-
	乘	*
	除	/
	整除	\
	求余	Mod
	加	+
	减	-
连接运算符	字符串连接	&
比较运算符	等于	=
	不等于	<>
	小于	<
	大于	>
	小于等于	<=
	大于等于	>=
	对象引用比较	Is
逻辑运算符	逻辑非	Not
	逻辑与	And
	逻辑或	Or
	逻辑异或	Xor
	逻辑等价	Eqv
	逻辑隐含	Imp

2.5.2 表达式

用运算符和括号将运算对象（或称为操作数，可能是常量、变量或字符串和数）连接起来就构成了表达式。各运算符要求的运算对象的类型，遵从的运算法则和大多数程序设计语言一致，比如“ & ”运算符用于串接字符串，“ + ”、“ － ”、“ * ”、“ / ”等要求运算量是数字，而“ Not ”、“ Or ”、“ And ”和“ Eqv ”等要求逻辑值等等。计算结果一般是同类型的值，也就是表达式的类型。运算遵从一定的优先、交换和结合法则，并可以用括号改变优先次序等。不相容的数据类型不能进行运算，要先用系统内部转换函数转换成相容。赋值语句的等号右边可以是表达式，子过程和函数的参数可以用同类型的表达式。

2.5.3 运算符优先级

VBScript 语言规定了运算符的优先级，在表达式求值时，先按运算符的优先级别高低次序执行，例如先乘除后加减。

可以使用括号越过这种优先级顺序，强制首先计算表达式的某些部分。运算时，总是先执行括号中的运算符，然后再执行括号外的运算符。但是，在括号中仍遵循标准运算符优先级。

当表达式包含多种运算符时，首先计算算术运算符，然后计算比较运算符，最后计算逻辑运算符。所有比较运算符的优先级相同，即按照从左到右的顺序计算比较运算符。算术运算符和逻辑运算符的优先级如下所示：

- 当乘号与除号同时出现在一个表达式中时，按从左到右的顺序计算乘、除运算符。同样当加与减同时出现在一个表达式中时，按从左到右的顺序计算加、减运算符。
- 字符串连接（&）运算符不是算术运算符，但是在优先级顺序中，它排在所有算术运算符之后和所有比较运算符之前。Is 运算符是对象引用比较运算符。它并不比较对象或对象的值，而只是进行检查，判断两个对象引用是否引用同一个对象。

2.6 小 结

本章从 VBScript 的数据类型入手，讲述了 VBScript 的基础知识，主要内容包括数据类型、变量、常量、运算符和表达式。

有了这些必备知识，我们将在下一章开始讨论 VBScript 的语句。

练习题

1．请将 VBScript 语言的数据类型和其他程序的数据类型作比较，VBScript 有哪些特点？

2．使用 Option Explicit 显式地定义变量有什么好处?

3．VBScript 中有几种数据类型？是什么？

4．下面哪些变量名是正确的？

（1）strvar1　（2）1strvar　（3）_strvar　（4）strvar.1

5．变量的作用域分为哪几种？在用法上有何不同？

6．请指出下列语句存在的错误

（1）

```
Dim r
Const PI = 3.14
r = 1
Function Area1(r)
    Area1 = PI * r*r
End Function
PI = 3.14059
Function Area2(r)
    Area2 = PI * r*r
End Function
```

（2）

```
Option Explicit
Dim OutNameStr
Sub msg()
    Dim MyName
    MyName = "Marry"
    Msgbox MyName
End Sub
  OutNameStr="我的名字叫"& MyName
```

7．求下列表达式的值

（1）x+z%3*(x+y)%5/2

设 x = 3.5, z = 8, y = 5.7

（2）(a+b)/2+x mod 3

设 a = 2, b = 3, x = 53。

第 3 章　语句及基本语法

语句规定程序中运算数据的顺序和方法。对于一般的程序，有 3 类基本的程序结构可以控制程序的流程，它们是顺序结构、分支结构和循环结构。有了这 3 种结构，任何复杂的计算都可以完成。

顺序结构的流程是按照事务完成的先后次序依次执行语句，即总是在完成前一句语句之后再执行后一句语句，并且执行过的语句不再执行。

分支结构应用于根据表达式不同的值决定所进行的操作，分支结构是指可以存在多种选择的结构。

循环结构用于对某一些语句进行反复执行，当符合（或不符合）某些条件时，循环语句执行结束。

一般来说，大多数程序总是同时包含这 3 种结构。其中，顺序结构最简单，一般不需要程序进行什么控制；分支结构可以使用条件语句来实现，循环可以使用循环语句来实现。本章我们将重点讲解条件语句和循环语句，对其他常用语句也作了一些介绍。

3.1　条 件 语 句

在 VBScript 中有两种条件语句：If…Then…Else 语句和 Select…Case 语句，执行这两条语句的时候，先对条件进行判断，然后根据条件执行相应的脚本。

3.1.1　If…Then…Else 语句

根据表达式的值有条件地执行一组语句，语法如下：

```
If condition Then statements [Else elsestatements ]
```

或者，使用块形式的语法：

```
If condition Then
[statements]
[ElseIf condition-n Then
[elseifstatements]] …
[Else
[elsestatements]]
End If
```

参数说明：

- condition：是数值或字符串的表达式，运算结果是布尔型的 True 或 False，如果

condition 是 NULL，则被视为 False。

- Statements：如果 condition 为 True 时，执行的一条或多条（以冒号分开）语句。
- condition-n：同 condition。
- elseifstatements：如果相关的 condition-n 为 True 时，执行的一条或多条语句。
- elsestatement：如果前面没有 condition 或 condition-n 表达式为 True 时，执行的一条或多条语句。

对于短小简单的测试，可以使用单行形式（第一种语法）。但块形式（第二种语法）提供了比单行形式更强的结构化与适应性，比较容易阅读、维护及调试。在多数情况下，我们建议使用第二种方法。

语法说明:

当程序运行到 If 块（第二种语法）时，将首先测试 condition。如果 condition 的值是 True，则执行 Then 之后的语句。如 condition 是 False，则每个 ElseIf 部分的条件式（如果有的话）会依次计算并加以测试。当找到某个为 True 的条件时，则其相关的 Then 之后的语句会被执行。如没有一个 ElseIf 语句是 True（或没有 ElseIf 子句），则将执行 Else 之后的语句。执行 Then 或 Else 之后的语句以后，将继续执行 End If 之后的语句。

Else 和 ElseIf 子句都是可选项。在 If 块中可以放置任意多个 ElseIf 子句，但是都必须在 Else 子句之前。If 块语句可以被嵌套，即被包含在另一个 If 块语句之中。

If 块语句必须是某一行的第一条语句，并且必须以 End If 语句结束。

注意：在单行语法中，可以执行多条语句作为 If...Then 判断的结果，但所有语句必须在同一行上并且以冒号分开，如下列语句所示：

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

下面我们将举例说明 If…Then…Else 语句的用法。

【例 3.1】　If…Then…Else 语句的应用

```
<HTML>
<HEAD>
<SCTIPT language="VBScript">
<!--
Sub Hello_OnClick()
'取当前时间的小时数
   NowHour = Hour(Now)
'当前时间小时数在 0 到 8（包括 0 和 8）之间
If NowHour >= 0 and NowHour <= 8 then
      Msgbox "亲爱的用户，早上好，起这么早，真是个勤快的人！"
'当前小时数在 8 到 12（包括 12）之间
   Elseif NowHour > 8 and NowHour < =12 then
      Msgbox "亲爱的用户，上午好！"
'当前小时数在 12 到 18（包括 18）之间
```

```
    Elseif NowHour > 12 and NowHour <= 18 then
        Msgbox "亲爱的用户，下午好！"
'当前小时数在 18 到 24（包括 24）之间
    Elseif NowHour > 18 then
        Msgbox "亲爱的用户，晚上好！这么晚了，还不睡觉，要注意身体哟！"
    End if
End Sub
-->
</SCRIPT>
<TITLE>条件语句 If...Then...Else</TITLE>
</HEAD>
<BODY>
    这个例子所实现的功能是根据现在的时间，确定给计算机前的用户以什么样的提示。
<P align=center >
<INPUT id=button1 name=Hello type=button value=单击这里>
</P>

</BODY>
</HTML>
```

运行效果如图 3.1 所示。

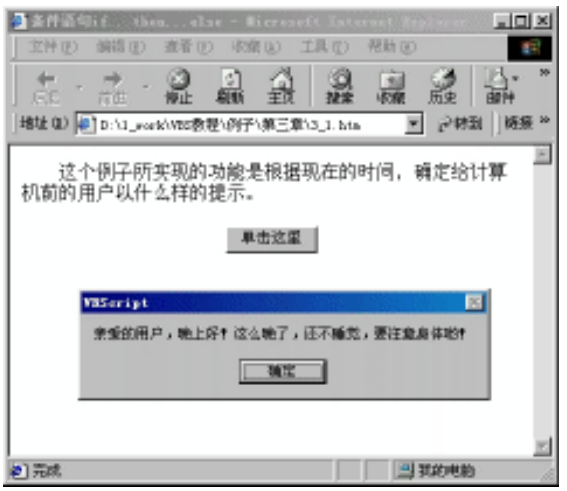

图 3.1　If…Then…Else 语句示例

这个例子所实现的功能是根据当前的系统时间，确定给计算机前的用户以什么样的提示。

首先，使用系统函数 Now 得到当前的时间，然后用系统函数 Hour 取得当前时间的小时，之后进入 If 块进行判断，如果当前小时数在 0 到 8 之间则视为早上，提示用户“亲爱的用户，早上好，起这么早，真是个勤快的人！”，如果这个条件不成立，也就是当前小时数不在 0 到 8（包括 0 和 8）之间，则判断是否在 8 到 12（包括 12）之间，以此类推，将

一天的时间分为 4 个时间段，根据不同的时间段给出不同的提示。

3.1.2　Select Case 语句

Select Case 语句是多分支语句。If…Then…Else 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个分支的选择。例如学生成绩分类（90 分以上为“A”等，80~89 为“B”等，70~79 为“C”等，60~69 为“D”等，60 分以下为“E”等）；人口统计分类（按年龄分为老、中、青、少、儿童）……等等。当然这些都可以用嵌套的 If 语句来处理，但是当分支很多的时候，则嵌套的 If 语句层数变得很多，程序冗长而且可读性降低。一般当分支达到 4 层以上时，选择 Select Case 当是明智的选择。

根据表达式的值执行几组语句之一，语法如下：

```
Select Case testexpression
[Case expressionlist-n
[statements-n]] . . .
[Case Else expressionlist-n
[elsestatements-n]]
End Select
```

参数说明：

- testexpression：任意数值或字符串表达式。
- expressionlist-n：如 Case 出现则为必选项。是一个或多个表达式的分界列表。
- statements-n：当 testexpression 与 expressionlist-n 中的任意部分匹配时，执行的一条或多条语句。
- elsestatements-n：当 testexpression 与 Case 子句的任何部分不匹配时，执行的一条或多条语句。

如果 testexpression 与任何 Case expressionlist 表达式匹配，则执行此 Case 子句和下一个 Case 子句之间的语句，对于最后的子句，则会执行该子句到 End Select 之间的语句，然后控制权会转到 End Select 之后的语句。如 testexpression 与多个 Case 子句中的 expressionlist 表达式匹配，则只有第一个匹配后的语句被执行。

Case Else 用于指示若在 testexpression 和任何其他 Case 选项的 expressionlist 之间未找到匹配，则执行 elsestatements。这并不是必要的，但最好是将 Case Else 语句置于 Select Case 块中以处理不可预见的 testexpression 值。如果没有 Case expressionlist 与 testexpression 匹配且无 Case Else 语句，则继续执行 End Select 之后的语句。

Select Case 语句可以是嵌套的，每一层嵌套的 Select Case 语句必须有与之匹配的 End Select 语句。

下面举例说明如何使用 Select Case 语句。

【例 3.2】　Select Case 语句的应用

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

```
<TITLE>条件语句 select case</TITLE>
</HEAD>
<BODY>

<P>  
这个例子，用来改变主页的背景颜色，您只要在菜单中选定您喜欢的颜色即可。</P>
<P align=center>
<FORM name="frmColor">
<SELECT name=cboColor>
   <OPTION value ="绿色" selected>绿色</OPTION>
   <OPTION value ="黑色" >黑色</OPTION>
   <OPTION value ="红色">红色</OPTION>
   <OPTION value ="蓝色">蓝色</OPTION>
   <OPTION value ="黄色">黄色</OPTION>
</SELECT></P>
<SCRIPT Language="VBScript">
<!--
Sub cboColor_OnClick()
'按照选中的下拉列表框中的选项设置主页的背景颜色
    Select Case frmColor.cboColor.value
'当用户选中黑色时，设置背景颜色为黑色
    Case "黑色"
        document.bgColor = vbBlack
'当用户选中绿色时，设置背景颜色为绿色
    Case "绿色"
        document.bgColor = vbGreen
'当用户选中红色时，设置背景颜色为红色
    Case "红色"
        document.bgColor = vbBlue
'当用户选中蓝色时，设置背景颜色为蓝色
    Case "蓝色"
        document.bgColor = vbRed
'当用户选中黄色时，设置背景颜色为黄色
    Case "黄色"
        document.bgColor = vbYellow
    End Select
-->
End Sub
</SCRIPT>
</FORM>
</BODY>
</HTML>
```

运行效果如图 3.2 所示。

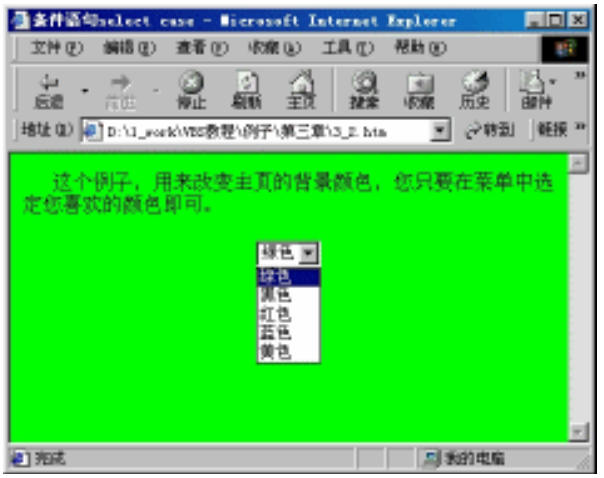

图 3.2　Select Case 语句示例

当用户在下拉列表框中选择颜色时，将触发下拉框的单击（OnClick）事件，在该事件编写代码，实现根据用户的不同选择，设置不同的页面背景颜色。

3.2　循 环 语 句

循环用于重复执行一组语句。循环可分为 3 类：一类在条件变为 False 之前重复执行语句，一类在条件变为 True 之前重复执行语句，另一类按照指定的次数重复执行语句。

在 VBScript 中可使用下列循环语句：

- 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：对于一个集合中的每项或数组中的每个元素，重复执行一组语句。

下面将分别详细介绍这 4 种循环语句。

3.2.1　Do…Loop 循环

可以使用 Do...Loop 语句多次（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。

（1）当条件为 True 时重复执行语句，语法如下：

```
Do While condition
    statements
Loop
```

或

```
Do
    statements
Loop While condition
```

While **关键字用于检查条件** condition **的值。有两种方式检查条件：在进入循环之前检查条件（如第一种语法所示）；或者在循环至少运行完一次之后检查条件（如第二种方法所示）。在第一种语法中，如果** condition **的值为** False**，则永远不会执行循环体中的语句，而直接转到循环体的外边执行其他语句。在第二种语法中，循环体中的语句至少执行一次，因为它是在执行完一次之后才进行判断。请比较下面的两个例子。**

【例 3.3**】 在循环之前检查条件的例子（使用** While **关键字）**

```
Sub ExecuteBeforChkByWhile ()
    Dim counter, myNumber
    counter = 0
    myNumber = 50
    Do While myNumber > 20
        myNumber = myNumber - 1
        counter = counter + 1
    Loop
    MsgBox "循环重复了 " & counter & " 次。"
End Sub
```

本例中总共执行了 30 **次循环，即** counter **的值为** 30**。**

【例 3.4**】 在循环之后检查条件的例子（使用** While **关键字）**

```
Sub ExecuteAfterChkByWhile()
    Dim counter, myNumber
    counter = 0
    myNumber = 50
    Do
        myNumber = myNumber - 1
        counter = counter + 1
    Loop While myNumber > 20
    MsgBox "循环重复了 " & counter & " 次。"
End Sub
```

在【例 3.4**】中，也执行了** 30 **次循环，即** counter **的值也是** 30**，但是假设我们将变量** myNumber **的初始值设为** 19**，则【例** 3.3**】的** ExecuteBeforChkByWhile **过程将一次也不执行，而【例** 3.4**】的** ExecuteAfterChkByWhile **在执行一次之后退出循环。因此这两个例子得到的结果不再一致，请读者在使用时注意。**

（2**）重复执行语句直到条件变为** True**，语法如下：**

```
Do Until condition
```

```
    statements
Loop
```

或

```
Do
    Statements
Loop Until condition
```

Until 关键字用于检查条件 condition 的值。有两种方式检查条件：在进入循环之前检查条件（如第一种语法所示）；或者在循环至少运行完一次之后检查条件（如第二种语法所示）。只要条件为 False，就会进行循环。请比较下面两个例子。

【例 3.5】　在进入循环之前检查条件的例子（使用 Until 关键字）

```
Sub ExecuteBeforChkByUntil()
    Dim counter, myNumber
    counter = 0
    myNumber = 50
    Do Until myNumber = 20
        myNumber = myNumber - 1
        counter = counter + 1
    Loop
    MsgBox "循环重复了 " & counter & " 次。"
End Sub
```

本例，总共执行了 30 次循环，即 counter 的值为 30。

【例 3.6】　在进入循环之后检查条件的例子（使用 Until 关键字）

```
Sub ExecuteAfterChkByUntil ()
    Dim counter, myNumber
    counter = 0
    myNumber = 50
    Do
        myNumber = myNum - 1
        counter = counter + 1
    Loop Until myNumber = 20
    MsgBox "循环重复了 " & counter & " 次。"
End Sub
```

与【例 3.5】的执行结果相同，循环也同样执行了 30 次，counter 的值为 30，但是当把 myNumber 的值设为 20 时，则【例 3.5】中的 ExecuteBeforChkByUntil 根本不进入循环体内，counter 的值仍为 0，而【例 3.6】中的 ExecuteAfterChkByUntil 在执行一次循环后，将 counter 置为 1 后退出循环。它们的执行结果是不一样的。

（3）退出循环

Exit Do 语句用于退出 Do...Loop 循环。因为通常只是在某些特殊情况下要退出循环（例

如为了避免死循环），所以我们只要在 If...Then...Else 语句的 True 语句块中使用 Exit Do 语句即可。如果条件为 False，循环将会照常运行。

在下面的示例中，myNum 的初始值将导致死循环。If...Then...Else 语句检查此条件，以防止出现死循环。

【例 3.7】 使用 Exit Do 防止出现死循环的例子

```
Sub ExitDoExample()
    Dim counter, myNumber
    counter = 0
    myNumber = 19
    Do Until myNumber = 20
        myNumber = myNumber - 1
        counter = counter + 1
        If myNumber < 0 Then
    Exit Do
        End if
    Loop
    MsgBox "循环重复了 " & counter & " 次。"
End Sub
```

如果在该循环体内没有 If…Then..Else 语句，则在 myNumber 的初始值 19 上进行减 1 的运算，以 myNumber = 20 的条件作为退出循环的限制是不可能实现的，如此将导致死循环。这只是一个简单的例子，很容易看出问题所在，但在实际的应用中，一些不太直观的条件往往会导致死循环的出现，希望能够引起读者的重视。

3.2.2 While…Wend 循环

当指定的条件为 True 时，执行一系列的语句，语法如下：

```
While condition
    statements
Wend
```

参数说明：

- condition：数值或字符串表达式，其计算结果为 True 或 False。如果 condition 为 Null，则 condition 被当作 False。
- statements：在条件为 True 时执行的一条或多条语句。

如果 condition 为 True，则 statements 中所有 Wend 语句之前的语句都将被执行，然后控制权将返回到 While 语句，并且重新检查 condition。如果 condition 仍为 True，则重复执行上面的过程；如果不为 True，则从 Wend 语句之后的语句继续执行程序。

While...Wend 循环可以是多层嵌套结构，每个 Wend 与最近的 While 语句对应。

下面举例说明如何使用 While...Wend 语句。

【例 3.8】　While…Wend 的使用

```
Dim Counter
Counter = 0                          ’初始化变量。
While Counter < 10                   ’测试计数器的值。
    Counter = Counter + 1            ’增加计数器。
    Alert   (“这是第” & Counter & “次循环”)
Wend                                 ' 计数器大于 9 时终止循环。
```

3.2.3　For…Next 循环

以指定次数重复执行一组语句，语法如下：

```
For counter = start To end [Step StepSize]
[statements]
[Exit For]
[statements]
Next
```

参数说明:

- counter：用做循环计数器的数值变量。这个变量不能是数组元素或用户自定义类型的元素。
- start：counter 的初值。
- end：counter 的终值。
- StepSize：counter 的步长。如果没有指定，则 Step 的默认值为 1。
- statements：For 和 Next 之间的一条或多条语句，将被执行指定次数。

StepSize 参数可以是正数，也可以是负数。StepSize 参数值决定循环的执行情况，如表 3.1 所示。

表 3.1　For…Next 语句的 StepSize 参数值与执行结果的对应关系

值	当下列条件成立时循环执行
正数或 0	counter <= end
负数	counter >= end

当循环启动并且所有循环中的语句都执行后，StepSize 值被加到 counter 中。这时，如果 counter 的值仍然在 start 和 end 之间（包括 start 和 end），则循环中的语句再次执行，否则退出循环并从 Next 语句之后的语句继续执行。

注意：在循环体内改变 counter 的值，将会使程序代码的阅读和调试变得更加困难。

Exit For 只能用于 For Each...Next 或 For...Next 结构中，提供另一种退出循环的方法，可在语句中的任意位置放置任意个 Exit For 语句。Exit For 经常和条件判断语句一起使用（例

如 If...Then)，并立即将控制权转移到 Next 之后的语句。

下面举例说明 For…Next 的用法。

【例 3.9】 For…Next 的使用

```
Dim I,nResult
NResult = 0
For I = 1 to 50
    NResult = nResult + I
Next
Msgbox "计算结果为：" & nResult
```

该实例计算从 1~50 所有的数值相加的结果。这个 For 语句指定了一个计数器 I，同时指定了该计数器的初值 1 和终值 50。Next 语句使计数器的值在每次执行时自动增 1。

使用 Step 关键字，可以根据自己的需要来递增或递减计数器变量，在下面的例子中，计数器变量 I 会在每次循环执行后递增 2。当循环执行完以后，变量 nResult 是从 1~50 中的数值中的奇数的和。

【例 3.10】 使用 Step 关键字的 For…Next 循环

```
Dim I,nResult
NResult = 0
For I = 1 to 50 step 2
    NResult = nResult + I
Next
Msgbox "计算结果为：" & nResult
```

可以将一个 For...Next 循环放置在另一个 For...Next 循环中，组成嵌套循环。每个循环中的 counter 要使用不同的变量名。下面的结构是正确的：

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
        . . .
        Next
    Next
Next
```

3.2.4 For Each…Next 循环

与以上 3 个循环语句不同，For Each…Next 循环语句既不指定循环的条件，也不指定循环的次数。而是对数组或集合中的每个元素重复执行一组语句，当对数组或集合中的元素都遍历一遍之后，就退出循环。语法如下：

```
For Each element In group
[statements]
[Exit For]
```

```
[statements]
Next [element]
```

参数说明：

- element：用来枚举集合或数组中所有元素的变量。对于集合，element 可能是 Variant 变量、通用 Object 变量或任意指定的 Automation 对象变量。对于数组，element 只能是 Variant 变量。
- group：对象集合或数组的名称。
- statements：对于 group 中的每一项执行的一条或多条语句。

如果 group 中有至少一个元素，就会进入 For Each 块执行。一旦进入循环，便首先对 group 中第一个元素执行循环中的所有语句。只要 group 中还有其他的元素，就会对每个元素执行循环中的语句。当 group 中没有其他元素时退出循环，然后从 Next 语句之后的语句继续执行。

可在循环的任意位置放置任意个 Exit For 语句，Exit For 经常和条件判断语句一起使用（例如 If...Then），并立即将控制权转移到 Next 之后的语句。

可以将一个 For Each...Next 循环放置在另一个之中，组成嵌套式 For Each...Next 循环。但是每个循环的 element 必须是唯一的。

注意：如果省略 Next 语句中的 element，则程序仍会象已包含它一样继续执行。
如果 Next 语句在其相应的 For 语句之前出现，则会产生错误。

下面举例说明如何使用 For Each...Next 语句。

【例 3.11】　For Each…Next 的使用

```
<HTML>
<HEAD>
<SCRIPT Language="VBScript">
Sub button1_OnClick()
Dim Arr1(5)
Dim i
i = 0
Arr1(0) = "游泳"
Arr1(1) = "音乐"
Arr1(2) = "文学"
Arr1(3) = "旅游"
Arr1(4) = "交友"
Arr1(5) =   "购物"

For Each element in Arr1
    Msgbox element
    document.frm_interest.elements(i).value = element
    i = i + 1
```

```
Next
End Sub
</SCRIPT>
<TITLE>循环语句 For Each...Next</TITLE>
</HEAD>
<BODY>
<P align=center>
<INPUT id=button1 name=button1 type=button value=我的爱好>
</P>
<FORM name="frm_interest">
<P align=center>
<INPUT id =text1 name =text1><BR>
<INPUT id =text2 name =text2><BR>
<INPUT id =text3 name =text3><BR>
<INPUT id =text4 name =text4><BR>
<INPUT id =text5 name =text5><BR>
<INPUT id =text6 name =text6><BR>
<INPUT id =text7 name =text7><BR>
</P>
</FORM>
<P> </P>
</BODY>
</HTML>
```

运行结果如图 3.3 **所示。**

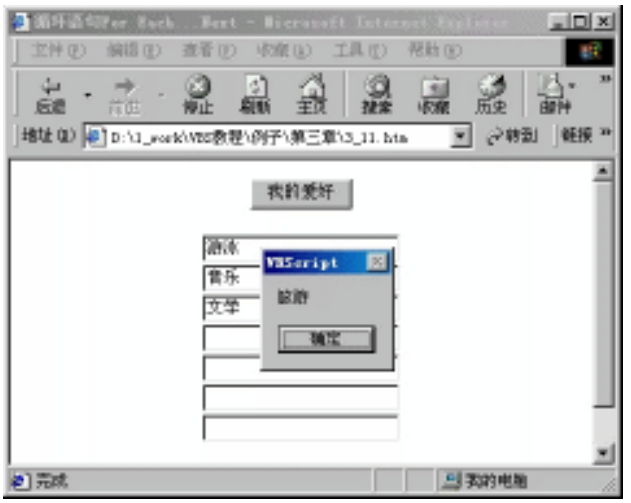

图 3.3　For Each…Next **示例运行效果**

3.3　其 他 语 句

上面我们较详细地介绍了 VBScript 中的常用语句，除了这些以外，还有很多其他不易归类的语句，我们将按字母顺序逐一讲解。

3.3.1　Call 语句

将控制权传递给 Sub 或 Function 过程，语法如下：

```
[Call] name [argumentlist]
```

参数说明：

- Call：可选关键字。如果指定此关键字，则必须用括号把 argumentlist 括起来。例如：

  ```
  Call MyProc(0)
  ```

- Name：必选项。要调用的过程名。
- Argumentlist：可选项。传递给过程的变量、数组或表达式列表，用逗号分隔每一项。

在调用过程时，不必使用 Call 关键字。然而，如果使用 Call 关键字调用要求参数的过程，则必须用括号将 argumentlist 括起来。如果省略 Call 关键字，那么必须也同时省略 argumentlist 参数两边的括号。使用 Call 语法调用内部函数或使用用户自定义函数，函数返回值都会被放弃。例如：

```
Call MyFunction("Hello World")
Function MyFunction(text)
  MsgBox text
End Function
```

3.3.2　Const 语句

定义常量，语法如下：

```
[Public | Private] Const constname = expression
```

参数说明：

- Public：可选项。该关键字用于在 Script 级中声明可用于所有脚本中所有过程的常数。不允许在过程中使用。
- Private：可选项。该关键字用于在脚本级中声明只可用在声明所在的脚本中的常数。不允许在过程中使用。
- Constname：必选项。常数的名称，根据标准的变量命名约定。
- Expression：必选项。文字或其他常数，或包括除 Is 外的所有算术运算符和逻辑运

算符的任意组合。

在默认情况下常数是公用的。过程中的常数总是专有的，其可见性无法改变。在 Script 中，可用 Private 关键字来改变脚本级常数可见性的默认值。

要在同一行中声明若干个常数，可用逗号将每个常数赋值分开。用这种方法声明常数时，如果使用了 Public 或 Private 关键字，则该关键字对该行中所有常数都有效。

常数声明中不能使用变量、用户自定义的函数或 VBScript 内部函数（如 Chr）。按定义，它们不能是常数。另外也不能从含有运算符的表达式中创建常数，即只允许使用简单常数。在 Sub 或 Function 过程中声明的常数是该过程的局部常数。在过程外声明的常数是声明所在的脚本中的全局常数。可以在任何使用表达式的地方使用常数。

下面举例说明如何使用 Const 语句：

```
Const MyVar = 459              ’常数默认为公有
Private Const MyString = "HELP"        ’定义私有常数
Const MyStr = "Hello", MyNumber    = 3.4567   在一行上定义多个常数
```

注意：常数能使脚本自己支持并且容易修改。不像变量，脚本在运行时，常数不能被无意中修改。

3.3.3 Dim 语句

声明变量并分配存储空间。语法如下：

```
Dim varname[([subscripts])][, varname[([subscripts])]] . . .
```

参数说明：

- Varname：变量的名称，符合标准变量命名约定。
- Subscripts：数组变量的维数，最多可以声明 60 维的数组。subscripts 参数使用以下语法：

  ```
  upperbound [,upperbound] . . .
  ```

  数组下界总是 0。

用 Dim 声明的 Script 级变量可用于脚本中的所有过程，过程级变量只能用于过程中。

也可用带空圆括号的 Dim 语句来声明动态数组。声明动态数组后，可在过程内使用 ReDim 语句来定义该数组的维数和元素。如果试图重新定义在 Dim 语句中已经显式指定维数的数组，则会发生错误。

注意：在过程中使用 Dim 语句时，通常将 Dim 语句放在过程的开始处。

下面举例说明如何使用 Dim 语句：

```
Dim Names(9)              ’声明一个具有 10 个元素的数组
Dim Names()               ’声明动态数组
Dim MyVar, MyNum          ’声明两个变量
```

3.3.4　Erase 语句

重新初始化固定大小数组的元素，并释放动态数组的存储空间。

```
Erase array
```

其中，array**：参数是指要清除的数组变量的名称。**

判断数组是固定长度数组（常规）还是动态数组是很重要的，这是因为 Erase **要根据数组的类型进行不同的操作。**Erase **无需为固定大小的数组还原内存。**Erase **按照表** 3.2 **设置固定数组的元素。**

表 3.2　Erase **对固定元素的影响**

数组的类型	Erase **对固定数组元素的影响**
固定数值数组	**将每个元素设置为** 0
固定字符串数组	**将每个元素设置为零长度字符串** ("")
对象数组	**将每个元素设置为特殊值** Nothing

Erase **释放动态数组所使用的内存。在程序再次引用该动态数组之前，必须使用** ReDim **语句来重新定义该数组变量的维数。**

下面举例说明如何使用 Erase **语句。**

```
Dim NumArray(9)
Dim DynamicArray()
ReDim DynamicArray(9)    '分配存储空间
Erase NumArray           '每一元素都被重新初始化
Erase DynamicArray       '释放数组占用的内存
```

3.3.5　Exit 语句

退出 Do...Loop、For...Next、Function **或** Sub **代码块。**

Exit **语句的语法有以下几种形式：**

```
Exit Do
Exit For
Exit Function
Exit Property
Exit Sub
```

语句说明：

- Exit Do**：提供一种退出** Do...Loop **语句的方法，只能在** Do...Loop **语句中使用。**Exit Do **将控制权转移到** Loop **语句之后的语句。在嵌套的** Do...Loop **语句中使用时，**Exit Do **将控制权转移到循环所在位置的上一层嵌套循环。**
- Exit For**：提供一种退出** For **循环的方法，只能在** For...Next **或** For Each...Next **循环中使用。**Exit For **将控制权转移到** Next **之后的语句。在嵌套的** For **循环中使**

用时，Exit For 将控制权转移到循环所在位置的上一层嵌套循环。

- Exit Function：立即从出现的位置退出 Function 过程，继续执行调用 Function 的语句后面的语句。
- Exit Property：立即从所在的 Property 过程中退出，继续执行下面调用 Property 过程的语句。
- Exit Sub：立即从出现的位置退出 Sub 过程，继续执行调用 Sub 的语句后面的语句。

下面举例说明如何使用 Exit 语句。

```
Sub RandomLoop
  Dim I, MyNum
  Do                                ’设置死循环
    For I = 1 To 1000               ’循环 1000 次
      MyNum = Int(Rnd * 100)        ’产生随机数
      Select Case MyNum             ’求随机数的值
        Case 17: MsgBox "Case 17"
          Exit For                  ’如果是 17，退出 For...Next
        Case 29: MsgBox "Case 29"
          Exit Do                   ’如果是 29，退出 Do...Loop
        Case 54: MsgBox "Case 54"
          Exit Sub                  ’如果是 54，退出 Sub 过程
        End Select
    Next
  Loop
End Sub
```

3.3.6 Function 语句

声明 Function 过程的名称、参数以及构成其主体的代码，语法如下：

```
[Public [Default]| Private] Function name [(arglist)]
[statements]
[name = expression]
[Exit Function]
[statements]
[name = expression]
End Function
```

参数说明：

- Public：表示 Function 过程可被所有脚本中的所有其他过程访问。
- Default：只与 Class 块中的 Public 关键字一起使用来表示 Function 过程是类的默认方法。如果在一个类中指定了不止一个 Default 过程，就有错误发生。
- Private：表示 Function 过程只可被声明它的脚本中的其他过程访问，或者如果函

数是一个数据类，那么 Function 过程只能被该类中的其他过程访问。

- Name：Function 的名称，遵循标准的变量命名约定。
- Arglist：代表调用时要传递给 Function 过程的参数的变量列表。用逗号隔开多个变量。
- Statements：在 Function 过程的主体中执行的任意语句组。
- Expression：Function 的返回值。

其中，arglist 参数包含下列语法和部分：

```
[ByVal | ByRef] varname[( )]
```

参数说明：

- ByVal：表示该参数是按值方式传递的。
- ByRef：表示该参数按引用方式传递。
- varname：代表参数变量的名称，遵循标准的变量命名约定。

如没有显式指定使用 Public 或 Private，则 Function 过程默认为公用，即它们对于脚本中的所有其他过程是可见的。Function 中局部变量的值在对过程的调用中不被保留。

不能在任何其他过程，例如 Sub 或 Property Get 中定义 Function 过程。

使用 Exit Function 语句可以从 Function 过程中立即退出。程序继续执行调用 Function 过程的语句之后的语句。可在 Function 过程的任何位置出现任意个 Exit Function 语句。

与 Sub 过程类似，Function 过程是可以获取参数、执行一系列语句并改变其参数值的独立过程。与 Sub 过程的不同之处是：当要使用由函数返回的值时，可以在表达式的右边使用 Function 过程，这与内部函数的使用方式一样。例如，Sqr、Cos 或 Chr。

在表达式中，可以通过使用函数名，并在其后用圆括号给出相应的参数列表来调用 Function 过程。有关调用 Function 过程的详细信息，请参阅 Call 语句。

注意：Function 过程是可以递归的，即该过程可以调用自身以完成某个给定的任务。但是，递归可能会导致堆栈溢出。

要从函数返回一个值，只需将值赋给函数名。在过程的任意位置都可以出现任意个这样的赋值。如果没有给 name 赋值，则过程将返回一个默认值，数值函数返回 0；字符串函数返回零长度字符串 ("")。如果在 Function 中没有对象引用被指定给 name（使用 Set），则返回对象引用的函数将返回 Nothing。

下面的例子说明如何给一个名为 BinarySearch 的函数赋返回值。在此例子中，将 False 赋给了该函数名，表示没有找到某个值。

```
Function BinarySearch(. . .)
    . . .
    ' 未找到该值。返回 False 值
    If lower > upper Then
        BinarySearch = False
        Exit Function
```

```
    End If
    . . .
End Function
```

在 Function 过程中使用的变量分为两类：一类是在过程内显式声明的，另一类则不是。在过程内显式声明的变量（使用 Dim 或等效方法）总是过程的局部变量。被使用但没有在过程中显式声明的变量也是局部变量，除非在该过程外更高级别的位置显式声明它们。

注意：VBScript 可能会重新排列数学表达式以提高内部效率。当 Function 过程修改数学表达式中变量的值时，应避免在同一表达式中使用该函数。

3.3.7 On Error 语句

启用或禁用错误处理程序。语法如下：

```
On Error Resume Next
On Error GoTo 0
```

如果在代码中未使用 On Error Resume Next 语句，所发生的运行时错误将显示错误信息，同时，代码的执行也随之终止。但是具体操作由运行代码的主机决定。主机有时可有选择地处理各类错误。在有些情况下，它可以在出错的地方激活脚本调试器，而在另一些情况下，由于主机无法通知用户，因此对所发生的错误没有明确说明，至于如何处理错误则完全取决于主机的功能。

在任意一个特殊过程中，只要在调用堆栈的地方启用错误处理程序，所发生的错误一般不会是致命性的。如果在一个过程中没有启用局部错误处理程序，当发生错误时，控制可通过堆栈调用转移，直到找到一个具有错误处理程序的过程，并在出错的地方处理错误。如果在调用堆栈的过程中没有找到错误处理程序，则在出错的地方显示错误信息，同时终止代码执行，或者通过主机来正确处理错误。

On Error Resume Next 会使程序按照产生错误的语句之后的语句继续执行，或是按照最近一次所调用的过程（该过程含有 On Error Resume Next 语句）中的语句继续运行。这个语句可以不顾运行时的错误，继续执行程序，之后可以在过程内部建立错误处理例程。在调用另一个过程时，On Error Resume Next 语句将变为非活动的。所以，如果希望在例程中进行内部错误处理，则应在每一个调用的例程中执行 On Error Resume Next 语句。

当调用另一过程时，禁止使用 On Error Resume Next 语句，因此如果想在例程中嵌入错误处理程序，则需要在每次调用例程时都应执行 On Error Resume Next 语句。当退出一个过程时，错误处理程序可恢复到它在进入退出过程之前的状态。

如果已启用 On Error Resume Next 错误处理程序，则可使用 On Error GoTo 0 禁用错误处理程序。

下面举例说明如何使用 On Error Resume Next 语句。

```
On Error Resume Next
Err.Raise 6    产生溢出错误
MsgBox ("Error # " & CStr(Err.Number) & " " & Err.Description)
```

```
Err.Clear      '清除错误
```

3.3.8　Option Explicit 语句

强制要求显式声明脚本中的所有变量。

```
Option Explicit
```

如果使用 Option Explicit**，该语句必须出现在脚本的任何其他语句之前。**

使用 Option Explicit **语句时，必须使用** Dim、Private、Public **或** ReDim **语句显式声明所有变量。如果试图使用未经声明的变量名，则会出现错误。**

提示：可用 Option Explicit **避免拼错已存在的变量名称。对于作用范围不清楚的变量，使用此语句可避免发生混淆。**

下面举例说明如何使用 Option Explicit **语句。**

```
Option Explicit        '强制显示声明变量
Dim MyVar              '声明变量
MyInt = 10             '未声明变量产生错误
MyVar = 10             '声明变量不产生错误
```

3.3.9　Private 语句

定义私有变量并分配存储空间。在 Class **块中定义私有变量。语法如下：**

```
Private varname[([subscripts])][， varname[([subscripts])]] . . .
```

参数说明：

- Varname**：变量的名称；遵循标准变量命名约定。**
- Subscripts**：数组变量的维数，最多可以声明** 60 **维的数组。**subscripts **参数使用下列语法：**

  ```
  upper [, upper] . . .
  ```

  数组的下界总是 0**。**

Private **语句变量只能在声明该变量的脚本中使用。**

在使用引用对象的变量之前，必须用 Set **语句将某个现有对象赋予此变量。在赋值之前，所声明的对象变量被初始化** Empty**。**

也可用带空圆括号的 Private **语句声明动态数组。声明动态数组后，可在过程内使用** ReDim **语句定义该数组的维数和元素。如果在** Private、Public **或** Dim **语句中已显式指定数组大小，却试图重新声明数组维数，就会发生错误。**

注意：在过程中使用 Private **语句时，通常将** Private **语句放在过程的开始处。**

下面例子说明了如何使用 Private **语句。**

```
Private MyNumber              '定义私有 Variant 变量
```

```
Private MyArray(9)                    ’定义私有数组变量
Private MyNumber, MyVar, YourNumber   ’定义多个私有 Variant 变量
```

3.3.10 Public 语句

定义公有变量并分配存储空间。在 Class 块中定义私有变量。语法如下：

```
Public varname[([subscripts])][, varname[([subscripts])]] . . .
```

参数说明：

- varname：变量的名称，遵循标准变量命名约定。
- subscripts：数组变量的维数，最多可以声明 60 维的数组。 subscripts 参数使用下列语法：

  ```
  upper [,upper] . . .
  ```

  数组的下界总是 0。

Public 语句变量可用于全部脚本中的所有过程。

在使用引用对象的变量之前，必须用 Set 语句将某个已有对象赋予该变量。在赋值之前，所声明的对象变量被初始化为 Empty。

也可用带空圆括号的 Public 语句来声明动态数组。声明动态数组后，可在过程内使用 ReDim 语句来定义该数组的维数和元素。如果试图重新声明数组变量的维数，且此数组变量的大小已在 Private、Public 或 Dim 语句中显式地指定，则会发生错误。

下面举例说明如何使用 Public 语句。

```
Public MyNumber                       ’定义公有 Variant 变量
Public MyArray(9)                     ’定义公有数目变量
Public MyNumber, MyVar, YourNumber    ’定义多个公有 Variant 变量
```

3.3.11 Redim 语句

在过程级中声明动态数组变量并分配或重新分配存储空间。语法如下：

```
ReDim [Preserve] varname(subscripts) [, varname(subscripts)] . . .
```

参数说明：

- Preserve：当更改现有数组最后一维的大小时保留数据。
- varname：变量名，遵循标准变量命名约定。
- subscripts：数组变量的维数，最多可以声明 60 维数组。subscripts 参数语法格式如下：

  ```
  upper [,upper] . . .
  ```

  数组的下界总是 0。

ReDim 语句通常用于指定或修改动态数组的大小，这些数组已用带有空括号的

Private、Public 或 Dim 语句（没有维数下标）正式声明过。可以重复使用 ReDim 语句更改数组维数和元素数目。

如果使用了 Preserve 关键字，就只能调整数组最后一维的大小，并且不能改变数组的维数。例如，如果数组只有一维，就可以修改该数组的大小，因为该维是最后的也是仅有的一维。但是，如果数组有两个或更多维，就只能改变末维的大小并保留数组内容。

下面例子说明如何不擦掉该数组中存在的数据，而增加动态数组的终止维数。

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```

注意：如果减小数组的大小，则将丢失被排除的元素中的数据。

变量初始化时，数值变量初始化为 0，字符串变量初始化为零长度字符串("")。在使用引用对象的变量前，必须使用 Set 语句将某个现有对象赋予该变量。在进行对象赋值以前，已声明的对象变量有特定值 Nothing。

3.3.12　Rem 语句

包含程序中的解释性注释。语法如下：

```
Rem comment
```

或

```
’comment
```

其中 comment 参数是指需要包含的注释文本。在 Rem 关键字和 comment 之间应有一个空格。

正如“语法”部分所示，可以用单引号 (') 代替 Rem 关键字。如果 Rem 关键字和语句在同一行，需要用分号来分隔它们。但如果使用单引号，则不需要在单引号和语句之间使用分号。

下面举例说明如何使用 Rem 语句。

```
Dim MyStr1, MyStr2
MyStr1 = "Hello" : Rem          ’语句和注释用冒号隔开
MyStr2 = "Goodbye"              ’这同样是注释不需要冒号
```

提示：Rem 在没有代码的行上加注释不必用冒号。

3.3.13　Sub 语句

声明 Sub 过程的名称、参数以及构成其主体的代码。语法如下：

```
[Public [Default]| Private] Sub name [(arglist)]
[statements]
[Exit Sub]
```

```
[statements]
End Sub
```

参数说明：

- Public：表示 Sub 过程可被所有脚本中的所有其他过程访问。
- Default：只与类块中的 Public 关键字连用，用来表示 Sub 过程是类的默认方法。如果在类中指定了不止一个 Default 过程，就会出错。
- Private：表示 Sub 过程只可被声明该过程的脚本中的其他过程访问。
- name：Sub 的名称，遵循标准变量命名约定。
- arglist：代表在调用时要传递给 Sub 过程的参数的变量列表。用逗号隔开多个变量。
- statements：在 Sub 过程主体内所执行的任何语句组。

其中，arglist **参数包含下列语法和部分：**

```
[ByVal | ByRef] varname[( )]
```

参数说明：

- ByVal：表示该参数是按值传递的。
- ByRef：表示该参数按引用传递。
- Varname：代表参数的变量名称，遵循标准变量命名约定。

如没有显式地指定使用 Public 或 Private，则 Sub 过程默认为公用，即它们对于脚本中的所有其他过程都是可见的。Sub 过程中局部变量的值在调用过程中不被保留。

不能在任何其他过程，例如 Function 或 Property Get 中定义 Sub 过程。

使用 Exit Sub 语句可以立即从 Sub 过程中退出。程序继续执行调用 Sub 过程的语句之后的语句。可以在 Sub 过程中任意位置出现任意个 Exit Sub 语句。

与 Function 过程相似之处是，Sub 过程是一个可以获取参数，执行一系列语句以及可改变其参数的值的独立过程。而与 Function 过程不同之处是，Function 过程可以返回值，而 Sub 过程不能用于表达式中。

可以使用过程名并跟随相应的参数列表来调用 Sub 过程。关于如何调用 Sub 过程的详细说明信息，请参阅 Call 语句。

注意：Sub 过程可以是递归的，即该过程可以调用自己来完成某个给定的任务，但是递归可能会导致堆栈溢出。

在 Sub 程中使用的变量分为两类：一类是在过程内显式声明的，另一类则不是。在过程内显式声明的变量（使用 Dim 或等效方法）总是局部变量。对于那些没有在过程中显式声明的变量也是局部的，除非在该过程外更高级别的位置显式地声明它们。

注意：过程可以使用没有在过程内显式声明的变量，但只要有任何 Script 级定义的名称与之同名，就会产生名称冲突。如果过程中引用的未声明的变量与其他的

过程、常数或变量的名称相同，则会认为过程引用的是脚本级的名称。要避免这类冲突，请使用 Option Explicit 语句强制显式声明变量。

3.4 小　结

在这一章中，我们着重讲述了 VBScript 的语句，主要内容包括：

- 条件语句两个：If…Then…Else 和 Select Case。
- 循环语句 4 个：Do…Loop、While…Wend、For…Next 和 For Each…Next。
- 其他语句 14 个：Call 语句、Const 语句、Dim 语句、Erase 语句、Exit 语句、Function 语句、On Error 语句、Option Explicit 语句、Pubic 语句、Redim 语句、Rem 语句、Redim 语句、Rem 语句和 Sub 语句。

其中重点和难点是条件语句和循环语句，请大家务必掌握。从下一章开始我们将讲述 VBScript 中的过程。

练习题

1．VBScript 中的条件语句有哪几种？它们之间的区别在哪里？

2．VBScript 中的循环语句有哪几种？它们之间的区别在哪里？

3．给出一百分制，要求输出成绩等级“A”、“B”、“C”、“D”、“E”。其中，90 分以上为“A”，80~89 为“B”，70~79 为“C”，60~69 为“D”，60 分以下为“E”。

要求：分别用 If…Then…Else 语句和 Select Case 语句实现，并比较它们在不同情况下的优缺点。

4．编写一程序，求方程 $ax^2+bx+c=0$ 的解。

提示：方程的解有下列几种可能：

（1）a=0，不是二次方程。

（2）$b^2-4ac=0$，有两个相等的实根。

（3）$b^2-4ac>0$，有两个不等的实根。

（4）$b^2-4ac<0$，有两个共轭复根。

5．编写一个程序，输出以下图案。

```
      *
    * * *
  * * * * *
* * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

第 4 章　VBScript 中的过程

到目前为止，我们已经在 HTML 文档中创建了很多的脚本，其中在大部分脚本中我们使用了过程，但并未详细地讨论过。本章我们将围绕这个话题展开讨论。

在本章中我们将学习以下内容：

- 过程的基本知识以及在 VBScript 脚本中如何使用过程。
- Sub 过程和 Function 过程的区别。
- 如何在自己的脚本中使用 VBScript 函数。

一个较大的程序一般应分为若干个程序模块，每个模块用来实现一个特定的功能。在程序设计中，常将一些常用的功能模块编写成过程，供公共调用。善于利用过程，可以减少重复编写程序的工作量。

过程是组成程序的逻辑单位，过程中的代码在过程被调用时执行。一个过程可以被其他的过程调用，通常用 Call 语句调用（也可以省略 Call，但用法稍有不同），也可以通过像按一下按钮等触发这样的一些事件来激发，使过程被执行。

用过程可以将紧密相关的语句封装在一起，实现某个特定功能。这样可以提高代码的可重用性和可读性，并且使程序更容易阅读。在 VBScript 中，过程被分为两类：Sub 过程和 Function 过程，Sub 与 Function 过程的最大区别在于它只执行一个操作而并不返回值，而 Function 过程有返回值。两者均可以有参数。有的书中将不返回值的 Sub 过程称为子过程，而将返回值的 Function 过程成为函数。下面我们将分别详细讨论这两种过程。

4.1　Sub 过程

Sub 过程是包含在 Sub 和 End Sub 语句之间的代码块，执行操作但不返回值。Sub 过程可以使用参数（指调用过程传递的常数、变量或表达式）。如果 Sub 过程无任何参数，则 Sub 语句必须包含空括号()。

下面的 Sub 过程使用两个固有的（或内置的）VBScript 函数，即 MsgBox 和 InputBox 来提示用户输入用户姓名。然后根据输入显示。

【例 4.1】　使用不带参数的 Sub 过程

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
<!--
Sub Sayhello()
    UserName = InputBox("请输入您的姓名。", 1)
```

```
        MsgBox "亲爱的" & UserName & "用户，您好！"
    End Sub
-->
</SCRIPT>
<TITLE>使用不带参数的 Sub 过程</TITLE>
</HEAD>
<BODY>
<SCRIPT Language="VBScript">
<!--
Call Sayhello
-->
</SCRIPT>
<P> </P>

</BODY>
</HTML>
```

本例中，首先定义了一个过程 Sayhello，它的作用是提示用户输入自己的姓名，然后根据不同的输入进行显示。我们在文件的主体部分<BODY></BODY>中调用该过程 Call Sayhello。那么，在页面载入浏览器的时候，就开始执行 Sayhello 中的代码。

运行结果如图 4.1 和图 4.2 所示。

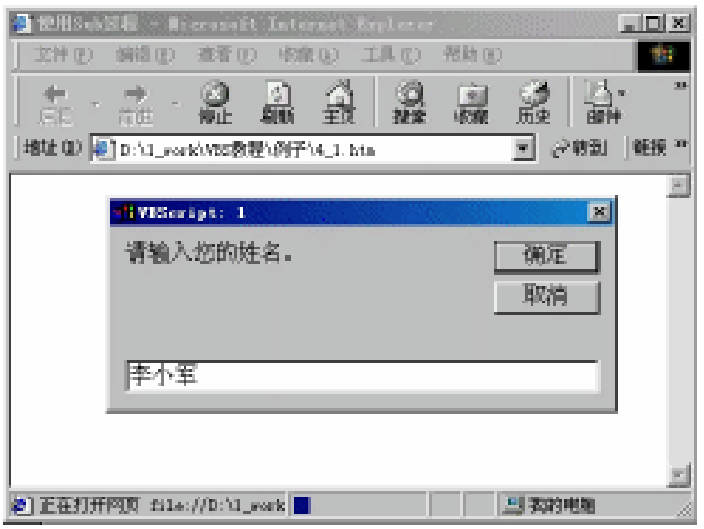

图 4.1　使用没有参数的 Sub 过程——提示用户输入姓名

在用户输入完自己的姓名之后，将根据用户的输入信息输出不同的信息。

这个过程是一个不带参数的过程，如果是带参数的过程执行情况又如何呢？让我们看下面的例子。

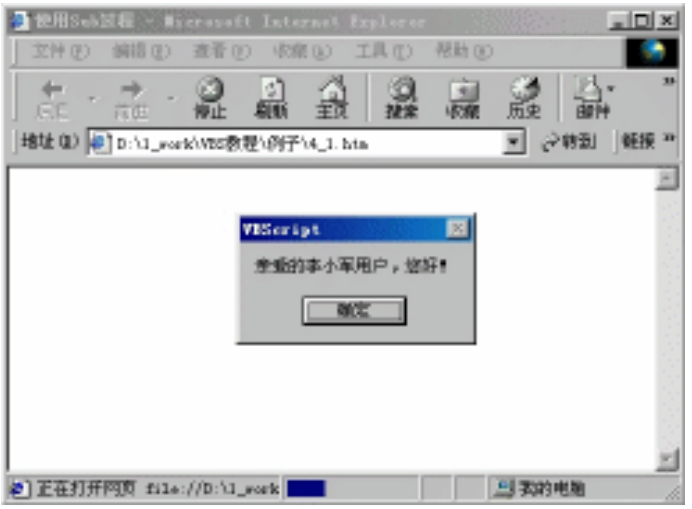

图 4.2　根据用户的输入显示不同的消息

【例 4.2】　使用带参数的 Sub 过程

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
<!--
Sub AddResult(startnum,endnum)
Dim Result
startnum = Cint(startnum)
endnum = Cint(endnum)
Result = 0
    For i = startnum to endnum
        Result = Result + i
    Next
    document.frm_add.result.VALUE = Result
End Sub
-->
</SCRIPT>
<TITLE>使用带参数的 Sub 过程</TITLE>
</HEAD>
<BODY>

<P>
<FORM name="frm_add">
请计算出从<INPUT name =startnum style="height: 22px; width: 52px">  到 <INPUT name =endnum style=" height: 22px; width: 53px">之间所有整数的和
</P>
```

```
<P><INPUT id=button1 name=button1 type=button value= 开 始 计 算 onclick="AddResult
document.frm_add.startnum.value ,document.frm_add.endnum.value" > 
</P>
<P>
结果为：<INPUT id=result name=result value=0.0 style=" height: 22px; width: 113px">
</P>
</FORM>
</BODY>
</HTML>
```

本例要实现的功能是：由用户输入一个起始数字和一个终止数字，单击“开始计算”按钮，在另一个文本框中显示计算结果。在过程 AddResult **中，将用户输入的起始数字** startnum **和终止数字** endsum **作为该过程的参数，定义了一个变量** Result **来存放计算结果。运行结果如图** 4.3 **所示。**

图 4.3　**使用带参数的** Sub **过程——计算**

其实，像这样的功能使用 Function **过程会更合适，我们可以将【例** 4.2**】改写为下面的【例** 4.3**】。**

【例 4.3**】　使用** Function **过程**

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
<!--
Function AddResult(startnum,endnum)
startnum = Cint(startnum)
endnum     = Cint(endnum)
Result = 0
```

```
        for i = startnum to endnum
            Result = Result + i
        next
        AddResult = Result
    End Function

    Sub button1_OnClick()
        document.frm_add.result.VALUE =
    AddResult(document.frm_add.startnum.VALUE,document.frm_add.endnum.VALUE)
    End Sub
    -->
    </SCRIPT>
    <TITLE>使用 Function 过程</TITLE>
    </HEAD>
    <BODY>

    <P>
    <FORM name="frm_add">
    请计算出从<INPUT name =startnum style="height: 22px; width: 52px">  到 <INPUT name
=endnum style =" height: 22px; width: 53px">之间所有整数的和
    </P>
    <P><INPUT id=button1 name=button1 type=button value=开始计算 > 
    </P>
    <P>
    结果为：<INPUT id=result name=result value=0.0 style="height: 22px; width: 113px">
    </P>
    </FORM>
    </BODY>
    </HTML>
```

本例中，计算是由过程 AddResult **来完成的，它将计算的结果作为返回值。在这个例子中，当用户单击“开始计算”按钮时，触发** button1 **的** Click **事件，在这个事件中，直接调用了** Function **过程** AddResult**，将返回值赋给文本框** result**。**

4.2 Function 过程

Function **过程是包含在** Function **和** End Function **语句之间的代码块。**Function **过程与** Sub **过程类似，但是** Function **过程可以返回值，它的返回值保存在一个和函数同名的变量中。**Function **过程可以使用参数（由调用过程传递的常数、变量或表达式）。如果** Function **过程无任何参数，则** Function **语句必须包含空括号** ()**。**Function **过程通过函数名返回一个值，这个值是在过程的语句中赋给函数名的。**Function **返回值的数据类型总是** Variant**。**

在上一节中，我们通过将 Function 过程与 Sub 过程对比，对 Function 过程作了一些介绍，相信读者已经对 Function 过程有了一个大概的认识。这里我们就不再举过多的例子描述了。

4.3　过程中传入或获得数据

给过程传递数据的途径是使用参数。参数被作为要传递给过程的数据的占位符。参数名可以是任何有效的变量名。使用 Sub 语句或 Function 语句创建过程时，过程名之后必须紧跟括号。括号中包含所有参数，参数间用逗号分隔。例如，在【例 4.3】中，startnum 和 endnum 是传递给 AddResult 过程的值的占位符：

```
Function AddResult(startnum,endnum)
    ……
End Function
```

要从过程中获取数据，必须使用 Function 过程。再强调一次，Function 过程可以返回值；Sub 过程不返回值。

4.4　在代码中使用 Sub 和 Function 过程

调用 Function 过程时，函数名必须用在变量赋值语句的右端或表达式中。例如：

```
Temp = AddResult(startnum,endnum)
```

或

```
MsgBox "计算结果为 " & AddResult(startnum,endnum) & " 摄氏度。"
```

调用 Sub 过程时，只需输入过程名及所有参数值，参数值之间使用逗号分隔。不需使用 Call 语句，但如果使用了此语句，则必须将所有参数包含在括号之中。

下面的例子显示了调用 AddResul 过程的两种方式。一种不使用 Call 语句；另一种使用，两种方式效果相同。

```
AddResult startnum,endnum
```

或

```
Call AddResult(startnum,endnum)
```

可见，当不使用 Call 语句进行调用时，括号被省略。

注意：如果对 Function 过程使用 Call 语句，将丢失其返回值。

4.5 小 结

本章我们讲述了 VBScript 的过程——Sub 过程和 Function 过程。Sub 过程执行一个操作，没有返回值；Function 过程执行一个操作后，可以有返回值。两者都可以有输入参数。

除了在脚本中使用自己创建的函数过程外，还可以使用 VBScript 中已经创建好的固有函数，这些函数对于编写庞大和复杂的脚本来说是大有裨益的。从下一章开始，我们将介绍 VBScript 的固有函数。

练习题

1．描述 Sub 过程和 Function 过程的区别。

2．如何调用过程？

3．在调用过程时，何时用括号，何时可以不用括号？

4．编写一个 Sub 过程，该过程限制了文件主体部分的每个输入文本框都不能为空，在单击按钮时，调用该过程，如果某个输入文本框为空，则给出提示。

5．编写一个 Function 过程，该过程将阿拉伯数字 1~7 转换为汉字的星期一到星期日，在文件的主体部分调用该过程，显示今天是星期几。

第5章 内部函数

在前几章的学习中，我们已经接触到函数的概念，并且使用过几个简单的函数，如Msgbox函数、Date函数等。从本章开始，我们将系统地介绍函数。

VBScript有一批创建好了的函数供用户使用，这些函数我们称为内部函数，以便与用户自定义的函数即外部函数相区分。这些函数包括字符串操作函数、数据转换函数、数学函数、时间和日期函数和布尔函数等。虽然可以自己编写脚本来实现同样的功能，但是直接调用VBScript的函数可以在编写代码的过程中给我们提供很大的方便。学习在设计网页中应用VBScript，一定要学会使用内部函数。

本章我们将重点介绍这些内部函数，其中重要的、常用的函数将举例说明如何使用。

5.1 字符串函数

VBScript中的字符串函数是用来处理字符串数据的，它可以对字符串进行操作并对其进行句法分析。句法分析文本数据的含义是将文本分为逻辑块，如可以完成取字符串长度、截取字符串、字符串大小写转换等功能。下面我们来看一下与字符串相关的函数，并通过实例来展现如何使用这些函数。

5.1.1 Asc 函数

返回字符串的第一个字母对应的ASCII字符代码。语法如下：

```
Asc(string)
```

参数说明：

string：该参数是任意有效的字符串表达式。如果 string 参数未包含字符，则将发生运行时错误。例如，Asc("Apple") 将返回“A”的ASCII字符码“65”。如果在使用这个函数时，所传入的参数是空，会产生一个运行时错误。

5.1.2 Chr 函数

返回与指定的 ASCII 字符代码相对应的字符。语法如下：

```
Chr(charcode)
```

参数说明：

charcode：该参数是可以标识字符的数字。从 0 到 31 的数字表示标准的不可打印的ASCII代码。例如，Chr(10)返回换行符。

下面例子利用 Chr 函数返回与指定的字符代码相对应的字符。

```
Dim MyChar
MyChar = Chr(66)      ’返回 B
MyChar = Chr(98)      ’返回 b
MyChar = Chr(63)      ’返回 ?
MyChar = Chr(38)      ’返回 &
```

5.1.3 InStr 函数

返回某字符串在另一字符串中第一次出现的位置，是用来进行文本搜索的工具。语法如下：

```
InStr([start, ]string1, string2[, compare])
```

参数说明：

- start：**可选项。数值表达式，用于设置每次搜索的开始位置。如果省略，将从第一个字符的位置开始搜索；如果** start **包含** Null**，则会出现错误；如果已指定** compare **参数，则必须要有** start **参数。**
- string1：**必选项。接受搜索的字符串表达式。**
- string2：**必选项。要搜索的字符串表达式。**
- compare：**可选项。指示在计算子字符串时使用的比较类型的数值。有关数值，请参阅表** 5.1**。如果省略，将执行二进制比较。**

表 5.1 InStr **函数的** compare **参数的值**

常数	值	描述
vbBinaryCompare	0	执行二进制比较
vbTextCompare	1	执行文本比较

该函数的返回值，参见表 5.2 **所示。**

表 5.2 InStr **函数的返回值**

条件	返回值
string1 **为零长度**	0
string1 **为** Null	Null
string2 **为零长度**	start
string2 **为** Null	Null
string2 **没有找到**	0
在 string1 **中找到** string2	**找到匹配字符串的位置**
start > Len(string2)	0

下面的例子利用 InStr **函数搜索字符串。**

```
Dim SearchString, SearchChar, Position
SearchString ="This is my homepage"          ’接受搜索的字符串
```

```
SearchChar = "h"                                    ’要搜索的字符串
Position = Instr(4, SearchString, SearchChar, 1)    ’从 SearchString 的第 4 个字符开始搜索 SearchChar，进行文本比较，该语句返回值为 12
Position = Instr(1, SearchString, SearchChar, 0)    ’从 SearchString 的第 1 个字符开始搜索 SearchChar，进行二进制比较，该语句返回值为 2
Position = Instr(SearchString, SearchChar)          ’省略了可选参数，表示从 SearchString 的第一个字符开始搜索 SearchChar，进行二进制比较，返回值为 2
Position = Instr(1, SearchString, "S")              ’从 SearchString 的第 1 个字符开始进行二进制的比较，搜索字符“S”，返回值为 0（因为 S 没有找到）
```

5.1.4　InstrRev 函数

InStrRev 函数的功能和 InStr 的功能类似，不同之处在于 InStrRev 函数是从字符串尾开始进行搜索，而 InStr 函数是从字符串首开始进行搜索。InStrRev 函数的语法如下：

```
InStrRev(string1, string2[, start[, compare]])
```

参数说明：

- string1：必选项，接受搜索的字符串表达式。
- string2：必选项，被搜索的字符串表达式。
- start：可选项，数值表达式，用于设置每次搜索的开始位置。如果省略，则默认值为-1，表示从最后一个字符的位置开始搜索。如果 start 包含 Null，则出现错误。
- compare：可选项。在计算子字符串时，指示要使用的比较类型的数值。如果省略，将执行二进制比较。有关数值，请参阅表 5.3。

表 5.3　InStrRev 函数的 compare 参数值

常数	Value	描述
vbBinaryCompare	0	执行二进制比较
vbDatabaseCompare	2	执行基于包含在数据库（在此数据库中执行比较）中的信息的比较

该函数的返回值，参见表 5.4 所示。

表 5.4　InStrRev 函数的返回值

条件	返回值
string1 为零长度	0
string1 为 Null	Null
string2 为零长度	start
string2 为 Null	Null
string2 没有找到	0
在 string1 中找到 string2	找到匹配字符串的位置
start > Len(string2)	0

下面是利用 InStrRev 函数搜索字符串的例子：

```
Dim SearchString, SearchChar, Position
SearchString ="This is my homepage"          ’接受搜索的字符串
SearchChar = "h"                             ’要搜索的字符串
Positon = InstrRev(SearchString, SearchChar, 12, 0)    ’从 SearchString 结尾的第 12 个字符开始搜索 SearchChar，比较类型是二进制比较，结果返回 12
Positon = InstrRev(SearchString, SearchChar, -1, 1)    ’从 SearchString 的结尾位置开始搜索 SearchChar，比较类型为文本比较，结果返回 12
Positon = InstrRev(SearchString, SearchChar, 6)        ’从 SearchString 结尾的第 6 个字符开始搜索 SearchChar，比较类型省略，表示默认的二进制比较，返回结果为 0
```

注意：InStrRev 函数的语法与 InStr 函数的语法并不一样。

5.1.5 LCase 函数

返回字符串的小写形式，语法如下：

```
LCase(string)
```

其中 string 参数是任意有效的字符串表达式。如果 string 参数中包含 Null，则返回 Null。
该函数只将大写字母转换成小写字母，所有小写字母和非字母字符保持不变。
例如：LCase("Apple") = "apple"。

5.1.6 UCase 函数

返回字符串的大写形式。语法如下：

```
UCase(string)
```

其中 string 参数是任意有效的字符串表达式。如果 string 参数中包含 Null，则返回 Null。
该函数只将小写字母被转换成大写字母，所有大写字母和非字母字符均保持不变。
例如：UCase("Apple") = "APPLE"

5.1.7 Left 与 LeftB 函数

1．Left 函数

返回指定数目的从字符串的左边算起的字符。语法如下：

```
Left(string, length)
```

参数说明：

- string：**字符串表达式，其最左边的字符被返回。如果 string 参数中包含 Null，则返回 Null。**
- length：**数值表达式，指明要返回的字符数目。如果是 0，返回零长度字符串（""）；如果大于或等于 string 参数中的字符总数，则返回整个字符串。**

可使用 Len 函数确定 string 参数中的字符数目。例如：Left("VBScript",3) = "VBS"

2．LeftB 函数

LeftB 函数的功能与 Left 函数相类似，不同之处在于 LeftB 函数返回的不是 length 个字符数，而是 length 个字节数，例如：

```
LeftB("VBScript",3) = "V"
LeftB("VBScript",4) = "VB"
```

注意：LeftB 函数与包含在字符串中的字节数据一起使用。length 不是指定返回的字符串数，而是字节数。

5.1.8　Right 与 RightB 函数

1．Right 函数

从字符串右边返回指定数目的字符，语法如下：

```
Right(string, length)
```

参数说明：

- string：字符串表达式，其最右边的字符被返回。如果 string 参数中包含 Null，则返回 Null。
- length：数值表达式，指明要返回的字符数目。如果为 0，返回零长度字符串；如果此数大于或等于 string 参数中的所有字符数目，则返回整个字符串。

例如：Right("VBScript",3) = "ipt"

2．RightB 函数

RightB 函数的功能与 Right 函数的功能相类似，不同之处在于 RightB 函数返回的不是 length 个字符数，而是 length 个字节数。

例如：

```
RightB("VBScript",3) = "t"
RightB("VBScript",4) = "pt"
```

5.1.9　Mid 与 MidB 函数

1．Mid 函数

从字符串中返回指定数目的字符。

```
Mid(string, start[, length])
```

参数说明：

- string：字符串表达式，将要从这个字符串中返回字符。如果 string 包含 Null，则返回 Null。

- start：string 中被提取的字符部分的开始位置。如果 start 超过了 string 中字符的数目，Mid 将返回零长度字符串（""）。
- length：要返回的字符数。如果省略或 length 超过文本的字符数（包括 start 处的字符），将返回字符串中从 start 到字符串结束的所有字符。

下面是利用 Mid 函数返回字符串中从第 6 个字符开始的 8 个字符：

```
Mid("I am a student!", 6, 9)
```

该表达式的返回值是“ a student ”。

2．MidB 函数

MidB 函数的功能与 Mid 函数的功能相类似，不同之处在于 MidB 函数返回的不是 length 个字符数，而是 length 个字节数。

例如：MidB("I am a student!", 5, 8) = "am a"

注意：使用 MidB 时，参数 Start 必须是奇数，否则将返回无法识别的乱码。

5.1.10 Len 与 LenB 函数

1．Len 函数

返回字符串内字符的数目。语法如下：

```
Len(string | varname)
```

参数说明：

- string：任意有效的字符串表达式。如果 string 参数包含 Null，则返回 Null。
- varname：任意有效的变量名。如果 varname 参数包含 Null，则返回 Null。

例如：Len("Hello,everyone!") = 15

2．LenB 函数

LenB 函数的功能与 Len 函数的功能相类似，不同之处在于 LenB 不是返回字符串中的字符数，而是返回用于代表字符串的字节数。

例如：LenB("Hello,everyone!") = 30

5.1.11 LTrim、RTrim 和 Trim 函数

Ltrim 函数返回不带前导空格的字符串，其语法为：LTrim(string)。

Rtrim 函数返回不带后续空格的字符串，其语法为：RTrim(string)。

Trim 函数返回不带前导与后续空格的字符串，其语法为：Trim(string)。

其中 string 参数是任意有效的字符串表达式。如果 string 参数中包含 Null，则返回 Null。

例如：

```
LTrim("    Student ")    = " Student"
RTrim("    Student ")    'MyVar 包含 "Student "
```

```
Trim("    Student ")    'MyVar 包含 "Student"
```

5.1.12　StrComp 函数

返回一个表明字符串比较结果的值。语法如下：

```
StrComp(string1, string2[, compare])
```

参数说明：

- string1：**必选项，任意有效的字符串表达式。**
- string2：**必选项，任意有效的字符串表达式。**
- compare：**可选项，指示在计算字符串时使用的比较类型的数值。如果省略，则执行二进制比较。有关数值，请参阅表** 5.5 **所示。**

表 5.5　StrComp **函数中** compare **的参数值**

常数	值	描述
vbBinaryCompare	0	**执行二进制比较。**
vbTextCompare	1	**执行文本比较。**

该函数的返回值，参见表 5.6 **所示。**

表 5.6　StrComp **函数的返回值**

条件	返回值
string1 **小于** string2	－1
string1 **等于** string2	0
string1 **大于** string2	1
string1 **或** string2 **为** Null	Null

下面例子利用 StrComp **函数返回字符串比较的结果。如果第** 3 **个参数为** 1 **执行文本比较；如果第** 3 **个参数为** 0**，省略执行二进制比较。**

```
Dim Str1, Str2, CompResult
Str1 = "VBScript"
Str2 = "vbscript"
CompResult = StrComp(Str1, Str2, 1)      '返回 0
CompResult = StrComp(Str1, Str2, 0)      '返回 －1
CompResult = StrComp(Str2, Str1)          '返回 1
```

5.1.13　StrReverse 函数

返回一个与指定字符串顺序相反的字符串。语法如下：

```
StrReverse(string1)
```

参数 string1 **是要进行字符反向的字符串。如果** string1 **是零长度字符串（""），则返回零长度字符串。如果** string1 **为** Null**，则会出现错误。**

例如：StrReverse("student")　= "tneduts"

5.1.14　String 函数

String **函数有两个参数，第一个是数字** N**，第二个是字符（或字符的** ASCII **码），返回值是由第二个字符重复** N **次组成的串。**

例如：Sting(5,65) ="AAAAA"　**或**　String(5,"A") = "AAAAA"

5.1.15　Space 函数

返回由指定数目的空格组成的字符串。语法如下：

```
Space(number)
```

参数 number **为字符串中用户所需的空格数。**

下面的例子利用 Space **函数返回由指定数目空格组成的字符串：**

```
Dim Str
Str = Space(5)                          ’返回具有 5 个空格的字符串
Str = "新北京" & Space(5) & "新奥运"        ’在两个字符串之间插入 5 个空格
```

5.1.16　Replace 函数

返回字符串，其中指定数目的某子字符串被替换为另一个子字符串。语法如下：

```
Replace(expression, find, replacewith[, compare[, count[, start]]])
```

参数说明：

- expression**：必选项，字符串表达式，它包含要替代的子字符串。**
- find**：必选项，被搜索的子字符串。**
- replacewith**：必选项，用于替换的子字符串。**
- start**：可选项，**expression **中开始搜索子字符串的位置。如果省略，默认值为** 1**。在和** count **关联时必须用。**
- count**：可选项，执行子字符串替换的数目。如果省略，默认值为**−1**，表示进行所有可能的替换。在和** start **关联时必须用。**
- compare**：可选项，指示在计算子字符串时使用的比较类型的数值。有关数值，请参阅表** 5.7 **所示。如果省略，缺省值为** 0**，这意味着必须进行二进制比较。**

表 5.7　Replace **函数中的** compare **参数的可选值**

常数	值	描述
vbBinaryCompare	0	执行二进制比较
vbTextCompare	1	执行文本比较

该函数的返回值，参见表 5.8 **所示。**

表 5.8　Replace 函数的返回值

条件	返回值
expression 为零长度	零长度字符串 ("")
expression 为 Null	错误
find 为零长度	expression 的副本
replacewith 为零长度	expression 的副本，其中删除了所有由 find 参数指定的内容
start > Len(expression)	零长度字符串
count 为 0	expression 的副本

Replace 函数的返回值是经过替换（从由 start 指定的位置开始到 expression 字符串的结尾）后的字符串，而不是原始字符串从开始至结尾的副本。

例如：

```
Replace("ABCeabce", "e", "D") =    "ABCDabcD"
Replace("ABCeabce ", "e", "D", 5，，－1, 1)    文本比较从第 3 个字符开始，返回 "abcD"
```

5.2　转 换 函 数

使用转换函数可以改变一个变量的子类型。通常，可以使用子类型数据转换函数书写代码，以显示某些操作的结果应当被表达为特定的数据类型，而非默认的数据类型。

5.2.1　CBool 函数

将一表达式的值转换为 Boolean 子类型。语法如下：

```
CBool(expression)
```

其中 expression 是任意有效的表达式。

如果 expression 是零，则返回 False；否则返回 True。如果 expression 不能解释为数值，则将发生运行时错误。

例如：

```
CBool(5 = 5) 的返回结果为 True
CBool(5 = 6) 的返回结果为 False
```

5.2.2　Cbyte 函数

将一表达式转换为 Byte 子类型。语法如下：

```
CByte(expression)
```

其中 expression 参数是任意有效的表达式。

CByte 函数用于进行从其他数据类型到 Byte 子类型的国际公认的格式转换。例如对十进制分隔符（如千分符）的识别，可能取决于系统的区域设置。

如果 expression 在 Byte 子类型可接受的范围（0～255 之间的整数）之外，则发生错误。

下面的例子利用 CByte 函数把 expression 转换为 Byte：

```
Dim DblVar, bytVar
DblVar = 236.2352                ’DblVar 是一个双精度值
BytVar = CByte(DblVar)           ’BytVar 包含 236
```

5.2.3 CCur 函数

将一表达式转换为 Currency 子类型。语法如下：

```
CCur(expression)
```

其中 expression 参数是任意有效的表达式。

CCur 函数用于进行从其他数据类型到 Currency 子类型的国际公认的格式转换。例如，对十进制分隔符和千位分隔符的识别取决于系统的区域设置。

下面的例子使用 CCur 函数将一个表达式转换成 Currency 类型：

```
Dim DblVar, CurVar
DblVar = 367.314582              ’DblVar 是双精度的
CurVar = CCur(DblVar * 2)        把 DblVar * 2 (734.629164) 的结果转换为 Currency (734.6292)
```

5.2.4 CDbl 函数

将一表达式转换为 Double 子类型。语法为：

```
CDbl(expression)
```

其中 expression 参数是任意有效的表达式。

CDbl 函数用于进行从其他数据类型到 Double 子类型的国际公认的格式转换。例如，十进制分隔符和千位分隔符的识别取决于系统的区域设置。

下面例子利用 CDbl 函数把 expression 转换为 Double：

```
Dim CurVar, DblVar
CurVar = CCur(324.546784)              ’CurVar 是 Currency 型（324.5468）
DblVar = CDbl(CurVar * 6.3 * 0.02)     ’把结果转换为 Double 型（40.8928968）
```

5.2.5 CInt 函数

将一表达式转换为 Integer 子类型。语法如下：

```
CInt(expression)
```

其中 expression 参数是任意有效的表达式。

CInt 函数用于进行从其他数据类型到 Integer 子类型的国际公认的格式转换。例如对十进制分隔符（如千分符）的识别，可能取决于系统的区域设置。

如果 expression 在 Integer 子类型可接受的范围（−32,768～32,767 之间的整数）之外，则发生错误。

下面例子利用 CInt 函数把值转换为 Integer：

```
Dim DblVar, IntVar
DblVar = 2345.5678          'DblVar 是 Double。
IntVar = CInt(DblVar)       'IntVar 包含 2346。
```

注意：CInt 不同于 Fix 和 Int 函数删除数值的小数部分，而是采用四舍五入的方式。 当小数部分正好等于 0.5 时，CInt 总是将其四舍五入成最接近该数的偶数。例如，0.5 四舍五入为 0，以及 1.5 四舍五入为 2.

5.2.6 CLng 函数

将一表达式转换为 Long 子类型。语法如下：

```
CLng(expression)
```

其中 expression 参数是任意有效的表达式。

CLng 函数用于进行从其他数据类型到 Long 子类型的的国际公认的格式转换。例如，对十进制分隔符和千位分隔符的识别取决于系统的区域设置。

如果 expression 取值不在 Long 子类型的允许范围（−2,147,483,648～2,147,483,647 之间的整数）内，则会出现错误。

下面的例子利用 CLng 函数把值转换为 Long：

```
Dim DblVal1, DblVal2, LngVar1, LngVar2
DblVal1= 35617.45
DblVal2= 35617.55              'DblVal1, DblVal2 是双精度值
LngVar1= CLng(DblVal1)         'LngVar1 包含 35617
LngVar2= CLng(DblVal2)         'LngVar2 包含 35618
```

注意：CLng 不同于 Fix 和 Int 函数删除小数部分，而是采用四舍五入的方式。当小数部分正好等于 0.5 时，CLng 函数总是将其四舍五入为最接近该数的偶数。如，0.5 四舍五入为 0，1.5 四舍五入为 2。

5.2.7 CStr 函数

将一表达式转换为 String 子类型。语法如下：

```
CStr(expression)
```

其中 expression 参数是任意有效的表达式。

CStr 函数用于替代 Str 函数来进行从其他数据类型到 String 子类型的国际公认的格式转换。例如对十进制分隔符的识别取决于系统的区域设置。

expression 根据表 5.9 决定返回的数据。

表 5.9 CStr 函数的 expression 参数及返回值

Expression 的子类型	CStr 返回值
Boolean	字符串，包含 True 或 False
Date	字符串，包含系统的短日期格式日期
Null	运行时错误
Empty	零长度字符串 ("")
Error	字符串，包含跟随有错误号码的单词 Error
其他数值	字符串，包含此数字

下面的例子利用 CStr 函数把数字转换为 String。

```
Dim DblVar, StrVar
DblVar = 567.231          'DblVar 是双精度值
StrVar = CStr(DblVar)     'StrVar 包含 "567.231"
```

5.2.8 CSng 函数

将一表达式转换为 Single 子类型。语法如下：

```
CSng(expression)
```

其中 expression 参数是任意有效的表达式。

CSng 函数用于进行从其他数据类型到 Single 子类型的国际公认的格式转换。例如，对十进制分隔符（如千分符）的识别取决于系统的区域设置。

如果 expression 在 Single 子类型允许的范围之外，则发生错误。

下面的例子利用 CSng 函数把值转换为 Single：

```
Dim DblVar1, DblVar2, SngVar1, SngVar2
DblVar1 = 65.1234115
DblVar2 = 65.1234555          'DblVar1, DblVar2 是双精度值
SngVar1 = CSng(DblVar1)       'SngVar1 包含 65.12341
SngVar2 = CSng(DblVar2)       'SngVar2 包含 75.12346
```

5.2.9 Cdate 函数

将一表达式转换为 Date 子类型。语法如下：

```
CDate(date)
```

date 参数是任意有效的日期表达式。

IsDate 函数用于判断 date 是否可以被转换为日期或时间。CDate 识别日期文字和时间文字，以及一些在可接受的日期范围内的数字。在将数字转换为日期时，数字的整数部分被转换为日期，分数部分被转换为从午夜开始计算的时间。

CDate 根据系统的区域设置识别日期格式。如果数据的格式不能被日期设置识别，则不能判断年、月、日的正确顺序。另外，如果长日期格式包含表示星期几的字符串，则不

能被识别。

下面的例子使用 CDate 函数将字符串转换成日期类型。一般不推荐使用硬件译码日期和时间作为字符串（下面的例子已体现），而是使用时间和日期文字（如 #10/19/1962#, #4:45:23 PM#）。

```
DateVar = "April 19, 2001"          ' 定义日期
ShortDateVar = CDate(DateVar)       ' 转换为日期数据类型 01-04-19
TimeVar = "6:35:36 PM"              ' 定义时间
ShortTimeVar = CDate(TimeVar)       ' 转换为日期数据类型 18:35:36
```

5.3　格式化函数

5.3.1　FormatCurrency 函数

返回格式化后的货币类型数值，货币符号的设置基于控制面板的区域属性设置。语法如下：

FormatCurrency(expression[,NumDigitsAfterDecimal[,IncludeLeadingDigit,UseParensForNegativeNumbers [,GroupDigits]]]))

参数说明：

- expression：必选项，要被格式化的表达式。
- NumDigitsAfterDecimal：可选项，指示小数点右侧显示位数的数值。默认值为 -1，指示使用的是计算机的区域设置。
- IncludeLeadingDigit：可选项，三态常数，指示是否显示小数值小数点前面的零。有关数值，请参阅表 5.10。
- UseParensForNegativeNumbers：可选项，三态常数，指示是否将负值置于括号中。有关数值，请参阅表 5.10。
- GroupDigits：可选项，三态常数，指示是否使用计算机区域设置中指定的数字分组符号将数字分组。有关数值，请参阅表 5.10。

表 5.10　FormatCurrency 函数的 IncludeLeadingDigit、UseParensForNegativeNumber 和 GroupDigits 参数的值

常数	值	描述
TristateTrue	－1	True
TristateFalse	0	False
TristateUseDefault	－2	使用计算机区域设置中的设置

当省略一个或多个可选项参数时，由计算机区域设置提供被省略参数的值，与货币值相关的货币符号的位置由系统的区域设置决定。

注意：除“显示起始的零”设置来自区域设置的“数字”标签外，所有其他设置信息均取自区域设置的货币标签。

下面例子利用 FormatCurrency **函数把** expression **格式化为** currency**，并且赋值给** CurrencyVar**：**

```
Dim CurrencyVar
CurrencyVar = FormatCurrency(580)   'CurrencyVar 包含 $580.00
```

5.3.2 FormatDateTime 函数

将一表达式格式化为日期或时间子类型。语法如下：

```
FormatDateTime(Date[, NamedFormat])
```

参数说明：

- Date**：必选项，要被格式化的日期表达式。**
- NamedFormat**：可选项，指示所使用的日期/时间格式的数值，如果省略，则使用** vbGeneralDate**。详见表** 5.11 **所示。**

表 5.11 FormatDateTime **函数的** NamedFormat **的参数值**

常数	值	描述
vbGeneralDate	0	显示日期和时间。如果有日期部分，则将该部分显示为短日期格式。如果有时间部分，则将该部分显示为长时间格式。如果都存在，则显示所有部分
vbLongDate	1	使用计算机区域设置中指定的长日期格式显示日期
vbShortDate	2	使用计算机区域设置中指定的短日期格式显示日期
vbLongTime	3	使用计算机区域设置中指定的时间格式显示时间
vbShortTime	4	使用 24 小时格式（hh:mm）显示时间

例如：FormatDateTime("23/04/2001", 1) = "2001 **年** 4 **月** 23 **日**"

注意：表达式 Date **的年份应写为** 4 **位。**

5.3.3 FormatNumber 函数

将一表达式格式化为数值子类型。语法如下：

```
FormatNumber(expression[,NumDigitsAfterDecimal[,IncludeLeadingDigit,UseParensForNegativeNumbers[,GroupDigits]]])
```

参数说明：

- expression**：必选项，要被格式化的表达式。**
- NumDigitsAfterDecimal**：可选项，指示小数点右侧显示位数的数值。默认值为**−1**，指示使用的是计算机的区域设置。**

- IncludeLeadingDigit：可选项，三态常数，指示是否显示小数值小数点前面的零。有关数值，请参阅表 5.12。
- UseParensForNegativeNumbers：可选项，三态常数，指示是否将负值置于括号中。有关数值，请参阅表 5.12。
- GroupDigits：可选项，三态常数，指示是否使用计算机区域设置中指定的数字分组符号将数字分组。有关数值，请参阅表 5.12。

表 5.12　FormatNumber 函数的 IncludeLeadingDigit、UseParensForNegativeNumbers 和 GroupDigits 的参数值

常数	值	描述
TristateTrue	－1	True
TristateFalse	0	False
TristateUseDefault	－2	使用计算机区域设置中的设置

当省略一个或多个可选项参数时，由计算机区域设置提供被省略参数的值。

注意：所有设置信息均取自区域设置的“数字”标签。

下面例子利用 FormatNumber 函数把数值格式化为带四位小数点的数：

```
FormatNumberVar = FormatNumber(1/3,4)    'FormatNumberVar 包含 0.3333
```

5.3.4　FormatPercent 函数

将一表达式格式化为尾随有%符号的百分比（乘以 100）。语法如下：

```
FormatPercent(expression[,NumDigitsAfterDecimal[,IncludeLeadingDigit [,UseParensForNegativeNumbers
[,GroupDigits]]]])
```

参数说明：

- expression：必选项，要被格式化的表达式。
- NumDigitsAfterDecimal：可选项，指示小数点右侧显示位数的数值。默认值为－1，指示使用的是计算机的区域设置。
- IncludeLeadingDigit：可选项，三态常数，指示是否显示小数值小数点前面的零。有关数值，请参阅表 5.13。
- UseParensForNegativeNumbers：可选项，三态常数，指示是否将负值置于括号中。有关数值，请参阅表 5.13。
- GroupDigits：可选项，三态常数，指示是否使用计算机区域设置中指定的数字分组符号将数字分组。有关数值，请参阅表 5.13。

表 5.13 FormatPercent 函数的 IncludeLeadingDigit、UseParensForNegativeNumbers 和 GroupDigits 的参数值

常数	值	描述
TristateTrue	－1	True
TristateFalse	0	False
TristateUseDefault	－2	使用计算机区域设置中的设置

当省略一个或多个可选项参数时，由计算机区域设置提供被省略参数的值。

注意：所有设置信息均取自区域设置的“数字”标签。

例如：FormatPercent(5/63) = 7.94%。

5.4 数 学 函 数

5.4.1 Abs 函数

返回数字的绝对值。语法如下：

```
Abs(number)
```

number 参数可以是任意有效的数值表达式。如果 number 包含 Null，则返回 Null；如果是未初始化变量，则返回 0。

数字的绝对值是无符号的数值大小。例如，Abs(-5) 和 Abs(5) 都返回 5。

5.4.2 Atn 函数

返回数值的反正切值。语法如下：

```
Atn(number)
```

number 参数可以是任意有效的数值表达式。

5.4.3 Cos 函数

返回某个角的余弦值。语法如下：

```
Cos(number)
```

number 参数可以是任何将某个角表示为弧度的有效数值表达式。

5.4.4 Exp 函数

返回 e（自然对数的底）的幂次方。语法如下：

```
Exp(number)
```

number 参数可以是任意有效的数值表达式。

5.4.5　Int 与 Fix 函数

返回数字的整数部分。语法如下：

```
Int(number)
Fix(number)
```

number 参数可以是任意有效的数值表达式。如果 number 参数包含 Null，则返回 Null。

Int 和 Fix 函数都删除 number 参数的小数部分并返回以整数表示的结果。

Int 和 Fix 函数的区别在于如果 number 参数为负数时，Int 函数返回小于或等于 number 的第一个负整数，而 Fix 函数返回大于或等于 number 参数的第一个负整数。例如，Int 将－10.5 转换为－11，而 Fix 函数将－10.5 转换为－10。

下面的例子说明 Int 和 Fix 函数如何返回数字的整数部分：

```
MyNumber = Int(101.8)      ' 返回 101
MyNumber = Fix(101.2)      ' 返回 101
MyNumber = Int(－101.8)    ' 返回 －102
MyNumber = Fix(－101.8)    ' 返回－101
MyNumber = Int(－101.2)    ' 返回 －102
MyNumber = Fix(－101.2)    ' 返回 －101
```

5.4.6　Log 函数

返回数值的自然对数，语法如下：

```
Log(number)
```

number 参数是任意大于 0 的有效数值表达式。

5.4.7　Rnd 函数

返回一个随机数，语法如下：

```
Rnd (number)
```

number 参数可以是任意有效的数值表达式。

Rnd 函数返回一个小于 1 但大于或等于 0 的值。number 的值决定了 Rnd 生成随机数的方式，请参见表 5.14。

表 5.14　Rnd 函数的 number 参数及其含义

number	Rnd 生成的随机数
小于零	每次都相同的值，使用 number 作为种子
大于零	序列中的下一个随机数
等于零	最近生成的数
省略	序列中的下一个随机数

因每一次连续调用 Rnd 函数时都用序列中的前一个数作为下一个数的种子，所以对于任何最初给定的种子都会生成相同的数列。

在调用 Rnd 之前，先使用无参数的 Randomize 语句初始化随机数生成器，该生成器具有基于系统计时器的种子。

要产生指定范围的随机整数，请使用以下公式：

```
Int((upperbound – lowerbound + 1) * Rnd + lowerbound)
```

这里，upperbound 是此范围的上界，而 lowerbound 是此范围内的下界。例如要产生一个 1～15 之间的随机数，则使用：

```
int(15 – 1 + 1 )* rnd + 1)  即 int(15 * rnd + 1)
```

由于 rnd 产生了一个 0～1 之间的值，所以乘以 15 之后的值在 0～15 之间，加 1 之后的值大于 1 小于 16，用 int 取整之后正好是在 1～15（包括 1 和 15）之间的整数。

注意：要重复随机数的序列，请在使用数值参数调用 Randomize 之前，立即用负值参数调用 Rnd。使用同样 number 值的 Randomize 不能重复先前的随机数序列。

5.4.8 Round 函数

返回按指定位数进行四舍五入的数值。其语法为：

```
Round(expression[, numdecimalplaces])
```

参数说明：

- expression：必选项，被四舍五入的数值表达式。
- Numdecimalplaces：可选项，数字类型，表明小数点右边有多少位进行四舍五入。如果省略，则 Round 函数返回整数。

下面的例子利用 Round 函数将数值四舍五入到两位小数：

```
Dim sngVar1, sngVar2
SngVar1 = 6.123456
SngVar2 = Round(sngVar1, 2)
```

执行上面语句之后，sngVar2 的值为 6.12。

5.4.9 Sgn 函数

返回表示数字符号的整数，其语法为：

```
Sgn(number)
```

参数 number 可以是任意有效的数值表达式。

该函数的返回值，参见表 5.15 所示。

表 5.15　Sgn 函数的返回值

number 值	Sgn 的返回值
大于零	1
等于零	0
小于零	－1

number 参数的符号决定 Sgn 函数的返回值。

例如：

```
MySign = Sgn(14) = 1
MySign = Sgn(－14) =  －1
MySign = Sgn(0) = 0
```

5.4.10　Sin 函数

返回某个角的正弦值。语法如下：

```
Sin(number)
```

参数 number 可以是任何将某个角表示为弧度的有效数值表达式。

Sin 函数结果的范围在－1～1 之间。

提示：将角度乘以 pi/180 即可转换为弧度，将弧度乘以 180/pi 即可转换为角度。

5.4.11　Sqr 函数

返回数值的平方根。语法如下：

```
Sqr(number)
```

参数 number 可以是任意有效的大于或等于零的数值表达式。如果 number 为空或小于零将产生一个运行时的错误。

5.4.12　Tan 函数

返回某个角的正切值。语法如下：

```
Tan(number)
```

参数 number 可以是任何将某个角表示为弧度的有效数值表达式。

5.5　日期和时间函数

5.5.1　Date 函数

返回当前系统日期。语法如下：

Date

该函数没有参数，返回日期值的格式信息取自区域设置的“日期”标签。但是可以使用格式化函数 FormatDateTime 改变它的格式。

5.5.2 Time 函数

返回当前系统时间。语法如下：

Time

该参数没有参数，返回时间的格式信息取自区域设置的“事件”标签，但是可以使用格式化函数 FormatDateTime 改变它的格式。

5.5.3 DateAdd 函数

向指定的日期添加指定的时间间隔。语法如下：

DateAdd(interval, number, date)

参数说明：

- interval：必选项，字符串表达式，表示要添加的时间间隔。有关数值，请参阅表 5.16。
- number：必选项，数值表达式，表示要添加的时间间隔的个数。数值表达式可以是正数（得到未来的日期）或负数（得到过去的日期）。
- date：必选项，Variant 或要添加 interval 的表示日期的文字。

表 5.16 DateAdd 函数的 interval 的参数值

设置	描述
yyyy	年
q	季度
m	月
y	一年的日数
d	日
w	一周的日数
ww	周
h	小时
n	分钟
s	秒

可用 DateAdd 函数从日期中添加或减去指定时间间隔。例如，可以使用 DateAdd 从当天算起 10 天以后的日期或从现在算起 30 分钟以后的时间。要向 date 添加以“日”为单位的时间间隔，可以使用“一年的日数”（“y”）、“日”（“d”）或“一周的日数”（“w”）。

例如，DateAdd("yyyy",2,"2001/4/23")将得到从 2001 年 4 月 23 日开始两年以后的日期，即 2003 年 4 月 23 日。

DateAdd 函数不会返回无效日期。如下例子表示将 2001 年 3 月 31 日加上一个月：

```
DateVar = DateAdd("m", 1, "31-Mar.-2001")
```

在这个例子中，DateAdd 返回 2001 年 4 月 30 日，而不是 2001 年 4 月 31 日。

如果计算的日期是在公元 100 年之前，则会产生错误。

如果 number 不是 Long 型值，则在计算前四舍五入为最接近的整数。

5.5.4　DateDiff 函数

返回两个日期之间的时间间隔，其语法为：

```
DateDiff(interval, date1, date2 [,firstdayofweek[, firstweekofyear]])
```

参数说明：

- interval 必选项，String expression 表示用于计算 date1 和 date2 之间的时间间隔。有关数值，请参阅表 5.17。
- date1, date2：必选项，日期表达式，用于计算的两个日期。
- firstdayofweek：可选项，指定星期中第一天的常数，如果没有指定，则默认为星期日。有关数值，请参阅表 5.18。
- firstweekofyear：可选项，指定一年中第一周的常数。如果没有指定，则默认为 1 月 1 日所在的星期。有关数值，请参阅表 5.19。

表 5.17　DateDiff 函数中的 interval 参数值

设置	描述
yyyy	年
q	季度
n	月
y	一年的日数
d	日
w	一周的日数
ww	周
h	小时
m	分钟
s	秒

表 5.18 DateDiff 函数的 firstdayofweek 参数值

常数	值	描述
vbUseSystem	0	使用区域语言支持（NLS）API 设置
vbSunday	1	星期日（默认）
vbMonday	2	星期一
vbTuesday	3	星期二
vbWednesday	4	星期三
vbThursday	5	星期四
vbFriday	6	星期五
vbSaturday	7	星期六

表 5.19 DateDiff 函数的 firstweekofyear 参数值

常数	值	描述
vbUseSystem	0	使用区域语言支持（NLS）API 设置
vbFirstJan1	1	由 1 月 1 日所在的星期开始（默认）
vbFirstFourDays	2	由在新年中至少有四天的第一周开始
vbFirstFullWeek	3	由在新的一年中第一个完整的周开始

DateDiff 函数用于判断在两个日期之间存在的指定时间间隔的数目。例如可以使用 DateDiff 计算两个日期相差的天数、月数、年数，或者当天到当年最后一天之间的星期数。

要计算 date1 和 date2 相差的天数，可以使用“一年的日数”（“y”）或“日”（“d”）。

当 interval 为“一周的日数”（“w”）时，DateDiff 返回两个日期之间的星期数。

如果 date1 是星期一，则 DateDiff 计算到 date2 之前星期一的数目。此结果包含 date2 而不包含 date1。

如果 date2 是星期日，DateDiff 将计算 date2，但即使 date1 是星期日，也不会计算 date1。

如果 interval 是“周”（“ww”），则 DateDiff 函数返回日历表中两个日期之间的星期数。函数计算 date1 和 date2 之间星期日的数目。

如果 date1 晚于 date2，则 DateDiff 函数返回负数。

firstdayofweek 参数会对使用“w”和“ww”间隔符号的计算产生影响。

如果 date1 或 date2 是日期文字，则指定的年度会成为日期的固定部分。但是如果 date1 或 date2 被包括在引号（" "）中并且省略年份，则在代码中每次计算 date1 或 date2 表达式时，将插入当前年份。这样就可以编写适用于不同年份的程序代码。

在 interval 为“年”（“yyyy”）时，比较 12 月 31 日和来年的 1 月 1 日，虽然实际上只相差一天。DateDiff 返回 1 表示相差一个年份。

下面的过程利用 DateDiff 函数显示今天与给定日期之间的间隔天数：

```
Function DiffADate(theDate)
DiffADate = "从当天开始的天数：" & DateDiff("d", Now, theDate)
```

End Function

5.5.5　DatePart 函数

返回给定日期的指定部分，语法如下：

DatePart(interval, date[, firstdayofweek[, firstweekofyear]])

参数说明：

- interval：**必选项，字符串表达式表示要返回的时间间隔。有关数值，请参阅表** 5.20。
- date：**必选项，要计算的日期表达式。**
- firstdayof week：**可选项，指定星期中的第一天的常数，如果没有指定，则默认为星期日。有关数值，请参阅表** 5.21。
- firstweekofyear：**可选项，指定一年中第一周的常数，如果没有指定，则默认为** 1 **月** 1 **日所在的星期。有关数值，请参阅表** 5.22。

表 5.20　DatePart **函数的** interval **参数值**

设置	描述
yyyy	年
q	季度
m	月
y	一年的日数
d	日
w	一周的日数
ww	周
h	小时
n	分钟
s	秒

表 5.21　DatePart **函数的** firstdayofweek **参数值**

常数	值	描述
vbUseSystem	0	使用区域语言支持（NLS）API 设置
vbSunday	1	星期日（默认）
vbMonday	2	星期一
vbTuesday	3	星期二
vbWednesday	4	星期三
vbThursday	5	星期四
vbFriday	6	星期五
vbSaturday	7	星期六

表 5.22 DatePart 函数的 firstweekofyear 参数值

常数	值	描述
vbUseSystem	0	使用区域语言支持（NLS）API 设置
vbFirstJan1	1	由 1 月 1 日所在的星期开始（默认）
vbFirstFourDays	2	由在新年中至少有四天的第一周开始
vbFirstFullWeek	3	由在新的一年中第一个完整的周（不跨年度）开始

DatePart 函数用于计算日期并返回指定的时间间隔。例如使用 DatePart 计算某一天是星期几或当前的时间。

firstdayofweek 参数会影响使用“w”和“ww”间隔符号的计算。

如果 date 是日期文字，则指定的年度会成为日期的固定部分。但是如果 date 被包含在引号（" "）中，并且省略年份，则在代码中每次计算 date 表达式时，将插入当前年份。这样就可以编写适用于不同年份的程序代码。

下面的示例利用 DatePart 函数获得日期并显示该日所在的季节。

```
TheQuarter = DatePart("q", "2001/4/24")      'TheQuarter 包含“2”
```

5.5.6 DateSerial 函数

对于指定的年、月、日，返回 Date 子类型的 Variant，语法如下：

```
DateSerial(year, month, day)
```

参数说明：

- year：从 100～9999 之间的数字或数值表达式。
- month：任意数值表达式。
- day：任意数值表达式。

要指定日期，如 2001 年 4 月 24 日，对应 DateSerial 函数中每个参数的取值范围都应该是可接受的，即日的取值应在 1～31 之间，月的取值应在 1～12 之间。但是，也可以使用表示某日之前或之后的年、月、日数目的数值表达式，为每个参数指定相对日期。

以下例子中使用了数值表达式代替绝对日期。在这里，DateSerial 函数返回 2001 年 4 月 24 日之前八年（2001 – 8）零五个月（4 – 5）又三天（24 – 3）的日期：即 1992 年 11 月 21 日。

```
DateVar = DateSerial(2001- 8, 4 - 5, 24 - 3)
```

对于 year 参数，若取值范围是 0～99，则被解释为 1900～1999 年。对于此范围之外的 year 参数，则使用四位数字表示年份（例如 1800 年）。

当任何一个参数的取值超出可接受的范围时，则会适当地进位到下一个较大的时间单位。例如，如果指定了 45，则这个天数被解释成一个月加上多出来的日数，多出来的日数取决于其年份和月份。但是如果参数值超出 -32,768 到 32,767 的范围，或者由 3 个参数

指定（无论是直接还是通过表达式指定）的日期超出了可以接受的日期范围，就会发生错误。

5.5.7　DateValue 函数

返回 Date 子类型的 Variant，语法如下：

```
DateValue(date)
```

date 参数应是字符串表达式，表示从 100 年 1 月 1 日到 9999 年 12 月 31 日中的一个日期。但是，date 也可以是表示上述范围内的日期、时间或日期时间混合的任意表达式。

如果 date 参数包含时间信息，则 DateValue 不会返回时间信息。但是如果 date 包含无效的时间信息（如 "100:100"），就会出现错误。

如果 date 是某一字符串，其中仅包含由有效的日期分隔符分隔开的数字，则 DateValue 将会根据为系统指定的短日期格式识别月、日和年的顺序。DateValue 还会识别包含月份名称（无论是全名还是缩写）的明确日期。例如，除了能够识别 04/26/2001 和 04/26/2001 之外，DateValue 还能识别 April 26 , 2001 和 Apr 26, 2001。

如果省略了 date 的年份部分，DateValue 将使用计算机系统日期中的当前年份。

下面的例子利用 DateValue 函数将字符串转化成日期。也可以利用日期文字直接将日期分配给 Variant 变量，例如，DateVar = #24/04/2001#。

```
Dim DateVar
DateVar = DateValue("April 26, 2001")      ' 返回日期
```

5.5.8　Day 函数

得到给定日期型参数的日子部分。返回一个 1～31 之间的一个整数（包括 1 和 31），代表一月中的某日。其语法为：

```
Day(date)
```

date 参数是任意可以代表日期的表达式。如果 date 参数中包含 Null，则返回 Null。

下面例子利用 Day 函数得到一个给定日期的日子部分：

```
DayVar = Day("Apr 27, 2001")      'DayVar 包含 27
```

5.5.9　Month 函数

得到给定日期型参数的“月”部分。返回 1～12 之间的一个整数（包括 1 和 12），代表一年中的某月。其语法为：

```
Month(date)
```

date 参数是任意可以代表日期的表达式。如果 date 参数中包含 Null，则返回 Null。

下面的例子利用 Month 函数得到给定日期的月部分：

```
Dim MonthVar
MonthVar = Month("Apr 27, 2001")    'MonthVar 包含 4
```

5.5.10 Year 函数

得到给定日期型参数的“年”部分，返回一个代表某年的整数。其语法为：

```
Year(date)
```

date 参数是任意可以代表日期的参数。如果 date 参数中包含 Null，则返回 Null。下面例子利用 Year 函数得到指定日期的年份：

```
Dim    YearVar
YearVar = Year("Apr 27,2001")              'YearVar 包含 2001
```

5.5.11 MonthName 函数

返回表明指定月份的字符串。其语法为：

```
MonthName(month[, abbreviate])
```

参数说明：

- month：必选项，月份的数值定义。例如，一月是 1，二月是 2，以此类推。
- abbreviate：可选项，Boolean 值，表明月份名称是否简写。如果省略，默认值为 False，即不简写月份名称。

下面的例子利用 MonthName 函数为日期表达式返回月份的缩写：

```
MonthNameVar1 = MonthName(10, True)       'MonthNameVar1 包含“Oct”
MonthNameVar2 = MonthName(4, false)       'MonthNameVar2 包含“April”
```

5.5.12 WeekDay 函数

返回代表一星期中某天的整数，表明给定日期是一星期中的星期几。其语法为：

```
Weekday(date, [firstdayofweek])
```

参数说明：

- date：可以代表日期的任意表达式。如果 date 参数中包含 Null，则返回 Null。
- firstdayofweek：指定星期中第一天的常数。如果省略，默认使用 vbSunday。设置值同表 5.18。

该函数的返回值，参见表 5.23 所示。

表 5.23 Weekday 函数的返回值

常数	值	描述
vbSunday	1	星期日
vbMonday	2	星期一
vbTuesday	3	星期二

（续表）

常数	值	描述
vbWednesday	4	星期三
vbThursday	5	星期四
vbFriday	6	星期五
vbSaturday	7	星期六

下面例子利用 Weekday 函数得到指定日期为星期几：

```
Dim    WeekDayVar
WeekDayVar = Weekday("Apr 27,2001")       'WeekDayVar 包含 6，表明这天是星期五
```

5.5.13　WeekDayName 函数

返回一个字符串，表示一星期中指定的某一天。其语法为：

```
WeekdayName(weekday, abbreviate, firstdayofweek)
```

参数说明：

- weekday：必选项。星期中某天的数值定义。各天的数值定义取决于 firstdayofweek 参数设置。
- abbreviate：可选项。Boolean 值，指明是否缩写表示星期各天的名称。如果省略，默认值为 False，即不缩写星期各天的名称。
- firstdayofweek：可选项。指明星期第一天的数值。关于数值，请参阅表 5.18。

下面例子利用 WeekDayName 函数返回指定的某一天：

```
Dim WeekDayNameVar
WeekDayNameVar = WeekDayName(6, True)    'WeekDayNameVar 包含“星期五”
```

5.5.14　Hour 函数

返回 0～23 之间的一个整数（包括 0 和 23），代表一天中的某一小时。其语法为：

```
Hour(time)
```

参数 time 是任意可以代表时间的表达式。如果 time 参数中包含 Null，则返回 Null。下面的例子利用 Hour 函数得到当前时间的小时：

```
HourVar = Hour(Now)              'HourVar 包含代表当前时间的小时数
```

5.5.15　Minute 函数

返回 0～59 之间的一个整数（包括 0 和 59），代表一小时内的某一分钟。其语法为：

```
Minute(time)
```

参数 time 是任意可以代表时间的表达式。如果 time 参数包含 Null，则返回 Null。下面的例子利用 Minute 函数返回当前时间的分钟数：

```
Dim MinuteVar
MinuteVar = Minute(Now)          'MinuteVar 包含代表当前时间的分钟数
```

5.5.16 Second 函数

返回 0～59 之间的一个整数（包括 1 和 59），代表一分钟内的某一秒。其语法为：

```
Second(time)
```

time 参数是任意可以代表时间的表达式。如果 time 参数中包含 Null，则返回 Null。下面的例子利用 Second 函数返回当前秒：

```
Dim SecondVar
SecondVar = Second(Now)              ' SecondVar 包含代表当前秒的数字
```

5.5.17 Now 函数

根据计算机系统设定的日期和时间返回当前的日期和时间值。其语法为：

```
Now
```

该函数没有参数。

5.5.18 TimeSerial 函数

返回一个 Date 子类型的 Variant，含有指定时、分、秒的时间。其语法为：

```
TimeSerial(hour,minute, second)
```

参数说明：

- hour：其值为从 0（12:00 A.M.）～23（11:00 P.M.）的数值或数值表达式。
- minute：任意数值表达式。
- second：任意数值表达式。

要指定一时刻，如 22:31:45，TimeSerial 的参数取值应在可接受的范围内；也就是说，小时应介于 0~23 之间，分和秒应介于 0~59 之间。但是，可以使用数值表达式为每个参数指定相对时间，这一表达式代表某时刻之前或之后的时、分或秒数。

下面的示例使用绝对时间数的表达式。TimeSerial 函数返回中午前 4（12~4）小时前的 15 分钟（-15），或 7:45:00 A.M.。

```
Dim TimeSerialVar
TimeSerialVar    = TimeSerial(12 -4, -15, 0)          ' 返回 7:45:00 AM
```

当任何一个参数的取值超出可接受的范围时，它会正确地进位到下一个较大的时间单位中。例如，如果指定了 90 分钟，则这个时间被解释成一小时三十分钟。但是，如果任何一个参数值超出-32768～32767 的范围，就会导致错误。如果使用 3 个参数直接指定的时间

或通过表达式计算出的时间超出可接受的日期范围，也会导致错误。

5.5.19　TimeValue 函数

返回包含时间的 Date 子类型的 Variant。其语法为：

```
TimeValue(time)
```

time 参数通常是代表从 0:00:00（12:00:00 A.M.）～23:59:59（11:59:59 P.M.）的字符串表达式（包括 0:00:00 和 23:59:59）。不过，time 也可以是代表该范围内任何时间的表达式。如果 time 参数包含 Null，则返回 Null。

可以采用 12 或 24 小时时钟格式输入时间。例如 "3:35PM" 和 "15:35" 都是有效的 time 参数。如果 time 参数包含日期信息，TimeValue 函数并不返回日期信息。但是，如果 time 参数包含无效的日期信息，则会出现错误。

下面的示例利用 TimeValue 函数将字符串转化为时间。也可以用日期文字直接赋时间给 Variant 类型的变量，例如，TimeVar = #5:35:26 PM#。

```
Dim TimeValueVar
TimeValueVar = TimeValue("4:35:17 PM")      'TimeValueVar 包含 "16:35:17"
```

5.6　布 尔 函 数

布尔函数的返回值总是真（True）或假（False），由于在 VBScript 中只有一种类型的变量，即 Variant，但很多时候需要进一步得到它的子类型。布尔函数可以判断变量的子类型。运用这些函数，能够根据变量的子类型，对变量进行合适的处理。

5.6.1　IsArray 函数

返回 Boolean 值，指明某变量是否为数组。其语法为：

```
IsArray(varname)
```

varname 参数可以是任意变量。

如果变量是数组，IsArray 函数返回 True；否则，函数返回 False。当变量中包含有数组时，使用 IsArray 函数很有效。

下面的例子利用 IsArray 函数验证 CheckVar 是否为一数组：

```
Dim CheckVar
Dim ArrVar (3)
ArrVar (0) = "A"
ArrVar (1) = "B"
ArrVar (2) = "C"
CheckVar = IsArray(ArrVar)        'CheckVar 包含 "True"
```

5.6.2 IsDate 函数

返回 Boolean 值，指明某表达式是否可以转换为日期。语法如下：

```
IsDate(expression)
```

expression 参数可以是任意可被识别为日期和时间的日期表达式或字符串表达式。

如果表达式是日期或可合法地转化为有效日期，则 IsDate 函数返回 True；否则函数返回 False。在 Microsoft Windows 操作系统中，有效的日期范围为公元 100 年 1 月 1 日到公元 9999 年 12 月 31 日。合法的日期范围随操作系统不同而不同。

下面的例子利用 IsDate 函数决定表达式是否能转换为日期型：

```
Dim DateVar1, DateVar2, DateVar3, CheckVar
DateVar1 = "April 27, 2001"
DateVar2 = #04/27/2001#:;
DateVar3 = "yes"
CheckVar = IsDate(DateVar1)          ’返回 True
CheckVar = IsDate(DateVar2)          ’返回 True
CheckVar = IsDate(DateVar3)          ’返回 False
```

5.6.3 IsEmpty 函数

返回 Boolean 值，指明变量是否已初始化。其语法为：

```
IsEmpty(expression)
```

expression 参数可以是任意表达式。然而，由于 IsEmpty 用于判断一个变量是否已初始化，故 expression 参数经常是一个变量名。

在声明变量之后，变量自动被赋值为 Empty，或者显式地将变量的值赋为 Empty。这时，使用 IsEmpty 判断，返回值为 True；否则函数返回 False。如果 expression 包含一个以上的变量，则总是返回 False。

下面的例子利用 IsEmpty 函数决定变量是否能被初始化：

```
Dim MyVar, CheckVar
CheckVar = IsEmpty(MyVar)            ’返回 True
MyVar = “This is a test.”            ' 为 MyVar 赋值
CheckVar = IsEmpty(MyVar)            ' 返回 False
MyVar = Empty                        ' 赋为 Empty
CheckVar = IsEmpty(MyVar)            ' 返回 True
```

5.6.4 IsNull 函数

返回 Boolean 值，指明表达式是否不包含任何有效数据。其语法为：

```
IsNull(expression)
```

expression 参数可以是任意表达式。

如果 expression 为 Null，则 IsNull 返回 True，即表达式不包含有效数据，否则 IsNull

返回 False。如果 expression 由多个变量组成，则表达式的任何组成变量中的 Null 都会使整个表达式返回 True。

Null 值指出变量不包含有效数据。在前面我们讲过 Null 与 Empty 不同，后者指出变量未经初始化。Null 与零长度字符串（""）也不同，零长度字符串往往指的是空串。

重点使用 IsNull 函数可以判断表达式是否包含 Null 值。在某些情况下想使表达式取值为 True，例如 If Var=Null 和 If Var<>Null，但它们通常总是为 False。这是因为任何包含 Null 的表达式本身就为 Null，所以表达式的结果为 False。

下面的例子利用 IsNull 函数决定变量是否包含 Null：

```
Dim MyVar, CheckVar
CheckVar = IsNull(MyVar)        ' 返回 False
MyVar = Null                    ' 赋为 Null
CheckVar = IsNull(MyVar)        ' 返回 True
MyVar = Empty                   ' 赋为 Empty
CheckVar = IsNull(MyVar)        ' 返回 False
```

提示：这个函数很有用，在很多时候，必须判断一个变量是否 isNull 之后才能够进行相应的处理，否则直接去操作的话，往往会发生错误。因为不能对一个包含 Null 的变量进行转换、运算等操作。

5.6.5 IsNumeric 函数

返回 Boolean 值，指明表达式的值是否为数字。其语法为：

```
IsNumeric(expression)
```

expression 参数可以是任意表达式。

如果整个 expression 被识别为数字，IsNumeric 函数返回 True；否则函数返回 False。如果 expression 是日期表达式，IsNumeric 函数返回 False。

下面的例子利用 IsNumeric 函数决定变量是否可以作为数值：

```
Dim MyVar, CheckVar
MyVar = 60                      ' 赋值
MyCheck = IsNumeric(MyVar)      ' 返回 True
MyVar = "123.425"               ' 赋值
MyCheck = IsNumeric(MyVar)      ' 返回 True
MyVar = "123ABC"                ' 赋值
CheckVar = IsNumeric(MyVar)     ' 返回 False
```

5.6.6 IsObject 函数

返回 Boolean 值，指明表达式是否引用了有效的 Automation 对象。其语法为：

```
IsObject(expression)
```

expression 参数可以是任意表达式。

如果 expression **是** Object **子类型变量或用户自定义的对象，则** IsObject **返回** True **；否则函数返回** False**。**

下面的例子利用 IsObject **函数决定标识符是否代表对象变量：**

```
Dim IntVar, ObjectVar ,CheckVar
Set ObjectVar = Me
CheckVar = IsObject(ObjectVar)          ’返回 True
CheckVar = IsObject(IntVar)             ’返回 False
```

5.6.7 TypeName 函数

返回一个字符串，提供有关变量的 Variant **子类型信息。其语法为：**

```
TypeName(varname)
```

varname **参数是必选项，可以是任何变量。**

该函数的返回值，参见表 5.24 **所示。**

表 5.24 TypeName **函数的返回值**

值	**描述**
Byte	**字节值**
Integer	**整型值**
Long	**长整型值**
Single	**单精度浮点值**
Double	**双精度浮点值**
Currency	**货币值**
Decimal	**十进制值**
Date	**日期或时间值**
String	**字符串值**
Boolean	Boolean **值；**True **或** False
Empty	**未初始化**
Null	**无有效数据**
<object type>	**实际对象类型名**
Object	**一般对象**
Unknown	**未知对象类型**
Nothing	**还未引用对象实例的对象变量**
Error	**错误**

下面的例子利用 TypeName **函数返回变量的子类型信息：**

```
Dim ArrayVar(4), DataTypeVar
```

```
NullVar = Null                              ' 赋 Null 值
DataTypeVar = TypeName("newland")           ' 返回 "String"
DataTypeVar = TypeName(20)                  ' 返回 "Integer"
DataTypeVar = TypeName(56.63)               ' 返回 "Double"
DataTypeVar = TypeName(NullVar)             ' 返回 "Null"
DataTypeVar = TypeName(ArrayVar)            ' 返回 "Variant()"
```

5.6.8　VarType 函数

返回指示变量子类型的值。其语法为：

```
VarType(varname)
```

varname **参数可以是任何变量。**
该函数的返回值，参见表 5.25 **所示。**

表 5.25　VarType **函数的返回值**

常数	值	描述
vbEmpty	0	Empty（未初始化）
vbNull	1	Null（无有效数据）
vbInteger	2	整数
vbLong	3	长整数
vbSingle	4	单精度浮点数
vbDouble	5	双精度浮点数
vbCurrency	6	货币
vbDate	7	日期
vbString	8	字符串
vbObject	9	Automation 对象
vbError	10	错误
vbBoolean	11	Boolean
vbVariant	12	Variant（只和变量数组一起使用）
vbDataObject	13	数据访问对象
vbByte	17	字节
vbArray	8192	数组

注意：这些常数是由 VBScript **指定的。所以，这些名称可在代码中随处使用，以代替实际值。**

VarType **函数从不通过自己返回** Array **的值。它总是要添加一些其他值来指示一个具体类型的数组。当** Variant **的值被添加到** Array **的值中以表明** VarType **函数的参数是一个数组时，它才被返回。例如，对一个整数数组的返回值是** 2 + 8192 **的计算结果，或** 8194**。如果**

一个对象有默认，则 VarType(object) 返回对象默认属性的类型。

下面函数利用 VarType 函数决定变量的子类型：

```
Dim CheckVar
CheckVar = VarType(100)                    ’返回 2
CheckVar = VarType(#04/28/2001#)           ’返回 7
CheckVar = VarType("newland")              ’返回 8
```

5.7 其他常用函数

除了上述的 6 种函数之外，VBScript 还提供了许多其他不大容易分类但却很重要且很常用的函数，如消息框函数、颜色函数等，我们在这里列出做一说明。

5.7.1 Msgbox 函数

可以这么说，在 VBScript 中，Msgbox 函数是最常用最有用的函数。它的功能是在对话框中显示消息，等待用户单击按钮，并返回一个值指示用户单击的按钮。而实际上我们经常利用这个函数进行调试。它的语法如下：

```
MsgBox(prompt[, buttons][, title][, helpfile, context])
```

参数说明：

- prompt：作为消息显示在对话框中的字符串表达式。prompt 的最大长度大约是 1024 个字符，这取决于所使用的字符的宽度。如果 prompt 中包含多个行，则可在各行之间用回车符 (Chr(13))、换行符 (Chr(10)) 或回车换行符的组合 (Chr(13) & Chr(10)) 分隔各行。
- buttons：数值表达式，表示指定显示按钮的数目和类型、使用的图标样式，默认按钮的标识以及消息框样式的数值的总和。有关数值，请参阅表 5.26。如果省略，则 buttons 的默认值为 0。
- title：显示在对话框标题栏中的字符串表达式。如果省略 title，则将应用程序的名称显示在标题栏中。
- helpfile：字符串表达式，用于标识为对话框提供上下文相关帮助的帮助文件。如果已提供 helpfile，则必须提供 context。在 16 位系统平台上不可用。
- context：数值表达式，用于标识由帮助文件的作者指定给某个帮助主题的上下文编号。如果已提供 context，则必须提供 helpfile。在 16 位系统平台上不可用。

表 5.26 Msgbox 函数的 buttons 参数值

常数	值	描述
vbOKOnly	0	只显示确定按钮
vbOKCancel	1	显示确定和取消按钮
vbAbortRetryIgnore	2	显示放弃、重试和忽略按钮

（续表）

常数	值	描述
vbYesNoCancel	3	显示是、否和取消按钮
vbYesNo	4	显示是和否按钮
vbRetryCancel	5	显示重试和取消按钮
vbCritical	16	显示临界信息图标
vbQuestion	32	显示警告查询图标
vbExclamation	48	显示警告消息图标
vbInformation	64	显示信息消息图标
vbDefaultButton1	0	第一个按钮为默认按钮
vbDefaultButton2	256	第二个按钮为默认按钮
vbDefaultButton3	512	第三个按钮为默认按钮
vbDefaultButton4	768	第四个按钮为默认按钮
vbApplicationModal	0	应用程序模式：用户必须响应消息框才能继续在当前应用程序中工作
vbSystemModal	4096	系统模式：在用户响应消息框前，所有应用程序都被挂起

第一组值（0~5）用于描述对话框中显示的按钮类型与数目；第二组值（16, 32, 48, 64）用于描述图标的样式；第三组值（0, 256, 512）用于确定默认按钮；而第四组值（0, 4096）则决定消息框的样式。在将这些数字相加以生成 buttons 参数值时，只能从每组值中取用一个数字。

该函数的返回值，参见表 5.27 所示。

表 5.27　MsgBox 函数的返回值

常数	值	按钮
vbOK	1	确定
vbCancel	2	取消
vbAbort	3	放弃
vbRetry	4	重试
vbIgnore	5	忽略
vbYes	6	是
vbNo	7	否

如果同时提供了 helpfile 和 context，则用户可以按 F1 键以查看与上下文相对应的帮助主题。

如果对话框显示取消按钮，则按 Esc 键与单击取消的效果相同。如果对话框包含帮助按钮，则有为对话框提供的上下文相关帮助。但是在单击其他按钮之前，不会返回任何值。

当 MicroSoft Internet Explorer 使用 MsgBox 函数时，任何对话框的标题总是包含"VBScript"，以便于将其与标准对话框区别开来。

下面的例子演示了 MsgBox 函数的用法：

```
Dim MsgVar
MsgVar = MsgBox ("新概念 VBScript 教程!", 257, "新概念系列丛书")
```

如果用户选择“确定”，MsgVar 得到 1；如果用户选择“取消”，MsgVar 得到 2。

示例运行的结果如图 5.1 所示。

图 5.1 Msgbox 函数示例

5.7.2 InputBox 函数

该函数在屏幕上显示一对话框，等待用户输入文本或单击按钮，并返回文本框内容。语法如下：

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])
```

参数说明：

- prompt：字符串表达式，作为消息显示在对话框中。prompt 的最大长度大约是 1024 个字符，这取决于所使用的字符的宽度。如果 prompt 中包含多个行，则可在各行之间用回车符 (Chr(13))、换行符 (Chr(10)) 或回车换行符的组合 (Chr(13) & Chr(10)) 以分隔各行。
- title：显示在对话框标题栏中的字符串表达式。如果省略 title，则应用程序的名称将显示在标题栏中。
- default：显示在文本框中的字符串表达式，在没有其它输入时作为默认的响应值。如果省略 default，则文本框为空。
- xpos：数值表达式，用于指定对话框的左边缘与屏幕左边缘的水平距离（单位为缇）。如果省略 xpos，则对话框会在水平方向居中。
- ypos：数值表达式，用于指定对话框的上边缘与屏幕上边缘的垂直距离（单位为缇）。如果省略 ypos，则对话框显示在屏幕垂直方向距下边缘大约三分之一处。
- helpfile：字符串表达式，用于标识为对话框提供上下文相关帮助的帮助文件。如果已提供 helpfile，则必须提供 context。
- context：数值表达式，用于标识由帮助文件的作者指定给某个帮助主题的上下文编号。如果已提供 context，则必须提供 helpfile。

如果同时提供了 helpfile 和 context，就会在对话框中自动添加“帮助”按钮。

如果用户单击确定或按下 Enter，则 InputBox 函数返回文本框中的内容。如果用户单击取消，则函数返回一个零长度字符串（""）。

下面例子利用 InputBox 函数显示一输入框并且把字符串赋值给输入变量，然后用 Msgbox 函数将输入的结果显示给用户。

```
Dim InputVar
InputVar = InputBox("请输入您的姓名：")
MsgBox (“嗨！”& InputVar & “你好！”)
```

运行结果如图 5.2 和图 5.3 所示。

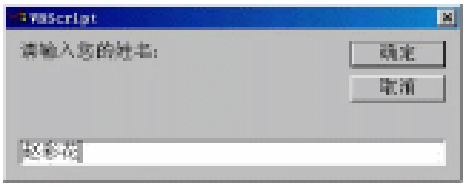

图 5.2　Input 函数示例

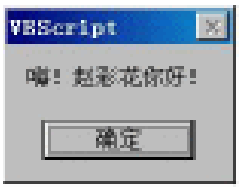

图 5.3　使用 Msgbox 函数输出

5.7.3　Rgb 函数

返回代表 RGB 颜色值的整数。语法如下：

```
RGB(red, green, blue)
```

参数说明：

- red：必选项，0～255 间的整数，代表颜色中的红色成分。
- green：必选项，0～255 间的整数，代表颜色中的绿色成分。
- blue：必选项，0～255 间的整数，代表颜色中的蓝色成分。

RGB 颜色值指定了红色、绿色、蓝色的相对强度，三色组合形成显示的特定颜色。低字节值表示红色，中字节值表示绿色，高字节值表示蓝色。

RGB 函数中任一超过 255 的参数都假定为 255。

5.7.4　Split 函数

返回基于 0 的一维数组，其中包含指定数目的子字符串。其语法为：

```
Split(expression[, delimiter[, count[, start]]])
```

参数说明：

- expression：必选项，字符串表达式，包含子字符串和分隔符。如果 expression 为零长度字符串，Split 返回空数组，即不包含元素和数据的数组。
- delimiter：可选项，用于标识子字符串界限的字符。如果省略，使用空格 ("") 作为分隔符。如果 delimiter 为零长度字符串，则返回包含整个 expression 字符串

的单元素数组。

- count：可选项，被返回的子字符串数目，－1 指示返回所有子字符串。
- compare：可选项，指示在计算子字符串时使用的比较类型的数值。有关数值，请参阅表 5.28。

表 5.28 Split 函数的 compare 参数值

常数	值	描述
vbBinaryCompare	0	执行二进制比较
vbTextCompare	1	执行文本比较

下面的例子利用 Split 函数从字符串中返回数组。函数对分界符进行文本比较，返回所有的子字符串。

```
Dim StringVar, ArrayVar, MsgVar
StringVar = "VBScript,JavaScript,ASP"
ArrayVar = Split(StringVar, ",", －1, 1)
’ArrayVar (0) 包含 "VBScript"
’ArrayVar (1) 包含 "JavaScript"
’ArrayVar (2) 包含 "ASP"
for I = 0 to 2
    Msgbox ArrayVar(i)
next
```

5.7.5 UBound 与 LBound 函数

1．Ubound 函数

返回指定数组维数的上界。其语法为：

```
UBound(arrayname[, dimension])
```

参数说明：

- arrayname：必选项，数组变量名，遵循标准变量命名约定。
- dimension：可选项，指定返回哪一维上界的整数。1 表示第一维，2 表示第二维，以此类推。如果省略 dimension 参数，则默认值为 1。

2．LBound 函数

返回指定数组维的下界。其语法为：

```
LBound(arrayname[, dimension])
```

参数说明：

- arrayname：数组变量名，遵循标准变量命名约定。
- dimension：指明要返回哪一维下界的整数。使用 1 表示第一维，2 表示第二维，

以此类推。如果省略 dimension 参数，默认值为 1。

任一维的下界都是 0。

UBound 函数与 LBound 函数一起使用，用于确定数组的大小。使用 LBound 函数可以确定数组某一维的下界。

下面的例子用 Ubound 函数取得数组每一维的上界：

```
Dim ArrayVar(100,5,6)
Dim UboundVar
UboundVar = UBound(ArrayVar, 1)          ' 返回 100
UboundVar = UBound(ArrayVar, 2)          ' 返回 5
UboundVar = UBound(ArrayVar, 3)          ' 返回 6
```

5.8　小　结

本章中，我们对 VBScript 中的内置函数作了详细的介绍，为了便于掌握，我们将 VBScript 中的内置函数分成 5 类，它们分别是：字符串函数、转换函数、数字函数、日期和时间函数、布尔函数和其他常用函数。

学会使用这些函数，对编程将会有很大的帮助。

练习题

1．VBScript 的内部函数分为哪几种？分别是什么？

2．将字符串“VBScript”转换为“V B S c r i p t”，即在原来的字符串中每隔一个字符加入一个空格。

3．将当前日期变为“xxxx 年 xx 月 xx 日”的格式。

4．描述 Int 函数、Fix 函数和 Round 函数的相同和不同之处。

5．计算今天是星期几，距离 2008 年 01 月 01 日还有多长时间？

6．计算字符串“We are good friends,aren't we?”包含了几个空格？

第 6 章　VBScript 中的对象和集合

由于 VBScript 不强调类和对象等面向对象的概念，因此在使用 VBScript 编程的过程中，我们很少体会到面向对象的编程方式。这样也有一个好处：进一步简化语言的学习和使用，也便于程序的编制和维护，使得程序员没有必要纠缠在难以理解的概念中。在 VBScript 中还是保留了对象这个概念，但是 VBScript 中的对象与 C++中的对象中的概念有所区别。在 VBScript 中，对象的概念只是一个具有一定属性和方法的集合。由于其所封装的属性和方法在功能上具有一定的独立性，所以可以很方便地将其实例化，然后使用它所提供的属性和方法，来完成各类功能。

VBScript 对其大部分对象都提供了集合支持，利用集合操作对象及其他相关元素，可以大大提高开发效率。

本章我们就对 VBScript 中的对象和集合作一介绍，并通过一些实例，重点讲解 VBScript 对象的使用方法。

6.1　VBScript 中的对象

VBScript 的对象是由一系列的属性（Property）和方法（Method）构成的。属性是对象的内嵌变量，方法是对象的内嵌函数。

VBScript 支持 3 种类型的对象：内置对象、浏览器对象和用户自定义对象。本章主要讲述内置对象。VBScript 的内置对象共有 7 种：

- Dictionary 对象
- Drive 对象
- Err 对象
- File 对象
- FileSystemObject 对象
- Folder 对象
- TextStream 对象

同时 VBScript 还支持浏览器对象，如 Window 对象和 History 对象等，主要用于客户端，VBScript 使用这些浏览器对象所提供的与浏览器的接口，实现了客户端的很多功能。对于浏览器对象我们将在第 8 章中详细描述。

VBScript 还允许用户自定义对象，这个功能为我们提供了更广阔的对象处理能力，自定义对象同样也有其属性和方法，并可在程序中像内置对象一样使用。不过在多数情况下，我们用不着亲自创建自己的对象，因为 VBScript 为我们提供了丰富的对象。

在服务器端创建对象的实例，一般使用 ASP 的 Server.CreateObject 方法，并使用语言的变量分配指令为对象实例命名，以后，可以通过这个变量名来访问该对象的实例，并且使用该对象提供的各种方法和属性。创建对象实例时，必须提供实例的注册名称（PROGID）。

使用 HTML 中的<OBJECT>标记同样可以创建对象实例。不过用户必须为 RUNAT 属性提供服务器值，同时也要为将在脚本语言中使用的变量名提供 ID 属性组。使用注册名称（PROGID）或注册号码（CLSID）可以识别该对象。下面的例子使用两种不同的方式创建对象。

如果我们想要实例化 IIS 所附带的 Browser Type 组件，则 PROGID 为 MSWC.BrowserType。使用如下语句创建一个对象实例：

使用 Server.CreateObject 创建对象：

```
Set browser = Server.CreatObject("MSWC.BrowserType")
```

注意：必须使用 Set 语句，因为对象的处理方式与通常的变量不同。

使用<OBJECT>标记创建对象：

```
<OBJECT RUNAT=Server ID=MyBt PROGID=" NSWC.BrowserType">
</OBJECT>
```

在客户端创建对象的实例，直接使用 VBScript 的 CreateObject 函数就可以了。

此外，在使用对象的时候，还可以使用对象的方法。方法是可以在对象上所执行或使用对象执行的活动。通过使用方法，可以方便地完成一系列任务。调用对象方法的语法为：

```
object.Method parameterslist
```

其中 parameters 作为参数随着用户所使用的对象方法不同而不同。

下面，我们将依次介绍 VBScript 中的对象。

6.1.1　Dictionary 对象

Dictionary 对象是保存数据键和项目对的对象，类似二元数组，把关键字和关联的项组合在一起。Dictionary 对象与 PERL 关联数组是等价的。项目（可以是任何形式的数据）被保存在数组中。每项都与唯一的键相关联。键值用于检索单个项目，通常是整数或字符串，但不能为数组。该对象的语法为：

```
Scripting.Dictionary
```

下面代码示范如何创建 Dictionary 对象：

```
Dim d                                          ’创建一个变量
Set d = CreateObject("Scripting.Dictionary")
d.Add "0", "星期日"                            ’添加键和项目
d.Add "1", "星期一"
d.Add "2", "星期二"
```

1．Dictionary 对象的属性

CompareMode 属性

该属性设置并返回在 Dictionary 对象中比较字符串关键字的比较模式。语法为：

```
object.CompareMode[ = compare]
```

Count 属性

该属性为只读属性。返回一个集合或 Dictionary 对象包含的项目数。语法为：

```
Object.Count
```

Key 属性

该属性在 Dictionary 对象中设置 key。语法为：

```
Object.Key(kdy) = newkey
```

其中 object 为 Dictionary 对象的名称。

参数 Key 必选，为要改变的 Key 值。

参数 Newkey 必选，表示代替指定 key 值的新值。

如果在更改 key 值时未找到 key，将出现运行时错误。下面例子说明如何使用 Key 属性：

```
Dim d                                    ’创建变量
Set d = CreateObject("Scripting.Dictionary")
d.Add "0", "星期日"                      ’添加键和项目对
d.Add "1", "星期一"
d.Add "2", "星期二"
d.Key("0") = "Sunday"                    ’将"0"的键设置为"Sunday"
```

Item 属性

该属性可读写，用来设置或返回 Dictionary 对象中指定的 key 对应的 item，或返回集合中基于指定的 key 的 item。语法为：

```
object.Item(key) [= newitem]
```

其中 object 为 Dictionary 对象的名称。

参数 key 必选，表示与检索或添加的 item 相关联的 key。

参数 newitem 是可选项，仅用于 Dictionary 对象，不能应用于集合。

如果提供此参数，则 newitem 是与指定的 key 相关联的项目的新值。

如果更改 item 时未找到 key，则使用指定的 newitem 创建一个新的 key。如果试图返回一个已有项目时未找到 key，则创建一个新的 key 并且它对应的项目为空值。

下面例子说明如何使用 Item 属性：

```
Dim d                                    ’创建变量
Set d = CreateObject("Scripting.Dictionary")
```

```
d.Add "0", "星期日"                    ’添加键和项目对
d.Add "1", "星期一"
d.Add "2", "星期二"
d.Key("0") = "Sunday"                  ’将"0"的键设置为"Sunday"
Item1 = d.Item("0")                    ’返回相关项目"Sunday"
```

2．Dictionary 对象的方法

Add 方法

向 Dictionary 对象添加键和项目对。语法为：

object.Add key, item

其中 object 为 Dictionary 对象的名称。

参数 key 必选，为与添加的 item 相关的 key。

参数 item 必选，为与添加的 key 相关的 item。

如果 key 已经存在，则会出现错误。

Exists 方法

如果在 Dictionary 对象中存在指定键，返回 True；如果不存在，返回 False。语法为：

```
object.Exists(key)
```

其中 object 为 Dictionary 对象名称。

参数 key 必选，为在 Dictionary 对象中查找的 Key 值。

下面例子说明如何使用 Exists 方法：

```
Function FindKey
    Dim d, msg                         ’创建一些变量
    Set d = CreateObject("Scripting.Dictionary")
    d.Add "0", "星期日"                ’添加一些键和项目
    d.Add "1", "星期一"
    d.Add "2", "星期二"
    If d.Exists("2") Then
       FindKey = True
    Else
FindKey = False
  End If
    End Function
```

Keys 方法

返回一数组，其中包含有 Dictionary 对象的所有现存键。语法为：

object.Keys

其中 object 为 Dictionary 对象的名称。

下列例子说明如何使用 Keys 方法：

```
Function DicKeys
```

```
Dim d,i ,allkey,strkeys                          ’创建一些变量
Set d = CreateObject("Scripting.Dictionary")
d.Add "0", "星期日"                              ’添加一些键和项目
    d.Add "1", "星期一"
d.Add "2", "星期二"
allkey = d.Keys                                  ’获取键
For i = 0 To d.Count -1                          ’循环使用数组
strkeys = strkeys & allkey(i) & "<BR>"           ’返回结果
Next
DicKeys = strkeys
End Function
```

Items 方法

返回一个数组，其中包含有 Dictionary **对象中的所有项目。语法为：**

```
object.Items
```

其中 object **为** Dictionary **对象的名称。**

下面例子说明如何使用 Items **方法：**

```
Function DicItems
Dim d,I,allitem,stritems                         ’创建一些变量
Set d = CreateObject("Scripting.Dictionary")
d.Add "0", "星期日"                              ’添加一些键和项目
    d.Add "1", "星期一"
d.Add "2", "星期二"
allitem = d.Items                                ’获取项目
For i = 0 To d.Count -1                          ’循环使用数组
stritems = stritems & allitems(i) & "<BR>"       ’返回结果
Next
DicItems = stritems
End Function
```

Remove 方法

从 Dictionary **对象中删除键和项目对。语法为：**

```
object.Remove(key)
```

其中 object **为** Dictionary **对象的名称。**

参数 key **必选，为要从** Dictionary **对象中删除的键和项目对相关联的** Key**。**

如果指定的键和项目对不存在，则会出现错误。

下面例子说明如何使用 Remove **方法：**

```
Dim d                                            ’创建变量
    Set d = CreateObject("Scripting.Dictionary")
    d.Add "0", "星期日"                          ’添加一些键和项目
```

```
    d.Add "1", "星期一"
    d.Add "2", "星期二"
    d.Remove("0")                                    ’删除第一个项目对
```

RemoveAll 方法

删除 Dictionary 对象中的所有键和项目对。语法为：

```
object.RemoveAll
```

其中 object 为 Dictionary 对象的名称。

下面例子说明如何使用 RemoveAll 方法：

```
Dim d                                                ’创建一些变量
    Set d = CreateObject("Scripting.Dictionary")
    d.Add "0", "星期日"                              ’添加一些键和项目
    d.Add "1", "星期一"
    d.Add "2", "星期二"
    d.RemoveAll                                      ’清除 dictionary。
```

3．程序实例

下面将通过一个实例加深对对象的认识，在这个例子中，我们将学会如何实例化一个对象，并使用它的方法和属性。

步骤一：制作页面

这个页面用到了 3 种控件：输入框 itemAdd、列表框 listBox 和一个按钮 cmdAdd。

做好的页面如图 6.1 所示。

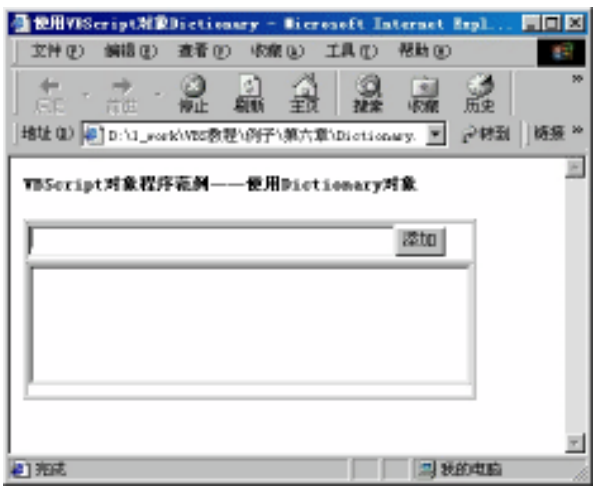

图 6.1　使用 VBScript 对象 Dictionary——页面布局

这个例子要实现的功能是：用户在单行文本输入框中输入信息并单击“添加”按钮时，在其下面的列表框中添加这条信息。并将输入焦点仍然置于输入框。如果输入框中没有添入信息，进行提示。

步骤二：添加代码

既然我们知道要实现的功能，下面就应该动手添加代码了。将通过 VBScript **的对象** Dictionary **来帮助我们完成这个功能。实现起来可以分为** 3 **个部分：**

（1**）在页面被装载时，创建一个** Dictionary **对象的实例** d**，并且将计数器** i **置为** 0**，其中** i **是这个对象的键值。这些代码写在** Window_OnLoad **中。**

```
Sub Window_OnLoad()
    I = 0
    Set d = CreateObject("Scripting.Dictionary")
End Sub
```

（2**）当用户单击了“ 添加 ”按钮时，如果没有输入任何信息，进行提示；如果输入不为空，需要将输入框内的信息** itemAdd.value **添加到列表框中，并且将输入焦点置于输入框中。我们将这个实现的过程编写到一个过程** AddItem **中。**

```
Sub AddItem(item)
  If item = "" then
      alert "不能添加空值！"
  Else
      d.add i,item
      Set e = document.createElement("OPTION")
      e.text = d.Item(i)
      e.value = i
      listBox.Options.Add e
      itemAdd.value = ""
      i = i + 1
      itemAdd.select ()
  End if
End Sub
```

其中参数 item **表示要添加的值。**

（3**）在单击“ 添加 ”按钮时，调用** AddItem **过程。**

```
    Sub cmdAdd_OnClick
        Call AddItem(itemAdd.value)
End Sub
```

运行的结果如图 6.2 **所示。**

6.1.2 FileSystemObject 对象

该对象提供对计算机文件系统的访问。允许我们在代码中操纵文本文件、文件夹和驱动器。其语法为：

```
Scripting.FileSystemObject
```

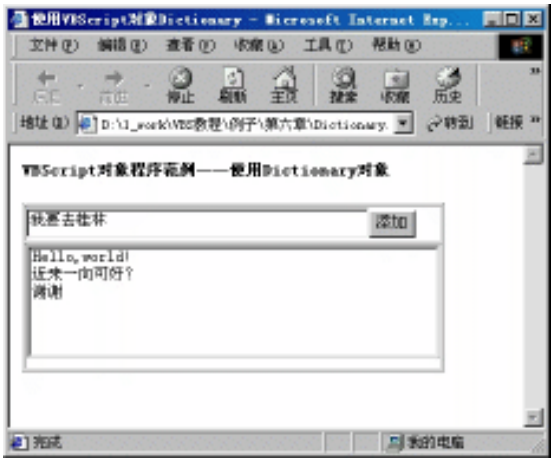

图 6.2　使用 VBScript 对象 Dictionary——运行效果

例如，使用以下语句实例化一个 FileSystemObject 对象

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

1．FileSystemObject 对象的属性

该对象只有一个属性——Drives 属性，返回由本地机器上所有 Drive 对象组成的 Drives 集合。语法为：

```
object.Drives
```

其中 object 应为 FileSystemObject 对象的名称。

无论是否插入媒体，可移动媒体驱动器都显示在 Drives 集合中。有关 Drives 集合，我们将在本章稍后的 VBScript 中的集合中详细讲述。

可以使用 For Each...Next 结构枚举 Drives 集合的成员，如例【例 6.1】所示。

【例 6.1】　使用 FileSystemObject 对象列出所有驱动器

```
<HTML>
<HEAD>
<TITLE>列出所有驱动器</TITLE>
<META http-equiv="Content-Type" content="text/html; charset=gb2312">
</HEAD>
<BODY>
列出所有驱动器
<TABLE border=1>
<SCRIPT Language="VBScript">
<!--
Sub ShowDriveList
    Dim fs,d,dc,s,n
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set dc = fs.Drives
```

```
    For Each d in dc
        document.write ("<TR><TD width=20>" & d.DriveLetter & "</TD><TD>")
        If d.DriveType = 3 then
            s = "Remove"
            n = d.ShareName
        Else
            s = "Local"
            n = d.VolumeName
        End if
        document.write ( s & "</TD><TD>" & n & "</TD></TR>")
    Next
End Sub
Call ShowDriveList
-->
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

该例的运行结果如图 6.3 **所示。**

图 6.3 **使用** FileSystemObject **对象列出所有驱动器**

2．FileSystemObject 对象的方法

该对象有许多方法，分述如下：

BuildPath 方法

向现有路径后添加名称。语法为：

```
object.BuildPath(path, name)
```

参数说明：

- object：**必选项，为** FileSystemObject **对象的名称。**

- path：必选项，要附加 name 的现有路径，可以是绝对或相对路径且无需指定现有文件夹。
- name：必选项，要附加到现有 path 的名称。

仅在必要时，BuildPath 方法在现有路径与该名称之间插入附加路径分隔符。
下面例子说明如何使用 BuildPath 方法：

```
Function GetBuildPath(path)
  Dim fso, newpath
  Set fso = CreateObject("Scripting.FileSystemObject")
  newpath = fso.BuildPath(path, "Sub Folder")
  GetBuildPath = newpath
End Function
```

CopyFile 方法
将一个或多个文件从某位置复制到另一位置。语法为：

```
object.CopyFile source, destination[, overwrite]
```

参数说明：

- object：必选项，应为 FileSystemObject 对象的名称。
- source：必选项，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：必选项，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：可选项，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。要注意，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。

仅能在 source 参数的路径最后一个组成部分中使用通配符。例如，可以使用：

```
FileSystemObject.CopyFile "c:\mydocuments\letters\*.doc", "c:\tempfolder\"
```

但是，不能使用：FileSystemObject.CopyFile"c:\mydocuments*\R1???97.xls"，"c:\tempfolder"。
如果 source 包含通配符或 destination 以路径分隔符（\）结束，则假定 destination 是现有文件夹，复制匹配文件到该文件夹。否则，假定 destination 为要创建的文件。在任一种情况下，复制单个文件时，会出现以下 3 种情况：

- 如果 destination 不存在，则复制 source。这是通常会发生的情况。
- 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。
- 如果 destination 是目录，则会出现错误。

如果 source 使用通配符，但并没有相匹配的文件时，则会出现错误。CopyFile 方法

在遇到出现的第一个错误时停止。该方法不会撤消错误发生前所作的任何更改。

CopyFolder 方法

将文件夹从某位置递归复制到另一位置。语法为：

```
object.CopyFolder source, destination[, overwrite]
```

参数说明：

- object：必选项，应为 FileSystemObject 对象的名称。
- source：必选项，表示指定文件夹的字符串。要复制一个或多个文件夹时，可使用通配符。
- destination：必选项，表示目标位置的字符串，复制 source 文件夹或子文件夹到该位置。不允许用通配符。
- overwrite：可选项，Boolean 值表明是否覆盖现有文件夹。如果为 True，则覆盖文件；如果为 False，则不覆盖文件。默认值是 True。

仅能在 source 参数的路径最后一个组成部分中使用通配符。例如，可以使用：

```
FileSystemObject.CopyFolder "c:\mydocuments\letters\*", "c:\tempfolder\"
```

但是，不可以使用：FileSystemObject.CopyFolder "c:\mydocuments**", "c:\tempfolder\"。

如果 source 包含通配符或 destination 以路径分隔符（\）结束，则假定 destination 是现有文件夹，在该文件夹中复制匹配文件夹或子文件夹。否则，假定 destination 是要创建的文件夹。在任一种情况下，复制单个文件夹时，会发生如下 4 种情况：

- 如果 destination 不存在，则复制 source 文件夹和其所有内容。这是通常会发生的情况。
- 如果 destination 是已经存在的文件，则出现错误。
- 如果 destination 是目录，则复制文件夹和其中的所有内容。如果在 destination 已经存在 source 包含的文件，且 overwrite 为 False，则会出现错误。否则，复制该文件覆盖现有文件。
- 如果 destination 是只读目录，在向该目录复制现有只读文件，且 overwrite 为 False 时，就会出现错误。

如果 source 使用通配符，但没有匹配文件时，也会出现错误。

CopyFolder 方法在遇到出现的第一个错误时停止。该方法不会撤消错误发生前所作的任何更改。

CreateFolder 方法

创建文件夹。语法为：

```
object.CreateFolder(foldername)
```

参数说明：

- object：必选项，应为 FileSystemObject 对象的名称。
- foldername：必选项，字符串表达式，指明要创建的文件夹。

如果指定的文件夹已经存在，则会出现错误。
下面例子说明如何使用 CreateFolder 方法：

```
Function CreateFolderDemo
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.CreateFolder("c:\NewFolder")
   CreateFolderDemo = f.Path
End Function
```

CreateTextFile 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。语法为：

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

参数说明：

- object：必选项，应为 FileSystemObject 或 Folder 对象的名称。
- filename：必选项，字符串表达式，指明要创建的文件。
- overwrite：可选项，Boolean 值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：可选项，Boolean 值指明是否以 Unicode 或 ASCII 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 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 文件。

以下例子说明如何使用 CreateTextFile 方法创建并打开文本文件：

```
Sub CreateAfile
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f = fso.CreateTextFile("c:\testfile.txt", True)
   f.WriteLine("This is a test.”)
   f.Close
End Sub
```

对于 filename 已经存在的文件，如果 overwrite 参数为 False，或未提供此参数时，则会出现错误。

DeleteFile 方法

删除指定的文件。语法为：

```
object.DeleteFile filespec[, force]
```

参数说明：

- object：**必选项，应为** FileSystemObject **的名称。**
- filespec：**必选项，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：**可选项，**Boolean **值。如果要删除只读文件，则该值为** True**；否则为** False **（默认）。**

如果没有找到匹配文件，则会出现错误。DeleteFile **方法在遇到出现的第一个错误时停止。该方法不会撤消错误发生前所作的任何更改。**

下面例子说明如何使用 DeleteFile **方法：**

```
Sub DeleteAFile(filespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.DeleteFile(filespec)
End Sub
```

DeleteFolder 方法

删除指定的文件夹和其中的内容。语法为：

```
object.DeleteFolder folderspec[, force]
```

参数说明：

- object：**必选项，应为** FileSystemObject **的名称。**
- folderspec：**必选项，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：**可选项，**Boolean **值。如果要删除只读文件夹，则该值为** True**；否则为** False **（默认）。**

DeleteFolder **方法不能区分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一个错误时停止。该方法不会撤消错误发生前所作的任何更改。**

下面例子说明如何使用 DeleteFolder **方法：**

```
Sub DeleteAFolder(filespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.DeleteFolder(filespec)
End Sub
```

DriveExists 方法

如果指定的驱动器存在，则返回 True**；否则返回** False**。语法为：**

```
object.DriveExists(drivespec)
```

参数说明：

- object：**必选项，应为** FileSystemObject **对象的名称。**
- drivespec：**必选项，驱动器号或指定的完整路径。**

对于可移动媒体驱动器，即使驱动器中没有插入媒体，DriveExists **方法仍返回** True**。用** Drive **对象的** IsReady **属性确定驱动器是否就绪。**

下面例子说明如何使用 DriveExists **方法：**

```
Function ReportDriveStatus(drv)
  Dim fso, msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  If fso.DriveExists(drv) Then
    msg = ("驱动器 " & UCase(drv) & " 存在。")
  Else
    msg = ("驱动器 " & UCase(drv) & " 不存在。")
  End If
  ReportDriveStatus = msg
End Function
```

FileExists **方法**

如果指定的文件存在返回 True**；否则返回** False**。语法为：**

```
object.FileExists(filespec)
```

参数说明：

- object：**必选项，应为** FileSystemObject **对象的名称。**
- filespec：**必选项，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，则必须提供完整路径名（绝对路径或相对路径）。**

下面例子说明如何使用 FileExists **方法：**

```
Function ReportFileStatus(filespec)
  Dim fso, msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  If (fso.FileExists(filespec)) Then
    msg = filespec & " 存在。"
  Else
    msg = filespec & " 不存在。"
  End If
  ReportFileStatus = msg
End Function
```

FolderExists 方法

如果指定的文件夹存在，则返回 True**；否则返回** False**。语法为：**

```
object.FolderExists(folderspec)
```

参数说明：

- object**：必选项，应为** FileSystemObject **的名称。**
- folderspec**：必选项，文件夹名称，表示要确定是否存在的文件夹。如果该文件夹不在当前文件夹中，则必须提供完整路径名（绝对路径或相对路径）。**

下面例子说明如何使用 FolderExists **方法：**

```
Function ReportFolderStatus(fldr)
  Dim fso, msg
  Set fso = CreateObject("Scripting.FileSystemObject")
  If (fso.FolderExists(fldr)) Then
    msg = fldr & " 存在。"
  Else
    msg = fldr & " 不存在。"
  End If
  ReportFolderStatus = msg
End Function
```

GetAbsolutePathName 方法

从提供的指定路径中返回完整且含义明确的路径。语法为：

```
object.GetAbsolutePathName(pathspec)
```

参数说明：

- object**：必选项，应为** FileSystemObject **的名称。**
- pathspec**：必选项，指定的路径，该路径要转换为完整且含义明确的路径。**

如果路径提供从指定驱动器根目录开始的完整引用，则该路径是完整且含义明确的。如果路径指定某映射驱动器的根目录文件夹，则该路径仅能以路径分隔符（\）结束。

假定当前目录为 c:\mydocuments\reports**，表** 6.1 **说明了** GetAbsolutePathName **方法执行的操作。**

表 6.1　GetAbsdutePathName **方法执行时的操作**

pathspec	返回路径
"c:"	"c:\mydocuments\reports"
"c:.."	"c:\mydocuments"
"c:\\\"	"c:\"
"c:*.*\may97"	"c:\mydocuments\reports*.*\may97"

（续表）

pathspec	返回路径
"region1"	"c:\mydocuments\reports\region1"
"c:\..\..\mydocuments"	"c:\mydocuments"

GetBaseName 方法

返回字符串，其中包含文件的基本名（不带扩展名），或者提供的路径说明中的文件夹。语法为：

```
object.GetBaseName(path)
```

参数说明：

- object：必选项，应为 FileSystemObject 的名称。
- path：必选项，指定文件或文件夹的路径，要返回其组成部分的基本名。

如果路径中没有任何文件或文件夹与指定的 path 参数匹配，则 GetBaseName 方法返回零长度字符串（""）。

下面例子举例说明如何使用 GetBaseName 方法：

```
Function GetTheBase(filespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetTheBase = fso.GetBaseName(filespec)
End Function
```

注意：GetBaseName 方法只能对提供 path 的字符串起作用。它不能试图分析一个路径，也不能检查指定路径是否存在。

GetDrive 方法

返回与指定路径中驱动器相对应的 Drive 对象。语法为：

```
object.GetDrive drivespec
```

参数说明：

- object：必选项，应为 FileSystemObject 的名称。
- drivespec：必选项，drivespec 可以是驱动器号（c）、带冒号的驱动器号（c:）、带有冒号与路径分隔符的驱动器号（c:\）或任何指定的网络共享（\\computer2\share1）。

对于网络共享，检查并确保该网络共享存在。

若 drivespec 与已接受格式不一致或不存在，就会出错。为了在调用 GetDrive 方法时使用标准路径字符串，使用下列序列得到与 drivespec 相匹配的字符串：

```
DriveSpec = GetDriveName(GetAbsolutePathName(Path))
```

下面示例说明如何使用 GetDrive **方法：**

```
Function ShowFreeSpace(drvPath)
   Dim fso, d, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set d = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & d.VolumeName    & "<BR>"
   s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
   s = s & " Kbytes"
   ShowFreeSpace = s
End Function
```

GetDriveName **方法**

返回包含指定路径中驱动器名的字符串。语法为：

```
object.GetDriveName(path)
```

参数说明：

- object**：必选项，应为** FileSystemObject **的名称。**
- path**：必选项，指定路径，要返回其组成部分的驱动器名。**

如果无法确定驱动器，则 GetDriveName **方法返回零长度字符串** ("")。
下面例子说明如何使用 GetDriveName **方法：**

```
Function GetAName(DriveSpec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   GetAName = fso.GetDriveName(Drivespec)
End Function
```

注意：GetDriveName **方法只能对提供** path **的字符串起作用，它不能试图分析一个路径，也不能检查指定路径是否存在。**

GetExtensionName **方法**

返回字符串，该字符串包含路径最后一个组成部分的扩展名。语法为：

```
object.GetExtensionName(path)
```

参数说明：

- object**：必选项，应为** FileSystemObject **的名称。**
- path**：必选项，指定路径，用于返回其组成部分的扩展名。**

对于网络驱动器，根目录（\）也认为是一个组成部分。
如果路径组成部分与 path **参数不匹配，则** GetExtensionName **方法返回零长度字符串**

("")。

下面例子说明如何使用 GetExtensionName 方法：

```
Function GetAnExtension(DriveSpec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetAnExtension = fso.GetExtensionName(Drivespec)
End Function
```

GetFile 方法

返回与指定路径中某文件相应的 File 对象。语法为：

```
object.GetFile(filespec)
```

参数说明：

- object：必选项，应为 FileSystemObject 的名称。
- filespec：必选项，filespec 是指定文件的路径（绝对路径或相对路径）。

如果指定文件不存在，则会出现错误。

下面例子说明如何使用 GetFile 方法：

```
Function ShowFileAccessInfo(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = f.Path & "<BR>"
  s = s & "Created: " & f.DateCreated & "<BR>"
  s = s & "Last Accessed: " & f.DateLastAccessed & "<BR>"
  s = s & "Last Modified: " & f.DateLastModified
  ShowFileAccessInfo = s
End Function
```

GetFileName 方法

返回指定路径（不是指定驱动器路径部分）的最后一个文件或文件夹。语法为：

```
object.GetFileName(pathspec)
```

参数说明：

- object：必选项，应为 FileSystemObject 的名称。
- pathspec：必选项，指定文件的路径（绝对路径或相对路径）。

如果 pathspec 不是以已命名的文件或文件夹结束，则 GetFileName 方法返回零长度字符串 ("")。

下面例子说明如何使用 GetFileName 方法：

```
Function GetAName(DriveSpec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  GetAName = fso.GetFileName(DriveSpec)
End Function
```

注意：GetFileName 方法只能对提供 path 的字符串起作用。它不能试图分析一个路径，也不能检查指定路径是否存在。

GetFolder 方法

返回与指定的路径中某文件夹相应的 Folder 对象。语法为：

```
object.GetFolder(folderspec)
```

参数说明：

- object：**必选项，应为** FileSystemObject **的名称。**
- folderspec：**必选项，**folderspec **是指定文件夹的路径（绝对路径或相对路径）。**

如果指定文件夹不存在，则会出现错误。

下面例子说明如何使用 GetFolder **方法返回文件夹对象：**

```
Sub AddNewFolder(path, folderName)
  Dim fso, f, fc, nf
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFolder(path)
  Set fc = f.SubFolders
  If folderName <> "" Then
    Set nf = fc.Add(folderName)
  Else
    Set nf = fc.Add("New Folder")
  End If
End Sub
```

GetParentFolderName 方法

返回字符串，该字符串包含指定的路径中最后一个文件或文件夹的父文件夹。语法为：

```
object.GetParentFolderName(path)
```

参数说明：

- object：**必选项，应为** FileSystemObject **的名称。**
- path：**必选项，指定路径，要返回文件或文件夹的父文件夹名。**

如果 path **参数指定的文件或文件夹无父文件夹，则** GetParentFolderName **方法返回零长度字符串（""）。**

下面例子说明如何使用 GetParentFolderName **方法：**

```
Function GetTheParent(DriveSpec)
   Dim fso
   Set fso = CreateObject("Scripting.FileSystemObject")
   GetTheParent = fso.GetParentFolderName(Drivespec)
End Function
```

注意：GetParentFolderName 方法只能对提供 path 的字符串起作用，它不能试图分析一个路径，也不能检查指定路径是否存在。

GetSpecialFolder **方法**

返回指定的特殊文件夹。语法为：

```
object.GetSpecialFolder(folderspec)
```

参数说明：

- object：**必选项，应为** FileSystemObject **的名称。**
- folderspec：**必选项，要返回的特殊文件夹名称。可以是“设置”部分列出的任何常数。**

folderspec **选项的取值如表** 6.2 **所示。**

表 6.2　olderspec **选项的取值**

常数	值	描述
WindowsFolder	0	Windows **文件夹，包含** Windows **操作系统安装的文件**
SystemFolder	1	System **文件夹，包含库、字体和设备驱动程序文件**
TemporaryFolder	2	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**

下面例子说明如何使用 GetSpecialFolder **方法：**

```
Dim fso, tempfile
Set fso = CreateObject("Scripting.FileSystemObject")

Function CreateTempFile
    Dim tfolder, tname, tfile
    Const TemporaryFolder = 2
    Set tfolder = fso.GetSpecialFolder(TemporaryFolder)
    tname = fso.GetTempName
    Set tfile = tfolder.CreateTextFile(tname)
    Set CreateTempFile = tfile
End Function

Set tempfile = CreateTempFile
```

```
tempfile.WriteLine "世界你好"
tempfile.Close
```

GetTempName 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。语法为：

```
object.GetTempName
```

可选项参数 object 为 FileSystemObject 对象的名称。

GetTempName 方法不创建文件，该方法仅提供临时文件名。CreateTextFile 使用该临时文件名创建文件。

下面例子说明如何使用 GetTempName 方法：

```
Dim fso, tempfile
Set fso = CreateObject("Scripting.FileSystemObject")

Function CreateTempFile
    Dim tfolder, tname, tfile
    Const TemporaryFolder = 2
    Set tfolder = fso.GetSpecialFolder(TemporaryFolder)
    tname = fso.GetTempName
    Set tfile = tfolder.CreateTextFile(tname)
    Set CreateTempFile = tfile
End Function

Set tempfile = CreateTempFile
tempfile.WriteLine "世界你好"
tempfile.Close
```

MoveFile 方法

将一个或多个文件从某位置移动到另一位置。语法为：

```
object.MoveFile source, destination
```

参数说明：

- object：必选项，应为 FileSystemObject 的名称。
- source：必选项，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：必选项，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。

如果 source 包含通配符或 destination 以路径分隔符（\）结束，则假定 destination 指定现有文件夹，将匹配文件移动到该文件夹中。否则，假定 destination 是要创建的目标文件。在任一种情况下，移动单个文件时，可能出现以下 3 种情况：

- 如果 destination 不存在，则进行文件移动。这是通常会发生的情况。
- 如果 destination 是已经存在的文件，则会出现错误。
- 如果 destination 是目录，则会出现错误。

如果在 source 使用通配符，但没有匹配文件时，也会出现错误。MoveFile 方法在遇到出现的第一个错误时停止。该方法不会撤消错误发生前所作的任何更改。

下面例子说明如何使用 MoveFile 方法:

```
Sub MoveAFile(Drivespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
  fso.MoveFile Drivespec, "c:\windows\desktop\"
End Sub
```

注意：GetParentFolderName **方法只有在已知的路径上起作用，它不能建立路径，也不能确定指定路径是否存在。**

MoveFolder 方法

将一个或多个文件夹从某位置移动到另一位置。语法为：

```
object.MoveFolder source, destination
```

参数说明：

- object：必选项，应为 FileSystemObject 的名称。
- source：必选项，要移动的文件夹的路径。source 参数字符串仅可在路径的最后一个组成部分中包含通配符。
- destination：必选项，指定路径，表示要将文件夹移动到该目标位置。destination 参数不能包含通配符。

如果 source 包含通配符或 destination 以路径分隔符（\）结束，则假定 destination 指定现有文件夹，将匹配文件移动到该文件夹中。否则，假定 destination 是要创建的目标文件夹。在任一种情况下，移动单个文件夹时，可能会发生以下 3 种情况：

- 如果 destination 不存在，则移动文件夹，这是通常会发生的情况。
- 如果 destination 是已经存在的文件，则会出现错误。
- 如果 destination 是目录，则会出现错误。

如果 source 使用通配符，但没有匹配文件夹时，则会出现错误。MoveFolder 方法在遇到出现的第一个错误时停止。该方法不会撤消错误发生前所作的任何更改。

下面例子说明如何使用 MoveFolder 方法:

```
Sub MoveAFolder(Drivespec)
  Dim fso
  Set fso = CreateObject("Scripting.FileSystemObject")
```

```
    fso.MoveFolder Drivespec, "c:\windows\desktop\"
End Sub
```

注意：只有当操作系统支持时，此方法才允许在两个卷之间移动文件夹。

OpenTextFile 方法

打开指定的文件并返回一个 TextStream **对象，可以读取、写入此对象或将其追加到文件。语法为：**

```
object.OpenTextFile(filename[, iomode[, create[, format]]])
```

参数说明：

- object：**必选项，应为** FileSystemObject **对象的名称。**
- filename：**必选项，字符串表达式，指明要打开的文件名称。**
- iomode：**可选项，输入/输出模式，是下列** 3 **个常数之一：**ForReading、ForWriting **或** ForAppending。
- create：**可选项，**Boolean **值，指出当指定的** filename **不存在时是否能够创建新文件。允许创建新文件时为** True，**否则为** False。**默认值为** False。
- format：**可选项，**3 **个** Tristate **值之一，指出以何种格式打开文件。若忽略此参数，则文件以** ASCII **格式打开。**

iomode **参数可为表** 6.3 **中设置值之一。**

表 6.3 iomode **参数的取值**

常数	值	描述
ForReading	1	**以只读模式打开文件。不能对此文件进行写操作**
ForWriting	2	**以只写方式打开文件。不能对此文件进行读操作**
ForAppending	8	**打开文件并在文件末尾进行写操作**

format **参数可为表** 6.4 **中设置值之一。**

表 6.4 format **参数的取值**

常数	值	描述
TristateUseDefault	－2	**以系统默认格式打开文件**
TristateTrue	－1	**以** Unicode **格式打开文件**
TristateFalse	0	**以** ASCII **格式打开文件**

以下例子说明如何使用 OpenTextFile **方法打开写文件：**

```
Sub OpenTextFileTest
    Const ForReading = 1, ForWriting = 2, ForAppending = 8
    Dim fso, f
```

```
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.OpenTextFile("c:\testfile.txt", For Writing, True)
    f.Write "嗨，你好！"
    f.Close
End Sub
```

3．程序实例

【例 6.2】　使用 FileSystemObject 显示驱动器信息

```
<HTML>
<HEAD>
<SCRIPT Language="VBScript">
<!--
Function ShowDriverInfo(drvPath)
    Dim fso,d,s,t
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set d = fso.GetDrive (fso.GetDriveName (drvPath))

    if not d.isReady then
        Msgbox ("驱动器没有准备好！")
        ShowFreeSpace = "驱动器没有准备好！"
        Exit Function
    end if

    Select Case d.DriveType
        Case 0: t = "未知"
        Case 1: t = "可移动"
        Case 2: t = "固定"
        Case 3: t = "网络"
        Case 4: t = "CD-ROM"
        Case 5: t = "RAM 磁盘"
    End Select

    s = "驱动器 " & ucase(drvPath) & " - "
    s = s & d.VolumeName & chr(13) & chr(10)
    s = s & "驱动器类型：" & t & chr(13) & chr(10)
    s = s & "文件类型为：" & d.FileSystem & chr(13) & chr(10)
    s = s & "序列号：" & d.SerialNumber & chr(13) & chr(10)
    s = s & "可用空间：" & FormatNumber(d.FreeSpace/1024,0) & "KB" & chr(13) & chr(10)
    s = s & "已用空间：" & (Clng(FormatNumber(d.TotalSize/1024,0)) –
            Clng(FormatNumber(d.FreeSpace/1024,0))) & "KB" & chr(13) & chr(10)
    s = s & "总空间：" & FormatNumber(d.TotalSize/1024,0) & "KB" & chr(13) & chr(10)
    ShowDriverInfo = s
End Function
```

```
Sub ShowDriveList()
   Dim fso,d,dc,e,t
   set fso = CreateObject("Scripting.FileSystemObject")
   set dc = fso.Drives
   For Each d in dc
       set e = document.createElement ("OPTION")

       Select Case d.DriveType
           Case 0: t = "未知"
           Case 1: t = "可移动"
           Case 2: t = "固定"
           Case 3: t = "网络"
           Case 4: t = "CD-ROM"
           Case 5: t = "RAM  磁盘"
       End Select

       e.Value = d.DriveLetter & ":"
       e.Text = d.DriveLetter & ":驱动器--" & t

       selectDriver.options.add e
   Next
End Sub

Sub selectDriver_OnChange()
    txtDriverInfo.value = ShowDriverInfo(selectDriver.value)
End sub

Sub window_onload()
    call ShowDriveList
End sub

Sub btnGet_mouseClicked()
    call ShowDriveList
End sub
-->
</SCRIPT>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>使用 FileSystemObject 对象读取驱动器信息</TITLE>
</HEAD>
<BODY>
<P align=center><STRONG>我的驱动器信息</STRONG> </P>
<P align =center> 
```

```
<SELECT    name=selectDriver size=10 style="left: 19px; top: 26px">
</SELECT>
<TEXTAREA name=txtDriverInfo rows=10></TEXTAREA> </P>

</BODY>
</HTML>
```

本例中使用了 FileSystemObject 对象和 Drive 对象和 Drivers 集合，并学习使用这些对象进行驱动器信息的访问方法。其中 FileSystemObject 对象是一个非常有用的对象，在很多情况下都会用到，对于 Drive 对象和 Drivers 集合我们将在以后的内容中讲到。

这个例子中用了两个控件，一个是下拉框控件，我们将其命名为 selectDriver，用来由用户选择一个驱动器；另一个是文本区控件，我们将其命名为 txtDriverInfo，用来显示用户选中的驱动器的信息。

在浏览器中打开该页面时，左边的下拉框中将自动填入用户计算机中的各个驱动器的名称。用户可以选择不同的驱动器名称，在右边的文本框内将出现相应的驱动器的信息，这些信息包括：驱动器的盘符、卷标、驱动器的类型、所采用的文件系统类型、序列号、该驱动器中的空间容量、可用容量和已用容量。如果所访问的驱动器没有准备好，如访问 CD-ROM 时，没有放入光盘，或者访问磁盘驱动器时，没有放入软盘，将提示“ 驱动器未准备好 !”。运行效果如图 6.4 所示。

图 6.4　使用 FileSystemObject 对象读取驱动器信息

下面我们对程序中的代码进行一下讲解。在本程序中，主要实现两个功能。其一，将所有的驱动器列出来；其二，分别把每个具体的驱动器的信息罗列出来。

在程序的开始，即页面被装载的时候，调用 ShowDriveList 过程，在下拉框中列出所有的驱动器的信息。所以，将调用 ShowDriveList 过程的语句放在窗口的 OnLoad 事件中。如下：

```
Sub Window_OnLoad()
    Call ShowDriveList
```

```
End Sub
```

当选中下拉框中其它选项的时候，将选中的驱动器的信息写在右边的文本区内。这个显示驱动器信息的功能由过程 ShowDriverInfo **来实现。**

```
Sub selectDriver_OnChange()
    TxtDriverInfo.value = ShowDriverInfo(selectDriver.value)
End Sub
```

下面我们专门来看 ShowDriveList **和** ShowDriverInfo **这两个过程。**

ShowDriveList **过程的代码如下：**

```
Sub ShowDriveList()
  Dim fso,d,dc,s,n,e,t
  set fso = CreateObject("Scripting.FileSystemObject")
  set dc = fso.Drives
  For Each d in dc
      set e = document.createElement ("OPTION")

      Select Case d.DriveType
          case 0: t = "未知"
          case 1: t = "可移动"
          case 2: t = "固定"
          case 3: t = "网络"
          case 4: t = "CD-ROM"
          case 5: t = "RAM  磁盘"
      End Select

      e.Value = d.DriveLetter & ":"
      e.Text = d.DriveLetter & ":驱动器--" & t

      selectDriver.options.add e
  Next
End Sub
```

程序说明：

- **首先定义变量**
  Dim fso,d,dc,e,t，**其中** fso **用于记录** FileSystemObject **对象，**d **用于记录** Drive **对象，**dc **用于记录** Drives **集合。**
- **建立** FileSystemObject **对象**

  ```
  Set fso = CreateObject("Scripting.FielSystemObject")
  ```

- **建立** Drives **集合**

```
Set dc = fso.Drives
```

- **访问** Drives **集合中的每个** Drive **对象**
  使用了 For Each d in dc **语句来依次使** Drives **访问集合中的每一个** Drive **对象。**
- **判断该驱动器的类型，获取驱动器的信息，并将该信息填入下拉框中。**

ShowDriverInfo **过程的代码如下：**

```
Function ShowDriverInfo(drvPath)
    Dim fso,d,s,t
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set d = fso.GetDrive (fso.GetDriveName (drvPath))

    if not d.isReady then
        Msgbox ("驱动器没有准备好！")
        ShowFreeSpace = "驱动器没有准备好！"
        Exit Function
    end if

    Select Case d.DriveType
        case 0: t = "未知"
        case 1: t = "可移动"
        case 2: t = "固定"
        case 3: t = "网络"
        case 4: t = "CD-ROM"
        case 5: t = "RAM  磁盘"
    End Select

    s = "驱动器  " & ucase(drvPath) & " - "
    s = s & d.VolumeName & chr(13) & chr(10)
    s = s & "驱动器类型：" & t & chr(13) & chr(10)
    s = s & "文件类型为：" & d.FileSystem & chr(13) & chr(10)
    s = s & "序列号：" & d.SerialNumber & chr(13) & chr(10)
    s = s & "可用空间：" & FormatNumber(d.FreeSpace/1024,0) & "KB" & chr(13) & chr(10)
    s = s & "已用空间：" & (Clng(FormatNumber(d.TotalSize/1024,0)) –
            Clng(FormatNumber(d.FreeSpace/1024,0))) & "KB" & chr(13) & chr(10)
    s = s & "总空间：" & FormatNumber(d.TotalSize/1024,0) & "KB" & chr(13) & chr(10)
    ShowDriverInfo = s
End Function
```

程序说明：

- **首先定义变量**
  Dim fso,d,s,t**，其中变量** fso **用于记录** FileSystemObject **对象，**d **用于记录** Drive **对**

象。

- **建立** FileSystemObject **对象**

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

- **根据** FileSystemObject **对象建立** Drive **对象**

```
Set d = fso.GetDrive (fso.GetDriveName (drvPath))
```

- **判断驱动器是否准备好，并作相应的操作**

```
if not d.isReady then
   Msgbox ("驱动器没有准备好！")
   ShowFreeSpace = "驱动器没有准备好！"
   Exit Function
end if
```

- **获取驱动器的类型信息**

```
Select Case d.DriveType
   Case 0: t = "未知"
   Case 1: t = "可移动"
   Case 2: t = "固定"
   Case 3: t = "网络"
   Case 4: t = "CD-ROM"
   Case 5: t = "RAM  磁盘"
End Select
```

- **显示驱动器的盘符和卷标**

```
s = "驱动器  " & ucase(drvPath) & " - "
s = s & d.VolumeName & chr(13) & chr(10)
```

- **显示驱动器的类型**

```
s = s & "驱动器类型：" & t & chr(13) & chr(10)
```

- **显示驱动器的文件类型**

```
s = s & "文件类型为：" & d.FileSystem & chr(13) & chr(10)
```

- **显示驱动器的序列号**

```
s = s & "序列号：" & d.SerialNumber & chr(13) & chr(10)
```

- **显示驱动器的容量信息，包括可用空间、已用空间和总空间**

```
s = s & "可用空间：" & FormatNumber(d.FreeSpace/1024,0) & "KB" & chr(13) & chr(10)
s = s & "总空间：" & FormatNumber(d.TotalSize/1024,0) & "KB" & chr(13) & chr(10)
```

6.1.3　Drive 对象

该对象提供对磁盘驱动器或网络共享属性的访问。该对象的语法为：

Scripting.FileSystemObject

1．Drive 对象的属性

AvailableSpace **属性**

该属性返回指定的驱动器或网络共享对于用户的可用空间大小。语法为：

object.AvailableSpace

DriveLetter **属性**

该属性为只读，返回本地驱动器或网络共享的驱动器号。语法为：

object.DriveLetter

DriveType **属性**

返回一个描述指定驱动器的类型的值。语法为：

object.DriveType

FileSystem **属性**

返回指定的驱动器使用的文件系统的类型。语法为：

object.FileSystem

FreeSpace **属性**

该属性为只读，返回指定的驱动器或网络共享对于用户的可用空间大小。语法为：

object.FreeSpace

IsReady **属性**

如果指定的驱动器就绪，返回 True**；否则返回** False**。语法为：**

object.IsReady

Path **属性**

返回指定文件、文件夹或驱动器的路径。语法为：

object.Path

路径不包含根目录。例如，C **驱动器的路径是** C:**，而不是** C:**。**
以下例子说明如何使用 Path **属性：**

```
Function ShowFileAccessInfo(filespec)
    Dim fso, d, f, s
```

```
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile(filespec)
    s = UCase(f.Path) & "<BR>"
    s = s & "创建时间：" & f.DateCreated & "<BR>"
    s = s & "上次访问时间：" & f.DateLastAccessed & "<BR>"
    s = s & "上次修改时间：" & f.DateLastModified
ShowFileAccessInfo = s
End Function
```

RootFolder **属性**

该属性为只读，返回一个 Folder **对象，表示指定驱动器的根文件夹。语法为：**

```
object.RootFolder
```

SerialNumber **属性**

返回十进制序列号，用于唯一标识一个磁盘卷。语法为：

```
object.SerialNumber
```

ShareName **属性**

返回指定的驱动器的网络共享名。语法为：

```
object.ShareName
```

TotalSize **属性**

返回驱动器或网络共享的总字节数。语法为：

```
object.TotalSize
```

VolumeName **属性**

设置或返回指定驱动器的卷标，可读写。语法为：

```
object.VolumeName [= newname]
```

object **为** Drive **对象的名称。**

参数 newname **是可选项。如果提供此参数，则** newname **为指定的** object **的新名称。**

2．Drive 对象的方法

Drive **对象没有任何方法。**

3．程序实例

【例 6.3**】** Drive **对象示例**

```
<HTML>
<HEAD>
<TITLE>
获取驱动器信息
```

```
</TITLE>
</HEAD>
<SCRIPT Language="VBScript">
<!--
Sub ShowDrvAtt(drvPath)
Dim fs,d,s
set fs = CreateObject("Scripting.FileSystemObject")
set d = fs.GetDrive(fs.GetDriveName(drvPath))
s = "驱动器 " & UCase(drvPath) & "-"
s = s & d.VolumeName & vbCrlf
s = s & "<BR>"
s = s & "文件系统：" & d.FileSystem
s = s & "<BR>"
s = s & "序列号：" & d.SerialNumber
s = s & "<BR>"
s = s & "卷标：" & d.VolumeName
document.write (s)
End Sub
ShowDrvAtt("c:")
-->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

在这个例子中，通过 Drive 对象，几乎显示了驱动器 C:的所有属性，运行结果如图 6.5 所示。

图 6.5　使用 VBScript 对象 Drive 运行示例

6.1.4 File 和 Folder 对象

File 对象提供了对文件的所有属性的访问，Folder 对象提供对文件夹所有属性的访问。这两个对象具有很多实用的方法和属性。由于它们有很多相同的属性和方法，我们在这里一并列出。

1．属性

Attributes 属性

设置或返回文件或文件夹的属性。可读写或只读（与属性有关）。语法为：

```
object.Attributes [= newattributes]
```

参数说明：

- Object：必选项，是 File 或 Folder 对象的名称。
- Newattributes：可选项，如果指定此参数，则 newattributes 为指定的 object 的属性的新值。

newattributes 参数可取表 6.5 中设置之一或其合理组合。

表 6.5 newattributes 参数的取值

常数	值	描述
Normal	0	普通文件。没有设置任何属性
ReadOnly	1	只读文件。可读写
Hidden	2	隐藏文件。可读写
System	4	系统文件。可读写
Directory	16	文件夹或目录。只读
Archive	32	上次备份后已更改的文件。可读写
Alias	1024	链接或快捷方式。只读
Compressed	2048	压缩文件。只读

如果对只读属性（别名，压缩或目录）有所改变，将被忽略。

当设置属性时，应首先阅读当前属性，然后按要求改变个别属性，最后反写属性。

以下例子说明如何使用 Attributes 属性：

```
Function ToggleArchiveBit(filespec)
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  If f.attributes and 32 Then
    f.attributes = f.attributes - 32
    ToggleArchiveBit = "清空归档位。"
  Else
```

```
    f.attributes = f.attributes + 32
    ToggleArchiveBit = "设置归档位。"
  End If
End Function
```

DateCreated 属性

只读属性，返回指定的文件或文件夹的创建日期和时间。语法为：

```
object.DateCreated
```

其中 object **为必选项。是** File **或** Folder **对象的名称。**
以下代码举例说明如何使用 DateCreated **属性：**

```
Function ShowFileInfo(filespec)
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  ShowFileInfo = "创建：" & f.DateCreated
End Function
```

DateLastAccessed 属性

只读属性，返回指定的文件或文件夹的上次访问日期和时间。语法为：

```
object.DateLastAccessed
```

其中 object **为必选项。是** File **或** Folder **对象的名称。**
以下例子说明如何使用 DateLastAccessed **属性：**

```
Function ShowFileAccessInfo(filespec)
  Dim fso, f, s
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFile(filespec)
  s = UCase(filespec) & "<BR>"
  s = s & "创建：" & f.DateCreated & "<BR>"
  s = s & "最后一次访问：" & f.DateLastAccessed & "<BR>"
  s = s & "最后一次修改：" & f.DateLastModified
  ShowFileAccessInfo = s
End Function
```

注意：此属性依赖于操作系统的行为。如果操作系统不支持提供时间信息，则 DateLastAccessed **属性不返回任何值。**

DateLastModified 属性

只读属性，返回指定的文件或文件夹的上次修改日期和时间。语法为：

```
object.DateLastModified
```

其中 object **为必选项。是** File **或** Folder **对象的名称。**
以下代码举例说明如何使用 DateLastModified **属性：**

```
Function ShowFileAccessInfo(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile(filespec)
    s = UCase(filespec) & "<BR>"
    s = s & "创建时间：" & f.DateCreated & "<BR>"
    s = s & "上次访问时间：" & f.DateLastAccessed & "<BR>"
    s = s & "上次修改时间：" & f.DateLastModified
ShowFileAccessInfo = s
End Function
```

Drive **属性**
只读属性，返回指定的文件或文件夹所在的驱动器的驱动器号。语法为：

```
object.Drive
```

其中 object **为必选项。是** File **或** Folder **对象的名称**
以下代码举例说明如何使用 Drive **属性：**

```
Function ShowFileAccessInfo(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile(filespec)
    s = f.Name & " 位于驱动器 " & UCase(f.Drive) & "<BR>"
    s = s & "创建时间：" & f.DateCreated & "<BR>"
    s = s & "上次访问时间：" & f.DateLastAccessed & "<BR>"
    s = s & "上次修改时间：" & f.DateLastModified
ShowFileAccessInfo = s
End Function
```

Name **属性**
设置或返回指定的文件或文件夹的名称。可读写。语法为：

```
object.Name [= newname]
```

其中 object **为必选项。是** File **或** Folder **对象的名称。**
Newname **为可选项。如果提供此参数，则指定的** object **名称更新为** newname**。**
以下例子说明如何使用 Name **属性：**

```
Function ShowFileAccessInfo(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile(filespec)
```

```
    s = f.Name & " 位于驱动器 " & UCase(f.Drive) & "<BR>"
    s = s & "创建时间：" & f.DateCreated & "<BR>"
    s = s & "上次访问时间：" & f.DateLastAccessed & "<BR>"
    s = s & "上次修改时间：" & f.DateLastModified
ShowFileAccessInfo = s
End Function
```

ParentFolder 属性

返回指定文件或文件夹的父文件夹。只读。语法为：

```
object.ParentFolder
```

其中 object **为** File、Folder **或** Drive **对象的名称。**

以下例子说明如何使用 ParentFolder **属性：**

```
Function ShowFileAccessInfo(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile(filespec)
    s = UCase(f.Name) & " 的父文件夹为 " & UCase(f.ParentFolder) & "<BR>"
    s = s & "创建时间：" & f.DateCreated & "<BR>"
    s = s & "上次访问时间：" & f.DateLastAccessed & vbCrLf
    s = s & "上次修改时间：" & f.DateLastModified
ShowFileAccessInfo = s
End Function
```

Path 属性

返回指定文件、文件夹或驱动器的路径。语法为：

```
object.Path
```

其中 object **为必选项。是** File **或** Folder **对象的名称。**

ShortName 属性

返回按照早期 8.3 **文件命名约定转换的短文件名。语法为：**

```
object.ShortName
```

其中 object **为必选项。是** File **或** Folder **对象的名称。**

以下例子说明如何使用 ShortName **属性：**

```
Function ShowShortName(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile(filespec)
    s = UCase(f.Name) & "<BR>"的短文件名"
        s = s & "是：" & f.ShortName
```

```
    ShowShortName = s
End Function
```

ShortPath 属性

返回按照 8.3 命名约定转换的短路径名。语法为：

```
object.ShortPath
```

其中 object 为必选项。是 File 或 Folder 对象的名称。

以下代码举例说明如何使用 ShortName 属性：

```
Function ShowShortPath(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile(filespec)
    s = UCase(f.Name) & "<BR>"的短路径名"
    s = s & "是：" & f.ShortPath
ShowShortPath = s
End Function
```

Size 属性

对于文件，返回指定文件的字节数；对于文件夹，返回该文件夹中所有文件和子文件夹的字节数。语法为：

```
object.Size
```

其中 object 为必选项。是 File 或 Folder 对象的名称。

以下例子说明如何使用 Folder 对象的 Size 属性：

```
Function ShowFolderSize(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFolder(filespec)
s = UCase(f.Name) & " 大小为 " & f.size & " 字节。"
ShowFolderSize = s
End Function
```

Type 属性

返回文件或文件夹的类型信息。例如，对于扩展名为 .TXT 的文件，返回“Text Document”。语法为：

```
object.Type
```

其中 object 为必选项。是 File 或 Folder 对象的名称。

以下例子说明如何使用 Type 属性返回文件夹的类型。在此示例中，试图向过程提供“回收站”或其他唯一文件夹的路径。

```
Function ShowFolderType(filespec)
    Dim fso, f, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFolder(filespec)
    s = UCase(f.Name) & " 的类型为 " & f.Type
End Function
ShowFolderType = s
```

Files 属性

返回由指定文件夹中所有 File 对象（包括隐藏文件和系统文件）组成的 Files 集合。语法为：

```
object.Files
```

其中 object 为 Folder 对象的名称。

以下例子说明如何使用 Files 属性：

```
Function ShowFileList(folderspec)
    Dim fso, f, f1, fc, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFolder(folderspec)
    Set fc = f.Files
    For Each f1 in fc
        s = s & f1.name
        s = s &   "<BR>"
    Next
ShowFileList = s
End Function
```

IsRootFolder 属性

如果指定的文件夹是根文件夹，返回 True；否则返回 False。语法为：

```
object.IsRootFolder
```

其中 object 为 Folder 对象的名称。

以下例子说明如何使用 IsRootFolder 属性：

```
Function DisplayLevelDepth(pathspec)
Dim fso, f, n
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.GetFolder(pathspec)
    If f.IsRootFolder Then
        DisplayLevelDepth ="指定的文件夹是根文件夹。"
    Else
        Do Until f.IsRootFolder
            Set f = f.ParentFolder
```

```
            n = n + 1
        Loop
        DisplayLevelDepth ="指定的文件夹是嵌套级为 " & n & " 的文件夹。"
    End If
End Function
```

SubFolders 属性

返回由指定文件夹中所有子文件夹(包括隐藏文件夹和系统文件夹)组成的 Folders 集合。语法为：

```
object.SubFolders
```

其中 object 为 Folder 对象的名称。

以下例子说明如何使用 SubFolders 属性：

```
Function ShowFolderList(folderspec)
    Dim fso, f, f1, s, sf
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFolder(folderspec)
    Set sf = f.SubFolders
    For Each f1 in sf
        s = s & f1.name
        s = s & "<BR>"
    Next
    ShowFolderList = s
End Function
```

2．方法

Copy 方法

将指定的文件或文件夹从某位置复制到另一位置。语法为：

```
object.Copy destination[, overwrite]
```

参数说明：

- object：必选项，为 File 或 Folder 对象的名称。
- destination：必选项，复制文件或文件夹的目标位置。不允许使用通配符。
- overwrite：可选项，Boolean 值。如果覆盖现有文件或文件夹，则 Boolean 值为 True（默认）；否则为 False。

对 File 或 Folder 应用 Copy 方法的结果与使用 FileSystemObject.CopyFile 或 FileSystemObject.CopyFolder 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。

下面的示例显示了 Copy 方法的使用：

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
MyFile.WriteLine("这是一个测试")
MyFile.Close
Set MyFile = fso.GetFile("c:\testfile.txt")
MyFile.Copy ("c:\windows\desktop\test2.txt")
```

Delete 方法

删除指定的文件或文件夹。语法为：

```
object.Delete force
```

参数说明：

- object：必选项，应为 File 或 Folder 对象的名称。
- force：可选项，Boolean 值。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）。

若指定文件或文件夹不存在，错误发生。有内容的文件夹和无内容的文件夹所使用的 Delete 方法无异。无论有无内容，指定文件夹皆被删除。

对 File 或 Folder 应用 Delete 方法的结果与使用 FileSystemObject.DeleteFile 或 FileSystemObject.DeleteFolder 执行的操作完全相同。

下面的示例说明了 Delete 方法的用法：

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
MyFile.WriteLine("这是一个测试。")
MyFile.Close
Set MyFile = fso.GetFile("c:\testfile.txt")
MyFile.Delete
```

Move 方法

将指定的文件或文件夹从某位置移动到另一位置。语法为：

```
object.Move destination
```

参数说明：

- object：必选项，应为 File 或 Folder 对象的名称。
- destination：必选项，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。

对 File 或 Folder 应用 Move 方法的结果与使用 FileSystemObject.MoveFile 或 FileSystemObject.MoveFolder 执行的操作完全相同。然而，要注意的是 FileSystemObject.MoveFile 或 FileSystemObject.MoveFolder 方法可移动多个文件或文件夹。

下面例子说明如何使用 Move 方法：

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
MyFile.WriteLine(“这是一个测试。”)
MyFile.Close
Set MyFile = fso.GetFile("c:\testfile.txt")
MyFile.Move "c:\windows\desktop\"
```

OpenAsTextStream 方法

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。语法为：

```
object.OpenAsTextStream([iomode, [format]])
```

参数说明：

- object：必选项，应为 File 对象的名称。
- iomode：可选项，输入/输出模式，是下列 3 个常数之一：ForReading、ForWriting 或 ForAppending。
- format：可选项，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 格式打开。

iomode 参数可为表 6.6 中设置之一。

表 6.6 iomode 参数的取值

常数	值	描述
ForReading	1	以只读模式打开文件。不能对此文件进行写操作
ForWriting	2	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	打开文件并在文件末尾进行写操作

format 参数可为表 6.7 设置之一。

表 6.7 format 参数的取值

常数	值	描述
TristateUseDefault	－2	以系统默认格式打开文件
TristateTrue	－1	以 Unicode 格式打开文件
TristateFalse	0	以 ASCII 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。

以下例子说明如何使用 OpenAsTextStream 方法：

```
Function TextStreamTest
    Const ForReading = 1, ForWriting = 2, ForAppending = 8
    Const TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0
    Dim fso, f, ts
    Set fso = CreateObject("Scripting.FileSystemObject")
    fso.CreateTextFile "test1.txt"                    '创建一个文件。
    Set f = fso.GetFile("test1.txt")
    Set ts = f.OpenAsTextStream(ForWriting, TristateUseDefault)
    ts.Write "嗨，你好！"
    ts.Close
    Set ts = f.OpenAsTextStream(ForReading, TristateUseDefault)
  TextStreamTest = ts.ReadLine
  ts.Close
End Function
```

CreateTextFile 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。语法为：

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

参数说明：

- object：必选项，为 FileSystemObject 或 Folder 对象的名称。
- Filename：必选项，字符串表达式，指明要创建的文件。
- Overwrite：可选项，Boolean 值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：可选项，Boolean 值指明是否以 Unicode 或 ASCII 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 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 文件。

以下例子说明如何使用 CreateTextFile 方法创建并打开文本文件：

```
Sub CreateAfile
  Dim fso, MyFile
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set MyFile = fso.CreateTextFile("c:\testfile.txt", True)
  MyFile.WriteLine("这是一个测试。")
  MyFile.Close
End Sub
```

对于 filename **已经存在的文件，如果** overwrite **参数为** False**，或未提供此参数时，则会出现错误。**

3．程序实例

【例 6.4**】 使用** File **对象得到文件的信息**

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>使用 File 对象得到文件的信息</TITLE>
</HEAD>
<BODY>
<SCRIPT Language="VBScript">
<!--
Sub cmdGetInfo_OnClick()
   GetFileInfo(document.frmFile.file1.value)
End Sub
Sub GetFileInfo(filepath)
   Dim fs,f,s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f    = fs.GetFile (filepath)
   s = "该文件的创建时间是：" & f.DateCreated
   s = s & chr(10) & "上次修改时间是：" & f.DateLastModified
   s = s & chr(10) & "文件大小为：" & f.Size
   s = s & chr(10) & "文件路径为：" & f.Path
   alert s
End Sub
-->
</SCRIPT>
<FORM name=frmFile>
<P><INPUT name=file1 type=file></P>
<P><INPUT name=cmdGetInfo type=button value=获取文件信息></P>
</FORM>
</BODY>
</HTML>
```

这个例子中包含了两个控件，一个是文件控件，用来选择一个文件；另一个是按钮控件，当用户单击这个按钮时，获取文件控件中选择的文件的属性。当然，还可以通过 File **对象的属性得到这个文件的更多信息。运行效果如图** 6.6 **所示。**

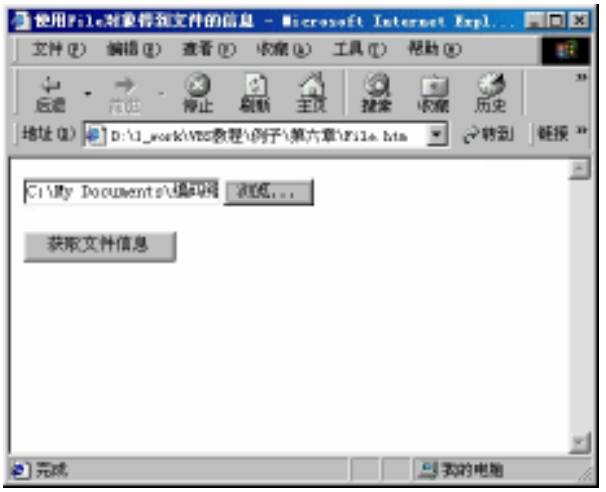

图 6.6　等待用户选择一个文件

用户选中一个文件后，获取该文件的信息。此时会弹出如图 6.7 所示的对话框。

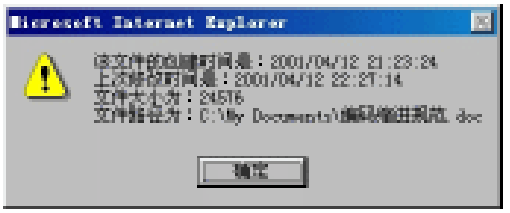

图 6.7　获取的文件信息对话框

6.1.5　TextStream 对象

使用 TextStream 对象可读取或写入文件数据，它为文件读写操作提供了必要的方法和属性。例如，可以创建一个新的文本文件或打开一个存在的文本文件，一旦这样做了，就有一个 TextStream 对象指向它，可以使用 TextStream 对象的属性和方法来操作文件。下面介绍 TextStream 对象的属性和方法。

1．TextStream 对象的属性

AtEndOfLine 属性

TextStream 文件中，如果文件指针指向行末标记，就返回 True；否则如果不是只读则返回 False。语法为：

```
object.AtEndOfLine
```

其中 object 为 TextStream 对象的名称。AtEndOfLine 属性仅应用于以读方式打开的 TextStream 文件，否则会出现错误。

AtEndOfStream 属性

如果文件指针位于 TextStream 文件末，则返回 True；否则如果不为只读则返回 False。

语法为：

```
object.AtEndOfStream
```

其中 object **为** TextStream **对象的名称。**AtEndOfStream **属性仅应用于以只读方式打开的** TextStream **文件，否则会出现错误。**

Column **属性**

只读属性，返回 TextStream **文件中当前字符位置的列号。语法为：**

```
object.Column
```

其中 object **是** TextStream **对象的名称。在写入新行字符后，但在写其他字符前，**Column **等于** 1**。**

Line **属性**

只读属性，返回 TextStream **文件中的当前行号。语法为：**

```
object.Line
```

其中 object **是** TextStream **对象的名称。文件刚刚打开并在写入任何信息前，**Line **等于** 1**。**

2．TextStream **对象的方法**

Read **方法**

从 TextStream **文件中读入指定数目的字符并返回结果字符串。语法为：**

```
object.Read(characters)
```

其中 object **是必选项，为** TextStream **对象的名称。**

Characters **为必选项，表示要从文件读的字符数目。**

下面例子说明如何使用 Read **方法从文件中读取** 8 **个字符并返回字符串结果：**

```
Function ReadTextFile
  Const ForReading = 1, ForWriting = 2, ForAppending = 8
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.Write "使用 Read 方法读取文件中指定数目的字符!"
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  ReadTextFile =   f.Read(8)
End Function
```

过程 ReadTextFile **的返回值为“ 使用** Read **方法 ”。**

ReadAll **方法**

读入全部 TextStream **文件并返回结果字符串。语法为：**

```
object.ReadAll
```

其中 object 为 TextStream 对象的名称。

对于大文件，使用 ReadAll 方法浪费内存资源。应该使用其他技术输入文件，例如按行读文件。

下面例子说明如何使用 ReadAll 方法从文件中读取全部字符并返回字符串结果：

```
Function ReadAllTextFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.Write "使用 ReadAll 方法从文件中读取全部字符并返回字符串结果!"
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  ReadAllTextFile =   f.ReadAll
End Function
```

过程 ReadAllTextFile 的返回值是“使用 ReadAll 方法从文件中读取全部字符并返回字符串结果!”。

ReadLine 方法

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。语法为：

```
object.ReadLine
```

其中 object 为 TextStream 对象的名称。

下面例子说明如何使用 ReadLine 方法从 TextStream 文件中读取字符并返回字符串:

```
Function ReadLineTextFile
   Const ForReading = 1, ForWriting = 2
   Dim fso, MyFile
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set MyFile = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
   MyFile.WriteLine "我们在学习 TextStream 对象。"
   MyFile.WriteLine "下面一个方法是 ReadLine。"
   MyFile.Close
   Set MyFile = fso.OpenTextFile("c:\testfile.txt", ForReading)
   ReadLineTextFile = MyFile.ReadLine
End Function
```

过程 ReadLine 返回字符串“我们在学习 TextStream 对象。”

Skip 方法

读取 TextStream 文件时跳过指定数目的字符。语法为：

```
object.Skip(characters)
```

参数说明：

- object：**必选项，为** TextStream **对象的名称。**
- Characters：**必选项，表示读取文件时跳过的字符数目。**

下面例子说明如何利用 Skip **方法在阅读文本文件前跳过开始的** 6 **个字符：**

```
Function SkipTextFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.Write "你看不到我，这就对了。"
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  f.Skip(6)
  SkipTextFile =   f.ReadLine
End Function
```

过程 SkipTextFile **的返回值是“ 这就对了。”。**

SkipLine **方法**

当读到 TextStream **文件时，跳过下一行。语法为：**

```
object.SkipLine
```

其中 object **为** TextStream **对象名称。**

跳过一行意味着读并放弃本行所有字符，同时包括下一新行字符内容。 如果文件不是以读方式打开则会出现错误。

下面例子说明如何使用 SkipLine **方法：**

```
Function SkipLineInFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.Write "VBScript 脚本很有趣！" & vbCrLf & "你想学会它吗？"
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  f.SkipLine
  SkipLineInFile = f.ReadLine
End Function
```

过程 SkipLineInFile **的返回值是“ 你想学会它吗？ ”**

Write **方法**

向 TextStream **文件写入指定字符串。语法为：**

```
object.Write(string)
```

参数说明：

- object：**必选项，为** TextStream **对象的名称。**
- String：**可选项，表示要写入文件的文本。**

指定的字符串被写入文件中，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。**

下面例子说明如何使用 Write **方法：**

```
Function WriteToFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.Write "使用 Write 方法在文件中写入字符。"
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  WriteToFile =   f.ReadLine
End Function
```

过程 WriteToFile **的返回值是“ 使用** Write **方法在文件中写入字符。”。**

WriteLine **方法**

向 TextStream **文件写入指定字符串，并附加一个换行符。语法为：**

```
object.WriteLine([string])
```

参数说明：

- object：**必选项，为** TextStream **对象的名称。**
- String：**可选项，要写入文件的文本。如果省略，将向文件写入新行字符。**

下面例子说明如何使用 WriteLine **方法：**

```
Function WriteLineToFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.WriteLine "使用 WriteLine 方法在文件中写入字符并附加一个换行符。"
  f.WriteLine "本行应该另起一行。"
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  WriteLineToFile = f.ReadAll
End Function
```

过程 WriteLineToFile **的返回值是：**

使用 WriteLine 方法在文件中写入字符并附加一个换行符。
本行应该另起一行。

WriteBlankLine 方法
在 TextStream 文件中写入指定数目的新行字符。语法为：

```
object.WriteBlankLines(lines)
```

参数说明：

- object：必选项，为 TextStream 对象的名称。
- Lines：必选项，要向文件写入的新行字符数目。

下面例子说明如何使用 WriteBlankLines 方法:

```
Function WriteBlankLinesToFile
  Const ForReading = 1, ForWriting = 2
  Dim fso, f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.OpenTextFile("c:\testfile.txt", ForWriting, True)
  f.WriteBlankLines 2
  f.WriteLine "我的前面有两个换行符。"
  Set f = fso.OpenTextFile("c:\testfile.txt", ForReading)
  WriteBlankLinesToFile = f.ReadAll
End Function
```

执行 WriteBlankLinesToFile 过程后，在文件 c:\testfile.txt 中写入了两个换行符和一串字符“我的前面有两个换行符。”

Close 方法
关闭打开的 TextStream 文件。语法为：

```
object.Close
```

其中 object 为 TextStream 对象的名称。
下面例子说明如何使用 Close 方法关闭打开的 TextStream 文件：

```
Sub CreateAFile
  Dim fso,f
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.CreateTextFile("c:\testfile.txt", True)
  f.WriteLine("这是一个测试。")
  f.Close
End Sub
```

6.1.6 Err 对象

在 VBScript 中，Err 对象是一个固有对象，在使用时，不需要创建其实例。Err 对象含

有关于运行时错误的信息。关于 Err 对象的详细介绍，我们在第 9 章“错误处理以及调试”中给出。

6.2　VBScript 中的集合

其实我们在前面讲 VBScript 对象的时候就已经用到了集合这个概念，并且还对其进行过一些操作。现在我们来具体地认识一下什么是 VBScript 中的集合。

VBScript 提供了 3 种集合：Drives 集合、Files 集合和 Folders 集合。所谓集合，就是一块存储数字、字符串、对象和其他值的区域。

集合的概念和数组有些相似，不同之处在于，当集合中的元素发生变化的时候，其他元素的位置会自动地发生变化，也就是说，当我们在集合中添加一个元素时，集合会自动增大，当我们在集合中删除一个元素的时候，集合会自动缩小，所以在使用的时候，不需要我们编写代码来手工调整，这一点要比数组方便。

要访问集合中的元素，有两种方式，一是通过元素的名称进行访问；二是利用索引编号获取保存在集合中相应的元素。

集合中每一个元素都拥有其位置索引，集合中元素索引编号从 1 开始，当我们向集合添加一个元素或者从集合中删除一个元素时，元素的索引会自动发生变化。一般情况下，不应该直接使用索引来访问集合中特定的元素，因为元素的索引有可能随时发生变化，而当我们需要枚举集合中所有元素的时候，使用索引来访问元素是很方便的。一般地，枚举集合中的所有元素都使用 VBScript 的 For Each…Next 结构。我们将在以后的例子中向大家介绍这一点。

6.2.1　Drives 集合

Drives 集合是所有可用驱动器的集合。无论是否插入媒体，可移动媒体驱动器都显示在 Drives 集合中。

以下例子说明如何获得 Drives 集合并使用 For Each...Next 语句枚举集合成员：

```
Function ShowDriveList ()
    Dim fso, d, dc, s, n
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set dc = fso.Drives
    For Each d in dc
      n = ""
   s = s & d.DriveLetter & " - "
        If d.DriveType = Remote Then
            n = d.ShareName
        ElseIf d.IsReady Then
            n = d.VolumeName
        End If
        s = s & n &"<BR>"
```

```
    Next
    ShowDriveList = s
End Function
```

Drives 集合有两个属性。

Count 属性

只读属性，返回一个集合或 Dictionary 对象包含的项目数。语法为：

```
object.Count
```

object 可以是“应用于”列表中列出的任何集合或对象的名称。

以下例子说明如何使用 Count 属性：

```
Function ShowKeys
  Dim a, d, i, s                ’创建一些变量
  Set d = CreateObject("Scripting.Dictionary")
  d.Add "a", "Athens"           ’添加一些键和项目
  d.Add "b", "Belgrade"
  d.Add "c", "Cairo"
  a = d.Keys                    ’获取键
  For i = 0 To d.Count -1       ’重复数组
    s = s & a(i) & "<BR>"       ’创建返回字符串
  Next
  ShowKeys = s
End Function
```

Item 属性

设置或返回 Dictionary 对象中指定的 key 对应的 item，或返回集合中基于指定的 key 的 item。可读写。语法为：

```
object.Item(key) [= newitem]
```

具体用法我们已经在介绍 Dictionary 对象的时候讲过，这里不再重复。

6.2.2 Files 集合

文件夹中所有 File 对象的集合。

以下例子说明如何获得 Files 集合并使用 For Each...Next 语句枚举集合成员。

【例 6.5】 使用 VBScript 的 Files 集合

```
<HTML>
<HEAD>
<SCRIPT Language="VBScript">
<!--
Function ShowFolderList(folderspec)
```

```
        Dim fso, f, f1, fc, s , i
        i = 1
        Set fso = CreateObject("Scripting.FileSystemObject")
        Set f = fso.GetFolder(folderspec)
        Set fc = f.Files
        For Each f1 in fc
            s = s & "<FONT size=2>" & i & ". " & f1.name & "</FONT>"
            s = s & "<BR>"
            i = i + 1
        Next
        fileCount = fc.Count
        ShowFolderList = s
        document.write s
End Function
call ShowFolderList("c:/")
-->
</SCRIPT>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY>
<P> </P>
</BODY>
</HTML>
```

该例的运行结果如图 6.8 所示。

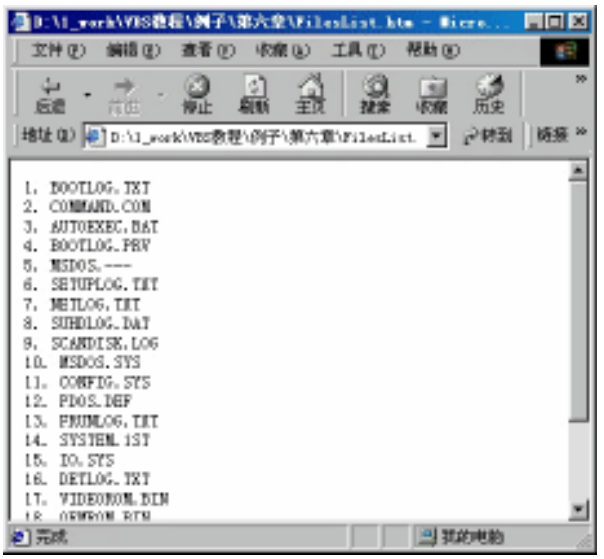

图 6.8　使用 VBScript 中的 Files 集合示例

同样 Files 集合也有 Count 属性和 Item 属性。

6.2.3 Folders 集合

包含在一个 Folder 对象中的所有 Folder 对象的集合。
以下例子说明如何获得 Folders 集合并使用 For Each...Next 语句枚举集合成员：

```
Function ShowFolderList(folderspec)
    Dim fso, f, f1, fc, s
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFolder(folderspec)
    Set fc = f.SubFolders
    For Each f1 in fc
        s = s & f1.name
        s = s & "<BR>"
    Next
    ShowFolderList = s
End Function
```

同样 Folders 集合也有 Count 属性和 Item 属性。
Folder 对象由一个方法：AddFolders 方法（Folders），该方法用来向 Folders 集合添加新 Folder。语法为：

```
object. Add(folderName)
```

参数说明：

- object：必选项，应为 Folders 集合的名称。
- FolderName：必选项，要添加的新 Folder 名称。

下面例子举例说明如何利用 Add 方法添加新文件夹：

```
Sub AddNewFolder(path, folderName)
  Dim fso, f, fc, nf
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set f = fso.GetFolder(path)
  Set fc = f.SubFolders
  If folderName <> "" Then
    Set nf = fc.Add(folderName)
  Else
    Set nf = fc.Add("New Folder")
  End If
End Sub
```

如果 folderName 已经存在，则会出现错误。

6.3　小　结

在本章中，我们学习了 VBScript 中的对象和集合，其中对象有：

- Dictionary 对象：保存数据键和项目对。
- FileSystemObject 对象：提供对计算机文件系统的访问。
- Drive 对象：提供对磁盘驱动器或网络共享的属性的访问。
- File 对象：提供对文件的所有属性的访问。
- Folder 对象：提供对文件夹所有属性的访问。
- TextStream 对象：有助于顺序访问文件。

集合包括：

- Drives 集合：所有可用驱动器的集合。
- Files 集合：文件夹中所有 File 对象的集合。
- Folder 集合：文件夹中的所有 Folder 对象的集合。

学会使用这些对象和集合，将会使你的程序具有更强大的功能。从下一章开始，我们将介绍表单中的控件及其事件。

练习题

1．在 HTML 中如何创建一个 VBScript 的对象实例？
2．如何向 Dictionary 对象中添加键和项目对？
3．当需要对文件系统进行访问时，使用哪个对象？
4．使用哪个集合来获得给定 Folder 对象中的所有文件和文件夹？
5．请设计一个程序，读出自己计算机上的所有驱动器信息。

第 7 章　表单中的控件及事件

到目前为止，我们已经学习了 VBScript 的大部分基本语法。总结起来，前面各章主要讲述了 VBScript 语法和内置函数，也就是说，讨论的是 VBScript 语言本身的结构。从这一章开始，我们将学习 VBScript 在网页中的高级应用。

在前几章中，我们创建的动态网页大都是由用户按了一个按钮后执行。本章我们将重点学习表单中的控件及事件的使用。

7.1　表单、控件与事件

我们知道，HTML 规范中常用的标记语法非常简单，这是 HTML 的一个特点，但是同时也使得它的功能受到很大的限制。HTML 表单的引入大大改变了这种状态。一个表单可以管理很多个界面元素，这些界面元素包括按钮、单选框等，其实，这些元素都是以 HTML 的形式存在的，但它们都可以接受用户的反馈要求。在很多情况下，我们都是通过对这些元素进行控制，从而完成许多我们想要的功能。只要使用过 Windows 系统的人，就可以很快掌握这样直观且自然的交互方式。利用表单可以完成 Web 网页同用户的交互工作——用户只要简单的填写表单，其余的工作就全部由表单自动完成。

网页中有了控件，就可以设计 VBScript 代码来响应用户在网页上的操作，从而使 Web 网页和用户交互，此种网页，我们将称之为活动网页，它能完成更多的工作。用户还可以设计自己的控件加到系统中。但是通常，多数用户是用不着自己设计控件的，因为系统已提供了足够丰富的内部控件，它们有自己的属性、方法和事件，我们直接拿来使用就可以了。内部控件上的主要事件是鼠标事件和键盘事件。本章主要学习使用内部控件及其用法。

事件是浏览器响应用户交互操作的一种机制。任何程序包括浏览器本身都有一套已经设计好的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。VBScript 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，这样就可以开发出更加具有交互性，更易使用的 Web 页面。

在 Windows 系统中，事件在窗口中生成消息，窗口的消息告诉应用程序或操作系统做什么以及让它们知道某个事件发生了。加在网页上的多数对象（控件也是一种对象）都能产生事件，事件发生（称为触发）时脚本代码将被执行。在前几章的应用中我们已经多次使用了按钮控件的 OnClick 事件，这个事件被触发时，OnClick 过程中的代码将被执行。

事件定义了用户与 Web 页面交互时产生的各种操作。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，自动调用事件处理函数，完成事件处理过程。

事件不仅可以在用户交互过程中产生，而且浏览器自己的一些动作也可能产生事件。例如，浏览器载入一个网页的时候，就会产生一个 Load 事件。

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。图 7.1 表示了通知事件和处理事件的过程。

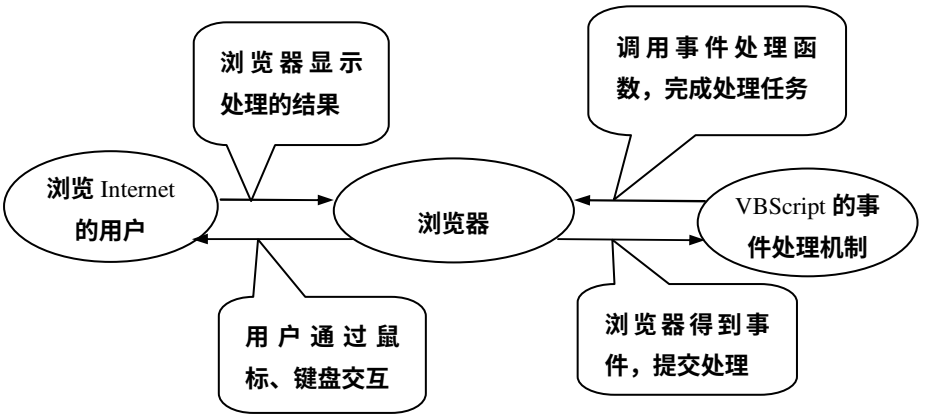

图 7.1　通知事件和处理事件的过程

事件处理过程是与对象紧密相连的，对象可以是表单也可以是表单内的控件，每一种对象都有若干个可以识别的事件。例如，单击某个按钮将产生按钮的 OnClick 事件。在程序执行的过程中，当某一个事件发生时，程序就转向响应控件的事件处理过程。事件处理过程的一般格式为：

```
Sub 对象名称_事件名称（参数表）
    ……
    代码
    ……
End Sub
```

7.2　表单与控件的使用

在 HTML 中使用表单，必须使用成对标记<FORM>和</FORM>。<FORM>标记的完整格式如下：

```
<FORM
action = url
class = classname
encTYPE = encoding
id = VALUE
lang = LANGUAGE
language = VBScript　| VBS | JavaScript | Jscript
method = get | post
name = EWRS
style = cssl-properties
```

target = window_name | _blank | _parent | _self | _top
title = text
event = Script
>

其中有一些参数是经常要用到的，而且对于所有完成的任务也是非常重要的。

- action：**定义了窗体所有提交到页面的** URL **地址。例如：**

  action="form_hdl.asp"

  表示这个表单提交后将由 form_hdl.asp **去处理。**

- language：**定义了当前的脚本语言以及所使用的脚本引擎。**
- method：**表示窗体数据提交的方式，可能的值是** GET **和** POST。
  GET **方式：如果是以** GET **方式提交，则告诉浏览器从提交的表单中将所有数据追加到查询字符串** QueryString **的** URL **中。这样就可以通过使用** Request.Querystring **集合被服务器端的脚本读取了。**
  其中，查询字符串是指追加到 URL **中的字符串，用以向服务器传送少量数据。具有追加查询字符串的** URL **语法为：**

  http://www.sitename.com/file.extension?field1=VALUE1&field2=VALUE2

  查询字符串自身有引导问号，包含几对字段名和分配的字段值。不同的名称和值对用“ & **”号分隔开，因为它与** URL **一起传送，所以其长度受到限制。**
  POST **方式：该方法不把数据作为字符串追加到** URL **中，而是在** FORM **集中包含了表单元素的值。在服务器中，用** Requet.Form **集合来读取数据。由于不需要将字符串附加到** URL **中传送，所以字段值的数目和大小几乎没有限制。但是，传送数据所需的时间会随着发送的数据量的增加而增加。**
- event：**可以是以下事件：**OnClick、OnDblClick、OnDragStart、OnHelp、OnKeyDown、OnKeyPress 、 OnKeyUp 、 OnMouseDown 、 OnMouseMove 、 OnMouseOut 、 OnMouseOver、OnReset、OnSelectStart **和** OnSubmit。

使用<INPUT>**标记在表单中引入控件。格式为：**

<INPUT
accesskey = key
align = left | center | right
alt = text
class = classname
disabled
dynsrc = url
id = VALUE
lang = LANGUAGE
language = VBScript | VBS | JavaScript | Jscript

```
lowsrc = url
maxlength = n
name = name
readonly
size = n
src = url
style = cssl-properties
tabindex = n
title = text
type = button | checkbox | file | hidden | image | password | radio |reset |Submit | text
value = VALUE
event = script
>
```

<INPUT>**标记最重要的参数是** type**，这个参数定义了将要引入控件的类型，如** text**、**button **和** checkbox **等等，默认状态下为** text**。每一种类型的控件及其具体属性与事件，我们将在以后的各节中详细讲述。**

7.3　文本框控件

文本框控件是应用最广泛的一种输入控件，是内部输入控件的缺省类型。用户可以在其中输入单行文本。文本框作为一种控件，有其自身的属性、方法和事件。实际上，所有的控件都有属性、方法和事件。

7.3.1　文本框的属性

disabled **属性**
控件不是活动的，即不能对该控件进行任何操作，默认情况下是活动的。

form **属性**
控件所属表单的名字。

name **属性**
代码中控件的名字。

value **属性**
控件的内容，可以用它来设置控件的初始值，当用户将其修改后，value **的值将变为修改后的值。我们也可以通过设置文本框的值来改变文本框的值，如：**

```
Document.form1.text1.value = "VBScript"。
```

例如，我们在 HTML **文档中引入一个文本框控件：**

```
<INPUT name = text1 type = text value = "第一个文本框" disabled>
```

这个控件的默认值为“第一个文本框”，且不能被改变（因为将其设为 disabled 了）。

另外，还有一种控件密码域控件，它与文本框控件基本相同，但是用户键入的每一个字符都以“*”显示，通常用来提示用户输入口令。我们可以这样定义一个密码域控件：

```
<INPUT name = password1 type = password>
```

7.3.2 文本框的方法

Focus 方法
得到聚焦。

Blur 方法
失去聚焦。

Select 方法
选中控件的内容，即文本框中的文字被全部加亮显示。

这 3 种方法经常用于有效性检验，比如当用户输入的资料在有效检验失败之后，将其输入焦点放在检验出错的那个元素上，以方便用户的再次输入。下面就举一个例子来说明这个用法，这个例子中用到了 focus、select 方法。

【例 7.1】 域的有效性检验

本例中的两个文本框要求分别输入姓名和年龄，其中姓名文本框不能为空，在年龄文本框中，如果输入的值小于 0 或不是数字则视为非法输入。

```
<HTML>
<HEAD>
<TITLE>域的有效性检验</TITLE>
</HEAD>
<BODY>
<FORM name = frm1 method = post action = "Form_hdl.asp">
请输入您的姓名：<INPUT name=name><P></P>
请输入您的年龄：<INPUT name=age><P></P>
<INPUT name=cmdOK type=button value=填好了 onClick = "Validation()">
<SCRIPT Language="VBScript">
<!--
Dim name,age
Sub validation()
name = trim(document.frm1.name.VALUE)
age    = trim(document.frm1.age.VALUE)
If isnull(age) or (age = "")Then
    age = 0
End If
If name = "" or isnull(name) Then
```

```
    alert ("姓名不能为空！")
    document.frm1.name.focus ()
    document.frm1.name.select ()
ElseIf not isnumeric(age) Then
    alert ("您的输入有错误！年龄应该为数字！")
    document.frm1.age.focus ()
    document.frm1.age.select ()
ElseIf Cint(age) < 0 Then
     alert ("您的输入有错误！年龄必须大于 0！")
     document.frm1.age.focus ()
     document.frm1.age.select ()
Else
     alert (name & "的年龄是" & age & "岁。")
End If
End Sub
-->
</SCRIPT>
</FORM>
<P> </P>
</BODY>
</HTML>
```

运行结果如图 7.2 所示。

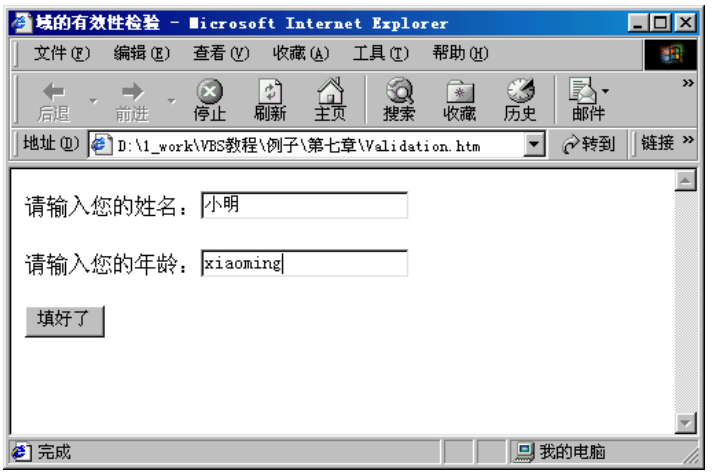

图 7.2　输入了错误的年龄

当用户的输入不符合要求时，就出现如图 7.3 所示的错误提示。

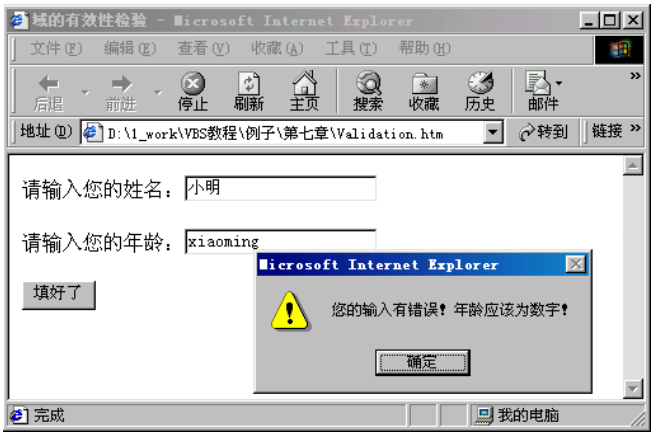

图 7.3　年龄输入不是数字，显示错误提示框

当用户单击提示框的“确定”按钮后，用户输入页面的出错的文本框获得聚焦，并且文本框中的文字加亮显示，如图 7.4 所示。

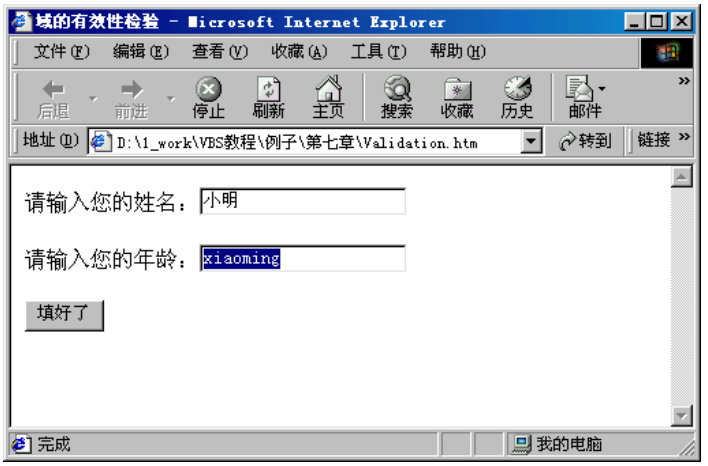

图 7.4　焦点停留在年龄文本框中，且文本框中的文字被选中加亮

这只是一个简单的实例，而实际上的合法性检验包含许多重要的内容，比如年龄除了必须为大于 0 的数字外，其长度也应该控制在一定的范围内等，但其原理都是相同的。

7.3.3　文本框的事件

OnFocus **事件**
得到聚焦时触发。

OnBlur **事件**
失去聚焦时触发。

OnChange **事件**
控件的值改变时触发。

OnSelect 事件

控件被选择时触发。

OnKeyDown、OnKeyUp 和 OnDeyPress 事件

当用户按下一个键时触发这些事件，当用户按键时，这 3 个事件的触发顺序为：OnKeyDown、OnKeyUp 和 OnDeyPress。

OnMouseMove 事件

移动鼠标时触发。

OnMouseOver

鼠标扫过时触发。

OnMouseOut 事件

鼠标扫过并脱离时触发。

OnMouseDown

当按下鼠标上一个键时触发。

OnMouseUp

释放鼠标上一个键时触发。

在【例 7.1】中，我们是在用户输入了所有的域之后，利用按钮的单击事件触发的。但是有一个更好的方法，就是在文本框刚失去聚焦（OnBlur）的时候，就去检验，使用户及早地发现自己的错误并加以改正。如果输入了错误的文本，它就要求你必须进行改正，你无法在其他的文本框内输入，除非改正了错误，否则警告框会跟随你的每一个动作出现，不管是你单击背景或其他程序，OnBlur 事件的处理函数总会出现一个报警框，令你头疼不已，所以此时最好正确输入。我们将【例 7.1】中的例子改写为使用 OnBlur 事件触发。

【例 7.2】

```
<HTML>
<HEAD>
<TITLE>域的有效性检验</TITLE>
</HEAD>
<BODY>
<FORM name = frm1 method = post action = "Form_hdl.asp">
请输入您的姓名：<INPUT name=name><P></P>
请输入您的年龄：<INPUT name=age><P></P>
<INPUT name=cmdOK type=button value=填好了 onClick = "Validation()">
<SCRIPT Language="VBScript">
<!--
Dim name,age
Sub name_OnBlur()
name = trim(document.frm1.name.VALUE)
If name = "" or isnull(name) Then
```

```
        alert ("姓名不能为空！")
        document.frm1.name.focus ()
        document.frm1.name.select ()
    End If
    End Sub

    Sub age_OnBlur()
    age = document.frm1.age.VALUE
    If isnull(age) or (age = "")Then
        age = 0
    End If
    If not isnumeric(age) Then
        alert ("您的输入有错误！年龄应该为数字！")
        document.frm1.age.focus ()
        document.frm1.age.select ()
    ElseIf Cint(age) < 0 Then
         alert ("您的输入有错误！年龄必须大于 0！")
         document.frm1.age.focus ()
         document.frm1.age.select ()
    End If
    End Sub

    Sub Validation()
        alert (name & "  的年龄是" & age & "岁。")
    End Sub
    -->
    </SCRIPT>
    </FORM>
    <P> </P>
    </BODY>
    </HTML>
```

在这个例子中，定义了 3 个过程：name_OnBlur()、age_OnBlur()**和** Validation()**，这 3 个过程分别实现检验姓名的有效性、检验年龄的有效性、显示姓名和年龄。而在【例** 7.1**】中，只定义了一个过程** Validation()**，在这个过程中实现了上述 3 个过程的功能，它仅在用户单击“ 填好了 ”按钮时被执行，因为在这个例子中需要用户填写的数据不多，所以这么做也没什么不妥。但是我们经常看到一些网页上尤其是用户申请注册时，需要填写很多的数据，在这种情况下，最好使用第二种方法，即一旦用户填完一个输入框，就进行检验，这样可以避免用户出现更多的错误。**

值得注意的是，在使用对象的事件时，千万不要创建重叠事件，即循环触发的事件。通俗一点地说，就是在处理这个事件的代码中触发的另一个事件，而在另一个事件的处理代码中又触发的这个事件，这样就可能造成了死循环，使整个过程不能进行下去。举例来

说，在【例 7.2】中，如果我们省略了以下这段代码：

```
If isnull(age) or (age = "")Then
    age = 0
End If
```

则程序就可能出现死循环，不能继续进行。我们分析一下，当在姓名的文本框内不填写任何文字，直接将光标放在年龄的文本框内，这个时候就触发了姓名文本框的 OnBlur 事件，开始执行。由于满足条件 name = ""，所以弹出“姓名不能为空！”的警告框，下一条语句要求姓名文本框重新得到聚焦，而这个时候因为年龄文本框已经得到聚焦，再要求将聚焦置于姓名文本框中，则又触发了年龄文本框的失去聚焦 OnBlur 事件。程序接下来执行 age_OnBlur()中的代码，由于满足条件 not isnumeric(age) ，所以弹出“您的输入有错误！年龄应该为数字！”的警告框，程序继续执行时，年龄文本框要求得到聚焦，从而又触发姓名文本框的失去聚焦 OnBlur 事件。如此反复，导致程序不能继续下去。

因此，我们在对 VBScript 中控件的事件编程时，一定小心，不要创建重叠的事件。

7.3.4　程序实例

【例 7.3】　变化的文字

```
<HTML>
<HEAD>
<TITLE>变化的文字</TITLE>
</HEAD>
<BODY>
<FORM name=frm1>
<P><INPUT name=text1 value="我叫鼠标">
</P>
</FORM>
<SCRIPT Language="VBScript">
<!--
Sub text1_OnMouseMove()
    document.frm1.text1.VALUE="欢迎您来到我身边！"
End Sub

Sub text1_OnMouseOut()
    document.frm1.text1.VALUE="真的要走了吗？"
End Sub
-->
</SCRIPT>
<P> </P>
</BODY>
</HTML>
```

这个例子很简单，只有一个文本框，应用了鼠标事件 OnMouseMove 和 OnMouseOut。

我们将文本框的初始值设为“我叫鼠标”，如图 7.5 所示；当鼠标移动到这个文本框上的时候，将文本框的值改为“欢迎您来到我身边！”如图 7.6 所示；当鼠标从文本框上移走的时候，将文本框的值改为“真的要走了吗？”，如图 7.7 所示。

图 7.5 文本框的初始值为“我叫鼠标”

图 7.6 鼠标移到文本框上，将值改为“欢迎您来到我身边！”

图 7.7 鼠标移出文本框时，将值改为“真的要走了吗？”

7.4　文本区控件

文本区 TEXTAREA 控件类似于文本控件，也可以用来在网页输入文本，不同之处在于可以输入多行文本，适合输入大量数据。在网页中文本区域控件通常不用输入标签创建，而是直接使用<TEXTAREA>和</TEXTAREA>成对标记来创建，<TEXTAREA>和</TEXTAREA>中间的文本就是该控件的初始值。

文本区控件包括下面的一些属性。

cols 属性

列数。

rows 属性

行数。

value 属性

控件的内容，使用<TEXTAREA>和</TEXTAREA>对标记引用该控件，不能用 value 属性设置控件的初始值，而只能在标记对中间设置初始值，当用户将其修改后，value 的值将变为改后的值。我们也可以通过设置文本框的值来改变文本框的值，例如：Document.form1.textarea1.value= "新概念 VBScript 教程"。

disabled 属性

控件不是活动的，即不能对该控件进行任何操作，默认情况下是活动的。

form 属性

控件所属表单的名字。

name 属性

代码中控件的名字。

属性 cols 和 rows 控制了控件在网页中的大小。

<TEXTAREA name="textarea1" rows=20 cols=40>文本区域控件一般用来输入多行文本,适合大量文本的输入。

</TEXTAREA>

图 7.8 显示了这个文本区域。

文本区域控件的方法和事件同文本框基本相同。可以用 focus 和 blur 得到聚焦或失去聚焦,使用 Select 方法选中其中的文本等,还可以用 OnFocus,OnBlur,OnChange 和 OnSelect 事件。

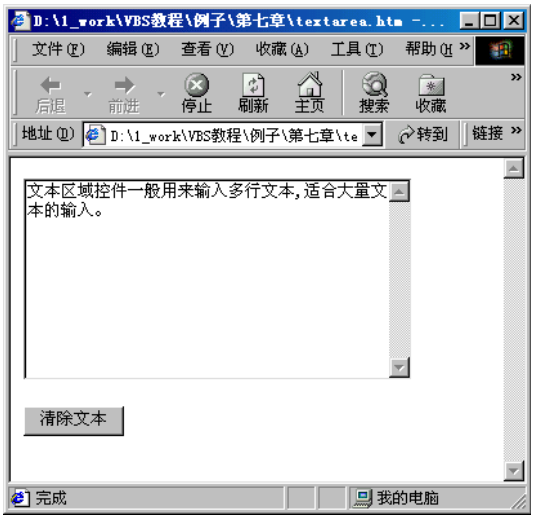

图 7.8 文本区域

7.5 按 钮 控 件

几乎所有的表单都离不开按钮，人们很多时候已习惯了使用按钮来发送命令。按钮控件分为 3 种：

- **普通按钮：**type = Button
- **提交按钮：**type = Submit
- **重置按钮：**type = ReSet

这 3 种按钮各有不同的功能，应该根据这些按钮的不同功能学会在不同的情况下使用不同的按钮。

7.5.1 普通按钮

一个普通的按钮可以用以下方式定义：

```
<INPUT type =button name=button1 value="Click Me">
```

type **表明了控件的类型，**name **指出代码中控件的名字，**value **描述的是在按钮上显示的文字。**

1．按钮的方法

Click **方法**

用户对按钮使用最多的动作莫过于单击按钮了。Click **方法是按钮最重要和最常用的方法。**Click **方法可以模拟单击按钮的动作，从而可以触发按钮的** onclick **事件，达到用户单击按钮相同的效果。**

Blur 方法

该方法使得按钮对象失去聚焦，并触发 onBlur 事件。

focus 方法

该方法使得按钮对象获得聚焦，并执行由 onFocus 事件所定义的处理代码。

2．按钮的事件

OnClick 事件

该事件是我们必须掌握的，从本书的开始，我们就开始接触按钮的单击事件，相信读者对这个事件已经比较熟悉。凡是单击了按钮之后，必定要触发 onClick 事件，所有的由单击按钮引起的处理事件均可以在 onClick 事件中完成。具体地说，就是当用户按下并释放了鼠标左键或当用户在按钮具备焦点时按下了键盘上的键，如 Enter 键、Esc 键和 SpaceBar 键时，都将触发 OnClick 事件。OnClick 事件的处理函数可以使用 VBScript 脚本来编写，这样也就将按钮控件和 VBScript 相联系了。

OnMouseDown 事件

当鼠标按下按钮并不弹起时，触发 OnMouseDown 事件。

OnMouseUp 事件

当鼠标从按钮上弹起时，触发 OnMouseUp 事件，OnMouseDown 事件和 OnMouseUp 事件都是在 OnClick 事件触发之前发生。

除上述常用事件外，按钮控件还有 OnBlur、OnDblClick、OnFocus、OnHelp、OnKeyDown、OnKeyPress、OnKeyUp、OnMouseMove、OnMouseOut、OnMouseOver、OnResize、OnSelect 事件，由于不很常用，就不再赘述了。

3．程序实例

【例 7.4】　按钮的单击事件

```
<HTML>
<HEAD>
<SCRIPT Language="VBScript">
<!--
Sub TotalCount(SelectObject)
   numberSelected = 0
   for i = 0 to (SelectObject.length-1)
         If SelectObject.item(i).selected = true Then
           numberSelected = numberSelected + 1
       End If
   next
   alert "选中的城市数量为：" & numberSelected
End Sub
-->
</SCRIPT>
<TITLE>几个城市</TITLE>
</HEAD>
```

```
<BODY>
<FORM name="selectForm">
<P>
<B>
请选择一些您想去的城市，可以选择多个（同时按 Ctrl）。
</B>
<BR>
<SELECT name="city" multiple size=7>
    <OPTION selected>北京</OPTION>
    <OPTION>上海</OPTION>
    <OPTION>深圳</OPTION>
    <OPTION>广州</OPTION>
    <OPTION>重庆</OPTION>
    <OPTION>杭州</OPTION>
    <OPTION>哈尔滨</OPTION>
</SELECT>
<P>
<INPUT type="button" value="看看我选中了几个城市"
        onClick="TotalCount(document.selectForm.city)">
</FORM>
<P> </P>
</BODY>
</HTML>
```

该例中用到了一个下拉框列表控件和一个按钮控件，其中下拉框控件是可多选的，用 multiple 标志。当用户选择城市并单击‘看看我选中了几个城市’按钮时，触发按钮的 OnClick 事件，请求执行 TotalCount 过程。TotalCount 过程计算用户选中的城市的数量，并以提示框告知用户。运行的结果如图 7.9 所示。

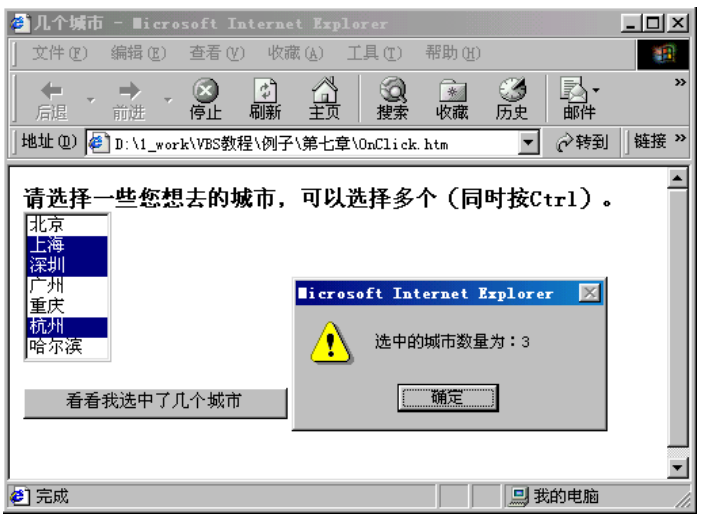

图 7.9 OnClick 事件程序示例——计算用户选中的城市个数

【例 7.5】　按钮的 OnMouseDown 事件和 OnMouseUp 事件

OnMouseDown 事件和 OnMouseUp 事件是与 OnClick 事件紧密相关的，在触发 OnClick 事件前先后触发 OnMouseDown 和 OnMouseUp 事件，本例在按钮按下时更改图片，在按钮弹起时恢复图片。

```
<HTML>
<HEAD>
<SCRIPT Language="VBScript">
<!--
Sub ChangeImage()
  document.frmImage.img1.src="image2.gIf"
End Sub
Sub RestoreImage()
  document.frmImage.img1.src="image1.jpg"
End Sub
-->
</SCRIPT>
<TITLE>改变图片</TITLE>
</HEAD>
<BODY>
<FORM name=frmImage>
<P>
<IMG name=img1 src="image1.jpg" align=center width=200 height = 200>
<P>
<INPUT type=button name=button1 value="改变图片"
        OnMouseDown="ChangeImage()"
        OnMouseUp="RestoreImage()">
</P>
</FORM>
<P> </P>
</BODY>
</HTML>
```

在该例中，当用户按下按钮时，触发 OnMouseDown 事件，该事件请求执行 ChangeImage 过程，在过程 ChangeImage 中，通过“ documents.frmImage.img1="image2.gIf" ”将另外一幅图片代替原图片；当鼠标弹起时，图像又恢复原样。这是通过触发 OnMouseUp 事件，并执行 RestoreImage 过程完成的。运行效果如图 7.10 和图 7.11 所示。

试一下，如果我们按下按钮但是并不在按钮上弹起，而是将鼠标从按钮上拖走，在其它位置弹起鼠标，我们发现，图片不能恢复成原来的那幅图片了。这是为什么呢？原来，因为鼠标没有在按钮上弹起，因此不能触发按钮的 OnMouseUp 事件，所以图片就恢复不过来了。

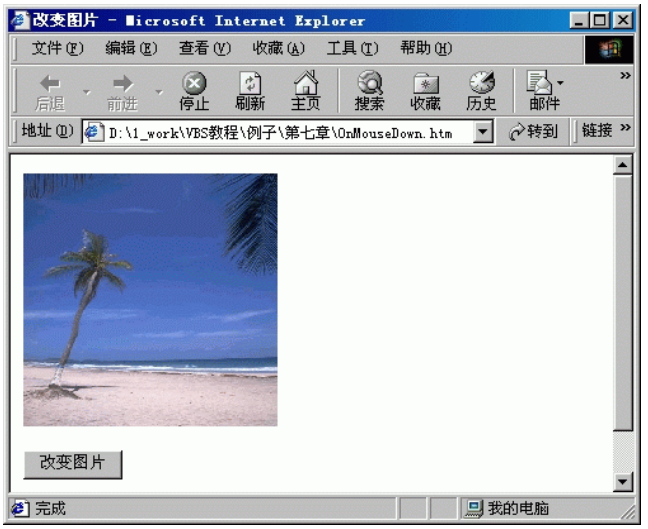

图 7.10　显示原图片的页面

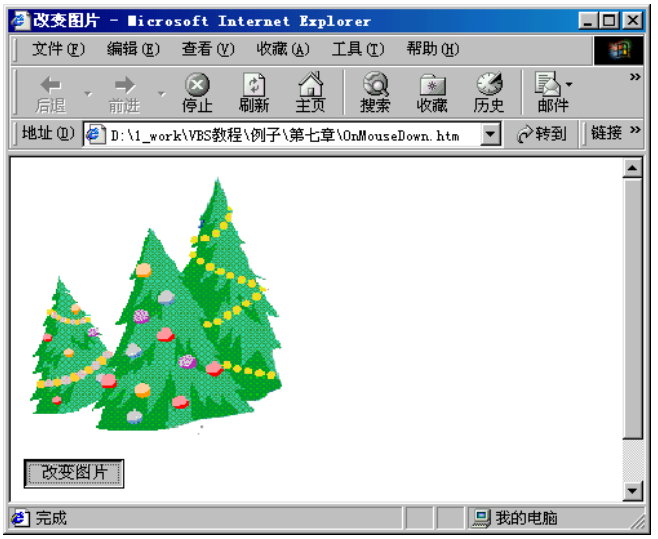

图 7.11　按下按钮且没有弹起时，原图片被替换

7.5.2　提交按钮

提交按钮是 type **为** Submit **的按钮，其不同于普通按钮是因为它被单击之后，执行特殊的功能——将表单的资料发往服务器。用下列语法定义一个提交按钮：**

```
<INPUT type=Submit value=Submit1>
```

响应提交按钮单击事件的是 OnSubmit **事件，而不是** OnClick **事件，它是** Form **表单的事件，因此可以将这个事件函数的声明写在**<FORM>**标记中。**

```
<HTML>
```

```
<HEAD>
<SCRIPT language="VBScript">
<!--
Function Validation()
   ......
End Function
-->
</SCRIPT>
<TITLE>提交检验</TITLE>
</HEAD>
<BODY>
<FORM name=frmSubmit onSubmit="Validation()">
......
<INPUT type=Submit name=Submit1 value="提交">
</FORM>
<P> </P>
</BODY>
</HTML>
```

上述这段简略代码表示当用户提交表单时，首先触发事件 OnSubmit**，该事件请求执行** Validation **过程，我们经常采用这种方式实现客户端的有效性检验。**

7.5.3　重置按钮

重置按钮是 type **为** Reset **类型的按钮，通常与提交按钮成对出现，它的功能是将其所属表单中的所有元素恢复到它们的初始值。单击重置按钮将触发** OnSet **事件，它也是表单的事件，因此也可以将这个事件函数的声明写在**<FORM>**标记中。**

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
<!--
Function ResetForm()
   ......
End Function
-->
</SCRIPT>
<TITLE>重置表单</TITLE>
</HEAD>
<BODY>
<FORM name=frmRetForm onSubmit="ResetForm()">
......
<INPUT type=reset name=reset1 value="重置">
</FORM>
```

```
<P> </P>
</BODY>
</HTML>
```

7.6 单选框控件

使用单选框和复选框可以减少用户的输入量，在网页设计过程中非常常用。单选框通常成组出现，同组的单选框的 name 属性值必须相同，任何时刻，一组单选框中只能有一个被选中（checked）。使用以下语法在网页中引入一个单选框：

```
<INPUT type=radio name=radio1 value=radio1 Checked>
```

参数 name 和 value 是必须的，可选参数 Checked 指示初始状态为 ON，也就是被选中状态。

例如，性别有男女之分，一般我们都将他们设计为一组单选框；又如所属的年龄段也只能单选。那么我们就可以这样设计：

```
<HTML>
<HEAD>
<TITLE>单选框</TITLE>
</HEAD>
<BODY>
<FORM name=frmRadio>
<FONT size=2>
您的性别：
<INPUT name=radio1 type=radio value=male Checked >男
<INPUT name=radio1 type=radio value=female>女
<P>
您的年龄：
<INPUT name =radio2 type =radio value =0>20 岁以下
<INPUT name =radio2 type =radio value =1 Checked>20-40 岁
<INPUT name =radio2 type =radio value =3>40-60 岁
<INPUT name =radio2 type =radio value =5>60 岁以上
</FONT>
</FORM>
</BODY>
</HTML>
```

运行效果如图 7.12 所示。

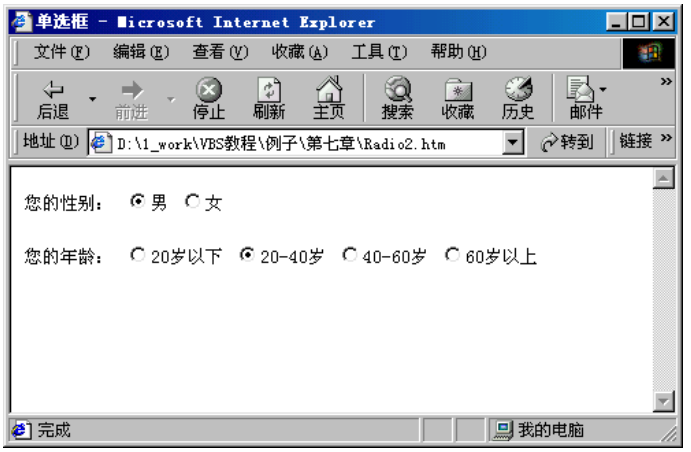

图 7.12　含有性别组和年龄组的单选框组

需要特别指出的是，单选框的 value 属性是必须的，如果不是用 value 属性，则无法将单选框的名称和它的内容对应起来。

单选框也有不少的方法，比较重要的有 Blur、Focus、Select、Click 等，与前面我们介绍过的控件的用法相同。只不过由于单选框主要用来方便用户的输入，使用它的方法的时候相对要少一些。需要注意的是，由于一个单选框组中实际上包含了多个控件，所以当我们在使用它的事件的时候，不能像使用以前许多事件处理的方法：控件名_事件名()，那样来触发单选框的事件。因为同一组中单选框的 name 相同，这样做，过程无法识别究竟是哪一个控件调用该过程，结果就不会执行。请比较以下两个例子。

【例 7.6】　单选框事件不正确的使用方法

```
<HTML>
<HEAD>
<TITLE>单选框</TITLE>
</HEAD>
<BODY>
<FORM name=frmRadio>
<FONT size=2>
您的性别：
<INPUT name=radio1 type =radio value=male Checked >男
<INPUT name =radio1 type =radio value =female >女
<P>
您的年龄：
<INPUT name =radio2 type =radio value =0>20 岁以下
<INPUT name =radio2 type =radio value =1 Checked>20-40 岁
<INPUT name =radio2 type =radio value =3>40-60 岁
<INPUT name =radio2 type=radio value =5>60 岁以上
</FONT>
</FORM>
```

```
</BODY>
</HTML>
<SCRIPT language="VBScript">
<!--
Sub radio1_OnClick()
If document.frmRadio.radio1.VALUE    = "male" Then
    alert ("您是一位男士！")
ElseIf document.frmRadio.radio1.VALUE = "female" Then
    alert ("您是一位女士！")
End If
End Sub
-->
</SCRIPT>
```

看完这段代码，也许你会说，这不很正确吗？前面的例子中我们好多不都这样写的吗？可是当你运行的时候，却不能出现预期的结果，无论你怎么单击这些按钮，它都不会告诉你“ 您是一位男士 !”或“ 您是一位女士 !”。这是因为在以前的例子中，我们处理的控件（如单选按钮或文本框）具有名称 name=radio1 **的只有一个，程序将会准确无误地找到这个控件并执行相应事件的代码。可是在上述代码中有一组（两个）相同名称的单选框控件“** <INPUT name=radio1 type=radio value=male Checked >**男 ”和“** <INPUT name=radio1 type=radio value=female >**女 ”，程序就不知道到底是哪个触发了过程** radio1_OnClick**，于是也就不知道要执行什么样的操作了。正确的方法应该如【例** 7.7**】。**

【例 7.7**】 单选框事件的正确的使用方法**

```
<HTML>
<HEAD>
<TITLE>单选框</TITLE>
</HEAD>
<BODY>
<FORM name=frmRadio>
<FONT size=2>
您的性别：
<INPUT name =radio1 type =radio value=male Checked onclick='msg("male")'>男
<INPUT name =radio1 type =radio value =female onclick='msg("female")'>女
<P>
您的年龄：
<INPUT name =radio2 type =radio value =0>20 岁以下
<INPUT name =radio2 type =radio value =1 Checked>20-40 岁
<INPUT name =radio2 type =radio value =3>40-60 岁
<INPUT name =radio2 type=radio value =5>60 岁以上
</FONT>
</FORM>
</BODY>
```

```
</HTML>
<SCRIPT Language="VBScript">
<!--
Sub msg(a)
If a = "male" Then
    alert ("您是一位男士！")
ElseIf a = "female" Then
    alert ("您是一位女士！")
End If
End Sub
-->
</SCRIPT>
```

本例中，我们在同一组的每一个控件中分别使用了 OnClick 当作它的一个属性，这个属性本身就是一条语句，将相应的参数传给要调用的过程 msg 并使其执行。这样不论你是选择“男”还是“女”都会调用过程 msg，只是参数不同罢了。运行结果如图 7.13 所示。当然，读者可以自己完成将性别组和年龄组结合起来的程序。

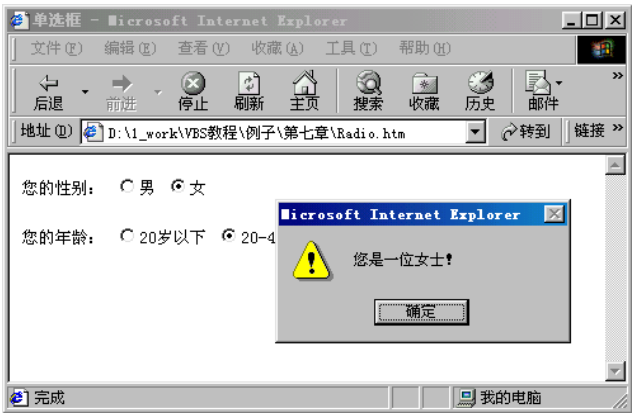

图 7.13　用户选择性别为“女”时的运行结果

另外，在使用单选框控件的属性和方法时，一样存在上述问题，请读者在实际应用中注意。

7.7　复选框控件

复选框允许用户选择多个选项。和单选框一样，同一组的复选框控件应该有相同的名字。使用以下语法在网页中引入一个复选框：

```
<INPUT type=checkbox name=checkbox1 value=checkbox1>
```

例如，一个人的兴趣爱好或者学生选课都可以使用复选框来设计，下面我们以一个人的兴趣爱好为例，设计一个如图 7.14 所示的简单页面。

图 7.14　复选框的使用

【例 7.8】　复选框控件的使用

```
<HTML>
<HEAD>
<TITLE>我的爱好</TITLE>
</HEAD>
<BODY>
<P>
<FONT size=2>    
请选择您的兴趣爱好，也许共同的爱好将使我们成为朋友。</FONT>
</P>
<CENTER>
<FORM name=frmCheckbox method=post>
<P><FONT size=2>
<INPUT type=checkbox name=checkbox1 value=游泳  onclick=Additem("游泳")>游泳
<INPUT type=checkbox name=checkbox1 value=登山  onclick=Additem("登山")>登山<BR>
<INPUT type=checkbox name=checkbox1 value=旅游  onclick=Additem("旅游")>旅游
<INPUT type=checkbox name=checkbox1 value=时装  onclick=Additem("时装")>时装<BR>
<INPUT type=checkbox name=checkbox1 value=购物  onclick=Additem("购物")>购物
<INPUT type=checkbox name=checkbox1 value=阅读  onclick=Additem("阅读")>阅读<BR>
<INPUT type=checkbox name=checkbox1 value=音乐  onclick=Additem("音乐")>音乐
<INPUT type=checkbox name=checkbox1 value=收藏  onclick=Additem("收藏")>收藏<BR>
<INPUT type=checkbox name=checkbox1 value=网络  onclick=Additem("网络")>网络
<INPUT type=checkbox name=checkbox1 value=其它  onclick=Additem("其它")>其它
</FONT>
</P>
<P>
<INPUT name=cmdOK type =button value=就这些了>
</P>
<SCRIPT Language="VBScript">
```

```
<!--
Dim myinterest
Sub Additem(item)
   myinterest = myinterest & item & chr(10)
End Sub

Sub cmdOK_OnClick()
   alert "您的爱好是：" & chr(13) & chr(10) & myinterest
End Sub
-->
</SCRIPT>
</FORM>
</CENTER>
</BODY>
</HTML>
```

本例运行效果参见图 7.14 所示。

这个例子实现的功能是，由用户选择自己的兴趣爱好，按下“就这些了”按钮时，将用户选择的结果显示出来，如图 7.15 和图 7.16 所示。

图 7.15　用户选择了 5 个复选框

图 7.16　用户选择复选框后，单击确定后显示结果

复选框和单选框的属性、方法和事件基本一样，这里我们就不再重复了。需要注意的是，和单选按钮一样，如果引入了一组复选框控件（它们具有相同的名称），在使用它的属性、方法和事件的时候，不能用诸如“控件名.属性名”、“控件名.方法名”和“控件名_事件名()”这样的方法来使用复选框的属性、方法和事件。因为同一组中的复选框的 name 相同，这样做，过程无法识别究竟是哪一个控件在调用，结果就不执行了。

7.8 下拉框控件

我们有时候会觉得单选框或复选框很占空间，如果一个单选框或复选框组中的选项超过 5 项的话，就有些过了。还好，我们可以使用下拉框来替代单选框或复选框。也许你会有所疑问，一个控件如何实现两个控件的功能？这是因为下拉框由一个关键的属性——multiple，这个属性指定了用户是否可以多选。也就是决定它是用来实现单选框的功能还是多选框的功能。通常使用如下格式在一个网页中引入一个下拉框：

```
<SELECT
accesskey = key
align=absbottom | absmiddle | baseline | bottom | left | middle | right | texttop |top
class=classname
datafld=colname
datasrc=#ID
disabled
id=value
lang=language
language=VBSCRPT | VBS | JAVASCRIPT | JSCRIPT
multiple
name=name
size=n
style=cssl-properties
tabindex=n
event=script
>
</SELECT>
```

其中，size 表示显示的项数（缺省是 1）。

我们看到下拉框不同于前面介绍的几种控件，它是由<SELECT></SELECT>标记对定义的。到现在为止，我们还没有看到下拉框的选项究竟在哪里定义。下拉框的选项是由<OPTION></OPTION>标记对插入<SELECT>和</SELECT>中间形成的，<OPTION>标记的格式为：

```
<OPTION
class=calssname
id=value
```

```
language= VBSCRPT | VBS | JAVASCRIPT | JSCRIPT
selected
value=value
event=script
>
</OPTION>
```

其中 selected **属性指出了下拉框的默认值，如果没有指定，就将下拉框的第一项作为默认值。**<OPTION>**中的** value **属性是必须的，如果不用** value **属性，则无法得到该选项的值。**

我们在单选框控件一节中，曾经讲过，使用单选框的属性、方法和事件的时候，如果多个控件的名称相同（为了使它们在同一组内），不能使用诸如“控件名.属性名”、“控件名.方法名”和“控件名_事件名()”这样的方法来使用单选框的属性、方法和事件。因为同一组中的单选框的 name **相同，这样做，过程无法识别究竟是哪一个控件在调用，结果就不会执行。但是，使用下拉框控件则不存在这样的问题，因为它只用了一个** name**。**

我们将 7.4 **节例子中的单选框用下拉框来代替并实现相同的功能。**

【例 7.9**】　下拉框控件的使用**

```
<HTML>
<HEAD>
<TITLE>单选框</TITLE>
</HEAD>
<BODY>
<FORM name=frmRadio>
<FONT size=2>
您的性别：<SELECT name=select1>
<OPTION value=male selected >男
<OPTION value =female>女
</SELECT>
<P>
您的年龄：<SELECT name=select2>
<OPTION value =0>20 岁以下
<OPTION value =1 SELECTED>20-40 岁
<OPTION value =3>40-60 岁
<OPTION value =5>60 岁以上
</SELECT>
</FONT>
</FORM>
</BODY>
</HTML>
<SCRIPT Language="VBScript">
<!--
Sub select1_OnClick()
```

```
If document.frmRadio.select1.VALUE    = "male" Then
    alert ("您是一位男士！")
ElseIf document.frmRadio.select1.VALUE = "female" Then
    alert ("您是一位女士！")
End If
End Sub
-->
</SCRIPT>
```

可以看出，我们所做的改动仅仅是将原来的单选框改成了下拉框，运行一下，太好了，可以运行了，运行结果如图 7.17 和图 7.18 所示。读者朋友们可以同时将年龄选项的程序一块做出来，其中的道理是一样的。

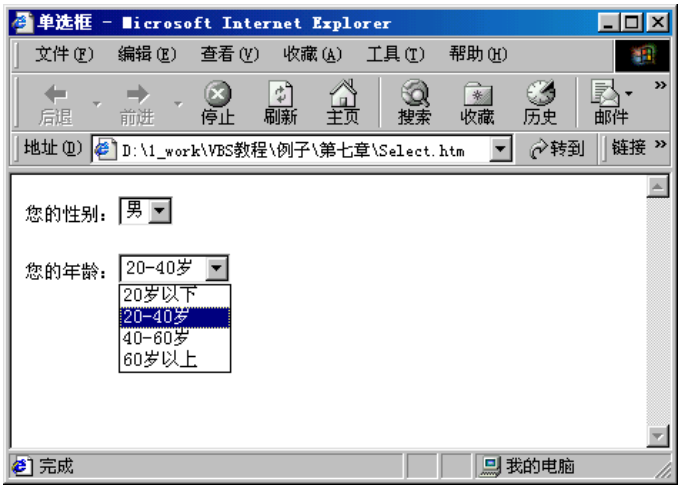

图 7.17　含有下拉框的初始页面

图 7.18　用户选择性别为“女”时的运行结果

下拉框有下面几个重要的属性：

Length **属性**

表示下拉框的项数。

Options **属性**

表示控件的选件数组。

SelectedIndex

表示当前选中项的下标。

下拉框控件有方法 focus **和** blur**，事件** OnBlur、OnFocus、OnClick **和** OnChange **等。在【例** 7.9**】中，我们用到了** OnClick **事件，其实，用** OnChange **事件更合适一些，我们只是为了与【例** 7.6**】比较才坚持用** OnClick **事件的。**OnClick **事件只要用户单击了该控件，就会触发，而** OnChange **事件只有用户选择了不同于上次的选项时才会触发。对于【例** 7.9**】来说，如果用户单击了一下这个下拉框控件，马上就弹起鼠标，在还来不及改变其值的情况下已经触发了** OnClick **事件，**OnClick **事件的处理结果是弹出一个提示框，它强迫用户单击“确定”按钮。对于用户来讲，似乎产生了一个错觉——其值不能改变。实际上，如果用户按下鼠标左键一直拖到要改变的值上再弹起鼠标，就会产生预期的结果了。所以使用** OnChange **事件处理更方便一些。**

下面，我们再举一例来说明下拉框控件的用法。

【例 7.10**】　学习使用下拉框的属性**

```
<HTML>
<HEAD>
<TITLE>选购饮料</TITLE>
</HEAD>
<BODY>
<FORM name=frmSelect>
<P>
<SELECT size="1" name="Drink">
      <OPTION value =01>啤酒</OPTION>
      <OPTION value =02>咖啡</OPTION>
      <OPTION value =03>绿茶</OPTION>
      <OPTION value =04>可乐</OPTION>
      <OPTION value =05>鲜奶</OPTION>
      <OPTION value =06>果汁</OPTION>
</SELECT> 
</P>
<SCRIPT Language="VBScript">
<!--
Sub Drink_OnChange()
i          = document.frmSelect.Drink.selectedIndex
Drinknum = document.frmSelect.Drink.VALUE
```

```
MyDrink    = document.frmSelect.Drink.item(i).text
'item = document.frmSelect.Drink.item(i).text
    alert "您选购的饮料是：" & Drinknum & "    " & MyDrink
End Sub
-->
</SCRIPT>
</FORM>
</BODY>
</HTML>
```

我们使用这个例子来向读者解释如何得到下拉框控件中选项的值和与其相对应的文本的值。

很多时候，显示的文本与其相应的 value 的值并不是相同的，因为在编程过程中，我们使用一些结构化的编码更方便一些，但是显示给用户这些值显然是不合实际的。比如说，在【例 7.10】中，我们需要用一些编码（或纯数字）来唯一标识一种饮料，实际上只需要这些编码就可以完成基本的功能了。但是如果把这些编码让用户看则是很荒唐的事，应该将每一种饮料相对应的中文名称显示给用户。那么，又如何分别取得这两项的值呢？【例 7.10】向我们演示了解决这一问题的方法的执行过程。

可见，使用 value 属性就可以得到下拉框控件选项中的值，其使用方法与其他控件相同。

这里，首先使用 document.frmSelect.Drink.selectedIndex 得到用户所选中的选项的下标，然后用 document.frmSelect.Drink.item(i).text 得到与其对应的文本值，其中 I 表示用户所选中的选项的下标。该例运行结果如图 7.19 所示。

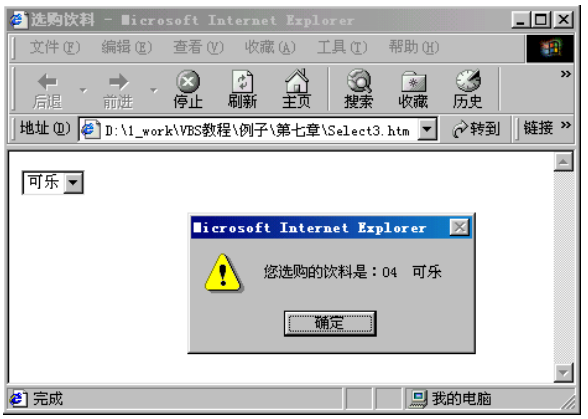

图 7.19 使用下拉框取得选项的值

7.9 隐 藏 控 件

隐藏控件是一种非常有用的控件。所谓隐藏控件，就是用户看不到的控件，但是编程人员经常需要使用它来存储一些信息，并在表单发送的时候将这些隐藏信息传递出去，由

于在 Web 上，不同的页面之间的信息不能共享。也就是说，当用户转到另一个页面上时，原来的页面上的变量存储的值就不能再用了。使用隐藏控件可以实现在不同页面上传递信息。这就是为什么使用隐藏控件的原因。

在网页上引入一个隐藏控件通常用下面的语句：

```
<INPUT type=hidden name=name value=value>
```

下面我们通过一个实例来说明隐藏控件的用法。

【例 7.11】　使用隐藏控件

```
<HTML>
<HEAD>
<SCRIPT Language="VBSCript">
<!--
Sub ChangeVALUE()
   document.frmHidden.recentVALUE.VALUE = document.frmHidden.text1.VALUE
End Sub
Sub DefaultVALUE()
   document.frmHidden.text1.VALUE = "这是一个隐藏控件的例子"
End Sub
Sub undo()
   document.frmHidden.text1.VALUE = document.frmHidden.recentVALUE.VALUE
End Sub
-->
</SCRIPT>
<TITLE>使用隐藏控件</TITLE>
</HEAD>
<BODY>
<P align=center>
<FORM name=frmHidden>
<INPUT name =text1 value =这是一个隐藏控件的例子 OnKeyDown=ChangeValue()>
<INPUT type =hidden name =recentvalue value =这是一个隐藏控件的例子> </P>
<P align=center>
<INPUT name =button1 type =button value =恢复默认值 onclick=DefaultVALUE()>
<INPUT name =button2 type =button value =撤销操作 onclick=undo()>
</P>
</FORM>
</BODY>
</HTML>
```

在这个例子中，用到了 4 个控件，一个文本框控件 text1，在这里可以任意修改其中的值；一个隐藏控件 recentvalue，用来记录最近输入到文本框控件中的值；两个按钮控件，一个用来恢复默认值，另一个用来取消最近一次的操作。这里依靠文本框控件的 OnKeyDown 事件来保持该控件与隐藏控件的一致性，每当用户在文本框中键入了一个字

符，就会触发该文本框的 OnKeyDown 事件，该事件将文本框中当时的值赋给隐藏控件。在这个事件中，用户输入的最后一个字符永远不会被接收，所以隐藏控件中的值总是与文本框中的值差一个按键操作所产生的字符。所以当用户单击“撤销操作”时，就会将隐藏控件中的值赋给文本框。运行结果如图 7.20 和图 7.21 所示。

图 7.20 在文本框中输入信息

图 7.21 撤销操作

7.10 文 件 控 件

在网页中引入一个文件控件使用下面的语句：

```
<INPUT type=file name=name>
```

文件控件可以用来上载文件。

7.11　图 像 控 件

在网页中引入一个图像控件使用下面的语句：

```
<INPUT type=image name=name src=image1.gIf>
```

src 指出图像的位置。单击图像控件将引起表单立即被提交，相当于提交按钮。

7.12　小　结

本章介绍了表单中的控件及其事件，这些控件有很多的属性、方法和事件。我们与用户的交互，基本上都是通过这些控件来完成的。对于常用控件的属性、方法和事件，也作了较为详细的描述，还需要读者多多实践。

VBScript 通常用来编写表单元素的事件处理函数，因此只有掌握了这些表单中的控件，才能设计出功能强大的页面。

练习题

1．如何在一个 HTML 文档中引入表单？表单有哪些重要的属性，分别代表什么含义？

2．表单中都可以包含哪些控件？如何引入？

3．文本区控件与文本框控件有什么区别？表现在什么地方？

4．按钮控件有几种类型？它们分别代表什么含义？

5．请设计一个包含下拉框和复选框的页面，如图 7.22 所示，在它们的单击事件中实现显示选中的条目的值，在按钮的单击事件中联合显示这两个控件的选择信息。

图 7.22　包含下拉框和复选框的页面

第 8 章　在 VBScript 中使用浏览器对象

由于 VBScript 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 VBScript 语言、网页以及浏览器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 VBScript 一样，在 VBScript 语言的实现中也必须考虑到对浏览器的操作和处理。这就是本章所要讨论的内容——浏览器对象。

在 VBScript 编程中，除了可以使用本身提供的固有对象外，还可以使用浏览器对象。浏览器根据目前的配置和当前载入的网页，向 VBScript 提供一些对象。VBScript 程序可以访问这些对象，从而得到当前网页以及浏览器本身的信息。

本章描述在浏览器（IE）中使用对象，并提供这些对象可在 VBScript 编程中直接访问的各种有用的属性和方法。

8.1　窗口对象 Window

Window 对象代表浏览器中打开的窗口，一般来说，当浏览器打开一个 HTML 文档时，就创建了一个 Window 对象；但是，如果文档定义了多个框架（Frame）时，浏览器为原始文档创建一个对象，并为每一个框架创建额外的 Window 对象。这些额外的 Window 对象是原始 Window 对象的子窗口，并受原始窗口中的动作的影响。

我们可以采用多种方法来引用窗口，一种是 Window，另一种是 Self。但是 Window 对象是浏览器对象中其他大部分对象的共同先祖，所以一般在 VBScript 程序中可以隐式地引用 Window 对象。在这种情况下，浏览器中 VBScript 解释程序总是自动地在这样的引用前面加上 Window 对象。例如，可以用 Document 而不是 Window.document 或 Self.document 来引用 Document 对象，用 Confirm()而不是 Window.confirm()或 Self.confirm()来引用 Confirm()方法，浏览器都可以得出正确的结果。但是我们建议采用显示引用的方法，这是一个编程的好习惯，将增强程序的可读性。如果我们想得到一个窗口的父窗口，可以使用 Parent 来引用。

Window 对象有很多属性、方法和事件，下面介绍一些常用的属性、方法和事件。

8.1.1　Window 对象的属性

opener 属性

opener 属性返回打开该窗口的窗口对象。下面的语句显示打开当前窗口的窗口的名字：

```
Msgbox "The Parent window is " & window.opener.name
```

parent 属性

返回当前浏览器对象在对象层次结构中的父对象。对于文件而言，父对象是其所在的窗口；对于使用框架的窗口，父对象是相依的框架集所在的窗口。Parent 属性是只读属性。不能改变。

top 属性

返回对象层次结构中最高的祖先。只读属性。

defaultStatus 属性

返回窗口底部状态区中显示的默认消息。

status 属性

用于设置或获取状态区中的信息。

8.1.2　Window 对象的方法

Alert 方法

显示一个带有一条信息和一个“确定”按钮的警告框。例如：

```
window.alert ("对不起，不能这样操作！")
```

上述语句的执行结果将在浏览器上出现一个如图 8.1 所示的警告框。

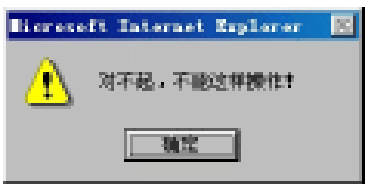

图 8.1　Window 对象的 Alert 方法示例

Confirm 方法

显示一个带有一条消息和一个“确定”按钮、一个“取消”按钮的确认框。例如：

```
window.confirm("您确定要删除此条目吗？")
```

上述语句的执行结果将在浏览器中出现一个如图 8.2 所示的确认框。

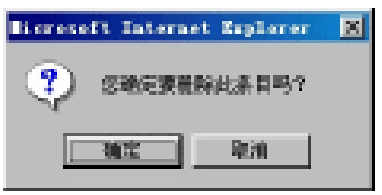

图 8.2　Window 对象的 Confirm 方法示例

实际上，Window 对象的 Alert 方法和 Confirm 方法都可以用函数 Msgbox 代替。Alert 方法相当于 Msgbox ("要显示得消息"，48)

Confirm 方法相当于 Msgbox ("要显示的消息"，65)

所不同的是，用 Window 对象的这两个方法显示，弹出的提示框的 title 部分显示“MicroSoft Internet Explorer”，而用 Msgbox 函数时显示的是“VBScript”。

Prompt 方法

显示一个带有一条消息和一个输入框的提示对话框。例如：

```
window.prompt ("请输入书名：","0")
```

上述语句的执行结果是在浏览器中出现一个如图 8.3 的输入框。

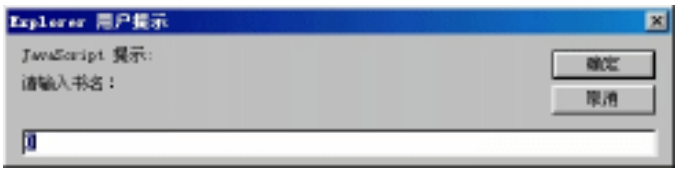

图 8.3 Window 对象的 Prompt 方法示例

该方法的后一个参数可选，表示默认状态下文本框的值，该方法的返回值为用户输入的值。

Open 方法

打开一个新的窗口，并根据所指定的 URL 地址加载文件，如果没有指定 URL，则加载空文档。其语法格式为：

```
window.open (URL,windowName,parameterList)
```

参数说明：

- URL：要打开的页面的地址，其值为字符串。
- winsowName：要打开窗口的名字，如果浏览器按这个名字找到该窗口，则在这个窗口中打开页面，如果未找到，浏览器将生成一个新的窗口。
- parameterList：是一个用逗号分隔的条目列表，用来指定窗口的大小和外观，其中的可能参数是：

  toolbar：指定是否有标准工具栏。
  location：指定是否显示 URL。
  directories：指定是否显示目录按钮。
  status：指定是否有状态条。
  menubar：指定是否有菜单栏。
  scrollbars：指定当文档大于窗口时是否有滚动条。
  resizable：指定窗口是否可以改变大小。
  width：指定以像素为单位的窗口宽度。
  height：指定以像素为单位的窗口高度。
  outerWidth：指定以像素为单位的窗口外部宽度。

outerHeight：指定以像素为单位的窗口外部高度。
left：指定以像素为单位的窗口距屏幕左边的位置。
top：指定以像素为单位的窗口举例屏幕顶部的位置。

除了 width 和 height 采用数值外，其他参数可以用值 1 或 yes 设定为“是”，或者用值 0 或 no 设定为“否”。

例如：

```
Window.open("new.htm","MyWindow","toolbar=no,location=yes,resizable=0,width=200,height=300")
```

Close 方法

用于关闭当前的浏览器窗口。

8.1.3　程序实例

在这个程序实例中，我们调用浏览器的 Window 对象的 Open 方法分别打开一个不含工具栏、菜单栏和状态条的简单窗口和一个包含工具栏、菜单栏和状态条的复杂窗口。需要注意的是，后一个打开的窗口并不是独立于前一个窗口存在的，而是替代前一个窗口的文档。同时，尽管定义了与前一个窗口不同的属性，但这些属性是无效的，仍保留前一个窗口的属性。

【例 8.1】　Window 对象程序实例

新建一个文件保存为 Openwindow.htm，代码清单如下：

```
<HTML>
<HEAD>
<TITLE>window 对象程序实例</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>打开不同的窗口</H1>
<INPUT type=button value="一个简单的小窗口" onclick    ="opensimplewindow()">
<INPUT type=button value="一个完整的窗口" onclick    ="opencomplexwindow()" >
</CENTER>
<SCRIPT Language="VBScript">
Sub opensimplewindow()
      window.open "simple.htm","" ,"height=200,width=300,
            status=no,toolbar=no,menubar=no,location=no"
End Sub

Sub opencomplexwindow()
      window.open "complex.htm","" ,"height=200,width=300,
            status=yes,toolbar=yes,menubar=yes,location=yes"
End Sub
</SCRIPT>
```

```
<P> </P>
</BODY>
</HTML>
```

其页面显示如图 8.4 所示。

图 8.4 打开窗口的主页面

在这个实例中，还要用到两个页面，一个为“一个简单的小窗口”，另一个为“一个完整的小窗口”，我们分别将其命名为 Simple.htm 和 Complex.htm。

Simple.htm

```
<HTML>
<HEAD>
<TITLE>没有状态栏、菜单栏和工具栏的窗口</TITLE>
</HEAD>
<BODY>
<P>这是一个简单的小窗口，没有状态栏、菜单栏和工具栏。</P>
<P> </P>
<INPUT type=button value="关闭" name=button1 onclick    = window.close()>
</BODY>
</HTML>
```

Complex.htm

```
<HTML>
<HEAD>
<TITLE>有状态条、菜单栏和工具栏的窗口</TITLE>
</HEAD>
<BODY>
<P>这个窗口应该包含状态条、菜单栏和工具栏。</P>
```

```
<P><INPUT name=button1 type=button    value=关闭  onclick=window.close()></P>
</BODY>
</HTML>
```

运行 Openwindow.htm，如图 8.4 所示。单击“一个简单的小窗口”，将打开如图 8.5 所示的简单窗口，可以看见它没有状态栏、工具栏和菜单栏。如果单击“一个完整的小窗口”，将打开如图 8.6 所示的窗口，可以看见这是一个包含状态栏、工具栏和菜单栏的完整窗口。

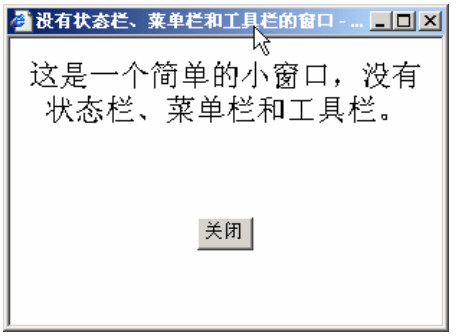

图 8.5　打开的没有状态栏、菜单栏、工具栏的窗口

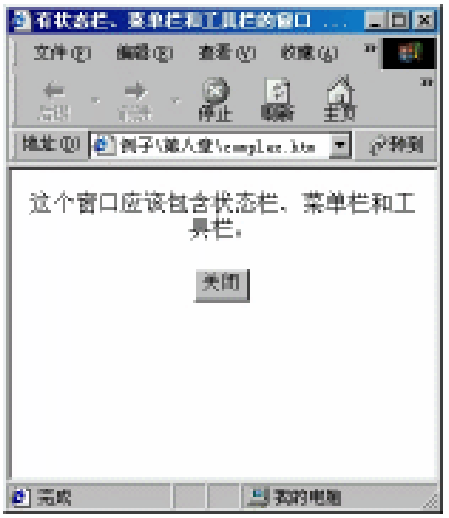

图 8.6　打开的包含状态条、菜单栏、工具栏的完整窗口

但是，如果打开了前一个窗口（比如打开了 Simple.htm），在不关闭它的情况下，要求打开 Complex.htm，则尽管我们要求打开具有状态栏、工具栏、菜单栏的窗口，可是仅仅显示了所要的文件，窗口的属性并没有改变，如图 8.7 所示。

同样，如果先打开的窗口是 Complex.htm，在没有将其关闭的情况下要求打开 Simple.htm，将出现同样的结果，即打开的简单窗口同样包含了状态栏、工具栏和菜单栏，如图 8.8 所示。

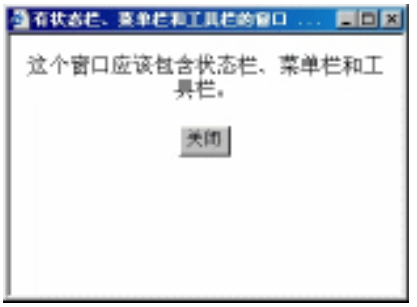

图 8.7 **继承了前一窗口** Simple.htm **的属性**

图 8.8 **继承了前一窗口** Complex.htm **的属性**

为了按照我们所定义的那样来显示后一个窗口，必须先关闭前一个窗口。或者将后一个窗口在新的窗口中打开，即 Open **方法的第二个参数值设一个新的窗口名字。如我们将** VBScript **代码改写为：**

```
<SCRIPT language="VBScript">
Sub opensimplewindow()
    window.open "simple.htm","window1" ,"height=200,width=300,
        status=no,toolbar=no,menubar=no,location=no"
End Sub

Sub opencomplexwindow()
    window.open "complex.htm","window2" ,"height=200,width=300,
        status=yes,toolbar=yes,menubar=yes,location=yes"
End Sub
</SCRIPT>
```

这样，“一个简单的小窗口”和“一个完整的小窗口”将在不同的窗口中打开，不再存在属性继承的关系。

8.2　文档对象 Document

在脚本对象模型中，使用最多的对象恐怕就是文档 Document 对象了。文档对象具有丰富的属性、方法和事件，还包含很多的控件对象及其他对象。

每个 Window 对象都包容一个 Document 对象。Document 对象代表了在给定的浏览器窗口中的 HTML 文件，Document 对象的属性结构对应 HTML 网页中的每一个超级链接、表单等元素的组织结构。Document 对象同样包含 HTML 文档的标题、前景色、背景色、链接颜色以及网页的其他属性。

8.2.1　Document 对象的属性

bgColor 属性
用于设置页面的背景颜色。

fgColor 属性
用于设置页面前景（文字）颜色。

linkColor 属性
用于设置超链接的颜色。

alinkColor 属性
用于设置被激活的超链接的颜色。

vlinkColor 属性
用于设置被访问过的超链接的颜色。

URL 属性
显示当前文件的 URL 地址。

title 属性
通过定义文档的标题，来标识文档。

activeElement 属性
标识了具有焦点的页面元素，通常是指控制元素。

anchors
指出网页中所有的锚点的信息，每一个锚点都是这个数组中的一个元素。

links 属性
指出网页中所有超链接的信息，每一个超链接都是这个数组中的一个元素。

forms 属性
指出网页中所有的表单信息，每一个表单都是这个数组的一个元素。

areas **属性**
指出网页中所有的区域信息，每一个区域都是这个数组的一个元素。

images **属性**
指出网页中所有图像的信息，每一个图像都是这个数组的一个元素。

applets **属性**
指出网页中所有 applet **的信息，每个** applet **都是这个数组中的一个元素。**

layers **属性**
指出网页中所有的层的信息，每一个层都是这个数组中的一个元素。

plugins **属性**
指出网页中所有的 plun_in **的信息，每一个插件都是这个数组中的一个元素。**

LastModified **属性**
上次修改的日期。

cookie **属性**
用来得到和创建 Cookie **信息。**

location **属性**
保存文档的所有 URL **信息。**

提示：对于上述这些属性，当为数组属性时，都有一个相应的属性 Length**。可以通过访问这个属性来获得这个数组的元素个数，这个个数也就是网页中该对象的个数。**

8.2.2 Document **对象的方法**

Clear()**方法**
该方法清除浏览器窗口的内容。

Close()**方法**
该方法关闭缓冲区，将缓冲区现有的内容全部写入网页。这样，上次用 Write **和** Writeln **输出的结果就出现在网页中。**

Open([mimeTYPE]) **方法**
使用指定的 mim **类型打开一个缓冲区，将** Write **和** Writeln **的内容写入缓冲区。**

Write()**方法**
向文档中写入文本。

Writeln()**方法**
向文档中写入文本，并自动换行。

GetSelection()方法

返回当前用户选定的一串字符。

注意：Window 对象和 Document 对象都有 Open 和 Close 方法，在不同的对象中这两种方法实现不同的功能，为了避免混淆，建议显式地使用 Window.open()和 Document.open()来调用相应的方法。

8.2.3　程序实例

【例 8.2】　Document 对象实例 1——设置页面的属性

本例学习使用 Document 对象的属性，用到了 Document 对象的 bgcolor 属性和 fgcolor 属性。

新建一个文件保存为 document1.htm。

```
<HTML>
<HEAD>
<SCRIPT Language="VBScript">
<!--
Sub radioBg_onChange()
   document.bgColor = document.frmColor.radioBg.VALUE
End Sub

Sub radioFg_onChange()
   document.fgColor = document.frmColor.radioFg.VALUE
End Sub
-->
</SCRIPT>
<TITLE>改变页面的设置</TITLE>
</HEAD>
<BODY>
<CENTER>
<P><STRONG>设置页面的属性 </STRONG>
<FORM name=frmColor>
<P align=center>
改变页面的背景颜色
<SELECT name=radioBg>
   <OPTION value ="#ffffff" selected>白色</OPTION>
   <OPTION value ="#ff0000">红色</OPTION>
   <OPTION value ="#008080">绿色</OPTION>
   <OPTION value ="#ffff00">黄色</OPTION>
   <OPTION value ="c0c0c0">灰色</OPTION>
   <OPTION value ="#ff00ff">紫色</OPTION>
   <OPTION value ="#0000ff">蓝色</OPTION>
   <OPTION value ="#000000">黑色</OPTION>
```

```
</SELECT> </P>
<P></P>
<P>改变页面中文字的颜色
<SELECT name=radioFg>
   <OPTION value ="#000000" selected>黑色</OPTION>
   <OPTION value ="#ff0000">红色</OPTION>
   <OPTION value ="#008080">绿色</OPTION>
   <OPTION value ="#ffff00">黄色</OPTION>
   <OPTION value ="c0c0c0">灰色</OPTION>
   <OPTION value ="#ff00ff">紫色</OPTION>
   <OPTION value ="#ffffff">白色</OPTION>
   <OPTION value ="#0000ff">蓝色</OPTION>
</SELECT>
</P>
<P></FORM> 
</P>
</CENTER>
</BODY>
</HTML>
```

运行结果如图 8.9 **所示。**

图 8.9　document **对象示例** 1——**改变页面设置**

这个页面中包含一些简单的文字和两个下拉框列表，所实现的功能是：当用户选中下拉框中的某种颜色时，将页面的背景颜色（或前景颜色，即文字颜色）设为用户选中的这种颜色。

【例 8.3】　Document **对象示例** 2——**使用** Write **方法**

这个实例我们将学习使用 Document **对象的方法。用到了** Document **对象的** Write **方法。**

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

```
<SCRIPT Language="VBScript">
<!--
Dim NewLine
NewLine = Chr(13) & Chr(10)
Sub writeText_onClick()
   tempName = Inputbox("请输入您的姓名：")
   tempCorp = Inputbox ("请输入您的工作单位：")
   tempFavo = Inputbox ("请输入您的兴趣爱好：")
   document.write "<HTML>" & NewLine
   document.write "<HEAD>" & NewLine
   document.write "<TITLE>在页面中添加文本</TITLE>" & NewLine
   document.write "</HEAD>" & NewLine
   document.write "<BODY>" & NewLine
   document.write "亲爱的<FONT size=2 color=red>" & tempName & "</FONT>用户，您好！<P></P>" &
NewLine
   document.write "您的工作单位是：<FONT size=2 color=red>" & tempCorp & "</FONT><P></P>" &
NewLine
   document.write "您的兴趣爱好是：<FONT size=2 color=red>" & tempFavo & "</FONT>" & NewLine
   document.write "</BODY>" & NewLine
   document.write "</HTML>"
End Sub

Sub writeButton_onClick()
   document.write "<HTML>" & NewLine
   document.write "<HEAD>" & NewLine
   document.write "<TITLE>在页面中添加按钮</TITLE>" & NewLine
   document.write "</HEAD>" & NewLine
   document.write "<BODY>" & NewLine
   document.write "<P align=center>" & NewLine
   document.write "<INPUT type=button name=button1 value=""Click Me"" onclick=""alert ('恭喜恭喜，这
是您创建的按钮！')"">" & NewLine
   document.write "</P>" & NewLine
   document.write "</BODY>" & NewLine
   document.write "</HTML>"
end sub

Sub writeImage_onClick()
   tempImage = INPUTbox("请输入图形的路径及文件名：")
   document.write "<HTML>" & NewLine
   document.write "<HEAD>" & NewLine
   document.write "<TITLE>在页面中添加图形</TITLE>" & NewLine
   document.write "</HEAD>" & NewLine
   document.write "<BODY>" & NewLine
```

```
    document.write "<P align=center>" & NewLine
    document.write "<IMG src=" & tempImage &" width=360 border=0>" & NewLine
    document.write "</P>" & NewLine
    document.write "</BODY>" & NewLine
    document.write "</HTML>"
End Sub
-->
</SCRIPT>
<TITLE>使用 document 对象的 write 方法</TITLE>
</HEAD>
<BODY>
<P align=center><STRONG>使用 document 对象的 write 方法</STRONG></P>
<FORM method="post">
<P algin=center>
    <INPUT type ="button" value ="在页面中添加文本" name ="writeText">
    <INPUT type ="button" value ="在页面中添加按钮" name ="writeButton">
    <INPUT type ="button" value ="在页面中添加图形" name ="writeImage">
</P>
</FORM>
</BODY>
</HTML>
```

本主页中只有 3 个按钮，如图 8.10 所示。

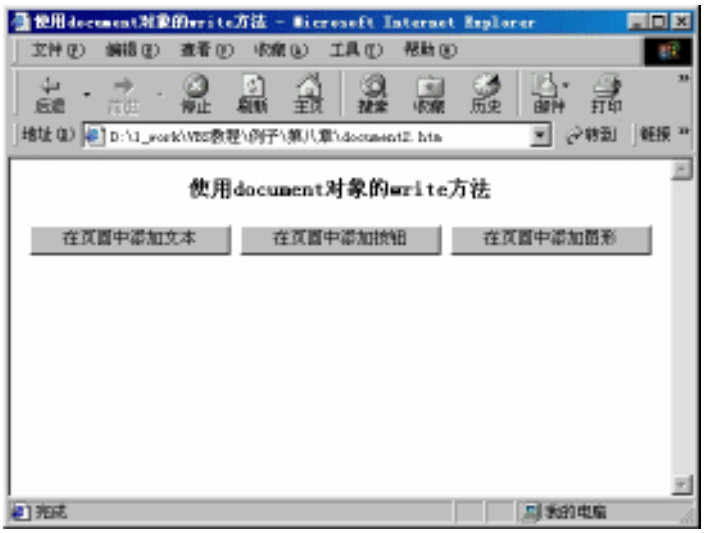

图 8.10　Document 对象实例 2——使用 Write 方法主页面

当用户单击“在页面中添加文本”时，首先用 inputbox 函数提示用户依次输入其姓名、工作单位、兴趣爱好，然后使用 Document 对象的 Write 方法将这些信息以一定的格式输出到页面上。如图 8.11 和图 8.12 所示。

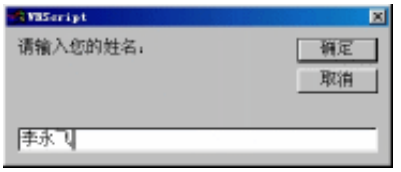

图 8.11　提示用户输入姓名

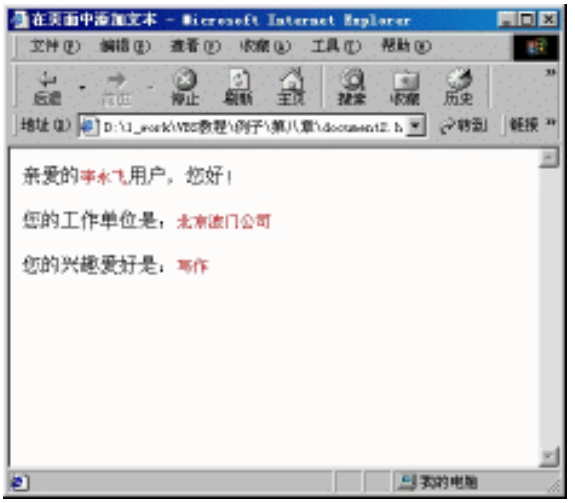

图 8.12　使用 Document 对象的 Write 方法在页面中添加文本

实际上，Document 对象的 Write 方法是建立了一个动态的 HTML 文档，并根据这个文档修改浏览器窗口的现实。其用法为 document.write 加上要书写的 HTML 文档的内容。这部分程序实际上就是把一个完整的 HTML 文件的每一句都用一个 Write 方法输出到页面上。

这部分程序有很多内容涉及到 HTML 标记的内容，不属于本书的重点范围，不作详细描述。

当用户单击“在页面中添加按钮”时，用 Document 对象的 Write 方法输出一个按钮，如图 8.13 所示。

图 8.13　使用 Document 对象的 Write 方法在页面中添加按钮

当用户单击“在页面中添加图形”时，首先用 inputbox 函数提示用户输入要添加的图形的路径，然后用 Document 对象的 Write 方法写出这个图形的引用路径，从而在页面中现实，如图 8.14 和图 8.15 所示。

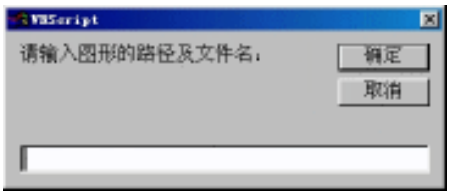

图 8.14 提示用户输入要引用图形的路径及文件名

图 8.15 使用 Document 对象的 Write 方法在页面中添加图形

8.3 历史对象 History

顾名思义，历史对象（History）包含有浏览器窗口的历史。也就是说，历史对象中包含着这个窗口中显示过的每个主页的列表。可以用它来回到曾经浏览过的页面。可以用 Window.history 来使用历史对象。

8.3.1 History 对象的属性

History 对象只有一个 length 属性，它指示现在有多少个 URL 存储在历史列表中。

8.3.2 History 对象的方法

Go（转到）方法

该方法用来转到某个页面，使浏览器窗口中显示用户选择的页面。

Forward（向前翻）方法

该方法用来向前翻页面，使浏览器窗口中显示当前页面的下一个页面。

Back（向后翻）方法

该方法用来向后翻页面，使浏览器窗口中显示当前页面的前一个页面。

8.3.3　程序实例

我们通过下面这个例子学习使用 History 对象的属性和方法。

【例 8.4】　History 对象示例——可切换的窗口

步骤一：制作页面。

本例中涉及到 4 个页面，我们分别命名为 history.htm（主页面）、firstwindow.htm（第一个窗口页面）、second.htm（第二个窗口页面）、thirdwindow.htm（第三个窗口页面）。

history.htm

```
<HTML>
<HEAD>
<TITLE>可切换的窗口</TITLE>
</HEAD>
<BODY>
<P align ="center"><FONT size=2><b>可切换的窗口</B></FONT></P>
<P></P>
<P align ="center">
<FONT size =2>
<A href ="firstWindow.htm">去第一个窗口</A>
</FONT>
</P>
<P align ="center">
<FONT size =2>
<A href ="secondwindow.htm">去第二个窗口</A>
</FONT>
</P>
<P align ="center">
<FONT size =2>
<A href ="thirdwindow.htm">去第三个窗口</A>
</FONT>
</P>

</BODY>
</HTML>
```

history.htm 是首页面分别链接到 3 个不同的窗口。如图 8.16 所示。

图 8.16 History 对象示例——首页面

当单击“去第一个窗口”时，链接到 firstwindow.htm **页面。**
当单击“去第二个窗口”时，链接到 secondwindow.htm **页面。**
当单击“去第三个窗口”时，链接到 thirdwindow.htm **页面。**

firstwindow.htm

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
<!--
Sub cmdBack_onClick()
'使用历史对象的 Back 方法，浏览器窗口中将按窗口中的序号显示当前窗口的下一个
'主页窗口
   history.back
End sub

Sub cmdForward_onClick()
'使用历史对象的 Forward 方法，浏览器窗口中将显示浏览器中显示过的前一个窗口
   history.Forward
End sub

Sub cmdGo_onClick()
'提示用户输入一个小于或等于浏览器中显示过的窗口的总数 history.length 的数
   itemnum = Inputbox("请输入一个小于或等于" & Cstr(history.length) & "的窗口号:")
'如果用户输入的不是数字，进行提示，否则将其转换为整数
If not isNumeric(itemnum) then
        msgbox "您输入的不是数字！"
Else
```

```
        itemnum = Cint(itemnum)
End if
'如果用户输入的数值小于或等于浏览器中显示过的窗口的总数的值，则使用 history
'的 Go 方法，在浏览其中显示序号为 itemnum 的页面
If itemnum <= history.length then
    history.go (itemnum)
End if
End sub
-->
</SCRIPT>
<TITLE>这是第一个窗口</title>
</HEAD>
<BODY>
<P align="center">
<FONT SIZE =2>
<STRONG>这是第一个窗口</STRONG>
</FONT>
</P>
<P align=left>
<FONT size =2>    
    您可以用下面的超链接浏览不同的窗口，也可以使用按钮来切换您刚才浏览过的窗口。
</FONT>
</P>
<P align=center>
<FONT size =2>
<A href="secondwindow.htm">去第二个窗口</A>
</FONT>
</P>
<P ALIGN=center>
<FONT SIZE =2>
<A href="thirdwindow.htm">去第三个窗口</A>
</FONT>
</P>
<P align="center">
<FONT size=2>
<A href="history.htm">返 回 主 页</A>
</FONT>
</P>
<P align=center>
<INPUT name =cmdBack type =button value =前一窗口> 
<INPUT name =cmdForward type =button value =后一窗口>  
<INPUT name =cmdGo type =button value =选择窗口>
</P>
```

```
</BODY>
</BODY>
```

运行结果，如图 8.17 **所示。**

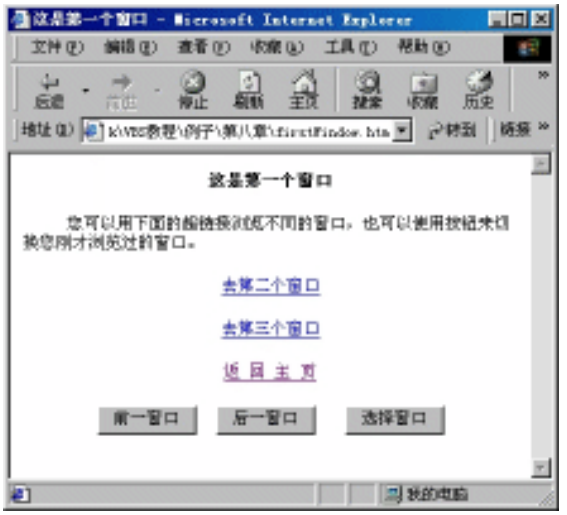

图 8.17　History **对象示例——第一个链接窗口页面**

在 firstwindow.htm **中，当单击“前一窗口”时，使用** History **的** Back **方法转到用户浏览过的前一个页面（如果有的话）；当单击“后一窗口”时，使用** History **的** Forward **方法转到当前页面的下一幅页面（如果有的话）；当单击“选择窗口时”，使用** inputbox **函数提示用户输入一个希望转到的页面号，然后使用** History **的** Go **方法跳转到该页面。**

另外的两个页面 secondwindow.htm **和** thirdwindow.htm **的实现与** firstwindow.htm **大致相同，在这里就不再赘述了，请读者自己完成。**

8.4　位置对象 Location

通过位置对象 Location **能得到当前窗口的** URL **的信息。**

8.4.1　Location 对象的属性

HREF **属性**
加载的 URL**。**

protocol **属性**
加载的 URL **的协议部分。**

host **属性**
加载的 URL **的主机域名或** IP **地址部分。**

port 属性
加载的 URL 的主机端口部分。

pathname 属性
加载的 URL 的路径部分。

search 属性
加载的 URL 的查询条件部分。

hash 属性
加载的 URL 的 hash 部分。

8.4.2　Location 对象的方法

Assign 方法
该方法可以通过设置一个新的 URL 地址，将页面跳转到这个新的 URL 地址上去。比如现在我们要跳转到 www.sohu.com/default.asp 上，可以采用下述方法：locaton.assign("http://www.soho.com/default.asp")

Reload 方法
该方法用来重载当前页面，也就是起到刷新页面的作用。其语法格式为：

```
object.reload([bReloadSource])
```

参数 bReloadSource 是可选的，是一个布尔值，当为 True 时，页面从服务器上重新下载；当为 False 时，页面根据 cache 中的内容刷新。默认值是 False。

Replace 方法：
该方法通过加载指定的 URL 地址的文件从而替换当前文档。这个方法同样也会从浏览器会话历史纪录中去掉当前文件，而 Assign 方法则不会将浏览器会话历史纪录中的当前文件去掉，这是两者的最大区别。亦即，使用 Replace 方法之后，在浏览器上将不可以再使用“回退”、“前进”等方法查看前一个被替换掉的页面。其语法格式如下：

```
object.replace(url)
```

8.4.3　程序实例

在下面的例子中，介绍利用位置对象（Location）制作一个由用户来控制的网页网址的程序。用户可以查看当前网页的网址，也可以查看当前网页网址的各个部分，还可以修改网页的网址，打开一个新的主页。

【例 8.5】　Location 对象示例——控制网页网址

1．新建一个文件保存为 location.htm
代码清单如下：

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

```
<TITLE>控制网页的网址</TITLE>
</HEAD>
<BODY bgColor=lightslategray>
<P align=center><FONT size=2><STRONG>控制网页的网址</STRONG></FONT></P>
<DIV align=center>
<P align=center>
<TABLE border=1 cellPadding=1 cellSpacing=1 width="75%">
<TR>
    <TD>
    <FONT size=2>查阅当前的 URL: </FONT>
    <P aling=center>
    <FONT SIZE=2>
    <INPUT name=cmdUrl type=button value=当前网页的网址>
    </FONT>
    </P>
    </TD>
</TR>
<TR>
    <TD>
    <FONT size =2>查阅 URL 的各个部分 </FONT>
    <P align=center>
    <FONT size=2>
    <INPUT name =cmdHttp type =button value =协议>
    <INPUT name =cmdHost type =button value =主机>
    <INPUT name =cmdHostName type =button value =主机名>
    <INPUT name =cmdPort type =button value =端口>
    <INPUT name =cmdPath type =button value =路径>
    </FONT>
    </P>
    </TD>
</TR>
<TR>
    <TD>
    <FONT size =2 >打开另一个网页 </FONT>
    <P align=center>
    <FONT size =2>
    <INPUT name=newurl type=button value=打开一个新的网页>
    </FONT>
    </P>
    </TD>
</TR>
</TABLE>
</P>
```

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

制作好的主页面如图 8.18 所示。

图 8.18　Location 对象示例——主页面

2．添加程序代码

本例中只使用了几个按钮控件，所以程序代码也全部使用对按钮的单击事件的操作。

步骤一：实现当用户单击“当前网页网址”时，用信息框显示当前网页的网址 URL。我们知道，当前网页的网址就是窗口对象中的位置对象 Location 的 href 属性，所以只需要读出 Location 对象的 href 属性，并用信息框显示即可。

```
Sub cmdUrl_onClick()
    Message = location.HREF
    Msgbox "当前网页的网址是：" & Chr(13) &Chr(10) & Message
End sub
```

步骤二：实现当用户单击“协议”、“主机名”、“端口”或“路径”时，程序在信息框中显示当前网页网址相应的协议、主机名、端口和路径部分。

```
Sub cmdHttp_onClick()
    Message = location.protocol
    Msgbox "所用协议是：" & Chr(13) &Chr(10) & Message
End sub
Sub cmdHost_onClick()
    Message = location.host
    Msgbox "主机名是：" & Chr(13) &Chr(10) & Message
End sub
```

```
Sub cmdPort_onClick()
    Message = location.Port
    Msgbox "端口是：" & Chr(13) &Chr(10) & Message
End sub
Sub cmdPath_onClick()
    Message = location.PathName
    Msgbox "路径是：" & Chr(13) &Chr(10) & Message
End sub
```

步骤三：实现当用户单击“打开一个新的网页”按钮时，将弹出对话框，要求用户输入新的网页网址，然后程序将利用 Location 对象的 href 属性，打开新的网页。方法就是把用户输入的网址赋值给 Location 的 href 属性。

```
Sub cmdNewUrl_onClick()
     NewUrl = inputbox("请输入新网页的网址：")
     Location.href = NewUrl
End sub
```

提示：在当前的窗口中打开新的网页，有多种方法:

- **利用位置对象的** Href **方法。**
- **利用位置对象的其他属性，可以修改网页网址中的任何一个部分或全部**
- **直接使用窗口对象的** navigate **属性，这个属性中包含了浏览器的信息。使用。方法是** navigate + **网址，格式如下：**navigate http://www.e-yao.net**，这样步骤三中的代码可以写成：**

```
Sub cmdNewUrl_onClick()
     NewUrl = inputbox("请输入新网页的网址：")
     Navigate NewUrl
End sub
```

8.5 导航对象 Navigator

由于各种浏览器支持的 HTML **集合不同，同时每一种浏览器自己额外也支持一些特性，不同版本的浏览器是有一定差异的，不同的浏览器不一定完全兼容，这就给网页设计者带来了一定的困难，因为网页设计者不得不在网页中做好准备，以使网页能够适用于不同的浏览器。**

导航对象 Navigator **包含了浏览器的信息。**Navigator **对象使我们能够很方便地了解浏览器的版本信息及其他的相关信息，从而为我们开发网站提供便利。**

8.5.1　Navigator 对象的属性

appCodeName 属性
表示浏览器的代码名称。

appMinorVersion 属性
表示应用程序的版本值。

appName 属性
表示当前浏览器的名称。

appVersion 属性
表示当前浏览器的版本号。

browerLANGUAGE 属性
表示当前浏览器的语言。

connectonSpeed 属性
表示当前会话的连接速度。

cookieEnabled 属性
表示浏览器是否支持客户端 Cookies。

cpuClass 属性
表示 CPU 类型。

onLine 属性
表示系统是否在线。

platForm 属性
表示浏览器运行的平台。

systemLANGUAGE 属性
表示系统运行的默认语言。

userAgent 属性
表示从客户端到服务器的 HTTP 协议的用户代理头部。

userLANGUAGE 属性
表示当前的用户语言。

userProfile 属性
提供允许脚本读取用户信息的请求并执行读动作的方法。

8.5.2　Navigator 对象的方法

JavaEnabled()方法
用户指出在该浏览器中是否可以使用 Java 语言，该方法返回一个布尔值，其语法格式

为：

```
BooleanVar = object.javaEnabled()
```

因为在许多网页中会用到 JavaApplet，所以在使用前一定要注意判断一下浏览器是否支持 Java，这关系到 JavaApplet 能否正常运行。

TaintEnabled 方法

用于判断是否允许将标记过的资料送到任何网络。这种方法可以防止从目录结构或是历史纪录中获得脚本，该方法也返回一个布尔值。

8.5.3 程序实例

在这个例子中，将利用 Navigator 对象的属性看一看我们所使用的浏览器的类型及浏览器的一些参数。

【例 8.6】 Navigator 对象示例——查看浏览器的属性

新建一个文件，命名为 browsertype.htm。

browsertype.htm

```
<HTML>
<HEAD>
<TITLE>关于我的浏览器</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>关于我的浏览器</H1>
</CENTER>
<SCRIPT Language="VBScript">
<!--
document.writeln ("<Per>")
document.writeln ("浏览器的类型：" & window.navigator.appName & "<br>")
document.writeln ("浏览器的版本号：" & window.navigator.appVersion & "<br>")
document.writeln ("浏览器运行平台：" & window.navigator.PlatForm & "<br>")
if window.navigator.javaEnabled then
    document.writeln ("我的浏览器支持 Java。")
else
    document.writeln ("我的浏览器不支持 Java。")
end if
document.writeln ("<Per>")
-->
</SCRIPT>
<P> </P>
</BODY>
</HTML>
```

运行结果如图 8.19 所示。

图 8.19　Navigtor 对象示例

8.6　表单对象 Form

一个文档对象可以包含一个或多个表单对象，这取决于是否在主页中使用了<FORM>标记。如果在主页中使用了表单对象，则 document.forms.length 代表了主页表单对象的数量。所有的 Form 对象组成一个叫 Forms 的数组。Form[0]对应了文档中的第一个 Form 对象，其他依此类推。

在 VBScript 脚本程序中访问特定的第 i 个表单对象，可以采用如下的语法：

```
document.forms(i – 1)
```

也可以通过 HTML 文档中表单的名字来访问每一个 Form 对象：

```
document.form("frm1")
```

要访问某一个表单中的控件元素，可以采用如下的语法：

```
document.forms(i - 1).MyElement  或者  document.forms(“frm1”).MyElement
```

8.6.1　Form 对象的属性

action 属性
包括当前表单的 action 的字符串。

encoding 属性
表单的编码，必须是 MIME 类型，如 text/html。

method 属性

表单提交数据的方法，可能的值是 GET 或 POST。

target 属性

表单处理结果显示的窗口。

8.6.2 Form 对象的方法

Form 对象有一个方法：Submit 方法，引起表单的数据被提交给浏览器。

8.7 小 结

在本章中，我们依次介绍了浏览器的 7 个对象：窗口对象 Window、文档对象 Document、历史对象 History、位置对象 Location、导航对象 Navigator 和表单对象 Form，并通过实例详细讲述了如何使用它们的属性和方法。这些浏览器对象在脚本编程中都是很常用的，请大家掌握。

从下一章开始，我们将学习窗口中的控件和事件。

练习题

1．设计一个小例子，使用 inputbox 函数获取用户的输入，并且用 Document 的 Write 方法输出用户的输入。

2．编写程序：在载入一个页面的同时，打开一个没有工具栏、没有菜单栏、没有状态栏、不可改变大小的宽度为 300、高度为 100 的小窗口。

提示：可以使用 Document 的 OnLoad 事件，如<BODY OnLoad="functionname()">

3．如果想要使当前的窗口向后回退一个页面，可以使用什么方法？

4．使用什么方法可以得到当前文档的 URL 信息以及 URL 的各部分值？

5．使用什么方法可以得到用户浏览器的类型？

6．在 VBScript 脚本程序中如何访问第 3 个表单对象？

第 9 章　错误处理以及调试

任何一个好的程序员在编写程序的同时，除了按用户需求完成所规定的功能外，还必须预测程序执行过程中的各种异常现象。如出现零除、数组越界或者是由于用户的输入引起的错误等等。

实际上，无论是用哪种语言，都不可避免的会出现错误，同样，使用 VBScript 语言编写脚本程序也会出现错误，无论你是多么的小心。况且与一般的高级程序设计语言相比，使用 VBScript 设计程序时，其本身就有一些不利因素使程序更容易产生错误。

- 使用 VBScirpt 设计编写出的代码直到运行前不作语法检查，没有编译程序帮助检查出语法错误；如果用 Notepad 或 Microsoft ActiveX control Pad 进行 VBScript 编码，你应当保证脚本代码中没有语法错误。你也可以使用 Visual Basic 开发环境输入脚本代码，然后剪粘到你的 HTML 页面中，如果有很多代码，这种方法也许是好的，但是一般情况下不值得这样做。
- 一个网页上的 VBScript 代码可能分散在网页的不同部分，可读性不是很好。

由于上述原因，这就要求我们在设计程序的时候，第一要小心，尽量避免错误，第二要在程序中编写错误处理的代码，捕捉出现的异常，进行适当的处理。

9.1　错误的种类

一般来说，在程序中出现的错误分为两种：

1．程序本身出现的错误

如语法错误、逻辑错误等，这种错误必须改正才能使程序正确运行。

引起语法错误的原因很多，比如键入错误引起的各种违背语法规则的错误，如使用 if 语句缺少 end if 等，这样的语法错误，VBScript 引擎不能识别，只有在运行脚本时才会报错。如果对语法规则很熟悉，排除这样的错误相对容易一些，如果代码很复杂，系统报告的出错位置和出错原因可能不准确，这样一来发现真正的错误就有些困难了，这时就需要采用某些跟踪手段缩小错误的范围，以便定位错误。

2．执行中错误

我们之所以把这种错误称为“执行中错误”(Run Time Error)，主要是因为这种错误在程序执行中发生，这种错误有些是无法估计出来的，有些无法用程序本身的逻辑来解决，因为它有可能是由于用户的非法输入造成的结果，也有的是在程序运行之前无法人为预测出的错误。

VBScript 的脚本代码约定，如果在程序执行过程中发生了错误，就要立即停止程序的

执行，并把错误信息报告出来。这样，就无法显示用户要求的页面，而且用来报告错误信息的主页由于没有进行规划也很不整齐。为了不使程序突然停止，不破坏主页的完整性和美观，就必须在程序中加入错误处理的程序段来解决。尤其在较复杂的大型程序中，更是要处处进行错误处理，才能保证程序运行时不会突然停止下来。

9.2 捕 获 错 误

严格的讲，VBScript 中的错误处理功能是比较弱的，当错误发生的时候，它没有办法直接调用错误处理程序，它发生错误的时候并没有产生中断或提示信息的机制。熟悉 Visual Basic 的读者知道，在 Visual Basic 中，错误发生时可以利用 On Error Goto ……语句，使程序直接跳转到 Goto 后边的标号指示的代码处。遗憾的是，VBScript 代码中不支持 On Error Goto……语句，程序不会从发生错误的地方跳转到处理错误的地方。

VBScript 只能利用 On Error Resume Next 和 On Error Goto 0 语句。前一个语句能捕捉到错误发生的事件，但是不能让程序的控制转到设计人员希望的位置。On Error Resume Next 语句的意思是忽略当前产生的错误，继续执行错误的语句之后的语句，或是按照最近一次所调用的过程（该过程含有 On Error Resume Next 语句）中的语句继续运行。这个语句可以不顾运行时错误，继续执行程序，之后你可以在过程内部建立错误处理例程。在调用另一个过程时，On Error Resume Next 语句变为非活动的。所以，如果希望在例程中进行内部错误处理，则应在每一个调用的例程中执行 On Error Resume Next 语句。

下面给出了 VBScript 的错误处理基本框架：

```
Sub MySub()
    On Error Resume Next
    ……
    在线处理
     If Err.Nember = ? then
        ……
     End if
     ……
     ' 报告脚本遇到的上一个错误信息
        Msgbox Err.Description,vbCritical
        ……
     End if
End Sub
```

注意：当使用 On Error Resume Next 语句后，除了最后一个错误，前面发生过的错误都已经被清除了，所以无法知道发生过的所有错误。

如果以后不想捕捉错误，让系统自己处理错误，使用 Error Goto 0 语句。执行这个语句后，程序不再捕捉错误，如果要再回到捕捉错误的状态，就要再加上 On Error Resume Next 语句。

desciption 属性

这个属性返回一个所发生错误的人可以读的懂的解释。错误处理代码应该将这个信息显示给用户。特别是当错误是由用户的输入引起时，让用户知道错误的原因才可能改正错误。

source 属性

返回或设置最初生成错误的对象或应用程序的名称。语法为：

```
object.source [= stringexpression]
```

其中 object 是 Err 对象，stringexpression 是字符串表达式，表示生成错误的应用程序。source 属性指定一个字符串表达式，此表达式通常是导致错误的对象的类名称或程序设计的 ID。当代码无法处理可访问对象产生的错误时，请使用 source 为用户提供信息。例如，如果访问 Microsoft Excel 并且产生一个被零除的错误，Microsoft Excel 将 Err.Number 设置为此错误的错误代码，并将 source 设置为 Excel.Application。注意，如果错误产生于另一个由 Microsoft Excel 调用的对象中，则 Excel 截获此错误并且将 Err.number 设置为它自己的被零除的代码。但是，它保留其他 Err 对象（包括 source），就象由产生错误的对象设置一样。

source 总是含有最初产生错误的对象名称，你可根据所访问对象的错误信息文档，编写处理错误的代码。若错误处理程序失败，则可使用 Err 对象信息为用户描述错误，使用 source 和其他 Err 通知用户最初导致错误的对象、错误描述等等。

在代码中产生错误时，source 是应用程序的程序设计 ID。

下面代码举例说明如何使用 source 属性:

```
On Error Resume Next
Err.Raise 6     ’产生溢出错误
MsgBox ("Error # " & CStr(Err.Number) & " " & Err.Description & Err.Source)
Err.Clear       ’清除错误
```

helpfile 属性

设置或返回帮助文件的完整有效路径。使用语法为：

```
object.helpfile [= contextID]
```

其中 object 为 Err 对象，contextID 是可选项，表示帮助文件的完整有效路径。

如果在 helpfile 中指定帮助文件，则当用户在错误消息对话框中单击帮助按钮（或按 F1 键）时，自动调用此文件。如果 helpcontext 属性包含指定文件的有效上下文 ID，则自动显示该主题。如果未指定 helpfile，则显示 VBScript Help 文件。

```
On Error Resume Next
Dim Msg
Err.Clear
Err.Raise 6      ’产生“溢出”错误
Err.helpfile = "yourHelp.hlp"
```

```
Err.HelpContext = yourContextID
If Err.Number <> 0 Then
    Msg = "按下 F1 键或 Help 查看" & Err.Helpfile & "与下列 HelpContext " & _
    " 有关的主题: " & Err.HelpContext
    MsgBox Msg, , "错误: " & Err.Description, Err.Helpfile, Err.HelpContext
End If
```

HelpContext 属性

设置或返回帮助文件主题的上下文 ID。使用语法为

```
object.HelpContext [= contextID]
```

其中 object 为 Err 对象，contextID 为可选项，表示在帮助文件中帮助主题的有效标识符。

如果在 helpfile 中指定帮助文件，则 helpcontext 属性用于自动显示标识的帮助主题。如果 helpfile 和 helpcontext 都为空，则检查 number 属性。如果它对应于 VBScript 运行时错误值，则使用该错误的 VBScript Help 上下文 ID 。如果 number 属性不对应于 VBScript 错误，则显示 VBScript Help 文件的目录。

下面例子说明如何使用 HelpContext 属性:

```
On Error Resume Next
Dim Msg
Err.Clear
Err.Raise 6       ’发生“溢出”错误
Err.Helpfile = "yourHelp.hlp"
Err.HelpContext = yourContextID
If Err.Number <> 0 Then
    Msg = "按下 F1 键或 Help 查看" & Err.Helpfile & " 与下列 " & _
    " HelpContext 有关的主题: " & Err.HelpContext
    MsgBox Msg, , "错误: " & Err.Description, Err.Helpfile, Err.HelpContext
End If
```

9.3.2　Err 对象的方法

Err 对象有两个方法。

Raise 方法

用于引发特定的错误，Raise 方法的语法为：

```
Object.Raise(number,source,description,helpfile,helpcontext)
```

其中的参数定义如下：

- object：为 Err 对象。
- number：Long 整数子类型，标识错误性质。VBScript 错误（有 VBScript 定义和用户定义两种错误）的范围在 0~65535 之间。

- source：命名最初产生错误的对象或应用程序的 string expression。当为 Automation 对象设置此属性时，请使用窗体 project.class。如未作任何指定，则使用当前 VBScript 项目的程序设计 ID。
- description：描述错误的字符串表达式。如未指定，则检查 number 的值。如可将其映射为 VBScript 运行时错误代码，则将 VBScript 提供的字符串作为 description 使用。如没有与 number 对应的 VBScript 错误，则使用通用错误信息。
- helpfile：Help 文件的完整合法的路径，在该 Help 文件中可找到此错误的帮助信息。如未指定，则 VBScript 将使用 VBScript Help 文件的完整合法的驱动器、路径和文件名。
- helpcontext：上下文 ID，标识 helpfile 中提供错误帮助的主题。如果省略，则使用与 number 属性对应的错误的 VBScript Help 文件上下文 ID（如果存在）。

除了 number 以外的所有参数都是可选项。如果使用 Raise，而不指定某些参数，且 Err 对象的属性设置含有未清除的值，则这些值将成为错误的值。

在 Automation object 中设置错误代码的 number 属性时，请向常数 vbObjectError 添加错误代码编号。例如，要生成错误号 1050，可将 number 属性赋值为 vbObjectError + 1050。

Clear 方法

用于清除 Err 对象中的所有信息。Clear 方法的语法为：

```
Object.Clear
```

在错误处理后，使用 Clear 显式地清除 Err 对象。此操作是必须的，例如使用 On Error Resume Next 延迟错误处理时。在任何时候执行下列语句，VBScript 自动调用 Clear 方法：

```
On Error Resume Next
Exit Sub
Exit Function
```

下面举例说明如何使用 Clear 方法：

```
'发生错误继续下一步
Err.Raise 6    '发生溢出错误
MsgBox ("Error # " & CStr(Err.Number) & " " & Err.Description)
Err.Clear      ' 清除错误
```

9.4 错误处理的例子

本节给出两个在 VBScript 中进行错误处理的例子。第一个例子说明在脚本中处理运行时刻错误，我们将用到前面介绍过的 On Error Resume Next 语句来捕获运行时错误。第二个错误展示了通过 Err 对象的 Raise 方法，使用编程的方法制造运行时错误。

【例 9.1】 捕获运行时错误的例子

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
<!--
Option Explicit
Sub window_onload()
Dim intLength
On Error Resume Next

intLength = -1
Msgbox Right("Hello World!",Cint(intLength))
If Err then
    Msgbox "错误" & err.number & "：" & err.description
    Err.clear
End if

intLength = 6
Msgbox Right("Hello World!",Cint(intLength))
If Err then
    Msgbox "错误" & err.number & "：" & err.description
    Err.clear
Else
    Msgbox "没有发生错误!"
End if
End Sub
-->
</SCRIPT>
<TITLE>捕获运行时错误</TITLE>
</HEAD>
<BODY>
<P> </P>
</BODY>
</HTML>
```

在本例中，使用 Option Explicit **语句强制显式声明变量。接下来是被主窗口调用的** Onload **事件定义，这是本例的执行部分。事件的第一行代码定义了变量** intLength**，用来指明要返回的字符数目。随后是** On Error Resume Next **语句，当发生错误时，执行发生错误后的下一条语句。首先将其置为**-1**，它将作为** Right()**函数的参数。由于表达式** Right(String,-1) **无效，将导致一个运行时错误。**Msgbox **后的语句用来测试是否发生了运行时错误。语句** if err then **等价于** if err.number > 0 then**，**Err **对象的值大于** 0 **时表示发生了运行时错误。**

使用 Err **对象的** number **属性得到错误号（本例中为** 5**），使用** description **属性得到错误描述（此时发生错误为无效的过程或参数），将弹出如图** 9.1 **所示的提示框。**

图 9.1 发生了运行时错误

接着我们使用 Err 对象的 Clear 方法清除 Err 对象，这样可以保证不会受到旧的属性值干扰。

此后，我们将 intLength 的值置为 6，调用同样的函数 Right()，因为此时函数的参数有效，所以不会发生错误。首先弹出提示框显示函数的取值结果，然后执行 else 部分的代码，显示“没有发生错误!”的提示框。如图 9.2 和图 9.3 所示。

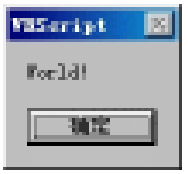

图 9.2 显示函数的取值结果

图 9.3 执行 else 部分的语句

【例 9.2】 使用 Err 对象的 Raise 方法产生一个错误

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
<!--
Option Explicit

Sub cmdDisErrDes_onclick()
Dim blnRaiseError
On Error Resume Next
If not isnumeric(document.frmErr.ErrNumber.value) then
    Msgbox "请输入数字！"
    document.frmErr.ErrNumber.value = ""
    document.frmErr.ErrNumber.onfocus()
Else
    blnRaiseError = Clng(document.frmErr.ErrNumber.value)
    If blnRaiseError then
        Err.Raise blnRaiseError
        Msgbox "Err" & blnRaiseError & "：" & err.description
    Else
        Msgbox "没有错误发生！"
    End If
End If
end sub
```

```
-->
</SCRIPT>
<TITLE>使用 Err 对象的 Raise 方法产生一个错误</TITLE>
</HEAD>
<BODY>

<P>
<FORM name=frmErr>
<INPUT name=ErrNumber>
<INPUT name=cmdDisErrDes type=button value=显示错误描述>
</FORM>
</P>

</BODY>
</HTML>
```

在这个例子中，我们用了两个控件，当用户在文本输入框中输入一个数字并且单击按钮控件时，将制造错误号为用户输入数字的错误，并显示该错误的描述。例如，当用户输入了数字 429，该例运行结果如图 9.4 和图 9.5 所示。

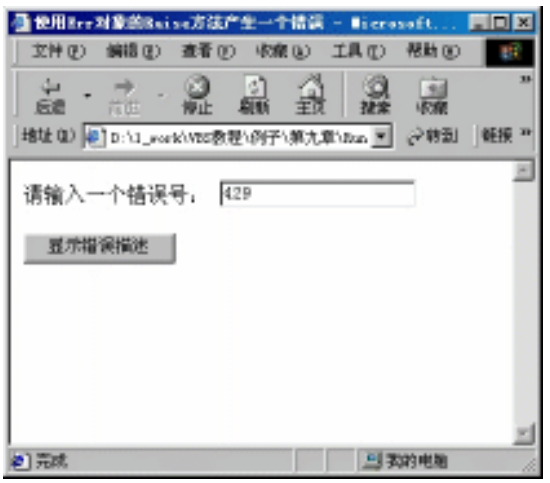

图 9.4　用户输入一个错误号 429

图 9.5　显示错误描述

9.5 常见错误分析

下面我们将初学者容易犯的错误列举出来，以起提醒作用。

（1）拼写错误。谁都会犯这个错误，有效的方法就是使用 Option Explicit 显式地声明变量。

（2）语句不匹配。在 if…then…语句中漏写了 end if。

（3）将两个单词拼作一个单词。如将 end if 写成了 endif

（4）表达式中搭配不当也会产生语法错误。例如下面的代码：

```
document.write "hello" & document.form1.txtname &
```

将会产生一个错误，因为在 VBScript 编译器要求在最后的“ & ”后面有东西。

（5）在使用 document.write 方法向浏览器输出文本的时候，经常会出现“ 语句未结束 ”或“ 无效字符 ”的错误。这是因为如果想要使输出到浏览器上的文本包含引号，写在 document.write 方法中的引号中的文本需要在希望出现引号的地方使用双引号表示。如下面的代码是正确的：

```
document.write "老师对我讲：""这次成绩不错，要继续努力。"""
```

尤其当需要输出的文字很多而且这些文字要用“ & ”号组合起来的时候，很容易犯这个错误，希望大家注意。

（6）类型不匹配。解决这个问题的方法是使用类型转换函数将两个变量的类型变为一致，再进行比较或运算。

（7）未注意 Integer 型数据的数值范围。

（8）调用函数时传入了不正确的参数。

（9）发生死循环。

（10）定义数组时，将定义的“ 元素个数 ”误以为是“ 可用的最大下标值 ”。

（11）将 VBScript 程序漏在 HTML 网页中。当使用<SCRIPT>标记在网页中引入代码时，一定要在代码结束的地方写上</SCRIPT>标记以表明程序代码的结束。

（12）创建对象错误。

VBScript 允许将 ActiveX 控件嵌入 Web 页面。ActiveX 控件的操作同 Visual Basic 应用程序中定制控件类似：属性可以被设定，方法可以被调用，事件可以被用户在页面的交互动作激发。虽然通过编程在 VBScript 脚本中可以掌握控件的全部特征，但同时如果你不能够正确地使用控件也会引起许多错误。

最常见的错误是<HTML>标记<OBJECT>的错误编码，<OBJECT>标记用来将一个 ActiveX 控件插入 Web 页面。由于此标记有许多自身的元素，例如 CODEBASE 和 ID，以及 ActiveX 控件特性的描述，因此设置中很容易出错或漏掉重要的一项。

<OBJECT>标记要求在 CLASSID 项中为控件确定全局唯一类标识符。全局唯一标识符是创建 OLE 控件时赋予 OLE 控件的唯一的字母数字串。它不仅标识了控件也确认使用的

是正确版本的控件。每一个置于 Web 页面上的 ActiveX 控件都必须有这样一个字串。显然，要是敲错了类 ID 中的任何一个字符，你都不会得到需要的 ActiveX 控件。

9.6　避免错误的一些建议

在使用 VBScript 编写程序的时候，养成良好的编程习惯可以帮助我们避免许多粗心大意造成的错误，节省许多测试和排错的时间。建议读者在设计自己的 VBScript 程序时，注意以下几点：

（1）将所有的变量都显式地声明，即在代码中使用 Option Explicit。

跟踪调试中最恼人的错误也许是拼错变量名，而且这种错误一般本人常常觉察不到，在没有发现错拼的变量名之前，代码显然无法正确执行。为了避免这个问题，应当使用 Option Explicit 语句。它会强制你声明脚本中使用的每一个变量。这样，如果你拼写错了变量，脚本执行时它会告知发现未定义的变量。如图 9.6 所示是一个例子。

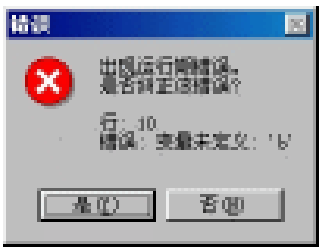

图 9.6　未定义变量错误对话框

注意：在编写代码中，有时我们需要将脚本分为几部分。那么必须在每一个 <SCRIPT>...</SCRIPT>对内加入 Option Explicit 语句。如果你忽略了在某一对中加入 Option Explicit 语句，那么该部分中就不会检查拼错的变量。

（2）尽量使用命名常量，少用具体的常数值。

（3）在进行变量之间的运算时，确保变量之间的相容性。

（4）尽量使用网页设计工具设计 HTML 和 VBScript 代码，减少人工键入。

（5）使用编码规范。编码规范可以使代码易读易懂。使用代码规范会使命名和使用变量直观而不易混淆。

编码规范中最重要的一部分是变量命名规范。表 9.1 给出了 Microsoft 推荐的前缀。

表 9.1　Microsoft 推荐的变量名前缀

数据类型	前缀	例子
布尔（Boolean）	bln	blnHasAccess
字节（Byte）	byt	bytCharacter
日期/时间（Date/Time）	dtm	dtmEndDate
双精度（Double）	bbl	dblTotalCharge

（续表）

数据类型	前缀	例子
错误（Error）	err	errFileError
整型（Integer）	int	intAge
长整型（Long）	lng	lngCount
对象（Object）	obj	objMember
单精度（Single）	sng	sngAverage
字符串（String）	str	strFileName

（6）设计错误捕捉和处理的代码。

9.7 调试程序错误的方法

作为一个脚本程序，并不像某些程序设计语言（例如 C 语言或 Visual Basic）那样，由许多软件能够对它进行调试。相对而言，调试脚本代码的工具是很少的（MicroSoft 公司于 1998 年初推出了 Script Debugger for Internet Explorer 软件，可专门来调试 Internet Explorer 下的脚本代码。该软件可以从 http://www.MicroSoft.com/Workshop 上免费下载）。我们在本节要介绍的是如何利用 VBScript 语言本身的功能对程序进行测试。

1．用消息框 Msgbox 函数输出调试信息

利用 Msgbox 函数，可以在一个特定的点显示一个变量的内容，或者用它来中止一段代码的执行，起到“断点”的作用。这是调试脚本程序经常用到的办法。比如，在编程的时候，如果对某个变量运行过程中的值产生怀疑，就可以利用 Msgbox 函数在程序执行的过程中把这个变量的内容显示出来。或者，如果在程序的某一段暂时停止程序的运行，看看当前页面的样子，就可以在希望停住的地方加入 Msgbox 函数，程序只有在用户单击了某个按钮之后，才能继续运行下去。

2．添加注释来逼近一个错误点

有时候，可能怎么也找不到程序什么地方发生了错误，使用这种方法是很有效的。在怀疑有错的地方注释掉一部分代码，看看错误是否依然发生，如果发生，表明不是这个地方发生的错误，可另找其他原因。如果错误不发生，表明此处出错，修改这段代码。

以上所说的调试办法都是很简单的，但是非常实用。也许，在你的开发过程中，还会发现更多的调试方法。

9.8 小 结

本章介绍了 VBScript 中错误处理的基本知识。主要内容包括错误种类、捕获错误、使用 Err 对象、常见错误分析、避免错误的建议和调试程序错误的方法。

重点需要掌握如何使用 Err 对象来捕获错误。掌握了这些方法可以帮助我们避免很多

错误，对于提高编程效率有很大的帮助。

从下一章开始，我们将学习在 VBScript 中使用 ASP 技术。

练习题

1．VBScript 中的错误分为哪两种？

2．为什么在使用 VBScript 编写脚本代码时容易出错？

3．要想在 VBScript 中捕获错误，使用什么语句？

4．Err 对象的 number 和 desciption 属性的含义是什么？

第 10 章　VBScript 与 ASP

VBScript 作为一种脚本语言，其应用是非常广泛的。在前面的章节中，我们学习的都是客户端的 VBScript 脚本，它们用一对<SCRIPT></SCRIPT>标记括起来插入 HTML 页面。用标记的 language 属性标识脚本语言是 VBScript。客户端的 VBScript 脚本由浏览器的脚本引擎在客户端执行。除了这种在客户机执行的 VBScript 脚本外，Microsoft 又开发了在 Web 服务器端执行的 VBScript 脚本和服务器端的 ActiveX 组件，通常称为 ASP 技术。

ASP 和传统的 HTML 有着本质的区别。由于 ASP 可以相当容易地写出动态交互的网页，尤其是嵌入脚本语言，如 VBScript 语言和 JavaScript 语言，使得网页的数据处理能力和交互能力大大提高，功能更加灵活。我们举个例子，假如你是一个商店仓库的管理员，老板要求你定时地修改网页表格中的商品价格和库存，如果使用的还是单纯的 HTML 的话，你会发现要修改如此大的数据量是一件多么可怕的事情！现在有了 ASP，我们只需要一些简单的脚本语言就可以解决这个问题。例如，你可以使用脚本语言方便地对数据库中的数据进行检索和更改。如果能够把 ActiveX 组件加入到网页中，就能够处理更多更复杂的任务，可以适应各种不同的需求。

ASP 与客户端 VBScript 脚本和 ActiveX 控件比较，VBScript 的语法、对象的属性、方法和事件的用法等有相似之处。但是服务器的对象模型要复杂的多，服务器端编程涉及的知识更多，技术也更复杂。本章将介绍这方面的最基础最重要的知识。

10.1　ASP 简介

10.1.1　ASP 的特点

ASP 是 Microsoft Active Server Pages 的缩写，意为“活动服务器网页”。它是服务器端的脚本编写环境。利用它可以产生和运行动态的、交互的、高性能的 Web 服务应用程序。主要性能是能够把 HTML 脚本组件等有机地结合在一起，形成一个能够在服务器上运行的应用程序，并把按用户要求制作的标准 HTML 页面送给客户端浏览器。

ASP 是从大家所熟悉的 CGI 和 IDC 技术上发展而来的，它在性能上很像传统的 CGI 程序。ASP 页面首先由 ASP 解释器解释执行，然后将产生的 HTML 结果传送给客户端浏览器。因此，当用户浏览 ASP 页面时，浏览器所需解释执行的仅仅是服务器端的执行结果——HTML 文档，而 HTML 文档能够广泛地被各种浏览器所识别，因此网页性能基本不受客户所用浏览器的影响。

从软件的技术层面看，ASP 具有如下特点：

- 无须编译。ASP 脚本集成于 HTML 当中，容易生成，无须编译或链接即可直接执

行。

- 独立于浏览器。ASP 脚本是在站点服务器端执行的，用户端的浏览器不需要支持它，而传送到客户浏览器的只是 ASP 执行结果所生成的常规 HTML 代码，这样可以保证页面开发者的程序代码不会被他人盗取。
- 面向对象。ASP 提供了一些内置对象。利用这些内置对象，可以使脚本功能更加强大。这些对象允许从浏览器中接受和发送信息。例如，利用 Request 对象可以接受 HTML 表单中的信息，并在脚本中响应。
- 与任何 ActiveX Scripting 语言兼容。

从应用的层面上来看，ASP 具有如下功能：

- 处理浏览器传送到站点服务器的表单输入。
- 访问和编辑服务器端的数据库内容。ASP 可以和诸如 SQL Server、Oracle 这样的数据库挂接。利用一些特别的对象集合，如 ActiveX Data Object（ADO），就可以在 ASP 中使用 SQL 语言，使用浏览器即可输入更新和删除站点服务器的数据库数据。
- 可以实现在多个主页之间的共享信息，以开发复杂的商务站点应用程序。

ASP 程序其实是以扩展名为.asp 的纯文本格式保存在 Web 服务器上的。只要将.asp 程序放在 Web 服务器的虚拟目录下（该目录需要有可执行权限），就可以通过 WWW 浏览器（如 Internet Explorer）来访问该 ASP 程序了。在 ASP 中，默认的脚本语言是 VBScript。读者可以根据各自的爱好，设置不同的默认脚本语言，如 JavaScript。

10.1.2　ASP 的环境要求

编辑和运行 ASP 文件需要一定的环境条件，用户必须提供这些条件才可建立 ASP 的开发环境。要运行 ASP 程序，必须在 Windows NT 服务器或 Windows 2000 服务器中安装 Internet Information Server（Internet 信息服务器，简称 IIS），或在 Windows 95/98 操作系统中安装 Personal Web Server（个人 Web 服务器，简称 PWS）。我们将在下面两节讲述如何建立 ASP 的开发环境。

运行 ASP 的硬件要求并不高，下面是推荐的硬件配置：

- CPU。只要能运行相应的操作系统（Windows 95/98，Windows NT/2000 Server）即可。我们建议使用 Pentium 级的芯片，否则速度会慢的令人难以接受。
- 内存。至少要有 32MB，如果运行 Windows NT/2000，则至少要 64MB。当然内存越大越好。
- 硬盘空间。至少要 40MB。建议安装相关的帮助文件，这样需要约 100MB 的硬盘空间。

10.1.3　ASP 的运行机制

在基于 HTTP 的网络传送过程中，最重要的 3 个实体是 Web 客户机、Web 服务器和网络。其中 Web 客户机和 Web 服务器的信息交换是通过客户端浏览器与 Web 服务器之间的

通信来进行的。Web 浏览器与 Web 服务器之间的通信机制如图 10.1 所示。

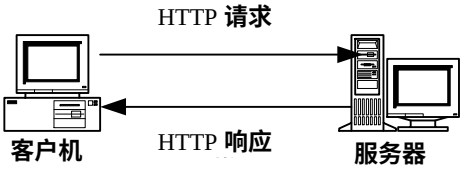

图 10.1 Web 浏览器与服务器之间的通信机制

为了理解 ASP 的运行机制，可以将 ASP 的工作分成下面几个过程：

（1）在浏览器的网址栏中写入 ASP 文件名称，并回车触发这个 ASP 请求。
（2）浏览器将这个 ASP 的请求发送给 Web 服务器（例如 IIS）。
（3）Web 服务器接收这个请求，并根据其.asp 后缀判断这是 ASP 请求。
（4）Web 服务器从硬盘或者内存中接收正确的 ASP 文件。
（5）Web 服务器将这个文件发送到一个叫做 ASP.DLL 的特定文件中。
（6）ASP 文件从头至尾执行并根据要求生成响应的静态网页。
（7）网页被送回浏览器，被浏览器解释执行并显示在浏览器上。

我们用图 10.2 来表示 ASP 的工作过程。

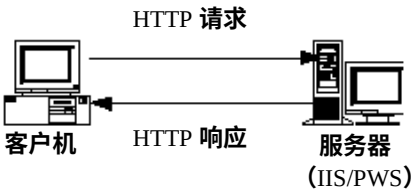

图 10.2 ASP 的工作过程

10.2 PWS 的安装及设置

PWS 是一个 Web 服务器,可用于需要规模向下的 IIS 做内部应用或开发的公司和个人。PWS 不同于 IIS，它只能容纳一个 Web 站点，PWS 的优点在于，它可以在廉价的操作系统如 Windows 95/98/NT 上运行。

10.2.1 在 Windows 98 上安装 PWS

在 Windows 98 上安装 PWS，只需要有 Windows 98 操作系统的光盘就可以了，因为在这张光盘上已经包含了 PWS。

注意：如果需要在 Windows 95 上安装 PWS，则必须从包含有 Windows 95 版的 Windows NT 4.0 Option Pack 上安装。

假设你的计算机上已经安装了 Windows 98 操作系统，请按照下面的步骤安装 PWS。

（1）将 Windows 98 光盘放入光驱中。

（2）找到目录“ add-ons\ pws\ ”。

（3）运行 Setup.exe，弹出如图 10.3 所示的 Personal Web Server Setup（个人 Web 服务器安装）对话框，单击“ 下一步 ”按钮。

图 10.3　Personal Web Server Setup 窗口

（4）在出现的如图 10.4 所示的对话框中有 3 个安装方式可供选择：“ 最小安装 ”、“ 典型安装 ”和“ 自定义安装 ”。前两种都是比较简单的安装，我们以“ 自定义安装 ”方式为例进行说明。

图 10.4　选择安装方式

（5）在选择“ 自定义安装 ”并单击“ 下一步 ”按钮后，将出现如图 10.5 所示的组件列表对话框让用户选择要安装的组件。

（6）单击选中“ 组件 ”列表中的“ Microsoft Data Access Components 1.5 ”选项，然后

单击“显示子组件”按钮，从出现的对话框中选取所有的子组件，以及每个子组件的所有子组件等。出现如图 10.6 所示的对话框。

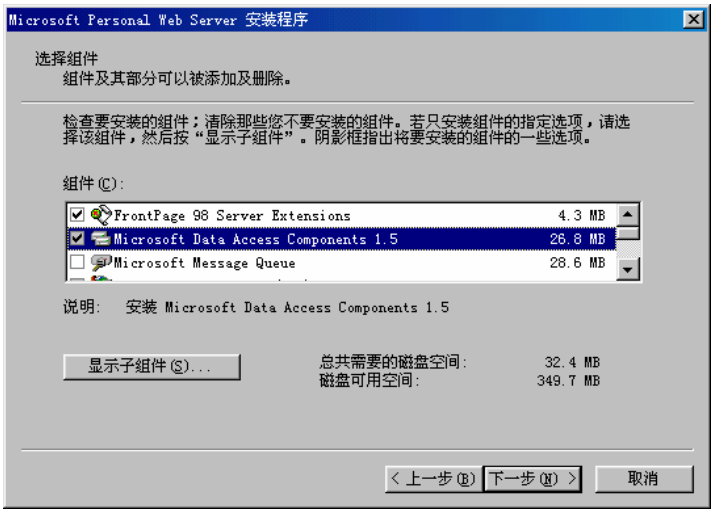

图 10.5 选择安装组件

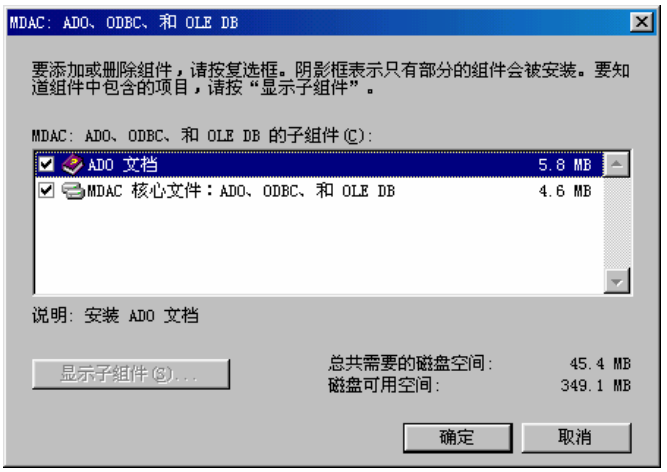

图 10.6 选择安装 Microsoft Data Access Components 1.5 的子组件

（7）采用相同的方法改变“组件”列表中“Personal Web Server(PWS)”选项的默认值，选中所有子组件。然后单击“下一步”按钮。出现如图 10.7 所示的对话框。

（8）安装向导将提示用户进行 Web 发布主目录的设置。在如图 10.8 所示的对话框中。如果 C 盘有足够空间，一般使用默认值，如果必要，也可以在“WWW 服务”文本框中键入需要的 Web 发布主目录的物理路径。然后单击“下一步”按钮。

（9）系统将出现让用户设置 MTS 安装文件夹的画面。一般只要接受系统的默认配置就可以了。单击“下一步”按钮，系统开始安装 PWS。

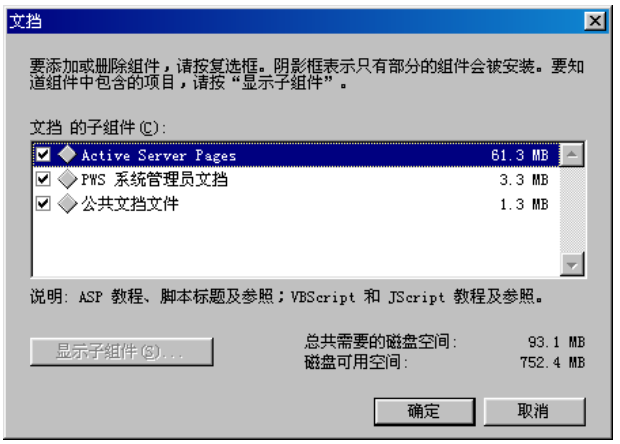

图 10.7　选择安装 Personal Web Server 的子组件

图 10.8　设置 Web 发布主目录

（10）安装完成后，重新启动系统。

系统重新启动后，在桌面上多了一个“发布”图标，程序>Internet Explorer **组中多了** Personal Web Server **组，如果** Web Server **已经启动，任务栏右边会出现** Personal Web Server **图标，使用它可以控制** PWS**。**

10.2.2　PWS 的设置

对于一般的用户来说，PWS **的设置包括两方面的内容：一是设置** IP **地址，一是设置目录属性。下面我们进行详细的说明。**

1．设置 IP **地址的步骤**

（1）打开“控制面板”，双击“网络”选项，打开“网络”对话框。如图 10.9 **所示。**

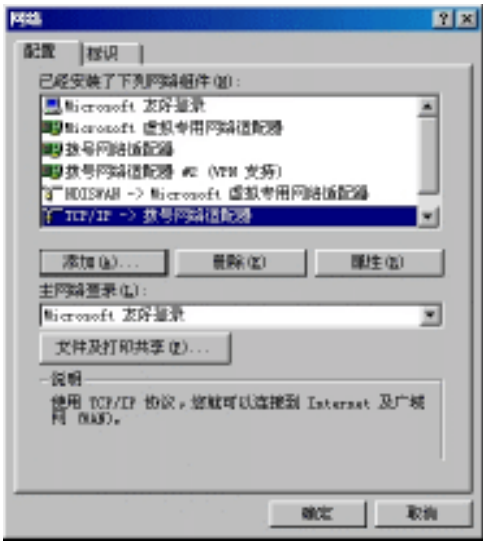

图 10.9 “网络”对话框

（2）选取“配置”选项卡。

（3）在协议列表中，选择“TCP/IP”协议，单击“属性”按钮，打开“TCP/IP 属性”对话框，如图 10.10 所示。

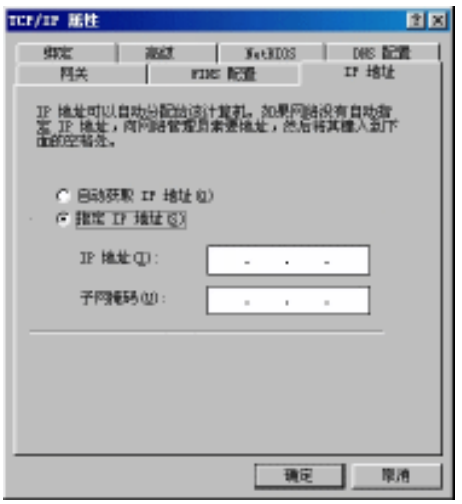

图 10.10 “TCP/IP 属性”对话框

（4）在出现的“IP 地址”选项卡中，可以看到有两种 IP 地址设置方式：“自动获取 IP 地址”和“指定 IP 地址”。我们选择“指定 IP 地址”。如果要建立一个 Internet 站点，一定要先向 ISP 申请一个全球唯一的 IP 地址。但如果所建立的只是公司的局域网或自己家里使用的机器，则 IP 地址可以自己设定，不必申请。

既然是自己使用的 IP 地址，就没有什么限制了，可以设置自己喜欢的 IP 地址，不过基于某种理由，有些 IP 地址是不可以设置的，这些 IP 地址包括 0.X.Y.Z、127.X.Y.Z、224.X.Y.Z~225.X.Y.Z。子网掩码最普遍的设置是 255.255.255.0，所代表的意义是同一局域

网的不同机器要互通，则 IP 地址的前三位码必须相同，最后一位必须相异。

（5）单击“确定”按钮，返回“网络”对话框。

（6）选择“标识”选项卡，如图 10.11 所示。在“计算机名”文本框中输入计算机的名称，单击“确定”按钮。

图 10.11　设置计算机名称

所设计算机名称用于内部识别，当进行浏览器与服务器的测试时，还可以用于 URL 定位。

（7）重新启动计算机，使上述所作的设置生效。

2．设置目录属性

（1）选择“开始”>“程序”>“Internet Explorer”>“Personal Web Server”>“个人 Web 管理器”，或者双击任务栏中的 Personal Web Server 图标，启动个人 Web 管理器，打开它的工作窗口，如图 10.12 所示。

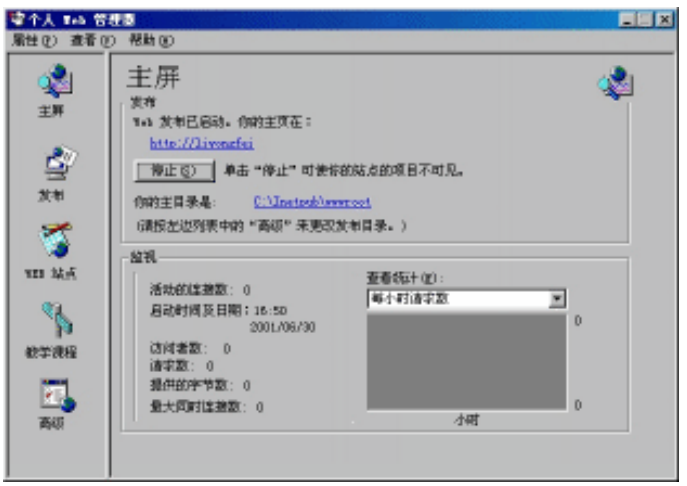

图 10.12　个人 Web 管理器的主窗口

（2）单击“高级”图标，在右边窗格中将出现高级选项窗口，如图 10.13 所示，在这里可以设置主目录的属性。

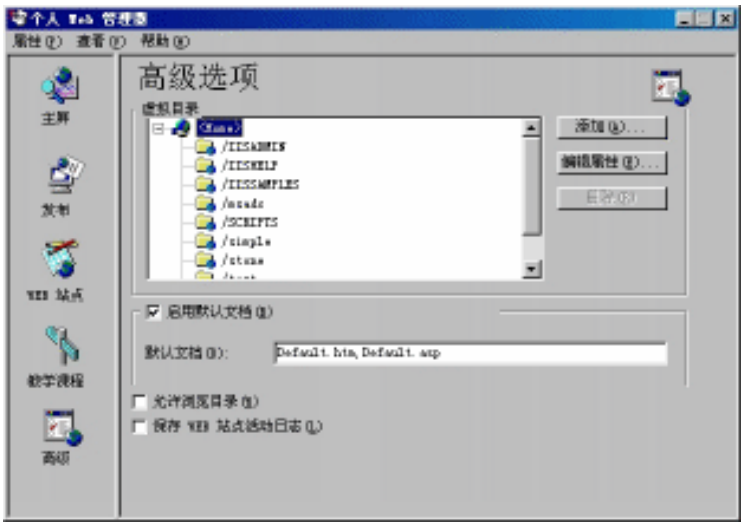

图 10.13 个人 Web 管理器高级选项

（3）设置虚拟目录。单击“添加”按钮，打开“添加目录”对话框，如图 10.14 所示。

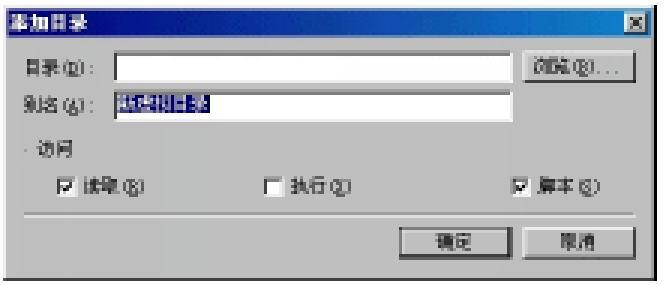

图 10.14 “添加目录”对话框

（4）在“目录”框中输入添加目录对应的物理路径，也可以通过单击“浏览”按钮，从打开的对话框中选择需要的目录，如图 10.15 所示。

（5）在“添加目录”对话框中的“访问”选项组中有 3 个选项，用于设置用户对 ASP 文件的访问权限。“读取”使用户有权查看网页的内容；“执行”使用户有权执行 ASP 编写的网页文件；“脚本”用于设定可以执行的脚本语句。要赋予用户相应的权限，只要将其复选框选中即可。之后单击“确定”按钮，返回个人 Web 管理器的“高级”选项对话框。

（6）设置默认文档。在个人 Web 管理的“高级选项”对话框，选取“启用默认文档”复选框表示如果用户请求页面时，只输入了网页的网址和目录，而没有指定具体的哪一个文件，浏览器将加载默认文档。一般情况下，默认文档为 default.htm、default.asp 或 index.htm。当然，你可以在“默认文档”的文本框中输入其他的默认文档。如果文档有多个，用逗号将它们隔开。服务器在接到没有指定文件名的 URL 请求时，将按次序搜寻本目录下的默认

文档，将第一个找到的默认文档传送到浏览器。

图 10.15　选择添加目录对应的文件夹

（7）允许浏览目录。在个人 Web 管理器“高级选项”对话框，选取“允许浏览目录”复选框表示当没有默认文档时（没有启用默认文档或找不到指定的默认文档），将返回该目录下所有子目录的文件的超文本列表。为了安全起见，建议不要使用此选项。

（8）保存 Web 站点活动日志。在个人 Web 管理器“高级选项”对话框，选取“保存 Web 站点活动日志”复选框，表示可建立日志文件来保存访问网站的活动情况。

完成这些设置后，用户就可以正式的使用 PWS 来发布站点并在浏览器中访问该站点了。至此，ASP 的开发环境也建立起来了。

提示：虚拟目录指不必位于主目录的目录。事实上，它可以驻留在另一个磁盘或计算机上，但可以通过客户浏览器通过 Internet 访问，就像它位于主目录中一样。

10.3　IIS 的安装及设置

IIS 的全称是 Internet Information Server，是集成于 Windows NT Server 或 Windows 2000 的 Web 服务器，利用它可以发布信息到 Web。PWS 上只能容纳一个 Web 站点，为了在相同的计算机上容纳多个 Web 站点，需要使用 IIS。

随着 Windows 2000 的推出，已有取代 Windows NT 的趋势，所以有关在 Windows NT 上安装 IIS 我们就不多说了，有一点注意的是，Windows NT 中所附带的 IIS，属于 Windows NT 4 Option Pack 的一部分，需要单独安装，因此，必须安装 Option Pack 才能运行 IIS。下面我们主要介绍在 Windows 2000 Server 上安装并设置 IIS 5.0 的过程。

10.3.1　在 Windows 2000 Server 上安装 IIS 5.0

IIS 5.0 在默认情况下安装在 Windows 2000 Server 中。使用控制面板中的“添加/删除程序”可以删除 IIS 或选择其他组件。

（1）单击“开始”，指向“设置”，单击“控制面板”，然后再启动“添加/删除程序”。出现如图 10.16 所示的画面。

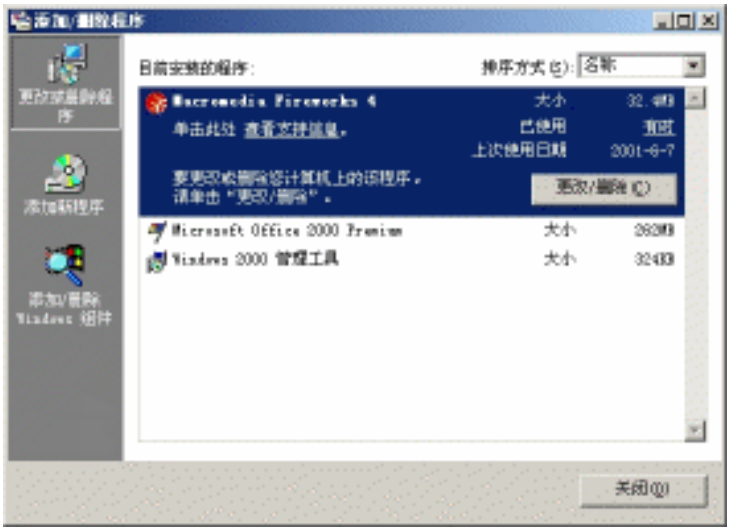

图 10.16　在控制面板中选择添加/删除程序

（2）选择“添加/删除 Windows 组件”，将出现 Windows 组件向导的画面，如图 10.17 所示，按照屏幕提示安装、删除或添加 IIS 组件。

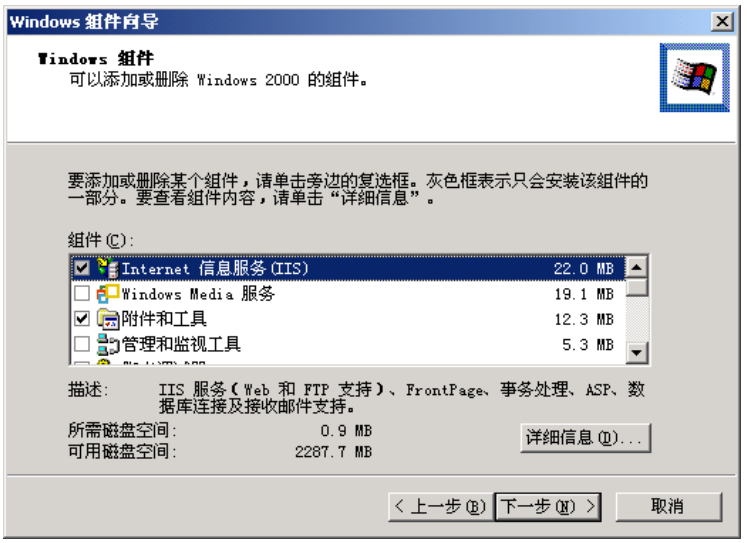

图 10.17　Windows 组件向导

（3）我们看到，在这些组件中包含一项叫做“Internet 信息服务（IIS）”，选中它，然后单击“详细信息”按钮，可以添加或删除其中的子组件，如图 10.18 所示。

（4）在如图 10.18 显示的 IIS 5.0 子组件中添加需要的组件或删除不需要的组件，单击“确定”按钮。IIS 5.0 的安装就完成了。

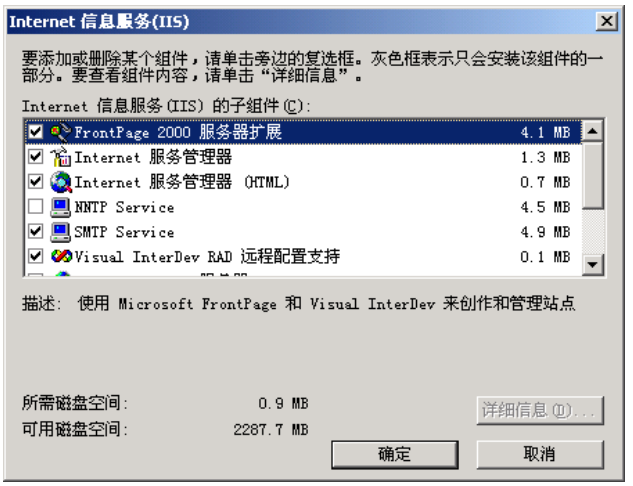

图 10.18　添加或删除 IIS 5.0 的子组件

10.3.2　IIS 5.0 的设置

完成了 IIS 5.0 的安装，就可以建立用于发布 Web 站点的 WWW 服务器了。WWW 服务器建设的核心是建立 Web 站点，包括 Web 站点设置、主页文件及目录设置、目录安全设置和错误信息设置等。具体操作如下。

1．创建一个新 Web 站点

首先我们建立一个新站点。事实上，一个 Web 站点仅仅是 IIS 服务器上的一个目录，并且该目录的访问权限由 IIS 控制。安装好 IIS 之后，它将在系统中自动建立一个默认 Web 站点，用户可以直接使用这个站点发布网页，也可以另外建立其他的站点使用。

（1）单击“开始”>“程序”>“管理工具”>“Internet 服务管理器”，打开管理控制台，如图 10.19 所示。

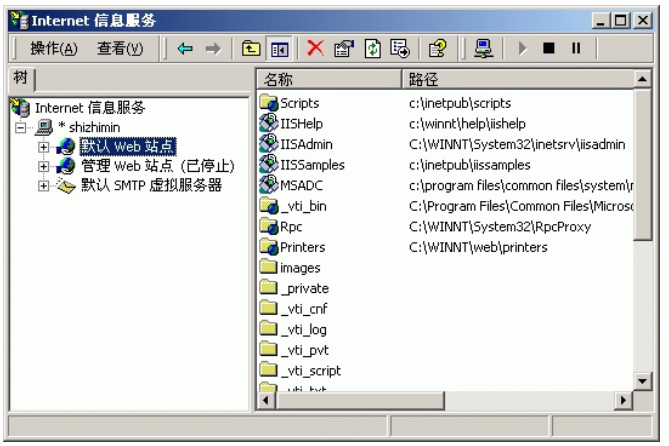

图 10.19　Internet 信息服务管理控制台

（2）选中该 IIS 服务器并单击鼠标右键，从快捷键菜单中选择“新建”>“Web 站点”，如图 10.20 所示。

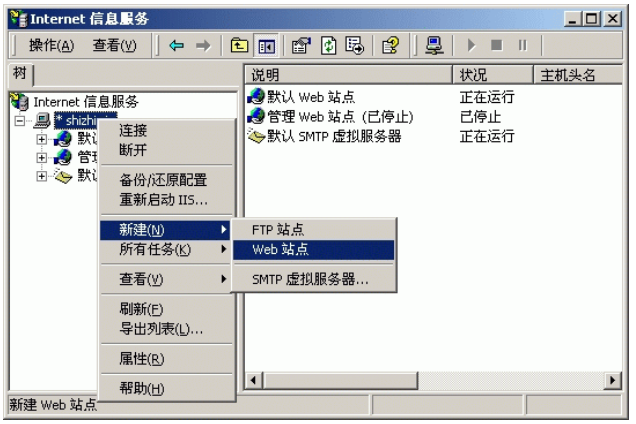

图 10.20 选择新建 Web 站点命令

（3）IIS 启动“Web 站点创建向导”，如图 10.21 所示。

图 10.21 Web 站点创建向导

（4）单击“下一步”按钮，在出现的如图 10.22 所示的对话框中，为将要建立的站点输入一个名字，比如“asptest”，然后单击“下一步”按钮。

（5）在出现的如图 10.23 所示的对话框中，设置 IP 地址和 TCP 端口号，从“输入 Web 站点使用的 IP 地址”下拉列表框中选取一个可以使用的 IP 地址。设置想要使用的 TCP 端口号，如 80。然后单击“下一步”按钮。

（6）这时，出现如图 10.24 所示的对话框，用来设置 Web 站点的主目录，就是把含有 Web 内容的文档的目录路径输入到“路径”文本框中，那么用户通过这个站点的名字就可以访问到这个目录中的内容了，而不必指出在本地的绝对路径。如果希望所有 Internet 用户都能访问到这一站点，请确保选中“允许匿名访问此 Web 站点（A）”复选框，单击“下一步”按钮。

图 10.22　输入 Web 站点的名字

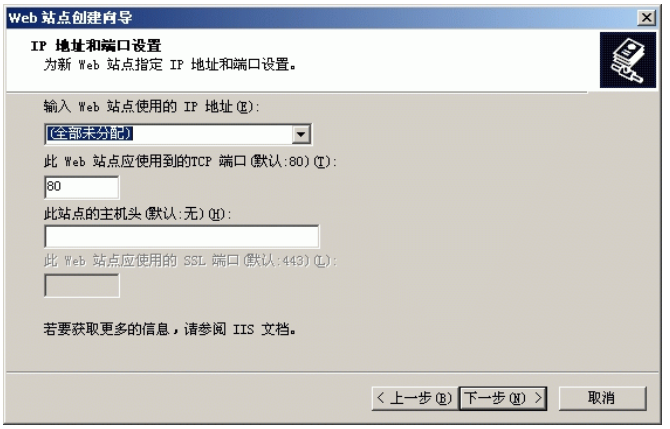

图 10.23　设置 Web 站点的 IP 地址和端口

图 10.24　设置 Web 主目录

（7）在接下来出现的如图 10.25 所示的对话框中，为该 Web 站点设置访问权限。单击“下一步”按钮，即完成 Web 站点的建立。

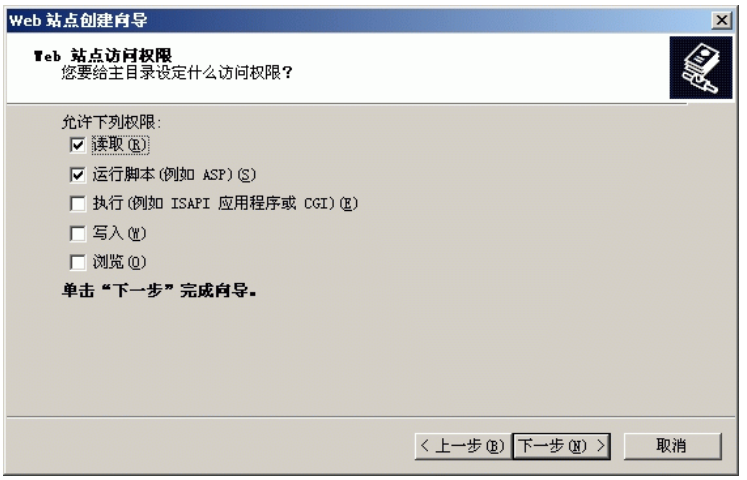

图 10.25 设置 Web 站点的访问权限

（8）现在，我们可以看到在 Internet 信息服务管理控制台中多了一个叫做“asptest”的 Web 站点，如图 10.26 所示。

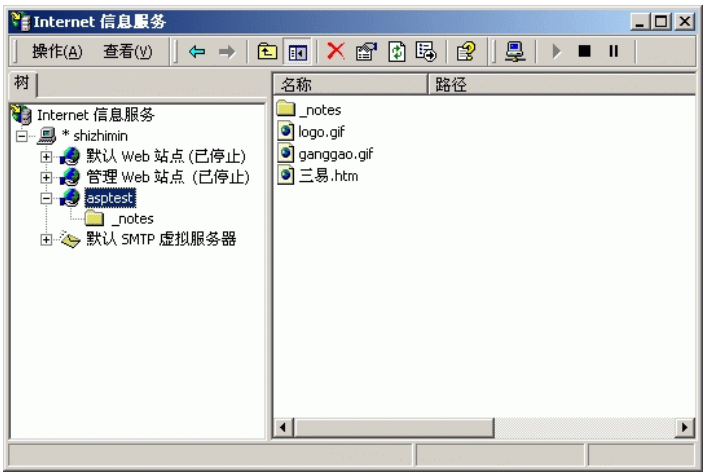

图 10.26 成功添加一个“asptest”站点

2．管理 Web 站点

设置主目录

我们在创建新站点的过程中有一个步骤是设置站点的主目录，它的作用是告诉浏览器到什么地方去找到网页文件，我们可以将网站的主目录设置为这个目录，但主目录并不是一成不变的，在必要的时候，可以将其他目录设置为主目录。

首先单击“开始”>“程序”>“管理工具”>“Internet 服务管理器”，打开管理控制台。选中要改变主目录的站点，单击右键，在快捷菜单中选择“属性”命令，在出现的对话框中选择“主目录”选项卡，将出现如图 10.27 所示的对话框。在该对话框中可以设置站点的主目录。

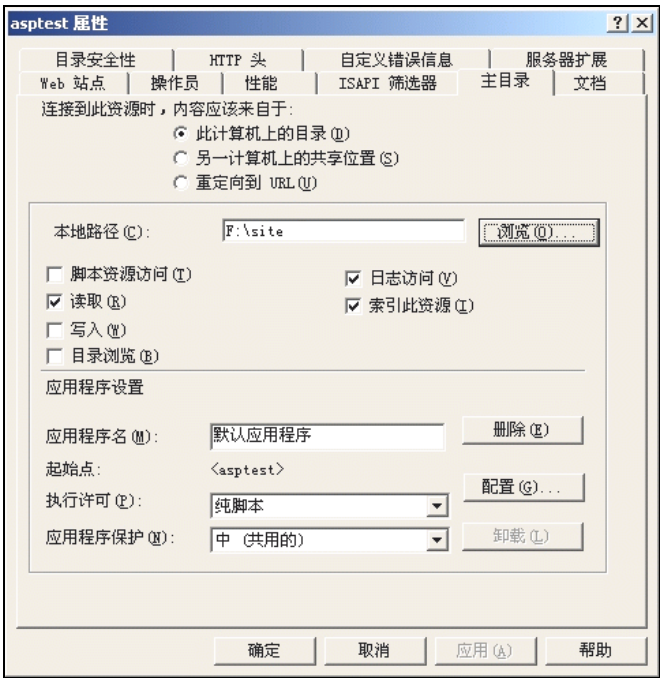

图 10.27　设置站点的主目录

首先选择目标资源的来源，一般选择第一项“此计算机上的目录”，在“本地路径”文本框中输入一个有效的路径，或者单击“浏览”按钮，在出现的选择文件夹对话框中选择需要的文件夹作为主目录。主目录的位置一旦改变，所有的 Internet 用户的请求都将被路由到这个新的位置。

之后单击“确定”按钮，返回“主目录”选项卡。在这里还可以设置主目录的访问许可：脚本资源访问、读取、写入、目录浏览、日志访问和索引此资源，一般保持默认的选择即可。特别要指出的是，出于安全上的考虑，请不要允许“目录访问”。

接下来要完成“应用程序设置”中的相关设置，常用的是“配置”部分，在图 10.27 中单击“应用程序设置”区域中的“配置”按钮，在出现的应用程序配置对话框中，打开“应用程序选项”选项卡，如图 10.28 所示。

在这里可以设置应用程序的基本配置，如“启用会话状况”、“启用缓冲”等。下面我们说明其基本含义。

（1）启用会话状况

如果选中该复选框，可以在用户访问你的站点时维护该用户的信息，一次访问期间叫作一次“会话”。可以使用 Session 对象来存储和处理属于用户的信息，每个用户的访问期

间由一个 Session 对象代表，这个对象在用户第一次单击你的应用程序时创建。

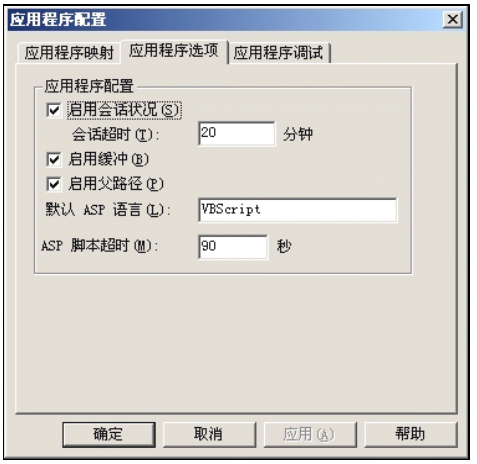

图 10.28 配置应用程序选项

默认时一个会话的存活期是 20 分钟，如果用户在 20 分钟之内没有作任何活动，则这个会话将会被清除，有时由于某种特殊的需要，我们需要将这个时间缩短或加长。那么就在“启用会话状态”下面的“会话超时”后的文本框中写入你期望的时间。

（2）启用缓冲

选中该复选框表示允许使用缓冲区，由于启用程序阶段的数据交换量比较大，所以通常要使用缓冲区来存放数据。这样可以加快数据的传输速率。

（3）启用父目录

就是允许使用上一级目录，即父目录。

（4）默认 ASP 语言

用户设置默认的 ASP 应用程序语言，通常都使用 VBScript 语言，如果你使用 JavaScript 更加熟练的话，也可以设为 JavaScript。

（5）ASP 脚本超时

ASP 程序属于解释执行代码而非一般的可执行程序，不能够自动运行。在这里设置 ASP 程序在浏览器中的解释执行的最大时间限制。如果超过了这个时间限制，以后的代码将不再解释执行。

除了上述这些配置，另外还可以进行应用程序调试功能的设置。在“应用程序配置”对话框中打开“应用程序调试”选项卡，如图 10.29 所示。

在“调试标志”选项组中设置有权调试 ASP 程序的一方——用户方或服务器方。如果选中“启用 ASP 服务器端脚本调试”复选框，则允许服务器一方调试 ASP 程序代码错误；如果选中“启用 ASP 客户端脚本调试”复选框，则允许用户方调试 ASP 程序代码。

在“脚本错误信息”选项组中设置将代码发送给用户时的方式。

实际上，应用程序的配置是比较复杂的，不过一般情况下我们直接采用默认设置就可以了。

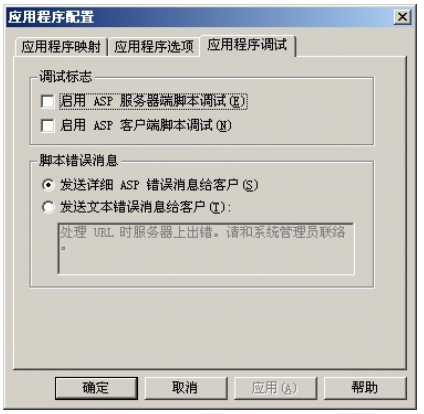

图 10.29　配置应用程序调试选项

设置虚拟目录

一个站点必须有一个主目录，主目录是一个缺省的位置，当用户的请求没有指定特定的文件时，IIS 将把用户的请求指向这个缺省目录。除了这种主目录之外，还有另外一种目录，叫做虚拟目录。

所谓虚拟目录，就是映射到站点目录树的目录，然而它在物理位置上驻留于该树之外，物理位置甚至可以是另一个服务器。虽然这些目录并不位于主目录之下，但是通常人们希望将它们映射到 Web 站点结构上，这可以通过添加虚拟目录来实现。处理虚拟目录时，IIS 把它作为主目录的一个子目录来对待；而对于 Internet 上的用户来说，访问时并不会感觉虚拟目录与其他站点中的其他目录之间有什么区别。具体的操作步骤如下：

（1）首先单击“开始”>“程序”>“管理工具”>“Internet 服务管理器”，打开管理控制台，定位在要添加虚拟目录的站点，如我们刚刚建立的 asptest 站点。

（2）右击该站点，选择“新建”>“虚拟目录”，如图 10.30 所示。

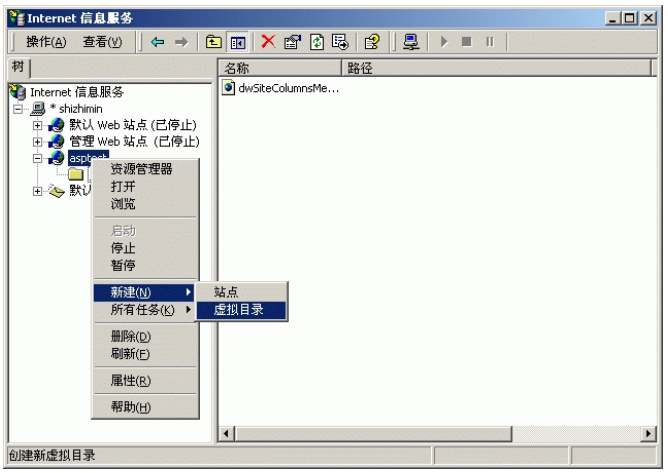

图 10.30　创建虚拟目录

（3）在出现的“虚拟目录创建向导”对话框中，单击“下一步”按钮。

（4）接下来出现的对话框要求输入虚拟目录的别名。该名称将作为虚拟子目录添加到主目录中，如图 10.31 所示。

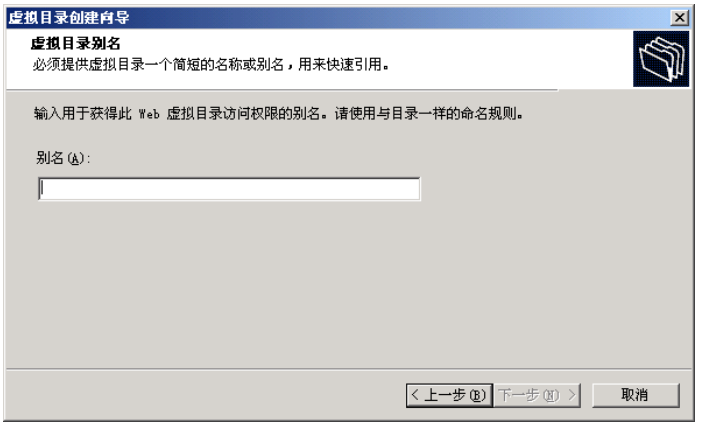

图 10.31　输入虚拟目录的别名

（5）为虚拟目录起好别名之后，单击“下一步”按钮，提示选择虚拟目录的物理位置。如果该目录在本地服务器中，可直接指定路径，如 c:\mywebs，如果在其他的服务器上，则需要指定服务器名和虚拟目录的共享名（这种情况下，此虚拟目录必须具有网络共享的属性），如\\srvname\mywebs，如图 10.32 所示

图 10.32　设置虚拟目录的物理路径

（6）单击“下一步”按钮，为虚拟目录设置访问权限，如图 10.33 所示。

（7）单击“下一步”按钮，所设置的虚拟目录就可以生效了。

给 Web 站点添加任意目录的能力可以帮助我们把驻留在不同位置甚至不同服务器上的数据放在一起。而在实际的物理存储上，将 Web 站点的负载分布到了多台服务器上，从而可以取得更高的处理速度。

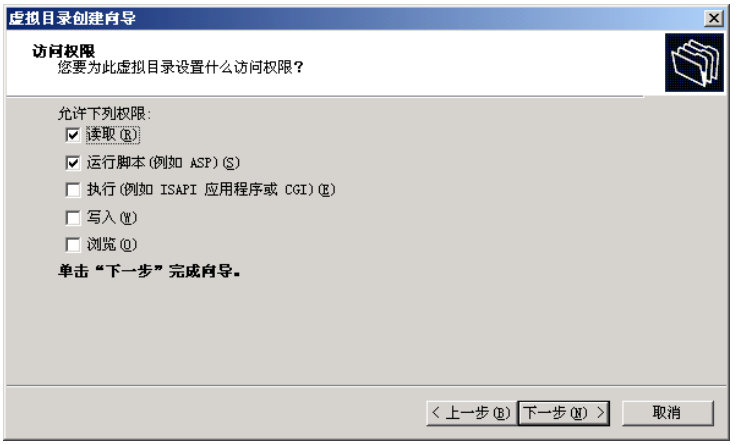

图 10.33　为虚拟目录设置权限

设置完虚拟目录之后，输入站点的 URL 地址并在后面加上斜杠“/”和虚拟目录的别名，就可以访问该虚拟目录了。比如站点的主目录是 www.website.com.cn，建立名为 news 的虚拟目录后，通过 www.website.com.cn/news 就可以访问这个虚拟目录。

设置主页文件

我们在访问 Web 站点时，很少说输入一个该站点的主页文件，但是我们总能得到一个显示正常的 Web 网页。这是因为在这个 Web 站点有一个默认的主页文件，用户只需输入 Web 服务器站点的地址，就可以指向该站点的默认主页文件，继而通过主页文件中的超链接访问站点中的其他网页。具体操作方法是：

在如图 10.27 所示的对话框中选择“文档”选项卡，出现如图 10.34 所示的对话框。

图 10.34　设置站点的默认文档

首先选中“启用默认文档”复选框，其中系统默认的文档有两个：Default.htm 和 Default.asp，如果你的站点的主页文件的名称和这两个之一相同，则可以使用默认的设置，否则单击“添加”按钮，从出现的“添加默认文档”对话框中输入你的主页文件名，如图 10.35 所示，单击“确定”按钮，这样就添加了一个主页文件。另外你还可以将原来的默认文档删除。如果指定了多个默认文档，服务器将按次序搜索本目录下的默认文档，将第一个找到的默认文档传送到浏览器，所以还可以使用左侧的移动符号来调整这些默认文档的次序。

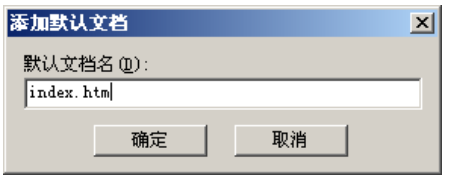

图 10.35 为站点添加默认文档

10.4 在 ASP 中使用 VBScript

10.4.1 设置主脚本语言

我们说，ASP 实际上是一种应用程序开发环境，而不是一种编程语言，它需要一种真正的程序语言来实现。VBScript 就是 IIS 服务器默认的脚本语言。

ASP 主脚本语言是用来处理在分界符 <% 和 %> 内部的命令的语言。默认的主脚本语言是 VBScript。可以将任何一种具有脚本引擎的脚本语言作为主脚本语言。可以逐页设置主脚本语言，也可以在 ASP 应用程序中设置所有页的主脚本语言。下面我们分别进行说明。

1．为某一页设置语言

要设置单个的主脚本语言，可将<% @ LANGUAGE %>指令添加到.asp 文件的开头，例如，我们要将本页的主脚本语言设置为 VBScript，可以使用下面的指令：

```
<% @ LANGUAGE = VBScript %>
```

如果对某页进行了设置，那么该页将忽略在应用程序中对所有页的全局设置。

2．为应用程序设置语言

在应用程序中，如果要为所有页设置主脚本语言，可在 Internet 服务器的“应用程序选项”选项卡上设置“默认的 ASP 语言”属性。参见图 10.28 所示。

10.4.2 在客户端使用 VBScript

在 ASP 程序中嵌入客户端脚本和我们在前几章中所举的例子一样，是通过使用<Script>标记和注释标记来实现的。在 ASP 程序中插入一段脚本的语法格式如下：

```
<SCRIPT language="脚本语言类型" runat="Client">
<!--
-->
</SCRIPT>
```

<SCRIPT>标记的 language 属性用以指定使用哪种脚本语言来编写脚本。runat 属性用于指定脚本运行的位置。该属性的取值为 Client 或 Server。当属性值为 Client 时表示该脚本将在客户端运行，当属性值为 Server 时表示该脚本将在服务器端运行，它的缺省值是 Client。

在本章之前，我们所举的例子都是在客户端运行脚本，使用方式是相同的，只不过不是 ASP 程序罢了。

10.4.3　在服务器端使用 VBScript

ASP 的强大功能之一就是它支持服务器端的脚本，在 ASP 程序中加入服务器端脚本的方法有两种。下面分别讲述。

1．使用<%和%>标记包含脚本

这是最简单也最常用的一种方法，其语法格式如下：

```
<%
ASP 命令或脚本语句
%>
```

例如，我们在 ASP 程序中，可以这样给某变量赋值：

```
<%
strA = "newland"
%>
```

在<%和%>之间不能包含 HTML 文档的内容，如果在脚本中间一定需要 HTML 的内容，则必须将其前后的脚本语句分段包含在<%和%>之中，否则程序将会出错。下面我们通过一个例子来说明，在这个例子中用到了 HTML 的<TABLE>等与表格有关的标记。

【例 10.1】　在 ASP 程序中分段显示的脚本语句

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<TABLE border=1 cellpadding=0 cellspacing=0 width=200 align=center>
<%
For i = 1 to 6
   If i mod 2 = 0 then
   %>
   <TR bgColor=#9db8e6><TD >彩色单元</TD></TD>
   <%
```

```
    Else
    %>
    <TR><TD>没有颜色</TD></TR>
    <%
    End If
Next
%>
</BODY>
</HTML>
```

该例的输出结果如图 10.36 **所示。**

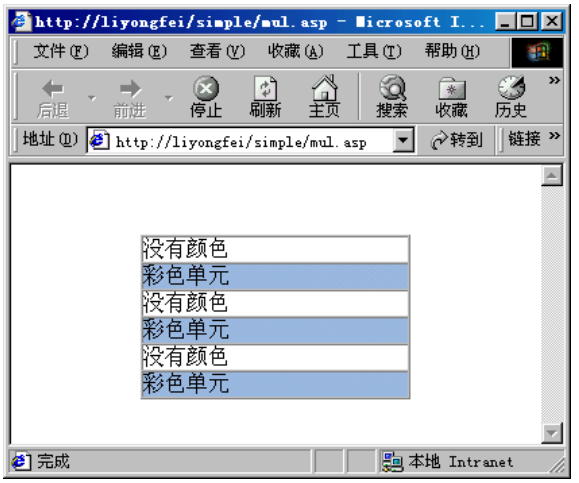

图 10.36 **脚本语句分段示例**

我们看到，在 HTML **标记**<TR><TD>……</TD></TR>**前后的脚本语句都分段包含在**<%**和**%>**之中，这样** ASP **解释器才能准确识别** HTML **标记，将其直接发送到客户端浏览器。**

但是使用上面的方法有时候使程序的可读性降低。还有一种常用的输出方法是使用 ASP **的内置对象** Response **的** Write **方法，将要输出的字符串或变量输出到浏览器中去。这样【例** 10.1**】可改写为：**

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<TABLE border=1 cellpadding=0 dellspacing=0 width=200 align=center>
<%
For i = 1 to 6
   If i mod 2 = 0 then
      Response.Write "<TR bgColor=#9db8e6><TD >彩色单元</TD></TR>"
   Else
```

```
        Response.Write "<TR><TD>没有颜色</TD></TR>"
    End if
Next
%>
</BODY>
</HTML>
```

它的输出效果和图 10.36 是相同的。使用 Response.write 方法使得整个脚本更加规范统一，脚本的可读性强，尤其在大段的脚本中，我们更多地是使用 Response.write 方法来输出显示。

2．使用<SCRIPT>标记

也可以使用<SCRIPT>标记在 ASP 程序中加入服务器脚本，其语法格式如下：

```
<SCRIPT language="脚本语言类型" runat="Server">
</SCRIPT>
```

在<SCRIPT>标记中必须使用 runat="Server"来声明该脚本是在服务器端执行的，否则，由于 runat 属性的缺省值是 Client，ASP 解释器将会把这段脚本传送到客户端，由客户端浏览器解释执行。language 属性用于指定采用哪一种脚本语言来编写该脚本。可以使用任何一种脚本语言来编写服务器端的脚本程序，前提是在 Web 服务器中安装了相应的脚本引擎。ASP 自身带有两个脚本引擎，支持 VBScript 和 Jscript 语言。

3．使用两种嵌入方式的不同

我们讲了在 ASP 中嵌入脚本的两种方法，虽然它们能够完成相同的功能。但是在使用的时候，采用<SCRIPT>标记和采用<%和%>标记还是有所不同的，二者之间主要有如下两点不同：

（1）执行时间不同

使用<SCRIPT>标记嵌入的脚本，如果指定的脚本语言是 VBScript，那么嵌套在其中的脚本将被最后执行，如果指定的脚本语言是 Jscript，那么嵌套在其中的脚本将被最先执行；而对于使用<%和%>标记嵌入的脚本，将会按照 ASP 文件中的书写顺序进行解释，只有当脚本解释到该处时，才会执行。

所以，在实际使用中，我们更多的使用<%和%>标记，原因是我们一般需要将 ASP 脚本嵌套在 HTML 的标签之间，按照预先设定的顺序执行。

（2）使用<SCRIPT>标记能够将多种语言混合使用

在实际的应用中，我们通常会遇到这样的情况，对于这种功能，用 VBScript 做比较合适，对于另外一个功能使用 Jscript 比较合适，或者是由于对脚本语言掌握的程度不一样，我们需要在这一段脚本中使用 VBScript，而在那一段脚本中使用 Jscript。下面的例子向我们显示了这一点。

【例 10.2】　混合使用两种脚本语言

```
<HTML>
```

```
<HEAD>
</HEAD>
<BODY>
<SCRIPT language="VBScript" runat="Server">
<!--
    Dim i
    For i = 1 to 3
        Response.Write "这是使用 VBScript 输出的句子<BR>"
    Next
-->
</SCRIPT>

<SCRIPT Language="JScript" RUNAT="Server">
    for (i = 1; i <= 3 ;i++)
     {
        Response.Write ("这是使用 JScript 输出的句子<BR>")
     }
</SCRIPT>
<P> </P>

</BODY>
</HTML>
```

本例中，我们分别使用了 VBScript **和** Jscript **的循环语句，将字符串分别显示** 3 **次，执行的结果如图** 10.37 **所示。从输出的结果可以看到，**Jscript **脚本先执行，然后是** HTML **文本中的**<HR>**，最后才是** VBScript **脚本被执行。**

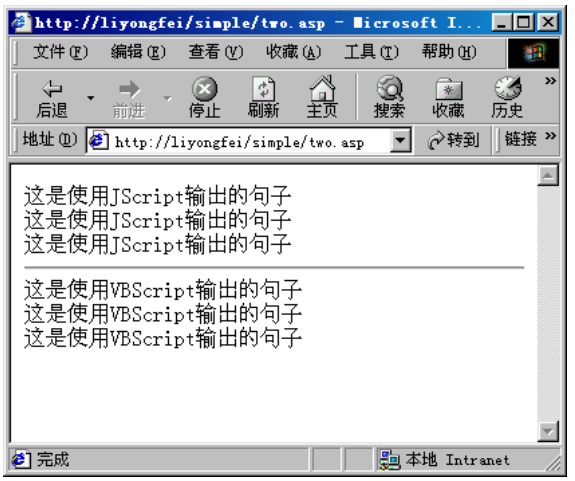

图 10.37 **混合式使用两种脚本语言的结果**

提示：在服务器端通过 ASP **使用** VBScript **时，有两个** VBScript **特征将失效。由**

于 ASP 脚本是在服务器端执行的，所以表示用户接口元素的 VBScript 语句 InputBox 和 MsgBox 将不被支持。另外，在服务器端的脚本中，请不要使用 VBScript 函数 CreateObject 和 GetObject，而要使用 Server.CreateObject，这样 ASP 就可以跟踪对象实例了。用 CreateObject 或 GetObject 创建对象不能访问 ASP 的内置对象，也不能参与事务处理。该规则的一个例外是当使用 Admin 对象和 JavamonikeRs 时。

由于 ASP 脚本是在服务器端处理的，所以即使客户端的浏览器不支持脚本语言，也不必通过包含进 HTML 注释标记来隐藏脚本，但客户端的脚本则通常需要这样处理。在内容送到浏览器之前，所有的 ASP 命令都已经被处理好了。可以用 HTML 注释将注释加进 HTML 页，注释将返回给浏览器，若用户浏览 HTML 源文件，就可以看见。

10.4.4　脚本性能问题

一般来说，Web 服务器处理器的速度是足够使用的（除非特别繁忙的站点），因为它们的主要任务是从磁盘中载入页面并发往客户端。因此，每个页面的请求结果都使处理器等待磁盘。这意味着执行 ASP 脚本通常对性能的影响非常小。而且如果在一个页面上某段脚本代码多次执行，而这段代码的编译版本已被高速缓存，那么只需执行它，而不必多次编译，这样对性能的影响就更小了。

当然，随着请求数量的增加，服务器负载也不断增加，解析和执行每一个 ASP 页面就有相应的代价，应尽可能压缩 ASP 解释器的工作量。下面是一些有用的提示。

1．避免在同一页面上混用脚本语言

如果同一页面上有几种脚本语言，ASP 不得不一个接一个地加载多种脚本引擎，并把相应的代码送给相应的引擎。这将降低处理速度，增加内存的使用量。另外一个副作用是，假如编写的是一个顺序执行的代码（而不是一系列从其他代码段调用的函数或子程序），可能会以与它们在页面中出现的顺序不同的顺序执行。

例如，下面的代码可能不会产生所希望的结果，因为无法确保 Jscript 代码结果在网页中是首先出现，或是在第三位出现。

```
<%@ language=JScript %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT language="JScript" runat="SERVER">
    Response.Write ("first<BR>")
</SCRIPT>

<SCRIPT Language="VBScript" runat="SERVER">
    Response.Write "second<BR>"
</SCRIPT>
```

```
<%
    Response.Write ("third<BR>")
%>
<P> </P>

</BODY>
</HTML>
```

2．在脚本和其他内容中过多的环境切换

每当 ASP 遇到一个脚本段，必须执行并把结果发到 IIS/PWS，然后再次返回去解释页面。因此，使用 Response.write 语句（只创建发往客户端的文本，类似于 print 命令行）能使页面的效率更高。例如下面这段 VBScript：

```
<BODY>
<%
intResult = 10 * 6 -1
strTheSum = "十乘以六减去一"
Response.Write " strTheSum " & "  的计算结果是：" & CStr(intResult)
%>

<%
intResult = 10 * 6 -1
strTheSum = "十乘以六减去一"
%>
<BODY>
<% = strTheSum %>  的结果是：<% = CStr(intResult) %>
</BODY>
```

3．构建单独的组件

例如在一个页面不得不做大量的运算，或者运行一个过于复杂的脚本，通常的好办法是构建一个组件，并安装在 Web 服务器上。组件通常是编译过的可执行代码，相对于解释 ASP 脚本代码，使用的效率更高。

10.5 一个使用 ASP 的例子

在 ASP 中使用 VBScript 脚本，一般我们都使用<%……%>标记。所有在这个标记中的内容被视为 VBScript 服务器端的脚本来执行。我们看一个小例子。

【例 10.1】 在 ASP 中使用 VBScript

```
<HTML>
<HEAD>
<TITLE>在 ASP 中使用 VBScript</TITLE>
</HEAD>
```

```
<BODY>
<%
For i = 6 to 1 step -1
   Response.Write "<h" & i & ">标题" & i
Next
%>

</BODY>
</HTML>
```

该例输出的结果如图 10.38 所示。

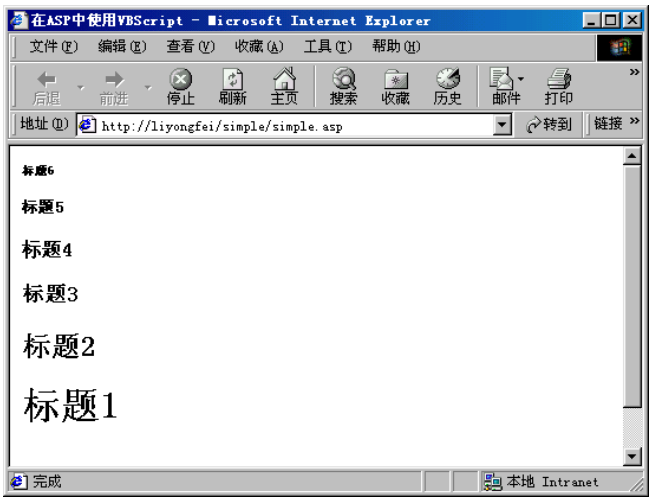

图 10.38　在 ASP 中使用 VBScript

请注意，在图 10.38 中，浏览器的地址栏中是一个 HTTP 的请求，而不是某个文件的路径，因此它是不能够通过在资源管理器中双击该文件打开的。这就是一种“基于浏览器”的概念。为了能够执行这个文件，必须按照前面所介绍的方法为其所在的文件夹建一个虚拟目录。

10.6　用 VBScript 存取数据库

要开发动态的、内容能够及时更新的网页，或开发基于 Web 的应用，将 Web 平台和数据库系统结合的技术十分重要。大多数应用系统都依赖数据库管理系统的工作数据，如果 Web 上的应用能够存取数据库中的数据，那么它的应用范围则会相当广泛，而实际上确实如此。本节我们将介绍使用 VBScript 访问数据库的技术。为了这么做，你将学习如何使用 Active Data Object（ADO，Active 数据对象），它提供了数据库和 Active Server Pages（ASP）之间的接口。可以在 ASP 中使用每一个通过 OLE DB（Object Linking 和 Embedding Database，对象链接和嵌入数据库）或 ODBC（Open Database Connectivity，开放数据库连接）访问每一个数据库。如果你没有使用过或者没有听说过 OLE DB 和 ODBC，也不要担

心，本节我们将会介绍这两个术语以及它们和 ASP 的关系。

如果想要在自己的 Web 服务器上运行这一节中的实例，需要一个 Microsoft Access、SQL Server 或者其他的数据库，并且能够从你的 Web 服务器上访问它。

10.6.1 ADO 对象模型

使用 VBScript 在服务器端利用数据库进行编程，需要利用 ADO 对象对数据库进行操作。ADO 是 ASP 内置的服务器对象，是数据库访问组件，将 ADO 与 ASP 结合能建立提供数据库信息的网页，能在网页中执行 SQL 语句，对数据库进行查询、插入、修改和删除等操作，而将 ADO 与 VBScript 或 JavaScript 结合起来使用能用来控制数据库的访问和查询结果的输出。ADO 可以连接到任何支持 ODBC 的数据库，比如 Access、SQL Server、Oracle 等，ADO 是位于 ODBC 之上的、高性能的数据操作接口。

ADO 包括 7 个对象，分别是：

（1）连接对象 Connection
（2）命令对象 Command
（3）记录集对象 Recordset
（4）错误对象 Error
（5）字段对象 Field
（6）参数对象 Parameters
（7）属性对象 Properties

利用这些功能强大的对象即可轻松完成对数据库复杂的操作。使用 ADO 设计访问数据库的要领如下：

1．创建数据源名

在使用 ODBC 时，经常看到 DSN 这个名词，它究竟是什么呢？对我们又有什么作用呢？DSN 的全称是 Date Source Name，即数据源名称。ODBC 是一种访问数据库的方法，只要系统中有相应的 ODBC 驱动程序，任何程序都可以通过 ODBC 操纵驱动程序的数据库。比如系统中有一个 Access 的 ODBC 驱动程序，那么即使没有 Access 软件，也可以在程序中对一个 Access 的 mdb 数据库进行添加、删除和更改的操作，而且不需要知道这个数据库是放在哪里的。只要写出 SQL 语句，ODBC 会做一切事情。在给 ODBC 驱动程序传 SQL 指令时，就使用 DSN 来告诉它到底操作的是哪一个数据库。如果数据库的平台变了，比如改用了 Oracle 的数据库，只要其中的表结构没有变，就不用改写程序，只要重新在系统中配置 DSN 就行了。所以我们说，DSN 是应用程序和数据库之间的桥梁，要通过 ODBC 访问数据库，前提就是必须配置好 DSN。一个 DSN 包含下面一些信息：

- DSN 的名字：指给这座桥梁取的一个名字，当程序访问数据库时，给系统传的就是这个名字，而不是数据库的实际名称。
- ODBC 驱动程序类型：只有指出驱动程序类型，在操作数据库时，系统才会知道调用哪个 ODBC 驱动程序来服务。
- 数据库：必须指定这座桥到底连接的是哪个数据库，因为不同的数据库系统，指

定数据库名字的方法有些不同。

在 Windows 98/2000 中提供了一个工具来完成上面的这件事，在控制面板中，选中“ODBC 数据源”，DSN 有 3 类，即用户 DSN、系统 DSN、文件 DSN。一般使用系统 DSN，因为这样可以让所有在该系统上操作的人都能使用这个 DSN。

打开“控制面板”，双击“ODBC 数据源”，进入如图 10.39 所示的对话框。

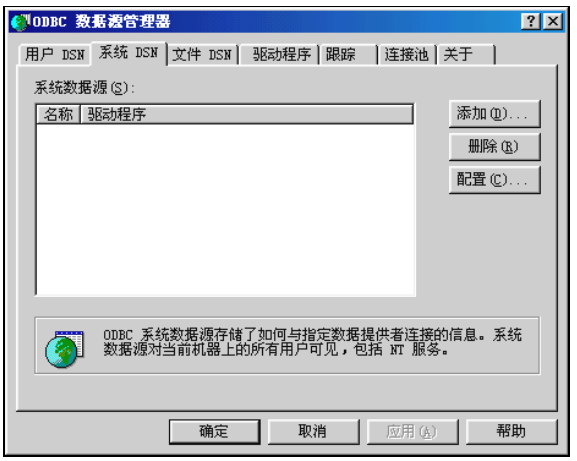

图 10.39　ODBC 数据源

选中标签“系统 DSN”，单击“添加”按钮，选择作为数据库的相应驱动程序，我们以 Microsoft Access（MDB）为例，如图 10.40 所示。

图 10.40　选择 ODBC 驱动程序

选中后，单击“完成”按钮，出现如图 10.41 的对话框。

将数据源名设为你喜欢的任意的名称，如“testdb”，单击“确定”按钮，再单击“选取”，在如图 10.42 所示的对话框中选取要使用的数据库，例如“c:\db1.mdb”，这样我们就完成了数据源名的创建。以后就可以用“testdb”来访问数据库“c:\db1.mdb”了，而不是直接用文件名“c:\db1.mdb”。这样做的好处是，一旦改变了文件名或存放路径，只要修改

DSN 中的配置就可以了，而不必逐个修改程序中所有的该数据库的名称。

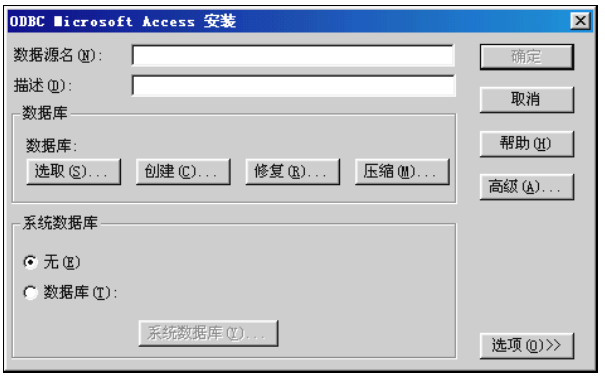

图 10.41 选择数据源

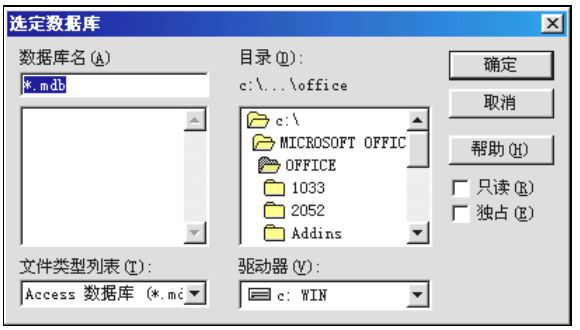

图 10.42 选取数据源相关联的数据库

创建了数据库之后，下面就可以创建数据库的链接了。

2．创建数据库链接（Connection）

用过 VB 的人都知道，DAO 中有链接（Connection），RDO 中有 ADOConnection。链接可以保持一些关于正在访问的数据的一些状态信息以及链接者的信息。在 ASP 文件中，如果要访问数据，必须首先创建与数据库的链接，其语法如下：

```
Set Conn = Server.CreateObject ("ADODB. Connnection")
```

执行完这条语句之后，Conn 便是一个 Connnection 对象。

3．打开数据库

调用 Conn.Open 方法打开数据库：

```
Conn.Open "testdb", "username", "password"
```

这条语句打开链接，第一个参数就是我们在第一步中设置好的 DSN，第二和第三个参数是访问数据库的用户名和口令，都是可选参数。

4．创建数据对象（RecordSet）

前面我们已经创建了数据库的链接，并且已经打开了这个链接，下面就该设定 SQL 命

令，并通过执行 SQL 命令来访问数据库了。我们把存取数据库的操作结果储存到 RecordSet 对象中。

ADO 中的数据对象通常保存的是查询结果。RecordSet 是 ADO 中最复杂的对象，有许多属性和方法，灵活运用可以达到许多好的效果。RecordSet 中保存的是一行一行的记录，并标有一个当前记录。

首先，我们写一条标准的 SQL 语句，例如：

```
SqlStr = "SELECT * FROM tab1"
```

然后，使用下面的方法执行该 SQL 语句：

```
Set Rs = Conn.Execute(SqlStr)
```

这条语句通过调用链接对象的 Execute 方法来将查询结果返回给一个数据对象 Rs。Execute 方法的参数是一个标准的 SQL 语句串，所以可以利用它方便地执行数据的插入、删除和修改等操作。

这条语句执行以后，对象 Rs 中就保存了表 tab1 中的所有记录。Rs 是一个 RecordSet 类型的对象。

5．操作数据对象

我们得到数据库中的数据，是为了对其进行相应的处理，比如将它们以表格的形式显示到页面上等。

在得到了一个 RecordSet 对象之后，如何将它的内容读出来呢？在数据库中最小的数据单位是字段，在 ADO 的对象中，与字段相对应的对象是 Field。例如：Rs.Field(0).name 代表字段 0 的抬头，Rs.Field(0).Value 代表字段 0 的内容。举个例子来说，在数据库中有一张表，其字段与内容如图 10.43 所示。

图 10.43　Access 数据库中的一张表

当我们使用前面的方法将这个数据库表中的数据读到一个 Rs 对象中后，这个 Rs 对象就具有了 Fields 集合，它包含了 Rs 的所有 Field 对象。使用下面的方法就可读取相关内容。

Rs.Field(0).name 代表“姓名”
Rs.Field(1).name 代表“年龄”
Rs.Field(2).name 代表“性别”

当当前光标在第一条记录上时，Rs.Field(0).Value 代表“张三”，Rs.Field(1).Value 代表 25，Rs.Field(2).Value 代表“男”。

使用 Recordset 对象的 MoveNext 方法将当前记录移动到下一行。

```
Rs.MoveNext
```

这时，Rs.Field(0).Value 就包含了“李四”，以此类推，Rs.Field(1).Value 包含了 23，Rs.Field(2)包含了“女”。

Recordset 对象的默认属性是 value，所以也可以使用 Rs.Field(0)或 Rs(0)这样的形式来返回当前记录的数据。

还可以使用另一种方法来读取记录集中的数据，就是使用数据库表中字段的名称。如要得到当前记录中字段“姓名”的数据，可以使用 Rs("姓名")，我们建议在编程的过程中使用这种取值方法，因为当数据库结构发生变化的时候，比如，在表结构中又添加了一个字段，这时如果使用指定列值的方法，可能就得修改程序了。但是如果使用指定字段名的方法，只要数据表中该字段的名称没有发生变化，就不用修改程序了。

6．关闭数据对象和链接对象

记住，在使用了 ADO 对象之后，一定要记住关闭它，因为它使用了一定的服务器资源。通过调用方法 Close 实现关闭，然后释放它。

```
Rs.close
Set Rs = nothing
Conn.close
Set Conn = nothing
```

至此，我们已经给出了用 ASP 访问数据库的全过程，包括创建数据源、创建链接、创建数据对象、操作数据对象、关闭并释放。我们将 ADO 访问数据库的过程用一幅图来说明，如图 10.44 所示。

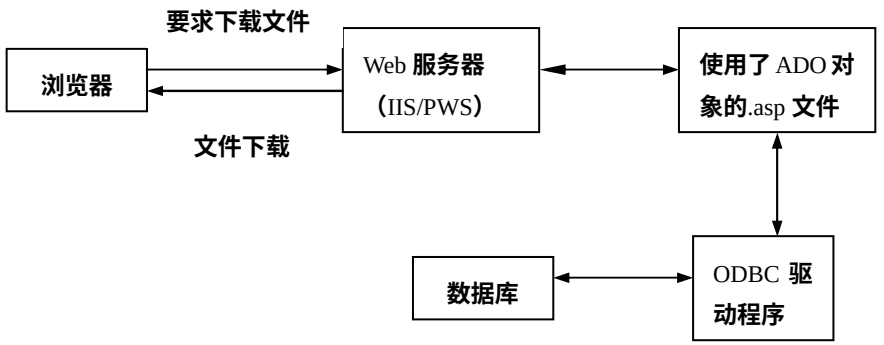

图 10.44　ADO 访问数据库的过程

实际上由于应用了面向对象的思想，所有的这些操作都比较简单，用户需要注意的仅仅是对数据结构的了解，能够编写一些 SQL 语句，也要清楚当前所操作的对象是什么，有什么属性等。只要对这些有了清醒的认识，再加上 ASP 的强大功能，在网络上应用自己的数据库，就是轻而易举的事了。

10.6.2　ADO 访问数据库的例子

下面介绍一个浏览数据库记录的例子。首先创建一个 Microsoft Access 数据库 sale.mdb，在这个数据库中，有一个客户订单表 ordeRs。如图 10.45 所示是这个表的内容。

图 10.45　数据库中表结构与记录

下面要做的，就是利用我们前面讲到的方法，在客户端的浏览器上，显示这个数据库表的全部记录。我们建立的 ODBC 数据源名称为 ordeRs。

ShowOrderlist.asp 的源代码：

```
<HTML>
<HEAD>
<TITLE>使用 ASP 浏览数据库表</TITLE>
</HEAD>
<BODY>
<P align=center><B>订单列表</B></P>
<TABLE align=center border=1 cellpadding=0 cellspacing=0 width=85%>
<TR bgColor="darkkhaki">
   <TD>序号</TD>
   <TD>顾客姓名</TD>
   <TD>联系电话</TD>
   <TD>联系地址</TD>
   <TD>定购商品</TD>
   <TD>数量</TD>
</TR>
<%
建立数据库连接
Set Conn = Server.CreateObject ("ADODB.Connection")
Set Rs = Server.CreateObject ("ADODB.RecordSet")
打开数据库
Conn.Open "orders"
写 SQL 语句
SqlStr = "Select * from orders order by ID"
执行 SQL 语句，并把执行结果存入 RS 对象中
Set Rs = Conn.Execute (SqlStr)
```

```
While not (Rs.EOF or Rs.BOF)

    Response.Write ("<TR>")
    Response.Write ("<TD>" & Rs("ID") & "</TD>")
    Response.Write ("<TD>" & Rs("顾客姓名") & "</TD>")
    Response.Write ("<TD>" & Rs("联系电话") & "</TD>")
    Response.Write ("<TD>" & Rs("联系地址") & "</TD>")
    Response.Write ("<TD>" & Rs("订购商品") & "</TD>")
    Response.Write ("<TD>" & Rs("商品数量") & "</TD>")
    Response.Write ("</TR>")
    Rs.MoveNext
Wend

Rs.Close
Conn.Close

%>

</TABLE>
</BODY>
</HTML>
```

如图 10.46 所示是查询的结果。

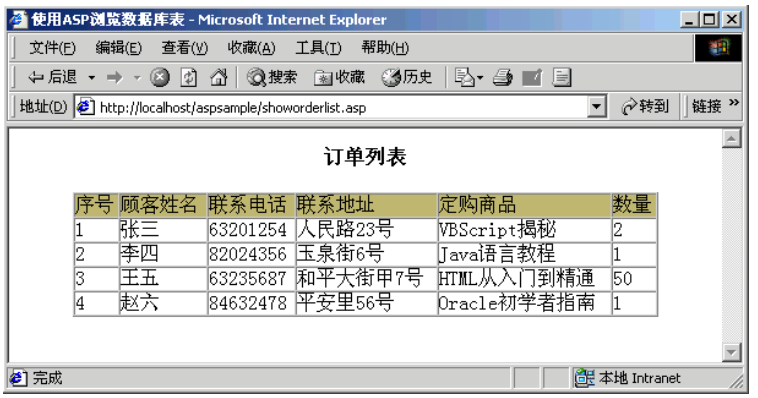

图 10.46　使用 ASP 读取数据库的结果

10.7　小　结

这一章中，我们主要学习了 ASP 的一些知识以及如何在 VBScript 中使用 ASP。主要内容包括 ASP 简介、PWS 的安装与设置、IIS 的安装与设置、如何在 VBScript 中使用 VBScript 和用 VBScript 存取数据库。

通过本章的学习，我们已学会了 ASP 开发环境的搭建，在 ASP 中使用 VBScript 和使用 VBScript 存取数据库，尤其是有关 ADO 对象模型要重点掌握。

练习题

1．运行 ASP 有什么要求？

2．ASP 文件是在服务器端（Web 服务器）还是在客户端（Web 浏览器）上处理？

3．.asp 文件的 ASP 代码是否从服务器传送到浏览器上？

4．什么是虚拟目录？

5．在 Web 站点上没有找到默认文档时，要允许用户看到一个目录下的所有文件，需要启用哪个设置？

6．如何在 ASP 中使用 VBScript？

7．描述使用 ASP 访问数据库的几个步骤，着重理解 ADO 的 3 个重要对象：连接对象 Connection、命令对象 Command 和记录集对象 Recordset。

8．制作一个商品查询系统，要求能根据商品名称查询出该商品的其他信息（如商品编号、剩余数量和价格等）。

9．制作一个简单的留言板程序，实现留言的提交和浏览留言。

第 11 章　VBScript 特效集锦

在前面的章节中，我们已经学习了 VBScript 编程的许多知识，详细讲解了众多语句的使用，并在每个知识点都穿插了有针对性的实例，以帮助读者迅速理解。本章我们将通过一些综合性较强的例子，向大家进一步展示 VBScript 的诸多妙用和强大功能。希望读者通过研读这些例子，能够提高综合运用知识的能力，学会一些使用 VBScript 编程的方法和技巧，并最终把所学知识运用到自己的网页设计中，设计出精致美观和功能强大的网页。其实，本章所举实例都很典型也很常用，你可以直接将其搬到自己的网页上，也可以稍加修改后使用。

11.1　状态栏特效

11.1.1　实例目标与技术要点

通过这个例子，我们将学会如何控制状态栏上的文字显示，本例要达到的效果是：字符像雨点一样从状态栏的右侧向左侧落下，直到碰到了边壁或已经落下的字符才停止。当所有的字符下落完毕，整个过程重新开始。

本例要点：

- 对字符串的处理与操作。
- 状态栏的使用。
- mid 函数的使用。
- setTimeout 函数和 clearTimeout 函数的使用。

11.1.2　实例过程

因为重点是掌握有关状态栏的知识，所以对于页面没有做什么工作。只包含了一个 HTML 文件必备的标签，在其中加入<SCRIPT>标签用于以后添加脚本代码，当然记住在网页下载的时候调用显示动态状态栏的过程 setMessage。

```
<HTML>
<HEAD>
<TITLE>状态栏特效</TITLE>
<SCRIPT Language="VBScript">
<!--
-->
</SCRIPT>
</HEAD>
```

```
<BODY Onload=" setMessage()">
这是一个状态栏特效的例子
</BODY>
</HTML>
```

下面我们为其添加代码。

先介绍状态栏效果用于最后显示的语句格式：

```
window.status = showingmessage
```

这个语句为 Window 对象的状态栏属性赋值，所赋的值将被作为字符串变量对待。状态栏是有固定长度的，字符串的长度超过这个固定长度的部分将不被显示。这就给一系列状态栏效果带来了发挥的契机。

本例的实现是通过下落过程由雨点字符和落地字符串之间的空格的减少来完成的。例如，将要显示的有效字符串为“abcde”，而已经有“abcd”4 个雨点落地了，那么当“e”开始下落时，用于显示的字符就是“abcd……e”。其中“……”表示若干个空格。随后空格逐渐减少，当空格减少到 0，就意味着雨点落地了，最后的字符串为“abcde”。使用这种方法，所有的字符一一落地，直到落地雨点的字符串长度等于有效字符串的长度为止。

下面开始书写动态状态栏的脚本：

```
Dim init_msg, str, msg, leftmsg
'定义有效字符串
init_msg = "新概念 VBScript，棒棒的 !!! "
'初始化变量
str = ""                    '空格逐渐减小的动态变换的文字
msg = ""                    '未显示完的字符串
leftmsg = ""                '已经落下的文字
'动态状态栏过程
'生成用于显示的字符串
'添加已经落下的雨点字
Sub setMessage()
'过程的重新开始
    If msg = "" Then
        str = " "
        msg = init_msg
        leftmsg = ""
    End If
    '已经有一个字符落到状态栏的左端
    If Len(str) = 1 Then
        leftmsg = leftmsg & str        '将重新设置已落地的的字符串
        str = mid(msg,1,1)             '截取未显示的字符串的第一个字符
        msg = mid(msg,2,len(msg))      '截取未显示字符串除第一个字符以外的字符
        For i = 0 to 139               '将 str 赋值为 140 个空格
和未显示字符串的第一个字符串的组合
```

```
                str = " " & str
            Next
        Else
    每次减少 10 个空格
            str = Mid(str,11,len(str))
        End If
        为状态栏赋值
        Window.status = leftmsg & str
        '反复调用过程过程使整个过程循环执行
        Window.setTimeout "setMessage()",200
End Sub
```

在程序中，一个关键的变量是 str。我们主要就是通过它来实现状态栏的动态变化的。在程序的开始，进行一些初始化的工作，将动态显示的字符串 str 设为一个空格，msg 设为有效字符串“ 新概念 VBScript，棒棒的 !!!”，而表示已经落地的字符的 leftmsg 变量当然应该是空值。

在程序中，一开始的状态栏的显示是 140 个空格加上有效字符串的第一个字符“ 新 ”，所执行语句为：

```
        leftmsg = leftmsg & str
        str = Mid(msg,1,1)
        msg = Mid(msg,2,len(msg))
        For i = 0 to 139
            str = " " & str
    Next
Window.status = leftmsg & str
```

这样的效果是在状态栏的最右端显示第一个字符，同时把尚未显示的字符存放在 msg 中，此时为除第一个字符以外的所有字符，即“ 概念 VBScript，棒棒的 !!!”。下一次执行 setMessage 的时候，str 的值将发生变化，如果 str 的长度不等于 1，也就是它所带的小雨点还没有降落到顶头的时候，将其中的空格减去 10，这表明它所带的小雨点以每次 10 个字符长度的频率向前移动。执行语句是：

```
str = Mid(str,11,len(str))
Window.status = leftmsg & str
```

当 str 的长度减至 1 的时候，表明其所带的小雨点落到了状态栏的左端。这时重新执行：

```
leftmsg = leftmsg & str
str = Mid(msg,1,1)
msg = Mid(msg,2,len(msg))
  For i = 0 to 139
    str = " " & str
  Next
```

```
Window.status = leftmsg & str
```

首先将已经落地的字符赋指给 leftmsg。由于此时的 msg 串中包含的是还没有显示过的字符“概念 VBScript，棒棒的！！！”，所以 str 中包含了 140 个空格和有效字符串中的第二个字符“概”，所以 left&str 的值就变为有效字符串的第一个字符串加上 140 个空格再加上第二个字符，即“新”加上 140 个空格再加上“概”。再次执行 setMessage 的时候，则是实现减少这两个有效字符中间的空格，依此来使得第二个字符向第一个字符逼近。

如此循环，直到所有的有效字符全部落到状态栏的左端。这时，msg 为空，如果再一次执行 setMessage 就会从开始重复这个过程。如此循环就产生了我们预期的效果。

11.1.3　实例效果与总结

下面我们来看一下实例的运行效果，如图 11.1 和图 11.2 所示。

图 11.1　雨点式状态栏效果抓拍 1

图 11.2　雨点式状态栏效果抓拍 2

在实例过程中，我们提到了一个概念——“有效字符串”，它的含义是用于想要显示的完整信息，而实际字符串表示的是每一时刻显示在状态栏上的信息。控制字符串的变化就是控制状态栏的动态效果，只要变化适宜，就不会影响有效字符串的显示。在本例中，我们多处使用了 Mid 函数来处理字符串。它的功能是截取指定字符串中指定位置指定个数的字符，然后返回一个 String 类型的 Variant 值。利用这个函数可以完成很多字符串的编程。

本例的第二个重点是 Window 对象的状态栏属性 status。对于视窗系统中的每一个窗口，都有其状态栏，状态栏的基本作用是显示当前窗口的状态。例如，当用户首次登录这个页面时，显示一句欢迎的信息，或者当浏览器窗口正在下载一个文件时，状态栏显示“正在下载……”的信息，这样用户对窗口目前所处的状态一目了然，而且可以根据它确定下一步的动作。其实，为状态栏赋值是一项很简单的工作，可以使用下面的语句：

```
object.status = string
```

这里 Object 指某个窗口对象，String 表示一个合法的字符串。status 属性是可读写的。为编程人员提供了很大的灵活性。

还有一个重要的函数——setTimeout 函数，该方法的适用范围是 Window 对象，它的功能是在规定的毫秒数经过之后计算一个表达式。语法为：

```
intervalID = object.setTimeout(expression,msec[,Language])
```

其中，expression 表示在规定的时间间隔执行代码。msec 为整数值或数字串（长），单位是毫秒，表示相隔多长时间后执行这部分代码。Language 选项规定语言的串代码以这种语言执行。

该函数返回一个唯一的标识符，只用于取消用 clearTimeout 方法进行的计算。

与 setTimeout 函数相关的一个函数是 clearTimeout 函数，clearTimeout 函数的功能是取消用 setTimeout 方法设置的超时。语法为：

```
object.cleatTimeout(timeoutID)
```

11.2 彩色波浪文字

11.2.1 实例目标与技术要点

在这个实例中，我们要完成一个文字变幻的任务，它通过更改文字的样式信息，改变文字的大小和颜色，达到一种波浪文字的效果。

本例要点：

- 样式单的定义与使用。
- 使用脚本访问样式单。
- 在网页中动态地改变元素的属性。

11.2.2　实例过程

本例的界面非常简单，因此它的 HTML **部分的代码也不多，为了便于讲解，我们还是将它写出来：**

```
<HTML>
<HEAD>
<TITLE>彩色文字</TITLE>
<STYLE>
<!--
.big1{font-family:comic sans MS;font-size:10pt;color:rgb(0,128,0}
.big2{font-family:comic sans MS;font-size:20pt;color:rgb(0,128,0}
.big3{font-family:comic sans MS;font-size:30pt;color:rgb(0,128,0}
.big4{font-family:comic sans MS;font-size:40pt;color:rgb(0,128,0}
.big5{font-family:comic sans MS;font-size:30pt;color:rgb(0,128,0}
.big6{font-family:comic sans MS;font-size:20pt;color:rgb(0,128,0}
.big7{font-family:comic sans MS;font-size:10pt;color:rgb(0,128,0)
-->
</STYLE>
<SCRIPT Language="VBScript">
<!--
……写入程序脚本代码
-->
</SCRIPT>

</HEAD>
<BODY OnClick="change()">

<P align="center">
    <SPAN class ="big1" id ="font1">One</SPAN>
    <SPAN class ="big2" id ="font2">two</SPAN>
    <SPAN class ="big3" id ="font3">Three</SPAN>
    <SPAN class ="big4" id ="font4">four</SPAN>
    < SPAN class ="big5" id ="font5">five</SPAN>
    < SPAN class ="big6" id ="font6">six</SPAN>
    < SPAN class ="big7" id ="font7">Seven</SPAN>
</P>

</BODY>
</HTML>
```

其中，有 7 **个**<SPAN>**标记，该标记中的文字是用于变换样式的文字。**7 **个样式分别为** big1~big7**，样式中**<STYLE>**定义了文字的字体（**font-family**）、大小（**font-size**）、颜色（**color**）** 3 **个属性。设它们的初始状态与** big1~big7 **一一对应，颜色全部是单一的黑色。**

接下来的任务是让文字动起来，我们要达到的目的是使文字像波浪那样一高一低地移动，同时颜色也不停地变换。所以，我们在上述的 HTML 文件中的<SCRIPT>标记中加入下列代码：

```
Dim stylefont(6)
stylefont(0) = "big1"
stylefont(1) = "big2"
stylefont(2) = "big3"
stylefont(3) = "big4"
stylefont(4) = "big5"
stylefont(5) = "big6"
stylefont(6) = "big7"

Dim colors(5)
colors(0) = "red"
colors(1) = "black"
colors(2) = "green"
colors(3) = "aqua"
colors(4) = "lime"
colors(5) = "blue"

Sub change()
   Dim i, tmpstyle
   tmpstyle = stylefont(0)
   For i = 0 to 5
      stylefont(i) = stylefont(i+1)
   Next

   stylefont(6) = tmpstyle

   For i = 0 to 6
      document.all("font"&(i+1)).className = stylefont(i)
   Next

   Randomize

   For i = 0 to 6
      document.all("font"&(i+1)).style.color = colors(round(rnd()*5))
   Next

   Call Window.setTimeout ("change",200)
End Sub
```

这段代码中，用数组变量 stylefont 定义了每个文字相对应的样式，数组变量 color 定义了 5 种颜色。过程 change 实现转换的过程，下面我们对其进行分析。

首先，定义局部变量 I 和 tmpstyle，I 是循环变量，tmpstyle 是一个中间变量，利用它来实现 7 个文字的样式的循环移动：

```
tmpstyle = styleoffont(0)
For i = 0 to 5
   stylefont(i) = stylefont(i+1)
Next

styleoffont(6) = tmpstyle
```

执行了这段代码后，正好将数组 stylefont 中存储的样式依次向后移动了一位，将最后一种样式移动到数组的第一个元素中。

之后，使用循环给 7 个文字赋予新的样式：

```
For i = 0 to 6
    document.all("font"&(i+1)).className = stylefont(i)
Next
```

然后初始化随机发生器，随机决定 7 个文字的颜色。

```
Randomize
For i = 0 to 6
    document.all("font"&(i+1)).style.color = colors(round(rnd()*5))
Next
```

函数 rnd 返回一个小于 1 但大于或等于 0 的值，所以四舍五入函数 round 带参数 rnd()*5 仍然是 0~5 之间的一个整数。只不过不能确定它是 0~5 中的哪一个。

在最后，这也是关键的部分，使用 setTimeout 方法循环执行本过程，以实现不断变幻的效果。

```
Window.setTimeout ("change",200)
```

在过程期间，不断地移动每个字体的样式，譬如开始时第 1 到第 7 个文字对应的是第 1 到第 7 号样式，循环几次后，属于第 1 个文字的样式可能就移动到了第 1 个文字上面，当然移动是顺序的、连贯的，这样才能由样式的不同产生波浪传递的效果。改变样式的通用格式是：

```
object.style.stylename = stylevalue
```

其中 Object 是一个对象或是对象的引用，每个可视元素都有样式 style 属性。stylename 是样式名称，如：Font-size、Font-style、Border、Cursor 等等。stylevalue 是要赋的值。使用这种方法可以设定每个单独的元素的样式。而使用下面的方法可以成批地赋予元素样式，事先需要把要赋的属性总结在一个样式类中，如前面的 big1~big7。然后用：

```
object.classname = classname
```

把样式名赋给对象引用的 classname 属性。在这个例子中，我们使用了两种方法，前者的例子是：

```
document.all("font"&(i+1)).style.color = colors(round(rnd()*5))
```

后者的例子是：

```
document.all("font"&(i+1)).className = stylefont(i)
```

11.2.3 实例效果与总结

利用上述代码，可以实现如图 11.3 所示的效果。

图 11.3 彩色波浪文字运行效果

这个例子中我们主要用到了下面 3 个知识点：

- 样式单的定义与使用。
- 使用脚本访问样式单。
- 在网页中动态的改变元素的属性。

1．样式单的定义与使用

样式单的使用实际上是 HTML 的内容，不是本书的重点，但是它的使用频率很高，是一项很重要的内容。我们在这里简单地解释一下。样式单有 3 种使用方法：内联式、嵌入式和链接式。

内联式的使用方法为：

```
<TAG style="attribute1:value1">
```

它是利用 HTML 的标签<TAG>的样式属性定义样式的，也是一种最简单的方式。它只对一个标记起作用。

嵌入式的使用方法为：

```
<HTML>
<HEAD>……</HEAD>
<STYLE>……</STYLE>
<BODY>……</BODY>
</HTML>
```

这是定义样式的一般方法，采用这种方法，可以定义通用样式类，可以为个别标签定义样式，还可以为同一标签定义不同的样式，例如：

```
<STYLE>
<!--
A:link {color : #000000; font-size: 12px; text-decoration: none}
A:visited { color: #000000; font-size: 12px; text-decoration: none}
A:active { color: #cc0000; font-size: 12px; text-decoration: none}
A:hover { color: #000099; text-decoration: underline}
-->
</STYLE>
```

我们通过这个样式单来控制整个网页中处于各种状态的超链接的颜色、字体和是否带下划线。

A:link 表示未被选中的超链接，将它的属性设置为黑色、12 像素大小，不带下划线。

A:visited 表示已经被浏览过的超链接，它的属性设置同前。

A:active 表示当前被选中的超链接，将它的属性设置为红色、12 像素大小，不带下划线。

A:hover 表示当鼠标放在这个超链接上但是还没移走的时候的超链接，它的属性设置为蓝色、12 像素大小，带下划线。

我们看到许多网页上的超链接都被控制了不显示下划线或者是某种特殊的字体或颜色，基本上都是通过这种方法实现的。

链接式的使用方法是：

```
<HEAD>
<LINK rel=stylesheet href=”stylefilename.css” type=”text/css”>
</HEAD>
```

它通过外部定义样式，再在文档内部引用的方法实现样式调用。使用这种方法，Web 页面在加载本身的同时，还将加载所指定的样式页，并用它对 Web 页进行处理。

当链接式样式与嵌入式样式和内联式样式三者之间发生冲突时，浏览器将采用内联式样式高于嵌入式样式，嵌入式样式高于链接式样式的优先级次序对 HTML 标记进行修饰。

2．使用脚本访问样式单

使用脚本可以在程序中控制所有元素的所有样式属性。使用脚本访问样式单属性的格式为：

```
Object.style.styleproperty
```

Object 表示网页上的任何一个元素，可以是 Document.body 这样的样式，也可以是某个元素的 ID，如 Text1。

style 表示 Object 拥有样式属性。

styleproperty 表示需要引用的样式属性，它们可以被在设计时期和程序运行时期自由赋值。

3．在网页中动态地改变元素的样式

改变网页上元素的属性有两种方法：

（1）在程序运行中，给每个元素的某个样式属性赋新值，比如本例中可以改变它的外观，这可以通过脚本访问方式属性的方法，格式为：

```
object.style.styleproperty
```

详细的解释上边已经有了，不再赘述。

（2）集体赋值法，就是通过改变 classname 属性将元素的一整套样式属性改变。一般的讲，有 style 属性的元素都有 classname 属性，因此这两种方法实际上是同一种方法的两个方面而已。希望通过这个实例，大家能够学会灵活地应用它们。

11.3 黑夜里的手电筒

11.3.1 实例目标与技术要点

在本实例中，我们将实现这样的效果：在网页上，只有在鼠标出现的地方，它的周围才有椭圆的光束将其范围内的文字照亮，而且随着鼠标的移动，这束光就跟着移动，就好像在黑夜里的手电筒一样。

本例要点：

- DIV 标记。
- 光照滤镜。
- Addambient 方法的使用。
- Addcone 方法的使用。
- MoveLight 方法的使用。

11.3.2 实例过程

首先，我们在 HTML 文件中加入一段文字，当然，这段文字可以根据读者的喜好随便添加，我们将这段文字放在<DIV>标记对中。为了效果明显，使用<BODY>标记的 Bgclor 属性将网页的背景设为黑色。为了页面的整齐，将想要显示的文字规范在一个表格内。基本的 HTML 页面如下：

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

```
<TITLE>黑夜里的一束光</TITLE>
<META content="text/html; charset=gb2312" http-equiv=Content-Type>
</HEAD>
<BODY bgColor=black>
<CENTER>
<H1><FONT color=#ffffff>黑夜里的一束光 </FONT></H1>
</CENTER>
<DIV id=mylightobject >
<CENTER>
<TABLE>
<TR>
  <TD>
  <FONT color="#ffffff">
  从这儿走，<BR>
  你要翻过五十座山，<BR>
  越过三十条河。<BR>
  在第三十条河上有一座桥，<BR>
  过了桥，<BR>
  你会看见一条小路。<BR>
  顺着小路往下走，<BR>
  路的尽头有一间屋子，<BR>
  门前种了一排高大的橡树。<BR>
  你拾三颗左手边第二株树下的鹅卵石，<BR>
  扔进右手边第一株树的树洞里，<BR>
  屋门就会开了。<BR>
  </TD>
</TR>
</TABLE>
</FONT>
</DIV>
</CENTER>
</BODY>
</HTML>
```

到现在，还没有添加任何脚本程序，先预览一下页面，很简单，只是按照我们规定的样式显示了这些文字，如图 11.4 **所示。**

HTML **中的**<DIV>**标记是在** HTML **文件中指定了一个容器，实际上就是** HTML **文件中的一个块。我们用它的** ID **属性来标识这个块** ID=mylightobject **，在这个块中放入想要显示的文字，那么以后可以通过操作这个块对象来操作这些文字了。**

下面开始为页面添加脚本。

在页面加载时，为前面定义了的块对象 Mylightobject **设置一个环境光源，颜色为** 0，0，255，**强度为** 30，**此外还需设置一个点光源。**

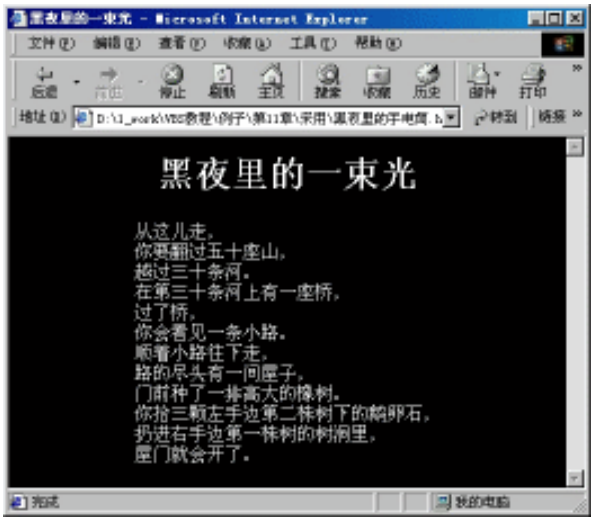

图 11.4　没有动态效果的页面

```
Sub window_onload()
    Call mylightobject.filters.light(0).addambient(0,0,0,30)
    Call mylightobject.filters.light(0).addcone(400,400,200,100,100,200,204,200,80,10)
End Sub
```

当鼠标移动的时候，移动光源，使光源的坐标随鼠标移动而改变。

```
Sub document_onmousemove()
    Call mylightobject.filters.light(0).movelight(1,window.event.x,window.event.y,0,-1)
End Sub
```

到这里，还有一项重要的属性没有设置，就是块对象 Mylightobject 的光照属性 Light，它的作用是在该对象所容纳的内容上建造一种灯光的效果。同时，还需要设置的就是该块对象的大小了。所以需要将<DIV id=mylightobject >**替换为：**

```
<DIV id=mylightobject
STYLE=" filter: light; height: 360px; left: 0px; position: relative; top: 20px; width:100% ">
```

好了，现在我们就完成所有的工作了，看一看效果是不是很酷？

11.3.3　实例效果与总结

该实例的运行效果如图 11.5 所示。

在这个例子中，主要涉及到以下几个知识点：

1．DIV 标记

DIV **标记的功能是把其中的文本当成一个整体，使之居左、居右或居中，显然**<DIV>**具有 align 属性，固而也可以有 left、right 和 center 3 种赋值。其实 DIV 标记由许多属性、方法和事件。本例用到的属性有：**

图 11.5　黑夜中的手电筒运行效果

id：标识这个对象的字符串。

height：设置该对象的高度。

left：设置该对象相对于文档左边缘的距离。

position：定义如何放置 DIV，有两种取值，relative 是指 DIV 与其他 HTML 元素的相对位置；absolute 则是指 DIV 在窗口中的绝对定位。

top：得到该对象相对于文档顶端的距离。

width：设置该对象的宽度。

filter：为该对象设置滤镜效果。

2．光照滤镜

在这个实例中，我们主要使用了光照滤镜 Light Filter 这种技术。

通过设置光照滤镜 Light Filter 的属性，可以控制光源的虚拟位置，还可以控制光的焦点的 x、y 坐标以及光的类型（比如是点光源还是锥形光源等）、颜色和强度等等。在每一个滤镜中，最多能够指定 10 个光源，如果要想在一个页面上产生 10 个以上的光源，必须使用多个滤镜。下面我们介绍一下该滤镜几个方法的使用。

（1）Addambient：为光照滤镜设置一个环境光源，其语法为：

```
object.filters.item("DXImageTransForm.Microsoft.Light").addAmbient(iRed, iGreen, iBlue, iStrength)
```

参数说明：

- iRed：该参数必选，用来设定颜色中的红基本色成分，取值是 0~255 之间的整数。
- IGreen：该参数必选，用来设定颜色中的绿基本色成分，取值是 0~255 之间的整数。
- IBlue：该参数必选，用来设定颜色中的蓝基本色成分，取值是 0~255 之间的整数。

- IStrength：该参数必选，用来设定光线的强度，它的取值从弱到强，可以是 0~100 之间的整数。所指定的强度是针对于目标坐标而言的。

环境光源是一束垂直于页面的没有方向的平行光束，可以为环境光源设置各种各样的颜色。

（2）Addcone：为光照滤镜设置一个锥形光源，其语法为：

```
object.filters.item( "DXImageTransForm.Microsoft.Light").addCone(iX1, iY1, iZ1, iX2, iY2, iRed, iGreen, iBlue, iStrength, iSpread)
```

参数说明：

- iX1：必选参数，整数，指定光源的 x 坐标。
- iY1：必选参数，整数，指定光源的 y 坐标。
- iZ1：必选参数，整数，指定光源在 z 轴线的高度。
- iX2：必选参数，整数，指定目标光源的 x 坐标。
- iY2：必选参数，整数，指定目标光源的 y 坐标。
- iRed ：必选参数，用来设定颜色中的红基本色成分，取值是 0~255 之间的整数。
- IGreen：必选参数，用来设定颜色中的绿基本色成分，取值是 0~255 之间的整数。
- IBlue：必选参数，用来设定颜色中的蓝基本色成分，取值是 0~255 之间的整数。
- IStrength：必选参数，用来设定光线的强度，它的取值从弱到强，可以是 0~100 之间的整数。
- iSpread ：必选参数，整数，用来指定光源的垂直位置和对象表面之间的角度。这个角度的取值可以是 0°~90°。越小的值产生的锥形光源就越小，大一些的取值将会产生一个倾斜的椭圆或者一个圆的光盘。

锥形光线随着与目标位置的距离的增加而消退。光线在它的焦点附近显示为一束非常强烈的光，然后随着距离的增加而逐渐减弱。

（3）Movelight：移动锥形光源的焦点或移动点光源的源点。其语法为：

```
object.filters.item("DXImageTransForm.Microsoft.Light").moveLight(iLightNumber, iX, iY, iZ, fAbsolute)
```

参数说明：

- iLightNumber：必选参数，整数，指定代表一个光源的标识符。
- IX：必选参数，整数，指定光源的 x 坐标。
- IY：必选参数，整数，指定光源的 y 坐标。
- IZ：必选参数，整数，指定光源在 z 坐标上的高度。
- FAbsolute：必选参数，布尔值，指定光源相对于初始位置是相对移动还是绝对移动。

掌握了以上的内容，就可以做出各种各样的具有光照效果的网页来了，我们再举一个简单的例子，代码如下：

```
<HTML>
<HEAD>
<TITLE>光照效果</TITLE>
</HEAD>
<STYLE>
.aFilter
{background-color: #FFFFFF;
font-size:12px;
filter:light();
color: cyan;
width: 120;}
</STYLE>

<SCRIPT Language="VBScript">
Sub fnInit()
    Dim iX2,iY2
    iX2=oDiv.offsetWidth
    iY2=oDiv.offsetHeight
    oDiv.filters(0).addCone 0,0,1,iX2,iY2,255,0,0,20,180
End Sub
</SCRIPT>
<BODY onLoad="fnInit()">
<DIV class="aFilter" id="oDiv">
    这段文本被红色的锥形光源照射，可不是一幅背景图片哟。
</DIV>
</BODY>
</HTML>
```

运行一下看看，效果如图 11.6 所示，是不是很棒？

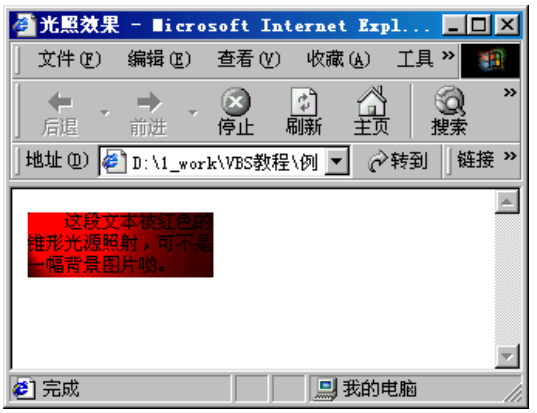

图 11.6　光照效果

11.4 动 态 按 钮

11.4.1 实例目标与技术要点

本例设计两种动态按钮，一种是能伸能缩的按钮，另一个是走马灯式的滚动按钮。这里是通过普通按钮上面的文字的变化，来达到动态效果的。

本例要点：

- input 标记的使用。
- mid 函数的使用。
- setTimeout 函数和 clearTimeout 函数的使用。

11.4.2 实例过程

该程序的页面设计很简单，下面是它的 HTML 部分。

```
<HTML>
<HEAD>
<TITLE>动态按钮</TITLE>
</HEAD>
<BODY>
<FORM name="frmCmd" onSubmit="null">
<P align=center>
    <INPUT type=button name="cmdScroll" align="center" value="">
<P align=center>
    <INPUT type=button name="cmdexpand" align="center" value="">

<SCRIPT Language="VBScript">
<!—
-->
</SCRIPT>
</FORM>
<P> </P>

</BODY>
</HTML>
```

使用上述代码建立两个按钮，按钮的值为空。下面为其添加脚本代码。

1．添加滚动按钮的脚本代码

产生滚动效果的基本思想是：充分利用按钮标题文字存储于按钮元素 value 属性中且可以在程序中改变的特点，利用每次倒换字符串的最后一个字符到字符串的首部的方法实现滚动的效果（实现从左到右的滚动）。这个方法也可以倒过来实现，即每次把字符串的头

一个字符放到最后一位上（实现从右到左的滚动），字符串的长度不会因此而改变，但字符串却好像滚动了起来。我们在这个例子中使用第二种方法，也就是实现从右到左的滚动。

下面是其实现的脚本代码部分：

```
Dim tid
Dim msg
msg = "白日依山尽，黄河入海流。"
Dim lenScroll
lenScroll = len(msg)
Sub scroll
        msg = mid(msg,2,lenScroll-1) & mid(msg,1,1)
        document.frmCmd.cmdScroll.value = msg
        tid = setTimeout("scroll()",200)
    End Sub
Call Scroll
```

首先定义计时器循环变量 I 和用于显示的按钮文字 msg，变量 lenScroll 取得按钮文字的长度，过程 scroll 用来实现文字的滚动。语句：

```
msg = Mid(msg,2,lenScroll-1) & Mid(msg,1,1)
```

将按钮文字的第一个字符倒到最后一个上去，并且把之后的值赋给 msg，然后使用 document.frmCmd.cmdScroll.value = msg 语句重新给按钮赋值。我们注意到，在表单内部定义的按钮元素，不能通过它的 ID 号直接访问，而必须通过 frmCmd.cmdScroll 这样的形式来访问。原因是除了 Form 元素外，网页上其他元素的 ID 都是平级的，但是一旦有了表单元素包含于 Form 元素中，就必须通过它去访问。它体现了 Form 元素的特殊性，它有点像一个窗口，使其内部成为一个独立的成分，而它是唯一让外部访问的内部的途径。

要保证产生动态效果还是要用 setTimeout 函数，可见这个函数在动态设计中有着举足轻重的作用。

2．添加伸缩按钮的脚本代码

产生伸缩效果的基本思想也是从改变按钮的 Value 属性开始的。设计一个初始字符串，每次截取这个字符串的一部分用于按钮标题的显示，截取部分由大到小，再有小到大，这样一来就产生了伸缩的效果。

脚本代码如下：

```
Dim i , flag, max, lenExpand
定义循环变量，这个变量决定每次所要截取字符串的长度
i       = 0
定义增量变量
flag = 1
定义用于显示按钮的文字
msg1 = "让我们一起来做运动吧！"
定义按钮长度
```

```
    lenExpand = len(msg1)
    '定义最大左侧截取长度
max = lenExpand/2
'实现伸缩按钮文字变换的过程
    Sub expand
      '循环变量递增
      i = i + flag
      '如果循环变量增至大于最大截取长度，则循环变量减 1
      If (i>max) Then
          i = i -1
          flag = -1
      End If

      '如果循环变量递减至小于最小截取长度 1 时，使循环变量增 1
      If (i<1) Then
          i = i + 1
          flag = 1
      End If

      '截取用于显示的字符串子串
      msg2 = Mid(msg1,i,lenExpand-2*i)
      '显示截取后的字符串
      frmCmd.cmdexpand.value = msg2
      '循环产生伸缩效果
      tid = setTimeout("expand()",200)
    End Sub
    '外部触发伸缩按钮的过程
    Call expand
```

程序中使用了一个技巧，每次截取字符串都用的是循环变量 I，但每次调用 Expand 过程时 I 的值都要发生变化。它的变化是以步长 Flag 为增量的，产生动态伸缩效果，其中重要的一环就是 Flag 变量，一开始它是正值，循环变量递增，而到了最大截取长度，它就变成了负值；当循环变量递减到小于最小截取长度（因为 I 如果小于 1，那么使用 mid 函数时程序就会出错）时，它又变为正值。所以截取的字符串的长度一会儿变大一会儿变小，如此循环就产生了预期的动态伸缩效果。

11.4.3 实例效果与总结

利用上述代码，我们可以实现如图 11.7 和图 11.8 所示的效果。

本实例介绍了利用字符串函数截取和按钮动态赋值相结合的方法，实现了动态按钮效果。本例的重点是<INPUT>标记的使用、Mid 函数的使用和 setTimeout 函数的使用。

图 11.7　动态按钮显示抓拍 1

图 11.8　动态按钮显示抓拍 2

<INPUT>标记是属于 HTML 中的内容，它规定了一种输入控制的窗体。关于该标记在本书的第 1 章有较为详细的介绍，这里就不多说了。重要的是在过程中动态地为按钮的 Value 赋值，也就是改变它的显示文字。再强调一遍，在表单内部定义的按钮元素，不能通过它的 ID 号直接访问，而必须通过 Formname.comname。这样的形式来访问。因为除了 Form 元素外，网页上其他元素的 ID 都是平级的，但是如果有表单元素包含于 Form 元素中，就必须通过它去访问。

至于 Mid 函数和 setTimeout 函数我们在前面都有介绍，就不再重复了。

11.5　测试点击速度

11.5.1　实例目标与技术要点

在这个实例中，我们将做一个小游戏：通过点击复选框，测试一下你使用鼠标的熟练

程度。具体地说，我们将测试在 20 秒内能够选中几个复选框，当时间超过 20 秒后，所做的选择都不再生效。

本例要点：

- 表单中复选框属性 Checked 的使用。
- document 对象的 Write 方法的使用。
- DateDIff 函数的使用。

11.5.2 实例过程

我们首先制作页面，本例需要一个文本框、10×10 个复选框和一个按钮。我们将文本框控件命名为 Num，100 个复选框的名字分别命名为：

element00……element09
element10……element19
……
element90……element99

当然对于这 100 个复选框控件如果一个一个往上手写是很不明智的做法，尽管你可以这么做。这里，我们使用嵌套的 For…Next 的语句和 Document 的 Write 方法输出这 100 个复选框控件。

```
For i = 0 to 9
    For j = 0 to 9
        document.write("<INPUT TYPE=""checkbox"" name = ""element")
        document.write (i & j & """ ")
        document.write("onClick=""display(document.frmboxs.element")
        document.write (i & j & ")"">")
    Next
    document.write("<BR>")
Next
```

使用上述语句输出的是一个 10 行 10 列的正方形。onClick=display(document.frmboxs.elementij)表示当用户触发该控件的 Click 事件时，将调用过程 display，这个过程需要一个参数，该参数表示当前被单击控件的名称。

我们将按钮控件的名称命名为 cmdRestart，它的作用是用来重新开始这个小游戏。制作好的页面如图 11.9 所示。

下面开始为页面添加累计点击次数的脚本代码。

```
'定义变量并初始化
'total 表示选中的复选框个数
'playflag 标志游戏是否开始
'starttime 记录游戏开始的时间
Dim total,playflag,starttime
total = 0
```

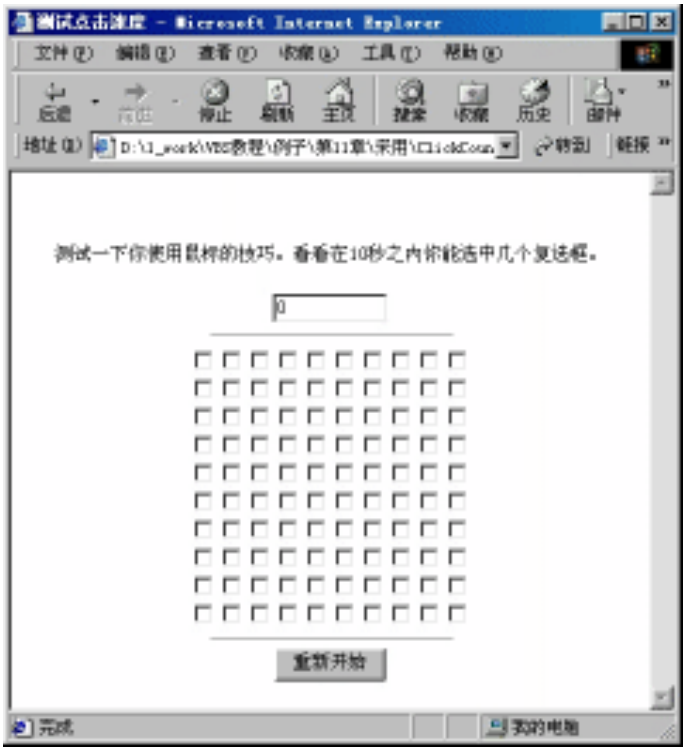

图 11.9　测试点击速度页面

```
playflag = false

Sub playflag (element)
    Dim now
    '如果游戏没有开始，使之开始，并将当前系统时间存在 starttime 中
    If not playflag Then
        playflag = true
        startTime = time()
    End If

    '如果当前时间已经超过开始时间 20 秒，提示警告信息，显示选中复选框的个数
    '并且将当前选中的那个复选框的选择标志清空，然后退出该过程
    If DateDIff("s",startTime,time()) > 20 Then
        Alert ("时间到！您在 20 秒之内选中了" & document.frmboxs.num.value & "个复选框！")
        element.checked = not element.checked
        Exit Sub
    End If

    '如果用户选中复选框则将点击次数加 1
'否则，用户不小心将原先选中的复选框清空，复选框不再加 1，
'该复选框不能重置
```

```
        If element.checked Then
            total = total + 1
        Else
            element.checked = true
        End If
        '将文本框的值设置为选中复选框的总数
        document.frmboxs.num.value = total
    End Sub
```

以下是重新开始的脚本代码。功能是将所有的控件和变量的值初始化。

```
    Function restart(Form)
        total = 0
        play = false
        For i = 1 to 100
            Form.elements(i).checked = false
        Next
        document.frmboxs.num.value = 0
    End Function
```

本例比较简单，程序很容易理解，所以对程序的过程就不再详细描述了。

11.5.3 实例效果与总结

测试本例，到指定时间即会弹出对话框，提醒时间到，且告诉选中了几个复选框，如图 11.10 **所示。**

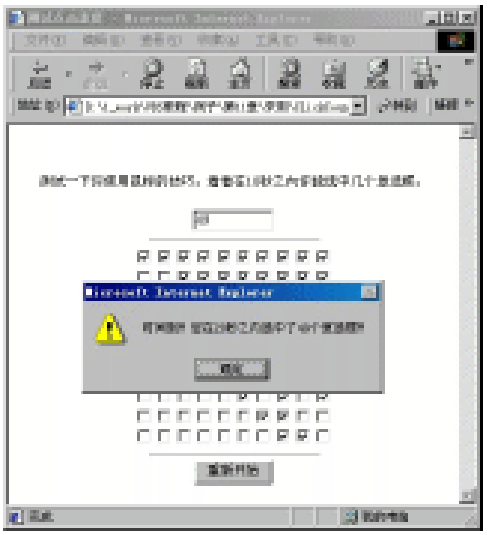

图 11.10 **测试点击速度演示效果**

本例的重点是：表单中复选框属性 Checked **的使用、**Document **对象的** Write **方法的使用和** DateDIff **函数的使用。**

1．复选框的属性 checked

我们主要使用了复选框的 Checked 属性，它的取值是 true 和 flase，分别表示是否选中还是没有选中。在很多时候我们都是通过控制这个属性的值来完成某种操作的。例如，当时间超过 20 秒之后，需要将用户选中的复选框作废，则使用 element.checked = not element.checked。

2．使用 Document 对象的 Write 方法

Document 的 Write 方法用于向文档中写入文本。一般如果输出的内容中不包含变量，直接使用 HTML 中的标签就可以了，但是根据程序需要，使用 Document 的 Write 方法可以灵活地利用变量向文档中写入不同的内容。例如，使用下面两种方法的效果是一样的。

- <P>hello world!</P>
- <SCRIPT language="VBScript">
  document.write "<P>hello world!</P>"
  </SCRIPT>

从上述例子中我们看到，使用 Document 的 Write 方法并没有什么优点。但是如果需求发生变化的时候又如何呢？比如需要输出 7 条相同的内容，只是它们的字体大小不一样而已。在单纯的 HTML 文件中，我们只能这样写：

```
<FONT size=7>hello world!<BR></FONT>
<FONT size=6>hello world!<BR></FONT>
<FONT size=5>hello world!<BR></FONT>
<FONT size=4>hello world!<BR></FONT>
<FONT size=3>hello world!<BR></FONT>
<FONT size=2>hello world!<BR></FONT>
<FONT size=1>hello world!<BR></FONT>
```

而使用 Document 的 Write 方法只需要写一次就可以完成了：

```
<SCRIPT language="VBScript">
For i = 7 to 1 step -1
    document.write "<FONT size=" & i &">hello world!<BR></FONT>"
Next
</Script>
```

这里使用了一个变量 i 来表示字体的大小，然后使用循环语句写出来，就达到上面用了 7 行代码才完成的功能。显示页面如图 11.11 所示。

对于前面测试点击速度的例子，我们需要 100 个基本相同的元素，所以使用这种方法是一种合理的选择。

图 11.11 使用 Write 方法向文档中写入信息

3．使用 DateDIff 函数

DateDIff 函数用于判断在两个日期之间存在的指定时间间隔的数目。例如可以使用 DateDIff 计算两个日期相差的天数，或者当天到当年最后一天之间的星期数。本例中使用 DateDIff("s",startTime,time())返回当前时间和游戏开始时间的时间差，其中第一个参数“s”表示返回两个时间相差的秒数，如果第一个时间晚于第二个时间，则返回负值。详细的讲解请参阅第 5 章。

11.6 漫游网页的小精灵

11.6.1 实例目标与技术要点

我们看到许多网站的主页上都有飞来飞去的广告小图片，碰到框架的四周时会反弹改变方向，而且当你拖动滚动条时小图片也会同步移动，这样可以保证小图片始终可见，再给小图片加上一个超链接将访问你网页的人带入另一番天地。我们就通过本例来实现这个功能。

在技术实现上，我们使用了 DHTML 中的一些知识，本例要点：

- scrollleft 属性。
- scrolltop 属性。
- offsetwidth 属性。
- offsetheight 属性。
- pixelleft 属性。
- pixeltop 属性。

11.6.2 实例过程

本效果实现的原理是：在网页中用<DIV>标记将小图片放入“Img 容器”内，然后使

用 VBScript 编程控制“容器”的移动从而使小图片在网页框架中四处漫游，源代码如下：

```
<HTML>
<HEAD>
<SCRIPT language="VBScript">
    '图片当前 X，Y 坐标位置及所用图片的宽度、高度
    Dim PosX, PosY, ImgWidth,ImgHeight
    '图片移动的增量
    Dim MoveStepX , MoveStepY
    '图片上下移动的方向指示，1 表示向下移动，－1 表示向上移动
    Dim MoveOrient
    '控制方向改变的快慢
    Dim SpeedOForientChange
    '控制某一方向运动的时间
    Dim TimeOfMoveOrient

  '进行初始化
    TimeOfMoveOrient = 0
    SpeedOForientChange = 2
    MoveOrient = 1
    ImgWidth = 100
    ImgHeight = 100
    MoveStepX = 2
    MoveStepY = 2
    PosX = 150
    PosY = 10

    Sub MoveImg                    '实现图片移动的子过程
      Dim Doc_ScrollLeft
      Dim Doc_ScrollTop
      Dim Doc_OffsetWidth
      Dim Doc_OffsetHeight

      TimeOfMoveOrient = SpeedOForientChange*(Rnd - 0.5)+TimeOfMoveOrient
      If TimeOfMoveOrient > 3 Then
          MoveStepX = -MoveStepX
          TimeOfMoveOrient = 0
      End If'

      If TimeOfMoveOrient <- 3 Then
          MoveStepX = -MoveStepX
          TimeOfMoveOrient = 0
      End If
```

```
        'x 坐标加一个图片的增量
        PosX= PosX + MoveStepX
        'y 坐标加一个图片的增量（当移动方向 MoveOrient 为 1 时，增量为正值，
'向下移动；当移动方向 MoveOrient 为-1 时，增量为负值，向上移动）
        PosY = PosY + MoveOrient * MoveStepY

        '取 body 到最左边界的距离
        Doc_ScrollLeft = window.document.body.scrollleft
        '取 body 到最上端的距离
        Doc_ScrollTop = window.document.body.scrolltop
        '取 body 相对于左上角的宽度
        Doc_OffsetWidth = window.document.body.offsetwidth
        '取 body 相对于左上角的高度
        Doc_OffsetHeight = window.document.body.offsetheight

        If PosX <= Doc_ScrollLeft Then
            PosX = Doc_ScrollLeft
            MoveStepX=-MoveStepX
        End If

        If PosX >= Doc_ScrollLeft + Doc_OffsetWidth - ImgWidth Then
            PosX = Doc_ScrollLeft + Doc_OffsetWidth - ImgWidth
            MoveStepX=-MoveStepX
        End If

        If PosY <= Doc_ScrollTop Then
            PosY = Doc_ScrollTop
            MoveOrient=-MoveOrient
        End If

        If PosY >= Doc_ScrollTop + Doc_OffsetHeight - ImgHeight Then
            MoveOrient = -MoveOrient
            PosY = Doc_ScrollTop + Doc_OffsetHeight - ImgHeight
        End If

        '确定小图片的位置
        document.all.Img.style.pixelleft= PosX
        document.all.Img.style.pixeltop= PosY

        Call settimeout("MoveImg", 10)
    End Sub
</SCRIPT>
```

```
<TITLE>漫游的小图片</TITLE>
<META http-equiv="Content-Type" content="text/html; charset=gb2312">
</HEAD>
<BODY bgcolor="lavEnder" onload="MoveImg">
<DIV id="Img" style="backgound-color: transparent; border-bottom: #000000 1px; border-left: #000000 1px; border-right: #000000 1px; border-top: #000000 1px; height: 60px; left: 232px;position: absolute; top: 151px; visibility: visible; width: 80px"  >
<A href="http://www.microsoft.com">
<IMG border=0 src="example.gIf" width=65 >
</A>
</DIV>
<P align=center><STRONG>这世界真美</STRONG></P>

<P><FONT size=2>   
你们走的时候，很想洒脱很想倜傥。你们真正走的时候，却是在夜晚悄悄地走的，还撑了一把漂亮的伞。我想我只有投奔雨了，不料雨停后的草地是馨香透亮的，微风亲切地拂面指耳。在这个时刻，世界真美。</FONT></P>

<P><FONT size=2>   
<!--
此处写上你喜欢的任意文字。
-->
</FONT></P>

</BODY>
</HTML>
```

11.6.3　实例效果与总结

运行本例，将会得到如图 11.12 所示的效果。

图 11.12　漫游网页的小精灵

在本例中，我们是通过控制“盛放”小图片的 DIV 容器来改变小图片的位置，并为<DIV>标记使用了内联式的样式单<DIV id="Img" style="……">，样式单是改变元素样式、位置和内容的基础，也就是动态 HTML 的基础，占有非常重要的位置，通过语句：

```
document.all.Img.style.pixelleft= PosX
document.all.Img.style.pixeltop= PosY
```

完成改变图片的位置的任务，除此之外，还需要理解几个属性：scrollleft 属性、scrolltop 属性、offsetwidth 属性、offsetheight 属性、pixelleft 属性和 pixeltop 属性，掌握了这几个属性的使用方法，你就会做出更精彩的网页来。

（1）scrollleft 属性：获取或设置对象的左端与窗口中可见内容的最左端之间的距离。它的返回值是以像素为单位的，其默认值为 0。它的值实际上等于对象的内容滚动的水平距离。如果该对象中没有包含超出显示区域的更多内容，这时 Scrollleft 的值就为 0，这种情况下不显示水平滚动条。所以对于没有滚动条的对象该属性的返回值总是 0。对于这种情况，设置这个属性的值是没有意义的。

（2）scrolltop 属性：获取或者设置对象的顶端与窗口中可见内容的最顶端之间的距离。它的值等于对象的内容滚动的垂直距离。所以对于没有垂直滚动条的对象该属性的返回值总是等于 0，对于这种情况，设置它的属性值是不会有任何影响的。它的返回值是以像素为单位的，其默认值为 0。如果该对象中没有包含超出显示区域的更多内容，这时 Scrolltop 的值就为 0，这种情况下不显示垂直滚动条。

（3）offsetwidth 属性：得到对象距离其原点的宽度。

（4）offsetheight 属性：得到对象距离其原点的高度。

（5）pixelleft 属性：获取或设置对象距左的距离。语法为：

```
object.style.pixelLeft [ = iLeft ]
```

（6）pixeltop 属性：获取或设置对象距顶的距离。语法为：

```
object.style.pixelTop [ = iTop ]
```

上述这些属性基本都是控制对象在窗口中的各种不同位置的，学会使用这些属性，对于我们理解和制作动态的网页有很大的帮助，当然还有更多我们在这里没有提到的属性，读者如果有兴趣的话，可以参考其他的书籍或资料，也可以到微软的 MSDN（http://msdn.microsoft.com）上面看一看，上面有很详细的解释。

11.7 万 年 历

11.7.1 实例目标与技术要点

本实例将制作一个功能强大的万年历，之所以称之为万年历，是因为只要你输入或选择一个有效的日期格式，该日历就可以显示指定日期所在月的日历。而且，还可以向前翻，

或向后翻。

本例要点：

- 样式单的使用。
- VBScript 内部函数——日期函数的使用。
- 显示或隐藏对象的方法。

11.7.2　实例过程

本例采用的是动态生成 HTML 代码的方法，因此无需专门制作静态的 HTML 框架，可以直接编写脚本。

文档的主体部分并不复杂，主要包含了两个<DIV>容器，这两个<DIV>容器分别用来放置两种不同表现形式的日历。一种是显示某一个月的所有天数的日历，我们把它叫做 daycalEnder，另一种是显示某一年的所有月份的日历，我们把它叫做 monthcalEnder。其实在最初设计的时候，它们都只是一个定义了一些格式的空的框架， 并不包含任何内容，只有当用户选择了某个日期的时候，才往里动态的填东西。所以在它们没有任何实际意义的数据时，将它的属性设为不可见。它们的外观大致是如图 11.13、图 11.14 所示的样子。

图 11.13　daycalEnder 的外观

图 11.14　monthcalEnder 的外观

我们将网页中用到的元素，如表格、表格单元、链接等所采用的格式定义了一个样式表，代码如下：

```
<STYLE type=text/css>
TABLE.cal  { color: #c0c0c0; width: 200px }
TD.cal          { font-family: tahoma; font-size: 10pt; text-align: center }
INPUT.cal  { color: white; background: black; border: none; width: 20 }
A               { text-decoration: none; }
A.cal:link   { color: white; }
A.cal:visited       { color: white; }
A.cal:active        { color: white; }
A.cal:hover         { color: white; }
A.tdy:link   { color: #c5f9c5; }
A.tdy:visited       { color: #c5f9c5; }
A.tdy:active        { color: white; }
A.tdy:hover         { color: gray; }
```

```
A.tdz:link    { color: yellow; font-weight: bold; }
A.tdz:visited       { color: yellow; font-weight: bold; }
#daycalEnder       { left: 116px; position: absolute; top: 100px }
#monthcalEnder   { left: 116px; position: absolute; top: 100px }
</STYLE>
```

为了功能实现上的方便，我们引入了 3 个隐藏控件，使用这 3 个隐藏控件分别来记录将要显示的日期的年份、月份和天数，将其分别命名为 yy、mm、dd。

```
<INPUT name=yy type=hidden >
<INPUT name=mm type=hidden >
<INPUT name=dd type=hidden>
```

所以它的 HTML 文档的源码为：

```
<HTML>
<HEAD>
<TITLE>万年历</TITLE>
<HEAD>
<STYLE type=text/css>
TABLE.cal  { color: #c0c0c0; width: 200px }
TD.cal              { font-family: tahoma; font-size: 10pt; text-align: center }
INPUT.cal   { color: white; background: black; border: none; width: 20 }
A               { text-decoration: none; }
A.cal:link    { color: white; }
A.cal:visited       { color: white; }
A.cal:active        { color: white; }
A.cal:hover         { color: white; }
A.tdy:link    { color: #c5f9c5; }
A.tdy:visited       { color: #c5f9c5; }
A.tdy:active        { color: white; }
A.tdy:hover         { color: gray; }
A.tdz:link    { color: yellow; font-weight: bold; }
A.tdz:visited       { color: yellow; font-weight: bold; }
#daycalEnder       { left: 116px; position: absolute; top: 100px }
#monthcalEnder   { left: 116px; position: absolute; top: 100px }
</STYLE>
<DIV id=daycalEnder style="display: none">
<TABLE class="cal" bgColor=black cellPadding=0 cellSpacing=0>
   <TBODY >
   <TR height=25>
      <TD class="cal" colspan=2 align=left>
<A href="#" onclick="javascript: prevmonth()" class=cal> <<</A>
</TD>
```

```
      <TD class="cal" id=monthandyear    colspan=3 align=center>月 年</TD>
      <TD class="cal" colspan=2 align=right>
          <A href="#" onclick="javascript: Nextmonth()" class=cal>>></A>
     </TD>
</TR>
   <TR    bgColor=LightSlateGray>
      <TD class="cal">Sun</TD>
      <TD class="cal">Mon</TD>
      <TD class="cal">Tues</TD>
      <TD class="cal">Wed</TD>
      <TD class="cal">Thur</TD>
      <TD class="cal">Fri</TD>
      <TD class="cal">Sat</TD>
   </TR>
   <TR height=20>

      <TD class="cal" id=11> </TD>
      <TD class="cal" id=12> </TD>
      <TD class="cal" id=13> </TD>
      <TD class="cal" id=14> </TD>
      <TD class="cal" id=15> </TD>
      <TD class="cal" id=16> </TD>
      <TD class="cal" id=17> </TD></TR>
   <TR height=20>
      <TD class="cal" id=21> </TD>
      <TD class="cal" id=22> </TD>
      <TD class="cal" id=23> </TD>
      <TD class="cal" id=24> </TD>
      <TD class="cal" id=25> </TD>
      <TD class="cal" id=26> </TD>
      <TD class="cal" id=27> </TD></TR>
   <TR height=20>
      <TD class="cal" id=31> </TD>
      <TD class="cal" id=32> </TD>
      <TD class="cal" id=33> </TD>
      <TD class="cal" id=34> </TD>
      <TD class="cal" id=35> </TD>
      <TD class="cal" id=36> </TD>
      <TD class="cal" id=37> </TD></TR>
   <TR height=20>
      <TD class="cal" id=41> </TD>
      <TD class="cal" id=42> </TD>
      <TD class="cal" id=43> </TD>
```

```
      <TD class="cal" id=44> </TD>
      <TD class="cal" id=45> </TD>
      <TD class="cal" id=46> </TD>
      <TD class="cal" id=47> </TD></TR>
    <TR height=20>
      <TD class="cal" id=51> </TD>
      <TD class="cal" id=52> </TD>
      <TD class="cal" id=53> </TD>
      <TD class="cal" id=54> </TD>
      <TD class="cal" id=55> </TD>
      <TD class="cal" id=56> </TD>
      <TD class="cal" id=57> </TD></TR>
    <TR height=20>
      <TD class="cal" id=61> </TD>
      <TD class="cal" id=62> </TD>
      <TD class="cal" id=63> </TD>
      <TD class="cal" id=64> </TD>
      <TD class="cal" id=65> </TD>
      <TD class="cal" id=66> </TD>
      <TD class="cal" id=67> </TD>
    </TR>
    </TBODY>
</TABLE>
</DIV>

<DIV id=monthcalEnder style="DISPLAY: none">
<TABLE class="cal" bgColor=black cellPadding=0 cellSpacing=0 >
    <TBODY>
    <TR height=25>
      <TD class="cal"    width=20>
          <A href="#" onclick="javaScript:prevyear()" class=cal><<<</A>
      </TD>
      <TD class="cal"    id=oneyear width=160 colspan=2>月 年</TD>
      <TD class="cal"    width=20>
           <A href="#" onclick="javaScript:Nextyear()" class=cal>>>>>></A>
      </TD>
  </TR>

    <TR>
      <TD class="cal"    bgColor=LightSlateGray colspan=4>请选择一个月份</TD></TR>

    <TR height=20>
      <TD class="cal" >
```

```
            <A class="cal" href="javascript: cmonth(1)"> Jan  </A>
        </TD>
        <TD class="cal" >
<A class="cal" href="javascript: cmonth(2)"> Feb </A>
</TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(3)"> Mar </A>
        </TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(4)"> Apr </A>
        </TD>
</TR>
    <TR height=20>
        <TD class="cal" >
             <A class="cal" href="javascript: cmonth(5)"> May </A>
        </TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(6)"> Jun </A>
        </TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(7)"> Jul  </A>
        </TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(8)"> Aug  </A>
        </TD>
</TR>
    <TR height=20>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(9)"> Sep </A>
        </TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(10)"> Oct  </A>
        </TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(11)"> Nov </A>
        </TD>
        <TD class="cal" >
            <A class="cal" href="javascript: cmonth(12)"> Dec </A>
        </TD>
    </TR>
    </TBODY>
</TABLE>
</DIV>
```

```
<INPUT name=yy type=hidden >
<INPUT name=mm type=hidden >
<INPUT name=dd type=hidden>

<BODY onload="init()" onmouseup="hidepcal()">
<P><FONT  size=2>您可以在该文本框内输入一个合法的日期，点击“显示日历”就可看到您输入日期所在的那月的日历。</FONT></P>
<P>
<INPUT name="mydate" value=" " >
<FONT size=2><STRONG>
<A href="javascript: ShowCalEnder(mydate)">显示日历</A>
<A href="javascript: hidepcal()">隐藏日历</A>
</STRONG></FONT> </P>
<P> </P>
</BODY>
</HTML>
```

在 daycalEnder **中，总共包含了** 6*7=42 **个空的单元格，这** 6 **和** 7 **代表的是什么含义呢？由于在一个月中，最多跨越** 6 **个星期，那么我们将** 6 **作为这个日历表格的行数；每个星期有** 7 **天，所以将** 7 **作为这个日历表格的列数。我们注意到每一个单元格** ID **的设置是有一定规律的，就像一个矩阵，实际上我们就是这个目的，使单元格按照如下的顺序排列：**

11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37
41	42	43	44	45	46	47
51	52	53	54	55	56	57
61	62	63	64	65	66	67

将来我们会按照这个规律依次往相应的单元格中填充数据，并且显示出来，以达到日历显示的目的。

同时，在 HTML **文档中，用了许多的**<A></A>**标记对，这些标记对的作用并不仅仅是超链接，大多数都是执行一个函数。下面我们分别解释一下这些链接的作用。**

<A href="#" onclick="javascript: prevmonth()" class=cal><<<</A>**表示用符号“** << **”来实现向前翻滚一个月，所以函数** prevmonth()**的作用就是显示前一个月的日历。**

<A href="#" onclick="javascript: Nextmonth()" class=cal>>>></A>**表示用符号“** >> **”来实现向后翻滚一个月，所以函数** Nextmonth()**的作用就是显示后一个月的日期。**

<A href="#" onclick="javaScript:prevyear()" class=cal><<<</A>**表示用符号“** << **”来实现向前翻滚一年，所以函数** prevyear()**的作用就是显示前一年的所有月份。**

<A href="#" onclick="javaScript:Nextyear()" class=cal>>>></A>**表示用符号“** >> **”来实现**

向后翻滚一年，所以函数 Nextyear()的作用就是显示后一年的所有月份。

<A class="cal" href="javascript: cmonth(1)"> Jan </A>表示用“ Jan ”来显示一月份的日历，相应的，<A class="cal" href="javascript: cmonth(2)"> Feb </A>表示用“ Feb ”来显示二月份的日历，以此类推，一直到 12 月。所以函数 cmonth(mnt)的作用就是显示 mnt 这个月的日历。

<A href="javascript: ShowCalEnder('mydate')">显示日历</A>表示显示日期参数 mydate 所在年份月份的日历。所以函数 ShowCalEnder(ddd)的作用就是显示 ddd 所在的那一年月的日历。

<A href="javascript: hidepcal()">隐藏日历</A> 表示将所有显示的日历隐藏。这是依靠过程 hidepcal()来实现的。

onload="init()" 用来将当前日期的年月日分别赋给隐藏的 3 个控件。

下面分别编写这几个函数的脚本。

1．Prevmonth()

Prevmonth()过程的功能是实现显示前一个月的日历。

```
Sub prevmonth()
    '使 daycalEnder 容器中的日历显示。
    showpcal()
    '将文本框中的日期值在当前的日期值的基础上减去 1 个月。以实现文本框中
    '的值与显示的日历相统一。
     document.all.mydate.value =
        FormatDateTime(DateAdd("m",-1,document.all.mydate.value),1)
    '用来记录需要显示月份的隐藏域的值减 1。如果在减 1 之后，月份的值变为
    '0，将日期置为前一年的 12 月。
    document.all.mm.value=document.all.mm.value-1
        If document.all.mm.value=0 Then
          document.all.mm.value=12
          document.all.yy.value=document.all.yy.value-1
    End If
    '显示日历
    Call pcalEndar()
End Sub
```

在这个过程中又调用了另外的两个过程——showpcal()和 pcalEndar()。这两个过程在多处被调用。我们有必要分别解释一下这两个过程。

showPCal()过程其实很简单，它只是使 daycalEnder 日历的显示隐藏而已，它只有一句代码，但是为了程序的可读性，我们仍然把它作为一个封装起来的过程来调用。

```
Sub showPCal()
    document.all("daycalEnder").style.display=""
End Sub
```

pcalEndar()过程是整个程序的核心部分，它实现的是将指定日期所在月的日历显示出来。

```
Sub pcalEndar()
  mnt=document.all.mm.value
  yyr=document.all.yy.value
  ddy=document.all.dd.value
  '返回某年月的 1 号是星期几
  fday=weekday(mnt & "/1/" & yyr)
  '判断该月有几天
  lday=dlookup(mnt, yyr)

  innerHTML="<a class='tdy' href='javascript: showmonthcalEnder()'> "& _
            mlookup(mnt) &   "   " & yyr & " </A>"
  document.all.monthandyear.innerHTML=innerHTML
  innerHTML="<a class='tdy' href='javascript: hidemonthcalEnder()'> "& _
          yyr & " </A>"
  document.all.oneyear.innerHTML=innerHTML

  For a=1 to 6                    '一月最多可以跨越 6 个星期
  For b=1 to 7                    '每周有 7 天
    c=c+1
    If c >= fday Then
       d=d+1
    Else
       d=0
    End If
    targetId=a & b
    If d<=lday Then
    '如果 d 等于今天，加重显示
      If d=day(now()) and int(mnt)=month(now()) and int(yyr)=year(now()) Then
    '以下 innerHTML 的含义是获取或设置对象的开始到结束之间的 HTML 描述。
         innerHTML="<a class='tdz'"
       Else
         innerHTML="<a class='cal'"
       End If
       innerHTML=innerHTML & "href='javascript: outday(" & d & ")'>& _
            nbsp;" & d & " </A>"
       If d=0 Then innerHTML=""
       document.all(targetId).innerHTML=innerHTML
    Else
       document.all(targetId).innerHTML=""
    End If
```

```
    Next
    fday=1
    Next
End Sub
```

这是本例中较为难懂的一个过程，其中它又调用了 5 **个过程——**mlookup (mnt)、dlookup(mm, yy)、showmonthcalEnder()、hidemonthcalEnder()**和** outday()**，不过它们都比较简单，代码分别如下：**

过程 dlookup()**的功能是判断某年的某月有多少天。**

```
Function dlookup(mm, yy)
    dlookup=31
    If mm<8 and mm/2=fix(mm/2) Then dlookup=30
    If mm>7 and mm/2<>fix(mm/2) Then dlookup=30

    If mm=2 Then
        lpyr=leapyear(yy)
        If lpyr=1 Then dlookup=29
        If lpyr=0 Then dlookup=28
    End If
End Function
```

其中调用了过程 leapyear(yy)**，它的作用是判断该年是否是闰年。**

```
Function leapyear(yyr)
leapyear=0
    If yyr/4=fix(yyr/4) Then leapyear=1
    If yyr/100=fix(yyr/100) Then leapyear=0
    If yyr/400=fix(yyr/400) Then leapyear=1
End Function
```

过程 mlookup()**的功能是将以阿拉伯数字表示的月份转换为简写的英文月份。**

```
Function mlookup(mnt)
Select Case mnt
    Case  1 mlookup="JAN"
    Case  2 mlookup="FEB"
    Case  3 mlookup="MAR"
    Case  4 mlookup="APR"
    Case  5 mlookup="MAY"
    Case  6 mlookup="JUN"
    Case  7 mlookup="JUL"
    Case  8 mlookup="AUG"
    Case  9 mlookup="SEP"
    Case 10 mlookup="OCT"
```

```
    Case 11 mlookup="NOV"
    Case 12 mlookup="DEC"
End Select
End Function
```

过程 showmonthcalEnder()**的功能是显示某年的所有月份的那种日历** monthcalEnder**。**

```
Sub showmonthcalEnder()
     document.all("daycalEnder").style.display="none"
     document.all("monthcalEnder").style.display=""
End Sub
```

过程 hidemonthcalEnder()**的功能是隐藏** monthcalEnder **格式的日历，显示** daycalEnder **格式的日历。**

```
Sub hidemonthcalEnder()
    showpcal()
    document.all("monthcalEnder").style.display="none"
End Sub
```

过程 outday()**所实现的功能是将用户选择的日期显示在日期输入框中。**

```
Sub outday(dd)
    mms=document.all.mm.value
    yys=document.all.yy.value
    dout = yys & "年" & mms & "月" & dd & "日"
    document.all.mydate.value = dout
End Sub
```

理解了过程 prevmonth**，下面的所有过程也就好理解了，实际上它们都是大同小异，因此，下面的部分过程我们只给出源代码，就不做太多解释了。**

2．Nextmonth()

Nextmonth()**所实现的功能是下一个月的日历。**

```
Sub Nextmonth()
     document.all.mydate.value =
          FormatDateTime(DateAdd("m",1,document.all.mydate.value),1)
    showpcal()
    document.all.mm.value=document.all.mm.value+1
 If document.all.mm.value=13 Then
       document.all.mm.value=1
       document.all.yy.value=document.all.yy.value+1
    End If
    Call pcalEndar()
End Sub
```

3．prevyear()

过程 prevyear()**的功能是显示前一年所有的月份。**

```
Sub prevyear()
document.all.mydate.value =
FormatDateTime(DateAdd("yyyy",-1,document.all.mydate.value),1)
    showmonthcalEnder()
    document.all.yy.value=document.all.yy.value-1
    Call pcalEndar()
End Sub
```

4．Nextyear()

过程 Nextyear()**的功能是显示后一年所有的月份。**

```
Sub Nextyear()
     document.all.mydate.value =
          FormatDateTime(DateAdd("yyyy",1,document.all.mydate.value),1)
     showmonthcalEnder()
     document.all.yy.value=document.all.yy.value+1
     Call pcalEndar()
End Sub
```

5．cmonth(mnt)

过程 cmonth(mnt)**的功能是显示参数** mnt **这个月的日历**

```
Sub cmonth(mnt)
     document.all.mydate.value = Year(document.all.mydate.value) & _
          "年" & mnt & "月" & Day(document.all.mydate.value) & "日"
     document.all.mm.value=mnt
     hidemonthcalEnder()
     Call pcalEndar()
End Sub
```

这个过程中调用了一个新的过程 hidemonthcalEnder()**，该过程的作用是使** monthcalEnder **这种格式的日历隐藏，使** daycalEnder **这种格式的日历显示。**

```
Sub hidemonthcalEnder()
    showpcal()
    document.all("monthcalEnder").style.display="none"
End Sub
```

6．ShowCalEnder(ddd)

过程 ShowCalEnder(ddd)**的功能是显示指定日期** ddd **所在月份的日历。**

```
Sub ShowCalEnder(ddd)
    ddate=document.all(ddd).value
```

```
    yys=year(ddate)
    mms=month(ddate)
    dds=day(ddate)

    If yys-document.all("yy").value<>0 or mms-document.all("mm").value<>0 Then
        document.all("yy").value=yys
        document.all("mm").value=mms
        pcalEndar()
    End If

    showPCal()
End Sub
```

在过程 ShowCalEnder(ddd)**中，又一次用到了** showPCal()**过程，前面我们已经作过介绍，它的功能是使** daycalEnder **日历的显示隐藏。**

7．hidepcal()

过程 hidepcal()**将日历隐藏。**

```
Sub hidePCal()
    document.all("daycalEnder").style.display="none"
    document.all("monthcalEnder").style.display="none"
End Sub
```

8．init()

进行初始化，将当前日期的年月日分别赋给隐藏的 3 **个控件，并且产生日历。**

```
Sub init()
    document.all.mm.value=month(now())
    document.all.yy.value=year(now())
    document.all.dd.value=day(now())
    Call pcalEndar()
      showPCal()
End Sub
```

到此为止，我们就将这个实例的实现过程分析完了，下面看一看它的运行效果吧。

11.7.3 实例效果与总结

这是一个功能较为完备的万年历，首先，打开这个文件时，将自动显示当天所在月份的日历，其中，当天的日期将加重显示，如图 11.15 **所示。**

如果点击日历最上端的年月份两端的“ << **”或“** >> **”可以实现日历向后翻一个月和向前翻一个月的功能，同时文本框中的日期发生相应的变化；单击其中的某一天，文本框中的日期也随之变化；单击年月份本身，将显示本年的所有月份，如图** 11.16 **所示。**

图 11.15　显示当天所在月份的日历

图 11.16　显示一年中所有的月份

在图 11.16 中，单击其中的某一个月，将显示这一年中选中月份的日历，单击“ << ”或“ >> ”可以实现向前翻一个月或向后翻一个月的功能，同时文本框中的日期随之发生变化。

在本例中，其实难点并不多，只是需要实现的功能较多而已。同时还有一些常用的算法，比如如何判断某年是否是闰年，如何判断某一月是多少天等，这些都是较为典型的算法，希望读者朋友们能够熟练掌握。

样式单的使用在本例中体现得非常明显，几乎所有用到的元素都是使用预先定义的样式单来控制的，在本例中使用的是嵌入式样式单，在其中定义了表格的属性、表单元的属性、链接的属性和 daycalEnder、monthcalEnder 块的属性。

本例中使用了一个非常有用的技巧，就是根据需要隐藏或显示页面中的特定元素，例如，当需要显示某年某月的所有天数的日历 daycalEnder 时，就将 monthcalEnder 的 disply 属性设为“ none ”，而将 daycalEnder 的属性设为“ ”，这种方法的使用范围非常广泛，是实

现动态网页的一个重要手段。

显示或隐藏一个对象的方法如下：

显示一个对象：object.style.visibility = "visible"

隐藏一个对象：object.style.visibility = "hidden"

由于对象默认的 Visibility 属性值是 Visible，所以当需要显示该对象时也可以使用 object.style.visibility = "none"将原来的“ hidden ”值冲掉。

要实现本例中的所有功能，还必须具备一点，那就是要对 VBScript 中的内部函数，尤其是日期函数相当熟悉。本例中我们用到了最基本的 year 函数、month 函数、day 函数以及 weekday 函数，这些函数可以从日期中解析出更小的部分，这些函数都以一个日期表达式作为参数，并返回一个数字。

Year 函数得到日期的年的部分；month 函数得到日期的月的部分；day 函数得到日期的日的部分；而 weekday 函数得到返回指定的日期所在的星期数，默认情况下 weekday 假定一个星期的第一天是星期日。如果你想把星期一作为一周中的第一天，可以使用如下的语句：

```
Weekday(DateVar, vbMonday)
```

可以把任何一天作为一周的第一天，要指定一周的第一天是星期几，只要将 vbSunday、vbMonday、vbTuesday、vbWednesday、vbThursday、vbFriday、vbSaturday 代替 weekday() 的第二个参数即可。

11.8 石头、剪刀、布游戏

11.8.1 实例目标与技术要点

我们都玩过石头、剪刀和布的游戏，这个实例就是模仿这个游戏的，参与游戏的两人是计算机和计算机前的用户，用户选择武器的方式是通过屏幕来选择的，计算机选择武器的方式是每次都产生一个随机数，我们将这个随机数按照一定的规律转化为“ 石头 ”、“ 剪刀 ”和“ 布 ”中的一种，然后进行二者之间的判断。游戏的规则大家都知道：石头赢剪刀，剪刀赢布，布赢石头。

本实例中使用了 ASP 技术，但是并不难懂。总结起来，有下面几个要点需要读者掌握：

- Randomize 语句。
- Form 表单的使用与处理方法。
- ASP 的 Request 对象。
- ASP 的 Response 对象。

11.8.2 实例过程

要使该实例能够正常运行，首先必须为其建立一个虚拟目录。具体方法请参阅第 10 章“ 建立 ASP 的开发环境 ”部分。

下面我们开始为程序的运行作准备。为了使你的页面更加生动，请准备几张图片来表示结果。

第一步，为计算机产生一个随机的选择，我们将这个过程写在一个 Function 过程中。

```
Function computerChooses()
'定义变量
Dim randomNum
Dim choice
'初始化随机数生成器
Randomize
'产生一个 1 到 15 之间的整数
randomNum = int(rnd*15)+1
'根据得到的随机数为变量 Choice 赋值
If randomNum = 1 OR randomNum = 3 OR randomNum = 7 OR randomNum = 8
OR randomNum = 15 OR randomNum = 12 Then
choice = "R"
ElseIf randomNum = 2 OR randomNum = 6 OR randomNum = 11 OR
randomNum = 13 Then
 choice = "S"
Else
 choice = "P"
End If
'产生计算机选择的武器
computerChooses = choice
End Function
```

第二步，定义游戏的规则，也就是确定谁赢谁输，我们将这个规则定义在 determineWinner 过程中。

```
Sub determineWinner(playerChoice, computerChoice)
Const Rock = "R"
Const Scissor = "S"
Const Paper = "P"
'二者选择的武器相同时
If playerChoice = computerChoice Then
Response.Write "<p><CENTER>"
Response.Write "<IMG src=""images/tie.gIf""><BR>"
Response.Write "<font color=""red""><b>"
Response.Write "看来我们不分胜负哟! "
Response.Write "</b></font></CENTER>"
'如果这个条件成立，就此退出该过程，不再执行下面的语句。
exit Sub
End If
'当人的选择是石头时的各种情况
```

```
If playerChoice = Rock Then
   '计算机的选择是剪刀
If computerChoice = Scissor Then
    Response.Write "<P><CENTER>"
    Response.Write "<IMG src=""images/rock_beats_scissors.gIf"">"
    Response.Write "您<font color=""red""><b>赢</b></font>了<BR>"
    Response.Write "您的石头砸烂了计算机的剪刀!</CENTER>"
exit Sub
End If

'计算机的选择是布
If computerChoice = Paper Then
    Response.Write "<P><CENTER>"
    Response.Write "<IMG src=""images/paper_beats_rock.gIf"">"
    Response.Write "您<font color=""red""><b>输</b></font>了<BR>"
    Response.Write "计算机的布包住了您的石头!</CENTER>"
    exit Sub
End If

End If

'人的选择是剪刀的各种情况
If playerChoice = Scissor Then
   '计算机的选择是布
If computerChoice = Paper Then
    Response.Write "<P><CENTER>"
    Response.Write "<IMG src=""images/scissors_beats_paper.gIf"">"
    Response.Write "您<font color=""red""><b>赢</b></font>了<BR>"
    Response.Write "您的剪刀剪碎了计算机的布!</CENTER>"
    exit Sub
End If

'计算机选择的是石头
If computerChoice = Rock Then
    Response.Write "<P><CENTER>"
    Response.Write "<IMG src=""images/rock_beats_scissors.gIf"">"
    Response.Write "您<font color=""red""><b>输</b></font>了<BR>"
    Response.Write "计算机的石头砸烂了您的剪刀!</CENTER>"
    exit Sub
End If
End If

'人的选择是布的各种情况
```

```
If playerChoice = Paper Then
   计算机的选择是石头
If computerChoice = Rock Then
    Response.Write "<P><CENTER>"
    Response.Write "<IMG src=""images/paper_beats_rock.gIf"">"
    Response.Write "您<font color=""red""><b>赢</b></font>了<BR>"
    Response.Write "您的布包住了计算机的石头!</CENTER>"
exit Sub
End If

   计算机的选择是剪刀
   If computerChoice = Scissor Then
      Response.Write "<P><CENTER>"
    Response.Write "<IMG src=""IMAGES/scissors_beats_paper.gIf"">"
    Response.Wreite "您<font color=""red""><b>输</b></font>了<BR>"
    Response.Write "计算机的剪刀剪碎了您的布!</CENTER>"
      exit Sub
   End If
End If

End Sub
```

第三步，制作页面，整个程序其实只需要一个页面就可以了，这个页面包括一个 Form **表单，在表单中包含了** 3 **个单选按钮，**Name **值为“** playerSelect **”，表示计算机前的用户选择的武器，这** 3 **个单选按钮的取值分别为“** R **”、“** S **”和“** P **”，分别代表石头、剪刀和布。做好的页面如图** 11.17 **所示。**

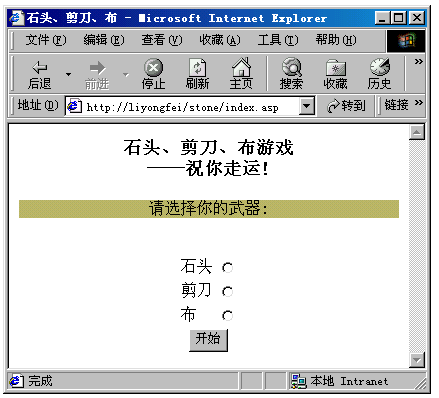

图 11.17　**石头、剪刀、布游戏的初始页面**

制作页面的源码如下：

```
<P align=center><FONT size=4><B> 石头、剪刀、布游戏<BR>
  ——祝你走运!</B></FONT></P>
```

```
<P align=center style="background-color: darkkhaki">请选择你的武器:</P>
<BR>
<CENTER>
<FORM action="index.asp " method="post">
<TABLE>
    <TR valign=top>
   <TD>石头</TD>
   <TD><INPUT type="radio" name="playerSelect" value="R"></TD>
    </TR>
    <TR valign=top>
   <TD>剪刀</TD>
   <TD><INPUT type="radio" name="playerSelect" value="S"></TD>
    </TR>
    <TR valign="top">
   <TD>布</TD>
   <TD><INPUT type="radio" name="playerSelect" value="P"></TD>
    </TR>
 </TABLE>
 <INPUT type="Submit" value="开始" >
</FORM>
</CENTER>
```

需要注意的是 Form **表单的属性，本例只设置了两个属性，**action **和** method**。**

action="index.asp "**表示该表单经过提交后将由页面** index.asp **来处理，实际上还是它本身。**

method="post"**表示数据传递的方法是** POST**，通过这种方式传递数据，则在服务器端，**FORM **集中包含了表单元素的值，用** Requet.Form **集合来读取数据。**

到现在为止，我们所需要的各部分都具备了，那如何把它们有机地联系起来使之正常的工作呢？无论如何，所要用到的过程都是要用到的，只要将它们加入到<%**和**%>**标记中就可以了。那如何确定在页面中是显示提示用户选择武器的页面呢，还是显示游戏的结果呢？那就得通过判断** Requst.Form("playerSelect")**中是否有值来确定了。当用户单击了“确定”按钮后，也就是提交了该表单，在服务器端就可以读到表单中的值了，如果** Requst.Form("playerSelect")**的值不为空，表示用户在请求结果，则显示游戏的结果，如果** Requst.Form("playerSelect")**的值为空，则请用户选择武器。另外，为了使用户能够重新开始游戏，还应该提供一个途径。我们将其写在过程** playAgin **中，并且在每一次显示结果之后调用它。**

```
Sub playAgain()
   Response.Write "<hr width=300 color=""darkkhaki"">"
Response.Write "<CENTER>是否想再来一次?<BR>"
   Response.Write "<A href=""index.asp"">是</A><BR>"
   Response.Write "<A href=""#"" onclick=self.close()>否</A><BR>"
```

```
   Response.Write "</CENTER>"
End Sub
```

该实例的这个程序结构如下：

```
<HTML>
<HEAD>
<TITLE>石头、剪刀、布</TITLE>
<META http-equiv="Content-Type" content="text/html; charset=gb2312">
<BODY>
<%

Function computerChooses()
      ……
End Function

Sub determineWinner(playerChoice, computerChoice)
    ……
End Sub

Sub playAgain()
    ……
End Sub
Dim player, computer
player = Request.Form("playerSelect")

If player <> "" Then
    player = Request.Form("playerSelect")
    computer = computerChooses
    determineWinner player, computer
    playAgain
Else
%>
<P align=center><FONT size=4><B> 石头、剪刀、布游戏<BR>
 ——祝你走运!</B></FONT></P>
<P align=center style="background-color: darkkhaki">请选择你的武器:</P>
<BR>
<CENTER>
<FORM action="index.asp " method="post">
<TABLE>
    <TR valign=top>
    <TD>石头</TD>
    <TD><INPUT type="radio" name="playerSelect" value="R"></TD>
    </TR>
```

```
    <TR valign=top>
    <TD>剪刀</TD>
    <TD><INPUT type="radio" name="playerSelect" value="S"></TD>
    </TR>
    <TR valign="top">
    <TD>布</TD>
    <TD><INPUT type="radio" name="playerSelect" value="P"></TD>
    </TR>
  </TABLE>
  <INPUT type="Submit" value="开始" >
</FORM>
</CENTER>
<%
End If
%>
</BODY>
</HTML>
```

11.8.3 实例效果与总结

由于计算机的选择是随机的，所以当用户选择一种武器的时候，它所得到的结果也是随机的，如图 11.18 **所示是当用户选择“剪刀”时的一种结果。**

图 11.18 **一次石头、剪刀和布游戏的结果**

本实例，重点应当掌握 Randomize **语句、**Form **表单的使用与处理方法、**ASP **的** Request **对象和** ASP **的** Response **对象的使用。**

1．Randomize 语句

Randomize **语句用来初始化随机数生成器。语法为：**

```
Randomize [number]
```

其中参数 number 可以是任何有效的数值表达式。

Randomize 使用 number 参数初始化 Rnd 函数的随机数生成器，赋给它新的种子值。如果省略 number，则使用系统计时器返回的值作为新的种子值。

如果不使用 Randomize，则第一次调用 Rnd 函数（无参数）时，它将使用相同的数字作为种子值，随后使用最后生成的数值作为种子值。

注意：要重复随机数的序列，请在使用数值参数调用 Randomize 之前，立即用负值参数调用 Rnd。使用相同的 number 值的 Randomize 不能重复先前的随机数序列。

下面例子说明如何使用 Randomize 语句：

```
Dim MyValue, Response
Randomize 初始化随机数生成器。
Do Until Response = vbNo
    MyValue = Int((6 * Rnd) + 1)          ’产生 1 到 6 之间的随机数。
    MsgBox MyValue
    Response = MsgBox ("是否再来一次? ", vbYesNo)
Loop
```

2．Form 表单的使用与处理

在开发动态网页的过程中，表单 Form 是一个非常重要的元素，我们经常通过它来实现与用户的交互。例如，Web 文档可以向用户提供表单，如申请表、通讯录、调查表、定购单、反馈等等。用户填写以后，提交给服务器上的程序进行处理。我们根据用户在表单中的输入来做出相应的不同反应。

表单是信息域的集合。这些信息域来自多种不同的形式，如文本框、文本区域、单选按钮、复选按钮、下拉框等。每一个表单都必须使用 FORM 元素（<FORM>和</FORM>标识表单的开始和结束）和组成每个控件的某些特别元素，如 INPUT、SELECT 和 TEXTAREA 元素等。

有关表单的属性（Action、Method）以及常用表单中的控件（如文本框、按钮等），我们在第 7 章中已经讲过，这里就不再多说了。我们主要学习如何在服务器端得到表单的值。

3．ASP 的 Request 对象

使用了表单的客户端 Web 界面将信息提交到服务器端，那么如何取得这些信息的值并对其进行处理呢？这就需要用到 ASP 的内置组件之一 Request 对象。

Request 对象是 ASP 中最有用的对象之一。它连接着 Web 客户机程序和 Web 服务器。通过这个连接，可以从以下几个集合中获取数据：

- HTTP QureyString。
- HTTP Request 体中的窗体数据。
- HTTP Cookies。
- 预定的 ServerVariables 集合。

Request 对象的调用方法如下：

```
Request[.Collection]("variable")
```

可以看出，在使用 Request 对象时，没有必要指定要引用的前面指出的 4 种集合中的哪一种，即 Collection 部分可以省略。当没有指出所要引用的集合时，Request 对象会自己按照前面列出的顺序从 4 种集合中一一搜索，直到找到第一个匹配的变量名。如果在所有的集合中都没有找到这个变量，Request 对象就会返回一个空值 NULL。

当然，一般来说，还是建议读者自己指定所要搜索的集合。因为这样可以优化运行速度，而且又可以减少错误的发生。尤其是两种不同的集合中都有同样的变量名称时，不指定集合，很有可能会得到错误的或者是不需要的变量值。

下面我们重点介绍一下 QueryString 集合和 Form 集合。

QueryString 集合

QueryString 数据集合取得客户端利用 URL 所传递的数据。说到 QueryString 集合，首先要了解 HTTP 传输的“GET”方法，“GET”方法是 HTTP 协议中的缺省请求方法，也是比较常用的一种方法，每个超链接使用的都是这种方法。通过对传送方法的判断，可以确定对一个.asp 文件的申请是通过链接还是通过窗口来进行的提交。当利用“GET”方法传输数据时，实际上就是把所要传输的数据作为查询字符串（QueryString）同 URL 一起送到服务器的。所谓查询字符串，就是在 URL 中，后面跟问号“?”的文本。

我们在网上浏览网页，经常会发现有时浏览器的地址栏中会出现类似这样的内容：

```
http://www.e-yao.net/info/news/class.asp?classid=85
```

其中，问号后面的字符串称为查询字符串。所谓 QueryString 集合，就是指它而言。因此，在 class.asp 中，使用 Request 对象<% Request.QueryString("classid") %>就会返回“85”这个字符串。

另外，QueryString 集合的所有成员都可以具有与之相关的多个值。例如下面的代码：

```
<FORM action="getdrink" method="get" name="frmdrink" >
请选择你喜欢的饮料：
<INPUT name=drink type=checkbox value=矿泉水>矿泉水
<INPUT name=drink type=checkbox value=可口可乐>可口可乐
<INPUT name=drink type=checkbox value=啤酒>啤酒
<INPUT name=drink type=checkbox value=鲜奶>鲜奶
<INPUT type="Submit" value="提交">
</FORM>
```

如果你选择了多种（比如说都选了）并且将表单提交给 getdrink.asp，则向服务器发送了下面的 HTTP GET 方法：

```
http://localhost/getdrink.asp?drink=矿泉水& drink=可口可乐&drink=啤酒&drink=鲜奶
```

如果想要输出其中的每一个值，可以使用 VBScript 中的 For Each item In 结构：

```
<%
For Each item In Request.QueryString(“drink”)
    Response.Write item
Next
%>
```

Form 集合

如果不使用“GET”方法时，可以采用“POST”方法。“POST”方法不是把信息作为字符串放在 URL 的末尾来传输的。因此，使用“POST”方法可以传送更多的信息，所以经常在提交表单时被采用。本例中的表单提交时，采用的就是“POST”方法。

当使用“POST”方法将表单元素的值提交给服务器时，可以作为 Form 集合的成员来检索。所以在本例中，采用 Request.Form("playerSelect")来取得用户选择的武器的值。

4．ASP 的 Response 对象

与 Request 获取客户端的 HTTP 信息相反，Request 对象是用于控制给用户发送信息，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 对象有许多方法，在本例中只用到了它的 Write 方法，除此外，我们还将介绍它的 Redirect 方法和 End 方法，这两个方法也都比较常用。

Write 方法

Write 方法是 Response 对象中最常用的方法之一。对于 ASP 开发来讲，最基本的过程就是向指向客户机的 HTML 输出流中写入数据。Response 对象的 Write 方法就是用来完成这一项功能的。Response.Write 的语法很简单，只需要一个引用的字符串或者一个能够返回字符串的函数。如：

```
Response.Write "<p>你好，北京！</p>"
```

我们知道，ASP 可以和 HTML 标记一起使用，当在大量的 ASP 代码中包含个别的 HTML 标记时，除了可以使用“<%”和“%>”将 ASP 程序分成若干段以外，也可以使用 ASP 所提供的 Response 对象的 Write 方法将其他字符直接传送至客户端。我们在本例中采用的就是这种方法。比如我们在定义游戏规则的时候，并没有使用 HTML 标记与<%和%>标记交叉的方法。而是直接将所有的需要输出的内容都由 Response.Write 来完成。拿一小段程序来说明，下面是当计算机的选择和人的选择相同的时候的执行结果。

```
<%
If playerChoice = computerChoice Then
   Response.Write "<P><CENTER>"
   Response.Write "<IMG src=""images/tie.gIf""><BR>"
   Response.Write "<FONT color=""red""><B>"
   Response.Write "看来我们不分胜负哟! "
   Response.Write "</B></FONT></CENTER>"
exit Sub
End If
%>
```

我们当然可以将上面的代码写为：

```
<%
If playerChoice = computerChoice Then
%>
<p><CENTER>
<IMG src=""images/tie.gIf""><BR>
<FONT color=""red""><B>看来我们不分胜负哟! </B></FONT>
</CENTER>
<%
End If
%>
```

有时为了程序的可读性，或者为了提高执行的性能，需要采取不同的输出方法。

当使用 VBScript 作为实现的脚本语言时，由于 VBScript 的静态字符常量的长度不能大于 1011 个字节，所以对输出长度超过 1022 个字节的静态字符串需要用一个变量来引用。

举个例子，在 VBScript 中，如果使用 Response 对象的 Write 方法直接输出包含 1050 个“ a ”的字符串是错误的。如下所示：

```
<%
Response.Write “aaaaaaaaa……aaaaaaaaa”
%>
```

要解决这个问题，需要将 1050 个“ a ”放在一个长度大于 1022 字节的字符串变量中，然后使用 Response.Write 输出这个变量。

```
<%
str1 = String(1050,”a”)
Response.Write str1
%>
```

另外，还要注意的是，在使用 Response 对象的 Write 方法时，输出的字符串中不允许含有“ %> ”字符。因为，服务器端的脚本引擎发现“ %> ”字符串后，会将其当作 ASP 脚本终止符而产生错误。当需要用 Write 方法输出‘ %> ’字符串时，可以使用转义字符串‘ %\> ”来代替或者将其加上双引号。

例如，若需要使用 Response 对象的 Write 方法输出字符串“ <HR width=80%> ”时，可以使用如下代码：

```
<%
Response.Write “<HR width=80%\>”
%>
```

或者

```
<%
```

```
Response.Write “<HR width=““80%””>”
%>
```

Redirect 方法

Redirect 方法使浏览器可以重新定向到另外一个 URL 上。但是要注意，在使用该方法之前，客户端不能有任何输出，也就是必须在网页的<HTML>标记之前调用。例如：

```
<HTML>
<HEAD>
<%
Dim a
a = 1
If a = 1 Then
    Response.Redirect "index.asp"
Else
    Response.Write "大家好！"
End If
%>
</HEAD>
<BODY>
</BODY>
</HTML>
```

运行该页面，将产生错误，如图 11.19 所示。

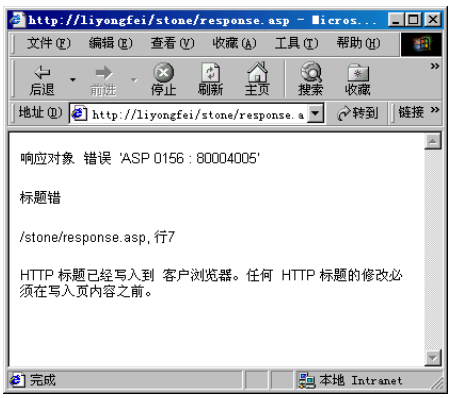

图 11.19　在<HTML>标记之后调用 Response.Recirect 方法

所以，必须将含有调用 Response.Recirect 方法的程序段放在文档的开头部分：

```
<%
Dim a
a = 1
If a = 1 Then
    Response.Redirect "index.asp"
```

```
Else
    Response.Write "大家好！"
End If
%>
<HTML>
<HEAD>
</HEAD>
<BODY>
</BODY>
</HTML>
```

这样，当你在浏览器的地址栏中敲入该.asp 文件的地址时，浏览器将会自动转到另外一个页面 index.asp 上。而不会输出“ 大家好 !”

End 方法

Response 对象的 End 方法用于告知 ASP 服务器停止处理 ASP 文件。

11.9 小 结

这是本书的最后一章，这一章中，我们通过 8 个实例，向大家展示了 VBScript 的强大功能。当然我们的目的并不是仅要求读者会制作这 8 个实例，重要的是通过实例和对实例的讲解，能够举一反三学会灵活使用 VBScript 的语句、过程和函数等。有关本章的技术要点在每一个例子的开头都有提示，并且在展示了例子的运行效果之后有比较详细的讲解。

相信读者在学习了这些例子之后，会对 VBScript 的编程有了更深的理解。如能将其应用到自己的网页设计中，那作者的目的就达到了。

练习题

1．在状态栏中显示特定文字的语法是什么？

2．设计一个程序，实现如下功能：

（1）改变网页上文字的内容。

（2）改变网页上文字的字体。

（3）改变网页上文字的大小。

（4）改变网页上文字的字体粗细。

提示：可以使用<DIV>标记，并使用样式单进行控制。

3．设计一个程序，动态显示当前时刻距离北京举办 2008 年奥运会的倒计时。

提示：使用日期函数和 setTimeout 函数。

4．在网页上做一个小精灵，使它一直停留在窗口的右下角。

5．结合上一章学习的有关 ASP 的知识，做一个同学通讯录程序。

附录 A　VBScript 的内置常量

1．颜色常量

此常量在 VBScript 中设置，在应用之前不必定义它们，可在代码中任意处应用它们以表明说明值。

表 A.1　颜色常量

常量	值	描述
vbBlack	&h00	黑色
vbRed	&hFF	红色
vbGreen	&hFF00	绿色
vbYellow	&hFFFF	黄色
vbBlue	&hFF0000	蓝色
vbMagenta	&hFF00FF	紫色
vbCyan	&hFFFF00	青色
vbWhite	&hFFFFFF	白色

2．比较常量

此常量在 VBScript 中设置，在应用之前不必定义它们，可在代码中任意处应用它们以表明说明值。

表 A.2　比较常量

常量	值	描述
vbBinaryCompare	0	执行二进制比较
vbTextCompare	1	执行文本比较

3．日期和时间常量

此常量在 VBScript 中设置，在应用之前不必定义它们，可在代码中任意处应用它们以表明说明值。

表 A.3　时期和时间常量

常量	值	描述
vbSunday	1	星期日
vbMonday	2	星期一
vbTuesday	3	星期二
vbWednesday	4	星期三

（续表）

常量	值	描述
vbThursday	5	星期四
vbFriday	6	星期五
vbSaturday	7	星期六
vbUseSystem	0	使用计算机区域设置中的日期格式
vbUseSystemDayOfWeek	0	由系统设置定义每周的第一天是星期几
vbFirstJan1	1	使用包含 1 月 1 日的星期（默认）
vbFirstFourDays	2	使用第一个至少包含新的年中四天的星期
vbFirstFullWeek	3	使用某年的第一个整周

4．日期格式常量

此常量在 VBScript 中设置，在应用之前不必定义它们，可在代码中任意处应用它们以表明说明值。

表 A.4 日期格式常量

常量	值	描述
vbGeneralDate	0	显示日期和/或时间。对于实数，显示日期和时间。如果没有小数部分，仅显示日期。如果没有整数部分，则仅显示时间。由系统设置决定日期和时间的显示
vbLongDate	1	按计算机中区域设置指定的长日期格式显示日期
vbShortDate	2	按计算机中区域设置指定的短日期格式显示日期
vbLongTime	3	按计算机中区域设置指定的长时间格式显示时间
vbShortTime	4	按计算机中区域设置指定的短时间格式显示时间

5．Locale ID (LCID)

表 A.5 区域设置 ID（LCID）表

区域设置描述	简写	十六进制值	十进制值	区域设置描述	简写	十六进制值	十进制值
南非荷兰语	af	0x0436	1078	印地语	hi	0x0439	1081
阿尔巴尼亚语	sq	0x041C	1052	匈牙利语	hu	0x040E	1038
阿拉伯语 - 阿拉伯	ar-ae	0x3801	14337	冰岛语	is	0x040F	1039
联合酋长国阿拉伯语 - 巴林	ar-bh	0x3C01	15361	印度尼西亚语	in	0x0421	1057
阿拉伯语 - 阿尔及利亚	ar-dz	0x1401	5121	意大利语 - 标准	it	0x0410	1040
阿拉伯语 - 埃及	ar-eg	0x0C01	3073	意大利语 - 瑞士	it-ch	0x0810	2064
阿拉伯语 - 伊拉克	ar-iq	0x0801	2049	日语	ja	0x0411	1041
阿拉伯语 - 约旦	ar-jo	0x2C01	11265	朝鲜语	ko	0x0412	1042

（续表）

区域设置描述	简写	十六进制值	十进制值	区域设置描述	简写	十六进制值	十进制值
阿拉伯语 - 科威特	ar-kw	0x3401	13313	拉脱维亚语	lv	0x0426	1062
阿拉伯语 - 黎巴嫩	ar-lb	0x3001	12289	立陶宛语	lt	0x0427	1063
阿拉伯语 - 利比亚	ar-ly	0x1001	4097	马其顿语	mk	0x042F	1071
阿拉伯语 - 摩洛哥	ar-ma	0x1801	6145	马来语 - 马来西亚	ms	0x043E	1086
阿拉伯语 - 阿曼	ar-om	0x2001	8193	马耳他语	mt	0x043A	1082
阿拉伯语 - 卡塔尔	ar-qa	0x4001	16385	挪威语 - 博克马尔	no	0x0414	1044
阿拉伯语 - 沙特阿拉伯	ar-sa	0x0401	1025	波兰语	pl	0x0415	1045
阿拉伯语 - 叙利亚	ar-sy	0x2801	10241	葡萄牙语 - 标准	pt	0x0816	2070
阿拉伯语 - 突尼斯	ar-tn	0x1C01	7169	葡萄牙语 - 巴西	pt-br	0x0416	1046
阿拉伯语 - 也门	ar-ye	0x2401	9217	拉托-罗马语	rm	0x0417	1047
巴斯克语	eu	0x042D	1069	罗马尼亚语	ro	0x0418	1048
白俄罗斯语	be	0x0423	1059	罗马尼亚语 - 摩尔多瓦	ro-mo	0x0818	2072
保加利亚语	bg	0x0402	1026	俄语	ru	0x0419	1049
加泰罗尼亚语	ca	0x0403	1027	俄语 - 摩尔多瓦	ru-mo	0x0819	2073
中文 - 中华人民共和国	zh-cn	0x0804	2052	塞尔维亚语 - 塞瑞利克	sr	0x0C1A	3098
中文 - 中华人民共和国香港特别行政区	zh-hk	0x0C04	3076	Setsuana	tn	0x0432	1074
中文 - 新加坡	zh-sg	0x1004	4100	斯洛文尼亚语	sl	0x0424	1060
中文 - 台湾地区	zh-tw	0x0404	1028	斯洛伐克语	sk	0x041B	1051
克罗地亚语	hr	0x041A	1050	索布语	sb	0x042E	1070
捷克语	cs	0x0405	1029	西班牙语 - 标准	es	0x040A	1034
丹麦语	da	0x0406	1030	西班牙语 - 阿根廷	es-ar	0x2C0A	11274
荷兰语	nl	0x0413	1043	西班牙语 - 玻利维亚	es-bo	0x400A	16394
荷兰语 - 比利时	nl-be	0x0813	2067	西班牙语 - 智利	es-cl	0x340A	13322
英语 - 澳大利亚	en-au	0x0C09	3081	西班牙语 - 哥伦比亚	es-co	0x240A	9226
英语 - 伯利兹	en-bz	0x2809	10249	西班牙语 - 哥斯达黎加	es-cr	0x140A	5130
英语 - 加拿大	en-ca	0x1009	4105	西班牙语 - 多米尼加共和国	es-do	0x1C0A	7178
英语 - 爱尔兰	en-ie	0x1809	6153	西班牙语 - 厄瓜多尔	es-ec	0x300A	12298
英语 - 牙买加	en-jm	0x2009	8201	西班牙语 - 危地马拉	es-gt	0x100A	4106

（续表）

区域设置描述	简写	十六进制值	十进制值	区域设置描述	简写	十六进制值	十进制值
英语 - 新西兰	en-nz	0x1409	5129	西班牙语 - 洪都拉斯	es-hn	0x480A	18442
英语 - 南非	en-za	0x1C09	7177	西班牙语 - 墨西哥	es-mx	0x080A	2058
英语 - 特立尼达岛	en-tt	0x2C09	11273	西班牙语 - 尼加拉瓜	es-ni	0x4C0A	19466
英语 - 英国	en-gb	0x0809	2057	西班牙语 - 巴拿马	es-pa	0x180A	6154
英语 - 美国	en-us	0x0409	1033	西班牙语 - 秘鲁	es-pe	0x280A	10250
爱沙尼亚语	et	0x0425	1061	西班牙语 - 波多黎各	es-pr	0x500A	20490
波斯语	fa	0x0429	1065	西班牙语 - 巴拉圭	es-py	0x3C0A	15370
芬兰语	fi	0x040B	1035	西班牙语 - 萨尔瓦多	es-sv	0x440A	17418
法罗语	fo	0x0438	1080	西班牙语 - 乌拉圭	es-uy	0x380A	14346
法语 - 标准	fr	0x040C	1036	西班牙语 - 委内瑞拉	es-ve	0x200A	8202
法语 - 比利时	fr-be	0x080C	2060	苏图语	sx	0x0430	1072
法语 - 加拿大	fr-ca	0x0C0C	3084	瑞典语	sv	0x041D	1053
法语 - 卢森堡	fr-lu	0x140C	5132	瑞典语 - 芬兰	sv-fi	0x081D	2077
法语 - 瑞士	fr-ch	0x100C	4108	泰语	th	0x041E	1054
盖尔语 - 苏格兰	gd	0x043C	1084	土耳其语	tr	0x041F	1055
德语 - 标准	de	0x0407	1031	汤加语	ts	0x0431	1073
德语 - 奥地利	de-at	0x0C07	3079	乌克兰语	uk	0x0422	1058
德语 - 列支敦士登	de-li	0x1407	5127	乌尔都语 - 巴基斯坦	ur	0x0420	1056
德语 - 卢森堡	de-lu	0x1007	4103	越南语	vi	0x042A	1066
德语 - 瑞士	de-ch	0x0807	2055	科萨语	xh	0x0434	1076
希腊语	el	0x0408	1032	意第绪语	ji	0x043D	1085
Hebrew	he	0x040D	1037	祖鲁语	zu	0x0435	1077

6．杂项常量

因为此常量在 VBScript 中被建立，你在使用之前不必定义它们。你可在代码中任意处使用它来表示说明值。

表 A.6 杂项常数

常量	值	描述
vbObjectError	-2147221504	**自定义错误号应大于该值，例如，**Err.Raise Number = vbObjectError + 1000

7．MsgBox 常量

表 A.7 MsgBox 常量

常量	值	描述
vbOKOnly	0	**只显示确定按钮**

（续表）

常量	值	描述
vbOKCancel	1	显示确定和取消按钮
vbAbortRetryIgnore	2	显示终止、重试和忽略按钮
vbYesNoCancel	3	显示是、否和取消按钮
vbYesNo	4	显示是和否按钮
vbRetryCancel	5	显示重试和取消按钮
vbCritical	16	显示临界消息图标
vbQuestion	32	显示警告询问图标
vbExclamation	48	显示警告消息图标
vbInformation	64	显示提示消息图标
vbDefaultButton1	0	第 1 个按钮是默认按钮
vbDefaultButton2	256	第 2 个按钮是默认按钮
vbDefaultButton3	512	第 3 个按钮是默认按钮
vbDefaultButton4	768	第 4 个按钮是默认按钮
vbApplicationModal	0	应用程序模式。用户必须响应消息框，才能继续在当前应用程序中工作
vbSystemModal	4096	系统模式。在 Win16 系统中，所有应用程序都将中止直到用户响应消息框。在 Win32 系统中，此常量提供一个应用程序模式信息框并总是保留在你可能正在运行的所有其他程序的顶部

8．三态常量

这些常量在 VBScript 中被建立，因此你在使用之前不必定义它们。你可在代码中任意处使用它们表示说明值。

表 A.8　三态常量

常量	值	描述
vbUseDefault	-2	使用来自计算机最初设置中的默认值。
vbTrue	-1	True
vbFalse	0	False

9．字符串常量

这些常量在 VBScript 中被建立，因此你在使用之前不必定义它们。你可在代码中任意处使用它们表示说明值。

表 A.9　字符串常量

常量	值	描述
vbCr	Chr(13)	回车符

（续表）

常量	值	描述
vbCrLf	Chr(13) & Chr(10)	回车符与换行符
vbFormFeed	Chr(12)	换页符；在 Microsoft Windows 中不适用
vbLf	Chr(10)	换行符
vbNewLine	Chr(13) & Chr(10) 或 Chr(10)	平台指定的新行字符；适用于任何平台
vbNullChar	Chr(0)	值为 0 的字符
vbNullString	值为 0 的字符串	与零长度字符串("")不同；用于调用外部过程
vbTab	Chr(9)	水平附签
vbVerticalTab	Chr(11)	垂直附签；在 Microsoft Windows 中不适用

10．VarType 常量

这些常量在 VBScript 中被建立，因此您在使用之前不必定义它们。 您可在代码中任意处使用它们表示说明值。

表 A.10 VarType 常量

常量	值	描述
vbEmpty	0	未初始化（默认）
vbNull	1	不包含任何有效数据
vbInteger	2	整型子类型
vbLong	3	长整型子类型
vbSingle	4	单精度子类型
vbDouble	5	双精度子类型
vbCurrency	6	货币子类型
vbDate	7	日期子类型
vbString	8	字符串子类型
vbObject	9	对象
vbError	10	错误子类型
vbBoolean	11	Boolean 子类型
vbVariant	12	Variant （仅用于变量数组）
vbDataObject	13	数据访问对象
vbDecimal	14	十进制子类型
vbByte	17	字节子类型
vbArray	8192	数组

附录 B　VBScript 运行时错误

如果 VBScript 脚本执行系统无法实施的操作，则会产生 VBScript 运行时错误。只有在运行脚本、为变量表达式赋值或分配内存时，才会产生 VBScript 运行时错误。

表 B.1　VBScript 运行时错误编号

错误编号	描述
429	ActiveX 部件无法创建对象
507	发生异常
449	参数不可选
17	无法执行请求的操作
430	类不支持自动化
506	类未被定义
11	被零除
48	加载 DLL 错误
5020	在正则表达式中需要 ')'
5019	在正则表达式中需要 ']'
432	在自动化操作中未找到文件名或类名
92	For 循环未初始化
5008	非法赋值
51	内部错误
505	无效的或不合格的引用
481	无效图片
5	无效过程调用或参数
5021	字符集越界
94	非法使用 Null
448	未找到命名参数
447	对象不支持当前的区域设置
445	对象不支持此操作
438	对象不支持该属性或方法
451	对象不是一个集合
504	对象不能安全创建
503	对象不能安全初始化
502	脚本对象不安全

（续表）

错误编号	描述
424	需要对象
91	未设置对象变量
7	内存不足
28	堆栈溢出
14	字符串空间溢出
6	溢出
35	未定义 Sub 或 Function
9	下标越界
5017	正则表达式中的语法错误
462	远程服务器不存在或不能访问
10	该数组为定长的或临时被锁定
13	类型不匹配
5018	错误的数量词
500	变量未定义
458	变量使用了一个 VBScript 中不支持的自动化类型
450	错误的参数个数或无效的参数属性值

附录 C　VBScript 语法错误

如果 VBScript **语句结构违反了一个或多个** VBScript **脚本语言语法规则，就会产生** VBScript **语法错误。**VBScript **语法错误通常在执行程序前，编译程序时产生。**

表 C.1　VBScript **语法错误编号**

错误编号	**描述**
1052	**在类中不能有多个缺省的属性/方法**
1044	**调用** Sub **时不能使用圆括号**
1053	**类初始化或终止不能带参数**
1058	**只能在** Property Get **中指定** Default
1057	**说明** Default **必须同时说明** Public
1005	**需要** '('
1006	**需要** ')'
1011	**需要** '='
1021	**需要** Case
1047	**需要** Class
1025	**需要语句的结束**
1014	**需要** End
1023	**需要表达式**
1015	**需要** Function
1010	**需要标识符**
1012	**需要** If
1046	**需要** In
1026	**需要整数常数**
1049	**在属性声明中需要** Let , Set **或** Get
1045	**需要文字常数**
1019	**需要** Loop
1020	**需要** Next
1050	**需要** Property
1022	**需要** Select
1024	**需要语句**
1016	**需要** Sub
1017	**需要** Then

（续表）

错误编号	描述
1013	需要 To
1018	需要 Wend
1027	需要 While 或 Until
1028	需要 While、Until，或语句未结束
1029	需要 With
1030	标识符太长
1014	无效字符
1039	无效 exit 语句
1040	无效 for 循环控制变量
1013	无效数字
1037	无效使用关键字 Me
1038	loop 没有 do
1048	必须在一个类的内部定义
1042	必须为行的第一个语句
1041	名称重定义
1051	参数数目必须与属性说明一致
1001	内存不足
1054	Property Let 或 Set 至少应该有一个参数
1002	语法错误
1055	不需要的 Next
1015	未终止字符串常数