

第4章 服务器进程和ASP Server对象

前面的章节已经研究了 ASP的一些内置对象。重点介绍的对象有 Request、Response、Session和Application对象。ASP中另一个比较主要的对象为 Server对象。本章重点介绍这个对象的背景知识和使用方法。

Server对象在服务器端脚本中通过实例和使用其他外部对象和组件，提供了一种扩展 ASP页的功能的方法。事实上，许多人认为这个对象是 ASP之所以能够流行的主要因素。引入 Server对象的意义很大，这意味着 ASP不必提供人们所需要的所有属性。它可调用其他应用程序和组件完成指定工作。

这也恰好符合了构建由独立的对象组成的应用程序的总体目标，而不是构建通常见到的那种耗尽硬盘空间的“可做每件事情”的巨型程序。不仅仅是在计算机的世界中，整个世界正在朝着组件和“即插即用”概念方面发展。如今，当汽车或电视机发生故障时，技师或工程师很可能会拔出有问题的部件并且插上一个新的部件，因此，汽车维修人员或电视机维修人员的工作也变成了面向对象的。

然而，IIS同样支持使用外部对象和与服务器环境进行交互作用的许多传统方法。这是一个特定的例外，这些方法并不是真正的 ASP组成部分，但通常的确非常有用，并且通过 Server对象的一些新特性已经与 ASP很好地进行了集成。本章将简要地回顾这些传统方法，然后详细地介绍ASP Server对象。

本章的主要内容为：

- 传统的服务器端包含(SSI)指令的背景知识和使用方法。
- Server对象所要完成的任务，以及与 SSI的比较。
- 如何使用Server对象实例、外部组件和应用程序。
- 如何使用Server对象执行封装的脚本或其他 ASP页面。
- 如何使用Server对象管理在脚本中出现的错误。
- 如何使用Server对象完成与 HTML或HTTP兼容的格式转换。

Server对象是ASP页中的错误处理过程的一部分，这在 IIS 5.0和ASP 3.0中是新的内容。本章介绍该对象是如何工作的。因为有单独的一章(第7章)专门讨论有关调试和错误处理方面的所有问题，所以本章只简要地讨论错误处理方法，并且仅限于 Server对象直接涉及的过程。

4.1 动态页中服务器端的处理

就服务器端处理而言，ASP是产生动态Web网页的一种相对较新的技术。动态页意味着什么呢？先暂时不考虑客户端相关功能上的进展，也不讨论客户端脚本、Java Applet、动态HTML或ActiveX控件等内容。这里的动态页是专指服务器响应客户端请求而产生的页面，并且根据情况每次产生的页面可能是不同的。

举个简单的例子，创建一个只包含当前日期和时间的页面。每次请求该页面时将显示一个不同的值，因为日期和时间取决于服务器的时钟，或取决于提供日期和时间的一个资源（例

如一个独立的服务器或来自于互联网上一个标准时钟)。当然,实际上动态页要比这复杂得多,也许显示数据库记录的当前值或者邮件服务器上等待着的邮件消息的摘要。重要的是服务器不仅阅读一个无格式的HTML页面、或磁盘上的文本文件以及把它们发送给客户,而且,必须完成一些工作来创建该页面。

Internet服务器应用编程接口

第1章介绍了创建动态页的一些方法。传统的技术是使用与Web服务器的一个接口,它被称为Internet服务器应用编程接口(Internet Server Application Programming Interface, ISAPI)。

ISAPI可用于执行其他的应用程序,这些应用程序通过C语言风格的stdin和stdout数据流函数来读取客户端请求的值并创建Web服务器的响应。ISAPI应用程序所必须做的全部事情就是编写相应结果页面的文本和HTML,并通过stdout函数输出到Web服务器。事实上ASP DLL内部真正做的事情是更面向对象的。

IIS自开始就支持ISAPI的应用程序和脚本解释器。它提供一个特殊的解释器动态链接库,给出访问服务器的请求和响应的另一种方法,尽管受到一定的限制。它通过服务器端包含指令实现,之所以这样说,是因为它们是在服务器上执行的,并且结果包含在传送给客户端的响应中。这个特性在IIS中是通过一个名为ssinc.dll的动态链接库实现的。缺省情况下,IIS把文件扩展名为.shtml、.shtm或.stm的任意页面都映射到这个动态链接库。打开默认Web站点的Properties对话框,在Application Setting中单击Configuration按钮,可以看到这种映射,如图4-1所示。

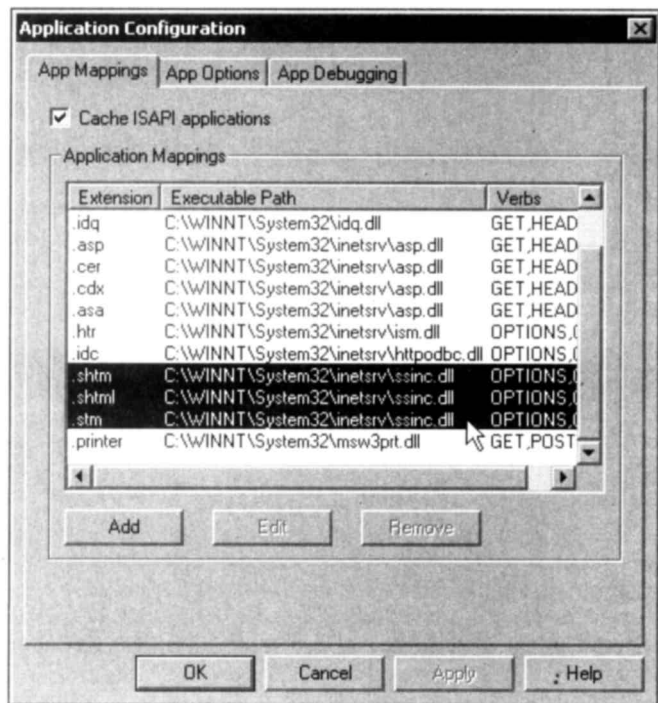


图4-1 Application Configuration对话框

这样,带有这些被映射的文件扩展名的页面将被传送给ssinc.dll进行处理。因此,执行页

面中所有的服务器端包含语句, 结果 (如有的话) 插入到服务器的响应中, 即插入到客户端接收到的页面中。

因为这些文件是映射到 ssinc.dll 文件而不是映射到 ASP 动态链接库 (asp.dll) 的, 所以在这些页面中的所有 ASP 代码将被忽略并且按照原有状态传送给客户端, 客户端将能够看到这些脚本。然而, 在 ASP 3.0 版本中有一个避免这种情况的方法, 稍后在讨论 Server 对象的 Execute 和 Transfer 方法时, 研究这个方法。

4.2 服务器端的包含指令

利用服务器端的包含 (SSI) 语句 (或者指令) 能够做些什么呢? 实际上不多, 除非打算创建在 Web 服务器上运行的可执行文件, 并通过 stdin 和 stdout 函数访问 ISAPI。这就意味能够用 C、C++ 或其他语言 (如 Delphi) 等编写它们, 但 VB 并不适合。此外, 使用 SSI 指令能够做的事情可达到与在 ASP 中实现同样好的效果。许多方法中, IIS 的 SSI 特性对使用这些特性的原有的 Web 网站和 Web 页面具有向下兼容性。

然而, 可能有时会希望在站点上使用 SSI 而不是 ASP。在 IIS 5.0 中, 服务器端的包含指令能够比以前更加容易地集成到一个远程站点上的 ASP 页, 它们是有用的, 特别是作为执行操作系统命令或原有的 CGI 应用程序的一种方式。后面将会非常详细地介绍可用的指令。

#include 指令是这些指令之一, 它已经与 ASP 一起使用了一段时间了, 同样也在 SSI 页中使用。事实上, 这已经对那些不具备传统的 Web 开发背景的 ASP 开发人员带来了许多混乱。

4.2.1 不可思议的 ASP #include 指令

在一个 ASP 页中, 可以使用 #include 指令把另一个文件的内容插入到当前的页面中:

```
<!-- #include file="/scripts/usefulbits.inc" -->
```

这条指令读取该文件的全部内容并插入到该页中, 替代 <!--# include...--> 行。这是一种非常有用的插入 HTML 段落的技术, 可反复使用。也常用该指令来插入代码段。例如, 如果有一个包含几个脚本函数 (或者只是单行脚本代码) 的文件同时在几个页面中使用, 则可以使用 #include 指令将其插入到需要它的每个页中。

通过把脚本和内容分开的方法, 给页面提供了一个组成层次。这意味着如果对脚本进行了修改, 在客户端再次打开该页面时, 脚本的修改情况自动地反映到使用包含文件的每个页面中。包含文件也是一种插入服务器特定的信息的简单方法, 所以把站点转移到另一个服务器不意味着必须编辑涉及原来服务器的所有页面 (明显的例子是数据库连接字符串或指定一个完整的 URL 或服务器名字的链接)。这可以极大地减少维护费用。

例如, 可以把下面的内容作为一个包含文件, 命名为 connect.inc:

```
<%  
strConnect = "SERVER=myserver;DATABASE=mydb;DRIVER={SQL Server};" _  
    & "UID=username;PWD=secretpassword"  
%>
```

然后可以在任何页中使用这个文件:

```
<!-- #include file="path_to_file\connect.inc" -->  
<%  
...  
strTheConnectionString = strConnect 'From include file
```

```
...  
%>
```

使用包含文件的另一种情况是有些内容需要按指定的时间间隔进行修改。例如，在 Wrox Web Developer站点上显示书目列表的网页，它包含了一个表，其中提供了所有的封面、书名和一些按钮，如图4-2所示。这个表的HTML和文本保留在一个单独的文件中，该文件通过一条单独的#include语句包含在主页中。每次一本新书加入到该网页所基于的数据库中时，那个包含文本文件根据该数据库的情况重新创建，并作为一个文本文件写到磁盘上。



图4-2 显示书目列表的网页

这个技术大大地减少了在 Web服务器和数据库服务器上的工作量，对该站点的访问者实现较快地响应。

1. 包含文件和ASP

在ASP网页(即带有.asp文件扩展名的网页)中使用的#include指令不能像一条真正的SSI指令那样进行处理，它仅是一条ASP能够识别并进行语法分析的特别指令。ssinc.dll直接用于执行SSI#include指令。然而这个由相应文件的内容替代#include指令的页面由ASP解释。

这意味着ASP对#include指令所进行的操作不实施控制。例如，可以试验以下代码：

```
<%  
'This will *not* work  
strIncludeURL = Request.Form("FileName")  
%>  
...  
<!-- #include file="<% = strIncludeURL %>" -->
```


ssinc.dll将查找名为<% =strIncludeURL %>的文件，并且不可能找到，因此这段代码不会工作。

2. 包含文件的安全性

如果没有包含可执行脚本，在 Web服务器上的 ASP网页不能通过 IIS 的 Web 服务程序下载到一个客户端。但是，有人已经发现了偶然的安全性漏洞，比如著名的 \$DATA 问题，所有在 NTFS 格式化的磁盘上保留 Web 内容的 Web 服务器都存在相应的问题。在 IIS 5.0 中这个问题已经得到解决。

\$DATA 问题的出现是因为在 Windows NTFS 驱动器上的所有文件都有一个缺省的“值”，即是该文件的内容，并且通过文件名加后缀“::\$DATA”来指示。将其增加到一个 ASP 页的 URL 的末尾将打乱 IIS 中的脚本映射关系，且允许服务器不对其中包含的脚本进行处理而下载该页面。对 ISS 4.0 和早期版本，有一个方法可以解决这个问题，或者可以只是增加几个映射来强制 IIS 正常地执行该网页：即增加对“.asp::\$DATA”和“.asa::\$DATA”的映射，两者都指向 asp.dll 文件。

包含文件的扩展名一般是 .inc 或 .txt。如果在站点上发现一个包含文件的路径和文件名，可通过把包含文件的 URL 键入到浏览器的地址栏中，下载该包含文件，而不会把其作为 ASP 网页的一部分来执行。为防止出现这样情况，特别是在文件包含有敏感信息（诸如一个数据库链接字符串）的情况下，可能希望包含文件的扩展名为 .asp。在这种情况下，如果试图下载一个包含文件，它将首先被传送到 ASP，ASP 将执行该文件中的所有脚本代码，并只发送出结果。如在包含文件中定义的一个链接字符串如下：

```
<%  
strConnect = "SERVER=myserver;DATABASE=mydb;DRIVER={SQL Server};" _  
    & "UID=username;PWD=secretpassword"  
Response.Write vbCrLf "Output a carriage return character"  
%>
```

客户端只能接受到单个回车符而不是脚本代码，因为该文件已经被 ASP 在服务器上执行了。如果不包含回车换行符，浏览器将挂起并等待一个响应（这并不是我们的问题，因为我们确实不打算允许用户直接访问这个文件）。

IIS 5.0 和 Windows 的访问控制列表

在 IIS 5.0 中，Microsoft 已经改变了 Web 服务器和操作系统访问服务器端包含文件的方法。

在 IIS 早期的版本中，当 ssinc.dll 载入一个虚拟 URL（即使用 VIRTUAL = "filename" 而不是 FILE = "filename"）定位的一个包含文件时，将绕过 Windows 本身的安全性检查并忽略该文件及所存储的目录上的任何安全性设置。现在，在 IIS 5.0 中，运行当前 ASP 或 SSI 页面的帐号必须与对该文件和目录在 Windows 访问控制列表 (ACL) 中设置的权限相一致。如果不一致，该 SSI 指令运行将失败。

4.2.2 服务器端包含指令概要

除了已经讨论过的 #include 语句以外，还有 IIS 支持的五条服务器端包含指令（记住，除 #include 以外，这些语句不能在 ASP 网页中执行）。这些服务器端包含指令及说明如表 4-1 所示。

表4-1 服务器端包含指令及说明

指 令	说 明
#include	<p>把一个指定文件的内容插入到将被发送给客户端的响应流中并代替该指令。例如：</p> <pre>< ! -- # include FILE = "usefulbits.inc" -- ></pre> <p>这条指令把名为usefulbits.inc文件的内容插入到响应中。这个文件可以由一个相对或全路径与文件名的组合描述，如FILE="..\scripts\myscr.inc"。通过使用VIRTUAL属性，可使用一个虚拟的相对或绝对路径来描述它，例如：</p> <pre>< ! -- #include VIRTUAL="/mysite/usefulbits.inc" -- > < ! -- #include VIRTUAL=" ../.. /thisbit/usefulbits.inc" -- ></pre>
#config	<p>说明在其后的指令中将用于数据、时间和文件大小以及返回给客户端的一般性的 SSI 错误信息的文本的格式。例如：</p> <pre>< ! -- #config ERRMSG="SSI Processing Error" -- ></pre> <p>设置SSI错误信息内容为'SSI Processing Error'。</p> <pre>< ! -- #config TIMEFMT = "%A, %B %d %M:%S" -- ></pre> <p>设置由其后的SSI指令返回的日期和时间的格式。这个例子设置了一个格式风格：Saturday, August 14 1999 10:34:50。可以用于格式字符串的标志的列表在附录 C 中给出。</p> <pre>< ! -- #config SIZEFMT = "BYTES" -- ></pre> <p>设置由其后的IIS指令返回的文件大小的单位。这个例子设置单位为字节。对 SIZEFMT可供选择的值是"ABBREV"，指明计算值将以千字节(KB)返回文件的大小</p>
#echo	<p>把一个HTTP环境变量的值插入到发送给客户端的响应流中并替换该指令。例如：</p> <pre>< ! -- #echo VAR="SERVER_NAME" -- ></pre>
#exec	<p>写出正在执行指令到该网页的服务器的名字</p> <p>执行一个程序或一个服务器外壳命令，例如</p> <pre><!--#exec CGI="/scripts/myapp.exe? value1= this & value2= that--</pre> <p>执行名为myapp.exe的CGI程序，允许传递查询字符串，程序在单独内存中执行。</p> <pre><!--#execCMD="cmd.exe/c iisreset/stop" - -></pre> <p>启动特定操作系统命令解释器(cmo.exe)并执行命令msreset/stop。/c表示当命令结束时，命令解释器也结束。使用CMO要添加下列注册表项：</p> <pre>HKEY_LOCAL_MACHINE / SYSTEM / CurrentControlSet / Services / w3SVC / Parameters / SSISetCmdDirective</pre> <p>设置值为1,并重新启动WWW服务，就允许CMD标志用于#exec指令中。值为0,则禁止使用，并防止未验证的使用</p>
#flastmod	<p>把一个指定的文件上一次修改的日期和时间插入到发送给客户端的响应流中并代替该指令。例如：</p> <pre>< ! -- #flastmod FILE="Default.asp" -- ></pre> <p>像#include指令一样，也可以使用虚拟路径对该文件进行定义，如：</p> <pre>VIRTUAL="/mysite/usefulbits.inc"</pre> <p>或</p> <pre>VIRTUAL=" ../.. /thisbit/usefulbits.inc"</pre>
#fsize	<p>把一个指定的文件的大小插入到发送给客户端的响应流中并代替该指令。例如：</p> <pre>< ! -- #fsize FILE="Default.asp" -- ></pre> <p>象#include指令一样，也可以使用虚拟路径对该文件进行定义，如：</p> <pre>VIRTUAL="/mysite/usefulbits.inc"</pre> <p>或</p> <pre>VIRTUAL=" ../.. /thisbit/usefulbits.inc"</pre>

1. IISRESET实用程序

iisreset.exe是由IIS 5.0提供的一个新的实用程序。作为一个命令行的实用程序，如果用于执行该实用程序的帐号具有管理员权限，它对于控制运行在本地或一个网络计算机上的

Internet连网服务是非常有用的。它可用于以正确的顺序停止或启动所有的服务、显示服务的状态、重新引导服务器以及允许或禁止服务的管理。例如：

```
iisreset /RESTART /TIMEOUT:30 /REBOOTONERROR
```

这将以正确的顺序停止和重新启动所有 Internet服务。如果一种服务在指定的超时周期 (30秒)内未能停止或重新启动, 服务器将重新引导。可以用在 CMD类型的#echo SSI指令中的一些开关, 使该页面不能进行匿名访问并且要求用户提供在目标服务器上具有管理员权限的有效帐号的详细情况。这个实用程序的完整描述和可用的命令开关在附录 C中。

2. NET STOP和NET START命令

如果用来执行实用程序 net.exe的帐号具有管理员权限, 它可以用来管理服务器上运行的任何服务 (既可以是本地的也可以是来自其他的一个计算机)。虽然不提倡把该程序用于 Internet服务 (如WWW或FTP服务), 但其停止和启动其他服务的功能是非常有用的。事实上, net命令同样可以用于一系列的其他网络相关命令。

语法是：

```
net [ start | stop ] service_name
```

例如, 可以用命令 net stop cisvc和 net start cisvc来停止和启动 Microsoft Indexing Service。可以用CMD类型的#echo SSI指令使该页面不能进行匿名访问并要求用户提供在目标服务器上具有管理员权限的有效帐号的详细情况。稍后将看到一个这样的例子。

在 Windows 2000帮助文件中可以找到 net命令的所有选项和开关的一个完整列表。

从Start菜单中选择 Help项, 在 Help窗口的 Index页查找 “ net commands ”。

4.2.3 服务器端包含指令的例子

本节提供了一些示例页面, 可以用来对各种服务器端包含语句进行实验。打开示例网页的子目录 Chapter04, 显示 “ SSI Directives and the ASP Server Object ” 主页 (即子目录 Chapter04中的 Default.asp), 如图4-3所示。

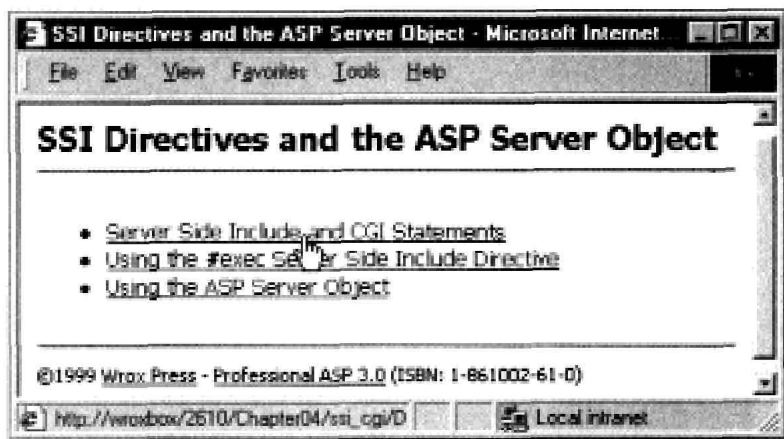


图4-3 示例网页

本书的所有示例都可以从我们的 Web网站下载。读者将在示例的子目录 Chapter04中发现本章其余部分的所有示例页面。

1. 使用SSI/CGI处理指令

单击链接进入“Server-Side Include and CGI Statements”页面，这将打开ssi_cgi.stm网页。需要注意的是该页面的文件扩展名为.stm，表明这不是一个ASP网页。该页面使用了前面已经讨论过的除#exec指令(稍后将看到)以外的所有SSI指令，且显示指令的使用方法和结果，如图4-4所示。

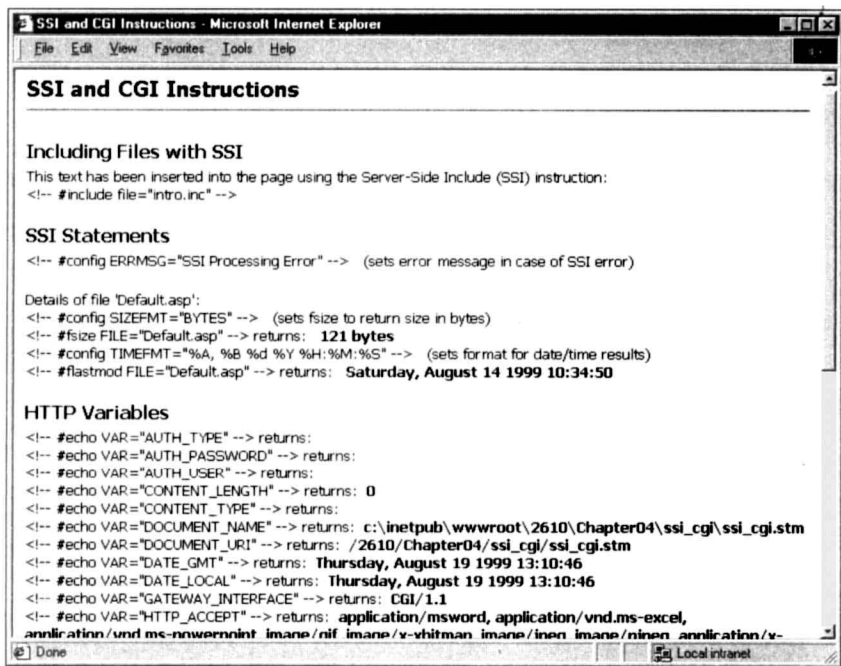


图4-4 示例网页显示的SSI/CGI处理指令情况

(1) #include指令

该页的开始部分“Include Files with SSI”，显示名为intro.inc的另一个单独文件的内容。下面是该文件的全部内容：

注意我们必须使用HTML条目“<”和“>”来显示网页中的尖括号。如果不这样做，它们就不能被当作注解元素部分看待，并引起其中的指令被执行。

在主ssi_cgi.stm页面中，把这个文件插入到该页中的代码是很简单的：

```
<!-- #include file="intro.inc" -->
```

(2) #config、#fsize和#flastmod指令

网页下一部分显示了与该页面在相同的目录中的文件 Default.asp的大小和最后被修改的时间。这里三次使用了#config指令：

- 一次是设置SSI错误信息。
- 一次是设置日期和时间的格式。
- 一次是设置文件大小计算的格式。

使用#fsize和#flastmod指令把值插入到该网页中：

```
<P><DIV CLASS="subhead">SSI Statements</DIV>
&lt;!-- #config ERRMSG="SSI Processing Error" --&gt; &nbsp;
```



```
(sets error message in case of SSI error)<BR>
<!-- #config ERRMSG="SSI Processing Error" --><P>

Details of file 'Default.asp':<BR>
<!-- #config SIZEFMT="BYTES" --> &nbsp;
(sets fsize to return size in bytes)<BR>
<!-- #config SIZEFMT="BYTES" -->

&lt;!-- #fsize FILE="Default.asp" -->
returns: &nbsp; <B><!-- #fsize FILE="Default.asp" --> bytes</B><BR>

&lt;!-- #config TIMEFMT="%A, %B %d %Y %H:%M:%S" --> &nbsp;
(sets format for date/time results)<BR>
<!-- #config TIMEFMT="%A, %B %d %Y %H:%M:%S" -->

&lt;!-- #flastmod FILE="Default.asp" -->
returns: &nbsp; <B><!-- #flastmod file="Default.asp" --></B><P>
```

(3) #echo指令

该页的最后部分(在屏幕上只能看到一部分)显示可以使用#echo指令访问的所有HTTP报头的内容。每一行的代码都是相同的,仅仅是VAR属性值有变化。附录G中给出了VAR属性的所有容许值的一个完整列表。

```
<DIV CLASS="subhead">HTTP Variables</DIV>
&lt;!-- #echo VAR="AUTH_TYPE" -->
returns: &nbsp;<B><!-- #echo var="AUTH_TYPE" --></B><BR>
&lt;!-- #echo VAR="AUTH_PASSWORD" -->
returns: &nbsp;<B><!-- #echo var="AUTH_PASSWORD" --></B><BR>
... etc ...
```

2. 使用#exec指令

#exec指令与其他的SSI指令相比使用起来困难一些,正因为如此,将其独立地放到了另一个页面上。可以从“ASP Sever Object and SSI Directives”主菜单上访问启动页面。

在该页面上,选择“Using the #echo Server-Side Include Directive”链接。这个操作打开“The SSI #exec Directive”页面,如图4-5所示。

这是一个ASP网页ssi_exec.asp。两个按钮用来打开.stm页面,该页面执行其中使用#exec指令所描述的动作。

(1) 在服务器上运行这个示例

在SSI #exec指令示例能够在服务器上工作之前,必须对一些配置进行修改。首先,需要在Web服务器的注册表中创建SSIEnableCmdDirective项(类型DWORD),位置在下面的键名下:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters
```

然后设置该值为1,如图4-6所示。

这样就允许#exec指令与CMD属性一起使用。

其次,必须对包含使用#exec指令的.stm文件的目录禁止匿名访问,客户端将被强制提供



图4-5 “The SSI #exec Directive”网页

帐号的详细情况，该帐号应是一个具有管理员级权限的帐号。这也是 net 命令正常工作的要求。

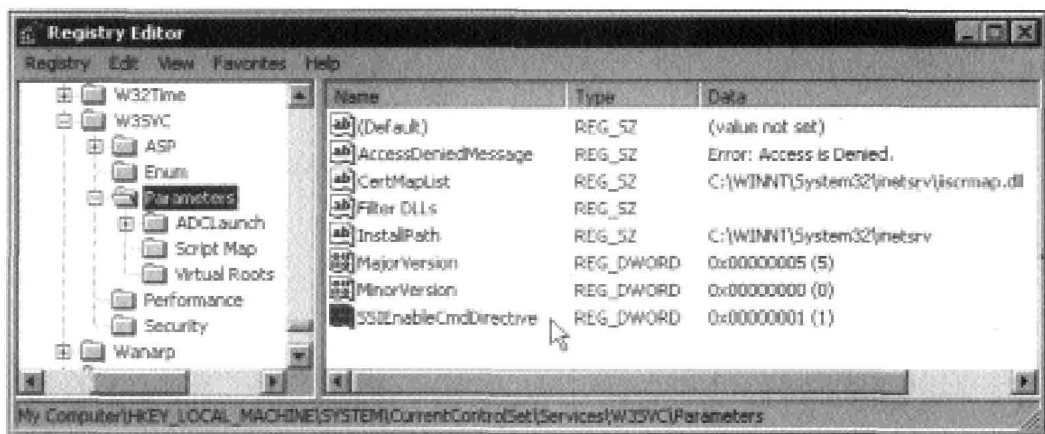


图4-6 注册表中的设置

激活Internet Services Manager应用程序，并选择包含使用 #exec指令的.stm文件的目录(在示例中，这些文件是 exec目录下的 start_cisvc.stm和stop_cisvc.stm)。然后打开该目录的 Properties对话框。在 Directory Security选项卡中单击 Anonymous access and authentication control区域中的Edit按钮，打开 Authentication Methods对话框，如图4-7所示。

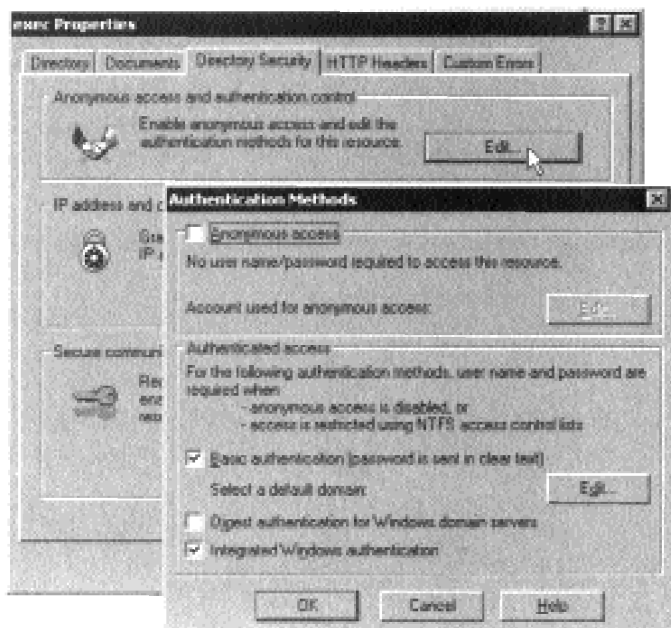


图4-7 设置验证方法的对话框

在这个对话框中不选中 Anonymous access 复选框。如果不使用 Internet Explorer 访问该页面，打开 Basic authentication 选项以允许非 IE 浏览器通过提交用户名/口令访问该页面。设置时，会出现一个有关安全性的警告，单击 Yes。现在浏览器将被强制出示合适的帐号和身份证明，

因为不能匿名访问该网页。

为了能看到启动和终止服务的结果，打开 Services MMC 插件”，终止 Indexing Service，如图 4-8 所示。

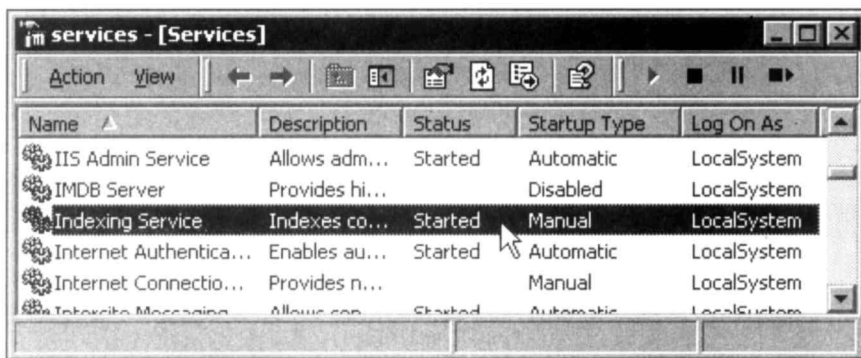


图 4-8 启动和终止服务的屏幕

(2) 启动和终止 Indexing Service

单击示例 Web 网页上的按钮，启动 Microsoft Indexing Service。

这个服务的短名称为 `cisvc`，它通常称为 Microsoft Index Server，名称中的“ci”字符，实际上代表“content indexer”。

出现提示时，输入在 Web 服务器上的具有管理员权限的一个帐号的用户名和口令。当该页面(`start_cisvc.stm`)打开时，你将感觉到一定的延迟，这是因为 `#exec` 指令载入一个窗口命令解释器(`cmd.exe`)的实例，然后执行 `net start` 命令。一旦服务启动(或者如果已经在运行)，将显示该页面的其余部分，如图 4-9 所示。

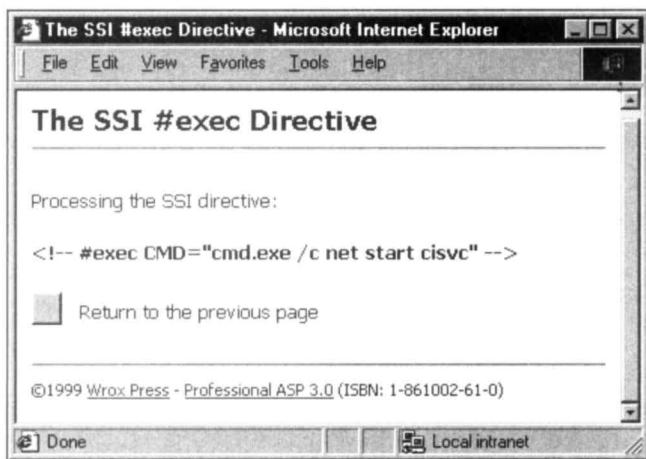


图 4-9 输入用户名和口令后显示的界面

这个页面的代码十分简单。可以看到 `#exec` 指令带有 `CMD` 属性，它设置为“`cmd.exe /c net start cisvc`”。窗体包含有重新回到前一页面的 `SUBMIT` 按钮：

```
<P>Processing the SSI directive:</P>
<P><B>&lt;!-- #exec CMD="cmd.exe /c net start cisvc" --&gt;</B></P>
<!-- #exec CMD="cmd.exe /c net start cisvc" -->
```

```
<FORM ACTION="../../ssi_exec.asp">  
<INPUT TYPE="SUBMIT" NAME="cmdOK" VALUE="&nbsp;&nbsp;&nbsp;">  
&nbsp;&nbsp; Return to the previous page<P>  
</FORM>
```

```
...
<!-- #exec CMD="cmd.exe /c net stop cisvc" -->
...
```

正像在前面看到的那样，通过服务器端包含的经 ISAPI 访问 Web 服务器的页面，用传统的动态页面指令和命令可以做相当多的事情。但同时也有一些明显的限制。

例如，可以从Request.ServerVariables集合检索到所有随同客户端的请求发送过来的 HTTP 报头的值。它几乎可以与使用SSI#echo相匹敌，同时具备的主要优点是把这些值作为字符串返回到代码中（而#echo指令简单地把这些值插入到页面中），因此可以根据自己的愿望来检索和维护这些字符串。许多相同的参数应用于#fsize和#flastmod指令，使用带有VBScript和JScript脚本引擎中的对象的脚本，同样也可以容易地获取这个信息。在后续章节你将看到相关的细节。

#exec指令既非常有用，又受到一定的限制。实际上该指令仅运行系统命令或定制的 CGI 应用程序，并不能给脚本提供真正的对进程的控制。ASP Server对象提供了一种全新的方法，与#exec指令相比，能够更安全 and 更容易地运行其他的应用程序或组件。当然，对于一些情况，尤其是在确实需要执行一个操作系统命令或一个原有的 CGI应用程序的地方，#exec是无法替代的。

为了研究Server对象，先概要介绍其所有可用的方法和属性，然后再进一步详细地进行讨论。

Server对象是专为处理服务器上的特定任务而设计的，特别是与服务器的环境和处理活动有关的任务。因此提供信息的属性只有一个，却有七种方法用来以服务器特定的方法格式化数据、管理其他网页的执行、管理外部对象和组件的执行以及处理错误。

1. Server对象的属性

Server对象的唯一一个属性用于访问一个正在执行的 ASP网页的脚本超时值，如表 4-2所示。

表4-2 Server对象的属性及说明

特 性	说 明
ScriptTimeout	<p>整型。缺省值为90。</p> <p>设置或返回页面的脚本在服务器退出执行和报告一个错误之前可以执行的时间(秒数)。达到该值后将自动停止页面的执行，并从内存中删除包含可能进入死循环的错误的页面或者是那些长时间等待其他资源的网页。这会防止服务器因存在错误的页面而过载。对于运行时间较长的页面需要增大这个值</p>

2. Server对象的方法

Server对象的方法用于格式化数据、管理网页执行和创建其他对象实例，如表 4-3所示。

表4-3 Server对象的方法及说明

方 法	说 明
CreateObject("identifier")	创建由identifier标识的对象(一个组件、应用程序或脚本对象)的一个实例，返回可以在代码中使用的一个引用。可以用于一个虚拟应用程序(global.asa页)创建会话层或应用程序层范围内的对象。该对象可以用其 ClassID来标识，如“{clsid:BD96C556-65A3...37A9}”或一个 ProgID串来标识，如“ADODB.Connection”
Execute("url")	停止当前页面的执行，把控制转到在url中指定的网页。用户的当前环境(即会话状态和当前事务状态)也传递到新的网页。在该页面执行完成后，控制传递回原先的页面，并继续执行Execute方法后面的语句
GetLastError()	返回ASP ASPError对象的一个引用，这个对象包含该页面在ASP处理过程中发生的最近一次错误的详细数据。这些由ASPError对象给出的信息包含文件名、行号、错误代码等等
HTMLEncode("string")	返回一个字符串，该串是输入值string的拷贝，但去掉了所有非法的HTML字符，如<、>、&和双引号，并转换为等价的HTML条目，即<、>、'"'等等
MapPath("url")	返回在url中指定的文件或资源的完整物理路径和文件名
Transfer("url")	停止当前页面的执行，把控制转到url中指定的页面。用户的当前环境(即会话状态和当前事务状态)也传递到新的页面。与Execute方法不同，当新页面执行完成时，不回到原来的页面，而是结束执行过程
URLEncode("string")	返回一个字符串，该串是输入值string的拷贝，但是在URL中无效的所有字符，如?、&和空格，都转换为等价的URL条目，即%3F、%26和+

4.3.2 创建其他对象的实例

在前一章中，讨论了ASP的虚拟应用程序概念，了解了虚拟应用程序通过ApplicationProtection设置为ASP网页中的组件和其他对象提供进程隔离。这沿续了第1章的讨论，即ASP的ObjectContext对象如何为ASP网页提供运行环境，以及如何使用在相同的环境中运行的其他组件和对象。

ASP Server对象提供创建这些组件和应用程序实例的功能，因此可用来扩充ASP脚本的能力。通过实现CreateObject方法的一个特定版本来实现这个功能。

1. 在VBScript和Jscript中创建对象实例

在VB或VBA中，可使用多种方法创建对象的实例。可以使用New关键字来创建指定类型的一个新对象：

```
Dim objNewObject As New MyComponent
```

然而，不能在ASP中用VBScript或JScript这么做，因为这些脚本引擎不能实现数据类型定义。不能声明一个变量为任意指定的数据类型，其变量都是Variants类型，或一个等价的类型(根据使用的脚本语言而定)。

在VB和VBA中另一个方法是使用CreateObject或GetObject方法。CreateObject方法的参数是一个ClassID(通常情况)或一个ProgID字符串，它返回相应类型的一个新对象：

```
Set objNewObject = CreateObject("ADODB.Connection")
```


当拥有一个指定的文档类型，并且想创建一个可以处理这种文档的对象实例时，通常使用GetObject方法：

```
Set objExcel = GetObject("C:\myfiles\sales.xlw")
```

也可以指定所需要的对象类型和文件名，在几种对象都能处理该文档类型的情况下，这种做法是非常有用的：

```
Set objExcel = GetObject("C:\myfiles\sales.xlw", "Excel.Application")
```

VBScript支持CreateObject和GetObject方法。JScript也有getObject方法，与VBScript中的GetObject工作方式相同。JScript中的ActiveXObject实现了与VBScript的CreateObject方法相同的功能。但这个函数常与JScript的new运算符协同使用：

```
objNewObject = new ActiveXObject("This.Object");
```

除了VB的New关键字在VBScript和JScript中不予支持以外，能够使用所有这些技术在一个ASP网页中创建对象的实例。然而，能够并不意味着应该，而且大多数情况下不应该在一个ASP网页中使用脚本引擎的对象创建函数。

2. 在ASP网页中创建对象实例

为了理解一般的脚本引擎对象创建方法为什么在 ASP网页中使用效果不理想，需进一步对ASP中的环境和ObjectContext对象进行讨论。

使用脚本引擎的一般方法在一个 ASP网页中创建一个对象实例时，该对象在当前执行的页面的环境中并未实例化。得不到ObjectContext对象的引用，所以不能使用该对象来访问页面的环境，即不能访问该页面环境中的值。

这意味着该对象不能使用内置的 ASP对象，即不能够访问在 Request、Response、Application和Session对象的集合中的值，也不能使用内置的 ASP对象提供的方法和属性。该对象也不能够与此环境中任何现有的事务进行交互。如果发生错误，不能使用ObjectContext方法放弃一个事务。

当然，你可能不想与该网页的环境进行交互。但是有其他的理由说明使用一般的对象创建方法通常是不明智的。IIS自动地在COM+运行期包装程序hllhost.dll中实例化对象，使得该对象可以在当前的虚拟应用程序中完全地共享和重新使用(缺省的Web网站本身是一个虚拟应用)。

你在上一章所看到的对一个虚拟应用程序的设置，既允许在 Web服务器的内存空间中创建对象，也可以在共享的或独立的进程外 DLLHost.dll实例中创建对象。如果使用一般的脚本引擎对象创建方法，将绕过所有的组件隔离和可扩展特性。而在使用 ASP Server对象的CreateObject方法时会自动地提供这些特性。

3. Server对象的CreateObject方法

为了试验CreateObject方法，打开示例的Chapter04主菜单页，单击“Using the ASP Server Object”链接，如图4-10所示。

这个链接打开一个名为show_server.asp的网页，该页面可以用来试验所有的Server对象的方法。它同时也显示Server对象唯一的属性ScriptTimeout的值、其缺省值是90秒，如图4-11所示。

在该页面的“Create an Instance of a Component”区域，有一个文本框，可以在其中键入想要在该网页的环境中创建的对象的ProgID字符串，甚至可以键入一个ClassID数值。这里文本框的缺省值已经设置为一个来自ActiveX数据对象库的公用对象的ProgID：ADODB.Connection。

...

</FORM>

...

当该页面重新载入时，该页中的一段 ASP 代码(位于<FORM>段的前面)将查看提交该窗体时，单击了哪个按钮。如果是名称为“ cmdCreateObject ”的按钮，该代码将读取文本框中的 ProgID 字符串。为防止用户输入的 ProgID 无效而导致执行中止，关闭缺省的错误处理，再尝试使用 Server.CreateObject 方法创建一个对象的实例。最后，再重新打开缺省的错误处理，通过使用 IsObject 函数检查是否创建了一个对象实例，并显示一个相应的信息：

```

QUOT = Chr(34) 'Double quote character
...
'Look for a command sent from the FORM section buttons
If Len(Request.Form("cmdCreateObject")) Then
    strProgID = Request.Form("txtProgID")
    On Error Resume Next 'Turn off default error handling
    Set objObject = Server.CreateObject(strProgID)
    On Error Goto 0 'Turn default error handling back on

    If IsObject(objObject) Then
        Response.Write "<B>Results:</B><BR>Sucessfully created object with " _
            & "ProgID of <B>" & QUOT & strProgID & QUOT & "</B><HR>"
    Else
        Response.Write "<B>Results:</B><BR>Failed to create object with " _
            & "ProgID of <B>" & QUOT & strProgID & QUOT & "</B><HR>"
    End If
End If
...

```

图4-12所示的是创建 ADODB.Connection 对象的结果。可以看到该对象已被正常实例化，已可以在代码中使用。

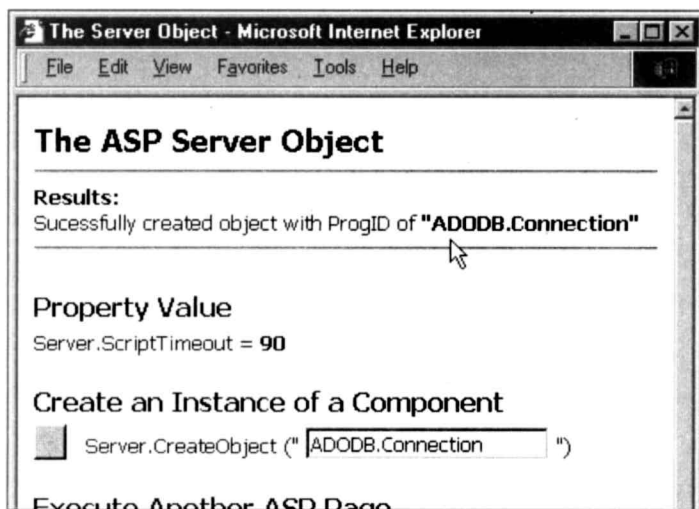


图4-12 ADODB.Connection 对象实例化的结果

本章不讨论如何使用这些对象，你可能已经对此很熟悉了。一旦创建了对象实例，就可以像在任何其他情况中一样使用它。调用对象的方法，读取或设置属性，与使用 VB 时一样；

或者用浏览器中客户端的 VBScript、JScript 使用它。

在接下来的章节中将对对象和组件的使用进行更加详细的介绍。我们将研究由脚本引擎实现的一些对象，以及 IIS 5.0/ASP 3.0 中的可安装组件，还有一些其他的免费或商用的组件，并讨论在各种情况下如何选择相应的组件。在本书的后面，甚至会说明创建自己的能够在 ASP 中使用的组件是非常简单的。

4.3.3 执行其他的网页

ASP 3.0 和 IIS 5.0 的新特性之一就是引入了可编程的服务器端重定向 (server-side redirection) 的概念。这意味着，可以把一个网页的控制和执行转到另外一个网页，而不需要在客户端使用 Response.Redirect 方法。

1. 客户端重定向带来的问题

ASP 编程人员通常使用 Response.Redirect 语句把一个页面载入到当前正在执行的网页。然而，许多人没有意识到这条语句不会自动地使服务器立即装入和执行新的网页。其真正做的是把一个 HTTP 重定向报头 (redirection header) 增加到由 Web 服务器发送给客户的输出流中。这个报头如下：

```
HTTP/1.1 302 Object Moved
Location newpage.asp
```

在这个报头中的标准 HTTP 状态信息 “302 object Moved”，告知浏览器所要求的资源已经发生移动。Location 报头提供相应的网页地址。当然这个地址不一定是真实的，现在正在做的事情就是“欺骗”浏览器，使浏览器认为可在另一个位置上找到所需要的网页。实际发生的是，服务器将执行所请求的网页，但是通知浏览器需要的网页已经发生移动。这就是在发送任何页面的内容到浏览器之前必须执行 Redirect 方法的原因。

当一个浏览器接受到 “302 object Moved” 信息时，中断当前的请求并为 Location 值中指定的网页发送一个新的请求。这与在网页的 <HEAD> 段使用一个 META HTTP-EQUIV 标记时的工作方式相同，前面给出的 HTTP 报头还可写为：

```
<META HTTP-EQUIV="REFRESH" CONTENT="0;URL=newpage.asp">
```

因此重定向实际上发生在客户端，而不是在服务器上。如果在这个连接的客户端有一个代理服务器在使用的话，可能会引起显示虚假消息。代理服务器通常会截取该状态信息，并且可能产生一个页面发送给提出原始请求的客户端。这就是在使用 Response.Redirect 时，“The object you requested has been moved and can be found [here](#)” 消息经常在客户机上显示的原因，正确地使用缓冲通常可以防止这个问题。

在 IIS 4.0 或更早的版本中使用 Response.Redirect 时，应该在 ASP 网页的开头打开缓冲，然后在执行 Response.Redirect 方法之前调用 Response.Clear。当然，在 ASP 3.0 中网页缓冲的缺省状态为打开，因此这不成问题。只要在执行该语句之前使用 Response.Clear，以前产生的输出将不会发送给客户。

2. 在 ASP 3.0 中服务器端的重定向

在 ASP 3.0 和 IIS 5.0 中，在几乎所有情况下，通过使用两个新的 Server 对象方法 Execute 和 Transfer，可以避免使用客户端重定向。这两个方法使控制立即转到另一个网页，该网页可以是一个 ASP 网页或者是任何其他资源，例如一个 HTTP 网页、压缩文件或其他类型的文件。

它们之间的不同之处是：Execute方法“调用”另一个的网页，与在脚本代码中调用一个子程序或函数非常相似。当另一个网页或资源已经执行完毕或传送到客户端时，控制返回到原网页中调用Execute方法的语句的下一条语句，并继续执行。而使用Transfer方法时，控制不再返回到原页面中，在控制传送到的网页或资源的末尾处，执行过程停止。

当前网页的环境也传送给了目标网页或资源，因此这两个方法更有用。网页环境包含了原有的ASP对象中的所有变量的值，例如Request、Response和Session对象的集合以及它们的所有属性。即使该网页不在同一个虚拟应用程序中，也将传送Application对象的环境。

结果是浏览器认为它仍在接收原先的页面，它并不了解服务器所做的事情。浏览器的地址栏一直显示相同的URL，并且Back、Forward和Refresh按钮正常地工作。在使用客户端重定向时，尤其是使用HTML META元素时，情况通常不是这样的。

传送到新的页面或资源的环境包括所有现存的事务状态(transaction state)。当前网页的环境用ASP的ObjectContext对象(在第1章中已经讨论过)进行封装。如果需要将这个对象作为一个正在进行的事务的一部分，可以在传送控制的目的页面中使用这个对象。

(1) Server对象的Execute和Transfer方法的使用

在前面的示例页面中，可以试验使用Execute和Transfer方法。该页面包含了在示例中已经提供的另一个文件的名字another_page.asp，它作为这两个方法的缺省参数值，如图4-13所示。

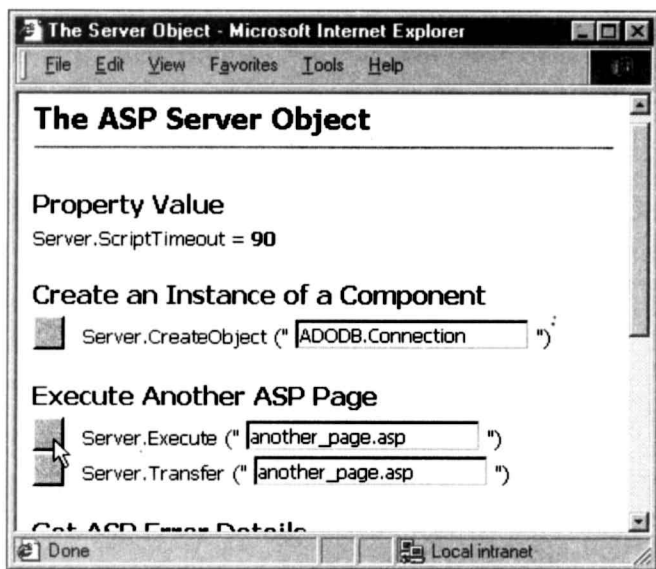


图4-13 使用Execute和Transfer方法的屏幕

单击Server.Execute和Server.Transfer方法的按钮，提交到此窗体并重新装载该窗体。在这个页面顶部的脚本代码查看是哪个按钮被单击。如果是cmdExecute或cmdTransfer按钮，则把当前网页的路径写入到输出流中，然后调用相应的方法，并传送与该按钮相联系的文本框中的值，然后再把当前页面的路径写到输出流中。

```
...
If Len(Request.Form("cmdExecute")) Then
    strPath = Request.Form("txtExecPath")
    Response.Write "Currently executing the page: <B> " _
        & Request.ServerVariables("SCRIPT_NAME") & "</B><BR>"
```



```

Server.Execute (strPath)
Response.Write "Currently executing the page: <B>" _
    & Request.ServerVariables("SCRIPT_NAME") & "</B><BR>"

End If

If Len(Request.Form("cmdTransfer")) Then
    strPath = Request.Form("txtTransferPath")
    Response.Write "Currently executing the page: <B>" _
        & Request.ServerVariables("SCRIPT_NAME") & "</B><BR>"

    Server.Transfer (strPath)
End If
...

```

当单击Server.Execute方法的按钮时，会看到当前页面的路径，这是由上面代码中的第一条Response.Write语句创建并显示的。后面接着的内容是来自被执行的网页（another_page.asp）的一些输出内容。在这之后是第二个Response.Write语句的输出内容，这表明控制又回到了原先的网页。屏幕如图4-14所示。

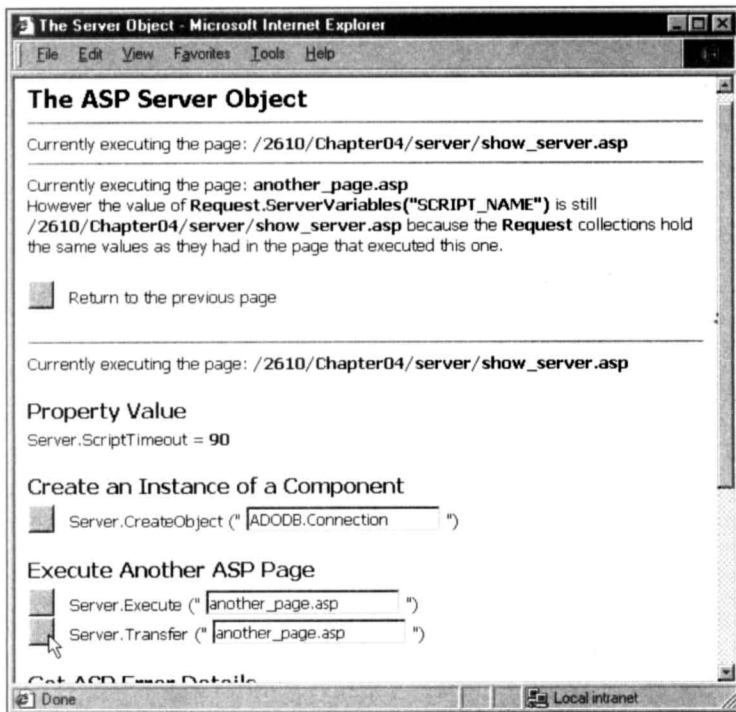


图4-14 Server.Execute方法的演示

页面的两条水平线之间的段落（显示当前执行的网页为 show_server.asp）来自原先的网页。在接下来的段落来自被执行的网页 another_page.asp。下面是该页面的完整代码：

```

<%@ LANGUAGE=VBSCRIPT %>
<HR>
Currently executing the page: <B>another_page.asp</B><BR>
However the value of <B>Request.ServerVariables("SCRIPT_NAME")</B> is still <BR>
<B><% = Request.ServerVariables("SCRIPT_NAME") %></B>
because the <B>Request</B> collections hold<BR>
the same values as they had in the page that executed this one.<BR>

```

注意，该页面执行时，不能使用`Request.ServerVariables("SCRIPT_NAME")`获取它的路径，环境仍然是原网页的。我们不得不把页面名作为文本写入，因为实在没有办法可以从环境中直接获取。

这里包括了一个返回前一个网页的按钮的原因是，通过在主网页中单击相对应的按钮，可以使用 `Server.Transfer` 方法调用这个页面。这次看到了完全相同的输出，只是没有第二次路径输出，因为是“传送”这个页面而不是“执行”该页面，所以控制不会回传给原先的网页，如图4-15所示。

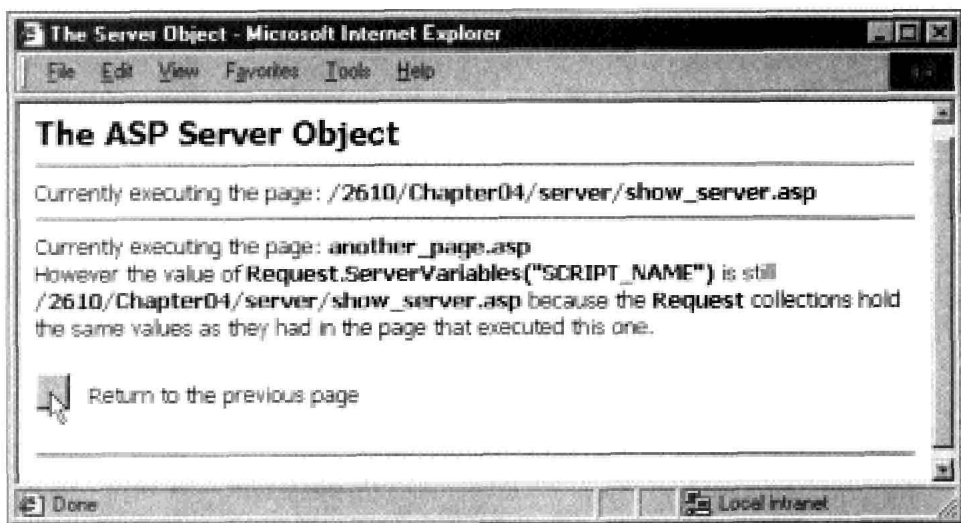


图4-15 Server.transfer的演示

(2) 从ASP执行SSI网页

目前有了一个方法，如果需要的话可在 ASP网页中成功地使用 SSI指令。虽然这种要求不常出现，但可实现。过去的问题是，由于在 SSI网页(文件扩展名是 .stm、.shtml和.shtm)中不能包含 ASP代码，所以程序不能“无缝”地重定向回到原先的网页，必须增加一个按钮或链接，以装载原先的或另外的 ASP网页。

现在，由于有了 `Server.Execute` 方法，可以执行一个 SSI 网页并且将控制自动返回到原先的网页，客户端意识不到这些过程正在进行。客户端只是看到原先的 ASP 网页和执行结果。来自于 SSI 网页的任何输出都“无缝”地插入到流中。当然，如果在 SSI 网页完成后，不想使原先的网页继续执行，可以使用 `Server.Transfer` 方法。

为了看到这个技术的执行，把前面使用过的 CGI-SSI例子网页的虚拟路径输入到 Server.Excute方法(或Server.Transfer方法)的文本框中。这个路径是“../ssi_cgi/ssi_cgi.stm”。在单击按钮对 Execute或Transfer方法进行调用以后，将看到 .stm网页已经执行，其中有 SSI指令的结果。在来自 ssi_cgi.stm的内容之后出现的是原先的网页的其余部分，虽然在图 4-16中看不到，但可通过滚动条看到该内容。

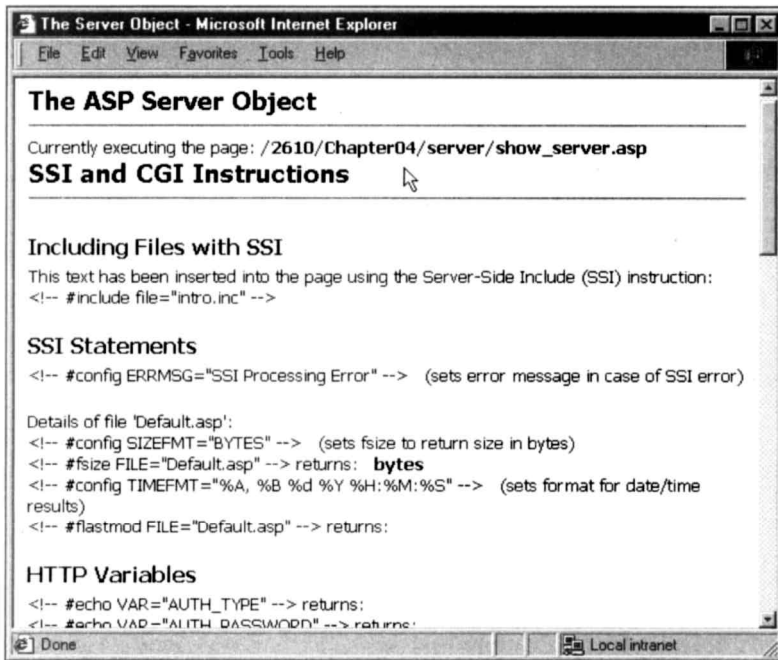


图4-16 执行Server.Execute方法后的屏幕

3. SSI #exec 指令的不足

遗憾的是Execute和Transfer方法一般不能与SSI的#exec指令一起工作，因为包含这个指令的.stm网页会在调用它的ASP网页的环境中运行。在大多数情况下，它需要运行于直接引用该网页的一个独立的环境中。

存在这样的限制真是遗憾，如果没有这种限制，我们通过 Server.Execute执行的网页可以“不可见地”包含来自于ASP网页的#exec指令。对前面的通过net stop和net start命令停止和启动Indexing Service的示例来说，这可能是一种理想的解决方案。

但是，我们必须求助于老的和已经验证的方法。当用户单击一个按钮时，简单地使用Response.Redirect方法来打开相关的网页：

```
<%  
'Look for a command sent from the FORM section buttons  
If Len(Request.Form("cmdStop")) Then  
    Response.Redirect("exec/stop_cisvc.stm")  
End If  
  
If Len(Request.Form("cmdStart")) Then  
    Response.Redirect("exec/start_cisvc.stm")  
End If  
%>
```

可以试着把使用#exec指令的一个SSI网页的虚拟路径输入到示例页面的Server.Execute和Server.Transfer方法的文本框中。前面使用过的#exec示例的虚拟路径是“../ssi_cgi/exec/start_cisvc.stm”和“../ssi_cgi/exec/stop_cisvc.stm”。

4.3.4 Server对象的错误处理

ASP没有错误处理机制一直受到批评。

在VBScript中,有一个On Error Resume Next语句,它使脚本解释器忽略运行期错误并继续脚本代码的执行。接着该脚本可以检查 Err.Number属性的值,判别是否出现了错误。如果出现错误,返回一个非零值。在ASP 3.0中,也可以使用On Error Goto 0“转回到”缺省的错误处理。在ASP 2.0中实际也进行这种处理,但是没有相应文档说明。

在JScript中,有一个新的错误处理功能: C语言风格的try和catch语句。然而所有的这些错误处理技术都不是由ASP或IIS实现的,而是由ASP使用的脚本引擎实现的。

第7章专门讨论脚本和脚本引擎涉及到的调试和错误处理技术。

同时,ASP和IIS的开发小组已经增加了一个新的功能,用于在ASP网页中进行错误处理。这分为两个部分: IIS错误页面的配置及使用ASP的一个新的方法和对象。

1. Server对象的GetLastError方法

在ASP 3.0中,Server对象有一个名为GetLastError的新方法。与VBScript的Err对象不同,不能为查看是否出现了错误而随时调用该方法,只能在一个ASP定制的错误网页中使用。如果像对Err对象进行操作那样,通过关闭缺省的错误处理(用On Error Resume Next语句)来使用,则GetLastError方法不能访问错误的详细数据。

GetLastError方法要做的事情是提供更多的有关错误源和错误原因的信息。GetLastError方法创建并返回一个对象的引用,该对象是一个名为ASPErrors的新对象。这个对象具有一系列的属性,这些属性返回有关在GetLastError方法调用之前出现的最新错误的信息。

2. ASPErrors对象的属性

ASPErrors对象提供了九个属性说明所出现的错误的性质和错误源,并返回引发错误的实际代码,其属性及说明如表4-4所示。

表4-4 ASPErrors对象的属性及说明

属 性	说 明
ASPCode	整型。由ASP/IIS产生的错误号,例如0x800A0009
ASPDDescription	字符串型。如果这个错误是与ASP相关的错误,这个属性是错误的详细说明
Category	字符串型。错误源,即ASP内部脚本语言、或一个对象
Column	整型。产生错误的文件中的字符位置
Description	字符串型。错误的简短说明
File	字符串型。错误出现时正在处理的文件的名称
Line	整型。产生错误的文件中的行号
Number	整型。一个标准的COM错误代码
Source	字符串型。引发错误的行的实际代码

3. 配置“单个网页”错误处理

在IIS中“不可思议”地出现一个错误(例如404 Not Found)时,页面看起来像是从服务器返回给客户端的一个错误信息页面,但实际上并不是这样。它们是普通的HTML网页,在对一个错误进行响应时被下载并且发送给客户端。这些网页通常称为定制的错误网页(custom error page)。

然而,错误网页作为IIS的缺省安装部分,可根据要求定制。事实上,也可以在IIS的早期版本中建立定制的错误网页。

在IIS 4.0中,可以为每种不同类型的HTTP协议或服务器错误指定一个定制的错误网页,为服务器上任意的Web网站中的每个目录建立一个定制的错误信息网页。

(1) IIS缺省的错误网页

由IIS提供的缺省错误页面放在 Web服务器的 WinNT\Help目录中。在 Windows 2000 中的 IIS 5.0 的环境下, 该页面放在 WinNT\Help\iishelp\common目录下, 如图4-17所示。

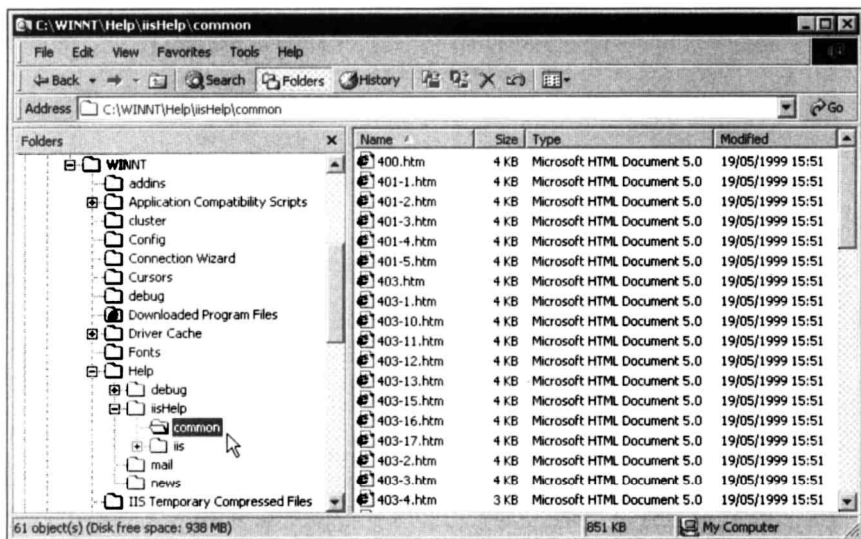


图4-17 缺省的错误页面位置

可在浏览器中打开这些文件查看结果, 或者在文本编辑器中查看 HTML源程序和脚本代码。当一个404错误出现时, 使用的页面是404b.htm, 这个文件包含一个客户端脚本代码部分, 它获得当前文档的URL(从document对象的url属性中检索)并在该页面中显示:

```
The page you are looking for might have been removed, had its name changed, or is
temporarily unavailable.<HR><P>Please try the following:</P>
<UL><LI>If you typed the page address in the Address bar, make sure that it is
spelled correctly.</LI>
<LI>Open the
<SCRIPT>
<!--
    if (((window.navigator.userAgent.indexOf("MSIE") > 0)
    && (window.navigator.appVersion.charAt(0) == "2"))) {
        Homepage();
    }
    //-->
</SCRIPT>
home page, and then look for links to the information you want.</li>
...
<SCRIPT>
function Homepage(){
<!--
    DocURL = document.URL;
    protocolIndex=DocURL.indexOf("://",4);
    serverIndex=DocURL.indexOf("/",protocolIndex + 3);
    BeginURL=DocURL.indexOf("#",1) + 1;
    urlresult=DocURL.substring(BeginURL,serverIndex);
    displayresult=DocURL.substring(protocolIndex + 3 ,serverIndex);
    document.write('<A HREF="' + urlresult + '"' + displayresult + "</a>");
}
//-->
</SCRIPT>
```


这会产生你经常看到的页面，如图 4-18所示。

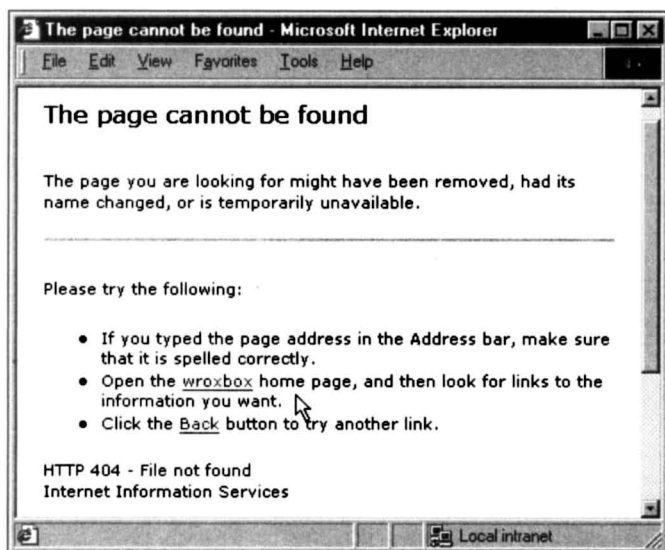


图4-18 产生404错误时的页面

(2) IIS中错误网页的映射

当IIS检测到一个错误时，会把相应的错误页面传送给客户端。如何判别应该向客户端发送那一个页面？很明显，网页的名字应具有解决这个问题的一些信息，但事实上文件名是不重要的。错误和错误网页文件之间的映射关系是在每个目录的 properties对话框的Custom Error选项卡中决定的。

在Internet Services Manager中，在想编辑映射关系的目录上单击右键，并选择 Properties。如果对示例文件进行设置，在 Chapter04目录中选择 server子目录，如图 4-19所示。

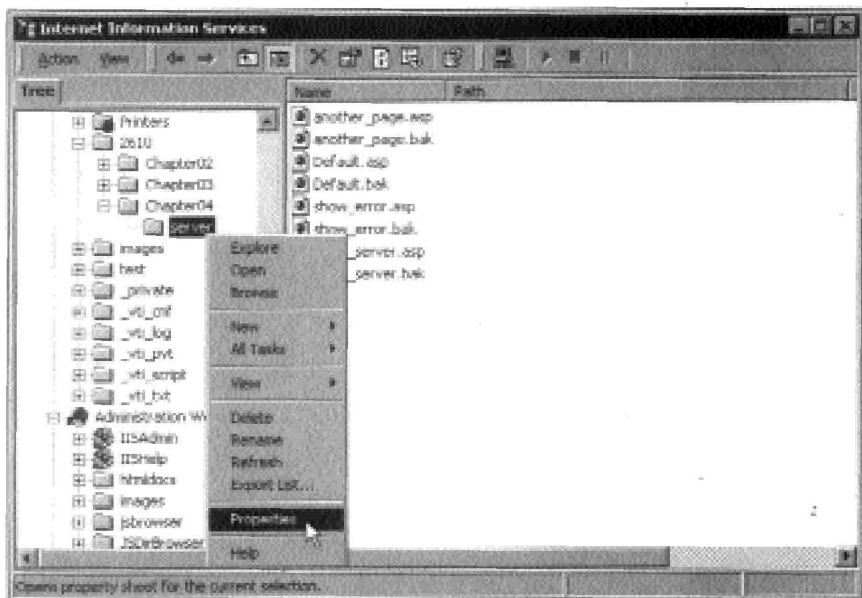


图4-19 设置属性时的页面屏幕

Properties对话框的Custom Errors选项卡在IIS安装时(除非已经进行过修改)设置了缺省映射关系的列表,如图4-20所示。

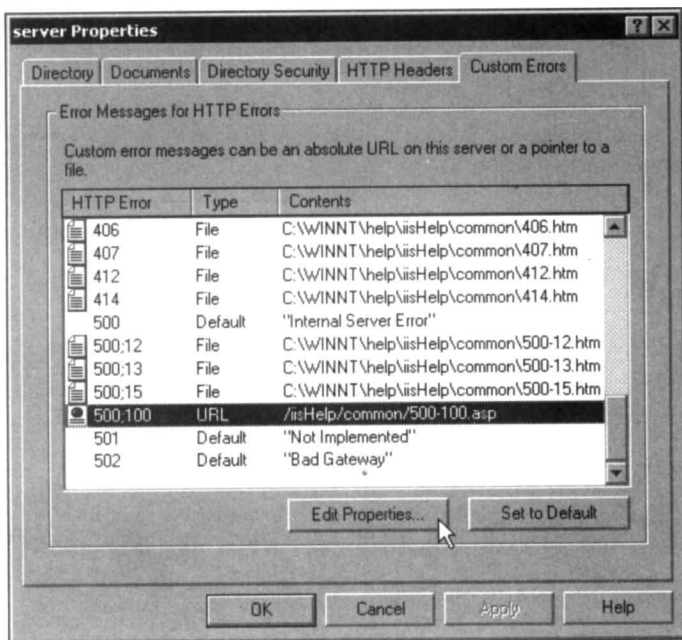


图4-20 映射关系的列表

靠近该列表的底部是HTTP错误500:100的一个条目。类型500错误是由ASP产生的,可以从中看出一些错误已经与错误网页建立了映射关系。这些错误都是一般性的错误,比如“Invalid Application”、“Server Shutting Down”等等。然而,如果ASP载入包含语法错误的页面,或者出现一个运行期错误,则出现500:100错误页面。在列表中显示的缺省映射关系表明,在这个目录中的一个文件出现上述错误时,将执行500-100.asp页面。

当一个ASP错误出现时,我们所看到的信息不再是一个普通的Web网页,而是一个ASP Web网页(也就是说它具有文件扩展名.asp)。也可以根据需要编辑该映射关系来指向另一个页面。

(3) 指定一个定制的错误网页

单击Custom Errors选项卡中的Edit Properties按钮,打开Error Mapping Properties对话框。在Message Type下拉列表中选择URL,键入自己的定制错误网页的完整虚拟路径,如图4-21所示。

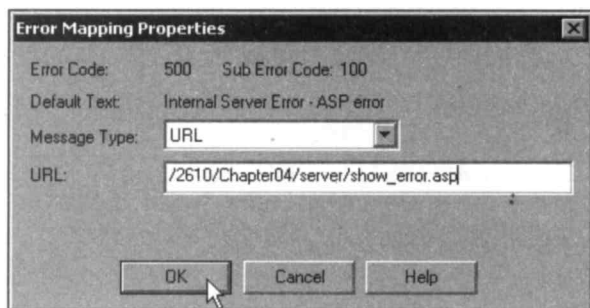


图4-21 指定错误页面的虚拟路径的屏幕

在图4-21中给出的值指向我们创建的与示例网页一起使用的一个定制错误网页。根据你安装示例文件的具体位置，可能要使用不同的路径。

现在无论何时出现一个 500:100类型的错误，将打开我们的定制错误页面。 Message Type的其他两个选项是：

- Default(缺省)：可以简单地输入一个短的文本信息，而不是指定一个发送给客户端的页面。
- File(文件)：指定一个HTTP错误网页的物理路径。

在选择File选项时，指定的网页由IIS载入，载入的方式与在Windows Explorer中双击要载入的文件时的方式相同。这意味着ASP网页不能使用这个选项，因为在这种情况下不会执行其中的任何脚本。

4. 使用GetLastError方法和ASPErrors对象

配置好IIS后，在编辑了错误映射属性的目录内的任一页面上出现一个与ASP相关的错误时，都将载入定制错误页面。实际上，现在已经设置了一个正常的脚本错误陷阱，因为在这个目录内的任何一个网页上的ASP运行期错误都将触发定制错误页面。

事实上在内部IIS通过Server.Transfer方法进行这种操作，这意味着能够访问正在运行的原网页的全部环境。可以在脚本环境中获取信息，这样可以根据所出现的错误决定要做什么。在此基础上，可以在定制的错误网页中检索ASPErrors对象，找到引起载入页面出错的错误的所有信息。

在IIS4.0中，编辑错误映射属性要做一些类似的工作。但是只有一般的500错误(“Internal Server Error”)在映射中是可用的。另外，当定制错误网页载入时，不会传送网页的环境，除了提供一个非特定的错误信息外，做其他任何工作都是比较困难的。

在以前例子中已经使用过ASP Server Object示例页面，在其中可以看到ASPErrors对象的详细情况。单击Server.GetLastError()对应的按钮，如图4-22所示。

这个操作会重新载入该网页，其中的ASP脚本查看点击的是哪个按钮。如果是Server.GetLastError()对应的名为cmdGetError的按钮，将执行一些示例代码，这些代码将会产生一个运行期脚本错误。

```
...  
If Len(Request.Form("cmdGetError")) Then  
    Dim arrThis(3)  
    arrThis(4) = "Causes an error"  
End If  
...
```

因为已对这个目录设置了错误网页映射，即配置为装入定制错误页面，所以当错误出现时，就打开这个页面(通过Server.Transfer方法在后台不可见地工作)，见图4-23。

(1) 示例错误网页代码的功能

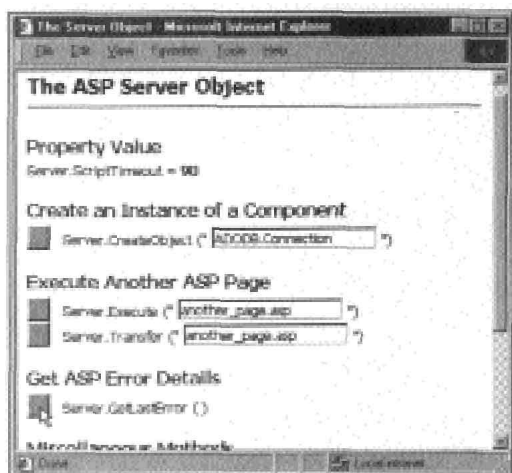


图4-22 查看ASPErrors对象的详细情况的屏幕

ASP内部对象集合或属性中的值。例如，如果检索来自 `Request.ServerVariables` 集合的 `HTTP_REFERER` 值，它将反映调用原网页的网页（即在错误出现之前的网页）的URL。在服务器把执行转到错误网页时，这个值不会发生变化，并且它不包含当错误发生时正在执行的网页的URL。

同样，`SCRIPT_NAME` 值将是包含该错误的网页的名字，而不是错误网页的URL。在一个错误网页已经装入时，通过检查浏览器地址栏中的URL，可以对此进行确认。但是在原网页的脚本变量中存储的值，在定制的错误网页中都是不可用的。

如果原ASP网页正在一个事务内运行，即在网页的最前面包含有一个 `<% @TRANSACTION= "..."%>` 指令，也应该确定是否需要在网页中采取一些方法，以退出该事务。例如可以调用内置 `ObjectContext` 对象的 `SetAbort` 方法：

```
ObjectContext.SetAbort 'Fail the transaction if an ASP error occurs
```

在本书的后面将介绍与事务的相关全部内容。

(2) 使用 `ASPErrors` 对象的属性

关于使用 `ASPErrors` 对象的属性，有以下几点值得注意的：

- 即使没有出现错误，`Number` 属性应该一直有一个值。如果ASP网页调用 `GetLastError` 方法时没有错误出现，该属性的值是0。通常情况下，对ASP脚本的运行期错误，`Number` 属性返回十六进制的值“0x800A0000”，加上标准的脚本引擎错误代码。例如，前面的例子对“Subscript out of Range”错误的返回值为“0x800A0009”，因为VBScript对该类型错误的错误代码是“9”。
- 当出现已经过一个错误时，`Category` 和 `Description` 属性将一直有一个值。
- `ASPCode` 属性的值由IIS产生，对大多数脚本错误将为空。更多情况下，涉及外部组件使用出错时有相应的值。
- `ASPDescription` 属性的值由ASP预处理程序产生，而不是由当前正在使用的脚本引擎产生的，并且对大多数脚本错误而言将是空的。更多情况下，对诸如对ASP内置对象调用无效的方法的错误有相应的值。
- `File`、`Source`、`Line` 和 `column` 属性仅在错误出现时，并且在错误的详细数据是可用的情况下才能进行设置。对一个运行期错误，`File` 和 `Line` 属性通常是有效的，但是 `column` 属性经常返回-1。当错误是一个阻止页面被ASP处理的语法错误，才返回 `Source` 属性。一般在这些情况下，`Line` 和 `Column` 属性是有效的。如果把 `Source` 属性的值写到页面，明智的办法是先将该值传给 `HTMLEncode`，以防在其含有非法的HTML字符。在本章的后面详细将地讨论 `HTMLEncode` 方法。

4.3.5 获取Server对象的路径信息

在对存储在Web网站上的文件进行操作时，需要获得文件的实际的物理路径，而不是使用虚拟路径或URL，尽管在其他网页中能用它们正常地定位文件。下一章中有一个例子，它使用 `FileSystemObject` 对Web站点的 `InetPub\WWWRoot` 文件夹中的文件进行读写。当创建自己的定制组件或者使用商业化的组件对文件系统进行访问时，经常需要为其提供一个文件的物理路径。

`Server` 对象的 `MapPath` 方法

可以从 `Request.ServerVariables` 集合中提取HTTP报头变量，它们包含了当前文件的物理路

径(在DOCUMENT_NAME和PATH_TRANSLATED报头中)。Server对象提供了一个方法MapPath,可以使用这个方法对我们能够提供一个有效的虚拟路径的任何文件提取相应的物理路径。可以在已经使用过的示例网页中看到使用MapPath方法,并可试验使用该方法。

如图4-24所示,在页面的底部的Miscellaneous Methods部分,有一个按钮执行Server.MapPath方法,并提供给它靠近该按钮的文本框中的值。在该网页的源代码中已经把该值设置为“/ishelp/default.htm”,这个文件应该自动地安装在计算机上。也可以输入另一个网页的URL。

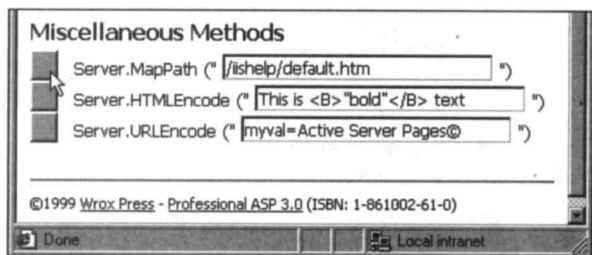


图4-24 使用Server.MapPath的屏幕

单击该按钮重新装载这个网页,执行该方法并在顶部显示结果,在下部显示原页面的其余部分,如图4-24所示。

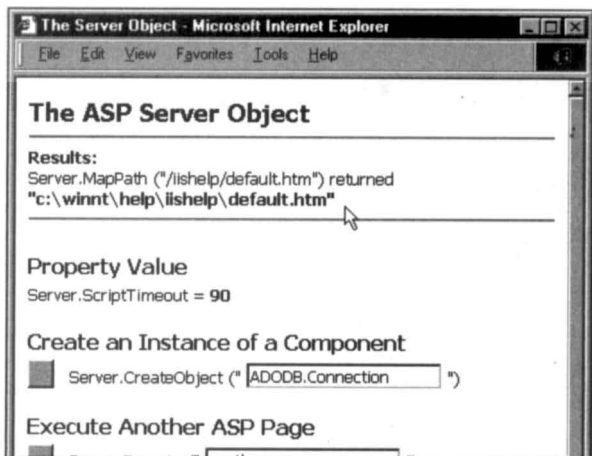


图4-25 显示Server.MapPath的结果

(1) 示例网页代码的功能

处理这个过程的代码是与前面在相似的示例文件中已经使用过的代码十分相似。

在该页面顶部的ASP脚本区域中,对单击的按钮的名字进行检查。在这种情况下,该按钮的名字将是cmdMapPath,简单地把相匹配的文本框中的值txtMapPath传送给Server.MapPath方法,并显示得到的结果:

```
If Len(Request.Form("cmdMapPath")) Then
    strValue = Request.Form("txtMapPath")
    Response.Write "<B>Results:</B><BR>Server.MapPath (" & QUOT & strValue _
        & QUOT & ") returned <B>" & QUOT & Server.MapPath(strValue) _
        & QUOT & "</B><HR>"
End If
```

(2) MapPath和虚拟应用程序目录

注意，MapPath方法为/iishelp/default.htm文件获取的结果在 Web服务器目录外，并在主winnt目录的help目录中。这清楚地证明了MapPath方法是非常有用的。

对于在缺省的 Web网站目录中的文件，其 URL的路径部分与物理路径通常是相同的。例如，一个文件存储在 Web服务器上：

```
C:\InetPub\WWWRoot\yourfiles\thisfile.asp
```

如果安装时已经在缺省目录中安装了的 Web根目录，则URL如下：

```
http://yoursite.com/yourfiles/thisfile.asp
```

然而，IIS Help文件安装在缺省 Web网站根目录外的一个虚拟目录中，所以用于对其进行访问的URL和物理路径之间没有直接的关联。只有通过使用 Server.MapPath方法才能获取真实的物理路径。

4.3.6 使用Server对象格式化数据

当前面讨论演示SSI指令的网页的代码时，碰巧遇到了使用 HTML的一个老问题。在一个 HTML网页中如何显示HTML代码？如果“照现在的样子”使用，也就是在相应的位置上使用所有的HTML字符，会被浏览器当作HTML进行解释和执行。这样当下列内容在浏览器中显示时：

This is the syntax of a <TABLE> element:

将不会显示文本<TABLE>，因为浏览器将其作为一个数据表的一个开始标记，并照此来执行。为了避免这种情况，必须把在 HTML中非法或无效的所有字符转换到等价的 HTML字符实体(character entity)。多数常见的字符如表4-5所示。

表4-5 字符与等价的HTML实体的关系

字 符	等价的HTML实体	字 符	等价的HTML实体
<	< ;	>	> ;
&	& ;	"	" ;
©	© ;	®	® ;

所有的实体以&号开始并以分号结束，是在一些语言中表明一个实体的标准方法的一部分，这些语言是基于SGML(标准化常规标识语言)规则的，如HTML语言。

1. 数字的HTML实体等价字符串

注意最后一个例子，已注册的商标符号®是一个以“#”字符为前缀的数字值，而不是相应含义的一个文本缩写(如copy对应版权符号©)。具有一个大于126 的ANSI代码值的所有字符在HTML中被表示为十进制字符的ANSI代码，以&#为前缀，以分号为后缀。

事实上，需要留心的是使用数字实体等价字符串要优先于一些较少被支持的文本实体字符串。一个例子是商标字符(®)，该字符的实体等价字符串为"™"但不是所有浏览器(例如Navigator)都能识别这个字符串，这种情况下，将在网页中显示该实体字符串。相反，使用™在所有浏览器中都能很好地工作。

2. Server对象的HTMLEncode方法

把HTML转换为文本是进行有效显示需要的，否则 HTML会被浏览器当成HTML来对待和

执行，这意味着必须对无效的字符进行编码，使其成为等价的 HTML 实体字符串。为管理这种转换，Server 对象提供了 HTML Encode 方法。可以在本书提供的 ASP Server Object 示例网页中练习使用这个方法。

简单地把一些文本输入到 HTML Encode 对应的文本框中并单击按钮。示例中提供了一些真实的 HTML 作为缺省的文本，如图 4-26 所示。

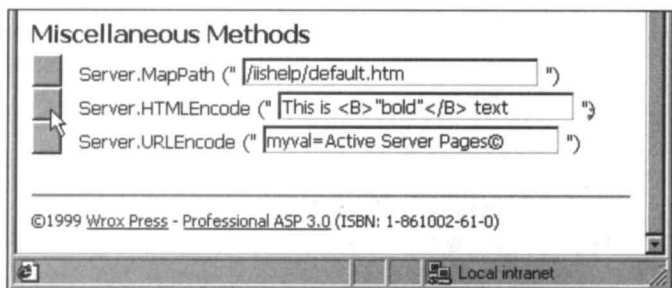


图4-26 使用HTML Encode方法的屏幕

重新载入该页面时，在该页面的顶部显示结果。HTML Encode 方法把尖括号转换成了 “<” 和 “>”，而且把双引号转换成为 “"”，如图 4-27 所示。

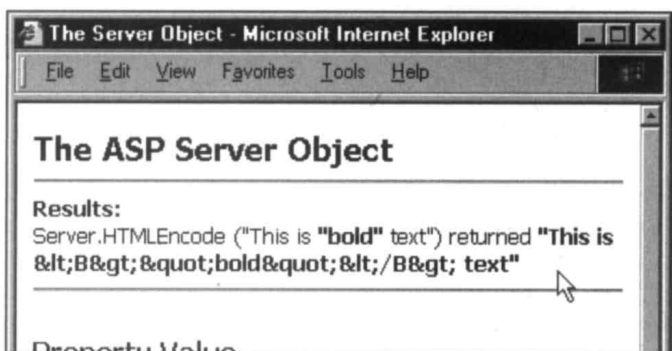


图4-27 使用HTML Encode方法1

(1) 示例网页代码的功能

关于得到的结果，有几个有趣地方。

首先，在方法名字后面的括号中已经丢掉了 和标记，相应增加了一个粗体文本部分。在网页中显示原有的值时，和被当成 HTML 提交了，所以和标记消失了，相关内容以粗体文本显示。

可以十分容易地避免这种情况。事实上，这就是设计 HTML Encode 方法的原因。原示例代码如下：

```
...
Response.Write "Server.HTML Encode (' & QUOT & strValue & QUOT & ') returned"
...
```

现在所能做的就是将 HTML Encode 方法应用于正在输出的值上：

```
...
strResult = Server.HTML Encode(strValue)
Response.Write "Server.HTML Encode (' & QUOT & strResult & QUOT & ') returned"
...
```

现在输出了一个十分有用的结果，如图 4-28所示。

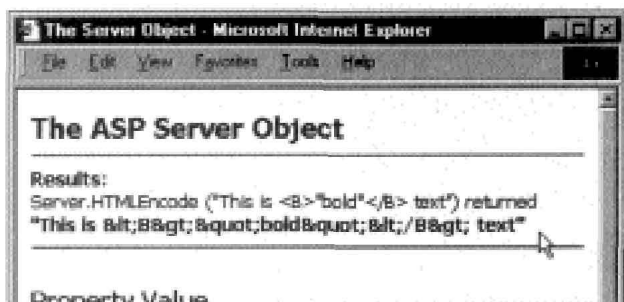


图4-28 使用HTMLEncode方法的结果2

现在已经解决了不提交 HTML而显示 HTML的问题。但是如果要在 HTML中显示 HTMLEncode方法的结果，而又不提交和处理这些结果，又会发生什么情况？为了解决这个问题，要从HTMLEncode方法本身考虑：

```
This is &lt;B&gt;&quot;bold&quot;&lt;/B&gt; text
```

上面的语句在HTML网页中得不到同样的显示结果，这是因为 HTML字符实体将被浏览器处理和执行，并显示为实体所替代的字符。换句话说，得到的是：

```
This is <B>"bold"</B> text
```

我们没看到实体。为了避免这种情况，可两次使用 Server.HTMLEncode方法。这就把“&”号变换为“&”，这样就得到了所需的显示结果。示例网页的这个部分的代码是：

```
If Len(Request.Form("cmdHTMLEncode")) Then
    strValue = Request.Form("txtHTMLEncode") 'Get the value from the text box
    strResult = Server.HTMLEncode(strValue) 'HTMLEncode to convert <, > and "
    strDisplay = Server.HTMLEncode(strResult) 'Then again to convert & to &amp;
    Response.Write "<B>Results:</B><BR>Server.HTMLEncode (' & QUOT & strResult _
        & QUOT & ') returned <B>' & QUOT & strDisplay & QUOT _
        & "</B><HR>"
End If
```

(2) HTMLEncode与HTML控件的缺省值

从上面可以看出，如果在一个HTML网页中要显示HTML代码，而又不使之被作为HTML进行处理和执行，HTMLEncode方法是非常有用的。在大多数普通的ASP网页中不大可能会遇到这种情况，除非使用包含有HTML的数据库或其他数据源中的数据，而又需要作为文本进行显示。

但是HTMLEncode方法真正有用的地方是，通过设置VALUE属性预设页面中文本类型的HTML控件的值。作为例子，可看一下已经用于练习HTMLEncode方法的示例网页的源程序。创建HTMLEncode对应的文本框的HTML在HTML页中定义如下：

```
...
<INPUT TYPE="TEXT" NAME="txtHTMLEncode" SIZE="35"
    VALUE="This is <B>&quot;bold&quot;</B> text">
...
```

这是“手工编码”而不是使用 Server.HTMLEncode方法。这里也只关心对双引号进行编码而不关心对尖括号的编码。为什么？这是因为如果没有这样做，该代码将被读为：

```
VALUE="This is <B>"bold"</B> text">
```

而在这种情况下尖括号不会带来问题，未编码的双引号则会。在文本框中替换的实际值将是 "This is "，即它将在第二个双引号字符处被截断。所以，在创建预置控件值的页面时，应该考虑使用HTMLEncode方法，以避免这些值被截断：

```
<%
strValue = Request.Form("txtSomeValue")
%>
...
<INPUT TYPE="TEXT" NAME="txtSomeValue"
      VALUE="<% = Server.HTMLEncode("strValue") %>">
...

```

当浏览器发送已经被 HTML 编码的一个控件的值给服务器时，自动进行反向译码。即服务器使用Request集合中原来格式的数据。

3. 格式化URL的数据

还有另外一种情况，就是经常需要把一个文本字符串变换成能够在 Web 网页中使用的另外一种格式。现代 Web 服务器和操作系统都十分友好地支持包含空格字符的文件名，但是我们所使用的URL可能包含有空格字符，由于 HTTP 使用的URL语法不允许有空格字符 (和几个其他字符)，可能会出现麻烦。

另外一种更普遍的情况也会出现麻烦。当把这些值作为 QueryString 集合的成员传送给服务器时，将被追加到 URL 的末尾 (在一个问号字符之后)。这种情况发生在 <FORM> 的 METHOD 属性被设置为 "GET" (或者是省略了 METHOD 属性) 的情况。换句话说，对于直接追加到 URL 上的值，都可能出现麻烦。这可能发生在 <A> 元素中：

```
<A HREF="http://myserver.com/mypage.asp?title=Instant JScript">
Instant JScript</A>
```

一些浏览器 (例如 Internet Explorer) 可以对此进行处理，因为它们在把 HTTP 请求发送到服务器之前，自动地执行必要的转换。然而，许多其他的浏览器不进行这种转换，并导致了 URL 通常在第一个空格或非法字符处被截断。这样在 Navigator 中，上面给出的链接要求的网页变为 http://myserver.com/mypage.asp?title=Instant。在服务器上，title 名字/值对的丢失部分会使代码失败。

考虑到 HTTP 协议定义的限制，必须从作为 HTTP 请求中的 URL 使用的字符串中删除非法的字符 (非法字符是所有那些 ANSI 代码在 126 之上的字符和 ANSI 代码在 126 以下的某些字符)。

ANSI 代码大于 126 的字符必须用百分号后跟十六进制形式的 ANSI 代码进行替换。这样，版权字符 (c) 变成 %A9。ANSI 代码在 126 之下在 URL 中不合法的字符，同样使用相应的替代字符串；如表 4-6 所示。

表4-6 字符与HTTP/URL代替物的关系

字 符	HTTP/URL代替物	字 符	HTTP/URL代替物
空格	+	\	%5C
'	%27]	%5D
!	%21	^	%5E
#	%23	`	%60
\$	%24	{	%7B

(续)

字 符	HTTP/URL代替物	字 符	HTTP/URL代替物
%	%25		%7C
&	%26	}	%7D
(%28	+	%2B
)	%29	<	%3C
/	%2F	=	%3D
:	%3A	>	%3E
;	%3B	Chr(10)	忽略
[%5B	Chr(13)	%0D

4. Server对象的URLEncode方法

Server对象提供了可以用来把任意字符串转换成相应的合法 HTTP URL的方法。可以利用示例网页对这个名为URLEncode的方法进行练习，如图4-29所示。

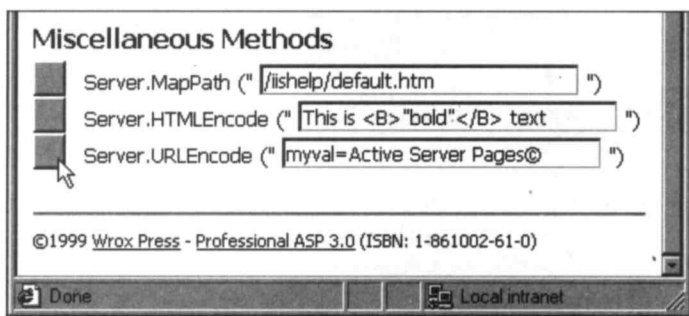


图4-29 使用HTMLEncode方法的屏幕

这里，输入的值作为URL是非法的，它包含了空格和ANSI代码大于126的字符。对这个值，使用URLEncode方法的结果是所有的空格被替换成一个加号，版权符号被替换为 %A9，如图4-30所示。

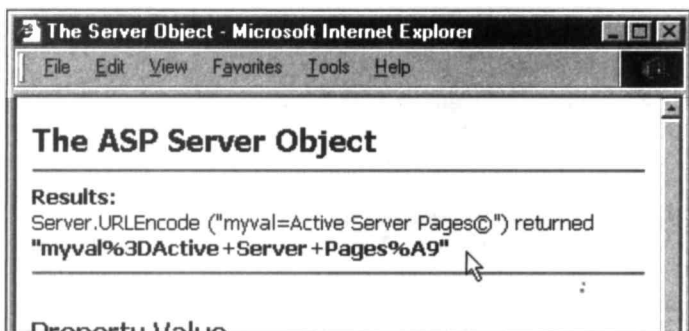


图4-30 使用HTMLEncode方法的结果

(1) 示例网页代码的功能

在示例网页中，处理这个功能的代码非常简单，仅仅检查是否单击了 URLEncode方法对应的按钮，如果单击了，把对应的文本框中的值传递给 Server.URLEncode方法并显示结果：

```
If Len(Request.Form("cmdURLEncode")) Then
    strValue = Request.Form("txtURLEncode")
```

```
Response.Write "<B>Results:</B><BR>Server.URLEncode (" & QUOT & strValue _
               & QUOT & ") returned <B>" & QUOT & Server.URLEncode(strValue) _
               & QUOT & "</B><HR>"
```

```
End If
```

(2) 对HTML元素和其他链接使用 URLEncode

URLEncode方法更普遍地用于把<A>元素或其他链接的值写到ASP网页。例如，如果在查询字符串中建立了一系列的链接，这些链接包含来自一个数据库的值，首先应该对这个字符串使用Server.URLEncode方法：

```
<%
strValue = Request.Form("txtSomeValue")

'Create the full URL for the link as an HTTP-legal string
strURL = "http://mysite.com/books.asp?title=" & Server.URLEncode("strValue")

'Make sure we don't have any non-legal HTML characters in the page text
strLink = Server.URLEncode(strValue)
%>
...
<A HREF="<% = strURL %>"><% = strValue %></A>
...

```

如果放入字符串strValue的值包含标题“Active Server Pages(c)”，将得到由这个代码段创建的如下所示的HTML：

```
<A HREF=" http://mysite.com/books.asp?title=Active+Server+Pages%A9">
Active Server Pages&copy;</A>
```

注意，我们不仅仅使用Server.URLEncode方法来建立一个合法的URL字符串，而且还对链接的文本使用了Server.URLEncode方法，以确保把所有非法的字符转换为合适的HTML等价实体。

和URLEncode方法一样，不用反译码ASP网页中的URL编码值。IIS自动地实现URL编码字符串的转换，该字符串在HTTP请求中转换为它们原先的格式，使得它们在内置对象中是可用的。

4.4 小结

在这一章中，通过在Web服务器上发生的处理过程，讨论了为Web网页提供动态内容所涉及的一些问题。这些问题的一部分不是直接地与ASP本身相关，但对这些问题的理解，将有助于理解基本的处理工作是如何进行的。

本章介绍了IIS如何支持传统的服务器端包含指令，有一些指令可能仍然是有用的。特别是，#exec指令对执行系统命令以及集成原有的应用程序都是有用的。同时也讨论了一条特别的服务器端包含指令——#include语句，了解了在ASP网页内部使用这条命令的相关问题。

然而，ASP Server对象占了本章的大部分。它提供了在ASP网页内管理服务器端处理过程的方法。在Web服务器和ASP的正确的环境中，它可用来创建其他对象、应用程序或组件的实例。它同时也提供了一系列的方法，这些方法允许执行其他的网页或资源，以及以正确方式格式化信息，以便在ASP脚本和网页中使用。

Server对象也带来了一个新的ASP内置对象：ASPErrors对象，它为脚本提供较好的错误处理方法。现在可以提供“正统的”脚本错误处理，并获取有关错误的信息。

本章的主要内容是：

- 传统的服务器端包含 (SSI) 指令的背景和用法。
- ASP Server 对象所要完成的任务，以及与 SSI 的比较。
- 如何使用 Server 对象实例化和使用外部组件和应用程序。
- 如何使用 Server 对象执行封装的脚本或其他 ASP 网页。
- 如何使用 Server 对象管理在脚本中出现的错误。
- 如何使用 Server 对象完成实现 HTML 或 HTTP 兼容的格式转换。

现在，已完成了对 ASP 内置对象的学习，在接下来的章节中将讨论如何扩充网页的功能。利用 Server 对象创建其他对象和组件的实例的能力，我们可以建立几乎能做任何事情的网页。