第20学时　对HTTP和CGI进行操作

在本学时中，你将要学习如何对 Web进行一系列有趣的操作。可以使用 CGI程序，使Web站点变得更加灵活，并且更加便于管理。

在本学时中，你将要学习：

- 如何将HTML程序从服务器传送到你的浏览器。
- 如何使CGI程序能够发送HTML文档。
- 如何将值直接传递给CGI程序。
- 服务器端的包含程序如何运行。
- 如何查询浏览器和服务器，以便找到你要的信息。

20.1　HTTP通信概述

在第17学时中，我们介绍了 Web浏览器（Netscape和Internet Explorer等）与Web服务器（Apache和IIS等）之间如何进行基本的通信。该学时介绍的通信方式显得过分简单了一些。现在我们对CGI程序的使用变得更加得心应手了，因此可以更加深入地探讨这个问题。在本学时的后面部分中，我们将要介绍进行这种通信时使用的一些方法，以便执行某些有意思的任务。

这种通信方式可以用一个协议来加以描述，这个协议称为超文本传输协议（ HTTP）。该协议目前的两个版本是HTTP 1.0和HTTP 1.1。在本学时介绍的一些例子中，两个版本均可适用。

描述Internet上使用的这些协议的 Internet标准文档称为 “Request For Comment(说明请求)”，即通常所说的 RFC。RFC由Internet工程组负责维护，你可以通过网址 http://www.ietf.org在Web上查看。专门介绍 HTTP的文档是RFC 1945和RFC2616。请注意，这些文档的技术性很强。

当你的Web浏览器初次与Web服务器连接时，浏览器向服务器发送一条初始消息，它类似下面的形式：

```
GET http://testserver/ HTTP/1.0
Connection: Keep-Alive
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Charset: iso-8859-1,*,utf-8
Accept-Encoding: gzip
Accept-Language: en, en-GB, de, fr, ja, ko, zh
Host: testserver:80
User-Agent: Mozilla/4.51 [en]C-c32f404p  (WinNT; U)
```

GET用于指明你试图接收的是什么 URL，以及你想要接受的是哪个版本的协议。在这个例子中，你接受的是HTTP 1.0版的协议。

connection行用于指明你希望这个连接为检索多个 Web页保持打开状态。按照默认设置，浏览器为检索每一帧、每一页和 Web页上的每个图形分别建立一个连接。命令 keep-Alive要求服务器使连接保持打开状态，以便使用相同的连接检索多个项目。

Accept行用于指明通过这个连接你愿意接受何种类型的数据。第一个 Accept行的结尾处的*/*表示你愿意接受任何种类的数据。下一行（ iso-8959-1等）表示字符编码可用于该文档。在这个例子中， Accept-Encoding表示gzip（GNU Zip）可用于对来自服务器的数据进行压缩，以便加快传输速度。最后， Accept-Language用于指明该浏览器能够接受何种语言（英语、大不列颠英语、德语和法语等）。

Host是你希望租用的Web站点的系统名。由于可以使用虚拟租用，因此该系统名可以不同于URL中的主机名。

最后一行，该浏览器将自己的身份通知 Web服务器，这个身份是 Mozilla/4.51[en]C-c32f404p（WinNT; U）。在Web技术中，该浏览器称为用户代理。

然后，服务器发送一个应答消息，它类似下面的形式：

```
GET http://testserver/ --> 200 OK
Date: Thu, 02 Sep 1999 19:54:39 GMT
Server: Netscape-Enterprise/3.5.1G
Content-Length: 2222
Content-Type: text/html
Last-Modified: Wed, 01 Sep 1999 17:12:03 GMT
```

这时，该应答消息后随你要检索的 Web页内容。

在这个消息中的GET行用于指明服务器是否将这个 Web页发送给你。状态200表示一切运行正常。服务器还在 Server行上标明自己的身份。在这个例子中，该服务器是 Netscape-Enterprise/3.51 G Web服务器。

Content-Length行表示2222字节的内容将被发送给浏览器。使用这个消息，你的浏览器就能够知道一个Web的内容完整性是50%还是60%等。Content-Type是发送给浏览器的 Web页的种类。如果是HTML页，这一行就设置为text/html。如果是图形页，它就设置为 image/jpeg。

Last-Modified日期表示自从该Web页上次被检索以来是否被修改了。大多数 Web浏览器都将Web页缓存起来，这样你就可以两次阅读一个 Web页，这时，该日期就可以与浏览器已经拥有的保存拷贝日期做比较。如果服务器上的Web页尚未修改，就没有必要再次下载整个Web页。

20.1.1 举例：人工检索Web页

如果你愿意的话，可以人工检索 Web页。当想要测定Web服务器发送的是否是正确的 Web页时，常常可以使用这个特性。

若要运用这个特性，需要一个专门的程序，称为 Telnet客户程序。Telnet客户程序是个远端访问程序，用于远程登录到 UNIX工作站。不过它常常用于执行调试 HTTP之类的任务。

如果你有一台UNIX计算机，可能已经安装了 Telnet。如果你拥有一台 Microsoft Windows计算机， Telnet可能已经作为你的网络实用程序的一部分安装好了。你只需要打开 Start菜单，使用Run选项，就可以运行Telnet客户程序。如果尚未安装该程序，或者使用的是 Macintosh计算机，你可以在任何较好的下载站点找到免费的 Telnet客户程序。

若要启动与Web服务器的通信，请在提示符处输入下面这个 Telnet命令：

```
$ telnet www.webserver.com 80
```

这里的www.webserver.com是Web服务器的名字， 80是你想要连接到的端口号（端口 80通常是Web服务器接收信息的端口）。如果你的Telnet客户程序是个图形处理程序，你必须在对话框中设置这些值。

当Telnet进行连接时，你可能看不到提示符或连接消息。请不必担心，这是正常的。HTTP期望客户机首先发出请求，而服务器却没有发出提示。在 UNIX下，你会得到一条消息，其内容如下：

```
Trying www.webserver.com
Connected to www.webserver.com
Escape character is '^]'
```

其他类型的系统，如Windows和Macintosh，则看不到这条消息。

你必须认真和迅速地键入下面这行命令：

```
GET http://www.webserver.com/ HTTP/1.0
```

键入这行命令后，请按Enter键两次。这时Web服务器应该作出响应，发出正常的 HTTP标题和Web站点的顶层页，然后切断连接。

20.1.2 举例：返回非文本信息

你的CGI程序不一定将HTML信息返回给浏览器。实际上，你的浏览器能够检索的任何信息，CGI程序都能够发送。

CGI模块中的header函数告诉浏览器，它准备使用 MIME内容类型（Content-Type）标题来接收何种类型的数据。 Content-Type标题用于描述后随的数据内容，这样，浏览器就知道如何处理该数据。

按照默认设置， header函数将一个text/html的内容类型描述发送给浏览器。浏览器识别后随的内容是带有HTML的文本。

通过告诉浏览器将会收到不同类型的数据，你就可以控制浏览器如何来处理该数据。数据可以作为图形来显示，也可以传递给浏览器的插件，甚至可以由浏览器启动的外部程序来运行。

若要使header函数能够发送非普通 text/html标题的某些信息，请使用 -type选项，如下所示：

```
print header(-type => MIME_type);
```

可以发送给浏览器的某些常用 MIME内容类型是 text/plain（指不需要转换的文本），image/gif和image.jpeg（指GIF和JPEG图形），以及application/appname（指应用程序appname特定的数据）。一种特殊的MIME内容类型称为application/octet-steam，它是指浏览器应该保持到一个文件的原始二进制数据。

如果你需要创建一个“当日图形”的Web站点，或者创建一个Web标题广告，就可以使用这种内容类型。每天修改 Web页，以便反映新图形的变化情况，这是很麻烦的。如果你出差在外，谁为你更新“当日图形”呢？为此，你可以使用一个静态 HTML页，并且使用Perl CGI程序，每天自动产生一个不同的图形。

在你的Web页中，使用下面这样的HTML代码：

```
<BODY>
Today's image of the day is:
<IMG SRC="/cgi-bin/daily_image.cgi">
</BODY>
```

在上面这个HTML代码中，请注意<IMG>标记的目标是个CGI程序，不是.gif或.jpg。接着，你需要一个放满图形的目录，图形的数量至少要与一个月的天数相同。你可以调用你喜欢的

任何图形，只要文件名以.jpg结尾即可。请注意，该程序能够非常容易地使用 GIF图形。

CGI程序daily_image.cgi类似程序清单20-1所示的形式。

程序清单20-1 当日图形的代码

```
1: #!/usr/bin/perl -w
2:
3:   use strict;
4:   use CGI qw(:all);
5:   my($imagedir, $day, @jpegs, $error);
6:
7:   $imagedir="/web/htdocs/pic_of_day";
8:   $error="/web/htdocs/images/error.jpg";
9:
10:  sub display_image {
11:      my($image)=@_;
12:      open(IMAGE, "$image") || exit;
13:      binmode STDOUT; binmode IMAGE;
14:      print <IMAGE>;
15:      close(IMAGE);
16:      exit;
17:  }
18:
19:  print header(-type => 'image/jpeg');
20:
21:  # Day of the month, 1-28, 29, 30, or 31
22:  $day=(localtime)[3];
23:  $day=$day-1;     # We want day 0-27, etc..
24:
25:  opendir(IMGDIR, $imagedir) || display_image($error);
26:  @jpegs=sort grep(/\.jpg$/, readdir IMGDIR);
27:  closedir(IMGDIR);
28:
29:  my $image="$imagedir/$jpegs[$day]";
30:  $image=$error if (not defined $jpegs[$day]);
31:  display_image($image);
```

第7行：这一行用于设定图形所在的目录。可以修改这个设置，以指明你将图形放在什么位置。

第8行：这一行非常奇怪，因为这个 CGI程序并不输出文本，并且因为它嵌入到的 HTML页时并不将输出显示为文本，你不能只是输出出错消息。如果无法打开 $imagedir目录，则变量$error包含将要显示的.jpg文件的名字。

第10~16行：这个子例程将图形显示在标准输出中，它将发送到浏览器。在 Windows平台上，STDOUT被视为一个文本文件，将.jpg输出到STDOUT将会损坏图形。因此，binmode用于使STDOUT和IMAGE成为二进制文件句柄。在 UNIX下，你不需要使用binmode，但是它不会造成损害。请注意第12行，如果图形打不开，就没有必要输出出错消息，程序只要退出即可。

第19行：这一行用于输出标准 HTTP标题，不过Content-Type将是image/jpeg，而不是通常的text/html。

第25行：图形目录被打开以便读取。如果图形目录没有打开，那么便用错误图像 $error来调用函数display-image（）。

第26行：这一行比较复杂，因此要循序渐进来操作。首先用 readdir读取目录。然后从该列表中取出以.jpg结尾的文件名。最后，对产生的列表进行排序，并赋予 @jpegs。

20.2 如何调用CGI程序的详细说明

到现在为止，我们介绍了启动CGI程序时使用的两种方法。第一种方法也是最简单的方法是通过一个链接来调用 CGI程序的URL，或者让用户将 URL键入浏览器。因此类似下面的这行代码可以用于启动和运行称为 time.cgi的程序：

```
<A HREF="http://server/cgi-bin/time.cgi">Click here for the time</A>
```

当通过该链接进行操作时， CGI程序time.cgi将由服务器运行，它的输出则作为一个新Web页来显示。这个示例代码简单明了，容易操作，与第 17学时中的“ Hello， World！ ”很相似。

启动CGI程序的另一个方法是使它成为一个 HTML填充式窗体的目标程序。例如，当单击Subamit按钮时，下面这个窗体便调用 CGI程序process.cgi：

```
<FORM METHOD=GET ACTION="/cgi-bin/process.cgi">
<INPUT TYPE=TEXT NAME=STUFF><BR>
<INPUT TYPE=SUBMIT>
</FORM>
```

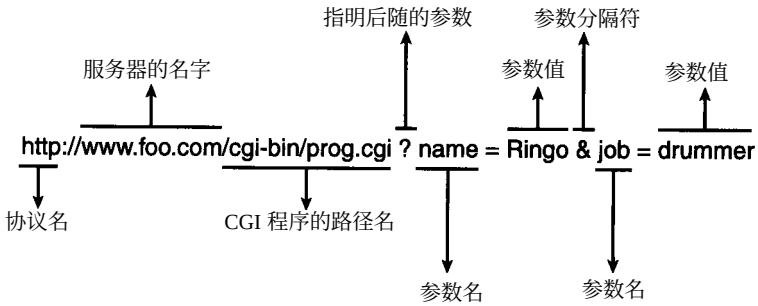
这个调用CGI程序的方法具有另一个优点，即你可以将参数传递给 CGI程序，以便进行处理。好了，这就是HTML窗体的总体情况。

20.2.1 将参数传递给CGI程序

通过链接将信息传递给Perl程序，这是不是很好呢？例如，是否可以在文档中设置一个点击的链接，它将“ 运行CGI程序foo.cgi，其X值等于this， Y值等于 that ”呢？你稍加努力，就可以达到这个目的。

首先必须在<A HREF>标记中使用一种特殊的URL。该URL的格式在图20-1中做了说明。

图20-1 包含各个参数的URL



每个参数都是你想要传递到 CGI程序中的一个值的名字（类似一个指明的 HTML窗体元素），该值是该名字的值。例如，若要创建一个链接，单击这个链接时，它将运行一个 CGI程序，其参数sign设置为Aries，year设置为1969，那么你可以输入下面这行代码：

```
<A HREF="http://www.server.com/cgi-bin/astrology.cgi?sign=Aries&year=1969">
Aries, year of the Rooster</A>
```

在这个CGI程序中，它的参数将像通常那样由 CGI模块的param函数进行处理：

```
#!/usr/bin/perl -w

use CGI qw(:all);
use strict;
```

```
print header;
print "The year ", param('year'), " and being born under ",
    param('sign'), "indicates you are brilliant.\n";
```

你可以根据需要传递任意数量的参数。如果你想传递一个空参数，即没有值的参数，只需要像下例中的author那样将它置空即可：

```
<A HREF="http://www.server.com/cgibin/book.cgi?author=&title=Beowulf">Beowulf</A>
```

20.2.2 特殊参数

当你调用带有此类参数的CGI程序时，应该了解使用某些特殊参数时要考虑的问题。某些字符属于特殊字符，不能成为 URL的组成部分。例如，？（问号）是个特殊字符，它可以作为URL的主要部分与参数之间的分隔标号。其他特殊字符还有 &、空格和引号等。

特殊字符的完整列表在Internet标准文档RFC 2396中列出。

若要将这些特殊字符中的某一个插入 URL，你必须对字符进行转义。在这种情况下，对字符进行转义意味着应该将它的 ASCII值转换成一个两位数十六进制数字，并在它的前面加上一个百分比符号。对“ Hello,World! ”的编码如下所示：

```
Hello%2C%20World
```

显然，创建一个URL转义字符串是非常麻烦的。 CGI模块提供了一个函数，它能够自动为你创建这样的字符串。下面这个代码段展示了如何输出一个带有正确编码的 URL：

```
#!/usr/bin/perl -w

use strict;
# The 'escape' function must be pulled in manually
use CGI qw(:all escape);

print header;
my $string="Hello, World!";
print '<A HREF="http://www.server.com/cgi-bin/parrot.cgi?message=
' ,    escape($string) , '">Click Me</A>';
```

上面这个代码可以产生一个正确进行 URL字符转义的HTML链接。请注意CGI 模块是如何用于代码的use CGI qw（:all escape）；。如果你使用CGI模块，那么escape函数通常不能供你的程序使用，你必须显式要求使用该函数。

下面这个程序创建了一个带有转义值的长得多的 URL：

```
#!/usr/bin/perl -w

use strict;
use CGI qw(:all escape);

my %books=( Insomnia => 'S. King',  Nutshell => 'O\'Reilly');
# Start with a base URL
my $url="http://www.server.com/cgi-bin/add_books.cgi?";

# Accumulate on the end of the URL with concatenation "."
foreach my $title (keys %books) {
    $url.=escape($title);  # Escape the title, add it
    $url.="=";
    $url.=escape($books{$title}); # Same with Author
```

```
    $url.="&";
}

print header;
print "<A HREF=$url>Add books to library</A>";
```

当CGI程序用param函数取出这些参数时，在 URL的结尾处的最后一个 &将被该CGI程序忽略。

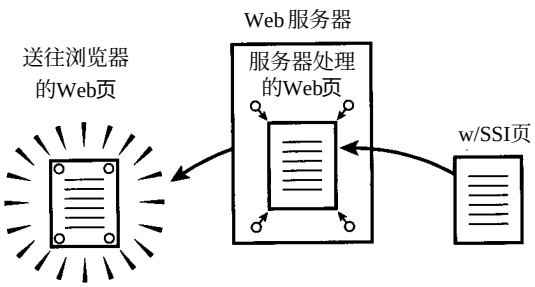
```
http://www.server.com/cgi-
bin/add_books.cgi?Insomnia=S.%20King&Nutshell=O%27Reilly&
```

20.3 服务器端的包含程序

当你设计Web页时，该页上的最常见的内容是静态的。有时该页的某些部分要进行修改，但是总的来说，该页的内容将保持不变。请看这样一个 Web页，它显示了一家公司的当前股票价格。该页的主要部分是静态的，比如导航栏、图形、徽标、使用信息、页眉、页脚和标题等。该页的重要部分，即股票价格，是通过读取某处的数据库和填写空格来生成的。

为了帮助你创建这种Web页，大多数Web服务器都支持一个特性，称为服务器端的包含程序（SSI），也称为服务器分析的 HTML。该特性使得 Web站点的开发者能够创建基本静态的 HTML Web页，并使该页的某些部分由 Web服务器在运行中重新编写（见图 20-2）。可以将该Web页视为填空式HTML文件，而CGI程序则为你将数据填入空格。

图20-2 当HTML页被处理时，Web服务器将数据填入该页



你的服务器管理员必须激活 SSI，使这些示例代码能够运行。为了使服务器能够正确地读取带有嵌入式 SSI的HTML，有时你必须为 HTML赋予带有.shtml或.stm扩展名的名字。请与你的服务器管理员联系，以便了解 SSI是如何在你的特定Web服务器上使用的，因为它支持的命令及其语句是各不相同的。

当Web服务器从磁盘上读取静态 HTML页时，它要寻找它能够替换的各个值的“ 标记 ”。在服务器分析的 HTML页中的 Apache Web服务器下，标号 <!--#echo var=“ LAST_MOCIFIED ”-->将使Apache能够替换标记中该 Web上次被修改的日期。浏览器看不到这个进程的发生，它只看到服务器替换时的这个日期。下表说明了这个进程的情况：

Web页	转换成的内容
<HTML>	<HTML>
<BODY>	<BODY>
This page was last changed:	This page was last changed:

(续)

Web页	转换成的内容
<!--#echo var= "LAST_MODIFIED" -->	Wednesday, 01-Sep-1999 21:29:31 EDT
</BODY>	</BODY>
</HTML>	</HTML>

Web服务器实现SSI的方法各有差异。有时标记的句法各不相同，有些服务器支持某些种类的标记，但是有些服务器则不支持这些标记。有些Web服务器根本不支持SSI。例如，Microsoft的Personal Web服务器就不支持SSI。本学时中使用的SSI HTML标记与Apache Web服务器及Microsoft的Internet Information服务器的标记是兼容的。在撰写本书时，后面两种服务器是Web上最流行的Web服务器。

本学时并不打算全面介绍SSI的所有特性，因为这些特性的数量太多，而且大多数特性是某些特定品牌的Web服务器所特有的。我们的目的是要介绍SSI标记#exec。你可以像下面这样将SSI #exec标号用于HTML文件：

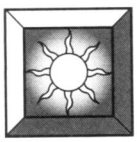
```
<!--#exec cgi="/cgi-bin/stockprice.cgi"-->
```

当Web服务器遇到这个#exec标记时，它将停下来执行stockprice.cgi这个CGI程序。该CGI程序的输出在送往浏览器时被插入HTML数据流。当CGI程序完成运行时，#exec标记后面的HTML文件的剩余部分被送往浏览器。

举例：使用SSI

在这个例子中，你将创建一个简单的Web页，用于输出“Hello,World”，然后根据一天中的时间，输出一个定制图形。首先你需要两个图形，一个用于晚上，另一个用于白天，如图20-3所示。

图20-3 白天和晚上使用的两个图形：day.jpg和night.jpg



day. jpg

night. jpg

然后你需要一个HTML样板文件，它带有问候你的信息，如下面的代码所示。如果你为自己创建这个文件，请记住必须在给HTML文件命名时使用扩展名.shtml或.stm，以便使服务器能够识别SSI标记。

```
<HTML>
<HEAD>
<TITLE>Welcome Page</TITLE>
</HEAD>
<BODY>
Welcome to this web page.  Currently, out my window I see:
<!--#exec cgi="/cgi-bin/sunmoon.cgi"-->
</BODY>
```

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

在程序sunmoon.cgi中，可以使用程序清单20-2所示的代码。

程序清单20-2 白天和晚上的问候程序

```
1:   #!/usr/bin/perl -w
2:
3:   use CGI qw(:all);
4:
5:   # The hour from localtime() is in 24-hour format
6:   my $hour=(localtime)[2];
7:   my $image;
8:
9:   # Before 6am or after 6pm, it's nighttime
10:  if ($hour<6 or $hour>18) {
11:          $image="night.jpg";
12:  } else {
13:          $image="day.jpg";
14:  }
15:  print header;
16:  print qq{<IMG SRC="$image" ALT="$image">\n};
```

第3行：由于这是个CGI 程序，因此你应该使之包含 CGI模块。qw（:all）用于确保你能够使用需要的任何函数。

第6行：列表上下文中的localtime返回一个描述当前时间的元素列表。这个问题已经在第4学时中做了介绍。localtime前后的括号将它置于一个列表上下文中， [2]则使该列表的第三个元素得以返回，并赋予$hour。元素#2是用24小时的格式来表示的时间。

第15行：标题仍然必须使用 CGI的header函数来输出，尽管这个输出显示在整个 Web页的中间位置上。

第16行：<IMG>标记既可以用$day的值来输出图形，也可以用 $night的值来输出图形。如果浏览器无法显示图形，则使用 ALT（替代）标记。

当浏览器在上午8点钟检索Web页时，产生Web页的源代码将如下所示：

```
<HTML>
<HEAD>
<TITLE>Welcome Page</TITLE>
</HEAD>
<BODY>
Welcome to this web page.  Currently, out my window I see:
<IMG SRC="day.jpg" ALT="day.jpg"z
</BODY>
</HTML>
```

20.4 部分环境函数简介

到现在为止，来自CGI模块的大多数函数都是用于控制浏览器的（如redirect或header函数），或者用于处理传递给 CGI程序的参数（如escape和param函数）。CGI模块中的全部函数都是为了向你提供关于你当前正在运行的系统的信息的。表 20-1显示了部分函数的列表，如果你要查看完整的列表，请在命令行提示符处键入 perldoc CGI，以便查看CGI模块的在线文档。

这些函数大多需要使用Web服务器提供的值，或者使用Web浏览器使用HTTP协议发送的值。Web浏览器可能提供某些虚假的值，比如referer值或use_agent，Web服务器有时也会返回不准确的值，例如，server_name并不总是返回你期望的值。

表20-1　部分环境函数一览表

函　数	说　明
referer	返回将你发送到该Web页的链接的URL。（不错，这个函数拼写错了。原先描述这个域的Internet标准包含一个拼写错误，现在国际上也将错就错，以便做到统一。）
user_agent	返回一个字符串，用于指明要求检索 Web页的浏览器的种类（如 Netscape、IE、Lynx）
remote_host	返回要求检索Web页的系统的主机名或IP地址。你究竟会得到哪个值，取决于你的Web服务器的配置和是否存在主机名
script_name	返回正在运行该程序的程序名，作为部分 URL（如/cgi-bin/foo.cgi）
server_name	返回托管CGI程序的服务器的名字
virtual_host	返回用于运行该CGI程序的虚拟主机名。该函数不同于server_name，因为一个服务器常常能够托管多个Web站点。Virtual_host返回被访问的特定Web站点的名字

下面是用于展示这些函数的一个短程序：

```
#!/usr/bin/perl -w

use strict;
use CGI qw(:all);

print header;

print "You were sent from: ", referer, "<BR>";
print "You are apparently running: ",
                user_agent, "<BR>";
print "Your system is called: ", remote_host,
                "<BR>";
print "The name of this program is: ",
                script_name, "<BR>";
print "It's running on the server: ",
                server_name, "<BR>";
print "The server's calling itself: ",
                virtual_host, "<BR>";
```

在一个测试用的Web服务器上运行该程序，将产生下面的结果：

```
You were sent from: http://testsys.net/links.html
You are apparently running: Mozilla/4.51 [en] (Win95; I)
Your system is called: 192.168.1.2
The name of this program is: /cgi/showstuff.cgi
It's running on the server: testsys
The server's calling itself: perlbook
```

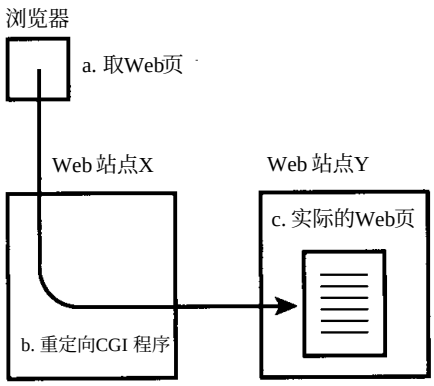
20.5　重定向

CGI程序中可以使用的一个非常有用的方法称为 HTTP重定向。当想要根据某个计算值让CGI程序加载另一个Web页时，就可以使用重定向。

例如，如果你有一连串的Web页要送往一个特定的浏览器，比如，这些 Web页包含一个插件，而这个插件只有在 Microsoft Windows的Netscape浏览器下才能使用，那么可以将该 Web

站点的访问者送往同一个URL，并让一个CGI程序将他们重定向到正确的Web页。图20-4说明了这个情况。

图20-4 一个Web站点重定向到另一个Web站点



若要实现重定向，需要使用CGI模块的redirect函数。redirect函数用于对前面介绍的HTTP通信进行操作，并使浏览器能够加载一个新Web页。

程序清单20-3包含一个短程序，用于将Windows下的Netscape用户重定向到一个Web页，并将所有其他浏览器重定向到另一个Web页。

程序清单20-3 根据浏览器进行重定向

```
1:  #!/usr/bin/perl -w
2:
3:  use CGI qw(:all);
4:  use strict;
5:  my($browser, $target);
6:
7:  # Fetch the browser's name
8:
9:  $browser=user_agent;
10: $target="http://www.server.com/generic.html";
11:
12: # Test for WinXX and Netscape
13: if ($browser=~/Mozilla/ and $browser=~/Win/) {
14:         $target="http://www.server.com/netscape.html";
15: }
16: print redirect( -uri => $target );
```

第9行：在$browser中抓取浏览器类型。

第10行：默认URL被放入$target中。任何非Netscape浏览器均被送往这里。

第13~14行：存放在$browser中的浏览器类型被核实，以确定它是否包含Mozilla或Win，如果包含，则赋予一个新目标地址。

第16行：重定向消息被发送到浏览器。

通过CGI的重定向是天衣无缝的，而通过其他方法（如使用JavaScript和HTML扩展名）进行的重定向则存在许多问题。并非所有平台都支持JavaScript，使用JavaScript中的window.location.href赋值语句可能无法产生正确的结果。如果将HTML的<META HTTP-EQUIV=“refresh”>标记用于重定向，就会在重定向进行之前产生明显的延迟，因为浏览器必须在重定向发生之前全部加载Web页。JavaScript也同样存在这个问题。HTTP重定向是在发送任何HTML之前发生的，并且几乎是即时进行的。

对于CGI模块的user_agent函数，Netscape的浏览器将自己标识为 Mozilla。这个名字是图形 Web浏览器原先的名字 Mosaic的变形。 Windows 95下的典型Netscape 4.51浏览器返回的user_agent名字类似Mozilla/4.51 - （Win95；I）。

20.6 课时小结

在本学时中，我们介绍了从服务器中检索 Web页时程序运行的情况，并且简要地讲述了HTTP协议。还介绍了如何通过链接来调用CGI程序，并将参数传递给程序，这可以用于SSI。此外，还介绍了如何进行HTTP通信，以便执行重定向操作，并获取关于浏览器和服务器的信息。

20.7 课外作业

20.7.1 专家答疑

问题：SSI示例代码似乎无法运行，为什么？

解答：SSI代码不能运行的原因很多。首先，应该检查你的 Web浏览器是否支持SSI，并非所有的Web浏览器都支持SSI。其次，应该确保Web服务器激活了SSI特性。第三，应该确保你的HTML文件拥有正确的扩展名，以便激活 SSI。你可以与服务器管理员取得联系，了解上述信息。最后，应该确保你的 HTML SSI标号使用的句法的正确。

如果你使用<!--#exec cgi-->标记，应该确保你在不使用 SSI来运行程序时你的 CGI程序运行正确。

在Web页已经加载的情况下使用浏览器中的“ view source（查看源代码）”选项，就能够知道服务器是否正在执行你的 SSI程序。如果你看到 Web页源代码中的SSI标记，服务器就不能识别和分析它们。

问题：Telnet示例代码无法运行，为什么？

解答：如果Telnet未能建立连接，那么应该确保你是针对 Web服务器的名字来使用 Telnet的，并且使用的端口是正确的，也许它是端口 80。你必须查看Telnet客户程序的文档，以便正确地设置端口号。

另一个常见问题是你无法看到自己键入的字符。有些 Telnet客户程序能够将你键入的字符反馈给你，有些则不能。请不必对此担心，你只需要认真进行操作就行了。这些字符必须认真发送。当你键入GET行后，务必按两下Enter键。

20.7.2 思考题

1) 下面这个URL能够按照你的期望运行吗？

```
<A HREF=" /cgi/foo.pl? name=Ben Franklin&Job=printer">
```

a. 是。

b. 否。你不能像这样将两个参数传递给一个 CGI程序。

c. 否。名字Ben Franklin中的空格是不允许的。

2) 服务器端的包含程序由什么来进行处理和展开？

a. 浏览器。

b. Web服务器。

c. 操作系统。

20.7.3 解答

1) 答案是c。你应该使用转义符正确地隐藏空格和其他特殊字符。

2) 答案是b。Web服务器负责将SSI HTML标记转换成它们的值，然后将它们发送给浏览器。

20.7.4 实习

- 使用Telnet客户程序，连接到你喜欢的Web站点之一，并设法人工检索Web页。