第24学时　建立交互式Web站点

如果你在Web站点上使用CGI程序的目的是使访问者能够进入你的站点，并且为他们提供一个可以访问的有价值的站点，那么除了建立访问计数器外，还必须设计一个更好的站点。

Web上最有价值的站点是能够提供频繁更新内容的站点。如果你的 Web页上的信息是静态的，人们就没有理由再次访问它。经过几次访问后，他们就会知道你的站点没有太大的变化，因此不会再来访问。

要使人们重访你的Web站点，方法之一是让他们在某种程度上参与站点的活动。人们作为一个群体，喜欢互相交谈。人们希望成为一个群体的成员，并且具有参与感，这是人的一个共性。

本学时中介绍的程序提供了一个进行每项操作的工具。第一个程序提供了一种方法，用于扫描Internet上的另一个信息源的内容，改变该内容的格式，并且在你的站点上显示这些内容，同时加上一些警告。第二个程序使得你的站点的访问者可以参与一个调查活动。

在本学时中，你将要学习：

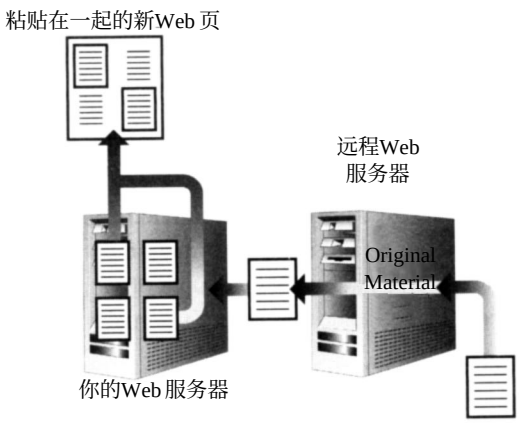
- 如何借用另一个Web站点的内容。
- 如何创建一个交互式民意测验站点。

24.1　借用另一个站点的内容

也许你看到过这样的Web站点，在Web页的某个位置显示了最新的股票行情信息、新闻标题或体育比赛的比分，尽管运行该站点的人与发布这些内容的机构并无关系。

出现这种情况的通常原因是：在你查看的系统上运行的程序常常可以访问信息的原始站点，并将信息拖拉过来，然后改变信息的格式，显示在目标 Web页上。图24-1显示了这个进程的运行情况。

图24-1　取出一个 Web页，改变其格式，然后将它重新显示



这里你会看到你的Web服务器通过它的CGI程序，已经变成一个Web客户机。当服务器为自己检索Web页时，它能将这些页连接在一起，然后再次显示该信息。

24.1.1　注意内容的版权问题

在你继续读下去，了解如何借用其他站点的内容之前，有几个问题必须了解。首先，如

果你要显示的信息不是你自己的信息，而是来自另一个 Web站点或数据库的信息，那么这些信息也许会受到版权法的保护。从另一个 Web站点那里借用信息，然后在你自己站点上显示，这会使你陷入严重的法律争端。如果触犯版权法，可能导致你的 Web站点和ISP被关闭，你可能被罚款、监禁或者受到法律指控。

触犯版权法也是一种粗鲁的行为。

如果你想在自己的 Web站点上使用其他来源的信息，首先要获得其他信息源的许可。大多数Web站点运营商允许你显示来自他们站点的内容。他们通常要求你考虑下列几个问题：

- 你应该清楚地注明内容的来源，可以用标题、链接和文字来注明。
- 你应该清楚地注明内容的版权，说明该内容是经过允许而使用的。
- 也许你无权通过“深层次链接”访问他们的 Web站点，也就是说通过几个层次的链接访问他们站点的Web页。他们希望你只链接到顶层的 Web页。
- 你只能偶尔更新你的 Web页版本。将其他站点的服务器隐藏在信息之下，以便使你的站点看上去更加出色，这种做法是不可取的。

Slashdot.org是个为技术用户提供内容的 Web站点，它的经营者允许我们使用他们的站点来演示本书中的示例代码。当你在自己的 Web页上运行这些示例代码之前，应该征得该站点的同意。如果要与 Slashdot.org取得联系，你可以在 http://www.slashdot.org上通过他们的 Web站点和FAQ找到有关的详细说明。

24.1.2 举例：检索标题

如果要在你的Web站点上显示Slashdot站点的新闻标题，请按下列步骤操作：

1) 通过服务器分析的HTML Web页，启动headlines.cgi CGI程序。

2) 然后该CGI程序查看它是否拥有存储在磁盘上的最新新闻标题拷贝。如果有，就使用之。如果没有，便从Slashdot.org的Web站点检索这些标题。

3) 接着CGI程序分析标题文件并显示标题。

若要从另一个 Web站点中检索 Web页或其他内容，需要一个模块，它不是标准 Perl模块LWP::Simple的组成部分。LWP模块使你能够从Internet上检索所有类型的信息，比如 Web页、FTP数据、新闻组文章等。

LWP::Simple模块被封装为libwww-perl模块包的一部分。这个模块包包含了许多个模块，分别用于检索 Web页、分析HTML、分析URL，遍历Web站点，还可以做许多其他事情。使用这些模块的好处是它们的安装是非常值得的。Libwww-perl模块包位于本书所附的光盘上。

LWP::Simple模块一旦安装，你就可以像下面这样检索 Web页：

```
use LWP::Simple qw(get);
$content=get("http://www.slashdot.org");
```

现在$content包含了该URL上的Web页的文本。这不是非常容易吗？

程序清单24-1到24-3展示了检索Slashdot的标题并显示这些标题时使用的程序。

程序清单24-1 Slashdot的标题程序的第一部分

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

```
2:
3:   use strict;
4:   use Fcntl qw(:flock);
5:   use LWP::Simple qw(get);
6:   use CGI qw(:all);
7:
8:   my $url="http://slashdot.org/slashdot.xml";
9:   my $cache="/tmp/slashcache";
10:  my $lockfile="/tmp/slashlock";
11: sub get_lock {
12:    open(SEM, ">$lockfile")
13:        || die "Cannot create lockfilee: $!";
14:    flock(SEM, LOCK_EX) || die "Lock failed: $!";
15: }
16: sub release_lock {
17:    close(SEM);
18: }
```

程序清单24-2 Slashdot的标题程序的第二部分

```
19:
20:  print header;
21:  # If the cache is older than about an hour, rebuild it
22:  get_lock();
23:  if ( (not -e $cache ) or ( (-M $cache) > .04)) {
24:      my $doc=get($url);
25:      if (defined $doc) {
26:              open(CF, ">$cache") || die "Writing to cache: $!";
27:              print CF $doc;
28:              close(CF);
29:      }
30:  }
31:  release_lock();
32:
```

程序清单24-3 Slashdot的标题程序的第三部分

```
33:  print "<H2>Slashdot.Org's Headlines as of ",
34:      scalar(gmtime((stat $cache)[9])),
35:      "GMT </H2>Updated Hourly!<P>";
36:
37:  open(CF, $cache) || die "Cannot open the cache: $!";
38:  my($title, $link);
39:  while(<CF>) {
40:      if (m,<title>(.*)</title>,) {
41:              $title=$1;
42:      }
43:      if (m,<url>(.*)</url>,) {
44:              $link=$1;
45:              print qq{<A HREF="$link">$title</A><BR>\n};
46:
47:      }
48:  }
49:  print "Copyright Slashdot.Org, used with permission.";
50:  close(CF);
```

第3~6行：为了编写本程序，需要许多不同的模块。应该使用模块 Fcntl，因为你必须锁定该程序的一部分，这样每次就只能由一个用户来运行该程序。必须使用 LWP::Simple模块（尤

其是get函数），以便从Slashdot的Web站点检索新闻标题。当然，你也需要 CGI模块，因为这是个CGI程序。

第8行：它包含文件的URL，该文件只包含新闻标题。文件的格式类似下面的形式：

```
<story>
    <title>Ask Slashdot: Internet Voting?</title>
    <url>http://slashdot.org/askslashdot/99/09/05/1732249.shtml</url>
    <time>1999-09-05 21:34:36</time>
    <author>Cliff</author>
     :
```

这里的每条新闻都用＜ story＞标记括了起来。该文件采用称为 XML标记语言的某种简化形式。这使得Perl程序能够很容易地处理该文件，你在下面可以看到这一点。

第9行的变量$cache包含了你临时存放 Slashdot标题所用文件的名字。使用该文件后，每当程序被调用时，你就不必访问 Slashdot的服务器来检索信息，因为你拥有一个本地拷贝。

当然，现在你应该熟悉get_lock()和release_lock()这两个子例程，因为你已经在另外 3个学时中看到过这些函数。之所以需要使用这些函数，原因是 $cache中的文件不应该每次都被多个程序更新，因此它必须锁定。

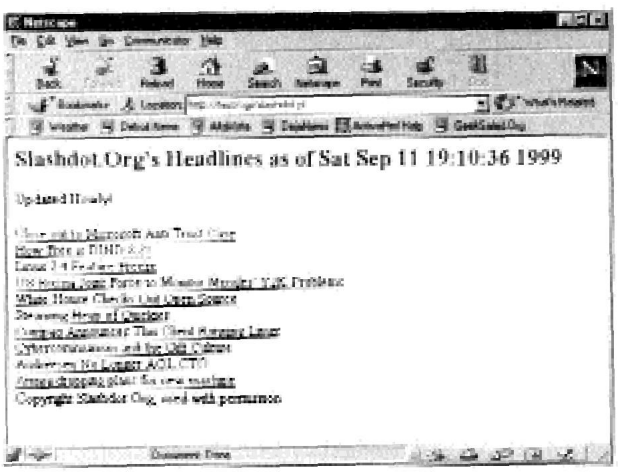
第23行：如果缓存文件不存在，请找出该文件，如果缓存文件已经存在 60分钟以上，就应该重建该文件。 Perl中的-M函数返回Perl程序启动以来该文件的修改时间，但返回的时间采用小数表示的格式。因此，如果该文件已经存在了一天， -M函数返回1，如果文件存在了6个小时， -M返回0.25 (四分之一天)，如果文件存在了1小时， -M返回0.0416666(约1／24天)。

第24行：用于检索包含标题的URL，如前面介绍的LWP：：Simple模块的get函数。下面几行用于将被检索的$doc中的文档写入缓存文件。如果 get方法运行失败，它返回undef，并且在第25行中对此进行检查。

请注意， get_lock()和release_lock()函数序列位于if语句的外面。这一点非常重要。如果CGI程序的一个实例正忙于更新缓存文件，你就不需要另一个实例来查看缓存文件是否存在，或者上次它是何时被修改的。

该程序的最后一部分最简单明了。第 33~35行用于输出简介和上次更新缓存文件的时间。第34行有点儿复杂，我们对此作一说明。首先， stat用于获取关于$cache中文件的信息，并将它作为一个列表返回。第二，列表的第 9个元素（上次修改时间）被取出。第三，用该时间用

图24-2 Slashdot.cgi程序的输出



localtime，在标量上下文中，它返回格式很好的时间字符串。

第40和43行用于从Slashdot中取出标题文件的＜title＞和<url>节。与标题和URL相匹配的部分由正则表达式保存在$1中，然后分别赋予$title和$link。由于<url>元素总是出现在<title>元素之后，因此，当<url>元素被看到时，$title和$link均可在第45行上输出。

在一般情况下，这些正则表达式不应该用来与 HTML相匹配。它们在这里起作用，原因是Slashdot的XML标题文件格式化很好，每一行只有一个 XML元素。如果文件格式要变更，该程序无法进行处理，你应该查看 Slashdot的FAQ，以了解什么发生了变化。

当程序最后运行起来时，其输出将类似图 24-2显示的形式。

当然，你应该运用你的HTML技巧使这个输出更加漂亮一些。

24.2 调查窗体

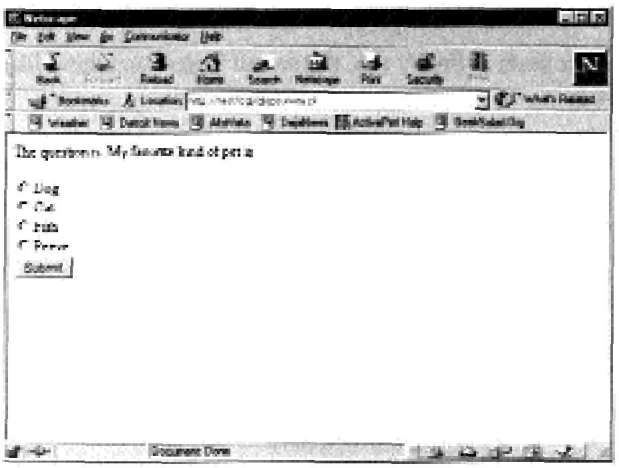
人人都希望成为某个重要人物。人人都希望他的观点能够引起人们的重视，甚至成为一个非常重要的观点，而且每个人都希望知道他的观点与别人的观点相比较的结果。这就是调查要达到的目的。

在下面这个练习中，我们将要介绍一个小程序，用来创建一个调查窗体，然后再创建一个用于输出调查结果的程序。调查程序是个文本文件，它包含一个问题，后面是一些选项，该文件被放入Web服务器上的一个目录下，其名字带有扩展名 .txt。该文本文件类似下面的形式，但是没有其他标点符号或空行：

```
My favorite kind of pet is:
Dog
Cat
Fish
Peeve
```

第一个程序用于查看该目录，找出带有扩展名 .txt的文件（如果存在多个文件，则取其最后一个文件），然后将问题作为窗体的一部分来显示，如图 24-3所示。

图24-3 调查窗体



使用纯文本文件的优点是：CGI程序可以运用文本文件来显示问题，并在以后显示问题的答复。如果你想增加新的调查文件，可以将另一个 .txt文件添加到该目录中，CGI程序能够自动开始使用该文件。它不需要进行很多的维护。

当用户选定一个选项且提交该窗体时，第二个 CGI程序便取出问题的答复，并将它写入与问题在同一个目录中的一个文件中。如果问题文件称为 foo.txt，那么答复将存放在 foo.answer 文件中。当程序写完答复后，它将重新读取所有的答复，并且显示调查结果。

24.2.1 调查窗体程序的第一部分：提出问题

在这个调查中提出问题的程序是非常简单明了的。比较复杂的部分是遍历存放调查信息的目录，找出目录中最后一个扩展名为 .txt的文件。实际上，由于提出问题和写出调查结果的这两个程序都需要查找该文件，因此可以将这部分程序编写成可以重复使用的函数，这样，你就可以在两个地方使用它了。程序清单24-4显示了调查窗体程序的第一部分。

程序清单24-4 显示调查窗体的程序的第一部分

```
1:  #!/usr/bin/perl -w
2:
3:  use strict;
4:  use CGI qw(:all);
5:  my($survey_dir);
6:  $survey_dir="/web/htdocs/poll";
7:
8:  sub find_last_file {
9:          my($type)=@_;
10:         my(@files, $last_file);
11:         # Open the directory, get the last file
12:         # of the correct type.
13:         opendir(SD, $survey_dir) || die "Can't open $survey_dir: $!";
14:         @files=reverse sort grep(/\.$type$/, readdir SD);
15:         closedir(SD);
16:         $last_file=$files[$#files];
17:         return($last_file);
18: }
19:
20: sub get_file_contents {
21:         my($type)=@_;
22:         my(@answers, $last_file);
23:
24:         $last_file=find_last_file($type);
25:
26:         return if (not defined $last_file);
27:         # Open that file, get the contents and return it.
28:         open(QF, "$survey_dir/$last_file")
29:                 || die "Can't open $last_file: $!";
30:         @answers=<QF>;
31:         close(QF);
32:         chomp @answers;   # Remove the newlines
33:         return(@answers);
34: }
```

在第6行上，$survey_dir包含了调查文件所在的目录。若要创建一个新调查文件，只要将一个扩展名为.txt的文本文件放入该目录即可，如本节开头介绍的那样进行操作。该目录必须是可供Web服务器的进程写入的目录。能够写入的目录意味着至少拥有 755（在UNIX系统中）访问权，或者拥有声明的客户写入权限（在 Windows下）。

函数find_last_file()根据扩展名.txt或.answer，在$survey_dir中按字母顺序寻找带有该扩展名的最后一个文件。这个通用型函数在以后供 get_file_contents()函数使用，并且用于下一节

中的调查写入程序。如果目录中不存在该类型的任何文件，那么 find_last_file()返回undef。

函数get_file_contents()再次将扩展名.txt或.answer作为参数，返回该目录中最后一个文件的内容。为了找到该文件名，它使用 find_last_file()函数。

程序清单24-5中显示的该程序的剩余部分是很短的。

程序清单24-5 显示调查窗体程序的第二部分

```
35:
36:  # Get the contents of the last text file, Q & A
37:  my($question, @answers)=get_file_contents("txt");
38:
39:  print header;
40:  print qq{<FORM ACTION="/cgi/writesurvey.cgi" METHOD=POST>\n};
41:  print "The question is: $question<P>\n";
42:  my $answer=0;
43:  foreach(@answers) {
44:      print "<INPUT TYPE=RADIO NAME=answer value=$answer>";
45:      print "$_<BR>\n";
46:      $answer++;
47:  }
48:  print qq{<INPUT TYPE=SUBMIT VALUE="Submit">};
49:  print qq{</FORM>};
```

当这个代码从第36行上开始运行时，函数 get_file_contents()将来自最后一个 .txt文件的内容的第一行加载到$question中，将文件的其余部分加载到 @answers中。

必须将第40行中的/cgi/writesurvey.cgi改为用于CGI调查程序第二部分的任何一个名字。

从那里开始，输出标题，该窗体的开始部分被发送到浏览器。 @answers中的每一行输出时旁边都有一个单选按钮。第一个答复／单选按钮的值为 0，第二个的值为1，以此类推，直到@answers中不再遗留任何答复为止。窗体的正文如下所示：

```
<INPUT TYPE=RADIO NAME=answer value=0>Dog<BR>
<INPUT TYPE=RADIO NAME=answer value=1>Cat<BR>
<INPUT TYPE=RADIO NAME=answer value=2>Fish<BR>
<INPUT TYPE=RADIO NAME=answer value=3>Peeve<BR>
```

当该窗体被提交时， answer的参数被传递给负责写出答复的 CGI程序，该程序称为第40行中的/cgi/writesurvey.cgi，但是你可以修改这个名字。该程序在下一节中介绍。

24.2.2 调查窗体程序的第二部分：计算调查结果

当用户点击调查窗体上的 Submit按钮后，实际程序就开始运行了。用户的选择被记录到一个文件中，调查结果必须制成表格，然后显示出来。

下面这个程序清单看上去很长，但是它的主体部分是你在以前已经见过的子例程。你在本书中的许多地方见过的文件锁定子例程 get_lock()和release_lock()，以及显示调查窗体程序中的get_file_contents()和find_last_file()子例程，构成了这个CGI程序的主体。

程序清单24-6中的代码是该程序的开始部分。请记住，由于你在以前已经看到过该程序的大部分代码，因此不应该对它的长度感到害怕。

程序清单24-6 接收调查结果的程序的第一部分

```
1:   #!/usr/bin/perl -w
2:
```

```
3:   use strict;
4:   use Fcntl qw(:flock);
5:   use CGI qw(:all);
6:
7:   my($survey_dir, $lockfile);
8:   $survey_dir="/web/htdocs/poll";
9:   $lockfile="/tmp/surveylock";
10:
11:  sub find_last_file {
12:      my($type)=@_;
13:      my(@files, $last_file);
14:          # Open the directory, get the last file
15:          # of the correct type.
16:          opendir(SD, $survey_dir) || die "Can't open $survey_dir: $!";
17:          @files=reverse sort grep(/\.$type$/, readdir SD);
18:          closedir(SD);
19:          $last_file=$files[$#files];
20:      return($last_file);
21:  }
22:
23:  sub get_file_contents {
24:          my($type)=@_;
25:          my(@answers, $last_file);
26:
27:      $last_file=find_last_file($type);
28:
29:      return if (not defined $last_file);
30:          # Open that file, get the contents and return it.
31:          open(QF, "$survey_dir/$last_file")
32:                || die "Can't open $last_file: $!";
33:          @answers=<QF>;
34:          close(QF);
35:          chomp @answers;    # Remove the newlines
36:          return(@answers);
37:  }
38: sub get_lock {
39:     open(SEM, ">$lockfile")
40:         || die "Cannot create lockfile: $!";
41:     flock(SEM, LOCK_EX) || die "Lock failed: $!";
42: }
43: sub release_lock {
44:     close(SEM);
45: }
```

到现在为止，程序清单 24-6中的所有代码对你来说应该是熟悉的。这里定义的子例程既可以来自上一个程序（比如 get_file_contents()和find_last_file()），也可能是 get_lock()和relase_lock()例程。同样，必须确保存放在 $survey_dir中的目录能够被Web服务器写入。

由于这部分代码都很简单明了，不必多加说明，因此我们可以转入程序清单 24-7的介绍。

程序清单24-7 接收调查结果的程序的第二部分

```
46:  my($question, @poss_answers)=get_file_contents("txt");
47:
48:  print header;
49:
50:  # Add their answer to the answer file
51:  if (defined param("answer")) {
52:      my($lastfile);
53:      get_lock();
```

```
54:      # Find the last survey file name, and create
55:      # the answer filename from that.
56:      $lastfile=find_last_file("txt");
57:      $lastfile=~s/txt/answer/;
58:
59:      open(ANS, ">>$survey_dir/$lastfile")
60:              || die "Can't write to $lastfile: $!";
61:      print ANS param("answer"), "\n";
62:      close(ANS);
63:      release_lock();
64:  }
65:
66:  my(@answers)=get_file_contents("answer");
67:  my(%results);
68:  # This accumulates how many times each
69:  # answer was given in a hash.
70:  foreach(@answers) {
71:      $results{$_}++;
72:  }
73:  my $ansno=0;
74:  foreach my $ans (@poss_answers) {
75:      $results{$ansno}=0 if (! exists $results{$ansno});
76:      print "$ans was selected $results{$ansno} times<BR>";
77:      $ansno++;
78:  }
```

这部分程序紧接着上一部分程序的结尾。当前提出的问题和对问题的答复分别存放在第46行中的$question和@poss_answers中。

从第50行起，本程序要查看用户是否提供了对调查问题的答复。请记住，如果用户愿意的话，他可以只点击 Submit按钮，而不提供答复。如果提供了答复，便在第 53行上用get_lock()函数取出一个锁，以防止多人同时更新调查结果。

在第56行上，找到了最后一个 .txt调查文件，比如说 first.txt，同时，可以用替换方式将.txt改为.answer，以便给出first.answer。答复文件被打开，当前的答复被附加给该文件，同时，relase_lock()函数对文件进行解锁，因为现在其他人可以安全地打开该文件了。

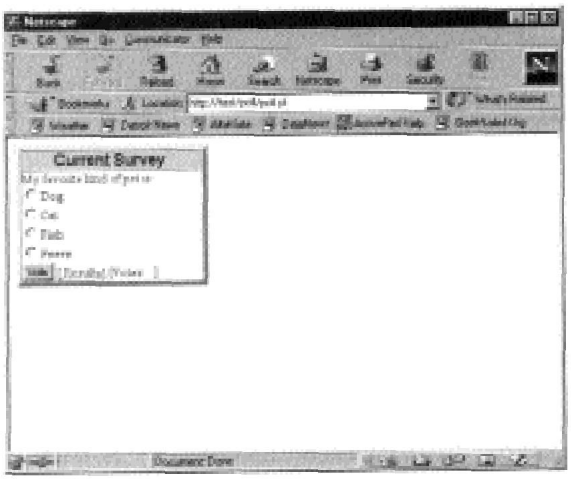
第66行：get_file_contents()函数用于获取调查结果，而不是调查的问题。这时，名叫%results的哈希结构得以创建，其关键字是问题的答复，即数字 0、1、2等，它的值是看到每个关键字的次数。

第74行起输出每个可能的答复。如果 %results中没有任何项目与某个答复相对应，那么调查结果必须是0。答复与给出的次数将在第 76行上输出。

图24-4显示了产生的调查窗体。如果你想使这个调查窗体看上去更好些，可以让 CGI程序显示对调查问题的答复（用带颜色的表格显示调查结果）和其他所有特性，使 HTML页看起来更加美观。

用于保存调查程序的目录（上例中的/web/htdocs/poll）必须是可供全世界的人都能写入的目录，以便使该程序能够运行。在Windows NT下，可以设置该目录的属性，使它可以被Guest（客方）写入。在UNIX下，可以使用chmod命令将访问权限设置为777。另外，如果用人工运行调查结果的接收程序，而不使用浏览器，那么它会创建一个Web服务器无法写入的.answer文件。如果是这样的话，你必须删除该文件，然后调查程序才能正确运行。

图24-4 经过格式美化的调查窗体



24.3 课时小结

在本学时中，创建了两个程序，使你的 Web页变得更加丰富多彩。首先，创建了一个从其他站点检索内容并将内容显示在你自己的 Web页上的程序。我们还讲述了向其他人借用 Web内容时应该注意的一些问题。接着又创建了一个调查程序，使 Web站点的访问者能够参与站点正在进行的活动。

24.4 课外作业

24.4.1 专家答疑

问题：在Web服务器上拥有一个可供所有人写入的目录（用于民意测试），是否会带来安全上的漏洞？

解答：是的，不过这个漏洞并不大。如果你的服务器是以一种合理的方式安装的，那么就无人能够将内容上载到你的站点。但是，如果你的服务器允许任何人将任何东西上载到任何地方，那么那里就可能存在滥用危险。如果你愿意的话，可以在创建 .txt文件的同时，创建.answer文件，以便解决这个问题。请记住使用 chmod使.answer文件成为所有人都能写入的文件。

问题：如果我仅仅从某个站点借用了一些新闻标题，我会不会受到指控？

解答：是的，你会受到指控，以前曾经发生过这样的的事情。 1999年2月， Microsoft与Ticketmaster两公司之间就因为这样的问题而对簿公堂。据说 Microsoft公司使用“ 深层次链接 ”进入了Ticketmaster公司的Web站点，为此Ticketmaster提出了诉讼。在这个案件中，没有出现侵犯版权的问题，但是有足够的证据说明 Microsoft公司进入了“ 深层次链接 ”。如果涉及到侵犯版权的问题，问题就严重了。

问题：有一个站点，我想从中借用其新闻标题。它类似 Slashdot的站点。但是该站点不具备很好的XML或文件供分析，因此我必须改用普通的 HTML文件。我应该如何分析该文件？

解答：如果你要分析 HTML文件，请不要使用正则表达式，也不要自己对它进行分析。对HTML文件进行分析并不像它看起来那样容易，并且几乎无法得到正确的结果。另外，即

使你试图用正则表达式来分析某些 HTML文件，它也不是到处都能取得成功的。 CPAN包含用于分析HTML文件的一些模块，这些模块都在CPAN的HTML节下，即HTML：：*下。

24.4.2 思考题

1) 若要从Web服务器检索一个HTML文件，我应该怎么做？
   a.使用LWP。
   b.打开到达该系统的套接字，然后检索数据。
   c.使用‘ lynx -dump ’或‘ netscape -print ’。
2) 如果LWP::Simple模块的get函数运行失败，它返回什么？
   a.出错消息，即“ No Documcnt（无文档）”
   b.空字符串，即“ ”
   c.undef

24.4.3 解答

1) 答案是a。虽然b和c也可以，但是这两种操作方法很不可靠，使用起来也很难。
2) 答案是c。这个答案在程序清单24-2后面的程序分析中作了解释。

24.4.4 实习

- 即使不使用图形模块，你也能很容易创建图 24-4中表示民意测验结果的条形图。为了创建这个条形图，你需要一个颜色正确的 1×1（或非常小的）.gif文件。若要制作条形图，你只需显示带有相应高度和宽度标号的 .gif文件，如下所示：

```
<IMG SRC="small.gif" HEIGHT=20 WIDTH=200 alt="bar graph">
```

大多数图形浏览器都能将这个小型 .gif文件放大为规定的大小。

你的任务是：使该民意测验程序输出带有条形图的结果。你必须计算各种类别投票的总数，再将这个总数除以每种类别投票的数量，确定条形图的正确宽度。例如，投票总数是 100，一个类别的投票数目为40，那么可以将条形图的最大宽度乘以 .4。

- 程序清单24-4中的调查程序可能因为有人多次投票而使调查结果产生偏差。你能设法避免这种情况吗？你可以保存一个文件，里面是所有被调查人地址的列表，不允出现重复。这可以防止有人在缓存代理的后面进行投票（第 23学时介绍了缓存代理）。请设计一种方法，只允许访问该站点的每个人投票一次。（正如你所知道的那样，这种方法不可能做到非常简单明了，它只是一种思路验证方法。）