第 5 章　代码重用与函数编写

本章将介绍如何通过代码重用更加轻松地编写一致性、可靠性和可维护性更高的代码。

我们将介绍模块化和代码重用的技巧，首先将介绍如何使用函数`require()`和`include()`在多个页面中实现代码重用。我们还将解释为什么这样做会优于在服务器端包含的做法。

本章所给出的例子涵盖了如何使用包含文件为网站创建统一风格的页面。我们将通过页面和表单生成函数来解释如何编写和调用自己的函数。

在本章中，我们将主要介绍以下内容：

- 代码重用的好处
- 使用`require()`和`include()`函数
- 函数介绍
- 定义函数
- 使用参数
- 理解作用域
- 返回值
- 参数的引用调用和值调用
- 实现递归
- 使用命名空间

5.1　代码重用的好处

软件工程师的一个目标就是通过重复使用代码来避免编写新的代码。这样做并不是因为他们懒，而是因为重新使用已有的代码可以降低成本、增加代码的可靠性并提高它们的一致性。在理想情况下，一个新的项目是这样创建的：它将已有的可重新利用的组件进行组合，并将新的开发难度降低到最小。

5.1.1　成本

在一个软件的有效生命周期中，相当多的时间是用在维护、修改、测试和文档化记录上，而不是最初花在编码上的时间。如果要编写商业代码，应该尽量限制结构中所用到的代码行数。一个最常使用的方法就是：重新使用已有的代码，而不是为一个新任务编写一个和原来代码只有微小区别的新代码。更少的代码意味着更低的成本。如果市场上已经存在能够满足需求的软件，那就购买软件。购买已有软件的成本总是要小于开发一个等价产品的成本。如果有现成的软件基本上能够满足要求，那就必须小心地使用它。修改已有的代码可能会比编写新代码更加困难。

5.1.2 可靠性

如果一个模块代码已经在代码结构中使用了，可以认为它是已经通过测试的。即使代码只有几行，在重写时仍然可能忽略两方面的内容，一是原作者融入其中的某些东西，二是代码测试发现缺陷后，对原来代码添加的一些东西。使用现存的成熟的代码通常要比新鲜的“绿色”代码更可靠。

5.1.3 一致性

系统的外部接口应该是一致的，其中包括用户接口和系统的外部接口。编写一段新的并且能够和系统函数的其他部分保持一致的代码需要花些心思和努力。如果重复使用运行在系统其他部分的代码，所实现的功能自然就会达到一致。

除了这些优点外，只要原来的代码是模块化的而且编写良好，那么重复使用代码还会节省许多工作。在工作时，可以试着辨认一下今后可能再次要调用的代码段。

5.2 使用require()和include()函数

PHP提供了两个非常简单却很有用的语句，它们允许重新使用任何类型的代码。使用一条require()或include()语句，可以将一个文件载入到PHP脚本中。通常，这个文件可以包含任何希望在一个脚本中输入的内容，其中包括PHP语句、文本、HTML标记、PHP函数或PHP类。

这些语句的工作方式类似于大多数Web服务器提供的服务器端包含方式以及C语言或C++中的#include语句。

require()和include()几乎是相同的。二者唯一的区别在于函数失败后，require()函数将给出一个致命的错误。而include()只是给出一个警告。

require()和include()也有两个变体函数，分别是require_once()和include_once()。正如你可能猜到的，这两个函数的作用是确保一个包含（included）的文件只能被引入一次。对于我们已经介绍过的例子——页眉和脚注（header and footer）——这个功能并不是非常有用。

当使用require()和include()来引入函数库时，它们才非常有用。使用这两个函数可以防止错误的引入同样的函数库两次，从而出现重复定义的错误。如果关心编码实践，可以考虑使用require()和include()，因为它们的运行速度较快。

5.2.1 文件扩展名和require()函数

如下所示的代码保存于reusable.php文件中：

```
<?php
  echo 'Here is a very simple PHP statement.<br />';
?>
```

如下所示的代码保存于main.php文件中：

```
<?php
  echo 'This is the main file.<br />';
  require( 'reusable.php' );
  echo 'The script will end now.<br />';
?>
```

如果载入reusable.php，当浏览器中显示出“Here is a very simple PHP statement”时，你不会感到奇怪。如果载入main.php，则会发生一件更有趣的事情。该脚本输出结果如图5-1所示。

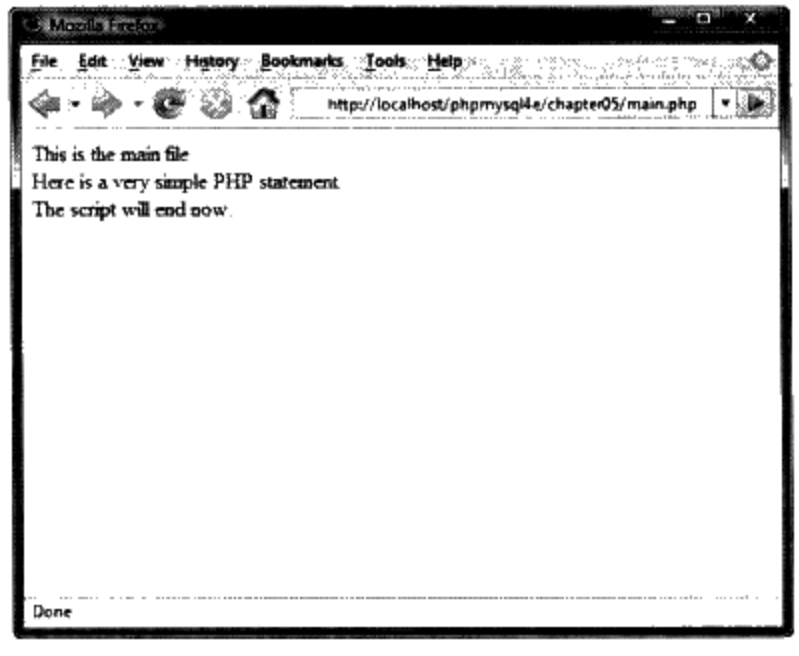

图5-1　main.php文件的输出显示require()语句的结果

当需要一个文件的时候，可以使用require()语句。在前面的例子中，我们使用的文件是reusable.php。当运行该脚本时，require()语句：

```
require( 'reusable.php' );
```

将被请求的文件内容代替，然后再执行脚本。这就意味着，当载入main.php文件时，它会像如下所示的代码那样执行：

```
<?php
  echo 'This is the main file.<br />';
  echo "Here is a very simple PHP statement.<br />";
  echo "The script will end now.<br />";
?>
```

当使用require()语句时，必须注意处理文件扩展名和PHP标记的不同方式。

PHP并不会查看所需文件的扩展名。这就意味着，只要不想直接调用这个文件，就可以任意命名该文件。当使用require()语句载入文件时，它会作为PHP文件的一部分被执行。

通常，如果PHP语句放在一个HTML文件（例如，名为page.html的文件）中时，它们是不会被处理的。PHP通常用来解析扩展名被定义成如.php的文件。（在Web服务器配置文件中可能不是这样）但是，如果通过require()语句载入这个page.html，文件内的任何PHP命令都会被处理。因此，可以使用任何扩展名来命名包含文件，但要尽量遵循一个约定，例如将扩展名命名为.inc或.php是一个很好的办法。

需要注意的一个问题是，如果扩展名为.inc或一些其他的非标准扩展名的文件保存在Web文档树中，而且用户可以在浏览器中直接载入它们，用户将可以以普通文本的形式查看源代码，包括任何密码。因此，将被包含文件保存在文档树之外，或使用标准的文件扩展名是非常重要的。

提示　在这个例子中，可重用文件（reusable.php）代码如下所示：

```
<?php
echo "Here is a very simple PHP statement.<br />";
?>
```

我们将文件中的PHP代码放到PHP标记之间。如果希望一个所需文件中的PHP代码能够被当成PHP代码进行处理，就必须这样做。如果不使用PHP标记，代码将会被视为文本或者HTML脚本，因此也就不会被执行。

5.2.2 使用require()制作Web站点的模板

如果Web页面具有一致的外观，可以在PHP中使用require()语句将模板和标准元素加入到页面中。

例如，一个虚构的TLA咨询公司的网站有许多页面，这些页面的外观看上去都如图5-2所示。当需要一个新页面的时候，开发人员可以打开一个已有页面，从文件中间剪切所需的文本，输入所需的新文本，然后以新的文件名保存。

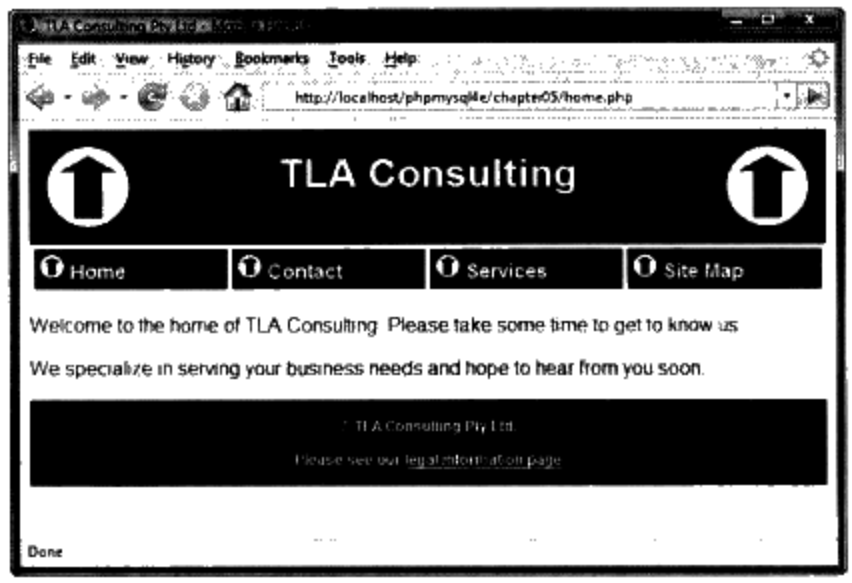

图5-2 TLA咨询公司网页有着统一的外观和风格

考虑这种情况：网站已经存在了一段时间，如今已有数十个、数百个甚至数千个页面都是同一种风格。现在，要对标准外观进行部分修改——这种修改可能是很微小的改变，例如，在每个脚注上加一个电子邮件地址，或者加上一个新的导航菜单入口。你希望对这数十个、数百个甚至数千个页面都做这种微小的修改吗？

相对于剪切粘贴数十个、数百个甚至数千个页面，直接重用各个页面中通用的HTML代码部分是一个更好的办法。程序清单5-1给出了图5-2所示页面（home.html）的源代码。

程序清单5-1 home.html——TLA咨询公司主页的HTML脚本

```
<html>
<head>
  <title>TLA Consulting Pty Ltd</title>
  <style type='text/css'>
    h1 {color:white; font-size:24pt; text-align:center;
        font-family:arial,sans-serif}
    .menu {color:white; font-size:12pt; text-align:center;
           font-family:arial,sans-serif; font-weight:bold}
    td {background:black}
    p {color:black; font-size:12pt; text-align:justify;
       font-family:arial,sans-serif}
    p.foot {color:white; font-size:9pt; text-align:center;
            font-family:arial,sans-serif; font-weight:bold}
    a:link,a:visited,a:active {color:white}
  </style>
</head>
<body>
```

```
  <!-- page header -->
  <table width="100%"cellpadding="12"cellspacing="0"border="0">
  <tr bgcolor="black">
    <td align="left"><img src="logo.gif"alt="TLA logo"height="70"
width="70"></td>
    <td>
        <h1>TLA Consulting</h1>
    </td>
    <td align="right"><img src="logo.gif"alt="TLA logo"height="70"
width="70"></td>
  </tr>
  </table>

  <!-- menu -->
  <table width="100%"bgcolor="white"cellpadding="4"cellspacing="4">
  <tr >
    <td width="25%">
      <img src="s-logo.gif"alt=""height="20"width="20">
      <span class="menu">Home</span></td>
    <td width="25%">
      <img src="s-logo.gif"alt=""height="20"width="20">
      <span class="menu">Contact</span></td>
    <td width="25%">
      <img src="s-logo.gif"alt=""height="20"width="20">
      <span class="menu">Services</span></td>
    <td width="25%">
      <img src="s-logo.gif"alt=""height="20"width="20">
      <span class="menu">Site Map</span></td>
  </tr>
  </table>
  <!-- page content -->
  <p>Welcome to the home of TLA Consulting.
  Please take some time to get to know us.</p>
  <p>We specialize in serving your business needs
  and hope to hear from you soon.</p>

  <!-- page footer -->
  <table width="100%"bgcolor="black"cellpadding="12"border="0">
  <tr>
    <td>
      <p class="foot">&copy; TLA Consulting Pty Ltd.</p>
      <p class="foot">Please see our
        <a href="legal.php">legal information page</a></p>
    </td>
  </tr>
  </table>
</body>
</html>
```

在程序清单5-1中，可以看到这个文件由许多不同的代码部分组成。HTML标题包含了在该页面中用到的级联风格样式单（CSS）中的样式定义。标有“page header”的部分显示了公司的名称和徽标，标有“menu”的部分创建了页面的导航条，而标有“page content”的部分是页面中的文本。再下面的就是脚注。通常，可以将这个文件分割，然后给这些部分分别命名为header.php、home.php和footer.php。文件header.php和footer.php都包含有在其他页面中可以重用的代码。

文件home.php可以代替home.html，它包含页面内容和两个require()语句，如程序清单5-2所示。

程序清单5-2 home.php——TLA公司主页的PHP脚本

```
<?php
  require( header.php');
?>
  <!   page content -->
  <p>Welcome to the home of TLA Consulting.
  Please take some time to get to know us.</p>
  <p>We specialize in serving your business needs
  and hope to hear from you soon.</p>
<?php
  require( footer.php');
```

home.php中的require()语句将载入header.php和footer.php。

正如前面所提到的，在通过require()调用它们的时候，文件的名称并不会影响对它们的处理。一个常见的约定就是调用那些包含在其他文件something.inc（此处inc代表include）中的部分文件代码，这些文件代码若不被调用，将会停止执行。但是这却不是推荐的基本策略，因为如果Web服务器没有专门设置，.inc文件将不会被解释成PHP代码。

如果打算这样做，可以将.inc文件保存在一个目录中，而这个目录可以被脚本访问，但是被引入的文件不会被Web服务器载入——也就是，放在Web文档树之外。这种设置是非常不错的，它可以防止下面两种情况的发生：a）如果文件扩展名是.php，但只包含部分页面或脚本，此时可能会引起错误。b）如果已经使用了其他扩展名，别人就可以读取源码。

文件header.php包含了页面使用的级联风格样式单定义以及程序清单5-3所显示的公司名称和导航菜单的表格。

程序清单5-3 header.php——所有TLA网站的页面可重复使用的页眉

```
<html>
<head>
  <title>TLA Consulting Pty Ltd</title>
  <style type='text/css'>
    h1 {color:white; font-size:24pt; text-align:center;
        font-family:arial,sans-serif}
    .menu {color:white; font-size:12pt; text-align:center;
           font-family:arial,sans-serif; font-weight:bold}
```

```
    td {background:black}
    p {color:black; font-size:12pt; text-align:justify;
       font-family:arial,sans-serif}
    p.foot {color:white; font-size:9pt; text-align:center;
            font-family:arial,sans-serif; font-weight:bold}
    a:link,a:visited,a:active {color:white}
  </style>
</head>
<body>

  <!-- page header -->
  <table width="100%" cellpadding="12" cellspacing="0" border="0">
  <tr bgcolor="black">
    <td align="left"><img src="logo.gif" alt="TLA logo" height="70" width="70"></td>
    <td>
       <h1>TLA Consulting</h1>
    </td>
    <td align="right"><img src="logo.gif" alt="TLA logo" height="70" width="70" /></td>
</tr>
</table>

<!-- menu -->
<table width="100%" bgcolor="white" cellpadding="4" cellspacing="4">
<tr>
  <td width="25%">
    <img src="s-logo.gif" alt="" height="20" width="20" />
    <span class="menu">Home</span></td>
  <td width="25%">
    <img src="s-logo.gif" alt="" height="20" width="20" />
    <span class="menu">Contact</span></td>
  <td width="25%">
    <img src="s-logo.gif" alt="" height="20" width="20" />
    <span class="menu">Services</span></td>
  <td width="25%">
    <img src="s-logo.gif" alt="" height="20" width="20" />
    <span class="menu">Site Map</span></td>
</tr>
</table>
```

文件footer.inc包含了在每个页面底部脚注处显示的表格。这个文件如程序清单5-4所示。

程序清单5-4　footer.php——所有TLA网站的页面可重复使用的脚注

```
<!-- page footer -->
  <table width="100%" bgcolor="black" cellpadding="12" border="0">
  <tr>
    <td>
      <p class="foot">&copy; TLA Consulting Pty Ltd.</p>
```

```
        <p class= "foot">Please see our <a href= "legal.php">
        legal information page</a></p>
      </td>
    </tr>
    </table>
  </body>
  </html>
```

这种方法很容易就使网站具有统一的风格，而且还可以通过输入如下所示的代码创建一个新的统一风格页面：

```
<?php require('header.php'); ?>
Here is the content for this page
<?php require( footer.php ); ?>
```

最重要的是，即使用这个页眉和脚注创建许多新页面后，也很容易修改页眉和脚注文件。

无论是做一个无关紧要的修改还是重新设计网站的外观，只需要进行一次修改。我们并不需要单独地对网站中的每个页面进行修改，因为它们都是载入页眉和脚注文件的。

这里，这个例子在页面正文、页眉和脚注处只使用了纯HTML，但这不是问题。有了这些文件，我们也可以用PHP命令动态地生成页面的某些部分。

如果希望保证一个文件将被当作纯文本或HTML，而且不会执行任何PHP，可以使用readfile()作为替代方法。这个函数将回显文件内容，不会对其进行解析。如果使用的是用户提供的文本，这可能就是一个重要的安全问题。

5.2.3 使用auto_prepend_file和auto_append_file

如果希望使用require()将页眉和脚注加入到每个页面中，还有另外一种办法。在配置文件php.ini中有两个选项auto_prepend_file和auto_append_file。通过这两个选项来设置页眉和脚注，可以保证它们在每个页面的前后被载入。使用这些指令包含的文件可以像使用include()语句包含的文件一样；也就是，如果该文件不存在，将产生一个警告。

对于Windows，其设置如下所示：

```
auto_prepend_file = "c:/Program Files/Apache Software
Froundation/Apache2.2//include/header.php"
auto_append_file = "c:/Program Files/Apache Group/Apache2/include/footer.php"
```

对于UNIX，其设置如下所示：

```
auto_prepend_file = "/home/username/include/header.php"
auto_append_file = "/home/username/include/footer.php"
```

如果使用了这些指令，就不需要再输入include()语句，但页眉和脚注在页面中不再是页面的可选内容。

如果使用的是Apache Web服务器，可以对单个目录进行不同配置选项的修改。这样做的前提是服务器允许重设其主配置文件。要给目录设定自动前加入和自动追加，需要在该目录中创建一个名为.htaccess的文件。这个文件需要包含如下两行代码：

```
php_value auto_prepend_file "/home/username/include/header.php"
php_value auto_append_file "/home/username/include/footer.php"
```

请注意，其语法与配置文件php.ini中的相应选项有所不同，和行开始处的php_value一样：没有等号。许多php.ini中的配置设定也可以按这种方法进行修改。

在.htaccess中设置选项，而不是php.ini中或是在Web服务器的配置文件中进行设置，将带来极大的灵活性。可以在一台只影响你的目录的共享机器上进行。不需要重新启动服务器而且不需要管理员权限。使用.htaccess方法的一个缺点就是目录中每个被读取和被解析的文件每次都要进行处理，而不是只在启动时处理一次，所以性能会有所降低。

5.3 在PHP中使用函数

函数存在于大多数的编程语言中。它们用于分隔那些能够完成独立而又明确的任务的代码。这使得代码更易于阅读，并且允许在每次需要完成同样任务的时候重复使用代码。

函数是一个给出了调用接口的自包含模块，它可以执行一些任务，还可以返回结果（可选的）。

你肯定已经见到过了许多函数。在前面的章节中，我们已经调用了许多PHP内置的函数。

此外，我们还编写了几个简单的函数，但是我们忽略了其中的细节。在这一节中，我们将更详细地介绍如何调用和编写函数。

5.3.1 调用函数

如下所示代码是调用函数最简单的例子：

```
function_name();
```

以上代码将调用一个名为function_name且不需要任何输入参数的函数。这行代码还忽略了任何可能的函数返回值。

许多函数确实就是这样调用的。在测试时，你会发现函数phpinfo()是非常有用的，因为它显示了已安装的PHP的版本、关于PHP的信息、Web服务器的设置和众多的PHP和服务器变量的值。这个函数不需要任何参数，通常可以忽略它的返回值，所以，可以使用如下所示方式调用函数phpinfo()：

```
phpinfo();
```

然而，大多数函数都需要一个或更多的参数，它们都是函数的输入参数。我们通过将数据或变量名放在函数名称后面的括号内，从而以参数形式传给函数。为函数提供一个参数并对其进行调用如下所示：

```
function_name('parameter');
```

在这个例子中，所使用的参数是一个只包含parameter的字符串，但是，依据函数的不同，如下所示的调用也是可以的：

```
function_name(2);
function_name(7.993);
```

```
function_name($variable);
```

在最后一行中，$variable可以是任何一种PHP变量，包括数组或对象。

参数可以是任何数据类型，但特定的函数通常会要求特定的数据类型。

可以通过函数原型来了解函数所需的参数个数，每一个参数所表示的对象以及每一个参数的数据类型。通常，在本书中，当我们描述一个函数时，会给出一个函数的原型。

fopen()的函数原型如下所示：

```
resource fopen ( string filename, string mode
                 [, bool use_include_path [, resource context]])
```

这个函数原型告诉了我们许多信息，知道如何正确地解释这些说明是非常重要的。在这个例子中，函数名称前面的单词“resource”告诉我们这个函数会返回一个资源（即一个打开的文件句柄）。而函数的参数在括号的里面。在fopen()的例子中，函数原型中给出了4个参数。文件名称、打开模式这两个参数都是字符串，而*use_include_path*是一个布尔值，而参数*context*是一个资源。*use_include_path*外面的方括号指明了这个参数是可选的。可以给可选参数赋值也可以忽略它们，如果忽略它们则会使用默认值。但是，请注意，一个具有多个可选值的函数，必须按照从右到左的顺序使用默认值。例如，当使用fopen()函数，可以不给出*context*参数，或者可以不提供*use_include_path*和*context*参数；但是，不能不提供*use_include_path*参数，而只提供zcontext参数。

在了解这个函数的函数原型后，可以知道下面的fopen()的调用是有效的：

```
$name = 'myfile.txt';
$openmode = 'r';
$fp = fopen($name, $openmode);
```

以上代码调用了fopen()函数，函数的返回值将保存在变量$fp中。对于这个例子来说，我们传递给函数一个名为$name的变量，它包含了要打开文件的名称，还有一个名为$openmode的变量，它包含了一个表示要打开文件的字符串，表示文件的打开模式。我们并没有给出第三个和第四个可选参数。

5.3.2 调用未定义的函数

如果调用一个并不存在的函数，会得到一个如图5-3所示的错误信息。

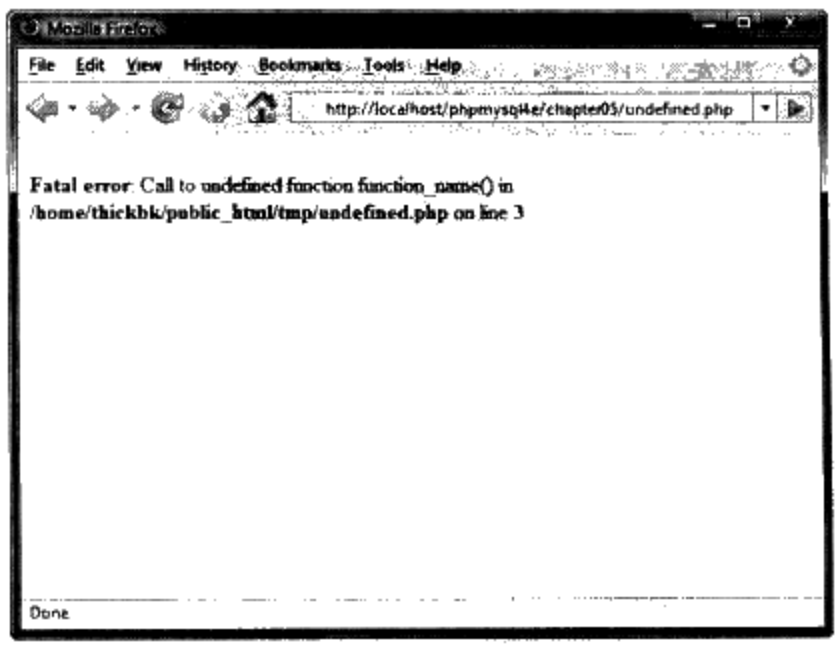

图5-3 调用一个不存在的函数而产生的错误信息

通常，PHP给出的错误信息是非常有用的。它可以告诉我们错误出现在哪个文件中，错误在文件中的哪一行，以及我们试图调用的函数名称。这样就可以很容易地找到并纠正错误。

如果看到这个错误信息，必须对两件事情进行检查：

- 函数名称的拼写是否正确。

■ 这个函数是否存在于所用的PHP版本中。

记住每个函数名称的正确拼写并不是一件容易的事情。例如，有些两个单词组成的函数名称在词与词之间有的有下画线而有的却没有。函数stripslashes()就是将两个单词连在一起，而函数strip_tags()则是用下画线将两个单词分开了。在函数调用中，错误地拼写函数名称将会导致如图5-3所示的错误信息。

本书中使用的许多函数在PHP 4版本中是不存在的，因为本书是假设所使用的PHP版本至少是5。在每个新的版本中，都会加入新的函数，而且如果使用的是一个较老的版本，新加入的功能和性能就将会有所升级。要了解什么时候有新函数加入，可以查看PHP的在线指南。试图调用一个在所使用的版本中还未声明的函数会导致如图5-3所示的错误信息。

看到这个错误消息的另一个原因就是所调用的函数是PHP扩展的一部分，而该部分并没有被载入。例如，如果尝试使用gd库（image manipulation图像操作函数库）的某些函数而没有安装gd，将看到这个错误消息。

5.3.3 理解字母大小写和函数名称

请注意，函数调用将不区分大小写，所以调用function_name()、Function_Name()或FUNCTION_NAME()都是有效的，而且都将返回相同的结果。可以按照便于自己阅读的方式任意使用大小写，但应该尽量保持一致。本书和大多数PHP文档使用的命名惯例是：所有都用小写字母。

注意到函数名称和变量名称是不同的，这一点很重要。变量名是区分大小写的，所以$Name和$name是两个不同的变量，但Name()和name()则是同一个函数。

5.4 理解为什么要定义自己的函数

在前面的章节中，你已经了解了使用PHP的某些内置函数的例子。但是，编程语言的真正功能是通过创建你自己的函数来实现的。

PHP内置函数允许和文件进行交互、使用数据库、创建图形，还可以连接其他服务器。

但是，在实际工作中，有许多时候所需要的东西是语言的创建者无法预见到的。

幸运的是，我们并不只局限于使用内置函数，因为可以编写自己的函数来完成任何所需的任务。我们的代码可能是已有函数和自己逻辑的混合体，通过它来完成我们的任务。

如果你正在为一个任务编写一段代码，而很可能这段代码将在一个脚本的多处或是多个脚本中都要使用，那么最明智的方法是将这段代码声明为函数。

声明一个函数可以让我们像内置函数那样使用自己的代码。只要简单地调用这个函数并提供给它必需的参数。这就意味着，在整个脚本中，都可以调用和多次重复使用相同的函数。

5.5 了解基本的函数结构

一个函数声明将创建或者声明一个新的函数。声明是以关键字function开始的，接下来，给出函数名称和必要的参数，然后再给出每次调用这个函数时要执行的代码。

在这里，给出一个常见的函数声明：

```
function my_function() {
  echo 'My function was called ;
}
```

这个函数声明是以function开始的，这样读者和PHP解释器都将知道这是一个用户定义的函数。该函数名称是my_function。可以使如下所示的命令调用这个新函数：

```
my_function();
```

正如你所猜到的，调用这个函数会在浏览器中显示文本“My function was called”。

内置函数在所有PHP脚本中都可以使用，但是如果声明了自己的函数，它们只是在声明它们的脚本中可以使用。将经常用到的函数包含在一个文件中是一个很好的主意。然后可以在所有脚本中调用require()语句，这样这些函数就可以使用了。

在一个函数中，花括号包括了完成所要求任务的代码。在花括号中，可以包含任何在PHP脚本的其他地方都合法的代码，其中包括函数调用、新的变量或函数声明、require()或include()语句类声明以及HTML脚本。如果希望在一个函数中退出PHP并输入HTML脚本，可以像在脚本其他地方做的那样——使用一个封闭的PHP标记，然后再编写HTML。

下面的代码是前面例子的一个合法的修改，其输出结果是一样的：

```
<?php
  function my_function() {
?>
My function was called
<?php
  }
?>
```

请注意，PHP代码被封闭在一对匹配的PHP开始和结束标记之间。在本书的大多数小段代码示例中，并没有使用这些标记。它们被显示出来是因为在这些例子中有这样的要求。

函数的命名

在给函数命名的时候，最重要的就是函数名称必须精炼但又要有描述性。如果函数是用来创建页眉的，那么pageheader()或page_header()是不错的名称。

函数命名具有如下几个限制：

- 函数名称不能和已有的函数重名。
- 函数名称只能包含字母、数字和下画线。
- 函数名称不能以数字开始。

许多语言允许重复使用函数名称。这个特性叫做函数的重载。但是PHP不支持函数重载，所以自定义函数不能和内置函数或是用户已定义的函数重名。请注意，虽然每个PHP脚本知道所有内置函数，但对于用户定义的函数，PHP只能识别那些存在于本脚本之中的。这就意味着，虽然可以在不同的文件中重复使用一个函数名，但这会引起混乱，所以应该避免。

如下所示的函数名称是合法的：

```
name()
```

```
name2()
name_three()
_namefour()
```

而如下所示的函数名称则是不合法的：

```
5name()
name-six()
fopen()
```

（如果最后一个函数不是因为已经存在了，那它就是合法的。）

请注意，虽然$name并不是一个函数的合法名称，但是一个类似于如下所示的函数调用：

```
$name();
```

也可以正确地执行，这是根据$name的值来确定的。其原因就是PHP可以取出保存在$name中的值，寻找具有那个名称的函数，并且调用该函数。这种函数类型被称为可变函数，而且有时候是有用的。

5.6 使用参数

要使函数正常工作，它们中的大多数都需要一个或多个参数。参数允许将数据传给函数。这里有一个只需要一个参数的函数例子。这个函数带有一个一维数组并将它按表格形式显示出来，如下所示：

```
function create_table($data) {
  echo '<table border=\"1\">';
  reset($data); // Remember this is used to point to the beginning
  $value = current($data);
  while ($value) {
     echo "<tr><td>".$value.'</td></tr>\n";
     $value = next($data);
  }
  echo "</table>";
}
```

如果按如下所示调用create_table()函数：

```
$my_array = array('Line one. ','Line two.','Line three.');
create_table($my_array);
```

将看到如图5-4所示的结果。

传递参数允许我们获得在函数外面生成的数据（在这个例子中，就是数组$data）将被传入函数中。

和内置函数一样，用户定义函数可以有多个参数和可选参数。我们有很多方式来改进create_table()函数，但却只有一个方法可以让调用者指明边界或表格的其他属性。在这里，我们给出了该函数的一个改进版本。它非常类似于改进前的函数，但允许调用者可选地设置表格的边界、宽度、单元大小和空白填充，如下所示：

```
<?php
```

```
function create_table2($data, $border=1, $cellpadding=4, $cellspacing=4 ) {
  echo '<table border=\"'.$border.'\" cellpadding=\"'.$cellpadding."\"
        cellspacing=\"".$cellspacing."\">';
  reset($data);
  $value = current($data);
  while ($value) {
     echo "<tr><td>".$value."</td></tr>\n";
     $value = next($data);
  }
  echo "</table>";
}


$my_array = array( 'Line one.','Line two.','Line three. );
create_table2($my_array, 3, 8, 8);
```

create_table2()函数的第一个参数还是必需的。而后三个参数都是可选的，因为已经在函数中为它们定义了默认值。所以可以调用create_table2()生成类似于图5-4所示的输出结果。

```
create_table2($my_array);
```

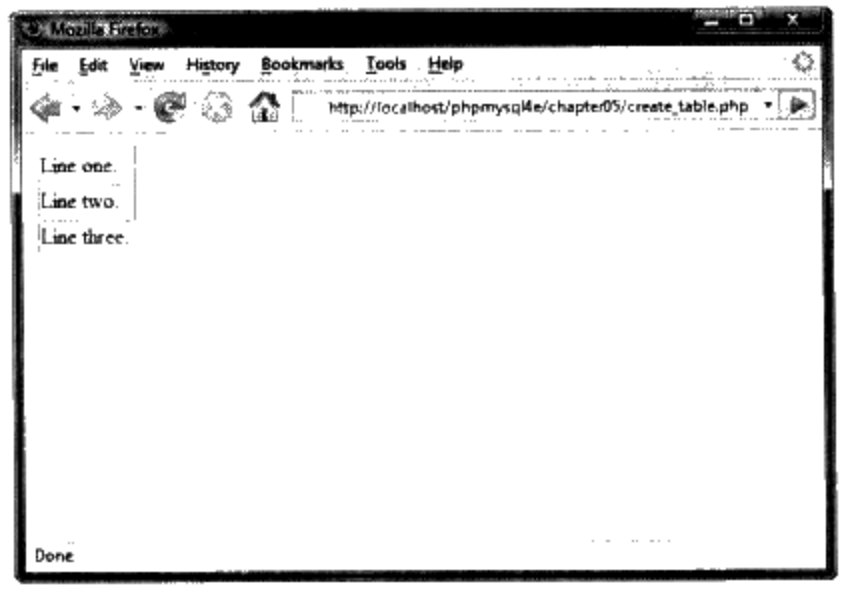

图5-4 调用函数create_table()后出现的HTML表格

如果希望以更分散的风格输出这些数据，可以按如下所示方式调用该函数：

```
create_table2($my_array, 3, 8, 8);
```

可选值不用全部给出；可以给出一部分而忽略一部分。参数将会按照从左到右的顺序进行赋值。

请记住，不能漏掉一个可选参数而给出参数列表中的后一个参数。在这个例子中，如果希望将一个值传给cellspacing，就必须也得传给cellpadding一个值。这是编程过程中的常见错误，也是可选参数在每个参数列表中最后被指定的原因。

如下所示的函数调用：

```
create_table2($my_array, 3);
```

是完全合法的，结果是$border被设为3，$cellpadding和$cellspacing都被设为默认值。

也可以声明能够接收可变参数数量的函数。通过3个帮助器函数：func_num_args()、func_get_arg()以及func_get_args()，可以确定已经传递了多少个参数以及这些参数的值。

例如，分析如下所示的函数：

```
function var_args() {
  echo "Number of parameters: ";
```

```
  echo func_num_args();

  echo '<br />';
  $args = func_get_args();
  foreach ($args as $arg) {
    echo $arg.'<br />';
  }
}
```

这个函数报告传递给它的参数个数并且打印每一个参数。func_num_args()函数将返回传入的参数个数。而func_get_args()函数将返回参数的数组。或者，可以使用func_get_arg()函数一次获得一个参数，该函数需要以希望访问的参数个数作为参数（参数从0开始）。

5.7 理解作用域

你可能已经注意到了，当在包含文件中使用变量的时候，只需要在脚本中的require()和include()语句前声明它们，但是在使用函数的时候，则要明确地将这些变量传递给函数。一方面是因为没有将变量传给所需或包含文件的机制，另一方面是因为变量的作用域相对于函数是不同的。

变量的作用域可以控制变量在哪里是可见并且可用的。不同的编程语言有不同的变量作用域规则。PHP具有相当简单的规则：

- 在函数内部声明的变量作用域是从声明它们的那条语句开始到函数末尾。这叫做函数作用域。这些变量称为局部变量。
- 在函数外部声明的变量作用域是从声明它们的那条语句开始到文件末尾，而不是函数内部。这叫做全局作用域。这些变量称为全局变量。
- 特殊的超级全局变量在函数内部和外部都是可见的。（请参阅第1章，“PHP快速入门教程”获得这些超级全局变量的更多信息。）
- 使用require()和include()并不影响作用域。如果这两个语句用于函数内部，函数作用域适用。如果它不在函数内部，全局作用域适用。
- 关键字“global”可以用来手动指定一个在函数中定义或使用的变量具有全局作用域。
- 通过调用unset($variable_name)可以手动删除变量。如果变量被删除，它就不在参数所指定的作用域中了。

下面的例子可能有助于我们更好地理解这些规则。

下面的代码没有输出。在这里，我们在函数fn()内部声明了一个名为$var的变量。因为这个变量是在函数内部声明的，所以它具有函数作用域并只在它所声明的地方存在，直到函数末尾。当在函数外部再次引用变量$var的时候，一个新的$var变量就会被创建。

这个新的变量具有全局作用域，在到达文件末尾之前都是可见的。不幸的是，如果唯一使用该变量的命令是echo，它将不会被赋值。

```
function fn() {
  $var = 'contents';
}
```

```
fn();
echo $var;
```

如下所示的例子刚好相反。我们在函数外部声明一个变量，然后在函数内部使用它：

```
<?
function fn() {
  echo 'inside the function, \$var = '.$var.'<br />";
  $var = "contents 2";
  echo 'inside the function, \$var = '.$var.'<br />";
}
$var = 'contents 1";
fn();
echo 'outside the function, \$var = '.$var.'<br />';
```

这段代码的输出如下所示：

```
inside the function, $var =
inside the function, $var = contents 2
outside the function, $var = contents 1
```

函数在被调用之前是不会执行的，所以第一条执行的语句是$var ='contents 1'。该语句创建了一个名为$var的变量，它具有全局作用域且值为“contents 1”。下一条执行的语句是调用函数fn()。函数内部的代码按顺序执行。函数第一行引用了一个名为$var的变量。当这行被执行时，就不能看到前面创建的变量$var了，所以它创建了一个具有函数作用域的新变量并将它显示出来。这就是输出的第一行。

函数内的下一行代码将变量$var的内容设为“contents 2”。因为是在函数内部，所以这一行改变了局部变量$var值，而不是那个全局变量。输出的第二行证实了这个改变。

函数执行到这里就结束了，这时脚本的最后一行被执行。这个echo命令显示了全局变量的值没有改变。

如果希望一个在函数内部创建的变量具有全局域，可以按如下方式使用关键字“global”：

```
function fn() {
  global $var;
  $var = "contents';
  echo "inside the function, \$var = ".$var."<br />";
}

fn();
echo "outside the function, \$var = '.$var.'<br />";
```

在这个例子中，变量$var被明确地声明为全局变量，这就意味着在函数调用结束之后，变量在函数外部也存在。这个脚本的输出如下所示：

```
inside the function, $var = contents
outside the function, $var = contents
```

请注意，变量的作用域是从执行global $var；这一行语句开始的。函数的声明可以在调用它之前或之后（请注意，函数的作用域不同于变量的作用域！），因此在哪里声明函数并

不重要，重要的是在哪里调用并执行其中的代码。

当一个变量要在整个脚本中都要用到时，也可以在脚本的开始处使用关键字“global”。

这可能是使用关键字global更常见的办法。

在前面的例子中，可以看到在函数的内部和外部重复命名一个变量名是合法的，而且两者互不影响。但是一般来说，这并不是一个好办法，因为如果不认真阅读代码并考虑作用域，人们可能会认为这些变量都是同一个。

5.8 参数的引用传递和值传递

如果希望编写一个名为increment()的函数来增加一个变量的值，我们可能会按如下方式编写这个函数：

```
function increment($value, $amount = 1) {
  $value = $value +$amount;
}
```

这段代码是没有用的。下面测试代码的输出结果是“10”。

```
$value = 10;
increment ($value);
echo $value;
```

$value的内容没有被修改。这要归因于作用域规则。这段代码将创建一个名为$value的变量，它的值是10。然后调用函数increment()。当函数被调用时，它内部的变量$value被创建。它的值加上1，所以$value在函数内部的值为11，直到函数结束，接下来我们返回到调用它的代码。在这段代码中，变量$value是一个不同的变量，具有全局域，所以它的值没有变。

解决这个问题的一个办法是将函数内的$value声明为全局变量，但这意味着为了使用这个函数，要进行变量运算的变量需要被命名为$value。

通常，函数获取参数的方式是值传递。当传递一个参数的时候，一个新的并且包含该传入值的变量被创建。它是原来那个变量的副本。可以以任意的方式修改它，但函数外部原来变量的值是不会改变的（这是PHP内部所实现的一个微小的简化）。

更好的办法是使用引用传递。这里，在参数被传递给函数的时候，函数不会再创建一个新变量，而是函数获得一个原来变量的引用。这个引用有一个变量名称，它以美元符号开始，可以像另一个变量那样使用它。其区别在于它不是获得变量本身的值，而是指向原来的值。任何对该引用的修改都会影响到原始变量值。

可以通过在函数定义的参数名前加一个地址符（&）来指定参数的引用传递。在函数调用处不用修改。

前面的increment()的例子就可以修改为引用传递参数，这样它就可以正常工作了。

```
function increment(&$value, $amount = 1) {
$value = $value +$amount;
}
```

现在，我们有了一个可运行的函数，而且可以任意给想要进行增量运算的变量命名。正如

前面所提到过的，在函数的内外使用同样的名称会引起混淆，所以我们给主脚本变量一个新的名称。如下所示的测试代码在调用increment()之前将显示10，调用之后会显示11。

```
$a = 10;
echo $a.'<br />';
increment ($a);
echo $a.'<br />';
```

5.9 使用Return关键字

关键字“return”将终止函数的执行。当一个函数的执行结束时，要么是因为所有命令都执行完了，要么就是因为使用了关键字“return”。在函数结束后，程序返回到调用函数的下一条语句。

如果调用了如下所示的函数，将只有第一条echo命令被执行：

```
function test_return() {
  echo 'This statement will be executed';
  return;
  echo "This statement will never be executed";
}
```

很明显，这并不是使用return命令的有用方法。通常，从函数中间返回的原因就是特定的条件已经被满足了。

一个错误条件是在程序执行到函数末尾之前使用“return”语句中断函数执行的最常见原因。例如，如果编写了一个能够判断两个数字大小的函数，而且当缺少任何一个参数时，你可能会希望退出函数执行。

```
function larger( $x, $y ) {
  if ((!isset($x)) || (!isset($y))) {
  echo "This function requires two numbers.';
return;
}
if ($x>=$y) {
  echo $x."<br/">;
} else {
    echo $y."<br/">;
  }
}
```

内置函数isset()将告诉我们一个变量是否已经被创建并被赋值了。在以上代码中，如果任何一个变量没有被赋值，就给出一条错误信息然后再返回。我们通过使用!isset()来进行测试，这意味着“没有被赋值”，所以if命令可以解释成“如果x没有被赋值或者y没有被赋值”。如果这两个条件中的任意一个为真，函数就返回。

如果“return”语句被执行了，函数中接下来的代码就会被忽略。程序执行将会返回到调用该函数的位置继续执行。如果两个参数都被赋值，则函数会显示两个中较大的那个。

如下所示代码：

```
$a = 1;
$b = 2.5;
$c = 1.9;
larger($a, $b);
larger($c, $a);
larger($d, $a);
```

将具有如下所示的输出：

```
2.5
1.9
This function requires two numbers
```

从函数返回一个值

希望从函数中退出并不是使用“return”语句的唯一理由。许多函数都使用“return”语句来与调用它们的代码进行交互。如果不仅仅是将larger()函数的比较结果输出，而是将比较的结果返回，那么这个函数会更加有用。通过这种方法，调用它的代码可以选择是否以及如何显示或者使用这个结果。等价的内置函数max()就是这样。

可以按如下所示方式编写larger()函数：

```
function larger ($x, $y) {
  if ((!isset($x)) || (!isset($y))) {
    return false;
  } else if ($x>=$y) {
    return $x;
  } else {
    return $y;
  }
}
```

在这里，函数返回了传入的两个参数中较大的那个。在出现错误的情况下，将返回一个明显不同的值。如果缺少其中任何一个数字，可以返回“false”。(使用这种方法唯一需要注意的是编程人员调用这个函数必须使用“===”测试返回类型，确保“false”不会与0混淆。)

作为比较，如果两个变量都没有被赋值，max()内置函数将不会返回任何东西，而如果只有其中一个变量被赋值，该内置函数将返回赋值的那个数。

如下所示的代码：

```
$a = 1; $b = 2.5; $c = 1.9;
echo larger($a, $b). <br />';
echo larger($c, $a).'<br />';
echo larger($d, $a).'<br />';
```

将产生如下所示的输出，因为$d并不存在，而且false并不是可见的：

```
2.5
1.9
```

通常，执行特定任务但又不返回任何具体值的函数将返回“true”或“false”来表示

函数执行是否成功。布尔值“true”和“false”可以分别用整数“1”和“0”来表示，虽然它们是不同的数据类型。

5.10 实现递归

PHP支持递归函数。*递归函数*就是函数调用自己本身。这些函数特别适用于浏览动态数据结构，例如连接列表和树。

但是，几乎没有基于Web的应用程序要求使用如此复杂的数据结构，所以我们很少使用递归函数。在很多情况下，递归可以用来取代循环，因为二者都是重复做一些事情。递归函数比循环慢而且要占用更多内存，所以应该尽可能多用些循环。

为了周全考虑，我们来看一个简单的例子，如程序清单5-5所示。

程序清单5-5 recursion.php——使用递归将一个字符串颠倒是非常简单的

```
<?php

function reverse_r($str) {
   if (strlen($str)>0) {
     reverse_r(substr($str, 1));
   }
   echo substr($str, 0, 1);
   return;
}

function reverse_i($str) {
   for ($i=1; $i<=strlen($str); $i++) {
      echo substr($str, -$i, 1);
   }
   return;
}

reverse_r('Hello');

reverse_i('Hello');
```

在这个程序清单中实现了两个函数。这两个函数都可以以相反的顺序打印字符串的内容。函数reverse_r()是通过递归实现的，而函数reverse_i()是通过循环实现的。

函数reverse_r()以一个字符串作为输入参数。当调用它的时候，它会继续调用它自己，每次传递从字符串的第二个到最后一个字符。例如，如果调用：

```
reverse_r('Hello');
```

它会用下面的参数多次调用自己：

```
reverse_r('ello');
reverse_r('llo');
reverse_r('lo');
reverse_r('o');
reverse_r('');
```

每次调用这个函数都在服务器的内存中生成一段该函数代码的新副本，但每次使用的参数是不同的。这有点像我们每次调用不同的函数。这样会使我们避免将这些函数的调用实例混淆。

在每次调用中，传入字符串的长度都会被测试。当到达字符串末尾的时候（`strlen()==0`），条件失败。最近的一次函数调用（`reverse_r('')`）会继续执行下一行代码，就是将传入字符串的第一个字符显示出来——在这个情况下没有字符，因为字符串是空的。

下一步，这个函数实例又将控制返回到调用它的实例中，也就是`reverse_r('o')`。这样就打印出了字符串的第一个字符（“o”）然后将控制返回到调用它的实例中。

如此继续打印一个字符后返回到调用它的上一层函数实例当中，直到程序控制返回主程序。

在递归方法中有一些非常优美而精确的东西。但是，在大多数情况下，最好还是使用循环方法。循环的代码也在程序清单5-5中给出。请注意，它没有递归方法长（虽然循环函数并不总是这样），但却也能实现相同的功能。最主要的不同在于，递归函数将在内存中创建几个自身的副本，而且将产生多次函数调用的开销。

当递归方法的代码比循环方法的代码更简短、更美观的时候，我们可能会选择使用递归，但是在应用领域通常不会这样。

虽然递归看上去更美观，但程序员常会忘记给出递归的终止条件。这意味着函数会一直重复下去直到服务器内存耗尽，或者达到了最大调用次数。

名称空间

通常，名称空间是一个抽象的容器，它可以包含一组标识符；在PHP中，名称空间可以包含你所定义的函数、常量以及类。从结构的角度看，为自定义函数和类定义名称空间的优点包括如下：

- 一个名称空间中的所有函数、类和常量都将自动冠以名称空间前缀。
- 非全路径的类、函数和常量名称将在运行时解析，在查看全局空间之前，将首先查看名称空间。

关于PHP中名称空间的更多信息和实用例子，请参阅PHP手册：

http://jp2.php.net/language.namespaces

5.11　进一步学习

`include()`、`require()`、`function`和`return`的用法在联机手册中也有介绍。要查找更多能够影响许多语言特性的、概念性的详细内容，例如递归、引用传递、值传递和作用域等，可以通过一本通用的计算机技术教材来获得，例如Dietel和Dietel's C++ *How To Program*。

5.12　下一章

现在我们已经了解了如何使用包含文件、请求文件和函数，可以使代码更易于维护而且可重复利用，在下一章中，我们将介绍面向对象的软件和PHP在这方面提供的支持。使用对象可以让读者接触到类似于本章中出现的各种概念，但对于复杂的项目来讲它具有更大的优势。