第一篇　使用PHP

第 1 章　PHP快速入门

本章简单介绍PHP的语法和语言结构。如果读者已经是PHP程序员，本章可能会弥补已有知识的一些不足。如果读者具有使用C、Perl、ASP或其他编程语言的背景，本章将帮助读者快速地掌握PHP语言。

在本书中，读者将通过取材于我们构建真实商业站点的实例来学习如何使用PHP。通常，一般的编程语言教科书只是通过非常简单的例子来介绍基本语法。我们决定不这么做。我们意识到读者最希望做的是运行这些例子，了解如何使用该语言，而不是逐个地学习类似于联机手册的语法和函数引用。

尝试运行这些例子。手工输入或从随书附带的文件载入这些例子，对它们进行修改或者分解它们，然后学习如何对它们进行修复。

在本章中，我们将以一个在线产品订单的例子为开始，学习在PHP中如何使用变量、操作符和表达式，其中还涵盖了变量类型和操作符优先级。读者还将学习如何访问订单中的表单变量，以及如何操作这些变量，从而计算出一个客户订单的总金额和税金。

接着，我们将使用PHP脚本开发一个能够验证客户输入数据的在线订单示例。

我们还将学习布尔值的概念，以及使用`if`、`else`、`?:`操作符以及`switch`语句的例子。最后，我们将学习循环语句，并使用这些语句编写一些可以生成HTML表格的PHP脚本。

在本章中，我们将主要介绍以下内容：

- 在HTML中嵌入PHP
- 添加动态内容
- 访问表单变量
- 理解标识符
- 创建用户声明的变量
- 检查变量类型
- 给变量赋值
- 声明和使用常量
- 理解变量的作用域
- 理解操作符和优先级
- 表达式
- 使用可变函数

- 使用if、else和switch语句进行条件判断
- 使用while、do和for迭代语句

1.1 开始之前：了解PHP

为了使用本章和本书所有例子，读者必须能够访问一个安装了PHP的Web服务器。

要掌握这些例子和实例研究，必须运行它们，并且尝试对其进行修改。此外，还将需要一个可以进行实验的测试平台。

如果机器还没有安装PHP，必须先安装PHP，或者让系统管理员为你安装。可以在附录A中找到安装指南。

1.2 创建一个示例应用：Bob的汽车零部件商店

任何服务器端脚本语言最常见的应用之一就是处理HTML表单。我们将为Bob汽车零部件商店（一个虚拟的汽车零部件公司）实现一个订单表单，从而开始PHP的学习。

1.2.1 创建订单表单

Bob的HTML程序员已经设置好Bob汽车零部件商店所销售的零部件订单。该订单如图1-1所示。这是一个相对比较简单的订单，类似于读者在Internet上看到的订单。

Bob希望能够知道他的客户订购了什么商品，订单的总金额以及该订单的税金。

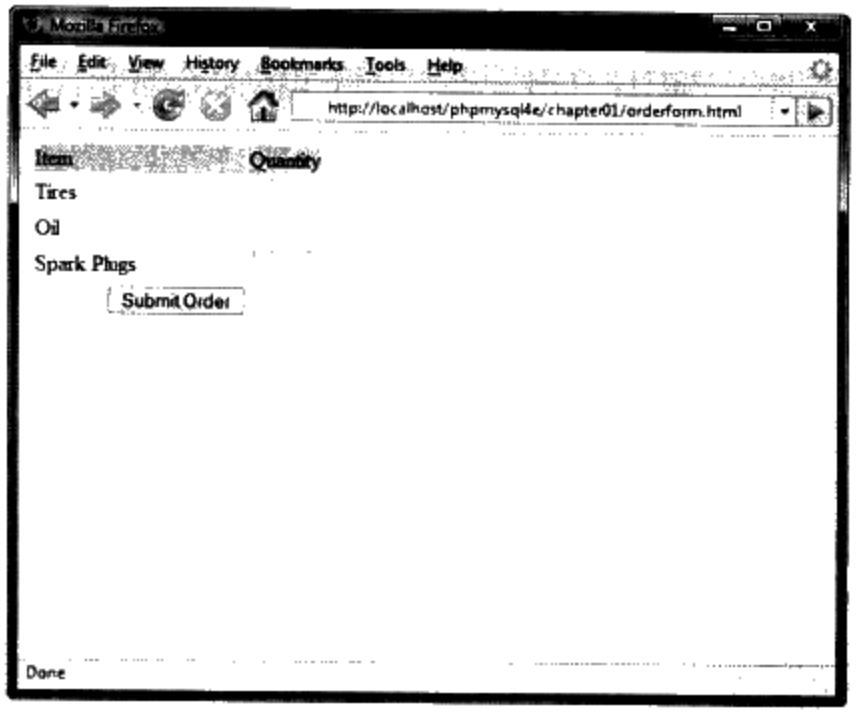

图1-1　Bob最初的订单只能记录商品和数量

程序清单1-1给出了该HTML页面的部分代码。

程序清单1-1　orderform.html——Bob基本订单表单的HTML代码

```
<form action= "processorder.php'method= "post">
<table border= "0'>
<tr bgcolor . "#cccccc'>
```

```
    <td width= "150">Item</td>
    <td width= "15">Quantity</td>
  </tr>
  <tr>
    <td>Tires</td>
    <td align= "center"><input type= "text"name= "tireqty"size= "3"
      maxlength= "3"/></td>
  </tr>
  <tr>
    <td>Oil</td>
    <td align= "center"><input type= "text"name= "oilqty"size= "3"
      maxlength= "3"/></td>
  </tr>
  <tr>
    <td>Spark Plugs</td>
    <td align= "center"><input type= "text name= "sparkqty "size= "3"
      maxlength= "3"/></td>
  </tr>
  <tr>
    <td colspan= "2"align= "center"><input type= "submit "value="Submit Order"/></td>
  </tr>
  </table>
  </form>
```

请注意，该表单的动作被设置为能够处理客户订单的PHP脚本名称（在稍后内容中，我们将编写该脚本）。一般地说，action属性值就是用户点击“Submit（提交）”按钮时将要载入的URL。用户在表单中输入的数据将按照method属性中指定的方法发送到这个URL，该方法可以是get（附加在URL的结尾）或post（以单独消息的形式发送）。

此外，还需要注意的是，表单域的名称——tireqty、oilqty和sparkqty。在这个PHP脚本中，我们还将使用这些名称。正是由于这一点，给表单域定义有意义的名称是非常重要的，因为当你编写PHP脚本时，我们就很容易记住这些名称。在默认的情况下，有些HTML编辑器将生成类似于field23的表单域名称。要记住这些名称是很困难的。如果表单域名称能够反映输入到该域的数据，PHP编程工作就会变得更加轻松。

读者可能会考虑对表单域名称的命名采用一种统一的编码标准，这样站点中的所有表单域名称就可以使用相同的格式。这样，无论在域名称中使用了词的缩写还是下画线，都可以轻松地记住它们。

1.2.2 表单处理

处理这个表单，我们需要创建在form标记的action属性中指定的脚本，该脚本为processorder.php。打开文本编辑器并创建该文件。输入如下所示的代码：

```
<html>
<head>
  <title>Bob's Auto Parts - Order Results</title>
```

```
</head>
<body>
<h1>Bob 's Auto Parts</h1>
<h2>Order Results</h2>
</body>
</html>
```

请注意，到目前为止，我们所输入的内容还只是纯HTML。现在，我们可以开始在这些脚本中添加一些简单的PHP代码。

1.3 在HTML中嵌入PHP

在以上代码的<h2>标记处，添加如下代码：

```
<?php
  echo '<p>Order processed.</p>';
?>
```

保存并在浏览器中载入该文件，填写该表单，点击“Submit Order”（提交表单）按钮。你将看到类似于图1-2所示的输出结果。

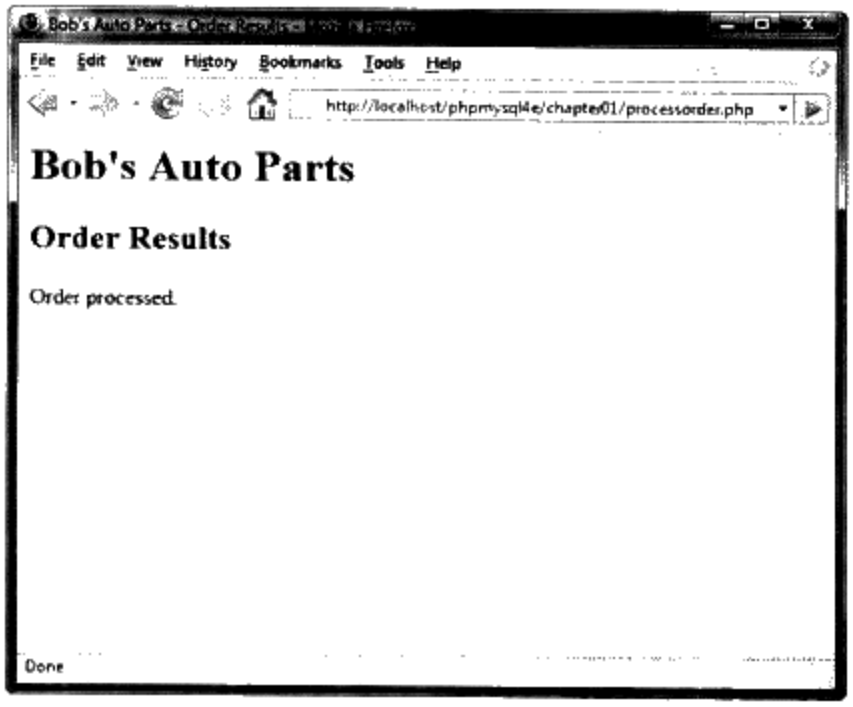

图1-2　传递给PHP echo语句中的文本显示在浏览器中

请注意，我们所编写的PHP代码是如何嵌入到一个常见的HTML文件中的。通过浏览器，查看该HTML的源代码。读者将看到如下所示的代码：

```
<html>
<head>
<title>Bob 's Auto Parts - Order Results</title>
</head>
<body>
<h1>Bob 's Auto Parts</h1>
<h2>Order Results</h2>
```

```
<p>Order processed.</p>
</body>
</html
```

以上代码并没有显示原始的PHP语句。这是因为PHP解释器已经运行了该脚本，并且用该脚本的输出代替了脚本本身。这就意味着，通过PHP，我们可以生成能在任何浏览器中查看的纯HTML——换句话说，用户的浏览器并不需要理解PHP。

这个例子简要地说明了服务器端脚本的概念。PHP脚本在Web服务器上被解释和执行，这与在用户机器上的Web浏览器中解释并执行的JavaScript及其他客户端技术是不同的。

现在，这个文件中的代码由如下4部分组成：

- HTML
- PHP标记
- PHP语句
- 空格

我们也可以添加注释。

在这个例子中的大多数语句行都只是纯HTML。

1.3.1 PHP标记

上例中的PHP代码是以“<? php”为开始，“?>”为结束。这类似于所有HTML标记，因为它们都是以小于号（<）为开始，大于号为结束（>）。这些符号（<?php和?>）叫做PHP标记，可以告诉Web服务器PHP代码的开始和结束。这两个标记之间的任何文本都会被解释成为PHP。而此标记之外的任何文本都会被认为是常规的HTML。PHP标记可以隔离PHP代码和HTML。

你也可以选择不同风格的PHP标记，如下内容将详细介绍。

PHP标记有4种不同的风格可供我们使用。如下所示的4段代码都是等价的。

- XML风格

```
<?php echo '<p>Order processed.</p>'; ?>
```

这是本书中将使用的标记风格。它是PHP推荐使用的标记风格。服务器管理员不能禁用这种风格的标记，因此可以保证在所有服务器上使用这种风格的标记，特别是编写用于不同服务器环境的应用程序时，这种标记风格尤为重要。这种风格的标记可以在XML（可扩展标记语言）文档中使用。通常，我们建议你使用这种风格。

- 简短风格

```
<? echo '<p>Order processed.</p>'; ?>
```

这种标记风格是最简单的，它遵循SGML（标准通用标记语言）处理说明的风格。要使用这种标记风格（输入字符最少）你必须在配置文件中启用short_open_tag选项，或者启用短标记选项编译PHP。在附录A，你可以找到关于如何使用这种标记风格的更多信息。不推荐使用这种风格的标记，因为这种风格在许多环境的默认设置中已经是不支持的。

- SCRIPT风格

```
<script language= 'php'> echo '<p>Order processed.</p>'; </script>
```

这种标记风格是最长的，如果读者使用过JavaScript或VBScript，就会熟悉这种风格。如果读者所使用的HTML编辑器无法支持其他标记风格，可以使用它。

■ ASP风格

```
<% echo '<p>Order processed.</p>' ; %>
```

这种标记风格与Active Server Pages（ASP）或ASP.NET的标记风格相同。如果在配置设定中启用了asp_tags选项，就可以使用它。如果读者所使用的编辑器不是专门为ASP或ASP.NET而设计的，就请不要使用它。但是请注意，在默认情况下，该标记风格是禁用的。

1.3.2 PHP语句

通过将PHP语句放置在PHP的开始和结束标记之间，我们可以告诉PHP解释器进行何种操作。在这个例子中，我们只使用了一种类型的语句：

```
echo '<p>Order processed.</p>';
```

正如读者可能已经猜到的那样，使用echo语句具有一个非常简单的结果：它将传递给其自身的字符串打印（或者回显）到浏览器。在图1-2中，可以看到该语句的结果，也就是“Order processed.”文本出现在浏览器窗口中。

请注意，在echo语句的结束处出现了一个分号。在PHP中，分号是用来分隔语句的，就像英文的点号用来分隔句子一样。如果读者以前使用过C或Java，将会习惯使用分号来分隔语句。

丢失这个分号是最容易出现的语法错误。但是，这也是最容易发现和修改的错误。

1.3.3 空格

间隔字符，例如换行（回车）、空格和Tab（制表符），都被认为是空格。正如读者可能已经知道的，浏览器将会忽略HTML的空格字符。PHP引擎同样会忽略这些空格字符。分析如下两段HTML代码：

```
<h1>Welcome to Bob 's Auto Parts!</h1><p>What would you like to order today?</p>
```

和

```
<h1>Welcome              to Bob's
Auto Parts!</h1>
<p>What would you like
to order today?</p>
```

这两段HTML代码将产生相同的输出，因为它们对浏览器来说都是相同的。但是，我们推荐在HTML的合适位置使用空格，因为这将提高HTML代码的可读性。这同样适用于PHP。虽然PHP语句之间完全没有必要添加任何空格字符，但是如果每一行放置一条单独的语句，将便于我们阅读代码。例如，

```
echo 'hello ';
echo 'world ';
```

和

```
echo 'hello ';echo 'world ';
```

是等价的，但是第一种代码更容易阅读。

1.3.4 注释

对于阅读代码的人来说，注释其实就相当于代码的解释和说明。注释可以用来解释脚本的用途、脚本编写人、为什么要按如此的方法编写代码、上一次修改的时间等。

通常，读者将在所有脚本中发现注释，最简单的PHP脚本除外。

PHP解释器将忽略注释中的任何文本。事实上，PHP分析器将跳过等同于空格字符的注释。

PHP支持C、C++和Shell脚本风格的注释。

如下所示的是一个C风格的注释，多行注释可以出现在PHP脚本的开始处：

```
/* Author: Bob Smith
   Last modified: Apri_ 10
   This script processes the customer orders.
*/
```

多行注释应该以/*为开始，*/为结束。与C语言中相同，多行注释是无法嵌套的。

你也可以使用C++风格的单行注释：

```
echo '<p>Order processed.</p> '; // Start printing order
```

或者Shell脚本风格：

```
echo '<p>Order processed.</p>  ; # Start printing order
```

无论何种风格的注释，在注释符号（#或//）之后行结束之前，或PHP结束标记之前的所有内容都是注释。

在如下所示的代码行中，关闭标记之前的文本，“here is a comment”是注释的一部分。而关闭标记之后文本，“here is not”将被当作是HTML，因为它位于关闭标记之外。

```
// here is a comment ?> here is not
```

1.4 添加动态内容

到这里，我们还没有使用PHP实现纯HTML不能实现的功能。

使用服务器端脚本语言的主要原因就是能够为站点用户提供动态内容。这是一个非常重要的应用，因为根据用户需求或随着时间的推进而变化的内容可以使得用户不断地访问这个站点。PHP就可以方便地实现这一功能。

举一个简单的例子。使用如下所示的代码替换processorder.php脚本中的PHP代码：

```
<?php
  echo "<p>Order processed at";
  echo date('H:i, jS F Y');
  echo "</p>";
?>
```

你也可以使用连接操作符（.）将其编写在一行代码中。

```
<?php
  echo "<p>Order processed at ".date('H:i, jS F Y').".</p>";
?>
```

在这段代码中，我们使用PHP的内置date()函数来告诉客户其订单被处理的日期和时间。当该脚本每一次运行时，将会显示不同的时间。该脚本的运行输出如图1-3所示。

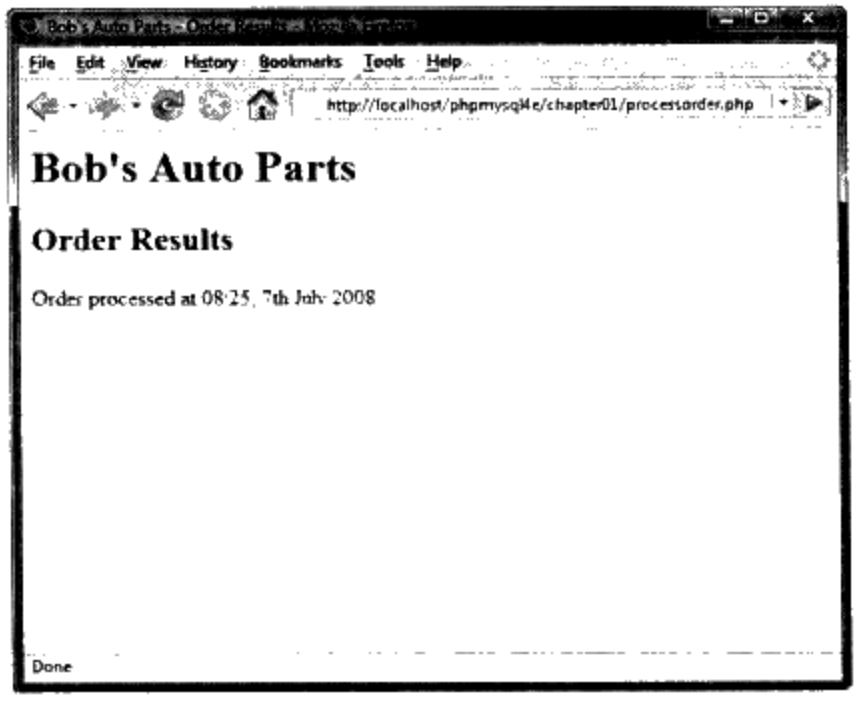

图1-3　PHP的date()函数返回一个格式化的日期字符串

1.4.1　调用函数

现在，看看date()函数的调用。这是函数调用的常见格式。PHP具有一个可供开发Web应用程序时使用的可扩展函数库。函数库中的大多数函数都需要传入的数据，并且返回一些数据。

看看如下所示的函数调用：

```
date('H:i, jS F Y')
```

请注意，我们将一个封闭在圆括号内的字符串（文本数据）传递给该函数。这个字符串就是函数的自变量或参数。这些自变量是输入的，而且函数本身使用这些自变量来输出某些特定结果。

1.4.2　使用date()函数

date()函数需要一个传递给它的变量是格式化字符串，这个字符串表示所需要的输出格式。字符串的每一个字母都表示日期和时间的一部分。H是24小时格式的小时，i是分钟，如果小时数和分钟数是个位数，需要在前面补0，j是该月的日期，不需要前面的补0，而s表示顺序后缀（在这个例子中，是“th”），F是月份的全称。

关于date()函数所支持的完整格式列表，请参阅第21章。

1.5 访问表单变量

使用订单的目的是为了收集客户订单。在PHP中，获得客户输入的具体数据是非常简单的，但是具体的方法还依赖于你所使用的PHP版本，以及php.ini文件的设置。

1.5.1 简短、中等以及冗长风格的表单变量

在PHP脚本中，可以用PHP变量的形式访问每一个表单域，其中PHP变量名称必须与表单域的名称一致。你可以很容易识别PHP的变量名称，因为它们都是以$符号开始的。（漏掉这个$符号是一个常见的编程错误。）

根据PHP版本和设置的不同，通过变量，可以有3种方法来访问表单数据。这些方法并没有正式的名称，因此我们给它们定义了3个昵称，分别是*简短风格*、*中等风格*和*冗长风格*。在任何情况下，一个页面上提交给PHP脚本的每一个表单域在PHP脚本中都是可以使用的。

你可以按如下所示的方法访问tireqty域的内容：

```
$tireqty                          // short style
$_POST[ 'tireqty']                // medium style
$HTTP_POST_VARS[ 'tireqty']       // long style
```

在这个例子以及整本书中，我们将使用中等风格（也就是，$_POST['tireqty']）来引用表单变量，但是为了便于使用，我们创建了简短风格的变量。然而，在代码中，我们采用这种风格而不是自动选择的风格来引用变量，是因为自动选择风格可能会带来安全性的问题。

对于读者自己的代码，读者可能会决定使用不同的方法，但是必须做出正确的选择。接下来我们将介绍上述不同的方法。

- 简短风格（$tireqty）非常方便，但是需要将register_globals配置选项设置为on。由于安全性的原因，在默认情况下，该选项的默认设定值为off。这种风格的标记容易导致产生安全性问题的错误，也就是为什么不推荐的原因。在PHP 6中，这个配置选项可能会被弃用，因此在新代码中使用这种风格的变量名称并不是好的想法。
- 中等风格（$_POST['tireqty']）是如今所推荐的。如果基于中等风格，创建简短版本的变量名称（我们将在整本书中使用这种方式）将不会产生安全性问题，只是便于使用的问题。
- 冗长风格（$HTTP_POST_VARS['tireqty']）是最详细的，但是请注意，它已经被弃用，因此从长远看，这种风格可能会被删除。这种风格过去曾是最容易移植的，但是如今可以通过register_long_arrays配置指令禁用它，这样可以改进性能。因此，同样地，如果不需要将你的代码安装在旧版本的服务器上，就没有必要在新代码中使用这种风格。

当使用简短风格时，脚本中的变量名称应该与HTML表单中的表单域名称相同。在脚本中，不需要声明变量或者创建这些变量。就如同向一个函数传递参数，这些变量将被传递到脚本中。如果读者使用这种风格，就可以使用类似于$tireqty的变量。表单中的tireqty域将在表单处理脚本中创建$tireqty变量。

对变量如此方便的访问是非常受欢迎的，但是在将该选项设置为on之前，读者应该想想为什么PHP开发小组将该选项设置为off。

像这样对变量的直接访问是非常方便的，但是这可能会使读者遇到破坏脚本安全性的编程错误。由于表单变量会自动转换成全局变量，因此在你所创建的变量与直接来自用户的不可信任的变量之间没有明显的区别。

如果没有对自己的变量赋给一个初始值，脚本用户就可以像表单变量一样传递变量和值，这样就可能造成混乱。如果选择使用方便的简短风格来访问变量，必须注意对你自己的变量赋予一个初始值。

中等风格涉及了从$_POST、$_GET或$_REQUEST数组之一检索变量。$_GET或$_POST数组之一都可以保存表单变量的细节。使用哪一个数组取决于提交表单时使用的方法是POST还是GET。此外，通过POST或GET方法及其组合方式提交的所有数据都可以通过$_REQUEST数组获得。

如果表单是通过POST方法提交的，tireqty文本输入框中的数据将保存在$_POST['tireqty']中。如果表单是通过GET方法提交的，数据将保存在$_GET['tireqty']。在任何一种情况下，数据都可以通过$_REQUEST['tireqty']获得。

这些数组被称作是超级全局（superglobal）变量。在我们介绍变量的作用域时，我们还将在稍后的内容详细介绍这些超级全局变量。

下面，让我们看一个创建便于使用的变量副本的例子。

要将一个变量的值复制给另一个变量，你可以使用赋值操作符。在PHP中，赋值操作符是等于号（=）。如下代码将创建一个名为$tireqty的新变量，并且将$POST['tireqty']的内容复制给这个新变量：

```
$tireqty = $_POST[  tireqty'];
```

将如下代码块放置在订单处理脚本的开始处。在本书中，处理表单数据的所有脚本的开始处都将包含与这个相似的代码块。由于这段代码不会产生任何输出，因此无论将这段代码放置在开始一个HTML页面的<html>和其他HTML标记之前还是之后，都不会有任何差异。通常，我们将这段代码放置在脚本的最开始处，这样容易查找。

```
<?php
  // create short variable names
  $tireqty = $_POST[ 'tireqty'];
  $oilqty = $_POST[ 'oilqty'];
  $sparkqty = $_POST[ 'sparkqty ];
?>
```

这段代码将创建3个新变量：$tireqty、$oilqty和$sparkqty，并且将通过POST方法从表单中传送过来的数据分别赋值给这3个变量。

要使该脚本能够完成一些可见的操作，可以在PHP脚本结束处添加如下所示的代码：

```
echo '<p>Your order is as follows: </p>';
echo $tireqty. 'tires<br /> ;
echo $oilqty. 'bottles of oil<br />';
```

```
echo $sparkqty. 'spark plugs<br />';
```

在这里，还没有检查变量内容，因此也无法确认一些重要数据已经进入了每一个表单域。尝试输入一些错误数据并且观察发生的事情。阅读本章的剩余部分后，读者可能希望尝试在该脚本中添加一些数据校验的逻辑。

像上例中，从用户输入直接获得输入并输出到浏览器是一个有风险的操作，它可能带来安全隐患。你应该对数据进行过滤。在第4章，我们将介绍输入过滤。在第16章将深入介绍安全性问题。

如果现在于浏览器中载入这个文件，该脚本输出结果将类似于图1-4。当然，具体的数值还取决于在表单中输入的数据。

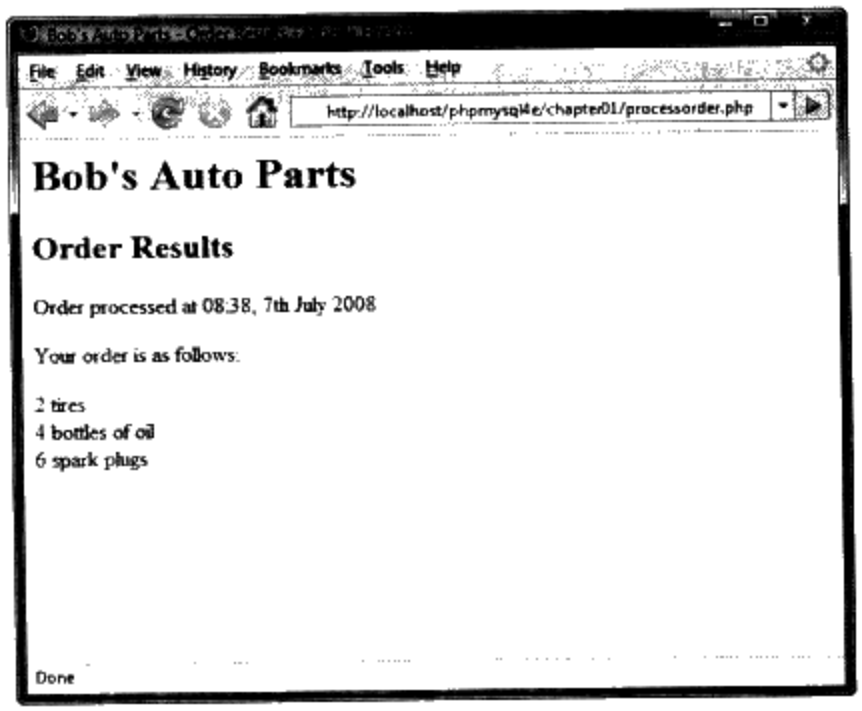

图1-4　在processorder.php中很容易访问用户输入的表单变量

在接下来的内容中，我们将介绍这个例子中几个有趣的方面。

1.5.2　字符串的连接

在这个示例脚本中，我们使用echo语句来打印用户在每一个表单域所输入的值，而这些值是跟随在一段说明性文本之后的。如果仔细查看echo语句，读者将发现在变量名称和后续文本之间存在一个点号（.），例如：

```
echo $tireqty. 'tires<br />';
```

这个点号是字符串连接符，它可以将几段文本连接成一个字符串。通常，当使用echo命令向浏览器发送输出时，将使用这个连接符。这可以用来避免编写多个echo命令。

对于任何简单类型的变量，都可以将变量写入到一个由双引号引起来的字符串中（数组变量要复杂一点，在第4章中将详细介绍数组和字符串的组合）。分析如下所示的例子：

```
echo "$tireqty tires<br />";
```

这个语句等价于本节介绍的第一个语句。这两种格式都是有效的，而且使用任何一种格式

都只是个人爱好问题。用一个字符串的内容来代替一个变量的操作就是插值（interpolation）。

请注意，插值操作只是双引号引用的字符串特性之一。不能像这样将一个变量名称放置在一个由单引号引用的字符串中。运行如下所示的代码：

```
echo '$tireqty tires<br /> ';
```

该代码将“$tireqty tires
”发送给浏览器。在双引号中，变量名称将被变量值所替代。而在单引号中，变量名称，或者任何其他文本都会不经修改而发送给浏览器。

1.5.3 变量和文本

在示例脚本中，每一个echo语句中连接在一起的变量和字符串是完全不同的。变量是表示数据的符号。字符串是数据本身。当我们在像这个脚本一样的程序中使用原始数据时，我们将其称之为文本，用来区分变量。$tireqty是一个变量，它是一个表示客户输入数据的符号。相反，'tires
'则是文本，它的值来自其字面值。记住了上一节中的第二个示例吗？PHP将用保存在变量中的值来代替字符串中的变量名称$tireqty。

请记住，已经介绍的两种字符串类型——一种是具有双引号的，而另一种是具有单引号的。PHP将试着计算双引号字符串，这样就导致了我们前面所看到的操作发生。而单引号字符串将被当作是真正的文本。

此外还有第3种指定字符串的方法：heredoc语法（<<<），Perl用户一定会熟悉这个语法。通过指定一个用来结束字符串的结束标记，Heredoc语法允许指定长字符串。

如下所示的代码创建了一个3行的字符串并且回显它们：

```
echo <<<theEnd
  line 1
  line 2
  line 3
theEnd
```

theEnd标记是非常模糊的，它只需要保证不会出现在文本中。要关闭一个heredoc字符串，可以在每一行的开始处放置一个关闭标记。

Heredoc字符串是插补的，就像双引号字符串。

1.6 理解标识符

标识符是变量的名称。（函数和类的名称也是标识符——我们将在第5章和第6章中详细介绍函数和类。）关于标识符，PHP定义了一些简单的规则：

- 标识符可以是任何长度，而且可以由任何字母、数字、下画线组成。
- 标识符不能以数字开始。
- 在PHP中，标识符是区分大小写的。$tireqty与$TireQty是不同的。交替地使用这些标识符是常见的编程错误。对于这个规则，函数名称是个例外——函数名称可以是任意大小写的。
- 一个变量名称可以与一个函数名称相同。这一点容易造成混淆，虽然是允许的，应该尽

量避免。此外，不能创建一个具有与已有函数同名的函数。

除了从HTML表单中传入的变量外，还可以声明并使用你自己的变量。

PHP的特性之一就是它不要求在使用变量之前声明变量。当第一次给一个变量赋值时，你才创建了这个变量——在下一节详细介绍。

就像我们将一个变量值复制给另一个变量一样，可以使用赋值操作符（=）给一个变量赋值。在Bob的站点上，我们希望计算出客户订购商品的总数和总金额。我们可以创建两个变量来保存这些数字。要创建两个变量，需要将每一个变量初始化为0。

在PHP脚本结束处中，添加如下所示的代码：

```
$totalqty = 0;
$totalamount = 0.00;
```

每一行代码都将创建一个变量并且赋给一个数值。你也可以将变量值赋值给一个变量，例如：

```
$totalqty = 0;
$totalamount = $totalqty;
```

1.7 检查变量类型

变量类型是指能够保存在该变量中的数据类型。PHP提供了一个完整的数据类型集。

不同的数据可以保存在不同的数据类型中。

1.7.1 PHP的数据类型

PHP支持如下所示的基本数据类型：

- Integer（整数）——用来表示整数
- Float（浮点数，也叫Double，双精度值）——用来表示所有实数
- String（字符串）——用来表示字符串
- Boolean（布尔值）——用来表示true或者false
- Array（数组）——用来保存具有相同类型的多个数据项（参阅第3章）
- Object（对象）——用来保存类的实例（参阅第6章）

此外还有两个特殊的类型：NULL（空）和resource（资源）。没有被赋值、已经被重置或者被赋值为特殊值NULL的变量就是NULL类型的变量。特定的内置函数（例如数据库函数）将返回resource类型的变量。它们都代表外部资源（例如数据库连接）。

基本上不能直接操作一个resource变量，但是通常它们都将被函数返回，而且必须作为参数传递给其他函数。

1.7.2 类型强度

PHP是一种非常弱的类型语言，或者动态类型语言。在大多数编程语言中，变量只能保存一种类型的数据，而且这个类型必须在使用变量之前声明，例如C语言。而在PHP中，变量的

类型是由赋给变量的值确定的。

例如，当我们创建$totalqty和$totalamount时，就确定了它们的初始类型，如下所示：

```
$totalqty = 0;
$totalamount = 0.00;
```

由于我们将0赋值给$totalqty，$totalqty就是一个整数类型的变量。同样，$totalamount是一个浮点类型的变量。

非常奇怪的是，我们可以在脚本中添加如下所示的语句：

```
$totalamount = 'Hello ;
```

$totalamount变量就可以是字符串类型的。PHP可以在任何时间根据保存在变量中的值来确定变量的类型。

这种在任何时间透明地改变变量类型的功能是非常有用的。

请记住，PHP将“自动地”获得输入的数据类型。一旦从变量中检索变量值，它将返回具有相同数据类型的数据。

1.7.3 类型转换

使用类型转换，可以将一个变量或值转换成另一种类型。这种转换与C语言的类型转换是相同的。只需在希望进行类型转换的变量之前的圆括号中插入需要转换的临时数据类型即可。

例如，我们可以使用类型转换声明上一节中的两个变量。

```
$totalqty = 0;
$totalamount = (float)$totalqty;
```

第2行代码的意思是“取出保存在$totalqty中的变量值，将其解释成一个浮点类型，并且将其保存在$totalamount”中。$totalamount变量将变成浮点类型。而被转换的变量并不会改变其类型，因此$totalqty仍然是整数类型。

你也可以使用PHP的内置函数来测试并设置类型，这将在本章稍后内容介绍。

1.7.4 可变变量

PHP提供了一种其他类型的变量——可变变量。可变变量允许我们动态地改变一个变量的名称。

可以看到，在这方面，PHP具有非常大的自由度——所有语言都允许改变变量的值，但是并没有太多的语言允许改变变量的类型，至于支持改变变量名称的语言就更少了。

这个特性的工作原理是用一个变量的值作为另一个变量的名称。例如，我们可以设置：

```
$varname = 'tireqty';
```

于是，我们就可以用$$varname取代$tireqty。例如，我们可以设置$tireqty的值：

```
$$varname = 5;
```

这个代码等价于：

```
$tireqty = 5;
```

这种代码看上去可能不太容易理解，但是我们将在以后的内容详细介绍它。不用单独列出并使用每一个表单变量，我们可以使用一个循环语句和一个变量来自动处理它们。

关于循环语句的使用将在本章稍后的第1.15.2节举例介绍。

1.8 声明和使用常量

正如读者在前面所看到的，我们可以改变保存在一个变量中的值。我们也可以声明常量。一个常量可以保存一个值，例如一个变量值，但是常量值一旦被设定后，在脚本的其他地方就不能再更改。

在示例应用中，可以将要出售的商品单价作为常量保存起来。你可以使用define函数定义这些常量：

```
define('TIREPRICE ', 100);
define('OILPRICE ', 10);
define('SPARKPRICE  ', 4);
```

现在，将这几行代码添加到脚本中。这样就有了3个用来计算顾客订单总金额的常量。

请注意，常量名称都是由大写字母组成的。这是借鉴了C语言的惯例，这样就可以很容易区分变量和常量。这个惯例并不是必需的，但是它却可以使代码变得更容易阅读和维护。

常量和变量之间的一个重要不同点在于引用一个常量的时候，它前面并没有$符号。

如果要使用一个常量的值，只需要使用其名称就可以了。例如，要使用一个以上已经创建的常量，可以输入：

```
echo TIREPRICE;
```

除了可以自己定义常量外，PHP还预定义了许多常量。了解这些常量的简单方法就是运行phpinfo()函数：

```
phpinfo();
```

这个函数将给出一个PHP预定义常量和变量的列表，以及其他有用的信息。我们将在以后的内容中逐步介绍它们。

变量和常量的另一个差异在于常量只可以保存布尔值、整数、浮点数或字符串数据。这些类型都是标量数据。

1.9 理解变量的作用域

作用域是指在一个脚本中某个变量可以使用或可见的范围。PHP具有6项基本的作用域规则。

- 内置超级全局变量可以在脚本的任何地方使用和可见。
- 常量，一旦被声明，将可以在全局可见；也就是说，它们可以在函数内外使用。
- 在一个脚本中声明的全局变量在整个脚本中是可见的，但不是在函数内部。
- 函数内部使用的变量声明为全局变量时，其名称要与全局变量名称一致。
- 在函数内部创建并被声明为静态的变量无法在函数外部可见，但是可以在函数的多次执行过程中保持该值（我们将在第5章全面介绍这个思想）。

■ 在函数内部创建的变量对函数来说是本地的，而当函数终止时，该变量也就不存在了。

$_GET和$_POST数组以及一些其他特殊变量都具有各自的作用域规则。这些被称作超级全局变量，它们可以在任何地方使用和可见，包括内部和外部函数。

超级全局变量的完整列表如下所示：

- $GLOBALS，所有全局变量数组（就像global关键字，这将允许在一个函数内部访问全局变量——例如，以$GLOBALS['myvariable']的形式。）
- $_SERVER，服务器环境变量数组
- $_GET，通过GET方法传递给该脚本的变量数组
- $_POST，通过POST方法传递给该脚本的变量数组
- $_COOKIE，cookie变量数组
- $_FILES，与文件上传相关的变量数组
- $_ENV，环境变量数组
- $_REQUEST，所有用户输入的变量数组，包括$_GET、$_POST和$_COOKIE所包含的输入内容（但是，不包括PHP 4.3.0版本以后的$_FILES）
- $_SESSION，会话变量数组

在本书以后的相关内容中，我们将逐个详细介绍这些变量。

在本章稍后介绍函数和类的时候，我们将详细介绍作用域。从现在开始，在默认情况下，我们所使用的所有变量都是全局变量。

1.10　使用操作符

操作符是用来对数值和变量进行某种操作运算的符号。我们必须使用其中的一些操作符来计算顾客订单总金额和应该缴纳的税金。

我们已经提到了两个操作符：赋值操作符（=）和字符串连接操作符（.）。现在，我们将了解完整的操作符列表。

一般地说，操作符可以带有1个、2个或者3个运算对象，其中大多数操作符都是带有两个运算对象。例如，赋值操作符就带有两个对象——左边的对象表示保存值的位置，右边的对象表示表达式。这些运算对象叫做操作数；即，要操作的对象。

1.10.1　算术操作符

算术操作符非常直观——它们就是常见的数学操作符。PHP的算术操作符如表1-1所示。

表1-1　PHP中的算术操作符

操 作 符	名　称	示　例
+	加	$a + $b
-	减	$a - $b
*	乘	$a * $b
/	除	$a / $b
%	取余	$a % $b

对于每一个操作符，我们可以保存运算后的结果。例如：

```
$result = $a + $b;
```

加法和减法与我们所想象的一样。这些操作符将$a和$b中的值相加减，然后再保存起来。还可以将减号当作一个一元操作符（也就是，一个只有一个运算对象或操作数的操作符）来使用，表示负值。例如：

```
$a = -1;
```

乘法和除法也与我们所想象的一样。请注意，我们使用星号（*）作为乘法操作符，而不是常规的乘法符号。同样，使用正斜线表示除法操作符，而不是常规的除法符号。

取余操作符返回的是$a除以$b以后的余数，请看如下所示的代码段：

```
$a = 27;
$b = 10;
$result = $a%$b;
```

变量$result中保存的值是27除以10以后的余数，也就是7。

应该注意到，算术操作符通常用于整型或双精度类型的数据。如果将它们应用于字符串，PHP会试图将这些字符串转换成一个数字。如果其中包含“e”或“E”字符，它就会被当作是科学表示法并被转换成浮点数，否则将会被转换成整数。PHP会在字符串开始处寻找数字，并且使用这些数字作为该字符串的值，如果没找到数字，该字符串的值则为0。

1.10.2　字符串操作符

我们已经了解并使用了唯一的字符串操作符。我们可以使用字符串连接操作符将两个字符串连接起来生成并保存到一个新字符串中，就好像我们使用加法操作符将两个数加起来一样。

```
$a = "Bob's ";
$b = "Auto Parts";
$result = $a.$b;
```

变量$result当前保存的值是“Bob's Auto Parts”字符串。

1.10.3　赋值操作符

我们已经了解了基本赋值操作符（=）。这个符号总是用作赋值操作符，其读法为“被设置为”。例如：

```
$totalqty = 0;
```

这个语句应该读成“$totalqty被设置为0”。我们将在本章后续小节介绍比较操作符时详细介绍其原因，但是如果将其读作等于，将会混淆它们。

1. 赋值运算返回值

与其他操作符一样，使用赋值操作符也会返回一个值。如果写入：

```
$a = $b
```

这个表达式的值就是将$a与$b加在一起所得到的结果。同样，如果写入：

```
$a = 0;
```

这个表达式的值为0。

这使你可以进行如下操作：

```
$b = 6 + ($a = 5);
```

这样赋给变量$b的值就是11。赋值运算的规则是：整个赋值语句的值将赋给左边的操作数。

当计算一个表达式的值时，可以使用圆括号来提高子表达式的优先级，正如上例所示。

这与数学当中的计算法则是相同的。

2. 复合赋值操作符

除了简单的赋值运算，PHP还提供了一系列复合的赋值操作符。每一个操作符都可以很方便地对一个变量进行运算，然后再将运算结果返回给原来的变量。例如：

```
$a +- 5;
```

以上语句等价于：

```
$a = $a + 5;
```

每一个算术操作符和字符串连接操作符都有一个对应的复合赋值操作符。表1-2给出了所有复合赋值操作符及其用途。

表1-2　PHP中的复合赋值操作符

操 作 符	使 用 方 法	等 价 于
+=	$a += $b	$a = $a + $b
-=	$a -= $b	$a = $a - $b
*=	$a *= $b	$a = $a * $b
/=	$a /= $b	$a = $a / $b
%=	$a %= $b	$a = $a % $b
.=	$a .= $b	$a = $a . $b

3. 前置递增递减和后置递增递减运算符

前置递增递减和后置递增递减运算符类似于+=和-=操作符，但是它们还存在一些区别。

所有递增操作符都有两个功能——将变量增加1后再将值赋给原变量。请看如下语句：

```
$a=4;
echo ++$a;
```

第2行代码使用了前置递增操作符，之所以这样命名是因为++符号出现在$a的前面。其运行结果是：首先将变量$a加1，再将加1后的结果赋值给原变量。这样，$a就变成了5，数值5被返回并显示到屏幕。整个表达式的值就是5。（请注意，实际上，保存在$a中的值已经发生变化：不仅仅是返回$a+1。）

但是，如果把++放在$a的后面，就是使用后置递增操作符。这个操作符的作用也有所不同。请看如下语句：

```
$a=4;
echo $a++;
```

这个语句的执行结果刚好相反。也就是，首先$a的值被返回并显示在屏幕上，然后，它再加1。这个表达式的值是4，也是屏幕上将要显示的结果。但是在执行完这个语句后，$a的值变成了5。

正如读者所想象的，操作符--的行为与操作符++的行为类似。但是，$a不是增加而是减少。

4. 引用操作符

引用操作符&可以在关联赋值中使用。通常，在将一个变量的值赋给另一个变量的时候，先产生原变量的一个副本，然后再将它保存在内存的其他地方。例如：

```
$a = 5;
$b = $a;
```

这两行代码首先产生$a的一个副本，然后再将它保存到$b中。如果随后改变$a的值，$b的值将不会改变：

```
$a = 7; // $b will still be 5
```

可以使用引用操作符&来避免产生这样的副本。例如：

```
$a = 5;
$b = &$a;
$a = 7; // $a and $b are now both 7
```

引用是非常有趣的。请记住，引用就像一个别名，而不是一个指针。$a和$b都指向了内存的相同地址。可以通过重置它们来改变所指向的地址，如下所示：

```
unset($a);
```

重置并不会改变$b(7)的值，但是可以破坏$a和值7保存在内存中的链接。

1.10.4　比较操作符

比较操作符用来比较两个值。比较操作符表达式根据比较结果返回逻辑值：true或false。

1. 等于操作符

相等的比较操作符==（两个等于号）允许测试两个值是否相等。例如，可以使用如下的表达式：

```
$a == $b
```

来测试$a和$b中的值是否相等。如果相等，这个表达式返回的结果为true，如果不等，这个表达式返回的结果为false。

这个操作符很可能会与赋值操作符“=”混淆。同时，即使出现混淆，程序也不会报错，但是通常不会返回你所希望的结果。一般地说，非0数值都是true，0值为false。假设按如下所示的语句初始化两个变量：

```
$a = 5;
$b = 7;
```

如果测试的是$a=$b，结果会是true。为什么呢？表达式$a=$b的值就是赋给左边的值，这个值为7。这是一个非0值，所以表达式的值是true。如果希望测试$a==$b，它的结果却是

false。这样，在编码中，就遇到了非常难发现的逻辑错误。通常，应该仔细检查这两个操作符的使用，确保所使用的操作符就是你要用的。

使用赋值操作符而不是等于比较操作符是一个很容易犯的错误，它可能在编程工作中多次出现。

2. 其他比较操作符

PHP还支持其他一些比较操作符。表1-3给出了所有比较操作符。需要注意的一点是，恒等操作符===（三个等于号）。只有当恒等操作符两边的操作数相等并且具有相同的数据类型时，其返回值才为true。例如，0=='0'将为true，但是0==='0'就不是true，因为左边的0是一个整数，而另一个0则是一个字符串。

表1-3　PHP中的比较操作符

操 作 符	名　称	使用方法
==	等于	$a == $b
===	恒等	$a === $b
!=	不等	$a != $b
!==	不恒等（比较操作符）	$a !== $b
<>	不等	$a <> $b
<	小于	$a < $b
>	大于（比较操作符）	$a > $b
<=	小于等于	$a <= $b
>=	大于等于	$a >= $b

1.10.5　逻辑操作符

逻辑操作符用来组合逻辑条件的结果。例如，我们可能对取值于0～100之间的变量$a的值感兴趣，那么我们可以使用“与”（AND）操作符测试条件$a>=0和$a<=100，如下所示：

```
$a >= 0 && $a <= 100
```

PHP支持逻辑与（AND）、或（OR）、异或（XOR）以及非（NOT）的运算。

表1-4给出了这个逻辑操作符的集合及其用法。

表1-4　PHP中的逻辑操作符

操 作 符	名　称	使用方法	结　果
!	非	!$b	如果$b是false，则返回true；否则相反
&&	与	$a && $b	如果$a和$b都是true，则结果为true；否则为false
||	或	$a || $b	如果$a和$b中有一个为true或者都为true时，其结果为true；否则为false
and	与	$a and $b	与&&相同，但其优先级较低
or	或	$a or $b	与||相同，但其优先级较低
xor	异或	$a xor $b	如果$a或$b为true，返回true，如果都是true或false，则返回false

操作符“and”和“or”比&&和||的优先级要低。在本章的后续内容中，我们将详细介绍优先级问题。

1.10.6 位操作符

位操作符可以将一个整型变量当作一系列的位（Bit，比特）来处理。在PHP中，读者可能发现它们并不经常使用，但是在这里，我们还是在表1-5中列出了它们。

表1-5 PHP中的位操作符

操作符	名称	使用方法	结果
&	按位与	$a & $b	将$a和$b的每一位进行“与”操作所得的结果
|	按位或	$a | $b	将$a和$b的每一位进行“或”操作所得的结果
~	按位非	~$a	将$a的每一位进行“非”操作所得的结果
^	按位异或	$a ^ $b	将$a和$b的每一位进行“异或”操作所得的结果
<<	左位移	$a << $b	将$a左移$b位
>>	右位移	$a >> $b	将$a右移$b位

1.10.7 其他操作符

到目前为止，除了我们已经介绍的操作符外，PHP还有一些其他操作符。

逗号操作符“,”是用来分隔函数参数和其他列表项的，这个操作符经常被附带地（非独立）使用。

两个特殊的操作符new和->，它们分别用来初始化类的实例和访问类的成员。它们将在第6章详细介绍。

此外，还有一些操作符，我们在这里简单地介绍一下。

1. 三元操作符

操作符?:语法格式如下所示：

```
condition ? value if true : value if false
```

三元操作符类似于条件语句if-else的表达式版本，这一点我们将在本章后续内容详细介绍。

举一个简单的例子：

```
($grade >= 50 ? 'Passed': 'Failed ')
```

这个表达式对学生级别进行评分，“Passed（及格）”或“Failed（不及格）”。

2. 错误抑制操作符

错误抑制操作符@可以在任何表达式前面使用，即任何有值的或者可以计算出值的表达式之前，例如：

```
$a = @(57/0);
```

如果没有@操作符，这一行代码将产生一个除0警告。使用这个操作符，这个警告就会被抑制住。

如果通过这种方法抑制了一些警告，一旦遇到一个警告，你就要写一些错误处理代码。

如果已经启用了PHP配置文件中的track_errors特性，错误信息将会被保存在全局变量$php_errormsg中。

3. 执行操作符

执行操作符实际上是一对操作符，它是一对反向单引号（``）。反向引号不是一个单引号，通常，它与~位于键盘的相同位置。

PHP将试着将反向单引号之间的命令当作服务器端的命令行来执行。表达式的值就是命令的执行结果。例如，在类似于UNIX的操作系统中，可以使用：

```
$out = `ls -la`;
echo '<pre> '.$out. '</pre> ';
```

在Windows服务器上，可以使用：

```
$out = `dir c:`;
echo '<pre> '.$out. '</pre> ';
```

这两种版本都会得到一个目录列表并且将该列表保存在$out中，然后，再将该列表显示在浏览器中或用其他方法来处理。

此外，还有其他方法可以执行服务器端的命令。我们将在第19章中详细介绍。

4. 数组操作符

PHP提供了一些数组操作符。数组元素操作符（[]）允许访问数组元素。在某些数组上下文中，也可以使用=>操作符。这些操作将在第3章详细介绍。

也可以使用许多其他数组操作符。我们会在第3章详细介绍它们，但是这里在表1-6中给出完整列表。

表1-6　PHP中的数组操作符

操 作 符	名　称	使用方法	结　果
+	联合	$a + $b	返回一个包含了$a和$b中所有元素的数组
==	等价	$a == $b	如果$a和$b具有相同的键值对，返回true
===	恒等	$a === $b	如果$a和$b具有相同的键值对以及相同的顺序，返回true
!=	非等价	$a != $b	如果$a和$b不是等价的，返回true
<>	非等价	$a <> $b	如果$a和$b不是等价的，返回true
!==	非恒等	$a !== $b	如果$a和$b不是恒等的，返回true

你将注意到，表1-6给出的数组操作符都有作用在标量变量上的等价操作符。只要你记得+执行了标量类型的加操作和数组的联合操作——即使你对其行为后面使用的实现算法不感兴趣——该行为也是有意义的。你不能将标量类型与数组进行比较。

5. 类型操作符

只有一个类型操作符：instanceof。这个操作在面向对象编程中使用，但是出于完整性方面的考虑，我们在这里也给出它（面向对象编程将在第6章详细介绍）。

instanceof操作符允许检查一个对象是否是特定类的实例，如下例所示：

```
class sampleClass();
$myObject = new sampleClass();
```

```
if ($myObject instanceof sampleClass)
  echo 'myObject is an instance of sampleClass ';
```

1.11　计算表单总金额

现在，你已经了解了如何使用PHP的操作符，下面就可以开始计算Bob订单表单的总金额和税金。要完成这些任务，可以将如下所示的代码添加到PHP脚本中：

```
$totalqty = 0;
$totalqty = $tireqty + $oilqty + $sparkqty;
echo "Items ordered: ".$totalqty."<br />";
$totalamount = 0.00;

define('TIREPRICE', 100);
define('OILPRICE', 10);
define('SPARKPRICE', 4);

$totalamount = $tireqty * TIREPRICE
             + $oilqty * OILPRICE
             + $sparkqty * SPARKPRICE;

echo 'Subtotal: $'.number_format($totalamount,2)."<br />";

$taxrate = 0.10; // local sales tax is 10%
$totalamount = $totalamount * (1 + $taxrate);
echo 'Total including tax: $'.number_format($totalamount,2)."<br />";
```

如果在浏览器窗口中刷新这个页面，将看到如图1-5所示的输出结果。

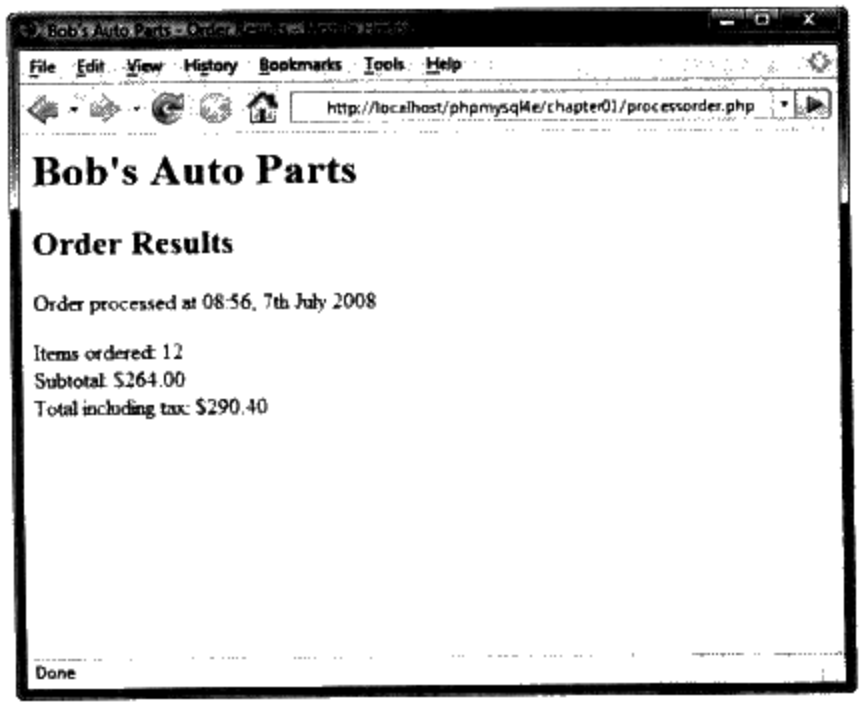

图1-5　显示了经过计算再格式化后的顾客订单总金额

可以看到，我们在这段代码中使用了一些操作符。我们使用了加号（+）和乘号（*）来计

算总量，还使用了字符串连接操作符（.）来格式化到浏览器的输出。

我们还使用了number_format()函数来格式化总金额的输出格式，将总金额的输出控制成带有两位小数的字符串。这个函数来自PHP的Math库。

如果要进一步了解计算过程，读者可能会问为什么计算会按着正确的算术顺序完成。

例如，请看如下代码：

```
$totalamount = $tireqty * TIREPRICE
             + $oilqty * OILPRICE
             + $sparkqty * SPARKPRICE;
```

总金额看上去是正确的，但是为什么乘号会在加号之前完成呢？答案就在于操作符的优先级，即操作符的执行顺序。

1.12　理解操作符的优先级和结合性

一般地说，操作符具有一组优先级，也就是执行它们的顺序。

操作符还具有结合性，也就是同一优先级的操作符的执行顺序。这种顺序通常有从左到右（简称左）、从右到左（简称右）或者不相关。

表1-7给出了PHP操作符的优先级和结合性。在这个表中，最上面的操作符优先级最低，按着表的由上而下的顺序，优先级递增。

表1-7　PHP中的操作符优先级

结 合 性	操 作 符
左	,
左	or
左	xor
左	and
右	print
左	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
左	?:
左	||
左	&&
左	|
左	^
左	&
不相关	== != === !==
不相关	< <= > >=
左	<< >>
左	+ - .
左	* / %
右	! ~ ++ -- (int) (double) (string) (array) (object) @
右	[ ]
不相关	New
不相关	()

请注意，我们还没有包括优先级最高的操作符：普通的圆括号。它的作用就是提高圆括号内部操作符的优先级。这样，在需要的时候，就可以避开操作符的优先级法则。

请记住这一部分的一个例子：

```
$totalamount - $totalamount * (1 + $taxrate);
```

如果写成：

```
$totalamount = $totalamount * 1 + $taxrate;
```

乘号就具有比加号更高的优先级，从而优先进行计算，这样就会得到一个错误的结果。

通过使用圆括号，可以强制先计算1+$taxrate子表达式。

可以在一个表达式中使用任意个圆括号，最里层圆括号的表达式将最先计算。

在上表中，另一个需要注意的但没有介绍的操作符是print语言结构，它等价于echo语句。这两个结构都将生成输出。

通常，在本书中，我们会使用echo，但是如果你认为print更容易阅读，也可以使用print语句。print和echo都不是真正的函数，但是都可以用带有参数的函数形式进行调用。二者都可以当作一个操作符：只要将要显示的字符串放置在echo或print关键字之后。

以函数形式调用print将使其返回一个值（1）。如果希望在一个更复杂的表达式中生成输出，这个功能可能是有用的，但是print要比echo的速度慢。

1.13　使用可变函数

在我们结束对变量和操作符的介绍之前，还要了解一下PHP的可变函数。PHP有一个函数库，这个函数库允许我们使用不同的方法来操作和测试变量。

1.13.1　测试和设置变量类型

大部分的可变函数都是用来测试一个函数的类型的。PHP中有两个最常见的函数，分别是gettype()和settype()。这两个函数具有如下所示的函数原型，通过它们可以获得要传递的参数和返回的结果：

```
string gettype(mixed var);
bool settype(mixed var, string type);
```

要使用gettype()函数，必须先给它传递一个变量。它将确定变量的类型并且返回一个包含类型名称的字符串：bool、int、double（对于浮点型）、string、array、object和resource。如果变量类型不是标准类型之一，该函数就会返回“unknown type（未知类型）”。

要使用settype()函数，必须先给它传递一个要被改变类型的变量，以及一个包含了上述类型列表中某个类型的字符串。

提示　本书和php.net文档都提到了“混合”数据类型。PHP并没有这个类型。但是，由于PHP在类型处理方面非常灵活，因此许多函数可以用许多（或者任意）的数据类型作为参数。这些类型所允许的参数通常都是伪“混合”类型。

我们可以按如下所示的方式使用这些函数：

```
$a = 56;
echo gettype($a). '<br />';
settype($a, 'double');
echo gettype($a). '<br />';
```

当第一次调用gettype()函数时，$a的类型是整数。在调用了settype()后，它就变成了双精度类型。

PHP还提供了一些特定类型的测试函数。每一个函数都使用一个变量作为其参数，并且返回true或false。这些函数如下。

- is_array()：检查变量是否是数组。
- is_double()、is_float()、is_real()（所有都是相同的函数）：检查变量是否是浮点数。
- is_long()、is_int()、is_integer()（所有都是相同的函数）：检查变量是否是整数。
- is_string()：检查变量是否是字符串。
- is_bool()：检查变量是否是布尔值。
- is_object()：检查变量是否是一个对象。
- is_resource()：检查变量是否是一个资源。
- is_null()：检查变量是否是为null。
- is_scalar()：检查该变量是否是标量，即，一个整数、布尔值、字符串或浮点数。
- is_numeric()：检查该变量是否是任何类型的数字或数字字符串。
- is_callable()：检查该变量是否是有效的函数名称。

1.13.2　测试变量状态

PHP有几个函数可以用来测试变量的状态。第一个函数就是isset()。它具有如下函数原型：

```
bool isset(mixed var );[;mixed var[,...]])
```

这个函数需要一个变量名称作为参数，如果这个变量存在，则返回true，否则返回false。

也可以传递一个由逗号间隔的变量列表，如果所有变量都被设置了，isset()函数将返回true。

还可以使用与isset()函数相对应的unset()函数来销毁一个变量。它具有如下所示的函数原型：

```
void unset(mixed var );[;mixed var[,...]])
```

这个函数将销毁一个传进来的变量。

函数empty()可以用来检查一个变量是否存在，以及它的值是否为非空和非0，相应的返回值为true或false。它具有如下所示的函数原型：

```
bool empty(mixed var);
```

现在，让我们来看一个使用这3个函数的例子。

尝试将如下所示的代码暂时添加到脚本中：

```
echo 'isset($tireqty):'.isset($tireqty). '<br />';
echo 'isset($nothere):'.isset($nothere). '<br />';
echo 'empty($tireqty):'.empty($tireqty). '<br />';
echo 'empty($nothere):'.empty($nothere). '<br />';
```

刷新页面，可以查看运行结果。

无论在那个表单域中输入了什么值，还是根本就没有输入任何值，isset()函数中的$tireqty变量都会返回1（true）。而在empty()函数中，它的返回值取决于在表单域中输入的值。

$nothere变量不存在，因此在isset()函数中它将产生一个空白结果（false），而在empty()函数中，将产生1（true）。

这些函数使用起来非常方便，可以确保用户正确地填写表单。

1.13.3 变量的重解释

可以通过调用一个函数来实现转换变量数据类型的目的。如下所示的3个函数可以用来实现这项功能：

```
int intval(mixed var [, int base]);
float floatval(mixed var);
string strval(mixed var);
```

每个函数都需要接收一个变量作为其输入，然后再将变量值转换成适当类型返回。

Intval()函数也允许在要转换的变量为字符串时指定转换的进制基数。（这样，就可以将十六进制的字符串转换成整数）。

1.14 根据条件进行决策

控制结构是一个程序语言中用来控制一个程序或脚本执行流程的结构，可以将它们分类为条件（或者分支）结构和重复结构（或循环结构）。

如果我们希望有效地响应用户的输入，代码就需要具有判断能力。能够让程序进行判断的结构称为条件。

1.14.1 if语句

我们可以使用if语句进行条件判断，但必须给出if语句的使用条件。如果条件为true，接下来的代码块就会被执行。If语句的条件必须用圆括号“()”括起来。

例如，如果一个访问者没有成功地在Bob的汽车零部件商店订购轮胎、汽油和火花塞，这很可能是由于她在完成填写表单之前不小心点击了“提交”按钮。页面应该能够告诉我们更有用的信息，而不是直接告诉我们“订单已经被处理”。

当访问者没有订购任何商品时，应该告诉用户“在前一页面你没有订购任何商品！”。

可以加入如下所示的if语句来实现这个功能：

```
if( $totalqty == 0)
  echo 'You did not order anything on the previous page!<br /> ';
```

在这里，我们所使用的条件为$totalqty==0。请记住，等于操作符（==）的作用与赋值操作符（=）的作用是不同的。

如果$totalqty等于0，那么条件$totalqty==0就会是true。如果$totalqty不等于0，条件表达式就会返回false。当条件为true时，echo语句就会被执行。

1.14.2 代码块

通常，根据一个条件语句（例如if语句）的动作不同，我们可能会希望执行多个语句。

我们可以将多个语句放在一起，将其组成一个代码块。要声明一个代码块，可以使用花括号将它们括起来：

```
if ($totalqty == 0) {
  echo '<p style="color:red">';
  echo 'You did not order anything on the previous page!';
  echo '</p>';
}
```

现在，这3行被花括号括起来的语句就组成了一个代码块。当条件语句为true时，这3行代码就会被执行。当条件语句为false时，这3行代码都将被忽略。

提示　正如我们已经介绍过的，PHP并不关心代码是如何布局的。但是，为了便于阅读代码，应该将它们缩进。通常，缩进可以使我们方便地找到一个if条件语句被满足时，哪些代码是可能要执行的，哪些语句是在代码块中，哪些语句是循环体或函数的一部分。在前面的例子中，可以发现需要根据if条件语句结果而执行的语句和包含在语句块中的语句都采用了缩进的格式。

1.14.3 else语句

通常，需要判断的不仅仅是希望执行的动作，还要判断一系列可能要执行的动作。

当if语句结果为false时，else语句可以使我们定义一个用来替换的动作。当Bob的顾客没有订购任何商品时，就要提示他们；如果他们订购了商品，就不需要提示他们，而是显示出他们所订购的商品。

如果对代码重新安排并且加入else语句，就可以显示提示信息或订购的汇总信息。

```
if ($totalqty == 0) {
  echo "You did not order anything on the previous page!<br />";
} else {
  echo $tireqty." tires<br />";
  echo $oilqty." bottles of oil<br />";
```

```
  echo $sparkqty." spark plugs<br />";
}
```

使用嵌套的if语句，可以建立更加复杂的逻辑处理。在接下来的代码中，我们不仅要在if条件$totalqty==0为true时显示提示信息，还要在每一个条件为true时显示每一个订单信息。

```
if ($totalqty == 0) {
  echo "You did not order anything on the previous page!<br />";
} else {
  if ($tireqty > 0)
    echo $tireqty." tires<br />";
  if ($oilqty > 0)
    echo $oilqty." bottles of oil<br />";
  if ($sparkqty > 0)
    echo $sparkqty." spark plugs<br />";
}
```

1.14.4 elseif语句

需要做出多个决定时，这些决定可能有多于两个的选项。我们可以使用elseif语句来建立一个多选项序列。elseif语句是else和if语句的结合。通过提供一系列的条件，程序将检查每一个条件，直到其找到一个为true的条件。

Bob为轮胎订单的大客户准备了一定的折扣。其折扣方案如下所示。

- 购买少于10个：没有折扣
- 购买10～49个：5%的折扣
- 购买50～99个：10%的折扣
- 购买100个以上：15%的折扣

可以使用条件表达式以及if和elseif语句来编写计算折扣的代码。这个例子必须使用“与”操作符（&&）将两个条件结合成一个条件。

```
if ($tireqty < 10) {
  $discount = 0;
} elseif (($tireqty >= 10) && ($tireqty <= 49)) {
  $discount = 5;
} elseif (($tireqty >= 50) && ($tireqty <= 99)) {
  $discount = 10;
} elseif ($tireqty >= 100) {
  $discount = 15;
}
```

请注意，这里可以将elseif语句随意写成elseif或else if——中间的空格是可有可无的。

如果要编写一系列的级联elseif语句，应该注意到，其中只有一个语句块将被执行。在这个例子中，对程序结果的计算并没有太大的关系，因为这些条件都是互斥的——每次都只有

一个条件为true。如果编写的条件语句其值同时为true的不止一个，那么只有第一个为true的条件下的语句或语句块将被执行。

1.14.5 switch语句

switch语句的工作方式类似于if语句，但是它允许条件可以有多于两个的可能值。在一个if语句中，条件或者为true，或者为false。而在switch语句中，只要条件值是一个简单的数据类型（整型、字符串或浮点型），条件就可以具有任意多个不同的值。

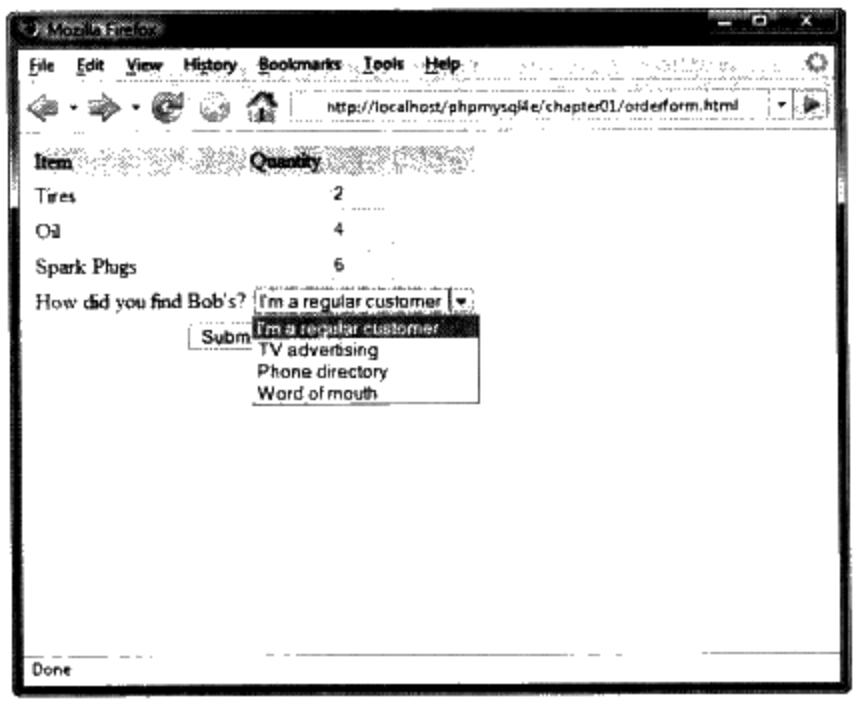

图1-6　现在的订单将询问访问者是通过哪种渠道知道Bob的汽车零部件商店

必须提供一个case语句来处理每一个条件值，并且提供相应的动作代码。此外，还应该有一个默认的case条件来处理没有提供任何特定值的情况。

Bob希望了解哪种广告对他的生意有所帮助。此时，可以在订单中加入一个调查问题，将如下所示的HTML代码插入到订单的表单体中，该表单的运行结果如图1-6所示。

```
<tr>
  <td>How did you find Bob  s?</td>
  <td><select name- "find">
        <option value = 'a">I 'm a regular customer</option>
        <option value = "b">TV advertising</option>
        <option value = "c">Phone directory</option>
        <option value = "d'>Word of mouth</option>
      </select>
  </td>

</tr>
```

上例的HTML代码中加入了一个新的表单变量（变量名为find），其值可以是“a”、“b”、“c”或“d”。可以使用一系列的if和elseif语句来处理这个新变量，如下所示：

```
if ($find -- "a") {
  echo "<p>Regular customer.</p>";
} elseif ($find == "b") {
  echo "<p>Customer referred by TV advert.</p>";
} elseif ($find == "c") {
  echo '<p>Customer referred by phone directory.</p>";
} elseif ($find == "d") {
  echo "<p>Customer referred by word of mouth.</p>';
```

```
} else {
  echo '<p>We do not know how this customer found us.</p>';
}
```

或者也可以用如下所示的switch语句来替换以上代码：

```
switch($find) {
  case "a" :
    echo "<p>Regular customer.</p>";
    break;
  case 'b' :
    echo "<p>Customer referred by TV advert.</p>';
    break;
  case "c" :
    echo "<p>Customer referred by phone directory.</p>";
    break;
  case "d" :
    echo '<p>Customer referred by word of mouth.</p>';
    break;
  default :
    echo "<p>We do not know how this customer found us.</p>';
    break;
}
```

（请注意，以上这两个例子都假设从$_POST数组中提取了$find变量。）

switch语句和if或elseif语句的行为有所不同。如果没有专门使用花括号来声明一个语句块，if语句只能影响一条语句。而switch语句刚好相反。当switch语句中的特定case被匹配时，PHP将执行该case下的代码，直至遇到break语句。如果没有break语句，switch将执行这个case以下所有值为true的case中的代码。当遇到一个break语句时，才会执行switch后面的语句。

1.14.6　比较不同的条件

如果读者对前面几节所介绍的语句还不熟悉，可能会问，“到底哪一个语句最好呢？”

其实，这个问题我们也无法回答。仅仅使用else、elseif或switch语句而不使用if语句无法完成任何事情。应该尽量在特定条件下使用特定的条件语句，即，根据实际情况来确定，这样可以使得代码具有更好的可读性。随着经验的不断积累，读者将慢慢体会到这一点。

1.15　通过迭代实现重复动作

计算机非常擅长的一件事情就是自动地、重复地执行任务。如果某些任务需要以相同的方式多次执行，可以使用循环语句来重复程序中的某些部分。

Bob希望在客户订单中加入一个运费表。对于Bob所使用的送货人来说，运费的多少取决于包裹要运送的距离。使用一个简单的公式就可以很容易地计算出所需的运费。

我们希望的运费表如图1-7所示。

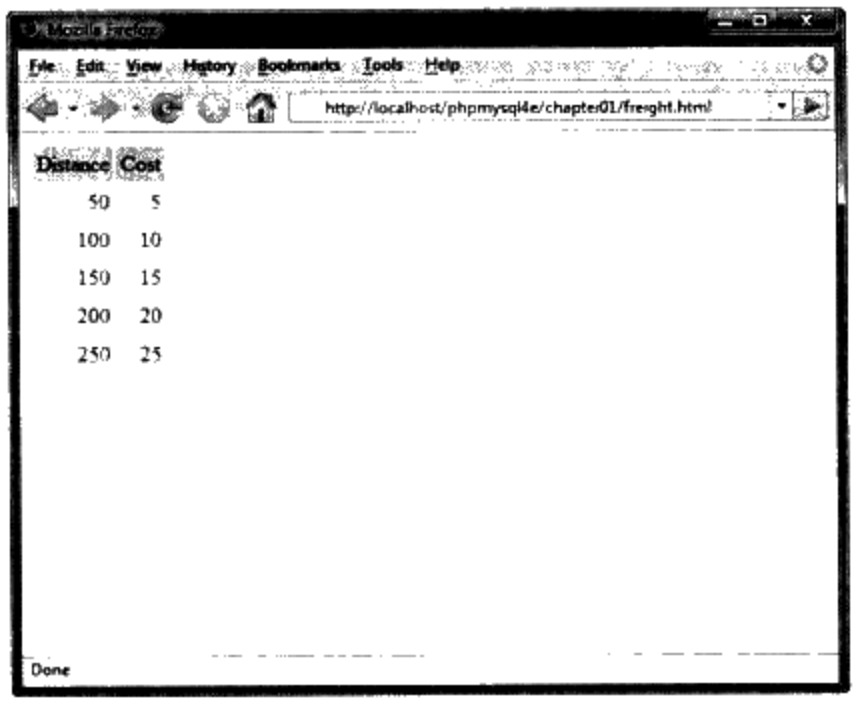

图1-7　运费表显示了运费随着距离的增加而增加

程序清单1-2给出了显示该运费表的HTML代码，可以看出这是一段很长而且重复的代码。

程序清单1-2　freight.html——Bob运费表的HTML代码

```
<html>
<body>
<table border= "0"cellpadding= "3">
<tr>
  <td bgcolor= '#CCCCCC "align= "center">Distance</td>
  <td bgcolor= '#CCCCCC "align= 'center">Cost</td>
</tr>
<tr>
  <td align= "right">50</td>
  <td align= 'right">5</td>
</tr>
<tr>
  <td align= "right">100</td>
  <td align= "right">10</td>
</tr>
<tr>
  <td align= 'right">150</td>
  <td align= 'right">15</td>
</tr>
<tr>
  <td align= 'right">200</td>
  <td align= "right">20</td>
</tr>
<tr>
  <td align= 'right'>250</td>
  <td align= 'right">25</td>
```

```
</tr>
</table>
</body>
</html>
```

如果我们能够使用一台低廉而又不知疲倦的计算机来完成这些，为什么还要找一个容易疲劳的人来输入这些HTML代码，并且还要付给他费用呢?

循环语句可以让PHP重复地执行一条语句或一个语句块。

1.15.1 while循环

PHP中最简单的循环就是while循环。就像if语句一样，它也依赖于一个条件。

while循环语句和if语句的不同就在于if语句只有在条件为true的情况下才执行后续的代码块一次，而while循环语句只要其条件为true，就会不断地重复执行代码块。

通常，当不知道所需的重复次数时，可以使用while循环语句。如果要求一个固定次数的重复，可以考虑使用for循环语句。

while循环的基本结构如下所示：

```
while( condition ) expression;
```

如下所示的while循环语句可以显示数字1～5。

```
$num = 1;
while ($num <= 5 ){
  echo $num. "<br /> ';
  $num++;
}
```

在每一次迭代的开始，都将对条件进行测试。如果条件为false，该语句块将不会执行，而且循环就会结束。循环语句后面的下一条语句将被执行。

我们可以使用while循环来完成一些更有意义的任务，例如显示图1-7所示的运费表。

程序清单1-3给出了使用while循环生成运费表的代码。

程序清单1-3　freight.php——用PHP生成的Bob运费表

```
<html>
<body>
<table border='0' cellpadding="3">
<tr>
  <td bgcolor="#CCCCCC" align="center">Distance</td>
  <td bgcolor "#CCCCCC" align='center">Cost</td>
</tr>
<?

$distance = 50;
while ($distance <= 250) {
  echo "<tr>
        <td align=\"right\">".$distance."</td>
```

```
        <td align=\"right\">".($distance / 10).'</td>
        </tr>\n';

  $distance += 50;
}

?>
</table>
</body>
</html>
```

为了使得由这个脚本所生成的HTML代码更具可读性，还需要增加一些新行和空格。正如我们已经介绍的，浏览器将忽略这些字符，但是这些字符对于阅读这段代码的人来说又非常重要。因此，必须经常查看这个HTML页面，确保页面输出就是你所希望的。

在程序清单1-3中，你会发现在一些字符串中出现了'\n'字符。当这个字符出现在一个引号的字符串中，它将被解释成一个换行字符。

1.15.2 for和foreach循环

前面所介绍的使用while循环的方法是非常常见的。我们可以设置一个计数器来开始循环。在每次迭代开始的时候，将在条件表达式中测试计数器。在循环的结束处，将修改计数器内容。

使用for循环，可以编写一个更为简洁和紧凑的代码来完成这种循环操作。for循环的基本结构是：

```
for(expression1; condition; expression2)
  expression3;
```

- *expression1*（表达式1）在开始时只执行一次。通常，可以在这里设置计数器的初始值。
- 在每一次循环开始之前，*condition*（条件）表达式将被测试。如果条件表达式返回值为false，循环将结束。通常，可以在这里测试计数器是否已经到达临界值。
- *expression2*（表达式2）在每一次循环结束时执行。通常，可以在这里调整计数器的值。
- *expression3*（表达式3）在每一次循环中执行一次。通常，这个表达式是一个包含大量循环代码的代码块。

我们可以用for循环重写程序清单1-3中的while循环语句。在这个例子中，这段PHP代码可以变为：

```
<?php
for ($distance = 50; $distance <= 250; $distance += 50) {
  echo "<tr>
        <td align=\'right\'>".$distance."</td>
        <td align=\'right\'>".($distance / 10).'</td>
        </tr>\n";}
?>
```

在功能方面，while版本的循环语句和for版本的循环语句是等价的。for循环更加紧凑，它节省了两行代码。

这两种循环是等价的——不能说哪种更好或者更糟糕。在特定的情况下，可以根据自己的喜好和感觉选择要使用的循环语句。

需要注意的一点是，我们可以将可变变量和for循环结合起来重复一系列的表单域。

例如，如果你具有名称为name1、name2、name3等的表单域，就可以像如下代码所示的这样进行处理：

```
for ($i=1; $i <= $numnames; $i++){
  $temp= 'name$i ';
  echo $$temp. '<br /> '; // or whatever processing you want to do
}
```

通过动态地创建变量名称，可以依次访问每一个表单域。

除了for循环外，PHP还提供了foreach循环语句，它专门用于数组的使用。我们将在第3章中详细介绍如何使用该语句。

1.15.3 do...while循环

现在，我们要介绍的最后一个循环语句与前面所介绍的循环语句有所不同。

do...while语句的常见结构如下所示：

```
do
    expression;
while( condition );
```

do...while循环与while循环不同，因为它的测试条件放在整个语句块的最后。这就意味着do...while循环中的语句或语句块至少会执行一次。

即使我们所采用的例子的条件在已开始就是false，而且永远不会是true，这个循环在检查条件和结束之前还是会执行一次。

```
$num = 100;
do{
  echo $num. '<br /> ";
}while ($num < 1 ) ;
```

1.16 从控制结构或脚本中跳出

如果希望停止一段代码的执行，根据所需要达到的效果不同，可以有3种方法来实现。

如果希望终止一个循环，可以使用在介绍switch循环时提到的break语句。如果在循环中使用了break语句，脚本就会从循环体后面的第一条语句处开始执行。

如果希望跳到下一次循环，可以使用continue语句。

如果希望结束整个PHP脚本的执行，可以使用exit语句。当执行错误检查时，这个语句非常有用。例如，可以按如下方式修改前面所介绍的例子：

```
if($totalqty == 0){
  echo 'You d d not order anything on the previous page!<br /> ';
  exit;
}
```

可以调用exit来终止PHP的执行，从而不会执行剩下的脚本。

1.17　使用可替换的控制结构语法

对于我们已经介绍过的所有控制结构，还有一个可替换的语法形式。它由替换开始花括号（{）的冒号（:）以及替换关闭花括号（}）的新关键字组成，这个新关键字可以是endif、endswitch、endwhile、endfor或endforeach，这是由所使用的控制结构确定的。对于do...while循环，没有可替换的语法。

例如，如下代码：

```
if ($totalqty == 0) {
  echo 'You did not order anything on the previous page!<br />';
  exit;
}
```

使用if和endif关键字，可以转换成如下所示的替换语法：

```
if ($totalqty == 0) :
  echo "You did not order anything on the previous page!<br />";
  exit;
endif;
```

1.18　使用declare

PHP的另一个控制结构是declare结构，它并没有像其他结构一样在日常编程中经常使用。这种控制结构的常见形式如下所示：

```
declare ( directive )
{
// block
}
```

这种结构用来设置代码块的执行指令——也就是，关于后续代码如何运行的规则。

目前，只实现了一个执行指令，ticks。它可以通过插入指令ticks=*n*来设置。它允许在代码块内部每隔*n*行代码运行特定的函数，这对于性能测试和调试来说是非常有用的。

在这里，介绍declare控制结构只是为了完整性的考虑。我们将在第25章和第26章中详细介绍一些关于如何使用ticks功能的例子。

1.19　下一章

现在，读者已经了解如何接收和操作客户的订单了。在下一章中，我们将介绍如何保存订单，以便以后的检索和执行。