

第3章 使用数组

本章将介绍如何使用一个重要的编程结构——数组。在前面的章节中，我们所介绍的变量都是标量变量，这些变量只能存储单个数值。数组是一个可以存储一组或一系列数值的变量。一个数组可以具有许多个元素。每个元素有一个值，例如文本、数字或另一个数组。一个包含其他数组的数组称为多维数组。

PHP支持数字索引数组和关联数组。如果曾经使用过任何其他编程语言，你可能会很熟悉数字索引的数组，但是如果你没有使用过PHP或Perl，无论是否曾经使用过Hash、Map或dictionary对象，你可能就从来没有见过关联数组。关联数组允许你使用更有意义的信息作为索引。每个元素除了可以使用数字索引外，还可以使用字符串或其他有意义的信息作为索引。

在本章中，我们将使用数组继续开发Bob汽车配件商店的例子，使用数组可以更容易地处理重复信息，例如客户的订单。而且，将写出更简洁、更整齐的代码来完成前面章节中所实现的文件处理操作。

在本章中，将主要介绍以下内容：

- 数字索引数组
- 非数字索引数组
- 数组操作符
- 多维数组
- 数组排序
- 数组函数

3.1 什么是数组

在第1章中介绍了标量变量。一个标量变量就是一个用来存储数值的命名区域。同样，一个数组就是一个用来存储一系列变量值的命名区域，因此，可以使用数组组织标量变量。

在前面的例子中，将以Bob的产品列表作为数组的示例。在图3-1中，可以看到一个按数组格式存储的3种产品的列表，数组变量的名称为`$products`，它保存了3个变量值。

（下面将介绍如何创建一个类似于这个数组变量的变量。）

拥有数组信息后，就可以用它完成很多有用的事情。使用第1章的循环结构，可以完成针对数组中每个值的相同操作，这样就可以节省许多工作。数组信息的整个集合可以作为一个单元进行移动。通过这种方式，只要使用一行代码，所有数值就可以传递给一个函数。例如，希望按字母顺序对产品进行排序。要完成此操作，可以将整个数组传递给PHP的`sort()`函数。

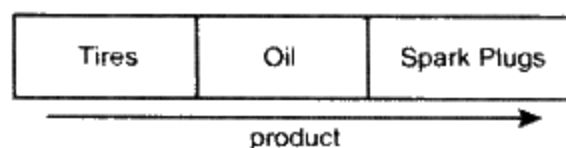


图3-1 Bob的产品信息可以用数组存储

存储在数组中的值称为数组元素。每个数组元素有一个相关的索引（也称为关键字），它可以用来访问元素。在大多数编程语言中，数组都具有数字索引，而且这些索引通常是从0或1开始的。

PHP允许间隔性地使用数字或字符串作为数组的索引。可以将数组的索引设置为传统的数字型，也可以将索引设置为任何所希望的索引，这样可以使得索引更有意义和有用（如果曾经使用过其他编程语言的关联数组或映射表，你可能就会熟悉这种方法）。根据是否使用标准数字索引数组或更有趣的索引值的不同，编程方法也各不相同。

下面，将先从数字索引数组开始。然后介绍如何使用用户自定义的关键字。

3.2 数字索引数组

大多数编程语言都支持这种数组。在PHP中，数字索引的默认值是从0开始的，当然也可以改变它。

3.2.1 数字索引数组的初始化

要创建如图3-1所示的数组，可以使用如下所示的代码：

```
$products = array('Tires', 'Oil', 'Spark Plugs');
```

以上代码将创建一个名为\$products的数组，它包含3个给定值——“Tires”、“Oil”和“Spark Plugs”。请注意，就像echo语句一样，array()实际上是一个语言结构，而不是一个函数。

根据对数组内容的需求不同，可能不需要再像以上例子一样对它们进行手工的初始化操作。如果所需数据保存在另一个数组中，可以使用运算符“=”简单地将数组复制到另一个数组。

如果需要将按升序排列的数字保存在一个数组中，可以使用range()函数自动创建这个数组。如下这行代码将创建一个1~10的数字数组：

```
$numbers = range(1, 10);
```

range()函数具有一个可选的第三个参数，这个参数允许设定值之间的步幅。例如，如需建立一个1~10之间的奇数数组，可以使用如下代码：

```
$odds = range(1, 10, 2);
```

range()函数也可以对字符进行操作，如下例所示：

```
$letters = range('a', 'z');
```

如果信息保存在磁盘文件中，可以从这个文件直接载入到数组中。这一点，将在3.9节中详细介绍。

如果数组中使用的数据保存在数据库中，可以从数据库中直接载入数组。在第11章中，将详细介绍这一点。

还可以使用不同的函数来提取数组中的一部分数据，或对数组进行重新排序。在3.10节，将详细介绍这些函数。

3.2.2 访问数组的内容

要访问一个变量的内容，可以直接使用其名称。如果该变量是一个数组，可以使用变量名称和关键字或索引的组合来访问其内容。关键字或索引将指定们要访问的变量。索引在变量名称后面用方括号括起来。

使用`$products[0]`、`$products[1]`、`$products[2]`，就可以使用数组`$products`的内容了。

在默认的情况下，0元素是数组的第一个元素。这和C语言、C++、Java以及许多其他编程语言的计数模式是相同的。如果你对这些内容很陌生，就应该先熟悉一下。

像其他变量一样，使用运算符“=”可以改变数组元素的内容。如下代码将使用“Fuses”替换第一个数组元素中的“Tires”。

```
$products[0] = 'Fuses';
```

而如下代码可以增加一个新的元素（“Fuses”）到数组末尾，这样，可以得到一个具有4个元素的数组：

```
$products[3] = 'Fuses';
```

要显示其内容，可以使用如下代码：

```
echo "$products[0] $products[1] $products[2] $products[3]";
```

请注意，虽然PHP的字符串解析功能非常强大和智能，但是可能会引起混淆。当你将数组或其他变量嵌入双引号中的字符串时，如果不能正确解释它们，可以将它们放置在双引号之外，或者使用在第4章中介绍的更复杂的语法。以上的echo语句是没有语法错误的，但是在本章后面出现的其他更复杂的例子中，读者将发现变量被放置在双引号之外。

就像PHP的其他变量一样，数组不需要预先初始化或创建。在第一次使用它们的时候，它们会自动创建。

如下代码创建了一个与前面使用`array()`语句创建的`$products`数组相同的数组：

```
$products[0] = 'Tires';
$products[1] = 'Oil';
$products[2] = 'Spark Plugs';
```

如果`$products`并不存在，第一行代码将创建一个只有一个元素的数组。而后续代码将在这个数组中添加新的数值。数组的大小将根据所增加的元素多少动态地变化。这种大小调整功能并没有在其他大多数编程语言中应用。

3.2.3 使用循环访问数组

由于数组使用有序的数字作为索引，所以使用一个for循环就可以很容易地显示数组的内容：

```
for ($i = 0; $i < 3; $i++) {
    echo $products[$i]. " ";
}
```

以上循环语句将给出类似于前面的结果，但是，相对于通过手工编写代码来操作一个大数

组来说，这样做需要手工输入的代码更少。使用一个简单的循环就可以访问每个元素是数字索引数组的一个非常好的特性。也可以使用foreach循环，这个循环语句是专门为数组而设计的。在这个例子中，可以按如下所示的方式使用它：

```
foreach ($products as $current) {  
    echo $current.' ' ;  
}
```

以上代码将依次保存\$current变量中的每一个元素并且打印它们。

3.3 使用不同索引的数组

在\$products数组中，允许PHP为每个元素指定一个默认的索引。这就意味着，所添加的第一个元素为元素0，第二个为元素1等。PHP还支持关联数组。在关联数组中，可以将每个变量值与任何关键字或索引关联起来。

3.3.1 初始化关联数组

如下所示的代码可以创建一个以产品名称作为关键字、以价格作为值的关联数组：

```
$prices = array( 'Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4);
```

关键字和值之间的符号只是一个在大于号之前的等于符号。

3.3.2 访问数组元素

同样，可以使用变量名称和关键字来访问数组的内容，因此就可以通过这样的方式访问保存在prices数组中的信息，例如\$prices['Tires']、\$prices['Oil']、\$prices['Spark Plugs']。

如下代码将创建一个与\$prices数组相同的数组。这种方法并不是创建一个具有3个元素的数组，而是创建一个只有一个元素的数组，然后再加上另外两个元素：

```
$prices = array( 'Tires'=>100 );  
$prices['Oil'] = 10;  
$prices['Spark Plugs'] = 4;
```

如下这段代码有些不同，但其功能与以上代码是等价的。在这种方法中，并没有明确地创建一个数组。数组是在向这个数组加入第一个元素时创建的。

```
$prices['Tires'] = 100;  
$prices['Oil'] = 10;  
$prices['Spark Plugs'] = 4;
```

3.3.3 使用循环语句

因为关联数组的索引不是数字，因此无法在for循环语句中使用一个简单的计数器对数组进行操作。但是可以使用foreach循环或list()和each()结构。

当使用foreach循环语句对关联数组进行操作时，foreach循环具有不同的结构。可以

在前面的例子中使用这个循环语句，也可以按如下方式使用关键字：

```
foreach ($prices as $key => $value) {
    echo $key." - ".$value."<br />";
}
```

如下所示的代码将使用each()结构打印\$prices数组的内容：

```
while ($element = each($prices)) {
    echo $element['key'];
    echo ' - ' ;
    echo $element['value'];
    echo '<br />';
}
```

以上脚本段的输出结果如图3-2所示。

在第1章中，介绍了while循环和echo语句。以上这段代码使用了前面从没有使用过的each()函数。这个函数将返回数组的当前元素，并将下一个元素作为当前元素。因为在while循环中调用each()函数，它将按顺序返回数组中每个元素，并且当它到达数组末尾时，循环操作将终止。

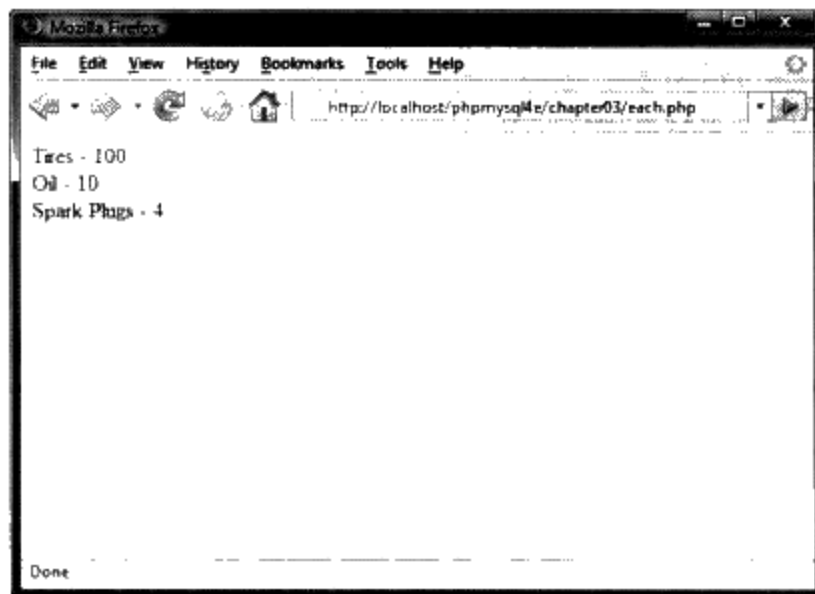


图3-2 数组循环使用each()语句

在这段代码中，变量\$element是一个数组。当调用each()时，它将返回一个带有4个数值和4个指向数组位置的索引的数组。位置key和0包含了当前元素的关键字，而位置value和1包含了当前元素的值。虽然这与选哪一种方法没什么不同，但这里选择了使用命名位置，而不是数字索引位置。

此外，还有一种更为高级和常见的方式来完成相同的操作。函数list()可以用来将一个数组分解为一系列的值。可以按照如下方式将函数each()返回的两个值分开：

```
while (list($product, $price) = each($prices)) {
    echo "$product - $price<br />";
}
```

以上代码使用each()从\$prices数组中取出当前元素，并且将它作为数组返回，然后再指向下一个元素。它还使用list()将从each()返回的数组中所包含0、1两个元素变为两个名为\$product和\$price的新变量。

我们可以循环遍历整个\$prices数组，使用如下所示的简短脚本显示它的内容：

```
reset($prices);
while (list($product, $price) = each($prices)) {
    echo "$product - $price<br />";
}
```

这段代码的输出结果与前面脚本的输出结果相同，但它更容易阅读，因为list()允许为新变量命名。

需要注意的一点是，当使用`each()`函数时，数组将记录当前元素。如果希望在相同的脚本中两次使用该数组，就必须使用函数`reset()`将当前元素重新设置到数组开始处。要再次遍历`prices`数组，可以使用如下所示的代码：

```
reset($prices);
while ( list( $product, $price ) = each( $prices ) )
    echo "$product: - $price<br />";
```

以上代码可以将当前元素重新设置到数组开始处，因此允许再次遍历数组。

3.4 数组操作符

只有一个特殊的操作符集适用于数组。这个集合中的大多数操作符都有与之对应的标量操作符，如表3-1所示。

表3-1 PHP中的数组操作符

操作符	名 称	示 例	结 果
+	联合	<code>\$a + \$b</code>	<code>\$a</code> 和 <code>\$b</code> 的联合。数组 <code>\$b</code> 将被附加到 <code>\$a</code> 中，但是任何关键字冲突的元素将不会被添加
==	等价	<code>\$a == \$b</code>	如果 <code>\$a</code> 和 <code>\$b</code> 包含相同元素，返回 <code>true</code>
===	恒等	<code>\$a === \$b</code>	如果 <code>\$a</code> 和 <code>\$b</code> 包含相同顺序和类型的元素，返回 <code>true</code>
!=	不等价	<code>\$a != \$b</code>	如果 <code>\$a</code> 和 <code>\$b</code> 不包含相同元素，返回 <code>true</code>
<>	不等价	<code>\$a <> \$b</code>	与 <code>!=</code> 相同
!==	不恒等	<code>\$a !== \$b</code>	如果 <code>\$a</code> 和 <code>\$b</code> 不包含相同顺序类型的元素，返回 <code>true</code>

这些操作符通常都是非常直观的，但是联合需要进一步解释一下。联合操作符尝试将`$b`中的元素添加到`$a`的末尾。如果`$b`中的元素与`$a`中的一些元素具有相同的索引，它们将不会被添加。即`$a`中的元素将不会被覆盖。

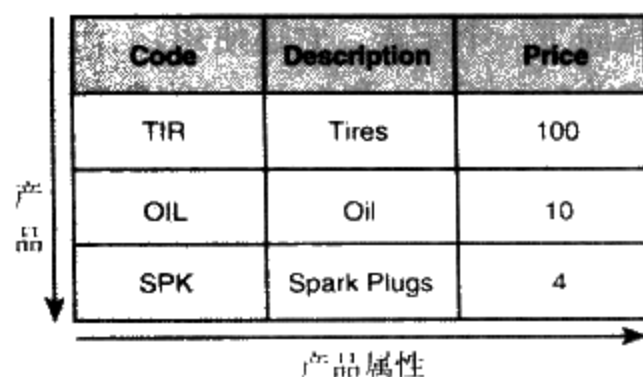
在表3-1中还可以看出，所有等价操作符都适用于标量变量。只要你记住了，`+`对标量类型执行加法操作，而联合对数组执行加法操作（即使你对集合算术不感兴趣）其行为还是明了的。通常，不能将数组与标量类型进行比较。

3.5 多维数组

数组不一定就是一个关键字和值的简单列表——数组中的每个位置还可以保存另一个数组。使用这种方法，可以创建一个二维数组。可以把二维组当成一个矩阵，或是一个具有宽度和高度或者行和列的网格。

如果希望保存Bob产品的多个数据，可以使用二维数组。图3-3用一个二维数组显示了Bob的产品，每一行代表一种产品，每一列代表一个产品属性。

使用PHP，可以编写如下代码来建立包含图3-3所示数据的数组：



	Code	Description	Price
产品	TIR	Tires	100
	OIL	Oil	10
	SPK	Spark Plugs	4

图3-3 可以用二维数组保存Bob产品的更多信息

```
$products = array( array( 'TIR', 'Tires', 100 ),
                    array( 'OIL', 'Oil', 10 ),
                    array( 'SPK', 'Spark Plugs', 4 ) );
```

可以从这个定义中看出\$products数组现在包含3个子数组。

回顾前面所介绍的，为了访问一个一维数组中的数据，需要使用数组的名称和元素的索引。除了一个元素具有两个索引——行和列外，二维数组和一维数组是类似的（最上边的是第0行，最左边的是第0列）。

要显示这个数组的内容，可以使用代码按顺序手动访问每个元素：

```
echo ' '. $products[0][0]. ' '. $products[0][1]. ' '. $products[0][2]. ' ' <br />';
echo ' '. $products[1][0]. ' '. $products[1][1]. ' '. $products[1][2]. ' ' <br />';
echo ' '. $products[2][0]. ' '. $products[2][1]. ' '. $products[2][2]. ' ' <br />';
```

此外，还可以使用双重for循环来实现同样的效果：

```
for ( $row = 0; $row < 3; $row++ ) {
    for ( $column = 0; $column < 3; $column++ ) {
        echo ' '. $products[$row][$column];
    }
    echo ' ' <br />';
}
```

以上两种代码都可以在浏览器中产生相同的输出，如下所示：

```
TIR|Tires|100|
OIL|Oil|10|
SPK|Spark Plugs|4|
```

这两个例子唯一的区别就是，如果对一个大数据组使用第二种代码，那么代码将简洁得多。

你可能更喜欢创建列名称来代替数字，如图3-3所示。要保存产品的相同集合，同时列名称为图3-3所给出的，可以使用如下所示的代码：

```
$products = array( array( 'Code'=> 'TIR',
                          'Description'=> 'Tires',
                          'Price'=> 100
                        ),
                    array( 'Code'=> 'OIL',
                          'Description'=> 'Oil',
                          'Price'=> 10
                        ),
                    array( 'Code'=> 'SPK',
                          'Description'=> 'Spark Plugs',
                          'Price'=> 4
                        )
                  );
```

如果希望检索单个值，那么使用这个数组会容易得多。请记住，将所描述的内容保存到用它的名称命名的列中，与将其保存到所谓的第一列中相比，前者更容易记忆。使用描述性索引，不需要记住某个元素是存放在[x][y]位置的。使用一对有意义的行和列的名称作为索引可以

使你很容易找到所需的数据。

然而，不能使用一个简单的for循环按顺序遍历每一列。使用如下代码可以显示这个数组内容：

```
for ( $row = 0; $row < 3; $row++){
    echo ' ' . $products[$row]['Code'] . ' ' . $products[$row]['Description'] .
        ' ' . $products[$row]['Price'] . ' ' <br />';
}
```

可以使用for循环遍历外部的数字索引数组\$products。\$products数组的每一行都是一个具有描述性索引的数组。在while循环中使用each()和list()函数，可以遍历整个内部数组。

因此，需要一个内嵌有while循环的for循环。

```
for ( $row = 0; $row < 3; $row++){
    while ( list( $key, $value ) = each( $products[$row] ) ){
        echo ' ' . $value . ' ';
    }
    echo ' ' <br />';
}
```

不必局限在二维数组上——按同样的思路，数组元素还可以包含新数组，这些新的数组又可以再包含新的数组。

三维数组具有高、宽、深的概念。如果能轻松地将一个二维数组想象成一个有行和列的表格，那么就可以将三维数组想象成一堆像这样的表格。每个元素可以通过层、行和列进行引用。

如果Bob要对他的产品进行分类，就可以使用一个三维数组来保存它们。图3-4显示了按三维数组方式保存的Bob产品。

通过这段定义了该数组的代码，可以发现三维数组是一个包含了数组的数组的数组。

```
$categories = array( array ( array( 'CAR_TIR', 'Tires', 100 ),
                                array( 'CAR_OIL', 'Oil', 10 ),
                                array( 'CAR_SPK', 'Spark Plugs', 4 )
                            ),
                    array ( array( 'VAN_TIR', 'Tires', 120 ),
                                array( 'VAN_OIL', 'Oil', 12 ),
                                array( 'VAN_SPK', 'Spark Plugs', 5 )
                            ),
                    array ( array( 'TRK_TIR', 'Tires', 150 ),
                                array( 'TRK_OIL', 'Oil', 15 )
                            )
                );
```

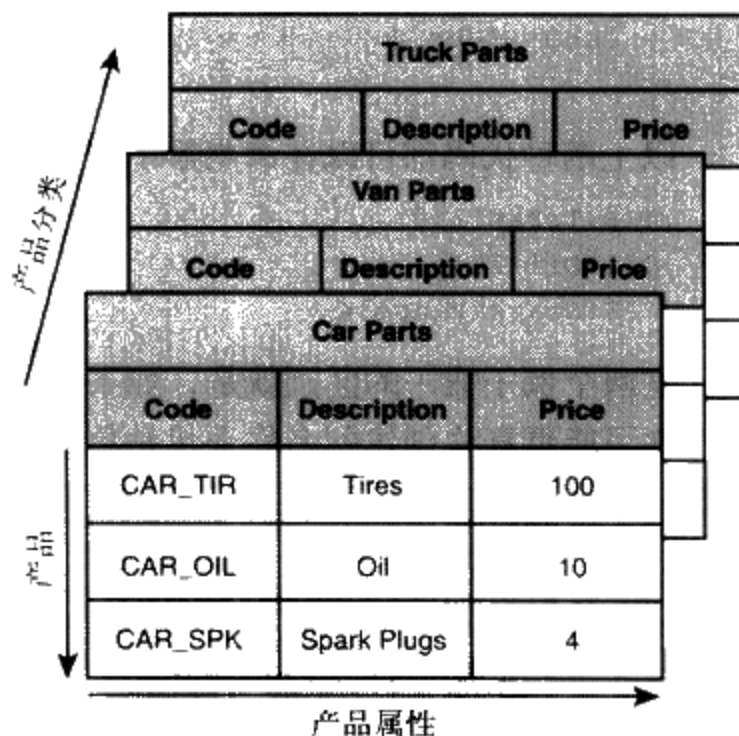


图3-4 三维数组允许你将产品分成不同的种类


```

        array( 'TRK_SPK', 'Spark Plugs' 6 )
    )
};

```

因为这个数组只有数字索引，可以使用嵌套的for循环来显示它的内容，如下所示：

```

for ($layer = 0; $layer < 3; $layer++) {
    echo "Layer $layer<br />";
    for ($row = 0; $row < 3; $row++) {
        for ($column = 0; $column < 3; $column++) {
            echo " | ".$categories[$layer][$row][$column];
        }
        echo " |<br />";
    }
}

```

根据创建多维数组的方法，可以创建四维、五维或六维数组。在PHP中，并没有设置数组维数的限制，但人们很难设想一个多于三维的数组。大多数的实际问题在逻辑上只需要使用三维或者更少维的数组结构就可以了。

3.6 数组排序

对保存在数组中的相关数据进行排序是一件非常有意义的事情。使用并且排序一个一维数组是非常简单的。

3.6.1 使用sort()函数

如下代码可以将数组按字母升序进行排序：

```

$products = array( 'Tires', 'Oil', 'Spark Plugs' );
sort($products);

```

现在，该数组所包含元素的顺序是：Oil、Spark Plugs、Tires。

还可以按数字顺序进行排序。如果具有一个包含了Bob产品价格的数组，就可以按数字升序进行排序，如下所示：

```

$prices = array( 100, 10, 4 );
sort($prices);

```

现在，产品价格的顺序将变成：4、10、100。

请注意，sort()函数是区分字母大小写的。所有大写字母都在小写字母的前面。所以A小于Z，而Z小于a。

该函数的第二个参数是可选的。这个可选参数可以传递SORT_REGULAR（默认值）、SORT_NUMERIC或SORT_STRING。指定排序类型的功能是非常有用的，例如，当要比较可能包含有数字2和12的字符串时。从数字角度看，2要小于12，但是作为字符串，'12'却要小于'2'。

3.6.2 使用asort()函数和ksort()函数对关联数组排序

如果用关联数组存储各个项目和它们的价格，就需要用不同的排序函数使关键字和值在排序时仍然保持一致。

如下所示的代码将创建一个包含3个产品及价格的数组，然后将它们按价格的升序进行排序：

```
$prices = array( 'Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4 );
asort($prices);
```

函数asort()根据数组的每个元素值进行排序。在这个数组中，元素值为价格而关键字为文字说明。如果不是按价格排序而按说明排序，就可以使用ksort()函数，它是按关键字排序而不是按值排序。这段代码会让数组的关键字按字母顺序排列——Oil、Spark Plugs、Tires：

```
$prices = array( 'Tires'=>100, 'Oil'=>10, 'Spark Plugs'=>4 );
ksort($prices);
```

3.6.3 反向排序

你已经了解了sort()、asort()和ksort()。这3个不同的排序函数都使数组按升序排序。它们每个都对应有一个反向排序的函数，可以将数组按降序排序。实现反向排序的函数是rsort()、arsort()和krsort()。

反向排序函数与排序函数的用法相同。函数rsort()将一个一维数字索引数组按降序排序。函数arsort()将一个一维关联数组按每个元素值的降序排序。函数krsort()将根据数组元素的关键字将一维数组按照降序排序。

3.7 多维数组的排序

对多于一维的数组进行排序，或者不按字母和数字的顺序进行排序，要复杂得多。PHP知道如何比较两个数字或字符串，但在多维数组中，每个元素都是一个数组。PHP不知道如何比较两个数组，所以需要建立一个比较它们的方法。在大多数情况下，单词和数字的顺序是显而易见的——但对于复杂的对象，问题就会多一些。

3.7.1 用户定义排序

这里有一个前面使用过的二维数组定义。这个数组存储了Bob的3种产品的代码、说明和价格：

```
$products = array( array( 'TIR', 'Tires', 100 ),
                   array( 'OIL', 'Oil', 10 ),
                   array( 'SPK', 'Spark Plugs', 4 ) );
```

如果对这个数组进行排序，最后的顺序会是怎样呢？因为我们知道各个数组内容所代表的意义，所以至少会有两种有用的排序方法。我们可能对产品的说明按字母排序，或者对价格按

大小排序。两种结果都有可能，但需要用函数`usort()`告诉PHP如何比较各个元素。要实现此功能，需要编写自己的比较函数。

如下所示的代码对订单数组中的第二列（说明），按字母进行排序：

```
function compare($x, $y) {
    if ($x[1] == $y[1]) {
        return 0;
    } else if ($x[1] < $y[1]) {
        return -1;
    } else {
        return 1;
    }
}

usort($products, 'compare');
```

到目前为止，在本书中们已经调用了许多PHP内置函数。为了对这个数组排序，必须定义了一个自己的函数。在第5章中，将详细介绍如何编写函数，但在此做一些简要的介绍。

我们用关键词`function`定义一个函数。需要给出函数的名称，而且该名称应该有意义，例如在这个例子中，函数被命名为`compare()`。许多函数都带有参数。`compare()`函数有两个参数：一个为`$x`，另一个为`$y`。该函数的作用是比较两个值的大小。

在这个例子中，`$x`和`$y`将是主数组中的两个子数组，分别代表一种产品。因为计数是从0开始的，说明字段是这个数组的第二个元素，所以为了访问数组`$x`的说明字段，需要输入`$x[1]`和`$y[1]`来比较两个传递给函数的数组的说明字段。

当一个函数结束的时候，它会给调用它的代码一个答复。该答复称为返回值。为了返回一个值，在函数中使用关键词`return`。例如，`return 1`；该语句将数值1返回给调用它的代码。

为了能够被`usort()`函数使用，`compare()`函数必须比较`$x`和`$y`。如果`$x`等于`$y`，该函数必须返回0，如果`$x`小于`$y`，该函数必须返回负数，而如果大于，则返回一个正数。根据`$x`和`$y`的值，该函数将返回0、1或-1。

以上代码的最后一行语句调用了内置函数`usort()`，该函数使用的参数分别是希望保存的数组（`$products`）和比较函数的名称（`compare()`）。

如果要想数组按另一种顺序存储，只要编写一个不同的比较函数。要按价格进行排序，就必须查看数组的第三列，从而创建如下所示的比较函数：

```
function compare($x, $y) {
    if ($x[2] == $y[2]) {
        return 0;
    } else if ($x[2] < $y[2]) {
        return -1;
    } else {
        return 1;
    }
}
```

当调用`usort($products, '$compare')`的时候，数组将按价格的升序来排序。

注意：当你通过运行这些代码来测试时，这些代码将不产生任何输出。这些代码只是将编写的大部分代码中的一部分。

`usort()` 中的“u”代表“user”，因为这个函数要求传入用户定义的比较函数。`asort` 和 `ksort` 对应的版本 `uasort()` 和 `uksort()` 也要求传入用户定义的比较函数。

类似于 `asort()`，当对非数字索引数组的值进行排序时，`uasort()` 才会被使用。如果值是简单的数字或文本则可以使用 `asort`。如果要比较的值像数组一样复杂，可以定义一个比较函数，然后使用 `uasort()`。

类似于 `ksort()`，当对非数字索引数组的关键字进行排序时才使用 `uksort()`。如果值是简单的数字或文本就使用 `ksort`。如果要比较的对象像数组一样复杂，可以定义一个比较函数，然后使用 `uksort()`。

3.7.2 反向用户排序

函数 `sort()`、`asort()` 和 `ksort()` 都分别对应一个带字母“r”的反向排序函数。用户定义的排序没有反向变体，但可以对一个多维数组进行反向排序。由于用户应该提供比较函数，因此可以编写一个能够返回相反值的比较函数。要进行反向排序，`$x` 小于 `$y` 时函数需要返回 1，`$x` 大于 `$y` 时函数需要返回 -1，这样就做成了一个反向排序。例如：

```
function reverse_compare($x, $y) {  
    if ($x[2] == $y[2]) {  
        return 0;  
    } else if ($x[2] < $y[2]) {  
        return 1;  
    } else {  
        return -1;  
    }  
}
```

调用 `usort($products, 'reverse_compare')`，数组会按价格的降序来排序。

3.8 对数组进行重新排序

在一些应用程序中，可能希望按另一种方式对数组排序。函数 `shuffle()` 将数组各元素进行随机排序。函数 `array_reverse()` 给出一个原来数组的反向排序。

3.8.1 使用 `shuffle()` 函数

Bob 想让其网站首页上的产品能够反映出公司的特色。他拥有许多产品，但希望能够从中随机地选出 3 种产品并显示在首页上。为了不至于让多次登录网站的访问者感到厌倦，他想让每次访问看到的 3 种产品都不同。如果将所有产品都存储在同一数组中，就很容易实现这个目标。程序清单 3-1 通过打乱数组并按随机顺序排列，然后从中选出前 3 种产品，显示这 3 种产品的图片。

程序清单3-1 bobs_front_page.php——使用PHP为Bob的汽车配件商店制作一个动态的首页

```

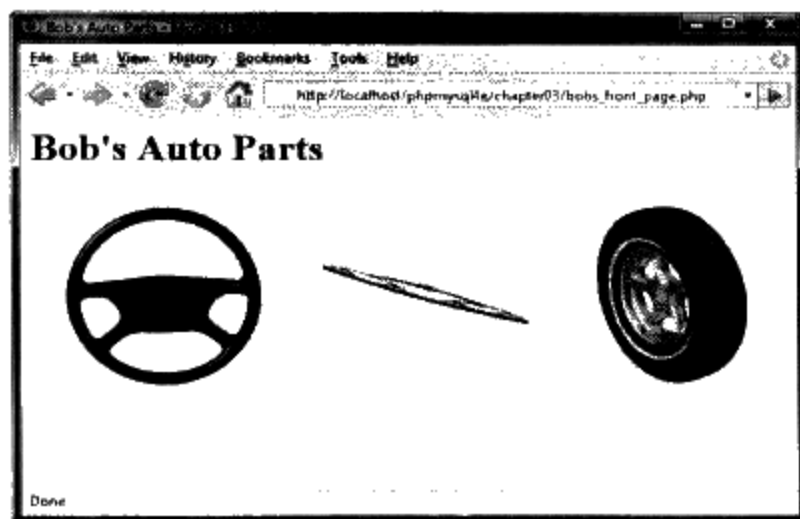
<?php
    $pictures = array('tire.jpg', 'oil.jpg', 'spark_plug.jpg',
                     'door.jpg', 'steering_wheel.jpg',
                     'thermostat.jpg', 'wiper_blade.jpg',
                     'gasket.jpg', 'brake_pad.jpg');

    shuffle($pictures);
?>
<html>
<head>
    <title>Bob's Auto Parts</title>
</head>
<body>

<h1>Bob's Auto Parts</h1>
<div align="center">
<table width = 100%>
<tr>
<?php
    for ($i = 0; $i < 3; $i++) {
        echo "<td align=\"center\"><img src=\"";
        echo $pictures[$i];
        echo "\"/></td>";
    }
?>
</tr>
</table>
</div>
</body>

```

因为以上代码将随机选择3个图片，所以每次登录并载入这个页面时，都会看到显示不同的页面，如图3-5所示。



选择产品

图3-5 shuffle() 允许随机选择3个产品

3.8.2 使用array_reverse()函数

array_reverse()函数使用一个数组作参数，返回一个内容与参数数组相同但顺序相反的数组。例如，可以使用很多方法创建一个按逆序包含数字10到1的数组。

因为单独使用range()函数将创建一个升序序列，所以必须使用rsort()函数或array_reverse()函数将数组中的数字变为降序。或者，也可以使用for循环通过一次一个元素的方式创建这个数组，如下所示：

```
$numbers = array();
for($i=10; $i>0; $i--) {
    array_push($numbers, $i);
}
```

一个for循环可以像这样按降序方式运行。可以将计数器的初始值设为一个大数，在每次循环末尾使用运算符“--”将计数器减1。

在这里，创建了一个空数组，然后使用array_push()函数将每个新元素添加到数组的末尾。请注意，和array_push()相反的函数是array_pop()，这个函数用来删除并返回数组末尾的一个元素。

或者，也可以使用array_reverse()函数将由range()函数所创建的数组进行反向排序。

```
$numbers = range(1,10);
$numbers = array_reverse($numbers);
```

请注意，array_reverse()函数将返回一个原数组修改后的副本。如果不再需要原来的数组，比如在这个例子中，可以用新的副本覆盖原来的版本。

如果数据只是一系列的整数，可以通过将-1作为range()函数的第三个可选步调参数，以相反的顺序创建该数组，如下所示：

```
$numbers = range(10, 1, -1);
```

3.9 从文件载入数组

第2章已经介绍了如何将客户的订单保存在一个文件中。文件中的每一行类似于如下所示：

```
15:42, 20th April 4 tires 1 oil 6 spark plugs $434.00 22 Short St, Smalltown
```

要处理或完成这个订单，就要将它重新载入数组中。程序清单3-2显示了当前的订单文件。

程序清单3-2 vieworders.php——使用PHP显示Bob的订单内容

```
<?php
//create short variable name
$DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];

$order = file("$DOCUMENT_ROOT/../orders/orders.txt");

$number_of_orders = count($order);
if ($number_of_orders == 0) {
```

```

    echo "<p><strong>No orders pending.  
    Please try again later.</strong></p>";
}

for ($i=0; $i<$number_of_orders; $i++) {
    echo $orders[$i]."<br />";
}

```

这个脚本的输出几乎和上一章中程序清单2-3的输出结果完全相同，如图2-4所示。这次，该脚本使用了file()函数将整个文件载入一个数组中。文件中的每行则成为数组中的一个元素。这段代码还使用了count()函数来统计数组中的元素个数。

此外，还可以将订单行中的每个区段载入到单独的数组元素中，从而可以分开处理每个区段或将它们更好地格式化。程序清单3-3很好地完成了这一功能。

程序清单3-3 vieworders2.php—用PHP分离、格式化并显示Bob的订单内容

```

<?php
    //create short variable name
    $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>
<html>
<head>
    <title>Bob's Auto Parts - Customer Orders</title>
</head>
<body>
<h1>Bob's Auto Parts</h1>
<h2>Customer Orders</h2>
<?php
    //Read in the entire file.
    //Each order becomes an element in the array
    $orders= file("$DOCUMENT_ROOT/./orders/orders.txt");

    // count the number of orders in the array
    $number_of_orders = count($orders);

    if ($number_of_orders == 0) {
        echo "<p><strong>No orders pending.  
        Please try again later.</strong></p>";
    }

    echo "<table border=\"1\">\n";
    echo "<tr><th bgcolor=\"#CCCCFF\">Order Date</th>  
        <th bgcolor=\"#CCCCFF\">Tires</th>  
        <th bgcolor=\"#CCCCFF\">Oil</th>  
        <th bgcolor=\"#CCCCFF\">Spark Plugs</th>  
        <th bgcolor=\"#CCCCFF\">Total</th>  
        <th bgcolor=\"#CCCCFF\">Address</th>  
        <tr>";

```

```

for ($i=0; $i<$number_of_orders; $i++) {
    //split up each line
    $line = explode("\t", $orders[$i]);

    // keep only the number of items ordered
    $line[1] = intval($line[1]);
    $line[2] = intval($line[2]);
    $line[3] = intval($line[3]);

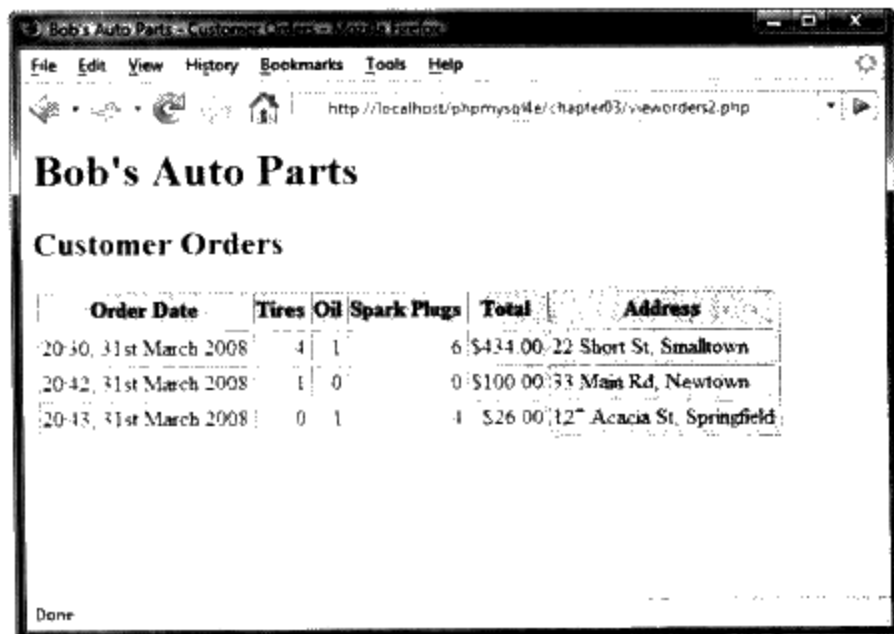
    // output each order
    echo "<tr>
        <td>". $line[0]. "</td>
        <td align='right'>". $line[1]. "</td>
        <td align='right'>". $line[2]. "</td>
        <td align='right'>". $line[3]. "</td>
        <td align='right'>". $line[4]. "</td>
        <td>". $line[5]. "</td>
    </tr>";

}

echo "</table>";
?>
</body>

```

程序清单3-3中的代码将整个文件载入数组中，但与程序清单3-2中的例子不同，在这里使用了`explode()`函数来分割每行，这样在开始打印前就可以再做一些处理与格式化。这个脚本输出结果如图3-6所示。



The screenshot shows a web browser window with the title 'Bob's Auto Parts - Customer Orders'. The address bar shows 'http://localhost/phpmysql4e/chapter03/vieworders2.php'. The page content includes the heading 'Bob's Auto Parts' and 'Customer Orders'. Below this is a table with the following data:

Order Date	Tires	Oil	Spark Plugs	Total	Address
20-30, 31st March 2008	4	1	6	\$434.00	22 Short St, Smalltown
20-42, 31st March 2008	1	0	0	\$100.00	33 Main Rd, Newtown
20-43, 31st March 2008	0	1	4	\$26.00	12 Acacia St, Springfield

图3-6 在用`explode()`函数分割订单记录后，可将订单的每个部分放入不同的表格单元以便美化输出效果

explode函数的原型如下所示:

```
array explode(string separator, string string |, int limit)
```

在前面一章中, 在保存数据的时候使用了制表符作为定界符, 因此, 将按如下方式调用:

```
explode( '\t', $orders[$i] )
```

这个函数可以将传入的字符串分割成一个个小块。每个制表符成为两个元素之间的断点。例如, 字符串:

```
"20:43, 31st March 2008\t0 tires\t1 oil \t4 spark plugs\t$26.00\t127 Acacia St, Springfield"
```

将被这个函数分割成“20:43, 31st March 2008”、“0 tires”、“1 oil”、“4 spark plugs”、“\$26.00”以及“127 Acacia St, Springfield”。

请注意, 这个函数的可选参数limit可以用来限制被返回的最大块数。

在这个例子中, 并没有做太多的处理。只是在一个有顶行说明的表格中显示了各种产品的数量以及一个能够显示数量代表意义的标题行, 而不是在每行中都输出tires、oil和spark plugs。

可以使用许多方法从字符串中提取数字。在这里, 使用了intval()函数。正如在第1章中所提过的, intval()函数可以将一个字符串转化成一个整数。这个转换是相当智能化的, 它可以忽略某些部分, 例如, 在这个例子中, 标签就不能转换成数字。在下一章中, 将详细介绍处理字符串的不同方法。

3.10 执行其他数组操作

到目前为止, 大概只介绍了一半的PHP数组处理函数。此外, 还有许多其他函数有时也非常有用。接下来, 将详细介绍它们。

3.10.1 在数组中浏览: each()、current()、reset()、end()、next()、pos()和prev()

前面已经提到, 每个数组都有一个内部指针指向数组中的当前元素。当使用函数each()时, 就间接地使用了这个指针, 但是也可以直接使用和操作这个指针。

如果创建一个新数组, 那么当前指针就将被初始化, 并指向数组的第一个元素。调用current(\$array_name)将返回第一个元素。

调用next()或each()将使指针前移一个元素。调用each(\$array_name)会在指针前移一个位置之前返回当前元素。next()函数则有些不同——调用next(\$array_name)是将指针前移, 然后再返回新的当前元素。

我们已经了解了reset()函数将返回指向数组第一个元素的指针。类似地, 调用end(\$array_name)可以将指针移到数组末尾。reset()和end()可以分别返回数组的第一个元素和最后一个元素。

要反向遍历一个数组, 可以使用end()和prev()函数。prev()函数和next()函数相反。它是将当前指针往回移一个位置然后再返回新的当前元素。

例如，如下所示的代码将反向显示一个数组的内容：

```
$value = end($array);
while ($value) {
    echo "$value<br />";
    $value = prev($array);
}
```

如果\$array数组的声明如下所示：

```
$array = array(1, 2, 3);
```

在这个例子中，浏览器中的输出结果就会是：

```
3
2
1
```

使用each()、current()、reset()、end()、next()、pos()和prev()，可以编写出你自己的、能按任何顺序浏览数组的代码。

3.10.2 对数组的每一个元素应用任何函数：array_walk()

有时候，读者可能希望以相同方式使用或者修改数组中的每一个元素。array_walk()函数允许进行这样的操作。函数array_walk()的原型如下所示：

```
bool array_walk(array arr, string func, [mixed userdata])
```

其调用方法类似于前面所介绍的usort()函数的调用，array_walk()函数要求声明一个你自己的函数。正如你所看到的，array_walk()函数需要三个参数。第一个是arr，也就是需要处理的数组。第二个是func，也就是用户自定义并将作用于数组中每个元素的函数。

第三个参数userdata是可选的，如果使用它，它可以作为一个参数传递给我们自己的函数。

在接下来的内容，读者将了解这个函数是如何工作的。

这个用户自定义函数可以是一个以指定格式显示各个元素的函数。如下所示的代码通过在\$array数组的每个元素中调用用户自定义的my_print()函数，从而将每个元素显示在一个新行中：

```
function my_print($value) {
    echo "$value<br />";
}
array_walk($array, 'my_print');
```

以上所编写的这个函数还需要有特定的符号。对于数组中的每个元素，array_walk将以关键字和保存在数组中的值为参数，此外，还可以以任何数据作为该函数的userdata参数。可以按如下方式调用函数：

```
your_function(value, key, userdata)
```

在大多数情况下，函数只能处理数组中的值。但是，在某些情况下，可能还需要使用userdata参数向函数传递一个参数。在少数情况下，可能还需要对数组关键字和值进行处理。

在MyPrint()函数中，可以忽略关键字参数和userdata参数。

在一个稍微复杂点的例子中，将编写一个带有一个参数的函数，这个函数可以用来修改数组的值。请注意，虽然对关键字并不感兴趣，但为了接收第三个参数变量，还是必须接收它：

```
function my_multiply(&$value, $key, $factor){
    $value *= $factor;
}
array_walk($array, 'my_multiply', 3);
```

在这里，定义了一个名为my_multiply()的函数，它可以用所提供的乘法因子去乘以数组中的每个元素。需要使用array_walk()函数的第三个参数来传递这个乘法因子。因为这个参数，所以在定义my_multiply()函数时必须带有三个参数——一个数组元素值(\$value)、一个数组元素的关键字(\$key)和参数(\$factor)。可以选择忽略这个关键字。

此外，还有一个需要注意的问题是传递参数\$value的方式。在my_multiply()的函数定义中，变量前面的地址符(&)意味着\$value是按引用方式传递的。按引用方式传递允许函数修改数组的内容。

我们将在第5章中详细介绍按引用方式的传递。如果你对这个术语还不太熟悉，那么现在就只需知道：为了使用引用传递，这里在变量名称前面加了一个地址符。

3.10.3 统计数组元素个数：count()、sizeof()和array_count_values()

在前面的例子中，已经使用函数count()对订单数组中的元素个数进行统计。函数sizeof()具有同样的用途。这两个函数都可以返回数组元素的个数。可以得到一个常规标量变量中的元素个数，如果传递给这个函数的数组是一个空数组，或者是一个没有经过设定的变量，返回的数组元素个数就是0。

函数array_count_values()更加复杂一些。如果调用array_count_values(\$array)，这个函数将会统计每个特定的值在数组\$array中出现过的次数（这就是数组的基数集）。这个函数将返回一个包含频率表的关联数组。这个数组包含数组\$array中的所有值，并以这些值作为关联数组的关键字。每个关键字所对应的数值就是关键字在数组\$array中出现的次数。

例如，如下代码：

```
$array = array(4, 5, 1, 2, 3, 1, 2, 1);
$sac = array_count_values($array);
```

将创建一个名为\$sac的数组，该数组包括：

关键字	值
4	1
5	1
1	3
2	2
3	1

其结果表示数值4、5、3在数组\$array中只出现一次，1出现了3次，2出现了两次。

3.10.4 将数组转换成标量变量：extract()

对于一个非数字索引数组，而该数组又有许多关键字-值对，可以使用函数extract()将它们转换成一系列的标量变量。extract()的函数原型如下所示：

```
extract(array var_array [, int extract_type [, string prefix]);
```

函数extract()的作用是通过一个数组创建一系列的标量变量，这些变量的名称必须是数组中关键字的名称，而变量值则是数组中的值。

如下所示的是一个简单的例子：

```
$array = array('key1' => 'value1', 'key2' => 'value2', 'key3' => 'value3');
extract($array);
echo "$key1 $key2 $key3";
```

这段代码的输出如下所示：

```
value1 value2 value3
```

这个数组具有3个元素，它们的关键字分别是：key1、key2和key3。使用函数extract()，可以创建3个标量变量\$key1、\$key2和\$key3。从输出结果中，可以看到\$key1、\$key2和\$key3的值分别为“value1”、“value2”和“value3”。这些值都来自原来的数组。extract()函数具有两个可选参数：extract_type和prefix。变量extract_type将告诉extract()函数如何处理冲突。有时可能已经存在了一个和数组关键字同名的变量，该函数的默认操作是覆盖已有的变量。表3-2给出了extract_type的可用值。

表3-2 extract_type的可用值

类 型	意 义
EXTR_OVERWRITE	当发生冲突时覆盖已有变量
EXTR_SKIP	当发生冲突时跳过一个元素
EXTR_PREFIX_SAME	当发生冲突时创建一个名为\$prefix_key的变量。必须提供prefix参数
EXTR_PREFIX_ALL	在所有变量名称之前加上由prefix参数的指定值。必须提供prefix参数
EXTR_PREFIX_INVALID	使用指定的prefix在可能无效的变量名称前加上前缀（例如，数字变量名称）。必须提供prefix参数
EXTR_IF_EXISTS	只提取已经存在的变量（也就是，用数组中的值覆盖已有的变量值）。这个参数对于数组到变量的转换时非常有用，例如，\$_REQUEST到一个有效的变量集合的转换
EXTR_PREFIX_IF_EXISTS	只有在不带前缀的变量已经存在的情况下，创建带有前缀的变量。这个值是在4.2.0版本中新增加的
EXTR_REFS	以引用方式提取变量

两个最常用的选项是EXTR_OVERWRITE（默认值）和EXTR_PREFIX_ALL。当知道会发生特定的冲突并且希望跳过该关键字或要给它加上前缀时，可能会用到其他选项。如下所示的是一个使用EXTR_PREFIX_ALL的简单例子。可以看到所有被创建的变量名称都具有前缀一下

画线-关键字名称的格式。

```
$array = array('key1' => 'value1', 'key2' => 'value2', 'key3' => 'value3');  
extract($array, EXTR_PREFIX_ALL, 'my_prefix');  
echo '$my_prefix_key1 $my_prefix_key2 $my_prefix_key3';
```

以上代码将再次输出：value1 value2 value3。

请注意，`extract()`可以提取出一个元素，该元素的关键字必须是一个有效的变量名称，这就意味着以数字开始或包含空格的关键字将被跳过。

3.11 进一步学习

本章介绍了PHP的数组函数中最有用的函数。没有介绍所有的数组函数。可以参阅PHP的在线手册 (<http://www.php.net/array>)，该手册给出了每一个数组函数的简单描述。

3.12 下一章

在下一章中，将介绍字符串处理的函数。将详细介绍搜索、替换、分割和合并字符串的函数，此外，还将介绍一些功能强大的正则表达式函数，这些函数几乎可以对字符串进行任何操作。