第3章 chapter 3

PHP 数组

3.1 什么是数组

生活中，一个笔记本电脑包只能放一台笔记本电脑，一个书包可以放多本书，一个抽屉可存放各种不同类型的物体。电脑包、书包和抽屉等都是可以容纳物体的容器。在PHP中，一个变量可以理解为只能存储单个值的容器，而一个数组可以理解为能存储多个值的容器。数组就像书包和抽屉一样，可以存储多个相同或不同类型的物体。数组是数据值的集合，是一个可以存储一组或一系列数值的变量。存储在数组中的值称为数组元素。一个数组可以具有许多个元素。PHP允许一个数组可以包括不同数据类型的元素，例如字符串、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，一个只包含值的数组称为一维数组。数组中的元素是在索引（也称下标）的基础上被引用和操作的。根据索引类型的不同，PHP数组可分为数字索引数组和关联数组。数字索引数组的索引是从0开始的整数，而关联数组的索引是字符串。实质上，在PHP内部所有的数组都被存储为关联数组，只是数字索引数组将索引特别指定到了数字。

注意：关联数组与数字索引数组唯一的区别在于索引的类型。与其他许多语言中的数组不同，PHP不需要在创建数组时指定数组的大小，甚至不需要在使用数组前先行声明。

3.2 数组的类型

3.2.1 数字索引数组

大多数编程语言都支持数字索引数组。数字索引数组是指数组元素的索引为整数的数组，索引从0开始，依次递增。当需要通过相对位置来确定数组元素时，通常使用数字索引数组。数字索引数组也称为枚举数组、标量数组或简单数组，表3-2-1是一个数字索引数组的示例。

表3-2-1　索引数组 $ name

数组元素	$ name[0]	$ name[1]	$ name[2]	$ name[3]	$ name[4]
元素值	张明	李四	王小五	赵勤	李小平

表中有一个索引数组，数组名为 $name，数组含有 5 个数组元素，这 5 个数组元素的值是“张明”、“李四”、“王小五”、“赵勤”和“李小平”。可以通过索引（也称下标）0 来引用“张明”，通过索引 1 来引用“李四”，通过索引 2 来引用“王小五”，依此类推。要引用 PHP 中的元素，需要使用一对中括号（[]），例如显示索引 2 的数组元素值的语句如下：

```
echo $name[2]
```

注意：PHP 的数字索引数组默认从 0 开始，而不是 1，这与 C、C++ 和 Java 等许多编程语言相同，数组最后一个索引号等于数组元素的个数减 1。

3.2.2 关联数组

数字索引数组在各种编程语言中非常常见。但对于关联数组，只有使用过 PHP 或 Perl 的程序员才不会感到陌生。关联数组允许数组中每个元素除了可以使用数字索引外，还可以使用字符串作为索引。如果需要通过名称访问数组中的元素，那么使用字符串作为数组的索引将是非常方便的。使用字符串作为索引的数组即称为关联数组。表 3-2-2 是一个关联数组的示例。

表 3-2-2 关联数组 $name

数组元素	$name["user1"]	$name["user2"]	$name["user3"]	$name["user4"]	$name["user5"]
元素值	张明	李四	王小五	赵勤	李小平

表中有一个关联数组，数组名为 $name，数组含有 5 个数组元素，这 5 个数组元素的值是“张明”、“李四”、“王小五”、“赵勤”和“李小平”。其索引值分别为字符串 userl、user2、user3、user4 和 user5。通过字符串索引值可以访问数组元素，比如显示索引为 user2 的数组元素的语句如下：

```
echo $name[user2]
```

注意：使用数字索引数组和关联数组的区别之一在于关联数组以字符串为索引，不能通过简单的加 1 或减 1 直接计算出数组中的下一个或前一个的有效索引值。

3.3 数组的创建和初始化

3.3.1 一维索引数组

在 PHP 中声明数组的方式主要有两种：一是应用函数 array 声明数组；一是直接为数组元素赋值。函数 array 用来新建一个数组。它接受一定数量用逗号分隔的 key => value 参数对。使用 array 函数创建数组的格式如下：

```
array([key=>] value,...)
```

key 是索引，又叫键值或者下标，它可以是数字或者字符串，value 可以是任何值。下

例是使用 array 创建下标为字符串的数组范例。

【例 3-3-1】（代码位置：\3\testIndexArray1.php）

```
<?php
  $myarray=array("1"=>"编","2"=>"程","3"=>"词","4"=>"典");
  print_r($myarray)
?>
```

程序运行后浏览器输出如下：

```
Array([1]=>编 [2]=>程 [3]=>词 [4]=>典)
```

上例使用函数 array 创建一个包含 4 个元素的数组，元素的下标分别为 1、2、3 和 4，同时，该函数还给元素赋初值，初值分别是“编”、“程”、“词”和“典”。下面是创建下标为整数的数组范例。

【例 3-3-2】（代码位置：\3\testIndexArray2.php）

```
<?php
  $phpjc=array(
    0=>'word',                    //指定下标 0
    3=>'excel',                   //指定非连续的下标 3
    'outlook',                    //未指定下标
    'access');                    //创建一个数组,并初始化数组元素
  print_r($phpjc);
?>
```

输出结果如下：

```
Array([0]=>word [3]=>excel [4]=>outlook [5]=>access)
```

上例定义了一个数组，名字叫 phpjc，第一个元素（即 $phpjc[0]）的值是 word，第二个元素（即 $phpjc[3]）值是 excel、outlook 和 access 分别是第三和第四个元素的值。值得注意的是，此处数组元素的下标并不是连续的，$phpjc[1]和 $phpjc[2]并未定义。如果数组元素不指定下标，则按递增的方式生成下标。此处尽管未指定下标 4 和 5，但第三个元素 $phpjc[4]和第四个元素 $phpjc[5]的下标会在第二个元素 $phpjc[3]的下标 3 基础上依次递增。

如果在创建数组时不知所创建数组的大小，或在实际编写程序时数组的大小可能发生变化，可以通过给变量赋予一个没有参数的 array()来创建空数组，然后可以通过使用方括号[]语法来添加值。下面是直接为数组元素赋值的范例。

【例 3-3-3】（代码位置：\3\testIndexArray3.php）

```
<?php
  $phpjc=array();                 //创建一个数组
  $phpjc[0]="one";                //直接为数组元素赋值
  $phpjc[1]="two";                //直接为数组元素赋值
  echo $phpjc[0]."<br>";
```

```
  echo $phpjc[1];
?>
```

上例先创建了一个数组，数组名为 phpjc，然后通过赋值运算符给数组元素赋值，最后输出数组元素的值。运行后浏览器输出结果如下：

```
one
two
```

下面是使用赋值的方式初始化数字索引数组的例子：

```
$name=array();
$name[0]="张明";
$name[1]="李四";
$name[2]="王小五";
$name[3]="赵勤";
Sname[4]="李小平";
```

当索引值是递增的，可以不在方括号内指定索引值，默认的索引值从 0 开始依次增加，即为 0、1、2、3 和 4 等，具体代码如下：

```
$name=array();
$name[]="张明";
$name[]="李四";
$name[]="王小五";
$name[]="赵勤";
Sname[]="李小平";
```

这种不显式地指定索引值的赋值方法，也称简单赋值方法，在初始化索引值连续递增的数组时非常简便。对于非连续索引的数组，对其初始化可以直接指定索引值，具体代码如下：

```
$name=array();
$name[0]="张明";
$name[10]="李四";
$name[20]="王小五";
$name[30]="赵勤";
$name[40]="李小平";
```

其中数组 name 有 5 个数组元素，其索引值分别为 0、10、20、30 和 40。它们对应的元素值分别为“张明”、“李四”、“王小五”、“赵勤”和“李小平”。

3.3.2 一维关联数组

关联数组的初始化可以采用与数字索引数组类似的方法，通过直接为数组元素赋值来完成。下面是以赋值的形式初始化关联数组的范例。

【例 3-3-4】（代码位置：\3\testAssociateArray1.php）

```
<?php
  $name=array();
  $name['userl']="张明";
  $name['user2']="李四";
  $name['user3']="王小五";
  $name['user4']="赵勤";
  $name['user5']="李小平";
  print_r($name);
?>
```

程序运行输出结果如下：

```
Array([userl]=>张明 [user2]=>李四 [user3]=>王小五 [user4]=>赵勤 [user5]=>李小平)
```

使用 array 函数来完成关联数组的初始化也是常用的一种方法，其中索引值与数组元素值之间需要使用运算符“=>”。下面是使用 array 初始化数组 $score 的范例。

【例 3-3-5】（代码位置：\3\testAssociateArray2.php）

```
<?php
  $score=array("张明"=>80,
      "李亮"=>90,
      "王平"=>67,
      "张晓红"=>77,
      "赵高"=>87);
  print_r($score);
?>
```

程序输出结果如下：

```
Array([张明]=>80 [李亮]=>90 [王平]=>67 [张晓红]=>77 [赵高]=>87)
```

上例程序等同于如下代码。

【例 3-3-6】（代码位置：\3\testAssociateArray3.php）

```
<?php
  $score=array();
  $score["张明"]=80;
  $score["李亮"]=90;
  $score["王平"]=67;
  $score["张晓红"]=77;
  $score["赵高"]=87;
  print_r($score);
?>
```

3.3.3 多维数组

如果数组 A 的元素仍为数组，那么数组 A 就构成了多维数组。一个包含数组的数

组称为二维数组，如果二维数组中的数组元素仍包含数组，就构成了一个三维数组，依此类推，可以创建四维数组、五维数组等多维数组。但最常见的是二维数组。可以这样理解多维数组，一个大容器里含有很多小容器，小容器里还可以再含有很多更小的容器，依此类推。为简单起见，多维数组主要介绍二维数组。可以把二维数组想像成一个矩阵，或者是一个具有宽度和高度或者行和列的表格。

下面是二维数组的范例。表 3-3-1 是成绩表，该成绩表含有 3 条记录，其中每一行表示一个学生记录，每一列分别代表学生的“姓名”、“学号”、“性别”和“成绩”。例中使用一个 3×4 的二维数组 info 表示该成绩表，3 表示数组 info 有 3 个元素，4 表示数组 info 中的每个元素都是一个含有 4 个元素的数组。

表 3-3-1 成绩表

姓 名	学 号	性 别	成 绩
张三	A01	男	90
李四	A02	男	12
老五	B02	男	80

【例 3-3-7】 （代码位置：\3\testMDimArray.php）

```
<?php
  $info=array(array("张三","A01","男","90"),
              array("李四","A02","男","12"),
              array("老五","B02","男","95"));
  print_r($info);
?>
```

程序输出结果如下：

```
Array([0]=>Array([0]=>张三 [1]=>A01 [2]=>男 [3]=>90 )[1]=>Array([0]=>李四
[1]=>A02 [2]=>男 [3]=>12 )[2]=>Array([0]=>老五 [1]=>B02 [2]=>男 [3]=>95 ))
```

数组 info 第一个元素：[0]=>Array([0]=>张三[1]=>A01[2]=>男[3]=>90)。
数组 info 第二个元素：[1]=>Array([0]=>李四[1]=>A02[2]=>男[3]=>12)。
数组 info 第三个元素：[2]=>Array([0]=>老五[1]=>B02[2]=>男[3]=>95)。
上例程序等同于下面的代码。

【例 3-3-8】 （代码位置：\3\testMDimArray1.php）

```
<?php
  $info=array();
  $info[0]=array("张三","A01","男","90");
  $info[1]=array("李四","A02","男","12");
  $info[2]=array("老五","B01","男","90");
  print_r($info);
?>
```

3.4 数组的遍历

3.4.1 一维数组的遍历

一般情况下，遍历一个数组有 for、while 和 foreach 三种方法。其中最简单方便的是 foreach。对于遍历同样一个数组，foreach 速度最快，最慢的则是 while。从原理上来看，foreach 是对数组副本进行操作（通过复制数组），而 while 则通过移动数组内部指标进行操作，按照一般逻辑会认为 while 应该比 foreach 快，因为 foreach 在开始执行的时候首先把数组复制进去，而 while 直接移动内部指标。但结果刚刚相反，原因是 foreach 是 PHP 内部实现循环而 while 是通用的循环结构。在应用中更常用 foreach 遍历数组。在 PHP 5 中，foreach 还可以遍历类的属性。使用 foreach 的不足是只能顺序读取数组元素而不能随机读取。使用 for 语句可以随机读取数组元素，但需要计算数组长度，并且不能读取键。使用 while 语句通常需要其他函数配合。下面是使用 foreach 遍历数组的范例。

【例 3-4-1】 （代码位置：\3\testArrayForeachTravel1.php）

```
<?php
  $colors=array('red','blue','green','yellow');
  foreach($colors as $color){
    echo "Do you like $color? <br />";
  }
?>
```

程序每次遍历数组 colors 时，数组中的每个元素都用变量 $color 表示，运行结果显示如下：

```
Do you like red?
Do you like blue?
Do you like green?
Do you like yellow?
```

上例遍历数组时并不能知道数组元素的下标。下面是使用 foreach 的另一种形式遍历数组的范例，该范例可以知道每个元素的下标。

【例 3-4-2】 （代码位置：\3\testArrayForeachTravel2.php）

```
<?php
  $colors=array('red','blue','green','yellow');
  foreach($colors as $key=>$color){
    echo "$key :Do you like $color? <br />";
}
?>
```

程序每次遍历数组 colors 时，数组元素的键用变量 $key 表示，元素的值用变量 $color 表示，运行结果显示如下：

```
0 :Do you like red?
1 :Do you like blue?
2 :Do you like green?
3 :Do you like yellow?
```

下面是使用 for 遍历数组的范例。

【例 3-4-3】（代码位置：\3\testArrayForTravel.php）

```
<?php
$arr=array("0"=>"zero","1"=>"one","2"=>"two");
for($i=0;$i <count($arr);$i++){
    $str=$arr[$i];
    echo "the number is $str.<br />";
}
?>
```

例中需要使用函数 count 计算数组 $arr 的长度，程序运行结果如下：

```
the number is zero.
the number is one.
the number is two.
```

下例是使用 while 遍历数组的范例，其需要使用函数 each 和 list 配合 while 才能遍历数组。

【例 3-4-4】（代码位置：\3\testArrayWhileTravel.php）

```
<?php
  $colors=array('red','blue','green','yellow');
  while(list($key,$val)=each($colors)){
    echo "Other list of $val.<br />";
  }
?>
```

例中每次循环会通过函数 each 取出数组 $colors 中的下一个元素，函数 list 的作用是把数组元素的键值和值分别赋给变量 $key 和 $val。程序运行结果如下：

```
Other list of red.
Other list of blue.
Other list of green.
Other list of yellow.
```

3.4.2 多维数组的遍历

多维数组和一维数组的遍历类似，下面是多维数组遍历的范例。

【例 3-4-5】（代码位置：\3\testMDimArrayForeachTravel1.php）

```
<?PHP
  //定义一个多维数组
```

```
  $more=array(
    "numbers"=>array(1,2,3,4,5,6,7),
    "colors"=>array("红","蓝","绿","黄","紫","青","橙")
  );
  //遍历一个多维数组
  foreach($more as $MYARRAY){
    foreach($MYARRAY as $key=>$value){
      echo $value;
    }
  }
?>
```

程序运行结果如下：

```
1234567红蓝绿黄紫青橙
```

值得注意的是，用这种方法遍历多维数组时，多维数组的低维元素应该是数组而不是值。如果多维数组中的元素含有数组和数值，则可以通过 is_array 函数判断元素是否是数组，并根据情况进行深度遍历，下面给出相应范例。

【例 3-4-6】（代码位置：\3\testMDimArrayForeachTravel2.php）

```
<?PHP
  //定义一个多维数组
  $more=array(
    "numbers"=>10,
    "colors"=>array("红","蓝","绿","黄","紫","青","橙")
  );
  //遍历一个多维数组
  foreach($more as $MYARRAY){
    if(is_array($MYARRAY))
      foreach($MYARRAY as $key=>$value){
        echo $value;
      }
    else echo $MYARRAY;
  }
?>
```

程序运行结果如下：

```
10红蓝绿黄紫青橙
```

3.5 数组函数库

PHP 提供专门用于数组的函数，这些函数作为 PHP 核心的一部分，无须安装即可使用。这些函数允许用多种方法来操作数组和与之交互。数组的本质是储存、管理和操作

一组变量。下面介绍排序、逆排序和删除等函数，其他函数请参看其他相关资料。

3.5.1 排序

可用函数 sort 对数组中的元素进行排序。其格式如下：

```
void sort (array array[,int sort_flags])
```

参数 array 表示要排序的数组，可选的第二个参数 sort_flags 可以用以下值改变排序的行为，其值和含义如下：

(1) 值 SORT_REGULAR 表示正常比较数组元素，即以数组元素原来的数据类型进行比较，其为默认值。

(2) 值 SORT_NUMERIC 表示数组元素被作为数字来比较。

(3) 值 SORT_STRING 表示数组元素被作为字符串来比较。

下面是数组排序的范例。

【例 3-5-1】 (代码位置：\3\testArraySort.php)

```
<?php
  $colors=array();
  $colors[10]='red';
  $colors[11]='blue';
  $colors[12]='green';
  $colors[13]='yellow';
  sort($colors);
  foreach($colors as $key=>$color){
      echo "Do you like $color? <br />";
  }
?>
```

例中以正常的方式排序数组 $colors 中的元素，程序运行结果如下：

```
Do you like blue?
Do you like green?
Do you like red?
Do you like yellow?
```

3.5.2 逆排序

使用 rsort 函数进行逆排序，下面是函数 rsort 进行逆排序的范例。

【例 3-5-2】 (代码位置：\3\testArrayRSort.php)

```
<?php
  $colors=array();
  $colors[10]='red';
  $colors[11]='blue';
```

```
  $colors[12]='green';
  $colors[13]='yellow';
  rsort($colors);
  foreach($colors as $key=>$color){
    echo "Do you like $color? <br />";
  }
?>
```

程序运行结果如下：

```
Do you like yellow?
Do you like red?
Do you like green?
Do you like blue?
```

3.5.3 打乱数组排序

函数 shuffle 用来打乱一个数组(即随机排列数组元素的顺序)。其格式如下：

```
void shuffle (array array)
```

下面是使用函数 shuffle 打乱数组排序的范例。

【例 3-5-3】 (代码位置：\3\testArrayShuffle.php)

```
<?php
  $colors=array();
  $colors[10]='red';
  $colors[11]='blue';
  $colors[12]='green';
  $colors[13]='yellow';
  shuffle($colors);
  foreach($colors as $key=>$color){
    echo "Do you like $color? <br />";
  }
?>
```

上例每次运行数组 $colors 的元素次序将发生变化，其输出结果都会不同。

3.5.4 删除数组元素

函数 array_splice 的作用是把数组中的一部分去掉并用其他值取代。其格式如下：

```
array array_splice ( array input,int offset [,int length [,array replacement]])
```

参数 input 表示要删除的数组，参数 offset 和 length 用来指定要删除元素的起始位置和个数。参数 input 和 offset 是必选的，length 是可选的。如果提供了 replacement 参

数，则用数组 replacement 中的元素取代数组 input 中被删除的元素。该函数会返回一个包含有被移除元素的数组。如果 offset 为正，则从数组 input 中该值指定的偏移量开始移除。如果 offset 为负，则从数组 input 末尾倒数该值指定的偏移量开始移除。如果省略参数 length，则移除数组中从 offset 到结尾的所有部分。如果 length 为正值，则移除 length 个元素。如果 length 为负值，则移除从 offset 到数组末尾倒数 length 为止中间所有的元素。

下面是删除指定位置的一个数组元素的范例。

【例 3-5-4】（代码位置：\3\testArraySplice1.php）

```
<?php
  $colors=array();
  $colors[10]='red';
  $colors[11]='blue';
  $colors[12]='green';
  $colors[13]='yellow';
  array_splice($colors,2,1);          //删除下标为 2 的元素
  foreach($colors as $key=>$color){
    echo "Do you like $color? <br />";
  }
?>
```

例中指定要删除数组 $colors 中的元素，从下标为 2 的元素开始删除，删除元素的个数是 1。程序运行结果如下：

```
Do you like red?
Do you like blue?
Do you like yellow?
```

下面的范例从数组指定位置删除之后的所有数组元素。

【例 3-5-5】（代码位置：\3\testArraySplice2.php）

```
<?php
  $colors=array();
  $colors[10]='red';
  $colors[11]='blue';
  $colors[12]='green';
  $colors[13]='yellow';
  array_splice($colors,2);            //从下标 2 开始删除之后的元素
  foreach($colors as $key=>$color){
      echo "Do you like $color? <br />";
  }
?>
```

例中将从数组 $colors 中的第二个元素开始删除，由于未指定删除元素个数，因此默认删除其后的所有元素，程序将删除元素 $colors[12]和 $colors[13]。程序运行结果

如下：

```
Do you like red?
Do you like blue?
```

下面的范例反向删除数组元素。

【例 3-5-6】（代码位置：\3\testAraySplice3.php）

```
<?php
  $colors=array();
  $colors[10]='red';
  $colors[11]='blue';
  $colors[12]='green';
  $colors[13]='yellow';
  array_splice($colors,-2);          //从数组最后一个元素反向删除 2 个元素
  foreach($colors as $key=>$color){
    echo "Do you like $color? <br />";
  }
?>
```

例中从数组 $colors 倒数第二个元素开始删除元素，由于未指定删除元素的个数，因此默认将删除其后面所有的元素，程序将删除元素 $colors[12]和 $colors[13]。程序运行结果如下：

```
Do you like red?
Do you like blue?
```

第4章 chapter 4

PHP 函数

编程过程通常会借用函数实现某种功能。一个函数的作用就相当于一台机器，不同的机器其作用会各不相同，就像放玉米进机器中，它加工出来的是爆米花。不同的函数能完成不同的特定的功能。PHP 函数(function)是指一段完成指定任务的已命名代码，函数可以遵照给它的一组值或参数完成任务，并且可能返回一个值。使用函数可以节省编译时间，因为无论调用多少次，函数只需被编译一次。当程序代码多了以后，如何组织这些程序？PHP 最初的设计原则是用函数来组织，这样可以让一段代码形成一个“程序模块”，不管在什么地方使用到相同功能时都可调用该函数，省去了重复编写代码的麻烦。函数可以把不同功能的代码独立到一处，减少耦合性，提高代码可重用性和增强程序的安全性。此外，函数还可以把代码和其他已实现的函数整合起来实现功能更加强大的函数。PHP 函数可分用户自定义函数和系统内置函数。PHP 中一个函数由 4 部分组成：函数名、参数、函数体和返回值。

4.1 用户自定义函数的定义

PHP 在提供大量内置函数的基础上，还开放了自定义函数的功能，允许开发者按照功能需求来编写实现特定目的的自定义函数。用户自定义函数的语法如下：

```
function functionname($args)
{
    表达式
}
```

function 为函数声明时的关键字；functionname 是要声明的函数的名称；参数 $args 表示外部传递给函数的参数变量，参数 $args 是可选的，而且可以是多个参数；表达式为函数要实现的功能主体，任何符合 PHP 语法的语句都可以作为表达式。

函数名是函数在程序代码中的识别名称，函数名可以是以字母或下划线开头后跟零个或多个字母、下划线和数字的任何字符串。函数名不区分大小写。给函数命名时不可使用已声明的函数名称或 PHP 内建的函数名称。函数外部可以通过参数 $args 把数据传递给函数体中的表达式加以运算处理，多个参数之间要用逗号隔开。当函数不需要任

何数值传入时，可以省略参数。下面是自定义函数的范例。

```
<?PHP
  function myfun($name)
  {
    echo $name;
    echo "hello";
    return "$name : hello";
  }
?>
```

上例中，function 是定义函数的关键字，myfun 是函数名，$name 是函数参数，花括号中的代码是函数体，return 表示函数有返回值。

4.2 函数的调用

用户定义好函数后，就可以在程序中直接调用该函数了，函数的调用方法很简单，格式如下：

```
functionname($args)
```

functionname 为要调用的函数名称，该函数需要已定义好，参数 $args 是传递给函数的值。调用函数时，直接用一个函数的函数名加一对半角括号即可。如果函数需要参数，则要在括号中给出函数所需要的参数表。在 PHP 中，用户自定义函数可以写在被调用位置的前面，也可以写在被调用位置的后面。这种调用方式也适用于调用系统内置函数。下面是调用函数的实例，在该例中，myfun 和 sayHello 是自定义函数，echo 是系统定义的函数。

【例 4-2-1】 （代码位置：\4\testCallFun.php）

```
<?PHP
  function myfun()
  {
      echo "hello";
  }
  myfun();              //调用自定义的函数 myfun
  sayHello();           //调用自定义的函数 sayHello
  function sayHello()
  {
      echo "Hello";
  }
?>
```

例中函数 myfun 的定义写在被调用位置的前面，而函数 sayHello 的定义写在被调用位置的后面。程序运行后浏览器输出：helloHello。

4.3 函数参数

有时主程序需要动态传递不同的值给函数处理，通过参数列表可以传递信息到函数。PHP 中传递参数的方式有按值传递参数和按址传递参数。按值传递参数是指父程序直接传递指定的值或变量给函数使用。由于所传递的值或变量与函数里的数值分别储存于不同的内存区块，所以当函数对所导入的数值作了任何变动时，并不会对父程序造成直接影响。这就好比用户 A 把资料的复印件给用户 B 使用，资料的原件和复印件互相独立，复印件（或原件）的损坏并不影响原件（或复印件），两者互不影响。相对于按值传递模式，按址传递参数并不会将父程序中的指定数值或目标变量传递给函数，而是把该数值或变量的内存储存区块相对地址导入函数之中。因此当该数值在函数中有任何变动时，会连带对父程序造成影响。下面是按值传递参数的范例。

【例 4-3-1】（代码位置：\4\testFunVParam.php）

```
<?PHP
  sayHello("小王");
  function sayHello($name)
  {
      echo "$name,Hello";
  }
?>
```

程序运行结果如下：

```
小王,Hello
```

下例是按值传递数组参数的范例。

【例 4-3-2】（代码位置：\4\testFunArrayParam.php）

```
<?php
  function takes_array($input)
  {
    echo "$input[0] +$input[1]=", $input[0]+$input[1];
  }
  $myvals=array(10,20);
  takes_array($myvals);
?>
```

程序运行结果如下：

```
10+20=30
```

下面是按址传递参数的范例，函数 print_A 有两个参数变量 $A 和 $B，前者是按值传递，后者是按地址传递，函数 print_A 的功能是改变两者的值并打印其值，由于参数 $B 是按地址传递，而参数 $A 是按值传递，所以主程序中的变量 $B 也被改变，而 $A

没有被改变。

【例 4-3-3】（代码位置：\4\testFunRefParam.php）

```
<?php
    $A="Today";
    $B="Monday";
    function print_A($A,&$B)
    {
        $A="今天";                    //修改变量$A的值
        $B=$A." is ".$B;              //修改变量$B的值
        echo "函数中变量 A 与变量 B 的值为<br>";
        echo "变量 A: $A <br>";
        echo "变量 B: $B <p>";
    }
                                      //以传址方式导入变量 B
    print_A($A,$B);
    echo "主程序中变量 A 与变量 B 的值为<br>";
    echo "变量 A: $A <br>";
    echo "变量 B: $B <p>";
?>
```

程序运行结果如下所示，从结果中可知，变量$A没有被改变，而变量$B被改变了。

```
函数中变量 A 与变量 B 的值为
变量 A: 今天
变量 B: 今天 is Monday

主程序中变量 A 与变量 B 的值为
变量 A: Today
变量 B: 今天 is Monday
```

如果在调用函数时没有指定参数的值，在函数中会使用参数的默认值。默认参数必须列在所有没有默认值参数的后面。下面是默认值传递参数的范例，函数 fun_sum 中的变量$b和$c默认值是0，它们必须写在参数$a的后面，程序运行输出30和60。

【例 4-3-4】（代码位置：\4\testSTParam.php）

```
<?php
  function fun_sum($a,$b=0,$c=0)
  {
    return $a+$b+$c;
  }
  echo fun_sum(10,20);          //$b的值是20,而变量$c的是0
  echo "<br/>";
  echo fun_sum(10,20,30);       //变量$b和$c值分别是20和30,而不是0
?>
```

4.4 函数返回值

外部程序通过参数变量传递数据给函数，函数处理数据后会返回结果给外部程序使用，外部程序和函数体之间就是通过参数变量和返回值来交互的。函数运行的结果是通过返回值反馈给父程序的。如果在一个函数中调用 return 语句，则立即结束此函数的执行并将 return 的参数作为函数的值返回。函数返回值用法如下：

```
function functionname(变量)
{
    表达式;
    return 变量;
}
```

下面是关于函数返回值的范例。

【例 4-4-1】（代码位置：\4\testFunRet.php）

```
<?php
    echo "用函数求累加：<br>";
    function test($a)
    {
        $sum=0;
        for($i=0;$i<=$a;$i++)
            $sum+=$i;
        return $sum;              //也可写成 return($sum)
    }
    echo "5 的累加是：".test(5)."<br>";
?>
```

上例函数 test 会累加 0 至 $a 的和，并把累加和作为返回值返回给主程序使用。程序运行结果如下：

```
用函数求累加：
5 的累加是：15
```

注意：return 是语言结构而不是函数，并不需要用括号将参数括起来。事实上不用括号比用括号更常见，其实用哪一种并无所谓。如果函数中没有 return 则函数结束时不会有返回值。函数也可以带有 return 而不返回任何值，下面是无返回值的范例。

【例 4-4-2】（代码位置：\4\testFunVoidRet.php）

```
<?php
    function test()
    {
        echo "hello";
        return ;                  //无返回值
```

```
    }
    test();
?>
```

关于return还有其他值得注意的地方，它并不局限于只用在函数中，还可以使用在其他地方。如果在全局范围中调用，则当前脚本文件中止运行。如果当前脚本文件是被include的或者require的，则控制交回调用文件。此外，如果当前脚本是被include的，则return的值会被当作include调用的返回值。如果在主脚本文件中调用return，则脚本中止运行。

4.5 内置函数

PHP内置函数是指PHP预提供的函数，不需要用户定义即可使用。内置函数可分为标准函数和扩展函数。标准函数是指编译到PHP内核中的函数，可以直接在程序中使用，不需要其他任何声明和加载库工作。扩展函数按照功能的不同被分门别类地封装在多个函数库中，扩展函数一般并不能在代码中直接使用，而是要按照程序需求有选择地开启和关闭。打开一个扩展函数库的方法很简单，只需打开PHP安装目录下的文件php-apache2handler.ini，将文件中“;extension=php_xxx.dll”前面的分号“;”去掉，并且保存php-apache2handler.ini和重新启动Apache Web服务器即可。常用的数据库和SOAP等扩展函数库都可通过这种方法启动和关闭。本节主要介绍标准函数。

4.5.1 die函数

函数die用来输出一条消息，并退出当前脚本。die的语法如下：

```
die(status)
```

如果参数status是字符串，则该函数会在退出前输出字符串。如果status是整数，这个值会被用作退出状态。退出状态的值为0～254。退出状态255由PHP保留，不会被使用。状态0用于表示成功地终止程序。exit和die相同。下面是函数die的范例。

【例4-5-1】 （代码位置：\4\testDie.php）

```
<?php
  $a=false;
  $a or die("false");              //变量$a为false则执行die语句输出false
  echo "true";                     //该语句不会被执行
?>
```

4.5.2 日期时间函数

函数date用来格式化一个本地时间/日期，其格式如下：

```
date(format [,timestamp])
```

整型参数 timestamp 表示时间截，是可选的，字符串 format 规定如何返回结果，例如，h 表示小时，i 表示分钟，s 表示秒。下面是函数 date 的范例。

【例 4-5-2】 （代码位置：\4\testDate.php）

```
<?php
  //Assuming today is: March 10th, 2001, 5:16:18 pm
  $today=date("F j, Y, g:i a");
  echo "$today <br/>";
  $today=date("m.d.y");
  echo "$today <br/>";
  $today=date("j, n, Y");
  echo "$today <br/>";
  $today=date("Ymd");
  echo "$today <br/>";
  $today=date('h-i-s, j-m-y, it is w Day z ');
  echo "$today <br/>";
  $today=date('\i\t \i\s \t\h\e jS \d\a\y.');
  echo "$today <br/>";
  $today=date("D M j G:i:s T Y");
  echo "$today <br/>";
  $today=date('H:m:s \m \i\s\ \m\o\n\t\h');
  echo "$today <br/>";
  $today=date("H:i:s");
  echo "$today <br/>";
?>
```

程序运行结果如下：

```
August 9, 2012, 8:09 am
08.09.12
9, 8, 2012
20120809
08-09-32, 9-08-12, 0931 0932 4 Thuam12 221
it is the 9th day.
Thu Aug 9 8:09:32 UTC 2012
08:08:32 m is month
08:09:32
```

函数 microtime 的主要作用是返回当前 UNIX 时间戳和微秒数。不过这个函数仅在支持 gettimeofday()系统调用的操作系统下可用。下面是关于函数 microtime 的范例，该范例先随机产生循环次数，然后获取开始循环时间，再执行循环，接着获取循环结束时间，最后通过计算两个时间的时间差获取循环的运行时间。