第 5 章　函数、数组与字符串操作

学习目标：

本章将介绍 PHP 语言中函数、数组和字符串的相关知识，包括其概念和语法规则。通过本章的学习可掌握函数及数组的语法规则和字符串的相关操作，在编写程序时能够熟练运用函数解决实际问题。本章学习要求如表 5-1 所示。

表 5-1　本章学习要求

知识要点	能力要求	相关知识
函数	理解函数的概念，掌握函数的语法规则	关键字、实参、形参、调用
数组	理解数组的概念，掌握数组的语法规则	一维数组、二维数组
字符串	掌握字符串的相关操作，了解正则表达式	

PHP 是一个热门的脚本语言，是从 Perl 和 C 语言发展而来的，与 Perl 和 C 语言有很多相似之处，用 PHP 语言可进行关于函数、数组与字符串的定义和相关操作，其基本内容可以包括函数的基本知识、一维数组、多维数组、数组排序、基本的字符串函数和正则表达式等。

5.1　函数

函数就是一段能够完成指定任务的已命名的 PHP 代码，这段代码可以有很少几个语句，也可以有几百行语句，可以遵照给它的一组值和参数来完成指定的任务，并且可能返回一个函数值。使用函数能够节省程序的编译时间，无论调用函数多少次，只需要页面进行一次编译，则可提高程序执行效率。使用函数的好处在于能够分离相关 PHP 代码，按照功能的不同设置不同的程序模块，欲实现某个功能时，只需要调用相应的函数，则可实现代码的重用，提高程序的可读性和可维护性。

利用函数进行模块化程序设计时，首先需要了解一些基本知识，包括函数的一般形式、参数和返回值，函数的调用方式和变量的作用域以及生命周期等。

5.1.1　函数的一般形式

PHP 中的函数包括内置函数和自定义函数两种类型。内置函数是 PHP 中已经定义的完成特定功能的函数，不需要对函数体进行修改就可以直接调用。自定义函数是程序设计者为了实现特殊的目的而编写的代码段，在 PHP 中自定义函数要用到关键字 function，这表示使用者要进行自定义函数的编写。

PHP 中的函数和其他高级语言一样，包括有返回值和无返回值两种类型。函数的命名规则与 C 语言相似，只能由字母，数字和下画线构成，且不能以数字开头，函数名称不区分

大小写。

要定义一个函数，可以使用下面的语法结构：

```
function function_name($arg1,$arg2,…)
{
    statement list              //执行的操作
    ⋮
    return $result;             //函数的返回值,也可以没有
}
```

其中，function 是关键字，代表要定义一个自定义函数；function_name 是自定义函数的名称；arg1、arg2 是函数的参数（即传递变量）；statement list 是函数体，完成指定的任务，函数体除了包含 PHP 代码外，还可以包括 HTML 代码；return 语句后面紧跟着函数的返回值，当函数在执行期间遇到 return 语句时，将跳出函数体返回到调用语句，并将变量 result 的值作为函数值返回，一个函数体内可以包含多个 return 语句。

【例 5-1】 定义一个函数，实现两个字符串的连接。

```
<?php
    function strcat($str1,$str2)
    {
        $str3=$str1.$str2;
        echo str3."<p>\n";
    }
?>
```

这个函数的输出结果是 str1 和 str2 两个字符串连接的结果。

在自定义函数时，函数的参数既可以在函数调用时进行赋值，也可以预先赋值，在函数定义的时候将初始值赋给参数。

【例 5-2】 定义一个函数，实现 3 个字符串连接。

```
<?php
    function strcat($str1,$str2,$str3="it's very good")
    {
        $str4=$str1.$str2.$str3;
        echo str4."<p>\n";
    }
?>
```

这个函数的特点在于当函数定义时，参数列中使用了预先赋值的参数。

定义过一个函数后，可以在程序的任何地方对函数进行调用，完成特定的功能。在自定义函数时，一个非常好的习惯是根据函数要完成的功能来给函数命名，良好的命名习惯不仅能增加程序的可读性，还有利于程序的调试和排错。

5.1.2 函数参数与返回值

1. 函数的参数

在函数定义的语法格式中，函数名后“()”中的变量就是函数的参数（即传递变量）。参

数可以有，也可以没有；可以是一个，也可以是多个。通过参数可以将信息传递到函数体中，参数可以是变量，也可以是常量，当一个函数包括多个参数时，各个参数中间用“,”进行分隔。

函数参数的传递有两种方式：按值(value)传递和按引用(reference)传递。

(1) 按值传递参数。PHP 中大多数情况下是按值传递参数，传递的是变量值，而不是变量在内存中的实际地址。在调用函数时，传递给函数的参数只是变量的一份副本，在函数体内对这份副本的任何操作都不会影响到原来变量的值。函数利用传递过来的值进行计算，再将结果返回，函数外的变量在内存中的地址不变，其变量值也不变。

【例 5-3】 一个函数参数按值传递的简单例子。

```
<?php
    function square ($r)
    {
        $r=$r * $r;
    }
    $a=2;
    square($a);
    echo $a;
?>
```

在这个例子中，使用按值传递的方式给 square()函数传递了变量 a 的值作为参数进行计算，函数的输出结果为 2。由于传值只是在函数体内改变参数的复制值，并不会改变原变量的值，所以最后输出变量 a 的值仍为 2 而不是 4。

(2) 按引用传递参数。程序设计中，有时希望函数能够对参数传递过来的变量值进行改动，并且这些改动能够在函数之外体现出来，这时就需要按引用方式来传递参数，这种方式可以对变量作直接的修改。

按引用传递参数时，参数必须是一个变量，并且要在参数列表中的变量名前加“&”表明该变量将按引用传递。按引用传递参数时，传递的是变量在内存中的地址，函数利用这个地址访问其所指的值，并改变这个值，从而使得变量的值发生改变。

【例 5-4】 按引用传递参数的例子。

```
<?php
    function square(&$r)
    {
        $r=$r+$r;
    }
    $a=2;
    square($a);
    echo $a;
?>
```

本例采用的是按引用传递参数，将 $a 的地址所指向的值 2 传递出去，并且这个值发生了变化，$a 也相应发生变化，$a 所处的内存地址所指向的值已经发生了变化，最后的输出结果是 4，变量 a 的最终值也是 4。

按引用传递参数与按值传递参数一样，在调用参数时没有什么区别，都是在函数体内直接使用。

(3) 默认参数值。自定义函数时，还可以给函数的参数赋一个初始值作为默认参数值，在调用函数时，即使没有给函数传递参数，函数也能够按照默认参数值来执行。

要指定一个默认参数，可以在函数声明时赋予该参数一个初始值，并且这个初始值只能是一个常量。

【例 5-5】 默认参数值举例。

```
<?php
    function stringcon($str1,$str2,$str3="beautiful.")
    {
        echo $str1.$str2.$str3."<br>";
    }
    stringcon("life ","is ");
    echo "<BR>";
    stringcon("life ","is ","very beautiful!");
?>
```

在这个例子中，首先定义了一个字符串连接函数，它一共有 3 个参数，其中前两个是普通参数，第三个参数具有默认参数值。函数的功能是将参数传递过来的字符串连起来，并在屏幕上显示出来。程序对定义的函数按照传值方式进行了两次调用，第一次调用时，给函数传递了两个参数值，第三个参数使用默认值；第二次调用时，给函数传递了 3 个参数值，所以函数直接将传递过来的 3 个字符串连接起来，输出结果。输出结果如下：

```
life is beautiful.
life is very beatiful!
```

使用默认参数还需要注意的默认参数的书写位置，有默认值的函数参数必须放在没有默认值的参数后面，即默认参数必须应放在参数列表的最后，否则 PHP 在执行函数时会出现无法预料的结果。

【例 5-6】 默认参数的书写位置不同造成程序执行结果与预期不符。

```
<?php
    function stringcon($str2=" can succeed!",$str1)
    {
        echo $str1.$str2."<br>";
    }
    stringcon("Everybody ");

?>
```

这个例子的输出结果是“Everybody”，并不是预期的“everybody can succeed!”。为了能够达到程序预期的执行效果，需要对自定义函数的参数位置进行调换，将默认参数放到参数表的最后。

【例 5-7】 将默认参数放到参数表最后。

```
<?php
    function stringcon($str1,$str2=" can succeed!")
    {
        echo $str1.$str2."<br>";
    }
    stringcon("Everybody ");

?>
```

经过修改后本例的输出结果是“Everybody can succeed!”，达到了程序的预期执行效果。

一个自定义函数中可以有任意数目的带有默认参数值的参数，但这些参数必须列在所有没有默认值的参数后面。

2. 函数返回值

在程序设计中，当一个函数被调用后，有时需要其返回一个或多个结果，这就是函数的返回值。在 PHP 中，可以使用关键字 return 来获得函数的返回值，且函数的返回值可以是任何类型的数值，包括数组和对象。

当需要函数返回单个结果时，可以直接使用 return 语句来实现。而当需要函数返回多个结果时，可以通过数组或者广义表等来实现。

【例 5-8】 一个函数返回单个值。

```
<?php
    function multiply($one,$two,$three)
    {
        $product=$one * $two * $three;
        return $product;
    }
    $pro=multiply(2,3,4);
    echo $pro;
?>
```

这个例子实现的功能是将 3 个数相乘并返回一个相乘的结果，最后的输出结果是 24。

【例 5-9】 一个函数的返回多个值。

```
<?php
    function addition($one,$two,$three)
    {
        $sum=$one+$two+$three;
        return array($one,$two,$three,$sum);
    }
    list($fac1,$fac2,$fac3,$sum)=addition(4,5,6);
?>
```

这个例子实现了一个函数返回多个值的功能，这个函数实现 3 个数相加的功能，共返回了 4 个值，包括传递过来的 3 个参数和最终计算结果。

5.1.3 函数调用

函数调用分为 3 种形式：一般调用、嵌套调用和递归调用。

1. 一般调用

函数的一般调用就是普通的函数调用，即直接用函数名称来调用函数，这也是函数调用最常用的方式。其调用形式如下：

```
$some_value =function_name([parameter,…]);
```

其中，some_value 是函数的返回值；function_name 是函数的名称；parameter 是函数的参数，可以省略，也可以是多个。

如果调用的是不带参数的函数，则 parameter 可以省略，但“()”不能省略。如果实参列表包含多个参数，则各参数之间用“,”分隔。实参与形参的个数应该相等（默认参数可以省略），且实参与形参的顺序必须对应。

【例 5-10】 比较两个数大小。

```
<?php
    function Max($x,$y)
    {
        if(x>y)
            return x;
        else
            return y;
    }
    echo max(10,3);
?>
```

本例为函数的一般调用，通过使用 max(10,3)的形式来调用自定义 max()函数，最终的输出结果为“10”。

2. 嵌套调用

函数的嵌套调用是指在一个自定义函数中调用其他自定义函数，其调用的语法格式与普通调用语法格式相同。

【例 5-11】 函数的嵌套调用。

```
<?php
    function B ($b)
    {
        $b=$b * $b;
        echo $b."<br>";
    }
    function A ($a)
    {
        $c=$a+$a;
        B($a);
        echo $c."<br>";
```

```
    }
    A(4);
?>
```

本例在定义函数 A()时对函数 B()进行调用,属于一个典型的函数嵌套调用。程序的执行过程如下。

(1) 执行程序 A 的前半部分“$c=8”。

(2) 调用程序 B。

(3) 执行程序 B:“$b=16”并且输出 $b 的值。

(4) 返回到程序 A,继续执行程序 A 的后半部分:输出 $c 的值,直到结束。

本例的运行结果是通过屏幕显示 16 和 8。

3. 递归调用

函数的递归调用是指在一个函数内部直接或间接地调用函数本身,其调用的语法格式与普通调用语法格式相同。需要注意的是,递归调用函数内部要有终止条件,否则会无限调用下去。

递归调用的特点是,将大而复杂的问题分解成与原问题相同但规模较小的问题,通过解决这些小规模问题,从而最终解决大而复杂的问题。

【例 5-12】 求一个数的阶乘。

问题分析:要想求出 n 的阶乘,首先需要求出 $n-1$ 的阶乘,即 $n!=n\times(n-1)!$,重复这个求解过程,直到 $n=1$ 为止。要解决这个问题就需要通过函数的递归调用来实现。

```
<?php
    function mul($a)
    {
        $result;
        if ($a<0)
            echo "can't calculate"."<br>";
        else
            if ($a==0||$a==1)
                $result=1;
            else
                $result=mul($a-1) * $a;      //此处进行函数的递归调用
        return ($result);
    }
    $c=mul(5);                               //输出 5 的阶乘
    echo $c;
?>
```

在这个例子中,定义函数 mul()时,在函数体内对其自身进行了调用,属于一个典型的函数嵌套调用。程序的运行结果为 120。

汉诺塔问题是一个古典的数学问题,也是一个用递归方法求解的典型例子。

【例 5-13】 汉诺塔问题。从前,一个寺院有 3 根柱子(1 号为原始柱,2 号为交换柱,3 号位目标柱),1 号柱上小下大地放着 64 个金盘(依次由小到大),有一队和尚想将这 64 个

金盘从 1 号柱移到 3 号柱上。规则是一次只允许移动一个,且在移动过程中,在 3 根柱子上都始终保持大盘在下,小盘在上。在移动过程中可以利用 2 号柱。全部移动完需要多长时间。要求编程序输出移动步骤。

程序代码如下:

```
<?php
    function move($d,$e)
    {
        global $step;
        $step=$step+1;
        echo $step.":";
        echo $d."->".$e."<br>";
    }
    function hanoi($n,$a,$b,$c)
        {
            if ($n==1)
            {
                move($a,$c);
            }
            else
            {
                hanoi($n-1,$a,$c,$b);
                move($a,$c);
                hanoi($n-1,$b,$a,$c);
            }
        }
        $num=4;
        $step=0;
        hanoi($num,"one","two","three");
?>
```

在本程序的代码中,函数 hanoi($n, $a, $b, $c)表示将 $n 个盘子从 $a 借助 $b 移到 $c 的过程。函数调用 move($d, $e)表示将一个盘子从 $d 移到 $e 的过程。 $d 和 $e 都表示 one、two、three 3 根柱子中的一根,根据每次情况不同取值,最终完成求解任务。

5.1.4 变量的作用范围和生命周期

变量的作用范围和生命周期是变量的两个很重要的性质,决定了程序的哪些部分可以访问变量,以及变量的生存时间。

1. 变量的作用范围

变量的作用范围,即通常所说的变量作用域,指的是变量在哪些范围内能被使用,在哪些范围内不能被使用。在使用 PHP 语言进行程序开发时,可以在任何位置声明变量,但是变量声明位置及声明方式的不同决定了变量作用域的不同。

在 PHP 中,按照变量作用域的不同可以将变量分为局部变量和全局变量两种。

(1) 局部变量。同其他程序语言类似,PHP 中的局部变量是声明在某一个函数体内的变量,该变量的作用范围仅限于其所在的函数体的内部,函数外的代码不能访问这个变量。如果在该函数体的外部引用这个变量,PHP 将会认为程序引用的是另外一个同名变量。

【例 5-14】 局部变量应用举例。

```
<?php
    function local()
    {
        $a="inside var.";       //在函数内部声明一个变量 a 并赋值
        echo "函数内部变量 a 的值为".$a."<br>";
    }
    local();
    $a="outside var.";          //在函数外部再次声明变量 a 并赋另一个值
    echo "函数外部变量 a 的值为".$a;
?>
```

本例中,由于两个变量 a 声明的位置不同,所以为两个不同的变量。其执行后的输出结果如下:

```
函数内部变量 a 的值为 inside var.
函数外部变量 a 的值为 outside var.
```

(2) 全局变量。在函数以外声明的变量为全局变量,它的作用范围最广泛,可以在程序的任何地方被访问。但是在默认情况下,不能在一个函数体内访问全局变量,要想在函数中访问一个全局变量,需要在函数中使用关键字 global(不区分大小写,也可以是 GLOBAL)。

通过使用全局变量,在程序设计时能够实现在函数内部引用函数外部的变量,或者在函数外部引用函数内部的变量。

【例 5-15】 全局变量应用举例。

```
<?php
    function local()
    {
        $a="inside var.";
        echo "在 local 函数内部获得变量 a 的值为".$a."<br>";
        global $b;                  //将变量 b 声明为全局变量
        $b="global var.";           //在 local 函数内部对变量 b 进行赋值
    }
    local();
    echo "在 local 函数外部获得变量 b 的值为".$b.
?>
```

运行结果如下:

```
在 local 函数内部获得变量 a 的值为 inside var.
在 local 函数外部获得变量 b 的值为 global var.
```

将一个变量声明为全局变量，还有另外一种方法，就是利用 $GLOBALS[]数组，此处不再详述。

注意：通过使用全局变量，虽然能够更加方便地操作变量，但是有时变量作用域的扩大会给程序开发带来麻烦，可能会引发一些预料不到的问题。所以应该谨慎使用全局变量。

2. 变量的生命周期

变量的生命周期指的是变量能够被使用的一个时间段，在这个时间段内变量是有效的，一旦超出这个时间段，变量就会失效，不能再访问到该变量的值。

PHP 对变量的生命周期有如下规定。

(1) 局部变量的生命周期为其所在函数被调用的整个过程。当局部变量所在的函数结束时，局部变量的生命周期也随之结束。

(2) 全局变量的生命周期为其所在的 PHP 脚本文件被调用的整个过程。当全局变量所在的脚本文件结束调用时，全局变量的生命周期结束。

程序设计中，有时希望一个变量的值能在其生命周期之外被调用，这时就需要将变量声明为静态变量，将一个变量声明为静态变量的方法是在变量前面加关键字 static。

【例 5-16】 静态变量举例。

```
<?php
    function stavar()
    {
        static $a=0;            //定义一个静态变量 a 并赋初始值为 0
        echo $a."<br>";         //输出变量 a 的值
        $a=$a+2;                //将变量 a 的值加 2 再次赋给变量 a
    }
    stavar();                   //调用函数 stavar()
    stavar();
    echo $a;                    //变量 a 是局部变量,不能在函数外使用,故不会输出任何值
?>
```

运行结果如下：

```
0
2
```

本例中每次调用函数 stavar()时，变量 a 的值都会增加 2。也就是说，每次调用函数结束以后，变量 a 都仍然存在，再次调用函数 stavar()时，变量 a 将会使用上一次调用该函数结束时得到的值。

从上面的例子也可以看出，虽然静态局部变量在函数调用结束后仍然存在，但是其他函数是不能引用它的。静态局部变量的作用范围与局部变量相同，但是生命周期与全局变量相同。

静态变量具有以下特点。

(1) 只有函数首次被调用时，才会给函数体内静态变量赋初始值。再次调用该函数时，静态变量保存的是上次调用这个函数后得到的值。

(2) 在为静态变量赋初值时，不可以将一个表达式赋给该静态变量。