第5章　PHP函数

一个基本的 PHP 脚本通常是由主程序和函数构成。这些函数不但造就了 PHP 脚本的主要功能，而且实现了程序的结构化，并且方便阅读者阅读代码。函数分为用户自定义函数和内建函数(系统内部函数)，函数的再次抽象可以作为类的一个方法，类的方法将在面向对象章节进行介绍。本章将主要介绍用户自定义函数、函数参数、变量函数和内建函数。

5.1　用户自定义函数

在开发过程中，经常要用到某种重复的操作或处理。如果每个操作都编写一段代码，既浪费编程人员的时间和精力，也对后期维护带来极大的麻烦。由于 PHP 内部自带有很多函数，但是这些函数不足以满足所有的任务，这时就需要使用用户自定义函数来解决各种实际问题。

5.1.1　自定义函数的编写和调用

在 PHP 中，一个用户自定义函数通常由四部分组成：函数名、参数、函数体和返回值。一个典型的语法格式如下：

```
<?php
function func_name($arg1, $arg2, … , $argn) {
     func_body; //这里是函数体，即函数的功能代码
     return $arg;//这里是返回值
}
?>
```

参数说明如下：

(1) function：声明自定义函数必使用的关键字，不可省略。

(2) func_name：函数名称，其命名规则与变量命名规则相同，区别是不能以“$”开头，有一点需要注意：函数名称是不区分大小写的。

(3) $arg1, $arg2, ... , $argn：传递给函数的参数，根据函数的情况，参数的数量根据需要而定，中间用逗号分隔，也可不带参数。

(4) func_body：自定函数的主体，实现需要的各种功能。

(5) return：将调用代码需要的值返回，并结束函数的运行。

当函数定义完成以后，接着就是调用此函数。调用函数非常简单，只需要引用函数名

并赋予正确的参数即可。

实例 5-1：一个简单的函数实例

```
<?php
function sum($a, $b) {
     $sum = $a + $b;
     return $sum;
}
$a = 10;
$b = 23;
echo "a=" . $a . ",b=" . $b . "<br>";
echo "a+b=" . sum($a, $b);

?>
```

运行上述代码，结果如图 5-1 所示。

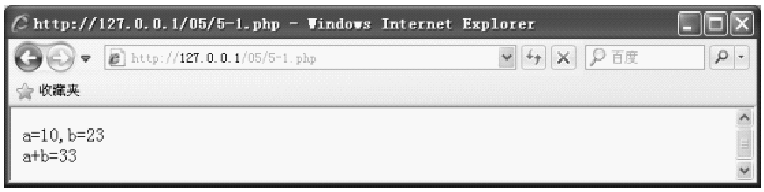

图 5-1　简单的函数实例

注意：

return 语句只能返回一个值，不能返回多个值。如果想返回多个值，可定义一个数组，将返回值存放在数组中，然后返回一个数组。

5.1.2　函数的动态调用

由于 PHP 支持可变化的 function 函数概念，这就意味着如果在一个变量名称的后面加上一对圆括号，PHP 就会去寻找与这个变量名字相同的函数，无论这个变量的数值是什么，函数都会被执行。在下面的这个实例中，通过对$myFunction 的两次赋值，实现了对于函数的动态调用。

实例 5-2：函数的动态调用

```
<?php
function write($text) {
     echo "<font size=13pt>";
     echo $text;
     echo "</font>";
}
function writeBold($text) {
     print("<font size=3pt><B>$text</B></font>");
```

```
}
$myFunction = "write";
$myFunction("只有刻苦学习知识，");
echo "<p>";
$myFunction = "writeBold";
$myFunction("才能将知识变成财富！");

?>
```

运行上述代码，结果如图 5-2 所示。

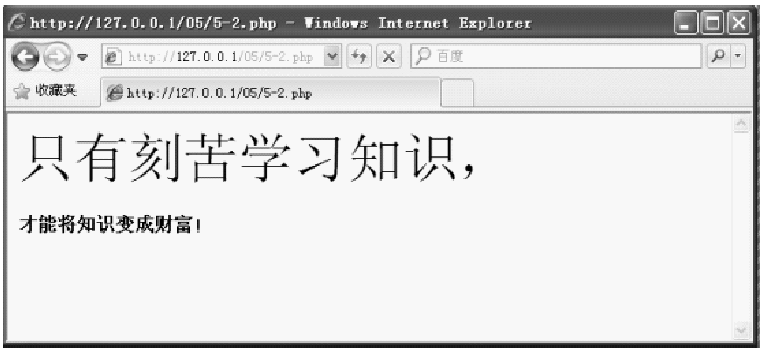

图 5-2　函数的动态调用

5.1.3　函数的递归

学过 C 语言的人都知道递归是怎么一回事，此处不再详细说明什么是递归。用递归处理问题，代码简洁、易读，下面就用斐波那契数列来说明函数的递归调用。

斐波那契数列又称黄金分割数列，指的是这样一个数列：1、1、2、3、5、8、13、21、…，即某一项为其前两项的和。

实例 5-3：递归函数

```
<?php
function Fibanacci($num) {
     if ($num == 1 || $num == 2)
          return 1;
     else {
          return Fibanacci($num-1) + Fibanacci($num-2);
     }
}
for($i = 1;$i <= 20;$i++) {
     echo Fibanacci($i) . ' ';
}
?>
```

运行上述代码，结果如图 5-3 所示。

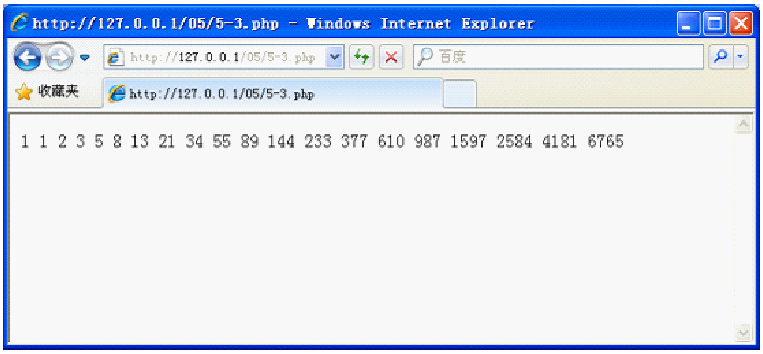

图 5-3　递归函数

知识点：

递归函数其实就是调用自身的函数。递归调用与循环实现的功能是相同的。

注意：

递归的优点是简洁、清晰，缺点是运行慢、占用内存多，如果不是很复杂的函数，尽量不使用递归，这对性能的帮助将很大。

5.1.4　函数的嵌套

函数嵌套跟函数递归形式上相同，只不过函数嵌套调用的是别的函数而非自身。嵌套可以将一个功能复杂的函数分解为多个子函数，再通过调用结合，提高函数的可读性。

实例 5-4：函数的嵌套

```
<?php
function a_func($arg1, $arg2) {
     echo ($arg1 + $arg2);
     echo "<br/ >";
     $var = 100;
     function b_func($arg1, $arg2) {
        var_dump($var);//输出 NULL
        echo "<br/ >";
        echo ($arg1 - $arg2);
        echo "<br/ >";
     }
     b_func(12, 13);
}

a_func(12, 13);
b_func(12, 13);

?>
```

运行上述代码，结果如图 5-4 所示。

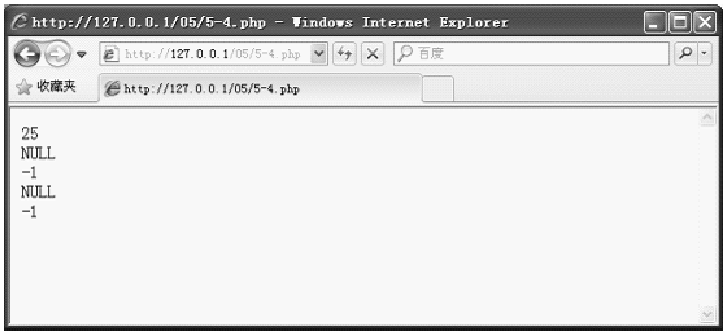

图 5-4　函数的嵌套(1)

知识点：

在该实例中，var_dump($var)语句输出为 NULL，这是因为在函数 b_func 中无法访问到变量$var，即使声明为 global 也不行。变量作用域这个问题在第 2 章中已经讨论过。

注意：

如果先调用 b_func(12,13)，再调用 a_func，就会提示 b_func 未定义，如图 5-5 所示。

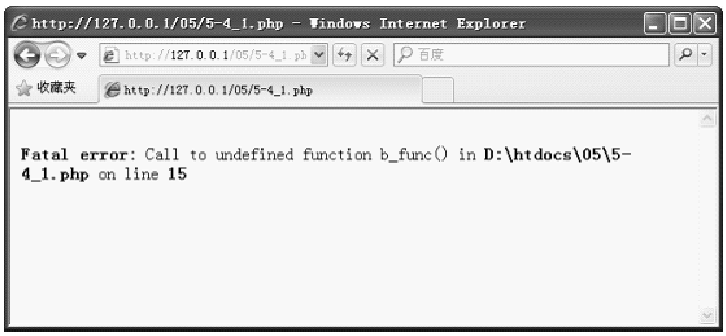

图 5-5　函数的嵌套(2)

5.1.5　函数的返回值

在 5.1.4 小节的实例中，都用到了函数的返回值，它们是通过 return 语句来实现的。函数的返回值可以是变量、常量、数组或表达式。

实例 5-5：函数的返回值

```
<?php
function return_test($n) {
     $t = $n > 1?$n * return_test($n-1):1;
     return $t;
}
echo return_test(10);
?>
```

运行上述代码，结果如图 5-6 所示。

图 5-6　函数的返回值

实例 5-6：返回一个数组

```
<?php
function arr_return($n1, $n2) {
     or($i = $n1;$i < $n2;$i++)
        $arr[$j++] = $i;
     return $arr;
}
$array = arr_return(4, 10);
for($i = 0;$i < count($array);$i++) {
     echo $array[$i];
     echo "<br/ >";
}
?>
```

运行上述代码，结果如图 5-7 所示。

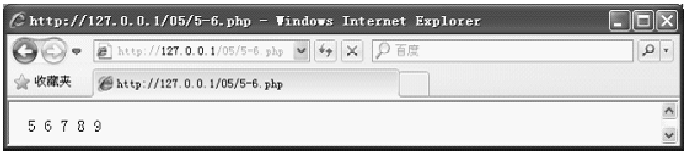

图 5-7　返回一个数组

知识点：

上面的代码求出两个整数之间的所有整数，因为两个整数间的数可能为多个，所以要返回多个数值，只能把这些数值定义到数组中，然后返回数组。

5.2　函数的参数

调用 PHP 函数时，根据函数的需要，可能会向函数传递参数，与 C 语言一样，调用时传入的参数叫实参，定义函数时的参数叫形参。参数的传递方式有按值传递、按引用传递和默认参数三种。

5.2.1　按值传递参数

按值传递参数是将实参的值复制到对应的形参中，然后在函数内部对形参进行操作，

操作结果不影响到实参，即函数返回后，实参的值不会改变。

实例 5-7：按值传递参数

```
<?php
function test_func($n) {
     $n = $n * $n * $n;
     echo "函数内：\$n=" . $n;
}
$n = 5;
test_func($n);
echo "<p>";
echo "函数外：\$n=$n";
?>
```

运行上述代码，结果如图 5-8 所示。

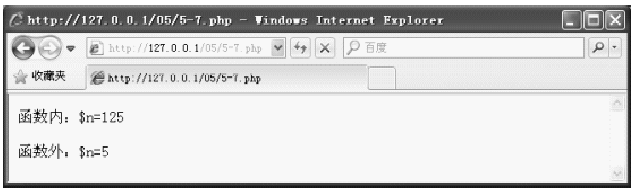

图 5-8　按值传递参数

知识点：

在该实例中，函数的功能是传入一个整数，然后求出其立方。在函数外部定义了一个变量$n，即要传入的参数。由结果可知，调用函数之后，并未改变函数外部变量$n。

5.2.2　按引用传递参数

按引用传递参数其实就是将变量的地址传递到形参中，最终的结果是在函数内部对参数进行了修改，在函数外部也能同步反映。按引用传递参数只需在参数前加“&”符号即可。

实例 5-8：按引用传递参数

```
<?php
function test_func(&$n) {
     $n = $n * $n * $n;
     echo "函数内：\$n=" . $n;
}
$n = 5;
test_func($n);
echo "<p>";
echo "函数外：\$n=$n";
?>
```

运行上述代码，结果如图 5-9 所示。

图 5-9　按引用传递参数

知识点：

从结果中可以看出，不管调用的是函数输出的结果还是函数外部的变量，在调用函数后，都已经改变了。

5.2.3　使用默认参数

函数还可以使用一个或多个默认的参数。如果在调用时不指定传递参数，则函数会按照默认值去处理。

实例 5-9：使用默认参数

```
<?php
function add(&$n, $m = 10) {
     $n = $n + $m;
}
$n = 10;
add($n);
echo "$n";
echo "<br/ >";
add($n, 25);
echo"$n";
?>
```

运行上述代码，结果如图 5-10 所示。

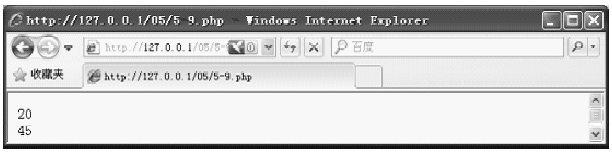

图 5-10　使用默认参数

知识点：

实例 5-9 中的函数用到了引用传递，第一次调用 add($n)，其结果为 10+10=20，可看出，不传入第二个参数时，将按默认参数完成任务；此时在函数外部，变量$n 也变成 20；然后再次调用 add($n,25)，其结果为 20+25=45。

5.2.4　可变参数的函数

通常用户定义函数时，设置的参数的个数是有限的。如果希望函数可以接受任意数量

的参数，则需要用到 PHP 提供的三个内建函数，分别是 func_num_args( )、func_get_arg( )以及 func_get_args( )函数。

func_num_args( )函数在用户自定义函数中使用，获取传入函数参数的个数。func_get_arg( )函数指定要存取的哪个参数，第一个参数值为 0；func_get_args( )函数表示将传入参数以数组的方式返回。

实例 5-10：可变参数的函数应用

```
<?php
function foo() {
     // func_num_args()函数取得可变参数的个数
     $numargs = func_num_args();
     echo "参数个数为: $numargs\n" . "<br />";
     // func_get_arg($i)获取第 i 个参数
     for($i = 0; $i < $numargs; $i++) {
        echo "第" . $i . "个参数是" . func_get_arg($i) . "<br />";
     }
     echo "<hr />";
     // func_get_args()获取所有的参数，然后返回到一个数组中
     $args = func_get_args();
     // 将数组用逗号分解成一个字符串
     $args = implode(",", $args);
     echo "用 func_get_args()函数取得的参数: $args";
}
foo("Hello", "World", "PHP");

?>
```

运行上述代码，结果如图 5-11 所示。

图 5-11　可变参数的函数应用

5.3　内 建 函 数

PHP 提供了 1000 多个内建函数，还有各种外部扩展库提供的函数库，它们非常丰富，囊括了各个方面。由于各个章节都会讲解到内建函数，因此下面讲解一些常用数学函数以及时间处理函数。

5.3.1　常用数学处理函数

常用数学处理函数有求绝对值、求幂、求进位制、三角函数、反三角函数等。

求绝对值：abs( )函数用来求一个数的绝对值，其函数原型如下：

```
number abs ( mixed $number )
```

求指数：exp( )函数用于计算以 e 为底数、以$arg 为指数的幂运算；pow( )函数用于计算以$base 为底数、以$exp 为指数的幂运算。其函数原型如下：

```
float exp ( float $arg )
number pow ( number $base , number $exp )
```

求对数：log( )函数用于计算以 e 为底的自然对数或者以$base 为底的$arg 对数；log10( )函数用于计算以 10 为底的对数。其函数原型如下：

```
float log ( float $arg [, float $base ] )
float log10 ( float $arg )
```

求平方根：sqrt( )函数用于计算参数$arg 的平方根，其函数原型如下：

```
float sqrt ( float $arg )
```

取整函数：floor( )函数向下取整；ceil( )函数向上取整；round( )函数将根据指定精度$precision 四舍五入，$precision 默认为 0。其函数原型如下：

```
float floor ( float $value )
float ceil ( float $value )
float round ( float $val [, int $precision ] )
```

求最大值、最小值：max( )函数和 min( )函数接受两个或者两个以上 number 型参数，max( )函数返回最大值，min( )函数返回最小值。其函数原型如下：

```
mixed max( number $arg1 , number $arg2 [, int $arg3] …[, int $argn]  )
mixed min( number $arg1 , number $arg2 [, int $arg3] …[, int $argn]  )
```

求进位制转换：base_convert( )函数用来进行进位制转换函数，用来转换原以$frombase 进制表示的$number 为以$tobase 进制表示的数，然后输出为字符串。其函数原型如下：

```
string base_convert ( string $number , int $frombase , int $tobase )
```

三角函数：sin( )函数求一个浮点型弧度值的正旋；cos( )函数求一个浮点型弧度值的余弦，tan( )函数求一个浮点型弧度值的正切。

```
float sin ( float $arg )
float cos ( float $arg )
float tan ( float $arg )
```

还有很多数学处理函数，读者朋友们可以在 PHP 手册中参考。

实例 5-11：数学处理函数实例

```
<?php
$a = -3.1415;
$b = 30;
$c = 5;
echo "\$a 的绝对值为：" . abs($a) . "<hr />";
echo "以 e 为底以\$b 为指数的值为：" . exp($b) . "<br />";
echo "以\$a 为底以\$b 为指数的值为：" . pow($a, $b) . "<hr />";
echo "\$b 以 e 为底数的对数为：" . log($b) . "<br />";
echo "\$b 以\$c 为底数的对数为：" . log($b, $c) . "<br />";
echo "\$b 以 10 为底数的对数为：" . log($b) . "<br />";
echo "\$b 的平方根为：" . sqrt($b) . "<hr />";
echo "取\$a 向下的整数为：" . floor($a) . "<br />";
echo "取\$a 向上的整数为：" . ceil($a) . "<br />";
echo "默认取\$a 四舍五入为：" . round($a) . "<br />";
echo "取\$a 小数点后两位四舍五入为：" . round($a, 2) . "<hr />";
echo "求\$a,\$b,\$c 的最大值为：" . max($a, $b, $c) . "<br />";
echo "求\$a,\$b,\$c 的最小值为：" . min($a, $b, $c) . "<hr />";
echo "进制转换，把 10 进制的\$b 转换为 2 进制：" . base_convert($b, 10, 2) . "<hr />";
echo "弧度值为 1 的正弦、余弦、正切分别为：" . "<br />" . sin(1) . "<br />" . cos(1) .
"<br />" . tan(1);

?>
```

运行上述代码，结果如图 5-12 所示。

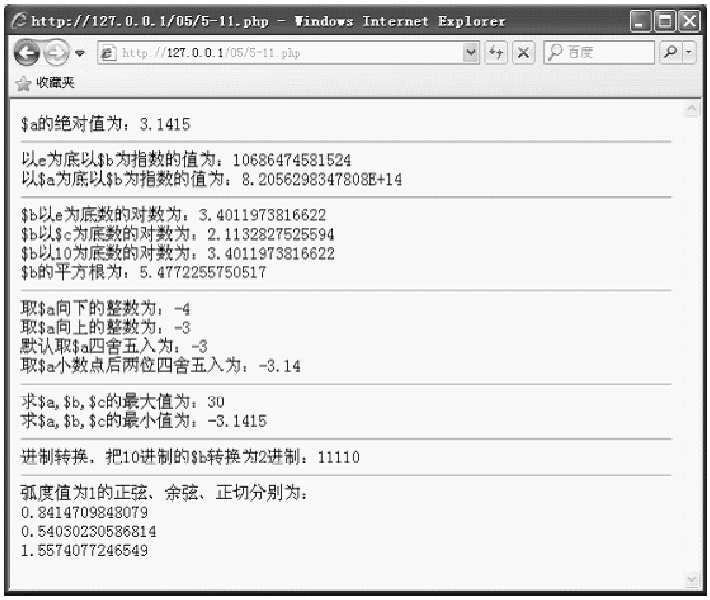

图 5-12　数学处理函数实例

5.3.2　常用时间处理函数

常用时间处理函数在 Web 编程中比数学处理函数更常用，可以使用这些函数取得 PHP 服务器的日期和时间，并且按照指定格式输出。

1．getdate()函数

getdate( )函数用来获取当前时间信息或者分析时间戳的具体意义。其函数原型如下：

```
array getdate ([ int $timestamp ] )
```

默认无参数时该函数返回根据当前本地时间戳得出的包含有日期信息的结合数组，如果有参数$timestamp，则返回根据 $timestamp 时间戳得出的包含有日期信息的结合数组。该返回的数组键名如表 5-1 所示

表 5-1　getdate()函数参数列表

键　　名	说　　明	返 回 例 子
seconds	秒的数字表示	0~59
Minutes	分钟的数字表示	0~59
hours	小时的数字表示	0~23
Mday	月份中第几天的数字表示	1~31
Wday	星期中第几天的数字表示	0(周天)~6(周六)
Mon	月份的数字表示	1~12
Year	4 位数字表示的年份	如 2012
Yday	一年中第几天的数字表示	0~365
Weekday	星期的完整表示(英文)	Sunday~Saturday
Month	月份的完整表示(英文)	January~December
0	从 UMIX 纪元开始至今的秒数	和 time( )、date( )返回值类似

实例 5-12：时间获取函数

```
<?php
$today = getdate();
print_r($today);
echo "<hr />";
// getdate()函数与gettimeofday()函数的区别
$today = gettimeofday();
print_r($today);

?>
```

运行上述代码，结果如图 5-13 所示。

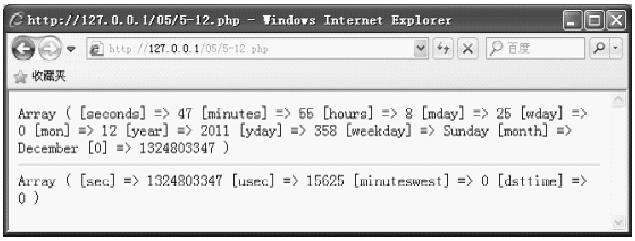

图 5-13　时间获取函数

注意：

getdate( )函数得出的时间不够精确，特别是在高精度程序中需要微秒级别的时间精度，这就需要 gettimeofday( )函数，用来返回精确到微妙级别的时间。

2．checkdate()函数

checkdate( )函数用来检验一个日期是否有效，经常用在检验用户数据的有效性上。其函数原型如下：

```
bool checkdate ( int $month , int $day , int $year )
```

如果参数$month 的值在 1～12 之间、$day 的值在给定的$month 所应该具有的天数范围之内(已经考虑闰年)、$year 的值在 1～32767 之间表示时间合法，函数返回 true，否则返回 false。

实例 5-13：检验日期的合法性

```
<?php
if (checkdate(12, 12, 2012)) {
      echo "2012 年 12 月 12 日日期合法! ";
} else {
      echo "2012 年 12 月 12 日日期 不 合法! ";
}
echo "<hr />";
if (checkdate(13, 12, 2012)) {
      echo "2012 年 13 月 12 日日期合法! ";
} else {
      echo "2012 年 13 月 12 日日期 不 合法! ";
}

?>
```

运行上述代码，结果如图 5-14 所示。

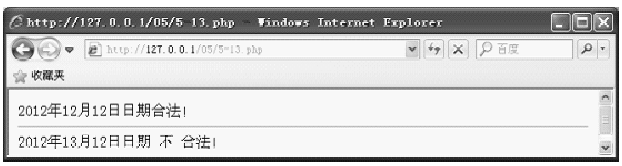

图 5-14　检验日期的合法性

3．date()函数

date( )函数是将日期和时间格式化输出，其函数原型如下：

```
string date ( string $format [, int $timestamp ] )
```

第一个参数$format 是指定的格式化字符串，如果格式化字符串无效，则原样输出；可选参数$timestamp 是给定的时间戳，如果没有设定此参数，则函数格式化输出当前本地时间。函数返回时间戳按照给定的格式化字符串产生的字符串。该函数所使用的$format 参数如表 5-2 所示。

表 5-2　$format 参数列表

format 字符	说　　明	取 值 范 围
时间	…	…
a	小写的上午和下午值	am 或者 pm
A	大写的上午和下午值	AM 或者 PM
B	Swatch Internet 标准时	000~999
g	小时，12 小时格式，无前导 0	1～12
G	小时，24 小时格式，无前导 0	0～23
h	小时，12 小时格式，有前导 0	01～12
H	小时，24 小时格式，有前导 0	00～23
i	有前导 0 的分钟数	00～59
s	有前导 0 的秒数	00～59
日	…	…
d	月份中的第几天，有前导 0	01～31
D	星期中的第几天，3 个字母表示	Mon～Sun
j	月份中的第几天，无前导 0	1～31
l	星期几，完整文本格式	Sunday～Saturday
N	ISO-8601 格式数字表示的星期中的第几天	1(表示星期一)～7(表示星期天)
S	每月天数后面的英文后缀，两个字符	St、nd、rd 或者 th。可以和 j 一起用
w	星期中的第几天，数字表示	0(表示星期天)～6(表示星期六)
z	年份中的第几天	0～366
星期	…	…
W	ISO-8601 格式年份中的第几周，每周从星期一开始	例如：42(当年的第 42 周)
月	…	…
F	月份，完整的文本格式	January 到 December
m	数字表示的月份，有前导 0	01～12
M	三个字母缩写表示的月份	Jan～Dec

(续表)

format 字符	说　　明	取 值 范 围
n	数字表示的月份，没有前导 0	1～12
t	给定月份所应有的天数	28～31
年		
L	是否为闰年	如果是闰年，为 1，否则为 0
o	ISO-8601 格式年份数字	如 2012
Y	4 位数字完整表示的年份	如 2012
y	两位数字表示的年份	如 99 或者 12
时区	…	…
e	时区标识	例如：UTC、GMT、Atlantic/Azores
I	是否为夏令时	如果是夏令时，为 1，否则为 0
O	与格林尼治时间相差的小时数	例如：+0200
P	与格林尼治时间(GMT)的差别，小时和分钟之间由冒号分隔	例如：+02:00
T	本机所在的时区	例如：EST、MDT
Z	时差偏移量的秒数。UTC 西边的时区偏移量总是负的，UTC 东边的时区偏移量总是正的	-43200～43200
完整的日期和时间	…	…
c	ISO 8601 格式的日期	2012-02-12T15:19:21+00:00
r	RFC 822 格式的日期	例如：Thu, 12 Dec 2012 16:01:07 +0200
U	从 UNIX 纪元(January 1 1970 00:00:00 GMT)开始至今的秒数	

实例 5-14：date( )函数

```
<?php
//设定要用的默认时区。自 PHP 5.1 可用
date_default_timezone_set('Asia/Shanghai');

echo "本地时间" . date("Y 年 m 月 j 日，I,H:i:s") . "<hr />";
echo "GMT 时间" . gmdate("Y 年 m 月 j 日，I,H:i:s") . "<hr />";
//输出类似：Monday
echo date("l") . "<hr />";
//输出类似：Monday 26th of December 2011 08:38:02 AM
echo date('l dS \of F Y h:i:s A') . "<hr />";
//输出：December 12, 2012 is on a Wednesday
echo "December 12, 2012 is on a " . date("l", mktime(0, 0, 0, 12, 12, 2012)) .
```

```
"<hr />";

/**
 * 在格式参数中使用常量
 */
//输出类似：Mon, 15 Aug 2005 15:12:46 UTC
echo date(DATE_RFC822) . "<hr />";
//输出类似：2000-07-01T00:00:00+00:00
echo date(DATE_ATOM, mktime(0, 0, 0, 12, 12, 2012));

?>
```

运行上述代码，结果如图 5-15 所示。

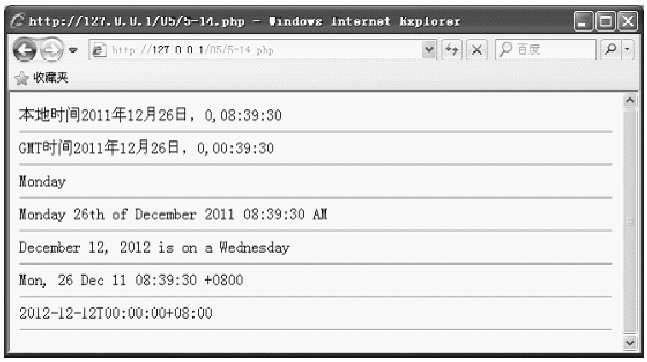

图 5-15　date( )函数实例

注意：

通过这个实例，又引出三个函数，一个是 data_default_timezone_set( )函数，用于设置时区；另一个 gmdate( )函数，是输出格林尼治日期和时间的；而 mktime( )函数返回一个 UNIX 时间戳，只实现时间戳意义不大，所以 mktime( )函数经常和 date( )函数联合使用，用来实现时间的计算。读者朋友们可查看 PHP 手册详细用法。

以上只是列举了一部分常用的日期时间函数，PHP 提供的日期时间函数还有很多，读者可查询 PHP 手册。

5.4　包 含 控 制

如果用户自定义了各种函数或者配置文件，并且想重复利用，则需要用到 require、include、require_once 以及 include_once 语句中的一个，将自己定义的函数或者配置文件引入到 PHP 脚本程序中。

5.4.1　require 和 include 语句

使用 include 语句引用外部文件时，当代码执行到 include 语句时才将外部文件引入并读取，如果所引用的文件发生错误时，系统会给出一个警告，而整个 PHP 文件继续向下运行。

而 require 语句的用法与 include 的用法相似，都是对外部文件的引用。但是在 PHP 脚本被执行前，PHP 解析器将用外部文件替换 require 语句，然后与其他语句组成新的 PHP 文件并执行该新的程序。

由于 require 语句是将一个源文件的内容完全复制到 PHP 脚本文件中，所以通常放置在 PHP 脚本的最前面，主要是用来引用需要的功函数以及公共类等。

require 和 include 几乎完全一样，除了处理失败的方式不同：include 产生一个 Warning，而 require 则导致一个 Fatal Error。

实例 5-15：require 和 include 语句

```
<?php
//在文件头部用 require 引入 config.php 文件
require 'config.php'
//满足条件$condition，则用 include 引入 one.php
if ($condition) {
      include 'one.php';
} else {
      include 'other.php';
}

?>
```

5.4.2　require_once 和 include_once 语句

require_once 和 include_once 语句与 require 和 include 语句类似。它们之间的区别如下。

require_once 和 include_once 语句会记住文件是否已经被包含，如果已经被包含了，则不会再次包含。这两个语句用于在脚本执行期间且同一个文件可能被包含不止一次的情况下，确保该文件只被包含一次，以避免函数和类重复定义及变量重新赋值等。

5.5　难 点 解 析

本章主要介绍了如何在 PHP 中自己定义函数、函数的参数、常用的一些函数以及控制包含语句。对于函数的参数，有按值传递参数、按引用传递参数和默认参数三种，还介绍了利用可变参数函数来突破函数参数的个数。而对于自定义函数和引用文件，它们既可以简化程序流程，还可以增加代码的重用性，以提高工作效率。

本章难点在于，用户自定义函数的编写以及包含控制语句，对于时间处理函数，也经

常用在 PHP 编程中，但这并非难点，只要能按部就班地在该用处使用，就不会出现太大的问题。一个用户自定义函数，通常包括四个部分：函数名、参数、函数体以及返回值。函数体就是实现用户需要实现的各种功能。当函数编写完毕，需要调用此函数，并赋予正确的参数。函数参数的传递有两种方式，一种是按值传递参数，另一种是按引用传递参数，特别要注意这两种传递参数方式在函数返回之后对参数值的改变情况。很多时候，用户可自定义很多函数或者配置文件，并且想重复利用，就需要用到各种包含控制语句，包括 require、include、require_once 以及 include_once 四个控制语句，它们各自有其特点，在使用时一定要注意其区别。

5.6　高手训练营

1. PHP 自定义函数的关键字是_____。
2. 函数的返回值通过_____语句设置。
3. 静态变量使用关键字_____来声明，静态变量的特点是_____。
4. 在 PHP 函数中，按值传递参数和通过引用传递参数有什么区别？
5. 什么叫局部变量？如果在一个函数内引用全局变量有哪些方法？
6. 如何在 PHP 文件中设置时区？
7. 编写一个函数，判断某一年是否为闰年。
8. 编写一个 PHP 函数，在网页中显示今天离元旦还有多少天。如果今天是元旦，则显示“元旦快乐!”