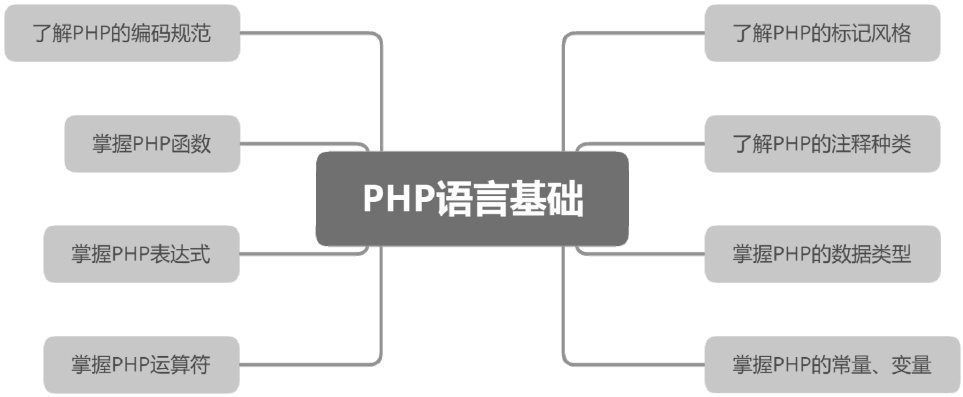
第 3 章

PHP 语言基础

PHP 的特点是易学、易用，但这并不代表随随便便就可以掌握 PHP。随着知识的深入，PHP 会越来越难学，基础的重要性也就愈加明显。掌握扎实的基础知识，是学会一门语言的核心，希望初学者能静下心来，牢牢掌握本章的知识，这样对以后的学习能起到事半功倍的效果。

本章主要介绍了 PHP 语言的基础知识，包括数据类型、常量、变量、运算符、表达式和自定义函数，并详细介绍了各种类型之间的转换、系统预定义的常量、变量、算术优先级和如何使用函数。最后介绍 PHP 编码规范。



3.1 PHP 的标记风格

和其他 Web 语言一样，PHP 也是使用各种标记对将 PHP 代码包含起来，以和 HTML 代码做区分。PHP 支持 4 种标记风格，分别是 XML 风格、脚本风格、简短风格和 ASP 风格。

（1）XML 风格是本书所用的标记风格，也是推荐广大开发者使用的标记风格（原因参见 3.9 节编码规范），服务器不能禁用。该风格使用“<?php … ?>”进行标记，在 XML、XHTML 中都可以使用。例如：

```
<?php
    echo "这是 XML 风格的标记";
?>
```

（2）脚本风格，使用“<script language="php">…</script>”进行标记。例如：

```
<script language="php">
    echo '这是脚本风格的标记';
</script>
```

（3）简短风格，使用“<?…?>”进行标记。例如：

```
<? echo '这是简短风格的标记'; ?>
```

（4）ASP 风格，使用“<%…%>”进行标记。例如：

```
<%
    echo '这是 ASP 风格的标记';
%>
```

说明

使用简短风格和 ASP 风格前，需要在 php.ini 中先进行配置。打开 php.ini 文件，将 short_open_tag 和 asp_tags 都设置为 On，重启 Apache 服务器即可。

3.2　PHP 注释的应用

注释即代码的解释和说明，一般放在代码的上方或代码的尾部，用来说明代码或函数的编写人、用途、时间等。注释放在代码尾部时，代码和注释之间应以 Tab 键进行分隔，以方便程序阅读。注释不会影响程序的执行，执行时注释部分会被解释器忽略不计。

PHP 支持 3 种注释格式，分别是单行注释、多行注释和 # 风格的注释。

（1）单行注释（//）来源于 C++语法格式，注释部分一般写在 PHP 语句的上方或后方。例如：

```
<?php
    //这是写在 PHP 语句上方的单行注释
    echo '使用 C++风格的注释';
?>
<?php
    echo '使用 C++风格的注释';              //这是写在 PHP 语句后面的单行注释
?>
```

（2）多行注释（/*…*/）来源于 C 语言语法格式，分为块注释和文档注释。多行注释不允许进行嵌套操作。其中，块注释示例如下：

```
<?php
    /*
    $a = 1;
    $b = 2;
    echo ($a + $b);
    */
    echo 'PHP 的多行注释';
?>
```

文档注释示例如下：

```
<?php
    /*说明：项目工具类
```

```
    *作者：小辛
    *E-mail:mingrisoft@mingrisoft.com
    */
    class Util
    {
        /**
        *方法说明：给字符串加前缀
        *参数：String $str
        *返回值：String
        */
        function addPrefix($str)
        {
            $str.= 'mingri';
            return $str;
        }
    }
?>
```

（3）#风格的注释。示例代码如下：

```
<?php
    echo '这是#风格的注释';                    #这是#风格的单行注释
?>
```

注意

单行注释中的内容不要出现“?>”标志，否则解释器会认为 PHP 脚本到此已结束。例如：

```
<?php
    echo '这样会出错的！！！！！'              //不会看到?>会看到
?>
```

上述代码的输出结果为：这样会出错的！！！！！会看到 ?>

3.3 PHP 的数据类型

PHP 支持 8 种数据类型，包括 4 种标量数据类型，即 boolean（布尔型）、string（字符串型）、integer（整型）和 float/double（浮点型）；2 种复合数据类型，即 array（数组）和 object（对象）；2 种特殊数据类型，即 resource（资源）和 null（空值）。

说明

PHP 中变量的类型通常不是由程序员设定的，确切地说，是 PHP 根据该变量使用的上下文在运行时决定的。

3.3.1　标量数据类型

标量数据类型是数据结构中最基本的单元，只能存储一个数据。PHP 中标量数据类型包括 4 种，如表 3.1 所示。

表 3.1　标量数据类型

类　　型	说　　明
boolean（布尔型）	最简单的类型，只有两个值：真（true）和假（false）
string（字符串型）	字符串就是连续的字符序列，可以是计算机所能表示的一切字符的集合
integer（整型）	整型数据类型用于存储整数
float（浮点型）	浮点数据类型用于存储小数

1．布尔型（boolean）

布尔型变量通常保存一个 true 值或 false 值，其中 true 和 false 是 PHP 的内部关键字。布尔型变量通常应用在条件判断语句或循环控制语句的表达式中。

【例 3.1】在 if 条件语句中判断变量$boo 中的值是否为 true，如果为 true，则输出“变量$boo 为真!”，否则输出“变量$boo 为假!!”。**（实例位置：资源包\TM\sl\3\1）**

```
<?php
    $boo = true;                          //声明一个 boolean 类型变量，赋初值为 true
    if ($boo == true)                     //判断变量$boo 是否为真
        echo '变量$boo 为真!';             //如果为真，则输出“变量$boo 为真!”
    else
        echo '变量$boo 为假!!';            //如果为假，则输出“变量$boo 为假!!”
?>
```

结果为：变量$boo 为真!

注意

在 PHP 中，不是只有 false 值才为假，一些特殊情况下，boolean 值也被认为是 false。这些特殊情况有 0、0.0、"0"、空白字符串（""）、只声明没有赋值的数组等。

说明

“$”是变量标识符，所有变量都以“$”开头，无论是声明变量还是调用变量，都应使用“$”。

2．字符串型（string）

字符串是连续的字符序列，由数字、字母和符号组成。字符串中的每个字符只占用一个字节。在 PHP 中，有 3 种定义字符串的方式，分别是单引号（'）、双引号（"）和定界符（<<<）。

单引号和双引号是经常使用的定义方式，定义格式如下：

```
<?php
    $a = '字符串';
?>
```

或

```
<?php
    $a = "字符串";
?>
```

两者的不同之处在于，双引号中所包含的变量会自动被替换成实际数值，而单引号中包含的变量则按普通字符输出。

【例 3.2】应用单引号和双引号输出同一个变量。**（实例位置：资源包\TM\sl\3\2）**

```
<?php
    $i = '只会看到一遍';                    //声明一个字符串变量
    echo "$i";                              //用双引号输出
    echo "<p>";                             //输出段落标记
    echo '$i';                              //用单引号输出
?>
```

单引号和双引号的输出结果完全不同，双引号输出的是变量的值，而单引号输出的是字符串“$i”。运行结果如图 3.1 所示。

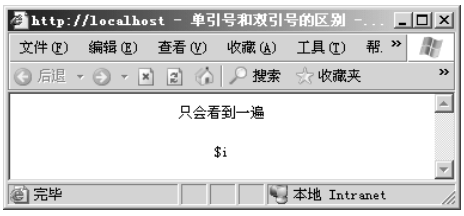


图 3.1　单引号和双引号的区别

两者之间另一处不同点是对转义字符的使用。使用单引号时，要想输出单引号，只要对单引号（'）进行转义即可，但使用双引号（"）时，还要注意“"”“$”等字符的使用。这些特殊字符都要通过转义符“\”来显示。常用的转义字符如表 3.2 所示。

表 3.2　转义字符

转 义 字 符	输　　出
\n	换行（LF 或 ASCII 字符 0x0A（10））
\r	回车（CR 或 ASCII 字符 0x0D（13））
\t	水平制表符（HT 或 ASCII 字符 0x09（9））
\\	反斜杠
\$	美元符号
\'	单引号
\"	双引号
\[0-7]{1,3}	此正则表达式序列匹配一个用八进制符号表示的字符，如\467
\x[0-9A-Fa-f]{1,2}	此正则表达式序列匹配一个用十六进制符号表示的字符，如\x9f

\n 和\r 在 Windows 系统中没有什么区别，都可以当作回车符。但在 Linux 系统中则是两种效果，在 Linux 中，\n 表示换到下一行，却不会回到行首；而\r 表示光标回到行首，但仍然在本行。如果读者使用 Linux 操作系统，可以尝试一下。

注意

如果对非转义字符使用了“\”，那么在输出时，“\”也会跟着一起被输出。

说明

定义简单字符串时使用单引号更加合适。使用双引号，PHP 将花费一些时间来处理字符串的转义和变量的解析。因此，如果没有特别的要求，定义字符串时应尽量使用单引号。

定界符（<<<）是从 PHP 4 开始支持的。在使用时后接一个标识符，然后是字符串，最后是同样的标识符结束字符串。定界符的格式如下：

```
$string = <<< str
要输出的字符串
str
```

其中，str 为指定的标识符。

【例 3.3】使用定界符输出变量的值。**（实例位置：资源包\TM\sl\3\3）**

```
<?php
    $i = '显示该行内容';                                    //声明变量$i
    echo <<< std
    这和双引号没有什么区别，\$i 同样可以被输出出来。<p>
    \$i 的内容为：$i
    std;
?>
```

运行结果如图 3.2 所示。可以看到，它和双引号没什么区别，包含的变量也被替换成实际数值。

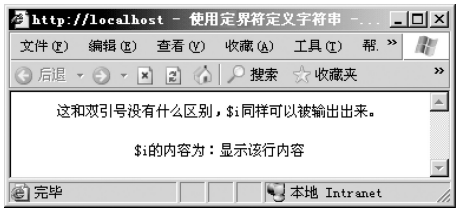


图 3.2　使用定界符定义字符串

注意

结束标识符必须单独另起一行，且不允许有空格。标识符前后有其他符号或字符时会发生错误。

3．整型（integer）

整型数据类型只能包含整数。在 32 位的操作系统中，有效的范围是−2147483648～+2147483647。整型数可以用十进制、八进制和十六进制来表示。如果用八进制，数字前面必须加 0；如果用十六进制，则需要加 0x。

注意

如果八进制中出现了非法数字（8 和 9），则后面的数字会被忽略掉。

【例 3.4】输出八进制、十进制和十六进制整数。**（实例位置：资源包\TM\sl\3\4）**

```
<?php
    $str1 = 1234567890;                                     //声明一个十进制整数
    $str2 = 0x1234567890;                                   //声明一个十六进制整数
    $str3 = 01234567890;                                    //声明一个八进制整数
    $str4 = 01234567;                                       //声明另一个八进制整数
    echo '数字 1234567890 不同进制的输出结果：<p>';
```

```
    echo '十进制的结果是：'.$str1.'<br>';                    //输出十进制整数
    echo '十六进制的结果是：'.$str2.'<br>';                  //输出十六进制整数
    echo '八进制的结果是：';
    if ($str3 == $str4) {                                    //判断$str3 和$str4 的关系
        echo '$str3 = $str4 = '.$str3;                       //如果相等，输出变量值
    } else {
        echo '$str3 != str4';                                //如果不相等，输出“$str3 != $str4”
    }
?>
```

运行结果如图 3.3 所示。

注意

如果给定的数值超出了 int 型所能表示的最大范围，将会被当作 float 型处理，这种情况称为整数溢出。同样，如果表达式的最后运算结果超出了 int 型的范围，也会返回 float 型。

4．浮点型（float）

浮点数据类型可以用来存储小数。它提供的精度比整数大得多。在 64 位的操作系统中，有效的范围是 1.7E−308～1.7E+308。在 PHP 早期版本中，浮点型的标识为 double，也叫作双精度浮点数，两者没有区别。

浮点型数据默认有两种书写格式，一种是标准格式，如 3.1415、−35.8 等；还有一种是科学记数法格式（即指数格式），如 3.58E1、849.72E−3 等。

【例 3.5】输出圆周率的近似值。采用 3 种不同的书写方法（圆周率函数、传统书写格式和科学记数法）表示，最后显示在页面上的效果都一样。**（实例位置：资源包\TM\sl\3\5）**

```
<?php
    echo '圆周率的 3 种书写方法：<p>';
    echo '第一种：pi() = '. pi() .'<p>';                     //调用 pi()函数输出圆周率
    echo '第二种：3.14159265359 = '. 3.14159265359 .'<p>';   //传统书写格式的浮点数
    echo '第三种：314159265359E-11 = '. 314159265359E-11 .'<p>';  //科学记数法格式的浮点数
?>
```

运行结果如图 3.4 所示。

图 3.3　不同进制的输出结果

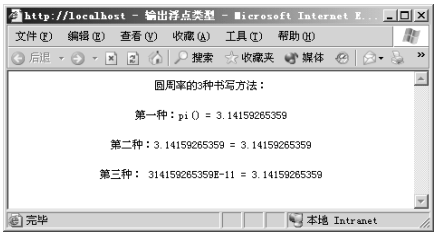


图 3.4　输出浮点类型

注意

浮点型的数值只是一个近似值，所以要尽量避免在浮点型数值之间比较大小，因为最后的结果往往是不准确的。

3.3.2　复合数据类型

复合数据类型包括两种：数组和对象，如表 3.3 所示。

表 3.3　复合数据类型

类　　型	说　　明
array（数组）	一组数据类型相同的变量的集合
object（对象）	对象是类的实例，使用 new 命令来创建

1．数组（array）

数组是一组数据的集合，它把一系列同类型数据组织起来，形成一个可操作的整体。数组中可以包括很多数据，如标量数据、数组、对象、资源以及 PHP 中支持的其他语法结构等。

数组中的每个数据都被称为一个元素，元素包括索引（键名）和值两个部分。元素的索引可以由数字或字符串组成，元素的值可以是多种数据类型。定义数组的语法格式有如下 3 种：

```
$array = array('value1', 'value2'…)
$array[key] = 'value'
$array = array(key1 => value1, key2 => value2…)
```

其中，key 是数组元素的下标，value 是数组下标所对应的元素。以下几种都是正确的声明数组格式：

```
$arr1 = array('This', 'is', 'an', 'example');
$arr2 = array(0 => 'php', 1 => 'is', 'the' => 'the', 'str' => 'best ');
$arr3[0] = 'tmpname';
```

声明数组后，数组中的元素个数还可以自由更改。只要给数组赋值，数组就会自动增加长度。在第 7 章中会详细介绍数组的相关知识。

2．对象（object）

世间万物皆为对象，对象包含方法和属性。在 PHP 中，用户可以自由使用面向过程和面向对象这两种开发方法。第 14 章中将详细介绍面向对象相关的知识。

3.3.3　特殊数据类型

特殊数据类型包括资源和空值两种，如表 3.4 所示。

表 3.4　特殊数据类型

类　　型	说　　明
resource（资源）	资源是一种特殊变量，又叫作句柄，保存了到外部资源的一个引用。资源是通过专门的函数来建立和使用的
null（空值）	特殊的值，表示变量没有值，唯一的值就是 null

1．资源（resource）

关于资源的类型，可以参考 PHP 手册后面的附录，里面有详细的介绍和说明。使用资源时，系统会自动启用垃圾回收机制，释放不再使用的资源，避免内存消耗殆尽。因此，资源很少需要手工释放。

2．空值（null）

空值，顾名思义，就是没有为该变量设置任何值。另外，空值（null）不区分大小写，null 和 NULL 效果是一样的。被赋予空值的情况有 3 种：未被赋任何值、被赋值 null、被 unset()函数处理过。

【例 3.6】被赋值为 null 的几种情况。**（实例位置：资源包\TM\sl\3\6）**

字符串 string1 被赋值为 null。string2 未被声明和赋值，所以也输出 null。string3 虽然被赋了初值，但被 unset()函数处理后，也变为 null 型。unset()函数的作用是从内存中删除变量。

```
<?php
    echo "变量(\$string1)直接赋值为 null：";
    $string1 = null;                                    //变量$string1 被赋空值
    $string3 = "str";                                   //变量$string3 被赋值 str
    if (!isset($string1))                               //判断$string1 是否为空值
        echo "string1 = null";
    echo "<p>变量(\$string2)未被赋值：";
    if (!isset($string2))                               //判断$string2 是否为空值
        echo "string2 = null";
    echo "<p>被 unset()函数处理过的变量(\$string3)：";
    unset($string3);                                    //释放$string3
    if (!isset($string3))                               //判断$string3 是否为空值
        echo "string3 = null";
?>
```

运行结果如图 3.5 所示。

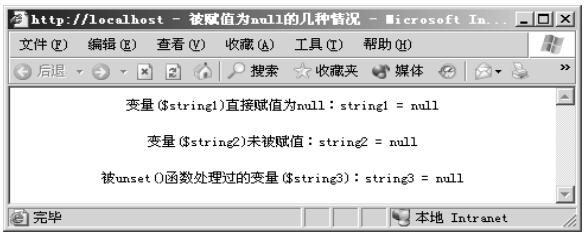


图 3.5　被赋值为 null 的几种情况

说明

is_null()函数用于判断变量是否为 null，该函数返回一个 boolean 型，如果变量为 null，则返回 true，否则返回 false。unset()函数用来销毁指定的变量。

注意

从 PHP 4 开始，unset()函数就不再有返回值，所以不要试图获取或输出 unset()。

3.3.4　数据类型转换

虽然 PHP 是弱类型语言，但有时仍然需要用到类型转换。PHP 中的类型转换和 C 语言一样，非常简单，只需在变量前加上用括号括起来的类型名称即可。允许转换的类型如表 3.5 所示。

表 3.5　类型强制转换

转换操作符	转 换 类 型	示　　例
(boolean)	转换成布尔型	(boolean)$num、(boolean)$str
(string)	转换成字符串型	(string)$boo、(string)$flo
(integer)	转换成整型	(integer)$boo、(integer)$str
(float)	转换成浮点型	(float)$str、(float)$str
(array)	转换成数组	(array)$str
(object)	转换成对象	(object)$str

注意

在进行类型转换的过程中应该注意：转换成 boolean 型时，null、0 以及未赋值的变量和数组会被转换为 false，其他的转换为 true；转换成整型时，布尔型的 false 转换为 0，true 转换为 1，浮点型的小数部分被舍去，字符型如果以数字开头就截取到非数字位，否则输出 0。

类型转换还可以通过 settype()函数来完成，该函数可以将变量转换成指定的数据类型。函数格式为：

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

其中，var 为指定的变量；type 为指定的类型，它有 7 个可选值，即 boolean、float、integer、array、null、object 和 string。如果转换成功，则返回 true，否则返回 false。

当字符串转换为整型或浮点型时，如果字符串是以数字开头的，就会先把数字部分转换为整型，再舍去后面的字符串；如果数字中含有小数点，则会取到小数点前一位。

【例 3.7】使用不同的方法对指定字符串进行类型转换。**（实例位置：资源包\TM\sl\3\7）**

```
<?php
    $num = '3.1415926r*r';                          //声明一个字符串变量
    echo '使用(integer)操作符转换变量$num 类型：';
    echo (integer)$num;                             //使用 integer 转换类型
    echo '<p>';
    echo '输出变量$num 的值：'.$num;                //输出原始变量$num
    echo '<p>';
    echo '使用 settype 函数转换变量$num 类型：';
    echo settype($num, 'integer');                  //使用 settype()函数转换类型
    echo '<p>';
    echo '输出变量$num 的值：'.$num;                //输出原始变量$num
?>
```

运行结果如图 3.6 所示。

可以看到，使用 integer 操作符能直接输出转换后的变量类型，原变量不发生任何变化。使用 settype()函数返回的是 1，也就是 true，且原变量被改变了。在实际应用中，可根据情况自行选择转换方式。

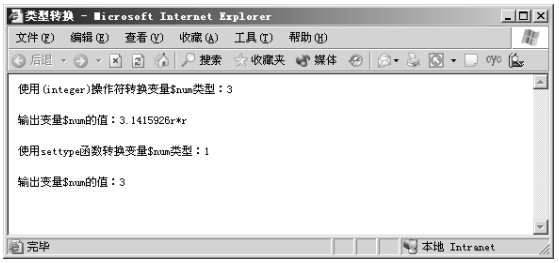


图 3.6　类型转换

3.3.5 检测数据类型

PHP 中内置了一系列检测数据类型的函数，可以对不同类型的数据进行检测，判断其是否属于某个类型，如果是则返回 true，否则返回 false。检测数据类型的函数如表 3.6 所示。

表 3.6 检测数据类型的函数

函　数	检 测 类 型	示　例
is_bool()	检查变量是否为布尔类型	is_bool(true)、is_bool(false)
is_string()	检查变量是否为字符串类型	is_string('string')、is_string(1234)
is_float/is_double()	检查变量是否为浮点类型	is_float(3.1415)、is_float('3.1415')
is_integer/is_int()	检查变量是否为整数	is_integer(34)、is_integer('34')
is_null()	检查变量是否为 null	is_null(null)
is_array()	检查变量是否为数组类型	is_array($arr)
is_object()	检查变量是否为对象类型	is_object($obj)
is_numeric()	检查变量是否为数字或数字组成的字符串	is_numeric('5')、is_numeric('bccd110')

由于检测数据类型的函数的功能和用法都是相同的，下面使用 is_numeric()函数来检测变量中的数据是否为数字，从而了解并掌握 is 系列函数的用法。

【例 3.8】检测变量是否是电话号码（即全由数字组成）。**（实例位置：资源包\TM\sl\3\8）**

```
<?php
    $boo = "043112345678";                                  //声明一个由数字组成的字符串变量
    if (is_numeric($boo))                                   //判断该变量是否由数字组成
        echo "Yes,the \$boo is a phone number: $boo!";      //如果是，输出该变量
    else
        echo "Sorry,This is an error!";                     //否则，输出错误语句
?>
```

结果为：Yes,the $boo is a phone number:043112345678!

3.4 PHP 常量

本节主要介绍 PHP 常量，包括常量的定义、使用以及预定义常量。

3.4.1 常量的定义和使用

常量就是值不可更改的量。常量值被定义后，在脚本的其他任何地方都不会再发生改变。一个常量由英文字母、下画线和数字组成，但数字不能作为首字母。

在 PHP 中使用 define()函数来定义常量，语法格式如下：

```
define(string constant_name, mixed value, case_sensitive=false)
```

该函数有 3 个参数，详细说明如表 3.7 所示。

表 3.7　define()函数的参数说明

参　数	说　明
constant_name	必选参数，指定常量名称，即标识符
value	必选参数，指定常量的值
case_sensitive	可选参数，指定是否大小写敏感。设定为 true，表示大小写不敏感

获取常量的值有两种方法：一种是使用常量名直接获取值；另一种是使用 constant()函数。

constant()函数和直接使用常量名输出的效果是一样的，优点是可以动态地输出不同的常量，在使用上要灵活很多。constant()函数的语法格式如下：

```
mixed constant(string const_name)
```

其中，const_name 为要获取常量的名称，也可为存储常量名的变量。如果成功，则返回常量的值；否则会提示错误信息，提示常量没有被定义。

要判断一个常量是否已经定义，可以使用 defined()函数，该函数的语法格式如下：

```
bool defined(string constant_name);
```

其中，constant_name 为要获取常量的名称，成功则返回 true，否则返回 false。

【例 3.9】比较 define()、constant()和 defined()函数。使用 define()函数定义一个常量，使用 constant()函数动态获取常量的值，使用 defined()函数判断常量是否已被定义。**（实例位置：资源包\TM\sl\3\9）**

```
<?php
    define("MESSAGE", "我是一名 PHP 程序员");   //定义常量 MESSAGE
    echo MESSAGE."<br>";                        //输出常量 MESSAGE
    echo Message."<br>";                        //输出 Message，因为未定义，会输出错误提示信息
    define("COUNT","我想要怒放的生命", true);    //定义常量 COUNT，大小写不敏感
    echo COUNT."<br>";                          //输出常量 COUNT
    echo Count."<br>";                      //输出常量 COUNT，因为设定大小写不敏感，Count 即 COUNT
    $name = "count";                            //将 count 赋给变量$name
    echo constant($name)."<br>";                //输出常量 COUNT
    echo (defined("MESSAGE"))."<br>";           //如果常量被定义，则返回 true，使用 echo 输出显示 1
?>
```

运行结果如图 3.7 所示。

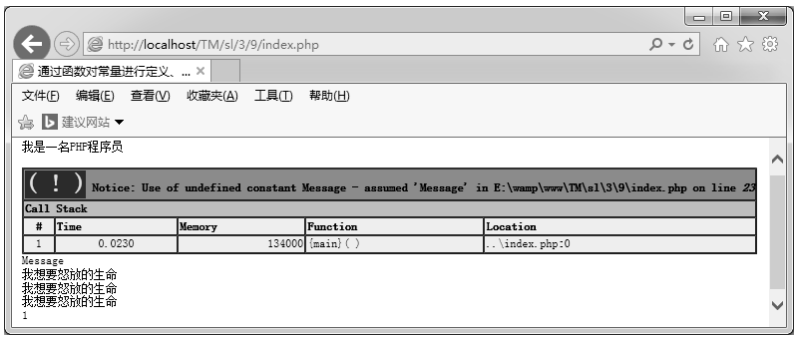


图 3.7　通过函数对常量进行定义、获取和判断

3.4.2 预定义常量

PHP 中可以使用预定义常量获取 PHP 中的信息。常用的预定义常量如表 3.8 所示。

表 3.8 PHP 的预定义常量

常 量 名	功 能
__FILE__	默认常量，显示 PHP 程序的当前文件名
__LINE__	默认常量，显示 PHP 程序的当前行数
PHP_VERSION	内置常量，显示 PHP 程序的版本，如 php6.0.0-dev
PHP_OS	内置常量，显示 PHP 解析器的操作系统，如 Windows
TRUE	该常量是一个真值（true）
FALSE	该常量是一个假值（false）
NULL	该常量是一个 null 值
E_ERROR	该常量指到最近的错误处
E_WARNING	该常量指到最近的警告处
E_PARSE	该常量指到解析语法有潜在问题处
E_NOTICE	该常量为发生不寻常处的提示，但不一定是错误处

说明

__FILE__和__LINE__中的“__”是两条下画线，而不是一条“_”。

以 E_开头的预定义常量用于 PHP 的错误调试，如需详细了解，请参考 error_ reporting() 函数。

预定义常量与用户自定义常量在使用上没有什么差别，下面来看一个例子。

【例 3.10】使用预定义常量输出 PHP 中的信息。（**实例位置：资源包\TM\sl\3\10**）

```
<?php
    echo "当前文件路径：".__FILE__;                    //输出__FILE__常量
    echo "<br>当前行数：".__LINE__;                    //输出__LINE__常量
    echo "<br>当前 PHP 版本信息：".PHP_VERSION;        //输出 PHP 版本信息
    echo "<br>当前操作系统: ".PHP_OS;                  //输出操作系统信息
?>
```

运行结果如图 3.8 所示。

说明

由于不同用户的操作系统和软件版本不同，执行本例所得的结果也不一定相同。

图 3.8 应用 PHP 预定义常量输出信息

3.5　PHP 变量

变量是指在程序执行过程中数值可以变化的量。变量通过一个名字（变量名）来标识。系统为程序中的每一个变量分配一个存储单元，变量名实质上就是计算机内存单元的命名。因此，借助变量名即可访问内存中的数据。

3.5.1　变量的赋值

和很多语言不同，在 PHP 中使用变量之前不需要先声明变量（PHP 4 之前需要声明变量），直接为变量赋值即可。PHP 中的变量名称用“$”和标识符表示。标识符由字母、数字或下画线组成，并且不能以数字开头。另外，变量名是区分大小写的。

变量赋值是指给变量赋一个具体的数据值，字符串和数值类型的变量可以通过赋值运算符“=”来实现赋值，格式如下：

```
<?php
    $name = value;
?>
```

对变量赋值时，要遵循变量的命名规则。例如，下面的变量命名是合法的：

```
<?php
    $thisCup = "oink";
    $_Class = "roof ";
?>
```

下面的变量命名则是非法的：

```
<?php
    $11112_var = 11112;                    //变量名不能以数字开头
    $@spcn = "spcn";                       //变量名不能以其他字符开头
?>
```

除了可对变量直接赋值外，还可以采用变量间赋值和引用赋值的方式定义一个变量。变量间进行赋值时，赋值后的两个变量使用各自的内存，互不干扰。引用赋值要使用“&”，是指用不同的名字访问同一个变量内容，改变其中一个变量的值时，另一个变量的值也会跟着发生变化。

【例 3.11】变量间进行赋值。**（实例位置：资源包\TM\sl\3\11）**

```
<?php
    $string1 = "mingribook";               //声明变量$string1
    $string2 = $string1;                   //使用$string1 初始化$string2
    $string1 = "mrbccd";                   //改变变量$string1 的值
    echo $string2;                         //输出变量$string2 的值
?>
```

结果为：mingribook

【例 3.12】变量$j 是变量$i 的引用，给变量$i 赋值后，$j 的值也会跟着发生变化。（**实例位置：资源包\TM\sl\3\12**）

```
<?php
    $i = "mingribook";                          //声明变量$i
    $j = & $i;                                  //使用引用赋值，这时$j 已经赋值为 mingribook
    $i = "mrbccd";                              //重新给$i 赋值
    echo $j;                                    //输出变量$j
    echo "<br>";
    echo $i;                                    //输出变量$i
?>
```

结果为：mrbccd
mrbccd

注意

变量引用和变量赋值的区别在于：变量赋值是将原变量内容复制下来，开辟一个新的内存空间来保存；而引用则是给变量的内容再起一个名字，有点类似于笔名。

3.5.2 变量的作用域

变量必须在有效范围内使用，如果超出有效范围，则变量也就失去其意义了。变量的作用域如表 3.9 所示。

表 3.9 变量的作用域

作 用 域	说 明
局部变量	在函数内部定义的变量，其作用域是所在函数
全局变量	被定义在所有函数以外的变量，其作用域是整个 PHP 文件。注意，全局变量在用户自定义函数内部是不可用的，如果希望在用户自定义函数内部使用全局变量，需要使用 global 关键字进行声明
静态变量	在函数调用结束后仍保留变量值，当再次回到其作用域时，又可以继续使用原来的值。注意，一般变量在函数调用结束后，其存储的数据值将被清除，所占的内存空间被释放。使用静态变量时，需要用关键字 static 进行声明

在函数内部定义的变量，其作用域为所在函数。如果在函数外赋值，将被认为是完全不同的另一个变量。在退出声明变量的函数时，该变量及相应的值会被清除。

【例 3.13】比较在函数内赋值的变量（局部变量）和在函数外赋值的变量（全局变量）。（**实例位置：资源包\TM\sl\3\13**）

```
<?php
    $example = "在……函数外";                    //声明全局变量
    function example() {
        $example = "……在函数内……";              //声明局部变量
        echo "在函数内输出的内容是：$example.<br>";  //输出局部变量
    }
```

```
    example();                                          //调用函数，输出变量值
    echo "在函数外输出的内容是：$example.<br>";          //输出全局变量
?>
```

运行结果如图 3.9 所示。

静态变量在很多地方都能用到。例如，在博客中使用静态变量记录来访人数，在聊天室中使用静态变量来记录用户的聊天内容等。

【例 3.14】使用静态变量和普通变量同时输出一个数据，查看两者的功能有什么不同。**（实例位置：资源包\TM\sl\3\14）**

```
<?php
    function zdy() {                                    //定义函数 zdy()
        static $message = 0;                            //初始化静态变量
        $message += 1;                                  //静态变量加 1
        echo $message." ";   }                          //输出静态变量
    function zdy1() {                                       //定义函数 zdy()
        $message = 0;                                   //声明函数内部变量（局部变量）
        $message += 1;                                  //局部变量加 1
        echo $message." ";   }                          //输出局部变量
    for ($i = 0; $i < 10; $i++ )    zdy();              //循环调用函数 zdy()，输出 1～10
    echo "<br>";
    for ($i = 0; $i < 10; $i++ )    zdy1();             //循环调用函数 zdy1()，输出 10 个 1
    echo "<br>";
?>
```

运行结果如图 3.10 所示。

图 3.9　局部变量的使用

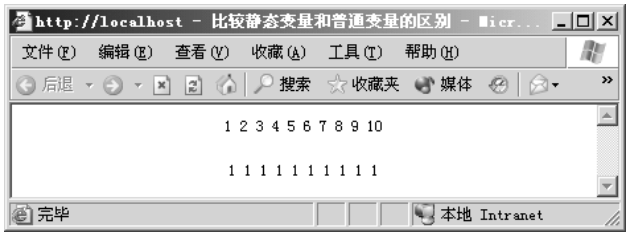


图 3.10　比较静态变量和普通变量的区别

自定义函数 zdy()输出的是 1～10 共 10 个数字，而 zdy1()函数输出的是 10 个 1。自定义函数 zdy()含有静态变量，而函数 zdy1()是一个普通变量。初始化都为 0，再分别使用 for 循环调用两个函数，结果是静态变量的函数 zdy()在被调用后保留了$message 中的值，而静态变量的初始化只是在第一次遇到时被执行，以后就不再对其进行初始化操作了，将会略过第 3 行代码不执行；而普通变量的函数 zdy1()在被调用后，其变量$message 失去原来的值，重新被初始化为 0。

全局变量虽然可以在 PHP 文件的任何地方访问，但是在用户自定义函数内部是不可用的。要想在用户自定义函数内部使用全局变量，要使用 global 关键字对其声明。

【例 3.15】在自定义函数中输出局部变量和全局变量的值。**（实例位置：资源包\TM\sl\3\15）**

```
<?php
    $hr = "黄蓉";                                       //声明全局变量$hr
    function lxt() {
```

```
        $gj = "郭靖";                                    //声明局部变量$gj
        echo $gj."<br>";                                 //输出局部变量的值
        global $hr;                                      //利用关键字 global 在函数内部定义全局变量
        echo $hr."<br>";                                 //输出全局变量的值
    }
    lxt();
?>
```

结果为：郭靖
　　黄蓉

3.5.3 可变变量

可变变量是一种独特的变量，它允许动态改变某个变量的名称。其工作原理是该变量的名称由另外一个变量的值来确定，实现过程就是在变量的前面再多加一个美元符号“$”。

【例 3.16】使用可变变量动态改变变量的名称。首先定义两个变量$a 和$b，并且输出变量$a 的值，然后使用可变变量来改变变量$a 的名称，最后输出改变名称后的变量值。**（实例位置：资源包\TM\sl\3\16）**

```
<?php
    $a = "b";                                            //声明变量$a
    $b = "我喜欢 PHP";                                   //声明变量$b
    echo $a;                                             //输出变量$a
    echo "<br>";
    echo $$a;                                            //通过可变变量输出$b 的值
?>
```

结果为：b
　　我喜欢 PHP

3.5.4 PHP 预定义变量

PHP 提供了很多非常实用的预定义变量，通过这些预定义变量可以获取用户会话、用户及本地操作系统的环境等信息。常用的预定义变量如表 3.10 所示。

表 3.10　预定义变量

变量的名称	说　明
$_SERVER['SERVER_ADDR']	当前运行脚本所在的服务器 IP 地址
$_SERVER['SERVER_NAME']	当前运行脚本所在的服务器主机名称。如果该脚本运行在一个虚拟主机上，则该名称由虚拟主机所设置的值决定
$_SERVER['REQUEST_METHOD']	访问页面时的请求方法，包括 GET、HEAD、POST、PUT 等。如果请求的方式是 HEAD，PHP 脚本将在送出头信息后中止（这意味着在产生任何输出后，不再有输出缓冲）
$_SERVER['REMOTE_ADDR']	正在浏览当前页面的用户的 IP 地址

续表

变量的名称	说　　明
$_SERVER['REMOTE_HOST']	正在浏览当前页面的用户的主机名。反向域名解析基于该用户的 REMOTE_ADDR
$_SERVER['REMOTE_PORT']	用户连接到服务器时所使用的端口
$_SERVER['SCRIPT_FILENAME']	当前执行脚本的绝对路径名。注意：如果脚本在 CLI 中被执行，作为相对路径，如 file.php 或者.../file.php，$_SERVER['SCRIPT_FILENAME']将包含用户指定的相对路径
$_SERVER['SERVER_PORT']	服务器所使用的端口，默认为 80。如果使用 SSL 安全连接，则这个值为用户设置的 HTTP 端口
$_SERVER['SERVER_SIGNATURE']	包含服务器版本和虚拟主机名的字符串
$_SERVER['DOCUMENT_ROOT']	当前运行脚本所在的文档根目录，在服务器配置文件中定义
$_COOKIE	通过 HTTPCookie 传递到脚本的信息。这些 cookie 多数是由执行 PHP 脚本时通过 setcookie()函数设置的
$_SESSION	包含与所有会话变量有关的信息，主要应用于会话控制和页面之间值的传递
$_POST	包含通过 POST 方法传递的参数的相关信息，主要用于获取通过 POST 方法提交的数据
$_GET	包含通过 GET 方法传递的参数的相关信息，主要用于获取通过 GET 方法提交的数据
$GLOBALS	由所有已定义全局变量组成的数组。变量名就是该数组的索引，它可以称得上是所有超级变量的超级集合

3.6　PHP 运算符

运算符是用来对变量、常量或数据进行计算的符号，它对一个值或一组值执行一个指定的操作。PHP 的运算符主要包括算术运算符、字符串运算符、赋值运算符、递增/递减运算符、位运算符、逻辑运算符、比较运算符和条件运算符，这里只介绍一些常用的运算符。

3.6.1　算术运算符

算术运算符（Arithmetic Operators）是处理四则运算的符号，在数字处理中用得最多。常用的算术运算符如表 3.11 所示。

表 3.11　常用的算术运算符

名　　称	操　作　符	示　　例	名　　称	操　作　符	示　　例
加法运算	+	$a + $b	除法运算	/	$a / $b
减法运算	-	$a-$b	取余数运算	%	$a % $b
乘法运算	*	$a * $b			

说明

在算术运算符中使用%求余，如果被除数（$a）是负数，那么取得的结果也是一个负值。

【例 3.17】算术运算符应用。（**实例位置：资源包\TM\sl\3\17**）

```
<?php
    $a = -100;                                  //声明变量$a
    $b = 50;                                    //声明变量$b
    $c = 30;                                    //声明变量$c
    echo "\$a = ".$a.",";                       //输出变量$a
    echo "\$b = ".$b.",";                       //输出变量$b
    echo "\$c = ".$c."<p>";                     //输出变量$c
    echo "\$a + \$b = ".($a + $b)."<br>";       //计算变量$a 加$b 的值
    echo "\$a - \$b = ".($a - $b)."<br>";       //计算变量$a 减$b 的值
    echo "\$a * \$b = ".($a * $b)."<br>";       //计算$a 乘$b 的值
    echo "\$a / \$b = ".($a / $b)."<br>";       //计算$a 除以$b 的值
    echo "\$a % \$c = ".($a % $c)."<br>";       //计算$a 除以$c 的余数
?>
```

运行结果如图 3.11 所示。

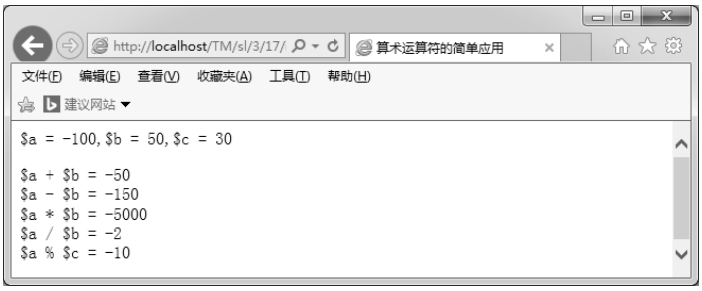


图 3.11　算术运算符的简单应用

3.6.2　字符串运算符

字符串运算符“.”可将两个字符串连接起来，结合成一个新的字符串。学习过 C 或 Java 语言的读者应注意，PHP 中“+”只能用作赋值运算符，而不能用作字符串运算符。

【例 3.18】对比“.”和“+”运算符的区别。（**实例位置：资源包\TM\sl\3\18**）

使用“.”时，变量$m 和$n 两个字符串将组成一个新的字符串 3.1415926r*r1。使用“+”时，PHP 会认为这是一次加法运算。如果“+”的两边有字符类型，则会自动转换为整型。例如，如果是字母，则输出为 0；如果是以数字开头的字符串，则会截取字串头部的数字，再进行运算。

```
<?php
    $n = "3.1415926r * r";                  //声明一个字符串变量，以数字开头
    $m = 1;                                 //声明一个整型变量
    $nm = $n.$m;                            //使用“.”运算符将两个变量连接起来
    echo $nm."<br>";
    $mn = $n + $m;                          //使用“+”运算符计算两个变量的和
```

```
    echo $mn."<br>";
?>
```

结果为：3.1415926r*r1
　　4.1415926

3.6.3　赋值运算符

赋值运算符包括基本赋值运算符“=”和 6 个复合赋值运算符，如表 3.12 所示。

表 3.12　常用赋值运算符

操　作	符　号	示　例	展 开 形 式	意　义
赋值	=	$a = 3	$a = 3	将右边的值赋给左边
加	+=	$a += 2	$a = $a+2	将右边的值加到左边
减	-=	$a -= 3	$a = $a-3	将右边的值减到左边
乘	*=	$a *= 4	$a = $a * 4	将左边的值乘以右边
除	/=	$a /= 5	$a = $a / 5	将左边的值除以右边
连接字符	.=	$a .= 'b'	$a = $a.'b'	将右边的字符加到左边
取余数	%=	$a %= 5	$a = $a % 5	将左边的值对右边取余数

3.6.4　递增或递减运算符

算术运算符适合在有两个或者两个以上不同操作数的场合使用，当只有一个操作数时，要体现其增减变化，可以使用递增运算符“++”或者递减运算符“--”。

递增或递减运算符有两种使用方法。一种是将运算符放在变量前面，即先将变量作加 1 或减 1 的运算，然后再将值赋给原变量，叫作前置递增或递减运算符；另一种是将运算符放在变量后面，即先返回变量的当前值，然后变量的当前值作加 1 或减 1 的运算，叫作后置递增或递减运算符。

【例 3.19】定义两个变量，分别利用递增和递减运算符进行操作。（**实例位置：资源包\TM\sl\3\19**）

```
<?php
    $a = 6;
    $b = 9;
    echo "\$a = $a, \$b = $b<p>";
    echo "\$a++ = " . $a++ ."<br>";        //先返回$a 的当前值，然后$a 的当前值加 1
    echo "运算后\$a 的值: ".$a."<p>";
    echo "++\$b = " . ++$b ."<br>";        //$b 的当前值先加 1，然后返回新值
    echo "运算后\$b 的值: ".$b;
    echo "<hr><p>";
    echo "\$a-- = " . $a-- ."<br>";        //先返回$a 的当前值，然后$a 的当前值减 1
    echo "运算后\$a 的值: ".$a."<p>";
    echo "\$b = " . --$b ."<br>";          //$b 的当前值先减 1，然后返回新值
    echo "运算后\$b 的值: ".$b;
?>
```

运行结果如图 3.12 所示。

图 3.12　递增和递减运算符

3.6.5　位运算符

位运算符可将两个数的二进制位从低位到高位对齐后进行位与、位或、位异或运算，也可对一个数的二进制位执行向左或向右移位以及取反运算。PHP 中的位运算符如表 3.13 所示。

表 3.13　位运算符

符　号	作　用	示　例	符　号	作　用	示　例
&	按位与	$m & $n	~	按位取反	~$m
|	按位或	$m | $n	<<	向左移位	$m << $n
^	按位异或	$m ^ $n	>>	向右移位	$m >> $n

【例 3.20】使用位运算符对变量值进行位运算操作。（**实例位置：资源包\TM\sl\3\20**）

```
<?php
    $m = 8;
    $n = 12;
    $mn = $m & $n;                    //8 和 12 位与运算，结果为 8
    echo $mn ." ";
    $mn = $m | $n;                    //8 和 12 位或运算，结果为 12
    echo $mn ." ";
    $mn = $m ^ $n;                    //8 和 12 位异或运算，结果为 4
    echo $mn ." ";
    $mn = ~$m;                        //对 8 进行位取反运算，结果为-9
    echo $mn ." ";
?>
```

结果为：8　12　4　−9

3.6.6　逻辑运算符

逻辑运算符用来组合逻辑运算的结果，是程序设计中一组非常重要的运算符。PHP 中的逻辑运算

符如表 3.14 所示。

表 3.14　PHP 的逻辑运算符

运　算　符	示　　例	结　　果
&&或 and（逻辑与）	$m and $n	当$m 和$n 都为真时，结果为真，否则为假
||或 or（逻辑或）	$m || $n	只要$m 为真或者$n 为真，结果就为真
xor（逻辑异或）	$m xor $n	当$m 和$n 一真一假时，结果为真
!（逻辑非）	!$m	当$m 为假时，结果为真；当$m 为真时，结果为假

在逻辑运算符中，逻辑与和逻辑或一共包括 4 种运算符号（&&、and、||和 or），它们分属于不同的优先级，从高到低分别为&&、||、and 和 or。

【例 3.21】使用逻辑或运算符号“||”和 or 进行相同的判断，比较输出的结果。**（实例位置：资源包\TM\sl\3\21）**

```
<?php
    $i = true;                  //声明一个布尔型变量$i，赋值为真
    $j = true;                  //声明一个布尔型变量$j，赋值为真
    $z = false;                 //声明一个布尔型变量$z，赋值为假
    if ($i or $j and $z)        //用 or、and 进行判断
        echo "true";            //如果 if 表达式为真，输出 true
    else
        echo "false";           //否则输出 false
    echo "<br>";
    if ($i || $j and $z)        //用||、and 进行判断
        echo "true";            //如果表达式为真，输出 true
    else
        echo "false";           //如果表达式为假，输出 false
?>
```

因为同一逻辑结构的两个运算符“||”和 or 的优先级不同，输出的结果也不同。

结果为：true

　　false

注意

两个 if 语句的判断表达式中，除了 or 和“||”不同外，其他完全一样，但最后的结果却正好相反。在实际应用中要多留心这样的细节。

3.6.7　比较运算符

比较运算符就是对变量或表达式的结果进行大小、真假等比较，如果比较结果为真，返回 true；如果比较结果为假，返回 false。PHP 中的比较运算符如表 3.15 所示。

表 3.15　PHP 的比较运算符

运　算　符	说　　明	示　　例	运　算　符	说　　明	示　　例
<	小于	$m<$n	==	相等	$m==$n
>	大于	$m>$n	!=	不等	$m!=$n
<=	小于或等于	$m<=$n	===	恒等	$m=== $n
>=	大于或等于	$m>=$n	!==	非恒等	$m!==$n

其中，不太常见的是“===”和“!==”。“$a === $b”说明$a 和$b 不仅数值相等，两者的类型也一样。“!==”和“===”的意义相近，“$a !== $b”说明$a 和$b 或者数值不等，或者类型不同。

【例 3.22】使用比较运算符对变量值进行比较。变量$value="100"，类型为字符串型，将$value 与数字 100 进行比较。var_dump()函数是系统函数，作用是输出变量的相关信息。**（实例位置：资源包\TM\sl\3\22）**

```
<?php
    $value = "100";                          //声明一个字符串变量$value
    echo "\$value = \"$value\"";
    echo "<p>\$value == 100: ";
    var_dump($value == 100);                 //结果为 boolean true
    echo "<p>\$value == true: ";
    var_dump($value == true);                //结果为 boolean true
    echo "<p>\$value != null: ";
    var_dump($value != null);                //结果为 boolean true
    echo "<p>\$value == false: ";
    var_dump($value == false);               //结果为 boolean false
    echo "<p>\$value === 100: ";
    var_dump($value === 100);                //结果为 boolean false
    echo "<p>\$value === true: ";
    var_dump($value === true);               //结果为 boolean false
    echo "<p>(10/2.0 !== 5): ";
    var_dump(10/2.0 !== 5);                  //结果为 boolean true
?>
```

运行结果如图 3.13 所示。

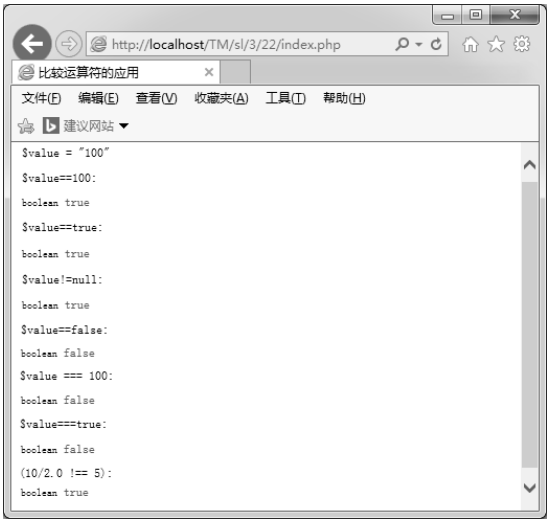


图 3.13　比较运算符的应用

3.6.8　条件运算符

条件运算符（? :），也称为三目运算符，用于根据一个表达式在另外两个表达式中选择一个，而不是用来在两个语句或者程序中选择。条件运算符最好放在括号里使用。

【例 3.23】应用条件运算符实现简单的判断功能。如果正确，则输出“条件运算”，否则输出“没有该值”。**（实例位置：资源包\TM\sl\3\23）**

```
<?php
    $value = 100;                                          //声明一个整型变量
    echo ($value==true) ? "条件运算" : "没有该值";          //对整型变量进行判断
?>
```

结果为：条件运算

3.6.9　运算符的优先级

运算符的优先级是指在应用中哪一个运算符先计算，哪一个运算符后计算。在 PHP 中，优先级高的运算先执行，优先级低的运算后执行，同一优先级的运算按照从左到右的顺序执行。也可以像四则运算那样使用小括号提升优先级，括号内的运算最先执行。表 3.16 从高到低列出了 PHP 中各运算符的优先级，同一行中的运算符具有相同优先级，此时它们的结合方向决定了求值顺序。

表 3.16　运算符的优先级

运　算　符	描　　述
clone new	clone 和 new
[	array()
++，—	递增/递减运算符
~ - (int) (float) (string) (array) (object) (bool) @	类型
instanceof	类型
!	逻辑操作符
* / %	算术运算符
+ − .	算术运算符和字符串运算符
<< >>	位运算符
< <= > >= <>	比较运算符
== != === !==	比较运算符
&	位运算符和引用
^	位运算符
|	位运算符
&&	逻辑运算符
||	逻辑运算符
?:	条件运算符
= += -= *= /= .= %= &= |= ^= <<= >>=	赋值运算符
and	逻辑运算符
xor	逻辑运算符
or	逻辑运算符
,	逗号运算符

这么多的级别，一次性全都记住是不现实的，也没有必要。如果写的表达式真的很复杂，而且包含了较多的运算符，不妨多使用括号，以减少出现逻辑错误的可能。例如：

```
<?php
    $a and (($b != $c) or (5 * (50 – $d)))
?>
```

3.7 PHP 表达式

表达式是构成 PHP 程序语言的基本元素，最基本的表达式形式是常量和变量。如$m=20，即表示将值 20 赋给变量$m。

简单但精确定义一个表达式的方式就是“任何有值的东西”。例如：

```
<?php
    12;
    $a = "word";
?>
```

这是由两个表达式组成的脚本，即 12 和$a="word"。此外，还可以进行连续赋值，例如：

```
<?php
    $b = $a = 5;
?>
```

因为 PHP 赋值操作的顺序是由右到左的，所以变量$b 和$a 都被赋值 5。

在 PHP 的代码中，使用分号“;”来区分表达式。可以这样理解，一个表达式再加上一个分号，就是一条 PHP 语句。

应用表达式能够做很多事情，如调用一个数组，创建一个类，给变量赋值等。表达式也可以包含在括号内。

注意

在编写程序时，注意表达式后面的分号“;”不要漏写。

3.8 PHP 函数

在 Web 开发过程中，经常要重复执行某种操作或处理，如数据查询、字符操作等。如果每个操作都要重新编写一次代码，不仅程序员会头痛不已，程序的运行效果也会大打折扣。使用 PHP 函数可让这些问题迎刃而解，下面就来介绍 PHP 函数的相关知识。

3.8.1 定义和调用函数

函数，就是将一些重复使用到的功能写在一个独立的代码块中，在需要时单独调用。创建函数的基本语法格式如下：

```
function fun_name($str1, $str2, …, $strn) {
    fun_body;
}
```

其中，function 为声明自定义函数时必须使用到的关键字；fun_name 为自定义函数的名称；$str1, $str2, …, $strn 为函数的参数；fun_body 为自定义函数的主体，是功能实现部分。

当函数被定义好后，所要做的就是调用这个函数。调用函数的操作十分简单，只需要引用函数名并赋予正确的参数，即可完成函数的调用。

【例 3.24】定义函数 example()，计算传入的参数的平方。（**实例位置：资源包\TM\sl\3\24**）

```
<?php
    /*自定义求平方函数*/
    function example($num) {
        echo "$num * $num = ".$num * $num;          //输出计算后的结果
    }
    example(10);                                    //调用函数
?>
```

结果为：10 * 10 = 100

3.8.2　在函数间传递参数

调用函数时，需要向函数传递参数，被传入的参数称为实参。而函数定义时的参数称为形参。参数传递的方式有值传递、引用传递和默认参数 3 种。

1．值传递方式

将实参的值复制到对应的形参中，在函数内部的操作针对形参进行，操作的结果不会影响到实参，即函数返回后，实参的值不会改变。

【例 3.25】定义函数 example()，将传入的参数值做一些运算后再输出。接着在函数外部定义变量 $m，即要传入的参数。最后调用函数 example($m)，输出函数的返回值$m 和变量$m 的值。（**实例位置：资源包\TM\sl\3\25**）

```
<?php
    function example($m) {                          //定义一个函数，传递参数$m 的值
        $m = $m * 5 + 10;
        echo "在函数内：\$m = ".$m;                  //输出形参的值
    }
    $m = 1;
    example($m);                                    //传递值，将$m 的值传递给形参$m
    echo "<p>在函数外  \$m = $m <p>";               //实参的值没有发生变化，输出 m=1
?>
```

运行结果如图 3.14 所示。

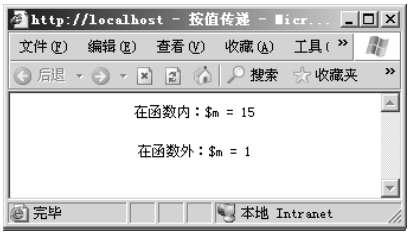


图 3.14　按值传递方式

2．引用传递方式

引用传递就是将实参的内存地址传递到形参中。这时，在函数内部的所有操作都会影响到实参的值，返回后，实参的值会发生变化。引用传递方式就是传值时在原基础上加“&”即可。

【例 3.26】仍然是例 3.25 中的代码，不同的地方就是多一个

“&”。（**实例位置：资源包\TM\sl\3\26**）

```
<?php
    function example(&$m) {                          //定义一个函数，同时传递参数$m 的地址
        $m = $m * 5 + 10;
        echo "在函数内：\$m = ".$m;                  //输出形参的值
    }
    $m = 1;
    example(&$m);                                    //传递值，将$m 的值传递给形参$m
    echo "<p>在函数外：\$m = $m <p>" ;                //实参的值发生变化，输出 m=15
?>
```

运行结果如图 3.15 所示。

图 3.15　按引用传递方式

3．默认参数（可选参数）方式

还有一种默认参数的方式，即可选参数。可以指定某个参数为可选参数，将可选参数放在参数列表末尾，并且给它指定一个默认值。

【例 3.27】使用可选参数实现简单的价格计算功能。设置自定义函数 values()的参数 $tax 为可选参数，其默认值为空。第一次调用该函数，并且给参数$tax 赋值 0.25，输出价格；第二次调用该函数，不给参数$tax 赋值，输出价格。（**实例位置：资源包\TM\sl\3\27**）

```
<?php
    function values($price, $tax=0) {                //定义一个函数，其中的参数$tax 初始值为 0
        $price = $price + ($price * $tax);           //声明一个变量$price，等于两个参数的运算结果
        echo "价格:$price<br>";                      //输出价格
    }
    values(100, 0.25);                               //为可选参数赋值 0.25
    values(100);                                     //没有给可选参数赋值
?>
```

结果为：价格:125
　　价格:100

注意

当使用默认参数时，默认参数必须放在非默认参数的右侧，否则函数可能出错。

3.8.3　从函数中返回值

通常，函数将返回值传递给调用者的方式是使用关键字 return。return 将函数的值返回给函数的调

用者，即将程序控制权返回到调用者的作用域。如果在全局作用域内使用 return 关键字，那么将终止脚本的执行。

【例 3.28】使用 return 关键字返回一个操作数。定义函数 values()，作用是输入物品的单价、重量，然后计算总金额，最后输出商品的价格。（**实例位置：资源包\TM\sl\3\28**）

```
<?php
    function values($price, $weight=0.45) {          //定义一个函数，函数中的$weight 参数有默认值
        $price = $price + ($price * $weight);        //计算物品金额
        return $price;                               //返回金额
    }
    echo values(100);                                //调用函数
?>
```

结果为：145

return 语句只能返回一个参数，也即只能返回一个值，不能一次返回多个值。如果要返回多个结果，就要在函数中定义一个数组，将返回值存储在数组中后再返回。

3.8.4　变量函数

PHP 支持变量函数。下面通过一个实例来介绍变量函数的具体应用。

【例 3.29】定义 3 个函数，接着声明一个变量，通过变量访问不同的函数。（**实例位置：资源包\TM\sl\3\29**）

```
<?php
    function come() {                                //定义 come()函数
        echo "来了<p>";
    }
    function go($name = "jack") {                    //定义 go()函数
        echo $name."走了<p>";
    }
    function back($string)                           //定义 back()函数
    {
        echo "又回来了，$string<p>";
    }
    $func = "come";                                  //声明一个变量，将变量赋值为 come
    $func();                                         //使用变量函数来调用函数 come()
    $func = "go";                                    //重新给变量赋值
    $func("Tom");                                    //使用变量函数来调用函数 go()
    $func = "back";                                  //重新给变量赋值
    $func("Lily");                                   //使用变量函数来调用函数 back()
?>
```

运行结果如图 3.16 所示。可以看到，函数的调用是通过改变变量名来实现的，在变量名后加上一对小括号，PHP 将自动寻找与变量名相同的函数，并执行它。如果找不到对应的函数，系统将会报错。该特性可用于实现回调函数和函数表等。

图 3.16　变量函数

3.9　PHP 编码规范

很多初学者对编码规范不以为然，认为对程序开发没有什么帮助，甚至认为遵循规范会影响自己学习和开发的进度。这种想法是很危险的。

如今的 Web 开发几乎不再可能一个人就完成全部，尤其是一些大型的项目，要十几人，甚至几十人才能共同完成。在开发过程中，难免会有新的开发人员参与进来，那么这个新的开发人员在阅读前任留下的代码时，就会有问题了——这个变量起到什么作用？那个函数实现什么功能？TmpClass 类在哪里被使用到了？……诸如此类的问题会层出不穷。这时，编码规范的重要性就体现出来了。

3.9.1　什么是编码规范

以 PHP 开发为例，所谓编码规范，就是指融合开发人员长时间积累下来的经验，形成的一种良好、统一的编程风格。这种良好、统一的编程风格会在团队开发或二次开发时起到事半功倍的效果。编码规范是一种总结性的说明和介绍，并不是强制性的规则。从长远的项目发展以及团队效率来考虑，遵守编码规范是十分必要的。

遵守编码规范的好处如下。

- ☑ 编码规范是对团队开发成员的基本要求。
- ☑ 开发人员可以了解任何代码，厘清程序的状况。
- ☑ 提高程序的可读性，有利于相关设计人员交流，提高软件质量。
- ☑ 防止刚接触 PHP 的人出于节省时间的需要，自创一套风格并养成终生的习惯。
- ☑ 有助于程序的维护，降低软件成本。
- ☑ 有利于团队管理，实现团队后备资源的可重用。

3.9.2　PHP 的书写规则

1. 缩进

使用制表符（Tab 键）对不同层级的代码进行缩进，缩进单位为 4 个空格左右。如果开发工具有多种，则需要在开发工具中统一进行设置。

2．大括号{ }

建议将一对大括号整体放到关键字的下方，保持同列。一般推荐使用此种方式。例如：

```
if ($expr)
{
    …
}
```

也可以首括号与关键词同行，尾括号与关键字同列。例如：

```
if ($expr) {
    …
}
```

3．关键字、小括号、函数、运算符

（1）小括号和关键字（如 if、for 等）间，使用空格进行分隔。例如：

```
if ($expr) {                    //if 和“(”之间有一个空格
    …
}
```

（2）函数后的小括号要和函数名紧密相连，这样可以有效区分 PHP 关键字和函数。例如：

```
round($num)                     //round 和“(”之间没有空格
```

（3）运算符与两边的变量或表达式间要有一个空格（字符连接运算符“.”除外）。例如：

```
while ($boo == true) {          //$boo 和“==”，true 和“==”之间都有一个空格
    …
}
```

（4）当代码段较长时，应在关键位置加入空行，以免阅读疲劳。注意，两个代码块间只使用一个空行，禁止使用多行。

（5）尽量不要在 return 返回语句中使用小括号。例如：

```
return 1;                       //除非是必要，否则不需要使用小括号
```

3.9.3　PHP 的命名规范

首先，类名、函数名、变量名等都必须见名知意，简单易懂，避免使用模棱两可的命名。除此之外，还有一些细致的要求，一起来学习下。

（1）类命名的要求和规范如下。

☑　使用大写字母作为词的分隔，其他字母均使用小写。

☑　名字的首字母使用大写。

☑　不要使用下画线（_）。

例如，Name、SuperMan、BigClassObject 等都是合理的类命名。

（2）类属性命名的要求和规范如下。

☑ 属性命名应该以字符 m 为前缀。

☑ 前缀 m 后采用与类命名一致的规则。

☑ m 总是在名字的开头起修饰作用，就像以 r 开头表示引用一样。

例如，mValue、mLongString 等都是合适的类属性命名。

（3）方法命名的要求和规范如下。

方法用于执行一个动作，达到某个目的，因此方法名应清晰说明方法是干什么用的。一般名称的前缀和后缀都有一定的规律，如 Is（判断）、Get（得到）、Set（设置）等。

方法的命名规范和类命名是一致的。例如：

```
class StartStudy {                          //设置类
    $mLessonOne = "";                       //设置类属性
    $mLessonTwo = "";                       //设置类属性
    function GetLessonOne() {               //定义方法，得到属性 mLessonOne 的值
        …
    }
}
```

（4）方法中参数命名的要求和规范如下。

☑ 第一个字符使用小写字母。

☑ 首字符后的所有字符参照类命名规则，首字符大写。例如：

```
class EchoAnyWord {
    function EchoWord($firstWord, $secondWord) {
        …
    }
}
```

（5）变量命名的要求和规范如下。

☑ 所有字母都使用小写。

☑ 使用“_”作为每个词的分界。

例如，$msg_error、$chk_pwd 等都是合适的变量命名。

（6）引用变量前要带前缀 r。例如：

```
class Example {
    $mExam = "";
    function SetExam(&$rExam) {
        …
    }
    function &rGetExam() {
        …
    }
}
```

（7）全局变量前应带前缀 g，如 global $gTest、global $g 等。

（8）常量/全局常量应该全部使用大写字母，单词之间用“_”来分隔。例如：

```
define('DEFAULT_NUM_AVG', 90);
define('DEFAULT_NUM_SUM', 500);
```

（9）静态变量前应带前缀 s，如：

```
static $sStatus = 1;
```

（10）函数命名时，所有单词都要使用小写字母，单词间使用“_”进行分隔。例如：

```
function this_good_idea() {
    …
}
```

注意，以上各种命名规则可以组合起来使用，例如：

```
class OtherExample {
    $msValue = "";                    //该参数既是类属性，又是静态变量
}
```

说明

这里介绍的只是一些简单的书写和名称规则，如果想了解更多的编码规范，可以参考 Zend_Framework 中文参考手册。

3.10　实践与练习

1．动态网页的特点是能够人机交互，但有时却需要限制用户的输入。使用 PHP 函数判断输入（这里先假定一个变量）数据是否符合下列要求：输入必须为全数字，输入数字的长度不允许超过 25，并且输入不允许为空。注：获取字符串长度函数为 strlen(string)。**（答案位置：资源包\TM\sl\3\30）**

2．获取当前访问者的计算机信息，如 IP、端口号等。**（答案位置：资源包\TM\sl\3\31）**

3．PHP 的输出语句有 echo、print、printf、print_r。尝试使用这 4 个语句输出数据，看它们之间有什么不同。**（答案位置：资源包\TM\sl\3\32）**