第 3 章　PHP 语言基础

学习目标：

本章介绍 PHP 语言的基础知识，包括基本语法、数据类型、常量与变量、运算与表达式，通过大量实例帮助学生学习并理解。通过本章的学习，应可对 PHP 语言有较为系统的了解，掌握 PHP 的语言基础，并能运用本章的知识进行简单的编程。本章学习要求如表 3-1 所示。

表 3-1　本章学习要求

知 识 要 点	能 力 要 求	相 关 知 识
PHP 入门	掌握 PHP 的标记风格、程序注释、在 HTML 中嵌入 PHP	XML、ASP、Script
数据类型	掌握 PHP 基本数据类型及数据类型转换，并能简单应用	强类型语言
常量与变量	掌握常量和变量的定义、应用范围及命名规则	转义字符
运算符与表达式	熟练使用 PHP 的运算符与表达式	优先级

在第 1 章中，读者对 PHP 已经有一定了解，但是编写 PHP 程序需要对它的语法有一定的了解。每一种语言都有它的语法特点，PHP 主要的语法特点是，所有 PHP 程序代码都是被放置在页面文件中的，一般 PHP 程序被放置在以“php”结尾的页面文件中。当然，也可以通过修改 PHP 的配置文件来更改页面的后缀名称。其次，PHP 程序代码必须放置在“<?php”标记与“?>”标记中间。还有在 PHP 语言中，“;”用来分隔单条 PHP 语句，这与 Java 语言及 C 语言中的用法是一致的。在实际开发过程中，每一条 PHP 语句都必须以分号结束，否则会出现错误。

3.1　PHP 入门

在介绍 PHP 的基本语法前，首先介绍 PHP 独特的标记风格、程序注释及如何在 HTNL 中嵌入 PHP。

3.1.1　PHP 标记风格

用户已经知道，要想让 Web 服务器能够区分 PHP 代码与普通 HTML 代码，就要将 PHP 代码放在特殊的标记内，PHP 共提供了以下 4 种不同的标记风格。

1. 短标记风格

使用短标记风格的 PHP 代码如下：

```
<?
    echo "Hello,everybody!";
?>
```

这种标记风格最简单，如果想使用短标记风格开发 PHP 程序，则必须将 PHP 配置文

件 php.ini 中的 short_open_tag 选项值设置为 on。不过在一般情况下用户不建议使用这种标记风格。

2. XML 标记风格

使用 XML 标记风格的 PHP 代码如下：

```
<?php
    echo "Hello,everybody! ";
?>
```

这种标记风格可以应用于不同的服务器环境。

3. ASP 标记风格

使用 ASP 标记风格的 PHP 代码如下：

```
<%
    echo "Hello,students!";
% >
```

这种标记风格与 ASP 或 ASP.NET 中的标记风格相同。如果习惯于 ASP 风格，则可以使用这种标记方式；如果想使用 ASP 标记风格开发 PHP 程序，则必须保证服务器配置文件 php.ini 中的 asp_tags 设置为 on。

4. Script 标记风格

使用 Script 标记风格的 PHP 代码如下：

```
<SCRIPT LANGUAGE='php'>
    echo " Hello,students!";
</SCRIPT>
```

这种标记风格与 JavaScript、VBScript 的标记风格相同。如果使用的 HTML 编译器不支持其他风格的标记，则可以选择使用这种标记风格。

在实际开发过程中，用户更推荐使用 XML 标记风格，因为使用该风格的 PHP 程序具有更好的可移植性，程序可以在各种服务器环境中正常运行。

3.1.2 PHP 程序的注释

注释是程序不可或缺的重要元素。通过注释不仅能够提高程序的可读性，还有利于开发人员之间的沟通及后期的维护。PHP 程序的注释非常灵活，可以采用 C、C++ 的注释方式，也可以使用 UNIX 系统中 Shell 的注释方式。这些不同的注释方式可以同时出现在一个文件中，被注释的内容都不会被编译器编译。

1. C++ 语言风格和 Shell 脚本风格的单行注释

C++ 和 Shell 脚本这两种注释风格的共同点都是单行注释，区别是 C++ 语言风格的注释方式使用“//”实现，而 Shell 脚本风格的注释方式使用“#”实现。下面看一下使用这两种注释风格的例子。

【例 3-1】 单行注释示例。

```
<html>
```

```
<head>
    <title>单行注释</title>
</head>
<body>
    <?php
        echo "Hello!<br>";          #在页面中输出"Hello!"
        echo "Hello,everyone!";     //在页面中输出"Hello,everyone!"
    ?>
    <?php                           //这是单行注释吗?>这不是单行注释,是网页内容
</body>
</html>
```

上面程序执行后的输出结果如图 3-1 所示。

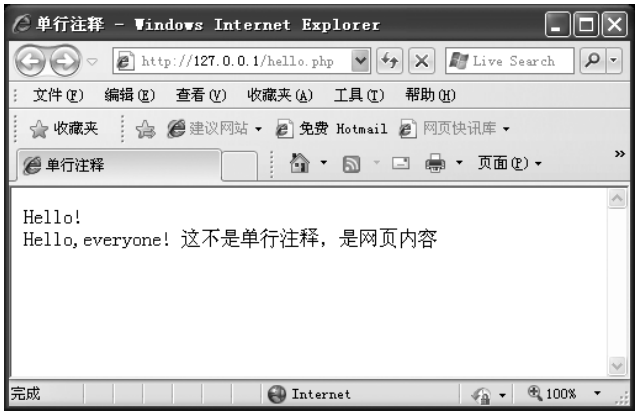

图 3-1　单行注释

从页面执行结果可以看出,注释标记与注释内容必须放在 PHP 开始标志(例如“<?php”)及结束标志(例如“?>”)之间,否则注释功能将不起作用,注释内容会作为网页内容在网页中显示出来。

2. C 语言风格的多行注释

当要添加的注释内容非常多时,为了便于阅读,用户通常会将注释内容分成多行。有时候在调试程序时,也需要将一大段程序代码作为注释内容屏蔽掉。虽然使用“//”或者“#”都能实现注释功能,但是用户需要在每行的开头都加入注释标记,就会很麻烦。这时候就需要用到多行注释,PHP 采用的是 C 语言的多行注释风格,注释内容以“/*”开始,以“*/”结束。下面来看使用这种注释风格的例子。

【例 3-2】 多行注释示例。

```
<html>
    <head>
        <title>多行注释</title>
    </head>
    <body>
        <?php
```

```
            echo "Hello!";
            /*第一行注释
              第二行注释
            */
        ?>
    </body>
</html>
```

上面程序的输出结果如图 3-2 所示。

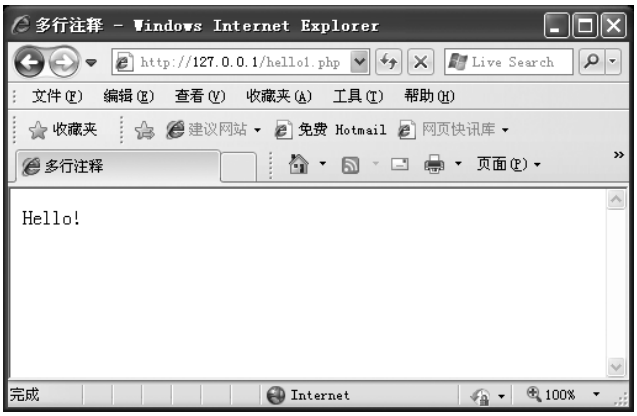

图 3-2　多行注释

当使用这种注释风格时，应该注意不要使注释嵌套出现，否则容易出现问题。

3.1.3　在 HTML 中嵌入 PHP

从上面的例子可以看出，在多数情况下，PHP 代码都是与 HTML 代码混杂在一起的。当包含了 PHP 程序的页面被请求时，Web 服务器会自动编译并处理页面中“<?php”与“?>”之间的代码，将处理结果以 HTML 的形式传送到页面，并在页面中显示处理结果。

可以通过下面的例子来了解上述处理结果。

【例 3-3】 在 HTML 中嵌入 PHP 代码及编译后生成的 HTML。

```
<html>
    <head>
        <title>在 HTML 中嵌入 PHP</title>
    </head>
    <body>
        <?php
            echo "Hello! let's begin!";
        ?>
    </body>
</html>
```

上面例子程序经编译后的源代码如下：

```
<html>
    <head>
```

```
        <title>
            在 HTML 中嵌入 PHP
        </title>
    </head>
    <body>
        Hello! Let's begin!
    </body>
</html>
```

在上面这段代码中，看到的全部都是普通的 HTML 代码，这说明 PHP 代码已经被 Web 服务器编译处理了。

3.2 数据类型

因为 PHP 是作为一种基于 Web 页面的嵌入式脚本语言，它的应用环境决定了数据类型不会很丰富，而且 PHP 不是强类型的语言，这一点也决定了它对程序中的数据类型的控制不会很严格。

3.2.1 PHP 支持的常见数据类型

PHP 对于数据类型的控制其实是非常松散的，不需要类型的声明，不同类型的数据也可以进行运算，它们之间的转化甚至不需要说明。PHP 所支持的主要的数据类型如表 3-2 所示。

表 3-2 PHP 支持的主要数据类型

数据类型	描　述	数据类型	描　述
integer	整数	array	数组
float	浮点型数	class	类
string	字符串		

变量和常量是 PHP 所要处理的基本的数据对象。常量的值和含义在程序运行中都是不会改变的，而变量是为了在程序运行过程中存储临时信息，或者用于保存中间结果而引用的，一旦程序运行结束，其在内存中所占用的临时空间就要被收回，它自然也就没有了。

3.2.2 数据类型转换

绝大多数编程语言（如 C 语言和 Java 语言）在声明变量的时候，必须明确指定变量的数据类型，并且一旦指定了某一变量的数据类型，该变量就只能存储这一种类型的数据。PHP 语言在数据类型的定义方面与其他编程语言有所区别。下面用户通过两个例子来比较一下 PHP 语言与其他语言在定义数据类型时的不同之处。

例如，在 Java 语言中，定义两个变量，如下面的代码所示：

```
String a="hello";          //定义一个 String 类型的变量 a
int b=1;                   //定义一个 int 类型的变量 b
```

而在 PHP 语言中，这两个变量定义的格式如下面的代码所示：

```
$a="ok";                    //定义一个变量 a
$b=2;                       //定义一个变量 b
```

通过比较上面两段代码不难发现，同样是定义两个变量，在 PHP 语言中用户并没有为变量指定任何数据类型。这在很多语言中是不允许的，但是在 PHP 语言中则不会出现任何问题。

例如，在 PHP 中还可以这样使用变量，如下面的代码所示：

```
$a="test";                  //定义一个变量 a,并将字符串"test"赋给 a
$a=3;                       //将数字 3 赋给 a
```

在上面的代码中，当字符串"test"赋值给变量 a 的时候，变量 a 成为一个字符串型的变量；而当数字 3 赋值给变量 a 时，变量 a 成为一个整型的变量。

用户能够像上面两个例子中那样定义和使用变量，主要由 PHP 语言类型转换的特殊性决定。PHP 中的类型转换包括两种方式，即自动类型转换和强制类型转换。下面分别介绍这两种类型转换的实现方式及应用过程。

1. 自动类型转换

自动类型转换是指，在定义变量时不需要指定变量的数据类型，PHP 会根据引用变量的具体应用环境将变量转换为合适的数据类型。在对变量进行赋值操作的时候，经常会用到自动类型转换，主要包括如下两种方式。

(1) 直接对变量进行赋值操作。直接对变量进行赋值操作是自动类型转换最简单的应用，变量的数据类型由赋予的值决定。也就是说，当把一个字符串类型的数据赋给变量时，该变量就是一个字符串类型的变量；当把一个整型数据赋给变量时，该变量就是一个整型的变量。

(2) 运用运算式结果对变量进行赋值操作。自动类型转换的第 2 种应用方式就是将一个运算式的结果赋值给一个变量。这种自动类型转换方式又可分为以下两种情况。

① 运算数为同一数据类型。这种情况处理起来比较简单，由于参与运算的所有运算数都是同一类型，所以被赋值的变量也属于这种类型。例如下面给出的代码：

```
$a=1.223;
$b=3.1415;
$c=$a+$b;
```

变量 a 与变量 b 都是浮点型变量，这两个变量进行相加运算并将运算结果赋值给变量 c，此时，变量 c 就成为了一个浮点型变量。

② 运算数为不同数据类型。如果所有运算数都是数字，则将选取占用字节最长的一种运算数的数据类型作为基准数据类型；如果运算数为字符串，则将该字符串转型为数字然后再进行求值运算。字符串转换为数字的规定为如果字符串以数字开头，则只取数字部分而去除数字后面部分，根据数字部分构成决定转型为整型数据还是浮点型数据；如果字符串以字母开头，则直接将字符串转换为 0。

例如下面给出的代码：

```
$a=1+1.223;
$b=2+"3.141miao";
```

```
$c=3+"world";
```

在第 1 个赋值运算式中，运算数包含了整型数字“1”和浮点型数字“1.223”，根据规定取浮点型数据类型作为基准数据类型。赋值后变量 a 的数据类型为浮点型。

在第 2 个赋值运算式中，运算数包含了整型数字 2 和字符串型数据"3.141miao"，首先将字符串转换为浮点型数据 3.141，然后进行加法运算。赋值后变量 b 的数据类型为浮点型。

在第 3 个赋值运算式中，运算数包含了整型数字 3 和字符串型数据"world"，首先将字符串转换为整型数字 0，然后进行加法运算。赋值后变量 c 的数据类型为整型。

2. 强制类型转换

强制类型转换允许用户手动将变量的数据类型转换成为指定的数据类型。PHP 强制类型转换与 C 语言中的类型转换相似，都是通过在变量前面加上一个“(　)”，并把目标数据类型填写在“(　)”中实现的。

在 PHP 中强制类型转换的具体实现方式如表 3-3 所示。

表 3-3　强制类型转换的实现方式

转换格式	转换结果	实现方式
(int),(integer)	将其他数据类型强制转换为整型	$a="3"; $b=(int)$a; 或 $b=(integer)$a;
(bool),(boolean)	将其他数据类型强制转换为布尔型	$a="3"; $b=(bool)$a; 或 $b=(boolean)$a;
(float),(double),(real)	将其他数据类型强制转换为浮点型	$a="3"; $b=(float)$a; $c=(double)$a; $d=(real)$a;
(string)	将其他数据类型强制转换为字符串	$a=3; $b=(string)$a;
(array)	将其他数据类型强制转换为数组	$a="3"; $b=(array)$a;
(object)	将其他数据类型强制转换为对象	$a="3"; $b=(object)$a;

虽然 PHP 提供了比较宽泛的类型转换机制，为开发者提供了很大便利，但同时也存在着一些问题——比如将字符串型数据转换为整型数据该如何转换、将整型数据转换为布尔型数据该如何转换等。如果没有对上述类似的情形做出明确规定，则在处理类型转换问题时就会出现一些问题。幸运的是，PHP 提供了相关的转换规定。

(1) 其他数据类型转换为整型。其他数据类型转换为整型的规则如表 3-4 所示。

表 3-4　其他数据类型转换为整型

原类型	目标类型	转换规则
浮点型	整型	向下取整，即不会四舍五入而是直接去掉浮点型数据小数点后边的部分，只保留整数部分

续表

原类型	目标类型	转 换 规 则
布尔型	整型	true 转换成整型数字 1，false 转换成整型数字 0
字符串	整型	(1) 字符串为纯整型数字，转换成相应的整型数字 (2) 字符串为带小数点的数字，转换时去除小数点后面的部分，保留整数部分 (3) 字符串以整型数字开头，转换时去除整型数字后面的部分，然后按照规则(1)进行处理 (4) 字符串以带小数点的数字开头，转换时去除小数点后面的部分，然后按规则(2)进行处理 (5) 字符串内容以非数字开头，直接转换为 0

【例 3-4】 其他数据类型转换为整型。

```
<?php
    $a="12";              //定义一个内容为纯数字的字符串型变量 a
    $b="12sunyang";       //定义一个数字开头的字符串型变量 b
    $c="3.141";           //定义一个内容为小数的字符串型变量 c
    $d="3.141a";          //定义一个以小数开头的字符串型变量 d
    $e="jack23";          //定义一个非数字开头的字符串型变量 e
    $f=TRUE;              //定义一个值为 true 的布尔型变量 f
    $g=FALSE;             //定义一个值为 false 的布尔型变量 g
    $h=3.1415;            //定义一个浮点型变量 h
    echo (int)$a."<br>";
    echo (int)$b."<br>";
    echo (int)$c."<br>";
    echo (int)$d."<br>";
    echo (int)$e."<br>";
    echo (int)$f."<br>";
    echo (int)$g."<br>";
    echo (int)$h."<br>";
?>
```

上面的程序执行后的结果如下：

```
12
12
3
3
0
1
0
3
```

浮点型数据向整型数据转换的时候，可能会出现一些不可预料的结果。如下面两行代码：

```
echo (int)0.1*0.7*100;
echo (int)1/3;
```

按照转换规则，得到的结果应该为 7 和 0，但实际运行以后得到的结果却是 6 和 0.33333333333。之所以得到这样的结果，是由于将浮点型数据向整型数据进行转换的时候出现了精度损失。如果需要高精度的运算结果，就不能采取这种强制类型转换的方式。

（2）其他数据类型转换为浮点型。其他数据类型转换为浮点型的规则如表 3-5 所示。

表 3-5　其他数据类型转换为浮点型

原类型	目标类型	转 换 规 则
整型	浮点型	将整型数据直接转换为浮点型，数值保持不变
布尔型	浮点型	true 转换成浮点型数字 1，false 转换成浮点型数字 0
字符串	浮点型	(1) 字符串为整型数字，直接转换成相应的浮点型数字 (2) 字符串以数字开头，转换时去除数字后面的部分，然后按照规则(1)进行处理 (3) 字符串以带小数点的数字开头，转换时直接去除数字后面的部分，只保留数字部分 (4) 字符串以非数字内容开头，直接转换为 0

【例 3-5】 其他数据类型转换为浮点型。

```
<?php
    $a="34";                    //定义一个内容为纯数字的字符串型变量 a
    $b="45miao";                //定义一个以数字开头的字符串型变量 b
    $c="2.75abc";               //定义一个以小数开头的字符串型变量 c
    $d="ok2.76";                //定义一个非数字开头的字符串型变量 d
    $e=123;                     //定义一个整型变量 e
    $f=FALSE;                   //定义一个值为 FALSE 的布尔型变量 f
    $g=TRUE;                    //定义一个值为 TRUE 的布尔型变量 g
    echo (float)$a."<br>";
    echo (float)$b."<br>";
    echo (float)$c."<br>";
    echo (float)$d."<br>";
    echo (float)$e."<br>";
    echo (float)$f."<br>";
    echo (float)$g."<br>";
?>
```

上面的程序的执行结果如下：

```
34
45
2.75
0
```

```
123
0
1
```

(3) 其他数据类型转换为布尔型。其他数据类型转换为布尔型的规则如表 3-6 所示。

表 3-6　其他数据类型转换为布尔型

原　类　型	目标类型	转 换 规 则
整型	布尔型	0 转换为 false,非零的其他整型数字转换为 true
浮点型	布尔型	0.0 转换为 false,非零的其他浮点型数字转换为 true
字符串	布尔型	空字符串或字符串内容为零转换为 false,其他字符串转换为 true
NULL	布尔型	直接转换为 false
数组	布尔型	空数组转换为 false,非空数组转换为 true

【例 3-6】 其他数据类型转换为布尔型。

```
<?php
    $a=0;                    //定义一个值为零的整型变量 a
    $b=45;                   //定义一个非零整型变量 b
    $c=0.0;                  //定义一个值为零的浮点型变量 c
    $d=3.14;                 //定义一个非零浮点型变量 d
    $e="";                   //定义一个空字符串型变量 e
    $f="0";                  //定义一个内容为零的字符串型变量 f
    $g="ok";                 //定义一个非空字符串型变量 g
    $h=NULL;                 //定义一个 NULL 型的变量 h
    $i=array("a","b","c");   //定义一个非空数组 i
    $j=array();              //定义空数组 j
    echo ((boolean)$a)."<br>";
    echo ((boolean)$b)."<br>";
    echo ((boolean)$c)."<br>";
    echo ((boolean)$d)."<br>";
    echo ((boolean)$e)."<br>";
    echo ((boolean)$f)."<br>";
    echo ((boolean)$g)."<br>";
    echo ((boolean)$h)."<br>";
    echo ((boolean)$i)."<br>";
    echo ((boolean)$j)."<br>";
?>
```

上面的程序的执行结果如下：

```
false
true
false
true
```

```
false
false
true
false
true
false
```

(4) 其他数据类型转换为字符串。其他数据类型转换为字符串的规则如表 3-7 所示。

表 3-7　其他数据类型转换为字符串

原 类 型	目 标 类 型	转 换 规 则
整型	字符串	转换时直接在整型数据两边加上“”作为转换后的结果
浮点型	字符串	转换时直接在浮点型数据两边加上“”作为转换后的结果
布尔型	字符串	true 转换为字符串"1",false 转换为字符串"0"
数组	字符串	直接转换为字符串"Array"
对象	字符串	直接转换为字符串"Object"
NULL	字符串	直接转换为空字符串

【例 3-7】 其他数据类型转换为字符串。

```
<?php
    $a=314;                              //定义一个整型变量 a
    $b=3.14;                             //定义一个浮点型变量 b
    $c=TRUE;                             //定义一个值为 true 的布尔型变量 c
    $d=FALSE;                            //定义一个值为 false 的布尔型变量 d
    $e=array("good","luck","hello");     //定义一个数组 e
    $f=NULL;                             //定义一个 NULL 型变量 f
    echo (string)$a."<br>";
    echo (string)$b."<br>";
    echo (string)$c."<br>";
    echo (string)$d."这里显示为空字符串<br>";
    echo (string)$e."<br>";
    echo (string)$f."这里显示为空字符串<br>";
?>
```

上面的程序执行后的结果如下：

```
314
3.14
1
这里显示为空字符串
Array
这里显示为空字符串
```

(5) 其他数据类型转换为数组。其他数据类型转换为数组的规则如表 3-8 所示。

表 3-8　其他数据类型转换为数组

原类型	目标类型	转 换 规 则
整型 浮点型 布尔型 字符串	数组	将这几个数据类型强制转换为数组时，得到的数组只包含一个数据元素，该数据就是未转换前的数据，并且该数据的数据类型也与未转换前相同
对象	数组	转换时将对象的成员变量的名称作为各数组元素的 key，而转换后数组每个 key 的 value 都为空 (1) 如果成员变量为私有的(private)，则转换后 key 的名称为“类名＋成员变量名” (2) 如果成员变量为公有的(public)，则转换后 key 的名称为“成员变量名” (3) 如果成员变量为受保护的(protected)，则转换后 key 的名称为“＊＋成员变量名”
NULL	数组	直接转换为一个空数组

【例 3-8】 其他数据类型转换为数组。

```
<?php
    $a=34;                                          //定义一个整型变量 a
    $b=2.718;                                       //定义一个浮点型变量 b
    $c=FALSE;                                       //定义一个布尔型变量 c
    $d="good";                                      //定义一个字符串型变量 d
    //定义一个类 A,包含 3 个不同属性的成员变量
    class A
    {
        private $pri;
        protected $pro;
        public $pub;
    }
    $e=new A();                                     //实例化一个 A 的对象 e
    $f=NULL;                                        //定义一个 NULL 类型的变量 f
    echo var_dump((array)$a)."<br>";
    echo var_dump((array)$b)."<br>";
    echo var_dump((array)$c)."<br>";
    echo var_dump((array)$d)."<br>";
    echo var_dump((array)$e)."<br>";
    echo var_dump((array)$f);
?>
```

上面的程序执行后的结果如下：

```
array(1) { [0]=>int(34) }
array(1) { [0]=>float(2.718) }
array(1) { [0]=>bool(FALSE) }
array(1) { [0]=>string(4) "good" }
array(1) { [Apri]=>NULL [Apro]=>NULL [Apub]=>NULL }
array(0) { }
```

(6) 其他数据类型转换为对象。其他数据类型转换为对象的规则如表 3-9 所示。

表 3-9　其他数据类型转换为对象

原类型	目标类型	转 换 规 则
整型 浮点型 布尔型 字符串	对象	将其他类型变量转换为对象时,将会新建一个名为 scalar 的属性,并将原变量的值存储在这个属性中
数组	对象	将数组转换为对象时,数组的 key 作为对象成员变量的名称,对应各个 key 的 value 作为对象成员变量保存的值
NULL	对象	直接转换为一个空对象

【例 3-9】 其他数据类型转换为对象。

```
<?php
    $a=(object)10;                                  //将整型数据转型为对象并赋给变量 a
    $b=(object)3.141;                               //将浮点型数据转型为对象并赋给变量 b
    $c=(object)TRUE;                                //将布尔型数据转型为对象并赋给变量 c
    $d=(object)NULL;                                //将 NULL 转型为对象并赋给变量 d
    $e=(object)array("a"=>13,"b"=>14,"c"=>15);
    $f=(object)"test";                              //将字符串转型为对象并赋给变量 f
    echo $a->scalar."<br>";
    echo $b->scalar."<br>";
    echo $c->scalar."<br>";
    echo $d->scalar."<br>";
    echo $e->c."<br>";
    echo $f->scalar."<br>";
?>
```

上面代码运行的结果如下:

```
10
3.141
true

15
test
```

3.3　常量与变量

常量和变量是 PHP 要处理的基本的数据对象。常量中最典型的一个例子就是圆周率,常量的值在程序运行前后是不会改变的。而变量是为了在程序运行过程中暂时保存一些中间结果的,它的值在程序运行期间是可以改变的。

3.3.1 常量

本节将要介绍 PHP 语言中的一个重要的概念——常量。常量就是从程序开始运行到结束都不变的量。在 PHP 中，常量是一种词法符号，用来表示固定的数值或字符，一旦定义初值，数值就不能再改变。在传统的程序中，常量是不经过任何说明就可以使用的，但是在脚本语言中，常量就有了一些变化。在 PHP 中主要有两种常量：普通常量和定义常量。

1. 普通常量

普通常量就是那些在程序中不变的值，它们不是符号，而是实在的数值或字符，如字符串"china"。普通常量可根据它们的值的类型的不同分成以下几类：数值常量、字符常量、字符串常量。

(1) 数值常量。数值常量主要分两种：整型和实型的常量。

整型的常数可以是十进制的、八进制的、十六进制的。十进制的常量就是由 0～9 这 10 个数码组成的整数，除了 0 以外，这个数的第一个数码不能是 0。八进制的常量就是由 0～7 这 8 个数码组成的整数，不能有超过这个范围以外的数码。八进制的数是以 0 开头的。十六进制的常量是由 0～9 这 10 个数码加上 A～F 这 6 个字母组成的，它的开头必须是 0x。整型常量的前面可以加负号或者正号来表示它们的值的正负，和用户通常书写的习惯一样，正号一般都省略。下面是一些数值常量的例子：

```
12345  0123  3456  0x678  0x981EA  0xedac
```

实型常量只能用十进制表示，它有两种表示形式，一种是普通的小数点形式，另一种是指数形式。例如，1.23E－7 就是一个标准的指数形式，其中的 1.23 是尾数，－7 为指数。

PHP 处理实型常量的方法很奇特，尤其是在数值计算当中的默认的类型转换当中，这在以后会详细地说明。下面是一些实型常量的例子，有普通的小数形式，也有指数形式：

```
1234.456  389.245  156.379e15  1.234e-10
```

(2) 字符常量。PHP 中的字符常量和 C 语言中的字符常量完全相同，是一个用“”括起来的字符。如'a'、'b'、'c'、'd'、'D'、'?' 等都是字符常量。注意，'d'和'D'是不同的字符常量。其中“”的作用就是告诉解释器，“” ””之间的是一个字符常量。它本身并不是这个字符常量的内容。

另外，PHP 也支持 C 语言中的转义序列。所谓的转义序列，就是用转义字符“\”后面跟一个字符或者是整型常量来表示一个单一的字符。若转义字符后面是一个整型常量，则这个常量必须是一个八进制或者十六进制的整数，例如'\0x123'、'\0176'等。但是这些数字的值不能大于 256。

在实际应用当中，转义字符常用来表示那些不能从键盘上输入或者是在屏幕上显示的字符。为了便于记忆，用户将 PHP 中常用的转义字符序列写成一张表，如表 3-10 所示。

注意：把“'”和“"”作为转义字符时，因为它们在 PHP 中有特殊的含义，例如“'”表示字符常量，“"”表示字符串常量。不过“'”也可以表示字符串常量。这将在字符串常量中介绍。

(3) 字符串常量。字符串常量就是由“" ””括起来的字符序列。它也是最常用的常量类型。它主要分为两种：单引号字符串和双引号字符串。

表 3-10　PHP 常用转义字符

表　示	功　能	表　示	功　能
\'	单引号	\\	反斜杠
\"	双引号	\r	回车
\t	制表符	\012	任意一个八进制数
\n	换行符	\0x34	任意一个十六进制数
\$	美元符号		

单引号字符串就是用单引号括起来的一串字符。例如：

```
'I am a beautiful girl.'
'I want to drink tea.'
```

在单引号字符串中，只能对“'”和“\”进行转义，而其他的转义字符都会被原样输出，也就是说，单引号字符串中可以出现除了“'”和“\”之外的所有其他字符。例如，字符串

```
'"world\t\n\r\023\0x78\\\'a\''
```

的输出是

```
"world\t\n\r\023\0x78\'a'
```

其中，只有“\\”和“\'”被转义输出。单引号字符串有一个重要的特点是如果需要在字符串中换行，不需要用转义字符，只需简单地在源码中键入换行符即可。

【例 3-10】　测试换行符。

```
<HTML>
    <HEAD>
        <TITLE>测试换行符</TITLE>
    </HEAD>
    <BODY>
        <?php
        echo "<pre>first step:
        open the door of the fridge.
        second step:
        put the elephant into the fridge.
        last stap:
        close the door of the fridge. ";
        ?>
    </BODY>
</HTML>
```

浏览器的显示结果如图 3-3 所示。

双引号字符串跟单引号字符串类似，不过它是用“"　"”括起来的一串字符。例如：

```
"I am a beautiful girl. "
```

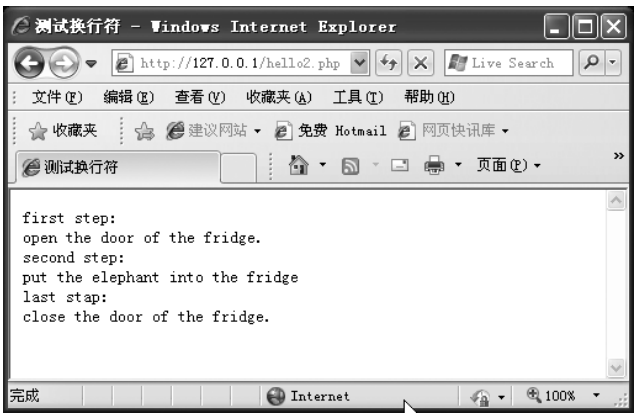

图 3-3　演示换行符的输出

```
"It is a sunny day today. "
```

双引号字符串的特点就是可以在字符串中加入转义字符序列和进行变量替换，需要注意的是“'”不必经过转义就可以直接输出。

2. 定义常量

定义常量是指那些用函数定义的常量。这种常量通常都是某些字符串或者是表达式的值的别名。也就是说，是为了方便才定义它们的。定义常量分为两种：内部常量和自定义常量。

(1) 内部常量。PHP 内部定义了很多常量，它们都是有特殊含义的。以下一些常量便是在 PHP 中定义的内部常量。

① PHP_VERSION。这个内部常量是 PHP 程序的版本，如 4.0.1-dev。

② PHP_OS。这个内部常量指的是执行 PHP 解释器的操作系统的名称，如 Windows。

③ _FILE_。这个内部常量是 PHP 程序文件名。若包含文件(include 或 require)，则在包含文件内的该常量为包含文件名，而不是包含它的文件名。

④ _LINE_。这个内部常量是 PHP 程序的行数。若包含文件(includes 或 require)，则在包含文件内的该常量为包含文件的行，而不是包含它的文件行。

⑤ E_ERROR。这个常量指最近的出错的地方。

⑥ E_NOTICE。这个常量指发生不寻常但又不一定是错误的地方。例如存取一个不存在的变量。

⑦ E_PARSE。这个常量是解释语法有潜在问题处。

⑧ E_WARNING。这个常量指上一个警告处。

⑨ TRUE。这个常量就是逻辑真值 true。

⑩ FALSE。这个常量就是指逻辑假值 false。

在上面这些内部常量中，以 E_开头的形式的常量可以参考 error_reporting()函数，有更多的相关说明。

(2) 自定义常量。由于 PHP 语言和所有其他脚本语言一样，所有的变量不经过说明就可以赋值，赋值时赋给变量的值就是一个传统意义上的常量。PHP 中对常量有专门的定义

和使用的方法，此时 PHP 定义的常量与 C 语言中使用命令 # define 定义的常量是完全类似的。也就是说，PHP 中的常量特指使用这种方式定义的具有特定含义的符号。

PHP 内部定义了很多常量，它们都是有特殊含义的，上面已经介绍了。当然，编写程序时，仅仅使用上面那些内部常量是不够的。对于这一点，PHP 提供了 define()的功能，它使程序员可以自己定义所需要的常量。

【例 3-11】 自己定义常量。

```
<?php
    define("theme","our life is so beautiful.");
    echo theme;
?>
```

从上面的例子可以看出，上面的代码是给一个字符串"our life is so beautiful"起了一个名字，称为 theme，它可以像变量一样使用。这可以防止用户一遍又一遍地写一个很长的字符串，只需用一个符号代替即可，既简便又防止了写长字符串出错。实际上，这完全与 C 语言中的 # define 的功能一样。

3.3.2 变量

变量用于存储值，例如数字、文本字符串或数组。一旦设置了某个变量，用户就可以在脚本中重复地使用它。命名一个变量的标识符号就是该变量的变量名。所谓的变量的声明，是指程序通知解释程序为这个变量预留一定的存储单元。由于 PHP 是脚本语言，所以它和其他脚本语言一样，都可以直接使用变量，而不用声明变量。也就是说，当需要的时候，PHP 解释器可以动态地为变量分配内存。

1. 变量类型

PHP 的变量类型常用的有 5 种：string、integer、double、array、object。

(1) string 即为字符串变量，无论是单一字符还是数千字的字符串，都是使用这个变量类型。值得注意的是，要指定字符串给字符串变量，就要在字符串的头尾加上“""(例如："这是字串")。在欲让字串换行时，可使用溢出字元，也就是“\”加上指定的符号，若是“\x”加上两位数字，如“\xFE”即表示十六进位字符，如表 3-11 所示。

表 3-11 溢出字元表示法

符　　号	意　　义	符　　号	意　　义
\\	反斜线	\n	换行
\"	双引号	\t	跳位 TAB
\r	送出 CR		

初始化字符串变量，用户只要简单的给它赋值即可。例如：

```
$mystring="我的字符串";
$name="me";
```

(2) integer 为整数变量。在 32 位的操作系统中，它的有效范围是－2147483648～

2147483647。要使用十六进制整数,可以在前面加 0x。

初始化整型变量的示例如下:

```
$grade=90;
$age=21;
$hexint=0x24;
```

(3) double 为浮点数变量。在 32 位的操作系统中,有效范围是 1.7×10^{-308} ~ 1.7×10^{308}。

初始化浮点数变量的示例如下:

```
$float1=2.712;
$float2=3.14E+2;
```

(4) array 为数组变量,可以是二维、三维或者多维数组,其中的元素也很自由,可以是 string、integer 或者 double,甚至是 array。

数组变量可以使用这两种方法来赋值:使用一系列连续数值,或使用 array()函数构造(见 Array functions 部分)。要将连续的数值加进数组,只需将要赋的值赋给不带下标的数组变量。该值会作为数组的最后元素加进数组中。例如:

```
$names[]="zhang";                    //$names[0]="wang"
$names[]="li";                       //$names[1]="zhao"
```

与 C、Perl 相似, PHP 数组的下标也是从 0 开始的。

数组的使用示例如下:

```
$array1=array("子", "丑", "寅", "卯");
$array2=array(
"天干"=>array("甲","乙","丙","丁"),
"地支"=>array("子", "丑", "寅", "卯"),
"数字"=>array(1, 2, 3, 4));
```

(5) object 为对象变量,目前在 PHP 中的对象不多,若论及对象,虽然微软公司的 ASP 对象仍然比 PHP 的内部对象多,但是,Web CGI 程式要求的是效率,以完全类导向的方式,恐怕使用者在浏览时也会因为程式执行速度过慢而很不耐烦。

要初始化一个对象,就需要用 new 语句建立该类型的变量。

```
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}
$bar=new foo;
$bar->do_foo();
```

还有布尔值(boolean),通常 1 即为 true,0 为 false。

PHP 程序可以在变量之间自由地转换形态，不必经过特殊的转换，但是在浮点数转成整数时就有点牵强了，不过可以先将浮点数转成字串，然后再处理。

2. 变量的命名规则

变量的命名规则如下。

(1) 变量名必须以字母或者“_”开头。

(2) 变量名只能包含字母、数字字符以及“_”。

(3) 变量名不能包含空格。如果变量名由多个单词组成，那么应该使用“_”进行分隔(如 $my_string)，或者以大写字母开头(如 $myString)。

要使用变量，只要在英文字串前面加“$”即可，目前变量名称仍不能使用中文。至于变量的大小写是不一样的，而且在设置变量名的时候一定要含义明确，而不要用那些含义不明的变量名，例如 $a、$b、$c 等。在开发 PHP 程序时，最好使用一定的变量名风格(指变量名中的单词的拼写、大小写、间隔等所使用的统一的规范)，以免因为变量大小写的问题，花许多无谓的时间去找寻问题点，这样会很麻烦。

下面介绍两种变量名风格。

(1) UNIX 的变量名风格。间隔使用“-”，变量名一律小写，单词用全名或者是公认的简写法，例如：

```
$studentname="xiaoming";
$code=3;
```

(2) Windows 变量名风格。变量名中的每个单词首字母大写，其余全部小写；单词用全名或者是公认的简写法，例如：

```
$StudentName="zhangsan";
$Code=153;
```

以下是一些不好的变量名风格：

```
$int1=20;
$string1="beautiful";
$float1=3.14;
```

在这些例子中，除了程序的编写者，其他人很难知道 $int1 代表什么，但是如果要问 $studentname 是什么意思，可以很容易看出来，它代表的意思是学生姓名。这就是使用有意义的变量名的好处。

对于变量，还有两个比较重要的特性，变量的作用域和变量的生命周期，这需要涉及函数，放到后面讲完函数后再讲。

3. 变量的声明和初始化

PHP 中可以直接使用变量，而不用提前声明一个变量及其类型。PHP 中的所有变量都以“$”开始。

在 PHP 中设置变量的正确方法如下：

```
$varname=value;
```

PHP 是一门松散类型的计算机语言。在 PHP 中，不需要在设置变量之前声明该变量，也不需要提前定义其类型。它们的类型只能在初始化的时候由系统自动确定，当初始化的值为整型时，其类型即为整型；当初始化值为字符串时，即为 string 类型。例如：

```
$name;
$class;
$code;
```

这些均是 PHP 中的变量的形式，但是它们的类型都不能确定。当赋值后类型即确定了。例如：

```
$name="yuanyuan";
$class="yiban";
$code=153;
```

以上的变量 name 和 class 即为 string 型，code 即为 int 型。

需要注意的是，声明的变量和初始化时（即赋值）的类型应该一致，如果不一致，编译时也不会出错。因为在 PHP 中不进行类型检查，而是直接对变量做强制的类型转换，得出的结果可能和预期的结果有很大的不同。所有，用户一定要记住这一点。

PHP 的入门者往往会忘记在变量的前面的“$”。如果那样做，变量将是无效的。

下面试着创建一个存有字符串的变量和一个存有数值的变量，如例 3-12 所示。

【例 3-12】 创建字符串变量及数值变量。

```
<?php
    $txt="good morning!";
    $number=10;
?>
```

在上面的例子可以看到，不必向 PHP 声明该变量的数据类型。根据变量被设置的方式，PHP 会自动地把变量转换为正确的数据类型。而在强类型的编程语言中，必须在使用前声明变量的类型和名称。

在 PHP 中，变量会在使用时被自动声明。

3.4 运算符与表达式

运算符指对操作数所进行的运算。操作数可以是前面所讲的常量或变量或下面将要讲到的表达式。与 C 语言一样，PHP 中的运算符也可以分为单目运算符、双目运算符和三目运算符。这里的“目”指的就是操作数。几目就是指该运算符可以对几个操作数同时进行操作。按运算符的功能分，基本的运算符类型可以分为算术运算符、赋值运算符、关系运算符、逻辑运算符、位运算符、字符串运算符以及其他运算符。

PHP 中的表达式是通过递归的方式定义的，定义如下。

（1）一个操作数就是一个表达式。

（2）由运算符连接在一起的两个操作数构成一个表达式。

（3）由运算符连接在一起的两个表达式构成一个表达式。