第 3 章
PHP的基本语法


学习目标 Objective

上一章讲述了 PHP 的环境搭建方法，本章将开始学习 PHP 的基本语法，主要包括 PHP 的标识符、编码规范、常量、变量、数据类型、运算符和表达式等。通过本章的学习，读者可以掌握 PHP 的基本语法知识和相关技能。


内容导航 Navigation

- 了解 PHP 标识符
- 熟悉 PHP 的编码规范
- 掌握常量的使用方法
- 掌握变量的使用方法
- 掌握数据类型
- 掌握运算符的使用方法
- 掌握表达式的使用方法
- 掌握创建多维数组的方法

3.1 PHP 7 的标记风格

目前，PHP 7 是以<?php ?>标识符为开始和结束标记的。也有人把这种默认风格称为 PHP 的 XML 风格。PHP 7 只支持这种标记风格，例如：

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

早期版本中还支持短风格、脚本风格和 ASP 风格。

1. 短风格

有时候，读者会看到一些代码中出现用<? ?>标识符表示 PHP 代码的情况。这就是所谓的“短风格”（short style）表示法，例如：

```
<? echo "这是 PHP 短风格的表示方式。"?>
```

这种表示方法在正常情况下并不推荐，并且在 php.ini 文件中 short_open_tags 的默认设置是关闭的。另外，以后提到的一些功能设置会与这种表示方法相冲突，比如与 XML 的默认标识符相冲突。

2. 脚本风格

有的编辑器由于跟以前程序的定义表示要区分开，对 PHP 代码完全采用另一种表示方式，即<script></script>的表示方式，例如：

```
<script language="php">
    echo "这是 PHP 的 script 表示方式。";
</script>
```

这十分类似 HTML 页面中 JavaScript 的表示方式。

3. ASP 风格

由于 ASP 的影响，为了照顾 ASP 使用者对 PHP 的使用，PHP 还提供了 ASP 的表示风格，例如：

```
<%
    echo "这是 PHP 的 ASP 的表示方式。";
%>
```

需要特别注意的是，上述三种风格只能在 PHP 5 或者更低的版本中使用，PHP 7 已经不再支持上述 3 种风格。

3.2 编码规范

由于现在的 Web 开发往往是多人一起合作完成的，因此使用相同的编码规范显得非常重要，特别是新的开发人员参与时，通常需要知道前面开发代码中变量或函数的作用等，这就需要统一的编码规范。

3.2.1 什么是编码规范

编码规范是一套某种编程语言的导引手册，这种导引手册规定了一系列语言的默认编程风格，以增强语言的可读性、规范性和可维护性。一个语言的编码规范主要包括文件组织、缩进、注释、声明、空格处理、命名规则等。

遵守编码规范有以下好处。

- 编码规范是团队开发中对每个成员的基本要求。编码规范的好坏是一个程序员成熟程度的表现。
- 提高程序的可读性，有利于开发人员互相交流。
- 良好一致的编程风格，在团队开发中可以达到事半功倍的效果。

- 有助于程序的维护，降低软件成本。

3.2.2 PHP 中的编码规范

PHP 作为一种高级语言，十分强调编码规范。

1. 表述

在 PHP 的正常表述中，每一句 PHP 语句都是以“;”结尾的，这个规范就告诉 PHP 要执行此语句，例如：

```
<?php
   echo "php 以分号表示语句的结束和执行。";
?>
```

2. 指令分隔符

在 PHP 代码中，每个语句后需要用分号结束命令。一段 PHP 代码中的结束标记隐含表示了一个分号，所以在 PHP 代码段的最后一行可以不用分号结束，例如：

```
<?php
   echo "这是第一个语句";       // 每个语句都加入分号
   echo "这是第二个语句";
   echo "这是最后一个语句"?>   // 结束标记“?>”隐含了分号，这里可以省略分号
```

3. 空白符

PHP 对空格、回车造成的新行、Tab 等留下的空白的处理也遵循编码规范。PHP 对它们都进行了忽略。这跟浏览器对 HTML 语言中的空白的处理是一样的。

合理利用空白符可以增强代码的可读性和清晰性。

⑴下列情况应该总是使用两个空白行：

① 两个类的声明之间。
② 一个源文件的两个代码片段之间。

⑵ 下列情况应该总是使用一个空白行：

① 两个函数声明之间。
② 函数内的局部变量和函数的第一个语句之间。
③ 块注释或单行注释之前。
④ 一个函数内的两个逻辑代码段之间。

⑶ 合理利用空格缩进可以提高代码的可读性。

① 空格通常用于关键字与括号之间，但是函数名称与左括号之间不能使用空格分开。
② 函数参数列表中的逗号后面通常会插入空格。
③ for 语句的表达式应该用逗号分开，后面添加空格。

4. 注释

为了增强可读性，在很多情况下，程序员都需要在程序语句的后面添加文字说明。而 PHP 要把它们与程序语句区分开，就需要让这些文字注释符合编码规范。

这些注释的风格包括 C 语言风格、C++风格和 SHELL 风格。

（1）C 语言风格如下：

```
/*这是 C 语言风格的注释内容*/
```

这种方法还可以多行使用：

```
/*这是
  C 语言风格
  的注释内容
*/
```

（2）C++风格如下：

```
//这是 C++风格的注释内容行一
//这是 C++风格的注释内容行二
```

（3）SHELL 风格如下：

```
#这是 SHELL 风格的注释内容
```

C++风格和 SHELL 风格只能一句注释占用一行，既可单独一行，也可使用在 PHP 语句之后的同一行。

5. 与 HTML 语言混合搭配

凡是在一对 PHP 开始和结束标记之外的内容都会被 PHP 解析器忽略，这使得 PHP 文件可以具备混合内容，使 PHP 嵌入 HTML 文档中，例如：

```
<HTML>
<HEAD>
    <TITLE>PHP 与 HTML 混合</TITLE>
</HEAD>
<BODY>
<?php
     echo "嵌入的 PHP 代码";
?>
</BODY>
<HTML>
```

3.3 常　量

在 PHP 中，常量是一旦声明就无法改变的值。本节将讲述如何声明和使用常量。

3.3.1 声明和使用常量

PHP 通过 define()命令来声明常量，格式如下：

```
define（"常量名"，常量值）；
```

常量名是一个字符串，通常在 PHP 的编码规范指导下使用大写英文字母表示，比如 CLASS_NAME 、MYAGE 等。

常量值可以是很多种 PHP 的数据类型，可以是数组、对象，当然也可以是字符和数字。

常量就像变量一样存储数值，但是与变量不同的是，常量的值只能设定一次，并且无论在代码的任何位置，都不能被改动。常量声明后具有全局性，在函数内外都可以访问。

【例 3.1】（实例文件：源文件\ch03\3.1.php）

```
<?php
   define("HUANY","欢迎学习 PHP 基本语法知识");  // 定义常量 HUANY
   echo HUANY;  // 输出常量值
?>
```

本程序运行结果如图 3-1 所示。

图 3-1 运行结果

【案例分析】

- 用 define 函数声明一个常量。常量的全局性体现为可在函数内外进行访问。
- 常量只能存储布尔值、整型数据、浮点型数据和字符串数据。

3.3.2 内置常量

PHP 的内置常量是指 PHP 在系统建立之初就定义好的一些常量。PHP 中预定义了很多系统内置常量，这些常量可以被随时调用。下面列出一些常见的内置常量。

（1）_FILE_：这个默认常量是文件的完整路径和文件名。若引用文件（include 或 require），则在引用文件内的该常量为引用文件名，而不是引用它的文件名。

（2）_LINE_：这个默认常量是 PHP 程序行数。若引用文件（include 或 require），则在引用文件内的该常量为引用文件的行，而不是引用它的文件行。

（3）PHP_VERSION：这个内置常量是 PHP 程序的版本，如 3.0.8-dev。

（4）PHP_OS：这个内置常量是指执行 PHP 解析器的操作系统名称，如 Linux。

（5）TRUE：这个常量是真值（true）。

（6）FALSE：这个常量是伪值（false）。

（7）E_ERROR：这个常量指到最近的错误处。

（8）E_WARNING：这个常量指到最近的警告处。

（9）E_PARSE：这个常量指到解析语法有潜在问题处。

（10）E_NOTICE：这个常量为发生不寻常，但不一定是错误处，例如存取一个不存在的变量。

（11）_DIR_：这个常量为文件所在的目录。该常量是在 PHP 5.3.0 版本中新增的。

（12）_FUNCTION_：这个常量为函数的名称。从 PHP 5 开始，此常量返回该函数被定义时的名字，并且区分大小写。

（13）_CLASS_：这个常量为类的名称。从 PHP 5 开始，此常量返回该类被定义时的名字，并且区分大小写。

下面举例说明系统常量的使用方法。

【例 3.2】（实例文件：源文件\ch03\3.2.php）

```
<?php
echo(__FILE__);         // 输出文件的路径和文件名
echo "<br>";          // 输出换行
echo(__LINE__);        // 输出语句所在的行数
echo "<br>";
echo(PHP_VERSION); // 输出 PHP 的版本
echo "<br>";
echo(PHP_OS);      // 输出操作系统名称
?>
```

本程序的运行结果如图 3-2 所示。

图 3-2　程序运行结果

【案例分析】

（1）echo "
"语句表示输出换行。

（2）echo(_FILE_)语句输出文件的文件名，包括详细的文件路径。echo(_LINE_)语句输出该语句所在的行数。echo(PHP_VERSION)语句输出 PHP 程序的版本。echo(PHP_OS)语句输出执行 PHP 解析器的操作系统名称。

3.4 变 量

变量像一个贴有名字标签的空盒子。不同的变量类型对应不同种类的数据，就像不同种类的东西要放入不同种类的盒子。

3.4.1 PHP 中的变量声明

PHP 中的变量不同于 C 或 Java 语言，因为它是弱类型的。在 C 或 Java 中，需要对每一个变量声明类型，但是在 PHP 中不需要这样做。

PHP 中的变量一般以“$”作为前缀，然后以字母 a~z 的大小写或者“_”下画线开头。这是变量的一般表示。

合法的变量名可以是：

```
$hello
$Aform1
$_formhandler (类似我们见过的$_POST 等)。
```

非法的变量名如：

```
$168
$!like
```

PHP 中不需要显式地声明变量，但是定义变量前进行声明并带有注释是一个好的程序员应该养成的习惯。PHP 的赋值有两种，即传值和引用，区别如下：

⑴ 传值赋值：使用“=”直接将赋值表达式的值赋给另一个变量。

⑵ 引用赋值：将赋值表达式内存空间的引用赋给另一个变量。需要在“=”左右的变量前面加上一个“&”符号。在使用引用赋值的时候，两个变量将会指向内存中同一个存储空间，所以任意一个变量的变化都会引起另一个变量的变化。

【例 3.3】（实例文件：源文件\ch03\3.3.php）

```
<?php
echo "使用传值方式赋值：<br/>";      // 输出 使用传值方式赋值
$a = "风吹草低见牛羊";
$b = $a;                  // 将变量$a 的值赋值给$b，两个变量指向不同的内存空间
echo "变量 a 的值为".$a."<br/>";      // 输出 变量 a 的值
echo "变量 b 的值为".$b."<br/>";      // 输出 变量 b 的值
$a = "天似穹庐，笼盖四野";     // 改变变量 a 的值，变量 b 的值不受影响
echo "变量 a 的值为".$a."<br/>";      // 输出 变量 a 的值
echo "变量 b 的值为".$b."<p>";        //输出 变量 b 的值
echo "使用引用方式赋值：<br/>";     //输出  使用引用方式赋值
$a = "天苍苍，野茫茫";
$b = &$a;                 // 将变量$a 的引用赋给$b，两个变量指向同一块内存空间
echo "变量 a 的值为".$a."<br/>";     // 输出 变量 a 的值
echo "变量 b 的值为".$b."<br/>";     // 输出 变量 b 的值
$a = "敕勒川，阴山下";
```

```
/*
改变变量 a 在内存空间中存储的内容，变量 b 也指向该空间，b 的值也发生变化
*/
echo "变量 a 的值为".$a."<br/>";  // 输出 变量 a 的值
echo "变量 b 的值为".$b."<p>";    // 输出 变量 b 的值
?>
```

本程序运行结果如图 3-3 所示。

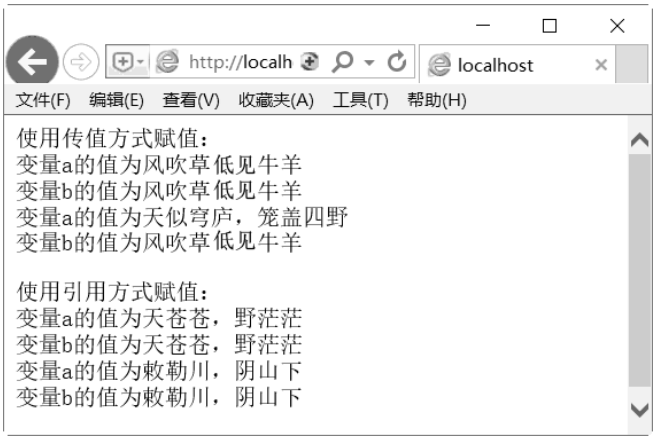

图 3-3　程序运行结果

3.4.2　可变变量与变量的引用

一般的变量很容易理解，但是有两个概念比较容易混淆，就是可变变量和变量的引用。

可变变量是一种特殊的变量，它允许动态改变一个变量名称。其工作原理是该变量的名称由另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“$”。

在一个变量前加上“&”，然后赋值给另一个变量，这就是变量的引用赋值。

通过下面的例子对它们进行说明。

【例 3.4】（实例文件：源文件\ch03\3.4.php）

```
<?php
  $aa = "bb";                              // 定义变量$aa 并赋值
  $bb = "征蓬出汉塞，归雁入胡天。";          //定义变量$bb 并赋值
echo $aa;              // 输出变量$aa
echo "<br>";
echo $$aa;                                 //通过可变变量输出变量$bb 的值
$bb = "大漠孤烟直，长河落日圆。";          //重新给变量$bb 赋值
echo "<br>";
echo $$aa;
echo "<br>";

$a = 100;
$b = 200;
echo $a;
```

```
    echo "<br>";
    echo $b;
    echo "<br>";
    $b = &$a;   //变量的引用
    echo $a;
    echo "<br>";
    echo $b;
    $b = 300;
    echo "<br>";
    echo $a;
    echo "<br>";
    echo $b;
?>
```

本程序运行结果如图 3-4 所示。

图 3-4　程序运行结果

【案例分析】

（1）在代码的第一部分，$aa 被赋值 bb。若$aa 相当 bb，则$$aa 相当于$bb。所以当$$aa 被赋值为“征蓬出汉塞，归雁入胡天。”时，打印$bb 就得到“征蓬出汉塞，归雁入胡天。”。反之，当$bb 变量被赋值为“大漠孤烟直，长河落日圆。”时，打印$$aa 同样得到“大漠孤烟直，长河落日圆。”。这就是可变变量。

（2）在代码的第二部分里，$a 被赋值 100，然后通过“&”引用变量$a 并赋值给$b。这一步的实质是，给变量$a 添加了一个别名$b。所以打印时，$a 和$b 都得出原始赋值 100。由于$b 是别名，和$a 指的是同一个变量，因此当$b 被赋值 300 后，$a 和$b 都得到新值 300。

（3）可变变量其实是允许改变一个变量的变量名，允许使用一个变量的值作为另一个变量的名。

（4）变量引用相当于给变量添加了一个别名，使用“&”来引用变量。其实两个变量名指的是同一个变量。就像是给同一个盒子贴了两个名字标签，两个名字标签指的是同一个盒子。

3.4.3　变量作用域

所谓变量作用域（scope），是指特定变量在代码中可以被访问到的位置。在 PHP 中有 6 种基本的变量作用域法则。

（1）内置超全局变量（built-in superglobal variables），在代码中的任意位置都可以访问到。

（2）常数（constants），一旦声明，就是全局性的，可以在函数内外使用。

（3）全局变量（global variables），在代码间声明，可在代码间访问，但是不能在函数内访问。

（4）在函数中声明为全局变量的变量就是同名的全局变量。

（5）在函数中创建和声明为静态变量的变量在函数外是无法访问的，但是这个静态变量的值可以保留。

（6）在函数中创建和声明的局部变量在函数外是无法访问的，并且在本函数终止时失效。

1. 超全局变量

superglobal 或 autoglobal 可以称为“超全局变量”或“自动全局变量”。这种变量的特性是，无论在程序的任何地方都可以访问到，无论是函数内还是函数外都可以访问到。而这些“超全局变量”就是由 PHP 预先定义好以方便使用的。

那么这些“超全局变量”或“自动全局变量”都有哪些呢？

- $GLOBALS：包含全局变量的数组。
- $_GET：包含所有通过 GET 方法传递给代码的变量的数组。
- $_POST：包含所有通过 POST 方法传递给代码的变量的数组。
- $_FILES：包含文件上传变量的数组。
- $_COOKIE：包含 cookie 变量的数组。
- $_SERVER：包含服务器环境变量的数组。
- $_ENV：包含环境变量的数组。
- $_REQUEST：包含用户所有输入内容的数组（包括$_GET、$_POST 和$_COOKIE）。
- $_SESSION：包含会话变量的数组。

2. 全局变量

全局变量其实就是在函数外声明的变量，在代码间都可以访问，但是在函数内是不能访问的。这是因为函数默认不能访问在其外部的全局变量。

通过下面的实例介绍全局变量的使用方法和技巧。

【例 3.5】（实例文件：源文件\ch03\3.5.php）

```
<?php
$a=100; // 全局变量
// 访问全局变量
function ff()
{
    $b=100; // 局部变量
    echo "变量 a 为: $a"; // 访问全局变量
    echo "<br>";
    echo "变量 b 为: $b"; // 访问局部变量
}

ff();                    // 函数内访问全局变量和局部变量
echo "<br>";
echo "变量 a 为: $a"; //函数外访问全局变量和局部变量
```

```
echo "<br>";
echo "变量 b 为: $y"; // 访问局部变量
?>
```

本程序运行结果如图 3-5 所示。

图 3-5　程序运行结果

出现上述结果是因为函数无法访问外部全局变量，但是在代码间可以访问全局变量。

如果想让函数访问某个全局变量，可以在函数中通过 global 关键字来声明，就是要告诉函数，它要调用的变量是一个已经存在或者即将创建的同名全局变量，而不是默认的本地变量。

通过下面的实例介绍 global 关键字的使用方法和技巧。

【例 3.6】（实例文件：源文件\ch03\3.6.php）

```
<?php
$a=100;   //全局变量
// 访问全局变量
function ff()
{
    $b=100; //局部变量
    global $a;          //函数内调用全局变量
    echo "变量 a 为: $a"; //访问全局变量
    echo "<br>";
    echo "变量 b 为: $b"; //访问局部变量
}

ff();                  // 函数内访问全局变量和局部变量
?>
```

本程序运行结果如图 3-6 所示。

图 3-6　程序运行结果

注意：在 PHP 7 中，global 关键字现在只能引用简单变量，例如：

```
global $$foo->bar;    // 这种写法不支持
global ${$foo->bar};  // 需用大括号来达到效果
```

另外，读者还可以通过“超全局变量”中的$GLOBALS 数组进行访问。

下面通过实例介绍$GLOBALS 数组。

【例 3.7】(实例文件：源文件\ch03\3.7.php)

```
<?php
$a=200;   //全局变量
// 访问全局变量
function ff()
{
    $b=100; //局部变量
    $a=$GLOBALS['a'];     // 通过$GLOBALS 数组访问全局变量
    echo "变量 a 为: $a"; //访问全局变量
    echo "<br>";
    echo "变量 b 为: $b"; //访问局部变量
}

ff();                  // 函数内访问全局变量和局部变量
?>
```

本程序运行结果如图 3-7 所示。

图 3-7　程序运行结果

从结果可以看出，上述两种方法都可以实现在函数内访问全局变量。

3. 静态变量

静态变量只是在函数内存在，在函数外无法访问。但是执行后，其值保留，也就是说这一次执行完毕后，静态变量的值保留，下一次再执行此函数，这个值还可以调用。

通过下面的实例介绍静态变量的使用方法和技巧。

【例 3.8】(实例文件：源文件\ch03\3.8.php)

```
<?php
   $person = 20;                    //
  function showpeople(){
     static $person = 5;
    $person++;
     echo '再增加一位客户，将会有 '.$person.' 位客户。<br>';
  }
  showpeople();
```

```
    echo $person.'人员。<br>';
    showpeople();
?>
```

本程序运行结果如图 3-8 所示。

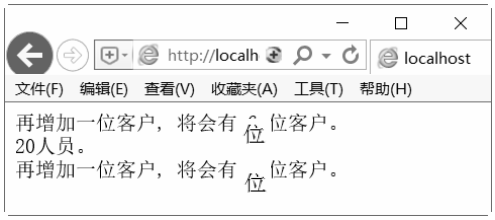

图 3-8　程序运行结果

【案例分析】:

（1）其中函数外的 echo 语句无法调用函数内的 static $person，它调用的是$person = 20。

（2）showpeople()函数被执行两次，这个过程中 static $person 的运算值得以保留，并且通过$person++进行了累加。

3.4.4　变量的销毁

当用户创建一个变量时，相应地在内存中有一个空间专门用于存储该变量，该空间引用计数加 1。当变量与该空间的联系被断开时，空间引用计数减 1，直到引用计数为 0，则成为垃圾。

PHP 有自动回收垃圾的机制，用户也可以手动销毁变量，通常使用 unset()函数来实现。该函数的语法格式如下：

```
void unset (变量)
```

其中，若变量类型为局部变量，则变量被销毁；若变量类型为全局变量，则变量不会被销毁。

【例 3.9】(实例文件：源文件\ch03\3.9.php)

```
<?php
$b= "大漠孤烟直，长河落日圆。";          //函数外声明全局变量
function xiaohui()  {                    //声明函数
    $a= 10;                              //函数外声明全局变量
    global $b;                           //函数内使用 global 关键字声明全局变量$b
    unset ($a);                          //使用 unset()销毁不再使用的变量$a
     echo  $a;                           //查看局部变量是否发生变化
}
xiaohui();                               //调用函数
echo  $b;                                //查看全局变量是否发生变化
?>
```

本程序运行结果如图 3-9 所示。

图 3-9　程序运行结果

3.5 数据类型

从 PHP 4 开始，PHP 中的变量不需要事先声明，赋值即可声明。在使用这些数据类型前，读者需要了解它们的含义和特性。下面介绍整型、浮点型、布尔型、字符串型、数组型、对象型以及 NULL 和 Resource 两个比较特殊的类型。

3.5.1　什么是类型

不同的数据类型其实就是所存储数据的不同种类。PHP 的不同数据类型主要包括：

- 整型（integer），用来存储整数。
- 浮点型（float），用来存储实数。
- 字符串型（string），用来存储字符串。
- 布尔型（boolean），用来存储真（true）或假（false）。
- 数组型（array），用来存储一组数据。
- 对象型（object），　用来存储一个类的实例。

作为弱类型语言，PHP 也被称为动态类型语言。在强类型语言（例如 C 语言）中，一个变量只能存储一种类型的数据，并且这个变量在使用前必须声明变量类型。而在 PHP 中，给变量赋什么类型的值，这个变量就是什么类型，例如以下几个变量：

```
$hello = "hello world";
```

由于 hello world 是字符串，因此变量$hello 的数据类型就是字符串类型。

```
$hello = 100;
```

同样，由于 100 为整型，因此$hello 也就是整型。

```
$wholeprice = 100.0;
```

由于 100.0 为浮点型，因此$wholeprice 就是浮点型。

由此可见，对于变量而言，如果没有定义变量的类型，则它的类型由所赋值的类型决定。

3.5.2　整型

整型是数据类型中最为基本的类型。在现有的 32 位运算器下，整型的取值是从-2147483648

到+2147483647。整型可以表示为二进制、八进制、十进制和十六进制。

要使用二进制表达，数字前必须加上 0b，要使用八进制表达，数字前必须加上 0，要使用十六进制表达，数字前必须加上 0x。

例如：

```
<?php
$a = 1234; // 十进制数
$a = -123; // 负数
$a = 0123; // 八进制数 (等于十进制 83)
$a = 0x1A; // 十六进制数 (等于十进制 26)
$a = 0b11111111; // 二进制数字 (等于十进制 255)
?>
```

在 PHP 7 中，整型值的字长可以用常量 PHP_INT_SIZE 来表示，最大值可以用常量 PHP_INT_MAX 来表示，最小值可以用常量 PHP_INT_MIN 表示。整型数的字长和平台有关，32 位平台下的最大值是 2147483647，64 位平台下的最大值通常大约是 9223372036854775807。

3.5.3 浮点型

浮点型用于表示实数。在大多数运行平台下，这个数据类型的大小为 8 个字节。它的近似取值范围是 2.2E-308~1.8E+308（科学计数法）。

例如：

```
-1.432
1E+07
0.0
```

3.5.4 布尔型

布尔型只有两个值，就是 true 和 false。布尔型是十分有用的数据类型，通过它，程序实现了逻辑判断的功能。

其他的数据类型基本都有布尔属性：

- 整型，为 0 时，其布尔属性为 false，为非零值时，其布尔属性为 true。
- 浮点型，为 0.0 时，其布尔属性为 false，为非零值时，其布尔属性为 true。
- 字符串型，为空字符串“”或者零字符串“0”时，其布尔属性为 false，包含除此以外的字符串时其布尔属性为 true。
- 数组型，若不含任何元素，其布尔属性为 false，只要包含元素，则其布尔属性为 true。
- 对象型和资源型，其布尔属性永远为 true。
- NULL 型，其布尔属性永远为 false。

3.5.5 字符串型

字符串型的数据是表示在引号之间的。引号分为双引号“"”和单引号“'”。这两种引号可以

表示字符串，但是这两种表示方法也有一定区别。

双引号几乎可以包含所有的字符，但是在其中的变量显示变量的值，而不是变量的变量名，有些特殊字符加上“\”符号就可以了；单引号内的字符是被直接表示出来的。

下面通过一个案例来讲述整型、浮点型、布尔型和字符串型数据的使用方法和技巧。

【例 3.10】（实例文件：源文件\ch03\3.10.php）

```
<?php
   $int1= 2012;      // 十进制整数
   $int2= 01223;     //八进制整数
   $int3=0x1223;     //十六进制整数
   echo "输出整数类型的值：";
   echo $int1;
   echo "\t";        //输出一个制表符
   echo $int2;       //输出 659
   echo "\t";
   echo $int3;       //输出 4643
   echo "<br>";
   $float1=54.66;
   echo $float1;     //输出 54.66
   echo "<br>";
   echo "输出布尔型变量：";
   echo (Boolean)( $int1);    //将 int1 整型转化为布尔变量
   echo "<br>";
   $string1="字符串类型的变量";
   echo $string1;
?>
```

本程序运行结果如图 3-10 所示。

图 3-10　程序运行结果

3.5.6　数组型

数组是 PHP 变量的集合，是按照“键值”与“值”的对应关系组织数据的。数组的键值既可以是整数，也可以是字符串。另外，数组不特意表明键值的默认情况下，数组元素的键值为从零开始的整数。

在 PHP 中，使用 list()函数或 array()函数来创建数组，也可以直接进行赋值。

下面使用 array()函数创建数组。

【例 3.11】（实例文件：源文件\ch03\3.11.php）

```
<?php
$arr=array                          // 定义数组并赋值
(
    0=>15,
    2=>1E+05,
    1=>"开始学习 PHP 基本语法了",
);
for ($i=0;$i<count($arr);$i++) // 使用 for 循环输出数组内容
{
  $arr1=each($arr);
  echo "$arr1[value]<br>";
}
?>
```

本程序运行结果如图 3-11 所示。

图 3-11　程序运行结果

【案例分析】

（1）程序中用“=>”为数组赋值，数组的下标只是存储的标识，没有任何意义，数组元素的排列以加入的先后顺序为准。

（2）本程序采用 for 循环语句输出整个数组，其中 count 函数返回数组的个数，echo 函数返回当前数组指针的索引值对，后面的章节将详细讲述 echo 函数的使用方法。

上面实例的语句可以简化如下。

【例 3.12】（实例文件：源文件\ch03\3.12.php）

```
<?php
$arr=array(15,1E+05,"开始学习 PHP 基本语法了");   // 定义数组并赋值
for ($i=0;$i<3;$i++)
{
  echo $arr[$i]."<br>";
}
?>
```

本程序运行结果如图 3-12 所示。从结果可以看出，这两种写法的运行结果相同。

图 3-12　程序运行结果

另外，读者还可以对数组的元素一个一个地赋值，下面举例说明。上面的语句可以简化如下。

【例 3.13】（实例文件：源文件\ch03\3.13.php）

```
<?php
$arr[0]=15;  // 对数组元素分别赋值
$arr[2]= 1E+05;
$arr[1]= "开始学习 PHP 基本语法了";
for ($i=0;$i<count($arr);$i++)
{
    $arr1=each($arr);
    echo "$arr1[value]<br>";
}
?>
```

本程序运行结果如图 3-13 所示。从结果可以看出，一个一个赋值的方法和上面两种写法的运行结果一样。

图 3-13　程序运行结果

3.5.7　对象型

对象就是类的实例。当一个类被实例化以后，这个被生成的对象被传递给一个变量，这个变量就是对象型变量。对象型变量也属于资源型变量。

3.5.8　NULL 型

NULL 类型是仅拥有 NULL 这个值的类型。这个类型用来标记一个变量为空。一个空字符串与 NULL 是不同的。在数据库存储时会把空字符串和 NULL 区分开处理。NULL 型在布尔判断时永远为 false。很多情况下，在声明一个变量的时候可以直接先赋值为 NULL 型，如$value = NULL。

3.5.9 资源类型

资源（resource）类型是十分特殊的数据类型。它表示 PHP 的扩展资源，可以是一个打开的文件，也可以是一个数据库连接，甚至可以是其他的数据类型。但是在编程过程中，资源类型却是几乎永远接触不到的。

3.5.10 数据类型之间的相互转换

数据从一个类型转换到另一个类型，就是数据类型转换。在 PHP 语言中，有两种常见的转换方式：自动数据类型转换和强制数据类型转换。

1. 自动数据类型转换

这种转换方法最为常用。直接输入数据的转换类型即可。

【2.14】(实例文件：源文件\ch02\2.14.php)

```
<?php
$a = "2";  //$a 是字符串
echo $a;
echo "<br>";
$a*=2;   //$a 现在是一个整数
echo $a;
echo "<br>";
$a*=1.4;  //$a 现在是一个浮点数
echo $a;
?>
```

程序运行结果如图 2-14 所示。

图 2-14 自动数据类型转换

2. 强制数据类型转换

在 PHP 语言中，可以使用 settype 函数强制转换数据类型，基本语法如下。

```
Bool settype(var,string type)
```

type 的可能值不能包含资源类型数据。如果转型成功，就返回 1，否则返回 0。

【例 3.15】(实例文件：源文件\ch03\3.15.php)

```
<?php
```

```
$flo1=100.86;  // 定义浮点型数据
echo settype($flo1,"int");  强制转换数据为整数并输出
echo "<br>";
echo $flo1;
?>
```

本程序运行结果如图 3-15 所示。这里返回结果为 1，说明浮点数 100.86 转型为整数 100 已经成功了。

图 3-15　程序运行结果

3.6 PHP 7 的新变化 1——整型处理机制的修改

PHP 7 以前的版本里，如果向八进制数传递了一个非法数字，例如 8 或 9，则后面其余数字会被忽略。在 PHP 7 中，将会出现编译错误。

例如下面的代码将会报错：

```
$a = 0792; // 9 是无效的八进制数字
```

在 PHP 7 中，如果位移负的位置，将会产生异常，例如：

```
var_dump(1 >> -1);
 // ArithmeticError: Bit shift by negative number
```

在 PHP 7 中，左位移如果超出位数，就会返回为 0，例如：

```
var_dump(1 << 64); // int(0)
```

而在 PHP 7 之前的版本中，运行结果和 CPU 的架构有关系，比如 x86 会返回 1。

在 PHP 7 中，右位移如果超出位数，就会返回 0 或者-1，例如：

```
var_dump(1 >> 64);  // int(0)
var_dump(-1 >> 64); // int(-1)
```

3.7 PHP 7 的新变化 2——标量类型的声明

默认情况下，所有的 PHP 文件都处于弱类型校验模式。PHP 7 加了标量类型声明的特性，标量类型声明有两种模式：强制模式（默认）和严格模式。

标量类型声明的语法格式如下：

```
declare(strict_types=1);
```

通过指定 strict_types 的值（1 或者 0）来表示校验模式：1 表示严格类型校验模式，作用于函数调用和返回语句；0 表示强制类型校验模式。

可以声明标量类型的参数类型包括 int、float、bool、string、interfaces、array 和 callable。

1. 强制模式

下面通过案例来学习强制模式的含义，代码如下：

```
<?php
// 强制模式
function sum(int $ints)
{
   return array sum($ints);
}
print(sum(2, '3', 4.1));
?>
```

上面程序的输出结果为 9。代码中的'3'转化为 3，4.1 先转换为整数 4，再进行相加操作。

2. 严格模式

下面通过案例来学习严格模式的含义，代码如下：

```
<?php
// 严格模式
declare(strict_types=1);
function sum(int  $ints)
{
   return array_sum($ints);
}
print(sum(2, '3', 4.1));
?>
```

以上程序由于采用了严格模式，因此如果参数中出现的不是整数类型，程序执行时就会报错，如图 3-16 所示。

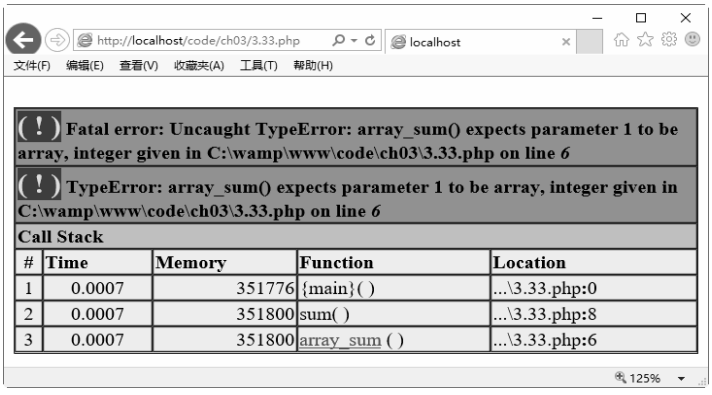

图 3-16　错误提示信息

3.8 运算符

PHP 包含多种类型的运算符，常见的有算术运算符、字符串运算符、赋值运算符、比较运算符和逻辑运算符等。

3.8.1 算术运算符

算术运算符是最简单、最常用的运算符。常见的算术运算符如表 3-1 所示。

表 3-1　常见的算术运算符

运算符	名称	运算符	名称
+	加法运算符	%	取余运算符
-	减法运算符	++	累加运算符
*	乘法运算符	--	累减运算符
/	除法运算符		

算术运算符的用法如下面的实例所示。

【例 3.16】（实例文件：源文件\ch03\3.16.php）

```
<?php
$a=13;            // 定义变量
$b=2;
echo $a."+".$b."=";
echo $a+$b."<br>";  //使用加法运算符
echo $a."-".$b."=";
echo $a-$b."<br>";  //使用减法运算符
echo $a."*".$b."=";
echo $a*$b."<br>";  //使用乘法运算符
echo $a."/".$b."=";
echo $a/$b."<br>";  //使用除法运算符
echo $a."%".$b."=";
echo $a%$b."<br>";  //使用求余运算符
echo $a."++"."=";
echo $a++."<br>";  //使用累加运算符
echo $a."--"."=";
echo $a--."<br>";  //使用累减运算符
?>
```

本程序运行结果如图 3-16 所示。

图 3-16　程序运行结果

除了数值可以进行自增运算外，字符也可以进行自增运算操作。例如，b++的结果将等于 c。

3.8.2 字符串运算符

字符串运算符是把两个字符串连接起来变成一个字符串的运算符，使用“.”来完成。如果变量是整型或浮点型，PHP 也会自动把它们转换为字符串输出，如下面的实例所示。

【例 3.17】（实例文件：源文件\ch03\3.17.php）

```
<?php
$a = "把两个字符串";                  // 定义字符串变量
$b = 100;
echo $a."连接起来，".$b."天。";   // 把字符串连接后输出
?>
```

本程序运行结果如图 3-17 所示。

图 3-17 程序运行结果

3.8.3 赋值运算符

赋值运算符的作用是把一定的数据值加载给特定变量。

赋值运算符的具体含义如表 3-2 所示。

表 3-2 赋值运算符的含义

赋值运算符	含义
=	将右边的值赋给左边的变量
+=	将左边的值加上右边的值赋给左边的变量
-=	将左边的值减去右边的值赋给左边的变量
*=	将左边的值乘以右边的值赋给左边的变量
/=	将左边的值除以右边的值赋给左边的变量
.=	将左边的字符串连接到右边
%=	将左边的值对右边的值取余数赋给左边的变量

例如，$a-=$b 等价于$a=$a-$b，其他赋值运算符与之类似。从表 3-2 可以看出，赋值运算符可以使程序更加简练，从而提高执行效率。

3.8.4 比较运算符

比较运算符用来比较两端数据值的大小。比较运算符的具体含义如表 3-3 所示。

表 3-3　比较运算符的含义

比较运算符	含义	比较运算符	含义
==	相等	>=	大于等于
! =	不相等	<=	小于等于
>	大于	===	精确等于（类型也要相同）
<	小于	!==	不精确等于

其中，“===”和“!==”需要特别注意一下。$b===$c 表示$b 和$c 不只是数值上相等，而且两者的类型也一样；$b!==$c 表示$b 和$c 有可能是数值不等，也可能是类型不同。

【例 3.18】（实例文件：源文件\ch03\3.18.php）

```
<?php
$value="15";
echo "\$value = \"$value\"";
echo "<br>\$value==15: ";
var dump($value==15);           //结果为:bool(true)
echo "<br>\$value==ture: ";
var dump($value==ture);         //结果为:bool(true)
echo "<br>\$value!=null: ";
var dump($value!=null);         //结果为:bool(true)
echo "<br>\$value==false: ";
var dump($value==false);        //结果为:bool(false)
echo "<br>\$value === 100: ";
var dump($value===100);         //结果为:bool(false)
echo "<br>\$value===true: ";
var dump($value===true);        //结果为:bool(false)
echo "<br>(10/2.0 !== 5): ";
var dump(10/2.0 !==5);          //结果为:bool(true)
?>
```

本程序运行结果如图 3-18 所示。

图 3-18　程序运行结果

3.8.5 逻辑运算符

编程语言最重要的功能之一就是进行逻辑判断和运算。逻辑和、逻辑或、逻辑否都是逻辑运算符。逻辑运算符的含义如表 3-4 所示。

表 3-4 逻辑运算的含义

逻辑运算符	含义	逻辑运算符	含义
&&	逻辑和	!	逻辑否
AND	逻辑和	NOT	逻辑否
||	逻辑或	XOR	逻辑异或
OR	逻辑或		

【例 3.19】（实例文件：源文件\ch03\3.19.php）

```
<?php
$a = true;
$b = false;
echo '$a && $b: ';
var_dump($a && $b);          //使用逻辑与运算符，结果为:false
echo '$a || $b: ';
var_dump($a || $b);          //使用逻辑或运算符，结果为:true
echo '!$a: ';
var_dump(!$a );              //使用逻辑否运算符，结果为:false
?>
```

本程序运行结果如图 3-19 所示。

图 3-19 程序运行结果

3.8.6 按位运算符

按位运算符是把整数按照“位”的单位来进行处理。按位运算符的含义如表 3-5 所示。

表 3-5　按位运算符的含义

按位运算符	名称	含义
&	按位和	例如，$a&$b，表示对应位数都为 1，结果该位为 1
|	按位或	例如，$a\|$b，表示对应位数有一个为 1，结果该位为 1
^	按位异或	例如，$a^$b，表示对应位数不同，结果该位为 1
~	按位取反	例如，~$b，表示对应位数为 0 的改为 1，为 1 的改为 0
<<	左移	例如，$a<<$b，表示将$a 在内存中的二进制数据向左移动$b 位数，右边移空补 0
>>	右移	例如，$a>>$b，表示将$a 在内存中的二进制数据向右移动$b 位数，左边移空补 0

【例 3.20】（实例文件：源文件\ch03\3.20.php）

```
<?php
$a = 7;                                          // 7 的二进制代码是 111
$b = 4;                                          // 4 的二进制代码是 100
echo '$a & $b = ' . ($a & $b) . '<br/>';         // 运算结果为二进制代码 100，即 4
echo '$a | $b = ' . ($a | $b) . '<br/>';         // 运算结果为二进制代码 111，即 7
echo '$a ^ $b = ' . ($a ^ $b) . '<br/>';         // 运算结果为二进制代码 011，即 3
?>
```

本程序运行结果如图 3-20 所示。

图 3-20　程序运行结果

3.8.7　否定控制运算符

否定控制运算符用在“操作数”之前，用于对操作数值进行真假的判断。它包含一个逻辑否定运算符和一个按位否定运算符。否定控制运算符的含义如表 3-6 所示。

表 3-6　否定控制运算符的含义

否定控制运算符	含义
!	逻辑否
~	按位否

3.8.8　错误控制运算符

错误控制运算符用“@”来表示。在一个操作数之前使用，用来屏蔽错误信息的生成。

【例 3.21】（实例文件：源文件\ch03\3.21.php）

```
<?php
```

```
$err = @(20 / 0) ;   // 如果不想显示这个错误，在表达式前加上“@”
?>
```

本程序运行结果如图 3-21 所示。

图 3-21　程序运行结果

3.8.9　三元运算符

三元运算符作用在三个操作数之间。这样的操作符在 PHP 中只有一个，即“？：”，语法形式如下：

```
(expr1) ? (expr2) : (expr3)
```

如果 expr1 成立，就执行 expr2，否则执行 expr3。

【例 3.22】（实例文件：源文件\ch03\3.22.php）

```
<?php
$a = 5;
$b = 6;
echo ($a > $b) ? "大于成立" : "大于不成立"; echo "<br/>";   //大于不成立
echo ($a < $b) ? "小于成立" : "小于不成立"; echo "<br/>";  //小于成立
?>
```

本程序运行结果如图 3-22 所示。

图 3-22　程序运行结果

3.8.10　运算符的优先级和结合规则

运算符的优先级和结合其实与正常的数学运算符的规则十分相似。

- 加减乘除的先后顺序与数学运算中的完全一致。
- 对于括号，先括号内再括号外。
- 对于赋值，由右向左运行，即依次从右边向左边的变量进行赋值。

3.9 PHP 7 的新变化 3——合并运算符和组合运算符

PHP 7 新增加的合并运算符“??”用于判断变量是否存在且值不为 NULL，如果是，它就会返回自身的值，否则返回它的第二个操作数。

语法格式如下：

```
(expr1) ? ? (expr2)
```

如果表达式 expr1 为真，就返回 expr1 的值；如果表达式 expr1 为假，就返回 expr2。

【例 3.23】（实例文件：源文件\ch03\3.23.php）

```
<?php
$a = '酒店还有房间';
$b = $a ?? '酒店已经没有房间';
echo $b;
?>
```

代码运行结果如图 3-23 所示。

图 3-23　合并运算符

PHP 7 新增加的组合运算符用于比较两个表达式$a 和$b，$a 小于、等于或大于$b 分别返回-1、0 或 1。

```
<?php
// 整型比较
echo( 5 <=> 5);echo "<br>";
echo( 5 <=> 6);echo "<br>";
echo( 6 <=> 5);echo "<br>";

// 浮点型比较
echo( 5.6 <=> 5.6);echo "<br>";
echo( 5.6 <=> 6.6);echo "<br>";
echo( 6.6 <=> 5.6);echo "<br>";
echo(PHP_EOL);

// 字符串比较
echo( "a" <=> "a");echo "<br>";
```

```
echo( "a" <=> "b");echo "<br>";
echo( "b" <=> "a");echo "<br>";
?>
```

代码运行结果如图 3-24 所示。

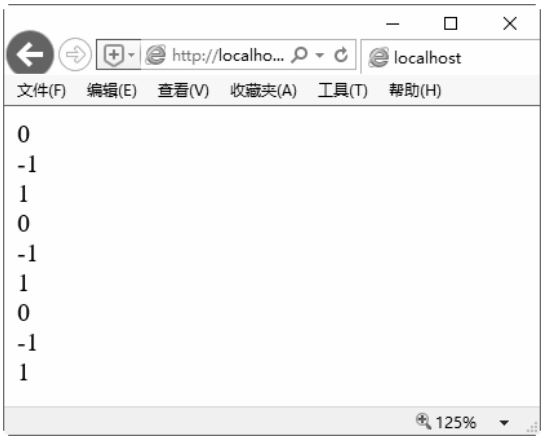

图 3-24　组合运算符

3.10 表达式

表达式是在特定语言中表达一个特定的操作或动作的语句。PHP 的表达式也有同样的作用。

一个表达式包含“操作数”和“操作符”。操作数可以是变量，也可以是常量。操作符则体现了要表达的各个行为，如逻辑判断、赋值、运算等。

例如，$a=5 就是表达式，而$a=5; 则为语句。另外，表达式也有值，例如$a=1 表达式的值为 1。

在 PHP 代码中，使用“;”号来区分表达式，即一个表达式和一个分号组成了一条 PHP 语句。在编写程序代码时，应该特别注意表达式后面的“;”，不要漏写或写错，否则会提示语法错误。

3.11 实战演练——创建二维数组

前面讲述了如何创建一维数组，本节讲述如何创建多维数组。多维数组和一维数组的区别是多维数组有两个或多个下标，它们的用法基本相似。

下面以创建二维数组为例进行讲解。

【例 3.24】（实例文件：源文件\ch03\3.24.php）

```
<?php
$arr[0][0]=10;                          //定义数组元素并赋值
$arr[0][1]=22;
$arr[1] [0]= 1E+05;
$arr[1] [1]= "开始学习 PHP 基本语法了";
for ($i=0;$i<count($arr);$i++)          //使用 for 语句输出二维数组
{
   for ($k=0;$k<count($arr[$i]);$k++)
{
$arr1=each($arr[$i]);
echo "$arr1[value]<br>";
}
}
?>
```

本程序运行结果如图 3-25 所示。

图 3-25　程序运行结果

3.12 高手甜点

甜点 1：如何灵活运用命名空间（namespace）？

命名空间作为一个比较宽泛的概念，可以理解为用来封装各个项目的方法。有点像是在文件系统中不同文件夹路径和文件夹当中的文件。两个文件的文件名可以完全相同，但是在不同的文件夹路径下，就是两个完全不同的文件。

PHP 的命名空间也是这样的一个概念。它主要用于在“类的命名”“函数命名”及“常量命名”中避免代码冲突和在命名空间下管理变量名和常量名。

命名空间是使用 namespace 关键字在文件头部定义的，例如：

```
<?php
namespace 2ndbuilding\number24;  //命名空间
class room{}
$room = new __NAMESPACE__.room;
?>
```

命名空间还可以拥有子空间，就像文件夹的路径一样。可以通过内置变量_NAMESPACE_来使用命名空间及其子空间。

甜点 2：如何快速区分常量与变量？

常量和变量的明显区别如下：

- 常量前面没有美元符号（$）。
- 常量只能用 define() 函数定义，而不能通过赋值语句定义。
- 常量可以不用理会变量范围的规则而在任何地方定义和访问。
- 常量一旦定义就不能被重新定义或者取消定义。
- 常量的值只能是标量。

第 4 章
PHP的语言结构

学习目标 Objective

任何一种语言都有程序结构，常见的有顺序结构、分支结构和循环结构。在学习程序结构前，读者还需要对函数的知识进行学习。本章主要介绍 PHP 语言中函数和语言结构的使用方法和技巧。

内容导航 Navigation

- 熟悉函数的使用方法
- 熟悉流程控制的概述
- 掌握条件控制结构
- 掌握循环控制结构
- 掌握条件分支结构的综合应用
- 掌握循环控制结构的综合应用

4.1 内置函数

函数的英文为 function，function 是功能的意思。顾名思义，使用函数就是要在编程过程中实现一定的功能，即通过代码块来实现一定的功能。比如通过一定的功能记录下酒店客人的个人信息，每到他生日的时候自动给他发送祝贺 email，并且这个发信“功能”可以重用，可更改为在某个客户的结婚纪念日时给他发送祝福 email。所以函数就是实现一定功能的一段特定的代码。

PHP 提供了大量的内置函数，方便程序员直接使用，常见的内置函数包括数学函数、字符串函数、时间和日期函数等。

下面以调用数学函数 rand()为例进行讲解。

【例 4.1】（实例文件：源文件\ch04\4.1.php）

```
<?php
echo rand ();                //返回随机整合
echo "<br>";
echo rand (1000,9999);      //产生一个 4 位随机整数
?>
```

本程序运行结果如图 4-1 所示。

图 4-1　程序运行结果

4.2 自定义函数

其实，更多的情况下，程序员需要的是自定义函数。

4.2.1 自定义和调用函数

自定义函数的语法结构如下：

```
function name_of_function( param1, param2, … ){
      statement
}
```

其中，name_of_function 是函数名，param1、param2 是参数，statement 是函数的具体内容。下面以自定义和调用函数为例进行讲解。本实例主要实现酒店欢迎信息。

【例 4.2】（实例文件：源文件\ch04\4.2.php）

```
<?php
function sayhello($customer){            //自定义函数 sayhello
   return $customer.", 欢迎您来到润慧酒店。";
}
echo sayhello('张先生');                 //调用函数 sayhello
?>
```

本程序运行结果如图 4-2 所示。

图 4-2　程序运行结果

值得一提的是，此函数的返回值是通过值返回的。也就是说 return 语句返回值时，创建了一个值的副本，并把它返回给使用此函数的命令或函数，在这里是 echo 命令。

4.2.2　向函数传递参数值

由于函数是一段封闭的程序，因此很多时候程序员都需要向函数内传递一些数据来进行操作。

```
function 函数名称（参数 1，参数 2）{
   算法描述，其中使用参数 1 和参数 2；
}
```

下面以计算酒店房间住宿费总价为例进行讲解。

【例 4.3】（实例文件：源文件\ch04\4.3.php）

```
<?php
function totalneedtopay($days,$roomprice){      // 声明自定义函数
      $totalcost = $days*$roomprice;            // 计算住宿费总价
        echo  "需要支付的总价:$totalcost"."元。"; // 计算住宿费总价
}
$rentdays = 3;                                  //声明全局变量
$roomprice = 168;
totalneedtopay($rentdays,$roomprice);           //通过变量传递参数
totalneedtopay(5,198);                          //直接传递参数值
?>
```

运行结果如图 4-2 所示。

图 4-3　程序运行结果

【案例分析】

（1）以这种方式传递参数值的方法就是向函数传递参数值。

（2）其中 function totalneedtopay($days,$roomprice){}定义了函数和参数。

（3）无论是通过变量$rentdays 和$roomprice 向函数内传递参数值，还是像 totalneedtopay (5,198) 这样直接传递参数值都是一样的。

4.2.3　向函数传递参数引用

向函数传递参数引用其实就是向函数传递变量引用。参数引用一定是变量引用，静态数值是没有引用一说的。由于在变量引用中已经知道，变量引用其实就是对变量名的使用，是对特定的变量位置的使用。

下面仍然以计算酒店服务费总价为例进行讲解。

【例 4.4】（实例文件：源文件\ch04\4.4.php）

```
<?php
```

```
    $fee = 300;
    $serviceprice = 50;
    function totalfee(&$fee,$serviceprice){      // 声明自定义函数，参数前多了&，表示
按引用传递
            $fee = $fee+$serviceprice;          // 改变形参的值，实参的值也会发生改变
             echo "需要支付的总价:$fee"."元。";
    }
    totalfee($fee,$serviceprice);    //函数外部调用 fun()函数前$fee =300
    totalfee($fee,$serviceprice);    //函数外部调用 fun()函数后$ fee =350
    ?>
```

运行结果如图 4-4 所示。

图 4-4　程序运行结果

【案例分析】

（1）以这种方式传递参数值的方法就是向函数传递参数引用。使用“&”符号表示参数引用。

（2）其中 function totalfee(&$fee,$serviceprice){}定义了函数、参数和参数引用。变量$fee 是以参数引用的方式进入函数的。当函数的运行结果改变了变量$fee 的引用时，在函数外的变量$fee 的值也发生了改变，也就是函数改变了外部变量的值。

4.2.4　从函数中返回值

在上述例子中，都是把函数运算完成的值直接打印出来。但是，很多情况下，程序并不需要直接把结果打印出来，而是仅仅给出结果，并且把结果传递给调用这个函数的程序，为其所用。

这里需要用到 return 关键字。下面以综合酒店客房价格和服务价格为例进行讲解。

【例 4.5】（实例文件：源文件\ch04\4.5.php）

```
    <?php
    function totalneedtopay($days,$roomprice){     // 声明自定义函数
           return $days*$roomprice;          // 返回酒店消费总价格
    }
    $rentdays = 3;
    $roomprice = 168;
    echo totalneedtopay($rentdays,$roomprice);
    ?>
```

运行结果如图 4-5 所示。

图 4-5　程序运行结果

【案例分析】

（1）在函数 function totalneedtopay($days,$roomprice)算法中，直接使用 return 把运算的值返回给调用此函数的程序。

（2）其中，echo totalneedtopay($rentdays,$roomprice);语句调用了此函数，totalneedtopay()把运算值返回给了 echo 语句，才有了上面的显示。当然这里也可以不用 echo 来处理返回值，也可以对它进行其他处理，比如赋值给变量等。

4.2.5　对函数的引用

无论是 PHP 中的内置函数，还是程序员在程序中自定义的函数，都可以简单地通过函数名调用。但是操作过程也有些不同，大致分为以下 3 种情况。

- 如果是 PHP 的内置函数，如 date()，可以直接调用。
- 如果这个函数是 PHP 的某个库文件中的函数，就需要用 include()或 require()命令把此库文件加载，然后才能使用。
- 如果是自定义函数，与引用程序在同一个文件中，就可以直接引用。如果此函数不在当前文件内，就需要用 include()或 require()命令加载。

对函数的引用实际上是对函数返回值的引用。

【例 4.6】（实例文件：源文件\ch04\4.6.php）

```
<?php
function &example($aa=1){                    //定义一个函数，别忘了加“&”符号
return $aa;                                 //返回参数$aa
}
$bb= &example("引用函数的实例");              //声明一个函数的引用$bb
echo $bb. "<br>";
?>
```

运行结果如图 4-6 所示。

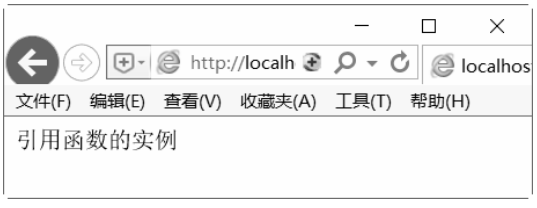

图 4-6　程序运行结果

【案例分析】

（1）本实例首先定义一个函数，然后变量$bb 将引用函数，最后输出变量$bb，它实际上是$aa 的值。

（2）和参数传递不同，在定义函数和引用函数时，都必须使用“&”符号，表明返回的是一个引用。

4.2.6 对函数取消引用

对于不需要引用的函数，可以做取消操作。取消引用使用 unset()函数来完成，目的是断开变量名和变量内容之间的绑定，此时并没有销毁变量内容。

【例 4.7】(实例文件：源文件\ch04\4.7.php)

```
<?php
$num = 166;                              //声明一个整型变量
$math = &$num;                           //声明一个对变量$num 的引用$math
echo "\$math is:  ".$math."<br>";        //输出引用$math
unset($math);                            //取消引用$math
echo "\$num is:  ".$num;                 //输出原变量
?>
```

运行结果如图 4-7 所示。

图 4-7 程序运行结果

本程序首先声明一个变量和变量的引用，输出引用后取消引用，再次调用原变量。从图 4-7 可以看出，取消引用后对原变量没有任何影响。

4.3 PHP 7 的新变化 1——声明函数返回值的类型

在 PHP 7 中，用户可以声明函数返回值的类型。可以声明的返回类型包括 int、float、bool、string、interfaces、array 和 callable。

下面通过案例来学习 PHP 7 如何声明函数返回值的类型。

【例 4.8】(实例文件：源文件\ch04\4.8.php)

```
<?php
declare(strict_types=1);
```

```
function returnIntValue(int $value): int
{
   return $value;
}

print(returnIntValue(60));
?>
```

以上程序执行结果如图 4-8 所示。

图 4-8　声明函数返回值的类型

4.4　PHP 7 的新变化 2——新增 intdiv()函数

在 PHP 7 中，新增了整除函数 intdiv()。语法格式如下：

```
intdiv(a, b);
```

该函数返回值为 a 除以 b 的值并取整。

【例 4.9】(实例文件：源文件\ch04\4.9.php)

```
<?php
echo intdiv(16, 3)."<br/>";
echo intdiv(10, 3) ."<br/>";
echo intdiv(8, 16) ."<br/>";
?>
```

本程序运行结果如图 4-9 所示。

图 4-9　程序运行结果

4.5 PHP 7 的新变化 3——括号在变量或函数中变化

在 PHP 7 中，用括号把变量或者函数括起来将不再起作用。

【例 4.10】(实例文件：源文件\ch04\4.10.php)

```
<?php
function getArray()
{
return [100, 200, 300,400];
}
$last = array_pop(getArray());
//所有版本的 PHP 在这里将会报错
$last = array_pop((getArray()));
//PHP5 或者更早的版本将不会报错
?>
```

注意第二句的调用是用圆括号包了起来，但还是报这个严格错误，如图 4-10 所示。

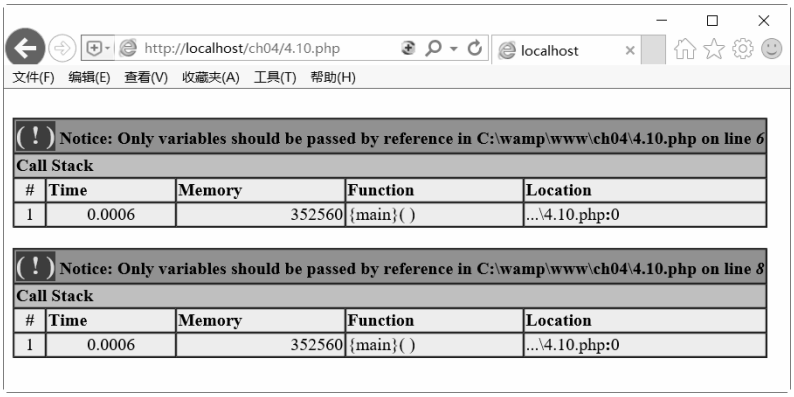

图 4-10　PHP 7 中的报错信息

PHP 7 之前的版本是不会报第 2 个错误的。例如，在 PHP 5 中的运行结果如图 4-11 所示。

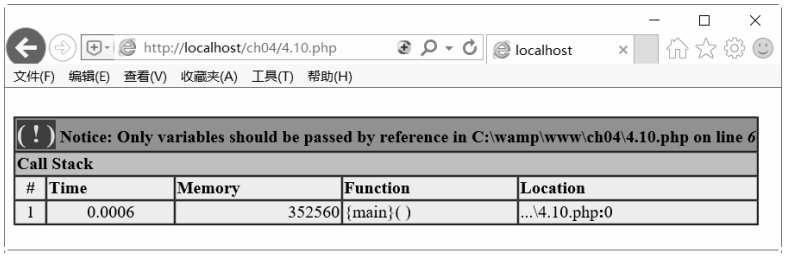

图 4-11　PHP 5 中的报错信息

4.6 PHP 7 的新变化 4——参数处理机制的修改

在 PHP 7 中，处理参数的机制出现了新的变化。

1. 不再支持重复参数命名

比如下面的代码执行的时候会报错：

```
public function ff($a, $b, $c, $c) {
        //
}
```

2. func_get_arg()和 func_get_args()函数的调整

在 PHP 7 中，func_get_arg()和 func_get_args()两个函数将返回参数当前的值，而不是传入时的值。当前的值有可能会被修改，所以当前值和传入时的值不一定相同。

【例 4.11】(实例文件：源文件\ch04\4.11.php)

```
<?php
function ff($x)
{
  $x++;
  var_dump(func_get_arg(0));
  var_dump(func_get_args(0));
}
ff(100);
?>
```

在 PHP 7 中，上面的代码会打印 101，而不是 100。运行结果如图 4-12 所示。如果想打印原始的值，调用的顺序调整一下即可，也就是将 x++操作放到两个函数的后面。

图 4-12　在 PHP 7 中的运行结果

在 PHP 5 或者更早的版本中，运行结果如图 4-13 所示。

图 4-13　在 PHP 5 中的运行结果

3. 在打印异常回溯信息时将显示修改后的值

例如下面的代码：

```
function foo($x)
{
    $x = 100;
    throw new Exception;
}
foo("这里是原始值");
```

在 PHP 7 中的运行结果：

```
Stack trace:
#0 file.php(4): foo(42)
#1 {main}
```

在 PHP 5 中的运行结果：

```
Stack trace:
#0 file.php(4): foo('string')
#1 {main}
```

这个调整不会影响代码的运行，不过在调试的时候需要注意这个变化。

4.7 包含文件

如果想让自定义的函数被多个文件使用，可以将自定义函数组织到一个或者多个文件中，这些收集函数定义的文件就是用户自己创建的 PHP 函数库。通过使用 require ()和 include()等函数可以将函数库载入脚本程序中。

4.7.1 require 和 include

require()和 include()语句不是真正意义的函数，属于语言结构。通过 include()和 require()语句都可以实现包含并运行指定文件。

（1）require()：在脚本执行前读入它包含的文件，通常在文件的开头和结尾处使用。

（2）include()：在脚本读到它的时候才将包含的文件读进来，通常在流程控制的处理区使用。

require()和 include()语句对于处理失败方面是不同的。当文件读取失败后，require 将产生一个致命错误，而 include 则产生一个警告。可见，如果遇到文件丢失需要继续运行，则使用 include；如果想停止处理页面，则使用 require。

【例 4.12】（实例文件：源文件\ch04\4.12.php 和 test.php）

其中，4.12.php 代码如下：

```
<?php
$a = '杨柳青青江水平';     //定义一个变量 a
$b = '闻郎江上唱歌声';    //定义一个变量 b
?>
```

test.php 代码如下：

```
<?php
echo " $a $b";   //未载入文件前调用两个变量
include '4.12.php';
echo " $a $b ";  //载入文件后调用两个变量
?>
```

运行 test.php，结果如图 4-14 所示。从结果可以看出，使用 include 时，虽然出现了警告，但是脚本程序仍然在运行。

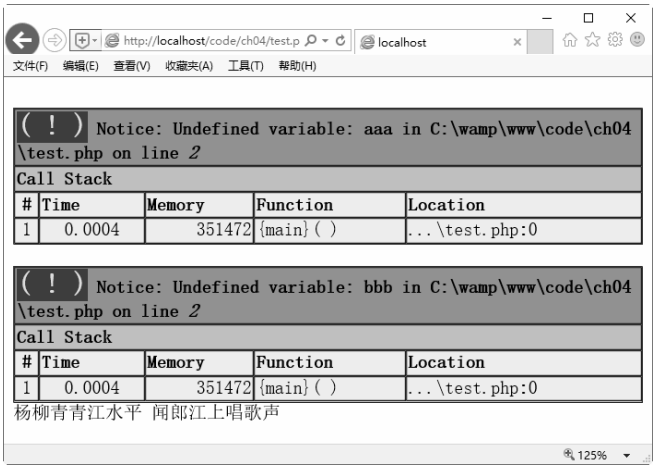

图 4-14　程序运行结果

4.7.2　include_once 和 require_once

include_once 和 require_once 语句在脚本执行期间包含并运行指定文件，作用与 include 和 require 语句类似，唯一的区别是，如果该文件的代码被包含了，则不会再次包含，只会包含一次，从而避免函数重定义以及变量重赋值等问题。

4.8 流程控制

流程控制也叫控制结构，在一个应用中用来定义执行程序流程。它决定了某个程序段是否会被执行和执行多少次。

PHP 中的控制语句分为 3 类：顺序控制语句、条件控制语句和循环控制语句。其中，顺序控制语句是从上到下依次执行的，这种结构没有分支和循环，是 PHP 程序中最简单的结构。下面主要讲述条件控制语句和循环控制语句。

4.8.1 条件控制结构

条件控制语句中包含两个主要的语句，一个是 if 语句，另一个是 switch 语句。

1. 单一条件分支结构（if 语句）

if 语句是最为常见的条件控制语句，格式为：

```
if（条件判断语句）{
        命令执行语句;
}
```

这种形式只是对一个条件进行判断。如果条件成立，就执行命令语句，否则不执行。

if 语句的流程控制图如图 4-15 所示。

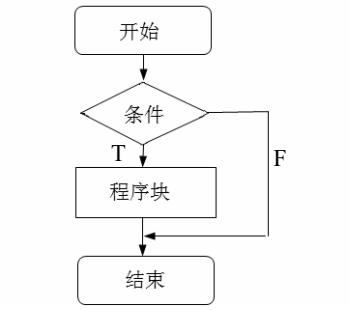

图 4-15　if 语句流程控制图

【例 4.13】（实例文件：源文件\ch04\4.13.php）

```
<?php
$num = rand(1,100);                          //使用 rand()函数生成一个随机数
if ($num % 2 != 0){                          //判断变量$num 是否为奇数
    echo "\$num = $num";                     //如果为奇数，输出表达式和说明文字
    echo "<br>$num 是奇数。";
}
?>
```

运行后刷新页面，结果如图 4-16 所示。

图 4-16　程序运行结果

【案例分析】

（1）此实例首先使用 rand()函数随机生成一个整数$num，然后判断这个随机整数是不是奇数，若是，则输出上述结果，若不是，则不输出任何内容，所以如果页面内容显示为空，就刷新页面。

（2）rand() 函数返回随机整数，语法格式如下：

```
rand(min,max)
```

此函数主要是返回 min 和 max 之间的一个随机整数。如果没有提供可选参数 min 和 max，则 rand() 返回 0 到 RAND_MAX 之间的伪随机整数。

2. 双向条件分支结构（if...else 语句）

如果是非此即彼的条件判断，可以使用 if…else 语句。它的格式为：

```
if（条件判断语句）{
      命令执行语句 A;
}else{
      命令执行语句 B;
}
```

这种结构形式首先判断条件是否为真，如果为真，就执行命令语句 A，否则执行命令语句 B。if…else 语句程序流程控制图如图 4-17 所示。

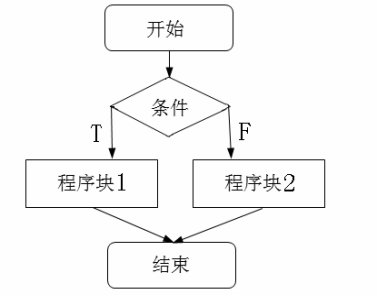

图 4-17　if…else 语句流程控制图

【例 4.14】（实例文件：源文件\ch04\4.14.php）

```
<?php
$d=date("D");                    //定义时间变量
if ($d=="Fri")                   //判断时间变量是否等于周五
  echo "今天是周五哦!";
else
  echo "可惜今天不是周五!";
```

```
?>
```

运行后结果如图 4-18 所示。

图 4-18　程序运行结果

3. 多向条件分支结构（elseif 语句）

在条件控制结构中，有时会出现多种选择，此时可以使用 elseif 语句。它的语法格式为：

```
    if（条件判断语句）{
          命令执行语句;
}elseif（条件判断语句）{
          命令执行语句;
}…
 else{
          命令执行语句;
}…
```

elseif 语句程序流程控制图如图 4-19 所示。

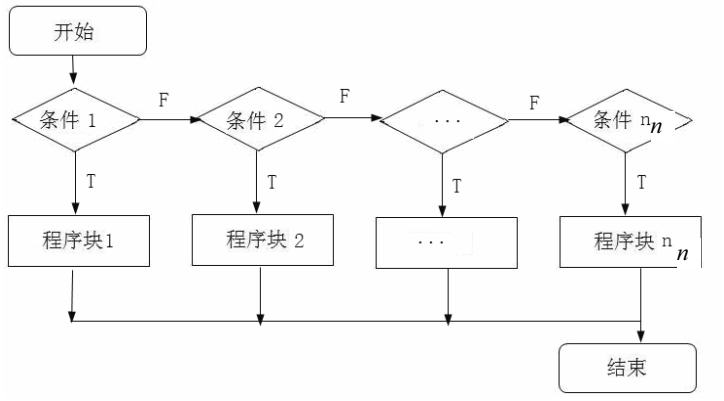

图 4-19　elseif 语句流程控制图

【例 4.15】（实例文件：源文件\ch04\4.15.php）

```
<?php
$score = 85;                          //设置成绩变量$score
if($score >= 0 and $score <= 60){   //判断成绩变量是否在 0~60 之间
echo "您的成绩为差";                    //如果是，说明成绩为差
}
elseif($score > 60 and $score <= 80){   //否则判断成绩变量是否在 61~80 之间
echo "您的成绩为中等";                    //如果是，说明成绩为中等
}else{                                  //如果两个判断都是 false，则输出默认值
echo "您的成绩为优等";              //说明成绩为优等
```

```
}
?>
```

运行后结果如图 4-20 所示。

图 4-20　程序运行结果

4. 多向条件分支结构（switch 语句）

switch 语句的结构给出不同情况下可能执行的程序块，条件满足哪个程序块，就执行哪个语句。它的语法格式为：

```
switch（条件判断语句）{
    case 可能判断结果 a:
          命令执行语句;
break;
    case 可能判断结果 b:
          命令执行语句;
break;
          …
    default:
          命令执行语句;
}
```

其中，若“条件判断语句”的结果符合某个“可能判断结果”，就执行其对应的“命令执行语句”。如果都不符合，则执行 default 对应的默认项的“命令执行语句”。

switch 语句的流程控制图如图 4-21 所示。

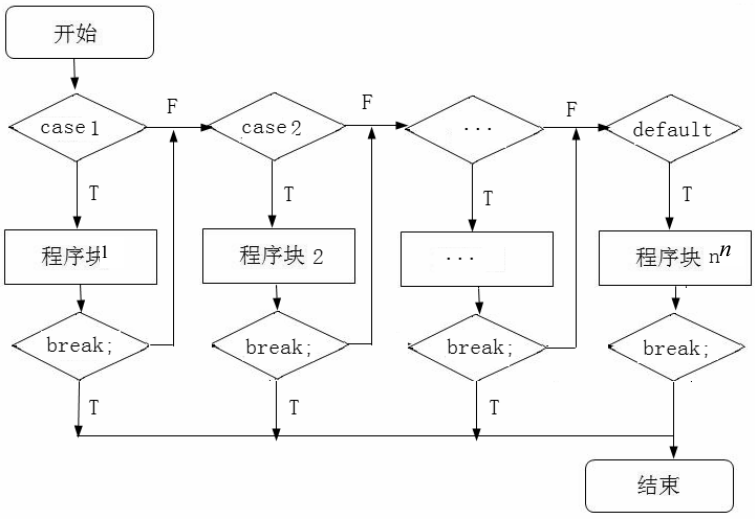

图 4-21　switch 语句流程控制图

【例 4.16】（实例文件：源文件\ch04\4.16.php）

```
<?php
$x=5;          //定义变量$x
switch ($x)      //判断$x 与 1~5 之间的关系
{
case 1:
  echo "数值为 1";
  break;
case 2:
  echo "数值为 2";
  break;
case 3:
  echo "数值为 3";
  break;
case 4:
  echo "数值为 4";
  break;
case 5:
  echo "数值为 5";
  break;
default:
  echo "数值不在 1 到 5 之间";
}
?>
```

运行后结果如图 4-22 所示。

图 4-22　程序运行结果

4.8.2　循环控制结构

循环控制语句主要包括 3 种，即 while 循环、do…while 循环和 for 循环。while 循环在代码运行的开始检查表述的真假；而 do…while 循环则在代码运行的末尾检查表述的真假，即 do…while 循环至少要运行一遍。

1. while 循环语句

while 循环的结构为：

```
while （条件判断语句）{
    命令执行语句；
}
```

其中，当“条件判断语句”为 true 时，执行后面的“命令执行语句”，然后返回条件表达式继续进行判断，直到表达式的值为假才能跳出循环，执行后面的语句。

while 循环语句的流程控制图如图 4-23 所示。

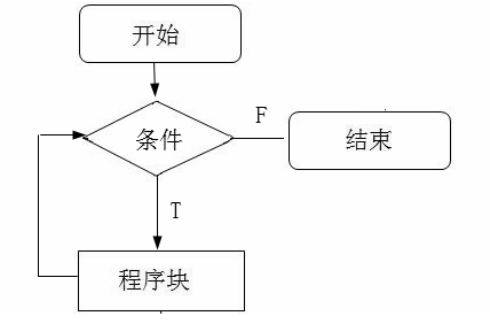

图 4-23　while 语句流程控制图

【例 4.17】（实例文件：源文件\ch04\4.17.php）

```
<?php
$num = 1;                          //定义变量$num
$str = "20 以内的奇数为：";          //定义变量$str
while($num <=20){                  //判断$num 是否小于或等于 20
    if($num % 2!= 0){              //判断$num 是否为奇数，为奇数则输出，否则做加一操作
        $str .= $num." ";
    }
    $num++;
}
echo $str;
?>
```

运行后结果如图 4-24 所示。

图 4-24　程序运行结果

本实例主要实现 20 以内的奇数输出。从 1~20 依次判断是否为奇数，如果是，则输出；如果不是，则继续下一次的循环。

2. do...while 循环语句

do…while 循环的结构为：

```
do{
   命令执行语句;
}while（条件判断语句）
```

先执行 do 后面的“命令执行语句”，其中的变量会随着命令的执行发生变化。当此变量通过 while 后的“条件判断语句”判断为 false 时，停止执行“命令执行语句”。

do…while 循环语句的流程控制图如图 4-25 所示。

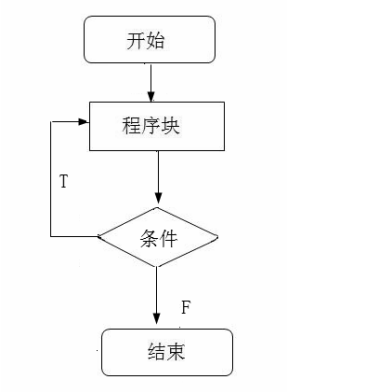

图 4-25　do…while 循环语句流程控制图

【例 4.18】（实例文件：源文件\ch04\4.18.php）

```
<?php
$aa = 0;                                    //声明一个整数变量$aa
while($aa != 0){                            //使用 while 循环输出
    echo "不会被执行的内容";                  //这句话不会被输出
}
do{                                         //使用 do…while 循环输出
    echo "被执行的内容";                      //这句话会被输出
}while($aa != 0);
?>
```

运行后结果如图 4-26 所示。从结果可以看出，while 语句和 do…while 语句有很大的区别。

图 4-26　程序运行结果

3. for 循环语句

for 循环的结构为：

```
for（expr1; expr2; expr3）
{
执行命令语句
 }
```

其中 expr1 为条件的初始值，expr2 为判断的最终值，通常都使用比较表达式或逻辑表达式充当判断的条件，执行完命令语句后，再执行 expr3。

for 循环语句的流程控制图如图 4-27 所示。

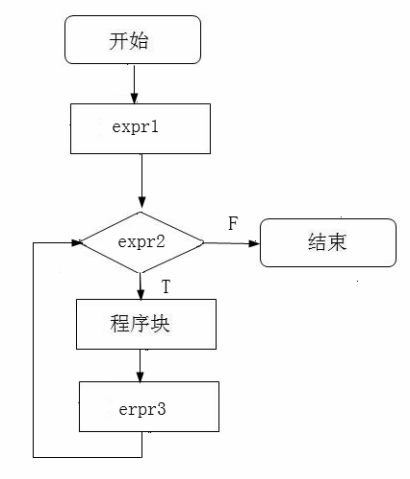

图 4-27　for 循环语句流程控制图

【例 4.19】（实例文件：源文件\ch04\4.19.php）

```
<?php
for($i=0;$i<4;$i++){                    //使用 for 循环输出
echo "for 语句的功能非常强大<br/>";
}
?>
```

运行结果如图 4-28 所示，从中可以看出命令语句执行了 4 次。

图 4-28　程序运行结果

4. foreach 循环语句

foreach 语句是常用的一种循环语句，经常被用来遍历数组元素。它的格式为：

```
foreach（数组 as 数组元素）{
      对数组元素的操作命令；
}
```

可以根据数组的情况分为两种，即不包含键值的数组和包含键值的数组。

不包含键值的：

```
foreach（数组 as 数组元素值）{
      对数组元素的操作命令；
}
```

包含键值的：

```
foreach（数组 as 键值 => 数组元素值）{
      对数组元素的操作命令；
}
```

每进行一次循环，当前数组元素的值就会被赋值给数组元素值变量，数组指针会逐一移动，直到遍历结束为止。

【例 4.20】（实例文件：源文件\ch04\4.20.php）

```
<?php
$arr=array("one", "two", "three");
foreach ($arr as $value)    //使用 foreach 循环输出
{
    echo "数组值: " . $value . "<br>";
}
?>
```

运行结果如图 4-29 所示，从中可以看出命令语句执行了 3 次。

图 4-29　程序运行结果

5. 流程控制的另一种书写格式

在一个含有多条件、多循环的语句中，包含多个“｛｝”，查看起来比较烦琐。流程控制语言的另一种书写方式是以“：”来代替左边的大括号，使用 endif；、endwhile；、endfor；、endforeach；和 endswitch；来替代右边的大括号，这种描述程序结构的可读性比较强。常见的格式如下。

条件控制语句中的 if 语句：

```
if(条件判断语句)：
     命令执行语句；
elseif(条件判断语句)：
    命令执行语句；
elseif(条件判断语句)：
    命令执行语句；
…
else：

     命令执行语句；
```

```
endif;
```

条件控制语句中的 switch 语句：

```
switch(条件判断语句):
    case  可能结果 a:
        命令执行语句;
    case  可能结果 b:
        命令执行语句;
    …
    default:
        命令执行语句;
endswitch;
```

循环控制语句中的 while 循环：

```
while(条件判断语句):
  命令执行语句
endwhile;
```

循环控制语句中的 do…while 循环：

```
do
   命令执行语句
while（条件判断语句）;
```

循环控制语句中的 for 循环：

```
for（起始表述；为真的布尔表述；增幅表述）:
命令执行语句
endfor;
```

【例 4.21】（实例文件：源文件\ch04\4.21.php）

```
<?php
    $mixnum = 1;
    $maxnum = 10;
    $tmparr[][] = array();
    $tmparr[0][0] = 1;
    for($i = 1; $i < $maxnum; $i++):
        for($j = 0; $j <= $i; $j++):
            if($j == 0 or $j == $i):
                $tmparr[$i][$j] = 1;
                   else:
                $tmparr[$i][$j] = $tmparr[$i - 1][$j - 1] + $tmparr[$i - 1][$j];
            endif;
        endfor;
    endfor;
    foreach($tmparr as $value):
        foreach($value as $vl)
            echo $vl.' ';
        echo '<p>';
    endforeach;
```

```
?>
```

运行结果如图 4-30 所示。从效果图可以看出，该代码使用新的书写格式实现了杨辉三角的排列输出。

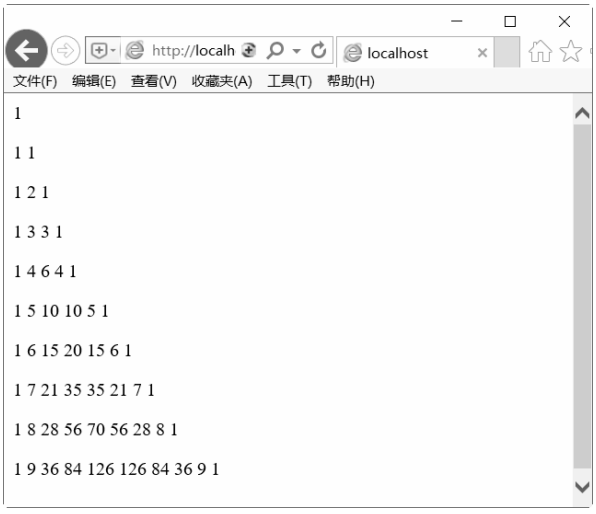

图 4-30　程序运行结果

6. 使用 break/continue 语句跳出循环

使用 break 关键字用来跳出（也就是终止）循环控制语句和条件控制语句中 switch 语句的执行，例如：

```
<?php
$n = 0;
while (++$n) {
    switch ($n) {
    case 1:
        echo "case one";
        break ;
    case 2:
        echo "case two";
        break 2;
    default:
        echo "case three";
        break 1;
    }
}
?>
```

在这段程序中，while 循环控制语句里面包含一个 switch 流程控制语句。在程序执行到 break 语句时，break 会终止执行 switch 语句，或者是 switch 和 while 语句。其中，在“case 1”下的 break 语句跳出 switch 语句；“case 2”下的 break 2 语句跳出 switch 语句和包含 switch 的 while 语句；“default”下的 break 1 语句和“case 1”下的 break 语句一样，只是跳出 switch 语句。其中，break

后带的数字参数是指 break 要跳出的控制语句结构的层数。

使用 continue 关键字的作用是跳开当前的循环迭代项，直接进入下一个循环迭代项，继续执行程序。下面通过一个实例说明此关键字的作用。

【例 4.22】（实例文件：源文件\ch04\4.22.php）

```
<?php
$n = 0;
while ($n++ < 6) {   //使用 while 循环输出
   if ($n == 2){
       continue;
   }
    echo $n."<br>";
}
?>
```

运行结果如图 4-31 所示。

图 4-31　程序运行结果

其中，continue 关键字在当 *n* 等于 2 的时候跳出本次循环，并且直接进入下一个循环迭代项，即 *n* 等于 3。另外，continue 关键字和 break 关键字一样，都可以在后面直接跟一个数字参数，用来表示跳开循环的结构层数。“continue”和“continue 1”相同，“continue 2”表示跳开所在循环和上一级循环的当前迭代项。

4.9 实战演练 1——条件分支结构综合应用

下面通过案例讲述条件分支结构的综合应用。

【例 4.23】（实例文件：源文件\ch04\4.23.php）

```
<?php
$members = Null;
function checkmembers($members){
  if ($members < 1){
    echo "我们不能为少于一人的顾客提供房间。<br>";
  }else{
    echo "欢迎来到润慧酒店。<br>";
  }
```

```
}
checkmembers(2);
checkmembers(0.5);
function checkmembersforroom($members){
  if ($members < 1){
    echo "我们不能为少于一人的顾客提供房间。<br>";
  }elseif( $members == 1 ){
    echo "欢迎来到润慧酒店。 我们将为您准备单床房。<br>";
  }elseif( $members == 2 ){
    echo "欢迎来到润慧酒店。 我们将为您准备标准间。<br>";
  }elseif( $members == 3 ){
    echo "欢迎来到润慧酒店。 我们将为您准备三床房。<br>";
  }else{
  echo "请直接电话联系我们，我们将依照具体情况为您准备合适的房间。<br>";
}
}
checkmembersforroom(1);
checkmembersforroom(2);
checkmembersforroom(3);
checkmembersforroom(5);
function switchrooms($members){
  switch ($members){
          case  1:
            echo "欢迎来到润慧酒店。 我们将为您准备单床房。<br>";
      break;
          case  2:
            echo "欢迎来到润慧酒店。 我们将为您准备标准间。<br>";
      break;
          case  3:
            echo "欢迎来到润慧酒店。 我们将为您准备三床房。<br>";
      break;
          default:
          echo "请直接电话联系我们，我们将依照具体情况为您准备合适的房间。";
      break;
        }
}
switchrooms(1);
switchrooms(2);
switchrooms(3);
switchrooms(5);
?>
```

运行结果如图 4-32 所示。

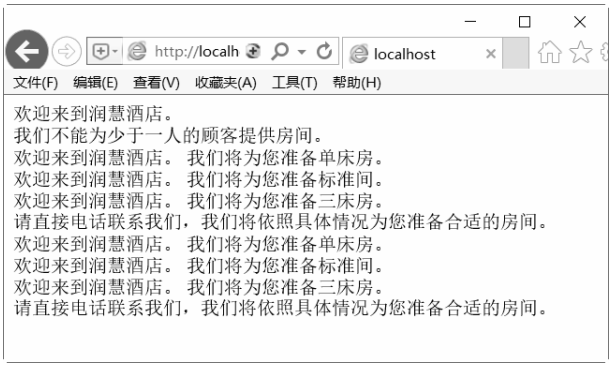

图 4-32　程序运行结果

其中，最后 4 行由 switch 语句实现，其他输出均由 if 语句实现。

4.10 实战演练 2——循环控制结构综合应用

下面以遍历已订房间门牌号为例介绍循环控制语句的应用技巧。

【例 4.24】（实例文件：源文件\ch04\4.24.php）

```
<?php
$bookedrooms = array('102','202','203','303','307');   //定义数组 bookedrooms
for ($i = 0; $i < 5; $i++){                          //循环输出数组 bookedrooms
       echo $bookedrooms[$i]."<br>";
}

function checkbookedroom_while($bookedrooms){  //定义函数
     $i = 0;
   while (isset($bookedrooms[$i])){
  echo $i.":".$bookedrooms[$i]."<br>";
  $i++;
  }
}
checkbookedroom_while($bookedrooms);
$i = 0;
do{
 echo $i."-".$bookedrooms[$i]."<br>";
 $i++;
   }while($i < 2);
?>
```

运行结果如图 4-33 所示。

图 4-33　程序运行结果

其中，102~307 由 for 循环实现。0：102~4：307 由 while 循环实现。0-102 和 1-202 由 do…while 循环实现。for 循环和 while 循环都完全遍历了数组$bookedrooms，而 do…while 循环由于条件为 while（$i＜2），因此 do 后面的命令执行了两次。

4.11 高手甜点

甜点 1：如何合理运用 include_once()和 require_once()？

答：include()和 require()函数在其他 PHP 语句执行之前运行，引入需要的语句并加以执行。但是每次运行包含此语句的 PHP 文件时，include()和 require()函数都要运行一次。include()和 require()函数如果在先前已经运行过，并且引入相同的文件，则系统会重复引入这个文件，从而产生错误。而 include_once()和 require_once()函数只是在此次运行的过程中引入特定的文件或代码，但是在引入之前，会先检查所需文件或者代码是否已经引入，如果引入将不再重复引入，从而不会造成冲突。

甜点 2：程序检查后正确，却显示 Notice: Undefined variable，为什么？

PHP 默认配置会报这个错误，就是警告将在页面上打印出来，虽然这有利于暴露问题，但是现实使用中会存在很多问题。通用解决办法是修改 php.ini 的配置，需要修改的参数如下：

（1）找到 error_reporting = E_ALL，修改为 error_reporting = E_ALL & ~E_NOTICE。
（2）找到 register_globals = Off，修改为 register_globals = On。