第 3 章

零基础开始学习——PHP 的基本语法

上一章讲述了 PHP 环境的搭建方法，本章开始学习 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 的基本语法知识和技能。

3.1 PHP 语言标识

作为嵌入式脚本语言，PHP 是以<?php ?>标识符为开始和结束标记的。当服务器解析一个 PHP 文件时，通过寻找开始标记和结束标记，告诉 PHP 开始和停止解析其中的代码。凡是标记语言以外的内容都会被 PHP 解释器忽略。但是，PHP 代码有不同的标识风格。下面学习其他类型的标识风格。

1. 脚本风格

有的编辑器对 PHP 代码完全采用另外一种表示方式，比如<script></script>的表示方式。例如：

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

这种表示方式十分类似于 HTML 页面中 JavaScript 的表示方式。

2. 简短风格

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。这种就是所谓的“短风格”(Short Style)表示法。例如：

```
<? echo "这是 PHP 短风格的表示方式。"?>
```

这种表示方式在正常情况下并不推荐。并且在 php.ini 文件中，short_open_tags 设置默认是关闭的。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。

3. ASP 风格

受 ASP 的影响，为了照顾 ASP 使用者对 PHP 的使用，PHP 提供了 ASP 标识风格。例如：

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

这种表示方式是在特殊情况下使用的，正常情况下并不推荐使用。

3.2 熟悉编码规范

由于现在的 Web 开发往往是多人一起合作完成的，所以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，往往需要知道前面开发的代码中变量或函数的作用等。如果使用统一的编码规范，这就容易多了。

3.2.1 什么是编码规范

编码规范规定了某种语言的一系列默认编程风格，用来增强这种语言的可读性、规范性和可维护性。编码规范主要包括语言下的文件组织、缩进、注释、声明、空格处理、命名规则等。

遵守 PHP 编码规范有以下好处。

(1) 编码规范是团队开发中对每个成员的基本要求。对编码规范遵循得好坏是一个程序员成熟程度的表现。

(2) 能够提高程序的可读性，利于开发人员互相交流。

(3) 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。

(4) 有助于程序的维护，可以降低软件成本。

3.2.2 PHP 的一些编码规范

PHP 作为高级语言的一种，特别强调编码规范。具体表现在以下几个方面。

1. 表述

比如在 PHP 的正常表述中，每一条 PHP 语句都是以“;”结尾，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. 注释

在 PHP 语言中，常见的注释包括以下几种风格。

(1) C 语言风格。例如：

```
/*这是 C 语言风格的注释内容*/
```

这种方法还可以多行使用。例如：

```
/*这是
  C 语言风格
  的注释内容
*/
```

注意：注释不能嵌套，因为 PHP 不进行块注释的嵌套检查，所以以下写法是错误的：

```
/*这是
echo "这里开始嵌套注释";/*嵌套注释时 PHP 会报错*/
*/
```

(2) C++风格。例如：

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

这种方法只能一句注释占用一行。使用时可单独一行，也可以使用在 PHP 语句之后的同一行。

(3) Shell 风格。例如：

```
#这是 Shell 风格的注释内容
```

这种方法只能一句注释占用一行。使用时可单独一行，也可以使用在 PHP 语句之后的同一行。

3. 空白

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 对它们都进行忽略。这跟浏览器对 HTML 语言中的空白的处理是一样的。

合理地运用空白符，可以增强代码的清晰性和可读性。

(1) 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。

(2) 下列情况应该总是使用一个空白行。

① 两个函数声明之前。

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。

(3) 合理利用空格可以通过代码的缩进提高可读性。

① 空格通常使用于关键字与括号之间，但是，函数名称与左括号之间不使用空格分开。

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. 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 代码段中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
```

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。可以使 PHP 嵌入到 HTML 文档中去。例如：

```
<HTML>
<HEAD>
<TITLE>PHP 与 HTML 混合搭配</TITLE>
</HEAD>
<BODY>
```

```
<?php
     echo "嵌入的 PHP 代码";
?>
</BODY>
</HTML>
```

3.3 常　　量

常量和变量是构成 PHP 程序的基础。下面讲述如何声明和使用常量。

3.3.1 声明和使用常量

在 PHP 中，常量是一旦声明就无法改变的值。PHP 通过 define()命令来声明常量。其语法格式如下：

```
define("常量名", 常量值,mixed value);
```

常量名是一个字符串，往往在 PHP 编码规范的指导下使用大写的英文字符来表示。例如 CLASS_NAME、MYAGE 等。常量值也可为表达式。mixed value 是可选参数，表示常量的名字是否区分大小写，如果设置为 true，则表示不区分大小写，默认为 false。

常量值可以是很多种 PHP 数据类型的，可以是数组，也可以是对象，当然还可以是字符和数字。

常量就像变量一样储存数值。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，它都不能被改动。

常量声明后具有全局性，函数内外都可以访问。

【例 3.1】声明和使用常量(示例文件 ch03\3.1.php)。

```
<?php
define("SHIGE","日出江花红胜火，春来江水绿如蓝。");//定义常量 SHIGE
echo SHIGE; //输出常量 SHIGE
?>
```

程序运行结果如图 3-1 所示。

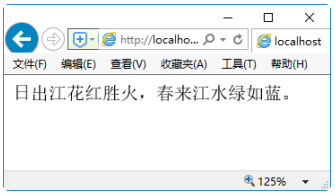

图 3-1　声明和使用常量

【案例剖析】

(1) 用 define 函数声明一个常量 SHIGE。而常量的全局性体现在可在函数内外进行访问。

(2) 常量只能储存布尔值、整型、浮点型和字符串数据。

3.3.2 使用系统预定义常量

PHP 的系统预定义常量是指 PHP 在语言内部预先定义好的一些常量。PHP 中预定了很多系统内置常量，这些常量可以被随时调用。例如，下面是一些常见的内置常量。

1. _FILE_

这个默认常量是 PHP 程序文件名。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数，而不是引用它的文件的行数。

3. PHP_VERSION

这个内置常量是 PHP 程序的版本，如 3.0.8-dev。

4. PHP_OS

这个内置常量指执行 PHP 解析器的操作系统名称，如 Linux。

5. TRUE

这个常量就是真值(true)。

6. FALSE

这个常量就是伪值(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

该常量为解析语法有潜在问题处。

10. E_NOTICE

这个量为发生异常(但不一定是错误)处。例如存取一个不存在的变量。

下面举例说明系统常量的使用方法。

【例 3.2】使用系统内置常量(示例文件 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
?>
```

程序运行结果如图 3-2 所示。

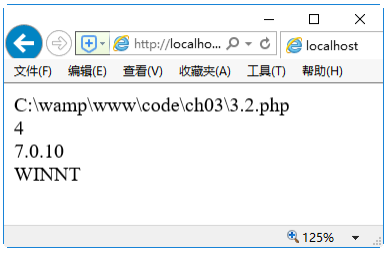

图 3-2 使用内置常量

【案例剖析】

(1) echo "<p>"语句表示为输出换行。

(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

变量像是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一样。

3.4.1 PHP 中的变量声明

与 C 语言或 Java 语言不同的是，PHP 中的变量是弱类型的。在 C 语言或 Java 语言中，需要对每一个变量声明类型，而在 PHP 中不需要这样做。这是极其方便的。

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

合法的变量名可以是：

```
$hello
$Aform1
$_formhandler
```

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，这是一个好的程序员应该养成的习惯。PHP 的赋值有两种，包括传值和引用，它们的区别如下。

(1) 传值赋值：使用“=”直接将赋值表达式的值赋给另一个变量。

(2) 引用赋值：将赋值表达式内存的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外一个变量的变化。

【例 3.3】使用传值赋值和引用赋值(示例文件 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 = "稻云不雨不多黄";
$b = $a;                    // 将变量$a 的值赋值给$b，两个变量指向不同内存空间
echo "变量 a 的值为".$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
$a = "荞麦空花早着霜";          // 改变变量 a 的值，变量 b 的值不受影响
echo "变量 a 的值为".$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 = "已分忍饥度残岁";
$b = &$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
$a = "更堪岁里闰添长";
/*
改变变量 a 在内存空间中存储的内容，变量 b 也指向该空间，b 的值也发生变化
*/
echo "变量 a 的值为".$a."<br/>";       // 输出变量 a 的值
echo "变量 b 的值为".$b." <br/>";      // 输出变量 b 的值
?>
```

程序运行结果如图 3-3 所示。

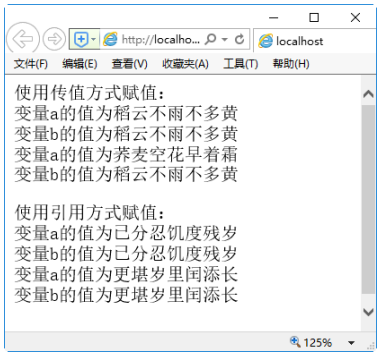

图 3-3　程序运行结果

3.4.2　可变变量和变量的引用

一般的变量很容易理解，但是有两种变量的概念比较难以理解，这就是可变变量和变量的引用。下面通过示例对它们进行介绍。

【例 3.4】使用可变变量和变量的引用(示例文件 ch03\3.4.php)。

```
<?php
$value0 = "guest";      //定义变量$value0 并赋值
$$value0 = "customer";  // 再次给变量赋值
echo $guest."<br/>";    // 输出变量
$guest = "张飞";        // 定义变量$guest 并赋值
echo $guest."\t".$$value0."<br/>";
$value1 = "王小明";     // 定义变量$value1
$value2 = &$value1;     // 引用变量并传递变量
echo $value1."\t".$value2."<br/>";
$value2 = "李丽";
echo $value1."\t".$value2;
?>
```

程序运行结果如图 3-4 所示。

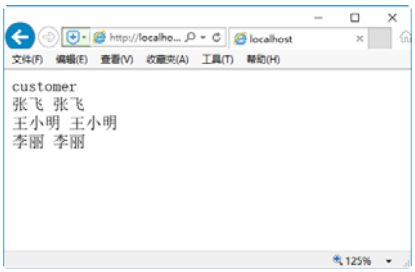

图 3-4　使用可变变量和变量的引用

【案例剖析】

(1)　在代码的第一部分中，$value0 被赋值为“guest”。而$value0 相当于 guest，则$$value0 相当于$guest。所以当$$value0 被赋值为“customer”时，打印$guest 就得到 customer。反之，当$guest 变量被赋值为“张飞”时，打印$$value0 同样得到“张飞”。这就是可变变量。

(2)　在代码的第二部分中，$value1 被赋值为“王小明”，然后通过&引用变量$value1 并赋值给$value2。而这一步的实质是，给变量$value1 添加了一个别名$value2。所以打印时，都得出原始赋值“王小明”。由于$value2 是别名，与$value1 指的是同一个变量，所以$value2

被赋值为“李丽”后，$value1 和$value2 都得到新值“李丽”。

(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外一个变量的名。

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

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：在代码中的任意位置都可以访问到。

(2) 常数：一旦声明，它就是全局性的。可以在函数内外使用。

(3) 全局变量：在代码中声明，可在代码中访问，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是可以保留的。

(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终止并退出。

1. 超全局变量

超全局变量的英文是 Superglobal 或者 Autoglobal(自动全局变量)。这种变量的特性是：不管在程序的任何地方，也不管是函数内或是函数外，都可以访问。而这些“超全局变量”就是由 PHP 预先定义好以方便使用的。

这些超全局变量或自动全局变量介绍如下。

- $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，在代码中都可以访问。但是在函数内是不能访问的。这是因为函数默认就不能访问在其外部的全局变量。

下面通过例子介绍全局变量的使用方法和技巧。

【例 3.5】使用全局变量(示例文件 ch03\3.5.php)。

```
<?php
$price = 3;     //定义全局变量
function showprice(){
   echo $price; //函数内调用全局变量
}
showprice();    //运行函数
echo '苹果的价格为'.$price.'元一公斤';
?>
```

程序运行结果如图 3-5 所示。

图 3-5　使用全局变量

【案例剖析】

出现上述结果，是因为函数无法访问到外部全局变量，但是在代码中都可以访问全局变量。

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。就是说，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。

下面通过例子介绍使用 global 关键字的方法和技巧。

【例 3.6】使用 global 关键字(示例文件 ch03\3.6.php)。

```
<?php
$price = 3;    //定义全局变量
function showprice(){
   global price;    //函数内声明全局变量
}
showprice();
echo '苹果的价格为'$price.'元一公斤';
?>
```

程序运行结果如图 3-6 所示。

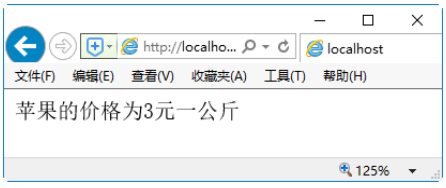

图 3-6　使用 global 关键字

另外，读者还可以通过超全局变量中的$GLOBALS 数组进行访问。

下面通过例子介绍如何使用$GLOBALS 数组。

【例 3.7】使用$GLOBALS 数组(示例文件 ch03\3.7.php)。

```
<?php
$aa = '九曲黄河万里沙';            // 定义全局变量
function showss(){
    $aa = $GLOBALS['aa'];        // 通过$GLOBALS 数组访问全局变量
    echo $aa.'<br/>';
```

```
}
showss();
echo $aa.',浪淘风簸自天涯';
?>
```

程序运行结果如图 3-7 所示。

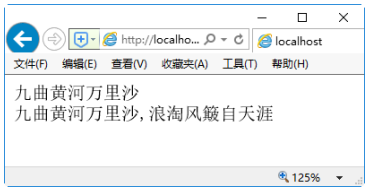

图 3-7 使用$GLOBALS 数组

3. 静态变量

静态变量只在函数内存在，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，这个静态变量的值保留，下一次再执行此函数，这个值还可以调用。

下面通过例子介绍静态变量的使用方法和技巧。

【例 3.8】使用静态变量(示例文件 ch03\3.8.php)。

```
<?php
$aa = 15;
function sumaa(){
static $aa = 6;   //初始化静态变量
$aa++;
echo '变量 aa 的值为：'.$aa.'<br/>';
}
sumaa();
echo $aa.'<br/>';
sumaa();
?>
```

程序运行结果如图 3-8 所示。

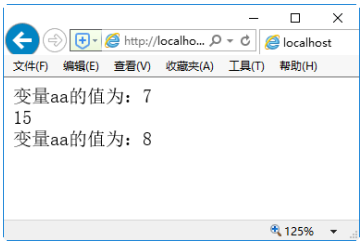

图 3-8 使用静态变量

【案例剖析】

(1) 其中函数外的 echo 语句无法调用函数内的 static $aa，它调用的是$aa = 15。

(2) 另外，sumaa()函数被执行了两次，这个过程中，static $aa 的运算值得以保留，并且通过$aa++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，则空间引用计数减 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，如果变量类型为局部变量，则变量被销毁；如果变量类型为全局变量，则变量不会被销毁。

【例 3.9】使用 unset()(示例文件 ch03\3.9.php)。

```
<?php
function showshi(){
    //声明函数
    global $aa;
    //函数内使用 global 关键字声明全局变量$aa
    unset ($aa);
    //使用 unset()销毁不再使用的变量$aa
}
$aa= '如今直上银河去，同到牵牛织女家。';
    //函数外声明全局变量
showshi();                  //调用函数
echo  $aa;                  //查看全局变量是否发生变化
?>
```

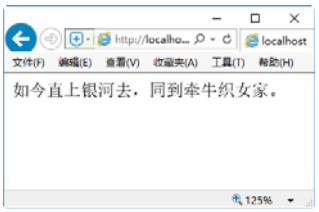

图 3-9　使用 unset()

程序运行结果如图 3-9 所示。

3.5　理解变量的类型

从 PHP 4 开始，PHP 中的变量不需要事先声明，赋值即声明。声明和使用这些数据类型前，读者需要了解它们的含义和特性。

3.5.1　什么是类型

不同的数据类型其实就是所储存数据的不同种类。PHP 主要有下列数据类型。

(1) 整型(integer)：用来储存整数。
(2) 浮点型(float 或 double)：用来储存实数。
(3) 字符串型(string)：用来储存字符串。
(4) 布尔型(boolean)：用来储存真(true)或假(false)。
(5) 数组型(array)：用来储存一组数据。
(6) 对象型(object)：用来储存一个类的实例。

作为弱类型语言，PHP 也被称为动态类型语言。在强类型语言(如 C 语言)中，一个变量只能储存一种类型的数据，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的。例如以下几个变量：

```
$hello = 'hello world';
//由于'hello world'是字符串，则变量$hello 的数据类型就为字符串类型
$hello = 100;
//由于 100 为整型，所以$hello 也就为整型
$wholeprice = 100.0;
//由于 100.0 为浮点型，所以$wholeprice 就是浮点型
```

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

3.5.2　整型

整型是数据类型中最为基本的类型。在 32 位的运算中，整型的取值范围是−2147483648 ~ +2147483647。整型可以表示为十进制、十六进制和八进制数形式。

例如：

```
3560        //十进制整数
01223       //八进制整数
0x1223      //十六进制整数
```

3.5.3 浮点型

浮点型就是实数。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。

而对于其他数据类型，基本都有布尔属性。

(1) 整型：为 0 时，其布尔属性为 false，为非零值时，其布尔属性为 true。

(2) 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。

(3) 字符串型：空字符串“”，或者零字符串“0”时，为 false，包含除此以外的字符串时为 true。

(4) 数组型：若不含任何元素，为 false，只要包含元素，则为 true。

(5) 对象型、资源型：永远为 true。

(6) 空型：永远为 false。

3.5.5 字符串型

字符串型的数据是引号之间的一串字符。引号有双引号""和单引号"两种。

但是这两种表示也有一定的区别。

双引号几乎可以包含所有的字符。但是其中显示的是变量的值，而不是变量的名。而有些特殊字符在使用时需要加上“\”这一转义符号。

单引号内的字符是被直接表示出来的。

下面通过一个例子来讲述上面几种类型的使用方法和技巧。

【例 3.10】使用各种数据类型(示例文件 ch03\3.10.php)。

```
<?php
$int1 = 2018;
$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;
?>
```

程序运行结果如图 3-10 所示。

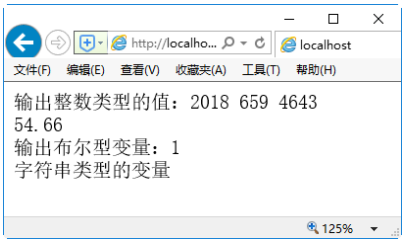

图 3-10　使用各种数据类型

3.5.6　数组型

数组是 PHP 变量的集合，它是按照“键”与“值”对应的关系组织数据的。数组的键可以是整数，也可以是字符串。

在默认情况下，数组元素的键为从 0 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】使用 array()函数创建数组(示例文件 ch03\3.11.php)。

```
<?php
$arr=array                              // 定义数组并赋值
(
0=>"墨梅",
2=>"元代：王冕",
1=>"吾家洗砚池头树，个个花开淡墨痕。",
3=>"不要人夸好颜色，只留清气满乾坤。"
);
for ($i=0; $i<count($arr); $i++)   // 使用 for 循环输出数组的内容
{
$arr1 = each($arr);
echo "$arr1[value]<br>";
}
?>
```

程序运行结果如图 3-11 所示。

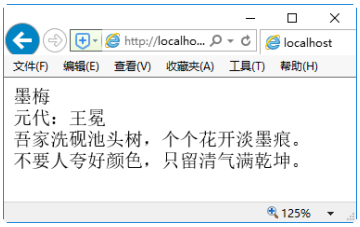

图 3-11　使用 array()函数创建数组

【案例剖析】

(1)　程序中用=>为数组赋值，数组的下标只是存储的标识，没有任何其他意义。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，each 函数返回当前数组指针的索引/值对，后面章节中将详细讲述函数的使用方法。

例 3.11 中的语句可以简化如下。

【例 3.12】使用 array()函数的简化语句(示例文件 ch03\3.12.php)。

```
<?php
$arr = array("墨梅","元代：王冕","吾家洗砚池头树，个个花开淡墨痕。","不要人夸好颜色，只留清气满乾坤。");
for ($i=0; $i<4; $i++)
{
echo $arr[$i]."<br>";
}
?>
```

程序运行结果如图 3-12 所示。

从结果可以看出，两种写法的运行结果完全一样。

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。上面的程序中的语句可以写成如下形式。

【例 3.13】通过逐个赋值的方式创建数组(示例文件 ch03\3.13.php)。

```
<?php
$arr[0] ="墨梅";                              // 对数组元素分别赋值
$arr[2] = "元代：王冕";
$arr[1] = "吾家洗砚池头树，个个花开淡墨痕。";
$arr[3] = "不要人夸好颜色，只留清气满乾坤。";
for ($i=0; $i<count($arr); $i++)   // 使用 for 循环输出数组的内容
{
$arr1 = each($arr);
echo "$arr1[value]<br>";
}
?>
```

程序运行结果如图 3-13 所示。

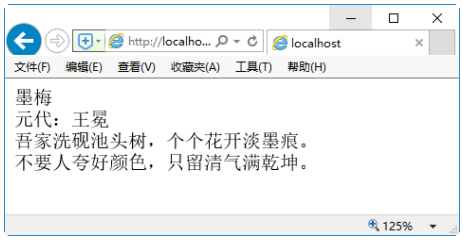

图 3-12 简化后的程序运行结果

图 3-13 逐个赋值时的程序运行结果

从结果可以看出，一个一个赋值的方法与上面两种写法的运行结果是一样的。

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 类型用来标记一个变量为空。但一个空字符串与一个 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

资源类型(Resources)是十分特殊的数据类型，表示了 PHP 的扩展资源。它可以是一个打开的文件，也可以是一个数据库连接，甚至还可以是其他数据类型。但是在编程过程中，资源类型却是很难接触到的。

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外一个类型，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。

例如，float 型转换为整数 int 型，小数点后面的数将被舍弃。如果 float 数超过了整数的取值范围，则结果可能是 0 或者整数的最小负数。

【例 3.14】自动数据类型转换(示例文件 ch03\3.14.php)。

```
<?php
$floaa=8.88;                  // 定义 float 类型
echo (int)$floaa."<br/>";     // 转换为整数类型输出
$flobb=4E32;                  // 超过整数取值范围
echo(int)$flobb;
?>
```

程序运行结果如图 3-14 所示。

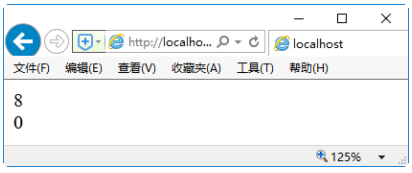

图 3-14 自动数据类型转换

2. 强制数据类型转换

在 PHP 中，可以使用 setType()函数强制转换数据类型。基本语法格式如下：

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

type 的可能值不能包含资源类型数据。

【例 3.15】强制数据类型转换(示例文件 ch03\3.15.php)。

```
<?php
$floaa = 6.66;
echo setType($floaa, "int")"<br/>";
echo $floaa;
?>
```

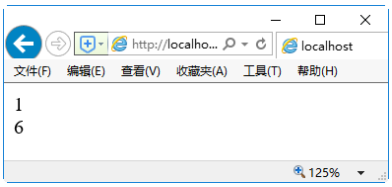

图 3-15 强制数据类型转换

程序运行结果如图 3-15 所示。转型成功，则返回 1，否则返回 0。变量 floaa 转换为整型后为 6。

3.6 PHP 7 的新变化——声明标量类型和函数返回值类型

在默认情况下，所有的 PHP 文件都处于弱类型校验模式。PHP 7 增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);
```

可以设置 strict_types 的值(1 或者 0)，1 表示严格类型校验模式，作用于函数调用和返回语句；0 表示弱类型校验模式。

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

下面通过案例来理解强制模式的含义。代码如下：

```
<?php
// 强制模式
function sum(int $ints)
{
   return array_sum($ints);
}
print(sum(2, '3', 8.18));
?>
```

上述程序输出结果为 13。代码中的'3'先转换为 3，8.18 先转换为整数 8，然后再进行相加操作。

2. 严格模式

下面通过案例来理解严格模式的含义。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));
?>
```

上述程序由于采用了严格模式，所以如果参数中出现不是整数的类型，程序执行时会报错，如图 3-16 所示。

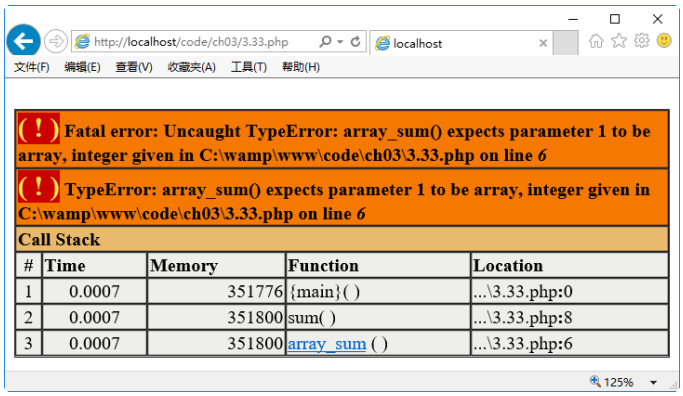

图 3-16　错误提示信息

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。代码如下：

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

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

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

程序执行结果如图 3-17 所示。

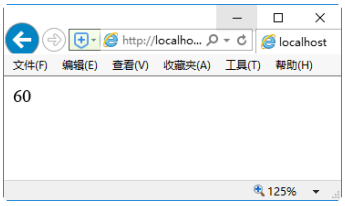

图 3-17　声明函数返回值的类型

3.7　使用运算符

PHP 包含 3 种类型的运算符。一元运算符、二元运算符和三元运算符。一元运算符用在一个操作数之前；二元运算符用在两个操作数之间；三元运算符用在三个操作数之间。

3.7.1　算术运算符

算术运算符是最简单也是最常用的运算符。常见的算术运算符如表 3-1 所示。

表 3-1 算术运算符

运 算 符	名 称
+	加法运算
-	减法运算
*	乘法运算
/	除法运算
%	取余运算
++	累加运算
--	累减运算

【例 3.16】使用算术运算符(示例文件 ch03\3.16.php)。

```
<?php
$a=13;
$b=2;
echo $a."+".$b."=";
echo $a+$b."<br>";
echo $a."-".$b."=";
echo $a-$b."<br>";
echo $a."*".$b."=";
echo $a*$b."<br>";
echo $a."/".$b."=";
echo $a/$b."<br>";
echo $a."%".$b."=";
echo $a%$b."<br>";
echo $a."++"."=";
echo $a++."<br>";
echo $a."--"."=";
echo $a--."<br>";
?>
```

程序运行结果如图 3-18 所示。

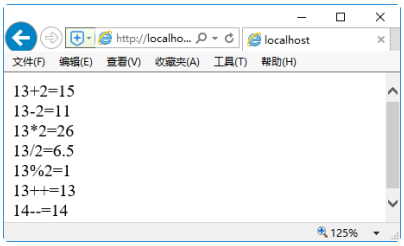

图 3-18 使用算术运算符

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++，结果将等于 c。

3.7.2 字符串连接符

字符运算符“.”把两个字符串连接起来，变成一个字符串。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。

【例 3.17】使用字符串连接符(示例文件 ch03\3.17.php)

```
<?php
$a = "等闲识得东风面，万紫千红总是春。";
$b = 10;
echo "我读了".$b."遍：".$a
?>
```

程序运行结果如图 3-19 所示。

图 3-19　使用字符串连接符

3.7.3　赋值运算符

赋值运算符的作用是把一定的数值加载给特定的变量。

赋值运算符的具体含义如表 3-2 所示。

表 3-2　赋值运算符

运 算 符	名　称
=	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。

3.7.4　比较运算符

比较运算符用来比较其两端数值的大小。比较运算符的具体含义如表 3-3 所示。

表 3-3　比较运算符

运 算 符	名　称
==	相等
!=	不相等
>	大于
<	小于
>=	大于等于

续表

运 算 符	名 称
<=	小于等于
===	精确等于(类型)
!==	不精确等于

其中，===和!==需要特别注意。$b===$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】使用比较运算符(示例文件 ch03\3.18.php)。

```
<?PHP
$value="15";
echo "\$value = \"$value\"";
echo "<br>\$value==15: ";
var_dump($value==15);           //结果为:bool(true)
echo "<br>\$value==true: ";
var_dump($value==true);         //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(true)
echo "<br>(10/2.0 !== 5): ";
var_dump(10/2.0 !==5);          //结果为:bool(true)
?>
```

程序运行结果如图 3-20 所示。

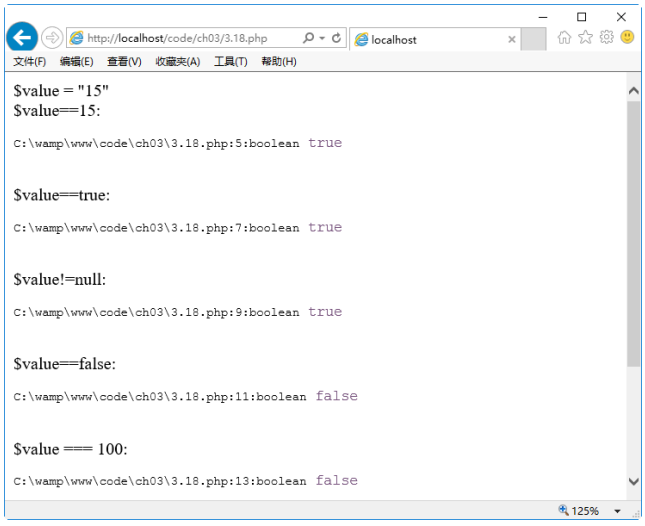

图 3-20　使用比较运算符

3.7.5 逻辑运算符

一个编程语言最重要的功能之一就是进行逻辑判断和运算。比如逻辑和、逻辑或、逻辑非。逻辑运算符的含义如表 3-4 所示。

表 3-4 逻辑运算符

运 算 符	名 称
&&、AND	逻辑和
‖、OR	逻辑或
!、NOT	逻辑非
XOR	逻辑异或

3.7.6 按位运算符

按位运算符是把整数以“位”为单位进行处理。按位运算符的含义如表 3-5 所示。

表 3-5 按位运算符

运 算 符	名 称
&	按位和
|	按位或
^	按位异或

3.7.7 否定控制运算符

否定控制运算符用在操作数之前，用于对操作数真假的判断。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

运 算 符	名 称
!	逻辑非
~	按位非

3.7.8 错误控制运算符

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

3.7.9 三元运算符

三元运算符“?:”是作用在三个操作数之间的。其语法格式如下：

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

如果表达式 expr1 为真，则返回 expr2 的值；如果表达式 expr1 为假，则返回 expr3。从 PHP 5.3 开始，可以省略 expr2，表达式为(expr1) ?: (expr3)，如果表达式 expr1 为真，则返回 expr1 的值；如果表达式 expr1 为假，则返回 expr3。例如以下代码：

```
<?php
$aa = '春花秋月何时了，往事知多少';
$bb = $aa ?: '没有古诗内容';
$cc = $aa ? '这里输出古诗的内容': '没有古诗内容';
echo $bb;
echo $cc;
?>
```

程序运行结果如图 3-21 所示。

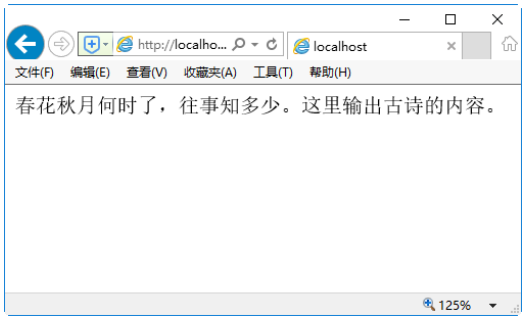

图 3-21 使用三元运算符

3.7.10 运算符的优先级和结合规则

运算符的优先级和结合规则其实与正常的数学运算符的规则十分相似。

(1) 加减乘除的先后顺序与数学运算中的完全一致。

(2) 对于括号，则先运行括号内，再运行括号外。

(3) 对于赋值，则由右向左运行，也就是依次从右边向左边的变量进行赋值。

3.8 PHP 7 的新变化——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，则返回 expr1 的值；如果表达式 expr1 为假，则返回 expr2。例如以下代码：

```
<?php
$aa = '众鸟高飞尽，孤云独去闲。';
$bb = $aa ?? '没有古诗内容';
echo $bb;
?>
```

程序运行结果如图 3-22 所示。

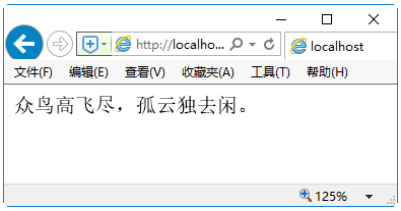

图 3-22　使用合并运算符

PHP 7 新增加的组合运算符，用于比较两个表达式 $a 和 $b，如果 $a 小于、等于或大于 $b 时，它分别返回-1、0 或 1。例如以下代码：

```
<?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>";
?>
```

程序运行结果如图 3-23 所示。

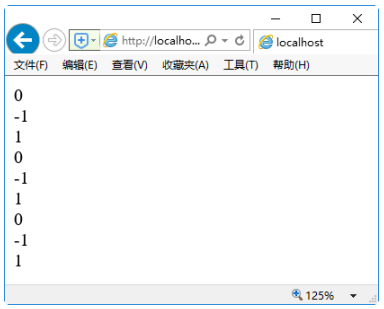

图 3-23　使用组合运算符

3.9　PHP 中的表达式

表达式是表达一个特定操作或动作的语句。表达式由操作数和操作符组成。

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，如表达式$a=1 的值为 1。

在 PHP 代码中，使用“;”号来区分表达式和语句，即一个表达式和一个分号组成一条 PHP 语句。在编写代码时，应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。

3.10 创建多维数组

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

下面给出创建二维数组的例子。

【例 3.19】创建二维数组(示例文件 ch03\3.19.php)。

```
<?php
$arr[0][0] = "月黑雁飞高";
$arr[0][1] = "单于夜遁逃";
$arr[1][0] = "欲将轻骑逐";
$arr[1][1] = "大雪满弓刀";
for ($i=0; $i<count($arr); $i++)
{
for ($k=0; $k<count($arr[$i]); $k++)
{
$arr1 = each($arr[$i]);
echo "$arr1[value]<br>";
}
}
?>
```

程序运行结果如图 3-24 所示。

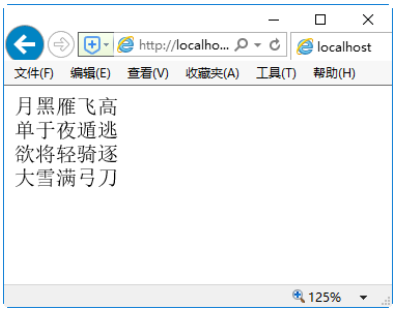

图 3-24 创建二维数组

3.11 疑 难 解 惑

疑问 1：如何灵活运用命名空间(namespace)?

答：命名空间(namespaces)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手段。比如，文件系统中不同文件夹路径中的两个文件的文件名可以完全相同，但由于是在不同的文件夹中，所以是两个完全不同的文件。

PHP 的命名空间也是这样一个概念。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。

命名空间使用 namespace 关键字在文件头部定义。例如：

```
<?php
namespace 2ndbuilding\number24;
class room{}
$room = new _NAMESPACE_.room;
?>
```

命名空间还可以拥有子空间，它们组合起来，就像文件夹的路径一样。对于命名空间的使用，可以通过内置变量__NAMESPACE__来使用命名空间及其子空间。

疑问 2：如何快速区分常量和变量？

答：常量和变量的明显区别如下。

(1) 常量前面没有美元符号($)。

(2) 常量只能用 define()函数定义，而不能通过赋值语句定义。

(3) 常量可以不用理会变量范围的规则，可以在任何地方定义和访问。

(4) 常量一旦定义就不能被重新定义或者取消定义。

(5) 常量的值只能是标量。

疑问 3：PHP 中常见的输出方式有几种？

答：在 PHP 中，常见的输出语句如下。

(1) echo 语句：可以一次输出多个值，多个值之间用逗号分隔。

(2) print 语句：只允许输出一个字符串。

(3) print_r()函数：可以把字符串和数字简单地打印出来，而数组则以括起来的键和值的列表形式显示，并以 Array 开头。但 print_r()输出布尔值和 NULL 的结果没有意义，因为都是打印"\n"。因此，用 var_dump()函数更适合调试。

var_dump()函数：判断一个变量的类型与长度，并输出变量的数值，如果变量有值输出的是变量的值并回返数据类型。此函数显示关于一个或多个表达式的结构信息，包括表达式的类型与值。