第 3 章　语 言 结 构

上一章我们学习了 PHP 的数据类型和常用的运算符，这些知识在本节中将作为语言的基本组成部分。本章将要讲解的内容是语言结构的知识。语言结构是一个程序的整体框架，它用来主导程序的走向，是程序的核心。

3.1　语　　句

语句是组成一个程序的最基本组成部分。经过各种语句的协调工作，即可构成一个功能完整的程序。

3.1.1　表达式

表达式是指由常量和变量等通过运算符连接起来而形成的一个有意义的算式。在前面的章节中我们已经多次使用过，如下即为一些表达式：

```
$x=6
$y+5-$z
$x>$y?$x:$y
```

3.1.2　表达式语句

在一个有效表达式的结尾加入一个分号即为一个表达式语句。如下所示即为表达式语句：

```
$x=6;            //赋值语句
$x++;            //递增语句
$x>$y;           //判断语句
```

表达式语句和表达式的区别在于：表达式代表的是一个数值，而表达式语句代表的是一种动作特征。PHP 程序中最常见的表达式语句为赋值语句。

3.1.3　复合语句和空语句

复合语句是使用花括弧将多条语句组合而成的一种语句格式，也称为语句块。复合语句从形式上看是多个语句的组合，但在语法意义上是一个整体，被看作一条语句。所以只要是可以使用简单语句的地方都可以使用复合语句，如下所示即为复合语句：

```
{                    //左花括弧表示复合语句开始
    $x='Hello';
    $y='world';      //符合语句中的多条语句
```

```
    $z=$x+$y;
    echo $z;
}                          //右花括弧表示复合语句结束
```

空语句即为不执行任何操作的语句，它的形式如下：

```
;                          //空语句只有一个分号
```

空语句常用于在某些场合占据一个语句的位置，例如 for 循环之中。

3.1.4　语句的执行顺序

在没有控制结构的语句中，程序是由程序开头逐句执行直到没有语句为止，但是这种结构有些功能是完成不了的，例如根据不同的状态输出不同的信息的操作。要完成这类操作，就需要用到一些语言结构，这就是我们接下来要学习的知识。

3.2　选 择 语 句

选择语句用于使程序在不同的条件下执行不同的语句。PHP 中的选择语句有 if 语句和 switch 语句，下面就来介绍他们。

3.2.1　if 语句

if 语句也称为条件语句，它有多种使用形式，包括 if 形式、if…else 形式和 if…elseif…else 形式，下面分别介绍这些形式的使用。

1. if 形式

if 形式是 if 语句最基本的形式，它的语法结构如下：

```
if (表达式)
    语句 1;
```

如果表达式的值为 TRUE，则执行语句 1，如果为 FALSE 则执行语句 1 之后的语句，这里的语句 1 可以是一个语句，也可以是一个复合语句。如果是复合语句，必须带有花括号。该结构的流程如图 3.1 所示。

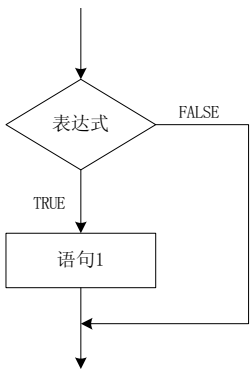

图 3.1　if 形式流程图

【示例 3-1】以下代码演示 if 选择语句的 if 形式用法。

```
01  <?php
02      $x=6;                                   //初始化两个变量
03      $y=7;
04      if($y>$x)                               //比较运算结果为 TRUE
05          $x=$y;                              //该语句将$y 的值赋值给$x
06      echo "两个变量中比较大的值是$x 。";      //输出两个变量中比较大的值
07  ?>
```

代码运行结果如图 3.2 所示。

以上代码的作用就是输出两个变量中比较大的值，如果变量 y 的值小于变量 x，则赋

值语句不会被执行。通过 if 语句来保证$x 值永远为最大的。

2. if…else 形式

if…else 形式的语法结构如下：

```
if (表达式)
    语句 1;
else
    语句 2;
```

如果表达式的值为 TRUE，则执行语句 1，为 FALSE 则执行语句 2，该结构的流程如图 3.3 所示。

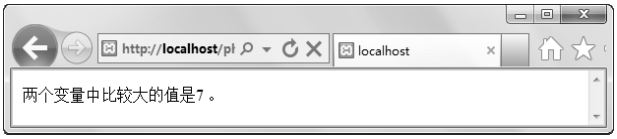

图 3.2　运行结果

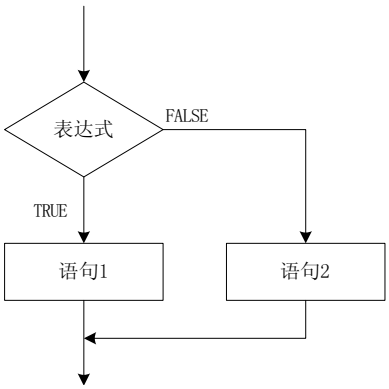

图 3.3　if…else 形式流程图

【示例 3-2】以下代码演示 if 选择语句的 if…else 形式用法。

```
<?php
    $x=16;                  //初始化两个变量
    $y=17;
    if($x>$y)               //进行比较运算并根据比较结果执行对应的语句
        echo "两个变量中比较大的值是$x";
    else
        echo "两个变量中比较大的值是$y";
?>
```

代码运行结果如图 3.4 所示。

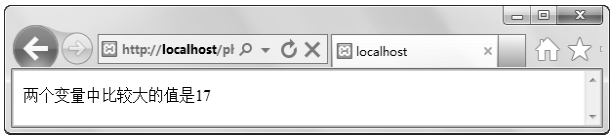

图 3.4　运行结果

由于变量 y 的值要大于变量 x 的值，因此 else 下的语句被执行。

3. if…elseif…else

if…elseif…else 的语法结构如下：

```
if (表达式 1)
```

```
    语句 1;
elseif(表达式 2)
    语句 2;
else
    语句 3;
```

注意：以上语法结构中的 elseif 项可以有若干个，这里只列出了最基本的形式。该语法结构中的 else 从句是可选的。

这种结构从上到下逐个对条件进行判断，一旦条件满足或者遇到 else 从句就执行与相关的语句，并跳过结构中的其他代码。该结构的流程图如图 3.5 所示。

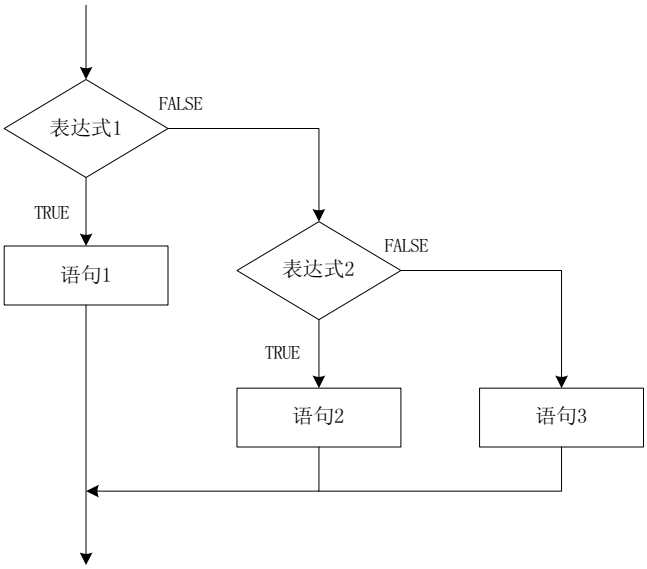

图 3.5　if…elseif…if 形式流程图

【示例 3-3】以下代码演示 if 选择语句的 if…elseif…else 形式的用法。

```
01  <?php
02      $score=90;                              //定义一个分数变量并初始化
03      if($score>=90&&$score<=100)             //对分数进行判断并输出对应评价
04          echo '这是一个优秀的成绩。';
05      elseif($score>=80&&$score<90)           //对分数进行判断并输出对应评价
06          echo '这是一个良好的成绩。';
07      elseif($score>=60&&$score<80)           //对分数进行判断并输出对应评价
08          echo '这个成绩需要努力。';
09      elseif($score<60&&$score>=0)            //对分数进行判断并输出对应评价
10          echo '这个成绩非常糟糕! ';
11      else                                  //不满足以上任何一个条件则输出合法性提示
12          echo '请确定成绩的合法性。';
13  ?>
```

当以上代码中$score=90 时，由于分数在 90～100 之间，因此第 4 行代码会被执行，输出结果如图 3.6 所示。

当$score=59 时，由于分数在 0～59 之间，因此第 10 行代码会被执行，输出结果如图 3.7 所示。

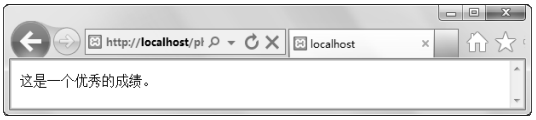

图 3.6　运行结果

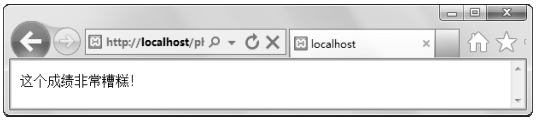

图 3.7　运行结果

当$score 为一个大于 100 或者小于 0 的数值时，default 后的语句会被执行，输出结果如图 3.8 所示。

4. if 语句嵌套

选择语句可以嵌套，也就是在一个选择结构中存在另一个选择结构，这是经常碰到的情况，但也是容易出错的地方，原因常出现在 if 和 else 的匹配问题。PHP 中的 else 总是会与最近的 if 匹配。我们首先来看一个使用正确嵌套的示例。

【示例 3-4】以下代码演示正确使用 if 语句嵌套。

```
<?php
    $operator='/';                  //定义一个变量并初始化
    $x=15;                          //定义两个操作数并初始化
    $y=0;
    if($operator=='+')              //判断运算符并执行相应运算
        echo "$x+$y=".$x+$y;
    elseif($operator=='-')
        echo "$x-$y=".$x-$y;
    elseif($operator=='*')
        echo "$x*$y=".$x*$y;
    elseif($operator=='/'){
        if($y==0)                   //循环嵌套，判断除法中的除数是否为 0
            echo '除数为 0，计算错误！';
        else
            echo "$x/$y=".$x/$y;
    }
    else
        echo '请输入一个正确的运算符！';
?>
```

代码运行结果如图 3.9 所示。

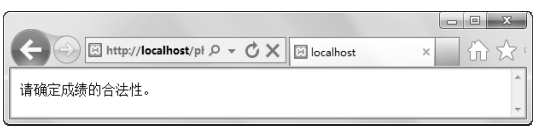

图 3.8　运行结果

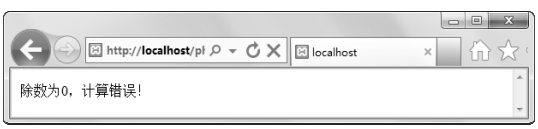

图 3.9　运行结果

以上代码的其他情况读者可以通过更改相应变量来查看，这里不再做演示。下面来演示一个嵌套出现歧义的示例。

【示例 3-5】以下代码演示一个会出现歧义的 if 语句嵌套。

```
<?php
    $x=6;                                   //定义并初始化一个变量
    if($x>0)                                //判断变量是否大于 0
        if($x<10)
            echo "0<$x<10";
    else                                    //当变量不大于 0 时输出
```

匹配

```
07          echo "$x<=0";
08  ?>
```

代码运行结果如图 3.10 所示。

当将代码中的$x 改为-6 时，运行的结果如图 3.11 所示。

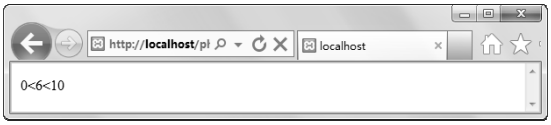

图 3.10　运行结果

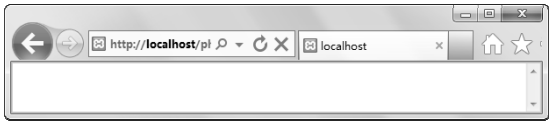

图 3.11　运行结果

可以从上面的运行结果看出，程序并没有输出我们期望的结果。这就是一个明显的嵌套错误，解决的办法就是，将 if 条件执行的语句改为复合语句的形式，修改后的代码如下：

```
01  <?php
02      $x=-6;                                  //定义并初始化一个变量
03      if($x>0){                               //将 if 判断后的代码改为语句块形式
04          if($x<10)
05              echo "0<$x<10";          匹配
06      }
07      else                                    //当变量不大于 0 时输出
08          echo "$x<=0";
09  ?>
```

代码运行结果如图 3.12 所示。

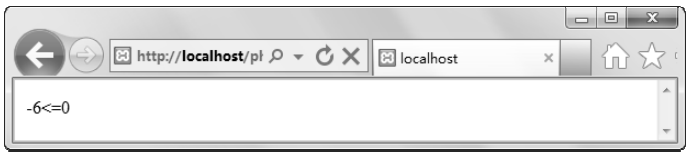

图 3.12　运行结果

我们可以看到，程序运行后就输出了正确的结果。

3.2.2　switch 语句

switch 语句用来实现按照不同的情况执行不同的语句。switch 语句常用于对变量的不同取值执行不同的语句。if…else…if 形式也可以用来实现类似的功能，但是它常用于针对变量的一个范围执行不同的语句，它的一般形式如下：

```
switch (表达式)
{
    case 常量 1:
        语句 1 或空;
        break;
    case 常量 2:
        语句 2 或空;
        break;
    …
    case 常量 n:
        语句 n 或空;
```

```
        break;
    default:
        语句 n+1 或空;
}
```

switch 语句执行时，首先计算表达式的值，并将它与每一个 case 后的常量进行比较。如果与某个常量相等，则执行对应的语句，遇到 break 语句则退出 switch 结构，否则就一直向下执行，直到遇到 break 或者 default。结构中的 default 不是必须有的，它用来匹配 case 情况之外的所有情况。switch 语句的程序流程如图 3.13 所示。

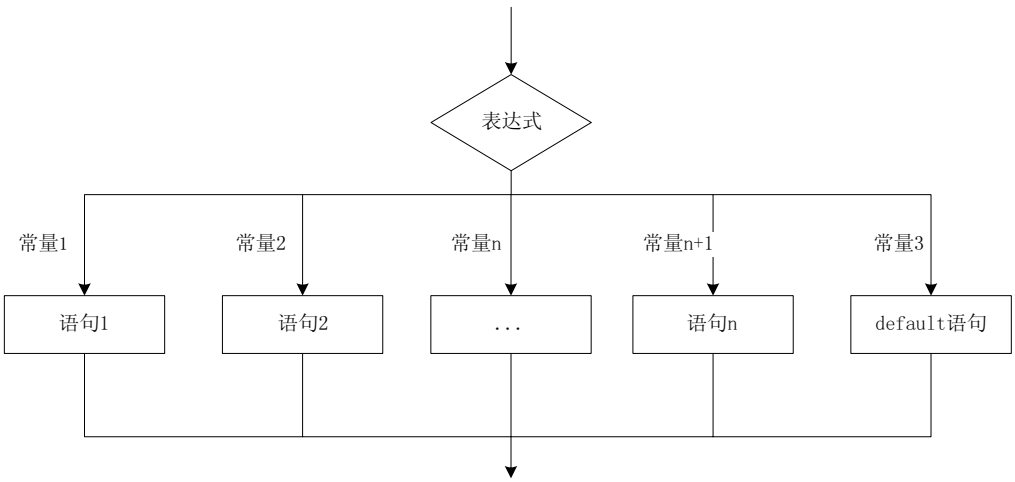

图 3.13　switch 流程图

【示例 3-6】以下代码演示 switch 语句的多种使用形式。

```
<?php
    $week=0;              //定义并初始化星期变量
    switch($week){
        case 0:           //变量为 0 的情况
            echo '星期日。';
            break;
        case 1:           //变量为 1 的情况
            echo '星期一。';
        case 2:           //变量为 2 的情况
            echo '星期二。';
            break;
        case 3:           //变量为 3 的情况
        case 4:           //变量为 4 的情况
            echo '星期三或者星期四。';
            break;
        case 5;           //变量为 5 的情况
            echo '星期五。';
            break;
        default:          //变量为以上 case 之外的所有情况
            echo '星期六。';
    }
?>
```

代码运行结果如图 3.14 所示。

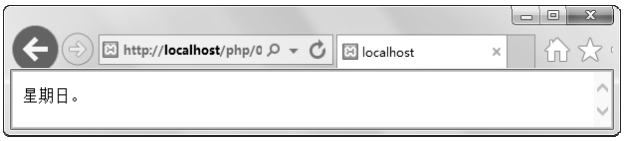

图 3.14　运行结果

以上代码中变量值为 0，因此运行结果会输出变量为 0 的情况。我们将代码中的 week 变量值设置为 1 后的运行结果如图 3.15 所示。

图 3.15　运行结果

以上运行结果输出了两个星期，是因为代码中 case 下的语句中没有 break 语句，因此程序无条件执行变量为 2 的情况下语句并未遇到 break 后停止。我们再将代码中 week 变量设置为 3 或者 4 后，运行结果如图 3.16 所示。

图 3.16　运行结果

变量为 3 或者 4 的运行情况是相同的，因为变量为 3 的情况下没有任何语句，因此会无条件执行后面情况下的语句，直到遇到 break。我们再将变量改为除 0～5 之外的任意类型值，运行程序后会输出如图 3.17 所示的结果，因为这些值都会被 default 匹配。

图 3.17　运行结果

在示例 3-6 简短的代码中就包含了 switch 的多种用法和技巧，读者应该多更换变量的值来查看输出结果并记住这些技巧。

3.3　循 环 语 句

计算机最擅长做的工作就是重复地执行系列的命令。循环语句就是指定一系列的语句并规定一个条件，让计算机重复执行这些语句直到满足规定的条件为止。PHP 中提供了 for 循环、while 循环和 do…while 循环，下面就来介绍这些知识。

3.3.1　for 循环

for 循环使用灵活性比较高，是 PHP 中使用最频繁的循环语句。for 循环的一般形式如下：

```
for (表达式 1; 表达式 2; 表达式 3)
{
  语句;
}
```

表达式 1 通常为赋值语句，用来初始化循环控制变量的初始值；表达式 2 通常为关系表达式或者逻辑表达式，用来确定何时停止循环；表达式 3 通常为递增或者递减表达式，用来对循环控制变量进行修改以逐步不满足表达式 2 的条件，否则就有可能造成无限循环。for 循环条件的流程图如图 3.18 所示。

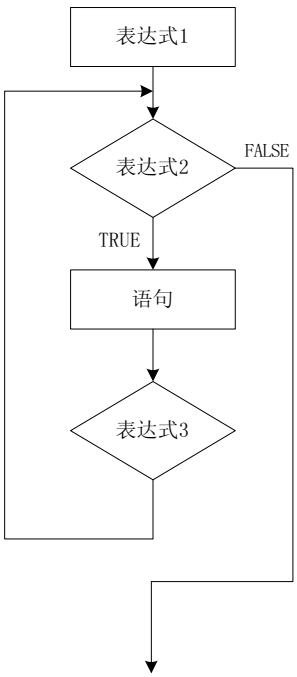

图 3.18　for 循环流程图

for 循环的执行过程如下：

（1）计算表达式 1 的值，为循环控制变量赋初值，该语句只在循环开始时执行一次。

（2）计算表达式 2 的值，如果其值为 TRUE，则执行循环体语句，否则退出循环。

（3）在每一次执行循环体语句结束后，运行一次表达式 3，以调整循环控制变量。然后返回第（2）步重新计算表达式 2 的值，依次重复，直到表达式 2 的条件不成立为止。

【示例 3-7】以下代码演示使用 for 循环输出数字 1～5。

```
01  <?php
02      for($i=1;$i<=5;$i++){
03          echo "$i<br />";          //循环体
04      }
05  ?>
```

代码运行结果如图 3.19 所示。

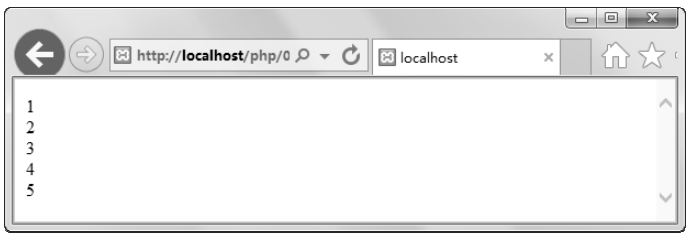

图 3.19　运行结果

for 循环的灵活之处就表现在它可以有多种形式，下面我们来分别介绍这些形式。

- 省略表达式 1：这种形式的表达式 1 通常写在 for 循环的外面。这种形式主要用在循环控制变量的初值不是已知常量，而是需要通过前面语句的执行计算得到的情况。

【示例 3-8】以下代码演示 for 循环省略表达式 1 的使用。

```
<?php
    $x=10;                    //初始化两个变量
    $y=5;
    $z=$x-$y;                 //初始化变量
    for(;$z<=5;$z++){         //省略表达式 1 的 for 循环
        echo "$z<br />";
    }
?>
```

代码运行结果如图 3.20 所示。

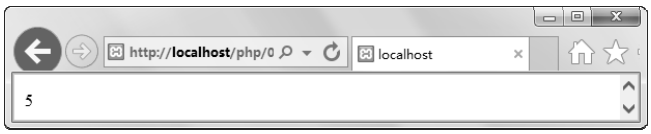

图 3.20　运行结果

将以上代码中的$y 的值改为 7 后，运行结果如图 3.21 所示。

图 3.21　运行结果

以上形式就是通过外部的运算来确定表达式 1 的值进而影响到 for 循环的次数。

- 省略表达式 2：因为表达式 2 默认值为 TRUE，因此这种形式如果在循环体中不加入跳转语句将会是一个无限循环。
- 省略表达式 3：这种形式的表达式 3 通常写在循环体内，在循环体内改变表达式的值，常用的形式如下所示。

```
<?php
    for($i=0;$i<=100;){          //省略表达式 3
        …
        $i=$x*3+4;               //在内部修改其值
        …
    }
?>
```

上面介绍的是省略 for 循环中表达式的情况。for 循环中不仅每个表达式可以为空，并且每个表达式还可以是由多个逗号分隔的表达式构成。

【示例 3-9】以下代码演示每个表达式为多个逗号分隔的表达式的情况。

```
<?php
    for($x=1,$y=2,$z=3;$x<=1,$y<=3,$z<=5;$x++,$y++,$z++){
                                          //多个逗号分隔的表达式
        echo "$x$y$z<br />";             //输出三个变量的值
    }
?>
```

代码运行结果如图 3.22 所示。

图 3.22　运行结果

从图 3.22 的运行结果可知，程序循环只进行了 3 次，这是因为虽然 for 循环中的表达式 2 所有用逗号分隔的表达式都会计算，但只取最后一个结果，因此$z 从 3 到 5 只用 3 次循环即可。

3.3.2　while 循环

while 循环与 for 循环相比使用比较简单，它通常用于循环次数不确定的情况，其一般形式如下：

```
while (表达式)
{
  语句;
}
```

while 循环在开始和每次执行循环体语句后均会判断表达式的值，如果为 TRUE，则执行循环体，如果为 FALSE 则退出 while 循环，它的执行流程图如图 3.23 所示。

【示例 3-10】以下代码演示使用 while 循环输出数字 1～5。

```
<?php
    $x=1;                  //初始化变量
    while($x<=5){          //执行 while 循环
        echo "$x<br />";
        $x++;
    }
?>
```

代码运行结果如图 3.24 所示。

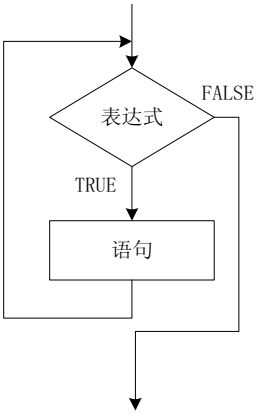

图 3.23　while 循环流程图

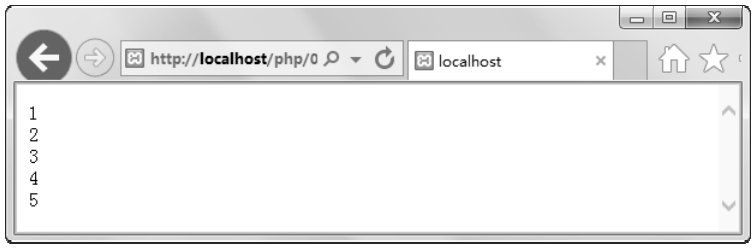

图 3.24　运行结果

我们使用 while 循环同样实现了输出数值 1～5。这里读者需要明白的一点是，通常循环语句间是可以进行转换的。

3.3.3　do…while 循环

do…while 循环与 while 类似，它的一般形式如下：

```
do
{
  语句;
}
while (表达式)
```

do…while 循环会首先执行一次循环体中的语句，然后再去判断表达式中的条件，如果为 TRUE 则继续执行循环体，如为 FALSE 则退出循环。也就是说，do…while 循环会保证循环体被执行一次，它执行的流程图如图 3.25 所示。

【示例 3-11】以下代码演示 do…while 循环的使用。

```
01  <?php
02      $x=10;          //初始化变量 x
03      do{
04          echo "$x";
05      }while($x<5);
06  ?>
```

代码运行结果如图 3.26 所示。

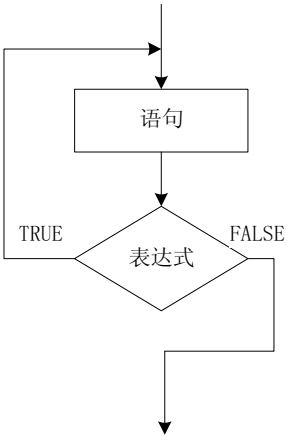

图 3.25　do…while 循环流程图

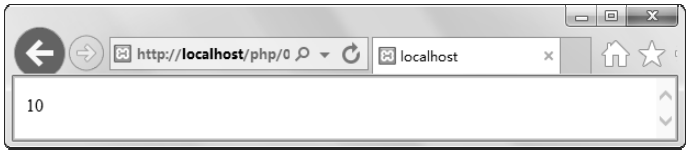

图 3.26　运行结果

在代码中的判断条件虽然为 FALSE，但是循环体语句仍会执行一次，这就是 do…while 循环的特性。

除了以上介绍的 3 种循环语句之外，PHP 还支持 foreach 循环语句，它是专门用来操作数组的循环语句，因此该循环将在数组学习部分进行讲解。

3.3.4　循环语句的嵌套

同选择语句类似，循环语句也是可以嵌套的，这里我们以实现如图 3.27 所示的输出来

编写代码。

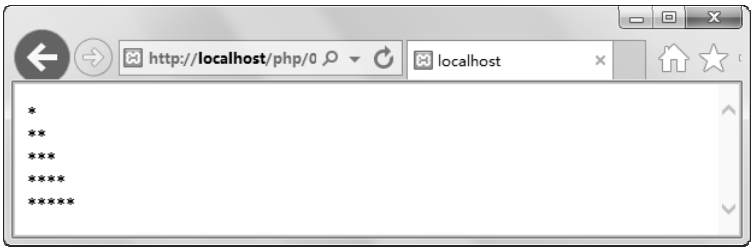

图 3.27　运行结果

上面输出的实现还是比较简单的，它的规律就在于在对应的行输出对应行数的星号(*)，然后在输出完毕后加入一个换行符即可。因此就可以通过 for 循环来每循环一次输出一个换行符，代码如下：

```
for(;;){
    echo '<br />';
}
```

每次循环开始，我们就可以在换行前输出对应的星号，这里同样使用 for 循环来实现，代码如下：

```
for(;;){
    echo '*';
}
```

由于输出输出的行数和星号数相同，因此可以通过这个关系将两个循环联系起来，简略代码如下：

```
for($i=$line;;){
    for(;$j=$i;){
        echo '*';
    }
    echo '<br />';
}
```

其中的变量 line 用来表示输出的行数，我们可以定义在循环之外用来修改行数。完整的代码如下：

```
01  <?php
02      $line=5;                        //用来控制行数
03      for($i=1;$i<=$line;$i++){
04          for($j=1;$j<=$i;$j++){
05              echo '*';               //输出星号
06          }
07          echo '<br />';              //输出换行
08      }
09  ?>
```

运行以上代码即可输出本节开始的效果图，读者可以通过改变 line 变量来控制输出的行数。

3.4　跳 转 语 句

跳转语句用于跳出一个循环结构或者跳转到指定的位置执行。PHP 中常用的跳转语句包括 break、continue、return 和 goto，下面就分别介绍这些语句。

3.4.1　break 语句

break 语句我们在学习 switch 语句时就多次使用过，它用来当前 for、foreach、while、do…while 或者 switch 结构的执行。break 可以接受一个可选的参数用来指定 break 跳出几层语言结构，默认值为 1，常用在循环嵌套中。

【示例 3-12】以下代码演示使用 break 语句跳出无限循环。

```
<?php
    for($x=1;;$x++){                    //省略表达式 2 的 for 循环将是无限循环
        echo "$x<br />";
        if($x==5){
            break;                      //使用 if 语句控制退出无限循环
        }                               跳转
    }
?>
```

代码运行结果如图 3.28 所示。

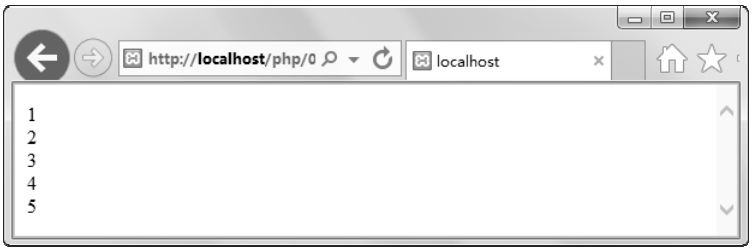

图 3.28　运行结果

从以上运行结果可以看出，这个无限循环并没有使浏览器崩溃，而是正确输出了结果，这也是 break 语句的常用方式。

【示例 3-13】以下代码演示使用 break 语句跳出多层结构。

```
<?php
    for($x=1;$x<=10;$x++){              //第一层 for 循环结构
        echo '1';
        for(;;){                        //第二层 for 循环结构
            echo '<br />2';
            break 2;                    //跳出两层循环结构
        }                               跳转
    }
?>
```

代码运行结果如图 3.29 所示。

以上代码通过 break 跳出了两个简单的 for 循环结构，在实际应用中，读者应该因地制宜灵活运用 break。

图 3.29　运行结果

3.4.2　continue 语句

continue 语句用来跳过本次循环中剩余的代码，直接进行下一次循环表达式的判断。与 break 语句类似，continue 语句也可以接受一个可选的参数来决定跳过多层结构。

【示例 3-14】以下代码演示使用 continue 语句跳过 3 的倍数数值的输出。

```
<?php
    for (i=1;i<=10;i++) {                    //for 循环输出数值
        if(i%3==0)                           //判断变量是否为 3 的整数倍
            continue;                        //跳过本次循环剩余语句
        echo "i<br />";                      //输出变量的值
    }
?>
```

跳转

代码运行结果如图 3.30 所示。

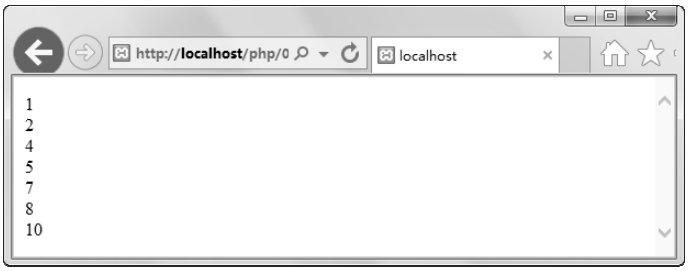

图 3.30　运行结果

以上代码通过循环输出 1～10 直接的数值，由于中间加入了跳转语句，致使是 3 的整数倍的数值不被输出。

【示例 3-15】以下代码演示使用 continue 语句跳出多层结构。

```
<?php
    $i=0;                                                    //初始化循环控制变量
    while($i++<3){
        echo '第一层 while 循环。<br />';                    //第一层循环输出
        while(TRUE){
            echo "第二层 while 循环。<br />";              //第二层循环输出
            while(TRUE){
                echo "第三层 while 循环。<br />";          //第三层循环输出
                continue 3;
            }
            echo "这里不会被输出<br />";                  //由于 continue 控制语句跳出多层
                                                             循环，因此这里不会被输出
        }
        echo "这里同样不会被输出。";                       //由于 continue 控制语句跳出多层
                                                             循环，因此这里不会被输出
    }
```

```
15  ?>
```

代码运行结果如图 3.31 所示。

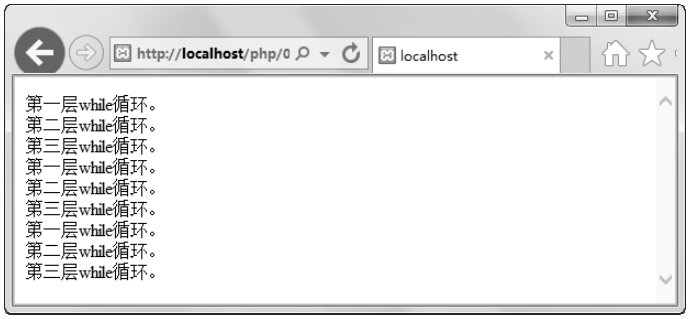

图 3.31　运行结果

3.4.3　goto 语句

goto 语句用来跳转到程序的指定位置开始执行，可以替代 break 跳出多层结构。PHP 中的 goto 语句被限制在只能在同一个文件和作用域中跳转。例如不可以跳入任何循环和 swith 结构中。goto 语句的常用形式如下：

```
goto tab
…
tab:
…
```

tab 用来表示程序要跳转到位置的标记与 goto 后的标记要一致。

【示例 3-16】以下代码演示使用 goto 语句使程序跳转执行。

```
<?php
    for($i=1;$i<=5;$i++){                  //使用 for 循环循环输出 1～5
        if($i==3)
            goto ECH;                      //当$i 为 3 时候跳出 for 循环
        echo "$i<br />";
    }
    ECH:                                   //执行 goto 语句后程序将从此处开始执行
    echo "此时\$i=$i";
?>
```

跳转

代码运行结果如图 3.32 所示。

图 3.32　运行结果

我们可以看到 goto 语句的使用非常方便，但是在编程过程中应该尽量少使用，过多地使用 goto 语句会使程序流程变得混乱，进而使程序阅读和调试都变得非常困难。PHP 中除了上述 3 种跳转语句之外，return 语句也是用来使程序进行跳转，return 语句的知识将在函

数的学习部分进行讲解。

3.5　小　　结

本章主要学习了构成程序整体框架的语言结构，这些知识包括语句的基本概念、选择语句、循环语句和跳转语句。其中的循环语句部分是本章的重点，而且由于循环语句的执行流程均有各自的特点，掌握起来也是比较难的。因此读者应该多实践，从实践中理解这些知识。

3.6　本 章 习 题

1. 使用 if…else 判断一个数是否大于等于 0 并输出如图 3.33 所示的结论。

图 3.33　运行效果

2. 使用 while 循环输出数字 1～9，执行结果应该类似图 3.34 所示。
3. 使用 for 循环输出 1～20 之间的偶数，执行结果应该类似图 3.35 所示。

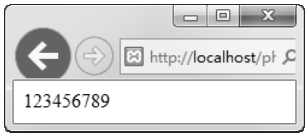

图 3.34　运行效果

图 3.35　运行效果

4. 使用任意一种循环，输出如图 3.36 所示的九九乘法表。

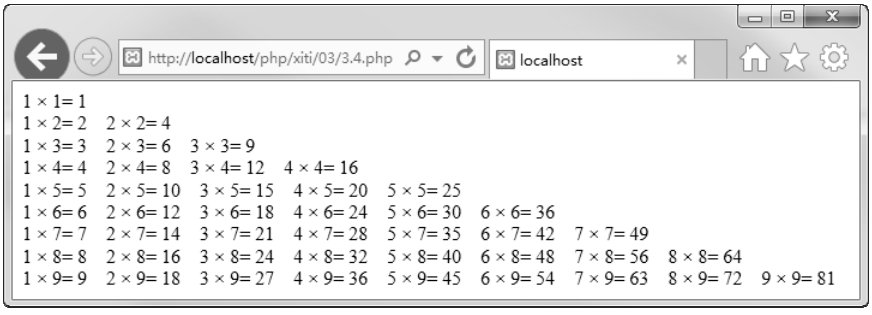

图 3.36　九九乘法表

第 4 章　函　　数

函数是由一系列语句组成的一个功能独立且完整的代码块。它可以用来完成一些指定的操作，并返回操作后的结果。使用函数可以简化代码结构，并降低代码编写工作量。本章将主要介绍使用函数的优势和函数的使用方法。

4.1　使用函数的优势

使用函数有两个最主要的优势：函数是一个完整而且独立的功能块，因而使用函数的代码结构更加清晰，可以明显增强代码的可读性；函数使得程序的可复用性非常强。PHP 系统默认已经定义了多种类型的千余个函数，这些函数可以被任何一个 PHP 程序使用，这就是函数可复用性的最明显表现。

4.2　使 用 函 数

PHP 中的函数分为两类，分别是系统函数和用户自定义函数。系统函数的使用比较简单，本节将重点介绍自定义函数的知识。

4.2.1　自定义函数和调用函数

自定义函数即为用户自己定义用来实现指定需求的函数。定义函数的一般形式如下：

```
function 函数名(参数列表){
    函数体;
    return 返回值;
}
```

定义函数需要使用 function 关键字；函数名是用来为这一系列操作定义的名称；参数列表是数据传入函数的入口；函数体是函数的注意功能实现部分；函数都有返回值，但在没有显式调用 return 语句返回一个值的时候，函数会默认返回空类型（void）。

【示例 4-1】下面就来定义一个函数用来实现两个数相加并返回操作结果的函数。其所示如下：

```
01  function add($x,$y){
02      return $x+$y;           //返回变量之和
03  }
```

以上代码中，add 是函数名；$x 和$y 是函数接受的参数；return 后的表达式即为函数的返回值。函数调用的一般形式如下：

```
函数名(参数列表);
```

函数名即在函数定义时指定的名称，参数列表需要对应函数定义时的参数列表传入。

【示例 4-2】以下代码演示使用自定义函数 add()计算两数之和。

```
01  <?php
02      function add($x,$y){                    //定义函数 add()
03          return $x+$y;
04      }
05      echo add(5,6);                          //调用函数并输出结果
06  ?>
```

调用

代码运行结果如图 4.1 所示。

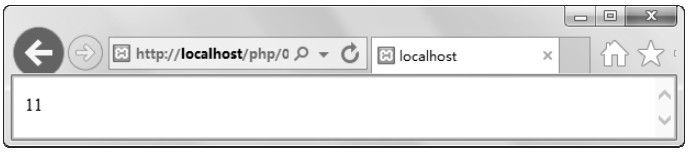

图 4.1　运行结果

我们可以看到只要通过使用正确定义的函数名，就可以轻易得到相应的结果。

在上面的学习中，我们已经认识到 return 可以将函数的运算结果返回。return 语句也是跳转语句的一种，它会立刻终止当前函数的执行并返回结果。因此，return 语句通常会放在整个函数体的最后。

4.2.2　函数的参数

函数的参数有实际参数和形式参数之分。形式参数即为一个形式，它只要可以保证参数能正确地传递到函数体中即可。因此，只要符合 PHP 命名规范的名称都可以使用。以下函数的功能是完全一致的。

```
function add($x,$y){
    return $x+$y;
}
function add($a,$b){                //使用不同的形式参数
    return $a+$b;
}

function add($abc,$def){            //使用不同的形式参数
    return $abc+$def;
}
```

函数在定义时可以为形式参数设定默认参数，默认参数的定义必须是从右向左，并且默认参数的右边不可有为设置默认参数的形式参数。以下是一些默认参数的示例。

```
function func($x,$y,$z=0){          //正确的默认参数设置方法
    …
}

function func($x,$y=1,$z=0){        //正确的默认参数设置方法
    …
}
```

```
function func($x=2,$y=1,$z=0){          //正确的默认参数设置方法
    …
}
function func($x=2,$y=1,$x){            //错误的默认参数设置方法
    …
}
function func($x=0,$y=1,$z){            //错误的默认参数设置方法
    …
}
```

【示例 4-3】以下代码演示为 add()函数设置默认参数后的运行效果。

```
<?php
    function add($x=2,$y=3){                    //定义函数并设置默认参数
        return $x+$y;
    }
    echo add();                                 //不传入参数调用 add()函数
    echo '<br />'.add(5);                       //传入一个参数调用 add()函数
    echo '<br />'.add(5,6);                     //传入两个参数调用 add()函数
?>
```

调用　$x=5　$x=5　$y=6

代码运行结果如图 4.2 所示。

图 4.2　运行结果

在调用的时候，传入的参数即为实际参数，它会从左向右依次传入函数的参数列表，设置默认值的参数将被替换。

4.2.3　参数的传递

参数的传递类似于变量的赋值。PHP 参数传递方式分为按值传递和按引用传递。类似地，按值传递将传递实际参数的副本，而引用传递则传递实际参数本身。

【示例 4-4】以下代码演示使用按值传递方式交换两个数值。

```
<?php
    function swap($x,$y){                       //定义交换数值函数
        $temp=$x;
        $x=$y;
        $y=$temp;
    }
    $m=5;
    $n=15;
    echo "交换前：<br />\$m=$m<br />\$n=$n";     //输出交换前变量的值
    swap($m,$n);                                //调用函数交换变量数值
    echo "<br />交换后：<br />\$m=$m<br />\$n=$n";   //输出交换后变量的值
?>
```

$x=$m
$y=$n

代码运行结果如图 4.3 所示。