



福州大学至诚学院
FUZHOU UNIVERSITY ZHICHENG COLLEGE

服务端开发技术

杨雄



3.流程控制与函数

01

选择结构

02

循环结构

03

流程控制语句

04

PHP函数

05

结构化编程

Part.1

3.1 选择结构

3.1 选择结构

控制结构类型

计算机**程序**是由一系列的**语句**构成的，为了实现某种特定的**算法**，需要对这些语句的**执行顺序**进行控制，从而控制程序的**执行流程**。

PHP的基本控制结构有3种：**顺序结构**、**选择结构**和**循环结构**。

顺序结构就是按照程序中语句出现的先后顺序依次执行，如赋值语句、输入输出语句等；

选择和循环结构需要一定的**流程控制语句**来控制程序的执行顺序。

3.1 选择结构

选择结构

PHP的选择结构分为**单分支**选择结构与**多分支**选择结构。单分支选择由**if else**语句实现；多分支选择由**if else**语句的嵌套，或由**switch case**语句来实现。

(1) 单分支

Bool值

可以是复合语句

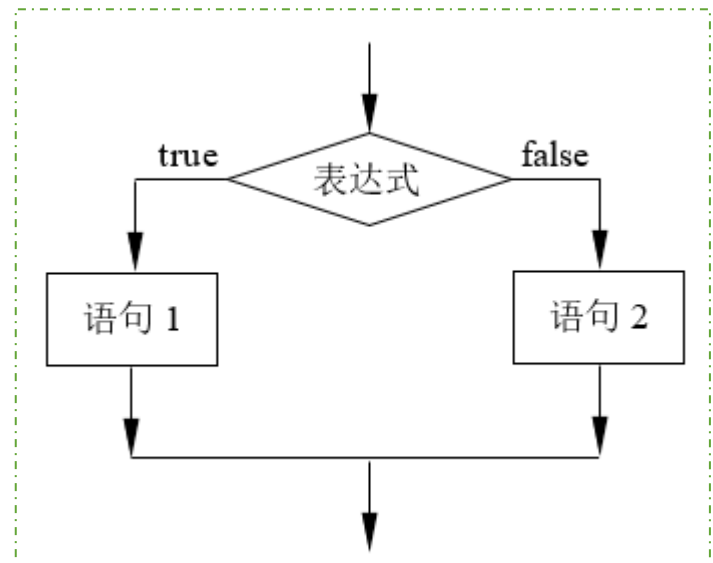
```
if (表达式)
```

```
    语句1
```

```
else
```

可以是空语句
(else略)

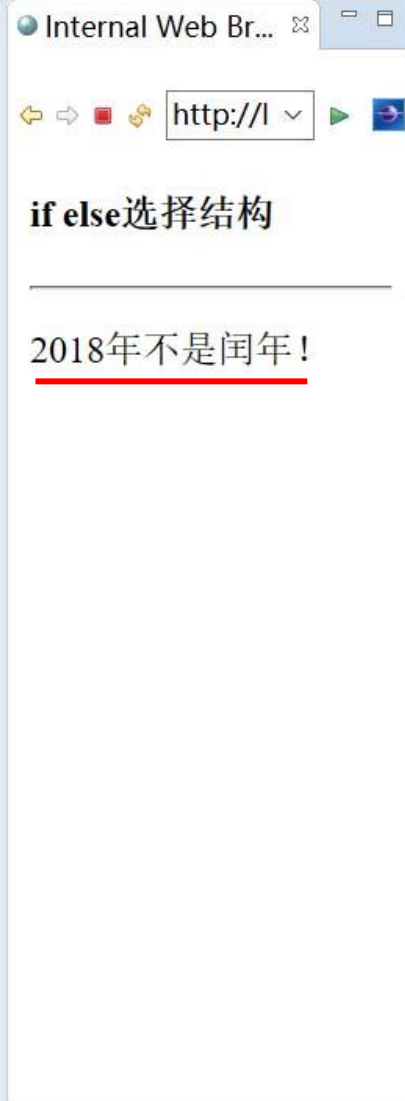
```
    语句2
```



3.1 选择结构

【例】 判断某年是否是闰年。

```
example4_1.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.1 选择结构</title>
6 </head>
7 <body>
8   <h4>if else选择结构</h4>
9   <hr />
10  <p><?php
11      $year = 2018;
12      $isLeapYear = null;
13      $result = null;
14      $isLeapYear = ($year % 4 == 0) && ($year % 100 != 0)
15                      || ($year % 400 == 0);
16      if ($isLeapYear)
17          $result = "年是闰年! ";
18      else
19          $result = "年不是闰年! ";
20      echo $year.$result;
21  ?></p>
22 </body>
23 </html>
```



3.1 选择结构

(2) 多分支

PHP有**3种**多分支结构：**嵌套if else**、**if elseif**、**switch**

```
if (表达式 1)
    if (表达式 2) 语句 1
    else 语句 2
else
    if (表达式 3) 语句 3
    else 语句 4
```

```
if (表达式 1) 语句 1
elseif (表达式 2) 语句 2
elseif (表达式 3) 语句 3
...
else 语句 n
```

```
switch (表达式)
{
    case 常量表达式 1: 语句 1
    case 常量表达式 2: 语句 2
    ...
    case 常量表达式 n: 语句 n
    default: 语句 n+1
}
```

3.1 选择结构

使用switch语句应注意下列问题。

- (1) switch语句后面的表达式可以是**整型或字符串**,
- (2) 每个常量表达式的值**不能相同**, 但次序不影响执行结果。
- (3) 每个case分支**可以有多条语句**, 但不必用{}包围,
- (4) 每个case语句只是一个入口标号, 并不能确定执行的终止点, **因此每个case分支最后应该加break语句**, 用来结束整个switch结构, **否则会从入口点开始一直执行到 switch结构的结束点。**
- (5) 当若干分支需要执行相同操作时, 可以使**多个case分支共用一组语句。**
- (6) default 子句**不是必需的**, 可以省略。

3.1 选择结构

【例】 使用嵌套语句，判断某年是否是闰年。

嵌套

嵌套

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.2 选择结构的嵌套</title>
6 </head>
7 <body>
8   <h4>if else结构的嵌套</h4><hr />
9   <p><?php
10       $year = 2000;
11       $result = null;
12       if ($year%100 != 0)
13           if ($year%4 == 0)
14               $result = "年是闰年! ";
15           else
16               $result = "年不是闰年! ";
17       else
18           if ($year%400 == 0)
19               $result = "年是闰年! ";
20           else
21               $result = "年不是闰年! ";
22       echo $year.$result;
23   ?></p>
24 </body>
25 </html>
```

Internal Web Browser

/example4_2.php

if else结构的嵌套

2000年是闰年!

3.1 选择结构

【例】 使用if elseif语句，判断某年是否是闰年。

index.php

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>if elseif语句</title>
6 </head>
7 <body>
8   <h4>if elseif语句</h4><hr />
9   <p><?php
10     $year = 2018;
11     $result = null;
12     if ($year%400 == 0)
13       $result = "年是闰年! ";
14     elseif(($year%100 != 0) && ($year%4 == 0))
15       $result = "年是闰年! ";
16     else
17       $result = "年不是闰年! ";
18     echo $year.$result;
19   ?></p>
20 </body>
21 </html>
```

Internal Web Browser

http://localhost/pp

if elseif语句

2018年不是闰年!

if ... elseif

3.1 选择结构

【例】 输入一个0~6的整数，将其转换成星期输出。

The screenshot shows a PHP script in a text editor and its execution in a web browser. The script uses a switch statement to convert a day number (0-6) into a Chinese weekday name. Annotations highlight key features of the switch statement.

```
10 <?php
11     $day = 5;
12     $weekday = null;
13     switch ($day) {
14         case 0:
15             $weekday = '星期日';
16             break;
17         case 1:
18             $weekday = '星期一';
19             break;
20         case 2:
21             $weekday = '星期二'; break;
22         case 3:
23             $weekday = '星期三'; break;
24         case 4:
25             $weekday = '星期四'; break;
26         case 5:
27             $weekday = '星期五'; break;
28         case 6:
29             $weekday = '星期六'; break;
30         default:
31             $weekday = '输入的数据不正确!';
32             break;
33     }
34     echo $weekday;
35 ?>
```

Annotations:

- 表达式(整型或字符串)**: Points to the `$day` variable in the `switch` statement.
- 常量表达式**: Points to the `case` labels (0, 1, 2, 3, 4, 5, 6).
- 可以省略**: Points to the `break;` statements at the end of each case.
- 跳出switch结构, 否则会顺序执行下面的语句**: Points to the `break;` statements.

Browser output:

switch选择结构

星期五

Part.2

3.2 循环结构

3.2 循环结构

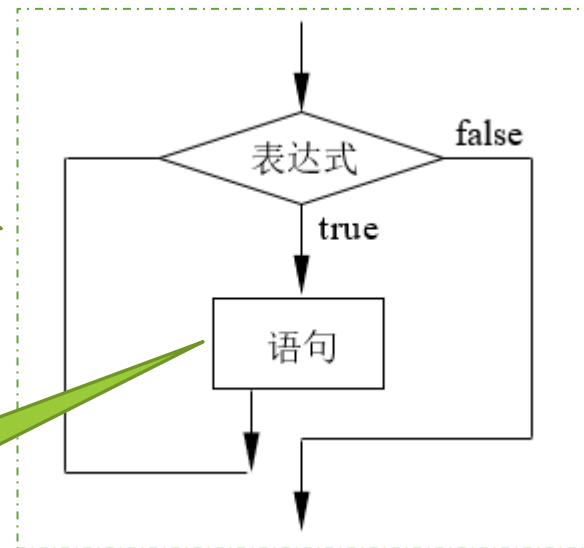
在程序执行过程中，有时候需要将一段**代码反复地执行**，这时就要用到**循环结构**。PHP提供了以下**4种形式**的循环控制结构。

(1) while循环

执行顺序

```
while (表达式) 语句
```

注意：其中应该包含改变循环条件表达式的语句，否则会出现死循环



3.2 循环结构

【例】 求1~100中所有自然数之和。

```
example4_4.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.4 while循环结构</title>
6 </head>
7 <body>
8   <h4>while循环结构</h4>
9   <hr />
10  <p><?php
11      $i = 1;
12      $sum = 0;
13      while ($i <= 100) {
14          $sum += $i;
15          $i++;
16      }
17      echo '从1~100所有自然数的和为: '.$sum;
18  ?></p>
19 </body>
20 </html>
```

条件表达式

改变表达式的值

Internal Web Browser

http://localhost/exam

while循环结构

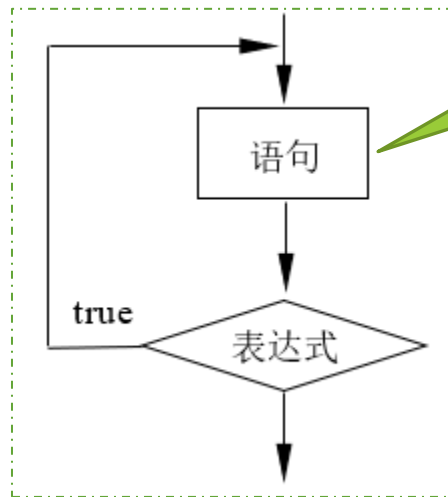
从1~100所有自然数的和为: 5050

3.2 循环结构

(2) do ... while循环

执行顺序

```
do 语句  
while (表达式)
```



注意：其中应该包含改变循环条件表达式的语句，否则会出现死循环

与while循环的主要区别在于，do while循环的循环体至少会被执行一次。

3.2 循环结构

【例】 使用do while循环，求1~100中所有自然数之和。

```
example4_5.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.5 do while循环结构</title>
6 </head>
7 <body>
8   <h4>do while循环结构</h4>
9   <hr />
10  <p><?php
11      $i = 1;
12      $sum = 0;
13
14      do {
15          $sum += $i;
16          $i++;
17      }while ($i <= 100);
18      echo '从1~100所有自然数的和为: ' . $sum;
19  ?></p>
20 </body>
21 </html>
```

Internal Web Browser
http://localhost/exam

do while循环结构

从1~100所有自然数的和为: 5050

改变表达式的值

条件表达式

3.2 循环结构

(3) for循环

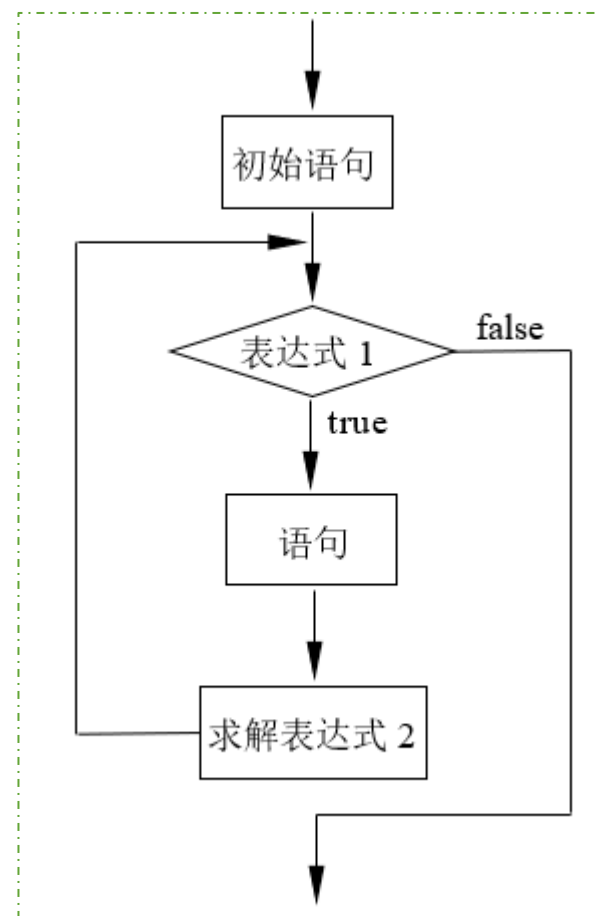
给控制变量赋初值

执行顺序

for(初始语句; 表达式 1; 表达式 2)
语句

循环控制条件

改变控制变量



3.2 循环结构

关于for循环结构的说明如下。

- (1) 初始语句、表达式1、表达式2都可以省略，**分号不能省略**。
- (2) **表达式1是循环控制条件**，如果省略，循环将无终止地进行下去，
- (3) 初始语句可以是一个**表达式语句或声明语句**。
- (4) 表达式2一般用于改变循环控制变量的值，如果省略或者是其他与循环条件无关的表达式，则应该在循环体中另有语句改变循环条件，**以保证循环能正常结束**。
- (5) 如果省略表达式1和3，只有表达式2，则完全等同于while循环结构。

3.2 循环结构

【例】 使用for循环，求1~100中所有自然数之和。

```
example4_6.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.6 for循环结构</title>
6 </head>
7 <body>
8   <h4>for循环结构</h4>
9   <hr />
10  <p><?php
11      $sum = 0;
12      for ($i = 1; $i <= 100; $i++){
13          $sum += $i;
14      }
15      echo '从1~100所有自然数的和为: '.$sum;
16      //在循环结构之前给循环控制变量赋值
17      $i = 1;
18      for ($sum = 0; $i <= 100; $i++){
19          $sum += $i;
20      }
21      echo '<br/>'. '从1~100所有自然数的和为: '.$sum;
22  ?></p>
23 </body>
24 </html>
```

Internal Web Browser

http://localhost/example

for循环结构

从1~100所有自然数的和为: 5050
从1~100所有自然数的和为: 5050

循环控制变量

循环控制条件

3.2 循环结构

(4) foreach循环

foreach是针对于**数据集**的循环，比如**数组**、**列表**等。

下面以数组为例来说明它的用法。假设数组名称为arr，数组元素的值用value表示，数据元素的键用key表示。

```
foreach($arr as $value) 语句  
或者：  
foreach($arr as $key => $value) 语句
```

foreach针对数组的循环，实际上就是对**数组的遍历**。每次循环时，将当前数组元素的值赋给变量value，如果是第2种方式，还需要将当前数组元素的键赋给变量key，直到数组的最后一个元素。

3.2 循环结构

【例】 使用foreach循环，求1~100中所有自然数之和。

The image shows a PHP code editor on the left and an internal web browser on the right. The code in the editor is as follows:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.7 foreach循环结构</title>
6 </head>
7 <body>
8   <h4>foreach循环结构</h4>
9   <hr />
10  <p><?php
11      $value = 0;
12      $sum = 0;
13      $i = 1;
14      $arr = array();
15      while ($i<=100) {
16          $arr[] = $i;
17          $i++;
18      }
19      foreach ($arr as $value)
20          $sum += $value;
21      echo '从1~100所有自然数的和为: '.$sum;
22  ?></p>
23 </body>
24 </html>
```

Annotations in the image:

- A green callout bubble labeled "给数组赋值" (Assign value to array) points to the while loop (lines 15-18).
- A red box highlights the foreach loop (lines 19-20).
- A green callout bubble labeled "遍历数组并求和" (Iterate over array and sum) points to the foreach loop.

The web browser on the right shows the output of the code:

foreach循环结构

从1~100所有自然数的和为: 5050

Part.3

3.3 流程控制语句

3.3 流程控制语句

上面介绍的**选择结构与循环结构**，本质上都属于**流程控制**的范畴。除此之外，PHP还提供了其他几个程序流程的控制语句，即**break**、**continue**、**goto**、**exit**。

1、break

在**switch选择**

在**循环结构**

The screenshot shows a code editor with a file named 'index.php' and an 'Internal Web Browser' window. The PHP code calculates the sum of natural numbers from 1 to 100 using a while loop. A red box highlights the loop body, and a green callout bubble points to the 'break;' statement with the text '跳出循环' (Exit loop). The browser window displays the result: '1~100内自然数的和为:5050'.

```
1 <?php
2 /**
3  * break语句
4  * 求1~100内自然数之和
5  */
6
7 $sum = 0;
8 $i = 1;
9 while ($i) {
10     $sum += $i;
11     if($i == 100) break;
12     $i++;
13 }
14 echo("1~100内自然数的和为:{$sum}");
```

跳出循环

Internal Web Browser
ppt_example/pptc402/
1~100内自然数的和为:5050

3.3 流程控制语句

2、continue

```
index.php ✕
1 <?php
2 /**
3  * continue语句
4  * 求1~100内自然
5  */
6
7 $sum = 0;
8 $i = 1;
9 while ($i) {
10     $sum += $i;
11     if($i == 100) continue;
12     $i++;
13 }
14 echo("1~100内自然数的和为:{$sum}");
```

将前面程序中的break换成continue后，程序能输出正确结果吗？

3.3 流程控制语句

continue.php

```
1 <?php
2 /**
3  * continue语句
4  * 求1~100内所有偶数之和
5  */
6
7 $sum = 0;
8 for ($i = 1; $i <= 100; $i++) {
9     if ($i % 2 != 0) continue;
10     $sum += $i;
11 }
12 echo("1~100内所有偶数之和为:{$sum}");
```

中断本次循环

Internal Web Browser

http://localhost/ppt_€

1~100内所有偶数之和为:2550

3.3 流程控制语句

3、goto

使程序的执行流程，**跳转到语句标号所指定的语句。**

goto语句的语法格式为：

goto 语句标号

其中“**语句标号**”是**用来标识语句的标识符**，放在语句的最前面，并用**冒号**与语句分开。例如：

goto END;

...

END:

echo '这是文件结尾!';

...

这里，“END”为语句标号，执行goto语句后，程序会跳转到由该标号标记的语句开始往下执行。

goto语句的使用会破坏程序的结构，**应该少用或不用。**

3.3 流程控制语句

4、exit

强行退出代码的运行，而不管它在代码的什么位置。

Part.4

3.4 PHP函数

3.4 PHP函数

函数是一种可以在任何被需要的时候执行的**代码块**。

在程序开发过程中，使用函数可以**提高代码的重用性**、**减少系统错误**、**缩短开发周期**，从而**提升系统开发效率**，并增强其可维护性及可靠性。

在PHP中，函数可以分为**用户自定义**与**系统内置**两种类型。

PHP提供了大量的**内置函数**，这些函数**成就了PHP的强大功能**。

1、函数的定义

在PHP中，由于版本的更新，函数定义的语法格式也出现了一些差别，这里使用**PHP 7中的函数定义格式**。

3.4 PHP函数

PHP 7格式：【可以兼容PHP5格式】

```
function 函数名(含类型说明的形式参数表): 类型说明符  
{  
    函数体  
}
```

关键词

PHP标识符，不区分大小写

形参

返回值类型

PHP 5格式：

实现函数功能的语句序列

```
function 函数名(形式参数表)  
{  
    函数体  
}
```

PHP7增强了函数参数及返回值数据类型的限制，与C++及Java等强类型语言类似。

3.4 PHP函数

【例】函数定义。

可以不指定类型

返回值类型

可以是变长参数

定义函数

函数名

形参类型

形参变量

返回值

返回值类型

```
example4_9.php
9  <p><?php
10  function myprint1(string $param1, int $param2):void {
11      echo $param1, $param2, '<br/><br/>';
12      return ;
13  }
14  function myprint2( $param1, $param2):void
15      echo $param1, $param2, '<br/><br/>';
16      return ;
17  }
18  function myprint3( $param1, $param2, ...$param):void {
19      echo $param1, $param2, $param, '<br/>';
20      return myprint3($param);
21      echo '<br/><br/>';
22      return ;
23  }
24  function myprint4(string $param1, int $param2):string {
25      echo $param1, $param2, '<br/><br/>';
26      return '这是函数myprint4返回的字符串!';
27  }
28
29  myprint1('a = ', 10);
30  myprint2('a = ', 10);
31  myprint3('a = ', 10, 11, 12);
32  $returnStr = myprint4('a = ', 10);
33  echo $returnStr;
34  ?></p>
```

PHP函数定义

```
a = 10
a = 10
a = 10
Array ( [0] => 11 [1] => 12 )
a = 10
```

这是函数myprint4返回的字符串！

3.4 PHP函数

内部函数：在PHP的函数中，可以定义另外一个函数。

The image shows a code editor window titled 'function.php' with the following PHP code:

```
1 <?php
2 /**
3  * PHP内部函数
4  */
5 function myprint() {
6     $result = null;
7     //内部函数
8     function sum($start, $end) {
9         $s = 0;
10        for ($i = $start; $i <= $end; $i++) {
11            $s += $i;
12        }
13        return $s;
14    }
15    $result = sum(1, 10);
16    echo $result;
17 }
18 myprint();
19 echo '<br />'.sum(1, 100);
```

Annotations in the image:

- A green callout bubble labeled **函数** (Function) points to the `myprint()` function definition.
- A green callout bubble labeled **内部函数** (Internal Function) points to the `sum()` function definition inside `myprint()`.
- A green callout bubble labeled **调用内部函数** (Call Internal Function) points to the `sum(1, 10)` call inside `myprint()`.
- Another green callout bubble labeled **调用内部函数** (Call Internal Function) points to the `sum(1, 100)` call at the bottom of the script.

The browser window on the right shows the output of the script: 55 and 5050.

3.4 PHP函数

2、函数的调用及参数传递

(1) 函数的调用

函数定义后，就可以在程序中调用该函数。语法格式如下：

函数名(实参列表);

实参列表应与函数定义中的**形参列表**相对应，即**参数的个数与数据类型**。

当形参列表中有**明确的数据类型说明**时，实参列表中对应的参数必须与之有**相同的数据类型**，就像C语言一样。

3.4 PHP函数

PHP函数的**调用方式**主要有以下5种：

使用函数名

这是最**基本的函数调用方式**，其调用格式符合上面给出的基本语法。

使用函数名变量

PHP支持**变量函数**，也就是可以声明一个变量，**通过变量来访问函数**。

如果一个**变量名后出现小括号“()”**，PHP将寻找**与变量的值同名的函数**，并且尝试调用它。

3.4 PHP函数

【例】 PHP的变量函数。

```
example4_10.php
17 }
18 function myprint3( $param1, $param2, ...$param):void {
19     echo $param1, $param2, '<br/>';
20     print_r($param);
21     echo '<br/><br/>';
22     return ;
23 }
24 function myprint4(string $param1, int $param2):string {
25     echo $param1, $param2, '<br/><br/>';
26     return '这是函数myprint4返回的字符串!';
27 }
28
29 $functionName = 'myprint1';
30 $functionName('a = ', 10);
31
32 $functionName = 'myprint2';
33 $functionName('a = ', 100);
34
35 $functionName = 'myprint3';
36 $functionName('a = ', 10, 11, 12);
37
38 $functionName = 'myprint4';
39 $returnStr = $functionName('a = ', 1000);
40
41 echo $returnStr;
42 ?></p>
```

通过变量
调用函数

Internal Web Browser
http://localhost/examplee/

PHP的变量函数

a = 10

a = 100

a = 10

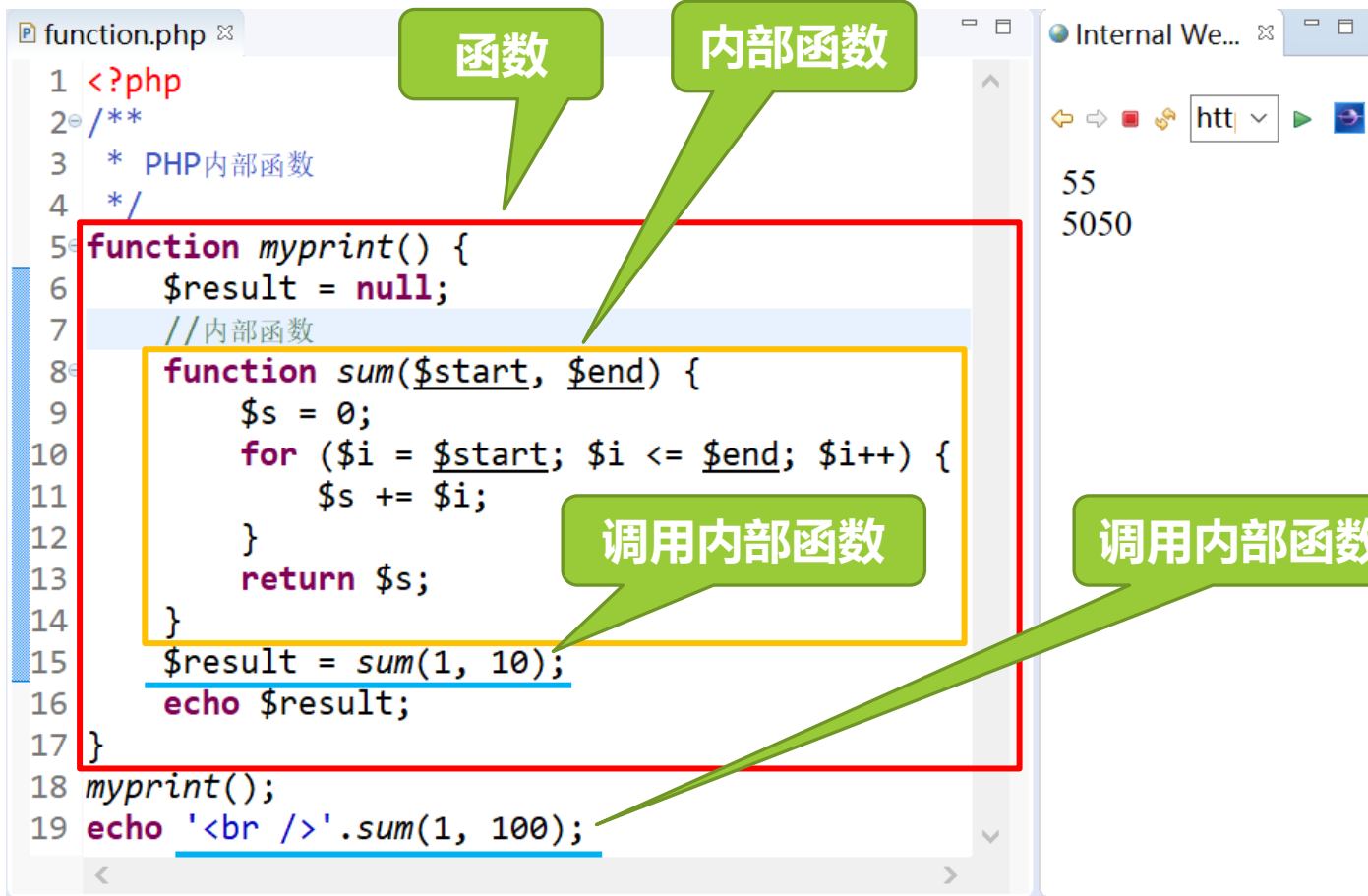
Array ([0] => 11 [1] => 12)

a = 1000

这是函数myprint4返回的字符串！

3.4 PHP函数

内部函数的调用



The screenshot shows a code editor with a file named 'function.php'. The code defines a function 'myprint()' and an internal function 'sum()'. A red box highlights the 'myprint()' function, with a green callout bubble labeled '函数' (Function) pointing to it. A yellow box highlights the 'sum()' function, with a green callout bubble labeled '内部函数' (Internal Function) pointing to it. A green callout bubble labeled '调用内部函数' (Call internal function) points to the line '\$result = sum(1, 10);' inside 'myprint()'. Another green callout bubble labeled '调用内部函数' (Call internal function) points to the line 'echo '
'.sum(1, 100);' at the bottom of the file. To the right of the code editor is a browser window titled 'Internal We...' showing the output '55' and '5050' on separate lines.

```
1 <?php
2 /**
3  * PHP内部函数
4  */
5 function myprint() {
6     $result = null;
7     //内部函数
8     function sum($start, $end) {
9         $s = 0;
10        for ($i = $start; $i <= $end; $i++) {
11            $s += $i;
12        }
13        return $s;
14    }
15    $result = sum(1, 10);
16    echo $result;
17 }
18 myprint();
19 echo '<br />'.sum(1, 100);
```

注意：

PHP的内部函数在主函数外面进行调用时，调用语句必须位于主函数调用语句之后。

3.4 PHP函数

函数的嵌套调用

【例】 PHP函数的嵌套调用。

example4_12.php

```
6 </head>
7 <body>
8   <h4>函数的嵌套</h4><hr />
9   <p><?php
10    function myprint1(string $param1, int $param2):void {
11        echo '这是myprint1函数的输出结果! ';
12        echo $param1, $param2;
13        return ;
14    }
15    function myprint2( $param1, $param2):void {
16        echo '这是myprint2函数的输出结果! ';
17        echo $param1, $param2, '<br/>';
18        myprint1($param1, $param2);
19        return ;
20    }
21    function myprint3( $param1, $param2, ...$param):void {
22        echo $param1, $param2, '<br/>';
23        print_r($param);
24        echo '<br/>';
25        myprint2($param1, $param2);
26        return ;
27    }
28    myprint3('a = ', 100, 20, 30);
29    ?></p>
30 </body>
31 </html>
```

Internal Web Browser

http://localhost/examplee/cha|

函数的嵌套

a = 100
Array ([0] => 20 [1] => 30)
这是myprint2函数的输出结果! a = 100
这是myprint1函数的输出结果! a = 100

3.4 PHP函数

函数的递归调用

【例】 PHP函数的递归调用。

函数

递归结
束条件

```
example4_13.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.13 PHP的函数调用</title>
6 </head>
7 <body>
8 <h4>函数的递归调用</h4><hr />
9 <p><?php
10 function fac(int $n):int {
11     $f = 0;
12     if ($n == 0) {
13         $f = 1;
14     }else{
15         $f = fac($n - 1) * $n;
16     }
17     return $f;
18 }
19
20 $n = 8;
21 echo "${n}的阶乘为: ".fac($n);
22
23 ?></p>
24 </body>
25 </html>
```

函数的递归调用

8的阶乘为: 40320

调用自己

3.4 PHP函数

(2) 函数的参数传递

在函数未被调用时，函数的形参并不占有实际的内存空间，也没有实际的值。只有在函数被调用时才为形参分配存储单元，并将实参与形参结合。

函数的参数传递：就是指形参与实参结合的过程。

PHP支持函数参数的传递方式有3种，即值传递、引用传递和默认参数。

值传递

值传递是指当发生函数调用时，给形参分配内存空间，并用实参来初始化形参，即直接将实参的值传递给形参。

这一过程是参数值的单向传递过程，一旦形参获得了值便与实参脱离关系，此后无论形参发生了怎样的改变，都不会影响到实参。

3.4 PHP函数

【例】编写函数，完成两个整型变量值的交换。

函数

形参

调用

实参

函数功能正确

程序输出错误

函数参数的值传递

调用函数前:
a = 8 b = 5

函数内部交换后
a = 5 b = 8

调用函数后:
a = 8 b = 5

3.4 PHP函数

引用传递

引用是一种特殊的变量，可以被认为是另一个变量的名字。

用引用作为形参，在函数调用时发生的参数传递，称为**引用传递**。

函数的调用**采用引用的参数传递方式**，函数调用时**传递的是实参的引用**，**是双向传递过程**。

3.4 PHP函数

【例】通过函数调用，完成两个整型变量值的交换。

函数

形参为引用

调用

引用传递

函数功能正确

程序输出正确

函数参数的引用传递

调用函数前:
a = 8 b = 5

函数内部交换后:
a = 5 b = 8

调用函数后:
a = 5 b = 8

3.4 PHP函数

默认参数

PHP中的函数在定义时，还可以为一个或多个形参指定默认值。

默认值必须是常量表达式，也可以是NULL，并且当使用默认参数时，任何默认参数必须放在任何非默认参数的右侧。

3.4 PHP函数

【例】 带默认参数的函数调用。

```
example4_16.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.16 带默认参数函数的调用</title>
6 </head>
7 <body>
8   <h4>带默认参数函数的调用</h4><hr />
9   <p><?php
10     function myprint(string $param1, int $param2 = 10):void {
11       echo $param1, $param2, '<br/><br/>';
12       return ;
13     }
14
15     echo '给函数传入1个实参:<br/>';
16     myprint('a = ');
17     echo '给函数传入2个实参:<br/>';
18     myprint('a = ', 100);
19   ?></p>
20 </body>
21 </html>
```

Internal Web Browser
http://localhost/e:
带默认参数函数的调用
给函数传入1个实参:
a = 10
给函数传入2个实参:
a = 100

使用默认参数

默认参数

不使用默认参数

3.4 PHP函数

(3) 回调函数

【例】 回调函数的定义及应用。

在PHP中，可以把函数名作为参数进行传递。采用这种方式传递参数的函数，称为回调函数。

The image shows a screenshot of a web browser displaying the output of a PHP script. The script is named `example4_17.php` and is located at `http://localhost/examplee`. The browser output shows the title "回调函数" and two lines of text: "10以内的所有偶数之和: 30" and "10以内能被3整除的整数之和: 18".

The PHP code is as follows:

```
7=<body>
8  <h4>回调函数</h4><hr />
9  <p><?php
10= function sum($fun, int $n, int $m):int {
11      $sum = 0;
12      for ($i = 0; $i <= $n; $i++) {
13          if($fun($i, $m)) {
14              $sum += $i;
15          }
16      }
17      return $sum;
18  }
19
20  function filter(int $n, int $mod):bool {
21      $flag = ($n % $mod == 0);
22      return $flag;
23  }
24
25  $fun = 'sum';
26  echo '10以内的所有偶数之和: '.$fun('filter',10,2).'\n';
27  echo '10以内能被3整除的整数之和: '.$fun('filter',10,3);
28  ?></p>
29 </body>
```

Annotations in the image:

- 回调函数**: Points to the `sum` function definition.
- 使用形参函数调用**: Points to the `$fun` parameter in the `sum` function definition.
- 函数名**: Points to the `$fun` parameter in the `sum` function definition and the `$fun` variable assignment.
- 函数名**: Points to the `filter` function definition.
- 函数调用**: Points to the `$fun('filter',10,2)` and `$fun('filter',10,3)` calls in the `echo` statements.

3.4 PHP函数

3、变量的作用域

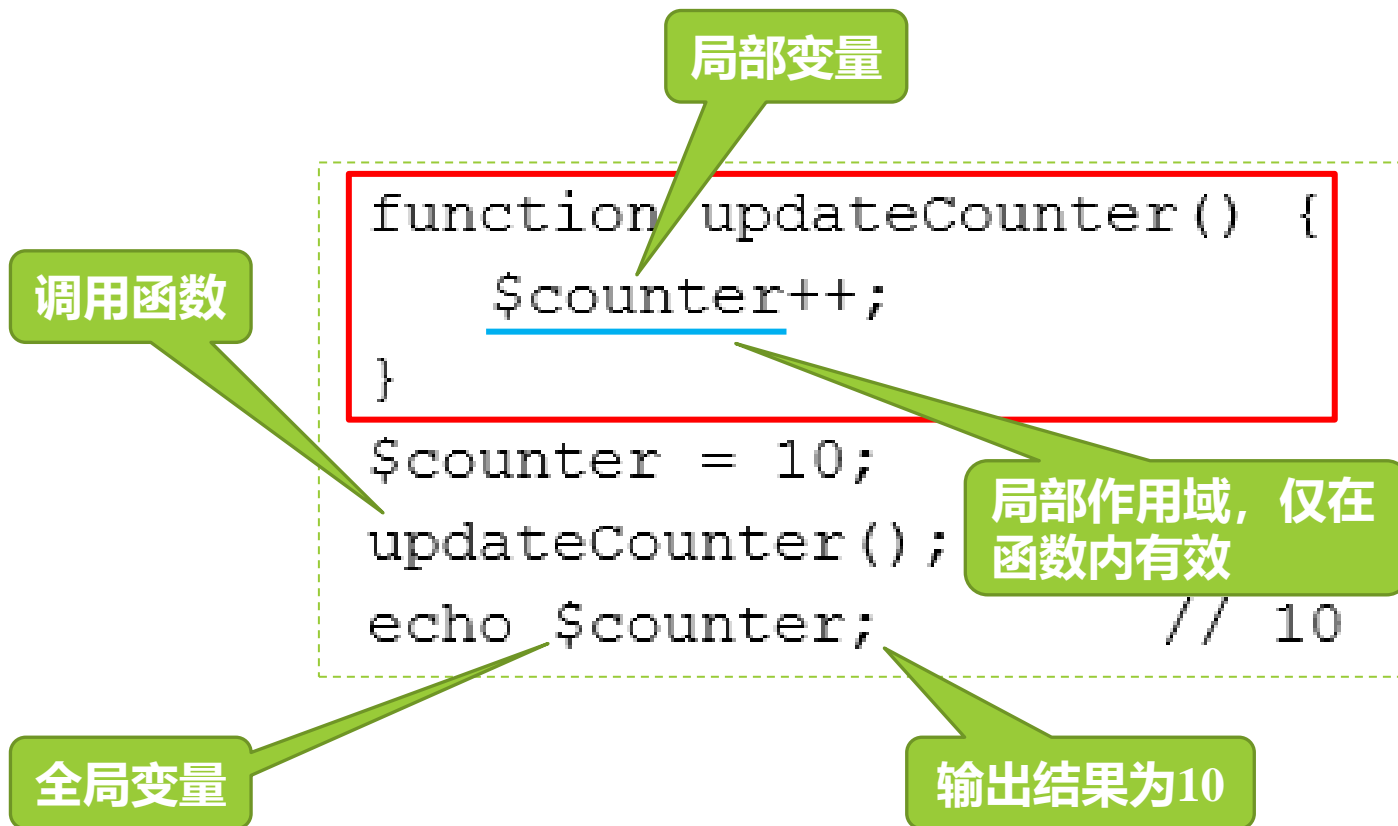
变量的作用域也就是**变量的有效范围**，或者说在程序的哪些部分可以访问它。PHP的变量有3种类型的作用域，即**局部**作用域、**全局**作用域和**静态**作用域。

(1) 局部作用域

在一个函数中声明的变量是该函数的**局部变量**。即它仅在该**函数的内部**（包括嵌套函数）**是可见的**，在函数的外部是不可访问的。

3.4 PHP函数

【例如】



3.4 PHP函数

(2) 全局作用域

在函数外面声明的变量称为**全局变量**，它们可以在**本文件的任何部分**被访问。

默认情况下全局变量在内部函数中是不可访问的。

为了让一个**函数能够访问全局变量**，可以在函数内使用**global关键字**来声明变量，或者是使用PHP的**\$GLOBALS数组**。

3.4 PHP函数

【例】 局部变量和全局变量。

```
example4_18.php
7<body>
8<h4>变量的作用域</h4><hr />
9<p><?php
10function updateCounter() {
11    $counter++;
12}
13function updateCounter1() {
14    global $counter;
15    $counter++;
16}
17function updateCounter2() {
18    $GLOBALS['counter']++;
19}
20
21$counter = 10;
22echo 'counter = ' . $counter . '<br/><br/>';
23@updateCounter();
24echo '第1次调用函数后: ' . $counter . '<br/><br/>';
25updateCounter1();
26echo '第2次调用函数后: ' . $counter . '<br/><br/>';
27updateCounter2();
28echo '第3次调用函数后: ' . $counter . '<br/><br/>';
29?></p>
30</body>
```

局部变量

全局变量

全局数组

Internal Web Browser

http://localhost/example

变量的作用域

counter = 10

第1次调用函数后: 10

第2次调用函数后: 11

第3次调用函数后: 12

3.4 PHP函数

(3) 静态作用域

用**static关键字**在函数中声明的变量称为**静态变量**。

静态变量在一个**函数被多次调用时**，**其值不会丢失**，但此变量**仅在该函数内是可见的**。

3.4 PHP函数

【例】静态变量。

```
example4_19.php
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" >
5 <title>例4.19 静态变量</title>
6 </head>
7 <body>
8   <h4>静态变量</h4><hr />
9   <p><?php
10     function updateCounter() {
11       static $counter = 0;
12       $counter++;
13       echo '静态变量counter现在为: '.$counter.'<br/><br/>';
14     }
15
16     $counter = 10;
17     echo 'counter = ' . $counter.'<br/><br/>';
18     updateCounter();
19     updateCounter();
20     updateCounter();
21     echo 'counter = ' . $counter;
22   ?></p>
23 </body>
24 </html>
```

静态变量

连续调用



静态变量值
在上一次基
础上加1

3.4 PHP函数

4、内置函数

内置函数是指**PHP预定义**的函数。

在实际编程中，更多的是使用系统内置函数。

PHP内置了大量的函数，实现了各种各样的业务逻辑功能。前面使用的输出函数、数据类型转换函数等，都是系统内置的函数。

熟悉**PHP内置函数**的方式很多，其中最简单的就是**使用PHP的帮助文档**，。在这里可以查询到函数的原型、使用方法以及示例代码。

3.4 PHP函数

PHP Manual

文件(E) 编辑(E) 查看(V) 转到(G) 帮助(H)

隐藏

查找

上一步

下一步

上一步

前进

刷新

主页

字体

打印

选项(O)

目录(C)

索引(N)

搜索(S)

收藏夹(I)

PHP 手册

版权信息

PHP 手册

入门指引

安装与配置

语言参考

安全

特点

函数参考

影响 PHP 行为的扩展

音频格式操作

身份认证服务

针对命令行的扩展

压缩与归档扩展

信用卡处理

加密扩展

数据库扩展

日期与时间相关扩展

文件系统相关扩展

国际化与字符编码支持

图像生成和处理

邮件相关扩展

数学扩展

非文本内容的 MIME 输出

进程控制扩展

其它基本扩展

其它服务

搜索引擎扩展

针对服务器的扩展

Session 扩展

文本处理

变量与类型相关扩展

Web 服务

Windows 专用扩展

XML 操作

图形用户界面(GUI) 扩展

使用 SystemTap 监控 PHP DTrace 静态探针 PHP 手册 影响 PHP 行为的扩展

PHP Manual

函数参考

Tip

参见 [扩展库列表 / 归类](#).

- [影响 PHP 行为的扩展](#)
 - [APC](#) — Alternative PHP Cache (可选 PHP 缓存)
 - [APCu](#) — APC User Cache
 - [APD](#) — Advanced PHP debugger
 - [bcompiler](#) — PHP 字节码编译器
 - [BLENC](#) — Blenc - BLOWfish ENCoder for PHP source scripts
 - [错误处理](#) — 错误处理和日志记录
 - [htscanner](#) — htaccess-like support for all SAPIs
 - [included](#) — Inclusion hierarchy viewer
 - [Memtrack](#)
 - [OPcache](#)
 - [输出控制](#) — 输出缓冲控制
 - [PHP 选项/信息](#) — PHP 选项和信息
 - [runkit](#)
 - [scream](#) — Break the silence operator
 - [uopz](#)
 - [Weakref](#) — Weak References
 - [WinCache](#) — Windows Cache for PHP
 - [Xhprof](#) — 层次式性能分析器
- [音频格式操作](#)

3.4 PHP函数

内置函数应用 – 案例

一、数学函数

二、字符串函数

- 1.判断类型函数
- 2.获取子串或子串位置
- 3.字符串其他相关函数

三、数组函数

- 1.数组的键和值
- 2.数组的其他操作函数

<https://blog.csdn.net/muwenbofx/article/details/120855524>

Part.5

3.5 结构化编程

3.5 结构化编程

1、文件包含

为了更好地组织代码，提高代码的重用性、以及代码的维护与更新效率，在实际开发过程中，常常按照功能将代码进行分类，并将其分别存放在不同的文件上，然后使用包含语句将它们组合成一个整体。

在PHP中，通常使用include、require、include_once或require_once语句来实现文件的包含。

include

include语法格式：

include '完整路径文件名'

或

include('完整路径文件名')

3.5 结构化编程

require

require语句虽然与include语句功能相似，但也略有差别。

在包含文件时，如果没有找到被包含的文件，include语句会发出警告信息，程序继续运行；而require语句则会发出致命错误，程序停止运行。

另外，对于include_once、require_once语句来说，与include、require的作用几乎相同；不同的是，带once的语句会先检查要导入的文件是否已经在该程序中的其他地方被包含过，如果有的话，就不再导入该文件，因而避免了同一文件被重复包含。

3.5 结构化编程

【例】重复包含的问题。



a.php



b.php



base.php



c.php

64 > www > include > base.php

```
<?php
```

```
function foo(){  
    echo 'foo';  
}
```

p64 > www > include > b.php

```
<?php
```

```
include "base.php";  
function foo_b(){  
    echo 'b';  
    foo();  
}
```

p64 > www > include > c.php

```
<?php
```

```
include "a.php";  
include "b.php";
```

(!) Fatal error: Cannot redeclare foo() (previously declared in C:\wamp64\www\include\base.php:2) in C:\wamp64\www\include\base.php on line 2

Call Stack

#	Time	Memory	Function
1	0.0013	409400	{main}()
2	0.0036	410040	include('C:\wamp64\www\include\b.php')

3.5 结构化编程

【例】文件包含的应用。



The screenshot shows a development environment with a code editor and a web browser. The code editor displays the file `example6_10a.php` with the following PHP code:

```
1 <?php include './include/myfunction.php' ?>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8" >
6 <title>例6.10 文件包含</title>
7 <style type="text/css">
8     #wd{
9         color:red;
10        font-weight:bold}
11 </style>
12 </head>
13 <body>
14 <h4>文件包含</h4><hr />
15 <p><?php
16     $today = getdate();
17     $weekday = tocweekday($today['wday']);
18     echo '今天是<span id="wd">'.$weekday.'</span>';
19     ?></p>
20 <?php include './include/footer.php';?>
21 </body>
22 </html>
```

The web browser, titled "Internal Web Browser", shows the rendered output at the URL `http://localhost/example/chapt`. The page content is:

文件包含

今天是**星期五**

At the bottom of the browser window, the footer text reads: 清华大学出版社 中国北京 2018.

3.5 结构化编程

2、自定义函数库

函数是结构化程序设计的**功能模块**，是**实现代码重用的核心**。

在实际编程过程中，为了更好地组织代码，**使自定义的函数可以在同一个项目的多个文件中使用**，通常将多个自定义的函数组织到同一个文件或多个文件中。这些**收集函数定义的文件就是创建的PHP函数库**。

函数库定义完成后，如果在PHP的脚本中想要使用库中的函数，只需要使用上一小节中介绍的**包含语句**，将函数库文件载入到脚本程序中即可。

函数库不是定义函数的PHP语法，而是编程时的一种设计模式。

3.5 结构化编程

【例】 使用自定义函数库，实现商品的列表显示。

example6_11.php

```
1 <?php
2 include './include/example6_11.data.php';
3 include './include/example6_11.func.php';
4 ?>
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta charset="UTF-8">
9 <title>例6.11</title>
10 <link href='./css/example6_11.css' rel='stylesheet'>
11 </head>
12 <body>
13 <h3>自定义函数库</h3><hr/>
14 <h4>商品列表</h4>
15 <?php echo table($thead, $items); ?>
16 </body>
17 </html>
```

```
23 $html .= '</tr>';
24 return $html;
25 }
26
```

Internal Web Browser

http://localhost/example/chapter06a/exa

自定义函数库

商品列表

序号	商品名称	单价(元)
1	PHP Web应用开发教程	50
2	面向对象程序设计	40
3	Visual C++程序设计	30
4	数据结构与应用教程	35

```
45 }
46 $html .= '</table>';
47 return $html;
48 }
```


3.5 结构化编程

3、命名空间

在使用包含语句进行文件包含的过程中，由于被包含的文件有可能来自不同的开发人员，所以在**常量、函数及类的命名上，就有可能发生冲突**。这时若使用命名空间，则可以解决这个问题。

从广义上来说，**命名空间是一种封装事物的方法**。例如，在操作系统中目录用来将相关文件分组，对于目录中的文件来说，它就扮演了命名空间的角色。

在PHP中，命名空间用来解决在编写类库或函数时碰到的以下两类问题。

- 用户编写的代码与PHP内部的类、函数、常量或第三方类、函数、常量之间的名字冲突。
- 为很长的标识符名称(通常是为了缓解上面的第一类问题而定义的)创建一个别名(或简短)，以提高源代码的可读性。

PHP中的命名空间，与C++中的“命名空间”以及Java中的“包”的概念相似。

3.5 结构化编程

【例】命名空间的定义与使用。

The image shows a development environment with a code editor and a web browser. The code editor has three tabs: `nnamespace.php`, `other_namespace.php`, and `example6_12.php`. The `nnamespace.php` tab is active, showing the following PHP code:

```
1 <?php
2 namespace WP;
3
4 function test() {
5     echo '这是WP命名空间';
6 }
```

The `other_namespace.php` tab is also visible, showing the following PHP code:

```
1 <?php
2 namespace MA;
3
4 function test() {
5     echo '这是MA命名空间';
6 }
```

The `example6_12.php` tab is also visible, showing the following PHP code:

```
1 <?php
2 use function WP\test;
3 include './include/nnamespace.php';
4 include './include/other_namespace.php';
5 ?>
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="UTF-8" >
10 <title>例6.12 命名空间</title>
11 </head>
12 <body>
13     <h4>命名空间</h4><hr />
14     <p><?php
15         WP\test();
16         MA\test();
17         test();
18     ?></p>
19 </body>
20 </html>
```

The web browser (Internal Web Browser) shows the output of the `example6_12.php` file. The address bar is `a/example6_12.php`. The page title is "命名空间". The content of the page is:

命名空间

这是WP命名空间 中的函数
这是MA命名空间 中的函数
这是WP命名空间 中的函数



补充

阅读代码

```
<?php
function makeyogurt($container = "bowl", $flavour)
{
    return "Making a $container of $flavour yogurt.\n";
}

echo makeyogurt("raspberry");
?>
```

总结

PHP7和PHP8版本的区别

```
echo 't' . 3 + 1 . 'w';
```

在PHP7中输出为: 1w

在PHP8中输出为: t4w

<https://www.php.net/manual/zh/language.operators.precedence.php>

总结

★流程控制：条件控制、循环控制

★函数定义和调用

★内置函数

★文件引用