

文件包含

文件包含：在一个 PHP 脚本中，去将另外一个文件（PHP）包含进来，去合作完成一件事情。

文件包含的作用

文件包含的意义：

1、 要么使用被包含文件中的内容，实现代码的共享（重用）：向上包含（索要）

向上包含：在当前脚本要用某个代码之前包含别的文件

2、 要么自己有东西可以给别的文件使用，实现代码的共享（重用）：向下包含（给予）

向下包含：在自己有某个东西的时候，需要别的脚本来显示（自己代码写完之后包含其他文件）

最大的作用：分工协作，每个脚本做的事情不一样，因此可以使用协作方式，让多个脚本共同完成一件事情。

文件包含四种形式

在 PHP 中文件的包含有四种形式（两种大形式）

Include：包含文件

Include_once：系统会自动判断文件包含过程中，是否已经包含过（一个文件最多被包含一次）

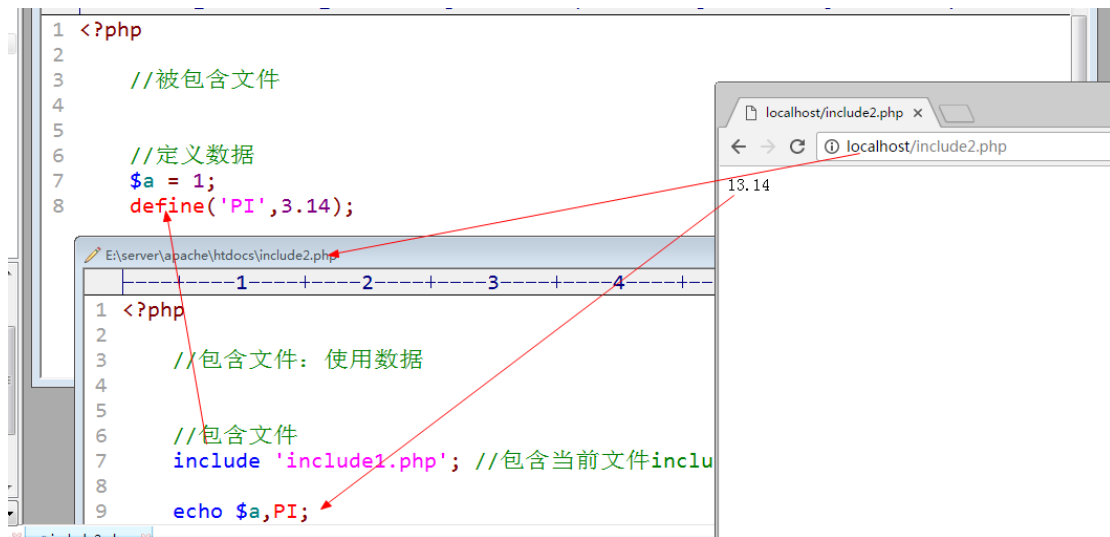
Require：与 include 相同

Require_once：以 include_once 相同

包含基本语法

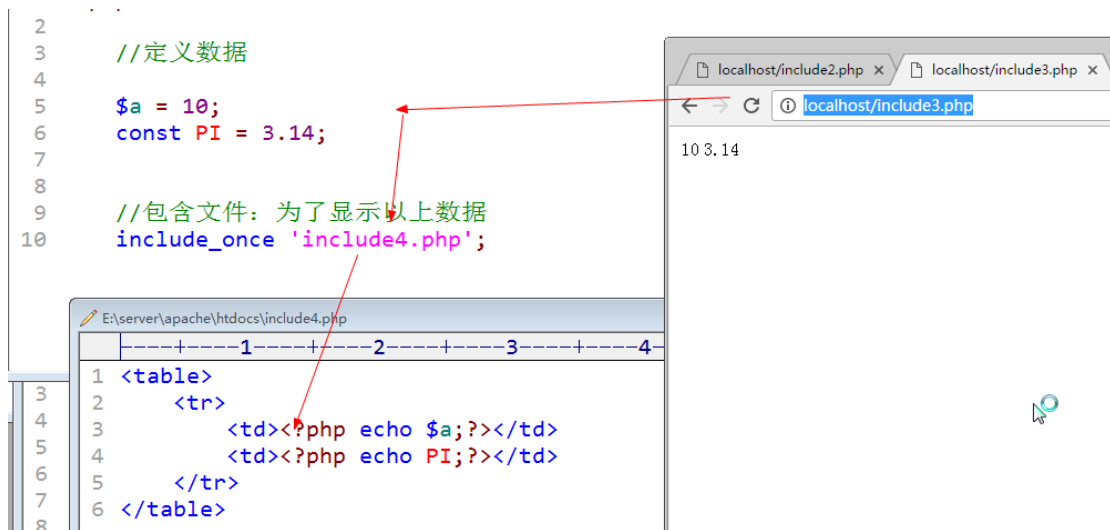
Include '文件名字';

Include('文件名字'); //文件名字：路径问题



以上方式：是先包含文件，后使用文件中的内容（向上包含）

向下包含：先准备内容，然后包含另外的文件，在另外的文件中，使用当前的内容



文件加载原理

PHP 代码的执行流程

- 1、读取代码文件（PHP 程序）
- 2、编译：将 PHP 代码转换成字节码（生成 opcode）
- 3、zendengine 来解析 opcode，按照字节码去进行逻辑运算
- 4、转换成对应的 HTML 代码

文件加载原理：

- 1、在文件加载（`include` 或者 `require`）的时候，系统会自动的将被包含文件中的代码相当于嵌入到当前文件中
- 2、加载位置：在哪加载，对应的文件中的代码嵌入的位置就是对应的 `include` 位置

3、 在 PHP 中被包含的文件是单独进行编译的


PHP 文件在编译的过程中如果出现了语法错误，那么会失败（不会执行）；但是如果被包含文件有错误的时候，系统会在执行到包含 `include` 这条语句的时候才会报错。

Include 和 require 区别

Include 和 `include_once` 的区别：


Include 系统会碰到一次，执行一次；如果对同一个文件进行多次加载，那么系统会执行多次；

```
1 //定义数据
2 $a = 1;
3 define('PI',3.14);
4
5 //包含文件
6 include 'include1.php'; //包含当前文件include2.php所在文件夹下
7
8 echo $a,PI;
9
10
11 //再次加重
12 include 'include1.php';
13
```



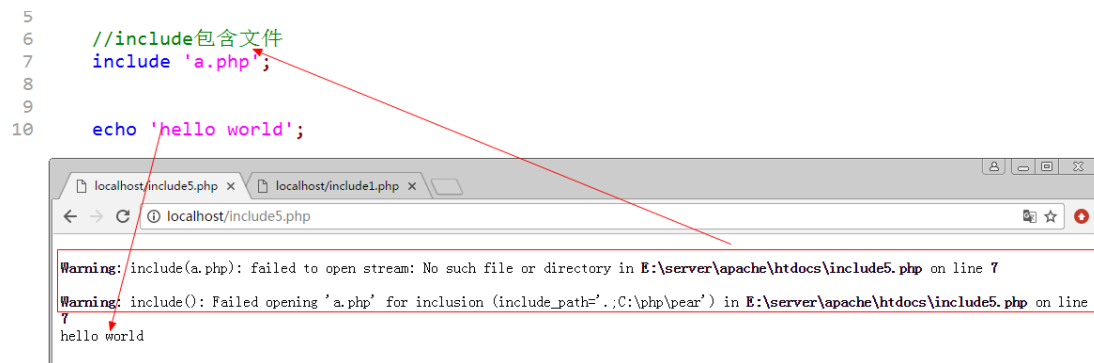
Include_once: 系统碰到多次，也只会执行一次。

```
2 //被包含文件
3
4 //定义数据
5 $a = 1;
6 define('PI',3.14);
7
8 //包含文件
9 include 'include1.php'; //包含当前文件include2.php所在文件夹下
10
11 echo $a,PI;
12
13 //再次加载
14 //include 'include1.php';
15
16 //include_once
17 include_once 'include1.php';
18
```

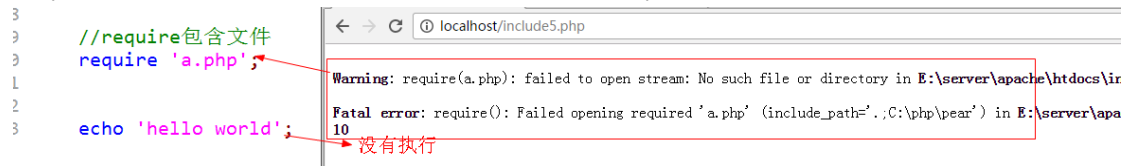


Require 和 include 的区别：本质都是包含文件，唯一的区别在于包含不到文件的时候，报错的形式不一样

Include 的错误级别比较轻：不会阻止代码执行



Require 要求较高：如果包含出错代码不再执行（require 后面的代码）



文件加载路径

文件在加载的时候需要指定文件路径才能保证 PHP 正确的找到对应的文件。

文件的加载路径包含两大类：

1、绝对路径

从磁盘的根目录开始（本地绝对路径）

Windows：盘符 C:/路径/PHP 文件

Linux：/路径/PHP 文件

从网站根目录开始（网络绝对路径）

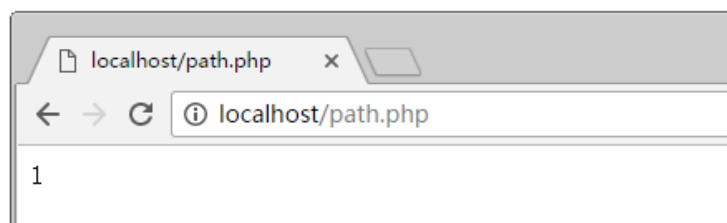
/：相对于网站主机名字对应的路径

Localhost/index.php -> E:/server/apache/htdocs/index.php

//绝对路径

```
include_once 'E:/server/apache/htdocs/include1.php';
```

```
echo $a;
```



2、相对路径：从当前文件所在目录开始的路径

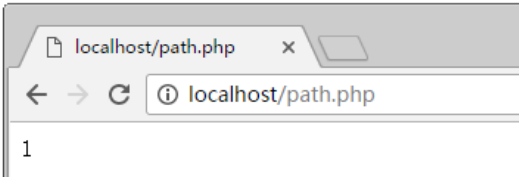
./：表示当前文件夹

../：上级目录（当前文件夹的上一层文件夹）

```

~      //绝对路径加载
4
5      //相对路径加载
6      //include_once 'include1.php'; //默认当前文件本身
7
8      //include_once './include1.php';
9
10     //复杂相对路径
11     include_once '../htdocs/include1.php';
12
13
14
15     echo $a;
16

```

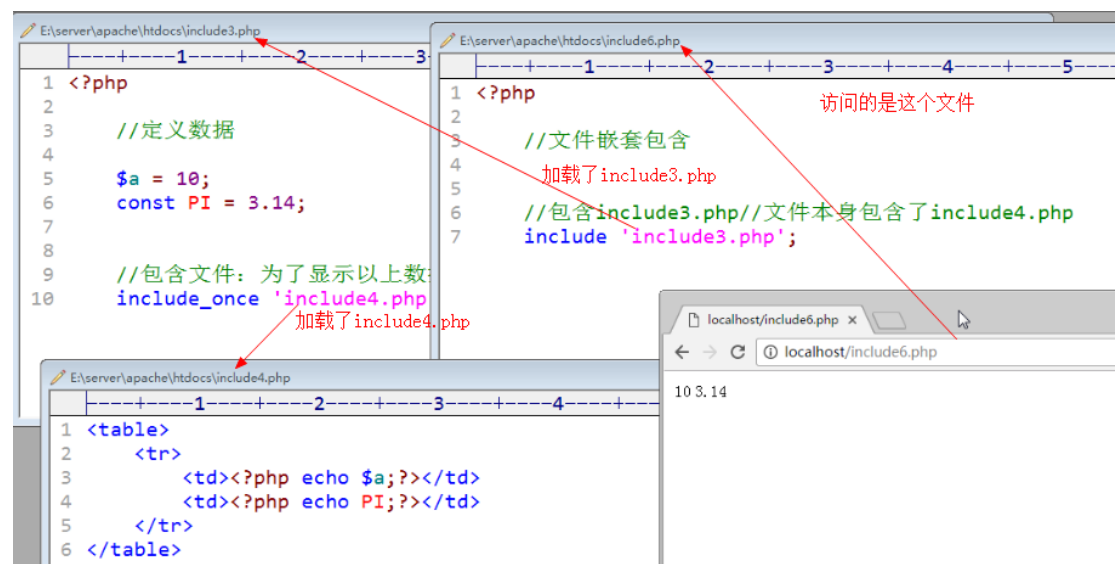


绝对路径和相对路径的加载区别

- 1、绝对路径相对效率偏低，但是相对安全（路径不会出问题）
- 2、相对路径相对效率高些，但是容易出错（相对路径会发生改变）

文件嵌套包含

文件嵌套包含：一个文件包含另外一个文件，同时被包含的文件又包含了另外一个文件。



The diagram shows three PHP files and their relationships:

- include3.php** (E:\server\apache\htdocs\include3.php):


```

1 <?php
2
3 //定义数据
4
5 $a = 10;
6 const PI = 3.14;
7
8 //包含文件: 为了显示以上数
9
10 include_once 'include4.php'

```
- include4.php** (E:\server\apache\htdocs\include4.php):


```

1 <table>
2   <tr>
3     <td><?php echo $a;?></td>
4     <td><?php echo PI;?></td>
5   </tr>
6 </table>

```
- include6.php** (E:\server\apache\htdocs\include6.php):

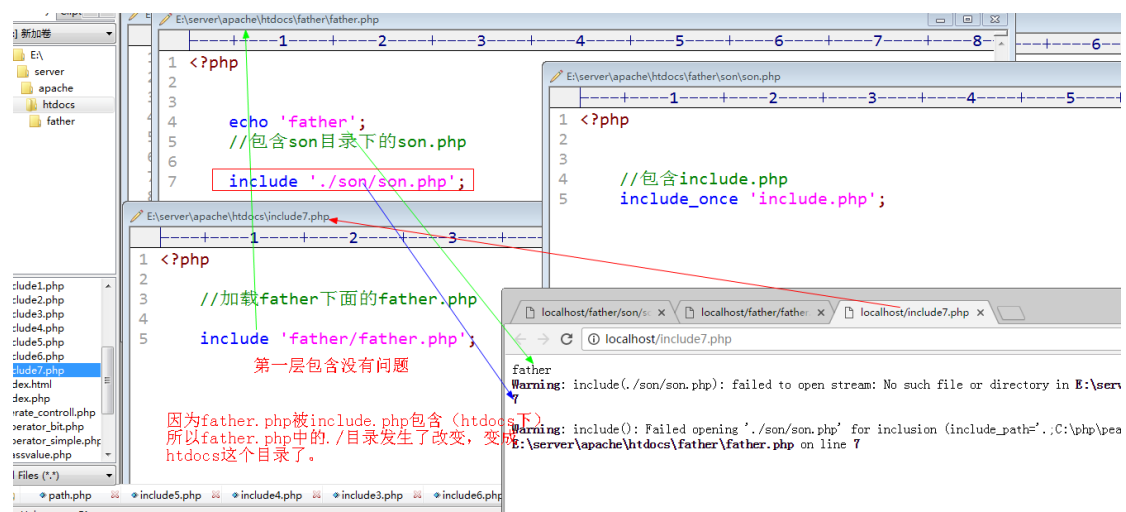

```

1 <?php
2
3 //文件嵌套包含
4 加载了include3.php
5
6 //包含include3.php//文件本身包含了include4.php
7 include 'include3.php';

```

Arrows indicate the flow of inclusion: include6.php includes include3.php, and include3.php includes include4.php. A browser screenshot at localhost/include6.php shows the output: 10 3.14.

嵌套包含的时候就很容易出现相对路径出错的问题：相对路径会因为文件的包含而改变（./和../）：windows 下面，每一个文件夹下都有./和../的文件夹。



张三 左边是 李四，李四左边是王五

张三把李四叫到自己的位置：李四与王五之间有两个位置，李四如果还按照左边伸手找王五就找不到

函数

函数的基本概念

函数：function，是一种语法结构，将实现某一个功能的代码块（多行代码）封装到一个结构中，从而实现代码的重复利用（复用）。

函数定义语法

函数有几个对应的关键点：function 关键字、函数名、参数（形参和实参）、函数体和返回值

基本语法如下：

```
Function 函数名([参数]){  
    //函数体  
    //返回值: return 结果;  
}
```

定义函数的目的：是为了实现代码的重复利用，一个功能一个函数（简单明了）

```

1 <?php
2
3 //函数
4
5
6 //定义函数
7 function display(){
8     //函数体
9     echo 'hello world'; //没有返回值
10 }

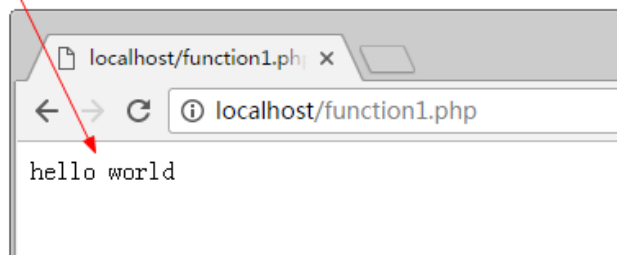
```

函数的使用：通过访问函数的名字+(); //如果函数在定义的过程中有参数，那么在调用的时候就必须传入对应的参数：函数是一种结构不会自动运行，必须通过调用才会执行

```

5
6 //定义函数
7 function display(){
8     //函数体
9     echo 'hello world'; //没有返回值
10 }
11
12
13 //调用函数
14 display();

```



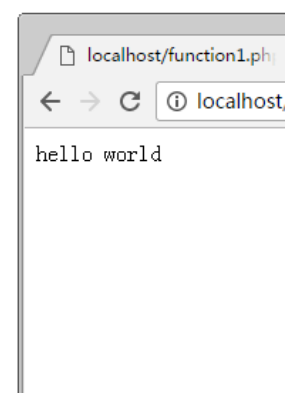
函数是在代码执行阶段，碰到函数名字的时候才会调用，不是在编译阶段。

函数的调用特点：只要系统在内存中能够找到对应的函数，就可以执行（函数的调用可以在函数定义之前）

```

3 //函数
4
5
6 //调用函数
7 display();
8
9
10 //定义函数
11 function display(){
12     //函数体
13     echo 'hello world'; //没有返回值
14 }
15
16

```



函数执行的内存分析：

- 1、 读取代码进入到代码段（编译：将代码变成字节码存储到内存）
- 2、 根据代码逐行执行

以上原因：编译和执行是分开的（先编译后执行）

函数命名规范

命名规范：由字母、数字和下划线组成，但是不能以数字开头

函数作为一种常用的结构，一般遵循以下规则：函数通常名字代表着函数的功能，而有些功能会比较复杂，可能一个单词不足以表达，需要多个组合。

- 1、 驼峰法：除了左边第一个单词外，后面所有的单词首字母都大写：`showParentInfo()`
- 2、 下划线法：单词之间通过下划线连接，单词都是小写：`show_parent_info()`

函数名字：在一个脚本周期中，不允许出现同名函数（通常在一个系统开发中都不会使用同名函数）

参数详解

函数的参数分为两种：形参和实参

形参

形参：形式参数，不具有实际意义的参数，是在函数定义时使用的参数


实参

实参：实际参数，具有实际数据意义的参数，是在函数调用时使用的参数

形参是实参的载体：实参在调用时通常是需要传入到函数内部参与计算（运算），那么需要在函数内部去找到实际数据所在的位置才能找到数据本身：需要实际调用的时候，将数据以实参的形式传递给形参：给形参赋值，从而使得函数内部可以用到外部数据。


```
3 //函数参数
4
5
6 //定义函数
7 function add($arg1,$arg2){ //形参可以有多个，使用逗号分隔即可
8     //函数体：可以直接使用形参运算
9     echo $arg1 + $arg2;
10 }
11
12 //调用函数
13 $num1 = 10;
14
15 add($num1,20); //传入的实参，可以是变量或者其他有值的表达式（变量、常量、运算符计算结果）
```

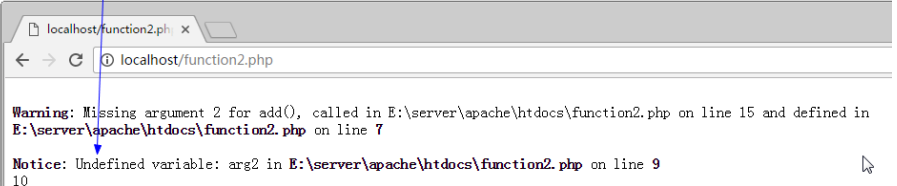
1、系统调用add函数：去内存中寻找是否有add函数：有
2、系统会在栈区开辟内存空间运行函数add
3、系统会查看函数本身是否有形参：有，两个
4、系统会判断调用的时候是否有实参：有，两个
5、系统默认会将实参的值\$num1,20取出顺序赋值给形参：\$arg1和\$arg2
6、执行函数体：运行
7、返回函数执行（返回值）



注意：

- 1、在 PHP 中允许实参多余形参（个数）：函数内部不用而已
- 2、在 PHP 中理论上形参个数没有限制（实际开发不会太多）
- 3、实参不能少于形参个数

```
5
6 //定义函数
7 function add($arg1,$arg2){ //形参可以有多个，使用逗号分隔即可
8     //函数体：可以直接使用形参运算
9     echo $arg1 + $arg2;
10 }
11
12 //调用函数
13 $num1 = 10;
14
15 add($num1); //传入的实参，可以是变量或者其他有值的表达式（变量、常量、运算符计算结果）
```




默认值

默认值：default value，指的是形参的默认值，在函数定义的时候，就给形参进行一个初始赋值；如果实际调用传入的参数（实参）没有提供，那么形参就会使用定义时的值来进入函数内部参与运算。

通常默认值是用在一些，一定会有某个数据参与，但是可能通常是某个我们知道的值。

```
12 //调用函数
13 $num1 = 10;
14
15 //add($num1,20); //传入的实参，可以是变量或者其他有值的表达式（变量、常量、运算符计算结果）
16
17 //函数的默认值
18 function jian($num1 = 0,$num2 = 0){ //当前的$num1是形参，在编译时不执行，即便执行也是在
19     //jian函数内部，不会与外部的$num1变量冲突
20     echo $num1 - $num2;
21 }
22
23 //调用：默认值如果存在，可以不用传入
24 jian($num1);
25 echo $num1;
```

\$num2没有实参传递数据值，就使用默认值



注意事项：

1、默认值的定义是放在最右边的（多个），不能左边形参有默认值，但是右边没有

函数外部定义的变量名字与函数定义的形参名字冲突（同名）是没有任何关联关系的；如果多个函数使用同样的形参名字也不冲突。

引用传递

实参在调用时会值赋值给形参，那么实际上使用的方式就是一种简单的值传递：将实参（如果是变量或者常量或者其他表达式）的结果（值）取出来赋值给形参：形参与外部实际传入的参数本身没有任何关联关系：只是结果一样。

有的时候，希望在函数内部拿到的外部数据，能够在函数内部改变，那么就需要明确告知函数（定义时），函数才会在调用的时候去主动获取外部数据的内存地址。以上这种定义形式参数的方式叫作引用传值。

基本定义语法：

```
Function 函数名(形参 1,&形参 2){  
    //函数体  
}
```

在调用的时候，必须给引用传值的参数位置传入实际参数，而且参数本身必须是变量。（变量才有指向的数据的内存地址）

```
27  
28 //引用传值  
29 function display($a,&$b){  
30     //修改形参的值  
31     $a = $a * $a;  
32     $b = $b * $b;  
33     echo $a, '<br>', $b, '<br/>';  
34 }  
35  
36  
37 //定义变量  
38 $a = 10;  
39 $b = 5;  
40  
41 //调用函数  
42 display($a,$b);  
43  
44 echo '<hr/>', $a, '<br/>', $b;
```

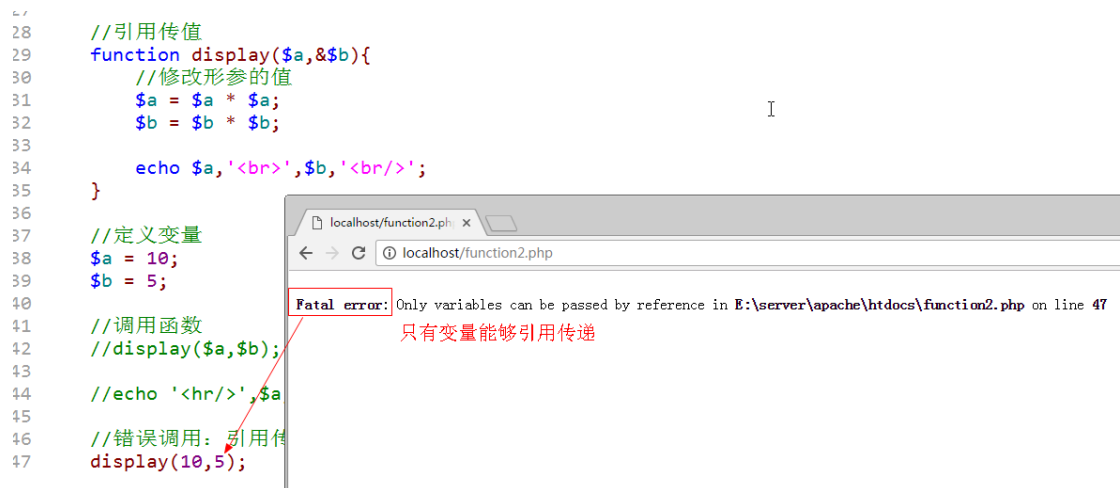
localhost/function2.php x
localhost/function2.php

100
25

10
25

说明：函数在定义的时候，对应的b形参采用的是取地址；所以在实参传入之后，系统b取到了外部变量b的内存地址，而a取的是值因此改变之后：a只改变了函数内部自己；b改变自己的同时也改变了外部。

引用传值注意事项：在传入实参的时候，必须传入变量



函数体

函数体：函数内部（大括号{}里面）的所有代码都称之为函数体

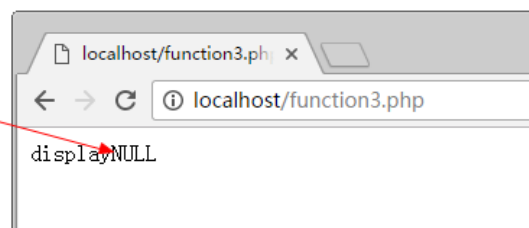
函数体：基本上所有的代码都可以实现

- 1、定义变量
- 2、定义常量
- 3、使用流程控制（分支、循环）
- 4、可以调用函数

函数返回值

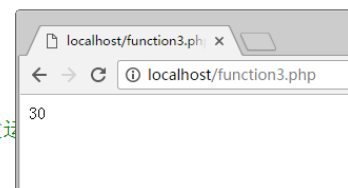
返回值：return，指的是将函数实现的结果，通过 return 关键字，返回给函数外部（函数调用处）：在 PHP 中所有的函数都有返回值。（如果没有明确 return 使用，那么系统默认返回 NULL）

```
3 //函数返回值
4
5 //定义函数
6 function display(){
7     //输出
8     echo __FUNCTION__; //输出当前函数名字
9 }
10
11 var_dump(display());
```



返回值作用：将计算结果返回给调用处

```
2
3 //加法运算
4 function add($num1,$num2){
5     return $num1 + $num2; //返回结果
6 }
7
8 $res = add(10,20); //外部定义变量接收函数返回值
9 echo $res;
```



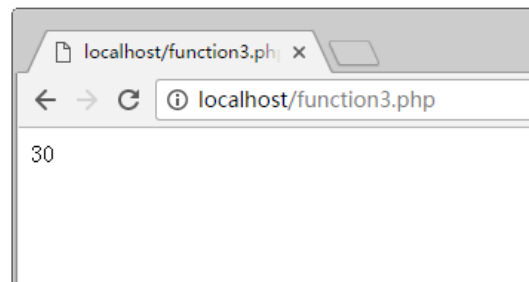
注意：函数的返回值可以是任意数据类型

Return 关键字：

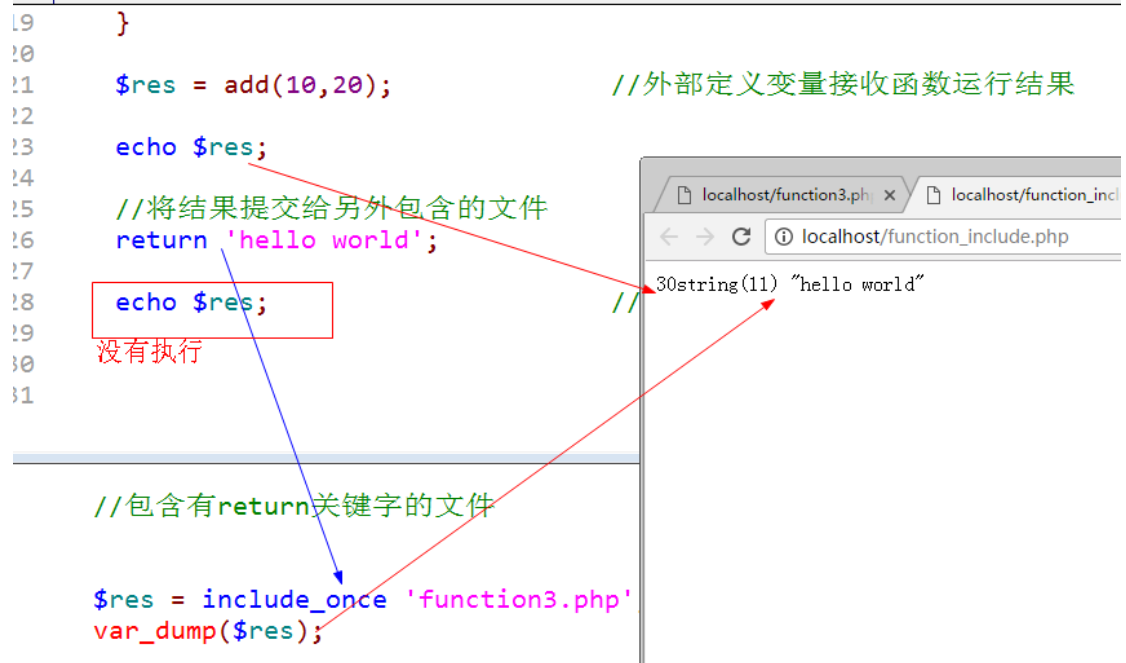
1、 return 在函数内部存在的价值：返回当前函数的结果（当前函数运行结束）

```
13 //加法运算
14 function add($num1,$num2){
15     return $num1 + $num2; //返回结果
16 }
17 //输出
18 echo $num1;
19
20
21 $res = add(10,20);
22 echo $res;
```

return 直接结束函数，所以后面所有内容都不再执行



2、 return 还可以在文件中直接使用（不在函数里面）：代表文件将结果 return 后面跟的内容，转交给包含当前文件的位置。（通常在系统配置文件中使用的较多），在文件中也代表中止文件后面的代码：return 之后的内容不会执行。



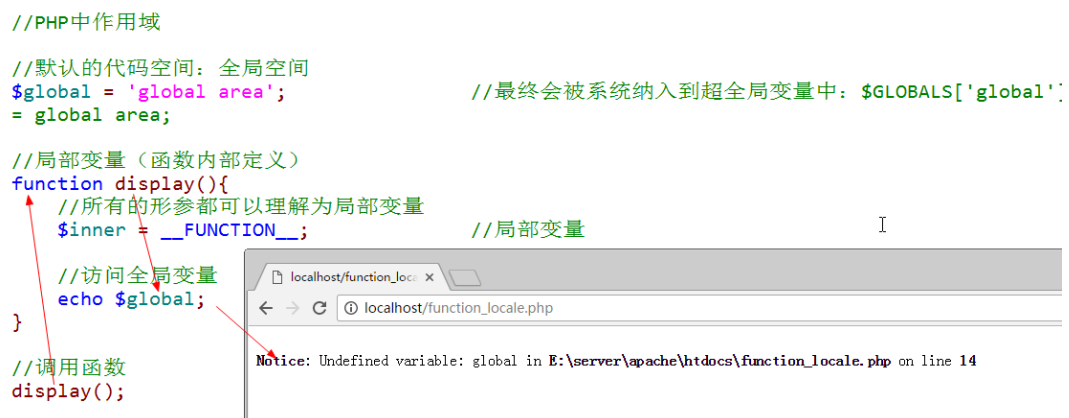
作用域

作用域：变量（常量）能够被访问的区域

- 1、 变量可以在普通代码中定义
- 2、 变量也可以在函数内部定义

在 PHP 中作用域严格来说分为两种：但是 PHP 内部还定义一些在严格意义之外的一种，所以总共算三种：

- 1、 全局变量：就是用户普通定义的变量（函数外部定义）
所属全局空间：在 PHP 中只允许在全局空间使用：理论上函数内部不可方法
脚本周期：直到脚本运行结束（最后一行代码执行完）



- 2、 局部变量：就是在函数内部定义的变量

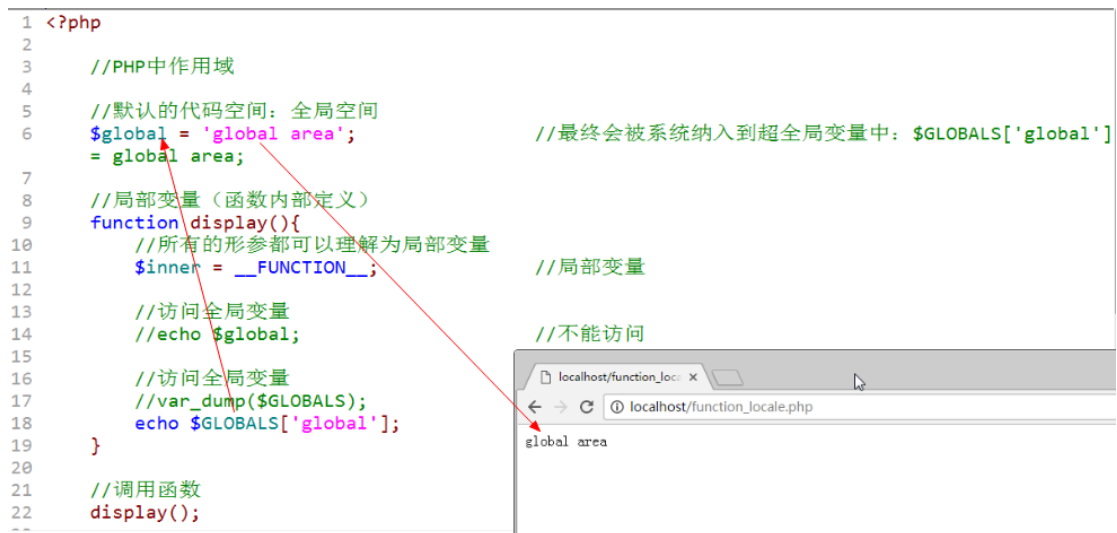
所属当前函数空间：在 PHP 中只允许在当前函数自己内部使用
函数周期：函数执行结束（函数是在栈区中开辟独立内存空间运行）



3、超全局变量：系统定义的变量（预定义变量：\$_SERVER、\$_POST 等）

所属超全局空间：没有访问限制（函数内外都可以访问）

超全局变量会将全局变量自动纳入到\$GLOBALS 里面，而\$GLOBALS 没有作用域限制，所以能够帮助局部去访问全局变量：但是必须使用数组方式



如果想函数内部使用外部变量：除了\$GLOBALS 之外，通过参数传值（如果要统一战线还可以使用引用传值）。

在 PHP 中，其实还有一种方式，能够实现全局访问局部，同时局部也可以访问全局：global 关键字

Global 关键字：是一种在函数里面定义变量的一种方式

- 1、 如果使用 global 定义的变量名在外部存在（全局变量），那么系统在函数内部定义的变量直接指向外部全局变量所指向的内存空间（同一个变量）；
- 2、 如果使用 global 定义的变量名在外部不存在（全局变量），系统会自动在全局空间（外部）定义一个与局部变量同名的全局变量

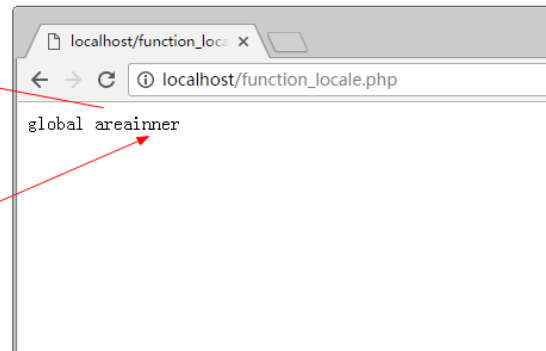
本质的形式：在函数的内部和外部，对一个同名变量（全局和局部）使用同一块内存地址保存数据，从而实现共同拥有。

基本语法：

Global 变量名; //不能赋值

变量名 = 值; //修改

```
3
3
1 //定义变量：使用全局变量
2 global $global; //全局空间存在
3 echo $global;
4
5 //定义变量：全局不存在
5 global $local;
7 $local = 'inner';
3 }
3
3 //调用函数
1 display();
2
3 //全局空间访问局部变量
4 //echo $inner;
5
5 //访问“局部”变量
7 echo $local;
```



虽然以上方式可以实现局部与全局的互访，但是通常不会这么用。一般如果会存在特殊使用，也会使用参数的形式来访问。（还可以使用常量：define 定义的）

静态变量

静态变量：static，是在函数内部定义的变量，使用 static 关键字修饰，用来实现跨函数共享数据的变量：函数运行结束所有局部变量都会清空，如果重新运行一下函数，所有的局部变量又会重新初始化。

基本语法：

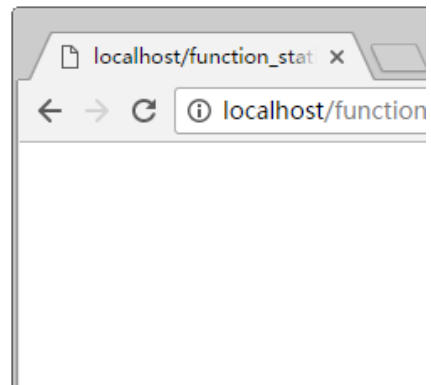
```
Function 函数名(){
    //定义变量
    Static $变量名 = 值; //通常会在定义的时候就直接赋值
}
```

```
//静态变量

//定义函数
function display(){
    //定义变量
    $local = 'local';

    //定义静态变量
    static $count = 1;
}

//调用
display();
```



静态变量的作用是为了跨函数共享数据（同一个函数被多次调用）

```
//静态变量

//定义函数
function display(){
    //定义变量
    $local = 1;

    //定义静态变量
    static $count = 1;

    echo $local++, $count++, '<br/>';
}

//调用
display();
display();
display();
```

静态变量的原理：系统在进行编译的时候就会对static这一行进行初始化：为静态变量赋值

//局部变量

//静态变量

11
12
13

函数在调用的时候，会自动跳过static关键字这一行



静态变量的使用：

- 1、 为了统计：当前函数被调用的次数（有没有替代方法？）
- 2、 为了统筹函数多次调用得到的不同结果（递归思想）

可变函数

可变函数：当前有一个变量所保存到值，刚好是一个函数的名字，那么就可以使用变量+()来充当函数名使用。

```
$变量 = 'display';
Function display(){

}
```

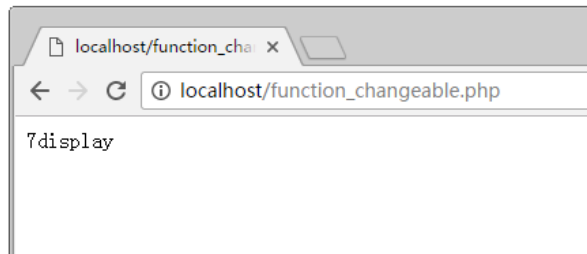
```
//可变函数
$变量();
```



```

3 //PHP可变函数
4
5 //定义函数
6 function display(){
7     echo __LINE__, __FUNCTION__;
8 }
9
10 //定义变量
11 $func = 'display';
12
13 //可变函数
14 $func();
15

```



可变函数在系统使用的过程中还是比较多的，尤其是使用很多系统函数的时候：需要用户在外定义一个自定义函数，但是是需要传入到系统函数内部使用。

```

//定义系统函数（假设）
function sys_function($arg1,$arg2){
    //给指定的函数（第一个参数），求对应的第二个参数值的4次方
    //为实际用户输入的数值进行处理
    $arg2 = $arg2 + 10;

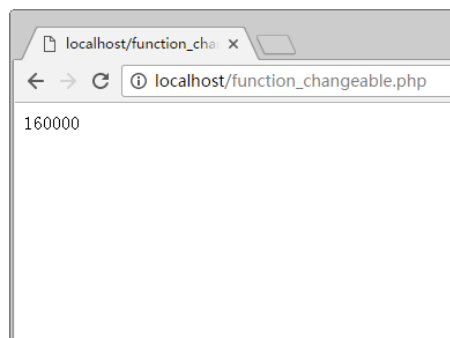
    return $arg1($arg2);    //user_function(20)
}

//定义一个用户函数：求一个数的四次方
function user_function($num){
    return $num * $num * $num * $num;
}

//求10的4次方
echo sys_function('user_function',10);

```

将一个用户定义的函数传入给另外一个函数（函数名）去使用的过程，称之为回调过程，而被传入的函数称之为回调函数



匿名函数

基本概念

匿名函数：没有名字的函数

基本语法：

```

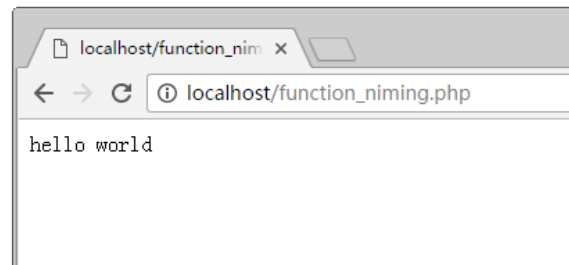
变量名 = Function(){
    函数体
};

```

```

4
5 //定义基本匿名函数
6 $func = function(){
7     //函数体
8     echo 'hello world';
9 };
10
11 //调用匿名函数：可变函数
12 $func();

```

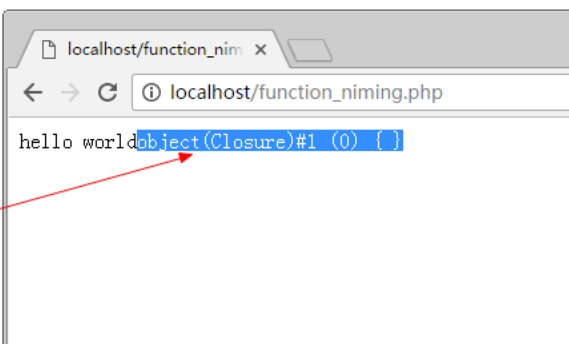


变量保存匿名函数，本质得到的是一个对象（Closure）

```

1 //定义基本匿名函数
2 $func = function(){
3     //函数体
4     echo 'hello world';
5 };
6
7 //调用匿名函数：可变函数
8 $func();
9
10 //查看变量内容
11 var_dump($func);

```



闭包

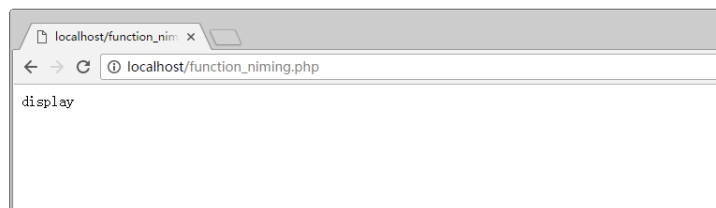
闭包：closure，一词来源于以下两者的结合：要执行的代码块（由于自由变量被包含在代码块中，这些自由变量以及它们引用的对象没有被释放）和为自由变量提供绑定的计算环境（作用域）

简单理解：函数内部有一些局部变量（要执行的代码块）在函数执行之后没有被释放，是因为在函数内部还有对应的函数在引用（函数的内部函数：匿名函数）

```

7 //闭包函数
8 function display(){
9     //定义变量：局部变量
10     $name = '__FUNCTION__';
11
12     //定义匿名函数
13     $innerfunction = function() use($name){ //use就是将外部变量（局部）保留给内部使用（闭包）
14         //函数内部的函数
15         echo $name;
16     };
17
18     //调用函数
19     $innerfunction();
20 }
21
22 display();

```



证明：函数的局部变量在函数使用完之后没有被释放？

- 1、使用内部匿名函数；
- 2、匿名函数使用句变量：use；
- 3、匿名函数被返回给外部使用；

```
34 //内包函数
35 function display1(){
36     //定义变量：局部变量
37     $name = __FUNCTION__;
38
39     //定义匿名函数
40     $innerfunction = function() use($name){ //use就是将外部变量（局部）保留给内部使用（闭包）
41         //函数内部的函数
42         echo $name;
43     };
44
45     //返回内部匿名函数
46     return $innerfunction;
47 }
48
49 $closure = display1();
50
51 //到51行位置：display1函数运行结束：理论上局部变量$name应该
52 $closure();
```

当前局部变量\$name在49行display1函数运行结束后并没有被释放，从而在外部调用内部匿名函数的时候可以被使用

localhost/function_nim X
localhost/function_niming.php
display1

伪类型

伪类型：假类型，实际上在 PHP 中不存在的类型。但是通过伪类型可以帮助程序员去更好的查看操作手册从而更方便学习。

伪类型主要有两种：在三大类八小类之外

Mixed：混合的，可以是多种 PHP 中的数据类型

Number：数值的，可以是任意数值类型（整形和浮点型）

string **gettype** (mixed *\$var*)

返回 PHP 变量的类型 *var*。

只要是PHP中对应的类型即可

Warning

不要使用 **gettype()** 来测试某种类型，因为其返回的字符串在未来的版本中可能需要改变的比较，它的运行也是较慢的。

使用 **is_*** 函数代替。

返回的字符串的可能值为：

- **"boolean"** (从 PHP 4 起)
- **"integer"**
- **"double"** (由于历史原因，如果是 **float** 则返回 "double"，而不是 "float")
- **"string"**
- **"array"**
- **"object"**
- **"resource"** (从 PHP 4 起)
- **"NULL"** (从 PHP 4 起)
- **"user function"** (只用于 PHP 3，现已停用)
- **"unknown tvne"**