昨日回顾 2

数组基础 3

数组基础 3

数组的分类 4

从键值关系： 4

从数组层次来分： 4

数组遍历 5

遍历基本语法 5

数组指针和遍历原理 5

数组遍历的流程图： 6

for+next+reset遍历数组 6

while+each()+list()遍历数组 6

foreach 遍历细节探讨 8

数组排序思想介绍 9

冒泡排序 9

选择排序 11

昨日回顾

函数基本使用

函数定义形式: function func1(形参1，形参2， .....){ 。。。。。 }

函数调用形式:

如没有返回值：func1(实参1，实参2， .....)；

如有返回值：通常我们都将调用语句当作一个“数据”来使用。

$v1 = func1(实参1，实参2， .....)；

$v1 = func1(实参1，实参2， .....) \* 5 + 3 + f2() / 3 ;

echo func1(实参1，实参2， .....);

echo func1(实参1，实参2， .....) \* 5 + 3;

也可以作为“实参”使用。

函数参数问题

在定义时，有形式参数（形参）：形参就是函数内部可用的局部变量而已。

在调用时，有实际参数（实参）：实参就是一个数据。

默认值参数：

function func1(形参1，形参2， .....，形参n1=值1， 形参n2=值2， ...... ){。。。。}

函数的参数传值问题

默认都是使用“值传递”——跟变量之间的传值方式一样。

可以在形参的前面加上引用符号“＆”，则可以实现“引用传递”——跟变量一样。

**注意：如果一个形参标识为引用传递，则实参只能使用变量了，而不能使用“直接值”**

参数的数量问题：

实参的通常应该跟形参“一一对应”。

如果有默认值形参，则可以省略对应默认值形参，但只能从右往左省略。

函数的返回值

默认情况下，使用的是“值传递”——返回的是数据本身，而不是对数据的引用。

但：

可以设定为“引用返回”，形式为：

function &func1(形参1，形参2， .....){ 。。。。。 }//这里肯定表示有返回值。

此时，调用函数也得加上引用符号：

$v1 = &func1(实参1，实参2， .....)；

函数的其他形式

1可变函数：函数名可以使用一个变量来代替，形成类似可变变量的“识别规则”

$f1 = “getArea”; //

$f1(); //这就是可变函数，实际调用的是: getArea()

2匿名函数: 就是定义函数的时候，没有给出名字的函数，通常有两种情况：

2.1: $f1 = function(){....}; //此时虽然函数没有名字，但其实f1也就当作它的名字来用，类似可变函数

2.2: 别的函数f2(匿名函数定义callback， 实参2，实参3，..... );

callback：代表这里的匿名函数，它也常常被称为“回调函数”——是指在这种场合下使用，就这样称呼。也叫做可调函数（callable）

变量的作用域：

局部：

静态局部：

全局：

超全局：

有关函数的编程思想

递归思想（递归函数）：

从大问题出发，逐级找出小一级问题的答案，直到最小问题，再逐级从小到大，返回每级问题的答案以得到最终问题的答案。

递推思想（迭代思想）

从最小一级问题出发，逐级往大方向找出大一级的问题的答案，直到最大问题（最终问题）的答案。

数组基础

数组基础

php中，数组的下标可以是整数，或字符串。

php中，数组的元素顺序不是由下标决定，而是由其“加入”的顺序决定。

定义：  
 $arr1 = array(元素1，元素2，。。。。。 );

array(1, 5, 1.1, “abc”, true, false); //可以存储任何数据，此时为“默认下标”，

array(2=>1, 5=>5, 3=>1.1, 7=>“abc”, 0=>true);//下标可以任意设定（无需顺序，无需连续）

array(2=>1, 5, 1=>1.1, “abc”, 0=>true)//可以加下标，也可以不加（默认下标）,下标分别是：2,3,1,4,0

//默认下标规则：前面已经用过的最大数字下标+1

array(2=>1, ‘dd’=>5, 1=>1.1, “abc”, 0=>true)//混合下标，同样遵循默认下标规则

array(-2=>1, ‘dd’=>5, 1.1, “abc”, true); //负数下标不算在整数下标中，而只当作字符下标

//则最好3项的下标是：0, 1, 2

array(2.7=>1, ‘dd’=>5, 1=>1.1, “abc”, 0=>true)；//浮点数下标为自动转换为整数，且直接抹掉小数

array(“2.7” =>1, ‘dd’=>5, “11”=>1.1, “abc”, true)//纯数字字符串下标，当作数字看待，

//则此时下标为：2, ‘dd’, 11, 12, 13

array(2=>1, ‘dd’=>5, true=>1.1, “abc”, false=>true)//布尔值当下标，则true为1，false为0；

array(2=>1, ‘dd’=>5, 2=>1.1, “abc”, true)//如果下标跟前面的重复，则单纯覆盖前面同名下标的值

//此时相当于为：array(2=>1.1, ‘dd’=>5, “abc”, true)

其他形式；

$arr1[] = 1;

$arr1[] = 5;

$arr1[] = 1.1; //直接在变量后面使用[]，就成为数组，并依次赋值。

。。。。

$arr2[‘aa’] = 1;

$arr2[‘bbbcc’] = 5;

$arrr2[5] = 1.1;

。。。。。。。。

这种形式写的下标，其实跟使用array语法结构几乎一样。

取值：通过下标。

赋值（同定义）：

数组的分类

从键值关系：

关联数组：通常是指下标为字符串，并且该字符串大体可以表达出数据的含义的数组。

例：$person = array(

“name” => “小花”,

“age”=>18,

“edu” => “大学毕业” ，

);

索引数组：

通常是指一个数组的下标是严格的从0开始的连续的数字下标——跟js数组一样。

从数组层次来分：

一维数组：

就是一个数组中的每一个元素值，都是一个普通值（非数组值）

$arr1 = array(

“name” => “小花”,

“age”=>18,

“edu” => “大学毕业”

);

二维数组：

一个数组中的每一项，又是一个一维数组。

$arr1 = array(

“name” => array(‘小花’, ‘小芳’, ‘小明’, );

“age”=> array(18, 22, 19),

“edu” => array(“大学毕业”, ‘中学’, ‘小学’)

);

多维数组：

依此类推。。。

多维数组的一般语法形式：

$v1 = 数组名[下标][下标][.....]

数组遍历

遍历基本语法

foreach( $arr as [ $key => ] $value ) //$key可以称为键变量，$value可以称为值变量。

{

//这里就可以对$key 和 $value 进行所有可能的操作——因为他们就是一个变量

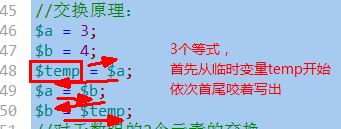
//$key 代表每次取得元素的下标，可能是数字，也可以能是字符串

//$value 代表每次取得元素的值，可能是各种类型。

//此循环结构会从数组的第一项一直遍历循环到最后一项，然后结束。

}

交换原理：



数组指针和遍历原理

每个数组，其内部都有一个“指针”，该指针决定了该数组当前取值的时候，取到的元素。

foreach遍历过程中，都是依赖与该指针而进行的！

举例：$arr1 = array(2=>1, ‘dd’=>5, 1=>1.1, “abc”, 0=>true)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 下标： | 2 | ‘dd’ | 1 | 3 | 0 |
| 值： | 1 | 5 | 1.1 | ‘abc’ | true |

指针除了负责foreach循环的位置设定之外，还有其他一些函数也依赖于该指针：

1， $v1 = current( $arr1); //取得$arr1中当前指针所指向的元素的值，如果没有指向元素，则为false

2， $v1 = key（$arr1）； //取得$arr1中当前指针所指向的元素的下标，。。。。。false

3， $v1 = next($arr1 )； //将指针移向“下一个元素”，然后取得该下一个元素的值；

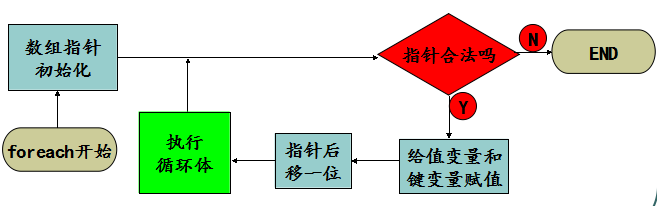
4， $v1 = prev($arr1)； //将指针移向“上一个元素”，然后取得该上一个元素的值

5， $v1 = reset($arr1）； //将指针移向“第一个元素”，然后取得该元素的值——数组指针初始化

6， $v1 = end($arr1）； //将指针移向“最后一个元素”，然后取得该元素的值

7， $v1 = each($arr1)； //取得当前元素的下标和值，然后移动指针到下一个位置。

数组遍历的流程图：



for+next+reset遍历数组

reset( $arr1 ); //重置数组，即：数组指针初始化，这里，返回的数据被“丢弃”了。

$len = count( $arr1);

for（$i = 0; $i < $len $i++){

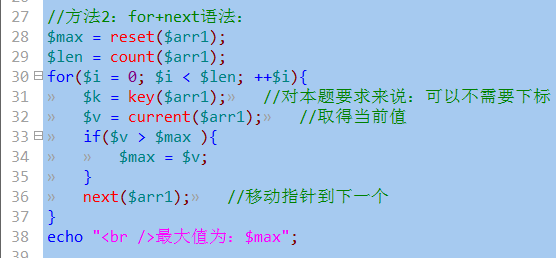
$key = key ($arr1 ); //下标

$value = current( $arr1 ); //值

//然后这里就可以对$key 和 $value进行任何作为变量的操作

next($arr1）；//这里，移动指针到下一个元素（也同时丢弃了返回值）

}



while+each()+list()遍历数组

each（）函数解释：

each()函数可以取得一个数组中的一个元素的下标和值，然后再放入一个新的数组中，并且指针后移一位。

该新的数组，有4个元素，但存储的是下标和值的“双份”，类似下述形式：

array(

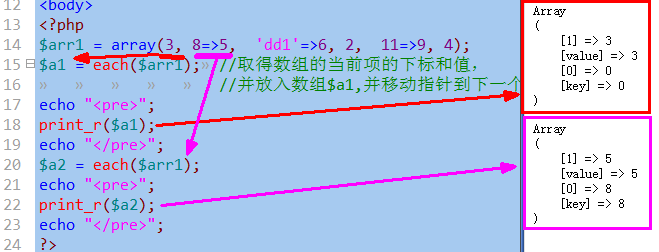
1 => 取出来的值，

‘value’ = >取出来的值，

0 = > 取出来的下标（键名），

‘key’ => 取出来的下标（键名）

);



list()函数解释：

使用形式：

list($v1, $v2, $v3,$v4 .... ）= 数组$arr1;

其作用是：依次取得数组$arr1中下标为0，1，2，3, ....的元素的值，并一次性放入多个变量中（一一对应）

即其相当于如下语句：

$v1 = $arr1[0];

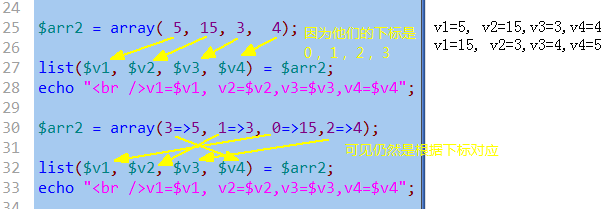
$v2 = $arr1[1];

$v3 = $arr1[2];

$v4 = $arr1[3];

..........

但是注意：只能实现这样的“从0开始的连续数字下标的元素的取值”（但并非要求数组的元素的顺序为同样的数字顺序）



然后开始使用这2个函数和while循环结构来实现数组遍历：

形式：

reset($arr1);

while ( list ($key, $value ) = each( $arr1) ) //从数组$arr1中一次次取出元素，

//当each到数组最后的时候，就返回false，即此时循环结束

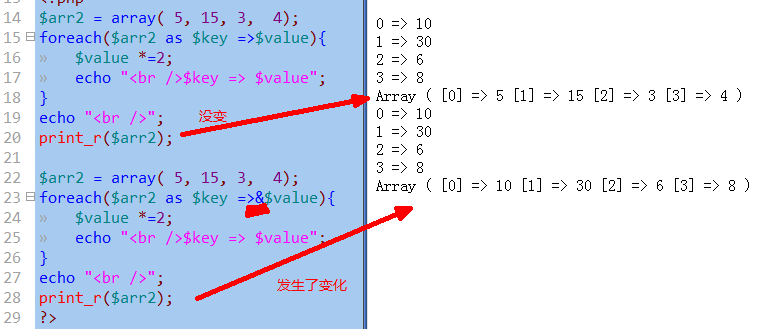
{

//这里，就可以对$key ,和$value进行操作了。

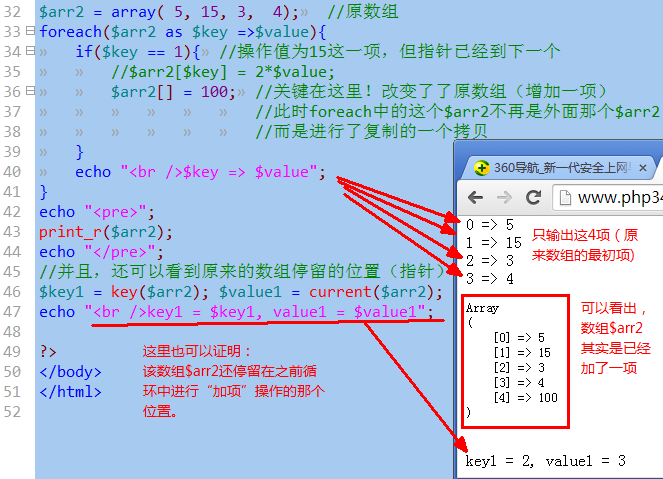
}

foreach 遍历细节探讨

* foreach也是正常的循环语法结构，可以有break和continue等操作。
* 遍历过程中值变量默认的传值方式是值传递。
* 遍历过程中值变量可以人为设定为引用传递：
  + foreach($arr as $key => &$value){ ... }
  + 键变量不可以设定为引用传递



* foreach默认是原数组上进行遍历。但如果在遍历过程中对数组进行了某种修改或某种指针性操作（就是指前面的指针函数），则会复制数组后在复制的数组上继续遍历循环。



* foreach中如果值变量是引用传递，则无论如何都是在原数组上进行。

数组排序思想介绍

参考：

http://zh.wikipedia.org/wiki/冒泡排序

冒泡排序

目标：将下列数组进行正序（从小到大）排列出来

$arr2 = array( 5, 15, 3, 4， 9， 11);

一般性逻辑描述：

１，对该数组从第一个元素开始，从左到右，相邻的２个元素比较大小：如果左边的比右边的大，则将他们俩交换位置，结果：

array( 5, 15, 3, 4， 9， 11);（原始）

array( **5, 15**, 3, 4， 9， 11);

array( 5, **3， 15**, 4， 9， 11);

array( 5, 3， **4, 15**， 9， 11);

array( 5, 3， 4, **9， 15**， 11);

array( 5, 3， 4, 9， **11， 15**);

此时，才“走完一轮回”，继续下一轮：

array( 5, 3， 4, 9， 11， 15);（初始）

array( **3 5**， 4, 9， 11， 15);

array( 3 **4， 5** 9， 11， 15);

array( 3 4， **5 9**， 11， 15);

array( 3 4， 5 **9， 11**， 15);

继续下一轮：

array( 3 4， 5 **9， 11**， 15);

。。。。。。。。

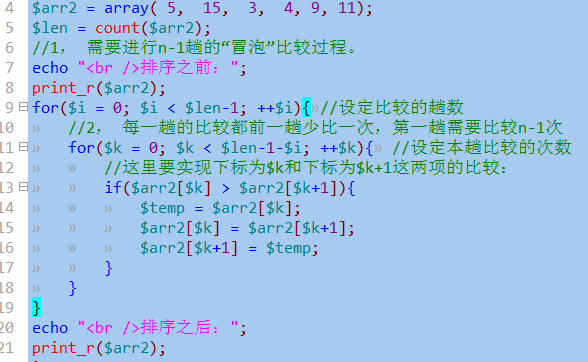
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 最初： | 5 | 15 | 3 | 4 | 9 | 11 |
| 第1趟之后： | 5 | 3 | 4 | 9 | 11 | 15 |
| 第2趟之后 | 3 | 4 | 5 | 9 | 11 | 15 |
| 第3趟之后 | 3 | 4 | 5 | 9 | 11 | 15 |
| 第4趟之后 | 3 | 4 | 5 | 9 | 11 | 15 |
| 第5趟之后 | 3 | 4 | 5 | 9 | 11 | 15 |

隐含的逻辑描述（假设数组有n项）：

1， 需要进行n-1趟的“冒泡”比较过程。

2， 每一趟的比较都前一趟少比一次，第一趟需要比较n-1次

3， 每趟比较，都是从数组的开头（0）开始，跟紧挨的元素比较，并进行交换（需要的时候）



选择排序

目标：将下列数组进行正序（从小到大）排列出来

$arr2 = array( 5, 15, 3, 4， 9， 11);

一般性逻辑描述：

第1趟：取得该数组中的最大值及其下标，然后跟该数组的最后一项“交换”（倒数第1项确定）

第2趟：取得该数组中除最后1项中的最大值及其下标，然后跟倒数第2项交换（倒数第2项确定）

第3趟：取得该数组中除最后2项中的最大值及其下标，然后跟倒数第3项交换（倒数第3项确定）

。。。。。。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 最初： | 5 | 15 | 3 | 4 | 9 | 11 |
| 第1趟之后： | 5 | 11 | 3 | 4 | 9 | 15 |
| 第2趟之后 | 5 | 9 | 3 | 4 | 11 | 15 |
| 第3趟之后 | 5 | 4 | 3 | 9 | 11 | 15 |
| 第4趟之后 | 3 | 4 | 5 | 9 | 11 | 15 |
| 第5趟之后 | 3 | 4 | 5 | 9 | 11 | 15 |

隐含的逻辑描述（假设数组有n项）：

1，要进行n-1趟才可能得出结论

2，每一趟要找的数据的个数都比前一趟少一个，第1趟要找n个

3，每次找出的最大值所在的项，和要与之进行交换的项的位置，依次减1，第一次的位置n-1

