昨日回顾

函数

函数名 命名规范(函数名不能重名,函数名不区分大小写[小驼峰])

可变函数 – 函数名可以作为一个变量的值

匿名函数 – 就是没有函数名的函数

参数列表 形参和实参.

参数值传递的形式 - 值传递和引用传递

默认值

参数的个数问题

函数体 return

变量的作用域

局部作用域 和 全局作用域

超全局变量$GLOBALS

变量的生命周期

static 关键字

static 可以延长 局部作用域的生命

递归调用

概念: 自己调用自己

2个要素:

递归从什么时候开始

递归从什么时候结束

阶乘

1! = 1;

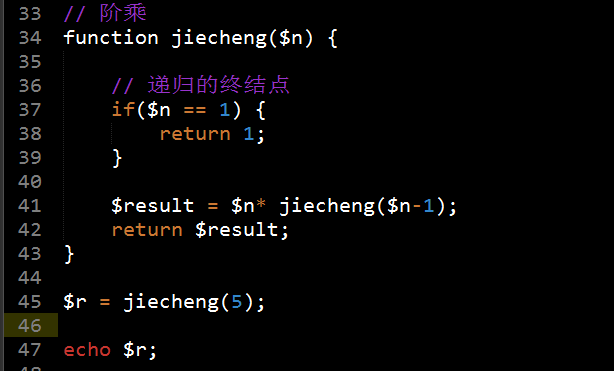
2!=2\*1; = 2\*1!;

3!=3\*2\*1; = 3\*2!

4!=4\*3\*2\*1; 4\*3!

…

n! = (n)\*(n-1)\*(n-2)…\*4\*3\*2\*1;



对数组进行遍历(把对应的值获取到就可以了).

# 第53讲 迭代思想

所谓的迭代，就是从前往后来解决问题！

迭代：就是将一个步骤重复的执行！

1 1 2 3 5 8 13 21 34 55 …

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  | 1 | 1 | 2 | 3 | 5 |  |  |  |  |
| 第3项 | Frist | second | Sum = first+second |  |  |  |  |  |  |
| 第4项 |  | first | second | Sum=first+second |  |  |  |  |  |
| 5 |  |  | First | Second | Sum= first+second |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |

第一步: 多去读题目或者 需求,把一些东西都读出来

第二步: 将话术写出来,形成思路

第三步: 将思路转换成代码. 再把思路转换成代码的过程中我们需要不断的以真实的数据作为校验依据.

以下都是思路:

① first = 1; second =1;

② 第三项 = first + second = sum;

③ 求第四项 first = 前一项的 second,

second = sum

sum = first + second

④ 求第五项 first = 前一项的 second

second = sum;

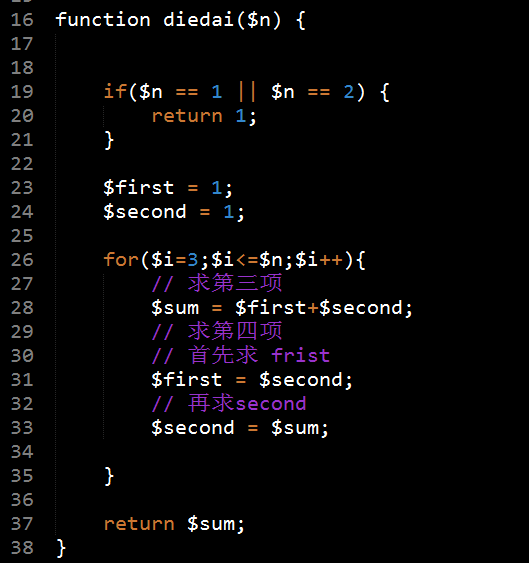
sum = first+second

⑤ 从第3项开始就是一个重复 for ;

求第三项重复1次

求第四项重复2次

求第五项重复3次



# 第54讲 数组初步

## 数组的组成

数组有点类似于数学上的集合！

数组，是由元素（element）组成的，每个元素都是数组中的一个成员！

每一个元素又有两部分构成：

元素名字(字符串和整形)：也叫作键（key） - 字符串，还叫作下标（index）-- 整形

元素的值：也叫作值（value）

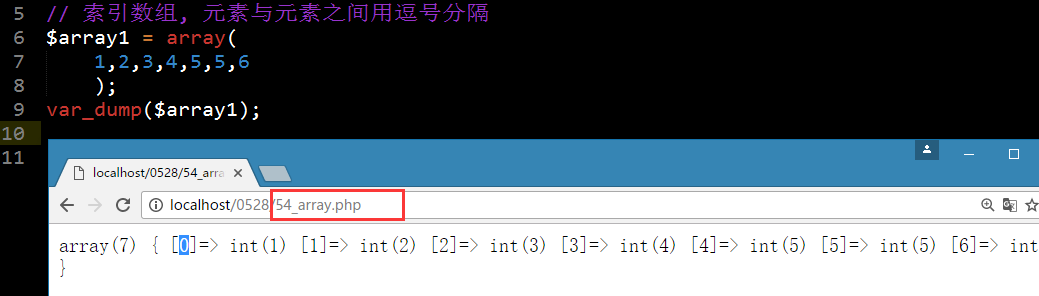
所以，我们通常也把一个元素叫作一个“键值对”（key/value）

## 创建数组

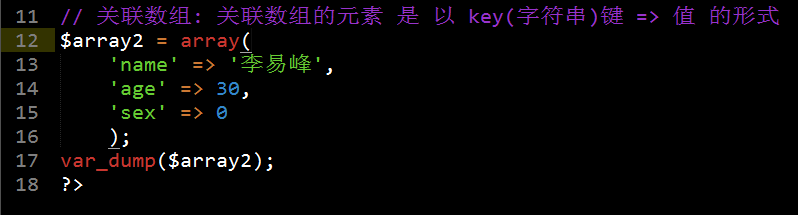
### 显示的创建

显示的创建数组是采用系统的函数 array() 这个函数

索引数组

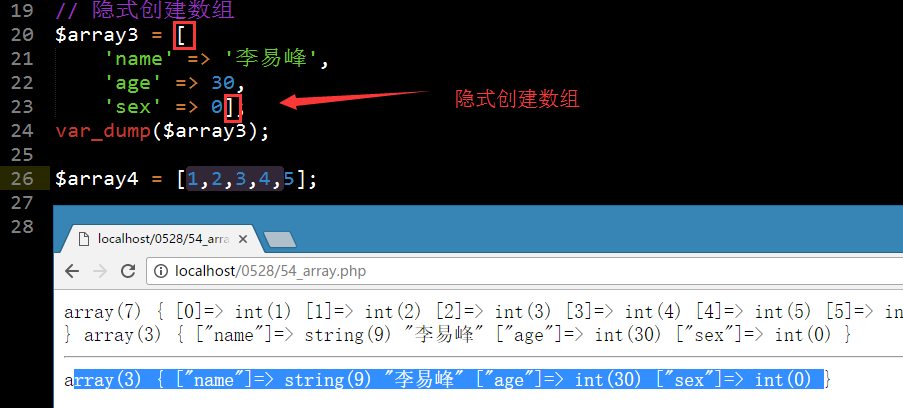


关联数组: 关联数组是以 键(key) => 值 的形式



### 隐式的创建

隐式创建数组是采用 中括号的形式[] ;



给数组添加数据 是采用

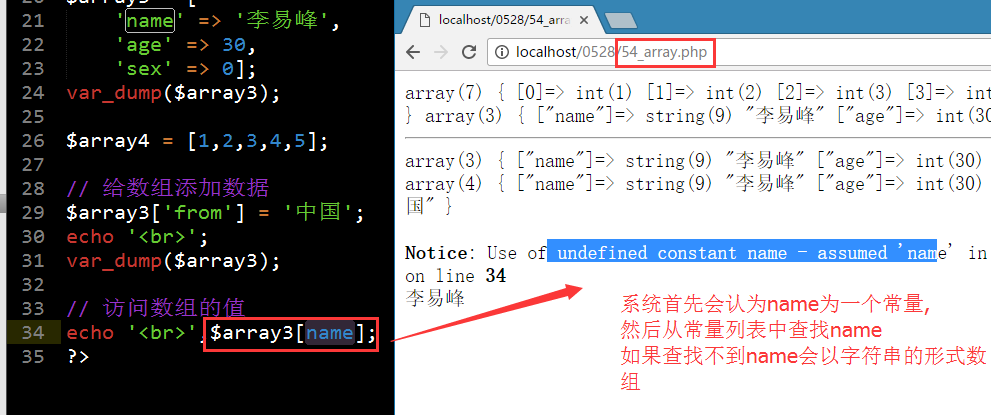
$数组名[‘名’] = 值 的形式

## 访问数组

就是利用数组名和元素的下标组合到一起来访问，其实就是利用中括号语法：

**数组名[下标]；**

注意：如果下标是一个字符串，不能省略掉下标中的引号！



# 第55讲 数组的分类

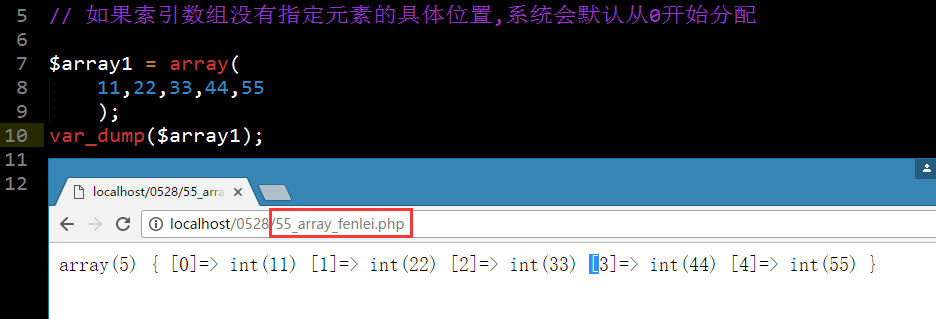
## 按照键值之间的对应关系

### 索引数组

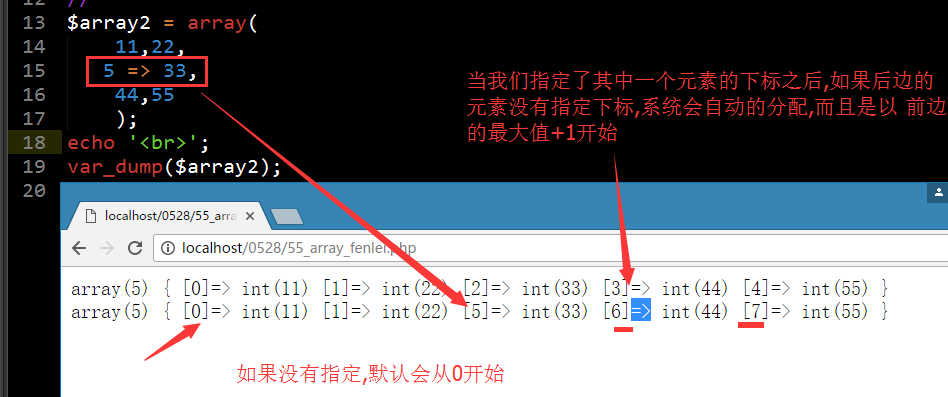
数组元素的键，不用于表示其对应的值的含义，而仅仅表示其在数组中所在的位置！

需要注意以下的几点：

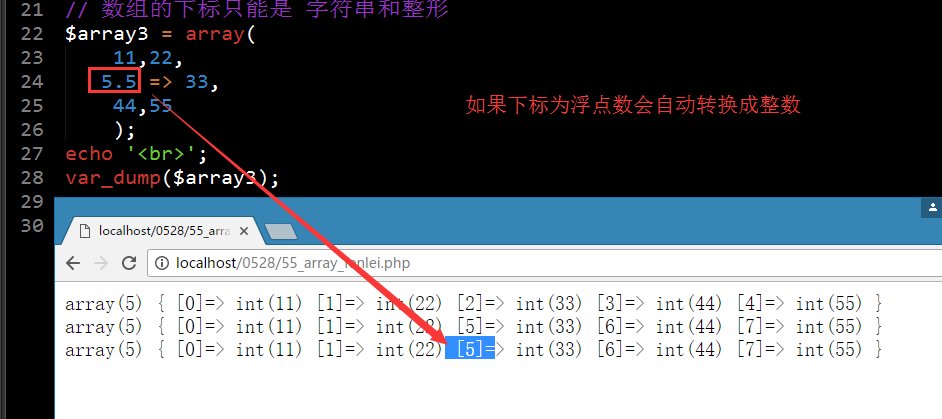
* 典型的，当一个数组的下标全部都省略的时候，系统会自动给元素分配下标值，从0开始，依次递增！

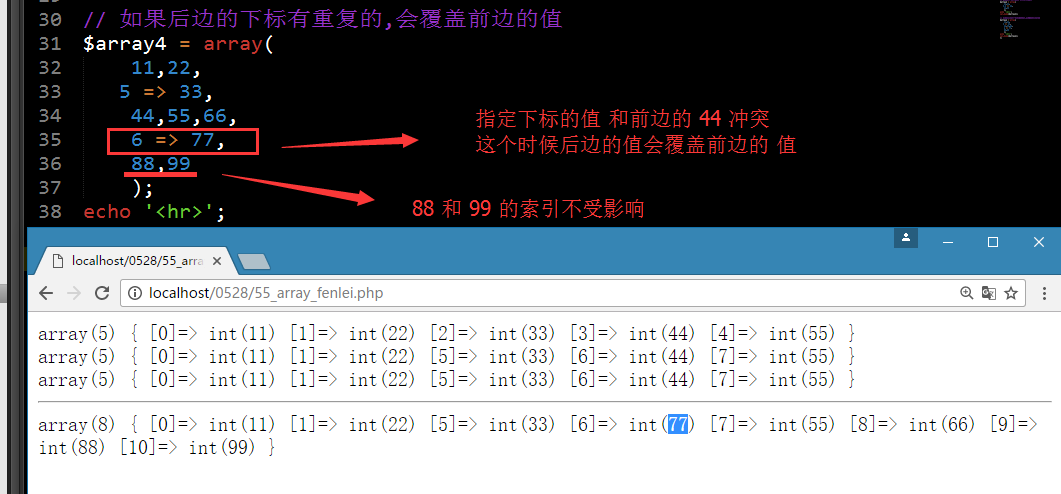


* 当省略其中的部分元素的下标的时候，系统也会自动的给这些元素分配下标，并且下标值是前面已经存在的最大整型的下标加1，如果没有，就从0开始！



* 默认的，数组的下标只能是字符串型和整型，虽然语法上允许写其他的类型，但是在创建数组的时候会自动转换成整型或字符串型

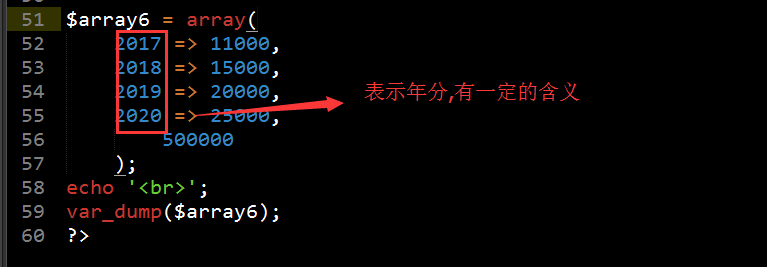


* 如果两个元素的下标值相同,后面的覆盖前面的
* 

### 关联数组

指数组元素的键和值之间，存在着某种对应关系，也就是说键的值是有意义的！

* 典型的，关联数组一般都是以字符串作为下标！
* 并不是说，关联数组的下标就一定是字符串型，最关键的是看元素的键与值之间有没有对应关系！

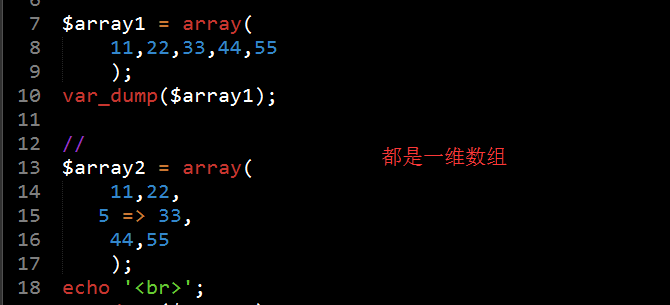


* 有时候，一个数组的下标既可以有整型数据也可以有字符串型数据，称之为“混合数组”

## 按照数组的维度

### 一维数组

简单来说，一维数组就是指该数组的所有的元素的值，都不是数组类型！



### 多维数组

超过一维的都可以叫多维数组，是指数组中的元素有部分或者全部又是一个数组！

元素的值还是数组



# 第56讲 数组的foreach遍历

遍历，就是依次得到并处理数组的每一个元素！

## 基本语法

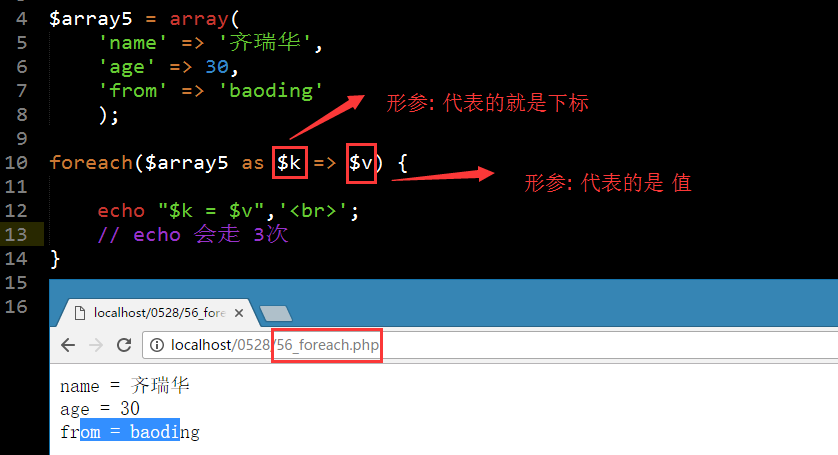
foreach($数组名 as $key => $value) {

}

简写形式:

foreach($数组名 as $value) {

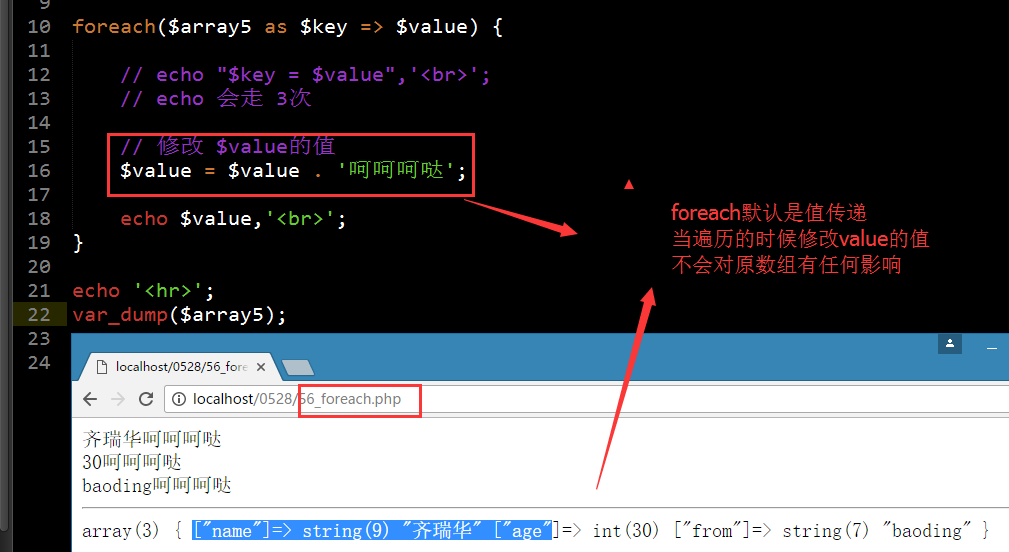
}



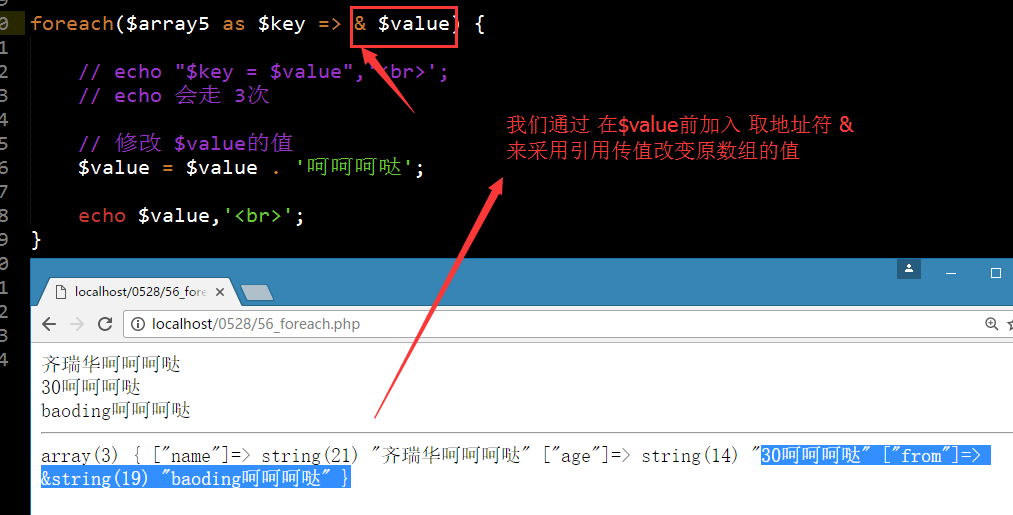
## 几个细节

① foreach遍历的时候,默认是对数组进行 拷贝 – 值传递.

既然是值传递,那么我们在foreach中修改 $value的值,会对原来的数组有影响吗? 没有!!!



② 我们可以通过 foreach($数组名 as $key => & $value) 这种形式 – 引用传值的形式来获取 $value的地址,来改变原数组的值



③ 我们不能通过任何形式来改变 $key的值



# 第57讲 数组的指针

数组名在栈区,数组的元素在堆区

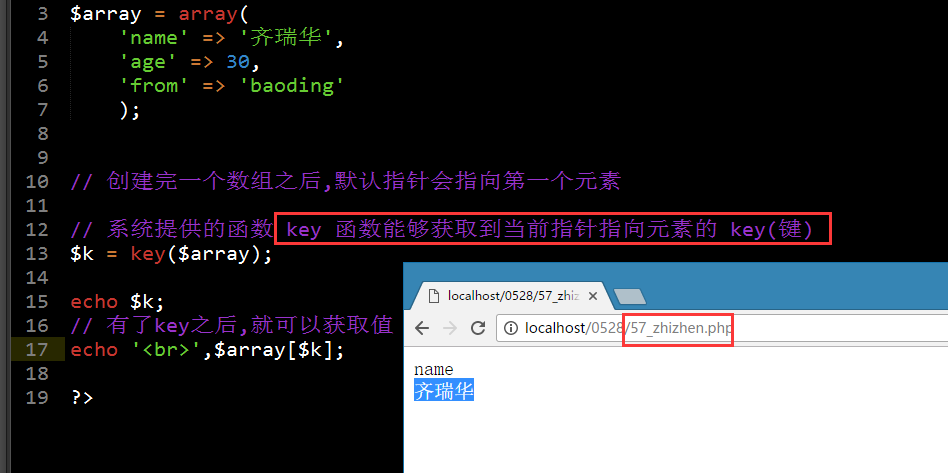
数组在分配的时候是分配的连续的区域.

其实，在每个数组内部，都存在一个数组元素的指针，它指向某个数组内的元素！并且该被指针指向的元素，就叫作当前元素！

默认的，如果我们定义完一个数组后什么都不做，数组的指针就应该指向该数组的第一个元素！

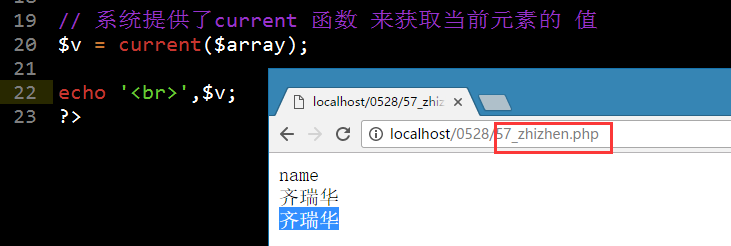
## Key

获取该数组当前元素的键！



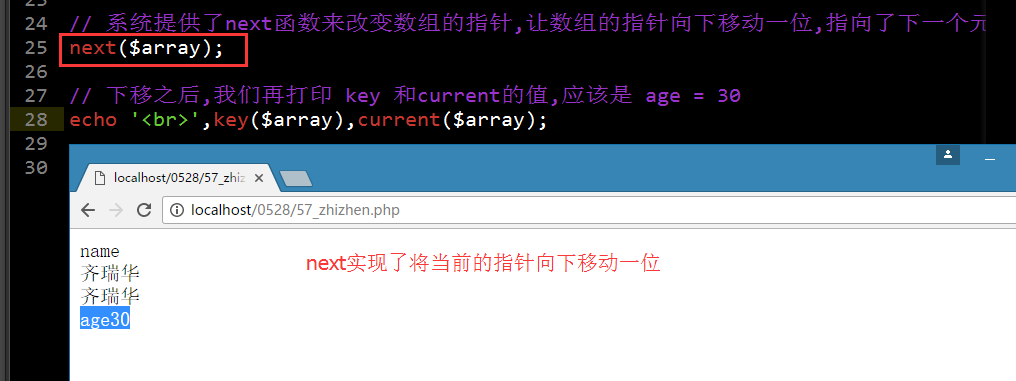
## Current

获取该数组当前元素的值！



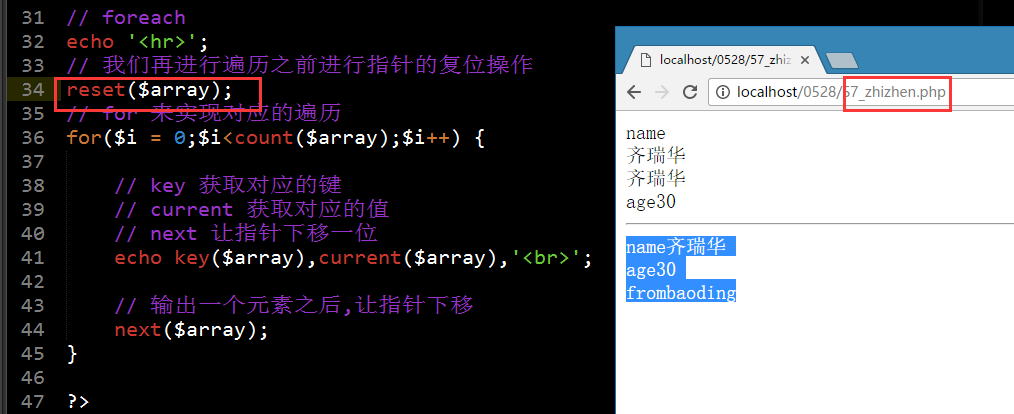
## Next

将数组的指针下移一位！



## Reset

将数组的指针进行重置（也就是指向第一个元素）！



## foreach原理



## 数组指针其他的相关函数

prev：previous单词的缩写，将数组的指针向上移动一位

end：将指针移动到最后一个元素上！

# 第58讲 while-each-list遍历

还有一个功能非常强大的函数，就是each函数，语法是：

each（数组名）

为什么强大：

同时具备了key，current，next函数的功能，既可以获得当前元素的信息（键值），又可以自动的将指针下移一位！

## 语法形式

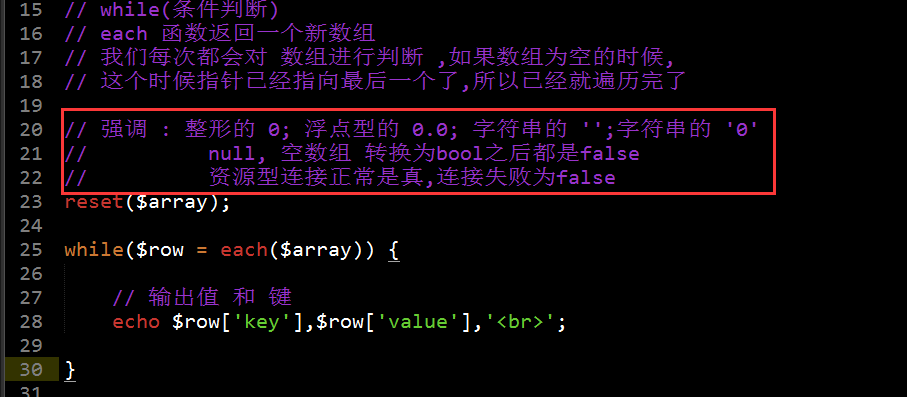
注意each的返回值：

把当前元素的键和值当成另外一个数组元素的值重新创建了一个新的数组，并为原数组的键值对分别创建了索引下标和关联下标：

其中关联的：key表示的是原数组当前元素的键，value表示的是原数组当前元素的值

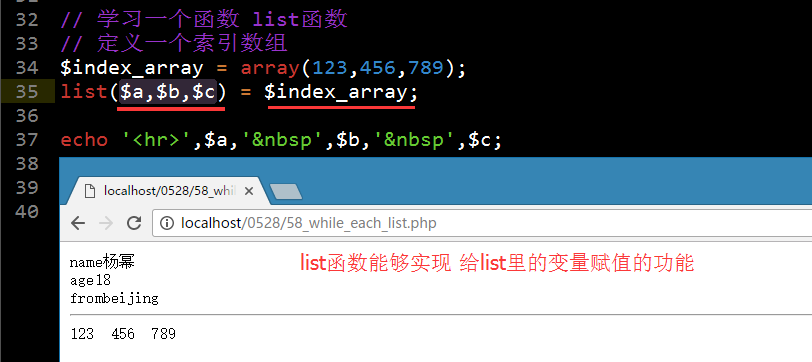
其中索引的：0表示的是原数组当前元素的键，1表示的是原数组当前元素的值

## 利用while和each实现数组遍历



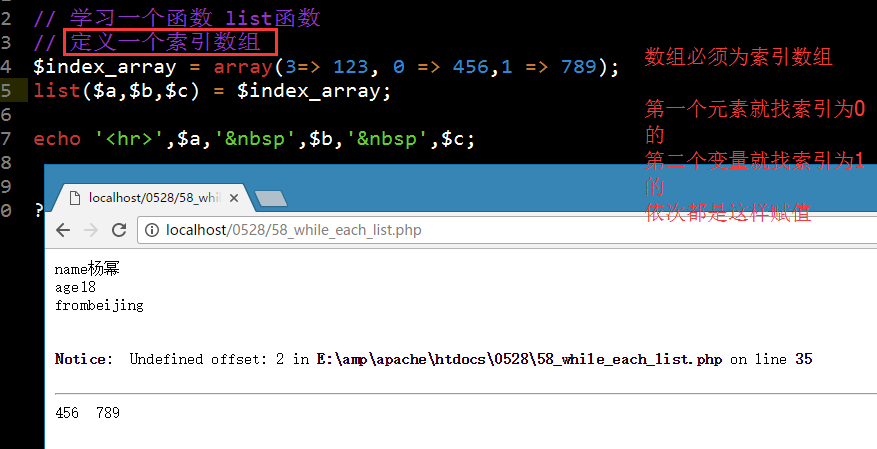
## list函数

利用一个索引数组，一次性为多个变量赋值！

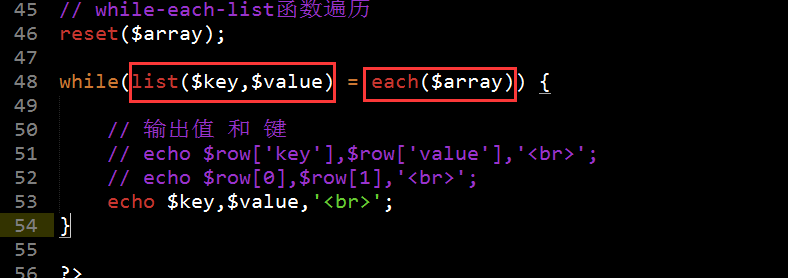


注意事项:

list的变量,第一个变量会找索引数组为0的值, 第二个变量会找索引数组为1的,顺次赋值



通过 while-each-list函数实现遍历



# 第59讲 数组的排序算法

## 冒泡排序

它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成

对以下数据进行排序

23 44 8 16 55 80

从大到小排列

第一次走访:

44 23 8 16 55 80

44 23 8 16 55 80

44 23 16 8 55 80

44 23 16 55 8 80

44 23 16 55 80 8

第一次走访,比较了5次,出现了最小的数

第二次走访:

44 23 16 55 80 8

44 23 16 55 80 8

44 23 55 16 80 8

44 23 55 80 16 8

第二次走访,比较了4次,出现了倒数第二个小的数

44 23 55 80 16 8

第三次走访:

44 23 55 80 16 8

44 55 23 80 16 8

44 55 80 23 16 8

第三次走访 比较了3次,出现了 倒数第三个小数

44 55 80 23 16 8

第四次走访,比较2次

55 44 80 23 16 8

55 80 44 23 16 8

第五次走访 比较1次

80 55 44 23 16 8

6个数 走访了 5次

第一次走访 比较了 5次

第二次走访 比较了 4次

第三次走访 比较了 3次

第四次走访 比较了 2次

第五次走访 比较了 1次

23 44 8 16 55 80

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 走访 |  |  |  |  |  |  | 比较的次数 |  |  |
|  | 23 | 44 | 8 | 16 | 55 | 80 |  |  |  |
| 1 | 44 | 23 | 16 | 55 | 80 | 8 | 5 |  |  |
| 2 | 44 | 23 | 55 | 80 | 16 | 8 | 4 |  |  |
| 3 | 44 | 55 | 80 | 23 | 16 | 8 | 3 |  |  |
| 4 | 55 | 80 | 44 | 23 | 16 | 8 | 2 |  |  |
| 5 | 80 | 55 | 44 | 23 | 16 | 8 | 1 |  |  |

6

如果我有 n个数 ,我们走访 n – 1次

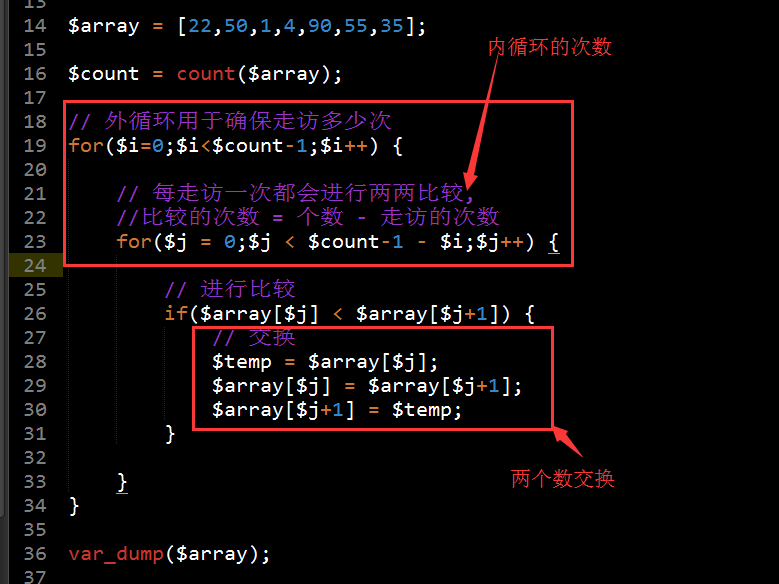
for($i=0;$i<n-1;$i++)

走访次数 和 比较次数 什么关系??

走访次数 + 比较次数 = 个数;

比较次数 = 个数 – 走访次数;

for($j = 0;$j<(n-1)-$i;$j++)

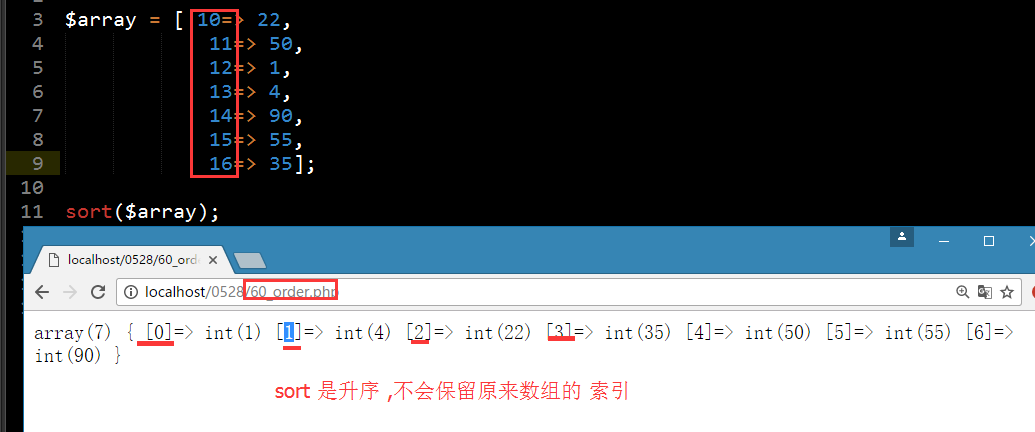


# 第60讲 排序函数

## Sort

升序就是 从小到大

对数组元素的值进行升序排序，也就是从低到高！



## Asort

对数组元素的值进行升序排序，也就是从低到高,保留原来键值对



## Rsort

降序就是 从大到小

对数组元素的值进行降序排序，也就是从高到低！

rsort 不会保留原来数组的索引

## Arsort

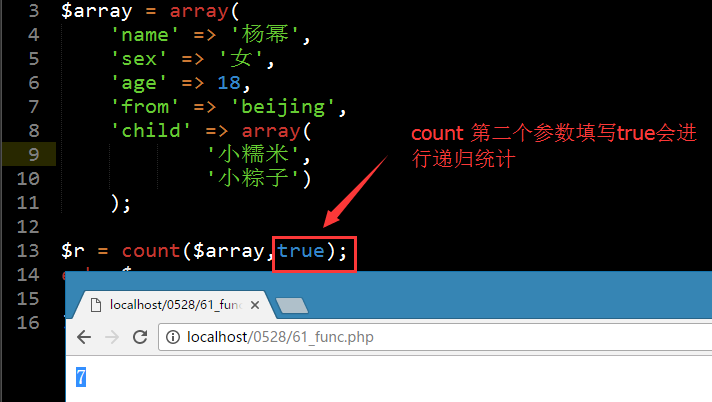
对数组元素的值进行降序排序，也就是从高到低,保留原来键值对

# 第61讲 数组的常用函数(上)

## count

计算并返回数组中的元素的个数！默认不会进行递归统计

多维数组,count默认不会对多维数组进行递归统计.



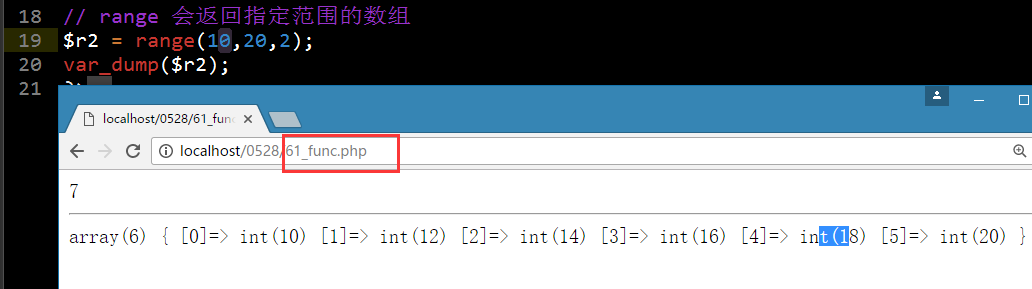
## Range

返回一个指定范围单元的数组

参数1: 开始值

参数2: 结束值

参数3: 间隔值

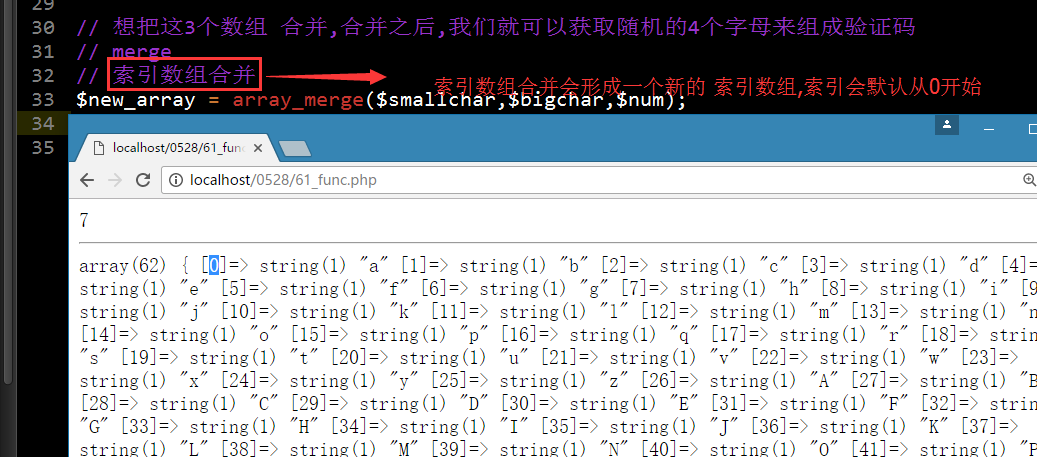


## array\_merge

将一个或多个数组的单元合并起来，一个数组的值附加到前一个数组的后面！

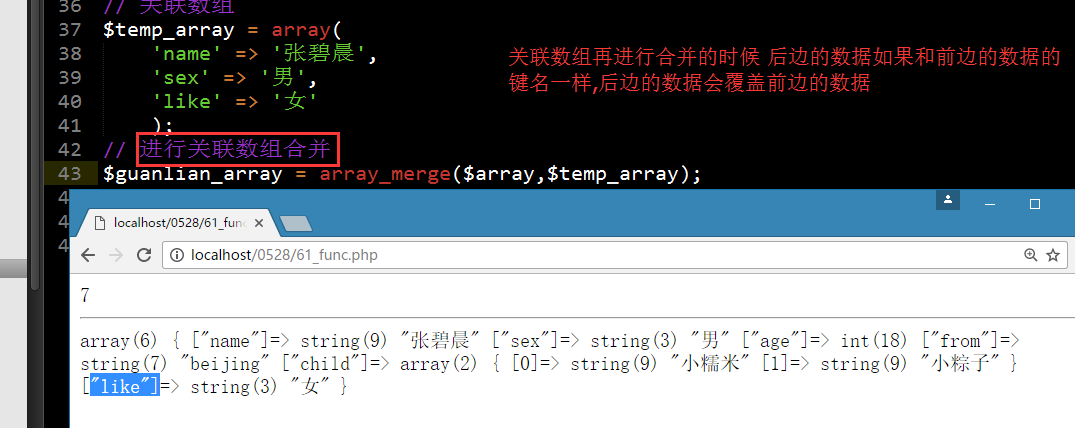
数组分为2类: 索引数组 和 关联数组

索引数组合并不会造成数据的丢失



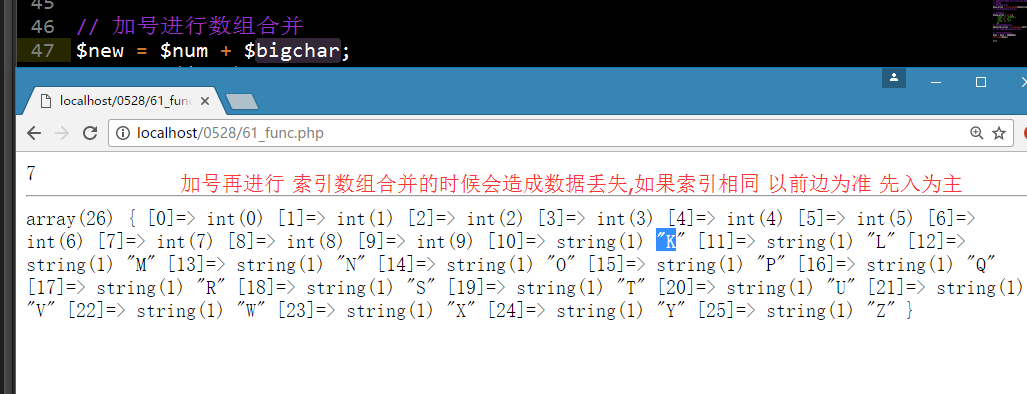
关联数组合并:

后边的值会进行覆盖前边的值

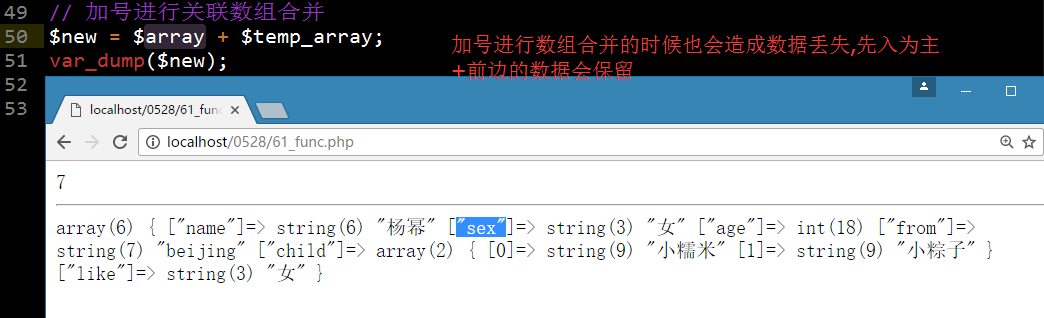


数组合并还可以采用 加号(+).

索引数组:

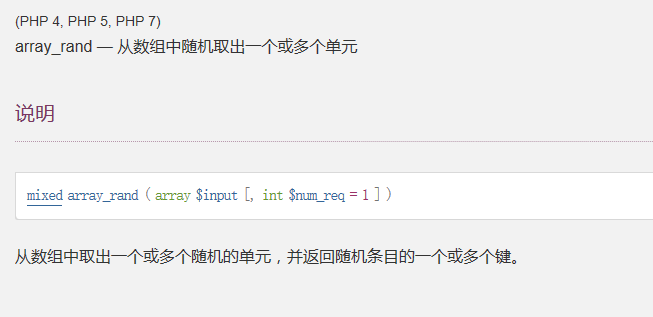


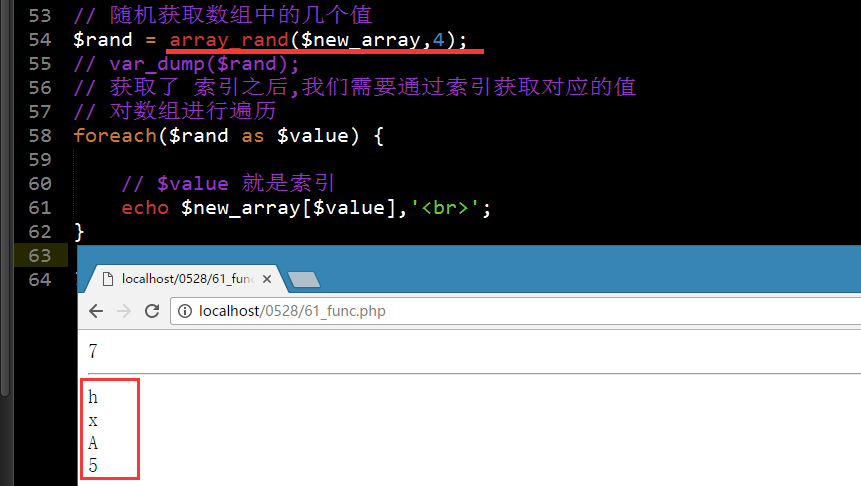
关联数组:



## array\_rand

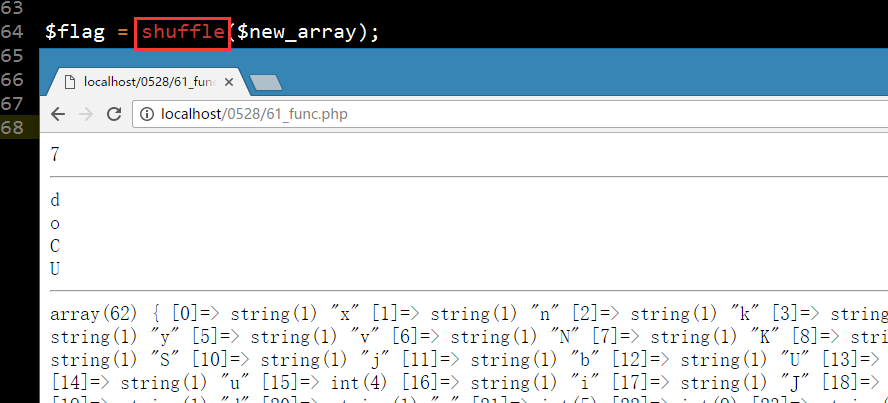
抽取一个或多个随机单元,并返回一个或多个键





## shuffle

打乱一个数组



制作一个验证码的函数就出来

① 先通过range 函数来创建3个数组

② 将三个数组合并为一个数组

③ 通过shuffle函数将原数组打乱

④ 通过array\_rand 获取4个随机下标

⑤ 对获取的随机下标进行遍历

# 第62讲 数组的常用函数(下)

## max和min

数组中的最大值和最小值

## array\_keys

取得一个数组中所有元素的键，形成一个新的索引数组！

## array\_values

取得一个数组中所有元素的值，形成一个新的索引数组！

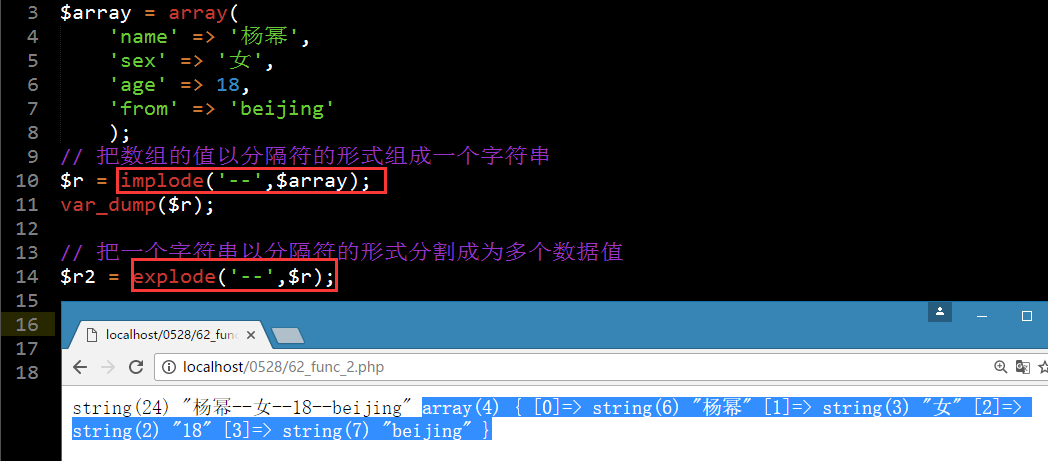
## Implode

将数组内所有元素的值，利用一个分隔符连接成一个字符串！



## Explode

将一个字符串，利用其中的分隔符，将其分割成多个部分，形成并返回一个数组！



# 数组的查找算法

## 含义

数组查找就是从数组中找一个元素的数据

数组查找的2种需求:

1.判断要找的数据是否存在

2.找出要查找的数据的位置(下标)

## 顺序查找算法

需求1: 判断要找的数据是否存在

需求2: 找出要查找的数据的位置(下标)

## 二分查找算法

二分查找的前提: 针对一个已经进行了排序的数组(即里面的数据已经有序了)

针对一个索引数组,默认索引是从0开始连续的数组