PHP 中的数组实际上是一个有序映射。映射是一种把 *values* 关联到 *keys* 的类型。此类型在很多方面做了优化，因此可以把它当成真正的数组，或列表（向量），散列表（是映射的一种实现），字典，集合，栈，队列以及更多可能性。由于数组元素的值也可以是另一个数组，树形结构和多维数组也是允许的。

解释这些结构超出了本手册的范围，但对于每种结构至少会提供一个例子。要得到这些结构的更多信息，建议参考有关此广阔主题的其它著作。

**语法**

**定义数组** [**array()**](https://www.php.net/manual/zh/function.array.php)

可以用 [array()](https://www.php.net/manual/zh/function.array.php) 语言结构来新建一个数组。它接受任意数量用逗号分隔的 *键（key） => 值（value）*对。

array( key => value

, ...

)

// 键（key）可是是一个整数 [integer](https://www.php.net/manual/zh/language.types.integer.php) 或字符串 [string](https://www.php.net/manual/zh/language.types.string.php)

// 值（value）可以是任意类型的值

最后一个数组单元之后的逗号可以省略。通常用于单行数组定义中，例如常用 *array(1, 2)* 而不是 *array(1, 2, )*。对多行数组定义通常保留最后一个逗号，这样要添加一个新单元时更方便。

自 5.4 起可以使用短数组定义语法，用 *[]* 替代 *array()*。

**Example #1 一个简单数组**

<?php  
$array = array(  
    "foo" => "bar",  
    "bar" => "foo",  
);  
  
// 自 PHP 5.4 起  
$array = [  
    "foo" => "bar",  
    "bar" => "foo",  
];  
?>

key 可以是 [integer](https://www.php.net/manual/zh/language.types.integer.php) 或者 [string](https://www.php.net/manual/zh/language.types.string.php)。value 可以是任意类型。

此外 key 会有如下的强制转换：

* 包含有合法整型值的字符串会被转换为整型。例如键名 *"8"* 实际会被储存为 *8*。但是 *"08"* 则不会强制转换，因为其不是一个合法的十进制数值。
* 浮点数也会被转换为整型，意味着其小数部分会被舍去。例如键名 *8.7* 实际会被储存为 *8*。
* 布尔值也会被转换成整型。即键名 *true* 实际会被储存为 *1* 而键名 *false* 会被储存为 *0*。
* [Null](https://www.php.net/manual/zh/language.types.null.php) 会被转换为空字符串，即键名 *null* 实际会被储存为 *""*。
* 数组和对象*不能*被用为键名。坚持这么做会导致警告：*Illegal offset type*。

如果在数组定义中多个单元都使用了同一个键名，则只使用了最后一个，之前的都被覆盖了。

**Example #2 类型强制与覆盖示例**

<?php  
$array = array(  
    1    => "a",  
    "1"  => "b",  
    1.5  => "c",  
    true => "d",  
);  
var\_dump($array);  
?>

以上例程会输出：

array(1) {

[1]=>

string(1) "d"

}

上例中所有的键名都被强制转换为 *1*，则每一个新单元都会覆盖前一个的值，最后剩下的只有一个 *"d"*。

PHP 数组可以同时含有 [integer](https://www.php.net/manual/zh/language.types.integer.php) 和 [string](https://www.php.net/manual/zh/language.types.string.php) 类型的键名，因为 PHP 实际并不区分索引数组和关联数组。

如果对给出的值没有指定键名，则取当前最大的整数索引值，而新的键名将是该值加一。如果指定的键名已经有了值，则该值会被覆盖。

**Example #3 混合** [**integer**](https://www.php.net/manual/zh/language.types.integer.php) **和** [**string**](https://www.php.net/manual/zh/language.types.string.php) **键名**

<?php  
$array = array(  
    "foo" => "bar",  
    "bar" => "foo",  
    100   => -100,  
    -100  => 100,  
);  
var\_dump($array);  
?>

以上例程会输出：

array(4) {

["foo"]=>

string(3) "bar"

["bar"]=>

string(3) "foo"

[100]=>

int(-100)

[-100]=>

int(100)

}

key 为可选项。如果未指定，PHP 将自动使用之前用过的最大 [integer](https://www.php.net/manual/zh/language.types.integer.php) 键名加上 1 作为新的键名。

**Example #4 没有键名的索引数组**

<?php  
$array = array("foo", "bar", "hallo", "world");  
var\_dump($array);  
?>

以上例程会输出：

array(4) {

[0]=>

string(3) "foo"

[1]=>

string(3) "bar"

[2]=>

string(5) "hallo"

[3]=>

string(5) "world"

}

还可以只对某些单元指定键名而对其它的空置：

**Example #5 仅对部分单元指定键名**

<?php  
$array = array(  
         "a",  
         "b",  
    6 => "c",  
         "d",  
);  
var\_dump($array);  
?>

以上例程会输出：

array(4) {

[0]=>

string(1) "a"

[1]=>

string(1) "b"

[6]=>

string(1) "c"

[7]=>

string(1) "d"

}

可以看到最后一个值 *"d"* 被自动赋予了键名 *7*。这是由于之前最大的整数键名是 *6*。

**用方括号语法访问数组单元**

数组单元可以通过 *array[key]* 语法来访问。

**Example #6 访问数组单元**

<?php  
$array = array(  
    "foo" => "bar",  
    42    => 24,  
    "multi" => array(  
         "dimensional" => array(  
             "array" => "foo"  
         )  
    )  
);  
  
var\_dump($array["foo"]);  
var\_dump($array[42]);  
var\_dump($array["multi"]["dimensional"]["array"]);  
?>

以上例程会输出：

string(3) "bar"

int(24)

string(3) "foo"

**Note**:

方括号和花括号可以互换使用来访问数组单元（例如 $array[42] 和 $array{42} 在上例中效果相同）。

自 PHP 5.4 起可以用直接对函数或方法调用的结果进行数组解引用，在此之前只能通过一个临时变量。

自 PHP 5.5 起可以直接对一个数组原型进行数组解引用。

**Example #7 数组解引用**

<?php  
function getArray() {  
    return array(1, 2, 3);  
}  
  
// on PHP 5.4  
$secondElement = getArray()[1];  
  
// previously  
$tmp = getArray();  
$secondElement = $tmp[1];  
  
// or  
list(, $secondElement) = getArray();  
?>

**Note**:

试图访问一个未定义的数组键名与访问任何未定义变量一样：会导致 **E\_NOTICE** 级别错误信息，其结果为 **NULL**。

**用方括号的语法新建／修改**

可以通过明示地设定其中的值来修改一个已有数组。

这是通过在方括号内指定键名来给数组赋值实现的。也可以省略键名，在这种情况下给变量名加上一对空的方括号（*[]*）。

$arr[key] = value;

$arr[] = value;

// key 可以是 [integer](https://www.php.net/manual/zh/language.types.integer.php) 或 [string](https://www.php.net/manual/zh/language.types.string.php)

// value 可以是任意类型的值

如果 *$arr* 还不存在，将会新建一个，这也是另一种新建数组的方法。不过并不鼓励这样做，因为如果 *$arr* 已经包含有值（例如来自请求变量的 [string](https://www.php.net/manual/zh/language.types.string.php)）则此值会保留而 *[]* 实际上代表着[字符串访问运算符](https://www.php.net/manual/zh/language.types.string.php#language.types.string.substr)。初始化变量的最好方式是直接给其赋值。。

要修改某个值，通过其键名给该单元赋一个新值。要删除某键值对，对其调用 [unset()](https://www.php.net/manual/zh/function.unset.php) 函数。

<?php  
$arr = array(5 => 1, 12 => 2);  
  
$arr[] = 56;    // This is the same as $arr[13] = 56;  
                // at this point of the script  
  
$arr["x"] = 42; // This adds a new element to  
                // the array with key "x"  
                  
unset($arr[5]); // This removes the element from the array  
  
unset($arr);    // This deletes the whole array  
?>

**Note**:

如上所述，如果给出方括号但没有指定键名，则取当前最大整数索引值，新的键名将是该值加上 1（但是最小为 0）。如果当前还没有整数索引，则键名将为 *0*。

注意这里所使用的最大整数键名*不一定*当前就在数组中。它只要在上次数组重新生成索引后曾经存在过就行了。以下面的例子来说明：

<?php  
// 创建一个简单的数组  
$array = array(1, 2, 3, 4, 5);  
print\_r($array);  
  
// 现在删除其中的所有元素，但保持数组本身不变:  
foreach ($array as $i => $value) {  
    unset($array[$i]);  
}  
print\_r($array);  
  
// 添加一个单元（注意新的键名是 5，而不是你可能以为的 0）  
$array[] = 6;  
print\_r($array);  
  
// 重新索引：  
$array = array\_values($array);  
$array[] = 7;  
print\_r($array);  
?>

以上例程会输出：

Array

(

[0] => 1

[1] => 2

[2] => 3

[3] => 4

[4] => 5

)

Array

(

)

Array

(

[5] => 6

)

Array

(

[0] => 6

[1] => 7

)

**实用函数**

有很多操作数组的函数，参见[数组函数](https://www.php.net/manual/zh/ref.array.php)一节。

**Note**:

[unset()](https://www.php.net/manual/zh/function.unset.php) 函数允许删除数组中的某个键。但要注意数组将*不会*重建索引。如果需要删除后重建索引，可以用 [array\_values()](https://www.php.net/manual/zh/function.array-values.php) 函数。

<?php  
$a = array(1 => 'one', 2 => 'two', 3 => 'three');  
unset($a[2]);  
/\* will produce an array that would have been defined as  
   $a = array(1 => 'one', 3 => 'three');  
   and NOT  
   $a = array(1 => 'one', 2 =>'three');  
\*/  
  
$b = array\_values($a);  
// Now $b is array(0 => 'one', 1 =>'three')  
?>

[foreach](https://www.php.net/manual/zh/control-structures.foreach.php) 控制结构是专门用于数组的。它提供了一个简单的方法来遍历数组。

**数组做什么和不做什么**

**为什么 *$foo[bar]* 错了？**

应该始终在用字符串表示的数组索引上加上引号。例如用 *$foo['bar']* 而不是 *$foo[bar]*。但是为什么呢？可能在老的脚本中见过如下语法：

<?php  
$foo[bar] = 'enemy';  
echo $foo[bar];  
// etc  
?>

这样是错的，但可以正常运行。那么为什么错了呢？原因是此代码中有一个未定义的常量（bar）而不是字符串（'bar'－注意引号），而 PHP 可能会在以后定义此常量，不幸的是你的代码中有同样的名字。它能运行，是因为 PHP 自动将*裸字符串*（没有引号的字符串且不对应于任何已知符号）转换成一个其值为该裸字符串的正常字符串。例如，如果没有常量定义为 **bar**，PHP 将把它替代为 *'bar'* 并使用之。

**Note**: 这并不意味着*总是*给键名加上引号。用不着给键名为[常量](https://www.php.net/manual/zh/language.constants.php)或[变量](https://www.php.net/manual/zh/language.variables.php)的加上引号，否则会使 PHP 不能解析它们。

<?php  
error\_reporting(E\_ALL);  
ini\_set('display\_errors', true);  
ini\_set('html\_errors', false);  
// Simple array:  
$array = array(1, 2);  
$count = count($array);  
for ($i = 0; $i < $count; $i++) {  
    echo "\nChecking $i: \n";  
    echo "Bad: " . $array['$i'] . "\n";  
    echo "Good: " . $array[$i] . "\n";  
    echo "Bad: {$array['$i']}\n";  
    echo "Good: {$array[$i]}\n";  
}  
?>

以上例程会输出：

Checking 0:

Notice: Undefined index: $i in /path/to/script.html on line 9

Bad:

Good: 1

Notice: Undefined index: $i in /path/to/script.html on line 11

Bad:

Good: 1

Checking 1:

Notice: Undefined index: $i in /path/to/script.html on line 9

Bad:

Good: 2

Notice: Undefined index: $i in /path/to/script.html on line 11

Bad:

Good: 2

演示此行为的更多例子：

<?php  
// Show all errors  
error\_reporting(E\_ALL);  
  
$arr = array('fruit' => 'apple', 'veggie' => 'carrot');  
  
// Correct  
print $arr['fruit'];  // apple  
print $arr['veggie']; // carrot  
  
// Incorrect.  This works but also throws a PHP error of level E\_NOTICE because  
// of an undefined constant named fruit  
//   
// Notice: Use of undefined constant fruit - assumed 'fruit' in...  
print $arr[fruit];    // apple  
  
// This defines a constant to demonstrate what's going on.  The value 'veggie'  
// is assigned to a constant named fruit.  
define('fruit', 'veggie');  
  
// Notice the difference now  
print $arr['fruit'];  // apple  
print $arr[fruit];    // carrot  
  
// The following is okay, as it's inside a string. Constants are not looked for  
// within strings, so no E\_NOTICE occurs here  
print "Hello $arr[fruit]";      // Hello apple  
  
// With one exception: braces surrounding arrays within strings allows constants  
// to be interpreted  
print "Hello {$arr[fruit]}";    // Hello carrot  
print "Hello {$arr['fruit']}";  // Hello apple  
  
// This will not work, and will result in a parse error, such as:  
// Parse error: parse error, expecting T\_STRING' or T\_VARIABLE' or T\_NUM\_STRING'  
// This of course applies to using superglobals in strings as well  
print "Hello $arr['fruit']";  
print "Hello $\_GET['foo']";  
  
// Concatenation is another option  
print "Hello " . $arr['fruit']; // Hello apple  
?>

当打开 [error\_reporting](https://www.php.net/manual/zh/errorfunc.configuration.php#ini.error-reporting) 来显示 **E\_NOTICE** 级别的错误（将其设为 **E\_ALL**）时将看到这些错误。默认情况下 [error\_reporting](https://www.php.net/manual/zh/errorfunc.configuration.php#ini.error-reporting) 被关闭不显示这些。

和在[语法](https://www.php.net/manual/zh/language.types.array.php#language.types.array.syntax)一节中规定的一样，在方括号（“*[*”和“*]*”）之间必须有一个表达式。这意味着可以这样写：

<?php  
echo $arr[somefunc($bar)];  
?>

这是一个用函数返回值作为数组索引的例子。PHP 也可以用已知常量，可能之前已经见过：

<?php  
$error\_descriptions[E\_ERROR]   = "A fatal error has occured";  
$error\_descriptions[E\_WARNING] = "PHP issued a warning";  
$error\_descriptions[E\_NOTICE]  = "This is just an informal notice";  
?>

注意 *E\_ERROR* 也是个合法的标识符，就和第一个例子中的 *bar* 一样。但是上一个例子实际上和如下写法是一样的：

<?php  
$error\_descriptions[1] = "A fatal error has occured";  
$error\_descriptions[2] = "PHP issued a warning";  
$error\_descriptions[8] = "This is just an informal notice";  
?>

因为 *E\_ERROR* 等于 *1*，等等。

**那么为什么这样做不好？**

也许有一天，PHP 开发小组可能会想新增一个常量或者关键字，或者用户可能希望以后在自己的程序中引入新的常量，那就有麻烦了。例如已经不能这样用 *empty* 和 *default* 这两个词了，因为他们是[保留字](https://www.php.net/manual/zh/reserved.php)。

**Note**: 重申一次，在双引号字符串中，不给索引加上引号是合法的因此 *"$foo[bar]"* 是合法的（“合法”的原文为 valid。在实际测试中，这么做确实可以访问数组的该元素，但是会报一个常量未定义的 notice。无论如何，强烈建议不要使用 $foo[bar]这样的写法，而要使用 $foo['bar'] 来访问数组中元素。--haohappy 注）。至于为什么参见以上的例子和[字符串中的变量解析](https://www.php.net/manual/zh/language.types.string.php#language.types.string.parsing)中的解释。

**转换为数组**

对于任意 [integer](https://www.php.net/manual/zh/language.types.integer.php)，[float](https://www.php.net/manual/zh/language.types.float.php)，[string](https://www.php.net/manual/zh/language.types.string.php)，[boolean](https://www.php.net/manual/zh/language.types.boolean.php) 和 [resource](https://www.php.net/manual/zh/language.types.resource.php) 类型，如果将一个值转换为数组，将得到一个仅有一个元素的数组，其下标为 0，该元素即为此标量的值。换句话说，*(array)$scalarValue* 与 *array($scalarValue)* 完全一样。

如果一个 [object](https://www.php.net/manual/zh/language.types.object.php) 类型转换为 [array](https://www.php.net/manual/zh/language.types.array.php)，则结果为一个数组，其单元为该对象的属性。键名将为成员变量名，不过有几点例外：整数属性不可访问；私有变量前会加上类名作前缀；保护变量前会加上一个 '\*' 做前缀。这些前缀的前后都各有一个 NULL 字符。这会导致一些不可预知的行为：

<?php  
  
class A {  
    private $A; // This will become '\0A\0A'  
}  
  
class B extends A {  
    private $A; // This will become '\0B\0A'  
    public $AA; // This will become 'AA'  
}  
  
var\_dump((array) new B());  
?>

上例会有两个键名为 'AA'，不过其中一个实际上是 '\0A\0A'。

将 **NULL** 转换为 [array](https://www.php.net/manual/zh/language.types.array.php) 会得到一个空的数组。

**比较**

可以用 [array\_diff()](https://www.php.net/manual/zh/function.array-diff.php) 和[数组运算符](https://www.php.net/manual/zh/language.operators.array.php)来比较数组。

**示例**

PHP 中的数组类型有非常多的用途。以下是一些示例：

<?php  
// This:  
$a = array( 'color' => 'red',  
            'taste' => 'sweet',  
            'shape' => 'round',  
            'name'  => 'apple',  
            4        // key will be 0  
          );  
  
$b = array('a', 'b', 'c');  
  
// . . .is completely equivalent with this:  
$a = array();  
$a['color'] = 'red';  
$a['taste'] = 'sweet';  
$a['shape'] = 'round';  
$a['name']  = 'apple';  
$a[]        = 4;        // key will be 0  
  
$b = array();  
$b[] = 'a';  
$b[] = 'b';  
$b[] = 'c';  
  
// After the above code is executed, $a will be the array  
// array('color' => 'red', 'taste' => 'sweet', 'shape' => 'round',   
// 'name' => 'apple', 0 => 4), and $b will be the array   
// array(0 => 'a', 1 => 'b', 2 => 'c'), or simply array('a', 'b', 'c').  
?>

**Example #8 使用 array()**

<?php  
// Array as (property-)map  
$map = array( 'version'    => 4,  
              'OS'         => 'Linux',  
              'lang'       => 'english',  
              'short\_tags' => true  
            );  
              
// strictly numerical keys  
$array = array( 7,  
                8,  
                0,  
                156,  
                -10  
              );  
// this is the same as array(0 => 7, 1 => 8, ...)  
  
$switching = array(         10, // key = 0  
                    5    =>  6,  
                    3    =>  7,   
                    'a'  =>  4,  
                            11, // key = 6 (maximum of integer-indices was 5)  
                    '8'  =>  2, // key = 8 (integer!)  
                    '02' => 77, // key = '02'  
                    0    => 12  // the value 10 will be overwritten by 12  
                  );  
                    
// empty array  
$empty = array();           
?>

**Example #9 集合**

<?php  
$colors = array('red', 'blue', 'green', 'yellow');  
  
foreach ($colors as $color) {  
    echo "Do you like $color?\n";  
}  
  
?>

以上例程会输出：

Do you like red?

Do you like blue?

Do you like green?

Do you like yellow?

直接改变数组的值自 PHP 5 起可以通过引用传递来做到。之前的版本需要需要采取变通的方法：

**Example #10 在循环中改变单元**

<?php  
// PHP 5  
foreach ($colors as &$color) {  
    $color = strtoupper($color);  
}  
unset($color); /\* ensure that following writes to  
$color will not modify the last array element \*/  
  
// Workaround for older versions  
foreach ($colors as $key => $color) {  
    $colors[$key] = strtoupper($color);  
}  
  
print\_r($colors);  
?>

以上例程会输出：

Array

(

[0] => RED

[1] => BLUE

[2] => GREEN

[3] => YELLOW

)

本例生成一个下标从 1 开始的数组。

**Example #11 下标从 1 开始的数组**

<?php  
$firstquarter  = array(1 => 'January', 'February', 'March');  
print\_r($firstquarter);  
?>

以上例程会输出：

Array

(

[1] => 'January'

[2] => 'February'

[3] => 'March'

)

**Example #12 填充数组**

<?php  
// fill an array with all items from a directory  
$handle = opendir('.');  
while (false !== ($file = readdir($handle))) {  
    $files[] = $file;  
}  
closedir($handle);   
?>

数组是有序的。也可以使用不同的排序函数来改变顺序。更多信息参见[数组函数](https://www.php.net/manual/zh/ref.array.php)。可以用 [count()](https://www.php.net/manual/zh/function.count.php) 函数来数出数组中元素的个数。

**Example #13 数组排序**

<?php  
sort($files);  
print\_r($files);  
?>

因为数组中的值可以为任意值，也可是另一个数组。这样可以产生递归或多维数组。

**Example #14 递归和多维数组**

<?php  
$fruits = array ( "fruits"  => array ( "a" => "orange",  
                                       "b" => "banana",  
                                       "c" => "apple"  
                                     ),  
                  "numbers" => array ( 1,  
                                       2,  
                                       3,  
                                       4,  
                                       5,  
                                       6  
                                     ),  
                  "holes"   => array (      "first",  
                                       5 => "second",  
                                            "third"  
                                     )  
                );  
  
// Some examples to address values in the array above   
echo $fruits["holes"][5];    // prints "second"  
echo $fruits["fruits"]["a"]; // prints "orange"  
unset($fruits["holes"][0]);  // remove "first"  
  
// Create a new multi-dimensional array  
$juices["apple"]["green"] = "good";   
?>

数组(Array) 的赋值总是会涉及到值的拷贝。使用[引用运算符](https://www.php.net/manual/zh/language.operators.php)通过引用来拷贝数组。

<?php  
$arr1 = array(2, 3);  
$arr2 = $arr1;  
$arr2[] = 4; // $arr2 is changed,  
             // $arr1 is still array(2, 3)  
               
$arr3 = &$arr1;  
$arr3[] = 4; // now $arr1 and $arr3 are the same  
?>