Arrays (Arreglos en PHP)

(Documento lo pueden encontrar en www.php.net)

Un array en PHP es realmente un mapa ordenado. Un mapa es un tipo de datos que asocia *valores* con *claves*. Este tipo es optimizado para varios usos diferentes; puede ser usado como una matriz real, una lista (vector), una tabla asociativa (una implementación de un mapa), diccionario, colección, pila, cola, y posiblemente más. Ya que los valores de un array pueden ser otros arrays, árboles y también son posibles arrays multidimensionales.

Una explicación sobre tales estructuras de datos está fuera del alcance de este manual, pero encontrará al menos un ejemplo de cada uno de ellos. Para más información, consulte la extensa literatura que existe sobre este amplio tema.

Sintaxis

Especificación con array()

Un array puede ser creado usando el constructor del lenguaje array(). Éste toma un cierto número de parejas *clave => valor* como argumentos.

```
array(
clave  => valor,
clave2 => valor2,
clave3 => valor3,
...
)
```

La coma después del elemento del array es opcional y se puede omitir. Esto normalmente se hace para arrays de una sola línea, esto es, es preferible *array(1, 2)* que *array(1, 2,)*. Por otra parte, para arrays multilínea, la coma final se usa comnúnmente, ya que permite la adición sencilla de nuevos elementos al final.

A partir de PHP 5.4 también se puede usar la sintaxis de array corta, que reemplaza *array()* con *[]*.

Ejemplo #1 Un array simple

```php
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// a partir de PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
```

```
];
?>
```

La clave puede ser un integer o un string. El valor puede ser de cualquier tipo.

Además, los siguientes moldeados de tipo en la clave producirá:

- Strings que contienen integers válidos serán moldeados a el tipo integer. P.e.j. la clave *"8"* en realidad será almacenada como *8*. Por otro lado *"08"* no será convertido, ya que no es un número integer decimal válido.
- Floats también serán moldeados en integers, lo que significa que la parte fraccionaria se elimina. P.e.j. la clave *8.7* en realidad será almacenada como *8*.
- Bools son moldeados a integers, también, p.e.j. la clave *true* en realidad será almacenada como *1* y la clave *false* como *0*.
- Null será moldeado a un string vacío, p.e.j. la clave *null* en realidad será almacenada como *""*.
- Arrays y objects *no pueden* ser usados como claves. Si lo hace, dará lugar a una advertencia: *Illegal offset type*.

Si varios elementos en la declaración del array usan la misma clave, sólo la última será usada y los demás son sobrescritos.

Ejemplo #2 Ejemplo de moldeado de tipo y sobrescritura

```
<?php
$array = array(
   1    => "a",
   "1"  => "b",
   1.5  => "c",
   true => "d",
);
var_dump($array);
?>
```

El resultado del ejemplo sería:

```
array(1) {
  [1]=>
  string(1) "d"
}
```

Como todas las claves en el ejemplo anterior se convierten en *1*, los valores serán sobrescritos en cada nuevo elemento y el último valor asignado *"d"* es el único que queda.

Los arrays PHP pueden contener claves integer y string al mismo tiempo ya que PHP no distingue entre arrays indexados y asociativos.

Ejemplo #3 Claves mixtas integer y string

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100   => -100,
    -100  => 100,
);
var_dump($array);
?>
```

El resultado del ejemplo sería:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

La clave es opcional. Si no se especifica, PHP usará el incremento de la clave integer más grande utilizada anteriormente.

Ejemplo #4 Arrays indexados sin clave

```
<?php
$array = array("foo", "bar", "hallo", "world");
var_dump($array);
?>
```

El resultado del ejemplo sería:

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hallo"
  [3]=>
  string(5) "world"
}
```

Es posible especificar la clave sólo para algunos de los elementos y dejar por fuera a los demás:

Ejemplo #5 Claves no en todos los elementos

```
<?php
$array = array(
        "a",
        "b",
    6 => "c",
        "d",
);
var_dump($array);
?>
```

El resultado del ejemplo sería:

```
array(4) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7]=>
  string(1) "d"
}
```

Como se puede ver el último valor *"d"* se le asignó la clave *7*. Esto es debido a que la mayor clave integer era *6*.

Acceso a elementos de array con la sintaxis de corchete

Los elementos de array se pueden acceder utilizando la sintaxis *array[key]*.

Ejemplo #6 Acceso a elementos de array

```
<?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"]);
?>
```

El resultado del ejemplo sería:

```
string(3) "bar"
int(24)
string(3) "foo"
```

A partir de PHP 5.4 es posible hacer referencia al array de el resultado de una llamada a la función o el método directamente. Antes sólo era posible utilizando una variable temporal.

Ejemplo #7 Hacer referencia al resultado array de funciones

```php
<?php
function getArray() {
    return array(1, 2, 3);
}

// en PHP 5.4
$secondElement = getArray()[1];

// anteriormente
$tmp = getArray();
$secondElement = $tmp[1];

// o
list(, $secondElement) = getArray();
?>
```

Nota:

Si intenta acceder a una clave del array que no se ha definido es lo mismo que el acceso a cualquier otra variable no definida: se emitirá un mensaje de error de nivel **E_NOTICE**, y el resultado será **NULL**.

Creación/modificación con sintaxis de corchete

Un array existente puede ser modificado al definir valores explícitamente en éste.

Esto se realiza mediante la asignación de valores al array, especificando la clave en corchetes. La clave también se puede omitir, resultando en un par de corchetes vacíos (*[]*).

```
$arr[clave] = valor;
$arr[] = valor;
// clave puede ser un integer o string
// valor puede ser cualquier valor de cualquier tipo
```

Si *$arr* aún no existe, se creará, así que esto es también una forma alternativa de crear un array. Sin embargo, esta práctica es desaconsejada ya que si *$arr* ya contiene algún valor (p.ej. un string de una variable de petición), entonces este valor estará en su lugar

y *[]* puede significar realmente el operador de acceso a cadenas. Siempre es mejor inicializar variables mediante una asignación directa.

Para cambiar un valor determinado, se debe asignar un nuevo valor a ese elemento con su clave. Para quitar un par clave/valor, se debe llamar la función unset() en éste.

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // Esto es lo mismo que $arr[13] = 56;
                // en este punto de el script

$arr["x"] = 42; // Esto agrega un nuevo elemento a
                // el array con la clave "x"

unset($arr[5]); // Esto elimina el elemento del array

unset($arr);    // Esto elimina el array completo
?>
```

Nota:

Como se mencionó anteriormente, si no se especifica una clave, se toma el máximo de los índices integer existentes, y la nueva clave será ese valor máximo más 1 (aunque al menos 0). Si todavía no existen índices integer, la clave será *0* (cero).

Tenga en cuenta que la clave integer máxima utilizada para éste *no es necesario que actualmente exista en el array*. Ésta sólo debe haber existido en el array en algún momento desde la última vez que el array fué re-indexado. El siguiente ejemplo ilustra este comportamiento:

```
<?php
// Crear un array simple.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Ahora elimina cada elemento, pero deja el mismo array intacto:
foreach ($array as $i => $value) {
   unset($array[$i]);
}
print_r($array);

// Agregar un elemento (note que la nueva clave es 5, en lugar de 0).
$array[] = 6;
print_r($array);

// Re-indexar:
$array = array_values($array);
$array[] = 7;
```

```
print_r($array);
?>
```

El resultado del ejemplo sería:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

Funciones útiles

Hay un buen número de funciones útiles para trabajar con arrays. Véase la sección funciones de array.

Nota:

La función unset() permite remover claves de un array. Tenga en cuenta que el array *no* es re-indexado. Si se desea un verdadero comportamiento "eliminar y desplazar", el array puede ser re-indexado usando la función array_values().

```
<?php
$a = array(1 => 'uno', 2 => 'dos', 3 => 'tres');
unset($a[2]);
/* producirá un array que se ha definido como
   $a = array(1 => 'uno', 3 => 'tres');
   y NO
   $a = array(1 => 'uno', 2 =>'tres');
*/

$b = array_values($a);
// Ahora $b es array(0 => 'uno', 1 =>'tres')
?>
```

La estructura de control foreach existe específicamente para arrays. Ésta provee una manera fácil de recorrer un array.

Recomendaciones sobre arrays y cosas a evitar

¿Por qué es incorrecto *$foo[bar]*?

Siempre deben usarse comillas alrededor de un índice de array tipo string literal. Por ejemplo, *$foo['bar']* es correcto, mientras que *$foo[bar]* no lo es. ¿Pero por qué? Es común encontrar este tipo de sintaxis en scripts viejos:

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

Esto está mal, pero funciona. La razón es que este código tiene una constante indefinida (bar) en lugar de un valor string ('bar' - note las comillas). Puede que en el futuro PHP defina constantes que, desafortunadamente para tal tipo de código, tengan el mismo nombre. Funciona porque PHP automáticamente convierte una *cadena pura* (un string sin comillas que no corresponda con símbolo conocido alguno) en un string que contiene la cadena pura. Por ejemplo, si no se ha definido una constante llamada **bar**, entonces PHP reemplazará su valor por la cadena *'bar'* y usará ésta última.

Nota: Esto no quiere decir que *siempre* haya que usar comillas en la clave. No use comillas con claves que sean constantes o variables, ya que en tal caso PHP no podrá interpretar sus valores.

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', true);
ini_set('html_errors', false);
// Array simple:
$array = array(1, 2);
$count = count($array);
for ($i = 0; $i < $count; $i++) {
    echo "\nRevisando $i: \n";
    echo "Mal: " . $array['$i'] . "\n";
    echo "Bien: " . $array[$i] . "\n";
    echo "Mal: {$array['$i']}\n";
    echo "Bien: {$array[$i]}\n";
}
?>
```

El resultado del ejemplo sería:

```
Revisando 0:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Mal:
```

```
Bien: 1
Notice: Undefined index:  $i in /path/to/script.html on line 11
Mal:
Bien: 1

Revisando 1:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Mal:
Bien: 2
Notice: Undefined index:  $i in /path/to/script.html on line 11
Mal:
Bien: 2
```

Más ejemplos para demostrar este comportamiento:

```php
<?php
// Mostrar todos los errores
error_reporting(E_ALL);

$arr = array('fruit' => 'apple', 'veggie' => 'carrot');

// Correcto
print $arr['fruit'];  // apple
print $arr['veggie']; // carrot

// Incorrecto. Esto funciona pero también genera un error de PHP de
// nivel E_NOTICE ya que no hay definida una constante llamada fruit
//
// Notice: Use of undefined constant fruit - assumed 'fruit' in...
print $arr[fruit];    // apple

// Esto define una constante para demostrar lo que pasa. El valor 'veggie'
// es asignado a una constante llamada fruit.
define('fruit', 'veggie');

// Note la diferencia ahora
print $arr['fruit'];  // apple
print $arr[fruit];    // carrot

// Lo siguiente está bien ya que se encuentra al interior de una cadena. Las constantes no
 son procesadas al
// interior de cadenas, así que no se produce un error E_NOTICE aquí
print "Hello $arr[fruit]";      // Hello apple

// Con una excepción, los corchetes que rodean las matrices al
// interior de cadenas permiten el uso de constantes
print "Hello {$arr[fruit]}";    // Hello carrot
print "Hello {$arr['fruit']}";  // Hello apple

// Esto no funciona, resulta en un error de intérprete como:
```

```
// Parse error: parse error, expecting T_STRING' or T_VARIABLE' or T_NUM_STRIN
G'
// Esto por supuesto se aplica también al uso de superglobales en cadenas
print "Hello $arr['fruit']";
print "Hello $_GET['foo']";

// La concatenación es otra opción
print "Hello " . $arr['fruit']; // Hello apple
?>
```

Cuando se habilita error_reporting para mostrar errores de nivel **E_NOTICE** (como por ejemplo definiendo el valor **E_ALL**), este tipo de usos serán inmediatamente visibles. Por omisión, error_reporting se encuentra configurado para no mostrarlos.

Tal y como se indica en la sección de sintaxis, lo que existe entre los corchetes cuadrados ('*[*' y '*]*') debe ser una expresión. Esto quiere decir que código como el siguiente funciona:

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

Este es un ejemplo del uso de un valor devuelto por una función como índice del array. PHP también conoce las constantes:

```
<?php
$error_descriptions[E_ERROR]   = "Un error fatal ha ocurrido";
$error_descriptions[E_WARNING] = "PHP produjo una advertencia";
$error_descriptions[E_NOTICE]  = "Esta es una noticia informal";
?>
```

Note que **E_ERROR** es también un identificador válido, asi como *bar* en el primer ejemplo. Pero el último ejemplo es equivalente a escribir:

```
<?php
$error_descriptions[1] = "Un error fatal ha ocurrido";
$error_descriptions[2] = "PHP produjo una advertencia";
$error_descriptions[8] = "Esta es una noticia informal";
?>
```

ya que **E_ERROR** es igual a *1*, etc.

¿Entonces porqué está mal?

En algún momento en el futuro, puede que el equipo de PHP quiera usar otra constante o palabra clave, o una constante proveniente de otro código puede interferir. Por ejemplo, en este momento no puede usar las palabras *empty* y *default* de esta forma, ya que son palabras clave reservadas.

Nota: Reiterando, al interior de un valor string entre comillas dobles, es válido no rodear los índices de array con comillas, así que *"$foo[bar]"* es válido. Consulte los ejemplos anteriores para más detalles sobre el porqué, así como la sección sobre procesamiento de variables en cadenas.

Conversión a array

Para cualquiera de los tipos: integer, float, string, boolean y resource, convertir un valor a un array resulta en un array con un solo elemento, con índice 0, y el valor del escalar que fue convertido. En otras palabras, *(array)$scalarValue* es exactamente lo mismo que *array($scalarValue)*.

Si convierte un object a un array, el resultado es un array cuyos elementos son las propiedades del object. Las claves son los nombres de las variables miembro, con algunas excepciones notables: las variables privadas tienen el nombre de la clase al comienzo del nombre de la variable; las variables protegidas tienen un caracter '*' al comienzo del nombre de la variable. Estos valores adicionados al inicio tienen bytes nulos a los lados. Esto puede resultar en algunos comportamientos inesperados:

```
<?php

class A {
   private $A; //  Este campo se convertirá en '\0A\0A'
}

class B extends A {
   private $A; // Este campo se convertirá en '\0B\0A'
   public $AA; // Este campo se convertirá en 'AA'
}

var_dump((array) new B());
?>
```

En el ejemplo anterior parecerá que se tienen dos claves llamadas 'AA', aunque en realidad una de ellas se llama '\0A\0A'.

Si convierte un valor **NULL** a array, obtiene un array vacío.

Comparación

Es posible comparar arrays con la función array_diff() y mediante operadores de arrays.

Ejemplos

El tipo array en PHP es bastante versátil. Aquí hay algunos ejempos:

```
<?php
// Esto:
$a = array( 'color' => 'red',
            'taste' => 'sweet',
```

```
            'shape' => 'round',
            'name'  => 'apple',
            4       // la clave será 0
           );

$b = array('a', 'b', 'c');

// . . .es completamente equivalente a
$a = array();
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name']  = 'apple';
$a[]        = 4;        // la clave será 0

$b = array();
$b[] = 'a';
$b[] = 'b';
$b[] = 'c';

// Después de que se ejecute el código, $a será el array
// array('color' => 'red', 'taste' => 'sweet', 'shape' => 'round',
// 'name' => 'apple', 0 => 4), y $b será el array
// array(0 => 'a', 1 => 'b', 2 => 'c'), o simplemente array('a', 'b', 'c').
?>
```

Ejemplo #8 Uso de array()

```
<?php
// Array como mapa de propiedades
$map = array( 'version'    => 4,
              'OS'         => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );

// Keys estrictamente numéricas
$array = array( 7,
                8,
                0,
                156,
                -10
              );
// esto es lo mismo que array(0 => 7, 1 => 8, ...)

$switching = array(         10, // key = 0
                     5    =>  6,
                     3    =>  7,
                     'a'  =>  4,
                             11, // key = 6 (el índice entero máximo era 5)
```

```
                '8'  =>  2, // key = 8 (integer!)
                '02' => 77, // key = '02'
                 0    => 12  // el valor 10 será reemplazado por 12
               );

// array vacío
$empty = array();
?>
```

Ejemplo #9 Colección

```
<?php
$colors = array('rojo', 'azul', 'verde', 'amarillo');

foreach ($colors as $color) {
    echo "¿Le gusta el $color?\n";
}

?>
```

El resultado del ejemplo sería:

```
¿Le gusta el rojo?
¿Le gusta el azul?
¿Le gusta el verde?
¿Le gusta el amarillo?
```

Modificar los valores del array directamente es posible a partir de PHP 5, pasándolos por referencia. Las versiones anteriores necesitan una solución alternativa:

Ejemplo #10 Cambiando elemento en el bucle

```
<?php
// PHP 5
foreach ($colors as &$color) {
    $color = strtoupper($color);
}
unset($color); /* se asegura de que escrituras subsiguientes a $color
no modifiquen el último elemento del arrays */

// Alternativa para versiones anteriores
foreach ($colors as $key => $color) {
    $colors[$key] = strtoupper($color);
}

print_r($colors);
?>
```

El resultado del ejemplo sería:

```
Array
(
    [0] => ROJO
    [1] => AZUL
    [2] => VERDE
    [3] => AMARILLO
)
```

Este ejemplo crea un array con base uno.

Ejemplo #11 Índice con base 1

```php
<?php
$firstquarter  = array(1 => 'Enero', 'Febrero', 'Marzo');
print_r($firstquarter);
?>
```

El resultado del ejemplo sería:

```
Array
(
    [1] => 'Enero'
    [2] => 'Febrero'
    [3] => 'Marzo'
)
```

Ejemplo #12 Llenado de un array

```php
<?php
// llenar un array con todos los ítems de un directorio
$handle = opendir('.');
while (false !== ($file = readdir($handle))) {
    $files[] = $file;
}
closedir($handle);
?>
```

Los Arrays son ordenados. El orden puede ser modificado usando varias funciones de ordenado. Vea la sección sobre funciones de arrays para más información. La función count() puede ser usada para contar el número de elementos en un array.

Ejemplo #13 Ordenado de un array

```php
<?php
sort($files);
print_r($files);
?>
```

Dado que el valor de un array puede ser cualquier cosa, también puede ser otro array. De esta forma es posible crear arrays recursivas y multi-dimensionales.

Ejemplo #14 Arrays recursivos y multi-dimensionales

```
<?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"
                                     )
                );

// Algunos ejemplos que hacen referencia a los valores del array anterior
echo $fruits["holes"][5];    // prints "second"
echo $fruits["fruits"]["a"]; // prints "orange"
unset($fruits["holes"][0]);  // remove "first"

// Crear una nueva array multi-dimensional
$juices["apple"]["green"] = "good";
?>
```

La asignación de arrays siempre involucra la copia de valores. Use el operador de referencia para copiar un array por referencia.

```
<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // $arr2 ha cambiado,
             // $arr1 sigue siendo array(2, 3)

$arr3 = &$arr1;
$arr3[] = 4; // ahora $arr1 y $arr3 son iguales
?>
```