Focus search box

array_merge_recursive »
« array_keys

- Manual de PHP
- Referencia de funciones
- Extensiones relacionadas con variable y tipo
- Arrays
- Funciones de Arrays

Change language: Spanish

Submit a Pull Request Report a Bug

array_map

(PHP 4 >= 4.0.6, PHP 5, PHP 7, PHP 8)

array_map — Aplica la retrollamada a los elementos de los arrays dados

Descripción ¶

array_map(callable `$callback`, array `$array1`, array `$...` = ?): array

array_map() devuelve un array que contiene todos los elementos de `array1` después de haber aplicado la función `callback` a cada uno de ellos. El número de parámetros que la función `callback` acepta debería coincidir con el número de arrays proporcionados a **array_map()**.

Parámetros ¶

`callback`

Función de retrollamada a ejecutar para cada elemento de cada array.

`array1`

Un array a recorrer con la función `callback`.

`...`

Lista variable de argumentos de tipo array a recorrer con la función `callback`.

Valores devueltos ¶

Devuelve un array que contiene todos los elementos de `array1` después de aplicar la función `callback` a cada uno de ellos.

Ejemplos ¶

Ejemplo #1 Ejemplo de array_map()

```
<?php
function cube($n)
{
    return($n * $n * $n);
}
```

```
$a = array(1, 2, 3, 4, 5);
$b = array_map("cube", $a);
print_r($b);
?>
```

Este ejemplo hace que *$b* contenga:

```
Array
(
    [0] => 1
    [1] => 8
    [2] => 27
    [3] => 64
    [4] => 125
)
```

Ejemplo #2 array_map() usando una función lambda (desde PHP 5.3.0)

```
<?php
$func = function($valor) {
    return $valor * 2;
};

print_r(array_map($func, range(1, 5)));
?>
```

```
Array
(
    [0] => 2
    [1] => 4
    [2] => 6
    [3] => 8
    [4] => 10
)
```

Ejemplo #3 array_map() - usando más arrays

```
<?php
function mostrar_en_español($n, $m)
{
    return("El número $n se llama $m en español");
}

function correspondencia_en_español($n, $m)
{
    return(array($n => $m));
}

$a = array(1, 2, 3, 4, 5);
$b = array("uno", "dos", "tres", "cuatro", "cinco");

$c = array_map("mostrar_en_español", $a, $b);
print_r($c);

$d = array_map("correspondencia_en_español", $a , $b);
print_r($d);
?>
```

El resultado del ejemplo sería:

```
// salida de $c
Array
(
    [0] => El número 1 se llama uno en español
    [1] => El número 2 se llama dos en español
    [2] => El número 3 se llama tres en español
    [3] => El número 4 se llama cuatro en español
    [4] => El número 5 se llama cinco en español
)

// salida of $d
Array
(
    [0] => Array
        (
            [1] => uno
        )

    [1] => Array
        (
            [2] => dos
        )

    [2] => Array
        (
            [3] => tres
        )

    [3] => Array
        (
            [4] => cuatro
        )

    [4] => Array
        (
            [5] => cinco
        )

)
```

Usualmente, cuando se usan dos o más arrays, estos deberían ser de la misma longitud, ya que la retrollamada se aplica en paralelo a los elementos correspondientes. Si los arrays son de longitudes diferentes, los más cortos se extenderán con elementos vacíos para que coincidan con la logintud del más largo.

Un uso interesante de esta función es la construcción de un array de arrays, lo que se puede llevar a cabo usando **`null`** como el nombre de la retrollamada.

Ejemplo #4 Crear un array de arrays

```php
<?php
$a = array(1, 2, 3, 4, 5);
$b = array("one", "two", "three", "four", "five");
$c = array("uno", "dos", "tres", "cuatro", "cinco");

$d = array_map(null, $a, $b, $c);
print_r($d);
?>
```

El resultado del ejemplo sería:

```
Array
(
    [0] => Array
        (
            [0] => 1
```

```
            [1] => one
            [2] => uno
        )

    [1] => Array
        (
            [0] => 2
            [1] => two
            [2] => dos
        )

    [2] => Array
        (
            [0] => 3
            [1] => three
            [2] => tres
        )

    [3] => Array
        (
            [0] => 4
            [1] => four
            [2] => cuatro
        )

    [4] => Array
        (
            [0] => 5
            [1] => five
            [2] => cinco
        )

)
```

El array devuelto conservará las claves del argumento array si y solo si se pasa exactamente un array. Si se pasa más de un array, el array devuelto tendrá claves secuenciales de tipo integer.

Ejemplo #5 array_map() - con claves de tipo string

```php
<?php
$arr = array("stringkey" => "value");
function cb1($a) {
    return array ($a);
}
function cb2($a, $b) {
    return array ($a, $b);
}
var_dump(array_map("cb1", $arr));
var_dump(array_map("cb2", $arr, $arr));
var_dump(array_map(null,  $arr));
var_dump(array_map(null, $arr, $arr));
?>
```

El resultado del ejemplo sería:

```
array(1) {
  ["stringkey"]=>
  array(1) {
    [0]=>
    string(5) "value"
  }
}
array(1) {
  [0]=>
  array(2) {
    [0]=>
```

```
    string(5) "value"
    [1]=>
    string(5) "value"
  }
}
array(1) {
  ["stringkey"]=>
  string(5) "value"
}
array(1) {
  [0]=>
  array(2) {
    [0]=>
    string(5) "value"
    [1]=>
    string(5) "value"
  }
}
```

Ver también ¶

- array_filter() - Filtra elementos de un array usando una función de devolución de llamada
- array_reduce() - Reduce iterativamente un array a un solo valor usando una función llamada de retorno
- array_walk() - Aplicar una función proporcionada por el usuario a cada miembro de un array

+ add a note

User Contributed Notes 8 notes

up
down
29
lukasz dot mordawski at gmail dot com ¶
8 years ago

Let's assume we have following situation:

```
<?php
class MyFilterClass {
    public function filter(array $arr) {
        return array_map(function($value) {
            return $this->privateFilterMethod($value);
        });
    }

    private function privateFilterMethod($value) {
        if (is_numeric($value)) $value++;
        else $value .= '.';
    }
}
?>
```

This will work, because $this inside anonymous function (unlike for example javascript) is the instance of MyFilterClass inside which we called it.
I hope this would be useful for anyone.

up
down
19
elfe1021 at gmail dot com ¶
8 years ago

Find an interesting thing that in array_map's callable function, late static binding does not work:

```
<?php
class A {
    public static function foo($name) {
        return 'In A: '.$name;
    }

    public static function test($names) {
        return array_map(function($n) {return static::foo($n);}, $names);
    }
}

class B extends A{
    public static function foo($name) {
        return 'In B: '.$name;
    }
}

$result = B::test(['alice', 'bob']);
var_dump($result);
?>
```

the result is:

```
array (size=2)
  0 => string 'In A: alice' (length=11)
  1 => string 'In A: bob' (length=9)
```

if I change A::test to

```
<?php
    public static function test($names) {
        return array_map([get_called_class(), 'foo'], $names);
    }
?>
```

Then the result is as expected:

```
array (size=2)
  0 => string 'In B: alice' (length=11)
  1 => string 'In B: bob' (length=9)
```

up
down
17
radist-hack at yandex dot ru ¶
14 years ago

To transpose rectangular two-dimension array, use the following code:

```
array_unshift($array, null);
$array = call_user_func_array("array_map", $array);
```

If you need to rotate rectangular two-dimension array on 90 degree, add the following line before or after (depending on the rotation direction you need) the code above:

```
$array = array_reverse($array);
```

Here is example:

```
<?php
$a = array(
  array(1, 2, 3),
```

```
  array(4, 5, 6));
array_unshift($a, null);
$a = call_user_func_array("array_map", $a);
print_r($a);
?>

Output:

Array
(
    [0] => Array
        (
            [0] => 1
            [1] => 4
        )

    [1] => Array
        (
            [0] => 2
            [1] => 5
        )

    [2] => Array
        (
            [0] => 3
            [1] => 6
        )

)
```

up
down
17
Mahn ¶
7 years ago

You may be looking for a method to extract values of a multidimensional array on a conditional basis (i.e. a mixture between array_map and array_filter) other than a for/foreach loop. If so, you can take advantage of the fact that 1) the callback method on array_map returns null if no explicit return value is specified (as with everything else) and 2) array_filter with no arguments removes falsy values.

So for example, provided you have:

```
<?php
$data = [
    [
        "name" => "John",
        "smoker" => false
    ],
    [
        "name" => "Mary",
        "smoker" => true
    ],
    [
        "name" => "Peter",
        "smoker" => false
    ],
    [
        "name" => "Tony",
```

```
        "smoker" => true
    ]
];
?>
```

You can extract the names of all the non-smokers with the following one-liner:

```
<?php
$names = array_filter(array_map(function($n) { if(!$n['smoker']) return $n['name']; }, $data));
?>
```

It's not necessarily better than a for/foreach loop, but the occasional one-liner for trivial tasks can help keep your code cleaner.

up
down
8

stijnleenknegt at gmail dot com ¶

14 years ago

If you want to pass an argument like ENT_QUOTES to htmlentities, you can do the follow.

```
<?php
$array = array_map( 'htmlentities' , $array, array_fill(0 , count($array) , ENT_QUOTES) );
?>
```

The third argument creates an equal sized array of $array filled with the parameter you want to give with your callback function.

up
down
7

CertaiN ¶

9 years ago

The most memory-efficient array_map_recursive().

```
<?php
function array_map_recursive(callable $func, array $arr) {
    array_walk_recursive($arr, function(&$v) use ($func) {
        $v = $func($v);
    });
    return $arr;
}
?>
```

up
down
1

Walf ¶

7 months ago

A general solution for the problem of wanting to know the keys in the callback, and/or retain the key association in the returned array:

```
<?php

/**
* Like array_map() but callback also gets passed the current key as the
* first argument like so:
* function($key, $val, ...$vals) { ... }
* ...and returned array always maintains key association, even if multiple
* array arguments are passed.
*/
```

```
function array_map_assoc(callable $callback, array $array, array ...$arrays) {
    $keys = array_keys($array);
    array_unshift($arrays, $keys, $array);
    return array_combine($keys, array_map($callback, ...$arrays));
}

?>
```

Because it uses array_map() directly, it behaves the same way in regard to ignoring the keys of subsequent array arguments. It also has the same variadic signature.

up
down
1

anonymous_user ¶
11 months ago

```
/**
  * Function which recursively applies a callback to all values and also its
  * keys, and returns the resulting array copy with the updated keys and
  * values.
  * PHP's built-in function array_walk_recursive() only applies the passed
  * callback to the array values, not the keys, so this function simply applies
  * the callback to the keys too (hence the need of working with a copy,
  * as also updating the keys would lead to reference loss of the original
  * array). I needed something like this, hence my idea of sharing it here.
  *
  * @param    callable    $func     callback which takes one parameter (value
  *                                                 or key to be updated) and returns its
  *                                                 updated value
  *
  * @param    array          $arr      array of which keys and values shall be
  *                                                 get updated
  */

function array_map_recursive(
    callable $func,
    array $arr
) {

      // Initiate copied array which will hold all updated keys + values
      $result = [];

      // Iterate through the key-value pairs of the array
      foreach ( $arr as $key => $value ) {

        // Apply the callback to the key to create the updated key value
        $updated_key = $func( $key );

        // If the iterated value is not an array, that means we have reached the
        // deepest array level for the iterated key, so in that case, assign
        // the updated value to the updated key value in the final output array
        if ( ! is_array( $value ) ) {

          $result[$updated_key] = $func( $value );

        } else {

          // If the iterated value is an array, call the function recursively,
```

```
            // By taking the currently iterated value as the $arr argument
            $result[$updated_key] = array_map_recursive(
              $func,
              $arr[$key]
            );

          }

        } // end of iteration through k-v pairs

        // And at the very end, return the generated result set
        return $result;

      } // end of array_map_recursive() function definition
```

+ add a note

- Funciones de Arrays
  - array_change_key_case
  - array_chunk
  - array_column
  - array_combine
  - array_count_values
  - array_diff_assoc
  - array_diff_key
  - array_diff_uassoc
  - array_diff_ukey
  - array_diff
  - array_fill_keys
  - array_fill
  - array_filter
  - array_flip
  - array_intersect_assoc
  - array_intersect_key
  - array_intersect_uassoc
  - array_intersect_ukey
  - array_intersect
  - array_is_list
  - array_key_exists
  - array_key_first
  - array_key_last
  - array_keys
  - array_map
  - array_merge_recursive
  - array_merge
  - array_multisort
  - array_pad
  - array_pop
  - array_product
  - array_push
  - array_rand
  - array_reduce
  - array_replace_recursive
  - array_replace
  - array_reverse
  - array_search
  - array_shift
  - array_slice
  - array_splice