array_multisort »
« array_merge_recursive

- 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_merge

(PHP 4, PHP 5, PHP 7, PHP 8)

array_merge — Combina dos o más arrays

Descripción ¶

array_merge(array `$array1`, array `$...` = ?): array

Combina los elementos de uno o más arrays juntándolos de modo que los valores de uno se anexan al final del anterior. Retorna el array resultante.

Si los arrays de entrada tienen las mismas claves de tipo string, el último valor para esa clave sobrescribirá al anterior. Sin embargo, los arrays que contengan claves numéricas, el último valor *no* sobrescribirá el valor original, sino que será añadido al final.

Los valores del array de entrada con claves numéricas serán renumeradas con claves incrementales en el array resultante, comenzando desde cero.

Parámetros ¶

`array1`

Array inicial a combinar.

`...`

Lista variable de arrays para combinar.

Valores devueltos ¶

Retorna el array resultante.

Ejemplos ¶

Ejemplo #1 Ejemplo de array_merge()

```
<?php
$array1    = array("color" => "red", 2, 4);
$array2    = array("a", "b", "color" => "green", "shape" => "trapezoid", 4);
$resultado = array_merge($array1, $array2);
```

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

El resultado del ejemplo sería:

```
Array
(
    [color] => green
    [0] => 2
    [1] => 4
    [2] => a
    [3] => b
    [shape] => trapezoid
    [4] => 4
)
```

Ejemplo #2 Ejemplo de sencillo de array_merge()

```
<?php
$array1    = array();
$array2    = array(1 => "data");
$resultado = array_merge($array1, $array2);
?>
```

¡No olvidarse de que las claves numéricas serán renumeradas!

```
Array
(
    [0] => data
)
```

Para anexar elementos del segundo array al primer array entretanto no se sobrescriban los elementos del primer array y no se reindexe, se ha de utilizar el operador + de unión de arrays.

```
<?php
$array1    = array(0 => 'zero_a', 2 => 'two_a', 3 => 'three_a');
$array2    = array(1 => 'one_b', 3 => 'three_b', 4 => 'four_b');
$resultado = $array1 + $array2;
var_dump($resultado);
?>
```

Las claves del primer array se preservaran. Si una clave existe en ambos arrays, se usará el elemento del primer array, y el elemento de la clave coincidente del segundo array será ignorado.

```
array(5) {
  [0]=>
  string(6) "zero_a"
  [2]=>
  string(5) "two_a"
  [3]=>
  string(7) "three_a"
  [1]=>
  string(5) "one_b"
  [4]=>
  string(6) "four_b"
}
```

Ejemplo #3 array_merge() con tipos que no son array

```
<?php
$comienzo = 'foo';
$fin = array(1 => 'bar');
$resultado = array_merge((array)$comienzo, (array)$fin);
```

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

El resultado del ejemplo sería:

```
Array
(
    [0] => foo
    [1] => bar
)
```

Ver también ¶

- array_merge_recursive() - Une dos o más arrays recursivamente
- array_replace() - Reemplaza los elementos del array original con elementos de array adicionales
- array_combine() - Crea un nuevo array, usando una matriz para las claves y otra para sus valores
- operadores de arrays

+ add a note

User Contributed Notes 6 notes

up
down
292
Julian Egelstaff ¶
13 years ago
In some situations, the union operator (+) might be more useful to you than array_merge. The array_merge function does not preserve numeric key values. If you need to preserve the numeric keys, then using + will do that.

ie:

```
<?php

$array1[0] = "zero";
$array1[1] = "one";

$array2[1] = "one";
$array2[2] = "two";
$array2[3] = "three";

$array3 = $array1 + $array2;

//This will result in::

$array3 = array(0=>"zero", 1=>"one", 2=>"two", 3=>"three");

?>
```

Note the implicit "array_unique" that gets applied as well. In some situations where your numeric keys matter, this behaviour could be useful, and better than array_merge.

--Julian

up
down
23
ChrisM ¶

11 months ago

I wished to point out that while other comments state that the spread operator should be faster than array_merge, I have actually found the opposite to be true for normal arrays. This is the case in both PHP 7.4 as well as PHP 8.0. The difference should be negligible for most applications, but I wanted to point this out for accuracy.

Below is the code used to test, along with the results:

```
<?php
$before = microtime(true);

for ($i=0 ; $i<10000000 ; $i++) {
    $array1 = ['apple','orange','banana'];
    $array2 = ['carrot','lettuce','broccoli'];

    $array1 = [...$array1,...$array2];
}

$after = microtime(true);
echo ($after-$before) . " sec for spread\n";

$before = microtime(true);

for ($i=0 ; $i<10000000 ; $i++) {
    $array1 = ['apple','orange','banana'];
    $array2 = ['carrot','lettuce','broccoli'];

    $array1 = array_merge($array1,$array2);
}

$after = microtime(true);
echo ($after-$before) . " sec for array_merge\n";
?>

PHP 7.4:
1.2135608196259 sec for spread
1.1402177810669 sec for array_merge

PHP 8.0:
1.1952061653137 sec for spread
1.099925994873 sec for array_merge
```

up
down
6

Andreas Hofmann ¶

11 months ago

In addition to the text and Julian Egelstaffs comment regarding to keep the keys preserved with the + operator:
When they say "input arrays with numeric keys will be renumbered" they MEAN it. If you think you are smart and put your numbered keys into strings, this won't help. Strings which contain an integer will also be renumbered! I fell into this trap while merging two arrays with book ISBNs as keys. So let's have this example:

```
<?php
    $test1['24'] = 'Mary';
    $test1['17'] = 'John';

    $test2['67'] = 'Phil';
```

```
    $test2['33'] = 'Brandon';

    $result1 = array_merge($test1, $test2);
    var_dump($result1);

    $result2 = [...$test1, ...$test2];    // mentioned by fsb
    var_dump($result2);
?>
```

You will get both:

```
array(4) {
  [0]=>
  string(4) "Mary"
  [1]=>
  string(4) "John"
  [2]=>
  string(4) "Phil"
  [3]=>
  string(7) "Brandon"
}
```

Use the + operator or array_replace, this will preserve - somewhat - the keys:

```
<?php
    $result1 = array_replace($test1, $test2);
    var_dump($result1);

    $result2 = $test1 + $test2;
    var_dump($result2);
?>
```

You will get both:

```
array(4) {
  [24]=>
  string(4) "Mary"
  [17]=>
  string(4) "John"
  [67]=>
  string(4) "Phil"
  [33]=>
  string(7) "Brandon"
}
```

The keys will keep the same, the order will keep the same, but with a little caveat: The keys will be converted to integers.

up
down
9

fsb at thefsb dot org ¶

2 years ago

We no longer need array_merge() as of PHP 7.4.

```
    [...$a, ...$b]
```

does the same as

```
    array_merge($a, $b)

and can be faster too.

https://wiki.php.net/rfc/spread_operator_for_array#advantages_over_array_merge
```

up
down
0
JoshE ¶
8 months ago

Not to contradict ChrisM's test, but I ran their code example and I got very different results for PHP 8.0.

```
Testing PHP 8.0.14
1.4955070018768 sec for spread
4.4120140075684 sec for array_merge
```

up
down
-1
php at k dot ull dot at ¶
29 days ago

Merge two arrays and retain only unique values.
Append values from second array.
Do not care about keys.

```
$array1 = [
    0 => 'apple',
    1 => 'orange',
    2 => 'pear',
];

$array2 = [
    0 => 'melon',
    1 => 'orange',
    2 => 'banana',
];

$result = array_keys(
    array_flip($array1) + array_flip($array2)
);

Result:
[
  [0] => "apple",
  [1] => "orange",
  [2] => "pear",
  [3] => "melon",
  [4] => "banana",
}
```

+ 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
- array_sum
- array_udiff_assoc
- array_udiff_uassoc
- array_udiff
- array_uintersect_assoc
- array_uintersect_uassoc
- array_uintersect
- array_unique
- array_unshift
- array_values
- array_walk_recursive
- array_walk
- array
- arsort
- asort
- compact
- count
- current
- end
- extract
- in_array
- key_exists
- key