Focus search box

array »
« array_walk_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_walk

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

array_walk — Aplicar una función proporcionada por el usuario a cada miembro de un array

Descripción ¶

array_walk(array `&$array`, callable `$callback`, mixed `$userdata` = `null`): bool

Aplica la función definida por el usuario dada por `callback` a cada elemento del array dado por `array`.

array_walk() no le afecta el puntero de arrays interno de `array`. **array_walk()** recorrerá el array completo sin tener en cuenta la posición del puntero.

Parámetros ¶

`array`

El array de entrada.

`callback`

Normalmente, `callback` asume dos parámetros. El primero, los valores de los parámetros de `array`, y el segundo la clave/índice.

Nota:

Si `callback` necesita trabajar con los valores reales del array, especifique el primer parámetro de `callback` como una referencia. Así, cualquier cambio hecho a esos elementos serán hechos al mismo array original.

Nota:

Muchas funciones internas (por ejemplo strtolower()) lanzarán una advertencia si se pasan más argumentos de los esperados y no son utilizables directamente como `callback`.

Sólo se pueden cambiar potencialmente los valores del parámetro `array`; no se puede alterar su estructura, esto es, el programador no puede añadir, destruir o reordenar elementos. Si la llamada de retorno no respeta este requisito, el comportamiento de esta función será indefinido e impredecible.

`userdata`

Si se proporciona el parámetro opcional `userdata`, éste será pasado como el tercer parámetro de la función dada por `callback`.

Valores devueltos ¶

Devuelve **`true`**.

Errores/Excepciones ¶

A partir de PHP 7.1.0, un ArgumentCountError será lanzado si la función `callback` requiere más de 2 parámetros (el valor y la clave del miembro del array). Anteriormente, si la función `callback` requería más de 2 parámetros, se generaba un error de nivel E_WARNING cada vez que **array_walk()** llamaba a `callback`.

Ejemplos ¶

Ejemplo #1 Ejemplo de array_walk()

```php
<?php
$frutas = array("d" => "limón", "a" => "naranja", "b" => "banana", "c" => "manzana");

function test_alter(&$elemento1, $clave, $prefijo)
{
    $elemento1 = "$prefijo: $elemento1";
}

function test_print($elemento2, $clave)
{
    echo "$clave. $elemento2<br />\n";
}

echo "Antes ...:\n";
array_walk($frutas, 'test_print');

array_walk($frutas, 'test_alter', 'fruta');
echo "... y después:\n";

array_walk($frutas, 'test_print');
?>
```

El resultado del ejemplo sería:

```
Antes ...:
d. limón
a. naranja
b. banana
c. manzana
... y después:
d. fruta: limón
a. fruta: naranja
b. fruta: banana
c. fruta: manzana
```

Ver también ¶

- array_walk_recursive() - Aplicar una función de usuario recursivamente a cada miembro de un array
- iterator_apply() - Llamar a una función para cada elemento de un iterador
- list() - Asignar variables como si fueran un array
- each() - Devolver el par clave/valor actual de un array y avanzar el cursor del array

- call_user_func_array() - Llamar a una llamada de retorno con un array de parámetros
- array_map() - Aplica la retrollamada a los elementos de los arrays dados
- foreach

+ add a note

User Contributed Notes 36 notes

up
down
226
bisqwit at iki dot fi ¶
18 years ago
It's worth nothing that array_walk can not be used to change keys in the array.
The function may be defined as (&$value, $key) but not (&$value, &$key).
Even though PHP does not complain/warn, it does not modify the key.

up
down
67
ezhacher at gmail dot com ¶
8 years ago
Calling an array Walk inside a class

```
If the class is static:
array_walk($array, array('self', 'walkFunction'));
or
array_walk($array, array('className', 'walkFunction'));

Otherwise:
array_walk($array, array($this, 'walkFunction'));
```

up
down
49
01001coder at gmail dot com ¶
4 years ago
I noticed that :

PHP ignored arguments type when using array_walk() even if there was

declare(strict_types=1) .

See this code as an example ...

```
<?php
declare(strict_types=1);

$fruits = array("butter" => 5.3, "meat" => 7, "banana" => 3);

function test_print(int $item2, $key) {
    echo "$key: $item2<br />\n";
}

array_walk($fruits, 'test_print');

?>
```

The output is :

```
butter: 5
meat: 7
banana: 3

whilst the expecting output is :

Fatal error: Uncaught TypeError: Argument 1 passed to test_print() must be of the type integer

because "butter" => 5.3 is float

I asked someone about it and they said "this was caused by the fact that callbacks called from
internal code will always use weak type". But I tried to do some tests and this behavior is not an
issue when using call_user_func().
```

up
down
3

fred ¶

6 years ago

```
Correction for the speed test from zlobnygrif.

<?php
// Test results
$array1 = test('array_walk');
$array2 = test('array_walk_list_each');
$array3 = test('array_walk_foreach1');
$array4 = test('array_walk_foreach2');

// Check arrays for equal
var_dump($array1 == $array2, $array1 == $array3, $array1 == $array4);

// Test function 1
function array_walk_list_each(&$array, $function, $userData = null) {
   /* make sure we walk the array each time */
    reset($array);
    while ( list($key, $value) = each($array) )
        $function($array[$key], $key, $userData);
}

// Test function 2
function array_walk_foreach1(&$array, $function, $userData = null) {
    foreach ($array as $key => &$value )
        $function($value, $key, $userData);
}

// Test function 3
function array_walk_foreach2(&$array, $function, $userData = null) {
    foreach ($array as $key => $value )
        $function($array[$key], $key, $userData);
}

function some_function(&$value, $key, $userData) {
    $value = "$key => $userData";
}

function test($function, $count = 10000, $arrayElements = 1000) {
    echo $function, ' ... ';
    $array = array_fill(0, $arrayElements, "some text value");
```

```
    $timer = microtime(true);
    for( $i = 0; ++$i < $count; )
        /* change data for each $i */
        $function($array, 'some_function', 'some user data ' . $i);
    printf("%.3f sec\n", microtime(true) - $timer);

    return $array;
}
```

up
down
8
chaley at brtransport dot com ¶
8 years ago

There is a note about 3 years ago regarding using this for trimming. array_map() may be cleaner for this. I haven't checked the time/resource impact:

```
$result = array_map("trim", $array);
```

up
down
14
taj at yahoo dot fr ¶
3 years ago

```
// We can make that with this simple FOREACH loop :

$fruits = array("d" => "lemon", "a" => "orange", "b" => "banana", "c" => "apple");

foreach($fruits as $cls => $vls)
{
  $fruits[$cls] = "fruit: ".$vls;
}

Results:

Array
(
    [d] => fruit: lemon
    [a] => fruit: orange
    [b] => fruit: banana
    [c] => fruit: apple
)
```

up
down
17
Maxim ¶
11 years ago

Note that using array_walk with intval is inappropriate.
There are many examples on internet that suggest to use following code to safely escape $_POST arrays of integers:

```
<?php
array_walk($_POST['something'],'intval'); // does nothing in PHP 5.3.3
?>
```

It works in _some_ older PHP versions (5.2), but is against specifications. Since intval() does not modify it's arguments, but returns modified result, the code above has no effect on the array and will leave security hole in your website.

You can use following instead:

```
<?php
```

```
$_POST['something'] = array_map(intval,$_POST['something']);
?>
```

up
down
5
EllisGL ¶
4 years ago

For those that think they can't use array_walk to change / replace a key name, here you go:

```
<?php
function array_explore(array &$array, callable $callback)
{
    array_walk($array, function(&$value, $key) use (&$array, $callback)
    {
        $callback($array, $key, $value);

        if(is_array($value))
        {
            array_explore($value, $callback);
        }
    });
}

/**
* Stolen from: https://stackoverflow.com/questions/13233405/change-key-in-associative-array-in-php
*/
function renameKey(array &$data, $oldKey, $newKey, $ignoreMissing = false, $replaceExisting =
false)
{
    if (!empty($data))
    {
        if (!array_key_exists($oldKey, $data))
        {
            if ($ignoreMissing)
            {
                return FALSE;
            }

            throw new \Exception('Old key does not exist.');
        }
        else
        {
            if (array_key_exists($newKey, $data))
            {
                if ($replaceExisting)
                {
                    unset($data[$newKey]);
                }
                else
                {
                    throw new \Exception('New key already exist.');
                }
            }

            $keys = array_keys($data);

            // Fix from EllisGL: http://php.net/manual/en/function.array-search.php#122377
            $keys[array_search($oldKey, array_map('strval', $keys))] = $newKey;
```

```
            $data = array_combine($keys, $data);

            return TRUE;
        }
    }

    return FALSE;
}

$array = [
    "_10fish" => 'xyz',
    "_11fish" => [
        "_22" => "a", "b", "c"
        ],
    "someFish" => [
        'xyz',
        '@attributes' => ['type' => 'cod']
        ]
    ];

array_explore($array, function(&$value, $key)
{
    // Replace key '@attrutes' with '_attributes'
    if('@attributes' === $key)
    {
        renameKey($value, $key, '_attributes');
    }

});

print_r($array);
?>
```

up
down
2
ludvig dot ericson at gmail dot com ¶
16 years ago

In response to 'ibolmo', this is an extended version of string_walk, allowing to pass userdata (like array_walk) and to have the function edit the string in the same manner as array_walk allows, note now though that you have to pass a variable, since PHP cannot pass string literals by reference (logically).

```
<?php
function string_walk(&$string, $funcname, $userdata = null) {
    for($i = 0; $i < strlen($string); $i++) {
        # NOTE: PHP's dereference sucks, we have to do this.
        $hack = $string{$i};
        call_user_func($funcname, &$hack, $i, $userdata);
        $string{$i} = $hack;
    }
}

function yourFunc($value, $position) {
    echo $value . ' ';
}

function yourOtherFunc(&$value, $position) {
```

```
    $value = str_rot13($value);
}

# NOTE: We now need this ugly $x = hack.
string_walk($x = 'interesting', 'yourFunc');
// Ouput: i n t e r e s t i n g

string_walk($x = 'interesting', 'yourOtherFunc');
echo $x;
// Output: vagrerfgvat
?>
```

Also note that calling str_rot13() directly on $x would be much faster ;-) just a sample.

up
down
18

rustamabd at gmail dot com ¶

12 years ago

Don't forget about the array_map() function, it may be easier to use!

Here's how to lower-case all elements in an array:

```
<?php
    $arr = array_map('strtolower', $arr);
?>
```

up
down
11

erelsgl at gmail dot com ¶

13 years ago

If you want to unset elements from the callback function, maybe what you really need is array_filter.

up
down
10

fantomx1 at gmail dot com ¶

6 years ago

Since array_walk cannot modify / change / reindex keys as already mentioned, i provide this small wrapping function which accomplishes passing array reference and index using closures , "use" keyword.

```
function indexArrayByElement($array, $element)
{
    $arrayReindexed = [];
    array_walk(
        $array,
        function ($item, $key) use (&$arrayReindexed, $element) {
            $arrayReindexed[$item[$element]] = $item;
        }
    );
    return $arrayReindexed;
}
```

up
down
14

Andrzej Martynowicz at gmail dot com ¶

17 years ago

It can be very useful to pass the third (optional) parameter by reference while modifying it permanently in callback function. This will cause passing modified parameter to next iteration of array_walk(). The exaple below enumerates items in the array:

```
<?php
function enumerate( &$item1, $key, &$startNum ) {
   $item1 = $startNum++ ." $item1";
}

$num = 1;

$fruits = array( "lemon", "orange", "banana", "apple");
array_walk($fruits, 'enumerate', $num );

print_r( $fruits );

echo '$num is: '. $num ."\n";
?>
```

This outputs:

```
Array
(
    [0] => 1 lemon
    [1] => 2 orange
    [2] => 3 banana
    [3] => 4 apple
)
$num is: 1
```

Notice at the last line of output that outside of array_walk() the $num parameter has initial value of 1. This is because array_walk() does not take the third parameter by reference.. so what if we pass the reference as the optional parameter..

```
<?php
$num = 1;

$fruits = array( "lemon", "orange", "banana", "apple");
array_walk($fruits, 'enumerate', &$num ); // reference here

print_r( $fruits );

echo '$num is: '. $num ."\n";
echo "we've got ". ($num - 1) ." fruits in the basket!";
?>
```

This outputs:

```
Array
(
    [0] => 1 lemon
    [1] => 2 orange
    [2] => 3 banana
    [3] => 4 apple
)
$num is: 5
we've got 4 fruits in the basket!
```

Now $num has changed so we are able to count the items (without calling count() unnecessarily).

As a conclusion, using references with array_walk() can be powerful toy but this should be done carefully since modifying third parameter outside the array_walk() is not always what we want.

up
down
3

alex_stanhope at hotmail dot com ¶
11 years ago

I wanted to walk an array and reverse map it into a second array. I decided to use array_walk because it should be faster than a reset,next loop or foreach(x as &$y) loop.

```
<?php
$output = array();
array_walk($input, 'gmapmark_reverse', $output);

function gmapmark_reverse(&$item, $index, &$target) {
    $target[$item['form_key']] = $index;
}
?>
```

In my debugger I can see that $target is progressively updated, but when array_walk returns, $output is empty. If however I use a (deprecated) call-by-reference:

```
<?php
array_walk($input, 'gmapmark_reverse', &$output);
?>
```

$output is returned correctly. Unfortunately there's not an easy way to suppress the warnings:

```
<?php
@array_walk($input, 'gmapmark_reverse', &$output);
?>
```

doesn't silence them. I've designed a workaround using a static array:

```
<?php
$reverse = array();
array_walk($input, 'gmapmark_reverse');
// call function one last time to get target array out, because parameters don't work
$reverse = gmapmark_reverse($reverse);

function gmapmark_reverse(&$item, $index = 0) {
  static $target;
  if (!$target) {
    $target = array();
  }
  if (isset($item['form_key'])) {
    $target[$item['form_key']] = $index;
  }
  return($target);
}
?>
```

up
down
3

brian at access9 dot net ¶
9 years ago

```
array_walk does not work on SplFixedArray objects:
<?php
$array = new SplFixedArray(2);
$array[0] = 'test_1';
$array[1] = 'test_2';

array_walk($array, function(&$val){
    $val .= '__';
    return $val;
});
foreach ($array as $a) {
    echo "$a\n";
}
?>

result is:
test_1
test_2
```

up
down
4
jab_creations_-at_-yahoo_-dot-_com ¶
13 years ago

Unfortunately I spent a lot of time trying to permanently apply the effects of a function to an array using the array_walk function when instead array_map was what I wanted. Here is a very simple though effective example for those who may be getting overly frustrated with this function...

```
<?php
$fruits = array("Lemony & Fresh","Orange Twist","Apple Juice");

print_r($fruits);
echo '<br />';

function name_base($key)
{
$name2 = str_replace(" ", "_", $key);
$name3 = str_replace("&", "and", $name2);
$name4 = strtolower($name3);
echo $name4.'<br />';
return $name4;
}
echo '<br />';

$test = array_map('name_base', $fruits);
$fruits_fixed = $test;
echo '<br />';
print_r($fruits_fixed);
?>
```

up
down
6
matthew at codenaked dot org ¶
12 years ago

Using lambdas you can create a handy zip function to zip together the keys and values of an array. I extended it to allow you to pass in the "glue" string as the optional userdata parameter. The following example is used to zip an array of email headers:

```
<?php

/**
* Zip together the keys and values of an array using the provided glue
*
* The values of the array are replaced with the new computed value
*
* @param array $data
* @param string $glue
*/
function zip(&$data, $glue=': ')
{
    if(!is_array($data)) {
        throw new InvalidArgumentException('First parameter must be an array');
    }

    array_walk($data, function(&$value, $key, $joinUsing) {
        $value = $key . $joinUsing . $value;
    }, $glue);
}

$myName = 'Matthew Purdon';
$myEmail = 'matthew@example.com';
$from = "$myName <$myEmail>";

$headers['From'] = $from;
$headers['Reply-To'] = $from;
$headers['Return-path'] = "<$myEmail>";
$headers['X-Mailer'] = "PHP" . phpversion() . "";
$headers['Content-Type'] = 'text/plain; charset="UTF-8"';

zip($headers);

$headers = implode("\n", $headers);
$headers .= "\n";

echo $headers;

/*
From: Matthew Purdon <matthew@example.com>
Reply-To: Matthew Purdon <matthew@example.com>
Return-path: <matthew@example.com>
X-Mailer: PHP5.3.2
Content-Type: text/plain; charset="UTF-8"
*/
?>
```

up
down
6

zlobnygrif at gmail dot com ¶

9 years ago

Some speed tests

```
<?php
// Test results
$array1 = test('array_walk');
$array2 = test('array_walk_list_each');
$array3 = test('array_walk_foreach1');
```

```
$array4 = test('array_walk_foreach2');

// Check arrays for equal
var_dump($array1 == $array2, $array1 == $array3, $array1 == $array4);

// Test function 1
function array_walk_list_each(&$array, $function, $userData = null) {
    while ( list($key, $value) = each($array) )
        $function($array[$key], $key, $userData);
}

// Test function 2
function array_walk_foreach1(&$array, $function, $userData = null) {
    foreach ($array as $key => &$value )
        $function($value, $key, $userData);
}

// Test function 3
function array_walk_foreach2(&$array, $function, $userData = null) {
    foreach ($array as $key => $value )
        $function($array[$key], $key, $userData);
}

function some_function(&$value, $key, $userData) {
    $value = "$key => $userData";
}

function test($function, $count = 10000, $arrayElements = 1000) {
    echo $function, ' ... ';
    $array = array_fill(0, $arrayElements, "some text value");

    $timer = microtime(true);
    for( $i = 0; ++$i < $count; )
        $function($array, 'some_function', 'some user data');
    printf("%.3f sec\n", microtime(true) - $timer);

    return $array;
}
?>

Output (PHP 5.4.9-4ubuntu2.2 (cli) (built: Jul 15 2013 18:24:39))
=========================
array_walk ... 13.572 sec
array_walk_list_each ... 0.027 sec
array_walk_foreach1 ... 15.356 sec
array_walk_foreach2 ... 17.416 sec
bool(true)
bool(true)
bool(true)

Output (PHP 5.5.0 (cli) (built: Jul 16 2013 17:59:42) - same server)
=========================
array_walk ... 4.776 sec
array_walk_list_each ... 0.006 sec
array_walk_foreach1 ... 4.482 sec
array_walk_foreach2 ... 5.166 sec
bool(true)
bool(true)
```

```
bool(true)

PHP 5.5 array_walk looks pretty good but list each is more and more quickly...
```

up
down
2
op_adept at yahoo dot co dot uk ¶
11 years ago

Prefix array values with keys and retrieve as a glued string, the original array remains unchanged. I used this to create some SQL queries from arrays.

```
<?php

function array_implode_prefix($outer_glue, $arr, $inner_glue, $prefix=false){
    array_walk( $arr , "prefix", array($inner_glue, $prefix) );
    return implode($outer_glue, $arr);
}

function prefix(&$value, $key, array $additional){
    $inner_glue = $additional[0];
    $prefix = isset($additional[1])? $additional[1] : false;
    if($prefix === false) $prefix = $key;

    $value = $prefix.$inner_glue.$value;
}

//Example 1:
$order_by = array("3"=>"ASC", "2"=>"DESC", "7"=>"ASC");
echo array_implode_prefix(",", $order_by, " ");
//Output: 3 ASC,2 DESC,7 ASC

//Example 2:
$columns = array("product_id", "category_id", "name", "description");
$table = "product";

echo array_implode_prefix(", ", $columns, ".", $table);
//Output:product.product_id, product.category_id, product.name, product.description

//Example 3 (function prefix) won't really be used on its own
$pre= "vacation";
$value = "lalaland";
prefix($value, $pre, array("."));
echo $value;

//Output: vacation.lalaland

?>
```

up
down
4
http://alex.moutonking.com/wordpress ¶
11 years ago

For completeness one has to mention the possibility of using this function with PHP 5.3 closures:

```
<?php
$names = array("D\'Artagnan", "Aramis", "Portos");
array_walk($names, function(&$n) {
  $n = stripslashes($n);
```

```
});
?>
```

The trap with array_walk being it doesn't return the array, instead it's modified by reference.

up
down
4

arekandrei at yandex dot ru ¶
12 years ago

You can use lambda function as a second parameter:

```
<?php
array_walk($myArray, function(&$value, $key){
    // if you want to change array values then "&" before the $value is mandatory.
});
?>
```

Example (multiply positive values by two):

```
<?php
$myArray = array(1, 2, 3, 4, 5);

array_walk($myArray, function(&$value, $index){
    if ($value > 0) $value *= 2;
});
?>
```

up
down
2

diyism ¶
14 years ago

When i pass the third parameter by reference in php5.2.5,
happened this: Warning: Call-time pass-by-reference has been deprecated - argument passed by value...

And to set allow_call_time_pass_reference to true in php.ini won't work, according to http://bugs.php.net/bug.php?id=19699 thus to work around:

```
<?php
array_walk($arrChnOut, create_function('&$v, $k, $arr_rtn', 'if ($k{0}!="_") {$arr_rtn[0]["_".$v[\'ID\']]=$v; unset($arr_rtn[0][$k]);}'), array(&$arrChnOut));
?>
```

up
down
6

@jfredys ¶
11 years ago

I was looking for trimming all the elements in an array, I found this as the simplest solution:

```
<?php
array_walk($ids, create_function('&$val', '$val = trim($val);'));
?>
```

up
down
2

manuscle at gmail dot com ¶
10 years ago

example with closures, checking and deleting value in array:

```
<?php
$array = array('foo' => 'bar', 'baz' => 'bat');

array_walk($array, function($val,$key) use(&$array){
    if ($val == 'bar') {
        unset($array[$key]);
    }
});

var_dump($array);
```

up
down
1
emre ¶
1 year ago

You can change the key or value with array_walk if you use the temporal returned array in global inside the function. For example:

```
$array = ['a'=>10, 'b'=>20];
$sequence = array ();

$newArray = array_values(array_walk($array, 'fn'));

function fn(&$val,$key){

global $sequence;

$sequence [] = $val;

}
```

No need to concern about the place of the internal pointer for the baby array. You have now rewinded, 0 based new array, string key one instead.

up
down
0
christopher at crmldnrs dot com ¶
5 months ago

```
public function big_endian_array_walk(array $array, $callback) {
      end($array);
      for($i=sizeof($array);$i>0;$i--) {
        $key = key($array);
        $value = array_pop($array);
        if(preg_match('/^[a-zA-Z_\x80-\xff][a-zA-Z0-9_\x80-\xff]*$/', $value)) {
          call_user_func_array($callback, [$value, $key]);
        }
      }
    }
```

up
down
0
christopher at crmldnrs dot com ¶
5 months ago

```
public function big_endian_array_walk(array $array, $callback) {
      end($array);
      for($i=sizeof($array);$i>0;$i--) {
```

```
        $key = key($array);
        $value = array_pop($array);
        if(preg_match('/^[a-zA-Z_\x80-\xff][a-zA-Z0-9_\x80-\xff]*$/', $value)) {
          call_user_func_array($callback, [$value, $key]);
        }
      }
    }
```

I just wanted to walk from the end to the beginning.

up
down
0

mystral77 at gmail dot com ¶

2 years ago

Hello,

If you want to add values with same key from two arrays :

```
<?php
function add(&$item,$key,$search) {
    $item += (is_array($search))?((isset($search[$key]))?$search[$key]:0):0;
}

$a = ["orange" => 2, "banana" => 3, "apple" => 1];
$b = ["orange" => 1, "apple" => 4];

array_walk($c,"add",$b);

echo "<pre>".print_r($c,true)."</pre>";
?>
```

This will output:

```
"orange" => 3,
"banana" => 3,
"apple" => 5
```

up
down
-1

vrrivaro at YESIUSEGMAIL dot gmail dot SO dot com ¶

5 years ago

The output of the example is only correct if viewed through a web browser. If you pass it through to PHP-CLI, you will get to see the additional HTML line breaks, however.

up
down
-1

espertalhao04 at hotmail dot com ¶

9 years ago

here is a simple and yet easy to use implementation of this function.
the 'original' function has the problem that you can't unset a value.
with my function, YOU CAN!

```
<?php
function array_walk_protected(&$a,$s,$p=null)
{
    if(!function_exists($s)||!is_array($a))
    {
        return false;
```

```
    }

    foreach($a as $k=>$v)
    {
        if(call_user_func_array($s,array(&$a[$k],$k,$p))===false)
        {
            unset($a[$k]);
        }
    }
}

function get_name(&$e,$i,$p)
{
    echo "$i: $e<br>";
    return false;
}

$m=array('d'=>'33','Y'=>55);

array_walk_protected($m,'get_name');

var_dump($m); //returns array(0) { }
?>
```

i called it array_walk_protected because it is protected against the unexpected behavior of unsetting the value with the original function.

to delete an element, simply return false!!!
nothing else is needed!
unsetting $e, under your created function, will keep the same array as-is, with no changes!

by the way, the function returns false if $a is not array or $s is not a string!

limitations: it only can run user defined functions.
i hope you like it!

up
down
-1

el_porno at web dot de ¶
17 years ago

You want to get rid of the whitespaces users add in your form fields...?
Simply use...:

```
class SomeVeryImportantClass
{
...
    public function mungeFormData(&$data)
    {
        array_walk($data, array($this, 'munge'));
    }

    private function munge(&$value, &$key)
    {
        if(is_array($value))
        {
            $this->mungeFormData($value);
        }
        else
```

```
        {
            $value = trim($value);
        }
    }
...
}

so...
$obj = new SomeVeryImportantClass;
$obj->mungeFormData($_POST);
```

___
eNc
up
down
-2
gold[at]evolved.net.nz ¶
9 years ago
For all those people trying to shoe-horn trim() into array_walk() and have found all these tricks to work around the issue with array_walk() passing 2 parameters to the callback...

Check out array_map().

http://php.net/array_map

It's all sorts of win.

For the record. I'm one of these people and after 15 years of php development I'm pleased to say that there's still things I'm learning. :) I just found out about array_map() myself...
up
down
-3
jerk at yoosic dot de ¶
15 years ago
if you want to modify every value of an multidimensional array use this function used here:

```
<?php

$array = array (1=>1, 2=> 2, 3 => array(1=>11, 2=>12, 3=>13));
$text = "test";

function modarr(&$array, $text) {
        foreach ($array as $key => $arr) {
                if(is_array($arr)) $res[$key] = modarr(&$arr,$text);
                // modification function here
                else $res[$key] = $arr.$text;
                }
        return $res;
}

$erg = modarr($array, $text);

print_r($erg);

?>
```

result will be_

```
<?php
```

```
Array ( [1] => 1test [2] => 2test [3] => Array ( [1] => 11test [2] => 12test [3] => 13test ) )
?>
```

up
down
-2
peterzuzek AT gmail DOT com ¶
14 years ago

I had some problems using this function - it didn't want to apply PHP-defined functions. So I decided to write my own - here it is. I had to use some generic-programming skills, didn't really checked the speed (I think it could be slow)... I believe it could be much better, but I don't know, how - well, I guess multiple array support and recursion would be nice. So?

Prototype:
bool arrayWalk(array &$arry, callback $callback, mixed $params=false)

```
<?php

    function arrayWalk(&$arry, $callback, $params=false) {
        $P=array(""); // parameters
        $a=""; // arguement string :)

        if($params !== false) { // add parameters
            if(is_array($params)) { // multiple additional parameters
                foreach($params as $par)
                    { $P[]=$par; }
            }
            else // just one additional
                { $P[]=$params; }
        }

        for( // create the arguement string
            $i=0; isset($P[$i]); ++$i
        )
            { $a.='$'.chr($i + 97).', '; } // random argument names

        $a=substr($a, 0, -2); // to get rid of the last comma and two spaces

        $func=create_function($a, 'return '.$callback.'('.$a.');'); // the generic function

        if(is_callable($func)) {
            for( // cycle through array
                $i=0; isset($arry[$i]); ++$i
            ) {
                $P[0]=$arry[$i]; // first element must be the first argument - array value
                $arry[$i] = call_user_func_array($func, $P); // assign the new value obtained by
the generic function
            }
        }
        else
            { return false; } // failure - function not callable

        return true; // success!
    } // arrayWalk()

?>
```

One big problem I've noticed so far - for example, if you wanted to use str_replace on the array, you'd fail - simply because of the arguement order of str_replace, where the string modified is

the third arguement, not the first as arrayWalk requires.

So, still some work left...

up
down
-3

Enlightened One ¶
17 years ago

Beware that "array ($this, method)" construct. If you're wanting to alter members of the "$this" object inside "method" you should construct the callback like this:

```
$callback[] = &$this;
$callback[] = method;
array_walk ($input, $callback);
```

Creating your callback using the array() method as suggested by "appletalk" results in a copy of $this being passed to method, not the original object, therefor any changes made to the object by method will be lost when array_walk() returns. While you could construct the callback with "array(&$this, method)", I believe this relies on the deprecated runtime pass-by-reference mechanism which may be removed in future releases of PHP. Better to not create a dependence on that feature now than having to track it down and fix it in the future.

up
down
-4

tufanbarisyildirim at gmail dot com ¶
11 years ago

Filter an array by using key.

```
<?php
    $product_1 = 'test';
    $product_2 = 'test 2';

    function array_key_filter($array,$callback = 'trim')
    {
        $filtered = array();
        array_walk($array,function ($degeri,$degisken_adi) use (&$filtered,$callback)
        {
            if($callback($degisken_adi))
            {
                $filtered[$degisken_adi] =  $degeri;
            }
        });

        return $filtered;
    }

    #using

    $degiskenler = array_key_filter(get_defined_vars(),function($key)
    {
        return strpos($key,'product_') === 0;
    });

    print_r($degiskenler);
?>
output:

Array
```

```
(
    [product_1] => test
    [product_2] => test 2
)
```

+ 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