array_reduce

(PHP 4 >= 4.0.5, PHP 5, PHP 7, PHP 8)

array_reduce — Reduce iterativamente un array a un solo valor usando una función llamada de retorno

Descripción ¶

array_reduce(array `$array`, callable `$callback`, mixed `$initial` = **`null`**): mixed

array_reduce() aplica iterativamente la función `callback` a los elementos de `array`, con el propósito de reducir el array a un solo valor.

Parámetros ¶

`array`

El array de entrada.

`callback`

callback(mixed `$carry`, mixed `$item`): mixed

`carry`

Conserva el valor de retorno de la iteración anterior; en el caso de que sea la primera iteración, conservará el valor de `initial`.

`item`

Conserva el valor de la iteración actual.

`initial`

Si el parámetro opcional `initial` está disponible, será usado al comienzo del proceso, o como un resultado final en caso de que el array esté vacío.

Valores devueltos ¶

Devuelve el valor resultante.

Si el array está vacío y no se proporciona el parámetro `initial`, **array_reduce()** devuelve **`null`**.

Historial de cambios ¶

Versión	Descripción
5.3.0	Se cambió el parámetro `initial` para permitir mixed, anteriormente era integer.

Ejemplos ¶

Ejemplo #1 Ejemplo de array_reduce()

```php
<?php
function suma($carry, $item)
{
    $carry += $item;
    return $carry;
}

function producto($carry, $item)
{
    $carry *= $item;
    return $carry;
}

$a = array(1, 2, 3, 4, 5);
$x = array();

var_dump(array_reduce($a, "suma")); // int(15)
var_dump(array_reduce($a, "producto", 10)); // int(1200), ya que: 10*1*2*3*4*5
var_dump(array_reduce($x, "suma", "No hay datos a reducir")); // string(22) "No hay datos a reducir"
?>
```

Ver también ¶

- array_filter() - Filtra elementos de un array usando una función de devolución de llamada
- array_map() - Aplica la retrollamada a los elementos de los arrays dados
- array_unique() - Elimina valores duplicados de un array
- array_count_values() - Cuenta todos los valores de un array

+ add a note

User Contributed Notes 18 notes

up
down
123
Hayley Watson ¶
15 years ago

```
To make it clearer about what the two parameters of the callback are for, and what "reduce to a
single value" actually means (using associative and commutative operators as examples may obscure
this).

The first parameter to the callback is an accumulator where the result-in-progress is effectively
assembled. If you supply an $initial value the accumulator starts out with that value, otherwise
it starts out null.
The second parameter is where each value of the array is passed during each step of the reduction.
The return value of the callback becomes the new value of the accumulator. When the array is
exhausted, array_reduce() returns accumulated value.

If you carried out the reduction by hand, you'd get something like the following lines, every one
of which therefore producing the same result:
```

```
<?php
array_reduce(array(1,2,3,4), 'f',         99             );
array_reduce(array(2,3,4),   'f',       f(99,1)          );
array_reduce(array(3,4),     'f',     f(f(99,1),2)       );
array_reduce(array(4),       'f',   f(f(f(99,1),2),3)    );
array_reduce(array(),        'f', f(f(f(f(99,1),2),3),4) );
f(f(f(f(99,1),2),3),4)
?>

If you made function f($v,$w){return "f($v,$w)";} the last line would be the literal result.

A PHP implementation might therefore look something like this (less details like error checking
and so on):
<?php
function array_reduce($array, $callback, $initial=null)
{
    $acc = $initial;
    foreach($array as $a)
        $acc = $callback($acc, $a);
    return $acc;
}
?>
```

up
down
64
directrix1 at gmail dot com ¶
7 years ago

```
So, if you were wondering how to use this where key and value are passed in to the function. I've
had success with the following (this example generates formatted html attributes from an
associative array of attribute => value pairs):

<?php

    // Attribute List
    $attribs = [
        'name' => 'first_name',
        'value' => 'Edward'
    ];

    // Attribute string formatted for use inside HTML element
    $formatted_attribs = array_reduce(
        array_keys($attribs),                       // We pass in the array_keys instead of the
array here
        function ($carry, $key) use ($attribs) {    // ... then we 'use' the actual array here
            return $carry . ' ' . $key . '="' . htmlspecialchars( $attribs[$key] ) . '"';
        },
        ''
    );

echo $formatted_attribs;

?>

This will output:
name="first_name" value="Edward"
```

up
down
51

souzacomprog at gmail dot com ¶

2 years ago

```
Sometimes we need to go through an array and group the indexes so that it is easier and easier to
extract them in the iteration.

<?php

$people = [
    ['id' => 1, 'name' => 'Hayley'],
    ['id' => 2, 'name' => 'Jack', 'dad' => 1],
    ['id' => 3, 'name' => 'Linus', 'dad'=> 4],
    ['id' => 4, 'name' => 'Peter' ],
    ['id' => 5, 'name' => 'Tom', 'dad' => 4],
];

$family = array_reduce($people, function($accumulator, $item) {
    // if you don't have a dad you are probably a dad
    if (!isset($item['dad'])) {
        $id = $item['id'];
        $name = $item['name'];
        // take the children if you already have
        $children = $accumulator[$id]['children'] ?? [];
        // add dad
        $accumulator[$id] = ['id' => $id, 'name' => $name,'children' => $children];
        return $accumulator;
    }

    // add a new dad if you haven't already
    $dad = $item['dad'];
    if (!isset($accumulator[$dad])) {
        // how did you find the dad will first add only with children
        $accumulator[$dad] = ['children' => [$item]];
        return $accumulator;
    }

    //  add a son to his dad who has already been added
    //  by the first or second conditional "if"

    $accumulator[$dad]['children'][] = $item;
    return $accumulator;
}, []);

var_export(array_values($family));

?>

OUTPUT

array (
  0 =>
  array (
    'id' => 1,
    'name' => 'Hayley',
    'children' =>
    array (
      0 =>
      array (
        'id' => 2,
        'name' => 'Jack',
        'dad' => 1,
      ),
    ),
  ),
  1 =>
  array (
    'id' => 4,
    'name' => 'Peter',
    'children' =>
    array (
      0 =>
      array (
        'id' => 3,
        'name' => 'Linus',
        'dad' => 4,
      ),
      1 =>
      array (
        'id' => 5,
        'name' => 'Tom',
        'dad' => 4,
      ),
    ),
  ),
)
```

```
<?php
$array = [
  [
    "menu_id" => "1",
    "menu_name" => "Clients",
    "submenu_name" => "Add",
    "submenu_link" => "clients/add"
  ],
  [
    "menu_id" => "1",
    "menu_name" => "Clients",
    "submenu_name" => "List",
    "submenu_link" => "clients"
  ],
  [
    "menu_id" => "2",
    "menu_name" => "Products",
    "submenu_name" => "List",
    "submenu_link" => "products"
  ],
];

//Grouping submenus to their menus

$menu =  array_reduce($array, function($accumulator, $item){
  $index = $item['menu_name'];

  if (!isset($accumulator[$index])) {
    $accumulator[$index] = [
      'menu_id' => $item['menu_id'],
      'menu_name' => $item['menu_name'],
      'submenu' => []
    ];
  }

  $accumulator[$index]['submenu'][] = [
    'submenu_name' => $item['submenu_name'],
    'submenu_link' => $item['submenu_link']
  ];

  return $accumulator;
}, []);

var_export(array_values($menu));

?>

OUTPUT

array (
  0 =>
  array (
    'menu_id' => '1',
    'menu_name' => 'Clients',
    'submenu' =>
    array (
      0 =>
      array (
        'submenu_name' => 'Add',
        'submenu_link' => 'clients/add',
      ),
      1 =>
      array (
        'submenu_name' => 'List',
        'submenu_link' => 'clients',
      ),
    ),
  ),
  1 =>
  array (
    'menu_id' => '2',
    'menu_name' => 'Products',
    'submenu' =>
    array (
      0 =>
      array (
        'submenu_name' => 'List',
        'submenu_link' => 'products',
      ),
    ),
  ),
)
```

up
down
6
849330489 at qq dot com ¶
3 years ago

```
The first parameter $array can be also be functions, which produces very interesting and powerful
result, which can be used to make an union of middlewares.
```

```
<?php

$f1 = function($x, $f){
    echo 'middleware 1 begin.'.PHP_EOL;
    $x += 1;
    $x = $f($x);
    echo 'middleware 1 end.'.PHP_EOL;
    return $x;
};


$f2 = function($x, $f){
    echo 'middleware 2 begin: '.PHP_EOL;
    $x += 2;
    $x = $f($x);
    echo 'middleware 2 end.'.PHP_EOL;
    return $x;
};

$respond = function($x){
    echo 'Generate some response.'.PHP_EOL;
    return $x;
};

$middlewares = [$f1, $f2];
$initial = $respond;
$foo = array_reduce($middlewares, function($stack, $item){
    return function($request) use ($stack, $item){
        return $item($request, $stack);
    };
}, $initial);

$x = 1;
echo $foo($x);

?>

//output:
middleware 2 begin:
middleware 1 begin.
Generate some response.
middleware 1 end.
middleware 2 end.
4
```

up
down
14
magnesium dot oxide dot play+php at gmail dot com ¶
8 years ago
You can reduce a two-dimensional array into one-dimensional using array_reduce and array_merge. (PHP>=5.3.0)

```
<?php

$two_dimensional = array();
$two_dimensional['foo'] = array(1, 2, 3);
$two_dimensional['bar'] = array(4, 5, 6);
```

```
$one_dimensional = array_reduce($two_dimensional, 'array_merge', array());
# becomes array(1, 2, 3, 4, 5, 6)
```

up
down
14
***Altreus* ¶**
8 years ago

```
You can effectively ignore the fact $result is passed into the callback by reference. Only the
return value of the callback is accounted for.

<?php

$arr = [1,2,3,4];

var_dump(array_reduce(
    $arr,
    function(&$res, $a) { $res += $a; },
    0
));

# NULL

?>

<?php

$arr = [1,2,3,4];

var_dump(array_reduce(
    $arr,
    function($res, $a) { return $res + $a;  },
    0
));

# int(10)
?>

Be warned, though, that you *can* accidentally change $res if it's not a simple scalar value, so
despite the examples I'd recommend not writing to it at all.
```

up
down
9
***ruslan dot zavackiy at gmail dot com* ¶**
10 years ago

```
If you want something elegant in your code, when dealing with reducing array, just unshift first
element, and use it as initial, because if you do not do so, you will + first element with first
element:

<?php
$arr = array(
    array('min' => 1.5456, 'max' => 2.28548, 'volume' => 23.152),
    array('min' => 1.5457, 'max' => 2.28549, 'volume' => 23.152),
    array('min' => 1.5458, 'max' => 2.28550, 'volume' => 23.152),
    array('min' => 1.5459, 'max' => 2.28551, 'volume' => 23.152),
    array('min' => 1.5460, 'max' => 2.28552, 'volume' => 23.152),
);
```

```
$initial = array_shift($arr);

$t = array_reduce($arr, function($result, $item) {
    $result['min'] = min($result['min'], $item['min']);
    $result['max'] = max($result['max'], $item['max']);
    $result['volume'] += $item['volume'];

    return $result;
}, $initial);
?>
```

up
down
15
php at keith tyler dot com ¶
12 years ago

```
If you do not provide $initial, the first value used in the iteration is NULL. This is not a
problem for callback functions that treat NULL as an identity (e.g. addition), but is a problem
for cases when NULL is not identity (such as boolean context).

Compare:

<?php
function andFunc($a, $b) {
  return $a && $b;
}
$foo = array(true, true, true);
var_dump(array_reduce($foo, "andFunc"));
?>

returns false! One would expect that it would return true because `true && true && true == true`!

Adding diagnostic output to andFunc() shows that the first call to andFunc is with the arguments
(NULL, true). This resolves to false (as `(bool) null == false`) and thereby corrupts the whole
reduction.

So in this case I have to set `$initial = true` so that the first call to andFunc() will be (true,
true). Now, if I were doing, say, orFunc(), I would have to set `$initial = false`. Beware.

Note that the "rmul" case in the example sneakily hides this defect! They use an $initial of 10 to
get `10*1*2*3*4*5 = 12000`. So you would assume that without an initial, you would get `1200/10 =
120 = 1*2*3*4*5`. Nope! You get big fat zero, because `int(null)==0`, and `0*1*2*3*4*5 = 0`!

I don't honestly see why array_reduce starts with a null argument. The first call to the callback
should be with arguments ($initial[0],$initial[1]) [or whatever the first two array entries are],
not (null,$initial[0]). That's what one would expect from the description.

Incidentally this also means that under the current implementation you will incur `count($input)`
number of calls to the callback, not `count($input) - 1` as you might expect.
```

up
down
4
kon ¶
9 years ago

```
Walking down related object's properties using array_reduce:

<?php
  $a=new stdClass;
  $a->b=new stdClass;
  $a->b->c="Hello World!\n";

  $reductionPath=array("b","c");

  print_r(
    array_reduce(
      $reductionPath,
      function($result, $item){
        return $result->$item;
      },
      $a
    )
  );
?>
```

up
down
2
cwu at nolo dot com ¶
7 years ago

The single value returned by array_reduce() can be an array -- as illustrated in the following example:

```
<?php
# calculate the average of an array
function calculate_sum_and_count($sum_and_count, $item)
{
  list($sum, $count) = $sum_and_count;
  $sum += $item;
  $count += 1;
  return [$sum, $count];
}

$a = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
$initial_sum_and_count = [0, 0];
list($sum, $count) = array_reduce($a, "calculate_sum_and_count", $initial_sum_and_count);
echo $sum / $count;
?>
```

up
down
2
bdechka at yahoo dot ca ¶
15 years ago

The above code works better this way.

```
<?php
function reduceToTable($html, $p) {
   $html .= "<TR><TD><a href=\"$p.html\">$p</a></td></tr>\n";
   return $html;
}

$list = Array("page1", "page2", "page3");

$tab = array_reduce($list, "reduceToTable");
echo "<table>".$tab . "</table>\n";
?>
```

up
down
1
Seanj.jcink.com ¶

16 years ago

The code posted below by bishop to count the characters of an array is simply... erm... well useless to me...

```
$array=Array("abc","de","f");
strlen(implode("",$array)); //6
```

works; and is much smaller. Probably much faster too.

up
down
1
yuki [dot] kodama [at] gmail [dot] com ¶
15 years ago

This code will reduce array deeply.

```
<?php
function print_s($s) {
    return is_null($s) ? "NULL" : (is_array($s) ? "Array" : ($s ? "TRUE" : "FALSE"));
}
function r_and_dp($a, $b) {
    echo "phase1:" . print_s($a) . "," . print_s($b) . "<br>\n";
    if(is_array($a)) {
        $a = array_reduce($a, "r_and_dp");
    }
    if(is_array($b)) {
        $b = array_reduce($b, "r_and_dp");
    }
    echo "phase2:" . print_s($a) . "," . print_s($b) . "<br>\n";
    $a = is_null($a) ? TRUE : $a;
    $b = is_null($b) ? TRUE : $b;
    echo "phase3:" . print_s($a) . "," . print_s($b) . "<br>\n";
    return $a && $b;
}
$bools = array(TRUE, array(FALSE, TRUE), TRUE);
echo print_s(array_reduce($bools, "r_and_dp")) . "<br>\n";

// result: FALSE
?>
```

When using boolean, you have to carefully set an "initial" argument.

```
<?php
function r_or_dp($a, $b) {
    if(is_array($a)) {
        $a = array_reduce($a, "r_or_dp");
    }
    if(is_array($b)) {
        $b = array_reduce($b, "r_or_dp");
    }
    return (is_null($a) ? FALSE : $a) || (is_null($b) ? FALSE : $b);
}
?>
```

up
down
1
itsunclexo at gmail dot com ¶
9 months ago

```
Let's see an example of array_reduce() to get the frequency of letters.

<?php

$items = "Hello";

$frequencies = array_reduce(str_split($items),
    function($result, $item) {
        if (isset($result[$item])) {
            $result[$item] += 1;
        } else {
            $result[$item] = 1;
        }
        return $result;
    },
    [] // note the initial is an array
);

print_r($frequencies);

?>

and output should be like:

Array
(
    [H] => 1
    [e] => 1
    [l] => 2
    [o] => 1
)
```

up
down
1

Julian Sawicki ¶

2 years ago

```
Array reduce offers a way to transform data.
Please look at the array below. The array has 4 nested array's.
The nested array's have the same keys. Only the value is different.

This code transforms the whole array. See below.

$array = array(
    0 => array('id' => '100', 'name' => 'Henk', 'age' => '30'),
    1 => array('id' => '101', 'name' => 'Piet', 'age' => '33'),
    2 => array('id' => '102', 'name' => 'Wim', 'age' => '43'),
    3 => array('id' => '103', 'name' => 'Jaap', 'age' => '53'),
);

$arr = array_reduce($array, function($carry, $item){

    $arr = array(
        'id' => $item['id'],
        'value' => $item['name'],
    );

    $id = $item['id'];
    $carry[$id] = $arr;
    return $carry;
}, array());

var_dump($arr);


// OUTPUT

array (size=4)
100 => array (size=2)
   'id' => string '100' (length=3)
   'value' => string 'Henk' (length=4)
101 => array (size=2)
   'id' => string '101' (length=3)
   'value' => string 'Piet' (length=4)
102 => array (size=2)
   'id' => string '102' (length=3)
   'value' => string 'Wim' (length=3)
103 => array (size=2)
   'id' => string '103' (length=3)
   'value' => string 'Jaap' (length=4)
```

up
down
-5
galley dot meng at gmail dot com ¶
5 years ago

```
If your array has string keys, you can reduce a two-dimensional array into one-dimensional using
array_reduce, array_merge and array_values. (PHP>=5.3.0)

<?php

$two_dimensional = array();

$two_dimensional['foo'] = array('a' => 1, 'b' => 2, 'c' => 3);
$two_dimensional['bar'] = array('a' => 4, 'b' => 5, 'c' =>6);

$one_dimensional = array_reduce($two_dimensional, 'array_merge', array());

$one_dimensional = array_reduce($two_dimensional, function ($one_dimensional, $value) {
    return array_merge($one_dimensional, array_values($value));
}, array());

# becomes array(1, 2, 3, 4, 5, 6)
```

up
down
-15
aiadfaris at yahoo dot de ¶
8 years ago

```
notice to function array_reduce()
I suppose the function rsum in the example 1 so it is not correct,
but
$ v + = $ w;
will output 15
```

up
down
-18
aiadfaris at yahoo dot de ¶

8 years ago

```
notice to function array_reduce()
I suppose the function rsum in the example 1 so it is not correct,
but
$ v + = $ w;
will output 15
```

+ add a note