call_user_func_array

(PHP 4 >= 4.0.4, PHP 5, PHP 7, PHP 8)

call_user_func_array — Llamar a una llamada de retorno con un array de parámetros

Descripción ¶

call_user_func_array(callable `$callback`, array `$param_arr`): mixed

Llama a la llamada de retorno dada por el primer parámetro `callback` con los parámetros de `param_arr`.

Parámetros ¶

`callback`

La función llamable (callable) a llamar.

`param_arr`

Los parámetros a pasar a la llamada de retorno, como matriz indexada.

Valores devueltos ¶

Devuelve el valor devuelto por la llamada de retorno, o **`false`** en caso de error.

Historial de cambios ¶

Versión	Descripción
5.3.0	La interpretación de palabras clave orientadas a objetos como `parent` y `self` ha cambiado. Anteriormente al llamarlas usando la sintaxis de dobles dos puntos emitiría una advertencia **`E_STRICT`** porque eran interpretadas como estáticas.

Ejemplos ¶

Ejemplo #1 Ejemplo de call_user_func_array()

```php
<?php
function foobar($arg, $arg2) {
    echo __FUNCTION__, " obtuvo $arg y $arg2\n";
}
```

```
class foo {
    function bar($arg, $arg2) {
        echo __METHOD__, " obtuvo $arg y $arg2\n";
    }
}


// Llamar a la función foobar() con 2 argumentos
call_user_func_array("foobar", array("uno", "dos"));

// Llamar al método $foo->bar() con 2 argumentos
$foo = new foo;
call_user_func_array(array($foo, "bar"), array("tres", "cuatro"));
?>
```

El resultado del ejemplo sería algo similar a:

```
foobar obtuvo uno y dos
foo::bar obtuvo tres y cuatro
```

Ejemplo #2 call_user_func_array() usando un nombre de espacio de nombres

```
<?php

namespace Foobar;

class Foo {
    static public function prueba($nombre) {
        print "¡Hola {$nombre}!\n";
    }
}

// A partir de PHP 5.3.0
call_user_func_array(__NAMESPACE__ .'\Foo::prueba', array('Gema'));

// A partir de PHP 5.3.0
call_user_func_array(array(__NAMESPACE__ .'\Foo', 'prueba'), array('Pedro'));

?>
```

El resultado del ejemplo sería algo similar a:

```
¡Hola Gema!
¡Hola Pedro!
```

Ejemplo #3 Usar una función lambda

```
<?php

$func = function($arg1, $arg2) {
    return $arg1 * $arg2;
};

var_dump(call_user_func_array($func, array(2, 4))); /* A partir de PHP 5.3.0 */

?>
```

El resultado del ejemplo sería:

```
int(8)
```

Notas ¶

Nota:

Antes de PHP 5.4, las variables referenciadas de `param_arr` son pasadas a la función por referencia, sin tener en cuenta si la función espera que el parámetro respectivo sea pasado por referencia. Esta forma de paso por refernia en tiempo de llamda no emite un aviso de obsolescencia, pero, no obstante, está obsoleta, y ha sido eliminada en PHP 5.4. Además, esto no se aplica a funciones internas, para las que la firma de la funcion es aceptada. Pasar por valor cuando la función espera un parámetro por referencia resulta en una advertencia, y call_user_func() devolverá **`false`** (existe, sin embargo, una excepción para valores pasados con cuenta de referencia = 1, como en los literales, ya que pueden convertirse en referencias sin efectos dañinos — aunque también sin escribir que esos valores tengan ningún efecto —; sin embargo, no confíe en este comportamiento, ya que la cuenta de referencia es un detalle de implementación y la solidez de este comportamiento es cuestinable).

Nota:

Las funciones de retorno de llamada que se registran con funciones como call_user_func() y **call_user_func_array()** no se llamarán si se produce una excepción en la función de retorno previa.

Ver también ¶

- call_user_func() - Llamar a una llamada de retorno dada por el primer parámetro
- ReflectionFunction::invokeArgs() - Invoca a la función con argumentos
- ReflectionMethod::invokeArgs() - Invoca un método con argumentos

+ add a note

User Contributed Notes 38 notes

up
down
34
sebastian dot rapetti at tim dot it ¶
1 year ago
```
Using PHP 8, call_user_func_array call callback function using named arguments if an array with
keys is passed to $args parameter, if the array used has only values, arguments are passed
positionally.

<?php

function test(string $param1, string $param2): void
{
    echo $param1.' '.$param2;
}

$args = ['hello', 'world'];
//hello world
call_user_func_array('test', $args);

$args = ['param2' => 'world', 'param1' => 'hello'];
//hello world
call_user_func_array('test', $args);

$args = ['unknown_param' => 'hello', 'param2' => 'world'];
```

```
//Fatal error: Uncaught Error: Unknown named parameter $unknown_param
call_user_func_array('test', $args);
?>
```

up
down
52
admin at torntech dot com ¶
7 years ago

```
As of PHP 5.6 you can utilize argument unpacking as an alternative to call_user_func_array, and is
often 3 to 4 times faster.

<?php
function foo ($a, $b) {
     return $a + $b;
}

$func = 'foo';
$values = array(1, 2);
call_user_func_array($func, $values);
//returns 3

$func(...$values);
//returns 3
?>

Benchmarks from https://gist.github.com/nikic/6390366
cufa   with 0 args took 0.43453288078308
switch with 0 args took 0.24134302139282
unpack with 0 args took 0.12418699264526
cufa   with 5 args took 0.73408579826355
switch with 5 args took 0.49595499038696
unpack with 5 args took 0.18640494346619
cufa   with 100 args took 5.0327250957489
switch with 100 args took 5.291127204895
unpack with 100 args took 1.2362589836121
```

up
down
3
simopelle04 at gmail dot com ¶
1 year ago

```
Since PHP 5.6 you can use the spread operator (argument unpacking) instead of
call_user_func_array().

    public function __call($name, $arguments) {
        if(method_exists($this->foo, $name))
            return call_user_func_array(array($this->foo, $name), $arguments);    //calls function
from inside $this->foo
        else
            throw new BadMethodCallException('Call to undefined method ' . static::class .
"::$name()", 0);    //the method does not exist on $this->foo
    }

is the same as

    public function __call($name, $arguments) {
        if(method_exists($this->foo, $name))
            return $this->foo->$name(...$arguments);    //calls function from inside $this->foo
        else
```

```
            throw new BadMethodCallException('Call to undefined method ' . static::class .
"::$name()", 0);    //the method does not exist on $this->foo
    }
```

up
down
20
dmitry dot revenko at businessmedia dot ru ¶
13 years ago

```
Just hope this note helps someone (I killed the whole day on issue).

If you use something like this in PHP < 5.3:
<?php call_user_func_array(array($this, 'parent::func'), $args); ?>
Such a script will cause segmentation fault in your webserver.

In 5.3 you should write it:
<?php call_user_func_array('parent::func', $args); ?>
```

up
down
6
jaxxed ¶
10 years ago

```
For anyone looking for the means to test for the first parameter before passing to this function,
look at the is_callable (http://php.net/manual/en/function.is-callable.php) variable handler.

<?php

$handler = array( 'MyClass', 'MyMethod');
$params = array(1,2,3,4);

if ( is_callable($handler) ) { call_user_func_array( $handler , $params ); }

?>
```

up
down
3
aj at ajbrown dot org ¶
12 years ago

```
Just a heads up, the second parameter MUST be an array if it's specified,  but that doesn't seem
to be enforced until ~5.3.

I just pulled my hair out with an old installation of CakePHP because it was passing NULL instead
of an empty array.
```

up
down
1
dnhuff at acm.org ¶
14 years ago

```
It appears that when PHP executes something like:

$a = array(1,2,3);
$b =& $a[1];

both $b and $a[1] are converted into references to a common value -- makes sense until you
transfer that to a call_user_func:

call_user_func_array('foo', $a);

suddenly, inside foo, the second parameter is passed by reference!
```

```
And you can't call this wrong, only another subtly of references.

Note it appears that ksort($a) will remove the reference as well as put the elements in key order
so you (probably) get what you expect. (see below on the use of a foreach ($a as &v).)
```

up
down
1
stanislav dot eckert at vizson dot de ¶
8 years ago

```
Please note, that when calling call_user_func_array() to redirect parameters between inherited
classes, you should not use $this, because $this always refers to the class which has been
instantiated. The following code even seems to crash PHP (PHP does not report error but the
process simply terminates), because the the parameters are redirected only one level up (to class
foo_bar2):

<?php

    class foo_bar1
    {
        public function __construct()
        {
            echo __CLASS__ . PHP_EOL;

            if (func_num_args() > 0)
            {
                $constructorArgs = func_get_args();
                call_user_func_array(array($this, 'parent::__construct'), $constructorArgs);
            }
            else
            {
                parent::__construct();
            }
        }
    }

    class foo_bar2 extends foo_bar1
    {
        public function __construct()
        {
            echo __CLASS__ . PHP_EOL;

            if (func_num_args() > 0)
            {
                $constructorArgs = func_get_args();
                call_user_func_array(array($this, 'parent::__construct'), $constructorArgs);
            }
            else
            {
                parent::__construct();
            }
        }
    }

    class foo_bar3 extends foo_bar2
    {
        public function __construct()
        {
            echo __CLASS__ . PHP_EOL;

            if (func_num_args() > 0)
            {
                $constructorArgs = func_get_args();
                call_user_func_array(array($this, 'parent::__construct'), $constructorArgs);
            }
            else
            {
                parent::__construct();
            }
        }
    }

    $f = new foo_bar3("abc");

?>

Instead, use the direct name of the class as string or, better, the magic constant __CLASS__ in
call_user_func_array(), like:

    call_user_func_array(array(__CLASS__, 'parent::__construct'), $constructorArgs);

Then the parameters will be correctly redirected to the lowest base class.
```

up
down
0
Anonymous ¶
7 years ago

```
$param_arr may be empty, though it can't be null.

<?php
function foo( $first = 'default1', $second = 'default2' ) {
    echo "first: '$first', second: '$second'\n";
}
call_user_func_array( 'foo', array( 'one', 'two' ) );
call_user_func_array( 'foo', array( 'only one' ) );
call_user_func_array( 'foo', array() );
call_user_func_array( 'foo', null );
?>

Output:
first: 'one', second: 'two'
first: 'only one', second: 'default2'
first: 'default1', second: 'default2'
/* error message or nothing printed depending on version */
```

up
down
-2
qwe at hi2 dot in ¶
1 year ago

```
Or you can call a function like
<?php
function foo($arg1, $arg2) {
    echo __FUNCTION__, " got $arg1 and $arg2\n";
}
function bar($arg1, $arg2) {
    echo __FUNCTION__, " got $arg and $arg2\n";
}

$run='foo';

$run('one', 'two');
```

up
down
0
Damin ¶
13 years ago

```
Those having the passing by reference issue can use this simple hack.
I´m really not sure WHY this works, but it does, and it does not make use of EVAL or other
questionable functions.
<?php
    function executeHook($name, $type='hooks'){
        $args = func_get_args();
        array_shift($args);
        array_shift($args);
        //Rather stupid Hack for the call_user_func_array();
        $Args = array();
        foreach($args as $k => &$arg){
            $Args[$k] = &$arg;
        }
        //End Hack
        $hooks = &$this->$type;
        if(!isset($hooks[$name])) return false;
        $hook = $hooks[$name];
        call_user_func_array($hook, $Args);
    }
?>

All it´s doing is copying the args ($args) into a new array ($Args) by reference, which i would
think would be identical to the original array in every way (that matters).

Note the code here is an example of usage. The actual hack is denoted by comments.
If someone knows a better alternative, by all means, i would love to see it.
```

up
down
0
thiago dot henrique dot mata at gmail dot com ¶
14 years ago

```
<?php
Class Delegate
{
    private $arrInstances = array();

    protected function addObject( $oElement )
    {
        // add one element on the end of the stack  //
        $this->arrInstances[] = $oElement;
    }

    public function __call( $strMethod, $arrParams )
    {
        // for each element in instance //
        foreach( $this->arrInstances as $oElement )
        {
            // get the class of the element //
            $strClass = get_class( $oElement );
            // get all methods of the class  //
            $arrMethods = get_class_methods( $strClass );
            // case the method exists into this class  //
            if( in_array( $strMethod , $arrMethods ) )
            {
                // prepare caller //
                $arrCaller = Array( $strClass , $strMethod );
                // return the result of the method into the object  //
                return call_user_func_array( $arrCaller, $arrParams );
            }
        }
        // any object has the method //
        // throw a exception //
        throw new Exception( " Method " . $strMethod . " not exist in this class " . get_class(
$this ) . "." );
    }
}

class Log
{
    public function sayHi()
    {
        print "hi!" . "<br/>\n";
    }

    public function sayMyName()
    {
        print "log" . "<br/>\n";
    }
}

class Other
{
    public function sayHello()
    {
        print "hello there!" . "<br/>\n";
    }

    public function sayMyName()
    {
        print "other" . "<br/>\n";
    }
}

class Example extends Delegate
{
    public function __construct()
    {
        $this->addObject( new Log() );
        $this->addObject( new Other() );
    }
}

$oExample = new Example();
$oExample->sayHi();
$oExample->sayHello();
$oExample->sayMyName();
```

```
/*
    hi!<br/>
    hello there!<br/>
    log<br/>
*/
?>
```

up
down
0
Anonymous ¶
16 years ago

```
For those wishing to implement call-by-name functionality in PHP, such as implemented e.g. in DB
apis, here's a quick-n-dirty version for PHP 5 and up
<?php
/**
* Call a user function using named instead of positional parameters.
* If some of the named parameters are not present in the original function, they
* will be silently discarded.
* Does no special processing for call-by-ref functions...
* @param string $function name of function to be called
* @param array $params array containing parameters to be passed to the function using their name
(ie array key)
*/
function call_user_func_named($function, $params)
{
    // make sure we do not throw exception if function not found: raise error instead...
    // (oh boy, we do like php 4 better than 5, don't we...)
    if (!function_exists($function))
    {
        trigger_error('call to unexisting function '.$function, E_USER_ERROR);
        return NULL;
    }
    $reflect = new ReflectionFunction($function);
    $real_params = array();
    foreach ($reflect->getParameters() as $i => $param)
    {
        $pname = $param->getName();
        if ($param->isPassedByReference())
        {
            /// @todo shall we raise some warning?
        }
        if (array_key_exists($pname, $params))
        {
            $real_params[] = $params[$pname];
        }
        else if ($param->isDefaultValueAvailable()) {
            $real_params[] = $param->getDefaultValue();
        }
        else
        {
            // missing required parameter: mark an error and exit
            //return new Exception('call to '.$function.' missing parameter nr. '.$i+1);
            trigger_error(sprintf('call to %s missing parameter nr. %d', $function, $i+1),
E_USER_ERROR);
            return NULL;
        }
    }
    return call_user_func_array($function, $real_params);
}
?>
```

up
down
-1
hong dot nguyen at k-edge dot com ¶
18 years ago

```
call_user_func_array can pass parameters as reference:

<?php
call_user_func_array(array(&$obj,$method),array(&$arg1,$arg2,$arg3))
?>

Use it as work-around for "Call-time pass-by-reference has been deprecated".
```

up
down
-1
james at gogo dot co dot nz ¶
17 years ago

```
Be aware the call_user_func_array always returns by value, as demonstrated here...

<?php
    function &foo(&$a)
    {
      return $a;
    }

    $b = 2;
    $c =& call_user_func_array('foo', array(&$b));
    $c++;
    echo $b . ' ' . $c;
?>

outputs "2 3", rather than the expected "3 3".

Here is a function you can use in place of call_user_func_array which returns a reference to the
result of the function call.

<?php
    function &ref_call_user_func_array($callable, $args)
    {
        if(is_scalar($callable))
        {
            // $callable is the name of a function
            $call = $callable;
        }
        else
        {
            if(is_object($callable[0]))
            {
                // $callable is an object and a method name
                $call = "\$callable[0]->{$callable[1]}";
            }
            else
            {
                // $callable is a class name and a static method
                $call = "{$callable[0]}::{$callable[1]}";
            }
        }

        // Note because the keys in $args might be strings
        // we do this in a slightly round about way.
        $argumentString = array();
        $argumentKeys = array_keys($args);
        foreach($argumentKeys as $argK)
        {
            $argumentString[] = "\$args[$argumentKeys[$argK]]";
        }
        $argumentString = implode($argumentString, ', ');
        // Note also that eval doesn't return references, so we
        // work around it in this way...
        eval("\$result =& {$call}({$argumentString});");
        return $result;
    }
?>
```

up
down
-1
aeolianmeson at 8n54tvv dot blitzeclipse dot com ¶
14 years ago

```
There's a possibility that call_user_func_array(), call_user_func(), and Exception::getTrace()
will cause a trace entry to not have the 'file' or 'line' elements.

Dustin Oprea
```

up
down
-1
taylor ¶
17 years ago

```
I came up with a better solution to the problem that I solve below with createObjArray that
maintains parameter type:

<?php

function createObjArray($type,$args=array()) {
     $paramstr = '';
     for ($i = 0; $i < count($args); $i++) {
           $paramstr .= '$args['.$i.'],';
     }
     $paramstr = rtrim($paramstr,',');

     return eval("return new $type($paramstr);");
}

?>

Would be good to add error checking, but it works.
```

up
down
-1
levi at alliancesoftware dot com dot au ¶
15 years ago

```
Regarding the comments below about calling parent constructors:

PHP5 with E_STRICT no longer allows calls as below:
```

```
<?php
// Causes an error with E_STRICT
call_user_func_array(array('parent', '__construct'), $args);
?>

It gives an error because you are trying to call a nonstatic function as if it was static. The
correct syntax is

<?php
// Works fine
call_user_func_array(array($this, 'parent::__construct'), $args);
?>
```

up
down
-1
gmail@asmqb7 ¶
5 years ago

```
Note that call_user_func() will completely trash debug_backtrace().

Solution:

<?php

your_function(...call_user_func())

?>

Note the 3 dots (array unpacking syntax).
```

up
down
-2
adamh at densi dot com ¶
19 years ago

```
call_user_func_array() is nifty for calling PHP functions which use variable argument length. For
example:

<?php
$array = array(
array("foo", "bar"),
array("bat", "rat"),
);

$values = call_user_func_array("array_merge", $array);

var_dump($values);
?>

/* output:
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(3) "bat"
  [3]=>
  string(3) "rat"
}
*/

The neat feature is that $array could have any number of arrays inside it.
```

up
down
-2
mrextreme at freemail dot hu ¶
12 years ago

```
If you are using PHP < 5.3 and want to call the parent class' __construct() with a variable
parameter list, use this:

<?php
public function __construct()
{
    $vArgs = func_get_args(); // you can't just put func_get_args() into a function as a parameter
    call_user_func_array(array('parent', '__construct'), $vArgs);
}
?>
```

up
down
-4
amer at o2 dot pl ¶
17 years ago

```
PLS notice that "patripaq at hotmail dot com" 's code will be valid if B EXTENDS A...
<?php
class B extends A{
...
}
?>
there>>"What I wanted to do is create an object that can manage any number and any kind of
parameters."

BUT IT IS NOT A POINT AT ALL

If you need to call just function with parameters:
call_user_func_array('Foo',$args);

If you need to call CLASS method (NOT object):
call_user_func_array(array('class', 'Foo'),$args);

If you need to call OBJECT method:
call_user_func_array(array(&$Object, 'Foo'),$args);

If you need to call method of object of object:
call_user_func_array(array(&$Object->Object, 'Foo'),$args);

If you need to call object method from within the very same object (NOT CLASS!):
call_user_func_array(array(&$this, 'Foo'),args);

The call_user_func_array ITSELF can manage any number and any kind of parameters. It can handle
ANY FUNCTION too as it is defined and that maybe partipaq wanted to manage.

What You actually need is object composition not inheritance. Make an instance from arguments.
<?php
...
class B{
   function __construct() {
        $args = func_get_args(); // Get arguments
        $this->OBJ = new A($args);
        call_user_func_array(array(&$this->OBJ, 'A'), $args );
    }
}
?>
Then there can be any number and any type of created object B parameters
```

up
down
-3
noone at example dot com ¶
12 years ago

```
For those of you that have to consider performance: it takes about 3 times as long to call the
function this way than via a straight statement, so whenever it is feasible to avoid this method
it's a wise idea to do so.

Note that eval() is about 10 times slower than a straight statement to call a function with
arguments, so this is definitely a better option than using eval() even if you only consider
performance.
```

up
down
-2
zonkiie ¶
5 years ago

```
An implementation where parameters are submitted by their name.
This function calls class functions, class methods, functions and anonymous functions.

/**
    * Calls a method, function or closure. Parameters are supplied by their names instead of their
position.
    * @param $call_arg like $callback in call_user_func_array()
    * Case1: {object, method}
    * Case2: {class, function}
    * Case3: "class::function"
    * Case4: "function"
    * Case5: closure
    * @param array $param_array A key-value array with the parameters
    * @return result of the method, function or closure
    * @throws \Exception when wrong arguments are given or required parameters are not given.
    */
function call_user_function_named_param($call_arg, array $param_array)
{
    $Func = null;
    $Method = null;
    $Object = null;
    $Class = null;
    // The cases. f means function name
    // Case1: f({object, method}, params)
    // Case2: f({class, function}, params)
    if(is_array($call_arg) && count($call_arg) == 2)
    {
        if(is_object($call_arg[0]))
        {
            $Object = $call_arg[0];
            $Class = get_class($Object);
        }
        else if(is_string($call_arg[0]))
        {
            $Class = $call_arg[0];
        }
        if(is_string($call_arg[1]))
        {
            $Method = $call_arg[1];
        }
    }
    // Case3: f("class::function", params)
    else if(is_string($call_arg) && strpos($call_arg, "::") !== FALSE)
    {
        list($Class, $Method) = explode("::", $call_arg);
    }
    // Case4: f("function", params)
    else if(is_string($call_arg) && strpos($call_arg, "::") === FALSE)
    {
        $Method = $call_arg;
    }
    // Case5: f(closure, params)
    else if(is_object($call_arg) && $call_arg instanceof \Closure)
    {
        $Method = $call_arg;
    }
    else throw new \Exception("Case not allowed! Invalid Data supplied!");
    if($Class) $Func = new \ReflectionMethod($Class, $Method);
    else $Func = new \ReflectionFunction($Method);
    $params = array();
    foreach($Func->getParameters() as $Param)
    {
        if($Param->isDefaultValueAvailable()) $params[$Param->getPosition()] = $Param-
>getDefaultValue();
        if(array_key_exists($Param->name, $param_array)) $params[$Param->getPosition()] =
$param_array[$Param->name];
        if(!$Param->isOptional() && !isset($params[$Param->getPosition()])) die("No Defaultvalue
available and no Value supplied!\r\n");
    }
    if($Func instanceof \ReflectionFunction) return $Func->invokeArgs($params);
    if($Func->isStatic()) return $Func->invokeArgs(null, $params);
    else return $Func->invokeArgs($Object, $params);
}

//Test code:

function func($arg1, $arg2 = "Jane")
{
    return $arg1 . "," . $arg2;
}

class tc1
{
    public static function func1($Arg1, $Arg2 = "HH")
    {
        return $Arg1 . " " . $Arg2;
    }
}

$func2 = function($a1, $a2)
{
   return "a1:" . $a1 . ", a2:" . $a2;
```

```
};

$ret = call_user_function_named_param('func', array("arg1"=>"Hello", "arg2"=>"Joe"));
echo "Call Return:" . print_r($ret, true) . PHP_EOL;
$ret = call_user_function_named_param(array(new tc1(), 'func1'), array("Arg1"=>"Hello",
"Arg2"=>"Joe"));
echo "Call2 Return:" . print_r($ret, true) . PHP_EOL;
$ret = call_user_function_named_param($func2, array("a1"=>"Hello", "a2"=>"Joe"));
echo "Call3 Return:" . print_r($ret, true) . PHP_EOL;
```

up
down
-2
jim at commercebyte dot com ¶
5 years ago

Before PHP 7.1.0, if you call a function without passing required arguments, a warning was generated, and the missing arguments defaulted to NULL.
In PHP 7.1.0, an ArgumentCountError is being thrown instead. (A little bit of nuisance IMHO, because it causes compatibility issues).
When you use <? call_user_func_array() ?>, same behavior applies - the exception is thrown if you missed arguments.
Here's a quick and dirty fix for it:

```
<?
function call_user_func_array_i($callbk,$args) {
    while(true) {
        try {
            return call_user_func_array($callbk,$args);
        } catch(ArgumentCountError $e) {
            $args[] = NULL;
        }
    }
}
?>
```

Replace your calls to <? call_user_func_array() ?> with <? call_user_func_array_i() ?>, and the good old behavior is back (except that it doesn't issue a warning, but you probably didn't need it anyway).
up
down
-3
Freek (at) Gruntjes.net ¶
13 years ago

I just noticed that when you use this function with parameters that need to be passed by reference it will not work.

```
<?php
function refFunc(&$var)
{
  $var .= 'bar';
}

$var = 'foo';
?>

call_user_func_array('refFunc', array($var));
echo $var;
```

will output 'foo' and not 'foobar'. Witch is logical since you are declaring a new variable with array($var) however not so obvious.
up

down
-4
Kris dot Craig at gmail dot com ¶
12 years ago

```
Many people have wondered how to effectively implement dispatch tables in PHP.  Here's my answer
to that (if you'll forgive my creative flair):

<?php

/*
* Using dispatch tables in PHP.
*
* --Kris Craig
*/

define( "YOUR_MOTHER",  1 );
define( "YOUR_FATHER",  2 );
define( "YOUR_BROTHER", 3 );
define( "YOUR_SISTER",  4 );

class MyFamily
{
    static $dispatch = array( YOUR_MOTHER => "Mom", YOUR_FATHER => "GetPrisonInmate", YOUR_BROTHER
=> "ReplaceName", YOUR_SISTER => "LazyGirl" );
    static $args = array( YOUR_MOTHER => array(), YOUR_FATHER => array( "55170-054", TRUE ),
YOUR_BROTHER => array(), YOUR_SISTER => array() );

    function GetDispatch( $fromwhere )
    {
        return call_user_func_array( array( self, self::$dispatch[$fromwhere] ),
self::$args[$fromwhere] );
    }

    function Mom()
    {
        return "Mommy loves you!";
    }

    function GetPrisonInmate( $PrisonerID, $GoodBehavior )
    {
        //Check prison records for his ID, then....

        if ( $GoodBehavior )
        {
            $parole = "APPROVED";
        }
        else
        {
            $parole = "DENIED";
        }

        return "Your father (#$PrisonerID) has $remaining years left in his sentence.  His most
recent parole application has been:  $parole";
    }

    function ReplaceName()
    {
        return "Her name is Sally now.";
    }

    function LazyGirl()
    {
        print "Your sister needs to get out more....";

        //sleep( pow( 60, 2 ) * 18 );  //You can sleep later!

        die( "Nah, I'm too tired." );
    }
}

print "Status on family member: " . MyFamily::GetDispatch( YOUR_FATHER );

?>
```

up
down
-3
wriver4 at gmail dot com ¶
6 years ago

```
Kris dot Craig at gmail dot com dispatch table does not work currently, but this does.
<?php
define("YOUR_MOTHER", 1);
define("YOUR_FATHER", 2);
define("YOUR_BROTHER", 3);
define("YOUR_SISTER", 4);

class MyFamily
{

    static $dispatch = array(
        YOUR_MOTHER => "mom",
        YOUR_FATHER => "getPrisonInmate",
        YOUR_BROTHER => "replaceName",
        YOUR_SISTER => "lazyGirl");
    static $args = array(
        YOUR_MOTHER => array(),
        YOUR_FATHER => array(
            "55170-054",
            TRUE,
            3),
        YOUR_BROTHER => array(),
        YOUR_SISTER => array());

    function getDispatch($fromwhere)
    {
        return call_user_func_array(array(
            get_class(),
            self::$dispatch[$fromwhere]), self::$args[$fromwhere]);
    }

    function mom()
    {
        return "Mommy loves you!";
    }

    function getPrisonInmate($PrisonerID, $GoodBehavior, $remaining,
            $notes = null)
    {
        //Check prison records for his ID, then....

        if ($GoodBehavior)
        {
            $parole = "APPROVED";
        }
        else
        {
            $parole = "DENIED";
        }

        return "Your father (#$PrisonerID) has $remaining years left in his sentence.  His most
recent parole application has been:  $parole";
    }

    function replaceName()
    {
        return "Her name is Sally now.";
    }

    function lazyGirl()
    {
        print "Your sister needs to get out more....";

        //sleep(pow(60, 2) * 18);  //You can sleep later!

        die("Nah, I'm too tired.");
    }

}
print "Status on family member: ".MyFamily::GetDispatch(YOUR_FATHER).'<br><br>';
```

up
down
-5
rrant (at) gmail (dot) com ¶
17 years ago

```
Just an extra for the post of amer at o2 dot pl:

If you need to call the PARENT method:
call_user_func_array(array('parent', 'method'), $args);

With that, if you need to call a constructor and/or add some extra code to the instantiation
process:

<?php
function __construct() {
    // Get the arguments
    $args = func_get_args();
    // Initialize parent with arguments
    call_user_func_array(array('parent', '__construct'), $args);
    // ... Your Code Here ...
}
?>

Note that your constructor pass all the arguments to the parent constructor and it doesn't matter
how many arguments you pass.
```

This is pretty useful for constructors with a variable number of arguments.

up
down
-4

eugene at artprime dot ru ¶

16 years ago

```
<?php
  return call_user_func_array(
    array(new ReflectionClass($className), 'newInstance'),
    $functionParameters
  );
?>
```

Look here: http://www.zend.com/zend/week/week182.php#Heading1

up
down
-4

Egor ¶

17 years ago

Note that, despite the name, this does work on builtin functions (and object methods with the array(&$obj, $method) syntax), not just user-defined functions and methods.

up
down
-5

crocodile2u at yandex dot ru ¶

16 years ago

Here is another version of createObjArray() function written here earlier by taylor.

Believing that using 'eval()' is at least "dirty", I came to the following solution (with a help of panchous - at phpclub dot ru forums). This solution utilizes the new Reflection API.

```
<?php
function & createObjArray($type, $args = array()) {
    $reflection = new ReflectionClass($type);
    $output     = call_user_func_array(array(&$reflection, 'newInstance'), $args);
    return $output;
}
?>
```

up
down
-8

richard_harrison at rjharrison dot org ¶

15 years ago

If you are thinking of using call_user_func_array to instantiate an object (see comments below using Reflection) then since v5.1.3 you can use the Reflection::newInstanceArgs() method.

```
<?php

// arguments you wish to pass to constructor of new object
$args = array('a', 'b');

// class name of new object
$className = 'myCommand';

// make a reflection object
$reflectionObj = new ReflectionClass($className);
```

```
// use Reflection to create a new instance, using the $args
$command = $reflectionObj->newInstanceArgs($args);

// this is the same as: new myCommand('a', 'b');
?>
```

up
down
-11
Brad Proctor ¶
12 years ago

```
This function is relatively slow (as of PHP 5.3.3) and if you are calling a method with a known
number of parameters it is much faster to call it this way:

$class->{$method}($param1, $param2);

vs

call_user_func_array (array($class, $method), array($param1, $param2));

But if you don't know how many parameters...

The wrapper function below is slightly faster, but the problem now is that you are making two
function calls.  One to the wrapper and one to the function.

However, If you are able to take this code out of the function and use it inline it is nearly
twice as fast (in most cases) as calling call_user_func_array natively.

<?php
function wrap_call_user_func_array($c, $a, $p) {
    switch(count($p)) {
        case 0: $c->{$a}(); break;
        case 1: $c->{$a}($p[0]); break;
        case 2: $c->{$a}($p[0], $p[1]); break;
        case 3: $c->{$a}($p[0], $p[1], $p[2]); break;
        case 4: $c->{$a}($p[0], $p[1], $p[2], $p[3]); break;
        case 5: $c->{$a}($p[0], $p[1], $p[2], $p[3], $p[4]); break;
        default: call_user_func_array(array($c, $a), $p);  break;
    }
}
?>
```

up
down
-4
Aurelien Marchand ¶
3 years ago

```
In response to  admin at torntech dot com and as shown on https://gist.github.com/nikic/6390366,
there is no good reason to use argument unpacking anymore for PHP 7.0.17, 7.1.3, 7.2.7

7.0.17
cufa with 0 args took 0.081735849380493
unpack with 0 args took 0.072259902954102
cufa with 100 args took 0.39941906929016
unpack with 100 args took 0.419842004776

7.1.3
cufa with 0 args took 0.074923992156982
unpack with 0 args took 0.07165002822876
cufa with 100 args took 0.37423300743103
```

```
unpack with 100 args took 0.41387891769409

7.2.7
cufa with 0 args took 0.064092874526978
unpack with 0 args took 0.048593997955322
cufa with 100 args took 0.35412693023682
unpack with 100 args took 0.36837601661682
```

up
down
-4
ben at benhunt dot com ¶
3 years ago

```
If you're thinking call_user_func_array has changed the array of multiple parameters to a string,
remember that it does not pass the array through to the called function as a single argument
(array), but creates one argument for each element in the array.
```

up
down
-7
james dot patrick at lifeshop dot vn ¶
7 years ago

```
I've found the solution to resolve my need while writing the str_replace function for processing
the multi array as first two arguments of str_replace built-in function(although its pass each
array of argument 1 & 2)

<?php
// custom the str_replace function
function p_str_replace($argFind, $agrReplace, $theString) {
  $needle_replace = $theString; // handle the string need to be replace
  array_walk($argFind, function($val, $key) use(&$needle_replace, $agrReplace) {
    $needle_replace = call_user_func_array('str_replace', array($val, $agrReplace[$key],
$needle_replace));
  });
  return $needle_replace;
}

// test
$the_str = array(
  'coa' => ':col: :op1: :val: AND :col: :op2: :val:'
);
$ope = array('>=', '<=');
$colsdata = array('date_start', '2015-10-14');

echo p_str_replace(array(
          array(':col:', ':val:'),
          array(':op1:', ':op2:')
        ), array(
            $colsdata,
            $ope
        ), $the_str['coa']);

echo '--------------------------------' . "\n";

// more complexity
$complex = '(:col: :op1: :val: AND :col: :op2: :val:) AND (:col2: :op1: :val2: AND :col2: :op1:
:val2:)';

echo p_str_replace(array(
  array(':col:', ':val:'),
  array(':col2:', ':val2:'),
  array(':op1:', ':op2:')
), array(
  array('date_start', '2015-10-01'),
  array('date_end', '2015-10-14'),
  array('>', '<')
), $complex);

?>

Sorry about my bad English :)
Hope this help someone.
```

up
down
-3
Anonymous ¶
2 years ago

```
<?php

namespace Foobar;

class Foo {

    public $email;
    public $name;
    public $age;

    public function setEmail($email) {
        $this->email = $email;
    }
    public function setName($name) {
        $this->name = $name;
    }
    public function setAge($age) {
        $this->age = $age;
    }
}

function getFields() {
    return [
        "age" => 112,
        "email" => "lol@lol.fr",
        "name" => "lol"
    ];
}

$foo = new Foo();

$callables = ["email" => $email,"name" => $name,"age" => $age] = getFields();

foreach ($callables as $func => $value) {
    call_user_func(array($foo,sprintf("set%s",ucfirst($func))),$value));
}

var_dump($foo);
```

+ add a note

- Funciones de Manejo de Funciones

  - call_user_func_array
  - call_user_func
  - forward_static_call_array
  - forward_static_call
  - func_get_arg
  - func_get_args
  - func_num_args
  - function_exists
  - get_defined_functions
  - register_shutdown_function
  - register_tick_function
  - unregister_tick_function
- Deprecated
  - create_function