Search

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

Веб-сервисы »
« var_dump

- Руководство по PHP
- Справочник функций
- Модули, относящиеся к переменным и типам
- Обработка переменных
- Функции для работы с переменными

var_export

(PHP 4 >= 4.2.0, PHP 5, PHP 7, PHP 8)

var_export — Выводит или возвращает интерпретируемое строковое представление переменной

Описание

var_export(mixed `$value`, bool `$return` = **`false`**): ?string

var_export() возвращает структурированную информацию о данной переменной. Функция аналогична var_dump() за одним исключением: возвращаемое представление является полноценным PHP-кодом.

Список параметров

`value`

Переменная, которую необходимо экспортировать.

`return`

Если передано и значение равно **`true`**, **var_export()** вернёт представление переменной вместо его вывода.

Возвращаемые значения

Возвращает представление переменной, если параметр `return` передан и равен **`true`**. В противном случае функция возвращает **`null`**.

Список изменений

Версия	Описание
8.2.0	Имена экспортируемых классов теперь полностью определены; ранее ведущий обратный слеш опускался.
7.3.0	Теперь объекты stdClass экспортируются в виде массива, приведённого к объекту (массив `(object) array( ... )`), вместо использования несуществующего метода **stdClass::__setState()**. Практический эффект заключается в том, что теперь stdClass можно экспортировать, и полученный код будет работать даже в более ранних версиях PHP.

Примеры

Пример #1 Примеры использования var_export()

```
<?php
$a = array (1, 2, array ("a", "b", "c"));
var_export($a);
?>
```

Результат выполнения приведённого примера:

```
array (
  0 => 1,
  1 => 2,
  2 =>
  array (
    0 => 'a',
    1 => 'b',
    2 => 'c',
  ),
)
```

```
<?php

$b = 3.1;
```

```php
$v = var_export($b, true);
echo $v;

?>
```

Результат выполнения приведённого примера:

```
3.1
```

Пример #2 Экспорт stdClass с PHP 7.3.0

```php
<?php
$person = new stdClass;
$person->name = 'ElePHPant ElePHPantsdotter';
$person->website = 'https://php.net/elephpant.php';

var_export($person);
```

Результат выполнения приведённого примера:

```
(object) array(
   'name' => 'ElePHPant ElePHPantsdotter',
   'website' => 'https://php.net/elephpant.php',
)
```

Пример #3 Экспорт классов

```php
<?php
class A { public $var; }
$a = new A;
$a->var = 5;
var_export($a);
?>
```

Результат выполнения приведённого примера:

```
A::__set_state(array(
   'var' => 5,
))
```

Пример #4 Использование __set_state()

```php
<?php
class A
{
public $var1;
public $var2;

public static function __set_state($an_array)
{
$obj = new A;
$obj->var1 = $an_array['var1'];
$obj->var2 = $an_array['var2'];
return $obj;
}
}

$a = new A;
$a->var1 = 5;
$a->var2 = 'foo';

eval('$b = ' . var_export($a, true) . ';'); // $b = A::__set_state(array(
// 'var1' => 5,
// 'var2' => 'foo',
// ));
var_dump($b);
?>
```

Результат выполнения приведённого примера:

```
object(A)#2 (2) {
  ["var1"]=>
  int(5)
  ["var2"]=>
  string(3) "foo"
}
```

Примечания

> **Замечание**:
>
> Переменные типа resource не могут быть экспортированы с помощью этой функции.
>
> **Замечание**:
>
> **var_export()** не обрабатывает циклические ссылки, так как было бы почти невозможно сгенерировать интерпретируемый PHP-код для такого случая. Если необходимо производить какие-то действия с полным представлением массива или объекта, используйте функцию serialize().

Внимание

До версии PHP 8.2.0, когда функция **var_export()** экспортировала объекты, ведущий обратный слеш не добавлялся в имя класса с указанным пространством имён для наилучшей обратной совместимости.

> **Замечание**:
>
> Для того, чтобы можно было использовать сгенерированный **var_export()** PHP-код, необходимо, чтобы все затронутые объекты реализовывали магический метод __set_state. Единственное исключение — stdClass, который экспортируется с использованием массива, приведённого к объекту.

Смотрите также

- print_r() - Выводит удобочитаемую информацию о переменной
- serialize() - Генерирует пригодное для хранения представление переменной
- var_dump() - Выводит информацию о переменной

User Contributed Notes 27 notes

26
steven at nevvix dot com ¶
4 years ago

I improved my previous varexport().

```
<?php
/**
* PHP var_export() with short array syntax (square brackets) indented 2 spaces.
*
* NOTE: The only issue is when a string value has `=>\n[`, it will get converted to `=> [`
* @link https://www.php.net/manual/en/function.var-export.php
*/
function varexport($expression, $return=FALSE) {
$export = var_export($expression, TRUE);
$patterns = [
"/array \(/" => '[',
"/^([ ]*)\)(,?)$/m" => '$1]$2',
"/=>[ ]?\n[ ]+\[/" => '=> [',
"/([ ]*)(\'[^\']+\') => ([\[\'])/" => '$1$2 => $3',
];
$export = preg_replace(array_keys($patterns), array_values($patterns), $export);
if ((bool)$return) return $export; else echo $export;
```

```
}

$array = [
'str' => 'Test
spaces',
0 => 33,
1 => TRUE,
[3,4,'d',[]],
'arr' => [
'text with spaces' => '[Tes\'t"s":
=> [
=>
[
{
spaces',
],
"str2" => "Test's'
} spaces",
'arr2' => [
'text with spaces' => [
'arr3' => [
'text with spaces' => 'Te": "st \' => [
spaces',
],
],
],
];
varexport($array);
// Result:
```
```
[
'str' => 'Test
spaces',
0 => 33,
1 => true,
2 => [
0 => 3,
1 => 4,
2 => 'd',
3 => [
],
],
'arr' => [
'text with spaces' => '[Tes\'t"s":
=> [
=> [
{
spaces',
],
'str2' => 'Test\'s\'
} spaces',
'arr2' => [
'text with spaces' => [
'arr3' => [
'text with spaces' => 'Te": "st \' => [
spaces',
],
],
],
]
```

NOTE: The only issue is when a string value has `=>\n[`, it will get converted to `=> [`

up
down
25

steven at nevvix dot com ¶

5 years ago

```
/**
* var_export() with square brackets and indented 4 spaces.
*/
<?php
function varexport($expression, $return=FALSE) {
$export = var_export($expression, TRUE);
$export = preg_replace("/^([ ]*)(.*)/m", '$1$1$2', $export);
$array = preg_split("/\r\n|\n|\r/", $export);
$array = preg_replace(["/\s*array\s\($/", "/\)(,)?$/", "/\s=>\s$/"], [NULL, ']$1', ' => ['], $array);
$export = join(PHP_EOL, array_filter(["["] + $array));
if ((bool)$return) return $export; else echo $export;
}
```

up
down
15

Mark P ¶

8 years ago

It doesn't appear to be documented, but the behaviour of `var_export()` changed in PHP 7.

Previously, `var_export(3.)` returned "3", now it returns "3.0".

up
down
21

chudinov at yahoo dot com ¶

10 years ago

Looks like since version 5.4.22 var_export uses the serialize_precision ini setting, rather than the precision one used for normal output of floating-point numbers.
As a consequence since version 5.4.22 for example var_export(1.1) will output 1.1000000000000001 (17 is default precision value) and not 1.1 as before.

```
<?php
//ouput 1.1000000000000001
var_export(1.1)
?>
```

up
down
12

laszlo dot heredy at gmail dot com ¶

13 years ago

Try this function instead of var_export($GLOBALS) or var_dump($GLOBALS) when all you want to know is the values of the variables you set on the current page.

```
<?php
function globalvars(){
$result=array();
$skip=array('GLOBALS','_ENV','HTTP_ENV_VARS',
'_POST','HTTP_POST_VARS','_GET',
'HTTP_GET_VARS',
'_COOKIE',
'HTTP_COOKIE_VARS','_SERVER',
'HTTP_SERVER_VARS',
'_FILES','HTTP_POST_FILES',
'_REQUEST','HTTP_SESSION_VARS',
'_SESSION');
foreach($GLOBALS as $k=>$v)
if(!in_array($k,$skip))
```

```
$result[$k]=$v;
return $result;
}//functionglobalvars

var_export(globalvars());
?>
```

up
down
13

dan at coders dot co dot nz ¶

10 years ago

I found that my complex type was exporting with
stdClass::__set_state()
in places. Not only was that strange and messy, it cannot be eval()-ed back in at all. Fatal error. Doh!

However a quick string-replace tidy-up of the result rendered it valid again.

```
$macro = var_export($data, TRUE);
$macro = str_replace("stdClass::__set_state", "(object)", $macro);
$macro = '$data = ' . $macro . ';';
```

And now the string I output *can* be evaluated back in again.

up
down
8

4n4jmza02 at sneakemail dot com ¶

13 years ago

I learned the hard way that if var_export encounters a resource handle it exports it as "NULL", even if it is a valid handle. The documentation states that a handle cannot be exported, but it does not describe what happens if you try to do so anyway.

I had been using var_export in some debugging code while tracing a problem with a resource handle not being generated and ended up thinking that null handles were still being generated long after the problem had been fixed.

up
down
11

linus at flowingcreativity dot net ¶

18 years ago

<roman at DIESPAM dot feather dot org dot ru>, your function has inefficiencies and problems. I probably speak for everyone when I ask you to test code before you add to the manual.

Since the issue of whitespace only comes up when exporting arrays, you can use the original var_export() for all other variable types. This function does the job, and, from the outside, works the same as var_export().

```
<?php

function var_export_min($var, $return = false) {
if (is_array($var)) {
$toImplode = array();
foreach ($var as $key => $value) {
$toImplode[] = var_export($key, true).'=>'.var_export_min($value, true);
}
$code = 'array('.implode(',', $toImplode).')';
if ($return) return $code;
else echo $code;
} else {
return var_export($var, $return);
}
}

?>
```

up

down
6
john dot risken at gmail dot com ¶
14 years ago
I didn't see this simple little item anywhere in the user notes. Maybe I'm blind!

Anyway, var_export and print_r both use spaces and carriage returns for formatting. Sent to an html page, most of the formatting is lost. This simple function prints a nicely formatted array to an html screen:

```
<?php
function pretty_var($myArray){
print str_replace(array("\n"," "),array("<br>"," "), var_export($myArray,true))."<br>";
}
?>
```

up
down
5
NitPicker ¶
10 years ago
When it comes to HTML output (as discussed below), it's all fun and games until someone pokes their eye out with a "<".

Surround it with "<pre>", but do remember to wrap it in htmlspecialchars() as well.
up
down
7
Glen ¶
16 years ago
Like previously reported, i find var_export() frustrating when dealing with recursive structures. Doing a :

```
<?php
var_export($GLOBALS);
?>
```

fails. Interestingly, var_dump() has some logic to avoid recursive references. So :

```
<?php
var_dump($GLOBALS);
?>
```

works (while being more ugly). Unlike var_export(), var_dump() has no option to return the string, so output buffering logic is required if you want to direct the output.
up
down
5
ravenswd at gmail dot com ¶
14 years ago
(This replaces my note of 3-July-2009. The original version produced no output if a variable contained an empty array, or an array consisting only of empty arrays. For example, $bigarray['x'] = array(); Also, I have added a second version of the function.)

The output can be difficult to decipher when looking at an array with many levels and many elements on each level. For example:

```
<?php
print ('$bigarray = ' . var_export($bigarray, true) . "\n");
?>
```

will return:

```
$bigarray = array(
... (500 lines skipped) ...
'mod' => 'charlie',
```

Whereas the routine below can be called with:

```
<?php
recursive_print ('$bigarray', $bigarray);
?>
```

and it will return:

```
$bigarray = array()
... (500 lines skipped) ...
$bigarray['foo']['bar']['0']['somethingelse']['mod'] = 'charlie'
```

Here's the function:

```
<?php
function recursive_print ($varname, $varval) {
if (! is_array($varval)):
print $varname . ' = ' . $varval . "<br>\n";
else:
print $varname . " = array()<br>\n";
foreach ($varval as $key => $val):
recursive_print ($varname . "['" . $key . "']", $val);
endforeach;
endif;
}
?>
```

For those who want a version that produces valid PHP code, use this version:

```
<?php
function recursive_print ($varname, $varval) {
if (! is_array($varval)):
print $varname . ' = ' . var_export($varval, true) . ";<br>\n";
else:
print $varname . " = array();<br>\n";
foreach ($varval as $key => $val):
recursive_print ($varname . "[" . var_export($key, true) . "]", $val);
endforeach;
endif;
}
?>
```

If your output is to a text file and not an HTML page, remove the
s.

up
down
5

jodybrabec at gmail dot com ¶

11 years ago

```
WORKAROUND for error "Nesting level too deep - recursive dependency":
ob_start();
var_dump($GLOBALS);
$dataDump = ob_get_clean();
echo $dataDump;
```

up
down
2

beverasrilakshmi at gmail dot com ¶

4 years ago

Just for fun, trying to understand the definition of "returns parsable string"....any type of variable passed to var_export, the return value will be a typecasted as string...

```
<?php

$var = 1;
var_dump($var); //type is int as expected
echo "<br>";
$var_after_export = var_export($var,true); //returning $var will now makes it a string
var_dump($var_after_export);

?>
```

up
down
4

sergei dot solomonov at gmail dot com ¶

11 years ago

```
<?php
$closure = function(){};

var_export($closure);

// output: Closure::__set_state(array())
?>
```

up
down
4

Anonymous ¶

12 years ago

There is an even simpler way to have clean output from var_export and print_r in html pages:

```
<?php
function pretty_var($myArray)
{
echo "<pre>";
var_export($myArray);
echo "</pre>";
}
?>
```

up
down
2

stangelanda at arrowquick dot com ¶

16 years ago

I have been looking for the best method to store data in cache files.

First, I've identified two limitations of var_export verus serialize. It can't store internal references inside of an array and it can't store a nested object or an array containing objects before PHP 5.1.0.

However, I could deal with both of those so I created a benchmark. I used a single array containing from 10 to 150 indexes. I've generate the elements' values randomly using booleans, nulls, integers, floats, and some nested arrays (the nested arrays are smaller averaging 5 elements but created similarly). The largest percentage of elements are short strings around 10-15 characters. While there is a small number of long strings (around 500 characters).

Benchmarking returned these results for 1000 * [total time] / [iterations (4000 in this case)]

serialize 3.656, 3.575, 3.68, 3.933, mean of 3.71
include 7.099, 5.42, 5.185, 6.076, mean of 5.95
eval 5.514, 5.204, 5.011, 5.788, mean of 5.38

Meaning serialize is around 1 and a half times faster than var_export for a single large array. include and eval were consistently very close but eval was usually a few tenths faster (eval did better this particular set of trials than usual). An opcode cache like APC might make include faster, but otherwise serialize is the best choice.

up
down

2

paul at worldwithoutwalls dot co dot uk ¶

19 years ago

var_export() differs from print_r() for variables that are resources, with print_r() being more useful if you are using the function for debugging purposes.

e.g.

```
<?php
$res = mysql_connect($dbhost, $dbuser, $dbpass);
print_r($res); //output: Resource id #14
var_export($res); //output: NULL
?>
```

up

down

1

php_manual_note at bigredspark dot com ¶

20 years ago

[john holmes]

True, but that method would require you to open and read the file into a variable and then unserialize it into another variable.

Using a file created with var_export() could simply be include()'d, which will be less code and faster.

[kaja]

If you are trying to find a way to temporarily save variables into some other file, check out serialize() and unserialize() instead - this one is more useful for its readable property, very handy while debugging.

[original post]

If you're like me, you're wondering why a function that outputs "correct PHP syntax" is useful. This function can be useful in implementing a cache system. You can var_export() the array into a variable and write it into a file. Writing a string such as

```
<?php
$string = '<?php $array = ' . $data . '; ?>';
?>
```

where $data is the output of var_export() can create a file that can be easily include()d back into the script to recreate $array.

The raw output of var_export() could also be eval()d to recreate the array.

---John Holmes...

up

down

0

rarioj at gmail dot com ¶

14 years ago

NOTE: If an object Foo has __set_state() method, but if that object contains another object Bar with no __set_state() method implemented, the resulting PHP expression will not be eval()-able.

This is an example (object Test that contains an instance of Exception).

```
<?php

class Test
{
public $one;
public $two;
public function __construct($one, $two)
{
$this->one = $one;
$this->two = $two;
}
```

```
public static function __set_state(array $array)
{
return new self($array['one'], $array['two']);
}
}

$test = new Test('one', new Exception('test'));

$string = var_export($test, true);

/* $string =
Test::__set_state(array(
'one' => 'one',
'two' =>
Exception::__set_state(array(
'message' => 'test',
'string' => '',
'code' => 0,
'file' => 'E:\\xampp\\htdocs\\test.Q.php',
'line' => 35,
'trace' =>
array (
),
'previous' => NULL,
)),
))
*/

eval('$test2 = '.$string.';'); // Fatal error: Call to undefined method Exception::__set_state

?>
```

So avoid using var_export() on a complex array/object that contains other objects. Instead, use serialize() and unserialize() functions.

```
<?php

$string = 'unserialize('.var_export(serialize($test), true).')';

eval('$test2 = '.$string.';');

var_dump($test == $test2); // bool(true)

?>
```

up
down
-2
php dot net at rupert-eibauer dot de ¶
1 year ago

Warning: var_export fails to create distinct property names if you have a private property in your class and one with the same name in a parent class.

```
<?php
class TestParent {
private $priv = "Parent Private";
protected $prot = "Parent Protected";
public $pub = "Parent Public";
}
class Test extends TestParent {
private $priv = "Private Info";
protected $prot = "Protected Info";
public $pub = "Public Info";
```

```
}
$t = new Test;
var_export($t);
Result:
Test::__set_state(array(
'priv' => 'Private Info',
'prot' => 'Protected Info',
'pub' => 'Public Info',
'priv' => 'Parent Private',
))
?>
```

To work around that problem, I created my own var_export, which also uses a more compact format, and using [] for arrays:

```
<?php
function var_export3($var, $indent = '') {
if (is_array($var)) {
$indent .= ' ';
$ret = '';
$i = 0;
$is_num = true;
foreach ($var AS $idx => $value) {
if ($ret != '')
$ret .= ",\n";
if ($is_num && $idx === $i) {
$i++;
$ret .= $indent.var_export3($value, $indent);
} else {
$is_num = false;
$ret .= $indent.var_export3($idx).'=>'.var_export3($value, $indent);
}
}
if ($ret == '')
return "[]";
return "[\n".$ret."]";
} else if (is_bool($var)) {
return $var ? 'true' : 'false';
} else if (is_int($var) || is_float($var)) {
return $var;
} else if (is_string($var)) {
if ($var === (string)(float)$var)
return $var;
return "'".str_replace(["\0", '\\', '\''], ["\\0", '\\', '\\\''], $var)."'";
} else if (is_object($var)) {
$class = get_class($var);
if ($class == 'stdClass') {
$prefix = ;
return "(object)".var_export3((array)$var, $indent);
} else {
return "$class::__set_state(".var_export3((array)$var, $indent).')';
}
}
}

echo var_export3($t)."\n";
Result:
Test::__set_state([
'\0Test\0priv'=>'Private Info',
'\0*\0prot'=>'Protected Info',
'pub'=>'Public Info',
'\0TestParent\0priv'=>'Parent Private'])
?>
```

Note that replacement for non-printable characters is not complete.

up

down

-1

wyattstorch42 at outlook dot com ¶

10 years ago

If you call var_export() on an instance of stdClass, it attempts to export it using ::__set_state(), which, for some reason, is not implemented in stdClass.

However, casting an associative array to an object usually produces the same effect (at least, it does in my case). So I wrote an improved_var_export() function to convert instances of stdClass to (object) array () calls. If you choose to export objects of any other class, I'd advise you to implement ::__set_state().

```
<?php
/**
* An implementation of var_export() that is compatible with instances
* of stdClass.
* @param mixed $variable The variable you want to export
* @param bool $return If used and set to true, improved_var_export()
* will return the variable representation instead of outputting it.
* @return mixed|null Returns the variable representation when the
* return parameter is used and evaluates to TRUE. Otherwise, this
* function will return NULL.
*/
function improved_var_export ($variable, $return = false) {
if ($variable instanceof stdClass) {
$result = '(object) '.improved_var_export(get_object_vars($variable), true);
} else if (is_array($variable)) {
$array = array ();
foreach ($variable as $key => $value) {
$array[] = var_export($key, true).' => '.improved_var_export($value, true);
}
$result = 'array ('.implode(', ', $array).')';
} else {
$result = var_export($variable, true);
}

if (!$return) {
print $result;
return null;
} else {
return $result;
}
}

// Example usage:
$obj = new stdClass;
$obj->test = 'abc';
$obj->other = 6.2;
$obj->arr = array (1, 2, 3);

improved_var_export((object) array (
'prop1' => true,
'prop2' => $obj,
'assocArray' => array (
'apple' => 'good',
'orange' => 'great'
)
));

/* Output:
(object) array ('prop1' => true, 'prop2' => (object) array ('test' => 'abc', 'other' => 6.2, 'arr' => array (0 => 1, 1 =>
```

```
2, 2 => 3)), 'assocArray' => array ('apple' => 'good', 'orange' => 'great'))
*/
?>
```

Note: This function spits out a single line of code, which is useful to save in a cache file to include/eval. It isn't formatted for readability. If you want to print a readable version for debugging purposes, then I would suggest print_r() or var_dump().

up
down
-1

ravenswd at gmail dot com ¶

14 years ago

The output can be difficult to decipher when looking at an array with many levels and many elements on each level. For example:

```
<?php
print ('$bigarray = ' . var_export($bigarray, true) . "\n");
?>
```

will return:

```
$bigarray = array(
... (500 lines skipped) ...
'mod' => 'charlie',
```

Whereas the routine below can be called with:

```
<?php
recursive_print ('$bigarray', $bigarray);
?>
```

and it will return:

```
$bigarray['firstelement'] = 'something'
... (500 lines skipped) ...
$bigarray['foo']['bar']['0']['somethingelse']['mod'] = 'charlie'
```

Here's the function:

```
<?php
function recursive_print ($varname, $varval) {
if (! is_array($varval)):
print $varname . ' = ' . $varval . "<br>\n";
else:
foreach ($varval as $key => $val):
recursive_print ($varname . "['" . $key . "']", $val);
endforeach;
endif;
}
?>
```

up
down
-2

cpmcgrat()uci ! edu ¶

8 years ago

When trying to use __set_state() to rebuild a huge, tricky class use the following:

```
class Foo
{
public $a;
public $b;
public $c;
```

```
public $d;
public $e;
public $f;
public $g;
public $h;
public $i;

public function __set_state($array)
{
$obj = new ArrayConfig;
foreach($array as $k => $v) {
eval('$obj->'.$k.' = '.$v.';');
}
return $obj;
}
}
```

This will return a reconstructed version of the class without having to manually type each individual object in the class manually (as shown in the __set_state() example)

up
down
-4

cmusicfan (at) gmail (daught) com ¶

14 years ago

Caution! var_export will add a backslash to single quotes (').

You may want to use stripslashes() to remove the mysteriously added backslashes.

up
down
-5

Zorro ¶

18 years ago

This function can't export EVERYTHING. Moreover, you can have an error on an simple recursive array:

```
$test = array();
$test["oops"] = & $test;

echo var_export($test);

=>

Fatal error: Nesting level too deep - recursive dependency? in ??.php on line 59
```

up
down
-5

kexianbin at diyism dot com ¶

11 years ago

to use my_var_export(), it is as beautiful as var_export() and as could deal with recursive reference as print_r():

```
<?php
function my_var_export($var, $is_str=false)
{$rtn=preg_replace(array('/Array\s+\(/', '/\[(\d+)\] => (.*)\n/', '/\[([^\d].*)\] => (.*)\n/'), array('array (', '\1 => \'\2\''."\n", '\'\1\' => \'\2\''."\n"), substr(print_r($var, true), 0, -1));
$rtn=strtr($rtn, array("=> 'array ('"=>'=> array ('));
$rtn=strtr($rtn, array(")\n\n"=>")\n"));
$rtn=strtr($rtn, array("'\n"=>"',\n", ")\n"=>"),\n"));
$rtn=preg_replace(array('/\n +/e'), array('strtr(\'\0\', array(\' \'=>\' \'))'), $rtn);
$rtn=strtr($rtn, array(" Object',"=>" Object'<-"));
if ($is_str)
{return $rtn;
}
else
```

```
{echo $rtn;
}
}

?>
```

＋add a note