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

Переменные извне PHP »
« Область видимости переменной

- Руководство по PHP
- Справочник языка
- Переменные

Change language: Russian

Переменные переменных

Иногда бывает удобно иметь переменными имена переменных. То есть, имя переменной, которое может быть определено и изменено динамически. Обычная переменная определяется примерно таким выражением:

```
<?php
$a = 'hello';
?>
```

Переменная переменной берет значение переменной и рассматривает его как имя переменной. В вышеприведённом примере *hello* может быть использовано как имя переменной при помощи двух знаков доллара. То есть:

```
<?php
$$a = 'world';
?>
```

Теперь в дереве символов PHP определены и содержатся две переменные: *$a*, содержащая "hello" и *$hello*, содержащая "world". Таким образом, выражение

```
<?php
echo "$a {$$a}";
?>
```

выведет то же, что и

```
<?php
echo "$a $hello";
?>
```

то есть, они оба выведут: hello world.

Для того чтобы использовать переменные переменных с массивами, вы должны решить проблему двусмысленности. То есть, если вы напишете *$$a[1]*, обработчику необходимо знать, хотите ли вы использовать *$a[1]* в качестве переменной, либо вам нужна как переменная *$$a*, а затем её индекс [1]. Синтаксис для разрешения этой двусмысленности таков: *${$a[1]}* для первого случая и *${$a}[1]* для второго.

К свойствам класса также можно получить доступ динамически. Переменное имя свойства будет разрешено в том контексте, в котором произойдёт вызов к нему. Например, в случае выражения *$foo->$bar*, локальная область видимости будет просканирована на наличие переменной *$bar*, значение которой будет использовано в качестве имени свойства объекта *$foo*. Это также работает и в том случае, если *$bar* осуществляет доступ к элементу массива.

Фигурные скобки могут также использоваться, чтобы чётко разграничить имя свойства. Они наиболее полезны при получении доступа к значениям внутри свойства, которое содержит массив, когда имя свойства состоит из нескольких частей, либо когда имя свойства содержит символы, которые иначе не действительны (например, из функции json_decode() или из SimpleXML).

Пример #1 Пример переменного имени свойства

```
<?php
class foo {
var $bar = 'I am bar.';
var $arr = array('I am A.', 'I am B.', 'I am C.');
var $r = 'I am r.';
}

$foo = new foo();
$bar = 'bar';
$baz = array('foo', 'bar', 'baz', 'quux');
echo $foo->$bar . "\n";
echo $foo->{$baz[1]} . "\n";

$start = 'b';
$end = 'ar';
```

```php
echo $foo->{$start . $end} . "\n";

$arr = 'arr';
echo $foo->{$arr[1]} . "\n";

?>
```

Результат выполнения приведённого примера:

```
I am bar.
I am bar.
I am bar.
I am r.
```

Внимание

Обратите внимание, что переменные переменных не могут использоваться с суперглобальными массивами PHP. Переменная `$this` также является особой, на неё нельзя ссылаться динамически.

＋add a note

User Contributed Notes 10 notes

up
down
548
userb at exampleb dot org ¶
13 years ago

```php
<?php

//You can even add more Dollar Signs

$Bar = "a";
$Foo = "Bar";
$World = "Foo";
$Hello = "World";
$a = "Hello";

$a; //Returns Hello
$$a; //Returns World
$$$a; //Returns Foo
$$$$a; //Returns Bar
$$$$$a; //Returns a

$$$$$$a; //Returns Hello
$$$$$$$a; //Returns World

//... and so on ...//

?>
```

up
down
11
marcin dot dzdza at gmail dot com ¶
5 years ago

The feature of variable variable names is welcome, but it should be avoided when possible. Modern IDE software fails to interpret such variables correctly, regular find/replace also fails. It's a kind of magic :) This may really make it hard to refactor code. Imagine you want to rename variable $username to $userName and try to find all occurrences of $username in code by checking "$userName". You may easily omit:

```php
$a = 'username';
echo $$a;
```

up

down
6
sebastopolys at gmail dot com ¶
1 year ago
In addition, it is possible to use associative array to secure name of variables available to be used within a function (or class / not tested).

This way the variable variable feature is useful to validate variables; define, output and manage only within the function that receives as parameter
an associative array :
array('index'=>'value','index'=>'value');
index = reference to variable to be used within function
value = name of the variable to be used within function

```
<?php

$vars = ['id'=>'user_id','email'=>'user_email'];

validateVarsFunction($vars);

function validateVarsFunction($vars){

//$vars['id']=34; <- does not work
// define allowed variables
$user_id=21;
$user_email='email@mail.com';

echo $vars['id']; // prints name of variable: user_id
echo ${$vars['id']}; // prints 21
echo 'Email: '.${$vars['email']}; // print email@mail.com

// we don't have the name of the variables before declaring them inside the function
}
?>
```

up
down
70
Anonymous ¶
18 years ago
It may be worth specifically noting, if variable names follow some kind of "template," they can be referenced like this:

```
<?php
// Given these variables ...
$nameTypes = array("first", "last", "company");
$name_first = "John";
$name_last = "Doe";
$name_company = "PHP.net";

// Then this loop is ...
foreach($nameTypes as $type)
print ${"name_$type"} . "\n";

// ... equivalent to this print statement.
print "$name_first\n$name_last\n$name_company\n";
?>
```

This is apparent from the notes others have left, but is not explicitly stated.

up
down
10
jefrey.sobreira [at] gmail [dot] com ¶
9 years ago
If you want to use a variable value in part of the name of a variable variable (not the whole name itself), you can do

like the following:

```
<?php
$price_for_monday = 10;
$price_for_tuesday = 20;
$price_for_wednesday = 30;

$today = 'tuesday';

$price_for_today = ${ 'price_for_' . $today};
echo $price_for_today; // will return 20
?>
```

up
down
8

Sinured ¶

16 years ago

One interesting thing I found out: You can concatenate variables and use spaces. Concatenating constants and function calls are also possible.

```
<?php
define('ONE', 1);
function one() {
return 1;
}
$one = 1;

${"foo$one"} = 'foo';
echo $foo1; // foo
${'foo' . ONE} = 'bar';
echo $foo1; // bar
${'foo' . one()} = 'baz';
echo $foo1; // baz
?>
```

This syntax doesn't work for functions:

```
<?php
$foo = 'info';
{"php$foo"}(); // Parse error

// You'll have to do:
$func = "php$foo";
$func();
?>
```

Note: Don't leave out the quotes on strings inside the curly braces, PHP won't handle that graciously.

up
down
17

mason ¶

13 years ago

PHP actually supports invoking a new instance of a class using a variable class name since at least version 5.2

```
<?php
class Foo {
public function hello() {
echo 'Hello world!';
}
}
$my_foo = 'Foo';
$a = new $my_foo();
```

```
$a->hello(); //prints 'Hello world!'
?>

Additionally, you can access static methods and properties using variable class names, but only since PHP 5.3

<?php
class Foo {
public static function hello() {
echo 'Hello world!';
}
}
$my_foo = 'Foo';
$my_foo::hello(); //prints 'Hello world!'
?>
```

up
down
8

herebepost (ta at ta) [iwonderr] gmail dot com ¶

7 years ago

While not relevant in everyday PHP programming, it seems to be possible to insert whitespace and comments between the dollar signs of a variable variable. All three comment styles work. This information becomes relevant when writing a parser, tokenizer or something else that operates on PHP syntax.

```
<?php

$foo = 'bar';
$

/*
I am complete legal and will compile without notices or error as a variable variable.
*/
$foo = 'magic';

echo $bar; // Outputs magic.

?>

Behaviour tested with PHP Version 5.6.19
```

up
down
8

Nathan Hammond ¶

16 years ago

These are the scenarios that you may run into trying to reference superglobals dynamically. Whether or not it works appears to be dependent upon the current scope.

```
<?php

$_POST['asdf'] = 'something';

function test() {
// NULL -- not what initially expected
$string = '_POST';
var_dump(${$string});

// Works as expected
var_dump(${'_POST'});

// Works as expected
global ${$string};
var_dump(${$string});
```

```
}

// Works as expected
$string = '_POST';
var_dump(${$string});

test();

?>
```

up
down
4
nils dot rocine at gmail dot com ¶
11 years ago

Variable Class Instantiation with Namespace Gotcha:

Say you have a class you'd like to instantiate via a variable (with a string value of the Class name)

```
<?php

class Foo
{
public function __construct()
{
echo "I'm a real class!" . PHP_EOL;
}
}

$class = 'Foo';

$instance = new $class;

?>
```

The above works fine UNLESS you are in a (defined) namespace. Then you must provide the full namespaced identifier of the class as shown below. This is the case EVEN THOUGH the instancing happens in the same namespace. Instancing a class normally (not through a variable) does not require the namespace. This seems to establish the pattern that if you are using an namespace and you have a class name in a string, you must provide the namespace with the class for the PHP engine to correctly resolve (other cases: class_exists(), interface_exists(), etc.)

```
<?php

namespace MyNamespace;

class Foo
{
public function __construct()
{
echo "I'm a real class!" . PHP_EOL;
}
}

$class = 'MyNamespace\Foo';

$instance = new $class;

?>
```

＋add a note