Garbage Collection
DTrace Dynamic Tracing

Function Reference
- Affecting PHP's Behaviour
- Audio Formats Manipulation
- Authentication Services
- Command Line Specific Extensions
- Compression and Archive Extensions
- Cryptography Extensions
- Database Extensions
- Date and Time Related Extensions
- File System Related Extensions
- Human Language and Character Encoding Support
- Image Processing and Generation
- Mail Related Extensions
- Mathematical Extensions
- Non-Text MIME Output
- Process Control Extensions
- Other Basic Extensions
- Other Services
- Search Engine Extensions
- Server Specific Extensions
- Session Extensions
- Text Processing
- Variable and Type Related Extensions
- Web Services
- Windows Only Extensions
- XML Manipulation
- GUI Extensions

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

Автоматическая загрузка классов »
« Свойства

Submit a Pull Request Report a Bug

Константы классов

Константы также могут быть объявлены в пределах одного класса. Область видимости констант по умолчанию `public`.

> **Замечание**:
>
> Константы класса могут быть переопределены дочерним классом. Начиная с PHP 8.1.0, константы класса не могут быть переопределены дочерним классом, если он определён как окончательный (final).

Интерфейсы также могут содержать константы. За примерами обращайтесь к разделу об интерфейсах.

К классу можно обратиться с помощью переменной. Значение переменной не может быть ключевым словом (например, `self`, `parent` и `static`).

Обратите внимание, что константы класса задаются один раз для всего класса, а не отдельно для каждого созданного объекта этого класса.

Пример #1 Объявление и использование константы

```
<?php
class MyClass
{
const CONSTANT = 'значение константы';

function showConstant() {
echo self::CONSTANT . "\n";
}
}

echo MyClass::CONSTANT . "\n";

$classname = "MyClass";
echo $classname::CONSTANT . "\n";

$class = new MyClass();
$class->showConstant();

echo $class::CONSTANT."\n";
?>
```

Специальная константа **`::class`**, которой на этапе компиляции присваивается полное имя класса, полезна при использовании с классами, использующими пространства имён.

Пример #2 Пример использования ::class с пространством имён

```
<?php
namespace foo {
class bar {
}

echo bar::class; // foo\bar
}
?>
```

Пример #3 Пример констант, заданных выражением

```
<?php
const ONE = 1;

class foo {
const TWO = ONE * 2;
const THREE = ONE + self::TWO;
const SENTENCE = 'Значение константы THREE - ' . self::THREE;
}
?>
```

Пример #4 Модификаторы видимости констант класса, начиная с PHP 7.1.0

```
<?php
class Foo {
public const BAR = 'bar';
private const BAZ = 'baz';
}
echo Foo::BAR, PHP_EOL;
echo Foo::BAZ, PHP_EOL;
?>
```

Результат выполнения приведённого примера в PHP 7.1:

```
bar

Fatal error: Uncaught Error: Cannot access private const Foo::BAZ in …
```

> **Замечание**:
>
> Начиная с PHP 7.1.0 для констант класса можно использовать модификаторы области видимости.

+ add a note

User Contributed Notes 12 notes

up
down
206
tmp dot 4 dot longoria at gmail dot com ¶
12 years ago

```
it's possible to declare constant in base class, and override it in child, and access to correct value of the const from
the static method is possible by 'get_called_class' method:
<?php
abstract class dbObject
{
const TABLE_NAME='undefined';

public static function GetAll()
{
$c = get_called_class();
return "SELECT * FROM `".$c::TABLE_NAME."`";
}
}

class dbPerson extends dbObject
{
const TABLE_NAME='persons';
}

class dbAdmin extends dbPerson
{
const TABLE_NAME='admins';
}

echo dbPerson::GetAll()."<br>";//output: "SELECT * FROM `persons`"
echo dbAdmin::GetAll()."<br>";//output: "SELECT * FROM `admins`"

?>
```

up
down
142

kuzawinski dot marcin at gmail dot com ¶

9 years ago

```
As of PHP 5.6 you can finally define constant using math expressions, like this one:

<?php

class MyTimer {
const SEC_PER_DAY = 60 * 60 * 24;
}

?>

Me happy :)
```

up

down

149

anonymous ¶

13 years ago

```
Most people miss the point in declaring constants and confuse then things by trying to declare things like functions or
arrays as constants. What happens next is to try things that are more complicated then necessary and sometimes lead to bad
coding practices. Let me explain...

A constant is a name for a value (but it's NOT a variable), that usually will be replaced in the code while it gets
COMPILED and NOT at runtime.

So returned values from functions can't be used, because they will return a value only at runtime.

Arrays can't be used, because they are data structures that exist at runtime.

One main purpose of declaring a constant is usually using a value in your code, that you can replace easily in one place
without looking for all the occurences. Another is, to avoid mistakes.

Think about some examples written by some before me:

1. const MY_ARR = "return array(\"A\", \"B\", \"C\", \"D\");";
It was said, this would declare an array that can be used with eval. WRONG! This is just a string as constant, NOT an
array. Does it make sense if it would be possible to declare an array as constant? Probably not. Instead declare the
values of the array as constants and make an array variable.

2. const magic_quotes = (bool)get_magic_quotes_gpc();
This can't work, of course. And it doesn't make sense either. The function already returns the value, there is no purpose
in declaring a constant for the same thing.

3. Someone spoke about "dynamic" assignments to constants. What? There are no dynamic assignments to constants, runtime
assignments work _only_ with variables. Let's take the proposed example:

<?php
/**
* Constants that deal only with the database
*/
class DbConstant extends aClassConstant {
protected $host = 'localhost';
protected $user = 'user';
protected $password = 'pass';
protected $database = 'db';
protected $time;
function __construct() {
$this->time = time() + 1; // dynamic assignment
}
}
?>
```

Those aren't constants, those are properties of the class. Something like "this->time = time()" would even totally defy the purpose of a constant. Constants are supposed to be just that, constant values, on every execution. They are not supposed to change every time a script runs or a class is instantiated.

Conclusion: Don't try to reinvent constants as variables. If constants don't work, just use variables. Then you don't need to reinvent methods to achieve things for what is already there.

up
down
115
delete dot this dot and dot dots dot gt at kani dot hu ¶
10 years ago

I think it's useful if we draw some attention to late static binding here:

```
<?php
class A {
const MY_CONST = false;
public function my_const_self() {
return self::MY_CONST;
}
public function my_const_static() {
return static::MY_CONST;
}
}

class B extends A {
const MY_CONST = true;
}

$b = new B();
echo $b->my_const_self ? 'yes' : 'no'; // output: no
echo $b->my_const_static ? 'yes' : 'no'; // output: yes
?>
```

up
down
93
Xiong Chiamiov ¶
9 years ago

const can also be used directly in namespaces, a feature never explicitly stated in the documentation.

```
<?php
# foo.php
namespace Foo;

const BAR = 1;
?>
```

```
<?php
# bar.php
require 'foo.php';

var_dump(Foo\BAR); // => int(1)
?>
```

up
down
47
jimmmy dot chief at gmail dot com ¶
7 years ago

Hi, i would like to point out difference between self::CONST and $this::CONST with extended class.
Let us have class a:

```
<?php
class a {
const CONST_INT = 10;
public function getSelf(){
return self::CONST_INT;
}

public function getThis(){
return $this::CONST_INT;
}
}
?>

And class b (which extends a)

<?php
class b extends a {
const CONST_INT = 20;

public function getSelf(){
return parent::getSelf();
}

public function getThis(){
return parent::getThis();
}
}
?>

Both classes have same named constant CONST_INT.
When child call method in parent class, there is different output between self and $this usage.

<?php
$b = new b();

print_r($b->getSelf()); //10
print_r($b->getThis()); //20

?>
```
up
down
29
nepomuk at nepda dot de ¶
7 years ago
```
[Editor's note: that is already possible as of PHP 5.6.0.]

Note, as of PHP7 it is possible to define class constants with an array.

<?php
class MyClass
{
const ABC = array('A', 'B', 'C');
const A = '1';
const B = '2';
const C = '3';
const NUMBERS = array(
self::A,
self::B,
self::C,
);
}
var_dump(MyClass::ABC);
var_dump(MyClass::NUMBERS);
```

```
// Result:
/*
array(3) {
[0]=>
string(1) "A"
[1]=>
string(1) "B"
[2]=>
string(1) "C"
}
array(3) {
[0]=>
string(1) "1"
[1]=>
string(1) "2"
[2]=>
string(1) "3"
}
*/
?>
```

up
down
1
powtac at gmx dot de ¶
9 months ago

Since it it is not mentioned in the description the following types can be set as a class constant: string, array, int, bool and probably float. But not objects.

```
<?php

class Test {
const arr = array();
const string = 'string';
const int = 99;
const bool = true;
}

var_dump(
(new Test())::arr,
(new Test())::string,
(new Test())::int,
(new Test())::bool
);

/* ouput for PHP 7.0.0+:

array(0) {
}
string(6) "string"
int(99)
bool(true)
```

up
down
9
wbcarts at juno dot com ¶
15 years ago

Use CONST to set UPPER and LOWER LIMITS

If you have code that accepts user input or you just need to make sure input is acceptable, you can use constants to set upper and lower limits. Note: a static function that enforces your limits is highly recommended... sniff the clamp() function below for a taste.

```
<?php

class Dimension
{
const MIN = 0, MAX = 800;

public $width, $height;

public function __construct($w = 0, $h = 0){
$this->width = self::clamp($w);
$this->height = self::clamp($h);
}

public function __toString(){
return "Dimension [width=$this->width, height=$this->height]";
}

protected static function clamp($value){
if($value < self::MIN) $value = self::MIN;
if($value > self::MAX) $value = self::MAX;
return $value;
}
}

echo (new Dimension()) . '<br>';
echo (new Dimension(1500, 97)) . '<br>';
echo (new Dimension(14, -20)) . '<br>';
echo (new Dimension(240, 80)) . '<br>';

?>
```

```
- - - - - - - -
Dimension [width=0, height=0] - default size
Dimension [width=800, height=97] - width has been clamped to MAX
Dimension [width=14, height=0] - height has been clamped to MIN
Dimension [width=240, height=80] - width and height unchanged
- - - - - - - -
```

Setting upper and lower limits on your classes also help your objects make sense. For example, it is not possible for the width or height of a Dimension to be negative. It is up to you to keep phoney input from corrupting your objects, and to avoid potential errors and exceptions in other parts of your code.

up
down
2
Nimja ¶
6 years ago

Note that this magic constant DOES NOT load classes. And in fact can work on classes that do not exist.

This means it does not mess with auto-loading.

```
<?php
$className = \Foo\Bar::class;
var_dump($className);
var_dump(class_exists($className, false));
?>
```

Will output:

```
string(7) "Foo\Bar"
bool(false)
```

up
down

3
***Paul* ¶**
8 years ago

Square or curly bracket syntax can normally be used to access a single byte (character) within a string. For example: $mystring[5]. However, please note that (for some reason) this syntax is not accepted for string class constants (at least, not in PHP 5.5.12).
For example, the following code gives "PHP Parse error: syntax error, unexpected '[' in php shell code on line 6".

```
<?php
class SomeClass
{
const SOME_STRING = '0123456790';
public static function ATest()
{
return self::SOME_STRING[0];
}
}
?>
```

It looks like you have to use a variable/class member instead.
up
down
1
***David Spector* ¶**
5 years ago

The usual comma-separated syntax can be used to declare several constants:

```
class STATE
{
const INIT=0, NAME_SEEN=1, ADDR_SEEN=2;
}
```

This shows the declaration of a set of enumeration literals suitable for use in a finite state machine loop. Reference such an enum by using syntax such as "STATE::INIT". Its actual type in this case will be integer.
+ add a note