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

Константы классов »
« Основы

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Свойства

Переменные, которые являются членами класса, называются *свойства*. Также их называют, используя другие термины, таких как *поля*, но в рамках этой документации, мы будем называть их *свойствами*. Они определяются с использованием хотя бы одного необязательного (за исключением `readonly`-свойств) модификатора (например, Область видимости, Ключевое слово static или, начиная с PHP 8.1.0, readonly), начиная с PHP 7.4, за которым следует необязательное объявление типа, за которым следует обычное объявление переменной. Это объявление может содержать инициализацию, но эта инициализация должна быть постоянным значением.

> **Замечание**:
>
> Устаревший способ объявления свойств класса — использование ключевого слова `var` вместо модификатора.
>
> **Замечание**: Свойство, объявленное без модификатора Область видимости, будет объявлено как `public`.

В пределах методов класса доступ к нестатическим свойствам может быть получен с помощью `->` (объектного оператора): *$this->property* (где `property` - имя свойства). Доступ к статическим свойствам осуществляется с помощью `::` (двойного двоеточия): *self::$property*. Дополнительную информацию о различии статических и нестатических свойств смотрите в разделе Ключевое слово static.

Псевдопеременная *$this* доступна внутри любого метода класса, когда этот метод вызывается из контекста объекта. *$this* - значение вызывающего объекта.

Пример #1 Определение свойств

```php
<?php
class SimpleClass
{
public $var1 = 'hello ' . 'world';
public $var2 = <<<EOD
hello world
EOD;
public $var3 = 1+2;
// неправильное определение свойств:
public $var4 = self::myStaticMethod();
public $var5 = $myVar;

// правильное определение свойств:
public $var6 = myConstant;
public $var7 = [true, false];

public $var8 = <<<'EOD'
hello world
EOD;

// Без модификатора области видимости:
static $var9;
readonly int $var10;

}
?>
```

> **Замечание**:
>
> Существуют различные функции для обработки классов и объектов. Смотрите справочник по функций для классов/объектов.

Объявления типов

Начиная с PHP 7.4.0, определения свойств могут включать Объявления типов, за исключением типа callable.

Пример #2 Пример использования типизированных свойств

```php
<?php

class User
{
public int $id;
public ?string $name;

public function __construct(int $id, ?string $name)
{
$this->id = $id;
$this->name = $name;
}
}

$user = new User(1234, null);

var_dump($user->id);
var_dump($user->name);

?>
```

Результат выполнения приведённого примера:

```
int(1234)
NULL
```

Перед обращением к типизированному свойству у него должно быть задано значение, иначе будет выброшено исключение Error.

Пример #3 Обращение к свойствам

```php
<?php

class Shape
{
public int $numberOfSides;
public string $name;

public function setNumberOfSides(int $numberOfSides): void
{
$this->numberOfSides = $numberOfSides;
}

public function setName(string $name): void
{
$this->name = $name;
}

public function getNumberOfSides(): int
{
return $this->numberOfSides;
}

public function getName(): string
{
return $this->name;
}
}

$triangle = new Shape();
$triangle->setName("triangle");
$triangle->setNumberofSides(3);
var_dump($triangle->getName());
```

```php
var_dump($triangle->getNumberOfSides());

$circle = new Shape();
$circle->setName("circle");
var_dump($circle->getName());
var_dump($circle->getNumberOfSides());
?>
```

Результат выполнения приведённого примера:

```
string(8) "triangle"
int(3)
string(6) "circle"

Fatal error: Uncaught Error: Typed property Shape::$numberOfSides must not be accessed before initialization
```

Readonly-свойства

Начиная с PHP 8.1.0, свойство можно объявить с помощью модификатора `readonly`, который предотвращает изменение свойства после инициализации.

Пример #4 Примеры readonly-свойств

```php
<?php
class Test {
public readonly string $prop;
public function __construct(string $prop) {
// Правильная инициализация.
$this->prop = $prop;
}
}
$test = new Test("foobar");
// Правильное чтение.
var_dump($test->prop); // string(6) "foobar"
// Неправильное переопределение. Не имеет значения, что присвоенное значение такое же.
$test->prop = "foobar";
// Ошибка: невозможно изменить readonly-свойство Test::$prop
?>
```

> **Замечание**:
>
> Модификатор readonly может применяться только к типизированным свойствам. Readonly-свойство без ограничений типа можно создать с помощью типа Mixed.

> **Замечание**:
>
> Статические readonly-свойства не поддерживаются.

Readonly-свойство можно инициализировать только один раз и только из области, в которой оно было объявлено. Любое другое присвоение или изменение свойства приведёт к исключению Error.

Пример #5 Неправильная инициализация readonly-свойств

```php
<?php
class Test1 {
public readonly string $prop;
}
$test1 = new Test1;
// Неправильная инициализация за пределами закрытой области.
$test1->prop = "foobar";
// Ошибка: не удаётся инициализировать readonly-свойство Test1::$prop из глобальной области
?>
```

> **Замечание**:
>
> Указание явного значения по умолчанию для readonly-свойств не допускается, потому что readonly-свойство со значением по умолчанию, по сути, то же самое, что и константа и поэтому не особенно полезно.

```php
<?php
class Test {
// Ошибка: у readonly-свойства Test::$prop не может быть значения по умолчанию
public readonly int $prop = 42;
}
?>
```

Замечание:

Readonly-свойства не могут быть уничтожены с помощью unset() после их инициализации. Однако можно уничтожить readonly-свойство до инициализации из области, в которой было объявлено свойство.

Модификации не обязательно являются простыми присвоениями, всё перечисленное ниже также приведёт к исключению Error:

```php
<?php
class Test {
public function __construct(
public readonly int $i = 0,
public readonly array $ary = [],
) {}
}
$test = new Test;
$test->i += 1;
$test->i++;
++$test->i;
$test->ary[] = 1;
$test->ary[0][] = 1;
$ref =& $test->i;
$test->i =& $ref;
byRef($test->i);
foreach ($test as &$prop);
?>
```

Однако readonly-свойства не исключают внутренней изменчивости. Объекты (или ресурсы), хранящиеся в readonly-свойствах по-прежнему могут быть изменены внутри:

```php
<?php
class Test {
public function __construct(public readonly object $obj) {}
}
$test = new Test(new stdClass);
// Правильное внутреннее изменение.
$test->obj->foo = 1;
// Неправильное переопределение.
$test->obj = new stdClass;
?>
```

Динамические свойства

При попытке присвоить несуществующее свойство объекту (object), PHP автоматически создаст соответствующее свойство. Это динамически созданное свойство будет доступно *только* для данного экземпляра класса.

Внимание

Динамические свойства устарели, начиная с PHP 8.2.0. Вместо этого рекомендуется объявлять свойство. Для работы с произвольными именами свойств, класс должен реализовать магические методы __get() и __set(). В крайнем случае, класс можно пометить атрибутом `#[\AllowDynamicProperties]`.

User Contributed Notes 5 notes

340

Anonymous ¶

11 years ago

In case this saves anyone any time, I spent ages working out why the following didn't work:

```
class MyClass
{
private $foo = FALSE;

public function __construct()
{
$this->$foo = TRUE;

echo($this->$foo);
}
}

$bar = new MyClass();
```

giving "Fatal error: Cannot access empty property in ...test_class.php on line 8"

The subtle change of removing the $ before accesses of $foo fixes this:

```
class MyClass
{
private $foo = FALSE;

public function __construct()
{
$this->foo = TRUE;

echo($this->foo);
}
}

$bar = new MyClass();
```

I guess because it's treating $foo like a variable in the first example, so trying to call $this->FALSE (or something along those lines) which makes no sense. It's obvious once you've realised, but there aren't any examples of accessing on this page that show that.

up

down

78

anca at techliminal dot com ¶

8 years ago

You can access property names with dashes in them (for example, because you converted an XML file to an object) in the following way:

```
<?php
$ref = new StdClass();
$ref->{'ref-type'} = 'Journal Article';
var_dump($ref);
?>
```

up

down

66

Anonymous ¶

12 years ago

$this can be cast to array. But when doing so, it prefixes the property names/new array keys with certain data depending on the property classification. Public property names are not changed. Protected properties are prefixed with a space-padded '*'. Private properties are prefixed with the space-padded class name...

```
<?php

class test
{
public $var1 = 1;
protected $var2 = 2;
private $var3 = 3;
static $var4 = 4;

public function toArray()
{
return (array) $this;
}
}

$t = new test;
print_r($t->toArray());

/* outputs:

Array
(
[var1] => 1
[ * var2] => 2
[ test var3] => 3
)

*/
?>
```

This is documented behavior when converting any object to an array (see </language.types.array.php#language.types.array.casting> PHP manual page). All properties regardless of visibility will be shown when casting an object to array (with exceptions of a few built-in objects).

To get an array with all property names unaltered, use the 'get_object_vars($this)' function in any method within class scope to retrieve an array of all properties regardless of external visibility, or 'get_object_vars($object)' outside class scope to retrieve an array of only public properties (see: </function.get-object-vars.php> PHP manual page).

up
down
26

zzzzBov ¶

13 years ago

Do not confuse php's version of properties with properties in other languages (C++ for example). In php, properties are the same as attributes, simple variables without functionality. They should be called attributes, not properties.

Properties have implicit accessor and mutator functionality. I've created an abstract class that allows implicit property functionality.

```
<?php

abstract class PropertyObject
{
public function __get($name)
{
if (method_exists($this, ($method = 'get_'.$name)))
{
return $this->$method();
}
else return;
}

public function __isset($name)
```

```
{
if (method_exists($this, ($method = 'isset_'.$name)))
{
return $this->$method();
}
else return;
}

public function __set($name, $value)
{
if (method_exists($this, ($method = 'set_'.$name)))
{
$this->$method($value);
}
}

public function __unset($name)
{
if (method_exists($this, ($method = 'unset_'.$name)))
{
$this->$method();
}
}
}

?>
```

after extending this class, you can create accessors and mutators that will be called automagically, using php's magic methods, when the corresponding property is accessed.

up
down
0

kchlin dot lxy at gmail dot com ¶

1 year ago

From PHP 8.1
It's easy to create DTO object with readonly properties and promoting constructor
which easy to pack into a compact string and covert back to a object.

```
<?php
# Conversion functions.
# Pack object into a compack JSON string.
function cvtObjectToJson( object $poObject ): string
{
return json_encode( array_values( get_object_vars( $poObject )));
}

# Unpack object from a JSON string.
function cvtJsonToObject( string $psClass, string $psString ): object
{
return new $psClass( ...json_decode( $psString ));
}

# DTO example class.
final class exampleDto
{
final public function __construct(
public readonly int $piInt,
public readonly ?int $pnNull,
public readonly float $pfFloat,
public readonly string $psString,
public readonly array $paArray,
public readonly object $poObject,
){}
```

```
}

# Example with export only public properties of given object.
$exampleDtoO = new exampleDto( 1, null, .3, 'string 4', [], new stdClass() );
$stringJson = cvtObjectToJson( $exampleDtoO );// [1,null,0.3,"string 4",[],{}]
$objectO = cvtJsonToObject( exampleDto::class, $stringJson );

# Check and var_dump variables.
echo $exampleDtoO == $objectO
? 'Objects equal, but not identical.'. PHP_EOL
: 'Objects not equal neither identical.'. PHP_EOL
;
var_dump($exampleDtoO, $stringJson, $objectO);

# Output
/*
Objects equal, but not identical
object(exampleDto)#6 (6) {
["piInt"]=>
int(1)
["pnNull"]=>
NULL
["pfFloat"]=>
float(0.3)
["psString"]=>
string(8) "string 4"
["paArray"]=>
array(0) {
}
["poObject"]=>
object(stdClass)#7 (0) {
}
}
string(29) "[1,null,0.3,"string 4",[],{}]"
object(exampleDto)#8 (6) {
["piInt"]=>
int(1)
["pnNull"]=>
NULL
["pfFloat"]=>
float(0.3)
["psString"]=>
string(8) "string 4"
["paArray"]=>
array(0) {
}
["poObject"]=>
object(stdClass)#9 (0) {
}
}
*/
```