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

Объекты и ссылки »
« Сравнение объектов

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Позднее статическое связывание

PHP реализует функцию, называемую позднее статическое связывание, которая может быть использована для того, чтобы получить ссылку на вызываемый класс в контексте статического наследования.

Если говорить более точно, позднее статическое связывание сохраняет имя класса указанного в последнем "неперенаправленном вызове". В случае статических вызовов это явно указанный класс (обычно слева от оператора `::`); в случае не статических вызовов это класс объекта. "Перенаправленный вызов" - это статический вызов, начинающийся с `self::`, `parent::`, `static::`, или, если двигаться вверх по иерархии классов, forward_static_call(). Функция get_called_class() может быть использована для получения строки с именем вызванного класса, а `static::` представляет её область действия.

Само название "позднее статическое связывание" отражает в себе внутреннюю реализацию этой особенности. "Позднее связывание" отражает тот факт, что обращения через `static::` не будут вычисляться по отношению к классу, в котором вызываемый метод определён, а будут вычисляться на основе информации в ходе исполнения. Также эта особенность была названа "статическое связывание" потому, что она может быть использована (но не обязательно) в статических методах.

Ограничения `self::`

Статические ссылки на текущий класс, такие как `self::` или `__CLASS__`, вычисляются используя класс, к которому эта функция принадлежит, как и в том месте, где она была определена:

Пример #1 Использование `self::`

```
<?php
class A {
public static function who() {
echo __CLASS__;
}
public static function test() {
self::who();
}
}

class B extends A {
public static function who() {
echo __CLASS__;
}
}

B::test();
?>
```

Результат выполнения приведённого примера:

```
A
```

Использование позднего статического связывания

Позднее статическое связывание пытается устранить это ограничение, предоставляя ключевое слово, которое ссылается на класс, вызванный непосредственно в ходе выполнения. Попросту говоря, ключевое слово, которое позволит вам ссылаться на `B` из `test()` в предыдущем примере. Было решено не вводить новое ключевое слово, а использовать `static`, которое уже зарезервировано.

Пример #2 Простое использование `static::`

```
<?php
class A {
public static function who() {
echo __CLASS__;
}
```

```
public static function test() {
static::who(); // Здесь действует позднее статическое связывание
}
}

class B extends A {
public static function who() {
echo __CLASS__;
}
}

B::test();
?>
```

Результат выполнения приведённого примера:

```
B
```

> **Замечание**:
>
> В нестатическом контексте вызванным классом будет тот, к которому относится экземпляр объекта. Поскольку `$this->` будет пытаться вызывать закрытые методы из той же области действия, использование `static::` может дать разные результаты. Другое отличие в том, что `static::` может ссылаться только на статические поля класса.

Пример #3 Использование `static::` в нестатическом контексте

```
<?php
class A {
private function foo() {
echo "success!\n";
}
public function test() {
$this->foo();
static::foo();
}
}

class B extends A {
/* foo() будет скопирован в B, следовательно его область действия по прежнему A,
и вызов будет успешным */
}

class C extends A {
private function foo() {
/* исходный метод заменён; область действия нового метода - C */
}
}

$b = new B();
$b->test();
$c = new C();
$c->test(); // потерпит ошибку
?>
```

Результат выполнения приведённого примера:

```
success!
success!
success!


Fatal error:  Call to private method C::foo() from context 'A' in /tmp/test.php on line 9
```

> **Замечание**:

Разрешающая область позднего статического связывания будет фиксирована вычисляющем её статическим вызовом. С другой стороны, статические вызовы с использованием таких директив как `parent::` или `self::` перенаправляют информацию вызова.

Пример #4 Перенаправленные и неперенаправленные вызовы

```
<?php
class A {
public static function foo() {
static::who();
}

public static function who() {
echo __CLASS__."\n";
}
}

class B extends A {
public static function test() {
A::foo();
parent::foo();
self::foo();
}

public static function who() {
echo __CLASS__."\n";
}
}
class C extends B {
public static function who() {
echo __CLASS__."\n";
}
}

C::test();
?>
```

Результат выполнения приведённого примера:

```
A
C
C
```


User Contributed Notes 33 notes

164
sergei at 2440media dot com ¶
15 years ago
Finally we can implement some ActiveRecord methods:

```
<?php

class Model
{
public static function find()
{
echo static::$name;
}
}
```

```
class Product extends Model
{
protected static $name = 'Product';
}

Product::find();

?>

Output: 'Product'
```

up
down
22

MelkiySoft ¶

5 years ago

```
<?php

class A
{

}

class B extends A
{
public static function foo () {
echo 'new self: ';
var_dump(new self());
echo '<br>new parent: ';
var_dump(new parent());
echo '<br>new static: ';
var_dump(new static());
}
}

class C extends B
{

}

c::foo();
============================
output:
//new self: object(B)#1 (0) { }
//new parent: object(A)#1 (0) { }
//new static: object(C)#1 (0) { }
```

up
down
61

mhh1422 at hotmail dot com ¶

10 years ago

For abstract classes with static factory method, you can use the static keyword instead of self like the following:

```
<?php

abstract class A{

static function create(){

//return new self(); //Fatal error: Cannot instantiate abstract class A

return new static(); //this is the correct way

}
```

```
}

class B extends A{
}

$obj=B::create();
var_dump($obj);

?>
```

up
down
12

sskaje at gmail dot com ¶

8 years ago

static::class and self::class can be used to get current class name,
work under 5.5 and 5.6
failed in 5.3.

```
<?php
class a{
function d() {
echo "=== self::class ===\n";
var_dump(self::class);
echo "=== static::class ===\n";
var_dump(static::class);
}
}
class b extends a{}
class c extends b{}

a::d();
b::d();
c::d();

/*
Output:

=== self::class ===
string(1) "a"
=== static::class ===
string(1) "a"
=== self::class ===
string(1) "a"
=== static::class ===
string(1) "b"
=== self::class ===
string(1) "a"
=== static::class ===
string(1) "c"

*/
```

up
down
6

backnot ¶

2 years ago

In the above example (#3) in order to make it work, you can change the child's method from 'private' to 'protected' (or public) and it will be called through 'static'.

```
<?php
class A {
```

```
private function foo() {
echo "success!\n";
}
public function test() {
$this->foo();
static::foo();
}
}

class B extends A {
/* foo() will be copied to B, hence its scope will still be A and
* the call be successful */
}

class C extends A {
protected function foo() { //note the change here
echo 'hello world!';
}
}

$b = new B();
$b->test();
$c = new C();
$c->test(); // 'success' 'hello world'
?>
```

up
down
6

tyler AT canfone [dot] COM ¶

15 years ago

```
@ php at mikebird

You can pass arguments to your constructor through your getInstance method, assuming you are running php5.

public static function getInstance($params = null) {
if (self::$objInstance == null) {
$strClass = static::getClass();
self::$objInstance = new $strClass($params);
}
return self::$objInstance;
}

This would pass the params to your constructor. Love for php.
```

up
down
10

tamilps2 at gmail dot com ¶

9 years ago

```
I have implemented enum using late static binding.

<?php
interface IEnum {
/**
* Only concrete class should implement this function that should behave as
* an enum.
*
* This method should return the __CLASS__ constant property of that class
*
* @return string __CLASS__
*/
public static function who();
}
```

```
abstract class Enum {

/**
* The selected value for the enum implementation
*
* @var mixed
*/
public $value;

public function __construct($value) {
$this->value = $value;
}

/**
* The factory method that creates the corresponding enum class.
*
* @param integer $type
* @return false|\class
*/
public static function Factory($type) {
if (empty($type)) {
return false;
}

// use of late static binding to get the class.
$class = static::who();

if (array_key_exists($type, static::$_enums)) {
return new $class($type);
}

return false;
}

public function getValue() {
return $this->value;
}

public static function getValues() {
return array_keys(static::$_enums);
}

public function getString() {
return static::$_enums[$this->value];
}

public function __toString() {
return static::$_enums[$this->value];
}

}

class Fruits extends Enum implements IEnum {

public static $_enums = array(
1 => 'Apple'
2 => 'Orange'
3 => 'Banana'
)

public static function who() {
```

```
return __CLASS__;
}
}

// Usage

// user input from dropdown menu of fruits list
$input = 3;

$fruit = Fruits::Factory($input);

$fruit->getValue(); // 3
$fruit->getString(); // Banana
?>
```

up
down
5

steven dot karas+nospam at gmail dot com ¶

13 years ago

This function can be used as a workaround for late static binding in PHP >= 5.1.0. There was another similar version of this function elsewhere, but used eval.

```
<?php

function & static_var($class, $name)
{
if (is_object($class))
{
$class = get_class($class);
}
elseif ( ! is_string($class))
{
throw new Exception('Must be given an object or a class name', NULL);
}

$class = new ReflectionClass($class);
return $class->getStaticPropertyValue($name);
}

?>
```

up
down
8

jakub dot lopuszanski at nasza-klasa dot pl ¶

13 years ago

Suprisingly consts are also lazy bound even though you use self instead of static:

```
<?php
class A{
const X=1;
const Y=self::X;
}
class B extends A{
const X=1.0;
}
var_dump(B::Y); // float(1.0)
?>
```

up
down
7

Andrea Giammarchi ¶

15 years ago

About static parameters, these work as expected.

```
<?php
class A {
protected static $__CLASS__ = __CLASS__;
public static function constructor(){
return static::$__CLASS__;
}
}

class B extends A {
protected static $__CLASS__ = __CLASS__;
}

echo B::constructor(); // B
?>
```

up
down
6

***kx* ¶**

15 years ago

At least as of PHP 5.3.0a2 there's a function get_called_class(), which returns the class on which the static method is called.

```
<?php

class a {
static public function test() {
print get_called_class();
}
}

class b extends a {
}

a::test(); // "a"
b::test(); // "b"

?>
```

up
down
3

***adam dot prall at thinkingman dot com* ¶**

13 years ago

Just a quick reminder to always check your syntax. While I love LSB, I thought it wasn't working:

```
static::$sKey = not set
```

…until I realized that I'd completely forgotten to make it a variable variable:

```
$sKey = 'testStaticClassVarNameThatExistsInThisClassesScope';

static::$$sKey = is set
```

…of course this applies anywhere in PHP, but because of the (current) newness late static bindings, I've seen lots of code with this particular snafu in it from others.

up
down
1

***joost dot t dot hart at planet dot nl* ¶**

14 years ago

PHP5.3 unavailable, yet in the need for 'static', I did the following.

Any objections? Personally I hate using the the eval() statement...

```
<?php

class mother
{
function setStatic( $prop, $val ) {
// After this, self:: refers to mother, yet next $class refers to...
//
$class = get_class( $this );
eval( "$class::\$$prop = \$$val;" );
}
}

class child extends mother
{
protected static $sProp;

function writer( $value ) {
parent::setStatic( 'sProp', $value );
}
function reader()
{
return self::$sProp;
}
}

$c = new child();
$c->writer( 3 );
echo $c->reader(); // 3

?>
```

up
down
2
5imun at github dot com ¶
1 year ago

Example of setting up static property in child class from parent only if it isn't already defined, many people would expect that output will be "Foo Bar" but instead we get "Foo Foo":

```
<?php
class Foo
{
public static string $A;

public static function init() {
return "Foo";
}
public static function get() {
if (!isset(static::$A)) {
static::$A = static::init();
}
return static::$A;
}
}

class Bar extends Foo {
public static function init() {
return "Bar";
}
}

$foo = new Foo();
```

```
$bar = new Bar();

echo $foo->get();
echo $bar->get();
?>

Output:
Foo
Foo
```

up
down
2

Taai ¶

11 years ago

I discovered an interesting thing. The class name string must be accessed directly from "flat" variable. Late static binding code that get's it's variable from array that is passed by class instance, throws an syntax error. Bug?

```
<?php
class A {

public $metadata = array('class' => 'A');

public static function numbers()
{
return 123;
}

}

$instance = new A();

// This throws an error
// Parse error: syntax error, unexpected '::' (T_PAAMAYIM_NEKUDOTAYIM)
var_dump( $instance->metadata['class']::numbers() );

// Get the class name and store it in "flat" variable and now it's ok
$class_name = $instance->metadata['class'];
var_dump( $class_name::numbers() );

// Other tests ------------------------------------------

$arr = array('class' => 'A');

// This works too.
var_dump( $arr['class']::numbers() );
?>
```

up
down
0

MikeT ¶

1 year ago

Word of caution static::class doesn't always work as you might expect

```
<?php
namespace NameSpace;

class Class
{
static function getClass()
{
return static::class;
}
}
```

```
Class::getClass()
?>
may return \NameSpace\Class or Class depending on context
```

up
down
0

aabweber at gmail dot com ¶

1 year ago

Simplest way to understand is to run this script:

```
<?php
class ParentClass
{
static $A = 'ParentVariable';

static function parentCall()
{
echo get_called_class() . ', self: ' . self::$A . "\n";
echo get_called_class() . ', static: ' . static::$A . "\n";
echo "---\n";
}
}

class ChildClass extends ParentClass
{
static $A = 'ChildVariable';

static function childCall()
{
echo get_called_class() . ', self: ' . self::$A . "\n";
echo get_called_class() . ', static: ' . static::$A . "\n";
echo get_called_class() . ', parent: ' . parent::$A . "\n";
echo "---\n";
}
}

echo "Late Static Bindings:\n";
ParentClass::parentCall();
ChildClass::parentCall();
ChildClass::childCall();
?>

----
Output:

Late Static Bindings:
ParentClass, self: ParentVariable
ParentClass, static: ParentVariable
---
ChildClass, self: ParentVariable
ChildClass, static: ChildVariable
---
ChildClass, self: ChildVariable
ChildClass, static: ChildVariable
ChildClass, parent: ParentVariable
```

up
down
0

Anonymous ¶

1 year ago

```
class P_Class {
```

```
public static $val = "Parent";
public static function setVal($val){
static::$val = $val;
}
public static function getVal(){
return static::$val;
}
}

class C_Class extends P_Class{}

C_Class::setVal("Child");
var_dump(C_Class::getVal());
var_dump(P_Class::getVal());

Output:
string(5) "Child"
string(5) "Child"
```

up
down
-2

5imun at github dot com ¶

1 year ago

You must be careful when getting static property of extended class from parent class, in example below you can see that using property_exists (method getA2) instead of isset with static keyword (method getA1) to check if the static property exist gives much more intuitive result:

```
<?php
class Foo
{
public static string $A;

public static function init() {
return static::class;
}

public static function getA1() {
if (!isset(static::$A)) {
static::$A = static::class;
}
return static::$A;
}

public static function getA2() {
if (property_exists(static::class, 'A')) {
static::$A = static::class;
}
return static::$A;
}
}
class Bar extends Foo {}

$foo = new Foo();
echo $foo->getA1();
echo $foo->getA2();
echo $foo->getA1();

$bar = new Bar();
echo $bar->getA1();
echo $bar->getA2();
echo $bar->getA1();
?>
```

```
Output:
Foo
Foo
Foo

Foo
Bar
Bar

Notice how $bar->getA1() returns "Foo" instead of "Bar" that many people would expect to see.
```

up
down
0

gern_ at hotmail dot com ¶

15 years ago

```
get_called_class for PHP < 5.3

<?php
/**
* Return called class name
*
* @author Michael Grenier
* @param int $i_level optional
* @return string
*/
function get_called_class ($i_level = 1)
{
$a_debug = debug_backtrace();
$a_called = array();
$a_called_function = $a_debug[$i_level]['function'];
for ($i = 1, $n = sizeof($a_debug); $i < $n; $i++)
{
if (in_array($a_debug[$i]['function'], array('eval')) ||
strpos($a_debug[$i]['function'], 'eval()') !== false)
continue;
if (in_array($a_debug[$i]['function'], array('__call', '__callStatic')))
$a_called_function = $a_debug[$i]['args'][0];
if ($a_debug[$i]['function'] == $a_called_function)
$a_called = $a_debug[$i];
}
if (isset($a_called['object']) && isset($a_called['class']))
return (string)$a_called['class'];
$i_line = (int)$a_called['line'] - 1;
$a_lines = explode("\n", file_get_contents($a_called['file']));
preg_match("#([a-zA-Z0-9_]+){$a_called['type']}
{$a_called['function']}( )*\(#", $a_lines[$i_line], $a_match);
unset($a_debug, $a_called, $a_called_function, $i_line, $a_lines);
if (sizeof($a_match) > 0)
$s_class = (string)trim($a_match[1]);
else
$s_class = (string)$a_called['class'];
if ($s_class == 'self')
return get_called_class($i_level + 2);
return $s_class;
}
?>
```

up
down
0

deadimp at gmail dot com ¶

15 years ago

I think this will be pretty helpful too.
My question is, can just 'static' by itself resolve to the late static class?
I ask this because it could help in making new instances of the derived class, from a base class, by calling a derived class's static method instead of having to create a new instance of the derived class - or explicitly defining a 'getClass' method for each derived class.
Example:

```
<?php
//There isn't really any purpose for this example I posted
//Just a random implementation
class Base {
static function useful() {
//Create a list of instances of the derived class
$list=array();
for ($i=0;$i<10;$i++) $list[]=new static(); //Here's the point in question
return $list;
}
}
class Derived extends Base {
static function somethingElse() {
//...
$list=static::useful();
}
}
?>
```

I'm not sure what kind of lexical / whatever-it's-called problems this would make with parsing. I don't think it could really collide with any contexts where you would use static otherwise - variable / method declaration.

Even more so, is there a way to get the class's name to which the keywords 'self', 'parent', or 'static' refer?
Example:

```
<?php
class Base {
static function stuff() {
echo "Self: ".get_class(self);
echo "Parent: ".get_class(parent);
echo "Derived: ".get_class(static);
}
}
class Derived extends Base {
static function stuff() {
static::stuff();
}
}
?>
```

I don't think there should be a massive bloat in the PHP core to support all of this, but it would be nice to take advantage of the dynamic nature of PHP.

And yet another side note:
If you're in the instance-level scope in a method of a base, and you want to get a top-level static, here's an ugly workaround (from Thacmus /lib/core.php - see SVN repo):

```
<?php
//Get reference [?] to static from class
//$class - Class name OR object (uses get_class())
//$var - Not gonna say
function& get_static($class,$var) { //'static_get'?
if (!is_string($class)) $class=get_class($class);
if (!@property_exists($class,$var)) {
trigger_error("Static property does not exist: $class::\$$var");
//debug_callstack(); //This is just a wrapper for debug_backtrace() for HTML
return null;
}
//Store a reference so that the base data can be referred to
```

```
//The code [[ return eval('return &'.$class.'::$'.$var.';') ]] does not work - can not return references...
//To establish the reference, use [[ $ref=&get_static(...) ]]
eval('$temp=&'.$class.'::$'.$var.';'); //using
return $temp;
}
?>
```

up
down
0

max at mastershrimp dot com ¶

15 years ago

If you are using PHP < 5.3.0 you might be interested in the following workaround for late static binding:
http://de2.php.net/manual/de/function.get-class.php#77698

up
down
-1

php at mikebird dot co dot uk ¶

15 years ago

This should make life easier and neater if you have a project with a lot of singleton classes e.g.

```
<?php

class Singleton {

public static $objInstance;

public static function &getInstance() {
if (self::$objInstance == null) {
$strClass = static::getClass();
self::$objInstance = new $strClass;
}
return self::$objInstance;
}

public static function getClass() {
return __CLASS__;
}

}

class Foo extends Singleton {

public $intBar;

public function __construct() {
$this->intBar = 1;
}

public static function getClass() {
return __CLASS__;
}

}



$objFooTwo = Foo::getInstance();
$objFooTwo->intBar = 2;

$objFooOne = Foo::getInstance();

if ($objFooOne->intBar == $objFooTwo->intBar) {
echo 'it is a singleton';
```

```
} else {
echo 'it is not a singleton';
}

?>
```

The above will output 'it is a singleton'. The obvious downfall to this method is not being able to give arguments to the constructor.

up
down
-1

sebastien at info-conseil dot fr ¶

15 years ago

Here is a small workaround I made for the static inheritance issue. It's not perfect, but it works.

```
<?php

// BaseClass class will be extended by any class needing static inheritance workaroud
class BaseClass {
// Temporarily stores class name for Entry::getStatic() and Entry::setNextStatic()
protected static $nextStatic = false;

// Returns the real name of the class calling the method, not the one in which it was declared.
protected static function getStatic() {
// If already stored
if (self::$nextStatic) {
// Clean and return
$class = self::$nextStatic;
self::$nextStatic = false;
return $class;
}

// Init
$backTrace = debug_backtrace();
$class = false;

// Walk through
for ($i=0; $i<count($backTrace); $i++) {
// If a class is defined
if (isset($backTrace[$i]['class'])) {
// Check if it is not a basic class
if (!in_array($backTrace[$i]['class'], array('BaseClass', 'GenericClass'))) {
return $backTrace[$i]['class'];
} else {
$class = $backTrace[$i]['class'];
}
} else {
// Returns last known class
return $class;
}
}

// Default
return $class;
}

// If a static method is called within global env, the previous method won't work, so we need to tell BaseClass which
public static function setNextStatic($class) {
// Save value
self::$nextStatic = $class;
}
}
```

```
// Generic class declaring various static methods
class GenericClass extends BaseClass {
public static $name = 'Generic';

public function getName() {
$static = get_class_vars(get_class($this));
return $static['name'];
}

public static function basicClassName() {
return self::$name;
}

public static function staticClassName() {
// Get real name
$staticName = self::getStatic();

// Return final class name
$static = get_class_vars($staticName);
return $static['name'];
}
}

// Final class
class SomeClass extends GenericClass {
public static $name = 'Some';

public static function returnClassNameWith($string) {
return $string.' : '.self::staticClassName();
}
}

// Instance call

// Will print 'Some'
$a = new SomeClass();
echo 'Name of $a : '.$a->getName().'<br />';

// Static calls

// Will print 'Generic'
echo 'Basic call to SomeClass::$name : '.SomeClass::basicClassName().'<br />';

// Will print 'Generic'
echo 'Global call to SomeClass::$name : '.SomeClass::staticClassName().'<br />';

// Will print 'Some'
BaseClass::setNextStatic('SomeClass');
echo 'Global call to SomeClass::$name with pre-set : '.SomeClass::staticClassName().'<br />';

// Will print 'Some'
echo 'Internal call to SomeClass::$name : '.SomeClass::returnClassNameWith('This is a ').'<br />';

?>
```

There are two issues with this workaround :
- if you call a static method from global env, you need to declare the name of the class BEFORE calling the method, otherwise the workaround won't work (see 3rd and 4th examples). But I assume good programming makes few calls to static methods from global scope, so this shouldn't be long to fix if you use it.
- the workaround fails to access to private or protected static vars, as it uses get_class_vars(). If you find any better solution, let us know.

With Php 5.3.0, upgrading will be easy : just delete the methods from the basic class, and search/replace any call to getStatic() and setNextStatic() by static:: - or one could use a selector on PHP_VERSION value to include either the BaseClass file with workaround or a BaseClass file using static::

up
down
-2

iamscrumpyjack ¶
15 years ago

I have been dying to see this issue resolved. I'm very much looking forward to the production release of PHP 5.3...

In my case I have been trying to do the following:

```
class A {
function __construct() {
echo "I was called by " . static::__CLASS__;
}
}

class B extends A {
function Foo() {
echo "I am class " . __CLASS__;
}
}

$b = new B; // Should echo "I was called by B"
$b->Foo(); // Should echo "I am class B"
```

At the moment I do the following workaround:

```
class A {
function __construct($child) {
echo "I was called by " . $child;
}
}

class B extends A {
function __construct() {
parent::__construct(__CLASS__);
}

function Foo() {
echo "I am class " . __CLASS__;
}
}

$b = new B; // Echos "I was called by B"
$b->Foo(); // Echo "I am class B"
```

As you can see, my current workaround has some overhead and is not as water-tight as the late static binding method.

up
down
-2

martinpauly [at] google mail [dot] com ¶
15 years ago

will this work for variables as well?

it would be great, if the following worked:

```
<?php
class A {
protected static $table = "table";
```

```
public static function connect(){
//do some stuff here
echo static::$table;
return static::getInstance(); //function getInstance() now can return classes A or B depending on the context it was
called
}
...
}

class B extends A {
protected static $table = "subtable";
...
}

$table = B::connect(); //hopefully the output will be: subtable
?>
```

up
down
-5

Anonymous ¶
13 years ago

```
THIS WORKED GREAT FOR ME:

<?php
abstract class ParentClass
{
static function parent_method()
{
$child_class_str = self::get_child_class();
eval("\$r = ".$child_class_str."::abstract_static();");
return $r;
}// CHILD MUST OVERRIDE TO PUT ITSELF INTO TRACE

protected abstract static function abstract_static();

private static function get_child_class()
{
$backtrace = debug_backtrace();
$num = count($backtrace);
for($i = 0; $i < $num; $i++)
{
if($backtrace[$i]["class"] !== __CLASS__)
return $backtrace[$i]["class"];
}
return null;
}
}

class ChildClass extends ParentClass
{
static function parent_method(){ return parent::parent_method(); }

protected static function abstract_static()
{
return __METHOD__."()";
}// From ParentClass
}

print "The call was: ". ChildClass::parent_method();
?>
```

up
down

-7

tfn dot yldrm at hotmail dot com ¶

13 years ago

example for static binding on 5.2 and real enumaration

```
<?php
/**
* Static Binding On PHP 5.2
* @author Tufan Baris YILDIRIM
* @since 26.10.2010
*/
abstract class EnumBase
{
protected $num = 0;
public $toString;
public $toInt;

public function __construct($enumKeyOrVal)
{

unset($this->toString,$this->toInt);

$enums = $this->enums();

if(
empty($enumKeyOrVal)
||
!(isset($enums[$this->num = $enumKeyOrVal])
||
($this->num = array_search($enumKeyOrVal,$enums)) !== false)
)
$this->num = 0;

/**
* 5.3 Version
*/
/*
* if(
empty($enumKeyOrVal)
||
!(isset(static::$enums[$this->num = $enumKeyOrVal])
||
($this->num = array_search($enumKeyOrVal,static::$enums)) !== false)
)
$this->num = 0;
*/
}

#5.3 e geçilirse gerek kalmaz.
public function vars()
{
return get_class_vars(get_class($this));
}

public function enums()
{
$vars = $this->vars();
return $vars['enums'];
}

public function __get($property)
{
if(method_exists($this,'__'.$property))
```

```
return $this->{'__'.$property}();
else
return $this->__toString();
}

public function __toInt()
{
return $this->num;
}

public function __toString()
{

$enums = $this->enums();

if(isset($enums[$this->num]))
{
return $enums[$this->num];
}
else
{
return $enums[0];
}

/**
* 5.3 Version
*/

/*
if(isset(static::$enums[$this->num]))
{
return static::$enums[$this->num];
}
else
{
return static::$enums[0];
}
*/
}
}

class Positions extends EnumBase
{
public static $enums = array(
0 => 'Bilinmiyor',
1 => 'Kale',
2 => 'Defans',
3 => 'Orta Saha',
4 => 'Forvet'
);
}



$a = new Positions('Orta Saha');
$b = new Positions(4);

$c = (string)$a; // Orta Saha
$d = (string)$b; // Forvet
?>
```

up
down
-5

jrfish dot x at gmail dot com ¶

13 years ago

```
consider this:

<?php
class A
{

// some stuff....

public static function getInstance()
{
return new self();
}

}

class B extends A
{
//stuff...
}

$obj = B::getInstance();

//versus

class A
{

// some stuff....

public static function getInstance()
{
return new static();
}

}

class B extends A
{
//stuff...
}

$obj = B::getInstance();
?>

also works the same way with static variables and constants
```

up

down

-7

kenneth at kennethjorgensen dot com ¶

14 years ago

```
Simple basic class which uses to get_called_class() to create singleton instances. A previous post by php at mikebird dot co dot uk explain how to do this, but the extended static variables require you to define them in child classes before they work.

<?php

abstract class Singleton {
private static $instances = array();

public function __construct() {
```

```
$class = get_called_class();
if (array_key_exists($class, self::$instances))
trigger_error("Tried to construct a second instance of class \"$class\"", E_USER_WARNING);
}

public static function getInstance() {
$class = get_called_class();
if (array_key_exists($class, self::$instances) === false)
self::$instances[$class] = new $class();
return self::$instances[$class];
}
}

class A extends Singleton {
}

class B extends Singleton {
}

$a1 = A::getInstance();
$a2 = A::getInstance();
$b1 = B::getInstance();
$b2 = B::getInstance();

if (get_class($a1) == "A" &&
get_class($a2) == "A" &&
get_class($b1) == "B" &&
get_class($b2) == "B" &&
$a1 === $a2 &&
$b1 === $b2)
echo "All good\n";
else
echo "FAIL!\n";

?>
```

You probably noticed the use of self:: rather than static::, this is because we want the static variable to be private, and using static:: will not allow us to do that.

up
down
-10

***tom* ¶**

14 years ago

Something you may find useful for passive code-testing:

```
<?php
class BaseClass {
function __get($id) {
throw new Exception("Trying to access undefined property '$id'.");
}
function __set($id) {
throw new Exception("Trying to access undefined property '$id'.");
}
}

class MyClass extends BaseClass {
// my implementation
}
?>
```

Using these magic function as described above will help you to find classes that try to access an undefined (and undocumented) class-member. In most cases: this is an error based on misspelled member names.

up
down
-6
Anonymous ¶
15 years ago
Trying to recreate an inheritable static part for an object through a singleton pattern.

```
<?php
/**
* "Inheritable static" for PHP < 5.3
* << Library/Inheritable.php >>
*/

abstract class Inheritable_Static extends Singleton
{
}

abstract class Inheritable
{
public static function getStatic($className)
{
// Use an abstract Singleton
return Singleton::getInstance($className . '_Static') ;
}

public function goStatic()
{
return self::getStatic(get_class($this)) ;
}
}

/**
* Abstract
* << Library/SayIt/Abstract.php >>
*/

abstract class SayIt_Abstract_Static extends Inheritable_Static
{
public $format ;
}

abstract class SayIt_Abstract extends Inheritable
{
protected $_name ;

public function __construct($name)
{
$this->_name = $name ;
}

final public function sayIt()
{
echo sprintf($this->goStatic()->format, $this->_name) . "\n" ;
}

}

/**
* Concrete
* << Library/SayIt/Hello.php >>
*/
```

```
class SayIt_Hello_Static extends SayIt_Abstract_Static
{
}

class SayIt_Hello extends SayIt_Abstract
{
public static function getStatic() { return parent::getStatic(__CLASS__) ; }
}

/**
* Test
*/

SayIt_Hello::getStatic()->format = 'Hello %s' ;

$w = new SayIt_Hello('World') ;
$j = new SayIt_Hello('Joe') ;

echo $w->sayIt() ; // Hello World
echo $j->sayIt() ; // Hello Joe
```

up
down
-9
jrfish dot x at gmail dot com ¶
13 years ago
also works the same way with static variables and constants
＋add a note