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

FAQ »
« Возврат к глобальному пространству

- Руководство по PHP
- Справочник языка
- Пространства имён

Change language: Russian

Правила разрешения имён

(PHP 5 >= 5.3.0, PHP 7, PHP 8)

Для целей этих правил разрешения приведём важные определения:

Определения имени пространства имён
Неполное имя

Идентификатор без разделителя пространств имён, например `Foo`

Полное имя

Идентификатор с разделителем пространств имён, например `Foo\Bar`

Абсолютное имя

Идентификатор с разделителем пространств имён, который начинается с разделителя пространств имён, например `\Foo\Bar`. Пространство имён `\Foo` — также абсолютное имя.

Относительное имя

Идентификатор, который начинается с ключевого слова `namespace`, например `namespace\Foo\Bar`.

Имена разрешаются по следующим правилам:

1. Абсолютные имена разрешаются в имя без ведущего разделителя пространства имён. Например, `\A\B` разрешается в `A\B`.
2. Относительные имена разрешаются в имя с заменой ключевого слова `namespace` текущим пространством имён. Если имя встречается в глобальном пространстве имён, префикс `namespace\` удаляется. Например, имя `namespace\A` внутри пространства имён `X\Y` разрешается в `X\Y\A`. То же имя в глобальном пространстве имён разрешается в `A`.
3. В полных именах первый сегмент имени преобразовывается с учётом текущей таблицы импорта класса или пространства имён. Например, если пространство имён `A\B\C` импортировано как `C`, то имя `C\D\E` преобразуется в `A\B\C\D\E`.
4. В полных именах, если не применялось правило импорта, текущее пространство имён добавляется к имени. Например, имя `C\D\E` внутри пространства имён `A\B` разрешится в `A\B\C\D\E`.
5. Неполные имена преобразовываются с учётом текущей таблицы импорта и типа элемента. То есть имена как у классов преобразовываются с учётом таблицы импорта классов или пространств имён, имена функций — с учётом таблицы импорта функций, а константы — таблицы импорта констант. Например, при записи `use A\B\C;`, вызов `new C()` разрешается в `A\B\C()`. Аналогично, при записи `use function A\B\foo;` вызов `foo()` разрешается в `A\B\foo`.
6. В начало неполных имён, если не применялось правило импорта и имя относится к элементу с именем как у класса, добавляется текущее пространство имён. Например, имя класса в выражении `new C()` внутри пространства имён `A\B` разрешится в имя `A\B\C`.
7. В неполных именах, если не применялось правило импорта и имя относится к функции или константе, а код лежит за пределами глобального пространства имён, имя разрешается при выполнении. Вот как разрешится вызов функции `foo()` в коде в пространстве имён `A\B`:
   1. Выполняется поиск функции из текущего пространства имён: `A\B\foo()`.
   2. PHP пытается найти и вызвать функцию `foo()` из *глобального пространства имён*.

Пример #1 Примеры разрешения имён

```php
<?php

namespace A;
use B\D, C\E as F;

// вызовы функций

foo(); // Сперва пытается вызвать функцию foo, определённую в пространстве имён A,
// Затем вызывает глобальную функцию foo

\foo(); // Вызывает функцию foo, определённую в глобальном пространстве
```

```php
my\foo(); // Вызывает функцию foo, определённую в пространстве имён A\my

F(); // Сперва пытается вызвать функцию F, определённую в пространстве имён A,
// Затем вызывает глобальную функцию F

// Ссылки на классы

new B(); // Создаёт объект класса B, определённого в пространстве имён A.
// Если класс не найден, то пытается сделать автозагрузку класса A\B

new D(); // Используя правила импорта, создаёт объект класса D, определённого в пространстве имён B,
// если класс не найден, то пытается сделать автозагрузку класса B\D

new F(); // Используя правила импорта, создаёт объект класса E, определённого в пространстве имён C,
// если класс не найден, то пытается сделать автозагрузку класса C\E

new \B(); // Создаёт объект класса B, определённого в глобальном пространстве,
// если класс не найден, то пытается сделать автозагрузку класса B

new \D(); // Создаёт объект класса D, определённого в глобальном пространстве,
// если класс не найден, то пытается сделать автозагрузку класса D

new \F(); // Создаёт объект класса F, определённого в глобальном пространстве,
// если класс не найден, то пытается сделать автозагрузку класса F

// Статические методы и функции пространства имён из другого пространства имён

B\foo(); // Вызывает функцию foo из пространства имён A\B

B::foo(); // Вызывает метод foo из класса B, определённого в пространстве имён A,
// если класс A\B не найден, то пытается сделать автозагрузку класса A\B

D::foo(); // Используя правила импорта, вызывает метод foo класса D, определённого в пространстве имён B,
// если класс B\D не найден, то пытается сделать автозагрузку класса B\D

\B\foo(); // Вызывает функцию foo из пространства имён B

\B::foo(); // Вызывает метод foo класса B из глобального пространства,
// если класс B не найден, то пытается сделать автозагрузку класса B

// Статические методы и функции пространства имён из текущего пространства имён

A\B::foo(); // Вызывает метод foo класса B из пространства имён A\A,
// если класс A\A\B не найден, то пытается сделать автозагрузку класса A\A\B

\A\B::foo(); // Вызывает метод foo класса B из пространства имён A,
// если класс A\B не найден, то пытается сделать автозагрузку класса A\B

?>
```


User Contributed Notes 9 notes

37
kdimi ¶
13 years ago
If you like to declare an __autoload function within a namespace or class, use the spl_autoload_register() function to register it and it will work fine.

down
33
rangel ¶
14 years ago
The term "autoload" mentioned here shall not be confused with __autoload function to autoload objects. Regarding the __autoload and namespaces' resolution I'd like to share the following experience:

->Say you have the following directory structure:

- root
| - loader.php
| - ns
| - foo.php

->foo.php

```
<?php
namespace ns;
class foo
{
public $say;

public function __construct()
{
$this->say = "bar";
}

}
?>
```

-> loader.php

```
<?php
//GLOBAL SPACE <--
function __autoload($c)
{
require_once $c . ".php";
}

class foo extends ns\foo // ns\foo is loaded here
{
public function __construct()
{
parent::__construct();
echo "<br />foo" . $this->say;
}
}
$a = new ns\foo(); // ns\foo also loads ns/foo.php just fine here.
echo $a->say; // prints bar as expected.
$b = new foo; // prints foobar just fine.
?>
```

If you keep your directory/file matching namespace/class consistence the object __autoload works fine.
But... if you try to give loader.php a namespace you'll obviously get fatal errors.
My sample is just 1 level dir, but I've tested with a very complex and deeper structure. Hope anybody finds this useful.

Cheers!
up
down
4
safakozpinar at NOSPAM dot gmail dot com ¶
13 years ago

As working with namespaces and using (custom or basic) autoload structure; magic function __autoload must be defined in global scope, not in a namespace, also not in another function or method.

```
<?php
namespace Glue {
/**
* Define your custom structure and algorithms
* for autoloading in this class.
*/
class Import
{
public static function load ($classname)
{
echo 'Autoloading class '.$classname."\n";
require_once $classname.'.php';
}
}
}

/**
* Define function __autoload in global namespace.
*/
namespace {

function __autoload ($classname)
{
\Glue\Import::load($classname);
}

}
?>
```

up
down
1

Kavoir.com ¶

10 years ago

For point 4, "In example, if the namespace A\B\C is imported as C" should be "In example, if the class A\B\C is imported as C".

up
down
-3

llmll ¶

9 years ago

The mentioned filesystem analogy fails at an important point:

Namespace resolution *only* works at declaration time. The compiler fixates all namespace/class references as absolute paths, like creating absolute symlinks.

You can't expect relative symlinks, which should be evaluated during access -> during PHP runtime.

In other words, namespaces are evaluated like __CLASS__ or self:: at parse-time. What's *not* happening, is the pendant for late static binding like static:: which resolves to the current class at runtime.

So you can't do the following:

```
namespace Alpha;
class Helper {
public static $Value = "ALPHA";
}
class Base {
public static function Write() {
echo Helper::$Value;
```

```
}
}

namespace Beta;
class Helper extends \Alpha\Helper {
public static $Value = 'BETA';
}
class Base extends \Alpha\Base {}

\Beta\Base::Write(); // should write "BETA" as this is the executing namespace context at runtime.

If you copy the write() function into \Beta\Base it works as expected.
```

up

down

-6

rangel ¶

14 years ago

```
The term "autoload" mentioned here shall not be confused with __autoload function to autoload objects. Regarding the
__autoload and namespaces' resolution I'd like to share the following experience:

->Say you have the following directory structure:

- root
| - loader.php
| - ns
| - foo.php

->foo.php

<?php
namespace ns;
class foo
{
public $say;

public function __construct()
{
$this->say = "bar";
}

}
?>

-> loader.php

<?php
//GLOBAL SPACE <--
function __autoload($c)
{
require_once $c . ".php";
}

class foo extends ns\foo // ns\foo is loaded here
{
public function __construct()
{
parent::__construct();
echo "<br />foo" . $this->say;
}
}
$a = new ns\foo(); // ns\foo also loads ns/foo.php just fine here.
echo $a->say; // prints bar as expected.
```

```
$b = new foo; // prints foobar just fine.
?>
```

If you keep your directory/file matching namespace/class consistence the object __autoload works fine.
But... if you try to give loader.php a namespace you'll obviously get fatal errors.
My sample is just 1 level dir, but I've tested with a very complex and deeper structure. Hope anybody finds this useful.

Cheers!

up
down
-6

CJ Taylor ¶
9 years ago

It took me playing with it a bit as I had a hard time finding documentation on when a class name matches a namespace, if that's even legal and what behavior to expect. It IS explained in #6 but I thought I'd share this with other souls like me that see it better by example. Assume all 3 files below are in the same directory.

file1.php
```
<?php
namespace foo;

class foo {
static function hello() {
echo "hello world!";
}
}
?>
```

file2.php
```
<?php
namespace foo;
include('file1.php');

foo::hello(); //you're in the same namespace, or scope.
\foo\foo::hello(); //called on a global scope.
?>
```

file3.php
```
<?php
include('file1.php');

foo\foo::hello(); //you're outside of the namespace
\foo\foo::hello(); //called on a global scope.
?>
```

Depending upon what you're building (example: a module, plugin, or package on a larger application), sometimes declaring a class that matches a namespace makes sense or may even be required. Just be aware that if you try to reference any class that shares the same namespace, omit the namespace unless you do it globally like the examples above.

I hope this is useful, particularly for those that are trying to wrap your head around this 5.3 feature.

up
down
-4

anrdaemon at freemail dot ru ¶
8 years ago

Namespaces may be case-insensitive, but autoloaders most often do.
Do yourself a service, keep your cases consistent with file names, and don't overcomplicate autoloaders beyond necessity.
Something like this should suffice for most times:

```
<?php

namespace org\example;
```

```
function spl_autoload($className)
{
$file = new \SplFileInfo(__DIR__ . substr(strtr("$className.php", '\\', '/'), 11));
$path = $file->getRealPath();
if(empty($path))
{
return false;
}
else
{
return include_once $path;
}
}

\spl_autoload_register('\org\example\spl_autoload');
?>
```

up
down
-7

dn dot permyakov at gmail dot com ¶
9 years ago

Can someone explain to me - why do we need p.4 if we have p.2 (which covers both unqualified and qualified names)?

＋add a note