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

Конструкторы и деструкторы »
« Константы классов

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Автоматическая загрузка классов

Большинство разработчиков объектно-ориентированных приложений используют такое соглашение именования файлов, в котором каждый класс хранится в отдельно созданном для него файле. Одна из самых больших неприятностей - необходимость писать в начале каждого скрипта длинный список подгружаемых файлов (по одному для каждого класса).

Функция spl_autoload_register() позволяет зарегистрировать необходимое количество автозагрузчиков для автоматической загрузки классов и интерфейсов, если они в настоящее время не определены. Регистрируя автозагрузчики, PHP получает последний шанс для интерпретатора загрузить класс прежде, чем он закончит выполнение скрипта с ошибкой.

Любая конструкция, подобная классу может быть автоматически загружена таким же образом, включая классы, интерфейсы, трейты и перечисления.

Предостережение

До PHP 8.0.0 можно было использовать __autoload() для автозагрузки классов и интерфейсов. Однако это менее гибкая альтернатива spl_autoload_register(), функция __autoload() объявлена устаревшей в PHP 7.2.0 и удалена в PHP 8.0.0.

> **Замечание**:
>
> Функция spl_autoload_register() может быть вызвана несколько раз, чтобы зарегистрировать несколько автозагрузчиков. Выброс исключения из функции автозагрузки, однако, прервёт этот процесс и не позволит запускать дальнейшие функции автозагрузки. По этой причине выбрасывать исключения из функции автозагрузки настоятельно не рекомендуется.

Пример #1 Пример автоматической загрузки

В этом примере функция пытается загрузить классы `MyClass1` и `MyClass2` из файлов *MyClass1.php* и *MyClass2.php* соответственно.

```php
<?php
spl_autoload_register(function ($class_name) {
include $class_name . '.php';
});

$obj = new MyClass1();
$obj2 = new MyClass2();
?>
```

Пример #2 Ещё один пример автоматической загрузки

В этом примере представлена попытка загрузки интерфейса `ITest`.

```php
<?php

spl_autoload_register(function ($name) {
var_dump($name);
});

class Foo implements ITest {
}

/*
string(5) "ITest"

Fatal error: Interface 'ITest' not found in ...
*/
?>
```

Смотрите также

- unserialize()
- unserialize_callback_func
- unserialize_max_depth
- spl_autoload_register()
- spl_autoload()
- __autoload()

+add a note

User Contributed Notes 6 notes

up
down
103
jarret dot minkler at gmail dot com ¶
14 years ago

```
You should not have to use require_once inside the autoloader, as if the class is not found it wouldn't be trying to look
for it by using the autoloader.

Just use require(), which will be better on performance as well as it does not have to check if it is unique.
```

up
down
57
str at maphpia dot com ¶
7 years ago

```
This is my autoloader for my PSR-4 clases. I prefer to use composer's autoloader, but this works for legacy projects that
can't use composer.

<?php
/**
* Simple autoloader, so we don't need Composer just for this.
*/
class Autoloader
{
public static function register()
{
spl_autoload_register(function ($class) {
$file = str_replace('\\', DIRECTORY_SEPARATOR, $class).'.php';
if (file_exists($file)) {
require $file;
return true;
}
return false;
});
}
}
Autoloader::register();
```

up
down
29
toi]n[enkayt[attaat]gmaal.com ¶
3 years ago

```
Autoloading plain functions is not supported by PHP at the time of writing. There is however a simple way to trick the
autoloader to do this. The only thing that is needed is that the autoloader finds the searched class (or any other
autoloadable piece of code) from the files it goes through and the whole file will be included to the runtime.

Let's say you have a namespaced file for functions you wish to autoload. Simply adding a class of the same name to that
file with a single constant property is enough to trigger the autoloader to seek for the file. Autoloading can then be
triggered by accessing the constant property.

The constant could be replaced by any static property or method or by default constructor. However, I personally find a
constant named 'load' elegant and informative. After all this is a workaround. Another thing to keep in mind is that this
```

introduces an unnecessary class to the runtime. The benefit of this is that there is no need to manually include or require files containing functions by path which in turn makes code maintaining easier. Such behaviour makes it easier to alter the project structure since manual includes need not to be fixed. Only the autoloader needs to be able to locate the moved files which can be automated.

A code file containing functions.
/Some/Namespace/Functions.php

```
<?php
namespace Some\Namespace;

class Functions { const load = 1; }

function a () {
}

function b () {
}
?>
```

Triggering autoloading of the file containing functions.
main.php

```
<?php
\Some\Namespace\Functions::load;

a ();
b ();
?>
```

up
down
19
Anonymous ¶
14 years ago

It's worth to mention, if your operating system is case-sensitive you need to name your file with same case as in source code eg. MyClass.php instead of myclass.php

up
down
1
kalkamar at web dot de ¶
15 years ago

Because static classes have no constructor I use this to initialize such classes.
The function init will (if available) be called when you first use the class.
The class must not be included before, otherwise the init-function wont be called as autoloading is not used.

```
<?php
function __autoload($class_name)
{
require_once(CLASSES_PATH.$class_name.'.cls.php');
if(method_exists($class_name,'init'))
call_user_func(array($class_name,'init'));
return true;
}
?>
```

I use it for example to establish the mysql-connection on demand.

It is also possilbe do add a destructor by adding this lines to the function:

```
<?php
if(method_exists($class_name,'destruct'))
register_shutdown_function(array($class_name,'destruct'));
?>
```

up
down

-3
info at atomosmaestoso dot com ¶
1 year ago
```
Autoloading Classes with spl_autoload_register() or spl_autoload() is the best and most modern way to securely code for
API integration.

It restricts the various attacks that can be faced by using a "polyfill" or framework that is subject to data injection.
Low level attacks, polyfill and framework vulnerabilities are some exploitations limited in using the core functionalities
of your host programming language.

Your loop-holes and target endpoints are vastly removed to the level of programming experience of the developer - in not
exposing the threats espoused to your programming language and its security protocols.

Each event you transfer data from one program to the next reveals another threat and another attack endpoint. When you are
production, it is at this point composer and other tools that gather requirements specific secure integration should limit
its use, such as PCI-DSS, HIPAA, or GDPR.

The use of a framework or polyfill gives an attacker hints at what point a function will access memory to produce intended
results. Visiting the late L1-Cache Terminal Fault - attacks that use machine language to access memory and read what
actually is happening will have all the details of what process is taking place and when.

Not to mention, when a product is open-source, the code is editable and easily compiled. Using access to machine level
integrations a simply 10 second loss of time to process could well as infer the entire application has experienced an
overhaul.

To deter this, and ensure maximum security for piece of mind and money-wise. The embedded resources of a programming
language should be utilized at maximal capacity to prevent an overhaul on multiple endpoints. Visiting a system in use is
not deletable or easily moved, removed or altered.
```
+ add a note