


Global or Singleton for database connection?

Asked 16 years, 3 months ago Modified 8 months ago Viewed 58k times  Part of [PHP](#) Collective



What is the benefit of using singleton instead of global for database connections in PHP? I feel using singleton instead of global makes the code unnecessarily complex.

81



Code with Global



```
$conn = new PDO(...);

function getSomething()
{
    global $conn;
    .
    .
    .
}
```

Code with Singleton

```
class DB_Instance
{
    private static $db;

    public static function getDBO()
    {
        if (!self::$db)
            self::$db = new PDO(...);

        return self::$db;
    }
}

function getSomething()
{
    $conn = DB_Instance::getDBO();
    .
    .
    .
}
```

If there's a better way of initializing database connection other than global or singleton, please mention it and describe the advantages it have over global or singleton.

Share

edited Sep 25, 2008 at 1:13

asked Sep 25, 2008 at 0:56

Improve this question

Follow



Imran

90.8k ● 26 ● 100 ● 131

If you are planning to use PDO at custom session handlers you should be aware of some peculiarities: stackoverflow.com/questions/2595860/pdo-prepare-silently-fails – Wabbitseason
May 25, 2010 at 13:20

9 Answers

Sorted by: Highest score (default)



105



I know this is old, but Dr8k's answer was *almost* there.

When you are considering writing a piece of code, assume it's going to change. That doesn't mean that you're assuming the kinds of changes it will have hoisted upon it at some point in the future, but rather that some form of change will be made.

Make it a goal mitigate the pain of making changes in the future: a global is dangerous because it's hard to manage in a single spot. What if I want to make that database connection context aware in the future? What if I want it to close and reopen itself every 5th time it was used. What if I decide that in the interest of scaling my app I want to use a pool of 10 connections? Or a configurable number of connections?

A **singleton factory** gives you that flexibility. I set it up with very little extra complexity and gain more than just access to the same connection; I gain the ability to change how that connection is passed to me later on in a simple manner.

Note that I say *singleton factory* as opposed to simply *singleton*. There's precious little difference between a singleton and a global, true. And because of that, there's no reason to have a singleton connection: why would you spend the time setting that up when you could create a regular global instead?

What a factory gets you is a way to get connections, and a separate spot to decide what connections (or connection) you're going to get.

Example

```
class ConnectionFactory
{
    private static $factory;
    private $db;
```

```

public static function getFactory()
{
    if (!self::$factory)
        self::$factory = new ConnectionFactory(...);
    return self::$factory;
}

public function getConnection() {
    if (!$this->db)
        $this->db = new PDO(...);
    return $this->db;
}
}

function getSomething()
{
    $conn = ConnectionFactory::getFactory()->getConnection();
    .
    .
    .
}

```

Then, in 6 months when your app is super famous and getting dugg and slashdotted and you decide you need more than a single connection, all you have to do is implement some pooling in the getConnection() method. Or if you decide that you want a wrapper that implements SQL logging, you can pass a PDO subclass. Or if you decide you want a new connection on every invocation, you can do that. It's flexible, instead of rigid.

16 lines of code, including braces, which will save you hours and hours and hours of refactoring to something eerily similar down the line.

Note that I don't consider this "Feature Creep" because I'm not doing any feature implementation in the first go round. It's border line "Future Creep", but at some point, the idea that "coding for tomorrow today" is *always* a bad thing doesn't jive for me.

Share

Improve this answer

Follow

edited Sep 10, 2014 at 21:56



samayo

16.5k ● 13 ● 93 ● 113

answered Oct 20, 2008 at 19:37



Jon Raphaelson

1,244 ● 1 ● 9 ● 10

-
- 3 Not sure, but I think you meant: public function getConnection() { if (!\$this->db) \$this->db = new PDO(...); return \$this->db; } – [Dycey](#) Aug 3, 2010 at 13:51

Thanks! Would I lose any of the benefits of using this method by using `return self::$factory->getConnection();` instead of `return self::$factory;` ?

– [Nico Burns](#) Oct 22, 2010 at 17:29

-
- 3 I'd like to use this code in a project I'm doing. Can I cite this page, and if so, what license is this text under? Is it CC-BY, BSD or something else? I've currently claimed this as "Unknown - believe Public Domain" but I'd like to attribute the correct license conditions on it.

– [JonTheNiceGuy](#) May 27, 2011 at 14:44

- 2 I think we should make "\$db" "\$this->db" in the getConnection() method, otherwise "private \$db" variable "is not used", that is not officially referenced anywhere. – [developer10](#) Mar 2, 2014 at 18:48

Hi your solution looks cool and scalable. Please let me know what needs to be implemented here self::\$factory = new ConnectionFactory(...); – [Ananda](#) Jun 13, 2015 at 7:53 ✎



16



I'm not sure I can answer your specific question, but wanted to suggest that global / singleton connection objects may not be the best idea if this if for a web-based system. DBMSs are generally designed to manage large numbers of unique connections in an efficient manner. If you are using a global connection object, then you are doing a couple of things:

1. Forcing you pages to do all database connections sequentially and killing any attempts at asynchronous page loads.
2. Potentially holding open locks on database elements longer than necessary, slowing down overall database performance.
3. Maxing out the total number of simultaneous connections your database can support and blocking new users from accessing the resources.

I am sure there are other potential consequences as well. Remember, this method will attempt to sustain a database connection for every user accessing the site. If you only have one or two users, not a problem. If this is a public website and you want traffic then scalability will become an issue.

[EDIT]

In larger scaled situations, creating new connections everytime you hit the datase can be bad. However, the answer is not to create a global connection and reuse it for everything. The answer is connection pooling.

With connection pooling, a number of distinct connections are maintained. When a connection is required by the application the first available connection from the pool is retrieved and then returned to the pool once its job is done. If a connection is requested and none are available one of two things will happen: a) if the maximum number of allowed connection is not reached, a new connection is opened, or b) the application is forced to wait for a connection to become available.

Note: In .Net languages, connection pooling is handled by the ADO.Net objects by default (the connection string sets all the required information).

Thanks to Crad for commenting on this.



I guess using singleton gives me the advantage of initializing the connection as late as possible, while if I use global, I'd probably initialize nearly at the beginning of the script. Am I correct? – [Imran](#) Sep 25, 2008 at 1:11

Correct, if you were to go down this road the singleton would provide that advantage. – [Dr8k](#) Sep 25, 2008 at 1:25

Actually, that all depends upon scale. In large scale web deployments, massive amounts of DB connections are evil. This is why apps like pgBouncer exist for PostgreSQL and Java does resource pooling. – [Gavin M. Roy](#) Sep 25, 2008 at 2:13

True, but I think proper connection pooling is different to just using global connection objects. Connection pooling still uses multiple connections, it just limits the maximum number and reuses them over time to alleviate setup overhead. – [Dr8k](#) Sep 25, 2008 at 2:48

6 Note that "global" in the case of PHP doesn't make the variable global across PHP pages. It just means that it can be accessed from within functions. – [Ates Goral](#) Oct 20, 2008 at 19:59



7

The singleton method was created to make sure there was only one instance of any class. But, because people use it as a way to shortcut globalizing, it becomes known as lazy and/or bad programming.



Therefore, I would ignore global and Singleton since both are not really OOP.



What you were looking for is **dependency injection**.



You can check on easy to read PHP based information related to dependency injection (with examples) at

https://symfony.com/doc/current/components/dependency_injection.html

Share

Improve this answer

Follow

edited Apr 9 at 2:50



Tyler

349 ● 4 ● 11

answered Dec 13, 2011 at 0:35



user1093284



3

Both patterns achieve the same net effect, providing one single access point for your database calls.



In terms of specific implementation, the singleton has a small advantage of not initiating a database connection until at least one of your other methods requests it. In practice in most applications I've written, this doesn't make much of a difference, but it's a potential advantage if you have some pages/execution paths which don't make any database calls at all, since those pages won't ever request a connection to the database.



One other minor difference is that the global implementation may trample over other variable names in the application unintentionally. It's unlikely that you'll ever accidentally declare another global \$db reference, though it's possible that you could overwrite it accidentally (say, you write if(\$db = null) when you meant to write if(\$db == null). The singleton object prevents that.

Share Improve this answer Follow

answered Sep 25, 2008 at 2:44



Adam Ness

6,273 ● 4 ● 29 ● 39



If you're not going to use a persistent connection, and there are cases for not doing that, I find a singleton to be conceptually more palatable than a global in OO design.

2



In a true OO architecture, a singleton is more effective than creating a new instance the object each time.



Share Improve this answer Follow

answered Sep 25, 2008 at 1:03



Gavin M. Roy

4,641 ● 4 ● 35 ● 29



On the given example, I see no reason to use singletons. As a rule of thumb if my only concern is to allow a single instance of an object, if the language allows it, I prefer to use globals

2



Share Improve this answer Follow

answered Sep 25, 2008 at 1:06



Dprado

1,650 ● 1 ● 13 ● 13



In general I would use a singleton for a database connection... You don't want to create a new connection everytime you need to interact to the database... This might hurt performance and bandwidth of your network... Why create a new one, when there's one available... Just my 2 cents...

1



RWendi



Share Improve this answer Follow

answered Sep 25, 2008 at 1:14



RWendi

1,422 ● 5 ● 21 ● 38

With initializing the connection in global scope, I'm initializing the connection once per page, and using that global variable in functions that need to interact with database. – [Imran](#) Sep 25, 2008 at 1:18

Using a singleton for a database connection is not the same as not re-creating the connection upon each interaction with the DBMS. A singleton is just that; one instance of a given class, and the only one allowed to exist globally. You might need to connect to different databases at once. – [Rob](#) Sep 25, 2008 at 1:28

i was refering to a singleton class that manages connections to the database. I dont see the point of creating a new connection object everytime you want to interact to a dbms. Of course if you need to connect to different database at once, you may need to create another connection object. – [RWendi](#) Sep 25, 2008 at 1:52



It is quite simple. Never use global OR Singleton.

0

[Share](#) [Improve this answer](#) [Follow](#)

answered Sep 25, 2008 at 0:59



[1800 INFORMATION](#)

135k ● 30 ● 163 ● 242



-
- 4 When it comes to Singleton pattern, always say never – [1800 INFORMATION](#) Sep 25, 2008 at 1:02
-
- 3 What if you want more than one log provider? Who says I can't log to a file, and the console? – [1800 INFORMATION](#) Sep 25, 2008 at 1:03
-
- 2 So you would then combine one anti-pattern (Singleton) with another (God Object) – [1800 INFORMATION](#) Sep 25, 2008 at 1:52
-
- 3 Yes. And so do the designers of every major OOP language, by permitting global class objects. If you're a monotheist and you want an object to represent God, then what you're looking for is a Singleton God Object. Anything else is incorrect. – [Steve Jessop](#) Sep 25, 2008 at 1:57
-
- 4 If I was a montheist and I wanted to create a God object, I might specify to use a MonotheistAbstractFactory Pattern to create my God Object? This would leave open the possiblility for polytheistic users to also use my program by specifying a PolytheistAbstractfactory. – [1800 INFORMATION](#) Sep 25, 2008 at 2:36
-



As advice both **singleton** and **global** are valid and can be joined within the same *system, project, plugin, product, etc ...* In my case, I make digital products for web (plugin).

0



I use only **singleton** in the main class and I use it by principle. I almost do not use it because I know that the main class will not instantiate it again



```
<?php // file0.php

final class Main_Class
{
    private static $instance;
    private $time;

    private final function __construct()
    {
        $this->time = 0;
    }
    public final static function getInstance() : self
    {
        if (self::$instance instanceof self) {
            return self::$instance;
        }

        return self::$instance = new self();
    }
    public final function __clone()
    {
        throw new LogicException("Cloning timer is prohibited");
    }
    public final function __sleep()
    {
        throw new LogicException("Serializing timer is prohibited");
    }
    public final function __wakeup()
    {
        throw new LogicException("UnSerializing timer is prohibited");
    }
}
```

Global use for almost all the secondary classes, example:

```
<?php // file1.php
global $YUZO;
$YUZO = new YUZO; // YUZO is name class
```

while at runtime I can use **Global** to call their methods and attributes in the same instance because I do not need another instance of my main product class.

```
<?php // file2.php
global $YUZO;
$YUZO->method1()->run();
$YUZO->method2( 'parameter' )->html()->print();
```

I get with the global is to use the same instance to be able to make the product work because I do not need a factory for instances of the same class, usually the instance factory is for large systems or for very rare purposes.

In conclusion: , you must if you already understand well that is the anti-pattern **Singleton** and understand the **Global**, you can use one of the 2 options or mix them but if I recommend not to abuse since there are many programmers who are very exception and faithful to the programming OOP, use it for main and secondary classes that you use a lot within the execution time. (It saves you a lot of CPU). 😊

Share

edited May 27, 2018 at 20:02

answered May 27, 2018 at 18:36

Improve this answer

Follow



LENIN

1,454 ● 1 ● 11 ● 15