Is there a best practice and recommended alternative to Session variables in MVC

Asked 10 years, 7 months ago Modified 7 years, 6 months ago

Viewed 35k times

56

Okay, so first off before anyone attempts to make a determination that this is a "duplicate" question; I have reviewed most of the posts on SO regarding similar questions but even in combination of all that has been said I still am somewhat at a dilemma as to the definitive or maybe I should say unanimous agreement on this.

I can however say that I have (based on the posts) conclusively determined that the answer is based on the scope of the requirement. But even with consideration of this, the opinions seem too diverse for me to make a decision as to how I should handle this.

My immediate requirement is that I need to persist variable data from 1 controller across many views. More specifically, I have a controller and corresponding view that handles shopping cart item counts and I would like to persist that data across multiple views. I am thinking that the _layout view is the most logical choice for this.

Now I have successfully accomplished this task by assigning the value to a Session variable which is

retrieved from my _layout view; so even when the user were to navigate any where within the site the number of items in the Shopping Cart will persist until either they leave the site or complete the checkout; in which case the variable will be cleared in code.

The posts I've read seemed biased to either staying away from Session variables in favor of Cookies and storing data in a database; or stating that for the intent purpose for which I propose to use them, Session variables are perfectly fine to use.

The other thing I've read suggests that Session variables can potentially impede overall performance if there is high traffic on the site since the information is stored on the server.

I personally cannot justify storing this type of information in a database and subsequently hitting the database as I'd imagine that this could also affect site performance and seems a bit overkill for storage of temporary data. TempData, ViewData and ViewBag do not work in persisting the data so they are not logical choices for the requirement IMO.

If there is another well suited alternative to the Session variable (which is working for me) I would like to know what it is.

2 posts that seem contradictory in effort of providing best recommendations leave me a bit confused.

Cons: Is it a good practice to avoid using Session State in ASP.NET MVC? If yes, why and how?

Pros: Still ok to use Session variables in ASP.NET mvc, or is there a better alternative for some things (like a cart)

Seems that this question (although presented in many different variations) has no definitive answer that I can conclude.

If there is a more preferrable way to accomplish this without overkill then that is the answer I'm in search of.

I read somewhere the use of MVC filters in tandem with the Global.ascx application start section as well, but this does not seem appropriate for variables set at the controller level as much as perhaps, static variables.

Can someone maybe squash (for lack of a better word) the many diverse opinions on the topic and maybe provide a more definitive answer to the question? I'm sure the diverse opinions have their place and I'm not attempting to discredit them. But having a definitive and possibly unanimous answer would be better; then I could sort through the other posts to determine what is best for my application.

Of course, if this question has no definitive answer; just tell me that and I'll attempt to derive my own answer from the other posts.

Thanks

==========
==========

UPDATED RESPONSE TO ANSWERS PROVIDED

Caching and Cookies seem to be a general preference from the responses however I've also noted the statement that caching its not an ideal candidate to use across multiple web server because synchronization can be a potential issue.

Giving credit to Tim, it's stated that Database storage is optimized and users have the option to return at a later time and continue where they left off.

That is an excellent point, but keeping foresight on probabilities; its likely a reasonable given that some users may not return leaving unneccessary data in the database.

So keeping the DB optimized and clean (which "to me" is of equal relevance) would require implementing a maintenance task to automatically expire those records based on a set threshold of time to account for those circumstances. Although a maintenance task is not an unquestionable option, I still think this adds just a bit more work to the task simply for the intent purpose of serving as temporary storage.

Nonetheless, I do respect Tim's recommendation and believe it deserves merit on countering my initial opinion to a degree; that a database would not seem to be a viable option for storing Temporary data; so I think the compromise would be to store the data in a database (given the scenario of a Shopping Cart or similar) perhaps after a checkout. This way as you previously stated, the data may be persistently tracked upon subsequent visits so you have a record of transactions. But more importantly, it would be data of those transactions having real relevance to persist to the database.

It was also stated that although Session is faster than Database; but notwithstanding to have its caveats that can to some degree be mitigated by other mechanisms such as leveraging the SessionStateBehavior attribute, just serving as one example.

BUT... I think Erik kind of drove the point home with the Dunning-Kruger Effect. Although, from the content and explanations for proposed answers given here; I seriously doubt the expertise of any of the individuals who have responded is any way questionable. Nonetheless, I tend to agree on the fact of getting a unanimous opinion may be somewhat of a higher than reasonable expectation on my part.

What I was more specifically looking for was a general consensus for a technique that would comfortably accomodate a diverse number of scenarios. In other

words, something that would accomodate not only my particular scenario but also provide the element of scalability to larger environments with potentially heavier traffic. This way a change in the programming would be either alleviated altogether or minimal at best.

==========

Summary based on the feedback:

1. Session variables seem to accomodate smaller case scenarios and when applicable, but they have some potential for persistence concerns among other notable discrepancies as stated very thorougly by Erik. So this option obviously will not fit a scalable model.
2. Caching is preferable over Session variables but again not neccessarily the "best" scalable option due to among other things to the potential synchronization complexities in web server farm environments as previously pointed out. But an option nonetheless.
3. Database storage is scalable but for the intent purpose of temporary volatile storage is probably not the most elegant option from a database perspective as it would require periodical cleanup. Personally,

having a strong foundation in database concepts earlier in my career this probably is not going to be something that many developers will likely agree with; but using the database for this purpose may suffice for Web Development from a programmers perspective; however from perspective of the DAL and DB development this (to me) has the potential for mandating an additional DB task to enforce an efficient backend.

4. Cookies seem to be a nice option having the combined "desirable" elements of Session variables and caching.

==========

CONCLUSION

Based on the answers; I think COOKIES and CACHING seem to be generally well rounded proposals for best practice across the board in combination with database storage when continued persistence is required after the fact; as potentially good candidates for scalability of the ones presented.

The ultimate choice between the 2 would seem to be based on the amount and type of data requiring storage (e.g. sensitive vs non-sensitive and whether or not there is any concern that the client may alter the data on their

end); in addition to special considerations for COOKIES in the fact that they may be disabled by the clients.

Obviously, there is no one size fits all solution as clearly pointed out and concluded from the answers provided but in terms of scalability; I may be wrong but these seem to be the BEST choices available.

Because all the responses are good; I'm fairly going to credit all the posts as useful and going to accept Erik's answer as a well rounded overall scalable solution. I wish I could select more than one accepted answer as I believe Tim's response was also very well layed out and concise.

Gupta's response was good also, but I wanted more elaboration of the proposed answer and not a repeat of previous posts.

Thanks Guys!

asp.net-mvc asp.net-mvc-3 asp.net-mvc-4 session session-variables

Share

Improve this question

Follow

edited May 23, 2017 at 11:54

Community Bot

1 • 1

asked May 2, 2014 at 0:39

Mark

1,687 • 2 • 25 • 54

2 I like this question, although it might be more suited to Programmers. – Rowan Freeman May 2, 2014 at 1:04

1 Thanks, this was really useful for me as well as I am currently umming and aaahing over whether to store a single value in a session,cookie or something else. In my case when a user logs into my site they are returned a non-identifiable value that needs persisting through a non-intensive site. – kolin Sep 4, 2015 at 13:47

1 Another possible solution, which is more feasible today than it was when this question was first asked, is HTTP 5 local storage. This is now much more widely supported (although you may still run into problem with some mobile browsers, and of course people still running very old desktop browsers). – Erik Funkenbusch Apr 6, 2016 at 15:07

3 Answers

Sorted by: Highest score (default)

28

You will never get unanimous opinion on anything in any large group of people. That's just human nature. Part of that stems from the Dunning-Kruger Effect which states that the less someone knows about a subject, the more likely they are to over value their expertise in that subject. In other words, lots of people think they know something, but only because they don't know they don't know it. Part of it is simply that people have different experiences, and some have found no problems with session, while others have in various situations, or vice versa...

So, to backup your research, which suggest that the answer depends heavily on the requirements, we need to

understand what your requirements are. If this is to be a high traffic site, with load balanced servers in a web farm, then stay as far away from session as you can. Sure, it's possible to share session in various ways in a server farm environment (session server, distribute cache server, etc..), but avoiding session will almost always be faster if you can help it.

If your site is a single server, and unlikely to ever grow beyond that. And your traffic patterns are relatively low, then session may be a useful option. However, you should always be aware that session is unreliable storage, and can disappear on you at any time. If the app pool is recycled, session is gone. If an uncaught exception bubbles up to the worker process, the session may be gone. If IIS thinks there's not enough memory, your session may be gone, regardless of any timeout values configured. You also can't always get reliable notification that a session has ended, since terminated sessions do not fire the Session_End event.

Another issue is that Session is serialized. In other words, IIS prevents more than one thread from writing to the session at a time, and it often does this by locking the session while a thread is running if it has not opted out of writable session locking. This can cause severe problems in some cases, and merely poor performance in others. You can mitigate this by marking various methods with a read-only session attribute if you aren't going to be modifying it in that method.

Ultimately, if you do choose to use session, then try to only use it for small, short lived things if at all possible, and if not possible then build in a way to "regenerate" the data if the session is lost. For instance, using your number of items in cart example, you could write a method that first checks to see if the value is there, and if not it goes out and loads it from the database. Always use this method to access the variable, rather than accessing it directly from session... this way, if the session is lost it will just reload it.

However, having said this... For the number of items in a cart, I would generally prefer to use a cookie for this information, since cookies get passed to the page on every load anyways, and this is a small discrete unit of data. Generally prefer Session for sensitive data that you want to prevent the user from being able to change.. number of items in the cart simply doesn't fit that rule.

Share Improve this answer Follow

edited May 2, 2014 at 2:45

answered May 2, 2014 at 2:32

Erik Funkenbusch

93.4k ● 29 ● 200 ● 292

When

11

Databases are highly optimized. A simple value like a shopping cart count is a good candidate for caching by



the database and (hopefully) cheap to compute outright. It may be a non-issue.

However, if you have ruled out other mechanisms, small, user-by-user values are viable candidates for session.

`Cache` is fine for site-wide values, or user-specific values with unique keys. However, synchronizing caches across multiple web servers can be difficult. Out of process session state will stay synchronized because it is stored in a single location (database or a state server).

Of course, there are many 3rd party caching alternatives with various options to keep them synchronized.

Regardless of where the count is temporarily stored, I'm of the opinion that shopping carts themselves should be stored in the database so that users have the option to return later and continue where they left off.

Performance

If you use out of process session state (e.g. in a load balanced environment and/or to make session more durable), it *will* hit a database or call an out of process service, but the call is relatively cheap unless you are serializing large object graphs.

Session is loaded once per request. Subsequent read access is very fast.

Writing to session can be detrimental to performance, even when there is no load. Why? most modern applications use asynchronous calls, and when multiple async calls hit an HTTP handler (page, controller, etc) that reads/writes session, ASP.Net will lock the session to serialize access. To avoid this, you can decorate your controllers with `[SessionState( SessionStateBehavior.ReadOnly )]`

Design

> Now I have successfully accomplished this task by assigning the value to a Session variable which is retrieved from my _layout view;

This seems like mixing concerns, i.e. having the view aware of the underlying storage mechanism. From a purist standpoint, I would set this value on a view model or at least put it in the `ViewBag`. From a practical standpoint, one or two values retrieved in this manner probably won't hurt anything, but beware of letting it grow much further.

> I read somewhere the use of MVC filters in tandem with the Global.ascx application start section as well, but this does not seem appropriate for variables set at the controller level as much as perhaps, static variables.

Static variables have perfectly legitimate uses, but you must understand them thoroughly or risk serious problems.

See my answers pertaining to static variables in ASP.Net:

- does aspx provide special treatment for c# static variables
- Static fields vs Session variables

Share Improve this answer Follow

edited May 23, 2017 at 12:18

Community Bot

1 ● 1

answered May 2, 2014 at 2:26

Tim M.

54.3k ● 14 ● 124 ● 166

1 Thank you Tim. Excellent information regarding the Static variable options which I will bookmark. I did also explore the decorative SessionState attribute earlier prior to my post but was not really fond of using it for this. – Mark May 2, 2014 at 11:49

1 Also, thanks for pointing out the potential mixing of concerns with my current design – Mark May 2, 2014 at 12:00

3

Session alternative in different prospective :-

When you keep something in session it breaks the primary rule in ASP.NET MVC. You can use these options as an alternative of session.

If your asp.net (MVC) session do boxing unboxing on the object then it makes a little load on the server. Try this idea

1. Caching :- Storing a List or something like large data in session is better can fit in Caching. You have control on whenever you want it to expire rather than user session.
2. If your app depends on JSON/Ajax data then you can use some kind of functionality provided in html5 (like WebSQL, IndexDB). it will not use the cookie so you can save some workload on the server.

Share Improve this answer Follow

edited Jun 11, 2017 at 10:14

answered May 2, 2014 at 2:01

Anirudha Gupta

9,279 ● 9 ● 56 ● 82

1 Good point regarding Session variables Gupta and your proposed options are notable as well. But I was specifically looking for some additional recommendation for best practice and why. This information is definitely useful though. Thanks! – Mark May 2, 2014 at 11:52