# Sessions in MVC

## Introduction

Session is used to store data values across requests. Whether you store some data values with in the session or not Asp.Net MVC must manage the session state for all the controllers in your application that is time consuming. Since, session is stored on server side and consumes server memory, hence it also affects your application performance.

## Session State

In ASP.NET, there are several types of session state modes that you can choose from, depending on your application's requirements. The session state mode determines where and how session data is stored. Here are the commonly used session state modes:

1. In-Process (InProc): In this mode, the session data is stored in the memory of the web server process. It provides fast access to session data but is limited to a single server. If the web server process restarts or the application pool is recycled, the session data is lost.
2. State Server (StateServer): In this mode, the session data is stored in a separate state server process, which can be located on a different machine. The session data is serialized and transferred between the web server and the state server. This mode allows sharing session data across multiple servers in a web farm scenario. However, session data must be serializable.
3. SQL Server (SQLServer): In this mode, the session data is stored in a SQL Server database. The session data is serialized and stored in a database table. This mode provides durability and scalability as session data is persisted even if the web server process restarts. It also allows sharing session data across multiple servers. However, it may introduce additional latency due to database access.
4. Custom: ASP.NET allows you to implement a custom session state provider by implementing the SessionStateStoreProviderBase class. This gives you the flexibility to store session data in a custom data store, such as a NoSQL database or a distributed cache.

To configure the session state mode, you need to modify the <sessionState> element in the web.config file. Here's an example of how to specify the session state mode:

<system.web>

<!-- Other configuration settings -->

<sessionState mode="InProc" timeout="20" />

</system.web>

public class HomeController : Controller

{

public ActionResult Index()

{

// Storing a value in session

Session["Username"] = "JohnDoe";

// Retrieving a value from session

string username = (string)Session["Username"];

// Checking if a session variable exists

if (Session["Username"] != null)

{

// Session variable exists

}

else

{

// Session variable does not exist

}

return View();

}

}

Or

public ActionResult Index()

{

// Storing a value in session

Session["Username"] = "JohnDoe";

// Retrieving a value from session

string username = (string)Session["Username"];

return View();

}

## Session Less Controller

If some of the controllers of your Asp.Net MVC application are not using session state features, you can disable session for those controllers and can gain slight performance improvement of your application. You can simplify session state for your application by using available options for session state.

In Asp.Net MVC, SessionState attribute provides you more control over the behavior of session-state by specifying the value of SessionStateBehavior enumeration as shown below:

|  |  |
| --- | --- |
| **Value** | **Description** |
| Default | The default Asp.Net behavior is used to determine the session state behavior. |
| Disabled | Session state is disabled entirely. |
| ReadOnly | Read-only session state behavior is enabled. |
| Required | Full read-write session state behavior is enabled. |



When a user visits a website, a unique session is created for that user. The session is identified by a session ID, which is typically stored in a cookie or appended to the URL. The session ID is used to associate subsequent requests from the same user with their specific session data.

To create a session and store a list in it, you can follow these steps in an ASP.NET application:

Start by creating a new session using the Session object. Typically, this is done in a controller action or any other appropriate location within your code.

public ActionResult MyAction()

{

// Create a new list

List<string> myList = new List<string>();

// Add items to the list

myList.Add("Item 1");

myList.Add("Item 2");

myList.Add("Item 3");

// Store the list in the session

Session["MyList"] = myList;

return View();

}

In the code above, a new list (myList) is created, and some sample items are added to it. Then, the list is stored in the session using the key "MyList".

Later, in another action or any other part of your application where you need to access the list from the session, you can retrieve it using the same key.

public ActionResult AnotherAction()

{

// Retrieve the list from the session

List<string> myList = Session["MyList"] as List<string>;

//List<string> myList = (List<string>)Session["MyList"];

// Do something with the list

if (myList != null)

{

// Access the items in the list

foreach (string item in myList)

{

// Process each item

// ...

}

}

return View();

}

## Update Session

To update a session in ASP.NET, you can simply assign a new value to the corresponding session variable. Here's an example of how you can update a session variable:

// Retrieve the existing value from the session

string oldValue = (string)Session["MyVariable"];

// Update the value

string newValue = "New Value";

Session["MyVariable"] = newValue;

## Delete Session

To delete a session in ASP.NET, you can use the Session.Clear() or Session.Abandon() method. Both methods have different effects on the session. Here's how you can delete a session using each method:

**Session.Clear():** The Session.Clear() method removes all keys and values from the session, effectively clearing its contents while keeping the session alive. Here's an example:

// Clear the session

Session.Clear();

**Session.Abandon():** The Session.Abandon() method terminates the current session and releases its resources. It removes all keys and values from the session, ends the session, and generates a new session identifier upon the next request. Here's an example:

// Abandon the session

Session.Abandon();

Overall, by using Session.Clear() or Session.Abandon(), you can delete a session in ASP.NET depending on your desired behavior and requirements.

Overall, by using Session.Clear() or Session.Abandon(), you can delete a session in ASP.NET depending on your desired behavior and requirements.

public **ActionResult** Logout()

{

// Abandon the session

Session.**Abandon**();

// Perform any additional logout logic

return RedirectToAction("Index", "Home");

}

## Is Session Exists

To check if a session is available or not in ASP.NET MVC 5, you can check the Session property for null or use the Session.IsNewSession property. Here are two common ways to check the availability of a session:

if (Session != null)

{

// Session is available

// Perform session-related operations

}

else

{

// Session is not available

// Handle the absence of session

}

To check if a session has values stored in it, you can check if the session variable is not null and if it contains a specific value or any value at all. Here's an example:

if (Session["MyVariable"] != null)

{

// Session variable is not null

// Check if the session variable has a specific value

if ((string)Session["MyVariable"] == "desiredValue")

{

// Session variable has the desired value

// Perform actions accordingly

}

else

{

// Session variable exists but has a different value

// Perform other actions

}

}

else

{

// Session variable is null or doesn't exist

// Handle the absence of session variable

}