Session Management Techniques in ASP.NET

Introduction

In this article we will try to see what are the different ways we can manage sessions in an ASP.NET application. When should we use which techniques and what are the pros and cons of using each technique.

Background

The basic stateless nature of web sites and web applications are rather very confusing to anyone who is starting with web development. ASP.NET provides various ways of handling the states. The basic overview of all the state management techniques can be found here.

In this article we will specifically be discussing the Session management techniques. Session are the server side method of managing the state of an application i.e. all the web applications’ state related info will be stored on server side if we use this technique. The benefit of having this technique is that since we are keeping all the state related information on server, the request and response becomes lightweight. Also, the chances of someone intercepting or changing this data is also reduced. But this technique does involve more resource usage on the server.

The advantages of using Session State are

- Better security
- Reduced bandwidth

The disadvantages of using Session state are

- More resource consumption of server.
- Extra code/care if a Web farm is used (we will discuss this shortly)

Let us start looking into how we can use the Session state in our application and see how we can configure it to use various techniques.

Getting to know the Players

Using Session State

ASP.NET allows us to save values using Session state. It is a global storage mechanism that is accessible from all pages in the Web application. Session state is stored in the Session key/value dictionary. This information will be user specific i.e. for each user
separate dictionary will be created and no one can access other session information. Below is the Example usage of sessions.

```csharp
//global.asax
void Session_Start(object sender, EventArgs e)
{
    // Code that runs when a new session is started
    Session["number"] = 0;
}

// Web forms
Session["number"] = Convert.ToInt32(Session["number"]) + 1;
Label6.Text = Session["number"].ToString();
```

Enumerating Session Management Techniques

Before we proceed, let us see what all session management techniques are present in the ASP.NET framework.

- In-Proc.
- SQLServer.
- StateServer.

How to configure Sessions

To configure the session management we need to specify the settings in the `web.config` file. Typical settings in `web.config` looks like:

```xml
<sessionState mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="Data Source=.\SQLEXPRESS;Trusted_Connection=Yes;"
    cookieless="false"
    timeout="100"/>
```

Let us see what each of these attributes mean.

- **mode**: This specifies the type of session management we want to use. It could be InProc, SQLServer, and StateServer.
- **stateConnectionString**: If we use StateServer as session management technique then this specifies the location of the server that is handling the session data.
- **sqlConnectionString**: If we use SQLServer as session management technique then this specifies the database `connectionstring` that will store the session data.
**cookieless**

This specifies whether we will be using cookies to identify sessions or we want session info appended in URL. It could be true or false.

**timeout**

This specifies the time for which the session should be active. After this much time of inactivity the session will expire.

---

**Single server vs Web Farm**

Website require a hosting server. If we have a high traffic website then to improve the performance we might choose to have multiple web servers. In case we want multiple web servers (Web Farm), We need to have all the requests handled at a single point i.e. a Load Balancer. The Load Balancer will accept all the requests and forward it to appropriate server (based on load on server and some other criteria)

**Single server**

![Single server setup with a Load Balancer](image)

*Setup with a Load Balancer*
Note: Why is this relevant will become clear in some time.

Configuring the Session Modes

Using In-Proc Session Mode

If we are using In-Proc session handling then all the session information will be stored in the server memory. This is the default session state in ASP.NET.

This approach works fine as long as we are keeping small session information. Since the memory location was handled by the ASP.NET worker thread only it involves a considerable overhead for the worker thread to manage these. Also, since this is in the memory of server, chances are that large session information would lead to more memory usage and thus lowering the performance.

Let us try to visualize this session management technique.
To configure the In-Proc session management we need to specify the sessionState element in the web.config file. Let us see how to configure the in-proc sessions.

```xml
<sessionState mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="Data Source=.\SQLEXPRESS;Trusted_Connection=Yes;"
    cookieless="false"
    timeout="100"/>
```

The only relevant options for In_proc session mode in this configuration elements are the mode (which specifies that we are using In_proc sessions), the cookieless (if true, will use to manage sessions otherwise append session management data in the URL) and timeout (the time after which the session should expire).

It is advisable to use the In-Proc sessions when we want very small amount of information in the sessions. Since the session data is stored in the memory of computer which is directly accessible by ASP.NET worker thread nothing extra needs to be done.

The problem with In-Proc mode session mode is that if the size of session data becomes large then it effects performance. Also, since the session data is in memory of server the application restart(ASP.NET) will effectively flush all the data.

**In-Proc sessions in Web Farm scenario**

As we have discussed above the web farm will have a load balancer accepting all the requests. It will then forward the request to any server based on some criteria. Let us try to see a scenario which could create problems for us if we are using In-Proc sessions.
Problem Scenario

1. The request from Computer1 reaches the load balancer and load balancer forwards it to server1.
2. The In-Proc session led the session data created in the server1’s memory.
3. The next request comes from Computer1 and this time load balancer pushes it to Server2.
4. Since the server1 had this user specific data in its memory server2 will not be able to recognize this request leading to failure of our application logic.

So what should we do to tackle this situation. The whole problem was because each server in our web form was keeping the session data in their own memory. If we could somehow move this data from each server memory to a centralized location then we will not face this problem. And that is where the SQLServer and stateServer comes to rescue. using these two approaches we can easily configure a central repository to store session data.

Note: There could still be some reasons(explained below) which will force you to use In-Proc sessions in a Web farm scenario. If this is the case then the load balancer should be configured to maintain connection affinity or sticky sessions so that the requests coming from a particular client will always be forwarded to the same server so that the session information can always be found.
Using SQLServer Session Mode

If we use the SqlServer mode of session management then the session data will be stored in the SQLServer. The benefit of having this scenario is that the data is stored in a centralized database rather than the server memory. Let us see how this can be configured from web.config

```xml
<sessionState mode="SQLServer"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="Data Source=.\SQLEXPRESS;Trusted_Connection=Yes;"
    cookieless="false"
    timeout="100"/>
```

Apart from this the SQLServer also needs to be configured to store the session data. To do that simply type aspnet_regsql on Visual studio command prompt and a configuration wizard will open. Here we can configure the SQLServer to cater the session data. The demo project has sqlexpress in local machine configured for this. To run the application, you will have to configure the sqlserver on local machine and perhaps change the web.config of the website.

**Note:** I have given cursory step by step guide to configure SqlServer because I want to keep this article at conceptual level rather than becoming a "How To" guide.

This way the server memory is not affected by the size of session data. Also, if we have a web farm then we can don’t find any problem in using session data as the session data is in a central location. Lets try to visualize this scenario
Using this mode will only involve the overhead of additional database access to retrieve the session data. But there is one important aspect to remember when we are planning to use this technique. We need to send and receive the data to a SqlServer database, which will be a separate server. To successfully send and receive the data to SqlServer all the data that is being kept in the Session has to be **serializable**. This sometimes becomes a very big issue when an application is being ported to web farm from a single server. That is where the load balancer's connection affinity will come in picture with In-Proc sessions.

**Using StateServer Session Mode**

If we use the **StateServer** mode of session management then the session data will be stored in a separate computer(server) and the session data will be handled by a windows service. The benefit of having this scenario is that the data is stored in a centralized location i.e. a state server rather than the individual server memory. Let us see how this can be configured from *web.config*

```xml
<sessionState mode="StateServer"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="Data Source=.\SQLEXPRESS;Trusted_Connection=Yes;"
    cookieless="false"
    timeout="100"/>
```
Apart from this the Server that will be used as state server also needs to be configured to store the session data. To do that simply run the service named ASP.NET state service to enable the desired Server to cater the session data. The demo project does the same on local machine configured for this. To run the application, you will have to run the service on local machine and perhaps change the web.config of the website.

Note: I have given cursory step by step guide to configure state server on local machine because I want to keep this article at conceptual level rather than becoming a "How To" guide.

This way the server memory is not affected by the size of session data. Also, if we have a web farm then we can don’t find any problem in using session data as the session data is in a central location. Let’s try to visualize this scenario

Using this mode will only involve the overhead of network latency to retrieve the session data. But there is one important aspect to remember when we are planning to use this technique. We need to send and receive the data to a state server. To successfully send and receive the data to the state server all the data that is being kept in the Session has to be serializable. This sometimes becomes a very big issue when an application is being ported to web farm from a single server. That is where the load balancer’s connection affinity will come in picture with In-Proc sessions.
Using the code

The test application contains 3 web sites with single page each. functionality wise all the web sites are doing the same thing but each of then is using a different session management technique.

To run all three websites the Sqlserver on local machine has to be configured to store session specific data(details above in the respective category) and the ASP.NET state service should be running on local machine. Also, the connectionstring will have to be modified accordingly to use the database on the running computer.

**Note:** Please refer the web.config of each website to get the subtle difference.

The application while running will look like:

![Application Interface]

**Points of Interest**

Sometime back I had a task of migrating a website running on a single server to a web farm. Since I was not aware of all the information I provided in this article it was a nightmare for me. I actually changed the in-proc mode to StateServer mode and the application started complaining about the data getting pushed into session not being \textit{serializable}.

I then took a deep breath and read everything about sessions. I now have some understanding on how to configure the sessions. I have tried to convey the same information through this article. And as for the task I had, I ended up using in-proc sessions and configured sticky sessions on load balancer. This is a temporary thing till I get a chance to make everything \textit{serializable} in this application.