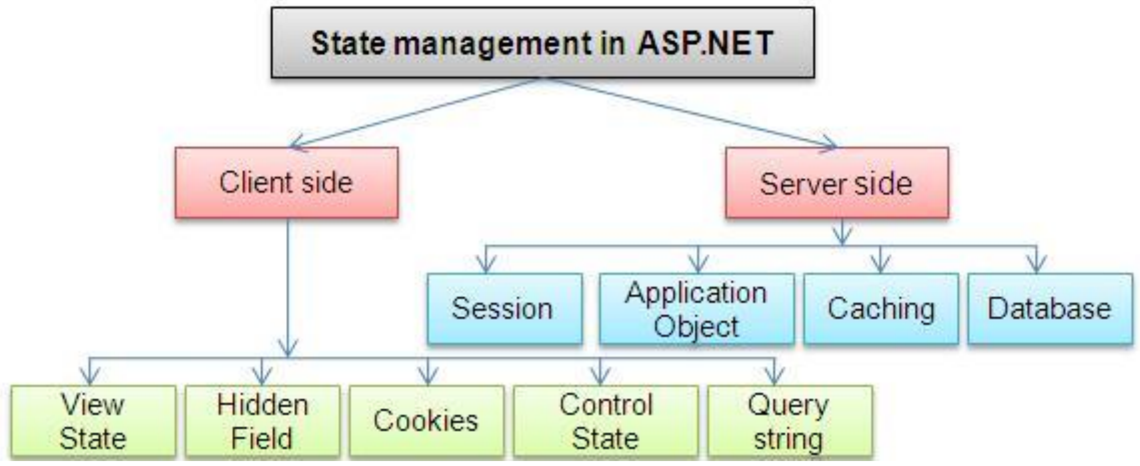
As we all know, web is stateless. A Web page is recreated every time it is posted back to the server. In traditional web programming, all the information within the page and control gets wiped off on every postback.

To overcome this problem, ASP.NET Framework provides various ways to preserve the states at various stages like controlstate, viewstate, cookies, session, etc. These can be defined in client side and server side state management.



Why state Management is important?

Web Pages developed in ASP.Net are HTTP based and HTTP protocol is a stateless protocol so web page is a stateless, which means that it is not capable of storing any information by itself. Because whenever a request for a web page comes from the client, by post-back of a web page the same page will not be returned to the client after processing. So all information associated with the page and the controls on the page would be lost with each round trip. Web server does not have any idea about the requests from where they coming means from same client or new clients. On each request web pages are created and destroyed.

Example of a stateless webpage is that a user is filling its online bank registration form and by mistake it fills some wrong information and after click on submit button its returned back due to invalid information, on this stage user would lost its whole filled information which he was filling from last one hour. So this kind of situations makes the need of State Management of a web page.

Solution of the above problem lies in State Management.

State management is the process by which you maintain state and page information over multiple requests for the same or different pages. ASP.Net technology offers following state management techniques.

Types of State Management in Asp.Net:

Client side state Management: View state, control state, hidden fields, cookies, and query strings all involve storing data on the client in various ways.

- View state
- Control state
- Hidden fields
- Cookies
- Query strings

Server side state Management: application state, session state, and profile properties all store data in memory on the server. Each option has distinct advantages and disadvantages, depending on the scenario.

- Session State
- Application State
- Cache

Client side State Management in Asp.Net:

View State:
ASP.Net technology provides View State feature to the web forms. View State is used to remember controls state when page is posted back to server. Asp.Net uses View State to track the values in the Controls. You can add custom values to the view state. The ViewState property provides a dictionary object for retaining values between multiple requests for the same page. This is the default method that the page uses to preserve page and control property values between round trips.

ASP.Net stores view state on client site in hidden field __ViewState in encoded form. When the page is processed, the current state of the page and controls is hashed into a string and saved in the page as a hidden field, for multiple hidden fields if the amount of data stored in the ViewState property exceeds the specified value in the MaxPageStateFieldLength property. When the page is posted back to the server, the page parses the view-state string at page initialization and restores property information in the page.

Code Semple:

```
public partial class _Default : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
if(ViewState["NameOfUser"] != null)
NameLabel.Text = ViewState["NameOfUser"].ToString();
else
NameLabel.Text = "Not set yet...";

}

protected void SubmitForm_Click(object sender, EventArgs e)
{
ViewState["NameOfUser"] = NameField.Text;
NameLabel.Text = NameField.Text;

}

}
```

View state can be used for storing any type of data because it is of object type. View state provides good performance for application without giving any load to server. You can enable and disable view state behavior of page and its control by specifying 'enableViewState' property to true and false. You can also store custom information in the view state as described in above code sample. This information can be used in round trips to the web server.

Cookie:
A cookie is a small amount of data that is stored either in a text file on the client file system or in-memory in the client browser session. Cookies are one of several ways to store data about web site visitors during the time when web server and browser are not connected. It contains site-specific information that the server sends to the client along with page output. Cookies can be temporary (with specific expiration times and dates) or persistent. Practically, cookie is a small text file sent by web server and saved by web browser on client machine.

Common use of cookies is to remember users between visits. You can use cookies to store information about a particular client, session, or application. The cookies are saved on the client device, and when the browser requests a page, the client sends the information in the cookie along with the request information. The server can read the cookie and extract its value. A typical use is to store a token (perhaps encrypted) indicating that the user has already been authenticated in your application.

Create a cookie in ASP.NET: The System.Web namespace offers a class called HttpCookie to create cookies.

```
// Use this line when you want to save a cookie
Response.Cookies["MyCookieName"].Value = "MyCookieValue";
```

```
// How long will cookie exist on client hard disk
Response.Cookies["MyCookieName"].Expires = DateTime.Now.AddDays(1);

// To add multiple key/value pairs in single cookie
Response.Cookies["VisitorData"]["FirstName"] = "Richard";
Response.Cookies["VisitorData"]["LastVisit"] = DateTime.Now.ToString();
```

Read a cookie in ASP.NET:

```
string MyCookieValue;
// We need to perform this check first, to avoid null exception
// if cookie not exists
if(Request.Cookies["MyCookieName"] != null)
MyCookieValue = Request.Cookies["MyCookieName"].Value;
```

Delete cookie in ASP.NET: To delete existing cookie we actually just set its expiration time to some time in the past. You can do it with code like this:

```
// First check if cookie exists
if (Request.Cookies["MyCookieName"] != null)
{
//Set its expiration time somewhere in the past
Response.Cookies["MyCookieName"].Expires = DateTime.Now.AddDays(-1);

}
```

Query String:

A query string is information that is appended to the end of a page URL. They are commonly used to hold data like page numbers or search terms or other data that isn't confidential. Unlike ViewState and hidden fields, the user can see the values which the query string holds without using special operations like View Source.

A typical query string might look like the following example:

http://DirectoryName/default.aspx?UserType=2&PostType=3

In the URL path above, the query string starts with a question mark (?) and includes two attribute/value pairs, one called "UserType" and the other called "PostType".

The Query String Structure: As written earlier, query strings are appended to the end of a URL. First a question-mark(?) is appended to the URL's end and then every parameter that we want to hold in the query string. The parameters declare the parameter name followed by = symbol which followed by the data to hold. Every parameter is separated with the ampersand(&) symbol. You should always use the HttpUtility.UrlEncode method on the data itself before appending it.

Get Query String Values on PageLoad:

```
Private void Page_Load (object sender, System.EventArgs e)
{
string _userType = "";
string _postType = "";

if(Request.QueryString["UserType "] != null )
_userType = Request.QueryString["UserType"];


if(Request.QueryString["PostType "] != null )
_postType = Request.QueryString["PostType"];

}
```

Hidden fields:

Hidden fields technique is widely used in ASP.NET programming. This is used to store a value that needs to be persisted across posts to the server. Hidden fields are html input control with hidden type that store hidden data in the html. It is rendered as an element. An example for a hidden field can look like this:

```
< input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
```

In the example above, I chose to show the event target hidden field in order to indicate that even in postback mechanism hidden fields are being used. The data stored in a hidden field is available when the form is processed on the server or when we use javascript. You can use the HiddenField control to store state values. It can store only one value in their value property. The value is saved as a string and therefore in order to use it for other types you need to perform casting. Hidden field’s data is submitted to the server only in HTTP post operation. You can see the stored data easily by using the View Source operation of the browser. The value of a HiddenField is rendered to the client browser, so it is not suitable for storing security-sensitive values. Be aware not to use hidden fields to store confidential data! The values has page context and therefore when you leave a page the data stored in the hidden fields is disposed.

<u>Server Side State Management:</u>

Session State:

ASP.NET allows you to save values by using session state. This is an instance of the HttpSessionState class. Session state provides a place to store values that will persist across page requests. You can store values that need to be persisted for the duration of a user's session in session variables. These variables are unique to each user session and can be accessed in any ASP.NET page within an application. You can set and access session information from within an ASP.NET application. Values stored in Session are stored on the server and will remain in memory until they are explicitly removed or until the Session expires. You can set and access session information from within an ASP.NET application. For example

```
//Assign a value to the myvariable session variable. Session["myvariable"] = "somevalue"; //Retrieve the value of the myvariable session variable. string myString; if (Session["myvariable"] != null)
myString = (string)Session["myvariable"];
```

It is important to remember that session variables are now objects. Thus, to avoid a run-time error, you should check whether the variable is set before you try to access it. Session variables are automatically discarded after they are not used for the time-out setting. The Session Timeout is adjustable through a web.config setting but increasing the timeout value can put memory pressure on your server that may be undesirable. < sessionState timeout="number of minutes" /> **Other commonly used Session methods are: Session.Abandon()**: removes the Session and all items that it contains **Session.Clear()**: removes all items from the Session **Session.RemoveAll()**: removes all items from the Session **Session.Remove("itemName")**: removes the item that was stored under the name "itemName" **<u>ASP.NET supports three modes of session state:</u> InProc**: (The Default) Session state exists within the process the web is using. **StateServer**: Session data is sent to the configured stateserver service. **SQLServer**: Session data is store in the configured sql server database. Note You can use all three modes with in-memory cookie or cookieless session ID persistence.

You can use session state to accomplish the following tasks:

- Uniquely identify browser or client-device requests and map them to an individual session instance on the server.
- Store session-specific data on the server for use across multiple browser or client-device requests within the same session.
- Raise appropriate session management events. In addition, you can write application code leveraging these events.

<u>Application State</u>

- Application state is one of the ways available to store some data on the server and faster than accessing the database.
- The data stored on the Application state is available to all the users(Sessions) accessing the application.
- So application state is very useful to store small data that is used across the application and same for all the users. Also we can say they are global variables available across the users.
- Don't store heavy data in ApplicationState because it is stored

on the server and can cause performance overhead if it is heavy.

- Technically the data is shared amongst users via HTTPApplcationState class and the data can be stored here in key value pair. It can also be accessed through Application property of the HTTPContext class.

How Application State Works

- An instance of HttpApplicationState is created when first time a request comes from any user to access any resource from the application. And this can be accessed through the property Application property of HTTPContext Object.
- All HTTPModules and Handlers have access to this property. The lifetime of the values spans through the lifetime of the ASP.NET application until the application is unloaded.
- Normally, we set these Application variables in Application_OnStart event in Global.asax file and access and modify through ASP.NET pages.

How to Save values in Application State

- One thing to keep in mind is that application state stores the data as of Object type, so at the time of reading, the values we need to convert it in the appropriate type.

- So normally, we use to store the Application wise data in Application state which is shared across the users. So we can save the data in Application_OnStart method in Global.asax file as:

  Global.asax

```
void Application_Start(object sender, EventArgs e)
  {
        Application["Message"] = "Welcome to my Website";
```

```
            }
```

How To Read Values from Application State

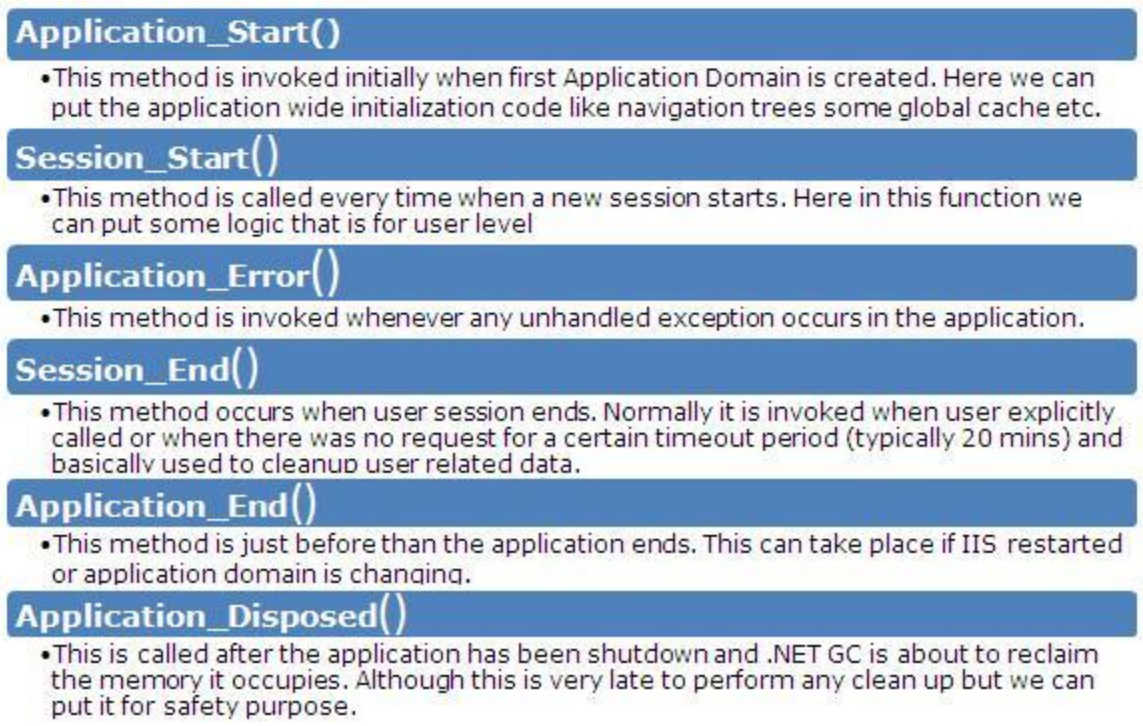
- We should just have a safety check to see whether the value we are accessing is null
- If there is no data in Application state then it will return null and if we'll try to cast it in any different type, it'll throw an exception.
- Because Application state stores the data in object form so we need to typecast after reading it.
- So we can read the value as:

Default.aspx -> Page_Load

```
if(Application["Message"] !=null)
{
     string message = Application["Message"] as string;
}
```

What is Global.asax

To handle **application** events or methods, we can have a file named *Global.asax* in the root directory of your**application**. At any single point of time, an `HTTPApplication` instance handles only one request, so we don't need to think about locking and unlocking of any non `static` members, but for `static` members we do require. I'll discuss it in detail in a later section of this article. Following are the commonly used events in the *global.asax* file.



Application_Start()
- This method is invoked initially when first Application Domain is created. Here we can put the application wide initialization code like navigation trees some global cache etc.

Session_Start()
- This method is called every time when a new session starts. Here in this function we can put some logic that is for user level

Application_Error()
- This method is invoked whenever any unhandled exception occurs in the application.

Session_End()
- This method occurs when user session ends. Normally it is invoked when user explicitly called or when there was no request for a certain timeout period (typically 20 mins) and basically used to cleanup user related data.

Application_End()
- This method is just before than the application ends. This can take place if IIS restarted or application domain is changing.

Application_Disposed()
- This is called after the application has been shutdown and .NET GC is about to reclaim the memory it occupies. Although this is very late to perform any clean up but we can put it for safety purpose.

Figure: Methods in Global.asax

For more information visit http://www.codeproject.com/Articles/87316/A-walkthrough-to-Application-State