ASP.NET State Management Techniques

This article is for complete beginners who are new to ASP.NET and want to get some good knowledge about ASP.NET State Management.

What is the need of State Management?

In today's world everything is becoming Internet / Web based. Whatever work it may be, whether related to Banking, Education, Entertainment or Shopping we can do it over the internet very easily. If we try to compare today's generation with the previous generation then I would simply say that we cannot live without the Internet.

As food, clothing and shelter are our basic needs, the Internet has also become a part of our basic necessities. If we think about our future, then I would say "Cloud Computing", which is already well developed and is currently still evolving, will take this Internet World to a higher level.

So now, enough of this lecture, and let's move on with the actual content.

Let us assume that someone is trying to access a banking website and he has to fill in a form.
So the person fills in the form and submits it. After submission of the form, the person realizes he has made a mistake. So he goes back to the form page and he sees that the information he submitted is lost. So he again fills in the entire form and submits it again. This is quite annoying for any user. So to avoid such problems "STATE MANAGEMENT" acts as a savior for developers like us.

If you do not understand what the word "STATE" means, then think about it, as some interaction between the User and the Server.

Definition - State Management can be defined as the technique or the way by which we can maintain / store the state of the page or application until the User's Session ends.

What is HTTP Protocol?

Hyper Text Transfer Protocol is the base or I would rather say, is the core part or the foundation of our World Wide Web. It basically makes use of the TCP Protocol for communicating with the server. A specific port is being used by this protocol while communicating with the server.

HTTP provides various methods which represent the actions it will perform while communicating with the server. Some of its methods are "GET", "POST", "DELETE", "HEAD", etc....

HTTP acts like a bridge between the client and the server.

If a user tries to access a website, say (http://www.asp.net/) and let us assume that the requested page is an ASPX page. So what happens is, the HTTP Protocol takes the request being sent by the client (using his browser) and sends it to the server. The server then locates the requested page and tells the ASP.NET engine to process the request.

The ASP.NET Engine processes the request and sends a response back to the client in an HTML format.

Once again the HTTP protocol is relevant; the HTTP protocol takes the response and sends it to the client, thus the response is shown in the client's browser.

So the HTTP Protocol, which is also called a "Request - Response" Protocol, acts like a bridge between the client and the server.

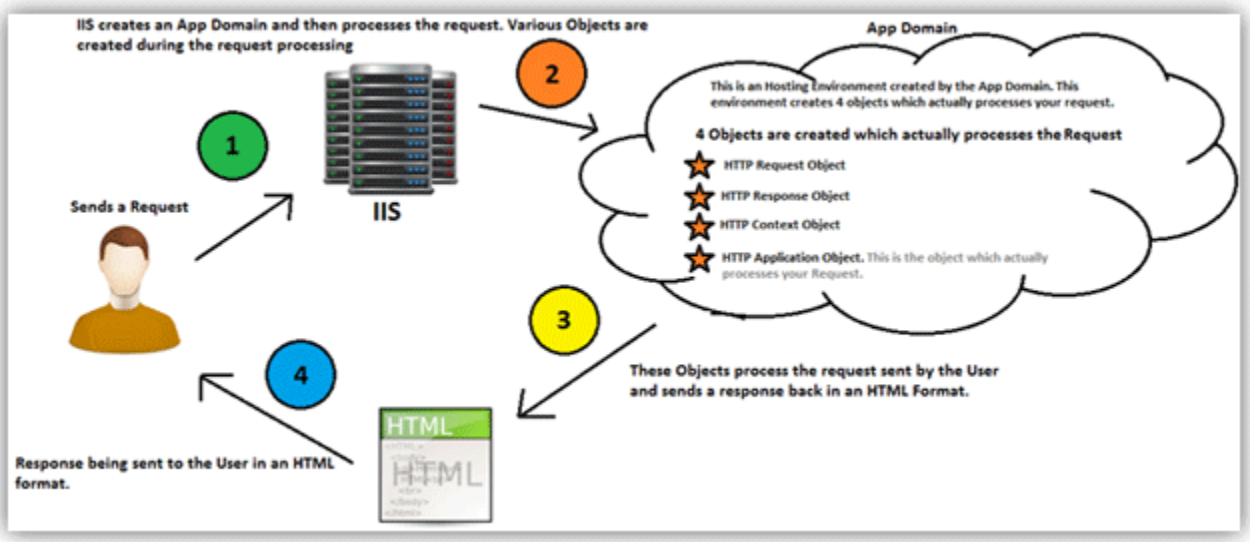
Some Technical Facts

It is very important to know what happens when the user sends a request to the server and what the server does to process that request. This is not related to state management, but it is the core part of any Request-Response Process. I will be explaining in brief about that process. The explanation includes information about: "What are all the objects created during the request?", "Who processes the Request?" etc...

Fact 1 - **This is just a brief description of the ASP.NET environment creation:**

- Whenever the user sends a request to the server, the server first locates the page and checks who will process the request.
- Then it creates an App Domain which is a space in the memory / Memory Unit which creates a Hosting Environment.
- It then creates 4 different Objects like:
    - Http Response Object
    - Http Request Object
    - Http Context Object
    - Http Application Object
- These objects then process the request and sends a response to the user in HTML format.

I have represented the entire flow in a diagram so that you will understand the concept in a much better way.



- **Note** - The App Domain is created only once for each Website or an application. The Http Application Object is also created only once. The creation happens when the user requests a website for the first time.
- To Learn more about the ASP.Net Environment Creation, please click on the following link, an article written by Shivprasad Koirala:

http://www.c-sharpcorner.com/uploadfile/shivprasadk/Asp-Net-application-and-page-life-cycle/

This is again a video link, which is beautifully presented by Shivprasad Sir on ASP.NET Environment Creatio:
http://www.youtube.com/watch?v=sOVBrLvhOHU

Fact 2 - HTTP Protocol is a Stateless protocol

- As I mentioned above, State basically means an interaction between the user and the server and the HTTP Protocol acts like a bridge between them.
- But since the HTTP Protocol is a Stateless protocol, it is not able to store the state during that session. With each request the objects are created and memory is being allocated. These resources are destroyed once the server completes its process.
- So the cycle of "Creation Of Objects and Utilization of Server Memory" ---- "Processing Of the Request" ---- "Destruction of the Created Resources" continues for each and every user's request.
- In such a case, the information given by the user or the State of an Application gets lost during each round trip. So this is where State Management helps.

I hope, by now you have understood the basic concept and the need of State Management. In the next part I will be explaining about the various techniques which ASP.NET provides to maintain the State of an Application.

So come on, let's move onward.

State Management Techniques

ASP.NET provides us with 2 ways to manage the state of an application. It is basically divided into the 2 categories:

1. Client Side State Management.
2. Server Side State Management.

Client Side State Management - It is a way in which the information which is being added by the user or the information about the interaction happened between the user and the server is stored on the client's machine or in the page itself. The server resources (e.g. server's memory) is not at all utilized during the process.

This management technique basically makes use of the following:

a. View State
b. Hidden Fields
c. Query String

d. Cookies

View State

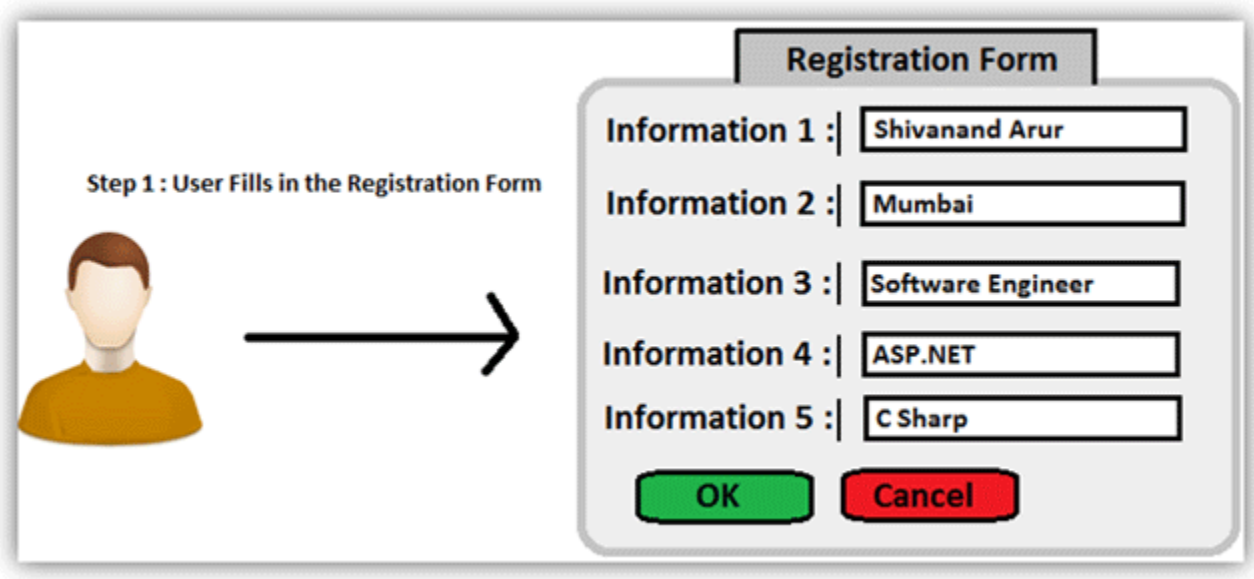
View State can be used to maintain the State at a page level. The term "Page Level" means that the information is being stored for a specific page and until that specific page is active (i.e. the page which is being currently viewed by the user). Once the user is re-directed or goes to some other page, the information stored in the View State gets lost.

It basically makes use of a "Dictionary Object" to store data, which means that the information is stored in a key and value pair. It stores this information in a Hidden field on the page itself in a hashed format. A View State can store a string value only of a specific length. If the length is exceeded then the excess information is stored in another hidden field.

Using a View State is quite simple. In fact, it is the default way for storing the page or the control's information. Typically the View State is used, when we want a user to be re-directed to the same page and the information being added by the user remains persistent until the user is on the same page.

Here I have shown how to use and assign a value to a View State and how to read a value from a View State.

Example - A user fills in a Registration Form.



Storing Value in a View State

```
protected void btnOK_Click(Object sender, EventArgs e)
{
    ViewState["Username"] = txtFName.Text;  // Storing the First Name of the User in the View State.
}
```

Information - Here in the code above, "Username" is the key and the value is the text being inputted by the user (txtFName.Text).

Retrieving Value from a View State

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!String.IsNullOrEmpty(Convert.ToString(ViewState["Username"])))
    {
        lblUsername.Text = Convert.ToString(ViewState["Username"]);
    }
}
```

Information - Here as you can see in the image, I am retrieving the value from the View State, in the Page Load event of a page by first checking if the View State is not empty or null and then assigning its value to a Label. This is just a simple example which is shown to make you understand about, how to use a View State.

View State Information is stored in a Hashed Format

```
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTA1Mjg2NTQ4M2Rkutjuh48p2Tc+/aL21UtgBqbW9aY=" />
</div>
```

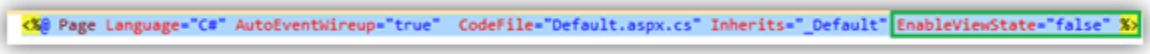
Information - If you look at the page source, then this is the way View State stores the value.

View State Settings

View State is customizable. With the term "Customizable" I mean, that we can apply settings to a View State to store a value at various levels. We can set the View State at various levels like:

1. **Setting View State at Application Level** - If we do not want our pages in the Application to use view state, then we can disable or enable it in the web.config file, as shown in the image below.

2. **Setting View State at Page Level** - If we do not want a specific page to use View State, then we can disable or enable it in the @ Page Directive which is the first line of our aspx page.



3. **Setting View State at Control Level** - If we do not want a specific control to use View State, then we can disable or enable it at a Control Level as follows:

Advantages of using a View State

1. It is very simple to use.
2. Data is stored in hashed format and hence a layman won't be able to understand the value of the View State (It still can be hacked by Hackers, so to make it more secure we should try to store the value in an encrypted format.). Check out this link http://msdn.microsoft.com/en-us/library/ms178199(v=vs.85).aspx to understand, how to secure View State.
3. It is customizable, as shown above.

Disadvantages of using a View State

1. Information is not encrypted, so it can be easy for a Hacker to get its value.
2. Cannot be used to store sensitive data (eg: Passwords, Credit Card Pins, etc).
3. Might make a page heavy if lots of data is stored in View State.

Hidden Fields

ASP.NET provides a server control called "Hidden Field" which can be used to store a value at a page level, which is similar to a View State. The value of the Hidden Field is sent along in the HTTP Form Collection along with the value of other controls.

Example - Take the same example of the User filling an online registration form.

Setting Value to a Hidden Field

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    hdfValue.Value = txtUsername.Text;
}
```

Information - Here I am setting a value to a Hidden field, as shown in the image above. I am taking input from the User (txtUsername.Text) and assigning it to the Hidden Field's Value property. It basically stores only 1 value in its property.

Retrieving Value from a Hidden Field

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!String.IsNullOrEmpty(hdfValue.Value))
    {
        lblName.Text = hdfValue.Value;
    }
}
```

Information - As you can see, I am retrieving a value from the value in the Hidden field and assigning it to a label. The Hidden Field's "Value" property returns a string by default. If you want an integer value, then you will have to convert it explicitly.

A Hidden Field stores a value at a Page Level

If you look at the page source after assigning a value to a Hidden Field, then you will see that it stores the value on the page itself. Once the user is redirected to some other page, then the value is lost.

```
<tr>
    <td>
        <span id="lblAns"></span>

        <input type="hidden" name="hdfValue" id="hdfValue" value="Shivanand" />

    </td>
</tr>
```

I had passed "Shivanand" as a value from my textbox and assigned that value to a hidden field.

Advantages

1. Very simple to use.
2. Hidden Fields store the value in the page itself, hence do not use server resources.

Disadvantages

1. Will make a page heavy, if too many Hidden Fields are used to store data.
2. Cannot store sensitive data, as the value that is stored is neither hashed, nor encrypted.

Query String

A Query String is a string which is appended to the end of the Page URL. It is very simple to use and can be used to send data across pages. It stores information in a key - value pair. A "?" signature is used to append the key and value to the page URL.

Way to pass a value using Query String

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    Response.Redirect(String.Format("Default.aspx?Username={0}", txtUsername.Text.Trim()));
}
```

Information - This preceding code will send the Username to another page and use that value on that page. We should never send sensitive data using Query String, since the data that is being sent can easily be tampered with by anybody. If you still want to send information using Query String, then encrypt the data using any ASP.NET Encryption technique so the data cannot be tampered with.

Way to read Query String value

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        string Name = this.Request.QueryString["Username"];
        lblUsername.Text = Name;
    }
}
```

Information - To read the value of the query string, you should use the Request Object as shown in the image above. You can send multiple parameters in the query string along with its respective value. For sending multiple parameters, you can separate the parameters using the "&"delimeter.

Cookies

ASP.Net provides another way of state management, which is by using Cookies. Cookies are one of the best ways of storing information. It is nothing but a text file which is stored on the client's machine.

When the user sends a request to the server, the server creates a cookie and attaches a header and sends it back to the user along with the response. The browser accepts the cookie and stores it at a specific location on the client's machine. Even large sites like Gmail, Facebook, Yahoo use cookies.
There are 2 ways of assigning / storing values in cookies.

1. **Using the Request Object**

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    this.Request.Cookies["Username"].Value = txtUsername.Text.Trim();
}
```

Information - Here I have made use of the Request object. The Cookies property of the HTTPResponse Object and the HTTPRequest Object can be used to assign values to the Cookies Collection and get values back from the Collection. We can also store multiple values in a cookie.

2. **Using the HTTPCookies Object**

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    HttpCookie myCookie = new HttpCookie("Username",txtUsername.Text.Trim());
    myCookie["Age"] = "22";
    myCookie.Expires.AddHours(2);
    Response.Cookies.Add(myCookie);
}
```

Information - Another way of adding values to a cookie is using the "HTTPCookie" class. Its constructor takes either 1 or 2 parameters. If you want your cookie to expire after a specified time then you can even set the expiration date for that cookie. And the last line "Response.Cookies.Add(myCookie)" will add that cookie to the Cookies Collection.

Advantages

1. Very easy to use.
2. Stored on the client's machine, hence no server resources are utilized.

Disadvantages

1. A user can disable cookies using browser settings.
2. Since the cookies are stored on the client's machine, there are chances that if the client's machine is hacked then the hacker can view these cookie values. Hence we should not store sensitive information in cookies.

Server Side State Management - It is another way which ASP.NET provides to store the user's specific information or the state of the application on the server machine. It completely makes use of server resources (the server's memory) to store information.

This management technique basically makes use of the following:

a. Application State
b. Session State

Application State:

1. If the information that we want to be accessed or stored globally throughout the application, even if multiple users access the site or application at the same time, then we can use an Application Object for such purposes.
2. It stores information as a Dictionary Collection in key - value pairs. This value is accessible across the pages of the application / website.
3. There are 3 events of the Application which are as follows
    - Application_Start
    - Application_Error
    - Application_End

Example - Just for an example, I am setting the Page title in the Application Start event of the Global.asax file.

Code for setting value to the Application Object - "PageTitle" is the Key and "Welcome to State Management Application" is the value.

```
void Application_Start(object sender, EventArgs e)
{
    this.Application["PageTitle"] = "Welcome to State Management Application";
}
```

Code for reading value from the Application Object

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        this.Page.Title = Convert.ToString(this.Application["PageName"]);

    }
}
```

Session State

Session is one of the most common way which is being used by developers to maintain the state of the application. The Session basically stores the values as a dictionary collection in key/value pairs. It completely utilizes server resources to store the data. It is a secure way of storing data, since the data will never be passed to the client.

For each and every user, a separate Session is created, and each and every Session has its Unique ID. This ID is being stored in the client's machine using cookies. If there are multiple users who are accessing a web application, then for each user a separate Session is created. If a single user logs in and logs out the Session is killed, then if the same user again logs into the application, then a new Session ID is being created for the same user.

The Session has a default timeout value (20 minutes). We can also set the timeout value for a session in the web.config file.

```
<sessionState cookieless="UseUri" mode="SQLServer"></sessionState>
```

There are various ways in which we can store a session and they are as follows:

1. OFF
2. InProc
3. State Server
4. SQL Server

OFF - If we do not want to use sessions in our application, then we can set the mode as "OFF".

InProc - This is the default mode which is used in ASP.NET to store session variables. InProc mode basically stores the session value in the process in which the ASP.NET application is running. Since it is stored in the same process, it provides high performance as compared to other modes.

State Server - If we use this mode, then a separate Windows Service which runs in a different process stores the Session. It basically isolates the Session from the ASP.NET running process. It provides less performance as compared to the Inproc mode.

SQL Server - This mode stores the Session in SQL Server instead of the server's memory. If our application is stored on multiple server machines, then it becomes difficult to use the "Inproc" mode since the request can go to any server for processing. So if we want to use sessions in such cases, then we can store the Session in SQL Server so that it becomes centralized and can be accessed by any server during the request process. It is the most secure way of storing Sessions but gives the lowest performance for an application.

Code to write values into a Session

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    Session["Username"] = txtUsername.Text.Trim();
    Response.Redirect("Default.aspx");
}
```

Code to read values from Session

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        lblUsername.Text = Convert.ToString(Session["Username"]);
    }
}
```

Written by Shivanand Arur on Sep 16, 2012.