**State Management in ASP.NET**

***Hyper Text Transfer Protocol (HTTP)***

is a communication protocol which is implemented in the "World Wide Web (WWW)". It is a request/response style protocol.

Clients (browsers, spider, etc) will request to a server (web server) and the server responds to these requests. HTTP uses TCP protocol for communication. It connects to a specific port (default is 80) to the server and communicates via that port. Once the response is received completely, client programs will be disconnected from the server. For each request, client programs have to acquire a connection with servers and do all the request cycles again.ASP.NET files are just text files which will be placed in the server and served upon the request. When a request comes for a page, the server will locate the requested file and ask the ASP.NET engine to serve the request. The ASP.NET engine will process the server tags and generate HTML for it and return back to the client. HTTP is a stateless protocol and the server will abandon the connection once the request is served.

Since HTTP is stateless, managing state in web applications is challenging. State management techniques are used to maintain user state throughout the application. Managing state in a stand-alone application is trivial as they don't have a request/response nature. Information you want to maintain over the lifetime of a web applications depends on your context. It can be from simple numbers to very complex objects. Understanding the state management techniques play a major role in creating efficient web applications. ASP.NET is very rich in state management techniques. The following are the commonly used state management techniques.

* QueryString
* Cookies
* Cache
* ViewState
* Session state
* Application state
* Static variables
* Profiles

**QueryString**

This is the most simple and efficient way of maintaining information across requests. The information you want to maintain will be sent along with the URL. A typical URL with a query string looks like

***www.uobabylon.edu.iq/search.aspx?q=hassan***

The URL part which comes after the ? symbol is called a QueryString. QueryString has two parts, a key and a value. In the above example, query is the key and hassan is its value. You can send multiple values through querystring, separated by the & symbol. The following code shows sending multiple values to the secondPage.aspx page.

***Response.Redirect("secondPage.aspx?id=1&name=hassan");***

The secondPage.aspx page will get the values like in the following table.

**Key Value**

**id 1**

**name hassan**

The following code shows reading the QueryString values in secondPage.aspx

*string id = Request.QueryString["id"];*

*string name = Request.QueryString["name"];*

**Cookie**

A cookie is a small file which is stored in the visitor's hard disk drive. This is helpful for storing small and trivial information. According to the RFC, a cookie can have a maximum size of 4KB. The web server creates a cookie, attaches an additional HTTP header to the response, and sends it to the browser. The browser will then create this cookie in a visitor's computer and includes this cookie for all further requests made to the same domain. Servers can read the cookie value from the request and retain the state.The server adds the following to the HTTP header for creating a cookie

***Set-Cookie: key=value***

The browser reads the above value and creates cookies at the user's end. It adds the cookie value to the request like

***Cookie: key=value***

*Note: The location where the cookie is stored is completly controlled by the browser. Sometimes it may keep the cookie in its memory instead of creating a file.*

**Session State**

A cookie is very simple and is not suitable for sophisticated storage requirements. Session state is a workaround for this problem and it gives a method to keep more complex objects securely. ASP.NET allows programmers to keep any type of objects in session. Data stored in session will be kept in server memory and it is protected as it will never get transmitted to a client. Every client that uses the application will have separate sessions. Session state is ideal for storing user specific information.

The following code shows storing a string value in session.

***Session["name"] = "Hassan";***

Session accepts a System.Object type. So you need a type cast when reading. Reading values from session is like

***string name = Session["name"] as string; or you can use .ToString();***

**View State**

Viewstate is a page-level state that is used to hold the property values of the controls in a webform between postbacks. Viewstate is a hidden field that holds the values of the controls in an encoded form. The property values of the controls are encoded in Base64 and stored in the hidden field called \_\_Viewstate. Viewstate can be enabled at control level, page level or at application level for all the pages.

View state is simply text. It is an aggregate of values of controls on a page. It's a string that contains values of page controls hashed and encoded in some manner. The view state contains no information about the server or the client. It only comprises information about the page itself and its controls. It lives along with the page in the user's browser.

As a rule, view state is stored right in the page and therefore it travels with it back and forth. Remember good old hidden input fields? View state is nothing more than a hidden input which holds a hash of control values. If you view the source of an ASP.NET web form you will see something like this:

***<input type="hidden" name="\_\_VIEWSTATE"***

***value="CEbzzzEnmmz+Bc8IDFlnpgCLJ/HB00...>***

Nobody in sane mind can read these strings fluently. What you see here is a base64 external link encoded stringThis is how you store a piece of useful data in the view state:

ViewState ["MyName"] = "Hassan";

When the page posts back its view state is taken apart (decoded) on the server and each control participating in the view state gets its value restored. There's an interesting gotcha you need to be aware of. Some controls get their values restored automatically (courtesy of ASP.NET) and you don't need to maintain their values

**Application State**

ASP.NET implements application state using the System.Web.HttpApplicationState class. It provides methods for storing information which can be accessed globally. Information stored on application state will be available for all the users using the website. Usage of application state is the same as sessions. The following code shows storing a value in an application variable and reading from it.

***Application["pageTitle"] = "Welcome to my website ";***

***// Reading the value from application variable***

***string pageTitle;***

***if (Application["pageTitle"] != null)***

***pageTitle = Application["pageTitle"].ToString();***

**Profile State**

Session data is lost when the visitor leaves the webpage. What if you need to persist all the user information for a long time? ASP.NET Profile is the answer. It provides a neat way to persist information for a long time. Creating a profile is trivial. You only need a few entries in the web.config file as seen below

<profile>

<properties>

<add name="Id" type="Int32"/>

<add name="Name"/>

<add name="Age" type="Int32"/>

</properties>

</profile>

*ASP.NET generates a strongly typed class for accessing profile data. Data type for the properties are chosen depending upon the type value. Default type is string if no type is specified. Following code shows setting values and reading back from profile.*

Profile is very handy in many situations. However, it has the following drawbacks

* It allows to keep only serializable types
* Reading data from profile requires database access which can potentially make your application less performant. If your website uses profiles heavily, you have to cache the results to avoid unncessary database calls.

**Keeping State in Static (shared in VB) Variables**

Static variables will have a lifetime until the application domain where it is hosted ends. ASP.NET hosts each website in a separate application domain to provide isolation with other websites hosted on the same server.

Consider you have a page where all product details are displayed. Product details are fetched from the database and filled into custom collection and returned back to the page. To avoid fetching product details all time for each visitors, we can load it when it is requested for the first time and keep it in a static variable to serve for the next requests.