HTTP is a stateless protocol. ASP.NET framework helps in storing the information regarding the state of the application, which consists of:

Page state

Session state

When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.

The ASP.NET runtime transforms the .aspx page into an instance of a class, which inherits from the base class page of the .Net framework. Therefore, each ASP.NET page is an object and all its components

The ASP.NET life cycle could be divided into two groups:

Application Life Cycle

Page Life Cycle

ASP.NET Application Life Cycle

The application life cycle has the following stages:

* User makes a request for accessing application resource, a page. Browser sends this request to the web server.
* A unified pipeline receives the first request and the following events take place:
  + An object of the class ApplicationManager is created.
  + An object of the class HostingEnvironment is created to provide information regarding the resources.
  + Top level items in the application are compiled.
* Response objects are created. The application objects such as HttpContext, HttpRequest and HttpResponse are created and initialized.
* An instance of the HttpApplication object is created and assigned to the request.
* The request is processed by the HttpApplication class. Different events are raised by this class for processing the request.

An ASP.NET page is also a server side file saved with the .aspx extension. It is modular in nature and can be divided into the following core sections:

* Page Directives
* Code Section
* Page Layout

Some events cause the form to be posted back to the server immediately, these are called the postback events. For example, the click event such as, Button.Click.

Some events are not posted back to the server immediately, these are called non-postback events.

For example, the change events or selection events such as TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.

Server Side

# ASP.NET - Server Side

We have studied the page life cycle and how a page contains various controls. The page itself is instantiated as a control object. All web forms are basically instances of the ASP.NET Page class. The page class has the following extremely useful properties that correspond to intrinsic objects:

* Session
* Application
* Cache
* Request
* Response
* Server
* User
* Trace

HttpContext.Current.Server.UrlEncode

HttpContext.Current.Server.HtmlEncode

## Request Object

The request object is an instance of the System.Web.HttpRequest class. It represents the values and properties of the HTTP request that makes the page loading into the browser.

## Response Object

The Response object represents the server's response to the client request. It is an instance of the System.Web.HttpResponse class.

The Response.Redirect() method allows transferring the user to another page, inside as well as outside the application. It requires a round trip.

# ASP.NET - HTML Server

The HTML server controls are basically the standard HTML controls enhanced to enable server side processing.

They are specifically converted to a server control by adding the attribute runat="server" and adding an id attribute to make them available for server-side processing.

# ASP.NET - Client Side

ASP.NET client side coding has two aspects:

* **Client side scripts** : It runs on the browser and in turn speeds up the execution of page. For example, client side data validation which can catch invalid data and warn the user accordingly without making a round trip to the server.
* **Client side source code** : ASP.NET pages generate this. For example, the HTML source code of an ASP.NET page contains a number of hidden fields and automatically injected blocks of JavaScript code, which keeps information like view state or does other jobs to make the page work.

**Client Side Source Code**

We have already discussed that, ASP.NET pages are generally written in two files:

* The content file or the markup file ( .aspx)
* The code-behind file

The content file contains the HTML or ASP.NET control tags and literals to form the structure of the page. The code behind file contains the class definition. At run-time, the content file is parsed and transformed into a page class.

# ASP.NET - Directives

ASP.NET directives are instructions to specify optional settings, such as registering a custom control and page language. These settings describe how the web forms (.aspx) or user controls (.ascx) pages are processed by the .Net framework.

The syntax for declaring a directive is:

<%@ directive\_name attribute=value [attribute=value] %>

## The Application Directive

The Application directive defines application-specific attributes. It is provided at the top of the global.aspx file.

The basic syntax of Application directive is:

<%@ Application Language="C#" %>

## The Assembly Directive

The Assembly directive links an assembly to the page or the application at parse time. This could appear either in the global.asax file for application-wide linking, in the page file, a user control file for linking to a page or user control.

The basic syntax of Assembly directive is:

<%@ Assembly Name ="myassembly" %>

## The Import Directive

The Import directive imports a namespace into a web page, user control page of application. If the Import directive is specified in the global.asax file, then it is applied to the entire application. If it is in a page of user control page, then it is applied to that page or control.

The basic syntax for import directive is:

<%@ namespace="System.Drawing" %>

## The Master Directive

The Master directive specifies a page file as being the mater page.

The basic syntax of sample MasterPage directive is:

<%@ MasterPage Language="C#" AutoEventWireup="true" CodeFile="SiteMater.master.cs" Inherits="SiteMaster" %>

## The Page Directive

The Page directive defines the attributes specific to the page file for the page parser and the compiler.

The basic syntax of Page directive is:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" Trace="true" %>

## The Register Directive

The Register derivative is used for registering the custom server controls and user controls.

The basic syntax of Register directive is:

<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>

# ASP.NET - Managing State

ASP.NET manages four types of states:

* View State
* Control State
* Session State
* Application State

## View State

The view state is the state of the page and all its controls. It is automatically maintained across posts by the ASP.NET framework.

The view state could be enabled or disabled for:

* **The entire application** by setting the EnableViewState property in the <pages> section of web.config file.
* **A page** by setting the EnableViewState attribute of the Page directive, as <%@ Page Language="C#" EnableViewState="false" %>
* **A control** by setting the Control.EnableViewState property.

## Session State

When a user connects to an ASP.NET website, a new session object is created. When session state is turned on, a new session state object is created for each new request. This session state object becomes part of the context and it is available through the page.

It can store all the primitive data types and arrays composed of primitive data types, as well as the DataSet, DataTable, HashTable, and Image objects, as well as any user-defined class that inherits from the ISerializable object.

## Application State

The ASP.NET application is the collection of all web pages, code and other files within a single virtual directory on a web server. When information is stored in application state, it is available to all the users.

To provide for the use of application state, ASP.NET creates an application state object for each application from the HTTPApplicationState class and stores this object in server memory. This object is represented by class file global.asax.

Application State is mostly used to store hit counters and other statistical data, global application data like tax rate, discount rate etc. and to keep the track of users visiting the site.

# ASP.NET - Validators

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

* RequiredFieldValidator
* RangeValidator
* CompareValidator
* RegularExpressionValidator
* CustomValidator
* ValidationSummary

## CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

<asp:CustomValidator ID="CustomValidator1" runat="server"

ClientValidationFunction=.cvf\_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>

## ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

* **ShowSummary** : shows the error messages in specified format.
* **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

<asp:ValidationSummary ID="ValidationSummary1" runat="server"

DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />

## Validation Groups

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup*property.

# ASP.NET - Database Access

ASP.NET allows the following sources of data to be accessed and used:

* Databases (e.g., Access, SQL Server, Oracle, MySQL)
* XML documents
* Business Objects
* Flat files