**Q.1 Explain difference between ASP.NET and Classic ASP.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | Classic ASP | ASP.NET |
| 1 | ASP is **interpreted.** | ASP.NET is **Compiled**. |
| 2 | **Classic ASP** uses a technology called ADO to connect and work with databases. | ASP.NET uses the ADO.NET technology. |
| 3 | ASP has Mixed HTML and coding logic. | ASP.NET html and coding part are separated by code behind files. |
| 4 | ASP is partially object oriented. | ASP.NET purely objects oriented. |
| 5 | For ASP No in-built support for XML. | ASP.NET full XML Support for easy data exchange. |
| 6 | Very less development and debugging tool available. Meaning that difficult to debug the code. | Various tools and compiler available. Microsoft Visual studio makes your debugging job easier. |
| 7 | It has no high level programming structure. Mixed of html and server side scripting. | ASP.NET gives you three tire architecture. It allows you to keep your business logic, views everything separate. Meaning that easy to enhance applications. |
| 8 | Error handling is very poor. | Error handling is very good. |
| 9 | It has no in built validation control. Meaning that validating page is difficult for developers. | In built validation controls. It has rich validation set - custom validator, range validator, regular expression, compare and require field validation control which makes your job easier. |

**Q. Explain ASP.NET Application Life cycle.**

**ANS:**

**Step 1:** The user sends a request to IIS. IIS first checks which ISAPI extension can serve this request. Depending on file extension the request is processed. For instance, if the page is an ‘.ASPX page’, then it will be passed to ‘aspnet\_isapi.dll’ for processing.  
  
**Step 2:** If this is the first request to the website, then a class called as ‘ApplicationManager’ creates an application domain where the website can run. As we all know, the application domain creates isolation between two web applications hosted on the same IIS. So in case there is an issue in one app domain, it does not affect the other app domain.  
  
**Step 3:** The newly created application domain creates hosting environment, i.e. the ‘HttpRuntime’ object. Once the hosting environment is created, the necessary core ASP.NET objects like ‘HttpContext’, ‘HttpRequest’ and ‘HttpResponse’ objects are created.  
  
**Step 4:** Once all the core ASP.NET objects are created, ‘HttpApplication’ object is created to serve the request. In case you have a ‘*global.asax*’ file in your system, then the object of the ‘*global.asax*’ file will be created. Please note *global.asax* file inherits from ‘HttpApplication’ class.  
**Note**: The first time an ASP.NET page is attached to an application, a new instance of ‘HttpApplication’ is created. Said and done to maximize performance, HttpApplication instances might be reused for multiple requests.  
  
**Step 5:** The HttpApplication object is then assigned to the core ASP.NET objects to process the page.  
  
**Step 6:** HttpApplication then starts processing the request by HTTP module events, handlers and page events. It fires the MHPM event for request processing.

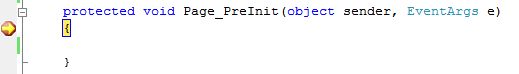
**Q. Explain ASP.NET Page Life cycle.**

**ANS**:

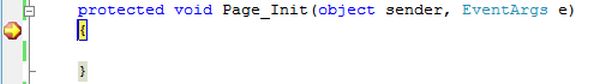
When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps.

The following are the page life cycle events in ASP.NET

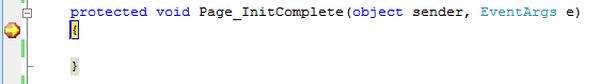
* **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page\_PreInit handler. In this create master pages dynamically.



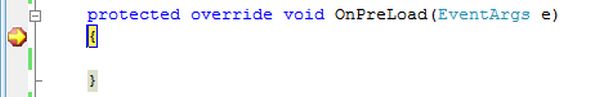
* **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page\_Init handler. This event fires after each control has been initialized. Each control's UniqueID is set and any skin settings have been applied.



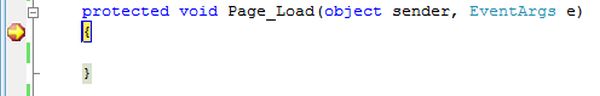
* **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking. Until now the viewstate values are not yet loaded, hence you can use this event to make changes to the view state that you want to ensure are persisted after the next postback. Use this event for processing tasks that require all initialization to be complete.



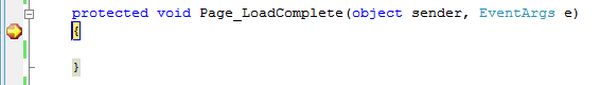
* **LoadViewState** - LoadViewState event allows loading view state information into the controls.
* **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
* **PreLoad** - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page\_PreLoad handler. Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.



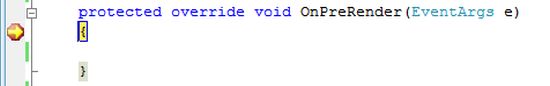
* **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page\_Load handler. This is the first place in the page lifecycle that all values are restored. You may also call Validate and check the value of IsValid in this method.



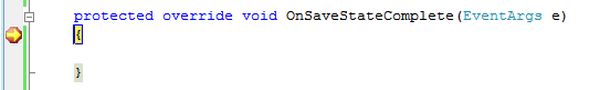
* **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page\_LoadComplete handler. Raised at the end of the event-handling stage.Use this event for tasks that require that all other controls on the page be loaded.



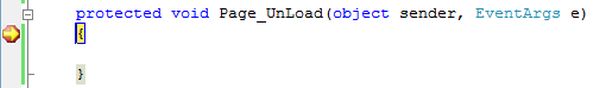
* **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered. Allow final changes to the page or its control.This event takes place before saving ViewState, so any changes made here are saved.For  example: After this event, you cannot change any property of a button or change any viewstate value.



* **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
* **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page\_Render handler.Raised after view state and control state have been saved for the page and for all controls.Before this event occurs, ViewState has been saved for the page and for all controls.



* **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page\_UnLoad handler. This event is used for cleanup code.At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object. Cleanup can be performed on: Instances of classes, in other words objects,Closing opened files,Closing database connections.



**Q.2 Explain different kinds of Validation Controls in C# with Example.**

**ANS**:

There are six types of validation controls in ASP.NET

1. RequiredFieldValidation Control
2. CompareValidator Control
3. RangeValidator Control
4. RegularExpressionValidator Control
5. CustomValidator Control
6. ValidationSummary

**1. RequiredFieldValidation Control –**

The RequiredFieldValidator control is simple validation control, which checks to see if the data is entered for the input control. You can have a RequiredFieldValidator control for each form element on which you wish to enforce Mandatory Field rule.

**Example-**

<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" Style="top: 98px;left: 367px; position: absolute; height: 26px; width: 162px" ErrorMessage="password required" ControlToValidate="TextBox2"></asp:RequiredFieldValidator>

**2. CompareValidator Control –**

The CompareValidator control allows you to make comparison to compare data entered in an input control with a constant value or a value in a different control. It can most commonly be used when you need to confirm password entered by the user at the registration time. The data is always case sensitive.

**Example-**

<asp:CompareValidator runat="server" id="cmpNumbers" controltovalidate="txtSmallNumber" controltocompare="txtBigNumber" operator="LessThan" type="Integer" errormessage="The first number should be smaller than the second number!" />

**3. RangeValidator Control –**

The RangeValidator Server Control is another validator control, which checks to see if a control value is within a valid range. The attributes that are necessary to this control are: MaximumValue, MinimumValue, and Type.

**Example-**

<asp:RangeValidator ID="RangeValidator1" runat="server" Style="top: 194px; left: 365px;  
position: absolute; height: 22px; width: 105px" ErrorMessage="RangeValidator" ControlToValidate="TextBox4" MaximumValue="100" MinimumValue="18" Type="Integer"></asp:RangeValidator>

**5. RegularExpressionValidator Control –**

A Regular expression is a powerful pattern matching language that can be used to identify simple and complex characters sequence that would otherwise require writing code to perform. It is used to validate complex expressions. These expressions can be phone number, email address, zip code and many more. Using Regular Expression Validator is very simple. Simply set the ValidationExpression property to any type of expression you want and it will validate it.

**Example-**

<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" Style="top: 234px;left: 366px; position: absolute; height: 22px; width: 177px"   
ErrorMessage="RegularExpressionValidator" ControlToValidate="TextBox5"   
ValidationExpression="\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*"></asp:RegularExpressionValidator>

**6. CustomValidator Control –**

The CustomValidator Control can be used on client side and server side. JavaScript is used to do client validation and you can use any .NET language to do server side validation.

**Example-**

|  |  |
| --- | --- |
|  | **Client Side code (Javascript):**  function TextBoxDCountyClient(sender, args) {  var v = document.getElementById('<%=TextBoxDTownCity.ClientID%>').value;  if (v == '') {  args.IsValid = false; // field is empty  }  else {  // do your other validation tests here...  }  }  **Server side code (C#):**  protected void TextBoxDTownCity\_Validate(object source, ServerValidateEventArgs args)  {  string v = TextBoxDTownCity.Text;  if (v == string.Empty)  {  args.IsValid = false; // field is empty  }  else  {  // do your other validation tests here...  }  } |

### 5. ValidationSummary Control –

ValidationSummary control allows us to display summary of all validation errors. We can display validation errors summary inline of a web page or a message box or both by using ShowMessageBox and Show Summary property value true or false. We can display validation messages as bulleted list, single paragraph or only list based on Display Mode.

**Q.3 What is HTTP Protocol.**

**ANS**:

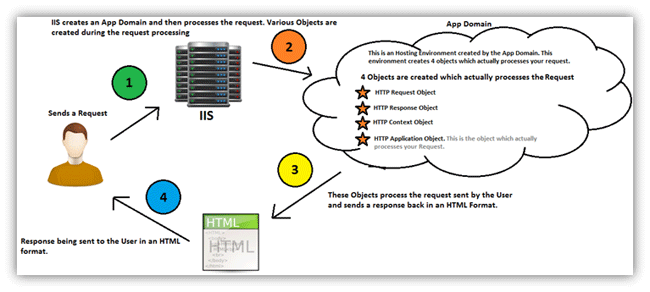
Hyper Text Transfer Protocol is the base or I would rather say, is the core part or the foundation of our World Wide Web. It basically makes use of the TCP Protocol for communicating with the server. A specific port is being used by this protocol while communicating with the server. HTTP provides various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

**Q.3 What happen when user sends request to server.**

**ANS**:

* Whenever the user sends a request to the server, the server first locates the page and checks that 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 send a response to the user in HTML format.



**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.

**Q.4 Explain State Management technique in ASP.NET with example.**

**ANS**:

**HTTP/HTTPS** is a **Stateless** Protocol.

**What does stateless actually mean?**

**Stateless** means, whenever we visit a website, our browser communicates with the respective server depending on our requested functionality or the request. The browser communicates with the respective server using the HTTP or HTTPs protocol.

But after that response, **what's next or what will happen when we visit that website again** after closing our web browser?

In this case HTTP/HTTPs doesn't remember what website or URL we visited or in other words we can say it doesn't hold the state of a previous website that we visited before closing our browser, that is called **stateless**. So our browsers are **stateless**.

There are two types of State Management Techniques in ASP.NET

1. Client-Side State Management

2. Server-Side State Management

**1. 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) are not at all utilized during the process.

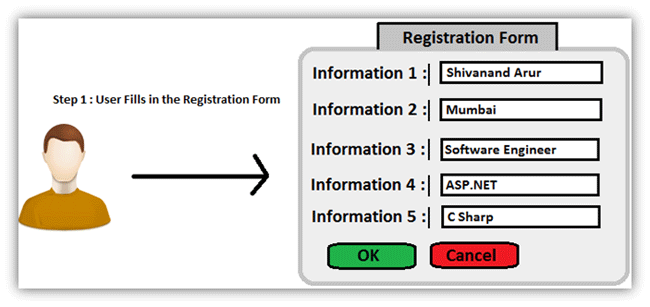
Client side state management techniques are:

* View State
* Hidden field
* Cookies
* Query Strings

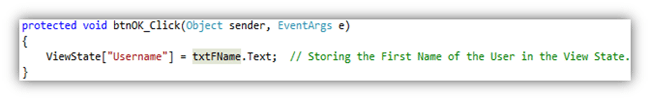
**A. ViewState**

View State is used for storing user data in ASP.NET. Sometimes in ASP.NET applications, users want to maintain or store their data temporarily after a post back. In this case, VIEW STATE is the most used and preferred way of doing this mechanism. 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. 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.

**Example** - A user fills in a Registration Form.

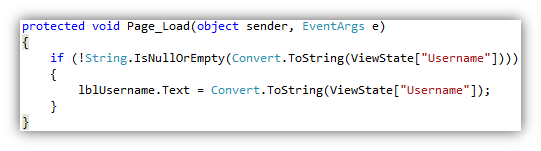


Storing Value in a View State



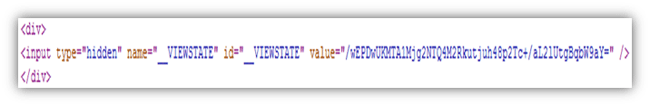
Here in the code above, "Username" is the key and the value is the text being inputted by the user (txtFName.Text).

Retrieving Value from a View State



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



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:

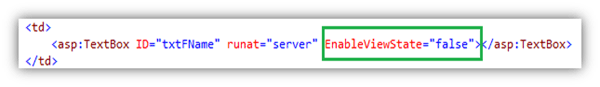
**1.Setting View State at Application Level** - If we do not want our pages in the Application to use view state, then we can disable or enable it in the web.config file, as shown in the image below.



**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.

View-State-Settings2.gif

**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:

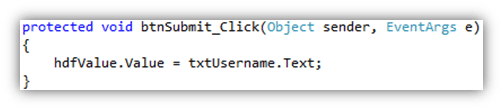


**B. Hidden Field -**

Like view state, hidden fields store data in an HTML form without displaying it in the user's browser. The data is available only when the form is processed. 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. Hidden Fields are similar to a text box but do not get displayed on the UI. However you can see the value of hidden fields when you view page source in the browser.

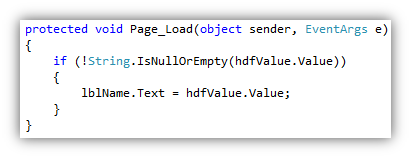
**Example** - Take the same example of the User filling an online registration form.

Setting Value to a Hidden Field



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



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.



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.

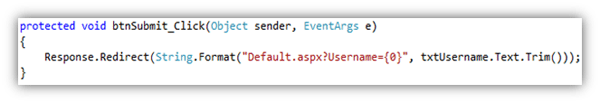
**Disadvantages**

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.

**C. Query String**

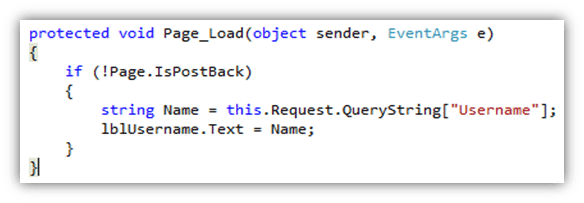
A Query String is a string which is appended to the end of the Page URL. It is very simple to use and can be used to send data across pages. It stores information in a key - value pair. A "?" signature is used to append the key and value to the page URL.

Way to pass a value using Query String



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



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 "&"delimiter.

**C. Cookies**

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, and Yahoo use cookies.

Cookie is a small text file that gets stored in users hard drive using client’s browser. Cookies are just used for the sake of user’s identity matching as it only stores information such as session’s ids, some frequent navigation or postback request objects.

Whenever we get connected to the internet for accessing particular service, that cookie file gets accessed from our hard drive via our browser for identifying user. The maximum size of a cookie file is 4 KB.

Create a cookie -> add a value to the **Response.Cookies HttpCookieCollection.**   
Read a cookie -> read values in **Request.Cookies**.

**Example-**

*int postbacks = 0;*

*if (****Request.Cookies["number"]*** *!= null)*

*{*

*postbacks = Convert.ToInt32(****Request.Cookies["number"].Value****) + 1;*

*}*

*// Generating Response*

*else*

*{*

*postbacks = 1;*

*}*

***Response.Cookies["number"].Value*** *= postbacks.ToString();*

*Result.Text = Response.Cookies["number"].Value;*

There are two types of Cookies –

##### **1. Persistent Cookie**

Cookie having expiration date is called persistent cookie. These types of cookies reach their end as their expiration dates comes to end. In this cookie, we set an expiration date.

**Example-**

*Response.Cookies["UserName"].Value = "Abhishek";*

*Response.Cookies["UserName"].Expires = DateTime.Now.AddDays(1);*

*HttpCookie aCookie = new HttpCookie("Session");*

*aCookie.Value = DateTime.Now.ToString();*

*aCookie.Expires = DateTime.Now.AddDays(1);*

*Response.Cookies.Add(aCookie);*

##### **1. Non-Persistent Cookie**

Non-Persistent cookies are temporary. They are also called in-memory cookies and session-based cookies. These cookies are active as long as the browser remains active, in other words if the browser is closed then the cookies automatically expire.

**Example –**

*HttpCookie aCookie = new HttpCookie("Session");*

*aCookie.Value = DateTime.Now.ToString();*

*aCookie.Expires = DateTime.Now.AddDays(1);*

*Response.Cookies.Add(aCookie);*

**2. 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.

Client side state management techniques are:

* Application State
* Session State

**A. Application State**

The Application object is an instance of the System.Web.HttpApplicationState class. You can create Application Level object like a name/value pair and share it across all users. You can declare Application level variable when the web application starts. Usually developers declare Application Level variables in Global.asax file. The Global.asax file contains events (Application\_Start Event, Application\_End Event and Application\_Error Event) which get associated with Application object. if multiple users access the site or application at the same time, then we can use an Application Object for such purposes.

**Example-**

*void Application\_Start(object sender, EventArgs e)*

*{*

*Application["CopyRightNote"] = "DotNetCurry CopyRight 2013-2014";*

*Application["HitCounter"] = 0;*

*}*

*void Session\_Start(object sender, EventArgs e)*

*{*

*Application.Lock();*

*Application["HitCounter"] =Convert.ToInt32(Application["HitCounter"]) + 1;*

*Application.Lock();*

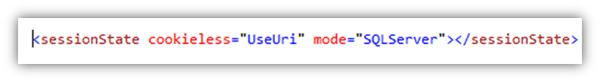
*}*

**B. Session State**

Session is one of the most common ways which is being used by developers to maintain the state of the application. Session objects are stored on the server side and are an instance of the System.Web.SessionState.HttpSessionState class. 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.



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

1. OFF
2. InProc
3. Out Proc

A.State Server

B.SQL Server

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

**2. InProc** - In-Process stores the session in memory on the web server. This is the default mode which is used in ASP.NET to store session variables.

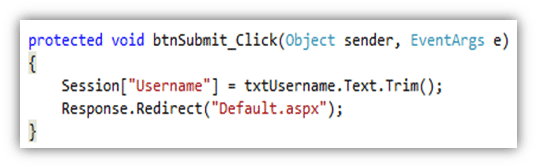
**3**. **OutProc** - OutProc means session will be stored on the server outside your application.

**A. State Server** -StateServer Stores session state in a service called the ASP.NET State Service. If we use this mode, then a separate Windows Service which runs in a different process stores the Session.

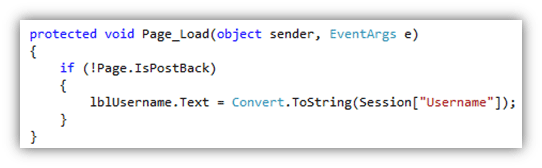
**A. SQL Server** - SQLServer Stores session state in a SQL Server database. This mode stores the Session in SQL Server instead of the server's memory.

**Example of Session -**

Code to write values into a Session



Code to read values from Session



**Q. What is the difference between Response.Redirect and Server.Transfer.**

**ANS:**

|  |  |  |
| --- | --- | --- |
| SR No | Response.Redirect | Server.Transfer |
| 1 | Response.Redirect() will send you to a new page, update the address bar and add it to the Browser History. On your browser you can click back. | Server.Transfer() does not change the address bar, we cannot hit back. One should use Server.Transfer() when he/she doesn’t want the user to see where he is going. Sometime on a "loading" type page. |
| 2 | It redirects the request to some plain HTML pages on our server or to some other web server. | It transfers current page request to another .aspx page on the same server. |
| 3 | It causes additional roundtrips to the server on each request. And hence puts extra load on server. Roundtrip is the combination of a request being sent to the server and response being sent back to browser. | It preserves server resources and avoids the unnecessary roundtrips to the server. |
| 4 | It enables to see the new redirected URL where it is redirected in the browser (and be able to bookmark it if it’s necessary). | It doesn’t show the real URL where it redirects the request in the users Web Browser. |
| 5 | ‘Response. Redirect’ sends message to the browser saying it to move to some different page, | ‘Server. Transfer’ does not send any message to the browser but rather it redirects the user directly from the server itself |
| 6 | With ‘Response. Redirect’ we can redirect the user to the both type of pages .html or .aspx e.g. Response. Redirect (“OtherPage.html”) OR Response. Redirect (“OtherPage.aspx”) | In case of ‘Server. Transfer’ we can redirect user to .asp or .aspx pages only e.g. Server. Transfer (“OtherPage.asp”) OR Server. Transfer (“OtherPage.aspx”) not to Server. Transfer (“OtherPage.html”). |
| 7 | When we want to allow our website’s URL can be copied then ‘Response. Redirect’ is better | But for security reasons ‘Server. Transfer’ is better because URL cannot be copied. |
| 8 | ‘Response. Redirect’ is a client process. | Server. Transfer’ is a server process |
| 9 | ‘Response. Redirect’ doesn’t preserve Query String and Form Variables from the original request. | Server.Transfer’ preserves Query String and Form Variables (optionally). |

**Q. What is catching in ASP.NET?**

**ANS:**

"A cache simply stores the output generated by a page in the memory and this saved output (cache) will serve us (users) in the future. **“Catching** can cache [store in memory] the output generated by a page and will serve this cached content for future requests. **Caching** is the technique of persisting data in memory for immediate access to the requesting program calls. Many in the developer community consider caching as one of the features available to improve performance of web applications.

**Q. Why we use Catching in ASP.NET?**

**ANS:**

**Scenario 1-**

Consider a page with a list of Employee names, contact numbers and mail-ids of existing employees of a company on an intranet accessible by all employees. This is very useful information that is available throughout the company and could also be one of the most accessed pages. The functionality of adding, updating or deleting is usually less intensive compared to more transaction-based systems like purchase orders, voucher creation and so on. Now in a normal scenario the process of querying the database for each request is not cost-effective in terms of server resources, hence it is much better to cache or persist the data to avoid this costly loss of resources. Any site is developed for a large number of users, in other words for the same request to work the same for a large number of clients so it is hard to provide the best performance for the server so we use caching.

**Scenario 2-**

Let us imagine a scenario when the contents of a web page are being pulled from a database. The user asks for contents depending on some criteria. Now if the database is getting changed very frequently that even between two requests of same user, we anticipate database change, then we can in no way cache the data that the user is requesting. But if the database is not getting changed that frequently, we can have some caching in place so that if the user is requesting the same data very frequently, we don't hit the database every time (since we know contents are not changed).The two keys terms here are frequency and criteria. Frequency is the number of times we are anticipating the user requests for a particular page and criteria is what governs the uniqueness of result that is being displayed on the page. Frequency is important because we need to figure out the interval in which database is changing and compare it with the frequency of user requests so that we can have caching in place and also make sure that user is not viewing outdated data. Criteria are important because we need to make sure that we have caching implemented for the page on every unique criterion. It should not be the case that user is requesting contents based on criteria01 and we are showing him the cached results of criteria00 (cached earlier for him).

**Q. Explain type of Catching in ASP.NET with Example?**

**ANS:**

There are three types of Catching in asp.net

1. Page Catching (Output)

2. Fragment Catching

3. Data Catching

**1. Page Catching (Output)-**

To cache an entire page's output we need to specify a directive at the top of our page, this directive is the **@OutputCache**. This is at the page level and one of the easiest means for caching pages. This requires one to specify the duration of the cache and the attribute of caching.

Let's figure out a simple demo of it.

**Syntax**: <%@ OutputCache Duration="60" VaryByParam="none" %>

The preceding syntax specifies that the page be cached for 60 seconds and the value "none" for the VaryByParam\* attribute ensures that there is a single cached page available for this duration specified.

**Duration Attribute –**

This attributes represents the time in seconds of how long the output cache should be stored in memory. After the defined duration the content stored in the memory will be cleared automatically.

**VarByParam Attribute –**

This is the most important attributes; you can't afford to miss that in the Output Cache directory statement. It generally defines the query string parameters to vary the cache (in memory). You can also multiple parameter names too, but for that you need to separate them using a semicolon (;).You can also specify it as "\*". In this case the cached content is varied for the entire parameters end using the querysrting.

**2. Fragment Catching-**

Caching of the entire page is not good because some part of the page is common for the entire application. So Fragmentation Catching is used for some portion of the page to be cached and we can do Fragmentation Caching using a User Control. In some scenarios we only need to cache only a segment of a page. For example a contact us page in a main page will be the same for all the users and for that there is no need to cache the entire page. So for that we prefer to use fragment caching option.

**For example:**

<%@ OutputCache Duration = 10 VaryByParam = "None" %>

Or:

<%@ OutputCache Duration = 5 VaryByParam = "None" VaryByCustom = "Browser" %>

**2. Data Catching-**

ASP.NET also supports caching of data as objects. We can store objects in memory and use them across various pages in our application. This feature is implemented using the Cache class. This cache has a lifetime equivalent to that of the application. Objects can be stored as name value pairs in the cache. A string value can be inserted into the cache as follows:

Cache["name"]="Smitha";

The stored string value can be retrieved like this:

if (Cache["name"] != null)

Label1.Text= Cache["name"].ToString();

To insert objects into the cache, the Add method or different versions of the Insert method of the Cache class can be used. These methods allow us to use the more powerful features provided by the Cache class. One of the overloads of the Insert method is used as follows:

Cache.Insert("Name", strName,

new CacheDependency(Server.MapPath("name.txt"),

DateTime.Now.AddMinutes(2), TimeSpan.Zero);

The first two parameters are the key and the object to be inserted. The third parameter is of type Cache Dependency and helps us set a dependency of this value to the file named *name.txt*. So whenever this files changes, the value in the cache is removed. We can specify null to indicate no dependency. The fourth parameter specifies the time at which the value should be removed from cache. [See example 5 for an illustration.] The last parameter is the *sliding expiration parameter* which shows the time interval after which the item is to be removed from the cache after its last accessed time.

**Q. What is Web config, App config and Machine Config file**

**ANS:**

**1. Web Config –**

It is a configuration file, which is used in web application and it can be an ASP.NET project or MVC project. Some project contains multiple web.config file inside the same project but with different folder. They have their unique benefits. You can create several web.config file in each folder with their unique benefits as per your project requirement. It is used to store the application level configuration. Sometimes it inherits the setting from the machine.config. It parses at runtime, means if you make any changes then web application will load all the settings in the config file. You don’t need to create a web.config file because it is auto generated when you create a new web application. If you want to create a web.config manually you can create it.

**Example –**

1. <?xml version="1.0" encoding="utf-8" ?>
2. <configuration>
3. <system.web>
4. </system.web>
5. </configuration>

**2. App Config –**

It is also a special type of configuration file which is basically used with Windows Services, Windows application, Console Apps or it can be WPF application or any others.  
It parses at compile time; it means if you edit the app.config when program is running, then you need to restart the application to reload the configuration setting into the program.  
When you run the application which contains the app.config, at the time of compilation a copy of app.config with different name moves into build folder for running the application, So that's why we need to restart the program if any changes made in app.config.  
It is not added auto when you create a project, to add this you need to go to solution explorer and choose Add New Item and choose “**Application Configuration File**”. Windows application always contains the App.config file into the project.

**Example –**

1. <?xml version="1.0"?>
2. <configuration>
3. <connectionStrings>
4. <add name="MyKey"
5. connectionString="Data Source=localhost;Initial Catalog=ABC;"
6. providerName="System.Data.SqlClient"/>
7. </connectionStrings>
8. </configuration>

**3. Machine Config –**

It is a special type of configuration file which creates into the OS when you install visual studio. This stores machine level configuration setting. Only one machine.config file exists into the system and it stores highest level of configuration settings.  
  
Machine.config settings apply to all web applications which is residing on the server. The setting of machine.config can be overridden by web.config’s settings. If your system does not contain the machine.config then you cannot execute the application.

**Path of Machine.config**

*32-bit System*

*%windir%\Microsoft.NET\Framework\[version]\config\machine.config*

*64-bit System*

*%windir%\Microsoft.NET\Framework64\[version]\config\machine.config*

**Sample Example**

1. <?xml version="1.0" encoding="UTF-8" ?>
2. <!--
3. Please refer to machine.config.comments for a description and
4. the default values of each configuration section.
6. For a full documentation of the schema please refer to
7. http://go.microsoft.com/fwlink/?LinkId=42127
9. To improve performance, machine.config should contain only those
10. settings that differ from their defaults.
11. -->
12. <configuration>
13. <configSections>
14. </configSections>
15. </configuration>

**Q. Explain difference between Machine.Config and Web.Config file.**

**ANS:**

|  |  |  |
| --- | --- | --- |
| SR No | Machine.Config | Web.Config |
| 1 | This is automatically installed when you install Visual Studio. Net. It exist exists in the c:\windows\microsoft.net\framework\version\config folder | This is automatically created when you create an ASP.Net web application project. |
| 2 | This is also called machine level configuration file | This is also called application level configuration file. |
| 3 | Only one machine.config file exists on a server. | There can be more than one web config file in single application |
| 4 | The machine.config file is the master configuration file on your system with a lot of default settings. | Web.config is the file for the local settings to be applied for a website which store configuration data in XML format. |
| 5 | The machine.config file file is at the highest level in the configuration hierarchy | Web.config file is to override the settings from the machine.config file. |
| 6 | The machine.config would be to share values between many applications on the server such as SMTP server | Web.config files contain application specific items such as database connection strings |
| 7 | If you make any changes to the web.config, web application will immediately load the changes | In case of machine.config you will have to restart the application. |
| 8 | Machine.config is configuration file for all the application in the IIS | Web.config is a configuration file for a particular application or folder. |

**Q. What are the event handlers that we can have in Global.asax file**

**ANS:**

The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET or by HttpModules.

The Global.asax file resides in the root directory of an ASP.NET-based application. The Global.asax file is parsed and dynamically compiled by ASP.NET.

The Global.asax file itself is configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.

The Global.asax file does not need recompilation if no changes have been made to it. There can be only one Global.asax file per application and it should be located in the application's root directory only.

The **Global.asax** contains two types of events those are -

1. Events which are fired for every request

2. Events which are not fired for every request

**1. Events which are fired for every request –**

**Application\_BeginRequest ()** – This event raised at the start of every request for the web application.

**Application\_AuthenticateRequest** – This event rose just before the user credentials are authenticated. We can specify our own authentication logic here to provide custom authentication.

**Application\_AuthorizeRequest()** – This event raised after successful completion of authentication with user’s credentials. This event is used to determine user permissions. You can use this method to give authorization rights to user.

**Application\_ResolveRequestCache()** – This event raised after completion of an authorization request and this event used in conjunction with output caching. With output caching, the rendered HTML of a page is reused without executing its code.

**Application\_AcquireRequestState()** – This event raised just before session-specific data is retrieved for the client and is used to populate Session Collection for current request.

**Application\_PreRequestHandlerExecute()** – This event called before the appropriate HTTP handler executes the request.

**Application\_PostRequestHandlerExecute()** – This event called just after the request is handled by its appropriate HTTP handler.

**Application\_ReleaseRequestState()** – This event raised when session specific information is about to serialized from the session collection.

**Application\_UpdateRequestCache()** – This event raised just before information is added to output cache of the page.

**Application\_EndRequest()** – This event raised at the end of each request right before the objects released.

**2. Events which are not fired for every request –**

**Application\_Start()** – This event raised when the application starts up and application domain is created.

**Session\_Start()** – This event raised for each time a new session begins, This is a good place to put code that is session-specific.

**Application\_Error()** – This event raised whenever an unhandled exception occurs in the application. This provides an opportunity to implement generic application-wide error handling.

**Session\_End()** – This event called when session of user ends.

**Application\_End()** – This event raised just before when web application ends.

**Application\_Disposed()** – This event fired after the web application is destroyed and this event is used to reclaim the memory it occupies.

**Q. Explain Cross Page Posting in asp.net with example.**

**ANS:**

Cross page postback is the concept which is used to submit one page (Default.aspx) controls to another page (Default2.aspx) and access those page (Default.aspx) control values in another page (Default2.aspx).

ASP.NET by default submit the forms to the same pages, cross page posting is submitted the form to a different page. This is usually required when you are creating a multipage form to collect information from the user on each page. when moving from the source to the target page.

"To use cross-page posting, you have to use the "postBackUrl" attribute to specify the page we want to post".

**Steps for Cross Page Posting -**  
  
**Step 1:** Add two webform "default.aspx" and "default2.aspx"   
  
**Step 2:** On default.aspx, drop down a button & set text property of this button as target page.  
  
Set the "postbackurl" property of a button to the url target page, default2.aspx.  
  
**Step 3:** On default2.aspx drop down a label.  
  
**Step 4:** In the page load event of default2.aspx  
  
We can access "**previouspage**" property to check if the page is bring accessed as cross page postback.

proctected void page\_load( object sender , eventArgs e)  
{  
     if( page.previouspage ! =null)  
    {  
    }  
}

**Step 5:** To redirect the value from the source page you must access control using the "findcontrol()" method of the the previouspage  
  
we will accessing the text propertyof the button control in "default.aspx"

proctected void page\_load( object sender , eventArgs e)  
{  
    if(page.previouspage !=null)  
   {  
        button btn = (button)(page.previouspage.findcontrol(button1));  
        lable1.text =btn.text;  
    }  
}

**Step 6:** Right click on default.aspx and set s start page

**Example-**

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Cross Page Postback Example in asp.net</title>

</head>

<body>

<form id="form1" runat="server">

<div>

<table>

<tr>

<td><b>Enter UserName:</b></td>

<td><asp:TextBox ID="txtUserName" runat="server"/></td>

</tr>

<tr>

<td><b>Enter Location:</b></td>

<td><asp:TextBox ID="txtLocation" runat="server"/></td>

</tr>

<tr>

<td></td>

<td><asp:Button ID="btnPostback" Text="Postback" runat="server" PostBackUrl="~/Default2.aspx" /></td>

</tr>

</table>

</div>

</form>

</body>

</html>

If you observe button control code I added property PostBackUrl="~/Default2.aspx" by using this property we will submit Default.aspx page control values to Default2.aspx page.

Now open Default2.aspx page and write the following code-

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Cross Page Postback Example in asp.net</title>

</head>

<body>

<form id="form1" runat="server">

<div>

<b><u>Default2.aspx Page</u></b><br /><br />

<label id="lblName" runat="server" /><br /><br />

<label id="lblLocation" runat="server" />

</div>

</form>

</body>

</html>

Now open **Default2.aspx** code behind file and add following namespaces

using System;

using System.Web.UI.WebControls;

Once namespaces added write the following code-

protected void Page\_Load(object sender, EventArgs e)

{

if (PreviousPage != null && PreviousPage.IsCrossPagePostBack)

{

TextBox txtName = (TextBox)PreviousPage.FindControl("txtUserName");

TextBox txtLocation = (TextBox)PreviousPage.FindControl("txtLocation");

lblName.InnerText = "Welcome to Default2.aspx page " + txtName.Text;

lblLocation.InnerText = "Your Location: " + txtLocation.Text;

}

else

{ss

Response.Redirect("Default.aspx");

}

}

**Q. What are Postback, IsPostBack and AutoPostBack in ASP.NET.**

**ANS:**

**Postback –**

**PostBack** is the name given to the process of submitting an ASP.NET page to the server for processing. **Postback** is actually sending all the information from client to web server, then web server process all those contents and returns back to the client.

**PostBack** is an event that is triggered when an action is performed by an ASP.Net Control.  For example when we click on an asp button, the data on the page is posted back to the server for processing. **PostBack** is the name given to the process of submitting an ASP.NET page to the server for processing.

**IsPostBack-**

**IsPostback** is normally used on page \_load event to detect whether page is going to reload(i.e postback or refresh) due to any asp control  of page.

**IsPostBack** having a boolean value; thus, the first time that the page loads the **IsPostBack** flag is false. Each time a PostBack occurs, the entire page including the Page\_Load is ‘posted back ‘and executed.

is used to check that a page is submitted to server first time or second time rendering, if code are writen  in **IsPostBack**  block,

Example-

This code will execute only once when this page runs first time. On any other submit event this page will be submitted to server but this block will not execute.

|  |
| --- |
| IsPostBack ----------------1st Time -------------------------------False  IsPostBack----------------2nd Time--------------------------------True  !IsPostBack----------------1st  Time-------------------------------True  !IsPsotBack-----------------2nd Time------------------------------ False |

**AutoPostBack -**

**Autopostback** is the mechanism, by which the page will be posted   
back to the server automatically based on some events in the web controls. In some of the web controls, the property called auto post back, which if set to true, will send the request to the server when an event happens in the control.

**How it is happening?**

Whenever we set **Autopostback** attribute to true in any of the controls, the .net framework will automatically insert few code in to the HTML generated to implement this functionality.

These are the additional items that the framework will inject to the HTML source for implementing **Autopostback** event.

a. A Javascript method with name \_\_doPostBack (eventtarget, eventargument)  
b. Two Hidden variables with name \_\_EVENTTARGET and \_\_EVENTARGUMENT  
c. OnChange JavaScript event to the control

**Q. Explain Tracing in ASP.NET.**

**ANS:**

**Tracing** is an activity to follow execution path and display the diagnostic information related to a specific Asp.Net web page or application that is being executed on the web server. Tracing can be enabled at development environment as well as in the production environment. This information can help you to investigate errors or unwanted results while ASP.NET processes a page request. You can view trace information at the bottom of individual pages and also you can use the trace viewer to view these trace information that is collected and cached by ASP.NET when tracing is enabled.

In Asp.Net Tracing is disabled by default. Trace statements are executed and shown only when tracing is enabled.

There are two types of tracing:

**(i) Page Level Tracing**

**(ii) Application Level Tracing**

**1. Page Level Tracing –**

We can control whether tracing is enabled or disabled for individual pages. If tracing is enabled, when the page is requested, ASP.NET appends to the page a series of tables containing execution details about the page request. Tracing is disabled by default in an ASP.NET application.

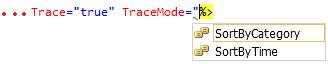
To enable Page Level Tracing follow the steps:

**Step 1:** Include Trace="true" in <%@ Page Title="" Language="C#"...%> directive, for example:

*<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true" CodeFile="Default.aspx.cs"Inherits="chat\_Default" Trace="true"%>*

Look at the above code, I'll be using Trace=true at the end.

**Step2:** Optionally,we can use TraceMode attribute in above <%@ Page Title="" Language="C#"...%> directive, for example:



SortByCategory: Set TraceMode to SortByTime to sort trace messages in the order in which they are processed.

SortByTime: Set TraceMode to SortByCategory to sort trace messages by the categories.

**2. Application Level Tracing –**

When we enable application level tracing, trace information is gathered and processed for each page in that application. We can enable application level tracing by using the trace element in the Web.config file.

Instead of enabling tracing for individual pages, you can enable it for your entire application. In that case, every page in your application displays trace information. Application tracing is useful when you are developing an application because you can easily enable it and disable it without editing individual pages. When your application is complete, you can turn off tracing for all pages at once.

 When you enable tracing for an application, ASP.NET collects trace information for each request to the application, up to the maximum number of requests you specify. The default number of requests is 10. You can view trace information with the trace viewer.

 By default, when the trace viewer reaches its request limit, the application stops storing trace requests. However, you can configure application-level tracing to always store the most recent tracing data, discarding the oldest data when the maximum number of requests is reached.

**Example –**

*<configuration>*

*<system.web>*

*<trace enabled="true" pageOutput="true" requestLimit="50"*

*localOnly="false" mostRecent="true"traceMode="SortByTime" />*

*</system.web>*

*</configuration>*

**Q. Explain difference between HTML control and ASP.NET Server Control.**

**ANS:**

|  |  |  |
| --- | --- | --- |
| SR No | HTML Control | ASP.NET Server Control |
| 1 | HTML control runs at **client side.** | ASP.Net controls run at **server side**. |
| 2 | You can run HTML controls at server side by adding attribute **runat=”server”**. | You cannot run ASP.Net Controls on client side as these controls have this attribute **runat=”server”** by default**.** |
| 3 | HTML controls are client side controls, so it does not provide **STATE management.** | ASP.Net Controls are Server side controls, provides **STATE management**. |
| 4 | HTML control does not require rendering. | ASP.Net controls require **rendering**. |
| 5 | As HTML controls runs on client side, execution is **fast**. | As ASP.Net controls run on server side, execution is **slow**. |
| 6 | HTML controls do not have any separate **class** for its controls. | ASP.Net controls have separate class for its each control. |
| 7 | HTML controls does not support **Object Oriented paradigm**. | With ASP.Net controls, you have full support of Object oriented paradigm. |
| 8 | HTML controls cannot be accessed form code behind files. | ASP.Net controls can be directly worked and accessed from **code behind files**. |
| 9 | HTML controls have **limited** set of properties and/or methods. | ASP.Net controls have **rich** set of properties and/or methods. |

**Q. Explain Authentication and Types of Authentication in ASP.NET**

**ANS:**

**Authentication** is the process of obtaining some sort of credentials from the users and using those credentials to verify the user's identity. **Authentication** is always precedes to Authorization. ASP.NET **authentication** means to identify the user or in other words its nothing but to validate that he exists in your database.

**Authentication**: Authentication is the process of determining the identity of a user based on the user’s credentials. The user’s credentials are usually in the form of user ID and password, which is checked against any credentials' store such as database. If the credentials provided by the user are valid, then the user is considered an authenticated user.

There are following types of Authentication in ASP.NET:

1. Windows Authentication

2. Forms Authentication

3. Passport Authentication

4. Anonymous Authentication

**1. Windows Authentication –**

Windows Authentication mode provides the developer to authenticate a user based on Windows user accounts. This is the default authentication mode provided by ASP.Net. You can easily get the Identity of the user by using User.Identity.Name. This will return the computer name along with the user name. Windows authentication also provides IsInRole method to find the role of the user and then you can give permissions to the user depending on the role. When you configure your ASP.NET application as windows authentication it will use local windows user and groups to do authentication and authorization for your ASP.NET pages.

Windows authentication is used for intranet web applications, where users are part of a windows domain-based network.

**Why Windows Authentication:**

1. Windows authentication is generally used if the users accessing the application belong to same organization.
2. This authentication method uses Windows accounts for validating users' credentials. This type of authentication is very good for intranet Web sites where we know our users.

**Example**- <system.web>

<authentication mode="Windows"/>

</system.web>

**2. Forms Authentication –**

**Form authentication** is used for internet web application. The advantage of form authentication is that users do not have to be member of a domain-based network to have access to your application. So the number of web application uses the form authentication in their web application. It is also known as cookie-based authentication. Many internet websites like Gmail,Amazon,Facebook etc. uses forms authentication. To access these applications we do not have to be member of their domain-based network.

Forms authentication classes are present in System.Web.Security.

**Example**- <system.web>

<authentication mode="Forms"/>

</system.web>

**3. Passport Authentication –**

**Passport authentication** can be used whenever you are using a single username and password combination to authenticate in to a group of website. The simplest example that i can give is that of a Google Gmail account. With a single emailid password combination u are able to access YouTube, Gmail, Google, blogger and most of the Google web applications. Passport authentication is a centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites.

**Passport authentication** identifies a user with using his or her e-mail address and a password and a single Passport account can be used with many different Web sites. **Passport authentication** is primarily used for public Web sites with thousands of users.

**Example**- <system.web>

<authentication mode="Passport"/>

</system.web>

**4. Anonymous Authentication –**

**Anonymous authentication** allows users to access the public areas of the web sites, without promoting the users for username and password. By default, the IUSR account, which was introduced in IIS 7.0 and replaces the IIS 6.0 IUSR\_computername account, is used to allow anonymous access. An application is a grouping of files that delivers content or provides services over protocols, such as HTTP. When you create an application in IIS, the application's path becomes part of the site's URL. By default, IIS 7 uses Anonymous authentication. You must disable **Anonymous authentication** for any Web site, Web application, or Web service for which you want to enable other authentication methods such as Basic or Windows authentication.

**Example**- <system.web>

<authentication mode="None"/>

</system.web>

**Q. Explain Impersonation in asp.net.**

**ANS:**

**Impersonation** is the process of executing code in the context of another user identity. By default, all ASP.NET code is executed using a fixed machine-specific account. To execute code using another identity we can use the built-in impersonation capabilities of ASP.NET. We can use a predefined user account or user's identity, if the user has already been authenticated using a windows account.

We can use the impersonation in these two scenarios:

1. To give each web application different permissions.
2. To use existing Windows user permission.

These two scenarios are fundamentally different. In the first one, impersonation defines a single, specific account. In this case, no matter what user access the application, and no matter what type of user-level security you use, the code will run under the account you've set. In the second one, the user must be authenticated by IIS. The web-page code will then execute under the identity of the appropriate user.

**Implement Impersonation:**

***Impersonate the Microsoft IIS Authenticated Account or User:***  
To impersonate the IIS authenticating user on every request for every page in an ASP.NET application, we must include an <identity> tag in the Web.config file of this application and set the impersonate attribute to true.

<identity impersonate="true" />

***Impersonate a Specific User:***  
To impersonate a specific user for all the requests on all pages of an ASP.NET application, you can specify the userName and password attributes in the <identity> tag of the Web.config file for that application.

<identity impersonate="true" userName="AccountNAME" password="Password" />

**Q. Explain Authorization in asp.net.**

**ANS:**

Authorization is the process of allowing an authenticated user to access the resources by checking whether the user has access rights to the system. Authorization helps you to control access rights by granting or denying specific permissions to an authenticated user.

Authorization is process of allowing or denying particular resources to user or role in asp.net. Authorization is the process of verifying if the user has rights/permission to access certain resources or sections of a website.

**Example –**

*<configuration>*

*<location path="Registration.aspx"> <!-- Path of your Registration.aspx page -->*

*<system.web>*

*<authorization>*

*<allow users="SureshDasari"/>*

*<deny users="\*"/> <!—deny all other users -->*

*</authorization>*

*</system.web>*

*</location>*

*</configuration>*

**Q. Explain Globalization and Localization in asp.net.**

**ANS:**

**Globalization** is the process of designing the application in such a way that it can be used by users from across the globe (multiple cultures).

**Localization** is the process of customizing your application for a given culture and locale.Localization is a process of designing the application so that it can display contents in the language of the user.

**ASP.Net Global and Local Resources**

ASP.Net has two set of Resource files i.e. Global and Local Resource files.

**Global Resources**

These resource files can be accessed on all pages throughout the applications.

**Local Resources**

These resource files are unique to each page and can be used only on that particular page.

**Q. Explain Page directive in asp.net.**

**ANS:**

Basically Page Directives are commands. These commands are used by the compiler when the page is compiled. ASP.NET Page Directives are instructions to specify optional setting, such as registering a custom control and page language. These settings are used to describe how ASP.NET web page (.aspx) or user control (ascx) page are processed by .NET Framework.

**How to use the directives in an ASP.NET page-**  
It is not difficult to add a directive to an ASP.NET page. It is simple to add directives to an ASP.NET page. You can write directives in the following format:  
  
<%@[Directive][Attributes]%>  
  
See the directive format, it starts with "<%@" and ends with "%>". The best way is to put the directive at the top of your page. But you can put a directive anywhere in a page. One more thing, you can put more than one attribute in a single directive.  
  
Here is the full list of directives:

* @Page
* @Master
* @Control
* @Import
* @Implements
* @Register
* @Assembly
* @MasterType
* @Output Cache
* @PreviousPageType
* @Reference

Let's discuss something about each directive.  
  
**@Page**  
when you want to specify the attributes for an ASP.NET page then you needs to use @Page Directive. As you know, an ASP.NET page is a very important part of ASP.NET, so this directive is commonly used in ASP.NET.  
  
Example:  
  
<%@Page Language="C#" AutoEventWIreup="false" CodeFile="Default.aspx.cs" Inherits="\_Default"%>  
  
**@Master**  
Now you have some information about @Page Directives. The @Master Directive is quite similar to the @Page Directive. The only difference is that the @master directive is for Master pages. You need to note that, while using the @Master Directive you define the template page's property. Then any content page can inherit all the properties defined in the Master Page. But there are some properties that are only available in a Master Page.  
  
Example  
  
<%@Master Language="C#" AutoEventWIreup="false" CodeFile="MasterPage1.master.cs" Inherits="MasterPage"%>  
  
**@Control**  
@Control builds ASP.NET user controls. When you use the directive you define the properties to be inherited by the user controls and theses values are assigned to the user controls  
  
Example:  
  
<%@Control Language="C#" Explicit="True" CodeFile="WebUserControl.ascx.cs" Inherits="WebUserControl" %>  
  
**@Import**  
As you know you need to define namespaces in your .cs class before using a C# or VB class. So the @Import Directive imports namespaces. This directive supports just a single attribute "namespace" and this attribute takes a string value that specifies the namespace to be imported. One thing you need to note is that the @Import Directive cannot contain more than one attribute/value pair. But you can use multiple lines.  
  
Example:  
  
<%@Import Namespace="System.Data"%>  
  
**@Implements**   
  
The @Implements Directive gets the ASP.NET pages to implement .Net framework interfaces. This directive only supports a single attribute interface.  
  
Example:  
  
<%@Implements Interface="System.Web.UI.IValidator"%>  
  
**@Register**  
When you create a user control and you drag that user control onto your page then you will see the @Register directive. This directive registers your user control on the page so that the control can be accessed by the page.  
  
Example:  
  
<%@ Register TagPrefix="MayTag Namespace="MyName.MyNameSpace" Assembly="MyAssembly"%>  
  
**@Assembly**   
  
The @Assembly Directive attaches assemblies to the page or an ASP.NET user control thereby all the assembly classes and interfaces are available to the class. This directive supports the two attributes Name and src. The Name attribute defines the assembly name and the src attribute defines the source of the assembly.  
  
Example:  
  
<%@Assembly Name="MyAssembly"%>  
<%@Assembly src="MYAssembly.cs">  
  
**@MasterType**  
The @MasterType Directive connects a class name to the ASP.NET page for getting strongly typed references or members contained in the specified Master Page. This directive supports the two attributes Typename and virtualpath. Typename sets the name of the derived class from which to get the strongly typed or reference members and virtualpath sets the location of the page from which these are retrieved.  
  
Example:  
  
<%@MasterType VirtualPath="/MasterPage1.master"%>  
  
**@output cache**  
It controls the output caching policies of an ASP.NET page.  
  
Example:  
  
<%@ OutputCache Duration ="180" VaryByParam="None"%>  
@Previouspagetype  
  
This directive specifies the page from which any cross-page posting originates.  
  
**@Reference**  
this directive declares that another page or user control should be complied along with the active page or control. This directive supports the single attribute virtualpath. It sets the location of the page or user control from which the active page will be referenced.  
  
Example:  
  
<%@Reference VirtualPath="~/MyControl.ascx"%>

**Q. Explain AutoEventWireUp in asp.net.**

**ANS:**

**AutoEventWireup** is a Boolean property which, when set to true then the page event handler methods are automatically wired with respective events. If this property set to false then the event handler methods need to be explicitly associated with their respective events.

The ASP.NET page framework supports an automatic way to associate page events and methods. If the **AutoEventWireup** attribute of the Page directive is set to true, the page framework calls page events automatically, specifically the Page\_Init and Page\_Load methods. In that case, no explicit Handles clause or delegate is needed.

* **AutoEventWireup** is an attribute in Page directive.
* **AutoEventWireup** is a Boolean attribute that indicates whether the ASP.NET pages events are auto-wired.
* **AutoEventWireup** will have a value true or false. By default it is true.

There is no event or method associated with Page\_Load. Those events whose inline event is not there but that should be executed, for that purposed AutoEventWireup="true".

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

**Q. Explain ScriptManager in asp.net.**

**ANS:**

**ScriptManager** control is the parent control that needs to be there on every page wherever we are trying to use ASP.NET AJAX controls. **ScriptManager** control manages client script for AJAX enabled ASP.NET pages. This control enables client script to use the type system extensions and support features for partial page rendering, web service calls etc. This control enables client script to use the type system extensions and support features for partial page rendering, web service calls etc.

**Why do we need the ScriptManager?**  
  
This **ScriptManager** control provides support for client-side AJAX features in an AJAX enabled web pages. It actually registers and loads the Microsoft AJAX library to enable the AJAX features.

Features of Script Manager are as follows:

1. Add a JavaScript proxy class for Web Services. This will help you to call the Web Service from JavaScript.  
     
     
     
   As shown above, when you register the Web Service reference, the Microsoft AJAX framework generates a proxy of the web service so that it can be accessed from JavaScript directly with the WebMethod name.
   1. function CallService() {
   2. WebService1.GetInfo("1", onSuccess, onFailure, null);
   3. }
   4. function onSuccess(result) {
   5. alert(result);
   6. }
   7. function onFailure(result) {
   8. alert("Error occured");
   9. }

This makes it easy for the developer to use a WebService call from JavaScript.

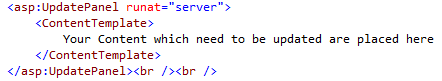
1. Registering and loading Custom Client Script.  
     
   The preceding figure shows how to add custom script files that you may need along with AJAX. You can add one or more .js files in a similar fashion.
2. Using Authentication, Profile and Role Services from Client Script  
     
   To enable authentication from a client script, you need to explicitly enable authentication service in the web config file. This will allow Microsoft AJAX to include proxy classes for ASP.NET 2.0 form authentication. Similarly for enabling a profile service and role service we need to add a respective tag as shown below.
   1. <system.web.extensions>
   2. <scripting>
   3. <webServices>
   4. <authenticationService enabled="true" />
   5. <profileService enabled="true" />
   6. <rolesService enabled="true" />
   7. </webServices>
   8. </scripting>
   9. </system.web.extensions>
3. Partial Page rendering is where a region of a page can be sent to the server and refreshed without a postback. This is a very important feature of AJAX. The ASP.NET AJAX [UpdatePanel](https://ajax.asp.net/Ajax/documentation/live/mref/T_System_Web_UI_UpdatePanel.aspx), [UpdateProgress](https://ajax.asp.net/Ajax/documentation/live/mref/T_System_Web_UI_UpdateProgress.aspx), and [Timer](https://ajax.asp.net/Ajax/documentation/live/mref/T_System_Web_UI_Timer.aspx) controls require a [ScriptManager](https://ajax.asp.net/Ajax/documentation/live/mref/T_System_Web_UI_ScriptManager.aspx) control to support partial-page rendering.

**Q. Explain UpdatePanel in asp.net.**

**ANS:**

ASP.NET **[UpdatePanel](https://ajax.asp.net/Ajax/Documentation/Live/mref/T_System_Web_UI_UpdatePanel.aspx)** controls enable you to build rich, client-centric Web applications. By using **[UpdatePanel](https://ajax.asp.net/Ajax/Documentation/Live/mref/T_System_Web_UI_UpdatePanel.aspx)** controls, you can refresh selected parts of the page instead of refreshing the whole page with a postback. This is referred to as performing a partial-page update. A Web page that contains a **[ScriptManager](https://ajax.asp.net/Ajax/Documentation/Live/mref/T_System_Web_UI_ScriptManager.aspx)** control and one or more **[UpdatePanel](https://ajax.asp.net/Ajax/Documentation/Live/mref/T_System_Web_UI_UpdatePanel.aspx)** controls can automatically participate in partial-page updates, without custom client script.

In the toolbox in the same group where the ScriptManager exists you may find one more control called UpdatePanel, as in:



**Properties and Methods of Update Panel are as follows:**

**ContentTemplate**: All the contents that must be updated asynchronously (means updated such that, other parts of the page remain untouched) are placed here. You can place any HTML content with any ASP.Net server control (like ListView) here.

**UpdateMode** and **ChildrenAsTriggers :** Decides when the contents will be updated. Possible values are:

* For UpdatePanel Always (Default), Conditional
* For ChildrenAsTriggers True (Default), False (only applicable when Update Mode is Conditional)

Let's discuss result with various combinations:

1. **UpdateMode Always** and **ChildrenAsTrigger True** : Contents will be refreshed on every full and partial Postback.
2. **UpdateMode Always** and **ChildrenAsTrigger True** : Exception: ChildrenAsTriggers cannot be set to false when UpdateMode is set to Always on UpdatePanel.
3. **UpdateMode Conditional** and **ChildrenAsTrigger** **True** : Contents will be refreshed:  
    On every full Postback

On Partial Postback caused by controls inside the same UpdatePanel.

1. **UpdateMode** **Conditional** and **ChildrenAsTrigger** **False**: Contents will be refreshed only on full Postback.
2. **Update** **method** : If at some point of time we want to explicitly refresh the UpdatePanel content, we can use this method.

**Q. Explain UpdatePanel trigger in asp.net.**

**ANS:**

**Triggers** : A Collection of **AsyncPostBackTrigger** and **PostBackTrigger**.

* **AsyncPostBackTrigger** : This property will take the two arguments Id of the control and an optional event and used when we want to refresh the UpdatePanel:
  1. On an event occurred by control present outside the UpdatePanel.
  2. on an event occurred by control present inside the UpdatPanel when the ChildrenAsTrigger is set to false.
  3. on an event occurred by control present inside the nested UpdatePanel.

* **PostBackTrigger** : Every control inside the UpdatePanel causes asynchronous PostBacks by default. But by using this property we can enable that control to cause a Postback instead of performing an asynchronous Postback.

**Q. Explain Master Page in asp.net.**

**ANS:**

ASP.NET Master Pages. ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behaviour that you want for all of the pages (or a group of pages) in your application. Master pages are templates which can be applied to your web pages to bring in consistent look and feel and uniform structure. By creating templates we also avoid lot of duplication of code across web pages.

**Q. What is ContenPlaceHolder in asp.net.**

**ANS:**

ContentPlaceHolder control is used to define a region for content in a Master Page, to hold the contents (asp: Content) of an associated Content control.  
On a master page we can have one or more ContentPlaceHolder controls. ContentPlaceHolder control defines a relative region for content in a master page, and renders all text, mark-up, and server controls from a related Content control found in a content page. A Content control is associated with a ContentPlaceHolder using its ContentPlaceHolderID property. Set the ContentPlaceHolderID property to the value of the ID property of the related ContentPlaceHolder control in a master page. More than one ContentPlaceHolder can be declared in a master page.Within a content page, only one Content control can supply the content for a ContentPlaceHolder in the master page. However, in each content page that uses a master page, you can have separate Content controls associated with the ContentPlaceHolder. For example, you can define the ContentPlaceHolder for the page title in a master page. For each content page that uses the master page, you can add a Content control that supplies the text and markup for the page title.

**Q. Can you dynamically assign a Master Page?**

**ANS:**

Yes, you can assign a master page dynamically during the PreInit stage using the Page class MasterPageFile property as shown in the code sample below.

void Page\_PreInit(Object sender, EventArgs e)  
{  
this.MasterPageFile = "~/MasterPage.master";  
}

**Q. Explain difference between GridView and Repeater in asp.net.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | GridView | Repeater |
| 1 | A GridView control is used when you want to display a set of data in a table format. | A Repeater is when you want to display data repeatedly, but not necessarily in a tabular format. |
| 2 | It was introduced with Asp.Net 2.0. | It was introduced with Asp.Net 1.0. |
| 3 | Automatically generates columns from the data source. | This feature is not supported. |
| 4 | Selection of row is supported. | Selection of row is not supported. |
| 5 | Editing of contents is supported. | Editing of contents is not supported. |
| 6 | Built-in Paging and Sorting is provided. | You need to write custom code. |
| 7 | Performance is very slow as compared to Repeater. | This is very light weight and fast data control among all the data control. |
| 8 | A GridView control displays data in Tabular format i.e. in Rows and Columns. | A Repeater control is like a container which fetches data from different sources and displays it in vertical or horizontal way. |

**Q. Explain difference between GridView and Repeater in asp.net.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | GridView | DataList |
| 1 | It was introduced with Asp.Net 2.0. | It was introduced with Asp.Net 1.0. |
| 2 | Built-in Paging and Sorting is provided. | You need to write custom code. |
| 3 | Built-in supports for Update and Delete operations. | Need to write code for implementing Update and Delete operations. |
| 4 | Supports auto format or style features. | This feature is not supported. |
| 5 | Repeat Direction property is not supported. | You can arrange data items horizontally or vertically in DataList by using property repeat Direction. |
| 6 | Doesn’t support customizable row separator. | Supports customizable row separator by using SeparatorTemplate. |
| 7 | Performance is slow as compared to DataList. | Performance is fast is compared to GridView. |

**Q. Explain difference between DataList and Repeater in asp.net.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | DataList | Repeater |
| 1 | Rendered as Table. | Template driven. |
| 2 | Automatically generates columns from the data source. | This feature is not supported. |
| 3 | Selection of row is supported. | Selection of row is not supported. |
| 4 | Editing of contents is supported. | Editing of contents is not supported. |
| 5 | You can arrange data items horizontally or vertically in DataList by using property Repeat Direction. | This feature is not supported. |
| 6 | Performance is very slow as compared to Repeater. | This is very light weight and fast data control among all the data control. |

**Q. Explain HttpHandler in asp.net.**

**ANS**:

**HTTP Handler** is the process which runs in response to a HTTP request. So whenever user requests a file it is processed by the handler based on the extension. So, custom http handlers are created when you need to special handling based on the file name extension. Let's consider an example to create RSS for a site. So, create a handler that generates RSS-formatted XML. Now bind the .rss extension to the custom handler. So from the above discussion, it is clear that **HTTPHandlers** are used by ASP.NET to handle the specific requests based on extensions.

HTTP handlers are the .NET components that implement the System.Web.IHttpHandler interface. Any class that implements the **IHttpHandler** interface can act as a target for the incoming HTTP requests. HTTP handlers are somewhat similar to ISAPI extensions. One difference between HTTP handlers and ISAPI extensions is that HTTP handlers can be called directly by using their file name in the URL, similar to ISAPI extensions.

HTTP handlers implement the following methods.

**ProcessRequest** - This method is actually the heart of all http handlers. This method is called to process http requests.

**IsReusable** - This property is called to determine whether this instance of http handler can be reused for fulfilling another request of the same type. HTTP handlers can return either true or false in order to specify whether they can be reused.

**Q. Explain HttpModule in asp.net.**

**ANS**:

HTTP Modules are plugged into the life cycle of a request. So when a request is processed it is passed through all the modules in the pipeline of the request. HTTP modules are .NET components that implement the System.Web.IHttpModule interface. These components plug themselves into the ASP.NET request processing pipeline by registering themselves for certain events. Whenever those events occur, ASP.NET invokes the interested HTTP modules so that the modules can play with the request. HttpModules take part in processing of the request by handling the Application events. There are number of events which you can handle during the HttpModule processing. For example - BeginRequest(), EndRequest(), AuthenticateRequest() etc.

An HTTP module is supposed to implement the following methods of the IHttpModule interface:

**Init** - This method allows an HTTP module to register its event handlers to the events in the HttpApplication object.

**Dispose** - This method gives HTTP module an opportunity to perform any clean up before the object gets garbage collected.

**Q. Explain difference between HttpHandler and HttpModule in asp.net.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | HttpHandler | HttpModule |
| 1 | **Meaning**:  An ASP.NET HTTP handler is the process (frequently referred to as the "endpoint") that runs in response to a request made to an ASP.NET Web application. The most common handler is an ASP.NET page handler that processes .aspx files. When users request an .aspx file, the request is processed by the page through the page handler. We can create our own HTTP handlers that render custom output to the browser.In order to create a Custom HTTP Handler,we need to Implement IHttpHandler interface(synchronous handler) or  Implement IHttpAsyncHandler(asynchronous handler). | **Meaning**:  An HTTP module is an assembly that is called on every request that is made to our application. HTTP modules are called as part of the ASP.NET request pipeline and have access to life-cycle events throughout the request. HTTP modules examine incoming and outgoing requests and take action based on the request. |
| 2 | **When to use HTTP handlers:**  **RSS feeds**: To create an RSS feed for a Web site, we can create a handler that emits RSS-formatted XML. We can then bind a file name extension such as .rss to the custom handler. When users send a request to your site that ends in .rss, ASP.NET calls our handler to process the request.  **Image server**: If we want a Web application to serve images in a variety of sizes, we can write a custom handler to resize images and then send them to the user as the handler’s response. | **When to use HTTP modules:**  **Security**: Because we can examine incoming requests, an HTTP module can perform custom authentication or other security checks before the requested page, XML Web service, or handler is called. In Internet Information Services (IIS) 7.0 running in Integrated mode, we can extend forms authentication to all content types in an application.  **Statistics and logging**: Because HTTP modules are called on every request; we can gather request statistics and log information in a centralized module, instead of in individual pages.  Custom headers or footers: Because we can modify the outgoing response, we can insert content such as custom header information into every page or XML Web service response. |
| 3 | **Number of HTTP handler called:**  During the processing of an http request, only one HTTP handler will be called. | **Number of HTTP module called:**  Whereas more than one HTTP modules may be called. |
| 4 | **Processing Sequence:**  In the asp.net request pipe line, http handler comes after http Module and it is the end point objects in ASP.NET pipeline. | **Processing Sequence:**  In the asp.net request pipe line, http Module comes first. |
| 5 | **What it does actually?**  HTTP Handler actually processes the request and produces the response | **What it does actually?**  HTTP module can work on request before and on the response after HTTP Handler |
| 6 | **HTTP Handler implement following Methods and Properties:**  **Process Request**: This method is called when processing asp.net requests. Here you can perform all the things related to processing request.  **IsReusable**: This property is to determine whether same instance of HTTP Handler can be used to fulfill another request of same type. | **Http Module implements following Methods and Properties:**  **InIt**: This method is used for implementing events of HTTP Modules in HTTPApplication object.  **Dispose**: This method is used perform clean-up before Garbage Collector destroy everything. |
| 7 | **Summary**:  If we need to create a request handler, for example we may want our own code to handle all .jpg image file requests like: http://mysite.com/filename.jpg, then we needs to use HttpHandlers for this purpose. | **Summary**:  If we want to modify a certain request, like we may want to perform some custom logic behind the scenes whenever user requests pages like mysite.com/default.aspx, we need to use HttpModules. We can create multiple HttpModules to filter any request. |
| 8 | **How to develop an ASP.NET handler:**  All we need is implementing IHttpHandler interface  public class MyHandler :IHttpHandler  {  public bool IsReusable  {  get { return false; }  }  public void ProcessRequest(HttpContext context)  {  }  } | **How to develop a Http Module:**  All we need is implementing System.Web.IHttpModule interface.  public class MyHttpModule : IHttpModule  {  public void Dispose()  {  }  public void Init(HttpApplication context)  {  //here we have to define handler for events such as BeginRequest ,PreRequestHandlerExecute ,EndRequest,AuthorizeRequest and ....  }  // you need to define event handlers here  } |