

“

# ASP.NET

---

## Da VíSue NoteBook

- Question Answer
- Exam Oriented
- Less Efficient Work
- Best Of Luck

”



Q. 1 Give the difference between the asp and asp.net.

Ans. There are several differences given below between asp and asp.net

ASP:

- 1) ASP is Interpreted language based on scripting languages like Jscript or VBScript.
- 2) ASP has Mixed HTML and coding logic.
- 3) Limited development and debugging tools available.
- 4) Limited OOPS support.
- 5) Limited session and application state management.
- 6) Poor Error handling system.
- 7) No in-built support for XML.
- 8) No fully distributed data source support.

ASP.NET:

- 1) ASP.Net is supported by compiler and has compiled language support.
- 2) Separate code and design logic possible.
- 3) Variety of compilers and tools available including the Visual studio.Net.
- 4) Completely Object Oriented.
- 5) Complete session and application state management.
- 6) Full proof error handling possible.
- 7) Full XML Support for easy data exchange.
- 8) Fully distributed data source support

Q.2 State and give the features of ASP.NET.

Ans. ASP.NET can be define and state as given below:

- ASP.NET is the name of the Microsoft technology used for web site development.
- ASP.NET is NOT a programming language like C# or VB.NET
- ASP.NET development requires a programming language like C# or VB.NET to write code.
- ASP stands for Active Server Pages.
- There are several other technologies exist for web development (Eg: PHP). ASP.NET is the technology from Microsoft and it he widely used on e.
- ASP.NET technology comes with a rich set of components and controls that make the web development very easy.
- Visual Studio .NET is the editor from Microsoft which helps you develop ASP.NET web sites faster and easily.
- IIS is the web server from Microsoft which supports ASP.NET. To develop ASP.NET web sites, you must have IIS installed in your computer.

There are several points can be given as the features of the asp.net which are:

- ✓ Better language support
- ✓ Programmable controls
- ✓ Event-driven programming
- ✓ XML-based components
- ✓ User authentication, with accounts and roles
- ✓ Higher scalability
- ✓ Increased performance - Compiled code
- ✓ Easier configuration and deployment
- ✓ Not fully ASP compatible

**ASP.NET Controls:** ASP.NET contains a large set of HTML controls. Almost all HTML elements on a page can be defined as ASP.NET control objects that can be controlled by scripts. ASP.NET also contains a new set of object-oriented input controls, like programmable list-boxes and validation controls. A new data grid control supports sorting, data paging, and everything you can expect from a dataset control.

**Event Aware Controls:** All ASP.NET objects on a Web page can expose events that can be processed by ASP.NET code. Load, Click and Change events handled by code makes coding much simpler and much better organized.

**ASP.NET Components:** ASP.NET components are heavily based on XML. Like the new AD Rotator, that uses XML to store advertisement information and configuration.

**User Authentication:** ASP.NET supports form-based user authentication, cookie management, and automatic redirecting of unauthorized logins.

**User Accounts and Roles:** ASP.NET allows user accounts and roles, to give each user (with a given role) access to different server code and executables.

**High Scalability:** Much has been done with ASP.NET to provide greater scalability.

Server-to-server communication has been greatly enhanced, making it possible to scale an application over several servers. One example of this is the ability to run XML parsers, XSL transformations and even resource hungry session objects on other servers.

**Compiled Code:** The first request for an ASP.NET page on the server will compile the ASP.NET code and keep a cached copy in memory. The result of this is greatly increased performance.

**Easy Configuration:** Configuration of ASP.NET is done with plain text files. Configuration files can be uploaded or changed while the application is running. No need to restart the server. No more metabase or registry puzzle.

**Easy Deployment:** No more server-restart to deploy or replace compiled code. ASP.NET simply redirects all new requests to the new code.

**Compatibility:** ASP.NET is not fully compatible with earlier versions of ASP, so most of the old ASP code will need some changes to run under ASP.NET. To overcome this problem, ASP.NET uses a new file extension ".aspx". This will make ASP.NET applications able to run side by side with standard ASP applications on the same server.

Q.3 Explain ASP.NET page directives.

**Ans.** Asp.Net Page directives are something that is a part of every asp.net pages. Page directives are instructions, inserted at the top of an ASP.NET page, to control the behavior of the asp.net pages. So it is type of mixed settings related to how a page should render and processed.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Sample.aspx.cs" Inherits="Sample" Title="Sample Page Title" %>
```

The @Page directive enables you to specify attributes and values for an Asp.Net Page to be used when the page is parsed and compiled. Every .aspx files should include this Page directive to execute. There are many attributes belong to this directive. We shall discuss some of the important attributes here.

The @Page directive enables you to specify attributes and values for an Asp.Net Page to be used when the page is parsed and compiled. Every .aspx files should include this @Page directive to execute. There are many attributes belong to this directive. We shall discuss some of the important attributes here.

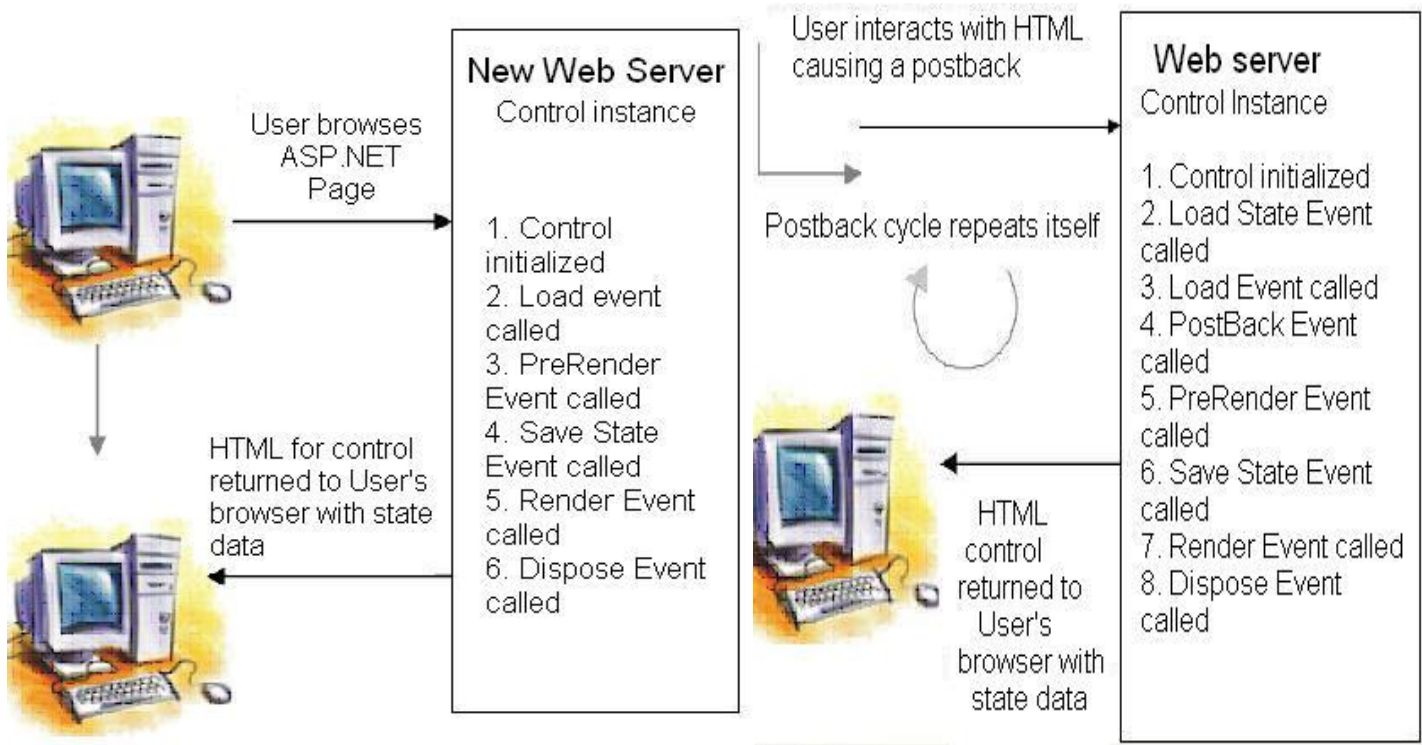
- a. AspCompat:** When set to True, this allows to the page to be executed on a single-threaded apartment. If you want to use a component developed in VB 6.0, you can set this value to True. But setting this attribute to true can cause your page's performance to degrade.
  - b. Language:** This attribute tells the compiler about the language being used in the code-behind. Values can represent any .NET-supported language, including Visual Basic, C#, or JScript .NET.
  - c. AutoEventWireup:** For every page there is an automatic way to bind the events to methods in the same .aspx file or in code behind. The default value is true.
  - d. CodeFile:** Specifies the code-behind file with which the page is associated.
  - e. Title:** To set the page title other than what is specified in the master page.
  - f. Culture:** Specifies the culture setting of the page. If you set to auto, enables the page to automatically detect the culture required for the page.
  - g. UICulture:** Specifies the UI culture setting to use for the page. Supports any valid UI culture value.
  - h. ValidateRequest:** Indicates whether request validation should occur. If set to true, request validation checks all input data against a hard-coded list of potentially dangerous values. If a match occurs, an HttpRequestValidationException Class is thrown. The default is true. This feature is enabled in the machine configuration file (Machine.config). You can disable it in your application configuration file (Web.config) or on the page by setting this attribute to false.
  - i. Theme:** To specify the theme for the page. This is a new feature available in Asp.Net 2.0.
  - j. SmartNavigation:** Indicates the smart navigation feature of the page. When set to True, this returns the postback to current position of the page. The default value is false.
  - k. MasterPageFile:** Specify the location of the MasterPage file to be used with the current Asp.Net page.
  - l. EnableViewState:** Indicates whether view state is maintained across page requests. true if view state is maintained; otherwise, false. The default is true.
  - m. ErrorPage:** Specifies a target URL for redirection if an unhandled page exception occurs.
  - n. Inherits:** Specifies a code-behind class for the page to inherit. This can be any class derived from the Page class.
- There are also other attributes which are of seldom use such as Buffer, CodePage, ClassName, EnableSessionState, Debug, Description, EnableTheming, EnableViewStateMac, TraceMode, WarningLevel, etc.

**Q.4 Explain ASP.NET page life cycle.**

Ans.

1. **OnInit** (Init) initializes each child control of the current
2. **LoadControlState**: Loads the ControlState of the control. To use this method the control must call the Page.RegisterRequiresControlState method in the OnInit method of the control.
3. **LoadViewState**: Loads the ViewState of the control.
4. **LoadPostData**: Is defined on interface IPostBackDataHandler. Controls that implement this interface use this method to retrieve the incoming form data and update the control's properties accordingly.
5. **Load (OnLoad)**: Allows actions that are common to every request to be placed here. Note that the control is stable at this time; it has been initialized and its state has been reconstructed.
6. **RaisePostDataChangedEvent**: Is defined on the interface IPostBackData-Handler. Controls that implement this interface use this event to raise change events in response to the Postback data changing between the current Postback and the previous Postback. For example if a TextBox has a TextChanged event and AutoPostback is turned off clicking a button causes the Text-Changed event to execute in this stage before handling the click event of the button which is raised in the next stage.
7. **RaisePostBackEvent**: Handles the client-side event that caused the Postback to occur
8. **PreRender** (OnPreRender): Allows last-minute changes to the control. This event takes place after all regular Post-back events have taken place. This event takes place before saving ViewState so any changes made here are saved.
9. **SaveControlState**: Saves the current control state to ViewState. After this stage any changes to the control state are lost. To use this method the control must call the Page.RegisterRequiresControlState method in the OnInit method of the control.
10. **SaveViewState**: Saves the current data state of the control to ViewState. After this stage any changes to the control data are lost.
11. **Render**: Generates the client-side HTML Dynamic Hypertext Markup Language (DHTML) and script that are necessary to properly display this control at the browser. In this stage any changes to the control are not persisted into ViewState.
12. **Dispose**: Accepts cleanup code. Releases any unman-aged resources in this stage. Unmanaged resources are resources that are not handled by the .NET common language runtime such as file handles and database connections.
13. **Unload**.





**Q.5 What is the main difference between the static and dynamic web page.**

Ans.

We can broadly classify web sites and web pages into two categories:

- 1. Static web pages
- 2. Dynamic web pages

**Static Web Pages**

A static web page is a page which has the same content always.

In case of static web pages, content is written in the page itself as plain html. Until the author of the web page updates the content, the content remains the same in the static pages. Static web pages are meant for providing information which does not change often. For example, visit <http://www.google.com/intl/en/about.html>. This page is a static page. The content is always the same (until they update the content by uploading a new html file to the web server). HTML files are used to create static web pages.

**Dynamic Web Pages**

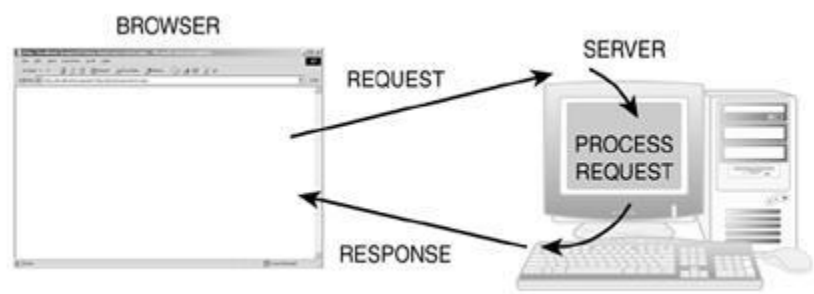
Dynamic web pages get content from database. Content is NOT hard-coded in the page itself.

Dynamic pages are created using "serverside code" when the page is loaded every time. An example for a dynamic page is this tutorial page itself. See the file name in the URL. The file name is "Tutorial8.aspx". We have used only one file to display any tutorial chapter. The chapter number is there as part of the URL (the chapter number is 8 for this chapter). When you type the URL in the browser, this page is dynamically created from database, based on the chapter id in the URL. The content of each chapter is stored in our database, not in the file itself. When you access the page, our server side code will check what is the TutorialId in the URL. Based on the TutorialId, it will retrieve appropriate chapter content from the database and dynamically create the web page. (You can try to change the TutorialId in the URL to some very large number and try. The page will give an error because our code will fail to get the corresponding chapter content from the database). This entire site has only very few files (like index.aspx, Tutorial.aspx etc) Dynamic web pages are created using technologies like ASP, ASP.NET, PHP etc. HTML pages cannot be dynamic. All HTML files are static pages. If you want to write dynamic pages, you must use some technologies like ASP.NET.

Q.6 Explain Request/Response Model of web processing.

Ans. A Web server is a computer that holds information about a Web site—its HTML pages, images, and so on. The client is the visitor to the Web site (specifically, the visitor's Web browser). Figure 1.1 illustrates this concept.

Figure 1.1. The request/response model.



Although this is a marvelous way to communicate and distribute information, it's rather simple and static. It can't provide any dynamic information or processing. The server simply waits around for someone to request information, and then it returns the data that's already stored on its hard drive without really looking at what it's sending. Generally, a static Web request follows these four steps:

- 1. The client (Web browser) locates a Web server through its URL (such as `www.microsoft.com`).
- 2. The client requests a page (such as `index.html`).
- 3. The server sends the requested document.
- 4. The client receives the document and displays it.

Once the client has received the information, the process is finished. The server has no idea what's happening on the client. How could it, since the server and client are two separate computers? They only communicate with one another during the request response process. Once the page has been delivered, the server doesn't care what happens.

Enter server processing. This comes in many forms, including the common gateway interface (CGI) and Microsoft's Active Server Pages, which is now referred to as classic ASP. In this scenario, the server takes a look at what it sends before it sends it, and it can take orders from the client. The server can return dynamic data, such as that from a database, calculations it performs, and anything else the client may ask for. The modified work flow is as follow s:

- 1. The client (Web browser) locates a Web server through its URL (such as `www.microsoft.com`).
- 2. The client requests a page (such as `index.html`).
- 3. The server examines the requested file and processes any code it contains.
- 4. The server translates the results of the processing to HTML (if necessary) and sends the requested document to the client.
- 5. The client receives the document and displays it.

Even in this scenario, the process is over once the client receives the page. The server has no idea what the client is doing unless it makes another request.

Q.7 Explain ASP.NET page structure.

Ans. ASP.NET pages are simply text files that have the `.aspx` file name extension, and can be placed on any web server equipped with ASP.NET. When a client requests an ASP.NET page, the web server passes the page to the ASP.NET runtime, a program that runs on the web server that's responsible for reading the page and compiling it into a .NET class. This class is then used to produce the HTML that's sent back to the user. Each subsequent request for this page avoids the compilation process: the .NET class can respond directly to the request, producing the page's HTML and sending it to the client, until such time as the `.aspx` files changes. This process is illustrated in Figure 2.1.

An ASP.NET page consists of the following elements:

- 1. directives
- 2. code declaration blocks
- 3. code render blocks
- 4. Web form
- 5. Web Server Controls

Directives

The directives section is one of the most important parts of an ASP.NET page. Directives control how a page is compiled, specify how a page is cached by web browsers, aid debugging (error-fixing), and allow you to import classes to use within your page's code. Each directive starts with <%@. This is followed by the directive name, plus any attributes and their corresponding values. The directive then ends with %>.

Code Declaration Blocks

However, if you're not working with code-behind pages, you must use code declaration blocks to contain all the application logic of your ASP.NET page. This application logic defines variables, subroutines, functions, and more. In our page, we've placed the code inside <script> tags, like so:

```
<script runat="server">
Sub mySub ()
'Code here
End Sub
</script>
```

Code Render Blocks

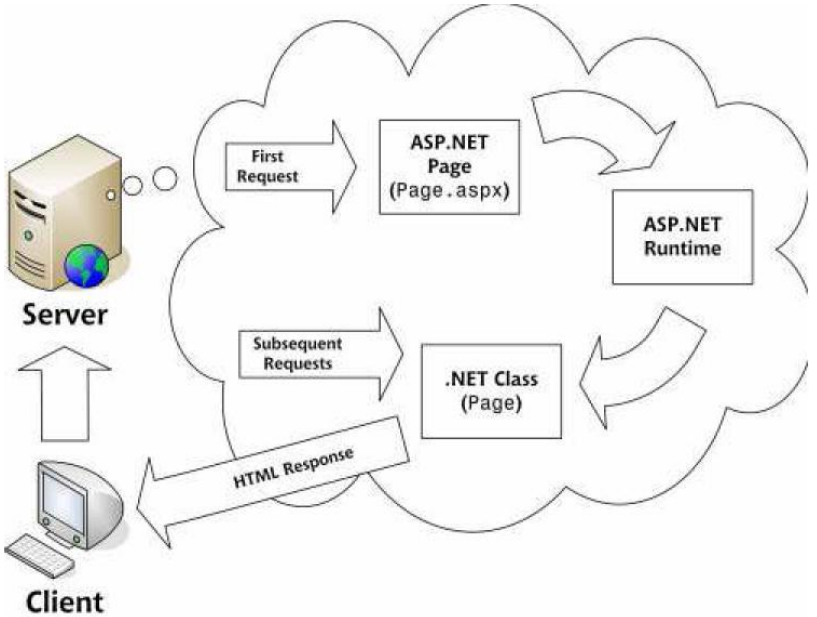
If you've had experience with traditional ASP, you might recognize these blocks. You can use code render blocks to define inline code or expressions that will execute when a page is rendered. Code within a code render block is executed immediately when it is encountered--usually when the page is loaded or rendered. On the other hand, code within a code declaration block (within <script> tags) is executed only when it is called or triggered by user or page interactions. There are two types of code render blocks? Inline code, and inline expressions--both of which are typically written within the body of the ASP.NET page.

WebForms:

Above Fig2.2 code shows form tage having attribute runat="Server" is called Web Form. Anything contained in a web form can be handling by asp.net

Web Server Controls

Above Fig.2.2 shows some of the label and textbox controls which are having attribute runat="sever" knows as Web Controls.



Q.8 What is the difference between client side code and server side code?

**Ans. Server side code:** When you request a page, the web server executes the code in the web page and generates HTML content for that page. It is this HTML content that is sent back to the browser so that it can be displayed to the user. The code that is executed by the web s server to generate the dynamic page is called "server side code". It is called "server side code" because it is executed by the web server. When you develop ASP.NET pages, you will use C# or VB.NET (or any other .NET compatible code) code to write the server side code. Server side code is used to retrieve and generate content for the dynamic pages. You may be using the code to retrieve the content from database or something like that.

When a dynamic page is requested, the server side code is executed on the server and a page is generated. Once the generated page comes to the browser, there is connection with the server. You cannot do anything from the browser to communicate with the server, then than requesting another page or same page again. So, if you want to access the database or something like that once the page is displayed, it is not possible.

**Client side code:** As you read above, it is not possible to access the server once the page is displayed. Moreover, the browser does not know anything about ASP.NET or .NET. The browser undersands only HTML and client side scripting languages. Client side coding is used to do basic operations on the browser. It cannot be used to access the server or database on the server etc. Client side coding is done using scripting languages like JavaScript, VbScript, and JScript etc. These scripting languages are easy to learn (they are different from vb.net and C#). The scripting languages provide only very minimal functionality. The main use of client side scripting is to validate user input before submitting a page to server. For example, suppose you have a "Registration" page. Once user enter all data and press the "submit" button, all the user input will be sent to server. In the server side, you may have written vb.net or C# code to validate all user inputs like Name cannot be empty etc. If the user do not enter a name and press submit, your server side code will generate an error message and return the page to you.

In this case, the page was sent to the server and came back with an error message. It was an expensive operation. It consumed lot of resources to send a page to the server and get it back to the browser with an error message. If we can catch this validation error in the browser itself, instead of sending to server and coming back with an error, that will save lot of time and resources. User need not wait to send the page to server and come back.

Q.9 What is the Postback and AutoPostBack Feature in ASP.NET.

Ans. Page object has an “IsPostBack” property which can be checked to know that is the page posted back.

If we want the control to automatically postback in case of any event, we will need to check this attribute as true. Example on a Combo Box change we need to send the event immediately to the server side then set the “AutoPostBack” attribute to true.

Q.10 Give the Difference Between Inline Code and Code Behind Model.

- Ans. To make the coding style user-friendly and keeping the old style ASP.NET provides two types of coding model.
- **Inline Code / Single page (Classical Model)**
  - **Code-Behind Model**

Inline Code

The coding style used by ASP developers is known as inline coding, inline code or in-page code because that was the only way to develop an ASP page. In ASP Pages scripting code working for generating desired output was intermixed with HTML code to create user-friendly pages as well as specific functionality was added to the web pages.

But with the latest ASP.NET introduction the old developers are moving from the old classical style of programming to the new style.

Code-Behind

The ASP.NET Framework has provided the way out to maintain the code for large web pages. Now you can design the HTML code page with .aspx extension separately and maintain the code files for the same .aspx page of .aspx.cs and .aspx.vb separately. This style of coding to develop web pages is called Code-Behind.

Code Behind approach is a better way to develop and design the .aspx page having basic layout of a web page containing all the necessary controls required for the GUI of the web page. Then include the C# or VB code behind class file for handling the events of controls. This mechanism separates the web page from design layout from the coding part.

Difference between Inline Code and Code-Behind:

Inline Code	Code Behind
1. The business logic is in <script> blocks in the same .aspx file that contains the HTML and controls.	1. The HTML and controls are in the .aspx file, and the business logic is in a separate .aspx.cs or .aspx.vb file.
2. When the page is deployed, the source code is deployed along with the Web Forms page, since it is physically in the .aspx file. Though, we are not able to see the code, only the results are rendered when the page runs.	2. All project class files (without the .aspx file ) are compiled into a .dll file, which are deployed to the <a href="#">server</a> Without any source code. When a request for the page is received, then an instance of the project .dll file is created and executed.
3. The .aspx file derives from the Page class.	3. The code for the page is compiled into a separate class from which the .aspx file derives.
4. When we write inline code we write code in the same page with Html code between scripting tags. So Each time when there is a request for page it compiles the code each time then server the page Like classic asp because inline code cannot create dll.	4. The code-behind approach also improved productivity (at some level) since the designer and the developer can continue working simultaneously on the same set of <a href="#">application</a> . It's also easier to build & test the UI and the business logic (DLL) - separately or combined.



Q.11 Explain controls collection of page class with example.

**Ans.** Every container control on the page, and the page itself, has a Controls collection that you can use to get to individual controls.

**To locate controls in the Controls collection:** Loop through the Controls collection of the container control. The collection is of type Control Collection, and returns objects of type Control.

The following example illustrates how to walk the Controls collection. The example assumes an ASP.NET Web page with at least one Textbox control on it, a Label control, and a Button control. The code gets all the child controls of the Page object. Because this would produce only a few high-level child controls, including the HtmlForm object, the code also walks the Controls collection of each individual child control. The code looks for text boxes by comparing the type of each control. When it finds a text box, it gets the text box's value and concatenates it into a string that is displayed at the end in a Label control.

This example finds only the controls contained in the Page object and the controls that are direct children of the page. It does not find text boxes that are children of a control that is in turn a child of the page. For example, if you added a Panel control to page, the Panel control would be a child of the HtmlForm control contained by the Page, and it would be found in this example. However, if you then added a Textbox control into the Panel control, the Textbox control text would not be displayed by the example, because it is not a child of the page or of a control that is a child of the page. A more practical application of walking the controls this way would be to create a recursive method that can be called to walk the Controls collection of each control as it is encountered. However, for clarity, the example below is not created as a recursive function.

**C# example:**

```
private void Button1_Click(object sender, System.EventArgs e)
{
    string allTextBoxValues = "";
    foreach (Control c in Page.Controls)
    {
        foreach (Control childc in c.Controls)
        {
            if (childc is TextBox)
            {
                allTextBoxValues += ((TextBox)childc).Text + ",";
            }
        }
    }
    if (allTextBoxValues != "")
    {
        Label1.Text = allTextBoxValues;
    }
}
```

Q.12 Explain the following classes with its members.

**Ans.** ASP.NET provides some Classes which make web development easier. Some of the classes used specifically for ASP.NET development are:

- System.Web.HttpRequest
- System.Web.HttpResponse
- System.Web.SessionState.HttpSessionState
- System.Web.ApplicationState

The above mentioned 4 objects are widely used in ASP.NET programming. It is important to learn these objects and their methods.

**Http Request:** In simple words, a web request is 'request sent from a client to the server, asking for a specific web page'. When you type a URL in a web browser or when you click on a hyper link in any web page, your browser is actually making a 'Request' to the server for the specific URL.

If you are using a browser to view a web page, then your browser is making the actual request to the server. You simply typed a URL in the browser, but the browser does lot of work in the background. It composes a Request in the proper format that the server can understand and sends the request to the server using the HTTP protocol.

The ASP.NET provides a class called HttpRequest which is defined in the namespace System.Web. This class provides various methods and properties which help you use various information related to a web request. An instance of this class is created by default in all the pages, so that you can use this object without creating again each time in all the pages. The name of this object is Request. In old ASP days, the Request object was the only way to retrieve the input data entered by the user in the input fields in the page. ASP.NET provides the event handling mechanism and web controls so that you can access user inputs in a more easy way.

In addition to that, a Request includes several others in formations including:

1. Details about the browser who makes the request (like version number, browser type etc)
2. Computer information like screen resolution, IP address of the user etc.
3. Cookies - the information stored in the client machine by the same web site.

4. Data inputted by the user (for example, when you register in a site, you are entering your registration details in the input fields. These details are sent to the server as part of the 'Request').

Commonly used Methods and Properties of Request object

- Request.FilePath
- Request.QueryString
- Request.Cookies
- Request.ServerVariables

**Http Response:** In the web server world, a Response is exactly opposite to the Request. A 'Response' is the message sent from the web server to the client, when client makes a 'Request'. For each request from a client, the server gives a response, unless there is an error. When you type a URL in a web browser or when you click on a hyper link in any web page, your browser makes a 'Request' to the server for the specific URL. Then servers process the request and send a response back.

The response includes several information about the page requested including the cookies to be saved on the client machine, the actual content to be displayed to the user etc. The browser accepts the response and processes it. Browser does several things including saving the cookies, checking the security etc and then displays the page content to the user. Note that displaying the page content to the user in the browser is only a small portion of the actual work.

The ASP.NET provides a class called `HttpResponse` which is defined in the namespace `System.Web`. This class provides various methods and properties which help you use various information related to a web response. An instance of this class is created by default in all the pages, so that you can use this object without creating again each time in all the pages. The name of this object is **Response**. In old ASP days, the Response object was the only way to write data to the page. ASP.NET provides the event handling mechanism and web controls so that you can write content to the page in an easier way.

- Response.Write
- Response.Cookies
- Response.Redirect

**Http Server Utility:** The last object today is the `HttpServerUtility` object, which provides several helper methods that are used in processing requests. You can use the name `Server` to access this object's members.

Redirecting Users

You've examined one way to redirect users with the `Response` object, but there are two other ways. The `Response.Redirect` method sends HTTP information to the browser, instructing it to go to another page. This requires a round trip that may not be necessary. Therefore, ASP.NET can also use the `Execute` and `Transfer` methods of the `HttpServerUtility` object to send users off to different pages. `Server.Transfer` simply transfers execution to another page. This doesn't require any information to be sent to the browser—it all occurs on the server without the user's knowledge. For instance, this code snippet examines a user's password (assuming he entered one in a form):

```
If (strPassword == "blahblah") _Server.Transfer("success.aspx")
```

If the password matches the string on line 1, you transfer him to another page. This is a valuable method for easily redirecting users. `Server.Execute` transfers ASP.NET execution to another page, but it returns to the original page when it's done. This is useful if you need to execute some code on another page (such as data processing, for instance) but continue on the same page. It works much like the branching logic we described yesterday, and follows the same format as `Server.Transfer`.

Formatting Strings

Typically, when you send output to the browser, it's interpreted as HTML. For example, the following prints a line break in the browser:  
`Response.Write("<br>")`  
However, there are times when you want the actual word to be printed. Instead of a line break, you want `<br>` to actually appear on the visitor's screen. To do this, you can use the `HtmlEncode` method of the `HttpServerUtility` object.

Controlling Scripts

The `HttpServerUtility` object provides one property to control the execution of ASP.NET scripts: `ScriptTimeout`. This value tells the server how long to wait before terminating a script or page. For example, the following tells ASP.NET that any code that has been executing for 90 seconds should be terminated:  
`Server.ScriptTimeout = 90`  
Occasionally, code you write may have a bug, such as an infinite loop, and may go on executing forever or until your server crashes. Using this property prevents that from happening. ASP.NET thinks any code that executes for longer than the `ScriptTimeout` period isn't working correctly and terminates it. On the other hand, occasionally some code really does take that long to execute (if it's very complex, for instance) and the `ScriptTimeout` property may be too short. Thus, you can easily extend the time.

**Q.13 Explain the advantage and disadvantages of ASP.NET Server Control.**

Ans. **Advantages:**

1. ASP .NET Server Controls can however detect the target browser's capabilities and render them accordingly. No issues for compatibility issues of Browsers i.e page that might be used by both HTML 3.2 and HTML 4.0 browsers code to be written by you.
2. Newer set of controls that can be used in the same manner as any HTML control like Calender controls. (No need of Activex Control for doing this which would then bring up issues of Browser compatibility).
3. Processing would be done at the server side. In built functionality to check for few values (with Validation controls) so no need to choose between scripting language which would be incompatible with few browsers.
4. ASP .NET Server Controls have an object model different from the traditional HTML and even provide a set of properties and methods that can change the outlook and behavior of the controls.
5. ASP .NET Server Controls have higher level of abstraction. An output of an ASP .NET server control can be the result of many HTML tags that combine together to produce that control and its events.

**Disadvantages:**

1. The control of the code is inbuilt with the web server controls so you have no much of direct control on these controls
2. Migration of ASP to any ASP.NET application is difficult. Its equivalent to rewriting your new application

**Q.14 Explain the advantage and disadvantages of HTML Server Control.**

Ans.

**HTML Server Controls**

**Advantages:**

1. The HTML Server Controls follow the HTML-centric object model. Model similar to HTML
2. Here the controls can be made to interact with Client side scripting. Processing would be done at client as well as server depending on your code.
3. Migration of the ASP project thought not very easy can be done by giving each intrinsic HTML control a `runat = server` to make it HTML Server side control.
4. The HTML Server Controls have no mechanism of identifying the capabilities of the client browser accessing the current page.
5. A HTML Server Control has similar abstraction with its corresponding HTML tag and offers no abstraction.

**Disadvantages:**

1. You would need to code for the browser compatibility.

**HTML Intrinsic Controls**

**Advantages:**

1. Model similar to HTML
2. Here the controls can be made to interact with Client side scripting

**Disadvantages:**

1. You would need to code for the browser compatibility
















**Q.15 List out the property of control class.**

Ans. There are several property of control class listed below:

- Adapter
- AppRelativeTemplateSourceDirectory
- BindingContainer
- ChildControlsCreated
- clientID
- ClientIDSeparator
- Context
- Controls
- DesignMode
- EnableTheming
- EnableViewState
- Events
- ID
- IdSeparator
- IsViewStateEnabled
- Visible
- ViewState

## Q.16 List out the properties of Html control class and draw hierarchy of html server control.

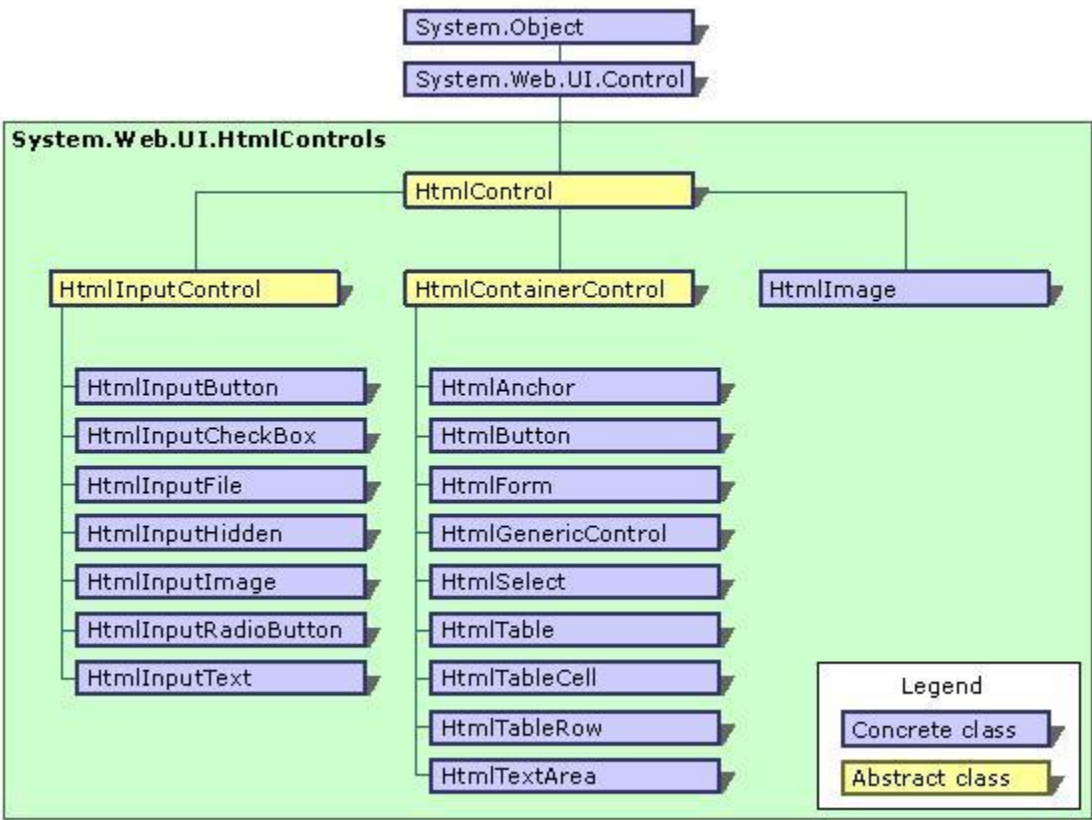
Ans.

 <b>Attributes</b>	Gets a collection of all attribute name and value pairs expressed on a server control tag within the .aspx file.
 <b>ClientID</b> (inherited from <b>Control</b> )	Gets the server control identifier generated by ASP.NET.
 <b>Controls</b> (inherited from <b>Control</b> )	Gets a <b>ControlCollection</b> object that represents the child controls for a specified server control in the UI hierarchy.
 <b>Disabled</b>	Gets or sets a value indicating whether the HTML server control is disabled.
 <b>EnableViewState</b> (inherited from <b>Control</b> )	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.
 <b>ID</b> (inherited from <b>Control</b> )	Gets or sets the programmatic identifier assigned to the server control.
 <b>NamingContainer</b> (inherited from <b>Control</b> )	Gets a reference to the server control's naming container, which creates a unique namespace for differentiating between server controls with the same <b>Control.ID</b> property value.
 <b>Page</b> (inherited from <b>Control</b> )	Gets a reference to the <b>Page</b> instance that contains the server control.
 <b>Parent</b> (inherited from <b>Control</b> )	Gets a reference to the server control's parent control in the page control hierarchy.
 <b>Site</b> (inherited from <b>Control</b> )	Gets information about the Web site to which the server control belongs.
 <b>Style</b>	Gets a collection of all cascading style sheet (CSS) properties applied to a specified HTML server control in the .aspx file.
 <b>TagName</b>	Gets the element name of a tag that contains a <b>runat=server</b> attribute and value pair.
 <b>TemplateSourceDirectory</b> (inherited from <b>Control</b> )	Gets the virtual directory of the <b>Page</b> or <b>UserControl</b> that contains the current server control.
 <b>UniqueID</b> (inherited from <b>Control</b> )	Gets the unique, hierarchically-qualified identifier for the server control.
 <b>Visible</b> (inherited from <b>Control</b> )	Gets or sets a value that indicates whether a server control is rendered as UI on the page.

### Protected Properties:

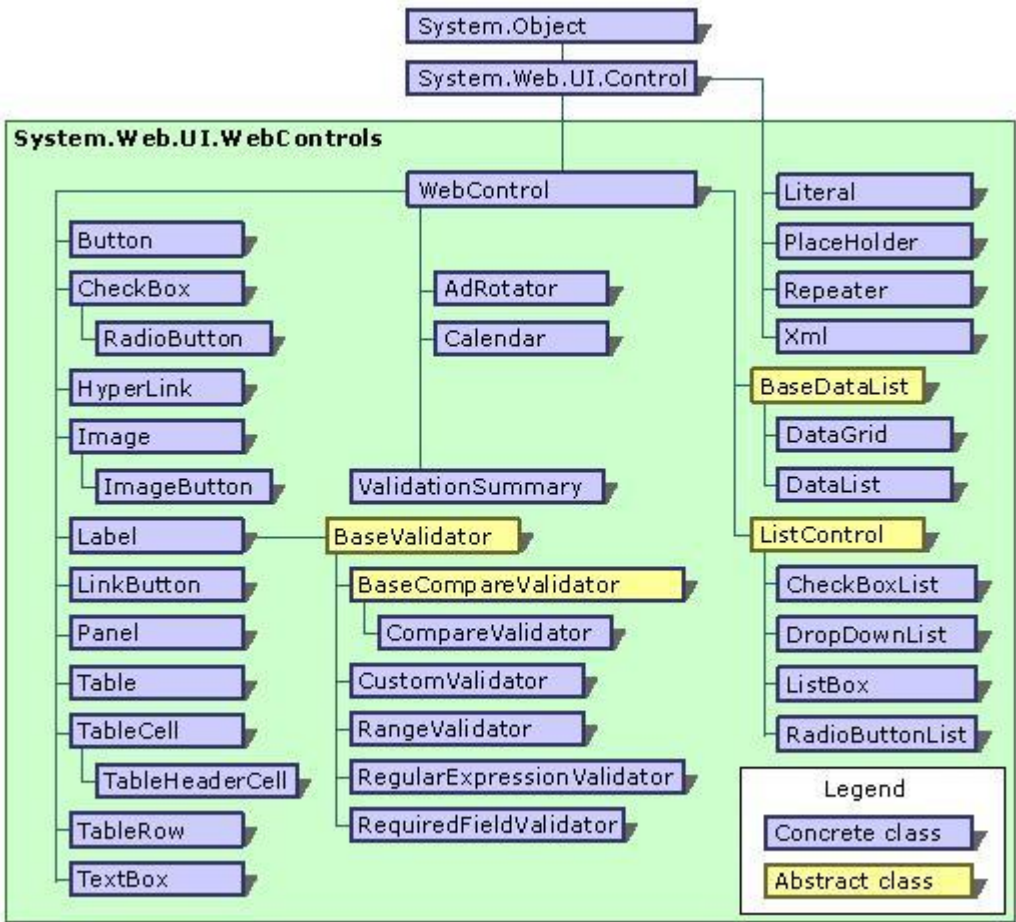
 <b>ChildControlsCreated</b> (inherited from <b>Control</b> )	Gets a value that indicates whether the server control's child controls have been created.
 <b>Context</b> (inherited from <b>Control</b> )	Gets the <b>HttpContext</b> object associated with the server control for the current Web request.
 <b>Events</b> (inherited from <b>Control</b> )	Gets a list of event handler delegates for the control. This property is read-only.
 <b>HasChildViewState</b> (inherited from <b>Control</b> )	Gets a value indicating whether the current server control's child controls have any saved view-state settings.
 <b>IsTrackingViewState</b> (inherited from <b>Control</b> )	Gets a value that indicates whether the server control is saving changes to its view state.
 <b>ViewState</b> (inherited from <b>Control</b> )	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
 <b>ViewStateIgnoresCase</b>	Overridden. See <b>Control.ViewStateIgnoresCase</b> .





Q.17 Draw hierarchy of Web server control and list out properties of web control class.

Ans.



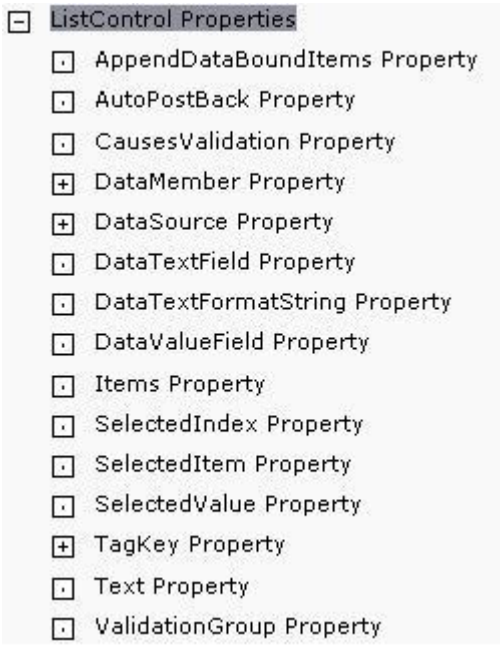
- There are several property of web control class listed below:
- Adapter
  - AppRelativeTemplateSourceDirectory
  - BindingContainer
  - ChildControlsCreated



- clientID
- ClentIDSeparator
- Context
- Controls
- DesignMode
- EnableTheming
- EnableViewState
- Events
- ID
- IdSeparator
- IsViewStateEnabled
- Visible
- ViewState
- IsTrackingViewState

Q. 18 List out the properties of list control class.

Ans.



Q. 19 How many types of validation controls are provided by ASP.NET ?

Ans. There are six main types of validation controls :-

**RequiredFieldValidator** It checks whether the control have any value. It's used when you want the control should not be empty.

**RangeValidator** It checks if the value in validated control is in that specific range. Example TxtCustomerCode should not be more than eight length.

**CompareValidator** It checks that the value in controls should match some specific value. Example Textbox TxtPie should be equal to 3.14.

**RegularExpressionValidator** When we want the control value should match with a specific regular expression.

**CustomValidator** It is used to define UserDefined validation.

**ValidationSummary** It displays summary of all current validation errors.

Q. 20 Explain the client side validation and server side validation ?

Ans. Client Side Validation: If the user is working with a browser that supports dynamic HTML (DHTML), ASP.NET validation controls can perform validation using client script. Because the controls can provide immediate feedback without a round trip to the server, the user experience with the page is enhanced.

Under most circumstances, you do not have to make any changes to your page or to the validation controls to use client-side validation. The controls automatically detect if the browser supports DHTML and perform their checking accordingly. Client-side validation uses the same error display mechanism as server-side validation.

Client side validation runs in the browser to check that the form values are of the correct type. JavaScript should always be used on the client side since it is supported by both Netscape and IE. (VBScript is supported only by IE).

Server side validation checks the code that is submitted to make sure it is correct. Good applications will use both server and client validation. One reason for this is that it would be possible for someone else to write a form that calls your .asp page - they might not use client side validation and might throw in garbage data.

Q.21 Explain the cause validation property of button control.

Ans. Causes Validation: It Gets or sets a value indicating whether validation is performed when the Button control is clicked.

The CausesValidation property specifies if a page is validated when a Button control is clicked.

Page validation is performed when a button is clicked by default.

This property is mostly used to prevent validation when a cancel or reset button is clicked.

Syntax

```
<asp:Button CausesValidation="TRUE|FALSE" runat="server" />
```

Example

The following example removes validation when a button is clicked:

```
<form runat="server">  
<asp:Button id="button1" runat="server"  
CausesValidation="FALSE" Text="Cancel" />  
</form>
```

Q.22 Explain the member of base validator class.

Ans. It serves as the abstract base class for validation controls.The BaseValidator type exposes the following members:

Constructor:[BaseValidator](#): Initializes a new instance of the [BaseValidator](#) class.

Methods:

- AddAttributesToRender Method
- CheckControlValidationProperty Method
- ControlPropertiesValid Method
- DataBind Method
- DetermineRenderUplevel Method
- EvaluateIsValid Method
- FindControl Method
- GetControlRenderID Method
- GetControlValidationValue Method
- GetValidationProperty Method
- OnInit Method
- OnPreRender Method
- OnUnload Method
- RegisterValidatorCommonScript Method
- RegisterValidatorDeclaration Method
- Render Method
- RenderControl Method
- Validate Method

Property:

- ☐

BaseValidator Properties
- ☐

AssociatedControlID Property
- ☐

ControlToValidate Property
- ☐

Display Property
- ☐

EnableClientScript Property
- ☐

Enabled Property
- ☐

ErrorMessage Property
- ☐

ForeColor Property
- ☐

IsValid Property
- ☐

PropertiesValid Property
- ☐

RenderUplevel Property
- ☐

SetFocusOnError Property
- ☐

TagKey Property
- ☒

Text Property
- ☐

ValidationGroup Property

Extension methods:

Explicit interface implementations:

Events:

Name	Description
DataBinding	Occurs when the server control binds to a data source. (Inherited from <a href="#">Control</a> .)
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested. (Inherited from <a href="#">Control</a> .)
Init	Occurs when the server control is initialized, which is the first step in its lifecycle. (Inherited from <a href="#">Control</a> .)
Load	Occurs when the server control is loaded into the <a href="#">Page</a> object. (Inherited from <a href="#">Control</a> .)
PreRender	Occurs after the <a href="#">Control</a> object is loaded but prior to rendering. (Inherited from <a href="#">Control</a> .)
Unload	Occurs when the server control is unloaded from memory. (Inherited from <a href="#">Control</a> .)

Q.23 Explain the validator control with its member.

Ans.

Validation server controls are a collection of controls that allow you to validate an associated input server control, such as a `TextBox`, and display a custom message when validation fails. Each validation control performs a specific type of validation. For example, you can validate against a specific value or a range of values by using the `CompareValidator` and `RangeValidator` controls, respectively. You can even define your own validation criteria by using the `CustomValidator` control. Since the error message is displayed in the validation control, you can control where the message is displayed on the Web page by placing the validation control at the desired location. You can also display a summary of the results from all validation controls on the page by using the `ValidationSummary` control.

By default, page validation is performed when a button control, such as `Button`, `ImageButton`, or `LinkButton`, is clicked. You can prevent validation from being performed when a button control is clicked by setting the `CausesValidation` property of the button control to `false`. This property is normally set to `false` for a cancel or clear button to prevent validation from being performed when the button is clicked.

Base Validation Control Properties

Details the properties that are shared by all validation controls.

Base Compare Validation Control Properties

Details the properties that are shared by all validation controls that perform typed comparisons, such as **`CompareValidator`** and **`RangeValidator`**.

CompareValidator Control

Details the ASP.NET syntax for the `CompareValidator` control. You can use this control to compare a user's entry against a constant value or the value of another control. The comparison operator determines what type of comparison to make (less than, equal, greater than, and so on).

CustomValidator Control

Details the ASP.NET syntax for the `CustomValidator` control. You can use this control to create custom server and client validation code.

RangeValidator Control

Details the ASP.NET syntax for the RangeValidator control. You can use this control to check whether a user's entry is between a specified upper and lower boundary. You can check ranges within pairs of numbers, alphabetic characters, and dates. Boundaries are expressed as constants.

RegularExpressionValidator Control

Details the ASP.NET syntax for the RegularExpressionValidator control. You can use this control to check that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.

RequiredFieldValidator Control

Details the ASP.NET syntax for the RequiredFieldValidator control. You can use this control to ensure that the user does not skip an entry.

Validation Summary Control

Details the ASP.NET syntax for the ValidationSummary control. This control displays a summary of all validation errors for all the validation controls on a page.

Q.24 Explain the Following controls.

Ans.AdRotator Control:

The AdRotator control presents ad images that, when clicked, navigate to a new Web location. Each time the page is loaded into the browser, an ad is randomly selected from a predefined list. The following sample illustrates using the AdRotator control. This control uses an XML file to store the ad information. The XML file must begin and end with an <Advertisements> tag. Inside the <Advertisements> tag there may be several <Ad> tags which defines each ad. The predefined elements inside the <Ad> tag are listed below:

Element	Description
<ImageUrl>	Optional. The path to the image file
<NavigateUrl>	Optional. The URL to link to if the user clicks the ad
<AlternateText>	Optional. An alternate text for the image
<Keyword>	Optional. A category for the ad
<Impressions>	Optional. The display rates in percent of the hits

• Syntax:

```
<asp:AdRotator AdvertisementFile="Ad1.xml"
runat="server" OnAdCreated="change_url"
target="_blank" />
```

• Properties:

Property	Description	.NET
AdvertisementFile	The path to the XML file that contains ad information	1.0
AlternateTextField	A data field to use instead of the Alt text for an advertisement	2.0
ImageUrlField	A data field to use instead of the ImageUrl attribute for an advertisement	2.0
KeywordFilter	A filter to limit ads after categories	1.0
NavigateUrlField	A data field to use instead of the NavigateUrl attribute for an advertisement	2.0
runat	Specifies that the control is a server control. Must be set to "server"	1.0
Target	Where to open the URL	1.0

Calendar Control:

Definition and Usage

The Calendar control is used to display a calendar in the browser. This control displays a one-month calendar that allows the user to select dates and move to the next and previous months.

Syntax:

```
<asp:Calendar DayNameFormat="Full" runat="server"
```

```
SelectionMode="DayWeekMonth" OnSelectionChanged="Date_Selected"
SelectMonthText="<*>"
SelectWeekText="Week"/>
</asp:Calendar>
```

▪ **Properties**

Property	Description
Caption	The caption of the calendar
CaptionAlign	The alignment of the caption text
CellPadding	The space, in pixels, between the cell walls and contents
CellSpacing	The space, in pixels, between cells
DayHeaderStyle	The style for displaying the names of the days
DayNameFormat	The format for displaying the names of the days
DayStyle	The style for displaying days
FirstDayOfWeek	What should be the first day of week
NextMonthText	The text displayed for the next month link
NextPrevFormat	The format of the next and previous month links
NextPrevStyle	The style for displaying next and previous month links
OtherMonthDayStyle	The style for displaying days that are not in the current month
PrevMonthText	The text displayed for the previous month link
runat	Specifies that the control is a server control. Must be set to "server"
SelectedDate	The selected date
SelectedDates	The selected dates
SelectedDayStyle	The style for selected days
SelectionMode	How a user is allowed to select dates
SelectMonthText	The text displayed for the month selection link

**ImageMap Control:**

**Definition and Usage**

Use the ImageMap control to create an image that contains defined hotspot regions. When a user clicks a hot spot region, the control can either generate a post back to the server or navigate to a specified URL.

**Example:**

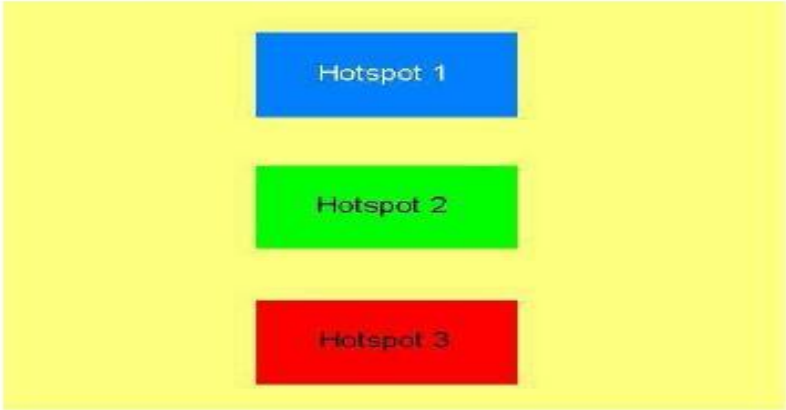
```
<asp:imagemap id="Buttons" imageurl="hotspot.jpg"
alternatetext="Navigate buttons"
runat="Server">
<asp:RectangleHotSpot
hotspotmode="Navigate"
NavigateUrl="navigate1.htm"
alternatetext="Button 1"
top="30"
left="175"
bottom="110"
right="355">
</asp:RectangleHotSpot>
<asp:RectangleHotSpot
hotspotmode="Navigate"
NavigateUrl="navigate2.htm"
alternatetext="Button 2"
top="155"
left="175"
bottom="240"
right="355">
</asp:RectangleHotSpot>
<asp:RectangleHotSpot
hotspotmode="Navigate"
NavigateUrl="navigate3.htm"
alternatetext="Button 3"
```



```
top="285"
left="175"
bottom="365"
right="355">
</asp:RectangleHotSpot>
</asp:imagemap>
</div>
</form>
</body>
</html>
```

▪ **Output:**

**ImageMap Class Mixed HotSpotMode Example**



**FileUpload Control:**

**Definition and Usage**

The FileUpload control enables you to upload file to the server. It displays a text box control and a browse button that allow users to select a file to upload to the server.

**Syntax**

```
<asp:FileUpload id="FileUpload1" AlternateText="You cannot upload files" runat="server" />
```

▪ **Properties & Methods:**

Property	Description
Accept	List of acceptable MIME types
Attributes	Returns all attribute name and value pairs of the element
Disabled	A Boolean value that indicates whether or not the control should be disabled. Default is false
id	A unique id for the element
MaxLength	The maximum number of characters allowed in this element
Name	The name of the element
PostedFile	Gets access to the posted file
runat	Specifies that the element is a server control. Must be set to "server"
Size	The width of the element
Style	Sets or returns the CSS properties that are applied to the control
TagName	Returns the element tag name
Type	The type of the element
Value	The value of the element
Visible	A Boolean value that indicates whether or not the control should be visible
Filename	Returns the name of file uploaded by user.

Wizard Control:

Definition and Usage

The Wizard control provides navigation through a series of steps that collect information incrementally from a user. Many websites include functionality that collects information from the end user (e.g. checking out on an ecommerce website). The Wizard consists of:

- **Collection of WizardSteps:** Each WizardStep contains a discrete piece of content to be displayed to the user. Only one WizardStep will be displayed at a time.
- **Navigation Area:** The navigation area below each WizardStep that contains the navigation buttons to go the next and pervious steps in the wizard.
- **SideBar:** An optional element that contains a list of all WizardSteps and provides a means to skip around the WizardSteps in a random order.
- **Header:** An optional element to provide consistent information at the top of the

WizardStep.

The StepType associated with each WizardStep determines the type of navigation buttons that will be displayed for that step. The StepTypes are:

- **Start:** Displays a Next button.
- **Step:** Displays Next and Previous buttons.
- **Finish:** Displays a Finish button.
- **Complete:** Displays no navigation buttons and hides the SideBar if it is displayed.
- **Auto:**One of the step types listed above is selected based on the order of the step in the collection (e.g. the first step will have a Next button).

Syntax

```
<asp:Wizard ID="Wizard1" runat="server">
<WizardSteps>
<asp:WizardStep ID="WizardStep1" runat="server"
Title="Step 1">
</asp:WizardStep>
<asp:WizardStep ID="WizardStep2" runat="server"
Title="Step 2">
</asp:WizardStep>
</WizardSteps>
</asp:Wizard>
```

Example:



XML Control:

Definition and Usage

The XML control is used to display an XML document or the results of an XSL Transform.

**Note:** At least one of the XML Document properties must be set or no XML document is displayed. You can also specify an XSLT document that will format the XML document before it is written to the output. You can format the XML document with the Transform property or the TransformSource property.

Example:

This example shows how to use the Xml control to display the result of an XSL Transform.

```
<html>
<body>
<form runat="server">
<asp:Xml DocumentSource="cdcatalog.xml" TransformSource="cdcatalog.xsl"
runat="server" />
</form>
<p><a href="cdcatalog.xml" target="_blank">View XML file</a></p>
<p><a href="cdcatalog.xsl" target="_blank">View XSL file</a></p>
</body>
</html>
```

▪ **Properties:**

Property	Description	.NET
ClientID		
Controls		
Document	Deprecated. Specifies an XML document using a System.Xml.XmlDocument object	1.0
DocumentContent	Specifies an XML string	1.0
<a href="#">DocumentSource</a>	Specifies a path to an XML file to display	1.0
EnableTheming		
runat	Specifies that the control is a server control. Must be set to "server"	1.0
SkinID		
Transform	Formats the XML document using a System.Xml.Xsl.XslTransform object	1.0
TransformArgumentList		
TransformSource	Specifies a path to an XSL Transform file	
XPathNavigator		

**Q.25 Explain the Global.asax with its event and one example.**

Ans. The Global.asax file (also known as the ASP.NET application file) is an optional file that is located in the application's root directory and is the ASP.NET counterpart of the Global.asa of ASP. This file exposes the application and session level events in ASP.NET and provides a gateway to all the application and the session level events in ASP.NET. This file can be used to implement the important application and session level events such as Application\_Start, Application\_End, Session\_Start, Session\_End, etc. This provides an overview of the Global.asax file, the events stored in this file and how we can perform application wide tasks with the help of this file.

**What is the Global.asax file?**

According to MSDN, "The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to a application-level events raised by ASP.NET." The Global.asax file is parsed and dynamically compiled by ASP.NET into a .NET Framework class the first time any resource or URL within its application namespace is activated or requested. Whenever the application is requested for the first time, the Global.asax file is parsed and compiled to a class that extends the HttpApplication class. When the Global.asax file changes, the framework reboots the application and the Application\_OnStart event is fired once again when the next request comes in. Note that 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.

**Events in the Global.asax file**

The following are some of the important events in the Global.asax file.

- Application\_Init
- Application\_Start
- Session\_Start
- Application\_BeginRequest
- Application\_EndRequest
- Application\_AuthenticateRequest
- Application\_Error
- Session\_End
- Application\_End

The Global.asax file is used in ASP.NET to specify the global objects and the application and the session level events that would be used by the application. It contains all the application and session level events that are used by the application. This article has discussed this file and its events in a lucid language.

**The purpose of using Global.asax file:**

The purpose of these event handlers is discussed in this section below.

**Application\_Init**

The Application\_Init event is fired when an application initializes the first time.

**Application\_Start**

The Application\_Start event is fired the first time when an application starts.

Session\_Start

The Session\_Start event is fired the first time when a user’s session is started. This typically contains for session initialization logic code.

Application\_BeginRequest

The Application\_BeginRequest event is fired each time a new request comes in.

Application\_EndRequest

The Application\_EndRequest event is fired when the application terminates.

Application\_AuthenticateRequest

The Application\_AuthenticateRequest event indicates that a request is ready to be authenticated. If you are using Forms Authentication, this event can be used to check for the user's roles and rights.

Application\_Error

The Application\_Error event is fired when an unhandled error occurs within the application.

Session\_End

The Session\_End Event is fired whenever a single user Session ends or times out.

Application\_End

The Application\_End event is last event of its kind that is fired when the application ends or times out. It typically contains application cleanup logic.

Q.26 Explain the web.config.

Ans.

Web.Config:

Web.config file, as it sounds like is a configuration file for the Asp .net web application. An Asp .net application has one web.config file which keeps the configurations required for the corresponding application. Web.config file is written in XML with specific tags having specific meanings.

Machine.config:

As web.config file is used to configure one asp .net web application, same way Machine.config file is used to configure the application according to a particular machine. That is, configuration done in machine.config file is affected on any application that runs on a particular machine. Usually, this file is not altered and only web.config is used which configuring applications.

What can be stored in Web.config file?

There are number of important settings that can be stored in the configuration file. Here are some of the most frequently used configurations, stored conveniently inside Web.config file.

- 1. Database connections
- 2. Session States
- 3. Error Handling
- 4. Security

Database Connections:

The most important configuration data that can be stored inside the web.config file is the database connection string. Storing the connection string in the web.config file makes sense, since any modifications to the database configurations can be maintained at a single location. As otherwise we'll have to keep it either as a class level variable in all the associated source files or probably keep it in another class as a public static variable. But it this is stored in the Web.config file, it can be read and used anywhere in the program. This will certainly save us a lot of alteration in different files where we used the old connection. Let’s see a small example of the connection string which is stored in the web.config file.

```
<configuration>
<appSettings>
<add key="ConnectionString"
value="server=localhost;uid=sa;pwd=;database=DBPerson" />
</appSettings>
</configuration>
```

As you can see it is really simple to store the connection string in the web.config file. The connection string is referenced by a key which in this case is "ConnectionString". The value attribute of the configuration file denotes the information about the database. Here we can see that it has database name, userid and password. You can define more options if you want. There is a very good website that deals with all sorts of connection strings.

Q.27 Explain the ViewState with advantages and disadvantages.

Ans. **ViewState** is a built-in structure for automatically retaining values amongst the multiple requests for the same page. The viewstate is internally maintained as a hidden field on the page but is hashed, providing greater security than developer-implemented hidden fields do.

Performance of viewstate varies depending on the type of server control to which it is applied. Label, TextBox, CheckBox, RadioButton, and HyperLink are server controls that perform well with ViewState. DropDownList, ListBox, DataGrid, and DataList suffer from poor performance because of their size and the large amounts of data making roundtrips to the server.

The viewstate describes how an object looks at that particular moment. For example, the viewstate for the text box contains the text "Chris", the viewstate for a button indicates whether or not it's being clicked, and so on. An application that keeps track of this information is said to maintain state.

If you fill out an HTML form and come back to it later, chances are the fields you've filled out will be empty. This is because the Web is a stateless medium—it doesn't allow you to keep track of viewstate or other such information. This was often a pain for traditional ASP developers because it required mechanisms to maintain and retrieve this information. ASP.NET makes this much easier.

ASP.NET automatically keeps track of the viewstate for you. This means that if you fill out an HTML form and click Submit, the values will still be there when the page comes back! This is an important part of ASP.NET and is integral to a number of different mechanisms.

ASP.NET does this by outputting hidden HTML form fields whenever you tell a form `runat="server"`. Remember the hidden field on line 3 ? That string of seemingly random characters is ASP.NET's way of telling itself what each control looks like. When the form is submitted, ASP.NET automatically retrieves this string and uses it to fill out the form information again. ASP.NET uses the fact that browsers can only understand HTML and writes itself reminders in the pages that are sent to the client.

For example, look at the following line written on the server:

```
<form runat="server">
```

This sends the following HTML code to the browser:

```
<form name="ctrl2" method="post" action="listing0201.aspx" id="ctrl2">
<input type="hidden" name="__VIEWSTATE" value="YTB6LTEwNzAyOTU3NjJlX1949e1355cb" />
```

ViewState management showcases ASP.NET's focus on making the Web a more traditional application environment.

Following are the benefits of using View state:-

- ✓ No server resources are required because state is in a structure in the page code.
- ✓ Simplicity
- ✓ States are retained automatically.
- ✓ The values in view state are hashed, compressed, and encoded, thus representing a higher state of security than hidden fields.
- ✓ View state is good for caching data in Web frame configurations because the data is cached on the client. Following are limitation of using View state:-
- ✓ Page loading and posting performance decreases when large values are stored because view state is stored in the page.
- ✓ Although view state stores data in a hashed format, it can still be tampered because it is stored in a hidden field on the page. The information in the hidden field can also be seen if the page output source is viewed directly, creating a potential security risk.

Below is sample of storing values in view state.

```
this.ViewState["EnterTime"] = DateTime.Now.ToString();
```

Q.28 Explain the querystring.

Ans. The QueryString collection is used to retrieve the variable values in the HTTP query string.

The HTTP query string is specified by the values following the question mark (?), like this:

```
<a href= "test.asp?txt=this is a query string test">Link with a query string</a>
```

The line above generates a variable named txt with the value "this is a query string test".

Query strings are also generated by form submission, or by a user typing a query into the address bar of the browser.

Note: If you want to send large amounts of data (beyond 100 kb) the Request.QueryString cannot be used.



Syntax  
Request.QueryString(variable)[(index)].Count]

Parameter	Description
variable	Required. The name of the variable in the HTTP query string to retrieve
index	Optional. Specifies one of multiple values for a variable. From 1 to Request.QueryString(variable).Count

Example 1

To loop through all the n variable values in a Query String:  
The following request is sent:

`http://www.w3schools.com/test/names.asp?n=John&n=Susan`

and names.asp contains the following script:

```
<%  
for i=1 to Request.QueryString("n").Count  
    Response.Write(Request.QueryString("n")(i) & "<br />")  
next  
%>
```

The file names.asp would display the following:

`John  
Susan`

A query string is information sent to the server appended to the end of a page URL.

Following are the **benefits** of using query string for state management:-

- ✓ No server resources are required. The query string containing in the HTTP requests for a specific URL.
- ✓ All browsers support query strings.

Following are **limitations** of query string:-

- ✓ Query string data is directly visible to user thus leading to security problems.
- ✓ Most browsers and client devices impose a 255-character limit on URL length.

Q.29 Explain the different way of passing information between to web pages.

Ans. You can pass information between pages in various ways, some of which depend on how the redirection occurs. The following options are available even if the source page is in a different ASP.NET Web application from the target page, or if the source page is not an ASP.NET Web page:

1. Use a query string.
2. Get HTTP POST information from the source page.

The following options are available only when the source and target pages are in the same ASP.NET Web application.

1. Use session state.
2. Create public properties in the source page and access the property values in the target page.
3. Get control information in the target page from controls in the source page.

**To use a query string to pass information**

1. In the source page when you specify the URL of the target page, include the information that you want to pass in the form of key-value pairs at the end of the URL.
2. In the target page, access query string values by using the [QueryString](#) property of the [HttpRequest](#) object.

**To get the values of controls from the source page in another application**

1. In the source page, include a form element that contains HTML elements (such as input or textarea) or ASP.NET server controls (such as TextBox or DropDownList controls) that post values when the form is submitted.
2. In the target page, read the Form collection, which returns a dictionary of name/value pairs, one pair for each posted value.

To use session state to pass information

1. In the source page, save the information that you want to pass in session state.
2. In the target page, read the saved information from session state.

To get public property values from the source page

1. On the source page, create one or more public properties and save the page.
2. On the target page, add an @PreviousPageType page directive that points to the source page.
3. In target page code, use strongly typed members of the PreviousPage property to read the source code properties.

Q.30 Explain all types of cookies.

Ans. A cookie is a small bit of text that accompanies requests and pages as they go between the Web server and browser. The cookie contains information the Web application can read whenever the user visits the site. Cookies are small files stored on the client computer.

Web sites can store small pieces of information as key-value pairs in the cookie files. Any information can be stored in cookie as key-value pairs. However, there is a limitation on the total size of data that can be sent as cookie. Also, it is not safe to store sensitive information in cookie. Since cookies are stored as text files in client computers, a smart user can edit and change the values in the cookie files.

Most of the web sites store information like your login name, last visit date etc so that when you visit the website next time, they can surprise you with messages like "Welcome back, Mr. John!" Also, you may have noticed that when you login to many websites, there is an option called "Remember my User Id". Those sites will store your user id into the cookies. When you visit the web site next time, they will retrieve the login id from the cookie. Based on the login id, they can retrieve all information about you from the database and automatically login you to the website. Note that they will not store all details about you. Only login id is stored in the cookie. And most often the login id will be stored as encrypted so that nobody can alter it to some other user's login id.

There are two types of cookies.

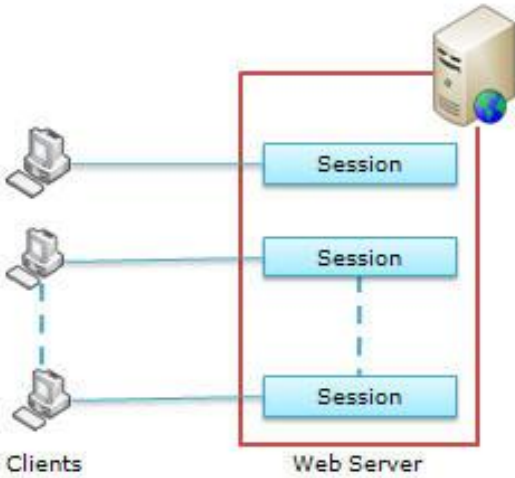
1. Persistent cookies
2. Non-persistent cookies **OR**
1. memorizable cookies
2. Non-memorizable cookies.

**Persistent** cookies are stored on your computer hard disk. They stay on your hard disk and can be accessed by web servers until they are deleted or have expired. Persistent cookies are not affected by your browser setting that deletes temporary files when you close your browser.

**Non-persistent** cookies are saved only while your web browser is running. They can be used by a web server only until you close your browser. They are not saved on your disk. Microsoft Internet Explorer 5.5 can be configured to accept non-persistent cookies but reject persistent cookies.

Q.31 Explain session and session state with its architecture.

Ans. Web is Stateless, which means a new instance of the web page class is re-created each time the page is posted to the server. As we all know HTTP is a stateless protocol, it can't hold the client information on page. If user inserts some information, and move to the next page, that data will be lost and user would not able to retrieve the information. So what we need? we need to store information. Session provides that facility to store information on server memory. It can support any type of object to store along with our custom object. For every client Session data store separately, means session data is stored as per client basis. Have a look at the following diagram.



State Management using session is one of the asp.net best features, because it is secure, transparent from users and we can store any kind of object with in it. Along with advantages, some times session can causes performance issue for heavy traffic sites because its stored on server memory and clients read data from the server itself. Now let's have a look at the advantages and disadvantages of using session in our web application.

Following are the basic advantages and disadvantages of using session.

Advantages:

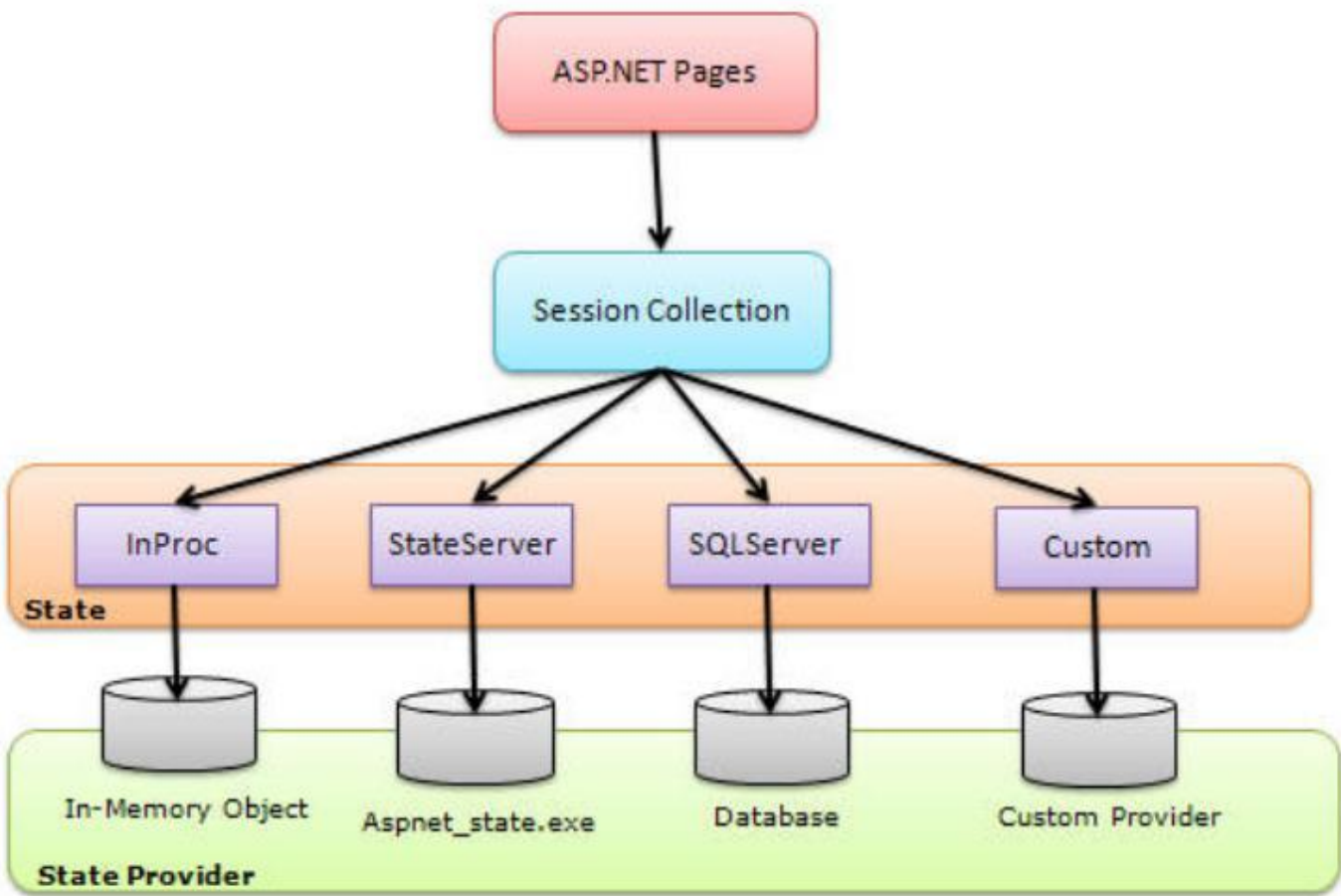
- It helps to maintain user states and data to all over the application.
- It can easily be implemented and we can store any kind of object.
- Stores every client data separately.
- Session is secure and transparent from user.

Disadvantages:

- Performance overhead in case of large volume of user, because of session data stored in server memory.
- Overhead involved in serializing and De-Serializing session Data. Because In case of StateServer and SQLServer session mode we need to serialize the object before store.

Session State Architecture:

For every Session State, there is Session Provider. Following diagram will show you how they are related.



We can choose the session State Provider based on which session state we are selecting. When ASP.NET request for any information based on session ID, session state and its corresponding provider are responsible for sending the proper information based on user. Following tables show, the session mode along with there provider Name.

If we consider about session state, It means all the settings that you have made for your web application for maintaining the session. Session State, it self is a big thing, It says all about your session configuration, either in web.config or from code behind. In web.config, <SessionState> elements used for setting the configuration of session. Some of them are Mode, Timeout, StateConnectionString, Custom provider etc.

Q.32 Explain httpsessionstate class with its member.

Ans.








Provides access to session-state values as well as session-level settings and lifetime management methods.

For a list of all members of this type, see HttpSessionState Members.



Public Properties

 CodePage	Gets or sets the code page identifier for the current session.
 Contents	Gets a reference to the current session-state object.
 Count	Gets the number of items in the session-state collection.
 IsCookieless	Gets a value indicating whether the session ID is embedded in the URL or stored in an HTTP cookie.
 IsNewSession	Gets a value indicating whether the session was created with the current request.
 IsReadOnly	Gets a value indicating whether the session is read-only.
 IsSynchronized	Gets a value indicating whether access to the collection of session-state values is synchronized (thread safe).
 Item	Overloaded. Gets or sets individual session values. In C#, this property is the indexer for the <b>HttpSessionState</b> class.
 Keys	Gets a collection of the keys of all values stored in the session.
 LCID	Gets or sets the locale identifier (LCID) of the current session.
 Mode	Gets the current session-state mode.
 SessionID	Gets the unique session ID used to identify the session.
 StaticObjects	Gets a collection of objects declared by <b>&lt;object Runat="Server" Scope="Session"/&gt;</b> tags within the ASP.NET application file global.asax.
 SyncRoot	Gets an object that can be used to synchronize access to the collection of session-state values.
 Timeout	Gets and sets the time-out period (in minutes) allowed between requests before the session-state provider terminates the session.

Public Methods

 Abandon	Cancels the current session.
 Add	Adds a new item to session state.
 Clear	Clears all values from session state.
 CopyTo	Copies the collection of session-state values to a one-dimensional array, starting at the specified index in the array.
 Equals (inherited from <b>Object</b> )	Overloaded. Determines whether two <b>Object</b> instances are equal.
 GetEnumerator	Gets an enumerator of all session state-values in the current session.
 GetHashCode (inherited from <b>Object</b> )	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
 GetType (inherited from <b>Object</b> )	Gets the <b>Type</b> of the current instance.
 Remove	Deletes an item from the session-state collection.
 RemoveAll	Clears all session-state values.
 RemoveAt	Deletes an item at a specified index from the session-state collection.
 ToString (inherited from <b>Object</b> )	Returns a <b>String</b> that represents the current <b>Object</b> .

Protected Methods

 Finalize (inherited from <b>Object</b> )	Overridden. Allows an <b>Object</b> to attempt to free resources and perform other cleanup operations before the <b>Object</b> is reclaimed by garbage collection. In C# and C++, finalizers are expressed using destructor syntax.
 MemberwiseClone (inherited from <b>Object</b> )	Creates a shallow copy of the current <b>Object</b> .

Q.33 Explain web application state.

Ans.

Web Application State

A Web page is recreated every time it is posted back to the server. In traditional Web programming this means that all the information within the page and information in the controls is lost with each round trip. To overcome this limitation of traditional Web programming, the ASP.NET page framework includes various options to help us preserve changes. One option involves keeping information on the client, directly in the page or in a cookie and another option involves storing information on the server between round trips. We will take a look at both the options.

Saving State in the Client

If we decide to save data in the client then that data is not stored on the server. This means that the page has to store all the data in controls so that it can be sent back to the server when the page is posted back to the server. The different ways in which you can save state in a client are discussed below:

HTML Hidden Form Fields

The default way of saving the state of the data is to use HTML hidden fields. A hidden field stores text data with the HTML <input style="hidden">. A hidden field will not be visible in the browser but we can set its properties just like we set properties for other standard controls. When a page is posted to the server, the content of a hidden field is sent in the HTTP Form collection along with the values of other controls. In order for hidden field values to be available during page processing, we must submit the page using an HTTP post method. That is, you cannot take advantage of hidden fields if a page is processed in response to a link or HTTP GET method.

Cookies

A cookie is a small text file that stores data on the client's machine or is persistent in-memory during the client browser session. It contains page-specific information the server sends to the client along with page output. Cookies can be temporary with specific expiration times and dates or persistent. We can use cookies to store information about a particular client, session, or application. Cookies are saved on client machine, and when the browser requests a page, it sends the information in the cookie along with the request information. The server can read the cookie and extract its value. Cookies are a relatively secure way of maintaining user-specific data as the browser can only send the data back to the server that originally created the cookie.

Query Strings

A query string is information added to the end of a page's URL. For example, it looks like the following:  
http://www.startvbdotnet.com/listbooks.aspx?category=asp&price=50

In the above URL, the query string starts with the question mark (?) and includes two attribute-value pairs, one called "category" and the other called "price." Query strings provide a simple and limited way of maintaining state information. Most browsers impose a 255-character limit on the length of the URL. Also, as the query



values are exposed to the Internet via the URL, in some cases security may be an issue. In order for query string values to be available during page processing, we must submit the page using an HTTP get method. You cannot take advantage of a query string if a page is processed in response to an HTTP post method.

View State

The Control.ViewState property provides a way for retaining values between multiple requests for the same page. This is the method that the page uses to preserve page and control property values between round trips. When a page is processed, the current state of the page and controls is hashed into a string and saved in the page as a hidden field. When the page is posted back to the server, the page parses the view state string at page initialization and restores property information in the page.

Saving State on the Server

We can also save an application's state on the server. Saving an application's state on the server provides with more security than client-side options. The different ways in which you can save state in a server are discussed below:

Application State

An application's state is stored using the application's state object (HttpApplicationState class) for each active Web Application. Application state is a global storage mechanism accessible from all pages in the Web application and is useful for storing information that needs to be maintained between server round trips and between pages. Application state is a key-value dictionary structure created during each request to a specific URL. You can add your application-specific information to this structure to store it between page requests. Once you add your application-specific information to application state, the server manages it for you.

Session State

Besides application state, ASP.NET also can store session states using a session state object (HttpSessionState class) for each active Web Application. Session state is similar to application state, but it is scoped to the current browser session. If different users are using an application, each user will have a different session state. If a user leaves the application and returns later, that user will have a different session state. Session state is a key-value dictionary structure for storing session-specific information that needs to be maintained between server round trips and between requests for pages. Once we add our application-specific information to session state, the server manages this object for us. Depending on the options we specify, session information can be stored in cookies, an out-of-process server, or a SQL Server.

Using Database

Maintaining state using database is a very common method when storing user-specific information where the information store is large. Database storage is particularly useful for maintaining long-term state or state that must be preserved even if the server must be restarted. The database approach is often used along with cookies. For example, when a user accesses an application, he might need to enter a username/password to log in. You can look up the user in your database and then pass a cookie to the user. The cookie might contain only the ID of the user in your database. You can then use the cookie in the requests that follow to find the user information in the database as needed.

Q.34 Explain state management in asp.net.

Ans. Web Pages developed in ASP.Net are HTTP based and HTTP protocol is a stateless protocol. It means that web server does not have any idea about the requests from where they coming i.e from same client or new clients. On each request web pages are created and destroyed.

So, how do we make web pages in ASP.Net which will remember about the user, would be able to distinguish b/w old clients(requests) and new clients(requests) and users previous filled information while navigating to other web pages in web site.

Solution of the above problem lies in State Management.

ASP.Net technology offers following state management techniques.

Client side State Management

- Cookies
- Hidden Fields
- View State
- Query String

Server side State Management

- Session State
- Application State

These state management techniques can be understood and by following simple examples and illustrations of the each techniques .

Q.35 Difference Between theme and CSS.

Ans.

- CSS usually deals with HTML code. You cannot apply CSS to certain ASP.NET specific server controls which are not present in HTML.
- You can apply Themes and skins to all ASP.NET controls with less effort. Themes and skins can be uniformly applied on both windows and asp.net applications.

- You can apply single theme to each page where as multiple style sheets can be applied to each page.
- Themes don't override or cascade style definitions by default the way CSS generally do. But you can selectively override local property settings for a control using StyleSheetTheme attribute in Themes.
- You can include CSS files in Themes which is applied as part of Theme structure but not vice-versa.
- You can apply theming for only those properties that have ThemeableAttribute attribute set to true in their control class.

Q.36 Explain the caching in asp.net.

Ans.

**Caching is a technique of persisting the data in memory for immediate access to requesting program calls. Many in the developer community consider caching as one of the features available to improve performance of Web applications.**

Known as caching, this technique can be used to temporarily store page output or application data either on the client or on the server, which can then be re-used to satisfy subsequent requests and thus avoid the overhead of re-creating the same information.

Caching is particularly suitable when you expect to return the same information in the same format for many different requests.

ASP.NET provides the following types of caching that can be used to build highly responsive Web applications:

**Output caching:** which caches the dynamic response generated by a request.

**Fragment caching:** which caches portions of a response generated by a request.

**Data caching:** which allows developers to programmatically retain arbitrary data across requests.

**Why Caching?**

Consider a page that has list of Employee name, 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 ordering, Voucher creation etc. Now in a normal scenario the process of querying database for each request is not cost-effective in terms of server resources, hence is lot better to cache or persist the data to avoid this costly loss of resources.

The .NET Advantage

ASP.NET provides the flexibility in terms of caching at different levels.

**1. Page Level Output Caching**

This is at the page level and one of the easiest means for caching pages. This requires one to specify Duration of cache and Attribute of caching.

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

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

\* VaryByParam can take various "key" parameter names in query string. Also there are other attributes like VaryByHeader, VaryByCustom etc. Please refer to MSDN for more on this.

**2. Fragment Caching**

Even though this definition refers to caching portion/s of page, it is actually caching a user control that can be used in a base web form page. In theory, if you have used include files in the traditional ASP model then this caching model is like caching these include files separately. In ASP.NET more often this is done through User Controls. Initially even though one feels a bit misleading, this is a significant technique that can be used especially when implementing "n" instances of the controls in various \*.aspx pages. We can use the same syntax that we declared for the page level caching as shown above, but the power of fragment caching comes from the attribute "VaryByControl". Using this attribute one can cache a user control based on the properties exposed.

Syntax: <%@ OutputCache Duration="60" VaryByControl="DepartmentId" %>

**3. Application Level Caching**

With Page level Output caching one cannot cache objects between pages within an application. Fragment caching is great in that sense but has limitations by using user controls as means to do. We can use the Cache object programmatically to take advantage of caching objects and share the same between pages. Further the availability of different overloaded methods gives a greater flexibility for our Cache policy like Timespan, Absolute expiration etc. But one of the biggest takes is the CacheDependency. This means that one can create a cache and associate with it a dependency that is either another cache key or a file.

In almost all Web applications there could be numerous master tables that act as lookups to application specific tables. For e.g. if you take up adding a Employee, usually one has master tables like "tblQualification" to get list of qualifications, "tblLocations" to get list of locations etc. These tables\* are usually set during the initial application configuration phase and could be modified once a month or even less than that.

Everywhere server-side caching is used you can and should use client-side caching as well. Even though the server can serve cached ASP.NET pages very fast it still rely on the browser to download and render the output. When adding client-side caching you get an enormous performance benefit when the page is visited more than once by the same browser.

Q.37 Explain methods to store data in cache.

Ans. **Using Cache Object:**

ASP.NET provides an easy to use caching mechanism. Simply, you can store data that require extensive server resources to be c created. By caching this data you are creating it once and then using it many time without recreating that data again. This can improve your application performanc e significantly.

You can cache your application data by using the cache class. The instance of this class is private to each application. The life time of this class is the life time of the application itself. The cache object is recreated each time the application is restarted. The cache object is global, means the data stored in it is available anywhere within the web application, and in this regard this is similar to the application object. The difference between the cache object and the application object is that the cache object has some more powerful features that allows you to control the cached data.

With the cache object, each item or data stored in the cache object can have its priority state, and can have an expiration time. The Cache object uses those two features in situation like this: When your system's memory becomes scarce, the cache object chooses items with low priority or seldom use to remove and this frees memory. This is how the cache object ensures that unnecessary items does not consume valuable server resources.

ASP.NET allows you to cache web pages or portions of them, calling this an output caching. By caching frequently requested pages or portions of them, you can substantially increases your web server throughput and get a fast page response. You can cache pages on devices like: the web browser making the request, the web server responding to the request, and any other cache capable devices such as proxy servers. There are two ways you can use for output caching: declaratively in a page / a configuration file, or programmatically using a cache API.

Using the OutputCache Directive:

You can use the @OutputCache directive to cache your web page or form in a declarative way. The OutputCache directive has the following attributes:

Duration: Specifies how long the page will be cached, measured in seconds.

Location: Specifies the device you will cache on. Can be: browser, server, or any. (The default is any)

CacheProfile: The name of the cache settings in the web.config file, to associate with the page. (this is optional)

NoStore: A Boolean value that indicate whether or not to allow or prevent secondary storage of sensitive information

Shared: A Boolean value that indicate whether or not the page output can be shared with multiple pages.

SqlDepandecy: A String value that identifies a string of database name and a table name associated with the output cache of this page or control.

VaryByParam: a semi colon delimited list of strings that gives you the ability to cache multiple responses from a single web form

VaryByControl: A semi colon delimited list of strings that gives you the ability to cache portions of web forms .

Q.38 What is difference between dataset and datareader?

Ans. Following are some major differences between dataset and datareader:-

- ✓ DataReader provides forward-only and read-only access to data, while the DataSet object can hold more than one table (in other words more than one rowset) from the same data source as well as the relationships between them.
- ✓ Dataset is a disconnected architecture while datareader is connected architecture.
- ✓ Dataset can persist contents while datareader can not persist contents, they are forward only.

Q.39 What are the major difference between classic ADO and ADO.NET?

Ans. Following are some major differences between both

- ✓ As in classic ADO we had client and server side cursors they are no more present in ADO.NET. Note it's a disconnected model so they are no more applicable.
- ✓ Locking is not supported due to disconnected model.
- ✓ All data persist in XML as compared to classic ADO where data persisted in Binary format also.

Q.40 What is the use of connection object?

Ans. They are used to connect a data to a Command object.

- ✓ An OleDbConnection object is used with an OLE-DB provider
- ✓ A SqlConnection object uses Tabular Data Services (TDS) with MS SQL Server

Q.41 What is the use of command object?

Ans.

They are used to connect connection object to Datareader or dataset. Following are the methods provided by command object :- ✓ ExecuteNonQuery :- Executes the command defined in the CommandText property against the connection defined in the Connection property for a query that does not return any row (an UPDATE, DELETE or INSERT). Returns an Integer indicating the number of rows affected by the query. ✓ ExecuteReader :- Executes the command defined in the CommandText property against the connection defined in the Connection property. Returns a "reader" object that is connected to the resulting rowset within the database, allowing the rows to be retrieved. ✓ ExecuteScalar :- Executes the command defined in the CommandText property against the connection defined in the Connection property. Returns only single value (effectively the first column of the first row of the resulting rowset) any other returned columns and rows are discarded. It is fast and efficient when only a "singleton" value is required

Q.42 What is XML and explain the need of XML.

Ans. XML (Extensible markup language) is all about describing data. Below is a XML which describes invoice data.

```
<?xml version="1.0" encoding="ISO-8859-1"?> <invoice> <productname>Shoes</productname> <qty>12</qty> <totalcost>100</totalcost> <discount>10</discount> </invoice>
```

An XML tag is not something predefined but it is something you have to define according to your needs. For instance in the above example of invoice all tags are defined according to business needs. The XML document is self explanatory, any one can easily understand looking at the XML data what exactly it means.

Remember XML was meant to exchange data between two entities as you can define your user friendly tags with ease. In real world scenarios XML is meant to exchange data. For instance you have two applications who want to exchange information. But because they work in two complete opposite technologies it's difficult to do it technically. For instance one application is made in JAVA and the other in .NET. But both languages understand XML so one of the applications will spit XML file which will be consumed and parsed by other applications

Q.43 What are the different functionalities for XML API for .net framework.

Ans.

The XML API for the .NET Framework comprises the following set of functionalities:

**XML readers** With XML readers the client application get reference to instance of reader class. Reader class allows you to scroll forward through the contents like moving from node to node or element to element. You can compare it with the “SqlDataReader” object in ADO.NET which is forward only. In short XML reader allows you to browse through the XML document.

**XML writers** Using XML writers you can store the XML contents to any other storage media. For instance you want to store the whole in memory XML to a physical file or any other media.

**XML document classes** XML documents provides a in memory representation for the data in an XMLDOM structure as defined by W3C. It also supports browsing and editing of the document. So it gives you a complete memory tree structure representation of your XML document.

Q.44 What is XmlTextReader.

Ans.

The “XmlTextReader” class helps to provide fast access to streams of XML data in a forward-only and read-only manner. It also checks if the XML is well-formed. But XmlTextReader does not validate against a schema or DTD for that you will need “XmlNodeReader” or “XmlValidatingReader” class . Instance of “XmlTextReader” can be created in number of ways.

For example if you want to load file from a disk you can use the below snippets.

```
XmlTextReader reader = new XmlTextReader(fileName);
```

To loop through all the nodes you need to call the “read()” method of the “XmlTextreader” object. “read()” method returns “true” if there are records in the XML document or else it returns “false”.

```
//Open the stream XmlTextReader reader = new XmlTextReader(file); while (reader.Read())
{ // your logic goes here string pdata = reader.Value }
// Close the stream reader.Close();
```

To read the content of the current node on which the reader object is you use the “value” property. As shown in the above code “pdata” gets the value from the XML using “reader.value”.

Q.45 What is different between authorization and authentication.

Ans.

These two concepts seem altogether similar but there is wide range of difference. Authentication is verifying the identity of a user and authorization is process where we check does this identity have access rights to the system. In short we can say the following authentication is the process of obtaining some sort of credentials<sup>198</sup> from the users and using those credentials to verify the user's identity. Authorization is the process of allowing an authenticated user access to resources. Authentication always proceeds to Authorization; even if your application lets anonymous users connect and use the application, it still authenticates them as being a nonymous.