ASP.Net

Data Transfer From One Webform to Another Webform

- Everything from page to controls etc in ASP.Net is an object
- Some commonly used Objects
  - Request
  - Response
  - Server
- Query strings
- Cookies
- Cross Page Postback
- Server.Transfer
    - Contex.Handler
- Session State
- Application State

Request Object

- Represents information coming into the web server
- Request Object is derived from HttpRequest Class.
- The System.Web namespace contains a useful class HttpRequest that can be used to read various HTTP values sent by a client during a web request.
- These Http values woud be used by a serverside program in acting upon a web request.
- HttpRequest Object, exposes the properties and methods we need to manipulate the requests by client computer from within our web application’s code.
- The request object contains all the information that the browser sends to our application when a page is requested or submitted

- This object includes any information that the user provides, like textbox values or check box value

- It also includes all sorts of additional information like cookies, values in URL querystring (if any), the path of the page that is running, the type of the browser that the user is using, the list of languages that are set in browser etc.
- Properties
  - **Application Path**: returns root path on the server
  - **Browser:** returns information about the browser
  - **Cookies:** returns a collection with cookies sent by the client
  - **Physical Path**: physical file system path corresponding to the requested URL

- **Querystring**: Gets the collection of Http querystring variables
- **Server variables**: gets a collection of web server variables
- **URLReferrer**: Gets information about the URL of the client's previous request that linked to the current URL
- **UserHost Address**: Gets IP host address of the remote client
- **USerHostname:** Gets DNS name of the remote client
- **Form:** gets a collection of form variables

Response Object

- Represents information going out from webserver to the browser
- Derived from HttpReponse Class, this object exposes the properties and methods, we need to manipulate the output sent to the client computer from within our web application's code
- Properties
  - **Buffer**: When true, the server buffers its output and sends it to the client after it has finished processing the request. When false, the output is sent to the client in pieces as it becomes available.
  - **Cookies:** a collection that contains the cookies placed on the client computer by web application
  - **Expires**: gets or sets a duration (in minutes) during which the current page in the clients cache will expire.

- Methods
  - **Redirect**: The redirect method accepts URL as an argument and redirects the client to the specified URL
  - **Write**: call this method to add information to output stream. Accepts a string, char, objects as an argument.
  - **Write file**: this methods sends the content of the file on the server's file system to the client

Server Object

- Server Object in ASP.Net is an instance of the System.web.HttpServerUtility class
- Properties
  - MachineName: Name of server computer
- Methods
  - **Execute:** Represents another page, whose URL is passed to the execute method as argument. Optionally we can retrieve the output generated by the second page and include it in the current page
  - **Transfer**: Aborts the execution of the current page and transfers control to new page. The new page is executed as if it were called directly by client application. This method differs from execute method in that the control is not returned to the original page
  - **MapPath**: reutrns the physical file path

Techniques to send data from one webform to another webform in ASP.Net

- Query strings
- Cookies
- Cross Page Postback
- Server.Transfer
    - Contex.Handler
- Session State
- Application State

Querystring

- Appended information to URL
- Querystrings are name/value collection pairs
- More than one value can be send through Querystring
- Querystring cannot be used to send large data as there is a limit on querystring length

Example: Use of Hyperlink Control

Webpage1.aspx

```
<asp:HyperLink   ID="HyperLink1“  runat="server"
NavigateUrl="~/webpage2.aspx?v1=100"> Go to page 2 </asp:HyperLink>
```

webpage2.aspx.vb

```
Protected Sub Page_Load(sender As Object, e As EventArgs) Handles Me.Load
Label1.Text = Request.QueryString(0)
        End Sub
```

Querystring

Example: Using Response.Redirect()

webPage1.aspx

```
Username = “rasid”
enrolNo= “1234”

Response.redirect(“webpage2.aspx? V1=” & username &  “&V2 = ” & enrolNo )
```

webPage2.aspx

```
Dim qs1 As string
Dim qs2 As String
qs1 = Request.Querystring(0)
qs2= Request.Querystring(1)
Label1.text=qs1
Label2.text=qs2
```

- Querystrings are visible to the users, hence should not be used to send sensitive information unless encrypted
- Symbol ? Indicates the beginning of querystring and symbol & is used to concatenate querystrings
- What if & is part of value we want to send?

Qurystring

- Demo

Cookies

- Cookies are little pieces of information that a server stores on a client machine.
- Cookies are generally used to store user preferences or other client specific information
  - Temporary Cookies: Remains in the memory until the browser is open, once the browser is closed cookie is deleted.
  - Persistent Cookies: Remains on the client computer even after browser is closed. Using expires property of HttpCookie Object, we can configure how long cookie remains

Cookies

Example

AddCookie.aspx

```
Dim objCookie As HttpCookie
    objCookie = New HttpCookie("preferences")
    objCookie.Values("color") = "red"
    objCookie.Values("fontface") = "arial"
    objCookie.Values("fontsize") = "4“

    objCookie.Expires = DateTime.Now.AddDays(30)

    Response.Cookies.Add(objCookie)
```

Makes Cookie persistent

ReadCookie.aspx

```
Dim objCookie As HttpCookie

objCookie = Request.Cookies("preferences")

Label1.Text = objCookie("color")
Label2.Text = objCookie("fontface")
Label3.Text = objCookie("fontsize")
```

- **Looking for Cookie on client machine (chrome)**

  Settings/Advance/Privacy and Security/ Content Settings/ Cookies/See all cookies and site data

  Search for localhost

- **Deleting Cookie**

  Settings/Advance/Privacy and Security/Clear Browsing data/ check cookies

Cookies

Demo

Cookies

- How to check if Browser supports Cookies ?

```
If (Request.Browser.Cookies) Then
        // Browser Supports Cookies
    Else
        // Browser Does not Support Cookies
    End If
```

Most of the modern Browsers support Cookies, therefore Request. Browser. Cookie always returns true

- How to check whether cookies are enabled ?

1. Write a test Cookie
2. Redirect to same page
3. Read the test Cookie
4. If cookie persist then cookies are enabled
5. Else- Cookies are disabled

Cookies

Disadvantages

- An individual cookie can contain a very limited information (no more than 4KB)
- We can store only string in a cookie not … for example datasets.
- Total 20 cookies can be used on a single website; if you exceed this, browser will delete older cookies.
- Browser dependent.
  - End user can stop accepting cookies by browsers, so it is recommended to check the users' state and prompt the user to enable cookies.

Crosspage postback

- Crosspage posting allows us to post one page to another page
- By default when we click a button the webform posts to itself
- If we want to post to another webform on a button click set the PostBackUrl property of the Button to the Page you want to post to.

Crosspage postback

Example

CrossPost1.aspx

```
 Name: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox> <br />
Cellphone: <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>  <br />
<asp:Button ID="Button1" runat="server" PostBackUrl="~/CrossPost2.aspx" Text="Button" />
```

CrossPost2.aspx.vb

```
Dim lastPage As Page
lastPage = Page.PreviousPage
Dim txtname As TextBox
txtname = DirectCast(lastPage.FindControl("TextBox1"), TextBox)
Label1.Text = txtname.Text
```

Crosspage postback

Demo

Server.Transfer

- Unlike query string, Url does not change in browser window
- Can be used to retrieve the data of previous page in new page (variables from original request are preserved)
- Cannot be used to navigate to web page external to the webserver
- Faster then response.redirect()

ServerTransferA.aspx.vb

```
Server.Transfer("serverTransferB.aspx")
```

ServerTransferB.aspx.vb

```
Dim previousformcollection As system. Collections.Specilised.Namevaluecollection
Previousformcollection = Request.form
Lblname.text = previousformcollection(txtname)
Lblemail.text=previousformcollection(txtemail)
```

Server.Transfer

Example: Using FindControl

ServerTransferA.aspx.vb

```
Server.Transfer(“serverTransferB.aspx”)
```

ServerTransferB.aspx.vb

```
Dim PreviousPage As Page
PreviousPage = Page.PreviousPage
Dim txtname As TextBox
txtname = DirectCast(lastPage.FindControl("TextBox1"), TextBox)
Label1.Text = txtname.Text
```

ASP.Net

Tracking Sessions

Session State Variables

- **Session variables are like single user global data**
- Session state variables are stored on the webserver by default and are kept for the lifetime of the session
- The life time of a session is determined by the timeout value in web.config file. The default value = 20 mins.
- Session State variables are available across all pages, but only for a given single session

SessionExample2.aspx.vb

```
Session("myname") = TextBox1.Text
 Session("mymobile") = TextBox2.Text
 Response.Redirect("sessionExample2B.aspx")
```

SessionExample2B.aspx.vb

```
Label2.Text = Session("myname")
Label3.Text = Session("mymobile")
```

It is always good to check, if session state variable is null before calling any of its methods such as Tostring() otherwise we may run into runtime “Null Reference Exception’

If (Session("mymobile") IsNot Nothing) Then ….. End if

Session State Variables

- We can add any object to session state , because objects stored in session state are stored on the server, session objects are not subject to the same size limitations as cookies. e.g we can add dataset to session.

Session(“myDataset”)=dstDataset

Session State Variables

Disable Session state

- Application performance can be improved by disabling session state, if it is not required.
- Session state can be turned off at the page or application level
  - To turnoff the session state at page level, Set EnableSessionState = “False” in page directive.
  - To turnoff session state at application level set sessionstate mode = “False” in web.cong file

- Session.Remove(“username”) → Deletes an item ‘username’ from session-state collection.
- Session.RemoveAll → Deletes all session state items
- Session.abandon → explicitly ends the user session.

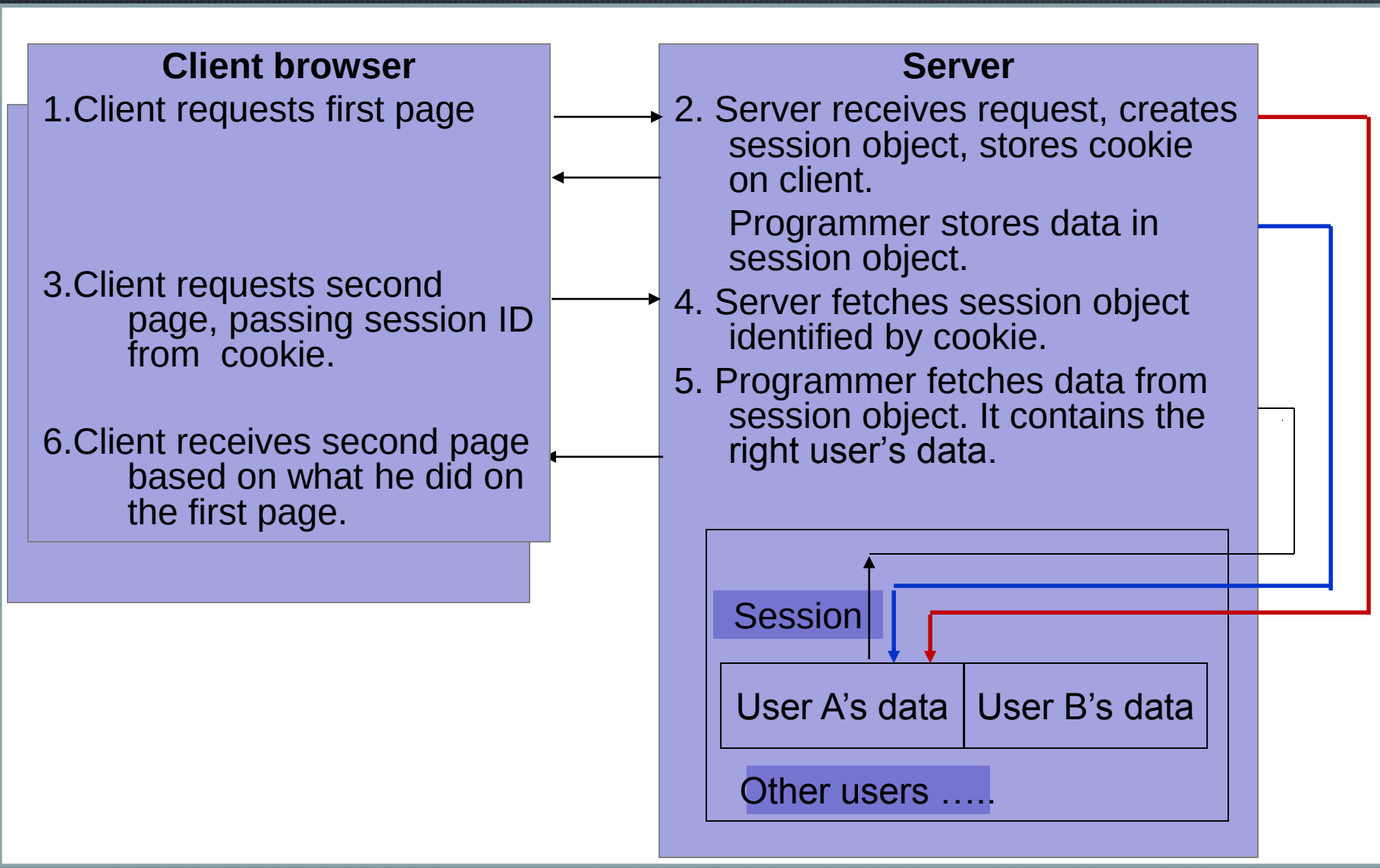
Starting user session

- A user session starts when a user requests the first page from a website. When the first page is requested, the webserver adds the asp.net session ID cookie to the user's browser.
  - We can detect the start of a new session by using the NewSession property of Session Object. When new session starts the new session property returns the value True.

    Response.write(session.NewSession)
  - Each user session is assigned a unique session ID. We can retrieve the value of a session ID like

    Response.write(session.sessionID)
  - A session is guaranteed to be unique among all current user sessions. However, a sessionID is not guaranteed to be unique over time, it might be recycled for a new session in the future

Understanding user session



Session State Variables

Demo

Cookie Less Sessions

- The ASP.NET framework includes an option to enable cookieless sessions. Cookieless sessions enable us to take advantage of session state without relying on browser cookies.

- When a user makes the first request to a Web site with cookieless sessions enabled, the URL used for the request is automatically modified to include the user's session ID.
- For example, if a user makes a request for http://mysite.com/mypage.aspx, the request is automatically modified to http://mysite.com/(nd4vqe2fnbmnwi451fwvda45)/mypage.aspx.

Cookie Less Sessions

- After the session ID is embedded in the URL of the first page request, the session ID will continue to be associated with the user throughout his or her visit to the Web site. The session ID is passed as part of the base URL whenever the user clicks a relative link or submits a form.

- Because the session ID is automatically passed from one page to another, the user can be tracked without relying on a cookie. So, by using cookieless sessions, we get all the advantages of session state without the worries about browser incompatibility.

Cookie Less Sessions

Enabling Cookieless Sessions

- We enable cookieless sessions by modifying a single attribute in the Web.Config file.

```
<configuration>
<system.web>
        <sessionState cookieless="true" />
</system.web>
</configuration>
```

- Cookieless sessions are fully compatible with managing state in process, in a Windows service, or in a database table.

One very significant limitation is imposed on us when we use cookieless sessions. we cannot use absolute URLs when linking between pages. We must design our Web site in such a way that every link uses a URL that is **relative** to the current page.

The following links do not work with a cookieless session:

```
<a href="/mypage.aspx">Click Here</a>
<a href="http://mysite.com/mypage.aspx">Click Here</a>
```

Session Mode

- Off : Disables session state for entire application
- inProc
- State Server
- SqlServer

Web.config

```
<configuration>
<system.web>
<sessionstate
Mode = “inProc”
StateConnectionString = “tcpip=  127.0.0.1:42424”  />
</system.web>
</configuration>
```

Session Mode

InProc

- Session state variables are stored on the web server memory inside ASP.Net worker process (w3wp.exe).
- This is the default session state mode
- If worker
- Easy to implement
- Better performance because session state memory is kept on web server
- Suitable for web applications hosted on a single server
- Disadvantages
  - process is restarted session variables get lost [End process (w3wp.exe) in Taskmanager]
  - Not suitable for web farms
  - Scalability could be an issue

Session Mode

State Server

- When the session state mode is set to stateserver, the session state variables are stored in a process called asp.net state service. This process is different from asp.net worker process
- The asp,ent state service can be present on the same machine as of web server or on a dedicated machine
- Advantages
  - ASP.Net worker process independent thus survives worker process restart
  - Can be used with web farms
  - Offers more scalability
- Disadvantages
  - Slower than inProc
  - Machine having stateserver go down resulting in lost of all the session variales (single point of failure)

```
<configuration>
<system.web>
<sessionstate
Mode = "StateServer"
StateConnectionString = "tcpip=
 127.0.0.1:42424"  />
</system.web>
</configuration>
```

By default Mode = "InProc"

We can run asp.net stateservice on a separate server by suppling the ip address of that server

We can also use the same state service with multiple web servers. For example we can build a web farms in which all the servers share the same state information stored on a separate server

Session Mode

Configure asp.net web application to use state server

- Step1 – start Asp.net windows service
- Step2 – Modify web.config file to set session state mode to value stateserver and specify the location of the state server.

Start/programs/AdministrativeTools/services.

Find asp.net state service and click start, startup type = autometic

OR type

Net start aspnet_state in command prompt.

Session Mode

SQLServer Session Mode

- The session state variables are stored in sqlserver database
- Steps to configure session state mode = sqlserver
  - 1. Create the ASPState database using aspnet_regsql.exe tool
  - 2. set session state mode = sqlserver and sqlconnectionstring in web.config file
- Create Aspstate database
  - Locate c:/Windows/Microsoft.Net/Framework64/v4.0/ap_regsql.exe
  - Go to command prompt
  - Change directory to above path
  - Command: asp_regsql.exe –S mysqlserver name -E – Ssadd – Sstype p
  - Open sqlserver and refresh -→ aspstate has been created
- Open IIS /Application Pools/Asp.Net 4.0 (right click)/Advanced setting/ process model/Identify “Local System”
- Advantages
  - Most reliable. Survives worker process recyclying and sqlserver restarts
  - More scalable then stateserver and Inproc
- Disadvantages
  - Slower

Application State Variables

- Application variables are available across all pages and across all sessions
- **Application variables are like Multiuser global data**
- Stored on webserver
- Cleared only when application ends

ApplicationExample2.aspx.vb

```
Application("myname") = TextBox1.Text
Application("mymobile") = TextBox2.Text
Response.Redirect("ApplicationExample2B.aspx")
```

ApplicationExample2B.aspx.vb

```
Label2.Text = Application("myname")
Label3.Text = Application("mymobile")
```

Application state variables are not thread safe. Lock and unlock methods of the application class must be used to protect against race conditions, deadlocks and access violations

```
Application.loc()
Application(variable_name) = application(variable_name) + 1
Application.unlock()
```

Self study

- Web.config
- Global.asax

Thank you