State Management

Chapter – 8

Traditional desktop application	web application
Single user	***Thousands of users can simultaneously run the same application on the same computer (the web server), each one communicating over a stateless HTTP connection.***
users interact with a continuously running application	***Illusion of continuously running application. clients need to be connected for only a few seconds at most, a web server can handle a huge number of nearly simultaneous requests without a performance hit.***
A portion of memory on the desktop computer is allocated to store the current set of working information.	***In a typical web request, the client connects to the web server and requests a page. When the page is delivered, the connection is severed, and the web server discards all the page objects from memory. By the time the user receives a page, the web page code has already stopped running, and there's no information left in the web server's memory.***

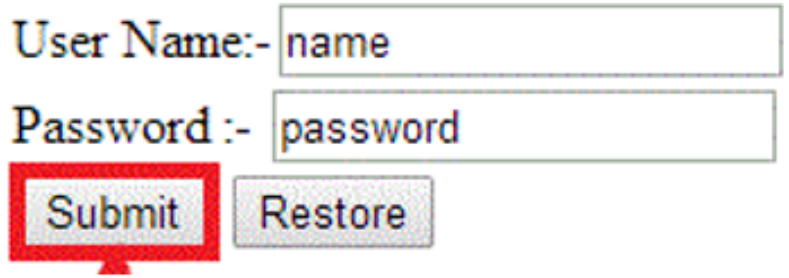
However, if you want to retain information for a longer period of time so it can be used over multiple postbacks or on multiple pages, you need to take additional steps.

Using View State

- One of the most common ways to store information is in view state.
- View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique.
- View state uses a hidden field that ASP.NET automatically inserts in the final, rendered HTML of a web page.
- It’s a perfect place to store information that’s used for multiple postbacks in a single web page.

Code without ViewState

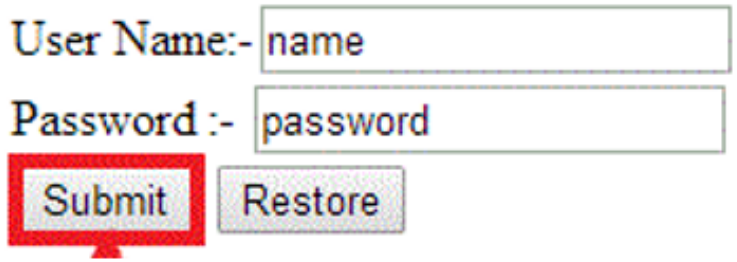
```
public string a, b;
protected void Button1_Click(object sender, EventArgs e)
{
   a = TextBox1.Text;
   b = TextBox2.Text;
   TextBox1.Text = TextBox2.Text = “  ”;
}
```



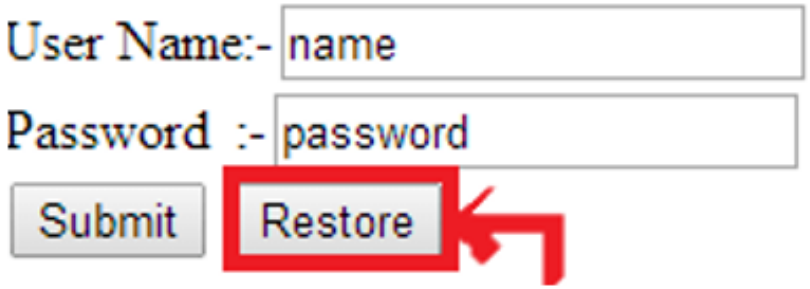
```
protected void Button2_Click(object sender, EventArgs e)
{
   TextBox1.Text = a;
   TextBox2.Text = b;
}
```

Code with ViewState

```
protected void Button1_Click(object sender, EventArgs e)
{
   ViewState["name"] = TextBox1.Text;
   ViewState["password"] = TextBox2.Text;
   TextBox1.Text = TextBox2.Text = "";
}
```



```
protected void Button2_Click(object sender, EventArgs e)
{
   if (ViewState["name"] != null)
   {
      TextBox1.Text = ViewState["name"].ToString();
   }
   if (ViewState["password"] != null)
   {
       TextBox2.Text = ViewState["password"].ToString();
   }
}
```



Limitations

- One of the most significant limitations with view state is that it’s tightly bound to a specific page.
- If the user navigates to another page, this information is lost

Cross-Page Posting

- A cross-page postback is a technique that extends the postback mechanism you've already learned about so that one page can send the user to another page, complete with all the information for that page.
- To use cross-posting, you simply set PostBackUrl to the name of another web form. When the user clicks the button, the page will be posted to that new URL with the values from all the input controls on the current page.
- Interaction between pages take place using Page.PreviousPage property

Code to Demonstrate Cross Page Posting

```
protected void page_load( object sender , eventArgs e)
{
   if (PreviousPage !=null)
         {
           Button btn = (Button)(PreviousPage.FindControl("Button1"));
            TextBox txt = (TextBox)(PreviousPage.FindControl("TextBox1"));
            TextBox1.Font.Size = 12;
            TextBox1.Font.Italic = true;
            TextBox1.ForeColor = System.Drawing.Color.Blue;
            TextBox1.Text="Previous Page is " + PreviousPage.Title + " \nYou
clicked "+ btn.Text + "\nWelcome  " + txt.Text;
         }
}
```

Query String

- This approach is commonly found in search engines. Here's an example:

http://www.google.com/search?q=dotnet+framework

- Well suited in database applications in which you present the user with a list of items that correspond to records in a database, such as products. The user can then select an item and be forwarded to another page with detailed information about the selected item.

- Information is limited to simple strings, which must contain URL-legal characters.
- Information is clearly visible to the user and to anyone else who cares to eavesdrop on the Internet.
- The enterprising user might decide to modify the query string and supply new values, which your program won't expect and can't protect against.
- Many browsers impose a limit on the length of a URL (usually from 1 KB to 2 KB).

Demo Code for Query String :

- Default1.aspx :

Response.Redirect("newpage.aspx?recordID=10&mode=full");

(or)

string url = “newpage.aspx?";

url += "Item=" + lstItems.SelectedItem.Text + "&";

url += "Mode=" + chkDetails.Checked.ToString();

- newpage.aspx :

string ID = Request.QueryString["recordID"];

URL Encoding

- One potential problem with the query string is that some characters aren’t allowed in a URL. All characters must be alphanumeric or one of a small set of special characters (including $-_.+!*'(),).
- Furthermore, some characters have special meaning. For example, the ampersand (&) is used to separate multiple query string parameters, the plus sign (+) is an alternate way to represent a space, and the number sign (#) is used to point to a specific bookmark in a web page. If you try to send query string values that include any of these characters, you’ll lose some of your data.

Demo Code for Query String

Default.aspx : (list box, checkbox and button)

Button Click Event :

```
protected void cmdGo_Click(Object sender, EventArgs e)
{
if (lstItems.SelectedIndex == -1)
        {       lblError.Text = "You must select an item.";      }
else
{
string url = "QueryStringRecipient.aspx?";
url += "Item=" + lstItems.SelectedItem.Text + "&";
url += "Mode=" + chkDetails.Checked.ToString();
Response.Redirect(url);
}
}
```

Newpage.aspx :

```
protected void Page_Load(Object sender, EventArgs e)
{
lblInfo.Text = "Item: " + Request.QueryString["Item"];
lblInfo.Text += "<br />Show Full Record: ";
lblInfo.Text += Request.QueryString["Mode"];
}
```

```
string url = “newpage.aspx?";
url += "Item=" + Server.UrlEncode(lstItems.SelectedItem.Text) + "&";
url += "Mode=" + chkDetails.Checked.ToString();
Response.Redirect(url);
```

Cookies

- Cookies are small files that are created in the web browser's memory (if they're temporary) or on the client's hard drive (if they're permanent).
- Can be easily used by any page in the application and even be retained between visits, which allows for truly long-term storage.
- Same drawbacks that affect query strings—namely,
- Limited to simple string information
- Easily accessible and readable if the user finds and opens the corresponding file.
- Poor choice for complex or private information or large amounts of data.
- Some users disable cookies on their browsers, which will cause problems for web applications that require them.
- Also, users might manually delete the cookie files stored on their hard drives.
- But for the most part, cookies are widely adopted and used extensively on many websites.
- using System.Net;

Demo Code for Creating Cookies

```
HttpCookie userInfo = new HttpCookie("userInfo");
userInfo["UserName"] = “TestUser";
userInfo["UserCity"] = “Mumbai";
userInfo.Expires =  DateTime.Now.AddMinutes(10);
Response.Cookies.Add(userInfo);
```

Demo Code for Retriving Cookies

```
string User_Name = string.Empty;
string User_City = string.Empty;
HttpCookie reqCookies = Request.Cookies["userInfo"];
if (reqCookies != null)
{
    User_Name = reqCookies["UserName"].ToString();
    User_City = reqCookies["UserCity"].ToString();
}
```

Session State Variables

To create session state variables :

```
Session["username"] = TextBox1.Text;
Session["Email"] = TextBox2.Text;
 Session["Subject"] = ListBox1.SelectedValue;
```

To retrieve the session variables:

```
String uname=Session[“username”].ToString();
```

Validation Controls

Chapter – 9

Types of User Errors during Data Entry
Leaving a field blank
Entering invalid data by mistake
Entering invalid data on purpose to break the code

Validation Controls

Control Class	Description
RequiredFieldValidator	Validation succeeds if input control doesn't contain an empty string.
RangeValidator	Validation succeeds if the input control contains a value within a specific numeric, alphabetic, or date range.
CompareValidator	Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that is specified.
RegularExpressionValidator	Validation succeeds if the value in an input control matches a specified regular expression.
CustomValidator	Validation is performed by a user-defined function.

Server-Side Validation

- Validator controls can be used to verify a page automatically when the user submits it or manually in your code.
- When using automatic validation, the user receives a normal page and begins to fill in the input controls.
- When finished, the user clicks a button to submit the page. Every button has a CausesValidation property, which can be set to true or false

Client-Side Validation

- In modern browsers, ASP.NET automatically adds JavaScript code for client-side validation. In this case, when the user clicks a CausesValidation button, the same error messages will appear without the page needing to be submitted and returned from the server. This increases the responsiveness of your web page.
- However, even if the page validates successfully on the client side, ASP.NET still revalidates it when it’s received at the server.

BaseValidator Class

Property	Description
ControlToValidate	Identifies the control that this validator will check.
ErrorMessage and ForeColor	If validation fails, the validator control can display a text message (set by the ErrorMessage property). By changing the ForeColor, you can make this message stand out in angry red lettering.
Display	Static / Dynamic
IsValid , Enabled, EnableClientScript	True / False

Validator-Specific Properties

Property	Description
RangeValidator	MaximumValue, MinimumValue, Type
CompareValidator	ControlToCompare, Operator, Type, ValueToCompare
RegularExpressionValidator	ValidationExpression
CustomValidator	ClientValidationFunction, ValidateEmptyText, ServerValidate event

Manual Validation

Disable validation and perform the work on your own

You can create manual validation in one of three ways:

- Use your own code to verify values. In this case, you won’t use any of the ASP.NET validation controls.
- Disable the EnableClientScript property for each validation control. This allows an invalid page to be submitted, after which you can decide what to do with it depending on the problems that may exist.
- Add a button with CausesValidation set to false. When this button is clicked, manually validate the page by calling the Page.Validate() method. Then examine the IsValid property and decide what to do.

Validation with Regular Expression

Character	Description
*	Zero or more occurrences of the previous character or subexpression. For example, 7*8 matches 7778 or just 8.
+	One or more occurrences of the previous character or subexpression. For example, 7+8 matches 7778 but not 8.
{ }	Groups a subexpression that will be treated as a single element. For example, (78)+ matches 78 and 787878.
{m,n}	Previous character (or subexpression) can occur from *m* to *n* times. A{1,3}
|	Either of two matches
[ ]	Matches one character in a range of valid characters.
[^]	Matches a character that isn't in the given range. [^A-B]
.	Any character except a newline.
\s	Any whitespace character (such as a tab or space).
\S	Any non-whitespace character.
\d	Any digit character
\D	Any character that isn't a digit.
\w	Any "word" character (letter, number, or underscore).
\W	Any character that isn't a "word" character (letter, number, or underscore).

Master Page

- Chapter 12 – Page 369

- [a-z][A-Z]+ - one lower case character followed by one or more uppercase character
- [a-zA-Z]+ - one or more lowercase/uppercase character
- [a-zA-Z]+[ ] [a-zA-Z]+ - one or more lowercase/uppercase character, followed by a space, followed by one or more lowercase/uppercase character