DOT NET

Unit 3

Caching

- caching is a concept of retaining important data for just a short period of time.
- Caching is often used to store information that’s retrieved from a database.
- A good caching strategy identifies the most frequently used pieces of data that are the most time-consuming to create and stores them
- *Cache data (or web pages) that are expensive*
- *Cache data (or web pages) that are used frequently*:

- *Performance* is a measure of how quickly a web page works for a single user. Caching improves performance because it bypasses bottlenecks like the database. As a result, web pages are processed and sent back to the client more quickly.
- *Scalability* measures how the performance of your web application degrades as more and more people use it at the same time. Caching improves scalability because it allows you to reuse the same information for requests that happen in quick succession. With caching, more and more people can use your website, but the number of trips to the database won't change very much. Therefore, the overall burden on the system will stay relatively constant,

Types of Caching

- *Output caching*: This is the simplest type of caching. It stores a copy of the final rendered HTML page that is sent to the client. The next client that submits a request for this page doesn't actually run the page. Instead, the final HTML output is sent automatically. The time that would have been required to run the page and its code is completely reclaimed.
- *Data caching*: This is carried out manually in your code. To use data caching, you store important pieces of information that are time-consuming to reconstruct (such as a DataSet retrieved from a database) in the cache. Other pages can check for the existence of this information and use it, thereby bypassing the steps ordinarily required to retrieve it.

- (No QueryString)

<%@ OutputCache Duration="20" VaryByParam="None" %>

- Twenty seconds may seem like a trivial amount of time, but in a high-volume site, it can make a dramatic difference. For example, you might cache a page that provides a list of products from a catalog.
- By caching the page for 20 seconds, you limit database access for this page to three operations per minute. Without caching, the page will try to connect to the database once for each client and could easily make dozens of requests in the course of 20 seconds.

Caching and the Query String

- You request a page without any query string parameter and receive page copy A.
- You request the page with the parameter ProductID=1. You receive page copy B.
- Another user requests the page with the parameter ProductID=2. That user receives copy C.
- Another user requests the page with ProductID=1. If the cached output B has not expired, it’s sent to the user.
- The user then requests the page with no query string parameters. If copy A has not expired, it’s sent from the cache.

Caching and QueryString

Developers overemphasize on the dynamic information in real-time. Caching enables us to use outdated data for a brief period of time.

- One example is if the page uses information from the current user's session to tailor the user interface. In this case, full page caching just isn't appropriate because the same page can't be reused for requests from different users (although fragment caching may help).
- Another example is if the page is receiving information from another page through the query string.

<%@ OutputCache Duration="20" VaryByParam="*" %>

- Selected QueryString Variables

<%@ OutputCache Duration="20" VaryByParam="ProductID" %>

<%@ OutputCache Duration="20" VaryByParam="ProductID;Type" %>

- ***<%@ OutputCache Duration="20" VaryByParam="ProductID;Type" %>***
- http://localhost:56315/OutputCaching.aspx
- http://localhost:56315/OutputCaching.aspx?CustomerID=12
- http://localhost:56315/OutputCaching.aspx?key=98534
- http://localhost:56315/OutputCaching.aspx?ProductID=5
- http://localhost:56315/OutputCaching.aspx?ProductID=452
- http://localhost:56315/OutputCaching.aspx?CustomerID=12&Type='CDN'

Multiple Caching Example

ebApplication11.WebForm1

```
namespace WebApplication11
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = "Current time : ";
            Label1.Text += DateTime.Now.ToString();
            //Label1.Text += "This is Copy with Id = " + Request.QueryString["id"];
            //Label1.Text += "QueryString - name will not create cached copies";
            Label1.Text += "-" + Request.QueryString["version"];
        }

        protected void Button2_Click(object sender, EventArgs e)
        {
            Response.Redirect("WebForm1.aspx?version=" + Button2.ID.ToString());
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            Response.Redirect("WebForm1.aspx?version=" + Button1.ID.ToString());
        }

        protected void Button3_Click(object sender, EventArgs e)
        {
            Response.Redirect("WebForm1.aspx?version=" + Button3.ID.ToString());
        }


    }
}
```

Data Binding in Desktop applications

- The basic principle of data binding is this: you tell a control where to find your data and how you want it displayed, and the control handles the rest of the details.
- In desktop or client/server applications, data binding involves creating a direct connection between a data source and a control in an application window. If the user changes a value in the onscreen control, the data in the linked database is modified automatically. Similarly, if the database changes while the user is working with it (for example, another user commits a change), the display can be refreshed automatically.

Data Binding in ASP.NET

- ASP.NET data binding works in one direction only. Information moves *from* a data object *into* a control. Then the data objects are thrown away, and the page is sent to the client.
- If the user modifies the data in a data-bound control, your program can update the corresponding record in the database, but nothing happens automatically.

Data Binding

- Simple Data Binding

To add information anywhere in the webpage. Does not involve database.

- Repeated Data Binding

To display database contents / contents from an array

This requires a special control such as listbox, checkboxlist or GridView control which has DataSource as one of the properties.

How to use data binding

- To use single-value binding, you must insert a data-binding expression into the markup in the .aspx file (not the code-behind file). To use repeated-value binding, you must set one or more properties of a data control. Typically, you'll perform this initialization when the Page.Load event fires.
- After you specify data binding, you need to activate it by calling the DataBind() method.
- The DataBind() method is a basic piece of functionality supplied in the Control class. It automatically binds a control and any child controls that it contains.
- With repeated-value binding, you can use the DataBind() method of the specific list control you're using. Alternatively, you can bind the whole page at once by calling the DataBind() method of the current Page object. After you call this method, all the data-binding expressions in the page are evaluated and replaced with the specified value.
- If you forget to use it, ASP.NET will ignore your data-binding expressions, and the client will receive a page that contains empty values.

Simple Data binding

- <%# expression_goes_here %>

/ebApplication1.WebForm1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication1
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected int TransactionCount;
        protected void Page_Load(object sender, EventArgs e)
        {
            TransactionCount = 10;
            this.DataBind();
        }
    }
}
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label id="Label1" runat="server" Font-Size="X-Large">
There were <%# TransactionCount %> transactions today.
I see that you are using <%# Request.Browser.Browser %>.
</asp:Label>

    </div>
    </form>
</body>
</html>
```

```
public partial class DataBindingUrl : System.Web.UI.Page
{
    protected string URL;

    protected void Page_Load(Object sender, EventArgs e)
    {
        URL = "Images/picture.jpg";
        this.DataBind();
    }
}
```

You can now use this URL to create a label, as shown here:

```
<asp:Label id="lblDynamic" runat="server"><%# URL %></asp:Label>
```

You can also use it for a check box caption:

```
<asp:CheckBox id="chkDynamic" Text="<%# URL %>" runat="server" />
```

or you can use it for a target for a hyperlink:

```
<asp:Hyperlink id="lnkDynamic" Text="Click here!" NavigateUrl="<%# URL %>"
 runat="server" />
```

You can even use it for a picture:

```
<asp:Image id="imgDynamic" ImageUrl="<%# URL %>" runat="server" />
```

Repeated-value data binding

- works with the ASP.NET list controls. To use repeated-value binding, you link one of these controls to a data source (such as a field in a data table).
- When you call DataBind(), the control automatically creates a full list by using all the corresponding values.
- *ListBox, DropDownList, CheckBoxList, and RadioButtonList*: These web controls provide a list for a single field of information.
- *GridView, DetailsView, FormView, and ListView*: These rich web controls allow you to provide repeating lists or grids that can display more than one field of information at a time.
- Practical – Page 480 - 486

```
protected void Page_Load(object sender, EventArgs e)
{
    List<string> fruit = new List<string>();
    fruit.Add("Kiwi");
    fruit.Add("Pear");
    fruit.Add("Mango");
    fruit.Add("Blueberry");
    fruit.Add("Apricot");
    fruit.Add("Banana");
    fruit.Add("Peach");
    fruit.Add("Plum");

    // Define the binding for the list controls.
    ListBox1.DataSource = fruit;
    DropDownList1.DataSource = fruit;
    CheckBoxList1.DataSource = fruit;
    RadioButtonList1.DataSource = fruit;



    // Activate the binding.
    this.DataBind();
}
```

```
protected void Page_Load(Object sender, EventArgs e)
{
    if (!this.IsPostBack)
    {
        // Use integers to index each item. Each item is a s
        Dictionary<int, string> fruit = new Dictionary<int,

        fruit.Add(1, "Kiwi");
        fruit.Add(2, "Pear");
        fruit.Add(3, "Mango");
        fruit.Add(4, "Blueberry");
        fruit.Add(5, "Apricot");
        fruit.Add(6, "Banana");
        fruit.Add(7, "Peach");
        fruit.Add(8, "Plum");

        // Define the binding for the list controls.
        MyListBox.DataSource = fruit;

        // Choose what you want to display in the list.
        MyListBox.DataTextField = "Value";

        // Activate the binding.
        this.DataBind();
    }
}
```

```
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Data.SqlClient;

namespace WebApplication1
{
    public partial class WebForm4 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!this.IsPostBack)
            {
            // Define the ADO.NET objects for selecting products from the database.
            string selectSQL = "SELECT Name, Price from Table_1";
            SqlConnection con = new SqlConnection("Data Source=MUM15121CPU1935\\DEMO1;Initial Catalog=books;Integrated Security=True"
            SqlCommand cmd = new SqlCommand(selectSQL, con);
            // Open the connection.
            con.Open();
            // Define the binding.
            ListBox1.DataSource = cmd.ExecuteReader();
            ListBox1.DataTextField="Name";
            ListBox1.DataValueField="Price";
            // Activate the binding.
            this.DataBind();
            con.Close();
            ListBox1.SelectedIndex = -1;
            }
        }
    }
}
```