

Server controls, Cookies

(chp6-8)

Slide
1

What Does ASP.NET Server Control Mean?

An ASP.NET server control is a tag written in a Web page to represent a programmable server-side object used for displaying a user interface element in a Web page. ASP.NET server controls are tags that can be understood by the server.

The ASP.NET page framework includes several built-in server controls that are designed to provide a more structured programming model for the Web.

These controls provide the following features:

- Automatic state management.
- Simple access to object values without having to use the `Request` object.

In addition to the built-in controls, the ASP.NET page framework also provides the ability to create user controls and custom controls. User controls and custom controls can enhance and extend existing controls to build a much richer user interface.

Web server controls

Web controls are like the HTML server controls such as Button, Textbox, and Hyperlink, except that Web controls have a standardized set of property names.

Web server controls offer the following advantages:

- Make it easier for manufacturers and developers to build tools or applications that automatically generate the user interface.

To use a Web server control, use the following syntax (which uses the TextBox control as an example):

ASP.NET (C#)

```
<asp:textbox text="hello world" runat=server />
```

to use an HTML server control, use the following syntax (which uses the HtmlInputText control as an example):

ASP.NET (C#)

```
<input type="text" value="hello world" runat=server />
```

Button control example

example

The following example demonstrates the declaration for a **submit** button control in an .aspx file.

```
<asp:Button id="SubmitButton"
  Text="Submit"
  OnClick="SubmitBtn_Click"
  runat="server"/>
```

Copy

The following example demonstrates the declaration for a **command** button control in an .aspx file.

```
<asp:Button id="SortAscendingButton"
  Text="Sort Ascending"
  CommandName="Sort"
  CommandArgument="Ascending"
  OnCommand="CommandBtn_Click"
  runat="server"/>
```

Copy

The following example shows an event-handling method that gets the button click and displays the information passed from the button in its **CommandName** and **CommandArgument** properties.

```
VB
Sub CommandBtn_Click(sender As Object, e As CommandEventArgs)
  Message.Text = "You clicked the " & e.CommandName & _
    " - " & e.CommandArgument & " button."
End Sub
[C#]
void CommandBtn_Click(Object sender, CommandEventArgs e)
{
  Message.Text = "You clicked the " + e.CommandName +
    " - " + e.CommandArgument + " button.";
}
```

Copy

By default, page validation is performed when a **Button** control is clicked. Page validation determines whether the input controls associated with a validation control on the page pass the validation rules specified by the validation control.

Common server controls

Name	HTML	Prefix
Label	<code>span</code>	<code>lbl</code>
TextBox	<code>input</code>	<code>txt</code>
CheckBox	<code>input/label</code>	<code>chk</code>
RadioButton	<code>input/label</code>	<code>rdo</code>
Button	<code>input</code>	<code>btn</code>
LinkButton	<code><a></code>	<code>lbtn</code>
ImageButton	<code>input</code>	<code>ibtn</code>
Image	<code>img</code>	<code>img</code>
ImageMap	<code>img/map</code>	<code>imap</code>
HyperLink	<code><a></code>	<code>hlnk</code>
FileUpload	<code>input</code>	<code>upl</code>

List server controls

Name	HTML	Prefix
DropDownList	select/option	ddl
ListBox	select/option	lst
CheckBoxList	input/label	cbl
RadioButtonList	input/label	rbl
BulletedList	ul or ol/li	blst

Common control events

Event	Attribute	Controls
Click	OnClick	Button Image button Link button Image map
Command	OnCommand	Button Image button Link button
TextChanged	OnTextChanged	Text box
CheckedChanged	OnCheckedChanged	Check box Radio button
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list List box Radio button list Check box list

A Click event handler wired by a Handles clause

The aspx for a button control

```
<asp:Button id="btnCancel" runat="server"  
    Text="Cancel Order" />
```

The event handler for the Click event of the control

```
Protected Sub btnCancel_Click(sender As Object,  
    e As EventArgs) Handles btnCancel.Click  
    Session.Remove("Cart")  
    Response.Redirect("Order.aspx")  
End Sub
```


- The HTTP protocol was designed to be a stateless protocol
 - Every request for a file or resource from a web server is treated as a brand new request
 - The web server does not associate any requests to any other requests



A **stateless** web application means that the server creates a new instance of the web page every time you request a web page. Every time you submit a request to the server, the information entered by the user in text boxes is sent to the server, but it isn't returned to the browser. Such requests can make you lose the **data** in each of such trips.

The **ASP.NET** page framework provides you with different state management features to preserve the control values and properties between such round trips to overcome such limitations.

We already talk about viewstate in previous video in Unit1 Week 4

View State is one of the methods of the ASP.NET page framework used to preserve and store the page and control values between round trips. It is maintained internally as a hidden field in the form of an encrypted value and a key.

The View State methods differ from the cache and cookies because the cookies are accessible from all the pages on your website, while the View State values are non-transferable, and thus you cannot access from different pages'

Cookie state

A cookie is a data received from a web application which is stored by the browser. All cookies are stored in a single file and will be included in the HTTP header on each request made to the same server. When a web application sets a cookie, it can provide an expiration date, a duration, apply restrictions to a specific domain and path to limiting where the cookie is sent and so on.

```
HTTP/2.0 200 OK
Content-type: text/html
Set-Cookie: first_cookie=hello; Domain=example.com
Set-Cookie: second_cookie=world; Expires=Wed, 21 Oct 2015 07:28:00 GMT;

[page content]
```

Each cookie can have one or more of the following attributes:

1. **Expires:** makes the cookie expire at a specific date
2. **Max-Age:** makes the cookie expire after a specific length of time

ASP.NET and cookies

In order to manage cookie data more easily, ASP.NET provides the **HttpCookie** class. Both request and response objects expose the collection of cookies through the **HttpCookieCollection** and each cookie is a name/value pair that are accessible through the Values collection of the **HttpCookie** class or indirectly through the default indexer provided by the class. To request that a client set a cookie, add a new **HttpCookie** instance to the response cookie collection before your page rendering.

```
protected void Page_Load(Object sender, EventArgs E)
{
    int cookieValue = 0;
    if (Request.Cookies["Test"] == null)
    {
        HttpCookie cookie = new HttpCookie("Test");
        cookie.Value = "Hello world";
        Response.Cookies.Add(cookie);
    }
    else
    {
        cookieValue = Convert.ToInt32(Request.Cookies["Test"].Value);
    }
}
```

Application concepts

- An ASP.NET *application* is the collection of pages, code, and other files within a single directory on a web server.
- An application begins when the first user requests a page that's a part of the application. Then, ASP.NET initializes the application before it processes the request for the page.
- As part of its initialization, ASP.NET creates:
 - an *application object* from the `HttpApplication` class
 - an *application state object* from the `HttpApplicationState` class
 - a *cache object* from the `Cache` class.
- These objects exist for the duration of the application, and items stored in application state or cache are available to all users of the application.
- Once an application has started, it doesn't normally end until the web server is shut down.

Cookies are small tokens of data that are stored on the computer. They are used to store user data on the client end. Some of the details stored on the client's computer include name, user id, contacts, and address. ASP.NET supports two sets of cookies; persistent and non-persistent.

A session is a state where user values are retrieved and kept on the webpage. It is used to pass data from one page to another in ASP.NET.

ASP.NET supports two sets of cookies;

persistent
and non-persistent.

Cookies that have an expiry date are referred to as persistent cookies, while those that don't expire are known as non-persistent cookies.

A session is a state where user values are retrieved and kept on the webpage.

Creating cookies in ASP.NET

The `Response.cookies` is used to create a cookie in ASP.NET. We assign a value to the cookie, as shown below.

```
<%  
Response.cookies("StudentName") = "StanleyWambui"  
%>
```

In the above example, our cookie's name is `StudentName` and its value is `Stanley Wambui`.

Next, we set the expiry date of the cookies, as highlighted below:

```
<%  
Response.cookies("StudentName") = "Stanley Wambui"  
Response.cookies("StudentName").Expires = #May 05 2022#  
%>
```


Retrieving value of cookies

The `Request.cookies` command is applied to reveal the cookies' value. We are retrieving the cookies named `StanleyWambui` to display on the page.

```
<%  
FirstName = Request.cookie("StudentName")  
Response.write("StudentName" = &FirstName)  
%>
```

The output will be `StudentName Stanley Wambui`.

Creating collection of cookies

It's possible to create a collection of cookies, rather than one at a time. The code below shows the creation of multiple cookies in ASP.NET.

```
<%  
Response.cookies("StudentName")("FullName") = "Stanley Wambui"  
Response.cookies("StudentName")("School") = "Computing and Informatics"  
Response.cookies("StudentName")("Campus") = "Nairobi"  
Response.cookies("StudentName")("Registration") = "380940"  
%>
```

Examples of cookies

`ASP.NET_SessionId=jsswpu5530hcyx2w3jfa5u55`

`EMail=mary@techknowsolve.com`

`user_ID=4993`

Two ways to create a cookie

```
New HttpCookie(name)
```

```
New HttpCookie(name, value)
```

Common properties of the HttpCookie class

Expires

Name

Secure

Value

Code that creates a session cookie

```
Dim nameCookie As New HttpCookie("UserName", userName)
```

Code that creates a persistent cookie

```
Dim nameCookie As New HttpCookie("UserName")  
nameCookie.Value = userName  
nameCookie.Expires = DateTime.Now.AddYears(1)
```

The `HttpCookieCollection` class

Common properties

Count

Item(name)

Common methods

Add(cookie)

Clear()

Remove(name)

A procedure that creates a new cookie and adds it to the HttpResponse object

```
Private Sub AddCookie()  
    Dim NameCookie As New HttpCookie("UserName",  
                                       txtUserName.Text)  
    NameCookie.Expires = Now.AddYears(1)  
    Response.Cookies.Add(NameCookie)  
End Sub
```

A procedure that retrieves the value of a cookie from the HttpRequest object

```
Protected Sub Page_Load(sender As Object,  
    e As EventArgs) Handles Me.Load  
    If Not IsPostBack Then  
        If Request.Cookies("UserName") IsNot Nothing Then  
            lblUserName.Text = "Welcome back "&  
                Request.Cookies("UserName").Value & ". "  
        End If  
    End If  
End Sub
```

More example of using cookie

For example, the following code adds a cookie within a controller action:

```
C# Copy  
  
public HttpResponseMessage Get()  
{  
    var resp = new HttpResponseMessage();  
  
    var cookie = new CookieHeaderValue("session-id", "12345");  
    cookie.Expires = DateTimeOffset.Now.AddDays(1);  
    cookie.Domain = Request.RequestUri.Host;  
    cookie.Path = "/";  
  
    resp.Headers.AddCookies(new CookieHeaderValue[] { cookie });  
    return resp;  
}
```

Notice that **AddCookies** takes an array of **CookieHeaderValue** instances.

To extract the cookies from a client request, call the **GetCookies** method:

```
C# Copy  
  
string sessionId = "";  
  
CookieHeaderValue cookie = Request.Headers.GetCookies("session-id").FirstOrDefault();  
if (cookie != null)  
{  
    sessionId = cookie["session-id"].Value;  
}
```

A **CookieHeaderValue** contains a collection of **CookieState** instances. Each **CookieState** represents one cookie. Use the indexer method to get a **CookieState** by name, as shown.

Using the **CookieHeaderValue** class, you can pass a list of name-value pairs for the cookie data. These name-value pairs are encoded as URL-encoded form data in the Set-Cookie header:

C#

Copy

```
var resp = new HttpResponseMessage();

var nv = new NameValueCollection();
nv["sid"] = "12345";
nv["token"] = "abcdef";
nv["theme"] = "dark blue";
var cookie = new CookieHeaderValue("session", nv);

resp.Headers.AddCookies(new CookieHeaderValue[] { cookie });
```

The previous code produces the following Set-Cookie header:

PowerShell

Copy

```
Set-Cookie: session=sid=12345&token=abcdef&theme=dark+blue;
```

The **CookieState** class provides an indexer method to read the sub-values from a cookie in the request message:

C#

Copy

```
string sessionId = "";
string sessionToken = "";
string theme = "";

CookieHeaderValue cookie = Request.Headers.GetCookies("session").FirstOrDefault();
if (cookie != null)
{
    CookieState cookieState = cookie["session"];

    sessionId = cookieState["sid"];
    sessionToken = cookieState["token"];
    theme = cookieState["theme"];
}
```


A procedure that deletes a persistent cookie

```
Private Sub DeleteCookie()  
    Dim NameCookie As New HttpCookie("UserName")  
    NameCookie.Expires = Now.AddSeconds(-1)  
    Response.Cookies.Add(NameCookie)  
End Sub
```

Viewstate example

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Response.Cookies["name"].Value = TextBox1.Text;
    ViewState["v1"] = complex_function(TextBox1.Text.ToString());
    if (Convert.ToInt32(ViewState["v1"].ToString()) > 50)
    {
        TextBox2.Visible = true;
    }
}

protected void Button2_Click(object sender, EventArgs e)
{
    if (TextBox2.Text == "")
    {
        Label2.Text = "Default user: " + ViewState["v1"].ToString();
    }
    else
    {
        Label2.Text = TextBox2.Text + ViewState["v1"].ToString();
    }
}

protected string complex_function(string v1)
{
    int a = Convert.ToInt32(v1);
    string result = (a*a).ToString();
    return result;
}
```