

State Management

Today You Will Learn

- The Problem of State
- View State
- Transferring Information Between Pages
- Cookies

The Problem of State

- How you store information over the lifetime of your application.
- In a traditional Windows program, users interact with a continuously running application.
- 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.

Advantage of Stateless Design:

- 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.

View State

- One of the most common ways to store information is in view state.
- View state uses a hidden field that ASP.NET automatically inserts in the final, rendered HTML of a web page.
- Example: If you change the text of a label, the Label control automatically stores its new text in view state. That way, the text remains in place the next time the page is posted back.

The ViewState Collection

- The ViewState property of the page provides the current view state information.
- This property provides an instance of the StateBag collection class.
- The StateBag is a dictionary collection, which means every item is stored in a separate “slot” using a unique string name.

View State

//The this keyword refers to the current Page object. It's optional.

```
this.ViewState["Counter"] = 1;
```

- When retrieving a value, you use the key name. You also need to cast the retrieved value to the appropriate data type using the casting syntax.
- This extra step is required because the ViewState collection stores all items as basic objects, which allows it to handle many different data types.

```
int counter;
```

```
counter = (int)this.ViewState["Counter"];
```

A View State Example

- A simple counter program that records how many times a button is clicked.

View State

```
public partial class SimpleCounter : System.Web.UI.Page
{
    protected void cmdIncrement_Click(object sender, EventArgs e)
    {
        int counter;
        if(ViewState["Counter"] == null)
        {
            counter = 1;
        }
        else
        {
            counter = (int)ViewState["Counter"] + 1;
        }
        ViewState["Counter"] = counter;
        lblCount.Text = "Counter :" + counter.ToString();
    }
}
```

The code checks to make sure the item exists in view state before it attempts to retrieve it. Otherwise, you could easily run into problems such as the infamous *null reference exception*.

View State

Making View State Secure

- View state information is stored in a single jumbled string that looks like this:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="dDw3NDg2NTI5MDg7Oz4=" />
```

- As you add more information to view state, this value can become much longer. Because this value isn't formatted as clear text.
- The view state information is simply patched together in memory and converted to a **Base64 string**.

Tamper-Proof View State

There are two choices to make View State more secure:

- The view state information is tamperproof by instructing ASP.NET to use a *hash code*.

View State

A **hash code** is sometimes described as a cryptographically strong checksum.

- **Private States:** View state contains some information you want to keep secret, you can enable view state encryption.

- You can turn on encryption for an individual page using the ViewStateEncryptionMode property of the Page directive:

```
<%@Page ViewStateEncryptionMode="Always" %>
```

- you can set the same attribute in a configuration file to configure view state encryption for all the pages in your website:

```
<configuration>
```

```
  <system.web>
```

```
    <pages viewStateEncryptionMode="Always" />
```

```
  </system.web>
```

```
</configuration>
```

To convert “auto” to Always” call `Page.RegisterRequiresViewStateEncryption()`

View State

- There are three choices for view state encryption setting—always encrypt (Always), never encrypt (Never), or encrypt only if a control specifically requests it (Auto).

Retaining Member Variables

- The basic principle is to save all member variables to view state when the Page.PreRender event occurs and retrieve them when the Page.Load event occurs.

Steps:

- A member variable that will be cleared with every postback.
- If IsPostBack Then Restore variables.
- In Page_PreRender event Persist variables.

View State

Disadvantages:

- It stores unnecessary information in view state, it will enlarge the size of the final page output and can thus slow down page transmission times.

Storing Custom Objects

- To store an item in view state, ASP.NET must be able to convert it into a stream of bytes so that it can be added to the hidden input field in the page. This process is called **serialization**.

[Serializable]

```
public class Customer
```

```
{
```

```
private string firstName;
```

View State

- To make your objects serializable, you need to add a Serializable attribute before your class declaration.

`[Serializable]`

- Then it can be stored in View State.

`Customer cust = new Customer("Marsala", "Simons")`

`ViewState["CurrentCustomer"] = cust`

- It is needed to casting the view state object.

`//Retrieve a customer from view state.`

`Customer cust;`

`cust = (Customer)ViewState["CurrentCustomer"]`

You can find more serializable classes in Visual Studio Help.

Transferring Information Between Pages

- One of the most significant limitations with view state is that it's tightly bound to a specific page.
- Two basic techniques to transfer information between pages:
 - cross-page posting
 - The query string

Cross-Page Posting

- A cross-page postback is a technique that extends the postback mechanism, so that one page can send the user to another page, complete with all the information for that page.
- The infrastructure that supports cross-page postbacks is a property named **PostBackUrl**, which is defined by the **IButtonControl** interface and turns up in button controls such as ImageButton, LinkButton, and Button.

Transferring Information Between Pages

- 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.
- Example—A page named `CrossPage1.aspx` that defines a form with two text boxes and a button. When the button is clicked, it posts to a page named `CrossPage2.aspx`.

`CrossPage1.aspx`

```
<asp:Button runat="server" ID="cmdPost"
    PostBackUrl="CrossPage2.aspx" Text="Cross-Page Postback" />
```

Getting More Information from the Source Page

Transferring Information Between Pages

- To get more specific details, such as control values, you need to cast the `PreviousPage` reference to the appropriate page class.

```
public partial class CrossPage2 : System.Web.UI. Page
{
    protected void Page_Load(Object sender, EventArgs e)
    {
        if( PreviousPage != null)
        {
            lblInfo.Text = "You came from a page titled " + PreviousPage.Title;
        }
    }
}
```

The page checks for null reference before attempting to access the `PreviousPage` object. If it's a null reference, no cross-page postback took place.

Transferring Information Between Pages

- You can manually specify the reference by adding the PreviousPageType directive to the .aspx page, that receives the cross-page postback right after the Page directive.

`<%@ PreviousPageType VirtualPath="~/CrossPage1.aspx" %>`

Disadvantage:

- This approach is more fragile because it limits you to a single page class.
- Once you've cast the previous page to the appropriate page type, you still won't be able to directly access the control objects it contains.
- That's because the controls on the web page are not publicly accessible to other classes. You can work around this by using **properties**.

Transferring Information Between Pages

The Query String

- Another common approach is to pass information using a query string in the URL.
- This approach is commonly found in search engines.

<http://www.google.co.in/search?q=organic+gardening>

- The query string is the portion of the URL after the question mark. In this case, it defines a single variable named q, which contains the string organic+gardening.

Advantages of Query String

- It is lightweight.
- It doesn't exert any kind of burden on the server.

Transferring Information Between Pages

Limitations of Query String

- 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 1KB to 2KB). For that reason, you can't place a large amount of information in the query string and still be assured of compatibility with most browsers.

Transferring Information Between Pages

//Go to newpage.aspx. Submit a single query string argument named recordID, and set to 10.

```
Response.Redirect("newpage.aspx?recordID=10")
```

- You can send multiple parameters as long as they're separated with an ampersand (&):

//Go to newpage.aspx. Submit two query string arguments: recordID (10) and mode (full).

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

- It can receive the values from the QueryString dictionary collection exposed by the built-in Request object:

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

A Query String Example

- The program presents a list of entries. When the user chooses an item by clicking the appropriate item in the list, the user is forwarded to a new page. This page displays the received ID number. This provides a quick and simple query string test with two pages.

Transferring Information Between Pages

URL Encoding

- One potential problem with the query string is that some characters aren't allowed in a URL.
- It supports for all characters must be alphanumeric or one of a small set of special characters (including \$-_.+!*'(),).
- Some characters have special meaning.
 - The ampersand (&) is used to separate multiple query string parameters
 - The plus sign (+) is an alternate way to represent a space
 - 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.
- To solve this problem **URL encoding** on text values before you place them in the query string.

Transferring Information Between Pages

- With URL encoding, special characters are replaced by escaped character sequences starting with the percent sign (%), followed by a two-digit hexadecimal representation.

Example: The & character becomes %26

- To perform URL encoding, you use the `UrlEncode()` and `UrlDecode()` methods of the `HttpServerUtility` class.

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

- Use the `UrlDecode()` method to return a URL-encoded string to its initial value.
- ASP.NET automatically decodes your values when you access them through the `Request.QueryString` collection in query strings.

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. Cookies provide another way that you can store information for later use.

Advantage:

- Cookies work transparently without the user being aware that information needs to be stored.
- Cookies can be easily used by any page in your application and even be retained between visits, which allows for truly long-term storage.

Disadvantages:

- Cookies are limited to simple string information.
- Cookies are easily accessible and readable if the user finds and opens the corresponding file.
- Some users disable cookies on their browsers, which will cause problems for web applications that require them.
- Users might manually delete the cookie files stored on their hard drives.

Cookies

Usage

- Both the Request and Response objects (which are provided through Page properties) provide a Cookies collection.
- Retrieve cookies from the **Request** object, and set cookies using the **Response** object.

- To set a cookie, just create a new **HttpCookie** object.

```
HttpCookie cookie = Request.Cookies["Preference"];  
if(cookie == null)  
{  
    cookie = new HttpCookie("Preference");  
}
```

Set the value for Cookie

```
cookie["LanguagePref"] = "English" ;
```

- Adding values to the Cookie

```
cookie["Country"] = "US" ;
```

Cookies

- Attach Cookie to the current web response

```
Response.Cookies.Add(cookie);
```

- A cookie added in this way will persist until the user closes the browser and will be sent with every request.
- Creating a longer-lived cookie, by setting an expiration date.

```
cookie.Expires = DateTime.Now.AddYears(1);
```

- Retrieve cookies by cookie name using the Request.Cookies collection.

```
HttpCookie cookie = Request.Cookies["Preferences"];
```

- Check to see whether a cookie was found with this name.

```
string language;  
if(cookie != null)  
{  
    language = cookie["LanguagePref"];  
}
```

Cookies

- The only way to remove a cookie is by replacing it with a cookie that has an expiration date that has already passed.

```
HttpCookie cookie = new HttpCookie("Preferences");  
cookie.Expires = DateTime.Now.AddDays(-1) ;  
Response.Cookies.Add(cookie) ;
```

A Cookie Example

- To try this example, begin by running the page, entering a name, and clicking the Create Cookie button. Then, close the browser, and request the page again. The second time, the page will find the cookie, read the name, and display a welcome message.



END OF LECTURE