Agenda: State Management Techniques

- Cookies
- Sessions

Understanding Cookies

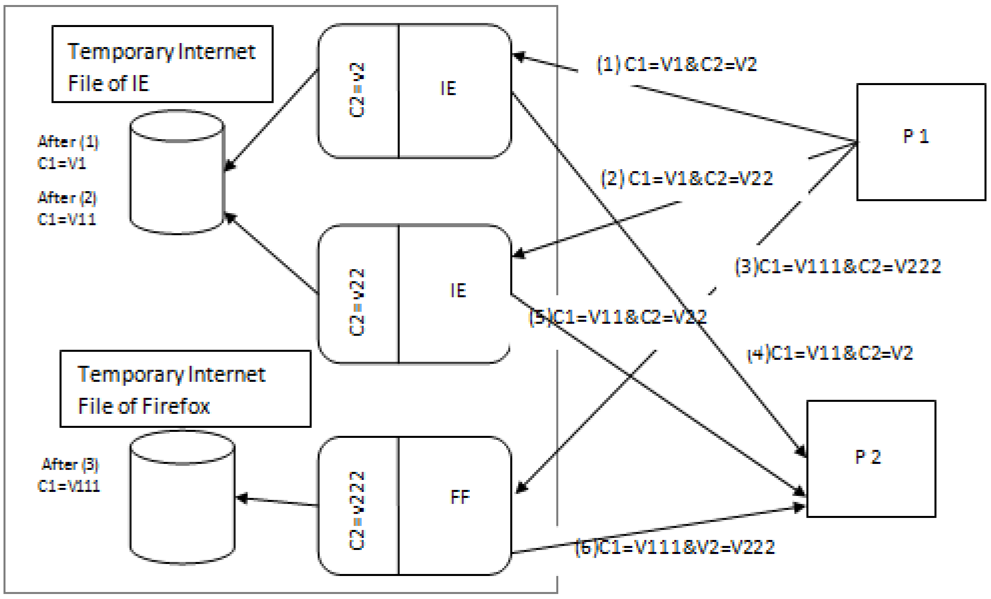
1. Cookies are used for sharing data across requests by same user/client and the data is stored on client/browser machine.
2. With every request the browser automatically includes all the cookies appropriate to that request. The cookie Name-Value pairs are submitted in the form of request header called "**Cookie**" and they are saved on the server in the form of CGI environmental variable (Server Variable) called as **HTTP_COOKIE**.

   HTTP_COOKIE: c1=v1&c2=v2&c3=v3.
3. Cookies generated on server (added to the response) are rendered to the browser in a response header called "**Set-Cookie**".
4. Every response cookie has **Name**, **Value**, **Domain**, **Path**, **Expires** and **Secure** attributes.These attributes are used by browser for deciding whether to include the cookie in the request or not (based on the URL of the request).

Note: Domain, Path, Expires and Secure attributes are not included with a cookie in the request.

<u>Types of Cookies:</u>

1. **Persistent Cookie**: Cookies which are permanantly stored on the client machine in a file. These cookie values are shared by all the browser instances of a particular type running on that machine i.e Cookies of IE cannot be used in Firefox browser but all the instances of IE will share a common set of Persistent cookies.
2. **Non-Persistent Cookie** These Cookies which are temporarily stored in the browser's memory. These are not shared across browser instance and are also called as **Session Cookie**. Every instance of IE will have a different set of Non-Persistent cookies.



Managing Cookies in ASP.NET MVC Application

Request.Cookies is a collection of **HttpCookie** objects contructed for every cookie included in the request.

Response.Cookies is a collection of **HttpCookie** objects created for every cookie to be included in the response so that these cookies can be appended to the existing list of cookies on the client machine.

- If a cookie added to the response is already existing on the client and also if its domain and path attributes match to the new cookie, the old cookie is overwritten with the new cookie.
- The only way of destroying the cookie already with client / browser is to add to the response a cookie with same name and path properties but an **expired date**.

Default values for the properties of Cookie

Value → " " **Path** → "/" **Expires** → (If not set then it's a Non persistent cookie) **Domain** → "<Current domain of the website>" **Secure** → False

Sample Code:

Add the Cookie to Response	To get the cookie from the request

`HttpCookie c = new HttpCookie(txtKey.Text);`
`c.Value = txtValue.Text;`
`c.Path = Request.ApplicationPath;`
`if (chkPersistance.Checked)`
`    c.Expires = DateTime.MaxValue;`
`if (chkSecured.Checked)`
`    c.Secure = true;`
`Response.Cookies.Add(c);`
`}`
	`string s;`
	`HttpCookie c = Request.Cookies[txtKey.Text];`
	`if (c == null)`
	`    s = "Missing";`
	`else`
	`    s = c.Value;`
	`}`

To get all Cookies from the request:	To remove the Cookie

`string s= "";`
`foreach (String key in Request.Cookies)`
`    s += key + " " + Request.Cookies[key].Value + "<br>";`
`}`
	`HttpCookie c = new HttpCookie(key);`
	`c.Expires = DateTime.Now.AddDays(-1);`
	`c.Path = Request.ApplicationPath;`
	`Response.Cookies.Add(c);`
	`}`

Example: Authentication using Cookies

UserProfile Model:

```
public class UserProfile
{
  public string Username { get; set; }
  public string Password { get; set; }
  public bool RememberMe { get; set; }
}
```

Login Controller:

```
public class LoginController : Controller
{
  // GET: Login
  public ActionResult Index()
  {
    ViewBag.ReturnUrl = Request.QueryString["ReturnUrl"];
    UserProfile user = new UserProfile();
    return View(user);
  }

  [HttpPost]
  public ActionResult Index(UserProfile user)
  {
    if (user.Username == user.Password)
    {
      HttpCookie cookie = new HttpCookie("Authcookie");
      cookie.Value = user.Username;
      if (user.RememberMe)
        cookie.Expires = DateTime.MaxValue;
      cookie.Path = Request.ApplicationPath;
      Response.Cookies.Add(cookie);
      string returl = Request.QueryString["ReturnUrl"];
      if (string.IsNullOrEmpty(returl))
        return Redirect("/");
      else
        return Redirect(returl);
    }
    else
    {
      ViewBag.Error = "Invalid Username or Password";
    }
    return View(user);
  }

  public ActionResult Logout()
  {
    HttpCookie cookie = new HttpCookie("Authcookie");
    cookie.Expires = DateTime.Now.AddDays(-1);
```

```
        cookie.Path = Request.ApplicationPath;
        Response.Cookies.Add(cookie);
        return Redirect("/");
    }
}
```

Login/INDEX.CSHTML

```
<h3>Login here</h3>
<div class="alert-warning">@ViewBag.Error</div>

@if (ViewBag.ReturnUrl != null)
{
    Html.BeginForm("Index", "Login", new { ReturnUrl = ViewBag.ReturnUrl });
}
else
{
    Html.BeginForm("Index", "Login");
}
<div class="form-group">
    @Html.Label("username")
    @Html.TextBox("Username", null, new { @class = "form-control" })
</div>
<div class="form-group">
    @Html.Label("Password")
    @Html.Password("Password", null, new { @class = "form-control" })
</div>
<div class="checkbox">
    <label>@Html.CheckBox("RememberMe") Remember Me</label>
</div>
<input type="submit" name="login" value="Login" class="btn btn-success" />
@{Html.EndForm();}
```

Home Controller:

```
private void Authenticate()
{
    HttpCookie cookie = Request.Cookies["AuthCookie"];
    if (cookie == null)
        Response.Redirect("/Login");
}
```

```
public ActionResult About()
{
   Authenticate();
   ViewBag.Message = "Your application description page.";
   return View();
}

public ActionResult Contact()
{
   Authenticate();
   ViewBag.Message = "Your contact page.";
   return View();
}
```

_Layout.cshtml

```
@if (Request.Cookies["Authcookie"] == null)
 {
    <li>@Html.ActionLink("Login", "Index", "Login")</li>
 }
 else
 {
    <li>@Html.ActionLink("Logout", "Logout", "Login")</li>
    <li><a href="#">@Request.Cookies["Authcookie"].Value</a></li>
 }
```

Cookie Dictionary

Every browser restricts the size of each cookie (approx to 4KB) and also the total number of cookies (to 20) which it would accept from a given domain.

A cookie which has many SubKey - Value pairs is called as "**CookieDictionary**".

Note: while using CookieDictionary don't use character "&" in its SubKey or Value.

To add Sub Key value pair to Cookie Dictionary

```
HttpCookie cookie = Request.Cookies[txtDictionaryName.Text];
if (cookie == null)
        cookie = new HttpCookie(txtDictionaryName.Text);
cookie.Values[txtSubKey.Text] = txtValue.Text;
cookie.Expires = DateTime.MaxValue;
Response.Cookies.Add(cookie);
```

To get the list of all sub keys and values from Cookie Dictionary

```
string s = "Cookie Dictionary Not Found";
HttpCookie cookie = Request.Cookies[txtDictionaryName.Text];
if (cookie == null || !cookie.HasKeys)
    return;
s = "";
foreach (string sk in cookie.Values)
   s += key + "-" + cookie.Values[sk] + "<br>";
```

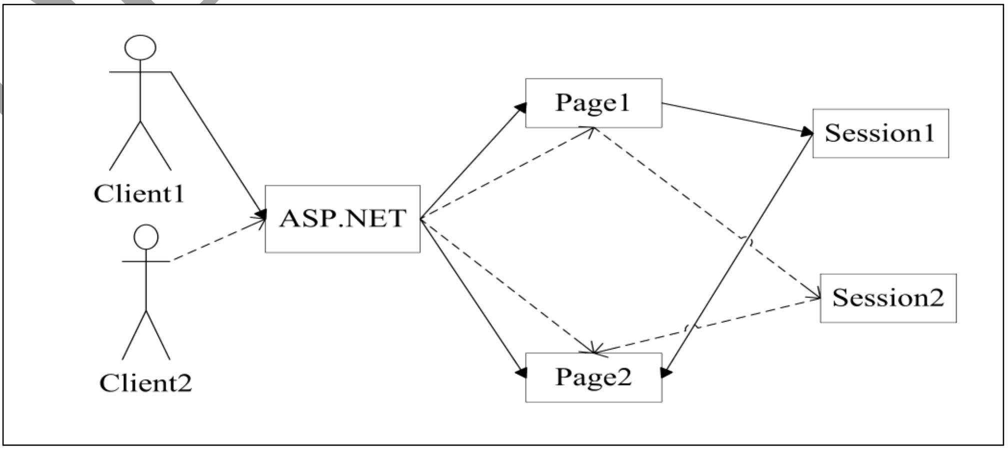
Session Management

A session is an **object on server** having key-value pairs created on the server on behalf of a given client. This object is created on the first request from the client and the same object is reused for all subsequent requests by that client.

The key-value pair is of type string-object and hence in session we can store any type of data unlike cookies where we can only store only strings. Also sessions can store any amount of data and also we can have unlimited number of key-value pairs.

When a new session is created a unique **ID** is assigned to it and the same is rendered in the response as a **non-persistent cookie.** With all subsequent requests, the browser includes this cookie and asp.net uses it for deciding which session object to be used for that client's request.

Time out period is defined for every session. The default value is 20 minutes. If the time after the last request by the client exceeds the timeout period then the session corresponding to that client is automatically destroyed on the server.
A Session is not immediately destroyed if the browser instance is closed on the client. Because the browser doesn't update the server when it is closed thus ASP.NET server keeps the session for the Timeout period even though it is not anymore useful.
Programmatically a session can be destroyed using **Session.Abandon()** but the session i.e. variables and its data would be destroyed after the response of the current page is rendered (but not immediately when that line of code is executed).
Session.RemoveAll() removes all keys in the session but the session object is not destroyed.



Example:

```
public ActionResult Index()
{
    int n = 0;
    if (Session["cnt"] != null)
    {
        n = (int)Session["cnt"];
    }
    n++;
    Session["cnt"] = n;
    return View();
}
```

Session Properties:

Session.IsNewSession: This property is used to check if it is the first request from the client.

Session.Timeout = 30 (default is 20 mins)

Configuring Session In web.config (Write the following after the section <System.Web>:

<sessionState timeout="30" **cookieless**="UseUri" **cookieName**="Demo"**/>**

Cookieless Session: If the Request from the client is received and if the URL doesn't have any SessionID then a new Session is created for the client and the request is redirected to the URL of current page with Session included.

Ex: http://localhost/WebApp/(S(4c31np554tn4hv55tt0zlc3m))/Default.aspx

Note: While giving link to other pages the URL must be framed using "~", this includes the application path and the session id.

If the session is managed using URI, url must be framed as below.

```
        string url = "http://" + Request.Url.Host + ":" + Request.Url.Port +
                    Response.ApplyAppPathModifier(Request.ApplicationPath) + "/Home/Index";
```

What is WebFarm?

- If a single instance of a web application spans over the multiple instances of the webserver running on the **different** machine is then it is called as **WebFarm**.

<sessionState mode="[InProc]**/**StateServer**/**SqlServer" **stateConnectionString="tcpip=localhost:42424**" **/>**

- If mode="InProc", Session variables are stored in the AppDomain (of the web application) of the worker process (aspnet_wp.exe)
- If session is used for a web application on webfarm or webgarden, the sessionState mode must be set to either **StateServer** or **SqlServer**.
- State server is a windows service and can be started using ControlPanel → AdminstrativeTools → Services → ASP.NET StateService (right click and select start).
- The port number on which the state service is running can be changed in windows registry at the following location: *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\aspnet_state\Parameters\Port.*

Static Class

We can add a a class the project and it can then be shared by all the webforms in the web application. By making the members of the class as shared/static we can share data between all the clients.

Step1: Add a class: GlobalSettings.cs

```
public static class GlobalSettings
{
    public static int SessionCounter;
}
```

Step 2: Add to Global.asax

```
void Session_Start(object sender, EventArgs e)
{
      Settings.SessionCounter++;
}
void Session_End(object sender, EventArgs e)
{
      Settings.SessionCounter--;
}
```

Step 3: In Index.cshtml.

```
<div>@GlobalSettings.Data %></div>
```

Step 5: Run the WebApplication from different browser and watch that the value is increasing.