

Mantener el estado en una aplicación ASP.NET

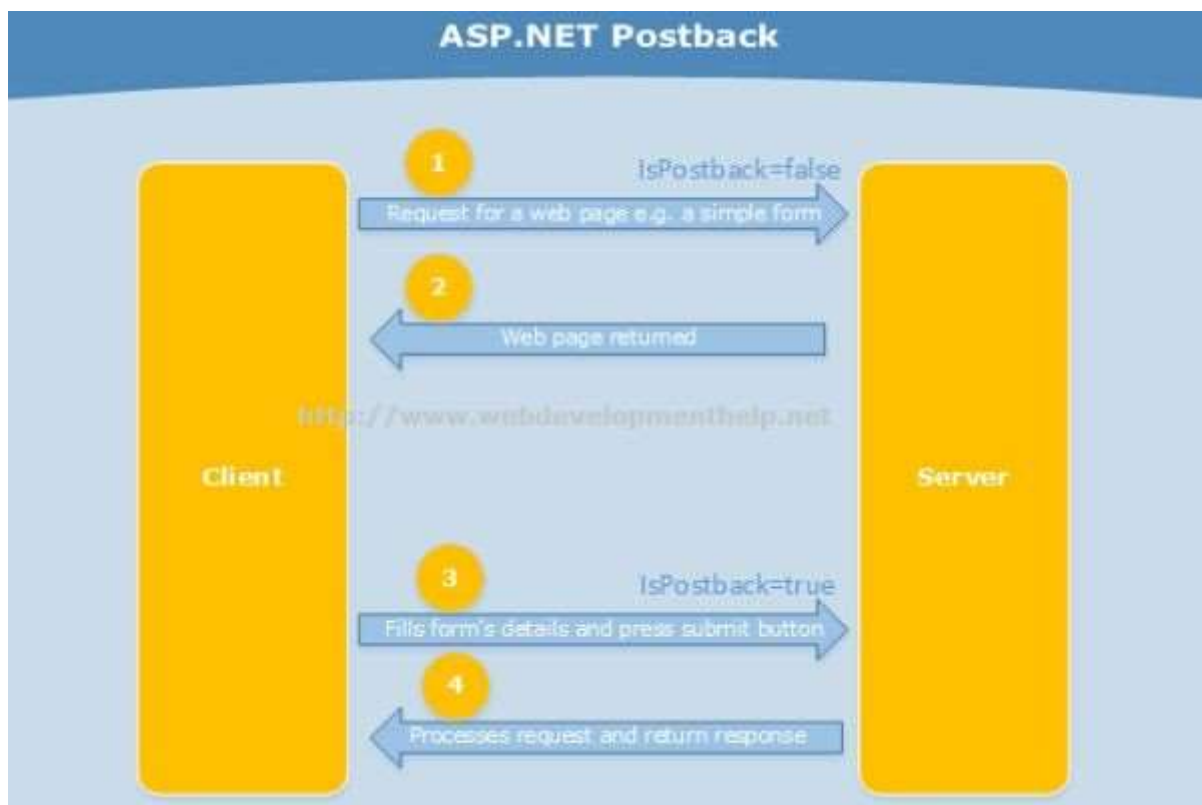
Concepto dePostBack

En el contexto de desarrollo web, un **postback** es una acción **HTTP POST** a la misma página que contiene el formulario. En otras palabras, el contenido del formulario es enviado de nuevo a la misma URL que la del formulario, es un **ida y vuelta** entre el cliente y servidor del mismo formulario.

Los postbacks son vistos comúnmente en formularios de edición, donde el usuario introduce información en un formulario y pulsa un botón "guardar" o "enviar", provocando un postback. Entonces, el servidor actualiza la misma página con la información que acaba de recibir.

Mediante el comando **Page.IsPostBack()** se obtiene un valor que indica si la página se está mostrando por primera vez o si se está cargando como respuesta a un postback.

En todos los casos al ejecutar un evento **Page_Load**, en cada Post a la pagina, se puede validar mediante el método indicado.



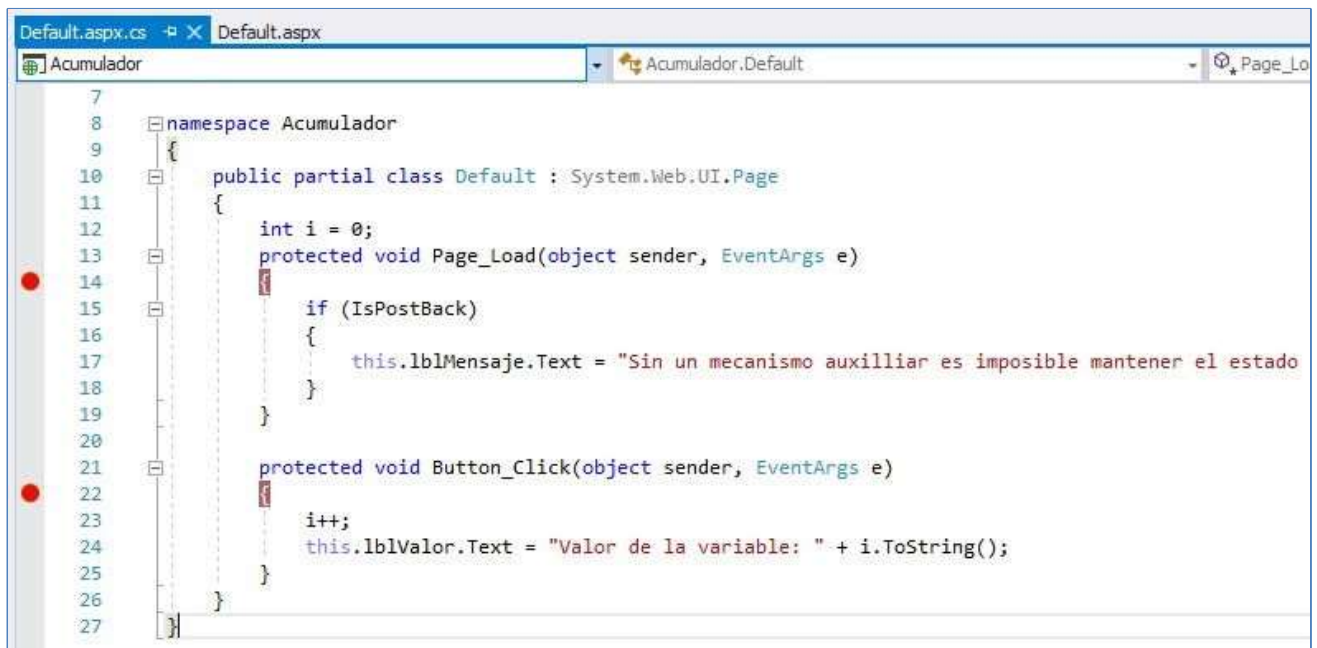
Concepto de estado en ASP.NET

Los formularios ASP.NET no funcionan de igual manera que un Windows Form (formulario de aplicación de escritorio). Por ejemplo, si observamos en el proyecto de ejemplo **Acumulador** (ubicado en la sección de Descargas de esta clase), vemos que la variable que se desea incrementar pierde su valor entre postback y postback. (**Roundtrip** : ida del cliente al servidor y vuelta al cliente). Ejecute la aplicación Acumulador y verá que al hacer click varias veces en el botón, el valor de la variable siempre queda en 1.



Este se debe a que cuando el cliente recibe la página transferida desde el servidor, esta página ya ha sido destruida en el mismo.

Luego pruebe ejecutar nuevamente la aplicación con ayuda del debugger, y en el archivo Default.aspx.cs ubique un breakpoint en el Page_Load y en el Button_Click, siga el código paso a paso entre idas y vueltas al servidor.



```
7
8 namespace Acumulador
9 {
10     public partial class Default : System.Web.UI.Page
11     {
12         int i = 0;
13         protected void Page_Load(object sender, EventArgs e)
14         {
15             if (IsPostBack)
16             {
17                 this.lblMensaje.Text = "Sin un mecanismo auxiliar es imposible mantener el estado";
18             }
19         }
20
21         protected void Button_Click(object sender, EventArgs e)
22         {
23             i++;
24             this.lblValor.Text = "Valor de la variable: " + i.ToString();
25         }
26     }
27 }
```

Asp.Net tiene distintas formas que permiten mantener el estado:

- Archivo **Web.Config**
- **Cookies**
- Objeto **ViewState**
- Variables de sesión
- Variables de aplicación

Archivo Web.Config

Es posible guardar valores de estado en el archivo **Web.Config**. Para ello existe una clave denominada **<configuration>** y dentro de ella una llamada **<appSettings>**.

Por ejemplo:

```
<appSettings>
    <add key="Empresa" value="Acme" />
</appSettings>
```

Podemos leer esta u otras claves ubicadas en el **web.config**, con código similar a:

```
string Empresa =  
System.Configuration.ConfigurationManager.AppSettings["Empresa"];
```

Desde el **web.config** siempre se recuperan valores del tipo string. Si deseamos guardar por ejemplo un valor entero deberemos convertir dicho string a entero.

Es posible también modificar el valor desde el código fuente en tiempo de ejecución de la aplicación:

```
Configuration webConfigApp =  
System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~");  
webConfigApp.AppSettings.Settings["Empresa"].Value = "Acme";  
webConfigApp.Save();
```

Normalmente se guardan en el web.config valores que no deberían cambiar frecuentemente y sobre todo en tiempo de ejecución. El ejemplo típico es un token o contraseña, cadena de conexión, etc. Nos referimos a valores de configuración de una aplicación.

Nota: Existe también el archivo machine.config que es el archivo de configuración general de todos los proyectos de aplicaciones web realizadas en .Net, obviamente por cada aplicación web existe el web.config que es el que sobrescribe lo que tenga el machine.config , por ejemplo, si por omisión en el machine.config hay ciertos tags de configuración para los proyectos como globalization, httpRuntime, trace, configSections, etc., pero posiblemente en un aplicación "x" se necesita una configuración especial y por lo tanto lo que trae el machine.config no sirve, es por ello que se ha de personalizar la configuración en el archivo web.config. Este archivo machine.config, se encuentra dentro de la carpeta de Windows, en la subcarpeta de instalación de la versión del framework, por ejemplo:

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config

Cookies

Son archivos de texto que se alojan en el dispositivo del cliente que consulta la Web.

Las cookies pueden ser de **sesión** o **persistentes**. **Cuidado!!!, información sensible no debería ser colocada en una cookie.** Puesto a que diversas aplicaciones podrían acceder a la misma.

Una cookie de sesión "vive" hasta que finaliza la sesión que la origina. En cambio, una cookie persistente queda almacenada físicamente en el cliente hasta que éste las elimina o estas expiran, ya que es posible especificar en el momento de crearlas el tiempo de vida que tendrá.

Además existen cookies simples y cookies multivalor. Las cookies multivalor pueden almacenar varios valores en una única cookie.

Por ejemplo, el siguiente código crea una cookie de sesión:

```
Response.Cookies["Fecha"].Value = DateTime.Now.ToString();
```

Podemos recuperarla con:

```
Request.Cookies["Fecha"].Value
```

Para crear una cookie persistente especificamos cuando debe expirar, en el atributo **Expires**:

```
Response.Cookies["Fecha"].Value = DateTime.Now.ToString();  
Response.Cookies["Fecha"].Expires = DateTime.Now.AddYears(1);
```

Su valor se recupera igual que una cookie de sesión.

No es posible borrar físicamente una cookie en el cliente. Para lograr que no sean tenidas en cuenta por nuestra aplicación debemos lograr que estas expiren. Por ejemplo, con una instrucción similar a esta:

```
Response.Cookies["Fecha"].Expires = DateTime.Now.AddDays(-1);
```

Una cookie multivalor se crea de la siguiente manera, indicando el nombre de la cookie y otro nombre como si fuera un campo de esa cookie:

```
Response.Cookies["FechaHora"]["Fecha"] = DateTime.Now.Date.ToString();  
Response.Cookies["FechaHora"]["Hora"] = DateTime.Now.Hour.ToString();  
Response.Cookies["FechaHora"].Expires = DateTime.Now.AddYears(1);
```

Para recuperar sus valores:

```
Request.Cookies["FechaHora"]["Fecha"]
```

```
Request.Cookies["FechaHora"]["Hora"]
```

Se recomienda un uso moderado de cookies ya que estas tienen varias desventajas:

- el usuario puede inhabilitar el uso de cookies.
- el usuario puede borrar las cookies cuando desee.
- si cambia de dispositivo, los valores guardados en el dispositivo anterior no estarán disponibles.
- pueden afectar su privacidad.

Objeto ViewState

El objeto **ViewState** básicamente es un campo oculto codificado en base 64 que genera el servidor de ASP.NET para mantener el estado de los controles de la página.

Si observamos el código fuente que llega al cliente de una página ASP.NET encontraremos algo similar a:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="UTbcvPwN9NZEasZGwtQs1LSrs26aGrZS75TeluEu0+0ayTAc1+476H/z/btsJGj3/izAmpkVY
MfY6sQe3jDZfBfQWE0mUr7GB18/i9T+m1/LPp8Goe0I2J10whGARtn/8tgPhvav9DZ3FkK0dwqmJD5jB
55k5LJh/832qKXwgSza1DmpS76AiTVAs3uMc2yL5hn+VoLQhM2IMIP9t4gM3w==" />
</div>
```



```
1
2
3 <!DOCTYPE html>
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head><title>
7
8 </title></head>
9 <body>
10 <form method="post" action="./Default.aspx" id="form1">
11 <div class="aspNetHidden">
12 <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="562D9FqWeBcIBI4NJZpG9sLW/6QPkcXJSNF/p721z2+d1A9cWCRkCvF04tblp+iGtMwzzuyC93/U" />
13 </div>
14
15 <div class="aspNetHidden">
16
17 <input type="hidden" name="__VIEWSTATEGENERATOR" id="__VIEWSTATEGENERATOR" value="CA0B0334" />
18 <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
19 value="pKTnJ7JI5LoU5f4KTLxpqxnMI4FmLQBTEsPnv45+zrA1D3oefvGOyZLWQ8rZmTZ6xSZzHD6EcnFL1l0hsxtjjaZM3TnX+w1Y6vN/iLHxdr2vPOI9u9SBcnmzQIMRAS" />
20 </div>
21 <div>
22 <h1>Acumulador</h1>
23 <hr />
24 <span id="lblValor">Valor de la variable: 0</span>
25 <hr />
26 <input type="submit" name="Button" value="Postback" id="Button" />
27 <hr />
28 <span id="lblMensaje"></span>
29 </div>
30 </form>
31 </body>
32 </html>
```

El servidor recupera dicha información cuando la página regresa luego de un PostBack de tal manera que pueda detectar los cambios de estado que se han producido.

Podemos aprovechar este mecanismo para generar y almacenar nuestros propios valores en el ViewState y recuperarlos cuando deseamos.

Por ejemplo, el siguiente código almacena el estado de un objeto en el **ViewState**:

```
Persona obj = new Persona("José");  
ViewState["Persona"] = obj;
```

Para recuperar el objeto almacenado podemos utilizar:

```
Persona obj = (Persona)ViewState["Persona"];
```

Es importante destacar que el objeto debe ser serializable.

Además hay que tener en cuenta que no es posible compartir el ViewState entre páginas.

Variables de sesión y variables de aplicación

Las **variables de sesión** y las **variables de aplicación** se almacenan en la memoria del servidor.

Las variables de sesión están asociadas a cada sesión (o sea, cada sesión puede mantener sus propias variables), mientras que las variables de aplicación son compartidas por todas las sesiones existentes en un momento dado en la ejecución de la aplicación.

En el siguiente ejemplo de código observamos cómo crear tanto una variable de sesión como una variable de aplicación:

```
Persona Jose = new Persona("José");  
Session["VariableSesion"] = Jose;  
  
Persona Maria = new Persona("Maria");  
Application["VariableAplicacion"] = Maria;
```

Al recuperarlas hay que convertirlas al tipo de datos en que fueron almacenadas:

```
Persona Jose = (Persona)Session["VariableSesion"];  
Persona Maria = (Persona)Application["VariableAplicacion"];
```

Global Application Class

El **Global Application Class** o **Global.asax** es una clase que una vez incorporada a nuestro proyecto nos permite capturar eventos que de otra forma es imposible capturar.

Por ejemplo:

```
protected void Application_Start(object sender, EventArgs e){}  
protected void Session_Start(object sender, EventArgs e) {} protected  
void Session_End(object sender, EventArgs e){}
```

El evento **Application_Start()** ocurre cuando se inicia la aplicación y es útil para inicializar por ejemplo variables de aplicación.

El evento **Session_Start()** ocurre cuando se inicia una sesión y es útil para inicializar variables de sesión.

El evento **Session_End()** ocurre cuando finaliza una sesión en forma explícita o se termina por “time out”, se terminó el tiempo límite de inactividad de la sesión.