



# ASP.NET 3.5

MANTENIMIENTO  
DE ESTADO

CLIPSA



DISTRIBUIDO POR:

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

CLIPSA

# ***Mantenimiento de Estado***

Una aplicación de Windows convencional se ejecuta reservando desde que inicia hasta que cierra parte de la memoria y los recursos del ordenador para mantener sus datos mientras los usuarios trabajan.

En una aplicación Web cada petición de una página al servidor Web provoca la creación de un objeto que representa la página, la ejecución de una serie de eventos, la generación de una respuesta HTML, y la eliminación final de la propia instancia una vez enviada la respuesta al usuario. Esto conlleva que toda la información que concierne al estado de la aplicación y la página se crea y elimina del servidor con cada solicitud que se atiende. El mantenimiento de estado consiste en mantener esa información entre peticiones de páginas.

La no persistencia de la información de estado supone un cambio obligado en la forma de trabajar de las aplicaciones Web frente a las aplicaciones de escritorio convencionales donde la información se mantiene en la memoria del ordenador en todo momento. Sin embargo, también presenta la ventaja de que muchos clientes pueden emplear simultáneamente la aplicación Web, ya que no se reservan recursos en el servidor salvo cuando se realiza una solicitud.

## **1.- ESTADO DE VISTA DE PAGINA ( ViewState )**

Aunque la Web es básicamente un entorno sin estado en el que los datos de una página se crean y destruyen con cada solicitud; ASP.NET introduce mecanismos que permiten preservar información a través de las diferentes solicitudes de páginas. Uno de los mecanismos comunes es el uso del *estado de vista*, o **viewstate**.

El mantenimiento del estado de vista de una página se lleva a cabo utilizando determinados campos ocultos inscritos dentro del código HTML.

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwULLTE0MDM4MzYxMjNkZKhwiwe/C7o2aH3bAw+3AfMSoEDF" />
```

Estos campos permiten al servidor enviar y recuperar los estados de los controles Web desde el navegador del cliente.

En caso de que no deseemos que se mantenga el valor de las propiedades de un determinado control ( estado del control ) entre peticiones; basta con asignar el valor 'false' a su propiedad *EnableViewState*. Todos los controles comparten esta propiedad. Es importante distinguir entre valores y propiedades. Los valores se recuperan entre peticiones por el envío de datos de sus equivalentes HTML al enviar el formulario. No dependen por lo tanto del estado de vista. Son las propiedades de los controles Web que los generaron las que se almacenan en los campos ocultos del estado de vista.

Si se desea desactivar el estado de vista para todos los controles de una página basta asignar el valor false la propiedad *EnableViewState* de la primera línea del fichero de diseño .aspx. Por defecto su valor 'true'.

## MANTENIMIENTO DE ESTADO EN APLICACIONES WEB

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" EnableViewState = "false"%>
```













Si desea desactivarse el mantenimiento del estado de vista para todas las páginas de una aplicación Web, basta añadir la línea en la sección <system.web> del fichero de configuración de la aplicación Web.Config:

```
<pages enableViewState="false" />
```

### 1.1.- La Bolsa de Estado( *System.Web.UI.StateBag* )

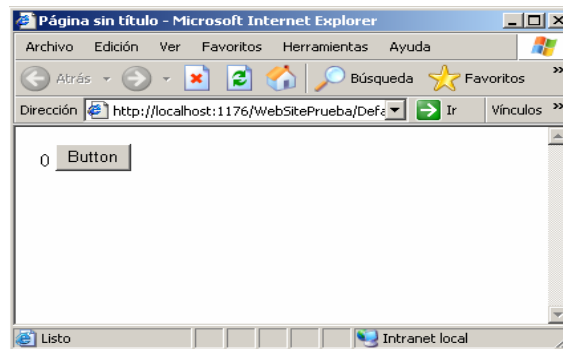
Si deseamos conservar entre peticiones de una página atributos o variables no asociadas a los controles Web es necesario emplear algún mecanismo que permite conservar sus valores, ya que en caso contrario éstos se pierden al finalizar el proceso de cada solicitud. Un buen mecanismo para evitar esta situación es utilizar la *bolsa de estado* para almacenar y recuperar sus valores entre recargas de la página.

La *bolsa de estado* está implementada por la clase *StateBag*. Esta clase define una colección de pares de elementos atributo-valor junto con los métodos para añadir y eliminar elementos de la misma. Cada valor debe almacenarse junto con una 'clave' única que permite posteriormente recuperar el valor:

Nombre	Descripción
 <b>Add</b>	Agrega un nuevo objeto <i>StateItem</i> al objeto <i>StateBag</i> . Si el elemento ya existe en el objeto <i>StateBag</i> , este método actualiza su valor.
 <b>Clear</b>	Quita todos los elementos del objeto <i>StateBag</i> actual.
 <b>Equals</b>	Determina si el objeto <i>Object</i> especificado es igual al objeto <i>Object</i> actual. (Se hereda de <i>Object</i> ).
 <b>Finalize</b>	Permite que un objeto <i>Object</i> intente liberar recursos y realizar otras operaciones de limpieza antes de que el objeto <i>Object</i> sea reclamado por el recolector de elementos no utilizados. (Se hereda de <i>Object</i> ).
 <b>GetEnumerator</b>	Devuelve un enumerador que recorre en iteración los pares de clave/valor de los objetos <i>StateItem</i> almacenados en el objeto <i>StateBag</i> .
 <b>GetHashCode</b>	Actúa como función hash para un tipo concreto. (Se hereda de <i>Object</i> ).
 <b>GetType</b>	Obtiene el objeto <i>Type</i> de la instancia actual. (Se hereda de <i>Object</i> ).
 <b>IsItemDirty</b>	Comprueba un objeto <i>StateItem</i> almacenado en el objeto <i>StateBag</i> para evaluar si se ha modificado desde la llamada a <i>Control.TrackViewState</i> .
 <b>MemberwiseClone</b>	Crea una copia superficial del objeto <i>Object</i> actual. (Se hereda de <i>Object</i> ).
 <b>Remove</b>	Quita el par clave/valor especificado del objeto <i>StateBag</i> .
 <b>SetDirty</b>	Establece el estado del objeto <i>StateBag</i> , así como la propiedad <i>Dirty</i> de cada uno de los objetos <i>StateItem</i> que contiene.
 <b>SetItemDirty</b>	Establece la propiedad <i>Dirty</i> del objeto <i>StateItem</i> especificado en el objeto <i>StateBag</i> .
 <b>ToString</b>	Devuelve una clase <i>String</i> que representa la clase <i>Object</i> actual. (Se hereda de <i>Object</i> ).

Los valores incluidos en la bolsa de estado se mantienen entre recargas de la misma página empleando campos ocultos en el código HTML. Para acceder al objeto bolsa de estado de la página debe emplearse la propiedad *ViewState* definida en la clase *Page*.

Supóngase una página que se desea muestre el número de veces que ha sido recargada. La página sólo consta de un control Label, y un control Button:



El código asociado a la página sería el siguiente:

```
public partial class Default2 : System.Web.UI.Page
{
    private string CONTADOR = "contador";
    protected void Page_Load(object sender, EventArgs e)
    {
        int valor;
        if (!IsPostBack)
        {
            // IsPostBack = FALSE : Primera solicitud de la pagina.
            valor = 0;
            ViewState[this.CONTADOR] = valor;
        } else {
            // IsPostBack = TRUE : Rellamada. La página ya ha sido solicitada
            // anteriormente.
            valor = Convert.ToInt32(ViewState[this.CONTADOR]);
            valor++;
            ViewState[this.CONTADOR] = valor;
        }
        this.lblContador.Text = valor.ToString();
    }
}
```

Cada vez que el usuario pulsa el botón, se produce una solicitud de la página al servidor que provoca la ejecución del evento Load. Si la propiedad *IsPostBack* de la página es 'false' significa que es la primera vez que ésta se solicita y debe insertarse el valor de "contador" en la bolsa de estado puesto que se encontrará vacía. En caso contrario; si la propiedad *IsPostBack* es 'true' indica que se trata de una recarga de la página ( postback ), por lo que el valor del contador ya debe existir y puede recuperarse para incrementarlo.

En todas las operaciones de recuperación de datos de la bolsa de estado resulta imprescindible comprobar siempre si la página es solicitada por primera vez, o si se trata de una recarga. Si se intenta recuperar un valor que no se ha definido previamente se obtiene un valor *null* como resultado. Por otro lado; dado que los valores deben almacenarse junto con una clave es conveniente asociar estas claves a constantes declaradas en el propio código de la página.

La bolsa de estado está optimizada para valores de tipos primitivos tales como números, cadenas, bytes, boléanos, matrices, y tablas. Debe advertirse no obstante; que el almacenamiento en la bolsa de estado de un excesivo volumen de información, produce una reducción en el rendimiento al aumentar el tamaño de los campos ocultos y provocar que la página tarde más en cargarse en el navegador del cliente.

## MANTENIMIENTO DE ESTADO EN APLICACIONES WEB

El siguiente código muestra un ejemplo de guardado y recuperación del valor de atributos miembro de una página entre sus recargas.

```
public partial class PreserveMembers : Page
{
    // Atributo cuyo valor se desea mantener entre recargas.
    private string datos;
    protected void Page_Load(Object sender, EventArgs e)
    {
        if (this.IsPostBack)
        {
            // Si es una recarga, se recupera
            // el valor del atributo en la instancia anterior.
            datos = (string)ViewState["clave"];
        }
    }
    protected void Page_PreRender(Object sender, EventArgs e)
    {
        // Guardado de la variable en la respuesta HTML para el cliente.
        ViewState["clave"] = datos;
    }
}
```

La recuperación de las variables se liga al evento Page\_Load tras comprobar si se trata de una recarga. El guardado se realiza en el evento Page\_PreRender para asegurar que los datos se guardan en el HTML de respuesta al cliente justo antes del envío de la misma.

En la bolsa de estado también es posible almacenar instancias de objetos, a condición de que las clases a las que pertenecen sean serializables.

Para que ASP.NET pueda almacenar en la bolsa de estado la información de un objeto, debe poder convertir dicha información en una secuencia de bytes que pueda codificarse para ser añadida a un campo oculto en el código HTML de la respuesta enviada al navegador del cliente. Para ello es necesario que los objetos sean instancias de clases marcadas como serializables. Para ello basta con añadir la partícula [serializable] al comienzo del código de la clase. Por ejemplo; la siguiente clase Cliente se emplea para identificar objetos clientes de una aplicación Web. Para poder almacenar objetos de esta clase en la bolsa de estado es necesario definir la clase como serializable:

```
[Serializable]
public class Cliente
{
    private string nombre;
    public string Nombre
    {
        get { return nombre; }
        set { nombre = value; }
    }
    private string apellido;
    public string Apellido
    {
        get { return apellido; }
        set { apellido = value; }
    }
    public Cliente(string _nombre, string _apellido)
    {
        Nombre = _nombre;
        Apellido = _apellido;
    }
}
```

Para almacenar un objeto en la bolsa de trabajo basta hacer:

```
// Declaración e instanciación de un objeto Cliente
Cliente obj = new Cliente("Irina", "Kurchatov");
// Almacenamiento en la bolsa de estado bajo la clave "ClienteActual"
ViewState["clienteActual"] = obj;
```



Es importante tener presente que la bolsa de trabajo es una colección que almacena cualquier tipo de valores por lo que siempre devuelve valores de tipo Object. Es responsabilidad del programador conocer y realizar la conversión al tipo del valor recuperado.

```
// Atributo cliente
private Cliente objClienteActual;

protected void Page_Load(Object sender, EventArgs e)
{
    if (this.IsPostBack)
    {
        // Recuperación y conversión del objeto Cliente
        objClienteActual = ViewState["contents"] as Cliente;
    }
}
```

Importante: El uso de la bolsa de estado permite preservar datos entre recargas de una página pero no entre solicitudes de páginas distintas. Para ello deben emplearse otro tipo de técnicas:

## 1.2.- Envío de datos entre páginas distintas ( *Cross-Page Posting* )

El mecanismo *CrossPage Posting* extiende el mecanismo de recarga de páginas visto ( *PostBack* ), permitiendo la transferencia de datos entre dos páginas en donde la primera envía datos a la segunda en vez de a si misma.

En este caso la recuperación de datos asociados a la instancia de la página anterior desde la solicitada se lleva a cabo a través de la propiedad *PreviousPage* de la clase *Page*. Esta propiedad devuelve una instancia de tipo *Page* que representa la instancia de la página desde la que se envían datos a la actual. A partir de esta instancia es posible recuperar valores de la misma invocando propiedades y métodos públicos.

Para realizar una envío de datos a otra pagina se emplea la propiedad *PostBackUrl* miembro de controles Web que producen recargas de la página tales como *Button*, *ImageButton*, y *LinkButton*.

Ejemplo: Supongamos una página ASP.NET llamada *clave.aspx* que contiene dos cajas de texto y un botón. El objetivo es que al pulsar el botón "Button" se invoque la página *datos.aspx* donde se han de mostrar el nombre de usuario y contraseña introducidos:

Usuario:  Clave:

El código de diseño de *clave.aspx* es:

```
<body>
  <form id="form1" runat="server">
    <div>
      Usuario: <asp:TextBox ID="txtUser" runat="server" ></asp:TextBox>
      Clave: <asp:TextBox ID="txtPassword" runat="server" ></asp:TextBox>
      <asp:Button ID="Button1" runat="server" Text="Button"
        PostBackUrl="~/datos.aspx" />
    </div>
  </form>
</body>
```

La propiedad *PostBackUrl* del control Web Button1 indica la página que se solicitará y a la que se transferirán los datos cuando sea pulsado. Recuérdese que sin *PostBackUrl*, el botón provocaría la solicitud de la propia página recargándola.

El código de la página de origen ( *clave.aspx* ) debe además añadir propiedades que permitan obtener desde la página de destino ( *datos.aspx* ) todos los datos a recuperar. En este caso, es necesario crear dos propiedades que devuelvan los valores dispuestos por el usuario en las cajas de texto de la página:

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

public partial class Clave : System.Web.UI.Page
{
    public String usuario
    {
        get { return this.txtUser.Text; }
    }

    public String contraseña
    {
        get { return this.txtPassword.Text; }
    }
}
```

La página destino ( *datos.aspx* ), es una página dotada únicamente de dos controles Web label donde mostrar los datos del usuario y contraseña introducidos en la página previa. El evento *Page\_Load* de esta página inspecciona la referencia de la propiedad *PreviousPage*. Si la página es solicitada poniendo su URL en un navegador no existe página previa y la propiedad retorna un valor *null*.

Insertión bajo la directiva Page en el código HTML de la página: *datos.aspx*.

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

Código del evento *Page\_Load* en el código de la página: *datos.aspx.cs*

```
protected void Page_Load(object sender, EventArgs e)
{
    if (PreviousPage != null)
    {
        // obtención de la referencia a la página anterior
        // del tipo de su Clave
        Clave pag = PreviousPage as Clave;
        this.lblUsuario.Text = pag.usuario;           // Invocación prop. Usuario
        this.lblContraseña.Text = pag.contraseña;     // Invocación prop. Contraseña
    }
}
```

La propiedad *PreviousPage* retorna una referencia de tipo Page. Para poder invocar a las propiedades de la página anterior es necesario realizar la conversión de tipo correspondiente.

## **2.- COOKIES ( *System.Web.HttpCookie* )**

Las *cookies* ( del inglés galletita ) proporcionan un medio para almacenar información en el navegador del usuario y poder recuperarla a continuación desde la misma o cualquier otra página de la aplicación Web. La información de las cookies puede incluso permanecer entre distintas visitas. Por ejemplo; es posible usar las cookies para almacenar las preferencias de un usuario de una aplicación Web, de modo que éstas se recuperen automáticamente la próxima vez que la utilice.

Por ejemplo, si un usuario solicita una página de una aplicación Web y ésta envía junto con la página una cookie que contiene la fecha y hora actuales; el navegador del usuario la almacenará en una carpeta en el disco duro de su equipo. Más adelante si el usuario accede nuevamente a la aplicación Web, el explorador buscará en el disco duro local una cookie asociada a su URL. Si existe alguna, el navegador la envía al servidor junto con la solicitud de la página. A continuación, la aplicación Web recupera los datos de la cookie y es capaz de determinar la fecha y hora en que el usuario visitó el sitio por última vez.

Esta información podría utilizarse para mostrar un mensaje al usuario o comprobar una fecha de caducidad.

### **2.1.- Limitaciones del uso de cookies**

Las cookies poseen unas limitaciones asociadas al navegador que utilice el usuario:

- Tamaño máximo de información: 4096 bytes.
- Número máximo de cookies asociadas al mismo sitio Web: 20
- No todos los navegadores la soportan ( aunque sí la mayoría )
- Pueden ser eliminadas por el usuario en cualquier momento.
- La información contenida puede ser manipulada o extraída con fines perversos. No es confiable.

Por último; resaltar que el uso de cookies puede estar deshabilitada en el navegador Web del usuario por motivos de seguridad.











Debido a estas limitaciones es conveniente no depender exclusivamente de ellas para el correcto funcionamiento de la aplicación Web. Para detectar si un navegador soporta cookies puede consultarse el valor de la propiedad booleana *Cookies* del objeto *HttpBrowserCapabilities* que se obtiene a partir de la propiedad *Browser* del objeto *Request* asociado a toda petición:

```
HttpBrowserCapabilities bc = Request.Browser;
Response.Write("Soporta cookies = " + bc.Cookies.ToString());
```

La propiedad *Cookies* no muestra si la configuración de seguridad impide el uso de cookies, únicamente si el navegador la soporta. Por todo ello; la mejor manera para comprobar el funcionamiento de las cookies en el navegador del usuario es enviar y recuperar una.

### La clase HttpCookie

Las cookies son representadas en ASP.NET mediante la clase *System.Web.HttpCookie*. La siguiente lista muestra sus propiedades miembro:

	Nombre	Descripción
	<a href="#">Domain</a>	Obtiene o establece el dominio al que se asociará la cookie.
	<a href="#">Expires</a>	Obtiene o establece la fecha y la hora de caducidad de la cookie.
	<a href="#">HasKeys</a>	Obtiene un valor que indica si una cookie tiene subclaves.
	<a href="#">HttpOnly</a>	Obtiene o establece un valor que especifica si se puede obtener acceso a una cookie mediante la secuencia de comandos de cliente.
	<a href="#">Item</a>	Obtiene un acceso directo a la propiedad <a href="#">HttpCookie.Values</a> . Se proporciona esta propiedad para que exista compatibilidad con versiones anteriores de las páginas Active Server (ASP).
	<a href="#">Name</a>	Obtiene o establece el nombre de una cookie.
	<a href="#">Path</a>	Obtiene o establece la ruta de acceso virtual que se transmitirá con la cookie actual.
	<a href="#">Secure</a>	Obtiene o establece un valor que indica si se va a transmitir la cookie a través de Secure Sockets Layer (SSL), es decir, sólo a través de HTTPS.
	<a href="#">Value</a>	Obtiene o establece un valor de cookie individual.
	<a href="#">Values</a>	Obtiene una colección de pares de clave/valor que se incluyen en un solo objeto cookie.

Propiedades de la clase *HttpCookie*

Al crear una cookie deben especificarse la propiedad *Name* con un valor único que la identifique para poder recuperarla posteriormente. Si se envían dos cookies con el mismo valor nombre la segunda sobrescribirá a la primera.

Cada cookie puede contener un valor mediante su propiedad *Value*, o un conjunto de subvalores mediante la propiedad *Values*.

La propiedad *Value* admite una cadena de caracteres como valor a almacenar.

```
HttpCookie MyCookie = new HttpCookie("Cookie1");
MyCookie.Value = "abc123";
```

La propiedad *Values* referencia a una colección de tipo *NameValueCollection* que contiene elementos clave-valor.

```
// Declaración e instanciación
HttpCookie MyCookie = new HttpCookie("Cookie1");
// Asignación de valores.
MyCookie.Values["val1"] = "1";
MyCookie.Values["val2"] = "2";
MyCookie.Values["val3"] = "3";
```

También es posible establecer una fecha y hora de caducidad de la cookie. Los navegadores eliminan las cookies caducadas cuando el usuario visita el sitio Web que las creó. La fecha de caducidad de una cookie indica cuando se estima que su información ya no resultará útil para la aplicación Web en caso de que el usuario volviera a solicitar alguna página.

Para ello debe emplearse la propiedad *Expires* que referencia una instancia de tipo *DateTime* indicando la fecha y hora de caducidad:

```
Dim aCookie As New HttpCookie("lastVisit")
aCookie.Value = DateTime.Now.ToString()
// Expira dentro de un día.
aCookie.Expires = DateTime.Now.AddDays(1)
```

Si no se establece fecha de caducidad a una cookie, ésta no será guardada de forma permanente por el navegador del usuario. En su lugar, la cookie se mantendrá como parte de la información de sesión del usuario. Cuando el usuario cierre el navegador Web dando por terminada su sesión; la cookie será eliminada. Este tipo de cookies se denominan 'no persistentes'.

No obstante; debe tenerse en cuenta el hecho de que el usuario puede borrar una cookie en cualquier momento independientemente de su fecha de caducidad.

### 2.2.- Envío y Lectura de Cookies.

ASP.NET incluye colecciones de cookies en las clases *HttpResponse*, y *HttpRequest*. En ambos casos, se trata de instancias de la clase *HttpCookieCollection* referenciadas por la propiedad *Cookies*:

- *HttpRequest.Cookies*: Contiene las cookies transmitidas por el navegador del usuario al servidor en el encabezado de petición.
- *HttpResponse.Cookies*: Contiene las nuevas cookies creadas en el servidor para ser transmitidas al navegador del usuario en el encabezado de respuesta.

Para escribir una cookie en el equipo del usuario basta con añadirla a la colección del objeto *Response* de la página:

```
// Declaración e Instanciación de la nueva cookie
HttpCookie aCookie = new HttpCookie("userInfo");
// Asignación de sus valores.
aCookie.Values["userName"] = "patrick";
aCookie.Values["lastVisit"] = DateTime.Now.ToString();
// Asignación de fecha de expiración
aCookie.Expires = DateTime.Now.AddDays(1);
// Cookie agregada a la colección de cookies para enviar.
Response.Cookies.Add(aCookie);
```

Cuando un explorador realiza una solicitud al servidor, envía las cookies para ese servidor junto con la solicitud. Para recuperarlas basta con acceder a la colección de la propiedad *Cookies* del objeto *HttpRequest* asociada a la instancia de la página solicitada por el usuario.

Es importante siempre comprobar si una cookie existe antes de utilizar su valor. De lo contrario se produce una excepción *NullReferenceException*.

Ejemplo de recuperación de cookie “userInfo” con múltiples subvalores:

```
// Comprobación de existencia de la cookie
if(Request.Cookies["userInfo"] != null)
{
    // Recuperación del objeto cookie
    HttpCookie objCookie= Request.Cookies["userInfo"];
    // Recuperación de subvalor userName
    Label1.Text = Server.HtmlEncode(objCookie.Values["userName"]);
    // Recuperación de subvalor lastVisit
    Label2.Text = Server.HtmlEncode(objCookie.Values["lastVisit"]);
}
```

Dado que los valores de una cookie se almacenan en el equipo del usuario, éstos pueden ser manipulados. El método HtmlEncode se emplea para filtrar esos valores e impedir que un usuario malintencionado inserte código HTML en una cookie y éste se integre en el código HTML de la página.

Al leer el valor o subvalores de una cookie, éstos se devuelven como tipo cadena texto; por lo tanto será necesario realizar una conversión de tipo para obtener el tipo de dato exacto que se almacenó originalmente en la cookie:

```
DateTime dt;
dt = DateTime.Parse(Request.Cookies["userInfo"].Values["lastVisit"]);
```

En el ejemplo anterior se recupera el dato de tipo fecha contenido en el subvalor “lastVisit” de la cookie “userInfo”. El dato debe convertirse mediante una conversión del tipo String devuelto al tipo DateTime.

### 2.3.- Modificar y Eliminar Cookies

Las cookies no pueden ser modificadas directamente. Lo único posible es modificarlas sobrescribiendo sus valores enviando una cookie con el mismo nombre. Supongamos que queremos incrementar en 1, el único valor de la cookie “counter”:

```
int counter;
// Comprobación de que la cookie existe.
if (Request.Cookies["counter"] == null)
    // Inicialización del contador a 0 si no existe.
    counter = 0;
else
    // Recuperación del contador si existe la cookie
    counter = int.Parse(Request.Cookies["counter"].Value);
// Incremento
counter++;
// Envío datos de la nueva cookie sobrescribiendo los anteriores.
HttpCookie newCookie = new HttpCookie("counter", counter);
newCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add( newCookie );
```

De igual manera, las cookies no pueden eliminarse directamente desde el servidor. La única opción es sobrescribirlas con una fecha de expiración anterior de modo que el propio navegador del usuario las elimine.

```
// Recuperación de la cookie original
HttpCookie aCookie = new HttpCookie("counter");
// Modificación de la fecha de expiración
aCookie.Expires = DateTime.Now.AddDays(-1)
// Envío datos de la nueva cookie
Response.Cookies.Add(aCookie)
```



### **3.- ESTADO DE SESION ( *System.Web.HttpSessionState* )**

Una sesión es un objeto de la clase *HttpSessionState* que se crea en la memoria del servidor Web cuando un usuario solicita una página. Este objeto se relaciona con el navegador que la envía, y se mantiene hasta que el usuario cierra el navegador, o transcurre un determinado tiempo sin recibir ninguna petición por su parte.

El objeto sesión permite a la aplicación Web reconocer las peticiones de un determinado usuario, y mantener valores asociados a él a lo largo de las páginas que solicite.

Cada usuario accediendo a la aplicación Web se corresponde con un objeto sesión. Estos objetos se almacenan normalmente en la memoria del servidor Web. La información de cada objeto sesión es intransferible, por lo que desde una página sólo es accesible el objeto sesión correspondiente al usuario que realizó la petición.

Cada objeto sesión posee un identificador único almacenado en la propiedad *SessionID* consistente en un valor alfanumérico de 120 bits. Este valor tiene las siguientes características:

- Cada identificador es único para cada instancia de sesión dentro de un mismo servidor Web
- Cada identificador se genera aleatoriamente no siguiendo un patrón predecible.

El objeto sesión se obtiene a partir del contexto de la página ( propiedad *Content de Page* ). Para acceder al objeto sesión puede emplearse dos formas:

1.- Invocando la propiedad *Session* del objeto *HttpContext* accesible a su vez mediante la propiedad *Context de Page*.

```
String UsuarioID = Context.Session.SessionID;
```

2.- Directamente empleando la propiedad *Sesión* de la clase *Page* que hace de atajo.

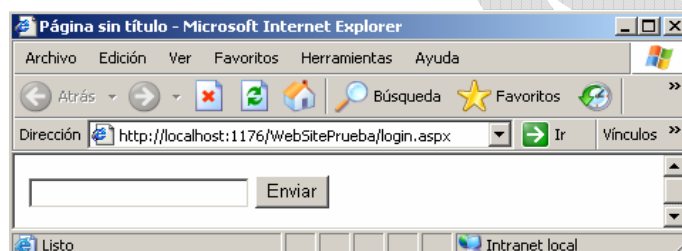
```
String usuarioID = Session.SessionID;
```

### 3.1.- Valores de Sesión

El estado de sesión se define como un conjunto de valores asociados a una determinada sesión. Estos datos se almacenan dentro de una colección interna a la que puede accederse mediante la propiedad *Contents* del objeto *Sesión* de la página.

La propiedad *Contents* hace referencia a una colección de pares clave-valor. Para recuperar un valor debe indicarse su clave, por lo que ésta debe ser única y no se permite almacenar dos valores con la misma clave. Para evitarlo se debe comprobar siempre antes de almacenar un valor, si ya existe otro con la misma clave. Si se solicita el valor de una clave inexistente se obtiene null como resultado.

Supongamos una página llamada login.aspx:

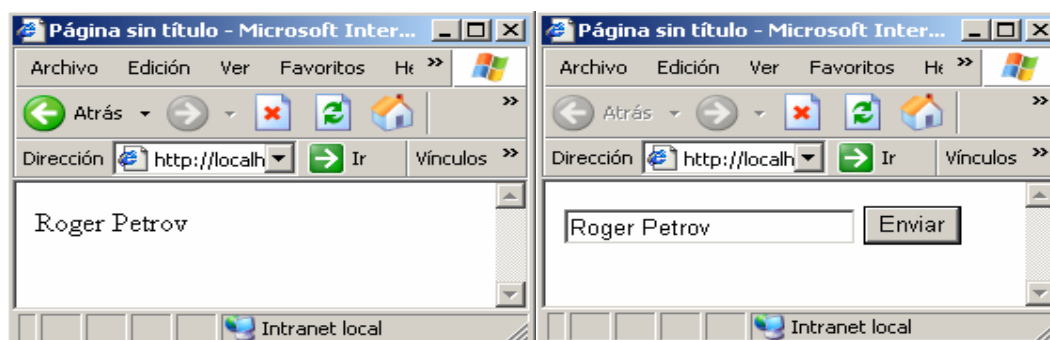


Esta página al pulsarse el botón de Enviar almacena el valor de la caja de texto txtNombre en el objeto sesión bajo la clave "NOMBRE", y redirecciona al usuario a la página user.aspx.

```
public partial class login : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (this.txtNombre.Text.Length > 0)
        {
            Session.Contents.Add("NOMBRE", this.txtNombre.Text);
            Response.Redirect("user.aspx");
        }
    }
}
```

En la página user.aspx, el contenido del valor "NOMBRE" del objeto sesión se muestra en la etiqueta lblNombre.

```
public partial class user : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.lblNombre.Text = (String)Session.Contents["NOMBRE"];
    }
}
```





Al escribirse un nombre en la caja de texto de la página login.aspx; se redirecciona al usuario a user.aspx, mostrándole el nombre escrito en la página anterior. A pesar de tratarse de instancias diferentes de páginas también diferentes; el objeto sesión persiste al tratarse del mismo usuario.

También puede utilizarse directamente el operador de indización para obtener e ingresar valores al estado de sesión;

```
Session["Texto"] = this.txtNombre.Text; // Inserción de valor con clave "Texto"
int i = (int)Session["ID"];           // Obtención del valor de la clave.
```

La clase *HttpSessionState* incluye además una serie de propiedades y métodos que permiten gestionar el almacenamiento de los valores.

- **Count:** Devuelve un valor entero con el número de elementos almacenados en la sesión.
- **IsNewSession:** Devuelve un valor booleano indicando si la sesión ha sido creada por la petición actual o no.
- **Timeout:** Permite fijar el tiempo máximo que puede permanecer activa una sesión sin ninguna petición por parte del usuario.
- **Clear():** Elimina todos los elementos almacenados en la sesión actual.
- **Abandon():** Elimina la sesión actual junto con todos los datos almacenados. Se utiliza para cerrar la sesión de un usuario en la aplicación Web. ( *log out* ).
- **SessionID:** Devuelve el identificador único de sesión asociado a un usuario de la aplicación Web.

### 3.2.- Configuración del estado de sesión

La configuración del estado de sesión se puede controlarse en cada página mediante el atributo *EnableSessionState* de la etiqueta <page> presente en el diseño de la página ( .aspx ).

```
<%@ Page Language="C#" EnableSessionState="True"
    AutoEventWireup="true"
    CodeFile="user.aspx.cs"
    Inherits="user" %>
```

La propiedad *EnableSessionState* admite uno de los tres valores definidos en el enumerador *PageEnableSessionState*:

Nombre de miembro	Descripción
<b>False</b>	El estado de la sesión está deshabilitado.
<b>ReadOnly</b>	El estado de sesión está habilitado, pero no se puede escribir en él.
<b>True</b>	El estado de sesión está habilitado.

## MANTENIMIENTO DE ESTADO EN APLICACIONES WEB

También se puede configurar el estado de sesión de una aplicación Web mediante el fichero Web.Config. Para ello se emplea la etiqueta `<sessionState>` dentro del bloque `<system.web>`:

```
<?xml version="1.0"?>
<configuration>
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <sessionState
      mode="InProc"
      cookieless="useCookies"
      timeout="20"/>
  </system.web>
</configuration>
```

Los atributos disponible para el control del estado de sesión configurables a través de la etiqueta `<sessionState>` son los siguientes:

- **mode:** Especifica si el estado de sesión está habilitado o inhabilitado para todas las páginas ASP.NET de una aplicación. Indica también dónde se almacena el estado de sesión con todos sus valores si está activo. Los posibles valores son:

Valor	Descripción
<b>Custom</b>	El estado de la sesión usa un almacén de datos personalizado para almacenar su información.
<b>InProc</b>	El estado de la sesión se encuentra en proceso con un proceso de trabajo de ASP.NET.
<b>Off</b>	El estado de la sesión está deshabilitado.
<b>SQLServer</b>	El estado de la sesión utiliza una base de datos de SQL Server fuera de proceso para almacenar la información de estado.
<b>StateServer</b>	El estado de la sesión utiliza el servicio de estado de ASP.NET fuera de proceso para almacenar la información de estado.

- **sqlConnectionString** ( sólo *mode* = "SQLServer" ): Especifica una cadena de conexión a un servidor de SQL Server que almacene en una base de datos los estados de sesión:

```
sqlConnectionString = "data source=127.0.0.1;userid=sa;password="
```

Por defecto la base de datos utilizada para el almacenamiento del estado de sesión es ASPState, y los estados se guardan en una tabla indexada por el identificador SessionID.

- **sqlCommandTimeout** ( sólo *mode* = "SQLServer" ): Cantidad de segundos que un comando SQL enviado a la base de datos que mantiene los estados de sesión puede tardar en completarse antes de ser cancelado.
- **stateConnectionString** ( sólo *mode* = "StateServer" ): Especifica el nombre o IP y puerto del servidor empleado para el almacenamiento del estado de sesión. Para que funcione, el servidor indicado debe tener ejecutándose el servicio ASP.NET State.

```
stateConnectionString="tcpip=127.0.0.1:42424"
```

## MANTENIMIENTO DE ESTADO EN APLICACIONES WEB

- **stateNetworkTimeout** ( sólo *mode* = "StateServer" ): Especifica en minutos el tiempo máximo de inactividad permitido de la conexión TCP/IP con el servidor de estado de sesión antes de cancelar la petición.
- **timeout**: Especifica en minutos la cantidad máxima de tiempo sin actividad que el servidor mantiene el estado de sesión para un usuario antes que expire. El valor predeterminado es de 20 minutos.
- **cookieless**: Las cookies se utilizan para almacenar en el navegador del cliente el identificador de sesión que le corresponde, y poder así reconocer después la sesión que corresponde a cada petición. Este parámetro determina cómo se utilizan las cookies para mantener el estado de sesión entre peticiones. Los posibles valores son:

Valor	Descripción
<b>AutoDetect</b>	ASP.NET determina si el explorador o el dispositivo que realizó la solicitud admite cookies. Si admite cookies, <b>AutoDetect</b> las utilizará para conservar los datos del usuario; en caso contrario, se utilizará un identificador en la cadena de consulta. Si admite el uso de cookies pero éstas están deshabilitadas, la característica que realiza la solicitud seguirá utilizándolas.
<b>UseCookies</b>	Las cookies conservan los datos del usuario sin tener en cuenta si el explorador o el dispositivo las admite.
<b>UseDeviceProfile</b>	ASP.NET determina si se pueden utilizar cookies en función del valor de <a href="#">HttpBrowserCapabilities</a> . Si el valor de <b>HttpBrowserCapabilities</b> indica que el explorador o el dispositivo admite cookies, se utilizan cookies; de lo contrario, se utiliza un identificador en la cadena de consulta.
<b>UseUri</b>	La característica que realiza la llamada utiliza la cadena de consulta para almacenar un identificador sin tener en cuenta si el explorador o el dispositivo admite cookies.

El siguiente esquema muestra todos los atributos de la etiqueta <sessionState> y sus valores posibles:

```
<sessionState
  mode="[Off|InProc|StateServer|SQLServer|Custom]"
  timeout="minutos para expirar sesión"
  cookieName="nombre identificador de cookie de sesión"
  cookieless="[true|false|AutoDetect|UseCookies|UseUri|UseDeviceProfile]"
  regenerateExpiredSessionId="[True|False]"
  sqlConnectionString="sql connection string"
  sqlCommandTimeout="number of seconds"
  allowCustomSqlDatabase="[True|False]"
  useHostingIdentity="[True|False]"
  stateConnectionString="tcpip=server:port"
  stateNetworkTimeout="número de segundos"
  customProvider="nombre de proveedor">
  <providers>...</providers>
</sessionState>
```

Para más detalles consultar el MSDN: [http://msdn.microsoft.com/es-es/library/h6bb9cz9\(VS.80\).aspx](http://msdn.microsoft.com/es-es/library/h6bb9cz9(VS.80).aspx)

### 3.3.- Administración del estado de sesión ( *Mode* )

ASP.NET proporciona diferentes formas de administrar el estado de sesión dependiendo de la configuración del Web.Config. Las opciones son las siguientes:

1. **Administración no habilitada:** Equivale a desactivar el estado de sesión en la aplicación Web. Esto provoca una mejora del rendimiento ya que las páginas no tienen que cargar ni almacenar el estado de sesión. Sin embargo; se pierde la capacidad de asociar un conjunto de datos a las peticiones de un determinado usuario.
2. **Guardado de sesión en proceso:** Los datos de sesión se almacenan en la memoria del proceso que ejecuta las peticiones del usuario. Esta es la técnica predeterminada por ASP.NET. Su ventaja es que es un sistema rápido y sencillo; pero dependiente del IIS y del servidor en el que se ejecuta. Si se produce un reinicio del IIS o del servidor; se pierde toda la información de las sesiones en curso.
3. **Guardado de sesión en un servidor de estado.** Esta opción permite que ASP.NET canalice todas las acciones de administración de sesión hacia un servicio independiente en otra máquina. Esta técnica es compatible con granjas de servidores que sirven la misma aplicación Web y comparten la información de sesiones en curso.
4. **Guardado de sesión en base de datos.** Esta opción permite que ASP.NET almacene y administre los estados de sesión empleando una base de datos en un servidor de SQL Server previamente configurado. Para crear las bases de datos necesarias pueden emplearse los scripts de SQL contenidos en **C:\Windows\Microsoft.NET\Framework\v2.0.50727**. La herramienta **asp\_regsql.exe** permite configurar las bases de datos necesarias automáticamente.

### 3.4.- Seguimiento del estado de sesión ( *Cookieless* )

Cuando una aplicación Web necesita utilizar el estado de sesión es necesaria una estrategia de seguimiento que permita conocer el origen de las peticiones para poder identificar a un cliente determinado. ASP.NET ofrece tres posibles estrategias para realizar este seguimiento:

1. **Seguimiento mediante Cookies:** Esta es la opción predeterminada por ASP.NET. Por cada usuario se genera un identificador de sesión complejo que se almacena en una cookie en el navegador del cliente. Cada vez que el usuario realiza una petición el servidor obtiene la identificación de sesión del cliente almacenada en la cookie y recupera su información para el código de la aplicación. Estas informaciones pueden visualizarse activando la función de traceo en el Web.Config.

#### Detalles de la solicitud

Id. de sesión:	pblrkzah12x33uuhkxmowi2m	Tipo de petición:	GET
Hora de la solicitud:	17/06/2009 11:26:21	Código de estado:	200
Codificación de la solicitud:	Unicode (UTF-8)	Codificación de la respuesta:	Unicode (UTF-8)

#### Colección de cookies de solicitud

Nombre	Valor	Tamaño
ASP.NET_SessionId	pblrkzah12x33uuhkxmowi2m	42

*Información de traza de una página ASP.NET mostrando identificador de sesión del usuario.*

2. **Seguimiento mediante URL:** Esta técnica consiste en seguir el estado de sesión incrustando el identificador de sesión del usuario como parte de la URL de petición de manera que el servidor puede extraerla al recibir cada petición y reconocer al usuario. Esta técnica es útil como alternativa al seguimiento en caso de que las cookies estén deshabilitadas en el navegador del cliente. Sin embargo, es bastante insegura ya que bastaría cortar y pegar la URL del navegador en otro para usurpar la identidad del usuario.
3. **Seguimiento por detección automática:** En este caso ASP.NET detecta al ejecutarse si el navegador del cliente tiene activadas las cookies. Si están activadas, el identificador de sesión se almacenará en una cookie en el navegador del cliente. En caso contrario; el identificador se almacenará en la URL.
4. **Aplicación de perfiles de dispositivo:** Esta opción hace ASP.NET compruebe el valor de la propiedad *SupportsRedirectWithCookie* del objeto *HttpBrowserCapabilities* para elegir entre emplear cookies o la propia URL para transmitir el identificador de sesión del usuario.

### 3.5.- Diferencias entre estado de sesión y estado de vista.

A la hora de decidir que estado emplear para retener la información necesaria entre peticiones de páginas es importante tener presente las siguientes cuestiones:

- 1.- El estado de vista solo permite retener los datos entre solicitudes de la misma página. El estado de sesión se mantiene desde la primera hasta la última petición del usuario independientemente de la página solicitada.
- 2.- El estado de vista se mantiene a través del cliente codificando los datos en las respuestas HTML, para recuperarlos en las recargas siguientes sin consumir memoria del servidor. El estado de sesión se almacena íntegramente en la memoria del servidor sin intervención del navegador del usuario.
- 3.- El estado de vista sólo almacena eficazmente tipos de datos primitivos, mientras que en el estado de sesión se puede almacenar cualquier tipo de dato sin problemas.
- 4.- El estado de vista son más vulnerables puesto que pueden ser modificados maliciosamente interviniendo el código HTML de respuesta. El estado de sesión no puede ser visto ni modificarse desde fuera del servidor.

CLIPSA

#### **4.- ESTADO DE APLICACIÓN ( *System.Web.HttpApplication* )**

El estado de aplicación hace referencia a un conjunto de datos que son accesibles desde cualquier página y sesión de la aplicación Web. El estado de aplicación permite almacenar valores de forma semejante al estado de sesión, sin embargo; sus valores son accesibles para todas las páginas y usuarios de la aplicación Web, y permanecen mientras el servidor continúe activo.

El estado de aplicación está representado por una instancia de la clase *System.Web.HttpApplication* accesible a través de la propiedad *Application* de la clase *Page*, o a través del contexto de la propiedad *Application* del contextode propia página accesible por la propiedad *Context* de la clase *Page*:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Acceso a objeto aplicación a través del contexto de la página
    HttpApplicationState application = this.Context.Application;

    // Acceso a estado aplicación a través de propiedad Application de la página.
    HttpApplicationState application = this.Application;
}
```

El estado de aplicación se mantiene en la memoria del servidor hasta que se reinicia el servidor Web ( IIS ), o se editan los archivos de configuración Web.Config o global.asax. Este “reinicio” es transparente a los usuarios ya que se lleva acabo una vez completadas todas las solicitudes pendientes.

##### **4.1.- Valores de aplicación**

El estado de aplicación permite almacenar elementos de pares clave-valor en una colección compartida globalmente para todas las páginas y usuarios de la aplicación Web. Todas las páginas y usuarios acceden a la misma colección.

Al igual que en el caso de sesión, la colección no admite dos valores con la misma clave, por lo que debe comprobarse si la clave existe antes de insertar un nuevo elemento. Por otro lado; si se solicita un valor cuya clave es inexistente se retorna un valor null.

La colección de valores de aplicación es accesible a través de la propiedad *Contents* del objeto *Application*:

```
// Inserta valor numérico con clave "ID"
this.Application.Contents.Add("ID", 34093);

// Recupera valor numérico con clave "ID"
int id = (int)Application.Contents["ID"];
```

También puede accederse a la colección empleando el indizador y la clave correspondiente al valor:

```
// Inserta valor numérico con clave "ID"
this.Application["ID"] = 34093;

// Recupera valor numérico con clave "ID"
int id = (int)Application["ID"];
```

Supóngase que tenemos una página principal en nuestra aplicación Web en la que queremos mostrar el número de personas que la han visitado:

```
public partial class user : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        int numVisitas;
        // Comprobar si ya existe el valor en el estado de aplicacion
        if (Application.Contents["VISITAS"] == null) {
            numVisitas = 0;
        } else {
            // Recuperacion de la variable de aplicación "VISITAS"
            numVisitas = (int)Application.Contents["VISITAS"]+1;
        }
        // Reasigno valor a la variable de aplicación "VISITAS"
        Application.Contents["VISITAS"] = numVisitas;
        this.Label1.Text = Application.Contents["VISITAS"].ToString();
    }
}
```

En este ejemplo, cada vez que se solicite la página se producirá un incremento de la variable "VISITAS" del estado de aplicación.

Primero se comprueba si ya existe la variable, ya que podría ser la primera solicitud desde el inicio de la aplicación Web y no haber sido creada por ninguna petición anterior. Si existe lo recupera, la incrementa, y la vuelve a reasignar. La siguiente petición ( sea el mismo usuario o no ) que se reciba accederá a este mismo valor para incrementarlo nuevamente.

### 4.2.- Concurrencia en el estado de aplicación

El estado de aplicación y sus valores son compartidos por todas las solicitudes que se hacen a la aplicación Web. Esto hace posible que dos o más solicitudes de varios usuarios puedan acceder y manipular al mismo tiempo los valores del estado de aplicación. Esto se denomina concurrencia.

La concurrencia no es un problema si sólo se leen valores; el problema sobreviene cuando varias páginas intentan modificar al mismo tiempo un mismo valor del estado de aplicación.

En el ejemplo anterior, podría llegar a suceder que dos usuarios accedan al mismo tiempo a la página principal de manera que las dos solicitudes obtengan el mismo valor, lo incrementen, y lo añadan. El resultado es que se contabilizaría una única visita cuando realmente han sido dos.

```
protected void Page_Load(object sender, EventArgs e)
{
    int numVisitas;
    // Comprobar si ya existe el valor en el estado de aplicacion
    if (Application.Contents["VISITAS"] == null) {
        numVisitas = 0;
    } else {
        // Recuperacion de la variable de aplicación "VISITAS"
        numVisitas = (int)Application.Contents["VISITAS"]+1;
    }
    // Reasigno valor a la variable de aplicación "VISITAS"
    Application.Contents["VISITAS"] = numVisitas;
    this.Label1.Text = Application.Contents["VISITAS"].ToString();
}
```



Para evitar esta situación se puede bloquear el estado de aplicación. Para ello se emplean los métodos *Lock* y *Unlock*.

```
protected void Page_Load(object sender, EventArgs e)
{
    Application.Lock();
    int numVisitas;
    // Comprobar si ya existe el valor en el estado de aplicación
    if (Application.Contents["VISITAS"] == null) {
        numVisitas = 0;
    } else {
        // Recuperación de la variable de aplicación "VISITAS"
        numVisitas = (int)Application.Contents["VISITAS"]+1;
    }
    // Reasigno valor a la variable de aplicación "VISITAS"
    Application.Contents["VISITAS"] = numVisitas;
    Application.Unlock();
    this.Label1.Text = Application.Contents["VISITAS"].ToString();
}
```

Supóngase dos usuarios A y B. El usuario A solicita la página del ejemplo que bloquea ( *Application.Lock()* ) el estado de aplicación para incrementar la variable VISITAS. Al mismo tiempo otro usuario B solicita la misma página, pero al intentar utilizar al estado de aplicación quedará detenido hasta que la petición del usuario A lo desbloquee ( *Application.Unlock()* ).

Aunque el uso de los métodos *Lock* y *Unlock* asegura que no se corrompan los datos, bloquear la aplicación normalmente produce una reducción del rendimiento de la misma. Se recomienda por lo tanto, que la información que se almacene en el estado de aplicación sea leída una vez que todos los valores han sido cargados, y que sean raramente modificados.

### 4.3.- Consideraciones sobre el uso del estado de aplicación

Al emplear el estado de aplicación para almacenar información deben tenerse en cuenta las siguientes consideraciones:

#### Escalabilidad:

El problema de la concurrencia en el estado de aplicación tiene un efecto directo sobre la relación entre tamaño y rendimiento. A menos que los datos del estado de aplicación sean únicamente de lectura, se producirá una reducción del rendimiento según aumente el número de peticiones simultáneas que reciba la aplicación. Esto es debido al bloqueo del estado de una petición y la esperar de las demás.

#### Memoria:

Cada dato almacenado en el estado de aplicación consume memoria, y lo mismo sucede con cada sesión. Es por ello que debe tenerse en cuenta el requerimiento de memoria en el servidor que puede necesitar la aplicación Web dependiendo del volumen de solicitudes que puede recibir.

#### Persistencia:

Los datos del estado de aplicación no se mantienen si se detiene la aplicación Web, o se reinicia el servidor. Si es necesario que el estado de aplicación se mantenga indefinidamente, debe ser almacenado en una base de datos o un archivo de datos permanentes.

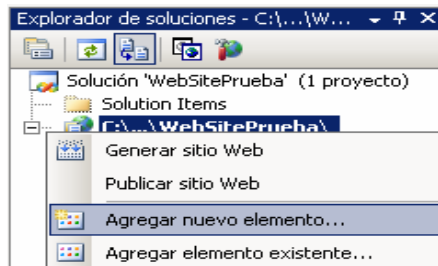
CLIPSA

## 5.- EL FICHERO DE CODIGO GLOBAL ( *global.asax* )

El archivo *global.asax* es un archivo de código global para toda la aplicación Web, en el que se pueden declarar funciones de atención a eventos asociados a los estados de sesión y aplicación.

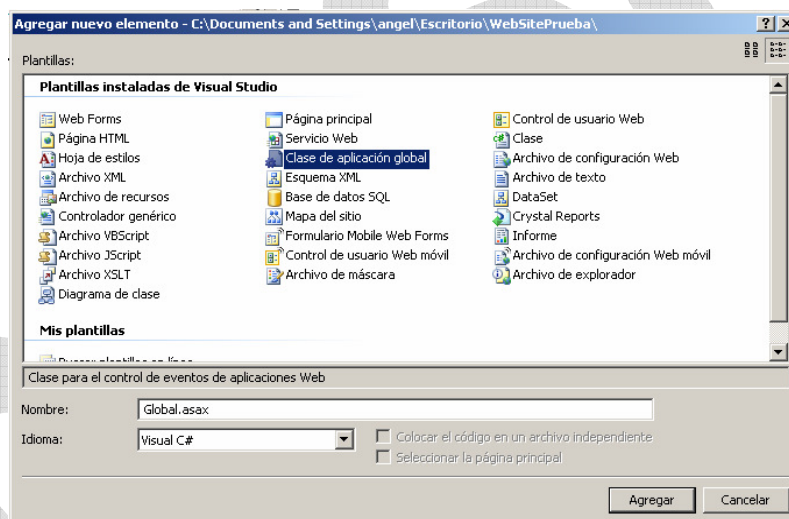
Cada aplicación ASP.NET sólo puede tener un único archivo *global.asax*. Este archivo reside normalmente en el directorio virtual raíz de la aplicación Web.

### 5.1.- Creación del fichero *global.asax*



Para crear un fichero *global.asax* en el IDE de Visual Studio 2005, debemos seleccionar el icono del proyecto Web en la ventana del Explorador de Soluciones, y seleccionar a continuación la opción “Agregar Nuevo Elemento” de su menú contextual.

Se abrirá entonces la ventana “Agregar nuevo elemento”, donde debemos seleccionar la opción “Clase de Aplicación Global”.



El fichero resultante ofrece el siguiente código. Obsérvese su parecido con los ficheros de diseño de las páginas ASP.NET ( ficheros .aspx ), pero excluyendo el código HTML.

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {
        // Código que se ejecuta al iniciarse la aplicación
    }
    void Application_End(object sender, EventArgs e)
    {
        // Código que se ejecuta cuando se cierra la aplicación
    }
    void Application_Error(object sender, EventArgs e)
    {
        // Código que se ejecuta al producirse un error no controlado
    }
    void Session_Start(object sender, EventArgs e)
    {
        // Código que se ejecuta cuando se inicia una nueva sesión
    }
    void Session_End(object sender, EventArgs e)
    {
    }
</script>
```

*Código predeterminado de fichero *global.asax* generado por el IDE Visual Studio 2005*

### 5.2.- Eventos del estado de aplicación y sesión.

Cuando la aplicación Web recibe la primera petición de usuario se ejecuta el método *Application\_Start*. Este método suele aprovecharse para la inicialización de variables y objetos globales necesarios para la aplicación Web.

```
void Application_Start(object sender, EventArgs e)
{
    // Código que se ejecuta al iniciarse la aplicación
    Application["strConnectionString"] = "SERVER=Zeus;DATABASE=Pubs;UID=sa;PWD=secret";
    string[] tables = { "MODEL", "TYPE", "POWER" };
}
```

- El método *Application\_End* es ejecutado por el sistema cuando todas las sesiones han concluido, o se realizan modificaciones en el fichero global.asax; en cuyo caso se finaliza y reinicia el estado de aplicación automáticamente.
- Cuando la aplicación recibe una petición proveniente de un usuario nuevo se ejecuta el método *Session\_Start* coincidiendo con la creación de un nuevo objeto sesión correspondiente. Este método suele emplearse para inicializar las variables de sesión que se utilizan en la aplicación.
- El método *Session\_End* se ejecuta cuando una sesión termina al cerrar el usuario el navegador, o bien expira cuando no se recibe ninguna petición en un margen de tiempo.
- El método *Application\_Error* se inicia al producirse una excepción no controlada en la aplicación Web. Esto permite capturarla en último extremo y tratarla evitando que la aplicación se detenga.

## **EJERCICIOS**

**1.-** Crea una aplicación que conste de una página HTML ( `index.html` ) provista de un formulario en el que se solicite el nombre, apellidos, edad, y estado civil del usuario. El formulario deberá enviar la información contra una página web ASP.NET ( `datos.aspx` ) que mostrará la información. Observa la diferencia entre los métodos de transferencia GET y POST

**2.-** Crear una aplicación web que muestre el mismo formulario del ejercicio anterior en una página ASP.NET ( `index.aspx` ), y después envíe la información a otra página ( `datos.aspx` ) que mostrará la información.

**3.-** Crea una página Web ASP.NET que muestre la IP del usuario y el número de visitas que ha recibido hasta el momento.

**4.-** Crea una página que solicite al usuario mediante un formulario un nombre de usuario y una contraseña. El nombre de usuario inicialmente será "ALUMNO" y la contraseña "CIPSA".

Si el usuario introduce correctamente el nombre de usuario y contraseña se le mostrará una página de bienvenida ( `welcome.aspx` ). Si se introduce mal la contraseña se le indicará el error coloreando de color rojo la caja de texto de contraseña. Si se introduce mal el nombre de usuario se mostrará una página de error ( `error.aspx` ).

Por seguridad, el usuario no debe poder ver en el navegador la URL de la página de bienvenida cuando acceda a ella.

**5.-** Modificar la aplicación anterior permitiendo que un usuario acceda como administrador a la aplicación. Para ello deberá introducir el nombre de usuario "ADMINISTRADOR" y la contraseña "ASPADMIN".

El administrador de la aplicación debe tener acceso a una página de control ( `control.aspx` ) donde deberán mostrarse un listado con la hora de todos los accesos a la aplicación desde el inicio de la misma, y el total de los mismos. La página de control deberá mostrarse un formulario que permite cambiar el nombre de usuario y contraseña para los usuarios.

**6.-** Crea una aplicación Web que muestre un formulario solicitando el nombre y edad al usuario la primera vez que entra. Una vez que el usuario ya ha entrado al menos una vez a la aplicación, ésta le reconocerá mostrando un saludo indicando su nombre, edad, y la fecha de la última vez que entró.

**7.-** Crea una aplicación Web que simule el funcionamiento de una pequeña tienda virtual con cesta de la compra.

Para ello crea una página principal en la que se muestre al usuario una serie de artículos en una lista. Cada artículo debe tener asociado un número de unidades en stock y un precio. El usuario podrá seleccionar una cantidad máxima de unidades del artículo a adquirir según las unidades existentes en stock. La página debe contar con dos botones: *Añadir Compra* y *Finalizar Compra*.

Si el usuario pulsa *Añadir Compra*; se le mostrará al usuario una página de confirmación en la que se le mostrará el artículo, las unidades, y el importe de la compra. Si el usuario confirma la adquisición, ésta se añade definitivamente al carrito de la compra y se retorna a la página principal para que pueda continuar adquiriendo productos. Si el usuario la rechaza se retorna sin más a la página principal.

Si el usuario pulsa *Finalizar Compra* en la página principal, se le mostrará una página con el listado de los artículos, cantidades, e importes incluyendo el total de todo lo adquirido.

Debe tenerse en cuenta que varios usuarios pueden acceder y emplear al mismo tiempo la aplicación Web.

**8.-** Crear una aplicación Web que simule la gestión de unas cuentas bancarias en una sucursal. De cada cuenta bancaria se almacena el DNI del titular, su nombre y apellidos, y el saldo de la misma.

La aplicación debe permitir añadir una cuenta nueva, comprobando que no exista ya una con el mismo DNI. Eliminar una cuenta ya existente o revisar sus datos mediante el DNI, permitiendo a su vez ingresar o extraer dinero. La aplicación no debe permitir en ningún caso retirar más capital del existente en la cuenta.

La aplicación deberá contar con una página principal provista de un pequeño menú que permitirá navegar al usuario a otras páginas encargadas de las diferentes tareas: *principal.aspx*, *alta.aspx*, *baja.aspx*, *modificaciones.aspx*.

La aplicación se hace para uso interno, por lo que todos los usuarios serán administradores y deben poder acceder a los mismos datos.

**9.-** Crea una aplicación que muestre una imagen en 9 fragmentos a modo de puzzle ( 3 x 3 casillas ). El usuario debe cambiar unas piezas por otras hasta conseguir restituir la imagen. Para intercambiar las imágenes fragmento el usuario seleccionará una ImageButton, y después otro. En ese momento se intercambiarán las imágenes entre ambos controles.

Cada vez que un usuario distinto acceda a la aplicación, los fragmentos de imagen aparecerán en diferente orden.