

# Tema 10. Capa de Interfaz II

Herramientas Avanzadas para el Desarrollo de Aplicaciones

# Indice

1. Controles de lista sencillos
2. Controles de Navegación
3. Controles de Validación
4. Objetos Session y Application
5. Eventos de Aplicación: Global.asax
6. Distribución de capas en el proyecto

1

Controles de lista  
sencillos

# Controles de Lista Sencillos

## DROPDOWNLIST



## LISTBOX



## CHECKBOXLIST



## RADIOBUTTON LIST



• Lista desplegable

• Lista desplegada

• Lista de botones de verificación

• Lista de botones de radio

# Controles de Lista Sencillos (listBox, dropDownList, radioButtonList, checkBoxList)

- Lista desplegable (dropDownList)



- ElementoLista1 (ListItem) **FILA 0**
- **ElementoLista2 *Seleccionado* (ListItem) FILA 1**
- ElementoLista3 (ListItem) **FILA 2**

# Añadir elementos de forma declarativa

```
<asp:DropDownList ID="DropDownList1" runat="server"
    onselectedindexchanged="DropDownList1_SelectedIndexChanged" Width="90px" AutoPostBack="False">
    <asp:ListItem Value="rojo1">rojo</asp:ListItem>
    <asp:ListItem>azul</asp:ListItem>
    <asp:ListItem>verde</asp:ListItem>
</asp:DropDownList>
```

# RepeatDirection

- RadioButtonList: lista de botones de radio

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server"
    onselectedindexchanged="RadioButtonList1_SelectedIndexChanged">
    <asp:ListItem>rojo</asp:ListItem>
    <asp:ListItem>azul</asp:ListItem>
    <asp:ListItem>verde</asp:ListItem>
</asp:RadioButtonList>
```

- CheckBoxList: lista de opciones múltiple

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server"
    onselectedindexchanged="CheckBoxList1_SelectedIndexChanged"
    RepeatDirection="Horizontal">
    <asp:ListItem>rojo</asp:ListItem>
    <asp:ListItem>azul</asp:ListItem>
    <asp:ListItem>verde</asp:ListItem>
</asp:CheckBoxList>
```

# Propiedad Items: colección de ListItems

## Objeto ListItem: propiedades

Miembros:

0	rojo
1	azul
2	verde

↑

↓

Propiedades de rojo:

✚ A ↓ | ☰

☐ Varios

Enabled	True
Selected	False
Text	rojo
Value	rojo1

- Text:
  - contenido visualizado
- Value:
  - valor oculto del código HTML
- Selected:
  - true o false (seleccionado o no)



# Propiedades de los Controles de Lista Sencillos

- Propiedad `SelectedIndex`
  - Indica la fila seleccionada como un índice que empieza en cero
- Propiedad `SelectedItem`
  - Permite que el código recupere un objeto **ListItem** que representa el elemento seleccionado

```
Label1.Text = DropDownList1.SelectedIndex.ToString();
```

```
Label1.Text = DropDownList1.SelectedItem.Text.ToString();
```

```
Label1.Text = DropDownList1.SelectedItem.Value.ToString();
```

```
Label1.Text=DropDownList1.SelectedValue;
```

# Controles de lista con selección múltiple

- **ListBox:**
  - Pueden seleccionarse varios elementos: propiedad **SelectionMode=Multiple**
- **CheckBoxList**
  - Siempre pueden seleccionarse varios elementos
- Para encontrar todos los elementos seleccionados necesitamos
  - recorrer la colección Items del control lista
  - comprobar la propiedad ListItem.Selected de cada elemento

```
Foreach ListItem i in DropDownList1.Items  
    { if (i.Selected==true)  
        .... }  
}
```

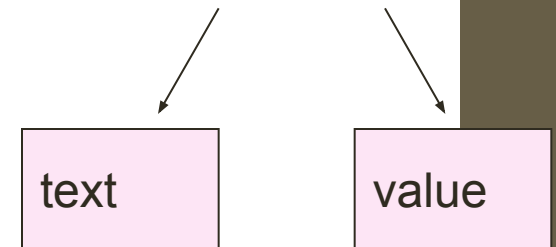
# Añadir elementos dinámicamente a una lista (código)

- Método *Add* del objeto *Items*

## EJEMPLO:

```
DropDownList1.Items.Add("rojo");
```

```
DropDownList1.Items.Add(new ListItem("rojo", "Red"));
```



2

Controles de  
navegación: Menu

# Control menu

- Se puede utilizar para crear un menú que colocaremos en la página maestra y otras ayudas de navegación.
- Permite añadir un menú principal con submenús y también nos permite definir menús dinámicos.
- Los elementos del Menu pueden añadirse directamente en el control o enlazarlos con una fuente de datos.
- En las propiedades podemos especificar la apariencia, orientación y contenido del menú.



# Static Display y Dynamic Display

- El control tiene dos modos de “**Display**”:
  - **Estático (static)**: El control Menu está expandido completamente todo el tiempo. Toda la estructura es visible y el usuario puede hacer click en cualquier parte.
  - **Dinámico (dynamic)**: En este caso solo son estáticas las porciones especificadas, mientras que los elementos hijos se muestran cuando el usuario mantiene el puntero del ratón sobre el nodo padre.

# Propiedades

- **StaticDisplayLevels.**
  - Esta propiedad permite especificar el número de niveles estáticos que queremos mostrar como raíz del menú (el mínimo es 1).
- **MaximumDynamicDisplayLevels**
  - Esta propiedad especifica cuantos niveles de nodos que aparecen de forma dinámica se mostraran después del nivel estático.

# Dynamic: cantidad de tiempo

- También podemos especificar la cantidad de tiempo que queremos que tarde la parte dinámica de un menú en desaparecer.
- Lo podemos especificar en milisegundos mediante la propiedad **DisappearAfter** del menu:
  - `Menu.DisappearAfter=1000;`
- El valor por defecto son 500 ms.



# Definición del contenido de menu

- Añadir objetos individuales MenuItem (de forma declarativa o programática) . También se le puede enlazar un archivo XML.
- Propiedades del control→ propiedad Items (colección de objetos MenuItem)

```
<asp:Menu ID="Menu1" runat="server" StaticDisplayLevels="3">
  <Items>
    <asp:MenuItem Text="File" Value="File">
      <asp:MenuItem Text="New" Value="New"></asp:MenuItem>
      <asp:MenuItem Text="Open" Value="Open"></asp:MenuItem>
    </asp:MenuItem>
    <asp:MenuItem Text="Edit" Value="Edit">
      <asp:MenuItem Text="Copy" Value="Copy"></asp:MenuItem>
      <asp:MenuItem Text="Paste" Value="Paste"></asp:MenuItem>
    </asp:MenuItem>
    <asp:MenuItem Text="View" Value="View">
      <asp:MenuItem Text="Normal" Value="Normal"></asp:MenuItem>
      <asp:MenuItem Text="Preview" Value="Preview"></asp:MenuItem>
    </asp:MenuItem>
  </Items>
</asp:Menu>
```

# Code-Behind C#

```
protected void Menu1_MenuItemClick(object sender,
    System.Web.UI.WebControls.MenuEventArgs e)
{
    switch(e.Item.Value)
    {
        case "Products":
            ...
            return;
        case "Services":
            ...
            return;
    }
}
```

[http://msdn.microsoft.com/en-us/library/16yk5dbv\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/16yk5dbv(v=vs.80).aspx)

[http://www.obout.com/em/doc\\_server.aspx](http://www.obout.com/em/doc_server.aspx)

3

## Controles de Validación

# Validación de datos

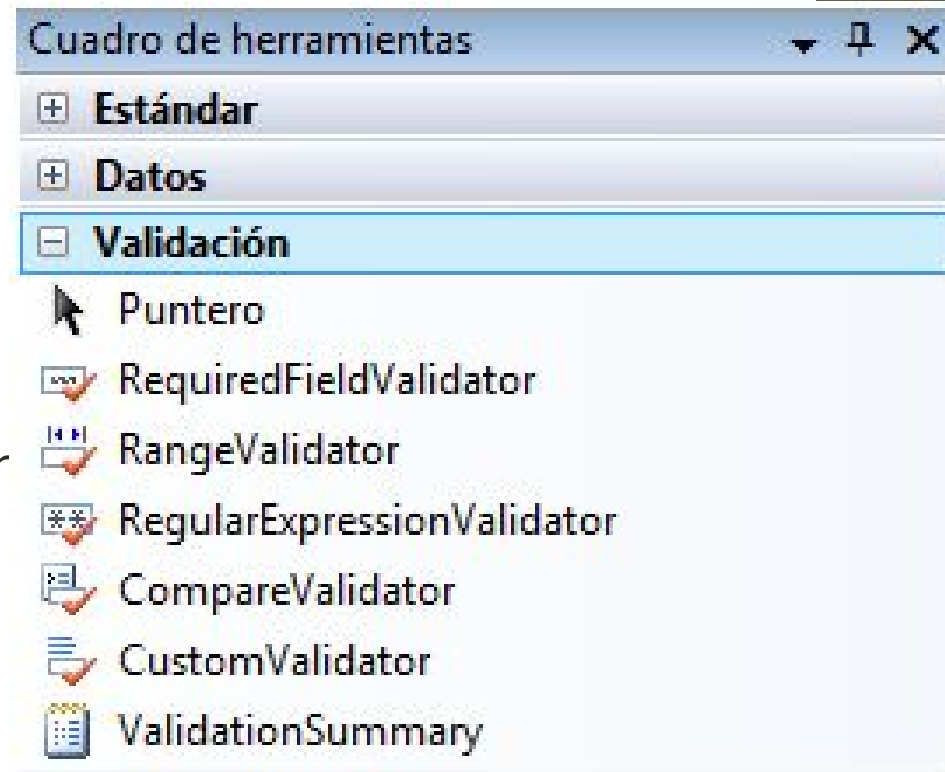
- Debemos asegurar que los usuarios introducen datos correctamente
  - Dirección de email
  - Número de teléfono
- ASP.Net proporciona un conjunto de controles de validación predefinidos
- Dos tipos de validación
  - Validación del lado del cliente
  - Validación del lado del servidor

# Validación de datos

- Lado del cliente:
  - Utilización de código JavaScript que valida los datos introducidos por el usuario, directamente en el navegador
- Lado del servidor:
  - Utilización de código (C#) para validar los datos de formularios una vez han sido enviados al servidor

# Controles de validación

- ASP.Net detecta si el navegador soporta validación del lado del cliente:
  - Generan el código JavaScript necesario para validar los datos
  - En otro caso, los datos del formulario se validan en el servidor



# Controles de validación

- Sobre diferentes controles web:
  - TextBox
  - ListBox
  - DropDownList
  - RadioButtonList
  - FileUpload

# Controles de Validación

- Propiedades
  - Para mostrar el mensaje de error **ErrorMessage**
  - Para indicar el control a validar **ControlToValidate**
- **Entrada requerida: RequiredFieldValidator**: La validación es OK cuando el control de entrada no contiene una cadena vacía.
  - InitialValue
- **Coincidencia de modelos: RegularExpressionValidator**: La validación es OK si el valor de un control de entrada se corresponde con una expresión regular especificada.
  - ValidationExpression



# Controles de validación

```
<form id="form1" runat="server">
  <div>
    Usuario: <asp:TextBox ID="TextBox1" runat="server">
      </asp:TextBox>
    <asp:RequiredFieldValidator ID="UserNameReq" runat="server"
      ControlToValidate="TextBox1"
      ErrorMessage="Introduce el usuario!!"> </asp:RequiredFieldValidator>
    Password: <asp:TextBox ID="TextBox2" runat="server">
      </asp:TextBox>
    <asp:RequiredFieldValidator ID="PasswordReq" runat="server"
      ControlToValidate="TextBox2"
      ErrorMessage="Introduce el password!!"> </asp:RequiredFieldValidator>
    <asp:Button ID="Button1" runat="server" Text="Enviar" />
  </div>
</form>
```

Usuario:  Introduce el usuario!!

Password:  Introduce el password!!

# Controles de Validación

- **Comparación con un valor: CompareValidator:** La validación es OK si el control contiene un valor que se corresponde con el valor de otro control especificado.
  - ControlToCompare
  - ControlToValidate
  - ValueToCompare
  - Type
  - Operator
- **Comprobación de intervalo: RangeValidator:** La validación es OK cuando el control de entrada contiene un valor dentro de un intervalo numérico, alfabético o temporal especificado.
  - MaximumValue
  - MinimumValue
  - Type

# Controles de Validación

- **CustomValidator**: La validación la realiza una función definida por el usuario.
  - ClientValidationFunction
  - OnServerValidate
- **Supongamos que queremos validar que el nombre de usuario sea único en la aplicación**

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>  
<asp:CustomValidator ID="CustomValidator1" ControlToValidate="TextBox1"  
    OnServerValidate="ComprobarUsuario" runat="server"  
    ErrorMessage="El usuario ya existe!!">  
</asp:CustomValidator>
```

# Controles de validación

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    { Button1.Text = "Correcto"; }
    else
    { Button1.Text = "Incorrecto"; }
}
```

```
protected void ComprobarUsuario(object sender,
    ServerValidateEventArgs e)
{
    string username = e.Value.ToLower();
    if (username == "sonia" || username == "irene")
    { e.IsValid = false; }
}
```

# Controles de Validación

- **ValidationSummary**: Este control muestra un resumen con todos los mensajes de error de cada control de validación.
  - ShowMessageBox:  
muestra un cuadro de diálogo resumen
  - ShowSummary

The screenshot shows a web form with three input fields: 'Usuario:', 'Password:', and 'Confirma:'. The 'Usuario:' and 'Password:' fields are highlighted in red, indicating validation errors. To the right of each field is a red error message: 'Introduce el usuario!!' and 'Introduce el password!!' respectively. Below the 'Confirma:' field, there are two red bullet points, each followed by the text 'Introduce el usuario!!'. At the bottom left of the form is a blue button labeled 'Enviar'. Overlaid on the form is a light blue dialog box with a yellow warning icon. The dialog box title is 'La página en http://localhost:49999 dice:'. It contains a list of error messages: '- Introduce el usuario!!' and '- Introduce el password!!'. At the bottom right of the dialog box is a blue button labeled 'Aceptar'.

Usuario:  Introduce el usuario!!


Password:  Introduce el password!!

Confirma: 

- Introduce el usuario!!
- Introduce el usuario!!

Enviar

La página en http://localhost:49999 dice:

 - Introduce el usuario!!  
- Introduce el password!!

Aceptar

# Propiedad Display

- Sirve para comprobar si el control de validación muestra mensajes de error. La propiedad Display puede establecerse en :
- **None** — El control de validación no aparece en la página.
- **Static** — Cada control de validación ocupa espacio aunque no sea visible el texto de un mensaje de error, lo que permite definir una presentación fija para la página.
- **Dynamic** — Los controles de validación no ocupan espacio a menos que muestren un mensaje de error, lo que les permite compartir la misma ubicación. No obstante, cuando se muestra el mensaje de error, la presentación de la página cambia, lo que a veces produce que los controles cambien las posiciones.
- La presentación dinámica requiere un explorador que admita Html dinámico (DHTML).

# El proceso de validación (servidor)

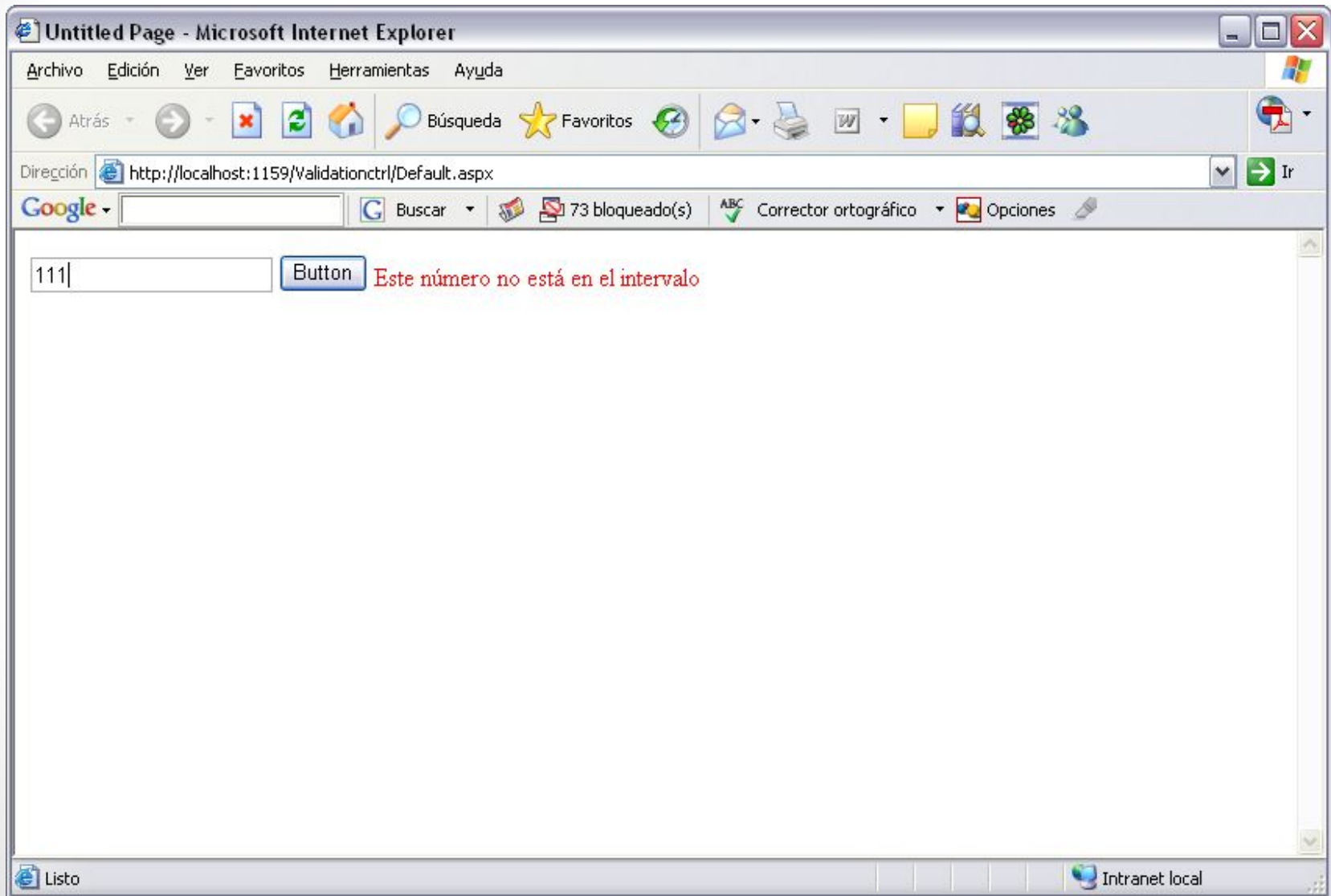
1. El usuario recibe una página y comienza a rellenar los valores de entrada. Al final el usuario pulsa un botón para enviar la página.
2. Cada control Button tiene una propiedad ***Causes Validation***.
  - Si esta propiedad es **False**, ASP.NET ignora los controles de validación.
  - Si está a **True** (Valor Predeterminado), ASP.NET valida automáticamente la página cuando el usuario pulsa el botón. Se realiza la validación de cada control de la página.
  - Cada control de validación expone su **propiedad IsValid**, La página también expone una propiedad IsValid que resume el estado **IsValid** de todos los controles de validación de la página.



```
<asp:RangeValidator ID="RangeValidator1"
    runat="server"
    ControlToValidate="TextBox1"
    ErrorMessage="Este número no está en
    el intervalo" MaximumValue="100"
    MinimumValue="10"    Type="Integer">
</asp:RangeValidator>
```



# Vista ejecución (range validator)



# Expresiones regulares

- **Dirección de correo electrónico**
  - Comprobar que existe una @, un punto y sólo permite caracteres que no sean espacios.
- **Contraseña**
  - Entre 4 y 10 caracteres y el primer carácter debe ser una letra.
- **Número de cuenta**
  - Secuencia de 4, 4, 2, y 10 dígitos, cada grupo separado por un guión.
- **Campo de longitud limitada**
  - Entre 4 y 10 caracteres incluyendo caracteres especiales (\*, &...)

# Sintaxis Expresiones Regulares

- `*` cero o más ocurrencias del carácter o subexpresión anterior.
- `+` una o más ocurrencias del carácter o subexpresión anterior
- `()` agrupa una subexpresión que se trata como un único elemento
- `|` Cualquiera de las dos partes (OR)
- `[]` se corresponde con un carácter en un intervalo de caracteres válidos [a-c]
- `{n}` exactamente n de los caracteres o subexpresiones anteriores
- `.` cualquier carácter excepto el salto de línea
- `?` el carácter anterior o la subexpresión anterior es opcional
- `^` comienzo de una cadena
- `$` fin de una cadena

...

- \s carácter de espacio en blanco (ej. tab o espacio)
- \S cualquier carácter no espacio
- \d cualquier carácter numérico
- \D cualquier carácter no dígito
- \w cualquier carácter alfanumérico (letra, número o carácter de subrayado)

# Solución...

- Correo electrónico
  - `\S+@\S+\.\S+`
- Contraseña
  - `[a-zA-Z]\w{3,9}`
- Número de cuenta
  - `\d{4}-\d{4}-\d{2}-\d{10}`
- Campo de longitud limitada
  - `\S{4,10}`

# Grupos de validación

- Los controles de validación se pueden asociar en grupos de validación a fin de que los controles que pertenezcan a un grupo común se validen juntos.
- Puede utilizar estos grupos para habilitar o deshabilitar de forma selectiva la validación para controles relacionados en una página.
- Hay que definir el nombre del grupo en los controles de validación y en el botón o otros controles de envío que causan validación.

# SetFocusOnError

- Se establece en controles de validación que causan que el primer control no válido reciba el foco.
- Propiedad de los controles de validación.
- Más información sobre estos controles

[http://msdn.microsoft.com/es-es/library/debza5t0\(v=vs.90\).aspx](http://msdn.microsoft.com/es-es/library/debza5t0(v=vs.90).aspx)

[http://www.elguille.info/colabora/NET2005/FernandoLuque\\_ASPValidar.htm](http://www.elguille.info/colabora/NET2005/FernandoLuque_ASPValidar.htm)

# 4

Mantenimiento de estado:  
Objetos Session y  
Application



# Objetos Session y Application

- Los objetos Session están asociados a un usuario particular y sirven como manera de transportar y mantener los datos del usuario en páginas web, como foros o sitios de comercio electrónico.
- Los objetos Application son compartidos por todos los usuarios y permiten almacenar información compartida por toda la aplicación web.
- En ASP.NET los objetos Session y Application están implementados como colecciones o conjuntos de pares nombre-valor.

# Qué es una sesión?

- Una **sesión** es el período de tiempo en el que un usuario particular interactúa con una aplicación web.
- Durante una sesión la identidad única de un usuario se mantiene internamente.
- Los datos se almacenan temporalmente en el servidor.
- Una sesión finaliza si hay un *timeout* o si tú finalizas la sesión del visitante en el código.

# Cuál es el uso de una sesión?

- Las sesiones ayudan a preservar los datos entre accesos sucesivos. Esto puede hacerse gracias a los objetos de sesión.
- **Los objetos de Sesión** nos permiten preservar las preferencias del usuario y otra información del usuario al navegar por la aplicación web.
- Ejemplo
- Website de comercio electrónico donde el visitante navega a través de muchas páginas y quiere seguir qué productos ha adquirido.

# Objeto Session

- **Session**: sirve para almacenar datos pertenecientes a un único usuario (en el ámbito de una sesión).

```
//Borra todos los valores de estado de la sesión  
Session.Clear();  
Session.Add("nombre","Homer");  
Response.Write(Session["nombre"]);
```

# En ASP.NET

- Las sesiones son tablas Hash en memoria con un timeout especificado.
- `Session["username"] = "Jose Martínez";`  
`Session["color"] = "Blue";`
- Asignamos los valores a las variables de sesión "username" y "color", respectivamente. Si necesito saber el "username" o "color" en páginas siguientes puedo usar `Session["username"]`, `Session["color"]`.
- Las sesiones en ASP.NET están identificadas usando enteros 32-bit long conocidos como Session IDs. El motor ASP genera estos session ID's de tal forma que se garantice que son únicos

# Objeto Session

Session Type	Qué hace	Ejemplo
<b>Session.Abandon</b>	<b>Abandona</b> (cancela) la sesión actual	
<b>Session.Remove</b>	<b>Borra un elemento</b> de la colección de estado de la sesión.	<b>Session["username"] =</b> <b>"Jose Martínez";</b> (Inicializa una variable de sesión) <b>Session.Remove["usernam</b> <b>"];</b> (Borra la variable de sesión "username")
<b>Session.RemoveAll</b>	<b>Borra todos los</b> elementos de estado de la sesión.	

Session Type	Qué hace	Ejemplo
Session.Timeout	Establece el the timeout ( <i>en minutos</i> ) para una sesión	Session.Timeout=30 (Si un usuario NO pide una página en la aplicación ASP.NET en 30 minutos la sesión expira.)
Session.SessionID	Recupera el ID de la sesión (propiedad de sólo lectura de una sesión) para la sesión.	
Session.IsNewSession	Es para comprobar que la sesión del usuario se creó con la petición actual p.ej. el usuario acaba de entrar al sitio web. La propiedad IsNewSession es cierta en la primera página de la aplicación.	

## Ejercicio

- Crear una aplicación web en la cual pidamos introducir un nombre de usuario y un botón enviar.
- Al pinchar en el botón que nos redirija a un segundo formulario en el cual pondremos “hola “ seguido del login introducido,:
  - Metiendo el login en una variable de sesión



# Botón con SESSION

Archivo Default.aspx.cs

```
protected void Button1_Click (object sender, EventArgs e)
{
    Session["login"] = TextBox1.Text;
    Response.Redirect("session2.aspx");
}
```

Archivo session2.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    Label3.Text = Session["Login"].ToString();
}
```

# Importante

Al crear la parte privada de la Web utilizaremos variables de sesión para controlar si el usuario ha entrado logueandose o ha entrado poniendo directamente la URL, en cuyo caso la variable de sesión estará vacía y no deberíamos permitir el acceso

# Variables de aplicación

- Y si queremos inicializar variables que estén disponibles en una sesión y sean las mismas para todos los usuarios??
- Esto supone que un cambio en el valor de una **variable de aplicación** se refleja en las sesiones actuales de todos los usuarios.

# Objeto Application

- Por ejemplo:
  - Se puede dar un valor a una variable de aplicación llamada SiteName  
`Application["SiteName"] = "Mi aplicación";`
  - Cualquier página de la aplicación puede leer esa cadena:  
`string appName = (string)Application["SiteName"];`
- Para eliminar cualquier variable del objeto Application:  
`Application.Remove("SiteName");`
- Para eliminar todas las variables:  
`Application.RemoveAll();`

# Variables de aplicación



## Ejercicio

- Crear una aplicación web que cuente el número de visitas que recibe
  - Utilizar variables de aplicación
  - Cuando llegue a 10 visitas el contador se debe reiniciar

# Variables de aplicación

- Primer paso:
  - En la página Default.aspx incluimos una etiqueta  
`<asp:Label ID="LabelCont" runat="server"></asp:Label>`
- Segundo paso:
  - En el archivo Default.aspx.cs (code behind) utilizar una variable Application para controlar el número de visitas

```
protected void Page_Load(object sender, EventArgs e) {  
    if (Application["PageCounter"] != null && (int)Application["PageCounter"] >= 10)  
    { Application.Remove("PageCounter"); }  
    if (Application["PageCounter"] == null)  
    { Application["PageCounter"] = 1; }  
    else  
    { Application["PageCounter"] =  
        (int)Application["PageCounter"] + 1; }  
    LabelCont.Text = Application["PageCounter"].ToString();  
}
```

# Variables de aplicación

- Problema:

- Dos personas cargan simultáneamente la página:

- El contador podría incrementarse sólo 1 unidad

`Application["PageCounter"] = (int)Application["PageCounter"] + 1`

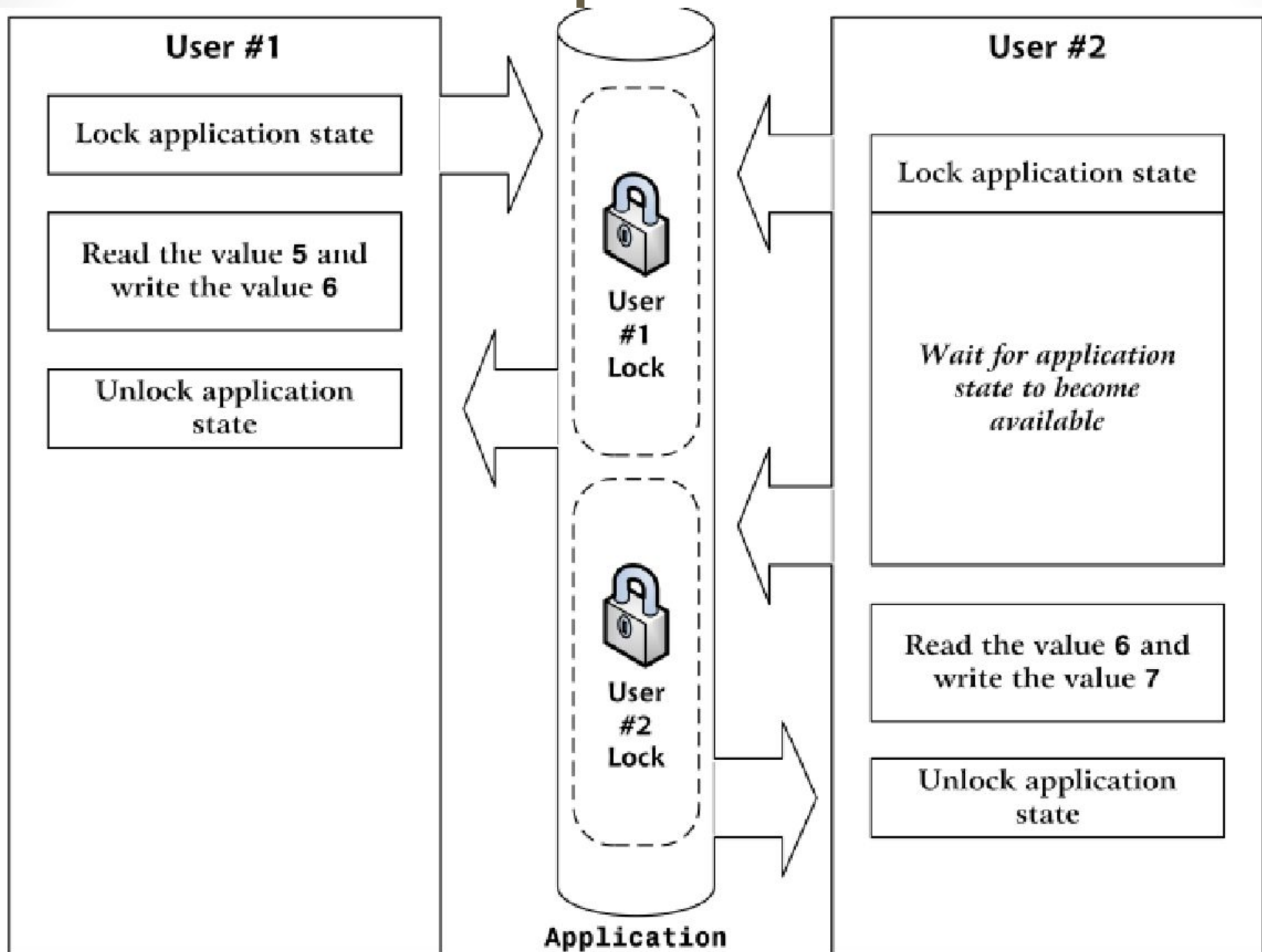
- La expresión de la derecha se evalúa primero
    - El usuario1 lee el valor de PageCounter almacenado en la aplicación
    - El usuario2 lee el valor de PageCounter almacenado en la aplicación
    - Ambos le suman 1 unidad pero el incremento final en lugar de ser 2 unidades es sólo 1

# Actualizar una variable de aplicación

- Solución:
  - En un instante concreto, varias sesiones pueden estar intentando cambiar el valor, aunque sólo una sesión estará autorizada a cambiarlo.
  - ASP.NET tiene exclusión mutua para este tipo de problemas:
    - **Application.Lock()** – Bloquea las variables de aplicación
    - **Application.Unlock()** – Desbloquea las variables de aplicación
  - Una vez que las variables están bloqueadas las sesiones que intentan cambiarlas tienen que esperar.



# Variables de aplicación



# Objeto Application

- **Application:** proporciona una manera sencilla de almacenar en el servidor datos comunes a todos los visitantes de nuestro sitio web.

```
Application.Lock();
```

```
Application.Add("edad",22);
```

```
int valor=(int)Application["edad"];
```

```
valor++;
```

```
Application["edad"]=valor;
```

```
Application.Unlock();
```

```
Response.Write(Application["edad"]);
```

# Problema

- Dónde inicializo una variable de aplicación para que no se reinicie cada vez (ej. Contador de visitas)????

5

Global.asax

# El archivo global.asax

- Permite escribir código de aplicación global.
- No contiene etiquetas HTML ni ASP.NET
- Se utiliza para definir variables globales y reaccionar a eventos globales.
- Contiene código de tratamiento de eventos que reacciona a los eventos de aplicación o sesión.

# El archivo global.asax

- Se añade a la aplicación web como un nuevo elemento Clase de aplicación global
  - Global.asax y Global.asax.cs

```
protected void Application_Start(object sender, EventArgs e)
{ Application["SiteName"] = "Mi aplicación"; }
```

```
protected void Session_Start(object sender, EventArgs e)
{
    Session.Timeout = 15;
    Response.Write("Servida el " + DateTime.Now.ToString());
}
```

- Importante:
  - Cualquier cambio en el archivo global.asax reiniciará la aplicación

# Ejemplo, uso de objetos de sesión

- Se quiere modificar el timeout por defecto (20min)
- Se puede hacer en cualquier lugar del código pero lo más recomendable es hacerlo en el archivo Global.asax

## **Archivo Global.asax**

```
protected void Session_Start(object sender, EventArgs e)
{
    Session.Timeout = 15;
}
```

# El archivo global.asax

- Sólo puede haber un archivo global.asax
- Debe residir en el directorio raíz de la aplicación

## Global.asax.cs

```
protected void Application_Error(object sender, EventArgs e)
{
    Response.Write("<b>");
    Response.Write("OOps! Ha ocurrido un error! </b>");
    Response.Write(Server.GetLastError().Message.ToString());
    Response.Write(Server.GetLastError().ToString());
    Server.ClearError();
}
```

**OOps! Ha ocurrido un error!** System.Web.HttpUnhandledException: Se produjo una excepción de tipo 'System.Web.HttpUnhandledException'. ---> System.DivideByZeroException:

## Default.aspx.cs

```
int j = 1;
int x = 0;
int k=j / x;
```



# CONTADOR DE VISITAS

## Añadir nuevo elemento...

- Clase de aplicación Global → Global.asax

```
void Application_Start(object sender, EventArgs e)
```

```
{
```

```
// Código que se ejecuta al iniciarse la aplicación
```

```
Application.Add("contador", 0);
```

```
}
```