5

Redireccionamiento en ASP .NET

y

Aspectos comunes de los controles

NAVEGACIÓN ENTRE PÁGINAS

En todos los ejemplos y ejercicios que hemos realizado hasta el momento, todas las páginas ASP.NET diseñadas se han enviado hacia sí mismas, lo que permite mantener el estado de la vista de los controles entre sucesivas llamadas a la página. Si empleamos la versión del lado del servidor de la etiqueta FORM no se puede enviar un formulario a una página diferente.

Si es imprescindible enviar un formulario a una página nueva, se puede emplear las etiquetas estándar de formulario HTML en lugar de los controles ASP.NET. El problema con este mecanismo es que se tiene acceso directo a los controles del formulario. Para poder acceder a la información de los mismos es necesario emplear la colección **Params** de la clase **HTTPRequest** (que representa la petición HTTP del usuario).

El objeto **HTTPRequest** de la página actual puede obtenerse mediante la propiedad **Request** del objeto **Page**.

Esta solución no suele ser la habitual, dado que en la mayoría de los casos se debería aprovechar el estado de la vista (por ejemplo, para validar datos) y mandar el formulario a sí mismo.

Response.Redirect

La mejor forma de transferir al usuario a otra página es utilizar el método **Redirect**() de la clase HTTPResponse (que permite enviar datos de respuesta HTTP a un cliente y contiene información sobre esa respuesta). Se accede al objeto HTTPResponse de una página a través de la propiedad **Response** de la misma.

En la **página origen** se realiza la redirección a la otra página: (en este caso mandando los parámetros en el mismo envío):

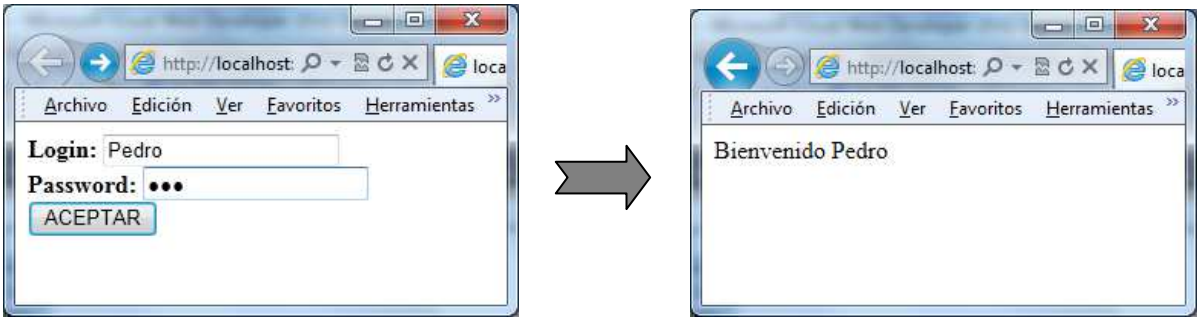
```
using System;
…
using System.Web.UI.WebControls;

public partial class RESPONSE : System.Web.UI.Page
{
    protected void autenticar(object sender, EventArgs e)
    {
       Response.Redirect("RESPONSE1.aspx?nombre=" + txtlogin.Text + "&contra=" +
       txtpassword.Text );
    }
}
```

En la **página destino** se pueden recoger de la siguiente forma

```
using System;
…
using System.Web.UI.WebControls;
```

```
public partial class RESPONSE1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        lblsaludo.Text = "Bienvenido " + Request.Params["nombre"];
       //o
        lblsaludo.Text = "Bienvenido " + Request.QueryString.Get("nombre");
     }
}
```



De esta forma las variables son pasadas en la barra de direcciones junto con la URL a la que queremos ir.

Nota: *El método Redirect tiene un parámetro boolean, que especifica si se termina la ejecución de la página actual:* ***true*** *, o no:* ***false****. Si se especifica* ***true*** *o si se utiliza el método* ***Redirect*** *especificando solamente la URL de destino, se producirá una excepción del tipo* ***ThreadAbortException****, que indica el término de la ejecución de la página que origina la navegación. Se debe considerar este comentario cuando se tiene sentencias* ***Redirect*** *dentro de bloques* ***Try****, ya que la excepción producida por el método* ***Redirect*** *puede no ser esperada en el flujo de la aplicación.*

También se pueden utilizar variables de Session para guardar los datos de los controles que queremos y recuperarlos en la página que nosotros queramos. De esta forma, además, no se enviarán las variables en la barra de dirección.

Pero, ¿Qué son las variables de sesión?

HTTP es un protocolo sin estado, es decir, el servidor Web trata cada solicitud HTTP de página como solicitud independiente; de forma predeterminada, el servidor no retiene información alguna sobre los valores de las variables que se utilizan durante las solicitudes anteriores. En consecuencia, la creación de aplicaciones Web que necesitan mantener la información de estado entre las solicitudes (aplicaciones que implementan carros de la compra, desplazamiento de datos, etc.) puede resultar complicada. El **estado de sesión** de ASP.NET identifica las solicitudes recibidas desde el mismo explorador durante un período limitado de tiempo como una sesión y proporciona la capacidad de conservar los valores de las variables durante la duración de esa sesión. Al utilizar el estado de la sesión, podemos asociar cualquier tipo de información con el usuario que queramos. Por ejemplo, podemos utilizar el estado de la sesión para mostrar el nombre del usuario en todas las páginas que solicite. O podemos utilizarlo para asociarle un carro de la compra según navega por las páginas. También podemos utilizarlo para permitir que el usuario rellene un formulario HTML de varias páginas. Cualquier elemento que añadamos a una variable sesión permanece en el servidor durante toda la visita del usuario al sitio Web y **finaliza cuando el usuario no ha solicitado ninguna página durante un período de 20 minutos** (tiempo determinado por defecto). Si pasado ese tiempo solicita una página, se le trata como un usuario nuevo.

La variable de sesión se crea con la siguiente sintaxis: **Session[“nombreVariable”]= valor**; Y cuando queramos recuperar el contenido de dicha variable aparecerá **Session[“nombreVariable”]** en la parte derecha de una asignación o en donde deseemos imprimir el valor.

Anteriormente hemos trabajado con variables **ViewState**, pero estas solo pueden ser utilizadas dentro de la misma página, es decir, el valor de dichas variables no será posible recuperarlo en una página distinta a la página en donde ha sido creada.

Página Origen

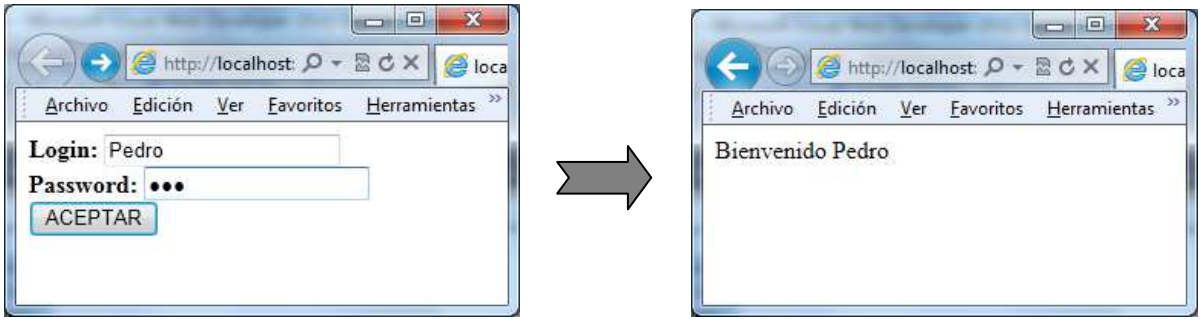
```
using System;
…
using System.Web.UI.WebControls;

public partial class RESPONSE : System.Web.UI.Page
{
    protected void autenticar(object sender, EventArgs e)
    {
        Session["nombre"] = txtlogin.Text;
        Session["contra"] = txtpassword.Text;
        Response.Redirect("RESPONSE1.aspx");
    }
}
```

Página Destino

```
using System;
…
using System.Web.UI.WebControls;

public partial class RESPONSE1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
       lblsaludo.Text = "Bienvenido " + Session["nombre"];
    }
}
```



Enlaces

Otro método para enlazar una página es utilizar el control **HyperLink,** que permite navegar hasta otra página html, asp o aspx.
La página destino se indica en la propiedad **NavigateURL** del control. El uso de este control implica que la operación de comunicación hacia otra página se realizará al hacer un clic en el cliente y no provocará ningún evento del lado del servidor.
Si se pretendiese pasar una variable o control a otra página se puede hacer

Server.Tranfer

Para transferir el control entre páginas aspx se puede utilizar también el método Transfer de la propiedad Server de la página (objeto Page). Este método es de la clase HttpServer.

Server.Transfer ("otrapagina.aspx")

y ejecuta dinámicamente el código de una página desde otra cualquiera. Si queremos comunicar variables o controles a la página destino debemos escribir:

Context.Items.Add ("variable",valor)

Se debe ejecutar en pimer lugar **Context.Item.Add** *("variable",valor)* y después ejecutar **Server.Transfer** *("otrapagina.aspx").*

En la página destino se recupera de la misma forma

x=Context.Items["mivariable"]

Hay que tener en cuenta que

Context.Items["variable"]

devuelve un objeto de tipo **object** y es necesario convertirlo al tipo que se desee. De esta manera se pueden pasar tanto variables de programa como controles Web. Hay que tener en cuenta que utilizando este método, cuando se llama a otra página termina la ejecución del Web Form que origina la llamada y navega hasta la página **aspx** de destino por lo que provoca también la excepción **ThreadAbortException**.

Cross-Page PostBack

El Cross-Page Posting existía en las antiguas versiones de ASP (1.0, 2.0 y 3.0), pero que desapareció en ASP .NET 1.0. Y consiste en la capacidad de poder enviar un formulario de una página ASPX a otra página ASPX, distinta de la actual.

En las versiones antiguas de ASP una operación muy común era la de hacer un envío de un formulario (post) a otra página ASP, y desde esta página ASP de destino se podía recuperar los valores de los distintos campos del formulario. La forma de recuperarlo era mediante la colección Form del objeto Request. Este esquema de trabajo resultará muy familiar a aquellos lectores que hayan desarrollado aplicaciones Web con ASP (1.0, 2.0 y 3.0).

En ASP .NET 1.x cuando se realizaba el envío de un formulario siempre se hacía sobre la propia página ASPX en la que se encuentra el formulario Web, no existe la posibilidad de indicar una página ASPX a la que enviar el formulario.

Pero ahora en ASP .NET 4.0 las cosas han vuelto a cambiar, en esta nueva versión si que podemos indicar una página ASPX a la que enviar el formulario que sea distinta de la actual. Esto lo podemos hacer de forma muy sencilla, haciendo uso de la propiedad **PostBackUrl** del control **Button**. El valor de esta propiedad será la ruta de la página ASPX a la que queremos enviar el formulario.

En la página **aspx** dentro de la etiqueta body aparecerá:

```
<asp:Button id="btnComprar" runat="server" Text="Comprar billetes"
PostBackUrl="~/Resultados.aspx"  />
```

En la página destino todavía se tiene acceso a la página original para acceder a la información de sus controles se debe hacer referencia a la propiedad PreviousPage del objeto Page (nuestra página y aplicar el método FindControl pasándole como parámetro el ID del control del que queremos obtener información.

En el código C# se pondrá:

PreviousPage.FindControl("idControl")

Con este métodos sólo se comunican los controles, no permite acceso a las variables definidas en el código C#.Se debe tener en cuenta que **PreviousPage.FindControl("idControl")** devuelve un **object** que tendrá que ser convertido a un tipo concreto para poder acceder a él.

```
TextBox txtnombre=(TextBox)PreviousPage.FindControl( "txtusuario" )

lblresultado.Text="Bienvenido "  + txtnombre.Text;
```

ESTILO DE LOS CONTROLES WEB

La especificación de hojas de estilo en cascada (Cascading Style Sheets, CSS) nos permite definir la apariencia y aspecto de los elementos de una página HTML, entendiendo por elementos las distintas etiquetas de HTML.

CSS (Cascade Style Sheets, hojas de estilo en cascada) es una recomendación basada en el estándar de Internet del W3C, que describe el conjunto de símbolos, reglas y asociaciones existentes entre los elementos del lenguaje que forman un documento HTML y su apariencia gráfica en pantalla: color, fondo, estilo de fuente, posición en el espacio, etc.

CSS establece los elementos necesarios para independizar el contenido de los documentos HTML de la forma con la que se presentan en pantalla en función del navegador empleado. CSS extiende las normas de formato del documento, sin afectar a la estructura del mismo. De esta manera, un error en un formato, un navegador que no soporta colores o cualquier otro error relacionado con el formato del documento no invalida el mismo. CSS además también aporta elementos dinámicos de formato, pues es posible capturar el estilo de los elementos cuando se produce una reacción sobre ellos.

Los controles Web ofrecen un completo soporte para las hojas de estilo, es decir, podemos aplicar los estilos de forma similar a como lo hacemos en HTML para personalizar el aspecto de nuestros controles.

Existen varias formas de aplicar estilos a los controles Web. A continuación vamos a comentar cada una de las posibilidades existentes.

1. Una de las formas de utilizar estilos con los controles Web es a través de una serie de propiedades que presentan los propios controles Web. Estas propiedades se encuentran fuertemente tipadas y representan los estilos más frecuentes que se pueden aplicar al control, color, tamaño de la letra, color de fondo, dimensiones, etc. La ventaja de aplicar los estilos de esta manera es que se comprueban las propiedades y sus valores en tiempo de compilación.

```
<asp:Label  runat="server"  id="etiqueta"  BackColor="Yellow"
BorderColor="Red" BorderStyle="Dotted"  ForeColor="Green" Font-
Size="12"Font-Bold="True" Font-Italic="True"
FontNames="Courier">Estilo</asp:Label>
```

2. Otra forma de aplicar estilos a un control Web es mediante la propiedad **CssClass**, a esta propiedad le asignaremos el nombre de clase que define el estilo que se desea aplicar sobre el control Web correspondiente. Las clases serán creadas en un script dentro de la cabecera de la página o con una hoja de estilos incrustada.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">

 <title>Pagina de Estilos</title>

    <style type="text/css">

           .clase{font:14 pt verdana;background-color:aqua ;color:purple}

    </style>

</head>
```

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="btnaceptar" runat="server" Text="ACEPTAR"  CssClass
="clase" onclick="btnaceptar_Click" />
    </div>
    </form>
</body>
</html>
```

3. Otra de las formas es crear un objeto de la clase **Style** y asignarle las propiedades con los valores que deseemos. Posteriormente se pasará como parámetro este objeto de tipo **Style** al método **ApplyStyle** del elemento que queramos.

```
protected void btnaceptar_Click(object sender, EventArgs e)
   {
            Style estilo = new Style();
            estilo.BorderColor = Color.Green;
            estilo.BorderStyle = BorderStyle.Dashed;
            estilo.ForeColor = Color.Blue;
            estilo.BackColor = Color.Red;
            estilo.Font.Name = "Arial";
            estilo.Font.Size=12;
            estilo.Font.Italic = true;
            btnaceptar.ApplyStyle(estilo);
   }
```

4. La última posibilidad de la que disponemos para aplicar estilos a nuestros controles Web es mediante la **propiedad Style**, que es una colección que nos permite asignar valores a las distintas propiedades del estilo del control. En este caso no se realiza ningún tipo de comprobación para indicar si la propiedad existe o no, únicamente se van añadiendo al control para luego mostrarse en el código HTML como parte de la propiedad **Style** del elemento HTML correspondiente.

```
protected void btnaceptar_Click(object sender, EventArgs e)
   {

       btnaceptar.Style["font-size"] = "20pt";
       btnaceptar.Style["font-family"] = "verdana";
       btnaceptar.Style["background-color"] = "pink";
       btnaceptar.Style["color"] = "20pt";
       btnaceptar.Style["border-style"] = "dotted";
       btnaceptar.Style["border-color"] = "blue";

   }
```

AÑADIR CONTROLES WEB EN TIEMPO DE EJECUCIÓN

En algunas ocasiones podemos tener la necesidad de añadir en tiempo de ejecución controles Web a nuestra página ASPX, esto puede ser debido a que son controles que se generan de forma dinámica tendiendo a elementos variables, como puede ser una consulta a una base de datos, una entrada indicada por el usuario, distintos factores de la ejecución de la aplicación, etc. En resumen, en este tipo de escenarios de ejecución no conocemos a priori los controles Web que va a contener nuestra página, y por lo tanto no los podremos crear en tiempo de diseño.

Será entonces necesario crear y añadir los controles Web cuando se realice la ejecución de la página ASPX.

Para añadir un control Web en tiempo de ejecución directamente a una página ASPX debemos hacerlo a través de la propiedad **Form** del objeto **Page**. La propiedad **Form**, que representa al formulario Web contenido en la página, posee la colección **Controls**, que contendrá todos los controles existentes dentro del formulario. Sobre esta colección **Controls** lanzaremos el método **Add**() pasándole como parámetro la instancia del control Web de servidor que deseamos añadir al formulario de la página.

Al igual que todas las colecciones **Controls** posee también los métodos **RemoveAt(pos)**, para borrar controles pasándole como parámetro la posición del elemento, el método **Remove(ID)** al que le pasaremos el ID del control que queremos borrar, **Clear()** para borrar todos los elementos del formulario y también poseen la propiedad **Count** que te da el número de controles que hay en el formulario de la página.

Creación de un control en ejecución:

```
public partial class CONTROLES : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
    {

       Button btnaceptar = new Button();
        btnaceptar.Text = "IR A GOOGLE";
        btnaceptar.Height = 50;
        btnaceptar.Width = 150;
        btnaceptar.Click += new EventHandler(navegar);
        Page.Form.Controls.Add(btnaceptar);

 }
    }
```

Creación de varios controles en Ejecución:

```
public partial class CONTROLES : System.Web.UI.Page
{

protected void Page_Load(object sender, EventArgs e)
    {
```

```
    String[] paginas = { "http://www.google.com", "http://www.yahoo.com",
"http://www.elcorreo.com", "http://www.elconfidencial.com",
"http://www.elpais.com", "http://www.elmundo.com" };

        for (int i = 0; i < paginas.Length; i++)
        {
            Button lapagina = new Button();
            lapagina.Text = paginas[i].Substring(paginas[i].IndexOf(".") + 1,
paginas[i].Length - paginas[i].IndexOf(".") - 5);
            lapagina.ID = "pagina" + i;
            lapagina.Width = 100;
            lapagina.Height = 50;
            lapagina.Click += new EventHandler(navegar);
            Page.Form.Controls.Add(lapagina);

        }
    }
```

Como habremos podido observar, aparece en la creación de los controles en ejecución la siguiente instrucción:

```
btnaceptar.Click += new EventHandler(navegar);
```

Esta instrucción es para crear un nuevo **manejador de eventos** (**EventHandler**), es decir, conseguir que el botón creado, en este caso, haga alguna función cuando hagamos un click sobre él. La instrucción se hace de la siguiente forma: Al método del control que queramos provocar, en nuestro caso un click del botón btnaceptar (**btnaceptar.Click**), le agregaremos un nuevo manejador de eventos **+= new EventHandler(nombreEvento)**. Este manejador de eventos necesita que se le pase como parámetro el nombre del evento que deberá ejecutar cuando hagamos el click en el botón. En nuestro caso se denomina **navegar**. Por tanto tenemos que crear un **evento navegar** como el que mostramos a continuación. Podemos generarlo copiando un evento del mismo tipo y cambiarle el nombre por **navegar**. Dentro de este evento ejecutaremos el código deseado.

```
protected void navegar(object sender, EventArgs e)
{
Button prueba = (Button)sender;

int posicion = Convert.ToInt32 (prueba.ID.Substring(prueba.ID.Length - 1));
Response.Redirect(paginas[posicion]);
}
```