Curso de Programación Web .NET

Unidad 4

“Web: ASP.NET 2”

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Índice

Manejar el estado de la web 3
Introducción 3
State 3
Qué es el PostBack 6
Introducción 6
IsPostBack 7
Postback en la práctica 7
Código del lado Cliente 9
Introducción 9
Scripts del lado del cliente 10
Client Side Source Code 11
AJAX 12
Introducción 12
Diagrama 12
Implementando AJAX en ASP.NET 13
El control ScriptManager 13
El control UpdatePanel 13

Manejar el estado de la web

Introducción

El protocolo HTTP en el cual se basa la web para funcionar es un protocolo que no maneja estado, esto quiere decir que cada vez que recargamos la página o también si la cerramos y la volvemos abrir todo trabajo que se había hecho, variables que quedaron en memoria, los valores de los controles u objetos creados son eliminados y vueltos a re-crear sin respetar los valores que contenían en el pasado.

Esto no sucede con las aplicaciones de tipo Escritorio ya que no se basan en el protocolo HTTP para funcionar las mismas se ejecutan directamente sobre la computadora y se valen de otras técnicas para persistir la información no presentes directamente en el protocolo HTTP.

State

Debido a que necesitamos persistencia para poder lograr desarrollar programas en la web se han creado técnicas para guardar la información entre pedidos y devolverlos cuando sea necesario. Los valores de todos los controles de la sesión del usuario se denomina State.

Hay distintos tipos de States:

- Lado Cliente
  - **View State:**
    Básicamente el ViewState es una bolsa que contiene variables las cuales pueden contener valores que queremos persistir luego de un postback. Muchas veces cuando enviamos información al servidor queremos que esta siga existiendo en la página. Por ejemplo cuantas veces tardamos tanto tiempo en rellenar un formulario muy largo y de repente al hacer click la página nos avisa de algún error en los campos y los borra a todos!. Este tipo de incidentes son los que soluciona el View State por ejemplo.

```
ViewState["variable"] = "valor"
```

  - **Control State**
    Por defecto todos los controles del form siguen el patrón de persistencia a menos que forcemos que no lo hagan, en cuyo caso al generar un PostBack los valores del control serán restaurados. Tanto View State como Control State descartan sus valores fuera del PostBack, es decir si la página se refresca o se cierra y se abre el navegador.

  - **Cookie**
    Las cookies son archivos que se guardan en el navegador web del cliente. Una de sus características es que se puede controlar la fecha que estos archivos caducan. Principalmente su ventaja es que son más persistentes que los dos métodos anteriores, al ser archivos se mantienen al refrescar una página o cerrar y abrir el navegador.

```
Request.Cookies["variable"] = "valor"
```

- Lado Servidor
  - **Session State**
    Esta técnica se persiste la información del lado servidor, el cliente no guarda los valores en su navegador. En este caso se utiliza el perfil creado en el servidor del cliente conectado a la página web , que ya contiene una estructura de información como la dirección IP, para guardar información que nos interesa persistir. El tiempo que esta información se almacena depende de la duración de la sesión configurada en el servidor, por ejemplo si el tiempo de vida de una sesión es de media hora ese es el tiempo que mi información va a estar almacenada en este espacio.
    Al ser del lado cliente no importa que acción realice con el explorador ya que la información siempre va a estar presente el tiempo que viva mi sesión.

```
Session["variable"] = "valor"
```

  - **Application State**
    El principio de esta técnica es muy similar a la anterior, es información almacenada en el servidor. La diferencia es que en este caso es a nivel aplicación y no depende de las sesiones. Es decir, si la aplicación se reinicia esta información desaparece.
    Otra característica no menor es que la información almacenada por aplicación no pertenece a ninguna sesión (o usuario) en particular, en vez de eso, todos los usuarios comparten estas variables. Por consiguiente es común que esta técnica se utilicen para configurar variables que dependen de todos los usuarios de la aplicación, por ejemplo: un contador de visitas.

```
Application["variable"] = "valor"
```

Qué es el PostBack

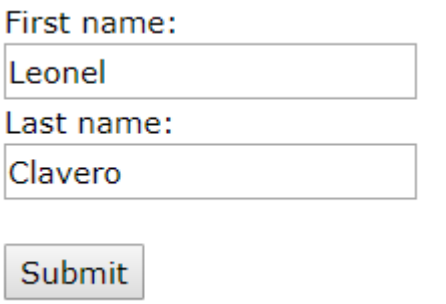
Introducción

Estuvimos hablando ya un poco del PostBack en las anteriores secciones, vamos a verlo ahora un poco más a fondo.

Por definición el postback es un mensaje HTTP POST de vuelta a la página que lo generó.

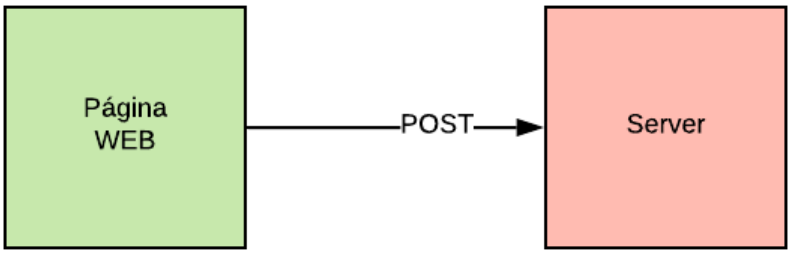
ASP.NET se basa en el modelo de eventos entre el cliente y el servidor, entonces el postback que es el que permite esto es crucial para este modelo, ya que recordemos que HTTP es state-less: cada acción que se realiza hace que toda la página se vuelva a cargar.

Por ejemplo si tenemos estos controles en un Web Form de ASP.NET:

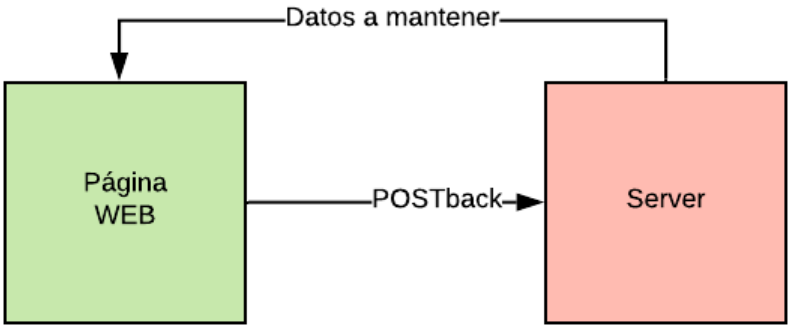


Cuando inicias un evento se ejecuta el Page_Load y siempre provoca el reinicio de la página.

Si el postback no existiese al hacer click los campos simplemente desaparecerían, ya que al hacer click genera una llamada al servidor y en cada llamada al mismo toda la información de la página es eliminada y cargada nuevamente:



El postback hace posible la comunicación entre el cliente con el servidor en la ejecución de eventos en la cual es crucial contar con un modelo de ida y vuelta desde el servidor para mantener los datos del lado del cliente:



Por esta razón en muchas aplicaciones web, se realiza una validación que posiblemente has visto, IsPostBack.

IsPostBack

IsPostBack no es más que una variable bool que se accede desde el code behind e indica si la página ha sido cargada por primera vez o si es una recarga (postback).

IsPostBack	**suceso**
False	Primera carga de la página.
True	Recarga de la página, por ejemplo: clickear un botón

Postback en la práctica

En el Page_Load del archivo C# (code behind) de una pagina aspx, debemos usar el validador junto con la variable **IsPostBack** del objeto **Page**.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack == True)
    {
        //Codigo de segunda a más cargas
    }
}
```

El código anterior se utiliza para indicar que contenido se va a utilizar en la segunda y en posteriores cargas ya que **IsPostBack** comienza en false.

Luego para cargar el contenido de la primer ejecución del a página, por ejemplo valores iniciales de las variables, debemos hacer lo siguiente:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack == False)
    {
        //Código solo en la primera carga
    }
}
```

En la primer ejecución (la del false) se carga toda la página y en la segunda (la del true) solamente los datos que queremos actualizar.

Así vamos a evitar descargar todo el contenido de la página en cada acción que hagamos que requiera solamente actualizar una parte de la misma. Esto va a hacer que la web tenga un mejor rendimiento, ahorrar memoria tanto en nuestro servidor como en la PC del cliente.

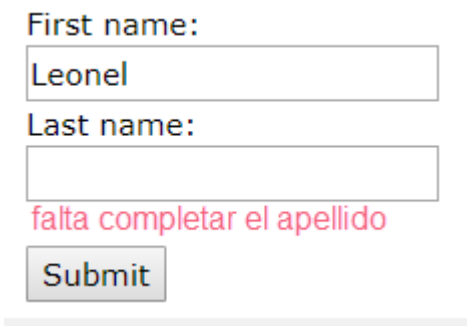
Sin IsPostBack sería muy complicado plantear una lógica de ida y vuelta entre el cliente y el servidor, la validación de datos sería muy engorrosa.

Código del lado Cliente

Introducción

Podemos dividir el código del lado del cliente de ASP.NET en dos categorías

Scripts del lado del cliente: Se ejecuta en el navegador Web lo que acelera la ejecución de la página. El ejemplo clásico es la validación de datos del lado del cliente que puede capturar datos no válidos y advertir al usuario en consecuencia sin hacer un viaje de ida y vuelta al servidor.



Código fuente del lado del cliente: Este código es generado por las páginas ASP.NET nosotros prácticamente no podemos modificarlo . El código fuente HTML de una página ASP.NET contiene campos ocultos y bloques de código JavaScript inyectados automáticamente. Estos son cruciales para el funcionamiento de ASP.NET y de la página ya que mantienen información como el estado de la vista o realizan otros trabajos para hacer que la página se muestre correctamente.

Scripts del lado del cliente

Los controles de ASP.NET permiten agregar código del lado del cliente escrito en Javascript o VBScript. Mismo como vimos arriba los controles ya generan por defecto su propio código Javscript para evitar ir y venir del servidor como en el caso de las validaciones.

Además de estos scripts, el control Button tiene una propiedad OnClientClick, que permite ejecutar scripts del lado del cliente, cuando se hace clic en el botón.

```
<asp:Button ID="btSuma" runat="server" Text="Suma" OnClientClick="fnMostrar()" />


<script  type="text/javascript">
        function fnMostrar() {
            alert('Hola Mundo');
}
</script>
```

document.getElementById("txtTextBox").value nos permite acceder a los valores de nuestros controles después de haberse convertido en elementos HTML desde Javascript.

Para poder seguir desarrollando en Javascript debemos tener conocimiento sobre el DOM y HTML. No es la intención de este curso explicar sobre estos conceptos de manera avanzada si no dar los lineamientos básicos para poder luego interiorizarse por su cuenta.

Los controles HTML tradicionales y de servidor tienen los siguientes eventos que pueden ejecutar un script cuando se generan:

onblur	Cuando el control pierde foco
onfocus	Cuando el control recibe foco
onclick	Cuando el control recibe un click
onchange	Cuando el valor del control es cambiado
onkeydown	Cuando el usuario presiona una tecla
onkeyup	Cuando el usuario suelta una tecla
onmouseover	Cuando el usuario mueve el mouse sobre el control
onserverclick	Cuando el control es clickeado genera el evento en el servidor

Client Side Source Code

Ya hemos visto que las páginas ASP.NET tienen dos archivos, uno el que carga el contenido y la estructura (aspx) y el otro que contiene el código (code behind).

El archivo de contenido contiene las etiquetas de control HTML o ASP.NET y los literales para formar la estructura de la página. El código detrás del archivo contiene la definición de la clase. En tiempo de ejecución, el archivo de contenido se analiza y se transforma en una clase de página.

Esta clase, junto con la definición de clase en el archivo de código, y el código generado por el sistema, juntos hacen que el código ejecutable (conjunto) procese todos los datos publicados, genere respuesta y lo envíe de vuelta al cliente.

AJAX

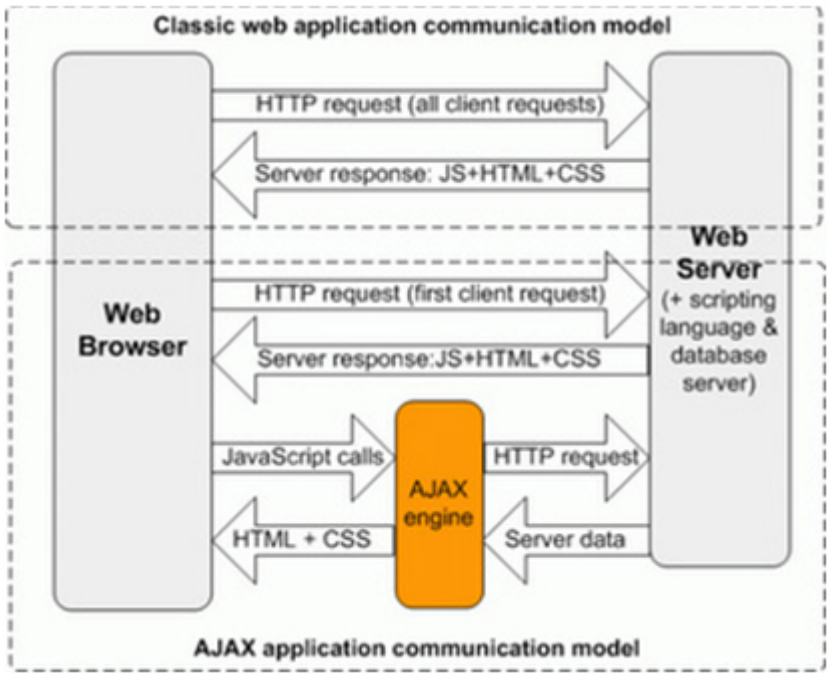
Introducción

AJAX es una técnica para crear páginas web rápidas y dinámicas ya que permite que las páginas web se actualicen de forma asíncrona al intercambiar pequeñas cantidades de datos con el servidor por detrás.

Las páginas web clásicas, que no usan AJAX, deben volver a cargar toda la página web si el contenido debe ser modificado. En cambio AJAX me permite actualizar solamente una parte de una página web sin volver a cargar toda la página.

Ejemplos de aplicaciones que utilizan AJAX: Google Maps, Gmail, Youtube y pestañas de Facebook.

Diagrama

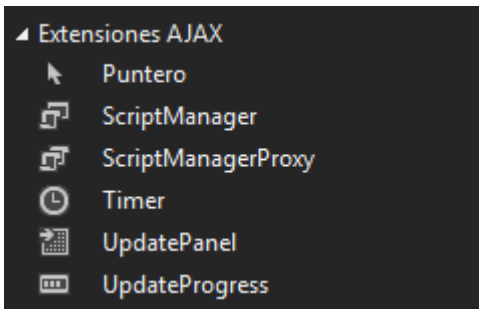


Como vemos en el diagrama en una página clásica no hay un motor de AJAX en el medio que realice tareas por detrás para recargar los datos dinámicamente. En una Página AJAX en el medio de la comunicación cliente servidor tenemos al motor AJAX que realiza el intercambio HTTP por nosotros y nos actualiza la porción de página que tiene que cambiar.

Implementando AJAX en ASP.NET

ASP.NET nos brinda controles para facilitar la implementación de AJAX.
Al igual que otros controles de servidor ASP.NET, estos controles de servidor AJAX también pueden tener métodos y controladores de eventos asociados, que se procesan en el lado del servidor.

La caja de herramientas de controles en el IDE de Visual Studio contiene un grupo ellos llamados **'Extensiones AJAX'**



El control ScriptManager

Este control es el más importante de todos y debe estar presente para que los demás controles de AJAX puedan funcionar. Es el que contiene el AJAX engine.

En las páginas web de ASP.NET lo van a ver con esta sintaxis

```
<asp:ScriptManager ID="ScriptManager1" runat="server"/>
```

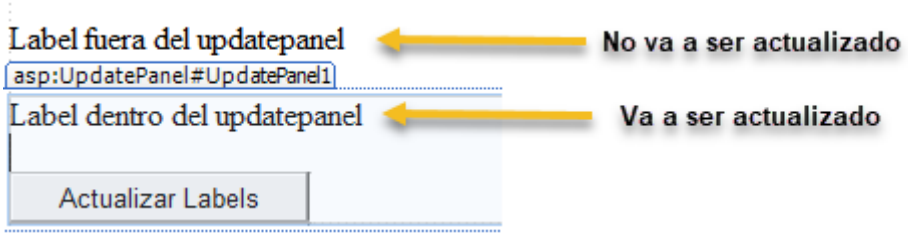
El control ScriptManager se encarga de la secuencia de comandos del lado del cliente para todos los controles del lado del servidor.

El control UpdatePanel

El control UpdatePanel es un panel como los demás paneles que hemos visto anteriormente. Como todos ellos actuá como un contenedor para los controles secundarios que se encuentran dentro de él.

La gran diferencia radica en su funcionamiento: Cuando un control en su interior desencadena un evento y por lo tanto requiere que se actualice, `UpdatePanel` interviene para iniciar su actualización asíncronica y solamente va a actualizar esa porción de la pagina web básicamente está aplicando la funcionalidad AJAX.

Por ejemplo si un control de botón está dentro del panel de actualización y se hace clic en él, solo se verán afectados los controles dentro del panel de actualización, los controles en las otras partes de la página no se verán afectados. Esto se denomina devolución parcial de publicaciones o devolución asíncrona.



Para actualizar el otro **label** al estar afuera de AJAX habría que actualizar toda la página lo cual puedo hacer luego.

En algunos momentos me puede interesar utilizar este tipo de técnicas para presentar al usuario información adicional en solo una parte de la página si que tenga que recargarla toda, por ejemplo poner en un control update panel un textbox que a medida que voy escribiendo se vaya autocompletando (al estilo google).