REVISTA DIGITAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES
TELEM@TICA
Revista Telem@tica. Vol. 12. No. 3, septiembre-diciembre, 2013, p.1-10 ISSN 1729-3804

Introducción a SignalR

MSc. Julio Cesar Jerez Camps camps@cujae.edu.cu

Ing. Raúl Castellanos Cabrera raulc@cujae.edu.cu

RESUMEN

En este artículo se describe qué es SignalR, y algunas de las funcionalidades que se pueden implementar con esta biblioteca. A grandes rasgos se abordan también las ventajas que trae su utilización con respecto a las prácticas comunes de programación Web dinámica; así como los diferentes casos en los cuales es viable su implementación.

Como vía para una mayor compresión del funcionamiento de la biblioteca se describe su funcionamiento interno y los objetivos que se persiguen con su aplicación a los sitios Web. Adicionalmente se tratan las vías de comunicación que pueden ser utilizadas por esta implementación así como las diferentes plataformas que las soportan.

PALABRAS CLAVES: Programación Web dinámica, ASP.NET, Actualizaciones de alta frecuencia

ABSTRACT

This article describes what SignalR is, and some of the solutions it was designed to create. It also give an account of the advantages we get by using it over the most commonly used dynamic web programming as well as the different cases in whichit is possible to use.

As a way to a better comprehension about how the SignalR library works its inner working process has been briefly described likewise is been done with its general objectives and the communication models that it can use as well as the different platforms that supports this models.

KEY WORDS: Dynamic Web Programming, ASP.NET, High Frequency Updates

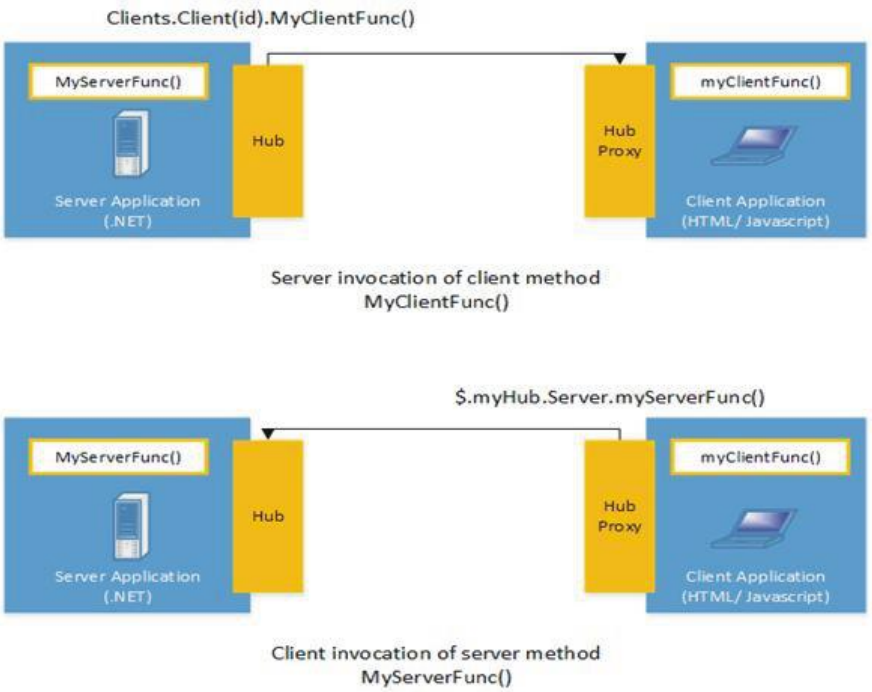
INTRODUCCIÓN

ASP.NET SignalR es una biblioteca para los desarrolladores de ASP.NET que simplifica el proceso de agregar funcionalidad de comunicación en tiempo real a las aplicaciones web. La funcionalidaden tiempo real de la web es la capacidad de tener código de servidor que envíe contenidos a los clientes conectados a medida que estén disponibles, en lugar de que el servidor espere que un cliente solicite nuevos datos.

Esta biblioteca se puede utilizar para añadir cualquier tipo de funcionalidad web "en tiempo real" para las aplicaciones ASP.NET. A pesar de que el chat se utiliza a menudo como un ejemplo, hay mucho más que se puede hacer. Cada vez que un usuario actualiza una página web para ver nuevos datos, o implementa en la página longpolling para recuperar nuevos datos, es un candidato para el uso de SignalR. Los ejemplos incluyen paneles de control, aplicaciones de monitoreo, aplicaciones colaborativas (como la edición simultánea de documentos), actualizaciones de progreso de trabajo y formularios en tiempo real.

Este tipo de implementación permite tipos completamente nuevos de aplicaciones web que requieren actualizaciones de alta frecuencia desde los servidores web, por ejemplo, juegos en tiempo real. Para un buen ejemplo de esto, ver el juego ShootR.

Además de las características antes mencionadas SignalRbrindauna API resulta sencilla para la creación de llamadas a procedimiento remoto (RPC) desde el servidor a funciones en JavaScript en los navegadores de los clientes (y otras plataformas de cliente) ; también incluye una API para la administración de conexión (por ejemplo, eventos de conexión y desconexión), y las agrupaciones de clientes.



Otra característica de esta biblioteca es que se encarga de la gestión de las conexiones de forma automática, y permite enviar mensajes a clientes específicos o a todos los clientes conectados simultáneamente, como una sala de chat. La conexión entre el cliente y el servidor es persistente, a diferencia de una conexión clásica HTTP, que se restablece con cada comunicación. Este tipo de implementación soporta la funcionalidad "server-push", con la cual el código de servidor puede llamar a código de cliente en el explorador mediante llamadas a procedimiento remoto (RPC), en lugar del modelo de solicitud-respuesta común en la web hoy en día.

SignalR es de código abierto, y se encuentra disponible a través de GitHub.

SignalR y WebSocket

SignalR utiliza la comunicación mediante el estándar WebSocket cuando es posible, en caso contrario vuelve a los protocolos de comunicación existentes anteriormente. Aunque ciertamente se podría implementar una aplicación usando WebSocket directamente, mediante SignalR es posible codificar la aplicación para aprovechar WebSocket sin tener que preocuparse por la creación de implementaciones específicas para los clientes más antiguos que no soporten WebSocket. El uso de esta biblioteca también le protege de tener que preocuparse de actualizaciones de WebSocket, puesto que se mantendrá siempre actualizada para soportar los cambios en el estándar de transporte subyacente, suministrando para las aplicaciones una interfaz consistente a través de las diferentes versiones de WebSocket.

Protocolos de Comunicación y Alternativas (Fallbacks)

SignalR es una abstracción sobre algunos de los medios de comunicación que son necesarios para hacer el trabajo en tiempo real entre el cliente y el servidor. Las aplicaciones que utiliza esta biblioteca inician las comunicaciones con conexiones HTTP, y se convierten a continuación a conexiones WebSocket si está disponible dicho estándar.

WebSocket es el protocolo ideal para SignalR, ya que hace un uso más eficiente de la memoria del servidor, tiene la latencia más baja, y tiene características importantes como la comunicación full dúplex entre el cliente y el servidor sin embargo, también tiene requisitos más estrictos para su uso. Si no se llegara a cumplir con las restricciones establecidas para este protocolo será necesario utilizar otros protocolos de comunicación, resultando en la disminución de la calidad de servicios de la aplicación implementada.

Algunos de los requerimientos del protocolo WebSocket son:

- Windows Server 2012 o Windows 8
- Framework 4.5 de .NET.
- Últimas versiones de Microsoft Internet Explorer, Google Chromeo Mozilla Firefox

Comunicación a través de HTML 5

Este tipo de comunicación depende de un soporte para HTML 5 que brinde el navegador del cliente. Si el navegador del cliente no admite el estándar HTML 5, se utilizará alguno de los medios de conexión anteriores.

•WebSocket (si el servidor y el navegador cliente soportan la utilización de WebSocket). WebSocket es el único medio de comunicación que establece una verdadera conexión persistente de dos vías entre el cliente y el servidor. Sin embargo, WebSocket también tiene los requisitos más exigentes, y está completamente soportado solo en las últimas versiones de Microsoft Internet Explorer, Google Chrome y Mozilla Firefox, y sólo tiene una aplicación parcial en otros navegadores como Opera y Safari.

•Eventos de Envío del Servidor (ServerSentEvents) también conocidos como EventSource (si el navegador soporta Servidor SentEvents) básicamente presente en todos los navegadores excepto Internet Explorer).

Comunicación a través de Comet

Los siguientes tipos de comunicación se basan en el modelo de aplicación Web Comet, en el que un navegador u otro cliente mantiene una petición HTTP de larga duración (Long-Held), que el servidor puede utilizar para enviar datos al cliente sin que el cliente la solicite.

•FrameEterno (ForeverFrame) sólo disponible para Internet Explorer. Este tipo de conexión crea un iframe oculto que hace una petición a un punto final en el servidor la cual no recibe respuesta y se mantiene activa todo el tiempo. A través de esta conexión el servidor se puede mantener enviando continuamente scripts hacia el cliente que se ejecutan inmediatamente, proporcionando una comunicación en tiempo real de un solo sentido desde el servidor al cliente. Para el transporte de solicitudes del cliente al servidor se utiliza una vía diferente de la utilizada entre servidor y cliente, este tipo de comunicación funciona como una solicitud HTML estándar, creando una nueva conexión para cada pieza de información que necesita ser enviada.

•Ajax longpolling. Long polling no crea una conexión persistente, en su lugar consulta el servidor con una petición de que se mantiene abierta hasta que el servidor responde, momento en el cual la conexión se cierra, y se solicita una nueva conexión inmediatamente. Esto puede introducir un tiempo de latencia mientras se restablece la conexión.

Para obtener más información sobre qué tipos de comunicación son compatibles y con qué configuraciones, consulte Plataformas compatibles.

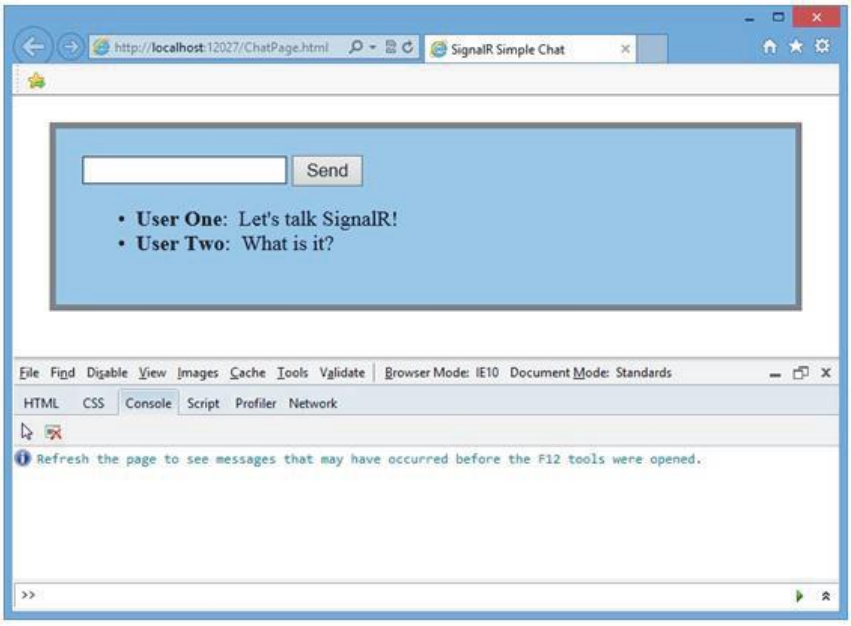
Monitoreo de Comunicaciones

Se puede determinar el tipo de comunicación que su aplicación está utilizando habilitando el registro del objeto “hub”, y abriendo la ventana de consola en su navegador.

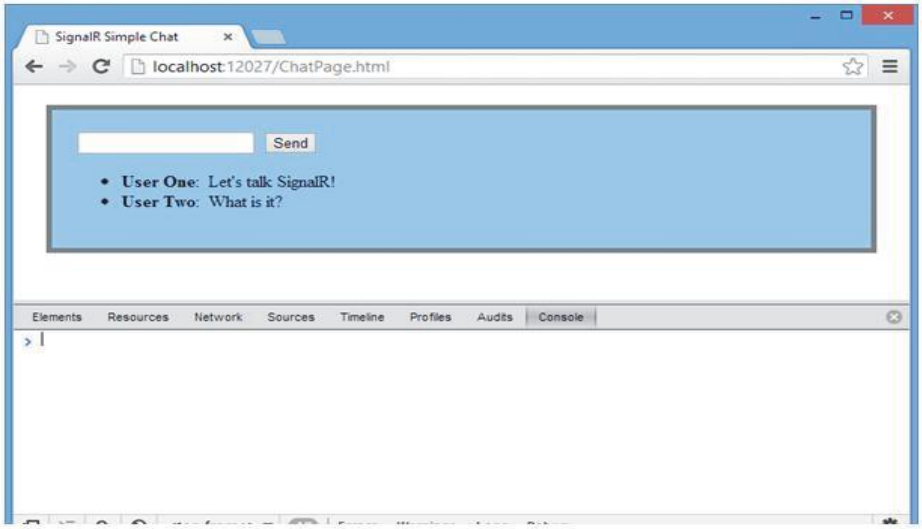
Para habilitar el registro de eventos de su hub en un navegador, añada el siguiente comando para la aplicación cliente:

```
$.connection.myHub.logging = true;
```

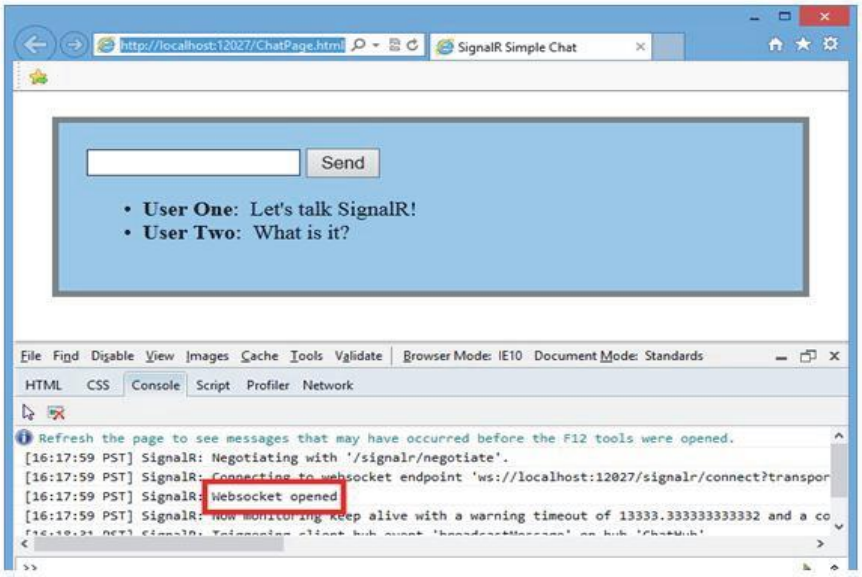
•En Internet Explorer, abra las herramientas para desarrolladores pulsando F12 y haga clic en la pestaña Consola.



•En Chrome, abra la consola presionando Ctrl + Shift + J.



•Con la consola abierta y el registro habilitado, usted será capaz de ver cual modo de comunicación está siendo utilizado por SignalR.



Especificando un tipo de comunicación

La negociación de una conexión toma una cierta cantidad de tiempo y recursos de cliente / servidor. Si se conocen las capacidades del cliente, entonces un tipo de comunicación puede ser especificada cuando se inicia la conexión del cliente. El siguiente fragmento de código muestra cómo se inicia una conexión utilizando Ajax Long Polling.

connection.start({ transport: 'longPolling' });

Se puede especificar un orden de respaldo si se quiere que el cliente pruebe conexiones específicos. El siguiente fragmento de código demuestra cómo intentar conectarse con WebSocket y en caso de fallo cambiar el tipo de conexión a Long Polling.

connection.start({ transport: ['webSockets','longPolling'] });

Las constantes de cadena para especificar medios de comunicación específicos se definen como se muestra a continuación:

- websockets
- foreverFrame
- serverSentEvents
- LongPolling

Conexiones y Concentradores (Hubs)

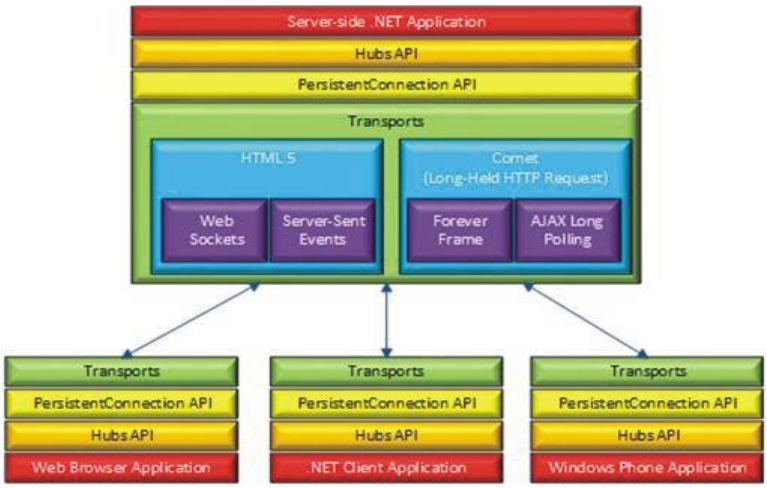
El API de SignalR contiene dos modelos de comunicación entre clientes y servidores: Conexiones persistentes y concentradores (Hubs).

Una conexión representa un punto final simple para enviar mensajes de difusión, de grupos de destinatarios o a un solo destinatario. La API de conexión persistente (representado en .NET por la clase PersistentConnection) da acceso directo al protocolo de comunicación de bajo nivel de SignalR. Usar el modelo de Conexiones le será familiar a los desarrolladores que han utilizado API tales como Windows CommunicationFoundation.

Un Hub es una vía de comunicación de más alto nivel desarrollada a partir de la API de conexión que permite al cliente y al servidor llamar a métodos entre sí directamente. SignalR maneja el envío entre máquinas, de manera transparente al desarrollador, permitiendo a los clientes llamar a métodos en el servidor de la misma facilidad que los métodos locales, y viceversa. El uso del modelo de comunicación Hub resultará familiar para los desarrolladores que han utilizado las API de invocación remota como .NETRemoting y adicionalmente permitirá pasar parámetros fuertemente definidos (stronglytypedparameters) a los métodos, permitiendo la unificación de los modelos de datos cliente y servidor.

Diagrama de la arquitectura

El siguiente diagrama muestra la relación entre hubs, conexiones persistentes, y las tecnologías subyacentes utilizadas para la comunicación.



¿Cómo funcionan los Hubs?

Cuando el código del lado del servidor llama a un método en el cliente, se envía un paquete a través de la conexión activa que contiene el nombre y los parámetros del método a ser llamado (cuando un objeto se envía como un parámetro de método, se serializa utilizando JSON). El cliente entonces compara el nombre del método que recibió con los métodos definidos en el código del lado del cliente. Si hay una coincidencia, el método de cliente se ejecuta utilizando los datos de los parámetros deserializados.

La llamada al método se puede controlar utilizando herramientas como Fiddler. La imagen siguiente muestra una llamada a un método enviada desde un servidor SignalR a un navegador web cliente

SignalR en el panel de registros de Fiddler. La llamada al método está siendo enviado desde un Hub llamado "MoveShapeHub", y el método que se invoca se llama "updateShape".

```
10:38:03:1298 Session846.WebSocket'WebSocket #846' - Pushing 104 bytes from server WebSocket
81 66 7B 22 43 22 3A 22 42 2C 31 35 7C 43 2C 30     f{"C":"B,15|C,0
7C 44 2C 30 7C 45 2C 30 22 2C 22 4D 22 3A 5B 7B  |D,0|E,0","M":[{
22 48 22 3A 22 4D 6F 76 65 53 68 61 70 65 48 75  "H":"MoveShapeHu
62 22 2C 22 4D 22 3A 22 75 70 64 61 74 65 53 68  b","M":"updateSh
61 70 65 22 2C 22 41 22 3A 5B 7B 22 6C 65 66 74  ape","A":[{"left
22 3A 35 30 31 2E 30 2C 22 74 6F 70 22 3A 33 30  ":501.0,"top":30
32 2E 30 7D 5D 7D 5D 7D                          2.0}]}]}
```

En este ejemplo, el nombre del Hub se identifica con el parámetro H; el nombre del método se identifica con el parámetro M, y los datos que se envían al método se identifican con el parámetro A.

Como elegir un modelo de comunicación

La mayoría de las aplicaciones deben utilizar la API de Hubs. La API de conexiones podría ser utilizada en las siguientes circunstancias:

- El formato del mensaje que se envía debe ser especificado.
- El desarrollador prefiere trabajar con un modelo de envío de mensajes en lugar de un modelo de invocación remota.
- Una aplicación existente que utiliza un modelo de mensajería está siendo modificado para utilizar SignalR.

CONCLUSIONES

SignalR es un framework potente y fácil de utilizar que nos permite mantener conexiones abiertas con el servidor de ASP.NET y que es posible aplicar a una gran variedad de escenarios.

Con el uso de este tipo de librerías podemos empezar a realizar aplicaciones realmente colaborativas y en tiempo real que alcanzan una mayor actualización de la información en todo momento sin tener que esperar por encuestas de los clientes. Adicionalmente vale mencionar que el uso de estas librerías puede ponerse en uso en los sistemas que ya se encuentran desarrollados sin tener que realizar modificaciones de impacto en los mismos.

REFERENCIAS BIBLIOGRÁFICAS

1. What is ASP.NET SignalR.GitHub Pages; 2013; Available from: http://signalr.net/.

2. ASP.NET SignalR. GitHub Pages; 2013; Available from: https://github.com/SignalR/SignalR.

3. Fletcher P. Introduction to SignalR 2013. Available from: http://www.asp.net/signalr/overview/getting-started/introduction-to-signalr.

4. Tutorial: Getting Started with SignalR2013. Available from: http://www.asp.net/signalr/overview/getting-started/tutorial-getting-started-with-signalr.

5. High-Frequency Realtime with SignalR 2013. Available from: http://www.asp.net/signalr/overview/getting-started/tutorial-high-frequency-realtime-with-signalr.