عنوان: **نگاهی به SignalR Hubs**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۲ ۹:۲۹
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, jQuery, SignalR

Hubs کلاس‌هایی هستند جهت پیاده سازی push services در SignalR و همانطور که در قسمت قبل عنوان شد، در سطحی بالاتر از اتصال ماندگار (persistent connection) قرار می‌گیرند. کلاس‌های Hubs بر مبنای یک سری قرار داد پیش فرض کار می‌کنند (ایده Convention-over-configuration) تا استفاده نهایی از آن‌ها را ساده‌تر کنند.
Hubs به نوعی یک فریم ورک سطح بالای RPC نیز محسوب می‌شوند (Remote Procedure Calls) و آن‌را برای انتقال انواع و اقسام داده‌ها بین سرور و کلاینت و یا فراخوانی متدی در سمت کلاینت یا سرور، بسیار مناسب می‌سازد. برای مثال اگر قرار باشد با persistent connection به صورت مستقیم کار کنیم، نیاز است تا بسیاری از مسایل serialization و deserialization اطلاعات را خودمان پیاده سازی و اعمال نمائیم.

قرار دادهای پیش فرض Hubs

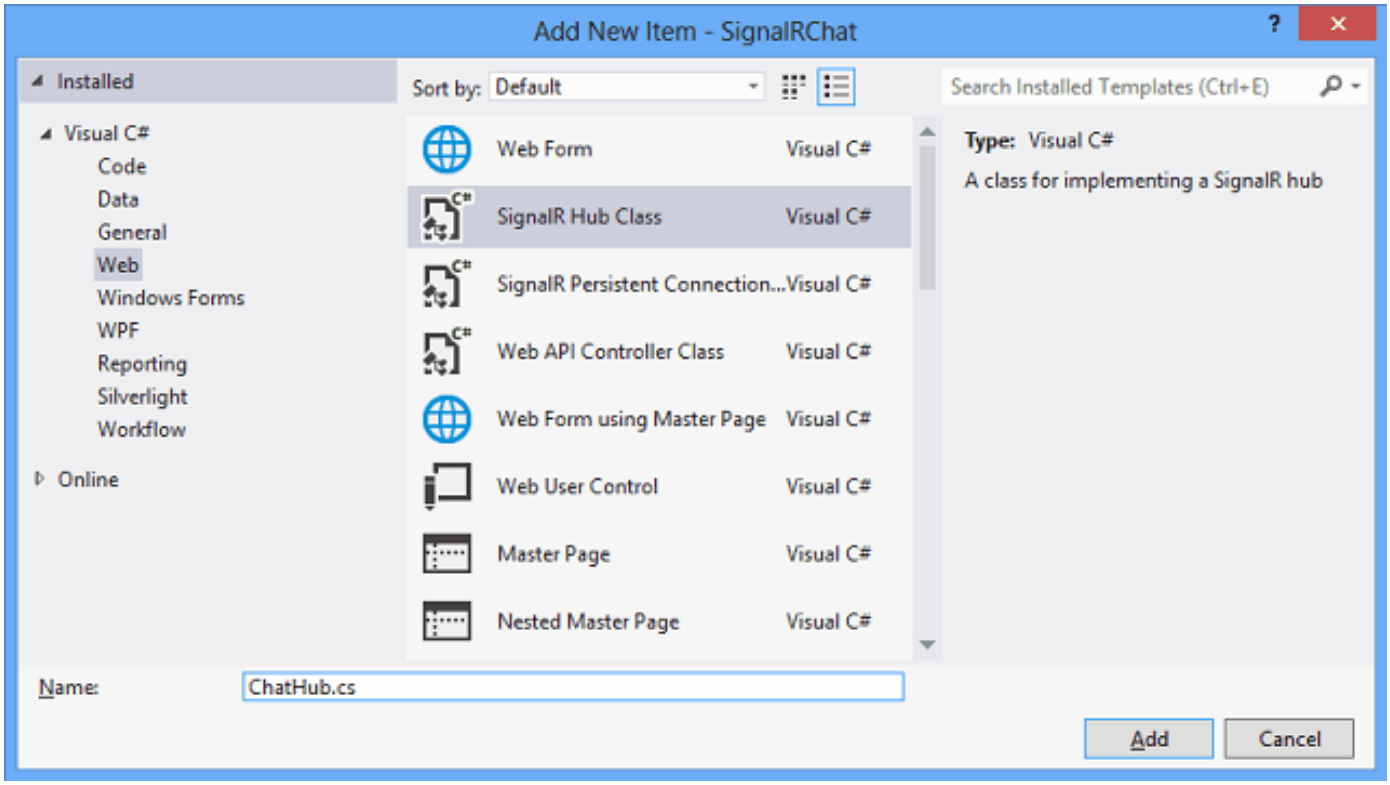
- متدهای public کلاس‌های Hubs از طریق دنیای خارج قابل فراخوانی هستند.
- ارسال اطلاعات به کلاینت‌ها از طریق فراخوانی متدهای سمت کلاینت انجام خواهد شد. (نحوه تعریف این متدها در سمت سرور بر اساس قابلیت‌های dynamic اضافه شده به دات نت 4 است که در ادامه در مورد آن بیشتر بحث خواهد شد)

مراحل اولیه نوشتن یک Hub

الف) یک کلاس Hub را تهیه کنید. این کلاس، از کلاس پایه Hub تعریف شده در فضای نام Microsoft.AspNet.SignalR باید مشتق شود. همچنین این کلاس می‌تواند توسط ویژگی خاصی به نام HubName نیز مزین گردد تا در حین برپایی اولیه سرویس، از طریق زیرساخت‌های SignalR به نامی دیگر (یک alias یا نام مستعار خاص) قابل شناسایی باشد. متدهای یک هاب می‌توانند نوع‌های ساده یا پیچیده‌ای را بازگشت دهند و همه چیز در اینجا نهایتا به فرمت JSON رد و بدل خواهد شد (فرمت پیش فرض که در پشت صحنه از کتابخانه معروف JSON.NET استفاده می‌کند؛ این کتابخانه سورس باز به دلیل کیفیت بالای آن، از زمان ارائه MVC4 به عنوان جزئی از مجموعه کارهای مایکروسافت قرار گرفته است).
ب) مسیریابی و Routing را تعریف و اصلاح نمائید.
و ... از نتیجه استفاده کنید.

تهیه اولین برنامه با SignalR

ابتدا یک پروژه خالی ASP.NET را آغاز کنید (مهم نیست MVC باشد یا WebForms). برای سادگی بیشتر، در اینجا یک ASP.NET Empty Web application درنظر گرفته شده است. در ادامه قصد داریم یک برنامه Chat را تهیه کنیم؛ از این جهت که توسط یک برنامه Chat بسیاری از مفاهیم مرتبط با SignalR را می‌توان در عمل توضیح داد.
اگر از VS 2012 استفاده می‌کنید، گزینه SignalR Hub class جزئی از آیتم‌های جدید قابل افزودن به پروژه است (منوی پروژه، گزینه new item آن) و پس از انتخاب این قالب خاص، تمامی ارجاعات لازم نیز به صورت خودکار به پروژه جاری اضافه خواهند شد.



و اگر از VS 2010 استفاده می‌کنید، نیاز است از طریق NuGet ارجاعات لازم را به پروژه خود اضافه نمائید:

```
PM> Install-Package Microsoft.AspNet.SignalR
```

اکنون یک کلاس خالی جدید را به نام ChatHub، به آن اضافه کنید. سپس کدهای آن را به نحو ذیل تغییر دهید:

```
using Microsoft.AspNet.SignalR;
using Microsoft.AspNet.SignalR.Hubs;

namespace SignalR02
{
    [HubName("chat")]
    public class ChatHub : Hub
    {
        public void SendMessage(string message)
        {
            Clients.All.hello(message);
        }
    }
}
```

همانطور که ملاحظه می‌کنید این کلاس از کلاس پایه Hub مشتق شده و توسط ویژگی HubName، نام مستعار chat را یافته است.
کلاس پایه Hub یک سری متد و خاصیت را در اختیار کلاس‌های مشتق شده از آن قرار می‌دهد. ساده‌ترین راه برای آشنایی با این متدها و خواص مهیا، کلیک راست بر روی نام کلاس پایه Hub و انتخاب گزینه Go to definition است.
برای نمونه در کلاس ChatHub فوق، از خاصیت Clients برای دسترسی به تمامی آن‌ها و سپس فراخوانی متد dynamic ایی به نام hello که هنوز وجود خارجی ندارد، استفاده شده است.
اهمیتی ندارد که این کلاس در اسمبلی اصلی برنامه وب قرار گیرد یا مثلا در یک class library به نام Services. همینقدر که از کلاس Hub مشتق شود به صورت خودکار در ابتدای برنامه اسکن گردیده و یافت خواهد شد.

مرحله بعد، افزودن فایل global.asax به برنامه است. زیرا برای کار با SignalR نیاز است تنظیمات Routing و مسیریابی خاص آن‌را اضافه نمائیم. پس از افزودن فایل global.asax، به فایل Global.asax.cs مراجعه کرده و در متد Application_Start آن

تغییرات ذیل را اعمال نمائید:

```
using System;
using System.Web;
using System.Web.Routing;

namespace SignalR02
{
    public class Global : HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            // Register the default hubs route: ~/signalr
            RouteTable.Routes.MapHubs();
        }
    }
}
```

یک نکته مهم

اگر از ASP.NET MVC استفاده می‌کنید، این تنظیم مسیریابی باید پیش از تعاریف پیش فرض موجود قرار گیرد. در غیراینصورت مسیریابی‌های SignalR کار نخواهند کرد.

اکنون برای آزمایش برنامه، برنامه را اجرا کرده و مسیر ذیل را فراخوانی کنید:

```
http://localhost/signalr/hubs
```

در این حال اگر برنامه را برای مثال با مرورگر chrome باز کنید، در این آدرس، فایل جاوا اسکریپتی SignalR، قابل مشاهده خواهد بود. مرورگر IE پیغام می‌دهد که فایل را نمی‌تواند باز کند. اگر به انتهای خروجی آدرس مراجعه کنید، چنین سطری قابل مشاهده است:

```
proxies.chat = this.createHubProxy('chat');
```

و کلمه chat دقیقا از مقدار معرفی شده توسط ویژگی HubName دریافت گردیده است.

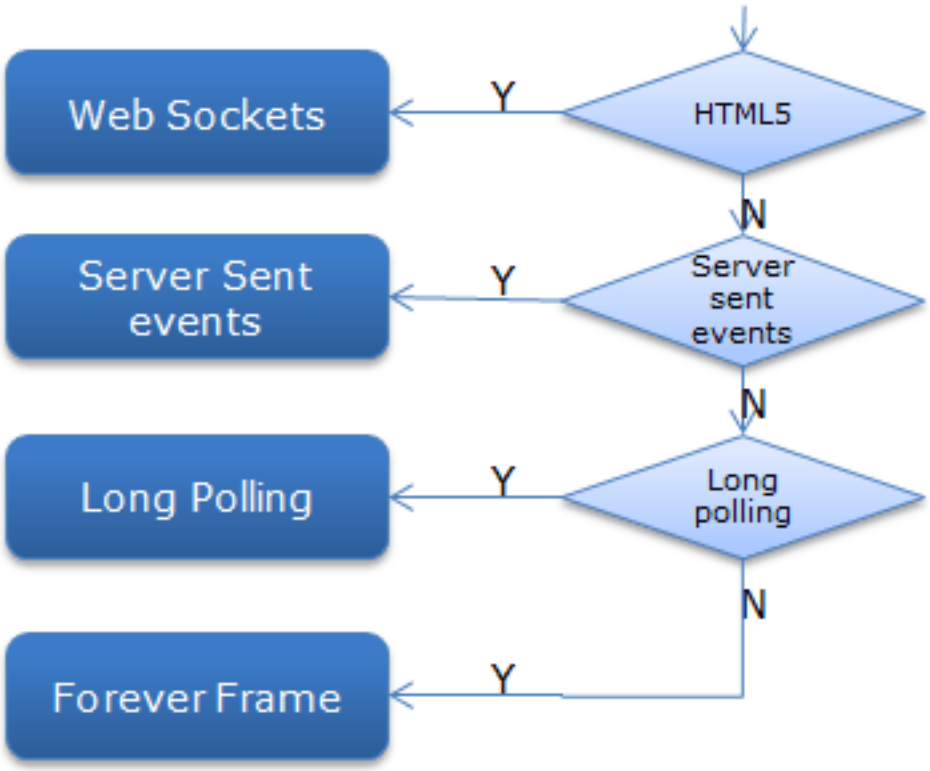
تا اینجا ما موفق شدیم اولین Hub خود را تشکیل دهیم.

بررسی پروتکل Hub

اکنون که اولین Hub خود را ایجاد کرده‌ایم، بد نیست اندکی با زیر ساخت آن نیز آشنا شویم.
مطابق مسیریابی تعریف شده در Application_Start، مسیر ابتدایی دسترسی به SignalR با افزودن اسلش SignalR به انتهای مسیر ریشه سایت بدست می‌آید و اگر به این آدرس یک اسلش hubs را نیز اضافه کنیم، فایل js metadata مرتبط را نیز می‌توان دریافت و مشاهده کرد.

زمانیکه یک کلاینت قصد اتصال به یک Hub را دارد، دو مرحله رخ خواهد داد:
الف) negotiate: در این حالت امکانات قابل پشتیبانی از طرف سرور مورد پرسش قرار می‌گیرند و سپس بهترین حالت انتقال، انتخاب می‌گردد. این انتخاب‌ها به ترتیب از چپ به راست خواهند بود:

```
Web socket -> SSE -> Forever frame -> long polling
```

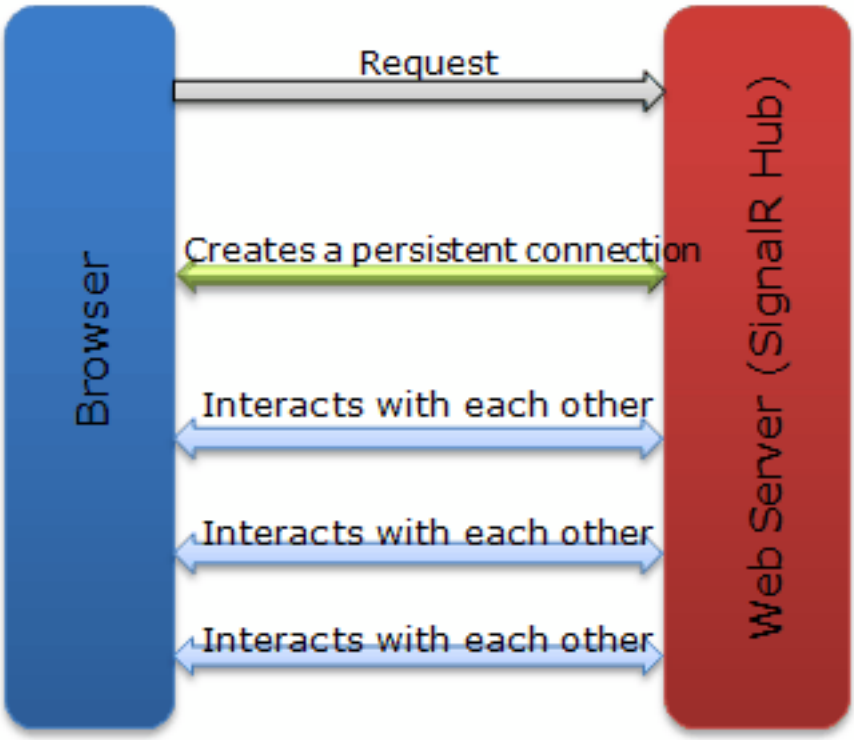


به این معنا که اگر برای مثال امکانات Web sockets مهیا بود، در همینجا کار انتخاب نحوه انتقال اطلاعات خاتمه یافته و Web sockets انتخاب می‌شود.
تمام این مراحل نیز خودکار است و نیازی نیست تا برای تنظیمات آن کار خاصی صورت گیرد. البته در سمت کلاینت، امکان انتخاب یکی از موارد یاد شده به صورت صریح نیز وجود دارد.
ب) connect: اتصالی ماندگار برقرار می‌گردد.

در پروتکل Hub تمام اطلاعات JSON encoded هستند و یک سری مخفف‌هایی را در این بین نیز ممکن است مشاهده نمائید که معنای آن‌ها به شرح زیر است:

```
C: cursor
M: Messages
H: Hub name
M: Method name
A: Method args
T: Time out
D: Disconnect
```

این مراحل را در قسمت بعد، پس از ایجاد یک کلاینت، بهتر می‌توان توضیح داد.



روش‌های مختلف ارسال اطلاعات به کلاینت‌ها

به چندین روش می‌توان اطلاعاتی را به کلاینت‌ها ارسال کرد:
1) استفاده از خاصیت Clients موجود در کلاس Hub
2) استفاده از خواص و متدهای dynamic
در این حالت اطلاعات متد dynamic و پارامترهای آن به صورت JSON encoded به کلاینت ارسال می‌شوند (به همین جهت اهمیتی ندارند که در سرور وجود خارجی دارند یا خیر و به صورت dynamic تعریف شده‌اند).

```
using Microsoft.AspNet.SignalR;
using Microsoft.AspNet.SignalR.Hubs;

namespace SignalR02
{
    [HubName("chat")]
    public class ChatHub : Hub
    {
        public void SendMessage(string message)
        {
            var msg = string.Format("{0}:{1}", Context.ConnectionId, message);
            Clients.All.hello(msg);
        }
    }
}
```

برای نمونه در اینجا متد hello به صورت dynamic تعریف شده است (جزئی از متدهای خاصیت All نیست و اصلا در سمت سرور وجود خارجی ندارد) و خواص Context و Clients، هر دو در کلاس پایه Hub قرار دارند.
حالت Clients.All به معنای ارسال پیامی به تمام کلاینت‌های متصل به هاب ما هستند.

3) روش‌های دیگر، استفاده از خاصیت dynamic دیگری به نام Caller است که می‌توان بر روی آن متد دلخواهی را تعریف و فراخوانی کرد.

```
//این دو عبارت هر دو یکی هستند
Clients.Caller.hello(msg);
```

```
Clients.Client(Context.ConnectionId).hello(msg);
```

انجام اینکار با روش ارائه شده در سطر دومی که ملاحظه می‌کنید، در عمل یکی است؛ از این جهت که Context.ConnectionId همان ConnectionId فراخوان می‌باشد.
در اینجا پیامی صرفا به فراخوان جاری سرویس ارسال می‌گردد.

4) استفاده از خاصیت dynamic ایی به نام Clients.Others

```
Clients.Others.hello(msg);
```

در این حالت، پیام، به تمام کلاینت‌های متصل، منهای کلاینت فراخوان ارسال می‌گردد.

5) استفاده از متد Clients.AllExcept
این متد می‌تواند آرایه‌ای از ConnectionIdهایی را بپذیرد که قرار نیست پیام ارسالی ما را دریافت کنند.

6) ارسال اطلاعات به گروه‌ها
تعداد مشخصی از ConnectionIdها یک گروه را تشکیل می‌دهند؛ مثلا اعضای یک chat room.

```
public void JoinRoom(string room)
        {
            this.Groups.Add(Context.ConnectionId, room);
        }

        public void SendMessageToRoom(string room, string msg)
        {
            this.Clients.Group(room).hello(msg);
        }
```

در اینجا نحوه الحاق یک کلاینت به یک room یا گروه را مشاهده می‌کنید. همچنین با مشخص بودن نام گروه، می‌توان صرفا اطلاعاتی را به اعضای آن گروه خاص ارسال کرد.
خاصیت Group در کلاس پایه Hub تعریف شده است.
نکته مهمی را که در اینجا باید درنظر داشت این است که اطلاعات گروه‌ها به صورت دائمی در سرور ذخیره نمی‌شوند. برای مثال اگر سرور ری استارت شود، این اطلاعات از دست خواهند رفت.

آشنایی با مراحل طول عمر یک Hub

اگر به تعاریف کلاس پایه Hub دقت کنیم:

```
public abstract class Hub : IHub, IDisposable
    {
        protected Hub();
        public HubConnectionContext Clients { get; set; }
        public HubCallerContext Context { get; set; }
        public IGroupManager Groups { get; set; }

        public void Dispose();
        protected virtual void Dispose(bool disposing);
        public virtual Task OnConnected();
        public virtual Task OnDisconnected();
        public virtual Task OnReconnected();
    }
```

در اینجا، تعدادی از متدها virtual تعریف شده‌اند که تمامی آن‌ها را در کلاس مشتق شده نهایی می‌توان override و مورد استفاده قرار داد. به این ترتیب می‌توان به اجزا و مراحل مختلف طول عمر یک Hub مانند برقراری اتصال یا قطع شدن آن، دسترسی یافت. تمام این متدها نیز با Task معرفی شده‌اند؛ که معنای غیرهمزمان بودن پردازش آن‌ها را بیان می‌کند.

تعدادی از این متدها را می‌توان جهت مقاصد logging برنامه مورد استفاده قرار داد و یا در متد OnDisconnected اگر اطلاعاتی را در بانک اطلاعاتی ذخیره کرده‌ایم، بر این اساس می‌توان وضعیت نهایی را تغییر داد.

ارسال اطلاعات از یک Hub به Hub دیگر در برنامه

فرض کنید یک Hub دوم را به نام MinitorHub به برنامه اضافه کرده‌اید. اکنون قصد داریم از داخل ChatHub فوق، اطلاعاتی را به آن ارسال کنیم. روش کار به نحو زیر است:

```
public override System.Threading.Tasks.Task OnDisconnected()
        {
            sendMonitorData("OnDisconnected", Context.ConnectionId);
            return base.OnDisconnected();
        }

        private void sendMonitorData(string type, string connection)
        {
            var ctx = GlobalHost.ConnectionManager.GetHubContext<MonitorHub>();
            ctx.Clients.All.newEvenet(type, connection);
        }
```

در اینجا با override کردن OnDisconnected به رویداد خاتمه اتصال یک کلاینت دسترسی یافته‌ایم. سپس قصد داریم این اطلاعات را توسط متد sendMonitorData به Hub دومی به نام MonitorHub ارسال کنیم که نحوه پیاده سازی آن‌را در کدهای فوق ملاحظه می‌کنید. GlobalHost.ConnectionManager یک dependency resolver توکار تعریف شده در SignalR است.
مورد استفاده دیگر این روش، ارسال اطلاعات به کلاینت‌ها از طریق کدهای یک برنامه تحت وب است (که در همان پروژه هاب واقع شده است). برای مثال در یک اکشن متد یا یک روال رویدادگردان کلیک نیز می‌توان از GlobalHost.ConnectionManager استفاده کرد.

نظرات خوانندگان

نویسنده: فرزاد حق
تاریخ: ۱۳۹۲/۰۱/۱۳ ۱۷:۳۳

واقعا عالیه، جناب نصیری

نویسنده: امیر نوروزیان
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۳:۴۳

با سلام
متاسفانه من SignalR Hub class دسترسی در سیستم ندارم ویژال 2012 و تنظیمات
NuGet انجام دادم .چیزی خاصی دیگر نیاز است

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۳:۴۸

- VS 2012 یک سری آپدیت داره.
+ من تمام مثال‌های این سری رو با VS 2010 پیاده سازی کردم. فقط از NuGet به روشی که عنوان شده استفاده کنید. نیازی به هیچ قالب اضافه‌تری ندارید.

نویسنده: امیر خلیلی
تاریخ: ۱۳۹۲/۰۸/۰۴ ۱۴:۳۹

من هم همین مشکل را دارم و با نصب NuGet باز هم کلاس SignalR Hub
برای انتخاب در لیست نبود

آیا برای این منظور همه اون آپدیت‌ها که فرمودین لازمه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۴ ۱۴:۴۵

نصب کنید خوبه؛ ولی ضروری نیست. با نصب از طریق NuGet فقط اسمبلی‌های لازم و فایل‌های لازم اضافه می‌شوند؛ نه قالب‌های VS.NET مرتبط. این قالب‌ها هم ضروری نیستند. مثلا یک کلاس Hub چیزی نیست جز یک کلاس ساده (نمونه‌اش در متن مطلب جاری هست) که از کلاس پایه Hub مشتق می‌شود (این کلاس رو دستی خودتون ایجاد کنید؛ الزامی نیست که ابزار اینکار را برای شما انجام دهد)

نویسنده: محمد احمدی آذر
تاریخ: ۱۳۹۲/۰۹/۱۲ ۱۱:۳۶

با سلام و تشکر از آموزش‌های روان شما
ممکن است دوستان در استفاده از این آموزش دچار اشکالی شوند که ناشی از بروز رسانی SignalR از ورژن 1 به 2 است.
در ورژن‌های جدیدتر SignalR از Owin برای ارتباط خود استفاده می‌کند بنابراین در صورت استفاده از دستور

```
RouteTable.Routes.MapHubs();
```

در Application Start به خطا می‌خوریم جهت حل این مشکل ابتدا باید یک فایل OwinStartUp.cs به برنامه اضافه شود و به صورت کد زیر SignalR را مپ کنیم:

```
public void Configuration(IAppBuilder app)

        {
            app.MapSignalR();
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۲ ۱۳:۲

ممنون. بله. این مورد در مطلب « نحوه‌ی ارتقاء برنامه‌های SignalR 1.x به SignalR 2.x » بیشتر بحث شده.

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۳۹۳/۰۸/۰۲ ۱۴:۳۷

یک hub را داخل یک پروژه دیگر از نوع class library قرار دادم و با ارجاع به یک کنسول که selfhost شده، میخواستم ازش استفاده کنم، کار نمیکند. به همین سادگی!

کد hub

```
[HubName("messageHub")]
    public class MessageHub : Hub
    {
        public void NotifyAllClients()
        {
            Clients.All.Notify();
        }
    }
```

کلاس startup

```
public partial class Startup
    {
        public void Configuration(IAppBuilder appBuilder)
        {
            var hubConfiguration = new HubConfiguration()
            {
                EnableDetailedErrors = true
            };

            appBuilder.MapSignalR(hubConfiguration);

            appBuilder.UseCors(CorsOptions.AllowAll);


        }
    }
```

نقطه آغازین برنامه:

```
static void Main(string[] args)
        {
            const string baseAddress = "http://localhost:9000/"; // "http://*:9000/";

            using (var webapp = WebApp.Start<Startup>(baseAddress))
            {
                Console.WriteLine("Start app...");

                var hubConnection = new HubConnection(baseAddress);
                IHubProxy messageHubProxy = hubConnection.CreateHubProxy("messageHub");

                messageHubProxy.On("notify", () =>
                {
                    Console.WriteLine();
                    Console.WriteLine("Notified!");
                });
```

```
                hubConnection.Start().Wait();

                Console.WriteLine("Start signalr...");

                bool dontExit = true;
                while (dontExit)
                {
                    var key = Console.ReadKey();
                    if (key.Key == ConsoleKey.Escape) dontExit = false;

                    messageHubProxy.Invoke("NotifyAllClients");
                }

            }
        }
```

اگر کلاس hub را به داخل پروژه‌ی slefhost منتقل کنم، کار میکند. اما در یک class library دیگر خیر.
نگارش دات نت و ارجاعات همه یکسان است. dotnet 4.5
آیا نکته ای جاانداخته شده در این نمونه کد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۲ ۱۵:۲

در مطلب نگاهی به گزینه‌های مختلف مهیای جهت میزبانی SignalR بیشتر بحث شده‌است:
« باید توجه داشت که در این حالت (self hosting) برخلاف روش ASP.NET Hosting، سایر اسمبلی‌های برنامه جهت یافتن Hubهای تعریف شده، اسکن نمی‌شوند »
یک راه حل برای رفع آن، افزودن سطر زیر به ابتدای برنامه است (قبل از شروع هر کد دیگری):

```
AppDomain.CurrentDomain.Load(typeof(Lib1.MessageHub).Assembly.FullName);
```