LEARNING signalr

#signalr

Table of Contents

About

Chapter 1: Getting started with signalr

Remarks

This section provides an overview of what signalr is, and why a developer might want to use it.

It should also mention any large subjects within signalr, and link out to the related topics. Since the Documentation for signalr is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
2.2.1	2016-07-13
2.2.0	2014-12-11
2.1.2	2014-09-09
2.1.1	2014-07-22
2.1.0	2014-05-21
2.0.3	2014-03-25
2.0.2	2014-01-22
2.0.1	2013-10-22
2.0.0	2013-08-29
1.2.2	2014-08-18
1.2.1	2014-02-10
1.2.0	2013-12-11
1.1.4	2013-10-22
1.1.3	2013-07-31
1.1.2	2013-05-24
1.1.1	2013-05-16
1.1.0	2013-05-11

Examples

Getting up and running

IIS / .NET version Requirements: See here

SignalR 2+

1. Install the NuGet package `Microsoft.AspNet.SignalR` (this is the whole SignalR solution) and it will ask you to install any dependencies for other packages. Accept the terms and install them as well.

2. Now that we've got the `.dlls` and client scripts needed to generate our own SignalR Hub, let's create one. Click on your Web project, add a folder named `Hubs` or `SignalR`, and in it add a class **`NameOfYourChoosing`**_`Hub`_. I will name mine `MessagingHub.cs`.

3. We need to derive from the base class `Hub` that is in our SignalR dll that we downloaded via NuGet. The code would look like :

```
[HubName("messagingHub")]
public class MessagingHub : Hub
{
    //empty hub class
}
```

   And in our Startup.cs class we can let the `IAppBuilder` know that we are going to use SignalR.

```
public partial class Startup
{
    public void Configuration(IAppBuilder app)
    {
        app.MapSignalR();
        ConfigureAuth(app);
    }
}
```

4. In order to reference our hub code on the Client, we will need to import/reference 2 scripts (Aside from the obvious jQuery reference). The main `jQuery.signalR-version.js` file and the generated `hubs.js` file that SignalR generates for our hub specifically. These resources may look like this:

```
// script tag src="/YourAppPath/scripts/jquery-1.6.4.js"
// script tag src="/YourAppPath/scripts/jquery.signalR-2.2.0.js"
// script tag src="/YourAppPath/signalr/hubs"
```

5. Since SignalR's JavaScript is built on top of jQuery (requires >= v1.6.4), the code for connecting and disconnecting to the hub should look fairly trivial. Here it is in all its' glory (wrapped in an IIFE) :

```
$(function() {
    //notice the camel casing of our hub name corresponding to the [HubName()] attribute
on the server
    var connection = $.connection.messagingHub;

    $.connection.hub.start().done(function () {
        alert("Connection Successful!");
    }).fail(function (reason) {
        console.log("SignalR connection failed: " + reason);
    });
});
```

6. As of right now, we should be able to run the app and establish a connection to the SignalR hub.

Using SignalR with Web API and JavaScript Web App, with CORS support.

Objective: Use SignalR for notification between Web API, and TypeScript/JavaScript based Web App, where Web API and the Web App is hosted in different domain.

Enabling SignalR and CORS on Web API: Create a standard Web API project, and install the following NuGet packages:

- Microsoft.Owin.Cors
- Microsoft.AspNet.WebApi.Cors
- Microsoft.AspNet.WebApi.Owin
- Microsoft.AspNet.SignalR.Core

After that you can get rid of the `Global.asax` and add a OWIN Startup class instead.

```
using System.Web.Http;
using System.Web.Http.Cors;
using Microsoft.Owin;
using Owin;

[assembly: OwinStartup(typeof(WebAPI.Startup), "Configuration")]
namespace WebAPI
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {

            var httpConfig = new HttpConfiguration();

            //change this configuration as you want.
            var cors = new EnableCorsAttribute("http://localhost:9000", "*", "*");
            httpConfig.EnableCors(cors);

            SignalRConfig.Register(app, cors);

            WebApiConfig.Register(httpConfig);

            app.UseWebApi(httpConfig);
        }
    }
}
```

Create the `SignalRConfig` class as follows:

```
using System.Linq;
using System.Threading.Tasks;
using System.Web.Cors;
using System.Web.Http.Cors;
using Microsoft.Owin.Cors;
using Owin;

namespace WebAPI
{
    public static class SignalRConfig
    {
        public static void Register(IAppBuilder app, EnableCorsAttribute cors)
        {

            app.Map("/signalr", map =>
            {
                var corsOption = new CorsOptions
                {
                    PolicyProvider = new CorsPolicyProvider
                    {
                        PolicyResolver = context =>
                        {
                            var policy = new CorsPolicy { AllowAnyHeader = true,
AllowAnyMethod = true, SupportsCredentials = true };

                            // Only allow CORS requests from the trusted domains.
                            cors.Origins.ToList().ForEach(o => policy.Origins.Add(o));

                            return Task.FromResult(policy);
                        }
                    }
                };
                map.UseCors(corsOption).RunSignalR();
            });
        }
    }
}
```

Till now we have just enabled SignalR with CORS on server side. Now lets see how you can publish events from server side. For this we need a `Hub`:

```
public class NotificationHub:Hub
{
    //this can be in Web API or in any other class library that is referred from Web API.
}
```

Now finally some code to actually broadcast the change:

```
public class SuperHeroController : ApiController
{
    [HttpGet]
    public string RevealAlterEgo(string id)
    {
        var alterEgo = $"The alter ego of {id} is not known.";
        var superHero = _superHeroes.SingleOrDefault(sh => sh.Name.Equals(id));
        if (superHero != null)
        {
            alterEgo = superHero.AlterEgo;

            /*This is how you broadcast the change.
             *For simplicity, in this example, the broadcast is done from a Controller,
             *but, this can be done from any other associated class library having access to
NotificationHub.
             */
            var notificationHubContext =
GlobalHost.ConnectionManager.GetHubContext<NotificationHub>();
            if (notificationHubContext != null)
            {
                var changeData = new { changeType = "Critical", whatHappened = $"Alter ego of
{id} is revealed." };

                //somethingChanged is an arbitrary method name.
                //however, same method name is also needs to be used in client.
                notificationHubContext.Clients.All.somethingChanged(changeData);
            }
        }
        return alterEgo;
    }
}
```

Thus, so far, we made the server side ready. For client side, we need `jQuery`, and `signalr` package. You may install both with `jspm`. Install the typings for both, if needed.

We will not be using the default generated JavaScript proxy. We will rather create a very simple class to handle the SignalR communication.

```
/**
 * This is created based on this gist: https://gist.github.com/donald-
slagle/bf0673b3c188f3a2559c.
 * As we are crreating our own SignalR proxy,
 * we don't need to get the auto generated proxy using `signalr/hubs` link.
 */
export class SignalRClient {

    public connection = undefined;
    private running: boolean = false;

    public getOrCreateHub(hubName: string) {
        hubName = hubName.toLowerCase();
        if (!this.connection) {
            this.connection = jQuery.hubConnection("https://localhost:44378");
        }

        if (!this.connection.proxies[hubName]) {
            this.connection.createHubProxy(hubName);
        }

        return this.connection.proxies[hubName];
    }

    public registerCallback(hubName: string, methodName: string, callback: (...msg: any[]) =>
void,
        startIfNotStarted: boolean = true) {

        var hubProxy = this.getOrCreateHub(hubName);
        hubProxy.on(methodName, callback);

        //Note: Unlike C# clients, for JavaScript clients,
        //      at least one callback needs to be registered,
        //      prior to start the connection.
        if (!this.running && startIfNotStarted)
            this.start();
    }

    start() {
        const self = this;
        if (!self.running) {
            self.connection.start()
                .done(function () {
                    console.log('Now connected, connection Id=' + self.connection.id);
                    self.running = true;
                })
                .fail(function () {
                    console.log('Could not connect');
                });
        }
    }
}
```

Lastly use this class to listen to change broadcasts, as follows:

```
/**
 * Though the example contains Aurelia codes,
 * the main part of SignalR communication is without any Aurelia dependency.
 */
import {autoinject, bindable} from "aurelia-framework";
import {SignalRClient} from "./SignalRClient";

@autoinject
export class SomeClass{

    //Instantiate SignalRClient.
    constructor(private signalRClient: SignalRClient) {
    }

    attached() {
        //To register callback you can use lambda expression...
        this.signalRClient.registerCallback("notificationHub", "somethingChanged", (data) => {
            console.log("Notified in VM via signalr.", data);
        });

        //... or function name.
        this.signalRClient.registerCallback("notificationHub", "somethingChanged",
this.somethingChanged);
    }

    somethingChanged(data) {
        console.log("Notified in VM, somethingChanged, via signalr.", data);
    }
}
```

Read Getting started with signalr online: https://riptutorial.com/signalr/topic/5633/getting-started-with-signalr

Chapter 2: Handling Connection Lifetimes

Parameters

Parameter	Details
stopCalled	This value tells you how a user disconnected, if its set to true then the user explicitly closed the connection, otherwise they timed out.

Remarks

It's worth noting that at while in these functions you still have access to the Context, therefore you can get the connectionId and identify who/what has disconnected.

Remember, a user can have **more then one connectionId **, so think about how you want to store all the connection IDs for one user

Examples

On connected

When ever a user connects to your hub, the `OnConnected()` is called. You can over ride this function and implement your own logic if you need to keep track of or limit the number of connections

```
  public override Task OnConnected()
    {
        //you logic here

        return base.OnConnected();
    }
```

On Disconnected

Overloading the disconnect function allows you to handle what to do when a user disconnects.

```
public override Task OnDisconnected(bool stopCalled)
{
    //Your disconnect logic here

    return base.OnDisconnected(stopCalled);
}
```

Example of informing others in a group that a user has disconnected

```
    /// <summary>
    /// Overrides the onDisconnected function and sets the user object to offline, this can be
checked when other players interacts with them...
    /// </summary>
    /// <param name="stopCalled"></param>
    /// <returns></returns>
    public override Task OnDisconnected(bool stopCalled)
    {
        var user = GetUserDictionaryData();
        if (user != null)
        {
            user.Status = PlayerStatus.offline;
            var playerJson = ReturnObjectAsJSON(user.Player);
            Clients.OthersInGroup(user.GroupId).userDisconnected(playerJson);
        }

        return base.OnDisconnected(stopCalled);
    }
```

No need to worry about connection in signalr.

If you want to notify other clients/users throughout the application ,you not need to worry about the connection because signalr new connection is created every time you visit other pages in the web app.

we can leverage the users feature of signalr to achieve the same. see the example below:

Here we are creating based on the projectid and all others users of that group get notified when we access the $.connection.notificationHub.server.NotifyOthersInGroup(projectid,projectname); from client javascript file

//Hub Example of SignalR

public class NotificationHub : Hub {

```
    public static readonly ConcurrentDictionary<string, HashSet<string>>
       ProjectLockUsers = new ConcurrentDictionary<string,HashSet<string>>();

    public void JoinGroup(int projectid)
    {
        string groupname = string.Format("ProjectLock_{0}", projectid);
        Groups.Add(Context.ConnectionId, groupname);

        AddUserToProjectLockGroup(groupname, Context.User.Identity.Name);
    }

    public void NotifyOthersInGroup(int projectid,string name)
    {
        string groupname = string.Format("ProjectLock_{0}", projectid);

        var allusers=null as HashSet<string>;

        if (ProjectLockUsers.TryGetValue(groupname, out allusers))
        {
            allusers.Remove(Context.User.Identity.Name);
            Clients.Users(allusers.ToList()).notifyUnlock(name);
        }
    }

    public override Task OnConnected()
    {
        return base.OnConnected();
    }
    public override Task OnDisconnected(bool stopCalled)
    {
        return base.OnDisconnected(stopCalled);
    }


    private void AddUserToProjectLockGroup(string projectId,string userId)
    {
        var userIds = null as HashSet<string>;


        if (Sessions.TryGetValue(projectId, out userIds) == false)
        {
            userIds = Sessions[projectId] = new HashSet<string>();
        }

        userIds.Add(userId);

        ProjectLockUsers[projectId] = userIds;
    }

}
```

Read Handling Connection Lifetimes online: https://riptutorial.com/signalr/topic/6527/handling-connection-lifetimes

Credits

S. No	Chapters	Contributors
1	Getting started with signalr	Community, Mark C., Sayan Pal, Scott Weldon
2	Handling Connection Lifetimes	ashish, Jay