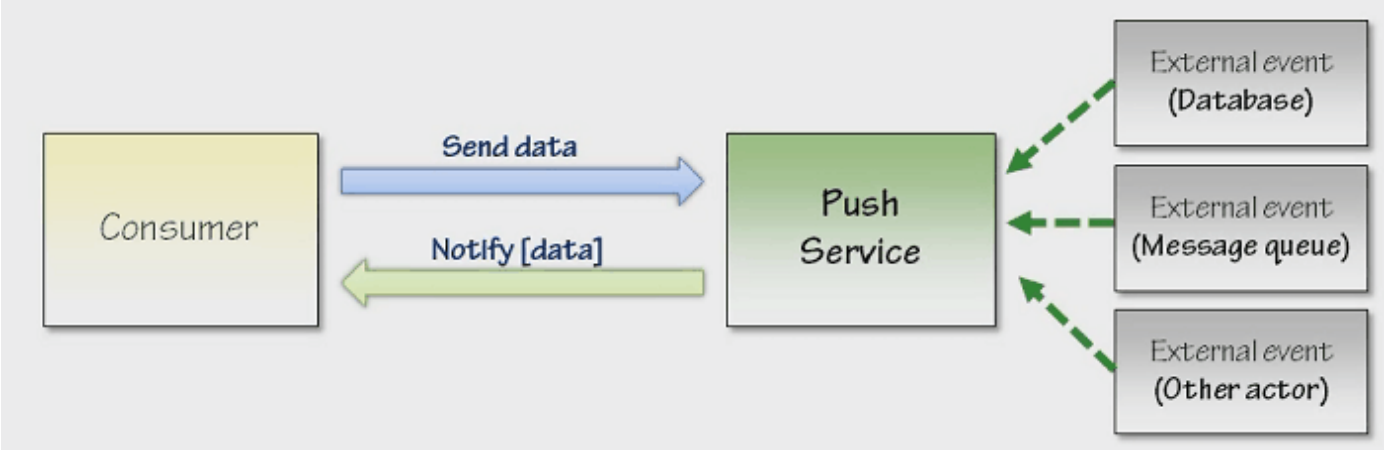
عنوان: **ارتباطات بلادرنگ و SignalR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۱ ۲۳:۲۳
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, jQuery, SignalR

زمانیکه صحبت از برنامه‌های بلادرنگ می‌شود با کاربرانی سر و کار داریم که نیاز دارند تا اطلاعات مورد نیاز خود را همواره و بلافاصله در آخرین وضعیت به روز آن مشاهده کنند. در این بین، کلاینت می‌خواهد یک برنامه وب باشد یا سیلورلایت و یا یک برنامه نوشته شده با WPF. حتی برنامه‌های موبایل را نیز باید به این لیست اضافه کرد.
در اینجا کلمه بلادرنگ به معنای ارسال اطلاعات از طرف سرور به کلاینت‌ها با فاصله زمانی بسیار کوتاهی از به روز رسانی اطلاعات صورت گرفته در سمت سرور است.
نمونه‌ای از این برنامه‌ها شامل موارد ذیل هستند:
- اطلاع رسانی همزمان به گروهی از کاربران
- جستجوهای زنده و به روز رسانی‌هایی از این دست
- نمایش بلادرنگ قیمت‌ها و وضعیت تجاری محصولات و سهام‌ها
- بازی‌های تعاملی
- برنامه‌های گروهی و تعاملی (مانند برنامه‌های Chat)
- برنامه‌های شبکه‌های اجتماعی (برای مثال پیام جدیدی دارید؛ شخص خاصی آنلاین یا آفلاین شد و امثال آن)

بنابراین به صورت خلاصه قصد داریم به ارائه بازخوردها و اطلاع رسانی‌های بلادرنگ یا نسبتا سریع و به روز از سمت سرور به کلاینت‌ها برسیم.
برای مثال یک دیتاگرید را درنظر بگیرید. دو کاربر در شبکه صفحه یکسانی را گشوده‌اند و یکی از آن‌ها مشغول به ویرایش و یا حذف اطلاعات است. در ارتباطات بلادرنگ کاربر یا کاربران دیگر نیز باید (یا بهتر است) در زمانیکه گرید یکسانی را گشوده‌اند، بلافاصله آخرین تغییرات را ملاحظه کنند. یا حتی حالتی را درنظر بگیرید که شخصی SQL Server management studio را گشوده و در آنجا مشغول به تغییر اطلاعات گردیده است. در این حالت نیز بهتر است آخرین تغییرات بلافاصله به اطلاع کاربران رسانده شوند.

معرفی الگوی Push service

البته باید دقت داشت که الگوی push service یک الگوی رسمی ذکر شده در گروه‌های مرسوم الگوهای طراحی نیست، اما مفهوم آن سرویسی است که چندین کار ذیل را انجام می‌دهد:
الف) پذیرش اتصالات از چندین مصرف کننده. مصرف کننده‌ها در اینجا الزاما محدود به کلاینت‌های وب یا دسکتاپ نیستند؛ می‌توانند حتی یک سرور یا سرویس دیگر نیز باشند.
ب) قادر است اطلاعات را به مصرف کننده‌های خود ارسال کند. این سرویس می‌تواند یک برنامه ASP.NET باشد یا حتی یک سرویس متداول ویندوز.
ج) در اینجا چندین منبع خارجی مانند یک بانک اطلاعاتی یا تغییرات رخ داده توسط یک سخت افزار که می‌توانند سبب بروز رخدادهایی در push service گردند نیز می‌تواند وجود داشته باشند. هر زمان که تغییری در این منابع خارجی رخ دهد، مایل هستیم تا مصرف کننده‌ها را مطلع سازیم.



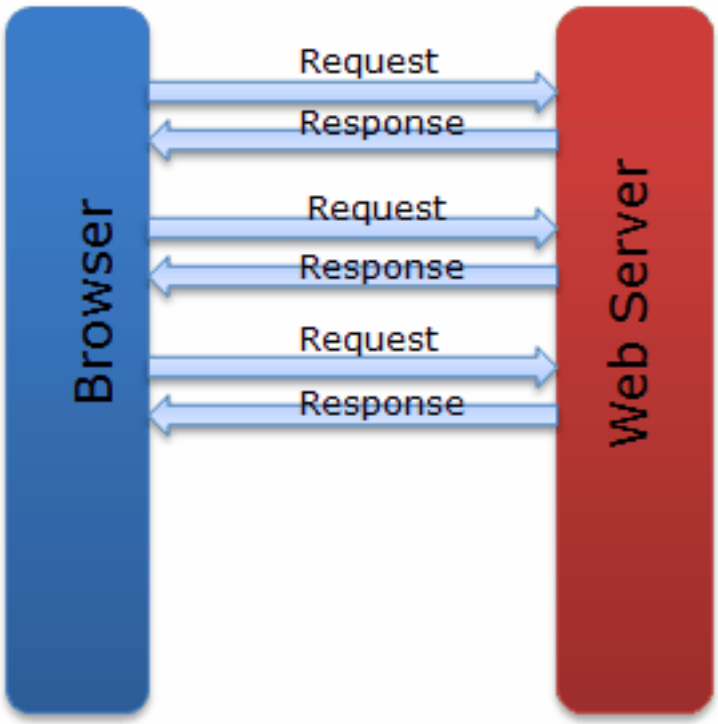
پروتکل HTTP و ارتباطات بلادرنگ

پروتکلی که در ارتباطات بلادرنگ مبتنی بر SignalR مورد استفاده قرار می‌گیرد، HTTP است و از قابلیت‌های Request و Response آن در اینجا بیشترین بهره برده می‌شود. پیاده سازی Push عموما بر مبنای یکی از روش‌های متداول زیر است:

1) Periodic polling

به این معنا که مثلا هر 10 ثانیه یکبار، کاری را انجام بده؛ مانند ارسال متناوب: آیا تغییری رخ داده؟ آیا تغییری رخ داده؟ و .... به همین ترتیب. این روش اصلا بهینه نبوده و منابع زیادی را خصوصا در سمت سرور مصرف خواهد کرد. برای مثال:

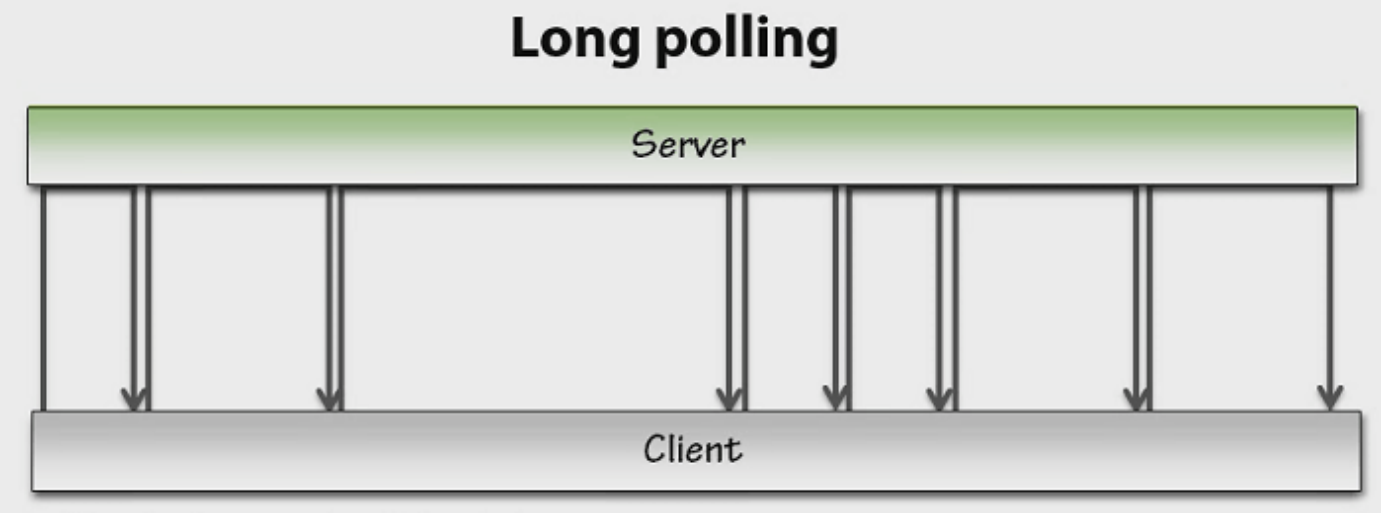
```
function getInfo() {
        $.ajax("url", function ( newInfo){
                if ( newInfo != null) {
                    // do something with the data
                }
        });
    // poll again after 20 seconds
    setTimeout(getInfo,20000);
}
// start the polling loop
getInfo();
```



2) Long polling

به آن HTTP Streaming یا Comet هم گفته می‌شود. این روش نسبتا هوشمند بوده و کلاینت اتصالی را به سرور برقرار خواهد کرد. سرور در این حالت تا زمانیکه اطلاعاتی را در دسترس نداشته باشد، پاسخی نخواهد داد. برای نمونه:

```
function getNewInfo(){
  $.ajax("url", function (newinfo) {
      // do something with the data
  // start the new request
      getNewINfo();
  });
}
// start the polling loop
getNewInfo();
```



این روش نسبت به حالت Periodic polling بهینه‌تر است اما نیاز به اتصالات زیادی داشته و همچنین تردهای بسیاری را در سمت سرور به خود مشغول خواهد کرد.

3) Forever frame

فقط در IE پشتیبانی می‌شود. در این روش یک Iframe مخفی توسط مرورگر تشکیل شده و از طریق آن درخواستی به سرور ارسال می‌شود. سپس سرور متناوبا با تزریق اسکریپت‌هایی به این Iframe سبب فراخوانی مجدد وضعیت خود می‌گردد. در این روش نیز به ازای هر درخواست و پاسخ، ارتباطات گشوده و بسته خواهند شد.

4) Server Sent Events یا SSE

این مورد جزو استاندارد HTML5 است. در اینجا اتصالی برقرار شده و داده‌ها از طریق اتصالات HTTP منتقل می‌شوند.

```
var eventSrc = new EventSource("url");
    // register event handler for the message
    eventSrc.addEventListener( "message",function (evt) {
    //process the data
});
```

این روش نیز بسیار شبیه به حالت long polling است. سرور تا زمانیکه اطلاعاتی را برای پاسخ دهی فراهم نداشته باشد، اتصال را باز نگه می‌دارد. به این ترتیب از لحاظ مقیاس پذیری گزینه بهتری است (نسبت به حالتیکه مدام اتصال برقرار و قطع می‌شود). اکثر مرورگرها منهای نگارش‌های قدیمی IE از این روش پشتیبانی می‌کنند.
تنها تفاوت آن با حالت long polling در این است که پس از ارائه پاسخ به کلاینت، اتصال را قطع نمی‌کند. Long polling نیز

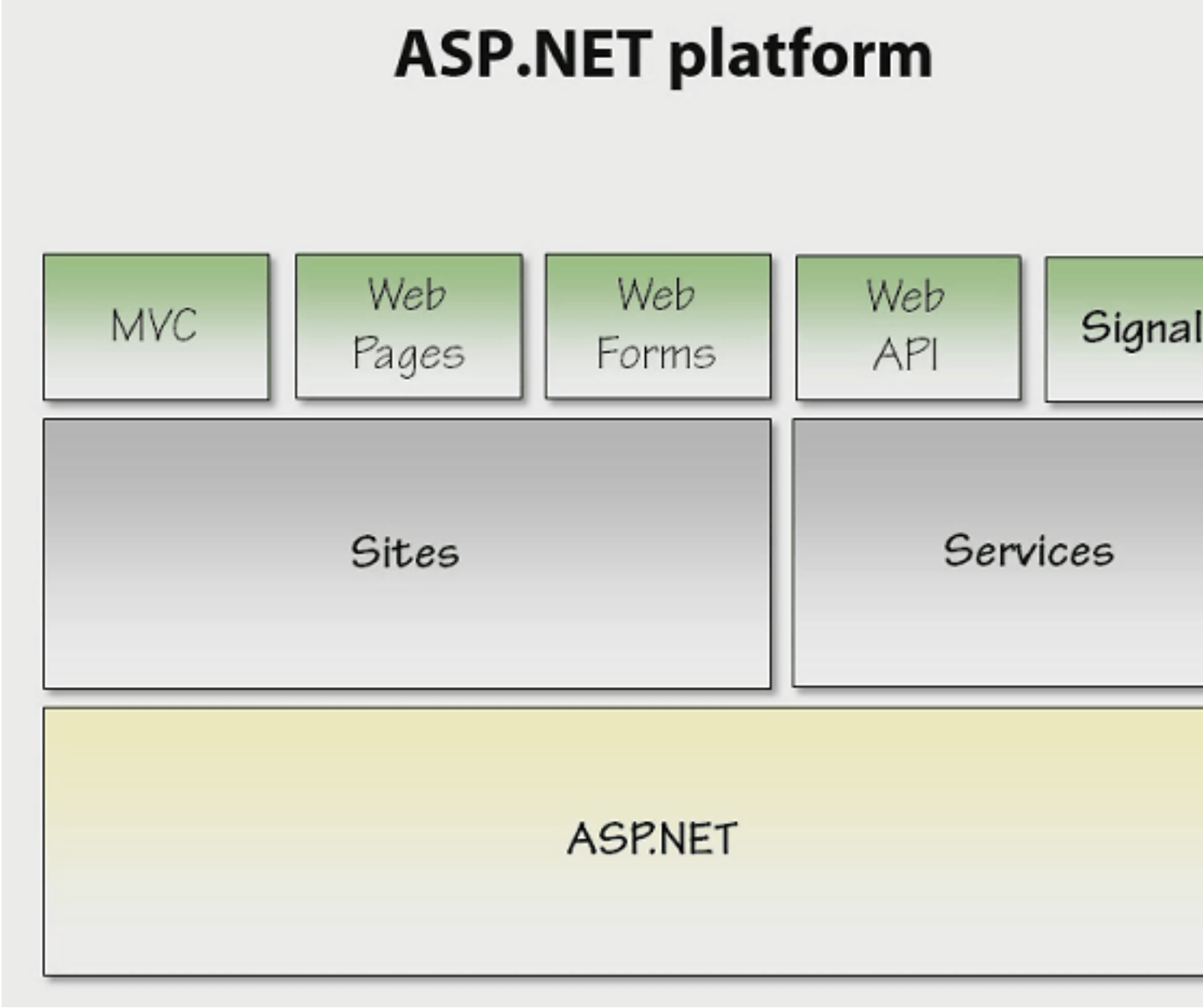
اتصال را باز نگه می‌دارد، اما این اتصال را بلافاصله پس از ارائه پاسخ، می‌بندد.

5) Web sockets

Web sockets در سکوی کاری ویندوز، تنها در ویندوزهای 8، ویندوز سرور 2012 و دات نت 4 و نیم پشتیبانی می‌شود. هرچند این روش در حال حاضر به عنوان بهترین روش Push مطرح است اما به دلیل محدودیتی که یاد شد، مدتی طول خواهد کشید تا استفاده گسترده‌ای پیدا کند.

```
var socket = new WebSocket("url");
socket.onmessage = function (msg) {
var newInfo = msg.data;
// do something with the data
}
// client can also send request to server
socket.send(.... )
```

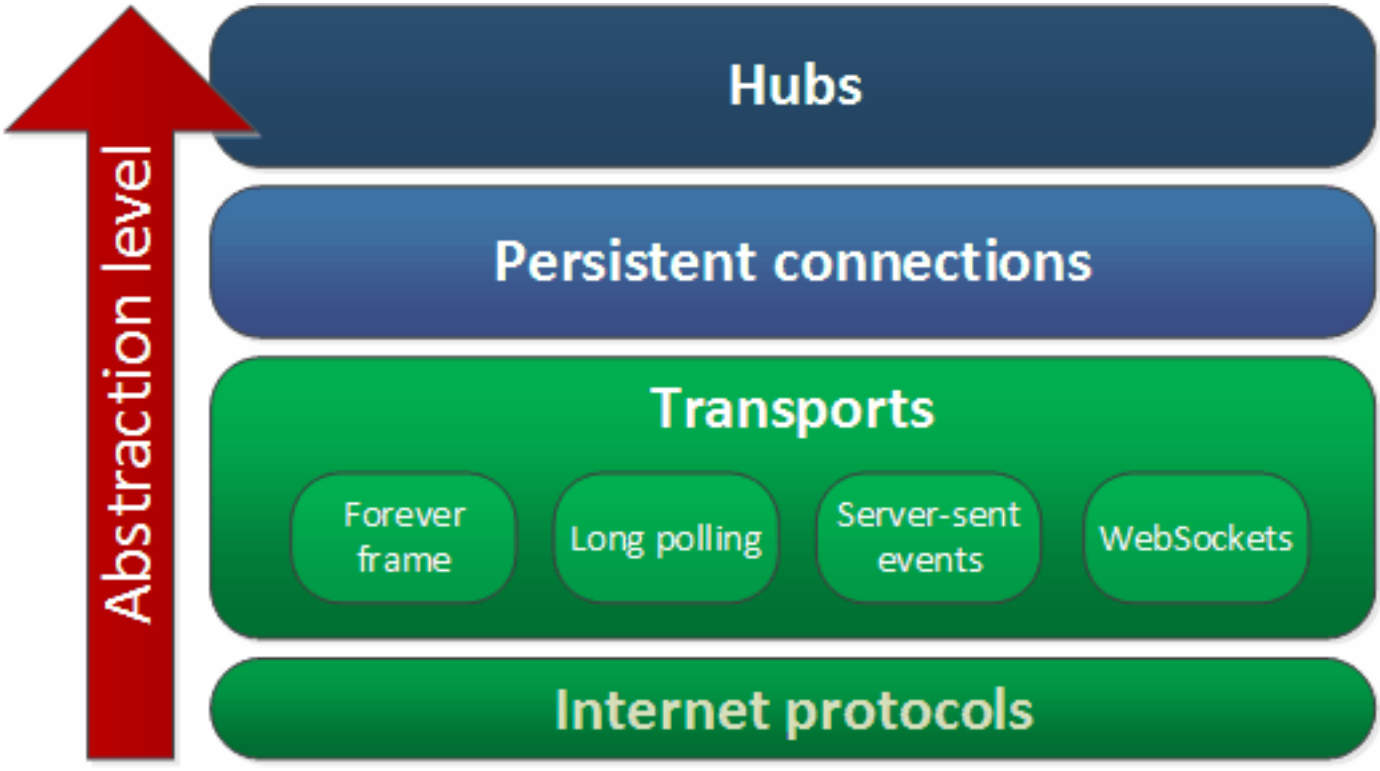
با این اوصاف آیا راه حل بهتر و میانه‌تری وجود دارد؟
بلی. اگر به وضعیت فعلی سکوی کاری ASP.NET نگاه کنیم:



SignalR را می‌توان مشاهده کرد که در گروه ساخت سرویس‌های آن قرار گرفته است. همانطور که ملاحظه می‌کنید، این سرویس جدید آنچنان وابستگی به سایر اجزای آن نداشته و می‌تواند خارج از ASP.NET نیز مورد استفاده قرار گیرد.

SignalR چیست؟

SignalR راه حلی است سمت سرور برای نوشتن push services. همچنین به همراه کتابخانه‌های سمت کاربری است که ارتباطات push services را در انواع و اقسام سکوهای کاری میسر می‌سازد. SignalR سورس باز بوده و برای اعمال غیرهمزمان (asynchronous) بهینه سازی شده است.
SignalR بر اساس مدل ذهنی اتصالات ماندگار (persistent connections) طراحی شده است. اتصالات ماندگار را باید به عنوان اتصالاتی سریع و غیرطولانی درنظر گرفت. در اینجا Signal یک اتصال است که اطلاعاتی به آن ارسال می‌گردد و هدف، انتقال قطعات کوچکی از اطلاعات است و هدف، ارسال حجم عظیمی از اطلاعات نیست. برای مثال اطلاع رسانی سریعی صورت گیرد که تغییراتی رخ داده است و سپس ادامه کار و دریافت اطلاعات واقعی توسط فرآیندهای متداول مثلا HTTP GET انجام شود. البته باید دقت داشت SignalR نیز نهایتا از یکی از 5 روش push بررسی شده در این قسمت استفاده می‌کند. اما بر اساس توانایی‌های کلاینت و سرور، به صورت هوشمند بهترین و بهینه‌ترین انتخاب را به کاربر ارائه می‌دهد.
اتصالات ماندگار قسمت سطح پایین SignalR را تشکیل می‌دهند. سطح بالاتر آن که این مفاهیم را به شکلی کپسوله شده ارائه می‌دهد، Hubs نام دارد که پایه اصلی دوره جاری را تشکیل خواهد داد.



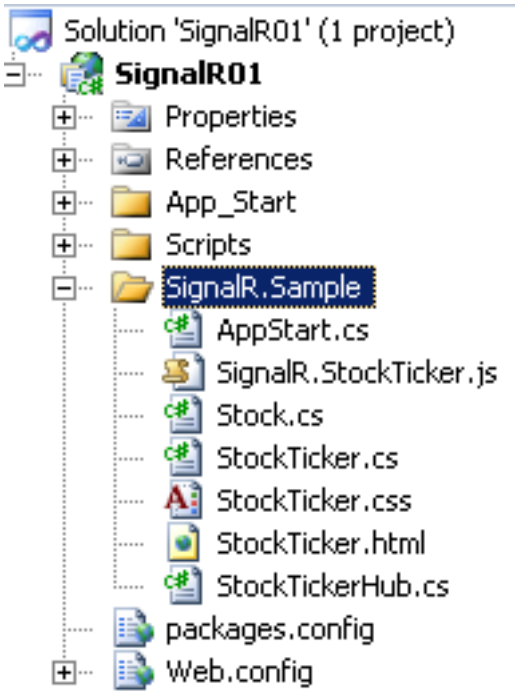
همانطور که عنوان شد، SignalR سورس باز بوده و دارای مخزن کدی عمومی در GitHub است. همچنین بسته‌های تشکیل دهنده‌ی آن از طریق NuGet نیز قابل دریافت هستند. این بسته‌ها شامل هسته SignalR و کلاینت‌های آن مانند کلاینت‌های WinRT، سیلورلایت، jQuery، ویندوز فون8 و امثال آن هستند.

شروع کار با SignalR

تیم SignalR مثالی مقدماتی از نحوه کار با SignalR را به صورت یک بسته NuGet ارائه داده‌اند که از طریق آدرس و فرمان ذیل قابل دریافت است:

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

قبل از اینکه این مثال را دریافت کنید نیاز است ابتدا یک برنامه ASP.NET جدید را آغاز نمائید (تفاوتی نمی‌کند که MVC باشد یا Web forms). سپس دستور فوق را فراخوانی کنید.



پس از دریافت مثال، یکبار پروژه را کامپایل کرده و سپس بر روی فایل StockTicker.html آن کلیک راست نموده و گزینه مشاهده در مرورگر را انتخاب کنید. همچنین برای اینکه این مثال را بهتر مشاهده کنید، بهتر است دو وهله از مرورگر را باز کرده و آدرس باز شده را در آن بررسی کنید تا اعمال تغییرات همزمان به کلاینت‌های متفاوت را بهتر بتوان بررسی و مشاهده کرد.

ASP.NET SignalR Stock Ticker Sample

Open Market | Close Market | Reset

Live Stock Table

Symbol	Price	Open	High	Low	Change	%
GOOG	570.80	570.3	570.84	569.68	▲ 0.5	0.09%
MSFT	30.39	30.31	30.39	30.31	▲ 0.08	0.26%
APPL	576.81	578.18	578.18	576.81	▼ -1.37	-0.24%

Live Stock Ticker

OG 570.80 ▲ *0.5 (0.09%)* **MSFT** 30.39 ▲ *0.08 (0.26%)* **APPL** 57

نظرات خوانندگان

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱:۲

مهندس مثل همیشه عالی بود.
میشه بفرمایین تفاوتش با برنامه نویسی سوکت چی می‌تونه باشه برای برنامه‌های چت؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۲ ۸:۴۸

در اینجا
- در سمت کلاینت فایروال مانعی نخواهد بود چون ارتباطات از طریق مرورگر (هم می‌تواند) انجام می‌شود.
- باز هم نهایتا از سوکت‌ها استفاده خواهد شد اما در سطحی بالاتر و بدون درگیری با جزئیات آن‌ها. اینبار یک فریم ورک آماده، تست شده و تهیه شده برفراز سوکت‌های دات نت و ویندوز در اختیار شما است. به علاوه این فریم ورک فراتر است از صرفا برقراری ارتباط و ارسال داده، بلکه حالت امکان اجرای متدهای خاصی در سمت کلاینت یا سرور را هم دارا است (بحث قسمت بعد).
- تنوع کلاینت‌ها. محدود به یک برنامه ویندوزی نخواهید بود. مثلا امکان استفاده از یک کلاینت jQuery، که برای اجرا، نیازی به سطح دسترسی خاصی ندارد، یا حتی یک کلاینت سیلورلایت یا اندروید و غیره هم برای آن تهیه کرده‌اند.
- امکان استفاده از IIS به عنوان سرور. همین مساله یعنی درگیر نشدن با مسایلی مانند مقیاس پذیری، مدیریت تعداد کانکشن‌های بالا و امثال آن.
- امکان یکپارچه کردن یک برنامه سرویس دهنده هاب با یک برنامه ASP.NET در کنار هم در یک پروژه.
و ...

نویسنده: Alireza Godazchian
تاریخ: ۱۳۹۲/۰۱/۱۳ ۱۰:۵۳

خیلی عالی توضیح می‌دهید
از شما واقعا سپاسگذاریم....

نویسنده: شهروزجعفری
تاریخ: ۱۳۹۲/۰۱/۱۳ ۱۶:۳۵

سلام آقای نصیری حرف R در SignalR به چه معناست

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۳ ۱۶:۴۳

real time یا بلادرنگ

نویسنده: مرتضی
تاریخ: ۱۳۹۲/۰۱/۱۳ ۲۳:۰

سلام-

بخاطر آموزشاتون واقعا سپاسگذارم-

لطفا میشه یه مثال با web sockets بذارید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۳ ۲۳:۲۷

لازم هست تمام قسمت‌ها را یکبار مطالعه کنید. در قسمت دوم (نگاهی به SignalR Hubs) در مورد نحوه انتخاب لایه transport به صورت خودکار بحث شده. در قسمت سوم (نگاهی به SignalR Clients) در طی یک نکته ویژه عنوان شده که همین مثال مورد بحث رو به چه صورتی و تنها در کجا می‌تونید بر اساس webSocket اجرا کنید.

نویسنده: مرتضی
تاریخ: ۱۳۹۲/۰۱/۱۴ ۰:۱

مهندس کامل مطالعش کردم-

منظورم یه سرفصل جدا با web sockets بود -

چون با web sokcets که کار کردم به یه مشکل کوچیک برخوردم-

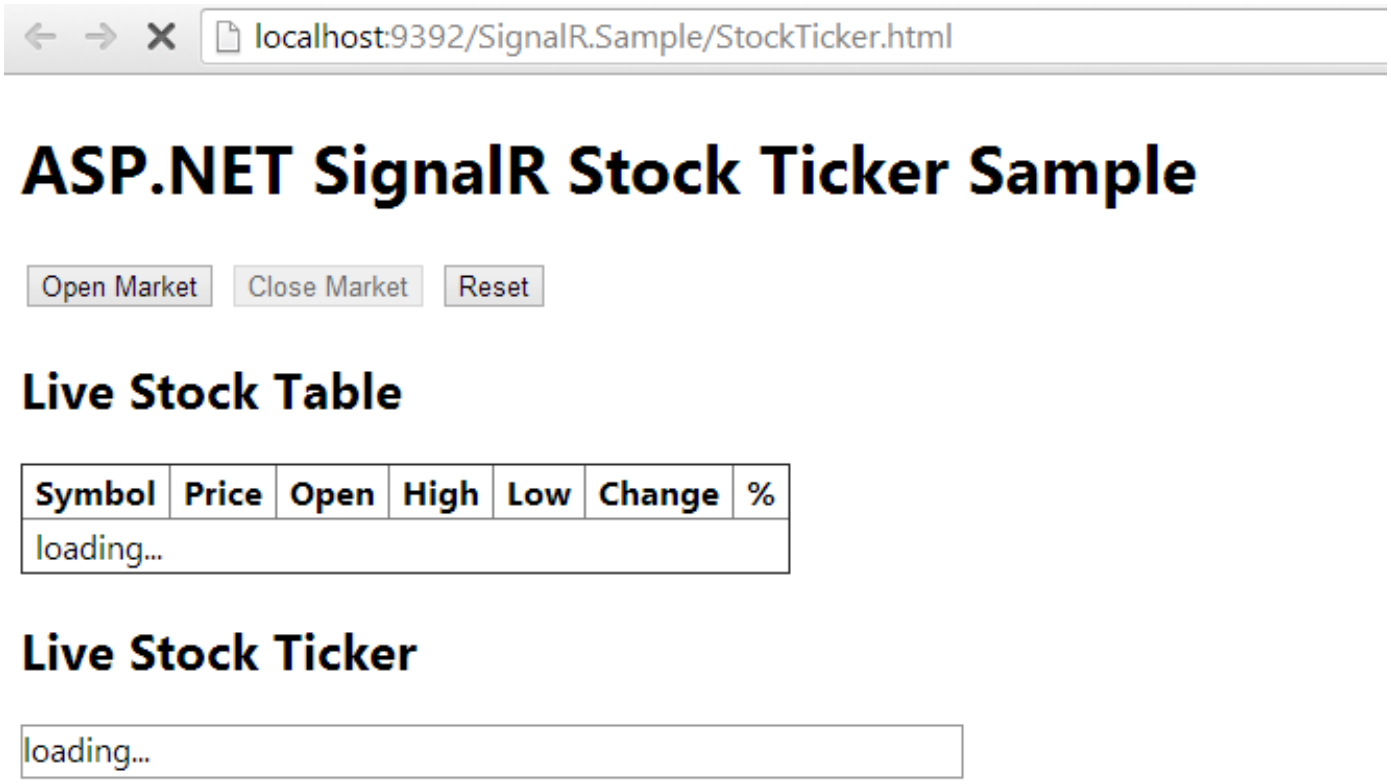
مرسی

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۴ ۰:۱۷

- نیازی به سرفصل جدا ندارد با توجه به خودکار بودن انتخاب لایه انتقال بر اساس توانایی سکوی کاری مورد استفاده (در حین کار با SignalR، وب سوکت فقط در ویندوز 8، IIS8 به همراه پروژه‌ای مبتنی بر دات نت 4 و نیم پشتیبانی می‌شود). سایر بحث‌ها و نکات یکی است و تفاوتی نمی‌کند. زمانیکه با Hub کار می‌کنید در لایه‌ای قرار دارید که این جزئیات از شما مخفی می‌شود و کار انتخاب خودکار است (تصویر abstraction level مطلب جاری).
+ دوره‌ها در سایت قسمتی دارند جهت پرسش و پاسخ اختصاصی که می‌شود مشکلات و سؤالات مرتبط به دوره را در آنجا ارسال کرد با توضیح بیشتر.

نویسنده: اردلان شاه قلی
تاریخ: ۱۳۹۲/۱۰/۲۹ ۱۸:۱۳

سلام من بعد از اضافه کردن سمپل نام برده و مشاهده‌ی صفحه ی StockTicker.html ، آنچه را شما در بالا نمایش داده اید نمی‌بینم.؟ تصویر چیزی که من می‌بینم به این صورت می‌باشد.



نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۹ ۱۸:۳۷

به نظر اسکریپت‌های آن بارگذاری نشده‌اند. در کروم روی دکمه F12 کلیک کنید تا کنسول آن ظاهر شود. بعد بررسی کنید آیا خطایی در برگه network آن گزارش شده یا حتی در کنسول لاگ‌های آن که خطاهای جاوا اسکریپتی را نمایش می‌دهد. با فایرباگ هم می‌شود این نوع برنامه‌ها را دیباگ کرد.

اطلاعات بیشتر: « عیب یابی و دیباگ برنامه‌های SignalR »

همچنین این مثال‌ها را از اینجا نیز می‌توانید دریافت کنید: SignalRSamples.zip

نویسنده: vici
تاریخ: ۱۳۹۲/۱۰/۳۰ ۲۱:۱۰

سلام

آقای نصیری برای کارهای ویندوزی مثل datagrid چه روشی رو پیشنهاد می‌کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۳۰ ۲۱:۱۷

در چهارمین قسمت این سری « نگاهی به گزینه‌های مختلف مهیای جهت میزبانی SignalR » شده‌است.

نویسنده: ح مرادف
تاریخ: ۱۳۹۲/۱۱/۱۹ ۱۸:۳۲

فوق العاده بود.
چند وقت پیش برای کار مصاحبه دادم و توی اون مصاحبه مطالعه در زمینه SignalR و angularJS بهم پیشنهاد شد.
خیلی وقت بود فرصت نمی‌کردم توی نت درباره این مورد سرچ کنم (مطلب انگلیسی خوشم نمیاد).
تا اینکه به دنبال سرچ درباره MVVM به این سایت عالی برخورد کردم.
واقعا کارتون یکه ....
تازه فهمیدم که اطلاعاتم خیلی قدیمیه و باید سریع بروزش کنم.
با تشکر از مقاله عالیتون.
خدا قوت
یا حق

نویسنده: ساده
تاریخ: ۱۳۹۳/۰۳/۱۱ ۱۴:۴۲

سلام

```
GET http://localhost:7186/signalr/hubs 500 (Internal Server Error)
```

نشون می‌ده.
من دایرکتوری پروژه رو چک کردم
اما اصلا چنین شاخه ای نیست تو پروژه که!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۱۱ ۱۴:۴۹

- در مورد مسیر *پویای* signalr/hubs در قسمت بعدی بیشتر بحث شده‌است.
- برای خطایابی نیاز به توضیحات بیشتری هست. اطلاعات بیشتر
- در کنسول لاگ‌های آن خطاهای جاوا اسکریپتی را نمایش می‌دهد. با فایرباگ هم می‌شود این نوع برنامه‌ها را دیباگ کرد.

اطلاعات بیشتر: « عیب یابی و دیباگ برنامه‌های SignalR »

عنوان: **نگاهی به SignalR Hubs**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۲ ۹:۲۹
آدرس: www.dotnettips.info
برچسب‌ها: 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

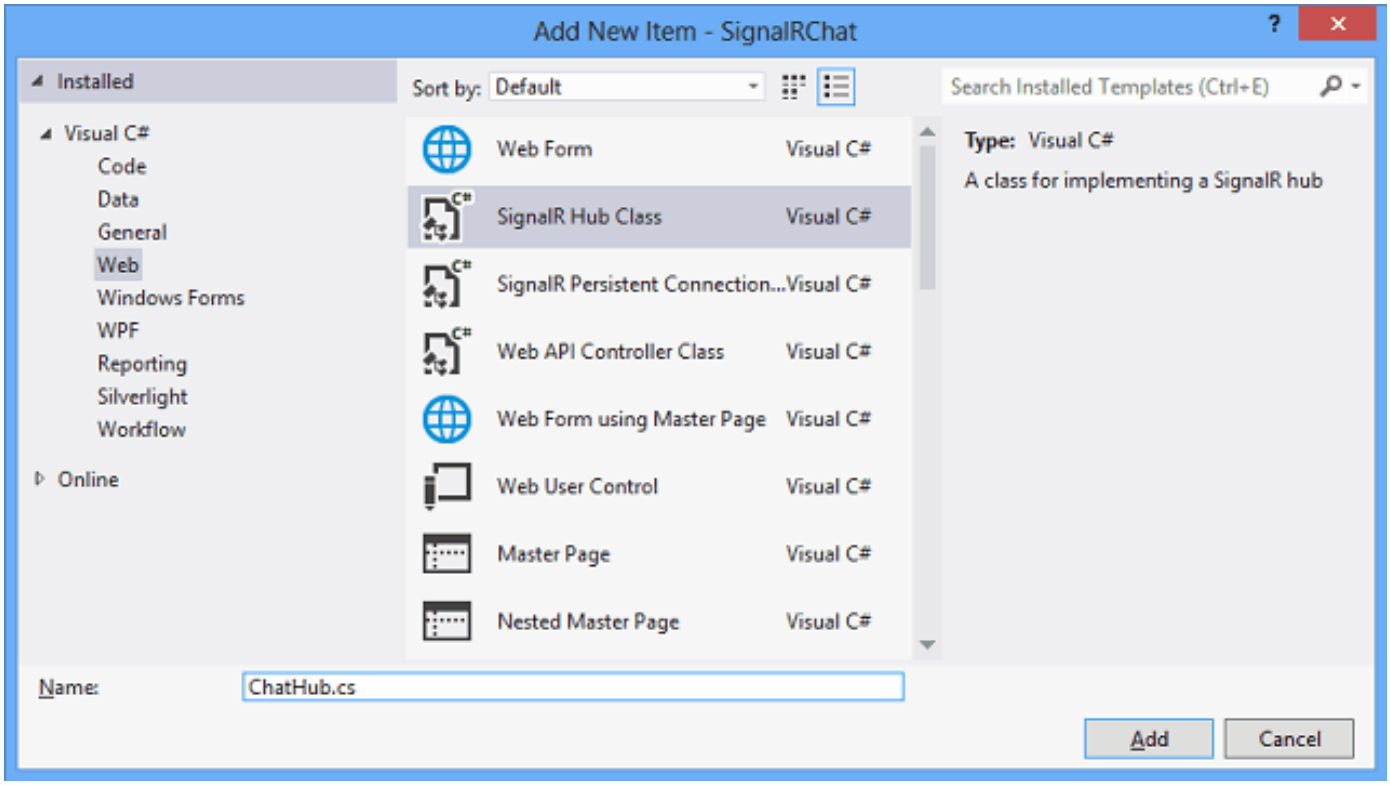
- متدهای public کلاس‌های Hubs از طریق دنیای خارج قابل فراخوانی هستند.
- ارسال اطلاعات به کلاینت‌ها از طریق فراخوانی متدهای سمت کلاینت انجام خواهد شد. (نحوه تعریف این متدها در سمت سرور بر اساس قابلیت‌های dynamic اضافه شده به دات نت 4 است که در ادامه در مورد آن بیشتر بحث خواهد شد)

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.

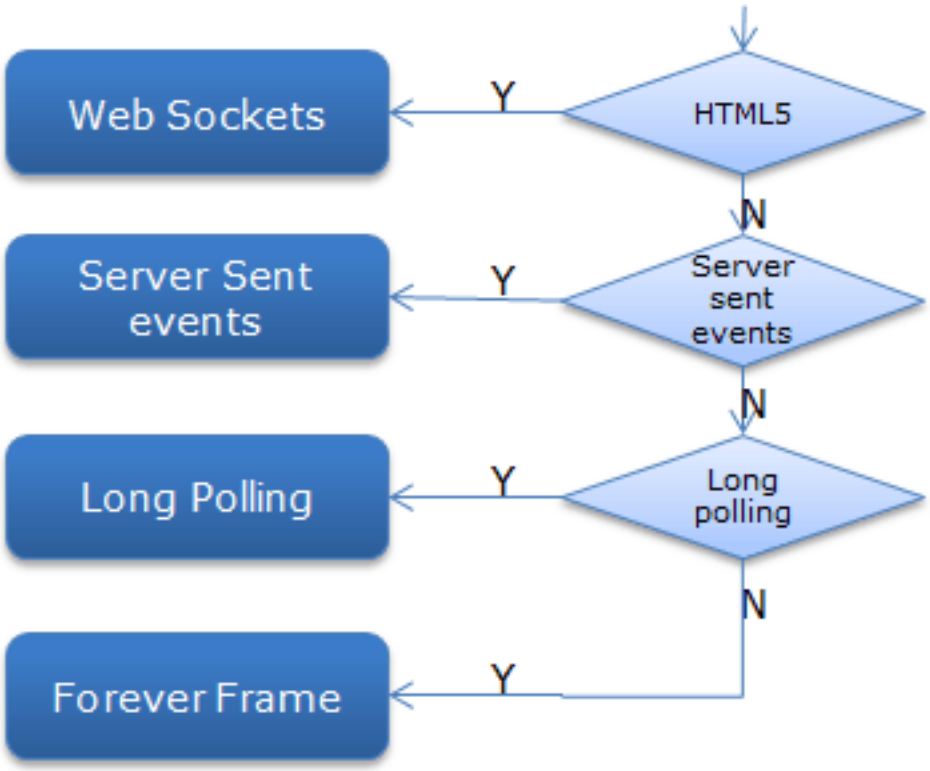
تا اینجا ما موفق شدیم اولین Hub خود را تشکیل دهیم.

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 metadata js 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
```

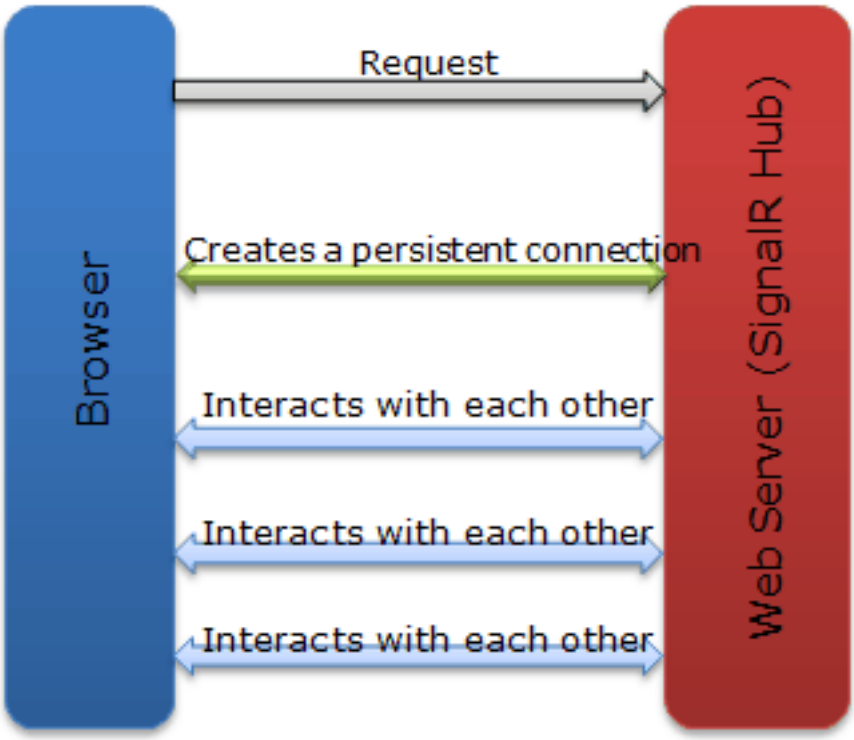


به این معنا که اگر برای مثال امکانات Web sockets مهیا بود، در همینجا کار انتخاب نحوه انتقال اطلاعات خاتمه یافته و Web sockets انتخاب می‌شود.
تمام این مراحل نیز خودکار است و نیازی نیست تا برای تنظیمات آن کار خاصی صورت گیرد. البته در سمت کلاینت، امکان انتخاب یکی از موارد یاد شده به صورت صریح نیز وجود دارد.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

عنوان: **نگاهی به SignalR Clients**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۲:۱۱
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, jQuery, SignalR

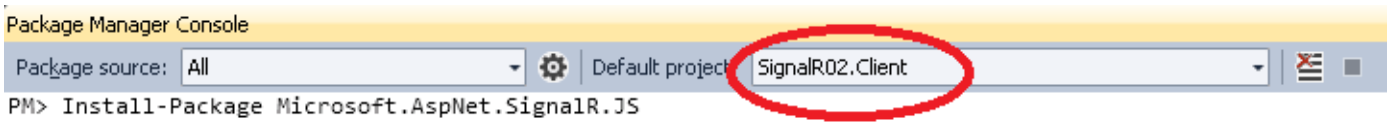
در قسمت قبل موفق به ایجاد اولین Hub خود شدیم. در ادامه، برای تکمیل برنامه نیاز است تا کلاینتی را نیز برای آن تهیه کنیم. مصرف کنندگان یک Hub می‌توانند انواع و اقسام برنامه‌های کلاینت مانند jQuery Clients و یا حتی یک برنامه کنسول ساده باشند و همچنین Hubهای دیگر نیز قابلیت استفاده از این امکانات Hubهای موجود را دارند. تیم SignalR امکان استفاده از Hubهای آن‌را در برنامه‌های دات نت 4 به بعد، برنامه‌های WinRT، ویندوز فون 8، سیلورلایت 5، jQuery و همچنین برنامه‌های CPP نیز مهیا کرده‌اند. به علاوه گروه‌های مختلف نیز با توجه به سورس باز بودن این مجموعه، کلاینت‌های iOS via Mono، iOS Native و Android via Mono را نیز به این لیست اضافه کرده‌اند.

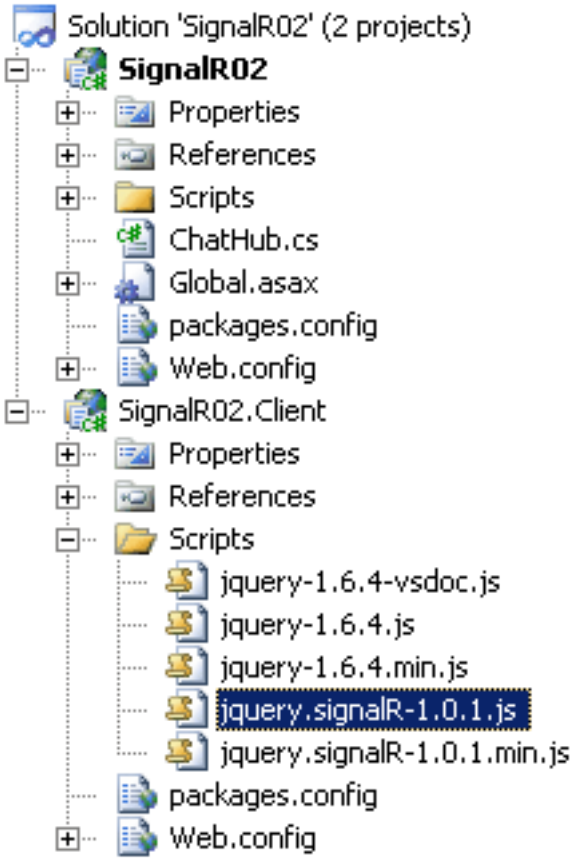
بررسی کلاینت‌های jQuery

با توجه به پروتکل مبتنی بر JSON سیگنال‌آر، استفاده از آن در کتابخانه‌های جاوا اسکریپتی همانند jQuery نیز به سادگی مهیا است. برای نصب آن نیاز است در کنسول پاور شل نوگت، دستور زیر را صادر کنید:

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

برای نمونه به solution پروژه قبل، یک برنامه وب خالی دیگر را اضافه کرده و سپس دستور فوق را بر روی آن اجرا نمائید. در این حالت فقط باید دقت داشت که فرامین بر روی کدام پروژه اجرا می‌شوند:





با استفاده از افزونه jQuery SignalR، به دو طریق می‌توان به یک Hub اتصال برقرار کرد:
الف) استفاده از فایل proxy تولید شده آن (این فایل، در زمان اجرای برنامه تولید می‌شود و یا امکان استفاده از آن به کمک ابزارهای کمکی نیز وجود دارد)
نمونه‌ای از آن‌را در قسمت قبل ملاحظه کردید؛ همان فایل تولید شده در مسیر signalr/hubs/ برنامه. به نوعی به آن Service contract نیز گفته می‌شود (ارائه متادیتا و قراردادهای کار با یک سرویس Hub). این فایل همانطور که عنوان شد به صورت پویا در زمان اجرای برنامه ایجاد می‌شود.
امکان تولید آن توسط برنامه کمکی signalr.exe نیز وجود دارد؛ برای دریافت آن می‌توان از طریق NuGet اقدام کرد (بسته Microsoft.AspNet.SignalR.Utils) که نهایتا در پوشه packages قرار خواهد گرفت. نحوه استفاده از آن نیز به صورت زیر است:

```
Signalr.exe ghp http://localhost/
```

در این دستور ghp مخفف generate hub proxy است و نهایتا فایلی را به نام server.js تولید می‌کند.

ب) بدون استفاده از فایل proxy و به کمک روش late binding (انقیاد دیر هنگام)

برای کار با یک Hub از طریق jQuery مراحل ذیل باید طی شوند:
1) ارجاعی به Hub باید مشخص شود.
2) روال‌های رخدادگردان تنظیم گردند.
3) اتصال به Hub برقرار گردد.
4) متدی فراخوانی شود.

در اینجا باید دقت داشت که امکانات Hub به صورت خواص

```
$.connection
```

در سمت کلاینت جی‌کوئری، در دسترس خواهند بود. برای مثال:

```
$.connection.chatHub
```

و نام‌های بکارگرفته شده در اینجا مطابق روش‌های متداول نام گذاری در جاوا اسکریپت، camel case هستند.

خوب، تا اینجا فرض بر این است که یک پروژه خالی ASP.NET را آغاز و سپس فرمان نصب Microsoft.AspNet.SignalR.JS را نیز همانطور که عنوان شد، صادر کرده‌اید. در ادامه یک فایل ساده html را به نام chat.htm، به این پروژه جدید اضافه کنید (برای استفاده از کتابخانه جاوا اسکریپتی SignalR الزامی به استفاده از صفحات کامل پروژه‌های وب نیست).

```
<!DOCTYPE>
<html>
<head>
    <title></title>
    <script src="Scripts/jquery-1.6.4.min.js" type="text/javascript"></script>
    <script src="Scripts/jquery.signalR-1.0.1.min.js" type="text/javascript"></script>
    <script src="http://localhost:1072/signalr/hubs" type="text/javascript"></script>
</head>
<body>
    <div>
        <input id="txtMsg" type="text" /><input id="send" type="button" value="send msg" />
        <ul id="messages">
        </ul>
    </div>
    <script type="text/javascript">
        $(function () {
            var chat;
            $.connection.hub.logging = true; //اطلاعات بیشتری را در جاوا اسکریپت کنسول مرورگر لاگ
می‌کند
            chat = $.connection.chat; //این نام مستعار پیشتر توسط ویژگی نام هاب تنظیم شده است
            chat.client.hello = function (message) {
                //متدی که در اینجا تعریف شده دقیقا مطابق نام متد پویایی است که در هاب تعریف شده است
                //به این ترتیب سرور می‌تواند کلاینت را فراخوانی کند
                $("#messages").append("<li>" + message + "</li>");
            };
            $.connection.hub.start(/*{ transport: 'longPolling' }*/); // فاز اولیه ارتباط را آغاز می‌کند

            $("#send").click(function () {
                // Hub's `SendMessage` should be camel case here
                chat.server.sendMessage($("#txtMsg").val());
            });
        });
    </script>
</body>
</html>
```

کدهای آن‌را به نحو فوق تغییر دهید.

توضیحات:

همانطور که ملاحظه می‌کنید ابتدا ارجاعاتی به jquery و jquery.signalR-1.0.1.min.js اضافه شده‌اند. سپس نیاز است مسیر دقیق فایل پروکسی هاب خود را نیز مشخص کنیم. اینکار با تعریف مسیر signalr/hubs انجام شده است.

```
<script src="http://localhost:1072/signalr/hubs" type="text/javascript"></script>
```

در ادامه توسط تنظیم connection.hub.logging سبب خواهیم شد تا اطلاعات بیشتری در javascript console مرورگر لاگ شود.

سپس ارجاعی به هاب تعریف شده، تعریف گردیده است. اگر از قسمت قبل به خاطر داشته باشید، توسط ویژگی HubName، نام chat را برگزیدیم. بنابراین connection.chat ذکر شده دقیقا به این هاب اشاره می‌کند.

سپس سطر chat.client.hello مقدار دهی شده است. متد hello، متدی dynamic و تعریف شده در سمت هاب برنامه است. به این ترتیب می‌توان به پیام‌های رسیده از طرف سرور گوش فرا داد. در اینجا، این پیام‌ها، به li ایی با id مساوی messages اضافه می‌شوند.

سپس توسط فراخوانی متد connection.hub.start، فاز negotiation شروع می‌شود. در اینجا حتی می‌توان نوع transport را نیز صریحا انتخاب کرد که نمونه‌ای از آن را به صورت کامنت شده جهت آشنایی با نحوه تعریف آن مشاهده می‌کنید. مقادیر قابل استفاده در آن به شرح زیر هستند:

```
- webSockets
- forverFrame
- serverSentEvents
- longPolling
```

سپس به رویدادهای کلیک دکمه send گوش فرا داده و در این حین، اطلاعات TextBox ایی با id مساوی txtMsg را به متد SendMessage هاب خود ارسال می‌کنیم. همانطور که پیشتر نیز عنوان شد، در سمت کلاینت، تعریف متد SendMessage باید camel case باشد.

اکنون به صورت جداگانه یکبار برنامه hub را در مرورگر باز کنید. سپس بر روی فایل chat.htm کلیک راست کرده و گزینه مشاهده آن را در مرورگر نیز انتخاب نمائید (گزینه View in browser منوی کلیک راست).

خوب! پروژه کار نمی‌کند! برای اینکه مشکلات را بهتر بتوانید مشاهده کنید نیاز است به JavaScript Console مرورگر خود مراجعه نمائید. برای مثال در مرورگر کروم دکمه F12 را فشرده و برگه Console آن‌را باز کنید. در اینجا اعلام می‌کند که فاز negotiation قابل انجام نیست؛ چون مسیر پیش فرضی را که انتخاب کرده است، همین مسیر پروژه دومی است که اضافه کرده‌ایم (کلاینت ما در پروژه دوم قرار دارد و نه در همان پروژه اول هاب).
برای اینکه مسیر دقیق hub را در این حالت مشخص کنیم، سطر زیر را به ابتدای کدهای جاوا اسکریپتی فوق اضافه نمائید:

```
$.connection.hub.url = 'http://localhost:1072/signalr'; //چون در یک پروژه دیگر قرار داریم
```

اکنون اگر مجدا سعی کنید، باز هم برنامه کار نمی‌کند و پیام می‌دهد که امکان دسترسی به این سرویس از خارج از دومین آن میسر نیست. برای اینکه این مجوز را صادر کنیم نیاز است تنظیمات مسیریابی پروژه هاب را به نحو ذیل ویرایش نمائیم:

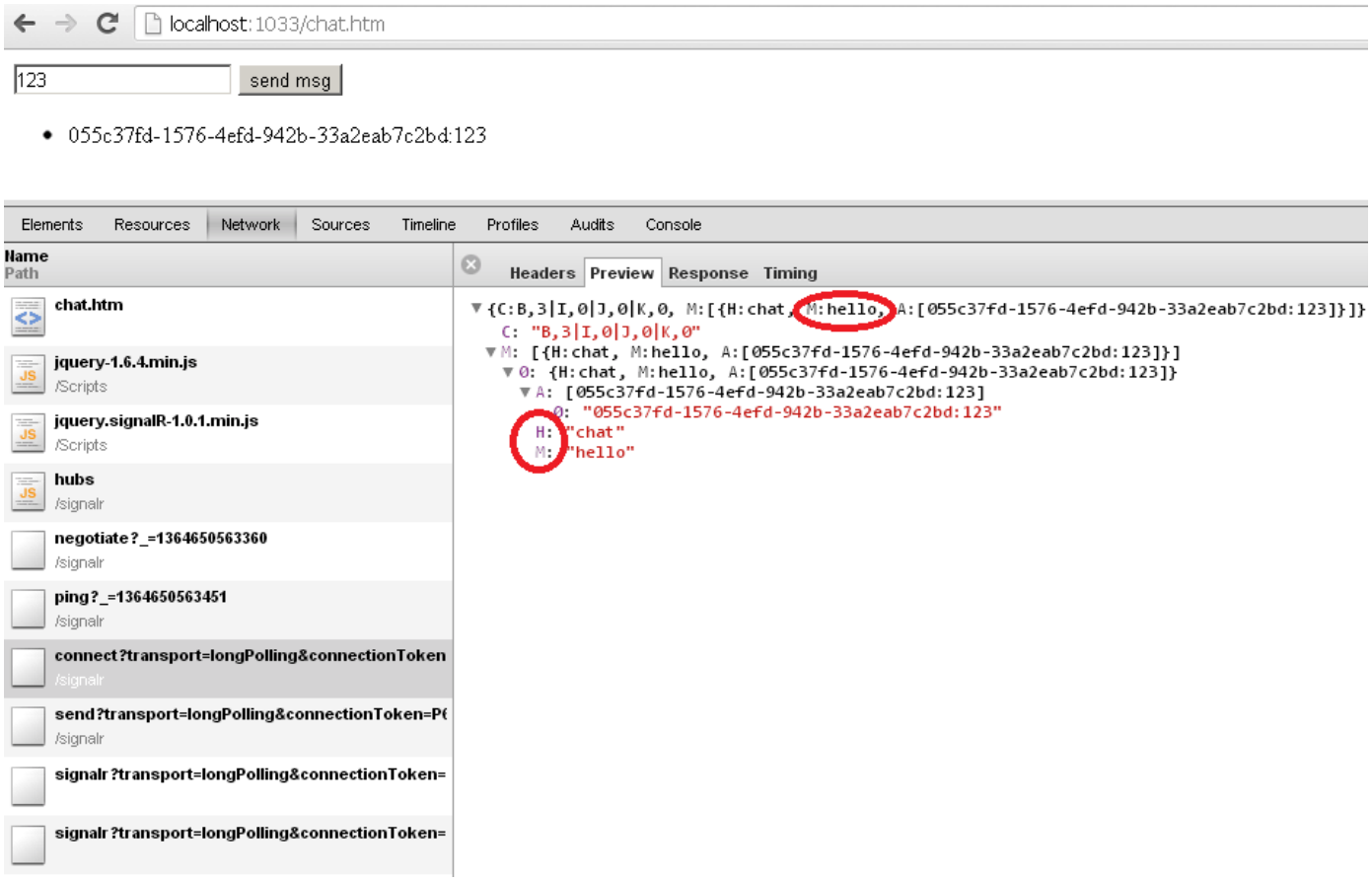
```
using System;
using System.Web;
using System.Web.Routing;
using Microsoft.AspNet.SignalR;

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

با تنظیم EnableCrossDomain به true اینبار فازهای آغاز ارتباط با سرور برقرار می‌شوند:

```
SignalR: Auto detected cross domain url. jquery.signalR-1.0.1.min.js:10
SignalR: Negotiating with 'http://localhost:1072/signalr/negotiate'. jquery.signalR-1.0.1.min.js:10
SignalR: SignalR: Initializing long polling connection with server. jquery.signalR-1.0.1.min.js:10
SignalR: Attempting to connect to
'http://localhost:1072/signalr/connect?transport=longPolling&connectionToken…NRh72omzsPkKqhKw2&connecti
onData=%5B%7B%22name%22%3A%22chat%22%7D%5D&tid=3' using longPolling. jquery.signalR-1.0.1.min.js:10
SignalR: Longpolling connected jquery.signalR-1.0.1.min.js:10
```

مطابق این لاگ‌ها ابتدا فاز negotiation انجام می‌شود. سپس حالت long polling را به صورت خودکار انتخاب می‌کند.



در برگه شبکه، مطابق شکل فوق، امکان آنالیز اطلاعات رد و بدل شده مهیا است. برای مثال در حالتیکه سرور پیام دریافتی را به کلیه کلاینت‌ها ارسال می‌کند، نام متد و نام هاب و سایر پارامترها در اطلاعات به فرمت JSON آن به خوبی قابل مشاهده هستند.

یک نکته:

اگر از ویندوز 8 (یعنی IIS8) و VS 2012 استفاده می‌کنید، برای استفاده از حالت Web socket، ابتدا فایل وب کانفیگ برنامه را باز کرده و در قسمت httpRunTime، مقدار ویژگی targetFramework را بر روی 4.5 تنظیم کنید. اینبار اگر مراحل negotiation را بررسی کنید در همان مرحله اول برقراری اتصال، از روش Web socket استفاده گردیده است.

تمرین 1

به پروژه ساده و ابتدایی فوق یک تکست باکس دیگر به نام Room را اضافه کنید؛ به همراه دکمه join. سپس نکات قسمت قبل را در مورد الحاق به یک گروه و سپس ارسال پیام به اعضای گروه را پیاده سازی نمائید. (تمام نکات آن با مطلب فوق پوشش داده شده است و در اینجا باید صرفا فراخوانی متدهای عمومی دیگری در سمت هاب، صورت گیرد)

تمرین 2

در انتهای قسمت دوم به نحوه ارسال پیام از یک هاب به هابی دیگر اشاره شد. این MonitorHub را ایجاد کرده و همچنین یک کلاینت جاوا اسکریپتی را نیز برای آن تهیه کنید تا بتوان اتصال و قطع اتصال کلیه کاربران سیستم را مانیتور و مشاهده کرد.

پیاده سازی کلاینت jQuery بدون استفاده از کلاس Proxy

در مثال قبل، از پروکسی پویای مهیای در آدرس signalr/hubs استفاده کردیم. در اینجا قصد داریم، بدون استفاده از آن نیز کار برپایی کلاینت را بررسی کنیم.

بنابراین یک فایل جدید html را مثلا به نام chat_np.html به پروژه دوم برنامه اضافه کنید. سپس محتویات آن‌را به نحو زیر تغییر دهید:

```
<!DOCTYPE>
<html>
<head>
    <title></title>
    <script src="Scripts/jquery-1.6.4.min.js" type="text/javascript"></script>
    <script src="Scripts/jquery.signalR-1.0.1.min.js" type="text/javascript"></script>
</head>
<body>
    <div>
        <input id="txtMsg" type="text" /><input id="send" type="button" value="send msg" />
        <ul id="messages">
        </ul>
    </div>
    <script type="text/javascript">
        $(function () {
            $.connection.hub.logging = true; //اطلاعات بیشتری را در جاوا اسکریپت کنسول مرورگر لاگ
می‌کند

            var connection = $.hubConnection();
            connection.url = 'http://localhost:1072/signalr'; //چون در یک پروژه دیگر قرار داریم
            var proxy = connection.createHubProxy('chat');

            proxy.on('hello', function (message) {
                //متدی که در اینجا تعریف شده دقیقا مطابق نام متد پویایی است که در هاب تعریف شده است
                //به این ترتیب سرور می‌تواند کلاینت را فراخوانی کند
                $("#messages").append("<li>" + message + "</li>");
            });

            $("#send").click(function () {
                // Hub's `SendMessage` should be camel case here
                proxy.invoke('sendMessage', $("#txtMsg").val());
            });

            connection.start();
        });
    </script>
</body>
</html>
```

در اینجا سطر مرتبط با تعریف مسیر اسکریپت‌های پویای signalr/hubs را دیگر در ابتدای فایل مشاهده نمی‌کنید. کار تشکیل proxy اینبار از طریق کدنویسی صورت گرفته است. پس از ایجاد پروکسی، برای گوش فرا دادن به متدهای فراخوانی شده از طرف سرور از متد proxy.on و نام متد فراخوانی شده سمت سرور استفاده می‌کنیم و یا برای ارسال اطلاعات به سرور از متد proxy.invoke به همراه نام متد سمت سرور استفاده خواهد شد.

کلاینت‌های دات نتی SignalR

تا کنون Solution ما حاوی یک پروژه Hub و یک پروژه وب کلاینت جی‌کوئری است. به همین Solution، یک پروژه کلاینت کنسول ویندوزی را نیز اضافه کنید.

سپس در خط فرمان پاور شل نوگت دستور زیر را صادر نمائید تا فایل‌های مورد نیاز به پروژه کنسول اضافه شوند:

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

در اینجا نیز باید دقت داشت تا دستور بر روی default project صحیحی اجرا شود (حالت پیش فرض، اولین پروژه موجود در solution است).

پس از نصب آن اگر به پوشه packages مراجعه کنید، نگارش‌های مختلف آن‌را مخصوص سیلورلایت، دات نت‌های 4 و 4.5، WinRT و ویندوز فون8 نیز می‌توانید در پوشه Microsoft.AspNet.SignalR.Client ملاحظه نمائید. البته در ابتدای نصب، انتخاب نگارش مناسب، بر اساس نوع پروژه جاری به صورت خودکار صورت می‌گیرد.

مدل برنامه نویسی آن نیز بسیار شبیه است به حالت عدم استفاده از پروکسی در حین استفاده از jQuery که در قسمت قبل بررسی گردید و شامل این مراحل است:

1) یک وهله از شیء HubConnection را ایجاد کنید.

2) پروکسی مورد نیاز را جهت اتصال به Hub از طریق متد CreateProxy تهیه کنید.
3) رویدادگردان‌ها را همانند نمونه کدهای جاوا اسکریپتی قسمت قبل، توسط متد On تعریف کنید.
4) به کمک متد Start، اتصال را آغاز نمائید.
5) متدها را به کمک متد Invoke فراخوانی نمائید.

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

namespace SignalR02.WinClient
{
    class Program
    {
        static void Main(string[] args)
        {
            var hubConnection = new HubConnection(url: "http://localhost:1072/signalr");
            var chat = hubConnection.CreateHubProxy(hubName: "chat");

            chat.On<string>("hello", msg => {
                Console.WriteLine(msg);
            });

            hubConnection.Start().Wait();

            chat.Invoke<string>("sendMessage", "Hello!");

            Console.WriteLine("Press a key to terminate the client...");
            Console.Read();
        }
    }
}
```

نمونه‌ای از این پیاده سازی را در کدهای فوق ملاحظه می‌کنید که از لحاظ طراحی آنچنان تفاوتی با نمونه ذهنی جاوا اسکریپتی ندارد.

نکته مهم

کلیه فراخوانی‌هایی که در اینجا ملاحظه می‌کنید غیرهمزمان هستند.

به همین جهت پس از متد Start، متد Wait ذکر شده‌است تا در این برنامه ساده، پس از برقراری کامل اتصال، کار invoke صورت گیرد و یا زمانیکه callback تعریف شده توسط متد chat.On فراخوانی می‌شود نیز این فراخوانی غیرهمزمان است و خصوصا اگر نیاز است رابط کاربری برنامه را در این بین به روز کنید باید به نکات به روز رسانی رابط کاربری از طریق یک ترد دیگر دقت داشت.

نظرات خوانندگان

نویسنده: Alireza Godazchian
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۵:۲

بسیار عالی بود. متشکریم....

نویسنده: پژمان پارسائی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۳:۴۶

ممنون. آموزشهای خیلی خوبی بود
به نظرتون چطوری میشه یک ارتباط رو فقط بین دو کلاینت ایجاد کرد؟ یعنی چت فقط بین دو نفر باشه و اون دو نفر هم مشخصاتشون از روی جداول دیتابیس خونده بشه (مثلا نام کاربری و نام و نام خانوادگی اونها)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۳:۵۵

نیاز هست به قسمت قبل و طراحی Hub رجوع کنید. خواندن اطلاعات از بانک اطلاعاتی در Hub صورت خواهد گرفت. همچنین هر اتصالی که به سرور برقرار می‌شود دارای یک Context.ConnectionId منحصربفرد است. بر این اساس برای ارسال پیام‌ها به دو شخص خاص باید این ConnectionIdها مدیریت شوند و زمانیکه این Id را داشتید، برای انتقال پیام به او فقط کافی است در سمت هاب متد زیر را فراخوانی کنید:

```
Clients.Client(SomeId).hello(msg)
```

نویسنده: پژمان پارسائی
تاریخ: ۱۳۹۲/۰۲/۱۱ ۰:۲۱

ConnectionIdها کجا نگهداری می‌شوند؟ مثلا برای یک کاربر خاص قصد داریم تا ConnectionId او را به دست بیاوریم. مثلا بر اساس user name او

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۲/۱۱ ۰:۴۷

Context.ConnectionId رو مثلا چیزی شبیه به سشن آی دی یک کاربر در ASP.NET درنظر بگیرید. دقیقا همان لحظه که به سرور و هاب متصل می‌شود، یک Context.ConnectionId منحصربفرد برای او تولید می‌شود. بر این اساس می‌شود به صورت اختصاصی به یک کاربر دسترسی یافت.
حالا در سمت کلاینت در این مثال بحث جاری پیغام سلام ارسال شده (برای توضیح مفاهیم). کاربر و کلاینت می‌تونه نام کاربری و کلمه عبور را در ابتدا به هاب ارسال کند. سپس بر این اساس سرور او را معتبر شمرده و Context.ConnectionId او را مورد پذیرش و پردازش قرار خواهد داد (یا خیر). بجای chat.server.sendMessage در مثال جاری مثلا یک chat.server.login را طراحی کنید. این متدی از Hub است که توسط کلاینت فراخوانی می‌شود. در اینجا پس از موفقیت آمیز بودن لاگین، ConnectionId او را معتبر شمرده و استفاده کنید.

نویسنده: علی نایینی
تاریخ: ۱۳۹۳/۰۲/۲۱ ۱۴:۲۳

اگه بخوایم از تو چند تا page به یه هاب وصل بشیم ، connectionIdها مدام عوض میشه. چه راه حلی برای این موضوع وجود داره؟
مثلا شما فرض کنید میخواهید در حین ورود به سیستم لیست یوزرها رو بگیرید ($.connection.hub) و یه جایی از برنامه میخواهید ورود به چت روم داشته باشید (($.connection.hub) و به این صورت آیدیها برای یه یوزر دو تا آیدی بوجود میاد ، راه

حل شما برای این مسئله چیه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۱ ۱۴:۲۵

« مدیریت نگاشت ConnectionIdها در SignalR به کاربران واقعی سیستم »

عنوان: **نگاهی به گزینه‌های مختلف مهیای جهت میزبانی SignalR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۲:۴۸
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, jQuery, SignalR

حداقل چهار گزینه برای Hosting سرویس‌های Hub برنامه‌های مبتنی بر SignalR وجود دارند که تا به اینجا، مورد دوم آن بیشتر بررسی گردید:

1) OWIN
2) ASP.NET Hosting
3) Self Hosting
4) Cloud و ویندوز Azure

1) OWIN

اگر به اسمبلی‌های همراه با SignalR دقت کنید، یکی از آن‌ها Microsoft.AspNet.SignalR.Owin.dll نام دارد. OWIN مخفف Open web server interface for .NET است و کار آن ایجاد لایه‌ای بین وب سرورها و برنامه‌های وب می‌باشد. یکی از اهداف مهم آن ترغیب دنیای سورس باز به تهیه ماژول‌های مختلف قابل استفاده در وب سرورهای دات نتی است. نکته‌ی مهمی که در SignalR و کلیه میزبان‌های آن وجود دارد، بنا شدن تمامی آن‌ها برفراز OWIN می‌باشد.

2) ASP.NET Hosting

بدون شک، میزبانی ASP.NET از هاب‌های SignalR، مرسوم‌ترین روش استفاده از این فناوری می‌باشد. این نوع میزبانی نیز برفراز OWIN بنا شده است. نصب آن توسط اجرای دستور پاور شل ذیل در یک پروژه وب صورت می‌گیرد:

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

3) خود میزبانی یا Self hosting

خود میزبانی نیز برفراز OWIN تهیه شده است و برای پیاده سازی آن نیاز است وابستگی‌های مرتبط با آن، از طریق NuGet به کمک فرامین پاور شل ذیل دریافت شوند:

```
PM> Install-Package Microsoft.AspNet.SignalR.Owin
PM> Install-Package Microsoft.Owin.Hosting -Pre
PM> Install-Package Microsoft.Owin.Host.HttpListener -Pre
```

مواردی که با پارامتر pre مشخص شده‌اند، در زمان نگارش این مطلب هنوز در مرحله بتا قرار دارند اما برای دموی برنامه کفایت می‌کنند.
مراحل تهیه یک برنامه ثالث (برای مثال خارج از IIS یا یک وب سرور آزمایشی) به عنوان میزبان Hubs مورد نیاز به این شرح هستند:
الف) کلاس آغازین میزبان باید با پیاده سازی اینترفیسی به نام IAppBuilder تهیه شود.
ب) مسیریابی‌های مورد نیاز تعریف گردند.
ج) وب سرور HTTP یا HTTPS توکار برای سرویس دهی آغاز گردد.

باید توجه داشت که در این حالت برخلاف روش ASP.NET Hosting، سایر اسمبلی‌های برنامه جهت یافتن Hubهای تعریف شده، اسکن نمی‌شوند. همچنین هنگام کار با jQuery مباحث عنوان شده در مورد تنظیم دسترسی‌های Cross domain نیز باید در اینجا اعمال گردند. به علاوه اجرای وب سرور توکار آن به دلایل امنیتی، نیاز به دسترسی مدیریتی دارد.

برای پیاده سازی یک نمونه، به برنامه‌ای که تاکنون تهیه کرده‌ایم، یک پروژه کنسول دیگر را به نام ConsoleHost اضافه کنید. البته باید درنظر داشت در دنیای واقعی این نوع برنامه‌ها را عموما از نوع سرویس‌های ویندوز NT تهیه می‌کنند.
در ادامه سه فرمان پاور شل یاد شده را برای افزودن وابستگی‌های مورد نیاز فراخوانی نمائید. همچنین باید دقت داشت که این

دستور بر روی پروژه جدید اضافه شده باید اجرا گردد.

```
using System;
using Microsoft.AspNet.SignalR;
using Microsoft.AspNet.SignalR.Hubs;
using Microsoft.Owin.Hosting;
using Owin;

namespace SignalR02.ConsoleHost
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapHubs(new HubConfiguration { EnableCrossDomain = true });
        }
    }

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

    class Program
    {
        static void Main(string[] args)
        {
            using (WebApplication.Start<Startup>("http://localhost:1073/"))
            {
                Console.WriteLine("Press a key to terminate the server...");
                Console.Read();
            }
        }
    }
}
```

سپس یک کلاس Startup را با امضایی که مشاهده می‌کنید تهیه نمائید. در اینجا مسیریابی و تنظیمات دسترسی از سایر دومین‌ها مشخص شده‌اند. در ادامه یک Hub نمونه، تعریف و نهایتا توسط WebApplication.Start، این وب سرور راه اندازی می‌شود.
اکنون اگر برنامه را اجرا کرده و به مسیر http://localhost:1073/signalr/hubs مراجعه کنید، فایل پروکسی تعاریف متادیتای مرتبط با سرور قابل مشاهده خواهد بود.
سمت کلاینت استفاده از آن هیچ تفاوتی نمی‌کند و با جزئیات آن پیشتر آشنا شده‌اید؛ برای مثال در کلاینت جی‌کوئری خاصیت connection.hub.url باید به مسیر جدید سرور هاب تنظیم گردد تا اتصالات به درستی برقرار شوند.

دریافت پروژه کامل مرتبط با این 4 قسمت (البته بدون فایل‌های باینری آن، جهت کاهش حجم 32 مگابایتی)
SignalRSamples.zip

عنوان: **عیب یابی و دیباگ برنامه‌های SignalR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۲ ۱۳:۴
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, jQuery, SignalR

1) برنامه SignalR در IE کار نمی‌کند.
پردازشگر json، در نگارش‌های اخیر IE به آن اضافه شده است. برای رفع این مشکل در نگارش‌های قدیمی، نیاز است از اسکریپت کمکی http://nuget.org/List/Packages/json2 استفاده نمائید. همچنین مرورگر IE را نیز باید وادار ساخت تا بر اساس آخرین موتور پردازشی خود کار کند:

```
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
```

2) هنگام فراخوانی مسیر signalr/hubs پیغام 404 (یافت نشد) دریافت می‌شود.
برای رفع این مشکل ابتدا اطمینان حاصل کنید که تنظیمات مسیریابی تعریف شده در فایل global.asax.cs موجود هستند.
در ادامه اطمینان حاصل نمائید مسیر اسکریپت‌های signalr/hubs به درستی تعریف شده‌اند:

```
<script type="text/javascript" src="/signalr/hubs"></script>
```

برای مثال در برنامه‌های MVC و وب فرم‌ها تعریف صحیح باید به شکل زیر باشد:

```
MVC:
<script type="text/javascript" src="@Url.Content("~/signalr/hubs")"></script>

Web forms:
<script type="text/javascript" src='<%= ResolveClientUrl("~/signalr/hubs") %>'></script>
```

همچنین وجود تنظیمات ذیل را در فایل وب کانفیگ برنامه نیز بررسی کنید:

```
<configuration>
   <system.webServer>
        <modules runAllManagedModulesForAllRequests="true">
        </modules>
    </system.webServer>
</configuration>
```

3) متدهای سمت کلاینت من فراخوانی نمی‌شوند.
بهترین راه برای مشاهده ریز جرئیات خطاها، ذکر سطر ذیل در کدهای سمت کلاینت جاوا اسکریپتی برنامه است:

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

و سپس مراجعه به کنسول جاوا اسکریپت مرورگر برای بررسی خطاهای لاگ شده.

4) خطای «Connection must be started before data can be sent» را دریافت می‌کنم.
همانطور که در قسمت قبل عنوان شد، کلیه فراخوانی‌های SignalR از نوع غیرهمزمان هستند. بنابراین باید با استفاده از callback و زمان فراخوانی آن‌ها که عموما پس از برقراری اتصال رخ می‌دهد، نسبت به انجام امور دلخواه اقدام کرد.

```
var connection = $.hubConnection('http://localhost:8081/');
               proxy = connection.createProxy('collectionhub')
               connection.start()
                    .done(function () {
                        proxy.invoke('subscribe', 'Product');
                        $('#messages').append('<li>invoked subscribe</li>');
```

```
                    })
                    .fail(function () { alert("Could not Connect!"); });
```

همانطور که در این مثال مشاهده می‌کنید، سطر proxy.invoke در یک callback فراخوانی شده است و نه بلافاصله در سطری پس از connection.start. هر زمان که اتصال به نحو موفقیت آمیزی برقرار شد، آنگاه متد subscribe در سمت سرور فراخوانی می‌گردد.
در حالت استفاده بدون پروکسی نیز چنین callbackهایی قابل تعریف هستند:

```
$.connection.hub.start()
                .done(function() {
                    myHub.server.SomeFunction(SomeParam) //e.g. a login or init
                         .done(connectionReady);
                })
                .fail(function() {
                    alert("Could not Connect!");
                 });
```

5) بعد از 10 اتصال به IIS، برنامه متوقف می‌شود.
این مورد، محدودیت ذاتی IIS 7 نصب شده بر روی ویندوز 7 است. بهتر است از یک IIS کامل موجود در ویندوزهای سرور استفاده کنید. در این سرورها عدد پیش فرض تنظیم شده 5000 اتصال است که در صورت نیاز با استفاده از دستور زیر قابل تغییر است:

```
appcmd.exe set config /section:system.webserver/serverRuntime /appConcurrentRequestLimit:100000
```

به علاوه ASP.NET نیز محدودیت 5000 اتصال به ازای هر CPU را دارد. برای تغییر آن باید به مسیر ذیل مراجعه

```
%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet.config
```

و سپس مقدار maxConcurrentRequestsPerCPU را تنظیم کرد:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
    <runtime>
        <legacyUnhandledExceptionPolicy enabled="false" />
        <legacyImpersonationPolicy enabled="true"/>
        <alwaysFlowImpersonationPolicy enabled="false"/>
        <SymbolReadingPolicy enabled="1" />
        <shadowCopyVerifyByTimestamp enabled="true"/>
    </runtime>
    <startup useLegacyV2RuntimeActivationPolicy="true" />
    <system.web>
        <applicationPool maxConcurrentRequestsPerCPU="20000" />
    </system.web>
</configuration>
```

به علاوه ASP.NET پس از رد شدن از حد maxConcurrentRequestsPerCPU، درخواست‌ها را در صف قرار می‌دهد. این مورد نیز قابل تنظیم است. ابتدا به مسیر ذیل مراجعه کرده

```
%windir%\Microsoft.NET\Framework\v4.0.30319\Config\machine.config
```

و سپس در صورت نیاز و لزوم، مقدار requestQueueLimit را تغییر دهید:

```
<processModel autoConfig="false" requestQueueLimit="250000" />
```

نظرات خوانندگان

نویسنده: ناصر طاهری
تاریخ: ۱۳۹۲/۰۱/۱۳ ۲۰:۲۹

بسیار عالی بود. تشکر

نویسنده: samin
تاریخ: ۱۳۹۲/۰۱/۱۴ ۱۰:۲۹

از زحمات بی دریغ شما سپاسگذارم. پاینده باشید

نویسنده: میثم ق
تاریخ: ۱۳۹۲/۰۲/۱۰ ۱۶:۴۸

سلام

خیلی خوب بود.دستت درد نکنه.

عنوان: **تزریق خودکار وابستگی‌ها در SignalR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۱/۱۵ ۱۶:۷
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, jQuery, SignalR

فرض کنید لایه سرویس برنامه دارای اینترفیس و کلاس‌های زیر است:

```
namespace SignalR02.Services
{
    public interface ITestService
    {
        int GetRecordsCount();
    }
}
```

```
namespace SignalR02.Services
{
    public class TestService : ITestService
    {
        public int GetRecordsCount()
        {
            return 10; // It's just a sample to test IOC's.
        }
    }
}
```

قصد داریم از این لایه، توسط تزریق وابستگی‌ها در Hub برنامه استفاده کنیم:

```
[HubName("chat")]
    public class ChatHub : Hub
    {
        //جهت آزمایش تزریق خودکار وابستگی‌ها
        private readonly ITestService _testService;
        public ChatHub(ITestService testService)
        {
            _testService = testService;
        }

        public void SendMessage(string message)
        {
            var msg = string.Format("{0}:{1}", Context.ConnectionId, message);
            Clients.All.hello(msg);

            Clients.All.hello(string.Format("RecordsCount: {0}", _testService.GetRecordsCount()));
```

برنامه، همان برنامه‌ای است که در دوره جاری تکمیل گردیده است. فقط در اینجا سازنده کلاس اضافه شده و سپس اینترفیس ITestService به عنوان پارامتر آن تعریف گردیده است. در ادامه می‌خواهیم کار وهله سازی و تزریق نمونه مرتبط را توسط StructureMap به صورت خودکار انجام دهیم.

برای این منظور یک کلاس جدید را به نام StructureMapDependencyResolver به برنامه اضافه کنید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNet.SignalR;
using StructureMap;

namespace SignalR02.Utils
{
    public class StructureMapDependencyResolver : DefaultDependencyResolver
    {
        private readonly IContainer _container;
        public StructureMapDependencyResolver(IContainer container)
        {
            if (container == null)
            {
                throw new ArgumentNullException("container");
```

```
            }
            _container = container;
        }
        public override object GetService(Type serviceType)
        {
            return !serviceType.IsAbstract && !serviceType.IsInterface && serviceType.IsClass
                                ? _container.GetInstance(serviceType)
                                : (_container.TryGetInstance(serviceType) ??
base.GetService(serviceType));
        }
        public override IEnumerable<object> GetServices(Type serviceType)
        {
            return
_container.GetAllInstances(serviceType).Cast<object>().Concat(base.GetServices(serviceType));
        }
    }
}
```

کار این کلاس، تعویض DefaultDependencyResolver توکار SignalR با StructureMap است. از این جهت که برای مثال در سراسر برنامه از StructureMap جهت تزریق وابستگی‌ها استفاده شده است و قصد داریم در قسمت Hub آن نیز یکپارچگی کار حفظ گردد.

برای استفاده از این کلاس تعریف شده فقط کافی است Application_Start فایل Global.asax.cs برنامه هاب را به نحو ذیل تغییر دهیم:

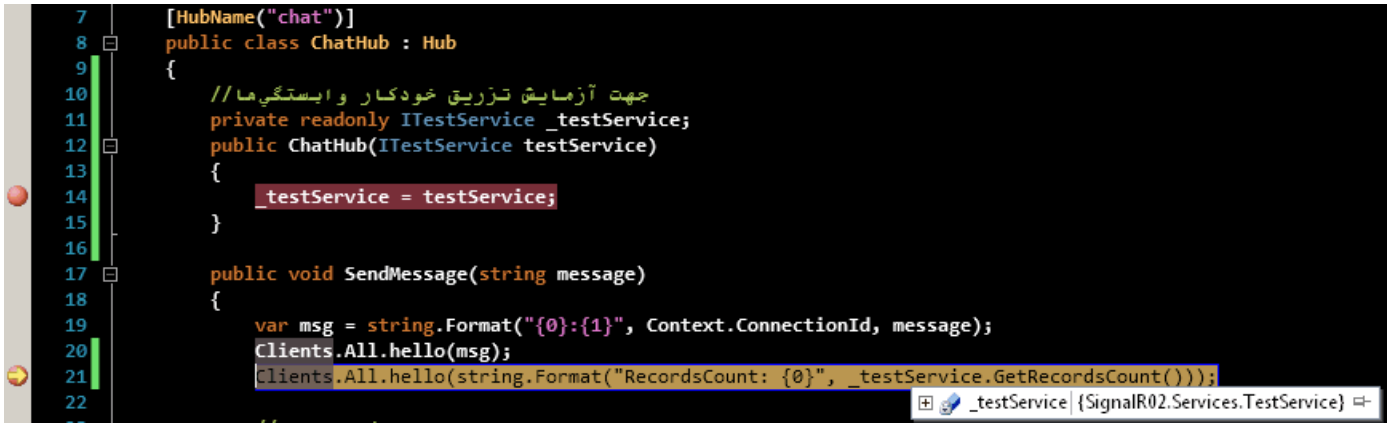
```
using System;
using System.Web;
using System.Web.Routing;
using Microsoft.AspNet.SignalR;
using SignalR02.Services;
using SignalR02.Utils;
using StructureMap;

namespace SignalR02
{
    public class Global : HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            ObjectFactory.Initialize(cfg =>
            {
                cfg.For<IDependencyResolver>().Singleton().Add<StructureMapDependencyResolver>();
                // the rest ...
                cfg.For<ITestService>().Use<TestService>();
            });
            GlobalHost.DependencyResolver = ObjectFactory.GetInstance<IDependencyResolver>();

            // Register the default hubs route: ~/signalr
            RouteTable.Routes.MapHubs(new HubConfiguration
            {
                EnableCrossDomain = true
            });
        }
    }
}
```

در اینجا در ابتدای کار IDependencyResolver توکار StructureMap با کلاس StructureMapDependencyResolver وهله سازی می‌گردد. سپس تعاریف متداول تنظیمات کلاس‌ها و اینترفیس‌های لایه سرویس برنامه اضافه می‌شوند. همچنین نیاز است GlobalHost.DependencyResolver توکار SignalR نیز به نحوی که ملاحظه می‌کنید مقدار دهی گردد.

اینبار اگر برنامه را اجرا کنید و سپس یکی از کلاینت‌های آن‌را فراخوانی نمائید، می‌توان مشاهده کرد که کار وهله سازی و تزریق وابستگی سرویس مورد استفاده به صورت خودکار انجام گردیده است:


[HubName("chat")]
public class ChatHub : Hub
{
//جهت آزمایش تزریق خودکار وابستگی‌ها
private readonly ITestService _testService;
public ChatHub(ITestService testService)
{
_testService = testService;
}
public void SendMessage(string message)
{
var msg = string.Format("{0}:{1}", Context.ConnectionId, message);
Clients.All.hello(msg);
Clients.All.hello(string.Format("RecordsCount: {0}", _testService.GetRecordsCount()));
_testService {SignalR02.Services.TestService}

عنوان: **مثال - نمایش درصد پیشرفت عملیات توسط SignalR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۱ ۱۵:۱۲
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, jQuery, SignalR

برنامه‌های وب در سناریوهای بسیاری نیاز دارند تا درصد پیشرفت عملیاتی را به کاربران گزارش دهند. نمونه ساده آن، گزارش درصد پیشرفت میزان دریافت یک فایل است و یا اعلام درصد انجام یک عملیات طولانی از سمت سرور به کاربر. در ادامه قصد داریم این موضوع را توسط SignalR پیاده سازی کنیم.

نکته‌ای در مورد نگارش‌های مختلف SignalR

اگر برنامه شما قرار است دات نت 4 را پشتیبانی کند، آخرین نگارش SignalR که با آن سازگار است، نگارش 1.1.3 می‌باشد.
بنابراین اگر دستور ذیل را اجرا کنید:

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

SignalR 2 را نصب می‌کند که با دات نت 4 و نیم به بعد سازگار است.
اگر دستور ذیل را اجرا کنید، Siganlr 1.x را نصب می‌کند که با دات نت 4 به بعد سازگار است:

```
PM> Install-Package Microsoft.AspNet.SignalR -Version 1.1.3
```

پیش فرض این مطلب نیز استفاده از نگارش 1.1.3 می‌باشد تا بازه بیشتری از وب سرورها را شامل شود.
با اینکار Microsoft.AspNet.SignalR.JS نیز به صورت خودکار نصب می‌گردد و به این ترتیب کلاینت جاوا اسکریپتی Siganlr نیز در برنامه قابل استفاده خواهد بود.

تنظیمات فایل Global.asax.cs

سطر فراخوانی متد RouteTable.Routes.MapHubs باید در ابتدای متد Application_Start فایل Global.asax.cs قرار گیرد (پیش از هر تنظیم دیگری). تفاوتی هم نمی‌کند که برنامه وب فرم است یا MVC. به این ترتیب مسیریابی‌های SignalR تنظیم شده و مسیر http://localhost/signalr/hubs قابل استفاده خواهد بود.

تنظیمات اسکریپت‌های سمت کلاینت مورد نیاز

پس از نصب بسته SignalR، سه اسکریپت ذیل باید به ابتدای صفحه وب اضافه شوند تا کلاینت‌های جاوا اسکریپتی SignalR بتوانند با سرور ارتباط برقرار کنند:

```
<script src="Scripts/jquery-1.6.4.min.js" type="text/javascript"></script>
<script src="Scripts/jquery.signalR-1.1.3.min.js" type="text/javascript"></script>
<script src="signalr/hubs" type="text/javascript"></script>
```

این تنظیمات نیز برای هر دو نوع برنامه‌های وب فرم و MVC یکسان است.

تعریف کلاس Hub برنامه

```
using Microsoft.AspNet.SignalR;

namespace WebFormsSample03.Common
{
    public class ProgressHub : Hub
    {
        /// <summary>
```

```
        /// این متد استاتیک تعریف شده تا در برنامه به صورت مستقیم قابل استفاده باشد
        /// یا می‌شد اصلا این متد تعریف نشود و از همان دریافت زمینه هاب در کنترلر استفاده گردد
        /// </summary>
        public static void UpdateProgressBar(int value, string connectionId)
        {
            var ctx = GlobalHost.ConnectionManager.GetHubContext<ProgressHub>();
            ctx.Clients.Client(connectionId).updateProgressBar(value); //فراخوانی یک متد در سمت کلاینت
        }
    }
}
```

متدی که در کلاس هاب برنامه تعریف شده، از نوع استاتیک است. از این جهت که می‌خواهیم این متد را در خارج از این هاب و در یک کنترلر Web API فراخوانی کنیم. زمانیکه متدی به صورت استاتیک تعریف می‌شود، ارتباط آن با وهله جاری کلاس یا this قطع خواهد شد. به همین جهت نیاز است تا از طریق متد GlobalHost.ConnectionManager.GetHubContext مجددا به context کلاس هاب دسترسی پیدا کنیم.

البته تعریف این متد در اینجا ضروری نبود. حتی می‌شد بدنه کلاس هاب را خالی تعریف کرد و متد GetHubContext را مستقیما داخل یک کنترلر فراخوانی نمود.

متد UpdateProgressBar، مقدار value را به تنها یک کلاینت که Id آن مساوی connectionId دریافتی است، ارسال می‌کند. این کلاینت باید یک callback جاوا اسکریپتی را جهت تامین متد پویای updateProgressBar تدارک ببیند.

کلاس Web API کنترلر دریافت فایل‌ها

فرقی نمی‌کند که برنامه شما از نوع وب فرم است یا MVC. امکانات Web API در هر دو نوع پروژه، قابل دسترسی است (همان ایده یک ASP.NET واحد).

بنابراین نیاز است یک کنترلر وب API جدید را به پروژه اضافه کرده و محتوای آن را به شکل ذیل تغییر دهیم:

```
using System.Threading;
using System.Web.Http;
using WebFormsSample03.Common;

namespace WebFormsSample03
{
    public class DownloadRequest
    {
        public string Url { set; get; }
        public string ConnectionId { set; get; }
    }

    public class DownloaderController : ApiController
    {
        public void Post([FromBody]DownloadRequest data)
        {
            //todo: start downloading the data.Url ....

            ProgressHub.UpdateProgressBar(10, data.ConnectionId);
            Thread.Sleep(2000);

            ProgressHub.UpdateProgressBar(40, data.ConnectionId);
            Thread.Sleep(3000);

            ProgressHub.UpdateProgressBar(64, data.ConnectionId);
            Thread.Sleep(2000);

            ProgressHub.UpdateProgressBar(77, data.ConnectionId);
            Thread.Sleep(2000);

            ProgressHub.UpdateProgressBar(92, data.ConnectionId);
            Thread.Sleep(3000);

            ProgressHub.UpdateProgressBar(99, data.ConnectionId);
            Thread.Sleep(2000);

            ProgressHub.UpdateProgressBar(100, data.ConnectionId);
        }
    }
}
```

اگر برنامه شما وب فرم است، باید تنظیمات مسیریابی ذیل را نیز به آن افزود. در برنامه‌های MVC4 این تنظیم به صورت پیش فرض وجود دارد:

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

namespace WebFormsSample03
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            // Register the default hubs route: ~/signalr
            RouteTable.Routes.MapHubs();

            RouteTable.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

کاری که در این کنترلر انجام شده، شبیه سازی یک عملیات طولانی توسط متد Thread.Sleep است. همچنین این کنترلر، id کلاینت درخواست کننده یک url را نیز دریافت می‌کند. بنابراین می‌توان به نحو بهینه‌ای، تنها نتایج پیشرفت عملیات را به این کلاینت ارسال کرد و نه به سایر کلاینت‌ها.
همچنین در اینجا با توجه به مسیریابی تعریف شده، باید اطلاعات را به آدرس api/Downloader از نوع Post ارسال کرد.

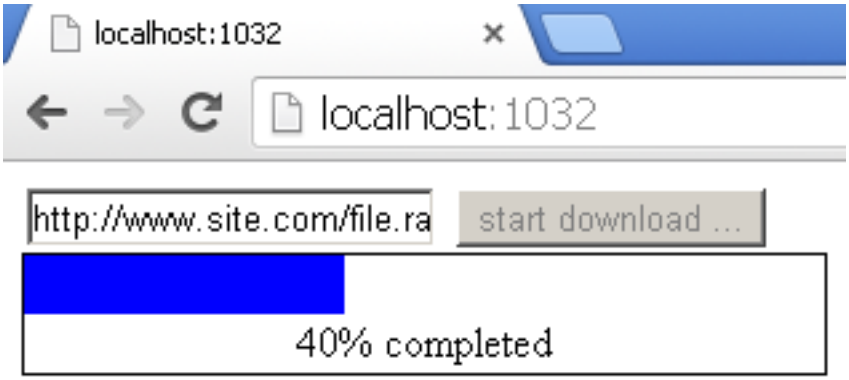
تعریف کلاینت متصل به Hub

در سمت سرور، متد پویای updateProgressBar فراخوانی شده است. اکنون باید این متد را در سمت کلاینت پیاده سازی کنیم:

```
<form id="form1" runat="server">
    <div>
    <input id="txtUrl" value="http://www.site.com/file.rar" type="text" />
        <input id="send" type="button" value="start download ..." />
        <br />
        <div id="bar" style="border: #000 1px solid; width:300px;"></div>
    </div>
    </form>
    <script type="text/javascript">
        $(function () {
            $.connection.hub.logging = true; //اطلاعات بیشتری را در جاوا اسکریپت کنسول مرورگر لاگ می‌کند
            var progressHub = $.connection.progressHub; //این نام مستعار پیشتر توسط ویژگی نام هاب تنظیم شده است
            progressHub.client.updateProgressBar = function (value) {
                //متدی که در اینجا تعریف شده دقیقا مطابق نام متد پویایی است که در هاب تعریف شده است
                //به این ترتیب سرور می‌تواند کلاینت را فراخوانی کند
                $("#bar").html(GaugeBar.generate(value));
            };
            $.connection.hub.start() // فاز اولیه ارتباط را آغاز می‌کند
            .done(function () {
                $("#send").click(function () {
                    $("#send").attr('disabled', 'disabled');
                    var myClientId = $.connection.hub.id;
                    // اکنون اتصال برقرار است به سرور
                    $.ajax({
                        type: "POST",
                        contentType: "application/json",
                        url: "/api/Downloader",
                        data: JSON.stringify({ Url: $("#txtUrl").val(), ConnectionId: myClientId })
                    }).success(function () {
                        $("#send").removeAttr('disabled');
                    }).fail(function () {
                        //
                    });
                });
            });
```

```
    });
</script>
```

بر روی این فرم، یک جعبه متنی که Url را دریافت می‌کند و یک دکمه‌ی آغاز کار دریافت این Url، وجود دارد.
در ابتدای کار صفحه، اتصال به progressHub برقرار می‌شود. اگر دقت کنید، نام این هاب با حروف کوچک در اینجا (در سمت کلاینت) آغاز می‌گردد.
سپس با تعریف یک callback به نام progressHub.client.updateProgressBar، پیام‌های دریافتی از طرف سرور را به یک افزونه progress bar جی‌کوئری، برای نمایش ارسال می‌کند.
کار اتصال به رویداد کلیک دکمه‌ی آغاز دریافت فایل، در متد done باید انجام شود. این callback زمانی فراخوانی می‌گردد که کار اتصال به سرور با موفقیت صورت گرفته باشد.
سپس در ادامه توسط jQuery Ajax، اطلاعات Url و همچنین Id کلاینت را به مسیر api/Downloader یا همان web api controller ارسال می‌کنیم.



کدهای کامل این مثال را از اینجا نیز می‌توانید دریافت نمائید:

WebFormsSample03.zip

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۰ ۱۱:۱

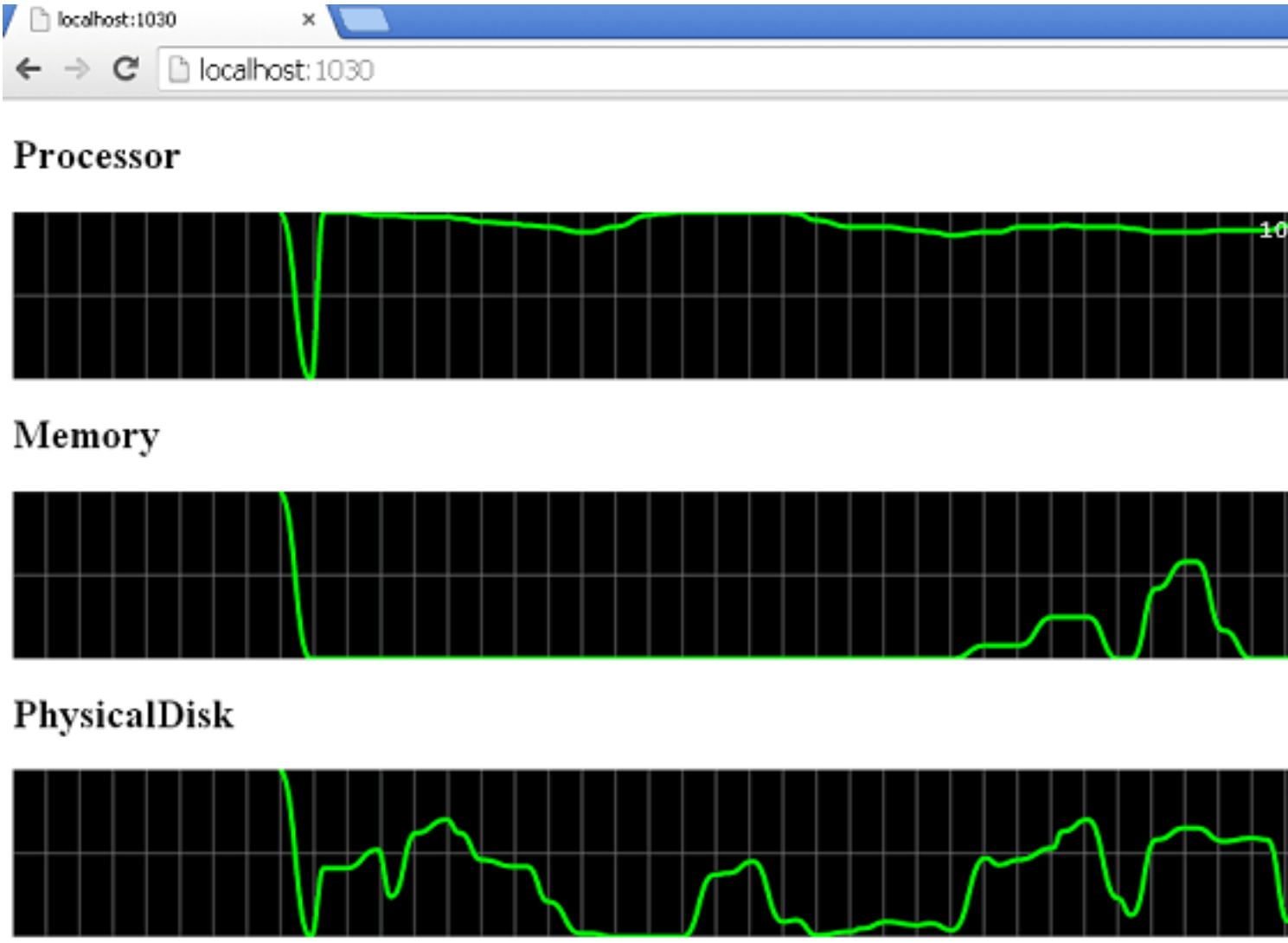
جهت اطلاع؛ همیشه آخرین نسخه x.1 مخصوص دات نت 4 را در صفحه ذیل بررسی کنید:

http://www.nuget.org/packages/Microsoft.AspNet.SignalR

برای مثال در این تاریخ Microsoft ASP.NET SignalR 1.1.4 نسخه آخر x.1 است و از لحاظ امنیتی نیاز است این به روز رسانی صورت گیرد.

عنوان: **مثال - نمایش بلادرنگ میزان مصرف CPU و حافظه سرور بر روی کلیه کلاینت‌های متصل توسط SignalR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۰:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, jQuery, SignalR

یکی از کاربردهای جالب SignalR می‌تواند به روز رسانی مداوم صفحه نمایش کاربران، توسط اطلاعات ارسالی از طرف سرور باشد. در ادامه قصد داریم به عنوان منبع داده، آمار کارآیی سرور را به کلاینت‌ها ارسال کنیم و سپس به تصویری همانند شکل ذیل برسیم:



در اینجا از Smoothie Charts برای ترسیم نمودارهای بلادرنگ سازگار با Canvas مخصوص HTML 5 استفاده شده است.

پیشنیازها

پیشنیازهای این مطلب با مطلب « مثال - نمایش درصد پیشرفت عملیات توسط SignalR » یکی است. برای مثال، نحوه دریافت وابستگی‌ها، تنظیمات فایل global.asax و افزودن اسکریپت‌ها، تفاوتی با مثال قبلی ندارند.

تهیه منبع داده اطلاعات نمایشی

```
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;

namespace SignalR04.Common
{
    public class Counter
    {
        public string Name { set; get; }
        public float Value { set; get; }
    }

    public class PerformanceCounterProvider
    {
        private readonly List<PerformanceCounter> _counters = new List<PerformanceCounter>();

        public PerformanceCounterProvider()
        {
            _counters.Add(new PerformanceCounter("Processor", "% Processor Time", "_Total", readOnly:
true));
            _counters.Add(new PerformanceCounter("Memory", "Pages/sec", readOnly: true));
            _counters.Add(new PerformanceCounter("PhysicalDisk", "% Disk Time", "_Total", readOnly:
true));
        }

        public IList<Counter> GetResults()
        {
            return _counters.Select(c => new Counter
                                    {
                                        Name = c.CategoryName,
                                        Value = c.NextValue()
                                    }).ToList();
        }
    }
}
```

کلاس PerformanceCounterProvider، سه مؤلفه کارآیی سرور را بررسی کرده و هربار توسط متد GetResults، آن‌ها را بازگشت می‌دهد. از این منبع داده، در هاب برنامه استفاده خواهیم کرد.

تهیه هاب ارسال داده‌ها به کلاینت‌ها

```
using System.Threading;
using Microsoft.AspNet.SignalR;
using ThreadTimer = System.Threading.Timer;

namespace SignalR04.Common
{
    public class PerformanceCounterHub : Hub
    {
        private ThreadTimer _threadTimer; //keep it alive
        private readonly PerformanceCounterProvider _perfService = new PerformanceCounterProvider();

        public PerformanceCounterHub()
        {
            _threadTimer = new ThreadTimer(timerCallback, null, Timeout.Infinite, 1000);
            _threadTimer.Change(dueTime: 1000, period: 2000);
        }

        private void timerCallback(object state)
        {
            var results = _perfService.GetResults();
            this.Clients.All.newCounters(results);
        }
    }
}
```

در این هاب، یک thread timer ایجاد شده است که هر دو ثانیه یکبار، اطلاعات را از PerformanceCounterProvider دریافت و سپس با فراخوانی this.Clients.All.newCounters، آن‌ها را به کلیه کلاینت‌های متصل ارسال می‌کند.
این هاب به صورت خودکار با اولین بار وهله سازی، پس از فراخوانی متد connection.hub.start در سمت کلاینت، شروع به کار

می‌کند.

کدهای سمت کلاینت نمایش نمودارها

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <script src="Scripts/jquery-1.6.4.min.js" type="text/javascript"></script>
    <script src="Scripts/jquery.signalR-1.1.3.min.js" type="text/javascript"></script>
    <script type="text/javascript" src='<%= ResolveClientUrl("~/signalr/hubs") %>'></script>
    <script src="Scripts/smoothie.js" type="text/javascript"></script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <div>
            <h2>Processor</h2>
            <canvas id="Processor" width="800" height="100"></canvas>
        </div>
        <div>
            <h2>Memory</h2>
            <canvas id="Memory" width="800" height="100"></canvas>
        </div>
        <div>
            <h2>PhysicalDisk</h2>
            <canvas id="PhysicalDisk" width="800" height="100"></canvas>
        </div>
    </div>
    </form>
    <script type="text/javascript">
        var ChartEntry = function (name) {
            var self = this;
            self.name = name;
            self.chart = new SmoothieChart({ millisPerPixel: 50, labels: { fontSize: 15} });
            self.timeSeries = new TimeSeries();
            self.chart.addTimeSeries(self.timeSeries, { lineWidth: 3, strokeStyle: "#00ff00" });
        };

        ChartEntry.prototype = {
            addValue: function (value) {
                var self = this;
                self.timeSeries.append(new Date().getTime(), value);
            },

            start: function () {
                var self = this;
                self.canvas = document.getElementById(self.name);
                self.chart.streamTo(self.canvas);
            }
        };

        $(function () {
            $.connection.hub.logging = true;
            var performanceCounterHub = $.connection.performanceCounterHub;
            var charts = [];
            performanceCounterHub.client.newCounters = function (updatedCounters) {
                $.each(updatedCounters, function (index, updateCounter) {
                    var entry;
                    $.each(charts, function (idx, chart) {
                        if (chart.name == updateCounter.Name) {
                            entry = chart;
                            return;
                        }
                    });

                    if (!entry) {
                        entry = new ChartEntry(updateCounter.Name);
                        charts.push(entry);
                        entry.start();
                    }
                    entry.addValue(updateCounter.Value);
                });
            };
            $.connection.hub.start();
        });
    </script>
</body>
```

```
</html>
```

- در ابتدا سه canvas بر روی صفحه قرار گرفته‌اند که معرف سه PerformanceCounter دریافتی از سرور هستند.
- id هر canavs به Name اطلاعات دریافتی از سرور تنظیم شده است تا نمودارها در جای صحیحی ترسیم شوند.
- سپس نحوه کپسوله سازی SmoothieChart را مشاهده می‌کنید؛ چطور می‌توان از آن یک شیء جاوا اسکریپتی ایجاد کرد و چطور اطلاعات را به آن اضافه نمود.
- نهایتا کار هاب را آغاز می‌کنیم. Callback ایی به نام performanceCounterHub.client.newCounters دقیقا متصل است به فراخوانی this.Clients.All.newCounters سمت سرور. در اینجا updatedCounters دریافتی، یک آرایه جاوا اسکریپتی است که هر عضو آن دارای Name و Value است. بر این اساس، تنها کافی است این مقادیر را که هر دو ثانیه یکبار به روز می‌شوند، به SmoothieChart برای ترسیم ارسال کنیم.

کدهای کامل این مثال را از اینجا نیز می‌توانید دریافت کنید:

SignalR04.zip

عنوان: **SignalR و خزنده‌های گوگل و سایر جستجوگرها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۳ ۱۳:۲۱
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, jQuery, SignalR

یکی از کارهایی که Googlebot و سایر خزنده‌های انواع و اقسام جستجوگرها انجام می‌دهند، سرک کشیدن به لینک‌هایی است که در صفحه قرار گرفته‌اند. اگر سایت شما از SiganlR استفاده کند، به طور قطع یک چنین تعریفی را خواهد داشت:

```
<script src="signalr/hubs" type="text/javascript"></script>
```

فراخوانی‌های نابجای این آدرس سبب خواهد شد تا علاوه بر بروز استثناهای اضافی لاگ شده در سمت سرور، اتصالاتی ماندگار نیز ایجاد شوند تا کلاینت و سرور بتوانند کار bidirectional communication را انجام دهند. برای رفع این مشکل تنها کافی است در فایل robots.txt سایت، مسیر یاد شده را ممنوعه اعلام کنید:

```
User-agent: *
Disallow: /signalr/
```

عنوان: **مثال - نمایش بلادرنگ تعداد کاربران آنلاین توسط SignalR**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱۴:۲۳
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, jQuery, SignalR

راه حل‌های زیادی برای محاسبه و نمایش تعداد کاربران آنلاین یک برنامه وب وجود دارند و عموما مبتنی بر کار با متغیرهای سشن یا Application و امثال آن هستند. این روش‌ها عموما دقیق نبوده و خصوصا قسمت قطع اتصال کاربر را نمی‌توانند دقیقا تشخیص دهند. به همین جهت نیاز به یک تایمر دارند که مثلا اگر در 5 دقیقه قبل، کاربری درخواست مشاهده آدرسی را به سرور ارسال نکرده بود، از لیست کاربران آنلاین حذف شود.
در ادامه بجای این روش‌ها، از SignalR برای محاسبه تعداد کاربران آنلاین و همچنین به روز رسانی بلادرنگ این عدد در سمت کاربر، استفاده خواهیم کرد.

تشخیص اتصال و قطع اتصال کاربران در SignalR

زیر ساخت‌های کلاس Hub موجود در SignalR، دارای متدهای ردیابی اتصال (OnConnected)، قطع اتصال (OnDisconnected) و یا برقراری مجدد اتصال کاربران (OnReconnected) هستند. با بازنویسی این متدها می‌توان به تخمین بسیار دقیقی از تعداد کاربران آنلاین یک سایت رسید.

پیشنیازهای بحث

پیشنیازهای این بحث با مطلب « مثال - نمایش درصد پیشرفت عملیات توسط SignalR » یکی است. برای مثال نحوه دریافت وابستگی‌ها، تنظیمات فایل global.asax و افزودن اسکریپت‌ها، تفاوتی با مثال یاد شده ندارند.

تعریف هاب کاربران آنلاین برنامه

```
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNet.SignalR;

namespace SignalR05.Common
{
    public class OnlineUsersHub : Hub
    {
        public static readonly ConcurrentDictionary<string, string> OnlineUsers = new
ConcurrentDictionary<string, string>();

        public void UpdateUsersOnlineCount()
        {
            // آی پی معرف یک کاربر است
            // اما کانکشن آی دی معرف یک برگه جدید در مرورگر او است
            // هر کاربر می‌تواند چندین برگه را به یک سایت گشوده یا ببندد
            var ipsCount = OnlineUsers.Select(x => x.Value).Distinct().Count();
            this.Clients.All.updateUsersOnlineCount(ipsCount);
        }

        /// <summary>
        /// اگر کاربران اعتبار سنجی شده‌اند بهتر است از
        /// this.Context.User.Identity.Name
        /// بجای آی پی استفاده شود
        /// </summary>
        protected string GetUserIpAddress()
        {
            object environment;
            if (!Context.Request.Items.TryGetValue("owin.environment", out environment))
                return null;

            object serverRemoteIpAddress;
            if (!((IDictionary<string, object>)environment).TryGetValue("server.RemoteIpAddress", out
serverRemoteIpAddress))
                return null;
```

```
            return serverRemoteIpAddress.ToString();
        }

        public override Task OnConnected()
        {
            var ip = GetUserIpAddress();
            OnlineUsers.TryAdd(this.Context.ConnectionId, ip);
            UpdateUsersOnlineCount();

            return base.OnConnected();
        }

        public override Task OnReconnected()
        {
            var ip = GetUserIpAddress();
            OnlineUsers.TryAdd(this.Context.ConnectionId, ip);
            UpdateUsersOnlineCount();

            return base.OnReconnected();
        }

        public override Task OnDisconnected()
        {
            // در این حالت ممکن است مرورگر کاملا بسته شده باشد
            // یا حتی صرفا یک برگه مرورگر از چندین برگه متصل به سایت بسته شده باشند
            string ip;
            OnlineUsers.TryRemove(this.Context.ConnectionId, out ip);
            UpdateUsersOnlineCount();

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

کدهای کامل هاب شمارش کاربران آنلاین را در اینجا ملاحظه می‌کنید؛ به همراه نکته‌ی نحوه‌ی دریافت IP کاربر متصل شده به سایت، در یک هاب. کار افزودن یا حذف این کاربران به ConcurrentDictionary تعریف شده، در روال‌های بازنویسی شده اتصال، قطع اتصال و اتصال مجدد یک کاربر، انجام شده است.

در اینجا، هم به IP کاربر و هم به ConnectionId او نیاز است. از این جهت که هر ConnectionId، معرف یک برگه جدید باز شده در مرورگر کاربر است. اگر صرفا IPها را پردازش کنیم، با بسته شدن یکی از چندین برگه مرورگر او که اکنون به سایت متصل هستند، آمار او را از دست خواهیم داد. این کاربر هنوز چندین برگه باز دیگر را دارد که با سایت در ارتباط هستند، اما چون IP او را از لیست حذف کرده‌ایم (در نتیجه بسته شدن یکی از برگه‌ها)، آمار کلی شخص را نیز از دست خواهیم داد. بنابراین هر دوی IP و ConnectionIdها باید پردازش شوند.

اگر برنامه شما دارای اعتبارسنجی است (یک صفحه لاگین دارد)، بهتر است بجای IP از this.Context.User.Identity.Name استفاده کنید.

کدهای سمت کلاینت نمایش آمار کاربران

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <script src="Scripts/jquery-1.6.4.min.js" type="text/javascript"></script>
    <script src="Scripts/jquery.signalR-1.1.3.min.js" type="text/javascript"></script>
    <script type="text/javascript" src='<%= ResolveClientUrl("~/signalr/hubs") %>'></script>
</head>
<body>
    <form id="form1" runat="server">
    online users count: <span id="usersCount"></span>
    </form>
    <script type="text/javascript">
        $(function () {
            $.connection.hub.logging = true;
            var onlineUsersHub = $.connection.onlineUsersHub;
            onlineUsersHub.client.updateUsersOnlineCount = function (count) {
                $('#usersCount').text(count);
            };
            $.connection.hub.start();
        });
    </script>
```

```
</body>
</html>
```

با توجه به اینکه در هاب تعریف شده، متد پویای updateUsersOnlineCount، آمار تعداد کاربران متصل را (تعداد آی پی‌های منحصربفرد متصل را) به کلاینت‌ها ارسال می‌کند، بنابراین در سمت کلاینت نیز با تعریف callback ایی به همین نام، می‌توان این آمار دریافتی را به کاربران سایت نمایش داد. آماری که به صورت خودکار با کم و زیاد شدن کاربران به روز شده و نیازی نیست کاربر به صورت دستی، صفحه را به روز کند.

کدهای کامل این مثال را از اینجا نیز می‌توانید دریافت کنید:

SignalR05.zip

عنوان: **مدیریت نگاشت ConnectionIdها در SignalR به کاربران واقعی سیستم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۸ ۱۱:۳۴
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, jQuery, SignalR

SignalR تنها از Context.ConnectionId خود با خبر است و بس. کاربران واقعی سیستم، پس از اعتبارسنجی می‌توانند با چندین و چند ConnectionId به سیستم متصل شوند؛ برای مثال گشودن چندین مرورگر یا باز کردن برگه‌های مختلف یک مرورگر و یا حتی استفاده از سایر کلاینت‌هایی که SignalR قابلیت کار کردن با آن‌ها را دارد. بنابراین باید بتوان بین ConnectionIdها و کاربران واقعی سیستم، تناظری را برقرار کرد و همچنین نباید تصور کرد که الزاما یک کاربر مساوی است با یک ConnectionId.

اعتبار سنجی کاربران در SignalR

تمام مباحث عنوان شده در مورد نحوه‌ی کار با Forms Authentication استاندارد یک برنامه وب، در SignalR نیز قابل دسترسی است. پس از اینکه کاربری به سایت وارد شد (با استفاده از روش‌های متداول؛ مانند یک صفحه‌ی لاگین)، اطلاعات او در یک Hub نیز قابل استفاده است. برای مثال می‌توان به خاصیت this.Context.User.Identity.IsAuthenticated دسترسی داشت.
به علاوه در این حالت برای محدود کردن دسترسی کاربران اعتبار سنجی نشده به یک هاب فقط کافی است فیلتر Authorize را به هاب اعمال کنیم. باید دقت داشت که این فیلتر در فضای نام Microsoft.AspNet.SignalR تعریف شده است.

```
[Authorize]
public class ChatHub : Hub
{
  //...
}
```

نگاشت اتصالات، به کاربران واقعی سیستم

```
public class User
    {
        public int Id { set; get; }
        public string Name { get; set; }
        // سایر خواص کاربر


        public HashSet<string> ConnectionIds { get; set; }
    }
```

با توجه به توضیحات ابتدای بحث، هر کاربر با چندین ConnectionId می‌تواند به سیستم متصل شود. بنابراین کلاس کاربران، دارای یک خاصیت اضافی که نیازی هم نیست تا به بانک اطلاعاتی نگاشت شود، به نام ConnectionIds همانند کلاس فوق خواهد بود.
سپس باید لیست اتصالات کاربر را در هربار اتصال و قطع اتصال او به روز کرد:

```
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNet.SignalR;

namespace SignalR05.Common
{
    public class User
    {
        public int Id { set; get; }
        public string Name { get; set; }
        // سایر خواص کاربر


        public HashSet<string> ConnectionIds { get; set; }
```

```
    }

    public class ChatHubHub : Hub
    {
        private static readonly ConcurrentDictionary<string, User> Users = new
ConcurrentDictionary<string, User>();

        public override Task OnConnected()
        {
            connect();
            return base.OnConnected();
        }

        private void connect()
        {
            var userName = Context.User.Identity.Name;
            var connectionId = Context.ConnectionId;

            var user = Users.GetOrAdd(userName,
                _ => new User
                {
                    Name = userName,
                    ConnectionIds = new HashSet<string>()
                });
            lock (user.ConnectionIds)
            {
                user.ConnectionIds.Add(connectionId);
            }
        }

        public override Task OnReconnected()
        {
            connect();
            return base.OnReconnected();
        }

        public override Task OnDisconnected()
        {
            var userName = Context.User.Identity.Name;
            var connectionId = Context.ConnectionId;

            User user;
            Users.TryGetValue(userName, out user);
            if (user != null)
            {
                lock (user.ConnectionIds)
                {
                    user.ConnectionIds.RemoveWhere(cid => cid.Equals(connectionId));

                    if (!user.ConnectionIds.Any())
                    {
                        User removedUser;
                        Users.TryRemove(userName, out removedUser);

                        ///Clients.Others.userDisconnected(userName);
                    }
                }
            }

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

در این مثال با بازنویسی متدهای اتصال، اتصال مجدد و قطع اتصال یک کاربر، توانسته‌ایم:
الف) نگاشتی را بین یک Id اتصال و یک User واقعی سیستم برقرار کنیم.
ب) لیست اتصالات یک کاربر را نیز در اختیار داشته و در زمان قطع اتصال یکی از برگه‌های مرورگر او، تنها یکی از این Idهای اتصال را از لیست حذف خواهیم کرد.

اگر این لیست دیگر Id متصلی نداشت، با فراخوانی متد فرضی Clients.Others.userDisconnected، می‌توان به سایر کاربران مثلا یک Chat، خروج کامل این کاربر را اطلاع رسانی کرد.
با داشتن لیست اتصالات یک کاربر، می‌توان به سایر کاربران اطلاع داد که مثلا کاربر جدیدی به Chat room وارد شده است:

```
Clients.AllExcept(user.ConnectionIds.ToArray()).userConnected(userName);
```

AllExcept در اینجا یعنی سایر کاربران منهای کاربرانی که Id اتصالات آن‌ها ذکر می‌شود. چون این Idها تمامی متعلق به یک کاربر هستند، فراخوانی فوق به معنای اطلاع رسانی به همه، منهای کاربر جاری متصل است.

نظرات خوانندگان

نویسنده: سعید صالحی
تاریخ: ۱۳۹۳/۰۴/۲۵ ۱۲:۵۰

با سلام
خسته نباشید
در صورتی که بخواهیم پیغام فقط به همین یوزری که لاگین کرده بره به جای ؟ توی دستور پایین باید چی بذاریم؟ یا اگه دستور دیگه ای باید استفاده کنیم ممنون می‌شم اگه راهنمایی کنید

```
context.Clients.User("?").displayNotification();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۲۵ ۱۳:۳۶

مطابق مطلب فوق باید ConnectionIdهای او را یافته و به آن‌ها پیام ارسال کنید. روش مدیریت و جمع آوری این ConnectionIdها با مثالی در اینجا بحث شده.
به صورت خلاصه باید تناظری را بین مشخصات کاربر لاگین شده به سیستم یا Context.User.Identity.Name و تمام Context.ConnectionId او برقرار کرد.
بعد با داشتن لیستی از ConnectionId های متناظر (ConcurrentDictionary مثال فوق)، می‌توان به کاربر خاصی پیام ارسال کرد.
در این دیکشنری، به ازای یک Context.User.Identity.Name (مشخصات کاربر لاگین شده)، لیست Idهای اتصال او موجود است.
بعد برای ارسال پیام به یک اتصال:

```
Clients.Client(someConnectionId).sayhello("....");
```

ارسال پیام به چند اتصال، یا لیستی از ConnectionId ها:

```
Clients.Clients(connectionIdsList).sayhello("....");
```

عنوان: **نحوه‌ی ارتقاء برنامه‌های SignalR 1.x به SignalR 2.x**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۹ ۱۸:۴۴
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, jQuery, SignalR

1) اگر هم اکنون یک پروژه جدید SignalR را آغاز و از طریق NuGet وابستگی‌های آن‌را اضافه کنید، به صورت خودکار SignalR نگارش 2 را در این تاریخ دریافت خواهید کرد. این نگارش صرفا با دات نت 4 و نیم به بعد سازگار است. بنابراین اولین کاری که باید برای ارتقاء پروژه‌های SignalR 1.x به نگارش جدید انجام دهید، تغییر Target framework پروژه به نگارش 4.5 است.
2) حذف وابستگی‌های قدیمی

```
Uninstall-Package Microsoft.AspNet.SignalR -RemoveDependencies
```

فرمان فوق را اگر در کنسول پاورشل نیوگت اجرا کنید، به صورت خودکار وابستگی‌های قدیمی SignalR را حذف می‌کند.
3) نصب فایل‌های جدید SignalR

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

برای این منظور تنها کافی است دستور فوق را اجرا نمائید.
4) به روز رسانی ارجاعات اسکریپتی

```
<script src="Scripts/jquery.signalR-2.0.0.min.js"></script>
```

ارجاع به افزونه جی‌کوئری SignalR نیز باید به نگارش 2 ارتقاء یابد.
5) حذف نحوه‌ی تعریف مسیریابی هاب‌های SignalR از فایل global.asax برنامه.

```
protected void Application_Start(object sender, EventArgs e)
{
   //RouteTable.Routes.MapHubs();
}
```

فایل یاد شده را گشوده و سطر فوق را از آن حذف کنید. سپس یک کلاس دلخواه جدید را مثلا به نام Startup، ایجاد و محتوای آن را به نحو ذیل تغییر دهید:

```
using Microsoft.Owin;
using Owin;

[assembly: OwinStartup(typeof(SignalRChat.Startup))]
namespace SignalRChat
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

این فایل به صورت خودکار در زمان آغاز برنامه‌های SignalR 2 مورد استفاده قرار می‌گیرد (با کمک ویژگی assembly: OwinStartup آن).
اگر از آخرین نگارش VS.NET استفاده می‌کنید، این کلاس را توسط گزینه Add -> New Item -> Owin Startup Class نیز می‌توانید اضافه نمائید.