1. 介绍
   1. 为什么使用.Net Core?跨平台；符合现代开发流程：依赖注入、自动化、容器化……
   2. 为什么使用SignalR？MQTT、XMPP……因为主要是面向Web端的即时通讯，SignalR更简单。
   3. 本次课不讲什么？公开课时间有限，不讲.Net Core、js、asp.net core身份验证等基础的东西。因为大家可能熟悉的前端框架不同，所以主要写原生js，不考虑浏览器兼容性等无关问题。
2. SignalR介绍

SignalR是.Net中用来实现网站中服务器端向客户端推送消息的功能的开源框架。可以用来实现站内实时聊天、站内消息通知、股票等实时数据更新等。

Http协议是浏览器端主动发起请求，服务器不能主动发起请求。有一些场景下需要服务器主动通知浏览器端，比如网站即时消息、审核通知、系统报警。可以使用Ajax长连接来解决，但是对于Web服务器的压力太大，尽量别用。如果浏览器端支持WebSocket（基本上现代浏览器都支持），那么可以利用WebSocket实现。

使用原生WebSocket开发难度比较高，微软提供了一个高度封装的框架SignalR，简化了Websocket开发。如果浏览器端支持WebSocket就用Websocket，否则就使用长连接，代码不变。

SignalR在Windows7、Windows 2008下作为服务器也可以正常调试运行，但是用的是长连接，服务器端只有在Windows8及以上、Windows 2012及以上才支持WebSocket。

总结：浏览器端支持Html5 WebSocket并且服务器端为Windows 2012及以上才支持WebSocket，否则性能惨死！

.Net Framework支持SignalR，.Net Core从2.2开始也正式支持SignalR。.Net Core下的SignalR和.Net Framework下的有一些不同，看文档的时候需要注意。其实.Net Core下面的SignalR正式叫法是SignalR Core，但是下面简称SignalR。

SignalR的客户端除了JavaScript之外，也支持.Net、Java、Android等。

1. SignalR服务器端基本使用
   1. 新建普通ASP.Net Core Web应用程序。我们后续还要用到MVC，所以使用“Web应用程序（模型视图控制器）”。注意要使用.Net Core 2.2及以上版本。
   2. 新建一个类文件命名为TestHub，内容是：

using System;

using Microsoft.AspNetCore.SignalR;

using System.Threading.Tasks;

namespace signalRTest1

{

public class TestHub : Hub

{

public async Task SendMessage(string message)

{

await Clients.All.SendAsync("OnMsg",DateTime.Now,message);

}

}

}

这里SendMessage方法就是用来向所有连接到这个Hub的客户端广播OnMsg消息，并且传递一些数据。

这样一个最简单的SignalR服务器端的Hub就完成了。Hub就是一个连接服务器和各个客户端之间的“路由器”，客户端要先连上一个Hub，服务器向客户端推送要经过Hub，A客户端给B客户端发消息也要经过Hub。

* 1. 项目添加对SignalR的支持

在startup.cs中

在public void ConfigureServices(IServiceCollection services)最后添加

services.AddSignalR();

在public void Configure(IApplicationBuilder app, IHostingEnvironment env)最后添加

app.UseWebSockets();

app.UseSignalR(routes =>

{

routes.MapHub<TestHub>("/testHub");

});

这样SignalR服务器端就开发完成了，无论是网页、安卓、IOS、Java、.Net客户端都可以连接上来收发消息了。

1. 网页做为客户端连接SignalR服务器端

网页端可以用单独的网站开发、运行，不过涉及到跨域的问题，后面再讲。这里先简单把网页客户端和SignalR服务器放在同一个项目中。

* 1. 添加js客户端库。在“解决方案资源管理器”中，右键单击项目，然后选择“添加” > “客户端库”。 在“添加客户端库”对话框中，对于“提供程序”，选择“unpkg”。 对于“库”，输入 @aspnet/signalr@1。本地保存的路径注意不要加@，否则后面cshtml中还要转义。
  2. wwwroot下新建一个1.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script src="/lib/signalr/dist/browser/signalr.js"></script>

</head>

<body>

<ul id="msgs">

</ul>

<input type="text" id="txtMsg"/><input type="button" id="btnSend" value="send"/>

<script type="text/javascript">

var txtMsg = document.getElementById("txtMsg");

var btnSend = document.getElementById("btnSend");

btnSend.disabled = true;

var connection = new signalR.HubConnectionBuilder().withUrl("/testHub").build();

connection.on("OnMsg", function (date, message) {

var msg = date + ":" + message;

var li = document.createElement("li");

li.textContent = msg;

document.getElementById("msgs").appendChild(li);

});

connection.start().then(function () {

btnSend.disabled = false;

}).catch(function (err) {

alert(err);

});

btnSend.onclick = function () {

connection.invoke("SendMessage", txtMsg.value)

.then(function () {

txtMsg.value = "";

txtMsg.focus();

})

.catch(function (err) {

alert(err);

});

};

</script>

</body>

</html>

then().catch()这种promise风格的api

客户端通过connection.invoke("SendMessage")调用服务器端方法，参数个数任意。

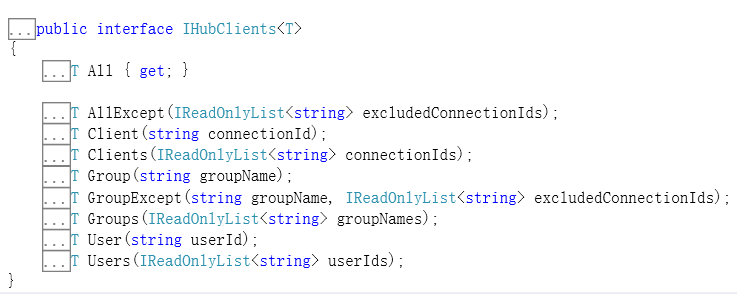
客户端通过connection.on()监听服务器端通过SendAsync发送过来的消息。

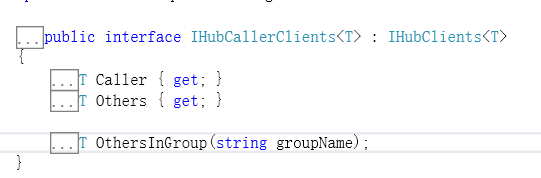
1. Hub类的成员
   1. Clients属性代表所有连接到Hub上的客户端；Groups属性代表群组。后面后详细讲
   2. Context属性代表请求的上下文，是HubCallerContext类型。HubCallerContext类型的ConnectionId属性代表的是连接的Id，一个客户端连接对应一个ConnectionId，本次连接期间这个ConnectionId不变化，到那时开新的页面或者刷新页面或者断开重连会得到一个新的Id；
   3. 几个可以重载的方法：OnConnectedAsync ()，当连接的时候调用；OnDisconnectedAsync ()断开连接的时候调用。可以通过这个来跟踪用户的在线状态。
2. SignalR群组管理

SignalR的Hub提供了群组机制，可以把用户放到某一个或者多个组中，然后针对组进行消息推送。客户端和组之间是多对多的关系

Hub的Groups属性是IGroupManager类型的，两个方法AddToGroupAsync(string connectionId, string groupName)、RemoveFromGroupAsync(string connectionId, string groupName)分别是根据connectionId把连接加入或者从某个组移除。组的名字自定义，如果之前不存在这个组，会自动创建一个组，因此不需要单独创建组，相同groupName为一组的。

Hub的Clients属性是主要成员有：





1. Caller 当前连接的客户端
2. Others，除了当前连接外的其他客户端
3. OthersInGroup(string groupName)名字为groupName的组的所有成员。
4. All：所有连接的客户端
5. Client(string connectionId)指定的ConnectionId对应的客户端
6. Group(string groupName) 组groupName中的客户端

选出需要的成员后，调用SendAsync发送消息，也可以携带参数数据。

1. Hub获取登录状态

Hub和asp.net 的运行线程是独立的，在Hub中不能直接读取asp.net的Session

登录信息在Web中，如何在Hub中获取登录信息呢？

SignalR支持ASP.Net身份验证，只要按照Identity的要求把用户Id等信息生成JwtBearer，把生成的JwtBearer的token值做为js端withUrl第二个参数{ accessTokenFactory: () => token }。只要在Hub类上标记[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]，那么在Hub方法中就可以通过this.Context.User拿到放到JWT中的用户信息。

JWT涉及到加密、解密过程，需要在加密端、解密端配置相同的Secret、Issuer、Audience值。

具体操作如下：

1. 在登陆端验证用户登录信息，如果正确用下面代码把用户信息生成JWTToken
2. jwtSecret（不能太短，否则会报错：Signature validation failed. Keys tried: [PII is hidden]）、issuer、audience的值自定义，但是要和Hub解密时候一致。claims中的内容也允许自定义，涉及asp.net 身份验证问题，不展开讲。

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

var claims = new[]

{

new Claim("UserName", "yzk"),

new Claim("UserId","888"),

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSecret));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(issuer,

audience,claims,

expires: DateTime.Now.AddDays(30),

signingCredentials: creds);

var access\_token = new JwtSecurityTokenHandler().WriteToken(token);

然后通过各种方式把生成的access\_token发给客户端。

1. 客户端在withUrl的时候带着access\_token

var connection = new signalR.HubConnectionBuilder().withUrl("/testHub", { accessTokenFactory: () => access\_token }).build();

1. Hub端配置

建一个空的类ApplicationUser

Startup.cs的public void ConfigureServices(IServiceCollection services)中

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddDefaultTokenProviders();

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.RequireHttpsMetadata = false;

options.SaveToken = true;

options.TokenValidationParameters = new TokenValidationParameters()

{

ValidIssuer = issuer,//ValidIssuer、ValidAudience必须设置，要和token生成方的值一致

ValidAudience = audience,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret)) //secret要和token生成方的值一致

};

//有些情况下JWT的值会放到queryString中的access\_token，而不是在头的Authentication中，因此需要从querystring的access\_token

//读取出来设置到头上

options.Events = new JwtBearerEvents

{

OnMessageReceived = context =>

{

var accessToken = context.Request.Query["access\_token"];

// If the request is for our hub...

var path = context.HttpContext.Request.Path;

if (!string.IsNullOrEmpty(accessToken) &&

(path.StartsWithSegments("/testHub")))//!!注意，这里一定要和实际的hub路径一致。否则会造成websocket协商失败，而降级为使用长连接

{

// Read the token out of the query string

context.Token = accessToken;

}

return Task.CompletedTask;

}

};

});

1. 在Hub上标注[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

然后在Hub的方法上就可以获取Identity中的信息了

var user = this.Context.User;

ClaimsIdentity claimsIdentity = (ClaimsIdentity)user.Identity;

foreach(var claim in claimsIdentity.Claims)

{

Console.WriteLine($"{claim.Type}=${claim.Value}");

}

1. Controller中调用Hub的操作

有时候需要在Controller中调用Hub中的操作实现群发消息等功能。

只要在Controller中通过构造函数注入IHubContext<TestHub>，在Controller中就可以操作Hub中的Clients、Groups等东西了。

private readonly IHubContext<TestHub> hubContext;

public HomeController(IHubContext<TestHub> hubContext)

{

this.hubContext = hubContext;

}

public IActionResult Test1()

{

this.hubContext.Clients.All.SendAsync("OnMsg", DateTime.Now, "来自Controller的消息");

return Content("ok"+DateTime.Now);

}

1. 介绍RuPengMessageHub.Server
   1. git地址：<https://github.com/yangzhongke/RuPengMessageHub>
   2. 项目结构分析

为了支持多应用接入，所以把Hub放到了单独的项目，Web端“RuPengMessageHub.TestWeb”只是其中一个介入的应用而已。运行RuPengMessageHub.TestWeb感受一下。

从RuPengMessageHub.TestWeb项目的开发者角度分析一下RuPengMessageHub.TestWeb的实现。

RuPengMessageHub.Server是Hub服务器项目。安全和业务角度考虑：没有把发送消息、加入群组等直接放到Hub中，而是放到了Controller进行操作。比如A应用允许发送含有某些文字的消息，B应用不允许；再比如A应用允许跨组发消息，B应用不允许。

RuPengMessageHub.NetSDK是提供的一个简化调用Hub服务器的Controller方法的.netsdk封装。其他语言也可以编写自己的sdk封装。

* 1. 如何在Redis通过前缀实现应用数据隔离。
  2. 需要完善的地方。
  3. 跨域访问

如果Hub服务器和Web服务器放到两个域名下，就要启用Cors。

在Hub服务器的Startup.cs中

public void ConfigureServices(IServiceCollection services)中的services.AddSignalR();之前加上services.AddCors();

public void Configure(IApplicationBuilder app, IHostingEnvironment env)的

// Make sure the CORS middleware is ahead of SignalR.

//注意顺序：UseCors、UseSignalR、UseMvc

app.UseCors(builder =>

{

builder.WithOrigins(corsSetting.Value.AllowedOrigins)

.AllowAnyHeader()

.WithMethods("GET", "POST")

.AllowCredentials();

});

var webSocketOptions = new WebSocketOptions()

{

KeepAliveInterval = TimeSpan.FromSeconds(120),

ReceiveBufferSize = 4 \* 1024

};

foreach(var origin in corsSetting.Value.AllowedOrigins)

{

webSocketOptions.AllowedOrigins.Add(origin);

}

app.UseWebSockets(webSocketOptions);