多奇·教育訓練

ASP.NET Core 5 開發實戰

應用開發篇 (SignalR)

多奇數位創意有限公司

技術總監 黃保翕（Will 保哥）

https://blog.miniasp.com

ASP.NET Core SignalR: Introduction

簡介 ASP.NET Core SignalR

ASP.NET Core SignalR 簡介

- ASP.NET Core SignalR 是一個開放原始碼開發框架
- 可提供網頁**即時訊息處理**的能力
- **伺服器**與**瀏覽器**之間可以進行**雙向溝通**
- 可從**伺服器端**直接向**瀏覽器**進行**主動訊息推播**
  - 可大幅減少瀏覽器持續向伺服器請求最新資料的行為

ASP.NET Core SignalR 重要特色

- 自動處理**連接管理**
  - 斷線後會自動重連
- 同時將訊息傳送至所有**已連線**的**用戶端**
  - 例如：聊天室應用
  - 允許即時發送訊息給**所有已連線**的用戶端
  - 將訊息傳送至**特定用戶端**或**用戶端群組**
- 架構延展性高
  - 很容易調整佈署的規模，可應付超大流量

ASP.NET Core SignalR 應用情境

- 需要高頻更新資料的程式
  - 如遊戲、社交網路、投票、拍賣、地圖和 GPS 應用程式等等
- 監控程式
  - 如儀表板、硬體設備資訊監控、CRM、即時銷售通知等等
- 協同作業
  - 如共筆程式、即時會議軟體
- 需要即時通知的程式
  - 如社群訊息、email、聊天室、遊戲等等

ASP.NET Core SignalR 傳輸技術

- SignalR 支援 3 種即時處理訊息的傳輸技術 (按照優先順序排列)
  1. WebSockets
  2. Server-Sent Events
  3. Long Polling

- SignalR 會自動選擇當下環境最適合的傳輸方法！

關於 WebSocket 傳輸技術

- 屬於標準的規範 (RFC 6455、RFC 7936)
- 是一種基於 TCP 連接的**全雙工**通訊協定
- 伺服器與瀏覽器只需要完成一次交握，即可建立持續性的連結，進行雙向資料傳輸
- 減少過去輪詢所造成的傳輸浪費
- 減少每次建立連線的時間耗損，資訊更即時

關於 Server-Sent Events 傳輸技術

- 由伺服器端持續發送**不會中斷的封包**(串流)
- 當有資料更新時，調整送出封包的內容
- 瀏覽器端針對進入的資料流變更來決定資料變更
- 資料流動只能是**單向**從伺服器傳到瀏覽器
- 由於是持續傳送，**對伺服器負載較大**

關於 Long Polling 傳輸技術

- 與**傳統輪詢方法**不同，傳統輪詢會在伺服器沒有更新資料時便結束連線
- 長時間輪詢 (Long Polling) 則是**建立一個長時間的連線**，伺服器在資料有更新時，才送出資料，並結束這次的連線
- 結束連線後，**立即再次發起 Long Polling 要求**，等待下次資料更新

關於 SignalR 最重要的概念：Hub

- **Hub** 是 SignalR 用來處理**伺服器**與**瀏覽器**溝通的橋樑
  - 瀏覽器可以透過程式呼叫伺服器上某個 Hub 中某段程式 (C#)
  - 反之，伺服器也可以決定要呼叫瀏覽器上的某段程式 (JavaScript)
- 溝通過程可以設定**強型別**參數
  - 支援模型綁定能力
- SignalR 支援兩種 Hub 傳輸協定
  - 純文字的 JSON 資料
  - 二進位的 Binary 資料 (基於 MessagePack 格式) (超高效率)

SignalR 支援的平台

- 瀏覽器支援 (ASP.NET Core SignalR JavaScript client)

Browser	Version
Microsoft Internet Explorer	11
Microsoft Edge	最新版
Mozilla Firefox	最新版
Google Chrome; includes Android	最新版
Safari; includes iOS	最新版

- 伺服器支援
  - 只要 ASP.NET Core 支援的作業系統平台都可以執行
  - 使用 WebSockets 時 IIS 必須為 8.0 以上 (Windows Server 2012+)

Getting Started with SignalR on ASP.NET Core

快速上手 ASP.NET Core SignalR

建立新的 ASP.NET Core 專案

- 建立新的 ASP.NET Core 專案
  - **`dotnet new webapp -n SignalRChat`**

- 關於 SignalR 的 NuGet 套件
  - ASP.NET Core 已經內建 SignalR 功能
  - 因此你不需要安裝任何其他相關套件！

- Tutorial: Get started with ASP.NET Core SignalR

安裝前端必要 JS 函式庫

- 安裝 LibMan (Microsoft Library Manager)
  - `dotnet tool install -g Microsoft.Web.LibraryManager.Cli`
- 安裝 JS 函式庫
  - `libman install @microsoft/signalr@latest -p unpkg`
    - `-d wwwroot/js/signalr`
    - `--files dist/browser/signalr.js`
    - `--files dist/browser/signalr.min.js`
- 相關連結
  - 使用 Visual Studio 中的 ASP.NET Core 使用 LibMan
    - [專案] > [加入] > [用戶端程式庫]
  - 搭配 ASP.NET Core 使用 LibMan 命令列介面 (CLI)
  - https://github.com/aspnet/LibraryManager

建立 SignalR Hub

- 建立 Hubs/ChatHub.cs

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRChat.Hubs
{
  public class ChatHub : Hub
  {
    public async Task SendMessage(string user, string message)
    {
      await Clients.All
        .SendAsync("ReceiveMessage", user, message);
    }
  }
}
```

繼承 SignalR 的 Hub，
用來管理用戶的連接及訊息的傳遞

建立一個方法提供給連結的 client 端呼叫

傳送一個訊息給所有的 clients

設定 SignalR 的 DI 容器與 Middleware

- 修改 Startup.cs 檔
  - ConfigureServices() 方法中，加入以下程式

```
services.AddSignalR();
```

  - Configure() 方法中，加入以下程式

```
app.UseEndpoint(endpoints =>
{
    endpoints.MapRazorPages();
    endpoints.MapHub<ChatHub>("/chatHub");
});
```

依照指定的**路由規則**選擇**使用的 Hub**

加入同源政策支援(CORS)

- 若 Client 端與伺服器位於不同來源，連線會被瀏覽器擋住(參考)
- 可以在 ASP.NET Core 設定允許的來源(參考)
  - ConfigureServices() 方法中，加入以下程式

```
services.AddCors();
```

  - Configure() 方法中，加入以下程式 （請加在 **app.UseRouting();** 之前）

```
app.UseCors(builder =>
{
  builder.AllowAnyHeader().AllowAnyMethod()
    .SetIsOriginAllowed(_ => true)
    .AllowCredentials();
});
```

從用戶端建立 SignalR 連線

- 請從 Add SignalR client code 複製原始碼
- 若以 Angular 為例，請先用 npm 安裝 @microsoft/signalr 套件

```
npm install @microsoft/signalr
```

- 然後從 AppComponent 設定與 SignalR 伺服器連線

```
import { HubConnectionBuilder } from '@microsoft/signalr';
...

this.connection = new HubConnectionBuilder()
  .withUrl('https://127.0.0.1:5001/chatHub')
  .build();

this.connection.on("ReceiveMessage", function (user, message) { });

this.connection.start();
```

設定要建立的連線資訊

設定 SignalR 回呼函式

開始建立連線

完整用戶端程式碼範例

https://github.com/coolrare/SignalRChat

```
<script src="~/js/signalr/dist/browser/signalr.js"></script>
<script>
var connection = new signalR.HubConnectionBuilder().withUrl("/chatHub").build();

connection.on("ReceiveMessage", function (user, message) {
    alert(`Hi ${user}, you said: ${message}`)
});

connection.start().then(function () {
    connection.invoke("SendMessage", 'Will', 'Hello World')
    .catch(function (err) {
        return console.error(err.toString());
    });
}).catch(function (err) {
    return console.error(err.toString());
});
</script>
```

前端專案：用戶端斷線自動重連機制

- 預設並不會自動斷線重連，需加入以下設定才會 (Reconnect clients)

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chatHub")
    .withAutomaticReconnect()
    .build();
```

- 設定**斷路器**機制 (Circuit Breaker)

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chatHub")
    .withAutomaticReconnect([0, 3000, 5000, 15000])
    .build();
```

前端專案：與 SignalR 伺服器溝通

- Client 端使用 **invoke** 方法傳送資料

```
sendMessage(name: string, message: string) {
  return this.connection.invoke('SendMessage', name, message);
}
```

- 對應伺服器端 Hub 類別中的方法名稱

```
public async Task SendMessage(string user, string message)
{
  await Clients.All.SendAsync("ReceiveMessage", user, message);
}
```

前端專案：與 SignalR 伺服器溝通

- Client 端使用 **on** 方法指定要接收的名稱

```
this.connection.on('ReceiveMessage', (name, message) => {
  console.log(name, message);
});
```

- 對應伺服器端 Hub 中呼叫 **SendAsync** 的參數

```
public async Task SendMessage(string user, string message)
{
  await Clients.All.SendAsync("ReceiveMessage", user, message);
}
```

.NET 專案：安裝套件與建立連線

- 安裝 Microsoft.AspNetCore.SignalR.Client
  - `dotnet add package Microsoft.AspNetCore.SignalR.Client`

- 與 SignalR 伺服器連線

設定要建立的連線資訊

```
var connection = new HubConnectionBuilder()
  .WithUrl("http://localhost:5000/chatHub")
  .Build();
```

```
await connection.StartAsync();
```

開始建立連線

.NET 專案：與 SignalR 伺服器溝通

- Client 端使用 **InvokeAsync** 方法傳送給伺服器端

```
await connection.InvokeAsync("SendMessage", name, message);
```

- Client 端使用 **On** 方法接收伺服器端的資料

```
connection.On<string, string>("ReceiveMessage",
  (user, message) => {
    Console.WriteLine($"{name}: {message}");
  });
```

SignalR: Advanced

其他 SignalR 開發技巧

Context 物件

- Context 物件用來保留每個連線的相關資料
- Context 物件常用屬性：

Property	Description
`ConnectionId`	每個連線的獨立 ID
`UserIdentifier`	驗證登入的使用者資訊
`User`	取得目前連線ID的使用者資訊
`Items`	用來取得/存放目前連線中共用的資訊，以便在不同方法中存取。(key/value 物件)

Clients 物件 - 1

- Clients 物件包含了目前所有連線的資訊
- Clients 物件常用**屬性**

Property	Description
`All`	取得所有連線的 Client 端
`Caller`	只取得在目前呼叫此方法的 Client 端
`Others`	取得除了目前呼叫此方法以外的所有 Client 端

Clients 物件 - 2

- Clients 物件包含了目前所有連線的資訊
- Clients 物件常用**方法**

Method	Description
AllExpect	取得所有的連線 Client 端(除了指定的連線ID以外)
Client	從單一個連線ID取得指定的 Client 端
Clients	從多個連線ID取得 Client 清單
Group	取得指定的群組
GroupExpect	除了指定的群組外，取得其他的所有群組
Groups	取得指定的群組清單
User	取得指定的某個使用者連線
Users	取得指定的多個使用者連線

使用強型別：避免使用 SendAsync - 1

- 使用 SendAsync 方法傳送資料給 Client 端的缺點是，必須**使用字串的方式(弱型別)**傳送，當拼錯字時，容易造成預期外的錯誤
- 可以透過建立一個介面，宣告 SendAsync 想要呼叫的方法簽章名稱
- 在 Hub 中，可以使用**繼承 Hub<T>** 的方式，讓 Client 有**強型別**的方法可以呼叫

使用強型別：避免使用 SendAsync - 1

- 宣告介面，包含傳送資料給 client 端的方法簽章

```
public interface IChatClient
{
  Task ReceiveMessage(string user, string message);
}
```

- 使用強型別的方式傳送資料給 client 端

```
public class ChatHub : Hub<IChatClient>
{
  public async Task SendMessage(string user, string message)
  {
    // 原來的 await Clients.All.SendAsync("ReceiveMessage", ...);
    // 現在有強型別的 ReceiveMessage 方法可以呼叫
    await Clients.All.ReceiveMessage(user, message);
  }
}
```

處理連線事件

- 透過覆寫 **OnConnectedAsync** 及 **OnDisconnectedAsync** 兩個 hook，可以在 Client 端**連線**和**斷線**後即時得知。

```
static int userCount = 0;

public override async Task OnConnectedAsync()
{
  ++userCount;
  await Clients.All.SendAsync(...);
  await base.OnConnectedAsync();
}

public override async Task OnDisconnectedAsync(Exception exception)
{
  --userCount;
  await Clients.All.SendAsync(...);
  await base.OnDisconnectedAsync(exception);
}
```

驗證和授權 - 1

- SignalR 可以搭配使用 ASP.NET Core Identity 來處理身分驗證及授權的工作， 並**將每個連線與使用者關聯起來**

- 方便使用者透過不同裝置進行 SignalR 連線時， 伺服器端能針對所有使用者連線中的裝置發送訊息

驗證和授權 - 2

- SignalR 伺服器端可在 **ConfigureServices()** 中，使用 **services.AddAuthentication()** 設定使用的授權方式

```
services
   .AddAuthentication(options => { /* 選項設定 */})
   .AddJwtBearer(options => { /* 使用 JWT Token 驗證設定*/ });
```

- 之後在 **Configure()** 使用設定好的驗證規則

```
app.UseAuthentication();
```

驗證和授權 - 3

- 在 Hub 設定中，可以加入 **[Authorize]** 屬性，在Client 端傳入的 Access Token 驗證不通過時，無法連線到指定的 Hub

```
[Authorize]
public class ChatHub : Hub<IChatClient>
{
}
```

驗證和授權 - 4

- 在 JavaScript Client 端中，可以使用 **accessTokenFactory()** 指定傳入的 Access Token

```
connect(accessToken: string): Promise<void> {
  this.connection = new HubConnectionBuilder()
    .withUrl(
      `${apiUrl}/chatHub`,
      { accessTokenFactory: () => accessToken })
    .build();

  return this.connection.start();
}
```

驗證和授權 - 5

- 在 .NET Client 端中，可以使用 **AccessTokenProvider** 指定傳入的 Access Token

```
var connection = new HubConnectionBuilder()
  .WithUrl(
    "http://localhost:5000/chatHub",
    options =>
    {
      options.AccessTokenProvider = async () =>
      {
        return "acceses_token";
      }
  })
.Build();
await connection.StartAsync();
```

依據不同使用者發送訊息

- 加入驗證功能後， SignalR 伺服器端可以使用 **`Context.User()`** 取得指定的使用者， 並發送訊息

```
public async Task TalkTo(string to, string message)
{
  await Clients
    .User(to)
    .SendAsync(...);
}
```

將 Client 端連線加入群組並發送訊息

- 使用 Groups 物件，可以將指定的連線 ID 加入群組，或從群組中移除

```
public async Task JoinGroup(string group)
{
  await Groups.AddToGroupAsync(Context.ConnectionId, group);
}

public async Task LeaveGroup(string group)
{
  await Groups.RemoveFromGroupAsync(Context.ConnectionId, group);
}

public async Task SendMessageToGroup(string group, string message)
{
  await Clients.Group(group)
    .ReceiveMessage(Context.UserIdentifier, message);
}
```

將當前連線加入指定群組

將當前連線從指定群組中移除

取得某個指定的群組

使用 IHubContext 來發送 SignalR 訊息 - 1

- 應用程式中，可以透過 IHubContext，在 Hub 之外的地方發送訊息。

- 在 ASP.NET Core 中，可以透過相依注入的方式，取得 Hub 的對應實體。

- 取得指定的 Hub 實體後，即可透過原有的方法發送訊息到 Client 端。

使用 IHubContext 來發送 SignalR 訊息 - 2

```
public class BroadcastController : ControllerBase
{
  private readonly IHubContext<ChatHub> _hubContext;

  public BroadcastController(IHubContext<ChatHub> hubContext)
  {
    _hubContext = hubContext;
  }

  [HttpPost]
  public async Task<IActionResult> Send(BroadcastMessage message)
  {
    await _hubContext.Clients.All
      .SendAsync("ReceiveMessage", "System", message.Message);
    return Ok();
  }
}
```

透過相依注入的方式，
取得 ChatHub 的實體

使用原有的 Hub 實體發送訊息

聯絡資訊

The Will Will Web

網路世界的學習心得與技術分享

http://blog.miniasp.com/

Facebook

Will 保哥的技術交流中心

http://www.facebook.com/will.fans

Twitter

https://twitter.com/Will_Huang

多奇·教育訓練

THANK YOU!

Q&A