[NET Core] 如何讀取 AppSettings.json 組態設定檔

12542 0 .NET Core 3.1 2021-03-29

.NET Core 跟以往 .NET Framework 存取組態設定檔有很大的不一樣，概略的比較如下：

- .NET Core
  - json format
  - appsettings.json 檔
  - 可繫結強型別
- .NET Framework
  - xml format
  - web.config / app.config
  - Settings.settings
    - Auto Generate Code
    - Scope
      - User
      - Applicationer

這裡就著重在 .NET Core 的組態設定

.NET Core App 讀取 Json 設定檔

安裝

Install-Package Microsoft.Extensions.Configuration.Json

新增一個 .NET Core 的測試專案，新增 appsettings.json 內容如下：

```
{
  "ConnectionStrings": {
    "DefaultConnectionString": "Server=(localdb)\\mssqllocaldb;Database=EFGetStarted.ConsoleApp.NewDb;Trusted_Connection=True;"
  },
  "Player": {
    "AppId": "testApp",
    "Key": "12345678990"
  }
}
```

讀取設定檔步驟

- 通過 IConfigurationBuilder 物件建立 IConfigurationRoot 物件。
- IConfigurationBuilder.SetBasePath 方法是設定檔案的基本路徑
- IConfigurationBuilder.AddJsonFile 方法是讀取設定檔的路徑，完整的路徑為 基本路徑 + AddJsonFile
- IConfigurationRoot[節點名稱] / IConfiguration[節點名稱] 取得設定值

代碼如下：

```
[TestMethod]
public void 讀取設定檔()
{
    var builder = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json");
    var config = builder.Build();

    Console.WriteLine($"AppId = {config["AppId"]}");
    Console.WriteLine($"AppId = {config["Player:AppId"]}");
    Console.WriteLine($"Key = {config["Player:Key"]}");
    Console.WriteLine($"Connection String = {config["ConnectionStrings:DefaultConnectionString"]}");
}
```

當區段不存在的時候，會得到 null

讀取連線字串

IConfiguration.GetConnectionString 方法，讀取 "ConnectionStrings" 區段

```
[TestMethod]
public void 讀取設定檔_GetConnectionString()
{
    var builder = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json");
    var config = builder.Build();
    var connectionString = config.GetConnectionString("DefaultConnection");

    //var dbContextOptions = new DbContextOptionsBuilder<LabEmployeeContext>()
    //                    .UseSqlServer(connectionString)
    //                    .Options;
}
```

IConfigurationProvider.TryGet

IConfigurationRoot.Provider 列出載入那些設定檔

除了可以用索引值讀取，也可以用 IConfigurationProvider.TryGet 讀取，這對動態繫結會很有用

```
[TestMethod]
public void 讀取設定檔_TryGet()
{
    var builder = new ConfigurationBuilder()
                  .SetBasePath(Directory.GetCurrentDirectory())
                  .AddJsonFile("appsettings.json");
    var config = builder.Build();

    //TryGet
    foreach (var provider in config.Providers)
    {
        provider.TryGet("Player:AppId", out var value);
        Console.WriteLine($"AppId = {value}");
    }
}
```

參數綁定強型別

安裝套件 Install-Package Microsoft.Extensions.Configuration.Binder

建立以下物件

```
public class AppSetting
{
    public ConnectionStrings ConnectionStrings { get; set; }

    public Player Player { get; set; }
}

public class AppSetting
{
    public ConnectionStrings ConnectionStrings { get; set; }

    public Player Player { get; set; }
}

public class Player
{
    public string AppId { get; set; }

    public string Key { get; set; }
}
```

ConfigurationBinder.Bind / Get擴充方法，直接將 IConfiguration 轉換成強型別物件

```
[TestMethod]
public void 綁定設定_擴充方法_Get()
{
    var builder = new ConfigurationBuilder()
                  .SetBasePath(Directory.GetCurrentDirectory())
                  .AddJsonFile("appsettings.json");
    var config     = builder.Build();
    var player = config.GetSection("Player").Get<Player>();
    Console.WriteLine($"AppId = {player.AppId}");
    Console.WriteLine($"Key = {player.Key}");
}
```

```
[TestMethod]
public void 綁定設定_擴充方法_Bind()
{
    var builder = new ConfigurationBuilder()
                  .SetBasePath(Directory.GetCurrentDirectory())
                  .AddJsonFile("appsettings.json");
    var config     = builder.Build();
    var appSetting = new AppSetting();
    config.Bind(appSetting);
    Console.WriteLine($"AppId = {appSetting.Player.AppId}");
    Console.WriteLine($"Key = {appSetting.Player.Key}");
    Console.WriteLine($"Connection String = {appSetting.ConnectionStrings.DefaultConnectionString}");
}
```

ASP.NET Core Web Application 讀取 Json 設定檔

Options Pattern

Options Pattern 會使用強型別的類別來提供相關參數設定，當 組態設定 依案例隔離到不同的類別時，應用程式會遵守兩個重要的軟體工程準則：

- 根據介面隔離原則 (ISP)：類別，僅取決於它們使用的配置設定
- 關注點分離：應用程式不同部分的設定不會彼此相依或結合

出自：https://docs.microsoft.com/zh-tw/aspnet/core/fundamentals/configuration/options

強型別的類別，有幾個要求：

- 具名類別。
- 所有公開屬性都會繫結。
- 欄位不會繫結。

IConfigureNamedOptions

- 當有不同區段，相同屬性，不需要額外在定義類別，即可綁定
- 區分大小寫

新增一個 ASP.NET Core Web Application -> API 範本

appsettings.json 內容如下

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",

  "ConnectionStrings": {
    "DefaultConnectionString": "Server=(localdb)\\mssqllocaldb;Database=EFGetStarted.ConsoleApp.NewDb;Trusted_Connection=True;"
  },
  "Player": {
    "AppId": "testApp",
    "Key": "12345678990"
  }
}
```

@ Program.cs

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

下段敘述出自 https://docs.microsoft.com/zh-tw/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1#default-configuration

CreateDefaultBuilder 會以下列順序提供應用程式的預設組態：

1. ChainedConfigurationProvider ：加入現有的 IConfiguration 做為來源。 在預設設定案例中，會新增主機配置，並將其設為_應用程式_設定的第一個來源。
2. appsettings.js使用 JSON 設定提供者。
3. appsettings。 Environment使用json 設定提供者的json 。 例如， appsettings。生產。json和appsettings。開發。json。
4. 應用程式在環境中執行時的密碼 Development 。
5. 使用環境變數設定提供者的環境變數。
6. 使用命令列設定提供者的命令列引數。

@ Startup.cs

Host.CreateDefaultBuilder 會將處理好的 IConfiguration 傳給 Startup 建構函數，代碼如下：

```
public class Startup
{
    public IConfiguration Configuration { get; }

    public Startup(IConfiguration configuration)
    {
        this.Configuration = configuration;
    }
}
```

設定中斷觀察結果如下：

```
public class Startup
{
    public IConfiguration Configuration { get; }

    public Startup(IConfiguration configuration)
    {
        this.Configuration = configuration;
    }

    // This method gets called by the runtime.
    public void Configure(IApplicationBuilder app
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
```

configuration = {ConfigurationRoot}

configuration	{Microsoft.Extensions.Configuration.ConfigurationRoot}
Providers	Count = 5
[0]	{Microsoft.Extensions.Configuration.ChainedConfigurationProvider}
[1]	{JsonConfigurationProvider for 'appsettings.json' (Optional)}
[2]	{JsonConfigurationProvider for 'appsettings.Development.json' (Optional)}
[3]	{EnvironmentVariablesConfigurationProvider}
[4]	{CommandLineConfigurationProvider}
Raw View	

注入 IConfiguration

@ WeatherForecastController.cs

Controller 開一個洞，讓建構函數依賴 IConfiguration，讓 ASP.NET Core 啟動時注入 IConfiguration，Controller 拿到 IConfiguration 做法就跟上面提到的操作一樣了，代碼如下

```
private IConfiguration _config;
public WeatherForecastController(IConfiguration config)
{
    this._config = config;
}
```

ConfigurationBinder.Bind / Get 擴充方法，直接將 IConfiguration 轉換成強型別物件

這樣的注入方式是預設的，只需要設定建構函數依賴即可

範例如下

```
{
  "Player1": {
    "AppId": "testApp",
    "Key": "12345678990"
  },
  "Player2": {
    "AppId": "testApp",
    "Key": "12345678990"
  }
}
```

Options interfaces

有以下幾種介面，請根據你的需求挑選

IOptions<TOptions>
IOptionsSnapshot<TOptions>
IOptionsMonitor<TOptions>

注入 IOptions

- 生命週期為 Singleton
- 不支援，在應用程式啟動後讀取設定資料。
- 不支援，IConfigureNamedOptions(不同的區端綁定相同的屬性)
  https://docs.microsoft.com/zh-tw/aspnet/core/fundamentals/configuration/options

@ Startup_InjectionIOptions.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    //注入 IOptions
    services.AddOptions();

    //注入 IConfiguration

    services.Configure<AppSetting>(this.Configuration);

    //services.AddSingleton<IConfiguration>(Configuration);
}
```

@ WeatherForecastController.cs

Controller 建構函數依賴改成 IOptions<AppSetting>

```
private AppSetting _appSetting;
public WeatherForecastController(IOptions<AppSetting> options)
{
    this._appSetting = options.Value;
```

```
}
```

將上面兩個參數合併，代碼如下

```
public WeatherForecastController(IOptions<AppSetting> options, IConfiguration config)
{
    this._config = config;
    this._appSetting = options.Value;
}
```

注入 IOptionsSnapshot

- 生命週期為 AddScoped。
- 可重新載入設定。
- 支援 IConfigureNamedOptions(不同的區端綁定相同的屬性)

為了演練 IConfigureNamedOptions，所以我將 appsettings.json 改成以下，Player1、Player2 裡面的屬性相同

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",

  "ConnectionStrings": {
    "DefaultConnectionString": "Server=(localdb)\\mssqllocaldb;Database=EFGetStarted.ConsoleApp.NewDb;Trusted_Connection=True;"
  },
  "Player": {
    "AppId": "testApp",
    "Key": "12345678990"
  },

  "Player1": {
    "AppId": "testApp",
    "Key": "12345678990"
  },
  "Player2": {
    "AppId": "testApp",
    "Key": "12345678990"
  }
}
```

@ Startup.cs

注入 Player1、Player2

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    //注入 IOptions
    services.AddOptions();

    //注入 IConfiguration
    services.Configure<AppSetting>(this.Configuration);
    services.Configure<Player>("Player1",this.Configuration.GetSection("Player1"));
    services.Configure<Player>("Player2",this.Configuration.GetSection("Player2"));
}
```

Controller 建構函數依賴 IOptionsSnapshot<AppSetting>

```
private Player _player1;
private Player _player2;
public WeatherForecastController(IOptionsSnapshot<Player> options)
{
    this._player1 = options.Get("Player1");
    this._player2 = options.Get("Player2");
}
```

注入 IOptionsMonitor

- 生命週期為 Singleton
- 變更通知
- 支援 IConfigureNamedOptions(不同的區端綁定相同的屬性)
- 可重新載入的設定
- 選擇性選項無效判定 (IOptionsMonitorCache<TOptions>)

注入方式跟 IOptionsSnapshot 一樣

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    //注入 IOptions
    services.AddOptions();

    //注入 IConfiguration

    services.Configure<AppSetting>(this.Configuration);
    services.Configure<Player>("Player1",this.Configuration.GetSection("Player1"));
    services.Configure<Player>("Player2",this.Configuration.GetSection("Player2"));
    //services.AddSingleton<IConfiguration>(Configuration);
}
```

讓建構函數依賴 IOptionsMonitor<AppSetting>

```
// TODO:依賴 IOptionsMonitor<Player>
public WeatherForecastController(IOptionsMonitor<Player> options)
{
    this._player1 = options.Get("Player1");
    this._player2 = options.Get("Player2");
}
```

綁定驗證

從 nuget 安裝，Install-Package Microsoft.Extensions.Options.DataAnnotations

注入 Validate

- services.AddOptions 方法有兩個擴充方法可以啟用驗證
- ValidateDataAnnotations：驗證屬性有掛 Attribute

[Required] public string AllowedHosts { get; set; }

- Validate：可以寫更複雜的驗證

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    //注入 IOptions
    //services.AddOptions();
    services.AddOptions<AppSetting>()
            .ValidateDataAnnotations()
            .Validate(p =>
                    {
                        if (p.AllowedHosts ==null)
                        {
                            return false;
                        }

                        return true;
                    }, "AllowedHosts must be value"); // Failure message.
    ;

    //注入 IConfiguration
    services.Configure<AppSetting>(this.Configuration);
}
```

建構函數依賴 IOptionsMonitor<AppSetting>

調用 option.Value 屬性觸發驗證

似乎，無法使用 IConfigureNamedOptions

```
public WeatherForecastController(IOptions<AppSetting> options)
{
    try
    {
        this._appSetting = options.Value;
    }
    catch (OptionsValidationException ex)
    {
        foreach (var failure in ex.Failures)
        {
            Console.WriteLine(failure);
        }
    }
}
```

執行結果如下：

```
// TODO:依賴 IOptions<AppSetting>
public WeatherForecastController(IOptions<AppSetting> options)   options = {OptionsManager}
{
    try
    {
        this._appSetting = options.Value;
    }
    catch (OptionsValidationException ex)   ex = {"DataAnnotation validation failed for members: 'AllowedHosts' with the error: 'The AllowedHosts fiel..."}
    {
        foreach (var failure:string in ex.Failures)  ≤ 1ms elapsed   ex = {"DataAnnotation validation failed for members: 'AllowedHosts' with the error: 'The AllowedHosts fiel..."}
        {
            Console.WriteLine(failure);
        }
    }
}
```

ex.Failures Count = 1

[0] "DataAnnotation validation failed for members: 'AllowedHosts' with the error: 'The AllowedHosts field is required.'."

Raw View

不依賴 Option Interface

IOptions 有幫我們處理組態設定，當然也可以讓物件不依賴它

這樣也會失去 Option Interface 帶來的優勢，比如，重新載入檔案、驗證

@ Startup.cs

取得 AppSetting 後注入 AppSetting，這裡我用 AddSingleton，可以根據你的需求變更

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    var appSetting = new AppSetting();
    this.Configuration.Bind(appSetting);

    //注入 AppSetting
    services.AddSingleton(appSetting);
}
```

可以把注入的動作搬到擴充方法

```
public static TConfig Configure<TConfig>(this IServiceCollection services, IConfiguration configuration)
    where TConfig : class, new()
{
    if (services == null)
    {
        throw new ArgumentNullException(nameof(services));
    }

    if (configuration == null)
    {
        throw new ArgumentNullException(nameof(configuration));
    }

    var config = Activator.CreateInstance<TConfig>();
    configuration.Bind(config);
    services.AddSingleton(config);
    return config;
}
```

這樣注入設定就可以省掉一些代碼了

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();
    services.Configure<AppSetting>(this.Configuration);
}
```

@ WeatherForecastController.cs

讓建構函數依賴 AppSetting 物件，處理組態設定的工作交給外部

```
public WeatherForecastController(AppSetting appSetting)
{
    this._appSetting = appSetting;
```