

作者序

.NET 7 是微軟新世代開發框架平台，其特性除了跨平台之外，整個架構做了全面大改寫，擁有全新的 Runtime、框架函式庫、基礎服務與 CLI 命令工具，賦予了全新生命與軟體思維，在撰寫 .NET 程式時，是一種完全不同的設計思路與體驗。同時亦迎合 ChatGPT 熱潮，新增一章教您用 JS 及 MVC 程式與 OpenAI API 串接，在您的應用程式中製作 ChatGPT 聊天問答效果。

而 ASP.NET Core MVC 又是 .NET 中最重要的框架之一，它承續了 ASP.NET MVC 5 薪火，扮演續往開來的角色，二者在 MVC 核心基礎上有似曾相似的影子，有些基礎技術可以沿用，但更多部分是全新的設計。如你以為 ASP.NET Core MVC 僅是 ASP.NET MVC 5 跨平台版本，只需換換 Runtime 或 SDK，就能輕鬆無痛切換成 ASP.NET Core MVC，可能會陷入一種誤謬。

事實上，ASP.NET Core 框架是整個大改寫，包括引入新的相依性注入、Configuration 組態系統、Hosting 主機、Middleware、CLI 命令工具等等，本質上早已脫離上一代 ASP.NET MVC 5 或 ASP.NET Web Forms 思維。無論你 ASP.NET MVC 5 如何精通，但若不學習 ASP.NET Core MVC 獨有框架機制，是絕無可能直接駕馭它，甚至連它的運作原理亦無法參透。

ASP.NET Core MVC 最令我感到驚艷的地方，是大量導入軟工的 Design Pattern 設計模式與 Principals 原則，在 ASP.NET Core 中到處可見軟工技術與思維層次的提升，完完全全超越了前一代 ASP.NET MVC 水平數倍不止，足見 .NET Core 框架設計小組的混厚功力與底蘊，才能打造出如此出色的框架。

然祭司撰寫此書最主要目的，是引領讀者循序漸進地探索 ASP.NET Core 技術殿堂，發現其技術之美與奧義，讓讀者技術與思維在這過程中得到提升與加值，獲得滿滿的知識豐收喜悅。

聖殿祭司 奚江華



範例目錄

範例 2-1	建立第一個 MVC 專案	2-8
範例 2-2	逐步建立自訂的 Controller、Model 及 View	2-14
範例 2-3	使用 ASP.NET Identity 建立使用者帳號及存取管制	2-24
範例 2-4	用 LibMan 升級與管理 Bootstrap 和 jQuery 前端函式庫	2-28
範例 4-1	透過 Options 選項模式讀取組態設定	4-21
範例 4-2	在 Controller 控制器中使用 Logging 記錄資訊	4-33
範例 5-1	Controller 傳遞資料給 View – 以寵物店為例	5-20
範例 5-2	製作員工通訊錄列表	5-28
範例 5-3	以 Scaffolding 快速建立 CRUD 的資料庫應用程式	5-37
範例 5-4	建立新的佈局檔讓 View 套用	5-53
範例 5-5	Controller / Action / View 名稱異動的練習	5-56
範例 6-1	以 Scaffolding 產生員工資料 CRUD 樣板	6-31
範例 6-2	以 Bootstrap 改造及美化 View 檢視的 UI 介面	6-37
範例 6-3	用 LibMan 安裝與升級 Bootstrap 用戶端程式庫	6-41
範例 6-4	以 Section 將 View 自訂的 css 及 js 產生到指定位置	6-44
範例 7-1	製作學生考試成績列表	7-22
範例 7-2	以 Razor 語法判斷成績高低及找出總分最高者	7-25
範例 7-3	在 View 中用 Razor、C# 8 及 LINQ 找出總分最高者	7-30
範例 7-4	將人物牌卡製作成 Partial View，供所有 View 呼叫使用	7-41
範例 7-5	傳遞 model 資料到 Partial View，動態生成不同的牌卡	7-45
範例 7-6	傳遞 ViewData 資料到 Partial View 的非同步語法	7-50

範例 7-7	傳遞 Model 資料到 Partial View 的非同步語法.....	7-51
範例 7-8	Partial View 結合 EF Core 資料庫存取.....	7-53
範例 8-1	用 Line 折線圖繪製月均溫.....	8-8
範例 8-2	Line 的點、線和填充模式變化之綜合演練.....	8-12
範例 8-3	用 Bar 長條圖繪製員工國外旅遊投票統計	8-15
範例 8-4	用 Radar 雷達圖繪製公司營運管理面向指標.....	8-17
範例 8-5	用 Pie 與 Doughnut Chart 繪製職務類型及學歷分佈比例	8-20
範例 8-6	MVC 以 Line 折線圖繪製各地區月份平均氣溫	8-23
範例 8-7	MVC 從 Controller 傳遞資料給 View 的 Line 折線圖繪製 月均溫.....	8-27
範例 8-8	MVC 以 Bar 長條圖統計國外旅遊投票數	8-30
範例 8-9	MVC 以 Radar 雷達圖進行兩類車種之六大面向比較.....	8-34
範例 8-10	MVC 用 Pie 與 Doughnut Chart 繪製年度產品營收及 地區貢獻度.....	8-37
範例 9-1	JSON 物件結構在 JavaScript 中的編解碼與存取	9-9
範例 9-2	JSON 陣列結構在 JavaScript 中的編解碼與存取	9-12
範例 9-3	在 Controller 及 View 中進行 JSON 編碼與解碼.....	9-16
範例 9-4	Controller 傳遞 JSON 資料給 View 繪製月均溫折線圖.....	9-21
範例 9-5	以 MVC 的 Controller / Action 建立 API 服務	9-25
範例 9-6	用 jQuery Ajax 呼叫遠端 API 取回 JSON 汽車銷售資料.....	9-29
範例 9-7	以 Ajax 呼叫後端 API 取回 JSON 資料，繪製汽車銷售 趨勢圖.....	9-34
範例 9-8	以 ASP.NET Core Web API 建立汽車銷售數據查詢專用 API 服務	9-46
範例 9-9	以 Postman 對 Web API 送出請求及接收資料.....	9-51
範例 9-10	以 ASP.NET Core 建立 Minimal APIs – 以 Todo 待辦事項 為例.....	9-53

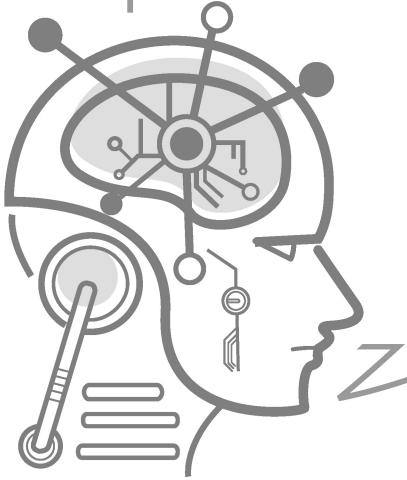
範例 10-1	用 Partial Tag Helper 呼叫部分檢視	10-6
範例 10-2	用 Form 標籤協助程式產生 action 屬性值.....	10-18
範例 10-3	用 Form Action 標籤協助程式產生 formaction 屬性值	10-20
範例 10-4	使用 Label 及 Input 標籤協助程式建立 Form 表單輸入畫面	10-23
範例 10-5	使用 Select 標籤協助程式建立 Country 下拉式選單	10-27
範例 10-6	Enum 列舉繫結到 Select 標籤協助程式.....	10-30
範例 10-7	用 Select 標籤協助程式產生具備 <optgroup> 選項群組的 下拉選單	10-32
範例 10-8	Select 標籤協助程式的多重選取.....	10-34
範例 10-9	用 TextArea 標籤協助程式輸入顧客意見調查.....	10-37
範例 10-10	對 Input 輸入欄位做驗證訊息及驗證摘要	10-41
範例 10-11	利用 Cache 標籤協助程式設定網頁內容快取.....	10-45
範例 10-12	建立自訂 Email 標籤協助程式	10-54
範例 10-13	建立自訂非同步 Email 標籤協助程式	10-56
範例 11-1	以 ValidationMessage 及 ValidationSummary 方法產生輸入 驗證	11-19
範例 11-2	自訂 HTML Helpers.....	11-33
範例 11-3	用 Migrations 建立資料庫及樣本資料	11-36
範例 11-4	顯示員工資料清單 – Index	11-45
範例 11-5	顯示員工明細資料 – Details	11-48
範例 11-6	新增員工資料 – Create.....	11-53
範例 11-7	編輯員工資料 – Edit.....	11-57
範例 11-8	刪除員工資料 – Delete.....	11-60
範例 12-1	建立 ProductList 產品列表之檢視元件	12-4
範例 12-2	在 Partial View 呼叫檢視元件.....	12-12
範例 12-3	傳遞參數至檢視元件的幾種方式	12-16

範例 13-1	網銀服務之 DI 相依性注入建立與調用	13-7
範例 13-2	將 IZipCodeService 服務注入 View，用以查詢郵遞區號	13-14
範例 13-3	透過 Service 注入 City 縣市資料到下拉式選單 UI	13-18
範例 13-4	在 View 直接注入 Configuration 態相依性	13-22
範例 13-5	用自訂 MyHtmlHelper 覆寫預設的 HTML Helpers	13-24
範例 14-1	App 組態的建立與讀取	14-6
範例 14-2	載入自訂 JSON、INI 及 XML 組態檔	14-10
範例 14-3	GetSection、GetChildren 與 AsEnumerable 方法讀取組態 區段	14-19
範例 14-4	利用 Options Pattern 讀取「今日特餐」組態資訊	14-29
範例 14-5	Options Pattern 結合前端 Select 協助標籤顯示電腦硬體選項	14-35
範例 15-1	使用 EF Core 對 Northwind 資料庫 Scaffolding 產出 CRUD 程式	15-12
範例 16-1	在 MVC 專案以 Code First 建立 Blog 應用程式及資料庫	16-4
範例 18-1	申請免費試用的 Azure 帳號	18-2
範例 18-2	將 ASP.NET Core MVC 應用程式部署至 Azure 的 App Service	18-7

CHAPTER

4

ASP.NET Core 框架與基礎服務



ASP.NET Core 是一個跨平台、高效能、開源框架，用來建立現代化、雲端基礎、Internet 連結（指 IoT）的應用程式。然而在背後支撐整個框架運作的是眾多基礎服務，包括了 Hosting、Configuration 組態系統、相依性注入、Middleware、Routing、Environment、Logging 等服務，因此了解每個服務功用，服務如何調整與設定，便是本章要談的內容。

4-1 ASP.NET Core 框架簡介

以下從三個面向介紹 ASP.NET Core 框架，闡述其架構設計之改變，對開發與執行帶來的正面影響，以及最終產生的好處與優勢。

❖ 使用 ASP.NET Core 之利益

ASP.NET Core 是 ASP.NET 4.x 的重新設計，架構上變得更精簡與模組化，提供以下好處：

- 能在 Windows、macOS 和 Linux 上開發、建置與執行
- 整合現代化、client-side 框架與開發流程
- 用統一劇本建立 Web UI 和 Web APIs
- 一個雲端就緒、環境為基底的組態系統
- 內建 Dependency Injection 相依性注入
- 一個輕量級、高效能和模組化 HTTP Request Pipeline
- 新增 Razor Page 和 Blazor 專案開發模式
- 支援使用 gRPC 託管遠端程式呼叫（RPC）服務
- 能夠裝載到 Kestrel、IIS、HTTP.sys、Nginx、Apache、Docker，或在你自己的程序中自我裝載（self-host）
- 支援 .NET Core Runtime 多版本並行（Side-by-side versioning）
- 可簡化現代化 Web 開發的工具
- Open-source 開源和以社群為中心

❖ 使用 ASP.NET Core 建立 Web UI 和 Web APIs

ASP.NET Core 在建立 Web UI 和 Web APIs 方面提供的功能有：

- MVC Pattern 能助你的 Web UI 和 Web APIs 更具有可測試性
- Razor Pages 是以 Page 為基礎的程式模型，使得建立 Web UI 更容易和更具生產性
- Razor Markup 為 Razor Pages 和 MVC Views 提供了更具生產力語法

- Tag Helpers 能夠讓 Server 端程式參與 Razor 檔中 HTML elements 元素的建立與轉譯（Rendering）
- 內建多種資料格式和內容協商，使你的 Web APIs 可以覆蓋廣泛的用戶端，包括瀏覽器和行動裝置
- Model Binding 自動將 HTTP Requests 對應到 Action 方法的參數
- Model Validation 自動執行 Client 端與 Server 端的驗證

❖ Client 端開發（指 Front-End 前端開發）

目前流行的前端框架像 Bootstrap、Angular、React，在 .NET CLI 或 Visual Studio 專案樣板中都有內建支援，在開發這類前端程式時，可得到很好的支援。到了 ASP.NET Core 3.0 時還新增 Blazor 框架支援，它是由 C# 撰寫 Web UI 前端互動程式的一種新專案。

由以上幾個面向可體認到，.NET 的開放性、跨平台能力、高效能、前後端解決方案豐性，都是大大超越前一代。

4-2 ASP.NET Core Fundamentals 基礎服務概觀

若要理解 ASP.NET Core 框架全貌，從它的基礎服務與機制探索起，了解它提供哪些功能，這些服務又是如何交織運作，便能概要掌握其大體技術光譜，下面是 ASP.NET Core 框架的基礎服務大分類圖。

這些服務支撐起整個 ASP.NET Core 應用程式的運行，而服務之間也彼此協同與連動，下面說明每個服務概要功能：

- Host：裝載與執行 .NET Core 應用程式的主機環境，它封裝了所有 App 資源，如 Server、Middleware、DI 和 Configuration，並實作 IHostedService

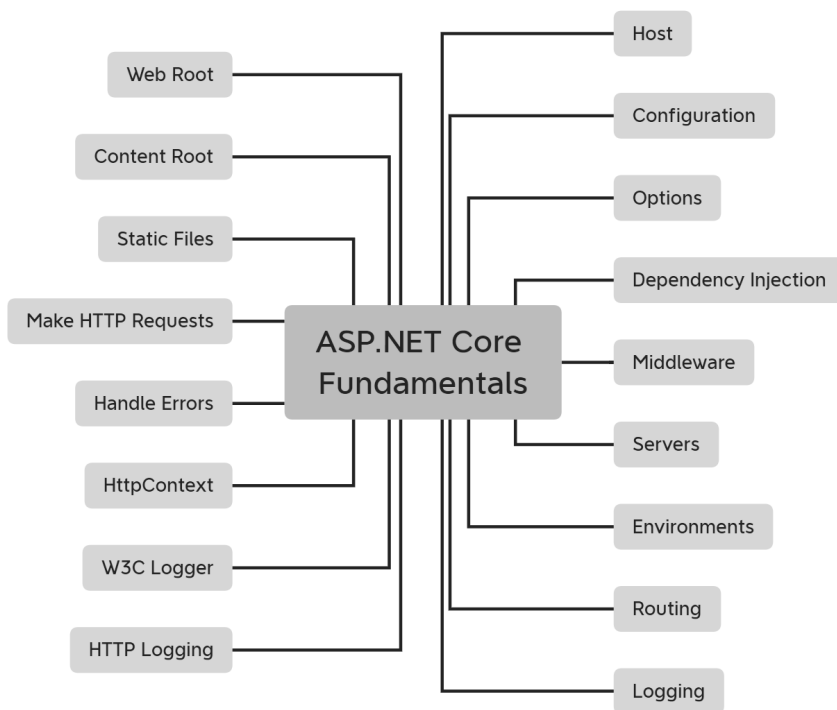


圖 4-1 ASP.NET Core Fundamental 基礎服務分類

- Server：指 HTTP Server 或 Web Server 伺服器，用於監聽 HTTP 請求與回應的網頁伺服器
- Dependency Injection：相依性注入，亦稱 DI Container
- Middleware：在處理 HTTP 請求的管線中，包含一系列 Middleware 中介軟體元件
- Configuration 組態：ASP.NET Core 的組態框架，提供 Host 和 App 所需的組態存取系統
- Options：是指 Options Pattern 選項模式，用類別來表示一組設定，.NET Core 中大量使用選項模式設定組態
- Environment：環境變數與機制，內建 Development、Staging 與 Production 三種環境
- Logging：資訊或事件的記錄機制

- Routing：自 ASP.NET Core 3.0 開始採用端點路由，它負責匹配與派送 HTTP 請求到應用程式執行端點
- Handle Errors：負責錯誤處理的機制
- Make HTTP Request：是 HttpClientFactory 實作，用於建立 HttpClient 實例
- Content Root：內容根目錄，代表專案目前所在的基底路徑
- Web Root：Web 根目錄，專案對外公開靜態資產的目錄

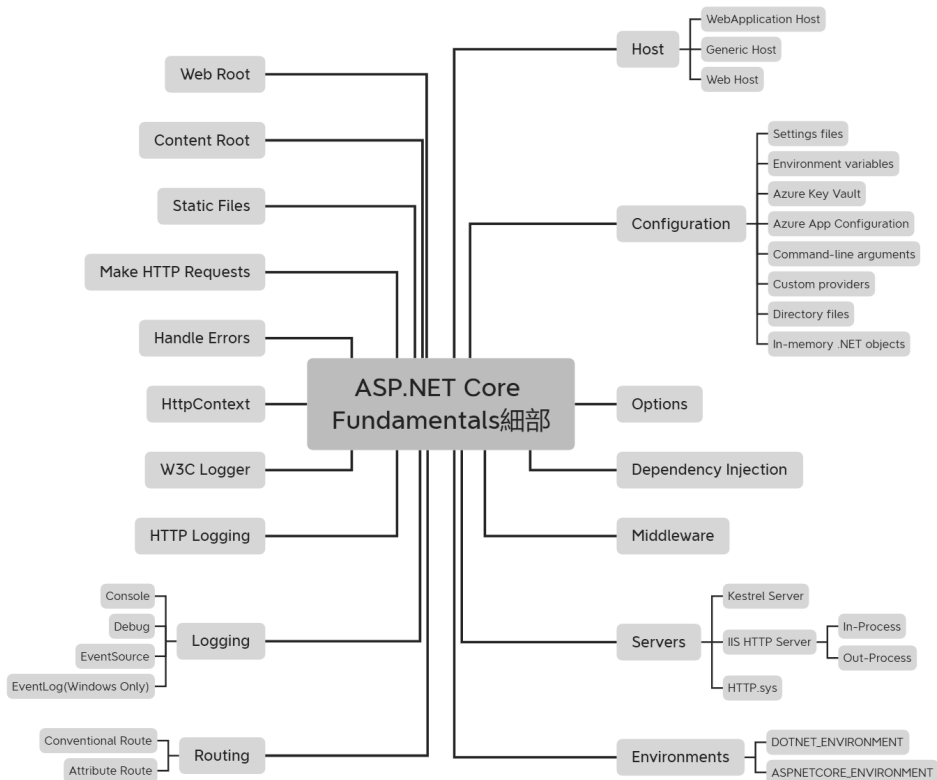


圖 4-2 ASP.NET Core Fundamental 基礎服務明細

以上 Fundamentals 服務如何影響 ASP.NET Core App ? 較為顯著的有：

- 掌控 ASP.NET Core App 系統運作
- 提供 Hosting 和 Web Server 組態設定
- 提供各種環境變數與組態值設定
- 提供多重環境組態設定：Development、Staging 和 Production
- 提供 DI 及 Middleware 設定
- 提供路由設定
- 提供效能調校、Logging 等一堆功能

是故，開發人員若想全面掌握 ASP.NET Core，必須熟悉這些基礎服務知識與技巧，方能輕鬆駕馭。

4-3 重要基礎服務簡介

本節針對最為重要的基礎服務做介紹，讓您了解每個服務負責什麼功能，以及如何叫用這些服務。

4-3-1 ASP.NET Core 應用程式載入過程

下圖是 ASP.NET Core 應用程式執行過程相關檔案，它有六個重要步驟，可與專案程式相對應，並以可目視及驗證的角度來論述，至於框架背景或底層不可視的部分就不列入討論。

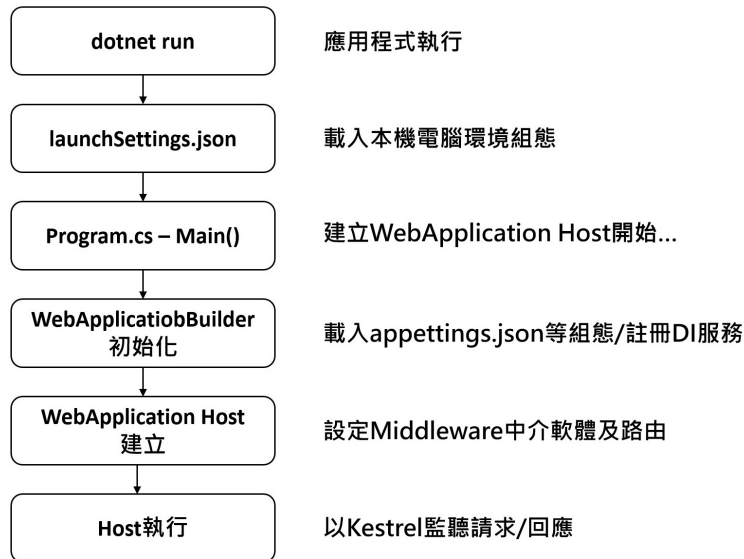


圖 4-3 ASP.NET Core App 主要執行與載入過程

過程說明：

1. .NET 應用程式啟動，無論是用 F5 / Ctrl + F5 / dotnet run 方式執行
2. 首先載入 launchSettings.json 組態，此組態是供本機電腦環境使用
3. 執行 Program.cs，其 Main()是程式進入點，透過 WebApplicationBuilder 建立 WebApplication Host 主機
4. WebApplicationBuilder 初始化時，會載入環境變數與組態，以及約 250 種以上框架提供的服務亦會加入到 DI Container，自訂服務亦是在 DI Container 註冊
5. WebApplicationBuilder.Build() 方法建立 WebApplication Host 主機，接著設定 HTTP Request 請求的 Middleware 中介軟體與路由
6. 設定好 Host 主機所有組態和軟體服務後，呼叫 Run()方法執行應用程式，Kestrel Web Server 開始傾聽 HTTP 請求，並回應結果

了解執行過程，可讓你串起整個基礎框架服務的執行順序，理解它們是在什麼階段被載入執行，又扮演何種角色，及彼此的關聯性。後續在說明個別服務功能時，才不會覺得是一群零散、各自為政的服務，同時在撰寫程式時，能更清楚什麼功能要在哪調整。

4-3-2 本機開發電腦環境組態檔 - launchSettings.json

當建立 ASP.NET Core 專案時，預設會有 launchSettings.json 和 appsettings.json 兩個組態檔，launchSettings.json 是本機開發電腦的環境組態檔，裡面分兩大類、三個區塊，第一類是 IIS 設定，第二類是 Profiles 設定。

📄 Properties/launchSettings.json

```
{
  "IIS 設定": {
    "iisSettings": {
      "windowsAuthentication": false,
      "anonymousAuthentication": true,
      "iisExpress": {
        "applicationUrl": "http://localhost:21358",
        "sslPort": 44310
      }
    }
  },
  "profiles 設定": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5239",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
```

```
"applicationUrl": "https://localhost:7299;http://localhost:5239",
"environmentVariables": {
  "ASPNETCORE_ENVIRONMENT": "Development"
}
},
"IIS Express": {
  "commandName": "IISExpress",
  "launchBrowser": true,
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Development"
  }
}
}
}
```

此組態檔會決定專案執行與行為，例如用 IIS Express 或 Kestrel 網頁伺服器執行，是否要啟動瀏覽器、環境變數或應用程式監聽的 URL 網址，深入部分，在第 14 章組態檔會解釋其行為與作用。

TIP

launchSettings.json 僅供本機電腦使用，不參與部署

4-3-3 Program.cs - Main() 建立 Host 主機

Program 的 Main() 主要任務是建立 Host 主機／執行，以下列出傳統及 Top Level Statement 語法對比：

Program.cs (傳統語法)

```
public class Program
{
    public static void Main(string[] args)
    {
        WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddControllersWithViews();
    }
}
```

```
WebApplication app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
}
```

Program.cs (Top Level Statement 語法)

```
//var builder = WebApplication.CreateBuilder(args);
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

//var app = builder.Build();
WebApplication app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
```

```
app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

兩種語法形式差異，僅在於是否使用 `Main()` 方法表達程式進入點，而建立 ASP.NET Core MVC 專案時，預設會使用 Top Level Statement，這部分是在建立專案時【不要使用最上層陳述式】來決定，若勾選會產出 `Main()` 方法，反之則無。



圖 4-4 勾選選擇專案的【不要使用最上層陳述式】

而 Host 是：裝載與執行 .NET 應用程式的主機環境，它封裝了所有 App 資源，如 Server、Middleware、DI 和 Configuration。換個說法，Host 是一個物件，封裝了前述種種相互依賴的服務，其目的只有一個，便是「生命週期管理」控制 App 應用程式啟動，及順利關閉 Host 主機。

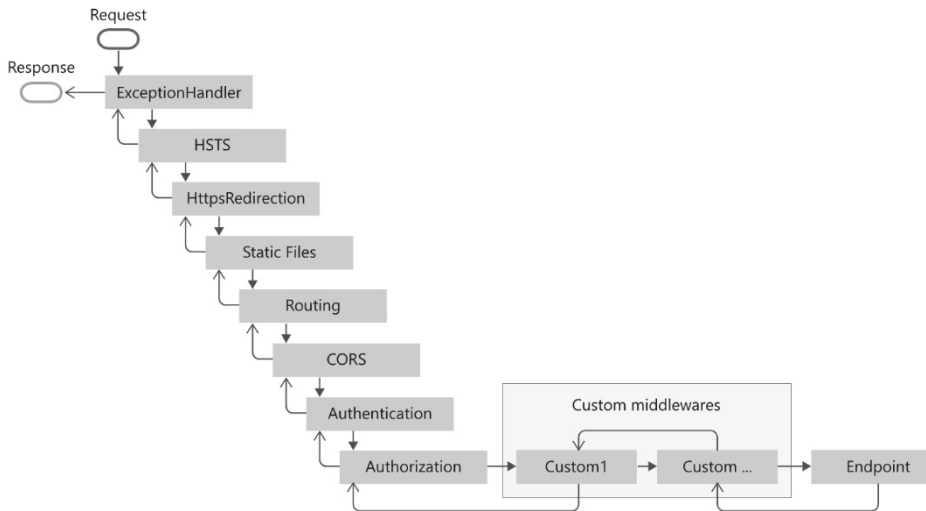


圖 4-6 Middleware 元件執行順序

4-3-5 Configuration 組態

ASP.NET Core 組態是基於 Key-Value Pairs 形式，組態提供者 (Configuration Providers) 從各種組態來源讀取資料後，再以 Key-Value 成對的方式儲存在組態系統中。

例如 ASP.NET Core 專案預設有 launchSettings.json 和 appsettings.json 兩個組態檔，前者是本機開發電腦環境組態檔，後者是給應用程式使用的組態檔，下面是 appsettings.json 組態內容。

📄 appsettings.json

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

← 預設組態設定

```

{
  "ConnectionStrings": {
    "DatabaseContext": "Server=(localdb)\\mssqllocaldb;Database=ProductDB;
      Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Developer": {
    "Name": "聖殿祭司",
    "Email": "dotnetcool@gmail.com",
    "Website": "https://www.codemagic.com.tw"
  }
}

```

資料庫連線字串設定

自訂組態設定

在新建 MVC 專案時，appsettings.json 僅有 Logging 和 AllowedHosts 兩區段，在此新增 ConnectionStrings 和 Developer 設定。在 View 以 @inject 調用 IConfiguration 實例，存取 ConnectionStrings 和 Developer 兩區段設定值。

Fundamental/ReadAppsettings.cshtml

```

@using Microsoft.Extensions.Configuration
@inject IConfiguration configuration
...
<p>DatabaseContext 資料庫連線 : @configuration["ConnectionStrings:DatabaseContext"]</p>

Developer 資訊如下:
<ul>
  <li>Name : @(configuration.GetValue<string>("Developer:Name"))</li>
  <li>Email: @(configuration.GetValue<string>("Developer:Email",
    "找不到 Email"))</li>
  <li>Website: @(configuration.GetSection("Developer:Website").Value)</li>
</ul>

```

讀取 ConnectionString 區段組態

讀取 Developer 區段組態

說明：以上用三種語法讀取組態值，至於實際語法為何如此，第 14 章有專門介紹，於此先不細述

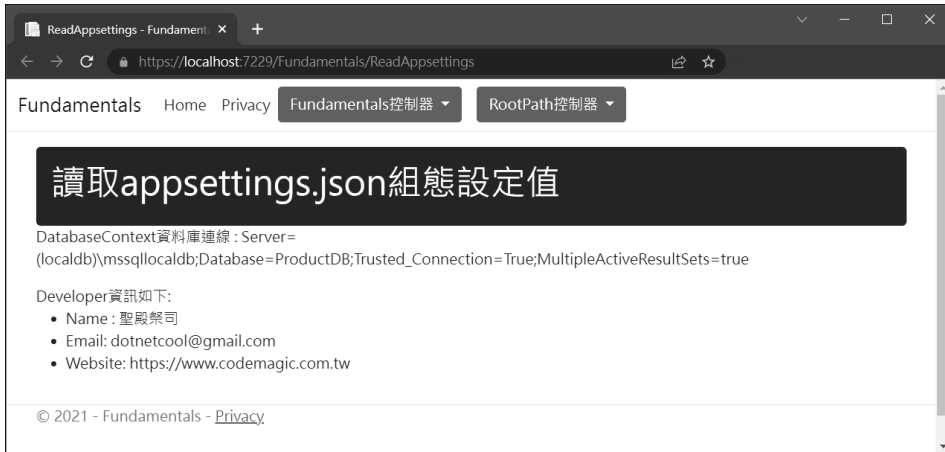


圖 4-7 讀取組態值

組態檔中有中文設定值，若執行時顯示亂碼，請用 Visual Studio 另存成 UTF-8 編碼格式即可解決。

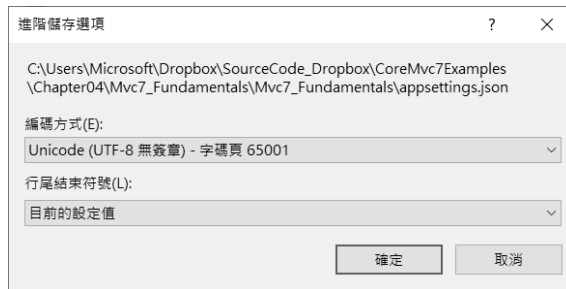


圖 4-8 以 UTF-8 編碼格式儲存

組態資料來源不僅支援 JSON 檔，完整支援如下，每種來源都有相對應的組態提供者負責讀取及解析：

- 環境變數
- 設定檔（JSON、XML、INI）
- 命令列參數
- 目錄檔案（Key-per-file）
- In-Memory .NET 物件
- Azure Key Vault
- Azure App Configuration
- 自訂 Provider

也就是 `GetCurrentDirectory` 方法回傳的路徑再補上「`/wwwroot`」就是 Web 根目錄路徑。

Web 根目錄 `wwwroot` 在 Visual Studio 中可直接看見，裡面皆為靜態資源檔，但凡要公開讓網路讀取的 `images`、`css`、`js`、`json` 或 `xml` 檔都是在此建立。

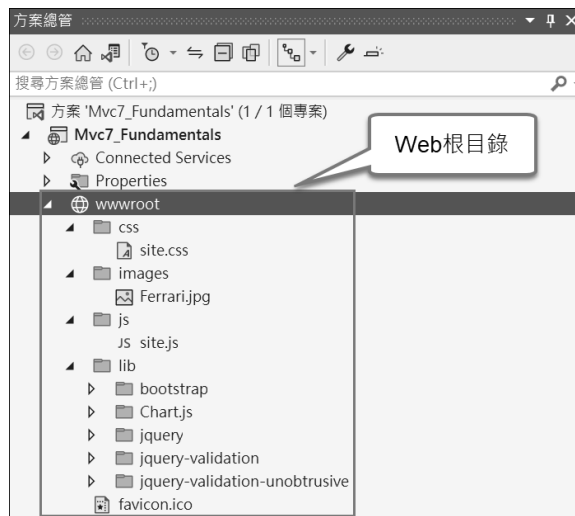


圖 4-12 Web 根目錄

❖ ContentRoot 和 WebRoot 路徑調整

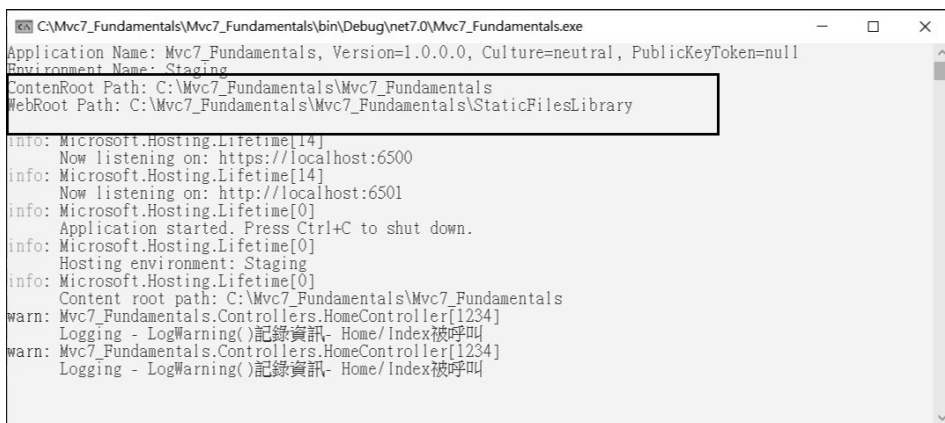
一般情況下，`ContentRoot` 和 `WebRoot` 使用系統預設值就行了，但若想對 `ContentRoot` 和 `WebRoot` 路徑做調整，可在 `CreateDefaultBuilder` 方法中，用 `UseContentRoot` 和 `UseWebRoot` 方法指定路徑參數：

📄 Program.cs

```
var builder = WebApplication.CreateBuilder(new WebApplicationOptions
{
    Args = args,
    ApplicationName = typeof(Program).Assembly.FullName,
    EnvironmentName = Environments.Staging,
```

```
ContentRootPath = Directory.GetCurrentDirectory(),
WebRootPath =
    Path.Combine(Directory.GetCurrentDirectory(), "StaticFilesLibrary")
});

Console.WriteLine($"Application Name: {builder.Environment.ApplicationName}");
Console.WriteLine($"Environment Name: {builder.Environment.EnvironmentName}");
Console.WriteLine($"ContentRoot Path: {builder.Environment.ContentRootPath}");
Console.WriteLine($"WebRoot Path: {builder.Environment.WebRootPath}");
```



```
C:\Mvc7_Fundamentals\Mvc7_Fundamentals\bin\Debug\net7.0\Mvc7_Fundamentals.exe
Application Name: Mvc7_Fundamentals, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Environment Name: Staging
ContentRoot Path: C:\Mvc7_Fundamentals\Mvc7_Fundamentals
WebRoot Path: C:\Mvc7_Fundamentals\Mvc7_Fundamentals\StaticFilesLibrary

info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:6500
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:6501
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Staging
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Mvc7_Fundamentals\Mvc7_Fundamentals
warn: Mvc7_Fundamentals.Controllers.HomeController[1234]
      Logging - LogWarning()記錄資訊- Home/Index被呼叫
warn: Mvc7_Fundamentals.Controllers.HomeController[1234]
      Logging - LogWarning()記錄資訊- Home/Index被呼叫
```

圖 4-13 自訂及顯示 Content Root 及 Web Root 路徑

特別是 Web 根目錄，若想將預設的 `wwwroot` 改用自己的「`StaticFilesLibrary`」目錄作為網路公開服務，可用 `WebApplicationOptions` 選項指定路徑參數，這樣 Web 根目錄之路徑就會改變。但是相對的，所有 `images`、`css`、`js`、`xml` 檔也必須搬移至新目錄才行，否則會讀不到對映的資源檔。

❖ 用 Middleware 設定靜態檔目錄

另一個跟 `WebRoot` 相關議題是，若在 `WebRoot` 之外有其他目錄存放著靜態資源檔，希望和 `wwwroot` 共同服務，那麼可用 `StaticFile` 中介軟體設定靜態檔目錄：

 Program.cs

```
...
app.UseStaticFiles(); //for the wwwroot folder

app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new
        PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(),
            "StaticFilesLibrary"))
});
```

或加用 RequestPath 屬性設定「/StaticFiles」目錄名稱：

```
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new
        PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(),
            "StaticFilesLibrary")),
    RequestPath = "/StaticFiles"
});
```

以上兩種方式二擇一，第一種沒有指定 RequestPath，檔案請求路徑維持「~/…»，第二種指定了 RequestPath，View 中的 的 src 請求路徑須改成「~/StaticFiles/…»：

 Views/RootPath/WebRootPath.cshtml

```

<br />

```

4-3-9 Logging 記錄

ASP.NET Core 內建記錄資訊的 Logging API，亦可與第三方 Logging 提供者（Providers）搭配使用，內建提供者有：

- Console
- Debug

- Windows 平台的 Event Tracing
- Windows 平台的事件記錄
- Azure App Service (需參考 Microsoft.Extensions.Logging.AzureAppServices 的 NuGet 套件)
- Azure Application Insights (需參考 Microsoft.Extensions.Logging.ApplicationInsights 的 NuGet 套件)

Logging 提供者會將 Log 記錄輸出或寫入到不同目的端，例如 Console 提供者會輸出 Log 記錄到 Console 中，事件記錄提供者就寫入事件檢視器，而 Azure Application Insights 儲存 Logs 在 Azure Application Insights 中。

ASP.NET Core 預設會加入 Console、Debug 和 Windows 平台的 Event Tracing 提供者：

Program.cs

```
var builder = WebApplication.CreateBuilder(args);

//加入 Logging Providers
builder.Logging.ClearProviders(); //清除所有 ILoggerProviders
builder.Logging.AddConsole();
builder.Logging.AddDebug();
builder.Logging.AddEventSourceLogger();
builder.Logging.AddEventLog(); //Windows Only

builder.Logging.AddAzureWebAppDiagnostics();
builder.Logging.AddApplicationInsights();
```

← 系統預設提供者

← 加入其他提供者

若加入多重 Logging 提供者，Logs 記錄能夠發送到多重目的作寫入，例如預設加入了四種提供者，那麼記錄資訊時，就會同時寫入到這四種目的端。

範例 4-2 在 Controller 控制器中使用 Logging 記錄資訊

以下在 Home 控制器/Index 動作方法使用 Logging 記錄資訊。

step01 | 在 Home 控制器建構函式注入 ILogger 相依性實例

 Controllers/HomeController.cs

```
...
using Microsoft.Extensions.Logging;

namespace Mvc7_Fundamentals.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }
    }
}
```

DI 相依性注入

指定 Category 分類名稱

說明：Category 分類名稱指定為控制器類別名稱，但指成任何字串執行不會產生錯誤

step02 | 在 Index 動作方法以 Log 方法記錄資訊

```
public IActionResult Index()
{
    EventId eventId = new EventId(1234, "我的記錄資訊");
    _logger.LogWarning(eventId, "LogWarning() 記錄資訊- Home/Index 被呼叫");
    //以上亦可寫成下面一行
    _logger.LogWarning(1234, "LogWarning() 記錄資訊- Home/Index 被呼叫");
    return View();
}
```

使用 LogWarning 方法記錄資訊

事件 Id

記錄資訊

說明：Logging 支援六種層級記錄方法：LogTrace、LogDebug、LogInformation、LogWarning、LogError、LogCritical。雖說每個方法都能使用，但因涉及系統 Loggin 預設組態層級關係，低於層級設定的記錄方法，資訊不會寫至記錄目的端，細節稍後再說明

step03 按 F5 執行，瀏覽 Home/Index 網址，在 Visual Studio【偵錯】→【視窗】→【輸出】→顯示輸出來源「偵錯」，可看見 Log 記錄輸出

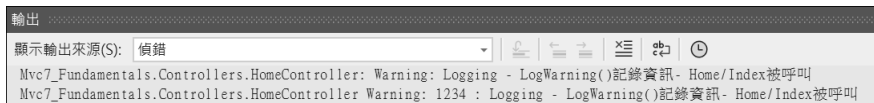


圖 4-14 在 Visual Studio 偵錯輸出中檢視 Log 記錄資訊

step04 另一種是用 dotnet run 執行，再瀏覽 Home/Index，於終端機視窗中亦可看到 Log 記錄輸出資訊



圖 4-15 在終端機視窗檢視 Log 記錄輸出資訊

step05 若有用 AddEventLog()方法加入事件記錄提供者，無論用哪種方式執行，皆會寫入 Windows 事件檢視器中（限 Windows 平台）



圖 4-16 Windows 平台事件檢視器中的記錄資訊

❖ Log Level 記錄層級

每個 Log 記錄時皆會指定一個 Log Level 列舉值，作用是指出記錄是嚴重或重要性程度，依重要性程度最高至最低，Log Level 列舉值有下表幾種。

表 4-2 Log 記錄層級

等級	代碼	方法	說明
None	6	--	指定記錄類別不應寫入訊息
Critical	5	LogCritical	發生需要立即注意的失敗。範例：資料遺失情況、磁碟空間不足
Error	4	LogError	發生無法處理的錯誤和例外狀況。這些訊息表示目前的作業或要求失敗，而不是整個應用程式的失敗。
Warning	3	LogWarning	針對異常或非預期的事件。通常會包含不會導致應用程式失敗的錯誤或狀況。
Information	2	LogInformation	追蹤應用程式的一般流程。可能具有長期值。
Debug	1	LogDebug	用於偵錯工具和開發。在生產環境中，請謹慎使用，因為這是大量的磁片區。
Trace	0	LogTrace	包含最詳細的訊息。這些訊息可能包含敏感性應用程式資料。這些訊息預設為停用，不應在生產環境中啟用。

那 Log Level 記錄層級會如何影響程式？在 appsettings.json 中設定如下：

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information", ← 預設為 Information 層級
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```