新世代 ASP.NET Core MVC 應用程式初體驗

CHAPTER 13

.NET 6 是微軟新世代開發技術，不但實現了.NET 框架的大一統、跨平台開發與執行，同時整個框架亦始無前例的大改造，全新的 Runtime、框架函式庫、基礎服務與 CLI 命令工具。再輔以 Visual Studio 2022 開發工具的大力支援，著實讓人耳目一新，讓人迫不急待想探索其神祕魔力。

13-1 什麼是.NET（Core）？

.NET 6 針對行動裝置、桌面、IoT 和雲端應用程式提供統一的 SDK、基底函式庫及 Runtime 執行環境，完成了首次平台的大一統工作。那什麼是.NET(Core)？簡單來說它就是跨平台版本的.NET 框架，也是繼.NET Framework 之後的新世代版本，下表做幾個面向對比。

表 13-1 .NET Core 與.NET Framework 名詞對比

	跨平台新世代	傳統 Windows 平台
.NET 平台	.NET（Core）	.NET Framework
網頁技術	ASP.NET Core	ASP.NET
網頁 MVC 框架	ASP.NET Core MVC	ASP.NET MVC
支援作業系統	Windows, macOS, Linux	Windows

.NET（Core）的使命不僅是跨平台，更重要的是，它具備更小的模組顆粒，意謂模組更新迭代更快、更容易，同時吃資源更少、啟動速度更快，也適合在 Microservices 環境中執行，並支援 Cloud 雲端平台、Mobile、Gaming、AI、IoT 等應用開發，堪稱微軟下一個十年的主流平台。

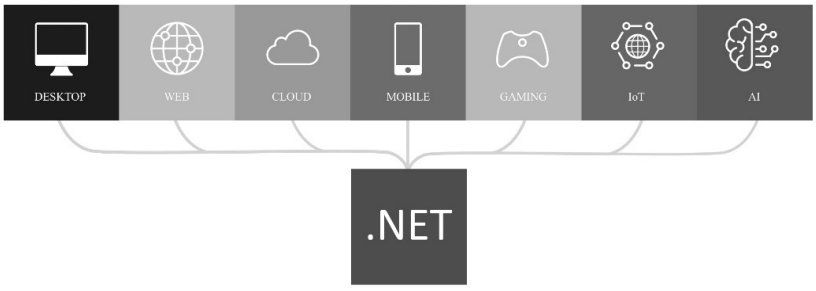

圖 13-1 .NET 平台支援軟體開發類型

❖ .NET Core 與.NET 名詞沿革與混淆

.NET 5 之前的命名是.NET Core 1、2 和 3，但到了.NET 5 時將 Core 的名稱拿掉，未來命名以.NET 6、7、8 方式延續下去，代表未來.NET 平台技術。前述立意雖好，但仍然無法擺脫其他方面混淆，原因是：

- .NET Core 3 之後沒有 4 版，而是 5 版，為避免和.NET Framework 4.x 混淆
- 但 5 版不叫.NET Core 5，而是.NET 5，強調這是未來.NET 實作
- .NET Core 3 世代的 MVC，因有 Core 字眼，故名 ASP.NET Core 3 MVC。那.NET 5 世代的 MVC，因移除 Core 字眼，所以 MVC 叫 ASP.NET 5 MVC？錯！仍叫 ASP.NET Core 5 MVC，目的是為避免和上一代 ASP.NET MVC 5 混淆
- 同樣地，Entity Framework Core 5.0 & 6 仍保留 "Core" 的名稱，以避免與上一代的 Entity Framework 5 和 6 混淆

故由上看來，Core 字眼仍牢牢存在 ASP.NET Core 及 EF Core 中，並未因.NET 移除了 Core 字眼而減少混淆。但本章論述，為同時涵蓋之前.NET Core 1、2、3 版本，故仍會以.NET Core 作為統稱。

❖ .NET Core 特色與賣點

.NET Core 具備眾多特色與賣點，包括：

- 跨平台執行：可在 Windows、Linux 及 macOS 作業系統上執行
- 跨架構一致性：在 x64、x86、ARM、ARM64、x64 Alpine 不同處理器架構執行.NET Core 程式仍保持相同的行為
- 跨平台開發工具：提供 Windows、Linux 及 macOS 作業系統上相對應的 Visual Studio 與 Visual Studio Code（簡稱 VS Code）開發工具，提供優質開發工具
- CLI 命令列工具：提供跨平台 CLI 命令列工具，可用於專案開發及 CI 連續整合的場景應用
- 彈性部署：可包含在您的應用程式中，也可以 side-by-side 並行安裝（用戶或機器範圍內），甚至可配合 Docker 容器使用
- 相容性：.NET Core 透過.NET Standard 與.NET Framework、Xamarin 和 Mono 保持相容性
- 開放原始碼：.NET Core 平台是使用 MIS 和 Apache 2 授權的開放原始碼
- 微軟技術支援：微軟對每個.NET Core 版本提供相對應的支援政策

在這 Highlight 一下，若有以下軟體環境面需求，特別適合使用.NET Core：

- 有跨平台需求
- 針對 Microservice

- 使用 Docker Containers
- 需要高效能及可擴展的系統
- 每個應用程式需要 side-by-side 的.NET 版本

13-2 .NET Core、ASP.NET Core、ASP.NET Core MVC 傻傻分不清

在網路上，對於.NET Core 一詞的意涵，常可看到.NET Core / ASP.NET Core / ASP.NET Core MVC 三種描述，此為同一件事，抑或不同？若您曾開發過任何.NET Framework 應用程式，下圖的對比就很直觀了。

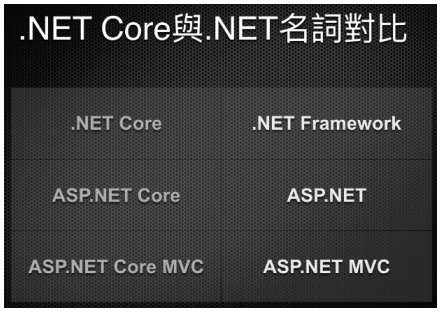

圖 13-2 .NET Core 與.NET 技術名詞對比

.NET Core 的對比就是.NET Framework，ASP.NET Core 對比是 ASP.NET，最後 ASP.NET Core MVC 則對比 ASP.NET MVC，相信應該很好理解。那麼.NET Core 實指什麼技術？.NET Core 係指平台框架，代表最廣泛的技術定義，而 ASP.NET Core 則代表網頁技術，而 ASP.NET Core MVC 則代表 ASP.NET Core 網頁技術中的 MVC 開發框架。

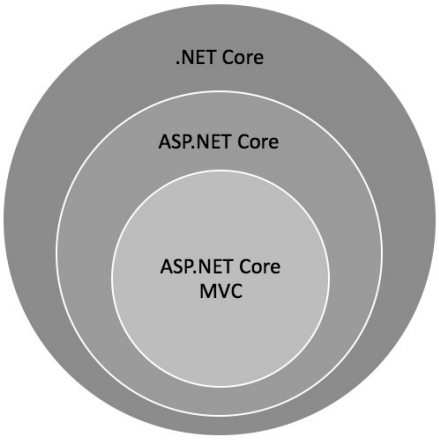

圖 13-3 .NET Core 範圍層級

表 13-2 .NET Core 技術類型之應用程式

技術類型	應用程式類型
.NET Core	ASP.NET Core、Console、WPF、Windows Form、Entity Framework Core 等
ASP.NET Core	MVC、Razor Page、Blazor、Web API、SignalR、gRPC Service、Worker Service
ASP.NET Core MVC	MVC 框架

TIP

.NET Core 3.0 開始支援 WPF 和 Windows Form 應用程式，但僅限於 Windows 平台，而不能跨 macOS 和 Linux。

以.NET Core 書籍來說，也真的有依這三種層級範圍作命名，不同命名方式代表內容聚焦於何種主題，以及技術探討跨越的深廣度。

13-7 建立 ASP.NET Core MVC 資料庫應用程式

本節以 ASP.NET Core 6 MVC 作為示範，故須使用 VS 2022 建立專案，而 VS 2019 最多只能建立 ASP.NET Core 5 MVC，雖說二者皆可進行練習，但 ASP.NET Core 5 MVC 的 DI 是在 Startup.cs 中註冊，而 Startup.cs 檔在 ASP.NET Core 6 MVC 預設的專案樣板中已不復存在，而是整併到 Program.cs 中，須注意這差別。

範例 13-1 以 ASP.NET Core MVC 及 EF Core 建立 Friend 朋友通訊錄程式

在此建立 Friend 朋友通訊錄網頁資料庫程式，包括了典型 MVC 程式：Model 模型、DbContext、種子資料、DI 相依性注入、appsettings.json 的資料庫連線、資料庫 Migration，再到 Controller、Action 及 View 的產生，步驟如下：

step01 建立 ASP.NET Core 6 MVC 專案

在 VS 2022 新增【ASP.NET Core Web 應用程式(Model-View-Controller)】→【下一步】→專案名稱輸入「CoreMvcApp」→架構選擇【.NET 6(長期支援)】→【建立】。

圖 13-10 選擇 ASP.NET Core Web 應用程式(Model-View-Controller)範本

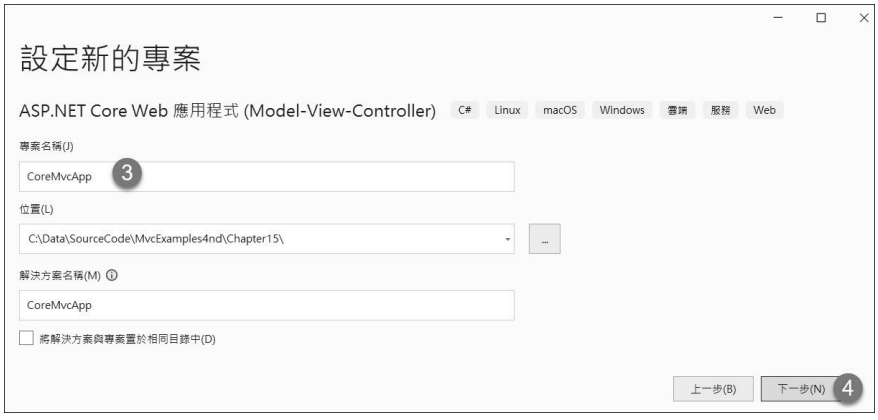

圖 13-11 建立 ASP.NET Core MVC 專案

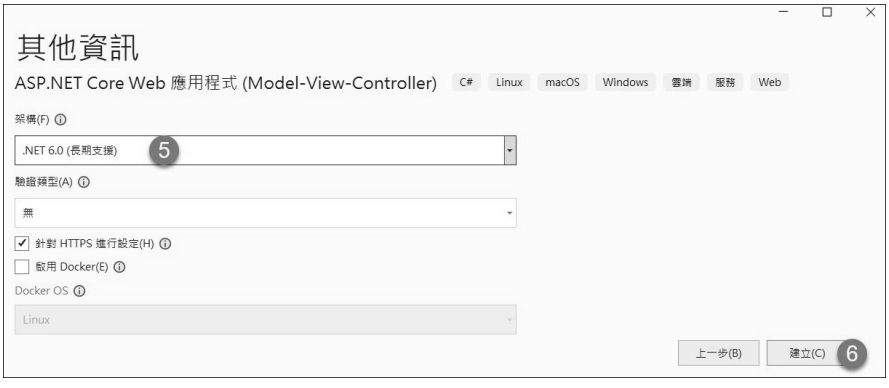

圖 13-12 選擇.NET 6 架構

step02 建立 Friend 資料模型

在 Models 資料夾新增一 Friend.cs 類別檔，並加入以下屬性。

Friend.cs

```
namespace CoreMvcApp.Models
{
    #nullable disable ◄── 將 nullable 設定關閉
    public class Friend
    {
        public int Id { get; set; }
        public string Name { get; set; }
```

```
        public string Email { get; set; }
        public string Mobile { get; set; }
    }
}
```

step03 將專案的 Nullable 設定改為 disable 關閉

在專案按滑鼠右鍵→【編輯專案檔】→將<Nullable>區段設定值改成 disable，目的是為了避免是宣告 Model 模型屬性時所引發的 CS8618 錯誤。

CoreMvcApp.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>disable</Nullable>  ← 改成 disable
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>
  ...
</Project>
```

step04 建立 FriendContext 類別及種子資料

在專案新增 Data 資料夾，加入 FriendContext.cs 類別檔程式與種子資料。

Data/FriendContext.cs

```
global using Microsoft.EntityFrameworkCore;
global using CoreMvcApp.Models;

namespace CoreMvcApp.Data
{
    public class FriendContext : DbContext
    {
        public FriendContext(DbContextOptions<FriendContext> options) :
base(options)
```

```
        {
        }

        public DbSet<Friend> Friends { get; set; }

        //建立種子資料
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
          modelBuilder.Entity<Friend>().ToTable("MyFriend").HasData(
            new Friend { Id=1, Name="Mary", Email="mary@gmail.com",
                  Mobile="0922355822" },
            new Friend { Id=2, Name="David", Email="david@gmail.com",
                  Mobile="0933123456" },
            new Friend { Id=3, Name="Rose", Email="rose@gmail.com",
                  Mobile="0955888163" }
                );
        }
    }
}
```

在 DI 容器中註冊 FriendContext

在 Program.cs 的 DI Container 註冊 FriendContext，如此才能透過 DI 將 FriendContext 實例注入到程式中。

Program.cs

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

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

builder.Services.AddDbContext<FriendContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString
      ("FriendContext")));

var app = builder.Build();
…
```

註冊 FriendContext

讀取資料庫連線設定

step05 在 appsettings.json 新增 FriendContext 資料庫連線設定

上一步驟會抓取 FriendContext 資料庫連線設定，故須在 appsettings.json 中加入。

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      …
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "FriendContext": "Server=(localdb)\\mssqllocaldb;Database=FriendDB;
      Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Developer": {
   …
  }
}
```

加入資料庫連線字串設定

step06 以 Migration 產生 FriendDB 資料庫及種子資料

在【工具】→【NuGet 套件管理員】→【套件管理器主控台】執行以下兩道命令，以 Migration 產生 FriendDB 資料庫及種子資料。

```
Add-Migration InitialDB
Update-Database
```

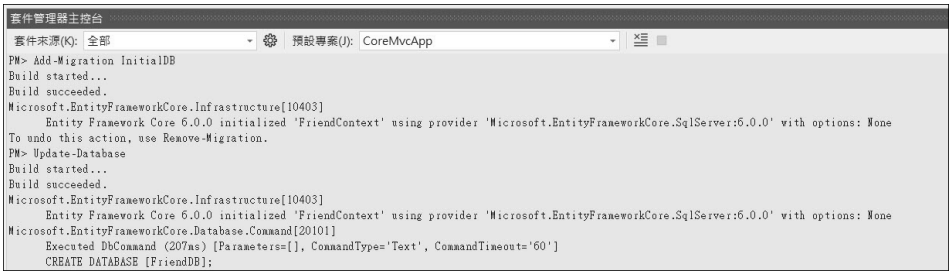

圖 13-13 以 Migration 產生 FriendDB 資料庫及種子資料

step07 以 SSMS 檢視 FriendDB 資料庫及種子資料

上一步驟若正確完成，則在 SQL Server 中可看見 FriendDB 資料庫，在 Friends 資料表中有三筆種子資料，後續會用程式讀取並顯示。

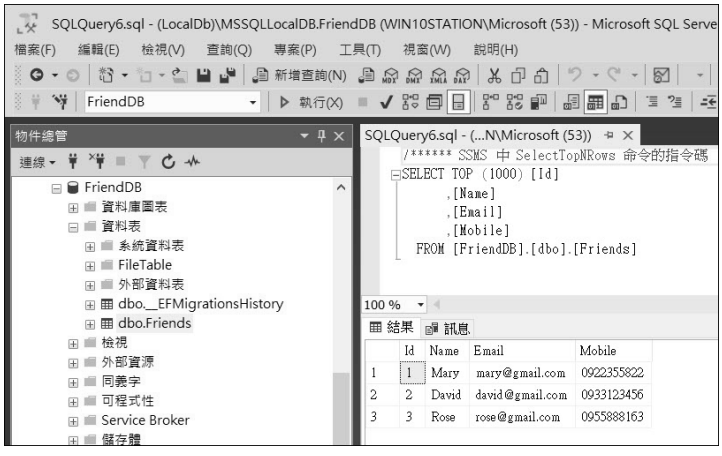

圖 13-14 以 SSMS 檢視 FriendDB 資料庫及種子資料

step08 建立 Friends 控制器

在 Controllers 資料夾新增 Friends 控制器，並建立以下程式。

```
using Microsoft.AspNetCore.Mvc;

namespace CoreMvcApp.Controllers
{
```

```
public class FriendsController : Controller
{
    private readonly FriendContext _context;

    public FriendsController(FriendContext context)
    {
        _context = context;
    }

    public async Task<IActionResult> Index()
    {
        List<Friend> friends = await _context.Friends.ToListAsync();

        return View(friends);
    }
}
}
```

以 DI 注入 FriendContext

以 FriendContext 讀取 Friends 資料表

step09 從 Index()產生 Index.cshtml 檢視

在 Index()按滑鼠右鍵→【新增檢視】→【Razor 檢視】→範本選擇【List】→模型類別選擇【Friend】→【新增】建立檢視。

圖 13-15 新增 Index 檢視

step 10 按 F5 執行並瀏覽 Friends/Index 網址

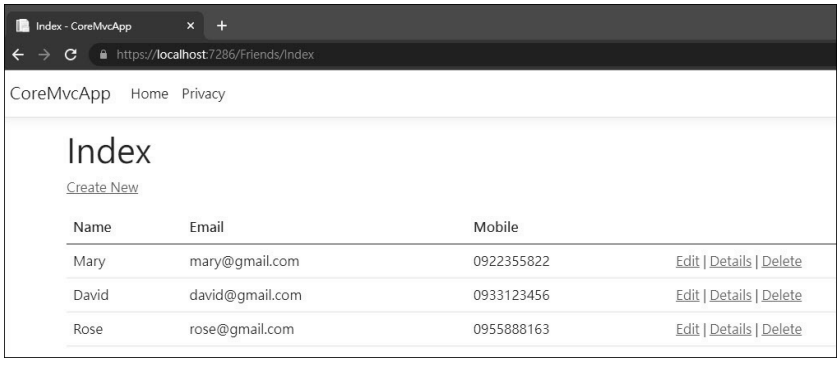

圖 13-16 瀏覽 Friends/Index 網址資料

13-8 結論

本章闡述了.NET Core 技術之意涵，並揭櫫組成平台功能的各項技術區塊，在這些新技術堆疊的協助下，開創了跨平台開發與執行之新局面。最後藉由一個 MVC 資料庫範例程式，勾勒出建立 ASP.NET Core 完整過程，讓您了解與比較它與上一代 ASP.NET MVC 5 二者間的異同。