

ASP.NET Core 5 開發實戰

基礎知識篇

多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

<https://blog.miniasp.com>



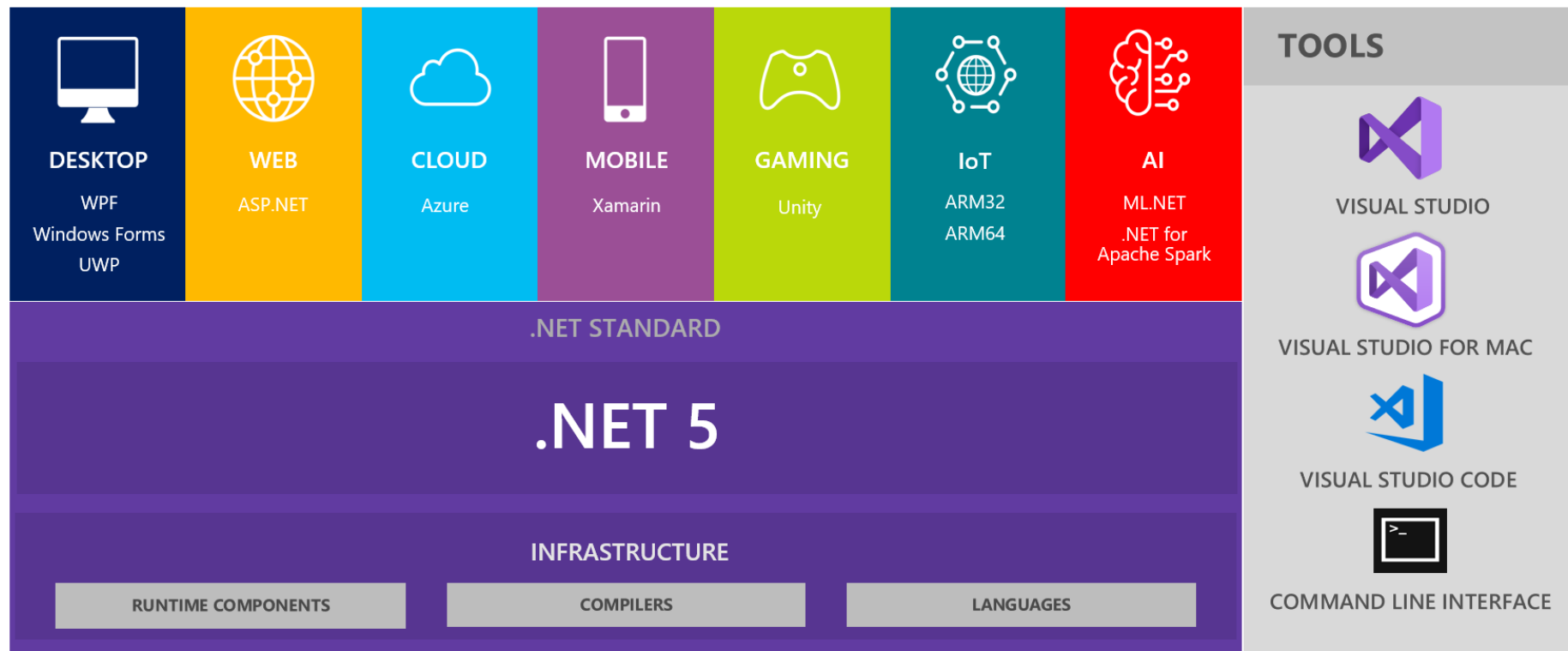


重新認識 .NET 生態系

目前三種不同的 .NET 實作



.NET - A unified platform



<https://devblogs.microsoft.com/dotnet/introducing-net-5/>

.NET Framework / .NET Core / .NET

- **.NET Framework**

- 僅支援 Windows 作業系統
- 歷史悠久、工具完整、豐富的生態圈

- **.NET Core**

- 跨平台 (Windows, Linux, macOS)
- 於 Windows 平台可自由取用 .NET Framework 的基礎類別庫 (BCL)

- **.NET**

- 跨平台 (Windows, Linux, macOS)
- 於 Windows 無法取用 .NET Framework 的基礎類別庫 (BCL)
- 但有提供 [.NET Upgrade Assistant](#) 自動升級工具

.NET Framework 技術架構

VB

C#

F#

通用語言規格 (CLS) (Common Language Specification)

通用型別系統 (CTS) (Common Type System)

.NET 框架類別庫 (FCL) (Framework Class Library)

.NET 基礎類別庫 (BCL) (Base Class Library)

通用語言執行階段 (CLR) (Common Language Runtime)

通用語言基礎結構 (CLI) (Common Language Infrastructure)

.NET 基礎類別庫 (BCL)

XML

XLinq / XML Document / XPath / XSD / XSL

SERIALIZATION

BinaryFormatter / Data Contract / XML

NETWORKING

Sockets / Http / Mail / WebSockets

IO

Files / Compression / MMF

THREADING

Threads / Thread Pool / Tasks

CORE

Primitives / Collections / Reflection / Interop / Linq

簡介 .NET Standard 標準類別庫

- .NET Standard 是一個**標準規格**
 - 定義出**所有 .NET 平台**需要支援的 API 有哪些
 - 所有 .NET 平台的應用程式，皆可參考 .NET Standard 標準類別庫
- .NET Standard **不包含任何實作**，只有**定義介面合約**
 - 不同的 .NET 平台，被要求必須實作 **.NET Standard 規格**中定義的 APIs
- .NET Standard 透過 [NETStandard.Library](#) 中繼套件來載入類別庫
 - **NETStandard.Library** 中繼套件透過 **netstandard.dll** 組件來連結相依的組件
 - **netstandard.dll** **沒有任何 API 實作**，僅提供 Runtime 知道有哪些 API 可用
- **從 .NET 5 開始，已經沒有 .NET Standard 了！**
 - 因為 .NET 5 不再支援 .NET Framework
 - 而且 .NET 5 保有 .NET Standard 的所有特性

建立 .NET 類別庫專案

- VS2019
 - [檔案] → [新增] → [專案] → [類別庫]
- VSCode
 - **dotnet new classlib -n lib1**
- 關於 .NET 類別庫 NuGet 套件
 - NuGet 套件本地快取路徑：**dotnet nuget locals all -l %USERPROFILE%\nuget\packages**
 - 清空 NuGet 套件本地快取：**dotnet nuget locals all -c**



簡介 .NET Core 平台

關於 .NET Core 重要特性

- **跨平台與多架構** (作業系統 & CPU 架構)
 - **Windows**、**macOS** 及 **Linux** 上都可執行，也可移轉到其他作業系統。
 - 在多個架構上 (包括 **x64**、**x86** 及 **ARM**) 可使用相同的行為來執行程式碼。
- **命令列工具** (CLI)
 - 所有使用 .NET Core 開發的情境都可以在命令列操作。
- **彈性的部署**
 - 可以內嵌在現有應用程式內、可以並行安裝、可以部署到 Docker 容器中。
- **平台相容性**
 - **.NET Core** 透過 **.NET Standard** 標準類別庫與其他平台相容
 - 例如 **.NET Framework**、**Xamarin** 及 **Mono** 等等。
- **開放原始碼**
 - .NET Core 平台是[開放原始碼](#)，使用 MIT 和 Apache 2 授權。
- **受 Microsoft 支援**
 - 每個版本的 .NET Core 皆由 Microsoft 官方提供技術支援。

選擇 .NET Core 或 .NET Framework

- 針對使用 .NET Core

- 你有**跨平台**的需求
- 你想採用 **微服務** 架構進行部署
- 你正在使用 **Docker** 等容器技術
- 你需要 **高效能、可延展** 的應用系統
- 你需要讓應用程式間可以**並存不同的 NET 版本** (降低相依性)

- 針對使用 .NET Framework

- 你**目前使用** .NET Framework 應用程式沒壞掉 (也不想改)
- 你的應用程式**找不到**可用的 .NET Core 第三方函式庫或套件
- 你需要使用的 .NET 技術**不適用** .NET Core (例如: Web Form)
- 你的應用程式必須使用**不支援** .NET Core 的平台

針對伺服器應用程式在 .NET Core/5+ 和 .NET Framework 之間進行選擇

了解 .NET Core 的組成項目

- 一個 .NET 執行階段 (Runtime)

提供型別系統、組件載入、GC、原生 Interop 及其他基本服務。

- 一組 Framework 函式庫 (Framework Libraries)

提供**基礎(primitive)**資料型別與一些好用的應用程式類別與工具類別。

- 一套 SDK 工具組及語言編譯器

提供軟體開發過程所需的基礎工具組，並可透過 .NET Core SDK 取得。

- 一支 dotnet 主程式 (app host)

它主要用來啟動與執行 .NET Core 應用程式，過程中它會自動選取正確的 Runtime，並將 Runtime 裝載到記憶體內，提供 DLL 組件的載入原則，然後啟動應用程式執行。

認識 .NET Core 版本設定

- .NET Core Runtime 遵循 [SemVer 語意化版本](#) 設定
- .NET Core SDK 不遵循 [SemVer 語意化版本](#) 設定
 - SDK 版本號碼的**第三個位置**同時傳達**次要版本**與**修補版本**號碼。
 - 每次發行 .NET Core SDK **次要版本**都會被乘以 100
 - 例如：次要版本 1、修補版本 2 將會以 102 表示

變更	.NET Core 執行階段	.NET Core SDK (*)
初始版本	2.2.0	2.2.100
SDK 修補程式	2.2.0	2.2.101
執行階段與 SDK 修補程式	2.2.1	2.2.102
SDK 功能變更	2.2.1	2.2.200

目前的 .NET Runtime 版本

^ Supported versions

Version	Status	Latest release	Latest release date	End of support
.NET 6.0	Preview ⓘ	6.0.0-preview.3	April 08, 2021	
.NET 5.0 (recommended)	Current ⓘ	5.0.6	May 11, 2021	
.NET Core 3.1	LTS ⓘ	3.1.15	May 11, 2021	December 03, 2022
.NET Core 2.1	LTS ⓘ	2.1.28	May 11, 2021	August 21, 2021

^ Out of support versions

The following releases have reached end of life, meaning they're no longer supported. We recommend moving to a supported release.

Version	Latest release	Latest release date	End of support
.NET Core 3.0	3.0.3	February 18, 2020	March 03, 2020
.NET Core 2.2	2.2.8	November 19, 2019	December 23, 2019
.NET Core 2.0	2.0.9	July 10, 2018	October 01, 2018
.NET Core 1.1	1.1.13	May 14, 2019	June 27, 2019
.NET Core 1.0	1.0.16	May 14, 2019	June 27, 2019

<https://dotnet.microsoft.com/download/dotnet-core>

透過 global.json 指定 .NET Core SDK 版本

- 列出所有已安裝 SDK 版本
 - `dotnet --list-sdks`
- 快速建立 global.json 檔案
 - `dotnet new globaljson`
- 列出所有已安裝的 Runtime 版本
 - `dotnet --list-runtimes`
- 快速建立 global.json 檔案並指定特定 SDK 版本
 - `dotnet new globaljson --sdk-version 3.1.409`
- [如何在多個 .NET Core SDK 版本之間進行切換 \(global.json\)](#)



.NET Core 新手上路

安裝 .NET Core SDK & Runtime

- <https://dot.net/> 或 <https://dotnet.microsoft.com/download>
 - 建置應用程式：**.NET Core SDK**
 - 執行應用程式：**.NET Core Runtime**
 - 所有版本下載：[All .NET Core downloads](#)
- Docker 容器映像 (Docker Hub)
 - [dotnet/core](#): .NET Core
 - [dotnet/core/sdk](#): .NET Core SDK
 - [dotnet/core/aspnet](#): ASP.NET Core Runtime
 - [dotnet/core/runtime](#): .NET Core Runtime
 - [dotnet/core/runtime-deps](#): .NET Core Runtime Dependencies
 - [dotnet/core/samples](#): .NET Core Samples

認識 .NET Core SDK CLI 命令列工具

- `dotnet help` 顯示執行與 SDK 命令說明
- `dotnet --version` 顯示目前**正在使用的** SDK 版本
- `dotnet --list-runtimes` 顯示**已安裝的** .NET Core Runtimes 版本
- `dotnet --list-sdks` 顯示**已安裝的** .NET Core SDK 版本
- `dotnet --info` 顯示完整的 .NET Core 版本資訊
- 查詢 [dotnet command](#) 命令參考

認識 .NET Core SDK 基本 CLI 命令

- `dotnet new` 初始化 C# 或 F# 範本**專案**或**檔案**
- `dotnet restore` 還原所指定應用程式的相依性
- `dotnet build` 建置 .NET Core 應用程式 (會自動還原套件)
- `dotnet clean` 清除建置時所輸出的相關檔案
- `dotnet msbuild` 執行 MSBuild 命令
- `dotnet run` 從專案目錄直接執行應用程式 (會自動建置)
- `dotnet test` 使用專案指定的**測試執行器**執行**單元測試**

透過 .NET CLI 設定專案與 NuGet 參考

- `dotnet list reference` 列出使用中的**專案參考**
- `dotnet add reference` 新增**專案參考**
- `dotnet remove reference` 移除**專案參考**

- `dotnet list package` 列出使用中的 **NuGet** 套件
- `dotnet add package` 新增 **NuGet** 套件
- `dotnet remove package` 移除 **NuGet** 套件

- `dotnet nuget locals` 列出或清除本機 NuGet 套件快取
- `dotnet nuget delete` 從伺服器刪除或取消列出套件
- `dotnet nuget push` 將套件推送至伺服器並發行

透過 .NET CLI 封裝與發行

- `dotnet pack` 替目前類別庫專案建立 NuGet 套件
- `dotnet publish` 發行 .NET Core 應用程式
 - `dotnet publish -c Release`
 - `dotnet publish -c Release /p:UseAppHost=false`
 - `dotnet publish -r win10-x64`
 - `dotnet publish -r win10-x64 --self-contained`
 - `dotnet publish -r win10-x64 --no-self-contained`
 - `dotnet publish -r win10-x64 /p:PublishSingleFile=true`
 - `dotnet publish -r win10-x64 /p:PublishTrimmed=true`

用 .NET CLI 建立 .NET Core 主控台專案

- 先建立並進入一個空白資料夾
- `mkdir solution1`
- `cd solution1`
- `dotnet new -l`
- `dotnet new console -o console1`
- `cd console1`
- 編輯 Program.cs (請修改 Main() 方法的程式碼)
- 查看 console1.csproj (請注意 TargetFramework 屬性)
- `dotnet restore` (非必要步驟)
- `dotnet build`
- `dotnet run`
- `dotnet clean`
- `dotnet publish -c Release`
- `dotnet bin\Release\netcoreapp3.1\console1.dll`

用 .NET CLI 建立 .NET 類別庫專案

- `cd ..` (回到 solution1 資料夾)
- `dotnet new classlib -o classlib1`
- `cd classlib1`
- 編輯 [Class1.cs](#) 任意原始碼 (請新增一個 `Go()` 方法並回傳字串)
- 查看 `classlib1.csproj` (請注意 `TargetFramework` 屬性)
- `dotnet build`
- `dotnet build -c Release`
- `dotnet msbuild /p:Configuration=Release`
- `dotnet clean`
- `dotnet pack`
- `dotnet publish`
- `dotnet publish -c Release`
- `dotnet msbuild /t:Publish /p:Configuration=Release`

用 .NET CLI 建立 .NET Core 單元測試

- `cd ..` (回到 solution1 資料夾)
- `dotnet new mstest -n classlib1.test`
- `cd classlib1.test`
- `dotnet add reference ../classlib1/classlib1.csproj`
- 編輯 [UnitTest1.cs](#) 單元測試檔 (`UnitTest1-mstest.cs`)
- 查看 `classlib1.test.csproj` (請注意 `TargetFramework` 屬性)
- `dotnet test -t` (列出專案中所有測試案例)
- `dotnet test` (執行測試)
- `dotnet build`

用 .NET CLI 管理專案參考與 NuGet 套件

- `cd ../console1/` (回到 console1 資料夾)
- `dotnet add reference ../classlib1/classlib1.csproj`
- `dotnet list reference`
- `dotnet remove reference ../classlib1/classlib1.csproj`
- `dotnet add package Newtonsoft.Json`
- `dotnet list package`
- `dotnet remove package Newtonsoft.Json`
- 你可以透過任意分享資料夾來安裝公司內部的 NuGet 套件
`dotnet add package -s \\MyServer\NuGet mylib`
- `dotnet nuget locals all --list`

透過 .NET CLI 建立方案檔

- `dotnet sln` 建立方案檔中新增、移除及列出專案的選項

- `cd ..` (回到 solution1 資料夾)
- `dotnet new sln`
- `dotnet sln solution1.sln add console1\console1.csproj`
- `dotnet sln solution1.sln add classlib1\classlib1.csproj`
- `dotnet sln solution1.sln add **/*.csproj` (Linux/macOS only)
- `dotnet restore solution1.sln`
- `dotnet build solution1.sln`
- `dotnet clean solution1.sln`
- `dotnet publish solution1.sln`
- `dotnet pack solution1.sln`

使用 VS 2019 建立 .NET Core 專案

- 建立 **Solution1** 空白方案
- 從方案建立 **MyLibrary** 類別庫(.NET Standard)專案
 - 設計 Sum 類別與方法
- 從方案建立 **MyLibrary.Test** MSTest 測試專案 (.NET Core)
 - 在單元測試專案加入參考 MyLibrary 專案
 - 設計測試類別 UnitTest1.cs
 - 使用 測試總管 (Test Explorer) 執行單元測試
- 從方案建立 **MyConsole** 主控台應用程式 (.NET Core) 專案
 - 在主控台應用程式專案加入參考 MyLibrary 專案
 - 設計主控台應用程式
 - 執行看結果



理解 .NET Standard 標準類別庫



.NET Standard 是定義一個標準介面
讓你可以透過此標準介面
在不同平台下皆可以使用一致 BCL 資源



為什麼要用 .NET Standard 標準類別庫

- 可以設計出**可重複使用**的類別庫
- 可以用於**不同的 .NET Framework 平台**
- 以「**套件**」(NuGet) 為單位進行共用
 - 一個套件可同時設定許多目標框架 (Target Framework)
- 徹底解決 **PCL 可攜式類別庫**的缺點與問題
 - PCL 可用的 API 是所支援 .NET 架構的 API 交集
 - PCL 不同 Profile 的類別庫就無法互相參考
- 它是 .NET 世界第一個針對跨平台所訂出的 **BCL 標準**！

使用 .NET Standard 可以解決哪些問題

- 每個 .NET 架構平台下，皆可以參考這個類別庫組件
 - 架構平台 ➔ .NET Framework, .NET Core, Xamarin, ...
 - 這個組件具有**跨作業系統平台**的特性
- 方便管理與維護
 - 使用 **NuGet** 來發佈這些套件
- 解決 PCL 不方便使用的問題（限制太多）
- 可以針對不同平台架構撰寫平台專屬的 API
- 如果你沒有**多架構平台**的需求
 - ➔ 就可能不需要 .NET Standard 這個解決方案

.NET Standard 定義 API

- dotnet/standard 中的 src/netstandard/ref 目錄有定義 APIs
 - 但只有 API 合約定義，沒有 API 實作
- 例如：[List<T>](#)

```
360     public partial class List<T> : System.Collections.Generic.ICollection<T>, System.Collect
361     {
362         public List() { }
363         public List(System.Collections.Generic.IEnumerable<T> collection) { }
364         public List(int capacity) { }
365         public int Capacity { get { throw null; } set { } }
366         public int Count { get { throw null; } }
367         public T this[int index] { get { throw null; } set { } }
368         bool System.Collections.Generic.ICollection<T>.IsReadOnly { get { throw null; } }
369         bool System.Collections.ICollection.IsSynchronized { get { throw null; } }
370         object System.Collections.ICollection.SyncRoot { get { throw null; } }
371         bool System.Collections.IList.IsFixedSize { get { throw null; } }
372         bool System.Collections.IList.IsReadOnly { get { throw null; } }
```

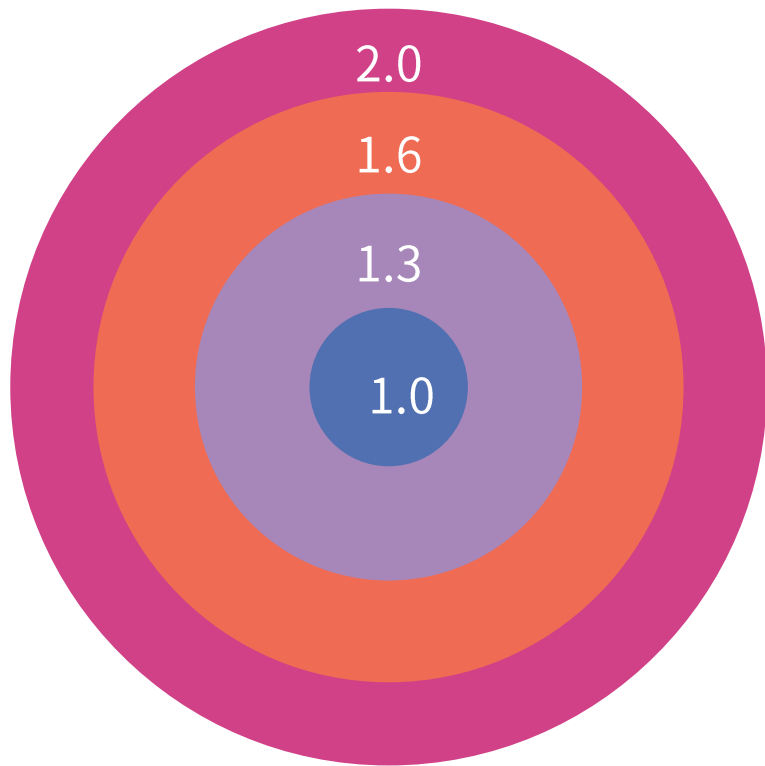
每個 .NET Standard 版本的最低平台版本

下表列出支援每個 .NET Standard 版本的最低平台版本。這表示所列出平台的更新版本也支援所對應 .NET Standard 版本。
例如，.NET Core 2.2 支援 .NET Standard 2.0 及更早版本。

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework ¹	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 ²	4.6.1 ²	4.6.1 ²	不適用 ³
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	10.0
通用 Windows 平台	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299	TBD
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	TBD

<https://docs.microsoft.com/zh-tw/dotnet/standard/net-standard>

該選用哪個 .NET Standard 版本



- 新版本將會包含所有舊版本所有 API
 - 版本標示方式：X.Y
 - 版本更新**只會新增** API **不會刪除**舊的
- **版本越新，可使用的 API 就越多**
- **版本越舊，可支援的 .NET 平台就越多**
- 反之，較高版本支援較多的 API，但支援較少的.NET開發環境

How .NET Standard relates to .NET Platforms

.NET Standard 版本控制規則

- 新版本的 API 只增不減 (Additive)

.NET Standard 版本就邏輯而言是同心圓，較新的版本會包含來自較舊版本的所有 API，版本之間不會有任何重大變更。

- 舊版本的 API 絕不改變 (Immutable)

.NET Standard 只要在確認交付新版之後，版本就會凍結，未來也不會有任何改變或更新。

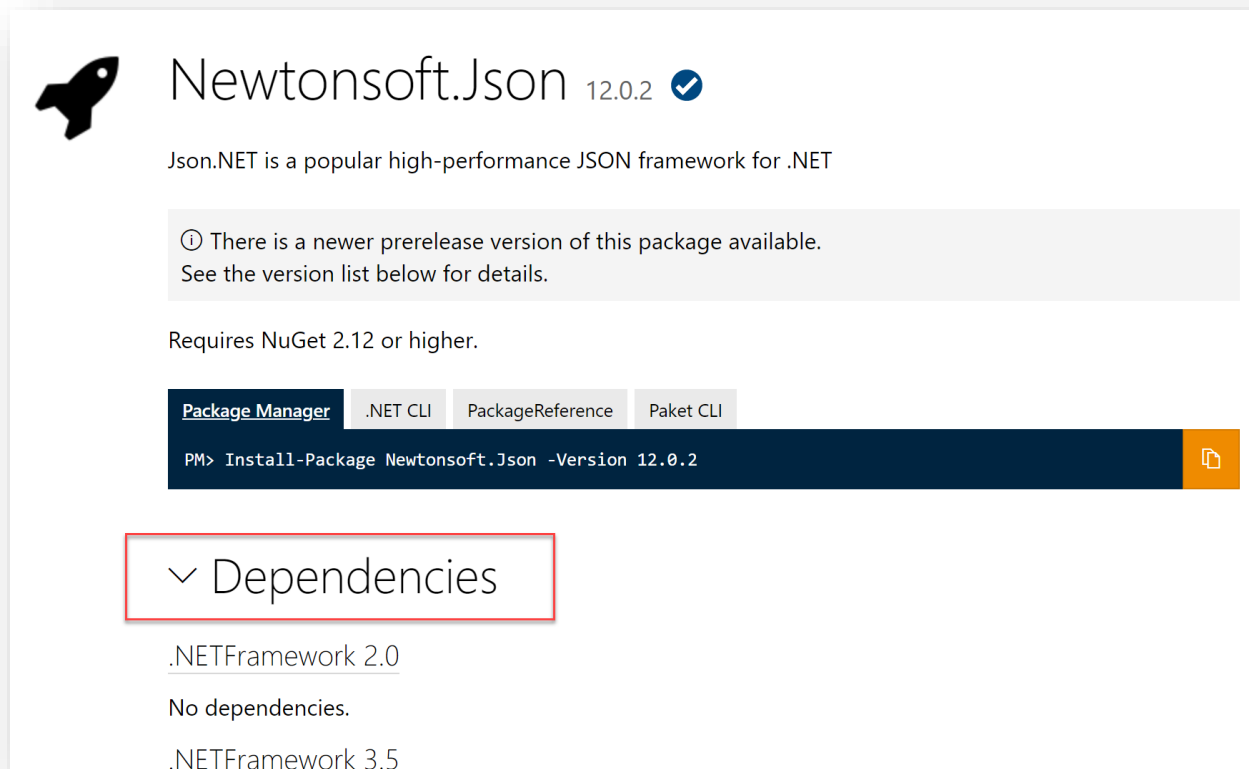
新版本的 API 通常會先在特定 .NET 平台進行實作 (例如 .NET Core)，如果 .NET Standard 審查委員會認為這些新的 API 應該在所有平台都提供，就會在下一個 .NET Standard 版本中新增這些 API。



.NET Standard 不同版本支援那些 API

- .NET Standard 各版本的 APIs 清單
 - <https://github.com/dotnet/standard/tree/master/docs/versions>
 - 上述文件包含**每一版與前一版本的差異比較**
 - .NET Standard 透過 [NETStandard.Library](#) 中繼套件發佈
- 可利用 [.NET API Browser](#) 或 [.NET API Catalog](#) 查詢可用 APIs
- 介紹好用工具：[.NET Portability Analyzer](#)
 - <https://github.com/Microsoft/dotnet-apiport#documentation>
 - [Brief Look at the .NET Portability Analyzer](#)

判斷 NuGet 套件是否支援 .NET Standard

- 以 [Newtonsoft.Json](#) 為例 (檢查是否有 **Dependencies** 資訊)



 **Newtonsoft.Json** 12.0.2 

Json.NET is a popular high-performance JSON framework for .NET

ⓘ There is a newer prerelease version of this package available.
See the version list below for details.

Requires NuGet 2.12 or higher.

Package Manager .NET CLI PackageReference Paket CLI

PM> Install-Package Newtonsoft.Json -Version 12.0.2

∨ **Dependencies**

.NETFramework 2.0

No dependencies.

.NETFramework 3.5

練習：升級 .NET Framework 類別庫專案

- 請下載以下 **.NET Framework 4.6.1** 專案
<https://github.com/coolrare/SimpleNeuralNetworkInCSharp>
- 將此專案升級為 .NET Standard 2.0 與 .NET 5 單元測試專案
 - NeuralNetworkCSharp (類別庫專案)
 - NeuralNetworkCSharpTests (單元測試專案)
- 升級步驟
 1. 建立新專案
 2. 搬動檔案
 3. 修正套件版本
 4. 調整少量程式碼

請多利用 [.NET Upgrade Assistant](#) 工具升級專案



了解 FDD 與 SCD 部署方式

.NET Core 應用程式部署方法

.NET Core 應用程式部署類型

- [Framework 相依部署 \(FDD\)](#) (Framework-Dependent Deployments)
 - 目標系統必須先安裝共用的 .NET Core Runtime 版本 ([Downloads](#))
- [Framework 相依可執行檔 \(FDE\)](#) (Framework-Dependent Executables)
 - 目標系統必須先安裝共用的 .NET Core Runtime 版本 ([Downloads](#))
 - 這種部署類型從 .NET Core 2.2 才開始支援
 - 這種部署類型從 **.NET Core 3.0** 開始為**預設值**
 - 可直接部署目標平台的**可執行檔**，無須透過 **dotnet** 公用程式即可執行
- [自封式部署 \(SCD\)](#) (Self-Contained Deployments)
 - 不仰賴任何存在於目標系統上的共用元件，但是部署檔案相當大
 - 包括 .NET Core 程式庫和 .NET Core 執行階段的所有元件，都隨附於應用程式，並與其他 .NET Core 應用程式隔離

認識 RID 執行階段識別項

- RID (Runtime Identifier) 可透過 **dotnet --info** 查詢！
- RID 用來識別一個 .NET Core 應用程式應該執行在哪個平台
- RID 基本格式
 - [os].[version]-[architecture]-[additional qualifiers]
- 常見的 RID 範例 ([.NET Core RID Catalog](#))
 - win-x64 win10-x64 win10-x86
 - linux-x64 linux-arm ubuntu.18.04-x64
 - osx-x64 osx.10.12-x64 osx.10.14-x64

使用 RID 的注意事項

- RID 是**隱晦字串** (opaque strings)，因此必須以黑箱視之。
- **不要**以**程式設計方式**建置 RID
- 使用已針對平台定義的 RID
- RID 必須是**特定的**，因此不要假設實際 RID 值會怎樣！

練習：發佈 .NET Core 應用程式

- 使用 CLI 發佈 .NET Core 應用程式

- **Framework 相依部署 (FDD)**

- ```
dotnet publish -c Release -p:UseAppHost=false
```

- **Framework 相依可執行檔 (FDE)**

- ```
dotnet publish -c Release -r <RID> --no-self-contained
```

- **自封式部署 (SCD)**

- ```
dotnet publish -c Release -r <RID> --self-contained
```

- ```
dotnet publish -c Release -r <RID> -p:PublishSingleFile=true
```

- ```
dotnet publish -c Release -r <RID> -p:PublishTrimmed=true
```

- 使用 Visual Studio 部署 .NET Core 應用程式

- [方案總管] → 在專案節點按下滑鼠右鍵 → [發行]



# 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 保哥的技术交流中心

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

Twitter

[https://twitter.com/Will\\_Huang](https://twitter.com/Will_Huang)



多奇·教育訓練

**THANK YOU!**

Q&A