


教學課程：使用 Visual Studio Code 建立 .NET 主控台應用程式

發行項 • 2021/12/04 • 

此頁面有所助益嗎？ 

選擇 .NET 版本

.NET 6	.NET 5	.NET Core 3.1
--------	--------	---------------

本文內容

[必要條件](#)

[建立應用程式](#)

[執行應用程式](#)

[增強應用程式](#)

[其他資源](#)

[下一步](#)

本教學課程說明如何使用 Visual Studio Code 和 .net CLI 來建立和執行 .net 主控台應用程式。Project 工作（例如建立、編譯和執行專案）是使用 .net CLI 來完成。您可以使用不同的程式碼編輯器來進行本教學課程，並在您想要的情況下在終端機中執行命令。

必要條件

- 已安裝 [c # 擴充](#) 功能的 [Visual Studio Code](#) 。如需有關如何在 Visual Studio Code 上安裝擴充功能的詳細資訊，請參閱 [VS Code 擴充功能 Marketplace](#) 。
- [.Net 6 SDK](#) 。

建立應用程式

建立名為 "HelloWorld" 的 .NET 主控台應用程式專案。

1. 啟動 Visual Studio Code 。
2. 從主功能表選取 [檔案 開啟資料夾 (檔案開啟 > ...) macOS) 。

3. 在 [開啟資料夾] 對話方塊中，建立 *HelloWorld* 資料夾並加以選取。然後按一下 [選取資料夾 (在 MacOS) 開啟]。

資料夾名稱預設會變成專案名稱和命名空間名稱。您稍後會在教學課程中假設專案命名空間為的程式碼中加入程式碼 `HelloWorld`。

4. 在 [您信任此資料夾中檔案的作者嗎?] 對話方塊中，選取 [是，我信任作者]。
5. 從主功能表中選取 [View terminal]，以在 Visual Studio Code 中開啟 終端 機。

終端 機會在 *HelloWorld* 資料夾中使用命令提示字元開啟。

6. 在 終端 機中，輸入下列命令：

.NET CLI	 複製
<pre>dotnet new console --framework net6.0</pre>	

專案範本會建立簡單的應用程式，藉由呼叫 `Program` 中的方法，在主控台視窗中顯示「Hello World」`Console.WriteLine(String)`。

C#	 複製
<pre>Console.WriteLine("Hello, World!");</pre>	

7. 使用下列程式碼取代 *Program.cs* 的內容：

C#	 複製
<pre>namespace HelloWorld { class Program { static void Main(string[] args) { Console.WriteLine("Hello World!"); } } }</pre>	

當您第一次編輯 `.cs` 檔案時，Visual Studio Code 會提示您新增遺失的資產，以建立和偵測應用程式。選取 [是]，Visual Studio Code 使用 啟動的 *json* 和工作。 *json* 檔案建立 *vscode* 資料夾。

`Program` 程式碼會使用單一方法定義類別，`Main` 這個方法會採用 `String` 陣列做為引數。`Main` 是應用程式進入點，是執行階段在啟動應用程式時會自動呼叫的方法。在應用程式啟動時所提供的所有命令列引數，都會在 `args` 陣列中提供。

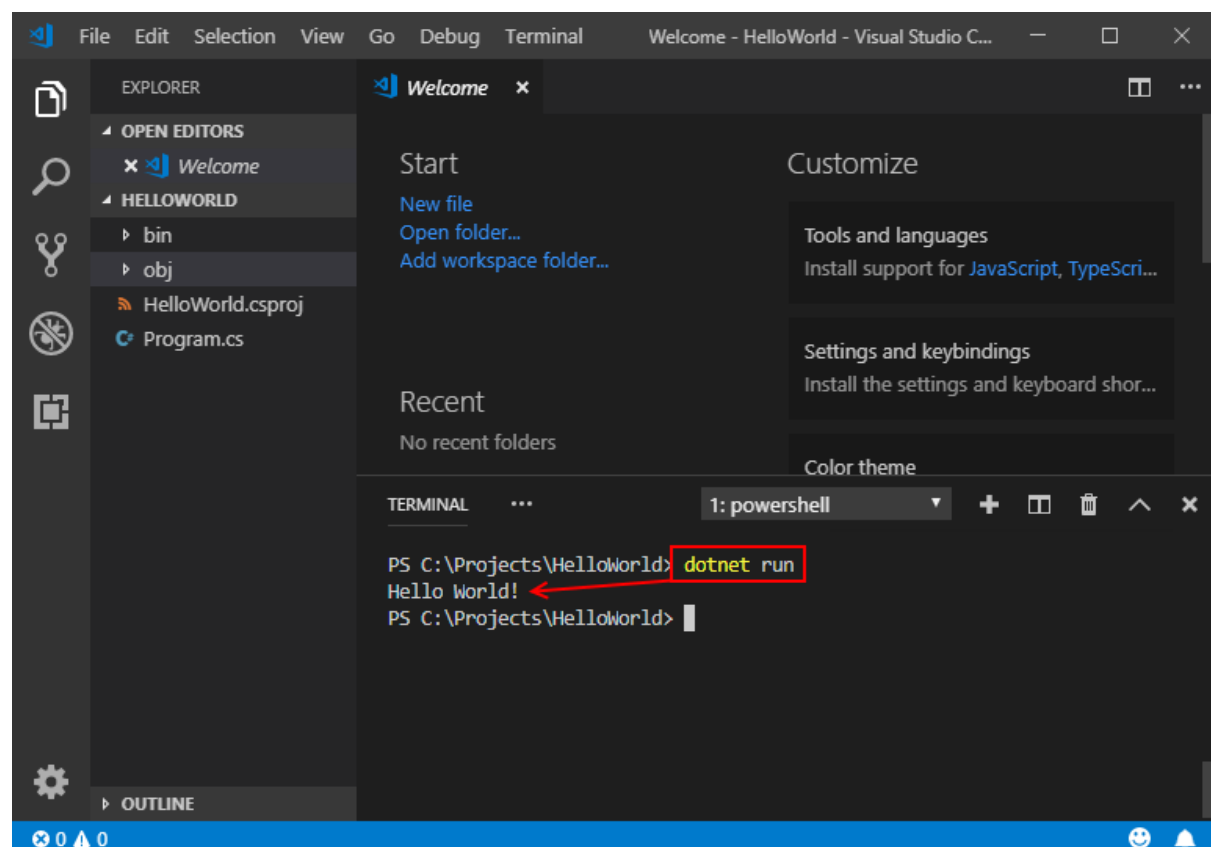
在最新版的 `C#` 中，名為 [最上層語句](#) 的新功能可讓您省略 `Program` 類別和 `Main` 方法。大部分現有的 `C#` 程式都不會使用最上層的語句，因此本教學課程不會使用這項新功能。但在 `C# 10` 中也有提供，無論您是在程式中使用它，都是使用樣式喜好設定。

執行應用程式

在終端機中執行下列命令：

.NET CLI	複製
<pre>dotnet run</pre>	

程式會顯示 "Hello World !" 並結束。




增強應用程式

增強應用程式以提示使用者輸入其名稱，並將它與日期和時間一起顯示。

1. 開啟 *Program.cs* 。
2. Main 以下列 程式 代碼取代 Program 中方法的內容，也就是呼叫的程式 `Console.WriteLine` 程式碼：

C#	 複製
<pre>Console.WriteLine("What is your name?"); var name = Console.ReadLine(); var currentDate = DateTime.Now; Console.WriteLine(\$"{Environment.NewLine}Hello, {name}, on {currentDate:d} at {currentDate:t}!"); Console.Write(\$"{Environment.NewLine}Press any key to exit..."); Console.ReadKey(true);</pre>	

這段程式碼會在主控台視窗中顯示提示，並等候使用者輸入字串，然後按  `Enter` 鍵。它會將此字串儲存在名為的變數中 `name`。此程式碼也會擷取 `DateTime.Now` 屬性的值，其中包含目前的當地時間，並將它指派至名為 `currentDate` 的變數。它會在主控台視窗中顯示這些值。最後，它會在主控台視窗中顯示提示，並呼叫 `Console.ReadKey(Boolean)` 方法來等候使用者輸入。

`NewLine` 是與平臺無關且與語言無關的方式，表示分行符號。替代方案是 `\n` 在 `c#` 和 `vbCrLf` Visual Basic 中。


字串前面的貨幣符號 (`$`) 可讓您在字串中將運算式（例如變數名稱）放在大括弧中。運算式值會插入字串以取代運算式。這個語法稱為 **插補字串**。

3. 儲存您的變更。

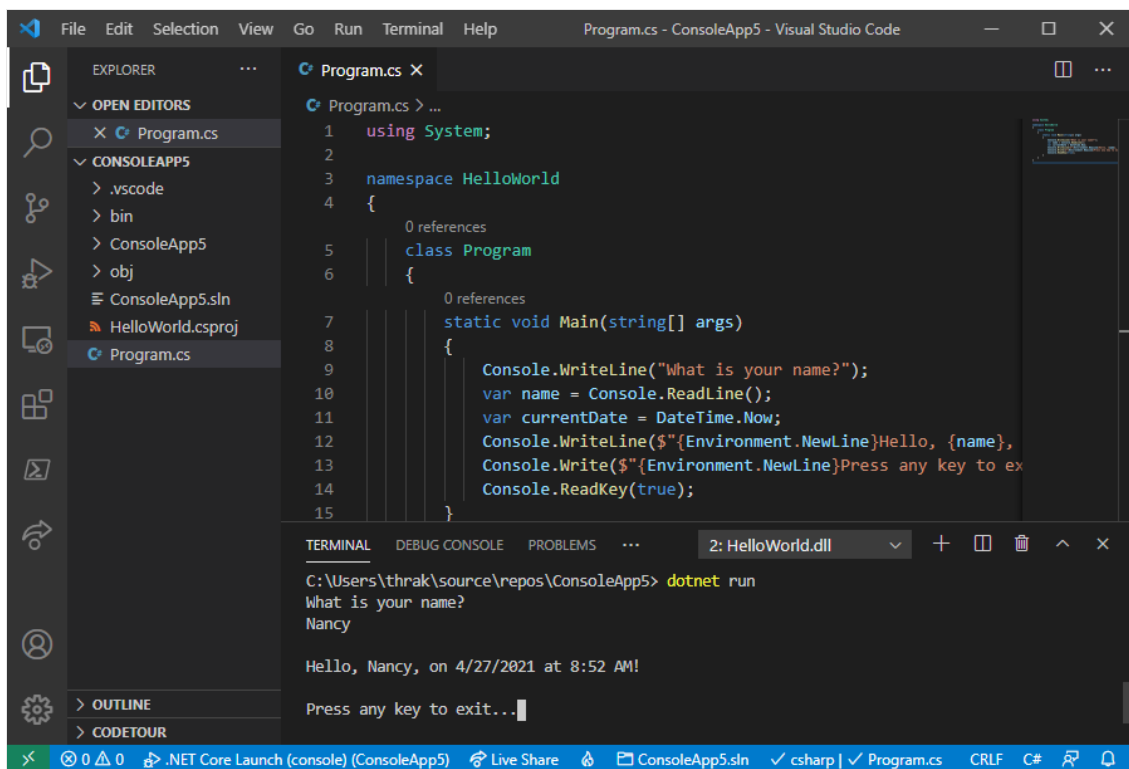
重要

在 Visual Studio Code 中，您必須明確地儲存變更。不同于 Visual Studio，當您建立並執行應用程式時，不會自動儲存檔案變更。

4. 重新執行程式：

.NET CLI	 複製
<pre>dotnet run</pre>	

5. 輸入名稱並按  `enter` 鍵，以回應提示。



6. 按任意鍵以結束該程式。

其他資源


- [設定 Visual Studio Code](#)

下一步

在本教學課程中，您已建立 .NET 主控台應用程式。在下一個教學課程中，您將會對應用程式進行偵錯工具。

使用 **Visual Studio Code** 來對 .NET 主控台應用程式進行偵錯工具

教學課程：使用 Visual Studio Code 來對 .NET 主控台應用程式進行偵錯工具

發行項 • 2021/11/03 • 

此頁面有所助益嗎？ 

選擇 .NET 版本

.NET 6	.NET 5	.NET Core 3.1
--------	--------	---------------

本文內容

[必要條件](#)

[使用 Debug build configuration](#)

[設定中斷點](#)

[設定終端機輸入](#)

[開始偵錯](#)

[使用偵錯主控台](#)

[設定條件式中斷點](#)

[逐步執行程式](#)

[使用發行組建設定](#)

[其他資源](#)

[下一步](#)

本教學課程介紹可在 Visual Studio Code 中使用的偵錯工具，以使用 .net 應用程式。

必要條件

- 本教學課程適用於您使用 Visual Studio Code 建立 .net 主控台應用程式中所建立的主控台應用程式。

使用 Debug build configuration

Debug 和 *Release* 為 .NET 的內建組建設定。您可以使用 Debug 組建設定進行偵錯工具，並使用發行設定來進行最終發行散發。

在「偵錯工具」設定中，程式會使用完整符號的 Debug 資訊進行編譯，而不會進行優化。最佳化會使偵錯變得複雜，因為原始程式碼與產生的指令之間關係較為複雜。程

式的發行設定沒有符號的 debug 資訊，而且已完全優化。

根據預設，Visual Studio Code 啟動設定會使用 Debug 組建設定，因此您不需要在進行偵錯工具之前加以變更。

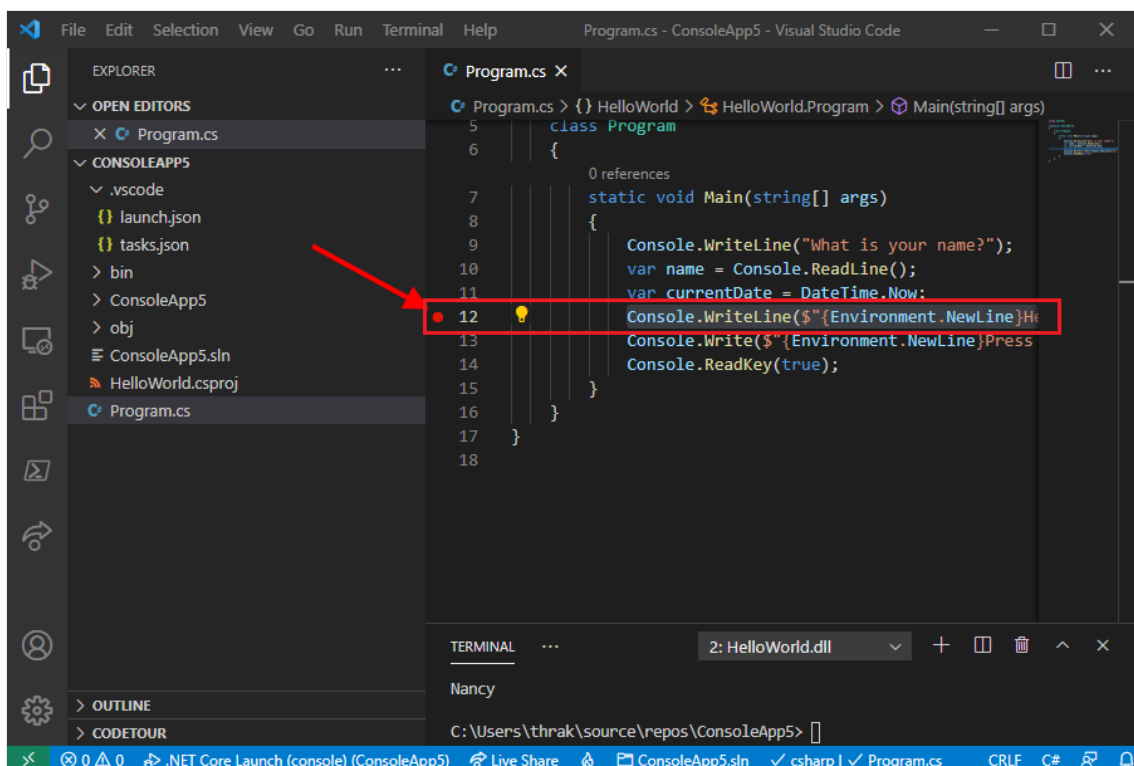
1. 啟動 Visual Studio Code。
2. 開啟您在使用 Visual Studio Code 建立 .net 主控台應用程式中建立的專案資料夾。

設定中斷點

中斷點 會在執行中斷點的那一行之前，暫時中斷應用程式的執行。

1. 開啟 程式.cs 檔案。
2. 在程式碼視窗的左邊界中按一下，在顯示名稱、日期和時間的行上設定 中斷點。左邊界是行號的左邊。設定中斷點的其他方式是按 **F9**，或在 > 選取程式程式碼時，從功能表中選擇 [執行 切換中斷點]。

Visual Studio Code 表示在左邊界中顯示紅點來設定中斷點的行。



設定終端機輸入

中斷點位於 `Console.ReadLine` 方法呼叫之後。偵錯主控台 不接受執行程式的終端機

輸入。若要在偵錯時處理終端輸入，您可以使用整合式終端 (其中一個 Visual Studio Code 視窗) 或外部終端。針對此教學課程，您會使用整合式終端。

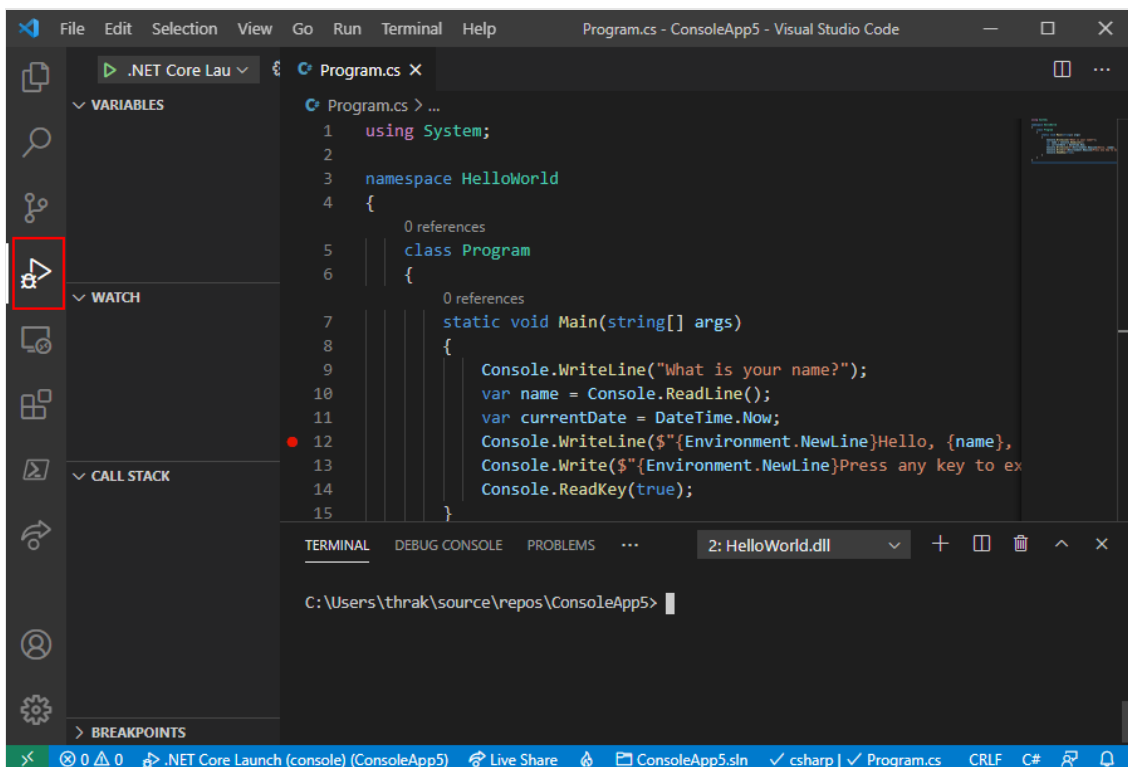
1. 開啟 `.vscode/launch.json`。
2. 將 `console` 設定從變更 `internalConsole` 為 `integratedTerminal`：

JSON	複製
<pre>"console": "integratedTerminal",</pre>	

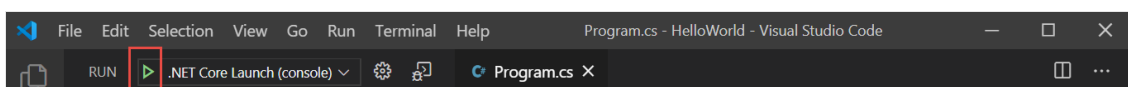
3. 儲存您的變更。

開始偵錯

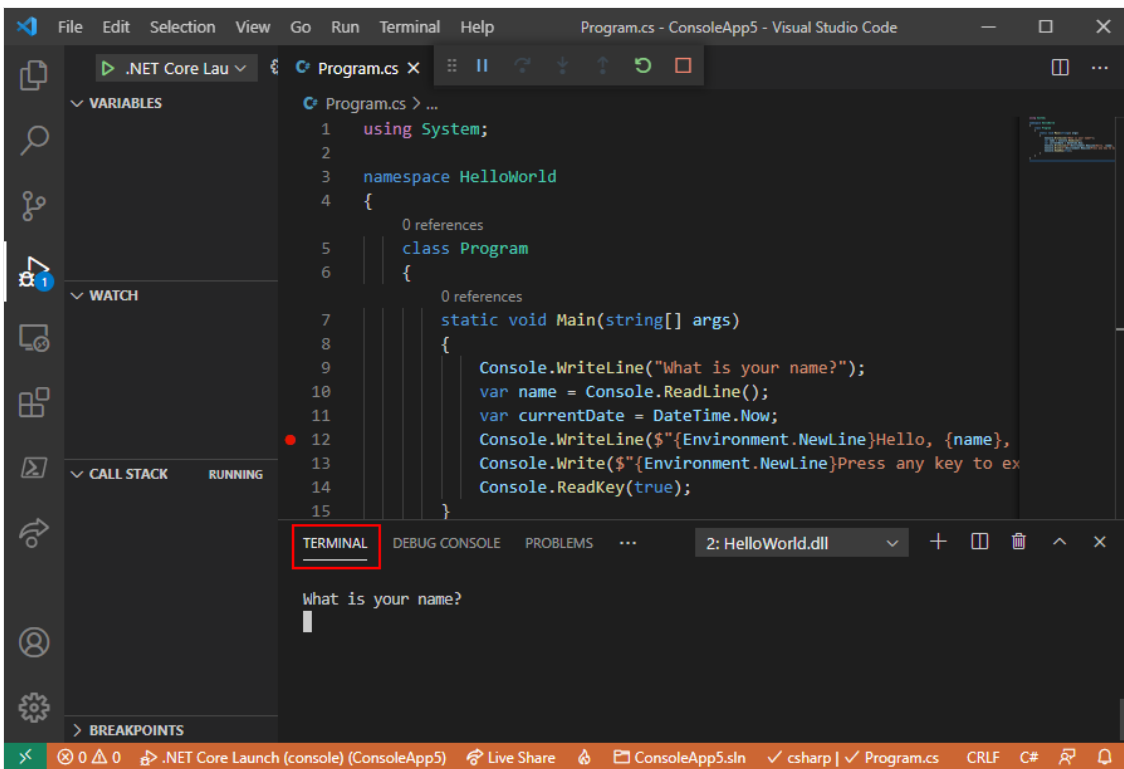
1. 選取左側功能表上的 [偵錯工具] 圖示，以開啟 [偵錯工具]。



2. 選取窗格頂端的綠色箭號，在 [.Net Core 啟動] (主控台) 旁。在偵測模式下啟動程式的其他方式是按 `F5`，或從功能表中選擇 [執行 > 開始調試 程式]。

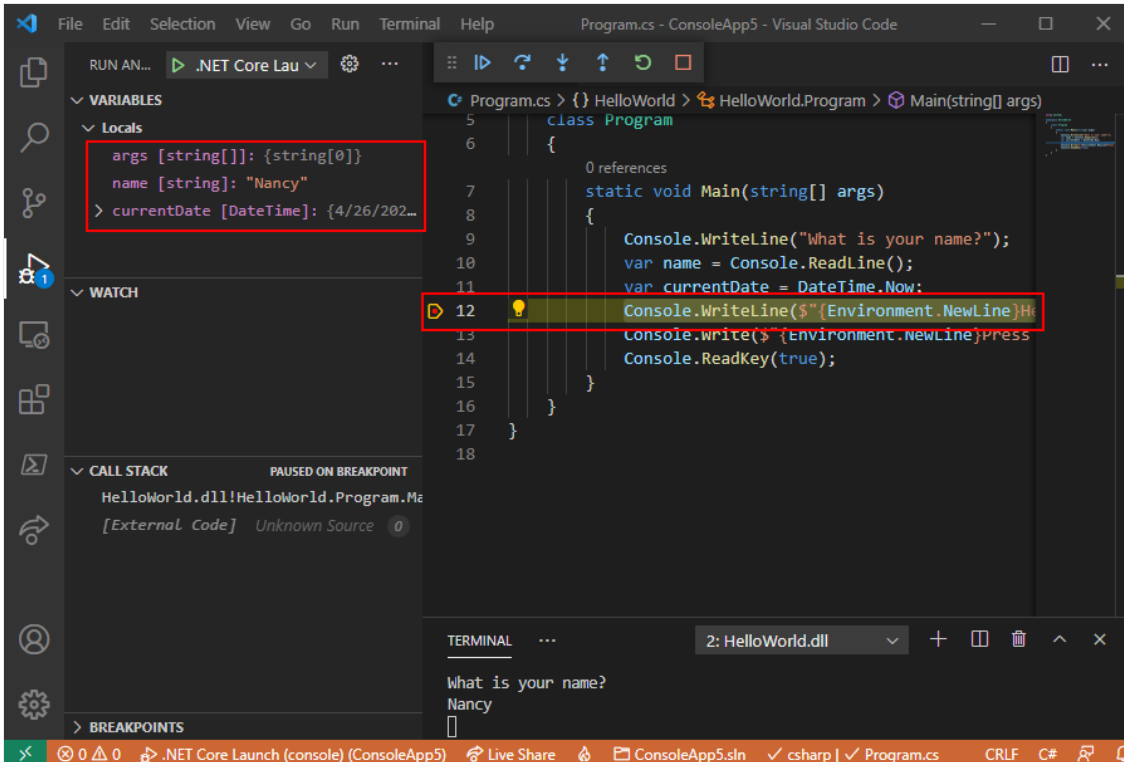


3. 選取 [終端機] 索引標籤，以查看「您的姓名為何？」提示程式在等候回應之前顯示。



4. 在終端機視窗中輸入字串，以回應提示輸入名稱，然後按 。

程式執行會在到達中斷點或方法執行之前停止 `Console.WriteLine`。[變數] 視窗的 [區域變數] 區段會顯示目前正在執行的方法中所定義變數的值。

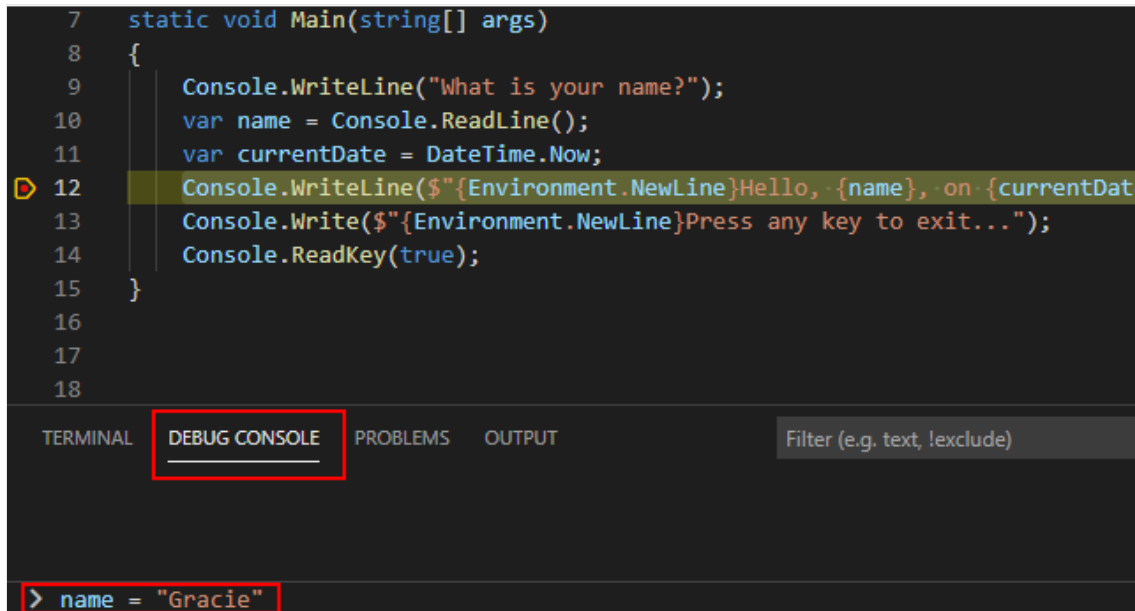


使用偵錯主控台

偵錯主控台 視窗可讓您與您要進行偵錯工具的應用程式互動。您可以變更變數的值，

以查看它對您的程式有何影響。

1. 選取 [偵錯主控台] 索引標籤。
2. 在 `name = "Gracie"` 偵錯主控台 視窗底部的提示字元中輸入，然後按 `enter` 鍵。



```
7 static void Main(string[] args)
8 {
9     Console.WriteLine("What is your name?");
10    var name = Console.ReadLine();
11    var currentDate = DateTime.Now;
12    Console.WriteLine($"{Environment.NewLine}Hello, {name}, on {currentDate}");
13    Console.WriteLine($"{Environment.NewLine}Press any key to exit...");
14    Console.ReadKey(true);
15 }
16
17
18
```

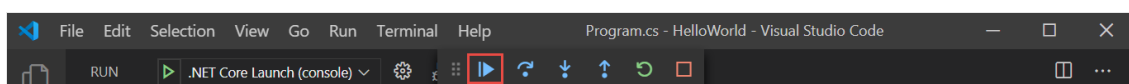
DEBUG CONSOLE

> name = "Gracie"

3. `currentDate = DateTime.Parse("2019-11-16T17:25:00Z").ToUniversalTime()` 在偵錯主控台 視窗的底部輸入，然後按下 `enter` 鍵。

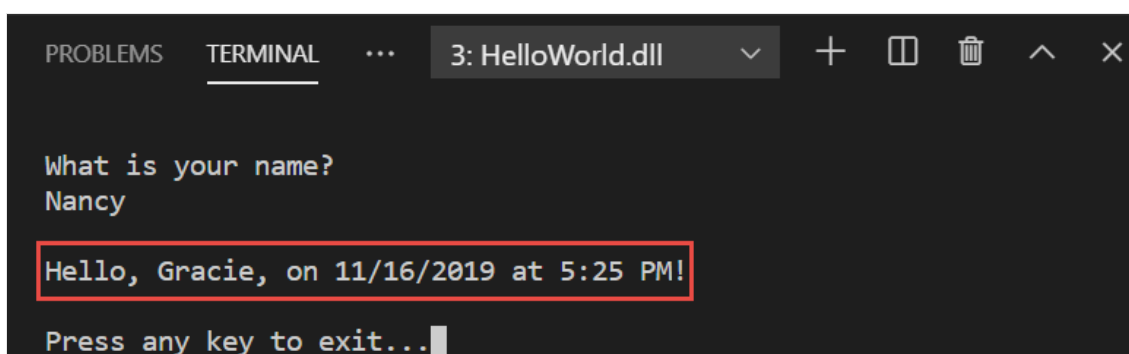
[變數] 視窗會顯示和變數的新值 `name` `currentDate` 。

4. 選取工具列中的 [繼續] 按鈕，以繼續程式執行。另一個繼續的方法是按 `F5` 。



5. 再次選取 [終端 機] 索引標籤。

[主控台] 視窗中顯示的值會對應到您在 偵錯主控台 中所做的變更。



PROBLEMS TERMINAL ... 3: HelloWorld.dll

What is your name?
Nancy

Hello, Gracie, on 11/16/2019 at 5:25 PM!

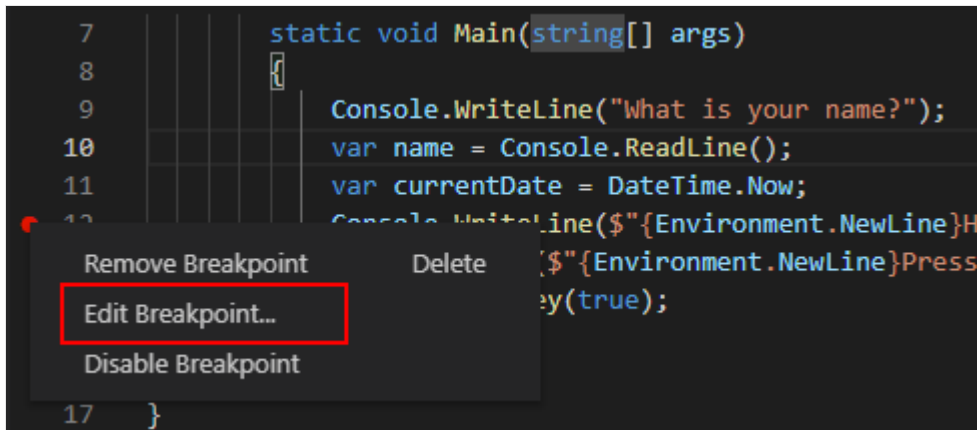
Press any key to exit...

6. 按任意鍵以結束應用程式，並停止偵錯工具。

設定條件式中斷點

程式會顯示使用者輸入的字串。如果使用者未進行任何輸入時，會發生什麼情況？您可以使用稱為 *條件式中斷點* 的實用調試功能來進行測試。

1. 以滑鼠右鍵按一下 (**Ctrl**-按一下代表中斷點的紅點上的 macOS)。在內容功能表中，選取 [編輯中斷點] 以開啟可讓您輸入條件運算式的對話方塊。



2. Expression 在下拉式清單中選取，輸入下列條件運算式，然後按 **enter** 鍵。



每次叫用中斷點時，偵錯工具就會呼叫 `String.IsNullOrEmpty(name)` 方法，而且只有在方法呼叫傳回時，才會在此行上中斷 `true`。

您可以指定叫用 *計數*（而不是條件運算式），以在執行指定次數的語句之前，中斷程式執行。另一個選項是指定 *篩選準則*，以根據執行緒識別碼、進程名稱或執行緒名稱，以這類屬性來中斷程式執行。

3. 按下 **F5** 來啟動程式並進行偵錯工具。
4. 在 [終端 機] 索引標籤中，當系統提示您輸入名稱時，請按 **enter** 鍵。

因為您 (指定的條件 `name` 是 `null` 或 `String.Empty`) 已滿足，所以程式執行會在到達中斷點時，以及方法執行之前停止 `Console.WriteLine` 。

[變數] 視窗會顯示變數的值 `name` 為 `""`、或 `String.Empty` 。

5. 在 偵錯主控台 提示字元中輸入下列語句，並按 `enter` 鍵，確認值為空字串。結果為 `true` 。

C#	複製
<pre>name == String.Empty</pre>	

6. 選取工具列上的 [繼續] 按鈕以繼續程式執行。
7. 選取 [終端 機] 索引標籤，然後按任意鍵以結束程式並停止偵錯工具。
8. 按一下程式碼視窗左邊界中的點，以清除中斷點。清除中斷點的其他方式是在選取程式碼時，按 `F9` 或從功能表中選擇 [執行] > 切換中斷點。
9. 如果您收到警告，指出中斷點條件將會遺失，請選取 [移除中斷點]。

逐步執行程式

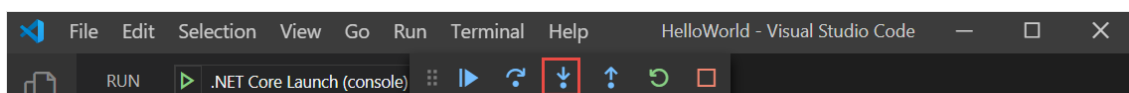
Visual Studio Code 也可讓您逐行執行程式並監視其執行情況。一般來說，您會設定中斷點，並遵循程式流程程式碼的一小部分。因為這個程式很小，所以您可以逐步執行整個程式。

1. 在方法的左大括弧上設定中斷點 `Main` 。
2. 按 `F5` 開始偵錯。

Visual Studio Code 反白顯示中斷點線。

此時，[變數] 視窗 `args` 會顯示陣列是空的，而且 `name` `currentDate` 具有預設值。

3. 選取 [執行 > 步驟] 或按 `F11` 鍵。



Visual Studio Code 反白顯示下一行。

4. 選取 [執行 > 步驟] 或按 `F11` 鍵。

Visual Studio Code 會 `Console.WriteLine` 針對名稱提示執行，並反白顯示下一行執行。下一行是的 `Console.ReadLine name`。[變數] 視窗不會變更，且 [終端機] 索引標籤會顯示「您的名為何？」提示。

5. 選取 [執行 > 步驟] 或按 `F11` 鍵。

Visual Studio 會反白顯示 `name` 變數指派。[變數] 視窗 `name` 會顯示仍然是 `null`。

6. 在 [終端機] 索引標籤中輸入字串，然後按 `enter`，以回應提示。

當您輸入時，[終端機] 索引標籤可能不會顯示您輸入的字串，但 `Console.ReadLine` 方法將會捕捉您的輸入。

7. 選取 [執行 > 步驟] 或按 `F11` 鍵。

Visual Studio Code 會反白顯示 `currentDate` 變數指派。[變數] 視窗會顯示呼叫方法所傳回的值 `Console.ReadLine`。[終端機] 索引標籤會顯示您在提示字元中輸入的字串。

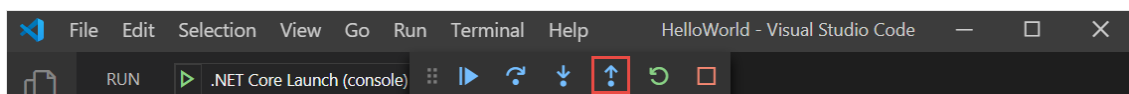
8. 選取 [執行 > 步驟] 或按 `F11` 鍵。

[變數] 視窗會顯示 `currentDate` 從屬性指派之後的變數值 `DateTime.Now`。

9. 選取 [執行 > 步驟] 或按 `F11` 鍵。

Visual Studio Code 會呼叫 `Console.WriteLine(String, Object, Object)` 方法。主控台視窗會顯示已格式化的字串。

10. 選取 [執行 > 跳出] 或按 `Shift` + `F11`。



11. 選取 [終端機] 索引標籤。

終端機會顯示「按任意鍵結束 ...」


12. 按任意鍵以結束該程式。

使用發行組建設定

測試應用程式的偵錯工具版本之後，您應該也要編譯並測試發行版本。發行版本所併入的編譯器優化，可能會影響應用程式的行為。例如，針對改善效能而設計的編譯器

優化，可能會在多執行緒應用程式中建立競爭條件。

若要建立並測試您主控台應用程式的發行版本，請開啟 終端 機並執行下列命令：

.NET CLI	 複製
<pre>dotnet run --configuration Release</pre>	

其他資源



- 在 [Visual Studio Code](#) 中偵錯

下一步

在本教學課程中，您已使用 Visual Studio Code 調試工具。在下一個教學課程中，您將發行應用程式的可部署版本。

使用 **Visual Studio Code** 發佈 .NET 主控台應用程式

教學課程：使用 Visual Studio Code 發佈 .NET 主控台應用程式

發行項 • 2021/11/03 •  

此頁面有所助益嗎？  

選擇 .NET 版本

.NET 6	.NET 5	.NET Core 3.1
---------------	--------	---------------

本文內容

[必要條件](#)

[發佈應用程式](#)

[檢查檔案](#)

[執行已發佈的應用程式](#)

[其他資源](#)

[下一步](#)

本教學課程說明如何發佈主控台應用程式，讓其他使用者可以執行它。發行會建立一組執行應用程式所需的檔案。若要部署檔案，請將檔案複製到目的電腦。

.net CLI 是用來發佈應用程式，因此您可以在本教學課程以外的程式碼編輯器中使用 Visual Studio Code（如果您想要的話）。

必要條件

- 本教學課程適用於您使用 Visual Studio Code 建立 .net 主控台應用程式中所建立的主控台應用程式。

發佈應用程式

1. 啟動 Visual Studio Code。
2. 開啟您在 使用 Visual Studio Code 建立 .net 主控台應用程式中建立的 *HelloWorld* 專案資料夾。
3. 從主功能表選擇 [**View** > **Terminal**]。


終端機會在 *HelloWorld* 資料夾中開啟。

4. 執行下列命令：

.NET CLI	 複製
<pre>dotnet publish --configuration Release</pre>	

預設組建設定為 *Debug*，所以此命令會指定 *發行* 組建設定。發行組建設定的輸出具有最基本的符號 *debug* 資訊，並且已完全優化。

命令輸出會類似下列範例：

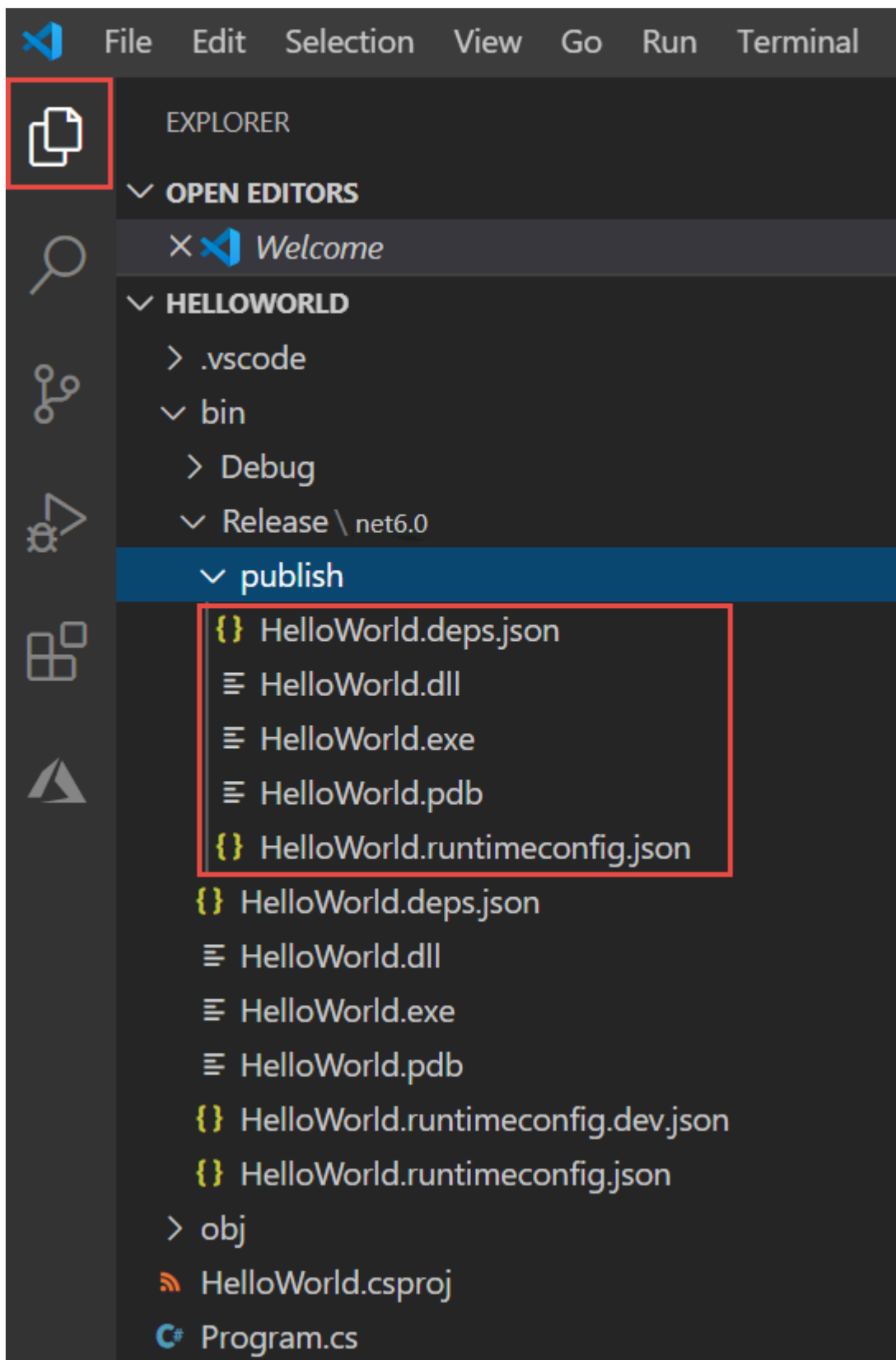
輸出	 複製
<pre>Microsoft (R) Build Engine version 16.7.0+b89cb5fde for .NET Copyright (C) Microsoft Corporation. All rights reserved. Determining projects to restore... All projects are up-to-date for restore. HelloWorld -> C:\Projects\HelloWorld\bin\Release\net6.0 \HelloWorld.dll HelloWorld -> C:\Projects\HelloWorld\bin\Release\net6.0\publish\</pre>	

檢查檔案

根據預設，發佈程式會建立與 *framework* 相依的部署，這是一種部署類型，其中已發佈的應用程式會在已安裝 .NET 執行時間的電腦上執行。若要執行已發佈的應用程式，您可以使用可執行檔，或 `dotnet HelloWorld.dll` 從命令提示字元執行命令。

在下列步驟中，您將查看發佈程式所建立的檔案。

1. 在左側導覽列中選取 [**Explorer**]。
2. 展開 *bin/Release/net 6.0/publish*。



如圖所示，已發行的輸出包含下列檔案：

- *HelloWorld.deps.json*

這是應用程式的執行時間相依性檔案。它會定義 .NET 元件和程式庫 (包括動態連結程式庫，其中包含執行應用程式所需的應用程式)。如需詳細資訊，請參閱 [執行時間設定檔](#)。

- *HelloWorld.dll*

這是應用程式的 [framework 相依部署](#) 版本。若要執行這個動態連結程式庫，請 `dotnet HelloWorld.dll` 在命令提示字元中輸入。執行應用程式的這個方法適用於已安裝 .NET 執行時間的任何平臺。

- *HelloWorld.exe* Linux 上的 (*HelloWorld*，而不是在 macOS 上建立。)

這是應用程式的 [framework 相依可執行檔](#) 版本。檔案是作業系統特定的。

- *HelloWorld.pdb* (對於部署為選用)

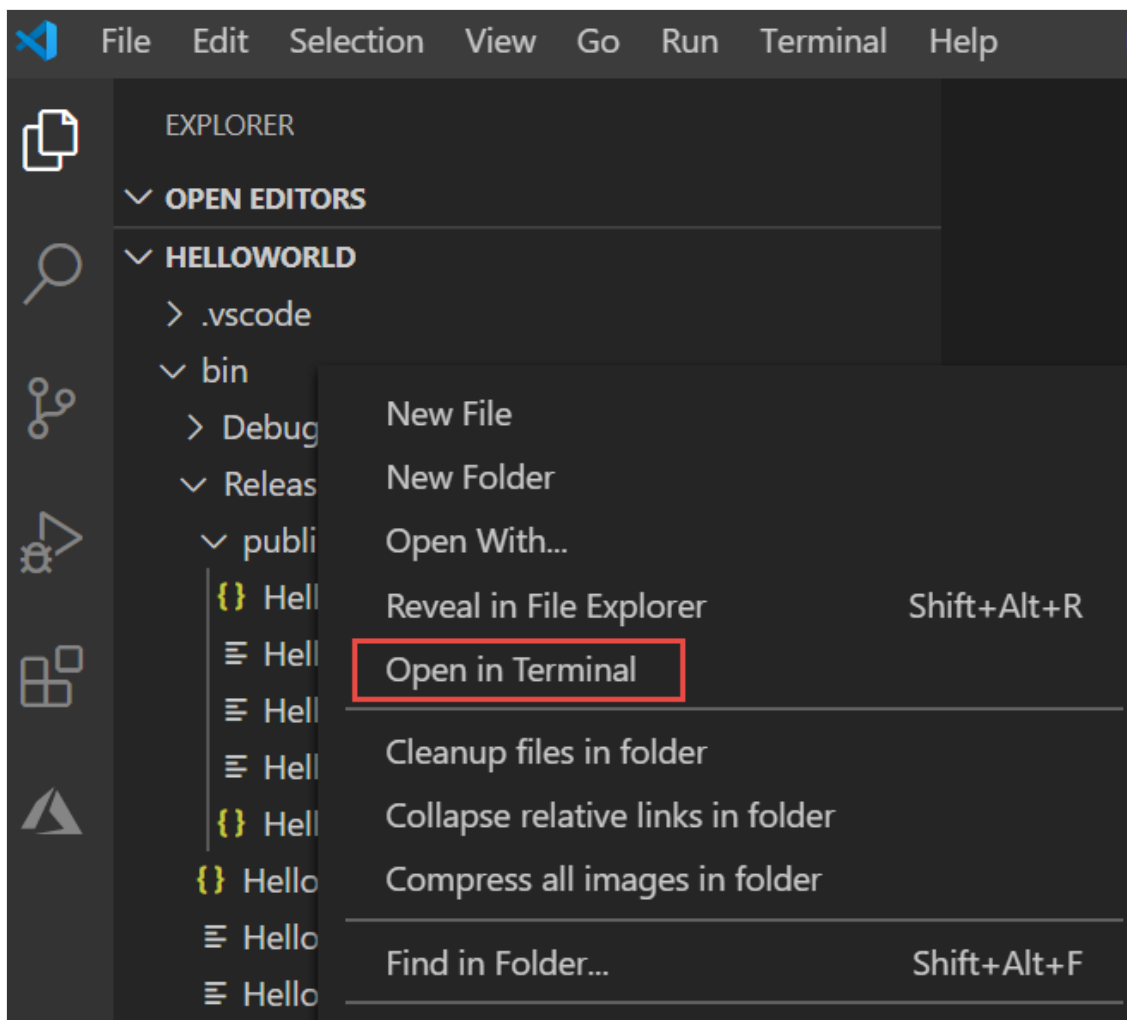
這是 debug 符號檔。此檔案不需要隨您的應用程式部署，但當您需要對應用程式發行的版本進行偵錯，則應該儲存它。

- *HelloWorld.runtimeconfig.json*

這是應用程式的執行時間設定檔案。它會識別您的應用程式建立用來執行的 .NET 版本。您也可以將設定選項新增至其中。如需詳細資訊，請參閱 [.net 執行時間設定](#)。

執行已發佈的應用程式

1. 在 [**Explorer**] 中，以滑鼠右鍵按一下 [發佈] 資料夾 (`Ctrl`-按一下 macOS)，然後選取 [在終端機中開啟]。



2. 在 Windows 或 Linux 上，使用可執行檔來執行應用程式。
 - a. 在 Windows 上輸入， `.\HelloWorld.exe` 然後按 `enter` 鍵。
 - b. 在 Linux 上，輸入 `./HelloWorld`，然後按 `enter` 鍵。
 - c. 輸入名稱以回應提示，然後按任意鍵以結束。
3. 在任何平臺上，使用下列命令來執行應用程式 `dotnet`：
 - a. 輸入 `dotnet HelloWorld.dll`，然後按 `enter` 鍵。
 - b. 輸入名稱以回應提示，然後按任意鍵以結束。

其他資源



- [.NET 應用程式部署](#)

下一步

在本教學課程中，您已發佈主控台應用程式。在下一個教學課程中，您將建立類別庫。

使用 **Visual Studio Code** 建立 **.NET** 類別庫

教學課程：使用 Visual Studio Code 建立 .NET 類別庫

發行項 • 2021/11/09 •  

此頁面有所助益嗎？  

選擇 .NET 版本

.NET 6

.NET 5

.NET Core 3.1

本文內容

[必要條件](#)

[建立方案](#)

[建立類別庫專案](#)

[將主控台應用程式新增至解決方案](#)

[新增專案參考](#)

[執行應用程式](#)

[其他資源](#)

[下一步](#)

在本教學課程中，您會建立包含單一字串處理方法的簡單公用程式程式庫。

「類別庫」會定義應用程式所呼叫的類型和方法。如果程式庫以 .NET Standard 2.0 為目標，則可以由任何 .net 執行 (，包括支援 .NET Standard 2.0 的 .NET Framework) 。如果程式庫以 .NET 6 為目標，則可由以 .NET 6 為目標的任何應用程式呼叫。本教學課程說明如何以 .NET 6 為目標。

當您建立類別庫時，可以將它散發為協力廠商元件，或做為配套的元件，以及一或多個應用程式。

必要條件

- 已安裝 [c# 擴充](#) 功能的 [Visual Studio Code](#) 。如需有關如何在 Visual Studio Code 上安裝擴充功能的詳細資訊，請參閱 [VS Code 擴充功能 Marketplace](#) 。
- [.Net 6 SDK](#) 。

建立方案

首先，建立空白的方案，將類別庫專案放置在中。方案可作為一或多個專案的容器。您會將其他相關的專案新增至相同的方案。

- 1. 啟動 Visual Studio Code。
- 2. 從主功能表選取 [macOS) 上的 [開啟 資料夾] (開啟 ...
- 3. 在 [開啟資料夾] 對話方塊中，建立 >classlibraryprojects 資料夾，然後按一下 [選取資料夾 (在 macOS) 開啟]。
- 4. 從主功能表中選取 [View terminal]，以在 Visual Studio Code 中開啟 終端 機 >。

終端 機會在 >classlibraryprojects 資料夾中使用命令提示字元開啟。

- 5. 在 終端 機中，輸入下列命令：

.NET CLI	複製
dotnet new sln	

終端機輸出如下列範例所示：

輸出	複製
The template "Solution File" was created successfully.	

建立類別庫專案

將名為 "StringLibrary" 的新 .NET 類別庫專案加入至方案。

- 1. 在終端機中執行下列命令，以建立程式庫專案：

.NET CLI	複製
dotnet new classlib -o StringLibrary	

-o 或 --output 命令指定要放置所產生輸出的位置。


終端機輸出如下列範例所示：

輸出	複製
----	----

```
The template "Class library" was created successfully.
Processing post-creation actions...
Running 'dotnet restore' on StringLibrary\StringLibrary.csproj...
    Determining projects to restore...
    Restored C:\Projects\ClassLibraryProjects\StringLibrary
\StringLibrary.csproj (in 328 ms).
Restore succeeded.
```

2. 執行下列命令，以將程式庫專案新增至方案：


.NET CLI

 複製

```
dotnet sln add StringLibrary/StringLibrary.csproj
```

終端機輸出如下列範例所示：

輸出


 複製

```
Project `StringLibrary\StringLibrary.csproj` added to the solution.
```

3. 請檢查以確定程式庫以 .NET 6 為目標。在 [**Explorer**] 中，開啟 [*StringLibrary*]/[*StringLibrary*]。

TargetFramework 元素會顯示專案的目標為 .net 6.0。

XML

 複製


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

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>

</Project>
```

4. 開啟 *Class1*，並以下列程式碼取代程式碼。

C#

 複製

```
namespace UtilityLibraries;

public static class StringLibrary
{
    public static bool StartsWithUpper(this string? str)
    {
        if (string.IsNullOrEmpty(str))
```

```
        return false;


        char ch = str[0];
        return char.IsUpper(ch);
    }
}
```

類別庫 (class library) `UtilityLibraries.StringLibrary` 包含名為的方法 `StartsWithUpper`。這個方法會傳回 `Boolean` 值，指出目前字串實例的開頭是否為大寫字元。Unicode 標準會區別大寫和小寫字元。如果是大寫字元，`Char.IsUpper(Char)` 方法會傳回 `true`。


`StartsWithUpper` 會實作為 [擴充方法](#)，如此您就可以呼叫它，就如同它是類別的成員一樣 `String`。

5. 儲存檔案。

6. 執行下列命令以建立方案，並確認專案編譯時沒有錯誤。

.NET CLI	 複製
<code>dotnet build</code>	


終端機輸出如下列範例所示：

輸出	 複製
<pre>Microsoft (R) Build Engine version 16.7.0+b89cb5fde for .NET Copyright (C) Microsoft Corporation. All rights reserved. Determining projects to restore... All projects are up-to-date for restore. StringLibrary -> C:\Projects\ClassLibraryProjects\StringLibrary \bin\Debug\net6.0\StringLibrary.dll Build succeeded. 0 Warning(s) 0 Error(s) Time Elapsed 00:00:02.78</pre>	


將主控台應用程式新增至解決方案

加入使用類別庫的主控台應用程式。應用程式會提示使用者輸入字串，並報告字串是否以大寫字元開頭。


1. 在終端機中執行下列命令，以建立主控台應用程式專案：

.NET CLI	 複製
<pre>dotnet new console -o ShowCase</pre>	


終端機輸出如下列範例所示：

輸出	 複製
<pre>The template "Console Application" was created successfully. Processing post-creation actions... Running 'dotnet restore' on ShowCase\ShowCase.csproj... Determining projects to restore... Restored C:\Projects\ClassLibraryProjects\ShowCase\ShowCase.csproj (in 210 ms). Restore succeeded.</pre>	


2. 執行下列命令，以將主控台應用程式專案新增至方案：

.NET CLI	 複製
<pre>dotnet sln add ShowCase\ShowCase.csproj</pre>	

終端機輸出如下列範例所示：

輸出	 複製
<pre>Project `ShowCase\ShowCase.csproj` added to the solution.</pre>	

3. 開啟 [展示]/[Program]，並將所有程式碼取代為下列程式碼。

C#	 複製
<pre>using UtilityLibraries; class Program { static void Main(string[] args) { int row = 0; do { if (row == 0 row >= 25) ResetConsole(); string? input = Console.ReadLine(); if (string.IsNullOrEmpty(input)) break; } } }</pre>	

```
        Console.WriteLine($"Input: {input}");
        Console.WriteLine("Begins with uppercase? " +
            $"{(input.StartsWithUpper() ? "Yes" : "No")}");
        Console.WriteLine();
        row += 4;
    } while (true);
    return;

// Declare a ResetConsole local method
void ResetConsole()
{
    if (row > 0)
    {
        Console.WriteLine("Press any key to continue...");
        Console.ReadKey();
    }
    Console.Clear();
    Console.WriteLine($"{Environment.NewLine}Press <Enter>
only to exit; otherwise, enter a string and press <Enter>:
{Environment.NewLine}");
    row = 3;
}
}
```

該程式碼會使用 `row` 變數來維護寫入至主控台視窗的資料列數目計數。只要超過或等於25，程式碼就會清除主控台視窗，並向使用者顯示訊息。

此程式會提示使用者輸入字串。它會指出該字串開頭是否為大寫字元。如果使用者按下 `Enter` 鍵但未輸入字串，則應用程式會結束，且主控台視窗會關閉。

4. 儲存您的變更。

新增專案參考

一開始，新的主控台應用程式專案沒有類別庫的存取權。若要允許它呼叫類別庫中的方法，請建立類別庫專案的專案參考。


1. 執行下列命令：

.NET CLI

 複製


```
dotnet add ShowCase/ShowCase.csproj reference
StringLibrary/StringLibrary.csproj
```

終端機輸出如下列範例所示：

輸出	 複製
Reference `..\StringLibrary\StringLibrary.csproj` added to the project.	


執行應用程式

1. 在終端中執行下列命令：

.NET CLI	 複製
<code>dotnet run --project ShowCase/ShowCase.csproj</code>	

2. 輸入字串並按 `enter` 鍵以嘗試程式，然後按 `enter` 鍵結束。

終端機輸出如下列範例所示：

輸出	 複製
<pre>Press <Enter> only to exit; otherwise, enter a string and press <Enter>: A string that starts with an uppercase letter Input: A string that starts with an uppercase letter Begins with uppercase? : Yes a string that starts with a lowercase letter Input: a string that starts with a lowercase letter Begins with uppercase? : No</pre>	

其他資源



- [使用 .NET CLI 開發程式庫](#)
- [.NET Standard 支援的版本和平臺](#)。

下一步

在本教學課程中，您已建立解決方案、新增程式庫專案，並加入使用該程式庫的主控台應用程式專案。在下一個教學課程中，您會將單元測試專案加入至方案。

使用 **Visual Studio Code** 以 .NET 測試 .NET 類別庫

教學課程：使用 Visual Studio Code 測試 .NET 類別庫

發行項 • 2021/11/03 •  

此頁面有所助益嗎？  

選擇 .NET 版本

.NET 6

.NET 5

.NET Core 3.1

本文內容

[必要條件](#)

[建立單元測試專案](#)

[新增專案參考](#)

[加入並執行單元測試方法](#)

[處理測試失敗](#)

[測試程式庫的發行版本](#)

[偵錯測試](#)

[其他資源](#)

[下一步](#)

本教學課程示範如何將測試專案加入至方案，以自動化單元測試。

必要條件


- 本教學課程適用於使用 [Visual Studio Code](#) 建立 .net 類別庫中所建立的方案。

建立單元測試專案

單元測試能在開發與發佈期間提供自動化的軟體測試。您在本教學課程中使用的測試架構是 [MSTest](#)。[MSTest](#) 是您可以從中選擇的三種測試架構之一。其他則為 [xUnit](#) 和 [nUnit](#)。

1. 啟動 Visual Studio Code。
2. 開啟 `ClassLibraryProjects` 您在使用 [Visual Studio Code](#) 建立 .net 類別庫中建立的方案。

3. 建立名為 "StringLibraryTest" 的單元測試專案。

.NET CLI	 複製
<pre>dotnet new mstest -o StringLibraryTest</pre>	

專案範本會使用下列程式碼建立 `UnitTest1.cs` 檔案：


C#	 複製
<pre>using Microsoft.VisualStudio.TestTools.UnitTesting; namespace StringLibraryTest { [TestClass] public class UnitTest1 { [TestMethod] public void TestMethod1() { } } }</pre>	

單元測試範本建立的原始程式碼會執行下列動作：

- 它會匯入 `Microsoft.VisualStudio.TestTools.UnitTesting` 命名空間，其中包含用於單元測試的類型。
- 它會將 `TestClassAttribute` 屬性套用至 `UnitTest1` 類別。
- 它會套用 `TestMethodAttribute` 屬性來定義 `TestMethod1`。

在叫用單元測試時，會自動執行在標記為`[TestClass]`的測試類別中，以`[TestMethod]`標記的每個方法。

4. 將測試專案加入至方案。

.NET CLI	 複製
<pre>dotnet sln add StringLibraryTest/StringLibraryTest.csproj</pre>	

新增專案參考

若要讓測試專案使用 `StringLibrary` 類別，請在專案中加入專案的參考 `StringLibraryTest StringLibrary`。

1. 執行下列命令：

.NET CLI

複製

```
dotnet add StringLibraryTest/StringLibraryTest.csproj reference
StringLibrary/StringLibrary.csproj
```

加入並執行單元測試方法

當 Visual Studio 叫用單元測試時，它會執行每個以屬性標記之類別中的屬性標記的方法 `TestMethodAttribute` `TestClassAttribute` 。 當找到第一個失敗或方法中包含的所有測試都成功時，測試方法便會結束。

最常見的測試會呼叫 `Assert` 類別的成員。 許多判斷提示方法都至少包括兩個參數，一個是預期的測試結果，另一個是實際的測試結果。 `Assert` 下表顯示某些類別最常呼叫的方法：

Assert 方法	函數
<code>Assert.AreEqual</code>	驗證兩個值或物件相等。 如果值或物件不相等，判斷提示就會失敗。
<code>Assert.AreSame</code>	驗證兩個物件變數參考相同的物件。 如果變數參考不同的物件，判斷提示就會失敗。
<code>Assert.IsFalse</code>	驗證條件為 <code>false</code> 。 如果條件為 <code>true</code> ，判斷提示就會失敗。
<code>Assert.IsNotNull</code>	確認物件不是 <code>null</code> 。 如果物件為 <code>null</code> ，判斷提示就會失敗。

您也可以 `Assert.ThrowsException` 在測試方法中使用方法，來指出預期會擲回的例外狀況類型。 如果未擲回指定的例外狀況，測試便會失敗。

在測試 `StringLibrary.StartsWithUpper` 方法時，您想提供幾個以大寫字元開頭的字串。 您預期此方法在這些情況下傳回 `true` ，因此您可以呼叫 `Assert.IsTrue` 方法。 同樣地，您想提供幾個開頭不是大寫字元的字串。 您預期此方法在這些情況下傳回 `false` ，因此您可以呼叫 `Assert.IsFalse` 方法。

因為您的程式庫方法會處理字串，您也想要確保它能成功處理 空字串 (`String.Empty`) 和 `null` 字串。 空字串是指沒有任何字元且其 `Length` 為0的字串。 `null` 字串是指尚未初始化的字串。 您可以 `StartsWithUpper` 直接呼叫作為靜態方法，並傳遞單一 `String` 引數。 或者，您可以 `StartsWithUpper` 在指派給的變數上呼叫做為擴充方法 `string null` 。

您會定義三個方法，每個方法會 [Assert](#) 針對字串陣列中的每個元素呼叫方法。您將呼叫方法多載，讓您指定在測試失敗時所要顯示的錯誤訊息。訊息會識別造成失敗的字串。

建立測試方法：

1. 開啟 *StringLibraryTest/UnitTest1*，並將所有程式碼取代為下列程式碼。


C#	複製
<pre>using Microsoft.VisualStudio.TestTools.UnitTesting; using UtilityLibraries; namespace StringLibraryTest; [TestClass] public class UnitTest1 { [TestMethod] public void TestStartsWithUpper() { // Tests that we expect to return true. string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" }; foreach (var word in words) { bool result = word.StartsWithUpper(); Assert.IsTrue(result, String.Format("Expected for '{0}': true; Actual: {1}", word, result)); } } [TestMethod] public void TestDoesNotStartWithUpper() { // Tests that we expect to return false. string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство", "1234", ".", ";", " " }; foreach (var word in words) { bool result = word.StartsWithUpper(); Assert.IsFalse(result, String.Format("Expected for '{0}': false; Actual: {1}", word, result)); } } [TestMethod]</pre>	


```
public void DirectCallWithNullOrEmpty()
{
    // Tests that we expect to return false.
    string?[] words = { string.Empty, null };
    foreach (var word in words)
    {
        bool result = StringLibrary.StartsWithUpper(word);
        Assert.IsFalse(result,
            String.Format("Expected for '{0}': false; Actual:
{1}",
                                word == null ? "<null>" : word,
                                result));
    }
}
```


方法中的大寫字元測試 `TestStartsWithUpper` 包含希臘文大寫字母 Alpha (u + 0391) 和斯拉夫文大寫字母 EM (U + 041C) 。 方法中的小寫字元測試 `TestDoesNotStartWithUpper` 包含希臘文小寫字母 Alpha (u + 03B1) 和斯拉夫文小寫字母 Ghe (U + 0433) 。

2. 儲存您的變更。

3. 執行測試：

.NET CLI	 複製
<code>dotnet test</code> StringLibraryTest/StringLibraryTest.csproj	

終端機輸出顯示所有測試皆通過。

輸出	 複製
<pre>Starting test execution, please wait... A total of 1 test files matched the specified pattern. Passed! - Failed: 0, Passed: 3, Skipped: 0, Total: 3, Duration: 3 ms - StringLibraryTest.dll (net6.0)</pre>	

處理測試失敗

如果您正在執行以測試為導向的開發 (TDD)，您會先撰寫測試，並在您第一次執行時失敗。然後，將程式碼新增至應用程式，讓測試成功。在本教學課程中，您已在撰寫應用程式程式碼驗證之後建立測試，因此您未看到測試失敗。若要在預期測試失敗時驗證測試失敗，請將不正確值加入測試輸入。

1. 修改 `TestDoesNotStartWithUpper` 方法中的 `words` 陣列，以包含字串「錯誤」。

C#	複製
<pre>string[] words = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство", "1234", ".", ";", " " };</pre>	

2. 執行測試：

.NET CLI	複製
<pre>dotnet test StringLibraryTest/StringLibraryTest.csproj</pre>	

終端機輸出顯示一個測試失敗，並針對失敗的測試提供錯誤訊息：「`IsFalse` 失敗。'Error' 的預期：`false`；實際：`True`」。因為失敗，所以在測試 "Error" 之後，陣列中沒有任何字串。

輸出	複製
<pre>Starting test execution, please wait... A total of 1 test files matched the specified pattern. Failed TestDoesNotStartWithUpper [28 ms] Error Message: Assert.IsFalse failed. Expected for 'Error': false; Actual: True Stack Trace: at StringLibraryTest.UnitTest1.TestDoesNotStartWithUpper() in C:\ClassLibraryProjects\StringLibraryTest\UnitTest1.cs:line 33 Failed! - Failed: 1, Passed: 2, Skipped: 0, Total: 3, Duration: 31 ms - StringLibraryTest.dll (net5.0)</pre>	

3. 移除您在步驟1中新增的字串 "Error"。重新執行測試和測試階段。

測試程式庫的發行版本

現在，在執行程式庫的偵錯工具時，所有測試都已通過，請針對程式庫的發行組建執行額外的測試。許多因素 (包括編譯器最佳化) 有時會在偵錯和發行組建之間導致不同的行為。

1. 使用發行組建設定執行測試：

.NET CLI	複製
<pre>dotnet test StringLibraryTest/StringLibraryTest.csproj</pre>	

```
--configuration Release
```

所有測試皆通過。

偵錯測試

如果您使用 Visual Studio Code 作為 IDE，您可以使用在使用您的單元測試專案來進行程式碼的偵錯工具時，使用 Visual Studio Code 的相同程式。請開啟 [*StringLibraryTest*]/[*UnitTest1*]，而不是啟動 展示 應用程式專案，而是選取 [偵錯工具行 7和8之間的 所有測試]。如果找不到，請按 **Ctrl** + **Shift** + **P** 開啟命令選擇區，然後輸入 重載視窗。

Visual Studio Code 啟動已附加偵錯工具的測試專案。執行將會在您已新增至測試專案或基礎程式庫程式碼的任何中斷點停止執行。

其他資源

- [.NET 中的單元測試](#)

下一步

在本教學課程中，您已對類別庫進行單元測試。您可以將程式庫以套件的形式發佈至 [NuGet](#)，以供其他人使用。若要深入瞭解，請遵循 [NuGet 教學課程](#)：

使用 **dotnet CLI** 建立及發佈套件



如果您將程式庫發佈為 NuGet 套件，其他人就可以安裝和使用它。若要深入瞭解，請遵循 [NuGet 教學課程](#)：

利用 **dotnet CLI** 安裝並使用套件

程式庫不需要以套件的形式散發。它可以與使用它的主控台應用程式配套。若要瞭解如何發佈主控台應用程式，請參閱本系列的先前教學課程：

使用 **Visual Studio Code** 發佈 .NET 主控台應用程式

教學課程：使用 Visual Studio Code 測試 .NET 類別庫

發行項 • 2021/11/03 •  

此頁面有所助益嗎？  

選擇 .NET 版本

.NET 6

.NET 5

.NET Core 3.1

本文內容

[必要條件](#)

[建立單元測試專案](#)

[新增專案參考](#)

[加入並執行單元測試方法](#)

[處理測試失敗](#)

[測試程式庫的發行版本](#)

[偵錯測試](#)

[其他資源](#)

[下一步](#)

本教學課程示範如何將測試專案加入至方案，以自動化單元測試。

必要條件


- 本教學課程適用於使用 [Visual Studio Code](#) 建立 .net 類別庫中所建立的方案。

建立單元測試專案


單元測試能在開發與發佈期間提供自動化的軟體測試。您在本教學課程中使用的測試架構是 [MSTest](#)。[MSTest](#) 是您可以從中選擇的三種測試架構之一。其他則為 [xUnit](#) 和 [nUnit](#)。

1. 啟動 Visual Studio Code。
2. 開啟 `ClassLibraryProjects` 您在使用 [Visual Studio Code](#) 建立 .net 類別庫中建立的方案。

3. 建立名為 "StringLibraryTest" 的單元測試專案。

.NET CLI	 複製
<pre>dotnet new mstest -o StringLibraryTest</pre>	

專案範本會使用下列程式碼建立 `UnitTest1.cs` 檔案：


C#	 複製
<pre>using Microsoft.VisualStudio.TestTools.UnitTesting; namespace StringLibraryTest { [TestClass] public class UnitTest1 { [TestMethod] public void TestMethod1() { } } }</pre>	

單元測試範本建立的原始程式碼會執行下列動作：

- 它會匯入 `Microsoft.VisualStudio.TestTools.UnitTesting` 命名空間，其中包含用於單元測試的類型。
- 它會將 `TestClassAttribute` 屬性套用至 `UnitTest1` 類別。
- 它會套用 `TestMethodAttribute` 屬性來定義 `TestMethod1`。

在叫用單元測試時，會自動執行在標記為`[TestClass]`的測試類別中，以`[TestMethod]`標記的每個方法。

4. 將測試專案加入至方案。

.NET CLI	 複製
<pre>dotnet sln add StringLibraryTest/StringLibraryTest.csproj</pre>	

新增專案參考

若要讓測試專案使用 `StringLibrary` 類別，請在專案中加入專案的參考 `StringLibraryTest StringLibrary`。

1. 執行下列命令：

.NET CLI

複製

```
dotnet add StringLibraryTest/StringLibraryTest.csproj reference
StringLibrary/StringLibrary.csproj
```

加入並執行單元測試方法

當 Visual Studio 叫用單元測試時，它會執行每個以屬性標記之類別中的屬性標記的方法 `TestMethodAttribute` `TestClassAttribute` 。 當找到第一個失敗或方法中包含的所有測試都成功時，測試方法便會結束。

最常見的測試會呼叫 `Assert` 類別的成員。 許多判斷提示方法都至少包括兩個參數，一個是預期的測試結果，另一個是實際的測試結果。 `Assert` 下表顯示某些類別最常呼叫的方法：

Assert 方法	函數
<code>Assert.AreEqual</code>	驗證兩個值或物件相等。 如果值或物件不相等，判斷提示就會失敗。
<code>Assert.AreSame</code>	驗證兩個物件變數參考相同的物件。 如果變數參考不同的物件，判斷提示就會失敗。
<code>Assert.IsFalse</code>	驗證條件為 <code>false</code> 。 如果條件為 <code>true</code> ，判斷提示就會失敗。
<code>Assert.IsNotNull</code>	確認物件不是 <code>null</code> 。 如果物件為 <code>null</code> ，判斷提示就會失敗。

您也可以 `Assert.ThrowsException` 在測試方法中使用方法，來指出預期會擲回的例外狀況類型。 如果未擲回指定的例外狀況，測試便會失敗。

在測試 `StringLibrary.StartsWithUpper` 方法時，您想提供幾個以大寫字元開頭的字串。 您預期此方法在這些情況下傳回 `true` ，因此您可以呼叫 `Assert.IsTrue` 方法。 同樣地，您想提供幾個開頭不是大寫字元的字串。 您預期此方法在這些情況下傳回 `false` ，因此您可以呼叫 `Assert.IsFalse` 方法。

因為您的程式庫方法會處理字串，您也想要確保它能成功處理 空字串 (`String.Empty`) 和 `null` 字串。 空字串是指沒有任何字元且其 `Length` 為0的字串。 `null` 字串是指尚未初始化的字串。 您可以 `StartsWithUpper` 直接呼叫作為靜態方法，並傳遞單一 `String` 引數。 或者，您可以 `StartsWithUpper` 在指派給的變數上呼叫做為擴充方法 `string null` 。

您會定義三個方法，每個方法會 [Assert](#) 針對字串陣列中的每個元素呼叫方法。您將呼叫方法多載，讓您指定在測試失敗時所要顯示的錯誤訊息。訊息會識別造成失敗的字串。

建立測試方法：

1. 開啟 *StringLibraryTest/UnitTest1*，並將所有程式碼取代為下列程式碼。


C#	複製
<pre>using Microsoft.VisualStudio.TestTools.UnitTesting; using UtilityLibraries; namespace StringLibraryTest; [TestClass] public class UnitTest1 { [TestMethod] public void TestStartsWithUpper() { // Tests that we expect to return true. string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" }; foreach (var word in words) { bool result = word.StartsWithUpper(); Assert.IsTrue(result, String.Format("Expected for '{0}': true; Actual: {1}", word, result)); } } [TestMethod] public void TestDoesNotStartWithUpper() { // Tests that we expect to return false. string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство", "1234", ".", ";", " " }; foreach (var word in words) { bool result = word.StartsWithUpper(); Assert.IsFalse(result, String.Format("Expected for '{0}': false; Actual: {1}", word, result)); } } [TestMethod]</pre>	

```
public void DirectCallWithNullOrEmpty()
{
    // Tests that we expect to return false.
    string?[] words = { string.Empty, null };
    foreach (var word in words)
    {
        bool result = StringLibrary.StartsWithUpper(word);
        Assert.IsFalse(result,
            String.Format("Expected for '{0}': false; Actual:
{1}",
                                word == null ? "<null>" : word,
                                result));
    }
}
```


方法中的大寫字元測試 `TestStartsWithUpper` 包含希臘文大寫字母 Alpha (u + 0391) 和斯拉夫文大寫字母 EM (U + 041C) 。 方法中的小寫字元測試 `TestDoesNotStartWithUpper` 包含希臘文小寫字母 Alpha (u + 03B1) 和斯拉夫文小寫字母 Ghe (U + 0433) 。

2. 儲存您的變更。

3. 執行測試：

.NET CLI	 複製
<pre>dotnet test StringLibraryTest/StringLibraryTest.csproj</pre>	

終端機輸出顯示所有測試皆通過。

輸出	 複製
<pre>Starting test execution, please wait... A total of 1 test files matched the specified pattern. Passed! - Failed: 0, Passed: 3, Skipped: 0, Total: 3, Duration: 3 ms - StringLibraryTest.dll (net6.0)</pre>	

處理測試失敗

如果您正在執行以測試為導向的開發 (TDD)，您會先撰寫測試，並在您第一次執行時失敗。然後，將程式碼新增至應用程式，讓測試成功。在本教學課程中，您已在撰寫應用程式程式碼驗證之後建立測試，因此您未看到測試失敗。若要在預期測試失敗時驗證測試失敗，請將不正確值加入測試輸入。

1. 修改 `TestDoesNotStartWithUpper` 方法中的 `words` 陣列，以包含字串「錯誤」。

C#	複製
<pre>string[] words = { "alphabet", "Error", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство", "1234", ".", ";", " " };</pre>	

2. 執行測試：

.NET CLI	複製
<pre>dotnet test StringLibraryTest/StringLibraryTest.csproj</pre>	

終端機輸出顯示一個測試失敗，並針對失敗的測試提供錯誤訊息：「`IsFalse` 失敗。'Error' 的預期：`false`；實際：`True`」。因為失敗，所以在測試 "Error" 之後，陣列中沒有任何字串。

輸出	複製
<pre>Starting test execution, please wait... A total of 1 test files matched the specified pattern. Failed TestDoesNotStartWithUpper [28 ms] Error Message: Assert.IsFalse failed. Expected for 'Error': false; Actual: True Stack Trace: at StringLibraryTest.UnitTest1.TestDoesNotStartWithUpper() in C:\ClassLibraryProjects\StringLibraryTest\UnitTest1.cs:line 33 Failed! - Failed: 1, Passed: 2, Skipped: 0, Total: 3, Duration: 31 ms - StringLibraryTest.dll (net5.0)</pre>	

3. 移除您在步驟1中新增的字串 "Error"。重新執行測試和測試階段。

測試程式庫的發行版本

現在，在執行程式庫的偵錯工具時，所有測試都已通過，請針對程式庫的發行組建執行額外的測試。許多因素 (包括編譯器最佳化) 有時會在偵錯和發行組建之間導致不同的行為。

1. 使用發行組建設定執行測試：

.NET CLI	複製
<pre>dotnet test StringLibraryTest/StringLibraryTest.csproj</pre>	

```
--configuration Release
```

所有測試皆通過。

偵錯測試

如果您使用 Visual Studio Code 作為 IDE，您可以使用在使用您的單元測試專案來進行程式碼的偵錯工具時，使用 Visual Studio Code 的相同程式。請開啟 [*StringLibraryTest*]/[*UnitTest1*]，而不是啟動 展示 應用程式專案，而是選取 [偵錯工具行 7和8之間的 所有測試]。如果找不到，請按 **Ctrl** + **Shift** + **P** 開啟命令選擇區，然後輸入 重載視窗。

Visual Studio Code 啟動已附加偵錯工具的測試專案。執行將會在您已新增至測試專案或基礎程式庫程式碼的任何中斷點停止執行。

其他資源

- [.NET 中的單元測試](#)

下一步

在本教學課程中，您已對類別庫進行單元測試。您可以將程式庫以套件的形式發佈至 [NuGet](#)，以供其他人使用。若要深入瞭解，請遵循 [NuGet 教學課程](#)：

使用 **dotnet CLI** 建立及發佈套件

如果您將程式庫發佈為 NuGet 套件，其他人就可以安裝和使用它。若要深入瞭解，請遵循 [NuGet 教學課程](#)：

利用 **dotnet CLI** 安裝並使用套件

程式庫不需要以套件的形式散發。它可以與使用它的主控台應用程式配套。若要瞭解如何發佈主控台應用程式，請參閱本系列的先前教學課程：

使用 **Visual Studio Code** 發佈 .NET 主控台應用程式

快速入門：利用 dotnet CLI 安裝並使用套件

發行項 • 2021/12/02

此頁面有所助益嗎？  

本文內容

[必要條件](#)

[建立專案](#)

[新增 Newtonsoft.Json NuGet 套件](#)

[在應用程式中使用 Newtonsoft.Json API](#)

[相關影片](#)

[下一步](#)

NuGet 套件包含可重複使用的程式碼，由其他開發人員提供您在專案中使用。請參閱[什麼是 NuGet？](#)了解背景知識。您可以使用 `dotnet add package` 命令將套件安裝到 .NET Core 專案中，如本文中針對熱門 `dotnet add package \`(英文)`\` 套件所述的內容。

安裝之後，請使用 `using <namespace>` 參考程式碼中的套件，其中 `<namespace>` 為您使用的套件專用。然後，您可以使用套件的 API。

提示

從 [nuget.org](#) 開始：瀏覽 [nuget.org](#) 是 .NET 開發人員通常用來尋找可在自己應用程式中重複使用之元件的方式。您可以直接搜尋 [nuget.org](#)，或在 Visual Studio 中尋找並安裝套件，如本文所示。


必要條件

- [.NET Core SDK](#)，它會提供 `dotnet` 命令列工具。從 Visual Studio 2017 開始，`dotnet CLI` 會自動與任何 .NET Core 相關工作負載一起安裝。

建立專案

您可以將 NuGet 套件安裝到某種類型的 .NET 專案。針對這個逐步解說，建立簡單的 .NET Core 主控台專案，如下所示：


1. 建立專案的資料夾。
2. 開啟命令提示字元並切換至新的資料夾。
3. 使用下列命令來建立專案：

.NET CLI	 複製
<pre>dotnet new console</pre>	


4. 使用 `dotnet run` 來測試應用程式已正確建立。

新增 Newtonsoft.Json NuGet 套件

1. 使用下列命令來安裝 `Newtonsoft.json` 套件：


.NET CLI	 複製
<pre>dotnet add package Newtonsoft.Json</pre>	

2. 在命令完成之後，開啟 `.csproj` 檔案來查看已新增的參考：


XML	 複製
<pre><ItemGroup> <PackageReference Include="Newtonsoft.Json" Version="12.0.1" /> </ItemGroup></pre>	

在應用程式中使用 Newtonsoft.Json API

1. 開啟 `Program.cs` 檔案，並在檔案頂端新增下列一行：

C#	 複製
<pre>using Newtonsoft.Json;</pre>	


2. 在 `class Program` 行之前，新增下列程式碼：

C#	 複製
<pre>public class Account { public string Name { get; set; } }</pre>	

```
public string Email { get; set; }  
public DateTime DOB { get; set; }  
}
```

3. 使用下列程式碼取代 `Main` 函式：


C#

 複製

```
static void Main(string[] args)  
{  
    Account account = new Account  
    {  
        Name = "John Doe",  
        Email = "john@nuget.org",  
        DOB = new DateTime(1980, 2, 20, 0, 0, 0, DateTimeKind.Utc),  
    };  
  
    string json = JsonConvert.SerializeObject(account,  
        Formatting.Indented);  
    Console.WriteLine(json);  
}
```

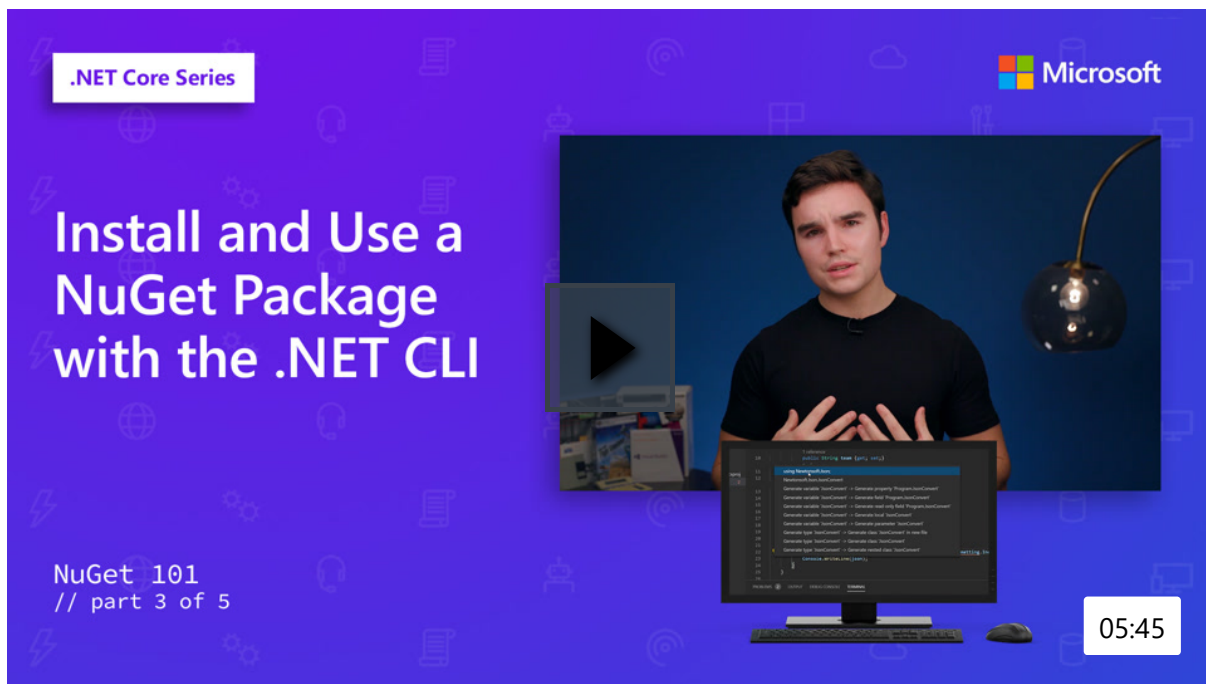
4. 使用 `dotnet run` 命令，建置並執行應用程式。在程式碼中，輸出應該是 `Account` 物件的 JSON 表示法：

輸出

 複製

```
{  
  "Name": "John Doe",  
  "Email": "john@nuget.org",  
  "DOB": "1980-02-20T00:00:00Z"  
}
```

相關影片



在[Channel 9](#) 和[YouTube](#) 上尋找更多 NuGet 影片。

下一步

恭喜，您安裝並使用了您的第一個 NuGet 套件！

使用 **dotnet CLI** 安裝和使用套件

若要深入探索 NuGet 所提供的功能，請選取下列連結。

- [套件耗用量的概觀及工作流程](#)
- [尋找及選擇套件](#)
- [專案檔中的套件參考](#)

建議的內容

使用 [nuget.exe CLI](#) 管理 NuGet 套件

使用 nuget.exe CLI 以處理 NuGet 套件的指示。

使用 [NuGet 套件的概觀和工作流程](#)

在專案中使用 NuGet 套件之程序的概觀，以及程序之其他特定部分的連結。

安裝 **NuGet** 用戶端工具

有關安裝用戶端工具、dotnet 和 nuget 命令列介面 (CLI)，以及適用於 Visual Studio 的套件管理員的指導方針。

NuGet.org 概觀

NuGet.org 概觀

顯示較多 ▼

快速入門：建立及發佈套件 (dotnet CLI)

發行項 • 2021/12/02

此頁面有所助益嗎？  

本文內容

[必要條件](#)

[建立類別庫專案](#)

[將套件中繼資料新增至專案檔](#)

[執行 pack 命令](#)

[發行套件](#)

[相關影片](#)

[下一步](#)

使用 `dotnet` 命令列介面 (CLI) 從 .NET 類別庫建立 NuGet 套件，並將它發行到 nuget.org 是個簡單的程序。

必要條件

1. 安裝 [.NET Core SDK](#)，其中包括 CLI。從 Visual Studio 2017 開始，`dotnet` CLI 會自動與任何 .NET Core 相關工作負載一起安裝。
2. 如果您還沒有帳戶，請在 nuget.org 上註冊一個免費帳戶 [\ \(英文\\)](#)。建立新的帳戶會傳送一封確認電子郵件。您必須確認帳戶，才可以上傳套件。

建立類別庫專案

您可以針對要封裝的程式碼使用現有的 .NET 類別庫專案，或建立一個簡單的專案，如下所示：


1. 建立一個名為 `AppLogger` 的資料夾。
2. 開啟命令提示字元並切換至 `AppLogger` 資料夾。
3. 輸入 `dotnet new classlib`，它會針對專案使用目前資料夾的名稱。

這會建立新的專案。

將套件中繼資料新增至專案檔

每個 NuGet 套件都需要資訊清單來描述套件的內容和相依性。在最後一個套件中，資訊清單是一個 `.nuspec` 檔案，這是從您納入專案檔中的 NuGet 中繼資料屬性所產生的檔案。

1. 開啟您的專案檔 (`.csproj`，`.fsproj` 或 `.vbproj` 根據您所使用的語言)，並在現有的標記內新增下列最基本屬性 `<PropertyGroup>`，並適當地變更值：

XML	 複製
<pre><PackageId>AppLogger</PackageId> <Version>1.0.0</Version> <Authors>your_name</Authors> <Company>your_company</Company></pre>	

📌 重要

為套件指定識別碼，此識別碼在 nuget.org 上或您使用的任何主機上都必須是唯一的。針對此逐步解說，我們建議在名稱中包含 "Sample" 或 "Test" (如同稍後發行步驟所做的)，讓套件能夠成為公開可見的 (儘管實際上不太可能會有人使用它)。


2. 新增任何選擇性的屬性，如 [NuGet 中繼資料屬性](#) 中所述。

📌 注意


針對公眾取用而建置的套件，請特別注意 **PackageTags** 屬性，因為標籤可協助其他人找到您的套件，並了解其用途。

執行 pack 命令

若要從專案建置 NuGet 套件 (`.nupkg` 檔案)，請執行 `dotnet pack` 命令，該命令也會自動建置專案：


.NET CLI	 複製
<pre># Uses the project file in the current folder by default dotnet pack</pre>	

輸出將顯示 `.nupkg` 檔案的路徑：

輸出	 複製
<pre>Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core Copyright (C) Microsoft Corporation. All rights reserved. Restore completed in 29.91 ms for D:\proj\AppLoggerNet\AppLogger \AppLogger.csproj. AppLogger -> D:\proj\AppLoggerNet\AppLogger\bin\Debug\netstandard2.0 \AppLogger.dll Successfully created package 'D:\proj\AppLoggerNet\AppLogger\bin\Debug \AppLogger.1.0.0.nupkg'.</pre>	

建置時自動產生套件

若要在您執行 `dotnet build` 時自動執行 `dotnet pack`，請將下列程式行加入專案檔的 `<PropertyGroup>` 中：

XML	 複製
<pre><GeneratePackageOnBuild>true</GeneratePackageOnBuild></pre>	

發行套件

一旦擁有 `.nupkg` 檔案之後，您會使用 `dotnet nuget push` 命令以及從 nuget.org 取得的 API 金鑰，將它發行到 nuget.org。

ⓘ 注意

病毒掃描：會掃描所有上傳至 nuget.org 的套件是否有病毒，並在發現任何病毒時予以拒絕。也會定期掃描 nuget.org 上列出的所有套件。

除非您取消列出已發行至 nuget.org 的套件，否則也會向其他開發人員公開顯示這些套件。若要私下裝載套件，請參閱裝載套件。

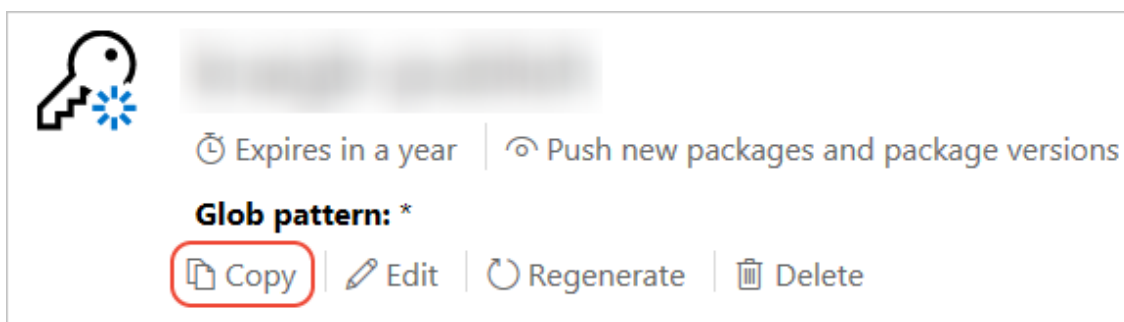
取得 API 金鑰

1. [登入 nuget.org 帳戶](#) \ (英文)，或者，如果您還沒有帳戶，則建立一個帳戶。

如需有關建立帳戶的詳細資訊，請參閱[個人帳戶](#)。

2. 選取您的使用者名稱 (在右上方)，然後選取 [API 金鑰]。

3. 選取 [建立]，為您的金鑰提供名稱，選取 [選取範圍 推播]。針對 [Glob 模式] 輸入 *，然後選取 [建立]。(如需範圍的詳細資訊，請參閱下方)。
4. 建立金鑰之後，選取 [複製] 以擷取 CLI 中您需要的存取金鑰：



⚠ 警告

一律將您的 **API** 金鑰保持為秘密！將您的 **API** 金鑰視為密碼，讓任何人都能代表您管理套件。如果不小心顯示您的 **API** 金鑰，您應該刪除或重新產生它。

📌 重要

將您的金鑰儲存在安全的位置，因為您稍後就無法再次複製此金鑰。如果您返回 **API** 金鑰頁面，則需要重新產生金鑰才能加以複製。如果您不想再推送套件，也可以移除 **API** 金鑰。

設定範圍可讓您針對不同用途建立個別的 **API** 金鑰。每個金鑰都有其到期的時間範圍，且可將範圍設定為特定套件 (或 **Glob** 模式)。每個金鑰也可將範圍設定為特定作業：新套件和更新的推送、僅更新的推送，或取消列入。透過設定範圍，您就可針對組織中管理套件的不同人員建立 **API** 金鑰，讓他們只具有所需的權限。如需詳細資訊，請參閱[限定範圍的 API 金鑰](#)。

使用 dotnet nuget push 發行

1. 變更為包含 `.nupkg` 檔案的資料夾。
2. 執行下列命令，指定套件名稱 (使用套件識別碼) 並使用您的 **API** 金鑰來取代金鑰值：

```
.NET CLI
```

 複製

```
dotnet nuget push AppLogger.1.0.0.nupkg --api-key  
qz2jga8p13dvn2akksyquwcs9ygggg4exypy3bhxy6w6x6 --source  
https://api.nuget.org/v3/index.json
```

3. dotnet 會顯示發程序的结果：


輸出	 複製
<pre>info : Pushing AppLogger.1.0.0.nupkg to 'https://www.nuget.org/api/v2 /package'... info : PUT https://www.nuget.org/api/v2/package/ info : Created https://www.nuget.org/api/v2/package/ 12620ms info : Your package was pushed.</pre>	

請參閱 [dotnet nuget push](#)。

發行錯誤

來自 `push` 命令的錯誤通常代表發生問題。例如，您可能忘記更新專案中的版本號碼，因而嘗試發行已經存在的套件。

當您嘗試使用主機上已經存在的識別碼來發行套件時，也會看到錯誤。例如，名稱 "AppLogger" 已經存在。在這種情況下，`push` 命令會產生下列錯誤：

輸出	 複製
<pre>Response status code does not indicate success: 403 (The specified API key is invalid, has expired, or does not have permission to access the specified package.).</pre>	

如果您使用剛建立的有效 API 金鑰，則此訊息表示命名衝突，這無法從錯誤的「權限」部分中完全清除。請變更套件識別碼、重新建置專案、重新建立 `.nupkg` 檔案，然後重試 `push` 命令。

管理已發行的套件

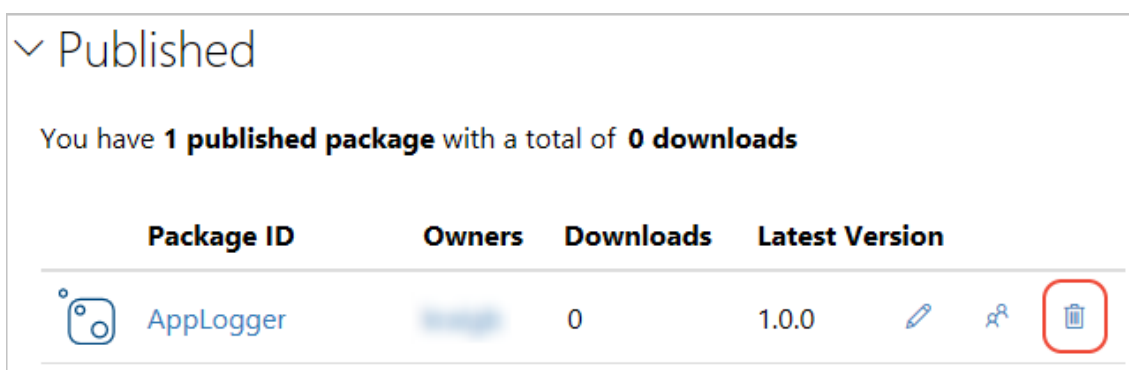
從 [nuget.org](#) 的設定檔中，選取 [管理封裝] 以查看您剛剛發行的套件。您也會收到確認電子郵件。請注意，可能需要一些時間才會編製套件的索引，並顯示在其他人可以找到它的搜尋結果中。在這段期間，套件頁面會顯示下列訊息：

⚠ **This package has not been published yet.** It will appear in search results and will be available for install/restore after both validation and indexing are complete. Package validation and indexing may take up to an hour. [Read more.](#)

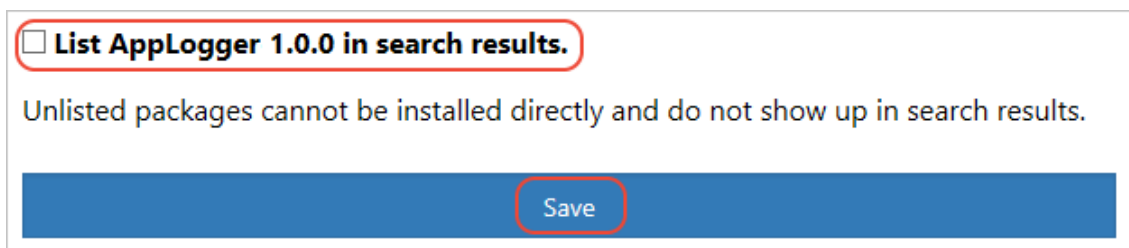
這樣就大功告成了！您剛剛已將第一個 NuGet 套件發行至 nuget.org，其他開發人員可以在他們自己的專案中使用它。

如果您在本逐步解說中建立的套件不是真的很實用 (例如，使用空類別庫建立的套件)，則您應該「取消列出」該套件，以便在搜尋結果中加以隱藏：

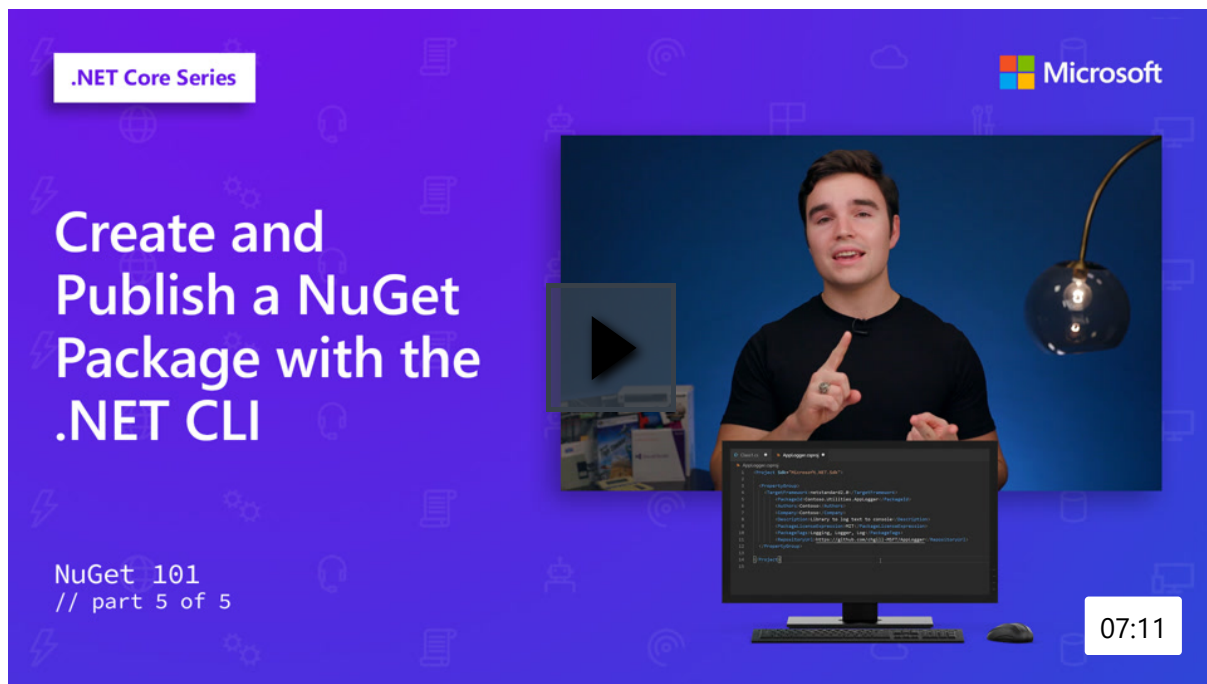
1. 在 nuget.org，選取您的使用者名稱 (頁面右上方)，然後選取 [管理套件]。
2. 在 [已發行] 下方找出您想要取消列出的套件，然後選取右邊的資源回收筒圖示：



3. 在後續頁面上，清除標籤為 [在搜尋結果中列出 (package-name)] 的方塊，然後選取 [儲存]：



相關影片



在 [Channel 9](#) 和 [YouTube](#) 上尋找更多 NuGet 影片。

下一步

恭喜，您建立了您的第一個 NuGet 套件！

建立套件

若要深入探索 NuGet 所提供的功能，請選取下列連結。

- [發佈封裝](#)
- [發行前套件](#)
- [支援多個目標架構](#)
- [套件版本控制](#)
- [新增授權運算式或檔案](#)
- [建立當地語系化的套件](#)
- [建立符號套件](#)
- [正在簽署套件](#)

建議的內容

[dotnet CLI NuGet 命令](#)

使用 dotnet 命令列介面之 NuGet 相關命令的簡短參考。

如何發佈 NuGet 套件

如何將 NuGet 套件發行至 [nuget.org](https://www.nuget.org) 或私用摘要以及如何在 [nuget.org](https://www.nuget.org) 上管理套件擁有權的詳細指示。

使用 dotnet CLI 建立 NuGet 套件

NuGet 套件設計和建立程序詳細指南，包含檔案和版本控制這類索引鍵決策點。

設定本機 NuGet 摘要

如何使用區域網路上的資料夾建立 NuGet 套件的本機摘要

顯示較多 ▼