.NET 將多個程式集合併成單一程式集的4+3 種方法

呂毅 • 發表於2019-06-30 • 更新於2020-01-10

寫.NET 程式的時候，我們常常會在專案的輸出目錄下發現一大堆的檔案。除了我們專案自己產生的組件之外，還能找到這個專案所依賴的一大堆依賴組件。有沒有什麼方法可以把這些依賴和我們的程式集合並在一起呢？

本文介紹四種將程序集和依賴打包合併在一起的方法，每種方法都有其不同的原理和優缺點。我將介紹這些方法的原理並幫助你決定哪種方法最適合你想要使用的場景。

四種方法

目前我已知的將.NET 組件與相依性合併到一起的方法有以下四種：

1. 使用.NET Core 3.0 自帶的PublishSingleFile 屬性合併依賴
2. 使用Fody
3. 使用SourceYard 原始碼包
4. 使用ILMerge（微軟所寫）或ILRepack（基於Mono.Ceil）
5. 其他方法

如果你還知道有其他的方法，歡迎留言指出，非常感謝！

上面的第五種方法我也會做一些介紹，要么是因為無法真正完成任務或者適用場景非常有限，要么是其原理我還不理解，因此只進行簡單介紹。

使用.NET Core 3.0 自帶的PublishSingleFile 屬性合併依賴

.NET Core 3.0 自Preview 5 開始，增加了發佈成單一exe 檔案的功能。

在你的專案文件中增加下面的兩行可以開啟此功能：

Diff

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

      <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>netcoreapp3.0</TargetFramework>
++      <RuntimeIdentifier>win10-x64</RuntimeIdentifier>
++      <PublishSingleFile>true</PublishSingleFile>
      </PropertyGroup>

    </Project>
```

第一行 `RuntimeIdentifier` 一定需要指定，因為發布的單一文件是特定於架構的。這裡，我們指定了win10-x64，你也可以指定為其他的值。可以使用的值你可以在這篇文章中查詢：

- .NET Core Runtime IDentifier (RID) catalog - Microsoft Docs

第二行 `PublishSingleFile` 即開啟發佈時單一檔案的功能。這樣，你在發布你的程式的時候可以得到一個單一的可執行程式。發布一個.NET Core 專案的方法是在命令列中輸入：

```
dotnet publish
```

當然，如果你沒有更改任何你的專案檔案（沒有增加上面的那兩行），那麼你在使用發布命令的時候就需要把這兩個屬性再增加上。因此完整的發布命令是下面這樣的：

Powershell

```
dotnet publish -r win10-x64 /p:PublishSingleFile=true
```

這裡的 `-r` 就等同於在專案中指定 `RuntimeIdentifier` 持續。這裡的 `/p` 是在項目中增加一個屬性，而增加的屬性名是 `PublishSingleFile` ，增加的屬性值是 `true` 。

使用.NET Core 3.0 這種自帶的發佈單一exe 的方法會將你的程式的全部文件（包括所有依賴文件，包括非託管組件，包括各種資源文件）全部打包到一個exe 中。當執行這個exe 的時候，會先將所有這些檔案產生到本機中一個暫存目錄下。只有第一次執行這個exe 的時候才會產生這個目錄和其中的文件，之後的執行是不會再產生的。

以下說一些.NET Core 3.0 發佈組件的一點擴充 - .NET Core 3.0 中對於發佈組件的三種處理方式可以放在一起使用：

- 裁切程序集（Assembly Trimmer）
- 提前編譯（Ahead-of-Time compilation，透過crossgen）*後面馬上會說到Microsoft.DotNet.ILCompiler*
- 單一檔案打包（Single File Bundling）*本小節*

關於.NET Core 3.0 中發布僅一個exe 的方法、原則和實踐，可以參見林德熙的部落格：

- dotnet core 發布只有一個exe 的方法

.NET Core 在GitHub 上開源：

- .NET Foundation

使用Fody

在你的專案中安裝一個NuGet 套件Costura.Fody。一般來說，安裝完之後，你編譯的時候就會產生僅有一個exe 的程式集了。

如果你繼續留意，可以發現專案中多了一個Fody 的專屬配置文件 `FodyWeavers.xml` ，內容如下：

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Weavers>
    <Costura/>
</Weavers>
```

僅僅到此為止你已經足夠利用Fody 完成程序集的合併了。

但是，如果希望對Fody 進行更精細化的配置，可以閱讀葉洪的部落格：

- .NET 合併組件（將dll 合併到exe 中） - Iron 的博客- CSDN博客

Fody 在GitHub 上開源：

- Fody/Fody: Extensible tool for weaving .net assemblies

SourceYard 原始碼包在程式集合併上是另闢蹊徑的合併方式。它不能幫助你將所有的依賴全部合併，但足以讓你在發布一些簡單應用程式的時候不引入大量的依賴。

例如，你可以考慮新建一個項目，然後安装下面的NuGet 套件：

- lindexi.src.MacAddress.Source

安裝完成之後，你就可以在你的專案中使用到此NuGet 套件為你帶來的取得MAC 位址的工具類別了。

C#

```C#
using System;
using lindexi.src;

namespace Walterlv.Demo
{
    internal static class Program
    {
        static void Main()
        {
            var macList = MacAddress.GetActiveMacAddress();
            foreach (var mac in macList)
            {
                Console.WriteLine(mac);
            }
        }
    }
}
```

編譯完你的項目，你會發現你的專案沒有攜帶任何依賴。你安裝的NuGet 套件並沒有成為你的依賴，反而成為你正在編譯的組件的一部分。

如果你要製作一個像上面那樣的原始碼包，只需要在你要製作NuGet 包的專案安裝上dotnetCampus.SourceYard，在你打包成NuGet 包的時候，就會產生一個普通的NuGet 包以及一個*. Source.nupkg 的原始碼包。將原始碼包上傳到nuget.org 上，其他人便可以安裝你製作的原始碼包了。

關於如何使用SourceYard 製作一個原始碼包的方法可以閱讀林德熙的部落格：

- SourceYard 製作原始碼包

關於能夠做出原始碼包的原理，可以閱讀我的部落格：

- 入門篇：將.NET Core 專案打一個最簡單的NuGet 原始碼包，安裝此包就像直接把源碼放進專案一樣
- 進階篇：從零開始製作NuGet 原始碼套件（全面支援.NET Core / .NET Framework / WPF 專案）

SourceYard 在GitHub 上開源：

- dotnet-campus/SourceYard: Add a NuGet package only for dll reference? By using dotnetCampus.SourceYard, you can pack a NuGet package with source code. By installing the new source code package, all source .

使用ILMerge 或ILRepack 等工具

ILMerge 和ILRepack 的合併就更加富有技術含量——當然坑也更多。

這兩個都是工具，因此，你需要將工具下載下來使用。你有很多種方法下載到工具使用，因此我會推薦不同的人群使用不同的工具。

ILMerge 命令列工具是微軟官方出品，下載網址：

- Download ILMerge 來自 Official Microsoft Download Center

其使用方法請參閱我的部落格：

- .NET 使用ILMerge 合併多個程序集，避免引入額外的依賴- walterlv

ILRepack

ILRepack 是基於Mono.Ceil 來進行IL 合併，其使用方法可以參考我的部落格：

- .NET 使用ILRepack 合併多個組件（替代ILMerge），避免引入額外的依賴- walterlv

ILMerge-GUI 工具（已過時，但適合新手隨便玩玩）

你可以在以下網址中找到ILMerge-GUI 的下載連結：

- wvd-vegt / ilmergegui / Downloads — Bitbucket

ILMerge-GUI 工具在Bitbucket 上開源：

- wvd-vegt / ilmergegui — Bitbucket

其他方法

使用Microsoft.DotNet.ILCompiler

可以將.NET Core 編譯為單一無依賴的Native 程式。

你需要先安裝一個預覽版的NuGet 套件Microsoft.DotNet.ILCompiler

關於Microsoft.DotNet.ILCompiler 的使用，你可以閱讀林德熙的部落格：

- dotnet core 使用CoreRT 將程式編譯為Native 程式

使用dnSpy

dnSpy 支援新增一個模組到組件，也可以建立模組，也可以將組件轉換為模組。因此，一個程式集可以包含多個模組的功能就可以被充分利用。

Warp 在GitHub 上開源：

- dgiagio/warp: Create self-contained single binary applications

其使用可參見林德熙的部落格：

- dotnet core 發布只有一個exe 的方法

各種方法的原理與使用情境比較

原理

使用.NET Core 3.0 自帶的 `PublishSingleFile` 屬性合併依賴，其原理是產生一個啟動器容器程式。最終沒有對程序進行任何修改，只是單純的打包而已。

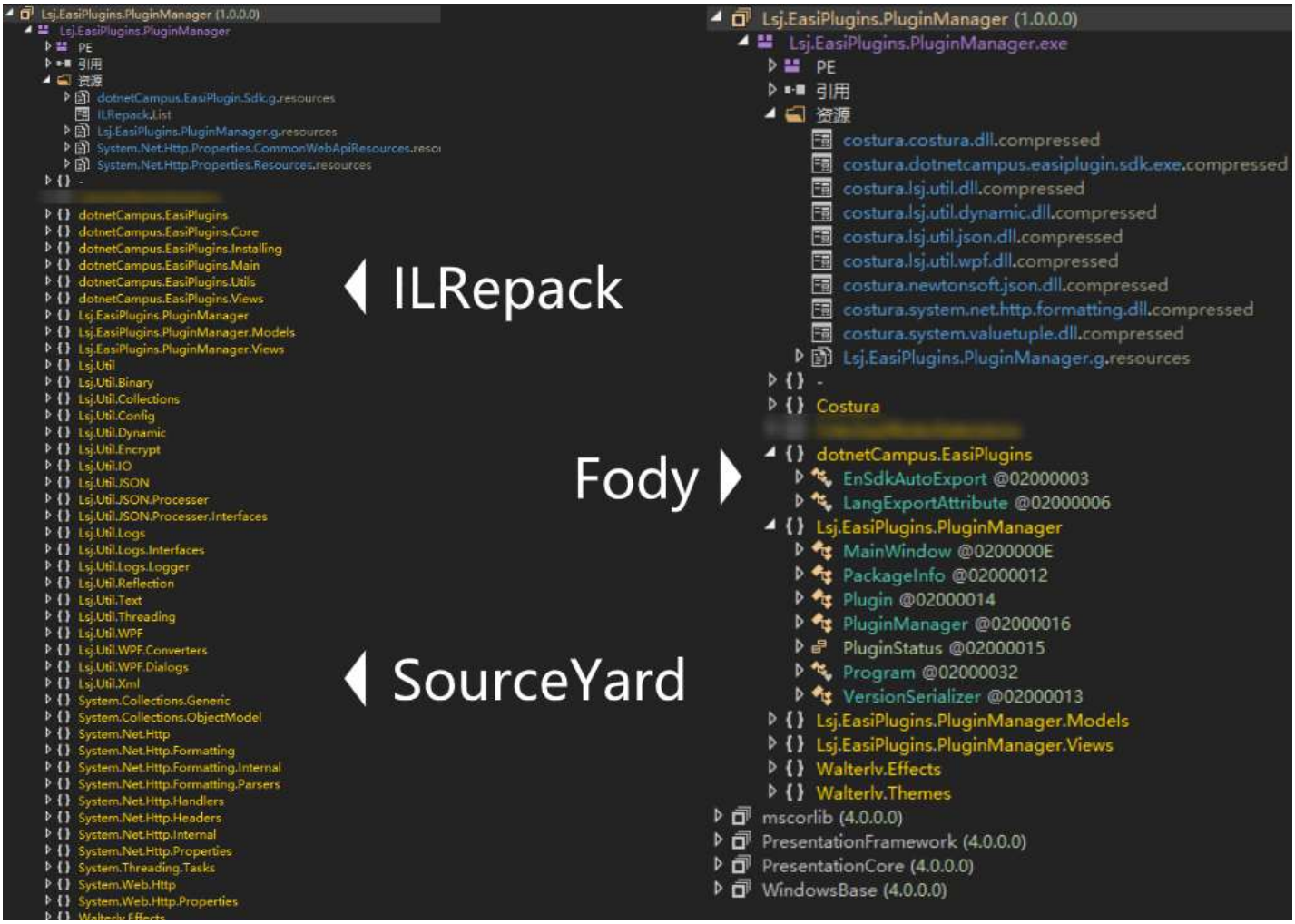
使用Fody，是將程式集依賴放到了資源裡面。當要載入組件的時候，會直接將資源中的程序集流載入到記憶體中。

使用SourceYard 原始碼包，是直接將原始碼合併到了目標專案裡面。

使用ILMerge / ILRepack，是在IL 層級對程序集進行了合併。

我們可以透過下面一張圖來感受一下後三種原理上的差異。

這是一個分別透過Fody、SourceYard 和ILMerge / ILRepack 產生的組件的反編譯圖。可以看到，對於ILRepack / ILMerge 和SourceYard，反編譯後看到的原始碼都在目標程式集中，而對於Fody，依賴僅出現在資源中。



由於其原理不同，所以其適用範圍和造成的副作用也不同。

如果你基於.NET Core 3.0 開發，並且也不在意在目標電腦上產生的臨時資料夾，那麼可以考慮使用 `PublishSingleFile` 屬性合併依賴。

如果你不在乎啟動效能以及記憶體消耗，那麼可以考慮Fody（這意味著小型程式比較適合採用）。

如果你的程式非常在乎啟動效能，那就需要考慮SourceYard、ILMerge / ILRepack 了。

對於ILMerge / ILRepack 和SourceYard 的比較，可以看下面這張表格：

方案	ILRepack / ILMerge	SourceYard
適用於	任意.NET 程式集	透過SourceYard 發布的NuGet 包
WPF	ILRepack 支持，ILMerge 不支持	支援
調試（支援）	僅支援一般方法的調試	支援一般程序集支援的所有調試方法
調試（不支援）	不支援非同步方法調試，不支援顯示局部變數	沒有不支持的
隱藏API	internal 的類型和成員可以隱藏	必須是private 類型和成員才能隱藏

可以發現，如果我們能夠充分將我們需要的套件透過SourceYard 發佈成NuGet，那麼我們將可以獲得比ILRepack / ILMerge 更好的編寫和調試體驗。

表格之外還有一些特別需要說明的：

1. ILRepack 額外支援修改WPF 編譯產生的Baml 文件，將資源的參考路徑修改成新組件的路徑。
2. SourceYard 的類型需要寫成private 才可以隱藏，但是只有內部類別才可以寫private，因此如果特別需要隱藏，請先寫一個內部類別。（因此，你可能會發現有一個型別有很多個分部類，每一個分部類中都是一個私有的內部類別）

開源社群

最後說一下，以上所說的所有方法全部是開源的，有問題歡迎在社群討論一起解決：

- .NET Foundation
- Fody/Fody
- dotnet-campus/SourceYard
- dotnet/ILMerge
- gluck/il-repack
- 0xd4d/dnSpy
- dgiagio/warp
- corert/pkg/Microsoft.DotNet.ILCompiler

本文會經常更新，請閱讀原文：https://blog.walterlv.com/post/how-to-merge-dotnet-assemblies.html，以避免陳舊錯誤知識的誤導，同時有更好的閱讀體驗。

登入GitHub 帳號進行評論

分享:

0 Comments - *powered by utteranc.es*

Write Preview

Sign in to comment

Styling with Markdown is supported

Sign in with GitHub