

使用 .NET Core CLI 实现 .NET Core 入门

2019/12/05 • 👤 👤

本文内容

[先决条件](#)

[Hello，控制台应用！](#)

[修改程序](#)

[使用多个文件](#)

[发布你的应用](#)

[结束语](#)

[请参阅](#)

本文介绍如何开始使用 .NET Core CLI 开发在 Windows、Linux 和 macOS 上运行的 .NET Core 应用。

如果不熟悉 .NET Core CLI，请参阅 [.NET Core 概述](#)。


先决条件

- [.NET Core SDK 3.1](#) 或更高版本。
- 按需选择的文本编辑器或代码编辑器。

Hello，控制台应用！

若要[查看或下载示例代码](#)，可以访问 dotnet/samples GitHub 存储库。有关下载说明，请参阅[示例和教程](#)。

打开命令提示符，创建一个名为“Hello”的文件夹。导航到创建的文件夹，键入下列内容：

.NET Core CLI	 复制
<pre>dotnet new console dotnet run</pre>	

让我们进行快速演练：

1. dotnet new console

[dotnet new](#) 会创建一个最新的 Hello.csproj 项目文件，其中包含生成控制台应用所必需的依赖项。此外，它还会创建一个 Program.cs，这是包含应用程序入口点的基本文件。

Hello.csproj：

XML	复制
<pre><Project Sdk="Microsoft.NET.Sdk"> <PropertyGroup> <OutputType>Exe</OutputType> <TargetFramework>netcoreapp2.2</TargetFramework> </PropertyGroup> </Project></pre>	

项目文件指定还原依赖项和生成程序所需的一切。

- <OutputType> 元素指定我们要生成的可执行文件，即控制台应用程序。
- <TargetFramework> 元素指定要定位的 .NET 实现代码。在高级方案中，可以指定多个目标框架，并在单个操作中生成所有目标框架。在本教程中，我们将仅针对 .NET Core 3.1 进行生成。

Program.cs：

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

该程序从 using System 开始，这意味着“将 System 命名空间中的所有内容都纳入此文件的作用域”。System 命名空间包括 Console 类。


接着定义一个名为 Hello 的命名空间。你可以将其更改为任何你喜欢的名称。在该命名空间中定义了一个名为 Program 的类，其中 Main 方法采用名为 args 的字符串数组。此数组包含在运行程序时所传递的参数列表。实际上，不使用此数

组，程序只会写入文本“Hello World!” “Hello World!”。稍后将对使用此参数的代码进行更改。


`dotnet new` 隐式调用 [dotnet restore](#)。`dotnet restore` 调用到 [NuGet](#)（.NET 包管理器）以还原依赖项树。NuGet 分析 `Hello.csproj` 文件、下载文件中定义的依赖项（或从计算机缓存中获取）并编写 `obj/project.assets.json` 文件，在编译和运行示例时需要使用该文件。

2. dotnet run

[dotnet run](#) 调用 [dotnet build](#) 来确保已生成要生成的目标，然后调用 `dotnet <assembly.dll>` 运行目标应用程序。

console	 复制
<pre>dotnet run Hello World!</pre>	

或者，还可以运行 `dotnet build` 来编译代码，无需运行已生成的控制台应用程序。这会基于项目的名称将已编译的应用程序作为 DLL 文件生成。在这种情况下，创建的文件命名为 `Hello.dll`。此应用可以使用 Windows 上的 `dotnet bin\Debug\netcoreapp3.1\Hello.dll` 运行（非 Windows 系统使用 / ）。

console	 复制
<pre>dotnet bin\Debug\netcoreapp3.1\Hello.dll Hello World!</pre>	


在编译应用时，会随 `Hello.dll` 一起创建特定于操作系统的可执行文件。在 Windows 上，这将是 `Hello.exe`；在 Linux 或 macOS 上，这将是 `hello`。在上面的示例中，用 `Hello.exe` 或 `Hello` 命名该文件。可以直接运行该可执行文件。

console	 复制
<pre>.\bin\Debug\netcoreapp3.1\Hello.exe Hello World!</pre>	

修改程序

让我们稍微更改一下程序。Fibonacci 数字很有意思，那么除了使用参数，让我们也来添加 Fibonacci 数字，让运行应用的用户开心一下。

1. 将 *Program.cs* 文件的内容替换为以下代码：

C#	 复制
<pre>using System; namespace Hello { class Program { static void Main(string[] args) { if (args.Length > 0) { Console.WriteLine(\$"Hello {args[0]}!"); } else { Console.WriteLine("Hello!"); } Console.WriteLine("Fibonacci Numbers 1-15:"); for (int i = 0; i < 15; i++) { Console.WriteLine(\$"{i + 1}: {FibonacciNumber(i)}"); } } static int FibonacciNumber(int n) { int a = 0; int b = 1; int tmp; for (int i = 0; i < n; i++) { tmp = a; a = b; b += tmp; } return a; } } }</pre>	

2. 运行 [dotnet build](#) 以编译更改。

3. 运行向应用传递参数的程序。使用 `dotnet` 命令运行应用时，将 `--` 添加到末尾。
`--` 右侧的任何内容都将作为参数传递到应用。在下面的示例中，值 `John` 传递到应用。


console	 复制
<pre>\$ dotnet run -- John Hello John! Fibonacci Numbers 1-15: 1: 0 2: 1 3: 1 4: 2 5: 3 6: 5 7: 8 8: 13 9: 21 10: 34 11: 55 12: 89 13: 144 14: 233 15: 377</pre>	

就是这么简单！可以按任意喜欢的方式修改 `Program.cs`。

使用多个文件

单个文件适用于简单的一次性程序，但如果要构建较为复杂的应用，则项目中可能会有多个代码文件。我们通过缓存一些 Fibonacci 值并添加一些递归特性来基于之前的 Fibonacci 示例进行构建。

1. 使用以下代码将新文件添加到名为 `FibonacciGenerator.cs` 的 `Hello` 目录：

C#	 复制
<pre>using System; using System.Collections.Generic; namespace Hello { public class FibonacciGenerator { private Dictionary<int, int> _cache = new Dictionary<int, int> (); private int Fib(int n) => n < 2 ? n : FibValue(n - 1) + FibValue(n - 2); } }</pre>	

```


private int FibValue(int n)
{
    if (!_cache.ContainsKey(n))
    {
        _cache.Add(n, Fib(n));
    }

    return _cache[n];
}


public IEnumerable<int> Generate(int n)
{
    for (int i = 0; i < n; i++)
    {
        yield return FibValue(i);
    }
}
}

```

2. 更改 `Program.cs` 文件中的 `Main` 方法，以实例化新的类并调用其方法，如下例所示：

C#	 复制
<pre> using System; namespace Hello { class Program { static void Main(string[] args) { var generator = new FibonacciGenerator(); foreach (var digit in generator.Generate(15)) { Console.WriteLine(digit); } } } } </pre>	


3. 运行 [dotnet build](#) 以编译更改。
4. 通过执行 [dotnet run](#) 来运行应用。 以下是程序输出：

console	 复制
<pre> \$ dotnet run 0 </pre>	

```
1
1
2
3
5
8
13
21
34
55
89
144
233
377
```

发布你的应用

准备好分发应用后，使用 [dotnet publish](#) 命令在 `bin\debug\netcoreapp3.1\publish\`（非 Windows 系统使用 `/`）处生成 `publish` 文件夹。可以将 `publish` 文件夹的内容分发到其他平台，只要这些平台安装了 `dotnet` 运行时即可。

console	 复制
<pre>dotnet publish Microsoft (R) Build Engine version 16.4.0+e901037fe for .NET Core Copyright (C) Microsoft Corporation. All rights reserved. Restore completed in 20 ms for C:\Code\Temp\Hello\Hello.csproj. Hello -> C:\Code\Temp\Hello\bin\Debug\netcoreapp3.1\Hello.dll Hello -> C:\Code\Temp\Hello\bin\Debug\netcoreapp3.1\publish\</pre>	

上面的输出可能会因当前文件夹和操作系统而有所不同，但输出应类似。

可以使用 [dotnet](#) 命令运行已发布的应用：

console	 复制
<pre>dotnet bin\Debug\netcoreapp3.1\publish\Hello.dll Hello World!</pre>	

如本文开头处所述，会随 `Hello.dll` 一起创建特定于操作系统的可执行文件。在 Windows 上，这将是 `Hello.exe`；在 Linux 或 macOS 上，这将是 `hello`。在上面的示例中，用 `Hello.exe` 或 `Hello` 命名该文件。可以直接运行已发布的可执行文件。

console	 复制
---------	--

```
.\bin\Debug\netcoreapp3.1\publish\Hello.exe
```

```
Hello World!
```

结束语

就是这么简单！现在，可以开始使用此处学到的基本概念来创建自己的程序了。

请参阅

- [使用 .NET Core CLI 组织和测试项目](#)
- [使用 .NET Core CLI 发布 .NET Core 应用](#)
- [.NET Core 应用程序部署](#)

此页面有帮助吗？

 是  否
