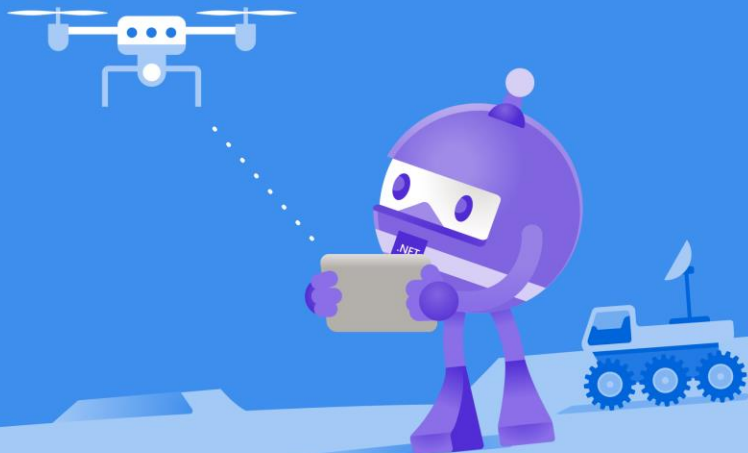


.NET Conf

探索 .NET 新世界



The Journey of Source Generator

Roberson Liou





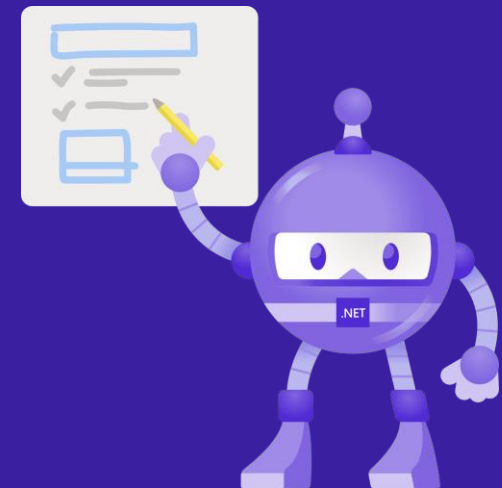
關於我

- 手沖咖啡科技宅
- Backend Engineer
- Microsoft MVP(2020-2021)
- twMVC 核心成員
- DevOps Taiwan 志工
- 工程良田的小球場



Outline

- Introduce Source Generator
- How to use Source Generator?
- Demo



Introduce Source Generator





What's the Source Generator?

- .NET 5 釋出的一個 Roslyn 新項目
- 在編譯階段產生程式碼
- 附加至最後的編譯結果
- Compile-time Meta Programing
 - Meta 來源：SyntaxTree、File、Config

What CAN it do?

- 取得 Roslyn 編譯後的 **Compilation**
- 取得外部專案的附加檔案
- 取得 **Config** 的自定義資訊
- 輸出實體檔案到指定資料夾中
- 與 **IDE**整合輸出診斷資訊

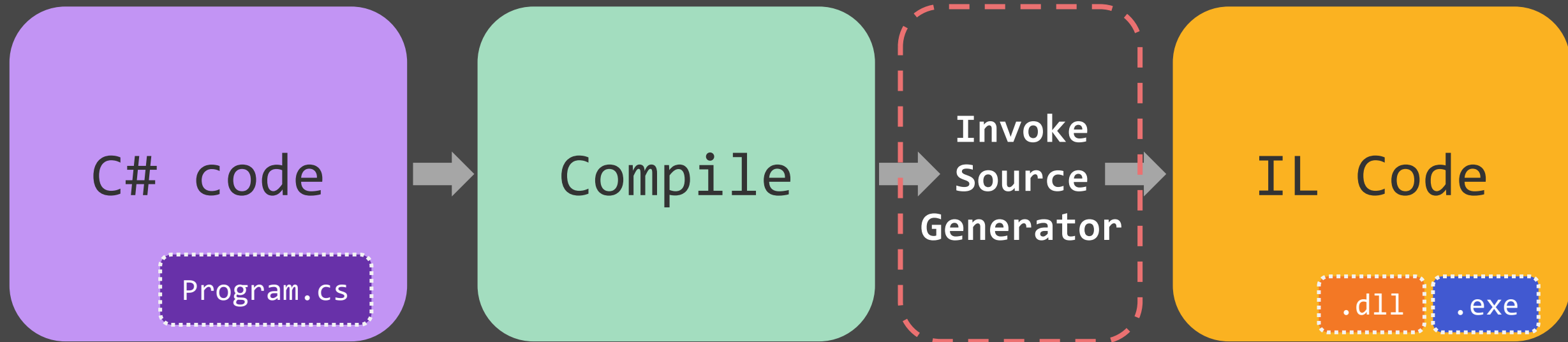
What can **NOT** it do?

- 不能修改原本的程式碼
 - 無法達到 AOP 效果
- 可以有多個 Generator，但彼此間**不能**相互參考
- 不能產生 C# 程式碼以外的檔案

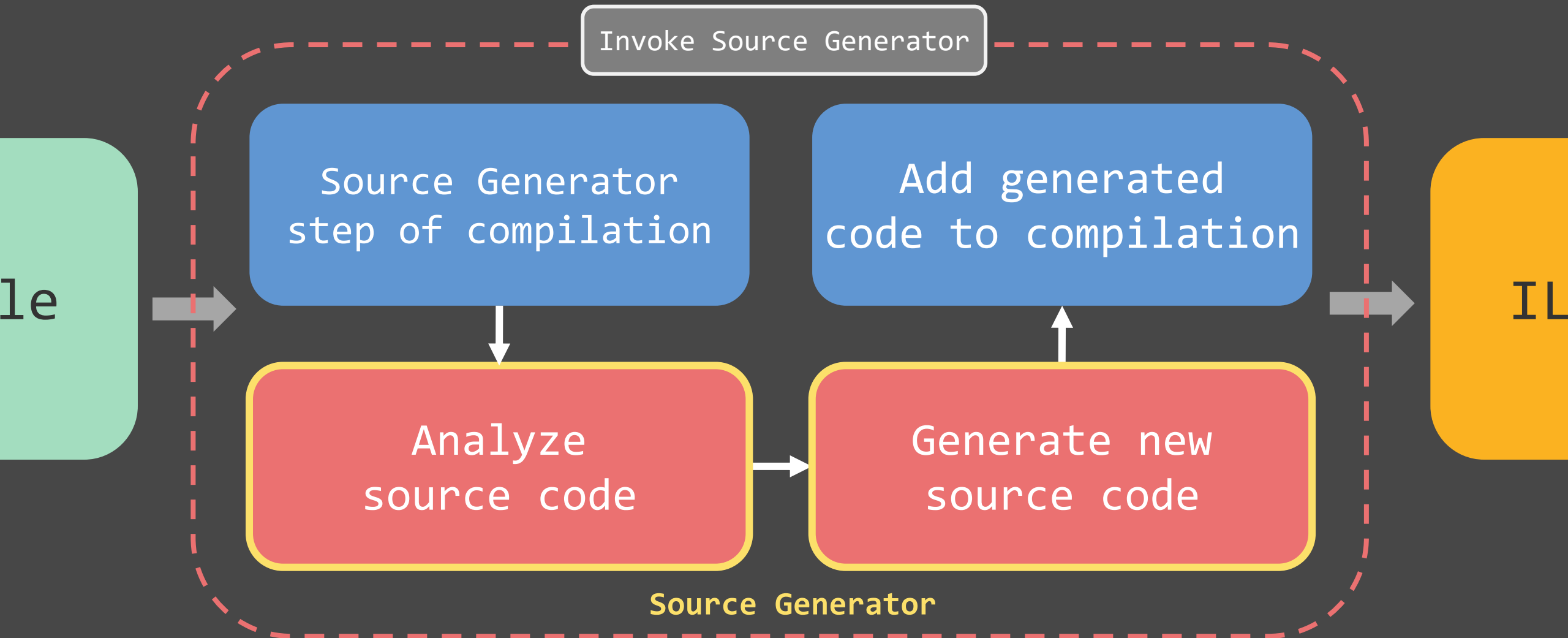
Lifecycle - C# Compilation



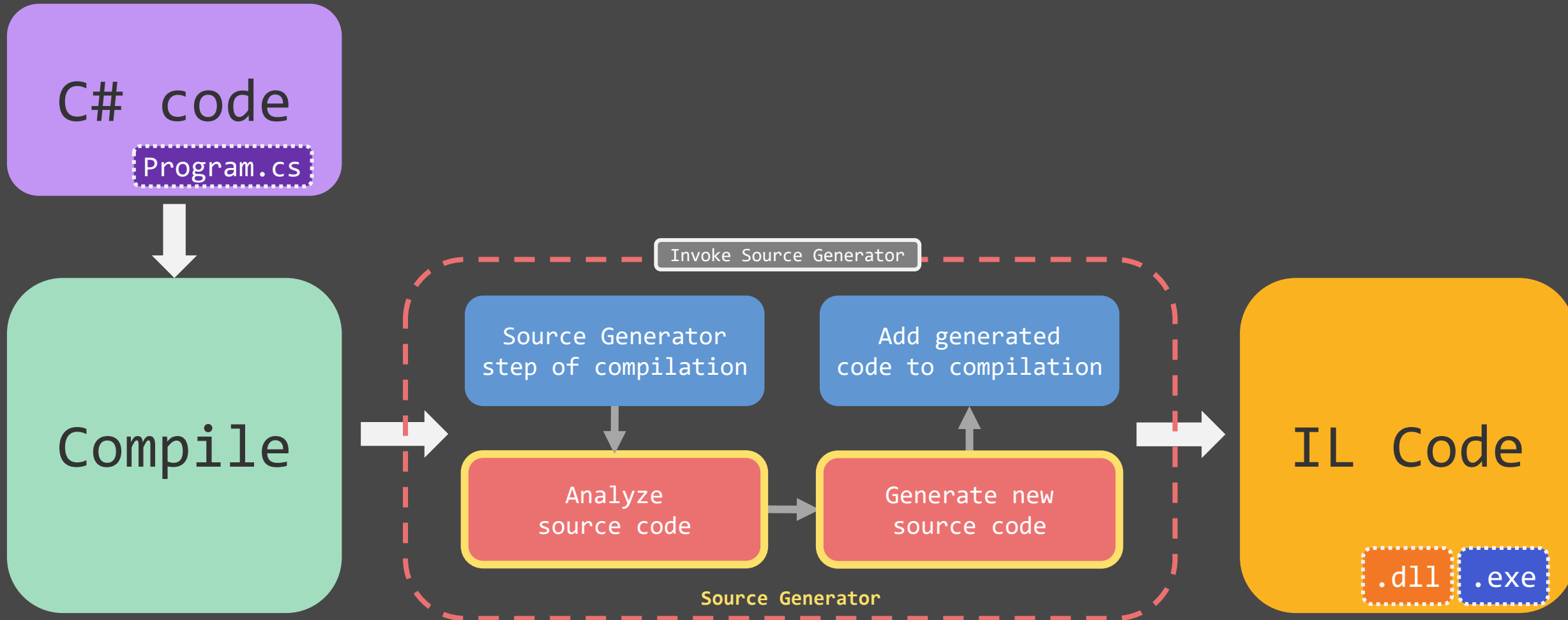
Lifecycle - Invoke Source Generator



Lifecycle - Source Generator



Lifecycle - Overall



Benefits

- 將 Runtime reflection 轉為 Compile-time
 - 提升效能
- 處理序列化檔案
 - .csv / .xml
- 擴增原始程式碼
 - partial class / partial method

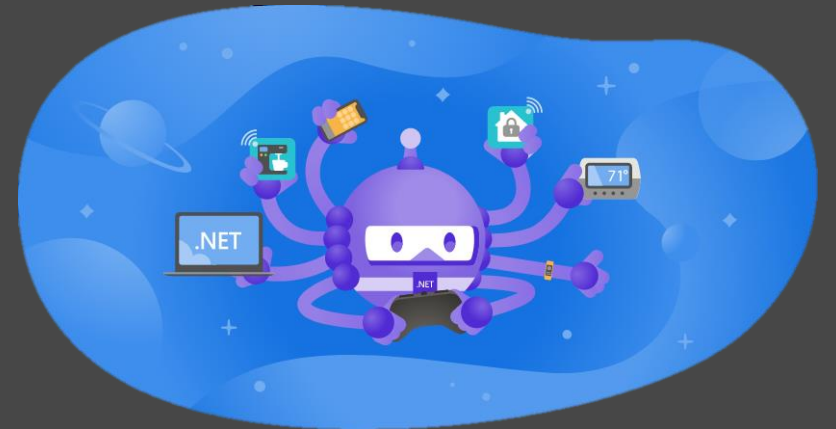
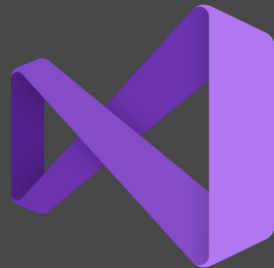


How to use Source Generator?



執行環境

- .NET 5 (SDK 5.0.100)
- Visual Studio 2019 (16.8.0+)
- JetBrains Rider 2020.3 / EAP 10



Visual Studio 2019

- 需手動叫用 Debugger 偵錯
- 可查看產生的程式碼定義
 - 會自動標註 **auto-generated**
- 重新建置後會有快取問題，要重開 VS 才能解決

BotAttribute.cs [generated] ✎ ✕

This file is auto-generated by the generator 'SourceGeneratorLib.Generators.MyGenerator' and cannot be edited.



JetBrains Rider EAP 10 / 2020.3

- 需透過 VS 2019 才能偵錯
- 可查看產生的程式碼定義
 - 不會自動標註 auto-generated
- 重新建置後即可查看新的建置程式碼





Generator 起手式

- 建立 **.NET Standard 2.0** 函式庫
- 安裝套件
 - Microsoft.CodeAnalysis.CSharp (v3.8.0)
 - Microsoft.CodeAnalysis.Analyzers (v3.3.1)

```
<ItemGroup>  
  <PackageReference Include="Microsoft.CodeAnalysis.CSharp"  
    Version="3.8.0" />  
  <PackageReference Include="Microsoft.CodeAnalysis.Analyzers"  
    Version="3.3.1" />  
</ItemGroup>
```




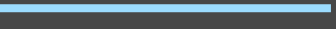
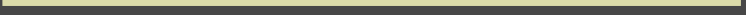
在專案中啟用 Generator

- 加入專案參考，需額外加入以下兩個屬性。
 - `OutputItemType="Analyzer"`
 - `ReferenceOutputAssembly="false"`

```
<ItemGroup>  
  <ProjectReference Include="..\MyGeneratorLib\MyGeneratorLib.csproj"  
    OutputItemType="Analyzer" ReferenceOutputAssembly="false" />  
</ItemGroup>
```

- 使用 NuGet 套件安裝 Generator 則無須另外設定。

Generator 核心成員

- `ISourceGenerator` 
- `GeneratorAttribute` 
- `GeneratorInitializationContext`  初始
- `GeneratorExecutionContext`  執行
- `ISyntaxReceiver`  篩選 **Syntax**

實作 Generator

- 於類別掛上 `[Generator]`
- 實作 `ISourceGenerator`
 - `Initialize`
 - `Execute`

```
[Generator]
public class MyGenerator : ISourceGenerator
{
    public void Initialize(GeneratorInitializationContext context){}
    public void Execute(GeneratorExecutionContext context){}
}
```

Generator - Initialize

- 主要成員為 **GeneratorInitializationContext**
- **context** 中具有一個 **CancellationToken**
- **RegisterForSyntaxNotifications** 可用來註冊 **ISyntaxReceiver**

```
public void Initialize  
    (GeneratorInitializationContext context)  
{  
    context.RegisterForSyntaxNotifications(  
        () => new MySyntaxReceiver());  
}
```



Example - MySyntaxReceiver

```
public class MySyntaxReceiver : ISyntaxReceiver
{
    public ClassDeclarationSyntax MyClassSyntax { get; private set; }

    public void OnVisitSyntaxNode(SyntaxNode syntaxNode)
    {
        if (syntaxNode is ClassDeclarationSyntax classSyntax)
        {
            if (classSyntax.Identifier.ValueText == "MyClass")
            {
                MyClassSyntax = classSyntax;
            }
        }
    }
}
```

Generator - Execute

- 主要成員為 `GeneratorExecutionContext`
- `context` 中具有一個 `CancellationToken`
- 從 `context` 取出各種 `meta`
- 依邏輯產生程式碼字串，並加到最後的編譯結果
- 向 IDE 回報診斷資訊



Generator - AddSource

- 將要產生的程式碼加到最後的編譯結果

```
public void Execute(GeneratorExecutionContext context)
{
    var codeText = @"namespace Sample { public class HelloWorld
    { public void Say() => Console.WriteLine(""Hello World""); } }";

    context.AddSource("HelloWorld", SourceText.From(codeText, Encoding.
UTF8));
}
```



Generator - CancellationToken

- 在 **Initial** 及 **Execute** 中都有提供
- Generator 會在 IDE 的背景執行
 - IDE 提供即時的 Intellisense
 - 在檔案內容變更時 Generator 會被呼叫
- 提供 IDE 一個**取消 Generator 執行**的機制
- 可降低 CPU 使用率



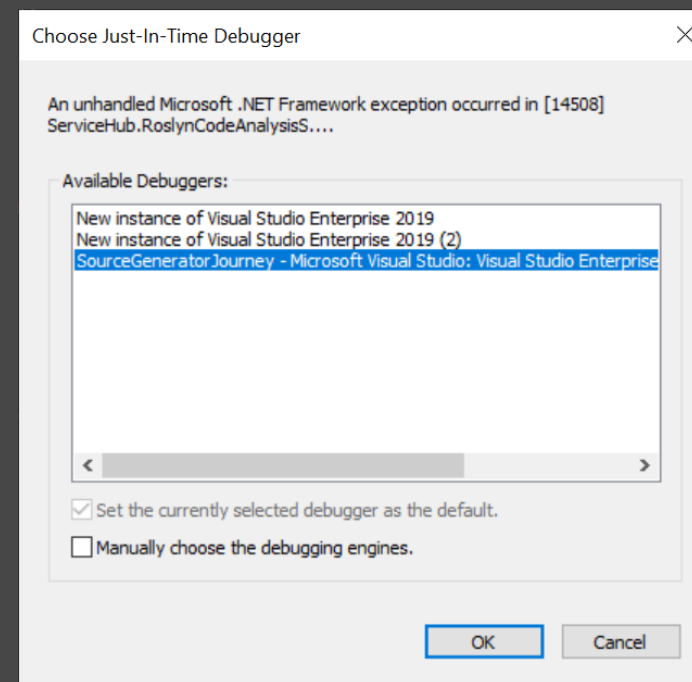
Example - CancellationToken

```
[Generator]
public class MyGenerator : ISourceGenerator
{
    public void Initialize(GeneratorInitializationContext context)
    {
        if (!context.CancellationToken.IsCancellationRequested)
        {
        }
    }
    public void Execute(GeneratorExecutionContext context)
    {
        if (!context.CancellationToken.IsCancellationRequested)
        {
        }
    }
}
```

如何偵錯 Generator

- 目前無法與 IDE 自動整合
- 必須透過程式手動叫用偵錯器
- 必須手動選擇偵錯執行的 IDE 視窗

```
public void Initialize  
    (GeneratorInitializationContext context)  
{  
    System.Diagnostics.Debugger.Launch();  
}
```



Getting Advanced

- How to retrieve Metadata?





Meta 可以從哪裡來?

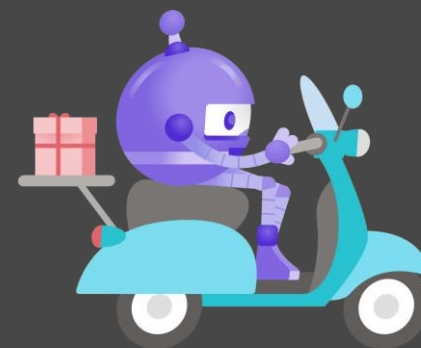
- **Compilation**

- SyntaxTree, SemanticModel

- SyntaxReceiver

- AdditionalFiles

- AnalyzerConfigOptions



取得 Syntax 資訊

- 從 `context.Compilation` 拿到首次編譯的結果
- 主要對象是 Roslyn 的 Syntax API
- 取得 **BotAttribute** 的 `AttributeSyntax`

```
var syntaxNodes = compilation
    .SyntaxTrees.SelectMany(s => s.GetRoot().DescendantNodes());
var attributeSyntaxs = syntaxNodes
    .Where((d) => d.IsKind(SyntaxKind.Attribute)).OfType<AttributeSyntax>();
var botAttributeSyntax = attributeSyntaxs.
    FirstOrDefault(x => x.Name.ToString() == "Bot");
```

取得 SyntaxReceiver 資訊

- 從 context 取得 SyntaxReceiver 後再做轉型

```
public void Execute(GeneratorExecutionContext context)
{
    MyClassSyntaxReceiver receiver =
        (MyClassSyntaxReceiver) context.SyntaxReceiver;

    var myClassSyntax = receiver.MyClassSyntax;
}
```




存取附加檔案資訊

- 必須手動於 **.csproj** 裡面宣告
- 可於 **xml** 標籤內自訂屬性資訊
 - 須明確標記為 **CompilerVisibleItemMetadata**

```
<ItemGroup>  
  <AdditionalFiles Include="Cars.csv" CacheObjects="true" />  
  <CompilerVisibleItemMetadata Include="AdditionalFiles"  
    MetadataName="CacheObjects" />  
</ItemGroup>
```

回報診斷訊息

- 使用 **ReportDiagnostic** 向 IDE 回報診斷訊息

```
Context.ReportDiagnostic(Diagnostic.Create(
    new DiagnosticDescriptor("MYERR001", "TestDiagnostic",
        $"Here is a error.", "source generator",
        DiagnosticSeverity.Error, true), Location.None));
```

Error List

Entire Solution

✖ 1 Error

⚠ 1 Warning

ℹ 0 of 2 Messages

⌵

Build + IntelliSense

	Code	Description	Project	File	Line	Suppression State
✖	MYERR001	Here is a error.	SourceGeneratorJourney	CSC	1	
⚠	MYWAR001	Here is a warring.	SourceGeneratorJourney	CSC	1	

將產生的程式輸出為實體檔案

- **EmitCompilerGeneratedFiles**：允許輸出檔案
- **CompilerGeneratedFilesOutputPath**：指定輸出路徑

```
<PropertyGroup>  
  <EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>  
  <CompilerGeneratedFilesOutputPath>  
    $(BaseIntermediateOutputPath)\GeneratedFiles  
  </CompilerGeneratedFilesOutputPath>  
</PropertyGroup>
```



如何打包成 NuGet package

- 設定 `GeneratePackageOnBuild` 為 `true`
- 設定 `IncludeBuildOutput` 為 `false`
- 設定 Generator 用的套件標示為 `PrivateAssets="all"`
- 設定 Generator 打包路徑於 `analyzer` 目錄下



如何打包成 NuGet package

```
<PropertyGroup>
  <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
  <IncludeBuildOutput>false</IncludeBuildOutput>
</PropertyGroup>
<ItemGroup>
  <PackageReference Include="Microsoft.CodeAnalysis.CSharp"
    Version="3.8.0" PrivateAssets="all" />
  <PackageReference Include="Microsoft.CodeAnalysis.Analyzers"
    Version="3.3.1" PrivateAssets="all" />

  <!-- 將 Generator 打包路徑指定在 analyzer 目錄下 -->
  <None Include="$(OutputPath)\$(AssemblyName).dll" Pack="true"
    PackagePath="analyzers/dotnet/cs" Visible="false" />
</ItemGroup>
```

參考外部的 NuGet 套件

- 當套件用途為供 Generator 執行過程用時
- 必須設定 `PrivateAssets` 及 `GeneratePathProperty` 屬性
- 必須額外將 d11 打包到指定的 analyzer 目錄底下
- 套件名稱要改為 `Pkg{PACKAGE_NAME}`
 - 遇到點 (.) 要改為底線 (_)
 - EX: `Newtonsoft.Json` => `PkgNewtonsoft.Json`



Example – Pack Newtonsoft.Json

```
<ItemGroup>
  <PackageReference Include="Newtonsoft.Json" Version="12.0.1"
    PrivateAssets="all" GeneratePathProperty="true" />

  <!-- 將套件 .dll 打包路徑指定在 analyzer 目錄下 -->
  <None Include="$(PkgNewtonsoft_Json)\lib\netstandard2.0\*.dll"
    Pack="true" PackagePath="analyzers/dotnet/cs" Visible="false" />
</ItemGroup>
```



使用案例介紹

- Hello World
- DotNetBot
- DataBuilder
- CSVToList
- Mapper



結語

- 尚未與 IDE 偵錯完全整合
- 繞來繞去最後還是得搞 Roslyn
- 寫起來很煩，用起來很爽
- 若要收納為團隊用途，請寫說明文件
- 期待 .NET 6 會推出更兇猛的功能

補充 – Syntax 分析工具介紹

- SyntaxViewer (Visual Studio 2019 內建)
- SharpLab
- LinqPad





補充 – 使用案例

- DataBuilder
- StrongInject
- InlineMapper
- 官方 Roslyn-SDK 範例



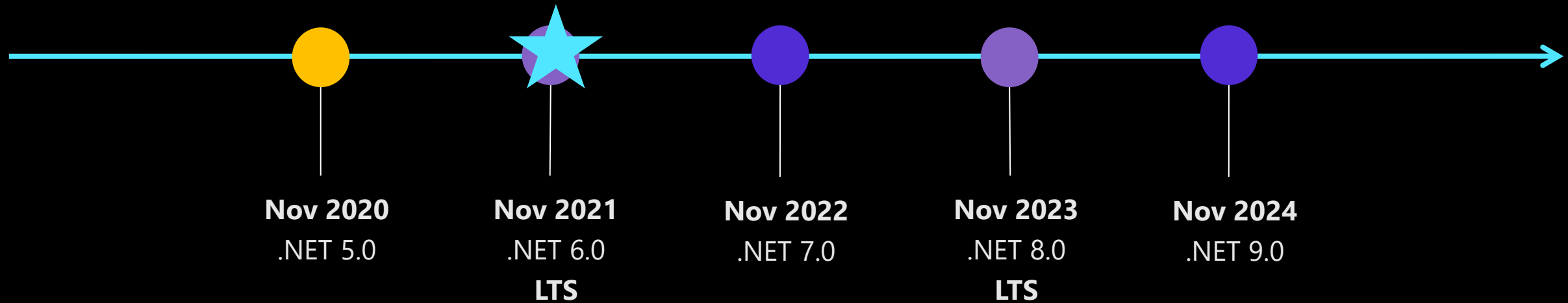
References (Microsoft)

- [Introducing C# Source Generators](#)
- [New C# Source Generator Samples](#)
- [Source Generators Cookbook](#)
- [Source Generators](#)

References (Other)

- Generating Code in C#
- Source Generators in .NET 5 with ReSharper
- Auto generate builders using Source Generator in .NET 5
- Thinking beyond Roslyn source generators and aspect-oriented programming (postsharp)

.NET Schedule



- .NET 5.0 released today!
- Major releases every year in November
- LTS for even numbered releases
- Predictable schedule, minor releases as needed

.NET Conf
2020

特別感謝

91APP
Technical Network



KKKTIX



HackMD



Microsoft

Build School

STUDY4
為 學 習 而 生

以及各位參與活動的你們

