ChatGPT

RimWorld 模組開發實作教學：編譯並注入 DLL 至遊戲

1. 使用 VSCode 建立 C# 類庫專案並編譯 DLL

在開始撰寫 RimWorld 模組程式碼之前，我們需要建立一個 C# 類庫（Class Library）專案，以產生可供 RimWorld 載入的 DLL 檔案。以下是在 VSCode 中設定專案的步驟：

1. **建立專案資料夾：** 在 RimWorld 安裝目錄的 `Mods` 子資料夾下新建一個以模組命名的資料夾（例如 `MyMod`）。在該資料夾中建立一個 `Source` 子資料夾，用來存放您的 VSCode 專案檔案和程式碼檔。專案結構可以設計如下（`MyMod` 即您的模組名稱，可自由命名）：

```
RimWorld/Mods/
└── MyMod/                    # 您的模組根目錄
    ├── Source/               # VSCode 專案與原始碼
    ├── Assemblies/           # 編譯輸出的 DLL 放置處
    └── About/                # 模組描述檔 (About.xml) 等
```

1. **初始化 C# 專案：** 使用 VSCode 開啟 `MyMod/Source` 資料夾。在 VSCode 終端機中，透過 .NET CLI 建立 .NET Framework 4.7.2 的類庫專案：

   ```
   dotnet new classlib -f net472 -n MyMod -o .
   ```

   這將產生一個名為 `MyMod.csproj` 的專案檔，目標 framework 為 .NET 4.7.2（RimWorld 基於 Unity 2019，相容 .NET Framework 4.7.2）。確保專案的目標 Framework 設定正確。

2. **參考 RimWorld 遊戲的 Assembly：** 為了使用 RimWorld 的 API，我們需要將遊戲的核心程式庫加入專案參考。在 VSCode 中開啟 `.csproj` 檔案，加入以下內容至 `<ItemGroup>` 中：

   ```
   <ItemGroup>
    <!-- 參考 RimWorld 核心程式 Assembly -->
    <Reference Include="Assembly-CSharp">
     <HintPath>C:\Path\To\RimWorld\RimWorldWin64_Data\Managed\Assembly-
   CSharp.dll</HintPath>
     <Private>False</Private> <!-- 不複製至輸出目錄 -->
    </Reference>
    <!-- 參考 Unity Engine 核心模組 -->
    <Reference Include="UnityEngine.CoreModule">
     <HintPath>C:
   \Path\To\RimWorld\RimWorldWin64_Data\Managed\UnityEngine.CoreModule.dll</
   HintPath>
     <Private>False</Private>
   ```

```
  </Reference>
</ItemGroup>
```

請將 `C:\Path\To\RimWorld\...` 換成您本機 RimWorld 安裝路徑中 `RimWorld*_Data/Managed` 資料夾的位置。上述設定等效於在 Visual Studio 中**瀏覽**並加入 `Assembly-CSharp.dll` 和 `UnityEngine.CoreModule.dll` 兩個檔案做為參考 [1] [2]。同時，我們將每個參考的 `<Private>` 設為 False，確保編譯時不會將這些遊戲 DLL 複製到輸出（因為 RimWorld 本體已經提供它們）。

3. **設定輸出路徑：** 編輯專案設定，將編譯輸出目錄指向模組的 Assemblies 資料夾。打開 `.csproj`，找到或新增 `<OutputPath>`：

```
<PropertyGroup>
  <OutputPath>..\Assemblies\</OutputPath>
  <Optimize>true</Optimize>      <!-- 建議釋出模式下啟用最佳化 -->
  <DebugType>none</DebugType>    <!-- 不產生 .pdb 偏好 -->
  <OutputType>Library</OutputType> <!-- 類庫專案 -->
  <TargetFramework>net472</TargetFramework>
</PropertyGroup>
```

如此可保證每次建置時，產生的 DLL 檔案會直接輸出到 `MyMod/Assemblies/` 資料夾 [3]。這使我們無需手動複製 DLL，即可讓 RimWorld 載入最新編譯的版本。

4. **編譯專案：** 完成以上設定後，在 VSCode 中執行 **Terminal > Run Build Task** 或在終端機輸入：

```
dotnet build MyMod.csproj -c Release
```

若設定正確，會在 `MyMod/Assemblies/` 資料夾下生成 `MyMod.dll` 檔案。

2. 將 DLL 放入模組資料夾並驗證自動載入

完成 DLL 編譯後，下一步是確保 RimWorld 能正確偵測並載入您的模組 DLL：

- **About 資料夾與描述檔：** 在模組根目錄下的 `About/` 資料夾建立 `About.xml`。這是 RimWorld 識別模組的必要檔案 [4]。一個簡單的 `About.xml` 範例如下：

```
<ModMetaData>
 <name>My Mod Name</name>
 <author>Your Name</author>
 <version>1.4</version> <!-- 目標遊戲版本，例如 RimWorld 1.4 -->
 <description>示範模組，展示如何載入自訂 DLL。</description>
</ModMetaData>
```

確保 `About.xml` 正確放置於 `MyMod/About/` 資料夾內，且拼字大小寫正確，否則 RimWorld 不會辨識此模組。

- **Assemblies 資料夾：** 確認前一步編譯產生的 `MyMod.dll` 已存在於 `MyMod/Assemblies/` 資料夾中。RimWorld 啟動時會自動掃描各模組目錄下的 Assemblies 資料夾，**將其中的 DLL 自動載入** [5] 。不需要額外的設定或指令，前提是檔名副檔名正確（**.dll**）且位於正確的資料夾。

- **模組結構檢查：** 您的模組目錄現在至少包含以下內容：

```
MyMod/
├─ About/
│    └─ About.xml           # 模組資訊檔
└─ Assemblies/
     └─ MyMod.dll           # 編譯出的 C# 程式庫
```

（若使用 XML 定義、貼圖或音效等資源，則可能還有 Defs、Textures 等資料夾，但本教學側重於代碼部分。）

- **啟動 RimWorld 測試：** 打開 RimWorld，在遊戲主選單點擊 **Mods**，在列表中找到您的模組名稱並確保**啟用**它。之後，點擊重新啟動讓遊戲載入模組。進入遊戲後，開啟開發者模式(Console)或檢視 RimWorld 的日誌檔（`Player.log`），確認是否有載入模組的相關訊息。如果模組的 DLL 成功載入，您應該能在日誌中看到 RimWorld 對該模組的載入記錄。接下來，我們將透過程式碼讓這個載入更為明顯。

3. 寫作 RimWorld 模組主類別（繼承 Verse.Mod）並初始化

RimWorld 提供 `Verse.Mod` 類別作為每個模組的入口點類別。遊戲會在載入您的 DLL 後**自動透過反射尋找**繼承自 `Verse.Mod` 的類別並**實例化**它。這意味著您的模組**主類別**（通常我們也稱為 **Mod 類**）的建構子將在遊戲啟動時被呼叫，使您可以在其中執行初始化邏輯。

撰寫模組主類： 在您的 VSCode 專案中新增一個 C# 檔案（例如 `MyMod.cs`），撰寫如下內容：

```
using RimWorld;
using Verse;

namespace MyModNamespace  // 選擇一個獨特的命名空間
{
  /// <summary>
  /// 模組的主類別，必須繼承 Verse.Mod。
  /// RimWorld 啟動時會自動建立此類的實例。
  /// </summary>
  public class MyMod : Mod
  {
    /// <summary>模組主類別的建構子，RimWorld 會傳入 ModContentPack。</summary>
    /// <param name="content">模組內容包，由 RimWorld 傳入。</param>
    public MyMod(ModContentPack content) : base(content)
    {
      // 在此處撰寫模組初始化邏輯
      Log.Message("模組 MyMod 已載入並初始化！");
      // 例如：在載入時打印日誌以驗證
    }
```

```
    }
}
```

上述程式碼中，我們定義了 `MyMod` 類別並繼承自 `Verse.Mod`。特別注意建構子的簽名必須是 `MyMod(ModContentPack content) : base(content)`，這是 RimWorld 掃描模組類別所要求的標準構造函式 [6]。在這個建構子中，我們可以執行任何初始化工作，例如設定預設值、配置自訂系統或打印確認訊息等。

自動註冊初始化邏輯： 由於 RimWorld 在載入模組時**自動**建立 `MyMod` 實例，您無需自行在程式碼中呼叫它。任何寫在建構子（或靜態建構子）裡的邏輯都會自動執行。例如，上述的 `Log.Message` 調用會在遊戲啟動時觸發，將訊息輸出到 RimWorld 日誌。您可以啟動遊戲確認該訊息出現，以驗證模組主類別確實被執行。

> **提示：** 除了 `Mod` 類別以外，RimWorld 也支援使用 `StaticConstructorOnStartup` 屬性讓靜態類別在啟動時執行靜態建構子 [7] [8]。然而，本範例選擇 `Mod` 類別的方式，可在需要模組設定(Mod Settings)或非常早期執行代碼時使用 [9]。一般初始化則建議在遊戲主選單載入前執行的靜態建構子方式，以確保遊戲資源已準備妥當。

4. 利用反射機制與 Mod 實例實現簡單功能

現在，我們已建立模組主類並在 RimWorld 啟動時成功執行其建構子。這背後實際上是 RimWorld 利用**反射**機制，搜尋我們 DLL 中繼承 `Verse.Mod` 的類別並呼叫其建構子完成初始化。透過這個機制，我們可以在模組載入時附加自訂的功能。

以下我們來製作一個簡單的示範功能：

- **載入時顯示訊息：** 我們在 `MyMod` 建構子內呼叫 `Log.Message("模組 MyMod 已載入並初始化！")`。這行代碼會在 RimWorld 載入模組時執行，將訊息輸出到開發者主控台 (或Player.log)。啟動遊戲後可在日誌中找到這行「模組 MyMod 已載入並初始化！」，證明我們的 DLL 已成功注入並執行。

- **修改遊戲行為的佔位函式：** 您也可以在這裡加入**佔位**程式碼，以日後擴展成實際的遊戲行為修改。例如，假設我們想在模組初始化時修改遊戲中某個值（如角色的移動速度）或掛接事件，我們可以先寫一個佔位函式：

  ```
  private void ApplyPatches()
  {
    // TODO: 將來透過 Harmony 或其他方式修改遊戲行為
    Log.Message("ApplyPatches() 被呼叫：此處將實現遊戲行為修改。");
  }
  ```

  然後在建構子中呼叫 `ApplyPatches()`。目前此函式僅輸出一行日誌，表示它被呼叫了。待日後我們準備真正修改遊戲行為時，可以在此函式中實作具體邏輯（例如使用 Harmony library 對遊戲原始方法進行前綴/後綴調用，以改變遊戲行為）。

通過上述方式，我們在模組載入階段利用 RimWorld 提供的反射掛接點（Mod 類別實例化）成功執行了自訂的初始化和功能函式。小結一下此流程：

- RimWorld 啟動時偵測到 `MyMod.dll` 並載入（放在 Assemblies 資料夾即可自動載入 [5]）。
- RimWorld 掃描 DLL，找到 `MyMod : Verse.Mod` 類。

- RimWorld 建立 `MyMod` 類的實例（傳入 `ModContentPack` 參數），執行我們在建構子中定義的所有操作。
- 我們的模組程式碼因此在遊戲初始化時被執行，可以進行日志輸出或進一步遊戲邏輯修改。

5. 整合自訂資源載入框架（IAssetLoader<T> 與 LoggingLoaderDecorator）

接下來，我們將結合使用者指定的 GitHub 專案 `angus945/moddable-study` 中的模組化架構概念，將 **資源載入介面**與**裝飾器**模式融入 RimWorld 模組。此步驟的目的是模擬「當 RimWorld 載入模組後，初始化一套自訂的資源載入框架」。我們將示範如何在模組主類中建立服務物件，使用 **泛型**介面 `IAssetLoader<T>` 來載入遊戲資源，並使用 `LoggingLoaderDecorator` 裝飾它以添加日誌記錄功能。

首先，假設我們在模組的程式碼中引入（或定義）了以下介面與類別：

```
/// <summary>
/// 資產載入器介面，定義載入資源的方法。
/// T 可以是任何資源型別（如 Texture2D、AudioClip 或自訂的資料定義類）。
/// </summary>
public interface IAssetLoader<T>
{
    T Load(string assetPath);
}

/// <summary>
/// 資產載入器的裝飾者，增加日誌紀錄功能的包裝。
/// </summary>
public class LoggingLoaderDecorator<T> : IAssetLoader<T>
{
    private readonly IAssetLoader<T> innerLoader;

    public LoggingLoaderDecorator(IAssetLoader<T> innerLoader)
    {
        this.innerLoader = innerLoader;
    }

    public T Load(string assetPath)
    {
        // 載入前的日誌
        Log.Message($"[資源載入] 開始載入資源: {assetPath}");
        // 調用內部實際的載入器
        T asset = innerLoader.Load(assetPath);
        // 載入後的日誌
        Log.Message($"[資源載入] 完成載入資源: {assetPath}");
        return asset;
    }
}
```

上述 `IAssetLoader<T>` 是一個泛型介面，提供 `Load(string)` 方法來載入指定路徑的資源並返回類型 `T` 的物件。`LoggingLoaderDecorator<T>` 則實現相同的介面，但在內部持有另一個

`IAssetLoader<T>`（透過建構子注入），在呼叫 `Load` 時會先後輸出日誌，再委派給內部載入器執行實際載入，達到**裝飾**（Decorator）模式的效果。

接著，我們需要一個實際的**資源載入器**實作。例如，我們可以建立一個簡單的載入器，用來從 RimWorld 的模組資料夾中讀取某類型的檔案。為了示範，此處創建一個假想的載入器 `TextAssetLoader`，從模組的 Def 資料夾讀取文字檔並返回其內容：

```
/// <summary>
/// 簡單的文字資源載入器，讀取給定路徑的文字檔內容。
/// </summary>
public class TextAssetLoader : IAssetLoader<string>
{
  public string Load(string assetPath)
  {
    // 假設 assetPath 是相對於模組某個資料夾的路徑
    string fullPath = System.IO.Path.Combine(GenFilePaths.ModContentPackPath, assetPath);
    // 讀取檔案並回傳文字內容
    return System.IO.File.ReadAllText(fullPath);
  }
}
```

> **注意：** 上述 `GenFilePaths.ModContentPackPath` 僅作示意，實際 RimWorld 提供的 API 可用 `ModContentPack` 來解析自己模組資料夾的路徑。例如，可透過 `content.RootDir`（在 `Mod` 類建構子參數取得的 `ModContentPack content`）來獲取模組根目錄，再組合出資源路徑。

現在在模組主類 `MyMod` 的建構子中，我們初始化這套資源載入框架：

```
public class MyMod : Mod
{
  // ... 前略 ...

  public MyMod(ModContentPack content) : base(content)
  {
    // 初始化自訂資源載入服務
    IAssetLoader<string> baseLoader = new TextAssetLoader();
    IAssetLoader<string> loaderWithLogging = new LoggingLoaderDecorator<string>(baseLoader);

    // 示範使用載入器載入一個資源（例如讀取一個定義檔）
    string defData = loaderWithLogging.Load("Defs/MyDef.txt");
    Log.Message($"自訂資源框架初始化完畢，範例資源內容：{defData.Substring(0, 20)}...");

    // 其他初始化邏輯...
  }
}
```

在此建構子裡：

- 我們建立了一個 `TextAssetLoader` 執行個體作為基礎載入器。
- 用 `LoggingLoaderDecorator<string>` 將上述載入器包裝起來，得到 `loaderWithLogging`。由於我們希望對載入過程打印日誌，所以使用裝飾器增強功能。
- 接著，示範性地呼叫 `loaderWithLogging.Load("Defs/MyDef.txt")` 嘗試載入模組內某個定義檔（路徑僅作為例子）。這會觸發我們裝飾器的日誌輸出，然後透過基礎載入器讀取檔案內容。最後將部分內容打印出來，證明載入框架在模組初始化時已經運作。

透過這種方式，我們成功將**模組化架構**引入 RimWorld 模組中：利用 `IAssetLoader<T>` 抽象資源讀取行為，並用 `LoggingLoaderDecorator` 添加額外功能（這充分利用了您對 **泛型**與**Decorator模式** 的熟悉程度）。未來，您可以擴充更多 `IAssetLoader` 的實作（例如載入紋理、載入音效等），並套用不同的裝飾者（例如緩存裝飾者、錯誤處理裝飾者等），構成一個靈活的資源載入服務。在 RimWorld 中，這些服務可以在模組初始化時啟動（如我們所示），然後供遊戲其餘部分或其他模組元件使用。

6. 範例程式碼與模組目錄結構總覽

經過上述步驟，我們已完成從建立專案、編譯 DLL、放入模組資料夾，到撰寫模組主類及整合自訂框架的所有工作。這裡將提供**最終專案結構**與**關鍵程式碼**的總覽，供您對照：

```
RimWorld/
└─ Mods/
   └─ MyMod/                          # 模組資料夾
      ├─ About/
      │   └─ About.xml                # 模組描述檔（名稱、版本等資訊）
      ├─ Assemblies/
      │   └─ MyMod.dll                # 編譯產生的 DLL，RimWorld 啟動時自動載入 [5]
      └─ Source/                      # 原始碼（非必需，方便開發與版本控制）
          ├─ MyMod.csproj             # C# 專案檔 (.NET Framework 4.7.2)
          ├─ MyMod.cs                 # 模組主類定義（MyMod : Verse.Mod）
          ├─ IAssetLoader.cs          # 資源載入介面定義
          ├─ LoggingLoaderDecorator.cs # 裝飾者類別定義
          └─ TextAssetLoader.cs       # 實際資源載入器實作
```

程式碼重點回顧：

- MyMod.cs（模組主類）：繼承 `Verse.Mod` 並實作建構子，在其中完成模組初始化（例如註冊服務、打印日誌）。這是模組進入點，RimWorld 透過它來執行我們的代碼 [6]。
- IAssetLoader.cs：定義泛型介面，為不同類型資源載入提供統一介面。
- LoggingLoaderDecorator.cs：使用 Decorator 模式包裝 `IAssetLoader`，添加載入過程的日誌紀錄功能，示範介面與泛型的進階應用。
- TextAssetLoader.cs：簡單的資源載入實作例子，演示如何讀取外部檔案作為遊戲資源。

所有提供的程式碼都經過簡化並附有註解，說明其作用與用途，便於初學者理解每一行代碼在模組開發中的角色。您可以以此為基礎進一步擴充功能：例如真正修改遊戲內某些行為（可藉由 Harmony 庫進行方法攔截）、載入更多種類型的資源，或是加入**Mod設定**畫面等。

透過本教學，您應該對 **「如何在 VSCode 中編譯 C# DLL 並將其作為 RimWorld 模組載入」** 有了完整的了解。從項目設置、DLL 放置到 RimWorld 自動載入機制，我們一步步驗證了整個流程，並額外展示了結合**泛型**與**Decorator**模式打造擴展性架構的實例。祝您在 RimWorld 模組開發之旅中玩得開心、學得愉快！ [1] [5]

---

[1] [2] [3] Modding Tutorials/Setting up a solution - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Setting_up_a_solution

[4] [5] Mod Folder Structure - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Mod_Folder_Structure

[6] Modding Tutorials/ModSettings - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/ModSettings

[7] [8] [9] Modding Tutorials/Hello World - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Hello_World