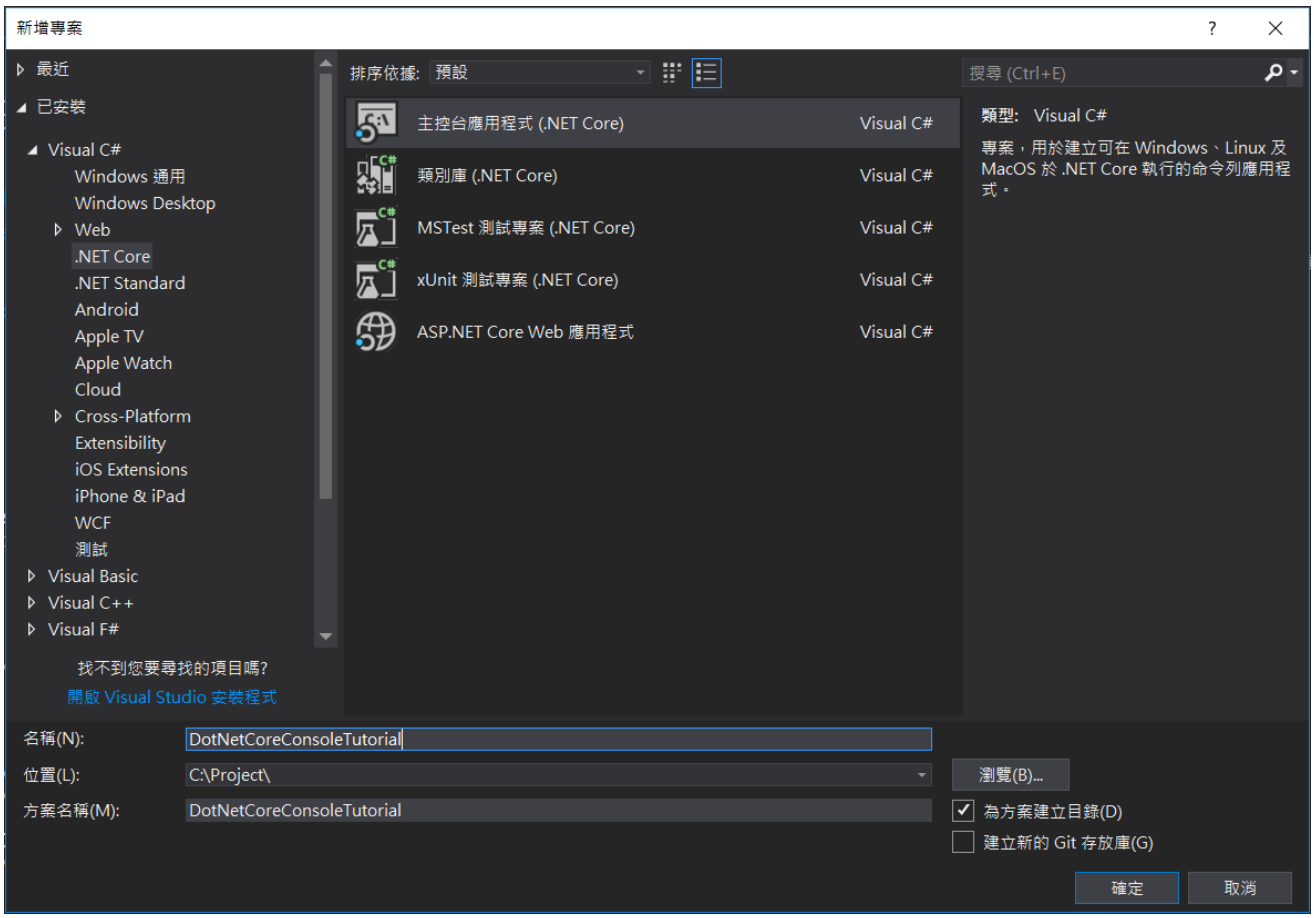
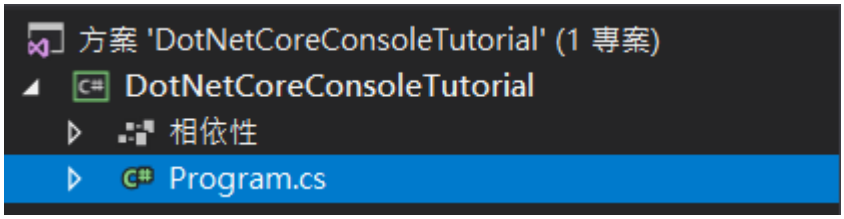
Build .net Core Console Application with Dependency Injection (DI)

建立 .net Core Console Application



檔案結構：



新增註冊容器 ServiceCollection

1. 使用 nuget 安裝 Microsoft.Extensions.DependencyInjection 套件
2. 在 Program.cs 中的 Main 方法中宣告註冊容器

```
static void Main(string[] args)
{
    var services = new ServiceCollection();
}
```

3. 新增一個 IService 介面及 Service 類別來處理邏輯服務，新增一個 IRepository 介面及 Repository 類別來處理資料服務

   IService.cs

```
public interface IService
{
    void DoWork();
}
```

   Service.cs

```
public class Service : IService
{
    private readonly IRepository _repository;

    public Service(IRepository repository)
    {
        _repoistory = repositoy;
    }

    public void DoWork()
    {
        _repository.DoWork();
    }
}
```

   IRepository.cs

```
public interface IRepository
{
    void DoWork();
}
```

Repository.cs

```
public class Repository : IRepository
{
    public void DoWork()
    {
        var input = Console.ReadLine();

        Console.WriteLine($"Input value: {input}");
    }
}
```

4. 在 Program.cs 將 IService, IRepository 註冊到容器中

```
static void Main(string[] args)
{
    var services = new ServiceCollection();

    services.AddSingleton<IService, Service>();
    services.AddSingleton<IRepository, Repository>();
}
```

5. 建立 Service Provider 供後續注入時提供需要的服務

```
static void Main(string[] args)
{
    var services = new ServiceCollection();

    services.AddSingleton<IService, Service>();
    services.AddSingleton<IRepository, Repository>();

    var serviceProvider = services.BuildServiceProvider();
}
```

至此已經完成了使用DI容器註冊服務，我們可以再稍微修改一下程式碼，將註冊的職責交由另一個方法來完成

```
static void Main(string[] args)
{
    var services = new ServiceCollection();
    // Register services to service collection
    ConfigureServices(services);
    var serviceProvider = services.BuildServiceProvider();
}
```

```
/// <summary>
/// Register services
/// </summary>
/// <param name="services">Service collection</param>
private static void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IService, Service>();
    services.AddSingleton<IRepository, Repository>();
}
```

開始使用DI開發應用程式

當註冊完成之後，即可開始使用DI來進行應用程式開發，有別於 ASP.NET Core 使用 UseMvc() 依據路由取得服務，在 .NET Core 需要自行從 service provider 取得服務

```
private static void Main(string[] args)
{
    var services = new ServiceCollection();
    // Register services to service collection
    ConfigureServices(services);
    var serviceProvider = services.BuildServiceProvider();

    // Get instance
    var service = serviceProvider.GetRequiredService<IService>();
}
```

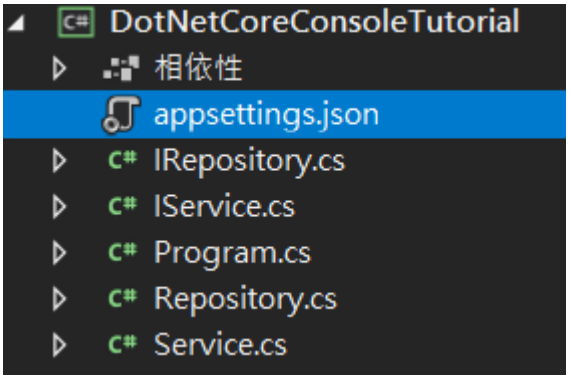
之後呼叫服務的方法即可開始進行業務

註冊 Configuration

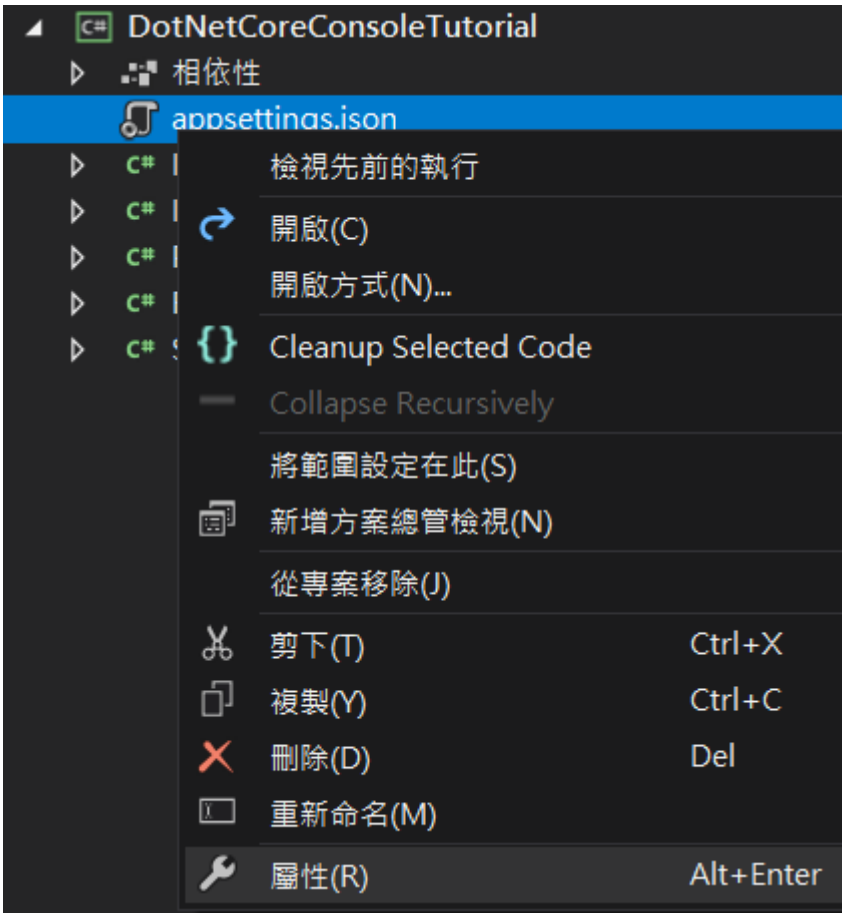
藉由註冊 Configuration，我們可以透過後續注入 IConfiguration 取用 appsettings.json 的設置來取得如資料庫連線等字串值

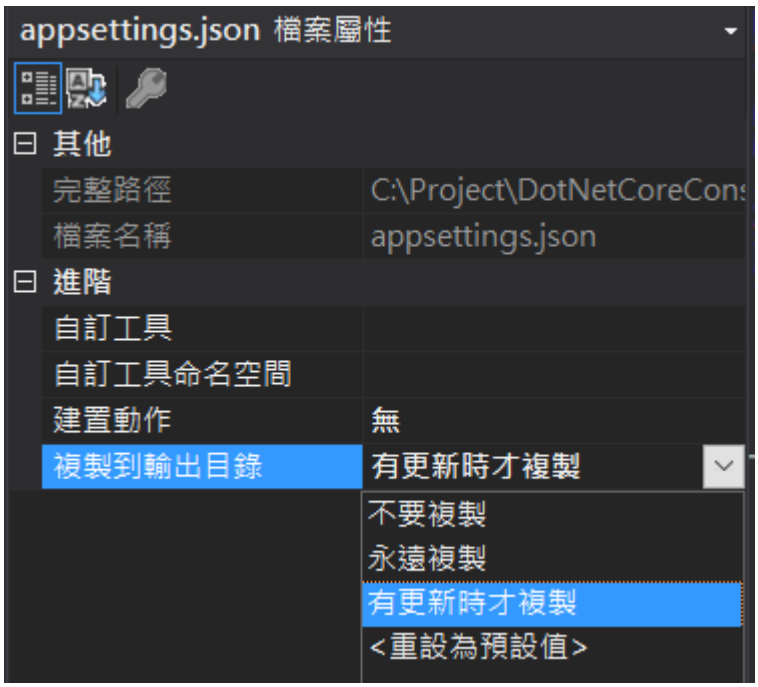
1. 使用 nuget 加入 Microsoft.Extensions.Configuration 及 Microsoft.Extensions.Configuration.Json 套件
2. 在目錄中加入 appsettings.json 設定檔，內容如下

```
{
  "ConnectionStrings": {
    "DefaultConnection": "DBConnection"
  }
}
```



3. 另外更改 appsettings.json 的屬性，確認建置時會複製到輸出目錄





4. 再透過 ConfigurationBuilder 來建立 Configuration 的相關設置

```
private static void ConfigureServices(IServiceCollection services)
{
    var configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .Build();

    ...
}
```

5. 將 Configuration 服務註冊進容器

```
private static void ConfigureServices(IServiceCollection services)
{
    var configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .Build();

    // Register configuration
    services.AddSingleton(configuration);

    ...
}
```

6. 修改 Repository.cs 來取用 appsettings.json 的 ConnectionStrings (別忘記注入 IConfiguration)

```
public class Repository : IRepository
{
    private readonly IConfigurationRoot _configuration;
```

```
    public Repository(IConfigurationRoot configuration)
    {
        _configuration = configuration;
    }

    public void DoWork()
    {
        var connection = _configuration.GetConnectionString("DefaultConnection");

        ...
    }
}
```

7. 執行來驗證是否成功注入
8. 之後若是需要加入不同的設定值，可以在 appsettings.json 加入其他設定值，並透過下方程式來取用

```
configuration["Property:Child"];
```

註冊 NLog 來記錄 log

1. 使用 nuget 加入 NLog.Extensions.Logging 套件
2. 建立 nlog.config 設定檔，可設置如下

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd" xsi:schemaLocation="NLog
NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      autoReload="true"
      internalLogFile="c:\temp\console-example-internal.log"
      internalLogLevel="Info">

  <!-- optional, add some variables
  https://github.com/nlog/NLog/wiki/Configuration-file#variables
  -->
  <!--<variable name="myvar" value="myvalue" />-->

  <!--
  See https://github.com/nlog/nlog/wiki/Configuration-file
  for information on customizing logging rules and outputs.
   -->
  <targets>
    <target name="Infofile" xsi:type="File"
            fileName="${basedir}/logs/${shortdate}/Info.txt"
            layout="${longdate} | ${level:uppercase=true} | ${logger} | ${message}
${newline}" />
  </targets>
  <rules>
    <!-- add your logging rules here -->
    <logger name="*" level="Info" writeTo="Infofile" />
  </rules>
</nlog>
```

3. 檔案屬性設定為「有更新時才複製」
4. 註冊 ILoggerFactory 及 ILogger 進服務容器，並設置 logging 等級

```
private static void ConfigureServices(IServiceCollection services)
{
    ...

    // Register logger factory
    services.AddSingleton<ILoggerFactory, LoggerFactory>();
    services.AddSingleton(typeof(ILogger<>), typeof(Logger<>));
    services.AddLogging((builder) =>
builder.SetMinimumLevel(LogLevel.Information));

    ...
}
```

5. 取得 Logger 實體並設置

```
private static void Main(string[] args)
{
    var services = new ServiceCollection();
    // Register services to service collection
    ConfigureServices(services);
    var serviceProvider = services.BuildServiceProvider();

    var loggerFactory = serviceProvider.GetRequiredService<ILoggerFactory>();

    // Configure NLog
    loggerFactory.AddNLog(new NLogProviderOptions { CaptureMessageTemplates = true,
CaptureMessageProperties = true });
    NLog.LogManager.LoadConfiguration("nlog.config");

    ...
}
```

6. 在需要紀錄 Log 的類別中注入 ILogger

```
public class Repository : IRepository
{
    private readonly IConfigurationRoot _configuration;
    private readonly ILogger<Repository> _logger;

    public Repository(IConfigurationRoot configuration, ILogger<Repository> logger)
    {
        _configuration = configuration;
        _logger = logger;
    }

    public void DoWork()
    {
```

```
            ...

            _logger.LogInformation("Doing work...");
        }
    }
```

7. 執行來驗證是否成功注入，可以在輸出目錄的 logs 資料夾中找到 log 紀錄

```
2018-11-15 10:59:29.1801 | INFO | DotNetCoreConsoleTutorial.Repository | Doing
work...
```