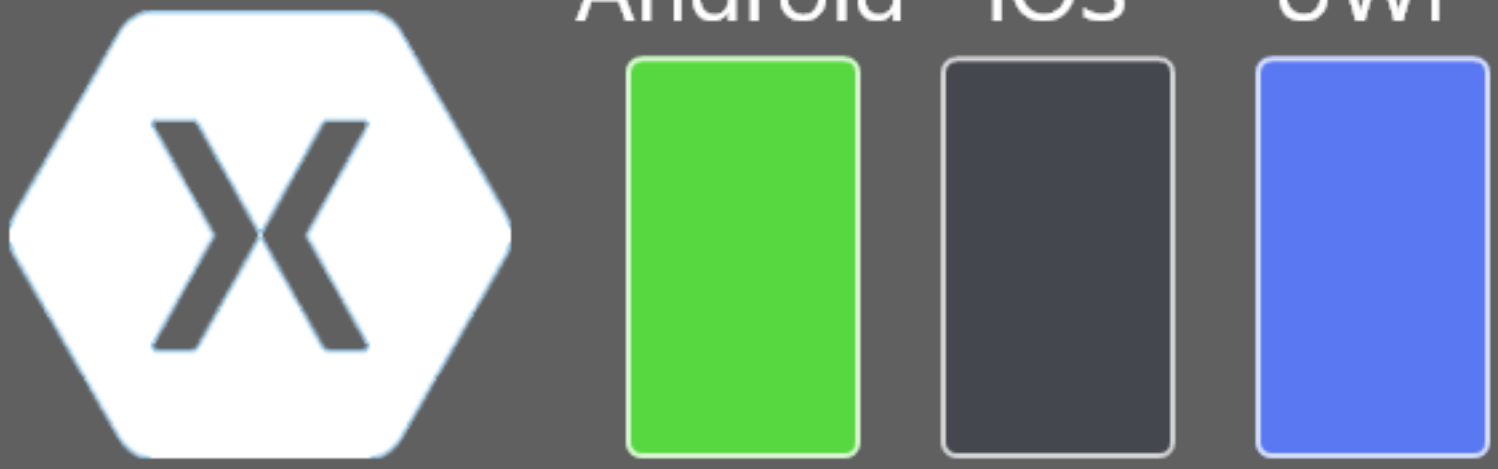
Xamarin.Forms Quick Start

快速入門



Vulcan Lee 著

Xamarin.Forms 快速入門

Xamarin.Forms Quick Start

Vulcan Lee

這本書的網址是 http://leanpub.com/xamarin-forms-quick-start

此版本發布於 2020-01-06

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

前言

Xamarin.Forms 為一個可以針對不同平台上的行動裝置，如：Android, iOS, UWP，協助開發者進行跨平台應用程式；Xamarin.Forms 是一套視覺介面工具組 UI Toolkit，透過宣告式的標記語言 XAML 來宣告與定義應用程式的每個頁面要出現的內容，而關於應用程式需要執行的各類商業邏輯部分，我們則需要使用 C# 程式語言在 .NET 平台上的運作。

我們知道在進行行動裝置應用程式開發的時候，需要進行該應用程式的各種畫面 UI 與商業邏輯程式碼的開發與設計，以往我們要進行開發出一套跨平台的應用程式，應用程式開發者需要精通各 App 平台提供的專屬 SDK 工具，透過 SDK 工具提供的使用者介面設計工具，來進行這些應用程式的頁面要顯示的內容的開發，因此，同樣一個頁面，應用程式開發者需要先使用 Android SDK 的使用者介面設計工具設計這些頁面畫面，接著又需要使用 iOS SDK 的使用者介面設計工具再設計一次同樣的頁面畫面。我們可以看到 Xamarin.Forms 帶來的好處那就是，，而是只需要定義一組 XAML 宣告檔案，就已經完成了關於行動應用程式的使用者視覺介面的設計，因為同樣的 XAML 宣告檔案在不同行動作業系統平台下執行時候，會使用原生 SDK 提供的原生使用者介面。

另外一個好處，那就是對於使用原生 SDK 工具來進行動應用程式的開發者而言，必須學習每個平台原生 SDK 指定的專屬程式語言，如：Java, Objective-C, Swift，同樣的，對於同樣的商業邏輯需求，開發者需要使用 Android SDK 的 Java 程式語言來設計一次，接者，需要使用 iOS SDK 的 Objective-C / Swift 語言再進行設計一次，這樣對於想要進行開發出可以在不同行動作業系統平台下的開發者而言，這是一個極為嚴峻的挑戰，因為開發者需要精通這些程式語言，並且也會面臨產生更多錯誤的機會；現在，在 Xamarin.Forms 開發環境之下，我們僅需要使用 C# 程式語言，便可以設計出可以在每個平台下執行的程式碼，而且，絕大多數比例的程式碼，統計上會有高於 80% 以上的程式碼，我們都僅需要撰寫一次，並且同時在不同平台下執行，只有很少比例的程式碼，因為需要呼叫原生平台

SDK 專屬的 API，才需要使用 C# 程式語言，在每個平台下額外撰寫呼叫特定 API 的程式碼。

因此，當您決定採用 Xamarin.Forms 來進行跨平台行動應用程式開發的首選之後，您可以享受到使用一套宣告式的標記語言 XAML 來設計使用者介面 UI 與只需撰寫一次 C# 程式碼，就已經完成了可以同時在 Android, iOS, UWP 平台下可以執行的行動應用程式。

對於想要採用 Xamarin.Forms 開發工具來進行跨平台行動應用程式開發的開發者而言，相信可以在 .NET 環境下執行 C# 程式語言的使用與應用上不會很陌生，可是絕大部分的 .NET 開發者，若本身沒有接觸或者使用過 WPF / UWP 專案開發，面對對於 XAML 這個宣告式的標記語言會感到相當的陌生與恐懼，另外，在本書撰寫的這個時間點，Visual Studio 尚未提供如同 WPF 開發環境中的友善 XAML 開發與設計操作介面，想要學習 XAML 宣告式的標記語言的開發者，必須面對如同 XML 文件中的各種眾多，繁雜的標籤 Tag 與不同的屬性。

作者從事多年 Xamarin.Forms 專案開發與教學工作，有鑑於絕大多數 Xamarin.Forms 新手，對於如何使用 Xamarin.Forms 開發工具進行專案開發與如何使用 Xamarin.Forms 提供的各種新開發技術，存在著各種疑惑與不之如著手學習，因此，著手撰寫這本書籍。

關於本書

這是一本帶領 Xamarin.Forms 新手開發者，可以透過書中介紹的各種知識、開發技能，配合練習專案實作，快速地學會使用 Xamarin.Forms 這個 UI Toolkit 來進行跨平台的行動應用裝置之應用程式開發工作。

這本書能提供什麼

在這本書裡面，將會提供 16 章的內容，分別是

- 開發前的安裝、設定準備工作 (共有三章)

  對於 Xamarin.Forms 開發新手，第一個學習卡關將會是如何安裝與設定一個可以進行 Xamarin.Forms 的開發環境；在這個部分將會詳細說明如何安裝與設定 Visual Studio 2019 ，使其可以順利的進行 Xamarin.Forms 的開發工作。
- 使用 C# 程式語言來直接開發 Xamarin.Forms App

  說明如何僅使用 C# 程式語言，就可以開發出 Xamarin.Forms 的應用程式的開發過程。
- 使用 XAML 標記宣告語言來開發 Xamarin.Forms App

  說明使用 XAML 宣告標記語言來進行頁面畫面的內容宣告，相關的商業邏輯則是使用程式碼後置 Code Behind 的方式來開發。
- 資料綁定 Data Binding

  資料綁定 Data Binding 是在 Xamarin.Forms 開發上，最為重要的技術，對於資料綁定的類型共有三種，這裡將會針對一般資料物件類型的綁定設計方式來說明如何使用。

- 更多資料綁定的用法

  這裡將會繼續介紹更多關於資料綁定的不同使用方式。

- 數值轉換器 Value Converter

  對於資料綁定的設計方法下，數值轉換器的應用扮演者相當重要的角色，透過設計不同的數值轉換器類別可以設計出許多可重複使用的商業邏輯，並且輕鬆地將不同型別的綁定目標與綁定來源屬性串接在一起。

- 命令綁定 Command Binding

  命令綁定是資料綁定的第二種類型，透過命令綁定可以不再需要使用以往需要透過事件訂閱的設計方式，與在程式碼後置區塊來進行相關商業邏輯的程式碼設計工作，全部都轉移到綁定來源的類別物件上。

- 事件轉命令行為 Event to Command Behavior

  Xamarin.Forms 並不是所有的檢視項目都有提供可綁定的命令屬性，但是一定會有提供事件觸發的設計方式，在這裡將會使用 Xamarin.Forms 的一個核心技術行為 Behaviors，將需要訂閱的事件與命令綁定在一起，這使得當事件被觸發的時候，可以執行所綁定的命令內的委派方法。

- 手勢操作 Gesture Recognizer

  在 Xamarin.Forms 內提供可以與使用者互動的項目不多，按鈕是其中一個，不過，Xamarin.Forms 提供了手勢辨識器功能，可以在讓何檢視項目上，宣告不同的手勢操作行為，當發生了這個手勢操作行為，將會觸發所指定的命令，例如，得知使用者點選了一個圖片 UI 控制項。

- MVVM Model-View-ViewModel 設計模式

  Xamarin.Forms 可以搭配 MVVM 的設計模式，讓 UI 視覺設計與呈現邏輯程式碼與商業邏輯程式碼分隔開來，這樣可以有助於程式開發流程、進行單元測試，因為，這解除了視覺控制項與程式碼之間的緊密耦合關係。

- 內建導航服務

  開發行動應用程式最為重要的設計工作，那就是能夠在不同的頁面之間進行切換，在這裡會先進行 Xamarin.Forms 預設提供的導航服務功能進行介紹，並且了解到更多設計上的問題，可能需要進一步的解決。

- 導航服務之封裝設計

  為了要解決 Xamarin.Forms 預設的導航服務的不足，已經可以在檢視模型中進行各種頁面導航操作，在這裡將會設計一個延伸導航服務類別，解決相關問題，讓開發過程更加的順暢。
- 相依服務 Dependency Service

  Xamarin.Forms 是個 UI 開發工具，它把 UI 設計抽象化了，並且可以讓使用 Xamarin.Forms 設計的 UI 畫面可以在不同平台下來顯示出來，可是，當需要某些功能一定需要透過原生 SDK API 才能夠運作的需求，並且取得原生 SDK API 的執行結果，這個時候就可以透過 Xamarin.Forms 提供的相依服務來滿足這樣的工作。
- 訊息中心 MessagingCenter

  訊息中心是一種 發行-訂閱模式，其中對於發行者這個角色可以在不知道任何訂閱者的情況下傳送訊息。同樣地，訂閱者也可以在不知道任何發行者的情況下訂閱特定訊息。透過這樣的特行，可以讓 Xamarin.Forms 的程式順利地執行原生平台下的 SDK API。

誰適合閱讀這本書

本書適合想要學會如何使用 Xamarin.Forms 工具來開發出跨平台的行動應用程式的開發者，這裡將會介紹各種 Xamarin.Forms 核心與應用開發技術與技巧，並且帶領大家了解到進階的開發技能，如：檢視模型定位器，延伸導航服務等。透過學習這些開發技術，將會有助於進行各種 Xamarin.Forms 應用程式開發能力的提升。

不過，讀者本身應該要具備 .NET / C# 的開發經驗與程式寫作技能，並且要有使用過 Visual Studio 2019 開發經驗。

這本書的範例專案將會是在 Windows 10 作業系統下，使用 Visual Studio 2019 開發工具開發出來的，由於使用 Xamarin.Forms 開發出來的專案可以在 Android / iOS / UWP 平台下執行，若想要體驗開發出來的專案且在 iOS 模擬器環境下執行效果，讀者需要額外

準備一台 Mac 電腦，並且在這台電腦上需要安裝 Xcode 與 Visual Studio for Mac 開發工具。

如何使用本書

在書中每個章節都設計了一個練習專案，透過逐步說明的方式來帶領讀者來了解到 Xamarin.Forms 專案是如何進行開發的，了解到為什麼需要使用這樣的開發方式與和其他設計方式差異。

本書中的所有講解範例專案都會放在 Github 上，您可以透過 Github 的 Xamarin-Forms-Quick-Start[1] 來取得這些講解範例專案，並且鼓勵大家可以到這個 Xamarin-Forms-Quick-Start Repository[2] 頁面，在螢幕的右上方，點選 Start 按鈕給予鼓勵，如同下圖箭頭所指向地方。

意見回饋

對於在學習 Xamarin.Forms 開發上，有任何的疑問與問題，可以到 Facebook Xamarin.Forms @ Taiwan[3] 社團與其他 Xamarin.Forms 開發者進行討論。

也建議加入 Facebook Xamarin 實驗室[4] 粉絲團，作者會經常在這裡貼出各種 Xamarin.Forms 的開發新資訊。

若您對於 Github 範例專案有任何問題，可以在這個 Github Repository Xamarin-Forms-Quick-Start[5] 上，建立一個 Issue，作者將會在這上面與您做討論。

[1] https://github.com/vulcanlee/Xamarin-Forms-Quick-Start
[2] https://github.com/vulcanlee/Xamarin-Forms-Quick-Start
[3] https://www.facebook.com/groups/XamarinFormstw
[4] https://www.facebook.com/vulcanlabtw/
[5] https://github.com/vulcanlee/Xamarin-Forms-Quick-Start

I 開發前的安裝、設定準備工作

特別注意事項

根據作者本身從事多年的 Xamarin.Forms 開發教學經驗，絕大多數的 Xamarin.Forms 開發新手，在開始進行 Visual Studio for Xamarin 開發工具的安裝與設定過程中，就會遇到許多問題，尤其會在某些地方卡關，造成還沒開始學習 Xamarin.Forms 的開發技術，光是建立專案與在不同作業系統平台下執行專案，都無法正常操作。因此，若您是第一次接觸 Xamarin.Forms 的讀者，建議您參考本章內容，逐步進行 Visual Studio for Xamarin 開發工具的相關安裝與設定工作。

在這裡先進行 Xamarin 開發環境的安裝與設定，總共分成 3 部分來說明

對於要使用 Xamarin.Forms 跨平台開發工具來進行行動裝置應用程式開發，首先要先進行 Xamarin 開發環境的安裝與設定，在這裡將會分成 4 部分來說明，而這裡將會使用 Visual Studio 2019 社群版本作為安裝過程的說明。在此將會假設開發者主要在 Windows 作業系統下使用 Visual Studio 2019 for Windows 來進行開發、設計、建置、除錯 Xamarin.Forms 的專案，而 Mac 電腦上的 Visaul Studio 2019 for Mac 其存在的目的只是要能夠產生出 Xamarin.iOS 的專案檔案而已，通常，是不會使用 Visual Studio for Mac 來進行 Xamarin.Forms 開發作業。

- 安裝前的準備工作

這裡將會說明當準備要進行安裝 Visual Studio 2019 for Xamarin.Forms 開發環境之前，需要確認與準備工作有那些，這裡的內容相當重要，因為，若電腦環境一旦規格不符合需求，整個安裝與設定過程，將會產生問題，也會造成某些功能無法使用。

- Windows 電腦與 Mac 電腦上的 Visual Studio 2019 安裝與相關相關設定

  在這裡，將會帶領大家，一步一步地在 Windows 作業系統下進行 Visual Studio 2019 開發工具的安裝，並且在 Visual Studio 2019 安裝完成之後，接下來該如何進行 Xamarin 開發環境的設定。另外，也會說明在 Mac 電腦上要如何安裝 Xcode 與 Visual Studio 2019 for Mac 這兩個開發工具。

- 安裝 Android 使用的模擬器與確認開發環境可以進行 Xamarin.Forms 專案開發

  透過這裡的說明，可以進行安裝設定 Google Android 原生模擬器，方便進行 Android 應用程式的開發與測試，當然，要使用哪種類型的模擬器，可以依照開發者的喜好來決定。

  最後，這也是最為重要的，在安裝與設定工作都完成之後，需要開始進行檢查與確認電腦環境是否可以真的進行 Xamarin 跨平台應用程式的開發，因此，一定要依據這裡說明內容，逐一進行檢測，確認開發環境都正確無誤並且可以運作正常。

1. 安裝前的準備工作

在進行 Visual Studio 2019 開發工具安裝之前，請務必要進行底下說明的各個檢查與準備工作，否則，在進行安裝 Visual Studio 2019 與 Xamarin 開發工具的時候，可能會發生莫名的異常問題。

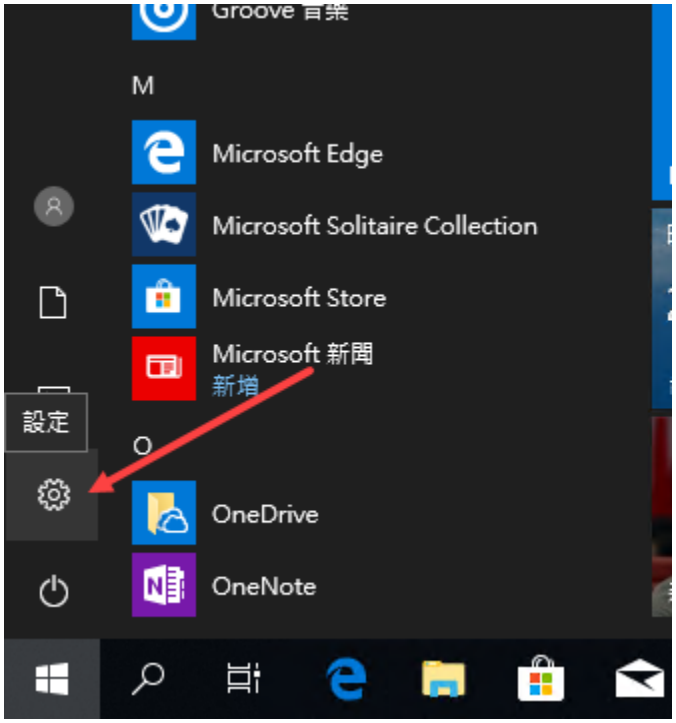
這裡需要的工作有

- 確認作業系統版本
- 確認硬體 BIOS 有啟用虛擬化功能
- 停用 Hyper-V 和 Windows Hypervisor 平台服務
- 啟用 UWP 開發人員模式
- 準備一台 macOS 的電腦主機

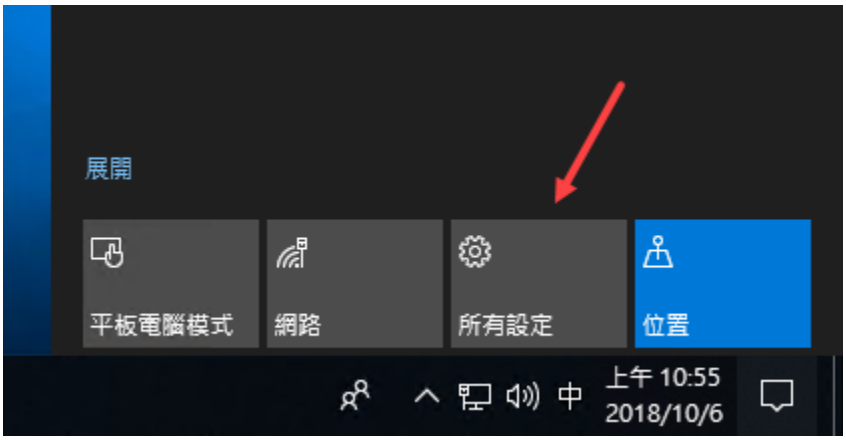
1.1 確認作業系統版本

以下說明內容將會使用的 Windows 10 專業版本作業系統，原則上，建議您使用當前最新的 Windows 10 作業系統的最新版本來安裝，若電腦不是這個版本，請使用 Windows Update 進行更新到組建版本。

欲檢查您的 Windows 10 作業系統版本，請點選左下角的視窗圖示，接著點選左下方的設定齒輪圖示，或者點選右下方通知圖示，選取齒輪圖示的設定按鈕

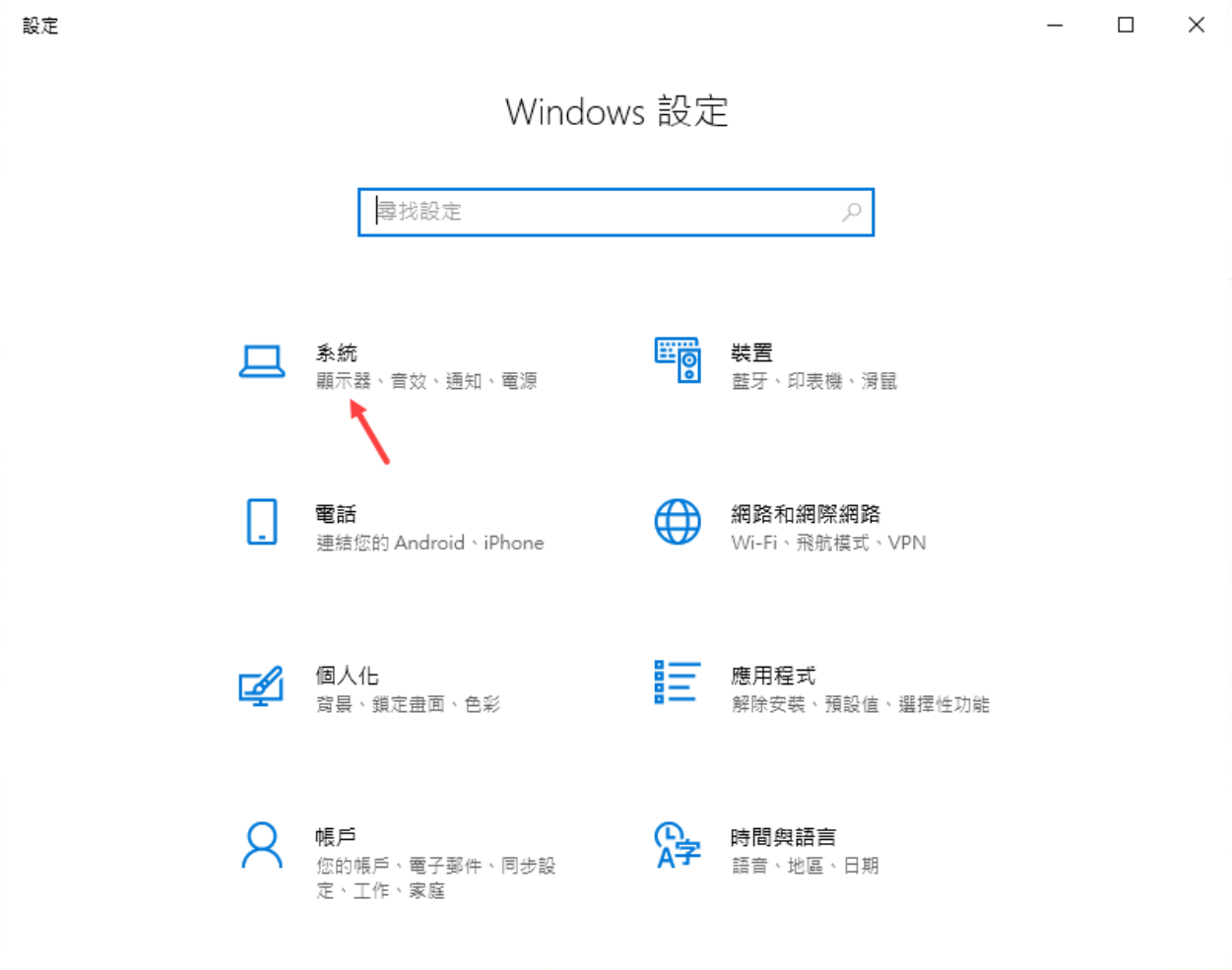


使用開始圖示來開啟 Windows 設定視窗



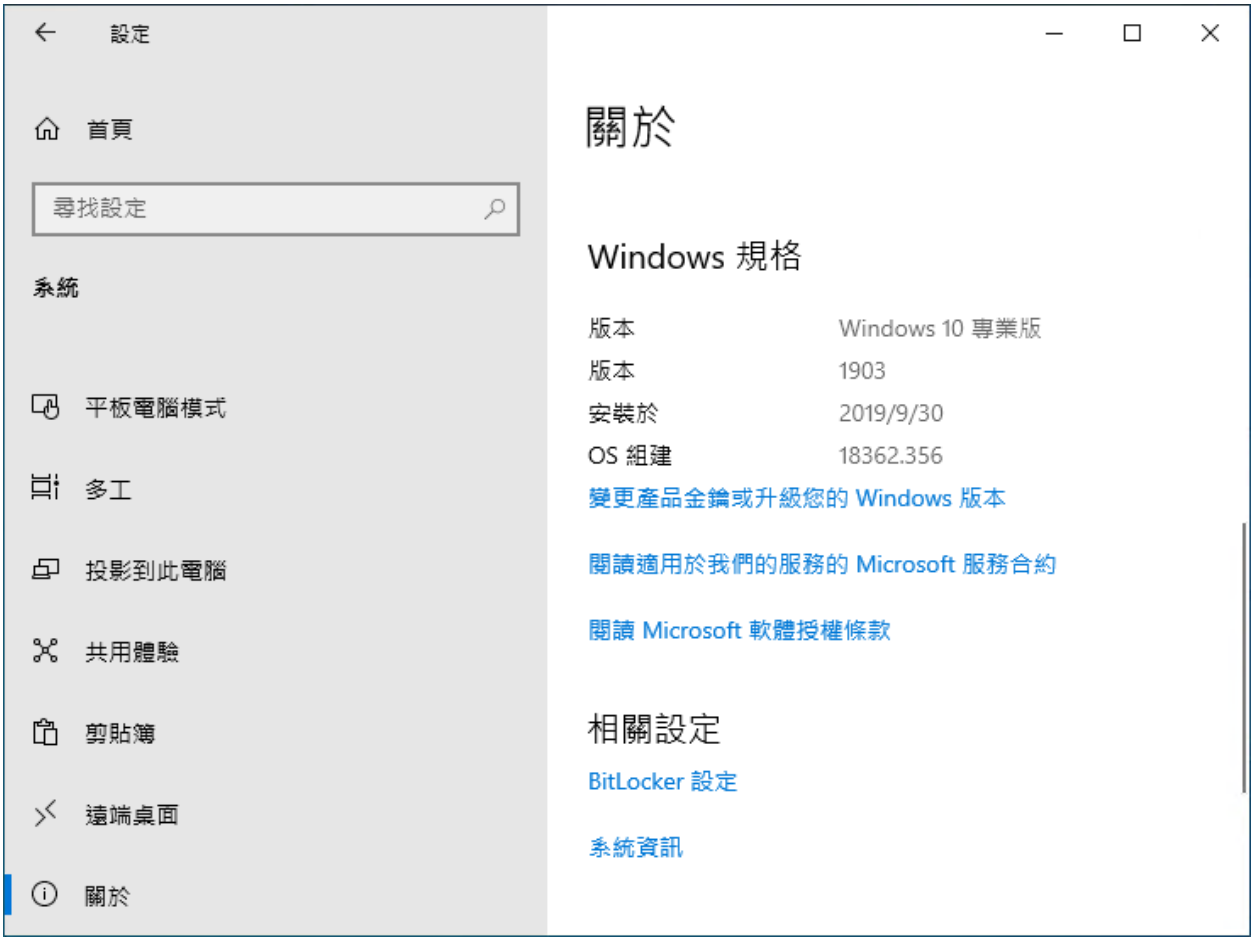
使用通知設定視窗來開啟 Windows 設定視窗

當進入到設定視窗之後，點選上方的系統圖示



在 Windows 設定窗內開啟系統功能視窗

緊接著在左邊的最下方選項，點選關於項目，就會看到視窗右邊出現的內容，請在視窗右邊區域，使用滑鼠往下捲動，就會查看到 Windows 規格區域，確認您使用作業系統版本至少為 Windows 10 專業版與組件 1903



Windows Setting 設定

1.2 確認硬體 BIOS 有啟用虛擬化功能

因為絕大多數的 Android 模擬器都會使用虛擬化技術，因此，，Windows 電腦 CPU 需要具備支援虛擬化的功能，若想要啟用電腦支援虛擬化功能，請確認 CPU 有支援此一功能，並且從 BIOS 啟用此功能。這個影片 how to enable hyper-v machine in BIOS[1] 將會展示如何設定 BIOS 可以使用虛擬化的功能。若電腦 BIOS 上沒有啟動此功能 **Intel Virtual Technology** 或者電腦的 BIOS 根本就沒有這項設定，建議使用實體手機來進行 Android

[1] https://www.youtube.com/watch?v=EGnv6zyLj-o

專案的開發與設計。

特別注意事項

原則上不同的虛擬化產品是無法混合一起使用的，例如 Hyper-V[2] 無法與其他的虛擬化產品 Intel HAXM[3] 或 VMware[4] 等等，共同使用，您只能夠選擇其一來使用。

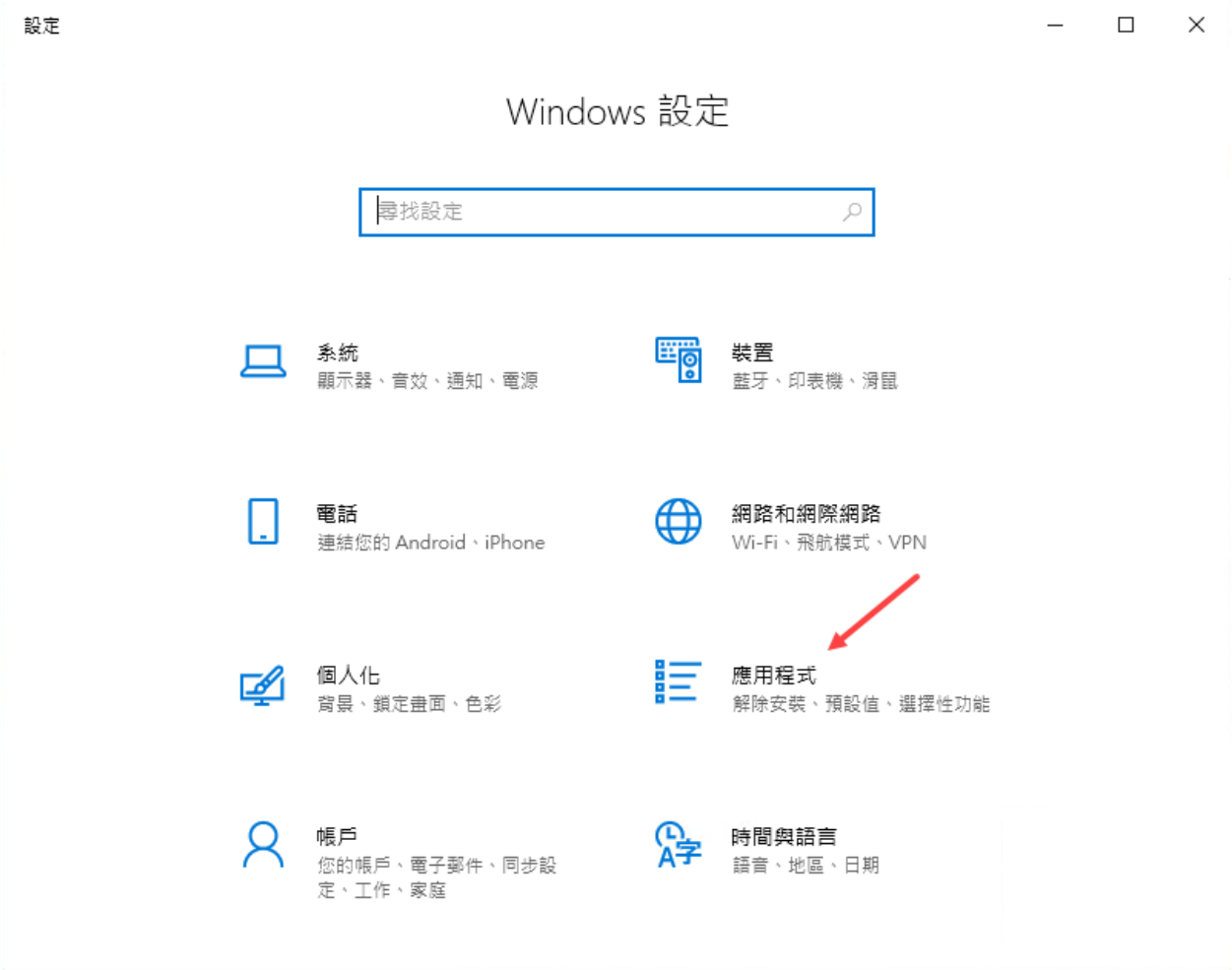
在這篇文章中將不會採用 Hyper-V 虛擬化技術，而會採用 Intel HAXM 虛擬化技術，對於 Android 模擬器部分，將會使用 [Android Emulator] 這個模擬器軟體。

1.2.1 停用 Hyper-V

若想要使用 Intel HAXM 虛擬化技術，進而直接使用 Android 原生地 x86 模擬器，需要先將這台電腦上的 Hyper-V 和 Windows Hypervisor 平台這兩項功能移除與停用；在這台電腦中，已經有安裝這兩個服務，所以需要先進行移除，若電腦中並沒有安裝這兩個服務，可以直接進行下一個步驟。

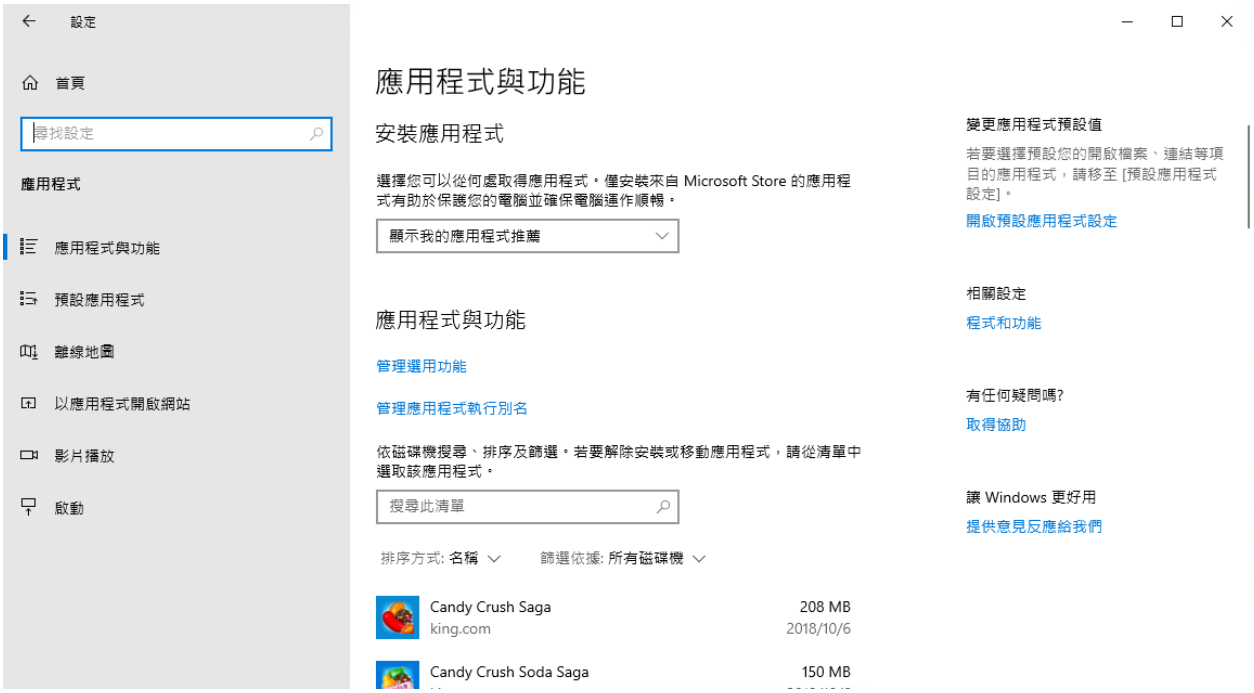
- 首先，依據上面的操作步驟，開啟 Windows 設定視窗 (請點選左下角的視窗圖示，接著點選左下方的設定齒輪圖示，或者點選右下方通知圖示，接著選取齒輪圖示的設定按鈕)

[2] https://docs.microsoft.com/zh-tw/virtualization/hyper-v-on-windows/reference/hyper-v-architecture
[3] https://software.intel.com/en-us/articles/intel-hardware-accelerated-execution-manager-intel-haxm
[4] https://www.vmware.com/tw.html



Windows 設定視窗

- 點選應用程式項目，將會出現下圖應用程式與功能視窗畫面

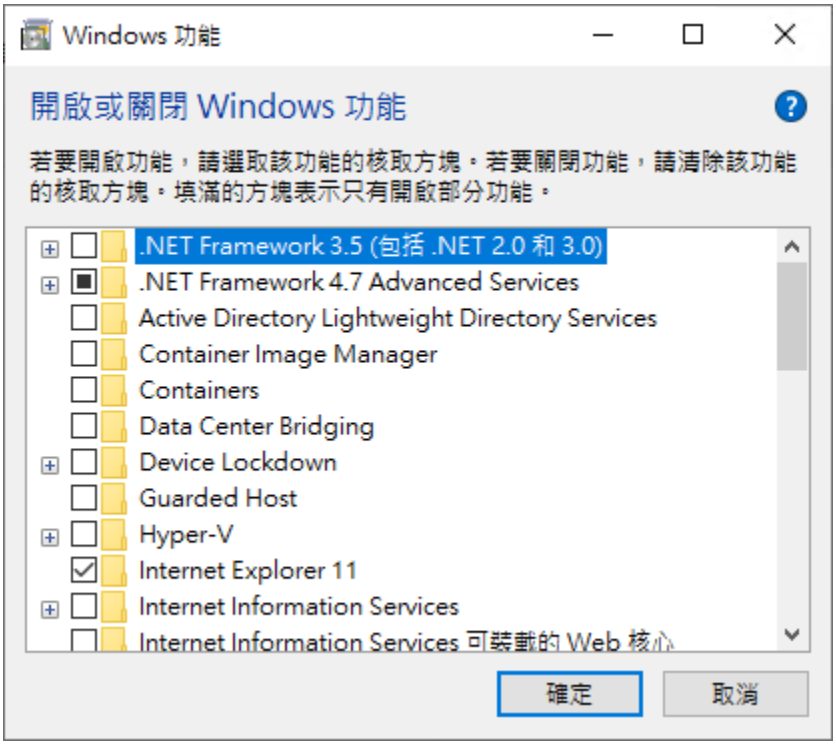


應用程式與功能視窗

- 在應用程式畫面右方，點選相關設定的程式和功能項目

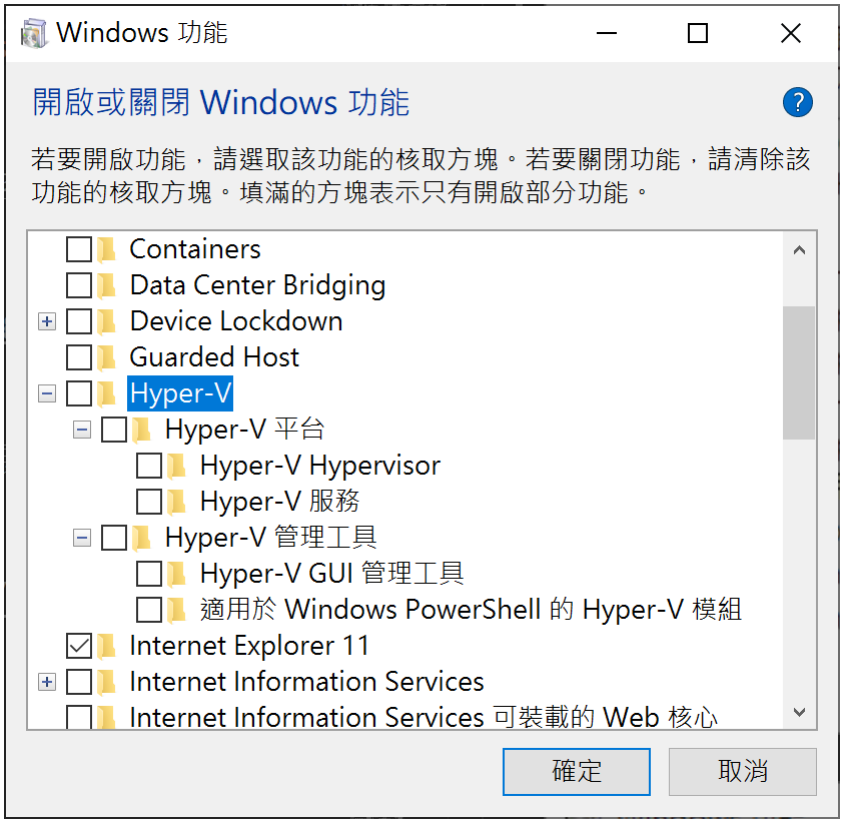
程式和功能視窗

- 當程式和功能視窗開啟之後，點選該視窗左方的開啟或關閉 Windows 功能

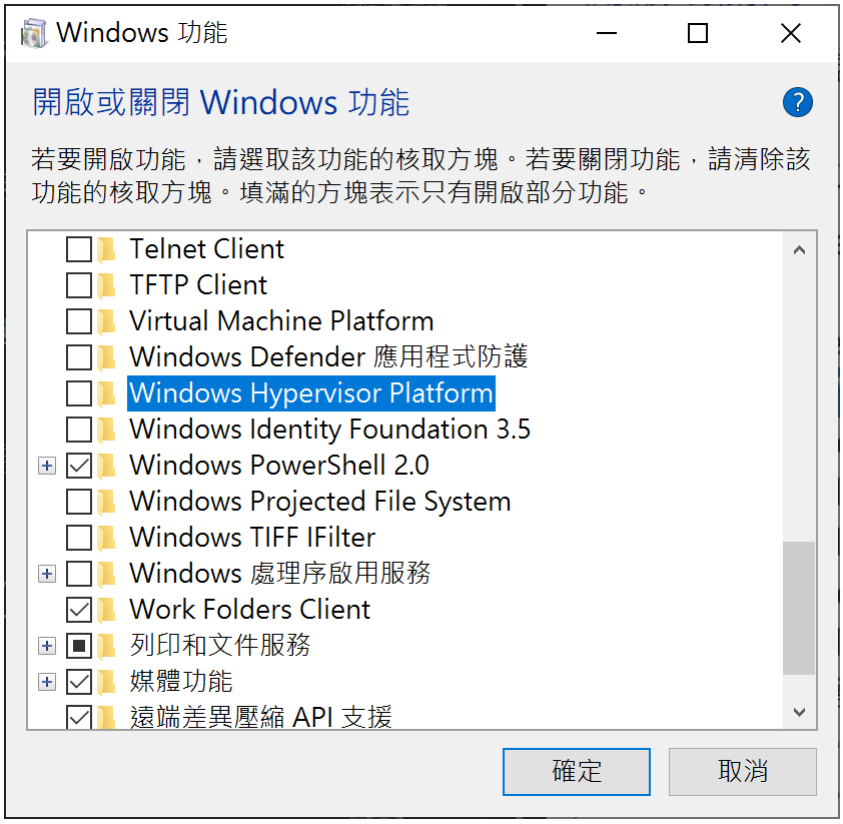


Windows 功能之設定視窗

- 現在，將會看到 [Windows 功能] 視窗出現在電腦上，請在下方的清單項目，找到 **Hyper-V** 與 **Windows Hypervisor Platform** 這兩個選項，請記得要取消勾選這兩個項目與其子項目，完成後請點選確定按鈕。

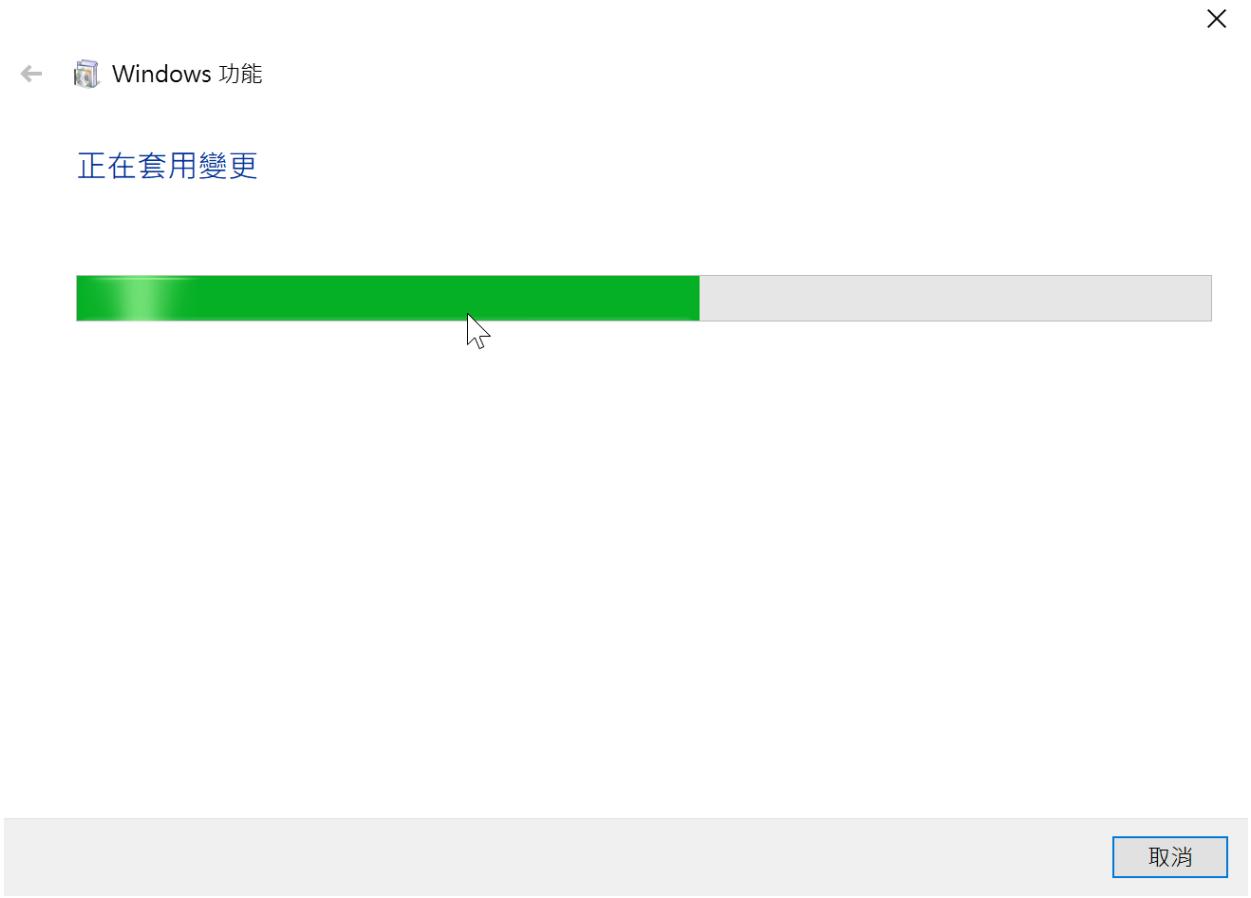


取消選擇安裝 Hyper-V 的所有功能



取消選擇安裝與啟用 Windows Hypervisor Platform

- 現在，Windows 將會開始安裝這兩個功能，請點選確定按鈕。



Windows 正在進行套用變更

- 當這兩個 Windows 功能移除這兩個服務之後，需要重新啟動作業系統，請點選立即重新啟動按鈕。

Windows 已完成要求的變更

1.3 啟用 UWP 開發人員模式

若想要使用 Xamarin.Forms 建立出可以同時在 Android / iOS / UWP 系統下執行的跨平台 App，就需要參考底下的步驟，先進行 UWP 開發人員模式的啟用

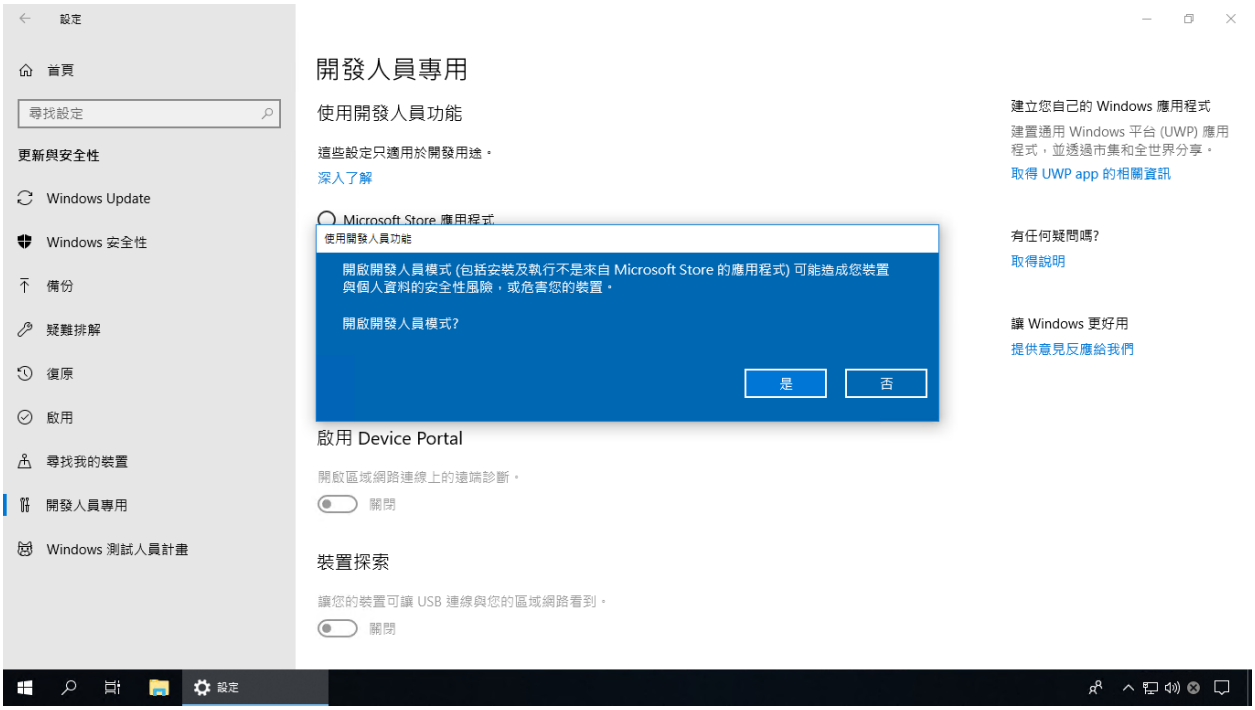
- 首先，還是一樣需要進入到 Windows 設定 (請點選左下角的視窗圖示，接著點選左下方的設定齒輪圖示，或者點選右下方通知圖示，接著選取齒輪圖示的設定按鈕)

Windows Setting 設定

- 請點選更新與安全性選項，並且點選更新與安全性視窗左方的開發人員專用選項

開發人員專用設定視窗

- 請點選右方內容的開發人員模式選項，現在顯示出使用開發人員功能對話窗，並且說明：開啟開發人員模式 (包括安裝及執行不是來自於 Microsoft Store 的應用程式) 可能造成您裝置與個人資料的安全性風險或危害您的裝置。開啟開發人員模式？請點選是按鈕。



開啟開發人員模式

- 若開發人員模式已經開啟之後，可以看到如同下圖畫面，在開發人員模式選下的下方，會顯示出：已安裝「開發人員模式」套件，現已啟用適用於桌面的遠端工具訊息。

已安裝「開發人員模式」套件

1.4 準備一台 macOS 的電腦主機

由於 Apple 公司的規定，當要進行 iOS 應用程式開發的時候，必須要透過一台有安裝 macOS 的蘋果硬體電腦來進行，因此，對於想要產生與進行 iOS 應用程式除錯的開發者，就需要準備一台 macOS 電腦主機。

對於這台 macOS 電腦，強烈建議要將作業系統升級到最新版本，以便符合 Xcode 開發工具的安裝要求，除了會安裝 Xcode 到這台電腦上，也會進行安裝 Visual Studio 2019 for Mac 工具。

2. Windows 電腦與 Mac 電腦上的 Visual Studio 2019 安裝與相關相關設定

現在，需要在 Windows 主機與 Mac 主機上來進行開發工具的安裝與設定。

這裡需要的工作有

- 在 Windows 作業系統電腦上安裝 Visual Studio 2019
- 在 Mac 作業系統電腦開發工具之安裝與設定

2.1 在 Windows 作業系統電腦上安裝 Visual Studio 2019

若您已經完成 Visual Studio 2019 安裝前的準備工作，現在需要開始進行 Visual Studio 2019 的安裝作業。

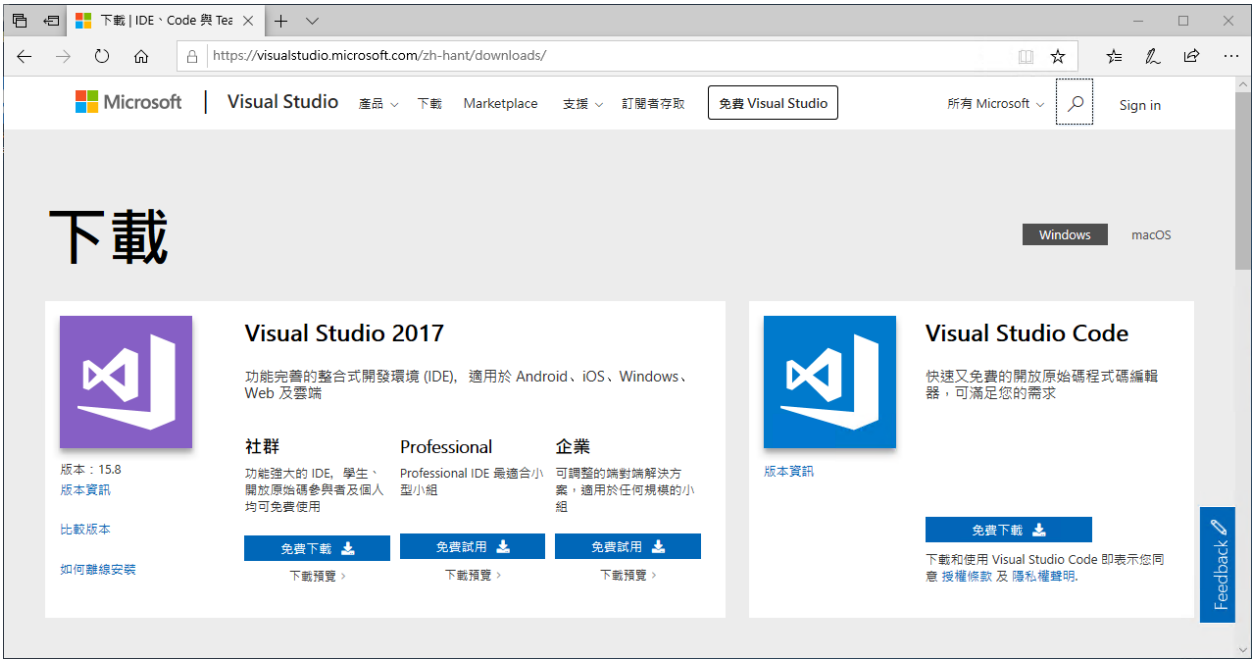
在這裡將會使用 Visual Studio 2019 Community 版本作為安裝說明，不過，任何一種 Visual Studio 2019 的版本，皆可以進行 Xamarin Cross-Platform 跨平台專案的開發，關於 Visual Studio 2019 的不同版本的比較，可以參考 比較 Visual Studio 2019 IDE[1]

更多關於 Visual Studio 2019 的安裝，可以參考 安裝 Visual Studio 2019[2]

[1] https://www.visualstudio.com/zh-hant/vs/compare/

[2] https://docs.microsoft.com/zh-tw/visualstudio/install/install-visual-studio

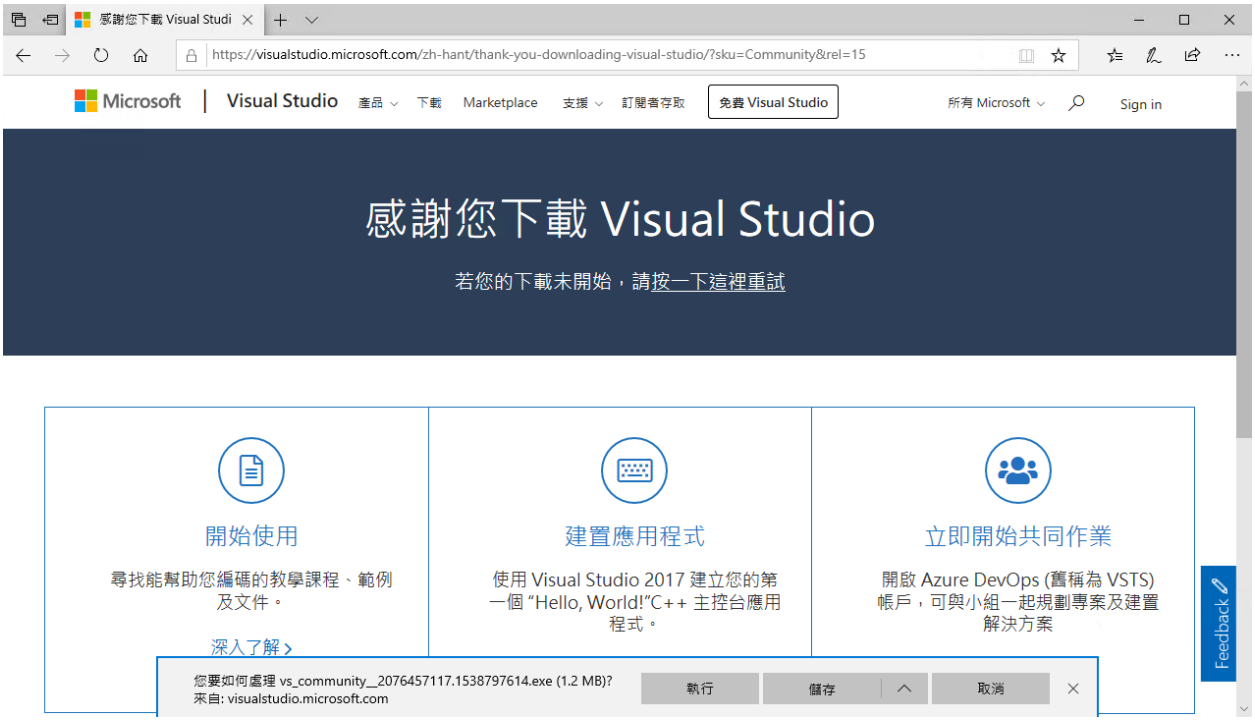
- 請用瀏覽器打開 Visual Studio 2019 安裝下載網頁[3]



Visual Studio 2019 安裝下載網頁

- 找到 Visual Studio 2019 下方的社群項目，點選其下方的免費下載按鈕，點選這個按鈕連結，當安裝檔案下載完成後，請點選執行按鈕，開始進行 Visual Studio 2019 社群版本的安裝；當安裝程式下載完成之後，請點選螢幕下方的執行按鈕，開始進行 Visual Studo 2019 社群版的安裝 (如同下面節圖所示)。

[3] https://www.visualstudio.com/zh-hant/downloads

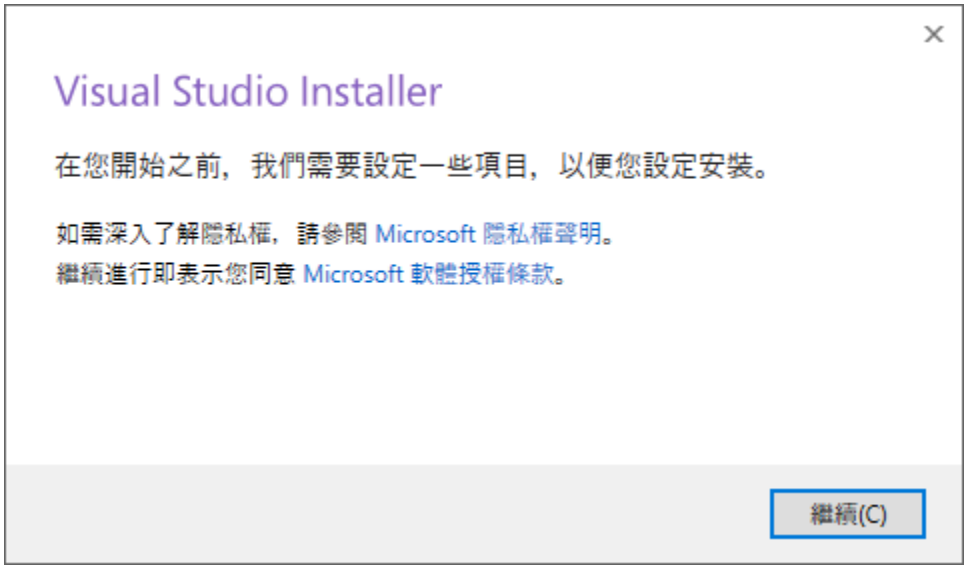


Visual Studio 2019 安裝程式下載完成

- 此時，螢幕會出現使用者帳戶控制對話窗，此對話窗詢問：您是否要允許此 App 變更您的裝置？這個時候，請點選是按鈕。

使用者帳戶控制對話窗

- 現在，看到的對話窗是 Visual Studio Installer，請在該對話窗右下方，點擊繼續按鈕。如此，Visual Studio Install 安裝程式，便會開始下載最新的安裝程式檔案到您的電腦。



Visual Studio Installer 開始安裝前的確認對話窗

- 將會看到 Visual Studio Community 2019 - 16.0.0 安裝程式啟動了。

  16.0.0 是作者寫這篇文章的時候，最新 Visual Studio Community 的版本編號，原則上，您每次進行 Visual Studio 2019 安裝作業的時候，您將會使用最新的 Visual Studio 2019 Installer 來安裝到最新的版本，因此，您將需要比 16.0.0 更新的版本編號出現在您的電腦畫面中。

正在安裝 Visual Studio Community 2019 - 16.0.0 視窗

- 請在工作負載標籤頁次中，至少需要勾選底下項目，才能夠完成 Xamarin 開發工具的安裝
  - 通用 Windows 開發平台
  - 使用 .NET 進行行動開發

  當您勾選這兩個工作負載項目之後，在 Visual Studio Community 2019 Installer 視窗的最右方，將會出現準備要安裝的相關元件清單說明

勾選要安裝通用 Windows 開發平台工作負載

勾選要安裝使用 .NET 進行行動開發工作負載

- 請點選個別元件標籤頁次，您將會看到底下的個別元件已經被勾選了

Visual Studio 2019 Installer 個別元件

安裝不同的 Visual Studio 2019 版本，可能會看到不通的套件選項項目內容

- .NET
  * .NET Framework 4.5 目標套件
  * .NET Framework 4.6.1 目標套件
  * .NET Framework 4.6.1 SDK
  * .NET Framework 4.7.2 SDK
  * .NET Framework 4.7.2 目標套件
  * .NET Native
  * .NET 可攜式程式庫目標套件
- Code 工具
  * (無)
- SDK、程式庫和架構
  * Android SDK 安裝程式 (API 層級 27)
  * OpenJDK (Microsoft 散發)
  * TypeScript 3.3 SDK

        * Windows 10 SDK (10.0.17763.0)
    - 偵錯和測試
        * .NET 分析工具
        * JavaScript 診斷
    - 模擬器
        * (無)
    - 程式碼工具
        * Developer Analytics Tools
        * NuGet 套件管理員
    - 編譯器、建置工具與執行階段
        * (無)
    - 編譯器、建置工具和執行階段
        * C# 與 Visual Basic Roslyn 編譯程式
        * MSBuild
    - 遊戲開發套件
        * 影像與 3D 模型編輯器
    - 開發活動
        * C# 與 Visual Basic
        * F# 語言支援
        * JavaScript 與 TypeScript 語言支援
        * Xamarin
        * Xamarin Remoted Simulator
    - 雲端、資料庫和伺服器
        * SQL Server 的 CLR 資料類型
- 為了要讓 Xamarin 開發環境可以正常運作與符合日常開發需求，所以，請依照底下清單，比對您電腦上的個別元件勾選項目，確認底下的項目都已經有勾選到 (額外需要安裝的項目將會以**粗體**文字標示)。
    - .NET

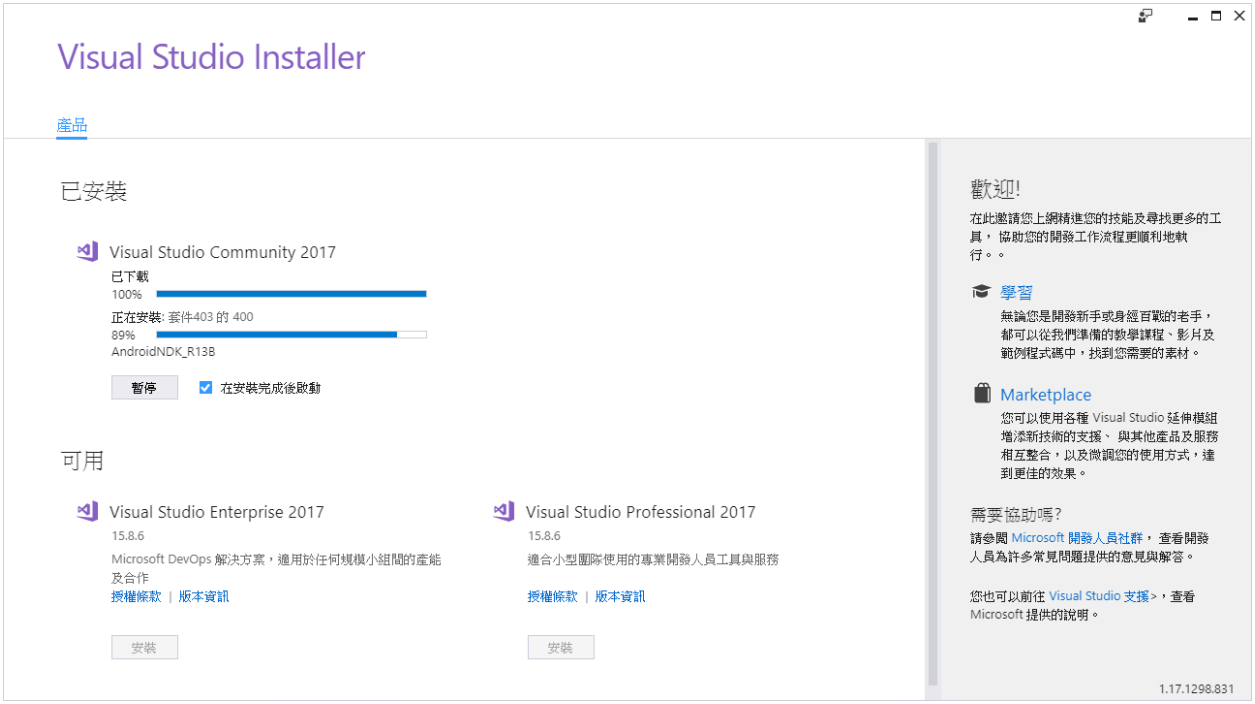
    * .NET Framework 4.5 目標套件
    * .NET Framework 4.6.1 目標套件
    * .NET Framework 4.6.1 SDK
    * .NET Framework 4.7.2 SDK
    * .NET Framework 4.7.2 目標套件
    * .NET Native
    * .NET 可攜式程式庫目標套件
- Code 工具
    * **Visual Studio 的 GitHub 擴充功能**
- SDK、程式庫和架構
    * Android SDK 安裝程式 (API 層級 25)(可用於以 C++ 進行行動裝置開發的本機安裝)
    * Android SDK 安裝程式 (API 層級 27)
    * OpenJDK (Microsoft 散發)
    * TypeScript 3.3 SDK
    * **USB 裝置連線**
    * Windows 10 SDK (10.0.17763.0)
- 偵錯和測試
    * .NET 分析工具
    * JavaScript 診斷
- 模擬器
    * **Google Android Emulator (API 層級 25) (本機安裝)**
    * **Intel Hardware Accelerated Execute Manager (HAXM) (本機安裝)**
- 程式碼工具
    * Developer Analytics Tools
    * **Git for Windows**
    * NuGet 套件管理員
    * **PreEmptive Protection - DotFuscator**

- 編譯器、建置工具與執行階段
  - (無)
- 編譯器、建置工具和執行階段
  - C# 與 Visual Basic Roslyn 編譯程式
  - MSBuild
- 遊戲開發套件
  - 影像與 3D 模型編輯器
- 開發活動
  - C# 與 Visual Basic
  - F# 語言支援
  - JavaScript 與 TypeScript 語言支援
  - Xamarin
  - Xamarin Remoted Simulator
- 雲端、資料庫和伺服器
  - SQL Server 的 CLR 資料類型

Visual Studio 2019 Installer 安裝詳細資料

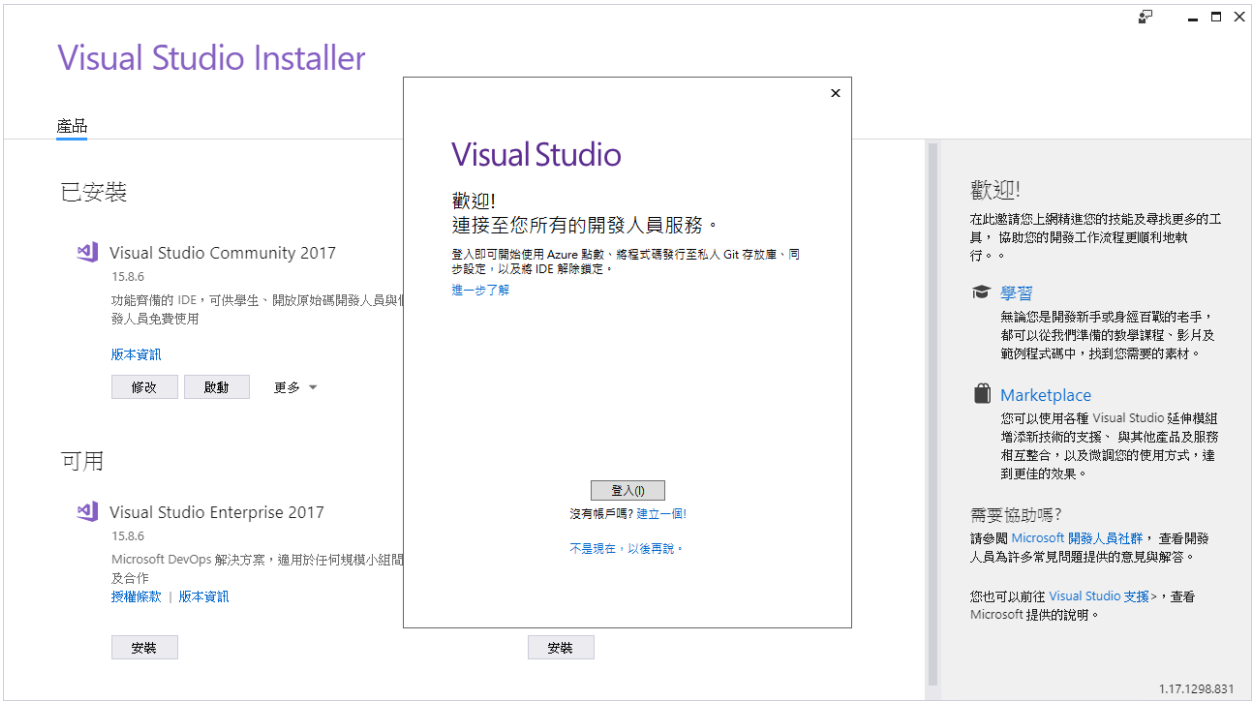
- 若確認無誤，請點選 Visual Studio Community 2019 Installer 程式畫面右下角的安裝按鈕，開始進行 Visual Studio Community 2019 的程式安裝。

  根據 Visual Studio Installer 的預估，這樣的安裝過程，大約需要用到 34.68 GB 的磁碟空間。



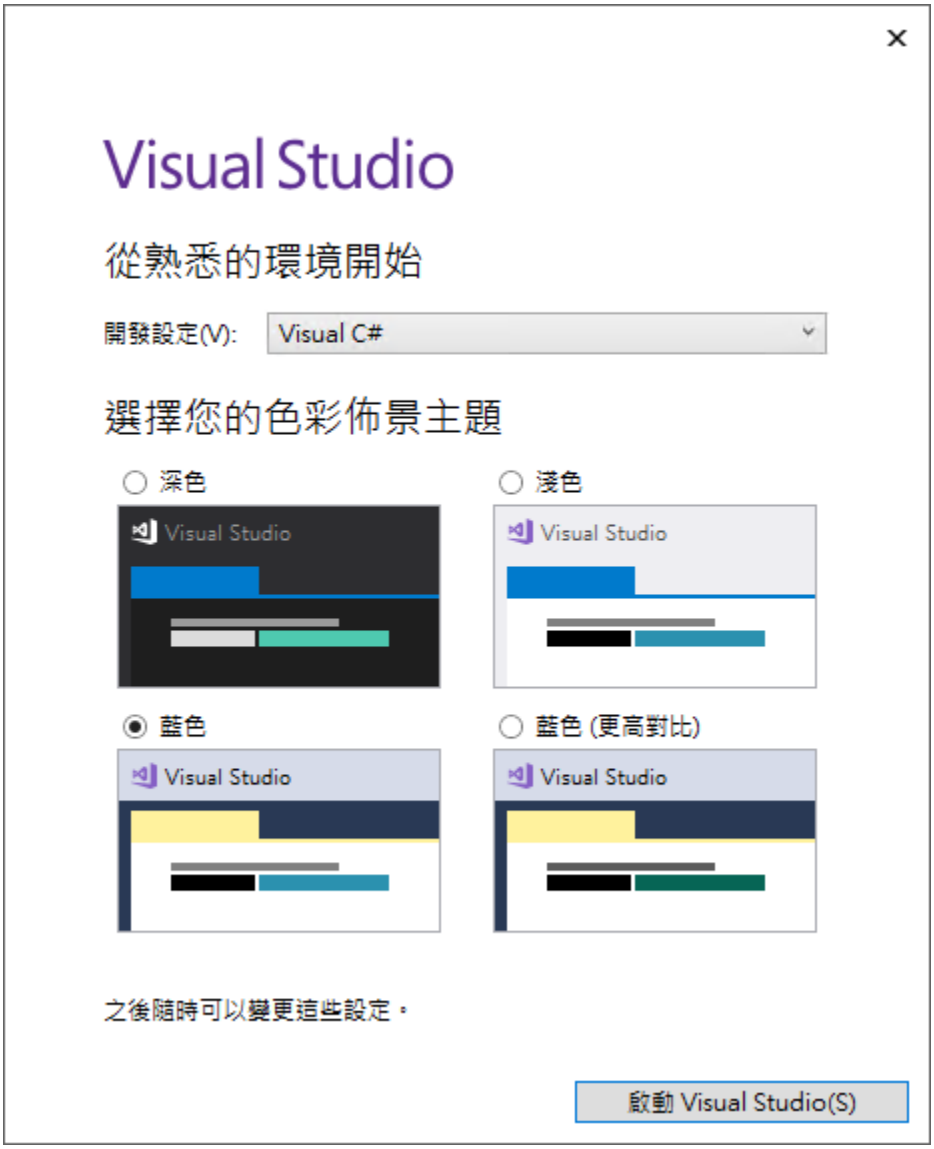
Visual Studio 2019 Installer

- 當安裝作業完成之後，將會看到如下圖的畫面，您可以選擇登入功能，或者點選不是現在，以後再說，在這裡點選後者，也就是點選 **不是現在，以後再說**。



Visual Studio 2019 登入畫面

- 現在需要設定開發設定條件，請在開發設定下拉選單中，選擇 Visual C#，完成後，點選啟動 Visual Studio 按鈕。



Visual Studio 2019 開發設定與主題佈景

- 底下螢幕截圖是 Visual Studio 2019 啟動後的畫面

Visual Studio 2019 第一次啟動畫面

2.2 在 Mac 作業系統電腦開發工具之安裝與設定

現在，要來開始進行安裝 Xamarin.iOS 需要用到的開發工具，在這裡所有的操作過程，將會採用一台全新新安裝的 macOS 系統，來進行執行過程步驟的解說說明。

在這裡的 macOS 將會是 Mojave 10.14.3 版本，想要知道這台 Mac 電腦的作業系統版本，可以從位於畫面左上角落的「蘋果」選單中，選擇「關於這台 Mac」，就可以查看的到。

macOS Mojave 版本資訊

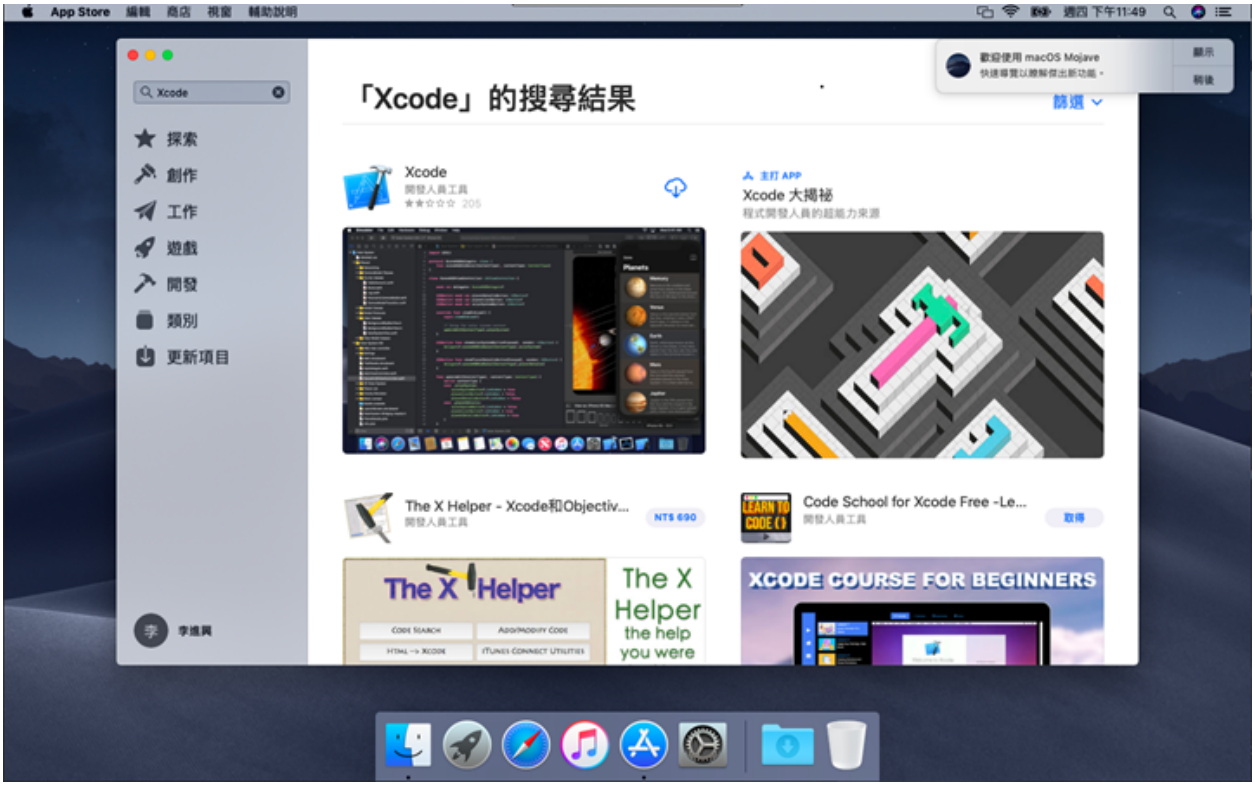
2.2.1 安裝 Xcode 開發人員工具

Xcode 提供建置 iOS 應用程式所需要用到的 SDK & IDE，因此，要在 Mac 電腦上能夠開發出 iOS 應用程式，就一定需要安裝這套開發工具。

- 打開 App Store 圖示
- 在左上方搜尋文字輸入盒，輸入 Xcode

搜尋 Xcode 這個關鍵字

- 當按下 Enter 按鍵之後，就會在第一個項目看到了 Xcode 這個 iOS 開發工具必備的軟體



搜尋出 Xcode 開發工具

- 點選 Xcode 這個項目，此時將會顯示 Xcode 這個開發人員工具的介紹內容，請點選該頁面右上方的下載圖示，以便進行下載與安裝這個軟體。

開始下載與安裝 Xcode

- 現在，為了要能夠從 App Store 下載這個 Xcode 軟體，此時，請在螢幕最上方的對話窗內，輸入您的 Apple ID 與密碼，完成後，請點選 [好] 按鈕。

輸入 Apple ID 和密碼

此時，剛剛點選的下載圖示

準備要下載的按鈕圖示

將會變成底下截圖，顯示出現在正在下載的進度，因此，在此需要等候 Xcode 這個開發人員工具下載完成，畢竟，這的軟體檔案大小也是很大的，需要花些時間來下載。

正在進行下載的按鈕圖示

- 當下載完成之後，在該視窗的右上方將會看到 [打開] 按鈕，請點選這個 [打開] 按鈕。

下載完成後的按鈕圖示

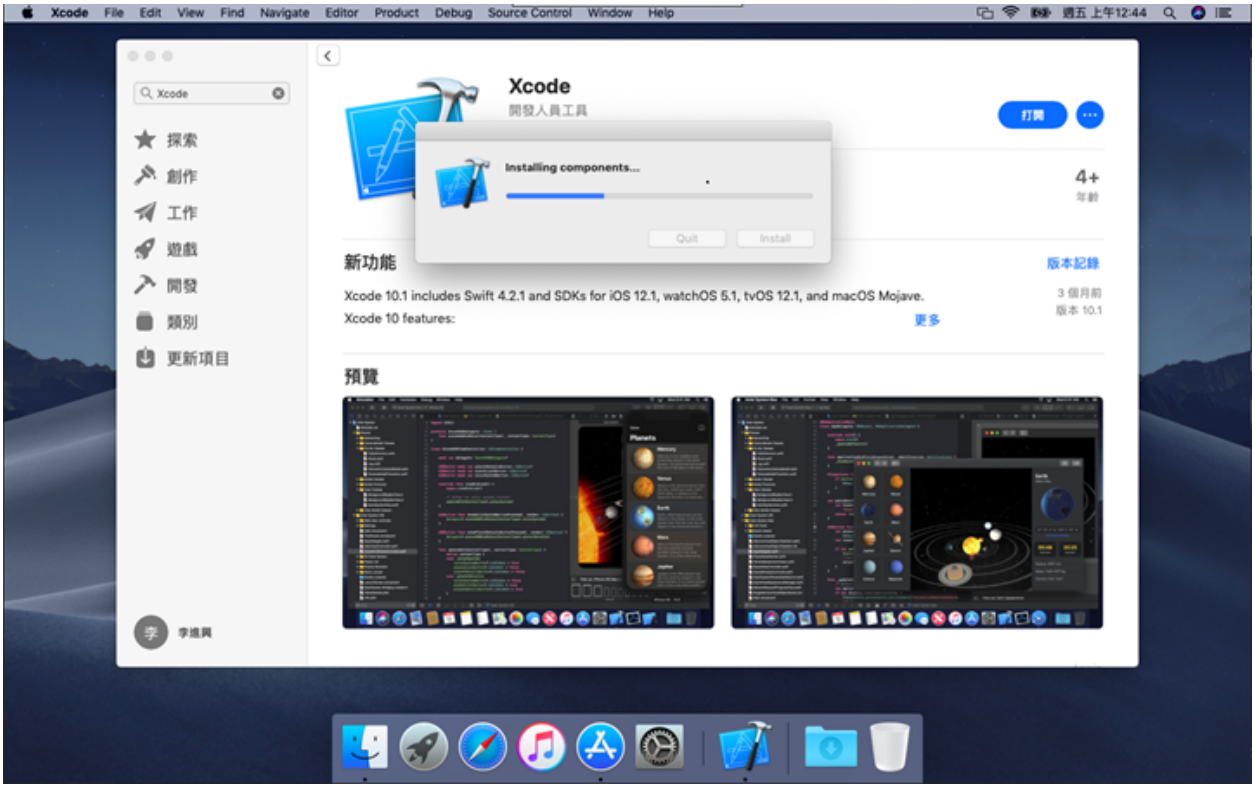
- 現在，將會看到 Xcode 已經啟動了，並且顯示一個 [Xcode and iOS SDK License Aggreement] 對話窗，這裡需要您同意接受 Xcode 的使用授權，因此，請點選該對話窗右下方的 [Agree] 按鈕

接受 Xcode 開發工具使用授權

- 此時，需要輸入這台系統的管理這帳號與密碼，以便進行 Xcode 需要處理的工作，因此，當輸入完成使用者名稱與密碼之後，請點選該對話窗右下方的 [好] 按鈕。

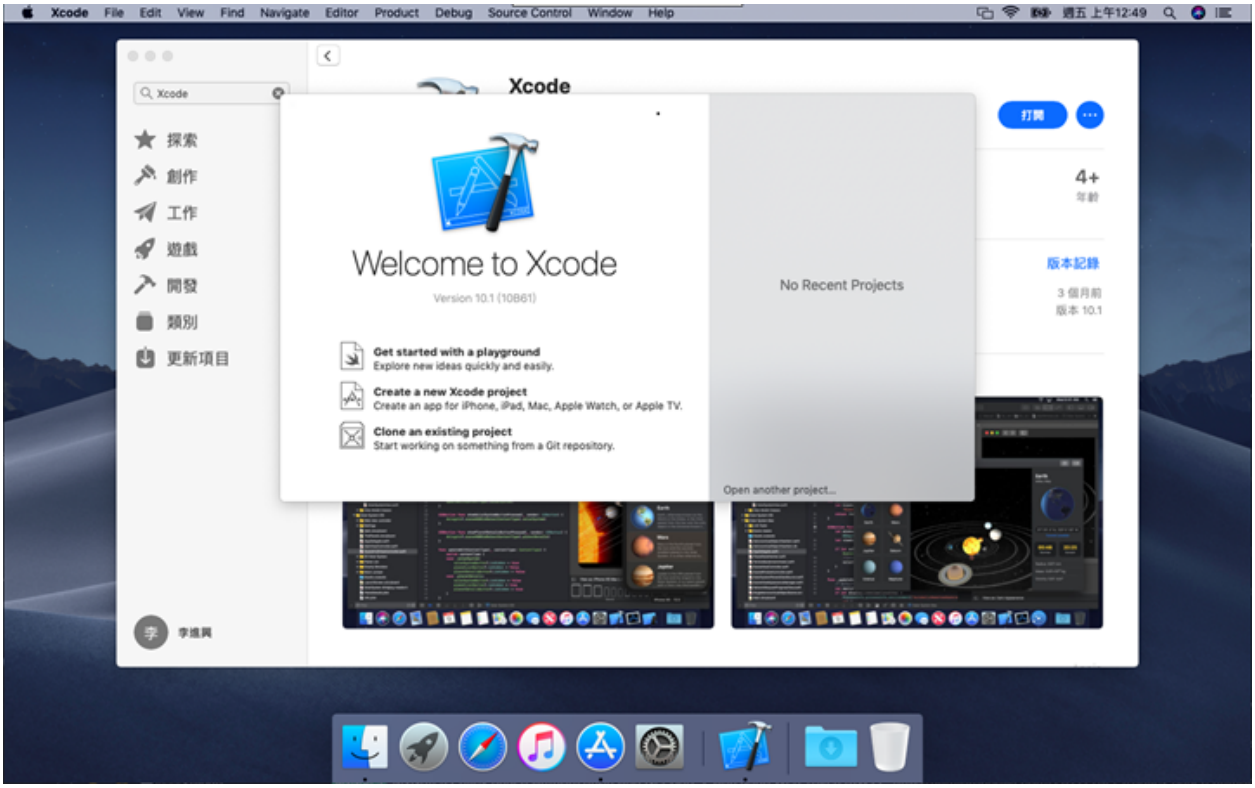
輸入安裝檔案到電腦上的帳號與密碼

- 現在，等候 Xcode 安裝相關元件到系統上



正在進行安裝 Xcode 軟體

- 當看到底下的畫面出現之後，那就表示在這台 Mac 電腦上的 Xcode 開發人員工具已經安裝好了。



Xcode 安裝完成的畫面

2.2.2 安裝 Visual Studio for Mac

Visual Studio for Mac 可以讓開發人員在這個 Mac 電腦上直接開發 Xamarin & ASP.NET Core 類型的專案，另外一個重要的功能那就是提供與 Windows 作業系統上的 Visual Studio 2019 相關建置 iOS 專案的能力。

- 現在，請打開 Safai 應用程式，搜尋 Visual Studio for Mac，將會看到第一個搜尋結果 [Visual Studio for Mac | Visual Studio]，請打開這個頁面

搜尋 Visual Studio for Mac 產品

- 在這裡，請點選左上方的 [下載 Visual Studio for Mac] 下載按鈕，開始下載這個安裝工具。

Visual Studio for Mac 官方網頁來下載安裝軟體

- 當安裝軟體下載完成之後，請在 [下載完成] 找到這個項目，並且打開這個檔案。

打開並執行下載完成的安裝檔案

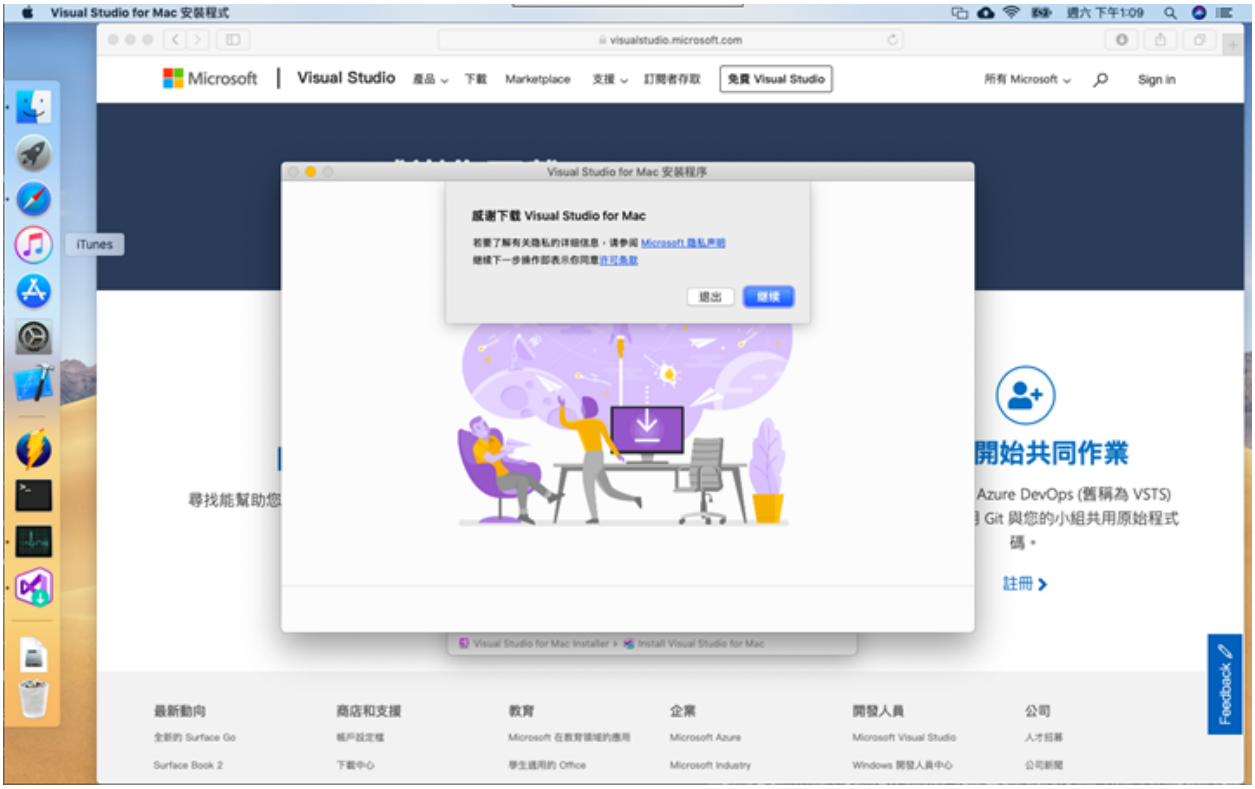
- 當這個檔案載入並且執行完成之後，將會出現如同底下畫面，請點選下載按鈕圖示，開始安裝 Viusl Studio for Mac

啟動 Visual Studio for Mac 安裝程式

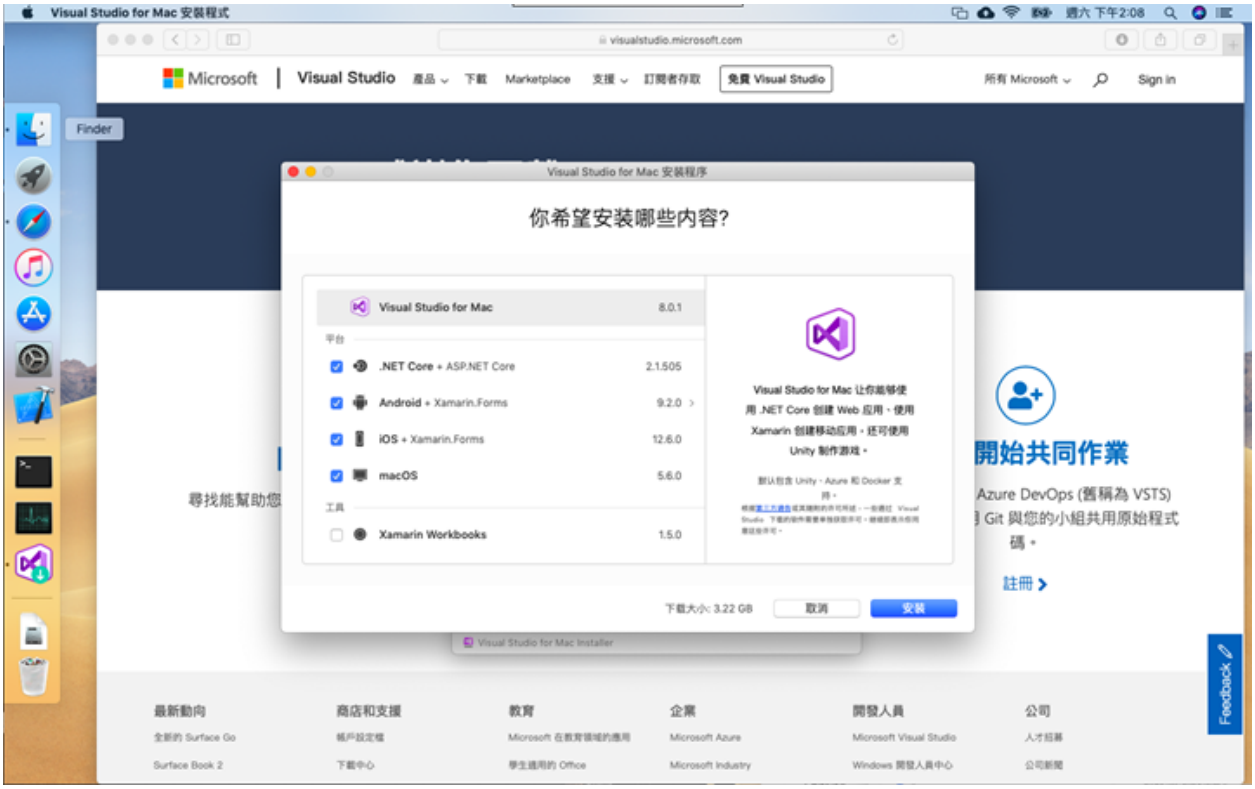
- 現在出現一個對話窗，顯示 [「Install Visual Studio for Mac」是一個從 Internet 下載的 App，確定要打開嗎？]，請點選該對話窗右下方的 [打開] 按鈕

確認要開啟

- 此時，Visual Studio for Mac 的安裝程式啟動起來了，並且顯一個對話窗，請點選該對話窗右下方的 [繼續] 按鈕

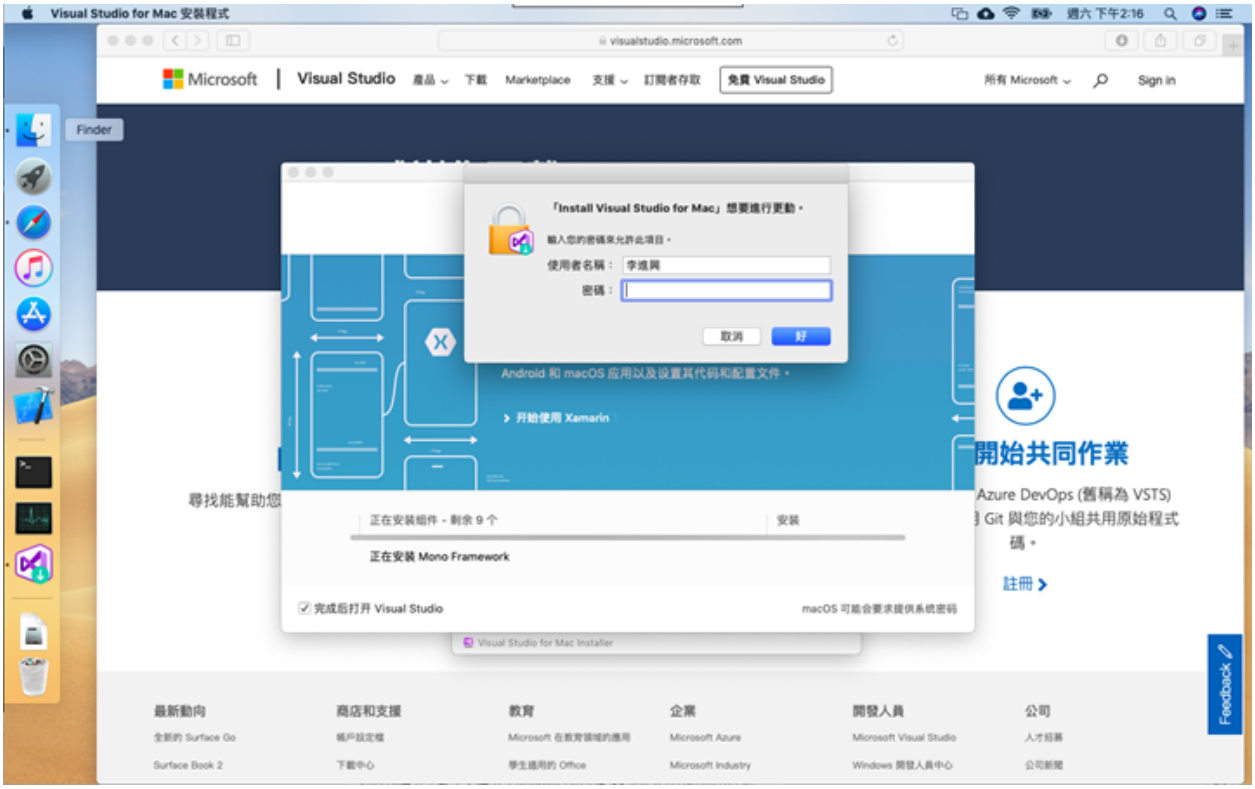


- 當出現如下圖，詢問希望安裝那些內容的時候，使用預設安裝選項即可，而最後一個選項 [Xamarin Workbooks] ，原則上大部分的人都不會使用到這個工具。因此，請點選該對話窗右下方的 [安裝] 按鈕。



要安裝 Visual Studio for Mac 的元件

- 現在，您將會看到安裝程式正在進行下載檔案，不過，此時，將會出現一個對話窗，要求您輸入這台系統的管理者帳號與密碼，以便安裝 Visual Studio for Mac 這個軟體，所以，請輸入使用者名稱與密碼之後，點選該對話窗右下方的 [好] 按鈕。從這個對話窗的右下方將會看到 [macOS 可能會多次需要系統密碼] 這樣的文字，而在此次安裝過程中，這個對話窗將會出現 2 次。



授權 Visual Studio for Mac 安裝檔案到 Mac 電腦上

- 當整個安裝過程完成之後，會出現底下視窗，請點選右下角的略過按鈕。

Visual Studio for Mac 安裝完成畫面

- 現在 Visual Studio for Mac 程式將會啟動起來，並且顯示在螢幕上。

Visual Studio for Mac 安裝完成畫面

2.2.3 在 Mac 上啟用遠端登入

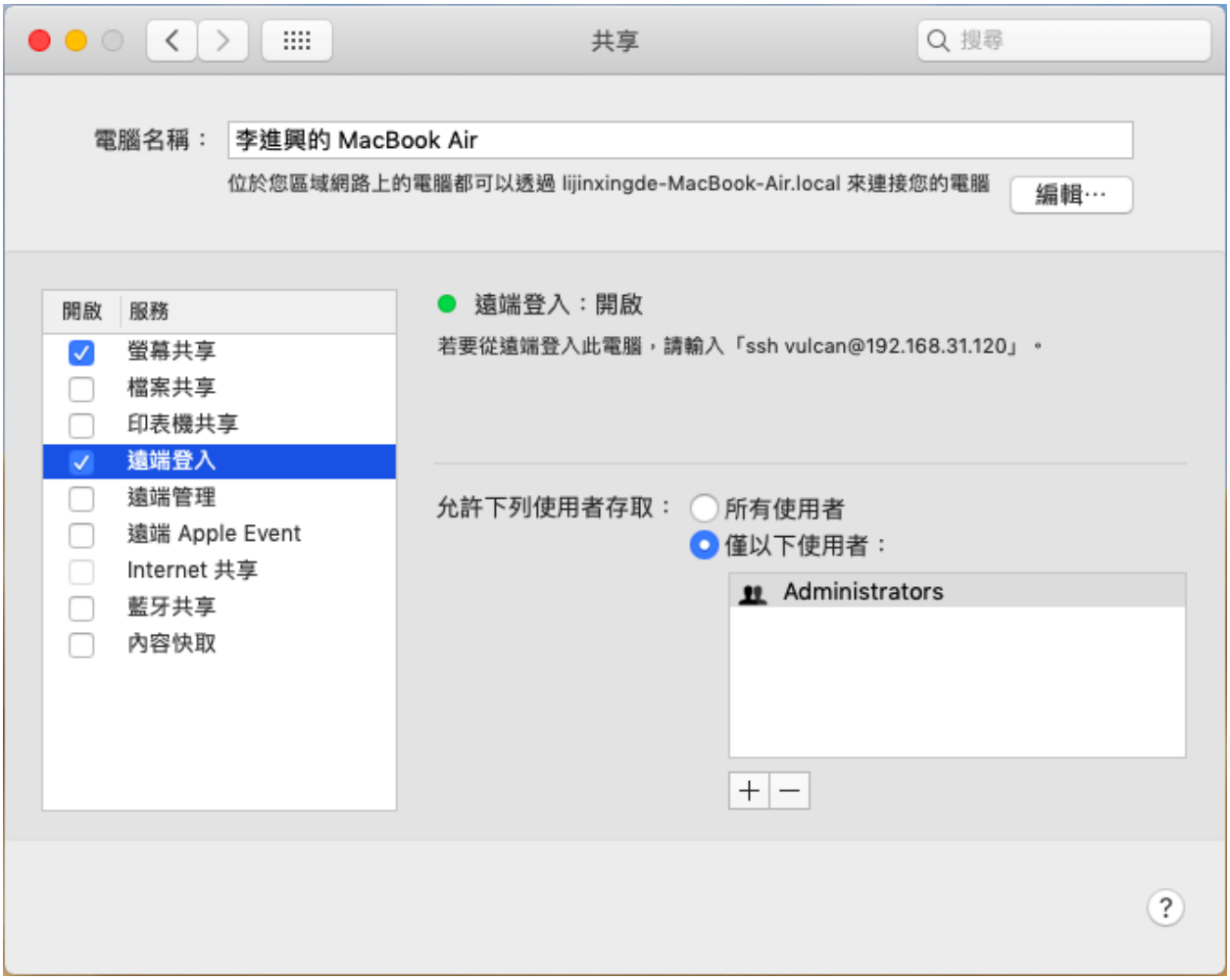
若想要在 Visual Studio 2019 for Windows 環境中，建立 Xamarin.iOS 的專案與進行除錯，就需要先在 Mac 電腦上啟用遠端登入，請在 Mac 開啟 [系統偏好設定]

系統偏好設定圖示

接著點選 [共享] 圖示，開啟 [共享] 視窗

Mac 電腦的共享設定圖示

在 [共享] 視窗內，最左邊的 [服務] 清單中找到的 [遠端登入] 項目，請勾選這個項目，另外，也需要確認右下方的 允許下列使用者存取中，要存取這台 Mac 電腦的帳號是否有被選取。



在共享對話窗中，啟用遠端登入

3. Visual Studio 2019 安裝後的相關設定與確認開發環境可以使用

當啟動完成 Visual Studio 2019 之後，需要開始進行 Android SDK 的設定與更新作業，確認相關機碼是否有正確產生出來。

在這個階段將會進行底下的相關設定

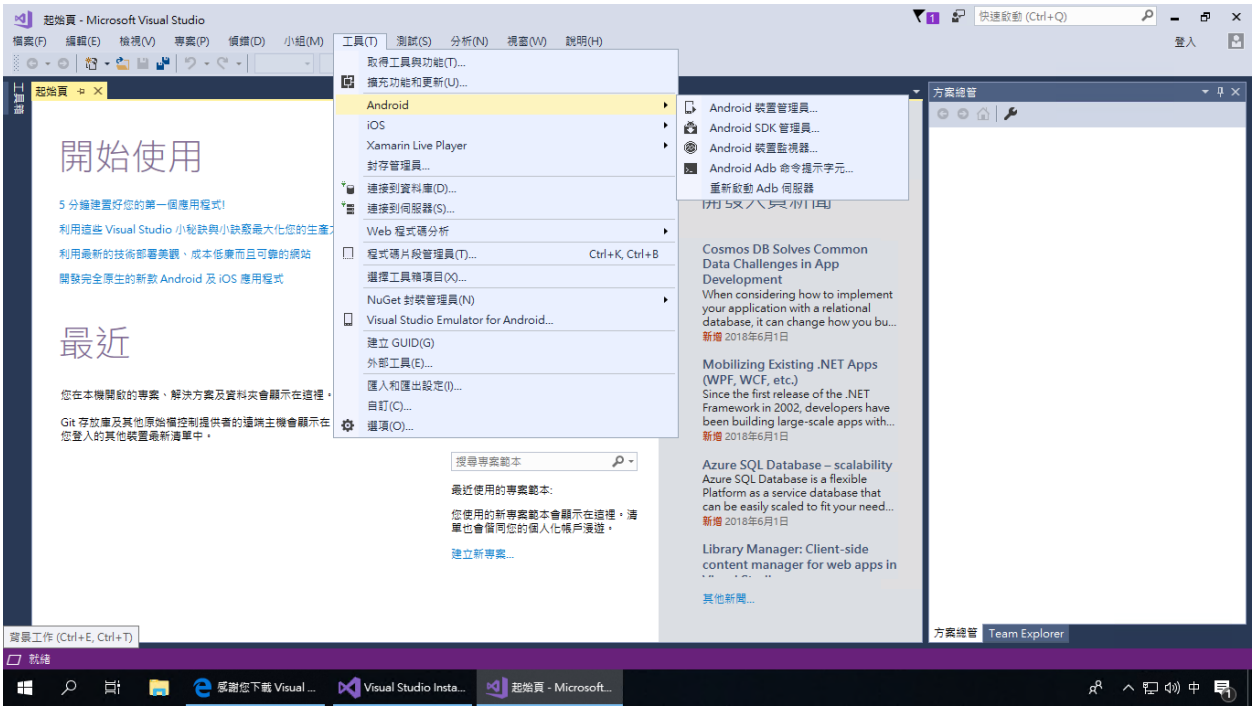
- 更新 Android SDK
- 安裝與啟動 Google Android 原生模擬器

3.1 更新 Android SDK

養成時常確保您的 Android SDK 有更新到最新版本的好習慣，需要進行 Android SDK 的設定與更新，確保 Xamarin.Android 專案可以正常順利的建置與執行 Android App。更多關於 Android SDK 的說明，可以參考 設定 Xamarin.Android 的 Android SDK[1]

- 開啟 Visual Studio 2019 程式
- 點選啟動後的視窗右下方的不使用程式碼繼續連結
- 請在 Visual Studio 程式中，點選功能表工具 > Android > Android SDK 管理員

[1] https://docs.microsoft.com/zh-tw/xamarin/android/get-started/installation/android-sdk?tabs=vswin

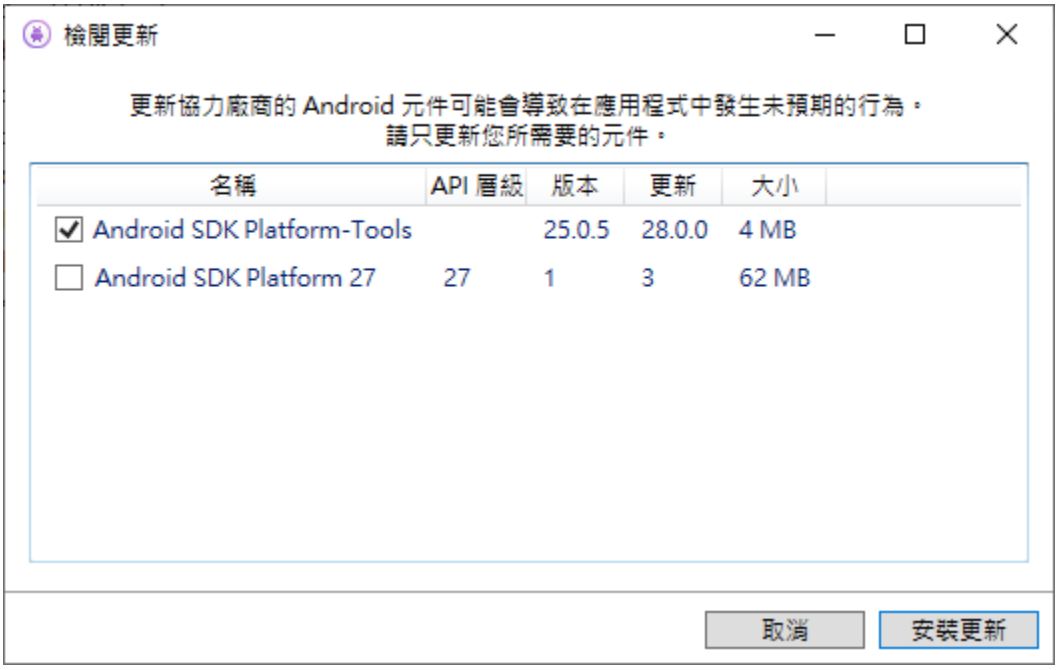


啟動 Android SDK 管理員

- 若有出現下方的使用者帳戶控制對話窗，請點選是按鈕

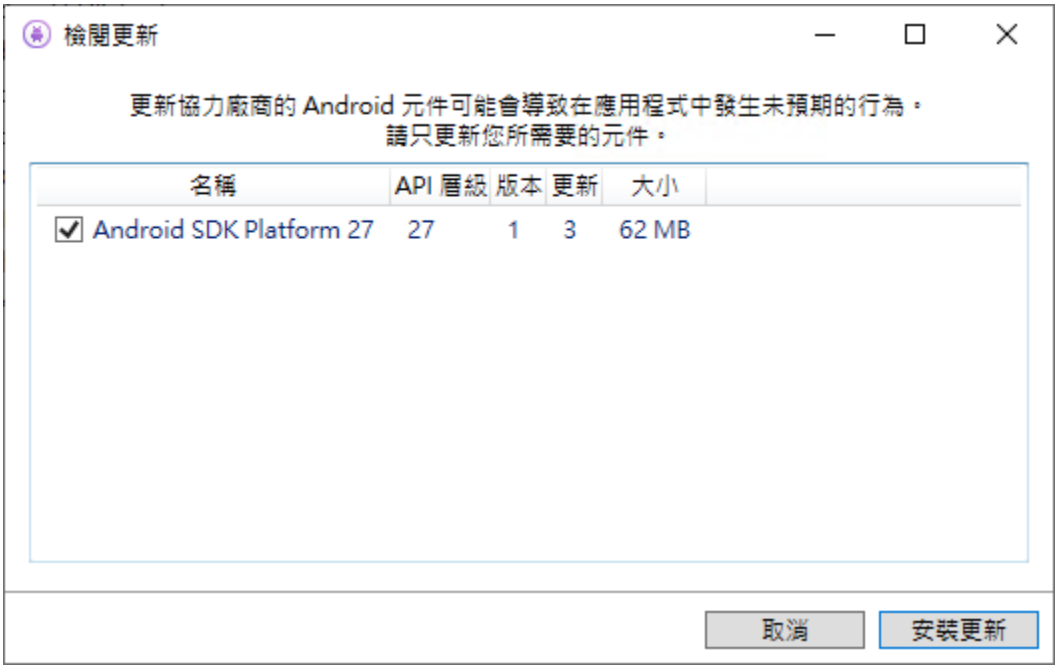
Visual Studio Emulator for Android

- 當 Android SDK 及工具視窗出現之後，請先點選工具標籤頁次，此時，您會發現到 Android SDK Platform-Tools 的版本是 25.0.5，並且在該項目的狀態欄位，該視窗的左下方，顯示的是有更新可用。
- 先點選左下角的 2 項可用的更新按鈕，當出現了檢閱更新對話窗的時候，只要選取 Android SDK Platform-Tools 這個項目要進行更新，接著，點選安裝更新按鈕。



檢閱更新對話窗

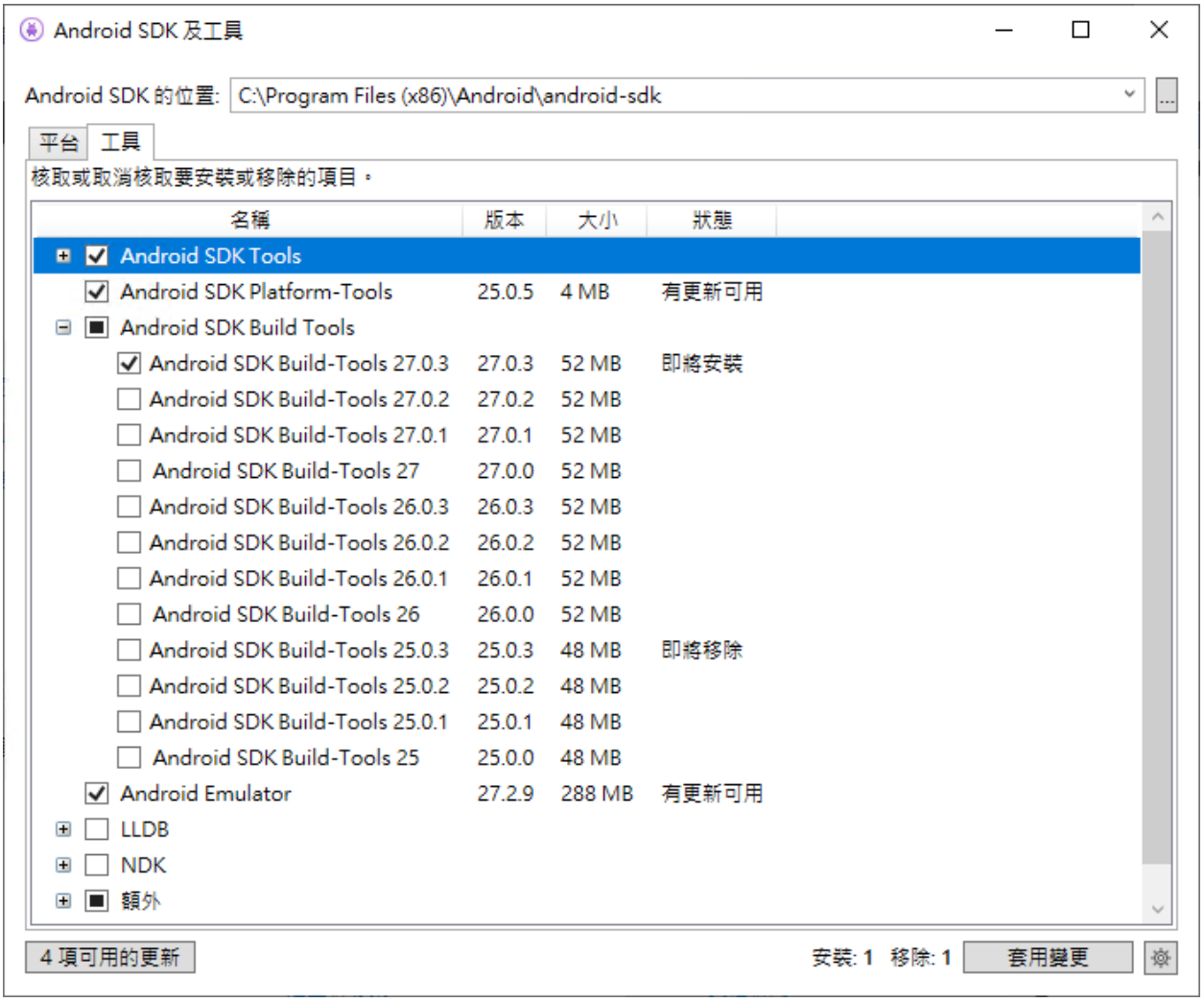
- 若成功安裝完成後，現在，只剩下 1 個可用更新了，接著點選左下角的 1 項可用的更新按鈕，當出現了檢閱更新對話窗的時候，選取 Android SDK Platform 27 這個項目要進行更新，接著，點選安裝更新按鈕。



檢閱更新對話窗

- 接下來，還是在工具標籤頁次，展開 Android SDK Build Tools 節點，取消勾選 Android SDK Build-Tools 25.0.3，並且勾選 Android SDK Build-Tools 27.0.3

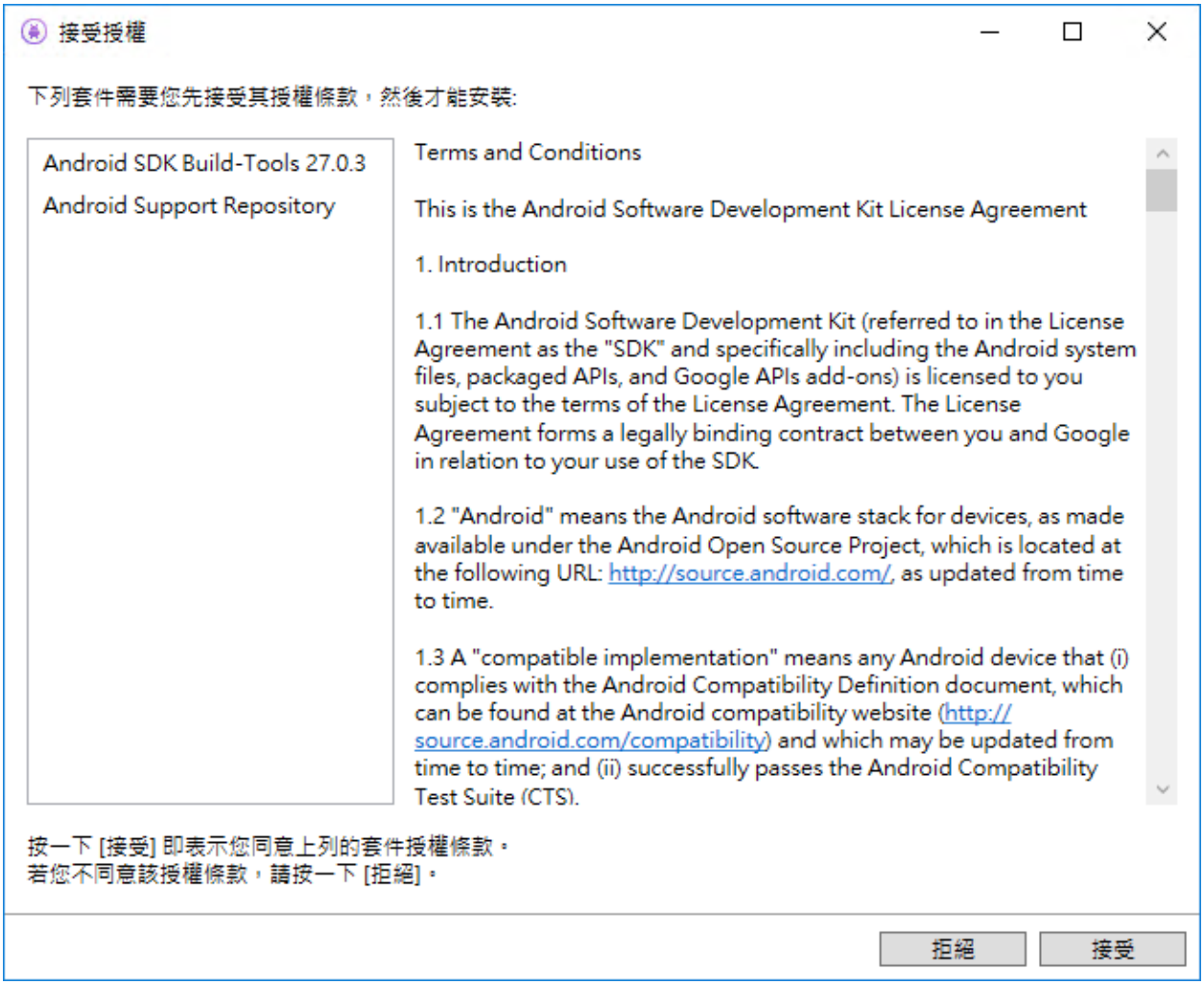
  展開額外節點，勾選 Android Support Repository 與 Google USB Driver 兩個選項，最後，點選右下方套用變更按鈕



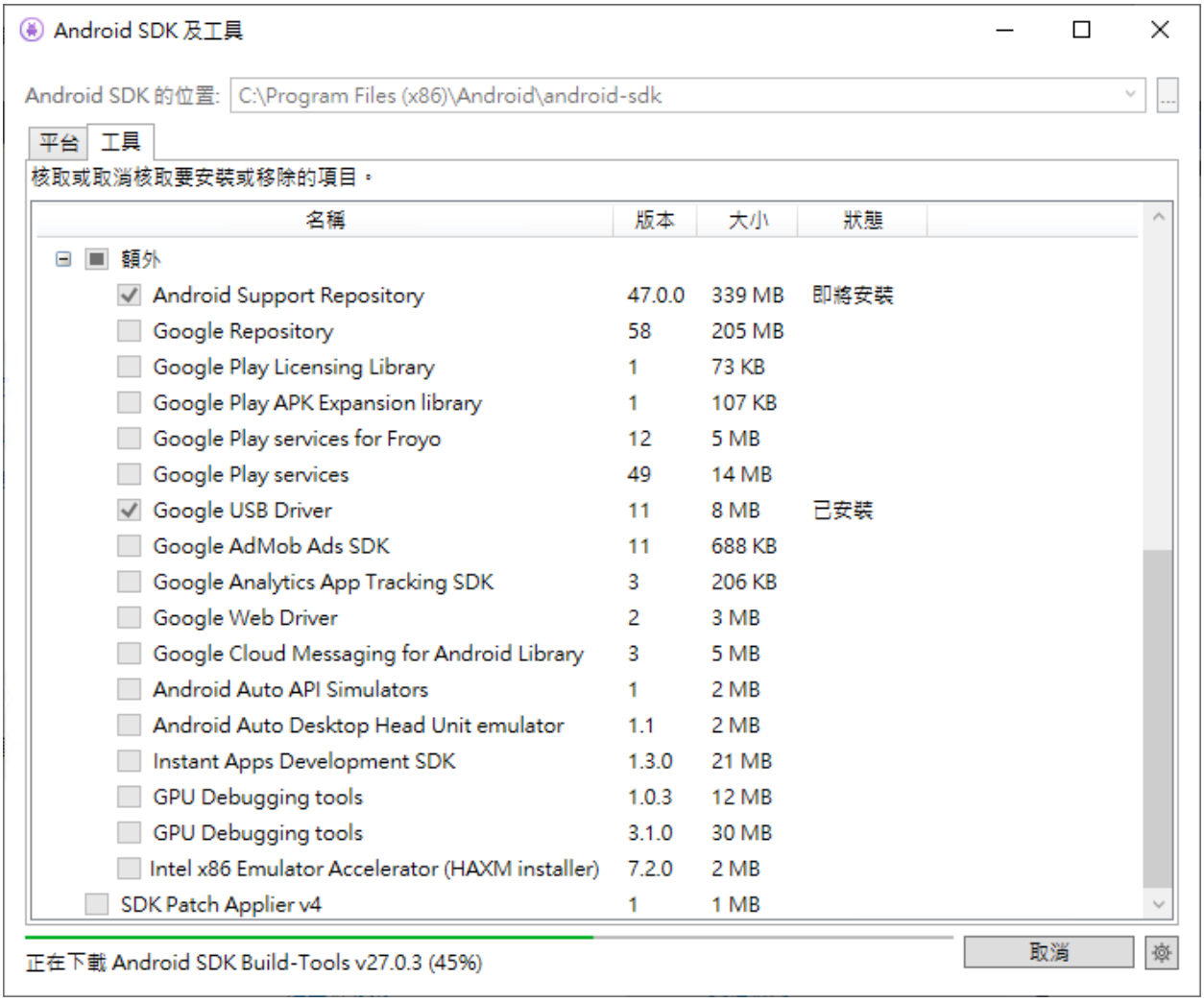
勾選 Android SDK Build-Tools 27.0.3

勾選 Android Support Repository

- 出現接受授權對話窗後，直接點選接受按鈕

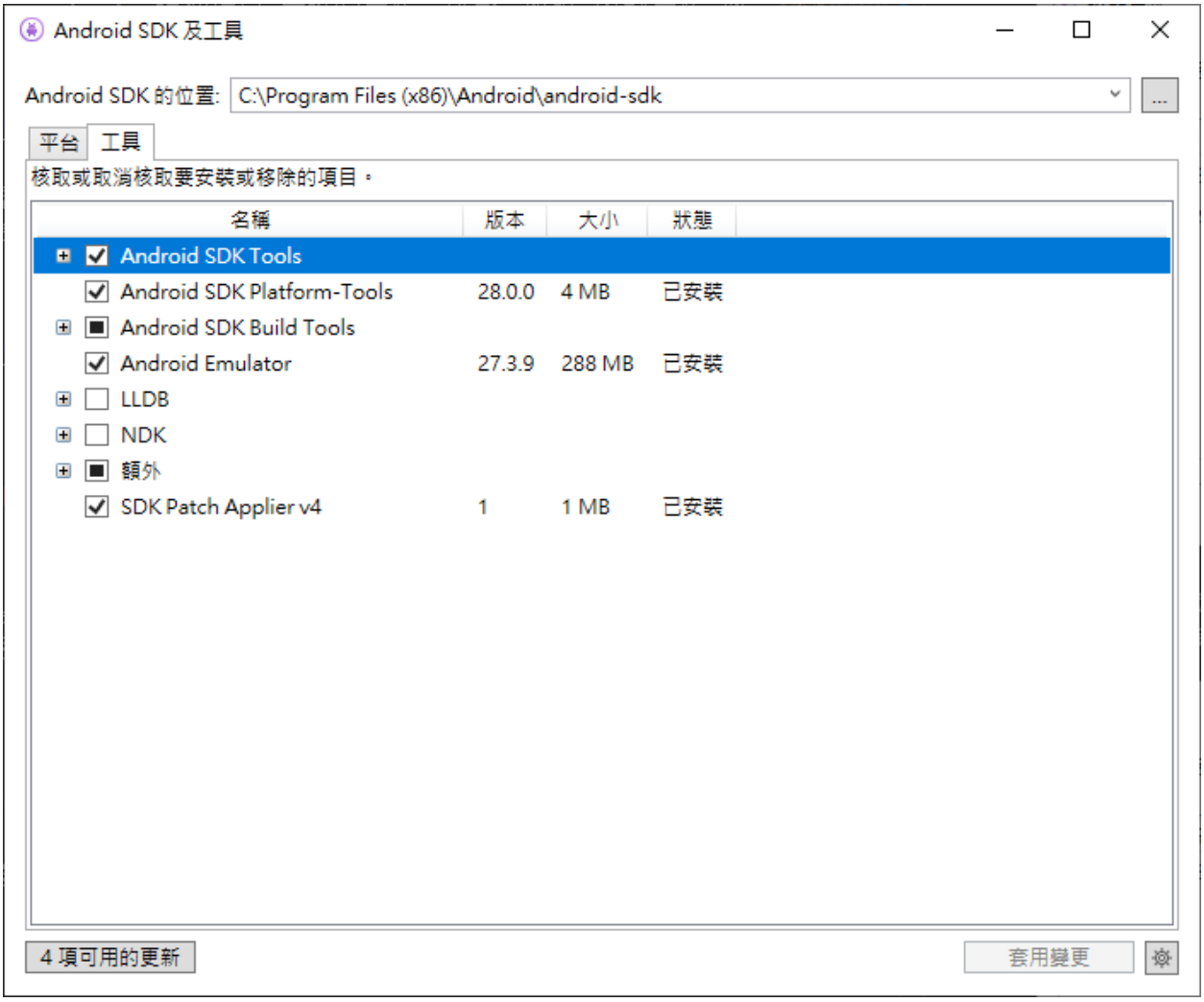

接受授權
下列套件需要您先接受其授權條款，然後才能安裝:
Android SDK Build-Tools 27.0.3
Android Support Repository
Terms and Conditions
This is the Android Software Development Kit License Agreement
1. Introduction
1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.
1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.
1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility Definition document, which can be found at the Android compatibility website (http://source.android.com/compatibility) and which may be updated from time to time; and (ii) successfully passes the Android Compatibility Test Suite (CTS).
按一下 [接受] 即表示您同意上列的套件授權條款。
若您不同意該授權條款，請按一下 [拒絕]。
拒絕
接受

接受授權對話窗



Android SDK 及工具

- 然後，在工具標籤頁次，勾選 Android Emulator，並且安裝起來。



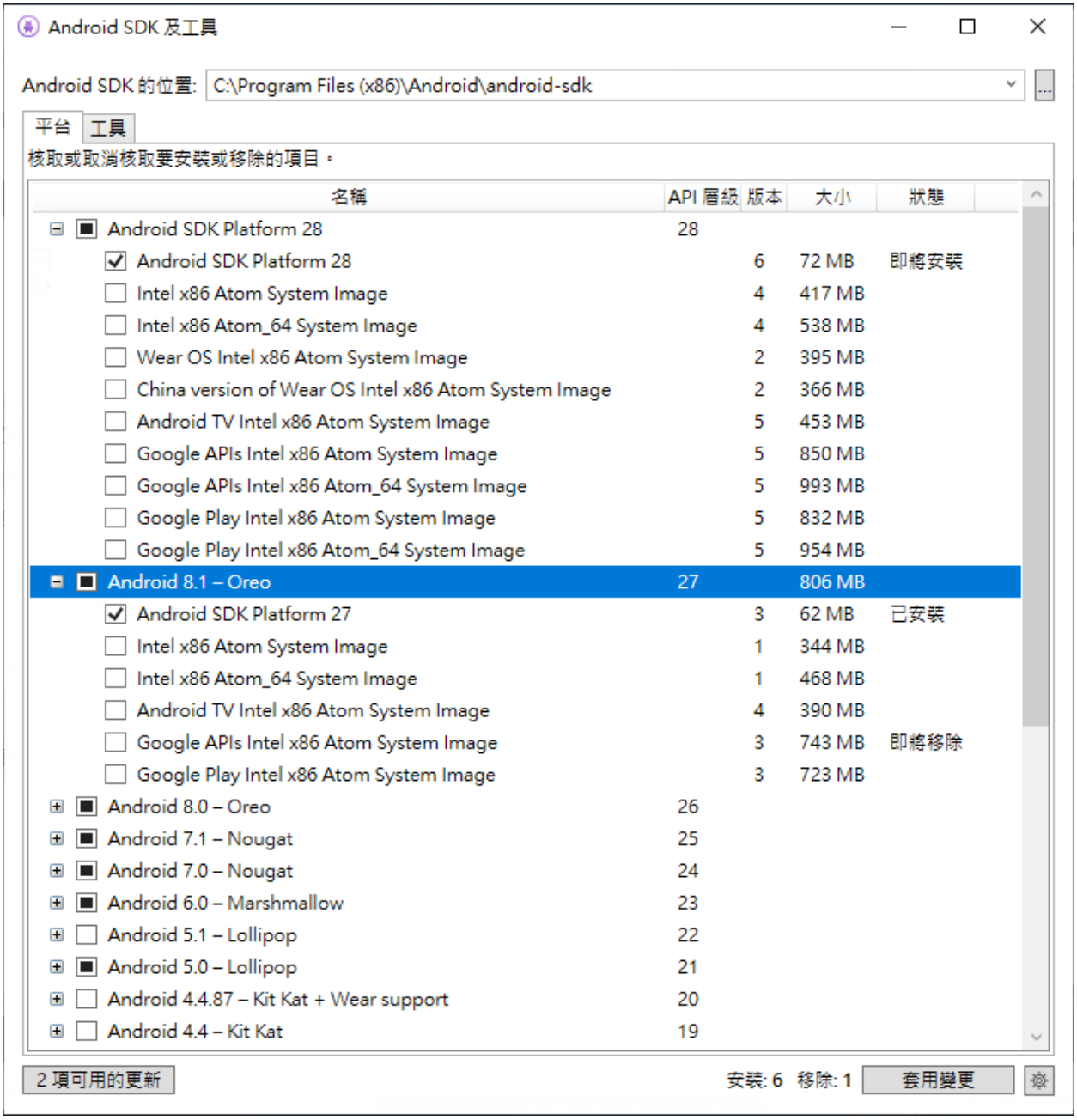
安裝 Android Emulator

- 現在要來設定平台標籤頁次，請點選標籤頁次

  在平台標籤頁次中，會有每個 Android SDK API 的版本項目，請依序展開您有興趣的 Android SDK API 版本，只需要勾選這個項目內的 Android SDK Platform X (X 為該 SDK API 版本編號)，其他的項目可以不用安裝。

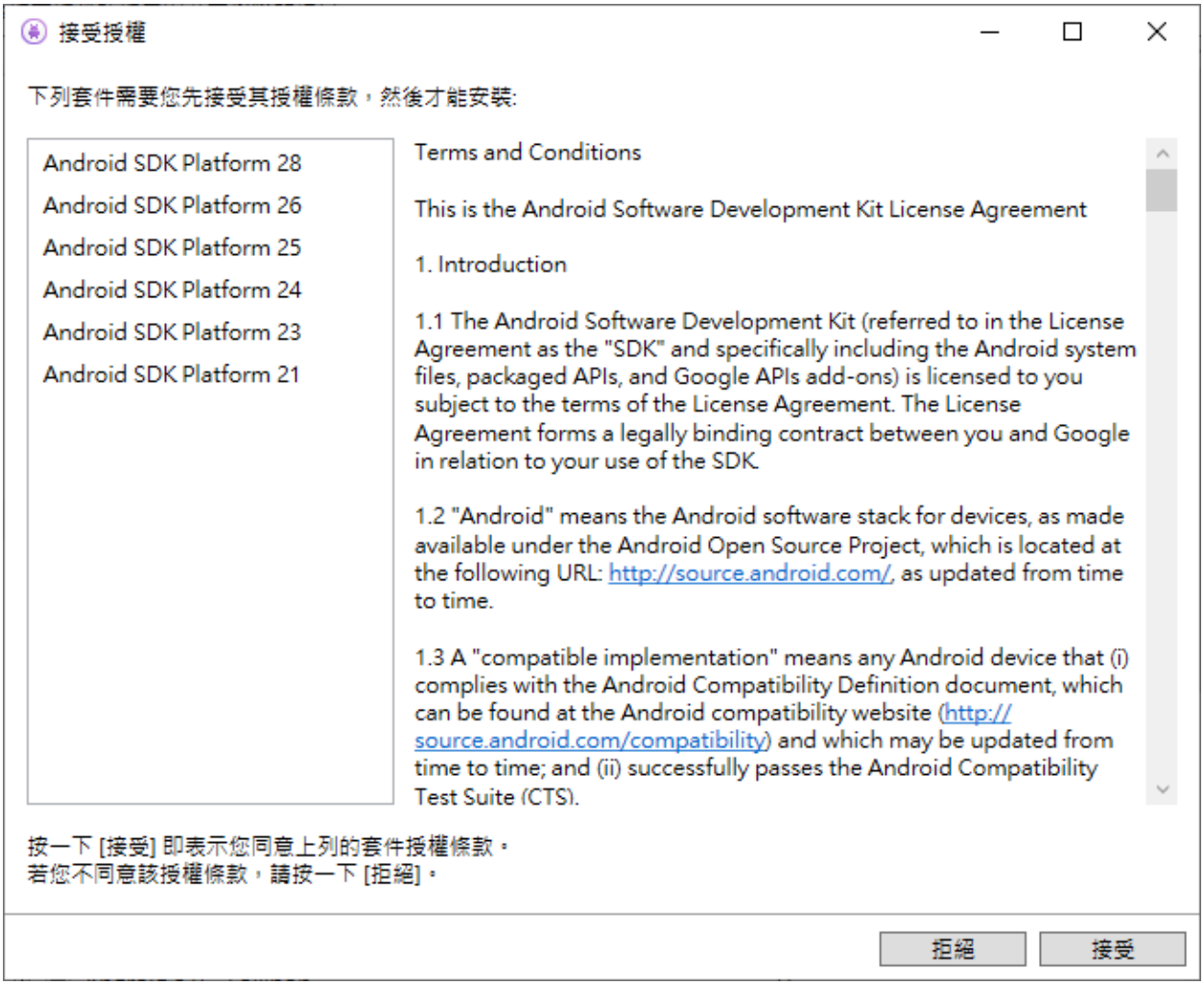
  在這個安裝過程中，將只會選擇 Android SDK Platform 28 (9.0) / 8.1 / 7.1 / 6.0 裡面的 Android SDK Platform X 項目。

  設定完成後，請點選右下角的套用變更按鈕



Android SDK 及工具的平台

- 當接受授權對話窗出現之後，請點選右下角的接受按鈕



接受授權對話窗

- 安裝完成之後，就可以把 Android SDK 及工具視窗關閉

3.2 安裝與啟動 Google Android 原生模擬器

在 Xamarin 開發工具下，可以選擇使用 Visual Studio 內建的 Visual Studio for Android Emulator 模擬器與 Google Android 原生模擬器這兩個模擬器的其中一個，當然，也可以選擇其他類型的 Android 模擬器，不過，對於無法在電腦上執行模擬器的情境下，是可以選擇使用實體裝置來進行開發和測試；不過，前面提到的兩者差異在於前者比較不耗

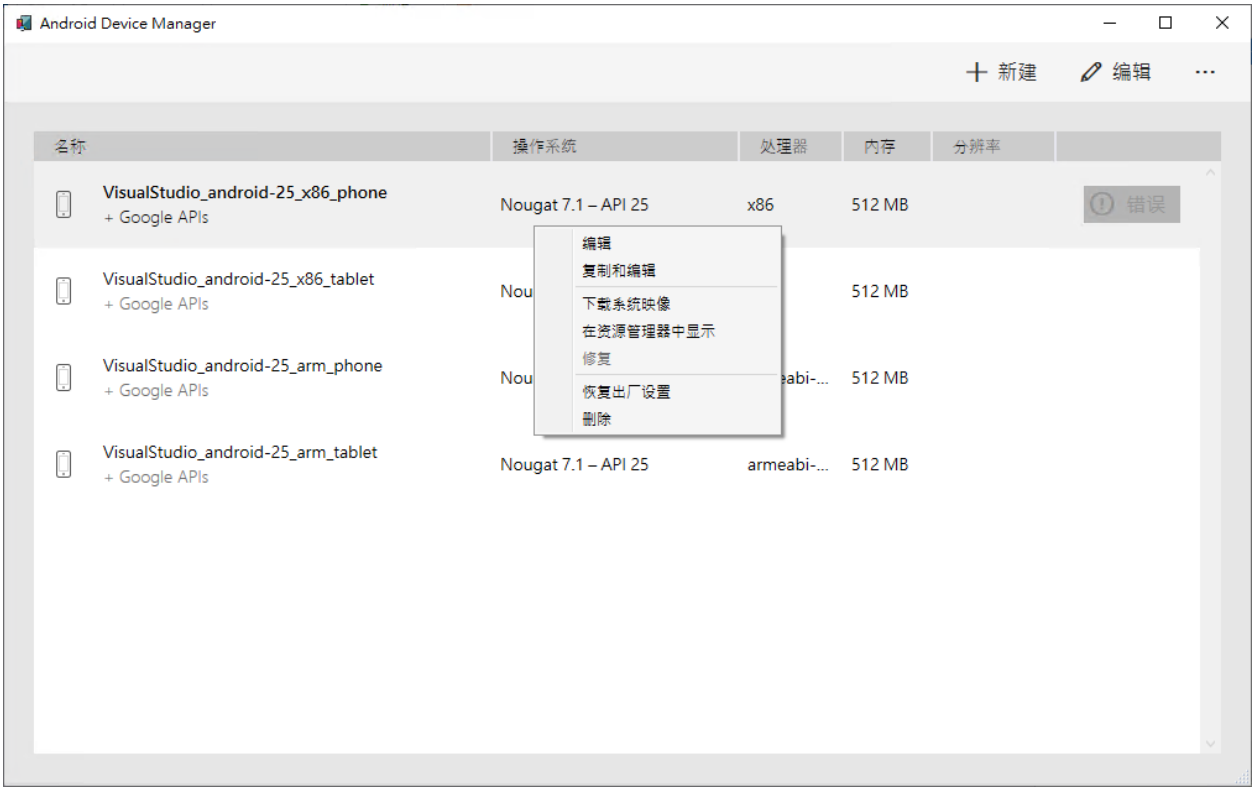
用 CPU 資源，但是，卻沒有預設安裝 Google Play Server，這對於有些開發應用上會比較麻煩，因為這些應用程式需要裝置上有運行 Google Play Server。

由於這裡將會使用 Google Android 原生模擬器與搭配 Intel HAXM 虛擬化技術，所以在安裝 Visual Studio 2019 的過程中，將會都安裝起來了，現在可以直接來使用與設定這個模擬器。

- 開啟 Visual Studio 2019 程式
- 點選啟動後的視窗右下方的不使用程式碼繼續連結
- 請在 Visual Studio 程式中，點選功能表工具 > Android > Android Device Manager
- 接著會出現使用者帳戶控制對話窗，請點選是按鈕

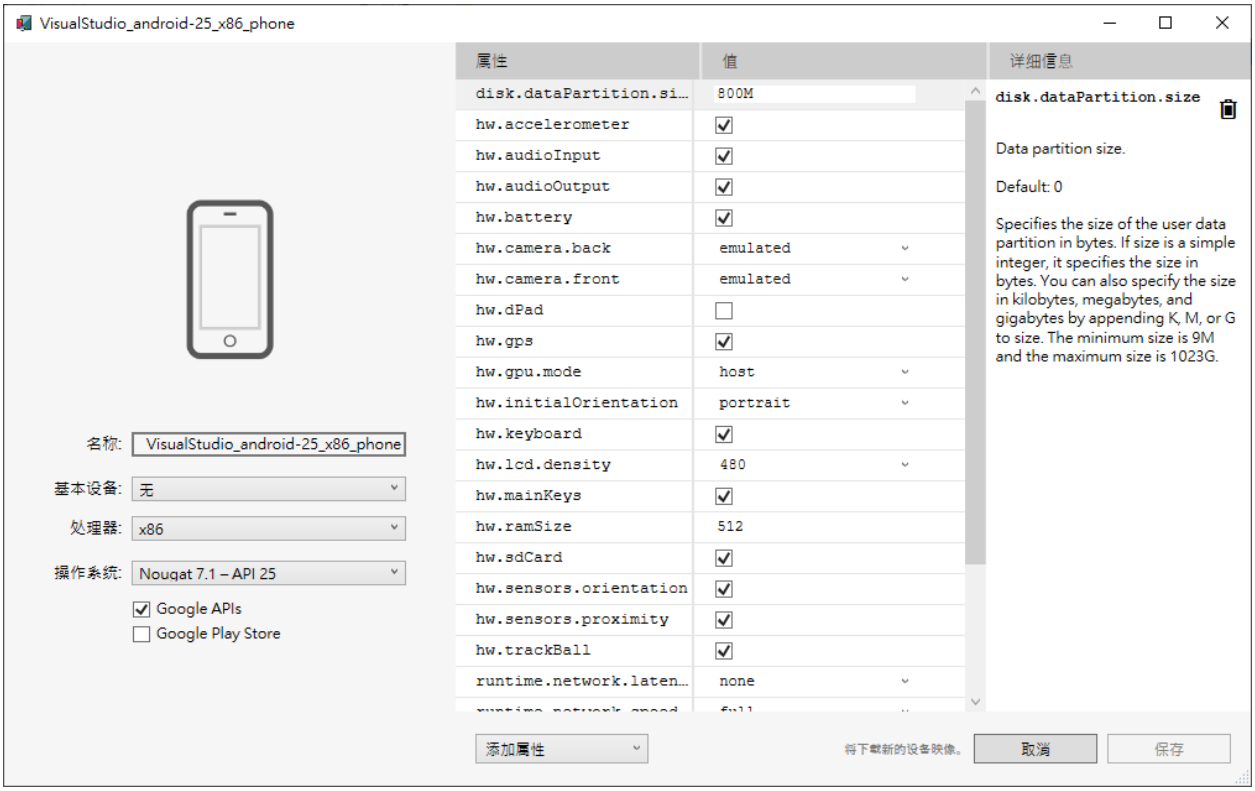
使用者帳戶控制對話窗

- 當 Android Device Manager 視窗出現後，請在第一個項目 VisualStudio_android-25_x86_phone ，使用滑鼠右擊這個項目，接著選擇編輯選項



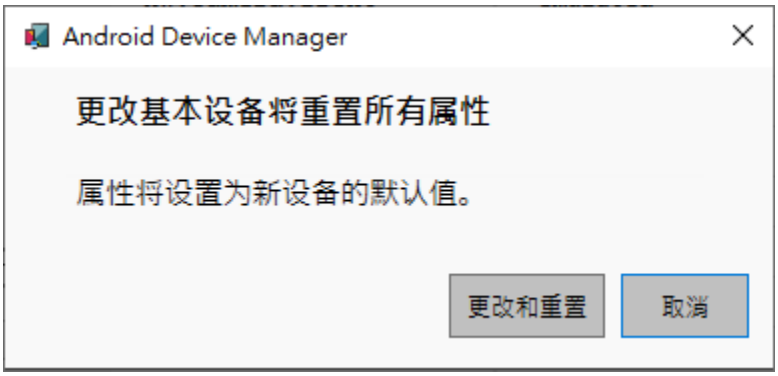
修改 Android Device Manager 的模擬器項目

- 此時會出現一個新的對話視窗，顯示這個模擬器之相關設定內容



VisualStudio_android-25_x86_phone 對話窗

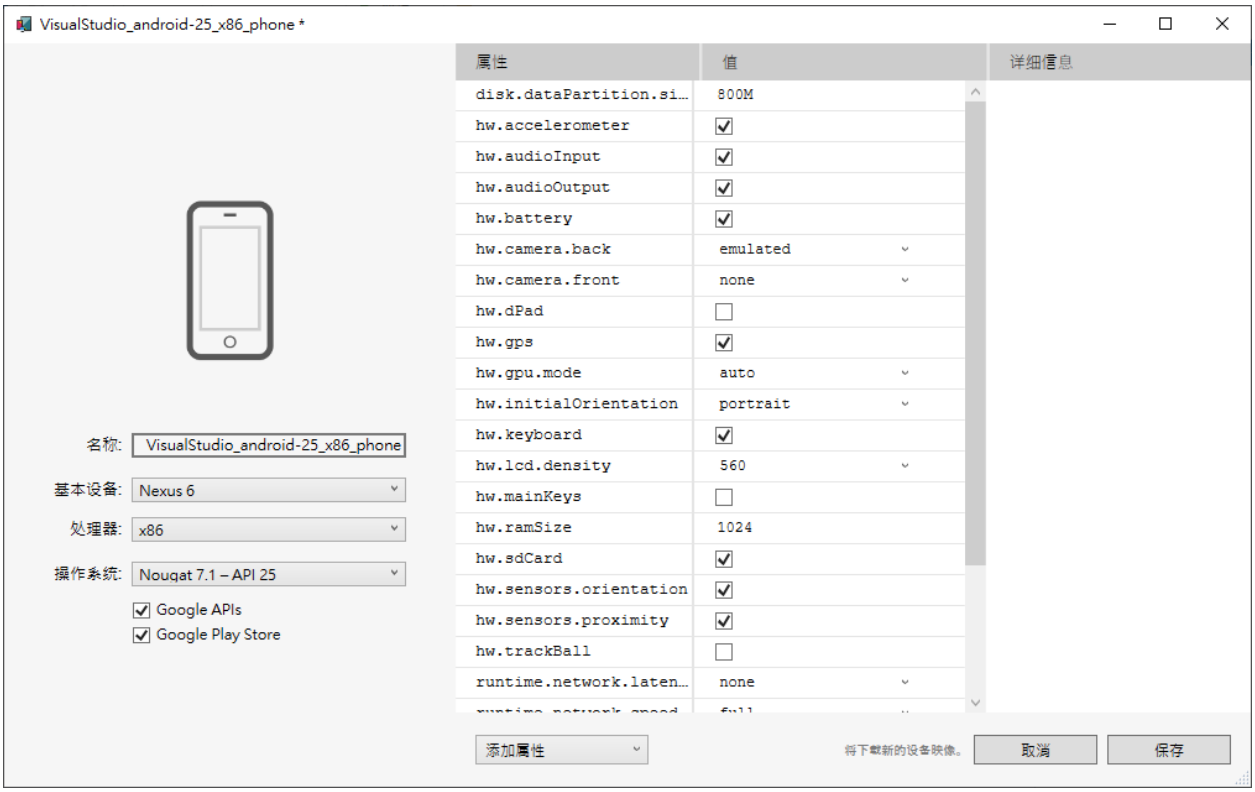
- 請點選左下方的基本設備下拉選單，選擇 Nexus 6 項目，現在將會看到一個警告對話窗，請點選更改和重置按鈕



更改基本設備將重置所有屬性

- 在 VisualStudio_android-25_x86_phone 對話窗的左下方，勾選 Google Play Store

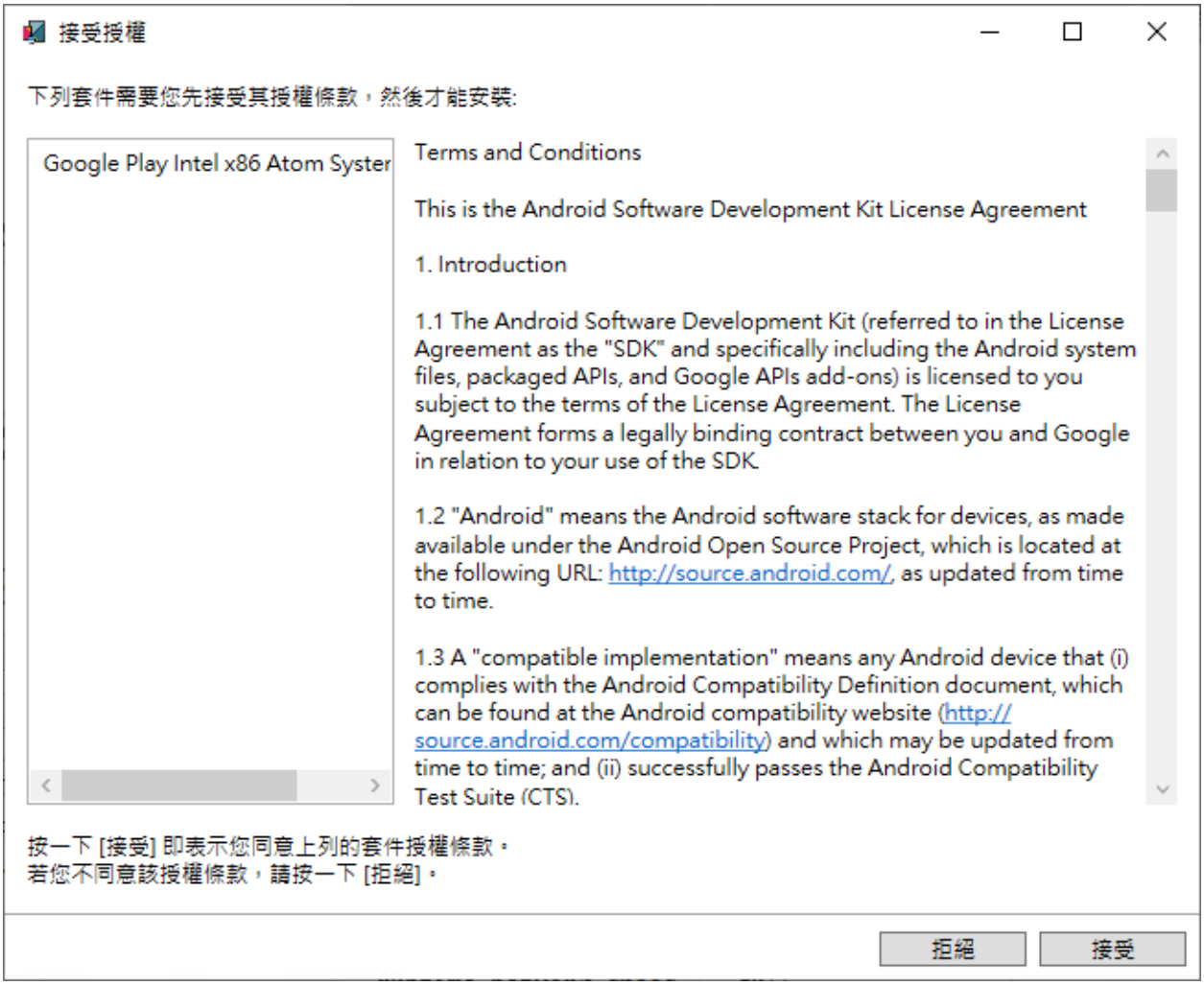
選項，最後點選該對話視窗右下方的保存按鈕。



VisualStudio_android-25_x86_phone 對話窗

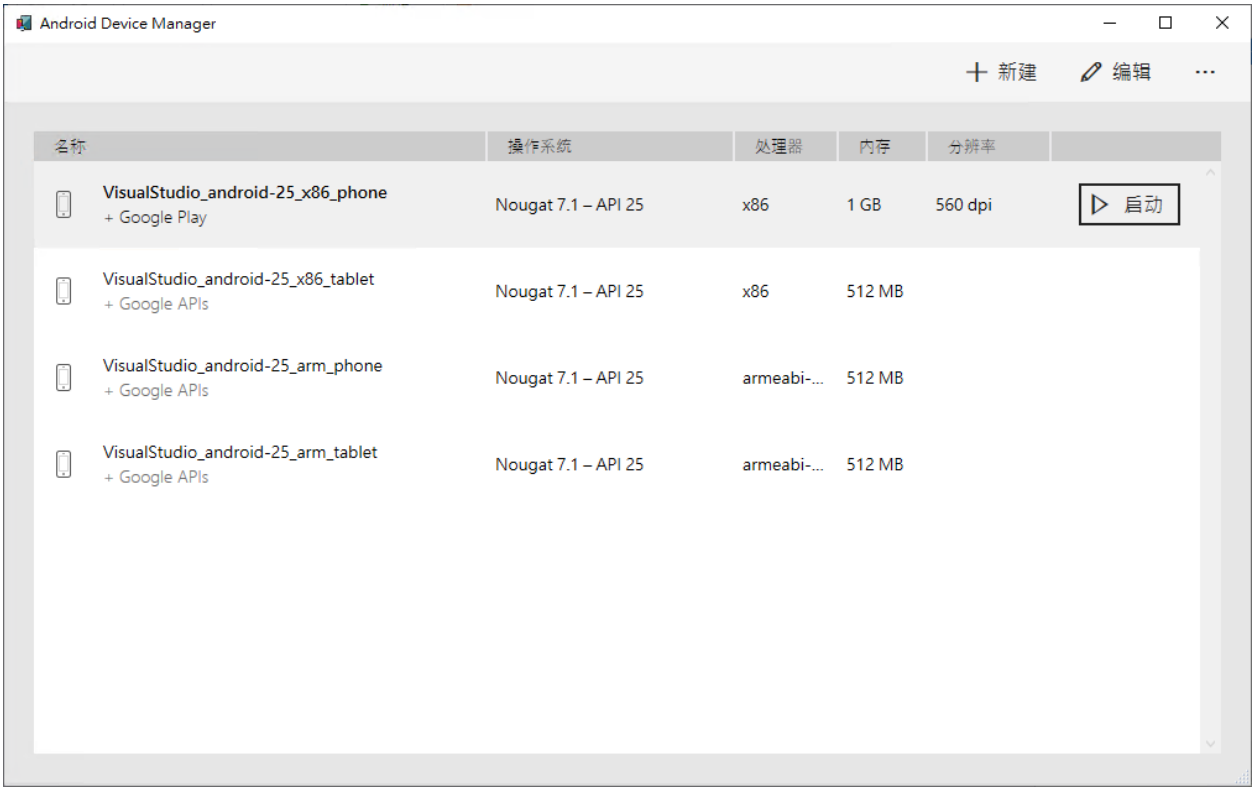
- 當出現接受授權對話窗，請點選接受按鈕。

  若沒有出現這個對話窗，可以跳過，繼續下一個設定動作



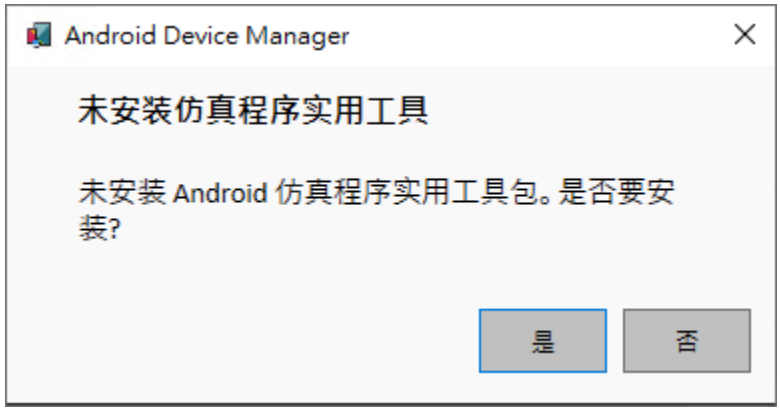
接受授權對話窗

- 當模擬器下載完成之後，請點選啟動按鈕。



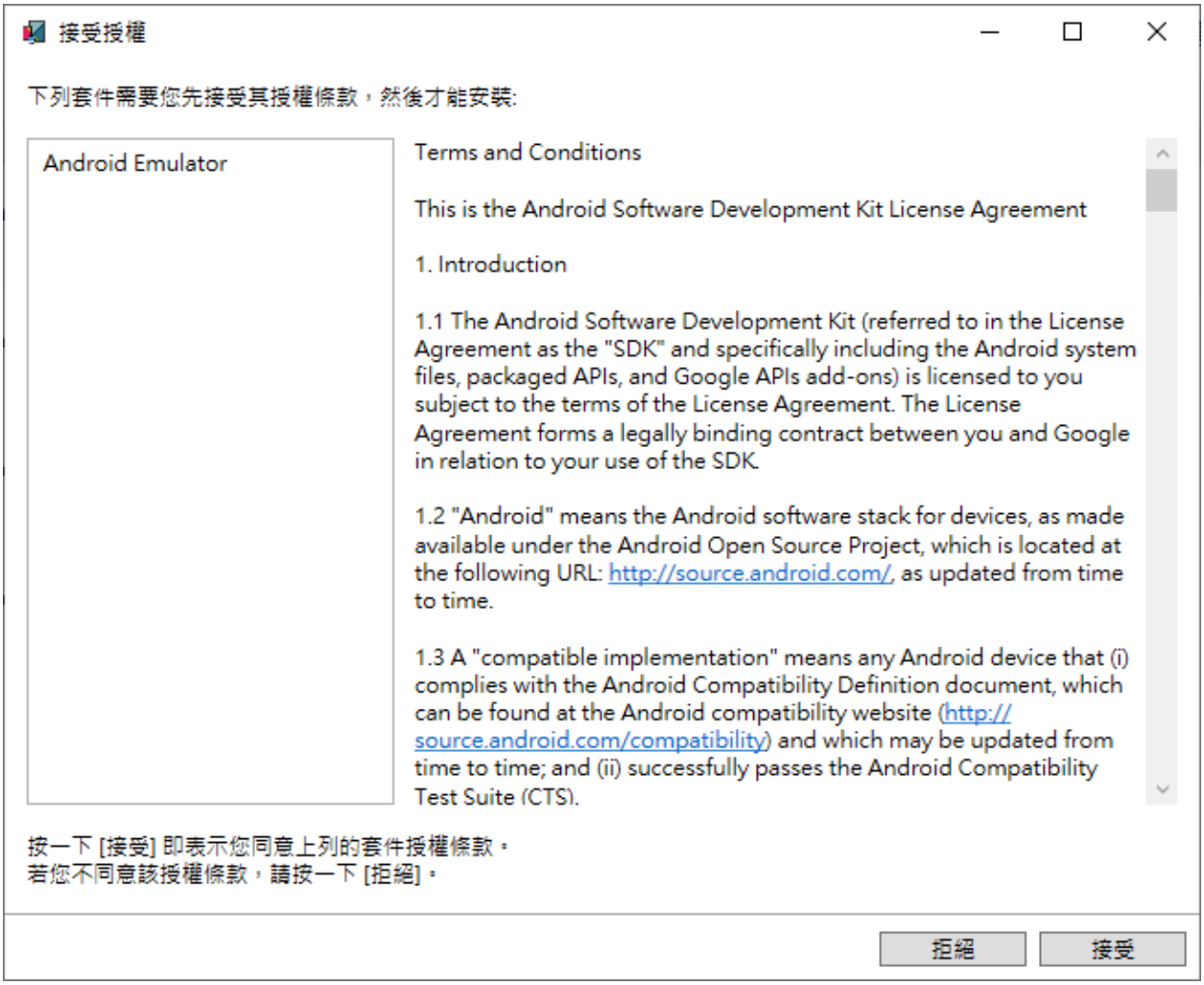
模擬器下載完成

- 接著會出現未安裝模擬器程式實用工具對話窗，請點選是按鈕。



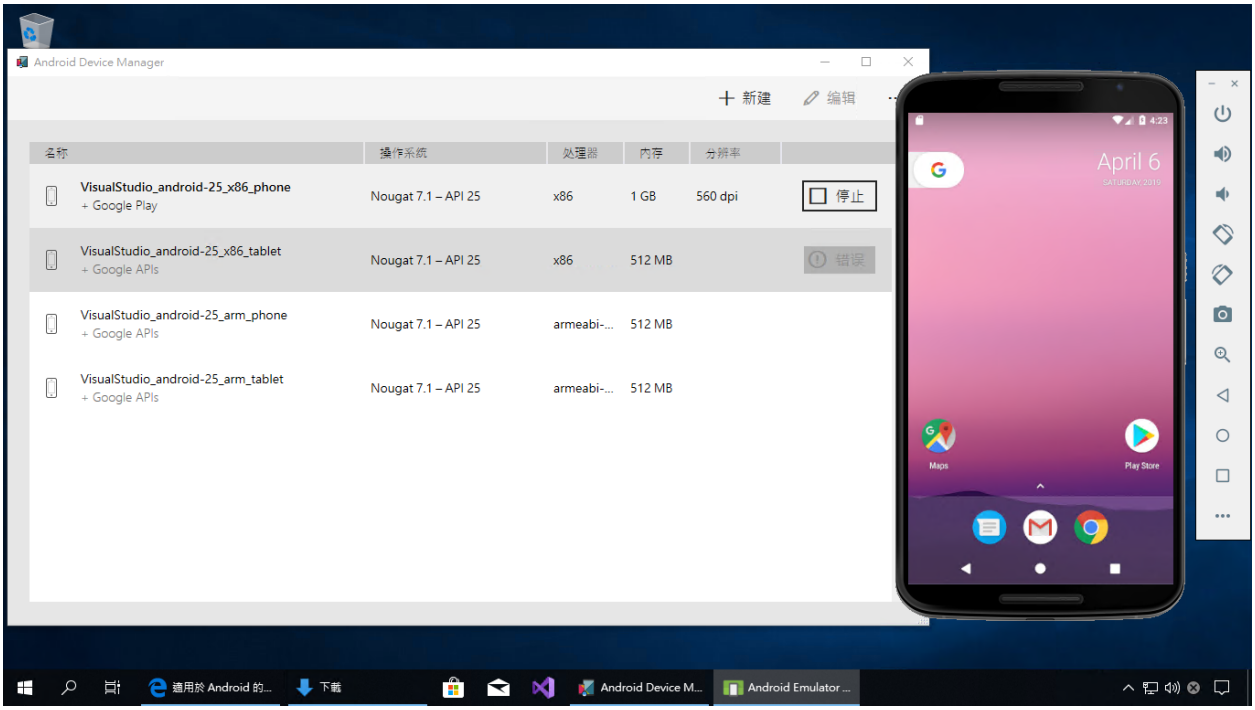
未安裝模擬器程式實用工具對話窗

- 當接受授權對話窗出現之後，請點選右下角的接受按鈕



接受授權對話窗

- 當 Android Emulator 軟體安裝完成之後，請重新點選啟動按鈕
- 現在可以開始使用 Google Android 所提供的模擬器，該模擬器中已經預設安裝了 Google Paly 服務應用程式



Google Android 所提供的模擬器成功啟動

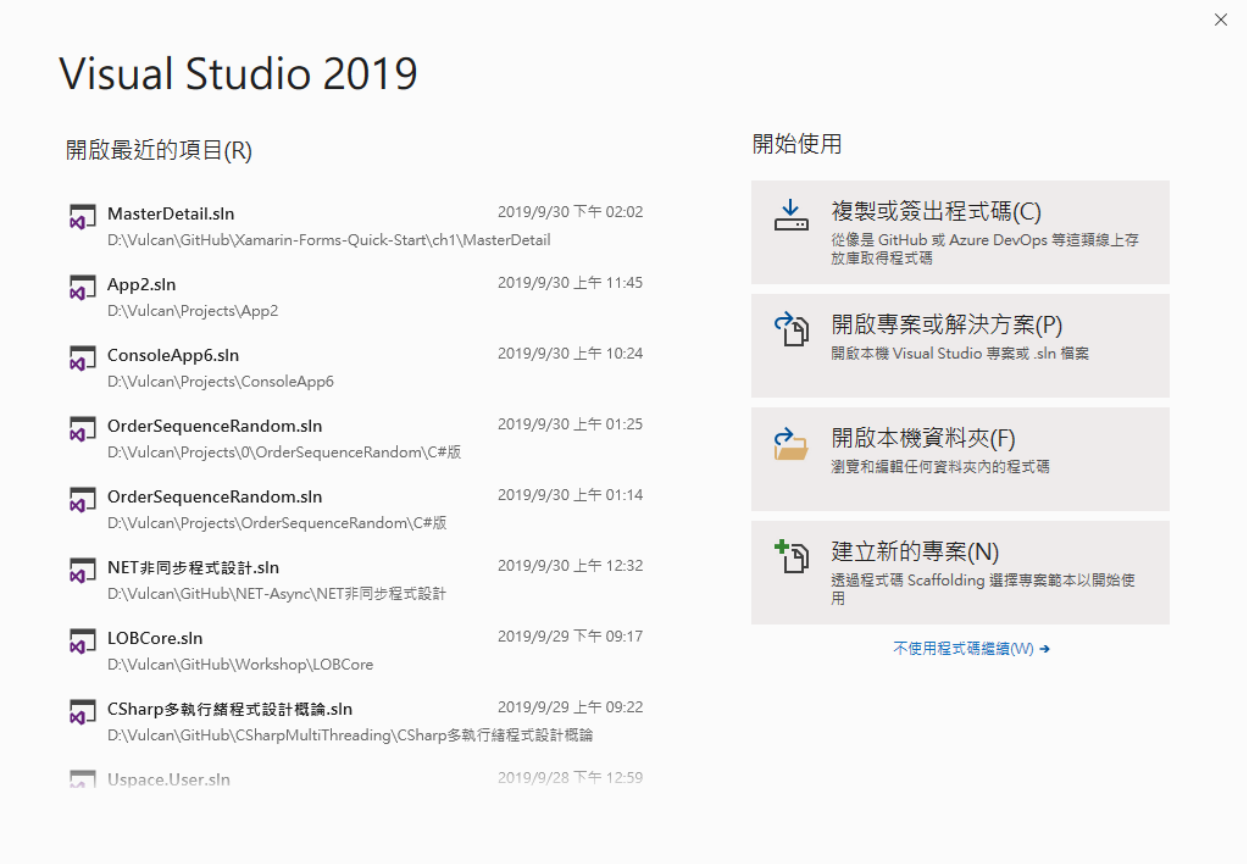
3.3 測試與確認開發環境可以進行 Xamarin.Forms 專案開發

您必須要確實進行這些檢查與測試工作，確保您的 Xamarin.Forms 開發環境可以正常運作。

- 測試可以建立 Xamarin.Forms 專案
- 建置與執行 Android 專案
- 建置與執行 iOS 專案
- 建置與執行 UWP 專案

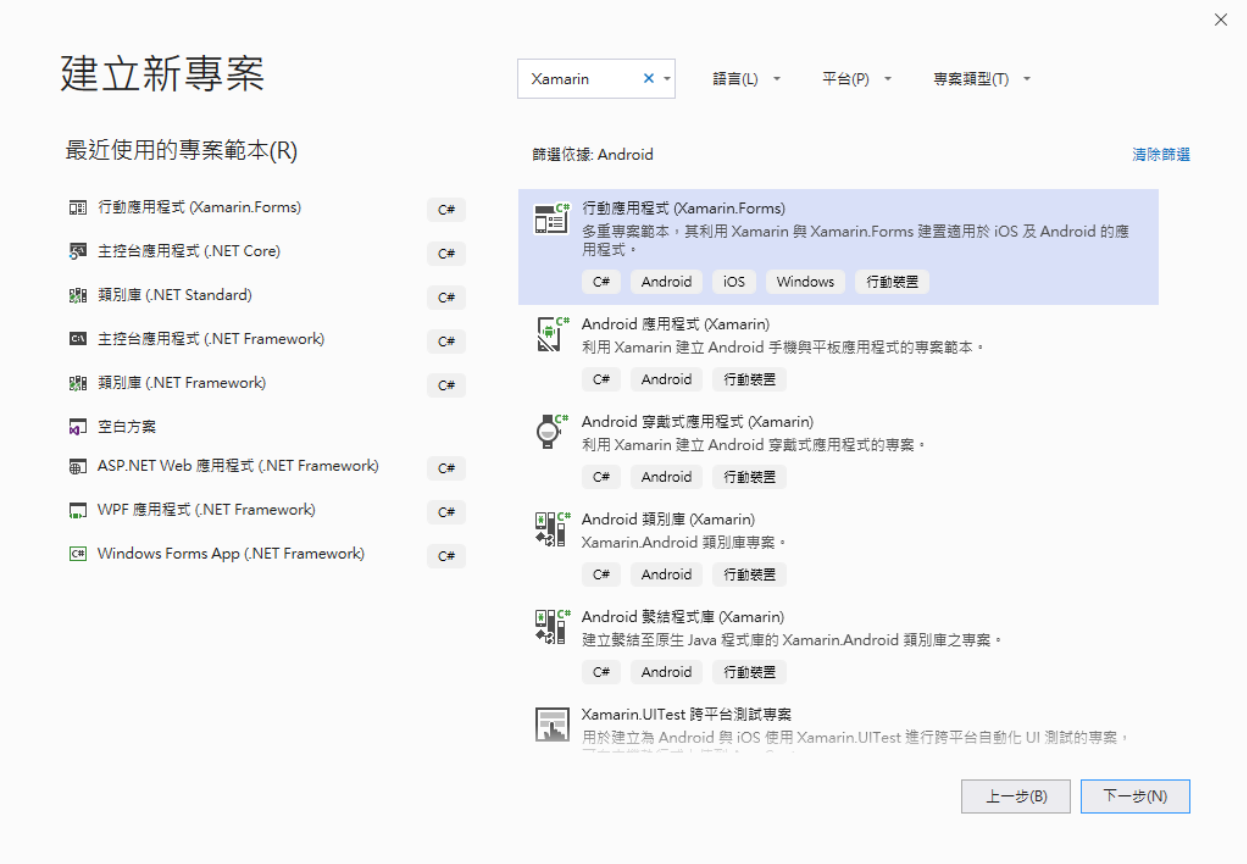
3.3.1 測試可以建立 Xamarin.Forms 專案

- 請在開啟 Visual Studio 2019 程式



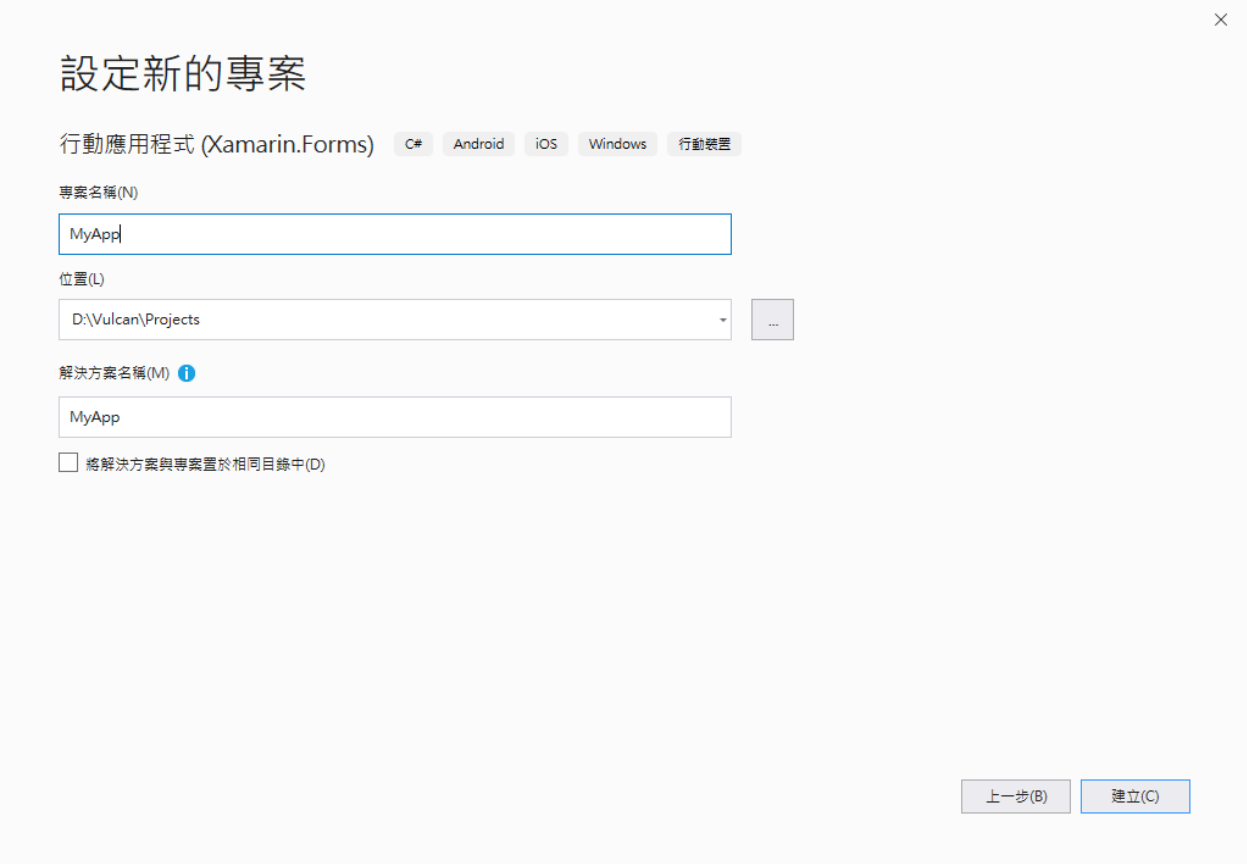
Visual Studio 2019 啟動對話窗

- 點選右下方的建立新的專案按鈕
- 當建立新專案對話窗出現的時候，在中間上方的搜尋文字輸入盒內，請輸入 Xamarin 這個關鍵字，如此，就會在右方顯示出與 Xamarin 相關的各種專案樣板可供選擇。



Visual Studio 2019 建立新專案對話窗

- 請選擇行動應用程式 (Xamarin.Forms) 這個專案樣版後，點選右下方的下一步按鈕
- 此時，將會顯示出設定新的專案對話窗，請在專案名稱欄位中，輸入這個跨平台專案的名稱，在此是輸入 MyApp

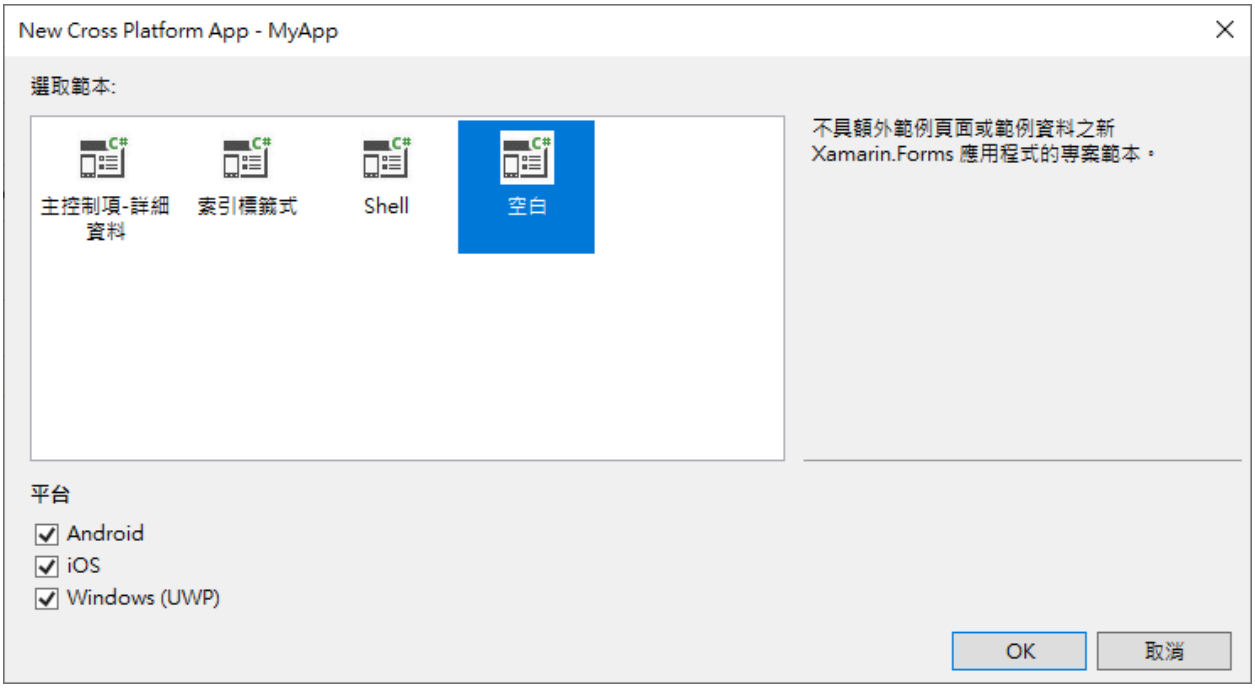


Visual Studio 2019 設定新的專案對話窗

- 請點選右下方建立按鈕，就可以建立此一 Xamarin.Forms 的專案
- 現在將會顯示出 New Cross Platform App - MyApp 這個對話窗，讓開發者選擇一個合適的專案樣板來開始進行專案開發

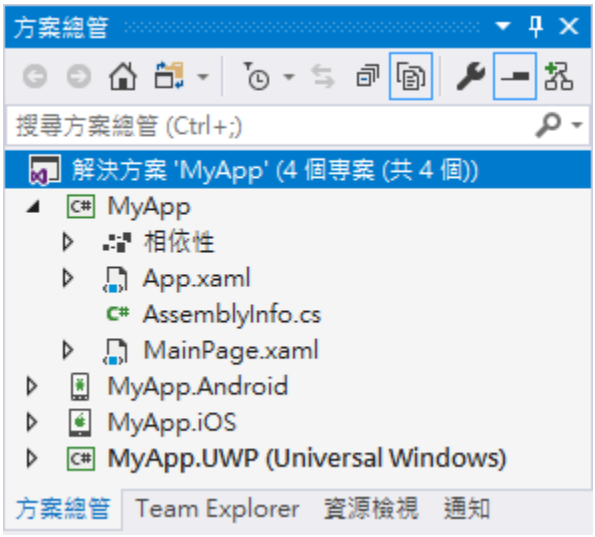
  在此，請在選取範本區域，點選空白這個選項

  接著，在下方平台的區域，勾選三個平台，也就是 Android / iOS / Windows (UWP)



Visual Studio 2019 New Cross Platform App - MyApp 對話窗

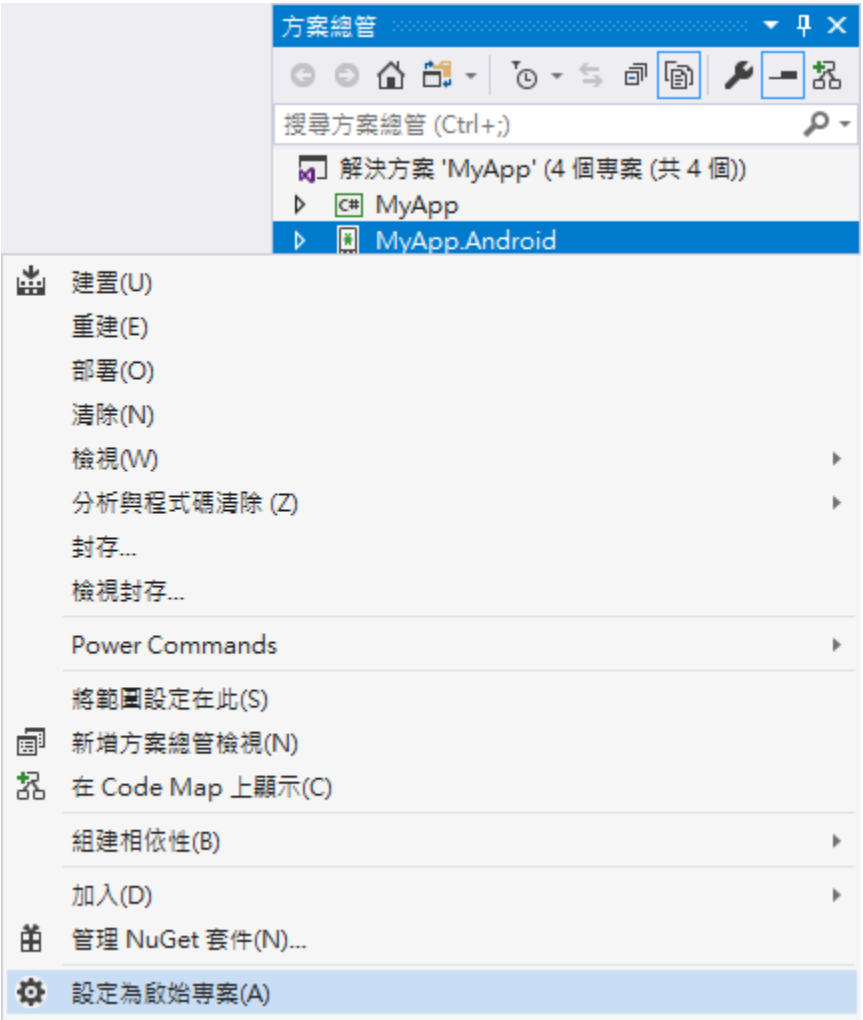
- Visaul Studio 2019 現在將會開始建立這個 Xamarin.Forms 的方案，完成後，該方案內將會有四個專案，分別是：Xamarin.Forms 核心專案、Xamarin.Android 原生專案、Xamarin.iOS 原生專案與 Windows UWP 原生專案。
- 從 Visual Studio 2019 方案總管中，可以看到總共有四個專案產生出來



Xamarin.Forms 專案

3.3.2 建置與執行 Android 專案

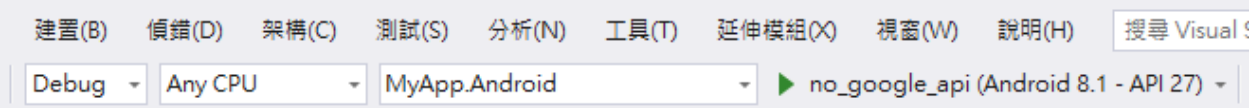
- 當 Xamarin.Forms 專案建立完成後，請使用滑鼠右擊 Android 專案選擇設定為起始專案



在方案總管中設定預設起始專案

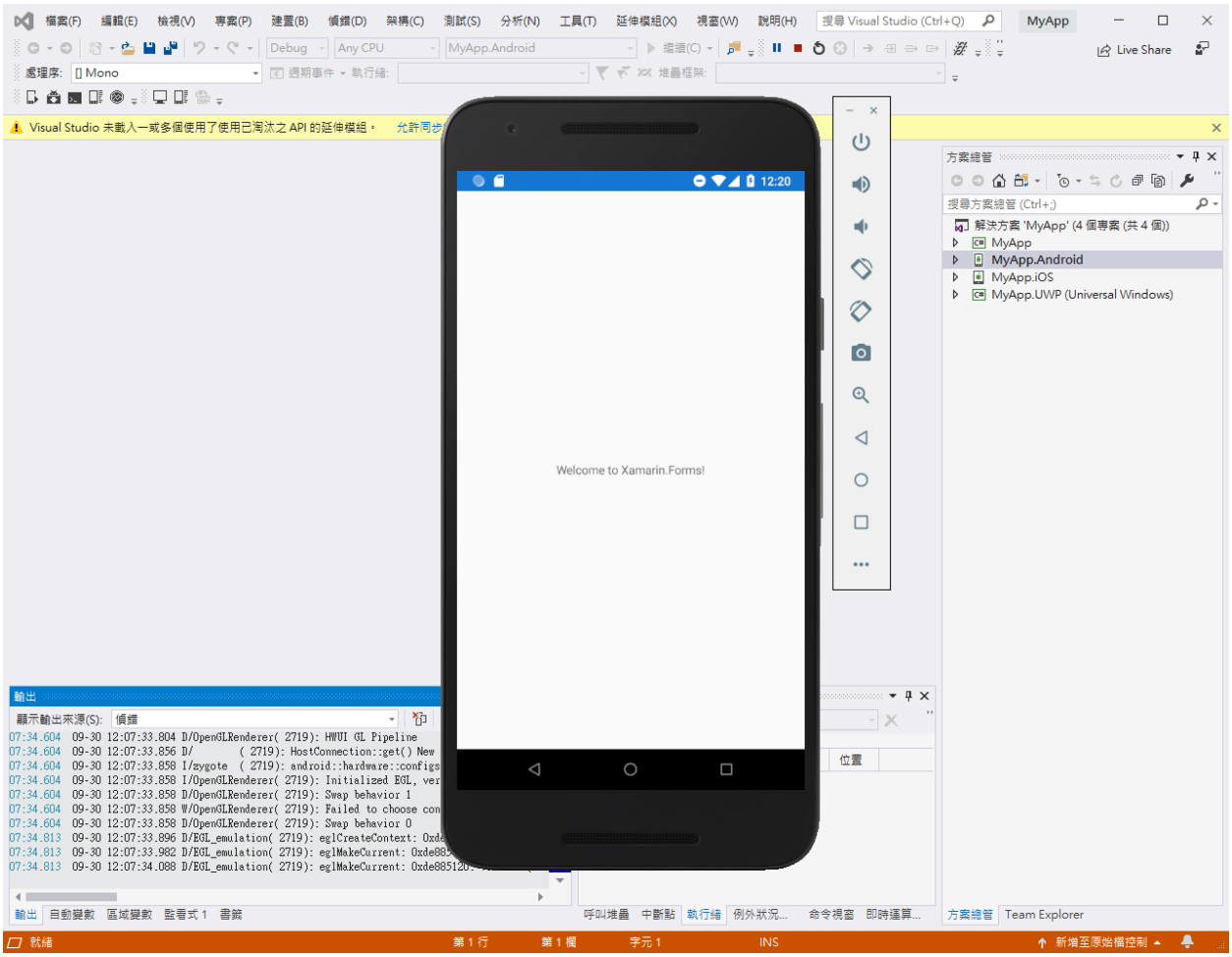
- 此時，您會在 Visual Studio 2019 中間上方區域，看到啟動專案已經設定為 Android 專案，並且也會看到可以執行的 Android 模擬器選項清單項目，選擇一個想要執行的模擬器項目。

  請點選有綠色三角形的模擬器項目，開始建置與在這個模擬器上來執行



執行 Android 專案

- 由於第一次進行專案建置，需要下載許多 NuGet 套件，因此，需要花費一些時間，請耐心等候一下，底下是成功執行完成的畫面



Xamarin.Forms 的 Android 專案成功執行結果

- 停止這個 Android 專案的執行 (點選工具列上的紅色正方形按鈕)

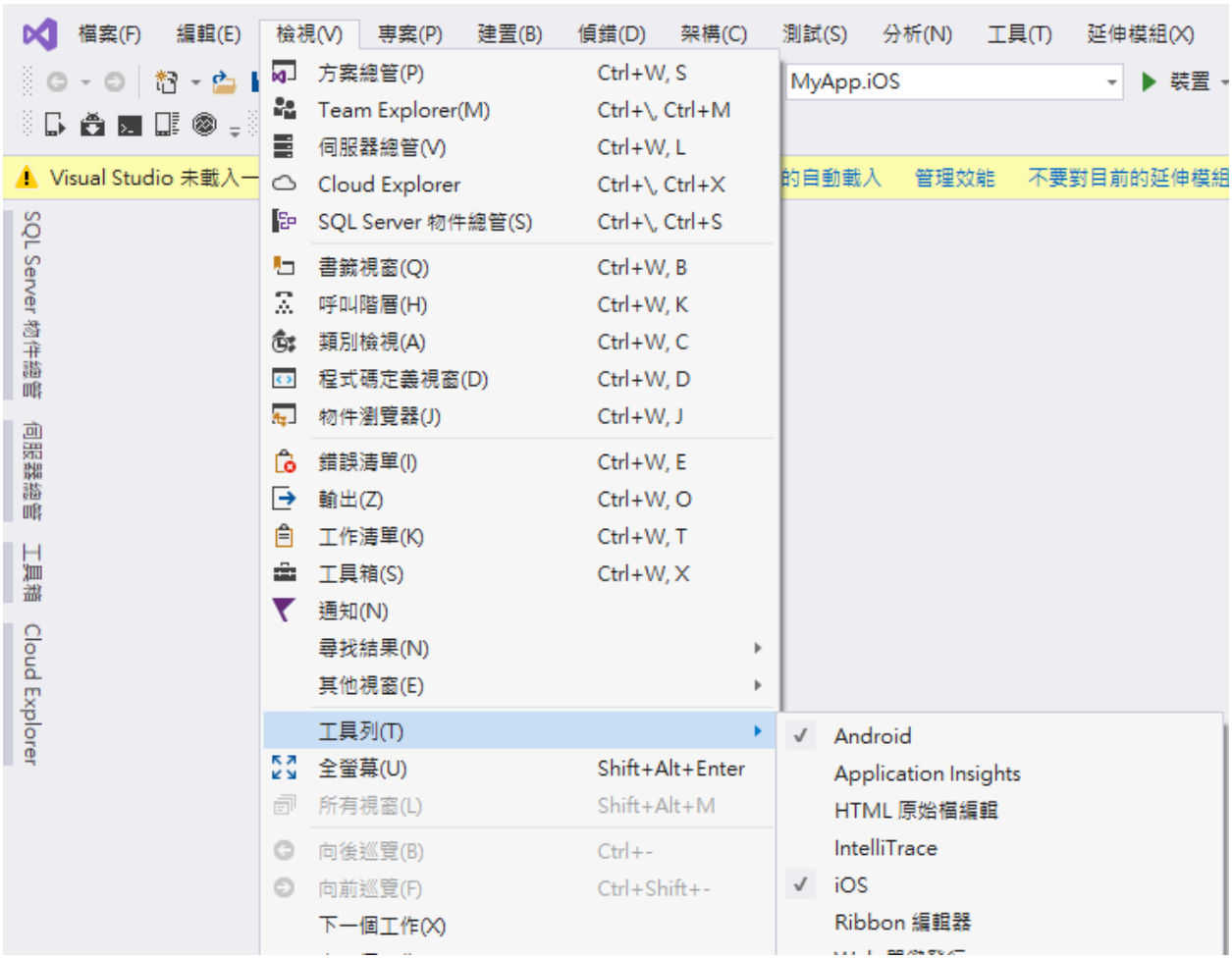
3.3.3 建置與執行 iOS 專案

- 滑鼠右擊 iOS 專案

  選擇設定為起始專案

- 建議顯示 iOS 工具列

  請點選功能表 [檢視] > [工具列] ，請勾選 [iOS] 這個功能表項目



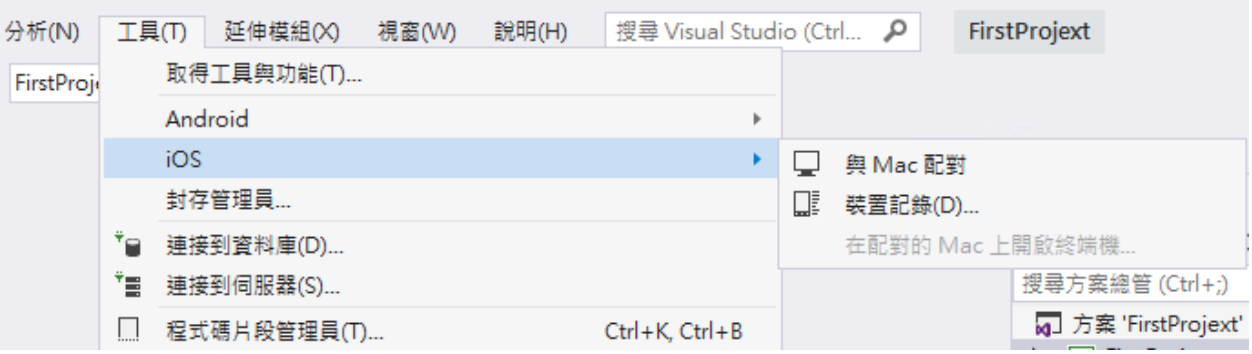
顯示 iOS 輔助工具列圖示

現在，可以在 Visual Studio 2019 最上方的工具列中，看到 iOS 使用的相關工具圖示

Pair to Mac 圖示

- 點選上方圖片中，紅色箭頭指向的圖示，[Pair to Mac - Disconnected]

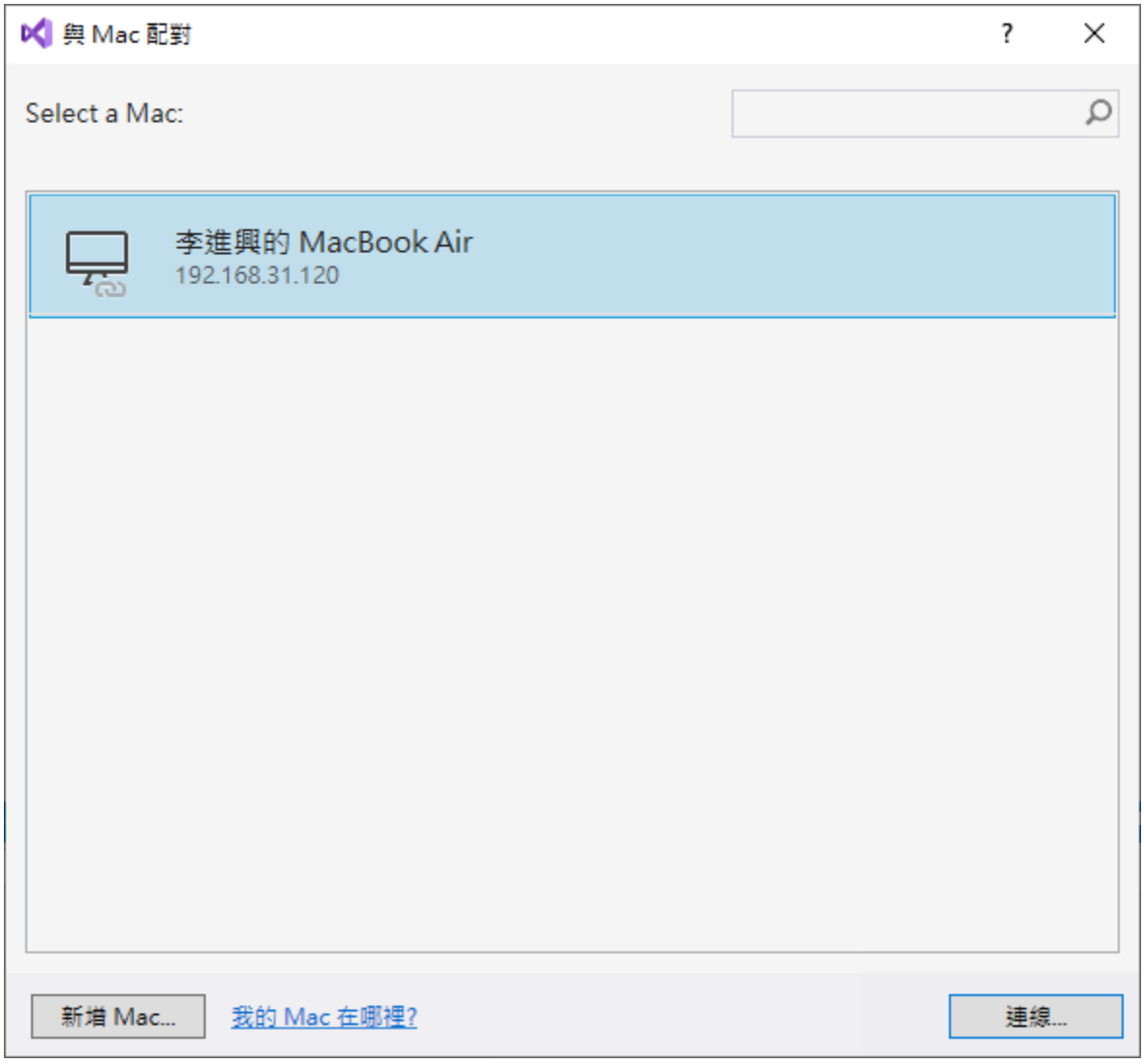
  當然，這個操作也是可以從功能表中點選 [工具] > [iOS] > [與 Mac 配對]



從功能表中來啟用與 Mac 配對功能

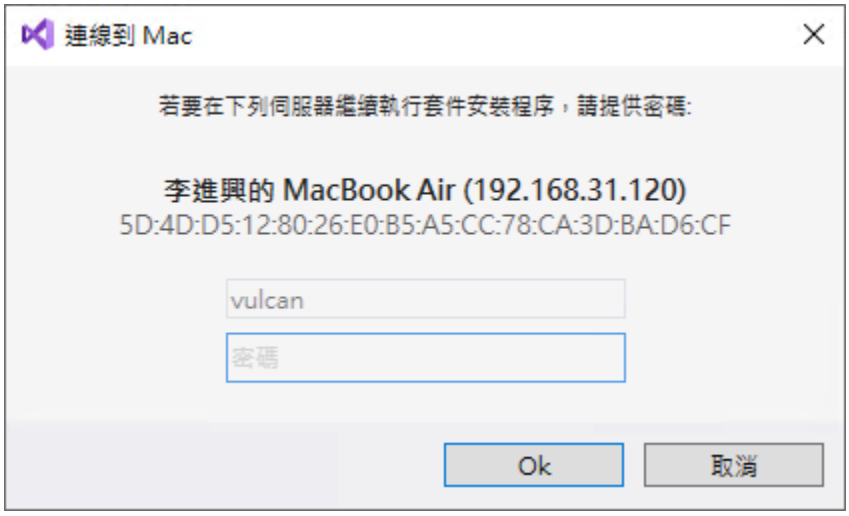
- 現在與 Mac 配對對話窗將會出現

  然而，Visual Studio 將會搜尋網路上是否有可以遠端存取的 Mac 電腦，若有存在的話，該台 Mac 電腦就會出現在清單中



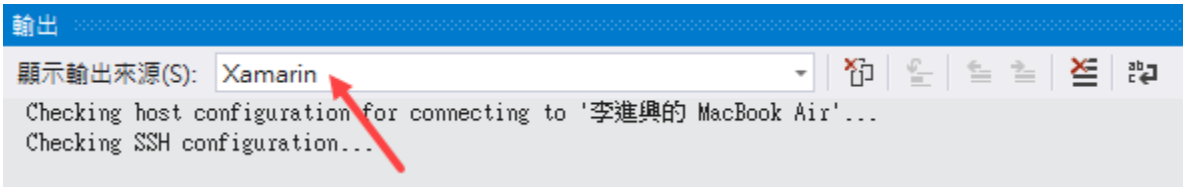
自動掃描網路並且是有開啟遠端存取的 Mac 電腦

- 在 Select a Mac 清單內，找到您的 Mac 電腦，並且使用滑鼠雙擊這個項目，此時，將會出現 [連線到 Mac] 對話窗，請在這個對話窗中，輸入遠端登入 Mac 電腦的使用者名稱與密碼，完成後，點選 [Login] 按鈕



輸入要遠端登入到 Mac 電腦的使用者帳號與密碼

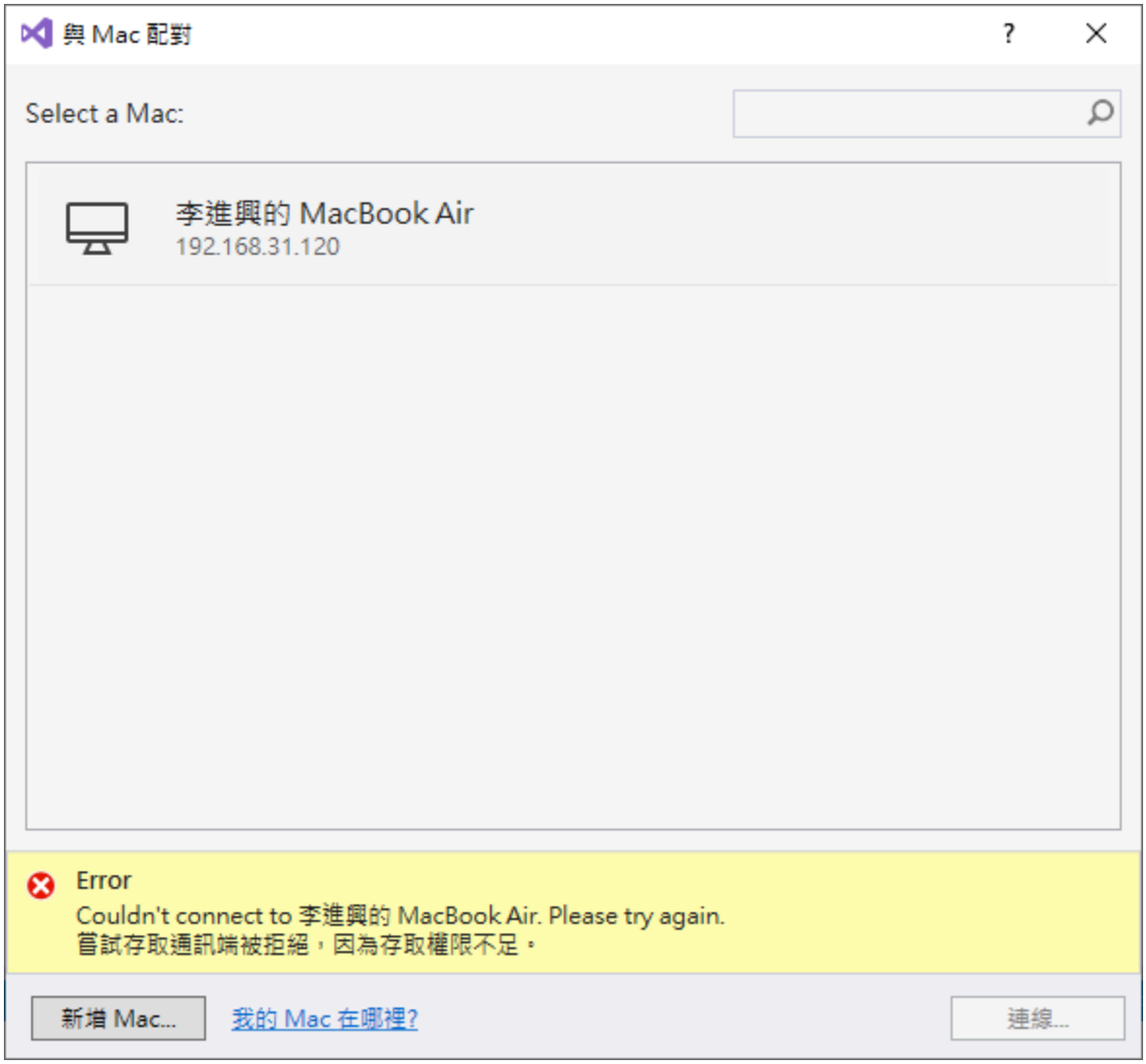
不過，在這裡強烈建議在 Visual Studio 的 [輸出] 視窗中，切換 [顯示輸出來源] 清單項目成為 [Xamarin]，因為，接下來要對遠端 Mac 電腦進行各種登入、更新等動作，都可以在這裡看到相關日誌訊息，最重要的是，當您無法連線到遠端的 Mac 電腦或著覺得連線速度有些緩慢，便可以從這些輸出日誌內容，找到真正發生的原因。



Windows 電腦與 Mac 電腦存取的日誌

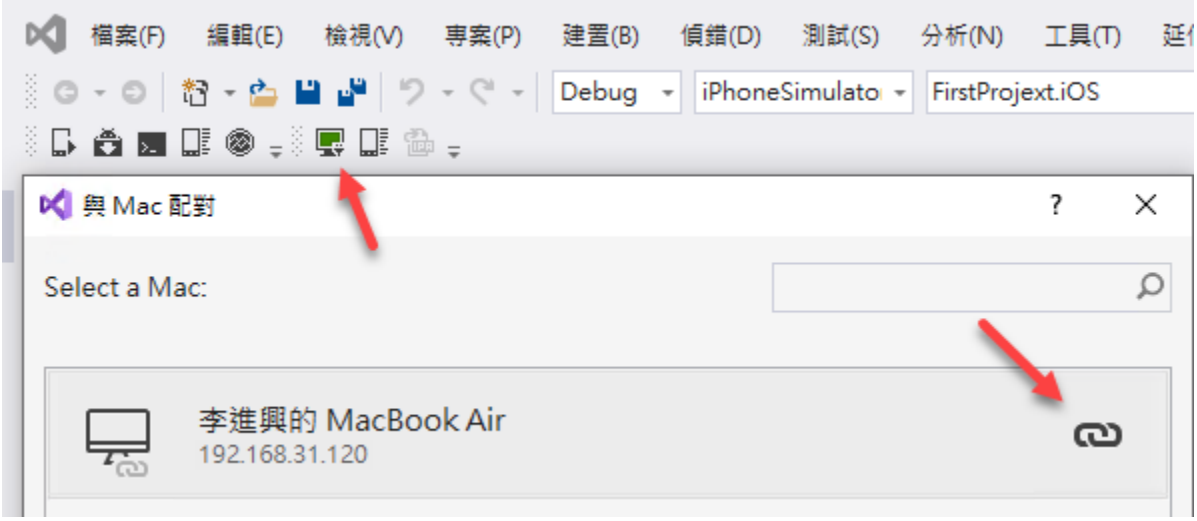
像是底下的為一個登入 Mac 電腦失敗的情境，在這裡顯示出 嘗試存取通訊端被拒絕，因為存取權限不足錯誤訊息，此時，可以從 Visual Studio 輸出視窗中，看到更加詳細的資訊。

Checking host configuration for connecting to ' 李進興的 MacBook Air'...
Checking SSH configuration...
正在檢查可用的磁碟空間...
正在檢查 Mono 安裝...
正在檢查 Xamarin iOS 安裝...
Checking host configuration for connecting to ' 李進興的 MacBook Air'...
Host ' 李進興的 MacBook Air' is configured correctly
Starting connection to ' 李進興的 MacBook Air'...
Starting connection to ' 李進興的 MacBook Air'...
Starting disconnection from 李進興的 MacBook Air...
Starting disconnection from 李進興的 MacBook Air...
The connection to ' 李進興的 MacBook Air' has been finished
Couldn't connect to 李進興的 MacBook Air. Please try again.



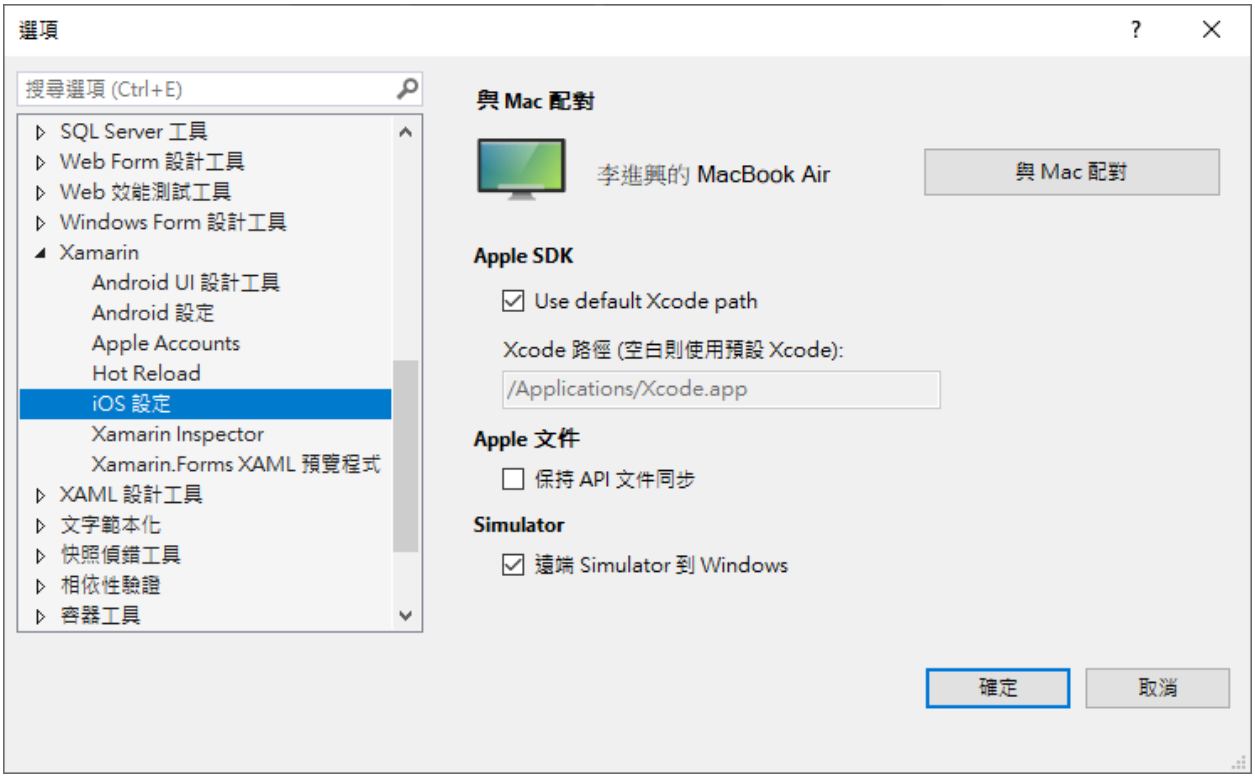
與 Mac 電腦連線失敗範例

- 當與遠端 Mac 電腦連線成功之後，工具列上的 [Pair to Mac] 圖示將會變成綠色螢幕，而且在 [與 Mac 配對] 對話窗中，剛剛連線的 Mac 電腦項目的右方，也會出現一個已經連線的圖示，也就是說，現在可以開始建置與執行 iOS 的專案了。



與 Mac 電腦連線成功的畫面

- 請點選功能表的 [工具] > [選項]，當選項對話窗顯示之後，請在左邊清單，展開 [Xamarin] 節點，找到 [iOS 設定] 項目，請確認右方的 [遠端 Simulator 到 Windwos] 選項要有勾選，最後點選右下方的 [確定] 按鈕

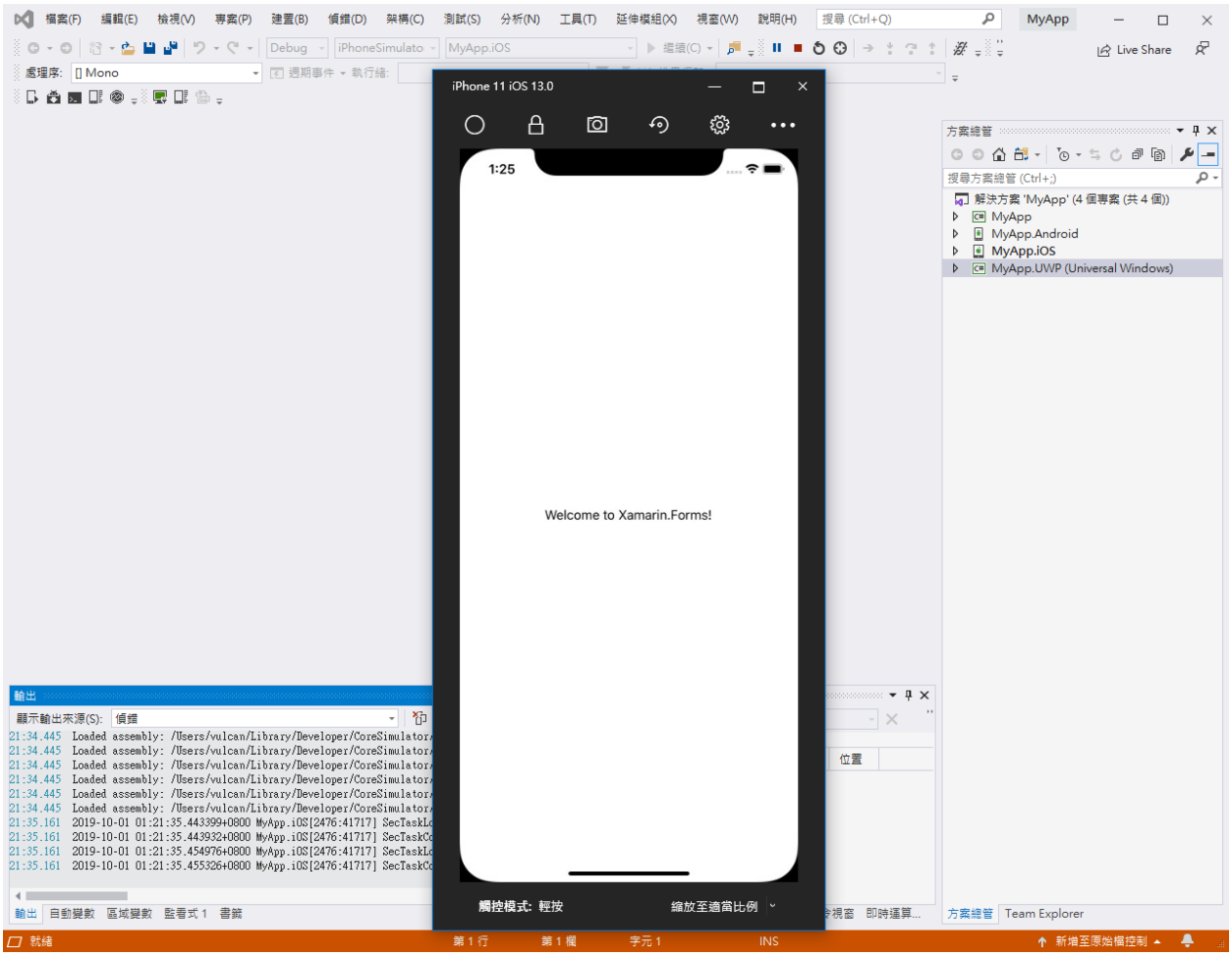


啟用遠端 Simulator 到 Windwos

- 請在 Visual Studio 2019 最上方的工具列中，找到 [方案平台] 下拉選項，請在這裡選擇 [iPhoneSimulator] 這個選項，此時，右方下拉選項就會顯示出遠端 Mac 電腦中的各式模擬器清單，選擇一個適合您的模擬器項目，接著點選該項目的綠色按鈕，開始建置與執行 iOS 專案。

切換使用 iOS 模擬器來進行除錯

- 底下將會是成功執行 iOS 專案的畫面，也就是說，當要模擬器來進行 iOS 專案執行或除錯的時候，iOS 模擬器將會直接顯示在 Windows 電腦中，如此，就不需要在進行開發過程，同時關注兩個系統螢幕上出現的內容了。



Xamarin.Forms 的 iOS 專案成功執行結果

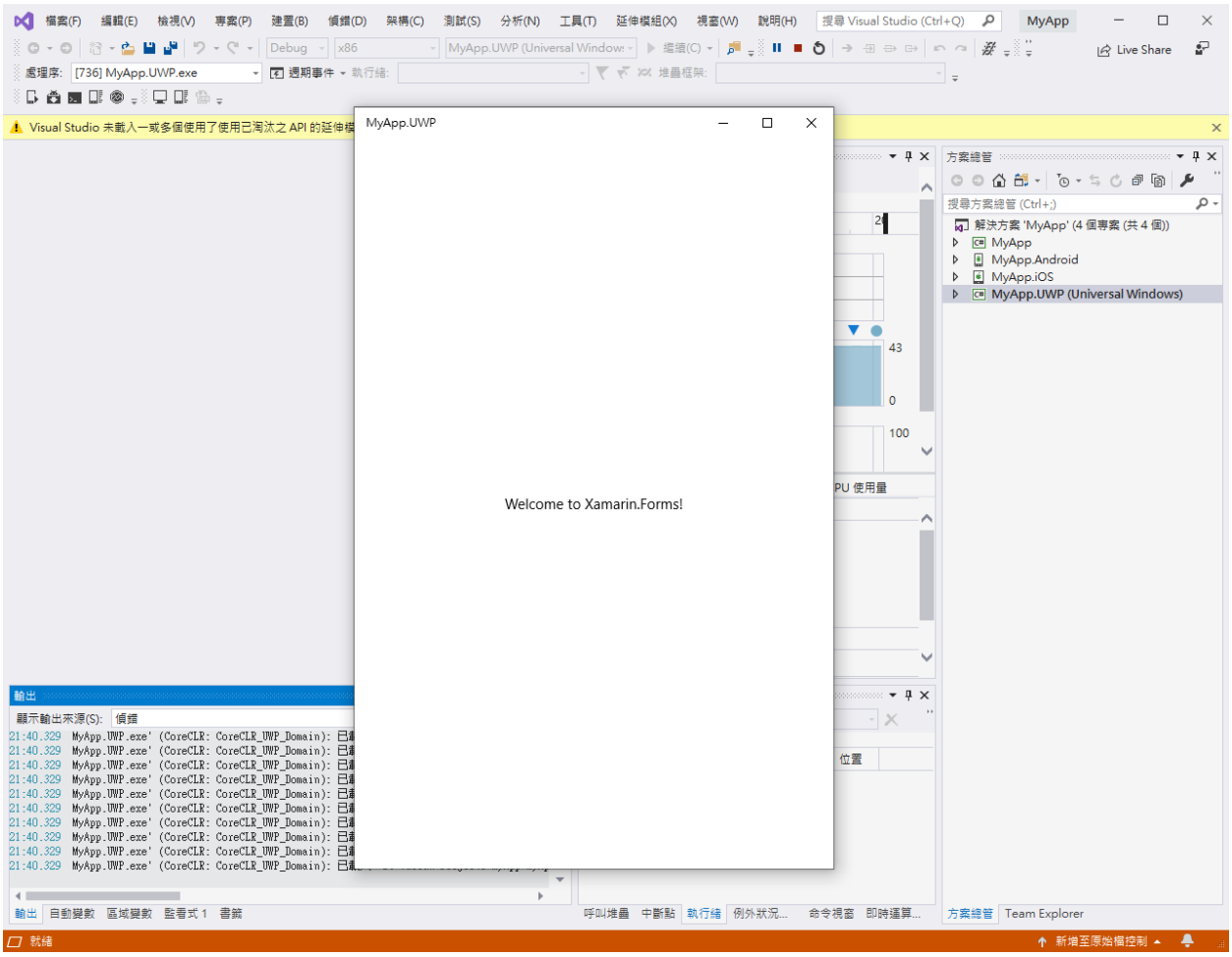
- 停止 iOS 專案的執行

3.3.4 建置與執行 UWP 專案

- 使用滑鼠右擊 UWP 專案

  選擇設定為起始專案
- 點選 Visual Studio 2019 最上方工具列綠色三角形的本機電腦

  底下將會是成功執行 UWP 專案的畫面



Xamarin.Forms 的 UWP 專案成功執行結果

- 停止 UWP 專案的執行

3.4 結論

若您的 Visual Studio 2019 可以成功建立 Xamarin.Forms 專案，可以建置在 Android / iOS / UWP 平台下執行，那麼，恭喜您，您的 Visual Studio 2019 開發環境，已經可以正常進行 Xamarin.Forms 跨平台專案開發了

II Xamarin.Forms 開發方式與基本概念

在這個部分，將會首先說明在使用 Xamarin.Forms UI 工具集 Toolkit 來進行跨平台專案開發的時候，可有三種方式來選擇，分別是：

1. 完全使用 C# 程式語言來進行頁面 UI 與該 App 的商業運作邏輯部分
2. 使用 Xamarin.Forms 內建的相關開發技術與使用標記語言 XAML 來描述每個頁面的 UI 會顯示的內容，對於商業邏輯部分將會使用 C# 程式語言搭配 Code Behind 的開發方式，另外，還會搭配資料綁定 Data Binding 的技術，當然，也可以自行設計支援 MVVM 設計模式的相關 API
3. 則是使用已經設計好的 MVVM 開發框架，無須自己再來設計這些 MVVM 會用到的 API，例如：Prims 等。

何謂 Xamarin.Forms

由於每個行動裝置開發平台，對於 UI 設計上與整個執行生命週期運作方式都不盡相同，而且都需要使用原生 SDK API 規範來進行開發，例如，採用 Xamarin.Android 與 Xamarin.iOS 的開發方式，就是使用這樣的開發過程。

所以，Xamarin.Forms 就將行動應用程式開發的 UI 設計上的需求進行抽象化，開發者可以透過 Xamarin.Forms 所提供的各種抽象化 UI 控制項元件的 API 來進行各種行動應用程式的畫面設計，一旦專案建置完成之後，就可以在 Android , iOS , UWP 平台下來進行執行，此時，Xamarin.Forms 的 UI 工具集 Toolkit 將會轉換這些使用抽象 UI 設計 API，成為使用當時執行平台的原生 SDK API 並且來顯示出這些 UI 控制項在螢幕上。這樣 Xamarin.Forms 達成了僅需要設計一個 UI 畫面的設計工作，就可以在不同行動裝置平台下執行的跨平台開發工具了。

關於 MVVM 設計模式

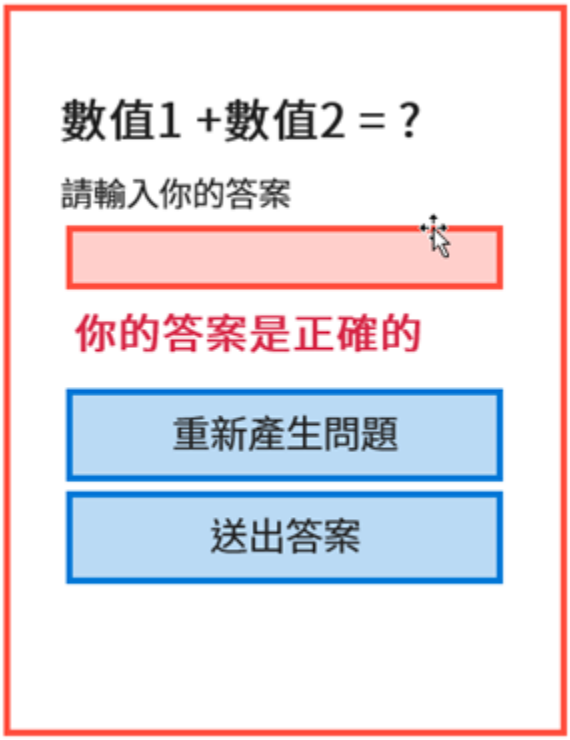
在本書中，將會先針對前兩者進行說明，對於第三種，使用市面上開發好的 MVVM 開發框架來進行開發出 Xamarin.Forms 的應用程式，稍後將會進行介紹，不過，建議可以採用市面已經存在的許多 MVVM 開發框架來使用，有興趣的人，可以到作者部落格[2] 看到更多關於如何使用 Prism 開發框架，使用 MVVM 設計模式開發出來的 Xamarin.Forms 設計過程。

為了讓讀者可以知道這兩種 Xamarin.Forms 的設計過程、開發方式與整個 Xamarin.Forms 方案架構，因此，將會帶領大家來設計一個遊戲，那就是設計一個手機 App，該 App 將會自動產生兩個數值，請使用者輸入這兩個數值相加後的結果；若答對的話，將會顯示答對的提示訊息，否則，將會顯示出答錯的提示訊息。底下的示意圖片將會是準備要來開發的 App 畫面。

[2] https://mylabtw.blogspot.com/

關於開發練習專案

為了要能夠讓讀者快速了解與學會這些 Xamarin.Forms 開發技術，每個主題都會使用一個範例專案來進行說明如何使用這些技術進行開發，因此，建議讀者跟著文章中的開發步驟，逐步進行開發練習，以便達到最大學習成效。



兩數相加的遊戲設計示意圖

4. 使用 C# 程式語言來直接開發 Xamarin.Forms App

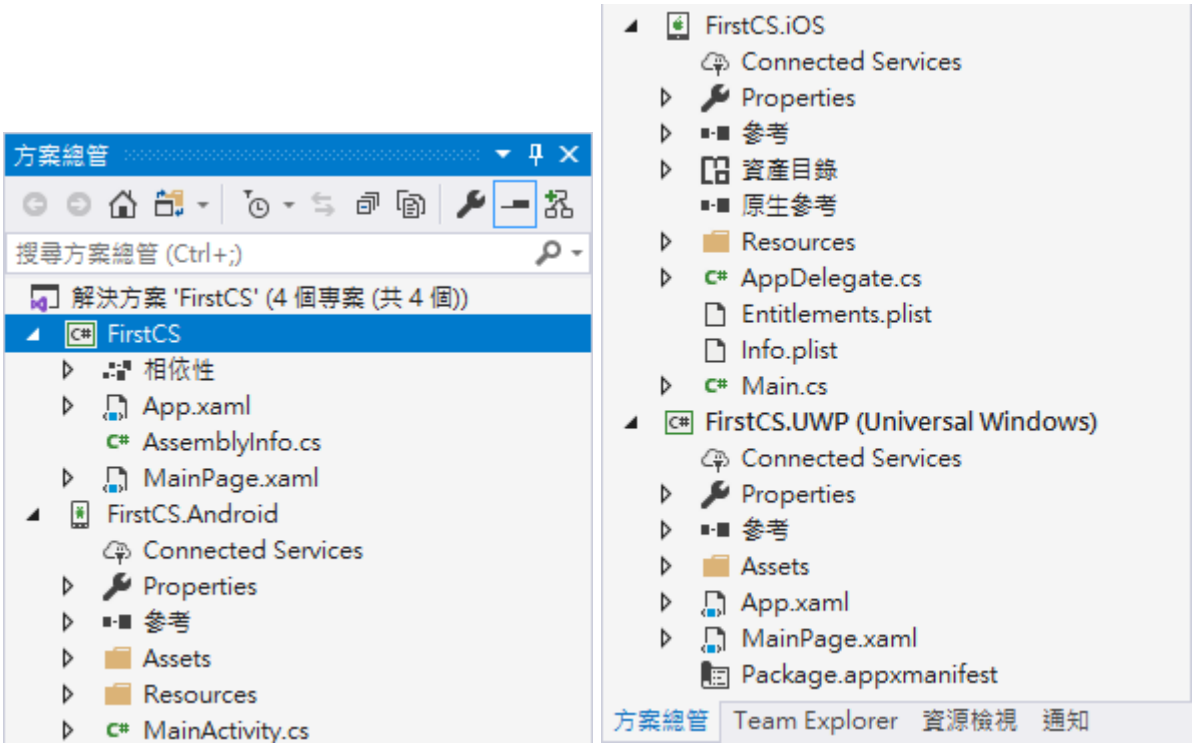
在這個章節，先來說明如何僅使用 C# 程式語言，就可以設計出一個 Xamarin.Forms 的跨平台 App。

4.1 建立一個 Xamarin.Forms 方案

- 請啟動 Visual Studio 2019
- 在啟動後的對話窗內，選擇右下方的 [建立新的專案] 選項
- 當顯示出 [建立新專案] 對話窗，請在中間上方的文字輸入盒內，輸入 Xamarin 這個關鍵字
- 此時將會在中間清單區域，找到 [行動應用程式 (Xamarin.Forms)] 這個選項，請點選這個項目
- 在 [設定新的專案] 對話窗內，請在專案名稱欄位內，輸入 FirstCS
- 請點選右下方的 [建立] 按鈕
- 現在將會跳出一個 [New Cross Platform App - FirstCS] 對話窗
- 請選擇選取範本區域內的 [空白] 項目
- 在下方的 [平台] 區域，請記得勾選 Android , iOS , Windows (UWP) 這三個檢查盒 Checkbox
- 點選右下方的 [OK] 按鈕
- 開始建立 Xamarin.Forms 開發方案

4.2 了解 Xamarin.Forms 方案的結構

現在，Visual Studio 2019 正在進行 Xamarin.Forms 跨平台方案的產生工作，當這個方案產生完成之後，將會看到這樣的方案結構出現在 Visual Studio 2019 的 [方案總管] 視窗內，其結果如下面螢幕截圖。



從上面畫面可以看的出來，當 Visual Studio 2019 建立一個 Xamarin.Forms 解決方案的時候，將會產生出四個專案，這些專案的意義如下所說明

- FirstCS

  這個專案是一個 .NET Standard 2.0 為基礎的類別庫，通常稱這個專案為 Xamarin.Forms 的核心 Core 專案，或者稱為共用核心專案。在這個專案內將可以存取 Xamarin.Forms 所提供的 UI 與相關 API。

  另外，將會在這裡撰寫整個應用程式的商業邏輯程式碼，簡單的來說，那就是對於一個跨平台專案開發過程中，將會在這個專案內設計不同平台都會使用到的共同 UI 與

商業邏輯程式碼，如此，對於相同的設計內容，就僅需要設計一次，並且可以提供給 Android , iOS , UWP 專案內來存取，達到最大共享程度。

Xamarin.Forms 這個工具對於 UI 設計部分，是將每個平台都會有提供的 UI 進行抽象化，讓程式設計師可以在 Xamarin.Fomrs 核心專案內使用這些 Xamarin.Forms 提供的 UI API 來進行設計；一旦專案要在如 Android 平台下執行的時候，就會自動轉換使用 Android 平台下提供的 UI 元件來顯示在裝置畫面上，若專案要 iOS 平台下運行，當然就會自動採用 iOS 平台下的 UI 元件顯示在螢幕上。

當然，每個平台下的 UI 使用方式與呼叫方法都不盡相同，這樣的問題對於採用 Xamarin.Forms UI Toolkit 工具集來進行開發的專案是不會有問題的，因為，這些複雜的轉換工作都會有 Xamarin.Forms 工具來解決。

至於非 UI 的商業邏輯部分，當然是要回歸到 .NET 平台所提供的解決方案，那就是這些商業邏輯的需求，都可以透過 C# 程式碼來完成，講的白話些，那就是原先在 .NET 開發框架下，是如何呼叫遠端 Web API、存取檔案、進行 JSON 序列與反序列化、圖片檔案的放大與縮小等等，還是使用相同的程式碼與相同的 C# 程式語言來進行設計；更美的的是，只要這些商業邏輯程式碼是採用 .NET Standard 類別庫所開發出來的，就可以直接在 Xamarin.Forms 專案中來參考與呼叫，無須任何轉換。

通常在進行跨平台專案開發的時候，大約會有 80%~95% 的程式碼是每個平台都會用到的，剩下的部分就需要在個別的專案下進行設計，因為，所要執行的相關工作是需要使用到每個平台下才會提供的 API。這些專屬平台下的 API 呼叫包括了：訊息推播、裝置感應器與資源的存取等等，這些 API 不論是在使用方式，類別名稱，方法名稱與參數型別和數量，設定與呼叫方式，都完全不相同，因此，需要在每個平台下個別進行設計。不過，許多這樣的功能都已經開發成為 NuGet 套件，因此，只需要安裝這些 NuGet 套件，就可以在 Xamarin.Forms 核心專案內使用這些套件類別庫提供的 API 進行呼叫，就可以存取與使用到原生平台下的各種資源了。

- FirstCS.Android

  這是個使用 Xamarin.Android 開發框架所建立的專案，通稱為 Android 原生專案，透過這個專案可以產生出一個可在 Android 平台下執行的 App，當然，在這個專案內所撰寫的任何程式碼，都可以呼叫各種 Android SDK 所提供的 API。在這個專案下

的相關 Android 系統開發的設定、參數設定、系統運作方式等等，完全與 Android 原生 SDK 相同，唯一的不同點那就是只有使用的程式語言是採用 .NET C#；因此，當想要針對 Android 專案進行相關擴充與設計的時候，這些可以使用的功能都會記載在 Android 原生 SDK 上。

這個專案也是一個 Android App 啟動的時候，第一要執行程式碼，也就是在 MainActivity.cs 檔案內的 MainActivity 這個類別，通稱為 Android 專案的程式進入點。

由於在建置一個 Xamarin.Forms for Android 專案的時候，會需要用到 Android 原生 SDK 的開發環境，因此，這台開發電腦上還是需要把相關 Android SDK 會用到的檔案與套件都安裝與設定好。

- FirstCS.iOS

  這是個使用 Xamarin.iOS 開發框架所建立的專案，通稱為 iOS 原生專案，透過這個專案可以產生出一個可在 iOS 平台下執行的 App，當然，在這個專案內所撰寫的任何程式碼，都可以呼叫各種 iOS SDK 所提供的 API。在這個專案下的相關 iOS 系統開發的設定、參數設定、系統運作方式等等，完全與 iOS 原生 SDK 相同，唯一的不同點那就是只有使用的程式語言是採用 .NET C#；因此，當想要針對 iOS 專案進行相關擴充與設計的時候，這些可以使用的功能都會記載在 iOS 原生 SDK 上。

  這個專案也是一個 iOS App 啟動的時候，第一要執行程式碼，也就是在 AppDelegate.cs 檔案內的 AppDelegate 這個類別，通稱為 iOS 專案的程式進入點。

  不過，要特別注意的是，當在進行 iOS App 開發的時候，需要一台 Mac 電腦，這台電腦上要安裝 Xcode 與 Visual Studio 2019 for Mac；另外，若想要針對在 iPhone 實體手機上進行開發除錯的時候，那就需要另外在準備一個蘋果開發者帳號，而且需要經過一些設定工作，才能夠在實體裝置上進行開發與除錯，否則，就僅能夠使用 iOS SDK 所提供的 iOS Simulator 模擬器來進行開發和除錯了。

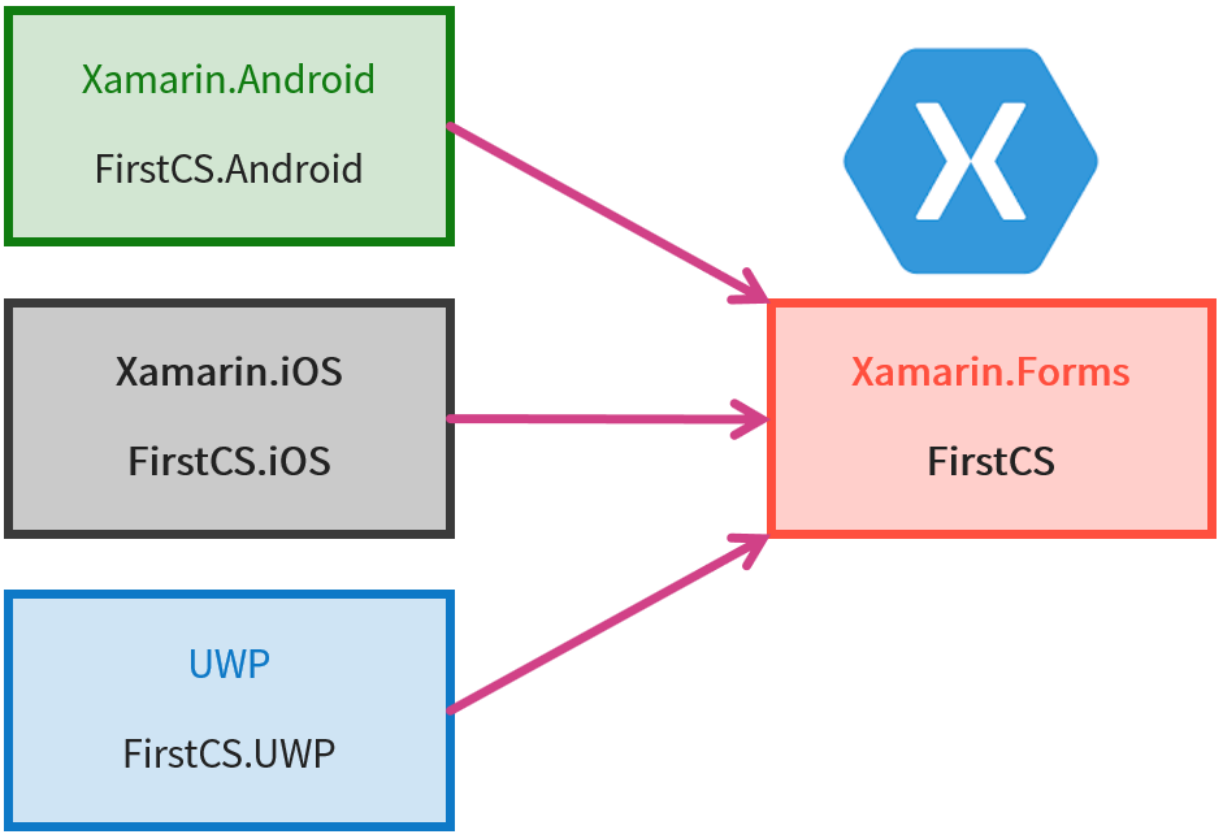
- FirstCS.UWP

  這是個使用 UWP 開發框架所建立的專案，通稱為 UWP 原生專案，這裡並沒有任何 Xamarin.UWP 這樣的開發框架存在，而就是直接使用微軟提供的 UWP SDK，透過

這個專案可以產生出一個可在 UWP 平台下執行的 App，當然，在這個專案內所撰寫的任何程式碼，都可以呼叫各種 UWP SDK 所提供的 API。在這個專案下的相關 UWP 系統開發的設定、參數設定、系統運作方式等等，完全與 UWP 原生 SDK 相同，唯一的不同點那就是只有使用的程式語言是採用 .NET C#；因此，當想要針對 UWP 專案進行相關擴充與設計的時候，這些可以使用的功能都會記載在 UWP 原生 SDK 上。

這個專案也是一個 UWP App 啟動的時候，第一要執行程式碼，也就是在 App.xaml.cs 檔案內的 App 這個類別，通稱為 UWP 專案的程式進入點。

所以對於所產生的 Xamarin.Forms 方案內的四個專案，將會存在如下圖的關係。



Xamarin.Forms 方案內的專案關係

在上圖的每個區塊，都有兩行文字標示，第一行文字表示該專案所使用的開發框架

Framework 的名稱，而下方的文字則是這個剛剛建立起來的練習專案名稱。

也就是說，對於 Android 類型的 App，將會由 FirstCS.Android 這個可以完全使用 Android 原生 SDK 的專案與 FirstCS 這個僅能夠使用 Xamarin.Forms 提供的相關 API 的專案所產生出來的兩個組件 Assembly 檔案所組成；並且，由於 FirstCS.Android 這個專案需要參考 FirstCS 專案，因此，在這個 FirstCS.Android ，Android 原生專案，除了可以呼叫與使用任何 Android 原生 SDK 提供的 API，也可以存取 FirstCS 專案內的相關型別與程式碼。

反過來說，對於 FirstCS 專案而言，在這個專案內就僅能夠呼叫 Xamarin.Forms 所提供的 API，無法呼叫任何 Android 原生 SDK 內的 API；當然，若想要這麼做，是可以使用之後就會介紹的相依服務 Dependency Service 與訊息中心 MessageingCenter 這兩個 Xamarin.Forms 內建的功能來完成。

而且最為重要的是，整開發整個跨平台 App 的時候，開發者幾乎所有的時間都會在 FirstCS 這個專案內來進行程式碼的設計，也就是要進行使用者畫面與商業邏輯的設計；對於前者將會使用這個 FirstCS 專案內安裝的 Xamarin.Forms NuGet 套件所提供的相關抽象 UI API，就可以設計出跨平台的 UI App，對於商業邏輯部分，直接使用 .NET C# 程式語言提供的技術即可。

所以當要產生出 iOS 類型的 App，就需要透過 FirstCS.iOS 這個專案與 FirstCS 專案所建置出來的兩個組件，就可以在 iOS 裝置性來執行；同樣的，對於 Windows UWP 類型的 App，也是使用同樣的規則，透過 FirstCS.UWP 專案與 FirstCS 專案建置出來的兩個組件，就可以在 Windows UWP 平台下電腦與裝置來運行了。

Xamarin.Forms 的套件

Xamarin.Forms UI Toolkit 工具集，其實就是一個 NuGet 套件[1]，也就是說，想要在當前的專案內要進行 Xamarin.Forms 需求的開發，就僅需要加入這個 Xamarin.Forms NuGet 套件即可，如此，就可以開始使用 Xamarin.Forms 所提供的相關 API，進行跨平台的設計。

也就是說，若要更新 Xamarin.Forms 到新版本，僅需要更新所有專案內的 Xamarin.Forms 這個 NuGet 套件到最新版本即可。

共用商業邏輯程式碼與螢幕 UI 控制項

對於一個行動裝置 App 而言，其實需要進行兩大類型需求的開發設計，那就是該 App 螢幕畫面 UI/UX 的設計、以及相關商業邏輯的需求設計。每個平台所用到的 API 方法與名稱和用法、UI 控制項的名稱與用法皆完全不相同，若在進行跨平台 App 開發的時候，需要針對 Android 平台與 iOS 平台下的螢幕畫面逐一特別進行設計，將會是個很浪費開發成本的工作。

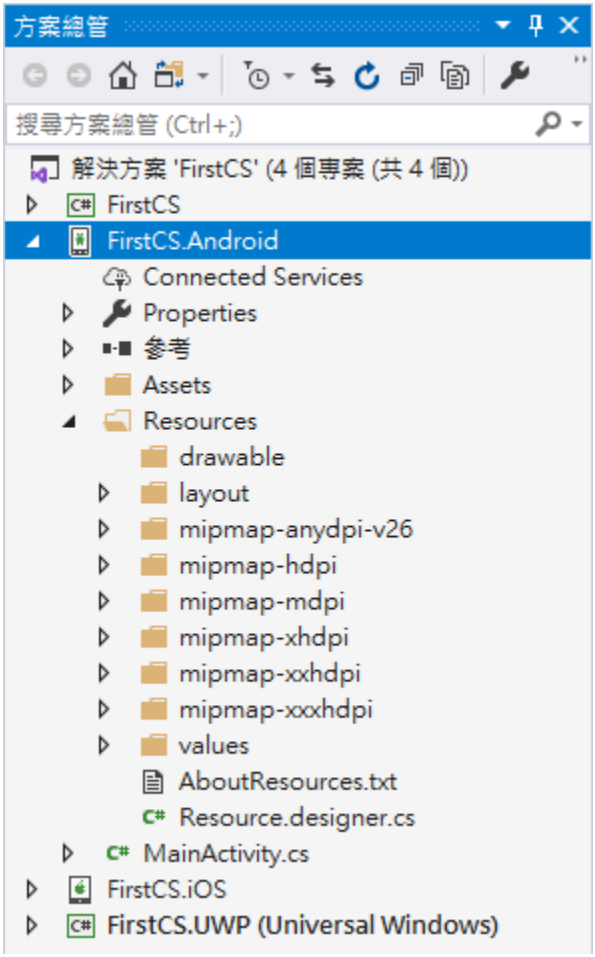
畢竟對於一個跨平台 App 而言，不論在哪個平台下運行，其實，都應該要呈現出同樣的 UI 控制項與文字內容才對，而不是要逐一特別在每個平台下都重複設計一次相同的 UI 畫面，Xamarin.Forms UI Toolkit 工具集就是用來解決這個問題，App 開發者僅需要使用 Xamarin.Fomrs 所提供的 API，就可以設計出同時在 Android / iOS / UWP 平台下的 UI 畫面，而且最重要的是，這些工作僅僅需要做一次即可，便可以將這些需求與 Android / iOS / UWP 的專案來共用，對於商業邏輯部分，直接使用 .NET C# 程式語言提供的技術即可。

因此，Xamarin.Forms 的解決方案就是要解決共用商業邏輯程式碼與螢幕 UI 控制項的問題，讓開發者無須專注在不同行動裝置平台，專心使用 Xamarin.Forms 所提供的 API ，就可以設計出跨平台的 App 了。

[1] https://www.nuget.org/packages/Xamarin.Forms/

4.2.1 了解 Xamarin.Android 專案的結構與運作方式

首先先來了解 Xamarin.Android 的專案結構，當展開 Xamarin.Android 專案之後，將會呈現如下圖的畫面。



Xamarin.Android 專案結構

請先展開 Properties 節點，將會看到 AndroidManifest.xml 檔案，這個檔案是用來設定 Android 原生專案的使用權限與相關設定的地方，這是一個 XML 資料格式的檔案，不過，在 Xamarin.Android 開發框架下，許多用使用 XML 來進行宣告的設定工作，可以透過

C# Attribute 屬性宣告的方式就可以直接完成設定工作，不需要自行使用 XML 語法來修改這個 AndroidManifest.xml 檔案，因為，在建置 Xamarin.Android 專案的時候，將會自動產生出相對應的 XML 設定宣告內容。

對於 App 中會使用到的本機圖片檔案資源，將會使用 Resources 目錄下的 drawable 目錄來儲存這些圖片檔案，從上面的螢幕截圖可以看到預設是沒有這個 drawable 這個目錄，因此，可以將要顯示在該 App 上的圖片檔案，透過檔案總管拖拉到這個 drawable 目錄下即可。

圖片檔案資源

在使用 Xamarin.Forms 開發跨平台 App 的時候，對於 App 中會使用到的圖片檔案資源，需要分別將這些圖片檔案拖拉到每個原生專案內的指定目錄下，而且對於同一個圖片資源，其檔案名稱最好是具有同樣的檔案名稱；另外，對於 Xamarin.Android 專案而言，這個 drawable 目錄是標示圖片檔案資源的根目錄，例如，若有個圖片檔案名稱為 MyImage.png 存在於 drawable 目錄下，在 Xamarin.Forms 專案內若想要參考這個圖片資源，僅需要標示這個圖片檔案名稱即可 (這裡是使用 XAML 的標記語言來宣告一個圖片控制項的用法 `<Image Source="MyImage.png">`)。

當然，在不同的原生專案平台下，圖片儲存的位置是不相同的，而且這些圖片一定需要存放在每個原生專案平台下 (為了要能夠使用自動放大倍率選取功能)，不需要將這些圖片檔案放置到 Xamarin.Forms 專案內。

對於非圖片類型的檔案，例如：文字檔案、聲音檔案、影片檔案、資料庫檔案等等，將需要把這些檔案存放到 Assets 目錄下。

對於 Xamarin.Android 原生專案內的檔案結構與使用方式，都與使用 Android 原生 SDK 開發用法相同。

4.2.2 Xamarin.Android 的專案進入點

最後來了解 Xamarin.Android 專案是如何運行的，一旦一個 Xamarin.Android 專案啟動之後，將會依照 Android 原生 SDK 的規範，使用 MainActivity 這個類別開始來執行，底下是這個檔案的程式碼。

MainActivity.cs

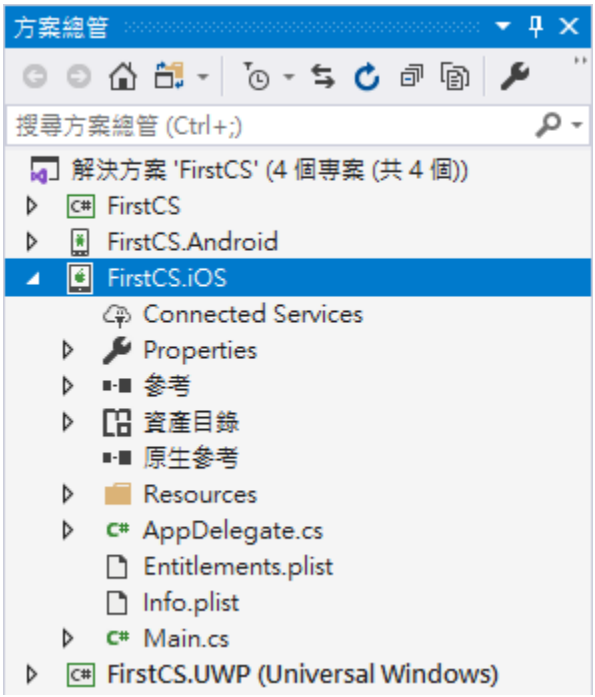
```
namespace FirstCS.Droid
{
   [Activity(Label = "FirstCS", Icon = "@mipmap/icon", Theme = "@style/MainTheme", \
MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges\
.Orientation)]
   public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompa\
tActivity
   {
      protected override void OnCreate(Bundle savedInstanceState)
      {
         TabLayoutResource = Resource.Layout.Tabbar;
         ToolbarResource = Resource.Layout.Toolbar;

         base.OnCreate(savedInstanceState);

         Xamarin.Essentials.Platform.Init(this, savedInstanceState);
         global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
         LoadApplication(new App());
      }
      public override void OnRequestPermissionsResult(int requestCode, string[] pe\
rmissions, [GeneratedEnum] Android.Content.PM.Permission[] grantResults)
      {
         Xamarin.Essentials.Platform.OnRequestPermissionsResult(requestCode, perm\
issions, grantResults);

         base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
      }
   }
}
```

在這個 MainActivity 類別內，有覆寫 override 一個 OnCreate 方法，這個方法將會是 Android 專案一開始執行的地方，其中這個敘述 `global::Xamarin.Forms.Forms.Init(this, savedInstanceState);` 會是開始進行 Xamarin.Forms 套件的初始化運作的程式碼呼叫，接者將會產生一個 App 類別執行個體 (這個 App 類別是宣告在 Xamarin.Forms 專案內的一個類別，等下會看到這個類別的介紹) ，使用敘述 `LoadApplication(new App());` 將整個執行控制權轉交到 Xamarin.Forms 專案內，也就是說，之後運行的程式碼都將會在 Xamarin.Forms 專案內來進行。

4.2.3 了解 Xamarin.iOS 專案的結構與運作方式

接著來查看 Xamarin.iOS 的原生專案結構，當展開 Xamarin.iOS 專案之後，將會呈現如下圖的畫面。

對於 Xamarin.iOS 原生專案內的檔案結構與使用方式，都與使用 iOS 原生 SDK 開發用法相同。



Xamarin.iOS 專案結構

在 Xamarin.iOS 專案下的 info.plist (這個檔案稱為 Information Proerties List 資訊屬性清單)，其性質有點像是 Xamarin.Android 專案內的 AndroidManifest.xml 檔案，他們都是使用 XML 語言來進行宣告相關設定；另外，對於 Entitlements.plist 檔案，也是用於設定這個 iOS 專案相關行為的一個設定檔案。

對於 Xamarin.iOS 專案下要用到的相關圖片檔案、文字檔案、、聲音檔案、影片檔案、資料庫檔案等等，將需要把這些檔案存放到 Resources 目錄下即可，而且這個目錄將會圖片檔案的根目錄。

4.2.4 Xamarin.iOS 的專案進入點

最後來了解 Xamarin.iOS 專案是如何運行的，一旦一個 Xamarin.iOS 專案啟動之後，將會依照 iOS 原生 SDK 的規範，使用 AppDelegate 這個類別開始來執行，底下是這個檔案的程式碼。

AppDelegate.cs

```
namespace FirstCS.iOS
{
    // The UIApplicationDelegate for the application. This class is responsible for \
launching the
    // User Interface of the application, as well as listening (and optionally respo\
nding) to
    // application events from iOS.
    [Register("AppDelegate")]
    public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsAppli\
cationDelegate
    {
        //
        // This method is invoked when the application has loaded and is ready to ru\
n. In this
        // method you should instantiate the window, load the UI into it and then ma\
ke the window
        // visible.
        //
        // You have 17 seconds to return from this method, or iOS will terminate you\
r application.
        //
        public override bool FinishedLaunching(UIApplication app, NSDictionary optio\
ns)
        {
            global::Xamarin.Forms.Forms.Init();
            LoadApplication(new App());

            return base.FinishedLaunching(app, options);
        }
    }
}
```

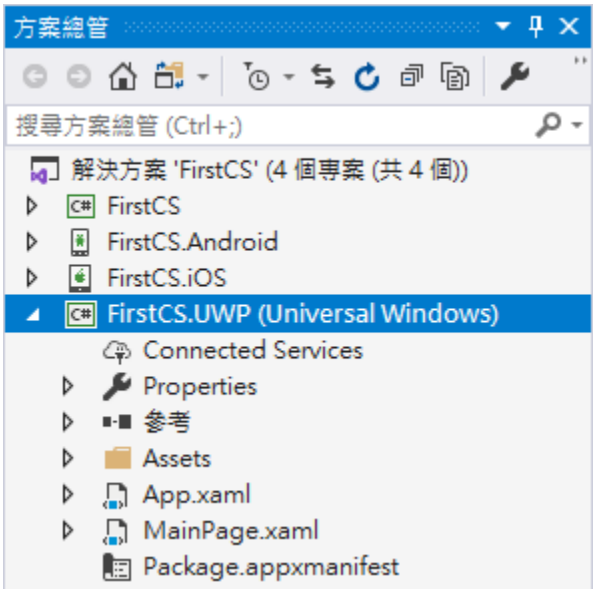
在這個 AppDelegate 類別內，有覆寫 override 一個 FinishedLaunching 方法，這個方法將會是 iOS 專案一開始執行的地方，其中這個敘述 `global::Xamarin.Forms.Forms.Init();` 會是開始進行 Xamarin.Forms 套件的初始化運作的程式碼呼叫，接者將會產生一個 App

類別執行個體 (這個 App 類別是宣告在 Xamarin.Forms 專案內的一個類別，等下會看到這個類別的介紹) ，使用敘述 `LoadApplication(new App());` 將整個執行控制權轉交到 Xamarin.Forms 專案內，也就是說，之後運行的程式碼都將會在 Xamarin.Forms 專案內來進行。

4.2.5 了解 Xamarin.UWP 專案的結構與運作方式

現在來了解 UWP 的原生專案結構，當展開 UWP 專案之後，將會呈現如下圖的畫面。

對於 UWP 原生專案內的檔案結構與使用方式，都與使用 UWP 原生 SDK 開發用法相同。



UWP 專案結構

對於 UWP 專案，相關專案行為與授權權限的宣告與設定，將會在 Package.appxmanifest 檔案中進行設定。

在 UWP 專案下的圖片檔案或者其他檔案資源，如同一般 .NET 專案開發一樣，可以自行

建立一個目錄，將這些檔案放置在任何方案資料夾內，不過，通常將會把相關 UWP 專案會用到的相關檔案存放到 Assets 目錄下。

4.2.6 UWP 的專案進入點

最後來了解 UWP 專案是如何運行的，一旦一個 UWP 專案啟動之後，將會依照 UWP 原生 SDK 的規範，使用 App 這個類別 (這裡的 App 類別將會宣告 UWP 專案內的一個類別，這與剛剛提到的 Xamarin.Forms 內的 App 類別是不同的) 開始來執行，底下是這個檔案的程式碼。


App.xaml.cs

```
namespace FirstCS.UWP
{
   /// <summary>
   /// Provides application-specific behavior to supplement the default Application\
 class.
   /// </summary>
   sealed partial class App : Application
   {
      /// <summary>
      /// Initializes the singleton application object.  This is the first line of\
 authored code
      /// executed, and as such is the logical equivalent of main() or WinMain().
      /// </summary>
      public App()
      {
         this.InitializeComponent();
         this.Suspending += OnSuspending;
      }

      /// <summary>
      /// Invoked when the application is launched normally by the end user.  Othe\
r entry points
      /// will be used such as when the application is launched to open a specific\
```

```
 file.
        /// </summary>
        /// <param name="e">Details about the launch request and process.</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {


            Frame rootFrame = Window.Current.Content as Frame;

            // Do not repeat app initialization when the Window already has content,
            // just ensure that the window is active
            if (rootFrame == null)
            {
                // Create a Frame to act as the navigation context and navigate to t\
he first page
                rootFrame = new Frame();

                rootFrame.NavigationFailed += OnNavigationFailed;

                Xamarin.Forms.Forms.Init(e);

                if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
                {
                    //TODO: Load state from previously suspended application
                }

                // Place the frame in the current Window
                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                // When the navigation stack isn't restored navigate to the first pa\
ge,
                // configuring the new page by passing required information as a nav\
igation
                // parameter
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            // Ensure the current window is active
            Window.Current.Activate();
        }
```

```
        /// <summary>
        /// Invoked when Navigation to a certain page fails
        /// </summary>
        /// <param name="sender">The Frame which failed navigation</param>
        /// <param name="e">Details about the navigation failure</param>
        void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
        {
            throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
        }

        /// <summary>
        /// Invoked when application execution is being suspended.  Application stat\
e is saved
        /// without knowing whether the application will be terminated or resumed wi\
th the contents
        /// of memory still intact.
        /// </summary>
        /// <param name="sender">The source of the suspend request.</param>
        /// <param name="e">Details about the suspend request.</param>
        private void OnSuspending(object sender, SuspendingEventArgs e)
        {
            var deferral = e.SuspendingOperation.GetDeferral();
            //TODO: Save application state and stop any background activity
            deferral.Complete();
        }
    }
}
```

想要看到上述程式進入點的程式碼，請在 UWP 專案內找到 App.xaml 這個檔案節點，點選該檔案名稱前面的三角形，展開這個節點，如此便會看到這個 App.xaml.cs 節點，使用滑鼠雙擊這個 App.xaml.cs 節點名稱，就會看到上述的程式碼。

在這個 App 類別內，有覆寫 override 一個 OnLaunched 方法，這個方法將會是 UWP 專案一開始執行的地方，其中這個敘述 `Xamarin.Forms.Forms.Init(e);` 會是開始進行 Xamarin.Forms 套件的初始化運作的程式碼呼叫。

接者請打開 MainPage.xaml.cs 這個檔案節點 (請找到 UWP 專案內的 MainPage.xaml 檔

案節點，點選該節點前面的三角形圖示，就會看到這個 MainPage.xaml.cs 檔案節點了)，此時，該 MainPage.xaml.cs 的程式碼如下所示

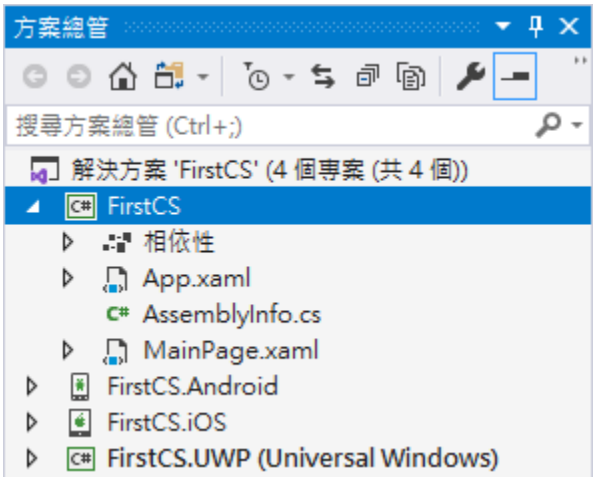
 MainPage.xaml.cs

```
namespace FirstCS.UWP
{
  public sealed partial class MainPage
  {
    public MainPage()
    {
      this.InitializeComponent();

      LoadApplication(new FirstCS.App());
    }
  }
}
```

在 MainPage.xaml.cs 的 MainPage 類別的建構函式內，將會產生一個 App 類別執行個體 (這個 App 類別是宣告在 Xamarin.Forms 專案內的一個類別，為了避免與 UWP 專案內的 App 類別相衝突，所以特別在 App 類別名稱前面加上了 FirstCS 命名空間來做區隔，等下會看到這個類別的介紹) ，使用敘述 `LoadApplication(new FirstCS.App());` 將整個執行控制權轉交到 Xamarin.Forms 專案內，也就是說，之後運行的程式碼都將會在 Xamarin.Forms 專案內來進行。

4.2.7 了解 Xamarin.Forms 專案的結構與運作方式

最後來看看 Xamarin.Forms 的共用專案 (也就是 Android , iOS , UWP 專案都會參考到的專案) 結構，當展開 Xamarin.Android 專案之後，將會呈現如下圖的畫面。



Xamarin.Forms 專案結構

對於上面觀察到的三個原生專案 Android , iOS , UWP ，將會歸納這三個專案的執行途徑

- Android 執行過程

  MainActivity 類別 > OnCreate 方法 > 建立 Xamarin.Forms 專案內的 App 執行個體 > 呼叫 LoadApplication 方法，將執行權限交由 Xamarin.Forms 的 App 類別的執行個體物件

- iOS

  AppDelegate 類別 > FinishedLaunching 方法 > 建立 Xamarin.Forms 專案內的 App 執行個體 > 呼叫 LoadApplication 方法，將執行權限交由 Xamarin.Forms 的 App 類別的執行個體物件

- UWP

  UWP 專案內的 App 類別 > MainPage 類別 > MainPage 建構函式 > 建立 Xamarin.Forms 專案內的 App 執行個體 > 呼叫 LoadApplication 方法，將執行權限交由 Xamarin.Forms 的 App 類別的執行個體物件

現在可以來觀察在 Xamarin.Forms 核心專案內的 App 類別，這個類別是定義在 App.xaml 這個 XAML 檔案內，在這個檔案節點，點選三角形圖示，就會看到 App.xaml.cs 這個節點，請打開這個節點，將會看到底下的程式碼

Xamarin.Forms 核心專案內的 App.xaml.cs

```
namespace FirstCS
{
   public partial class App : Application
   {
      public App()
      {
         InitializeComponent();

         MainPage = new MainPage();
      }

      protected override void OnStart()
      {
         // Handle when your app starts
      }

      protected override void OnSleep()
      {
         // Handle when your app sleeps
      }

      protected override void OnResume()
      {
         // Handle when your app resumes
      }
   }
}
```

對於 App.xaml 這個 XAML 檔案，將可以用來宣告整個 Xamarin.Forms 應用程式全域使用的 XAML 資源 Resource 與樣式 Style，這裡將會使用 XAML 標記語言來進行設計。而 App.xaml.cs 則是這個 XAML 檔案的 Code Behind 程式碼，在這裡的建構函式程式碼將指定整個 Xamarin.Forms App 所要顯示的第一個頁面，從這裡可以從 C# 敘述 `MainPage = new MainPage();` 看出來，這個 Xamarin.Forms App 第一個要顯示的頁面將會是 MainPage 這個頁面物件。

對於這個 Xamarin.Forms 的 App 類別，除了擁有 MainPage 這個屬性，可以用來指定整個應用程式會看到的第一個頁面，該類別也提供了 Xamarin.Forms 應用程式的生命週期管理上的相關事件，例如 OnStart() , OnSleep() ,OnResume() 這三個方法，將會於 Xamarin.Forms 應用程式啟動時候、進入到背景模式 (也就是螢幕看不到這個應用程式的情境)、從背景模式回到前景模式下，將會觸發這些方法，當然，程式設計師便可以在這些方法內進行更多的應用程式控制方面的設計工作。

最後，這個應用程式將會顯示出 MainPage 這個頁面，底下將會是這個應用程式執行的結果畫面

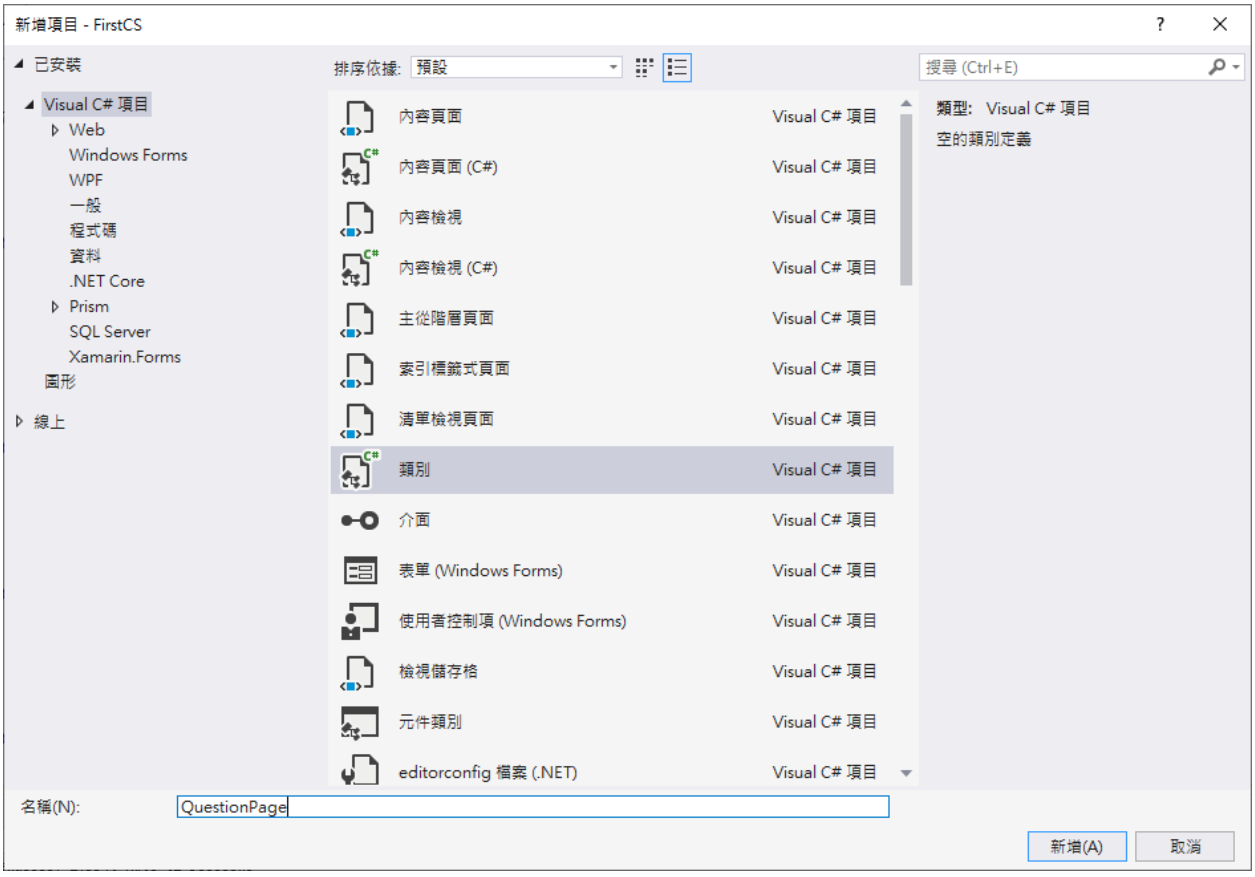
預設 Xamarin.Forms 專案樣板執行結果

4.3 開始僅使用 C# 程式碼來設計兩數相加的遊戲

現在來體驗如何僅單純使用 C# 程式碼，就可以完成一個 Xamarin.Forms 頁面的設計，而且這個頁面將會於應用程式啟動的時候，顯示出一個具有兩數相加的遊戲功能應用程式。

4.3.1 建立一個新遊戲頁面類別

- 請滑鼠右擊 FirstCS 核心專案
- 從彈出功能表中，點選 [加入] > [類別] 選項
- 現在將會看到 [新增項目 - FirstCS] 這個對話窗出現
- 確認中間清單中已經選擇了 [類別] 這個項目
- 請在下方 [名稱] 欄位中，輸入 QuestionPage
- 點選右下方的 [新增] 按鈕，新增這個類別檔案項目



在 Xamarin.Forms 核心專案內新增一個類別檔案

4.3.2 讓新增類別繼承 ContentPage

若你對於微軟的 GUI 類型的應用程式有開發經驗，則相信對於底下的程式碼作法將不會很陌生

對於剛剛建立產生的 QuestionPage 類別，請使用底下程式碼進行修正


QuestionPage.cs

```
public class QuestionPage : ContentPage
{
}
```

何謂內容頁面 ContentPage

在 Xamarin.Forms 中，內容頁面將會佔據整個螢幕，並且可以在這個內容頁面內透過不同的版面配置 Layout 與各種視覺控制項目，讓使用者可以看到所要顯示的各種內容

首先請將這裡類別前面加上 public 存取修飾詞，接著在這裡將會讓 QuestionPage 繼承 Xamarin.Forms 提供的 ContentPage，這是因為若想要在 Xamarin.Forms 設計一個新頁面，需要繼承 ContentPage 這個類別，該類別提供了許多功能可以讓程式設計師在這個頁面內加入許多的版面配置 Layout 與 UI 控制項，如此，就會在裝置螢幕上顯示出相關畫面內容。

現在，這個 QuestionPage 已經具備了可以開始加入更多版面配置與 UI 控制項的能力了。

4.3.3 設計該遊戲頁面

在這個遊戲頁面中，將會用到 4 個文字控制項 (Label)、1個文字輸入盒 (Entry)、與兩個按鈕 (Button)，所以，請在該類別內加入這些欄位宣告與定義。

- 文字控制項 Label

  在 Xamarin.Forms 中，若想要在螢幕上顯示文字字串內容，需要透過 Label 這個 UI 控制項，透過這個控制項的 Text 屬性來指定要顯示文字內容，這樣就可以顯示在裝置螢幕上了；當然，該控制項還提供了許多其他功能屬性可以使用，例如：可以指定字體大小、字體顏色、背景顏色等等。

- 文字輸入盒 Entry

  當想要讓這個應用程式使用者可以輸入相關資訊到這個應用程式內，可以使用文字輸入盒 Entry 這個 UI 控制項，使用者可以點選這個控制項，並且使用軟體鍵盤來輸入相關的文字資訊；當然，該控制項也供了許多其他功能屬性可以使用，例如：可以指定字體大小、字體顏色、背景顏色等等，另外，還可以使用 Keyboard 這個屬性來指定當時輸入所要顯示的鍵盤模式，由於在這裡要使用者輸入兩個整數相加的結果，因此，將會指定當使用點選這個控制項之後，將會顯示數值模式的軟體鍵盤在裝置螢幕上。

- 按鈕 Button

  按鈕在行動裝置應用程式開發中，是個相當重要的 UI 控制項，它可以讓使用點選這個控制項，而按鈕 Button UI 控制項將會提供一個點選事件可以用來訂閱當使用者點選這個按鈕的時候，可以觸發、執行這個訂閱事件的委派方法；在這個遊戲中，將會提供兩個按鈕，第一個按鈕將會讓應用程式重新產生一個新的數值相加的題目，而第二個按鈕將會是用來檢查使用者輸入的答案是否正確，若使用者輸入的答案不正確，將會在螢幕上顯示出答錯了文字內容，若使用者答對的話，將會顯示答對了文字內容，這個文字內容將會使用 Label 文字控制項來顯示在螢幕上，並且將會設定這個文字以橘色、字體 30 大小的狀態顯示在螢幕上。


QuestionPage.cs

```
int value1 = 0;
int value2 = 0;

Label lbValue1 = new Label();
Label lbValue2 = new Label();
Label lbQuestion = new Label();
Entry entAnswer = new Entry();
Button btnReQuestion = new Button();
Button btnSubmit = new Button();
Label lbMessage = new Label();
```

在這裡也宣告了兩個整數欄位，分別是 value1 , value2，這兩個欄位將會儲存此次猜謎題目的兩個相加的數值。

對於這個遊戲，將會使用底下的 CreateQuestion 方法，來產生出相關題目，而產生出來的題目，將會儲存來 value1 , value2 這兩個欄位內。


QuestionPage.cs

```
void CreateQuestion()
{
    Random random = new Random();
    value1 = random.Next(10, 99);
    value2 = random.Next(10, 99);

    lbValue1.Text = $"{value1}";
    lbValue2.Text = $"{value2}";

    lbQuestion.Text = $" 請問 {value1} + {value2} = 多少?";
    lbMessage.Text = "";
}
```

CreateQuestion 方法內將會使用 Random 類別，隨機產生出兩個數值，接著將會把這兩個數值分別使用 lbValue1.Text 與 lbValue2.Text 這兩個屬性，設定到這兩個文字控制項內，這樣，在螢幕上就會看到此次產生問題的兩個整數值；另外，將會利用 lbQuestion.Text 這個控制項，顯示出一段文字，讓使用這清楚的知道這個遊戲詢問的問題是什麼？最後，會把 lbMessage.Text 這個文字控制項的文字屬性值設定為空字串，也就是把使用者是否回答題目正確與否的文字訊息暫時清除掉。

現在，將會需要建立一個 QuestionPage 建構函式，因為將會需要在這個建構函式內設定這個頁面上所有顯示的 UI 控制項的位置與順序，以滿足這個遊戲畫面的設計


QuestionPage.cs 的建構函式

```
CreateQuestion();

lbValue1.FontSize = 20;
lbValue1.TextColor = Color.Blue;
lbValue2.FontSize = 20;
lbValue2.TextColor = Color.Blue;

lbQuestion.FontSize = 20;
lbQuestion.TextColor = Color.Red;
lbQuestion.Text = $" 請問 {value1} + {value2} = 多少?";

entAnswer.Keyboard = Keyboard.Numeric;

lbMessage.FontSize = 30;
lbMessage.TextColor = Color.Orange;
```

在這個建構函式一開始將會呼叫 CreateQuestion()，以便產生出新的遊戲問題題目，接著將會需要分別設定這些文字控制項 Label 與文字輸入盒 Entry 控制項要顯示在螢幕上的控制屬性，這裡將會設定要顯示文字的大小與顏色，另外，針對文字輸入盒 UI 控制項，將會使用 `entAnswer.Keyboard = Keyboard.Numeric` 這個敘述來設定該文字輸入盒 UI 控制項，需要使用數值類型的鍵盤，來讓使用者輸入內容。

該建構函式接下來將會需要進行按鈕控制項的相關設計工作，程式碼如下所示：


QuestionPage.cs 的建構函式

```
btnReQuestion.Text = " 產生問題";
btnReQuestion.Clicked += (s, e) =>
{
   CreateQuestion();
};

btnSubmit.Text = " 提交";
btnSubmit.Clicked += (s, e) =>
{
   int sum = int.Parse(entAnswer.Text);
   if ((value1 + value2) == sum)
   {
      lbMessage.Text = " 答對了";
   }
   else
   {
      lbMessage.Text = " 答錯了";
   }
};
```

在此先使用 btnReQuestion.Text 來設定重新產生題目的按鈕上面的名稱，接著使用 btnReQuestion.Clicked 事件來訂閱一個 Lambda 匿名委派方法，在這個委派方法內將會呼叫 CreateQuestion() 方法，如此，就會產生出新的題目，而因為 value1 與 value2 這兩個數值也有所變動，因此，螢幕上也會顯示出這個新題目的相關提示文字。

對於使用者要確認是否答對題目的按鈕，將會使用 btnSubmit.Text 這個屬性來設定該按鈕的顯示文字，只要使用這點選了這個按鈕，將會觸發這裡 btnSubmit.Clicked 所訂閱的委派事件，也就是這裡所指定 Lambda 匿名委派方法；在該委派方法將會首先透過 entAnswer.Text 屬性取得使用者輸入的答案內容，並且將其轉換成為一個整數 (在這裡將為了要簡化說明整個遊戲應用程式的開發與設計過程，將沒有做相關例外情況的檢查，例如，使用者沒有輸入任何答案或者輸入的內容不是一個整數這樣的情況，因此，若使用者沒有輸入任何答案或者輸入不是整數的文字，將會造成這個應用程式崩潰，造成 App 閃退的現象)。

當取得使用者輸入的答案，就會與原先題目，也就是 value1 + value2 的結果進行比對，

看看是否得到相同的數值，若兩者不相同，將會設定 lbMessage.Text 為答錯了這樣的文字內容，此時，螢幕上也會產生顯示出這個文字；當然，當使用者回答是正確的答案，螢幕上將會出現答對了這樣的文字。

相關的商業處理邏輯已經都開發與設計完成了，現在就需要針對螢幕畫面上要顯示那些 UI 控制項以及在顯示在那些位置和順序來進行設計，這裡將會使用底下的程式碼來完成。

QuestionPage.cs 的建構函式

```
Content = new StackLayout
{
   Children = {
      new Label { Text = " 數值1", FontSize=14 },
      lbValue1,
      new Label { Text = " 數值2", FontSize=14 },
      lbValue2,
      new Label { Text = " 問題", FontSize=14 },
      lbQuestion,
      new Label { Text = " 答案", FontSize=14 },
      entAnswer,
      new StackLayout
      {
         Orientation = StackOrientation.Horizontal,
         Children = {
            btnReQuestion,
            btnSubmit
         }
      },
      lbMessage
   }
};
```

對於要在 ContentPage 這個內容頁面上顯示畫面，其 ContentPage 有個屬性 Content 可以使用，只要將相關的 UI 控制項設定到這個 Content (內容屬性)，就會在螢幕上顯示出

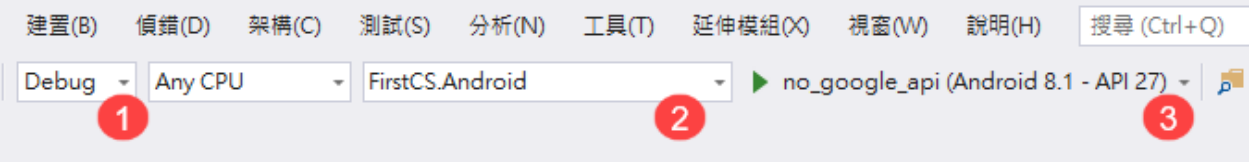
這個控制項；不過，對於 Content 這個屬性，僅能夠指定一個 UI 控制項，因此，在此需要把多個 UI 控制項設定到一個版面配置檢視 (Layout View) 內。

在 Xamarin.Forms 內共提供三種類型的使用者顯示內容，第一種是頁面 Page (這裡會有內容頁面、導航頁面、導航抽屜頁面、旋轉木馬頁面等等) ，第二個是版面配置 Layout (常用的版面配置有格子版面配置 Grid、推疊版面配置 StackLayout、捲動式版面配是 ScrollView 等等)，第三種則是豐富的 UI 控制項 (文字標籤、文字輸入盒、多行文字輸入盒、按鈕、矩形區塊、日期選取器、時間選取器等等)。

在此，將會使用一個 StackLayout 這個版面配置，將要顯示的 UI 控制項放到這個版面配置的 Children 屬性內，這樣，這些控制項將會使用預設垂直排列的方式來顯示在螢幕上；而對於要顯示的兩個按鈕，在這裡將會另外建立一個 StackLayout 版面配置物件，指定 StackLayout.Orientation 屬性值為 StackOrientation.Horizontal，表示等下要加入到這個推疊版面配置內的兩個按鈕，將會採用水平的方式來排列並顯示到螢幕上。

將過這樣的設計，若想要馬上執行這個專案，請使用底下的方式：

- 在標號1 位置，其為 [方案組態]，請點選為 [Debug]
- 在標號2 位置，其為 [起始專案]，請選擇使用 [FirstCS.Android] 項目，要使用 Android 平台來進行執行與測試
- 在標號3 位置，其為可用的 Android 模擬器或者實體裝置，選擇想要執行的設備
- 最後，點選標號 3 前方的綠色按鈕，就可以使用 Android 平台的方式，來執行這個 Xamarin.Forms 專案



Visual Studio 2019 工具列面板 - 執行 Xamarin.Forms 專案

當然，很不幸的，整個模擬器畫面將不會出現剛剛所設計的畫面內容，那麼，問題到底出現在哪裡呢？

4.3.4 變更 Xamarin.Forms 應用程式的起始頁面

還記得在最前面介紹 Xamarin.Forms 核心專案內的 App.xaml.cs 檔案的時候，在這個 App 類別內的建構函式內，其中有段 C# 敘述 `MainPage = new MainPage();` ，這裡是要用來指定當 Xamarin.Forms 應用程式啟動的時候，第一個要顯示的頁面物件。

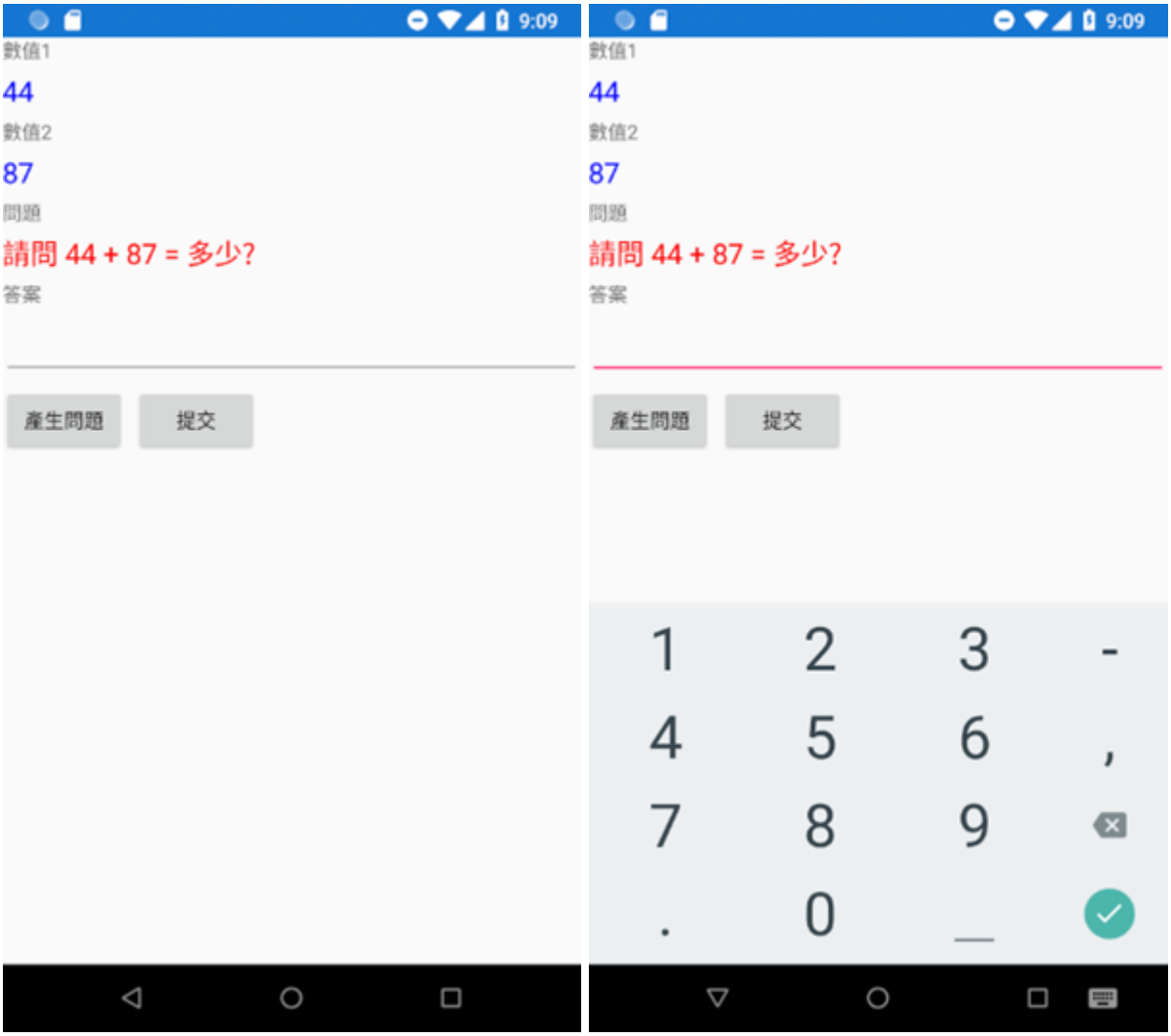
由於在這裡設計了一個 QuestionPage 類別物件，這將會是這個 Xamarin.Forms 應用程式所要顯示出來的第一個頁面，也是剛剛所設計的遊戲頁面。

App.xaml.cs

```
public partial class App : Application
{
   public App()
   {
      InitializeComponent();

      MainPage = new QuestionPage();
   }
   ...
}
```

因此，請在 App 類別的建構還是內，將 MainPage 變更成為 QuestionPage 這個頁面類別名稱。

現在，請重新執行這個專案



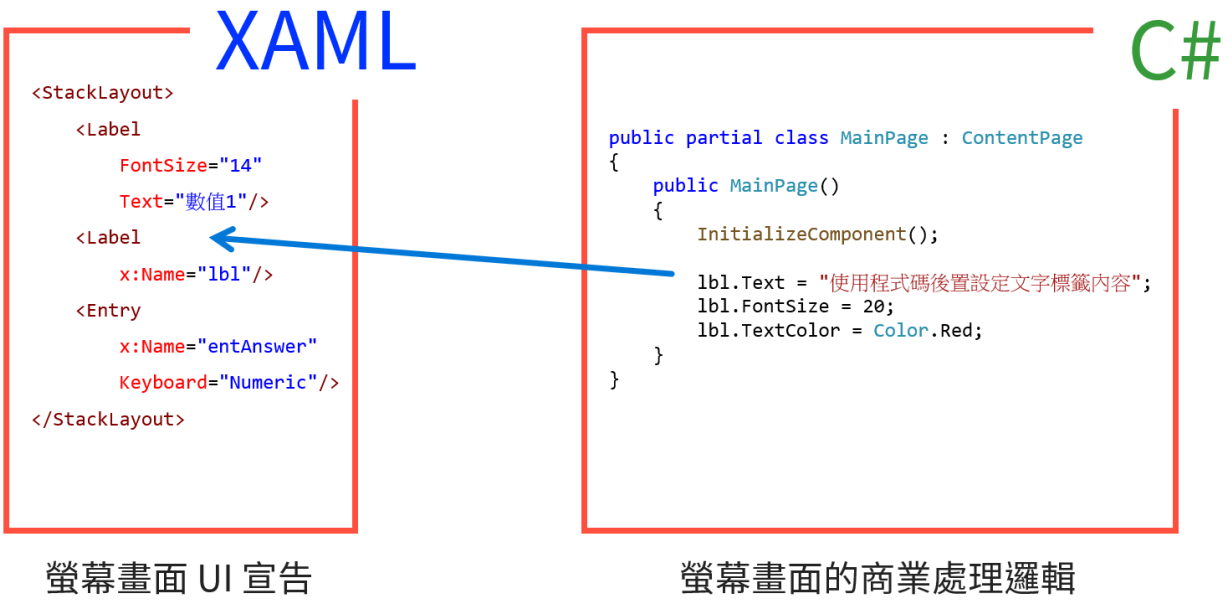
當應用程式一啟動之後，將會顯示如左上圖的畫面，其中數值1 / 數值2 的藍色數值，將會隨機產生出來的，這裡所看到的畫面就是剛剛使用 C# 程式語言所設計出來的 QuestionPage 類別所設計的結果。弱點選答案的文字輸入盒欄位，將會出現如右上圖的畫面，此時螢幕上將會顯示出軟體鍵盤，而且這個鍵盤是使用數值模式的鍵盤，方便使用者直接輸入兩數相加的結果數值到 App 內。

現在可以在文字輸入盒輸入 131 這個答案值，接著點選下方的提交按鈕，這樣，在按鈕的下方將會出現答對了這個文字內容；接著來輸入一個錯誤的答案，在這裡輸入 111 這個數值，接著點選提交按鈕，因為答案不正確，所以，將會在螢幕上看到答錯了這個文字內容。

5. 使用 XAML 標記宣告語言來開發 Xamarin.Forms App

在上一章已經說明如何僅透過 C# 程式碼來進行 Xamarin.Forms 的專案開發，也了解到整個 App 是如何從原生專案到 Xamarin.Forms 核心專案的運作流程，然而，在 Xamarin.Forms UI 工具集中提供了另外一個相當好用的工具，那就是 XAML ；XAML 是 eXtensible Application Markup Language 延伸應用程式標記語言的縮寫，其讓開發人員可以在 Xamarin.Forms 應用程式中使用標記定義使用者介面，而不是使用程式碼，這樣設計方式的好處就如何在設計網頁畫面的時候，會使用 HTML 標記語言來進行網頁畫面的設計，而相關的商業邏輯將會使用 JavaScript 程式語言來進行設計。

下圖表達了當使用 XAML 宣告標記語言與程式碼後置 Code Behind 關係



XAML 與程式碼後置 Code Behind

當在使用 XAML 進行應用程式頁面畫面的時候，僅需要使用 XAML 來宣告該頁面要使用的 UI 控制項和需要放在那些版面配置內，一旦透過 XAML 來描述出該頁面要顯示的內容，因為其呈現出一個階層關係架構，可以很容易地了解未來在實際裝置上要呈現出來的樣貌，而且十分容易針對所顯示出來螢幕畫面內容進行除錯；而在 XAML 標記語言中是無法使用相關程式設計邏輯處理功能，所以，XAML 將是很單純的扮演著在該頁面上該顯示出哪些使用者控制項的宣告角色，這比起單純僅使用 C# 程式語言來開發 Xamarin.Forms 應用程式，可以說大幅提升整體的開發速度、效能與品質；這就像是若要開發出一個網頁方面的應用程式，若僅是單純使用 JavaScript 不使用 HTML 來進行設計，這樣的設計過程將會比較不靈活而且不易除錯。

若想要在程式碼後置區塊的 C# 程式碼來存取畫面上的 UI 控制項，可以透過延伸標記 `x:Name` 語法來建立一個類別欄位變數，透過這個欄位變數就可以設定這個視覺控制項的相關屬性與設定其表現行為。

XAML 的好處

XAML 是個宣告式的標記語言，可以用來宣告螢幕上要顯示的各種內容與控制項，這可以讓 UI 設計與程式邏輯設計切割開來，提升整體應用程式的開發效能與品質

XAML 特別適合搭配熱門的 MVVM Model-View-ViewModel 模型-檢視-檢視模型應用程式架構使用：XAML 會定義透過以 XAML 為基礎的資料系結連結至 ViewModel 程式碼的 View，關於這個部分將會於後面的章節進行介紹。

因此，在這裡將僅會使用 XAML 這個標記宣告語言來進行相同兩數相加的遊戲畫面設計，而相關的遊戲運作商業邏輯，將會採用傳統的 Windows Forms 的 Code Behind 程式碼後置的設計方式來使用 C# 程式語言進行設計這些商業邏輯。

5.1 建立一個 Xamarin.Forms 方案

首先要先來建立一個使用 XAML 方式來設計的 Xamarin.Forms 專案

- 請啟動 Visual Studio 2019
- 在啟動後的對話窗內，選擇右下方的 [建立新的專案] 選項
- 當顯示出 [建立新專案] 對話窗，請在中間上方的文字輸入盒內，輸入 Xamarin 這個關鍵字
- 此時將會在中間清單區域，找到 [行動應用程式 (Xamarin.Forms)] 這個選項，請點選這個項目
- 在 [設定新的專案] 對話窗內，請在專案名稱欄位內，輸入 FirstXAML
- 請點選右下方的 [建立] 按鈕
- 現在將會跳出一個 [New Cross Platform App - FirstXAML] 對話窗
- 請選擇選取範本區域內的 [空白] 項目
- 在下方的 [平台] 區域，請記得勾選 Android , iOS , Windows (UWP) 這三個檢查盒 Checkbox

- 點選右下方的 [OK] 按鈕
- 開始建立 Xamarin.Forms 開發方案

5.2 了解 XAML 檔案的運作方式

在這個建立好的 Xamarin.Forms 方案，對於 Xamarin.Forms 使用的核心專案已經預設建立起一個 MainPage.xaml 這個檔案，因此，對於專案將不再建立一個新的 XAML 頁面檔案，將沿用 MainPage.xaml 這個檔案。

首先先不要急於開始進行任何相關的設計工作，先來檢視這個 XAML 檔案與其 Code Behind 檔案的內容，首先，使用滑鼠雙擊核心專案內的 MainPage.xaml 這個檔案節點，此時 Visual Studio 2019 將會顯示這個 XAML 內容，其 XAML 標記語言的內容如下。

在 XAML 檔案內，其本身就是一個 XML 語法格式，也就是說，要遵守 XML 的使用規範；每個 XAML 檔案必須僅能夠指定的根標籤 (Root Tag) 或者說根項目 (Root Element)，通常來說，需要指定一個 Xamarin.Forms 中提供的頁面項目，例如：ContentPage 內容頁面、MasterDetailPage 抽屜頁面、NavigationPage 導航頁面、TabbedPage 多標籤頁面、CarouselPage 旋轉木馬頁面；當然，也有可能指定 ContnetView 來指定這個 XAML 檔案為一個使用者控制項或者是 DataTemplate 用來宣告要顯示的檢視樣板。在 Xamarin.Forms 的每個 XAML 檔案，絕大多數都是一個 ContentPage 內容頁面，因為需要設計這個 App 在螢幕上出現的 UI 內容。

在 XAML 檔案的第一行是一個 XML 的宣告，通常會出現在 XML 文件的第一行。接下來則是指定這個 XAML 檔案要宣告的第一個根項目，在這個看到了使用了 `<ContentPage ...>` 這樣的標籤來宣告這個 XAML 檔案的根項目為一個內容頁面，在這個標籤內使用許多屬性來進行設定這個內容頁面的各種狀態，這就像是在使用 C# 程式語言來建立一個型別的執行個體，接著使用該型別提供各種屬性來進行設定該執行個體的狀態相同的操作過程。

第一個屬性為 xmlns 這個屬性值是用來宣告這個 XAML 檔案內預設使用的命空間，在這個 xmlns 之後要指定一個 URI ，在這裡使用的是 `"http://xamarin.com/schemas/2014/forms"`

來指定這個 XAML 文件可以使用各種標籤與支援的功能；這裡的 URI 僅是一個文字來標明 XAML 文件的版本，並不一定需要存在於網路上會有這個網頁存在，因此，同樣使用 XAML 標記宣告語言的開發框架 WPF / UWP，他們所使用的 URI 則會有不同的內容。這樣的宣告需要在 XAML 文件中的根項目來宣告，否則會有問題。

第二個屬性使用 xmlns:x ，這裡是指定一個新增加的自訂的命名空間，而這裡將會指向 XAML 提供的內建標記延伸 (Markup Extension) 功能，例如：可以在底下的 XAML 文件中，在某些標籤內使用 `x:Name=" 名稱"` 的方式來指定 Code Behind 要存取的變數名稱。這樣的作法將會將是宣告這個 XAML 文件，可以使用 x 這個前置詞 Prefix 來指向這個新建立的命名空間，使用這個命名空間所提供的相關功能。

第三個屬性宣告一個前置詞 d 的命名空間，xmlns:d ，而第四個屬性使用了前置詞 mc 的命名空間，xmlns:mc，這兩個新增加的命名空間通常使用到的機會不大，前者將會用於設計時期可以使用的相關功能，後者將是標記相容性語言功能。

接下來的 `mc:Ignorable="d"` 指定 XAML 處理器 XML 可以忽略標記檔案中所遇到的命名空間前置詞，在這裡指定了 d 這個命名空間，也就是在建置時期，將會忽略掉該 XAML 文件中所有遇到 d 命名空間的宣告。

原則上，剛剛所提到的內容是每個新建立的 XAML 文件檔案在根項目上都會出現的宣告內容，開發者並不需要特別去深入瞭解這些功能，也可以輕鬆地使用 XAML 文件宣告每個 App 頁面要顯示的各種 UI 內容。

最後一個是 `x:Class="FirstXAML.MainPage"` ，這裡使用到 x: 這個剛剛宣告的新增命名空間，也就是指向 XAML 的延伸功能之命名空間內的 Class 屬性，這是用來指定 Code Behind 程式碼後置之部分類別 (Partial Class) 關係，這個部分會在等下看到 Code Behind 程式碼後置的內容之後，就會更佳的清楚。

然而在進行 XAML 文件設計這些螢幕上要出現 UI 的時候，往往需要參考一些自己開發的類別或者像是數值轉換器甚至自己設計的其他 XAML 文件，甚至要參考所安裝的第三方套件內的 XAML 項目，這個時候，就需要在這個地方來自行增加額外的命名空間

與指定一個新的前置詞 Prefix，如此，透過新產生的命名空間就可以參考與使用這些非 Xamarin.Forms 預設提供的功能與 API 了。

了解完根項目中的相關屬性設定，接下來要來看到這個內容頁面內的其他 UI 宣告，在 XAML 中，透過使用不同的標籤/項目的宣告，將會形成一個階層式的關係，也就是說標籤可以擁有一個或者一個以上的子標籤 (是否可以擁有多個子標籤，要看當時的標籤是否有支援這樣的功能)，但是，每個標籤僅能夠指向一個父標籤。

從上一章的內容可以了解到，對於 ContentPage 內容頁面僅能有指定一個標籤，而且要使用 Content 屬性來設定，可是從上面的 XAML 宣告卻看不到這個 Content 屬性存在；而且，對於 StackLayout 堆疊版面配置 (對於版面配置這樣的 XAML 項目而言，可以是為一個容器，通常一個版面配置 Layout 內可以指定多個 XAML 標籤，根據版面配置的名稱不同，而會採用不同的方式來進行配置該版面配置內的 XAML 標籤要如何顯示在螢幕上，這包括了排列方式、顯示位置、自動計算要使用螢幕的空間大小，這些部分都會由 XAML 系統自行計算出來，Xamarin.Forms 程式設計師不需要特別去了這些運作方式)，將會使用 Children 這個屬性來將其指定的多個 XAML 項目 (也許是 UI 控制項或者是版面配置項目) 採用水平排列或者垂直排列的方式來顯示在螢幕上；可是，在上面的 XAML 文件上，卻也看不到任何 Children 這個屬性的宣告。

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             x:Class="FirstXAML.MainPage">

  <ContentPage.Content>
    <StackLayout>
      <StackLayout.Children>
        <!-- Place new controls here -->
        <Label Text="Welcome to Xamarin.Forms!"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />
```

```
        </StackLayout.Children>
    </StackLayout>
  </ContentPage.Content>

</ContentPage>
```

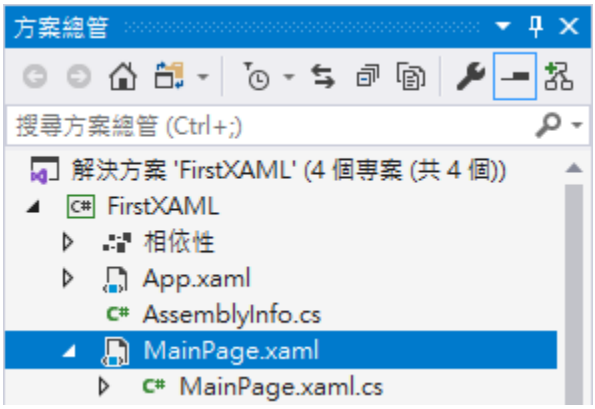
其實，不論是 ContentPage 的 Content 屬性，或者是 StackLayout 的 Children 屬性，通稱為是種 Content Property 內容屬性，對於 XAML 中的內容屬性，在進行使用 XAML 語言宣告的時候，是可以省略不寫的，也不會造成編譯與執行上的錯誤；因此，若要特別標示出這兩個內容屬性來進行 XAML 文件宣告，將會如上面 XAML 文件的寫法。這裡使用的 `ContentPage.Content` 與 `StackLayout.Children` 這樣的用法，在 XAML 中稱之為 Property Element 屬性項目用法，通常若要宣告一個屬性的內容，在 XAML 中僅能夠使用字串方式來進行宣告，然而，當要指定一個複雜的物件或者組合物件到一個項目的屬性內的時候，就需要使用屬性項目這樣的用法。

在這裡可以看到額外新增了兩個 `ContentPage.Content` 與 `StackLayout.Children` 內容屬性，來宣告這兩個屬性所擁有的子項目是甚麼？

內容屬性 Content Property

當看到的內容屬性名稱為 Content，則表示僅能夠指定一個標籤到這個 Content 屬性內，不過，當看到內容屬性的名稱為 Children 的時候，則表示可以指定一個以上的標籤到這個屬性內；若想要在 Content 內容屬性指定多的 XAML 項目時候，可以指定一個版面配置標籤到這個 Content 內容屬性內，並且將多個 XAML 屬性加入到這個版面配置內的內容屬性內。

接下來請在 MainPage.xaml 這個節點前面找的三角形圖示，使用滑鼠點擊這個三角形圖示，現在，可以在方案總管視窗內看到 MainPage.xaml.cs 這個檔案節點，使用滑鼠雙擊這個節點，就可以看到這個 MainPage.xaml.cs 程式碼內容，而這個 MainPage.xaml.cs 就是大家所稱作的 Code Behind 。



展開 MainPage.xaml 節點，就會看到 Code Behind 檔案

MainPage.xaml.cs

```
namespace FirstXAML
{
   // Learn more about making custom code visible in the Xamarin.Forms previewer
   // by visiting https://aka.ms/xamarinforms-previewer
   [DesignTimeVisible(false)]
   public partial class MainPage : ContentPage
   {
      public MainPage()
      {
         InitializeComponent();
      }
   }
}
```

這個 MainPage.xaml.cs 檔案，這是稱作為程式碼後置的地方，由於 XAML 這個文件檔案僅能夠宣告整個裝置螢幕要顯示出哪些 UI 控制項，並無法在 XAML 文件內撰寫任何程式碼來進行相關商業邏輯的設計 (所以，在這裡才會稱作在 XAML 進行設計的過程，是要宣告各種 XAML 項目，而不是要設計相關具有邏輯處理的程式碼)，所以，對於每個 XAML 頁面將會提供一個 [頁面名稱.xaml.cs] 的節點，可以在這裡撰寫相關 C# 的程式碼。

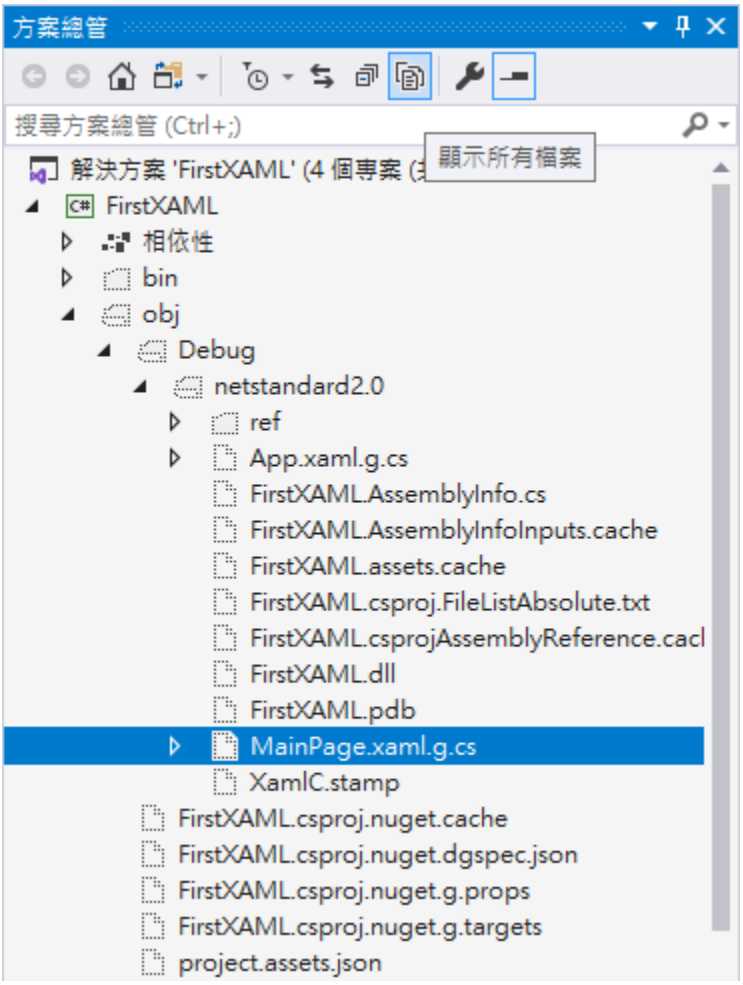
在這個 Main 頁面的 Code Behind 程式碼後置的原始碼中，可以看到這個 MainPage 是繼承了 ContentPage 內容頁面這個類別，並且這個類別也是一個部分類別；這也就表示了這個 MainPage 類別將會有一個同樣宣告為 `partial class MainPage` 的程式碼出現在這個專案內，在編譯時期的時候，就會將這兩個同名類別組合成為一個 MainPage 類別，可是，你無法在 Visual Studio 方案總管內很容易地找到這個檔案在哪裡。

另外，這個 MainPage 類別也會有一個建構函式，而且裡面將會呼叫 InitializeComponent() 這個方法，可是，在上面的類別定義中，也看不到這個方法定義在哪裡。

請勿移除或者註解 InitializeComponent() 這個方法呼叫

當在程式碼後置區塊看到這個頁面類別的建構函式內的 `InitializeComponent()` 方法，請不要把它註解或者移除掉，否則，將會造成這個頁面無法正常運作。

現在，請先建置 Xamarin.Forms 這個核心專案，也就是 FirstXAML



找到 MainPage.xaml 的編譯器產生的程式碼

建置完成之後，請在方案總管上點選 [顯示所有檔案] 的圖示，顯示核心專案建置後所產生的所有檔案內容，請展開這些資料夾 [obj] > [Debug] > [netstandard2.0] ，將會在這個資料夾內看到一個 [MainPage.xaml.g.cs] 節點，這個檔案是由編譯器在建置過程中產生出來的，也就是每個 XAML 文件檔案，都會產生出一個這個檔案；請打開這個 [MainPage.xaml.g.cs] 檔案。

MainPage.xaml.g.cs

```
//------------------------------------------------------------------------------
// <auto-generated>
//     這段程式碼是由工具產生的。
//     執行階段版本:4.0.30319.42000
//
//     對這個檔案所做的變更可能會造成錯誤的行為，而且如果重新產生程式碼，
//     變更將會遺失。
// </auto-generated>
//------------------------------------------------------------------------------

[assembly: global::Xamarin.Forms.Xaml.XamlResourceIdAttribute("FirstXAML.MainPage.xa\
ml", "MainPage.xaml", typeof(global::FirstXAML.MainPage))]

namespace FirstXAML {


    [global::Xamarin.Forms.Xaml.XamlFilePathAttribute("MainPage.xaml")]
    public partial class MainPage : global::Xamarin.Forms.ContentPage {

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private void InitializeComponent() {
            global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(MainPage\
));
        }
    }
}
```

從這個由編譯器產生的 C# 檔案中，看到了另外一個 `partial class MainPage` 的部分類別定義，也看到了 `InitializeComponent()` 方法定義，這裡就可以解決剛剛產生的疑惑，原來 InitializeComponent() 方法是由編譯器自動產生的，在這個方法內將會呼叫 `global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml` 方法，將剛剛宣告的 XAML 文件讀取出來。

對於這個編譯器產生的 MainPage 部分類別，可以從最上方的註解內容看到，這個檔案的內容不要去修改它，去修改這些內容是沒有用的，因為每次建置專案的時候，這個檔案都會重新產生一次，若對這個檔案修改任何程式碼，只要開始重建，這些修改後的程式碼

將會不見了。另外，LoadFromXaml 方法會讀取 XAML 文件，針對該文件中的所有標籤，例如：<ContentPage> , <StackLayout> , <Label> ，都會建立一個 CLR 的執行個體，這些執行個體產生之後，會根據 XAML 文件中各個屬性的宣告，進行這些執行個體的 .NET 屬性設定，這些過程就像是前一章中自行使用 C# 程式碼來設計頁面內容的程式碼相同。

不過，要如何存取 XAML 文件中的標籤執行個體，例如，想要設定文字標籤 Label 的 Text 屬性內容，或者設定該文字標籤的顏色與字體大小需求，這個時候就需要使用到 x: 前置詞 Prefix 所提供的 XAML 標記延伸擴充功能。

XAML 與編譯器

編譯器會產生相關程式碼，讓 XAML 宣告式標記語言檔案可以正常運作，所以，不要去修改編譯器產生的任何程式碼。

5.3 了解如何在 Code Behind 程式碼來存取 XAML 文件中的項目

要了要能夠讓程式碼後置的 C# 可以存取 XAML 中的相關項目 Element，請再度打開 MainPage.xaml 文件檔案，在 Label 標籤內加入 `x:Name="lbl"` 這個 XAML 標記延伸屬性宣告，一旦使用 x:Name 這個標記延伸屬性指定一個識別名稱，在程式碼後置的地方就可以使用 C# 程式碼來存取這個 lbl 指向的 Label 項目，進行呼叫該 Label 類別內的方法與存取 Label 類別內的屬性。

MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin.com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        x:Class="FirstXAML.MainPage">

  <ContentPage.Content>
    <StackLayout>
      <StackLayout.Children>
        <!-- Place new controls here -->
        <Label Text="Welcome to Xamarin.Forms!"
            x:Name="lbl"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />
      </StackLayout.Children>
    </StackLayout>
  </ContentPage.Content>

</ContentPage>
```

請再度打開 MainPage.xaml.cs 檔案，在這個 MainPage 建構函式內，使用 `lbl.Text = " 使用程式碼後置設定文字標籤內容";` 敘述來設定該文字標籤要顯示的內容與使用 `lbl.FontSize = 20;` 與 `lbl.TextColor = Color.Red;` 兩個敘述來指定該文字標籤的狀態，也就是設定該文字標籤的屬性值。

MainPage.xaml.cs

```
public partial class MainPage : ContentPage
{
   public MainPage()
   {
      InitializeComponent();

      lbl.Text = " 使用程式碼後置設定文字標籤內容";
      lbl.FontSize = 20;
      lbl.TextColor = Color.Red;
   }
}
```

C# 的建構函式

在 C# 程式語言中，建構函式將會扮演著將這個類別產生的執行個體進行各種成員的初始化工作。

為什麼現在可以從取這個 XAML 文件中的文字標籤項目了呢？請重新建置這個 FirstXAML 核心專案，接著請展開這些資料夾 [obj] > [Debug] > [netstandard2.0] ，同樣的將這個資料夾內的 [MainPage.xaml.g.cs] 節點再度打開來查看。

MainPage.xaml.g.cs

```
[global::Xamarin.Forms.Xaml.XamlFilePathAttribute("MainPage.xaml")]
public partial class MainPage : global::Xamarin.Forms.ContentPage {

    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tas\
ks.XamlG", "2.0.0.0")]
    private global::Xamarin.Forms.Label lbl;

    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tas\
ks.XamlG", "2.0.0.0")]
    private void InitializeComponent() {
        global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(MainPage));
        lbl = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.F\
orms.Label>(this, "lbl");
    }
}
```

現在看到這個 MainPage.xaml.g.cs 檔案內的程式碼與之前所看到有所不同，這裡宣告了一個 MainPage 類別欄位，使用 `private global::Xamarin.Forms.Label lbl;` 敘述來宣告，並且透過了 `global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.L "lbl");` 方法找出該 XAML 文件中的有使用 x:Name 標記延伸宣告識別名稱為 lbl 的項目，並且將該文字標籤執行個體物件設定到 lbl 這個類別欄位上。由於在這個編譯器產生的類別使用的是 C# 部分類別來進行設計，因此，在這編譯器產生的 MainPage 部分類別所宣告的任何欄位，都可以在該 XAML 頁面本身提供的程式碼後置的 MainPage 部分類別中來使用，因為，最終他們會編譯成為同一個 MainPage 類別。

5.4 使用 XAML 設計兩數相加的遊戲畫面

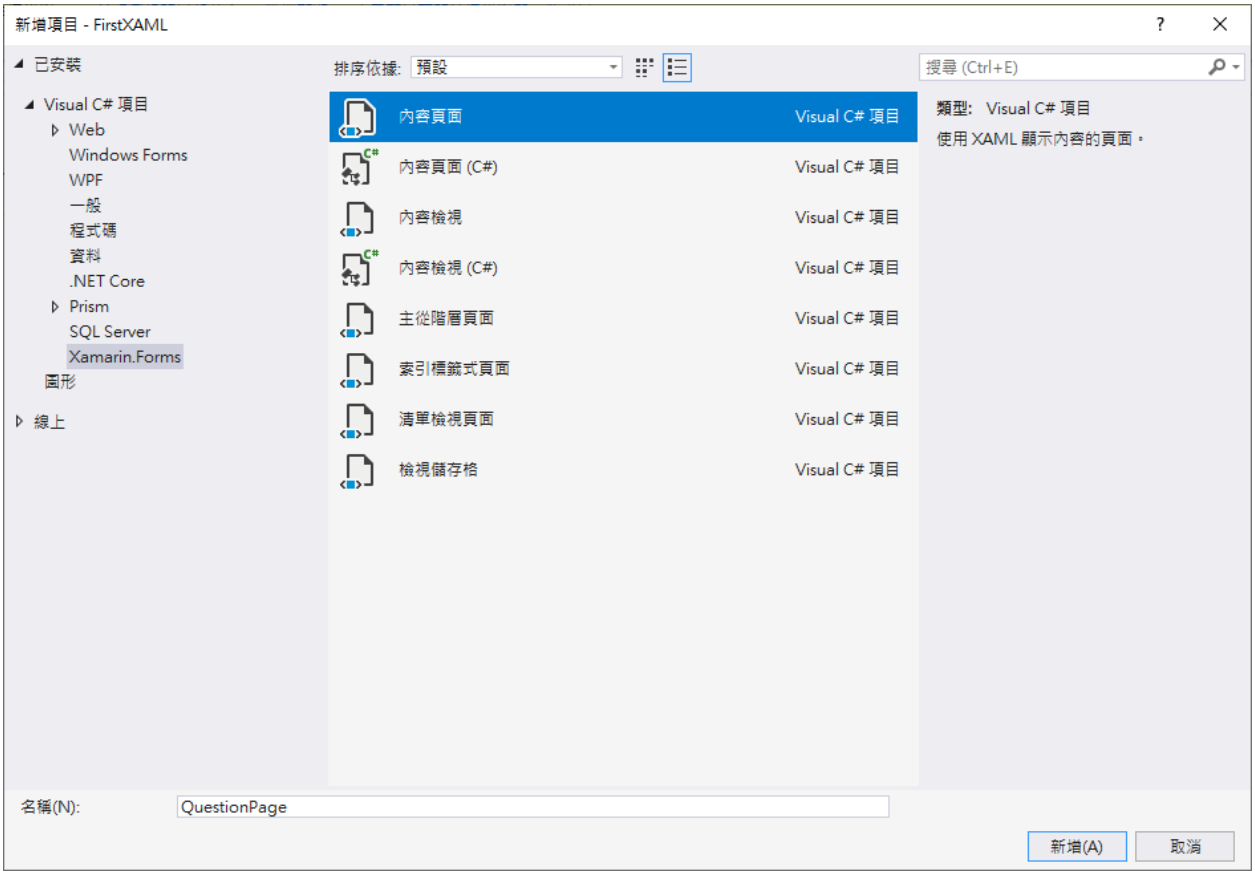
現在，終於要開始使用 XAML 文件來設計出上一章同樣功能與畫面內容的兩數相加的遊戲，不過在此將會使用 XAML 文件與程式碼後置來完成，而不是全部都使用 C# 程式碼來逐一產生每個類別的執行個體這樣的設定工作。

5.4.1 建立一個新遊戲 XAML 頁面

- 請滑鼠右擊 FirstXAML 核心專案
- 從彈出功能表中，點選 [加入] > [新增項目] 選項
- 現在將會看到 [新增項目 - FirstXAML] 這個對話窗出現
- 在對話窗左方點選 [Xamarin.Forms] 清單選項
- 確認中間清單中已經選擇了 [內容頁面] 這個項目

  > 請注意不要選擇 [內容項目 C#] 這個選項，否則，將會產生一個 C# 類別檔案，而沒有 XAML 文件檔案

- 請在下方 [名稱] 欄位中，輸入 QuestionPage
- 點選右下方的 [新增] 按鈕，新增這個類別檔案項目



建立一個新的 XAML 內容頁面

5.4.2 設計該遊戲頁面的 XAML 文件內容

現在可以開始來使用 XAML 文件來進行這個遊戲頁面的設計，請打開 QuesionPage.xaml 這個文件檔案，輸入底下的 XAML 標記宣告內容。

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             x:Class="FirstXAML.QuestionPage">

    <StackLayout>
        <Label/>
        <Label/>
        <Label/>
        <Label/>
        <Label/>
        <Label/>
        <Label/>
        <Entry/>
        <StackLayout
            Orientation="Horizontal">
            <Button/>
            <Button/>
        </StackLayout>
        <Label/>
    </StackLayout>

</ContentPage>
```

在這裡會先把這個頁面會使用到的相關 XAML 項目輸入在這個 QuestionPage.xaml XAML 文件檔案內，並且會使用適當的版面配置檢視 Layout View (可以視為一個 XAML 項目的容器 Container) 來進行這個版面配置容器裡面的 XAML 檢視的排列與自動計算要顯示的位置。另外，也針對兩個按鈕使用一個具有水平排列的 StackLayout 堆疊版面配置，依照需要來顯示這兩個按鈕。

由於使用 XAML 文件來進行整體的 App 的設計，因此，可以完全透過 XAML 文件來宣告這些要顯示在螢幕上的各種 XAML 項目，在這個時候完全不需要考量到任何的商業邏輯運算或者使用任何 C# 程式語言，對於各種商業邏輯部分，將會等下針對需要使用程式碼後置來控制的 XAML 項目，使用 `x:Name=" 識別名稱"` 這個標記延伸語法來標示在相對應的

XAML 項目上，這樣就可以程式碼後置中使用 C# 程式語言來存取這些 XAML 控制項。

現在，把剛剛設計好的 XAML 文件，針對每個 XAML 項目加入適當的屬性宣告以及 x:Name=" 識別名稱" 這個標記延伸語法。

QuestionPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             x:Class="FirstXAML.QuestionPage">

    <StackLayout>
        <Label
            FontSize="14"
            Text=" 數值1"/>
        <Label
            x:Name="lbValue1"
            FontSize="20"
            TextColor="Blue"/>
        <Label
            FontSize="14"
            Text=" 數值2"/>
        <Label
            x:Name="lbValue2"
            FontSize="20"
            TextColor="Blue"/>
        <Label
            FontSize="14"
            Text=" 問題"/>
        <Label
            x:Name="lbQuestion"
            FontSize="20"
            TextColor="Red"/>
        <Label
            FontSize="14"
```

```
      Text=" 答案"/>
    <Entry
      x:Name="entAnswer"
      Keyboard="Numeric"/>
    <StackLayout
      Orientation="Horizontal">
      <Button
        x:Name="btnReQuestion"
        Text=" 產生問題"/>
      <Button
        x:Name="btnSubmit"
        Text=" 提交"/>

    </StackLayout>
    <Label
      x:Name="lbMessage"
      FontSize="30"
      TextColor="Orange"/>
  </StackLayout>

</ContentPage>
```

經過把一開始設計的 XAML 文件進行加入相關屬性設定，例如在 Entry 項目中，使用 Keyboard 來設定這個文字輸入盒要使用與顯示的軟體鍵盤樣式、在不同的文字標籤內使用 FontSize 與 TextColor 屬性來宣告這個文字標籤要顯示的文字顏色與字體大小；對外，對於 Button 這兩個按鈕，其相對應的點擊觸發事件，可以宣告在 XAML 文件內，也可以使用程式碼後置來使用 C# 程式語言來進行綁定、訂閱事件的工作。對於需要在程式碼後置區域來存取的 XAML 項目，也使用 x:Name 這個標記延伸語法，標示在整個 XAML 文件適當的標籤上了。

現在，先來建置這個 FirstXAML 專案，看看由編譯器產生的 QuestionPage 頁面的部分類別程式碼，有些甚麼變化；請在建置完成之後，找到 QuestionPage.xaml.g.cs 這個檔案並且打開見

QuestionPage.xaml.g.cs

```
namespace FirstXAML {


    [global::Xamarin.Forms.Xaml.XamlFilePathAttribute("QuestionPage.xaml")]
    public partial class QuestionPage : global::Xamarin.Forms.ContentPage {

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Label lbValue1;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Label lbValue2;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Label lbQuestion;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Entry entAnswer;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Button btnReQuestion;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Button btnSubmit;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Label lbMessage;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build\
.Tasks.XamlG", "2.0.0.0")]
        private void InitializeComponent() {
            global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(Question\
Page));
            lbValue1 = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::\
Xamarin.Forms.Label>(this, "lbValue1");
            lbValue2 = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::\
Xamarin.Forms.Label>(this, "lbValue2");
```

```
            lbQuestion = global::Xamarin.Forms.NameScopeExtensions.FindByName<global\
::Xamarin.Forms.Label>(this, "lbQuestion");
            entAnswer = global::Xamarin.Forms.NameScopeExtensions.FindByName<global:\
:Xamarin.Forms.Entry>(this, "entAnswer");
            btnReQuestion = global::Xamarin.Forms.NameScopeExtensions.FindByName<glo\
bal::Xamarin.Forms.Button>(this, "btnReQuestion");
            btnSubmit = global::Xamarin.Forms.NameScopeExtensions.FindByName<global:\
:Xamarin.Forms.Button>(this, "btnSubmit");
            lbMessage = global::Xamarin.Forms.NameScopeExtensions.FindByName<global:\
:Xamarin.Forms.Label>(this, "lbMessage");
        }
    }
}
```

在這個 QuestionPage.xaml.g.cs 檔案內，將會看到更多的私有欄位的宣告，這是因為在這個 QuestionPage.xaml 的 XAML 文件中，使用了很多的 x:Name 標記延伸宣告。

現在請打開這個 QuestionPage.xaml.cs 頁面的程式碼後置檔案，先加入一個 CreateQuestion 方法到這個 QuestionPage 類別內

QuestionPage.xaml.cs

```
void CreateQuestion()
{
    Random random = new Random();
    value1 = random.Next(10, 99);
    value2 = random.Next(10, 99);

    lbValue1.Text = $"{value1}";
    lbValue2.Text = $"{value2}";

    lbQuestion.Text = $" 請問 {value1} + {value2} = 多少?";
    lbMessage.Text = "";
}
```

這個 CreateQuestion 方法與上一章單純使用 C# 程式語言來直接開發 Xamarin.Forms App 所設計的程式碼完全相同，這是因為在這個 XAML 文件中所宣告的相關 XAML 控制項的 x:Name 識別名稱，都與前一章所使用的變數名稱相同。

接著要來完成這個遊戲的其他商業邏輯設計程式碼，完成後的結果如下所示：

QuestionPage.xaml.cs

```
public partial class QuestionPage : ContentPage
{
   int value1 = 0;
   int value2 = 0;

   public QuestionPage()
   {
      InitializeComponent();
      CreateQuestion();
      btnReQuestion.Clicked += (s, e) =>
      {
         CreateQuestion();
      };
      btnSubmit.Clicked += (s, e) =>
      {
         int sum = int.Parse(entAnswer.Text);
         if ((value1 + value2) == sum)
         {
            lbMessage.Text = " 答對了";
         }
         else
         {
            lbMessage.Text = " 答錯了";
         }
      };
   }

   void CreateQuestion()
   { ...   }
}
```

首先，同樣的要宣告兩個欄位 value1, value2 用來記錄此次出的題目的兩個要相加的數值內容，接著，要在建構函式內先執行 CreateQuestion() 方法，建立新的猜謎題目內容，接著，要進行兩個按鈕物件的 Clicked 事件的綁定設計，這裡使用 Lambda 匿名委派方法來設計當按鈕按下之後，所要觸發與執行的程式碼，這裡的兩個按鈕的觸發執行程式碼，將會與前一章所設計的兩個按鈕程式碼邏輯相同。

將過這樣的設計，若想要馬上執行這個專案，請使用底下的方式：

- 在標號1 位置，其為 [方案組態]，請點選為 [Debug]
- 在標號2 位置，其為 [起始專案]，請選擇使用 [FirstXAML.Android] 項目，要使用 Android 平台來進行執行與測試
- 在標號3 位置，其為可用的 Android 模擬器或者實體裝置，選擇想要執行的設備
- 最後，點選標號 3 前方的綠色按鈕，就可以使用 Android 平台的方式，來執行這個 Xamarin.Forms 專案

Visual Studio 2019 工具列面板 - 執行 Xamarin.Forms 專案

當然，很不幸的，整個模擬器畫面將不會出現剛剛所設計的畫面內容，而會顯示出底下的畫面

執行結果的畫面

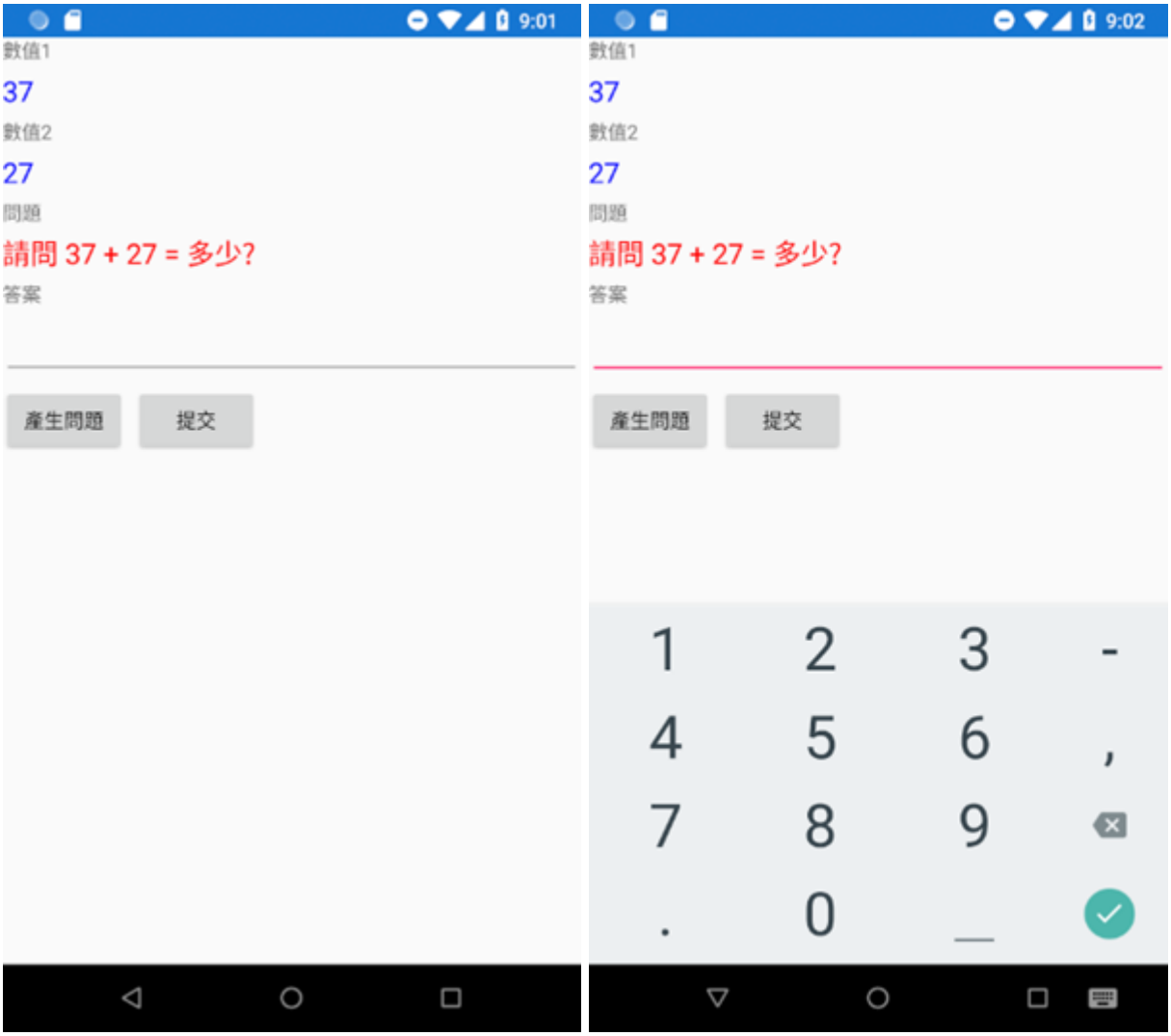
這個顯示來的畫面將會剛剛設計在 MainPage.xaml 這個 XAML 文件上與其程式碼後置所執行出來的結果。

5.4.3 變更 Xamarin.Forms 應用程式的起始頁面

回想一下想要指定 Xamarin.Forms 第一個出現的頁面，是要修正哪個地方法，現在需要指定 Xamarin.Forms 程式一啟動之後，需要先顯示 QuestionPage 這個頁面，所

以，請在 Xamarin.Forms 核心專案內的 App.xaml.cs 檔案，在這個 App 類別內的建構函式內，其中有段 C# 敘述 `MainPage = new MainPage();` ，將其修正為 `MainPage = new QuestionPage();` 這樣的敘述就可以解決此問題了。

現在，請重新執行這個專案



當應用程式一啟動之後，將會顯示如左上圖的畫面，其中數值1 / 數值2 的藍色數值，將會隨機產生出來的，這裡所看到的畫面就是剛剛使用 C# 程式語言所設計出來的 QuestionPage 類別所設計的結果。弱點選答案的文字輸入盒欄位，將會出現如右上圖的畫面，此時螢幕上將會顯示出軟體鍵盤，而且這個鍵盤是使用數值模式的鍵盤，方便使用

者直接輸入兩數相加的結果數值到 App 內。

現在可以在文字輸入盒輸入 64 這個答案值，接著點選下方的提交按鈕，這樣，在按鈕的下方將會出現答對了這個文字內容；接著來輸入一個錯誤的答案，在這裡輸入 67 這個數值，接著點選提交按鈕，因為答案不正確，所以，將會在螢幕上看到答錯了這個文字內容。

5.5 App.xaml 的應用

在使用 XAML 標記宣告語言來進行 Xamarin.Forms 應用程式開發的時候，可以適度將 UI 部分獨立在 XAML 文件檔案中來進行宣告，而因為 XAML 這個語言中並沒有任何的邏輯處理的程式碼可以來開發與設計，因此，就僅是很單純的做整個螢幕的 UI 宣告而已，對於該頁面上的商業處理邏輯，在這一章中將會使用每個 XAML 頁面都會有的 Code Behind 程式碼後置的機制來進行設計，在這裡就可以使用平常習慣撰寫的 C# 程式語言來解決這些商業邏輯上的相關需求。

然而當需要在程式碼後置的 C# 程式碼來存取 XAML 頁面上的 UI 項目的時候，就可以透過 XAML 中提供的 x:Name 標記延伸功能，設定一個識別名稱，編譯器就會在這個頁面類別中產生一個欄位變數，如此，C# 程式碼就可以透過這個欄位變數來控制要如何顯示這個 UI 控制項了。

而在 XAML 文件中宣告的相關項目，最終在建置階段，編譯器把每個 XAML 項目 Element 都產生出一個 .NET 執行個體出來，該執行個體所宣告的型別，就是該 XAML 項目的名稱，而這些 XAML 檢視相關型別，都可以在 Xamarin.Forms 的 NuGet 套件中找到。

最後，來看看當使用 XAML 來進行跨平台 App 設計的時候，另外一個好處。根據這章使用 XAML 所設計出來的遊戲 App 中，在 QuestionPage.xaml 文件檔案內，都會在不同 UI 控制項來設定不同的屬性，這樣可以讓整個 App 呈現出多樣性的樣貌，可是，好像許多的屬性設定似乎都有重複的宣告，讓整個 XAML 文件檔案呈現出比較複雜的文件內容。

在 XAML 這個生態內，可以使用樣式 Style 來集中管理相似 XAML 項目的屬性設定，只要符合條件的 XAML 項目，這些 XAML 項目將會直接套用這裡事先宣告好的相關屬性設定，如此，將會大幅簡化了 XAML 文件設計，而且當有要進行變更的時候，也可以在樣式宣告區段直接變更，如此，整個應用程式有用到這樣式的頁面，都會自動套用該樣式宣告的新屬性設定值。

那麼，對於整個 Xamarin.Forms 系統都會用到的樣式 Style 是要宣告在哪裡呢？這當然需

要宣告在 Xamarin.Forms 應用程式的進入點 Entry Point，而這個進入點就是 App.xaml 這個檔案，所以，請打開 FirstXAML 這個核心專案內的 App.xaml 檔案，將會呈現如下面的 XAML 文件。

App.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin.com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        x:Class="FirstXAML.App">
  <Application.Resources>

  </Application.Resources>
</Application>
```

在這個 App.xaml 檔案中可以看的出來，其根項目 Root Element 為 Application 這個標籤，而且這裡有使用到開放的 `<Application.Resources></Application.Resources>` 屬性項目 Property Element 標籤，不過，該屬性項目的是沒有任何宣告內容，這是因為這個地方是將是要提供 Xamarin.Forms 開發者來進行宣告相關樣式、資源的地方，而且，只要是在這裡進行宣告的 XAML 項目，整個 Xamarin.Forms 應用程式，不論是哪個頁面，都可以存取的到。

現在，請先依照底下的內容，在 `<Application.Resources></Application.Resources>` 屬性項目區段內，完成四個樣式 Style 的宣告。

App.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             x:Class="FirstXAML.App">
  <Application.Resources>
    <ResourceDictionary>
      <Style TargetType="Label">
        <Setter Property="FontSize" Value="14"/>
      </Style>
      <Style x:Key="QuestionLabel" TargetType="Label">
        <Setter Property="FontSize" Value="20"/>
        <Setter Property="TextColor" Value="Blue"/>
      </Style>
      <Style x:Key="QuestionTitleLabel" TargetType="Label">
        <Setter Property="FontSize" Value="20"/>
        <Setter Property="TextColor" Value="Red"/>
      </Style>
      <Style x:Key="MessageLabel" TargetType="Label">
        <Setter Property="FontSize" Value="30"/>
        <Setter Property="TextColor" Value="Orange"/>
      </Style>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

在 Xamarin.Forms 內可以使用 `<Style>` 這個標籤來宣告樣式，而樣式共有兩種類型，一種是隱含樣式 Implicit Styles，另外一種是明確樣式 Explicit Styles。在進行樣式宣告的時候，若沒有使用到這個 x:Key 標記延伸語法來宣告的鍵值 (這個鍵值將會於其他頁面中，可以透過這個鍵值來參考到這個樣式的宣告內容)，則這樣的樣式宣告則屬於明確樣式，反之，若在宣告樣式的時候有使用到 x:Key 標記延伸語法的時候，這樣的樣式則屬於隱含樣式。

不論隱含樣式或者明確樣式，都需要使用 `TargetType` 這個屬性值來指定這個樣式可以套用在哪種 XAML 項目上，在上面的 App.xaml 內容中，使用了 `TargetType="Label"` 這樣

的屬性宣告，表示這些樣式僅能夠套用在屬於 Label 文字標籤的項目上。

在每個樣式中，可以使用 `<Setter Property="FontSize" Value="20"/>` 這樣的語法來設定這個 XAML 項目中的某個屬性，其屬性值要設定為多少？以這個範例中，若該樣式被套用的話，則該文字標籤將的字體大小將會設定為 20。

所以，只要是以明確樣式 (沒有使用到 x:Key 標記延伸語法) 對特定 XAML 項目 (使用 TargetType 來指定要套用的項目) 進行宣告，只要是在該應用程式內的任何頁面內，有宣告使用這個 XAML 項目，此時該 XAML 項目的屬性預設值將會是該明確樣式中所宣告的屬性值；然而，若想要讓某個 XAML 項目來套用特定的隱含樣式 (沒有使用到 x:Key 標記延伸語法) ，則需要該隱含樣式內的 x:Key 設定值。

現在，將 QuestionPage.xaml 文件檔案打開

QuestionPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             x:Class="FirstXAML.QuestionPage">

  <StackLayout>
    <Label
      Text=" 數值1"/>
    <Label
      x:Name="lbValue1"
      Style="{StaticResource QuestionLabel}"/>
    <Label
      Text=" 數值2"/>
    <Label
      x:Name="lbValue2"
      Style="{StaticResource QuestionLabel}"/>
```

```
    <Label
      Text=" 問題"/>
    <Label
      x:Name="lbQuestion"
      Style="{StaticResource QuestionTitleLabel}"/>
    <Label
      Text=" 答案"/>
    <Entry
      x:Name="entAnswer"
      Keyboard="Numeric"/>
    <StackLayout
      Orientation="Horizontal">
      <Button
        x:Name="btnReQuestion"
        Text=" 產生問題"/>
      <Button
        x:Name="btnSubmit"
        Text=" 提交"/>

    </StackLayout>
    <Label
      x:Name="lbMessage"
      Style="{StaticResource MessageLabel}"/>
  </StackLayout>

</ContentPage>
```

請將 Label 標籤中移除掉已經在樣式內宣告的屬性設定內容，這樣是不是整個 XAML 文件已經清爽多了呢？對於需要使用隱含樣式來指定某個特定樣式來設定這個項目的預設屬性值，可以使用該 XAML 項目的 Style 這個屬性名稱來宣告。

例如，對於 `x:Name="lbMessage"` 這個文字標籤，將會使用這樣的語法 `Style="{StaticResource MessageLabel}"` 來宣告這個文字標籤，要套用 MessageLabel 這個隱含樣式宣告的內容，而 MessageLabel 樣式宣告那些屬性，可以在 App.xaml 這個 XAML 文件檔案中找的到。

現在，可以再度執行這個專案，將會發現所顯示的遊戲畫面，每個文字標籤的文字大小、顏色，都與沒有套用樣式前都是相同的，若下次要調整某個文字標籤大小或者顏色，僅需

要在 App.xaml 內找到相關樣式宣告，直接修改該樣式宣告內容，這樣，整個 XAML 項目顯示的結果就會全部以最新狀態來顯示。

版權頁

Xamarin.Forms 快速上手: Xamarin 開發系列叢書

檔案格式：EPUB3、PDF、MOBI

版本： 1.0
日期： 2019.11

作者： Vulcan Lee 李進興