ASP.NET async 基本心法

2019-02-20 08:42 PM 6 18,681

如果你常看微軟新技術的範例程式碼，應不難發現 async 與 await 關鍵字已如野火燎原，無所不在。 對我這個 .NET 開發老骨頭來說，寫多執行緒、用 Task 處理非同步作業難不倒我，之前甚至也研究過 await Deadlock，但每回要用 async、await 卻有種說不上來的彆扭，琢磨是沒掌握到關鍵心法，以致出招卡卡。

這兩天無意讀到 Matthew Jones 這篇 The Ultimate Guide to Asynchronous Programming in C# and ASP.NET <https://exceptionnotfound.net/asynchronous-programming-in-asp-net-csharp-ultimate-guide/> 寫得極其淺顯，通篇沒有半行程式碼，讀完卻讓我有茅塞頓開之感，對 async、await 陌生感大減，極力推薦大家一讀，而這篇則是我摘要整理的筆記。

非同步(Asynchronous)不在於提高效能(Performance)，而是增加產能(Throughput)

非同步追求的是在相同時間內處理更多請求，而非以更快的速度處理掉一個請求。總體來看，同樣的請求量在更短時間內做完，說成「效能變好」也不算錯，但要記住，非同步的核心精神在於減少等待，讓執行緒同時處理更多作業藉以提升產能。

作者用廚房的例子說明同步與非同步處理。同步廚房的每道菜都要指派一位廚師全程負責，從接受訂單，切菜備料到送進烤箱，之後在烤箱旁~~乾等滑手機~~等待烘焙完成出菜。非同步廚房則採機動調度，等待焗烤期間廚師可轉去處理其他菜肴。想當然爾，非同步廚房可以用更少的廚師做出更多的菜。

非同步不等於多執行緒

多緒執行的精神在於建立多條執行緒，將多個工作交給不同執行緒個別處理，靠分工加速；而非同步的重點則是允許執行緒在等待時間先處理其他作業(Task, 也可翻譯為工作，個人偏好作業，以與 Work 區隔)，透過消除閒置增加產能。

.NET 實現非同步作業的做法是在需要等待時建立 SynchronizationContext, 讓執行緒中斷作業先去處理其他作業，等待結束後再回頭繼續後半段。而繼續執行作業的執行緒可以是當初建立 SynchronizationContext 的那一條執行緒，也可以是其他執行緒。

非同步只對 I/O 相關作業有效，對吃 CPU 的作業沒轍

由前面的說明，我們知道非同步提升產能的關鍵在於善用等待時間先處理其他作業。當作業涉及大量消耗 CPU 的重度運算 ，代表執行緒將從頭忙到尾，既然不會閒下來等待，也就不可能分神再處理其他作業。 相反的，若作業涉及外部資源或與 I/O 相關，例如：存取資料庫、呼叫 Web API, 等待回應的期間即可透過非同步讓執行緒先處理其他作業。

換言之，若是一堆吃 CPU 的作業，增加執行緒肯定可以加速；若為要等待 I/O 回應的作業，增加執行緒用處不大，改為非同步作業才算對症下藥。

async 像病毒一樣會傳染

這是開始寫 async/await 肯定非常有感的一件事 - **async 具有傳染性**，一旦你在方法前面加上 async 關鍵字，不得了， 裡面呼叫外部方法必須加上 await 才合規格，而要加 await 該外部方法順理成章也得加上 async, 接著外部方法中又被要求使用 await... 像病毒般一路蔓延。 此一設計的理由是為了避免同步與非同步寫法混用以防止在 GUI/ASP.NET 情境產生 Deadlock。(註：作者的第二篇文章 <https://exceptionnotfound.net/asynchronous-programming-asp-net-csharp-practical-guide-refactoring/> 有 Deadlock 實例，或者也可參考我的舊文 await與Task.Result/Task.Wait()的Deadlock問題)

回傳型別

宣告為 async 的 .NET 方法必須傳回以下三種型別之一：

1. Task
   作業結束時將控制權還給呼叫端
2. Task<T>
   作業結束時回傳型別為 T 的物件給呼叫端
3. void
   採射後不理(Fire-and-Forget)哲學，呼叫後即失去掌握
   async void 在實務上不應使用，事件處理器是唯一例外

如何將同步程式重構為非同步

因為 async 像病毒會傳染，如要將同步寫法程式重構為非同步，建議採「由下而上」(Bottom-Up)策略。若物件模型有多層，例如：User 擁有 Album 及 Blog，而 Album 包含 Photo、Blog 包含 Post。建議先從底層物件(Photo、Post)開始加上 async 改成同步化，再逐步向上調整。若從上層 User 加上 async，Blog 與 Album 要跟著改，接著又迫使 Post 與 Photo 也得調，一次牽動的幅度較大。

讀後心得

過去我對非同步的應用觀念侷限於 UI 操作，想的都是 WPF、WinForm 或網頁的按鈕動作採非同步執行避免畫面凍結。擴及到 AJAX HTTP 請求上，我總認為每個 HTTP Request都是一去一回，本質已是同步化流程，ASP.NET 又何必扯上 async 庸人自擾？

體認到「非同步能消除等待閒置，更充分利用執行緒資源」的核心精神，便不難理解 async 是讓相同數量的執行緒處理更多 HTTP Request 的捷徑，意味著相同的網站硬體藉由 ASP.NET 非同步化就能達成更高的產能(Throughput)，這等好事，不試一下嗎?

有了這層認知，理解 ASP.NET 採用 async 的優點，未來開發 ASP.NET / ASP.NET Core 專案，建議一開始就貫徹 Async All the Way，省去之後的重構工程。

對非同步實做有興趣深入了解的同學，推薦蔡煥麟老師一系列精彩的剖析文章：

- » ASP.NET 應用程式與非同步處理 <https://www.huanlintalk.com/2013/01/aspnet-application-and-asynchronous.html>
- » ASP.NET 4.5 非同步呼叫之射後不理 <https://www.huanlintalk.com/2013/01/aspnet-45-fire-and-forget-async-call.html>
- » ASP.NET 程式鎖死與 SynchronizationContext <https://www.huanlintalk.com/2016/01/asyc-deadlock-in-aspbet.html>