閱讀筆記- 使用 .NET Async/Await 的常見錯誤

2019-07-23 09:30 PM 15 5,506

看到一場 NDC 演講(Norwegian Developers Conference <https://ndcsydney.com/page/about-ndc> , NDC 是 歐洲最大的 .NET / Agile 開發者研討會)，獲益不少，筆記分享之。

» 影片：https://www.youtube.com/watch?v=J0mcYVxJEl0 <https://www.youtube.com/watch?v=J0mcYVxJEl0>
» 投影片：https://www.slideshare.net/secret/M1kWxIKW7q20ku <https://www.slideshare.net/secret/M1kWxIKW7q20ku>

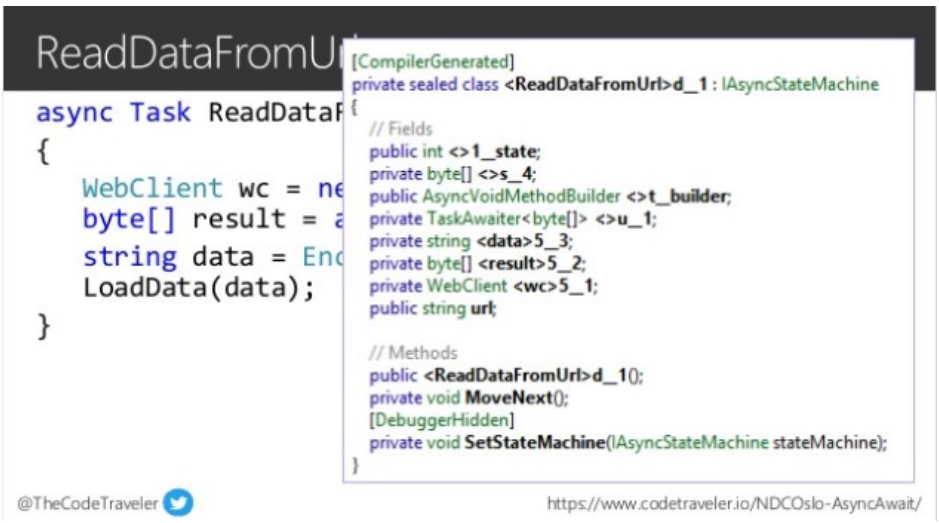
摘要重點如下：

1. 講師 Brandon Minnick <https://developer.microsoft.com/zh-tw/advocates/brandon-minnick> 是微軟開發大使，分享自己開發 Xamarin 手機程式過程曾犯過的 async/await 錯誤。
   (不理解原理程式沒寫好，出現許多光怪陸離的奇妙錯誤，從 Stack Trace 根本想不出來程式怎能跑成這副德行。)
2. 多執行緒執行能充分利用 CPU 運算資源，提升程式運作效能，.NET 開發者很幸福有現成的 Thread Pool 管理機制，開發者只需交付工作，.NET 會自己協調調度執行緒將其完成。
3. 編譯器將每個 await 前後拆成三段，例如：

```
async Task ReadDataFromUrl(string url)
{
    var wc = new WebClient();
    byte[] result = await wc.DownloadDataTaskAsync(url);
    string data = Encoding.UTF8.GetString(result);
    LoadData(data);
}
```

   await 之前的程式由 Thread 1 執行，wc.DownloadDataTaskAsync() 交給 Thread 2 執行，此時 Thread 1 被釋放可處理其他事(當 Thread 1 是 UI Thread, 這點就格外重要，UI Thread 忙碌時，畫面無法更新介面會凍結)，待 DownloadDataTaskAsync() 完成後，繼續由 Thread 1 做完下面的事。
   [2019-08-26 更新]感謝讀者 This Wayne 補充，在影片評論中 另一位 MVP Stephen Cleary 提出 DownloadDataTaskAsync() 將不會動用另一條 Thread 的主張。Stephen 有一篇Blog 文章 <https://blog.stephencleary.com/2013/11/there-is-no-thread.html> 詳細剖析了 I/O 動作底層機制，說明這類低階 IO 處理是用類似「借用原 Thread」的方式執行，並不會觸發 Thread 切換。但 async 讓 UI Thread 不必等待結果防止凍結的理論不變。

4. .NET 編譯遇到 async 方法時，會將其轉換一個 IAsyncStateMachine 類別：(執行檔大小增加約 100 Bytes)



而在 MoveNext() 方法裡，程式被拆成前後兩段，switch 不同 case 的程式片段可由不同執行緒執行：

```
public void MoveNext()
{
 uint num = (uint)this.$PC;
 this.$PC = -1;
 try
 {
     switch (num)
     {
         case 0:
             this.<wc>__0 = new WebClient();
             this.$awaiter0 = this.<wc>__0.DownloadDataTaskAsync(this.url).GetAwaiter();
             this.$PC = 1;
             //...
             return;
             break;
         case 1:
             this.<result>__1 = this.$awaiter0.GetResult();
             this.<data>__2 = Encoding.ASCII.GetString(this.<result>__1);
             this.$this.LoadData(this.<data>__2);
             break;
         default:
             return;
     }

 }
 catch (Exception exception)
 {
     //...

 }
 this.$PC = -1;
 this.$builder.SetResult();
}
```

偵錯過程有時會在 Stack Trace 看到原始碼沒寫過的 MoveNext() 方法，就是 .NET 背後加工造成的。

5. MoveNext() 中使用 try ... catch 捕捉例外，在 await 時會拋出，但如果你採取 Fire-and-Forget 策略，呼叫 async 方法卻不等結果，程式將忽略這些錯誤繼續執行， 產生難以預期的結果。
6. 錯誤修正範例 1

當使用 UI Thread 呼叫 async 方法時，勿使用 SomeAsyncMethod().Wait(), 它會佔用 UI Thread 等待結果，等

待期間 UI 將凍結無法操作。

改成 await SomeAsyncMethod(); 可避免。

7. 錯誤修正範例 2

   若 await 後的程式操作不必限定 await 前的 Thread 接手處理(例如：與 UI Thread 無關)，建議將 await SomeAsyncMethod() 改成 await SomeAsyncMethod().ConfigureAwait(false)， 允許 .NET 調撥閒置的 Thread 處理，不必等待與佔用寶貴的 UI Thread。

   補充：此招僅適用 WinForm/WPF/ASP.NET 4.x。Console 程式與 ASP.NET Core 的等待器不記錄 SynchronizationContext， 故 ConfigureAwait() 無影響。延伸閱讀：有關Task ConfigureAwait() 的一些事情| 微軟開發工具資訊分享- 點部落 <https://dotblogs.com.tw/aspnetshare/2018/06/03/configureawaiter>

8. 錯誤範例 3

```
Task<string> GetData()
{
     try
     {
         return SomeAsyncMethod(); //SomeAsyncMethod傳回Task<string>
     }
     catch (Exception ex)
     {
         Log(ex);
         throw;
     }
}
```

   傳回還在執行中的 Task， 你將永遠抓不到 Exception。

9. ValueTask 可以改善效能，原理是大量反覆執行時，ValueTask 會儲存在 Stack 記憶體，比 Task 放 Heap 記憶體更有效率。延伸閱讀：C# 7.0 新增 ValueTask 的用意 by kinanson <https://dotblogs.com.tw/kinanson/2018/04/25/091013>

10. async void 在發動後無從掌握，除非是為了符合事件函式簽章回傳值 void 之外，勿用。延伸閱讀：async 與 await by Huanlin學習筆記 <https://www.huanlintalk.com/2016/01/async-and-await.html>

11. 需要得到結果才繼續執行的場合，請使用 .GetAwaiter().GetResult() 取代 .Result， 雖然也會鎖定 Thread， 至少會拋回包含明確 StackTrace、程式碼位置的例外物件。(.Wait()/.Result拋回的是 AggrateException)。

12. 作者發明了 SafeFireAndForget() 方法，明確表達此處真的不用等結果不是不小心寫錯，並保留處理例外的機制。

13. 不要 return await， 拿掉 async， 改 return Task 就好。(除非是用在 try / catch 或 using 區塊)

    例如：

```
async Task<string> SomeMethod()
{
    //... some logic ...
    return await AnotherAsyncMethodReturnString();
}
```

    前面提過加 async 後該函式會被轉成 IAsyncStateMachine 類別並增加 100 Bytes， 但在此案例卻沒帶來任何好處，徒增檔案大小及無謂 Thread 切換。建議拿掉 async/await：

```
Task<string> SomeMethod()
{
    //... some logic ...
    return AnotherAsyncMethodReturnString();
}
```

    但 return await AnotherAsyncMethodReturnString() 如果有被包在 try/catch 或 using 區塊裡就另當別論。

讚　242 人說這個讚。成為朋友中第一個說讚的人。