C# .NET Blazor Research

針對 .NET / CLR / C# / Blazor / .NET Core / .NET Framework / OOP / Design Pattern 等相關程式開發議題進行研究與寫成相關心得筆記

Blazor 快速體驗 電子書 Hands-On Lab 動手練習

2019年2月24日 星期日

.NET Framework 執行緒 Thread 與 工作 Task 的非同步方法建立與使用

.NET Framework 執行緒 Thread 與 工作 Task 的非同步方法建立與使用

若你正在觀看此篇文章，那麼你將會對於 **為什麼需要使用非同步程式設計，真的可以提升整體應用程式的執行效能嗎？** 問題更有興趣的。

在 .NET Framework 中，可以透過各種不同類別，建立與設計出各種不同的非同步應用程式，在這篇文章中，將要來一次性的瞭解各種類別的建立與使用方式，設計出一個非同步的應用，這些類別分別是： 執行緒 Thread、背景工作者 BackgroundWorker、定時器 Timer、執行緒的集區 ThreadPool、工作 Task、工作產生工廠方法 Task.Factory.StartNew、自動建立工作 Task.Run。

了解更多關於 [使用 async 和 await 進行非同步程式設計] 的使用方式

了解更多關於 [Thread Class] 的使用方式

了解更多關於 [Task Class] 的使用方式

執行緒 Thread 無傳入參數

想要建立一個執行緒物件，並且啟動該執行緒進行非同步的計算，可以先建立一個 ThreadStart 委派物件，在其建構式傳入一個無參數的委派方法，接著，把這個 ThreadStart 委派物件，傳入 Thread 類別的建構式內，當然，還有更為簡單的作法，那就是不需要使用剛剛說明的過程，直接使用這一的敘述 `Thread thread = new Thread(MyDoWork);` 也就是，將一個無參數的委派方法傳入到該 Thread 建構函式參數內。

不論使用哪種方法，當使用 new 運算子建立起一個 Thread 物件之後，該執行緒物件內的委派方法並不會被執行，程式設計師需要使用該執行緒物件的 Start() 方法，啟動該執行緒該使執行。

在這個範例中，將會先顯示出主執行緒的 受管理的 Managed 執行緒 ID 號碼，接著啟動該執行緒，在新的執行緒中，將會先顯示現行執行緒的 受管理的 Managed 執行緒 ID 號碼(透過這兩個號碼，可以確認當時至少有兩個執行緒正在執行中)，接著會休息 3 秒鐘，而後會進行計算處理，將計算結果儲存到 sum 這個欄位中。

```
class Program
{
    static int sum = 0;
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        ThreadStart threadDelegate = new ThreadStart(MyDoWork);
        Thread thread = new Thread(threadDelegate);
        //Thread thread = new Thread(MyDoWork);
        thread.Start();
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
        Console.WriteLine($"Sum = {sum}");
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static void MyDoWork()
    {
        Console.WriteLine($"MyDowork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        int foo1 = 40;
        int foo2 = 80;
        sum = foo1 + foo2;
        Console.WriteLine($"MyDowork 執行完畢");
    }
}
```

這樣設計的的程式碼，可能是一般程式設計師在第一次接觸使用執行緒來設計非同步應用的情境，可是，對於非同步程式設計領域，這是相當複雜與繁瑣的，稍不注意，將會造成不是程式設計師所預期的情況，而且，想要進行除錯，可說是相當不容易。

首先，執行這個應用程式，當專案執行之後，請等候 3 秒鐘以上，接著按下任一按鍵，現在應該會看到如下輸出結果。

在這裡看到主程式會在執行緒1下來執行，而 MyDowork 將會在執行緒3下來執行，此時，執行緒3下執行的 MyDowork 在休息三秒鐘之後，將會把計算結果儲存到 sum 欄位內；現在當使用者按下任一按鍵後，主執行緒便會將 sum 欄位內的值顯示在螢幕上，也就是看到的底下結果。

```
Main 執行緒 ID=1
Press any key for continuing...
MyDowork 執行緒 ID=3
MyDowork 執行完畢
Sum = 120
Press any key for continuing...
```

現在，嘗試另外一種執行操作模式，也就是，專案執行之後，請在 3 秒鐘內，按下任何一按鍵，現在，將會看到底下的結果。

疑，為什麼現在看到的 sum 欄位值竟然顯示為 0，而且對於輸出的文字順序也與前面執行結果不太相同。根據程式設計邏輯，這裡模擬要取得這個非同步的計算結果，至少需要 3 秒鐘的時間，才能夠把計算結果儲存到 sum 欄位內，若有任何情況，想要在 3 秒鐘內就想要提早讀取 sum 欄位值，當然是無法讀取到非同步計算的結果，因為，這個非同步計算的程式碼還在繼續執行中，尚未算出結果。

```
Main 執行緒 ID=1
Press any key for continuing...
MyDowork 執行緒 ID=3
Sum = 0
Press any key for continuing...
MyDowork 執行完畢
```


熱門文章

C# H
POST
Web
若你正在觀
用 HttpClien
要授權的 AP
設計範例 這
們來使用最
現在滿多人
用 ...

JWT Token
他 Web API
HttpClient

效能嗎？ 問
談絕大部分
正確的 C# 看
們採用 工作

要授權的 AP
設計範例 這
於 [HttpClie
關於 [...

C# H
POST
與接
若你正在觀
用 HttpClien
要授權的 AP
設計範例 這
Http 通訊協
Protocol), t

C# 的
C# 的
當在
工作程式設

執行緒 Thread 有傳入參數

```
class Program
{
    static int sum = 0;
    class MyClass
    {
        public int value1 { get; set; }
        public int value2 { get; set; }
    }
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        ParameterizedThreadStart threadDelegate = new ParameterizedThreadStart(MyDoWork);
        Thread thread = new Thread(threadDelegate);
        //Thread thread = new Thread(MyDoWork);
        thread.Start(new MyClass { value1 = 40, value2 = 80 });
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
        Console.WriteLine($"Sum = {sum}");
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static void MyDoWork(object obj)
    {
        MyClass myClass = obj as MyClass;
        Console.WriteLine($"MyDwork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        sum = myClass.value1 + myClass.value2;
        Console.WriteLine($"MyDwork 執行完畢");
    }
}
```

```
Main 執行緒 ID=1
Press any key for continuing...
MyDwork 執行緒 ID=3
MyDwork 執行完畢
Sum = 120
Press any key for continuing...
```

```
Main 執行緒 ID=1
Press any key for continuing...
MyDwork 執行緒 ID=3
Sum = 0
Press any key for continuing...
MyDwork 執行完畢
```

等候非同步執行緒結束

在前面兩個 執行緒 Thread 無傳入參數 與 執行緒 Thread 有傳入參數 的範例中，當主執行緒啟動一個新的執行緒，執行非同步的處理程序，此時，在主執行緒端視無法知道非同步執行緒端的程序何時會完成，所以，會造成使用者操作的程序不同，而會得到不同的結果，因此，要如何解決此一問題呢？

這個時候，需要使用執行緒同步化 Synchronization 的設計，在底下的範例中，使用的是一個最為簡單的設計方式，也就是當主執行緒啟動一個非同步的執行緒之後，當主執行緒想要取得非同步的執行緒的最後執行結果，需要透過執行緒物件，呼叫 Join 方法，此時，主執行緒將會被封鎖 Block 並且直到非同步執行緒執行完成之後，才會繼續執行；此時，若來讀取 sum 欄位值的話，因為非同步執行緒已經執行完成，因此，必定可以讀取到最後執行結果內容。

```
class Program
{
    static int sum = 0;
    class MyClass
    {
        public int value1 { get; set; }
        public int value2 { get; set; }
    }
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        ParameterizedThreadStart threadDelegate = new ParameterizedThreadStart(MyDoWork);
        Thread thread = new Thread(threadDelegate);
        //Thread thread = new Thread(MyDoWork);
        thread.Start(new MyClass { value1 = 40, value2 = 80 });
        Console.WriteLine("等候非同步執行緒結束...");
        thread.Join();
        Console.WriteLine($"Sum = {sum}");
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static void MyDoWork(object obj)
    {
        MyClass myClass = obj as MyClass;
        Console.WriteLine($"MyDwork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        sum = myClass.value1 + myClass.value2;
        Console.WriteLine($"MyDwork 執行完畢");
    }
}
```

底下將會為上述範例程式碼的執行結果，而且，每次執行結果都會顯示出相同的輸出內容。

```
Main 執行緒 ID=1
等候非同步執行緒結束...
MyDwork 執行緒 ID=3
MyDwork 執行完畢
Sum = 120
Press any key for continuing...
```

背景工作者 BackgroundWorker

BackgroundWorker 這個類別可以讓指定的委派方法，可以在不同執行緒上背景執行作業，當背景執行緒執行完成之後，將會執行是先定義的 回呼 callback 委派方法，進行接下來的處理工作；在這樣的設計方法之下，並不需要直接去建立一個執行緒物件，就可以設計出非同步運行的功能， BackgroundWorker 這個物件會自動的建立一個執行緒，在背景來執行相關作業。

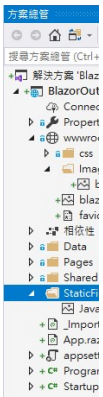
在這個範例中，首先建立一個 BackgroundWorker 物件，接著需要設定兩個委派事件，第一個是當呼叫了方法 RunWorkerAsync 之後，需要開始進行非同步處理作業的時候，此時，可以透過 DoWorkEventHandler 定義的事件，這裡是宣告了 MyDoWork 委派事件，來執行所需要的背景處理工作，在這裡，如同上面的範例相同，也是先暫時休息 3 秒鐘，模擬這個時候要處理其他相關工作，接著要把兩個整數數值相加起來。

在使用 DoWorkEventHandler 的委派事件方法內，要如何取得要計算的兩個整數數值呢？這個時候，當呼叫 RunWorkerAsync 方法，可以把要傳送到 DoWorkEventHandler 委派事件的引數傳送過去，而在 DoWorkEventHandler 委派方法內，可以透過 DoWorkEventArgs.Argument 取得要計算的兩個整數數值。

精選文章

ASP.NET Cor
與其他目錄的

ASP.NET Core B
錄的靜態圖片使
練習如何在 Bla
圖片檔案；這些
專案的網站根目
案資料夾，宣告
網...



文章分類

- C# (51)
- ASP.NET (2)
- Visual Studio

搜尋此網誌

檢舉濫用情形

當 DoWorkEventHandler 委派事件方法處理完成之後，可以把處理完成的結果內容，設定到 DoWorkEventArgs.Result 屬性內，緊接著將會 回呼 callback 當初宣告的 RunWorkerCompletedEventHandler 委派事件方法，在這個方法內，可以透過參數 RunWorkerCompletedEventArgs.Result 來取得非同步計算的結果。

```
class Program
{
    static int sum = 0;
    class MyClass
    {
        public int value1 { get; set; }
        public int value2 { get; set; }
    }
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        BackgroundWorker bkWorker = new BackgroundWorker();
        bkWorker.DoWork += new DoWorkEventHandler(MyDoWork);
        bkWorker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(MyDoWorkCompleted);
        bkWorker.RunWorkerAsync(new MyClass { value1 = 40, value2 = 80 });
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static void MyDoWorkCompleted(object sender, RunWorkerCompletedEventArgs e)
    {
        Console.WriteLine($"MyDoWorkCompleted 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Console.WriteLine($"Sum = {e.Result}");
        Console.WriteLine($"MyDoWorkCompleted 執行完畢");
    }
    private static void MyDoWork(object sender, DoWorkEventArgs e)
    {
        MyClass myClass = e.Argument as MyClass;
        Console.WriteLine($"MyDwork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        int fooSum = myClass.value1 + myClass.value2;
        Console.WriteLine($"MyDwork 執行完畢");
        e.Result = fooSum;
    }
}
```

底下是使用 BackgroundWorker 這個物件，實作出一個非同步作業的輸出結果。

```
Main 執行緒 ID=1
Press any key for continuing...
MyDwork 執行緒 ID=3
MyDwork 執行完畢
MyDoWorkCompleted 執行緒 ID=4
Sum = 120
MyDoWorkCompleted 執行完畢
```

定時器 Timer

定時器 這種類型的物件，也是可以建立出非同步的應用作業，只不過，在這裡將會在指定的時間之內，執行事先宣告的委派方法。在這個範例，將會使用 System.Threading.Timer 這個類別的定時器類別 (在 .NET Framework 內，提供了許多種不同情境可以使用定時器物件)，設定了當定時器物件建立之後，在 1 秒鐘後，便會開始執行，而後是每 2 秒鐘執行一次。

當使用 System.Threading.Timer 建構式，可以傳送一個物件直到委派方法內。

```
class Program
{
    static void Main(string[] args)
    {
        Timer timer = new Timer(MyDoWork, "我要定時運行", 1000, 2000 );
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static void MyDoWork(object state)
    {
        Console.Write($"{state} ");
    }
}
```

底下將會是上述程式碼的執行結果。

```
Press any key for continuing...
我要定時運行 我要定時運行 我要定時運行 我要定時運行 我要定時運行
```

執行緒的集區 ThreadPool

當 C# 程式設計師使用 Thread 類別建立起一個 Thread 物件的時候，系統需要花費成本來建立起一個執行緒物件，而且每個執行將會耗用 1MB 記憶體空間，而當該執行緒使用完成之後，就會透過 .NET CLR 內的 Garbage Collection 垃圾回收功能，把這個執行緒物件回收；若所設計的應用程式經常的需要建立執行緒物件、回收記憶體物件，對於整體系統效能而言，也是一個相當的負擔。

而 執行緒集區 就是要來解決這個問題，當 .NET 應用程式啟動的時候，在執行緒集區內就會事先建立好一些執行緒，當程式開發者需要用到一個背景執行緒的時候，可以透過 QueueUserWorkItem 方法，傳送一個委派方法，此時，執行緒集區將會從自己內部找出一個執行緒，開始背景執行這個委派方法；當該背景執行緒執行完畢之後，執行緒集區便會將該執行緒回收回來，以便下次有程式碼需要一個執行緒的時候，可以提供出來使用，因此，這些執行緒就不再需要被 CLR GC 機制回收。

在底下的範例中，透過 ThreadPool.QueueUserWorkItem 靜態方法，從執行緒集區找出一個可用的執行緒，開始執行 MyDoWork 這個方法，不過，您將會發現到，您無法取得執行緒集區執行的執行緒物件，也就無法使用 Join 方法來等候這個背景執行緒完成工作，但是，在 .NET Framework 類別庫中，還有更多關於執行緒同步的類別可以使用，想要使用執行緒集區做到執行緒間的同步，不是一個問題。

```
class Program
{
    static int sum = 0;
    class MyClass
    {
        public int value1 { get; set; }
        public int value2 { get; set; }
    }
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        ThreadPool.QueueUserWorkItem(new WaitCallback(MyDoWork), new MyClass { value1 = 40, value2 = 80 });
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
        Console.WriteLine($"Sum = {sum}");
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static void MyDoWork(object obj)
    {
        MyClass myClass = obj as MyClass;
        Console.WriteLine($"MyDwork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        sum = myClass.value1 + myClass.value2;
        Console.WriteLine($"MyDwork 執行完畢");
    }
}
```

底下為使用執行緒集區計算兩個數值的相加結果。

```
Main 執行緒 ID=1
Press any key for continuing...
MyDwork 執行緒 ID=3
MyDwork 執行完畢
Sum = 120
Press any key for continuing...
```

若專案執行後，在 3 秒鐘內按下任何一按鍵，將會得到不同的結果。

```
Main 執行緒 ID=1
Press any key for continuing...
MyDwork 執行緒 ID=3
Sum = 0
Press any key for continuing...
MyDwork 執行完畢
```

工作 Task - 手動建立與啟動

工作 Task 這個類別，是現在 .NET Framework 開發上，微軟建議使用於設計非同步 / 多工應用的類別，畢竟，直接使用上述的幾種做法(執行緒、執行緒集區)，都會存在著許多問題。工作類別是在 .NET Framework 4.0 於 TPL 架構提供的功能，可以方便開發者輕鬆建立出非同步作業的物件。

在底下的範例中，建立一個 Task 物件，在這裡使用的是泛型工作類別，因為，這個工作將會要有回傳結果。在 Task 類別建構函式，可以指定要非同步執行的委派方法是哪個，以及要傳送到這個非同步方法的參數是甚麼？當使用 Task 物件，將不會看到任何執行緒的物件，但是，整個工作物件在執行運作過程中，將會是透過執行緒來做到多工、非同步作業效果。

使用 Task 類別建立出來的 Task 物件，非同步作業不會自動執行，需要特別呼叫 Task.Start() 方法，這樣，這個非同步工作才會正常執行；當要非同步執行委派方法的時候，工作物件預設會使用執行緒集區來取得一個執行緒，讓這個執行緒來執行這個委派方法。

在主執行緒想要等待非同步工作執行完成，可以使用工作的 Wait() 方法，不過，執行這個方法將會在這當時這個執行緒進入封鎖

Block 狀態，直到非同步工作執行完成之後，才會繼續執行下去。

想要取得非同步工作的執行結果，可以透過 Task.Resut 屬性值來取得。

```
class Program
{
    class MyClass
    {
        public int value1 { get; set; }
        public int value2 { get; set; }
    }
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Task<int> task = new Task<int>(MyDoWork, new MyClass { value1 = 40, value2 = 80 });
        task.Start();
        task.Wait();
        Console.WriteLine($"Sum = {task.Result}");
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static int MyDoWork(object obj)
    {
        MyClass myClass = obj as MyClass;
        Console.WriteLine($"MyDwork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        int fooSum = myClass.value1 + myClass.value2;
        Console.WriteLine($"MyDwork 執行完畢");
        return fooSum;
    }
}
```

這裡是直接建立一個工作物件，執行一個非同步工作的輸出結果。

```
Main 執行緒 ID=1
MyDwork 執行緒 ID=3
MyDwork 執行完畢
Sum = 120
Press any key for continuing...
```

靜態工廠方法產生工作 - 自動執行

上面的做法：建立一個 Task 物件，並且啟動該非同步工作的作法，有點繁瑣，在 .NET Framework 4.0 同樣的提供一個靜態工廠方法 Task.Factory.StartNew， 可以建立一個非同步工作，並且該工作會立即執行。這個靜態方法也提供了許多多載方法可以選擇，開發者可以依據自己的需求選擇適合的呼叫參數。

```
class Program
{
    class MyClass
    {
        public int value1 { get; set; }
        public int value2 { get; set; }
    }
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Task<int> task = Task.Factory.StartNew<int>(MyDoWork, new MyClass { value1 = 40, value2 = 80 });
        task.Wait();
        Console.WriteLine($"Sum = {task.Result}");
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static int MyDoWork(object obj)
    {
        MyClass myClass = obj as MyClass;
        Console.WriteLine($"MyDwork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        int fooSum = myClass.value1 + myClass.value2;
        Console.WriteLine($"MyDwork 執行完畢");
        return fooSum;
    }
}
```

這是使用靜態方法 Task.Factory.StartNew 所建立的非同步工作的執行結果

```
Main 執行緒 ID=1
MyDwork 執行緒 ID=3
MyDwork 執行完畢
Sum = 120
Press any key for continuing...
```

建議使用的工作物件建立方式

在 .NET Framework 4.5 的時候，推出一個更輕巧、容易使用的建立與起動工作的靜態方法：Task.Run, 微軟官方也是建議，若在設計非同步工作應用，沒有特別獨特的需求，建議使用 Task.Run 來建立一個非同步工作。

當在使用 Task.Run 建立，想要將參數傳遞到非同步方法內，其沒有類似 Task.Factory.StartNew 多載方法，不過，可以透過 Lambda 來做到相同的結果。

```
class Program
{
    class MyClass
    {
        public int value1 { get; set; }
        public int value2 { get; set; }
    }
    static void Main(string[] args)
    {
        Console.WriteLine($"Main 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Task<int> task = Task.Run<int>(()=>MyDoWork(new MyClass { value1 = 40, value2 = 80 }));
        task.Wait();
        Console.WriteLine($"Sum = {task.Result}");
        Console.WriteLine("Press any key for continuing...");
        Console.ReadKey();
    }
    private static int MyDoWork(object obj)
    {
        MyClass myClass = obj as MyClass;
        Console.WriteLine($"MyDwork 執行緒 ID={Thread.CurrentThread.ManagedThreadId}");
        Thread.Sleep(3000);
        int fooSum = myClass.value1 + myClass.value2;
        Console.WriteLine($"MyDwork 執行完畢");
        return fooSum;
    }
}
```

這是使用靜態方法 Task.Run 所建立的非同步工作的執行結果

```
Main 執行緒 ID=1
MyDwork 執行緒 ID=3
MyDwork 執行完畢
Sum = 120
Press any key for continuing...
```

對於已經具備擁有 .NET / C# 開發技能的開發者，可以使用 Xamarin.Forms Toolkit 開發工具，便可以立即開發出可以在 Android / iOS 平台上執行的 App；對於要學習如何使用 Xamarin.Forms & XAML 技能，現在已經推出兩本電子書來幫助大家學這這個開發技術。

這兩本電子書內包含了豐富的逐步開發教學內容與相關觀念、各種練習範例，歡迎各位購買。

想要購買 Xamarin.Forms 快速上手 電子書，請點選 這裡

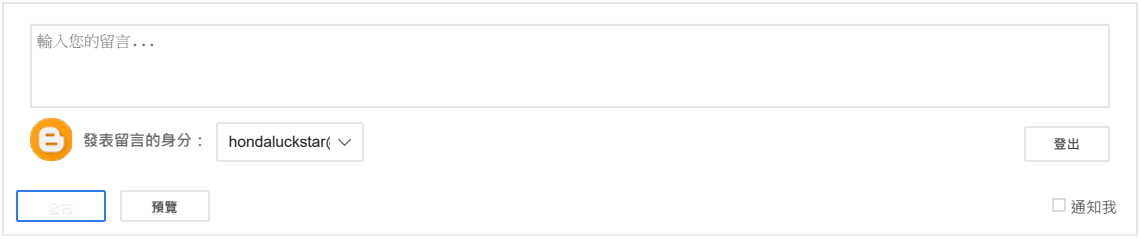
想要購買 XAML in Xamarin.Forms 基礎篇 電子書，請點選 這裡

By Vulcan lee 於 2月 24, 2019

標籤： Asynchronous, C#, MVP2018

沒有留言:

張貼留言



這篇文章的連結

建立連結

訂閱： 張貼留言 (Atom)