{B}

執行緒

Visual C# 2008

在應用程式開發上，可利用多執行緒（Multithread）同時執行多個工作，所謂多執行緒是指系統內有數個連續性的工作同時進行，多執行緒爲多個執行緒同時執行，而屬於同一應用程式的執行緒之間，可共享程式碼，因此執行緒會使用較少的系統資源並可增加執行效率。

在 Microsoft .Net Framework 中，有關執行緒的類別，是由 System.Threading 命名空間所提供，包括設計多執行緒程式的類別與介面、執行緒群組管理的 ThreadPool 類別、處理同步（Synchronization）、優先順序（Priority）、排程、等候等執行緒的類別。

System.Threading 命名空間包含以下之類別成員：

類別：

- AutoResetEvent
- Interlocked
- ManualResetEvent
- Monitor
- Mutex
- ReaderWriterLock
- RegisteredWaitHandle
- SynchronizationLockException
- Thread
- ThreadAbortException
- ThreadExceptionEventArgs
- ThreadInterruptedException
- ThreadPool
- ThreadStateException
- Timeout
- Timer
- WaitHandle

結構：

- LockCookie
- NativeOverlapped

委派：

- IOCompletionCallback
- ThreadExceptionEventHandler
- ThreadStart
- TimerCallback
- WaitCallback
- WaitOrTimerCallback

列舉型別：

- ApartmentState
- ThreadPriority
- ThreadState

► 建立執行緒

欲建立執行緒，可使用 System.Threading.Thread 類別的建構函式產生執行緒，Thread 類別負責管理多執行緒功能，其建構函式：

```
public Thread(ThreadStart start);
```

其中參數 start 爲參考欲呼叫之 ThreadStart 委派，表示在執行緒開始執行時所呼叫的方法，通常爲自訂類別。

其例外錯誤：

- System.ArgumentNullException：參數 start 爲 null。

通常執行緒程式之語法爲：

```
// 主程式：建立執行緒
using System.Threading ;
...
  try {
    ThreadClass tc = new ThreadClass(args1, arg2, ...) ;
    ThreadStart ts = new ThreadStart(tc.ThreadProc) ;

    Thread myThread = new Thread(ts) ;
    ...
  }
  catch (ArgumentNullException ex) {
    ...
  }

// 自訂 Thread 類別
using System.Threading ;
...
public class ThreadClass {
  ...
  // 建構函式
  public ThreadClass(... args1, ... args2, ...) {
    ...
  }

  // 自訂執行緒主程式
  public void ThreadProc() {
    ...
  }
}
```

在主程式中，使用 System.Threading.Thread 類別的建構函式建立執行緒，其中 ThreadClass 爲自訂之類別，以負責在執行緒中欲執行之程式。主程式可藉由參數傳遞，交由自訂的 ThreadClass 類別處理，在自訂的類別中，參數的承接則由 ThreadClass 類別的建構函式處理，例如：

```
// 建構函式
public ThreadClass(... args1, ... args2, ...) {
  ...
}
```

另外，若傳遞之參數有不同形式，可利用物件導向的多型（Polymorphism）方式，定義數個建構函式以不同的方式處理參數傳遞，例如：

```
// 建構函式 1
public ThreadClass(... args1) {
  ...
}

// 建構函式 2
public ThreadClass(... args1, ... args2, ...) {
  ...
}
```

其次，在主程式的 Thread 類別建構函式中，以 AddressOf 關鍵字定義自訂 ThreadClass 類別的程式切入點，則參考欲呼叫之 ThreadStart 委派，例如：

```
ThreadStart ts = new ThreadStart(tc.ThreadProc) ;
Thread myThread = new Thread(ts) ;
```

本例的切入點（ThreadStart 委派）爲自訂 ThreadClass 類別的 ThreadProc：

```
// 自訂執行緒主程式
public void ThreadProc() {
  ...
}
```

如此便完成執行緒的建立，請參考以下之程式片段：

```
// 主程式：建立執行緒
using System;
```

```
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace ServerSocket {
  class Server {
    static void Main(string[] args) {
      try {
        Socket serverSocket = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp) ;
        IPAddress serverIP =
            Dns.Resolve("localhost").AddressList[0] ;
        string Port = "80" ;
        IPEndPoint serverhost = new IPEndPoint(serverIP,
            Int32.Parse(Port)) ;
        serverSocket.Bind(serverhost) ;
        serverSocket.Listen(50) ;

        ListenClient lc = new ListenClient(serverSocket) ;
        ThreadStart serverThreadStart = new
            ThreadStart(lc.ServerThreadProc);
        Thread serverthread = new Thread(serverThreadStart);

        serverthread.Start() ;
      }
      catch (Exception ex) {
        ...
      }
    }
  }
}

// 自訂 Thread 類別
using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

```
namespace ServerSocket {
  public class ListenClient {
    private System.Net.Sockets.Socket serverSocket ;
    private System.Net.Sockets.Socket clientSocket ;

    // 建構函式
    public ListenClient(Socket serverSocket) {
      this.serverSocket = serverSocket ;
    }

    public void ServerThreadProc() {
      while (true) {
        try {
          clientSocket = serverSocket.Accept() ;

          IPEndPoint clientInfo = (IPEndPoint)
              clientSocket.RemoteEndPoint ;
          IPEndPoint serverInfo = (IPEndPoint)
              serverSocket.LocalEndPoint ;
          ...
        }
        catch (Exception ex) {
          ...
        }
      }
    }
  }
}
```

► 啓動執行緒

以 System.Threading.Thread 類別建立執行緒之後，接著以 Thread 類別的 Start 方法啓動執行緒：

```
public void Start();
```

其例外錯誤：

- System.Threading.ThreadStateException：執行緒已被啓動。
- System.Security.SecurityException：無適當權限啓動此執行緒。
- System.OutOfMemoryException：無足夠記憶體啓動此執行緒。

例如：

```
ThreadStart ts = new ThreadStart(tc.ThreadProc) ;
Thread myThread = new Thread(ts) ;

myThread.Start() ;
```

需說明的是，Thread 類別中的 ThreadState 屬性可判斷目前執行緒的狀態。當執行緒建立之後，ThreadState 屬性爲 Unstarted，當執行緒啓動（Start）之後，ThreadState 屬性則爲 Running，相關執行緒狀態如下表所示：

ThreadState 屬性	執行緒狀態
AbortRequested	執行緒呼叫 Abort 方法
Running	以 Start 方法啓動執行緒、另一執行緒呼叫 Interrupt 或 Resume 方法
Stopped	執行緒回應 Abort 或終止執行緒
Suspended	執行緒回應 Suspend 方法
SuspendRequested	執行緒呼叫 Suspend 方法
Unstarted	建立執行緒
WaitSleepJoin	執行緒呼叫 Sleep、另一物件的 Wait 或另一執行緒的 Join 方法

如此便完成執行緒的建立。

► 改變執行緒狀態

欲停止執行緒，可使用 Thread 類別的 Abort 方法停止：

```
public void Abort();

public void Abort(object stateInfo);
```

其例外錯誤：

- System.Threading.ThreadStateException：執行緒已被停止。
- System.Security.SecurityException：無適當權限停止此執行緒。

例如：

```
ThreadStart ts = new ThreadStart(tc.ThreadProc) ;
Thread myThread = new Thread(ts) ;

myThread.Start() ;
...
myThread.Abort() ;
```

除 Abort 方法停止執行緒之外，Thread 類別提供 Interrupt、Resume、Sleep、Suspend 方法分別中斷、繼續暫停、暫停執行緒：

- Interrupt：中斷狀態爲 WaitSleepJoin 的執行緒。
- Resume：繼續暫停執行緒。
- Sleep：將執行緒暫停一段時間 (使用者自訂)，以給予相同或較低優先順序的執行緒執行機會。
- Suspend：暫停執行緒。

例如：

```
myThread.Interrupt() ;
...
myThread.Resume() ;
...
myThread.Sleep() ;
...
myThread.Suspend() ;
...
```

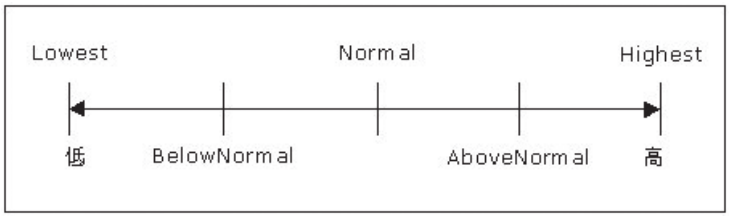
► 執行緒優先順序

每個執行緒都具有優先順序，作業系統會配置較長的 CPU 時間給優先順序高的執行緒，執行緒的優先順序因作業系統而異，Microsoft Windows NT 提供七個階層，Unix 作業系統提供 255 或更多個階層，但爲了保持 .Net Framework 的一致性，因此 .Net Framework 有自己定義的優先順序。

Microsoft .Net Common Language Runtime 的執行緒是屬於 Priority-Based（優先順序），因此每個執行緒可被指定優先順序以決定其執行優先順序，當執行緒建立時，其預設執行優先順序爲 Normal。

其次，可使用 `System.Threading.Thread.Priority` 屬性取得或設定執行緒的優先順序，爲以下 `System.Threading.ThreadPriority` 之列舉值：

- `ThreadPriority.Highest`：最高執行優先順序。
- `ThreadPriority.AboveNormal`：優先順序爲 `Highest` 執行緒之後、`Normal` 執行緒之前。
- `ThreadPriority.Normal`：預設執行優先順序，爲 `AboveNormal` 執行緒之後、`BelowNormal` 執行緒之前。
- `ThreadPriority.BelowNormal`：優先順序爲 `Normal` 執行緒之後、`Lowest` 執行緒之前。
- `ThreadPriority.Lowest`：最低執行優先順序。



執行緒的執行優先順序

❊【參考資料】

[1] Microsoft Developer Network.