(T26)討論Thread(執行緒)  
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92  
=======================================================================  
(T26)討論Thread(執行緒)

(T26-1)討論Thread(執行緒)

(T26-2)討論Thread(執行緒)、Join  
=======================================================================  
0. Summary

-----------

1. New Project

1.1. Create New Project : Sample

-----------

2. Sample : Program.cs

-----------

3. Console App(.NET Framework) - SampleV2 - Program.cs  
=======================================================================

0. Summary

1.

Thread

1.1.

In computing, a process is an instance of

a computer program that is being executed.

Windows Task Manager provide limit functions

to control the process in Windows operation system.

A process has one main thread and

might have some other threads.

Each tread executes the different piece of code.

For a single core machine,

Asynchronous Programming might reduce performance

because of context-switching.

However, Most machine nowadays normally have multiple cores CPU

which allows Asynchronous Programming provide a way

to run thread/Tasks simultaneously.

1.2.

Thread member.

1.2.1.

ThreadObject.Join()

ThreadObject.Join(int millisecondsTimeout)

ThreadObject.Join(TimeSpan timeout)

Reference:

https://msdn.microsoft.com/en-us/library/system.threading.thread.join(v=vs.110).aspx

Blocks the calling thread until ThreadObject terminates or until timeout.

Return true if the thread has been terminated;

Return false if the thread has not been terminated and time out.

1.2.2.

Thread.IsAlive

Reference:

https://msdn.microsoft.com/en-us/library/system.threading.thread.isalive(v=vs.110).aspx

Return true if this thread has been started

and has not terminated normally or aborted;

otherwise, false.

2.

//Stopwatch stopwatch = Stopwatch.StartNew();

//stopwatch.Stop();

//stopwatch.ElapsedMilliseconds

will return the timespan by milliseconds

3.

Get the number of CPU cores.

3.1.

See the Task Manager.

3.2.

In .Net,

Environment.ProcessorCount

3.3.

In command line,

echo %NUMBER\_OF\_PROCESSORS%

=============================================

1. New Project

1.1. Create New Project : Sample

File --> New --> Project... -->

Visual C# -->  **Console App** **(.Net Framework)** -->

Name: **Sample**







=============================================

2. Sample : Program.cs

using System;

using System.Diagnostics;

using System.Threading;

using OnlineGame;

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

            ////1. =========================================================

            ////No Task, No Thread

            //Console.WriteLine("1. RunSlow() ; No Task, No Thread =================");

            //RunSlow();

            ////2. =========================================================

            ////Thread

            //Console.WriteLine("2. RunSlow2(); Thread =================");

            //RunSlow2();

            ////3. =========================================================

            ////Parameterized Thread

            //Console.WriteLine("3. ParameterizedThread();  Parameterized Thread =================");

            //ParameterizedThread();

            //4. =========================================================

            //Thread Join(), IsAlive.

            ////4.1. -----------------------------------------------------

            //Console.WriteLine("4.1. ThreadJoinAndIsAlive();    Thread Join(), IsAlive. =================");

            //ThreadJoinAndIsAlive();

            //4.2. -----------------------------------------------------

            Console.WriteLine("4.2. ThreadJoinAndIsAlive2();   Thread Join(), IsAlive. =================");

            ThreadJoinAndIsAlive2();

            Console.ReadLine();

        }

        //1. =========================================================

        //No Task, No Thread

        static void RunSlow()

        {

            Stopwatch stopwatch = Stopwatch.StartNew();

            slowMethodA();

            slowMethodB();

            slowMethodC();

            slowMethodD();

            new Gamer().RunSlowE();

            Console.WriteLine($"Environment.ProcessorCount == {Environment.ProcessorCount} ; this is the number of CPU cores");

            stopwatch.Stop();

            Console.WriteLine($"RunSlow stopwatch.ElapsedMilliseconds : {stopwatch.ElapsedMilliseconds} milliseconds");

        }

        //Stopwatch stopwatch = Stopwatch.StartNew();

        //stopwatch.Stop();

        //stopwatch.ElapsedMilliseconds

        //will return the timespan by milliseconds

        static void slowMethodA()

        {

            Console.WriteLine("beginning of slowMethodA()");

            // Sleep for N million seconds.

            Thread.Sleep(1999);

            Console.WriteLine("End of slowMethodA()");

        }

        static void slowMethodB()

        {

            Console.WriteLine("beginning of slowMethodB()");

            // Sleep for N million seconds.

            Thread.Sleep(2000);

            Console.WriteLine("End of slowMethodB()");

        }

        static void slowMethodC()

        {

            Console.WriteLine("beginning of slowMethodC()");

            // Sleep for N million seconds.

            Thread.Sleep(1998);

            Console.WriteLine("End of slowMethodC()");

        }

        static void slowMethodD()

        {

            Console.WriteLine("beginning of slowMethodD()");

            // Sleep for N million seconds.

            Thread.Sleep(1997);

            Console.WriteLine("End of slowMethodD()");

        }

        //2. =========================================================

        //Thread

        //Tread parameter can be Anonymous methods,

        //static, non-static method, or more.

        //2.1. --------------------------------------------------------

        static void RunSlow2()

        {

            Stopwatch stopwatch = Stopwatch.StartNew();

            Thread slowMethodAThread = new Thread(new ThreadStart(slowMethodA));

            slowMethodAThread.Start();

            Thread slowMethodBThread = new Thread(slowMethodB);

            slowMethodBThread.Start();

            Thread slowMethodCThread = new Thread(delegate () { slowMethodC(); });

            slowMethodCThread.Start();

            Thread slowMethodDThread = new Thread(() => slowMethodD());

            slowMethodDThread.Start();

            Thread slowMethodEThread = new Thread(new Gamer().RunSlowE);

            slowMethodEThread.Start();

            stopwatch.Stop();

            Console.WriteLine($"RunSlow2 stopwatch.ElapsedMilliseconds : {stopwatch.ElapsedMilliseconds} milliseconds");

        }

        //3. =========================================================

        //Parameterized Thread

        static void printIntInput(object input)

        {

            Console.WriteLine($"Beginning of static void printIntInput(object input), input is {input} -----------");

            Console.WriteLine(

                int.TryParse(input.ToString(), out int intInput) ?

                $"intInput == {intInput}" :

                $"intInput is not int");

            Console.WriteLine($"End of static void printIntInput(object input), input is {input} ----------");

        }

        static void ParameterizedThread()

        {

            //3.1. ----------------------------------

            // Create an instance ParameterizedThreadStart delegate

            // with static printIntInput() method as parameter.

            // ParameterizedThreadStart delegate has only one Object type parameter.

            // Thus, printIntInput(object input) need to match it.

            Thread parameterizedThread = new Thread(new ParameterizedThreadStart(printIntInput));

            //Now, parameterizedThread represent a Thread to run printIntInput(object input).

            //So we need to pass an object into it.

            //Here, 7 is int, but it need to be casted to object type

            //in order to pass into printIntInput(object input).

            parameterizedThread.Start((object)7);

            //3.2. ----------------------------------

            // The previous code can be rewrited as the following.

            //The .Net will automatically convert new Thread(printIntInput)

            //to new Thread(new ParameterizedThreadStart(printIntInput)).

            // In addition, it will automatically convert parameterizedThread2.Start(8)

            //to parameterizedThread.Start((object)8)

            Thread parameterizedThread2 = new Thread(printIntInput);

            parameterizedThread2.Start(8);

            //ParameterizedThreadStart delegate and Thread.Start(Object) is not type safe.

            //It will have a compiler error

            //if you pass (object)"AAA" into printIntInput(object input).

            //Thread and Mutex is a little bit out of date.

            //Using Async and Await is better.

            //But it is still good to understand Thread concept.

        }

        //4. =========================================================

        //Thread Join(), IsAlive.

        //4.1. ---------------------------------------------------

        static void ThreadJoinAndIsAlive()

        {

            Console.WriteLine("Beginning of ThreadJoinAndIsAlive() ----------------- ");

            Thread slowMethodAThread = new Thread(slowMethodA);

            slowMethodAThread.Start();

            Thread slowMethodBThread = new Thread(slowMethodB);

            slowMethodBThread.Start();

            for (int i = 1; i <= 10; i++)

            {

                Console.WriteLine($"slowMethodAThread.IsAlive : {slowMethodAThread.IsAlive}");

                if (slowMethodAThread.IsAlive)

                    Thread.Sleep(300);

                else

                    break;

            }

            Console.WriteLine(slowMethodAThread.Join(1000) ?

                "slowMethodAThread has finished" :

                "slowMethodAThread is time out after 1 second.");

            //Return true if the thread has been terminated;

            //Return false if the thread has not been terminated and time out.

            slowMethodBThread.Join();

            Console.WriteLine("slowMethodBThread has finished");

            Console.WriteLine("End of ThreadJoinAndIsAlive() ----------------- ");

        }

        //4.2. ---------------------------------------------------

        static void ThreadJoinAndIsAlive2()

        {

            Console.WriteLine("Beginning of ThreadJoinAndIsAlive2() ----------------- ");

            Thread slowMethodAThread = new Thread(slowMethodA);

            slowMethodAThread.Start();

            Thread slowMethodBThread = new Thread(slowMethodB);

            slowMethodBThread.Start();

            Console.WriteLine(slowMethodAThread.Join(1000) ?

                "slowMethodAThread has finished" :

                "slowMethodAThread is time out after 1 second.");

            slowMethodBThread.Join();

            Console.WriteLine("slowMethodBThread has finished");

            for (int i = 1; i <= 10; i++)

            {

                Console.WriteLine($"slowMethodAThread.IsAlive : {slowMethodAThread.IsAlive}");

                if (slowMethodAThread.IsAlive)

                    Thread.Sleep(300);

                else

                    break;

            }

            Console.WriteLine("End of ThreadJoinAndIsAlive2() ----------------- ");

        }

        //1.

        //ThreadObject.Join()

        //ThreadObject.Join(int millisecondsTimeout)

        //ThreadObject.Join(TimeSpan timeout)

        //Reference:

        //[https://msdn.microsoft.com/en-us/library/system.threading.thread.join(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/system.threading.thread.join%28v=vs.110%29.aspx)

        //Blocks the calling thread until ThreadObject terminates or until timeout.

        //Return true if the thread has been terminated;

        //Return false if the thread has not been terminated and time out.

        //2.

        //Thread.IsAlive

        //Reference:

        //[https://msdn.microsoft.com/en-us/library/system.threading.thread.isalive(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/system.threading.thread.isalive%28v=vs.110%29.aspx)

        //Return true if this thread has been started

        //and has not terminated normally or aborted;

        //otherwise, false.

    }

}

//1. ========================================================

namespace OnlineGame

{

    public class Gamer

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public override string ToString()

        {

            return $"Id=={Id} ; Name=={Name}";

        }

        public void RunSlowE()

        {

            Console.WriteLine("beginning of slowMethodE()");

            // Sleep for N million seconds.

            Thread.Sleep(1996);

            Console.WriteLine("End of slowMethodE()");

        }

    }

}

/\*

1.

Thread

1.1.

In computing, a process is an instance of

a computer program that is being executed.

Windows Task Manager provide limit functions

to control the process in Windows operation system.

A process has one main thread and

might have some other threads.

Each tread executes the different piece of code.

For a single core machine,

Asynchronous Programming might reduce performance

because of context-switching.

However, Most machine nowadays normally have multiple cores CPU

which allows Asynchronous Programming provide a way

to run thread/Tasks simultaneously.

1.2.

Thread member.

1.2.1.

ThreadObject.Join()

ThreadObject.Join(int millisecondsTimeout)

ThreadObject.Join(TimeSpan timeout)

Reference:

[https://msdn.microsoft.com/en-us/library/system.threading.thread.join(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/system.threading.thread.join%28v=vs.110%29.aspx)

Blocks the calling thread until ThreadObject terminates or until timeout.

Return true if the thread has been terminated;

Return false if the thread has not been terminated and time out.

1.2.2.

Thread.IsAlive

Reference:

[https://msdn.microsoft.com/en-us/library/system.threading.thread.isalive(v=vs.110).aspx](https://msdn.microsoft.com/en-us/library/system.threading.thread.isalive%28v=vs.110%29.aspx)

Return true if this thread has been started

and has not terminated normally or aborted;

otherwise, false.

----------------------------

2.

//Stopwatch stopwatch = Stopwatch.StartNew();

//stopwatch.Stop();

//stopwatch.ElapsedMilliseconds

will return the timespan by milliseconds

----------------------------

3.

Get the number of CPU cores.

3.1.

See the Task Manager.

3.2.

In .Net,

Environment.ProcessorCount

3.3.

In command line,

echo %NUMBER\_OF\_PROCESSORS%

\*/

/\*

1. RunSlow() ; No Task, No Thread =================

beginning of slowMethodA()

End of slowMethodA()

beginning of slowMethodB()

End of slowMethodB()

beginning of slowMethodC()

End of slowMethodC()

beginning of slowMethodD()

End of slowMethodD()

beginning of slowMethodE()

End of slowMethodE()

Environment.ProcessorCount == 8 ; this is the number of CPU cores

RunSlow stopwatch.ElapsedMilliseconds : 9993 milliseconds

==================

每次跑的時候, 肯定是ABCDE順序不變

\*/

/\*

2. RunSlow2(); Thread =================

beginning of slowMethodB()

beginning of slowMethodA()

beginning of slowMethodC()

beginning of slowMethodD()

RunSlow2 stopwatch.ElapsedMilliseconds : 26 milliseconds

beginning of slowMethodE()

End of slowMethodB()

End of slowMethodA()

End of slowMethodD()

End of slowMethodE()

End of slowMethodC()

========================

每次跑的時候ABCDE順序都不一樣

\*/

/\*

3. ParameterizedThread();  Parameterized Thread =================

Beginning of static void printIntInput(object input), input is 8 -----------

intInput == 8

End of static void printIntInput(object input), input is 8 ----------

Beginning of static void printIntInput(object input), input is 7 -----------

intInput == 7

End of static void printIntInput(object input), input is 7 ----------

==============

每次跑的時候，都不一樣，有可能8在前面，有可能7在前面

\*/

/\*

4.1. ThreadJoinAndIsAlive();    Thread Join(), IsAlive. =================

Beginning of ThreadJoinAndIsAlive() -----------------

slowMethodAThread.IsAlive : True

beginning of slowMethodA()

beginning of slowMethodB()

slowMethodAThread.IsAlive : True

slowMethodAThread.IsAlive : True

slowMethodAThread.IsAlive : True

slowMethodAThread.IsAlive : True

slowMethodAThread.IsAlive : True

slowMethodAThread.IsAlive : True

End of slowMethodA()

End of slowMethodB()

slowMethodAThread.IsAlive : False

slowMethodAThread has finished

slowMethodBThread has finished

End of ThreadJoinAndIsAlive() -----------------

=============================

A.

slowMethodAThread.IsAlive 是在for loop裡面

他會每隔300ms檢查一次

slowMethodAThread.IsAlive ()

看看是否slowMethodAThread還活著

-----------------

B.

我們知道slowMethodA裡面有

Thread.Sleep(1999);

所以slowMethodA的執行時間會比1999ms還要多一點

然後每隔300ms檢查一次slowMethodAThread.IsAlive ()

所以slowMethodAThread.IsAlive : True

大概會執行7次

因為 300ms \* 7 = 2100ms

估計大概是6~7次

-----------------

C.

在的for loop的時候

要到slowMethodAThread.IsAlive : False的時候

也就是當要到slowMethodAThread結束的時候

才會脫離for loop

接著, 才執行

Console.WriteLine(slowMethodAThread.Join(1000) ?

"slowMethodAThread has finished" :

"slowMethodAThread is time out after 1 second.");

首先Join(timeout)的用法是

Return true if the thread has been terminated;

Return false if the thread has not been terminated and time out.

所以, 不管你使用.Join(1000)或是.Join(1)

也就是不管你等待1000ms 或是 等待 1ms

都會是return true.

因為slowMethodAThread早就結束了

\*/

/\*

4.2. ThreadJoinAndIsAlive2();   Thread Join(), IsAlive. =================

Beginning of ThreadJoinAndIsAlive2() -----------------

beginning of slowMethodA()

beginning of slowMethodB()

slowMethodAThread is time out after 1 second.

End of slowMethodB()

End of slowMethodA()

slowMethodBThread has finished

slowMethodAThread.IsAlive : False

End of ThreadJoinAndIsAlive2() -----------------

=============================

A.

我們知道slowMethodA裡面有

Thread.Sleep(1999);

所以slowMethodA的執行時間會比1999ms還要多一點

接著, 才執行

Console.WriteLine(slowMethodAThread.Join(1000) ?

"slowMethodAThread has finished" :

"slowMethodAThread is time out after 1 second.");

首先Join(timeout)的用法是

Return true if the thread has been terminated;

Return false if the thread has not been terminated and time out.

所以 Join(1000) 意思是

等待1000ms後，

因為 slowMethodA 還沒執行完畢

所以產生timeout (過時)

所以 Join(1000) 會Return false

因此會WriteLine出

"slowMethodAThread is time out after 1 second."

至此

slowMethodA已經因為timeout而結束執行

所以WriteLine出

"End of slowMethodA()"

-----------------

B.

slowMethodAThread.IsAlive 是在for loop裡面

他會每隔300ms檢查一次

slowMethodAThread.IsAlive ()

看看是否slowMethodAThread還活著

但是, 現在問題是

在還沒進入for loop之前

slowMethodA已經因為timeout而結束執行

所以一進入for loop的時候

第一輪的時候

馬上WriteLine出

"slowMethodAThread.IsAlive : False"

\*/

3. Console App(.NET Framework) - SampleV2 - Program.cs

----------------------------------------------------------------------------------------------------------------

**Console App(.NET Framework) - SampleV2**

----------------------------------------------------------------------------------------------------------------

1.

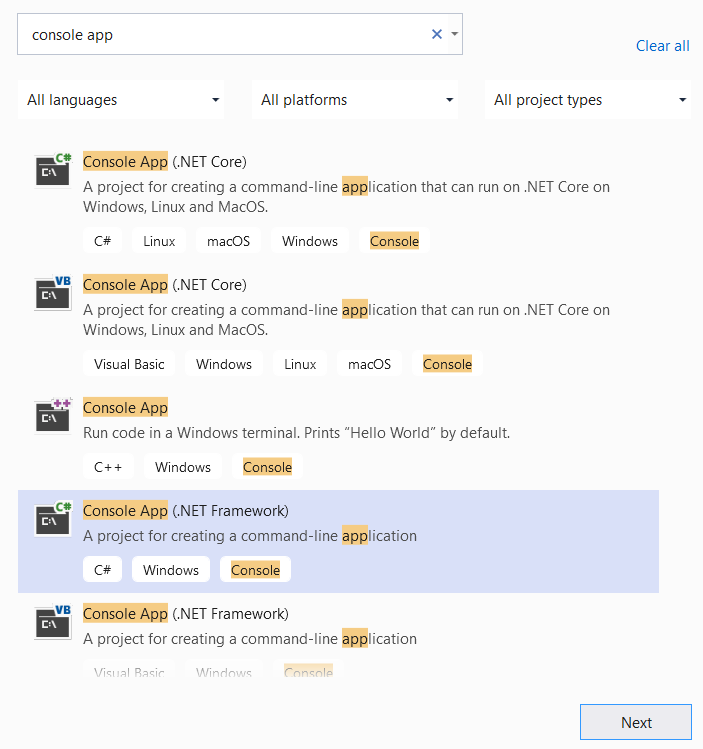
<https://www.facebook.com/groups/934567793358849/posts/1979140725568212/>

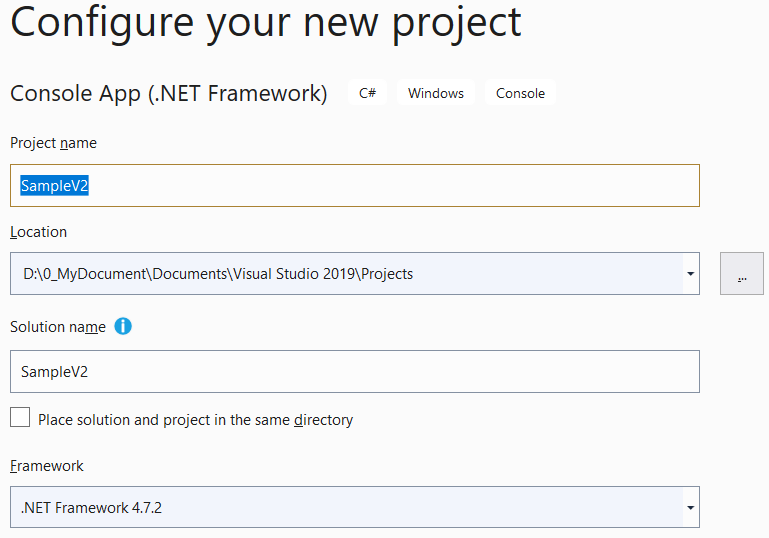
Console App(.NET Framework)

-->

SampleV2

----------------------------------------------------------------------------------------





----------------------------------------------------------------------------------------------------------------

2.

Program.cs

---------------------------------------------------------------------------------------------------

using System;

using System.Threading;

namespace SampleV2

{

    class Program

    {

        static void Main(string[] args)

        {

            //5. =========================================================

            //Thread Join()

            ////5.1. -----------------------------------------------------

            Console.WriteLine("5.1. ThreadJoin1(); A.Start();A.Join();B.Start();B.Join(); =================");

            ThreadJoin1();

            //5.2. -----------------------------------------------------

            Console.WriteLine("5.2. ThreadJoin2(); A.Start();B.Start();A.Join();B.Join(); =================");

            ThreadJoin2();

            Console.ReadLine();

        }

        //1. =========================================================

        //No Task, No Thread

        static void slowMethodA()

        {

            Console.WriteLine("beginning of slowMethodA()");

            // Sleep for N million seconds.

            Thread.Sleep(1999);

            Console.WriteLine("End of slowMethodA()");

        }

        static void slowMethodB()

        {

            Console.WriteLine("beginning of slowMethodB()");

            // Sleep for N million seconds.

            Thread.Sleep(2000);

            Console.WriteLine("End of slowMethodB()");

        }

        //5. =========================================================

        //Thread Join()

        static void ThreadJoin1()

        {

            Console.WriteLine("Beginning of ThreadJoin1() ----------------- ");

            Thread slowMethodAThread = new Thread(slowMethodA);

            Thread slowMethodBThread = new Thread(slowMethodB);

            slowMethodAThread.Start();

            slowMethodAThread.Join();

            slowMethodBThread.Start();

            slowMethodBThread.Join();

            Console.WriteLine("End of ThreadJoin1() ----------------- ");

        }

        static void ThreadJoin2()

        {

            Console.WriteLine("Beginning of ThreadJoin2() ----------------- ");

            Thread slowMethodAThread = new Thread(slowMethodA);

            Thread slowMethodBThread = new Thread(slowMethodB);

            slowMethodAThread.Start();

            slowMethodBThread.Start();

            slowMethodAThread.Join();

            slowMethodBThread.Join();

            Console.WriteLine("End of ThreadJoin2() ----------------- ");

        }

    }

}

/\*

5.1. ThreadJoin1(); A.Start();A.Join();B.Start();B.Join(); =================

Beginning of ThreadJoin1() -----------------

beginning of slowMethodA()

End of slowMethodA()

beginning of slowMethodB()

End of slowMethodB()

End of ThreadJoin1() -----------------

5.2. ThreadJoin2(); A.Start();B.Start();A.Join();B.Join(); =================

Beginning of ThreadJoin2() -----------------

beginning of slowMethodA()

beginning of slowMethodB()

End of slowMethodB()

End of slowMethodA()

End of ThreadJoin2() -----------------

\*/

/\*

5.1. ThreadJoin1(); A.Start();A.Join();B.Start();B.Join(); =================

Beginning of ThreadJoin1() -----------------

beginning of slowMethodA()

End of slowMethodA()

beginning of slowMethodB()

End of slowMethodB()

End of ThreadJoin1() -----------------

5.2. ThreadJoin2(); A.Start();B.Start();A.Join();B.Join(); =================

Beginning of ThreadJoin2() -----------------

beginning of slowMethodA()

beginning of slowMethodB()

End of slowMethodA()

End of slowMethodB()

End of ThreadJoin2() -----------------

\*/

/\*

1.

//static void ThreadJoin1()

//{

//    Console.WriteLine("Beginning of ThreadJoin1() ----------------- ");

//    Thread slowMethodAThread = new Thread(slowMethodA);

//    Thread slowMethodBThread = new Thread(slowMethodB);

//    slowMethodAThread.Start();

//    slowMethodAThread.Join();

//    slowMethodBThread.Start();

//    slowMethodBThread.Join();

//    Console.WriteLine("End of ThreadJoin1() ----------------- ");

//}

-----------------------------------------

"ThreadJoin1" is "CurrentThread"

"slowMethodAThread" is "ChildThread" of the "CurrentThread"

"slowMethodBThread" is "ChildThread" of the "CurrentThread"

----------------

//slowMethodAThread.Start();

"CurrentThread" has started first, and then "slowMethodAThread" started.

----------------

//slowMethodAThread.Join();

"slowMethodAThread" join the "CurrentThread".

It means "slowMethodAThread" has to be finished,

and then "CurrentThread" can continue.

----------------

//slowMethodBThread.Start();

While "CurrentThread" is still runing, and then "slowMethodBThread" started,

----------------

//slowMethodAThread.Join();

"slowMethodBThread" join the "CurrentThread".

It means "slowMethodBThread" has to be finished,

and then "CurrentThread" can continue.

---------------------------------

"ThreadJoin1"是"CurrentThread"(目前Thread)

"slowMethodAThread"是"CurrentThread"(目前Thread)的"ChildThread"(子Thread)

"slowMethodBThread"是"CurrentThread"(目前Thread)的"ChildThread"(子Thread)

----------------

//slowMethodAThread.Start();

"CurrentThread"先開始跑,然後"slowMethodAThread"才開始跑

----------------

//slowMethodAThread.Join();

"slowMethodAThread" join the "CurrentThread".

這意思是"slowMethodAThread"必須先結束

"CurrentThread"才可以繼續跑

----------------

//slowMethodBThread.Start();

當"CurrentThread"還在繼續跑,然後"slowMethodBThread"才開始跑

----------------

//slowMethodBThread.Join();

"slowMethodBThread" join the "CurrentThread".

這意思是"slowMethodBThread"必須先結束

"CurrentThread"才可以繼續跑

-------------------------------------------------------------------

2.

static void ThreadJoin2()

//{

//    Console.WriteLine("Beginning of ThreadJoin2() ----------------- ");

//    Thread slowMethodAThread = new Thread(slowMethodA);

//    Thread slowMethodBThread = new Thread(slowMethodB);

//    slowMethodAThread.Start();

//    slowMethodBThread.Start();

//    slowMethodAThread.Join();

//    slowMethodBThread.Join();

//    Console.WriteLine("End of ThreadJoin2() ----------------- ");

//}

-----------------------------------------

"ThreadJoin1" is "CurrentThread"

"slowMethodAThread" is "ChildThread" of the "CurrentThread"

"slowMethodBThread" is "ChildThread" of the "CurrentThread"

----------------

//slowMethodAThread.Start();

"CurrentThread" has started first, and then "slowMethodAThread" started.

----------------

//slowMethodBThread.Start();

While "CurrentThread" is still runing, and then "slowMethodBThread" started,

----------------

//slowMethodAThread.Join();

"slowMethodAThread" join the "CurrentThread".

It means "slowMethodAThread" has to be finished,

and then "CurrentThread" can continue.

However, the "slowMethodBThread" is still runing.

Therefore, "slowMethodBThread" might be finished first.

----------------

//slowMethodAThread.Join();

"slowMethodBThread" join the "CurrentThread".

It means "slowMethodBThread" has to be finished,

and then "CurrentThread" can continue.

---------------------------------

"ThreadJoin1"是"CurrentThread"(目前Thread)

"slowMethodAThread"是"CurrentThread"(目前Thread)的"ChildThread"(子Thread)

"slowMethodBThread"是"CurrentThread"(目前Thread)的"ChildThread"(子Thread)

----------------

//slowMethodAThread.Start();

"CurrentThread"先開始跑,然後"slowMethodAThread"才開始跑

----------------

//slowMethodBThread.Start();

當"CurrentThread"還在繼續跑,然後"slowMethodBThread"才開始跑

----------------

//slowMethodAThread.Join();

"slowMethodAThread" join the "CurrentThread".

這意思是"slowMethodAThread"必須先結束

"CurrentThread"才可以繼續跑

但是其實"slowMethodBThread"還在繼續跑

所以"slowMethodBThread"有可能會先結束(看運氣)

----------------

//slowMethodBThread.Join();

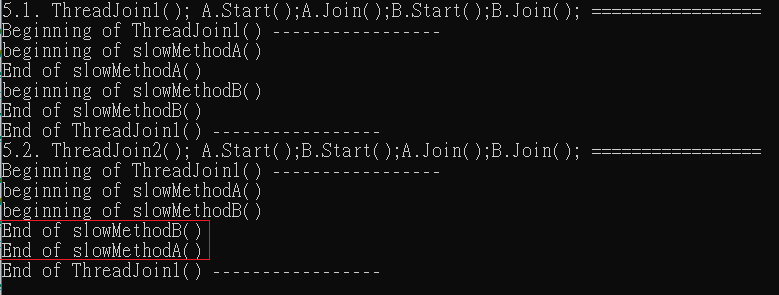
"slowMethodBThread" join the "CurrentThread".

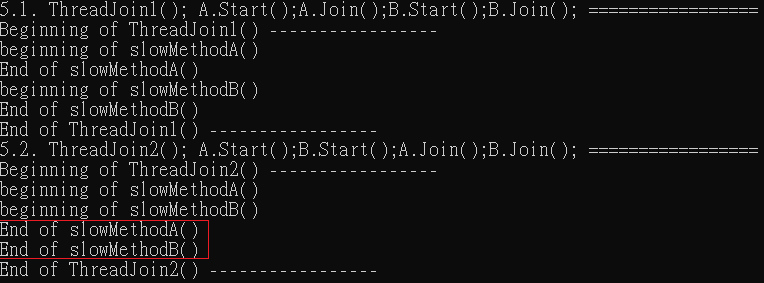
這意思是"slowMethodBThread"必須先結束

"CurrentThread"才可以繼續跑

\*/

----------------------------------------------------------------





----------------------------------------------------------------------------------------------------------------