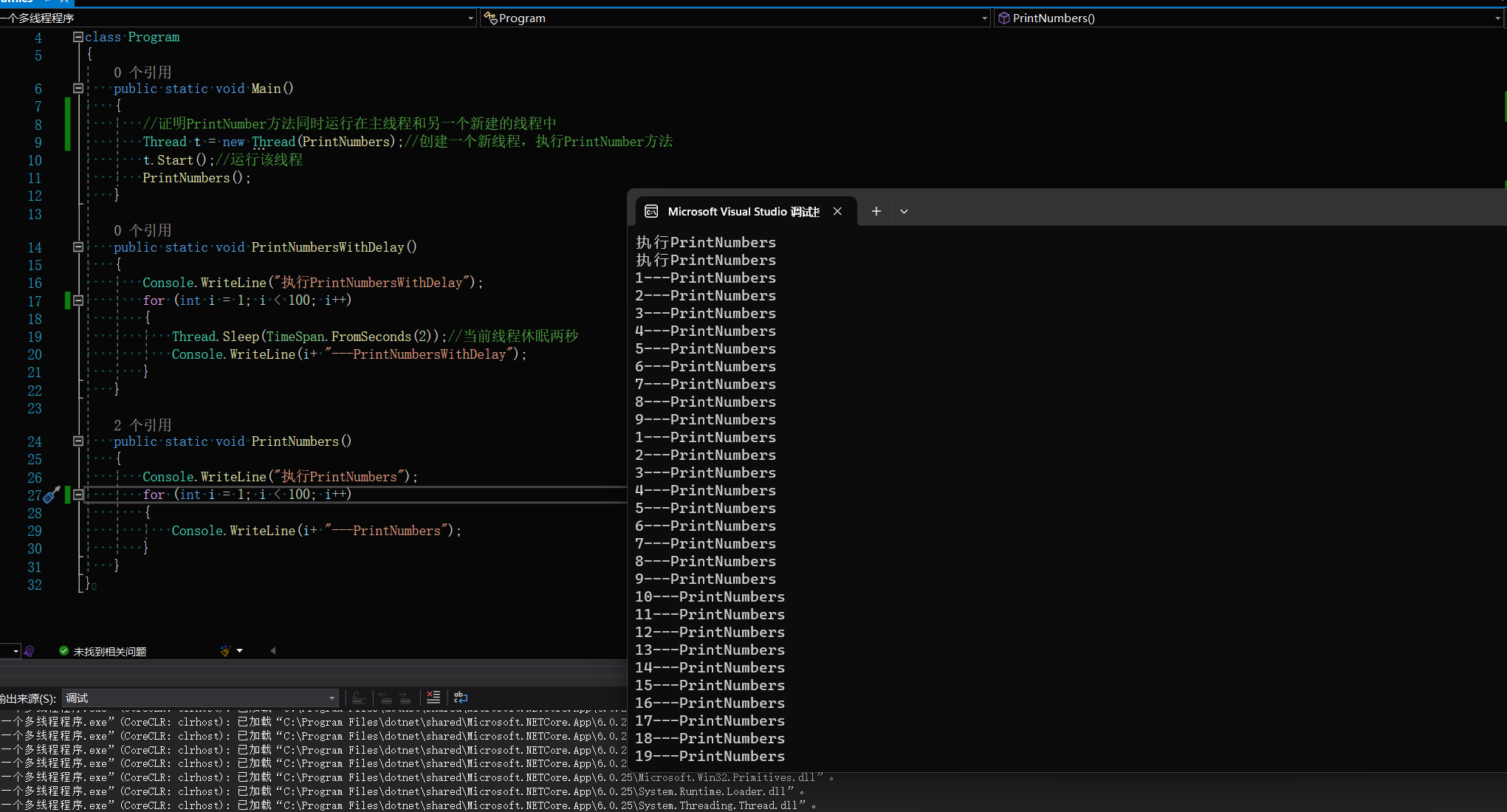
使用C#创建一个多线程程序

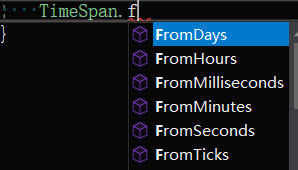
Ps：Task底层是对线程进行了一次封装，功能也更完整，现版本多使用Task

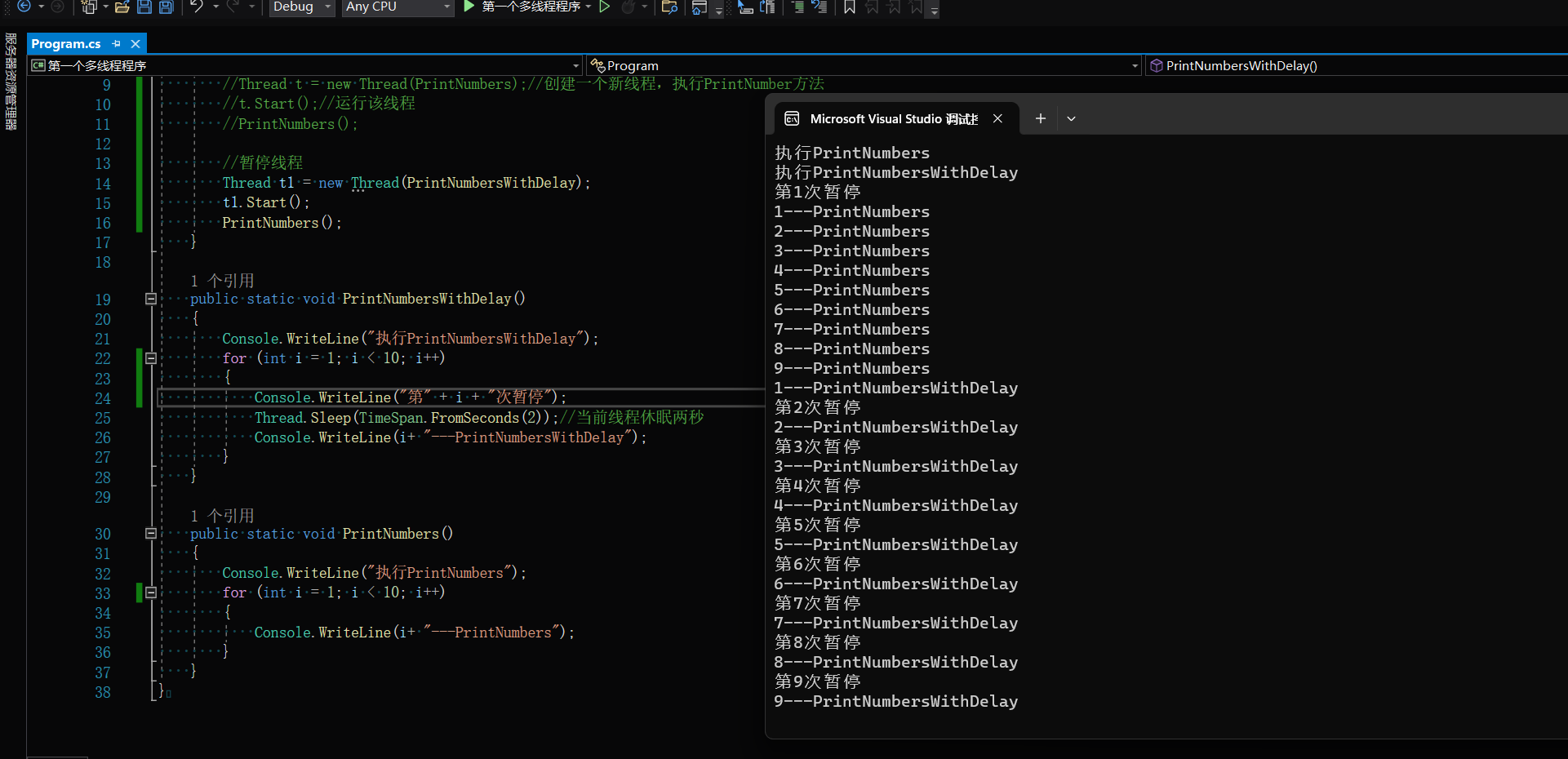
创建线程使用Thread类，构造函数传入该线程执行的方法，实例调用Start方法

交替输出，证明方法是同时运行在主线程和新建线程中

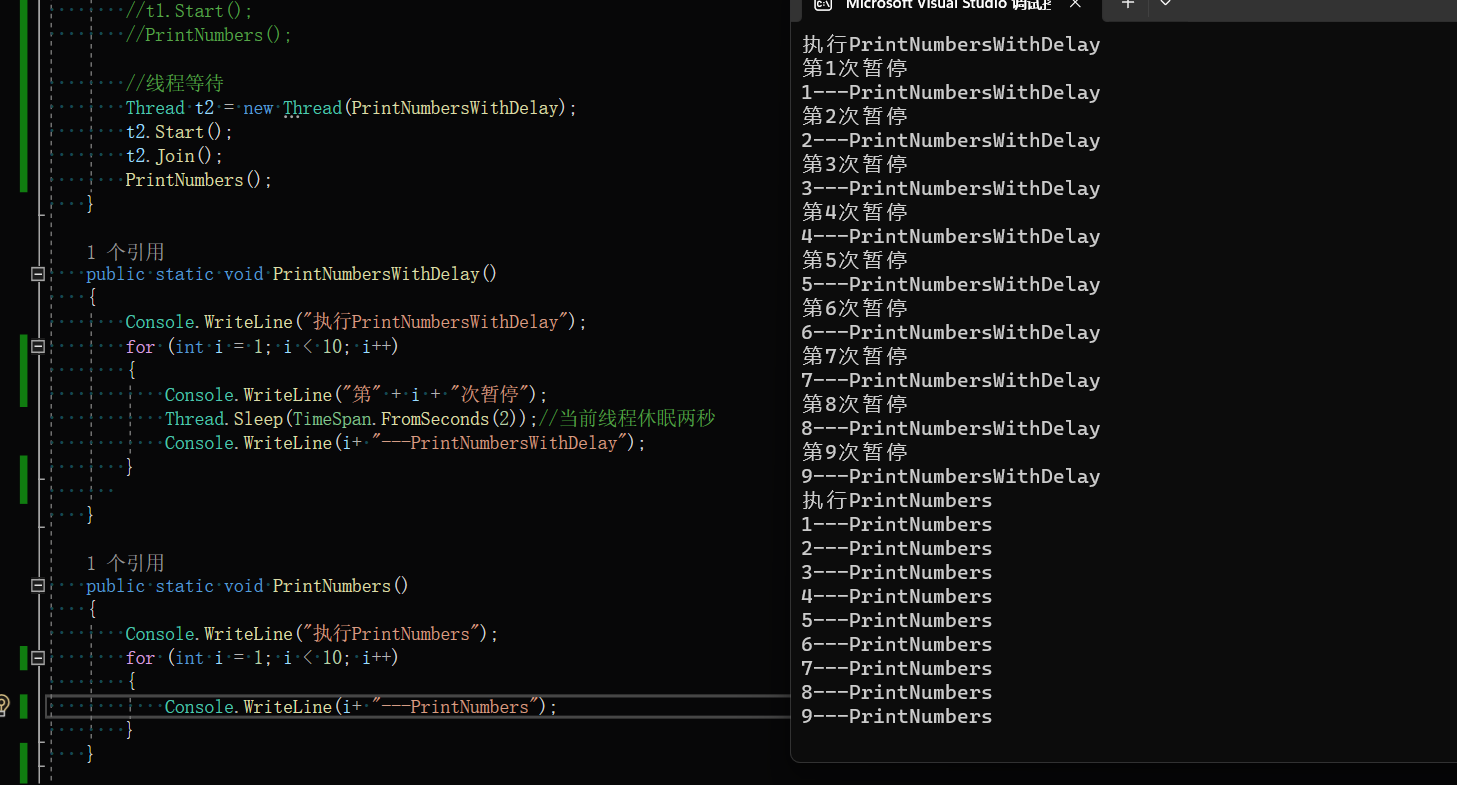


暂停线程的方法：TimeSpan类提供的五个方法



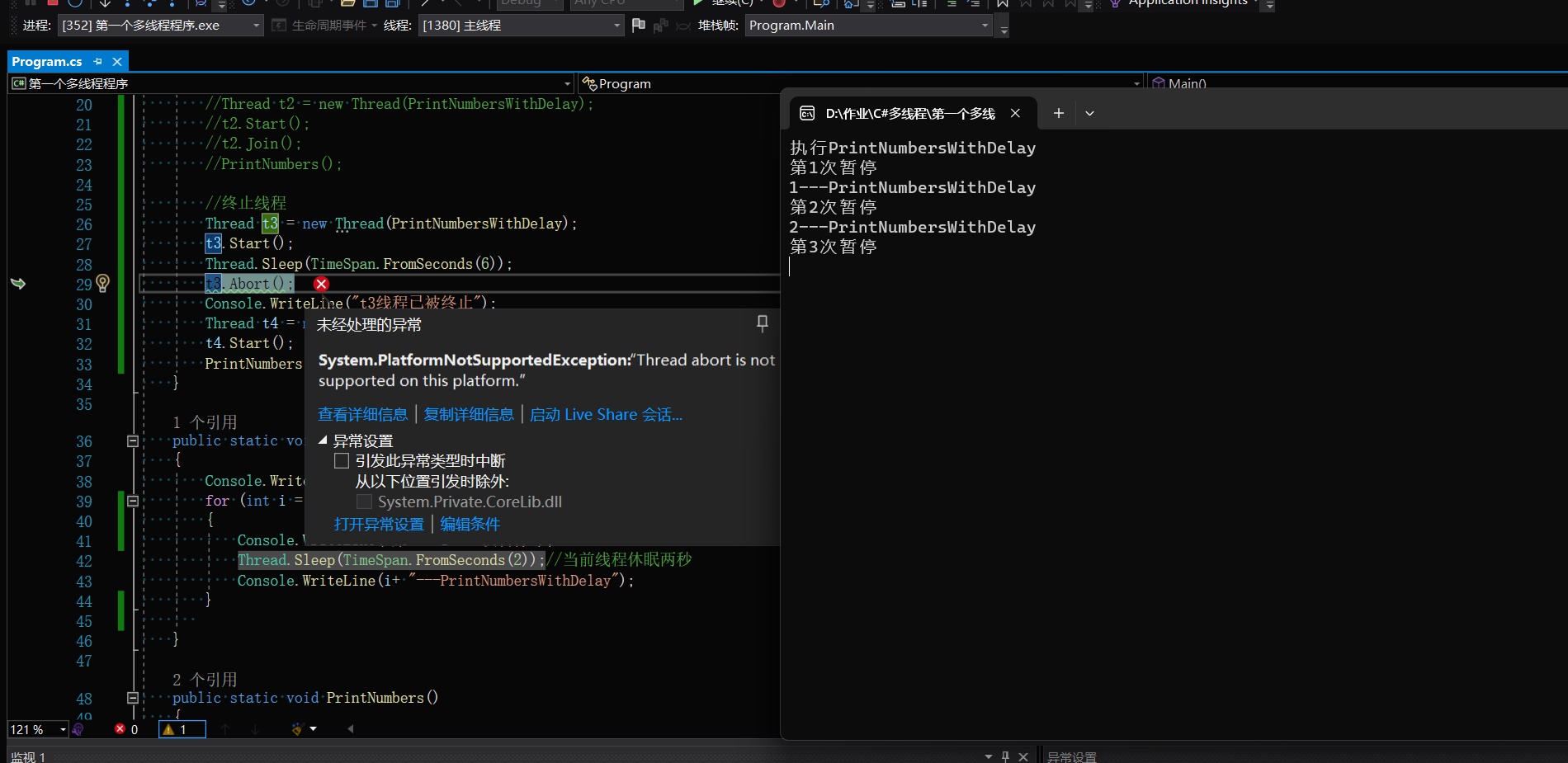


线程等待，使用Join方法完成同步操作，让其他线程在调用Join方法的线程执行完毕后继续执行。反映在下面这个例子中就是主线程在t2线程的方法执行完毕后继续执行



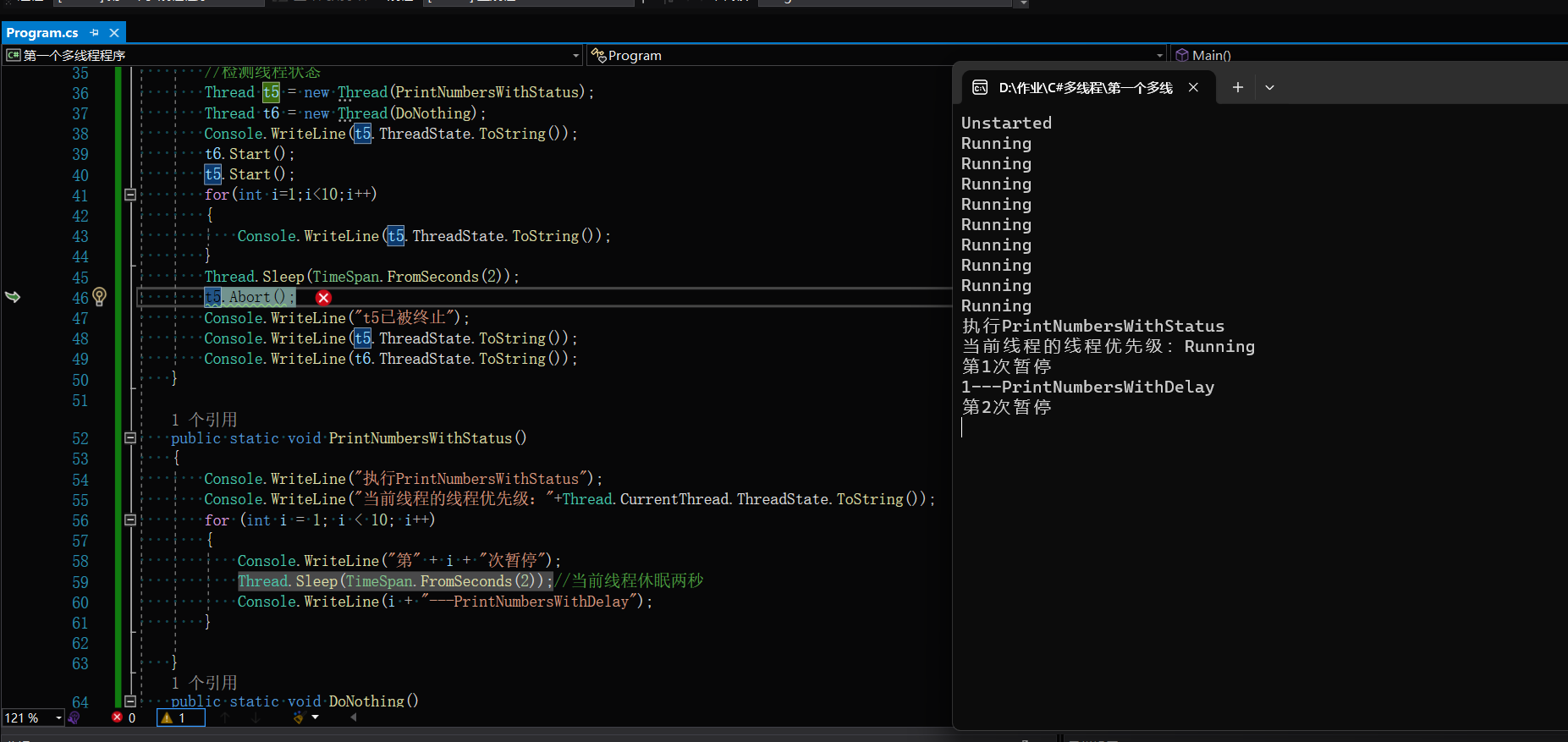
Ps：如果调用Join的线程阻塞了，导致线程永远也无法完成，如果没有超时检测之类的措施，那么整个程序会无限期停止运行。

终止线程（这里是使用Abort方法），抛出此异常的原因是执行Abort方法，立即摧毁一个线程，通过抛出一个异常的方式摧毁一个线程

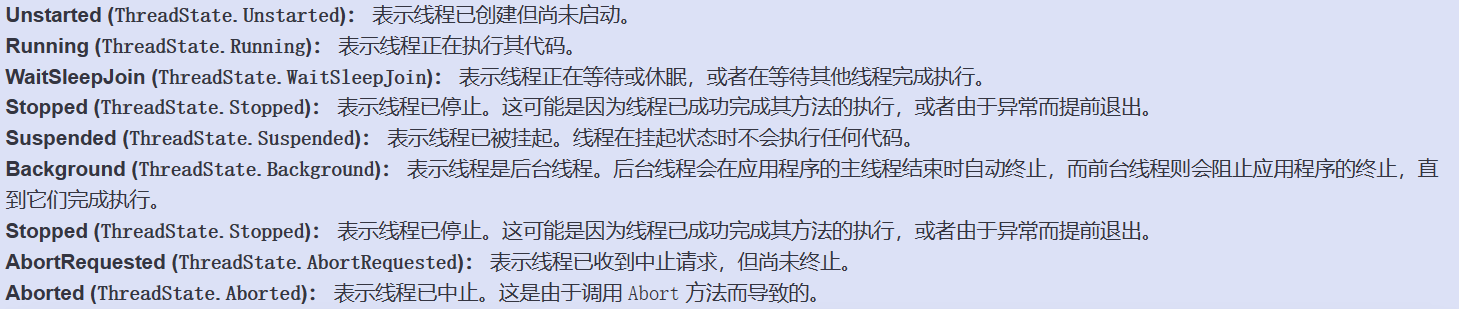


Ps：使用Abort也不一定能终止线程，因为目标线程可以处理这个异常并调用Thread.ResetAbort方法来拒绝被终止

获取线程状态：始终可以通过Thread.CurrentThread获取当前线程，通过当前线程的ThreadState枚举对象可以获取当前线程的状态



Ps:线程状态包含以下这些



线程优先级

文本

描述已自动生成

电脑萤幕画面

描述已自动生成

文本

描述已自动生成

在单核上运行该进程时，高优先级线程运行次数远多于低优先级线程，可以使用Thread.Priority属性设置线程的优先级

Ps：

1. ProcessorAffinity 属性是一个位掩码，用于表示可用的处理器。通过设置不同位的值，可以指定进程可以在哪些核心上运行。在本例中，new IntPtr(1) 将整数 1 转换为 IntPtr 对象，表示在第一个核心上运行。这样设置处理器亲和性后，进程的线程将只能在指定的核心上执行。

2. 在C#中，IntPtr 是一个结构体（struct），用于表示本机大小的有符号整数，其大小与指针或处理器的本机大小相对应。具体来说，IntPtr 可以被用来保存内存地址，也可以用于与非托管代码进行交互，例如在调用 Win32 API 或其他本机库时。

关于线程优先级、线程状态以及进程的更详细的笔记，在《CLR Via C#》的“线程基础“那一章的笔记

前台线程和后台线程：

前台线程和后台线程的区别：应用程序必须运行完所有的前台线程才可以退出；而对于后台线程，应用程序则可以不考虑其是否已经运行完毕而直接退出，所有的后台线程在应用程序退出时都会自动结束。

前台线程和后台线程的使用场景：一般前台线程用于需要长时间等待的任务，比如监听客户端的请求；后台线程一般用于处理时间较短的任务，比如处理客户端发过来的请求信息。例如，通常后台线程非常适合于完成后台任务，应该将被动侦听活动的线程设置为后台线程，而将负责发送数据的线程设置为前台线程，这样，在所有的数据发送完毕之前该线程不会被终止。

图形用户界面, 文本

描述已自动生成

这个例子证明了前台线程执行完毕或终止的时候，后台进程也会随之终止，不管后台线程是否执行完毕。主线程退出，不管后台线程有没有执行完，都会把后台线程结束掉。

Ps：

1. 托管线程池中的线程都是后台线程，使用显示创建如new Thread方式创建的线程默认都是前台线程，主线程是前台线程。

2. 可以将前台线程修改为后台线程，方式是设置Thread.IsBackground 属性为true。

向线程传递参数：

向线程传递参数的办法：1.在Start中传递线程执行的方法所需的对应类型的参数2.使用lambda表达式并在其中调用方法并创建并传递局部参数

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

Ps：使用lambda表达式引用另一个C#对象的方式被称为“闭包“。当在lambda表达式中使用任何局部变量时C#会自动实现一个类，并将该变量作为该类的一个属性（因此lambda表达式可以用于延长局部变量的生存周期）。这样可能会导致的问题，在多个lambda类中使用相同的变量，会共享该变量的值，因此可以在上个例子的最后一部分看到输出i的大小都为20。

使用lock关键字

Lock关键字：使用lock关键字可以锁定一个对象，如果一个线程锁定了一个对象，需要访问该线程的其他所有线程都会处于阻塞状态，并等待直到该对象解除锁定。（但lock关键字会带来很大的性能损耗）

为什么要使用lock关键字：lock关键字可以保证线程安全，当多个线程共享并可以操作同一个实例对象时，如果不使用lock关键字，那么线程是不安全的，各个线程可能会得到错误的操作结果。

下面是在进行加减操作时使用lock关键字的例子：

文本, 信件

描述已自动生成

使用Monitor类锁定资源：

Monitor类的用处：Monitor类拥有一个TryEnter方法，该方法接收一个超时参数，如果在超时参数过期时还没有获得被lock保护的资源，那么该方法返回false。

Monitor类解决的问题：解决死锁问题，如果进行一个操作需要两个lock，分别锁住A和B资源，现在有两个线程一个线程锁住了A另一个线程锁住了B，那么就引发了死锁，两个线程都会被一直阻塞。Monitor类解决死锁问题。

文本

描述已自动生成

文本

描述已自动生成

这个例子开头就锁住lock1，过1秒锁住二。例子的第一部分，使用Monitor类的TryEnter方法，没锁住和锁住都会执行相应的，语句不会死锁；而例子第二部分在执行时若lock2已被锁住则一定会发生死锁线程阻塞，不同于例子的第一部分可以执行另一种情况的语句。

线程中的异常处理：

一般情况下不要在线程中抛出异常，而应该使用try-catch语句去捕获异常。