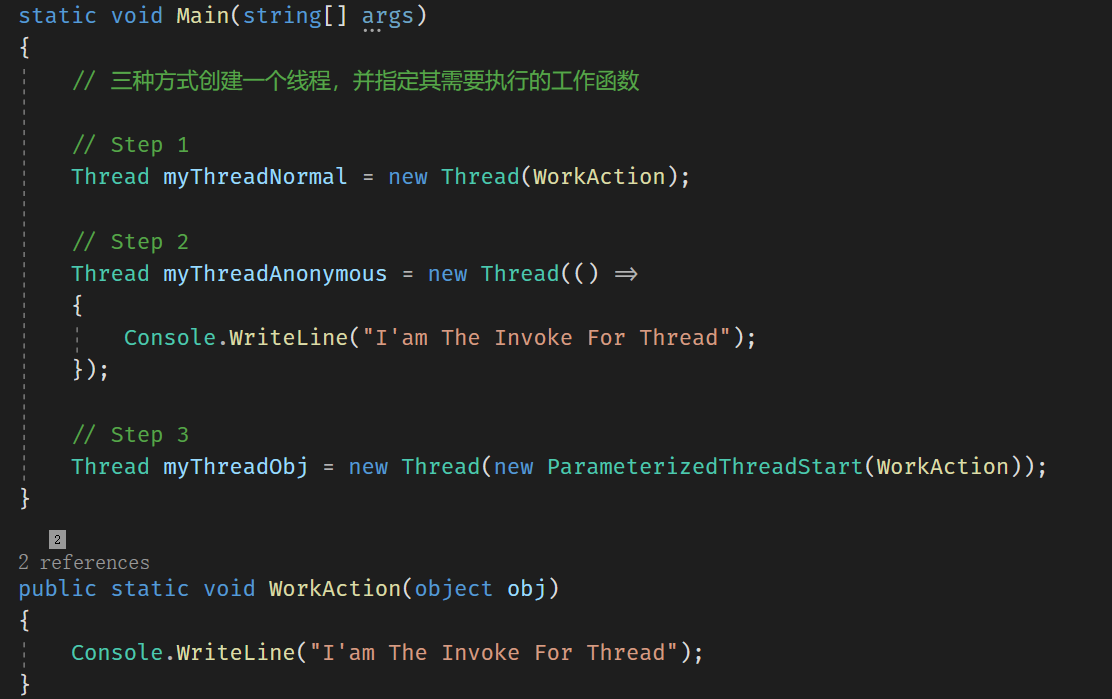
**ASP.NET的处理模型**

**如何在C#中使用多线程**

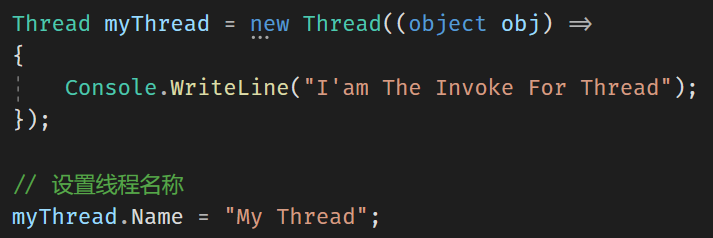
1. **Thread：线程对象**

**-> 如何创建一个Thread**

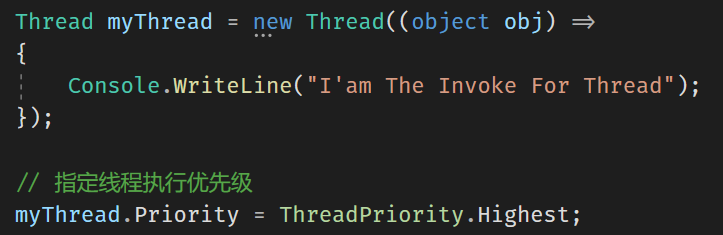


**-> 属性**

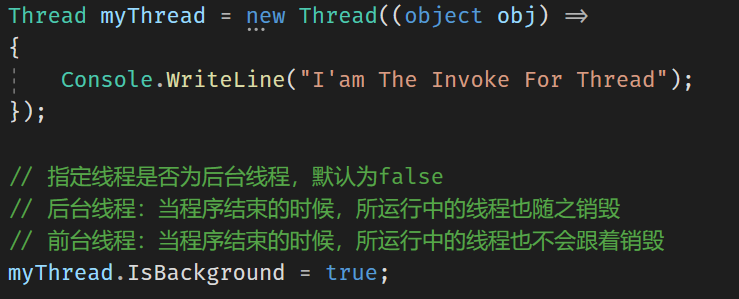
1. Name：获取或设置当前线程的名称



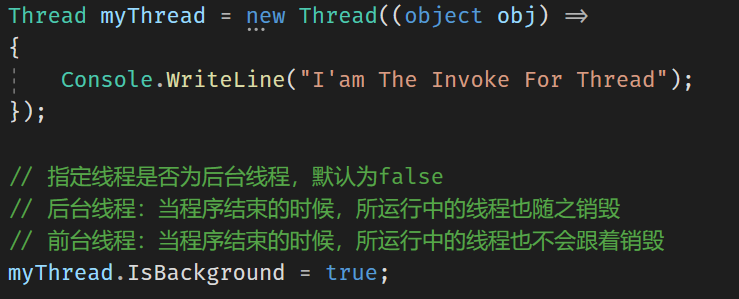
1. Priority：设置当前线程CPU分配的优先级



1. IsBackground：设置当前线程是否为后台线程

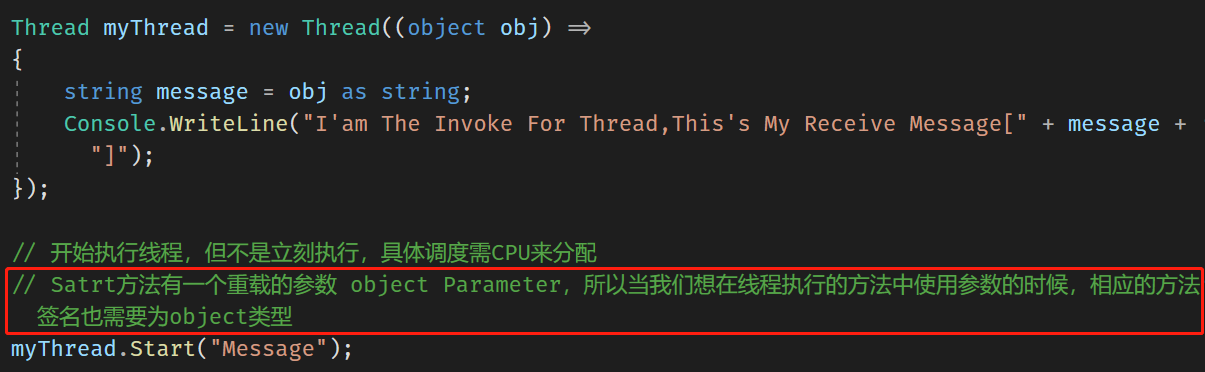


1. Thread.CurrentThread：获取当前正在运行中的线程的实例

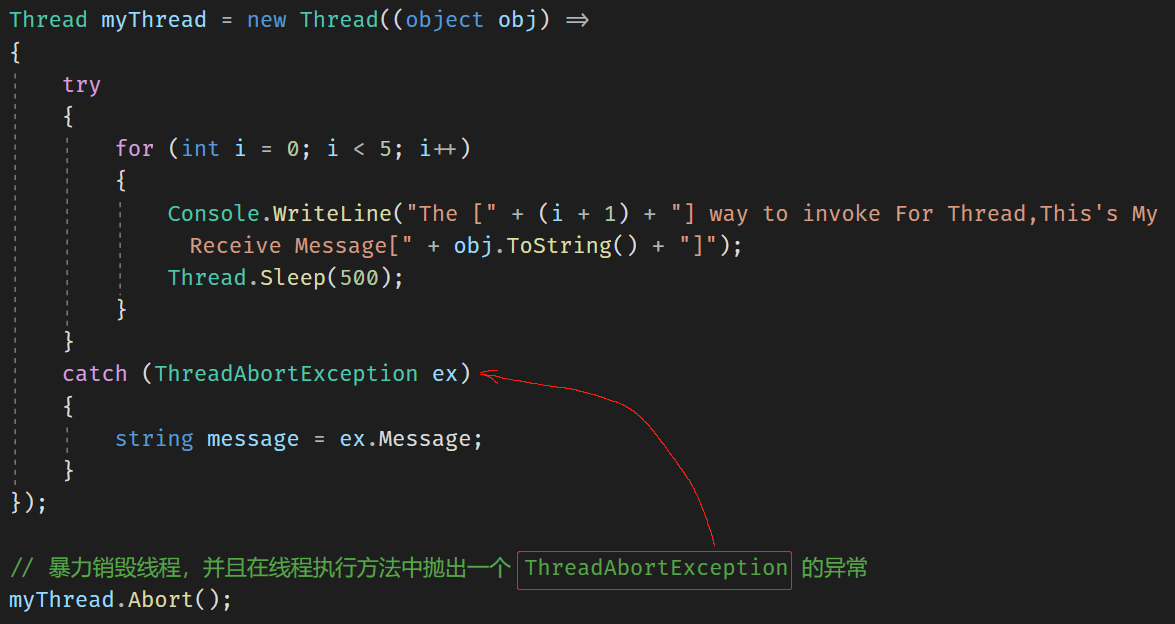


**-> 函数**

1. Start(object Parameter)：开始当前线程的任务，但不是立刻开启，具体的调度由CPU去分配



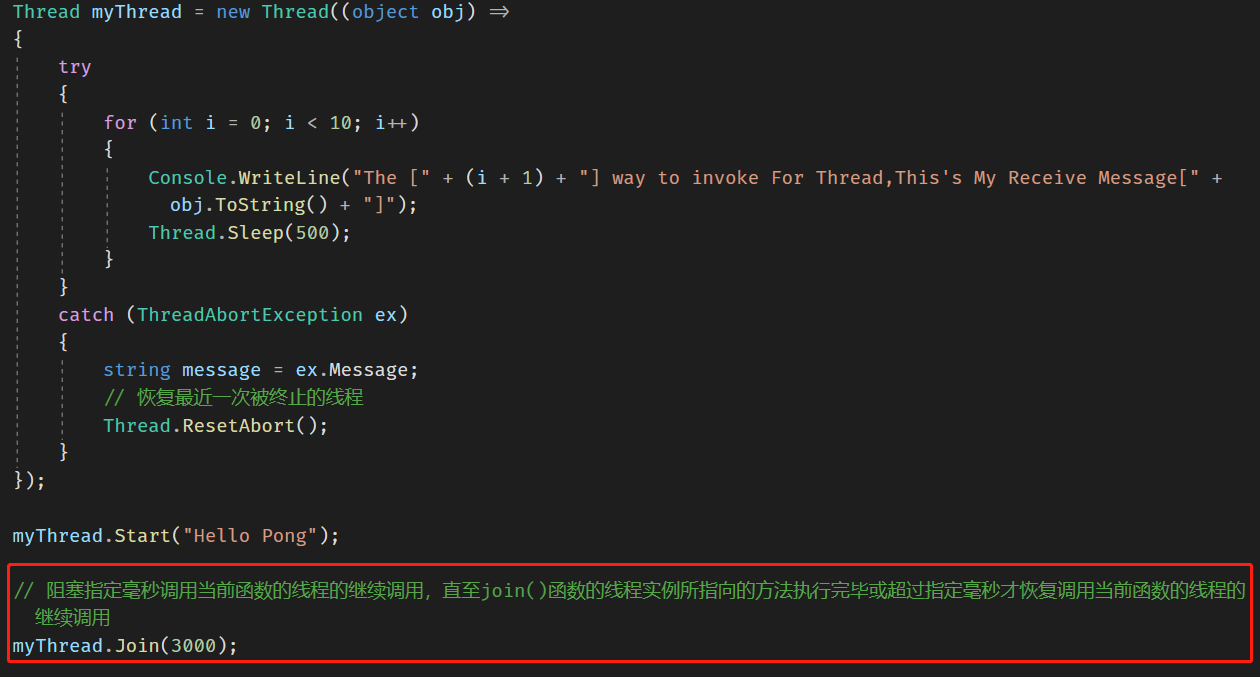
1. Abort()：暴力终止线程，并且抛出一个可捕获的异常



1. ResultAbort()：恢复最近一次被终止的线程的调用



1. Join()：阻塞指定毫秒调用当前函数的线程的继续调用，直至join()函数的线程实例所指向的方法执行完毕或超过指定 毫秒才恢复调用当前函数的线程的继续调用



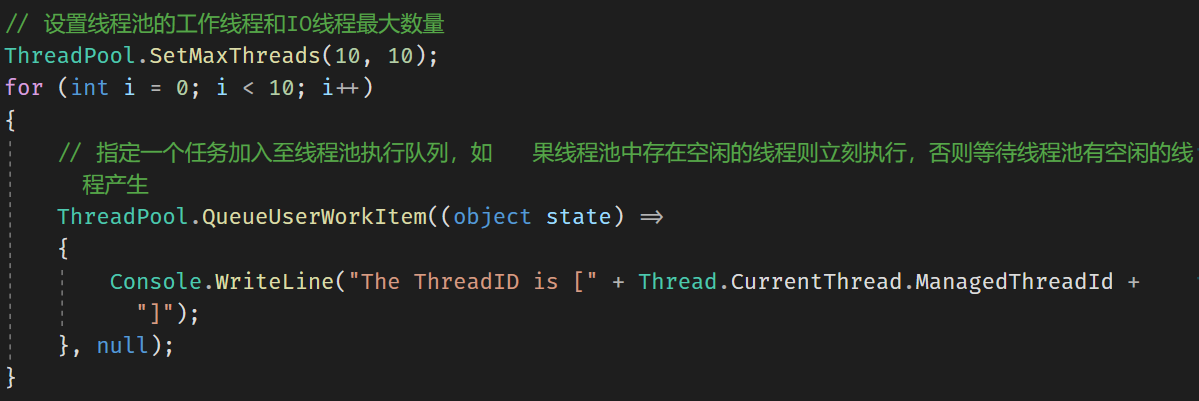
1. Thread.Sleep(int milliSeconds)：当前线程睡眠指定毫秒
2. **ThreadPool：线程池**

**-> 因为线程的开启和注销是很浪费系统资源的（每个线程至少需要1MB的内存开销），所以针对一些需要频繁开启线程且任务时间较少的情况下，我们可以使用线程池技术以提升线程的利用率**

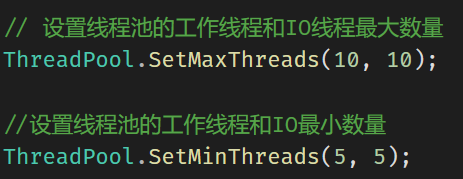
**-> 在线程池中的线程优先级都是【Normal】，并且它们都是【后台线程】，开发人员无法手动的对它们进行调整**

**-> 函数**

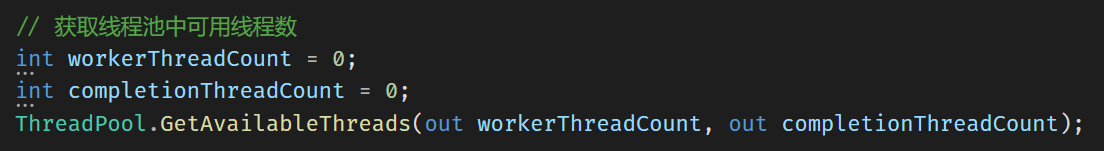
1. QueueUserWorkItem(WaitCallBack callBack,object state)：指定一个任务加入至线程池执行队列，如 果线程池中存在空闲的线程则立刻执行，否则等待线程池有空闲的线程产生



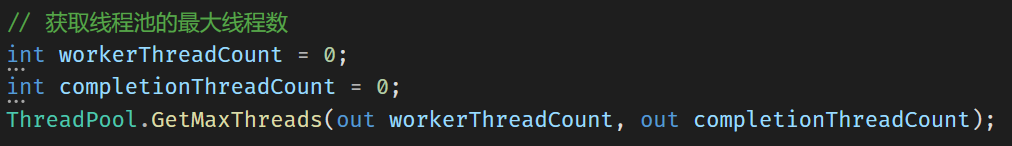
1. SetMax/MinThreads(int workerThreads,int completionPortThreads)：设置线程池的工作线程和IO 线程最大/最 小数量



1. GetAvailableThreads(out int workerThreads, out int completionPortThreads)：获取线程池中可用线程数



1. GetMaxThreads(out int workerThreads, out int completionPortThreads)：获取线程池的最大线程数



1. RegisteredWaitHandle RegisterWaitForSingleObject(WaitHandle waitObject,WaitOrTimerCallback callBack, object state,uint millisecondsTimeOutInterval,bool executeOnlyOnce)

-> 使用线程池中的线程构造一个定时器，每millisecondsTimeOutInterval毫秒在线程池中取一个线程去执行处理函数callback

-> RegisteredWaitHandle：定时器的返回值，可以根据该返回值的[RegisteredWaitHandle.Unregister(WaitHandle waitObject)] 去完成定时器的结束

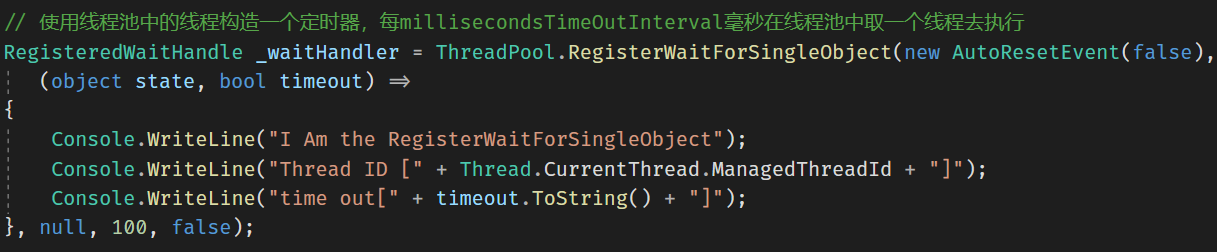
-> WaitHandle waitObject：重置标识

-> WaitOrTimerCallback callBack：处理函数

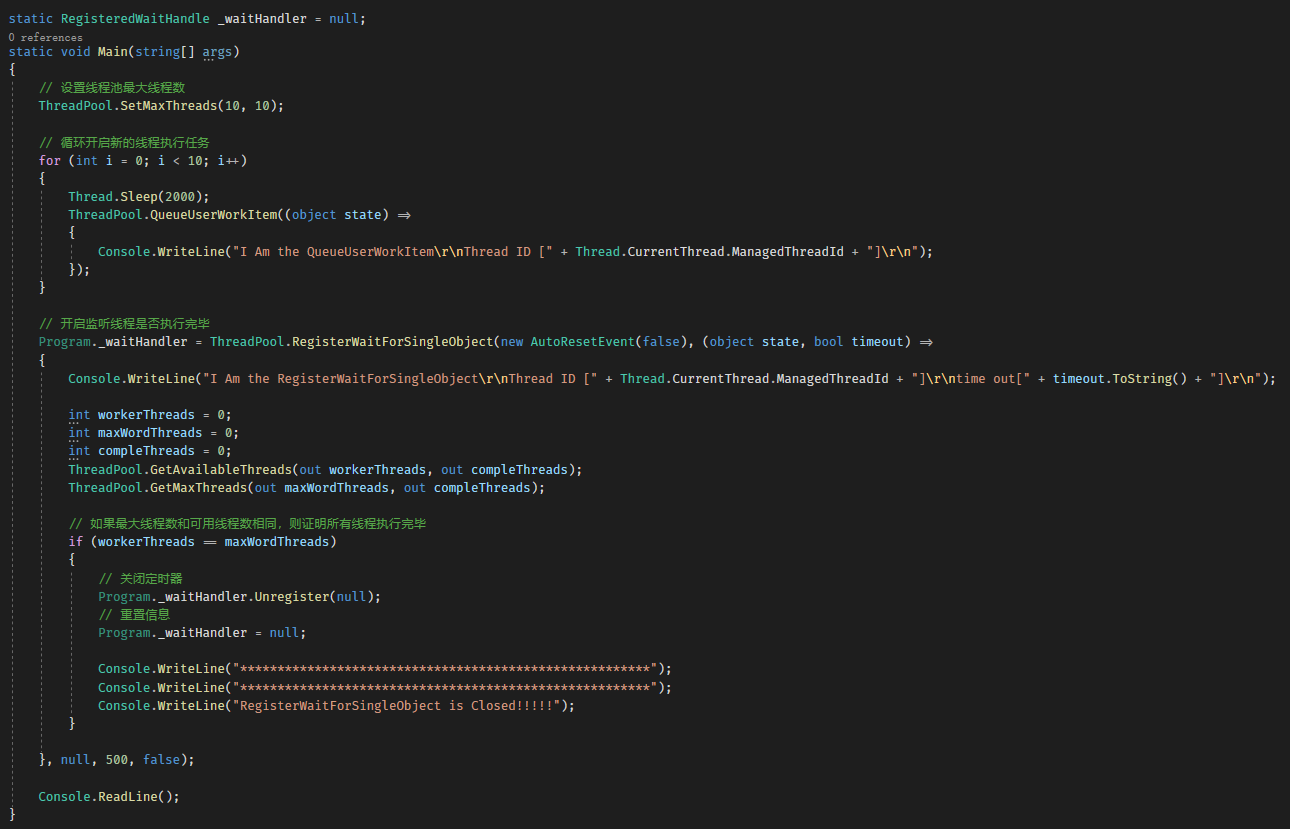
-> object state：处理参数

-> uint millisecondsTimeOutInterval：多少毫秒执行一次

-> bool executeOnlyOnce：是否只执行一次



**-> 判断线程池中的任务是否全部完成**

****

1. **Task：任务**

**[请点击这里！！](任务.md)**

1. **异步委托**

**[请点击这里！！](../../2018年/2月6日/委托、事件/委托的详解.docx)**

1. **Async**

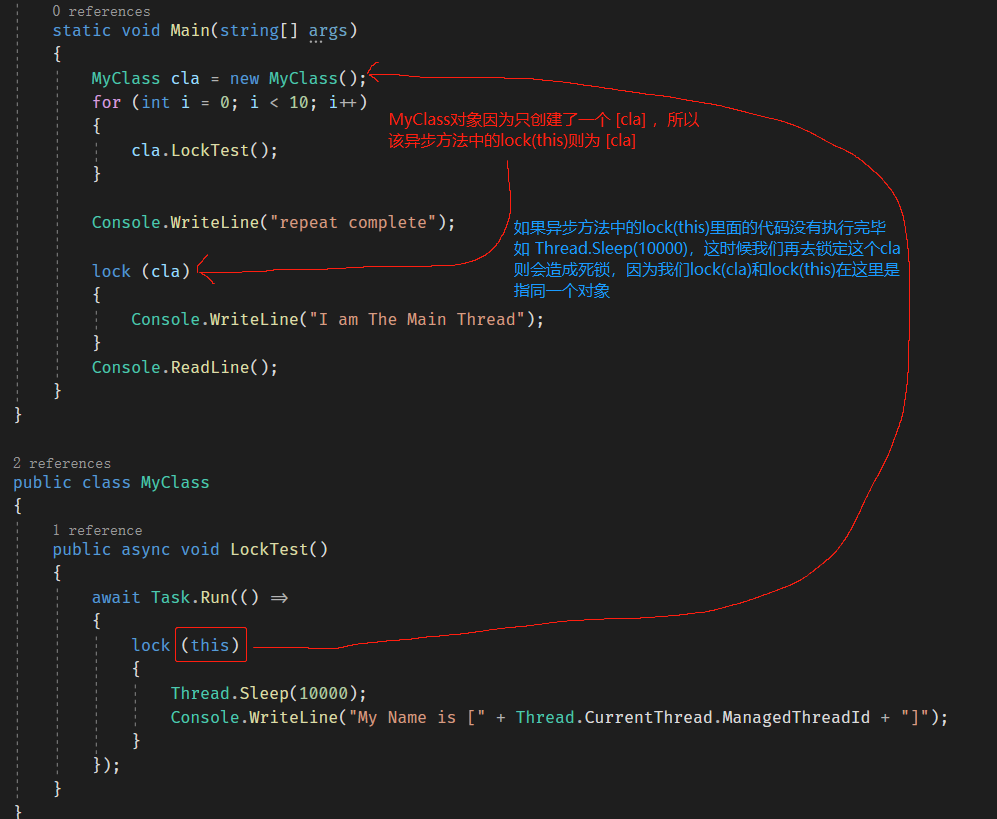
**未完待续。。。**

**线程的同步**

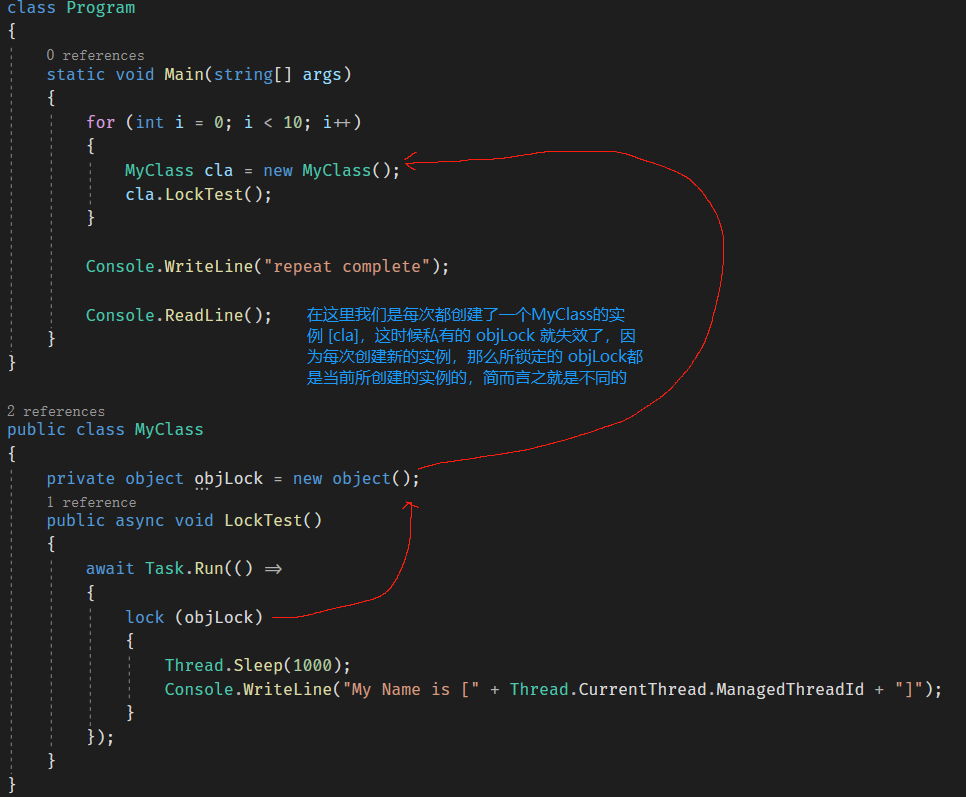
1. **lock(object)**

**-> 获取object实例的排他锁，并锁定它的实例，当线程1使用lock(object)锁定了object对象的实例后，线程2此时执行lock(object)这段代码的时候，就会造成阻塞，等待线程1退出lock(object)代码块后才能进入该区间中**

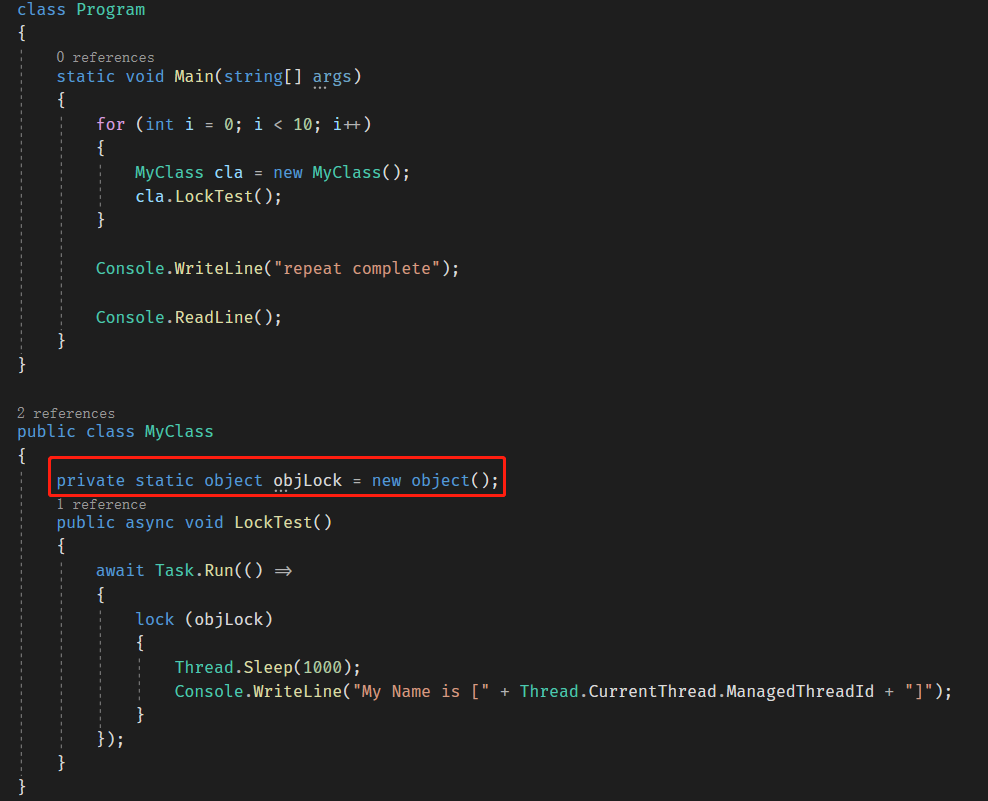
**-> lock要保证所锁定的实例不是公共的，否则容易造成不必要的死锁，比如以下情况**



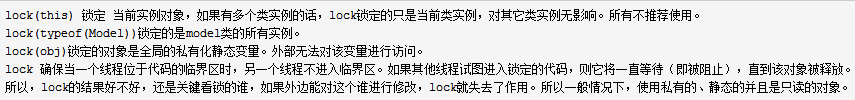
**-> lock对象要保证是私有的，否则就会发生以上的情况，但是光靠私有还存在缺陷，例如以下情况**

****

**-> lock对象最佳的做法为 一个 私有的、静态的、引用类型 的变量**



**->关于其他的用法**



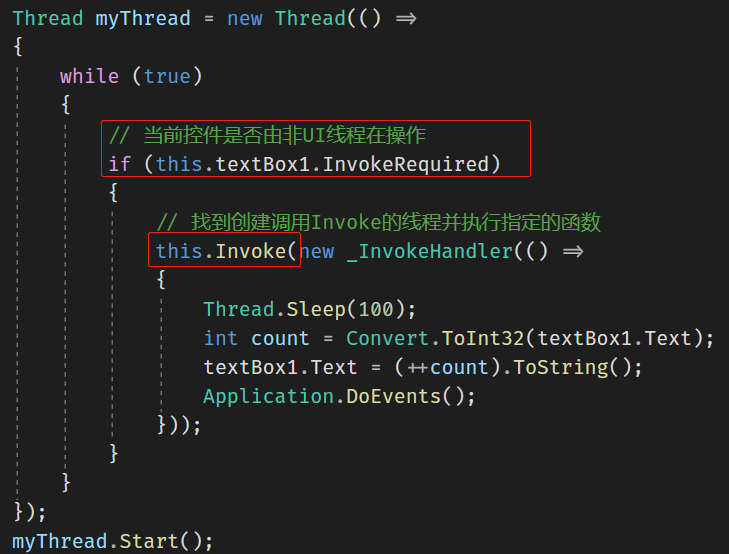
**WinForm中的跨线程访问控件问题**

**-> 在WinForm当中，当我们在一个新开的线程里面访问由UI线程所创建的控件的时候，是无法对控件内容进行操作的，为解决这一问题，以下有几种方案去实现**

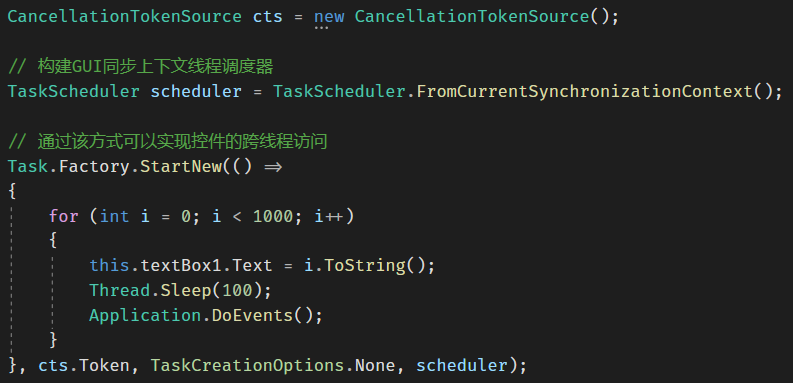
**-> 取消跨线程访问机制（不推荐）**

**微信截图_20190601195512**

**-> 通过 control/Form.Invoke() 搭配 control/Form.InvokeRequired 属性来实现**

****

**-> 指定Task的实例使用GUI同步上下文线程调度器进行调用**

****