**.NET程序集高级-多线程**

* **线程是什么？**

线程是可执行应用程序的基本执行单元。线程是应用程序真正执行代码逻辑。

* **为什么要多线程？**

应用程序总是从Main()函数开始执行的。而操作系统会将执行的任务交给一个在进程中的默认应用程序域中的默认的主线程。倘若没有在主线程中创建次线程（工作者线程），则该应用程序是单线程的。但是，一般的应用程序都是图形界面应用程序，所以应用程序要对用户的操作及时作出响应。比如，用户请求远程下载文件，而如果不单独创建工作者线程处理，则主线程将处理这个任务，但由于文件过大，单线程情况下，用户发出其他请求时，由于主线程正在执行任务，因此新请求未被及时响应（看起来像是程序卡死阻塞，其实是主线程在忙其他事）。

因此，多线程对于执行复杂操作的软件来说，是必须的。

* **多线程要克服的困难有哪些？**
* 无法控制操作系统和CRL对线程的调度

线程调度中，操作系统会在一定的时间片上根据进程优先级对并发（同步执行）的线程轮流切换执行。

* 切换不同优先级的线程：

优先级低的线程需要等待优先级高的线程完成结束后，才能被操作系统唤醒执行。

* 切换相同优先级的线程：

相同优先级的线程间，当一个线程在CPU上运行一个单位的时间片后，它将被挂起等待下一次切换执行，一直到线程完成。而排在它后面的同优先级的线程此时立即被唤醒执行。

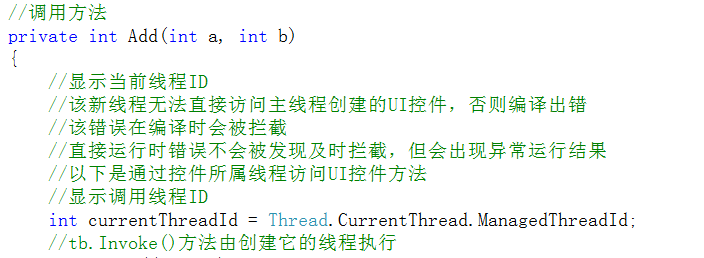
因此，假如应用程序创建一个新的线程，很有可能因为优先级低于正在执行的线程而被挂起等待，更有可能的是，排在它前面的相同优先级的多个线程也正在等待执行。因此，启动一个线程并不意味它被立即执行，我们也不知道它执行的确切时间。

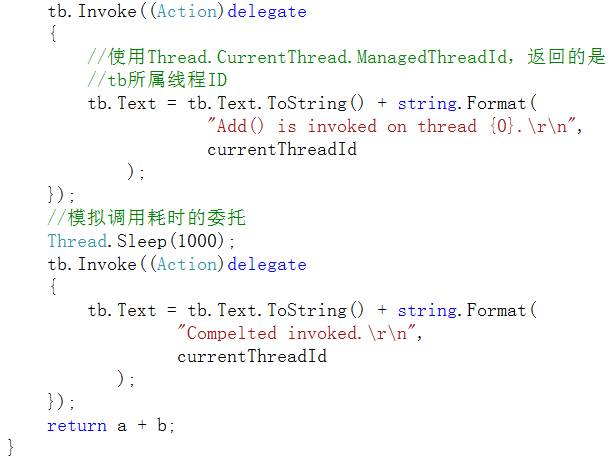
* 原子操作问题

对于单线程的应用程序域，它是线程安全的，因为其中的变量只被主线程访问。但是，对于多线程来说，变量是不安全的，线程是不稳定的。因为一个线程可能调用一个对象的方法，在未执行完该方法时被挂起，而另一个线程可能访问该对象，而其访问的数据可能是虚假无用的，从而引发无法预测的bug，而且这些bug是难以重现和调试的。

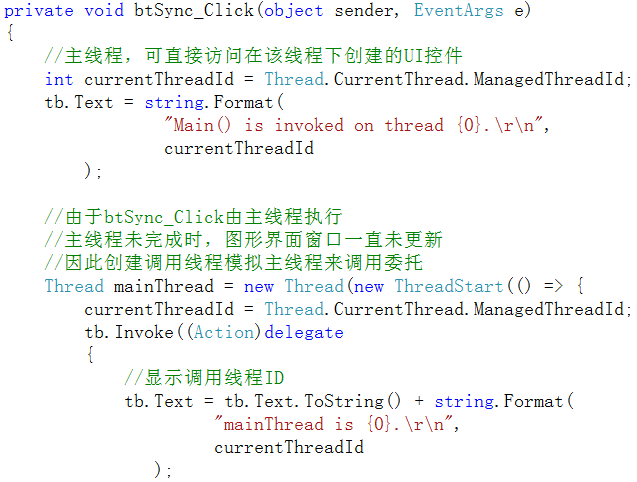
* **如何解决多线程带来的问题？**
* 针对无法控制操作系统和CRL对线程的调度的问题，没有解决的方法。实际上，随着CPU执行速度的优化加速，达到开启线程并立即执行的效果不再是一个问题，至少看上去是这样。
* 针对原子操作问题，.NET提供原子锁lock、async和await等控制线程同步。
* **多线程应用之异步委托调用**



****

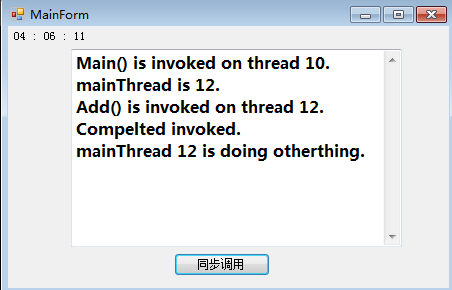
****

在自定义委托类型时，我们有时是在主线程中同步调用委托引用的方法，即该方法将由主线程负责执行，这样，在该方法调用完成之前，主线程余下的任务将被阻塞。





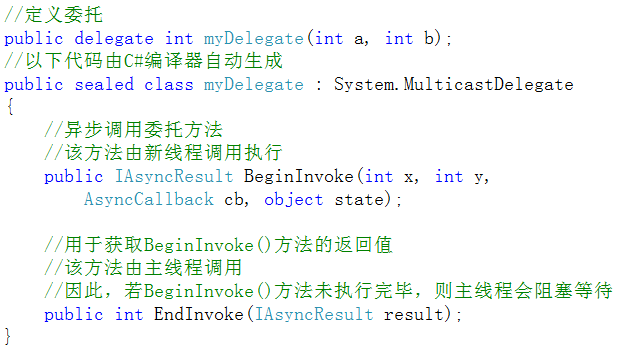
运行结果：



调用线程mainThread以同步方式调用委托的方法后被阻塞，等待委托耗时执行2秒左右，调用线程才打印出“Compelted invoked.”。

* 异步调用

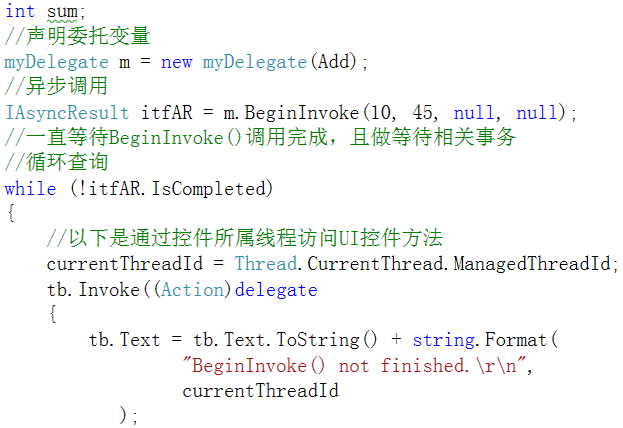
委托可以通过BeginInvoke()和EndInvoke()异步调用。

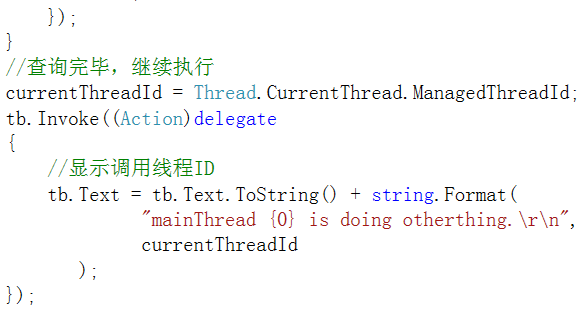


其中BeginInvoke()方法的返回值是接口IAsyncResult，它与EndInvoke()的参数必须匹配。如果执行的委托无返回值，则无需调用EndInvoke()方法，此过程主线程将不会被阻塞。但是，如主线程需要获取委托返回值而调用EndInvoke()方法时，若BeginInvoke()方法未完成时，主线程将一直被阻塞。

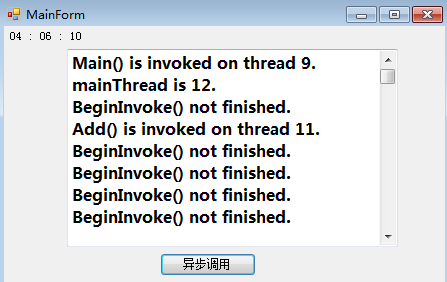
因此，以下分成两种情况来获取委托返回值：

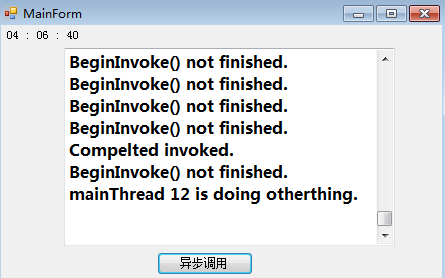
* 主线程一直查询等待返回值，并在等待期间做其他事务



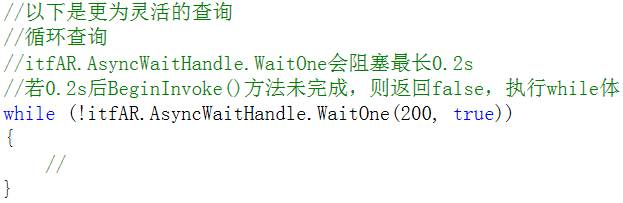


运行结果：



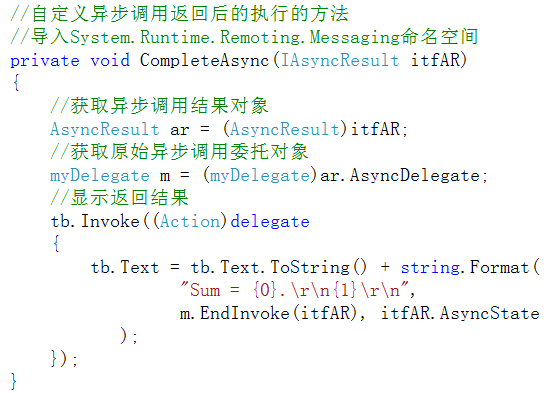


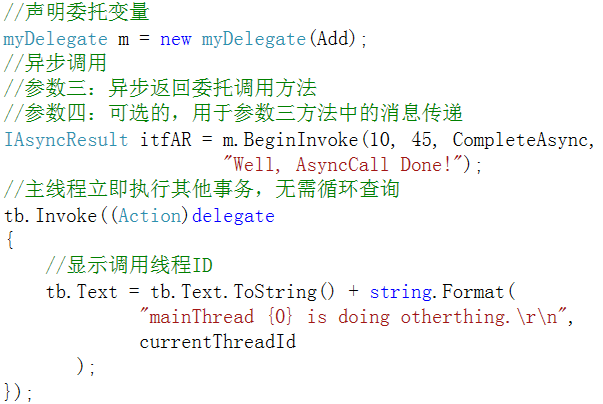
针对查询体，以下是更为灵活的查询：



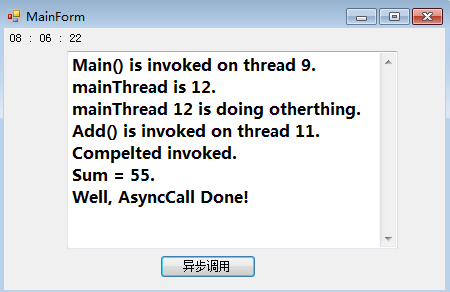
* 主线程被通知委托调用返回

方法BeginInvoke()的第三个参数接受委托AsyncCallback代理的void AsyncCallback(IAsyncResult itfAR)，这样， 只要异步调用结束后，AsyncCallback委托将自动调用引用的方法（消息通知）。





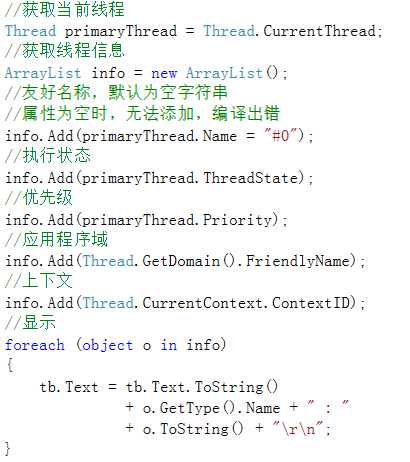
运行结果：



* **多线程应用之线程操作类Thread**

类Thread位于System.Threading下，它可以创建一个新线程，并对线程对象进行各种操作。

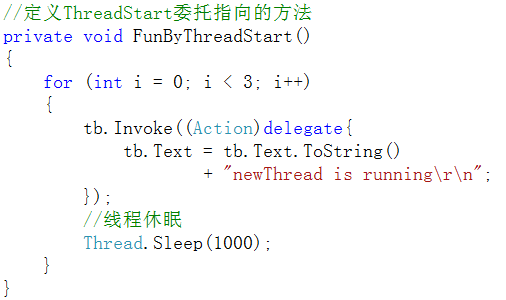
* 获取线程的信息



* 创建工作者线程

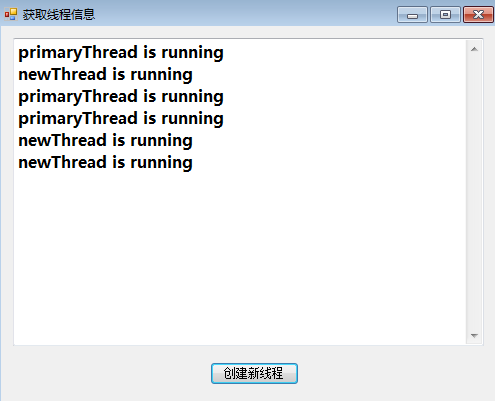
工作者线程的创建需要一个委托变量参数，该委托代理的方法必须是void返回值，但可以有任意参数，也可以无参数。

* 无参数委托ThreadStart

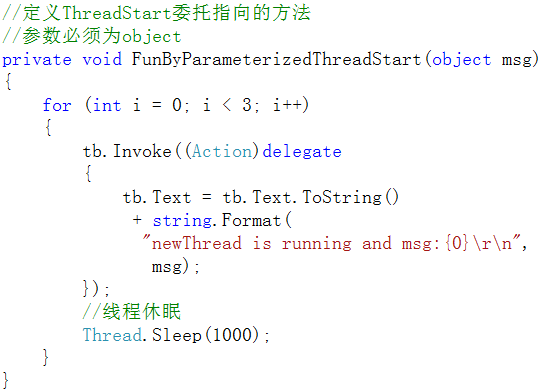


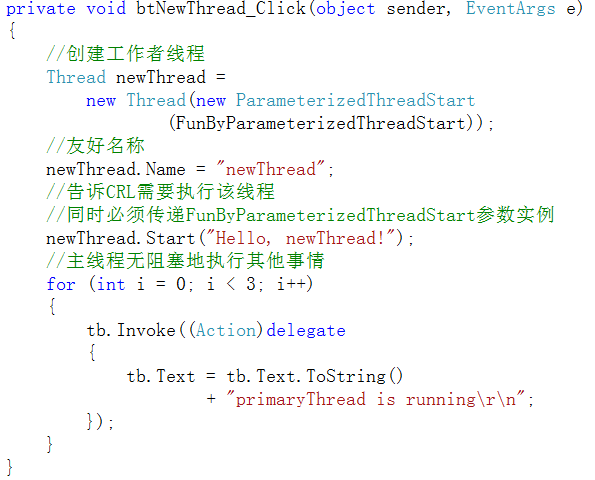


运行结果：

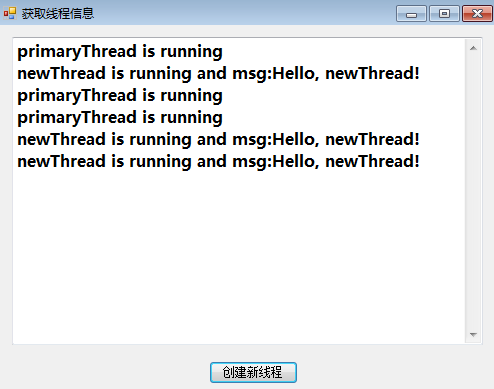


* 任意参数委托ParameterizedThreadStart





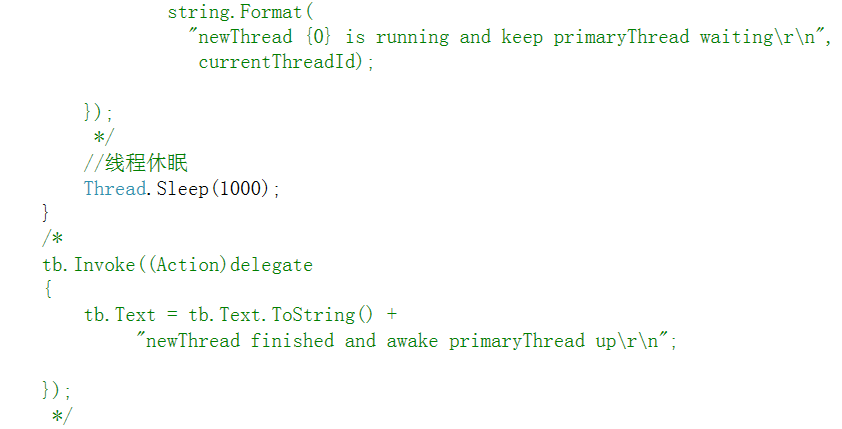
* 运行结果：

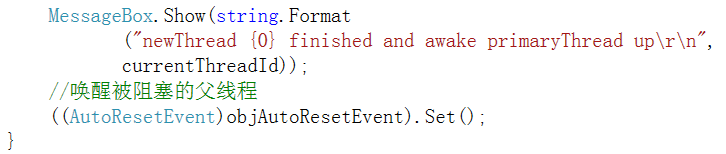


* 如何阻塞父线程？

上文创建工作者线程的两种方法都是主线程（父线程）创建该工作者线程后，两个线程都可以异步执行自己的事务。但我们仍可以通过使用类AutoResetEvent让父线程在工作者线程完成之前阻塞等待（跟异步调用委托的主线程阻塞那样）。

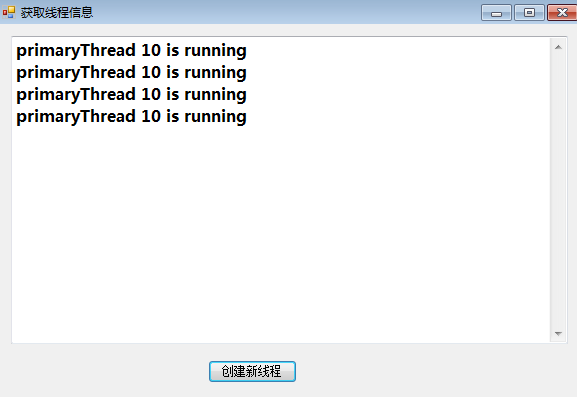
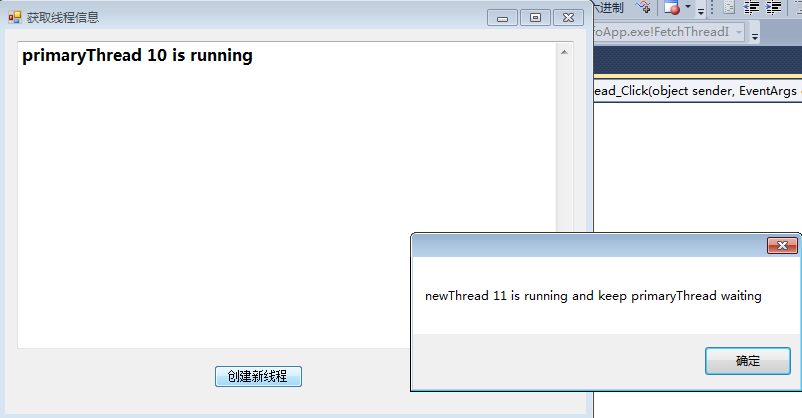








运行结果：

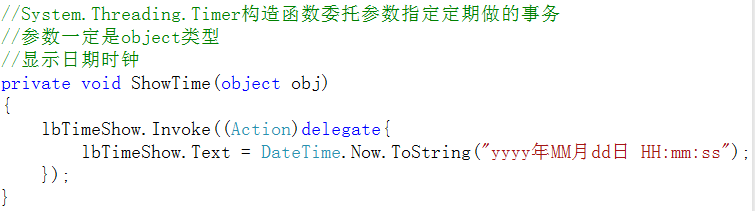


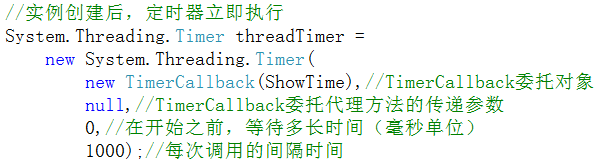
* 前台线程和后台线程

前台线程比后台线程（又称守护线程）的角色更重要，所有前台线程都结束时，不管后台线程是否结束，应用程序将被终止退出。但前台线程彼此不影响。后台线程一般否则做一些无关紧要的事情，比如每隔几分钟ping一次邮件服务器查看邮箱信息，或者显示当前天气。应用程序的第一个线程是主线程，它从Main()入口开始执行，它永远都是前台线程。其他随后创建的线程都是工作者线程。其中，由Thread创建的新线程，除非指定Thread.IsbgroundThread属性为true，否则工作者线程默认也是前台线程。而线程池里面的线程都是后台线程。其中，由并行任务库PTL的并行数据处理、并行任务和必须LINQ查询所使用的线程都是线程池里面的已存在的后台线程。

* **多线程应用之使用线程定时器System.Threading.Timer**

倘若应用程序需要每隔一段时间做一些无关紧要的事务，比如显示时间，那么，可以使用类System.Threading.Timer实例化线程定时器对象，并在构造函数里面指定委托参数TimerCallback指定的方法。



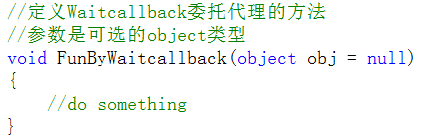


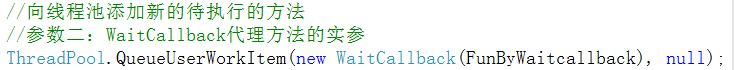
* **System.Threading.Timer和System.Timers有什么区别？**

System.Threading.Timer创建的线程都是后台线程，而System.Timers创建的线程是前台线程。

* **多线程应用之线程池**
* 什么是线程池？

线程池由CRL在加载运行应用程序时自动创建的，它里面有固定数量的线程组，每个线程可以独立执行异步委托调用的BeginInvoke()方法，或者由线程池操作类ThreadPool的静态方法QueueUserWorkItem()参数委托Waitcallback代理的方法。应用程序只有一个线程池。





* 为什么要线程池？

线程池减少了线程的创建、开始和停止的次数，提高了效率。

* 线程池里面的线程有什么缺点？
* 都是后台线程
* 优先级不能修改，默认为Normal
* 无法通过线程名等标记来获取、退出或挂起