最简单的开启多线程方法

# 一、Paraller

Paraller.Invoke（）开启多个方法

Paraller.For()多线程，开启迭代。

**代码示例:**

List tempList=new List()

Paraller.For(0,100,(I,**item**)=>{if(tempList.Count==300){item.Stop()}tempList.Add(i)})

**二、 Parallel中途退出循环和异常处理**

Break: 当然这个是通知并行计算尽快的退出循环，比如并行计算正在迭代100，那么break后程序还会迭代所有小于100的。

Stop：这个就不一样了，比如正在迭代100突然遇到stop，那它啥也不管了，直接退出

2、异常处理

　　首先任务是并行计算的，处理过程中可能会产生n多的异常，那么如何来获取到这些异常呢？普通的Exception并不能获取到异常，然而为并行诞生的AggregateExcepation就可以获取到一组异常。

声明开启多线程

命名空间：using system.Threading;

Thread t=new Thread (Fun) Fun()表示t线程开始时执行的方法

当线程们引用了一些公用的目标实例的时候，他们会共享数据.

问题就是一个线程在判断if块的时候，正好另一个线程正在执行WriteLine语句——在它将done设置为true之前。

补救措施是当读写公共字段的时候，提供一个*排他锁*；C#提供了*lock*语句来达到这个目的：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class ThreadSafe {    static bool done;  **static object locker = new object();**      static void Main() {      new Thread (Go).Start();      Go();    }      static void Go() {  **lock (locker) {**        if (!done) { Console.WriteLine ("Done"); done = true; }      }    }  } |

**临时暂停，或阻止是多线程的协同工作，同步活动的本质特征。等待一个排它锁被释放是一个线程被阻止的原因，另一个原因是线程想要暂停或*Sleep*一段时间：**

|  |  |
| --- | --- |
| 1 | Thread.Sleep (TimeSpan.FromSeconds (30));         // Block for 30 seconds |

一个线程也可以使用它的Join方法来等待另一个线程结束：

|  |  |
| --- | --- |
| 1  2  3 | Thread t = new Thread (Go);           // Assume Go is some static method  t.Start();  t.Join();                             // Wait (block) until thread t ends |

**线程是如何工作的**

   线程被一个线程协调程序管理着——一个CLR委托给操作系统的函数。线程协调程序确保将所有活动的线程被分配适当的执行时间；并且那些等待或阻止的线程——比如说在排它锁中、或在用户输入——都是不消耗CPU时间的。

**何时使用多线程**

    多线程程序一般被用来在后台执行耗时的任务。

  另一个多线程的用途是在方法中完成一个复杂的计算工作。

  一个C#程序称为多线程的可以通过2种方式：明确地创建和运行多线程，或者使用.NET framework的暗中使用了多线程的特性——

比如*BackgroundWorker*类, *线程池*，

*threading timer*，远程服务器，

或Web Services或ASP.NET程序。在后面的情况，人们别无选择，必须使用多线程；一个单线程的ASP.NET web server不是太酷，即使有这样的事情；幸运的是，应用服务器中多线程是相当普遍的；唯一值得关心的是提供适当锁机制的静态变量问题。

**创建开始使用多线程**

  线程有一个IsAlive属性，在调用Start()之后直到线程结束之前一直为true。一个线程一旦结束便不能重新开始了。

**命名线程**

  线程可以通过它的Name属性进行命名，这非产有利于调试：可以用Console.WriteLine打印出线程的名字，Microsoft Visual Studio可以将线程的名字显示在调试工具栏的位置上。线程的名字可以在被任何时间设置——但只能设置一次，重命名会引发异常。

  程序的主线程也可以被命名，下面例子里主线程通过CurrentThread命名：

**前台和后台线程**

  线程默认为前台线程，这意味着任何前台线程在运行都会保持程序存活。C#也支持后台线程，当所有前台线程结束后，它们不维持程序的存活。

  改变线程从前台到后台不会以任何方式改变它在CPU协调程序中的优先级和状态。

  线程的IsBackground属性控制它的前后台状态，如下实例：

好的方式是明确等待任何后台工作线程完成后再结束程序，可能用一个timeout（大多用*Thread.Join*）

拥有一个后台工作线程是有益的，最直接的理由是它当提到结束程序它总是可能有最后的发言权。交织以不会消亡的前台线程，保证程序的正常退出。**抛弃一个前台工作线程是尤为险恶的**，尤其对Windows Forms程序，因为程序直到主线程结束时才退出（至少对用户来说），但是它的进程仍然运行着。在Windows任务管理器它将从应用程序栏消失不见，但却可以在进程栏找到它。除非用户找到并结束它，它将继续消耗资源，并可能阻止一个新的实例的运行从开始或影响它的特性。

   对于程序失败退出的普遍原因就是存在“被忘记”的前台线程。

**线程优先级**

  线程的Priority 属性确定了线程相对于其它同一进程的活动的线程拥有多少执行时间，

**异常处理**

任何线程创建范围内try/catch/finally块，当线程开始执行便不再与其有任何关系

**补救方法是在线程处理的方法内加入他们自己的异常处理**：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static void Main() {     new Thread (Go).Start();  }    static void Go() {    try {      ...      throw null;      // 这个异常在下面会被捕捉到      ...    }    catch (Exception ex) {      记录异常日志，并且或通知另一个线程      我们发生错误      ...    } |

# 同步基础

**1.1   阻止 (Blocking)**

     当一个线程通过上面所列的方式处于等待或暂停的状态，被称为被阻止。一旦被阻止，线程立刻放弃它被分配的CPU时间，将它的ThreadState属性添加为WaitSleepJoin状态，不在安排时间直到停止阻止。停止阻止在任意四种情况下发生（关掉电脑的电源可不算！）：

* 阻止的条件已得到满足
* 操作超时（如果timeout被指定了）
* 通过Thread.Interrupt中断了
* 通过Thread.Abort放弃了

   当线程通过（不建议）Suspend 方法暂停，不认为是被阻止了。