昆明学院学报 2010,32(3):82~85 CN 53-1211/G4 ISSN 1674-5639
Journal of Kunming University

C#后台处理与多线程技术的应用

卜春芬
(昆明学院 物理科学与技术系,云南 昆明 650031)

摘要:计算机后台处理和多线程技术主要用于处理并行业务,使前台程序和后台程序同步执行,防止程序"假死".在微软新一代的.net 编程平台中,提供了 4 种实现后台处理和多线程的编程方法:BackgroundWorker 控件专门处理后台程序; Threading 类和 ThreadPool 类实现多线程编程;Timer 类实现定时处理.结果表明,多线程编程可增加程序执行的效率,缩短程序总体运行时间.

关键词:多线程;控件;线程;线程池; 计时器
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1674-5639(2010)03-0082-04

The Practical Application of C# Background Processing and Multi-Threading Technology

PU Chun-fen
(Physics Science and Technology Department, Kunming University, Yunnan Kunming 650031, China)

Abstract: The background processing and multi-threading technology provides a way for Parallel work to make the foreground program and background program processed at same time to avoid application suspending. In the last version of .net platform, there are four components provided for background processing and multi-threading technology: Background Worker provides an easy way to run time-consuming operations on a background thread; Threading and Thread pool enable multithreaded programming; Timer provides a mechanism for executing a method at specified intervals. According to the practice, the application gets better efficiency and shorter time cost with multi-threading programming.

Key words: multithreaded programming; BackgroundWorker; Threading; ThreadPool; Timer

多线程一般应用于需要执行耗费大量时间操作的情况,例如从网上下载几百兆数据或者在数据库中一次性读取或写入十多万条记录,可能要耗费几十分钟,此时,你的程序主界面可能会处于"假死"状态,打开 Windows 任务管理器可以看到程序进程占用系统 CPU 时间高达 70% ~80%,鼠标处于"忙"的状态,用户的操作基本不能响应.这时多线程可"大显身手",即新建一个后台线程,在后台线程中执行耗费大量时间的工作,而让主线程进入等待状态,随时准备响应用户的操作.作为微软最新一代的编程平台,Visual Studio.net 提供了一个简单易行的可视化控件 BackgroudWorker 来帮助程序员应付这种局面,同时还提供了 Threading,ThreadPool 和 Timer 类来实现多线程编程,比较 C++,VB 和 VC 等编程语言来说,.net 编程的难度更低,代码更简洁.

1 多线程

在 Windows 系统中每个程序运行后都会打开一个或多个进程,而每个进程又由一个或者多个线程组成.多线程的好处在于它可以提高系统资源的利用率.在多线程程序中,当一个线程必须等待时,例如 CPU 在等待外部设备或网络响应时,CPU 可执行其它操作,这样就大大缩短了程序执行的时间.然而,多线程也有不利的因素,多线程的每个线程都要占用内存,线程越多占用的内存也就越多.特别需要注意的是,多线程对共享资源的访问也会相互影响,可能会造成程序死锁,严重时会造成系统崩溃.因此,多线程之间需要协调管理.例如多个线程同时读写数据库时,首先要判断数据库的状态,在一个线程读入或写入结束后再执行下一个命令.

2 后台处理和多线的实现方式

在 C#中实现后台处理和多线程有 4 种方式:BackgroudWorker 控件;Threading(线程);ThreadPool(线程池)和 Timer(计时器)[1-3].BackgroudWorker 控件只能在后台执行,并不能实现多个线程的并行处理.真正能实现多线程的是 System.Threading 命名空间 Thread 类,Threadpool 类和 Timer 类.其中 Thread 可对创建的线程进行开始、挂起、终止和等待等一系列控制操作,但是 TreadPool 创建后是不能终止的,只能由系统自由调度执行,而 Timer 适用于需周期性调用的方法.

2.1 BackgroudWorker 控件

BackgroudWorker 控件是由微软 Visual Studio 编

收稿日期:2010-05-12
作者简介:卜春芬(1978—),女,云南鹤庆人,讲师,主要从事教育技术学及计算机软件与理论研究.

程平台提供的可视化控件,可以以最简单的方式实现多线程编程. 从控件的名称就可以看出来,这个控件常用于当需要执行比较耗费时间的任务时,响应用户事件. 这个控件通常结合 ProgressBar 控件使用,BackgroudWorker 控件处理后台数据,响应用户操作,而 ProgressBar 控件显示程序执行进度. 使用 BackgroudWorker 控件非常方便,只要从 Visual Studio 工具箱中将 BackgroundWorker 控件图标拖到窗体中即可.

如在 BackgroundWorker 的 DoWork 事件中执行耗时操作.

```
private void backgroundWorkerPing_DoWork(object sender,DoWorkEventArgs e)
{
…
this. backgroundWorkerPing. ReportProgress
(i);//传递已处理的数据数量
}
```

BackgroundWorker 控件的 ProgressChanged 事件会自动实时响应程序处理过程,可以使用 ProgressBar 进度条控件来显示处理进度.

```
private void backgroundWorkerPing _ ProgressChanged(object sender,ProgressChangedEventArgs e)
{
//设置 ProcessBar 控件显示的进度条
this. progressBarPingProcess. Value = e. ProgressPercentage + 1
}
```

可以通过检测 BackgroundWorker 控件的 CancellationPending 属性是否为 ture 来判断用户是否中断了程序的执行.

例如本人开发的一个 xDNSMng 域名管理程序,该程序的主要功能是从一个数据库表中读取域名列表(域名总数大概有 5 000 ~ 8 000 个,其中大多数的域名可能是过期的),然后通过互联网来逐一获取 IP 地址,并将获取的 IP 地址写入数据库. 经过测试,如果使用单线程处理,5 000 个域名耗时大概 50 min,在这段时间内程序基本处于“假死”状态,且执行过程不能中途取消. 如果使用 BackgroudWorker 控件,虽然不能缩短程序执行时间,但是可以随时响应用户操作和中途取消程序的执行,并保存执行前的数据.

2.2 线程(Threading)

在 C#中创建一个新线程是非常容易的,下面的语句创建了一个新的线程:

```
Thread thread = new Thread(new ThreadStart(ThreadFunc));
thread. Start();
```

该线程对象通过一个 System. Threading. ThreadStart 类的一个实例以类型安全的方法来调用 ThreadFun 函数,在函数中执行并行操作.

Thread 类还提供了很多的方法,比较常用的方法有:

Thread. Start():启动线程;

Thread. Suspend():挂起线程,或者如果线程已挂起,则不起作用;

Thread. Resume():恢复执行已挂起的线程;

Thread. Interrupt():中止处于 Wait 或 Sleep 或 Join 线程状态的线程;

Thread. Join():阻塞调用线程,直到某个线程终止时为止;

Thread. Sleep():将当前线程阻塞指定的毫秒数;

Thread. Abort():终止线程的执行.

Thread 类有很多的属性,其中比较重要有:

Thread. Priority 属性:获取或设置线程的调度优先级,可设置为 normal, lowest, highest, belownormal 和 abovenormal,线程优先级越高,获得 CPU 时间的可能性也越高.

Thread. IsBackground 属性:获取或设置该线程是否是后台线程,只要程序的主线程结束了,后台线程也就结束了.

Thread. ThreadState 属性:获取或设置当前线程的状态. Threadstate 比 Thread. IsAlive 属性能提供更多的信息. 例如:当一个线程实例刚创建时,它的值是 unstarted;当调用 start()启动之后,它的值是 running. 一旦线程被创建,它至少处于其中一个状态中,直到终止. 通过检测 ThreadState 属性的状态,可以很好的实现对线程的管理和控制,在微软的 MSDN 中对此有详细的说明.

需要特别注意的是虽然有多个线程同时存在,但是在某一个时刻,CPU 只能执行其中一个. 线程的 Start()方法只是启动了该线程,而并不保证其线程方法能立即得到执行. 它只是保证该线程对象能分配到 CPU 的时间,而实际的执行还要由操作系统根据处理器时间和线程的优先级来决定.

另一个问题是 ThreadStart 类调用的线程方法,ThreadFun 没有参数和返回值,这就限制了我们的应用. 但是在实际应用时,我们可以使用全局变量或调用类的方法来传递参数和返回值. 例如:

```
for(int i =0;i < this. dt. Rows. Count;i + + )
{
…
MyDNS md = new MyDNS(strDNS,strConn)
Thread th = new Thread(new ThreadStart(md. getIPs));
th. Start();
Thread. Sleep(200);//等待 200 ms,执行 getIPs 方法以获取 IP 地址列表;
th. Abort();//当线程执行 200 ms 后终止线程;
```

```
}
```

MyDNS 类的定义如下：

```
public MyDNS ( string dns, OleDbConnection conn);
{
this. dns = dns;//域名字符串;
this. conn = conn;//数据库连接;
public void getIPs( );
  {
…
ips = Dns. GetHostAddresses( this. dns);
  }
}
```

上面的代码构造了一个 MyDNS 类，线程调用该类的方法 md. PingandSave 来实现参数传递，并使用 for 循环建立了多个线程同时执行获取 IP 地址的操作，使用 GetHostAddresses 方法来获得域名的 IP 地址列表. 经过测试，如果该域名是可用的，这个方法执行时间大约 50 ~ 200 ms；如果该域名不能返回 IP 地址，这个方法执行时间大概 100 ~ 800 ms. 因此，可以使用多线程对此方法的执行时间做一个限制，如果线程执行时间超过 200 ms，则终止该线程，这样可以节约程序执行的时间.

所以通过使用多线程获取 IP 地址列表，限制 GetHostAddresses 方法的执行时间，极大的提高了程序的执行效率，程序执行时间由采用单线程时的 50 min 降低到大约 30 min.

2.3 线程池(ThreadPool)

ThreadPool 类提供一个由系统维护的线程池，线程池维护一个或多个线程，操作系统从请求队列中提取线程进行处理. 与多线程不同的是，线程池中的线程的启动、终止不是由设计的程序来控制的，而是由操作系统自动调用.

ThreadPool 类的使用非常简单，WaitCallback 是一个委托方法[4]，表示线程池线程要执行的回调方法，它的参数 Object state 包含了回调方法要使用的对象. 它的原型如下：

```
public delegate void WaitCallback (Object state)
```

QueueUserWorkItem 方法是将一个线程加入到线程池队列中，由操作系统自动调用执行，参数 object 将传递给 WaitCallback 所代表的方法，我们可以将多个参数封装到 Object 对象中，这个对象可以是字符串、数组、结构和类等任意类型，这样就很好的解决了多线程传递参数的问题.

一个简单完整的例子如下：

```
using System;
using System. Threading;
public class ThreadBase
{
public static void Main( )
  {
System. Threading. WaitCallback waitCallback = new WaitCallback( th);
ThreadPool. QueueUserWorkItem ( waitCallback,"第 1 个线程");
ThreadPool. QueueUserWorkItem ( waitCallback,"第 2 个线程");
ThreadPool. QueueUserWorkItem ( waitCallback,"第 3 个线程");
ThreadPool. QueueUserWorkItem ( waitCallback,"第 4 个线程");
Console. ReadLine( );
  }
public static void th( object state)
  {
Console. WriteLine ("线程开始…… {0}", (string) state);
Thread. Sleep( 10000);
Console. WriteLine ("线程结束…… {0}", (string) state);
  }
}
```

输出如下：

```
线程开始……第 1 个线程
线程开始……第 2 个线程
线程开始……第 3 个线程
线程开始……第 4 个线程
线程结束……第 1 个线程
线程结束……第 2 个线程
线程结束……第 3 个线程
线程结束……第 4 个线程
```

ThreadPool 类还提供多个方法，对线程池进行简单查询：

GetAvailableThreads 获得可用线程数，也就是 GetMaxThreads 方法返回的最大线程数和当前活动线程数相减的值；

GetMaxThreads 获得可同时处于活动状态的最大线程数；

SetMaxThreads 设置可同时处于活动状态的最大线程数，所有大于此数目的请求将保持排队状态，直到线程池线程变为可用；

2.4 计时器(Timer)

有一种情况是线程平常都处于休眠状态，只是周期性地被唤醒，这种情况可以使用 Timer 类来处理. 当定时器启动后，系统将自动执行 timercallback 代理对象指定的方法. Timer 也是由操作系统来管理线程. Timer 对象的构造为：

```
public timer(
timercallback callback,//指定了 TimerCallback 代理对象所需调用的方法
```

```
object state,//传递参数给要调用的方法
int duetime,//延迟时间——计时开始的时刻距现在的时间
int period//定时器的时间间隔
);
```

调用 Timer. Change( ) 方法可以修改定时器的设置,下面代码将时间间隔设置为 2 s,停止计时 10 s 后生效:

```
timer. Change(10 000,2 000);
```

下面这段代码简单演示了 Timer 类的用法:

```
timer t = new timer (new timercallback (timercall),3,1000,1000);
public static void timercall(object ob)
{
…
}
```

System. Threading. Timer 是一个“轻量化”的计时器,. net 还提供了 System. Windows. Forms. Timer 类和 System. Timers. Timer,可应用于 windows 窗体界面,并具有其它更多的功能.

3　小结

C#后台处理技术可以有效防止系统“假死”,并能够实时响应用户操作,结合其它控件可以实时显示程序的执行进度,并可在任意时间终止进程执行. 多线程技术可并行执行多个线程,缩短程序等待网络或外部设备的响应时间,提高了程序执行效率,缩短了耗时程序的总体运行时间.

[参考文献]

[1]内格尔,鲁滨逊. C#高级编程[M]. 李铭,译. 第 6 版. 北京:清华大学出版社,2008:478 - 517.

[2]沃森,内格尔. C#入门经典[M]. 齐立波,译. 第 3 版. 北京:清华大学出版社,2006:187 - 188.

[3]阿切尔. C#技术内幕[M]. 侯晓霞,柴洪辉,译. 北京:清华大学出版社,2002:277 - 299.

[4]陈钟,刘强,张高. C#编程语言程序设计与开发[M]. 北京:清华大学出版社,2003:121 - 129.

(上接第 68 页)

4　结束语

在数字水印算法中同时使用纠错编码和图像置乱变换,提高了 LDPC 码的纠错效率、数字水印的稳健性;图像置乱将密码学意义上的变换技术引入了数字水印,使得水印的嵌入和提取都需要密钥才能进行,很大程度上提高了数字水印安全性,这增强了数字水印的实用性. 但是由于图像置乱变换将成块的错误分散了,使得该算法对裁剪攻击这种过于密集攻击的抵抗力有所下降,这一点还需进一步改进.

[参考文献]

[1]陆正福,王涛. 符合 GSAKMP 的覆盖多播水印协议的设计与分析[J]. 计算机工程与设计, 2008,29(1):225 - 258.

[2]陆正福,叶锐,王国栋. 基于移动代理的多播水印协议[J]. 云南大学学报:自然科学版,2004,26(4):306 - 311.

[3]陆正福,王涛. 一类基于主从代理的应用层多播水印协议的设计与分析[J]. 云南大学学报:自然科学版,2007,29(3):234 - 240.

[4]叶锐,陆正福,王国栋. 多播水印协议中指纹编码方案的研究[J]. 昆明师范高等专科学校学报,2003,25(4):35 - 38.

[5]王育民,李晖,梁传甲. 信息论与编码理论[M]. 北京:高等教育出版社,2005.

[6]文红,符初生,周亮. LDPC 码原理与应用[M]. 四川:电子科技大学出版社,2006.

[7]周统和,乔秦宝,谢亮. 数字图像盲水印算法与 LDPC 码的联合方案[J]. 电子技术应用,2006(4):40 - 42.

[8]霍智勇,朱秀昌. 基于 LDPC 码的数字图像水印技术研究[J]. 中国图象图形学报,2007,12(11):2018 - 2025.

[9]王昕,袁东风. 借助 LDPC 码提高数字水印鲁棒性[J]. 计算机工程与科学,2006,28(8):140 - 142.

[10]刘家胜,黄贤武,朱灿焰,等. 基于 m 序列变换和混沌映射的图像加密算法[J]. 电子与信息学报,2007,29(6):1476 - 1479.

[11]李名威,冯勇,李林静. 一种新的基于像素置乱的图像加密算法[J]. 黑龙江科技信息,2007(8):68 - 69.

[12]李源. 基于图像置乱和纠错码的数字水印预处理研究[D]. 昆明:云南大学,2007.

[13]DAVID J C. Good error correcting codes based on very sparse matrices[J]. IEEE Transactions on Information Theory, 1999, 45(2): 399 - 431.