Wpf配置软件开发

开发工具

1. 安装VS2010
2. 安装Apex 组件
3. 增加引用到项目中
4. 增加WPFTOOLkit
5. 增加WPFtoolkit extend

知识

1. MVVM模式
2. C#知识
3. 网络UDP相关知识

[**C#多线程学习(四) 多线程的自动管理(线程池)**](http://www.cnblogs.com/xugang/archive/2008/03/23/1118584.html)

在多线程的程序中，经常会出现两种情况：

**一种情况：**   应用程序中，线程把大部分的时间花费在等待状态，等待某个事件发生，然后才能给予响应  
                  这一般使用ThreadPool（线程池）来解决；

**另一种情况：**线程平时都处于休眠状态，只是周期性地被唤醒  
                  这一般使用Timer（定时器）来解决；

ThreadPool类提供一个由系统维护的线程池（可以看作一个线程的容器），该容器需要 Windows 2000 以上系统支持，因为其中某些方法调用了只有高版本的Windows才有的API函数。

将线程安放在线程池里，需使用ThreadPool.QueueUserWorkItem()方法，该方法的原型如下：

//将一个线程放进线程池，该线程的Start()方法将调用WaitCallback代理对象代表的函数  
public static bool QueueUserWorkItem(WaitCallback);

//重载的方法如下，参数object将传递给WaitCallback所代表的方法  
public static bool QueueUserWorkItem(WaitCallback, object);  
  
**注意：**  
ThreadPool类是一个静态类，你不能也不必要生成它的对象。而且一旦使用该方法在线程池中添加了一个项目，那么该项目将是无法取消的。

在这里你无需自己建立线程，只需把你要做的工作写成函数，然后作为参数传递给ThreadPool.QueueUserWorkItem()方法就行了，传递的方法就是依靠WaitCallback代理对象，而线程的建立、管理、运行等工作都是由系统自动完成的，你无须考虑那些复杂的细节问题。  
  
**ThreadPool 的用法：**首先程序创建了一个ManualResetEvent对象，该对象就像一个信号灯，可以利用它的信号来通知其它线程。  
本例中，当线程池中所有线程工作都完成以后，ManualResetEvent对象将被设置为有信号，从而通知主线程继续运行。

**ManualResetEvent对象有几个重要的方法：**  
初始化该对象时，用户可以指定其默认的状态（有信号/无信号）；  
在初始化以后，该对象将保持原来的状态不变，直到它的Reset()或者Set()方法被调用：  
Reset()方法：将其设置为无信号状态；  
Set()方法：将其设置为有信号状态。  
WaitOne()方法：使当前线程挂起，直到ManualResetEvent对象处于有信号状态，此时该线程将被激活。然后，程序将向线程池中添加工作项，这些以函数形式提供的工作项被系统用来初始化自动建立的线程。当所有的线程都运行完了以后，ManualResetEvent.Set()方法被调用，因为调用了ManualResetEvent.WaitOne()方法而处在等待状态的主线程将接收到这个信号，于是它接着往下执行，完成后边的工作。  
  
ThreadPool 的用法示例：

using System;  
using System.Collections;  
using System.Threading;  
  
namespace ThreadExample  
{  
    //这是用来保存信息的数据结构，将作为参数被传递  
    public class SomeState  
    {  
　　    public int Cookie;  
　　    public SomeState(int iCookie)  
　　    {  
        Cookie = iCookie;  
　　    }  
    }  
  
    public class Alpha  
    {  
　　public Hashtable HashCount;  
　　public ManualResetEvent eventX;  
　　public static int iCount = 0;  
　　public static int iMaxCount = 0;  
　　  
        public Alpha(int MaxCount)   
　　{  
　        HashCount = new Hashtable(MaxCount);  
　        iMaxCount = MaxCount;  
　　}  
  
　　//线程池里的线程将调用Beta()方法  
　　public void Beta(Object state)  
　　{  
　　    //输出当前线程的hash编码值和Cookie的值  
　        Console.WriteLine(" {0} {1} :", Thread.CurrentThread.GetHashCode(),((SomeState)state).Cookie);  
      Console.WriteLine("HashCount.Count=={0}, Thread.CurrentThread.GetHashCode()=={1}", HashCount.Count, Thread.CurrentThread.GetHashCode());  
      lock (HashCount)   
      {  
　　　　    //如果当前的Hash表中没有当前线程的Hash值，则添加之  
　　　　    if (!HashCount.ContainsKey(Thread.CurrentThread.GetHashCode()))  
　　      　    HashCount.Add (Thread.CurrentThread.GetHashCode(), 0);  
　  　　    HashCount[Thread.CurrentThread.GetHashCode()] =   
            ((int)HashCount[Thread.CurrentThread.GetHashCode()])+1;  
　    　}  
          int iX = 2000;  
          Thread.Sleep(iX);  
          //Interlocked.Increment()操作是一个原子操作，具体请看下面说明  
          Interlocked.Increment(ref iCount);  
  
          if (iCount == iMaxCount)  
          {  
　    　    Console.WriteLine();  
　　　　    Console.WriteLine("Setting eventX ");  
　　　　    eventX.Set();  
   　    }  
　　  }  
  }  
  
        public class SimplePool  
        {  
            public static int Main(string[] args)  
            {  
                Console.WriteLine("Thread Pool Sample:");  
                bool W2K = false;  
                int MaxCount = 10;//允许线程池中运行最多10个线程  
                //新建ManualResetEvent对象并且初始化为无信号状态  
                ManualResetEvent eventX = new ManualResetEvent(false);  
                Console.WriteLine("Queuing {0} items to Thread Pool", MaxCount);  
                Alpha oAlpha = new Alpha(MaxCount);   
                //创建工作项  
                //注意初始化oAlpha对象的eventX属性  
                oAlpha.eventX = eventX;  
                Console.WriteLine("Queue to Thread Pool 0");  
                try  
                {  
                    //将工作项装入线程池   
                    //这里要用到Windows 2000以上版本才有的API，所以可能出现NotSupportException异常  
                    ThreadPool.QueueUserWorkItem(new WaitCallback(oAlpha.Beta), new SomeState(0));  
                    W2K = true;  
                }  
                catch (NotSupportedException)  
                {  
                    Console.WriteLine("These API's may fail when called on a non-Windows 2000 system.");  
                    W2K = false;  
                }  
                if (W2K)//如果当前系统支持ThreadPool的方法.  
                {  
                    for (int iItem=1;iItem < MaxCount;iItem++)  
                    {  
                        //插入队列元素  
                        Console.WriteLine("Queue to Thread Pool {0}", iItem);  
                        ThreadPool.QueueUserWorkItem(new WaitCallback(oAlpha.Beta), new SomeState(iItem));  
                    }  
                    Console.WriteLine("Waiting for Thread Pool to drain");  
                    //等待事件的完成，即线程调用ManualResetEvent.Set()方法  
                    eventX.WaitOne(Timeout.Infinite,true);  
                    //WaitOne()方法使调用它的线程等待直到eventX.Set()方法被调用  
                    Console.WriteLine("Thread Pool has been drained (Event fired)");  
                    Console.WriteLine();  
                    Console.WriteLine("Load across threads");  
                    foreach(object o in oAlpha.HashCount.Keys)  
                        Console.WriteLine("{0} {1}", o, oAlpha.HashCount[o]);  
                }  
                Console.ReadLine();  
                return 0;  
            }  
        }  
    }  
  
}

**程序中应该引起注意的地方：**SomeState类是一个保存信息的数据结构，它在程序中作为参数被传递给每一个线程，因为你需要把一些有用的信息封装起来提供给线程，而这种方式是非常有效的。  
程序出现的InterLocked类也是专为多线程程序而存在的，它提供了一些有用的原子操作。

原子操作：就是在多线程程序中，如果这个线程调用这个操作修改一个变量，那么其他线程就不能修改这个变量了，这跟lock关键字在本质上是一样的。