在.NET work 2.0中提供了SerialPort类，该类主要实现串口数据通信等。本文章将本人在学习过程中从网络上搜集到的相关信息写出来供大家参考。

下面主要介绍该类的主要属性（表1）和方法（表.2）。

如果需要了解更多的信息请登录<http://msdn.microsoft.com/zh-cn/library/system.io.ports.serialport(VS.80).aspx>查看。

相关文章

《使用System.IO.Ports读取COM口数据》

<http://www.devasp.net/net/articles/display/727.html>

使用SerialPort类的方法：

方法一：

首先要添加

using System.IO;  
using System.IO.Ports;

1...在类的内部定义SerialPort com;

2...打开串口

            com = new SerialPort();  
            com.BaudRate = 115200;  
            com.PortName = "COM1";  
            com.DataBits = 8;  
            com.Open();//打开串口

3...发送数据

            Byte[] TxData ={1,2,3,4,5,6,7,8 };  
            com.Write(TxData, 0, 8);

4...接收数据

     4.1使用事件接收

     this.com.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(this.OnDataReceived);

private void OnDataReceived(object sender, SerialDataReceivedEventArgs e)

    4.2使用线程接收

     接收数据启动一个线程，使其接收。

在类的内部定义

        Thread \_readThread;  
        bool \_keepReading;

打开串口后启动线程

            \_keepReading = true;  
            \_readThread = new Thread(ReadPort);  
            \_readThread.Start();

线程函数

view plaincopy to clipboardprint?  
private void ReadPort()    
{    
    while (\_keepReading)    
    {    
        if (com.IsOpen)    
        {    
            byte[] readBuffer = new byte[com.ReadBufferSize + 1];    
            try  
            {    
                // If there are bytes available on the serial port,    
                // Read returns up to "count" bytes, but will not block (wait)    
                // for the remaining bytes. If there are no bytes available    
                // on the serial port, Read will block until at least one byte    
                // is available on the port, up until the ReadTimeout milliseconds    
                // have elapsed, at which time a TimeoutException will be thrown.    
                int count = com.Read(readBuffer, 0, com.ReadBufferSize);    
                String SerialIn = System.Text.Encoding.ASCII.GetString(readBuffer, 0, count);    
                if (count != 0)    
                    //byteToHexStr(readBuffer);    
                    Thread(byteToHexStr(readBuffer,count));    
            }    
            catch (TimeoutException) { }    
        }    
        else  
        {    
            TimeSpan waitTime = new TimeSpan(0, 0, 0, 0, 50);    
            Thread.Sleep(waitTime);    
        }    
    }    
}  
        private void ReadPort()  
        {  
            while (\_keepReading)  
            {  
                if (com.IsOpen)  
                {  
                    byte[] readBuffer = new byte[com.ReadBufferSize + 1];  
                    try  
                    {  
                        // If there are bytes available on the serial port,  
                        // Read returns up to "count" bytes, but will not block (wait)  
                        // for the remaining bytes. If there are no bytes available  
                        // on the serial port, Read will block until at least one byte  
                        // is available on the port, up until the ReadTimeout milliseconds  
                        // have elapsed, at which time a TimeoutException will be thrown.  
                        int count = com.Read(readBuffer, 0, com.ReadBufferSize);  
                        String SerialIn = System.Text.Encoding.ASCII.GetString(readBuffer, 0, count);  
                        if (count != 0)  
                            //byteToHexStr(readBuffer);  
                            Thread(byteToHexStr(readBuffer,count));  
                    }  
                    catch (TimeoutException) { }  
                }  
                else  
                {  
                    TimeSpan waitTime = new TimeSpan(0, 0, 0, 0, 50);  
                    Thread.Sleep(waitTime);  
                }  
            }  
        }

方法二：使用C#自带的SerialPor控件。

1...在“工具箱”的“组件”中选择SerialPor控件添加。

2...设置串口并打开

serialPort1.PortName = "COM1";

serialPort1.BaudRate = 9600;

serialPort1.Open();

3...写入数据可以使用Write或者下面的函数

serialPort1.WriteLine(str);

4...添加数据接收的事件

private void serialPort1\_DataReceived(object sender, SerialDataReceivedEventArgs e)

使用中的一些常见问题

C#中SerialPort类中DataReceived事件GUI实时处理方法（来自[wanglei\_wan@yahoo.com.cn](mailto:wanglei_wan@yahoo.com.cn) 的看法）  
MSDN：从 SerialPort 对象接收数据时，将在辅助线程上引发 DataReceived 事件。由于此事件在辅助线程而非主线程上引发，因此尝试修改主线程中的一些元素（如 UI 元素）时会引发线程异常。如果有必要修改主 Form 或 Control 中的元素，必须使用 Invoke 回发更改请求，这将在正确的线程上执行.进而要想将辅助线程中所读到的数据显示到主线程的Form控件上时，只有通过Invoke方法来实现  
下面是代码实例：  
private void serialPort1\_DataReceived(object sender, SerialDataReceivedEventArgs e)  
{  
   int SDateTemp = this.serialPort1.ReadByte();  
   //读取串口中一个字节的数据  
   this.tB\_ReceiveDate.Invoke(    
//在拥有此控件的基础窗口句柄的线程上执行委托Invoke(Delegate)  
//即在textBox\_ReceiveDate控件的父窗口form中执行委托.  
new MethodInvoker(             
/\*表示一个委托，该委托可执行托管代码中声明为 void 且不接受任何参数的任何方法。 在对控件的 Invoke 方法进行调用时或需要一个简单委托又不想自己定义时可以使用该委托。\*/  
delegate{                  
    /\*匿名方法,C#2.0的新功能，这是一种允许程序员将一段完整代码区块当成参数传递的程序代码编写技术，通过此种方法可 以直接使用委托来设计事件响应程序以下就是你要在主线程上实现的功能但是有一点要注意，这里不适宜处理过多的方法，因为C#消息机制是消息流水线响应机制，如果这里在主线程上处理语句的时间过长会导致主UI线程阻塞，停止响应或响应不顺畅,这时你的主form界面会延迟或卡死      \*/                    
this.tB\_ReceiveDate.AppendText(SDateTemp.ToString());//输出到主窗口文本控件  
this.tB\_ReceiveDate.Text += " ";}  
    )  
    );  
}

如何知道当前电脑有哪个串口

在窗体上添加一个comboBox控件。

然后使用comboBox1.Items.AddRange(System.IO.Ports.SerialPort.GetPortNames()); 或者

string[] portList = System.IO.Ports.SerialPort.GetPortNames();  
            for (int i = 0; i < portList.Length; ++i)  
            {  
                string name = portList[i];  
                comboBox1.Items.Add(name);  
            }

具体请参考<http://msdn.microsoft.com/zh-tw/library/system.io.ports.serialport.getportnames.aspx>

[**C# Invoke、begininvoke的用法**](https://www.cnblogs.com/ybqjymy/p/14281286.html)

**invoke和begininvoke 区别**

一直对invoke和begininvoke的使用和概念比较混乱，这两天看了些资料，对这两个的用法和原理有了些新的认识和理解。

 首先说下，invoke和begininvoke的使用有两种情况：

  1. control中的invoke、begininvoke。

  2. delegrate中的invoke、begininvoke。

  这两种情况是不同的，我们这里要讲的是第1种。下面我们在来说下.NET中对invoke和begininvoke的官方定义。

  control.invoke(参数delegate)方法:在**拥有**此控件的基础窗口句柄的线程上**执行**指定的委托。

  control.begininvoke(参数delegate)方法:在**创建**控件的基础句柄所在线程上**异步执行**指定委托。

  根据这两个概念我们大致理解invoke表是**同步**、begininvoke表示**异步**。

**如果你的后台线程在更新一个UI控件的状态后不需要等待，而是要继续往下处理，那么你就应该使用BeginInvoke来进行异步处理。**

**如果你的后台线程需要操作UI控件，并且需要等到该操作执行完毕才能继续执行，那么你就应该使用Invoke。**

我们来做一个测试。

**invoke** 例子：

[复制代码](javascript:void(0);)

1 private void button1\_Click(object sender, EventArgs e)

2 {

3 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+"AAA");

4 invokeThread = new Thread(new ThreadStart(StartMethod));

5 invokeThread.Start();

6 string a = string.Empty;

7 for (int i = 0; i < 3; i++) //调整循环次数，看的会更清楚

8 {

9 Thread.Sleep(1000);

10 a = a + "B";

11 }

12 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+a);

13 }

14

15 private void StartMethod()

16 {

17 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+"CCC");

18 button1.Invoke(new invokeDelegate(invokeMethod));

19 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+"DDD");

20 }

21

22 private void invokeMethod()

23 {

24 //Thread.Sleep(3000);

25 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString() + "EEE");

26 }

[复制代码](javascript:void(0);)

结论：我们运行后，看下程序的运行顺序，1AAA->3CCC和1BBB->1EEE ->3DDD 。

解释：主线程运行1AAA，然后1BBB和子线程3CCC同时执行，然后通过invoke来将invokemethod方法提交给主线程，然后子线 程等待主线程执行，直到主线程将invokemethod方法执行完成（期间必须等待主线程的任务执行完成，才会去执行invoke提交的任务），最后执 行子线程3DDD。

**begininvoke** 例子：

[复制代码](javascript:void(0);)

1 private void button1\_Click(object sender, EventArgs e)

2 {

3 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+"AAA");

4 invokeThread = new Thread(new ThreadStart(StartMethod));

5 invokeThread.Start();

6 string a = string.Empty;

7 for (int i = 0; i < 3; i++) //调整循环次数，看的会更清楚

8 {

9 Thread.Sleep(1000);

10 a = a + "B";

11 }

12 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+a);

13 }

14

15 private void StartMethod()

16 {

17 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+"CCC");

18 button1.BeginInvoke(new invokeDelegate(invokeMethod));

19 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString()+"DDD");

20 }

21

22 private void beginInvokeMethod()

23 {

24 //Thread.Sleep(3000);

25 MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString() + "EEEEEEEEEEEE");

26 }

[复制代码](javascript:void(0);)

结论： 我们运行后看看执行的结果：1AAA->1BBB和3CCC->1EEE和3DDD。

解释： 主线程运行1AAA，然后1BBB和子线程3CCC同时执行，然后通过begininvoke来将invokemethod方法提交给主线程，然后主线程执行1EEE（主线程自己的任务执行完成）， 同时子线程继续执行3DDD。

通过这个两段代码的测试比较，我们会发现其实invoke和begininvoke所提交的委托方法都是在主线程中执行的，其实根据我invoke 和begininvoke的定义我们要在子线程中来看这个问题，在invoke例子中我们会发现invoke所提交的委托方法执行完成后，才能继续执行 DDD；在begininvoke例子中我们会发现begininvoke所提交的委托方法后，子线程讲继续执行DDD，不需要等待委托方法的完成。 那么现在我们在回想下invoke（同步）和begininvoke（异步）的概念，其实它们所说的意思是相对于子线程而言的，其实对于控件的调用总是由 主线程来执行的。我们很多人搞不清这个同步和异步，主要还是因为我们把参照物选错了。其实有时候光看概念是很容易理解错误的。

**解决从不是创建控件的线程访问它**

**在多线程编程中，我们经常要在工作线程中去更新界面显示，而在多线程中直接调用界面控件的方法是错误的做法，Invoke 和 BeginInvoke 就是为了解决这个问题而出现的，使你在多线程中安全的更新界面显示。  
  
正确的做法是将工作线程中涉及更新界面的代码封装为一个方法，通过 Invoke 或者 BeginInvoke 去调用，两者的区别就是一个导致工作线程等待，而另外一个则不会。  
  
而所谓的“一面响应操作，一面添加节点”永远只能是相对的，使 UI 线程的负担不至于太大而已，因为界面的正确更新始终要通过 UI 线程去做，我们要做的事情是在工作线程中包揽大部分的运算，而将对纯粹的界面更新放到 UI 线程中去做，这样也就达到了减轻 UI 线程负担的目的了。  
  
举个简单例子说明下使用方法,比如你在启动一个线程,在线程的方法中想更新窗体中的一个TextBox..**

[复制代码](javascript:void(0);)

1 using System.Threading;

2

3 //启动一个线程

4 Thread thread=new Thread(new ThreadStart(DoWork));

5 thread.Start();

6

7 //线程方法

8 private void DoWork()

9 {

10 　　this.TextBox1.Text="我是一个文本框";

11 }

[复制代码](javascript:void(0);)

如果你像上面操作,在VS2005或2008里是会有异常的...   
  
正确的做法是用Invoke\BeginInvoke

[复制代码](javascript:void(0);)

1 using System.Threading;

2 namespace test

3 {

4 　　public partial class Form1 : Form

5 　　{

6 　　　　public delegate void MyInvoke(string str1,string str2);

7 　　　　public Form1()

8 　　　　{

9 　　　　　　InitializeComponent();

10

11

12 　　　　}

13 　　　　public void DoWork()

14 　　　　{

15 　　　　　　MyInvoke mi = new MyInvoke(UpdateForm);

16 　　　　　　this.BeginInvoke(mi, new Object[] {"我是文本框","haha"});

17 　　　　}

18 　　　　public void UpdateForm(string param1,string parm2)

19 　　　　{

20 　　　　　　this.textBox1.Text = param1+parm2;

21 　　　　}

22 　　　　private void button1\_Click(object sender, EventArgs e)

23 　　　　{

24 　　　　　　Thread thread = new Thread(new ThreadStart(DoWork));

25 　　　　　　thread.Start();

26 　　　　}

27 　　}

28 }

[复制代码](javascript:void(0);)

注意代理的使用！

**后面再次补充**

　　在 WinForm开发过程中经常会用到线程，有时候还往往需要在线程中访问线程外的控件，比如：设置textbox的Text属性等等。如果直接设置程序必 定会报出：从不是创建控件的线程访问它，这个异常。通常我们可以采用两种方法来解决。一是通过设置control的属性。二是通过delegate,而通 过delegate也有两种方式，一种是常用的方式，另一种就是匿名方式。下面分别加以说明.

　　首先，通过设置control的一个属性值为false.我们可以在Form\_Load方法中添加：Control.CheckForIllegalCrossThreadCalls=false;来解决。设置为false表示不对错误线程的调用进行捕获。这样在线程中对textbox的Text属性进行设置时就不会再报错了。  
　　其次，通过delegate的方法来解决。  
普通的委托方法例如：

[复制代码](javascript:void(0);)

1 delegate void SafeSetText(string strMsg);

2 private void SetText(string strMsg)

3 {

4 if(textbox1.InvokeRequired)

5 {

6 SafeSetText objSet=new SafeSetText(SetText);

7 textbox1.Invoke(objSet,new object[]{strMsg});

8 }

9 else

10 {

11 　　textbox1.Text=strMsg;

12 }

13 }

[复制代码](javascript:void(0);)

在线程内需要设置textbox的值时调用SetText方法既可。我们还可以采用另一种委托的方式来实现，那就是匿名代理，例如：

[复制代码](javascript:void(0);)

1 delegate void SafeSetText(string strMsg);

2 private void SetText2(string strMsg)

3 {

4 　　SafeSetText objSet = delegate(string str)

5 {

6 textBox1.Text = str;

7 }

8 textBox1.Invoke(objSet,new object[]{strMsg});

9 }

[复制代码](javascript:void(0);)

这样同样可以实现。  
个人觉得还是采用代理好些。

在C# 3.0及以后的版本中有了Lamda表达式，像上面这种匿名委托有了更简洁的写法。.NET Framework 3.5及以后版本更能用Action封装方法。例如以下写法可以看上去非常简洁：

[复制代码](javascript:void(0);)

1 void ButtonOnClick(object sender,EventArgs e)

2 {

3 this.Invoke(new Action(()=>

4 {

5 button.Text="关闭";

6 }));

7 }

[复制代码](javascript:void(0);)

最新：

1 Invoke(() =>

2 {

3 button.Text="关闭";

4 });