文章编号：2096-1472(2017)-03-21-03

多线程在WinForm窗体开发中的应用研究

周 岚
（江苏联合职业技术学院徐州财经分院，江苏 徐州 221008）

摘 要：通常我们使用异步完成许多计算型的耗时操作，取得应用程序运行所需要的部分数据，再将它们绑定在UI中呈现，这个过程由于数据量偏大，窗体会出现“失去响应”的情况，而线程技术的使用可以方便的实现并发执行，提升资源的利用率，提高程序处理效率，解除“假死”这种糟糕的体验。本文通过对C＃多线程技术及委托方法的介绍，分析研究了在WinForm窗体开发中解决假死状态的两种方法，给出实例及相关代码，并对这两种方法的特点进行了总结。

关键词：多线程；假死；委托；BackGroundWorker控件
中图分类号：TP311.11 **文献标识码**：A

Research on the Application of Multi-Threading in WinForm Development

ZHOU Lan
(*Xuzhou Finance and Economics Branch,Jiangsu Lianhe Technical Institute,Xuzhou* 221008,*China*)

Abstract: Asynchronous manners are usually adopted to implement lots of time-consuming computing operation,in order to achieve the data required by the application and bind them to be presented in UI.Due to the great amount of data,the form often stops responding.The multi-thread technology can facilitate the implementation of concurrency,promote the resource utilization,improve processing efficiency,and avoid the terrible experience of "suspended animation".Based on the C# multi-thread technology and principal methods,the paper analyzes two solutions to the problems of suspended animation in the WinForm development,provides examples and related code,and summarizes the characteristics of two solutions.

Keywords:multi-threading;suspended animation;commission,BackGroundWorker widget

1 引言(Introduction)

通常我们使用异步完成许多计算型、IO型的复杂、耗时操作，去取得我们的应用程序运行所需要的一部分数据[1]。在取得这些数据后，我们需要将它们绑定在UI中呈现。当数据量偏大时，我们会发现窗体变成了空白面板。此时如果用鼠标点击，窗体标题将会出现“失去响应”的字样，而实际上UI线程仍在工作着，这对用户来说是一种极度糟糕的体验。

我们打个比方：比如在上传图片的时候，我们会对上传成功的图片再进行一些相关的处理，一般保存原图，再生成一张小图给一些应用做预览图。如果读取原图再处理的过程由应用程序实现，读取原图需要时间为1秒，处理图片需要2秒，现在有5张大图，所需要花费的时间就是(1+2)*5=15秒，根据图片的不同，那么单位时间会更多，总计时间就会更久，如果有一百万张图要进行相同的处理，那么我们等待的时间将会是15*1000000秒，大约是174天，这种等待是不是有点儿无法忍受。

那么，我们是不是可以把程序设计更好一点儿，让应用程序在读取文件的时候同时处理上一个已读入的文件，这样就好像是同时在做两件事情，一边烧开水，一边打毛衣，可以尽可能的缩短时间。多线程的引入可以帮助应用程序实现这种更理想的状态，减少客户端的响应，同时也提升了CPU的使用率。

2 线程(Threads)

在CPU制造工艺已经达到了物理极限的今天，除非技术有质的突破来进一步提高处理器的速度，但是，我们所要处理的数据量却没有一刻停止它飞速增长的脚步，所以，并行处理技术将成为未来发现的趋势，并行处理技术的核心是对线程的操作[2]。线程，作为轻量级进程(Lightweight Process，LWP)，是程序执行流的最小单元，多线程是指从软件或者硬件上实现多个线程并发执行的技术。

其实，在开发的应用软件中，大多数线程的数量都不止一个，多个线程可以并发的执行，共享进程的全局变量和堆的数据。它的优势在于，当某个操作陷入长时间的等待，或者，一些计算可能会消耗大量的时间，这时会出现和用户之间的交互中断，如果采用多线程，一个线程等待(负责计算)的时候，其他线程可以执行(另一个线程负责交互)，保证CUP的利用率[3]。下面我们来具体看看，多线程在WinForm窗体开发中是如何解除假死的。

3 解决假死的方法(The method to solve the dead)

什么是假死呢？凡是WinForm的应用程序，如果程序

执行的是一个非常冗长的处理操作(比如文件查询、批量的计算、大量文件的上传或下载等)，程序在执行的时候，用户界面会被锁定，虽然主活动窗口一直在运行，但用户没有办法与程序进行交互，窗体的位置和大小也不能移动和改变，就好像“死”在那里不能动一样，用户不能产生良好的使用体验[4]。如何做才能使得这个程序有响应，消除这种“假死”的状态呢？答案就是在后台线程中执行这个操作。现在介绍两种方法来消除这种“假死”的现象。一种是利用BackgroundWorker控件实现；另一种是采用线程加委托的方法消除“假死”。下面我们分别来进行介绍。

3.1 利用BackGroundWorker 控件解除假死

使用BackgroundWorker控件可以在后台单独的线程上执行操作，通常用于数据库操作、文件下载等相对耗时一般要求后台处理的任务，使用起来比较简单[5]。现在我们用BackGroundWorker控件设计一个模拟1000个文件复制过程的进度条，当我们点击“文件复制模拟”按钮时，进度条会显示“正在复制”字样，以及完成复制的百分比，如图1所示，这样做的好处是：用户可以随时了解应用程序执行的进度，而不至于陷入盲目焦躁的等待。

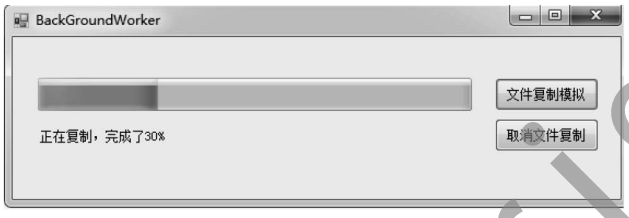

图1 模拟1000个文件复制的进度条

Fig.1 Simulation of the progress of the replication of the 1000 files

模拟1000个文件复制的进度条，代码如下：

```
private void bcWorker_DoWork(object sender,DoWorkEventArgs e)
        {
            ///模拟1000个文件复制
    int complete=0;
    for (inti=1;i<=1000;i++)
    {
    complete=(int)(((float)i/1000)*100);
    bcWorker.ReportProgress(complete);
    if (bcWorker.CancellationPending)
                {
    e.Cancel=true;
    return;
                }
    Thread.Sleep(50);
            }
        }
    private void bcWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
      {
    progress.Value=e.ProgressPercentage;
    lbl_msg.Text="正在复制，完成了"+e.ProgressPercentage.ToString()+"%";
      }
    private void bcWorker_RunWorkerCompleted(object sender,RunWorkerCompletedEventArgs e)
        {
    if (e.Cancelled)
        {
    MessageBox.Show(this,"用户取消了操作","提示",MessageBoxButtons.OK,MessageBoxIcon.Information);
        }
    else
    MessageBox.Show(this,"复制完成","提示",MessageBoxButtons.OK,MessageBoxIcon.Information);
        }
```

3.2 采用线程加委托的方法解决假死

前面我们说过，在winform开发时，如果要对某控件显示的内容进行操作，而这些内容的来源很耗时，会阻塞UI主线程，造成界面的假死，在操作完成之前，界面是不能接收任何响应的。我们可以采用线程+异步委托的方法来确保，即便是耗时的数据操作也不会影响UI的显示和操作的流畅性[6]。

现在想模拟一个图片上传的功能，当点击上传按钮时，弹出“上传文件进度“对话框；当上传结束后，对话框自动关闭。如图2所示。

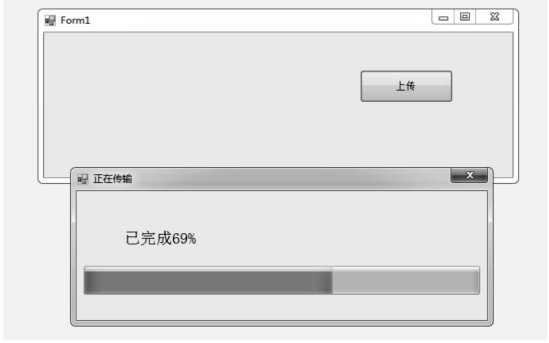

图2 模拟文件上传的进度条

Fig.2 Simulation file upload progress bar

首先实现窗体2中图片上传进度窗体中设置进度条的代码，具体如程序如下：

```
public void setTxt(string title,string txt)
```

```
{
this.Text=title;
this.label1.Text="已完成"+Convert.ToString((int)((float)(float.Parse(txt)/1000)*100))+"%";
this.progressBar1.Value=(int)((float)(float.Parse(txt)/1000)*100);
}
```

其次，在窗体1中，实现模拟文件上传的相关代码，先定义了两个委托变量，用来设置窗体显示上传进度和关闭窗体2，并通过委托调用了窗体2中设置进度条的方法setTxt，程序如下：

```
public delegate void setFormTxt(string title,string txt);
public delegate void closeForm();
public void sendFile()
        {
            for (int i=0; i < 1000; i++)
            {if (f.InvokeRequired)
                {setFormTxt s=new setFormTxt(setFmTxt);
                    f.Invoke(s, new String[]{"正在传输",i.ToString()}); }
                Thread.Sleep(10);
            }
            if (f.InvokeRequired)
           {closeForm c=new closeForm(closeTip);
                f.Invoke(c, null); }
        }
```

最后在上传按钮中实现通过线程调用上传文件方法，并且打开显示上传进度的窗体2，代码程序如下：

```
f=new Form2();
            Thread t=new Thread(sendFile);
t.Start();
f.ShowDialog();
```

当文件上传成功后，调用UI线程上的closeTip方法，关闭窗体2。

跨线程直接访问控件在C＃中是被禁止，还好我们有InvokeRequired，用它就可以解决这个问题。当一个控件的InvokeRequired属性值为真时，说明有一个创建它以外的线程想访问它。此时它将会在内部调用new MethodInvoker(LoadGlobalImage)来完成下面的步骤，这个做法保证了控件的安全[7]。举个例子更如易理解，假如有人想找你借钱，他可以直接在你的钱包中拿吗？这样是不是太不安全了？所以，必须让别人先要告诉你，你再从自己的钱包把钱拿出来借给别人，这样更安全，也更合乎逻辑。

4　结论(Conclusion)

本文通过对C＃多线程技术及委托方法的介绍，分析了在WinForm窗体开发中出现假死状态的原因，并且详细的说明了如何使用BackGroundWorker控件解除假死，以及如何采用线程加异步委托的方法解决假死。当然，在实际的开发过程中解除假死的方法还有很多，比如我们也可以利用Application.DoEvents()来解决这个问题[8]。当然，多线程并非是程序员的圣杯，使用起来也并非一劳永逸，它的使用会增加的内存负担、要求CUP有更强的处理能力、“死锁”也是不可回避的、如果不使用合理的同步结构，以保证独占的数据访问方式，那么，数据损坏也是多线程处理所要面对的一个巨大问题等等。对于以上问题，我们可以对此进行更深入的研究[9]。随着制造水平的提高和技术的发展，CPU已经进入了超线程、多核的时代，相信在不久的未来，我们一定可以用更优化的方式，设计出更加实用的、高效的应用程序，带给用户全新的操作体验。

参考文献(References)

[1] CAI Yunfei,TANG Zhenmin,ZHAO Chunxia.New Layered SOA-Based Architecture for Multi-Robots Cooperative Online SLAM[J].Chinese Journal of Electronics,2014,01:25-30.

[2] Chen.S.M.,J.M.Tan.Handling Multicriteria Fuzzy Decision-Making Problems Based on VagueSet-Theory[J].Fuzzy Setsand Systems,1994,67(2):163-172.

[3] Colvin,J.,Tobler,N.,Anderson,J.A..Productivity and Multiscreen Computer Displays[J].Rocky Mountain Communication Review,2007,2(01):31-53.

[4] 庞丽萍.操作系统原理(第四版)[M].北京:华中科技大学出版社,2015.

[5] (美)斯托林斯.陈向群,译.操作系统:精髓与设计原理(原书第6版)[M].北京:机械工业出版社,2010.

[6] (美)Andrew Troelsen.精通C#(第6版)[M].北京:人民邮电出版社,2013.

[7] (英)里・麦克莱恩・霍尔(Gary McLean Hall).C#敏捷开发实践[M].北京:人民邮电出版社,2016.

[8] (美)阿坝哈瑞,(美)阿坝哈瑞.著果壳中的C#——C#50权威指南[M].北京:水利水电出版社,2013.

[9] 秦婧.构建高质量的C#代码[M].北京:清华大学出版社,2011.

作者简介：

周　岚(1977-)，女，硕士，副教授.研究领域：程序设计，软件开发与数据库.