Ms.Jacky
用.NET来创造互联网的新符号. C#,ASP.NET,XML,JS,AJAX等WEB开发技术支持!

Visual C#中使用线程

Posted on 2008-04-06 12:26 K. 阅读(814) 评论(1) 编辑 收藏 网摘 所属分类: 技术文章, 代码发布 .

简介

编写多线程 Microsoft® 消息队列 (MSMQ) 触发器应用程序向来是一件让人畏惧的事情。不过，.NET 框架线程和消息类的出现使这项工作变得比以前容易了。这些类允许您使用任何适用于 .NET 框架的语言来编写多线程应用程序。以前，像 Microsoft Visual Basic® 之类的工具对线程的支持十分有限。因此不得不使用 C++ 来编写多线程代码，通过 Visual Basic 构建由多个过程或 ActiveX DLL 组成的解决方案（这种解决方案一点也不理想），或者干脆完全放弃多线程。使用 .NET 框架，您可以构建各种多线程应用程序，而不用考虑选择使用哪种语言。

本文将逐步介绍构建侦听并处理来自 Microsoft 消息队列的多线程应用程序的过程。本文将着重讨论两个名称空间 System.Threading 和 System.Messaging。示例代码是用 C# 语言编写的，但您可以轻松地将其转换为您所使用的语言。

线程背景
在 Win32 环境中，线程有三种基本模式：单线程、单元线程和自由线程。

单线程
您最初编写的某些应用程序很可能是单线程应用程序，仅包含与应用程序进程对应的线程。进程可以被定义为应用程序的实例，拥有该应用程序的内存空间。大多数 Windows 应用程序都是单线程的，即用一个线程完成所有工作。

单元线程
单元线程是一种稍微复杂的线程模式。标记用于单元线程的代码可以在其自己的线程中执行，并限制在自己的单元中。线程可以被定义为进程所拥有的实体。处理时将调度该进程。在单元线程模式中，所有线程都在主应用程序内存中各自的子段范围内运行。此模式允许多个代码实例同时但独立地运行。例如，在 .NET 之前，Visual Basic 仅限于创建单元线程组件和应用程序。

自由线程
自由线程是最复杂的线程模式。在自由线程模式中，多个线程可以同时调用相同的方法和组件。与单元线程不同，自由线程不会被限制在独立的内存空间。当应用程序必须进行大量相似而又独立的数学计算时，您可能需要使用自由线程。在这种情况下，您需要生成多个线程使用相同的代码示例来执行计算。可能 C++ 开发人员是仅有的编写过自由线程应用程序的应用程序开发人员，因为像 Visual Basic 6.0 这样的语言几乎不可能编写自由线程应用程序。

常用链接

我的随笔
我的空间
我的短信
我的评论
更多链接

留言簿

给我留言
查看私人留言

我参加的小组

web标准设计
ASP.NET
Web技术联盟

我参与的团队

成都.NET俱乐部(0/0)

我的标签

马云(1)
创业(1)
计科系(1)
网站(1)

使用线程模式

为了使您对线程模式有一定的概念，我们可以将其想象为从一所屋子搬到另一所屋子。如果您采用单线程方法，则需要您自己完成从打包到扛箱子再到拆包的所有工作。如果您使用单元线程模式，则表示您邀请了好朋友来帮忙。每个朋友在一个单独的房间里工作，并且不能帮助在其他房间工作的人。他们各自负责自己的空间和空间内的物品搬运。如果您采用自由线程方法，您仍然邀请相同的朋友来帮忙，但是所有朋友可以随时在任何一个房间工作，共同打包物品。与此类似，您的房子就是运行所有线程的进程，每个朋友都是一个代码实例，搬运的物品为应用程序的资源和变量。

本示例解释了不同线程模式的优点和缺点。单元线程比单线程要快，因为有多个组件实例在工作。在某些情况下，自由线程比单元线程更快更有效，这是因为所有事情同时发生，并且可以共享所有资源。但是，当多线程更改共享资源时，这可能会出现问题。假设一个人开始使用箱子打包厨房用具，此时另一个朋友进来了，要使用同一个箱子打包浴室的东西。第一个朋友在箱子上贴上了"厨房用具"，另一个朋友用"洗漱用品"标签覆盖了原标签。结果，当您拆包时，就会发生将厨房用品搬到浴室的情况。

示例应用程序

第一步是要检查示例应用程序的设计。应用程序将生成多个线程，每个线程都侦听来自 MSMQ 队列的消息。本示例使用两个类，主 Form 类和自定义 MQListen 类。Form 类将处理用户界面并创建，管理和破坏辅助线程。MQListen 类包含所有代码，包括辅助线程运行所需的消息队列因素。

准备应用程序

要启动应用程序，请打开 Visual Studio .NET 并创建一个名为 MultiThreadedMQListener 的新 C# Windows 应用程序。打开窗体的属性，将其命名为 QueueListenerForm。画出初始窗体后，将两个标签、两个按钮、一个状态栏和两个文本框拖放到窗体上。将第一个文本框命名为 Server，第二个文本框命名为 Queue。将第一个按钮命名为 StartListening，第二个按钮命名为 StopListening。可以保留状态栏的默认名称 statusBar1。
下一步，单击 Project（项目）菜单并单击 Add Reference（添加引用），以向 System.Messaging 名称空间添加一个引用。在 .NET 组件列表中找到并选择 System.Messaging.Dll。名称空间包含与 MSMQ 队列通信所使用的类。
下一步，单击 File（文件）菜单，然后单击 Add New Item（添加新项），以在项目中添加一个新类。选择 Class（类）模板并将其命名为 MQListen。在类的顶部添加下列 using 语句：

```
// C#
using System.Threading;
using System.Messaging;
```

System.Threading 名称空间允许您访问所有必要的线程功能，在本例中，您可以访问 Thread 类和 ThreadInterruptException 构造函数。该名称空间还包括许多其他高级功能，本文不作详细讨论。System.Messaging 名称空间允许您访问 MSMQ 功能，包括向队列发送消息和接收队列消息。在本例中，您将使用 MessageQueue 类来接收消息。还必须在主窗体代码中添加 using System.Threading。

所有引用就位后，您就可以开始编写代码了。

随笔分类(25)

AJAX技术(1)
Database(1)
Js技术
VS技术
WEB开发(1)
代码发布(3)
个人资料
技术工具
技术文章(13)
其他文章(6)
网络安全

随笔档案(23)

2008年4月 (19)
2007年11月 (4)

相册

.NET Group

MyLink

最新随笔

1. OLEDB是什么玩艺儿?
2. ASP.NET 和 IIS 配置自定义扩展名网页
3. 网页设计必须注意的29个问题
4. Web 2.0网站流行使用的颜色元素
5. 动漫中心展
6. Internet Explorer 8 Beta1提供中文版下载
7. Atlas：微软的Ajax

辅助线程

首先需要构建封装所有线程工作的 MQListen 类。将下列代码插入 MQListen 中。

```
 1  public class MQListen
 2{
 3|private string m_MachineName;
 4|private string m_QueueName;
 5|
 6|// 构造函数接收必要的队列信息。
 7|public MQListen(string MachineName, string QueueName)
 8{
 9|m_MachineName = MachineName;
10|m_QueueName = QueueName;
11}
12|
13|
14|// 每个线程用来侦听 MQ 消息的一种唯一方法
15|public void Listen()
16{
17|// 创建一个 MessageQueue 对象。
18|System.Messaging.MessageQueue MQ = new
19|System.Messaging.MessageQueue();
20|
21|// 设置 MessageQueue 对象的路径属性。
22|MQ.Path = m_MachineName + "\\private$\\" + m_QueueName;
23|
24|// 创建一个 Message 对象。
25|System.Messaging.Message Message = new
26|System.Messaging.Message();
27|// 重复上述步骤，直到收到中断。
28|while (true)
29{
30|try
31{
32|// 休眠以在中断发出时捕捉中断。
33|System.Threading.Thread.Sleep(100);
34|// 将 Message 对象设置为与接收函数的结果相等。
35|// 持续时间（天、小时、分钟、秒）。
36|Message = MQ.Receive(new TimeSpan(0, 0, 0, 1));
37|
38|// 显示已接收消息的标签
39|System.Windows.Forms.MessageBox.Show(" Label: " + Message.Label);
```

8. 什么是AJAX？
9. 马云致天下所有创业者（转）
10. Visual C#中使用线程


```
40 |
41 ├}
42 |catch (ThreadInterruptedException e)
43 {
44 |// 从主线程捕捉 ThreadInterrupt 并退出。
45 |Console.WriteLine("Exiting Thread");
46 |Message.Dispose();
47 |MQ.Dispose();
48 |break;
49 ├}
50 |catch (Exception GenericException)
51 {
52 |// 捕捉接收过程中抛出的所有异常。
53 |Console.WriteLine(GenericException.Message);
54 ├}
55 ├}
56 ├}
57 └}
58
59
```

代码讨论

MQListen 类包含一个不同于构造函数的函数。该函数封装每个辅助线程要执行的所有工作。在主线程中，您向线程构造函数传递一个对此函数的引用，以便在启动线程时执行该函数。

Listen 所做的第一件事情是设置一个消息队列对象。MessageQueue 构造函数通过三种实现进行重载。第一种实现使用两个参数：一个字符串参数，指定侦听队列的位置；一个布尔值参数，指示是否为访问队列的第一个应用程序赋予独占读取队列的权限。第二种实现只使用队列路径参数，第三种实现不使用参数。为了简便起见，您可以使用第三种实现，在下一行分配路径。

如果您引用了队列，则必须创建一个消息对象。消息构造函数也有三种实现方式。如果您想将消息写入队列，则可以使用前两种实现。这两种实现采用两个对象：一个是位于消息正文中的对象；一个是定义如何将对象序列化到消息正文的 IMessageFormatter 对象。在本例中，您将从队列中读取数据，以初始化空的消息对象。

初始化对象后，您需要输入执行所有工作的主循环。然后，当主线程调用 Interrupt 终止这些线程时，则只有在线程处于等待、睡眠或连接状态下才会被中断。如果没有处于上述三种状态，则要等到下次进入这三种状态中的一种时才会被中断。要确保辅助线程进入等待、睡眠或连接状态，请调用位于 System.Threading 名称空间的 Sleep 方法。对于使用过 Windows API 睡眠函数的 C++ 和 Visual Basic 开发人员而言，Sleep 方法并不陌生。它只使用一个参数：线程处于睡眠状态的毫秒数。如果您从未调用过 Sleep，辅助线程将永远不会进入可以接收中断请求的状态，而会无限制地继续下去，除非您手动关闭进程。

MQ Receive 方法有两种实现。第一种实现不使用参数，将一直等待接收消息。第二种实现（本例使用这种实现）使用 TimeSpan 对象指定一个超时值。TimeSpan 构造函数包含四个参数：日、小时、分钟和秒。在本例中，Receive 方法在超时和返回前将等待一秒种。

收到的消息将被分配给先前创建的消息对象，然后，便可以对其进行处理了。本例打开一个带有标签的消息框，并删除了此消息。如果您想在实际使用中采用此代码，则可以在此处放置任何消息处理代码。

当辅助线程收到 Interrupt 请求后，将发出一个 ThreadInterruptedException 异常。要捕捉此异常，请在 try-catch 块中包含 Sleep 和 Receive 函数。您应当指定两个捕获：第一个用于捕获中断异常，第二个用于处理捕获到的错误异常。捕获到中断异常时，请首先将其写入线程正在退出的调试窗口。下一步，对队列对象和消息对象调用 Dispose 方法，以保证所有内存都被清空并发送到内存回收器。最后，中断 while 循环。

函数退出 while 循环后，关联的线程也将立即结束，代码为 0。在调试窗口，您将看到一则消息，例如“The thread '' (0x660) has exited with code 0 (0x0)”（线程 '' (0x660) 已经退出，代码为 0 (0x0)）。现在，线程已经退出该环境，并已自动被破坏。主线程和辅助线程都不需要执行专门的清除操作。

主窗体

下一步是向窗体添加代码以生成辅助线程并针对各辅助线程启动 MQListen 类。首先，请向窗体添加下列函数：

```
1  private void StartThreads()
2{
3 int LoopCounter; // 线程计数
4 StopListeningFlag = false; // 跟踪辅助线程是否应当
5 // 终止的标志。
6 
7 // 将一个包含 5 个线程的数组声明为辅助线程。
8 Thread[] ThreadArray = new Thread[5];
9 
10 // 声明包含辅助线程的所有代码的类。
11 MQListen objMQListen = new
12 MQListen(this.ServerName.Text,this.QueueName.Text);
13 
14 for (LoopCounter = 0; LoopCounter < NUMBER_THREADS; LoopCounter++)
15{
16 // 创建一个 Thread 对象。
17 ThreadArray[LoopCounter] = new Thread(new
18 ThreadStart(objMQListen.Listen));
19 // 启动线程将调用 ThreadStart 委托。
20 ThreadArray[LoopCounter].Start();
```

```
}

statusBar1.Text = LoopCounter.ToString() + " listener threads started";

while (!StopListeningFlag)
{
// 等待用户按下停止按钮。
// 在等待过程中，让系统处理其他事件。
System.Windows.Forms.Application.DoEvents();
}

statusBar1.Text = "Stop request received, stopping threads";
// 向每个线程发送一个中断请求。
for (LoopCounter = 0;LoopCounter < NUMBER_THREADS; LoopCounter++)
{
ThreadArray[LoopCounter].Interrupt();
}

statusBar1.Text = "All Threads have been stopped";
}
```

代码讨论

要启动此函数，请创建一个包含 5 个项目的线程数组。此数组将保持对所有线程的引用，以备将来使用。

MQListen 类的构造函数使用两个参数：包含消息队列的计算机名以及要侦听的队列的名称。构造函数使用文本框中的值来为这两个参数赋值。

要创建线程，您需要进入循环以初始化每个线程对象。Thread 构造函数要求您向其传递一个委托，该委托在调用线程的 Start 方法时指向要调用的函数。您希望线程开始使用 MQListen.Listen 函数，但该线程并不是一个委托。为了满足线程构造函数的要求，您必须传递一个 ThreadStart 对象，该对象将创建一个给定函数名称的委托。此时，请向 ThreadStart 对象传递一个对 MQListen.Listen 函数的引用。由于该数组元素已被初始化，请立即调用 Start 来开始线程。

所有线程开始后，请用相应的消息来更新窗体中的状态栏。随着线程的运行和侦听队列，主线程将等待用户请求应用程序停止侦听。为此，主线程将进入一个 while 循环，直至您单击 StopListening 按钮更改 StopListeningFlag 的值。在此等待循环中，将允许应用程序使用 Forms.Application.DoEvents 方法处理其他需要处理的工作。对于熟悉 Visual Basic 的读者来说，这一点与旧的 DoEvents 方法相同。对于熟悉 C++ 的读者来说，这等于编写一个 MSG 泵。

当用户单击 StopListening 按钮时，该循环将退出并进入线程关闭代码。要关闭所有线程，代码必须检查线程数组，并向每个线程发送一个中断信号。在此循环内部，请对数组中的每个线程调用 Interrupt 方法。调用此方法之前，MQListen 类中的代码将继续正常执行。因此，您可以对每个辅助线程调用 Interrupt，而不必考虑线程是否正在处理其他事件。完成后，线程类将处理所有线程的清除。最后，请在退出前更新主窗体中的状态栏。

现在，您需要在按钮后添加代码。请向 StartListening 按钮的 Click 事件添加以下代码：

```
statusBar1.Text = "Starting Threads";

 StartThreads();

 这将更新状态栏并调用 StartThreads 方法。对于 StopListening 按钮，您只需使用以下代码将 StopListeningFlag 设置为 True：


// C#

 StopListeningFlag = true;

  最后一步是为 StopListeningFlag 添加窗体级的变量。请在窗体代码的顶部添加以下行：

// C#

 private bool StopListeningFlag = false;

```

要测试应用程序，您可以下载 MQWrite，这是一个写入消息队列的示例应用程序。

多线程代码问题

您已经完成了示例代码，因此您已经具备编写自己的多线程应用程序所需的工具。线程可以显著提高某些应用程序的性能和可伸缩性。在功能增强的同时，您还必须了解线程有危险的一面。使用线程可能会破坏您的应用程序，这样的情况确实存在。线程可能会阻止运行，造成无法预料的后果，甚至会导致应用程序停止运行。

如果您有多个线程，请确保它们之间不存在互相等待以到达某一点或完成的情况。如果操作错误，可能会导致死锁状态，两个线程都无法完成，因为它们都在相互等待。

如果多线程要求访问不能轻易共享的资源（如软盘驱动器、串行端口或红外线端口），您可能需要避免使用线程或需要使用一种更高级的线程工具（如 synclocks 或 mutexes）来管理并发性。如果两个线程试图同时访问这些资源，其中一个线程将无法获得资源，或者会导致数据损坏。

使用线程的另一个常见问题是竞争状态。如果一个线程正在将数据写入文件，而另一个线程正在从该文件中读取数据，您将无

法知道哪个线程先完成。这种情况称为竞争状态，因为两个线程都在竞相到达文件末尾。如果读取线程快于写入线程，则将返回无法预料的结果。

使用线程时，还应当考虑所有线程是否都能够完全独立地进行工作。如果确实需要来回传递数据，在数据相对简单的情况下，只要小心操作即可。传递复杂对象时，来回移动这些对象的封送代价将十分可观。这将导致操作系统管理的额外开销并且会降低总体性能。

另一个问题是将代码转交给其他开发人员的传递成本。虽然 .NET 确实使线程变得容易，但请注意，维护您代码的下一位开发人员必须了解要使用的线程。尽管这不是避免使用线程的理由，但是它充分说明了应该提供足够的代码注释。

这些问题本身并不能打消您使用线程的热情，但您在设计应用程序和决定是否使用线程时应该考虑到这些问题。遗憾的是，本文无法详细论述某些避免这些问题的方法。如果您已决定使用线程但遇到了上述某些问题，请检查 synclocks 或 mutexes 看是否能解决问题或引导您使用其他解决方案。

总结
有了上述信息，您就可以编写使用线程的应用程序。不过，在编写过程中，请记住上面提到的问题。如果使用得当，那么，与单线程相比，多线程应用程序将具有更好的性能和可伸缩性。但是，如果使用不当，使用线程会适得其反，并且会导致应用程序不稳定。

Feedback

发表评论

昵称: [登录] [注册]

主页:

邮箱: (仅博主可见)

评论内容: 闪存 个人主页

提交 登录 注册

[使用Ctrl+Enter键快速提交评论]

个人主页上线测试中
今天你闪了吗?
2009博客园纪念T恤