一路转圈的雪人

人生彪悍，但也需解释

公告

07年不小心从越南语专业毕业，10年开始转行码农界。喜欢乱写些东西，先把一些想法先写下来，以后慢慢拓展。其实书写是为了更好地思考，能知其然更知其所以然。

喜欢玩劲乐团（能全连31级的高级《追忆》哦）,DJMAX,跑跑卡丁车。

多线程之旅七——GUI线程模型，消息的投递(post)与处理（IOS开发前传）

基于消息的GUI构架

在过去的日子中，大部分编程语言平台的GUI构架几乎没有发生变化。虽然在细节上存在一些差异，比如在功能和编程风格上，但大部分都是采用了相同的构架来响应用户输入以及重新绘制屏幕。这种构架可以被总结为“单线程且基于消息”。

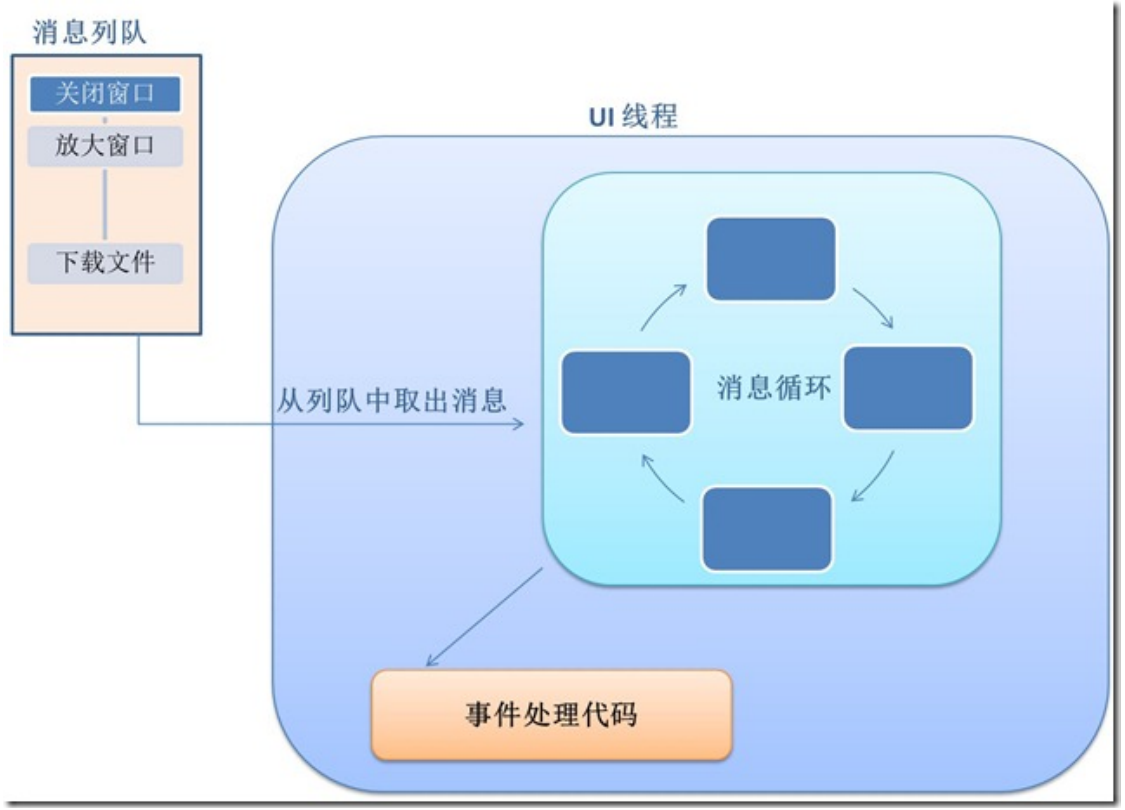
```
Message msg;

While(GetMessage(msg))
{
    TranslateMessage(msg);
    DispatchMessage(msg);
}
```

这段代码可以称为消息循环。在这个循环中，执行顺序是串行的，一个GetMessage只能在前一个GetMessage执行完以后才能执行。

拿WPF或WindowsForm举例，每个线程至少会创建一个拥有消息列队的窗口，并且这个线程的任务之一就是处理列队中的各个消息。只要在应用程序中调用了Application.Run,那么执行这个方法的线程就默认被赋予了这个任务。随后的所有GUI事件，例如用户引发的事件（点击按钮，关闭窗口等），系统引发的事件（重绘窗口，调整大小等），以及应用程序中自定义组件的特定事件等，都将把相应的消息投递给这个消息列队来实现。这意味着，在调用了run之后，随后发生的大部分工作都是由事件处理器为了响应GUI事件而生成的。

如图：



GUI线程

Gui线程负责取走（get）和分发（dispatch）消息，同时负责描绘界面，如果GUI线程阻塞在分发处理消息这一步的话，那么消息就会在消息队列中积累起来，并等待GUI线程回到消息列队来。

如果阻塞的是一个长时间的操作，比如下载一个文件的话，假设10秒钟，那么用户在10秒钟内都不能进行任何操作，因为线程没法获取新的消息进行处理。

2011年4月 (1)
2011年3月 (1)
2011年2月 (1)
2011年1月 (8)

C#编程

Extjs

WCF

最新评论

1. Re:浅谈SQL SERVER中的物理联接算法
很好理解
--茗::流

2. Re:谈谈javascript中的prototype与继承
@coco543oj.func1=function(){var a=0;};oj是一个函数实例，你只是给一个实例的属性赋值了，当然没法访问。
oj.prototype.func2=function(){.....
--一路转圈的雪人

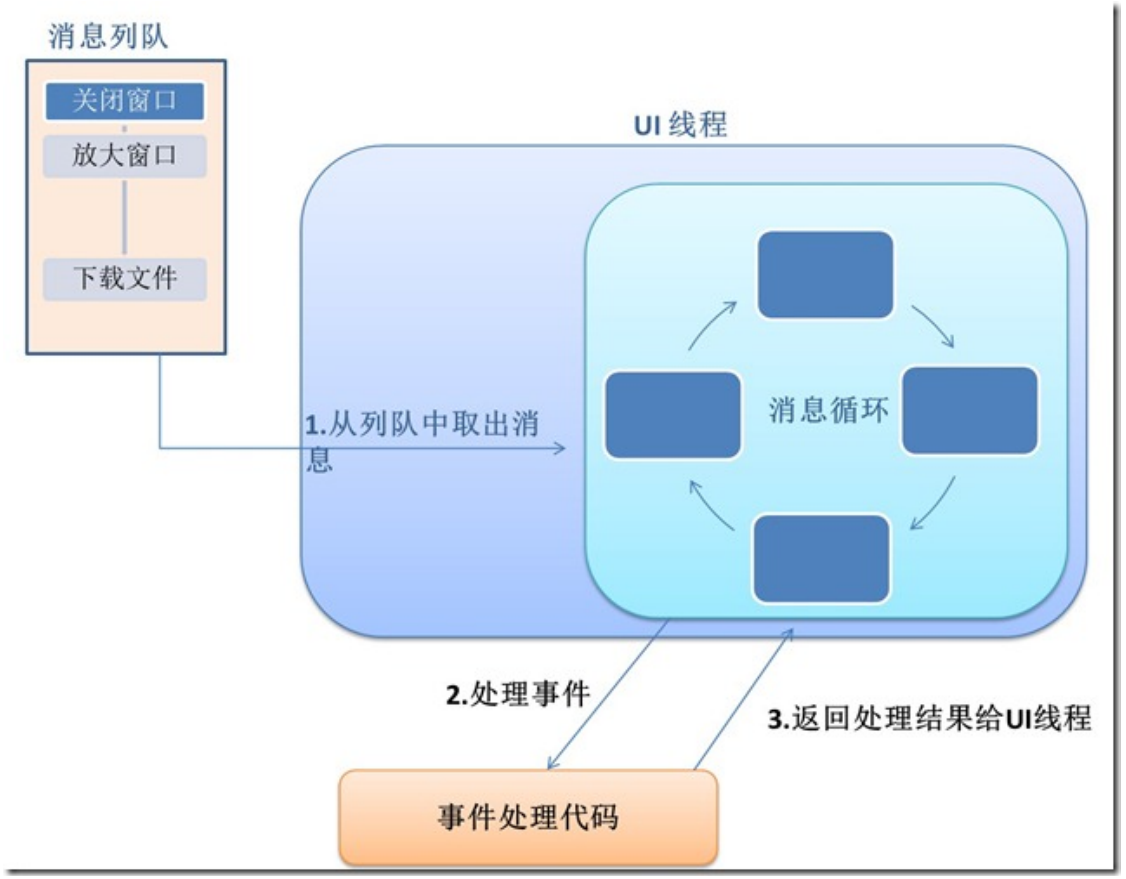
3. Re:多线程之旅：解读async和await
@gw2010要看SynchronizationContext里包含了多少线程。UI线程只有一个...
--一路转圈的雪人

4. Re:谈谈javascript中的prototype

这就是为什么在Windows中存在着MsgWaitForMultipleObjects的原因，这个API使得线程在等待的同时仍然可以运行消息循环。在.NET中，你连这个选择都没有。

消息分发时要考虑到复杂的重入性问题，很难确保一个事件处理器阻塞时，可以安全分发其他GUI事件以响应消息。

因此，一种相对而言更容易掌握的解决方法就是只有在GUI线程中的代码才能够操纵GUI控件，在更新GUI时所需要的其他数据和计算都必须在其他线程中完成，而不是在GUI线程上。如图：



通常这意味着把工作转交给线程池完成，然后在得到结果后把结果合并回GUI线程上。这也就是我们接下来要介绍的两个类。

SynchronizationContext 和 BackgroundWorker

SynchronizationContext 对不同线程间的调度操作进行同步，把一些异步操作的结果Post回GUI线程里。

WPF中DispatcherSynchronizationContext的实现

```
public  override  void  Post(SendOrPostCallback  d,  object  state)
{
    _dispatcher.Beginlnvoke(DispatcherPriority.Normal,  d,  state);
    }
    public  override  void  Send(SendOrPostCallback  d,  object  state)
    {
    _dispatcher.lnvoke(DispatcherPriority.Normal,  d,  state);
}
```

有些情况下，如在控制台中我们不能通过SynchronizationContext类的Current属性获取SynchronizationContext实例，我们包装了一下这个方法。

```
private static AsyncCallback SyncContextCallback(AsyncCallback callback) {
   // Capture the calling thread's SynchronizationContext-derived object
   SynchronizationContext sc = SynchronizationContext.Current;

   // If there is no SC, just return what was passed in
   if (sc == null) return callback;
```

1. 毕业5年回忆录：那年我毕业了(22)
2. 一道简单的面试题，据说90%人不能在30分钟内做出来(20)
3. 浅谈SQL SERVER中的物理联接算法(12)
4. 深入 聚集索引与非聚集索引(一)(10)
5. 多线程之旅：解读async和await(10)
6. 谈谈.NET中常见的内存泄露问题——GC、委托事件和弱引用(10)
7. 牢骚与javascript中的this(10)
8. 谈谈javascript中的prototype与继承(8)
9. 2012年读书计划 ，争取在世界末日来之前看完(7)
10. 用TCP/IP实现自己简单的应用程序协议:最后再返回来看HTTP协议(6)

推荐排行榜

1. 深入 聚集索引与非聚集索引(一)(18)
2. 谈谈.NET中常见的内存泄露问题——GC、委托事件和弱引用(13)
3. 浅谈SQL SERVER中的物理联接算法(12)
4. 和我一起来学

```
   // Return a delegate that, when invoked, posts to the captured SC a method that
   // calls the original AsyncCallback passing it the IAsyncResult argument
   return asyncResult => sc.Post(result => callback((IAsyncResult)result), asyncResult);
}
```

这个方法将一个普通的AsyncCallback方法转换成特殊的AsyncCallback 方法，它通过SynchronizationContext 来调用。这样无论线程模型中是否含有GUI线程，都可以正确的调用。

```
internal sealed class MyWindowsForm : Form {
   public MyWindowsForm() {
      Text = "Click in the window to start a Web request";
      Width = 400; Height = 100;
   }

   protected override void OnMouseClick(MouseEventArgs e) {
      // The GUI thread initiates the asynchronous Web request
      Text = "Web request initiated";
      var webRequest = WebRequest.Create("http://Wintellect.com/");
      webRequest.BeginGetResponse(SyncContextCallback(ProcessWebResponse), webRequest);
      base.OnMouseClick(e);
   }

   private void ProcessWebResponse(IAsyncResult result) {
      // If we get here, this must be the GUI thread, it's OK to update the UI
      var webRequest = (WebRequest)result.AsyncState;
      using (var webResponse = webRequest.EndGetResponse(result)) {
         Text = "Content length: " + webResponse.ContentLength;
      }
   }
}
```

iOS（一）ObjectC的语法(11)
5. 详解JavaScript中的函数与闭包(9)
6. 和我一起来学iOS（二）iOS中的一些约定、模式与三种回调机制(7)
7. 和我一起来学iOS（五）负责表现的CALayer(6)
8. 多线程之旅之三——Windows内核对象同步机制(6)
9. 谈.NET,由编译器开始谈起(6)
10. 字节和字符,对信息进行编码(5)

这其实就是AsyncOperationManager的基本原理。

```
public  static  class  AsyncOperationManager
{
    public  static  SynchronizationContext  {  getj  setj  }
    public  static  AsyncOperation  CreateOperation( object  userSuppliedState ) ;
}
```

BackGroundWorker是在前面所说的基础上构建起来的更高层次的抽象，它对GUI程序中一些最常用的操作给出了规范的定义。有三个事件：

DoWork 、 ProgressChanged 和 RunWorkerCompleted

在程序中调用RunWorkerAsync方法则会启动DoWork事件的事件处理，当在事件处理过程中，调用 ReportProgress方法则会启动ProgressChanged事件的事件处理，而当DoWork事件处理完成时，则会触发 RunWorkerCompleted事件。

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            backgroundWorker1.WorkerReportsProgress = true;
            backgroundWorker1.WorkerSupportsCancellation = true;
        }

        private void startAsyncButton_Click(object sender, EventArgs e)
        {
            if (backgroundWorker1.IsBusy != true)
            {
                // Start the asynchronous operation.
                backgroundWorker1.RunWorkerAsync();
            }
        }

        private void cancelAsyncButton_Click(object sender, EventArgs e)
        {
```

```
            if (backgroundWorker1.WorkerSupportsCancellation == true)
            {
                // Cancel the asynchronous operation.
                backgroundWorker1.CancelAsync();
            }
        }

        // This event handler is where the time-consuming work is done.  private void
backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {
            BackgroundWorker worker = sender as BackgroundWorker;

            for (int i = 1; i <= 10; i++)
            {
                if (worker.CancellationPending == true)
                {
                    e.Cancel = true;
                    break;
                }
                else
                {
                    // Perform a time consuming operation and report progress.
                    System.Threading.Thread.Sleep(500);
                    worker.ReportProgress(i * 10);
                }
            }
        }

        // This event handler updates the progress.  private void backgroundWorker1_ProgressChanged(object
sender, ProgressChangedEventArgs e)
        {
            resultLabel.Text = (e.ProgressPercentage.ToString() + "%");
        }

        // This event handler deals with the results of the background operation.  private void
backgroundWorker1_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
        {
            if (e.Cancelled == true)
            {
```

```
                resultLabel.Text = "Canceled!";
            }
            else if (e.Error != null)
            {
                resultLabel.Text = "Error: " + e.Error.Message;
            }
            else
            {
                resultLabel.Text = "Done!";
            }
        }
    }
```

分类: 多线程

绿色通道： 好文要顶 关注我 收藏该文 与我联系 

一路转圈的雪人
关注 - 5
粉丝 - 178
+加关注

1 0

(请您对文章做出评价)

« 上一篇：多线程之旅六——异步编程模式，自己实现IAsyncResult
» 下一篇：总结自己的Git常用命令

posted on 2012-11-17 20:47 一路转圈的雪人 阅读(1780) 评论(1) 编辑 收藏

发表评论

#1楼 2012-11-19 09:42 | john23.net

学习了

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部