xugang

从2007年开始，记录一个DotNET程序员的成长

C#多线程学习(三) 生产者和消费者

前面说过，每个线程都有自己的资源，但是代码区是共享的，即每个线程都可以执行相同的函数。这可能带来的问题就是几个线程同时执行一个函数，导致数据的混乱，产生不可预料的结果，因此我们必须避免这种情况的发生。

C#提供了一个关键字lock，它可以把一段代码定义为互斥段（critical section），互斥段在一个时刻内只允许一个线程进入执行，而其他线程必须等待。在C#中，关键字lock定义如下：

lock(expression) statement_block

expression代表你希望跟踪的对象，通常是对象引用。

如果你想保护一个类的实例，一般地，你可以使用this；

如果你想保护一个静态变量（如互斥代码段在一个静态方法内部），一般使用类名就可以了。

而statement_block就是互斥段的代码，这段代码在一个时刻内只可能被一个线程执行。

下面是一个使用lock关键字的典型例子，在注释里说明了lock关键字的用法和用途。

示例如下：

```
using System;
```

```
using System.Threading;


namespace ThreadSimple
{
   internal class Account
   {
      int balance;
      Random r = new Random();

      internal Account(int initial)
      {
         balance = initial;
      }

      internal int Withdraw(int amount)
      {
         if (balance < 0)
         {
            //如果balance小于0则抛出异常
            throw new Exception("Negative Balance");
         }
         //下面的代码保证在当前线程修改balance的值完成之前
         //不会有其他线程也执行这段代码来修改balance的值
         //因此，balance的值是不可能小于0 的
         lock (this)
         {
            Console.WriteLine("Current Thread:"+Thread.CurrentThread.Name);
```

```
            //如果没有lock关键字的保护，那么可能在执行完if的条件判断之后
            //另外一个线程却执行了balance=balance-amount修改了balance的值
            //而这个修改对这个线程是不可见的，所以可能导致这时if的条件已经不成立了
            //但是，这个线程却继续执行balance=balance-amount，所以导致balance可能小于0
            if (balance >= amount)
            {
                Thread.Sleep(5);
                balance = balance - amount;
                return amount;
            }
            else
            {
                return 0; // transaction rejected
              }
        }
    }
    internal void DoTransactions()
    {
        for (int i = 0; i < 100; i++)
        Withdraw(r.Next(-50, 100));
    }
}

internal class Test
{
    static internal Thread[] threads = new Thread[10];
    public static void Main()
```

```
        {
            Account acc = new Account (0);
            for (int i = 0; i < 10; i++)
            {
                Thread t = new Thread(new ThreadStart(acc.DoTransactions));
                threads[i] = t;
            }
            for (int i = 0; i < 10; i++)
                threads[i].Name=i.ToString();
            for (int i = 0; i < 10; i++)
                threads[i].Start();
            Console.ReadLine();
        }
    }
}
```

Monitor 类锁定一个对象

当多线程公用一个对象时，也会出现和公用代码类似的问题，这种问题就不应该使用lock关键字了，这里需要用到System.Threading中的一个类Monitor，我们可以称之为监视器，Monitor提供了使线程共享资源的方案。

Monitor类可以锁定一个对象，一个线程只有得到这把锁才可以对该对象进行操作。对象锁机制保证了在可能引起混乱的情况下一个时刻只有一个线程可以访问这个对象。

Monitor必须和一个具体的对象相关联，但是由于它是一个静态的类，所以不能使用它来定义对象，而且它的所有方法都是静态的，不能使用对象来引用。下面代码说明了使用Monitor锁定一个对象的情形：

```
    ......
    Queue oQueue=new Queue();
```

```
.....
Monitor.Enter(oQueue);
.....//现在oQueue对象只能被当前线程操纵了
Monitor.Exit(oQueue);//释放锁
```

如上所示，当一个线程调用Monitor.Enter()方法锁定一个对象时，这个对象就归它所有了，其它线程想要访问这个对象，只有等待它使用Monitor.Exit()方法释放锁。为了保证线程最终都能释放锁，你可以把Monitor.Exit()方法写在try-catch-finally结构中的finally代码块里。

对于任何一个被Monitor锁定的对象，内存中都保存着与它相关的一些信息:
其一是现在持有锁的线程的引用;
其二是一个预备队列，队列中保存了已经准备好获取锁的线程;
其三是一个等待队列，队列中保存着当前正在等待这个对象状态改变的队列的引用。

当拥有对象锁的线程准备释放锁时，它使用Monitor.Pulse()方法通知等待队列中的第一个线程，于是该线程被转移到预备队列中，当对象锁被释放时，在预备队列中的线程可以立即获得对象锁。

下面是一个展示如何使用lock关键字和Monitor类来实现线程的同步和通讯的例子，也是一个典型的生产者与消费者问题。
这个例程中，生产者线程和消费者线程是交替进行的，生产者写入一个数，消费者立即读取并且显示（注释中介绍了该程序的精要所在）。

用到的系统命名空间如下:

```
using System;
using System.Threading;
```

首先，定义一个被操作的对象的类Cell，在这个类里，有两个方法：ReadFromCell()和WriteToCell。消费者线程将调用ReadFromCell()读取cellContents的内容并且显示出来，生产者进程将调用WriteToCell()方法

向cellContents写入数据。

示例如下：

```
public class Cell
{
       int cellContents; // Cell对象里边的内容
       bool readerFlag = false; // 状态标志，为true时可以读取，为false则正在写入
       public int ReadFromCell( )
       {
          lock(this) // Lock关键字保证了什么，请大家看前面对lock的介绍
          {
             if (!readerFlag)//如果现在不可读取
             {
                try
                {
                   //等待WriteToCell方法中调用Monitor.Pulse()方法
                   Monitor.Wait(this);
                }
                catch (SynchronizationLockException e)
                {
                   Console.WriteLine(e);
                }
                catch (ThreadInterruptedException e)
                {
                   Console.WriteLine(e);
                }
             }
```

```
            Console.WriteLine("Consume: {0}",cellContents);
            readerFlag = false;
            //重置readerFlag标志，表示消费行为已经完成
            Monitor.Pulse(this);
            //通知WriteToCell()方法（该方法在另外一个线程中执行，等待中）
        }
        return cellContents;
    }

    public void WriteToCell(int n)
    {
        lock(this)
        {
            if (readerFlag)
            {
                try
                {
                    Monitor.Wait(this);
                }
                catch (SynchronizationLockException e)
                {
                        //当同步方法（指Monitor类除Enter之外的方法）在非同步的代码区被调用
                    Console.WriteLine(e);
                }
                catch (ThreadInterruptedException e)
                {
                        //当线程在等待状态的时候中止
```

```
                    Console.WriteLine(e);
                }
            }
            cellContents = n;
            Console.WriteLine("Produce: {0}",cellContents);
            readerFlag = true;
            Monitor.Pulse(this);
            //通知另外一个线程中正在等待的ReadFromCell()方法
        }
    }
}
```

下面定义生产者类 CellProd 和消费者类 CellCons ，它们都只有一个方法ThreadRun()，以便在Main()函数中提供给线程的ThreadStart代理对象，作为线程的入口。

```
public class CellProd
{
   Cell cell; // 被操作的Cell对象
   int quantity = 1; // 生产者生产次数，初始化为1

   public CellProd(Cell box, int request)
   {
      //构造函数
      cell = box;
      quantity = request;
   }
   public void ThreadRun( )
   {
```

```
        for(int looper=1; looper<=quantity; looper++)
        cell.WriteToCell(looper); //生产者向操作对象写入信息
    }
}

public class CellCons
{
    Cell cell;
    int quantity = 1;

    public CellCons(Cell box, int request)
    {
            //构造函数
        cell = box;
        quantity = request;
    }
    public void ThreadRun( )
    {
        int valReturned;
        for(int looper=1; looper<=quantity; looper++)
        valReturned=cell.ReadFromCell( );//消费者从操作对象中读取信息
    }
}
```

然后在下面这个类MonitorSample的Main()函数中，我们要做的就是创建两个线程分别作为生产者和消费者，使用CellProd.ThreadRun()方法和CellCons.ThreadRun()方法对同一个Cell对象进行操作。

```
public class MonitorSample
{
   public static void Main(String[] args)
   {
      int result = 0; //一个标志位，如果是0表示程序没有出错，如果是1表明有错误发生
      Cell cell = new Cell( );

      //下面使用cell初始化CellProd和CellCons两个类，生产和消费次数均为20次
      CellProd prod = new CellProd(cell, 20);
      CellCons cons = new CellCons(cell, 20);

      Thread producer = new Thread(new ThreadStart(prod.ThreadRun));
      Thread consumer = new Thread(new ThreadStart(cons.ThreadRun));
      //生产者线程和消费者线程都已经被创建，但是没有开始执行
      try
      {
   producer.Start( );
   consumer.Start( );

   producer.Join( );
   consumer.Join( );
   Console.ReadLine();
      }
      catch (ThreadStateException e)
      {
   //当线程因为所处状态的原因而不能执行被请求的操作
   Console.WriteLine(e);
```

```
    result = 1;
        }
        catch (ThreadInterruptedException e)
        {
    //当线程在等待状态的时候中止
    Console.WriteLine(e);
    result = 1;
        }
        //尽管Main()函数没有返回值，但下面这条语句可以向父进程返回执行结果
        Environment.ExitCode = result;
    }
}
```

在上面的例程中，同步是通过等待Monitor.Pulse()来完成的。首先生产者生产了一个值，而同一时刻消费者处于等待状态，直到收到生产者的“脉冲(Pulse)”通知它生产已经完成，此后消费者进入消费状态，而生产者开始等待消费者完成操作后将调用Monitor.Pulese()发出的“脉冲”。

它的执行结果很简单：

```
Produce: 1
Consume: 1
Produce: 2
Consume: 2
Produce: 3
Consume: 3
...
...
Produce: 20
```

Consume: 20

事实上，这个简单的例子已经帮助我们解决了多线程应用程序中可能出现的大问题，只要领悟了解决线程间冲突的基本方法，很容易把它应用到比较复杂的程序中去。

posted on 2008-03-23 17:17 钢钢 阅读(3058) 评论(1) 编辑 收藏 网摘 所属分类: C#

刷新评论列表 刷新页面 返回页首

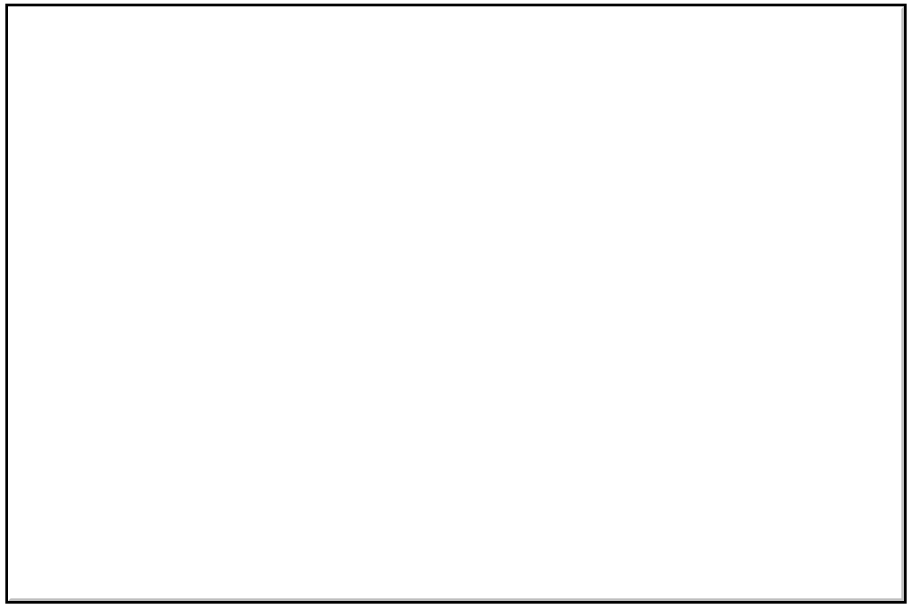
发表评论

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

链接：切换模板
导航：网站首页 个人主页 社区 新闻 博问 闪存 网摘 招聘 找找看 Google搜索

最新**IT**新闻**:**
Delphi 2010初体验
谷歌经济学家：搜索关键词表明美经济正复苏
Facebook应吸取谷歌经验避免重蹈雅虎覆辙
唐骏传授成功秘笈：创业要有自己的“杀手锏”
商业周刊：企业用户不愿甲骨文壮大 称其店大欺客

相关链接**:**
系列教程: C#多线程学习
消费者来揭秘·广告潜规则谁不知道！
著名CEO致英语学习者的公开信
成为优秀翻译者的学习方法
口译积累：全球增长不能仅依赖美国消费者
去和lucy对话，或者练习音标
“微软最有影响力开发者”正在招募

导航
博客园
首页
新随笔
联系
订阅XML
管理

统计
随笔 - 166
文章 - 27
评论 - 579

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
查看留言

我参加的小组

web标准设计
jQuery
读书(Books)
ASP.NET
AJAX
.Net+MySQL
博客园期刊团队

我参与的团队

.NET 控件与组件开发(0/0)
湖南.NET俱乐部(0/0)
ASP.NET AJAX (Atlas)学习(0/0)
asp.net开发团队(0/0)

随笔分类(124)

AJAX(4) (rss)
C#(22) (rss)
DotNET(27) (rss)
GDI(2) (rss)
JavaScript(20) (rss)
MS_SQL(4) (rss)
MySQL(1) (rss)
Oracle(6) (rss)
编程规范(6) (rss)
设计模式(2) (rss)
我的项目(9) (rss)

相关知识(21) (rss)

随笔档案(166)

2008年12月 (1)
2008年11月 (1)
2008年10月 (1)
2008年9月 (2)
2008年8月 (4)
2008年7月 (1)
2008年6月 (4)
2008年5月 (4)
2008年4月 (7)
2008年3月 (8)
2008年2月 (7)
2008年1月 (19)
2007年12月 (26)
2007年11月 (21)
2007年10月 (24)
2007年9月 (15)
2007年8月 (12)
2007年7月 (9)

文章分类(26)

Java学习(1) (rss)
读书笔记(16) (rss)
经典收藏(9) (rss)

收藏夹(199)

ASP.NET 技术(128) (rss)
Enterprise Library(5) (rss)
PetShop 4架构分析 (15) (rss)
网络教程 (36) (rss)
优秀Blog地址(15) (rss)

Blogs

Guushuuse
LoveCherry
Nicky
Q.yuhen
Scott Guthrie

怪怪
蝈蝈俊
黄忠成
李天平
孟岩
木子
萧寒
张子阳

Tools

Convert vbnet to C# (rss)
DotNet API (rss)
My Diagram
Text Translation (rss)
WorldLingo (rss)

Websites

AspAlliance (rss)
C# Corner (rss)
C#开源资源大全 (rss)
C#语言参考视频 (rss)
cnBeta (rss)
CodePlex (rss)
codeusing.com (rss)
connectionstrings.com (rss)
CSDN DotNet
dotNET_程序小作坊 (rss)
DotNetTools (rss)
DotNet开源社区
http://csharp-source.net (rss)
ItPub论坛 (rss)
MSDN中国 (rss)
MSProject 开源技术 (rss)
planet-source-code.com (rss)
Sawin (rss)
开源中国
懒人图库 (rss)
浪曦视频在线 (rss)
中国协议分析网 (rss)

积分与排名

积分 - 216335
排名 - 224

最新评论 XML

1. Re:程序员心理小测试：你是否患上抑郁症？
天啊,我得了13分,我是不是应该去曙光医院看看啊!
--purplesu

2. Re:Donald Knuth 简介
虽然学编程一年了,但对这行业总是迷茫,怎样才算会编程也不知道
--Greenhan

3. Re:一个C#的加锁解锁示例
没看懂什么意思呀
--王维

4. Re:将GridView导入到Excel和word(完全可实现)
我的加了还是倒不出来你呢
--ASP.NET爱好

5. Re:程序员心理小测试：你是否患上抑郁症？
看来，我也得了抑郁怔拉----该怎么办呢----真的好烦啊-------
--伤心雨

阅读排行榜

1. C#多线程学习(一) 多线程的相关概念(9310)
2. IIS日志分析方法及工具(9230)
3. Javascript 刷新框架及页面的方法总集(9020)
4. (新)中华人民共和国劳动合同法(8263)
5. C#多线程学习(二) 如何操纵一个线程(7237)