

一个真正优秀的程序员应该关注于知识的分享

< 2008年6月 >						
日	一	二	三	四	五	六
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	<b>23</b>	<b>24</b>	<b>25</b>	26	27	28
29	30	1	2	3	4	5

与我联系

发短消息

搜索

錦序價漕十垚錦佻勾IT鏈整  
姦緇帆徽  
IT浜y搨鋹欸清鏢°€佻觶鑾  
×蒋浹蹂坏鑾  
戩€佻 鍊 錄賺攢鑾娉郴  
緇燠況鋹慍墜澶氬 IT鏈整  
姦涓氫姦鋹  
www.teamsun.com.cn

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

C# 线程资源同步方式总结

在现代的程序开发中,资源的同步是一个比较重要的课题,在.Net中,对这部分有很丰富类库供我们使用,现在总结一下在各种情况下对资源同步的机制。

1.将字段声明为volatile

当一个字段被声明为volatile时,CLR中一些管理代码和内存的内部机制将负责对字段进行同步,并且总能保证读取到的字段信息都为最新的值,被声明为volatile的字段必须具备以下特征之一

- 1.为引用类型
- 2.一个指针(在不安全代码中)
- 3.sbyte,byte,short,ushort,int,uint,char,float,bool
- 4.一个使用底层类型的枚举类型

2.使用System.Threading.Interlocked 类

在许多增对整数的操作中,我们都比较容易忽视线程的问题,例如执行下列代码

i = i + 1;

实际上,上述代码分为3步骤

- 1). 从内存中,读取i的值
- 2). 将读取出来的值加1
- 3). 将新的值写入内存中

在单线程上,这个操作不会有任何问题,但是当i被多个线程访问时,问题就出现了,对i进行修改的线程,在上述的任何一部都有可能被其它读取线程打断,想象一下,当操作线程执行完第二步,准备将新的值写入内存中时,此时其它读取线程获得了执行权,这时读取到的i的值并不是我们想要的,因此,这个操作不具备原子性,在.Net中,使用Interlocked类能确保操作的原子性,Interlocked类有以下的方法

Increment  
Decrement

随笔分类

- .Net Framework (rss)
- C# 技术(3) (rss)
- J2EE (rss)
- SQL (rss)
- 设计模式 (rss)

随笔档案

2008年6月 (3)

最新评论 XML

1. re: C# 线程资源同步方式总结  
good (Tony Zhou)
2. re: C# 线程资源同步方式总结  
哦 lz和1F,2F, 4F的资料都很棒哦, 特别  
是4F。 在这里说一声, 谢谢了 一个真正优  
秀的程序员应该关注于知识的分享 -----  
-----... (Steven Chen)
3. re: C# 线程资源同步方式总结  
对,对于lock是否需要static,我认为在单例  
模式下,lock关键字锁定的Object对象似乎  
没有必要是static的,声明为private static  
给人的感觉这个对象锁是全局锁,但它又是..  
(VincentWP)
4. re: C# 线程资源同步方式总结  
由于字符串的拘留池机制, 建议不要lock字  
符串。(鼠·神·泪.NET)
5. re: C# 线程资源同步方式总结  
@HelloCode  
private 没错, 是否 static 还要看情况。  
  
@VincentWP  
互相学习,:-) (Angel Lucifer)

上述的方法的参数都为带ref 标识的参数,因此我们说,这些方法保证了数据的原子性,在多线程中,涉及到整数的操作时,数据的原子性值得考虑,Interlocked的操作代码如下

```
int i = 0;
System.Threading.Interlocked.Increment(ref i);
Console.WriteLine(i);
System.Threading.Interlocked.Decrement(ref i);
Console.WriteLine(i);
System.Threading.Interlocked.Exchange(ref i, 100);
Console.WriteLine(i);
```

输出信息如下



3.使用lock关键字

地球人都知道,使用lock关键字可以获取一个对象的独占权,任何需要获取这个对象的操作的线程必须等待以获取该对象的线程释放独占权,lock提供了简单的同步资源的方法,与Java中的synchronized关键字类似。

lock关键字的使用示例代码如下

```
object o = new object();
lock (o)
{
    Console.WriteLine("O");
}
```

4.使用System.Theading.Monitor类进行同步

System.Threading.Monitor类提供了与lock类似的功能,不过与lock不同的是,它能更好的控制同步块,当调用了Monitor的Enter(Object o)方法时,会获取o的独占权,直到调用Exit(Object o)方法时,才会释放对o的独占权,可以多次调用Enter(Object o)方法,只需要调用同样次数的Exit(Object o)方法即可,Monitor类同时提供了TryEnter(Object o,[int])的一个重载方法,该方法尝试获取o对象的独占权,当获取独占权失败时,将返回false,查看如下代码

阅读排行榜

- 1. C# 线程资源同步方式总结(3200)
- 2. .Net 下国际化资源文件的应用(一)(429)
- 3. 基于.Net Attribute 应用的自动验证(340)

评论排行榜

- 1. C# 线程资源同步方式总结(11)
- 2. 基于.Net Attribute 应用的自动验证(2)
- 3. .Net 下国际化资源文件的应用(一)(0)

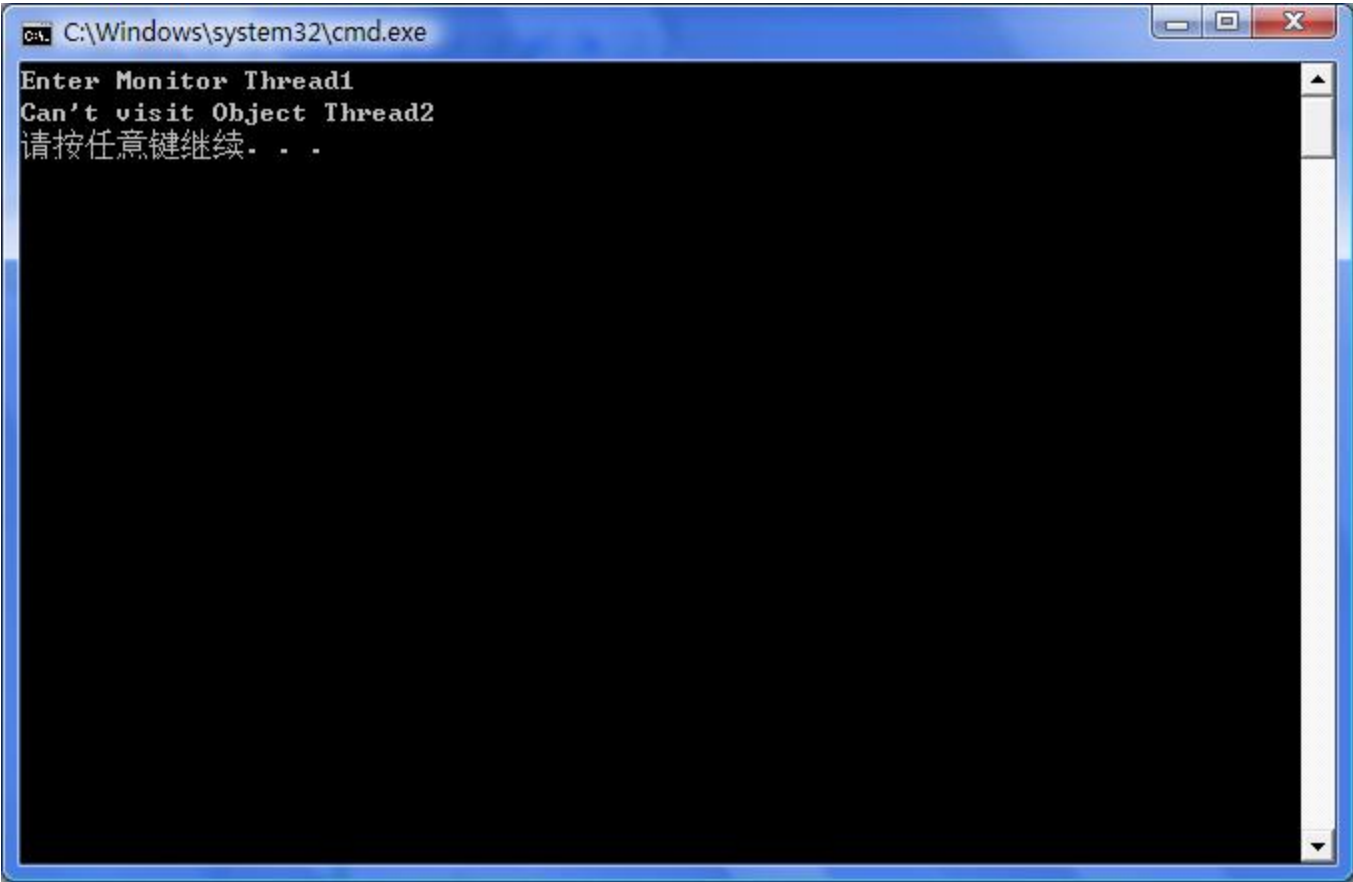
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
namespace MonitorApplication
{
    class Program
    {
        private static object m_monitorObject = new object();
        static void Main(string[] args)
        {
            Thread thread = new Thread(Do);
            thread.Name = "Thread1";
            Thread thread2 = new Thread(Do);
            thread2.Name = "Thread2";
            thread.Start();
            thread2.Start();
            thread.Join();
            thread2.Join();
        }

        static void Do()
        {
            if (!Monitor.TryEnter(m_monitorObject))
            {
                Console.WriteLine("Can't visit Object " + Thread.CurrentThread.Name);

                return;
            }
            try
            {
                Monitor.Enter(m_monitorObject);
                Console.WriteLine("Enter Monitor " + Thread.CurrentThread.Name);
                Thread.Sleep(5000);
            }
            finally
            {
                Monitor.Exit(m_monitorObject);
            }
        }
    }
}
```

```
    {  
        ...  
    }  
}
```

当线程1获取了m\_monitorObject对象独占权时,线程2尝试调用TryEnter(m\_monitorObject),此时会由于无法获取独占权而返回false,输出信息如下



可以看到线程2无法获取到m\_monitorObject的独占权,因此输出了一条错误信息.

Monitor类比lock类提供了一种更优秀的功能,考虑一下如下的场景

- 1.当你进入某家餐馆时,发现餐馆里坐满了客人,但是你又不想换地方,因此,你选择等待。
- 2.当餐馆内的服务员发现有空位置时,他通知前台的服务生,服务生给你安排了一个座位,于是你开始享受国际化的大餐。

使用**Monitor**类就可以实现如上的应用需求,在**Monitor**类中,提供了如下三个静态方法

```
Monitor.Pulse(Object o)
Monitor.PulseAll(Object o)
Monitor.Wait(Object o ) // 重载函数
```

当调用**Wait**方法时,线程释放资源的独占锁,并且阻塞在**wait**方法直到另外的线程获取资源的独占锁后,更新资源信息并调用**Pulse**方法后返回。

我们模拟上述代码如下

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
namespace MonitorApplication
{
    class Program
    {
        private static object m_isFull = new object();
        static void Main(string[] args)
        {
            Thread thread = new Thread(HaveLunch);
            thread.Name = "HaveLunchThread";
            Thread thread2 = new Thread(SeatChange);
            thread2.Name = "SeatChangeThread";
            thread.Start();
            System.Threading.Thread.Sleep(2000);
            thread2.Start();
            thread.Join();
            thread2.Join();
        }

        private static void HaveLunch()
        {
            lock (m_isFull)
            {
                Console.WriteLine("Wati for seta");
                Monitor.Wait(m_isFull);
                Console.WriteLine("Have a good lunch!");
            }
        }
    }
}
```

```
private static void SeatChange()  
{  
    lock (m_isFull)  
    {  
        Console.WriteLine("Seat was changed");  
        Monitor.Pulse(m_isFull);  
    }  
}
```

输出信息如下



```
Wati for seta  
Seat was changed  
Have a good lunch!  
请按任意键继续. . .
```

可见,使用**Monitor**,我们能实现一种唤醒式的机制,相信在实际应用中也有不少类似的场景。

## 5.使用System.Threading.Mutex(互斥体)类实现同步

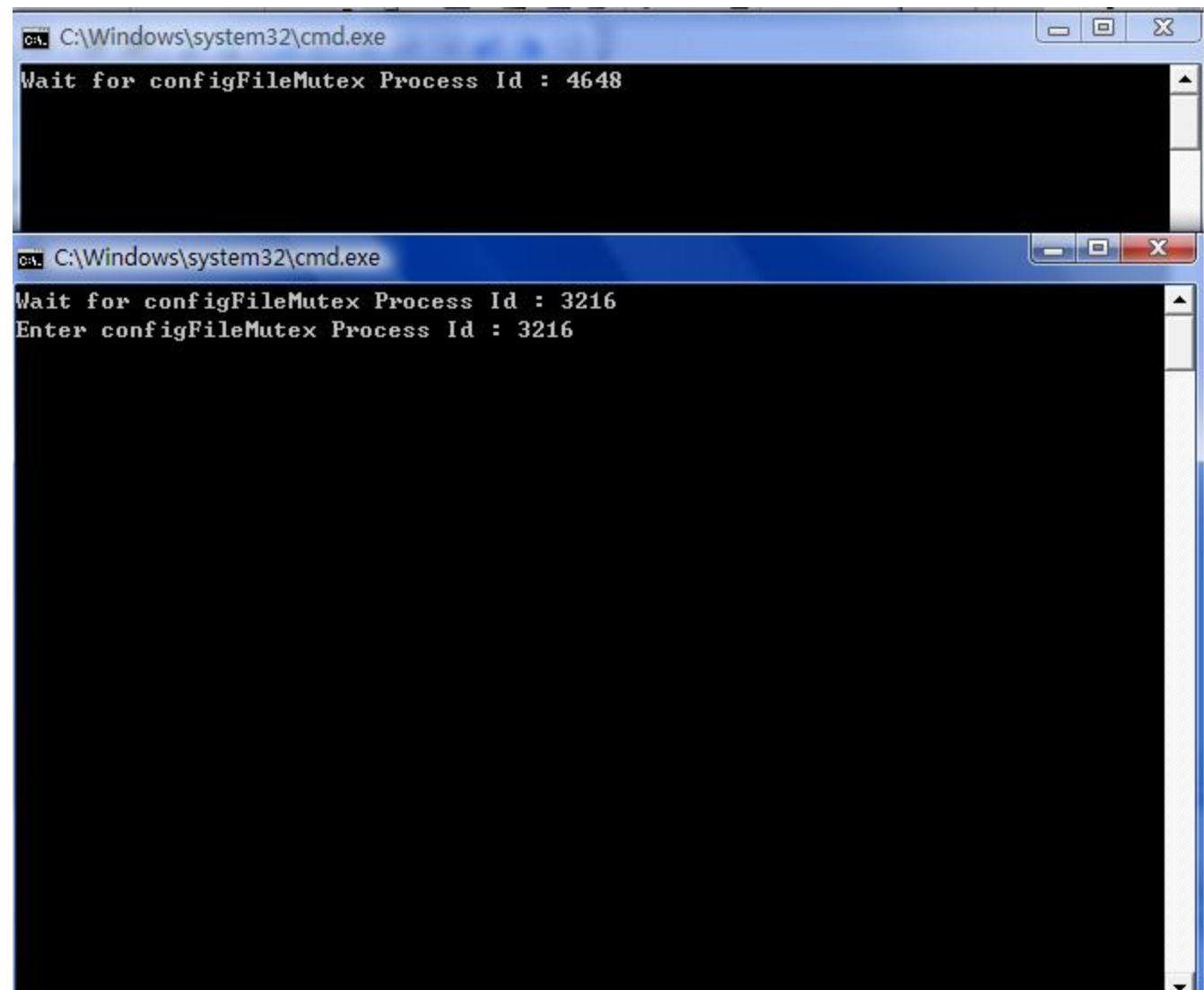
在使用上,**Mutex**与上述的**Monitor**比较接近,不过**Mutex**不具备**Wait**,**Pulse**,**PulseAll**的功能,因此,我们不能使用**Mutex**实现类似的唤醒的功能,不过**Mutex**有一个比较大的特点,**Mutex**是跨进程的,因此我们可以在同一台机器甚至远程的机器上的多个进程上使用同一个互斥体。

考虑如下的代码,代码通过获取一个称为**ConfigFileMutex**的互斥体,修改配置文件信息,写入一条数据,我们同时开启两个相同的进程进行测试

```
using System;  
using System.Collections.Generic;  
using System.Text;  
using System.Threading;  
using System.IO;  
using System.Diagnostics;  
namespace MonitorApplication  
{
```

```
class Program
{
    static void Main(string[] args)
    {
        Mutex configFileMutex = new Mutex(false, "configFileMutex");
        Console.WriteLine("Wait for configFileMutex Process Id : " + Process.GetCurrentProcess().Id);
        configFileMutex.WaitOne();
        Console.WriteLine("Enter configFileMutex Process Id : " + Process.GetCurrentProcess().Id);
        System.Threading.Thread.Sleep(10000);
        if (!File.Exists(@"..\config.txt"))
        {
            FileStream stream = File.Create(@"..\config.txt");
            StreamWriter writer = new StreamWriter(stream);
            writer.WriteLine("This is a Test!");
            writer.Close();
            stream.Close();
        }
        else
        {
            String[] lines = File.ReadAllLines(@"..\config.txt");
            for (int i = 0; i < lines.Length; i++)
                Console.WriteLine(lines[i]);
        }
        configFileMutex.ReleaseMutex();
        configFileMutex.Close();
    }
}
```

运行截图如下

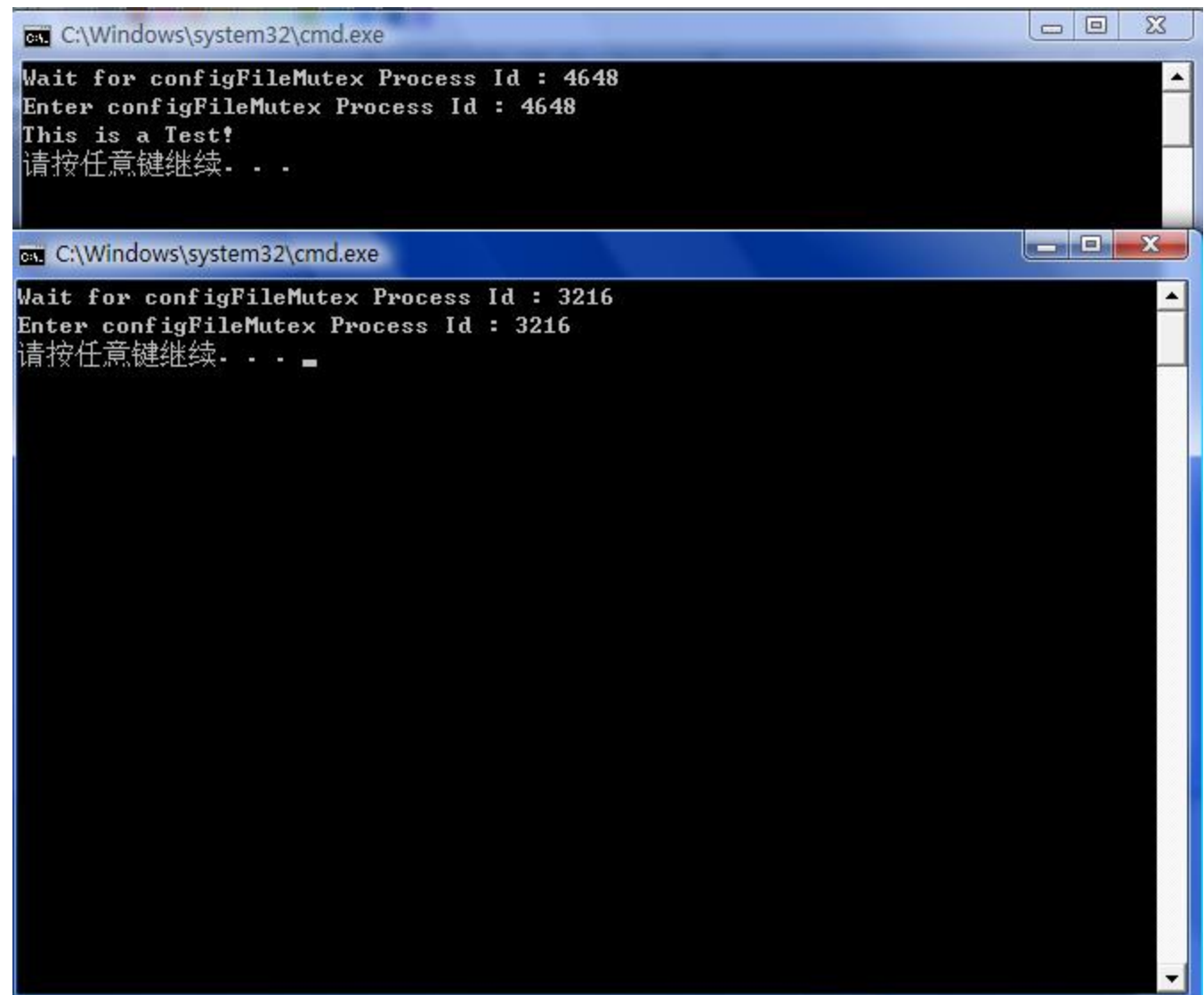


The image shows two overlapping Windows command prompt windows. The top window has a title bar that reads 'C:\Windows\system32\cmd.exe' and contains the text 'Wait for configFileMutex Process Id : 4648'. The bottom window also has a title bar that reads 'C:\Windows\system32\cmd.exe' and contains two lines of text: 'Wait for configFileMutex Process Id : 3216' followed by 'Enter configFileMutex Process Id : 3216'. Both windows have a black background with white text.

此时,PID 为 4628的进程正在等待PID 为 3216的进程释放configFileMutex的互斥体,因此它阻塞在WaitOne函数中,当PID为 3216的进程添加并写入了一条信息

至config.txt文件后,释放configFileMutex的独占权,PID为4628的进程获取configFileMutex的独占权,并从config.txt文件中读取输出PID3216进程写入的信息,截图如下





```
C:\Windows\system32\cmd.exe
Wait for configFileMutex Process Id : 4648
Enter configFileMutex Process Id : 4648
This is a Test!
请按任意键继续. . .

C:\Windows\system32\cmd.exe
Wait for configFileMutex Process Id : 3216
Enter configFileMutex Process Id : 3216
请按任意键继续. . .
```

可见使用**Mutex**进行同步,同步的互斥体是存在于多个进程间的。

## 6. 使用System.Threading.ReaderWriterLock类实现多用户读/单用户写的同步访问机制

在考虑资源访问的时候,惯性上我们会对资源实施**lock**机制,但是在某些情况下,我们仅仅需要读取资源的数据,而不是修改资源的数据,在这种情况下获取资源的独占权无疑会影响运行效率,因此**.Net**提供了一种机制,使用**ReaderWriterLock**进行资源访问时,如果在某一时刻资源并没有获取写的独占权,那么可以获得多个读的访问权,单个写入的独占权,如果某一时刻已经获取了写入的独占权,那么其它读取的访问权必须进行等待,参考以下代码

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.IO;
using System.Diagnostics;
namespace MonitorApplication
{
    class Program
    {
        private static ReaderWriterLock m_readerWriterLock = new ReaderWriterLock();

        private static int m_int = 0;
        static void Main(string[] args)
        {
            Thread readThread = new Thread(Read);
            readThread.Name = "ReadThread1";
            Thread readThread2 = new Thread(Read);
            readThread2.Name = "ReadThread2";
            Thread writeThread = new Thread(Writer);
            writeThread.Name = "WriterThread";
            readThread.Start();
            readThread2.Start();
            writeThread.Start();
            readThread.Join();
            readThread2.Join();
            writeThread.Join();
        }

        private static void Read()
        {
            while (true)
            {
                Console.WriteLine("ThreadName " + Thread.CurrentThread.Name + " AcquireReaderLock");

                m_readerWriterLock.AcquireReaderLock(10000);

                Console.WriteLine(String.Format("ThreadName : {0} m_int : {1}", Thread.CurrentThread.Name, m_int));
            }
        }
    }
}
```

```
        m_readerWriterLock.ReleaseReaderLock();
    }
}

private static void Writer()
{
    while (true)
    {
        Console.WriteLine("ThreadName " + Thread.CurrentThread.Name + " AcquireWriterLock");

        m_readerWriterLock.AcquireWriterLock(1000);

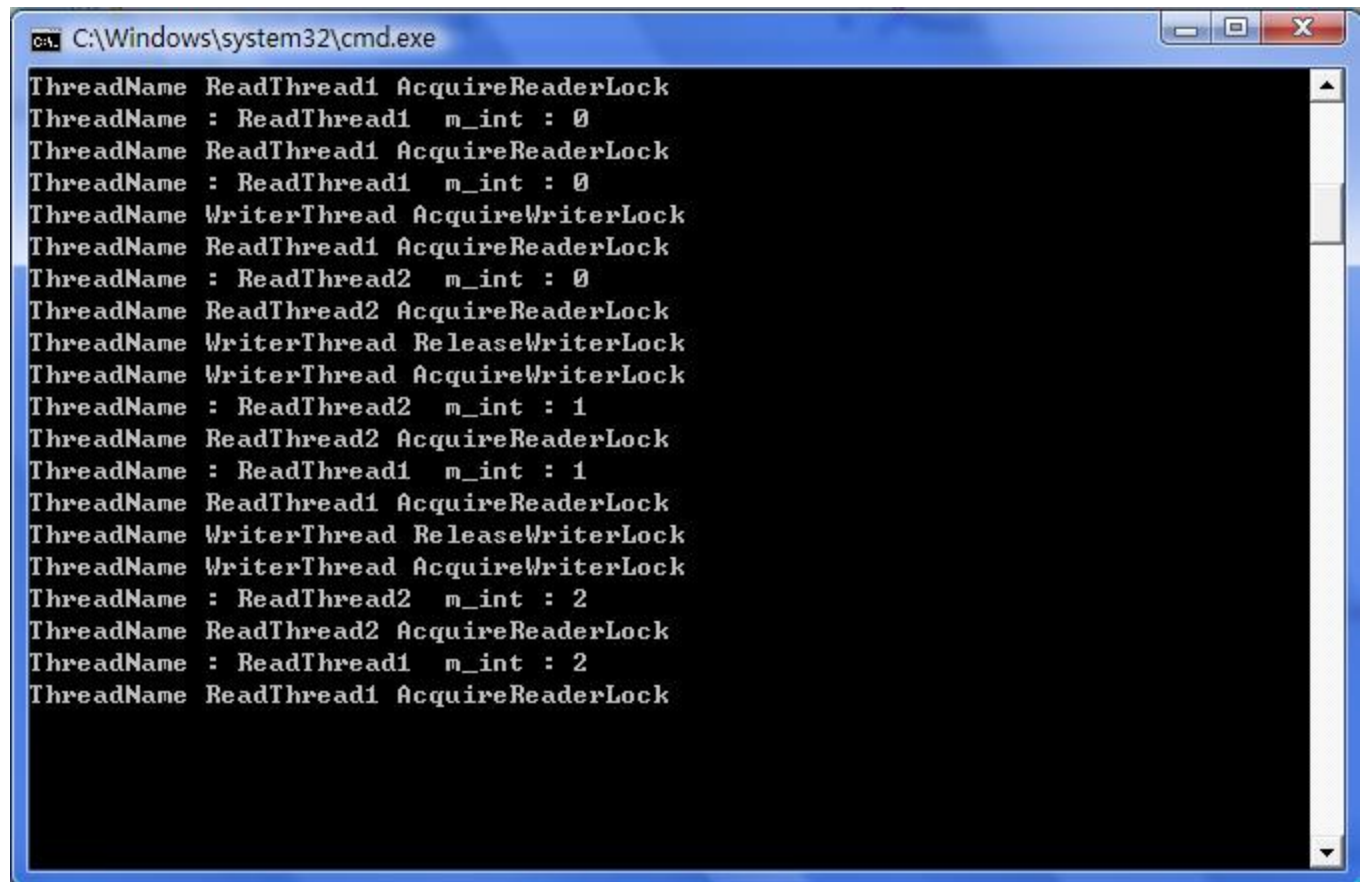
        Interlocked.Increment(ref m_int);

        Thread.Sleep(5000);

        m_readerWriterLock.ReleaseWriterLock();

        Console.WriteLine("ThreadName " + Thread.CurrentThread.Name + " ReleaseWriterLock");
    }
}
}
```

在程序中,我们启动两个线程获取m\_int的读取访问权,使用一个线程获取m\_int的写入独占权,执行代码后,输出如下

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a blue title bar and standard Windows window controls (minimize, maximize, close). The command prompt displays a series of log messages in a monospaced font. The messages are organized into pairs, where each pair represents a thread's action and its current state. The threads involved are ReadThread1, ReadThread2, and WriterThread. The actions include "AcquireReaderLock", "AcquireWriterLock", "ReleaseWriterLock", and "m\_int" updates. The states are indicated by "m\_int : 0" or "m\_int : 1" or "m\_int : 2". The sequence of messages shows ReadThread1 and ReadThread2 acquiring reader locks, WriterThread acquiring a writer lock, and then ReadThread1 and ReadThread2 acquiring reader locks again after the writer lock is released. The messages are as follows:  
ThreadName ReadThread1 AcquireReaderLock  
ThreadName : ReadThread1 m\_int : 0  
ThreadName ReadThread1 AcquireReaderLock  
ThreadName : ReadThread1 m\_int : 0  
ThreadName WriterThread AcquireWriterLock  
ThreadName ReadThread1 AcquireReaderLock  
ThreadName : ReadThread2 m\_int : 0  
ThreadName ReadThread2 AcquireReaderLock  
ThreadName WriterThread ReleaseWriterLock  
ThreadName WriterThread AcquireWriterLock  
ThreadName : ReadThread2 m\_int : 1  
ThreadName ReadThread2 AcquireReaderLock  
ThreadName : ReadThread1 m\_int : 1  
ThreadName ReadThread1 AcquireReaderLock  
ThreadName WriterThread ReleaseWriterLock  
ThreadName WriterThread AcquireWriterLock  
ThreadName : ReadThread2 m\_int : 2  
ThreadName ReadThread2 AcquireReaderLock  
ThreadName : ReadThread1 m\_int : 2  
ThreadName ReadThread1 AcquireReaderLock

可以看到,当WriterThread获取到写入独占权后,任何其它读取的线程都必须等待,直到WriterThread释放掉写入独占权后,才能获取到数据的访问权,  
应该注意的是,上述打印信息很明显显示出,可以多个线程同时获取数据的读取权,这从ReadThread1和ReadThread2的信息交互输出可以看出。

## 7.使用System.Runtime.Remoting.Contexts.SynchronizationAttribute对类对象进行同步控制

当我们确定某个类的实例在同一时刻只能被一个线程访问时,我们可以直接将类标识成Synchronization的,这样,CLR会自动对这个类实施同步机制,  
实际上,这里面涉及到同步域的概念,当类按如下设计时,我们可以确保类的实例无法被多个线程同时访问

- 1). 在类的声明中,添加System.Runtime.Remoting.Contexts.SynchronizationAttribute属性。
- 2). 继承至System.ContextBoundObject

需要注意的是,要实现上述机制,类必须继承至System.ContextBoundObject,换句话说,类必须是上下文绑定的。

一个示范类代码如下

```
[System.Runtime.Remoting.Contexts.Synchronization]
public class SynchronizedClass : System.ContextBoundObject
{
    [
    ]
}
```

还有AutoResetEvent,ManualReset,EventWaitHandle,Semaphore等可以实现资源的控制,不过它们更多是是基于一种事件唤醒的机制,如果有兴趣可以查阅MSDN相关的文档。

小弟我刚进入Net学习没多久,写得有什么地方不好,欢迎各位大侠多提意见,希望能与更多的人一起交流,学习

0 0

(请您对文章做出评价)

posted @ 2008-06-25 14:19 VincentWP 阅读(3200) 评论(11) 编辑 收藏 网摘 所属分类: C# 技术

发表评论

#1楼2008-06-25 15:51 | 钢钢 回复 引用 查看

不错，不知道我博客里的东东能不能对你有所帮助 ^\_^

C#多线程学习(一) 多线程的相关概念  
<http://www.cnblogs.com/xugang/archive/2008/04/06/1138856.html>

C#多线程学习(二) 如何操纵一个线程

C#多线程学习(三) 生产者和消费者

C# 多线程学习(四) 多线程的自动管理(线程池)

C# 多线程学习(五) 多线程的自动管理(定时器)

C# 多线程学习(六) 互斥对象

(以上都在一个分类中)

#2楼	[ 楼主 ]	2008-06-25 16:02   VincentWP	回复 引用 查看
谢谢钢钢,你的资料对我很有帮助,谢谢			
#3楼		2008-06-25 16:11   飄lá十蕩去	回复 引用 查看
写的不错。			
#4楼		2008-06-25 16:28   Angel Lucifer	回复 引用 查看
强烈不推荐使用 <b>ReaderWriterLock</b> 和 <b>SynchronizationAttribute</b> 。			
<b>ReaderWriterLock</b> 性能太低,可以考虑使用 <b>ReaderWriterLockSlim</b> (.NET 3.5新增)。			
<b>SynchronizationAttribute</b> 的毛病跟 <b>lock(this)</b> 和 <b>lock(typeof(TypeName))</b> 一样,容易导致死锁。			
<b>volatile</b> 要小心使用,按理也在不推荐之列。详情可以参考:			
<a href="#">并发数据结构:谈谈volatile变量</a>			
#5楼	[ 楼主 ]	2008-06-25 16:35   VincentWP	回复 引用 查看
谢谢 Angel Lucifer,我对C# 中性能的概念还很模糊,一定拜读一下			
#6楼		2008-06-25 17:07   HelloCode	回复 引用 查看
<b>lock</b> 关键字锁定的 <b>object</b> 对象需要声明为 <b>private static</b> 的才行。			

以前看到的一片微软的文章讲到如果是非静态的，就会产生多个object，锁住的将不是同一个object。

如果是非private,声明为Public之类的，则可能其他对象锁住这个object造成死锁。

#7楼2008-06-25 17:35 | Angel Lucifer

[回复](#) [引用](#) [查看](#)

@HelloCode  
private 没错，是否 static 还要看情况。

@VincentWP  
互相学习,-)

#8楼2008-06-25 17:45 | 鼠·神·泪.NET

[回复](#) [引用](#) [查看](#)

由于字符串的拘留池机制，建议不要lock字符串。

#9楼[ 楼主 ]2008-06-25 17:50 | VincentWP

[回复](#) [引用](#) [查看](#)

对,对于lock是否需要static,我认为在单例模式下,lock关键字锁定的Object对象似乎没有必要是static的,声明为private s  
人的感觉这个对象锁是全局锁,

但它又是私有的,意义就混淆了.

#10楼2008-06-25 23:02 | Steven Chen

[回复](#) [引用](#) [查看](#)

哦 lz和1F,2F，4F的资料都很棒哦，特别是4F。

在这里说一声，谢谢了

一个真正优秀的程序员应该关注于知识的分享

-----

#11楼2008-06-26 09:57 | Tony Zhou

[回复](#) [引用](#) [查看](#)

good

本市人员可享受**50-100%**政府补贴 合格颁发国家职业资格和微软双认证  
[www.zili.cn](http://www.zili.cn)

[刷新评论列表](#) [刷新页面](#) [返回首页](#)

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](#)

[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





China-pub 计算机图书网上专卖店! 6.5万品种 2-8折!

China-Pub 计算机绝版图书按需印刷服务

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:

我们到底该怎么学技术? 如何成为一个优秀的技术人员?

一个连英语都不会的.NET程序员能走多远?

博客园.NET频道, 专业.NET技术门户

一个连英语都不会的.NET程序员怎么提高?

程序员的网上家园

一个连英语都不会的C++程序员怎么提高?

ASP.NET MVC 专题, 从零开始学.NET技术