

朱燧

--书到读透处，酒于微醺时

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

公告

最近在读...



我的最新[book.csdn.net](#)

Good Bye Michael...

与我联系
[发短消息](#)

.Net线程问题解答

把遇到过的对.Net线程的一些问题和误解集中起来和大家分享,也希望大家能一起补充,热烈欢迎讨论

目录

基础篇

- 怎样创建一个线程
- 受托管的线程与 Windows线程
- 前台线程与后台线程
- 名为BeginXXX和EndXXX的方法是做什么用的
- 异步和多线程有什么关联

WinForm多线程编程篇

- 我的多线程WinForm程序老是抛出InvalidOperationException，怎么解决？
- Invoke，BeginInvoke干什么用的，内部是怎么实现的
- 每个线程都有消息队列吗？
- 为什么Winform不允许跨线程修改UI线程控件的值
- 有没有什么办法可以简化WinForm多线程的开发

线程池

- 线程池的作用是什么？
- 所有进程使用一个共享的线程池,还是每个进程使用独立的线程池？
- 为什么不要手动线程池设置最大值？
- .Net线程池有什么不足？

同步

- CLR怎样实现lock(obj)锁定？
- WaitHandle是什么,他和他的派生类怎么使用
- 什么是用双锁实现Singleton,为什么要这样做,为什么有人说双锁检验是不安全的
- 互斥对象（Mutex）、事件（Event）对象与lock语句的比较

搜索

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

- 博客园仿真足球交流小组

我参与的团队

- MVP(微软最有价值专家)团队(0/0)

我的标签

- 地震(1)
- 架构(1)
- 静态话(1)
- MSMQ(1)
- NLB(1)

随笔分类(86)

- .Net研究(12) (rss)
- Asp.Net(4) (rss)
- English(1) (rss)
- Python(4) (rss)
- VisualStudio(4) (rss)
- Web前台(5) (rss)
- 翻译(3) (rss)
- 管理(2) (rss)
- 排错于Windbg (rss)
- 企业服务(1) (rss)
- 书评(1) (rss)
- 数据库(6) (rss)
- 水煮java(1) (rss)
- 随想(4) (rss)
- 笑一笑(10) (rss)
- 性能(5) (rss)
- 转载收藏(23) (rss)

随笔档案(77)

- 2009年7月 (1)
- 2009年6月 (1)

什么时候需要锁定

- 只有共享资源才需要锁定
- 把锁定交给数据库
- 了解你的程序是怎么运行的
- 业务逻辑对事务和线程安全的要求
- 计算一下冲突的可能性
- 请多使用lock,少用Mutex

Web和IIS

- 应用程序池,WebApplication,和线程池之间有什么关系
- Web页面怎么调用异步WebService

基础篇

怎样创建一个线程

我只简单列举几种常用的方法,详细可参考.Net多线程总结(一)

一) 使用Thread类

```
ThreadStart threadStart=new ThreadStart(Calculate);//通过ThreadStart委托告诉子线程讲执行什么方法,这里执行一个计算圆周长的方法
Thread thread=new Thread(threadStart);
thread.Start(); //启动新线程

public void Calculate(){
    double Diameter=0.5;
    Console.WriteLine("The perimeter Of Circle with a Diameter of {0} is {1}"Diameter,Diameter*Math.PI);
}
```

二) 使用Delegate.BeginInvoke

```
delegate double CalculateMethod(double Diameter); //申明一个委托,表明需要在子线程上执行的方法的函数签名
static CalculateMethod calcMethod = new CalculateMethod(Calculate);//把委托和具体的方法关联起来
static void Main(string[] args)
{
    //此处开始异步执行,并且可以给出一个回调函数(如果不需要执行什么后续操作也可以不使用回调)
    calcMethod.BeginInvoke(5, new AsyncCallback(TaskFinished), null);
    Console.ReadLine();
}

//线程调用的函数,给出直径作为参数,计算周长
public static double Calculate(double Diameter)
{
}
```

2009年5月 (5)
2009年4月 (1)
2009年3月 (9)
2008年9月 (1)
2008年5月 (3)
2008年4月 (3)
2008年3月 (2)
2008年1月 (5)
2007年12月 (9)
2007年11月 (6)
2007年10月 (3)
2007年9月 (4)
2007年8月 (14)
2007年7月 (10)

文章分类(4)

水煮.net(3) (rss)
水煮java (rss)
水煮javascript (rss)
水煮web(1) (rss)
水煮架构 (rss)

朋友的博客

ocan的博客
蝈蝈俊.net

我的其他博客

我在CSDN的Blog

友情链接

爱百科-中文百科
爱百科:一个网友的作品,内容速度
可用性都还不错,他们的口号是--可
信赖的中文博客

积分与排名

积分 - 121480
排名 - 481

最新评论 XML

1. Re:VisualStudio2005技巧集
合--你真的会使用断点吗?
学习了
--波 象山
2. Re:.Net线程问题解答
太感谢楼主的辛苦劳动 拜谢
--Alifellod
3. Re:谈谈网站静态化
拜读大作,受益匪浅。 另我写过一
篇关于网站缓存方面的文章,可以
作为你的补充。
--lijun123
4. Re:网站跨站点单点登录

```
        return Diameter * Math.PI;
    }

//线程完成之后回调的函数
public static void TaskFinished(IAsyncResult result)
{
    double re = 0;
    re = calcMethod.EndInvoke(result);
    Console.WriteLine(re);
}
```

三) 使用ThreadPool.QueueuserworkItem

```
WaitCallback w = new WaitCallback(Calculate);
//下面启动四个线程,计算四个直径下的圆周长
ThreadPool.QueueUserWorkItem(w, 1.0);
ThreadPool.QueueUserWorkItem(w, 2.0);
ThreadPool.QueueUserWorkItem(w, 3.0);
ThreadPool.QueueUserWorkItem(w, 4.0);
public static void Calculate(double Diameter)
{
    return Diameter * Math.PI;
}
```

下面两条来自于<http://www.cnblogs.com/tonyman/archive/2007/09/13/891912.html>

受托管的线程与 Windows线程

必须要了解,执行.NET应用的线程实际上仍然是Windows线程。但是,当某个线程被CLR所知时,我们将它称为受托管的线程。具体来说,由受托管的代码创建出来的线程就是受托管的线程。如果一个线程由非托管的代码所创建,那么它就是非托管的线程。不过,一旦该线程执行了受托管的代码它就变成了受托管的线程。

一个受托管的线程和非托管的线程的区别在于,CLR将创建一个System.Threading.Thread类的实例来代表并操作前者。在内部实现中,CLR将一个包含了所有受托管线程的列表保存在一个叫做ThreadStore地方。

CLR确保每一个受托管的线程在任意时刻都在一个AppDomain中执行,但是这并不代表一个线程将永远处在一个AppDomain中,它可以随着时间的推移转到其他的AppDomain中。

从安全的角度来看,一个受托管的线程的主用户与底层的非托管线程中的Windows主用户是无关的。

前台线程与后台线程

启动了多个线程的程序在关闭的时候却出现了问题,如果程序退出的时候不关闭线程,那么线程就会一直的存在,但是大多启动的线程都是局部变量,不能一一的关闭,如果调用Thread.CurrentThread.Abort()方法关闭主线程的话,就会出现ThreadAbortException异常,因此这样不行。后来找到了这个办法: Thread.IsBackground 设置线程为后台线程。

msdn对前台线程和后台线程的解释: 托管线程或者是后台线程,或者是前台线程。后台线程不会使托管执行环境处于活动状态,除此

@xuefly 跨浏览器也可以吗? 请问你实现了没有

--郑州--飞猫

5. Re:Web中使用多线程来增强用户体验

没有解答我所需要的解答的问题。

。

--蜗牛身上的一只蚂蚁

阅读排行榜

- 1. web架构设计经验分享(11305)
- 2. .Net线程问题解答(10625)
- 3. .Net多线程总结(一)(8896)
- 4. .Net多线程总结(二)-BackgroundWorker(6255)
- 5. 分页实现方法的性能比较(6106)

评论排行榜

- 1. .Net线程问题解答(86)
- 2. VisualStudio2005技巧集合--你真的会使用断点吗?(44)
- 3. web架构设计经验分享(39)
- 4. 谈谈网站静态化 (37)
- 5. 【组图】地震前线归来--心中的震撼(32)

之外, 后台线程与前台线程是一样的。一旦所有前台线程在托管进程(其中 .exe 文件是托管程序集)中被停止, 系统将停止所有后台线程并关闭。通过设置 Thread.IsBackground 属性, 可以将一个线程指定为后台线程或前台线程。例如, 通过将

Thread.IsBackground 设置为 true, 就可以将线程指定为后台线程。同样, 通过将 IsBackground 设置为 false, 就可以将线程指定为前台线程。从非托管代码进入托管执行环境的所有线程都被标记为后台线程。通过创建并启动新的 Thread 对象而生成的所有线程都是前台线程。如果要创建希望用来侦听某些活动(如套接字连接)的前台线程, 则应将 Thread.IsBackground 设置为 true, 以便进程可以终止。

所以解决办法就是在主线程初始化的时候, 设置: Thread.CurrentThread.IsBackground = true;

这样, 主线程就是后台线程, 在关闭主程序的时候就会关闭主线程, 从而关闭所有线程。但是这样的话, 就会强制关闭所有正在执行的线程, 所以在关闭的时候要对线程工作的结果保存。

经常看到名为BeginXXX和EndXXX的方法, 他们是做什么用的

这是.net的一个异步方法名称规范

.Net在设计的时候为异步编程设计了一个异步编程模型 (APM), 这个模型不仅是使用.NET的开发人员使用, .Net内部也频繁用到, 比如所有的Stream就有BeginRead, EndRead, Socket, WebRequet, SqlCommand都运用到了这个模式, 一般来讲, 调用BegionXXX的时候, 一般会启动一个异步过程去执行一个操作, EndEnvoke可以接收这个异步操作的返回, 当然如果异步操作在EndEnvoke调用的时候还没有执行完成, EndInvoke会一直等待异步操作完成或者超时

.Net的异步编程模型 (APM) 一般包含BeginXXX, EndXXX, IAsyncResult这三个元素, BeginXXX方法都要返回一个IAsyncResult, 而EndXXX都需要接收一个IAsyncResult作为参数, 他们的函数签名模式如下

IAsyncResult BeginXXX(...);

<返回类型> EndXXX(IAsyncResult ar);

BeginXXX和EndXXX中的XXX, 一般都对应一个同步的方法, 比如FileStream的Read方法是一个同步方法, 相应的BeginRead(), EndRead()就是他的异步版本, HttpRequest有GetResponse来同步接收一个响应, 也提供了BeginGetResponse和EndGetResponse这个异步版本, 而IAsynResult是二者联系的纽带, 只有把BeginXXX所返回的IAsyncResult传给对应的EndXXX, EndXXX才知道需要去接收哪个BeginXXX发起的异步操作的返回值。

这个模式在实际使用时稍显繁琐, 虽然原则上我们可以随时调用EndInvoke来获得返回值, 并且可以同步多个线程, 但是大多数情况下当我们不需要同步很多线程的时候使用回调是更好的选择, 在这种情况下三个元素中的IAsynResult就显得多余, 我们一不需要用其中的线程完结标志来判断线程是否成功完成(回调的时候线程应该已经完成了), 二不需要他来传递数据, 因为数据可以写在任何变量里, 并且回调时应该已经填充, 所以可以看到微软在新的.Net Framework中已经加强了对回调事件的支持, 这总模型下, 典型的回调程序应该这样写

a.DoWork+=new SomeEventHandler(Caculate);
a.CallBack+=new SomeEventHandler(callback);
a.Run();

(注: 我上面讲的是普遍的用法, 然而BeginXXX, EndXXX仅仅是一种模式, 而对这个模式的实现完全取决于使用他的开发人员, 具体实现的时候你可以使用另外一个线程来实现异步, 也可能使用硬件的支持来实现异步, 甚至可能根本和异步没有关系(尽管几乎没有人会这样做) -----比如直接在Beginxxx里直接输出一个"Helloworld", 如果是这种极端的情况, 那么上面说的一切都是废话, 所以上面的探讨并不涉及内部实现, 只是告诉大家微软的模式, 和框架中对这个模式的经典实现)

异步和多线程有什么关联

有一句话总结的很好：多线程是实现异步的一种手段和工具

我们通常把多线程和异步等同起来，实际是一种误解，在实际实现的时候，异步有许多种实现方法，我们可以用进程来做异步，或者使用纤程，或者硬件的一些特性，比如在实现异步IO的时候,可以有下面两个方案：

- 1) 可以通过初始化一个子线程，然后在子线程里进行IO，而让主线程顺利往下执行，当子线程执行完毕就回调
- 2) 也可以根本不使用新线程，而使用硬件的支持（现在许多硬件都有自己的处理器），来实现完全的异步，这是我们只需要将IO请求告知硬件驱动程序，然后迅速返回，然后等着硬件IO就绪通知我们就可以了

实际上DotNet Framework里面就有这样的例子，当我们使用文件流的时候，如果制定文件流属性为同步，则使用BeginRead进行读取时，就是用一个子线程来调用同步的Read方法，而如果指定其为异步，则同样操作时就使用了需要硬件和操作系统支持的所谓IOCP的机制

WinForm多线程编程篇

我的多线程WinForm程序老是抛出InvalidOperationException，怎么解决？

在WinForm中使用多线程时，常常遇到一个问题，当在子线程（非UI线程）中修改一个控件的值：比如修改进度条进度，时会抛出如下错误

Cross-thread operation not valid: Control 'XXX' accessed from a thread other than the thread it was created on.

在VS2005或者更高版本中，只要不是在控件的创建线程（一般就是指UI主线程）上访问控件的属性就会抛出这个错误，解决方法就是利用控件提供的Invoke和BeginInvoke把调用封送回UI线程，也就是让控件属性修改在UI线程上执行，下面列出会报错的代码和他的修改版本

```
ThreadStart threadStart=new ThreadStart(Calculate);//通过ThreadStart委托告诉子线程讲执行什么方法
Thread thread=new Thread(threadStart);
thread.Start();
public void Calculate(){
    double Diameter=0.5;
    double result=Diameter*Math.PI;
    CalcFinished(result);//计算完成需要在一个文本框里显示
}
public void CalcFinished(double result){
    this.TextBox1.Text=result.ToString();//会抛出错误
```



```
    }
```

上面加粗的地方在debug的时候会报错，最直接的修改方法是修改Calculate这个方法如下

```
delegate void changeText(double result);

public void Calculate(){
    double Diameter=0.5;
    double result=Diameter*Math.PI;
    this.BeginInvoke(new changeText(CalcFinished),t.Result); //计算完成需要在一个文本框里显示
}
```

这样就ok了，但是最漂亮的方法是不去修改Calculate，而去修改CalcFinished这个方法，因为程序里调用这个方法的地方可能很多，由于加了是否需要封送的判断，这样修改还能提高非跨线程调用时的性能

```
delegate void changeText(double result);

public void CalcFinished(double result){
    if(this.InvokeRequired){
        this.BeginInvoke(new changeText(CalcFinished),t.Result);
    }
    else{
        this.TextBox1.Text=result.ToString();
    }
}
```

上面的做法用到了Control的一个属性InvokeRequired（这个属性是可以在其他线程里访问的），这个属性表明调用是否来自另非UI线程，如果是，则使用BeginInvoke来调用这个函数，否则就直接调用，省去线程封送的过程

Invoke，BeginInvoke干什么用的，内部是怎么实现的？

这两个方法主要是让给出的方法在控件创建的线程上执行

Invoke使用了Win32API的SendMessage，

UnsafeNativeMethods.PostMessage(new HandleRef(this, this.Handle), threadCallbackMessage, IntPtr.Zero, IntPtr.Zero);

BeginInvoke使用了Win32API的PostMessage

UnsafeNativeMethods.PostMessage(new HandleRef(this, this.Handle), threadCallbackMessage, IntPtr.Zero, IntPtr.Zero);

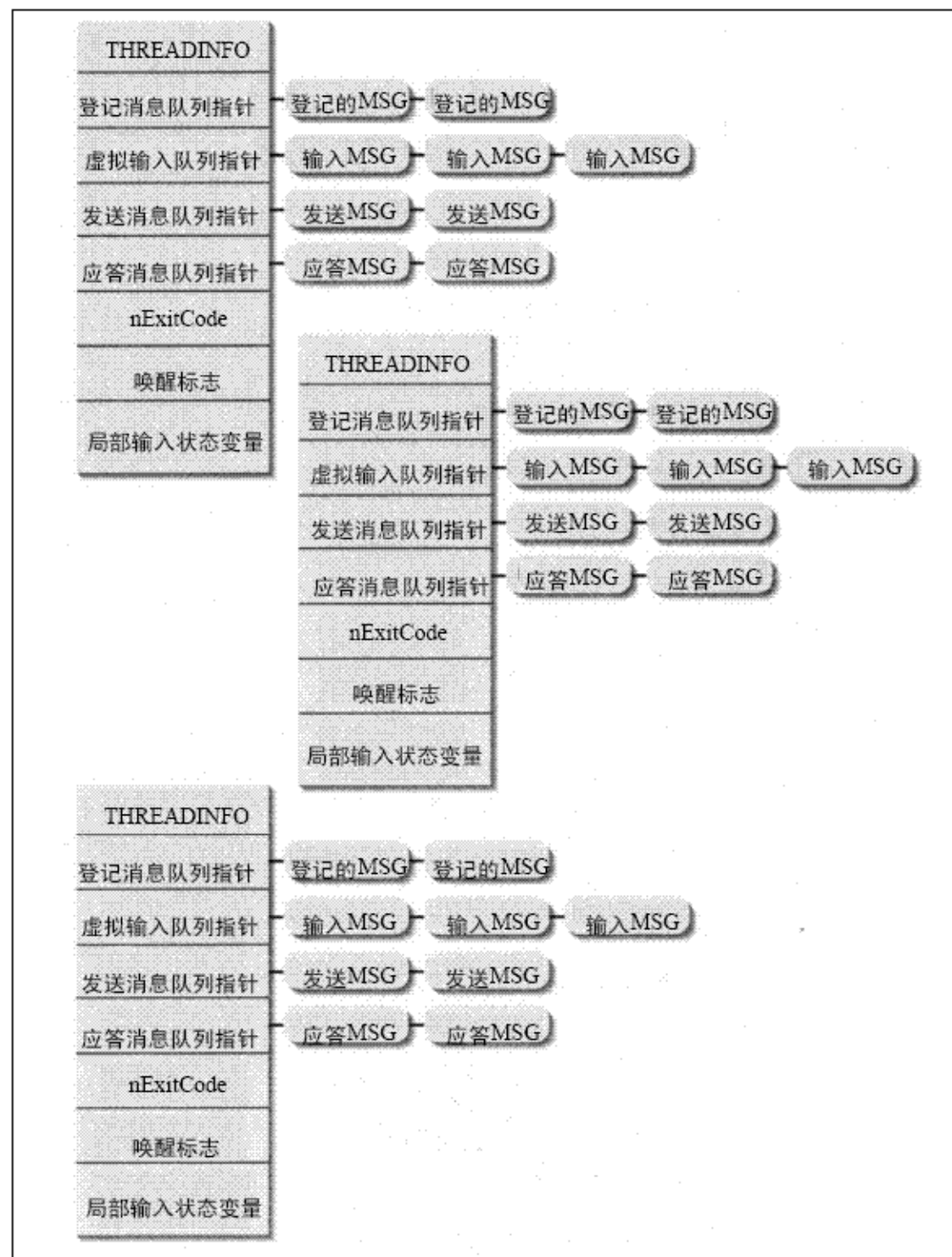
这两个方法向UI线程的消息队列中放入一个消息，当UI线程处理这个消息时，就会在自己的上下文中执行传入的方法，换句话说凡是使用BeginInvoke和Invoke调用的线程都是在UI主线程中执行的，所以如果这些方法里涉及一些静态变量，不用考虑加锁的问题

每个线程都有消息队列吗?

不是,只有创建了窗体对象的线程才会有消息队列(下面给出<Windows 核心编程>关于这一段的描述)

当一个线程第一次被建立时,系统假定线程不会被用于任何与用户相关的任务。这样可以减少线程对系统资源的要求。但是,一旦这个线程调用一个与图形用户界面有关的函数(例如检查它的消息队列或建立一个窗口),系统就会为该线程分配一些另外的资源,以便它能够执行与用户界面有关的任务。特别是,系统分配一个`THREADINFO`结构,并将这个数据结构与线程联系起来。

这个`THREADINFO`结构包含一组成员变量,利用这组成员,线程可以认为它是在自己独占的环境中运行。`THREADINFO`是一个内部的、未公开的数据结构,用来指定线程的登记消息队列(*posted-message queue*)、发送消息队列(*send-message queue*)、应答消息队列(*reply-message queue*)、虚拟输入队列(*virtualized-input queue*)、唤醒标志(*wake flag*)、以及用来描述线程局部输入状态的若干变量。图2 6 - 1描述了`THREADINFO`结构和与之相联系三个线程。



为什么**Winform**不允许跨线程修改**UI**线程控件的值

在**vs2003**下,使用子线程调用**ui**线程创建的控件的属性是不会有问题的,但是编译的时候会出现警告,但是**vs2005**及以上版本就会有这样的问题,下面是**msdn**上的描述

"当您在 **Visual Studio** 调试器中运行代码时, 如果您从一个线程访问某个 **UI** 元素, 而该线程不是创建该 **UI** 元素时所在的线程, 则会引发 **InvalidOperationException**。调试器引发该异常以警告您存在危险的编程操作。**UI** 元素不是线程安全的, 所以只应在创建它们的线程上进行访问"

从上面可以看出,这个异常实际是**debugger**耍的花招,也就是说,如果你直接运行程序的**exe**文件,或者利用运行而不调试(**Ctrl+F5**)来运行你的程序,是不会抛出这样的异常的.大概**ms**发现**v2003**的警告对广大开发者不起作用,所以用了一个比较狠一点的方法.

不过问题依然存在:既然这样设计的原因主要是因为控件的值非线程安全,那么**DotNet framework**中非线程安全的类千千万万,为什么偏偏跨线程修改**Control**的属性会有这样严格的限制策略呢?

这个问题我还回答不好,希望博友们能够予以补充

有没有什么办法可以简化**WinForm**多线程的开发

使用**backgroundworker**,使用这个组建可以避免回调时的**Invoke**和**BeginInvoke**,并且提供了许多丰富的方法和事件

参见[.Net多线程总结\(二\)-BackgroundWorker](#),我在这里不再赘诉

线程池

线程池的作用是什么

作用是减小线程创建和销毁的开销

创建线程涉及用户模式和内核模式的切换,内存分配,**dll**通知等一系列过程,线程销毁的步骤也是开销很大的,所以如果应用程序使用了完一个线程,我们能把线程暂时存放起来,以备下次使用,就可以减小这些开销

所有进程使用一个共享的线程池,还是每个进程使用独立的线程池?

每个进程都有一个线程池,一个**Process**中只能有一个实例,它在各个应用程序域(**AppDomain**)是共享的, **.Net2.0** 中默认线程池的大小为工作线程**25**个,**IO**线程**1000**个,有一个比较普遍的误解是线程池中会有**1000**个线程等着你去取,其实不然, **ThreadPool**仅仅保留相当少的线程,保留的线程可以用**SetMinThread**这个方法来自设置,当程序的某个地方需要创建一个线程来完成工作时,而线程池中又没有空闲线程时,线程池就会负责创建这个线程,并且在调用完毕后,不会立刻销毁,而是把他放在池子里,预备下次使用,同时如果线程超过一定时间没有被使用,线程池将会回收线程,所以线程池里存在的线程数实际是个动态的过程

为什么不要手动线程池设置最大值?

当我首次看到线程池的时候,脑袋里的第一个念头就是给他设定一个最大值,然而当我们查看ThreadPool的SetMaxThreads文档时往往会看到一条警告:不要手动更改线程池的大小,这是为什么呢?

其实无论FileStream的异步读写,异步发送接受Web请求,甚至使用delegate的beginInvoke都会默认调用 ThreadPool,也就是说不仅你的代码可能使用到线程池,框架内部也可能使用到,更改的后果影响就非常大,特别在iis中,一个应用程序池中的所有WebApplication会共享一个线程池,对最大值的设定会带来很多意想不到的麻烦

线程池的线程为何要分类?

线程池有一个方法可以让我们看到线程池中可用的线程数量:GetAvaliableThread(out workerThreadCount,out iocompletedThreadCount),对于我来说,第一次看到这个函数的参数时十分困惑,因为我期望这个函数直接返回一个整形,表明还剩多少线程,这个函数居然一次返回了两个变量.

原来线程池里的线程按照公用被分成了两大类:工作线程和IO线程,或者IO完成线程,前者用于执行普通的操作,后者专用于异步IO,比如文件和网络请求,注意,分类并不说明两种线程本身有差别,线程就是线程,是一种执行单元,从本质上来讲都是一样的,线程池这样分类,举例来说,就好像某施工工地现在有1000把铁锹,规定其中25把给后勤部门用,其他都给施工部门,施工部门需要大量使用铁锹来挖地基(例子土了点,不过说明问题还是有效的),后勤部门用铁锹也就是铲铲雪,铲铲垃圾,给工人师傅修修临时住房,所以用量不大,显然两个部门的铁锹本身没有区别,但是这样的划分就为管理两个部门的铁锹提供了方便

线程池中两种线程分别在什么情况下被使用,二者工作原理有什么不同?

下面这个例子直接说明了二者的区别,我们用一个流读出一个很大的文件(大一点操作的时间长,便于观察),然后用另一个输出流把所读出的文件的一部分写到磁盘上

我们用两种方法创建输出流,分别是

创建了一个异步的流(注意构造函数最后那个true)

```
FileStream outputfs=new FileStream(writepath, FileMode.Create, FileAccess.Write, FileShare.None,256,true);
```

创建了一个同步的流

```
FileStream outputfs = File.OpenWrite(writepath);
```

然后在写文件期间查看线程池的状况

```
string readpath = "e:\\RHEL4-U4-i386-AS-disc1.iso";
string writepath = "e:\\kakakak.iso";
byte[] buffer = new byte[9000000];

//FileStream outputfs=new FileStream(writepath, FileMode.Create, FileAccess.Write, FileShare.None,256,true);
//Console.WriteLine("异步流");
//创建了一个同步的流

FileStream outputfs = File.OpenWrite(writepath);
```

```
Console.WriteLine("同步流");

//然后在写文件期间查看线程池的状况

ShowThreadDetail("初始状态");

FileStream fs = File.OpenRead(readpath);

fs.BeginRead(buffer, 0, 90000000, delegate(IAsyncResult o)
{
    outputfs.BeginWrite(buffer, 0, buffer.Length,
        delegate(IAsyncResult o1)
        {
            Thread.Sleep(1000);

            ShowThreadDetail("BeginWrite的回调线程");

        }, null);

    Thread.Sleep(500); //this is important cause without this, this Thread and the one used for BeginRead May seem to be same one
},
    null);

Console.ReadLine();
```

```
public static void ShowThreadDetail(string caller)
{
    int IO;
    int Worker;
    ThreadPool.GetAvailableThreads(out Worker, out IO);
    Console.WriteLine("Worker: {0}; IO: {1}", Worker, IO);
}
```

输出结果

异步流

Worker: 500; IO: 1000

Worker: 500; IO: 999

同步流

Worker: 500; IO: 1000

Worker: 499; IO: 1000

这两个构造函数创建的流都可以使用BeginWrite来异步写数据,但是二者行为不同,当使用同步的流进行异步写时,通过回调的输出我们可以看到,他使用的是工作线程,而非IO线程,而异步流使用了IO线程而非工作线程

其实当没有制定异步属性的时候,.Net实现异步IO是用一个子线程调用fs的同步Write方法来实现的,这时这个子线程会一直阻塞直到调用完成.这个子线程其实就是线程池的一个工作线程,所以我们可以看到,同步流的异步写回调中输出的工作线程数少了一,而使用异步流,在进行异步写时,采用了 IOCP方法,简单说来,就是当BeginWrite执行时,把信息传给硬件驱动程序,然后立即往下执行(注意这里没有额外的线程),而当硬件准备就绪,就会通知线程池,使用一个IO线程来读取

.Net线程池有什么不足

没有提供方法控制加入线程池的线程:一旦加入线程池,我们没有办法挂起,终止这些线程,唯一可以做的就是等他自己执行

- 1)不能为线程设置优先级
- 2)一个Process中只能有一个实例,它在各个AppDomain是共享的。ThreadPool只提供了静态方法,不仅我们自己添加进去的WorkItem使用这个Pool,而且.net framework中那些BeginXXX、EndXXX之类的方法都会使用此Pool。
- 3)所支持的Callback不能有返回值。WaitCallback只能带一个object类型的参数,没有任何返回值。
- 4)不适合用在长期执行某任务的场合。我们常常需要做一个Service来提供不间断的服务(除非服务器down掉),但是使用ThreadPool并不合适。

下面是另外一个网友总结的什么不需要使用线程池,我觉得挺好,引用下来

如果您需要使一个任务具有特定的优先级。

如果您具有可能会长时间运行(并因此阻塞其他任务)的任务。

如果您需要将线程放置到单线程单元中(所有 ThreadPool 线程均处于多线程单元中)。


如果您需要与该线程关联的稳定标识。例如,您应使用一个专用线程来中止该线程、将其挂起或按名称发现它。

锁定与同步

CLR怎样实现lock(obj)锁定?

从原理上讲,lock和Synchronized Attribute都是用Monitor.Enter实现的,比如如下代码

```
object lockobj=new object();
lock(obj){
    //do things ...
}
```

在编译时,会被编译为类似

```
try{
  Monitor.Enter(obj){
    //do things ...
  }
}
catch{ ... }
finally{
  Monitor.Exit(obj);
}
```

而[**MethodImpl(MethodImplOptions.Synchronized)**]标记为同步的方法会在编译时被lock(this)语句所环绕
所以我们只简单探讨Monitor.Enter的实现

(注:DotNet并非使用Win32API的CriticalSection来实现Monitor.Enter, 不过他为托管对象提供了一个类似的结构叫做Syncblk)

每个对象实例头部都有一个指针,这个指针指向的结构,包含了对对象的锁定信息,当第一次使用Monitor.Enter(obj)时,这个obj对象的锁定结构就会被初时化,第二次调用Monitor.Enter时,会检验这个object的锁定结构,如果锁没有被释放,则调用会阻塞



WaitHandle是什么,他和他的派生类怎么使用

WaitHandle是Mutex, Semaphore, EventWaitHandler, AutoResetEvent, ManualResetEvent共同的祖先, 他们包装了用于同步的内核对象, 也就是说这些内核对象的托管版本。

Mutex: 类似于一个接力棒,拿到接力棒的线程才可以开始跑,当然接力棒一次只属于一个线程(**Thread Affinity**),如果这个线程不释放接力棒(**Mutex.ReleaseMutex**),那么没办法,其他所有需要接力棒运行的线程都知道能等着看热闹

Semaphore: 类似于一个小桶,里面装了几个小球,凡是拿到小球就可以跑,比如指定小桶里最初有四个小球,那么开始的四个线程就可以直接拿着自己的小球开跑,但是第五个线程一看,小球被拿光了,就只好乖乖的等着有谁放一个小球到小桶里(**Semaphore.Release**),他才能跑,但是这里的游戏规则比较特殊,我们可以随意向小桶里放入小球,也就是说我可以拿走一个小球,放回去俩,甚至一个都不拿,放回去5个,这样就有五个线程可以拿着这些小球运行了,我们可以规定小桶里有开始有几个小球(构造函数的第一个参数),也可以规定最多不能超过多少小球(构造函数的第二个参数)

ManualResetEvent, AutoResetEvent可以参考<http://www.cnblogs.com/uubox/archive/2007/12/18/1003953.html>

什么是用双锁实现**Singleton**,为什么要这样做,双锁检验是不安全的吗?

使用双锁检验技巧来实现单件,来自于**Java**社区

```
public static MySingleton Instance{
    get{
        if(_instance!=null){
            lock(_instance){
                if(s_value==null){
                    _instance= new MySingleton();
                }
            }
        }
    }
}
```

这样做其实是为了提高效率,比起

```
public static MySingleton Instance{

    get{

        lock(_instance){

            if(s_value==null){

                _instance= new MySingleton();

            }

        }

    }

}
```

前一种方法在**instance**创建的时候不需要用**lock**同步,从而增进了效率

在**java**中这种技巧被证明是不安全的详细见<http://www.cs.umd.edu/~pugh/java/memoryModel/>

但是在**.Net**下,这样的技巧是成立的,因为**.Net**使用了改进的内存模型

并且在**.Net**下,我们可以使用**LazyInit**来实现单件


```
private static readonly _instance=new MySingleton()  
  
public static MySingleton Instance{  
  
get{return _instance}  
  
}
```

当第一此使用_**instance**时,CLR会生成这个对象,以后再访问这个字段,将会直接返回

互斥对象（**Mutex**）,信号量(**Semaphore**),事件（**Event**）对象与**lock**语句的比较

首先这里所谓的事件对象不是**System.Event**，而是一种用于同步的内核机制

互斥对象和事件对象属于内核对象，利用内核对象进行线程同步，线程必须要在用户模式和内核模式间切换，所以一般效率很低，但利用互斥对象和事件对象这样的内核对象，可以在多个进程中的各个线程间进行同步。

lock或者**Monitor**是.net用一个特殊结构实现的,不涉及模式切换,也就是说工作在用户方式下，同步速度较快,但是不能跨进程同步

什么时候需要锁定?

刚刚接触锁定的程序员往往觉得这个世界非常的危险,每个静态变量似乎都有可能产生竞争

首先锁定是解决竞争条件的,也就是多个线程同时访问某个资源,造成意想不到的结果,比如,最简单的情况,一个计数器,如果两个线程同时加一,后果就是损失了一个计数,但是频繁的锁定又可能带来性能上的消耗,还有最可怕的情况,死锁

到底什么情况下我们需要使用锁,什么情况下不用呢?

只有共享资源才需要锁定

首先,只有可以被多线程访问的共享资源才需要考虑锁定,比如静态变量,再比如某些缓存中的值,属于线程内部的变量不需要锁定

把锁定交给数据库

数据库除了存储数据之外,还有一个重要的用途就是同步,数据库本身用了一套复杂的机制来保证数据的可靠和一致性,这就为我们节省了很多的精力.保证了数据源头上的同步,我们多数的精力就可以集中在缓存等其他一些资源的同步访问上了

了解你的程序是怎么运行的

实际上在**web**开发中大多数逻辑都是在单个线程中展开的,无论**asp.net**还是**php**,一个请求都会在一个单独的线程中处理,其中的大部分变量都是属于这个线程的,根本没有必要考虑锁定,当然对于**asp.net**中的**application**对象中的数据,我们就要小心一些了

WinForm中凡是使用**BeginInvoke**和**Invoke**调用的方法也都不需要考虑同步,因为这用这两个方法调用的方法会在**UI**线程中执行,因此实际是同步的,所以如果调用的方法中存在某些静态变量,不需要考虑锁定

业务逻辑对事务和线程安全的要求

这条是最根本的东西,开发完全线程安全的程序是件很费时费力的事情,在电子商务等涉及金融系统的案例中,许多逻辑都必须严格的线程安全,所以我们不得不牺牲一些性能,和很多的开发时间来做这方面的工作,而一般的应用中,许多情况下虽然程序有竞争的危险,我们还是可以不使用锁定,比如有的时候计数器少一多一,对结果无伤大雅的情况下,我们就可以不用去管他

计算一下冲突的可能性

我以前曾经谈到过,架构不要过设计,其实在这里也一样,假如你的全局缓存里的某个值每天只有几百或者几千个访问,并且访问时间很短,并且分布均匀(实际上这是大多数的情况),那么冲突的可能性就非常的少,也许每500天才会出现一次或者更长,从7*24小时安全服务的角度来看,也完全符合要求,那么你还会为这样万分之一的可能性花80%的精力去设计吗?

请多使用**lock**,少用**Mutex**

如果你一定要使用锁定,请尽量不要使用内核模块的锁定机制,比如.net的**Mutex**,**Semaphore**,**AutoResetEvent**,**ManuResetEvent**,使用这样的机制涉及到了系统在用户模式和内核模式间的切换,所以性能差很多,但是他们的优点是可以跨进程同步线程,所以应该清楚的了解到他们的不同和适用范围

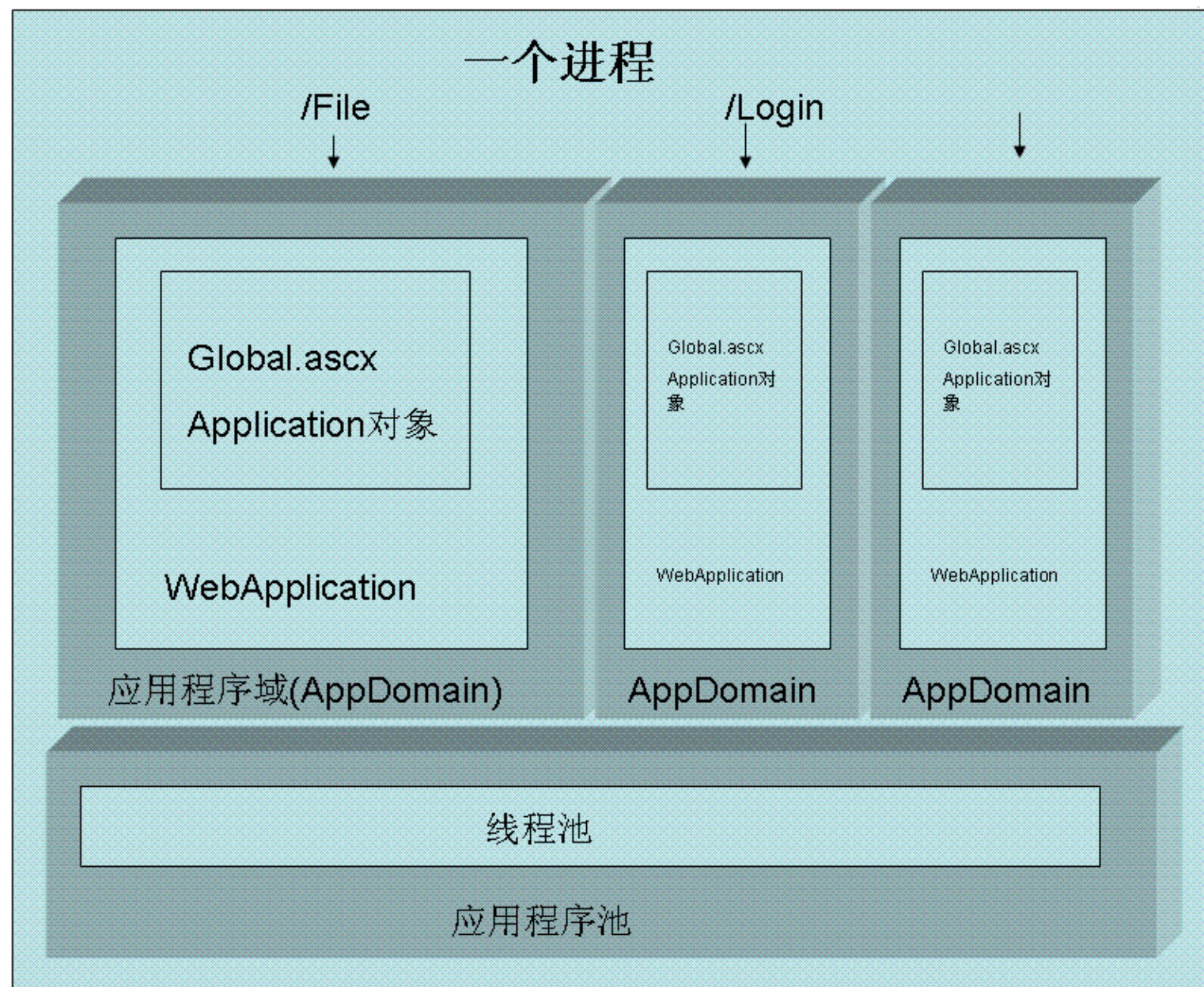
Web和IIS

应用程序池,**WebApplication**,和线程池之间有什么关系

一个应用程序池是一个独立的进程,拥有一个线程池,应用程序池中可以有多个**WebApplication**,每个运行在一个单独的**AppDomain**中,这些**WebApplication**公用一个线程池

不同的**AppDomain**保证了每个**WebApplication**的静态变量不会互相干扰,不同的应用程序池保证了一个网站瘫痪,其他不同进程中的站点还能正常运行

下图说明了他们的关系



Web页面怎么调用异步WebService

把Page的Async属性设置为true,就可以调用异步的方法,但是这样调用的效果可能并不如我们的相像,请参考[Web中使用多线程来增强用户体验](#)

推荐文章

<http://www.cnblogs.com/uubox/archive/2007/12/18/1003953.html>(内核对象同步,讲的很通俗易懂ManuResetEvent,AutoResetEvent)

<http://alang79.blogdriver.com/alang79/456761.html>

A low-level Look at the ASP.NET Architecture

参考资料

<Windows 核心编程>这本书里对内核对象的描述比较详尽
<.Net框架程序设计>和上面一本一样也是大牛Jeffery Richard的作品

朱燧的技术博客,转载请注明出处

<http://yizhu2000.cnblogs.com>

<http://blog.csdn.net/yizhu2000>

 订阅到九点

 订阅到九点

 推荐到九点

0浜烘悝鑽

1

0

(请您对文章做出评价)

posted on 2008-01-03 15:48 朱燧:-) 阅读(10625) 评论(86) 编辑 收藏 网摘 所属分类: .Net研究

评论

#1楼 2008-01-03 16:04 鞠强

up !!!
[回复](#) [引用](#) [查看](#)

#2楼 2008-01-03 16:17 aspnetx

不错,最近正好用到.
[回复](#) [引用](#) [查看](#)

#3楼 2008-01-03 16:46 craboYang

牛! 受教了!

更多示例讲解更好.
[回复](#) [引用](#) [查看](#)

#4楼 2008-01-03 16:50 stonezhu

这样的总结不错,UP
[回复](#) [引用](#) [查看](#)

#5楼[楼主] 2008-01-03 16:52 二毛五

楼上说的对,本来我也想这样,每个问题,配段程序,不过原谅我偷个懒吧,每写一个新的示例我都会想到一些新的问题,内容越来越多,真要一个个问题讲清楚,也许明年才能发出来和大家分享了:p
[回复](#) [引用](#) [查看](#)

#6楼 2008-01-03 17:04 Clark Zheng

潜力贴, 先回后看
[回复](#) [引用](#) [查看](#)

#7楼 2008-01-03 17:07 Linxi[未注册用户]

谁敢说lz是二毛五? 简直是二亿五。总结得这么超值得知识。
[回复](#) [引用](#)

#8楼 2008-01-03 17:25 隐姓埋名

收藏之~
谢谢~
[回复](#) [引用](#) [查看](#)

#9楼 2008-01-03 17:25 1-2-3

收藏
[回复](#) [引用](#) [查看](#)

#10楼 2008-01-03 17:27 江南白衣

不错
[回复](#) [引用](#) [查看](#)

#11楼 2008-01-03 17:36 郝婧

呵呵 今天我去听JAVA的课 就是讲到线程的课,我是学.NET 可是教材上没有线程的知识,就上你这来学学吧
[回复](#) [引用](#) [查看](#)

#12楼 2008-01-03 17:37 egmkang

这个文章很好.
[回复](#) [引用](#) [查看](#)

#13楼 2008-01-03 17:43 edist

支持...
[回复](#) [引用](#) [查看](#)

#14楼 2008-01-03 17:45 dcc[未注册用户]

好文,收藏
[回复](#) [引用](#)

#15楼 2008-01-03 17:54 beyondjay

经典!
[回复](#) [引用](#) [查看](#)

#16楼 2008-01-03 18:00 马可香蕉

不错, 收藏
[回复](#) [引用](#) [查看](#)

#17楼 2008-01-03 18:05 风景年华

好文章!!
收藏!!
[回复](#) [引用](#) [查看](#)

#18楼 2008-01-03 18:06 beyondjay

<http://www.cs.umd.edu/~PUGH/JAVA/MEMORYMODEL>
这个连接打不开
[回复](#) [引用](#) [查看](#)

#19楼 2008-01-03 19:39 JerryChou

最后一张图未显示
[回复](#) [引用](#) [查看](#)

#20楼 2008-01-03 19:52 Leepy

不错，先收藏了！
[回复](#) [引用](#) [查看](#)

#21楼 2008-01-03 19:54 Yanggh[未注册用户]

楼主你太棒了
[回复](#) [引用](#)

#22楼 2008-01-03 21:23 风过 无痕

对于一个winform，我想利用用户输入的数据计算。计算完之后，可以操作winform么？
[回复](#) [引用](#) [查看](#)

#23楼 2008-01-03 21:30 阿土伯[未注册用户]

示例错误很多,只能说明意思。
[回复](#) [引用](#)

#24楼 2008-01-03 21:40 路西菲尔[未注册用户]

请多使用lock,少用Mutex
如果你一定要使用锁定,请尽量不要使用内核模块的锁定机制,比如.net的Mutex,Semaphore,AutoResetEvent,ManuResetEvent,使用这样的机制涉及到了系统在用户模式和内核模式间的切换,所以性能差很多,但是他们的优点是可以跨进程同步线程,所以应该清楚的了解到他们的不同和适用范围
=====
在线程较少(小于等于4)或多核下lock就是critical的性能不咋样,应该降低锁力度而不是靠lock.而且lock也要用到内核对象
有兴趣的可以看看我写的比lock快的锁
<http://blog.csdn.net/fuadam/archive/2007/12/18/1949630.aspx>
[回复](#) [引用](#)

#25楼[楼主] 2008-01-03 21:45 二毛五

--引用-----
风过 无痕： 对于一个winform，我想利用用户输入的数据计算。计算完之后，可以操作winform么？

你是想在另外一个线程计算,然后再winform中的控件上显示的话,当然可以,,你可以把显示计算结果的操作封装在一个方法里面,计算完后用BeginInvoke或者Invoke来调用这个方法,如果你所说的"操作winform"只是操作一些自己定义的字段的话,就不需要用BeginInvoke了

[回复](#) [引用](#) [查看](#)

#26楼[楼主] 2008-01-03 21:55 二毛五

--引用-----
路西菲尔： 请多使用lock,少用Mutex
如果你一定要使用锁定,请尽量不要使用内核模块的锁定机制,比如.net的Mutex,Semaphore,AutoResetEvent,ManuResetEvent,使用这样的机制涉及到了系统在用户模式和内核模式间的切换,所以性能差很多,但是他们的优点是可以跨进程同步

线程,所以应该清楚的了解到他们的不同和适用范围

=====

在线程较少(小于等于4)或多核下lock就是critical的性能不咋样,应该降低锁力度而不是靠lock.而且lock也要用到内核对象
有兴趣的可以看看我写的比lock快的锁

<http://blog.csdn.net/fuadam/archive/2007/12/18/1949630.aspx>

lock绝对不会用到内核对象,这点请注意,你的博客打不开(csdn的blog歇了),但是前几天我恰巧看到过一片文章讲自己实现比Monitor.Enter快的锁,不知道是不是你的大作,里面提到Monitor.Enter是用内核来实现的,这是不对的,有两个错误
一是:DotNet并非使用Win32API的CriticalSection来实现Monitor.Enter,不过他为托管对象提供了一个类似的结构叫做Syncblk
二是:即使是CriticalSection也不涉及内核对象,win32可以用来同步的内核对象包括Process,Thread,File,控制台输入,文件变化通知,Mutex,Semaphore,Event,Timer

不过在多核下lock的性能我并没有尝试过,希望能一同探讨

[回复](#) [引用](#) [查看](#)

#27楼[楼主] 2008-01-03 22:01 二毛五

实例随手写就,只为说明意思,如若想要运行,自行修改调试

[回复](#) [引用](#) [查看](#)

#28楼 2008-01-03 23:06 搜索人生

。net好像无法即时控制线程终止

[回复](#) [引用](#) [查看](#)

#29楼[楼主] 2008-01-03 23:23 二毛五

--引用-----

搜索人生：。net好像无法即时控制线程终止

无法及时控制是什么意思,使线程终止.net肯定是有方法的,那么你所说的无法及时控制是指什么

[回复](#) [引用](#) [查看](#)

#30楼 2008-01-04 08:31 白菜园

好文,但好像少了ReaderWriterLock的介绍

[回复](#) [引用](#) [查看](#)

#31楼 2008-01-04 08:40 CooS

学习学习

[回复](#) [引用](#) [查看](#)

#32楼 2008-01-04 09:06 Rivers Zhao

好文, up~!

[回复](#) [引用](#) [查看](#)

#33楼 2008-01-04 09:11 overred

总结的不错
不过还应该介绍下Net2.0 里的新东西： ParameterizedThreadStart 和BackgroundWorker 以及抛弃1.x里的suspend和resume后，使用waitone、 Mutex等替代的方法
呵呵

最近狂用多线程处理：
<http://www.cnblogs.com/overred/archive/2007/12/31/cjtoo.html>
[回复](#) [引用](#) [查看](#)

#34楼 2008-01-04 09:30 ytzong

不错，希望多些WEBFORM多线程操作大数据的实例
[回复](#) [引用](#) [查看](#)

#35楼 2008-01-04 09:33 ithurricane

好文，收藏

[回复](#) [引用](#) [查看](#)

#36楼 2008-01-04 09:37 斧头帮少帮主

收藏...
[回复](#) [引用](#) [查看](#)

#37楼[楼主] 2008-01-04 09:46 二毛五

to overred:
关于ParameterizedThreadStart和BackgroundWorker下面两个blog有讨论
[多线程总结二](#)
[多线程总结一](#)

to 白菜园 :
又被你看起来偷懒了,ReadWriteLock和IOCP我都想写写来着,不过对于ReadWriteLock我实在想不出来比和<.Net 框架程序设计>这两本fery Richard的大作更丰富的内容了,而要从头介绍,篇幅也大了,所以只好忍痛...
IOCP自己觉得理解还不到位,况且许多前辈也写的很好

希望下次有心得的时候能补上

[回复](#) [引用](#) [查看](#)

#38楼 2008-01-04 10:02 路西菲尔[未注册用户]

to 二毛五：
那篇文章是我写的,最近csdn的blog极不稳定.

可能是我描述上或理解上有问题.我想说的是criticle如果不使用内核中的调度器怎么实现让线程阻塞和恢复呢.
[回复](#) [引用](#)

#39楼 2008-01-04 10:07 蛙蛙池塘

@二毛五
lock貌似确实有时会用到内核对象，你自己用windbg抓dump看看，有时候就是用了win32的临界区对象，还有什么ay什么lock,我也不知道为什么，按理说moniter是用clr的同步块儿实现的。

readerwriterlock建议不要用，性能不行。
[回复](#) [引用](#) [查看](#)

#40楼 2008-01-04 10:14 蛙蛙池塘

刚说错了，我抓dump看到了AwareLock，这是也是一个托管对象，不是内核对象，但原理和win32临界区类似。

以下引用自<http://blog.joycode.com/gangp/articles/36225.aspx>
关于lock的基本知识

托管对象的结构如下：

m_SyncBlockValue

对象指针 —> m_pMethodTable

Data

在每个托管对象的开始是该对象类型的方法表。在方法表之前是m_SyncBlockValue。

m_SyncBlockValue 的高6位用来标记m_SyncBlockValue的用途。SyncBlockValue 的低26位用来存储哈希码，SyncBlock索引或SpinLock。

低26位值的含义由高6位来决定。

在lock一个对象时，CLR首先将m_SyncBlockValue当做一个SpinLock使用。

如果CLR在有限次的重试后无法获得SpinLock的拥有权，CLR将SpinLock升级为一个AwareLock。AwareLock类似与Windows中的CriticalSection。

[回复](#) [引用](#) [查看](#)

#41楼 2008-01-04 10:20 路西菲尔[未注册用户]

请问@蛙蛙池塘
在读多写少时,不用.net的这个性能不好的rwlock,用什么呢

回来自己写一个看看.net的这个rwlock是不是真不行,我觉得ms给的东西不是全部都好用

[回复](#) [引用](#)

#42楼 2008-01-04 10:28 钢钢

.Net线程问题解答 很好，我也正在看线程的知识

[回复](#) [引用](#) [查看](#)

#43楼[楼主] 2008-01-04 10:48 二毛五

--引用-----
路西菲尔： 请问@蛙蛙池塘
在读多写少时,不用.net的这个性能不好的rwlock,用什么呢

回来自己写一个看看.net的这个rwlock是不是真不行,我觉得ms给的东西不是全部都好用

关于ReadWriteLock,jeffery Richard 自己有托管实现,据说性能比ms的好了很多

[回复](#) [引用](#) [查看](#)

#44楼 2008-01-04 10:54 蛙蛙池塘

@路西菲尔
你自己用moniter或者自旋锁写一个也行呀，懂原理了还不好办呀，读是一个Q，写用一个变量，写线程获取锁后就让一个线程执行，读线程获取锁后一个Q的所有线程都依次执行。

.net的rwlocker有很多不好的地方，不支持递归，性能差，用不好会出现写线程饥饿...
如楼主所说《CLR框架设计》的作者的网站上有一个代替rwlock的对象。

[回复](#) [引用](#) [查看](#)

#45楼 2008-01-04 11:01 spido[未注册用户]

蛙蛙池塘：
最近遇到了许多问题,都没法用托管的调试器,Windbg确实很强大,一直想搞,不过还没有入门,蛙蛙能给推荐下吗?

[回复](#) [引用](#)

#46楼 2008-01-04 11:27 蛙蛙池塘

以前整理过一些资料（不包括最近一个月博客园朋友发的windbg的资料），如下

驱动器 E 中的卷是 新加卷
卷的序列号是 4823-C0AD

E:\book\debug_stduy 的目录

[.]
[..]
1.txt
Debugging.Applications.For.Microsoft..NET.And.Microsoft.Windows.chm
DotNet同步策略 .doc
kernel_debugging_tutorial.doc
link.txt
Microsoft.Press.Debugging.Microsoft.dot.NET.2.0.Applications.Nov.2006.chm
Production Debugging for .NET Framework Applications.pdf
sos参考.mht
UsermodeTsPaper_not_bookversion.pdf
windbg 脚本程序编写.doc
WinDbg.chm
windbg_link.txt
windbg_tbasic.chm
windbg参考.txt
windbg文章整理.doc
【原创】ANTS Profiler(for _net)的分析、调试及破解.mht
为多线程处理同步数据.doc
使用WINDBG和SOS调试死锁.doc
可靠性最佳做法.doc
同步、异步、阻塞和非阻塞的概念.doc
同步基元概述.doc
多线程基于.NET应用程序迅速响应.doc
托管线程处理的最佳做法.doc
简明x86汇编语言教程.doc
调试分布式Web应用程序.mht
通往WinDbg的捷径（一）.doc
28 个文件 19,564,964 字节
2 个目录 20,202,332,160 可用字节

你加我QQ我发给你,415492354
[回复](#) [引用](#) [查看](#)

#47楼[楼主] 2008-01-04 11:29 二毛五

好东西,我先领一份
[回复](#) [引用](#) [查看](#)

#48楼 2008-01-04 11:29 蛙蛙池塘

哦，对了
NET框架应用程序产品调试指南(第一章).TXT
NET框架应用程序产品调试指南(第二章).TXT
这两个不能给你，是未经允许的。
[回复](#) [引用](#) [查看](#)

#49楼 2008-01-04 13:22 jun

--引用-----

实际上在web开发中大多数逻辑都是在单个线程中展开的,无论asp.net还是php,一个请求都会在一个单独的线程中处理,其中的大部分变量都是属于这个线程的,根本没有必要考虑锁定

我的印象是每个请求的确是一个单独的线程,但不必考虑锁定却是因为asp.net为每个请求都生成一个Page的实例,所以互不干扰.

[回复](#) [引用](#) [查看](#)

#50楼 2008-01-04 14:11 a feng

“后台线程不会使托管执行环境处于活动状态”

请问什么叫“活动状态”，如何判断？

[回复](#) [引用](#) [查看](#)

#51楼 2008-01-04 14:11 varptr

ManualResetEvent类, AutoResetEvent类、ReaderWriterLock的介绍：

<http://www.cnblogs.com/uubox/archive/2007/12/18/1003953.html>

[回复](#) [引用](#) [查看](#)

#52楼[楼主] 2008-01-04 14:24 二毛五

--引用-----

a feng: “后台线程不会使托管执行环境处于活动状态”

请问什么叫“活动状态”，如何判断？

其实应该这样说,如果使用前台线程,在所有线程执行完成前,进程无法终止
而使用后台线程,当进程的所有前台线程都终止后，公共语言运行时将对仍处于活动状态的后台线程调用 **Abort** 方法，以结束该进程。

[回复](#) [引用](#) [查看](#)

#53楼 2008-01-04 15:38 a feng

@二毛五

谢谢，你这篇精彩的文章关注的人很多啊

另外想请问一下

“锁定与同步”那副漂亮的图是哪里来的，能否给个URL？

[回复](#) [引用](#) [查看](#)

#54楼 2008-01-04 15:45 jojo

有很多免费资源供下载!!

<http://www.alldnnskins.com.cn> (电信)

<http://www.alldnnskins.cn> (网通)

[回复](#) [引用](#) [查看](#)

#55楼 2008-01-04 16:22 绿蚂蚁

学习了，刚好用上IsBackground属性
thanks

[回复](#) [引用](#) [查看](#)

#56楼 2008-01-04 16:58 浴盆

使用Delegate.BeginInvoke
最后
public static void TaskFinished(IAsyncResult result)
{
result=calcMethod.EndInvoke(result);
Console.WriteLine(result);
}
提示错误
无法将 IAsyncResult 转成 double
在什么原因呢?

[回复](#) [引用](#) [查看](#)

#57楼 2008-01-04 17:07 YanziMyWife

太强大了!

[回复](#) [引用](#) [查看](#)

#58楼 2008-01-04 21:45 yizhu2000[未注册用户]

--引用-----
a feng: @二毛五
谢谢，你这篇精彩的文章关注的人很多啊
另外想请问一下
"锁定与同步"那副漂亮的图是哪里来的，能否给个URL?

<http://msdn.microsoft.com/msdnmag/issues/05/05/JITCompiler/>
[回复](#) [引用](#)

#59楼 2008-01-04 21:47 yizhu2000[未注册用户]

--引用-----
浴盆：使用Delegate.BeginInvoke
最后
public static void TaskFinished(IAsyncResult result)
{
result=calcMethod.EndInvoke(result);
Console.WriteLine(result);
}

提示错误
无法将 **IAsyncResult** 转成 **double**
在什么原因呢?

改成**double re;re=calcMethod.EndInvoke(result)**
[回复](#) [引用](#)

#60楼 2008-01-05 10:12 [浴盆](#)

--引用-----
yizhu2000: --引用-----
浴盆: 使用Delegate.BeginInvoke
最后
public static void TaskFinished(IAsyncResult result)
{
result=calcMethod.EndInvoke(result);
Console.WriteLine(result);
}
提示错误
无法将 **IAsyncResult** 转成 **double**
在什么原因呢?

改成**double re;re=calcMethod.EndInvoke(result)**

你好,非常感谢,现在编译可以通过了,但是为什么没有输出结果呢?
[回复](#) [引用](#) [查看](#)

#61楼 2008-01-05 10:21 yizhu2000[未注册用户]

--引用-----
浴盆: --引用-----
yizhu2000: --引用-----
浴盆: 使用Delegate.BeginInvoke
最后
public static void TaskFinished(IAsyncResult result)
{
result=calcMethod.EndInvoke(result);
Console.WriteLine(result);
}
提示错误
无法将 **IAsyncResult** 转成 **double**
在什么原因呢?

改成**double re;re=calcMethod.EndInvoke(result)**

你好,非常感谢,现在编译可以通过了,但是为什么没有输出结果呢?

请输出**re**变量
[回复](#) [引用](#)

#62楼 2008-01-05 10:57 浴盆

--引用-----

yizhu2000: --引用-----

浴盆: --引用-----

yizhu2000: --引用-----

浴盆: 使用Delegate.BeginInvoke

最后

```
public static void TaskFinished(IAsyncResult result)
```

```
{
```

```
result=calcMethod.EndInvoke(result);
```

```
Console.WriteLine(result);
```

```
}
```

提示错误

无法将 IAsyncResult 转成 double

在什么原因呢?

改成double re;re=calcMethod.EndInvoke(result)

你好,非常感谢,现在编译可以通过了,但是为什么没有输出结果呢?

请输出re变量

非常感谢你的热心解答

```
public static void TaskFinished(IAsyncResult result)
```

```
{
```

```
double re;
```

```
re=calcMethod.EndInvoke(result);
```

```
Console.WriteLine(re);
```

```
}
```

我现在改成的是这样的,但是还是没有结果输出到控制台

其他的地方和你给的例子一样,也通过编译了

难道还有别的地方不对吗?

[回复](#) [引用](#) [查看](#)

#63楼 2008-01-05 14:26 a feng

@yizhu2000

谢谢,牛人,好人:)

[回复](#) [引用](#) [查看](#)

#64楼 2008-01-05 14:29 a feng

@yizhu2000

您给的 URL没有那图麻

<http://msdn.microsoft.com/msdnmag/issues/05/05/JITCompiler/>

我问的是

“

每个对象实例头部都有一个指针,这个指针指向的结构,包含了对象的锁定信息,当第一次使用**Monitor.Enter(obj)**时,这个obj对象的锁定结构就会被初时化,第二次调用**Monitor.Enter**时,会检验这个object的锁定结构,如果锁没有被释放,则调用会阻塞 ”
下面的那图

[回复](#) [引用](#) [查看](#)

#65楼 2008-01-05 14:31 a feng

@yizhu2000
看到了，原来在弹出窗口里：)
[回复](#) [引用](#) [查看](#)

#66楼 2008-01-05 16:11 Enzo

o(∩_∩)o... 支持下 给你留言了!
[回复](#) [引用](#) [查看](#)

#67楼[楼主] 2008-01-05 21:21 二毛五

--引用-----
Enzo: o(∩_∩)o... 支持下 给你留言了!

你要的大概是这个Process.GetCurrentProcess().Kill(),下面就是在一个线程里杀死当前进程
static void Main(string[] args)
{
Thread t = new Thread(delegate() { Process.GetCurrentProcess().Kill(); Thread.Sleep(20000); });
t.Start();
while(true){}
}
[回复](#) [引用](#) [查看](#)

#68楼[楼主] 2008-01-05 21:32 二毛五

to浴盆:
没有输出多半是因为窗口在输出前关闭了,在begininvoke后面加个Console.ReadLine()试试

我把上面的这段程序实例修改了一下,这次是可运行版的
[回复](#) [引用](#) [查看](#)

#69楼 2008-01-06 08:17 浴盆

--引用-----
二毛五: to浴盆:
没有输出多半是因为窗口在输出前关闭了,在begininvoke后面加个Console.ReadLine()试试

我把上面的这段程序实例修改了一下,这次是可运行版的

呵呵,非常感谢,我已经可以运行了
再次感谢你的热心解答
对了,可否请问一下你的MSN或者是QQ号

[回复](#) [引用](#) [查看](#)

#70楼 2008-01-07 13:20 yizhu2000[未注册用户]

我的msn是yizhu2000@hotmail.com,上的不是很多,欢迎讨论
[回复](#) [引用](#)

#71楼 2008-01-07 15:06 brin

呵呵,不错,支持下
[回复](#) [引用](#) [查看](#)

#72楼 2008-04-28 13:12 路过178877[未注册用户]

不错不错,好文章。
[回复](#) [引用](#)

#73楼 2008-05-08 09:27 fantasy_kli

```
WaitCallback w = new WaitCallback(Calculate);
//下面启动四个线程,计算四个直径下的圆周长
ThreadPool.QueueUserWorkItem(w, 1.0);
ThreadPool.QueueUserWorkItem(w, 2.0);
ThreadPool.QueueUserWorkItem(w, 3.0);
ThreadPool.QueueUserWorkItem(w, 4.0);
public static void Calculate(object Diameter)
{
    double result = ((double)Diameter) * Math.PI;
}
```

3)所支持的Callback不能有返回值。WaitCallback只能带一个object类型的参数,没有任何返回值。

[回复](#) [引用](#) [查看](#)

#74楼 2008-05-19 23:10 cc_net[未注册用户]

恩,写的很好,很强大!
我也是刚开始学习了写多线程。下次转下你的文章
[回复](#) [引用](#)

#75楼 2008-06-03 17:48 赖文华(RICHFIT小赖)
很不错 收藏! http://www.cnblogs.com/GavinCome/archive/2008/06/03/1212960.html 回复 引用 查看
#76楼 2008-07-15 11:24 王文明[未注册用户]
太谢谢了! 回复 引用
#77楼 2008-09-03 13:51 逖靖寒
好文章，顶 回复 引用 查看
#78楼 2008-09-03 13:56 Justin
牛，一篇文章包含了这么多内容 回复 引用 查看
#79楼 2008-11-25 11:15 杭州律师事务所[未注册用户]
不错不错,收藏了慢慢品味 回复 引用
#80楼 2008-12-02 23:26 包建强
老弟， 精华集的CLR分册开始，你有文章入选， 请加我MSN：bjq_ren@hotmail.com 回复 引用 查看
#81楼 2009-01-06 17:08 .NET学徒
好文章！收藏了慢慢看！ 回复 引用 查看
#82楼 2009-01-15 10:10 adpost
学习,受益非浅 回复 引用 查看
#83楼 2009-02-03 11:36 小伦

收藏了~~谢谢楼主的辛苦整理~~
[回复](#) [引用](#) [查看](#)

#84楼 2009-07-19 20:09 65465[未注册用户]

good
[回复](#) [引用](#)

#85楼 2009-08-07 15:07 Alifellod

太感谢楼主的辛苦劳动
拜谢
[回复](#) [引用](#) [查看](#)



发表评论

[刷新评论列表](#) [刷新页面](#) [返回页首](#)

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](#)

[免费学习asp.net课程](#)

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](#)

[寻找18-28岁待业者](#)

权威:华浦ISEP国际软件工程师 免费:就业讲座帮您找到好工作

[www.isen.com.cn](#)

[找个老外学英语](#)

迅速提高英语口语 口语就要这样练

[www.MarsEnglish.com](#)

[专业Web报表.NET应用-博计](#)

IIS服务发布报表,无插件,纯HTML报表 开发Web报表不用写SQL,不用GridView

[www.honzereport.com](#)



[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:

博客园.[NET](#)频道, 专业.[NET](#)技术门户

博客园.[net](#)频道上线啦!

[.Net](#)新手专题

[ASP.NET](#)技术内幕

[ASP.NET MVC](#) 专题, 从零开始学.[NET](#)技术

[.Net](#)程序员如何30天突破英语听说?

你必须知道的.[NET](#)

Powered by:

[博客园](#)

Copyright © 朱燧:-)