**使用线程 Monitor.Wait() 和 Monitor.Pulse()**

Wait() 和 Pulse() 机制用于线程间交互。当在一个对象上使用Wait() 方法时，访问这个对象的线程就会一直等待直到被唤醒。Pulse() 和 PulseAll() 方法用来通知等待的线程醒来的。下面是关于Wait() 和 Pulse() 方法如何运行的例子，WaitAndPulse.cs:

Wait() 和 Pulse() 方法仅可以在Enter() 和 Exit() [代码](http://www.xuebuyuan.com/)块内部

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

namespace ThreadEx

{

public class LockMe

{

}

class WaitPalsel

{

private int result = 0;

private LockMe lM;

public WaitPalsel(LockMe l)

{

this.lM = l;

}

public void CriticalSection()

{

Monitor.Enter(this.lM);

Console.WriteLine("WaitPulse1: Entered Thread " + Thread.CurrentThread.GetHashCode());

for (int i = 1; i <= 5; i++)

{

Monitor.Wait(this.lM);

Console.WriteLine("WaitPulse1: Result =" + result++ + "ThreadID"

+ Thread.CurrentThread.GetHashCode());

Monitor.Pulse(this.lM);

}

Console.WriteLine("WaitPulse1: Exiting Thread " + Thread.CurrentThread.GetHashCode());

Monitor.Exit(this.lM);

}

}

class WaitPulse2

{

private int result = 0;

private LockMe lM;

public WaitPulse2()

{ }

public WaitPulse2(LockMe l)

{

this.lM = l;

}

public void CriticalSection()

{

Monitor.Enter(this.lM);

Console.WriteLine("WaitPulse2: Entered Thread " + Thread.CurrentThread.GetHashCode());

for (int i = 1; i < 5; i++)

{

Monitor.Pulse(this.lM);

Console.WriteLine("WaitPulse2: Result ="+result++ +"ThreadID"

+Thread.CurrentThread.GetHashCode());

Monitor.Wait(this.lM);

Console.WriteLine("WaitPulse2: WokeUp");

}

Console.WriteLine("WaitPulse2 Exiing Thread " + Thread.CurrentThread.GetHashCode());

Monitor.Exit(this.lM);

}

}

class Program

{

static void Main(string[] args)

{

LockMe l = new LockMe();

WaitPalsel e1 = new WaitPalsel(l);

WaitPulse2 e2 = new WaitPulse2(l);

Thread t1 = new Thread(new ThreadStart(e1.CriticalSection));

t1.Start();

Thread t2 = new Thread(new ThreadStart(e2.CriticalSection));

t2.Start();

Console.ReadLine();

}

}

}

在Main() 方法中，我们创建了一个LockMe对象。然后创建了两个对象，WaitPulse1, WaitPulse2, 接着把它们委托给线程以便于线程可以调用这两个对象的CriticalSection()方法。注意WaitPulse1和WaitPulse2这两个对 象中的LockMe实例是不同的，因为传递给对应构造函数的对象引用不同。初始化完对象以后，我们创建了两个线程t1 和 t2, 并向这两个线程分别传递各自的CriticalSection()函数。

假设WaitPulse1.CriticalSection() 先执行，线程t1 进入方法的关键部分并在锁住LockMe对象后在for循环中执行Monitor.Wait()。由于执行了Monitor.Wait(), 所以它得等待其他线程调用Monitor.Pulse()方法(一个运行时通知)来将其唤醒。我们锁住LockMe对象是因为我们只希望在任意时间仅有一 个对象访问共享LockMe实例。

注意当线程执行Monitor.Wait()方法时，它会暂时释放LockMe对象上的锁，这样其他线程就可以访问LockMe对象。在线程t1进 入等待状态后，线程t2可以自由地访问LockMe对象。尽管这两个线程都有自己的LockMe对象(WaitPulse1, WaitPulse2)，但是它们都引用同一个对象。线程t2获得LockMe对象上的锁并进入 WaitPulse2.CriticalSection()方法。当它进入for循环时，它给等待LockMe对象的线程(本例中是t1)发送一个运行时 通知(Monitor.Pulse())然后进入等待状态。

最终，t1醒来并获得LockMe对象的锁。线程t1然后访问result变量并向等待LockMe对象的线程(本例中为t2)发送一个运行时通知。如此反复直到for循环结束。

如果你依据[程序](http://www.xuebuyuan.com/" \o "程序" \t "_blank)的输出结果来看上面的描述，那么我们说的概念会非常清晰易懂。要注意每个Enter()方法都有一个Exit()方法匹配否则程序不应该结束。

Enter()方法接受一个对象作为参数。如果参数为null 或者参数时一个方法名或者一个值类型的对象(比如int型)，Enter()方法都会抛出异常。