**.NET程序集高级-多线程进阶**

**之**

**同步锁解决并发问题**

如果多个线程并发访问同一对象的共享资源，则需要运用同步锁解决并发问题。

* **使用C#推荐的同步锁lock**

lock同步锁用于锁住多个线程并发访问同一个对象的相同资源（如对象的方法），从而实现线程安全并发访问，即一个线程正在访问该方法时，其他线程阻塞等待。lock必须要有一个锁标记（一个成员对象引用），针对lock锁住的方法的访问权限不同，锁标记有以下规则：

* 方法为praivte时，锁标记可以为this

//定义线程同步访问对象

public class interviewee

{

//私有方法

private void priFun()

{

//多线程同步锁

//锁标记：this

lock (this)

{

//模拟耗时任务

}

//下面任务不会阻塞其他同步线程

}

}

* 方法为public时，锁标记为私有成员对象

//定义线程同步访问对象

public class interviewee

{

//锁标记

private object threadLock = new object();

public void pubFun()

{

//多线程同步锁

//锁标记：threadLock

lock (threadLock)

{

//模拟耗时任务

}

//下面任务不会阻塞其他同步线程

}

}

* 方法为静态时，锁标记为静态私有成员对象

//定义线程同步访问对象

public class interviewee

{

//锁标记

private static object staticthreadLock = new object();

public void pubFun()

{

//多线程同步锁

//锁标记：staticthreadLock

lock (staticthreadLock)

{

//模拟耗时任务

}

//下面任务不会阻塞其他同步线程

}

}

* **使用类System.Thread.Monitor同步**

实际上，关键字lock所修饰的同步锁，C#最后会编译成使用System.Thread.Monitor进行同步：

//多线程同步锁

//锁标记：threadLock

lock (threadLock)

{

//模拟耗时任务

}

等效于

//多线程同步锁

//锁标记：threadLock

Monitor.Enter (threadLock);

//捕获如何异常

try

{

//模拟耗时任务

}

//确保同步锁能够被释放

finally

{

Monitor.Exit (threadLock);

}

* **使用类System.Thread.Interlocked同步**

使用类System.Thread.Interlocked提供的静态方法，可以对数值运算进行原子操作，如下面的原子操作：

* 递加1：Increment(ref intVal)
* 递减1：Decrement(ref intVal)

lock (threadLock)

{

intVal++;

}

等效于

//返回递加1后的值

int newVal = System.Thread.Interlocked. Increment(ref intVal) ;

* **使用类特性[Synchronization]同步**

[Synchronization]位于System.Runtime.Remoting.Contexts命名空间下面。使用该特性的类必须继承ContextBoundObject类（同步上下文），这样，该类中的所有方法将被自动赋予同步锁。

//定义线程同步访问对象

//使用特性[Synchronization]进行同步

//类必须继承ContextBoundObject

[Synchronization]

public class interviewee : ContextBoundObject

{

//私有方法已经实现线程同步

private void priFun()

{

//模拟耗时任务

}

//公共方法已经实现线程同步

public void pubFun()

{

//模拟耗时任务

}

}