Java.util.concurrent同步器(Synchronizer)框架

摘要

在J2SE1.5的java.util.concurrent包里，很多同步器(locks,barriers等)的实现都用到了一个小的框架，他们建立在AbstractQueuedSynchronizer类之上。AQS框架提供了一些通用的机制，像自动管理同步状态，线程阻塞，线程解锁，线程排队。这篇论文描述了AQS框架的理论基础，设计，实现，用法和性能

类别和主题

并发编程设计，并行编程设计

一般用语

算法(Algorithms)，测量(Measurement)，性能(Performance)，设计(Design)

关键字

同步(Synchronization)，Java

1. 简介

Java的J2SE的1.5版引入了java.util.concurrent包。它是一些中级并发支持的类的集合遵循JCP(Java Community Process)JSP166。这些组件是一套同步器---抽象数据类型(ADT)类。它实现了一个内部的同步状态(例如标志一个锁是锁住或者解锁)，更新和检查状态的操作，以及在状态需要时至少一个会引起线程阻塞的方法，当一些其他线程改变同步状态允许它重新恢复的方法。并发包里包括各种形式的排他锁(exclusion locks)，读写锁(read-write locks)，信号量(semaphores)，栅栏(barriers)，futures，甚至指示器(indicators)及可相互切换的队列(handoff queues)。

众所周知，几乎所有的同步器都可以实现其他的组建。例如，可以使用重入锁(reentrant locks)建立一个信号量(semaphores)，反之也行。然而这样做通常足够复杂，经常不灵活。是一个二流的工程观点。更进一步说，这样做从概念上是枯燥的。如果没有这些构造本质上比其他开发者更原始，开发人员不应被迫任意选择其中一个作为建立他人的基础。所以，JSR166围绕AbstractQueuedSynchronizer建立了一个小的框架，提供了一些在并发包提供的大多数同步器组件使用的通用的机制。论文剩下的部分这个框架的需要。主要包括框架的设计，实现，使用例子，和一些展示它性能特点的测量方式。

1. 要求
   1. 功能性

同步器执行两重方法：至少一个acquire操作，以阻塞调用的线程至少/直到同步器状态允许它执行，至少一个release操作，可以允许一个或多个阻塞的线程解锁以改变同步状态。

Java.util.concurrent包并没有为同步器定义一个单独的统一API。有些是通过通用接口(例如Lock)定义的，其他的仅仅包括专门的版本。所以，acquire和release操作在不用的类里，有不同的名称和形式。例如：Lock的lock()方法，Semephore的acquire()方法，CountDownLatch的await()方法，FutureTask的get()方法都映射到了AQS框架的acquire操作上。然而，并发包在不同类间建立一致性的便利去支持一系列通用的使用操作。富有意义的，每个同步器都支持：

1. 非阻塞及阻塞的同步重试(例如tryLock)
2. 操作超时，以便于应用可以放弃等待
3. 发生异常时取消操作，通常分为一个版本的获取时可以取消的，另一个不可以。

同步器根据它管理的排他状态发生改变---在某一时刻只能有一个线程可以继续通过一个可能的阻塞点---相反也可能多个线程可以分享状态，至少有时执行。普通的锁当然仅仅有排他状态，但是例如计数信号量，可以在数量允许时可以被尽可能多的线程获取。为了更广泛的应用，AQS框架必须支持这两种操作模型。

Java.util.concurrent包也定义了Condition接口，支持监视样式的await/signal操作。它与排他锁Lock类有关，它的实现从本质上讲与它关联的锁Lock类交织在一起。

* 1. 性能目标

Java内置的锁(近似于使用synchronized方法和阻塞)长久以来有性能问题，有很多关于他们构建的文章。然而，内置锁的主要目的是在单处理器里，在单线程上下文中使用时，最小化的空间消耗(因为每个java对象都能当作锁)和最小化的时间消耗。这些在同步器中都不关系：程序员只有在需要时才使用同步器，所以不是必须会影响到空间消耗，也不会浪费。同步器在多线程设计(通常是日益增长的多处理器)中排他性使用，至少像预期那样在竞争时发生。所以通常优化JVM锁的策略基本上就是零竞争场景。其他的是可预测的慢路径(slow paths)不是正确的策略对典型的多线程服务器应用严重依赖java.util.concurrent包。

取而代之，这里基础的性能目标是可测量的：预测构建效率，甚至在同步器竞争时。理论上，无论多少线程去竞争同步点，都应该是常数。主要目标是在一些线程被允许通过同步点时，减小总的时间，而不是不这样做。然而这些必须平衡

1. 设计和实现

同步器最好的方式就是非常直接。

一个获取(acquire)操作流程大概这样：

while(同步状态不允许获取){

如果线程还没排队，当前线程入队；

尽可能的阻塞当年线程

}

一个释放(release)操作大概是：

更新同步状态；

if(状态允许一个阻塞线程获取)

非阻塞一个或多个排队的线程；

这些操作需要三个基础组件的协作：

1. 自动管理同步状态
2. 阻塞和非阻塞线程
3. 维护排队队列

创建一个框架，允许这三个组件独立运行是可行的。但是这样既不高效也不可用。例如在队列节点持有的信息必须与非阻塞需要配合，对外暴露的方法签名依赖同步状态的性质。

同步器框架中心设计决定选取在三个组件中每个都有一个具体的实现。但仍然允许在如何使用它们的范围广泛的选项。这些有意的限制适应性的范围，但是提供了足够高效的支持。这是实践从来不是一个不可用的框架()在应用它时。

1. 用例
2. 性能
3. 结论
4. 致谢
5. 参考