# 多线程

## 阻塞队列

### BlockingQueue

根据jdk1.7，查看源码注释得到下面的记录。

提供的方法如下：

#### boolean add(E e)

在队列中插入一个元素。如果队列**没有容量限制**或者**有可用的空间**，插入成功，并且返回true，否则，抛出异常IllegalStateException。

这个方法是返回一个boolean类型的返回值，但是注释里只说失败了会抛出异常，没说失败了会返回false，应该就是不返回false。

假如队列存在容量限制，我们推荐优先使用offer方法。

#### boolean offer(E e)

在队列中插入一个元素。如果队列**有可用的空间**，插入成功，并且返回true；如果没有可用的空间，返回false。

当队列存在容量限制的时候，我们推荐优先使用该方法。

上面两个插入方法，注释里并没有提及没有可用空间会阻塞线程的问题，所以认为是如果没有可用空间，立即返回了。

#### void put(E e)

在队列中插入一个元素，当没有可用的空间的时候，会阻塞插入线程，直到有可用空间成功插入元素为止。

#### boolean offer(E e, long timeout, TimeUnit unit)

这个和前面的offer一样，只是加了一个等待时间，当没有可用空间的时候，会等待一个时间。成功插入，返回true；如果等待时间结束，队列仍然没有可用空间，返回false。

#### E take()

返回并且一处队列的头元素，如果队列当前为空，会阻塞取元素的线程，直到队列有可用的元素为止。

#### E poll(long timeout, TimeUnit unit)

返回并且一处队列的头元素，如果队列当前为空，会阻塞取元素的线程tiemout的时间。如果成功取到，返回对应的元素；否则，返回null。

#### ArrayBlockingQueue

该类实现了接口BlockingQueue。

类注释翻译整理如下：

使用一个有界的数组缓存数据，是一个先进先出(FIFO)的队列。其它的说明，无非就是插入和取出元素时阻塞之类的。

重点说一下关于等待策略：

当有很多线程来取元素的时候，如果使用公平策略，能严格执行先进先出策略，但是这样会降低系统的吞吐量。如果使用不公平策略，则不能完全保证先进先出策略，比如消费者线程，当有可用的元素的时候，只能保证这些线程一起去争夺这个可用元素，但是不能保证先进先出。

对于添加元素和删除元素都使用**同一个锁**。

## 可重入锁

字面意思就是当一个线程获取到了某个锁以后，该线程可以多次获得该锁。

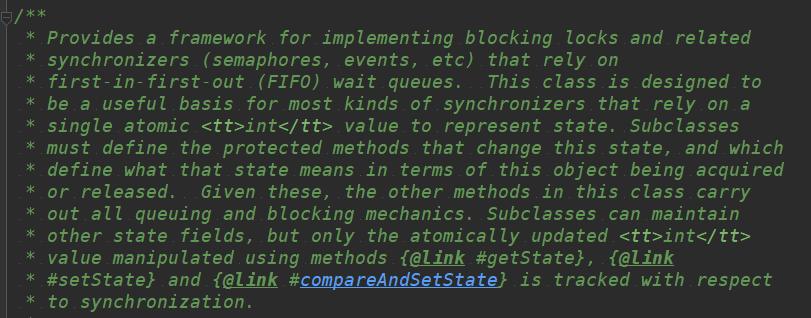
可重入锁的意义就在于可以防止死锁。

### AbstractQueuedSynchronizer

说明：只是把源码里的注释翻译了一下，并且简单写一下在使用的时候，需要重写的方法，以及方法的作用。并会提供几个Demo。

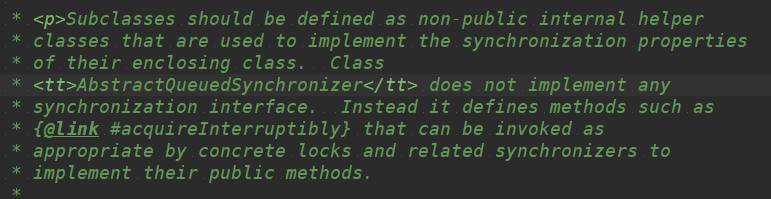
首先，说明一下这个类的作用。

1. 通过使用一个先进先出(FIFO)的队列为我们实现**阻塞锁**和**同步器**提供了一个框架。该类可以作为实际中大部分同步器的实现基础，它所依赖的是一个代表状态的int类型的值（下面用state代替）。子类必须去实现该类中的protected方法，这些方法是用来改变state的值，state的值代表了锁的释放和占有。该类里面其它的方法实现了队列以及阻塞机制。获取state的值只能使用getState，修改state的值只能使用setState或者compareAndSetState。

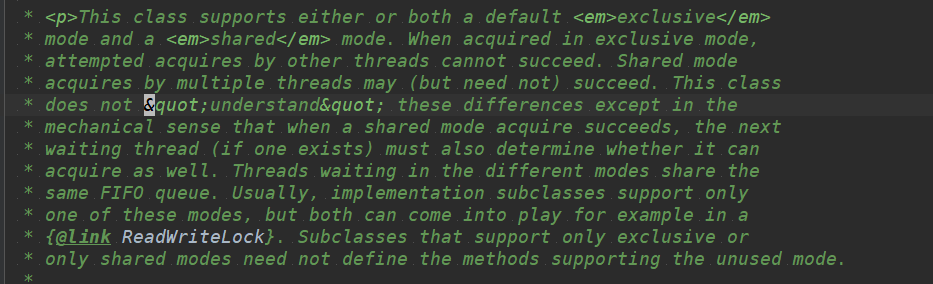


1. 继承该类的子类被推荐为声明为一个**非静态的内部类(non-public internal helper classes)**。

~~该类的内里机制并不知道，按照官方推荐实现就好，另外这一部分缺失了一部分翻译。~~



1. 该框架类支持**独占锁**和**共享锁**。独占模式下，锁被一个线程获取到后，其它线程只能等待；共享模式下，如果锁被一个线程获取到了，其它线程再次尝试获取锁，也需要判断，不是直接就获取成功了。通常，在实际应用中，我们只需要实现一个模式就好了，要么使用独占锁，要么使用共享锁。但是也有同时实现的，比如ReadWriteLock，写是独占锁，读是共享锁。



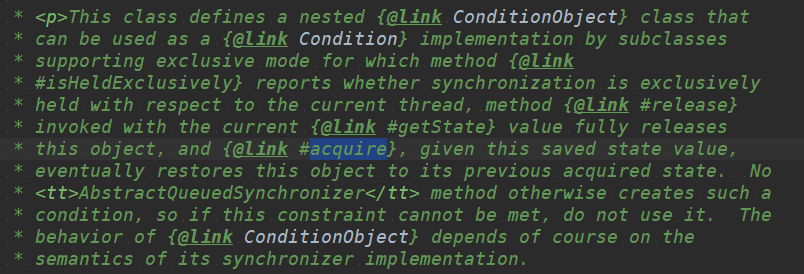
1. 该框架类内部还实现了一个ConditionObject(实现接口Condition)。

ConditionObject里的几个简单的方法：

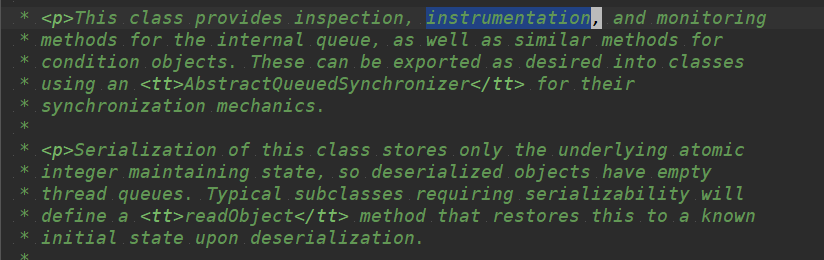
isHeldExclusively：当前线程是独占模式还是共享模式

release：释放锁

acquire：获取锁



1. 剩下的两段无所谓了。



接下来的就是官方提供的用法了。

需要重写的方法：

**特别提示**：获取和修改状态的方法。上面已经说过了。

<ul>

<li> {@link #tryAcquire}

<li> {@link #tryRelease}

<li> {@link #tryAcquireShared}

<li> {@link #tryReleaseShared}

<li> {@link #isHeldExclusively}

</ul>

好了，接下来就是该框架类里其它辅助的内部类的解释了。

#### Node

这是等待队列的Node类。

等待队列是一个锁的队列。**锁是自旋锁(其实就是个死循环，释放锁的当前线程通知一下下一个线程。还给自己起了一个这么高大上的名字，这个方法都经常用好吧)。**

这个类比较简单，简单介绍一下里面的几个成员变量和成员方法。

成员变量：

所有的成员变量都是volatile修饰的。

waitStatus：有以下几个状态值。

SIGNAL: 当前线程的**继承者**处于阻塞(通过park方法)状态。当当前线程释放锁

或者被中断以后，需要释放继承者(unpark)。

CANCELLED: 当前线程由于超时或者中断被取消了。处于该状态的线程不会被

阻塞（鬼知道阻塞到底是几个意思，可能就是被取消的永远不会获

取到锁了？）。

CONDITION: This node is currently on a condition queue.

It will not be used as a sync queue node

until transferred, at which time the status

will be set to 0. (Use of this value here has

nothing to do with the other uses of the

field, but simplifies mechanics.)

PROPAGATE: A releaseShared should be propagated to other

nodes. This is set (for head node only) in

doReleaseShared to ensure propagation

continues, even if other operations have

since intervened.

0: None of the above

prev：

next：

thread：

nextWaiter：

#### Sync