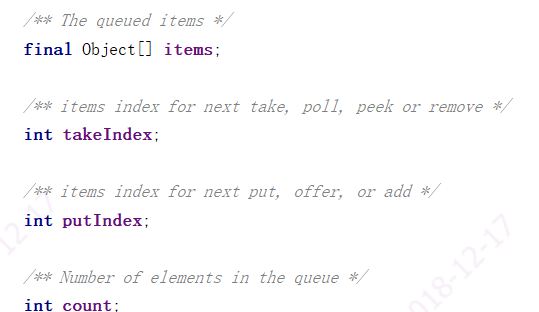
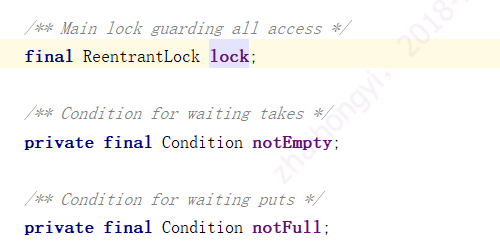
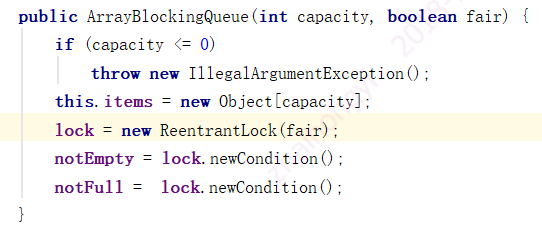
# ArrayBloQueue





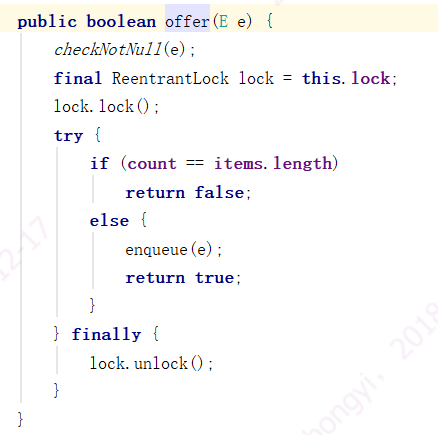
阻塞队列的成员变量，阻塞队列是线程安全的，使用了数组存储，分别提供了读写指针，同时为阻塞的读写方法提供了信号量notEmpty、notFull。

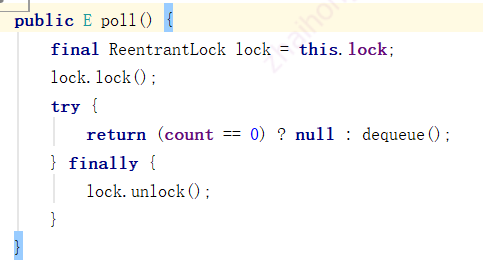


阻塞队列甚至还支持公平锁。

1. **非阻塞方法offer()，poll()**

非阻塞方法分析起来相对容易，因为线程不需要阻塞和唤醒。

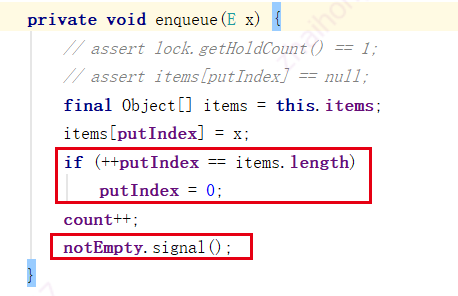


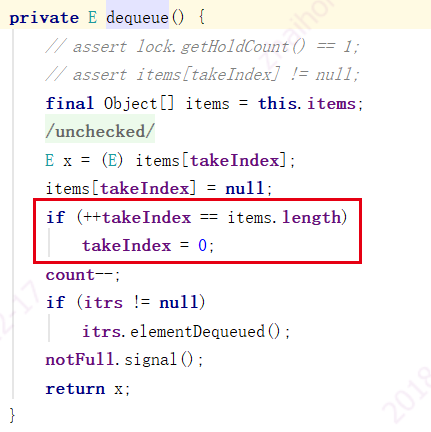


入队和出队方法由enqueue()和dequeue()完成。先看下enqueue()，会将元素放置在putIndex位点上，接着putIndex向后移动，红框一是关键，当putIndex已经越界，会重置为0，从头开始插入。原来是一个环形队列啊。但这样不会有问题吗，不会覆盖原来的值吗？【疑问一】。插入后自然要通知等待notEmpty的线程获取值。

dequeue()向后移动takeIndex指针，如果越界则从头开始。同时维护count，队列中元素的个数。出队后会通知等待notFull的线程放入元素。从语义上很好理解。

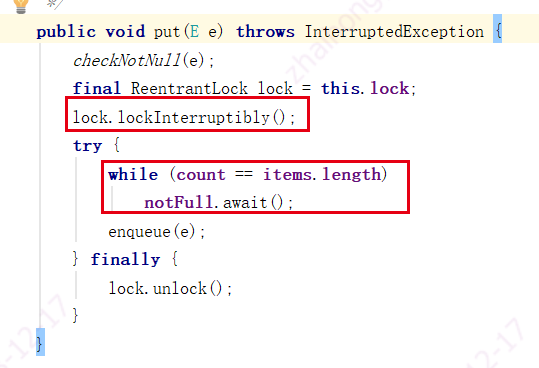
非阻塞方法当队列满了或为空时直接返回false或null，不会等待，这就是非阻塞的语义。

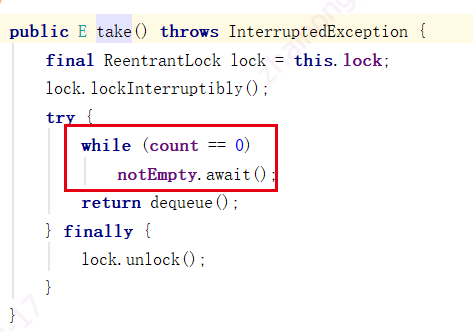




1. 阻塞方法put()，take()

阻塞方法不同，当队列满了或为空，阻塞方法会等待队列不满(notFull)或队列不空(notEmpty)。通过阅读源码，当我们需要实现类似的阻塞方法时，可以参考JDK的实现。





首先put方法的第一个红框。对于阻塞方法JDK采用了lockInterruptibly()，而不是lock方法。原因大致可以猜想，因为阻塞方法等待时间长，当等待锁的线程中断，则该线程应直接抛出异常，而不是继续等待锁。DelayQueue中的阻塞方法也是这样用的，所以当我们在使用锁，且锁的时间可能较长时，也应使用lockInterruptibly()方法。

对于put方法，当队列满时，会一直等待notFull，当有元素出队，则唤醒；take方法，当队列为空时，会一直等待notEmpty，当有元素入队，则唤醒。

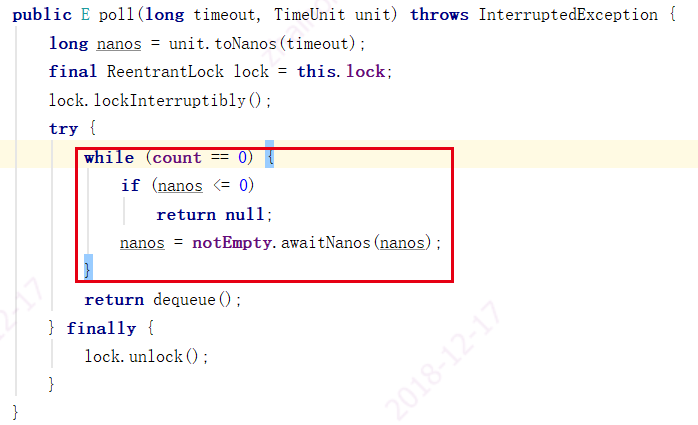
对于【疑问一】，takeIndex也会向后移动，因此当putIndex为0时，位点0的元素也是null，队列不满，可以放入位点0。当takeIndex也为0，即元素出队，则此时队列已满，调用非阻塞方法会直接返回false，调用阻塞方法，会一直等待位点0出队。

1. 带超时的阻塞方法offer(long)，poll(long)

当队列满或为空时会等待超时时间。如果在等待期间满足非满或非空条件，等待会被唤醒，唤醒后再竞争锁，拿到锁的线程继续执行。如果条件不满足则返回false或null。

由于count、putIndex、takeIndex的更新都在锁内部，线程安全且变更对于其他线程可见，不需要volatile声明。这里超时控制比DelayQueue简单多了。哈哈哈





1. 迭代器

待续