https://mp.weixin.qq.com/s/PR\_w2xkRVJFUAeTN\_1wwJQ

# 可重入读写锁ReentrantReadWriteLock的使用详解

ReentrantReadWriteLock是一把可重入读写锁，这篇文章主要是从使用的角度帮你理解，希望对你有帮助。

### 一、性质

**1、可重入**

如果你了解过synchronized关键字，一定知道他的可重入性，可重入就是同一个线程可以重复加锁，每次加锁的时候count值加1，每次释放锁的时候count减1，直到count为0，其他的线程才可以再次获取。

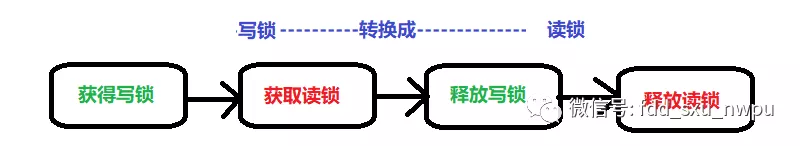
**2、读写分离**

我们知道，对于一个数据，不管是几个线程同时读都不会出现任何问题，但是写就不一样了，几个线程对同一个数据进行更改就可能会出现数据不一致的问题，因此想出了一个方法就是对数据加锁，这时候出现了一个问题：

线程写数据的时候加锁是为了确保数据的准确性，但是线程读数据的时候再加锁就会大大降低效率，这时候怎么办呢？那就对写数据和读数据分开，加上两把不同的锁，不仅保证了正确性，还能提高效率。

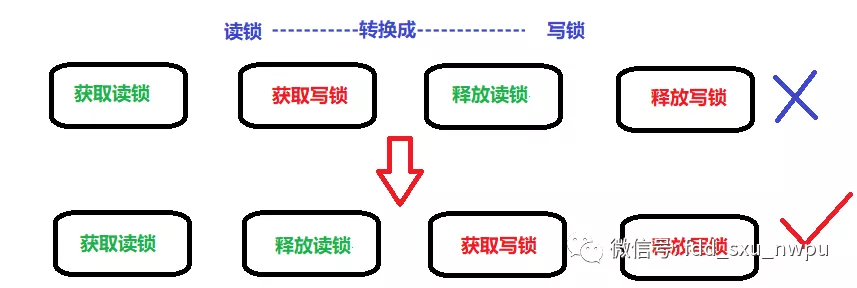
**3、可以锁降级**

线程获取写入锁后可以获取读取锁，然后释放写入锁，这样就从写入锁变成了读取锁，从而实现锁降级的特性。



**4、不可锁升级**

线程获取读锁是不能直接升级为写入锁的。需要释放所有读取锁，才可获取写锁，

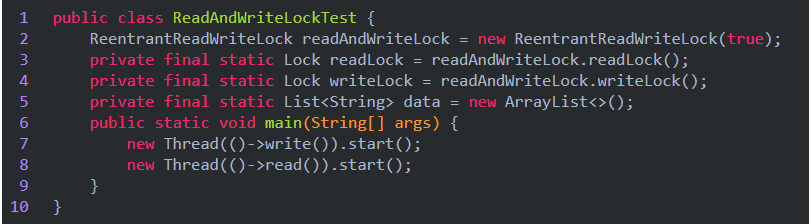


我们理解了上面的概念之后，接下来我们看看如何去使用。

### 二、使用

**1、基本使用**

使用很简单，那就是在写方法和读方法分开使用两把锁。



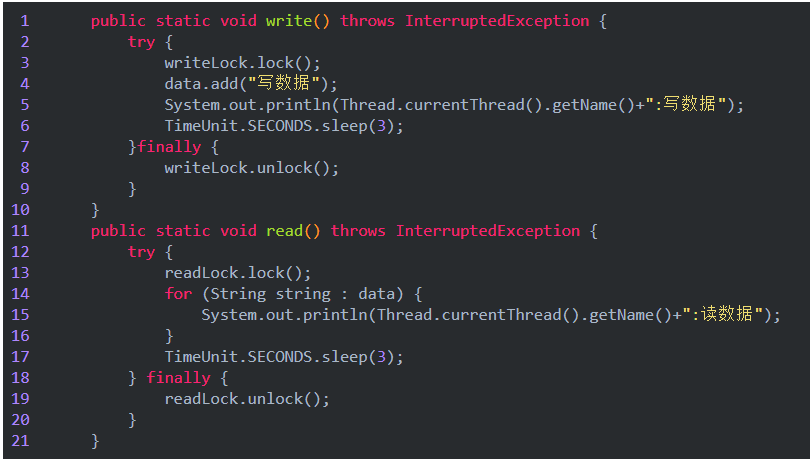
（1）首先定义一个ReentrantReadWriteLock。

（2）通过上面readAndWriteLock分别获取readLock和writeLock。

（3）接下来定义一个List，用于指代数据。

（4）最后在main方法中，使用两个线程分别调用不同的方法。

我们看看读方法和写方法是如何实现的吧。

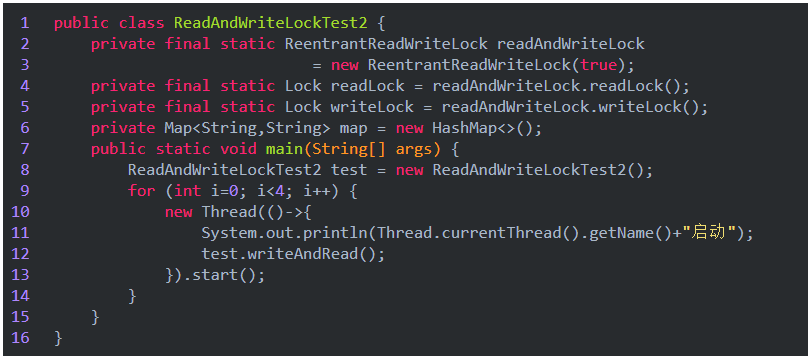


在写方法中：使用writeLock获取一把写锁，然后内部List写入数据，最后在finally中释放写锁。

在读方法中：使用readLock获取一把读锁，然后内部List读取数据，最后再finally中释放读锁。

**2、锁升级**

升级的意思就是，读锁在获取写锁之前，一定要先释放读锁。看个例子。这个例子对oracle官网的例子改动了一下



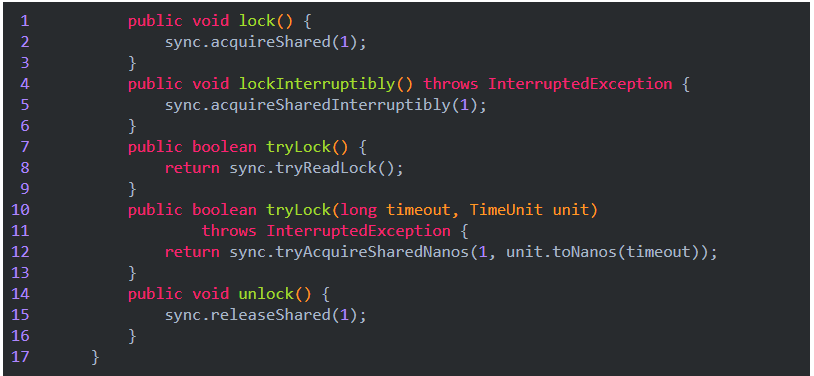
在这里我们还是首先获取读锁和写锁，然后在main方法中定义了4个线程，执行writeAndRead方法。我们看看这个方法如何实现。



在这个方法中，首先获取读锁，在获取写锁之前，一定要先释放读锁。这符合我们读写数据的一般规则。

**3、其他方法**

对于其他方面的使用，我们可以直接看读写锁的源码，其ReadLock是属于ReentrantReadWriteLock的内部类，在下一篇再说。一篇文章实在有点长。



**lock：获取一个锁。**

**lockInterruptibly：可中断的获取锁。**

**tryLock：阻塞式获取锁，没有获取就一直等待，直到成功。**

**tryLock(long timeout, TimeUnit unit)：获取一个锁，在指定的时间内获取。**

**unlock：释放一个锁。**

其实对于ReentrantReadWriteLock的使用还是比较常见的，比如说缓存机制中。