lock.lock和try catch finally之间的关系

在RocketMQ中存在一段这样的代码，

```
public RegisterBrokerResult registerBroker(
        final String clusterName,
        final String brokerAddr,
        final String brokerName,
        final long brokerId,
        final String haServerAddr,
        final TopicConfigSerializeWrapper topicConfigWrapper,
        final List<String> filterServerList,
        final Channel channel) {
        RegisterBrokerResult result = new RegisterBrokerResult();
        try {
            try {
                  /**
                 * 路由表的注册需要加写锁，防止并发修改RouteInfoManager中的路由表。
                 *
                 */
                this.lock.writeLock().lockInterruptibly();

                Set<String> brokerNames = this.clusterAddrTable.get(clusterName);
                //省略其他代码对路由表的修改
            } finally {
                this.lock.writeLock().unlock();
            }
        } catch (Exception e) {
            log.error("registerBroker Exception", e);
        }

        return result;
    }
```

对于lock和try catch的关系，我们一般会认为如下 使用方式，但是下面的这种使用方式存在一定的问题，就是 如果lockInterruptibly 没有获取到锁而抛出了异常，这会导致执行finally，但是在finally中我们主动释放锁。 释放锁的前提条件是你成功获取到了锁，如果没有获取到锁执行了unlock可能会抛出异常：unlock can throw exception indeed. The unlock method will throw an Exception if current thread didn't obtain the lock, which means some exception happens when call the lock method.
Thank you for your replay.

```
    public byte[] getAllTopicList() {
        TopicList topicList = new TopicList();
        try {
            this.lock.readLock().lockInterruptibly();
            topicList.getTopicList().addAll(this.topicQueueTable.keySet());
        } catch (Exception e) {
            log.error("getAllTopicList Exception", e);
        } finally {
            this.lock.readLock().unlock();
        }
        return topicList.encode();
    }
```

java官方文档对unlock的说明

```
unlock()
释放锁。
实现注意事项
Lock实现通常会对哪个线程可以释放锁施加限制(通常只有锁的持有者才能释放锁)，如果违反了限制，可能会抛出(未检查的)异常。任何限制和异常类型都必须由Lock实现记录。
```

因此正确的书写方式应该是这种，外部trycatch能够保证 unlock的异常被catch住。参考：https://github.com/apache/rocketmq/issues/2814

```
    public byte[] getAllTopicList() {
        TopicList topicList = new TopicList();
```

```
        try {
            try {
                this.lock.readLock().lockInterruptibly();
                topicList.getTopicList().addAll(this.topicQueueTable.keySet());
            } finally {
                this.lock.readLock().unlock();
            }
        } catch (Exception e) {
            log.error("getAllTopicList Exception", e);
        }

        return topicList.encode();
    }
```

最为标准的使用方式 [建议]lock.lock()应该放在try之前 #287 https://github.com/alibaba/p3c/issues/287

建议新增规则，lock.lock()应该放在try之前:

```
lock.lock();
try {
     doSomething();
} finally {
      lock.unlock();
}
```

如果按照下边写法，当lock.lock()报错时也会进入finally释放锁，根据Lock.unlock()文档，当非锁持有线程调用该方法时会抛出unchecked异常：

```
try {
      lock.lock();
      doSomething();
 } finally {
       lock.unlock();
 }
```

【强制】在try代码块之前调用Lock实现类的lock()方法，避免由于加锁失败，导致finally调用unlock()抛出异常。
说明：在lock方法中可能抛出uncheck异常，如果放在try代码块中，必然触发finally中的unlock方法的执行，它会调用AQS的tryRelease方法，（取决于具体实现类）。根据Lock接口中的unlock描述，对未加锁的对象解锁抛出unchecked异常，如：IllegalMonitorStateException，虽然都是加锁失败造成程序中断，但是真正加锁出错信息可能被后者覆盖。
反例：
Lock lock = new XxxLock();
preDo();
try {
// 无论加锁是否成功，unlock都会执行
lock.lock();
doSomething();
} finally {
lock.unlock();
}

9. 【强制】在使用阻塞等待获取锁的方式中，必须在 try 代码块之外，并且在加锁方法与 try 代码块之间没有任何可能抛出异常的方法调用，避免加锁成功后，在 finally 中无法解锁。

说明一：如果在 lock 方法与 try 代码块之间的方法调用抛出异常，那么无法解锁，造成其它线程无法成功获取锁。

说明二：如果 lock 方法在 try 代码块之内，可能由于其它方法抛出异常，导致在 finally 代码块中，unlock 对未加锁的对象解锁，它会调用 AQS 的 tryRelease 方法（取决于具体实现类），抛出 IllegalMonitorStateException 异常。

说明三：在 Lock 对象的 lock 方法实现中可能抛出 unchecked 异常，产生的后果与说明二相同。

正例：

```
Lock lock = new XxxLock();
// ...
lock.lock();
try {
    doSomething();
    doOthers();
} finally {
    lock.unlock();
}
```

反例：

```
Lock lock = new XxxLock();
// ...
```



```
try {
    // 如果此处抛出异常，则直接执行 finally 代码块
    doSomething();
    // 无论加锁是否成功，finally 代码块都会执行
    lock.lock();
    doOthers();
} finally {
    lock.unlock();
}
```

10. 【强制】在使用尝试机制来获取锁的方式中，进入业务代码块之前，必须先判断当前线程是否持有锁。锁的释放规则与锁的阻塞等待方式相同。

说明：Lock 对象的 unlock 方法在执行时，它会调用 AQS 的 tryRelease 方法（取决于具体实现类），如果当前线程不持有锁，则抛出 IllegalMonitorStateException 异常。

正例：

```
Lock lock = new XxxLock();
// ...
boolean isLocked = lock.tryLock();
if (isLocked) {
    try {
        doSomething();
        doOthers();
    } finally {
        lock.unlock();
    }
}
```