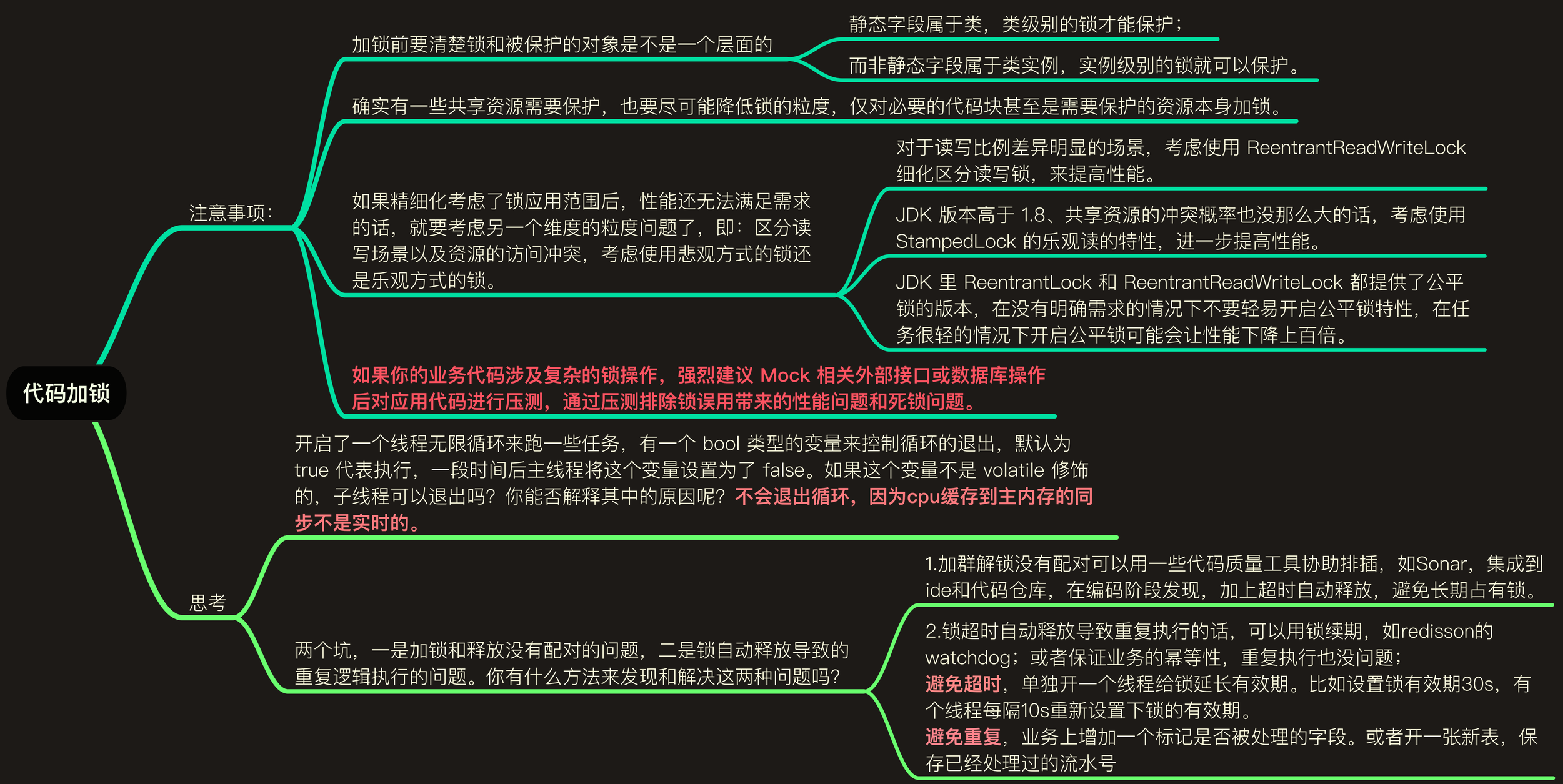
代码加锁
注意事项：
加锁前要清楚锁和被保护的对象是不是一个层面的
静态字段属于类，类级别的锁才能保护；
而非静态字段属于类实例，实例级别的锁就可以保护。
确实有一些共享资源需要保护，也要尽可能降低锁的粒度，仅对必要的代码块甚至是需要保护的资源本身加锁。
如果精细化考虑了锁应用范围后，性能还无法满足需求的话，就要考虑另一个维度的粒度问题了，即：区分读写场景以及资源的访问冲突，考虑使用悲观方式的锁还是乐观方式的锁。
对于读写比例差异明显的场景，考虑使用 ReentrantReadWriteLock 细化区分读写锁，来提高性能。
JDK 版本高于 1.8、共享资源的冲突概率也没那么大的话，考虑使用 StampedLock 的乐观读的特性，进一步提高性能。
JDK 里 ReentrantLock 和 ReentrantReadWriteLock 都提供了公平锁的版本，在没有明确需求的情况下不要轻易开启公平锁特性，在任务很轻的情况下开启公平锁可能会让性能下降上百倍。
如果你的业务代码涉及复杂的锁操作，强烈建议 Mock 相关外部接口或数据库操作后对应用代码进行压测，通过压测排除锁误用带来的性能问题和死锁问题。
思考
开启了一个线程无限循环来跑一些任务，有一个 bool 类型的变量来控制循环的退出，默认为 true 代表执行，一段时间后主线程将这个变量设置为了 false。如果这个变量不是 volatile 修饰的，子线程可以退出吗？你能否解释其中的原因呢？不会退出循环，因为cpu缓存到主内存的同步不是实时的。
两个坑，一是加锁和释放没有配对的问题，二是锁自动释放导致的重复逻辑执行的问题。你有什么方法来发现和解决这两种问题吗？
1.加群解锁没有配对可以用一些代码质量工具协助排插，如Sonar，集成到ide和代码仓库，在编码阶段发现，加上超时自动释放，避免长期占有锁。
2.锁超时自动释放导致重复执行的话，可以用锁续期，如redisson的watchdog；或者保证业务的幂等性，重复执行也没问题；
避免超时，单独开一个线程给锁延长有效期。比如设置锁有效期30s，有个线程每隔10s重新设置下锁的有效期。
避免重复，业务上增加一个标记是否被处理的字段。或者开一张新表，保存已经处理过的流水号