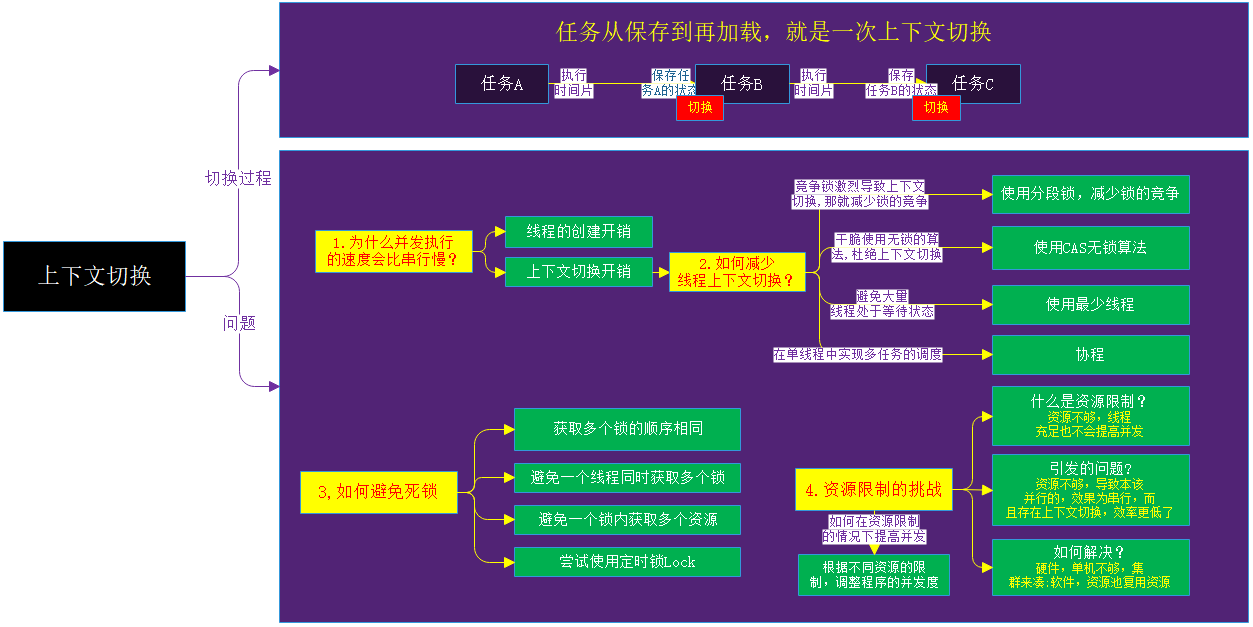
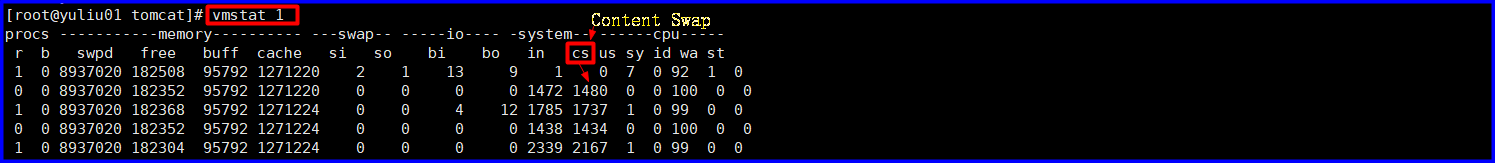
# 并发编程的挑战



## 测试上下文切换次数和时长



## 减少上下文切换-使用最少线程实战

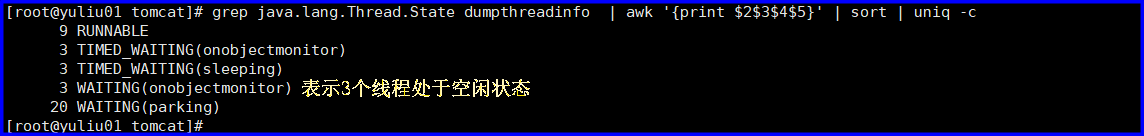
### 用jstack命令dump线程信息

jstack 15436 > /home/znmobile/tomcat/dumpthreadinfo



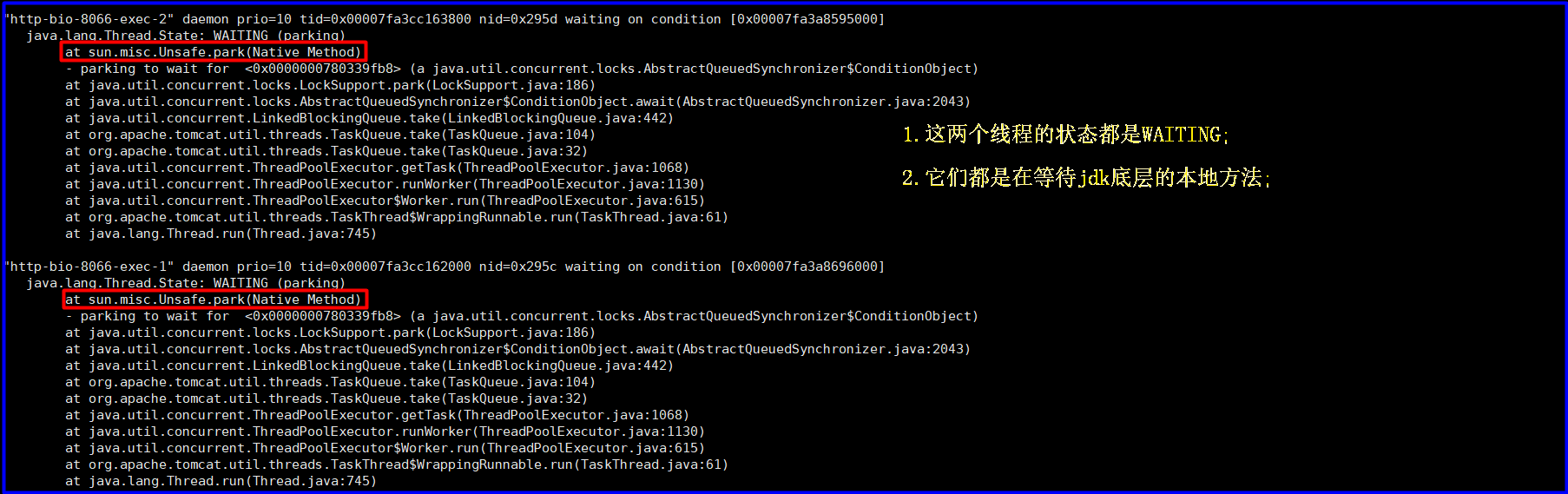
### 统计所有线程都在什么状态

grep java.lang.Thread.State dumpthreadinfo | awk '{print $2$3$4$5}' | sort | uniq –c

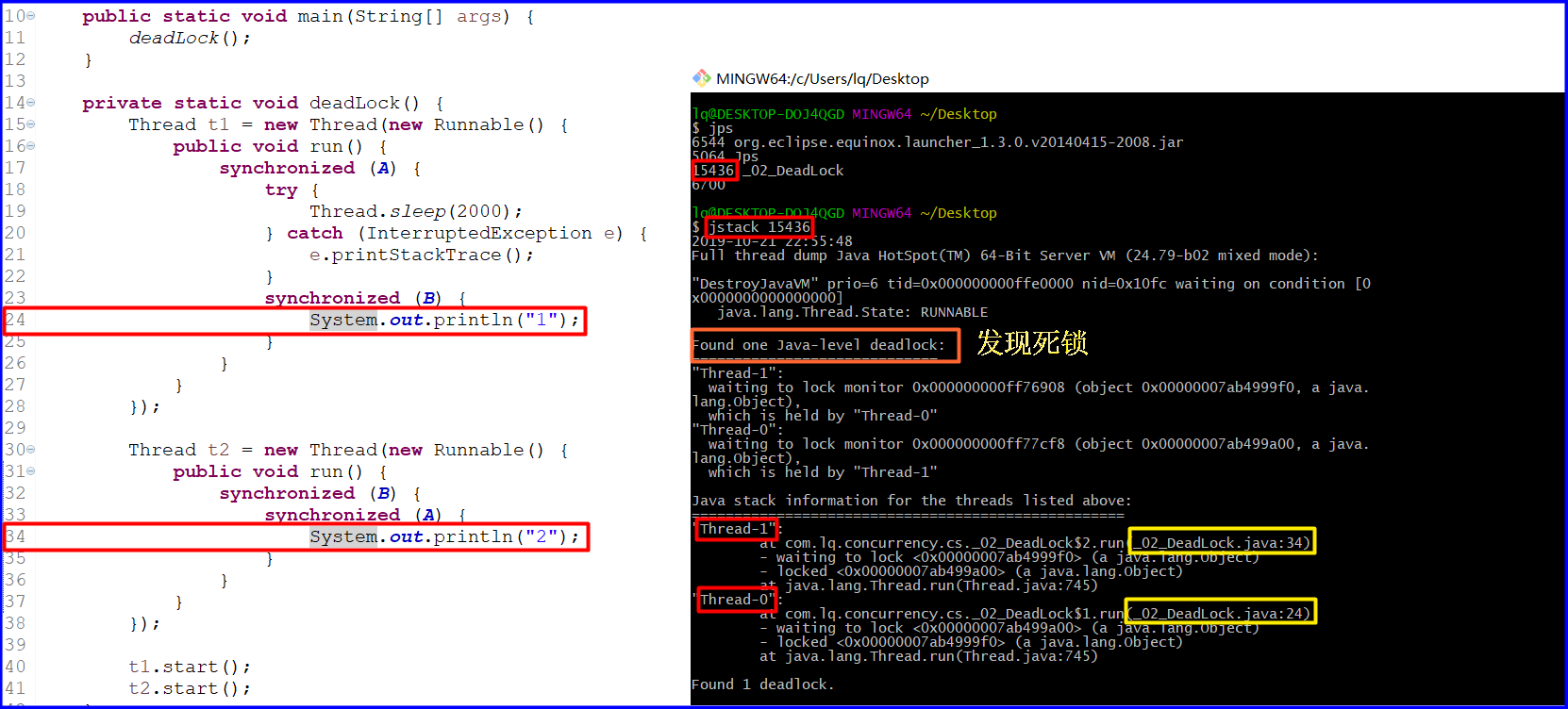


### 查看WAITING的线程在干什么

可以通过减少配置文件中，默认创建的线程数，来避免大量空闲线程存在。

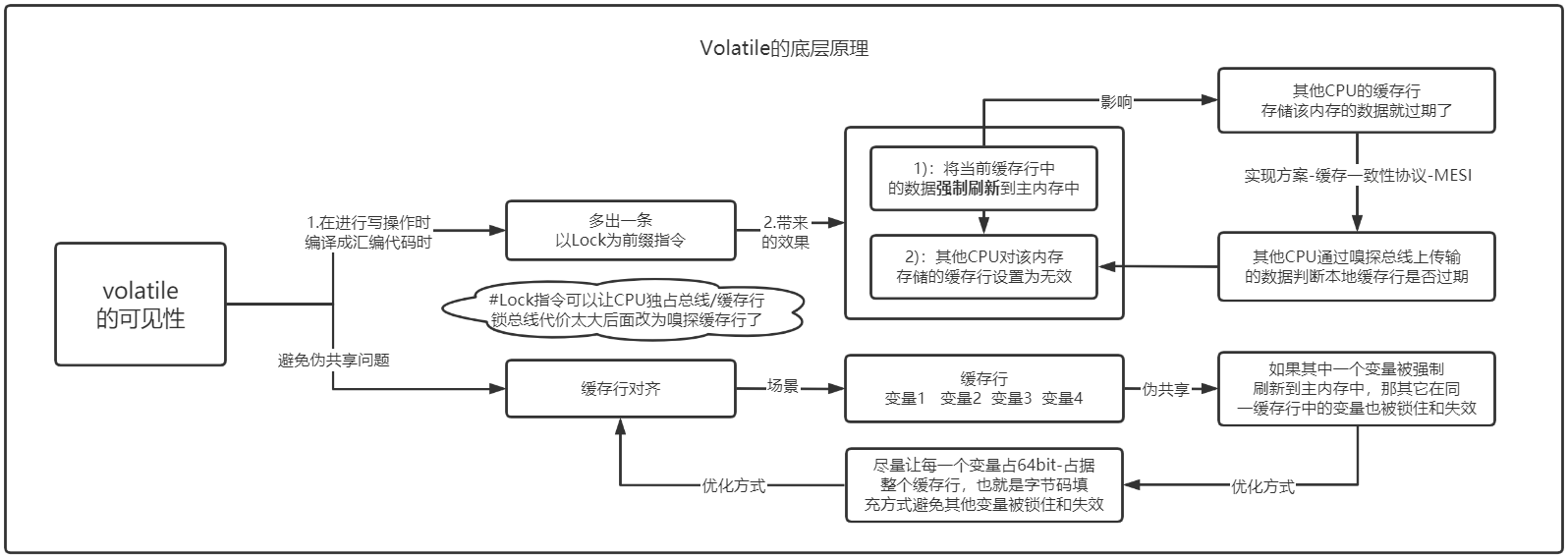


## 代码模拟死锁并观察是哪里发生死锁

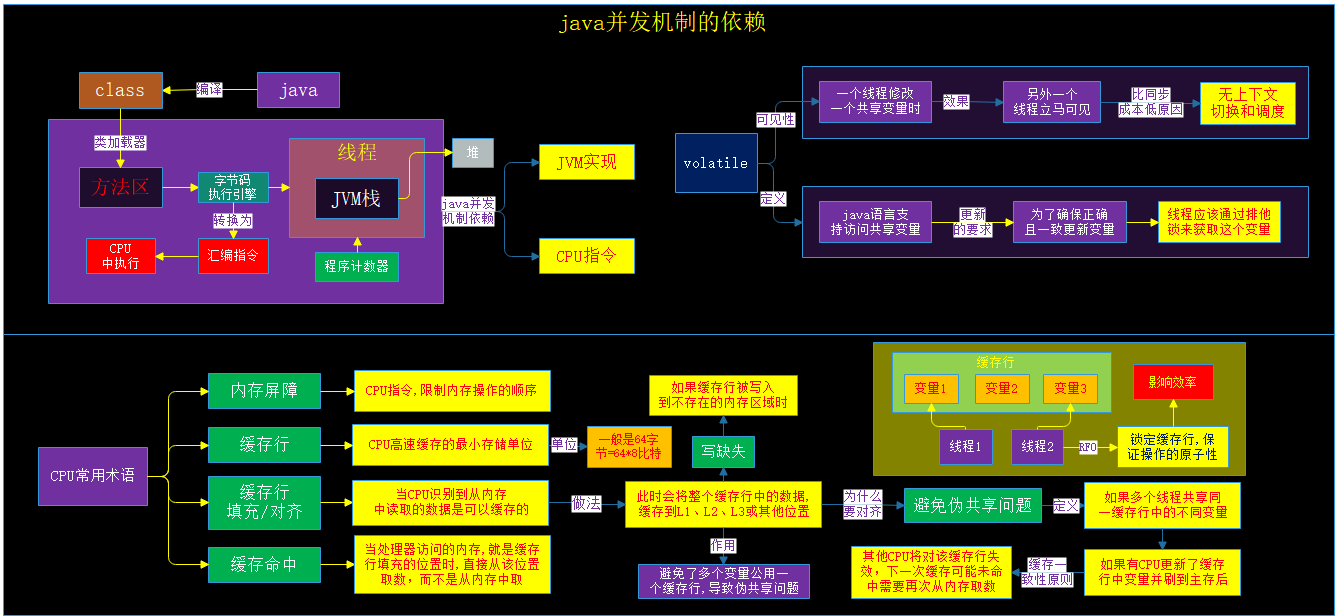


# Java并发机制的底层实现原理

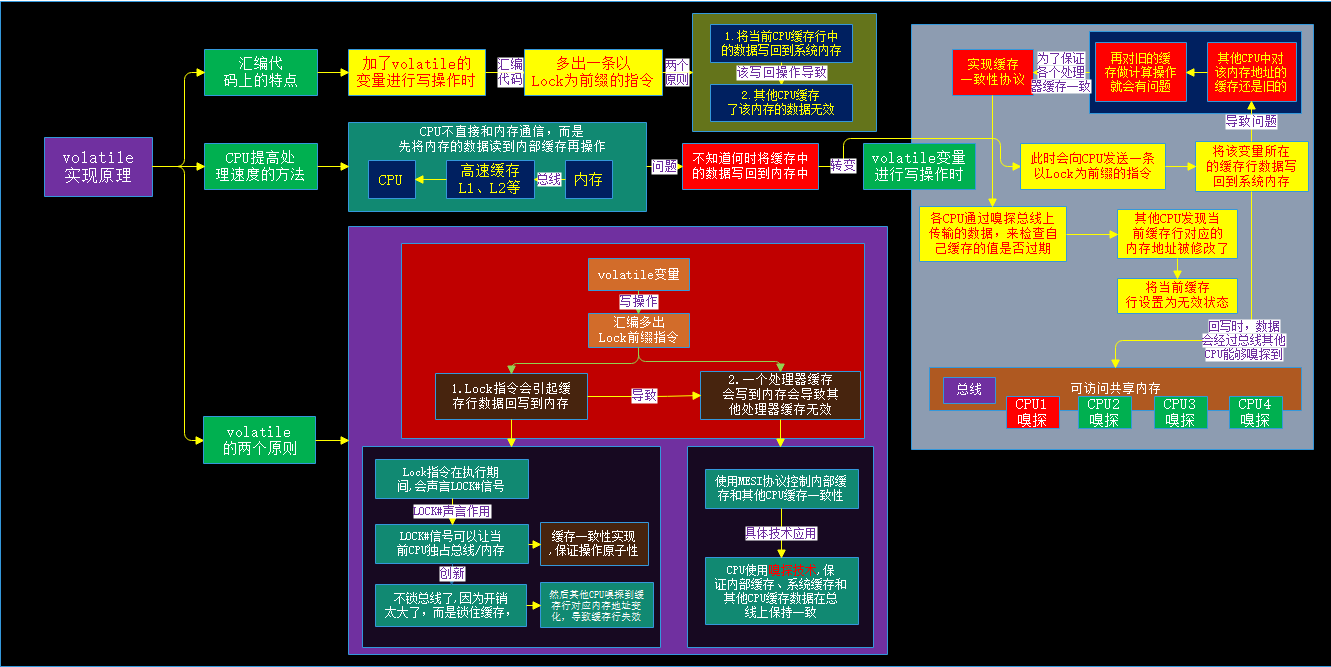
## volatile的原理和优化



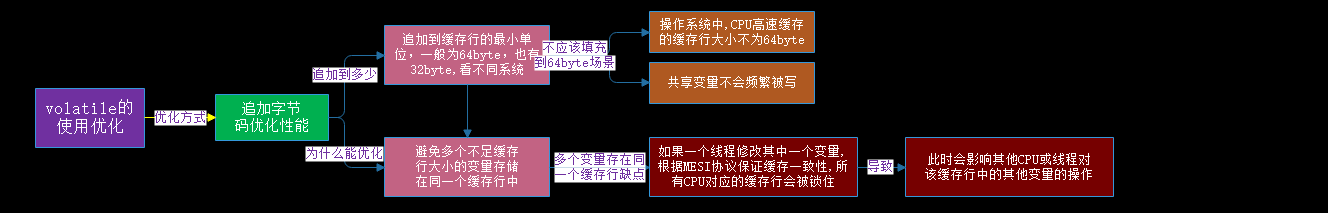
### 并发依赖和术语



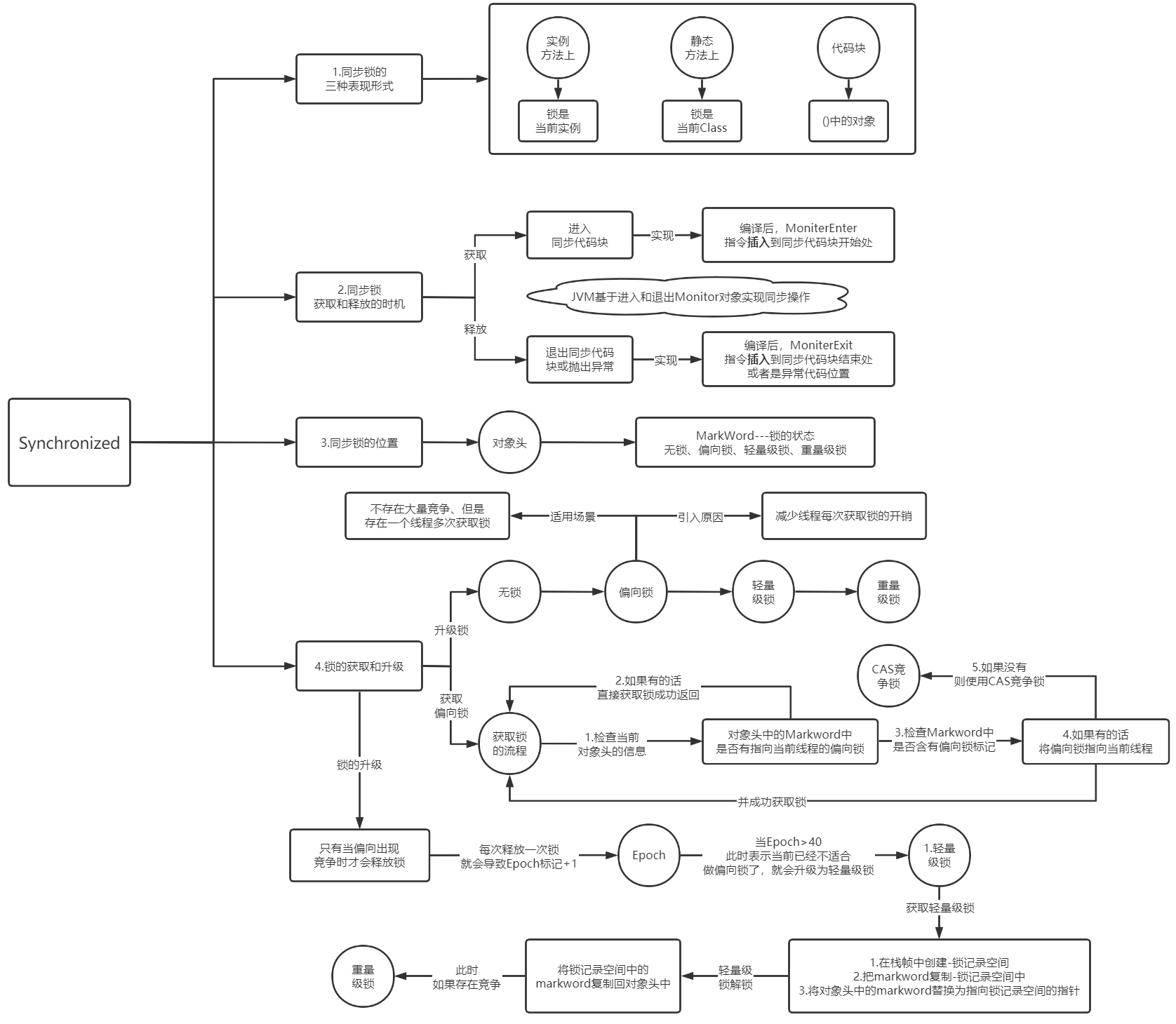
### volatile的原理



### volatile的优化



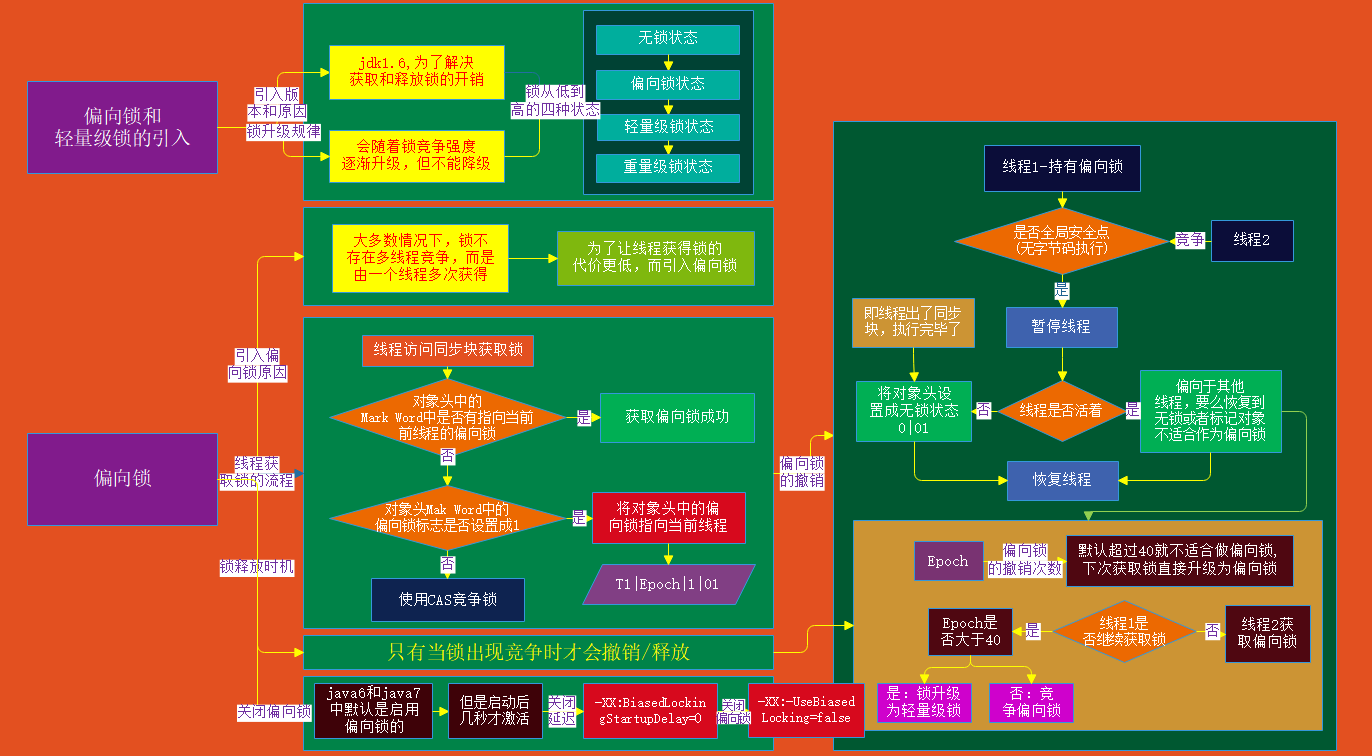
## synchronized的原理

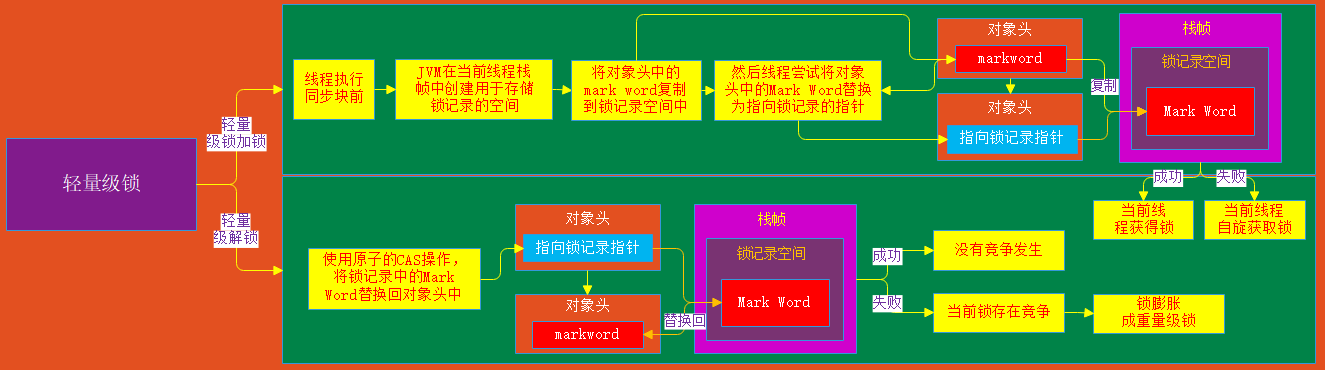


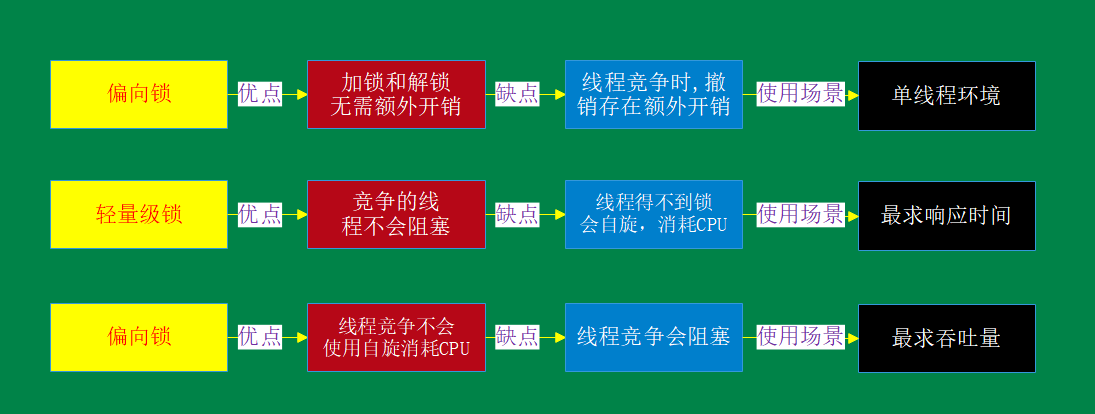
### 对象头



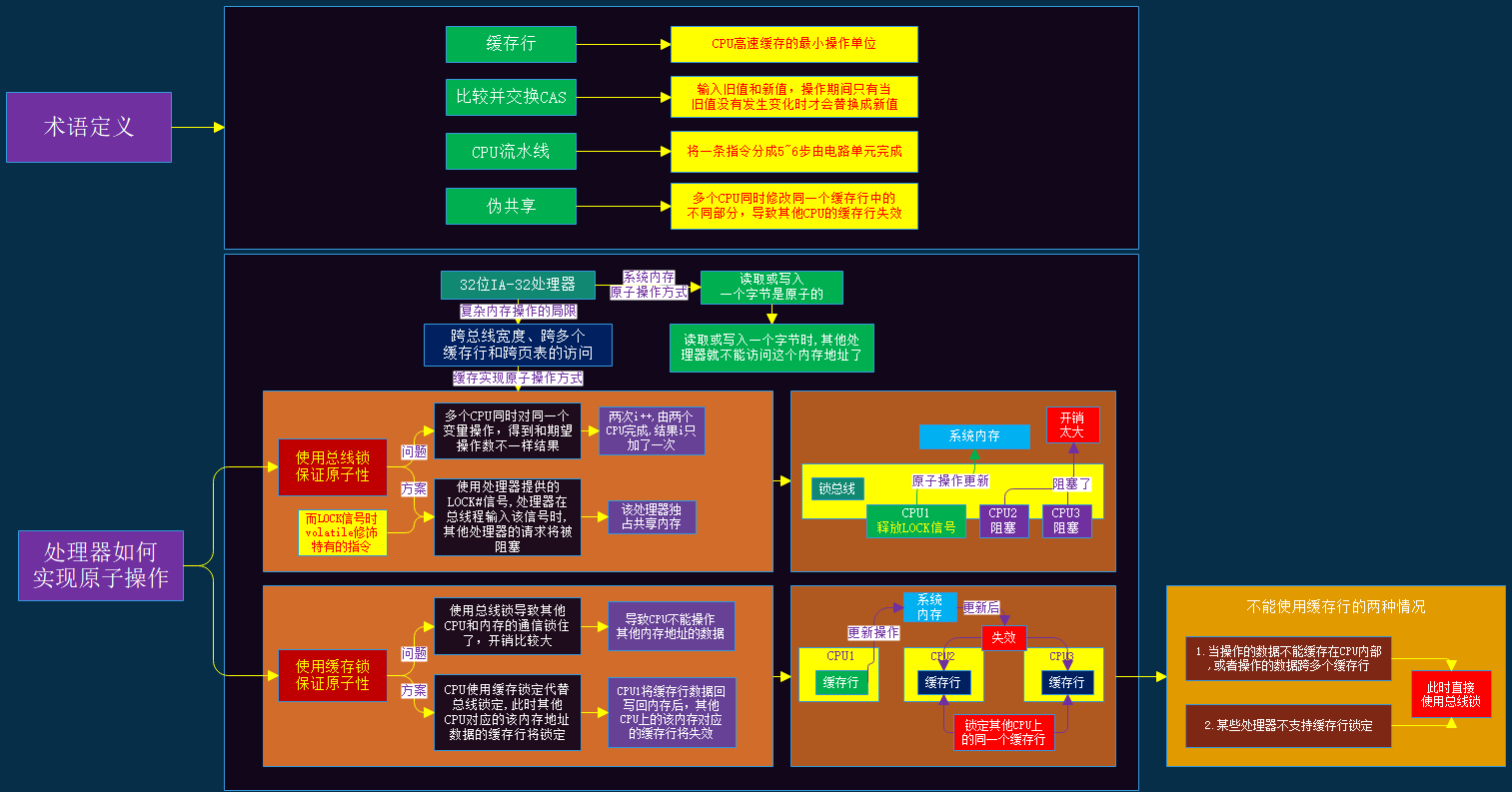
### 锁的升级

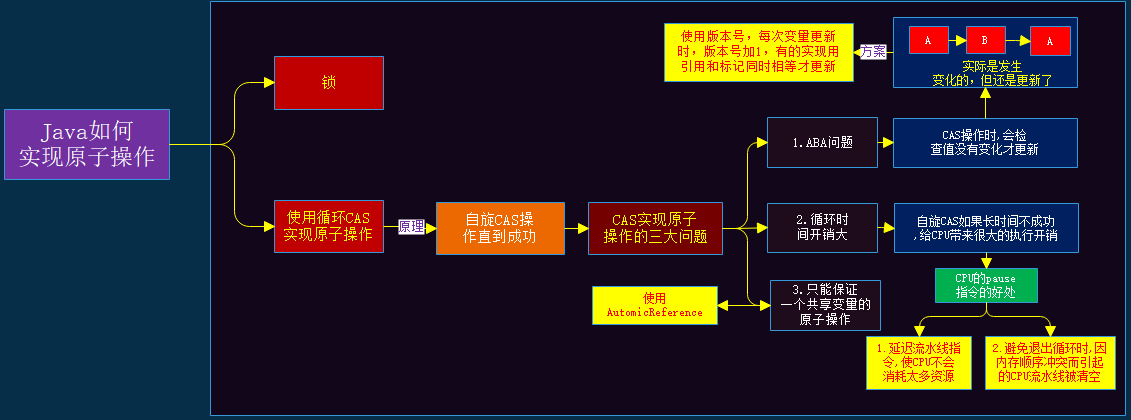






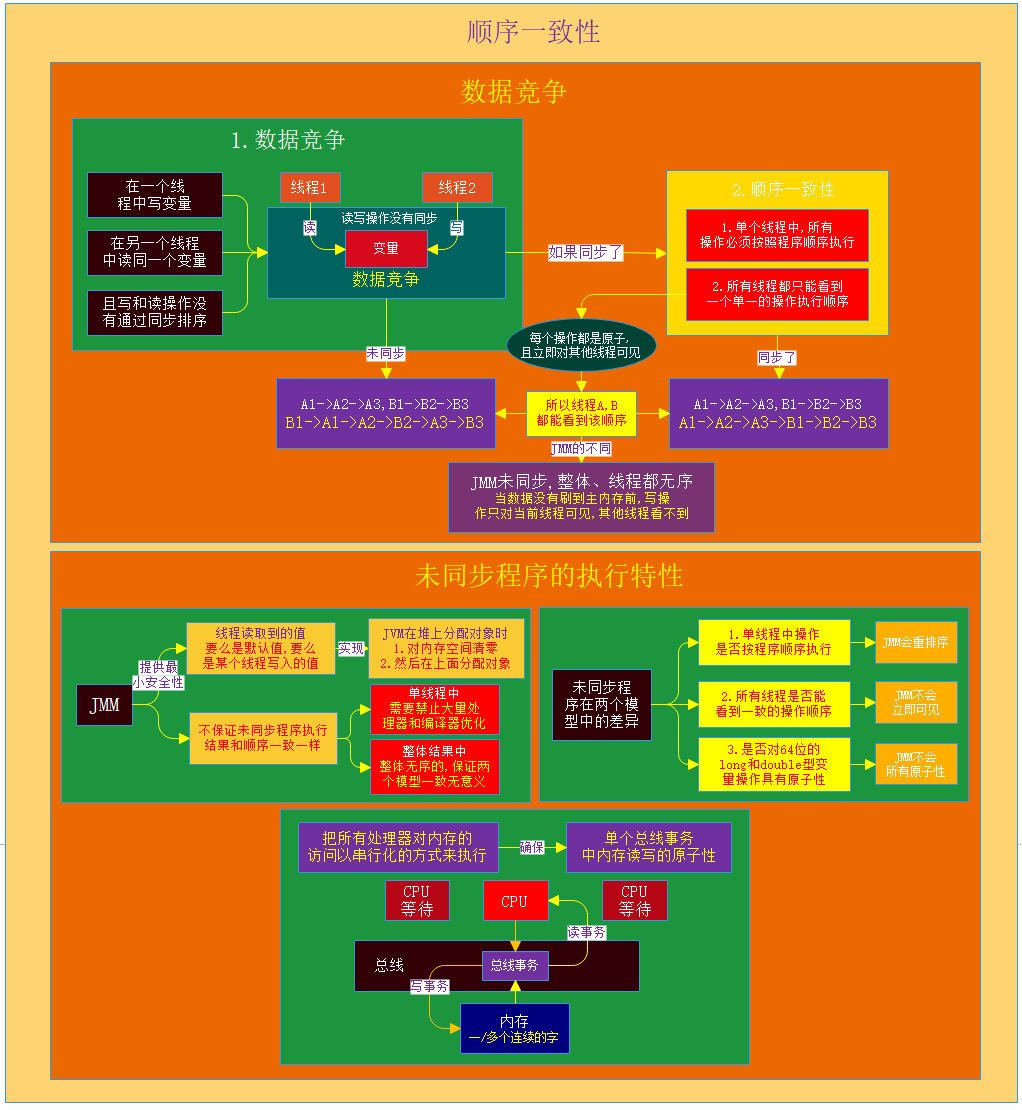
## 原子操作的实现原理

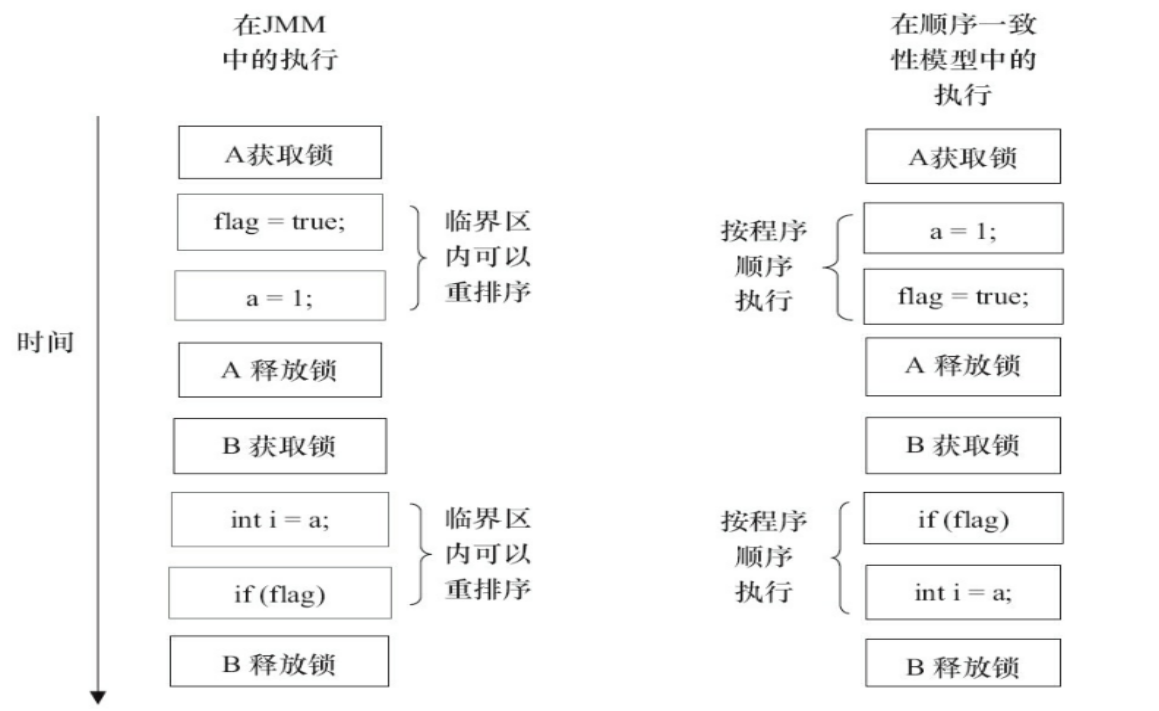




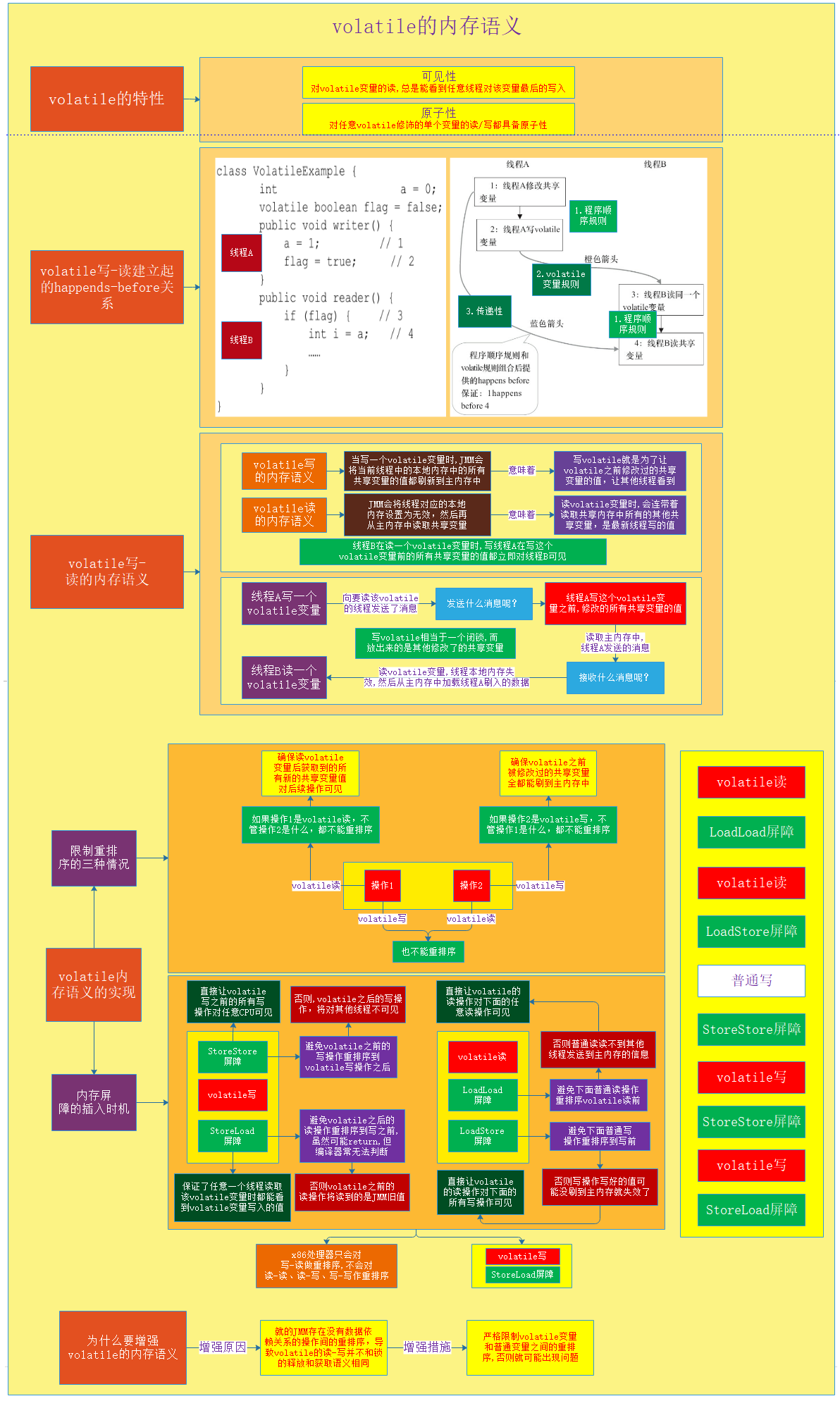
# Java内存模型

## 顺序一致性

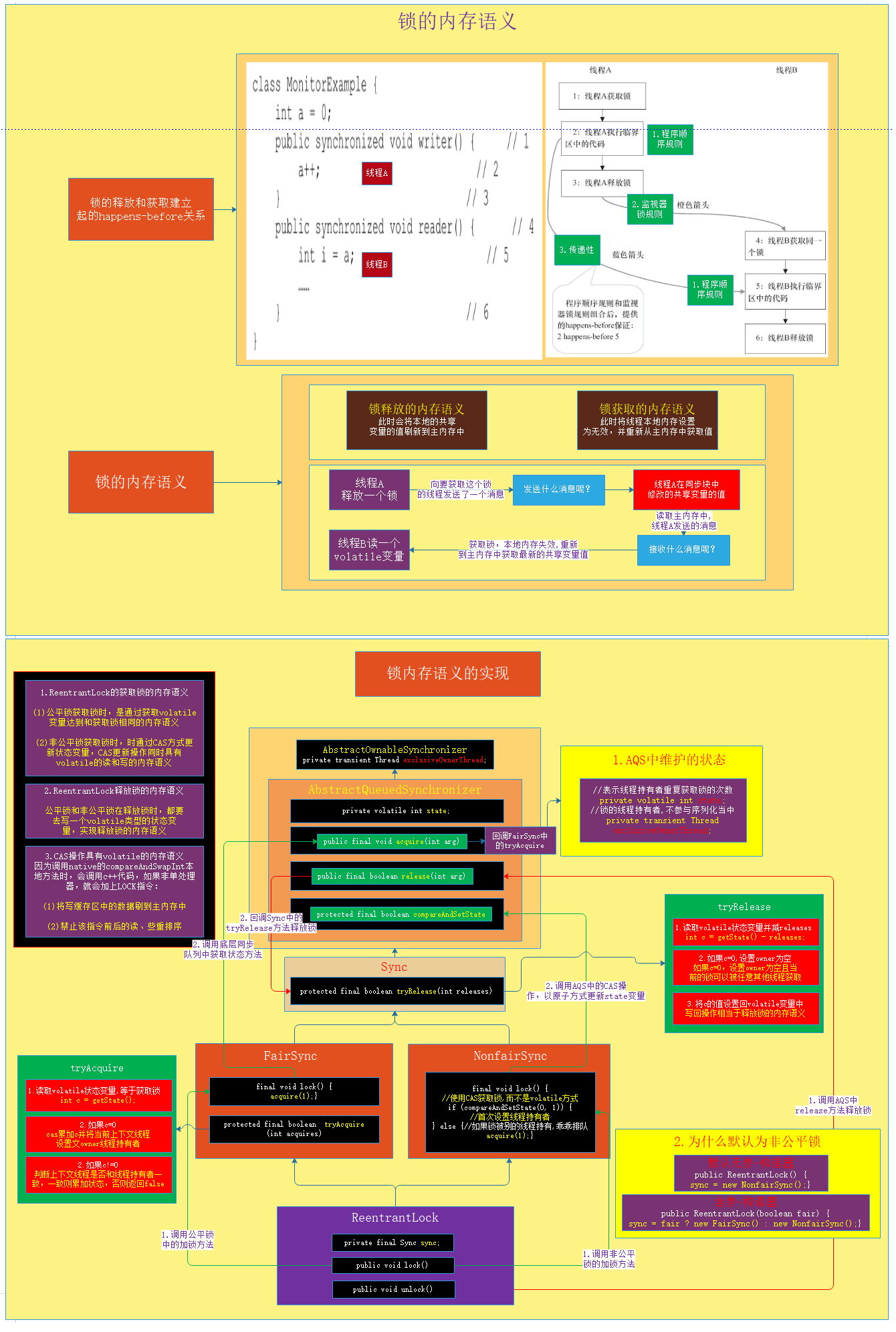




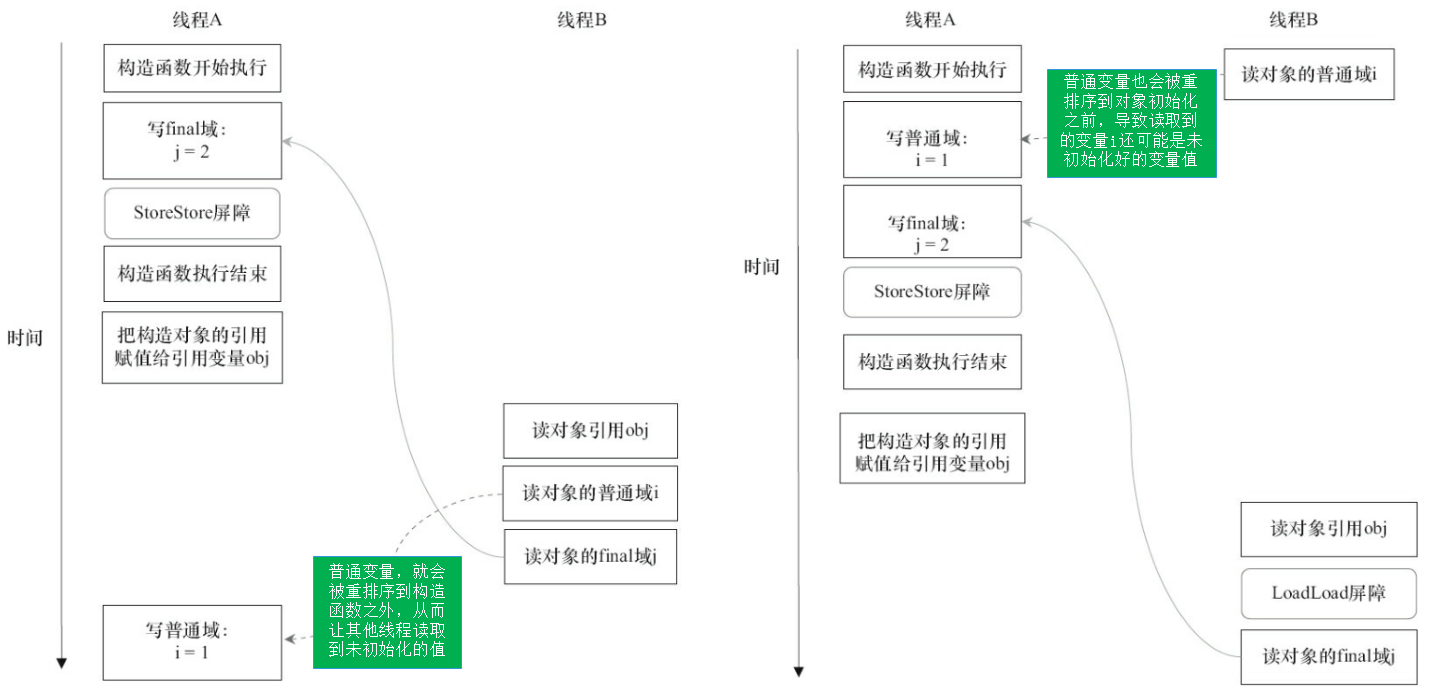
## volatile的内存语义

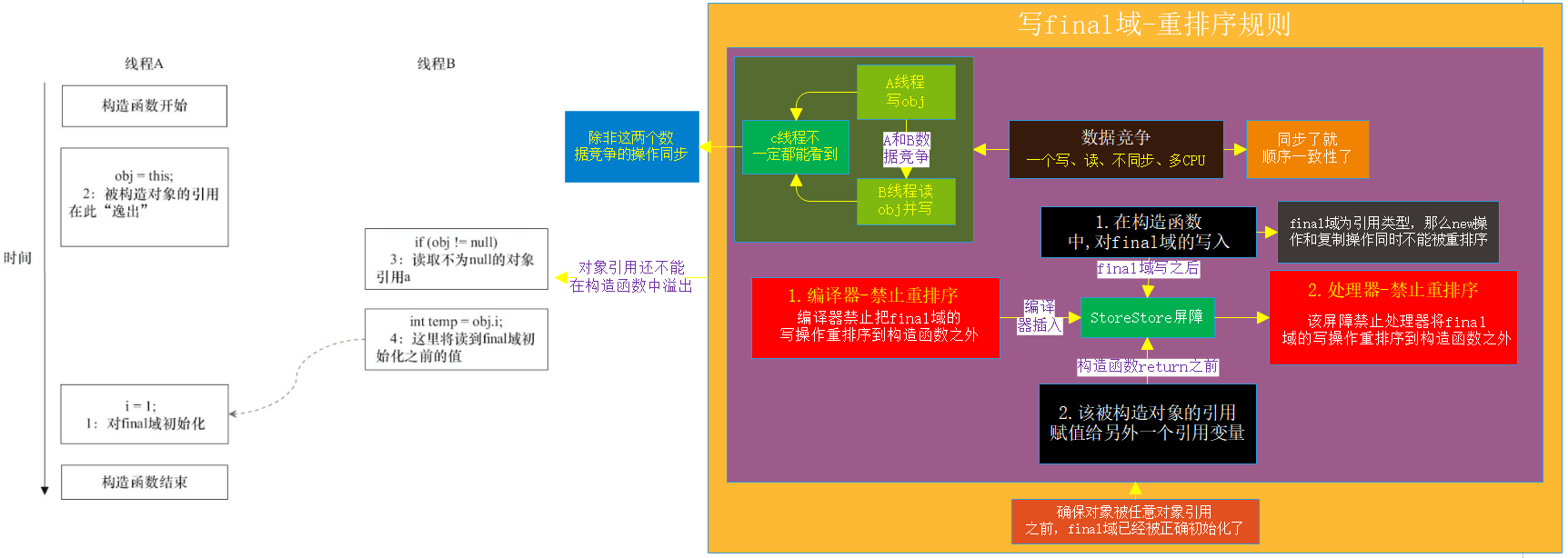


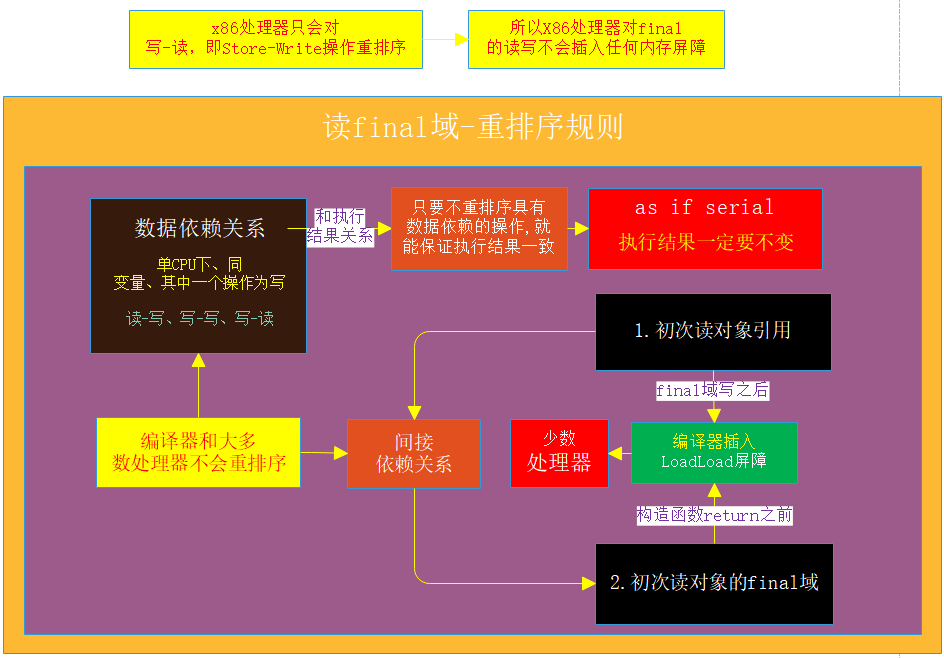
## 锁的内存语义



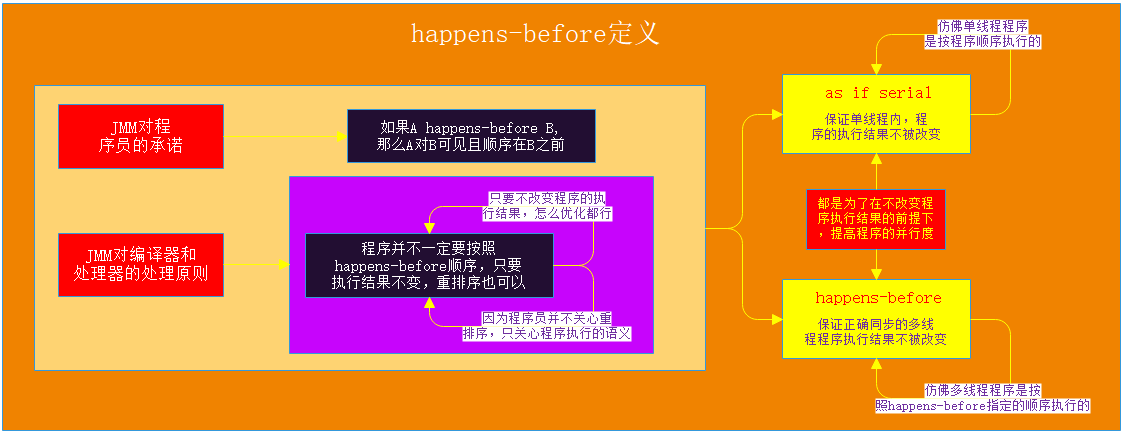
## final域的内存语义

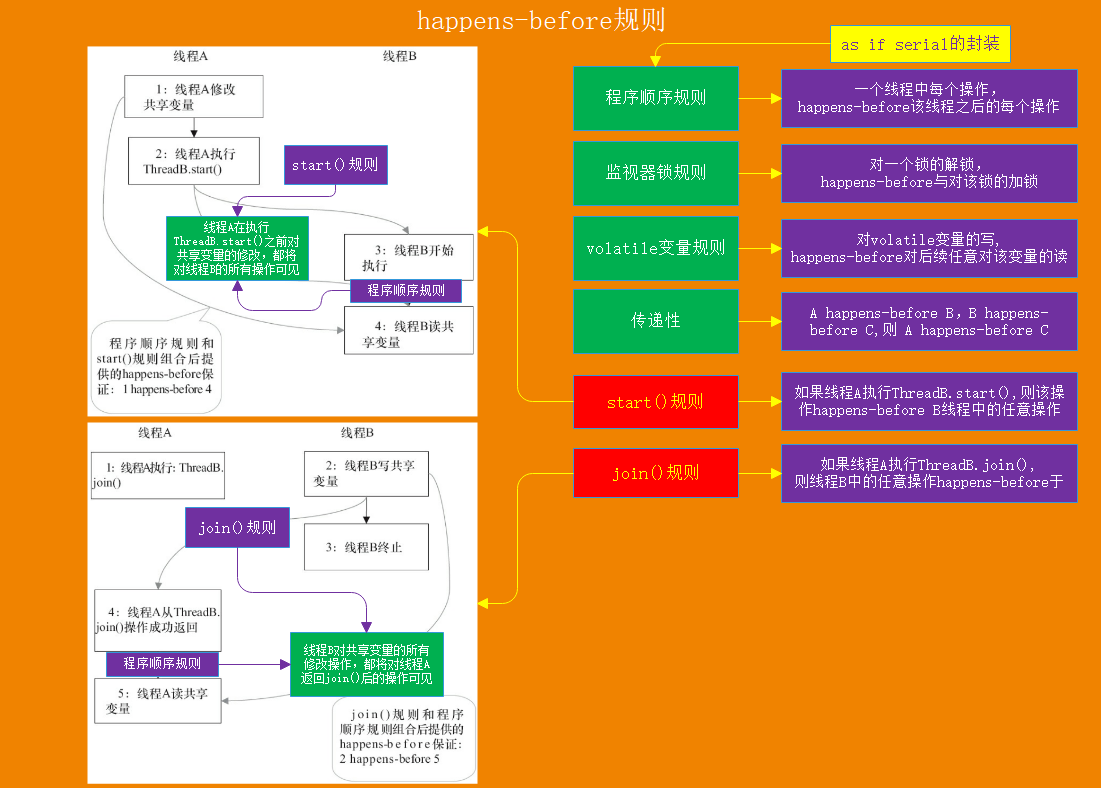






## happens-before





# Java并发编程的基础