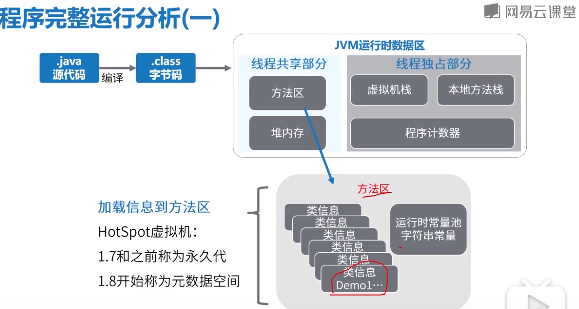
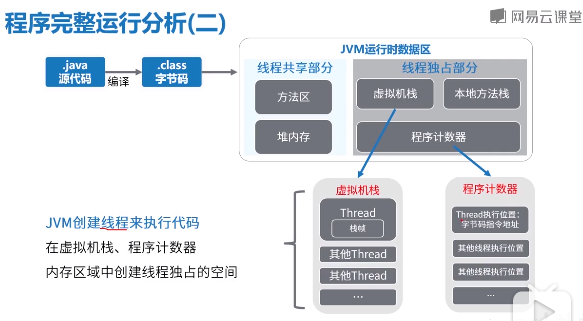
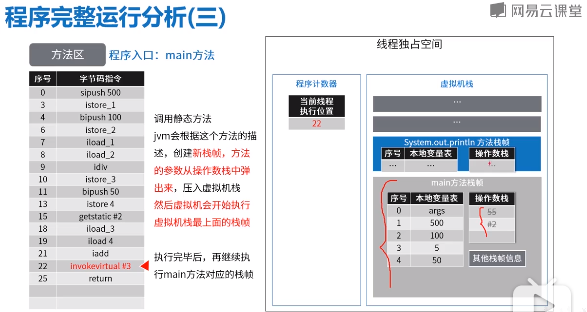
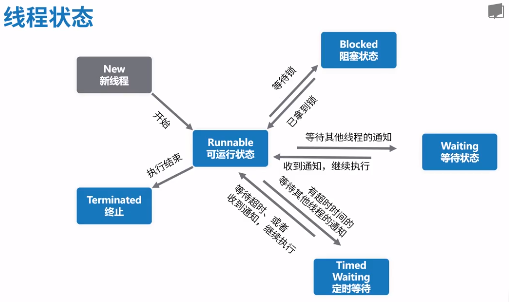
## java程序运行原理分析



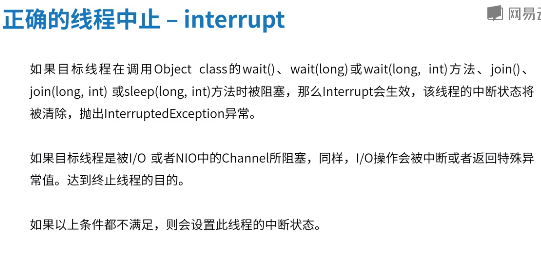


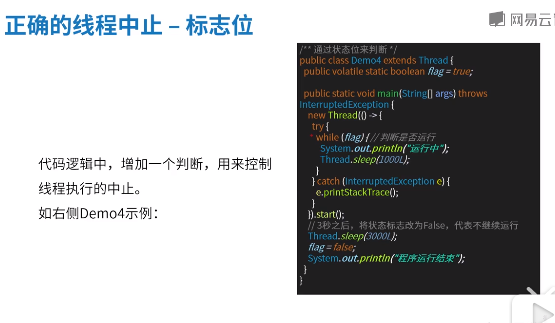


## 线程状态

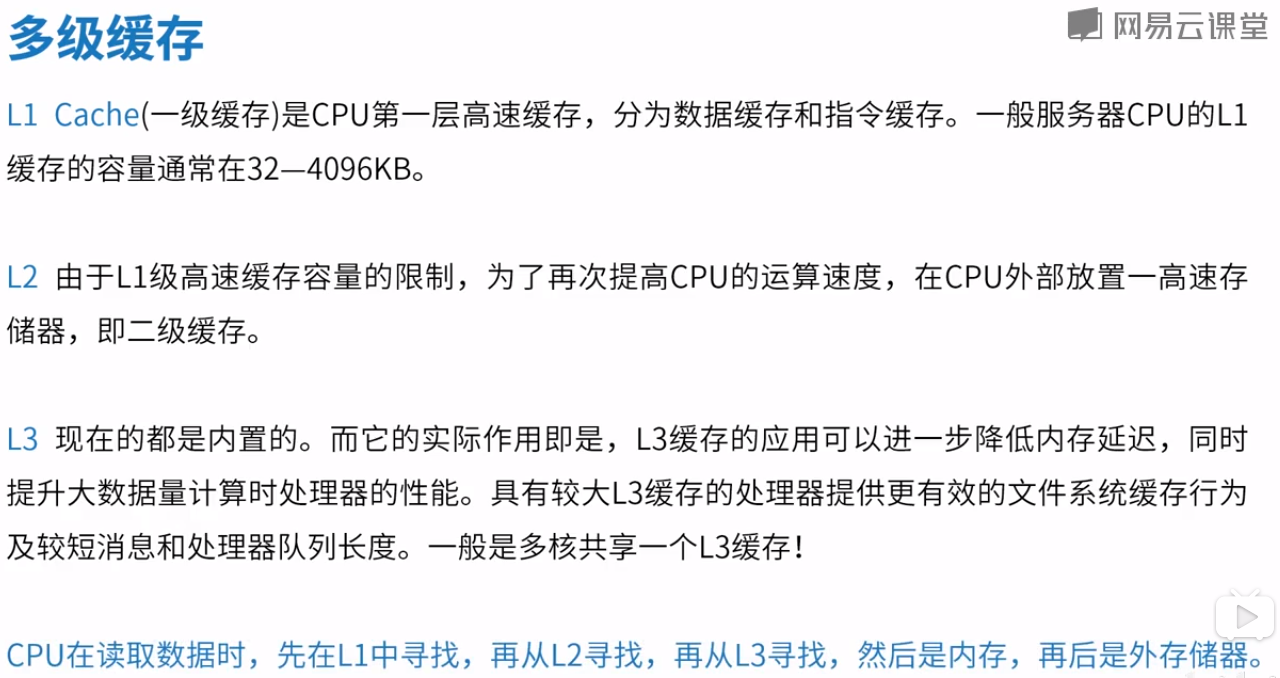


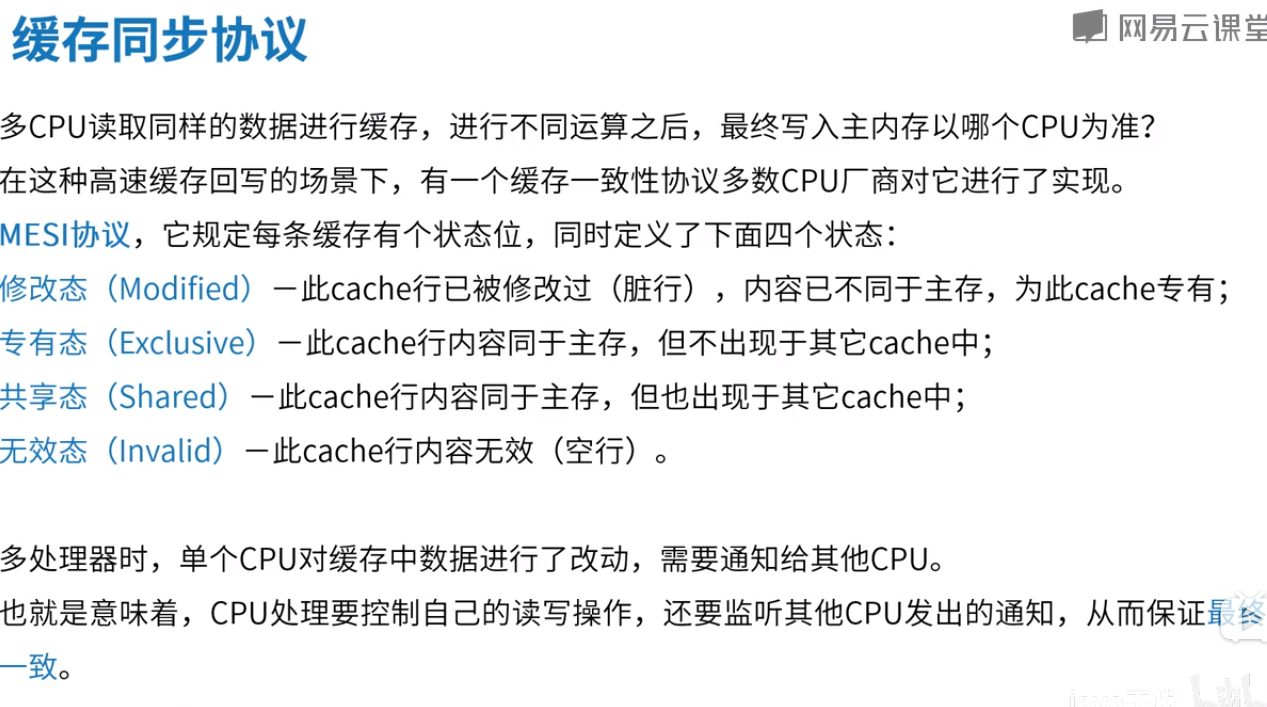
## 线程终止

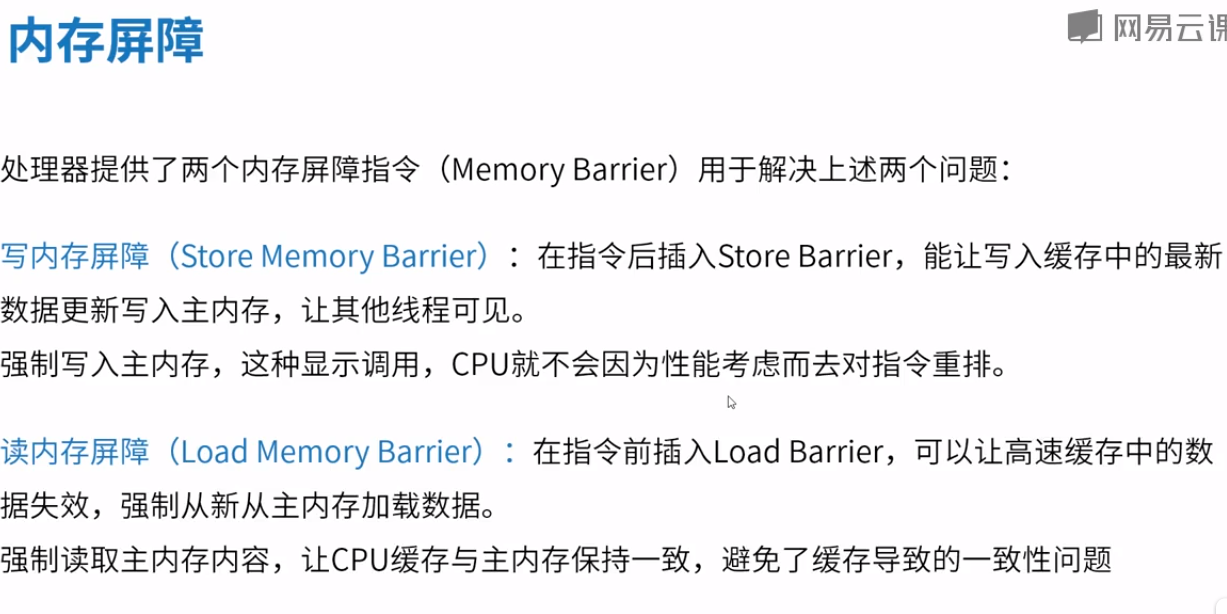




## CPU多级缓存与内存屏障







## 线程通信

##### 线程通信方式

* + 1. 文件共享
    2. 网络共享
    3. 共享变量
    4. **Jdk提供的线程协调API**

##### 文件共享/网络共享

线程1写数据，线程2读数据，实现共享

##### 变量共享

共享变量让不同线程写入读取实现数据交互

##### 线程协作API

* + 1. Suspend & resume（被弃用）弃用原因：容易死锁：
       1. Suspend挂起之后不会释放锁。
       2. Suspend比resume后执行。如果suspend未及时执行，resume先执行，就会死锁。
    2. Wait & notify/notifyAll
       1. Wait 等待后会释放锁
       2. 但是顺序不对同样会出现死锁。如果notify在wait之前，就会永远等待，死锁。
    3. Park & unpark机制，许可机制，令牌机制
       1. Park不会释放锁，因此也可能出现死锁状态
       2. 顺序无所谓，因为是许可证机制，先有跟后有都能执行，不会死锁。

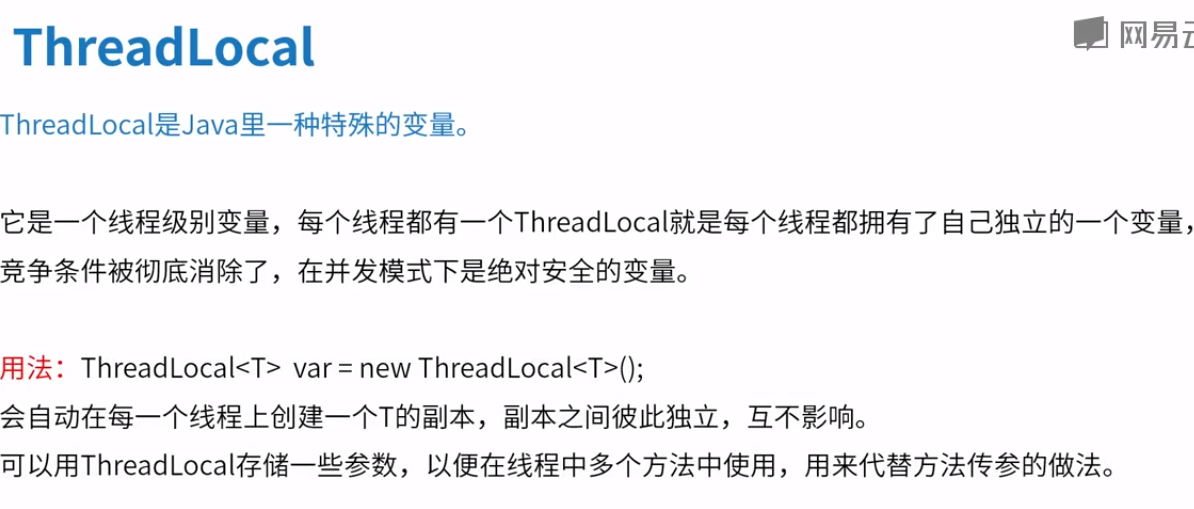
##### 防止伪唤醒

* + 1. 应该在循环中检查等待条件，防止伪唤醒或错误警报导致跳出等待。

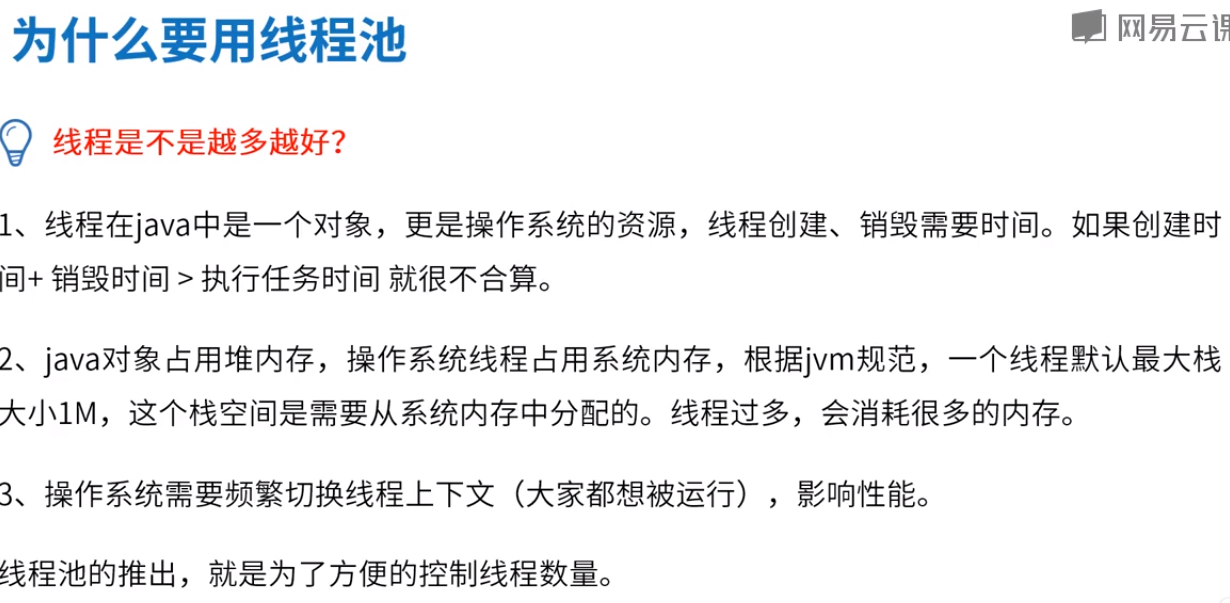
## 线程封闭

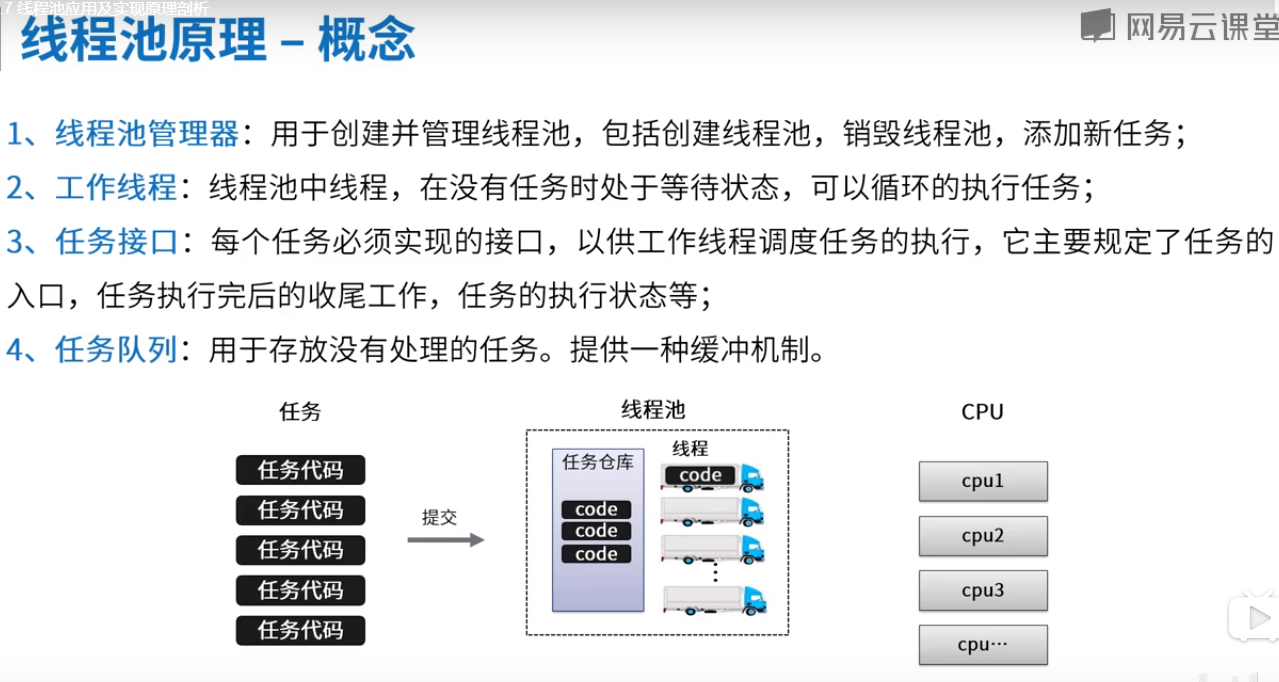
将数据封闭在线程中而避免使用同步的技术叫线程封闭。无共享数据。

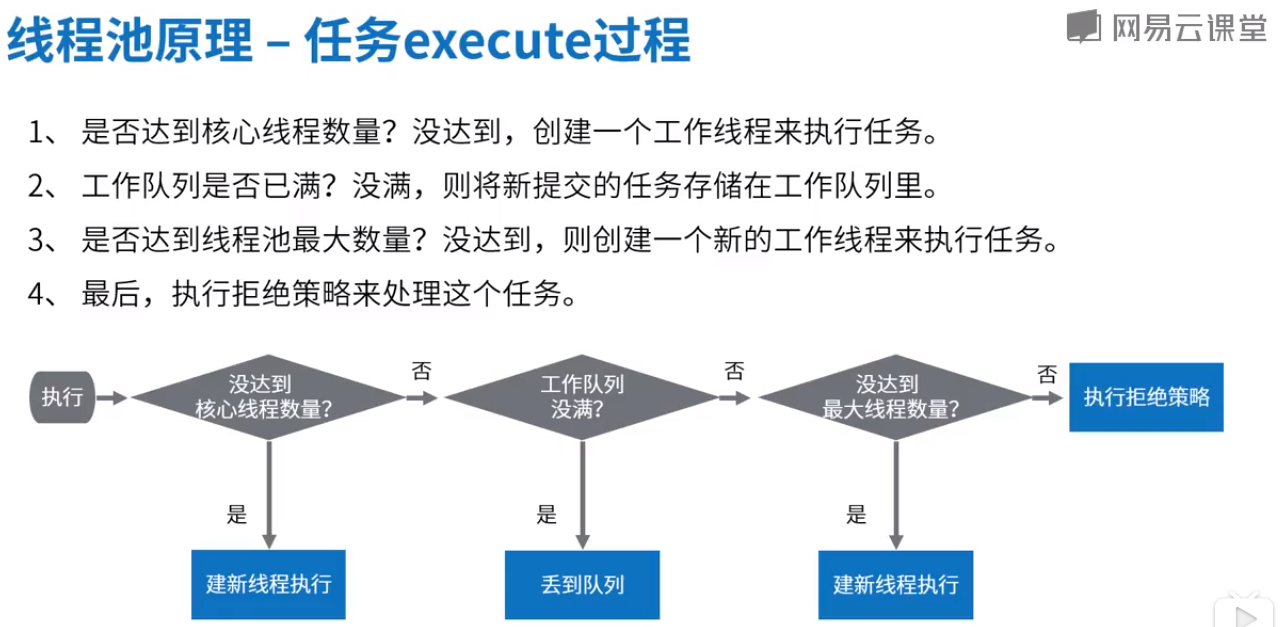
具体：ThreadLocal 与局部变量（栈封闭）

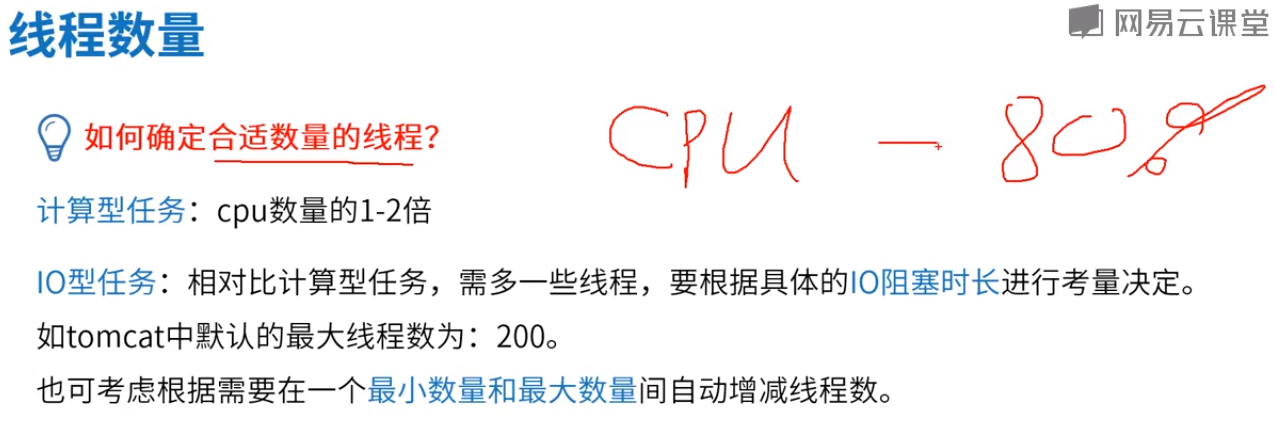


## 线程池

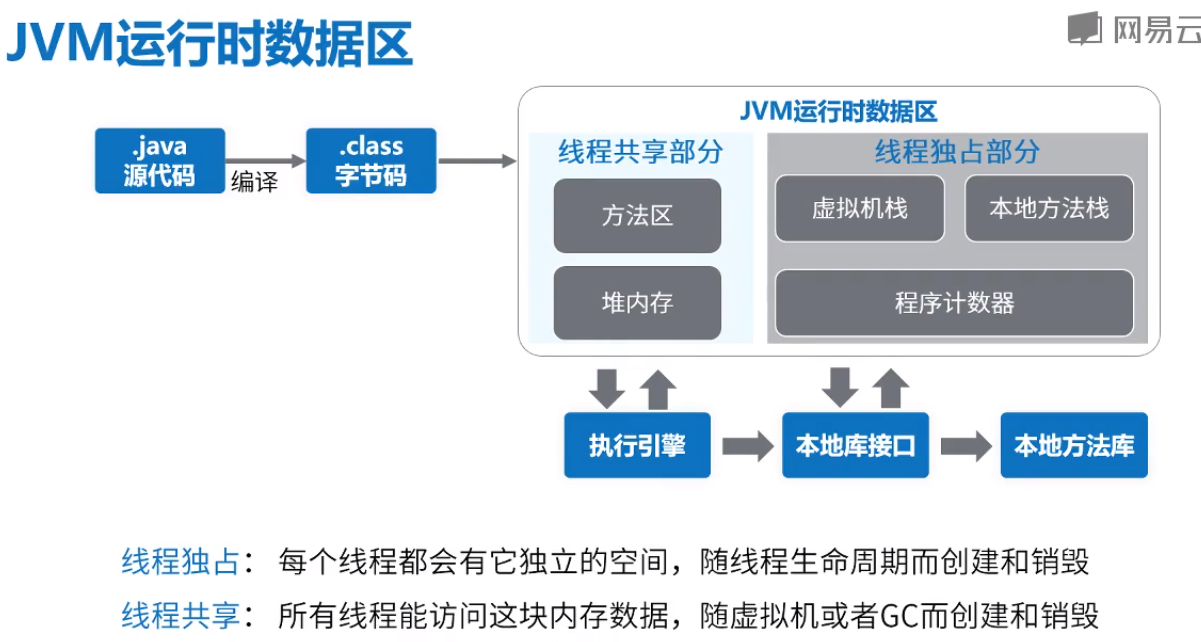








## JVM内存模型



## 线程安全之原子操作

##### CAS(compare and swap)

CAS操作需要两个数值，旧值A和新值B，先比较如果旧值没有发生变化，则交换为新值，否则不交换。

**注意：CAS仅仅针对单个变量的操作**

Java中的sun.misc.Unsafe类提供compareAndSwapInt(),compareAndSwapLong()等。JDK1.9以后提供VarHandle类，该类提供compareAndSet()方法。

缺点：由于要不断尝试直到交换好新值，所以会消耗不必要的cpu时间。

原子性操作类 java.util.concurrent.atomic包，内部原理就是CAS实现，AtomicInteger, AtomicLong, AtomicIntegerArray,AtomicReference, 自增更新器LongAdder, DoubleAdder等

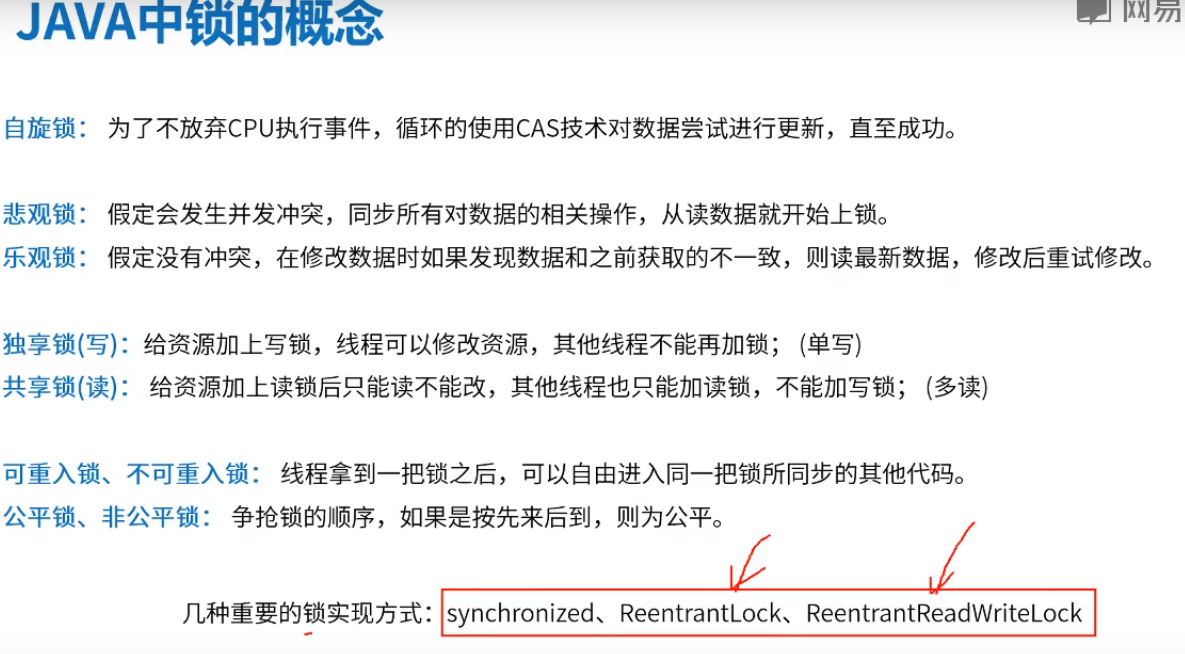
自增更新器（如LongAdder）内部实现时，拆分为多个cell（内存块不同），同时进行相加，如同分工分布式计算一样，最后再把cell相加，从而提高性能。适合频繁更新单不频繁读取的场景。

LongAccumulator，更新器增强版，可实现自定义计算，利用同样的优化方式，但是只适用于具有交换律的运算。

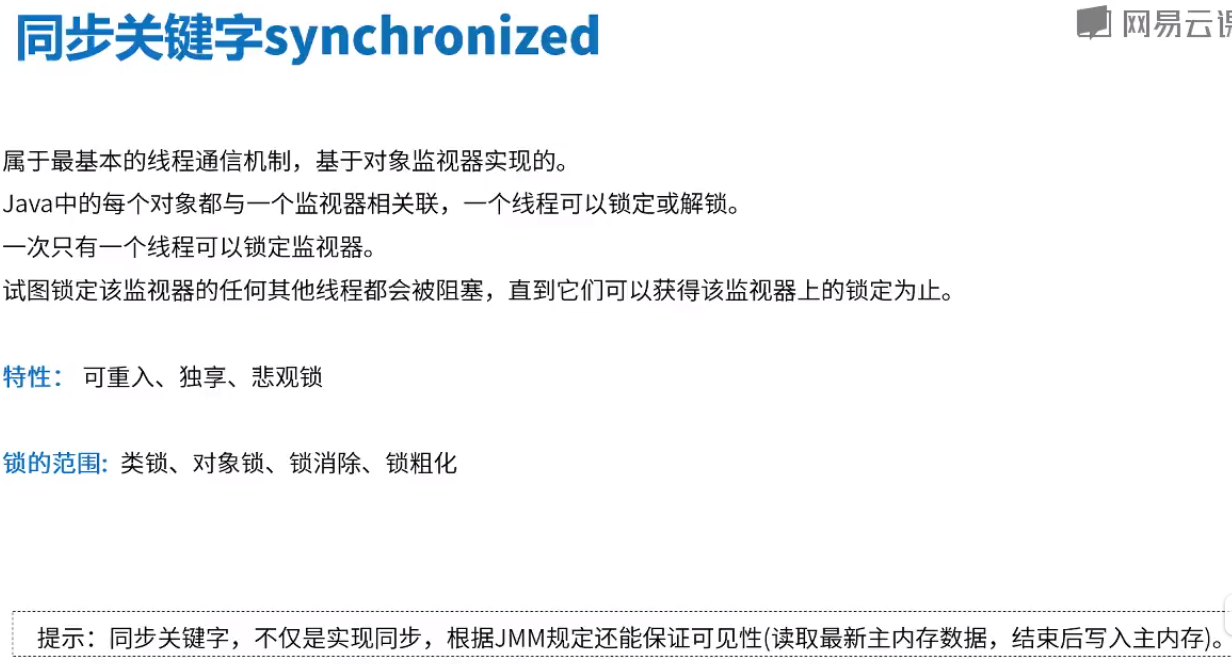
自增代码实现性能比较：同步代码块>更新器>Atomic

## JAVA锁

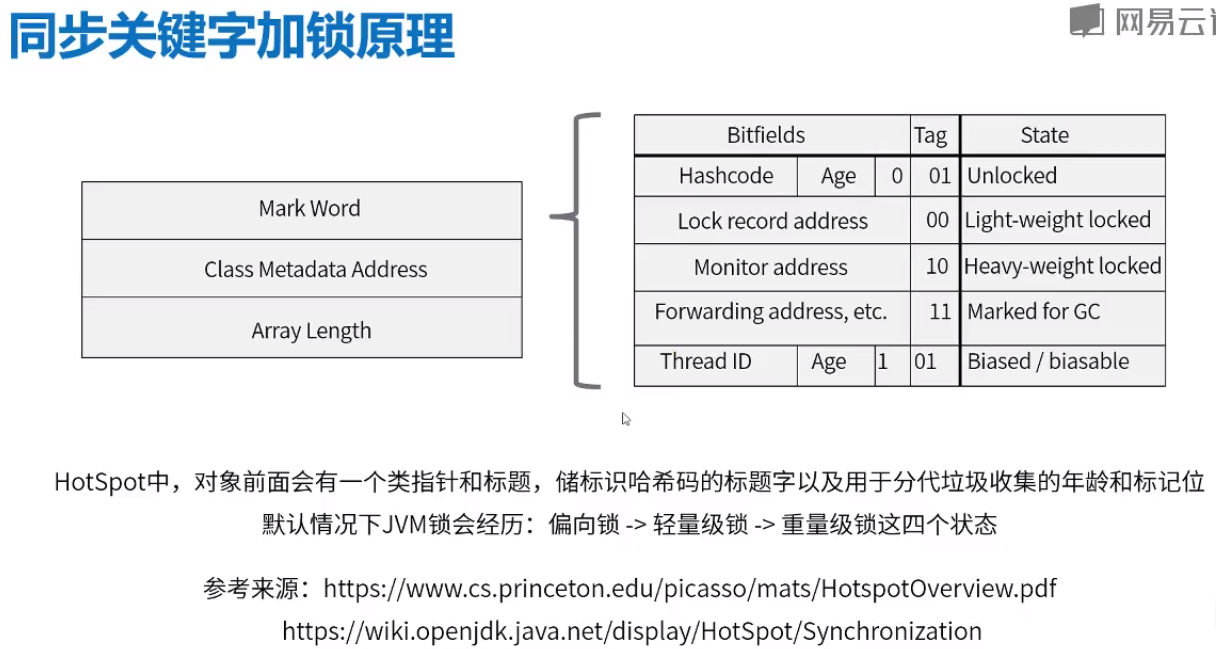
##### 锁的概念

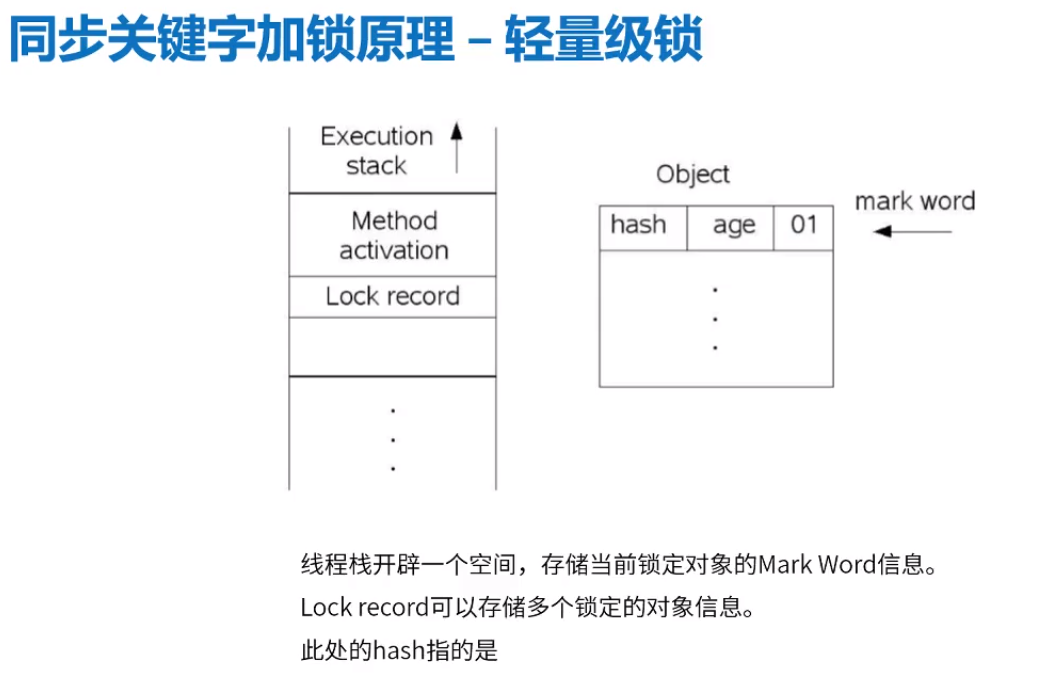


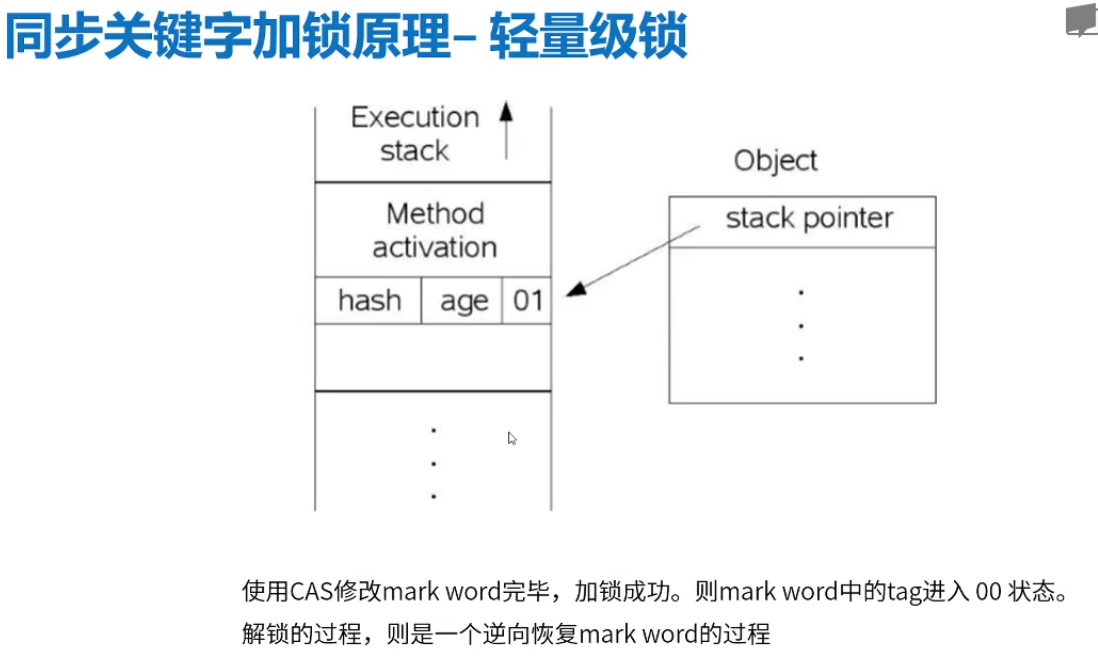
##### 同步关键字synchronized



##### 同步关键字实现原理

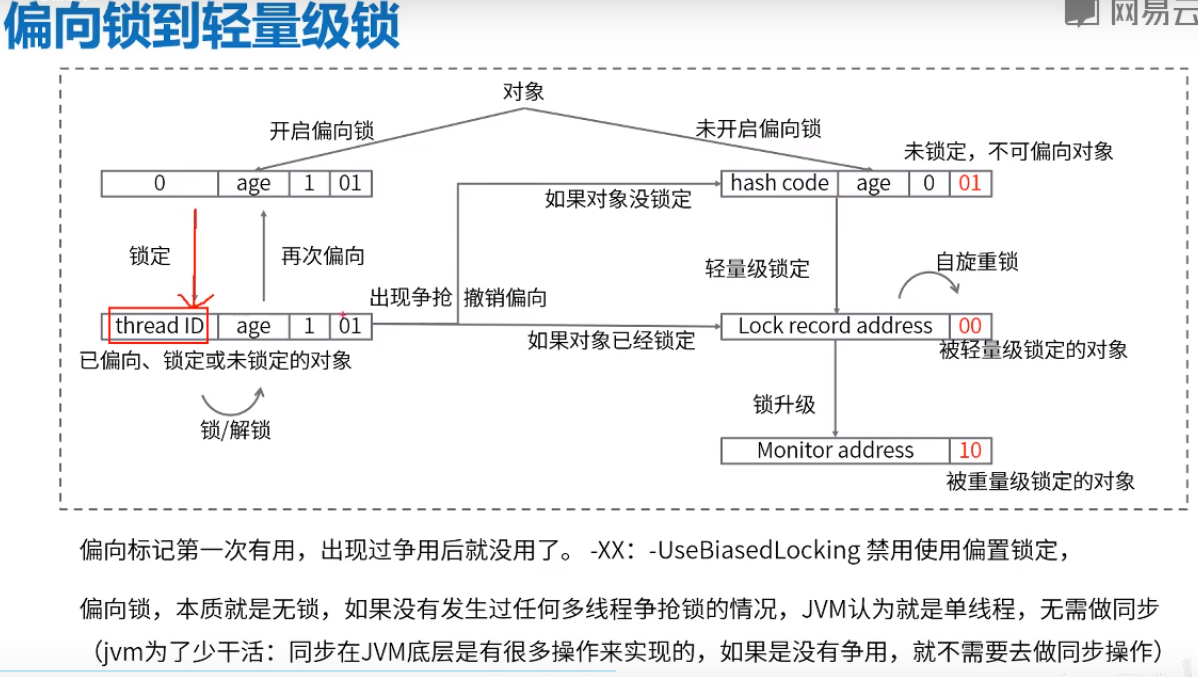


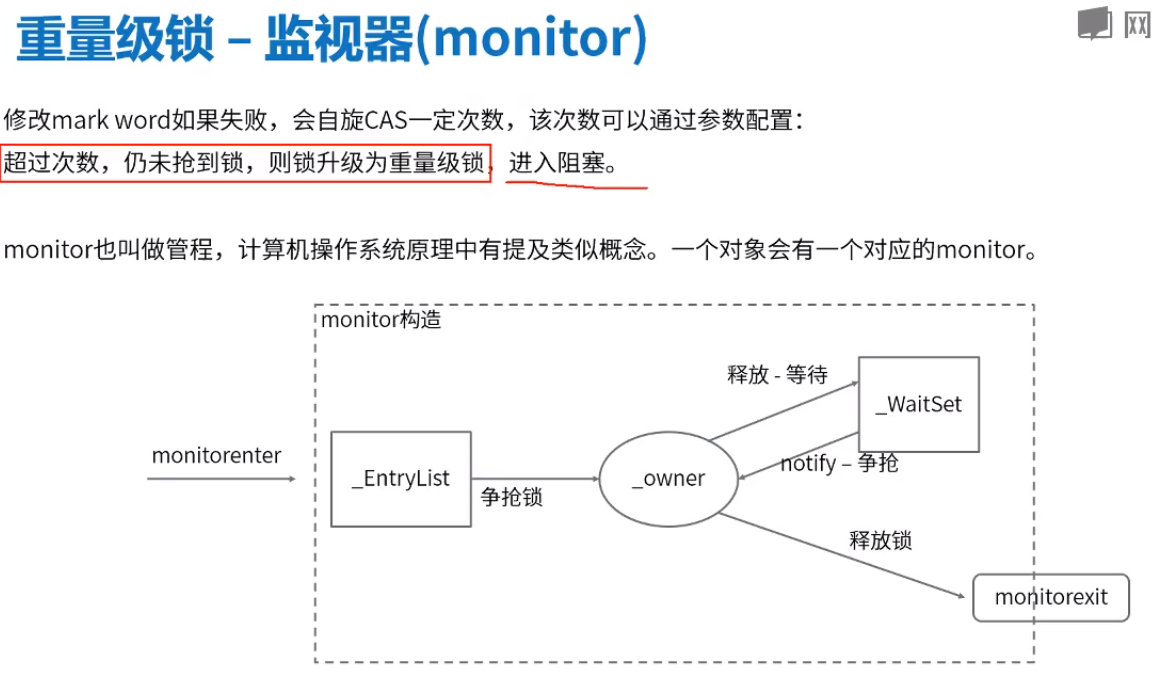




加锁与升级：

先使用偏向锁， 当有人争抢锁，升级到轻量级锁，也就是CAS自旋，如果自旋数达到一定量，则升级为重量级锁，也就是监视器锁。





## Lock





