一. CAS ?

1. 什么是CAS ?

CAS 操作包含三个操作数 —— 内存位置（V）、预期原值（A）和新值(B)。 如果内存位置的值与预期原值相匹配，那么处理器会自动将该位置值更新为新值 。否则，处理器不做任何操作。无论哪种情况，它都会在 CAS 指令之前返回该 位置的值。（在 CAS 的一些特殊情况下将仅返回 CAS 是否成功，而不提取当前值。）CAS 有效地说明了“我认为位置 V 应该包含值 A；如果包含该值，则将 B 放到这个位置；否则，不要更改该位置，只告诉我这个位置现在的值即可。

2. CAS的目的 ？

利用CPU的CAS指令，同时借助JNI来完成Java的非阻塞算法。其它原子操作都是利用类似的特性完成的。而整个J.U.C都是建立在CAS之上的，因此对于synchronized阻塞算法，J.U.C在性能上有了很大的提升。

3. CAS存在的问题 ?

1. ABA问题

ABA问题的解决思路就是使用版本号。在变量前面追加上版本号，每次变量更新的时候把版本号加一，那么A－B－A 就会变成1A-2B－3A。

从Java1.5开始JDK的atomic包里提供了一个类AtomicStampedReference来解决ABA问题。这个类的compareAndSet方法作用是首先检查当前引用是否等于预期引用，并且当前标志是否等于预期标志，如果全部相等，则以原子方式将该引用和该标志的值设置为给定的更新值。

1. 循环时间长开销大

自旋CAS如果长时间不成功，会给CPU带来非常大的执行开销。

1. 只能保证一个共享变量的原子操作

从Java1.5开始JDK提供了AtomicReference类来保证引用对象之间的原子性，你可以把多个变量放在一个对象里来进行CAS操作。

二. JUC

1. [BlockingQueue](mk:@MSITStore:F:\文档\jdk.API(1.6-1.9)\jdk%20api%201.8_google.CHM::/java/util/concurrent/BlockingQueue.html) <E>

阻塞队列，常用的为LinkedBlockingQueue 双向队列FIFO，则其头部为存放最久的元素，其使用了两个锁，head 锁与 tail锁，当进行offer/put 元素时会锁定tail锁，当poll/take元素时锁定head锁，但当remove操作时会锁定两个锁，因为removes时会遍历整个队列，而且并无法确定会移除哪一个元素。

2. ConcurrentMap<K,V>

Map的多线程实现方式，1.8前使用Segment<K,V> extends ReentrantLock

原理： 在读和写时都使用了unsafe的CAS方放进行操作；在写数据时，如果出现hash冲突的情况，则会使用synchronized 进行加锁，当处理完hash冲突的元素后方会解锁。

3. CopyOnWriteArrayList<E>

一个读写分离的arraylist,读不加锁，写加锁，锁使用ReentrantLock

原理：使用了一个Object[] array数组来存储元素，当读时，互不干涉，直接查询即可；当写时，先进行加锁，内部会copy一个数组副本进行操作，当修改完成后直接将array指向副本

4. CountDownLatch

使用了AbstractOwnableSynchronizer 来实现具体的方法，AQS不是很懂

5. CyclicBarrier

6. Executors

7. Semaphore