## 看看一线大厂面试题

### sychronied修饰普通方法和静态方法的区别？什么是可见性?

对象锁是用于对象实例方法，或者一个对象实例上的，类锁是用于类的静态方法或者一个类的class对象上的。我们知道，类的对象实例可以有很多个，但是每个类只有一个class对象，所以不同对象实例的对象锁是互不干扰的，但是每个类只有一个类锁。

但是有一点必须注意的是，其实类锁只是一个概念上的东西，并不是真实存在的，类锁其实锁的是每个类的对应的class对象。类锁和对象锁之间也是互不干扰的。

可见性是指当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值。

由于线程对变量的所有操作都必须在工作内存中进行，而不能直接读写主内存中的变量，那么对于共享变量V，它们首先是在自己的工作内存，之后再同步到主内存。可是并不会及时的刷到主存中，而是会有一定时间差。很明显，这个时候线程 A 对变量 V 的操作对于线程 B 而言就不具备可见性了 。要解决共享对象可见性这个问题，我们可以使用volatile关键字或者是加锁。

### 锁分哪几类？



### CAS无锁编程的原理。

使用当前的处理器基本都支持CAS()的指令，只不过每个厂家所实现的算法并不一样，每一个CAS操作过程都包含三个运算符：一个内存地址V，一个期望的值A和一个新值B，操作的时候如果这个地址上存放的值等于这个期望的值A，则将地址上的值赋为新值B，否则不做任何操作。

CAS的基本思路就是，如果这个地址上的值和期望的值相等，则给其赋予新值，否则不做任何事儿，但是要返回原值是多少。循环CAS就是在一个循环里不断的做cas操作，直到成功为止。

还可以说说CAS的三大问题。

1. ABA问题
2. 开销问题
3. 只能修改单一变量

### ReentrantLock的实现原理。

线程可以重复进入任何一个它已经拥有的锁所同步着的代码块，synchronized、ReentrantLock都是可重入的锁。在实现上，就是线程每次获取锁时判定如果获得锁的线程是它自己时，简单将计数器累积即可，每 释放一次锁，进行计数器累减，直到计算器归零，表示线程已经彻底释放锁。

底层则是利用了JUC中的AQS来实现的。

### AQS原理 （小米 京东）

是用来构建锁或者其他同步组件的基础框架，比如ReentrantLock、ReentrantReadWriteLock和CountDownLatch就是基于AQS实现的。它使用了一个int成员变量state表示同步状态，通过内置的FIFO队列来完成资源获取线程的排队工作。它是CLH队列锁的一种变体实现。它可以实现2种同步方式：独占式，共享式。

AQS的主要使用方式是继承，子类通过继承AQS并实现它的抽象方法来管理同步状态，同步器的设计基于模板方法模式，所以如果要实现我们自己的同步工具类就需要覆盖其中几个可重写的方法，如tryAcquire、tryReleaseShared等等。

这样设计的目的是同步组件（比如锁）是面向使用者的，它定义了使用者与同步组件交互的接口（比如可以允许两个线程并行访问），隐藏了实现细节；同步器面向的是锁的实现者，它简化了锁的实现方式，屏蔽了同步状态管理、线程的排队、等待与唤醒等底层操作。这样就很好地隔离了使用者和实现者所需关注的领域。

在内部，AQS维护一个共享资源state，通过内置的FIFO来完成获取资源线程的排队工作。该队列由一个一个的Node结点组成，每个Node结点维护一个prev引用和next引用，分别指向自己的前驱和后继结点，构成一个双端双向链表。

### Synchronized的原理以及与ReentrantLock的区别。（360）

synchronized (this)原理：涉及两条指令：monitorenter，monitorexit；再说同步方法，从同步方法反编译的结果来看，方法的同步并没有通过指令monitorenter和monitorexit来实现，相对于普通方法，其常量池中多了ACC\_SYNCHRONIZED标示符。

JVM就是根据该标示符来实现方法的同步的：当方法被调用时，调用指令将会检查方法的 ACC\_SYNCHRONIZED 访问标志是否被设置，如果设置了，执行线程将先获取monitor，获取成功之后才能执行方法体，方法执行完后再释放monitor。在方法执行期间，其他任何线程都无法再获得同一个monitor对象。

Reentrantlock 可重入锁

### Synchronized做了哪些优化 （京东）

引入如自旋锁、适应性自旋锁、锁消除、锁粗化、偏向锁、轻量级锁、逃逸分析

等技术来减少锁操作的开销。

逃逸分析

如果证明一个对象不会逃逸方法外或者线程外，则可针对此变量进行优化：

同步消除synchronization Elimination，如果一个对象不会逃逸出线程，则对此变量的同步措施可消除。

锁消除和粗化

锁消除：虚拟机的运行时编译器在运行时如果检测到一些要求同步的代码上不可能发生共享数据竞争，则会去掉这些锁。

锁粗化：将临近的代码块用同一个锁合并起来。

消除无意义的锁获取和释放，可以提高程序运行性能。

### Synchronized static与非static锁的区别和范围（小米）

对象锁是用于对象实例方法，或者一个对象实例上的，类锁是用于类的静态方法或者一个类的class对象上的。我们知道，类的对象实例可以有很多个，但是每个类只有一个class对象，所以不同对象实例的对象锁是互不干扰的，但是每个类只有一个类锁。

但是有一点必须注意的是，其实类锁只是一个概念上的东西，并不是真实存在的，类锁其实锁的是每个类的对应的class对象。类锁和对象锁之间也是互不干扰的。

### volatile 能否保证线程安全？在DCL上的作用是什么？

不能保证，在DCL的作用是：volatile是会保证被修饰的变量的可见性和 有序性，保证了单例模式下，保证在创建对象的时候的执行顺序一定是

1.分配内存空间

2.实例化对象instance

3.把instance引用指向已分配的内存空间,此时instance有了内存地址,不再为null了

的步骤, 从而保证了instance要么为null 要么是已经完全初始化好的对象。

### volatile和synchronize有什么区别？（B站 小米 京东）

volatile是最轻量的同步机制。

volatile保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。但是volatile不能保证操作的原子性，因此多线程下的写复合操作会导致线程安全问题。

关键字synchronized可以修饰方法或者以同步块的形式来进行使用，它主要确保多个线程在同一个时刻，只能有一个线程处于方法或者同步块中，它保证了线程对变量访问的可见性和排他性，又称为内置锁机制。

### 什么是守护线程？你是如何退出一个线程的？

Daemon（守护）线程是一种支持型线程，因为它主要被用作程序中后台调度以及支持性工作。这意味着，当一个Java虚拟机中不存在**非**Daemon线程的时候，Java虚拟机将会退出。可以通过调用Thread.setDaemon(true)将线程设置为Daemon线程。我们一般用不上，比如垃圾回收线程就是Daemon线程。

线程的中止：

要么是run执行完成了，要么是抛出了一个未处理的异常导致线程提前结束。

暂停、恢复和停止操作对应在线程Thread的API就是**suspend()、resume()和stop()**。但是这些API是过期的，也就是不建议使用的。因为会导致程序可能工作在不确定状态下。

安全的中止则是其他线程通过调用某个线程A的interrupt()方法对其进行中断操作，被中断的线程则是通过线程通过方法isInterrupted()来进行判断是否被中断，也可以调用静态方法Thread.interrupted()来进行判断当前线程是否被中断，不过Thread.interrupted()会同时将中断标识位改写为false。

### sleep 、wait、yield 的区别，wait 的线程如何唤醒它？（东方头条）

yield()方法：使当前线程让出CPU占有权，但让出的时间是不可设定的。也不会释放锁资源。所有执行yield()的线程有可能在进入到就绪状态后会被操作系统再次选中马上又被执行。

yield() 、sleep()被调用后，都不会释放当前线程所持有的锁。

调用wait()方法后，会释放当前线程持有的锁，而且当前被唤醒后，会重新去竞争锁，锁竞争到后才会执行wait方法后面的代码。

Wait通常被用于线程间交互，sleep通常被用于暂停执行，yield()方法使当前线程让出CPU占有权。

wait 的线程使用notify/notifyAll()进行唤醒。

### sleep是可中断的么？（小米）

sleep本身就支持中断，如果线程在sleep期间被中断，则会抛出一个中断异常。

### 线程生命周期。

Java中线程的状态分为6种：

1. 初始(NEW)：新创建了一个线程对象，但还没有调用start()方法。

2. 运行(RUNNABLE)：Java线程中将就绪（ready）和运行中（running）两种状态笼统的称为“运行”。

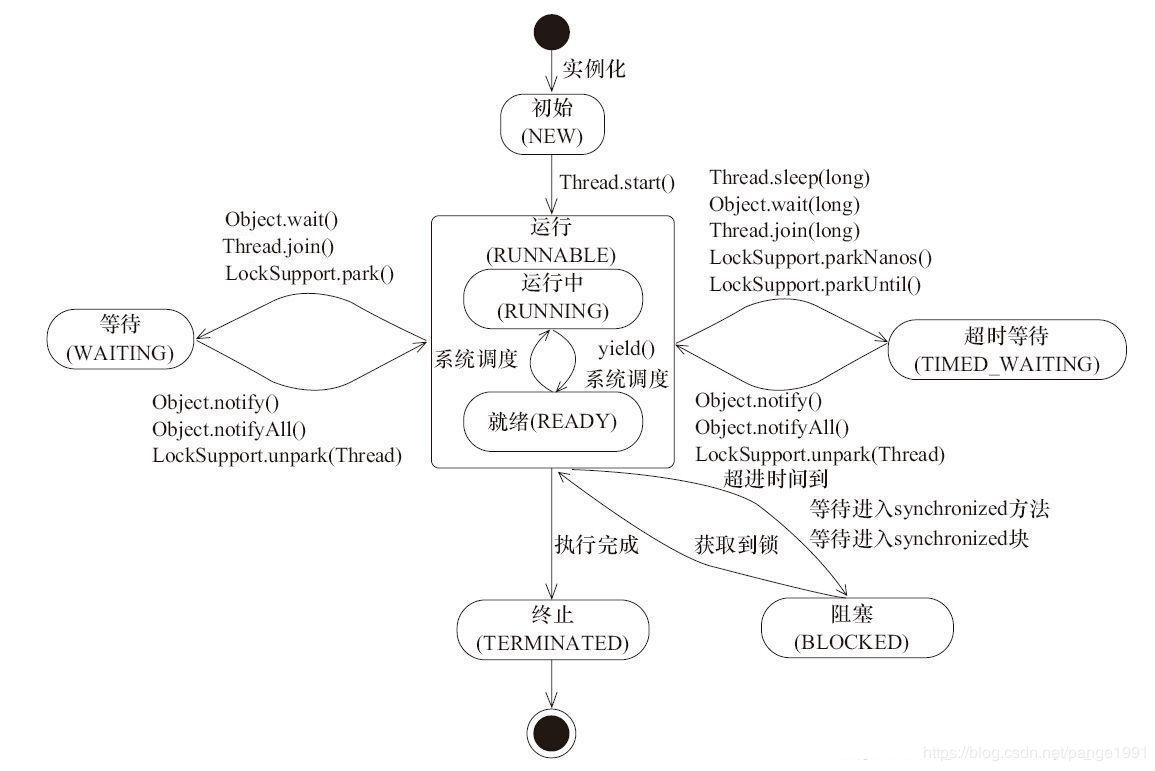
线程对象创建后，其他线程(比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获取CPU的使用权，此时处于就绪状态（ready）。就绪状态的线程在获得CPU时间片后变为运行中状态（running）。

3. 阻塞(BLOCKED)：表示线程阻塞于锁。

4. 等待(WAITING)：进入该状态的线程需要等待其他线程做出一些特定动作（通知或中断）。

5. 超时等待(TIMED\_WAITING)：该状态不同于WAITING，它可以在指定的时间后自行返回。

6. 终止(TERMINATED)：表示该线程已经执行完毕。



### ThreadLocal是什么？

ThreadLocal是Java里一种特殊的变量。ThreadLocal为每个线程都提供了变量的副本，使得每个线程在某一时间訪问到的并非同一个对象，这样就隔离了多个线程对数据的数据共享。

在内部实现上，每个线程内部都有一个ThreadLocalMap，用来保存每个线程所拥有的变量副本。

### 线程池基本原理。

在开发过程中，合理地使用线程池能够带来3个好处。

第一：降低资源消耗。第二：提高响应速度。第三：提高线程的可管理性。

1）如果当前运行的线程少于corePoolSize，则创建新线程来执行任务（注意，执行这一步骤需要获取全局锁）。

2）如果运行的线程等于或多于corePoolSize，则将任务加入BlockingQueue。

3）如果无法将任务加入BlockingQueue（队列已满），则创建新的线程来处理任务。

4）如果创建新线程将使当前运行的线程超出maximumPoolSize，任务将被拒绝，并调用RejectedExecutionHandler.rejectedExecution()方法。

### 有三个线程T1，T2，T3，怎么确保它们按顺序执行？

可以用join方法实现。