* 什么是Daemon线程？它有什么意义？  
         答：所谓后台(daemon)线程，是指在程序运行的时候在后台提供一种通用服务的线程，并且这个线程并不属于程序中不可或缺的部分。因此，当所有的非后台线程介绍时，程序也就终止了，同时会杀死进程中的所有后台线程。反过来说，只要有任何非后台线程还在运行，程序就不会终止。必须在线程启动之前调用setDaemon()方法，才能把它设置为后台线程。注意：后台进程在不执行finally子句的情况下就会终止其run()方法。
* java如何实现多线程之间的通讯和协作？  
        答： Java提供了3个非常重要的方法来巧妙地解决线程间的通信问题。这3个方法分别是：wait()、notify()和notifyAll()。它们都是Object类的最终方法，因此每一个类都默认拥有它们。虽然所有的类都默认拥有这3个方法，但是只有在synchronized关键字作用的范围内，并且是同一个同步问题中搭配使用这3个方法时才有实际的意义
* 上述两个正确吗？待确认

在java 1.5后提供了 volatile 保证读线程一定在写线程完成之后.---所以又衍生出volatile的问题。

缓存一致性协议。最出名的就是Intel 的MESI协议，MESI协议保证了每个缓存中使用的共享变量的副本是一致的。它核心的思想是：当CPU写数据时，如果发现操作的变量是共享变量，即在其他CPU中也存在该变量的副本，会发出信号通知其他CPU将该变量的缓存行置为无效状态，因此当其他CPU需要读取这个变量时，发现自己缓存中缓存该变量的缓存行是无效的，那么它就会从内存重新读取。

当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。

通过synchronized和Lock也能够保证可见性，synchronized和Lock能保证同一时刻只有一个线程获取锁然后执行同步代码，并且在释放锁之前会将对变量的修改刷新到主存当中。因此可以保证可见性。

### Thread 类中的start() 和 run() 方法有什么区别？

start()方法被用来启动新创建的线程，而且start()内部调用了run()方法，这和直接调用run()方法的效果不一样。当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

Java中Runnable和Callable有什么不同？

Runnable和Callable都代表那些要在不同的线程中执行的任务。Runnable从JDK1.0开始就有了，Callable是在JDK1.5增加的。它们的主要区别是Callable的 call() 方法可以返回值和抛出异常，而Runnable的run()方法没有这些功能。Callable可以返回装载有计算结果的Future对象。

 DB优化

        a、在数据库设计的时候就要考虑到后期的维护，数据库三范式是我们设计数据库索要遵循的原则。

        b、索引的建立：建立索引要适当，如果一个表经常用来被查询，对于增加和修改很少被用到，我们就可以为这个表建立索引，因为对于增加和修改和删除操作时，我们对索引的维护要大大超过索引给我们带来的效率。

        c、表字段的类型选择要恰当

包括字段的长度、类型等，要根据实际存储的数据进行选择，长度不要过长，否则会影响效率。

        d、外键要慎用，因为主键代表这一张表，而外键代表一群表，对表之间进行了关联，在删除修改等需要我们关联。

        e、在数据库操作上

                尽量使用prepareStatement，少用Statement，因为PrepareStatement是进行预编译的。

                connection设置为readOnly，Connection是对书库连接，属于重量级，我们使用即可。

                连接池的使用，我们可以修改数据库默认的连接数。

设计模式和编程模型，讲一下传统一个请求一个线程的模式和selector模式的区别，讲一下监听模式和广播模式的异同等等，都是多线程编程常见的概念

### Java中什么是竞态条件？

竞态条件会导致程序在并发情况下出现一些bugs。多线程对一些资源的竞争的时候就会产生竞态条件，如果首先要执行的程序竞争失败排到后面执行了，那么整个程序就会出现一些不确定的bugs。这种bugs很难发现而且会重复出现，因为线程间的随机竞争。

1. 使用 interrupt 方法终止线程

使用Thread提供的interrupt()方法，因为该方法虽然不会中断一个正在运行的线程，但是它可以使一个被阻塞的线程抛出一个中断异常，从而使线程提前结束阻塞状态，退出堵塞代码。

注意：在Thread类中有两个方法可以判断线程是否通过interrupt方法被终止。一个是静态的方法interrupted（），一个是非静态的方法isInterrupted（），这两个方法的区别是interrupted用来判断当前线是否被中断，而isInterrupted可以用来判断其他线程是否被中断。

* 死锁：是指两个或两个以上的进程（或线程）在执行过程中，因争夺资源而造成的一种相互等待的现象，若无外力作用，他们将无法推进下去；
* 活锁：是指两个线程优先级相同，都礼让不走，就这样一直僵持下去；
* 饿死：在单线程情况下，A、B两个线程，A先执行；A在执行过程中，C线程来了，B让C先执行；C在执行过程中，D线程来了，B也让D先执行，就这样B一直都是等待状态。
* 竞态条件：多个线程竞争同一个变量，导致数据的不正确性，线程的访问顺序是不可控的，会影响最终的结果。

 https://blog.csdn.net/Clifnich/article/details/78447524

产生死锁的必要条件：

1、互斥使用（资源独占）

　　一个资源每次只能给一个进程使用（比如写操作）

2、占有且等待：

　　进程在申请新的资源的同时，保持对原有资源的占有

3、不可抢占：

　　资源申请者不能强行从资源占有者手动夺取资源，资源只能由占有者自愿释放

4、循环等待：

　　A等待B占有的资源，B等待C占有的资源，C等待D占有的资源，..........N等待A的资源，形成一个线程等待回路

### Java中notify 和 notifyAll有什么区别？

notify可能会导致死锁，而notifyAll则不会

任何时候只有一个线程可以获得锁，也就是说只有一个线程可以运行synchronized 中的代码

使用notifyall,可以唤醒  
所有处于wait状态的线程，使其重新进入锁的争夺队列中，而notify只能唤醒一个。

wait() 应配合while循环使用，不应使用if，务必在wait()调用前后都检查条件，如果不满足，必须调用notify()唤醒另外的线程来处理，自己继续wait()直至条件满足再往下执行。

notify() 是对notifyAll()的一个优化，但它有很精确的应用场景，并且要求正确使用。不然可能导致死锁。正确的场景应该是 WaitSet中等待的是相同的条件，唤醒任一个都能正确处理接下来的事项，如果唤醒的线程无法正确处理，务必确保继续notify()下一个线程，并且自身需要重新回到WaitSet中.

**为什么wait, notify 和 notifyAll这些方法不在thread类里面？**

明显的原因是JAVA提供的锁是对象级的而不是线程级的，每个对象都有锁，通过线程获得。如果线程需要等待某些锁那么调用对象中的wait()方法就有意义了。如果wait()方法定义在Thread类中，线程正在等待的是哪个锁就不明显了。简单的说，由于wait，notify和notifyAll都是锁级别的操作，所以把他们定义在Object类中因为锁属于对象。

**为什么wait和notify方法要在同步块中调用？**

1. 只有在调用线程拥有某个对象的独占锁时，才能够调用该对象的wait(),notify()和notifyAll()方法。
2. 如果你不这么做，你的代码会抛出IllegalMonitorStateException异常。
3. 还有一个原因是为了避免wait和notify之间产生竞态条件。

wait()方法强制当前线程释放对象锁。这意味着在调用某对象的wait()方法之前，当前线程必须已经获得该对象的锁。因此，线程必须在某个对象的同步方法或同步代码块中才能调用该对象的wait()方法。

在调用对象的notify()和notifyAll()方法之前，调用线程必须已经得到该对象的锁。因此，必须在某个对象的同步方法或同步代码块中才能调用该对象的notify()或notifyAll()方法。

调用wait()方法的原因通常是，调用线程希望某个特殊的状态(或变量)被设置之后再继续执行。调用notify()或notifyAll()方法的原因通常是，调用线程希望告诉其他等待中的线程:"特殊状态已经被设置"。这个状态作为线程间通信的通道，它必须是一个可变的共享状态(或变量)。

**为什么你应该在循环中检查等待(wait)条件?---其实不太懂，因为没写过。**

处于等待状态的线程可能会收到错误警报和伪唤醒，如果不在循环中检查等待条件，程序就会在没有满足结束条件的情况下退出。因此，当一个等待线程醒来时，不能认为它原来的等待状态仍然是有效的，在notify()方法调用之后和等待线程醒来之前这段时间它可能会改变。

1、一般来说，wait肯定是在某个条件调用的，不是if就是while

2、放在while里面，是防止出于waiting的对象被别的原因调用了唤醒方法，但是while里面的条件并没有满足（也可能当时满足了，但是由于别的线程操作后，又不满足了），就需要再次调用wait将其挂起。

3、其实还有一点，就是while最好也被同步，这样不会导致错失信号。

### Java多线程中调用wait() 和 sleep()方法有什么不同？

java程序中wait 和 sleep都会造成某种形式的暂停，它们可以满足不同的需要。wait()方法用于线程间通信，如果等待条件为真且其它线程被唤醒时它会释放锁，而sleep()方法仅仅释放CPU资源或者让当前线程停止执行一段时间，但不会释放锁。

Java中synchronized 和 ReentrantLock 有什么不同？

相似点：

这两种同步方式有很多相似之处，它们都是加锁方式同步，而且都是阻塞式的同步，也就是说当如果一个线程获得了对象锁，进入了同步块，其他访问该同步块的线程都必须阻塞在同步块外面等待，而进行线程阻塞和唤醒的代价是比较高的.

区别：

这两种方式最大区别就是对于Synchronized来说，它是java语言的关键字，是原生语法层面的互斥，需要jvm实现。而ReentrantLock它是JDK 1.5之后提供的API层面的互斥锁，需要lock()和unlock()方法配合try/finally语句块来完成。

Synchronized进过编译，会在同步块的前后分别形成monitorenter和monitorexit这个两个字节码指令。在执行monitorenter指令时，首先要尝试获取对象锁。如果这个对象没被锁定，或者当前线程已经拥有了那个对象锁，把锁的计算器加1，相应的，在执行monitorexit指令时会将锁计算器就减1，当计算器为0时，锁就被释放了。如果获取对象锁失败，那当前线程就要阻塞，直到对象锁被另一个线程释放为止。

由于ReentrantLock是java.util.concurrent包下提供的一套互斥锁，相比Synchronized，ReentrantLock类提供了一些高级功能，主要有以下3项：

1.等待可中断，持有锁的线程长期不释放的时候，正在等待的线程可以选择放弃等待，这相当于Synchronized来说可以避免出现死锁的情况。

2.公平锁，多个线程等待同一个锁时，必须按照申请锁的时间顺序获得锁，Synchronized锁非公平锁，ReentrantLock默认的构造函数是创建的非公平锁，可以通过参数true设为公平锁，但公平锁表现的性能不是很好。

3.锁绑定多个条件，一个ReentrantLock对象可以同时绑定对个对象。

**有三个线程T1,T2,T3,下面方法可以确保它们按顺序执行的有**

在多线程中有多种方法让线程按特定顺序执行，你可以用线程类的join()方法在一个线程中启动另一个线程，另外一个线程完成该线程继续执行。为了确保三个线程的顺序你应该先启动最后一个(T3调用T2，T2调用T1)，这样T1就会先完成而T3最后完成。

实际上先启动三个线程中哪一个都行，  
因为在每个线程的run方法中用join方法限定了三个线程的执行顺序。

### 21. Thread类中的yield方法有什么作用？

Yield方法可以暂停当前正在执行的线程对象，让其它有相同优先级的线程执行。它是一个静态方法而且只保证当前线程放弃CPU占用而不能保证使其它线程一定能占用CPU，执行yield()的线程有可能在进入到暂停状态后马上又被执行。

### 22. Java中Semaphore是什么？

Java中的Semaphore是一种新的同步类，它是一个计数信号。从概念上讲，从概念上讲，信号量维护了一个许可集合。如有必要，在许可可用前会阻塞每一个 acquire()，然后再获取该许可。每个 release()添加一个许可，从而可能释放一个正在阻塞的获取者。但是，不使用实际的许可对象，Semaphore只对可用许可的号码进行计数，并采取相应的行动。信号量常常用于多线程的代码中，比如数据库连接池。

### 23. 如果你提交任务时，线程池队列已满。会时发会生什么?

事实上如果一个任务不能被调度执行那么ThreadPoolExecutor’s submit()方法将会抛出一个RejectedExecutionException异常。

### 24. Java线程池中submit() 和 execute()方法有什么区别？

两个方法都可以向线程池提交任务，execute()方法的返回类型是void，它定义在Executor接口中, 而submit()方法可以返回持有计算结果的Future对象，它定义在ExecutorService接口中，它扩展了Executor接口，其它线程池类像ThreadPoolExecutor和ScheduledThreadPoolExecutor都有这些方法。

### 25. 如何在Java中创建Immutable对象？

可是Java没有@Immutable这个注解符，要创建不可变类，要实现下面几个步骤：通过构造方法初始化所有成员、对变量不要提供setter方法、将所有的成员声明为私有的，这样就不允许直接访问这些成员、在getter方法中，不要直接返回对象本身，而是克隆对象，并返回对象的拷贝。

https://www.jianshu.com/p/710a9c116007

OOP: Object Oriented Programming,[面向对象](https://www.baidu.com/s?wd=面向对象&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)的程序设计