## **Java实现线程有哪几种方式？**

1、继承Thread类实现多线程  
2、实现Runnable接口方式实现多线程  
3、使用ExecutorService、Callable、Future实现有返回结果的多线程

## **多线程同步有哪几种方法？**

Synchronized关键字，Lock锁实现，分布式锁等。

## ****Runnable和Thread用哪个好？****

Java不支持类的多重继承，但允许你实现多个接口。所以如果你要继承其他类，也为了减少类之间的耦合性，Runnable会更好。

## ****Java中notify和notifyAll有什么区别？****

notify()方法不能唤醒某个具体的线程，所以只有一个线程在等待的时候它才有用武之地。而notifyAll()唤醒所有线程并允许他们争夺锁确保了至少有一个线程能继续运行。

## ****为什么wait/notify/notifyAll这些方法不在thread类里面？****

这是个设计相关的问题，它考察的是面试者对现有系统和一些普遍存在但看起来不合理的事物的看法。回答这些问题的时候，你要说明为什么把这些方法放在Object类里是有意义的，还有不把它放在Thread类里的原因。一个很明显的原因是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？****

主要是因为Java API强制要求这样做，如果你不这么做，你的代码会抛出IllegalMonitorStateException异常。还有一个原因是为了避免wait和notify之间产生竞态条件。

## **什么是死锁？如何避免死锁？**

死锁就是两个线程相互等待对方释放对象锁。

## ****启动线程方法start()和run()有什么区别？****

只有调用了start()方法，才会表现出多线程的特性，不同线程的run()方法里面的代码交替执行。如果只是调用run()方法，那么代码还是同步执行的，必须等待一个线程的run()方法里面的代码全部执行完毕之后，另外一个线程才可以执行其run()方法里面的代码。

## **多线程之间如何进行通信？**

wait/notify

## **什么是线程池？**

很简单，简单看名字就知道是装有线程的池子，我们可以把要执行的多线程交给线程池来处理，和连接池的概念一样，通过维护一定数量的线程池来达到多个线程的复用。

## **线程池的好处**

我们知道不用线程池的话，每个线程都要通过new Thread(xxRunnable).start()的方式来创建并运行一个线程，线程少的话这不会是问题，而真实环境可能会开启多个线程让系统和程序达到最佳效率，当线程数达到一定数量就会耗尽系统的CPU和内存资源，也会造成GC频繁收集和停顿，因为每次创建和销毁一个线程都是要消耗系统资源的，如果为每个任务都创建线程这无疑是一个很大的性能瓶颈。所以，线程池中的线程复用极大节省了系统资源，当线程一段时间不再有任务处理时它也会自动销毁，而不会长驻内存。

## **什么是活锁、饥饿、无锁、死锁？**

死锁、活锁、饥饿是关于多线程是否活跃出现的运行阻塞障碍问题，如果线程出现了这三种情况，即线程不再活跃，不能再正常地执行下去了。

**死锁**

死锁是多线程中最差的一种情况，多个线程相互占用对方的资源的锁，而又相互等对方释放锁，此时若无外力干预，这些线程则一直处理阻塞的假死状态，形成死锁。

举个例子，A同学抢了B同学的钢笔，B同学抢了A同学的书，两个人都相互占用对方的东西，都在让对方先还给自己自己再还，这样一直争执下去等待对方还而又得不到解决，老师知道此事后就让他们相互还给对方，这样在外力的干预下他们才解决，当然这只是个例子没有老师他们也能很好解决，计算机不像人如果发现这种情况没有外力干预还是会一直阻塞下去的。

**活锁**

活锁这个概念大家应该很少有人听说或理解它的概念，而在多线程中这确实存在。活锁恰恰与死锁相反，死锁是大家都拿不到资源都占用着对方的资源，而活锁是拿到资源却又相互释放不执行。当多线程中出现了相互谦让，都主动将资源释放给别的线程使用，这样这个资源在多个线程之间跳动而又得不到执行，这就是活锁。

**饥饿**

我们知道多线程执行中有线程优先级这个东西，优先级高的线程能够插队并优先执行，这样如果优先级高的线程一直抢占优先级低线程的资源，导致低优先级线程无法得到执行，这就是饥饿。当然还有一种饥饿的情况，一个线程一直占着一个资源不放而导致其他线程得不到执行，与死锁不同的是饥饿在以后一段时间内还是能够得到执行的，如那个占用资源的线程结束了并释放了资源。

**无锁**

无锁，即没有对资源进行锁定，即所有的线程都能访问并修改同一个资源，但同时只有一个线程能修改成功。无锁典型的特点就是一个修改操作在一个循环内进行，线程会不断的尝试修改共享资源，如果没有冲突就修改成功并退出否则就会继续下一次循环尝试。所以，如果有多个线程修改同一个值必定会有一个线程能修改成功，而其他修改失败的线程会不断重试直到修改成功。之前的文章我介绍过JDK的CAS原理及应用即是无锁的实现。

可以看出，无锁是一种非常良好的设计，它不会出现线程出现的跳跃性问题，锁使用不当肯定会出现系统性能问题，虽然无锁无法全面代替有锁，但无锁在某些场合下是非常高效的。

## **Synchronized有哪几种用法？**

锁类、锁方法、锁代码块。

## **Fork/Join框架是干什么的？**

大任务自动分散小任务，并发执行，合并小任务结果。

## ****Java中用到了什么线程调度算法？****

抢占式。一个线程用完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。