线程面试题

不管你是新程序员还是老手，你一定在面试中遇到过有关线程的问题。Java 语言一个重要的特点就是内置了对并发的支持，让 Java 大受企业和程序员的欢迎。大多数待遇丰厚的 Java 开发职位都要求开发者精通多线程技术并且有丰富的 Java 程序开发、调试、优化经验，所以线程相关的问题在面试中经常会被提到。

在典型的 Java 面试中， 面试官会从线程的基本概念问起， 如：为什么你需要使用线程，如何创建线程，用什么方式创建线程比较好（比如：继承 thread 类还是调用 Runnable 接口），然后逐渐问到并发问题像在 Java 并发编程的过程中遇到了什么挑战，Java 内存模型，JDK1.5 引入了哪些更高阶的并发工具，并发编程常用的设计模式，经典多线程问题如生产者消费者，哲学家就餐，读写器或者简单的有界缓冲区问题。仅仅知道线程的基本概念是远远不够的， 你必须知道如何处理死锁，竞态条件，内存冲突和线程安全等并发问题。掌握了这些技巧，你就可以轻松应对多线程和并发面试了。

许多 Java 程序员在面试前才会去看面试题，这很正常。因为收集面试题和练习很花时间，所以我从许多面试者那里收集了 Java 多线程和并发相关的 50 个热门问题。我只收集了比较新的面试题且没有提供全部答案。想必聪明的你对这些问题早就心中有数了， 如果遇到不懂的问题，你可以用 Google 找到答案。若你实在找不到答案，可以在文章的评论中向我求助。你也可以在这找到一些答案 Java 线程问答 Top 12。

下面是 Java 线程相关的热门面试题，你可以用它来好好准备面试。

1. 什么是线程？

线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对运算密集型任务提速。比如，如果一个线程完成一个任务要 100 毫秒，那么用十个线程完成改任务只需 10 毫秒。Java 在语言层面对多线程提供了卓越的支持，它也是一个很好的卖点。欲了解更多详细信息请点击这里。

2. 线程和进程有什么区别？

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。更多详细信息请点击这里。

3. 如何在 Java 中实现线程？

在语言层面有两种方式。java.lang.Thread 类的实例就是一个线程但是它需要调用 java.lang.Runnable 接口来执行，由于线程类本身就是调用的 Runnable 接口所以你可以继承 java.lang.Thread 类或者直接调用 Runnable 接口来重写 run ()方法实现线程。更多详细信息请点击这里.

4. 用 Runnable 还是 Thread？

这个问题是上题的后续，大家都知道我们可以通过继承 Thread 类或者调用 Runnable 接口来实现线程，问题是，那个方法更好呢？什么情况下使用它？这个问题很容易回答，如果你知道 Java 不支持类的多重继承，但允许你调用多个接口。所以如果你要继承其他类，当然是调用 Runnable 接口好了。更多详细信息请点击这里。

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

这个问题经常被问到，但还是能从此区分出面试者对 Java 线程模型的理解程度。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。更多讨论请点击这里

6. 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。我的博客有更详细的说明。

7. Java 中 CyclicBarrier 和 CountDownLatch 有什么不同？

CyclicBarrier 和 CountDownLatch 都可以用来让一组线程等待其它线程。与 CyclicBarrier 不同的是，CountdownLatch 不能重新使用。点此查看更多信息和示例代码。

8. Java 内存模型是什么？

Java 内存模型规定和指引 Java 程序在不同的内存架构、CPU 和操作系统间有确定性地行为。它在多线程的情况下尤其重要。Java 内存模型对一个线程所做的变动能被其它线程可见提供了保证，它们之间是先行发生关系。这个关系定义了一些规则让程序员在并发编程时思路更清晰。比如，先行发生关系确保了：

线程内的代码能够按先后顺序执行，这被称为程序次序规则。

对于同一个锁，一个解锁操作一定要发生在时间上后发生的另一个锁定操作之前，也叫做管程锁定规则。

前一个对 Volatile 的写操作在后一个 volatile 的读操作之前，也叫 volatile 变量规则。

一个线程内的任何操作必需在这个线程的 start ()调用之后，也叫作线程启动规则。

一个线程的所有操作都会在线程终止之前，线程终止规则。

一个对象的终结操作必需在这个对象构造完成之后，也叫对象终结规则。

可传递性

我强烈建议大家阅读《Java 并发编程实践》第十六章来加深对 Java 内存模型的理解。

9. Java 中的 volatile 变量是什么？

volatile 是一个特殊的修饰符，只有成员变量才能使用它。在 Java 并发程序缺少同步类的情况下，多线程对成员变量的操作对其它线程是透明的。volatile 变量可以保证下一个读取操作会在前一个写操作之后发生，就是上一题的 volatile 变量规则。点击这里查看更多 volatile 的相关内容。

10. 什么是线程安全？Vector 是一个线程安全类吗？

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。一个线程安全的计数器类的同一个实例对象在被多个线程使用的情况下也不会出现计算失误。很显然你可以将集合类分成两组，线程安全和非线程安全的。Vector 是用同步方法来实现线程安全的， 而和它相似的 ArrayList 不是线程安全的。

11. ava 中什么是竞态条件？举个例子说明。

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。一个例子就是无序处理，详见答案。

12. Java 中如何停止一个线程？

Java 提供了很丰富的 API 但没有为停止线程提供 API。JDK 1.0 本来有一些像 stop ()，suspend () 和 resume ()的控制方法但是由于潜在的死锁威胁因此在后续的 JDK 版本中他们被弃用了，之后 Java API 的设计者就没有提供一个兼容且线程安全的方法来停止一个线程。当 run () 或者 call () 方法执行完的时候线程会自动结束，如果要手动结束一个线程，你可以用 volatile 布尔变量来退出 run ()方法的循环或者是取消任务来中断线程。点击这里查看示例代码。

13. 一个线程运行时发生异常会怎样？

这是我在一次面试中遇到的一个很刁钻的 Java 面试题，简单的说，如果异常没有被捕获该线程将会停止执行。Thread.UncaughtExceptionHandler 是用于处理未捕获异常造成线程突然中断情况的一个内嵌接口。当一个未捕获异常将造成线程中断的时候 JVM 会使用 Thread.getUncaughtExceptionHandler ()来查询线程的 UncaughtExceptionHandler 并将线程和异常作为参数传递给 handler 的 uncaughtException ()方法进行处理。

14. 如何在两个线程间共享数据？

你可以通过共享对象来实现这个目的，或者是使用像阻塞队列这样并发的数据结构。这篇教程《Java 线程间通信》(涉及到在两个线程间共享对象)用 wait 和 notify 方法实现了生产者消费者模型。

15. Java 中 notify 和 notifyAll 有什么区别？

这又是一个刁钻的问题，因为多线程可以等待单监控锁，Java API 的设计人员提供了一些方法当等待条件改变的时候通知它们，但是这些方法没有完全实现。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。我的博客有更详细的资料和示例代码。

16. 为什么 wait, notify 和 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。你也可以查看这篇文章了解更多。

17. 什么是 ThreadLocal 变量？

ThreadLocal 是 Java 里一种特殊的变量。每个线程都有一个 ThreadLocal 就是每个线程都拥有了自己独立的一个变量，竞争条件被彻底消除了。它是为创建代价高昂的对象获取线程安全的好方法，比如你可以用 ThreadLocal 让 SimpleDateFormat 变成线程安全的，因为那个类创建代价高昂且每次调用都需要创建不同的实例所以不值得在局部范围使用它，如果为每个线程提供一个自己独有的变量拷贝，将大大提高效率。首先，通过复用减少了代价高昂的对象的创建个数。其次，你在没有使用高代价的同步或者不变性的情况下获得了线程安全。线程局部变量的另一个不错的例子是 ThreadLocalRandom 类，它在多线程环境中减少了创建代价高昂的 Random 对象的个数。查看答案了解更多。

18. 什么是 FutureTask？

在 Java 并发程序中 FutureTask 表示一个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等方法。只有当运算完成的时候结果才能取回，如果运算尚未完成 get 方法将会阻塞。一个 FutureTask 对象可以对调用了 Callable 和 Runnable 的对象进行包装，由于 FutureTask 也是调用了 Runnable 接口所以它可以提交给 Executor 来执行。

19. Java 中 interrupted 和 isInterruptedd 方法的区别？

interrupted () 和 isInterrupted ()的主要区别是前者会将中断状态清除而后者不会。Java 多线程的中断机制是用内部标识来实现的，调用 Thread.interrupt ()来中断一个线程就会设置中断标识为 true。当中断线程调用静态方法 Thread.interrupted ()来检查中断状态时，中断状态会被清零。而非静态方法 isInterrupted ()用来查询其它线程的中断状态且不会改变中断状态标识。简单的说就是任何抛出 InterruptedException 异常的方法都会将中断状态清零。无论如何，一个线程的中断状态有有可能被其它线程调用中断来改变。

20. 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。

21. 为什么你应该在循环中检查等待条件?

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。这就是在循环中使用 wait ()方法效果更好的原因，你可以在 Eclipse 中创建模板调用 wait 和 notify 试一试。如果你想了解更多关于这个问题的内容，我推荐你阅读《Effective Java》这本书中的线程和同步章节。

22. Java 中的同步集合与并发集合有什么区别？

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。在 Java1.5 之前程序员们只有同步集合来用且在多线程并发的时候会导致争用，阻碍了系统的扩展性。Java5 介绍了并发集合像 ConcurrentHashMap，不仅提供线程安全还用锁分离和内部分区等现代技术提高了可扩展性。更多内容详见答案。

23. Java 中堆和栈有什么不同？

为什么把这个问题归类在多线程和并发面试题里？因为栈是一块和线程紧密相关的内存区域。每个线程都有自己的栈内存，用于存储本地变量，方法参数和栈调用，一个线程中存

储的变量对其它线程是不可见的。而堆是所有线程共享的一片公用内存区域。对象都在堆里创建，为了提升效率线程会从堆中弄一个缓存到自己的栈，如果多个线程使用该变量就可能引发问题，这时 volatile 变量就可以发挥作用了，它要求线程从主存中读取变量的值。

更多内容详见答案。

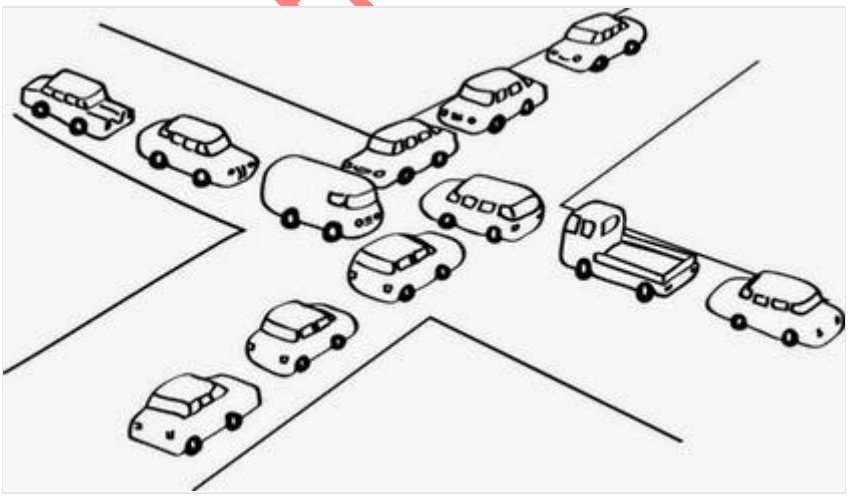
24. 什么是线程池？为什么要使用它？

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从 JDK1.5 开始，Java API 提供了 Executor 框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。更多内容详见这篇文章。

25. 如何写代码来解决生产者消费者问题？

在现实中你解决的许多线程问题都属于生产者消费者模型，就是一个线程生产任务供其它线程进行消费，你必须知道怎么进行线程间通信来解决这个问题。比较低级的办法是用 wait 和 notify 来解决这个问题，比较赞的办法是用 Semaphore 或者 BlockingQueue 来实现生产者消费者模型，这篇教程有实现它。

26. 如何避免死锁？



Java 多线程中的死锁

死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。这是一个严重的问题，因为死锁会让你的程序挂起无法完成任务，死锁的发生必须满足以下四个条件：

互斥条件：一个资源每次只能被一个进程使用。

请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。

循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

避免死锁最简单的方法就是阻止循环等待条件，将系统中所有的资源设置标志位、排序，规定所有的进程申请资源必须以一定的顺序（升序或降序）做操作来避免死锁。这篇教程有代码示例和避免死锁的讨论细节。

27. Java 中活锁和死锁有什么区别？

这是上题的扩展，活锁和死锁类似，不同之处在于处于活锁的线程或进程的状态是不断改变的，活锁可以认为是一种特殊的饥饿。一个现实的活锁例子是两个人在狭小的走廊碰到，两个人都试着避让对方好让彼此通过，但是因为避让的方向都一样导致最后谁都不能通过走廊。简单的说就是，活锁和死锁的主要区别是前者进程的状态可以改变但是却不能继续执行。

28. 怎么检测一个线程是否拥有锁？

我一直不知道我们竟然可以检测一个线程是否拥有锁，直到我参加了一次电话面试。在 java.lang.Thread 中有一个方法叫 holdsLock ()，它返回 true 如果当且仅当当前线程拥有某个具体对象的锁。你可以查看这篇文章了解更多。

29. 你如何在 Java 中获取线程堆栈？

对于不同的操作系统，有多种方法来获得 Java 进程的线程堆栈。当你获取线程堆栈时，JVM 会把所有线程的状态存到日志文件或者输出到控制台。在 Windows 你可以使用 Ctrl + Break 组合键来获取线程堆栈，Linux 下用 kill -3 命令。你也可以用 jstack 这个工具来获取，它对线程 id 进行操作，你可以用 jps 这个工具找到 id。

30. JVM 中哪个参数是用来控制线程的栈堆栈小的

这个问题很简单， -Xss 参数用来控制线程的堆栈大小。你可以查看 JVM 配置列表来了解这个参数的更多信息。

31. Java 中 synchronized 和 ReentrantLock 有什么不同？

Java 在过去很长一段时间只能通过 synchronized 关键字来实现互斥，它有一些缺点。比如你不能扩展锁之外的方法或者块边界，尝试获取锁时不能中途取消等。Java 5 通过 Lock 接口提供了更复杂的控制来解决这些问题。 ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义且它还具有可扩展性。你可以查看这篇文章了解更多

32. 有三个线程 T1，T2，T3，怎么确保它们按顺序执行？

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。你可以查看这篇文章了解更多。

33. 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。点击这里查看更多 yield 方法的相关内容。

34. Java 中 ConcurrentHashMap 的并发度是什么？

ConcurrentHashMap 把实际 map 划分成若干部分来实现它的可扩展性和线程安全。这种划分是使用并发度获得的，它是 ConcurrentHashMap 类构造函数的一个可选参数，默认值为 16，这样在多线程情况下就能避免争用。欲了解更多并发度和内部大小调整请阅读我的文章 How ConcurrentHashMap works in Java。

35. 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。更多详细信息请点击这里。

36. 如果你提交任务时，线程池队列已满。会时发会生什么？

这个问题问得很狡猾，许多程序员会认为该任务会阻塞直到线程池队列有空位。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。

37. 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。更多详细信息请点击这里。

38. 什么是阻塞式方法？

阻塞式方法是指程序会一直等待该方法完成期间不做其他事情，ServerSocket 的 accept ()方法就是一直等待客户端连接。这里的阻塞是指调用结果返回之前，当前线程会被挂起，直到得到结果之后才会返回。此外，还有异步和非阻塞式方法在任务完成前就返回。更多详细信息请点击这里。

39. Swing 是线程安全的吗？ 为什么？

你可以很肯定的给出回答，Swing 不是线程安全的，但是你应该解释这么回答的原因即便面试官没有问你为什么。当我们说 swing 不是线程安全的常常提到它的组件，这些组件不能在多线程中进行修改，所有对 GUI 组件的更新都要在 AWT 线程中完成，而 Swing 提

供了同步和异步两种回调方法来进行更新。点击这里查看更多 swing 和线程安全的相关内容。

40. Java 中 invokeAndWait 和 invokeLater 有什么区别？

这两个方法是 Swing API 提供给 Java 开发者用来从当前线程而不是事件派发线程更新 GUI 组件用的。InvokeAndWait ()同步更新 GUI 组件，比如一个进度条，一旦进度更新了，进度条也要做出相应改变。如果进度被多个线程跟踪，那么就调用 invokeAndWait () 方法请求事件派发线程对组件进行相应更新。而 invokeLater ()方法是异步调用更新组件的。更多详细信息请点击这里。

41. Swing API 中那些方法是线程安全的？

这个问题又提到了 swing 和线程安全，虽然组件不是线程安全的但是有一些方法是可以被多线程安全调用的，比如 repaint ()， revalidate ()。JTextComponent 的 setText ()方法和 JTextArea 的 insert () 和 append () 方法也是线程安全的。

42. 如何在 Java 中创建 Immutable 对象？

这个问题看起来和多线程没什么关系， 但不变性有助于简化已经很复杂的并发程序。Immutable 对象可以在没有同步的情况下共享，降低了对该对象进行并发访问时的同步化开销。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。我的文章 how to make an object Immutable in Java 有详细的教程，看完你可以充满自信。

43. Java 中的 ReadWriteLock 是什么？

一般而言，读写锁是用来提升并发程序性能的锁分离技术的成果。Java 中的 ReadWriteLock 是 Java 5 中新增的一个接口，一个 ReadWriteLock 维护一对关联的锁，一个用于只读操作一个用于写。在没有写线程的情况下一个读锁可能会同时被多个读线程持有。写锁是独占的，你可以使用 JDK 中的 ReentrantReadWriteLock 来实现这个规则，它最多支持 65535 个写锁和 65535 个读锁。

44. 多线程中的忙循环是什么?

忙循环就是程序员用循环让一个线程等待，不像传统方法 wait ()， sleep () 或 yield () 它们都放弃了 CPU 控制，而忙循环不会放弃 CPU，它就是在运行一个空循环。这么做的目的是为了保留 CPU 缓存，在多核系统中，一个等待线程醒来的时候可能会在另一个内核运行，这样会重建缓存。为了避免重建缓存和减少等待重建的时间就可以使用它了。你可以查看这篇文章获得更多信息。

45. volatile 变量和 atomic 变量有什么不同？

这是个有趣的问题。首先，volatile 变量和 atomic 变量看起来很像，但功能却不一样。Volatile 变量可以确保先行关系，即写操作会发生在后续的读操作之前， 但它并不能保证原子性。例如用 volatile 修饰 count 变量那么 count++ 操作就不是原子性的。而 AtomicInteger 类提供的 atomic 方法可以让这种操作具有原子性如 getAndIncrement () 方法会原子性的进行增量操作把当前值加一，其它数据类型和引用变量也可以进行相似操作。

46. 如果同步块内的线程抛出异常会发生什么？

这个问题坑了很多 Java 程序员，若你能想到锁是否释放这条线索来回答还有点希望答对。无论你的同步块是正常还是异常退出的，里面的线程都会释放锁，所以对比锁接口我更喜欢同步块，因为它不用我花费精力去释放锁，该功能可以在 finally block 里释放锁实现。

47. 单例模式的双检锁是什么？

这个问题在 Java 面试中经常被问到，但是面试官对回答此问题的满意度仅为 50%。一半的人写不出双检锁还有一半的人说不出它的隐患和 Java1.5 是如何对它修正的。它其实是一个用来创建线程安全的单例的老方法，当单例实例第一次被创建时它试图用单个锁进行性能优化，但是由于太过于复杂在 JDK1.4 中它是失败的，我个人也不喜欢它。无论如何，即便你也不喜欢它但是还是要了解一下，因为它经常被问到。你可以查看 how double checked locking on Singleton works 这篇文章获得更多信息。

48. 如何在 Java 中创建线程安全的 Singleton？

这是上面那个问题的后续，如果你不喜欢双检锁而面试官问了创建 Singleton 类的替代方法，你可以利用 JVM 的类加载和静态变量初始化特征来创建 Singleton 实例，或者是利用枚举类型来创建 Singleton，我很喜欢用这种方法。你可以查看这篇文章获得更多信息。

49. 写出 3 条你遵循的多线程最佳实践

这种问题我最喜欢了，我相信你在写并发代码来提升性能的时候也会遵循某些最佳实践。以下三条最佳实践我觉得大多数 Java 程序员都应该遵循：

给你的线程起个有意义的名字。
这样可以方便找 bug 或追踪。OrderProcessor, QuoteProcessor or TradeProcessor 这种名字比 Thread-1. Thread-2 and Thread-3 好多了，给线程起一个和它要完成的任务相关的名字，所有的主要框架甚至 JDK 都遵循这个最佳实践。

避免锁定和缩小同步的范围
锁花费的代价高昂且上下文切换更耗费时间空间，试试最低限度的使用同步和锁，缩小临界区。因此相对于同步方法我更喜欢同步块，它给我拥有对锁的绝对控制权。

多用同步类少用 wait 和 notify
首先，CountDownLatch, Semaphore, CyclicBarrier 和 Exchanger 这些同步类简化了编码操作，而用 wait 和 notify 很难实现对复杂控制流的控制。其次，这些类是由最好的企业编写和维护在后续的 JDK 中它们还会不断优化和完善，使用这些更高等级的同步工具你的程序可以不费吹灰之力获得优化。

多用并发集合少用同步集合　　这是另外一个容易遵循且受益巨大的最佳实践，并发集合比同步集合的可扩展性更好，所以在并发编程时使用并发集合效果更好。如果下一次你需要用到 map，你应该首先想到用 ConcurrentHashMap。我的文章 Java 并发集合有更详细的说明。

50. 如何强制启动一个线程？

这个问题就像是如何强制进行 Java 垃圾回收，目前还没有觉得方法，虽然你可以使用 System.gc ()来进行垃圾回收，但是不保证能成功。在 Java 里面没有办法强制启动一个线程，它是被线程调度器控制着且 Java 没有公布相关的 API。

51. Java 中的 fork join 框架是什么？

fork join 框架是 JDK7 中出现的一款高效的工具，Java 开发人员可以通过它充分利用现代服务器上的多处理器。它是专门为了那些可以递归划分成许多子模块设计的，目的是将所有可用的处理能力用来提升程序的性能。fork join 框架一个巨大的优势是它使用了工作窃取算法，可以完成更多任务的工作线程可以从其它线程中窃取任务来执行。你可以查看这篇文章获得更多信息。

52. 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。你可以查看这篇文章获得更多信息。

以上就是 50 道热门 Java 多线程和并发面试题啦。我没有分享所有题的答案但给未来的阅读者提供了足够的提示和线索来寻找答案。如果你真的找不到某题的答案，联系我吧，我会加上去的。这篇文章不仅可以用来准备面试，还能检查你对多线程、并发、设计模式和竞态条件、死锁和线程安全等线程问题的理解。我打算把这篇文章的问题弄成所有 Java 多线程问题的大合集，但是没有你的帮助恐怖是不能完成的，你也可以跟我分享其它任何问题，包括那些你被问到却还没有找到答案的问题。这篇文章对初学者或者是经验丰富的 Java 开发人员都很有用，过两三年甚至五六年你再读它也会受益匪浅。它可以扩展初学者尤其有用因为这个可以扩展他们的知识面，我会不断更新这些题，大家可以在文章后面的评论中提问，分享和回答问题一起把这篇面试题完善。