Java 面试题

作者：XinyiZhang
原文链接：https://ld246.com/article/1601273641987
来源网站：链滴

<h2 id="Java线程面试">Java 线程面试</h2>

进程和线程之间有什么不同?

一个进程是一个独立(self contained)的运行环境,它可以被看作一个程序或者一个应用.而线程是进程中执行的一个任务.Java 运行环境是一个包含了不同的类和程序的单一进程.线程可以被称为轻量进程.线程需要较少的资源来创建和驻留在进程中,并且可以共享进程中的资源.
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来存储本地数据.

多线程编程的好处是什么?

在多线程程序中,多个线程被并发的执行以提高程序的效率,CPU 不会因为某个线程需要等待资源进入空闲状态.多个线程共享堆内存(heap memory),因此创建多个线程去执行一些任务会比创建多个程更好.

用户线程和守护线程有什么区别?

当我们在 Java 程序中创建一个线程,它就被称为用户线程.一个守护线程是在后台执行并且不会阻 JVM 终止的线程.当没有用户线程在运行的时候,JVM 关闭程序并且退出.一个守护线程创建的子线程然是守护线程.

我们如何创建一个线程?

实现 Runnable 接口,然后将它传递给 Thread 的构造函数,创建一个 Thread 对象;
直接继承 Thread 类.

有哪些不同的线程生命周期?

当我们在 Java 程序中新建一个线程时,它的状态是 New;
当我们调用线程的 start()方法时,状态被改变为 Runnable;
线程调度器会为 Runnable 线程池中的线程分配 CPU 时间并且将它们的状态改变为 Running;</l
>
其他的线程状态还有 Waiting、Blocked 和 Dead;

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.和直接调用 run()方法的果不一样.
当调用 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.

如何让正在运行的线程暂停一段时间?

可以使用 Thread 类的 Sleep()方法让线程暂停一段时间.需要注意的是,这并不会让线程终止,一旦休眠中唤醒线程,线程的状态将会被改变为 Runnable,并且根据线程调度,它将得到执行.

你对线程优先级的理解是什么?

每一个线程都是有优先级的,一般来说,高优先级的线程在运行时会具有优先权,但这依赖于线程调的实现,这个实现是和操作系统相关的(OS dependent).我们可以定义线程的优先级,但是这并不能保证优先级的线程会在低优先级的线程前执行.线程优先级是一个 int 变量(从 1-10),1 代表最低优先级,10 表最高优先级.

什么是线程调度器(Thread Scheduler)和时间分片(Time Slicing)?

线程调度器是一个操作系统服务,它负责为 Runnable 状态的线程分配 CPU 时间.一旦我们创建一线程并启动它,它的执行便依赖于线程调度器的实现.
时间分片是指将可用的 CPU 时间分配给可用的 Runnable 线程的过程.分配 CPU 时间可以基于程优先级或者线程等待的时间.
线程调度并不受到 Java 虚拟机控制,所以由应用程序来控制它是更好的选择(也就是说不要让你的序依赖于线程的优先级).

在多线程中，什么是上下文切换(context-switching)?

上下文切换是存储和恢复 CPU 状态的过程,它使得线程执行能够从中断点恢复执行.上下文切换是任务操作系统和多线程环境的基本特征.

你如何确保 main()方法所在的线程是 Java 程序最后结束的线程?

可以使用 Thread 类的 joint()方法来确保所有程序创建的线程在 main()方法退出前结束.

线程之间是如何通信的?

当线程间是可以共享资源时,线程间通信是协调它们的重要的手段.
Object 类中 wait()、notify()、notifyAll()方法可以用于线程间通信关于资源的锁的状态.

为什么线程通信的方法 wait(), notify()和 notifyAll()被定义在 Object 类里?

Java 的每个对象中都有一个锁(monitor,也可以称为监视器).并且 wait()、notify()等方法用于等对象的锁或者通知其他线程对象的监视器可用.在 Java 的线程中并没有可供任何对象使用的锁和同步器这就是为什么这些方法是 Object 类的一部分,这样 Java 的每一个类都有用于线程间通信的基本方法.</i>

为什么 wait(), notify()和 notifyAll()必须在同步方法或者同步块中被调用?

当一个线程需要调用对象的 wait()方法的时候,这个线程必须拥有该对象的锁,接着它就会释放这对象锁并进入等待状态直到其他线程调用这个对象上的 notify()方法.同样的,当一个线程需要调用对象 notify()方法时,它会释放这个对象的锁,以便其他在等待的线程就可以得到这个对象锁.由于所有的这方法都需要线程持有对象的锁,这样就只能通过同步来实现,所以他们只能在同步方法或者同步块中被用.

为什么 Thread 类的 sleep()和 yield()方法是静态的?

Thread 类的 sleep()和 yield()方法将在当前正在执行的线程上运行.所以在其他处于等待状态的程上调用这些方法是没有意义的.这就是为什么这些方法是静态的.它们可以在当前正在执行的线程中作,并避免程序员错误的认为可以在其他非运行线程调用这些方法.

如何确保线程安全?

在 Java 中可以有很多方法来保证线程安全——同步,使用原子类(atomic concurrent classes),实并发锁,使用 volatile 关键字,使用不变类和线程安全类.

volatile 关键字在 Java 中有什么作用?

当我们使用 volatile 关键字去修饰变量的时候,所以线程都会直接读取该变量并且不缓存它.这就保了线程读取到的变量是同内存中是一致的.

同步方法和同步块，哪个是更好的选择?

同步块是更好的选择,因为它不会锁住整个对象(当然你也可以让它锁住整个对象).同步方法会锁住个对象,哪怕这个类中有多个不相关联的同步块,这通常会导致他们停止执行并需要等待获得这个对象的锁.

如何创建守护线程?

使用 Thread 类的 setDaemon(true)方法可以将线程设置为守护线程,需要注意的是,需要在调用 s art()方法前调用这个方法,否则会抛出 IllegalThreadStateException 异常.

什么是 ThreadLocal?

ThreadLocal 用于创建线程的本地变量,我们知道一个对象的所有线程会共享它的全局变量,所以些变量不是线程安全的,我们可以使用同步技术.但是当我们不想使用同步的时候,我们可以选择 Thread ocal 变量.
每个线程都会拥有他们自己的 Thread 变量,它们可以使用 get()、set()方法去获取他们的默认值者在线程内部改变他们的值.ThreadLocal 实例通常是希望它们同线程状态关联起来是 private static 性.

什么是 Thread Group？为什么不建议使用它?

ThreadGroup 是一个类,它的目的是提供关于线程组的信息.
ThreadGroup API 比较薄弱,它并没有比 Thread 提供了更多的功能.它有两个主要的功能:

一是获取线程组中处于活跃状态线程的列表;
二是设置为线程设置未捕获异常处理器(ncaught exception handler).

但在 Java 1.5 中 Thread 类也添加了 setUncaughtExceptionHandler(UncaughtExceptionHan

ler eh)方法,所以 ThreadGroup 是已经过时的,不建议继续使用.

什么是 Java 线程转储(Thread Dump)，如何得到它?

线程转储是一个 JVM 活动线程的列表,它对于分析系统瓶颈和死锁非常有用.有很多方法可以获取程转储——使用 Profiler,Kill -3 命令,jstack 工具等等.
我更喜欢 jstack 工具,因为它容易使用并且是 JDK 自带的.由于它是一个基于终端的工具,所以我可以编写一些脚本去定时的产生线程转储以待分析.

什么是死锁(Deadlock)？如何分析和避免死锁?

死锁是指两个以上的线程永远阻塞的情况,这种情况产生至少需要两个以上的线程和两个以上的资.
分析死锁,我们需要查看 Java 应用程序的线程转储.我们需要找出那些状态为 BLOCKED 的线程和们等待的资源.每个资源都有一个唯一的 id,用这个 id 我们可以找出哪些线程已经拥有了它的对象锁.</l>
避免嵌套锁,只在需要的地方使用锁和避免无限期等待是避免死锁的通常办法.

什么是 Java Timer 类？如何创建一个有特定时间间隔的任务?

java.util.Timer 是一个工具类,可以用于安排一个线程在未来的某个特定时间执行.Timer 类可以安排一次性任务或者周期任务.
java.util.TimerTask 是一个实现了 Runnable 接口的抽象类,我们需要去继承这个类来创建我们己的定时任务并使用 Timer 去安排它的执行.

什么是线程池？如何创建一个 Java 线程池?

一个线程池管理了一组工作线程,同时它还包括了一个用于放置等待执行的任务的队列.
java.util.concurrent.Executors 提供了一个 java.util.concurrent.Executor 接口的实现用于创建程池.

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任务要 100 毫秒,那么用十个线程完成该任务只需 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.

用 Runnable 还是 Thread?

Java 不支持类的多重继承,但允许调用多个接口.所以如果要继承其他类,当然是调用 Runnable 接好.

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,Callable 是在 JDK1.5 增加的.

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.

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.

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

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

什么是线程安全? Vector 是一个线程安全类吗?

如果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.

Java 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,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.

例子就是无序处理

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.
JDK1.0 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,可以用 volatile 尔变量来退出 run()方法的循环或者是取消任务来中断线程.

一个线程运行时发生异常会怎样?

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.

如何在两个线程间共享数据?

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.

Java 中 notify 和 notifyAll 有什么区别?

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,只有一个线程在等待的时候它才有用武之地.
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.

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 也调用了 Runnable 接口所以它可以提交给 Executor 来执行.

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 isInterupted()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,一个线的中断状态有可能被其它线程调用中断来改变.

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,如果不这么做,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.

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otify()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()方法效果好的原因.

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.
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.
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技术提高了可扩展性.

Java 中堆和栈有什么不同?

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,方法参数和调用,一个线程中存储的变量对其它线程是不可见的.
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.它要求线从主存中读取变量的值.

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)

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费,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.

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

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.

Java 中活锁和死锁有什么区别?

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以改变但是却不能继续执行.

怎么检测一个线程是否拥有锁?

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体对象的锁.

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.也可以用 jstack 这个工具来获取,它对线程 id 进行操作,可以用 jps 这个工具找到 id.

JVM 中哪个参数是用来控制线程的栈堆栈小的

-Xss 参数用来控制线程的堆栈大小.

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,有一些缺点.比如不能扩展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展性.

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,可以用线程类的 join()方法在一个线程中启动另一线程,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,T 调用 T1),这样 T1 就会先完成而 T3 最后完成.

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()的线程有可能进入到暂停状态后马上又被执行.

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,这样在多线程况下就能避免争用.

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,信号量维护了一个许可合.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.

如果你提交任务时，线程池队列已满。会时发会生什么?

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 RejecteExecutionException 异常.

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, 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.

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.

Swing 是线程安全的吗？ 为什么?

Swing 不是线程安全的.
Swing 的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wing 提供了同步和异步两种回调方法来进行更新.

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.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.</li

invokeLater()方法是异步调用更新组件的.

Swing API 中那些方法是线程安全的?

repaint()、revalidate().
JTextComponent 的 setText()方法和 JTextArea 的 insert() 和 append() 方法也是线程安全的.

如何在 Java 中创建 Immutable 对象?

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.

Java 中的 ReadWriteLock 是什么?

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.写锁是占的,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 6535 个读锁.

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PU 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待重建的时间就可以使用它了.

volatile 变量和 atomic 变量有什么不同?

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.例用 volatile 修饰 count 变量那么 count++ 操作就不是原子性的.
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.

如果同步块内的线程抛出异常会发生什么?

无论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.

单例模式的双检锁是什么?

它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.

如何在 Java 中创建线程安全的 Singleton?

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.

写出 3 条你遵循的多线程最佳实践

给线程起个有意义的名字.这样可以方便找 bug 或追踪.OrderProcessor,QuoteProcessor or TraeProcessor 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 ConcurrentHashMap.

如何强制启动一个线程?

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.</l>

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线程中窃取任务来执行.

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.

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.

<h2 id="Java并发面试">Java 并发面试</h2>

什么是原子操作？在 Java Concurrency API 中有哪些原子类(atomic classes)?

原子操作是指一个不受其他操作影响的操作任务单元.原子操作是在多线程环境下避免数据不一致须的手段.
java.util.concurrent.atomic 包提供了 int 和 long 类型的装类,它们可以自动的保证对于他们的作是原子的并且不需要使用同步.

Java Concurrency API 中的 Lock 接口(Lock interface)是什么？对比同步它有什么优势?

Lock 接口比同步方法和同步块提供了更具扩展性的锁操作.他们允许更灵活的结构,可以具有完全同的性质,并且可以支持多个相关类的条件对象.它的优势有:

可以使锁更公平;
可以使线程在等待锁的时候响应中断;
可以让线程尝试获取锁,并在无法获取锁的时候立即返回或者等待一段时间;
可以在不同的范围,以不同的顺序获取和释放锁;

什么是 Executors 框架?

Executor 框架同 java.util.concurrent.Executor 接口在 Java 5 中被引入.Executor 框架是一个据一组执行策略调用,调度,执行和控制的异步任务的框架.
无限制的创建线程会引起应用程序内存溢出.所以创建一个线程池是个更好的的解决方案,因为可限制线程的数量并且可以回收再利用这些线程.
利用 Executors 框架可以非常方便的创建一个线程池.

什么是阻塞队列？如何使用阻塞队列来实现生产者-消费者模型?

java.util.concurrent.BlockingQueue 的特性是: 当队列是空的时,从队列中获取或删除元素的操将会被阻塞,或者当队列是满时,往队列里添加元素的操作会被阻塞.
阻塞队列不接受空值,当你尝试向队列中添加空值的时候,它会抛出 NullPointerException.
阻塞队列的实现都是线程安全的,所有的查询方法都是原子的并且使用了内部锁或者其他形式的并控制.
BlockingQueue 接口是 java collections 框架的一部分,它主要用于实现生产者-消费者问题.

什么是 Callable 和 Future?

Java 5 在 concurrency 包中引入了 java.util.concurrent.Callable 接口,它和 Runnable 接口很似,但它可以返回一个对象或者抛出一个异常.
Callable 接口使用泛型去定义它的返回类型.Executors 类提供了一些有用的方法去在线程池中执 Callable 内的任务.由于 Callable 任务是并行的,我们必须等待它返回的结果.
java.util.concurrent.Future 对象为我们解决了这个问题.在线程池提交 Callable 任务后返回了一 Future 对象,使用它我们可以知道 Callable 任务的状态和得到 Callable 返回的执行结果.

Future 提供了 get()方法让我们可以等待 Callable 结束并获取它的执行结果.

什么是 FutureTask?

FutureTask 是 Future 的一个基础实现,我们可以将它同 Executors 使用处理异步任务.通常我们需要使用 FutureTask 类,但当我们打算重写 Future 接口的一些方法并保持原来基础的实现时,它就变非常有用.我们可以仅仅继承于它并重写我们需要的方法.

什么是并发容器的实现?

Java 集合类都是快速失败的,这就意味着当集合被改变且一个线程在使用迭代器遍历集合的时候,代器的 next()方法将抛出 ConcurrentModificationException 异常.
并发容器支持并发的遍历和并发的更新.
主要的类有 ConcurrentHashMap、CopyOnWriteArrayList 和 CopyOnWriteArraySet.

Executors 类是什么?

Executors 为 Executor、ExecutorService、ScheduledExecutorService、ThreadFactory 和 Callable 类提供了一些工具方法.
Executors 可以用于方便的创建线程池.

