## Java多线程面试问题

http://www.cnblogs.com/dolphin0520/p/3932934.html

**1. 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。而线程是在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级进程。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。

### 2. 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而进入空闲状态。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进程更好。举个例子，Servlets比CGI更好，是因为Servlets支持多线程而CGI不支持。

### 3. 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止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依然是守护线程。

### 4. 我们如何创建一个线程？

有两种创建线程的方法：一是实现Runnable接口，然后将它传递给Thread的构造函数，创建一个Thread对象；二是直接继承Thread类。若想了解更多可以阅读这篇关于如何在[Java中创建线程](http://www.journaldev.com/1016/java-thread-example-extending-thread-class-and-implementing-runnable-interface)的文章。

### 5. 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时间并且讲它们的状态改变为Running。其他的线程状态还有Waiting，Blocked 和Dead。读这篇文章可以了解更多关于[线程生命周期](http://www.journaldev.com/1044/life-cycle-of-thread-understanding-thread-states-in-java)的知识。

### 6. 可以直接调用Thread类的run()方法么？

当然可以，但是如果我们调用了Thread的run()方法，它的行为就会和普通的方法一样，为了在新的线程中执行我们的代码，必须使用Thread.start()方法。

### 7. 如何让正在运行的线程暂停一段时间？

我们可以使用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，一旦从休眠中唤醒线程，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。

### 8. 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，但这依赖于线程调度的实现，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，但是这并不能保证高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代表最高优先级。

### 9. 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。一旦我们创建一个线程并启动它，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时间可以基于线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程序依赖于线程的优先级）。

### 10. 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。上下文切换是多任务操作系统和多线程环境的基本特征。

### 11. 你如何确保main()方法所在的线程是Java程序最后结束的线程？

我们可以使用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。这里有一篇文章关于[Thread类的joint()方法](http://www.journaldev.com/1024/java-thread-join-example-with-explanation)。

### 12.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。点击[这里](http://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example)有更多关于线程wait, notify和notifyAll.

### 13.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，也可以成为监视器) 并且wait()，notify()等方法用于等待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

### 14. 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，接着它就会释放这个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的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。由于所有的这些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调用。

### 15. 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线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工作，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。

### 16.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)，实现并发锁，使用volatile关键字，使用不变类和线程安全类。在[线程安全教程](http://www.journaldev.com/1061/java-synchronization-and-thread-safety-tutorial-with-examples)中，你可以学到更多。

### 17. 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。这就确保了线程读取到的变量是同内存中是一致的。

### 18. 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）。同步方法会锁住整个对象，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上的锁。

### 19.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start()方法前调用这个方法，否则会抛出IllegalThreadStateException异常。

### 20. 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，所以这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ThreadLocal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或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属性。在[ThreadLocal例子](http://www.journaldev.com/1076/java-threadlocal-example-to-create-thread-local-variables)这篇文章中你可以看到一个关于ThreadLocal的小程序。

### 21. 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(UncaughtExceptionHandler eh) 方法，所以ThreadGroup是已经过时的，不建议继续使用。

### 22. 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线程转储——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，所以我们可以编写一些脚本去定时的产生线程转储以待分析。读这篇文档可以了解更多关于[产生线程转储](http://www.journaldev.com/1053/how-to-generate-thread-dump-in-java)的知识。

### 23. 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源。

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的线程和他们等待的资源。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。

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，阅读这篇文章去学习[如何分析死锁](http://www.journaldev.com/1058/java-deadlock-example-and-how-to-analyze-deadlock-situation)。

### 24. 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类可以用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自己的定时任务并使用Timer去安排它的执行。

这里有关于[java Timer的例子](http://www.journaldev.com/1050/java-timer-and-timertask-example-tutorial)。

### 25. 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线程池。[线程池例子](http://www.journaldev.com/1069/java-thread-pool-example-using-executors-and-threadpoolexecutor)展现了如何创建和使用线程池，或者阅读[ScheduledThreadPoolExecutor](http://www.journaldev.com/2340/java-scheduledthreadpoolexecutor-example-to-schedule-tasks-after-delay-and-execute-periodically)例子，了解如何创建一个周期任务。

## Java并发面试问题

### 1. 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必须的手段。

int++并不是一个原子操作，所以当一个线程读取它的值并加1时，另外一个线程有可能会读到之前的值，这就会引发错误。

为了解决这个问题，必须保证增加操作是原子的，在JDK1.5之前我们可以使用同步技术来做到这一点。到JDK1.5，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操作是原子的并且不需要使用同步。可以阅读这篇文章来了解[Java的atomic类](http://www.journaldev.com/1095/java-atomic-operations-atomicinteger-example)。

### 2. 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，可以具有完全不同的性质，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

阅读更多[关于锁的例子](http://www.journaldev.com/2377/java-lock-example-and-concurrency-lock-vs-synchronized)

### 3. 什么是Executors框架？

Executor框架同java.util.concurrent.Executor 接口在Java 5中被引入。Executor框架是一个根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，因为可以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，阅读这篇文章可以了解[如何使用Executor框架创建一个线程池](http://www.journaldev.com/1069/java-thread-pool-example-using-executors-and-threadpoolexecutor)。

### 4. 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作将会被阻塞，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发控制。

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。

阅读这篇文章了解[如何使用阻塞队列实现生产者-消费者问题。](http://www.journaldev.com/1034/java-blockingqueue-example-implementing-producer-consumer-problem)

### 5. 什么是Callable和Future?

Java 5在concurrency包中引入了java.util.concurrent.Callable 接口，它和Runnable接口很相似，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行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任务后返回了一个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。

阅读这篇文章了解更多[关于Callable，Future的例子](http://www.journaldev.com/1090/java-callable-future-example)。

### 6. 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。通常我们不需要使用FutureTask类，单当我们打算重写Future接口的一些方法并保持原来基础的实现是，它就变得非常有用。我们可以仅仅继承于它并重写我们需要的方法。阅读[Java FutureTask例子](http://www.journaldev.com/1650/java-futuretask-example-program)，学习如何使用它。

### 7.什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，迭代器的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，阅读这篇文章了解[如何避免ConcurrentModificationException](http://www.journaldev.com/378/how-to-avoid-concurrentmodificationexception-when-using-an-iterator)。

### 8. 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

### 9.为什么说ConcurrentHashMap是弱一致性的？以及为何多个线程并发修改ConcurrentHashMap时不会报ConcurrentModificationException？

　　参考：<http://ifeve.com/java-concurrent-hashmap-2/>

<http://ifeve.com/concurrenthashmap-weakly-consistent/>

<http://blog.csdn.net/liuzhengkang/article/details/2916620>