# Java面试题整理

## 并发

### 1. 在java中守护线程和本地线程区别？

java 中的线程分为两种：守护线程（Daemon）和用户线程（User）。 任何线程都可以设置为守护线程和用户线程，通过方法Thread.setDaemon(boolon)；true 则把该线程设置为守护线程，反之则为用户线程。 Thread.setDaemon()必须在 Thread.start()之前调用，否则运行时会抛出异常。

两者的区别：

唯一的区别是判断虚拟机(JVM)何时离开，Daemon 是为其他线程提供服务，如果全部的 User Thread 已经撤离，Daemon 没有可服务的线程，JVM 撤离。也可以理解为守护线程是 JVM 自动创建的线程（但不一定），用户线程是程序创建的线程；比如 JVM 的垃圾回收线程是一个守护线程，当所有线程已经撤离，不再产生垃圾，守护线程自然就没事可干了，当垃圾回收线程是 Java 虚拟机上仅剩的线程时，Java 虚拟机会自动离开。

扩展：Thread Dump 打印出来的线程信息，含有 daemon 字样的线程即为守护进程，可能会有：服务守护进程、编译守护进程、windows 下的监听Ctrl+break 的守护进程、Finalizer 守护进程、引用处理守护进程、GC 守护进程。

### 2. 线程与进程的区别

进程是操作系统分配资源的最小单元，线程是操作系统调度的最小单元。 一个程序至少有一个进程,一个进程至少有一个线程。

### 3. 什么是多线程中的上下文切换

多线程会共同使用一组计算机上的 CPU，而线程数大于给程序分配的 CPU 数量时，为了让各个线程都有执行的机会，就需要轮转使用 CPU。不同的线程切换使用 CPU 发生的切换数据等就是上下文切换。

### 4. 死锁与活锁的区别，死锁与饥饿的区别？

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互相等待的现象，若无外力作用，它们都将无法推进下去。

产生死锁的必要条件：

1、互斥条件：所谓互斥就是进程在某一时间内独占资源。 2、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。 3、不剥夺条件:进程已获得资源，在末使用完之前，不能强行剥夺。 4、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。

活锁：任务或者执行者没有被阻塞，由于某些条件没有满足，导致一直重复尝试，失败，尝试，失败。

活锁和死锁的区别在于，处于活锁的实体是在不断的改变状态，所谓的“活”， 而处于死锁的实体表现为等待；活锁有可能自行解开，死锁则不能。

饥饿：一个或者多个线程因为种种原因无法获得所需要的资源，导致一直无法执行的状态。

Java 中导致饥饿的原因： 1、高优先级线程吞噬所有的低优先级线程的 CPU 时间。 2、线程被永久堵塞在一个等待进入同步块的状态，因为其他线程总是能在它之 前持续地对该同步块进行访问。 3、线程在等待一个本身也处于永久等待完成的对象(比如调用这个对象的 wait 方法)，因为其他线程总是被持续地获得唤醒。

### 5. Java 中用到的线程调度算法是什么？

采用时间片轮转的方式。可以设置线程的优先级，会映射到下层的系统上面的优先级上，如非特别需要，尽量不要用，防止线程饥饿。

### 6. 为什么使用 Executor 框架

每次执行任务创建线程 new Thread()比较消耗性能，创建一个线程是比较耗时、耗资源的。

调用 new Thread()创建的线程缺乏管理，被称为野线程，而且可以无限制的创建，线程之间的相互竞争会导致过多占用系统资源而导致系统瘫痪，还有线程之间的频繁交替也会消耗很多系统资源。

接使用 new Thread() 启动的线程不利于扩展，比如定时执行、定期执行、定时定期执行、线程中断等都不便实现。

### 7. 在 Java 中 Executor 和 Executors 的区别

Executors 工具类的不同方法按照我们的需求创建了不同的线程池，来满足业务的需求。 Executor 接口对象能执行我们的线程任务。 ExecutorService 接口继承了 Executor 接口并进行了扩展，提供了更多的方法我们能获得任务执行的状态并且可以获取任务的返回值。 使用 ThreadPoolExecutor 可以创建自定义线程池。 Future 表示异步计算的结果，他提供了检查计算是否完成的方法，以等待计算的完成，并可以使用 get()方法获取计算的结果。

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

原子操作（atomic operation）意为”不可被中断的一个或一系列操作” 。 处理器使用基于对缓存加锁或总线加锁的方式来实现多处理器之间的原子操作。在 Java 中可以通过锁和循环 CAS 的方式来实现原子操作。 CAS 操作——Compare & Set，或是 Compare & Swap，现在几乎所有的 CPU 指令都支持CAS 的原子操作。

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 类型的原子包装类，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。

java.util.concurrent 这个包里面提供了一组原子类。其基本的特性就是在多线程环境下，当有多个线程同时执行这些类的实例包含的方法时，具有排他性，即当某个线程进入方法，执行其中的指令时，不会被其他线程打断，而别的线程就像自旋锁一样，一直等到该方法执行完成，才由 JVM 从等待队列中选择一个另一个线程进入，这只是一种逻辑上的理解。

原子类：AtomicBoolean，AtomicInteger，AtomicLong，AtomicReference 原子数组：AtomicIntegerArray，AtomicLongArray，AtomicReferenceArray 原子属性更新器：AtomicLongFieldUpdater，AtomicIntegerFieldUpdater，AtomicReferenceFieldUpdater 解决 ABA 问题的原子类：AtomicMarkableReference（通过引入一个 boolean来反映中间有没有变过），AtomicStampedReference（通过引入一个 int 来累加来反映中间有没有变过）

### 9.