面试题——多线程

1. 并行和并发有什么区别？

答：并发在单核和多核都可存在，即同一时间有多个可以执行的进程。但在单核中同一时刻只有一个进程获得CPU，并行是指同一时间多个进程在微观上都在真正的执行，这就只有在多核的情况下了。

2. 线程和进程的区别？

答：线程：是程序执行流的最小单元，是系统独立调度和分配CPU（独立运行）的基本单位

进程：是资源分配的基本单位。一个进程包括多个线程。

区别：地址空间、资源拥有

线程与资源分配无关，它属于某一个进程，并与进程内的其他线程一起共享进程的资源

每个进程都有自己一套独立的资源（数据），供其内的所有线程共享

不论是大小，开销线程要更“轻量级”

一个进程内的线程通信比进程之间的通信更快速，有效。（因为共享变量）

3. 守护线程是什么？

答：守护线程是个服务线程，服务于其他线程

典型案例：垃圾回收线程

4. 创建线程有哪几种方式？

答：继承Thread类创建线程

实现Runnable接口创建线程

通过Callable和Future创建线程

5. 说一下 runnable 和 callable 有什么区别？

答：runnable 没有返回值，callable 可以拿到有返回值，callable 可以看作是 runnable 的补充。

6. 线程有哪些状态？

答：创建、就绪、运行、阻塞、死亡

7. sleep()和wait() 有什么区别？

答：sleep() 可以在任何地方使用

wait() 只能在同步方法或同步块中使用

8. notify()和notifyAll()有什么区别？

答：notify是唤醒某个线程

notifyAll是唤醒所有暂停线程

9. 线程的 run()和 start()有什么区别？

答：run() 相当于线程的任务处理逻辑的入口方法

start() 的作用是启动相应的线程

10. 创建线程池有哪几种方式？

答：线程池创建有四种常见方式：newSingleThreadExecutor，newCachedThreadPool，

newFixedThreadPool，newScheduledThreadPool

newSingleThreadExecutor()：它的特点在于工作线程数目被限制为1，操作一个无界的工作队列，所以它保证了所有任务的都是被顺序执行，最多会有一个任务处于活动状态，并且不允许使用者改动线程池实例，因此可以避免其改变线程数目；

newCachedThreadPool()：它是一种用来处理大量短时间工作任务的线程池，具有几个鲜明特点：它会试图缓存线程并重用，当无缓存线程可用时，就会创建新的工作线程；如果线程闲置的时间超过 60 秒，则被终止并移出缓存；长时间闲置时，这种线程池，不会消耗什么资源。其内部使用 SynchronousQueue 作为工作队列；

newFixedThreadPool(int nThreads)：重用指定数目（nThreads）的线程，其背后使用的是无界的工作队列，任何时候最多有 nThreads 个工作线程是活动的。这意味着，如果任务数量超过了活动队列数目，将在工作队列中等待空闲线程出现；如果有工作线程退出，将会有新的工作线程被创建，以补足指定的数目 nThreads；

newScheduledThreadPool(int corePoolSize)：和newSingleThreadScheduledExecutor()类似，创建的是个 ScheduledExecutorService，可以进行定时或周期性的工作调度，区别在于单一工作线程还是多个工作线程；

11. 线程池都有哪些状态？

答：RUNNING：这是最正常的状态，接受新的任务，处理等待队列中的任务。

SHUTDOWN：不接受新的任务提交，但是会继续处理等待队列中的任务。

STOP：不接受新的任务提交，不再处理等待队列中的任务，中断正在执行任务的线程。

TIDYING：所有的任务都销毁了，workCount 为 0，线程池的状态在转换为 TIDYING 状态时，会执行钩子方法 terminated()。

TERMINATED：terminated()方法结束后，线程池的状态就会变成这个。

12. 线程池中 submit()和 execute()方法有什么区别？

答：execute()：只能执行 Runnable 类型的任务。

submit()：可以执行 Runnable 和 Callable 类型的任务。

13. 在 java 程序中怎么保证多线程的运行安全？

答：方法一：使用安全类，比如 Java. util. concurrent 下的类。

方法二：使用自动锁 synchronized。

方法三：使用手动锁 Lock。

14. 多线程锁的升级原理是什么？

答：synchronized 锁升级原理：在锁对象的对象头里面有一个 threadid 字段，在第一次访问的时候 threadid 为空，jvm 让其持有偏向锁，并将 threadid 设置为其线程 id，再次进入的时候会先判断 threadid 是否与其线程 id 一致，如果一致则可以直接使用此对象，如果不一致，则升级偏向锁为轻量级锁，通过自旋循环一定次数来获取锁，执行一定次数之后，如果还没有正常获取到要使用的对象，此时就会把锁从轻量级升级为重量级锁，此过程就构成了 synchronized 锁的升级。

锁的升级的目的：锁升级是为了减低了锁带来的性能消耗。在Java 6之后优化 synchronized 的实现方式，使用了偏向锁升级为轻量级锁再升级到重量级锁的方式，从而减低了锁带来的性能消耗。

15. 什么是死锁？

答：当线程 A 持有独占锁a，并尝试去获取独占锁 b 的同时，线程 B 持有独占锁 b，并尝试获取独占锁 a 的情况下，就会发生 AB 两个线程由于互相持有对方需要的锁，而发生的阻塞现象，我们称为死锁。

16. 怎么防止死锁？

答：尽量使用tryLock(long timeout, TimeUnit unit)

的方法(ReentrantLock、ReentrantReadWriteLock)，设置超时时间，超时可以退出防止死锁。

尽量使用 Java. util. concurrent 并发类代替自己手写锁。

尽量降低锁的使用粒度，尽量不要几个功能用同一把锁。

尽量减少同步的代码块。

17. ThreadLocal 是什么？有哪些使用场景？

答：ThreadLocal用于保存某个线程共享变量。使用场景：解决数据库连接，Session管理。

18. 说一下 synchronized 底层实现原理？

答： synchronized 是由一对 monitorenter/monitorexit 指令实现的，monitor 对象是同步的基本实现单元。在 Java 6 之前，monitor 的实现完全是依靠操作系统内部的互斥锁，因为需要进行用户态到内核态的切换，所以同步操作是一个无差别的重量级操作，性能也很低。但在 Java 6 的时候，Java 虚拟机对此进行了大刀阔斧地改进，提供了三种不同的 monitor 实现，也就是常说的三种不同的锁：偏向锁（Biased Locking）、轻量级锁和重量级锁，大大改进了其性能。

19. synchronized 和 volatile 的区别是什么？

答：volatile 是变量修饰符；synchronized 是修饰类、方法、代码段。

volatile 仅能实现变量的修改可见性，不能保证原子性；而 synchronized 则可以保证变量的修改可见性和原子性。

volatile 不会造成线程的阻塞；synchronized 可能会造成线程的阻塞。

20. synchronized 和 Lock 有什么区别？

答：synchronized 可以给类、方法、代码块加锁；而 lock 只能给代码块加锁。

synchronized 不需要手动获取锁和释放锁，使用简单，发生异常会自动释放锁，不会造成死锁；而 lock 需要自己加锁和释放锁，如果使用不当没有 unLock()去释放锁就会造成死锁。

通过 Lock 可以知道有没有成功获取锁，而 synchronized 却无法办到。

21. synchronized 和 ReentrantLock 区别是什么？

答：主要区别如下：

ReentrantLock 使用起来比较灵活，但是必须有释放锁的配合动作；

ReentrantLock 必须手动获取与释放锁，而 synchronized 不需要手动释放和开启锁；

ReentrantLock 只适用于代码块锁，而 synchronized 可用于修饰方法、代码块等。

22. 说一下 atomic 的原理？

答：atomic 主要利用 CAS (Compare And Wwap) 和 volatile 和 native 方法来保证原子操作，从而避免 synchronized 的高开销，执行效率大为提升。