线程、多线程和线程池面试专题

1、开启线程的三种方式？

1）继承 Thread 类，重写 run()方法，在 run()方法体中编写要完成的任务 new Thread().start();

2）实现 Runnable 接口，实现 run()方法 new Thread(new MyRunnable()).start();

3）实现 Callable 接口 MyCallable 类，实现 call()方法，使用 FutureTask 类来包装 Callable 对象，使用 FutureTask 对象作为 Thread 对象的 target 创建并启动线程；调用 FutureTask 对象的 get()方法来获得子线程执行结束后的返回值。

FutureTask<Integer> ft = new FutureTask<Integer>(new MyCallable());

new Thread(ft).start();

2、run()和 start()方法区别

run()方法只是线程的主体方法，和普通方法一样，不会创建新的线程。只有调用 start()方法，才会启动一个新的线程，新线程才会调用 run()方法，线程才会开始执行。

3、如何控制某个方法允许并发访问线程的个数？

创建 Semaphore 变量，Semaphore semaphore = new Semaphore(5, true);

当方法进入时，请求一个信号，如果信号被用完则等待，方法运行完，释放一个信号，释放的信号新的线程就可以使用。

4、在 Java 中 wait 和 seelp 方法的不同

wait()方法属于 Object 类，调用该方法时，线程会放弃对象锁，只有该对象调用 notify()方法后本线程才进入对象锁定池准备获取对象锁进入运行状态。

sleep()方法属于 Thread 类，sleep()导致程序暂停执行指定的时间，让出 CPU，但它的监控状态依然保存着，当指定时间到了又会回到运行状态，sleep()方法中线程不会释放对象锁。

5、谈谈 wait/notify 关键字的理解

notify: 唤醒在此对象监视器上等待的单个线程

notifyAll(): 通知所有等待该竞争资源的线程

wait: 释放 obj 的锁 ,导致当前的线程等待 ,直接其他线程调用此对象的 notify()或 notifyAll()方法

当要调用 wait()或 notify()/notifyAll()方法时，一定要对竞争资源进行加锁，一

般放到 synchronized(obj)代码中。当调用 obj.notify/notifyAll 后，调用线程依旧持有 obj 锁，因此等待线程虽被唤醒，但仍无法获得 obj 锁，直到调用线程退出 synchronized 块 ,释放 obj 锁后 ,其他等待线程才有机会获得锁继续执行。

6、什么导致线程阻塞？

（1）一般线程阻塞

1）线程执行了 Thread.sleep(int millsecond)方法，放弃 CPU，睡眠一段时间，一段时间过后恢复执行；

2）线程执行一段同步代码，但无法获得相关的同步锁，只能进入阻塞状态，等到获取到同步锁，才能恢复执行；

3）线程执行了一个对象的 wait()方法，直接进入阻塞态，等待其他线程执行 notify()/notifyAll()操作；

4）线程执行某些 IO 操作，因为等待相关资源而进入了阻塞态，如 System.in，但没有收到键盘的输入，则进入阻塞态。

5）线程礼让，Thread.yield()方法，暂停当前正在执行的线程对象，把执行机会让给相同或更高优先级的线程，但并不会使线程进入阻塞态，线程仍处于可执行态，随时可能再次分得 CPU 时间。线程自闭，join()方法，在当前线程调用另一

个线程的 join()方法，则当前线程进入阻塞态，直到另一个线程运行结束，当前线程再由阻塞转为就绪态。

6）线程执行 suspend()使线程进入阻塞态，必须 resume()方法被调用，才能使线程重新进入可执行状态。

7、线程如何关闭？

1) 使用标志位

2）使用 stop()方法，但该方法就像关掉电脑电源一样，可能会发生预料不到的问题

3）使用中断 interrupt()

```
public class Thread {
    // 中断当前线程
    public void interrupt();
    // 判断当前线程是否被中断
    public boolen isInterrupt();
    // 清除当前线程的中断状态，并返回之前的值
    public static boolen interrupted();
}
```

但调用 interrupt()方法只是传递中断请求消息，并不代表要立马停止目标线程。

8、讲一下 java 中的同步的方法

之所以需要同步，因为在多线程并发控制，当多个线程同时操作一个可共享的资源时，如果没有采取同步机制，将会导致数据不准确，因此需要加入同步锁，确保在该线程没有完成操作前被其他线程调用，从而保证该变量的唯一性和准确性。

1）synchronized 修饰同步代码块或方法

由于 java 的每个对象都有一个内置锁，用此关键字修饰方法时，内置锁会保护整个方法。在调用该方法前，需获得内置锁，否则就处于阴塞状态。

2）volatile 修饰变量

保证变量在线程间的可见性，每次线程要访问 volatile 修饰的变量时都从内存中读取，而不缓存中，这样每个线程访问到的变量都是一样的。且使用内存屏障。

3）ReentrantLock 重入锁，它常用的方法有 ReentrantLock()：创建一个 ReentrantLock 实例

lock()获得锁 unlock()释放锁

4）使用局部变量 ThreadLocal 实现线程同步，每个线程都会保存一份该变量的副本，副本之间相互独立，这样每个线程都可以随意修改自己的副本，而不影响其他线程。常用方法 ThreadLocal()创建一个线程本地变量；get()返回此线程局部的当前线程副本变量；initialValue()返回此线程局部变量的当前线程的初始值；set(T value)将此线程变量的当前线程副本中的值设置为 value

5) 使用原子变量，如 AtomicInteger，常用方法 AtomicInteger(int value)创建个有给定初始值的 AtomicInteger 整数；addAndGet(int data)以原子方式将给定值与当前值相加

6）使用阻塞队列实现线程同步 LinkedBlockingQueue<E>

9、如何保证线程安全？

线程安全性体现在三方法：

1）原子性：提供互斥访问，同一时刻只能有一个线和至数据进行操作。

JDK 中提供了很多 atomic 类，如 AtomicInteger\AtomicBoolean\AtomicLong，它们是通过 CAS 完成原子性。JDK 提供锁分为两种：synchronized 依赖 JVM 实现锁，该关键字作用对象的作用范围内同一时刻只能有一个线程进行操作。另一种是 LOCK,是 JDK 提供的

代码层面的锁，依赖 CPU 指令，代表性是 ReentrantLock。

2）可见性：一个线程对主内存的修改及时被其他线程看到。

JVM 提供了 synchronized 和 volatile，volatile 的可见性是通过内存屏障和禁止重排序实现的，volatile 会在写操作时，在写操作后加一条 store 屏障指令，将本地内存中的共享变量值刷新到主内存；会在读操作时，在读操作前加一条 load 指令，从内存中读取共享变量。

3）有序性：指令没有被编译器重排序。

可通过 volatile、synchronized、Lock 保证有序性。

10、两个进程同时要求写或者读，能不能实现？如何防止进程的同步？

我认为可以实现，比如两个进程都读取日历进程数据是没有问题，但同时写，应该会有冲突。

可以使用共享内存实现进程间数据共享。

11、线程间操作 List

12、Java 中对象的生命周期

1）创建阶段（Created）：为对象分配存储空间，开始构造对象，从超类到子类对 static 成员初始化 ,超类成员变量按顺序初始化 ,递归调用超类的构造方法，子类成员变量按顺序初始化，子类构造方法调用。

2）应用阶段（In Use）：对象至少被一个强引用持有着。

3）不可见阶段（Invisible）：程序运行已超出对象作用域

4）不可达阶段（Unreachable）：该对象不再被强引用所持有

5）收集阶段（Collected）：假设该对象重写了 finalize()方法且未执行过，会去执行该方法。

6）终结阶段（Finalized）：对象运行完 finalize()方法仍处于不可达状态，等待垃圾回收器对该对象空间进行回收。

7）对象空间重新分配阶段（De-allocated）：垃圾回收器对该对象所占用的内存空间进行回收或再分配，该对象彻底消失。

13、static synchronized 方法的多线程访问和作用

static synchronized 控制的是类的所有实例访问，不管 new 了多少对象，只有

一份，所以对该类的所有对象都加了锁。限制多线程中该类的所有实例同时访问 JVM 中该类对应的代码。

14、同一个类里面两个 synchronized 方法，两个线程同时访问的问题

如果 synchronized 修饰的是静态方法，锁的是当前类的 class 对象，进入同步代码前要获得当前类对象的锁；

普通方法，锁的是当前实例对象，进入同步代码前要获得的是当前实例的锁；

同步代码块，锁的是括号里面的对象，对给定的对象加锁，进入同步代码块库前要获得给定对象锁；

如果两个线程访问同一个对象的 synchronized 方法，会出现竞争，如果是不同对象，则不会相互影响。

15、volatile 的原理

有 volatile 变量修饰的共享变量进行写操作的时候会多一条汇编代码，lock addl $0x0，lock 前缀的指令在多核处理器下会将当前处理器缓存行的数据会写回到系统内存，这个写回内存的操作会引起在其他 CPU 里缓存了该内存地址的数据无效。同时 lock 前缀也相当于一个内存屏障，对内存操作顺序进行了限制。

16、synchronized 原理

synchronized 通过对象的对象头（markword)来实现锁机制，java 每个对象都有对象头，都可以为 synchronized 实现提供基础，都可以作为锁对象，在字节码层面 synchronized 块是通过插入 monitorenter monitorexit 完成同步的。持有 monitor 对象，通过进入、退出这个 Monitor 对象来实现锁机制。

17、谈谈 NIO 的理解

NIO(New Input/ Output) 引入了一种基于通道和缓冲区的 I/O 方式，它可以使用 Native 函数库直接分配堆外内存，然后通过一个存储在 Java 堆的 DirectByteBuffer 对象作为这块内存的引用进行操作，避免了在 Java 堆和 Native 堆中来回复制数据。 NIO 是一种同步非阻塞的 IO 模型。同步是指线程不断轮询 IO 事件是否就绪，非阻塞是指线程在等待 IO 的时候，可以同时做其他任务。同步的核心就是 Selector，Selector 代替了线程本身轮询 IO 事件，避免了阻塞同时减少了不必要的线程消耗，非阻塞的核心就是通道和缓冲区，当 IO 事件就绪时，可以通过写道缓冲区，保证 IO 的成功，而无需线程阻塞式地等待。

-synchronized 和 volatile 关键字的区别

-synchronized 与 Lock 的区别

-ReentrantLock 、synchronized 和 volatile 比较

1）volatile：解决变量在多个线程间的可见性，但不能保证原子性，只能用于修饰变量，不会发生阻塞。volatile 能屏蔽编译指令重排，不会把其后面的指令排到内存屏障之前的位置，也不会把前面的指令排到内存屏障的后面。多用于并行

计算的单例模式。volatile 规定 CPU 每次都必须从内存读取数据，不能从 CPU 缓存中读取，保证了多线程在多 CPU 计算中永远拿到的都是最新的值。

2）synchronized：互斥锁，操作互斥，并发线程过来，串行获得锁，串行执行代码。解决的是多个线程间访问共享资源的同步性，可保证原子性，也可间接保证可见性，因为它会将私有内存和公有内存中的数据做同步。可用来修饰方法、代码块。会出现阻塞。synchronized 发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生。非公平锁，每次都是相互争抢资源。

3）lock 是一个接口，而 synchronized 是 java 中的关键字，synchronized 是内置语言的实现。lock 可以让等待锁的线程响应中断。在发生异常时，如果没有主动通过 unLock()去释放锁，则可能造成死锁现象，因此使用 Lock 时需要在 finally 块中释放锁。

4）ReentrantLock 可重入锁，锁的分配机制是基于线程的分配，而不是基于方法调用的分配。ReentrantLock 有 tryLock 方法，如果锁被其他线程持有，返回 false，可避免形成死锁。对代码加锁的颗粒会更小，更节省资源，提高代码性能。ReentrantLock 可实现公平锁和非公平锁，公平锁就是先来的先获取资源。ReentrantReadWriteLock 用于读多写少的场合，且读不需要互斥场景。

-ReentrantLock 的内部实现

https://www.cnblogs.com/xrq730/p/4979021.html

-lock 原理

https://blog.csdn.net/qpzkobe/article/details/78586619

-死锁的四个必要条件？

-怎么避免死锁？

-对象锁和类锁是否会互相影响？

-什么是线程池，如何使用?

https://blog.csdn.net/wolf909867753/article/details/77500625

-Java 的并发、多线程、线程模型

https://blog.csdn.net/dengyuaner/article/details/80053941

https://blog.csdn.net/u010843421/article/details/80905625

-谈谈对多线程的理解

https://blog.csdn.net/dongmeng1994/article/details/54586466

-多线程有什么要注意的问题？

https://blog.csdn.net/saizo123/article/details/64543853

-谈谈你对并发编程的理解并举例说明

https://blog.csdn.net/maozhr720/article/details/76017671

https://blog.csdn.net/u012891504/article/details/51332298

-谈谈你对多线程同步机制的理解？

https://blog.csdn.net/u012891504/article/details/51332298

-如何保证多线程读写文件的安全？

https://blog.csdn.net/erice_e/article/details/72845892

-多线程断点续传原理、断点续传的实现

https://www.cnblogs.com/wangzehuaw/p/5610851.html

（五）并发编程有关知识点（这个是一般 Android 开发用的少的，所以建议多去看看）：

平时 Android 开发中对并发编程可以做得比较少，Thread 这个类经常会用到，但是我们想提升自己的话，一定不能停留在表面，,我们也应该去了解一下 java 的关于线程相关的源码级别的东西。