# **1、并发**

### 1、为什么使用并发编程，优点？

充分利用多核CPU的计算能力：通过并发编程的形式可以将多核CPU的计算能力发挥到极致，性能得到提升；

方便进行业务拆分，提升系统并发能力和性能：在特殊的业务场景下，先天的就适合于并发编程。现在的系统动不动就要求百万级甚至千万级的并发量，而多线程并发编程正是开发高并发系统的基础，利用好多线程机制可以大大提高系统整体的并发能力以及性能。面对复杂业务模型，并行程序会比串行程序更适应业务需求，而并发编程更能吻合这种业务拆分

### 2、并发编程的缺点？

并发编程的目的就是为了能提高程序的执行效率，提高程序运行速度，但是并发编程并不总是能提高程序运行速度的，而且并发编程可能会遇到很多问题，比如：内存泄漏、上下文切换、线程安全、死锁等问题

### 3、并发编程的三要素？Java中如何保证多线程运行安全？

并发编程三要素（线程的安全性问题体现在）：

原子性：原子，即一个不可再被分割的颗粒。原子性指的是一个或多个操作要么全部执行成功要么全部执行失败。

可见性：一个线程对共享变量的修改,另一个线程能够立刻看到。（synchronized,volatile）

有序性：程序执行的顺序按照代码的先后顺序执行。（处理器可能会对指令进行重排序）

出现线程安全问题的原因：  
线程切换带来的原子性问题、缓存导致的可见性问题、编译优化带来的有序性问题

解决办法：JDK Atomic开头的原子类、synchronized、LOCK，可以解决原子性问题， synchronized（执行前加载变量最新值，结束前将变量值刷新回主存）、 volatile、LOCK（lock在unlock操作前会把工作内存的所有改变同步到主内存，实现机制是state被volatile修饰具体解释lock中有），可以解决可见性问题

### 4、并发与并行的区别？

**并发**：多个任务在同一个CPU核上，按细分的时间片轮流(交替)执行，从逻辑上来看那些任务是同时执行。

**并行**：单位时间内，多个处理器或多核处理器同时处理多个任务，是真正意义上的“同时进行”。

**串行**：有n个任务，由一个线程按顺序执行。由于任务、方法都在一个线程执行所以不存在线程不安全情况，也就不存在临界区的问题。

做一个形象的比喻：  
并发 = 两个队列和一台咖啡机。  
并行 = 两个队列和两台咖啡机。  
串行 = 一个队列和一台咖啡机。

### 5、什么是多线程？多线程的优劣？

多线程：多线程是指程序中包含多个执行流，即在一个程序中可以同时运行多个不同的线程来执行不同的任务。

多线程的好处：  
可以提高 CPU 的利用率。在多线程程序中，一个线程必须等待的时候，CPU可以运行其它的线程而不是等待，这样就大大提高了程序的效率。也就是说允许单个程序创建多个并行执行的线程来完成各自的任务。

多线程的劣势：  
线程也是程序，所以线程需要占用内存，线程越多占用内存也越多；  
多线程需要协调和管理，所以需要CPU时间跟踪线程；

现成的阻塞与唤醒需要在用户态与内核态相互在转换，十分消耗性能；  
线程之间对共享资源的访问会相互影响，必须解决竞用共享资源的问。

# 2、线程

### 1、线程与进程的区别？

**进程**：一个在内存中运行的应用程序。每个进程都有自己独立的一块内存空间，一个进  
程可以有多个线程，比如在Windows系统中，一个运行的xx.exe就是一个进程。

**线程**：进程中的一个执行任务（控制单元），负责当前进程中程序的执行。一个进程至  
少有一个线程，一个进程可以运行多个线程，多个线程可共享数据。

### 2、线程与进程的区别？

线程具有许多传统进程所具有的特征，故又称为轻型进程(Light—Weight Process)或进程元； 而把传统的进程称为重型进程(Heavy—Weight Process)，  
它相当于只有一个线程的任务。在引入了线程的操作系统中，通常一个进程都有若干个线程， 至少包含一个线程。

**根本区别**：进程是操作系统资源分配的基本单位，而线程是处理器任务调度和执行的基本单位

**资源开销**：

每个进程都有独立的代码和数据空间（程序上下文），程序之间的切换会有较大的开销；

线程可以看做轻量级的进程，同一类线程共享代码和数据空间，每个线程都有自己独立的运行栈和程序计数器（PC），线程之间切换的开销小。

**包含关系**：如果一个进程内有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的；线程是进程的一部分，所以线程也被称为轻权进程或者轻量级进程。

**内存分配**：同一进程的线程共享本进程的地址空间和资源，而进程之间的地址空  
间和资源是相互独立的

**影响关系**：一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个  
线程崩溃整个进程都死掉。所以多进程要比多线程健壮。

### 3、什么是上线文切换？

**具体原理**：Jvm创建线程对象后，调用start（）方法后会通过底层native方法在内核创建一个与之对应的内核线程，只有内核才有分配时间片的权限，线程CPU时间片使用完之后，会把寄存器与缓存中的数据回写到主存的内核空间里栈空间的TSS任务状态段，当线程再次获得时间片时从内核空间复制数据回寄存器与缓存中，线程继续执行，CPU寄存器，其实就是来控制代码段和数据段的指令及数据读取的地方，Java运行时的数据应该是在用户空间

多线程编程中一般线程的个数都大于CPU核心的个数，而一个CPU核心在任意时刻只能被一个线程使用，为了让这些线程都能得到有效执行，CPU采取的策略是为每个线程分配时间片并轮转的形式。当一个线程的时间片用完的时候就会重新处于就绪状态让给其他线程使用，这个过程就属于一次上下文切换。

概括来说就是：当前任务在执行完 CPU 时间片切换到另一个任务之前会先将寄存器与缓存中的数据回写到主存的内核空间里栈空间的TSS任务状态段，以便下次再切换回这个任务时，可以从内核空间复制数据回寄存器与其他缓存中。任务从保存到再加载的过程就是一次上下文切换。

上下文切换通常是计算密集型的。也就是说，它需要相当可观的处理器时间，在每秒几十上百次的切换中，每次切换都需要纳秒量级的时间。所以，上下文切换对系统来说意味着消耗大量的CPU时间，事实上，可能是操作系统中时间消耗最大的操作。

Linux相比与其他操作系统（包括其他类Unix系统）有很多的优点，其中有一项就是，其上下文切换和模式切换的时间消耗非常少。

### 4、守护线程和用户线程有什么区别呢？

守护线程和用户线程  
**用户 (User) 线程**：运行在前台，执行具体的任务，如程序的主线程、连接网络的子线程等都是用户线程  
**守护 (Daemon) 线程**：运行在后台，由jvm创建，为其他前台线程服务。也可以说守护线程是JVM中非守护线程的“佣人”。一旦所有用户线程都结束运行，守护线程会随 JVM一起结束工作main 函数所在的线程就是一个用户线程啊，main 函数启动的同时在 JVM内部同时还启动了好多守护线程，比如垃圾回收线程。

比较明显的区别之一是用户线程结束，JVM 退出，不管这个时候有没有守护线程运行。而守护线程不会影响JVM的退出。

注意事项：  
1 . setDaemon(true)必须在start()方法前执行， 否则会抛出IllegalThreadStateException 异常  
2. 在守护线程中产生的新线程也是守护线程  
3. 不是所有的任务都可以分配给守护线程来执行， 比如读写操作或者计算逻辑  
4. 守护(Daemon)线程中不能依靠finally块的内容来确保执行关闭或清理资源的逻辑。因为我们上面也说过了一旦所有用户线程都结束运行，守护线程会随 JVM 一起结束工作，所以守护(Daemon)线程中的finally语句块可能无法被执行

**应用：**

Tomcat中应用守护线程机制，如果虚拟机中用户线程全部关闭，则tomcat也会关闭，所以在controller内创建的线程也为守护线程，因为母线程是守护线程，故子线程也为守护线程

可通过setDaemon(false)指定为用户线程

下面代码可打印守护线程信息

Set<Thread> threadSet = Thread.getAllStackTraces().keySet();  
   for(Thread thread: threadSet) {  
       System.out.println("dumpAllThreadsInfo thread.name=" + thread.getName()  
               + ";group=" + thread.getThreadGroup()  
               + ";isDaemon=" + thread.isDaemon()  
               + ";priority=" + thread.getPriority());  
   }

### 5、生成死锁的四个必要条件是？

**互斥条件**：线程(进程)对于所分配到的资源具有排它性，即一个资源只能被一个线程(进程)占用，直到被该线程(进程)释放

**请求与保持条件**：一个线程(进程)因请求被占用资源而发生阻塞时，对已获得的资源保持不放。  
**不剥夺条件**：线程(进程)已获得的资源在末使用完之前不能被其他线程强行剥夺，只有自己使用完毕后才释放资源。

**循环等待条件**：当发生死锁时，所等待的线程(进程)必定会形成一个环路（类似于死循环），造成永久阻塞

### 6、创建线程的四种方式

创建线程有四种方式：

继承 Thread 类；

实现 Runnable 接口；

实现 Callable 接口；

使用 Executors 工具类创建线程池

继承 Thread 类  
步骤  
1 、定义一个Thread类的子类，重写run方法，将相关逻辑实现，run()方法就是线程要执行的业务逻辑方法  
2、创建自定义的线程子类对象  
3、调用子类实例的star()方法来启动线程

实现Runnable接口  
步骤  
1、定义Runnable接口实现类MyRunnable， 并重写run()方法  
2.、创建MyRunnable实例myRunnable， 以myRunnable作为target创建Thead对象， 该Thread对象才是真正的线程对象  
3、调用线程对象的start()方法

实现 Callable 接口  
步骤  
1、创建实现Callable接口的类myCallable  
2、以myCallable为参数创建FutureTask对象  
3、将FutureTask作为参数创建Thread对象  
4、调用线程对象的start()方法

FutureTask获取结果时会阻塞

### 7、runnable 和 callable 有什么区别？

**相同点**：都是接口、都可以编写多线程程序、都采用Thread.start()启动线程；

**主要区别：**

Runnable接口run方法无返回值，Callable接口call 方法有返回值，是个泛 型，与Future、FutureTask配合可以用来获取异步执行的结果 ；

Runnable接口run方法只能抛出运行时异常，且无法捕获处理；Callable 接口call方法允许抛出异常，可以获取异常信息

注：Callalbe接口支持返回执行结果，需要调用FutureTask.get()得到，此方法会阻塞主进程的继续往下执行，如果不调用不会阻塞。

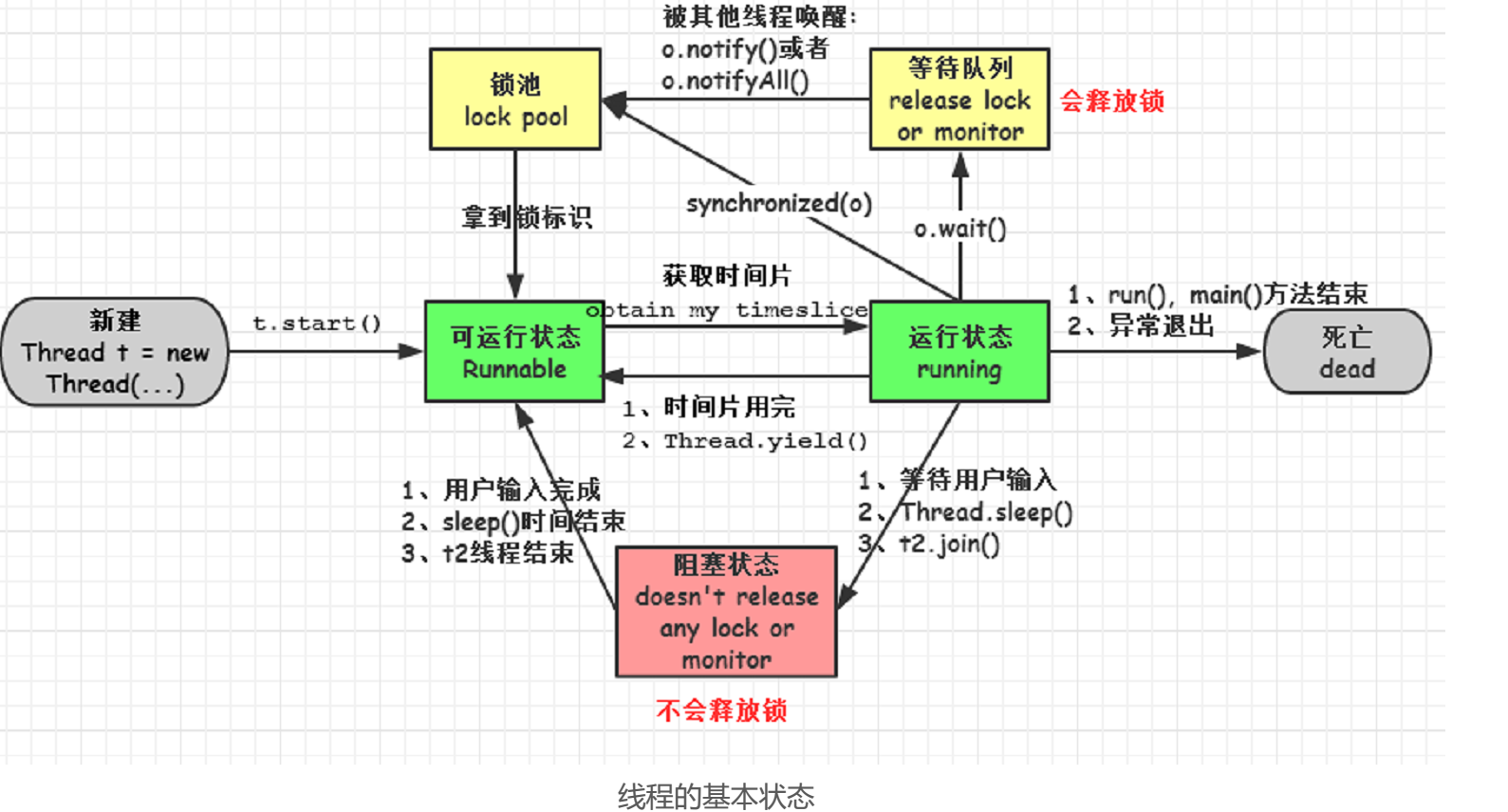
### 8、线程的 run()和 start()有什么区别？

调用run方法只是相当于main主线程调用一个普通方法，在方法执行完成之前不能执行其他代码，start会启动线程，调用后线程进入就绪状态，无需等待run方法执行完毕主线程即可执行其它代码，run方法可以执行多次，start方法只能执行一次

### 9、什么是 FutureTask

FutureTask 表示一个异步运算的任务。FutureTask 里面可以传入一个 Callable的具体实现类， 可以对这个异步运算的任务的结果进行等待获取、判断是否已经完成、取消任务等操作。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方法将会阻塞。一个 FutureTask 对象可以对调用了Callable和Runnable 的对象进行包装，由于FutureTask也是Runnable接口的实现类，所以FutureTask也可以放入线程池中

### 10、线程的生命周期及五种基本状态？



1 、新建(new)：新创建了一个线程对象。  
2. 可运行(runnable)：线程对象创建后，当调用线程对象的 start()方法，  
该线程处于就绪状态，等待被线程调度选中，获取cpu的使用权。  
3. 运行(running)：可运行状态(runnable)的线程获得了cpu时间片（timeslice），执行程序代码。注：就绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行， 首先必须处于就绪状态中；  
4. 阻塞(block)：处于运行状态中的线程由于某种原因，暂时放弃对 CPU的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才有机会再次被 CPU 调用以进入到运行状态。

阻塞的情况分三种：  
(一). 等待阻塞：运行状态中的线程执行 wait()方法，JVM会把该线程放入等待队  
列(waitting queue)中，使本线程进入到等待阻塞状态；  
(二). 同步阻塞：线程在获取 synchronized 同步锁失败(因为锁被其它线程所占用)，则JVM会把该线程放入锁池(lock pool)中，线程会进入同步阻塞状态；  
(三). 其他阻塞: 通过调用线程的 sleep()或 join()或发出了 I/O 请求时， 线程会进入到阻塞状态，该状态不会释放锁资源。当 sleep()状态超时、join()等待线程终止或者超时、或者 I/O 处理完毕时，线程重新转入就绪状态。

5. 死亡(dead)：线程run()、main()方法执行结束，或者因异常退出了run()方法，则该线程结束生命周期。死亡的线程不可再次复生。

### 11、Java 中用到的线程调度算法是什么？

计算机通常只有一个 CPU，在任意时刻只能执行一条机器指令，每个线程只有获得CPU 的使用权才能执行指令。所谓多线程的并发运行，其实是指从宏观上看，各个线程轮流获得 CPU的使用权，分别执行各自的任务。在运行池中，会有多个处于就绪状态的线程在等待 CPU，JAVA 虚拟机的一项任务就是负责线程的调度，线程调度是指按照特定机制为多个线程分配CPU的使用权。  
有两种调度模型：分时调度模型和抢占式调度模型。  
分时调度模型是指让所有的线程轮流获得 cpu 的使用权，并且平均分配每个线程占用的 CPU的时间片这个也比较好理解。  
Java虚拟机采用抢占式调度模型，是指优先让可运行池中优先级高的线程占用

### 12、线程的调度策略

线程调度器选择优先级最高的线程运行，但是，如果发生以下情况，就会终止线程的运行：  
（1） 线程体中调用了yield方法让出了对cpu的占用权利  
（2） 线程体中调用了sleep方法使线程进入睡眠状态  
（3） 线程由于IO操作受到阻塞  
（4） 另外一个更高优先级线程出现  
（5） 在支持时间片的系统中，该线程的时间片用完

### 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）。

### 13、请说出与线程同步以及线程调度相关的方法。

（1）wait()：使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；  
（2）sleep()：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要处理 InterruptedException 异常；  
（3）notify()：唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM 确定唤醒哪个线程，而且与优先级无关；  
（4）notityAll()：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；

### 14、sleep() 和 wait() 有什么区别？

两者都可以暂停线程的执行类的不同：sleep() 是Thread线程类的静态方法，wait()是Object类的方法。

是否释放锁：sleep()不释放锁；wait()释放锁。

用途不同：Wait 通常被用于线程间交互/通信，sleep 通常被用于暂停执行。

用法不同：wait() 方法被调用后，线程不会自动苏醒，需要别的线程调用同一个对象上的 notify() 或者 notifyAll() 方法。 sleep() 方法执行完成后，线程会自动苏 醒。或者可以使用wait(long timeout)超时后线程会自动苏醒

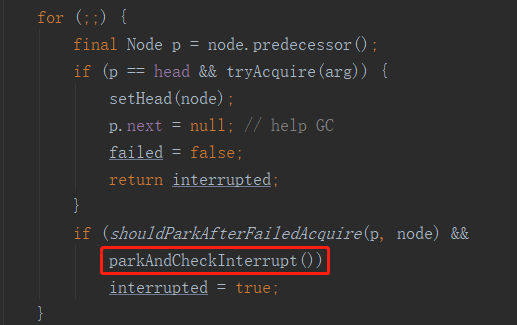
### 15、为什么 wait(), notify()和 notifyAll()必须在同步方法或者同步块中被调用？

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。

### 16、你是如何调用wait()方法的？使用 if 块还是循环？为什么？

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，程序就会在没有满足结束条件的情况下退出。wait() 方法应该在循环调用，因为当线程获取到 CPU 开始执行的时候，其他条件可能还没有满足，所以在处理前，循环检测条件是否满足会更好。

例如AQS中阻塞唤醒线程的操作



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

java中，任何对象都可以作为锁，并且 wait()，notify()等方法用于等待对象的锁或者唤醒线程，在Java的线程中并没有可供任何对象使用的锁，所以任意对象调用方法一定定义在Object类中。  
wait(),notify()和 notifyAll()这些方法在同步代码块中调用有的人会说，既然是线程放弃对象锁，那也可以把wait()定义在Thread类里面啊，新定义的线程继承于Thread类， 也不需要重新定义wait()方法的实现。然而，这样做有一个非常大的问题，一个线程完全可以持有很多锁，你一个线程放弃锁的时候，到底要放弃哪个锁？当然了，这种设计并不是不能实现，只是管理起来更加复杂。

### 18、Thread 类中的 yield 方法有什么作用？

使当前线程从执行状态（运行状态）变为可执行态（就绪状态） 。  
当前线程到了就绪状态，那么接下来哪个线程会从就绪状态变成执行状态呢？可能是当前线程，也可能是其他线程，看系统的分配了。

### 19、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。

### 20、线程的sleep()方法和yield()方法有什么区别？

（1）sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；  
（2）线程执行sleep()方法后转入阻塞（blocked）状态，而执行 yield()方法后转入就绪（ready）状态；  
（3）sleep()方法声明抛出 InterruptedException，而 yield()方法没有声明任何异常；  
（4）sleep()方法比 yield()方法（跟操作系统 CPU 调度相关）具有更好的可移植性，通常不建议使用yield()方法来控制并发线程的执行。

### 21、Java 中 interrupted 和 isInterrupted 方法的区别？

**interrupt**：用于中断线程。调用该方法的线程的状态为将被置为”中断”状态。  
注意：线程中断仅仅是置线程的中断状态位，不会停止线程。需要用户自己去监视线程的状态为并做处理。支持线程中断的方法（也就是线程中断后会抛出interruptedException 的方法）就是在监视线程的中断状态，一旦线程的中断状态被置为“中断状态”，就会抛出中断异常。

**interrupted**：是静态方法，查看当前中断信号是true还是false并且清除中断信号。如果一个线程被中断了，第一次调用interrupted则返回true，第二次和后面的就返回false 了。  
**isInterrupted**：查看当前中断信号是true还是false

### 22、notify()和notifyAll()有什么区别？

如果线程调用了对象的wait()方法，那么线程便会处于该对象的等待池中，等待池中的线程不会去竞争该对象的锁。  
notifyAll()会唤醒所有的线程，notify()只会唤醒一个线程。  
notifyAll()调用后，会将全部线程由等待池移到锁池，然后参与锁的竞争，竞争成功则继续执行，如果不成功则留在锁池等待锁被释放后再次参与竞争。而notify()只会唤醒一个线程，具体唤醒哪一个线程由虚拟机控制。

### 23、线程间的同步方法大体可分为两类：

用户模式和内核模式。顾名思义，内核模式就是指利用系统内核对象的单一性来进行同步， 使用时需要切换内核态与用户态，而用户模式就是不需要切换到内核态，只在用户态完成操作。用户模式下的方法有：原子操作（例如一个单一的全局变量），临界区（在同一时刻只能有一个任务访问的代码区通常使用锁实现）。内核模式下的方法有：事件，信号量，互斥量

### 24、在监视器(Monitor)内部， 是如何做线程同步的？

在 java 虚拟机中，每个对象( Object 和 class )通过某种逻辑关联监视器,每个监视器和一个对象引用相关联，为了实现监视器的互斥功能，每个对象都关联着一把锁。  
一旦方法或者代码块被 synchronized 修饰，那么这个部分就放入了监视器的监视区域，确保一次只能有一个线程执行该部分的代码，线程在获取锁之前不允许执行该部分的代码

另外 java 还提供了显式监视器( Lock )和隐式监视器( synchronized )两种锁方  
案

### 25、如果你提交任务时，线程池队列已满，这时会发生什么

（1）如果使用的是无界队列 LinkedBlockingQueue，也就是无界队列的话，没关系，继续添加任务到阻塞队列中等待执行，因为 LinkedBlockingQueue 可以近乎认为是一个无穷大的队列，可以无限存放任务  
（2）如果使用的是有界队列比如 ArrayBlockingQueue，会根据maximumPoolSize 的值增加线程数量，如果增加了线程数量还是处理不过来，ArrayBlockingQueue 也满了，那么则会使用拒绝策略RejectedExecutionHandler 处理满了的任务，默认是 AbortPolicy

### 26、什么叫线程安全？ servlet 是线程安全吗?

线程安全是编程中的术语，指某个方法在多线程环境中被调用时，能够正确地处理多个线程之间的共享变量，使程序功能正确完成。  
Servlet不是线程安全的，servlet 是单实例多线程的，当多个线程同时访问同一个方法，是不能保证共享变量的线程安全性的。  
Struts2 的action是多实例多线程的，是线程安全的，每个请求过来都会new一个新的 action 分配给这个请求，请求完成后销毁。  
SpringMVC的Controller是线程安全的吗？不是的，和 Servlet 类似的处理流程。  
Struts2 好处是不用考虑线程安全问题；Servlet 和 SpringMVC 需要考虑线程安  
全问题，但是性能可以提升不用处理太多的 gc，可以使用 ThreadLocal 来处理

### 27、线程类的构造方法、 静态块是被哪个线程调用的

这是一个非常刁钻和狡猾的问题。请记住：线程类的构造方法、静态块是被 new这个线程类所在的线程所调用的，而run方法里面的代码才是被线程自身所调用的。  
如果说上面的说法让你感到困惑，那么我举个例子，假设 Thread2中new 了Thread1 ， main函数中new了Thread2，那么：  
（1）Thread2的构造方法、静态块是 main 线程调用的，Thread2的run()方法是Thread2自己调用的  
（2）Thread1的构造方法、静态块是Thread2调用的，Thread1的run()方法是Thread1自己调用的