**什么是程序？**

是含有指令和数据的文件，被存储在磁盘或其他的数据存储设备中，也就是说程序是静态的代码。

**什么是进程？**  
可以简单的认为一个应用程序就是一个进程，不过有些应用程序会启动多个进程，即一个应用程序至少会启动一个进程

进程和进程之间的内存是独立的，这样可以提高应用程序的稳定性和安全性。

**什么是线程？**

线程是进程中的一个执行场景，一个进程可以启动多个线程  
使用多线程的可以提高CPU的使用率，不是提高执行速度。

**主线程的概念**

所有进程，至少会有一个线程即主线程，即main方法开始执行，就会有一个看不见的主线程存在。

### 并行和并发

并行就是两个或两个以上的任务同时运行，就是甲任务进行的同时，乙任务也在进行。(需要多核CPU)  
并发是指两个或两个以上的任务都请求运行，而CPU只能接受一个任务，就把这两个任务安排轮流进行，由于时间间隔较短，使人感觉两个任务都在运行。

### 多线程的优点

多线程就是为了充分利用cpu的资源，**可能会**提高程序执行效率。这里强调的是可能，使用多线程并不一定会提高程序的运行效率。

**多线程的缺点**

线程多的话，cpu会频繁的在线程之间切换，影响性能，可能也会出现一些bug。

**设计复杂**  
多线程中共享堆内存和方法区，因此里面的一些数据是可以共享的，在设计时要确保数据的准确性

**资源消耗增多**  
栈内存是不共享的，如果启用多个线程的话会占用更多的内存

**多线程创建的三种方式**

**继承Thread类创建线程类**

（1）定义Thread类的子类，并重写该类的run方法，该run方法的方法体就代表了线程要完成的任务。因此把run()方法称为执行体。

（2）创建Thread子类的实例，即创建了线程对象。

（3）调用线程对象的start()方法来启动该线程。

**通过Runnable接口创建线程类**

（1）定义runnable接口的实现类，并重写该接口的run()方法

（2）创建 Runnable实现类的实例，并依此实例作为Thread的target来创建Thread对象，该Thread对象才是真正的线程对象。

（3）调用线程对象的start()方法来启动该线程。

**通过Callable接口创建线程**

（1）创建Callable接口的实现类，并重写call()方法，该call()方法将作为线程执行体，并且有返回值。

（2）创建ExecutorService线程池

（3）将自定义类的对象放入线程池里面，获取线程的返回结果

（4）关闭线程池，不再接收新的线程，未执行完的线程不会被关闭

**匿名类**

使用匿名类，继承Thread,重写run方法，直接在run方法中写业务代码

匿名类的一个好处是可以很方便的访问外部的局部变量。

前提是外部的局部变量需要被声明为final。

### 线程池

线程池是初始化一个多线程应用程序过程中创建一个**线程集合**，然后在需要执行新的任务时直接去这个线程集合中获取，而不是创建一个线程。任务执行结束后，线程回到池子中等待下一次的分配。

**线程池的作用**  
解决创建单个线程耗费时间和资源的问题。

**创建线程池**

**Executors.newFixedThreadPool(int nThreads);**  
通过传入的int类型参数来指定创建线程池中的线程数，如果任务数量大于线程数量，则任务会进行等待。

**Executors.newCachedThreadPool();**  
会根据需要创建新线程的线程池，如果线程池中的线程数量小于任务数时，会创建新的线程，线程池中的线程最大数量是Integer.MAX\_VALUE，int类型的最大值。如果线程的处理速度小于任务的提交速度时，会不断创建新的线程来执行任务，这样有可能会因为创建过多线程而耗尽CPU 和内存资源。

**多线程创建的三种方式对比**

**继承Thread**

优点:可以直接使用Thread类中的方法,代码简单

缺点:继承Thread类之后就不能继承其他的类

**实现Runnable接口**

优点:即时自定义类已经有父类了也不受影响，因为可以实现多个接口

缺点: 在run方法内部需要获取到当前线程的Thread对象后才能使用Thread中的方法

**实现Callable接口**

优点：可以获取返回值，可以抛出异常

缺点：代码编写较为复杂

**Thread类方法**

**sleep()**

主要的作用是让当前线程停止执行，把cpu让给其他线程执行，但不会释放对象锁和监控的状态，到了指定时间后线程又会自动恢复运行状态

注意：线程睡眠到期自动苏醒，并返回到可运行状态，不是运行状态。sleep()中指定的时间是线程不会运行的最短时间。因此，sleep()方法不能保证该线程睡眠到期后就开始执行

**join()**

使调用该方法的线程在此之前执行完毕，也就是等待该方法的线程执行完毕后再往下继续执行。

**yield()**

该方法与sleep()类似，只是不能由用户指定暂停多长时间，并且yield（）方法只能让同优先级的线程有执行的机会。

**简述线程，程序、进程的基本概念。以及他们之间关系是什么**

与进程不同的是同类的多个线程共享同一块内存空间和一组系统资源，所以系统在产生一个线程，或是在各个线程之间作切换工作时，负担要比进程小得多，也正因为如此，线程也被称为轻量级进程。

系统运行一个程序即是一个进程从创建，运行到消亡的过程。简单来说，一个进程就是一个执行中的程序，它在计算机中一个指令接着一个指令地执行着，同时当程序在执行时，将会被操作系统载入内存中。 线程是进程划分成的更小的运行单位。线程和进程最大的不同在于基本上各进程是独立的，而各线程则不一定，因为同一进程中的线程极有可能会相互影响。从另一角度来说，进程属于操作系统的范畴，主要是同一段时间内，可以同时执行一个以上的程序，而线程则是在同一程序内几乎同时执行一个以上的程序段。

**设置线程优先级**

可以通过使用Thread类中的setPriority方法设置线程的优先级。

setPriority()方法接收一个int类型的参数，通过这个参数可以指定线程的优先级，取值范围是整数1~10，优先级随着数字的增大而增强。

在Thread类中封装了三个int类型的数字:

**优先级最低：public final static int MIN\_PRIORITY = 1;**

**优先级居中：public final static int NORM\_PRIORITY = 5;**

**优先级最高：public final static int MAX\_PRIORITY = 10;**

**可以使用Thread类中的interrupt方法唤醒正在睡眠的线程，调用interrupt方法会抛出一个InterruptedException的异常。**

**线程有哪些基本状态？这些状态是如何定义的?**

新建(new)：新创建了一个线程对象。

可运行(runnable)：线程对象创建后，其他线程(比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获 取cpu的使用权。

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。

阻塞(block)：阻塞状态是指线程因为某种原因放弃了cpu使用权，也即让出了cpu timeslice，暂时停止运行。直到线程进入可运行(runnable)状态，才有 机会再次获得cpu timeslice转到运行(running)状态。阻塞的情况分三种： (一). 等待阻塞：运行(running)的线程执行o.wait()方法，JVM会把该线程放 入等待队列(waitting queue)中。 (二). 同步阻塞：运行(running)的线程在获取对象的同步锁时，若该同步锁 被别的线程占用，则JVM会把该线程放入锁池(lock pool)中。 (三). 其他阻塞: 运行(running)的线程执行Thread.sleep(long ms)或t.join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。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，线程重新转入可运行(runnable)状态。

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。

**java里线程分2种，**

1、守护线程，比如垃圾回收线程，就是最典型的守护线程。

2、用户线程，就是应用程序里的自定义线程。

**什么是守护线程**

守护线程又被称为“服务进程”“精灵线程”“后台线程”，是指在程序运行是在后台提供一种通用的线程，这种线程并不属于程序不可或缺的部分。 通俗点讲，任何一个守护线程都是整个JVM中所有非守护线程的“保姆”。

用户线程和守护线程几乎一样，唯一的不同之处就在于如果用户线程已经全部退出运行，只剩下守护线程存在了，JVM也就退出了。 因为当所有非守护线程结束时，没有了被守护者，守护线程也就没有工作可做了，也就没有继续运行程序的必要了，程序也就终止了，同时会“杀死”所有守护线程。 也就是说，只要有任何非守护线程还在运行，程序就不会终止。

在Java语言中，守护线程一般具有较低的优先级，它并非只由JVM内部提供，用户在编写程序时也可以自己设置守护线程，例如将一个用户线程设置为守护线程的方法就是在调用start()方法启动线程之前调用对象的setDaemon(true)方法，若将以上括号里的参数设置为false，则表示的是用户进程模式。

需要注意的是，当在一个守护线程中产生了其它线程，那么这些新产生的线程默认还是守护线程，用户线程也是如此。

**为何要使用同步？**

java允许多线程并发控制，当多个线程同时操作一个可共享的资源变量时（如数据的增删改查），将会导致数据不准确，相互之间产生冲突，因此加入同步锁以避免在该线程没有完成操作之前，被其他线程的调用，从而保证了该变量的唯一性和准确性。

**同步方法**

即有synchronized关键字修饰的方法。由于java的每个对象都有一个内置锁，当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，需要获得内置锁，否则就处于阻塞状态。synchronized表示当前线程，当前线程独占了对象，如果有其他线程试图占有对象，就会等待，直到当前线程释放对对象的占用。该对象又叫同步对象，所有的对象，都可以作为同步对象，为了达到同步的效果，必须使用同一个同步对象，synchronized关键字也可以修饰静态方法，此时如果调用该静态方法，将会锁住整个类

**同步代码块**

即有synchronized关键字修饰的语句块。被该关键字修饰的语句块会自动被加上内置锁，从而实现同步。多个线程在执行synchronized同步代码块时，代码块括号里面可以传入任意对象，但一定要保证多个线程访问的是同一个对象。多用synchronized(this)｛｝

**同步是一种高开销的操作，因此应该尽量减少同步的内容。通常没有必要同步整个方法，使用synchronized代码块同步关键代码即可。**

**非线程安全的集合转换为线程安全**

Collections.synchronizedList，可以把ArrayList转换为线程安全的List。与此类似的，还有HashSet,LinkedList,HashMap等等非线程安全的类

**死锁**

死锁是指由于两个或者多个线程互相持有对方所需要的资源，导致这些线程处于等待状态，无法前往执行

**产生死锁的四个必要条件：**

互斥条件：一个资源每次只能被一个进程使用。

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。

不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。

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。

**避免死锁的方法**

固定加锁的顺序(针对锁顺序死锁)

开放调用(针对对象之间协作造成的死锁)

使用定时锁

**volatile的作用**

可以保持共享变量的可见性，即一个线程修改一个共享变量后，另一个线程能够读取到这个修改后的值。volatile只能修饰变量，不能修饰方法

**原子性和非原子性**

原子性：即一个操作或者多个操作 要么全部执行并且执行的过程不会被任何因素打断，要么就都不执行。

非原子性：不符合原子性的就是非原子性

**如果一段程序是具有原子性的，那么这段程序就不会出现线程安全问题。**

**volatile是非原子性的。synchronized是原子性的。**

**两个线程之间的通信**

多线程环境下CPU会随机的在线程之间进行切换，如果想让两个线程有规律的去执行，那就需要两个线程之间进行通信，在Object类中的两个方法wait和notify可以实现通信。

wait方法可以使当前线程进入到等待状态，在没有被唤醒的情况下，线程会一直保持等待状态。

notify方法可以随机唤醒单个在等待状态下的线程。

**notify()与notifyAll()的区别**

**notify():**

唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程。选择是任意性的，并在对实现做出决定时发生。线程通过调用其中一个 wait 方法，在对象的监视器上等待。

直到当前线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。

**notifyAll():**

唤醒在此对象监视器上等待的所有线程。线程通过调用其中一个 wait 方法，在对象的监视器上等待。

直到当前线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。

**锁池和等待池**

**锁池:**假设线程A已经拥有了某个对象(注意:不是类)的锁，而其它的线程想要调用这个对象的某个synchronized方法(或者synchronized块)，由于这些线程在进入对象的synchronized方法之前必须先获得该对象的锁的拥有权，但是该对象的锁目前正被线程A拥有，所以这些线程就进入了该对象的锁池中。

**等待池:**假设一个线程A调用了某个对象的wait()方法，线程A就会释放该对象的锁后，进入到了该对象的等待池中

**wait方法与notify方法必须在同步块内执行,即synchronized(obj之内).**

**只有在调用线程拥有某个对象的独占锁时，才能够调用该对象的wait(),notify()和notifyAll()方法**

**在 Java中，所有对象都能够被作为"监视器monitor"——指一个拥有一个独占锁，一个入口队列和一个等待队列的实体**

**sleep和notify方法不释放对象锁,wait方法释放锁**