1.线程的几种状态：

1. NEW：尚未启动的线程处于此状态
2. RUNNABLE：在Java虚拟机中执行的线程处于此状态
3. BLOCKED：被阻塞等待监视器锁定的线程处于此状态
4. WAITING：正在等待另一个线程执行特定动作的线程处于此状态
5. TIMED\_WAITING：正在等待另一个线程执行动作达到指定等待时间的线程处于此状态
6. TERMINATED：已经退出的线程处于此状态

线程的同步机制：

1. 在多线程编程中，一些敏感数据不允许被多个线程同时访问，此时就使用同步访问技术，保证数据在任何同一时刻，最多有一个线程访问，以保证数据的完整性
2. 也可以理解为：线程同步，即当有一个线程在对内存进行操作时，其他线程都不可以对这个内存地址进行操作，知道该线程完成操作，其他线程才能对该内存地址进行操作
3. 同步代码块：synchronized（对象）{//得到对象的锁，才能操作同步代码

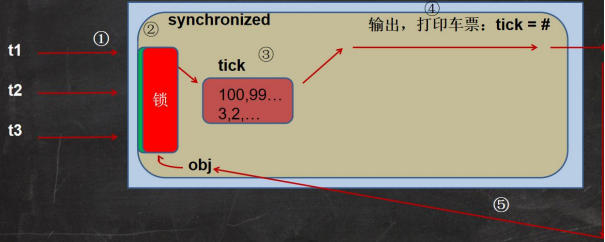
//需要被同步的代码}

1. synchronized还可以放在方法中声明，表示整个方法为同步方法：

Public synchronized void m (String name) {

//需要被同步的代码

}

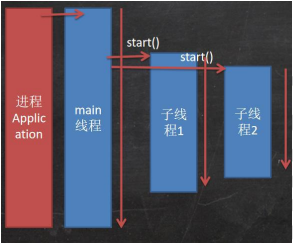


线程相关概念：

1. 单线程：同一时刻只允许执行一个线程
2. 多线程：同一时刻可以执行多个线程
3. 并发：同一时刻多个任务交替执行，形成一种“貌似同时”的错觉，简单来说就是单核CPU实现的多任务就是并发
4. 并行：同一时刻多个任务同时执行，多核CPU可以实现并行

2.创建线程的两种方式：

1. 继承Thread类，重写run方法
2. 实现Runnable接口，重写run方法



3.volatile关键字：

volatile是一个变量修饰符，只能用来修饰变量，无法修饰方法及代码块等。对于volatile变量，当对volatile变量进行写操作的时候，JVM会向处理器发送一条lock前缀的指令，将这个缓存中的变量回写到系统主存中。

但是就算写回到内存，如果其他处理器缓存的值还是旧的，再执行计算操作就会有问题，所以在多处理器下，为了保证各个处理器的缓存是一致的，就会实现缓存一致性协议。

缓存一致性协议：每个处理器通过嗅探在总线上传播的数据来检查自己缓存的值是不是过期了，当处理器发现自己缓存行对应的内存地址被修改，就会将当前处理器的缓存行设置成无效状态，当处理器要对这个数据进行修改操作的时候，会强制重新从系统内存里把数据读到处理器缓存里。

所以，如果一个变量被volatile所修饰的话，在每次数据变化之后，其值都会被强制刷入主存。而其他处理器的缓存由于遵守了缓存一致性协议，也会把这个变量的值从主存加载到自己的缓存中。这就保证了一个volatile在并发编程中，其值在多个缓存中是可见的。

4.线程终止：

1. 当线程完成任务后会自动退出
2. 可以通过使用变量来控制run方法的方式退出停止线程，即通知方式

用户线程和守护线程  
（1）用户线程：也叫做工作线程，当线程的任务执行完成或以通知的方式结束

（2）守护线程：一般是为工作线程服务的，当所有的用户线程结束，守护线程自动结束（常见的守护线程：垃圾回收机制）

5.使用线程池的好处：

1. 减少在创建和销毁线程上所花的时间以及系统资源的开销
2. 如不使用线程池，有可能造成系统创建大量线程而导致消耗完系统内存

线程池的执行过程：

1. 首先检测线程池运行状态，如果不是RUNNING，则直接拒绝，线程池要保证在RUNNING的状态下执行任务
2. 如果workerCount < corePoolSize，则创建并启动一个线程来执行新提交的任务
3. 如果workerCount >= corePoolSize，且线程池内的阻塞队列未满，则将任务添加到该阻塞队列中
4. 如果workerCount >= corePoolSize && workerCount < maximumPoolSize，且线程池内的阻塞队列已满，则创建并启动一个线程来执行新提交的任务
5. 如果workerCount >= maximumPoolSize，并且线程池内的阻塞队列已满, 则根据拒绝策略来处理该任务, 默认的处理方式是直接抛异常

线程池的几大参数：

1. corePoolSize：核心池的大小。在创建了线程池后，默认情况下，线程池中并没有任何线程，而是等待有任务到来才创建线程去执行任务，除非调用了prestartAllCoreThreads()或者prestartCoreThread()方法，从这2个方法的名字就可以看出，是预创建线程的意思，即在没有任务到来之前就创建corePoolSize个线程或者一个线程
2. maximumPoolSize：线程池最大线程数。这个参数也是一个非常重要的参数，它表示在线程池中最多能创建多少个线程
3. keepAliveTime：表示线程没有任务执行时最多保持多久时间会终止。默认情况下，只有当线程池中的线程数大于corePoolSize时，keepAliveTime才会起作用，直到线程池中的线程数不大于corePoolSize，即当线程池中的线程数大于corePoolSize时，如果一个线程空闲的时间达到keepAliveTime，则会终止，直到线程池中的线程数不超过corePoolSize
4. unit：参数keepAliveTime的时间单位，有7种取值，在TimeUnit类中有7种静态属性
5. workQueue：一个阻塞队列，用来存储等待执行的任务，这个参数的选择也很重要，会对线程池的运行过程产生重大影响

6.利用Thread.sleep()休眠来实现线程池的定时任务

7.synchronized 和 ReentrantLock 的异同比较：

把代码块声明为 synchronized，有两个重要后果，通常是指该代码具有 原子性（atomicity）和 可见性（visibility）。原子性意味着一个线程一次只能执行由一个指定监控对象（lock）保护的代码，从而防止多个线程在更新共享状态时相互冲突。可见性则更为微妙；它要对付内存缓存和编译器优化的各种反常行为。

ReentrantLock 类实现了 Lock ，它拥有与 synchronized 相同的并发性和内存语义，但是添加了类似锁投票、定时锁等候和可中断锁等候的一些特性。此外，它还提供了在激烈争用情况下更佳的性能。（换句话说，当许多线程都想访问共享资源时，JVM 可以花更少的时候来调度线程，把更多时间用在执行线程上）

  Lock 和 synchronized 有一点明显的区别 —— lock 必须在 finally 块中释放。否则，如果受保护的代码将抛出异常，锁就有可能永远得不到释放。

8.Synchronized 修饰静态方法时获取的锁和修饰非静态方法时获取的锁有什么不同？

（1）Synchronized修饰非静态方法的锁，其实是属于当前对象的锁，属于对象锁

（2）Synchronized修饰静态方法的锁，其实是属于当前类，属于类锁，类锁是一个抽象概念，为了区别对象锁

9.并发安全的集合类有：

1. Hashtable：通过synchronized关键字保证线程安全
2. Vector 矢量队列，实现了List，通过synchronized关键字保证线程安全，跟Hashtable类似
3. StringBuffer和StringBuilder中，Stringbuffer是线程安全的，通过synchronized关键字保证线程安全