**1、 认识Thread和Runnable**   
  
Java中实现多线程有两种途径：继承Thread类或者实现Runnable接口。Runnable是接口，建议用接口的方式生成线程，因为接口可以实现多继承，况且Runnable只有一个run方法，很适合继承。在使用Thread的时候只需继承Thread，并且new一个实例出来，调用 start()方法即可以启动一个线程。   
  
Thread Test = new Thread();   
  
Test.start();   
  
在使用Runnable的时候需要先new一个实现Runnable的实例，之后启动Thread即可。   
  
Test impelements Runnable;   
  
Test t = new Test();   
  
Thread test = new Thread(t);   
  
test.start();   
  
总结：Thread和Runnable是实现java多线程的2种方式，runable是接口，thread是类，建议使用runable实现 java多线程，不管如何，最终都需要通过thread.start()来使线程处于可运行状态。   
  
**2、 认识Thread的start和run**   
  
1） start：   
  
用start方法来启动线程，真正实现了多线程运行，这时无需等待run方法体代码执行完毕而直接继续执行下面的代码。通过调用Thread类的 start()方法来启动一个线程，这时此线程处于就绪（可运行）状态，并没有运行，一旦得到cpu时间片，就开始执行run()方法，这里方法 run()称为线程体，它包含了要执行的这个线程的内容，Run方法运行结束，此线程随即终止。   
  
2） run：   
  
run()方法只是类的一个普通方法而已，如果直接调用Run方法，程序中依然只有主线程这一个线程，其程序执行路径还是只有一条，还是要顺序执行，还是要等待run方法体执行完毕后才可继续执行下面的代码，这样就没有达到写线程的目的。   
  
总结：调用start方法方可启动线程，而run方法只是thread的一个普通方法调用，还是在主线程里执行。   
  
**3、 线程状态说明**   
  
线程状态从大的方面来说，可归结为：初始状态、可运行状态、不可运行状态和消亡状态，具体可细分为上图所示7个状态，说明如下：   
  
1） 线程的实现有两种方式，一是继承Thread类，二是实现Runnable接口，但不管怎样，当我们new了thread实例后，线程就进入了初始状态；   
  
2） 当该对象调用了start()方法，就进入可运行状态；   
  
3） 进入可运行状态后，当该对象被操作系统选中，获得CPU时间片就会进入运行状态；   
  
4） 进入运行状态后case就比较多，大致有如下情形：   
  
﹒run()方法或main()方法结束后，线程就进入终止状态；   
  
﹒当线程调用了自身的sleep()方法或其他线程的join()方法，就会进入阻塞状态(该状态既停止当前线程，但并不释放所占有的资源)。当 sleep()结束或join()结束后，该线程进入可运行状态，继续等待OS分配时间片；   
  
﹒当线程刚进入可运行状态(注意，还没运行)，发现将要调用的资源被锁牢(synchroniza,lock)，将会立即进入锁池状态，等待获取锁标记(这时的锁池里也许已经有了其他线程在等待获取锁标记，这时它们处于队列状态，既先到先得)，一旦线程获得锁标记后，就转入可运行状态，等待OS分配 CPU时间片；   
  
﹒当线程调用wait()方法后会进入等待队列(进入这个状态会释放所占有的所有资源，与阻塞状态不同)，进入这个状态后，是不能自动唤醒的，必须依靠其他线程调用notify()或notifyAll()方法才能被唤醒(由于notify()只是唤醒一个线程，但我们由不能确定具体唤醒的是哪一个线程，也许我们需要唤醒的线程不能够被唤醒，因此在实际使用时，一般都用notifyAll()方法，唤醒有所线程)，线程被唤醒后会进入锁池，等待获取锁标记。   
  
﹒当线程调用stop方法，即可使线程进入消亡状态，但是由于stop方法是不安全的，不鼓励使用，大家可以通过run方法里的条件变通实现线程的 stop。   
 **4、 Timer 和 Timer Task 的使用**   
  
Timer 是一种定时器工具，用来在一个后台线程计划执行指定任务，这些任务可以被执行一次，也可以被定期执行。每个 Timer 对象对应一个后台线程，顺序地执行所有计时器任务。如果完成某个计时器任务的时间太长，那么它会“独占”计时器的任务执行线程，从而可能延迟后续任务的执行。对 Timer 对象最后的引用完成并且所有未处理的任务都已执行完成后，计时器的任务执行线程会正常终止（并且成为垃圾回收的对象）。TimerTask是一个抽象类，实现了Runable接口，它的子类代表一个可以被Timer计划的任务。   
  
1） 一个简单的Demo，让大家对Timer、TimerTask的使用有感性的认识。   
  
  
  
2） Timer和TimerTask的常用api函数说明   
  
  
  
这里强调Timer类的schedule和scheduleAtFixedRate的区别。schedule和 scheduleAtFixedRate的区别在于，schedule以固定的相对时间间隔执行，如果某一次执行被延时了，往后的执行的执行时间也会相对延时；而scheduleAtFixedRate是以绝对的时间间隔执行，如果某一次执行被延时，它的后一次执行的延时将会缩短（scheduleAtFixedRate会把已经过去的时间也作为周期执行）。schedule注重的是时间间隔的稳定，而 scheduleAtFixedRate注重的是执行频率的稳定。   
  
3） Timer的终止   
  
默认情况下，只要一个程序的timer线程在运行，那么这个程序就会保持运行。当然，你可以通过以下四种方法终止一个timer线程：   
  
a）调用timer的cancle方法。你可以从程序的任何地方调用此方法，甚至在一个timer task的run方法里；   
  
b）让timer线程成为一个daemon线程（可以在创建timer时使用new Timer(true)达到这个目地），这样当程序只有daemon线程的时候，它就会自动终止运行；   
  
c）当timer相关的所有task执行完毕以后，删除所有此timer对象的引用（置成null），这样timer线程也会终止；   
  
d）调用System.exit方法，使整个程序（所有线程）终止。   
  
总结：Timer和TimerTask可以简单理解为Timer定时器在触发TimerTask任务调用，通常用schedule和 scheduleAtFixedRate方法来调用timertask任务，cancle来终止任务调用。Timer简单易用，比较适合提供轻量级的计时器功能，但是对时效性很强的任务调度请用其它方法来实现（正如javadoc所述”Timer does not offer real-time guarantees: it schedules tasks using the Object.wait(long) method”）。