1. Java多线程高级知识:
   1. 多线程有几种状态
      1. NEW ：初始状态，线程被创建，没有调用start（）;
      2. RUNNABLE：运行状态，Java线程把操作系统中的就绪和运行两种状态统一称为“运行中”
      3. BLOCKED：线程进入等待状态，线程因为某种原因，放弃了CPU的使用权

阻塞的几种情况：

A. 等待阻塞：运行的线程执行了wait()，JVM会把当前线程放入等待队列

B. 同步阻塞：运行的线程在获取对象的同步锁时，如果该同步锁被其他线程占用了，JVM会把当前线程放入锁池中

C. 其他阻塞：运行的线程执行sleep(),join()或者发出IO请求时，JVM会把当前线程设置为阻塞状态，当sleep()执行完，join()线程终止，IO处理完毕线程再次恢复

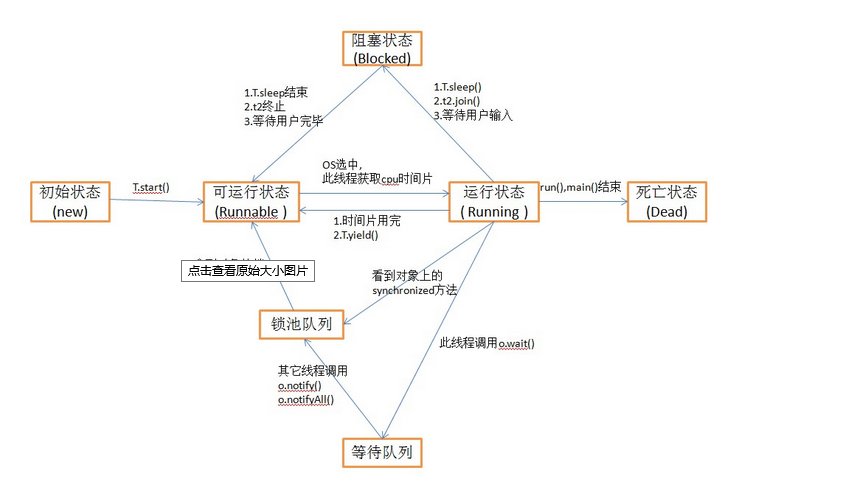
* + 1. WAITING：死等。
    2. TIME\_WAITING：超时等待状态，超时以后自动返回；
    3. TERMINATED：终止状态，当前线程执行完毕
  1. Wait与sleep的区别。

·区别:

* + 1. wait让当前线程等待，并释放所占用的锁；
    2. Sleep只是让当前线程进入阻塞状态，并不释放占用的锁

·相同点就是都会使当前功能停止；

1. 线程状态之间的相互转换
   1. 当实例化一个线程之后，首先进入初始状态，即New状态，此时线程在启动的时候并不是立刻就运行，而是要等到操作系统调度之后才运行，然后调用系统底层start()进入运行状态,即runnable,其中运行状态中包括运行（running）和就绪（ready）两种状态，这两种状态在操作系统的调度下可以互相转换，如果运行中的线程时间片被CPU抢占的话就会变成就绪状态；运行中的线程通过调用synchronized方法或synchronized块进入阻塞状态，即blocked，当线程获取到锁之后进入运行状态；如果线程在执行过程中调用了sleep(),wait(),join(),Locksupported.parkUtil()等方法时，会进入等待状态（waiting）或超时等待状态，即timed\_waiting，再次调用notify(),notifyAll(),Locksupported.unpark()等方法时，又会重新进入运行时状态，当线程执行完成时，就进入了终止状态，即terminated状态。



1. 并发——CAS机制
   1. 在了解CAS机制之前我们首先需要对乐观锁，悲观锁有一个基本的概念。

* 乐观锁：在并发下对数据进行修改时保持乐观的态度，认为在自己修改数据的过程中，其他线程不会对同一个数据进行修改，所以不对数据加锁，但是会在最终更新数据前，判断一下这个数据有没有被修改，若没有被修改，才将它更新为自己修改的值；
* 悲观锁：在并发下对数据进行修改时保持悲观的态度，认为在自己修改数据的过程中，其他线程也会对数据进行修改，所以在操作前会对数据加锁，在操作完成后才将锁释放，而在释放锁之前，其他线程无法操作数据；

CAS其实就是乐观锁的一种实现方式，而悲观锁比较典型的就是Java中的synchronized。下面我就来详细介绍一下CAS的相关概念。

CAS全称compare and swap——比较并替换，它是并发条件下修改数据的一种机制，包含三个操作数：

* 需要修改的数据的内存地址（V）；
* 对这个数据的旧预期值（A）；
* 需要将它修改为新的值（B）；

  CAS的操作步骤如下：

1. 修改前记录数据的内存地址V；
2. 读取数据的当前的值，记录为A；
3. 修改数据的值变为B；
4. 查看地址V下的值是否仍然为A，若为A，则用B替换它；若地址V下的值不为A，表示在自己修改的过程中，其他的线程对数据进行了修改，则不更新变量的值，而是重新从步骤2开始执行，这被称为**自旋**；

  通过以上四个步骤对内存中的数据进行修改，就可以保证数据修改的原子性。CAS是乐观锁的一种实现，所以这里介绍的步骤和乐观锁的定义差不多，还是很好理解的。