java线程在一定条件下，状态会发生变化。线程一共有以下几个状态：开始状态、就绪状态、运行状态、阻塞状态、终止状态。

开始状态：新创建了一个对象。

就绪状态：线程创建对象后，其他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，变得可运行，只等待获取CPU的使用权，即在就绪状态的线程除CPU以外，其他运行的资源都已经获得了。

运行状态:就绪状态的线程获取了CPU，执行程序代码。

阻塞状态:阻塞状态是线程因为某种原因放弃了CPU的使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。

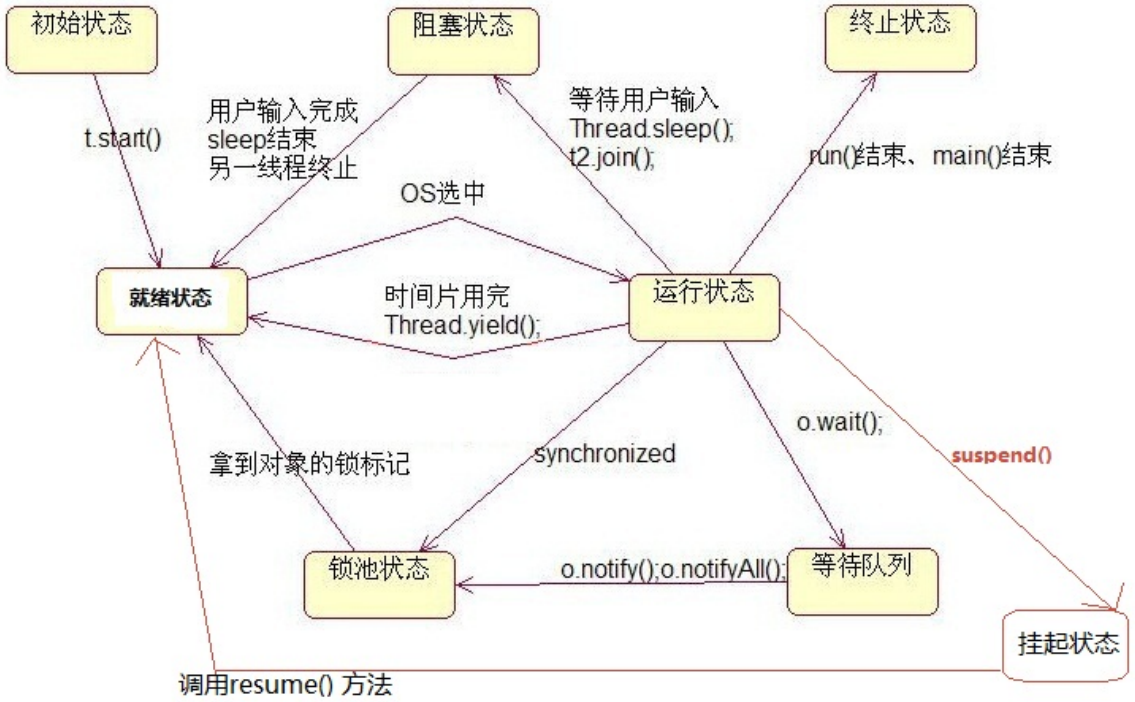
阻塞的情况分三种：

1.等待阻塞:运行的线程执行wait方法，该线程会释放占用的所有资源，JVM会把该线程放入“等待池”中。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或者notifyALL方法才能唤醒。

同步阻塞:运行的线程获取对象的同步锁的时候，若该同步锁被其他线程占用，则JVM会把该线程放入“锁池”中。

其他阻塞：运行的线程执行sleep和join方法，或者发出了IO请求时，JVM会把该线程置为阻塞线程。

终止状态：线程执行完了或者因异常退出run方法时候，该线程结束生命。



其余状态都不是很复杂，当线程进入到运行状态后就比较复杂了。

1.run方法或者main方法结束后，线程就进入了终止状态。

2.当线程调用sleep方法或者其他线程的join方法，线程让出CPU，然后就会进入阻塞状态，这个状态不会释放任何锁，当sleep结束或者join结束后，该线程进入可运行状态，继续等待os分配cpu时间片。典型的：sleep被用在等待某个资源就绪的情形，测试发现条件不满足后，让线程阻塞一段时间，直到条件满足为止。

3.线程调用了yield方法，意思是放弃当前获得的cpu时间片，回到就绪状态，这时与其他线程同处于竞争状态，os有可能会接着又让这个线程进入运行状态；调用yield的效果等价于调度程序认为该线程已经执行了足够的时间片从而需要转到另一个线程。yield只是使当前线程重新回到可执行状态，所以执行yield的线程有可能在进入到可执行状态后马上又被执行。

4.当线程进入可运行状态，发现将要调用的资源synchroniza，获得不到锁标记，将会进入锁池标记，等待获取锁标记，一旦线程获得锁标记后，就转入就绪状态，等待os分配cpu时间片。

5.suspend方法和resume方法：两个方法配套使用，suspend使得线程进入阻塞状态，并且不会自动回复，必须其对应的resume方法被调用，才能使得线程重新进入可执行状态，例如：suspend和resume被用在等待另一个线程产生的结果的情形，测试发现结果还没有产生后，让线程阻塞，另一个线程产生了结果后，调用resume方法使其恢复。

6.wait和notify方法，当线程调用wait方法后会进入等待队列中，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和notifyall方法才能唤醒。线程唤醒后会进入锁池，等待获取锁标记。

注意点：

1.wait和notify方法　与 suspend和resume方法的区别在于：suspend及其所有方法在线程阻塞时都不会释放占用的锁，而wait和notify会释放所有的资源。

2.notify方法导致解除阻塞的线程是从因调用该对象的wait方法而阻塞的线程是随机选择的，我们无法预料哪一个线程将会被选择，而notifyall方法将把因调用该对象的wait方法而阻塞的所有线程一次性全部解除。

3.wait和notify方法这一对却直接隶属于object类，也就是说，所有对象都拥有这一方法，因为这一对方法阻塞时要释放占用的锁，而锁是任何对象都具有的，调用任意对象的　wait()　方法导致线程阻塞，并且该对象上的锁被释放。wait和notify方法必须在synchronized方法中调用，理由是只有在synchronized方法或块中当前线程才占有锁，才有锁释放。