JAVA 线程相关的概念

作者：yhm2
原文链接：https://ld246.com/article/1578473237647
来源网站：链滴

JAVA线程基础

本文仅是本人理解，有不对的地方，欢迎留言批评指正

线程状态

初始状态

```
public enum State {
    /**
     * Thread state for a thread which has not yet started.
     */
    NEW,

    /**
     * Thread state for a runnable thread.  A thread in the runnable
     * state is executing in the Java virtual machine but it may
     * be waiting for other resources from the operating system
     * such as processor.
     */
    RUNNABLE,

    /**
     * Thread state for a thread blocked waiting for a monitor lock.
     * A thread in the blocked state is waiting for a monitor lock
     * to enter a synchronized block/method or
     * reenter a synchronized block/method after calling
     * {@link Object#wait() Object.wait}.
     */
    BLOCKED,

    /**
     * Thread state for a waiting thread.
     * A thread is in the waiting state due to calling one of the
     * following methods:
     * <ul>
     *   <li>{@link Object#wait() Object.wait} with no timeout</li>
     *   <li>{@link #join() Thread.join} with no timeout</li>
     *   <li>{@link LockSupport#park() LockSupport.park}</li>
     * </ul>
     *
     * <p>A thread in the waiting state is waiting for another thread to
     * perform a particular action.
     *
     * For example, a thread that has called <tt>Object.wait()</tt>
     * on an object is waiting for another thread to call
     * <tt>Object.notify()</tt> or <tt>Object.notifyAll()</tt> on
     * that object. A thread that has called <tt>Thread.join()</tt>
     * is waiting for a specified thread to terminate.
     */
    WAITING,
```

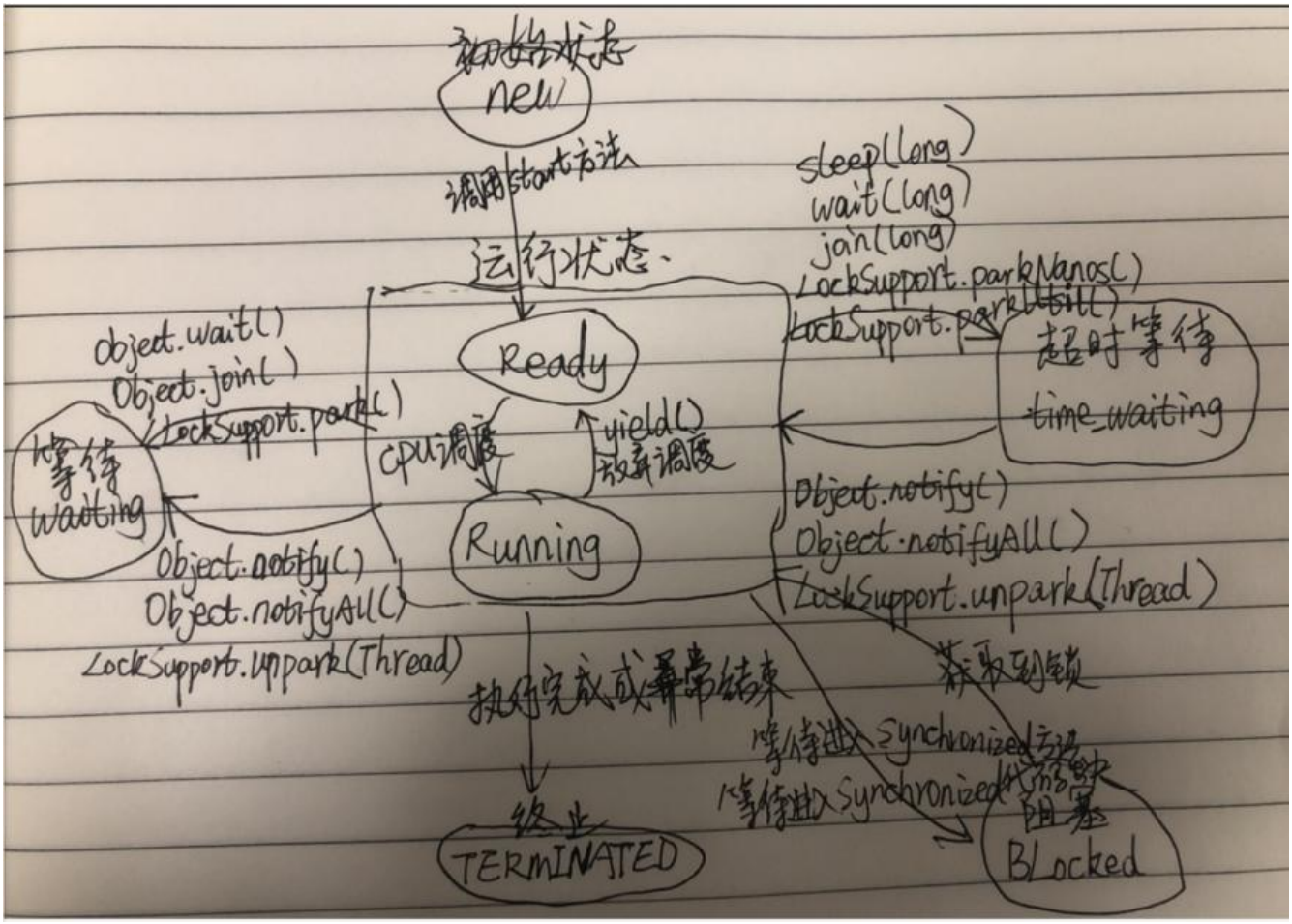
```
    /**
     * Thread state for a waiting thread with a specified waiting time.
     * A thread is in the timed waiting state due to calling one of
     * the following methods with a specified positive waiting time:
     * <ul>
     *   <li>{@link #sleep Thread.sleep}</li>
     *   <li>{@link Object#wait(long) Object.wait} with timeout</li>
     *   <li>{@link #join(long) Thread.join} with timeout</li>
     *   <li>{@link LockSupport#parkNanos LockSupport.parkNanos}</li>
     *   <li>{@link LockSupport#parkUntil LockSupport.parkUntil}</li>
     * </ul>
     */
    TIMED_WAITING,

    /**
     * Thread state for a terminated thread.
     * The thread has completed execution.
     */
    TERMINATED;
}
```

上面是JDK源码定义的线程状态

- NEW：新建状态，初始状态
- RUNNABLE：可执行状态，得到CPU调度后就可以执行
- BLOCKED：阻塞状态，比如未获取到锁、调用了wait方法进入等待阻塞状态、调用了sleep方法进了休眠等待状态
- WAITING：调用了wait方法进入的状态
- TIMED_WAITING：调用了sleep方法，还未到达时间前，线程的状态
- TERMINATED：终止状态

线程状态变化

变化图仅是个人理解：



变化图解析：

● 初始状态new: 线程创建完毕，还未调用start方法前的状态。

● ready和running状态：这2个状态都是运行状态，在ready状态的线程，在获取到CPU执行之间之就进入running状态；cpu时间片到了或者被其他线程争夺了cpu时间片，则该线程进入到ready状态也可以是当前线程主动调用yield( )方法，让给其他线程时间片去执行。

● 在运行状态中的线程，当调用Object.wait()方法、Object.join()方法、LockSupport.park()方法，该线程进入阻塞waiting状态；当其他线程调用了Object.notify()、Object.notifyAll()方法、LockSupport.unpark(Thread t)方法，则此线程进入到了ready状态，也就是进入到了等待队列。

● 在运行状态的线程，当执行完线程所有工作、或者发生异常，但是没有捕获到异常的情况下，线程入到TERMINATED(终止)状态。

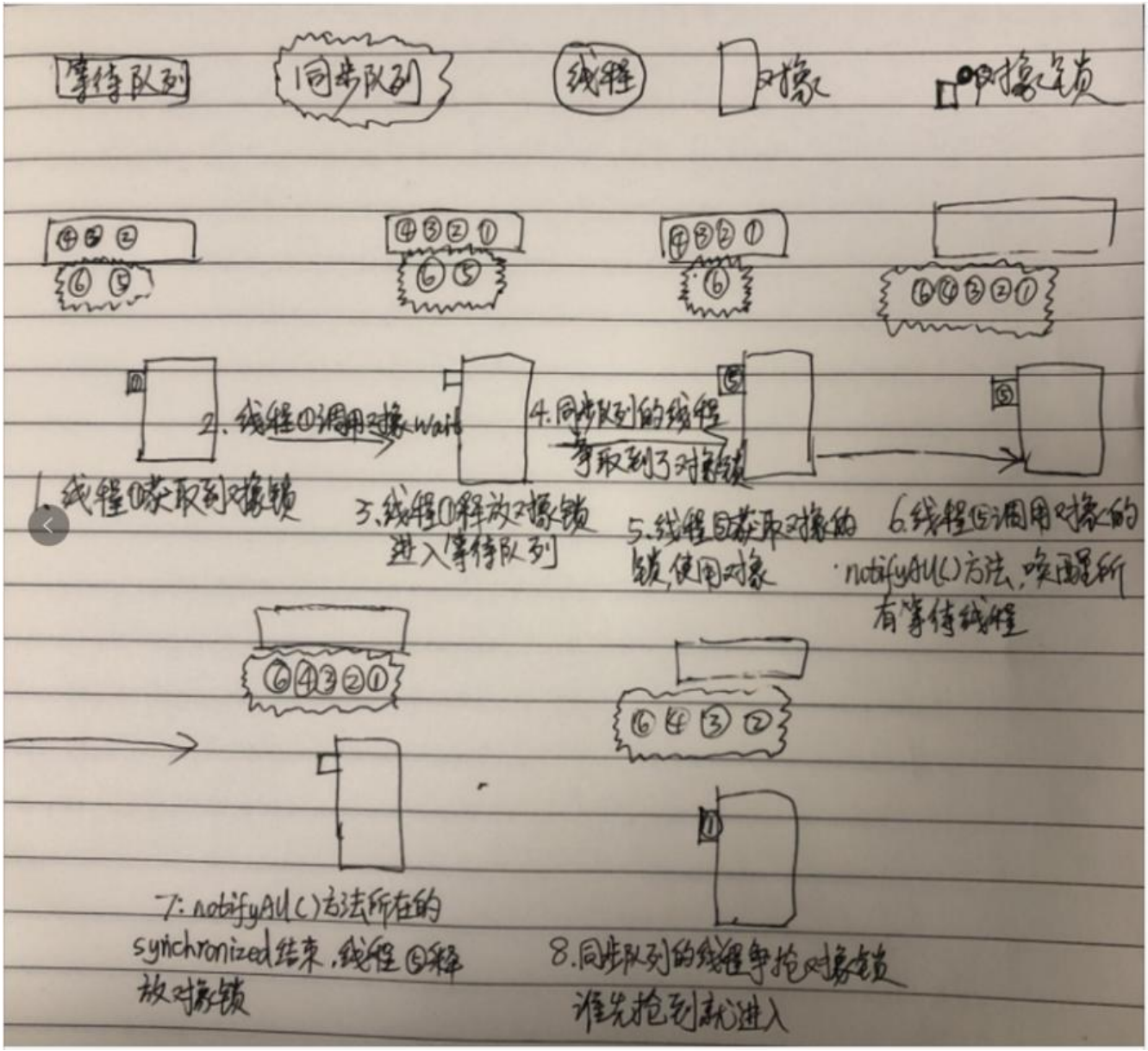
●

● 在运行状态的线程，当需要和其他线程竞争锁资源时，则此线程未获得锁就进入阻塞状态（bloced状态），比如竞争Synchronized关键字修饰的方法或者Synchronized修饰的代码块，此线程进入了同步队列，所有竞争该锁的线程都会先进入同步队列，当某一个线程获取到锁，则该线程进入运行态（在ready状态和running状态来回切换，直到该线程释放锁，走出Synchronized代码块或方法）如果获取到了竞争锁，则该线程继续运行。

● 在运行状态的线程，当调用Thread.sleep(long)、Object.wait(long）、Object.join(long)、LockSpport.parkNanos(long) 、LockSupport.parkUtil(long 从1970.1.1到将来某个时间的毫秒数)，此线程进入超时等待状态（time_waiting）；当此线程被其他线程调用Object.notify()、Object.notifAll()、LockSupport.unpark(Thread t) 时，则此线程进入运行状态。

线程竞争锁资源图解

(方框为等待队列，带波浪线的框为同步队列，圆圈为线程，大长方形为对象，小正方形为对象锁)



起初等待队列中有4、3、2线程，同步队列中有6、5线程，这些线程都需要“最终获取到对象锁才能行完毕”，首先1线程获得了对象的锁， 它正在执行步骤：

1、线程执行到某一个时刻

2、它调用了wait()方法，自己进入了超时等待状态。

3、1线程而且释放了自己占用的锁资源对象

4、同步队列中的线程开始竞争锁资源

5、5线程争抢到了对象锁的资源

6、5线程在执行的过程中，主动调用了notifyAll()方法，唤醒了所有线程，使所有线程都进入到了同队列

7、5线程执行完毕Synchronized包含的逻辑，释放了锁资源

原文链接：JAVA 线程相关的概念

8、同步队列中的6、4、3、2、1线程都可以竞争锁资源

线程中几个方法的对比

- Thread.sleep(long time)方法，睡眠指定时间（毫秒），不会释放对象锁。
- Thread.yield()方法，当前线程让步给其他线程执行，但是可能又会被重新选中到CPU执行。
- Thread.wait(long time) ,释放锁资源，到了time的毫秒数，进入同步队列重新竞争锁资源。
- Thread.join(long time) ,当前线程调用其他线程的join方法，自己进入waiting/time_waiting状态其他线程执行完毕，或者到了time的时间，当前线程进入就绪（ready）状态。当前线程不会释放锁源。
- obj.notify()和obj.notifyAll()方法，唤醒锁对象等待队列中的线程，notify()是唤醒任意一个，notifiAll()是唤醒所有。