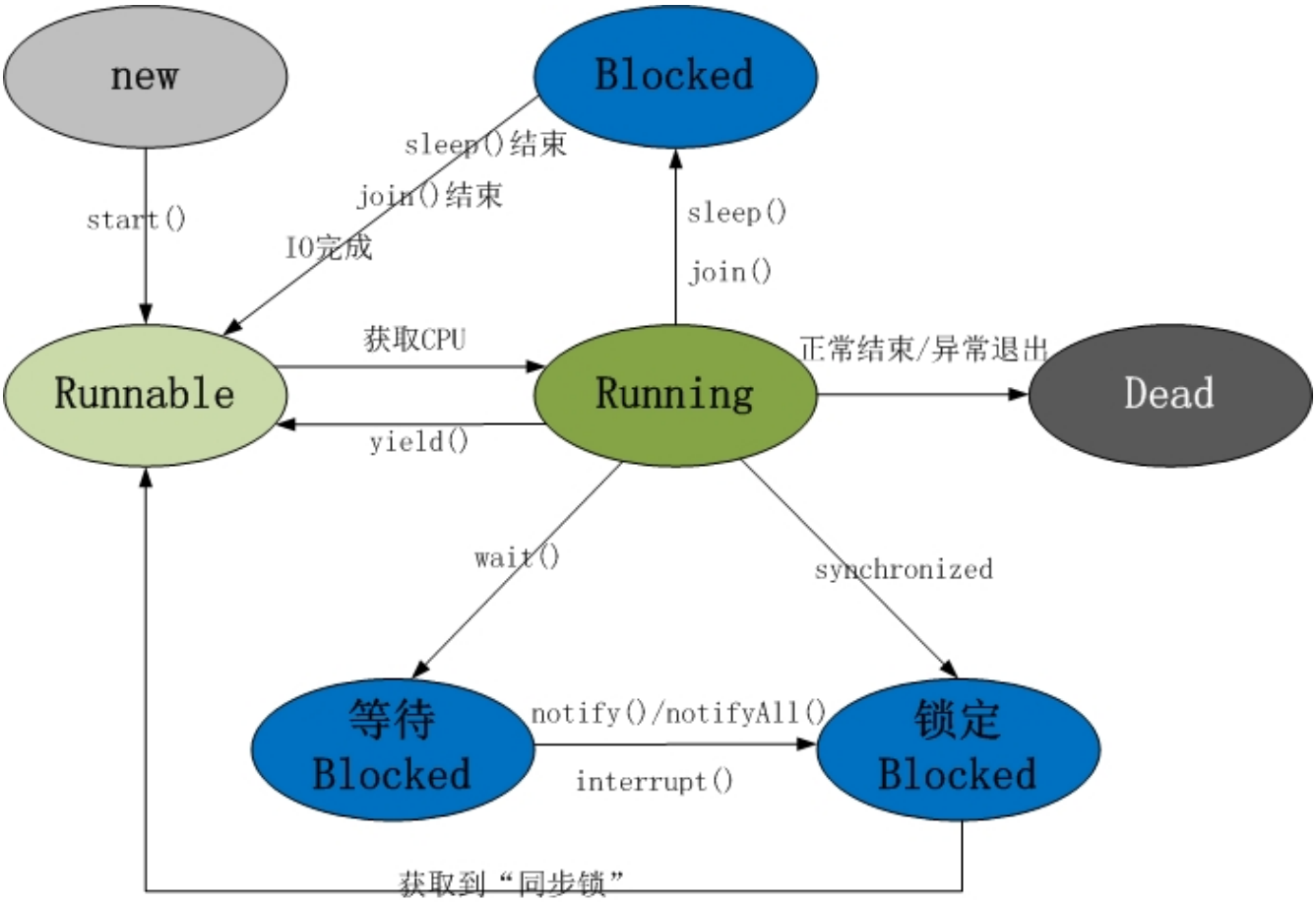
# 线程状态及转化

线程状态图




说明： 线程共包括以下5种状态。

1. 新建状态(New)　　: 线程对象被创建后，就进入了新建状态。例如，Thread thread = new Thread()。
2. 就绪状态(Runnable): 也被称为“可执行状态”。线程对象被创建后，其它线程调用了该对象的start()方法，从而来启动该线程。例如，thread.start()。处于就绪状态的线程，随时可能被CPU调度执行。
3. 运行状态(Running) : 线程获取CPU权限进行执行。需要注意的是，线程只能从就绪状态进入到运行状态。
4. 阻塞状态(Blocked) : 阻塞状态是线程因为某种原因放弃CPU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：　(01) 等待阻塞 -- 通过调用线程的wait()方法，让线程等待某工作的完成。　(02) 同步阻塞 -- 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态。　(03) 其他阻塞 -- 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。
5. 死亡状态(Dead)　: 线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

## 1. wait(), notify(), notifyAll()等方法介绍

在Object.java中，定义了wait(), notify()和notifyAll()等方法。wait()的作用是让当前线程进入等待状态，同时，wait()也会让当前线程释放它所持有的锁。而notify()和notifyAll()的作用，则是唤醒当前对象上的等待线程；notify()是唤醒单个线程，而notifyAll()是唤醒所有的线程。

Object类中关于等待/唤醒的API详细信息如下： **notify()** -- 唤醒在此对象监视器上等待的单个线程。 **notifyAll()** -- 唤醒在此对象监视器上等待的所有线程。 **wait()** -- 让当前线程处于“等待(阻塞)状态”，“直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法”，当前线程被唤醒(进入“就绪状态”)。 **wait(long timeout)** -- 让当前线程处于“等待(阻塞)状态”，“直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量”，当前线程被唤醒(进入“就绪状态”)。 **wait(long timeout, int nanos)** -- 让当前线程处于“等待(阻塞)状态”，“直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量”，当前线程被唤醒(进入“就绪状态”)。

## 2. 为什么notify(), wait()等函数定义在Object中，而不是Thread中

Object中的wait(), notify()等函数，和synchronized一样，会对“对象的同步锁”进行操作。

wait()会使“当前线程”等待，因为线程进入等待状态，所以线程应该释放它锁持有的“同步锁”，否则其它线程获取不到该“同步锁”而无法运行！ OK，线程调用wait()之后，会释放它锁持有的“同步锁”；而且，根据前面的介绍，我们知道：等待线程可以被notify()或notifyAll()唤醒。现在，请思考一个问题：notify()是依据什么唤醒等待线程的？或者说，wait()等待线程和notify()之间是通过什么关联起来的？答案是：依据“对象的同步锁”。

负责唤醒等待线程的那个线程(我们称为“**唤醒线程**”)，它只有在获取“该对象的同步锁”(**这里的同步锁必须和等待线程的同步锁是同一个**)，并且调用notify()或notifyAll()方法之后，才能唤醒等待线程。虽然，等待线程被唤醒；但是，它不能立刻执行，因为唤醒线程还持有“该对象的同步锁”。必须等到唤醒线程释放了“对象的同步锁”之后，等待线程才能获取到“对象的同步锁”进而继续运行。

总之，notify(), wait()依赖于“同步锁”，而“同步锁”是对象锁持有，并且每个对象有且仅有一个！这就是为什么notify(), wait()等函数定义在Object类，而不是Thread类中的原因。

## 3. yield()介绍

**yield()的作用是让步。它能让当前线程由“运行状态”进入到“就绪状态”，从而让其它具有相同优先级的等待线程获取执行权；但是，并不能保证在当前线程调用yield()之后，其它具有相同优先级的线程就一定能获得执行权；也有可能是当前线程又进入到“运行状态”继续运行！**

## 4. yield() 与 wait()的比较

**我们知道，wait()的作用是让当前线程由“运行状态”进入“等待(阻塞)状态”的同时，也会释放同步锁。而yield()的作用是让步，它也会让当前线程离开“运行状态”。它们的区别是：(01) wait()是让线程由“运行状态”进入到“等待(阻塞)状态”，而不yield()是让线程由“运行状态”进入到“就绪状态”。(02) wait()是会线程释放它所持有对象的同步锁，而yield()方法不会释放锁。**

```
// YieldLockTest.java 的源码
public class YieldLockTest{

    private static Object obj = new Object();

    public static void main(String[] args){
        ThreadA t1 = new ThreadA("t1");
        ThreadA t2 = new ThreadA("t2");
        t1.start();
        t2.start();
    }
```

```
    static class ThreadA extends Thread{
        public ThreadA(String name){
            super(name);
        }
        public void run(){
            // 获取obj对象的同步锁
            synchronized (obj) {
                for(int i=0; i <10; i++){
                    System.out.printf("%s [%d]:%d\n", this.getName(), this.getPriority(),
i);
                    // i整除4时，调用yield
                    if (i%4 == 0)
                        Thread.yield();
                }
            }
        }
    }
}
```

**(某一次)运行结果**:

```
t1 [5]:0
t1 [5]:1
t1 [5]:2
t1 [5]:3
t1 [5]:4
t1 [5]:5
t1 [5]:6
t1 [5]:7
t1 [5]:8
t1 [5]:9
t2 [5]:0
t2 [5]:1
t2 [5]:2
t2 [5]:3
t2 [5]:4
t2 [5]:5
t2 [5]:6
t2 [5]:7
t2 [5]:8
t2 [5]:9
```

**结果说明**： 主线程main中启动了两个线程t1和t2。t1和t2在run()会引用同一个对象的同步锁，即synchronized(obj)。在t1运行过程中，虽然它会调用Thread.yield()；但是，t2是不会获取cpu执行权的。因为，t1并没有释放“obj所持有的同步锁”！

## 5. sleep()介绍

sleep() 定义在Thread.java中。 sleep() 的作用是让当前线程休眠，即当前线程会从“运行状态”进入到“休眠(阻塞)状态”。sleep()会指定休眠时间，线程休眠的时间会大于/等于该休眠时间；在线程重新被唤醒时，它会由“阻塞状态”变成“就绪状态”，从而等待cpu的调度执行。

## 6. sleep() 与 wait()的比较

**我们知道，wait()的作用是让当前线程由“运行状态”进入“等待(阻塞)状态”的同时，也会释放同步锁。而sleep()的作用是也是让当前线程由“运行状态”进入到“休眠(阻塞)状态”。但是，wait()会释放对象的同步锁，而sleep()则不会释放锁。下面通过示例演示sleep()是不会释放锁的。**

```
// SleepLockTest.java的源码
public class SleepLockTest{

    private static Object obj = new Object();

    public static void main(String[] args){
        ThreadA t1 = new ThreadA("t1");
        ThreadA t2 = new ThreadA("t2");
        t1.start();
        t2.start();
    }

    static class ThreadA extends Thread{
        public ThreadA(String name){
            super(name);
        }
        public void run(){
            // 获取obj对象的同步锁
            synchronized (obj) {
                try {
                    for(int i=0; i <10; i++){
                        System.out.printf("%s: %d\n", this.getName(), i);
                        // i能被4整除时，休眠100毫秒
                        if (i%4 == 0)
                            Thread.sleep(100);
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

主线程main中启动了两个线程t1和t2。t1和t2在run()会引用同一个对象的同步锁，即synchronized(obj)。在t1运行过程中，虽然它会调用Thread.sleep(100)；但是，t2是不会获取cpu执行权的。因为，t1并没有释放“obj所持有的同步锁”！ 注意，若我们注释掉synchronized (obj)后再次执行该程序，t1和t2是可以相互切换的。