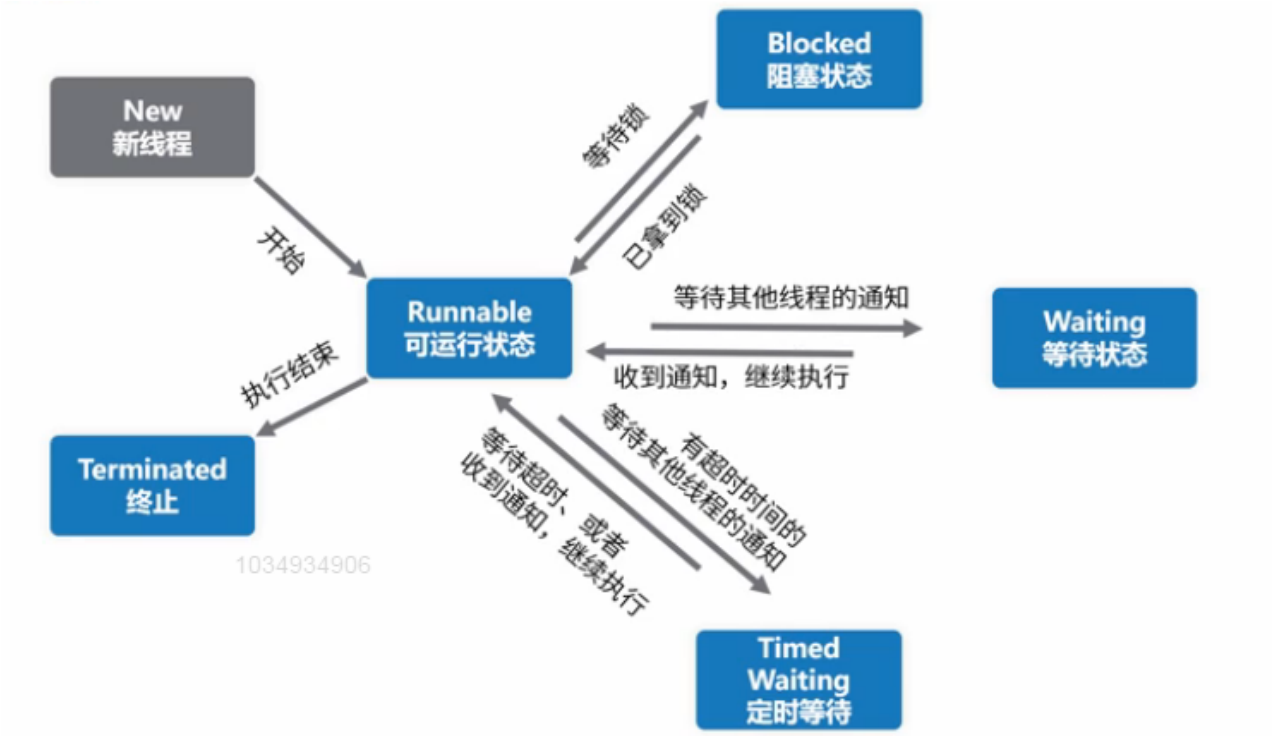
线程状态

线程状态

在java.lang.Thread.State中定义了6个线程状态

1. New，当我们定义了一个线程尚未调用start方法也即尚未启动的线程的线程状态。
2. Runnable，可运行线程的状态，等待CPU调度。
3. Blocked，线程阻塞等待监视器锁定的线程状态。处于Synchronized同步代码块或方法中被阻塞。
4. Waiting，等待线程的线程状态。Object.wiat，Thread.join，LockSupport.park的不带超时的方式。
5. Timed Waiting，具有指定等待时间的等待线程的线程状态。Thread.sleep，Object.wait，Thread.join，LockSupport.parkNanos，LockSupport.parkUntil的带超时的方式。
6. Terminated，终止线程的线程状态。线程正常完成执行或者出现异常。

   下面是线程状态切换的示意图：



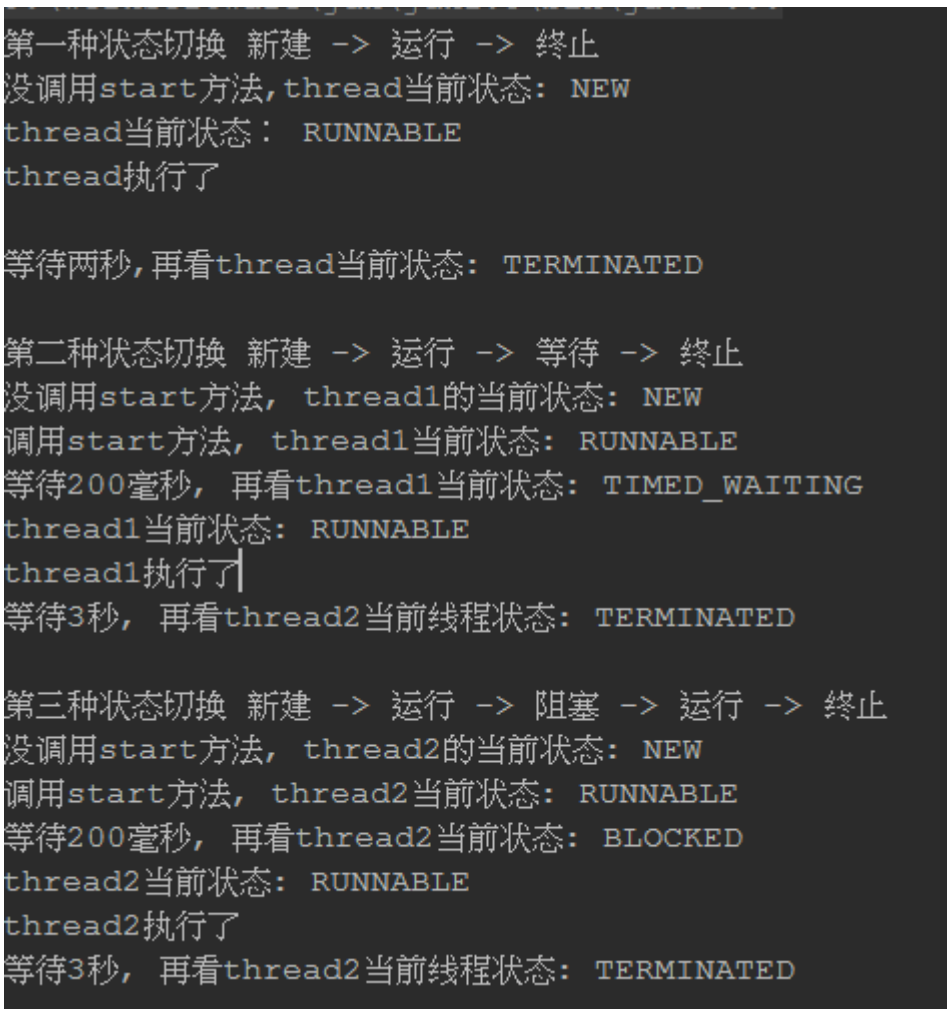
线程状态切换代码演示

```
public class ThreadStateDemo {

    public static Thread thread;

    public static ThreadStateDemo threadStateDemo;

    public static void main(String[] args) throws Exception {
```

```
        // 第一种状态切换 新建 -> 运行 -> 终止
        System.out.println("第一种状态切换 新建 -> 运行 -> 终止");
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("thread当前状态： " +
Thread.currentThread().getState().toString());
                System.out.println("thread执行了");
            }
        });
        System.out.println("没调用start方法,thread当前状态: " + thread.getState().toString());
        thread.start();
        Thread.sleep(2000L);
        System.out.println();
        System.out.println("等待两秒,再看thread当前状态: " + thread.getState().toString());
        // thread.start()线程终止后再调用start方法会抛出IllegalThreadStateException

        System.out.println();
        System.out.println("第二种状态切换 新建 -> 运行 -> 等待 -> 终止");
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(1500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("thread1当前状态: " +
Thread.currentThread().getState().toString());
                System.out.println("thread1执行了");
            }
        });
        System.out.println("没调用start方法, thread1的当前状态: " +
thread1.getState().toString());
        thread1.start();
        System.out.println("调用start方法, thread1当前状态: " +
thread1.getState().toString());
        Thread.sleep(200L);
        System.out.println("等待200毫秒, 再看thread1当前状态: " +
thread1.getState().toString());
        Thread.sleep(3000L);
        System.out.println("等待3秒, 再看thread2当前线程状态: " +
thread1.getState().toString());

        System.out.println();
        System.out.println("第三种状态切换 新建 -> 运行 -> 阻塞 -> 运行 -> 终止");
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronized (ThreadStateDemo.class) {
                    System.out.println("thread2当前状态: " +
Thread.currentThread().getState().toString());
                    System.out.println("thread2执行了");
```

```
                }
            }
        });

        synchronized (ThreadStateDemo.class) {
            System.out.println("没调用start方法，thread2的当前状态：" +
thread2.getState().toString());
            thread2.start();
            System.out.println("调用start方法，thread2当前状态：" +
thread2.getState().toString());
            Thread.sleep(200L);
            System.out.println("等待200毫秒，再看thread2当前状态：" +
thread2.getState().toString());
        }
        Thread.sleep(3000L);
        System.out.println("等待3秒，再看thread2当前线程状态：" +
thread2.getState().toString());
    }
}
```

运行结果如下图：



可见和我们状态转换图保持一致。