线程终止

不正确的线程终止-Stop

Stop，终止线程，并且清除监控器锁的信息，但是可能导致线程安全问题，它会强行终止线程，JDK不建议使用。

Destroy，JDK未实现。

下面我们来看一下调用Stop造成线程不安全的例子。

```
public class ThreadStopDemo {
    public static void main(String[] args) throws InterruptedException {
        StopThread stopThread = new StopThread();
        stopThread.start();
        Thread.sleep(1000);
        stopThread.stop();
        while (stopThread.isAlive()) {

        }
        stopThread.print();
    }
}

class StopThread extends Thread {

    private int i = 0;

    private int j = 0;

    @Override
    public void run() {
        synchronized (this) {
            i++;
            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            j++;
        }
    }

    public void print() {
        System.out.println("i = " + i + ", " + "j = " + j);
    }
}
```

预期结果i和j的值都应该是1，实际结果是i=1，j=0。stop破坏的同步代码块的线程安全语义。

正确的线程终止-interrupt

如果目标线程在调用Object的wait()，wait(long)或wait(long,int)方法，join()，join(long,int)或者sleep(long,int)方法时被阻塞，那么interrupt会生效，该线程的中断状态被清除，抛出InterruptedException异常。

如果目标线程是被I/O或者NIO中的Channel所阻塞，同样I/O操作会被中断或者返回特殊的异常值。达到终止线程的目的。

如果以上条件都不满足，则会设置此线程的中断状态。

正确的线程终止-标志位

代码逻辑中增加一个判断，用来控制线程执行的终止。

```
public class ThreadStopFlagDemo extends Thread {
  public volatile static boolean flag = true;

  public static void main(String[] args) throws InterruptedException {
    new Thread(() -> {
      try {
        while (flag) { // 判断是否运行
          System.out.println("运行中");
          Thread.sleep(1000L);
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }).start();
    // 3秒之后，将状态标志改为False，代表不继续运行
    Thread.sleep(3000L);
    flag = false;
    System.out.println("程序运行结束");
  }
}
```