停止线程

原理：使用interrupt来**通知**，而不是强制

1. 如何正确停止线程？interrupt 通知线程停止，需要启动的线程进行 '配合' 才能中断线程

   a. 普通情况中断线程

```
/**
 * Run 方法内没有sleep 或者 wait方法时，停止线程
 */
public class RightWayStopThreadWithoutSleep implements Runnable{
    @Override
    public void run() {
        int num = 0;
        // !Thread.currentThread().isInterrupted() 用于检测是否有中断信号通知
        while ( !Thread.currentThread().isInterrupted() && num <
Integer.MAX_VALUE / 2){
            if (num % 10000 == 0){
                System.out.println(num + "是 10000 的倍数");
            }
            num++;
        }
        System.out.println("任务结束! ");
    }

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

        Thread thread = new Thread(new RightWayStopThreadWithoutSleep());
        thread.start();  // 启动子线程
        Thread.sleep(1000);  // 睡眠1 m
        thread.interrupt(); // 中断线程，interrupe 只是通知，并不会进行强制中断，中
断线程需要 子线程的响应配合

    }
}
```

   b. 在 线程被阻塞的情况下，演示中是 线程 在sleep 的情况下

```
/**
 *  在 sleep的情况下中断线程， 在sleep情况下， 使用try -- catch 进行异常的捕获
 */
public class RightWayStopThreadWithSleep {

    public static void main(String[] args) throws InterruptedException {
        Runnable runnable = ()->{
            int num = 0;
            try {
                while (num < 300 && !Thread.currentThread().isInterrupted())
{
                    if (num % 100 == 0){
                        System.out.println(num + "是 100 的倍数! ");
                    }
                    num++;  // 如果该句 注销， 去掉
!Thread.currentThread().isInterrupted() 的方法，该程序会如何执行？
```

```
                    //  该线程会一直执行，不会被打断，不能被异常捕获，因为interrupt只是通知 线程需要中断，但是不能强制，没有措施让该线程退出循环，所以一直执行，与下一种方法 进行对比
                }

                 // 线程休眠 1 s
                 Thread.sleep(1000);
            } catch (InterruptedException e){  // 捕获到中断线程的异常，在sleep情况下， 使用try -- catch 进行异常的捕获，
                System.out.println("进入异常！！");
                e.printStackTrace();
    // 在sleep 的情况下， 捕获到 中断后，会消除 中断位， 所以在循环中出现 try catch 捕获 sleep 异常可能会出现不能中断的情况
                System.out.println(Thread.currentThread().isInterrupted());
            }
        };
        Thread thread = new Thread(runnable);
        thread.start();
        Thread.sleep(5); // 主线程 休眠
        thread.interrupt(); // 中断线程，子线程 在 sleep 的状态下被中断
    }
}
```

c.线程在每次迭代后都阻塞

```
/**
 *  如果在执行过程中，每次循环都会调用 sleep 或者 wait方法，则不要每次迭代都检查是否需要中断
 */
public class RightWayStopThreadWithSleepEveryLoop {

    public static void main(String[] args) throws InterruptedException {
        Runnable runnable = ()->{
            int num = 0;
            try {
                while (num < 10000 ){
                    if (num % 100 == 0){
                        System.out.println(num + "是 100 的倍数！");
                    }
//                    num++;
                    // 该睡眠（阻塞）是在 循环中， 与上一个方法中的注释对比， 该方法中去掉了 !Thread.currentThread().isInterrupted() 条件，但是一样会捕获到异常
                    Thread.sleep(10); // 说明在 睡眠的时候发生中断 通知可以进入到异常
                }

            } catch (InterruptedException e){  // 捕获到中断线程的异常，在sleep情况下， 使用try -- catch 进行异常的捕获
                System.out.println("进入异常！！");
                e.printStackTrace();
                System.out.println(Thread.currentThread().isInterrupted());
// false
            }
        };
        Thread thread = new Thread(runnable);
        thread.start();
```

```
        Thread.sleep(2000); // 主线程 休眠
        thread.interrupt(); // 中断线程，子线程 在 sleep 的状态下被中断
    }
}
```

d. 出现不能中断的情况

```
/**
 *  如果 while中放置  try/catch ，则会导致中断失效， 这是因为 sleep 会使中断标志位失效，当在大循环中进行 try catch ,虽然可以捕获中断，但是会马上进入下一次循环
 */
public class CantInterrupt {

    public static void main(String[] args) throws InterruptedException {
        Runnable runnable = ()->{
            int num = 0;
            while (num < 10000 && !Thread.currentThread().isInterrupted() ){
                if (num % 100 == 0){
                    System.out.println(num + "是 100 的倍数! ");
                }
                    num++;
                // 进行睡眠 ,循环内进行异常的捕获,
                if (Thread.currentThread().isInterrupted()){
                    System.out.println("********************"); // 不会打印
                }

                try {
                    Thread.sleep(10);// sleep 会消除中断的 标志位， 导致循环的中断标志位判断 !Thread.currentThread().isInterrupted() 失效
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };

        Thread thread = new Thread(runnable);
        thread.start();
        Thread.sleep(2000); // 主线程 休眠
        thread.interrupt(); // 中断线程
    }
}
```