线程中断机制

2022年10月21日 17:01

1、从阿里巴巴蚂蚁金服面试题开始

以下三个方法分别是什么？分别在什么场景下用？

void	**interrupt**() Interrupts this thread.
static boolean	**interrupted**() Tests whether the current thread has been interrupted.
boolean	**isAlive**() Tests if this thread is alive.
boolean	**isDaemon**() Tests if this thread is a daemon thread.
boolean	**isInterrupted**() Tests whether this thread has been interrupted.

- 如何中断一个运行中的线程？
- 如何停止一个运行中的线程？

莫慌，下面将会给出答案

2、什么是线程中断机制？

首先

一个线程不应该由其他线程来强制中断或停止，而是***应该由线程自己自行停止***，自己来决定自己的命运。

所以，Thread.stop，Thread.suspend，Thread.resumer都已经被弃用了

其次

在Java中没有办法立即停止一条线程，但是停止线程却显得尤为重要，如取消一个耗时操作。

因此，Java提供了一种用于停止线程的**协商机制**——中断，即中断标识协商机制。

中断只是一种协作协商机制，Java没有给中断增加任何语法，中断的过程完全需要程序员自己实现。

若要中断一个线程，你需要手动调用该线程的**interrupt**方法，该方法也仅仅是将线程对象的中断标识设成true；

接着你需要自己写代码不断地检测当前线程地标识位，如果为true，标识别的线程请求这条线程中断，

此时究竟该做什么需要你自己写代码实现。

每个线程对象中都有一个中断标识位，用于标识线程是否被中断；该标识位为true表示中断，为false表示未中断；

通过调用线程对象的interrupt方法将该线程的标识位设为true；可以在别的线程中调用，也可以在自己的线程中调用。

3、中断相关API方法之三大方法说明

public void interrupt()	实例方法，Just to set the interrupt flag 实例方法interrupt()仅仅是设置线程的中断状态为true，发起一个协商而不会立刻停止线程
public static boolean interrupted()	静态方法，Thread.interrupted() 判断线程是否被中断并清除当前中断状态。 这个方法做了两件事： 1 返回当前线程的中断状态，测试当前线程是否已被中断 2 将当前线程的中断状态清零并重新设为false，清除线程的中断状态 此方法有点不好理解，如果连续两次调用此方法，则第二次调用将返回false，因为连续调用两次的结果可能不一样
public boolean isInterrupted()	实例方法 判断当前线程是否被中断（通过检查中断标志位）

4、大厂面试题中断机制考点

（1）如何停止中断运行中的线程？

- 通过一个volatile变量实现

```
// volatile变量
private static volatile boolean isStop = false;

public static void main(String[] args) {
    new Thread(() -> {
        while (true) {
            if (isStop) {
                System.out.println(Thread.currentThread().getName() + "\t isStop被修改为true，程序停止");
                break;
            }
```

```
            System.out.println(Thread.currentThread().getName() + " running ------");
        }
    }, "t1").start();

    // 暂停20毫秒
    try {
        TimeUnit.MILLISECONDS.sleep(20);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    new Thread(() -> {
        // 设置标识位为true
        isStop = true;
    } ,"t2").start();
}
```

- 通过AtomicBoolean实现

```
// 原子变量
private static AtomicBoolean atomicBoolean = new AtomicBoolean(false);

public static void main(String[] args) {
    new Thread(() -> {
        while (true) {
            if (atomicBoolean.get()) {
                System.out.println(Thread.currentThread().getName() + "\t atomicBoolean被修改为true, 程序停止");
                break;
            }
            System.out.println(Thread.currentThread().getName() + " running ------");
        }
    } ,"t1").start();

    // 暂停20毫秒
    try {
        TimeUnit.MILLISECONDS.sleep(20);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    new Thread(() -> {
        // 设置标识位为true
        atomicBoolean.set(true);
    }, "t2").start();
}
```

- 通过Thread类自带的中断API实例方法实现

  在需要中断的线程中不断监听中断状态，一旦发生中断，就执行相应的中断处理业务逻辑stop线程

```
public static void main(String[] args) {
    Thread t1 = new Thread(() -> {
        while (true) {
            // 判断当前的线程标识位，如果位true，则中断线程
            if (Thread.currentThread().isInterrupted()) {
                System.out.println(Thread.currentThread().getName() + "\t Interrupted被修改为true, 程序停止");
                break;
            }
            System.out.println(Thread.currentThread().getName() + " running ------");
        }
    }, "t1");
    t1.start();

    System.out.println("-----t1的默认中断标识位: " + t1.isInterrupted());

    // 暂停20毫秒
    try {
        TimeUnit.MILLISECONDS.sleep(20);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    new Thread(() -> {
        // 调用interrupt方法，设置标识位为true
        t1.interrupt();
    }, "t2").start();
}
```

具体来说，当对一个线程，调用interrupt()时：

1. 如果线程处于**正常活动状态**，那么会将线程的中断标志设置为true，仅此而已。
   被设置中断标志的线程将继续正常运行，不受影响。
   所以，interrupt()并不能真正的中断线程，需要被调用的线程自己进行配合才行。

2. 如果线程处于被阻塞状态（例如处于sleep，wait，join等状态），在别的线程中调用当前线程对象的interrupt方法，
   那么线程将立即退出被阻塞状态，并抛出一个InterruptedException异常。

（2）当前线程的中断标识为true，是不是线程就立刻停止?

```
public static void main(String[] args) {
    // 实例方法interrupt()仅仅是设置线程的中断状态为true，并不会停止线程
    Thread t1 = new Thread(() -> {
        for (int i = 0; i < 300; i++) {
            System.out.println("----: " + i);
        }
        // true
        System.out.println("t1线程调用interrupt()后的中断标识位02: " + Thread.currentThread().isInterrupted());
    }, "t1");
    t1.start();

    System.out.println("t1默认的中断标识位: " + t1.isInterrupted()); // false

    try {
        // 暂停2毫秒
        TimeUnit.MILLISECONDS.sleep(2);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    t1.interrupt();
    System.out.println("t1线程调用interrupt()后的中断标识位01: " + t1.isInterrupted()); // true

    try {
        // 暂停2000毫秒
        TimeUnit.MILLISECONDS.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    // 暂停2000毫秒，此时t1线程已经死去，那么他的中断标识位为false
    // 即，如果线程是not alive的，那么他的中断标识位为false，不管此前是否设置过
    System.out.println("t1线程调用interrupt()后的中断标识位03: " + t1.isInterrupted()); // false
}
```

注意看下面这段代码：

```
public static void main(String[] args) {
    Thread t1 = new Thread(() -> {
        while (true) {
            if (Thread.currentThread().isInterrupted()) {
                System.out.println("t1线程 中断标识位: " + Thread.currentThread().isInterrupted() + " 程序停止");
                break;
            }
            // 如果线程里面有sleep wait方法，那么将会抛出下面异常，并且线程不会停止
            try {
                TimeUnit.MILLISECONDS.sleep(200);
            } catch (InterruptedException e) {
                // 如果没有这句话，那么会抛出异常，并且线程永远不会停止
                // 因为sleep抛出异常后，中断标识位被设置为false，所以就陷入死循环了!
                Thread.currentThread().interrupt();     // 再调用一次Interrupt，线程停止
                e.printStackTrace();
            }
            System.out.println("hello t1!");
        }
    }, "t1");
    t1.start();

    try {
        TimeUnit.MILLISECONDS.sleep(1);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    new Thread(() -> {
        t1.interrupt();
    }, "t2").start();
}
```

原因如下：

public void interrupt()

Interrupts this thread.

Unless the current thread is interrupting itself, which is always permitted, the checkAccess method of this thread is invoked, which may cause a SecurityException to be thrown.

If this thread is blocked in an invocation of the wait(), wait(long), or wait(long, int) methods of the Object class, or of the join(), join(long), join(long, int), sleep(long), or sleep(long, int), methods of this class, then its interrupt status will be cleared and it will receive an InterruptedException.

If this thread is blocked in an I/O operation upon an InterruptibleChannel then the channel will be closed, the thread's interrupt status will be set, and the thread will receive a ClosedByInterruptException.

If this thread is blocked in a Selector then the thread's interrupt status will be set and it will return immediately from the selection operation, possibly with a non-zero value, just as if the selector's wakeup method were invoked.

If none of the previous conditions hold then this thread's interrupt status will be set.

Interrupting a thread that is not alive need not have any effect.

Throws:

SecurityException - if the current thread cannot modify this thread

Sleep方法抛出InterruptedException后，中断标识也被清空置为false，我们在catch没有通过调用th.interrupt()方法再次将中断标识置为true，这就导致了无限循环

总结：中断只是一种协商机制，修改中断标识位仅此而已，不是立刻stop打断！

（3）静态方法Thread.interrupted()，谈谈你的理解

public static boolean interrupted()	静态方法，Thread.interrupted() 判断线程是否被中断并清除当前中断状态。 这个方法做了两件事： 1 返回当前线程的中断状态，测试当前线程是否已被中断 2 将当前线程的中断状态清零并重新设为false，清除线程的中断状态 此方法有点不好理解，如果连续两次调用此方法，则第二次调用将返回false，因为连续调用两次的结果可能不一样

中断标识被清空，如果该方法被连续调用两次，第二次调用将返回false。除非当前线程在第一次和第二次调用该方法之间再次被interrupt

```
// 测试当前线程是否被中断（检查中断标志），返回一个boolean并清除中断状态
// 第二次再调用时中断状态已经被清除，将返回一个false
System.out.println(Thread.currentThread().getName() + "\t" + Thread.interrupted()); // false
System.out.println(Thread.currentThread().getName() + "\t" + Thread.interrupted()); // false

System.out.println("----1");
Thread.currentThread().interrupt(); // 中断标识位设置为true
System.out.println("----2");

System.out.println(Thread.currentThread().getName() + "\t" + Thread.interrupted()); // true
System.out.println(Thread.currentThread().getName() + "\t" + Thread.interrupted()); // false
```

源码分析：

```
public static boolean interrupted() {
    // 调用isInterrupted方法，传递参数true，代表清除中断标识位
    return currentThread().isInterrupted(true);
}

public boolean isInterrupted() {
    // 调用isInterrupted方法，传递参数false，代表不清除中断标识位
    return isInterrupted(false);
}

// 底层是C
private native boolean isInterrupted(boolean ClearInterrupted);
```

5、线程中断相关的方法总结

public void interrupt()，interrupt()方法是一个实例方法

它通知目标线程中断，也仅是设置目标线程的中断标志位为true

public boolean isInterrupted()，isInterrupted()方法也是一个实例方法

它判断当前线程是否已被中断（通过检查中断标志位）并获取中断标志

public static boolean interrupted()，Thread类的静态方法interrupted()

返回当前线程的中断转换的真实值（boolean类型）后会将当前线程的中断状态设为false，

此方法调用之后会清除当前线程的中断标志位（将中断标志置位false），返回当前值并清零置false