如何正确的停止一个线程

superxcp
2018.08.28 19:56 字数 1367 阅读 301 评论 0 喜欢 0

停止一个线程意味着在任务处理完任务之前停掉正在做的操作，也就是放弃当前的操作。停止一个线程可以用Thread.stop()方法，但最好不要用它。虽然它确实可以停止一个正在运行的线程，但是这个方法是不安全的，而且是已被废弃的方法。
在java中有以下3种方法可以终止正在运行的线程：

使用退出标志，使线程正常退出，也就是当run方法完成后线程终止。
使用stop方法强行终止，但是不推荐这个方法，因为stop和suspend及resume一样都是过期作废的方法。
使用interrupt方法中断线程。

1. 停止不了的线程

   interrupt()方法的使用效果并不像for+break语句那样，马上就停止循环。调用interrupt方法是在当前线程中打了一个停止标志，并不是真的停止线程。

```java
public class MyThread extends Thread {
    public void run(){
        super.run();
        for(int i=0; i<500000; i++){
            System.out.println("i="+(i+1));
        }
    }
}

public class Run {
    public static void main(String args[]){
        Thread thread = new MyThread();
        thread.start();
        try {
            Thread.sleep(2000);
            thread.interrupt();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

输出结果：

```
...
i=499994
i=499995
i=499996
i=499997
i=499998
i=499999
i=500000
```

2. 判断线程是否停止状态

   Thread.java类中提供了两种方法：

this.interrupted(): 测试当前线程是否已经中断；
this.isInterrupted(): 测试线程是否已经中断；
那么这两个方法有什么图区别呢？
我们先来看看this.interrupted()方法的解释：测试当前线程是否已经中断，当前线程是指运行this.interrupted()方法的线程。

```java
public class MyThread extends Thread {
    public void run(){
        super.run();
        for(int i=0; i<500000; i++){
            i++;
//            System.out.println("i="+(i+1));
        }
    }
}

public class Run {
    public static void main(String args[]){
        Thread thread = new MyThread();
        thread.start();
        try {
            Thread.sleep(2000);
            thread.interrupt();

            System.out.println("stop 1??" + thread.interrupted());
            System.out.println("stop 2??" + thread.interrupted());
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

运行结果：

```
stop 1??false
stop 2??false
```

类Run.java中虽然是在thread对象上调用以下代码：thread.interrupt(), 后面又使用

```java
System.out.println("stop 1??" + thread.interrupted());
System.out.println("stop 2??" + thread.interrupted());
```

来判断thread对象所代表的线程是否停止，但从控制台打印的结果来看，线程并未停止，这也证明了interrupted()方法的解释，测试当前线程是否已经中断。这个当前线程是main，它从未中断过，所以打印的结果是两个false.

如何使main线程产生中断效果呢？

```java
public class Run2 {
    public static void main(String args[]){
        Thread.currentThread().interrupt();
        System.out.println("stop 1??" + Thread.interrupted());
        System.out.println("stop 2??" + Thread.interrupted());

        System.out.println("End");
    }
}
```

运行效果为：

```
stop 1??true
stop 2??false
End
```

方法interrupted()的确判断出当前线程是否是停止状态。但为什么第2个布尔值是false呢？ 官方帮助文档中对interrupted方法的解释：
测试当前线程是否已经中断。线程的中断状态由该方法清除。 换句话说，如果连续两次调用该方法，则第二次调用返回false。

下面来看一下inInterrupted()方法。

```java
public class Run3 {
    public static void main(String args[]){
        Thread thread = new MyThread();
        thread.start();
        thread.interrupt();
        System.out.println("stop 1??" + thread.isInterrupted());
        System.out.println("stop 2??" + thread.isInterrupted());
    }
}
```

运行结果：

```
stop 1??true
stop 2??true
```

isInterrupted()并为清除状态，所以打印了两个true。

3. 能停止的线程--异常法

   有了前面学习过的知识点，就可以在线程中用for语句来判断一下线程是否是停止状态，如果是停止状态，则后面的代码不再运行即可：

```java
public class MyThread extends Thread {
    public void run(){
        super.run();
        for(int i=0; i<500000; i++){
            if(this.interrupted()) {
                System.out.println("线程已经终止， for循环不再执行");
                break;
            }
            System.out.println("i="+(i+1));
        }
    }
}

public class Run {
    public static void main(String args[]){
        Thread thread = new MyThread();
        thread.start();
        try {
            Thread.sleep(2000);
            thread.interrupt();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

运行结果：

```
...
i=202053
i=202054
i=202055
i=202056
```

线程已经终止， for循环不再执行
上面的示例虽然停止了线程，但如果for语句下面还有语句，还是会继续运行的。看下面的例子：

```java
public class MyThread extends Thread {
    public void run(){
        super.run();
        for(int i=0; i<500000; i++){
            if(this.interrupted()) {
                System.out.println("线程已经终止， for循环不再执行");
                break;
            }
            System.out.println("i="+(i+1));
        }

        System.out.println("这是for循环外面的语句，也会被执行");
    }
}
```

使用Run.java执行的结果是：

```
...
i=180136
i=180137
i=180138
i=180139
```

线程已经终止， for循环不再执行
这是for循环外面的语句，也会被执行
如何解决语句继续运行的问题呢？ 看一下更新后的代码：

```java
public class MyThread extends Thread {
    public void run(){
        super.run();
        try {
            for(int i=0; i<500000; i++){
                if(this.interrupted()) {
                    System.out.println("线程已经终止， for循环不再执行");
                        throw new InterruptedException();
                }
                System.out.println("i="+(i+1));
            }

            System.out.println("这是for循环外面的语句，也会被执行");
        } catch (InterruptedException e) {
            System.out.println("进入MyThread.java类中的catch了。。。");
            e.printStackTrace();
        }
    }
}
```

使用Run.java运行的结果如下：

```
...
i=203798
i=203799
i=203800
线程已经终止， for循环不再执行
进入MyThread.java类中的catch了。。。
java.lang.InterruptedException
    at thread.MyThread.run(MyThread.java:13)
```

4. 在沉睡中停止

   如果线程在sleep()状态下停止线程，会是什么效果呢？

```java
public class MyThread extends Thread {
    public void run(){
        super.run();

        try {
            System.out.println("线程开始。。。");
            Thread.sleep(200000);
            System.out.println("线程结束。");
        } catch (InterruptedException e) {
            System.out.println("在沉睡中被停止, 进入catch， 调用isInterrupted()方法的结
            e.printStackTrace();
        }

    }
}
```

使用Run.java运行的结果是：

```
线程开始。。。
在沉睡中被停止, 进入catch， 调用isInterrupted()方法的结果是: false
java.lang.InterruptedException: sleep interrupted
    at java.lang.Thread.sleep(Native Method)
    at thread.MyThread.run(MyThread.java:12)
```

从打印的结果来看， 如果在sleep状态下停止某一线程，会进入catch语句，并且清除停止状态值，使之变为false。

前一个实验是先sleep然后再用interrupt()停止，与之相反的操作在学习过程中也要注意：

```java
public class MyThread extends Thread {
    public void run(){
        super.run();
        try {
            System.out.println("线程开始。。。");
            for(int i=0; i<10000; i++){
                System.out.println("i=" + i);
            }
            Thread.sleep(200000);
            System.out.println("线程结束。");
        } catch (InterruptedException e) {
             System.out.println("先停止，再遇到sleep，进入catch异常");
            e.printStackTrace();
        }

    }
}

public class Run {
    public static void main(String args[]){
        Thread thread = new MyThread();
        thread.start();
        thread.interrupt();
    }
}
```

运行结果：

```
i=9998
i=9999
先停止，再遇到sleep，进入catch异常
java.lang.InterruptedException: sleep interrupted
    at java.lang.Thread.sleep(Native Method)
    at thread.MyThread.run(MyThread.java:15)
```

5. 能停止的线程---暴力停止

   使用stop()方法停止线程则是非常暴力的。

```java
public class MyThread extends Thread {
    private int i = 0;
    public void run(){
        super.run();
        try {
            while (true){
                System.out.println("i=" + i);
                i++;
                Thread.sleep(200);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Run {
    public static void main(String args[]) throws InterruptedException {
        Thread thread = new MyThread();
        thread.start();
        Thread.sleep(2000);
        thread.stop();
    }
}
```

运行结果：

```
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9

Process finished with exit code 0
```

6.方法stop()与java.lang.ThreadDeath异常
调用stop()方法时会抛出java.lang.ThreadDeath异常，但是通常情况下，此异常不需要显示地捕捉。

```java
public class MyThread extends Thread {
    private int i = 0;
    public void run(){
        super.run();
        try {
            this.stop();
        } catch (ThreadDeath e) {
            System.out.println("进入异常catch");
            e.printStackTrace();
        }
    }
}

public class Run {
    public static void main(String args[]) throws InterruptedException {
        Thread thread = new MyThread();
        thread.start();
    }
}
```

stop()方法以及作废，因为如果强制让线程停止有可能使一些清理性的工作得不到完成。另外一个情况就是对锁定的对象进行了解锁，导致数据得不到同步的处理，出现数据不一致的问题。

7. 释放锁的不良后果

   使用stop()释放锁将会给数据造成不一致性的结果。如果出现这样的情况，程序处理的数据就有可能遭到破坏，最终导致程序执行的流程错误，一定要特别注意：

```java
public class SynchronizedObject {
    private String name = "a";
    private String password = "aa";

    public synchronized void printString(String name, String password){
        try {
            this.name = name;
            Thread.sleep(100000);
            this.password = password;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

public class MyThread extends Thread {
    private SynchronizedObject synchronizedObject;
    public MyThread(SynchronizedObject synchronizedObject){
        this.synchronizedObject = synchronizedObject;
    }

    public void run(){
        synchronizedObject.printString("b", "bb");
    }
}

public class Run {
    public static void main(String args[]) throws InterruptedException {
        SynchronizedObject synchronizedObject = new SynchronizedObject();
        Thread thread = new MyThread(synchronizedObject);
        thread.start();
        Thread.sleep(500);
        thread.stop();
        System.out.println(synchronizedObject.getName() + "  " + synchronizedObject.
    }
}
```

输出结果：

```
b  aa
```

由于stop()方法以及在JDK中被标明为“过期/作废”的方法，显然它在功能上具有缺陷，所以不建议在程序张使用stop()方法。

8. 使用return停止线程

   将方法interrupt()与return结合使用也能实现停止线程的效果：

```java
public class MyThread extends Thread {
    public void run(){
        while (true){
            if(this.isInterrupted()){
                System.out.println("线程被停止了！");
                return;
            }
            System.out.println("Time: " + System.currentTimeMillis());
        }
    }
}

public class Run {
    public static void main(String args[]) throws InterruptedException {
        Thread thread = new MyThread();
        thread.start();
        Thread.sleep(2000);
        thread.interrupt();
    }
}
```

输出结果：

```
...
Time: 1467072288503
Time: 1467072288503
Time: 1467072288503
线程被停止了！
```

不过还是建议使用“抛异常”的方法来实现线程的停止，因为在catch块中还可以将异常向上抛，使线程停止事件得以传播。