Java并发框架——AQS中断的支持

标签： 中断 多线程 抢占式 协作式 InterruptedException

2014年11月23日 22:49:15 2392人阅读 评论(1) 收藏

分类： 多线程&并发（23） JVM（31）

线程的定义给我们提供了并发执行多个任务的方式，大多数情况下我们会让每个任务都自行执行结束，这样事务的一致性，但是有时我们希望在任务执行中取消任务，使线程停止。在java中要让线程安全、快速、可下来并不是一件容易的事，java也没有提供任何可靠的方法终止线程的执行。回到第六小节，线程调度策略占式和协作式两个概念，与之类似的是中断机制也有协作式和抢占式。

历史上Java曾经使用stop()方法终止线程的运行，他们属于抢占式中断。但它引来了很多问题，早已被JD调用stop()方法则意味着①将释放该线程所持的所有锁，而且锁的释放不可控。②即刻将抛出ThreadDeat不管程序运行到哪里，但它不总是有效，如果存在被终止线程的锁竞争；第一点将导致数据一致性问题，这理解，一般数据加锁就是为了保护数据的一致性，而线程停止伴随所持锁的释放，很可能导致被保护的数据一致性，最终导致程序运算出现错误。第二点比较模糊，它要说明的问题就是可能存在某种情况stop()方法时终止线程，甚至可能终止不了线程。看如下代码会发生什么情况，看起来线程mt因为执行了stop()方止，按理来说就算execut方法是一个死循环，只要执行了stop()方法线程将结束，无限循环也将结束。其实因为我们在execute方法使用了synchronized修饰，同步方法表示在执行execute时将对mt对象进行加锁Thread的stop()方法也是同步的，于是在调用mt线程的stop()方法前必须获取mt对象锁，但mt对象锁被方法占用，且不释放，于是stop()方法永远获取不了mt对象锁，最后得到一个结论，使用stop()方法停止线靠，它未必总能有效终止线程。

```
public class ThreadStop {

public static voidmain(String[] args) {

        Thread mt= new MyThread();

        mt.start();

        try {

            Thread.currentThread().sleep(100);

        } catch(InterruptedException e) {

            e.printStackTrace();

        }

        mt.stop();

}


static classMyThread extends Thread {

        publicvoid run() {

            execute();

        }

        privatesynchronized void execute() {

            while(true) {

            }

        }
```

}

}

经历了很长时间的发展，Java最终选择用一种协作式的中断机制实现中断。协作式中断的原理很简单，其
对中断标识进行标记，某线程设置某线程的中断标识位，被标记了中断位的线程在适当的时间节点会抛出
获异常后做相应的处理。实现协作中断有三个要点需要考虑：①是在Java层面实现轮询中断标识还是在
现；②轮询的颗粒度的控制，一般颗粒度要尽量小周期尽量短以保证响应的及时性；③轮询的时间节点的
实就是在哪些方法里面轮询，例如JVM将Thread类的wait()、sleep()、join()等方法都实现中断标识的轮询

中断标识放在哪里？中断是针对线程实例而言，从Java层面上看，标识变量放到线程中肯定再合适不过了
由JVM维护，所以中断标识具体由本地方法维护。在Java层面仅仅留下几个API用于操作中断标识，如下，

```
public class Thread{

    public voidinterrupt() {……}

    public BooleanisInterrupted() {……}

    public static Booleaninterrupted() {……}

}
```

上面三个方法依次用于设置线程为中断状态、判断线程状态是否中断、清除当前线程中断状态并返回它之
通过interrupt()方法设置中断标识，假如在非阻塞线程则仅仅只是改变了中断状态，线程将继续往下运行
在可取消阻塞线程中，如正在执行sleep()、wait()、join()等方法的线程则会因为被设置了中断状
InterruptedException异常，程序对此异常捕获处理。

上面提到的三个要点，第一是轮询在哪个层面实现，这个没有特别的要求，在实际中只要不出现逻辑问题
层面或JVM层面实现都是可以的，例如常用的线程睡眠、等待等操作是通过JVM实现，而AQS框架里面
放到Java实现，不管在哪个层面上去实现，在轮询过程中都一定要能保证不会产生阻塞。第二是要保证轮
度尽可能的小周期尽可能短，这关系到中断响应的速度。第三点是关于轮询的时间节点的选取。

针对三要点来看看AQS框架中是如何支持中断的，主要在等待获取锁的过程中提供中断操作，下面是伪
需增加加红加粗部分逻辑即可实现中断支持，在循环体中每次循环都对当前线程中断标识位进行判断，一
线程被标记为中断则抛出InterruptedException异常，高层代码对此异常捕获处理即完成中断处理。总
是ASQ框架获取锁的中断机制是在Java层面实现的，轮询时间节点选择在不断做尝试获取锁操作过程中
环的颗粒度比较小，响应速度得以保证，且循环过程不存在阻塞风险，保证中断检测不会失效。

```
if(尝试获取锁失败) {

    创建node

    使用CAS方式把node插入到队列尾部

    while(true){

    if(尝试获取锁成功并且 node的前驱节点为头节点){

把当前节点设置为头节点

    跳出循环

}else{

    使用CAS方式修改node前驱节点的waitStatus标识为signal

    if(修改成功){

        挂起当前线程

        if(当前线程中断位标识为true)

            抛出InterruptedException异常

}

}
```

```
}
```

判断线程是否处于中断状态其实很简单，只需使用Thread.interrupted()操作，如果为true则说明线程处
位，并清除中断位。至此AQS实现了支持中断的获取锁操作。

此节从java发展过程分析了抢占式中断及协作式中断，由于抢占式存在一些缺陷现在已不推荐使用，而协
作为推荐做法，尽管在响应时间较长，但其具有无可比拟的优势。协作式中断我们可以在JVM层面实现，
以在Java层面实现，例如AQS框架的中断即是在Java层面实现，不过如果继续深究是因为Java留了几个AP
操作线程的中断标识位，这才使Java层面实现中断操作得以实现。对于java的协作式中断机制有人肯定有人
批评者说java没有抢占式中断机制，且协作式中断机制迫使开发者必须维护中断状态，迫使开发者必
InterruptedException。但肯定者则认为，虽然协作式中断机制推迟了中断请求的处理，但它为开发人员
灵活的中断处理策略，响应性可能不及抢占式，但程序健壮性更强。