二

17 如何正确关闭线程池？shutdown 和 shutdownNow 的区别？

在本课时我们主要学习如何正确关闭线程池？以及 shutdown() 与 shutdownNow() 方法的区别？首先，我们创建一个线程数固定为 10 的线程池，并且往线程池中提交 100 个任务，如代码所示。

```
ExecutorService service = Executors.newFixedThreadPool(10);
 for (int i = 0; i < 100; i++) {
    service.execute(new Task());
 }
```

那么如果现在我们想关闭该线程池该如何做呢？本课时主要介绍 5 种在 ThreadPoolExecutor 中涉及关闭线程池的方法，如下所示。

- void shutdown;
- boolean isShutdown;
- boolean isTerminated;
- boolean awaitTermination(long timeout, TimeUnit unit) throws InterruptedException;
- List shutdownNow;

下面我们就对这些方法逐一展开。

shutdown()

第一种方法叫作 shutdown()，它可以安全地关闭一个线程池，调用 shutdown() 方法之后线程池并不是立刻就被关闭，因为这时线程池中可能还有很多任务正在被执行，或是任务队列中有大量正在等待被执行的任务，调用 shutdown() 方法后线程池会在执行完正在执行的任务和队列中等待的任务后才彻底关闭。但这并不代表 shutdown() 操作是没有任何效果的，调用 shutdown() 方法后如果还有新的任务被提交，线程池则会根据拒绝策略直接拒绝后续

新提交的任务。

isShutdown()

第二个方法叫作 isShutdown()，它可以返回 true 或者 false 来判断线程池是否已经开始了关闭工作，也就是是否执行了 shutdown 或者 shutdownNow 方法。这里需要注意，如果调用 isShutdown() 方法的返回的结果为 true 并不代表线程池此时已经彻底关闭了，这仅仅代表线程池开始了关闭的流程，也就是说，此时可能线程池中依然有线程在执行任务，队列里也可能有等待被执行的任务。

isTerminated()

第三种方法叫作 isTerminated()，这个方法可以检测线程池是否真正“终结”了，这不仅代表线程池已关闭，同时代表线程池中的所有任务都已经都执行完毕了，因为我们刚才说过，调用 shutdown 方法之后，线程池会继续执行里面未完成的任务，不仅包括线程正在执行的任务，还包括正在任务队列中等待的任务。比如此时已经调用了 shutdown 方法，但是有一个线程依然在执行任务，那么此时调用 isShutdown 方法返回的是 true ，而调用 isTerminated 方法返回的便是 false ，因为线程池中还有任务正在在被执行，线程池并没有真正“终结”。直到所有任务都执行完毕了，调用 isTerminated() 方法才会返回 true，这表示线程池已关闭并且线程池内部是空的，所有剩余的任务都执行完毕了。

awaitTermination()

第四个方法叫作 awaitTermination()，它本身并不是用来关闭线程池的，而是主要用来判断线程池状态的。比如我们给 awaitTermination 方法传入的参数是 10 秒，那么它就会陷入 10 秒钟的等待，直到发生以下三种情况之一：

1. 等待期间（包括进入等待状态之前）线程池已关闭并且所有已提交的任务（包括正在执行的和队列中等待的）都执行完毕，相当于线程池已经“终结”了，方法便会返回 true；
2. 等待超时时间到后，第一种线程池“终结”的情况始终未发生，方法返回 false；
3. 等待期间线程被中断，方法会抛出 InterruptedException 异常。

也就是说，调用 awaitTermination 方法后当前线程会尝试等待一段指定的时间，如果在等待时间内，线程池已关闭并且内部的任务都执行完毕了，也就是说线程池真正“终结”了，那么方法就返回 true，否则超时返回 fasle。

我们则可以根据 awaitTermination() 返回的布尔值来判断下一步应该执行的操作。

shutdownNow()

最后一个方法是 shutdownNow()，也是 5 种方法里功能最强大的，它与第一种 shutdown 方法不同之处在于名字中多了一个单词 Now，也就是表示立刻关闭的意思。在执行 shutdownNow 方法之后，首先会给所有线程池中的线程发送 interrupt 中断信号，尝试中断这些任务的执行，然后会将任务队列中正在等待的所有任务转移到一个 List 中并返回，我们可以根据返回的任务 List 来进行一些补救的操作，例如记录在案并在后期重试。shutdownNow() 的源码如下所示。

```
public List<Runnable> shutdownNow() {
    List<Runnable> tasks;
    final ReentrantLock mainLock = this.mainLock;
    mainLock.lock();
    try {
        checkShutdownAccess();
        advanceRunState(STOP);
        interruptWorkers();
        tasks = drainQueue();
    } finally {
        mainLock.unlock();
    }
    tryTerminate();
    return tasks;
}
```

你可以看到源码中有一行 interruptWorkers() 代码，这行代码会让每一个已经启动的线程都中断，这样线程就可以在执行任务期间检测到中断信号并进行相应的处理，提前结束任务。这里需要注意的是，由于 Java 中不推荐强行停止线程的机制的限制，即便我们调用了 shutdownNow 方法，如果被中断的线程对于中断信号不理不睬，那么依然有可能导致任务不会停止。可见我们在开发中落地最佳实践是很重要的，我们自己编写的线程应当具有响应中断信号的能力，正确停止线程的方法在第 2 讲有讲过，应当利用中断信号来协同工作。

在掌握了这 5 种关闭线程池相关的方法之后，我们就可以根据自己的业务需要，选择合适的方法来停止线程池，比如通常我们可以用 shutdown() 方法来关闭，这样可以让已提交的任务都执行完毕，但是如果情况紧急，那我们就可以用 shutdownNow 方法来加快线程池“终结”的速度。

上一页 下一页