以ThreedPoolExecutor为例，来介绍下如何优雅的关闭线程池。

1.线程中断：

在介绍线程池关闭之前，先介绍下Thread的interrupt

在程序中，我们是不能随便中断一个线程的，因为这是极其不安全的操作，我们无法知道这个线程正运行在什么状态，它可能持有某把锁，强行中断可能导致锁不能释放的问题；或者线程可能在操作数据库，强行中断导致数据不一致混乱的问题。正因此，java里将Thread的stop方法设置为过时，以禁止大家使用。

一个线程什么时候可以退出呢？当然只有线程自己才能知道。

所以我们这里要说的Thread的interrupt方法，本质不是用来中断一个线程。是将线程设置一个中断状态。

当我们调用线程的interrupt方法，它有两个作用：

1.如果此线程处于阻塞状态（比如调用了wait方法，io等待），则会立马退出阻塞，并抛出InterruptedException异常，然后让线程退出。

2.如果此线程正处于运行之中，则线程不受任何影响，继续运行，仅仅是线程的中断标记被设置为true。所以线程要在适当的位置通过调用isinterrupted方法来查看自己是否被中断，并做退出操作。

线程池的关闭：

线程池提供了两个关闭的方法：shutdownNow和shutdown方法 。

shutdownNow方法的解释是：线程池拒接收新提交的任务，同时立马关闭线程池，线程池里的任务不再执行。

shutdown方法的解释是：线程池拒接收新提交的任务，同时等待线程池里的任务执行完毕后关闭线程池。

以上的说法虽然没错，但是还有很多细节，比如调用shutdown方法后，正在执行任务的线程做出什么反应？？正在等待任务的线程又做出什么反应？线程在什么情况下才会彻底退出。如果不了解这些细节，在关闭线程池时就难免遇到像线程池关闭不了、关闭线程池出现报错等情况。

再说这些关闭线程池细节之前，需要强调的一点是，调用完shutdownNow和shutdown方法后，并不代表线程池已经完成关闭操作，它只是异步的通知线程池进行关闭处理。如果要同步等待线程池彻底关闭后才继续往下执行，需要调用awaitTermination方法进行同步等待。

shutdownNow

我们看一下shutdownNow方法的源码：

```
public List<Runnable> shutdownNow() {
    List<Runnable> tasks;
    final ReentrantLock mainLock = this.mainLock;
    mainLock.lock();
    try {
        checkShutdownAccess();
        advanceRunState(STOP);1
        interruptWorkers();2
        tasks = drainQueue();3
    } finally {
        mainLock.unlock();
    }
    tryTerminate();
    return tasks;
}
```

在shutdownNow方法里，重要的三句代码如上：

第一句就是原子性的修改线程池的状态为stop状态。

第三句就是将队列里面还没有执行的任务放到列表中，返回给调用方。

第二句是遍历线程池里的所有工作线程，然后调用线程的interrupt方法。如下图：

记住看：：

https://www.cnblogs.com/qingquanzi/p/9018627.html

https://hacpai.com/article/1488023925829