冯师兄，昨天查了一下，发现真的没有一劳永逸直接杀死线程的方法，原本有的stop方法被禁用了，因为线程不安全。

网上查了一下java5以后的解决方案都是设置状态位。

所以我还是打算用您昨天给我讲的方法，但是为了尽可能的减少底层线程代码的污染，现在的具体实现是这样的：

1. 将大线程的开始运行以及等待结束方式改写成start()和join()的组合，防止CountDownLatch状态位对提前结束操作产生的干扰（直觉）。
2. 编写单例全局变量endFlag，并用volatile关键字修饰保证其原子性。
3. 在总线程开始全部任务之前，必须把endFlag的值设置为false，然后开始线程。
4. 在所有内部小线程的主要调用方法中，凡是遇到循环的开始，或是觉得比较耗时的操作之前，都加上对endFlag的判断，一旦发现为true，小线程内部逻辑程序return；结束，这样子虽然没做完，也是正常的结束，即便有延迟，也不会时间太长。
5. 如果想结束线程，把状态位改成true即可，小线程内部程序跑着跑着自己判断出状态位已经变成了true，就自己跳出了，把做到一半的阶段任务杂质（已经存的日志等数据库）清除。
6. 这样大线程不提供结束的操作，大线程的任务就是执行完小线程，不管是正常运行结束，还是发现状态位为true提前结束，还是异常退出，反正执行结束了任务就完成了。

这是我测试程序的某一次运行结果（又可能不一样，稍有延迟，但是很快就结束了）：

begin!

Thread-0

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

0

1

2

Thread-8

Thread-6

Thread-4

Thread-2

Thread-9

Thread-7

Thread-5

Thread-3

Thread-1

all thread has over!

冯师兄帮我看一下，这样子的方案可不可行，还有什么漏洞？完了我再向您请教，嘿嘿。