对线程池 ExecutorService 的各种关闭方式的研究

作者：zjhch123
原文链接：https://ld246.com/article/1488023925829
来源网站：链滴

对线程池ExecutorService的关闭的研究

最近在使用ExecutorService的时候，对其关闭操作的概念非常模糊。查阅了许多文章、问答之后，了一个总结。

1. 概述

Java提供的接口 java.util.concurrent.ExecutorService是一种异步执行的机制，可以让任务在后台执。其实例就像一个线程池，可以对任务进行统一的管理。

2. 研究

1. 理论研究

Java提供的对ExecutorService的关闭方式有两种，一种是调用其shutdown()方法，另一种是调用shudownNow()方法。这两者是有区别的。

以下内容摘自源代码内的注释

// shutdown()
Initiates an orderly shutdown in which previously submitted tasks are executed, but no new taks will be accepted.
Invocation has no additional effect if already shut down.
This method does not wait for previously submitted tasks to complete execution. Use awaitTrmination to do that.

// shutdownNow()
Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.
This method does not wait for actively executing tasks to terminate. Use awaitTermination to do that.
There are no guarantees beyond best-effort attempts to stop processing actively executing taks. For example, typical implementations will cancel via interrupt, so any task that fails to rescond to interrupts may never terminate.

两大段英文是什么意思呢？我来简单总结一下。

shutdown:

1. 调用之后不允许继续往线程池内继续添加线程;
2. 线程池的状态变为 SHUTDOWN状态;
3. 所有在调用 shutdown()方法之前提交到ExecutorSrvice的任务都会执行;
4. 一旦所有线程结束执行当前任务， ExecutorService才会真正关闭。

shutdownNow():

1. 该方法返回尚未执行的task的List;
2. 线程池的状态变为 STOP状态;
3. 阻止所有正在等待启动的任务,并且停止当前正在执行的任务;

简单点来说，就是:

shutdown()调用后，不可以再submit新的task，已经submit的将继续执行

shutdownNow()调用后，试图停止当前正在执行的task，并返回尚未执行的task的list

2. 源码分析

针对源代码的分析，可以参考这篇文章

JAVA线程池shutdown和shutdownNow的区别

3. 实战

所有实战代码，均可在github上下载并使用。github

1. Test1

```
// Test1
ExecutorService service = Executors.newFixedThreadPool(2);
Runnable run = () -> {
    try {
        Thread.sleep(5000);
        System.out.println("thread finish");
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
};
service.execute(run);
service.shutdown();
service.execute(run);
```

当调用shutdown()之后，将不能继续添加任务，否则会抛出异常RejectedExecutionException。并当正在执行的任务结束之后才会真正结束线程池。shutdownNow()的试验可以看Test2。

2. Test2

```
// Test2
ExecutorService service = Executors.newFixedThreadPool(2);
Runnable run = () -> {
    try{
        Thread.sleep(5000);
        System.out.println("thread finish");
    }catch(InterruptedExceptione){
        e.printStackTrace();
    }
};

service.execute(run);
service.shutdownNow();
```

使用shutdownNow()，若线程中有执行sleep/wait/定时锁等，直接终止正在运行的线程并抛出interrpt异常。因为其内部是通过Thread.interrupt()实现的。

但是这种方法有很强的局限性。因为如果线程中没有执行sleep等方法的话，其无法终止线程。如接来的Test3所示。

3. Test3

```
// Test3
ExecutorService service = Executors.newFixedThreadPool(1);
Runnable run = () -> {
    long num = 0;
    boolean flag = true;
    while(flag) {
        num += 1;
        if(num == Long.MAX_VALUE) flag = false;
    }
};
service.execute(run);
service.shutdownNow();
```

很多代码中都会有这样的情况，比方说使用循环标记flag循环执行一些耗时长的计算任务， 直到满足个条件之后才设置循环标记为false。

如Test3代码所示(循环等待的情况)，shutdownNow()无法终止线程。

如果遇到这种情况，可以使用如Test4中的方法。

4. Test4

```
// Test4
ExecutorService service = Executors.newFixedThreadPool(1);
Runnable run = () -> {
    long sum = 0;
    while(true && !Thread.currentThread().isInterrupted()) {
        sum += 1;
    }
};
service.execute(run);
service.shutdownNow();
```

对于循环等待的情况，可以引入变量Thread.currentThread().isInterrupted()来作为其中的一个判断件。

具体可参见Stop an infinite loop in an ExecutorService task

isInterrupted()方法返回当前线程是否有被interrupt。

shutdownNow()的内部实现实际上就是通过interrupt来终止线程，所以当调用shutdownNow()时，isInterrupted()会返回true。

此时就可以跳出循环等待。

然而这也不是最优雅的解决方式，具体可以参见Test5。

5. Test5

```
// Test5
ExecutorService service = Executors.newFixedThreadPool(1);
```

```
Runnable run = () -> {
    long sum = 0;
    boolean flag = true;
    while(flag && !Thread.currentThread().isInterrupted()) {
        sum += 1;
        if(sum == Long.MAX_VALUE) flag = false;
    }
};
service.execute(run);
service.shutdown();
try {
    if(!service.awaitTermination(2, TimeUnit.SECONDS)) {
        service.shutdownNow();
    }
} catch (InterruptedException e) {
    service.shutdownNow();
}
```

这里。先调用shutdown()使线程池状态改变为SHUTDOWN，线程池不允许继续添加线程，并且等正在执行的线程返回。

调用awaitTermination设置定时任务，代码内的意思为2s后检测线程池内的线程是否均执行完毕（就老师告诉学生，“最后给你2s钟时间把作业写完”），若没有执行完毕，则调用shutdownNow()方。

4. 关于更多

关于shutdown(), shutdownNow()和awaitTermination()方法，我在网上发现一个非常优雅的举例是一位日本人写的文章

我搬运一下译文。

翻译[Java]ExecutorService的正确关闭方法

3. 参考资料

1. Stop an infinite loop in an ExecutorService task
2. ExecutorService对象的shutdown()和shutdownNow()的区别
3. shutdown和shutdownNow--多线程任务的关闭（转）
4. 线程服务ExecutorService的操作shutdown方法和shutdownNow方法
5. ExecutorService 的理解与使用
6. 翻译[Java]ExecutorService的正确关闭方法