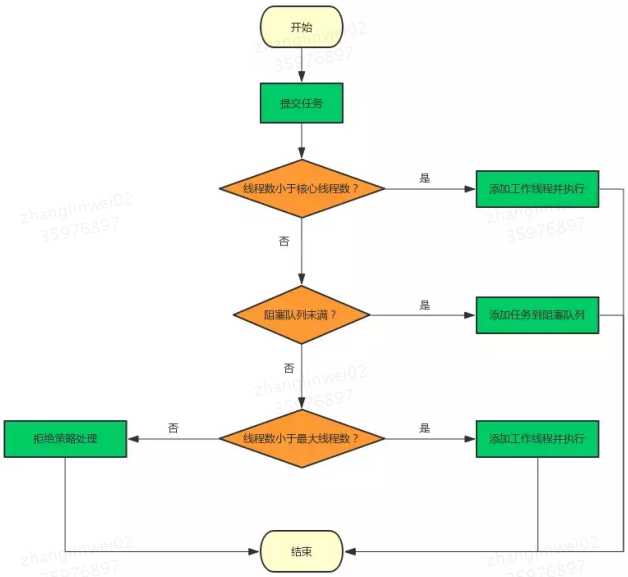
ThreadPoolExector

C-2 创建: 张林伟, 最后修改: 张林伟 今天 11:27

execute方法工作流程：



1. 当workerCount < corePoolSize，创建线程执行任务。
2. 当workerCount >= corePoolSize&&阻塞队列workQueue未满，把新的任务放入阻塞队列。
3. 当workQueue已满，并且workerCount >= corePoolSize，并且workerCount < maximumPoolSize，创建线程执行任务。
4. 当workQueue已满，workerCount >= maximumPoolSize，采取拒绝策略,默认拒绝策略是直接抛异常。

Worker.java

代码块 Java

```java
private final class Worker
    extends AbstractQueuedSynchronizer
    implements Runnable {

        ...
        public void run() {
            runWorker(this);
        }
        ...
    }

final void runWorker(Worker w) {
    // 拿到当前线程
    Thread wt = Thread.currentThread();
    // 拿到当前任务
    Runnable task = w.firstTask;
    // 将Worker.firstTask置空 并且释放锁
    w.firstTask = null;
    w.unlock(); // allow interrupts
    boolean completedAbruptly = true;
    try {
        // 如果task或者getTask不为空，则一直循环
        while (task != null || (task = getTask()) != null) {
            // 加锁
            w.lock();
            // If pool is stopping, ensure thread is interrupted;
            // if not, ensure thread is not interrupted.  This
            // requires a recheck in second case to deal with
            // shutdownNow race while clearing interrupt
```

```
            //  return ctl.get() >= stop
            // 如果线程池状态>=STOP 或者 (线程中断且线程池状态>=STOP)且当前线程没有中断
            // 其实就是保证两点：
            // 1. 线程池没有停止
            // 2. 保证线程没有中断
            if ((runStateAtLeast(ctl.get(), STOP) ||
                 (Thread.interrupted() &&
                  runStateAtLeast(ctl.get(), STOP))) &&
                !wt.isInterrupted())
                // 中断当前线程
                wt.interrupt();
            try {
                // 空方法
                beforeExecute(wt, task);
                Throwable thrown = null;
                try {
                    // 执行run方法(Runable对象)
                    task.run();
                } catch (RuntimeException x) {
                    thrown = x; throw x;
                } catch (Error x) {
                    thrown = x; throw x;
                } catch (Throwable x) {
                    thrown = x; throw new Error(x);
                } finally {
                    afterExecute(task, thrown);
                }
            } finally {
                // 执行完后， 将task置空， 完成任务++， 释放锁
                task = null;
                w.completedTasks++;
                w.unlock();
            }
        }
        completedAbruptly = false;
    } finally {
        // 退出工作
        processWorkerExit(w, completedAbruptly);
    }
```

Worker为什么不使用ReentrantLock来实现呢?

tryAcquire方法它是不允许重入的，而ReentrantLock是允许重入的。对于线程来说，如果线程正在执行是不允许其它锁重入进来的。

线程只需要两个状态，一个是独占锁，表明正在执行任务；一个是不加锁，表明是空闲状态。

runWorker方法的执行过程：

1. while循环中，不断地通过getTask()方法从workerQueue中获取任务
2. 如果线程池正在停止，则中断线程。否则调用3.
3. 调用task.run()执行任务；
4. 如果task为null则跳出循环，执行processWorkerExit()方法，销毁线程workers.remove(w);

参考资料：

https://juejin.im/post/5c9c98a85188252d8b13bc14

仅供内部使用，未经授权，切勿外传