**ThreadPoolExecutor的execute方法**

步骤一：

tp = new ThreadPoolExecutor(params) 如下：

public ThreadPoolExecutor(int corePoolSize,

                          int maximumPoolSize,

                          long keepAliveTime,

                          TimeUnit unit,

                          BlockingQueue<Runnable> workQueue,

                          ThreadFactory threadFactory,

                          RejectedExecutionHandler handler)

步骤二：

tp.execute(command)

public void execute(Runnable command)

背景知识引入ctl

private final AtomicInteger ctl = new AtomicInteger(ctlOf(RUNNING, 0));

**这个变量使用来干嘛的呢？**

它的作用有点类似我们在《ReadWriteLock接口及其实现ReentrantReadWriteLock》中提到的读写锁有读、写两个同步状态，而AQS则只提供了state一个int型变量，此时将state高16位表示为读状态，低16位表示为写状态。这里的clt同样也是，**它表示了两个概念**：

workerCount：当前有效的线程数

runState：当前线程池的五种状态，Running、Shutdown、Stop、Tidying、Terminate。

int型变量一共有32位，线程池的五种状态runState至少需要3位来表示，故workCount只能有29位，所以代码中规定线程池的有效线程数最多为2^29-1。

//ThreadPoolExecutor

private static final int COUNT\_BITS = Integer.SIZE – 3;  //32-3=29，线程数量所占位数

private static final int CAPACITY = (1 << COUNT\_BITS) – 1; //低29位表示最大线程数，2^29-1

//五种线程池状态

private static final int RUNNING = -1 << COUNT\_BITS; /int型变量高3位（含符号位）101表RUNING

private static final int SHUTDOWN = 0 << COUNT\_BITS; //高3位000

private static final int STOP = 1 << COUNT\_BITS; //高3位001

private static final int TIDYING = 2 << COUNT\_BITS; //高3位010

private static final int TERMINATED = 3 << COUNT\_BITS; //高3位011

execute(Runnable command)

//ThreadPoolExecutor#execute

/\*\*

\* Executes the given task sometime in the future.  The task

\* may execute in a new thread or in an existing pooled thread.

\*

\* If the task cannot be submitted for execution, either because this

\* executor has been shutdown or because its capacity has been reached,

\* the task is handled by the current {@code RejectedExecutionHandler}.

\*

\* @param command the task to execute

\* @throws RejectedExecutionException at discretion of

\*         {@code RejectedExecutionHandler}, if the task

\*         cannot be accepted for execution

\* @throws NullPointerException if {@code command} is null

\*/

public void execute(Runnable command) {

    if (command == null)

        throw new NullPointerException();

    /\*

     \* Proceed in 3 steps:

     \*

     \* 1. If fewer than corePoolSize threads are running, try to

     \* start a new thread with the given command as its first

     \* task.  The call to addWorker atomically checks runState and

     \* workerCount, and so prevents false alarms that would add

     \* threads when it shouldn't, by returning false.

     \*

     \* 2. If a task can be successfully queued, then we still need

     \* to double-check whether we should have added a thread

     \* (because existing ones died since last checking) or that

     \* the pool shut down since entry into this method. So we

     \* recheck state and if necessary roll back the enqueuing if

     \* stopped, or start a new thread if there are none.

     \*

     \* 3. If we cannot queue task, then we try to add a new

     \* thread.  If it fails, we know we are shut down or saturated

     \* and so reject the task.

     \*/

    //由ctl.get()获取当前有效的线程数和线程池的状态

    int c = ctl.get();

    //workerCountOf(c)获取当前的线程数

    if (workerCountOf(c) < corePoolSize) {

        //创建工作线程执行任务

        if (**addWorker(command, true)**)

            return;

        c = ctl.get();

    }

    //isRunning(c)判断线程池在运行状态

    //workQueue是创建线程池时，用于存放线程的队列

    if (isRunning(c) && workQueue.offer(command)) {

        //由ctl.get()获取当前有效的线程数和线程池的状态

        int recheck = ctl.get();

        //再次检测线程池的运行状态

        //如果线程池没有在Running状态，将刚刚放入

        if (! isRunning(recheck) && **remove(command))**

            //根据具体的拒绝策略，拒绝command

            reject(command);

        //再次检查当前线程池中的线程数量

        else if (workerCountOf(recheck) == 0)

**//增加空的worker**

**addWorker(null, false);**

    }

    //如果线程池没有在Running状态，或command入队失败，才执行到下面代码

    //创建新的worker，失败则拒绝

    else if (!**addWorker(command, false)**)

        reject(command);

}

翻译

原谅我蹩脚的英语，只能借助google翻译

/\*\*

\* Executes the given task sometime in the future.  The task

\* may execute in a new thread or in an existing pooled thread.

\*

\* If the task cannot be submitted for execution, either because this

\* executor has been shutdown or because its capacity has been reached,

\* the task is handled by the current {@code RejectedExecutionHandler}.

\*

\* @param command the task to execute

\* @throws RejectedExecutionException at discretion of

\*         {@code RejectedExecutionHandler}, if the task

\*         cannot be accepted for execution

\* @throws NullPointerException if {@code command} is null

\*/

在将来的某个时间执行给定的任务，该任务可以在新线程或现有池线程中执行。

如果由于线程池已关闭或已达到Queue最大长度而无法提交任务执行，则该任务由当前的

{@code RejectedExecutionHandler}处理

    /\*

     \* Proceed in 3 steps:

     \*

     \* 1. If fewer than corePoolSize threads are running, try to

     \* start a new thread with the given command as its first

     \* task.  The call to addWorker atomically checks runState and

     \* workerCount, and so prevents false alarms that would add

     \* threads when it shouldn't, by returning false.

     \*

     \* 2. If a task can be successfully queued, then we still need

     \* to double-check whether we should have added a thread

     \* (because existing ones died since last checking) or that

     \* the pool shut down since entry into this method. So we

     \* recheck state and if necessary roll back the enqueuing if

     \* stopped, or start a new thread if there are none.

     \*

     \* 3. If we cannot queue task, then we try to add a new

     \* thread.  If it fails, we know we are shut down or saturated

     \* and so reject the task.

     \*/

1.如果正在运行的线程少于corePoolSize线程，请尝试使用给定的命令启动第一个任务。 对addWorker的调用自动检查runState和workerCount，因此可以防止假警报并通过返回false拒绝不应该执行的线程。

2.如果任务可以成功入队，那么我们仍然需要再次检查我们是否应该添加线程（因为现有的自上次检查后死亡）或自从进入此方法以来，该线程池池已关闭。 所以我们重新检查状态，并在必要时回退队列，或者如果没有线程，则启动一个新线程

3.如果我们无法将任务入队，那么我们尝试添加一个新的线程。 如果失败，那么说明线程池关闭或任务队列饱和因此拒绝任务。

参考地址

[https://www.jb51.net/article/116521.html](https://www.jb51.net/article/116521.htm)