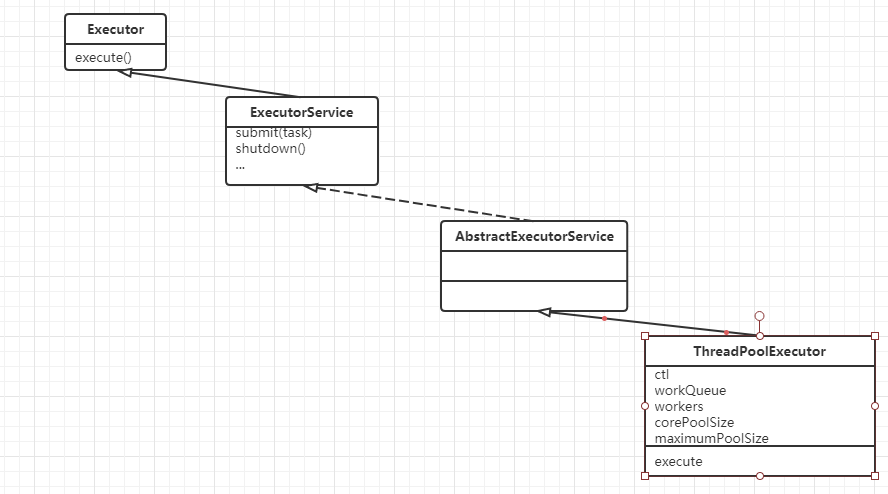
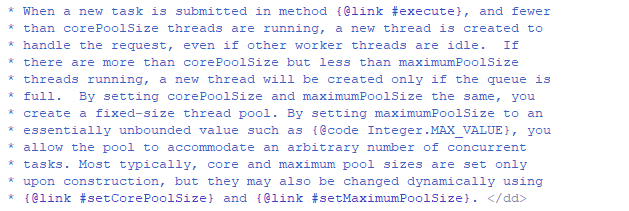
# ThreadPoolExecutor(jdk1.7.0\_67)

先看一下ThreadPoolExecutor的类层次结构。ThreadPoolExecutor继承AbstractExecutorService抽象类，而AbstractExecutorService实现了ExecutorService接口，ExecutorService继承了Executor接口。在ExecutorService接口中，声明了submit方法，如果想使用submit提交任务，必须将ThreadPoolExecutor强转为ExecutorService类型才可以，在submit方法中调用了execute方法，下面会将两个方法的不同之处。

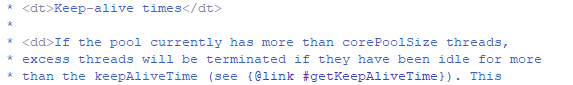


# 类的说明

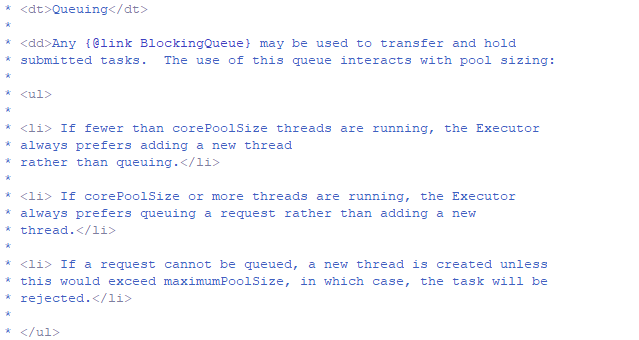
在类注释中有几个很重要的点我们一次来看一下。



这里简要说明了当执行一个任务时的流程。当调用execute方法时，如果线程池中的线程数小于corePoolSize，那么不管线程池中的线程是否处于空闲状态，都会创建一个新的线程处理请求。如果线程数大于corePoolSize小于maximumPoolSize，那么将请求添加到任务队列中；如果任务队列已满，创建新的线程处理请求。一般情况下corePoolSize和maximumPoolSize是通过在构造器执行时设置的，但是也可以通过调用setCorePoolSize和setMaximumPoolSize更改corePoolSize和maximumPoolSize的值。

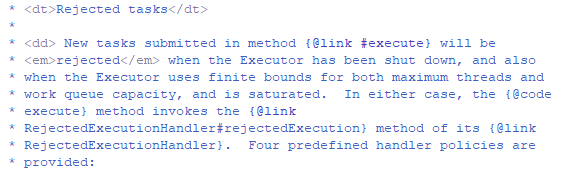


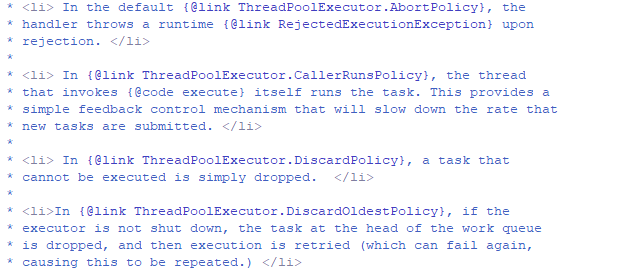
线程池中超过corePoolSize的线程将会被结束，如果该线程正处于空闲状态并且存活时间已经超过了keepAliveTime。



这个指任务队列。任意的阻塞队列被用来保存提交的任务，任务队列的使用受线程池中线程数量的影响，主要有以下几方面。

1. 如果线程数少于corePoolSize，创建新的线程执行任务。
2. 线程数大于corePoolSize小于maximumPoolSize，将任务保存到任务队列中。
3. 任务队列已满并且线程数达到了maximumPoolSize，这时，提交的任务将交给拒绝策略处理，下面会将拒绝策略。





当线程池已关闭或者任务队列和线程数已经饱和，会调用rejectedExecution拒绝新任务。

有四种拒绝策略：

1. 默认的AbortPolicy，该处理会抛出RejectedExecutionException运行时异常。
2. CallerRunsPolicy，由调用线程执行该任务。
3. DiscardPolicy，丢弃该任务。
4. DiscardOldestPolicy，丢弃任务队列中头部的任务，即最早添加的任务。然后调用execute方法。
5. 用户可以自定义拒绝策略，只需要实现RejectedExecutionHandler接口即可。

# 属性

ThreadPoolExecutor有几个重要的属性需要说明一下。

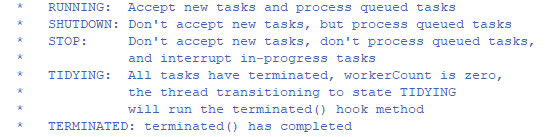
## 2.1ctl



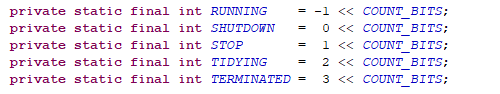
线程池的状态字段，这个字段打包了线程的两个属性，工作线程总数workerCount和线程池运行状态runState。workerCount用高3位表示，runState用低29位表示，也就是线程池的最大容量为229-1。



runState有一下几种状态：



1. RUNNING，表示线程池正在运行，能够接收新的任务并且能处理任务队列中的任务。
2. SHUTDOWN，不再接收新的任务但是能够处理任务队列中的任务。
3. STOP，不再接收新的任务，并且不再处理任务队列中的任务。
4. TIDYING，表示所有的任务已经结束，所有的工作线程也已经停止，线程池转为TIDYING状态。
5. TERMINATED，调用terminated方法之后转为该状态。



可以看到RUNNING用111表示，SHUTDOWN用000表示，STOP用001表示，TIDYING用010表示，TERMINATED用011表示。这里低29位都为0。

ctl默认值为111......0，即runState为RUNNING，workerCount为0。



runStateOf用来获取线程池的状态，workerCountOf用来获取工作线程的数量。

## 2.2workQueue



任务队列，用来储存提交的任务，工作线程从中取出任务来执行。

## 2.3workers



用来存放工作线程的集合。

## 2.4largestPoolSize



线程池线程最多的数量。

## 2.5completedTaskCount



完成任务的数量。

## 2.6keepAliveTime



线程的超时时间，前面已经提到过。

## 2.7corePoolSize



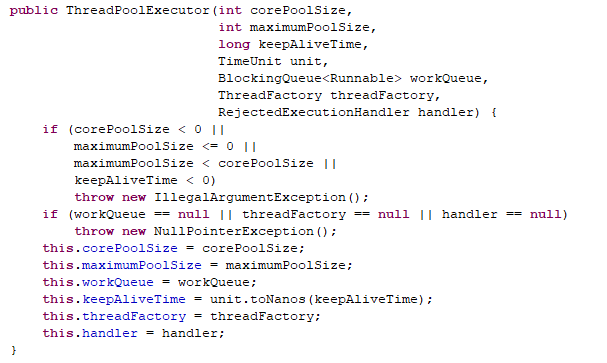
核心线程的数量，核心线程会一直保存存活。

## 2.8maximumPoolSize



最大线程的数量，不会超过229-1。

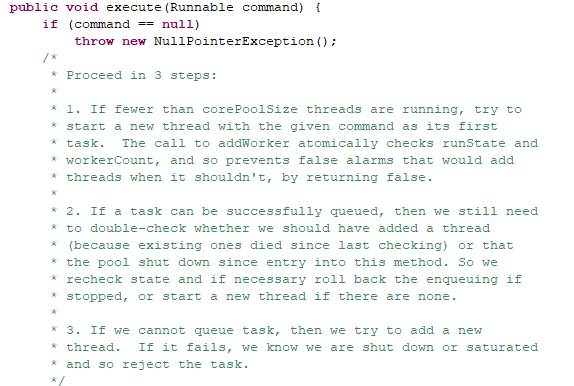
# 构造器

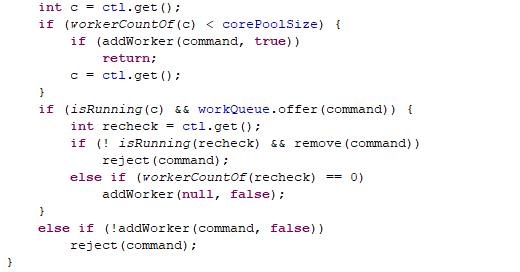


其他的构造器都调用了这个构造器来初始化。构造器中初始化了corePoolSize，maximumPoolSize等属性。

# 方法

## 4.1execute

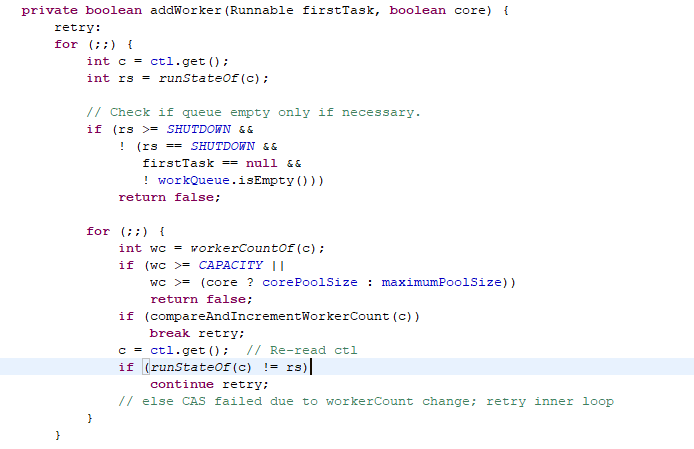


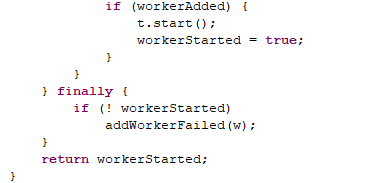
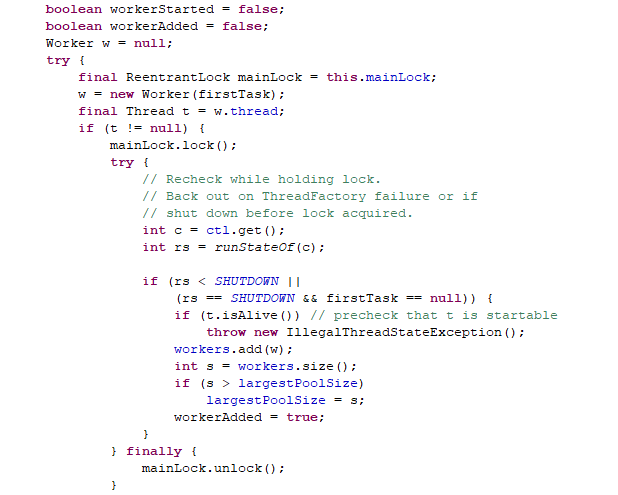


处理的过程分为三步，前面已经讲述过了。

在第二步成功添加到任务队列中，会再一次检查线程池的状态，如果这时线程池被停止（调用了shutdown或者shutdownNow），添加到任务队列中的任务会被移除，并使用拒绝策略处理该任务。否则判断线程池中线程数是否为0，有可能之前存活的线程超时死亡了，那么创建一个非核心线程。  
 如果任务队列已满，那么创建非核心线程，如果创建失败，说明线程数超过了maximumPoolSize，则拒绝任务。

## 4.2addWorder





先来看一次addWorker方法的两个参数。

firstTask，工作线程执行的第一个任务，当线程池线程数少于corePoolSize或者线程数超过corePoolSize少于maximumPoolSize并且任务队列已满时，execute方法会创建工作线程并将提交的任务作为工作线程执行的第一个任务传入addWorker中。

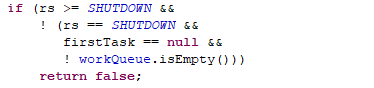
core，boolean类型，true表示使用corePoolSize作为线程数量的上届，false表示使用maximumPoolSize作为线程数量的上届。

先看一个该方法的前半部分，for循环中的操作。这里用了两层循环来更新workerCount的值，外层循环判断线程池的状态，内层循环判断线程池的线程数量。具体来看一下。

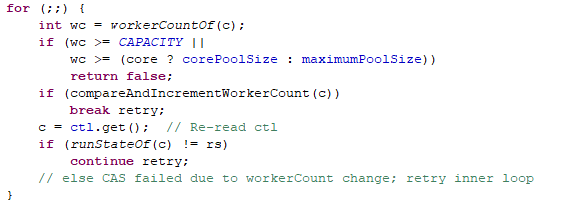
（1）首先获取线程池的运行状态rs，



1. 如果线程池处于running状态或者线程池处于shutdown并且firstTask为null并且任务队列为空，那么继续执行，否则返回false。



1. 内层循环不断尝试设置workerCount的值，直至成功或者线程数超出上界返回false为止。首先获取了线程池中线程的数量wc，如果线程数量超出容量或者上届直接返回false，否则使用CAS更新workerCount的值，更新成功跳出外层循环，更新失败重新比较线程池的状态是否跟rs一样，一样继续内层循环（这里CAS更新失败的原因是其他线程更新了workerCount，所以这里继续内层循环重新获取workerCount值），不一样继续外层循环。

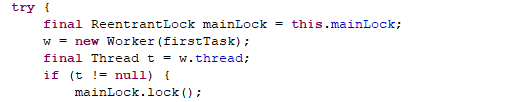


接下来的部分就是创建线程并启动线程的过程。

1. 记录线程启动和线程创建的状态workerStarted和workerAdded，并初始化为false。



1. 获取线程池的可重入锁，创建worker。如果执行worker的线程t不为null继续加锁继续执行，否则返回false。



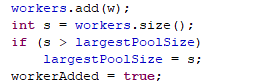
1. 再次检查线程池的的运行状态rs，如果线程池处于running状态或者shutdown状态并且firstTsk为null，继续执行，否则返回false。



1. 判断执行worker的线程是否启动过。如果已经启动，抛出IllegalThreadStateException异常。



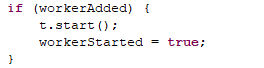
1. 将worker添加到工作线程集合中，并更新largestPoolSize。设置workerAdded等于true，表示工作线程已创建完成并添加到工作线程集合中完成。



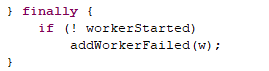
1. 释放锁。

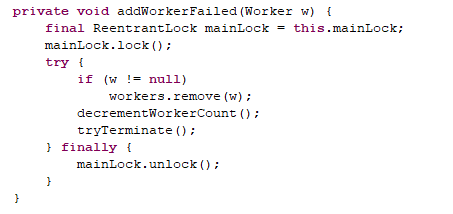


1. 启动工作线程。线程启动成功设置workerStarted等于true。

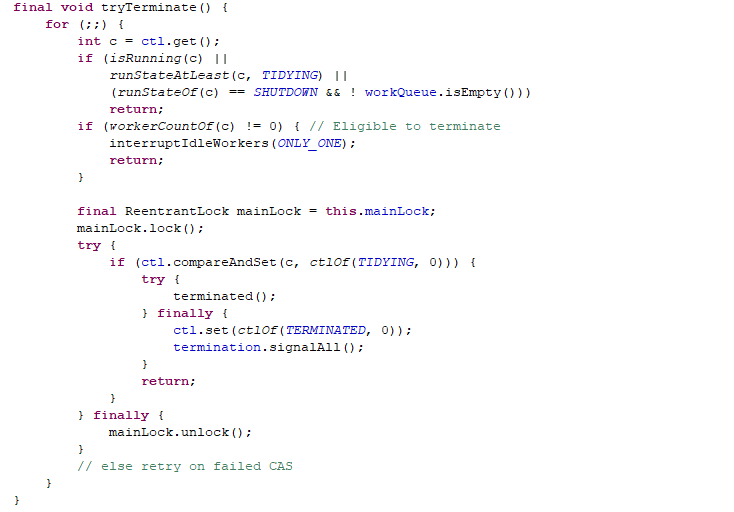


1. 如果w没有正常启动，移除w并减少workerCount。然后调用tryTerminate尝试结束线程池。





## 4.3tryTerminate



tryTerminate方法只会尝试中断空闲的线程。空闲线程指的是阻塞等待获取任务的线程。

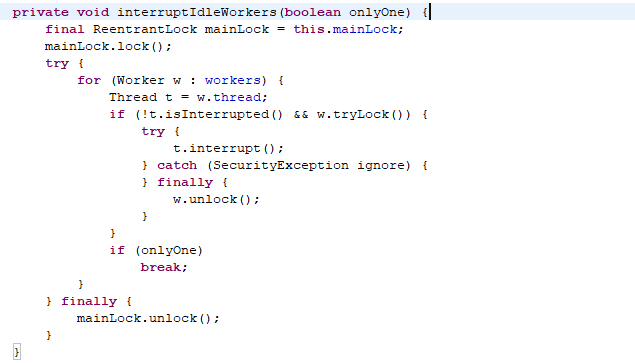
下面三中情况该方法会直接退出。

1. 线程池正处于运行状态。
2. 线程池处于TIDYING或者TERMINATED状态，说明线程池已经终止过了。
3. 如果线程池处于SHUTDOWN状态并且任务队列不为空。

三种情况都不满足，说明线程池正处于SHUTDOWN或者STOP状态并且任务队列为空，这时，如果工作线程数量不为0，会尝试中断一个空闲线程。如果工作线程数为0，加锁并设置线程池状态，然后唤醒等待线程池终止完成的线程。

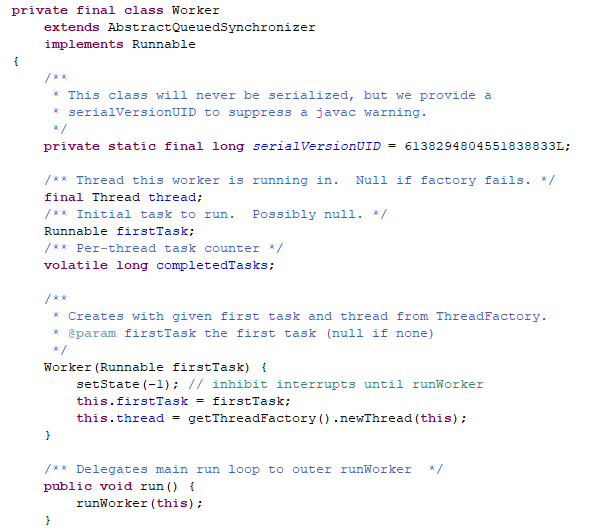
terminated是一个空实现。

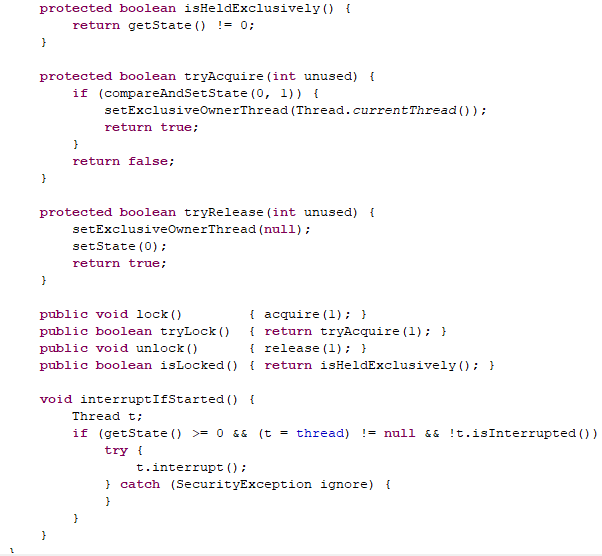
## 4.4interruptIdleWorkers



加锁并中断工作线程。如果线程已经中断过或者获取锁失败，那么不会再中断该线程。这里使用tryLock方法获取锁，如果获取锁失败会立即返回false，并且Worker实现了一个独占锁，这样做防止在Worker执行任务的时候被中断。下面会看到Worker的实现。

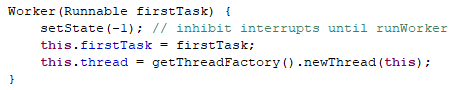
## 4.5worker内部类





Worker实现了一个独占锁。

Thread属性表示执行该工作线程的线程。firstTask工作线程执行的第一个任务。completedTasks表示完成的任务总数。

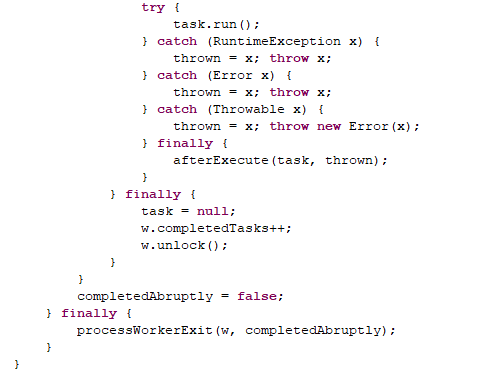
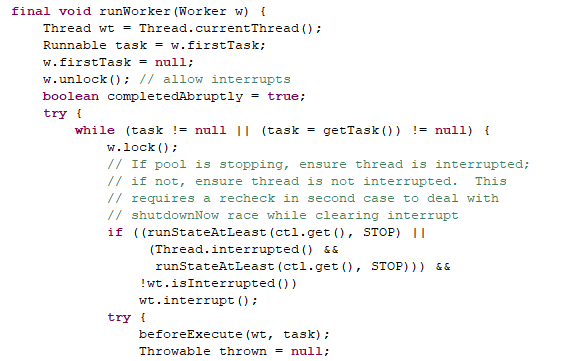


该构造器初始化了firstTask与thread。



调用了runWorker方法执行任务。

下面主要看一下runWorker方法。



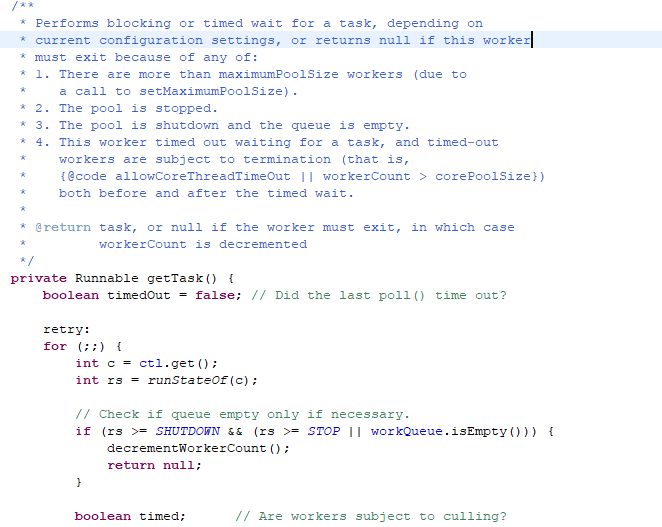
（1）该方法在获取到任务的时，首先加锁，目的是防止在执行任务期间被中断。接下来判断是否要中断线程。

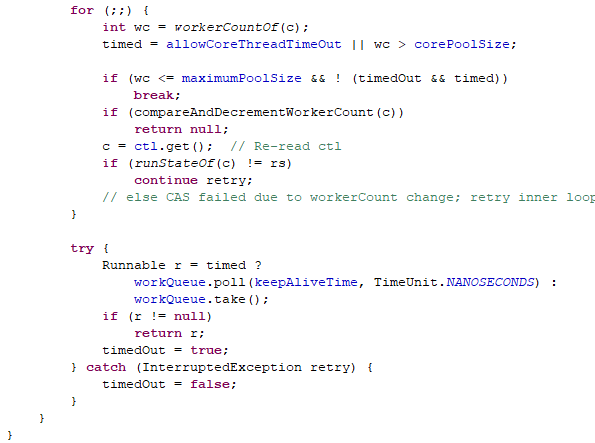
（2）如果>=STOP，中断该工作线程。否则检查线程是否已经中断并重置中断标志，如果线程已经中断那么在重新检查线程池的状态，如果>=stop并且线程未中断，中断该线程。

（3）执行任务。beforeExecute与afterExecute都是空实现，用来自定义线程池时在任务执行之前和之后做一下操作。

（4）当getTask获取null或者出现异常调用processWorkerExit方法处理线程退出。

## 4.6getTask



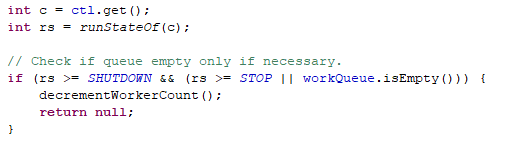


该方法会一直循环获取task，直到满足以下几种情况返回null。

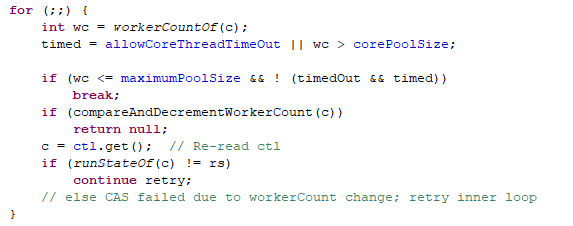
1. 超出maximumPoolSize数量的线程。出现这种情况可能是调用了两次setMaximumPoolSize方法并且第二次设置的比第一次小。
2. 线程池STOP。
3. 线程池SHUTDOWN并且任务队列为空。
4. 设置了allowCoreThreadTimeOut或者超出corePoolSize的线程超时获取task会返回null。

接下来看一下该方法的具体执行流程。

1. 首先检查线程池的状态，如果线程>=SHUTDOWN并且任务队列为空或者线程池>=STOP，减少工作线程数量并返回null。



1. 检查线程池内线程数，只有当线程数少于maximumPoolSize并且满足以下条件才会获取任务。
2. 设置了allowCoreThreadTimeOut，线程未超时。
3. 未设置allowCoreThreadTimeOut，线程未超时。
4. 未设置allowCoreThreadTimeOut，线程已超时，但是当前线程为核心线程。

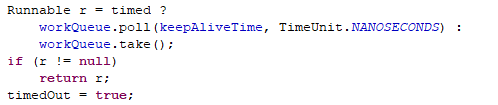


1. poll超时获取，当等待keepAliveTime时间仍然未获取到任务，那么返回null。

take如果任务队列为空，会一直阻塞，直到获取到任务为止。

timed为true的情况有以下几种：

1. 设置了allowCoreThreadTimeOut，
2. 未设置allowCoreThreadTimeOut，当前线程属于非核心线程。

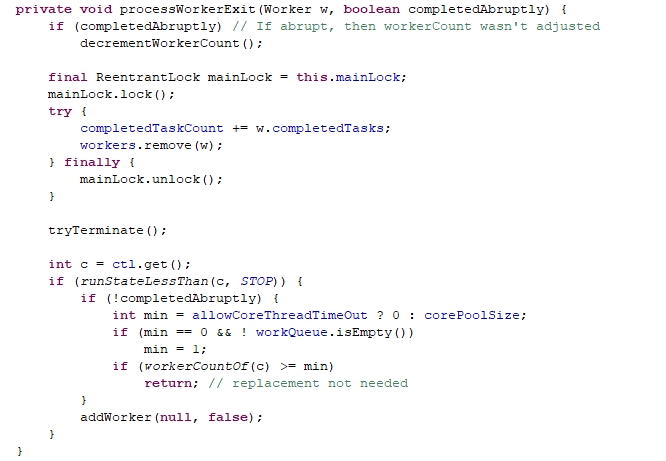


1. 如果出现中断异常， 那么重置timeOut。



## 4.7processWorkerExit

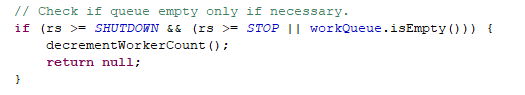
在执行任务期间出现异常或者getTask返回null将调用processWorkerExit处理任务退出。



completedAbruptly：true表示因为异常导致worker线程死亡。

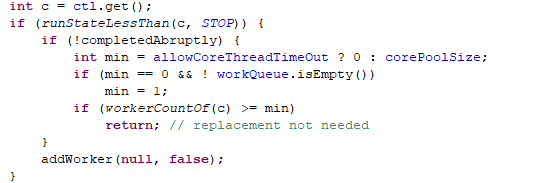


（1）这里只在异常出现的情况才减少工作线程数量，因为工作线程正常结束是getTask返回null时，在getTask方法中，在返回null操作前都确保减少了工作线程数。

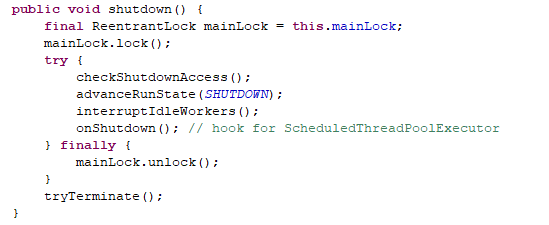




1. 之后就是增加任务完成数completedTaskCount，从工作队列中移除工作线程。然后调用tryTerminate尝试结束线程池。
2. 然后判断线程池状态，如果处于RUNNING或者SHUTDOWN状态，再判断工作线程结束原因。如果工作线程由于异常结束，那么重新添加一个工作线程代替该线程工作。如果工作线程正常结束，线程池中少于corePoolSize线程正在运行或者任务队列不为空并且没有工作线程在工作，那么也重新添加一个工作线程。



## 4.8shutDown



shutDown只中断空闲的线程。

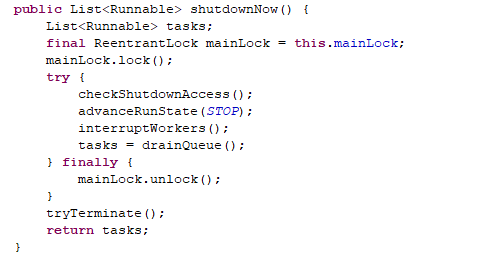
首先检查调用者是否具备中断线程的权限，然后设置线程池状态为SHUTDOWN，然后中断空闲线程。onShutDown是一个钩子函数。



false表示中断所有的空闲线程。

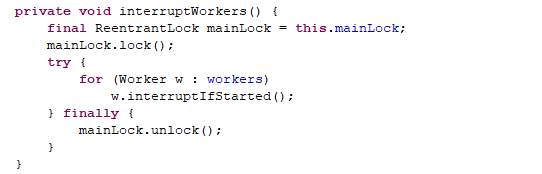
## 4.9shutDownNow

shutDownNow会中断所有线程并返回等待处理的任务列表。不管线程是否处于空闲状态还是正在处理任务都会被中断。



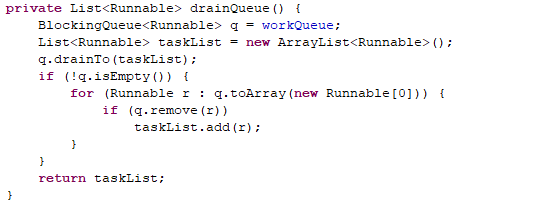
这里的流程跟shutDown差不多，只是这里调用了interruptWorkers方法和drainQueue方法。

## 4.10interruptWorkers



设置所有线程的中断标志位。

## 4.11drainQueue

 这里将任务队列中的线程都拷贝到了taskList中，之后有重新检查了任务队列是否为空。