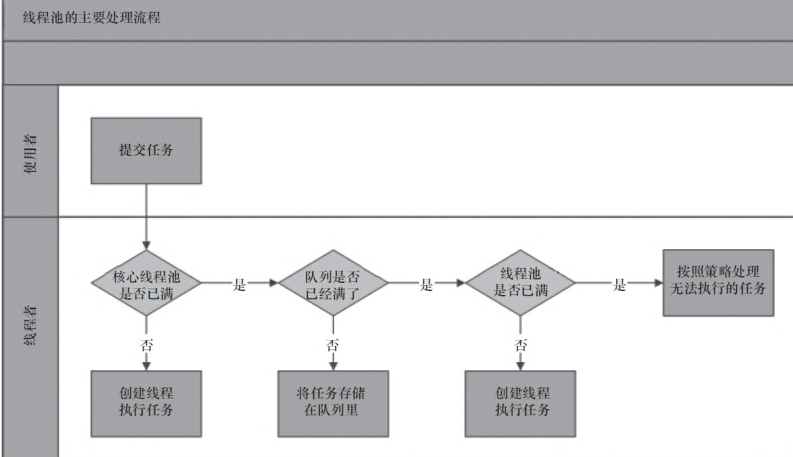
第一：**降低资源消耗**。通过重复利用**已创建的线程降低线程创建和销毁造成的消耗**。

第二：**提高响应速度**。当任务到达时，**任务可以不需要等到线程创建就能立即执行**。

第三：**提高线程的可管理性**。线程是稀缺资源，如果无限制地创建，不仅会消耗系统资源，

还会降低系统的稳定性，使用线程池可以进行**统一分配、调优和监控**。但是，要做到合理利用线程池，必须对其实现原理了如指掌。

1.线程池的实现原理



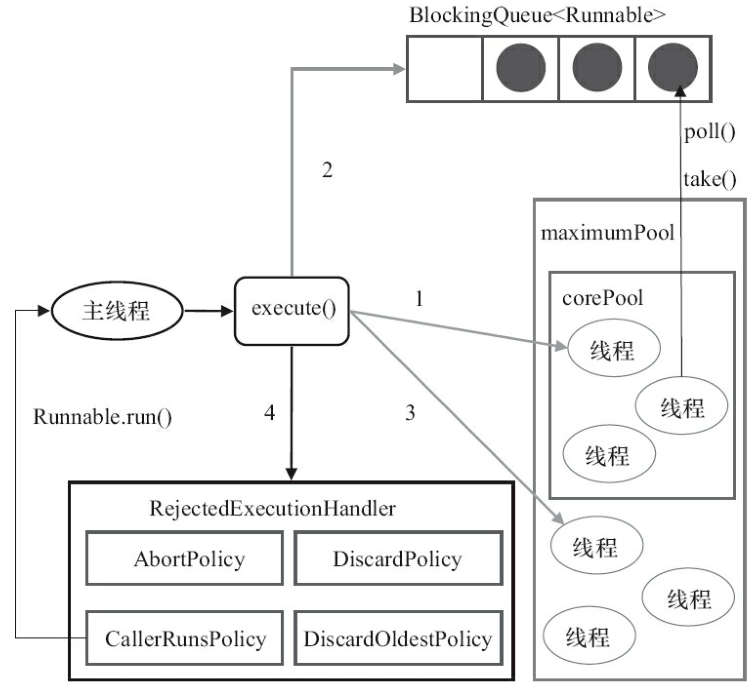
从图中可以看出，当提交一个新任务到线程池时，线程池的处理流程如下。

1）线程池判断**核心线程池**里的线程**是否都在执行任务**。如果不是，则创建一个**新的工作线程来执行任务**。如果核心线程池里的线程都在执行任务，则进入下个流程。

2）线程池判断**工作队列是否已经满**。如果工作队列没有满，则将新提交的任务存储在这个**工作队列**里。如果工作队列满了，则进入下个流程。

3）线程池判断线程池的线程是否**都处于工作状态**。如果没有，则创建一个**新的工作线程**来执行任务。如果**已经满了**，则交给**饱和策略**来处理这个任务。

Execute方法执行过程



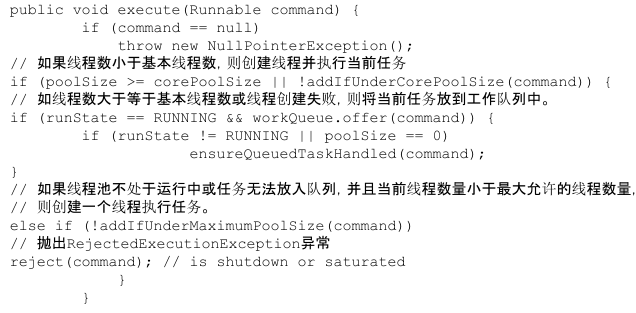
①如果当前运行线程少于核心线程的容量，就创建新线程执行任务（获取全局锁）

②如果运行线程等于或多于核心线程容量，就将任务加入阻塞队列

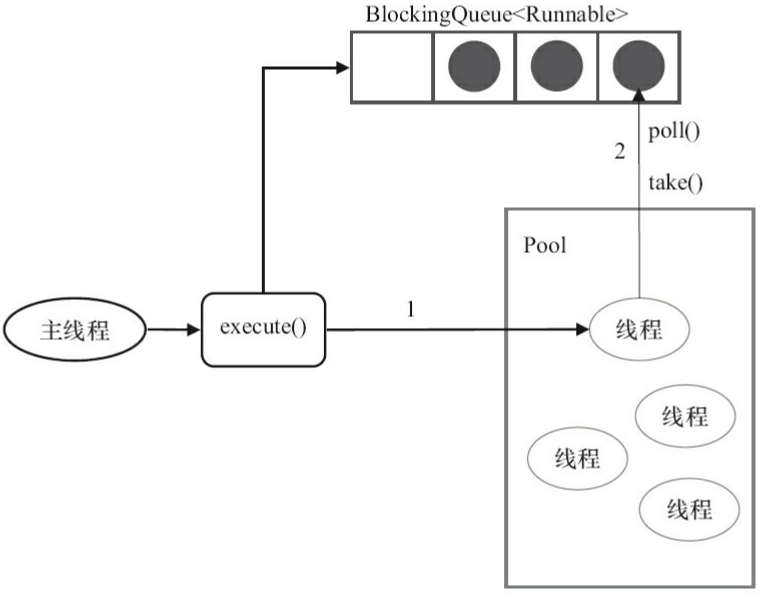
③如果阻塞队列已满，创建新线程处理任务

④如果超过最大线程数maximumPoolSize，任务将被拒绝，调用RejectedExecutionHandler.rejectedExecution()方法。

执行execute()方法时，尽可能避免获取全局锁，如果当前运行线程数大于等于corePoolSize，几乎所有的execute方法都是执行步骤2，不需要获取全局锁



工作线程：线程池创建线程时，会将线程封装成工作线程worker，**worker执行完任务**后，还会**循环获取工作队列的任务**来执行。



①execute方法**创建一个线程时**，会让**这个线程执行当前任务**

②这个线程执行**完上图1的任务**后，会反复从**BlockingQueue获取任务执行**

**2.线程池的使用**

**2.1线程池的创建**

可以通过ThreadPoolExecutor来创建一个线程池。

new ThreadPoolExecutor(**corePoolSize**, **maximumPoolSize**, **keepAliveTime**, **milliseconds**, **runnableTaskQueue**, **handler**);

①corePoolSize（核心线程池的基本大小）：当提交一个任务到线程池时，线程池会**创建一个线程来执行任务**，即使其他空闲的基本线程**能够执行新任务也会创建线程**，等到需要执行的任务数大于核心线程池基本大小时就不再创建。

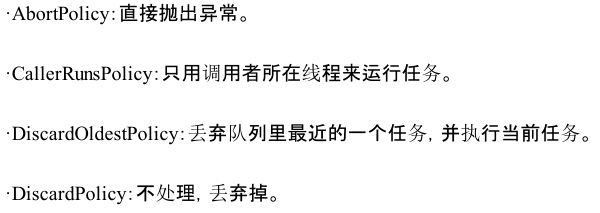
②runnableTaskQueue（任务队列）:用于保存**等待执行的任务**的**阻塞队列**

ArrayBlockingQueue、LinkedBlockingQueue、SynchronousQueue、PriorityBlockingQueue无界

③maximumPoolSize：线程池允许创建的**最大线程数**。如果**队列满了**，并且**已创建的线程数小于最大线程数**，则线程池会再创建新的线程执行任务。无界队列没有效果

④ThreadFactory：用于设置创建线程的工厂，可以通过线程工厂给每个创建出来的线程设置更有意义的名字。

⑤RejectedExecutionHandler（饱和策略）：当**队列和线程池**都满了，说明线程池处于**饱和**状态，那么必须采取一种策略**处理提交**的新任务。这个策略默认情况下是**AbortPolicy**，表示无法处理新任务时抛出异常。

**keepAliveTime**（线程活动保持时间）：线程池的工作线程**空闲**后，**保持存活的时间**。所以，如果任务很多，并且每个任务执行的**时间比较短**，可以**调大时间**，提高**线程的利用率**。

**TimeUnit**（线程活动保持时间的单位）：可选的单位有天（DAYS）、小时（HOURS）、分钟（MINUTES）、毫秒（MILLISECONDS）、微秒（MICROSECONDS，千分之一毫秒）和纳秒（NANOSECONDS，千分之一微秒）

**2.2 向线程池提交任务**

Execute和submit

execute()方法用于提交不需要返回值的任务，所以无法判断任务是否被线程池执行成功。

**threadsPool.execute(new Runnable() {**

**@Override**

**public void run() {**

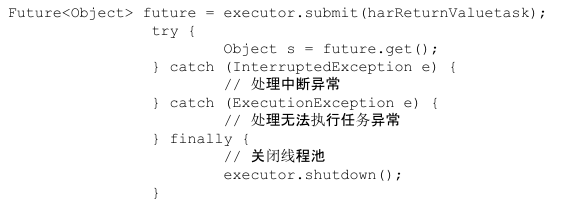
**// TODO Auto-generated method stub**

**}**

**});**

**submit()方法用于提交需要返回值的任务**

一个future类型的对象，通过这个future对象可以判断任务是否执行成功，并且可以通过future的get()方法来获取返回值，**get()方法会阻塞当前线程直到任务完成**，



**2.3线程池的监控**

TaskCount:线程池需要执行的任务数

completeTaskCount：已完成的任务数

largestPoolSize：线程池里**曾经创建过的最大线程数**量

getPoolSize：线程池的线程数量