链滴

ThreadPoolExecutor 使用详解

作者：hymn

原文链接：https://ld246.com/article/1592992562770

来源网站：链滴

为了保证每一个线程都成功，需要重写拒绝机制，阻塞提交线程
默认的提交线程用的是
offer方法，是非阻塞的提交线程，当队列满时，不会执行
put 方法，是阻塞的提交线程，保证每个线程都执行

源码如下

```
/**
     * Inserts the specified element into this queue, waiting if necessary
     * for space to become available.
     *
     * @param e the element to add
     * @throws InterruptedException if interrupted while waiting
     * @throws ClassCastException if the class of the specified element
     *         prevents it from being added to this queue
     * @throws NullPointerException if the specified element is null
     * @throws IllegalArgumentException if some property of the specified
     *         element prevents it from being added to this queue
     */
    void put(E e) throws InterruptedException;
```

参数名　作用

corePoolSize　核心线程池大小

maximumPoolSize　最大线程池大小

keepAliveTime　线程池中超过corePoolSize数目的空闲线程最大存活时间；可以allowCoreThreadTmeOut(true)使得核心线程有效时间

TimeUnit　keepAliveTime时间单位

workQueue　阻塞任务队列

threadFactory　新建线程工厂

RejectedExecutionHandler　当提交任务数超过maxmumPoolSize+workQueue之和时，任务会给RejectedExecutionHandler来处理

重点讲解：

其中比较容易让人误解的是：corePoolSize，maximumPoolSize，workQueue之间关系。

1.当线程池小于corePoolSize时，新提交任务将创建一个新线程执行任务，即使此时线程池中存在空线程。

2.当线程池达到corePoolSize时，新提交任务将被放入workQueue中，等待线程池中任务调度执行

3.当workQueue已满，且maximumPoolSize>corePoolSize时，新提交任务会创建新线程执行任务

4.当提交任务数超过maximumPoolSize时，新提交任务由RejectedExecutionHandler处理

5.当线程池中超过corePoolSize线程，空闲时间达到keepAliveTime时，关闭空闲线程

6.当设置allowCoreThreadTimeOut(true)时，线程池中corePoolSize线程空闲时间达到keepAliveTie也将关闭

总结：

1. 用ThreadPoolExecutor自定义线程池，看线程是的用途，如果任务量不大，可以用无界队列，如任务量非常大，要用有界队列，防止OOM

2. 如果任务量很大，还要求每个任务都处理成功，要对提交的任务进行阻塞提交，重写拒绝机制，改阻塞提交。保证不抛弃一个任务

3. 最大线程数一般设为2N+1最好，N是CPU核数

4. 核心线程数，看应用，如果是任务，一天跑一次，设置为0，合适，因为跑完就停掉了，如果是常线程池，看任务量，是保留一个核心还是几个核心线程数

5. 如果要获取任务执行结果，用CompletionService，但是注意，获取任务的结果的要重新开一个线获取，如果在主线程获取，就要等任务都提交后才获取，就会阻塞大量任务结果，队列过大OOM，以最好异步开个线程获取结果

可选择的阻塞队列BlockingQueue详解

在重复一下新任务进入时线程池的执行策略：

如果运行的线程少于corePoolSize，则 Executor始终首选添加新的线程，而不进行排队。（如果当运行的线程小于corePoolSize，则任务根本不会存入queue中，而是直接运行）

如果运行的线程大于等于 corePoolSize，则 Executor始终首选将请求加入队列，而不添加新的线。

如果无法将请求加入队列，则创建新的线程，除非创建此线程超出 maximumPoolSize，在这种情下，任务将被拒绝。

主要有3种类型的BlockingQueue：

无界队列

队列大小无限制，常用的为无界的LinkedBlockingQueue，使用该队列做为阻塞队列时要尤其当心当任务耗时较长时可能会导致大量新任务在队列中堆积最终导致OOM。阅读代码发现，Executors.nwFixedThreadPool 采用就是 LinkedBlockingQueue，而楼主踩到的就是这个坑，当QPS很高，发数据很大，大量的任务被添加到这个无界LinkedBlockingQueue 中，导致cpu和内存飙升服务器挂。

有界队列

常用的有两类，一类是遵循FIFO原则的队列如ArrayBlockingQueue与有界的LinkedBlockingQueu，另一类是优先级队列如PriorityBlockingQueue。PriorityBlockingQueue中的优先级由任务的Comarator决定。

使用有界队列时队列大小需和线程池大小互相配合，线程池较小有界队列较大时可减少内存消耗，降cpu使用率和上下文切换，但是可能会限制系统吞吐量。

在我们的修复方案中，选择的就是这个类型的队列，虽然会有部分任务被丢失，但是我们线上是排序志搜集任务，所以对部分对丢失是可以容忍的。

同步移交队列

如果不希望任务在队列中等待而是希望将任务直接移交给工作线程，可使用SynchronousQueue作为待队列。SynchronousQueue不是一个真正的队列，而是一种线程之间移交的机制。要将一个元素放SynchronousQueue中，必须有另一个线程正在等待接收这个元素。只有在使用无界线程池或者有饱策略时才建议使用该队列。