链滴

# 高并发程序设计（5）——JDK 并发包（线程池）

作者：wanglei0622

原文链接：https://ld246.com/article/1488875982172

来源网站：链滴

<h2>线程池</h2>
<ul>
<li>线程本身也是要占用内存空间，大量的线程会抢占系统资源，如果处理不当会导致out of memor异常，即便没有，大量的线程回收也会给GC带来很大压力。</li>
<li>在实际生产环境中，线程的数量必须得到控制，线程池就是为此诞生的。</li>
<li>JDK提供一套Executor框架，帮助开发人员高效的控制线程，其本质就是线程池。</li>
</ul>
<h2>Executors</h2>
<ul>
<li>扮演线程池工厂的角色，通过Executors可以获得一个有特定功能的线程池。</li>
<li>newFixedThreadPool():返回固定数量的线程池，有新任务提交时，如果线程池有空闲线程，则即执行，否则暂存一个任务队列，等待空闲线程。</li>
<li>newSingleThreadPool():返回一个只有一个线程的线程池，如果有多余任务提交时，保存在一个务队列中，按先入先出的顺序执行队列中的任务。</li>
<li>newCachedThreadPool():返回一个可以根据实际情况调整的线程池，线程数量不确定，有空闲程就复用，没有就创建新的，所有线程在任务处理完后，返回线程池进行复用。</li>
<li>newSingleThreadScheduledExecutor():返回一个扩展了ExecutorService接口的对象，线程池小为1，可以在指定时间执行任务，如在某个固定的延时后执行，或周期性执行。</li>
<ul>
<li>如果任务的执行时间超过调度时间，ScheduledExecutorService 不会让任务堆叠出现，会等执结束再等待延时时间。</li>
<li>调度程序不保证任务无限期执行下去，在程序抛出异常的时候，后续所有执行都会被中断，因此做好异常的处理。</li>
</ul>
<li>newScheduledThreadPool():返回同上的对象，但是可以指定线程数量。</li>
</ul>
<h2>ThreadPoolExecutor</h2>
<ul>
<li>线程池工厂返回的线程池都是ThreadPoolExecutor类的封装。</li>
<li>线程池最重要的构造函数ThreadPoolExecutor(corePoolSize, maximumPoolSize, keepAliveTme, unit, workQueue, threadFactory, handler)</li>
<ul>
<li>corePoolSize:指定了线程池的线程数量。</li>
<li>maximumPoolSize:指定线程池中最大线程数量。</li>
<li>keepAliceTime:当线程池线程数量超过corePoolSize时，多余空闲线程的存活时间，空闲超过这时间就会被销毁。</li>
<li>unit:keepAliceTime的时间单位。</li>
<li>workQueue:任务队列，被提交但未被执行的任务。</li>
<ul>
<li>SynchronousQueue:直接提交队列，他是一个特殊的BlockingQueue，没有容量，每一个插入作，都要等待一个响应的删除操作，反之，每一个删除都要等待响应的插入。提交的任务并不会被正保存，总是将新任务提交给线程执行，如果没有空闲线程，创建新的，如果线程数达到最大值，执行绝策略。</li>
<li>ArrayBlockingQueue:有界任务队列，线程池实际线程数小于corePoolSize，则会优先创建新线，否则加入任务队列，若队列满了，在总线程数不大于maximumPoolSize前提下，创建新线程执行务，若大于max则执行拒绝。</li>
<li>LinkedBlockingQueue:无界队列，线程数达到corePoolSize后就不再新建，新提交的任务进入列等待，若处理的速度和新增的速度差异太大，无界队列会保持快速增长，直到内存溢出。</li>
<li>PriorityBlockingQueue:总是确保高优先级任务先执行，之前的都是先进先出算法处理任务。</l>
</ul>
<li>threadFactory：线程工程，一般使用默认的就可以。</li>
<li>handler:拒绝策略。来不及处理时，拒绝后如何处理。</li>
<ul>

<li>JDK有提供几个拒绝策略，都是通过实现RejectedExecutionHandler接口，也可以通过这个接口定义拒绝时的策略。</li>
</ul>
</ul>
<li>使用自定义线程池时，要选择合适的并发队列作为任务的缓冲。</li>
<li>扩展线程池，提供beforeexecute()、afterExecute()、terminated()三个空的实现。</li>
<li>要合理设置线程池线程的数量。一般设置cpu的个数就行。</li>
<li>线程池可能会吃掉程序抛出的异常，最简单的方法是放弃submit(),使用execute(),或打印submit()的返回值，但是都只能得到部分的异常堆栈，只能知道在哪里抛出异常，但是不知道任在哪里调用的。这就需要扩展线程池，加入提交任务线程的堆栈信息。</li>
</ul>
 
 