Java中容易踩到的“坑”系列之线程池篇

在有了java.util.concurrent包后，通常都会采用Executors或ThreadPoolExecutor来创建线程池，但这两个类都挺容易用错，如果不仔细阅读它的API或源码的话，很容易就踩进坑里。

通过new ThreadPoolExecutor可创建相应配置的线程池，但这个接口挺容易被误用，也许会导致行为和你想要的不太一样，之前在微博上我出了下面这道题。

有以下两个线程池：

```
ThreadPoolExecutor aPool =
new ThreadPoolExecutor(10,20,5,TimeUnit.MINUTES,new
SynchronousQueue<Runnable>());  同步队列
ThreadPoolExecutor bPool =
new ThreadPoolExecutor(10,20,5,TimeUnit.MINUTES,new
ArrayBlockingQueue<Runnable>(10));  基于数组的有界队列
```

目前已有10个请求正在处理中，请问当第11个请求进来时aPool和bPool分别会如何处理？

从评论上来看，答错的人还是有一些的，这道题的正确答案是：
当第11个请求进来时，aPool会启动线程来立刻处理，bPool会放入BlockingQueue，等待前面的线程处理完才会被处理。 无容量的同步队列 有容量的有界队列

任务入队列策略、线程创建策略

原因是ThreadPoolExecutor的实现机制为当目前运行的线程数 >= corePoolSize（也就是上面的10）时，ThreadPoolExecutor会将请求放进Queue中（例如上面的aPool是SynchronousQueue），如放失败且目前的线程数 < maxPoolSize（也就是上面的20），那么就直接启动新线程来处理，如目前运行的线程数等于maxPoolSize，则执行对应的策略，上面的aPool和bPool使用的都是默认的拒绝并跑出异常的策略；如放成功则进行再次的检查（例如线程池是否被shutdown、运行的线程数是否小于corePoolSize），然后返回。

拒绝执行策略

SynchronousQueue是一个很特殊的Queue，当往这个Queue放东西时，必须有另外一个线程在从这个Queue里拿，如没有，则直接失败，在上面的场景中，当第11个请求进来时，往这个Queue中放就将失败，而这个时候运行的线程数又小于maxPoolSize，因此将启动新线程进行处理。
而bPool采用的是ArrayBlockingQueue，put将成功，可见在bPool的情况下，想要运行的线程数增加到11个，必须是10个线程已经在处理中，并且ArrayBlockingQueue已经排队了10个请求，那么这个时候如果再有第21个请求进来，才会启动第11个线程进行处理，刚用这个API时，很容易会认为只有当线程数已经达到了maxPoolSize后才会往

坑

Queue里放。

Executors是ThreadPoolExecutor的包装，基于它可以更简单的去创建线程池，但在使用它创建线程池也要特别的注意，下面就说说这个类中常用的几个方法创建的线程池的一些特征：

1. newCachedThreadPool
这种方式创建出来的线程池corePoolSize为0，maxPoolSize为Integer.MAX_VALUE，BlockingQueue为SynchronousQueue，因此这个线程池的行为为如线程均在运行中，新任务需要执行，则直接启动线程，直到运行的线程数达到Integer.MAX_VALUE。
这种线程池在某些情况下可能会创建非常大量的线程，鉴于这个原因，不推荐使用。
2. newFixedThreadPool
这种方式创建出来的线程池corePoolSize和maxPoolSize均为指定的大小，BlockingQueue为LinkedBlockingQueue，因此这个线程池的行为为如同时处理的请求小于指定的大小，那么就创建新的线程来处理，如达到了指定的大小，则放入Queue中。
这种线程池在某些情况下可能会出现Queue中堆积了很多的任务，导致内存被耗光的现象，因此也不推荐使用，如有类似需求，建议使用限定大小的ArrayBlockingQueue等。

最后无论是用哪种方式创建的线程池，最好都增加下名字，可通过实现ThreadPoolFactory接口来做到（例如这个），否则的话当线程池的线程处于等待任务处理时，如这个时候jstack，会看到一堆类似pool-x-thread-y的线程名字，类似如下：

NamedThreadFactory

[code]
“pool-19-thread-1” prio=10 tid=0x00007fe6676f8000 nid=0x17df7 waiting on condition [0x0000000072bee000]
java.lang.Thread.State: TIMED_WAITING (parking)
at sun.misc.Unsafe.park(Native Method)
– parking to wait for <0x00000000b0412c98> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:196)
at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:2025)
at java.util.concurrent.DelayQueue.take(DelayQueue.java:164)
at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:609)
at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:602)
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:947)

at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:907)
at java.lang.Thread.run(Thread.java:662)
[/code]
如这个时候问题出在这个线程池创建的线程过多时，那排查起来就比较麻烦些（那就只能挂起btrace来跟踪）。

2013年3月28日 bluedavy Java
花名：毕玄，真名：林昊

《Java中容易踩到的“坑”系列之线程池篇》有6个想法

zxpan
2013年3月29日 02:11

貌似上述都不推荐，有木有最佳实践可以推荐下呢？谢谢。

回复

bluedavy
2013年3月29日 02:25

@zxpan
最佳模式就是直接通过new ThreadPoolExecutor来创建。

```
new ThreadPoolExecutor(10, 20,
        5L, TimeUnit.MINUTES,
        new SynchronousQueue<>(),
        new NamedThreadFactory("dubbo", true));
```

回复

```
new ThreadPoolExecutor(10, 20,
        5L, TimeUnit.MINUTES,
        new SynchronousQueue<>(),
        new ThreadFactoryBuilder()
                .setDaemon(true)
                .setNameFormat("guava-thread-%d")
                .build());
```

llandyl
2013年8月9日 16:25

不错，学习了。

回复

lanhuai

2013年11月18日 16:45

带限定大小的LinkedBlockingQueue也可以吧

回复

zhouwei

2017年1月1日 14:22

http://www.grepcode.com/file/repo1.maven.org/maven2/com.facebook.jcommon/concurrency/0.1.20/com/facebook/concurrency/NamedThreadFactory.java#NamedThreadFactory
您好！我想请教您下，这个
public ~~synchronized~~ Thread newThread(Runnable r)
方法为什么要加synchronized同步呢？
我感觉没有必要加同步啊，还请解惑，非常感谢！

回复

bluedavy

2017年1月11日 15:40

确实没必要…不知道为什么要加…

回复