Threadpool

目录

[总结 1](#_Toc7274)

[自动并行 1](#_Toc28262)

[总结 2](#_Toc5609)

[30thrd 8k/min 3](#_Toc32042)

[8thrd 7k/30s 4](#_Toc19323)

[4thrd 6k/30s 4](#_Toc15351)

别急，其实我们可以分组记忆：

第1组（线程数量相关的）：

1. corePoolSize: 核心线程数。即使线程池中没有任务，这些线程也不会被销毁，因为创建和销毁线程是需要消耗CPU资源的
2. maximumPoolSize: 线程池中允许创建的最大线程数

# 总结

1. 一般我们使用多线程时会用ExecuterService，构造用new ThreadPoolExecutor()，一般不使用Executors提供了构造线线程池方法，避免出现OOM；
2. 线程池相对于线程组(本文没提到)更好管理；
3. 在JDK7之后可以用ForkJoinPool，相对于ExecuterService执行效率更快。

首先，fork/join方法具备暂停执行中的任务的作用，这使得所有的任务只需要几个线程就能运行。如果以上代码中传入一个200万元素的数组，会产生多达400万个任务，但是要运行它们，却只需要几个线程甚至是一个。如果使用ThreadPoolExecutor运行类似任务，则需要400万个线程，因为每个线程都必须等待其子任务完成，而这些子任务只有在线程池中有额外线程可用的时候才可以完成。所以fork/join的暂停可以让我们使用原本不能使用的算法，这是性能上的一大优势。

### 自动并行

Java具备自动并行化某些类型代码的能力，而这个能力依赖于ForkJoinPool。JVM会为此创建一个通用的fork-join线程池，它是ForkJoinPool类的一个静态对象，大小默认为机器上的可用处理器数量。

在Arrays类的方法当中，这种自动并行很多见，比如使用快速排序算法对数组进行排序，对数组中的每个元素进行操作的方法等。在流处理当中也有用到，借此可以对集合中的每个元素进行操作（串行或者并行）。

1. ForkJoinPool 最适合的是****计算密集型的任务****，如果存在 I/O，线程间同步，sleep() 等会造成线程长时间阻塞的情况时，最好配合使用 ManagedBlocker。

* 方案二：ExecutorService多线程方式实现  
  在 Java 1.5 引入 ExecutorService 之后，****基本上已经不推荐直接创建 Thread 对象，而是统一使用 ExecutorService****。毕竟从接口的易用程度上来说 ExecutorService 就远胜于原始的 Thread，更不用提 java.util.concurrent 提供的数种线程池，Future 类，Lock 类等各种便利工具。

部集中在了 compute() 这个函数里，仅用了14行就实现了完整的计算过程。特别是，在这段代码里没有显式地“把任务分配给线程”，只是分解了任务，****而把具体的任务到线程的映射交给了 ForkJoinPool 来完成。****

* 方案四：采用并行流（JDK8以后的推荐做法）

耗时效率方面解释：Fork/Join 并行流等当计算的数字非常大的时候，优势才能体现出来。也就是说，如果你的计算比较小，或者不是CPU密集型的任务，不太建议使用并行处理

所以对于大多数情况， Executors::newFixedThreadPool(int nThreads) 是当我们想要使用线程池时首先考虑的选择对象。对于计算密集型的任务它通常能提供近于最优的吞吐量，对于 IO 密集型的任务也不会使任何问题变得更糟。至少如果在我们使用这些 Executor 遇到问题并进行性能调优时，不会毫无头绪。

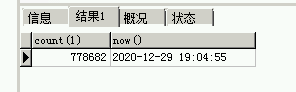
### **总结**

本篇文章我看了 Executors 类提供给我们的选择，以及何时使用各个 Executor 的策略。对于 CPU 密集型的任务，**newFixedThreadPool** 可以适用大多数场景，除非明确知道另外一个选择更好。但是，对于 IO 密集型的任务，并不简单。可以通过将 IO 调用包装到 **ManagedBlocker** 里并使用 **ForkJoinPool** 来增强它内部的并行能力。

* **Thread**API的其他部分不仅很少使用，而且几乎完全不暴露给程序员。从Java 5开始，鼓励程序员通过**ExecutorService**s间接创建和启动线程，这样，**Thread**类中的混乱就不会造成很大的危害。新的Java开发人员不必接触其中的大多数，也不必接触过时的痕迹。因此，保留**Thread**API的教学成本很小。

## 30thrd 8k/min





## 8thrd 7k/30s





## 4thrd 6k/30s

