<https://mp.weixin.qq.com/s/F1GpUjSXtYitFs4Ijqig7g>

## 线程数，射多少更舒适？

线程数的设置的最主要的目的是为了充分并合理地使用 CPU 和内存等资源，从而最大限度地提高程序的性能，因此让我们一起去探索吧！

首先要考虑到 CPU 核心数，那么在 Java 中如何获取核心线程数？

可以使用 Runtime.getRuntime().availableProcessor() 方法来获取（可能不准确，作为参考）

在确认了核心数后，再去判断是 CPU 密集型任务还是 IO 密集型任务：

* CPU 密集型任务：比如像加解密，压缩、计算等一系列需要大量耗费 CPU 资源的任务，**大部分场景下都是纯 CPU 计算**。
* IO 密集型任务：比如像 MySQL 数据库、文件的读写、网络通信等任务，这类任务**不会特别消耗 CPU 资源，但是 IO 操作比较耗时，会占用比较多时间**。

在知道如何判断任务的类别后，让我们分两个场景进行讨论：

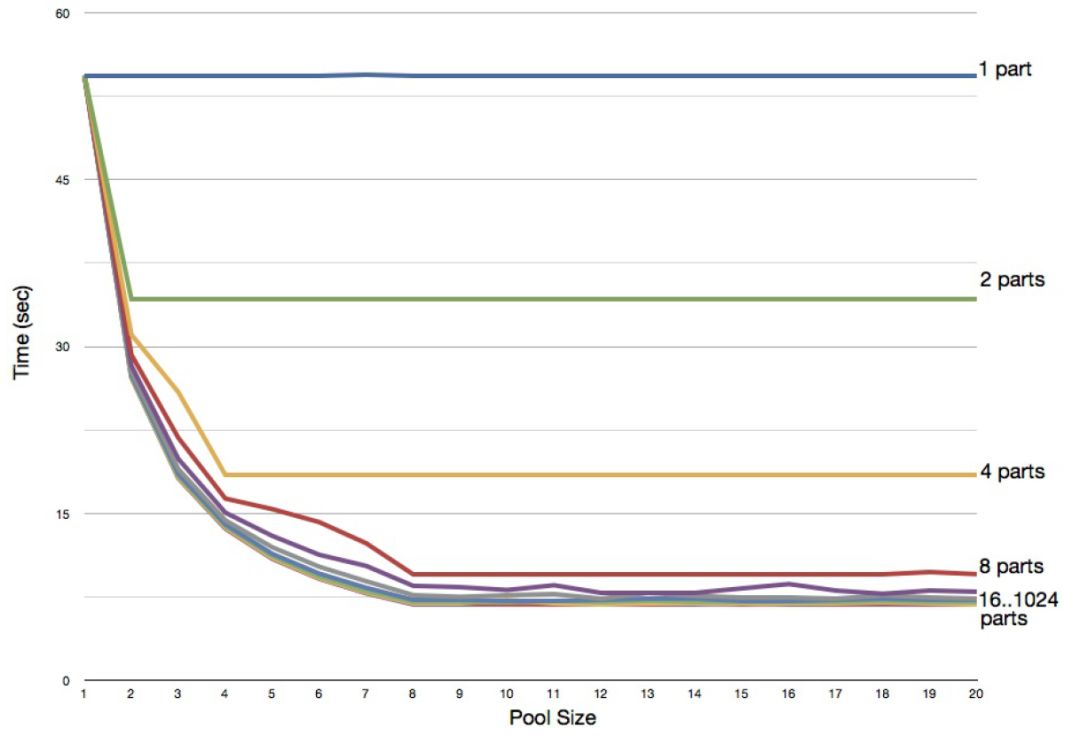
## **CPU 密集型任务**

对于 CPU 密集型计算，多线程本质上是提升多核 CPU 的利用率，所以对于一个 8 核的 CPU，每个核一个线程，理论上创建 8 个线程就可以了。

如果设置过多的线程数，实际上并不会起到很好的效果。此时假设我们设置的线程数量是 CPU 核心数的 2 倍，因为计算任务非常重，会占用大量的 CPU 资源，所以这时 CPU 的每个核心工作基本都是满负荷的，而我们又设置了过多的线程，每个线程都想去利用 CPU 资源来执行自己的任务，这就会造成不必要的上下文切换，此时线程数的增多并没有让性能提升，反而由于线程数量过多会导致性能下降。

因此，**对于 CPU 密集型的计算场景，理论上**线程的数量 = CPU 核数**就是最合适的**，不过通常把线程的数量设置为**CPU 核数 +1**，会实现最优的利用率。即使当密集型的线程由于偶尔的内存页失效或其他原因导致阻塞时，这个额外的线程也能确保 CPU 的时钟周期不会被浪费，从而保证 CPU 的利用率。

如下图就是在一个 8 核 CPU 的电脑上，通过修改线程数来测试对 CPU 密集型任务（素数计算）的性能影响。



可以看到线程数小于 8 时，性能是很差的，在线程数多于处理器核心数对性能的提升也很小，因此可以验证公式还是具有一定适用性的。

除此之外，我们最好还要同时考虑在同一台机器上还有哪些其他会占用过多 CPU 资源的程序在运行，然后对资源使用做整体的平衡。

## **IO 密集型任务**

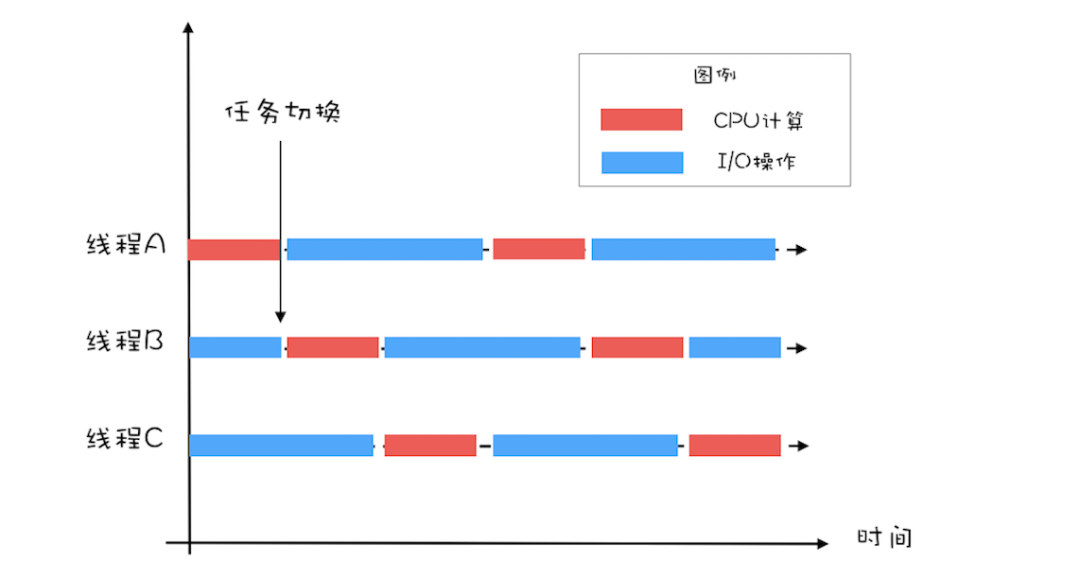
对于 IO 密集型任务最大线程数一般会大于 CPU 核心数很多倍，因为 IO 读写速度相比于 CPU 的速度而言是比较慢的，如果我们设置过少的线程数，就可能导致 CPU 资源的浪费。而如果我们设置更多的线程数，那么当一部分线程正在等待 IO 的时候，它们此时并不需要 CPU 来计算，那么另外的线程便可以利用 CPU 去执行其他的任务，互不影响，这样的话在任务队列中等待的任务就会减少，可以更好地利用资源。

对于 IO 密集型计算场景，最佳的线程数是与程序中 CPU 计算和 IO 操作的耗时比相关的，《Java并发编程实战》的作者 Brain Goetz 推荐的计算方法如下：

线程数 = CPU 核心数 \* (1 + IO 耗时/ CPU 耗时)

通过这个公式，我们可以计算出一个合理的线程数量，如果任务的平均等待时间长，线程数就随之增加，而如果平均工作时间长，也就是对于我们上面的 CPU 密集型任务，线程数就随之减少。可以采用 APM 工具统计到每个方法的耗时，便于计算 IO 耗时和 CPU 耗时。

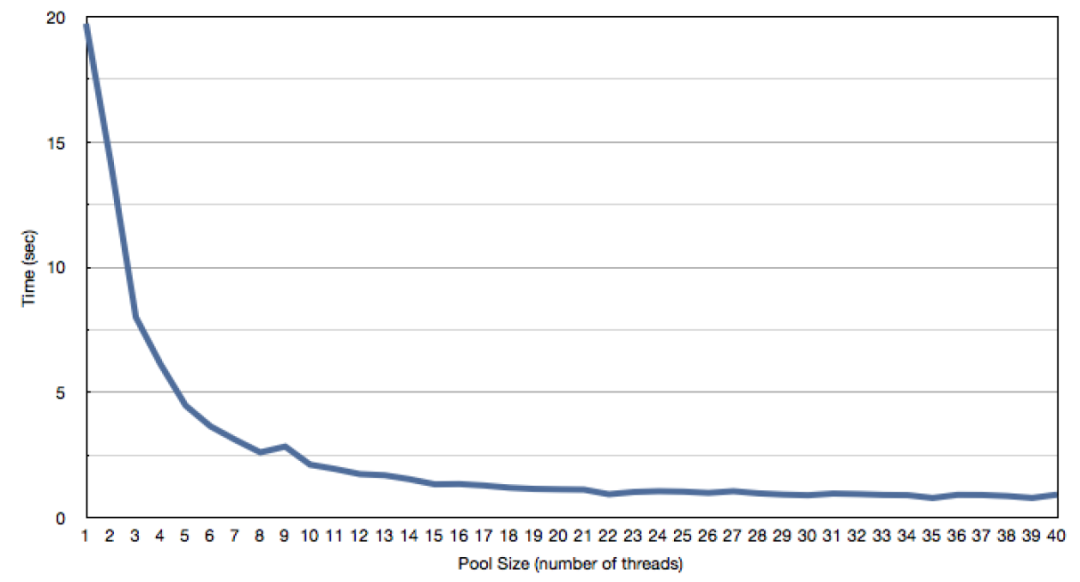
在这里引用Java并发编程实战中的图，方便大家更容易理解：



还有一派的计算方式是《Java虚拟机并发编程》中提出的：

线程数 = CPU 核心数 / (1 - 阻塞系数)

其中计算密集型阻塞系数为 0，IO 密集型阻塞系数接近 1，一般认为在 0.8 ~ 0.9 之间。比如 8 核 CPU，按照公式就是 2 / ( 1 - 0.9 ) = 20 个线程数



上图是 IO 密集型任务的一个测试，是在双核处理器上开不同的线程数（从 1 到 40）来测试对程序性能的影响，可以看到线程池数量达到 20 之后，曲线逐渐水平，说明开再多的线程对程序的性能提升也毫无帮助。

太少的线程数会使得程序整体性能降低，而过多的线程也会消耗内存等其他资源，所以如果想要更准确的话，可以进行压测，监控 JVM 的线程情况以及 CPU 的负载情况，根据实际情况衡量应该创建的线程数，合理并充分利用资源。

同时，有很多线程池的应用，比如 Tomcat、Redis、Jdbc 等，每个应用设置的线程数也是不同的，比如 Tomcat 为流量入口，那么线程数的设置可能就要比其他应用要大。

# **总结**

通过对线程数设置的探究，我们可以得知线程数的设置首先和 CPU 核心数有莫大关联，除此之外，我们需要根据任务类型的不同选择对应的策略，线程的平均工作时间所占比例越高，就需要越少的线程；线程的平均等待时间所占比例越高，就需要越多的线程；针对不同的程序，进行对应的实际测试就可以得到最合适的选择。