10 | Java线程（中）：创建多少线程才是合适的？

2019-03-21 王宝令

在Java领域，实现并发程序的主要手段就是多线程，使用多线程还是比较简单的，但是使用多少个线程却是个困难的问题。工作中，经常有人问，“各种线程池的线程数量调整成多少是合适的？”或者“Tomcat的线程数、Jdbc连接池的连接数是多少？”等等。那我们应该如何设置合适的线程数呢？

要解决这个问题，首先要分析以下两个问题：

1. 为什么要使用多线程？
2. 多线程的应用场景有哪些？

为什么要使用多线程？

使用多线程，本质上就是提升程序性能。不过此刻谈到的性能，可能在你脑海里还是比较笼统的，基本上就是快、快、快，这种无法度量的感性认识很不科学，所以在提升性能之前，首要问题是：如何度量性能。

度量性能的指标有很多，但是有两个指标是最核心的，它们就是延迟和吞吐量。**延迟**指的是发出请求到收到响应这个过程的时间；延迟越短，意味着程序执行得越快，性能也就越好。 **吞吐量**指的是在单位时间内能处理请求的数量；吞吐量越大，意味着程序能处理的请求越多，性能也就越好。这两个指标内部有一定的联系（同等条件下，延迟越短，吞吐量越大），但是由于它们隶属不同的维度（一个是时间维度，一个是空间维度），并不能互相转换。

我们所谓提升性能，从度量的角度，主要是**降低延迟，提高吞吐量**。这也是我们使用多线程的主要目的。那我们该怎么降低延迟，提高吞吐量呢？这个就要从多线程的应用场景说起了。

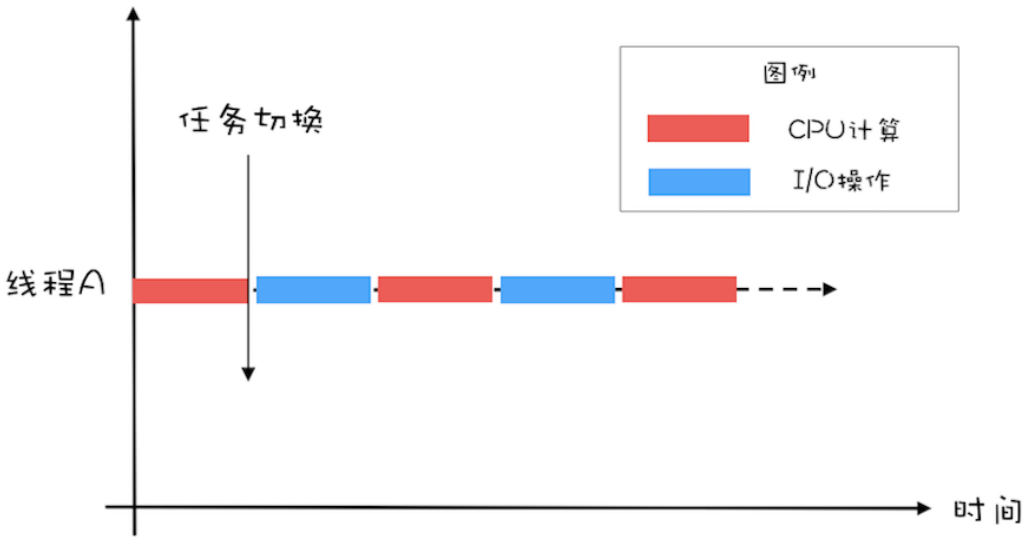
多线程的应用场景

要想“降低延迟，提高吞吐量”，对应的方法呢，基本上有两个方向，一个方向是**优化算法**，另一个方向是**将硬件的性能发挥到极致**。前者属于算法范畴，后者则是和并发编程息息相关了。那计算机主要有哪些硬件呢？主要是两类：一个是I/O，一个是CPU。简言之，**在并发编程领域，提升性能本质上就是提升硬件的利用率，再具体点来说，就是提升I/O的利用率和CPU的利用率**。

估计这个时候你会有个疑问，操作系统不是已经解决了硬件的利用率问题了吗？的确是这样，例如操作系统已经解决了磁盘和网卡的利用率问题，利用中断机制还能避免CPU轮询I/O状态，也提升了CPU的利用率。但是操作系统解决硬件利用率问题的对象往往是单一的硬件设备，而我们的并发程序，往往需要CPU和I/O设备相互配合工作，也就是说，**我们需要解决CPU和I/O设备综合利用率的问题**。关于这个综合利用率的问题，操作系统虽然没有办法完美解决，但是却给我们提供了方案，那就是：多线程。

下面我们用一个简单的示例来说明：如何利用多线程来提升CPU和I/O设备的利用率？假设程序按照CPU计算和I/O操作交叉执行的方式运行，而且CPU计算和I/O操作的耗时是1:1。

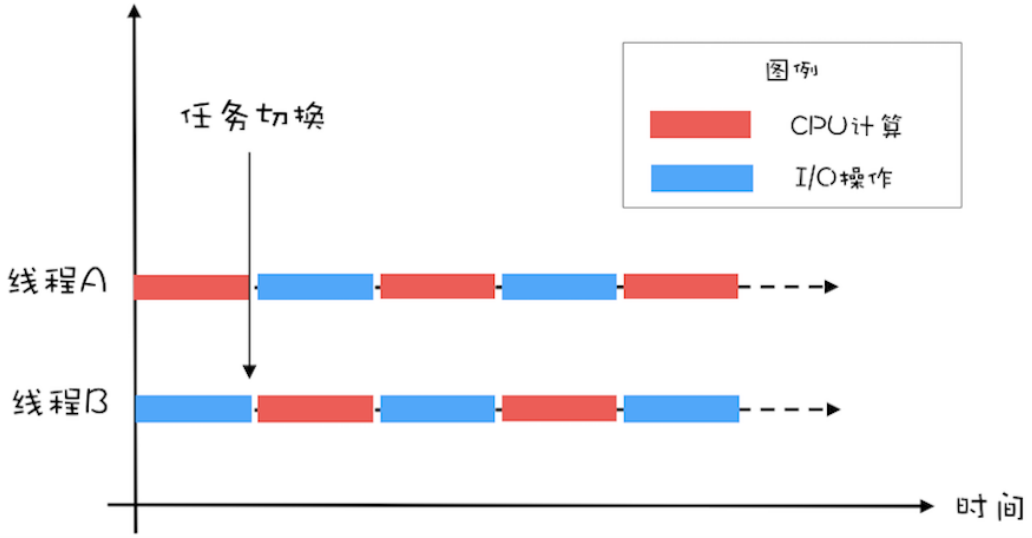
如下图所示，如果只有一个线程，执行CPU计算的时候，I/O设备空闲；执行I/O操作的时候，CPU空闲，所以CPU的利用率和I/O设备的利用率都是50%。



单线程执行示意图

如果有两个线程，如下图所示，当线程A执行CPU计算的时候，线程B执行I/O操作；当线程A执行I/O操作的时候，线程B执行CPU计算，这样CPU的利用率和I/O设备的利用率就都达到了

100%。

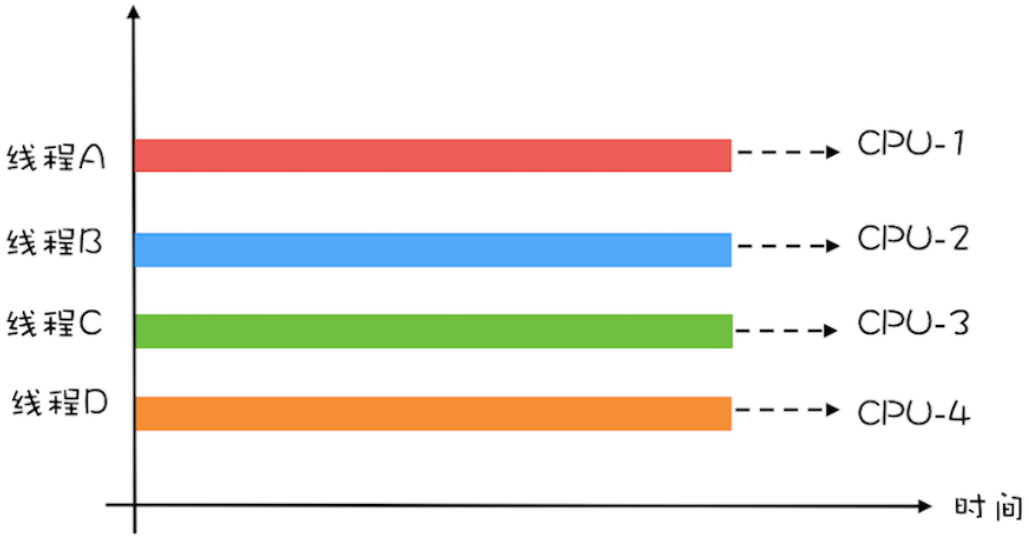


二线程执行示意图

我们将CPU的利用率和I/O设备的利用率都提升到了100%，会对性能产生了哪些影响呢？通过上面的图示，很容易看出：单位时间处理的请求数量翻了一番，也就是说吞吐量提高了1倍。此时可以逆向思维一下，**如果CPU和I/O设备的利用率都很低，那么可以尝试通过增加线程来提高吞吐量**。

在单核时代，多线程主要就是用来平衡CPU和I/O设备的。如果程序只有CPU计算，而没有I/O操作的话，多线程不但不会提升性能，还会使性能变得更差，原因是增加了线程切换的成本。但是在多核时代，这种纯计算型的程序也可以利用多线程来提升性能。为什么呢？因为利用多核可以降低响应时间。

为便于你理解，这里我举个简单的例子说明一下：计算1+2+.....+100亿的值，如果在4核的CPU上利用4个线程执行，线程A计算[1，25亿)，线程B计算[25亿，50亿)，线程C计算[50，75亿)，线程D计算[75亿，100亿]，之后汇总，那么理论上应该比一个线程计算[1，100亿]快将近4倍，响应时间能够降到25%。一个线程，对于4核的CPU，CPU的利用率只有25%，而4个线程，则能够将CPU的利用率提高到100%。



多核执行多线程示意图

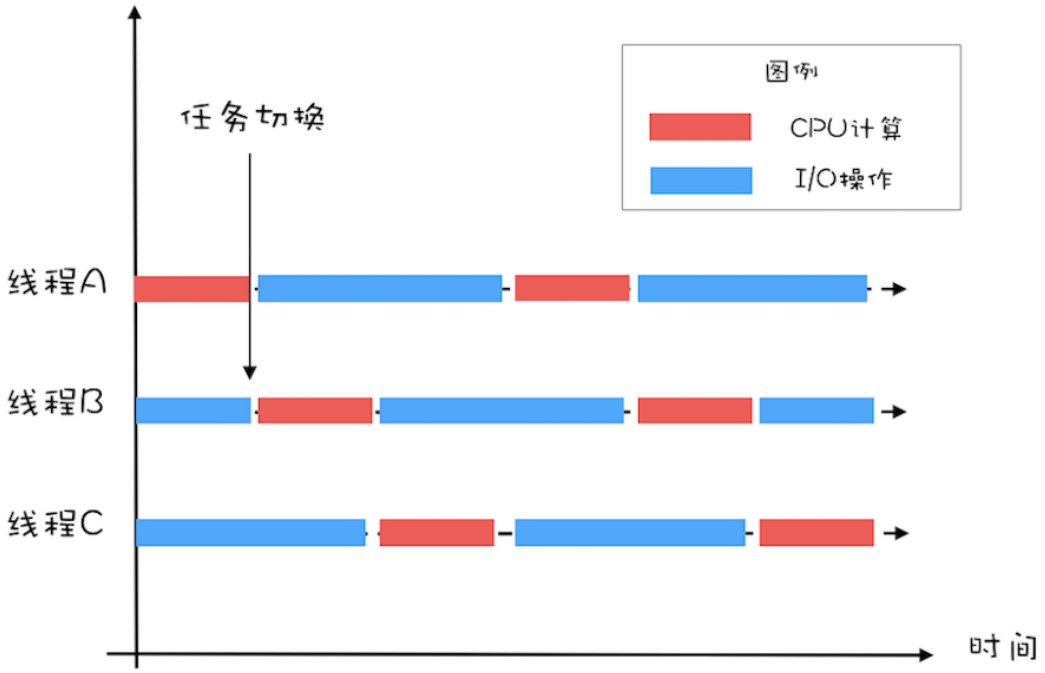
创建多少线程合适？

创建多少线程合适，要看多线程具体的应用场景。我们的程序一般都是CPU计算和I/O操作交叉执行的，由于I/O设备的速度相对于CPU来说都很慢，所以大部分情况下，I/O操作执行的时间相对于CPU计算来说都非常长，这种场景我们一般都称为I/O密集型计算；和I/O密集型计算相对的就是CPU密集型计算了，CPU密集型计算大部分场景下都是纯CPU计算。I/O密集型程序和CPU密集型程序，计算最佳线程数的方法是不同的。

下面我们对这两个场景分别说明。

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4核的CPU，每个核一个线程，理论上创建4个线程就可以了，再多创建线程也只是增加线程切换的成本。所以，**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**。不过在工程上，**线程的数量一般会设置为“CPU核数+1”**，这样的话，当线程因为偶尔的内存页失效或其他原因导致阻塞时，这个额外的线程可以顶上，从而保证CPU的利用率。

对于I/O密集型的计算场景，比如前面我们的例子中，如果CPU计算和I/O操作的耗时是1:1，那么2个线程是最合适的。如果CPU计算和I/O操作的耗时是1:2，那多少个线程合适呢？是3个线程，如下图所示：CPU在A、B、C三个线程之间切换，对于线程A，当CPU从B、C切换回来时，线程A正好执行完I/O操作。这样CPU和I/O设备的利用率都达到了100%。



三线程执行示意图

通过上面这个例子，我们会发现，对于I/O密集型计算场景，最佳的线程数是与程序中CPU计算和I/O操作的耗时比相关的，我们可以总结出这样一个公式：

> 最佳线程数=1 +（I/O耗时 / CPU耗时）

我们令R=I/O耗时 / CPU耗时，综合上图，可以这样理解：当线程A执行IO操作时，另外R个线程正好执行完各自的CPU计算。这样CPU的利用率就达到了100%。

不过上面这个公式是针对单核CPU的，至于多核CPU，也很简单，只需要等比扩大就可以了，计算公式如下：

> 最佳线程数=CPU核数 * [1 +（I/O耗时 / CPU耗时）]

总结

很多人都知道线程数不是越多越好，但是设置多少是合适的，却又拿不定主意。其实只要把握住一条原则就可以了，这条原则就是**将硬件的性能发挥到极致**。上面我们针对CPU密集型和I/O密集型计算场景都给出了理论上的最佳公式，这些公式背后的目标其实就是**将硬件的性能发挥到极致**。

对于I/O密集型计算场景，I/O耗时和CPU耗时的比值是一个关键参数，不幸的是这个参数是未知的，而且是动态变化的，所以工程上，我们要估算这个参数，然后做各种不同场景下的压测来验证我们的估计。不过工程上，原则还是**将硬件的性能发挥到极致**，所以压测时，我们需要重点关注CPU、I/O设备的利用率和性能指标（响应时间、吞吐量）之间的关系。

课后思考

有些同学对于最佳线程数的设置积累了一些经验值，认为对于I/O密集型应用，最佳线程数应该为：2 * CPU的核数 + 1，你觉得这个经验值合理吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

精选留言

CHEN川 36

更多的精力其实应该放在算法的优化上，线程池的配置，按照经验配置一个，随时关注线程池大小对程序的影响即可，具体做法：可以为你的程序配置一个全局的线程池，需要异步执行的任务，扔到这个全局线程池处理，线程池大小按照经验设置，每隔一段时间打印一下线程池的利用率，做到心里有数。

看到过太多的代码，遇到要执行一个异步任务就创建一个线程池，导致整个程序的线程池大到爆，完全没必要。而且大多数时候，提高吞吐量可以通过使用缓存、优化业务逻辑、提前计算好等方式来处理，真没有必要太过于关注线程池大小怎么配置，如果小了就改大一点，大了改小一点就好，从老师本文的篇幅也可以看出来。

经验值不靠谱的另外一个原因，大多数情况下，一台服务器跑了很多程序，每个程序都有自己的线程池，那CPU如何分配？还是根据实际情况来确定比较好。

2019-03-21

多拉拉f 15

多拉格·five 15

问一下老师，这个线程配置比我在其他的资料也看过，但是最后那个公式没见过，方便说一下如何测试IO/CPU 这个耗时比例吗

2019-03-21

作者回复

比较简单的工具就是apm了

2019-03-22

不靠谱的琴谱 12

如果我一个cpu是4核8线程 这里线程数数量是4+1还是8+1（cpu密集类型）

2019-03-21

aksonic 8

早起的鸟果然有食吃，抢到了顶楼，哈哈。

对于老师的思考题，我觉得不合理，本来就是分CPU密集型和IO密集型的，尤其是IO密集型更是需要进行测试和分析而得到结果，差别很大，比如IO/CPU的比率很大，比如10倍，2核，较佳配置：2*（1+10）=22个线程，而2*CPU核数+1 = 5，这两个差别就很大了。老师，我说的对不对？

2019-03-21

作者回复

不但起的早，还看懂了

2019-03-22

假行僧 7

个人觉得公式话性能问题有些不妥，定性的io密集或者cpu密集很难在定量的维度上反应出性能瓶颈，而且公式上忽略了线程数增加带来的cpu消耗，性能优化还是要定量比较好，这样不会盲目，比如io已经成为了瓶颈，增加线程或许带来不了性能提升，这个时候是不是可以考虑用cpu换取带宽，压缩数据，或者逻辑上少发送一些。最后一个问题，我的答案是大部分应用环境是合理的，老师也说了是积累了一些调优经验后给出的方案，没有特殊需求，初始值我会选大家都在用伪标准

2019-03-21

作者回复

□□

2019-03-21

探索无止境 5

老师早上好，当应用来的请数量过大，此时线程池的线程已经不够使用，排队的队列也已经满了，那么后面的请求就会被丢弃掉，如果这是一个更新数据的请求操作，那么就会出现数据更新丢失，老师有没有什么具体的解决思路？期待解答

2019-03-21

作者回复

单机有瓶颈，就分布式。

数据库有瓶颈，就分库分表分片

2019-03-22

阿冲

4

老师，你好！有个疑惑就是我在写web应用的时候一般都是一个请求里既包含cpu计算（比如字符串检验）又包含操作（比如数据库操作），这种操作就是一个线程完成的。那么这种情况按你写的这个公式还起作用吗？c#里面有对io操作基本都封装了异步方法，很容易解决我刚说的问题(调用异步方法就会切换线程进行io操作，等操作完了再切回来)。java要达到这种效果代码一般怎么写比较合适？

2019-03-21

作者回复

就是针对一个线程既有cpu也有io的，这个才是io密集型

2019-03-22

董宗磊

3

思考题：认为不合理，不能只考虑经验，还有根据是IO密集型或者是CPU密集型，具体问题具体分析。

看今天文章内容，分享个实际问题；我们公司服务器都是容器，一个物理机分出好多容器，有个哥们设置线程池数量直接就是：Runtime.getRuntime().availableProcessors() * 2；本来想获取容器的CPU数量 * 2，其实Runtime.getRuntime().availableProcessors()获取到的是物理机CPU合数，一下开启了好多线程 ^_^

2019-03-21

作者回复

新版的jvm开始支持docker了，老版本问题还挺多

2019-03-22

zsh0103

3

请问老师，

1 在现实项目如何计算I/O耗时与CPU耗时呢，比如程序是读取网络数据，然后分析，最后插入数据库。这里网络读取何数据库插入是两次IO操作，计算IO耗时是两次的和吗？

2. 如果我在一台机器上部署2个服务，那计算线程数是要每个服务各占一半的数量吗？

3. 如果我用一个8核CPU的机器部署服务，启动8个不同端口的相同服务，和启动一个包含8个线程的服务在处理性能上会有区别吗？

2019-03-21

作者回复

1.两次之和

2.理论值仅仅适用部署一个服务的场景。

3.有区别

2019-03-22

Weixiao

2

最佳线程数 =1 +（I/O 耗时 / CPU 耗时），

文中说，1表示一个线程执行io，另外R个线程刚好执行完cpu计算。

这里理解有点问题，这个公式是按照单核给出的，所以不可能存在同时R个线程执行cpu计算。所以我理解文章中说反了，应该是1个线程在执行cpu，然后有R个线程可以同时在执行io，这样cpu的利用率为100%

2019-03-24

作者回复

你对照着图理解一下，cpu时间上没有重叠

2019-03-24

摇山樵客™ 2

在4核8线程的处理器使用Runtime.availableProcessors()结果是8，超线程技术属于硬件层面上的并发，从cpu硬件来看是一个物理核心有两个逻辑核心，但因为缓存、执行资源等存在共享和竞争，所以两个核心并不能并行工作。超线程技术统计性能提升大概是30%左右，并不是100%。另外，不管设置成4还是8，现代操作系统层面的调度应该是按逻辑核心数，也就是8来调度的（除非禁用超线程技术）。所以我觉得这种情况下，严格来说，4和8都不一定是合适的，具体情况还是要根据应用性能和资源的使用情况进行调整。这是个人的理解，请老师指正。

2019-03-22

作者回复

工作中都是按照逻辑核数来的，理论值和经验值只是提供个指导，实际上还是要靠压测。

2019-03-22

QQ怪 2

我就想问下如何测试io耗时和cpu耗时

2019-03-21

作者回复

apm工具可以

2019-03-22

已忘二 2

老师，有个疑问，就是那个I/O和CPU比为2:1时，CPU使用率达到了100%，但是I/O使用率却到了200%，也就是时刻有两个I/O同时执行，这样是可以的么？I/O不需要等待的么？

2019-03-21

作者回复

io有瓶颈后，cpu使用率就上不去了

2019-03-22

狂战俄洛伊 2

对于这个思考题，我觉得是比较合理。
因为经验是经过大量实践的结果，是符合大多数的情况，而且是一种快速估计的方法。
我看留言区里很多都说不合理，并且给出了例子。我觉得他们说的也没错，只是举出了经验没覆盖到的情况而已。
这里我还有个疑问，这篇文章中都是在讲一台机器工作的情况下。我想问的是如果是在一个集群里，这个线程数又该怎么计算？
例如有三台机器构成一个集群，这三台机器的cpu分别是8核，4核，2核。就打算是cpu密集型，这时候该怎么计算线程数？

2019-03-21

作者回复

每台机器算自己的，发挥出每台机器的硬件能力就可以了

2019-03-21

姜戈 2

2*CPU核数+1，我觉得不合理，针对IO密集型，老师提供的公式是：CPU核数*（1+IO耗时/CPU耗时）。2*CPU核数+1这个公式相当于这里有个潜在估计，假设了IO消耗时间与CPU消耗时间1:1，再加一个线程用来预防其中有某个线程被阻塞，及时顶上。针对IO密集型，要考虑的就是IO耗时与CPU耗时之比！这个经验公式只是针对其中1:1耗时比一种情况，不够全面！

2019-03-21

陈华应 1

理论加经验加实际场景，比如现在大多数公司的系统是以服务的形式来通过docker部署的，每个docker服务其实对应部署的就一个服务，这样的情况下是可以按照理论为基础，再加上实际情况来设置线程池大小的，当然通过各种监控来调整是最好的，但是实际情况是但服务几十上百，除非是核心功能，否则很难通过监控指标来调整线程池大小。理论加经验起码不会让设置跑偏太多，还有就是服务中的各种线程池统一管理是很有必要的

2019-03-31

作者回复

说的太对了！！！

2019-03-31

空知 1

刚好看了这个文章 https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing 里面就有讲 connections = ((core_count * 2) + effective_spindle_count)

2019-03-22

马晓光 1

实际项目中怎么确定IO耗时、CPU耗时？

2019-03-22

作者回复

apm工具可以精确到方法耗时，io相关的方法一般是知道的

2019-03-22

曾轼麟 1

老师我记得csapp那本书中说过，x86架构的CPU是拥有超程技术的，也就是一个核可以当成两个使用，AMD的却没有，不知道您的这个计算公式是否适合其它厂商的CPU呢？

2019-03-22

作者回复

都按照逻辑核数设置，最终还是要根据压测数据调整的

2019-03-22

walkingonair 1

当I/O 耗时远远大于CPU耗时时，"2 * CPU 的核数 + 1"会导致所有线程在长时间下都处于等待I/O操作的状态，而无法合理利用CPU

2019-03-21