多核平台下的Java优化 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/645/2021_2022__E5_A4_9A_E6_A0_B8_E5_B9_B3_E5_c104_645253.htm 现在多核 CPU 是主流。利用多核技术，可以有效发挥硬件的能力，提升吞吐量，对于 Java 程序，可以实现并发垃圾收集。但是 Java 利用多核技术也带来了一些问题，主要是多线程共享内存引起了。目前内存和 CPU 之间的带宽是一个主要瓶颈，每个核可以独享一部分高速缓存，可以提高性能。JVM 是利用操作系统的“ 轻量级进程 ”实现线程，所以线程每操作一次共享内存，都无法在高速缓存中命中，是一次开销较大的系统调用。所以区别于普通的优化，针对多核平台，需要进行一些特殊的优化。 代码优化 线程数要大于等于核数 如果使用多线程，只有运行的线程数比核数大，才有可能榨干 CPU 资源，否则会有若干核闲置。要注意的是，如果线程数目太多，就会占用过多内存，导致性能不升反降。JVM 的垃圾回收也是需要线程的，所以这里的线程数包含 JVM 自己的线程 尽量减少共享数据写操作 每个线程有自己的工作内存，在这个区域内，系统可以毫无顾忌的优化，如果去读共享内存区域，性能也不会下降。但是一旦线程想写共享内存（使用 volatile 关键字），就会插入很多内存屏障操作（Memory Barrier 或者 Memory Fence）指令，保证处理器不乱序执行。相比写本地线程自有的变量，性能下降很多。处理方法是尽量减少共享数据，这样也符合“ 数据耦合 ”的设计原则。 使用 synchronize 关键字 在 Java1.5 中，synchronize 是性能低效的。因为这是一个重量级操作，需要调用操作接口，导致有可能加锁消耗的系统时

间比加锁以外的操作还多。相比之下使用 Java 提供的 Lock 对象，性能更高一些。但是到了 Java1.6，发生了变化。synchronize 在语义上很清晰，可以进行很多优化，有适应自旋，锁消除，锁粗化，轻量级锁，偏向锁等等。导致在 Java1.6 上 synchronize 的性能并不比 Lock 差。官方也表示，他们也更支持 synchronize，在未来的版本中还有优化余地。 使用乐观策略 传统的同步并发策略是悲观的。表现语义为：多线程操作一个对象的时候，总觉得会有两个线程在同时操作，所以需要锁起来。乐观策略是，假设平时就一个线程访问，当出现了冲突的时候，再重试。这样更高效一些。Java 的 AtomicInteger 就是使用了这个策略。 使用线程本地变量（ThreadLocal） 使用 ThreadLocal 可以生成线程本地对象的副本，不会和其他线程共享。当该线程终止的时候，其本地变量可以全部回收。 类中 Field 的排序 可以将一个类会频繁访问到的几个 field 放在一起，这样他们就有更多的可能性被一起加入高速缓存。同时最好把他们放在头部。基本变量和引用变量不要交错排放。 批量处理数组 现在处理器可以用一条指令来处理一个数组中的多条记录，例如可以同时向一个 byte 数组中读或者写 store 记录。所以要尽量使用 System.arraycopy （）这样的批量接口，而不是自己操作数组。 JVM 优化 启用大内存页 现在一个操作系统默认页是4K.如果你的 heap 是4GB，就意味着要执行1024*1024次分配操作。所以最好能把页调大。这个配额设计操作系统，单改 Jvm 是不行的。Linux 上的配置有点复杂，不详述。 在 Java1.6 中 UseLargePages 是默认开启的，LasrgePageSzieInBytes 被设置成了4M.笔者看到一些情况下配置成了128MB，在官方的性能测

试中更是配置到256MB. 启用压缩指针 Java 的64的性能比32慢，原因是因为其指针由32位扩展到64位，虽然寻址空间从4GB扩大到 256 TB，但导致性能的下降，并占用了更多的内存。所以对指针进行压缩。压缩后的指针最多支持32GB 内存，并且可以获得32位 JVM 的性能。 在 JDK6 0update 23 默认开启了，之前的版本可以使用-XX： UseCompressedOops 来启动配置。 性能可以看这个评测，性能的提升是很可观。 启用 NUMA numa 是一个 CPU 的特性。SMP 架构下，CPU 的核是对称，但是他们共享一条系统总线。所以 CPU 多了，总线就会成为瓶颈。在 NUMA 架构下，若干 CPU 组成一个组，组之间有点对点的通讯，相互独立。启动它可以提高性能。 NUMA 需要硬件，操作系统，JVM 同时启用，才能启用。Linux 可以用 numactl 来配置 numa，JVM 通过-XX： UseNUMA 来启用。 激进优化特性 在 Java1.6 中，激进优化（AggressiveOpts）是默认开启的。激进优化是一般有一些下一个版本才会发布的优化选项。但是有可能造成不稳定。前段时间以讹传讹的 JDK7的 Bug，就是开启这个选项后测到的。 逃逸分析 让一个对象在一个方法内创建后，如果他传递出去，就可以称为方法逃逸；如果传递到别的线程，成为线程逃逸。如果能知道一个对象没有逃逸，就可以把它分配在栈而不是堆上，节约 GC 的时间。同时可以将这个对象拆散，直接使用其成员变量，有利于利用高速缓存。如果一个对象没有线程逃逸，就可以取消其中一切同步操作，很大的提高性能。 但是逃逸分析是很有难度的，因为花了 cpu 去对一个对象去分析，要是他不逃逸，就无法优化，之前的分析血本无归。所以不能使用复杂的算法，同时现在的 JVM 也没有实

现栈上分配。所以开启之后，性能也可能下降。 可以使用-XX： DoEscapeAnalysis 来开启逃逸分析。 高吞吐量 GC 配置 对于高吞吐量，在年轻态可以使用 Parallel Scavenge，年老态可以使用 Parallel Old 垃圾收集器。 使用-XX：UseParallelOldGC 开启 可以将-XX：ParallelGCThreads 根据 CPU 的个数进行调整。可以是 CPU 数的1/2或者5/8 低延迟 GC 配置 对于低延迟的应用，在年轻态可以使用 ParNew，年老态可以使用 CMS 垃圾收集器。 可以使用-XX：UseConcMarkSweepGC 和-XX： UseParNewGC 打开。 可以将-XX：ParallelGCThreads 根据 CPU 的个数进行调整。可以是 CPU 数的1/2或者5/8 可以调整-XX：MaxTenuringThreshold（晋升年老代年龄）调高，默认是15.这样可以减少年老代 GC 的压力 可以-XX：TargetSurvivorRatio，调整 Survivor 的占用比率。默认50%.调高可以提供 Survivor 区的利用率 可以调整-XX：SurvivorRatio，调整 Eden 和 Survivor 的比重。默认是8.这个比重越小，Survivor 越大，对象可以在年轻态呆更多时间。 protectAndRun("render_ads.js::google_render_ad", google_handleError, google_render_ad). 百考试题温馨提示：本内容来源于网络，仅代表作者个人观点，与本站立场无关，仅供您学习交流使用。其中可能有部分文章经过多次转载而造成文章内容缺失、错误或文章作者不详等问题，请您谅解。如有侵犯您的权利，请联系我们，本站会立即予以处理。编辑特别推荐: #0000ff>Java读取文件内容再编辑 #0000ff>JS获取单选与多选按纽的值 #0000ff>每一种文件类型所对应的ContentType 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com