25 | 性能调优什么时候应该停止？

2021-07-13 尉刚强

《性能优化高手课》 课程介绍 >

讲述：尉刚强
时长 15:25 大小 14.12M

你好，我是尉刚强。

我在以往参与性能优化项目的过程中，曾不止一次地被问到过：**软件性能调优什么时候应该停止呢？**因为我发现，有很多研发人员在做性能调优的过程中，可能进展并不理想，而且也因为性能优化目标迟迟没有达成，然后就陷入了性能调优什么时候才能结束的迷茫中。

其实，这个问题也曾困扰过我，我当时在参与第一个性能优化项目的时候，每天的工作就是寻找代码中的一些低效率实现，然后修改重构并验证性能提升效果，日复一日。所以我当时就很想说服团队 Leader 这个性能调优任务可以结束了。但是，首先我都没有办法说服我自己，也是基于这个原因，我才开始思考这个问题。

所以这节课，我想和你按照线性递进的思考逻辑，去寻找解答这个问题的最佳答案。我会先和你一起讨论之所以产生这个问题的根源，然后带你剖析下正确的性能调优的方法步骤。最后在这个过程中，我们就可以寻找到在做性能调优时一些需要喊停的死角区域，从而就可以在实际的软件开发设计中，更高效地做好性能调优工作，更容易达成性能优化的目标。

好，那么下面，我们就先来思考下，这个问题提出背后的原因是什么呢？

为什么会提出这个问题？

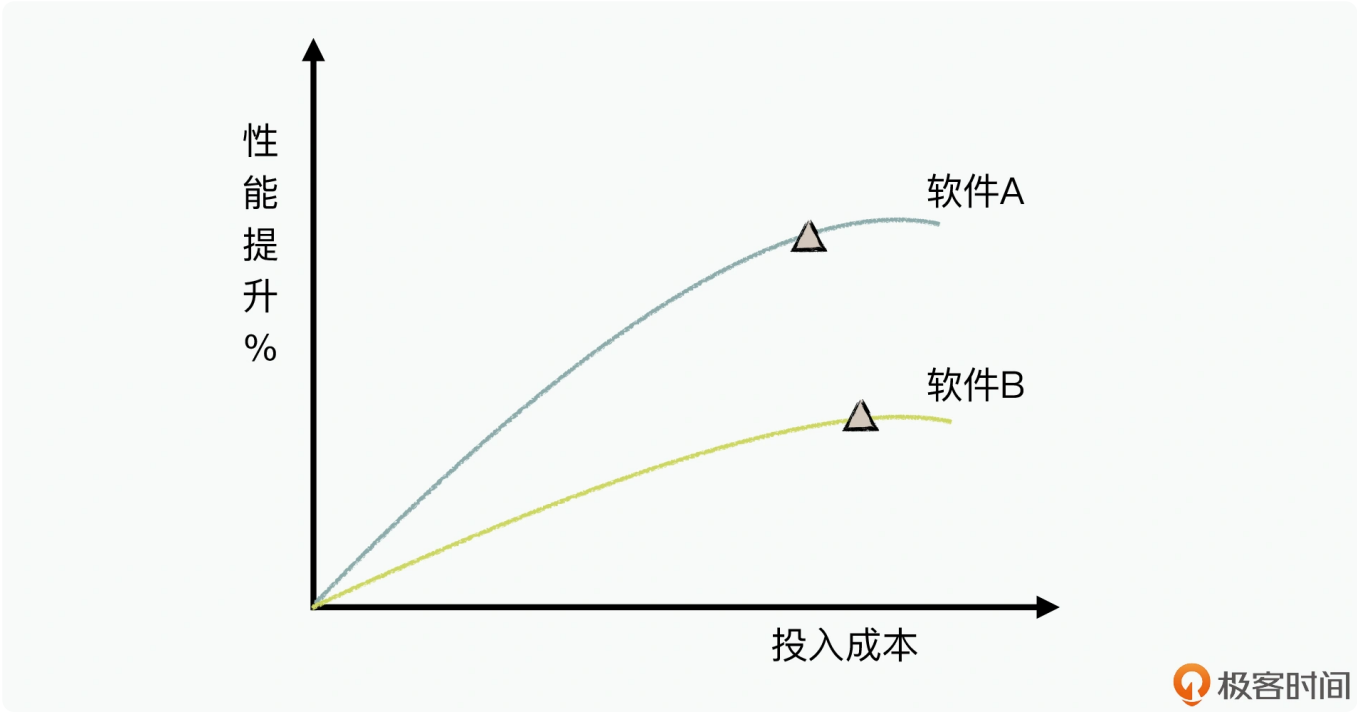
我们先来做个假设，现在团队开发的一个软件产品需要进行性能调优，其指定的性能调优目标是提升 20%。

那么我们来思考一下，这个目标好达成吗？

其实**在进行深入性能分析之前，我们是很难回答这个问题的**。这是因为，不同软件的设计与实现差异很大，而针对性能这个模块，我们可以优化提升的空间是不一样的。

我举个真实的例子，在我曾经参与的一个协议栈报文子系统的性能优化项目中，因为在代码实现优化中减少了一次内存拷贝，就一次性将系统处理的性能提升了 20%。而在我参与的另一个配置管理子系统的优化项目中，由于没有找到比较大的性能优化点，所以花费了很长时间与精力，才将性能提升了 10% 左右。

所以我才说，不同软件系统的性能优化提升效果和优化投入成本之间的关系差异很大，具体可以参考下图：



在这个图中，你可以观察到两个比较明显的规律：

1. 不同软件系统（软件 A 和软件 B）在性能调优的过程中，能够达到的性能提升百分比上限是不一样的。
2. 在性能调优的前期，投入很少成本就可以获取比较好的性能提升效果，但是在性能调优的中后期，获取同样多的性能收益，需要花费的精力和成本会越来越大。

其实，在进行性能调优的时候，首要追求的目标应该是**最大的投资收益比**，也就是获取的性能优化收益值和消耗工作量成本之间的比值要最高。

所以在理想情况下，我们应该将性能调优目标设定到一个性能提升临界值（通常会接近性能提升的上限）。如果达到这个临界值就意味着，即使后续进行再多的性能调优工作，我们能获取的性能收益都会越来越有限。那么在这个时候，我们就可以适当调整下性能调优的节奏。如前面的示意图所示，软件 A 和软件 B 的性能调优目标设置的临界值，可能会在三角形所标识的位置附近。

但问题是，对于一个软件系统来说，**性能调优提升目标的临界值要设定成多少才是合理的呢？我们要如何确定这个临界值呢？**

所以在一般情况下，研发团队在设定性能调优目标时，会采取两种方式：

第一种，**以客户关注的性能需求目标为导向**。比如我之前参与的百万表单数据查询分析优化项目，它的核心目标就是客户在操作过程中不卡顿，因此我只需把查询请求响应时间优化到 1 秒内即可；

第二种，**以降低产品的部署运维成本为导向**。这种方式通常会先敲定一个性能提升百分比，比如将系统服务的响应时间降低 20%（从 100ms 到 80ms），减少产品部署使用的集群机器规模 20%，等等。

不过这里你也要注意，就是不管采用哪种方式制定的性能调优目标，它都可能无法与软件优化可以达到的临界值完全匹配。那么在这种场景下，就会很容易导致性能调优的目标没有达成，但是性能调优任务却无法继续开展的情况。

所以，我们在性能调优的过程中，一定要谨记一点：**未经分析就敲定性能优化的目标是不可取的。**

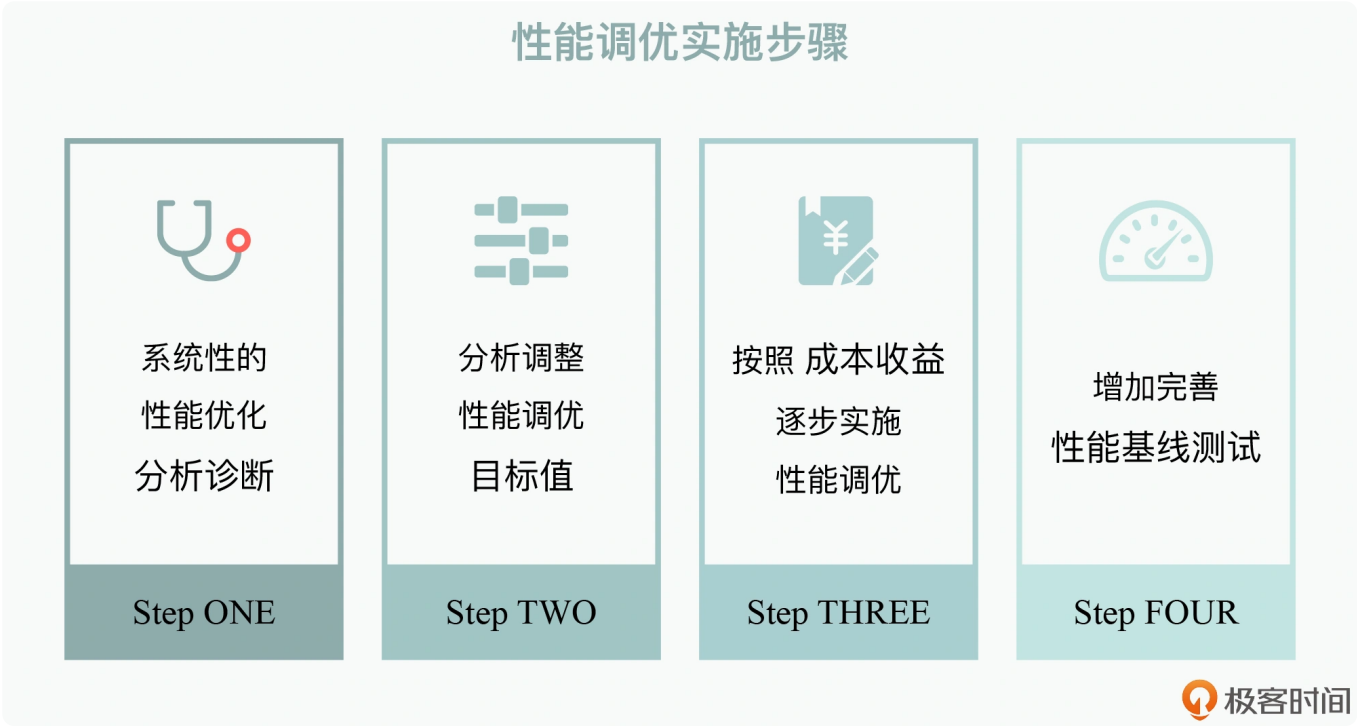
既然如此，那么正确开展和实施性能调优的方法步骤是什么呢？下面我就带你来分析分析。

正确开展性能调优的方法步骤

实际上，很多研发团队心目中的性能调优工作，可能就是选择一款代码 Profiling 工具，然后针对软件执行期间进行性能分析，逐个寻找热点函数，最后进行修改和优化。

但是我们要知道，这种方法存在很大的局限性，**它可以识别出的性能优化点非常有限**。比如说，并发设计、通信设计、IO 设计等软件设计引入的性能问题，它都无法识别出来；不仅如此，软件编码实现层引入的性能问题，它也都无法识别出来，比如数据结构和算法选择等。

所以这里，我就根据以往参与的性能优化项目经验，总结出了实施性能调优的方法步骤，接下来我就给你具体分析一下。



第一步，系统性的性能优化分析诊断。

这里你可以采用第 22 讲学习的性能问题分析定位方法，自顶向下地分析识别所有导致性能劣化的可优化点。这里输出的应该包含软件设计优化点、软件实现优化点等比较完整的列表，比如调整并发任务拆分、调整数据结构、选择性能优化模式，等等。

第二步，分析调整性能调优目标值。

这一步的意思就是根据识别出的性能优化点，分析修改后的性能提升收益。但要注意，这一步针对每个优化点的分析过程是不一样的，而且并没有统一的方法参考。

因此这里为了帮助你更好地理解这个过程，我给你举两个我以前参与的性能优化案例，来具体说明下。

案例 1：一个协议栈报文子系统的性能优化项目

在这个项目中，我们通过基于性能优化分析诊断后发现：业务在处理过程中，对报文数据执行了一次 copy 操作，但是协议在处理过程中，只修改了报文数据头部很少一部分字节的信息。所以在这种场景下，业务中的 copy 操作开销就可以优化掉。那么优化修改后的性能提升值有多少呢？这里我根据 copy 的数据量在单板上进行了测量计算，然后就在优化修改之前，计算出了性能的预期收益。

案例 2：一个后端微服务的性能优化项目

在这个项目中，我们经过性能优化分析诊断后发现，业务存在很多的慢查询操作，对软件性能造成的影响比较大。而在进一步分析后发现，这些慢查询所获取数据其实是很少变化的，所以就考虑采用缓存策略来优化性能。那么在这种场景下，我就可以根据慢查询的请求处理时延和请求的频次，来分析计算出引入 Cache 场景下的性能提升收益。

总之你要知道，对于性能优化点来说，性能提升收益分析是一个非常重要的环节，不应该被忽视。

第三步，按照成本收益逐步实施性能调优。

接下来，我们就可以对性能优化点按照优先级进行排序，逐步修改并验证优化效果。那么在对性能优化点进行排序时，我们需要考虑的因素主要有几个：性能收益大小、修改的工作量大小，以及对软件质量产生的影响（比如导致软件变复杂、引入故障风险高等）。

另外这里你要记住，如果把编译期选项配置优化和编码实现优化进行优先级排序，在同等性能收益的情况下，一般来说编译期优化的修改工作量会比较小，引入故障的风险率比较低，所以优先级应该更高一些。

> 补充：拿性能调优对软件质量产生的影响来说，并不是所有的软件调优工作都会导致软件质量变差。比如，你选择了更适合业务特性的数据结构和算法，可能不仅没有导致质量变差，可能还会让业务处理逻辑更加清晰、更易理解。

第四步，增加完善性能基线测试。

当性能调优合入后，你就可以同步修改完善性能基线测试了。这部分你可以参考第 20 讲学习的内容，复习下更好地看护软件系统性能的实现思路。

不过事实上，很少有研发团队可以按照上述的步骤来实施性能调优，因此在性能调优过程中，就容易陷入僵局，而且还花费很大的精力，却并没有给软件产品带来价值提升。所以在这个时候，研发团队就应该及时喊停，重新调整性能调优的工作方式与节奏了。

那么你再想想，除此之外，在其他什么样的场景下，还应该及时喊停呢？接下来我就给你分享下，我在实践过程中总结出来的答案。

什么时候需要喊停性能调优工作？

这里，我总结了以前参与软件性能调优过程中，观察到性能调优的一些反模式，你可以参考下，如果你发现自己所在的研发团队也出现这些状况时，就应该考虑调整下性能调优的方向和策略。

第一种性能调优反模式是：**性能调优严重破坏了软件的质量。**

这里我举一个真实的案例。在我曾经参与的一个嵌入式系统性能优化项目中，原来的性能优化团队发现，通过宏替换个别函数调用会带来性能提升，于是就几乎将代码中的所有函数都通过宏重新实现来整改替换。而最后导致的后果就是：大量的宏实现函数导致代码编写和阅读成本显著增大；同时在代码整改的过程中，引入了非常多的故障，而且很长时间无法得到很好的解决；更糟糕的是，最后的软件性能优化效果也没有达到预期。

其实，这种严重破坏软件设计质量的性能调优还是比较普遍的，比如说，在代码中随意添加条件分支进行特殊处理、最后因为加入太多特殊流程，导致代码很难再添加新的业务特性。

第二种性能调优反模式是：**盲目修改代码来尝试优化。**

有的性能优化团队为了提升指令 Cache 命中率，会随机调整函数的位置。比如说，把一个函数从一个文件中搬移到另外一个文件中；或者把一个函数从一个类搬移到另外一个类中，来判断 Cache 命中率是否有提升。这种性能调优方式，由于背后并没有理论指导，即使可以获取到一些短暂的性能提升收益，也是不稳定的，所以我们应该尽量避免这样做。

第三种性能调优反模式是：**在业务的非性能瓶颈点上反复调优。**

举个简单的例子，软件的查询请求处理的吞吐量，受制于底层网络传输带宽值的上限，理论上不可能再提升。这个时候，还在持续分析调优软件实现，期望提升吞吐量是没有任何意义的。

第四种性能调优反模式是：**没有价值驱动的性能调优。**

其实这种情况也挺常见，在软件系统中存在一些服务 / 组件（比如：操作事务记录，配置管理后台等），它们的处理性能并不会直接影响用户感受，而且占用的机器资源都很少，这时候如果还投入很大的工作量去优化软件性能，其实是没有意义的。

小结

今天这节课，我们学习了正确实施性能调优的方法步骤，不过总的来说，在性能调优的过程中，需要深入到业务与实现中，并使用专栏中学习到很多技术与方法，才能更高效支持性能调优工作。这里你需要注意，分析出性能优化点应该是包含了软件设计、软件编码等多方位的性能优化点，而且应该按照一定的优先级排序来逐步实施优化。

同时，在性能调优过程中还有很多比较常见的误区，比如严重破坏软件设计的性能调优、盲目修改代码的调优、非性能瓶颈点上的反复调优、没有价值驱动的性能调优等。你也可以一起学习下，避免在性能调优过程中出现类似的问题。

在学习完今天的课程后，你应该就可以掌握性能调优的方法和节奏，然后比较轻松地回答这个问题：性能调优应该在什么时候停止？

思考题

在性能调优的过程中，如果你发现导致业务请求时延长的原因，是存在很多的数据库慢查询请求，那么你该怎么优化呢？

欢迎给我留言，分享你的思考和观点。如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。

上一篇 24 | 动态监控：你的产品系统中有动态监控的能力吗？

下一篇 26 | 一个嵌入式实时系统都要从哪些方面做好性能优化？

精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。