服务端问题剖析浅谈

bd7xzz 已于 2023-10-20 21:05:17 修改

这是一篇很随性的浅谈，主要围绕着作为一个服务端程序员如何解决疑难杂症这个话题。我是一名使用 java 的程序员，在我的认知范围内，java 还是擅长于服务端业务编程。即：拥有完善的解决企业级信息化问题的生态，适用于服务端非极端性能要求下的一种编程语言。这里的重点是：

1. 生态健全，能够快速应对各种复杂业务。
2. 面向服务端，众所周知 java 在客户端（浏览器、桌面端、嵌入式）领域做的不是很好，只有安卓还说得过去。
3. 性能中立，现代 jvm 性能在理论上不次于 c++，但中间件、操作系统等对性能、高并发要求苛刻的场景中，选择 java 作为开发语言的很少。
   所以对于 80% 的 java 程序员，可能就是写写业务代码。甚至在中国计算机行业环境下，不进入大厂是无法接触到高并发和大数据量的。导致自身技术无法提升，即使是死记硬背八股文，最后也只是停留在理论（N 多天后就忘了），到头来也只是个干业务的 java 程序员。当然，进了大厂也不一定能接触到核心技术，一样是个普通的 java 程序员。所以对于很多人 35 岁就终结了职场生涯。
   不是 java 程序员还是什么？答：服务端工程师。
   为啥叫服务端工程师？一个优秀的程序员，不只局限于一种 java 语言，可以是多种语言或基于 java 语言能够做到极致，解决一切关于服务端相关的问题。这样的程序员，我认为可称为服务端工程师。其应该具有的能力和素质包括不限于：
4. 用自己和团队所能把控的技术解决一切服务端需求（无论是纯业务的还是技术的）
5. 具有独立的设计和思考能力，能够对技术实现方案提出质疑（无论是质疑别人还是自己）
6. 负责、不要甩锅，但知道边界（在你的职责范围内切勿干涉别人的领域和负责的事）
7. 能够解决一切服务端遇到的问题，这里包括不限于：

- 自己和他人产生的问题：自己做的东西遇到问题自己要能解决，历史遗留的老系统（无数人写的屎山代码）在没有任何说明和帮助的情况下快速接手并解决问题。
- 非功能性和功能性问题：功能性问题指的是业务问题。非功能性问题指的是性能、稳定性、可扩展性等技术问题。
  可能很多人认为，有架构师和基础架构组帮忙解决非功能性问题就够了，我只要能开发业务需求就可以了。如果你这样想，必定会面临 35 岁的噩梦。

上面四点对一个服务端工程师的定义，本文只想讨论：一个服务端程序员如何解决疑难杂症，即怎么能够解决一切服务端遇到的问题。
我认为一个优秀的服务端工程师，表现能力的时刻不只是设计和编码，很重要一点是解决问题的思路和手段。一般打法为：遇到问题 -> 提出猜想 -> 分析并验证 -> 解决问题 -> 复盘。
这个打法不是为了卷，而是为了解决问题并对自己掌握的知识做迭代更新。
提出猜想：根据过往的经验，先粗略评估出排查问题的方向，可以根据过往经验和理论知识去定。这样遇到问题不会手忙脚乱。
分析并验证：有了大体方向，需要使用工具、实验去印证方向是否正确。不正确，无法解决问题，退回第一步重新提出猜想，继续验证，直到能够给出解决问题的最终方案。
解决问题：有了最终方案，解决问题就简单了。
复盘：复盘不是为了追责举办一场批斗大会，而是总结解决问题的思路、分享经验。复盘不局限于形式，譬如写一个简单的分析过程分享给大家，甚至是自己总结成笔记（说出来、写出来远比自己大脑里想过一遍理解的更深刻）。
解决功能性问题取决于自己对业务的理解，对系统的理解。这两点通常是需要时间的，比如亲身参与过系统的设计和功能的开发，或过往经历有业务领域的知识储备。但有的时候并不是这么美好，比如：

1. 你刚到一家新公司的第一天，领导不容分说就让你上手干活。
2. 你接手了一个庞大的老系统，之前的研发人员都走了，没留下任何文档，而你对该业务领域并不熟悉。
   这时，你唯一能做的就是硬着头皮上，通过一些普适的技术手段对这个黑盒进行分析。
   解决非功能性问题取决于你对计算机基础技术的掌握程度，无论是在设计开发阶段还是上线后，基础牢靠的人总会提前发现问题并解决掉，或在出现问题时有独到的手段解决问题。基础本质上就是大学的课程：
3. 数据结构与算法：设计开发阶段可以针对面临的问题，找出合适的算法。这不是说要自己写个多牛 b 的算法和数据结构，而是能否找出一个现成的东西（中间件、数据库、类库等）解决问题。比如了解 redis 的数据结构原理，选一个合适的结构存储你的数据，能够成倍加速你的应用性能。
4. 操作系统：在面对高并发和大数据量涌入服务器的时候，操作系统也不是那么可靠。或者操作系统实现的机制中可以进行调优，加快应用的计算速度。
5. 网络：我们都知道 TCP 是可靠传输的，但实际并不那么可靠。总会遇到丢包、超时重传等问题，会导致用户体验极差。我们要有能力排查这些问题。
6. 编程语言基础：编程语言基础不只是语法规则，对 jvm 内存管理、线程、各种类库的深入理解，可以在出现问题时快速定位问题。
7. 计算机组成原理：同样面对高并发和大数据量的情况，对 CPU、内存、磁盘、网络的硬件工作原理有深入的理解，加上对操作系统的深入理解，能够加速解决一些极端的问题。

本质上，一个服务端工程师，不需要掌握多种编程语言，熟悉多种数据库、中间件的使用。上面的基础打牢靠了就非常不错了。任何一个大厂的面试，无论是八股文还是面试官的随性提问，都离不开这些基础。这种刁钻刻薄的问题不是为了卷，而是复杂系统，在高并发的流量冲击下，每一行代码都需要深思熟虑。

无论是功能性问题和非功能性问题，都可以使用的普适手段：

1. run 起来：即使是接手的屎山代码也要先跑起来，才能观察其内部机制。遇到线上问题，先考虑能否重放有问题的请求复现问题。

2. 寻找可观测的手段：系统若是有链路跟踪，可以使用链路跟踪分析问题。没有可以尝试查找日志（可能需要熟悉操作系统命令），观察监控系统的各种指标（所以需要对常用的业务和非业务指标名词有基本的了解）。
3. 使用一些工具：我很喜欢收集一些工具，比如观测 CPU、内存、磁盘、网络的工具、jvm 分析工具等。
   不得不说，对各种工具熟练使用，能够加快排查问题的速度。但对专业工具能够熟练使用的前提，是要有扎实的基本功和理论知识。所以我的建议是在你的职业生涯中，要不断加强基本功（**数据结构与算法、操作系统、网络、编程语言基础、计算机组成原理**）
   我的团队曾在毫无准备的情况下，接手了一个遗留的老系统，这并不是我们所负责的业务领域，对其领域知识可以说为 0。接手第一天就有业务方找来报各种线上问题，这个系统没有任何文档交接给我们。我们拿到问题后，是这样做的：
4. 对用户请求使用 tcpdump 在服务器上进行抓包，观察里面的请求参数、请求 URL，转换成 curl 命令尝试重放请求（这里要考虑是否对业务有损）
5. 基于重放的请求，观察日志
6. 根据请求参数和日志，找到代码的入口点，仔细分析每行代码的执行结果，从而解决问题。

另外一个真实场景，我们在热点流量的冲击下，偶现单机大量的熔断异常，导致个别机器接口请求错误率非常高。最初认为是调用下游接口过慢导致的熔断，找到下游同事一起排查，在下游的视角下其接口整体耗时并不高，在我们熔断的请求中他们接口耗时很低，并返回了正常的结果，所以问题出现在我们系统内部。我们是这样排查的：

1. 观察监控系统中出问题的单机 CPU、内存、网络等负载是否有飙升
2. 观察 gc 日志，分析出现问题时间段的 gc 情况，是否 stw 时间过长
3. 上面 2 点都没有异常，那么是不是请求线程执行慢了？使用 jstack 打印线程栈，发现死锁
4. 根据死锁的栈，逐行去跟踪类库和 Jdk 源码，发现是 dnsjava 在做 dns udp 的情况下出现了死锁的 bug（这个问题不是 dnsjava，这是 jdk1.8 的一个 bug）
5. 解决办法：要么遇到了重启服务器，要么升级 jdk 版本，要么升级 glibc 版本
   上面俩个场景可以看到熟练掌握分析工具，扎实的基本功是很重要的一件事。我们遇到过很多千奇百怪的问题，人为导致的、操作系统或基础组件实现的 bug、操作系统或 jvm 参数配置不正确等等。很多问题，很少的 QPS 或数据量是无法触发的。
   这里我也列出一些常用的工具：

CPU、内存、磁盘、网络

- Linux 基本命令：top、vmstat、ps、free、netstat、lsof、strace、/proc、/sys
- 其他工具：bcc-tools、perf、sysstat、valgrind、systemtap、heap profiling、tcpdump、wireshark、Intel VTune Profiler

JVM

- jdk 自带工具：jstack、jstat、jps、jmap、jhat、HSDB、visualvm、jconsole

- 其他工具：arthas、jprofiler、mat、gcviwer
- 在线工具：gceasy、heaphero、perfma

日志分析

- Linux 基本命令：tail、grep、zgrep、zcat、awk、sed、less、more、sort、uniq
- 其他工具：ELK

如果你对性能和各种疑难杂症分析有兴趣，推荐看的资料：

书：

1. 性能之巅
2. bpf 之巅
3. 高性能超标量 CPU
4. 现代 CPU 性能分析与优化
5. 深入理解 java 虚拟机
6. 垃圾回收的算法与实现

一些优秀的网站和公众号：

8. 开发内功修炼微信公众号
9. 低并发编程公众号
10. https://easyperf.net
11. https://www.brendangregg.com
12. https://heapdump.cn