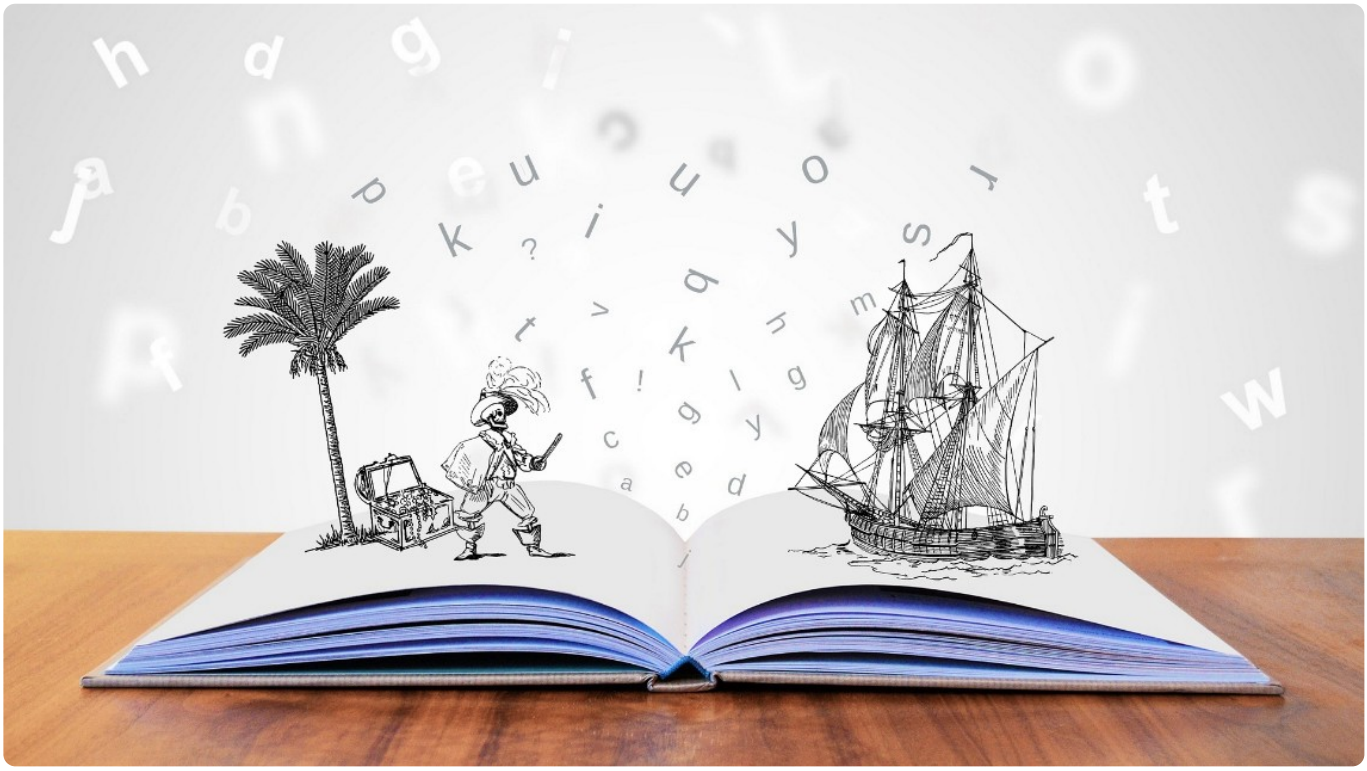
加餐3 | 定位应用问题，排错套路很重要

2020-04-09 朱晔

Java业务开发常见错误100例 进入课程 >



讲述：王少泽
时长 23:16 大小 21.31M

你好，我是朱晔。

咱们这个课程已经更新 13 讲了，感谢各位同学一直在坚持学习，并在评论区留下了很多高质量的留言。这些留言，有的是分享自己曾经踩的坑，有的是对课后思考题的详细解答，还有的是提出了非常好的问题，进一步丰富了这个课程的内容。

有同学说，这个课程的案例非常实用，都是工作中会遇到的。正如我在开篇词中所说，这个课程涉及的 100 个案例、约 130 个小坑，有 40% 来自于我经历过或者是见过的 200 线上生产事故，剩下的 60% 来自于我开发业务项目，以及日常审核别人的代码发现的问题。确实，我在整理这些案例上花费了很多精力，也特别感谢各位同学的认可，更希望你们能继续坚持学习，继续在评论区和我交流。

也有同学反馈，排查问题的思路很重要，希望自己遇到问题时，也能够从容、高效地定位到根因。因此，今天这一讲，我就与你说说我在应急排错方面积累的心得。这都是我多年担任技术负责人和架构师自己总结出来的，希望对你有所帮助。当然了，也期待你能留言与我说说，自己平时的排错套路。

在不同环境排查问题，有不同的方式

要说排查问题的思路，我们首先得明白是在什么环境排错。

如果是在自己的开发环境排查问题，那你几乎可以使用任何自己熟悉的工具来排查，甚至可以进行单步调试。只要问题能重现，排查就不会太困难，最多就是把程序调试到 JDK 或三方类库内部进行分析。

如果是在测试环境排查问题，相比开发环境少的是调试，不过你可以使用 JDK 自带的 jvisualvm 或阿里的 Arthas，附加到远程的 JVM 进程排查问题。另外，测试环境允许造数据、造压力模拟我们需要的场景，因此遇到偶发问题时，我们可以尝试去造一些场景让问题更容易出现，方便测试。

如果是在生产环境排查问题，往往比较难：一方面，生产环境权限管控严格，一般不允许调试工具从远程附加进程；另一方面，生产环境出现问题要求以恢复为先，难以留出充足的时间去慢慢排查问题。但，因为生产环境的流量真实、访问量大、网络权限管控严格、环境复杂，因此更容易出问题，也是出问题最多的环境。

接下来，我就与你详细说说，如何在生产环境排查问题吧。

生产问题的排查很大程度依赖监控

其实，排查问题就像在破案，生产环境出现问题时，因为要尽快恢复应用，就不可能保留完整现场用于排查和测试。因此，是否有充足的信息可以了解过去、还原现场就成了破案的关键。这里说的信息，主要就是日志、监控和快照。

日志就不用多说了，主要注意两点：

确保错误、异常信息可以被完整地记录到文件日志中；

确保生产上程序的日志级别是 INFO 以上。记录日志要使用合理的日志优先级，DEBUG 用于开发调试、INFO 用于重要流程信息、WARN 用于需要关注的问题、ERROR 用于

阻断流程的错误。

对于监控，在生产环境排查问题时，首先就需要开发和运维团队做好充足的监控，而且是多个层次的监控。

主机层面，对 CPU、内存、磁盘、网络等资源做监控。如果应用部署在虚拟机或 Kubernetes 集群中，那么除了对物理机做基础资源监控外，还要对虚拟机或 Pod 做同样的监控。监控层数取决于应用的部署方案，有一层 OS 就要做一层监控。

网络层面，需要监控专线带宽、交换机基本情况、网络延迟。

所有的中间件和存储都要做好监控，不仅仅是监控进程对 CPU、内存、磁盘 IO、网络使用的基本指标，更重要的是监控组件内部的一些重要指标。比如，著名的监控工具 Prometheus，就提供了大量的exporter来对接各种中间件和存储系统。

应用层面，需要监控 JVM 进程的类加载、内存、GC、线程等常见指标（比如使用Micrometer来做应用监控），此外还要确保能够收集、保存应用日志、GC 日志。

我们再来看看快照。这里的“快照”是指，应用进程在某一时刻的快照。通常情况下，我们会为生产环境的 Java 应用设置 -XX:+HeapDumpOnOutOfMemoryError 和 -XX:HeapDumpPath=...这 2 个 JVM 参数，用于在出现 OOM 时保留堆快照。这个课程中，我们也多次使用 MAT 工具来分析堆快照。

了解过去、还原现场后，接下来我们就看看定位问题的套路。

分析定位问题的套路

定位问题，首先要定位问题出在哪个层次上。比如，是 Java 应用程序自身的问题还是外部因素导致的问题。我们可以先查看程序是否有异常，异常信息一般比较具体，可以马上定位到大概的问题方向；如果是一些资源消耗型的问题可能不会有异常，我们可以通过指标监控配合显性问题点来定位。

一般情况下，程序的问题来自以下三个方面。

第一，程序发布后的 Bug，回滚后可以立即解决。这类问题的排查，可以回滚后再慢慢分析版本差异。

第二，外部因素，比如主机、中间件或数据库的问题。这类问题的排查方式，按照主机层面的问题、中间件或存储（统称组件）的问题分为两类。

主机层面的问题，可以使用工具排查：

- CPU 相关问题，可以使用 top、vmstat、pidstat、ps 等工具排查；
- 内存相关问题，可以使用 free、top、ps、vmstat、cachestat、sar 等工具排查；
- IO 相关问题，可以使用 lsof、iostat、pidstat、sar、iotop、df、du 等工具排查；
- 网络相关问题，可以使用 ifconfig、ip、nslookup、dig、ping、tcpdump、iptables 等工具排查。

组件的问题，可以从以下几个方面排查：

- 排查组件所在主机是否有问题；
- 排查组件进程基本情况，观察各种监控指标；
- 查看组件的日志输出，特别是错误日志；
- 进入组件控制台，使用一些命令查看其运作情况。

第三，因为系统资源不够造成系统假死的问题，通常需要先通过重启和扩容解决问题，之后再进行分析，不过最好能留一个节点作为现场。系统资源不够，一般体现在 CPU 使用高、内存泄漏或 OOM 的问题、IO 问题、网络相关问题这四个方面。

对于 CPU 使用高的问题，如果现场还在，具体的分析流程是：

- 首先，在 Linux 服务器上运行 top -Hp pid 命令，来查看进程中哪个线程 CPU 使用高；
- 然后，输入大写的 P 将线程按照 CPU 使用率排序，并把明显占用 CPU 的线程 ID 转换为 16 进制；
- 最后，在 jstack 命令输出的线程栈中搜索这个线程 ID，定位出问题的线程当时的调用栈。

如果没有条件直接在服务器上运行 top 命令的话，我们可以用采样的方式定位问题：间隔固定秒数（比如 10 秒）运行一次 jstack 命令，采样几次后，对比采样得出哪些线程始终处于运行状态，分析出问题的线程。

如果现场没有了，我们可以通过排除法来分析。CPU 使用高，一般是由下面的因素引起的：

突发压力。这类问题，我们可以通过应用之前的负载均衡的流量或日志量来确认，诸如 Nginx 等反向代理都会记录 URL，可以依靠代理的 Access Log 进行细化定位，也可以通过监控观察 JVM 线程数的情况。压力问题导致 CPU 使用高的情况下，如果程序的各资源使用没有明显不正常，之后可以通过压测 +Profiler（jvisualvm 就有这个功能）进一步定位热点方法；如果资源使用不正常，比如产生了几千个线程，就需要考虑调参。

GC。这种情况，我们可以通过 JVM 监控 GC 相关指标、GC Log 进行确认。如果确认是 GC 的压力，那么内存使用也很可能会不正常，需要按照内存问题分析流程做进一步分析。

程序中死循环逻辑或不正常的处理流程。这类问题，我们可以结合应用日志分析。一般情况下，应用执行过程中都会产生一些日志，可以重点关注日志量异常部分。

对于内存泄露或 OOM 的问题，最简单的分析方式，就是堆转储后使用 MAT 分析。堆转储，包含了堆现场全貌和线程栈信息，一般观察支配树图、直方图就可以马上看到占用大量内存的对象，可以快速定位到内存相关问题。这一点我们会在第 5 篇加餐中详细介绍。

需要注意的是，Java 进程对内存的使用不仅仅是堆区，还包括线程使用的内存（线程个数 * 每一个线程的线程栈）和元数据区。每一个内存区都可能产生 OOM，可以结合监控观察线程数、已加载类数量等指标分析。另外，我们需要注意看一下，JVM 参数的设置是否有明显不合理的地方，限制了资源使用。

IO 相关的问题，除非是代码问题引起的资源不释放等问题，否则通常都不是由 Java 进程内部因素引发的。

网络相关的问题，一般也是由外部因素引起的。对于连通性问题，结合异常信息通常比较容易定位；对于性能或瞬断问题，可以先尝试使用 ping 等工具简单判断，如果不行再使用 tcpdump 或 Wireshark 来分析。

分析和定位问题需要注意的九个点

有些时候，我们分析和定位问题时，会陷入误区或是找不到方向。遇到这种情况，你可以借鉴下我的九个心得。

第一，考虑“鸡”和“蛋”的问题。比如，发现业务逻辑执行很慢且线程数增多的情况时，我们需要考虑两种可能性：

一是，程序逻辑有问题或外部依赖慢，使得业务逻辑执行慢，在访问量不变的情况下需要更多的线程数来应对。比如，10TPS 的并发原先一次请求 1s 可以执行完成，10 个线程可以支撑；现在执行完成需要 10s，那就需要 100 个线程。

二是，有可能是请求量增大了，使得线程数增多，应用本身的 CPU 资源不足，再加上上下文切换问题导致处理变慢了。

出现问题的时候，我们需要结合内部表现和入口流量一起看，确认这里的“慢”到底是根因还是结果。

第二，考虑通过分类寻找规律。在定位问题没有头绪的时候，我们可以尝试总结规律。

比如，我们有 10 台应用服务器做负载均衡，出问题时可以通过日志分析是否是均匀分布的，还是问题都出现在 1 台机器。又比如，应用日志一般会记录线程名称，出问题时我们可以分析日志是否集中在某一类线程上。再比如，如果发现应用开启了大量 TCP 连接，通过 netstat 我们可以分析出主要集中连接到哪个服务。

如果能总结出规律，很可能就找到了突破点。

第三，分析问题需要根据调用拓扑来，不能想当然。比如，我们看到 Nginx 返回 502 错误，一般可以认为是下游服务的问题导致网关无法完成请求转发。对于下游服务，不能想当然就认为是我们的 Java 程序，比如在拓扑上可能 Nginx 代理的是 Kubernetes 的 Traefik Ingress，链路是 Nginx->Traefik-> 应用，如果一味排查 Java 程序的健康情况，那么始终不会找到根因。

又比如，我们虽然使用了 Spring Cloud Feign 来进行服务调用，出现连接超时也不一定就是服务端的问题，有可能是客户端通过 URL 来调用服务端，并不是通过 Eureka 的服务发

现实现的客户端负载均衡。换句话说，客户端连接的是 Nginx 代理而不是直接连接应用，客户端连接服务出现的超时，其实是 Nginx 代理宕机所致。

第四，考虑资源限制类问题。观察各种曲线指标，如果发现曲线慢慢上升然后稳定在一个水平线上，那么一般就是资源达到了限制或瓶颈。

比如，在观察网络带宽曲线的时候，如果发现带宽上升到 120MB 左右不动了，那么很可能就是打满了 1GB 的网卡或传输带宽。又比如，观察到数据库活跃连接数上升到 10 个就不动了，那么很可能是连接池打满了。观察监控一旦看到任何这样的曲线，都要引起重视。

第五，考虑资源相互影响。CPU、内存、IO 和网络，这四类资源就像人的五脏六腑，是相辅相成的，一个资源出现了明显的瓶颈，很可能会引起其他资源的连锁反应。

比如，内存泄露后对象无法回收会造成大量 Full GC，此时 CPU 会大量消耗在 GC 上从而引起 CPU 使用增加。又比如，我们经常会把数据缓存在内存队列中进行异步 IO 处理，网络或磁盘出现问题时，就很可能会引起内存的暴涨。因此，出问题的时候，我们要考虑到这一点，以避免误判。

第六，排查网络问题要考虑三个方面，到底是客户端问题，还是服务端问题，还是传输问题。比如，出现数据库访问慢的现象，可能是客户端的原因，连接池不够导致连接获取慢、GC 停顿、CPU 占满等；也可能是传输环节的问题，包括光纤、防火墙、路由表设置等问题；也可能是真正的服务端问题，需要逐一排查来进行区分。

服务端慢一般可以看到 MySQL 出慢日志，传输慢一般可以通过 ping 来简单定位，排除了这两个可能，并且仅仅是部分客户端出现访问慢的情况，就需要怀疑是客户端本身的问题。对于第三方系统、服务或存储访问出现慢的情况，不能完全假设是服务端的问题。

第七，快照类工具和趋势类工具需要结合使用。比如，jstat、top、各种监控曲线是趋势类工具，可以让我们观察各个指标的变化情况，定位大概的问题点；而 jstack 和分析堆快照的 MAT 是快照类工具，用于详细分析某一时刻应用程序某一个点的细节。

一般情况下，我们会先使用趋势类工具来总结规律，再使用快照类工具来分析问题。如果反过来可能就会误判，因为快照类工具反映的只是一个瞬间程序的情况，不能仅仅通过分析单一快照得出结论，如果缺少趋势类工具的帮助，那至少也要提取多个快照来对比。

第八，不要轻易怀疑监控。我曾看过一个空难事故的分析，飞行员在空中发现仪表显示飞机所有油箱都处于缺油的状态，他第一时间的怀疑是油表出现故障了，始终不愿意相信是真的缺油，结果飞行不久后引擎就断油熄火了。同样地，在应用出现问题时，我们会查看各种监控系统，但有些时候我们宁愿相信自己的经验，也不相信监控图表的显示。这可能会导致我们完全朝着错误的方向来排查问题。

如果你真的怀疑是监控系统有问题，可以看一下这套监控系统对于不出问题的应用显示是否正常，如果正常那就应该相信监控而不是自己的经验。

第九，如果因为监控缺失等原因无法定位到根因的话，相同问题就有再出现的风险，需要做好三项工作：

做好日志、监控和快照补漏工作，下次遇到问题时可以定位根因；

针对问题的症状做好实时报警，确保出现问题后可以第一时间发现；

考虑做一套热备的方案，出现问题后可以第一时间切换到热备系统快速解决问题，同时又可以保留老系统的现场。

重点回顾

今天，我和你总结分享了分析生产环境问题的套路。

第一，分析问题一定是需要依据的，靠猜是猜不出来的，需要提前做好基础监控的建设。监控的话，需要在基础运维层、应用层、业务层等多个层次进行。定位问题的时候，我们同样需要参照多个监控层的指标表现综合分析。

第二，定位问题要先对原因进行大致分类，比如是内部问题还是外部问题、CPU 相关问题还是内存相关问题、仅仅是 A 接口的问题还是整个应用的问题，然后再去进一步细化探索，一定是从大到小来思考问题；在追查问题遇到瓶颈的时候，我们可以先退出细节，再从大的方面捋一下涉及的点，再重新来看问题。

第三，分析问题很多时候靠的是经验，很难找到完整的方法论。遇到重大问题的时候，往往也需要根据直觉来第一时间找到最有可能的点，这里甚至有运气成分。我还和你分享了我的九条经验，建议你在平时解决问题的时候多思考、多总结，提炼出更多自己分析问题的套路和拿手工具。

最后，值得一提的是，定位到问题原因后，我们要做好记录和复盘。每一次故障和问题都是宝贵的资源，复盘不仅仅是记录问题，更重要的是改进。复盘时，我们需要做到以下四点：

记录完整的时间线、处理措施、上报流程等信息；

分析问题的根本原因；

给出短、中、长期改进方案，包括但不限于代码改动、SOP、流程，并记录跟踪每一个方案进行闭环；

定期组织团队回顾过去的故障。

思考与讨论

1. 如果你现在打开一个 App 后发现首页展示了一片空白，那这到底是客户端兼容性的问题，还是服务端的问题呢？如果是服务端的问题，又如何进一步细化定位呢？你有什么分析思路吗？
2. 对于分析定位问题，你会做哪些监控或是使用哪些工具呢？

你有没有遇到过什么花了很长时间才定位到的，或是让你印象深刻的问题或事故呢？我是朱晔，欢迎在评论区与我留言分享你的想法，也欢迎你把这篇文章分享给你的朋友或同事，一起交流。

上一篇　加餐2 | 带你吃透课程中Java 8的那些重要知识点（下）

精选留言 (8)

写留言

hellojd 置顶
2020-04-09

我们线上k8s管理服务，有时候oom，服务重启由k8s触发的，这将导致设置的生成dump文件无效。有好的思路吗？

展开

作者回复: 1、需要明白，xmx设置的堆只是java进程使用内存的一部分 https://stackoverflow.com/questions/53451103/java-using-much-more-memory-than-heap-size-or-size-correctly-docker-memory-limi

所以你需要通过监控排查到底哪部分内存超限，但是heap的oom dump肯定是需要做的，并且配置-XX:NativeMemoryTracking=detail -XX:+UnlockDiagnosticVMOptions -XX:+PrintNMTStatistics，需要的时候可以通过jcmd <pid> VM.native_memory summary/detail排查

2、在排查清问题之前可以适当放开k8s的limit，以便你可以观察到内存增长的区域，方便排查问题

3、可以和运维沟通一下，在oom killed的时候（oom killed应该会让pod状态变为unhealth，这个时候可以触发hook）能否做一下heapdump、jstack等，数据不要保存在容器里

当然，像Darren的回复，在死之前做几次dump也是可以的

2

Darren
2020-04-09

首先先试着回答下@hellojd童鞋的问题，不一定对，仅供参考：k8s应该在cpu或者内存的使用率上做报警，大于90%的时候可以dump和jstack一次，甚至jstat也可以做，然后95%的时候也同样执行一次，
甚至98或者99的时候也可以做一次，这样不仅可以保留现场，同时还可以对比。可以更好的排查问题。...

展开

作者回复: 感谢分享

4

2020-04-09

本文理解：

应对出错其实主要就是三个阶段。

1. before：保证留有合理的日志...

展开

作者回复: 总结的很好

2

2020-04-09

如果你现在打开一个 App 后发现首页展示了一片空白，那这到底是客户端兼容性的问题，还是服务端的问题呢？如果是服务端的问题，又如何进一步细化定位呢？你有什么分析思路吗？

首先，切换设备，或者模拟请求，查看服务是否正常访问。以排除客户端问题。...

展开

作者回复:

1

2020-04-09

真不敢相信，如此高质量的内容，竟然只是选学的不定期加餐！！！

作者回复: ：）

1

hanazawakana
2020-04-12

问题1，首先看下客户端的崩溃监控，再看下是不是只有这个机器有问题，如果不同机型也有问题，再看下是不是某个机器底层操作系统的问题，比如Android O，再看下是不是某个厂商操作系统的问题，如果是普遍存在的，考虑服务端问题，看下对应接口的日志有没有报错信息，如果没有日志，再分析下是不是nginx或者网络的问题

展开


yinchi_516564
2020-04-09

分享几个遇到的运维犯的错误：
1、现象：同一个请求有时候能查出结果，有时候返回为空
　原因：经排查是运维把应急时候用的服务器(直接返回200)误添加在了nginx代理中
2、现象：同一笔请求有时候很快，有时候超时60s
　原因：运维路由规则配错导致 到mongodb 的去程和回程路径不一导致原来的mongo...

展开

作者回复:


Husiun
2020-04-09

高质量内容，期待老师的更新；个人在实际应用中还主要是top定位，课后问题1个人没有相关经验，我的思路是先定位客户端问题再一步步排查到服务端，之后再top定位具体服务问题