基于eBPF的程序摄像头
——Trace-Profiling的设想

主讲人：茛程

2022-11-12

芨程的自我介绍

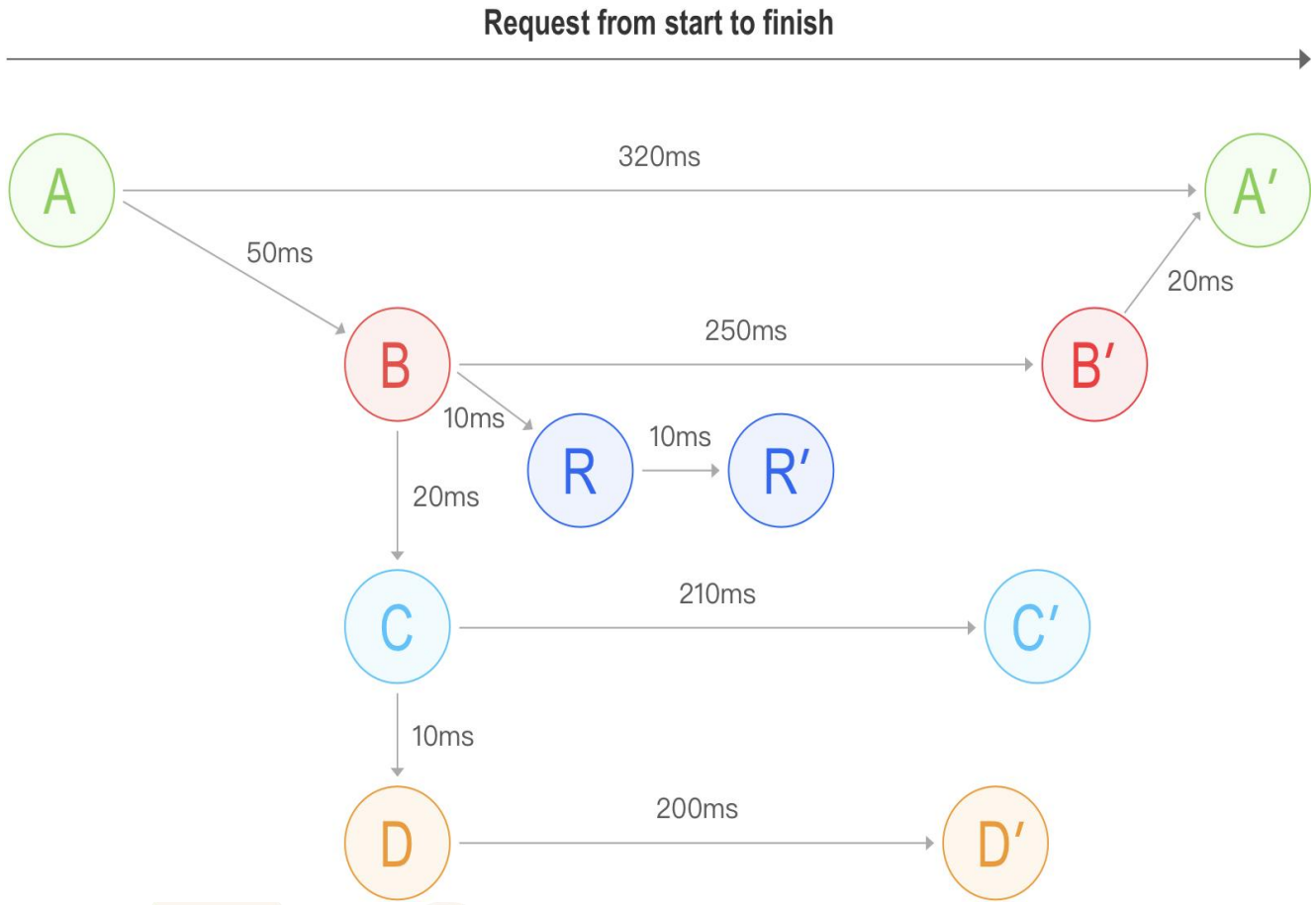
- 2010年浙江大学SEL实验室带队老师
- 2016年谐云科技联合创始人兼CTO
- 2022年创建Kindling开源项目

目录

01

云原生环境可观测性挑战

当前可观测性手段的不足



B'C'D'
为什么执行
时间都达到了
200ms以上？

log？ ？ ？
Trace？ ？ ？
Metric？ ？ ？

可观测性挑战：节点异常根因定位困难

Challenges across maturity levels

Key challenges and concerns associated with observability have shifted since our 2021 survey. This year respondents are:

- **Struggling with the ability to correlate data from multiple sources in a timely fashion (according to 29%, up from 23% a year ago).**
- **Collecting an amount of data that exceeds human capacity to digest (27%, up from 21%).**
- **Experiencing a lack of visibility across distributed environments (26%, up from 20%).**
- **Using legacy tools that lack visibility to cloud-native environments (26%, flat year-over-year, but unseated as the most frequently cited challenge).**

Observability leaders' top concerns are a little different. The inhuman amount of data tops their list, followed by "observability tools lack visibility into legacy application environments" (which placed seventh overall), followed by the lack of visibility across distributed environments, the struggle to correlate data, and the legacy tool challenge.

- 系统层指标无法感知业务健康程度
- Metric、Logging、Trace融合关联有难度

当前可观测性提供都是程序执行留痕

log与Trace能够部分解决用户代码层面的问题

metric可以覆盖从用户代码到系统调用库

用户代码

类库代码

JVM代码

glibc库

系统调用

类jstack的线程剖析能够部分发现用户代码与类库代码现场

当前可观测性工具没有办法去发现这层的问题

这些代码为什么慢？为什么之前不慢？

一般没有问题，有问题多半也是使用不当造成的

02

老刑侦的破案经验与光学摄像头

纸上得来终觉浅，绝知此事要躬行。

足迹衍生出来的知识

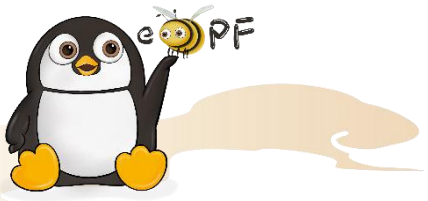
- 根据经验：大部分人的脚印和身高的比大约是1：7

- 夏秋之夜，上半夜留下的脚印，上面往往有昆虫爬过的痕迹。下半夜留下的脚印，由于地面比较潮，泥土易碎裂，脚印的边缘往往不很清楚。

- 少年罪犯步子短，脚印瘦小，脚印之间的距离往往不规则，步行的路线往往弯曲。青年罪犯往往脚印大，步子跨得大，脚印之间的距离均匀，走直线。 中年罪犯走路稳、慢，脚印间的距离变短。老年罪犯的步幅变得更短，足迹中脚后跟的压力比脚掌重。

- 脚印前浅后深一般前者多是运动员、工人等体力劳动者。

  脚印前深后浅政府公务员、律师、教师等职业



打怪升级是专家涨经验成长的必经之路

中国开始推广光学摄像头之后的新闻标题

“平安上海”魅力尽显——盗窃案“断崖式”下降
破案率达历史最高水平

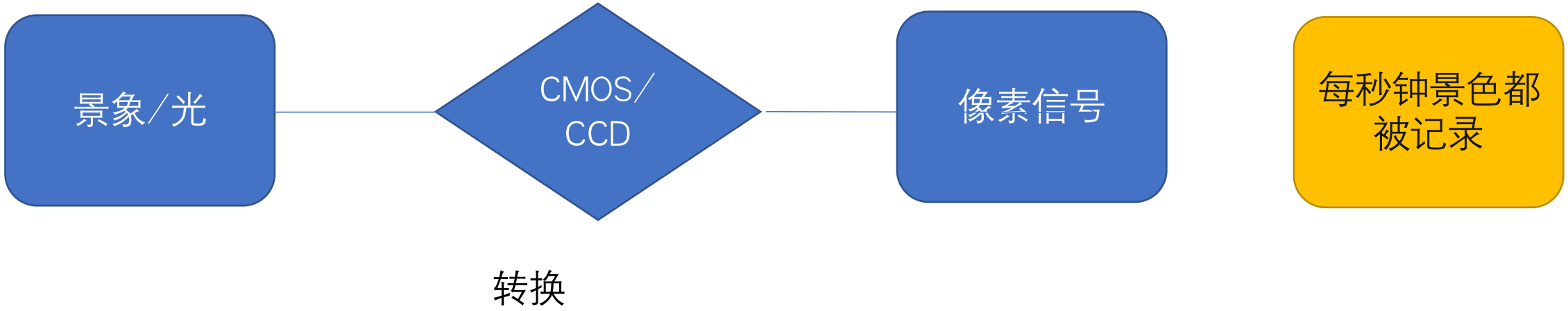
时间：2019-11-27　字体：大 中 小

03

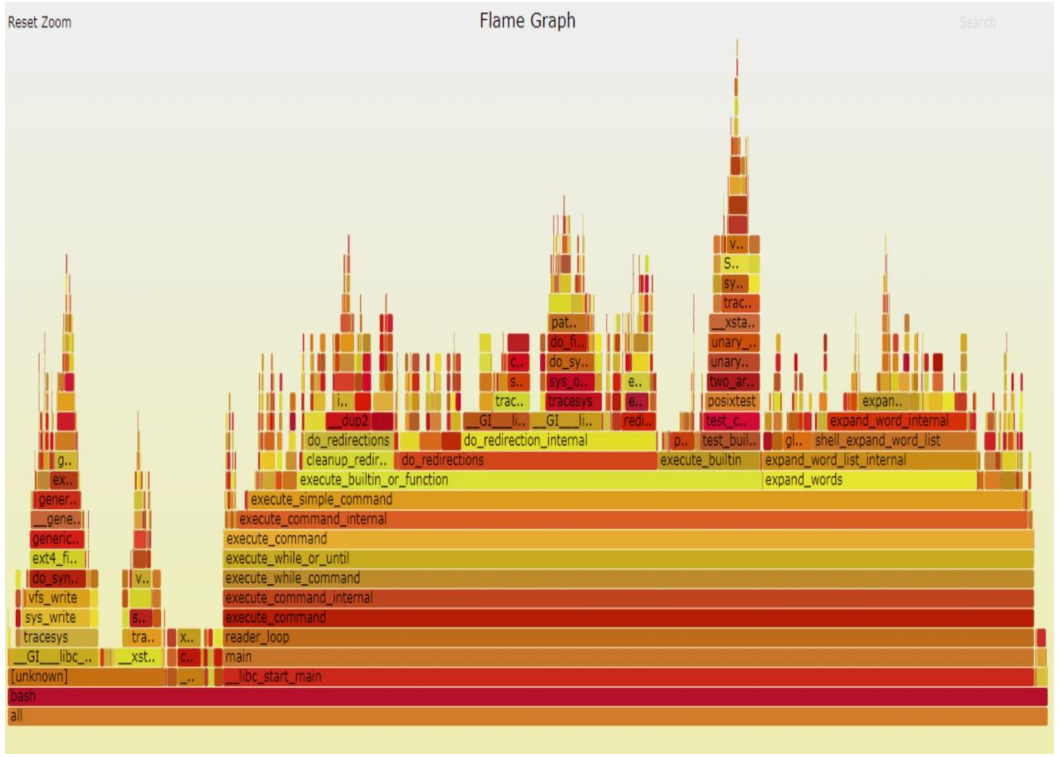
基于eBPF的程序摄像头构想

纸上得来终觉浅，绝知此事要躬行。

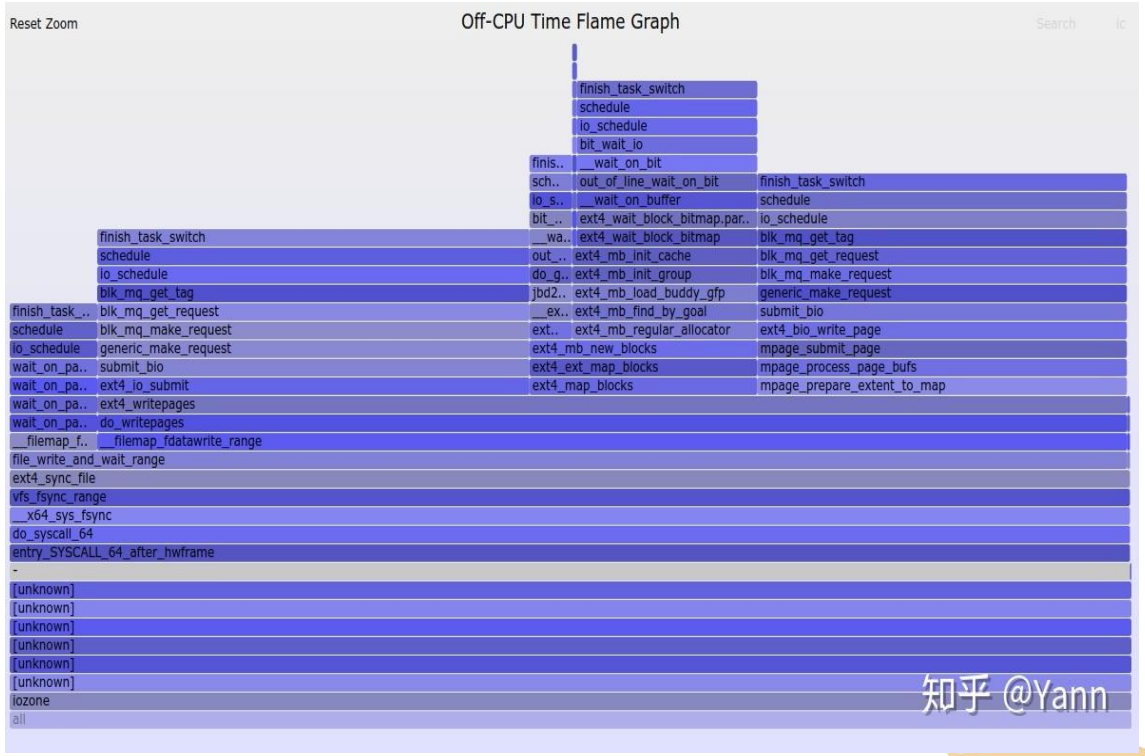
光学摄像头的工作与程序摄像头类比



业界已经有的基础

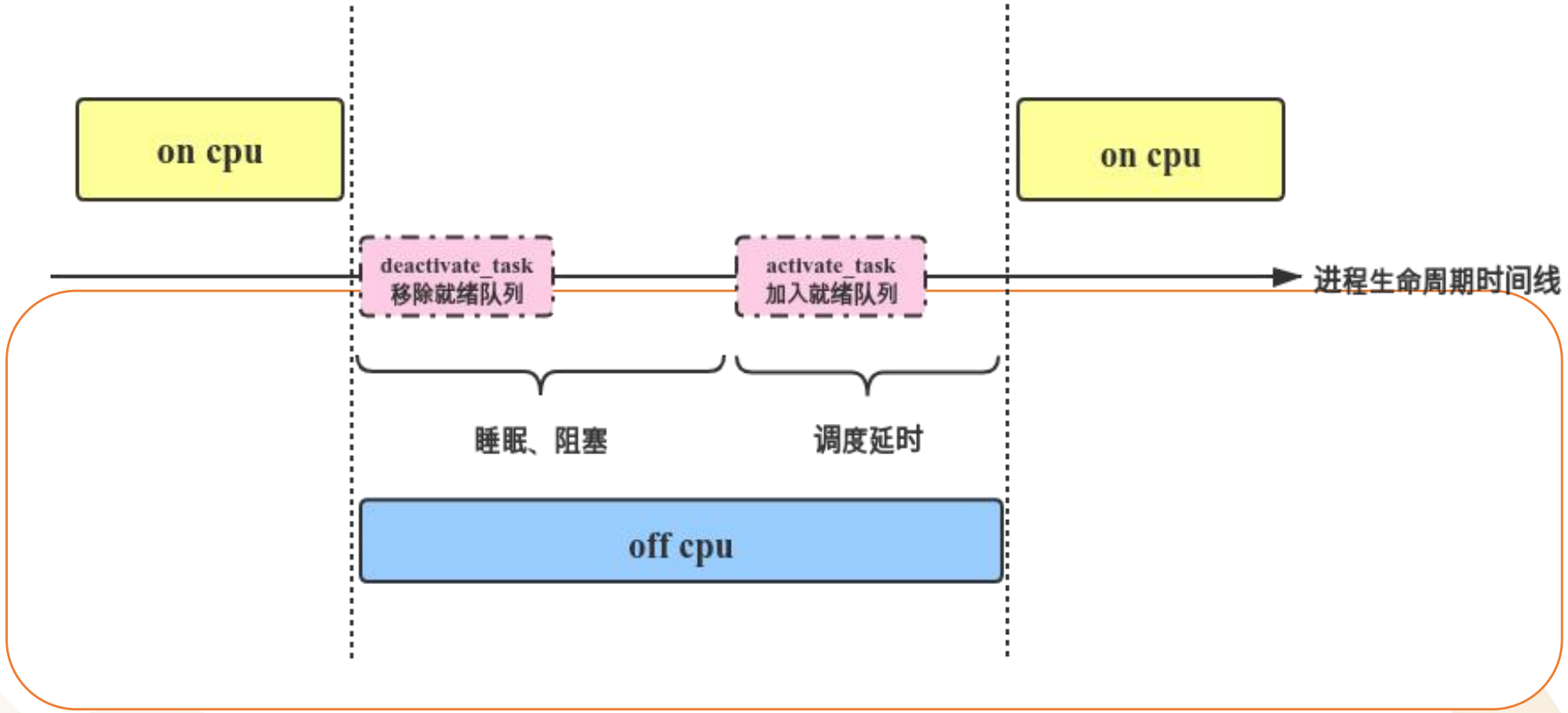


CPU火焰图



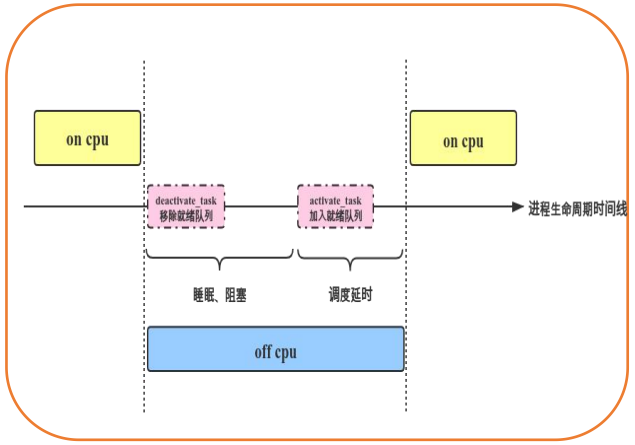
OffCPU火焰图

按照程序执行过程对齐OnCPU与OffCPU到线程粒度

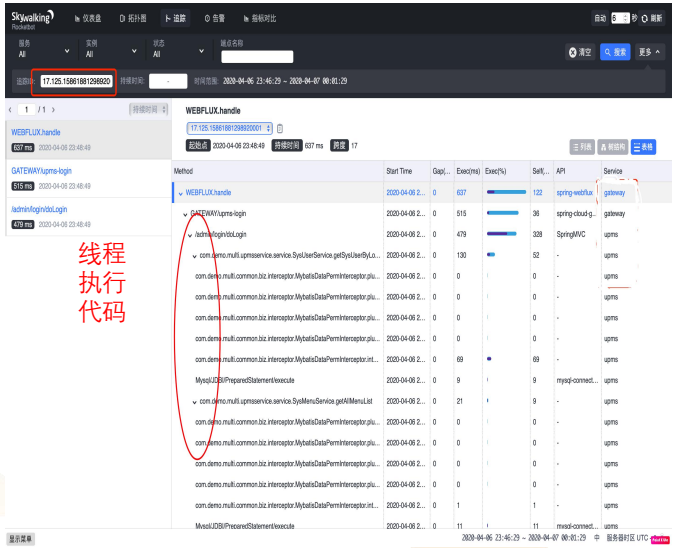


程序在操作系统上执行过程

程序摄像头的放大清晰效果——关联Trace、metric、log



onCPU与Offcpu都是线程执行情况的体现

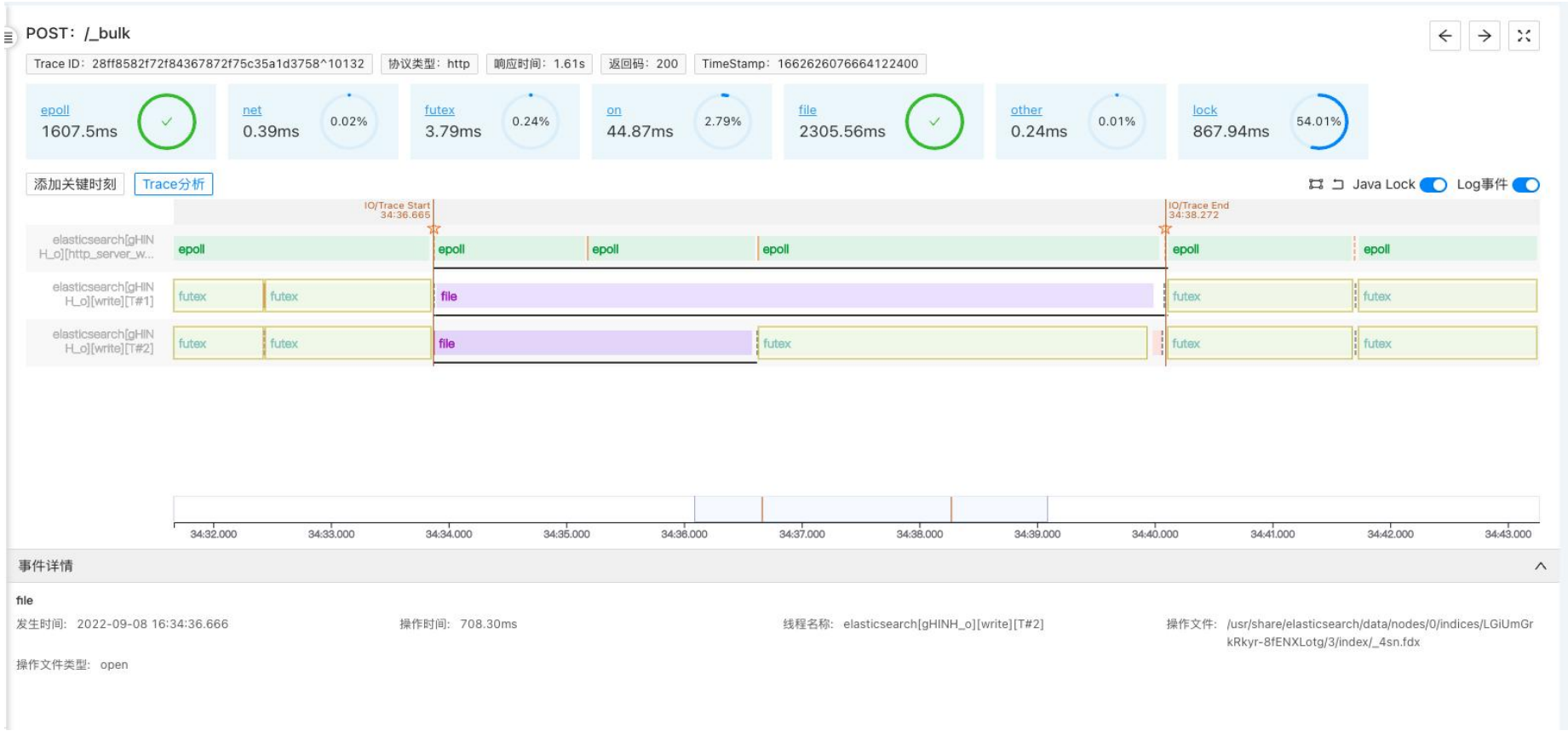


OnCPU与OffCPU以线程为执行单位

所有的日志输出都可以归类到某个线程输出

Trace执行以线程为执行单位

将OnCPU与OffCPU执行过程按照线程与Trace关联——形成程序摄像头



通过程序摄像头观察ElasticSearch执行Bulk插入的情况

程序摄像头在可观测性的生态中的位置——关联tracing、metric、log

tracing与log关联

tracing：代码维度

log：代码维度

traceid输出到日志当中即能很好的关联tracing与log

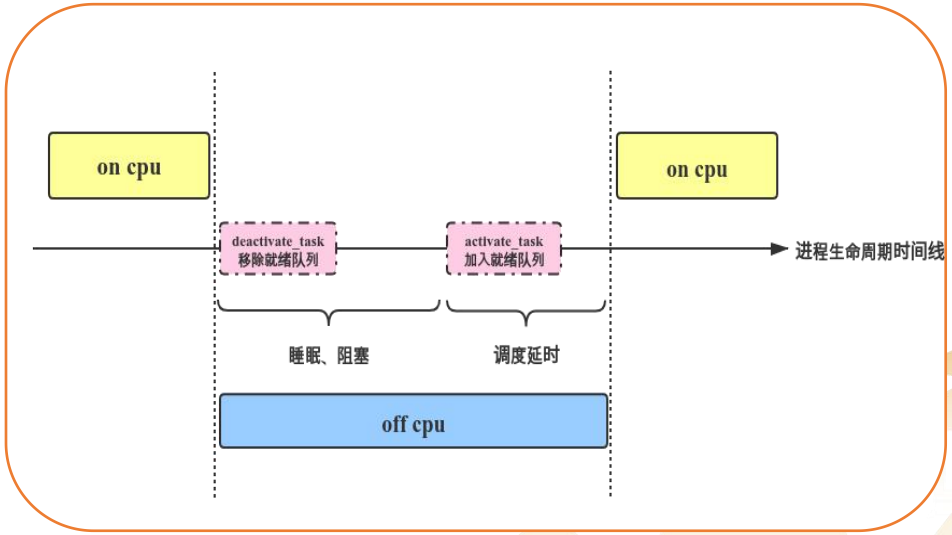
tracing与metric关联

tracing：代码维度

metric：资源维度为主

除了时间关联，没有关联的key

Kindling解法：利用eBPF将线程执行代码过程转换成资源消耗过程，然后每个环节在时间段关联相关metric



程序摄像头标准的技术术语——Trace Profiling

- How all threads were executed is recorded and can be replayed.
- The exact thread which executed the trace span is highlighted.
- The logs printed by each thread are collected and correlated to the relative thread with its timestamp.
- The code execution flame graph is correlated to the time series where the CPU is busy.
- The network-related metrics are correlated to the time series where the network syscalls are executing.
- The file-related metrics are correlated to the time series where the file syscalls are executing.

04

eBPF程序摄像头预期效果——使用场景介绍

纸上得来终觉浅，绝知此事要躬行。

程序摄像头在可观测性领域使用场景

1、TOMCAT接受请求过程

2、程序执行过程中锁占比时间比较长

2、并发过高，线程池不够用

3、程序由于Java GC导致执行时间较长

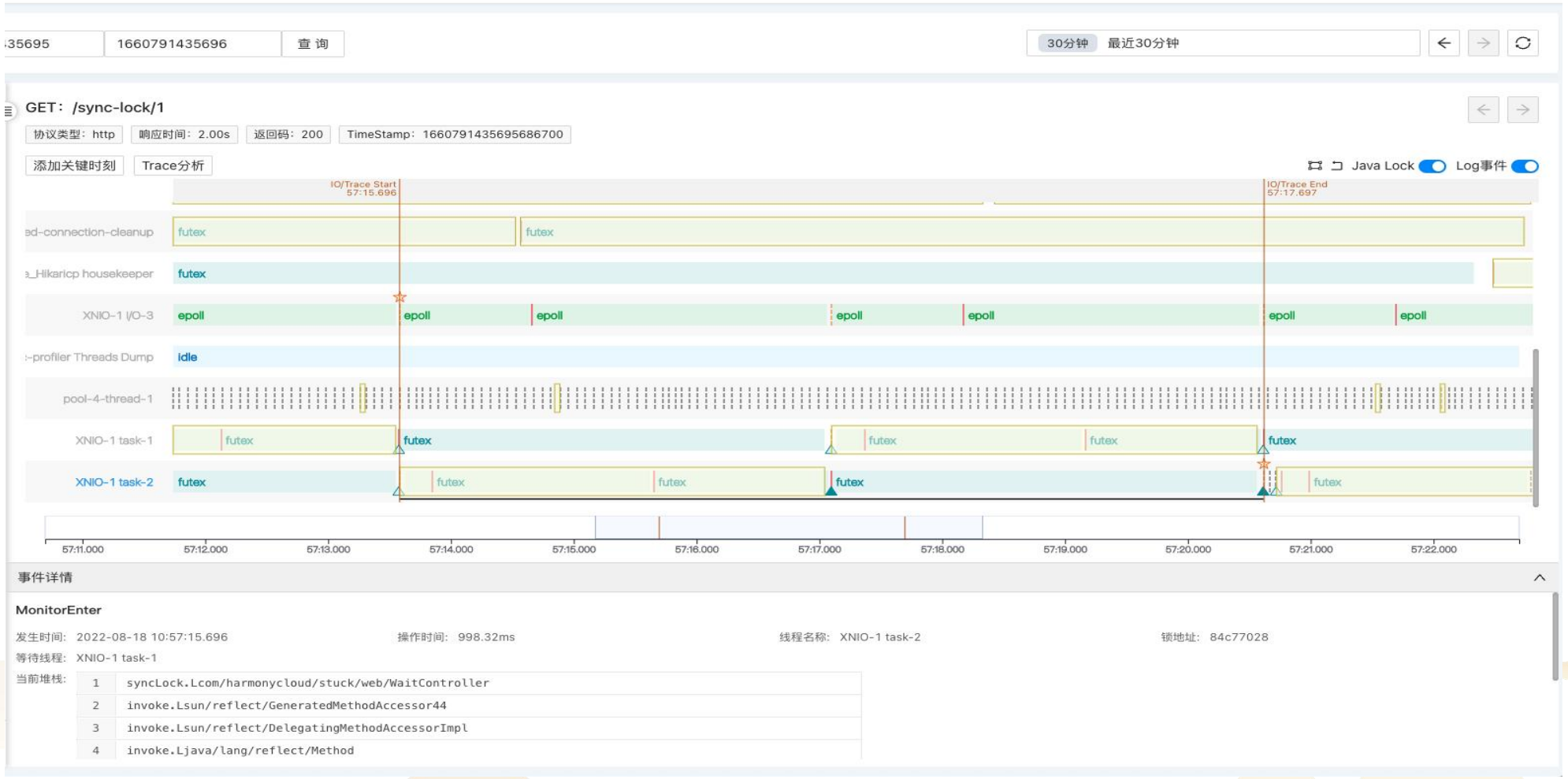
4、程序自身执行了CPU开销非常大的代码

5、网络依赖执行较慢

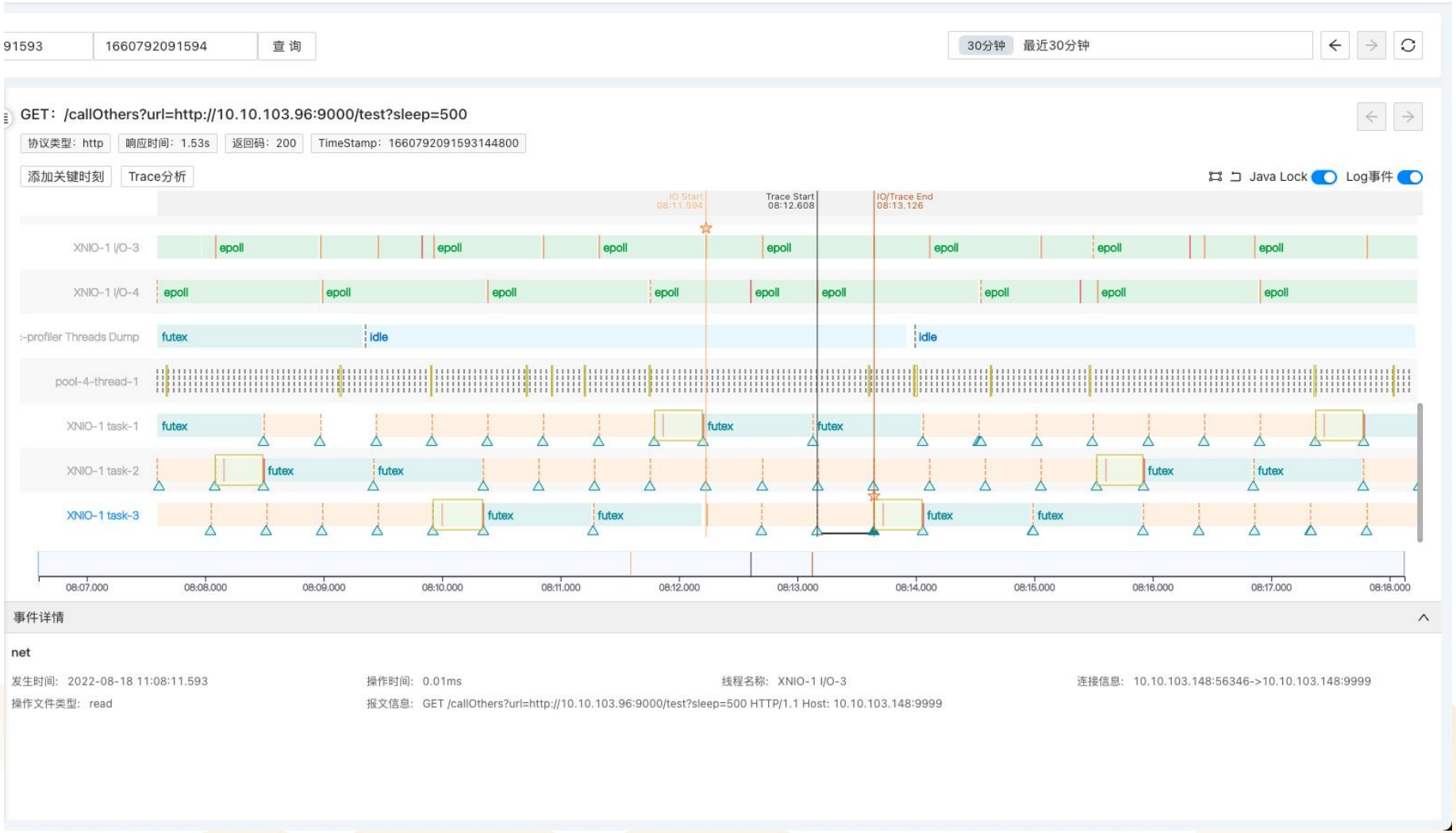
想了解程序异常退出最后现场吗？哪些线程在干什么？
想了解程序执行时，依赖资源（网络、存储）是否正常响应吗？
想知道线上故障，是否是依赖库有bug引起的吗？
想知道机器IO遇到瓶颈点，程序在干什么导致的吗？
想了解一次请求慢的过程吗？如何确认CPU资源在程序执行过程中产生了竞争？
想了解程序hang住时，线程分别在做啥吗？
想了解程序发生锁时相关线程的日志信息吗？锁持有的堆栈和被哪个线程长期占有？
想了解程序CPU突然飙高的原因吗？
想了解用户请求是否受到GC的影响吗？
想了解高并发访问的情况下，用户请求是否有排队吗？增加线程池是否能够解决问题？
当用户请求实际执行结果与预期不符合时，想了解实际执行请求的完整日志信息吗？

TOMCAT接受请求的过程

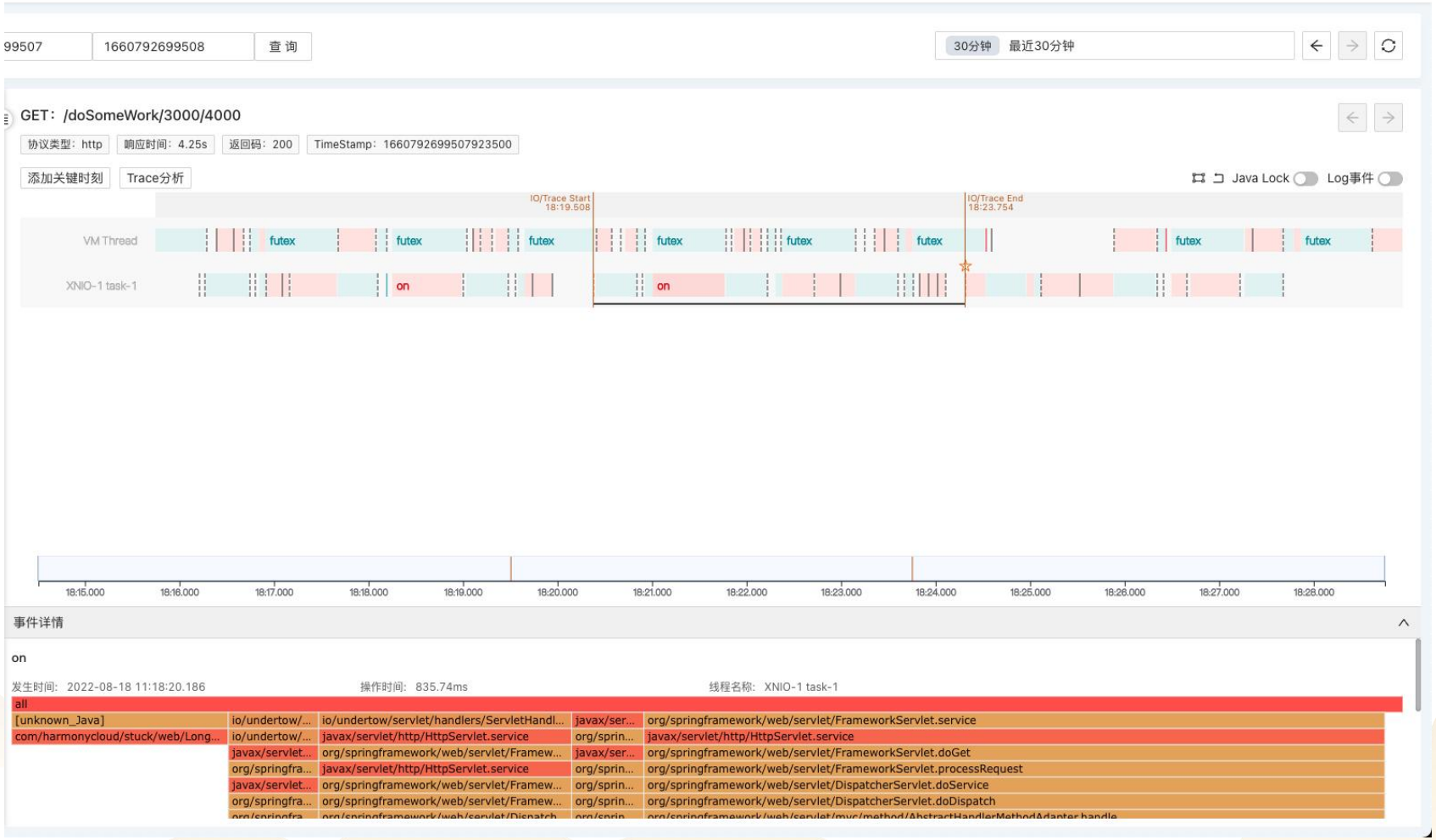
程序执行过程中锁占比时间比较长



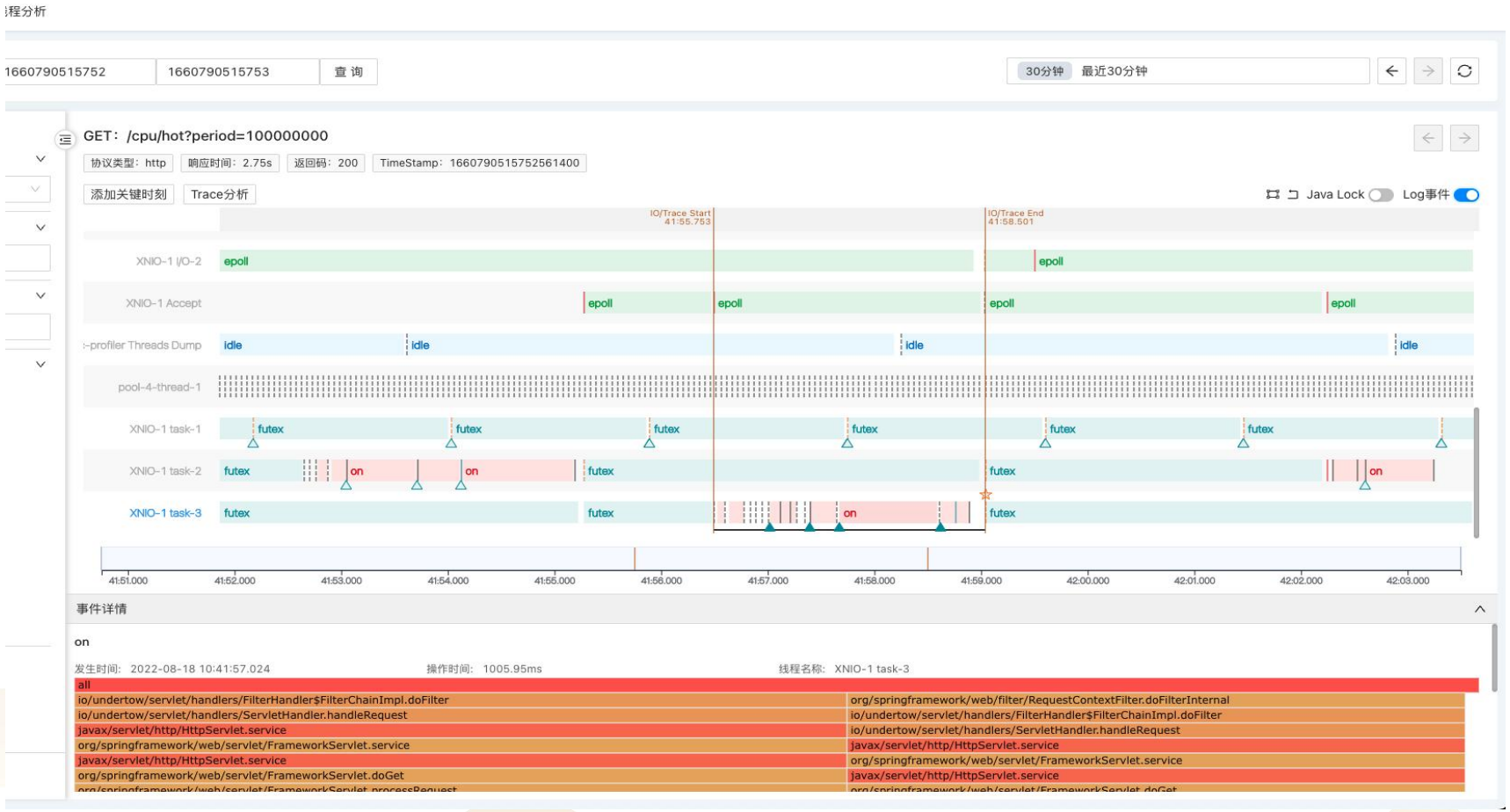
程序并发过高，线程池不够用

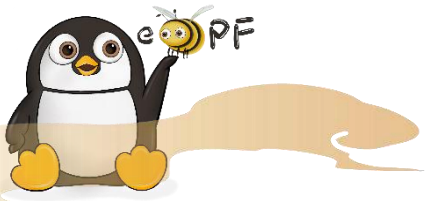


Java程序由于GC的原因被暂停执行



Java程序自身代码消耗较多的CPU





Thanks~!