AI 时代下的服务器固件

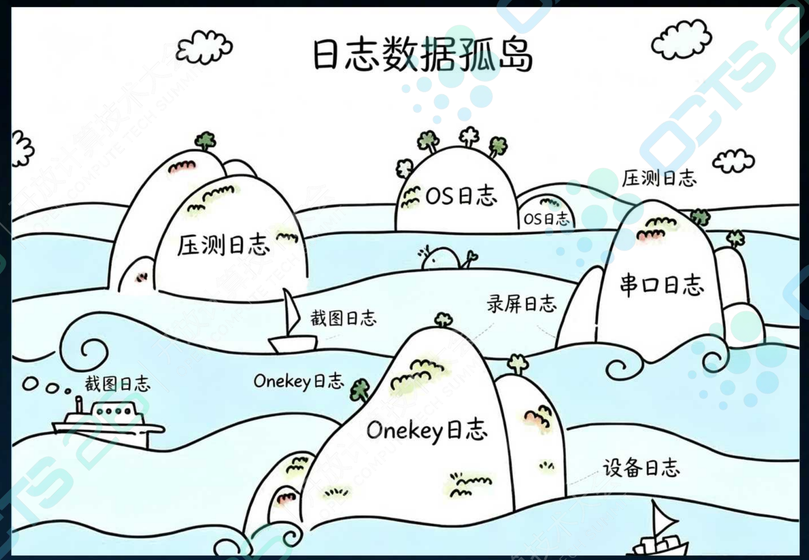
问题定位新范式

方海涛 · 联想 · 软件开发工程师

2026 开放计算技术大会

目录

问题定位为何如此艰难？—— 日志与定位的困境



图示：复杂系统中数据孤岛导致的信息割裂现状，使得故障排查如同在碎片化的信息海洋中捞针。

01 日志侧 · 信息的迷雾

信息孤岛与解析耗时：一次排查涉及超10个异构数据源，格式混乱，人工解析平均耗时**35分钟+**，效率极低。

时序断裂与因果难寻：缺乏统一时间轴，关键事件无法精确对齐，导致故障定位周期常**跨天**。

02 定位侧 · 经验的壁垒

依赖直觉与新手难破：面对海量日志，排查高度依赖专家“直觉”，复杂问题定位耗时以**小时级**计算，新人难以独立上手。

多因杂糅与根因掩盖：超50%的复杂故障存在多根因耦合，高优先级告警极易掩盖真正根因，导致反复排查。

问题定位为何如此艰难？—— 响应、经验与协同的瓶颈

03 响应侧：被动的代价

⚠ 无效告警刷屏

无效告警占比高达60%以上，工程师陷入机械性“降噪”，关键故障信号被海量噪音淹没。

偶现问题难复现

偶现BUG复现周期漫长，现场环境复杂且无法随意调试，缺乏上下文数据支撑。

04 知识侧：经验的流失

高度依赖专家经验

复杂系统排障极度依赖少数核心专家，新人培养周期长达数月。

经验难以沉淀复用

排障思路与解决方案散落在个人笔记或聊天记录中，未形成结构化知识库，导致同类问题重复踩坑。

05 协同侧：沟通的壁垒

跨部门协同低效

故障排查涉及多团队协作，信息同步存在壁垒，导致平均修复时间（MTTR）倍增。

从告警到根因：一条标准化的问题定位流水线

1 故障触发

sensor / SEL / LED

2 现场抓取

串口 / minicom

3 远程拉取

Jira / OneKey

4 日志分析

log-analyzer

5 崩溃定位

coredump + gdb

6 复盘沉淀

lesson-learned

核心设计思想：原子化与标准化

将复杂的问题定位工单拆解为一系列“输入明确、输出明确、副作用小”的原子技能模块。

Skill 01. jira-operator 工单系统的连接器

🔐功能定位：操作 Jira工单系统，覆盖连接、搜索、创建、流转及附件下载管理，提供统一CLI入口。

Skill 02. serial-console 串口调试控制台

📡 功能定位：统一封装 ssh 跳板、tmux 会话及 minicom 配置，实现“一键直达”的串口调试

📥 输入：连接拓扑 / 凭据 → 📤 输出：可交互串口会话 + 启动 / 内核日志

Skill 03. prj-src-mapper 项目源码映射器

功能定位：项目目录统一解析，给所有 BMC 分析技能提供唯一可信的源码根目录。

核心价值：源码版本管理与路径映射，保障全链路数据的准确性与一致性。

Skill 04. log-analyzer 海量日志的导航仪

📊功能定位：BMC 一键日志分析，提取版本 / 资产 / 严重异常，按问题定向收敛证据。

1

解压入仓

建独立工作区
解压日志包
规整文件结构

2

模板匹配

内置问题特征库
错误关键词对齐
锁定分析方向

3

聚焦分析

整机健康巡检
或错误关键词筛选
深入定向排查

4

智能报告

自动输出结构化报告
标注证据行号
+可信度评分

💡设计哲学：拒绝「大海捞针」，追求「少而准地回答具体问题」

Skill 05. coredump-analyzer 崩溃日志分析器

💥功能定位：core dump 自动化分析，融合 rootfs + debug 符号，交叉跑 GDB 和 eu-stack

💡重点日志模块如：vuart / sol / Perfetto 可提炼为sub skill

日志分析 + 崩溃定位

log-analyzer

一键日志体检

1. 解压日志
2. 匹配模板
3. 输出报告

coredump-analyzer

崩溃栈反汇编

1. 准备 squashfs sysroot
2. gdb-multiarch + eu-stack
3. 根因 + 修复建议

两路输出：结构化 Markdown 报告（崩溃点 / 调用栈 / 根因）

Skill 06. bug-analyzer-jira 离线工单分析引擎

📥 功能定位：离线工单分析，以 Jira 单据 / 评论 / 附件日志为入口，结合本地源码生成根因报告

Skill 07. bug-analyzer-online 在线机器实时诊断

📡 功能定位：在线机器分析，连接 BMC，实时抓 OneKey 日志后做根因定位

离线 / 在线 两轨：抓日志


一台机器出现异常
离线轨（Jira 工单）
bug-analyzer-jira
自动下载 → 解压 → 解析 → 出报告
在线轨（机器还活着）
bug-analyzer-online
读 FRU → 抓 OneKey → 抓 EventLog
共同前置：prj-src-mapper
唯一源码路径，杜绝项目错位

Skill 08. cq-lesson-learned 经验的炼金术士

功能定位：工程复盘沉淀，从 LLM对话上下文中提炼经验和教训

1. **识别触发**
   AGENTS.md + skill 列表叠加提示
2. **加载 SKILL.md**
   Core Protocol 进上下文
3. **执行协议**
   query / apply / 搜同类 KU（集体智库）
4. **写入知识库**
   propose → SQLite

直击痛点：拒绝“无效复盘”

解决传统复盘会议效率低、结论模糊、经验无法有效转化为团队技术资产的问题，实现经验的可追溯与复用。

CQ：基于 MCP 的共享记忆库

1 Agent 层： 任何兼容 MCP 的 Agent 都可接入 · 当前已集成 OpenCode + cq-toolkit

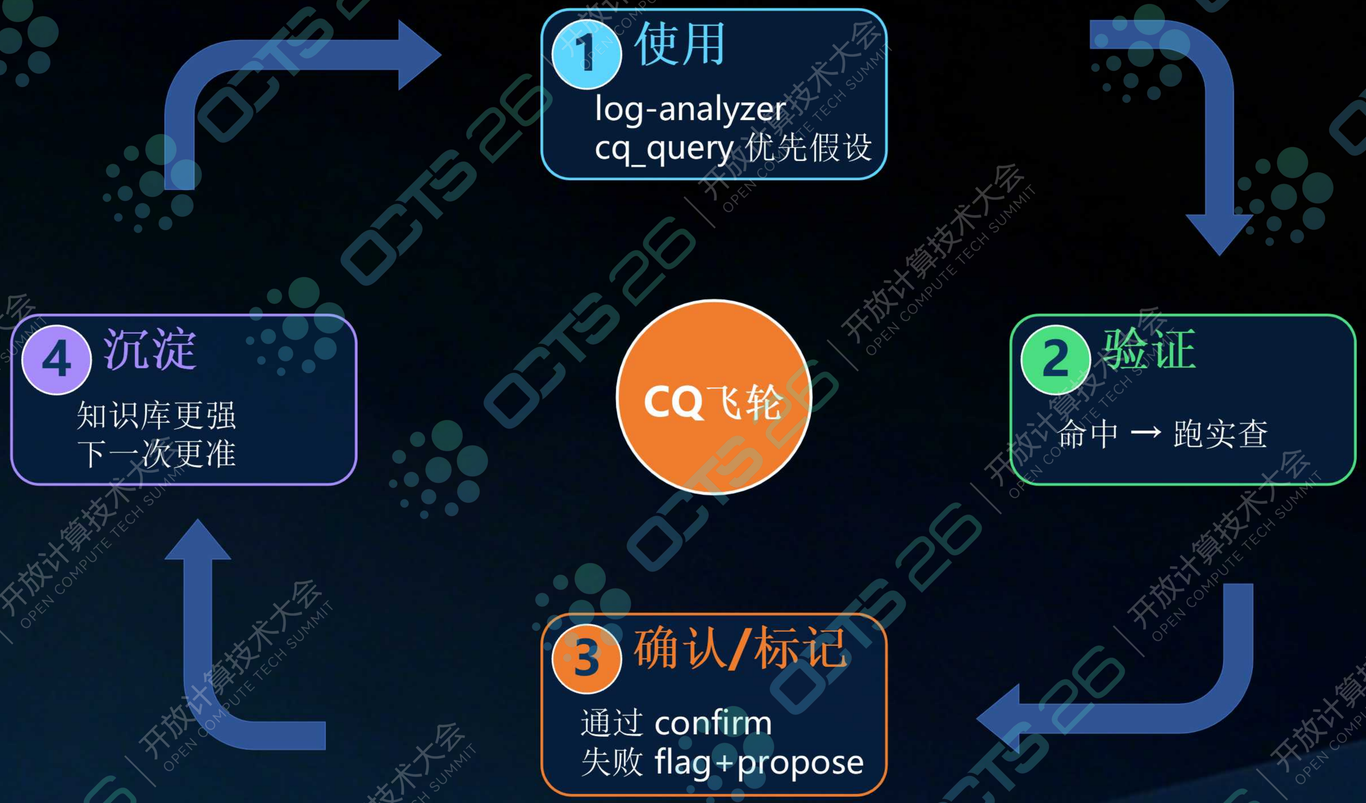
2 MCP 协议： Anthropic 提出的开放标准 · 标准化对接 Agent 与外部知识服务

3 CQ Server： SQLite + GitLab SQL 快照后端

query	confirm	flag	propose	status
检索	+0.1 置信	−0.15 失效	需 /cq:reflect	统计

💡群体确认过滤过时知识 • 初始置信度 0.5 • 这是 CQ 与传统 Wiki 最本质的区别

CQ 接进 log-analyzer 工作流



8 个 Skill + 1 MCP 分层架构

入口层	bug-analyzer-jira	bug-analyzer-online
分析层	log-analyzer	coredump-analyzer
基础工具层	jira-operator	serial-concole	prj-src-mapper
经验沉淀层	cq-lesson-learn	

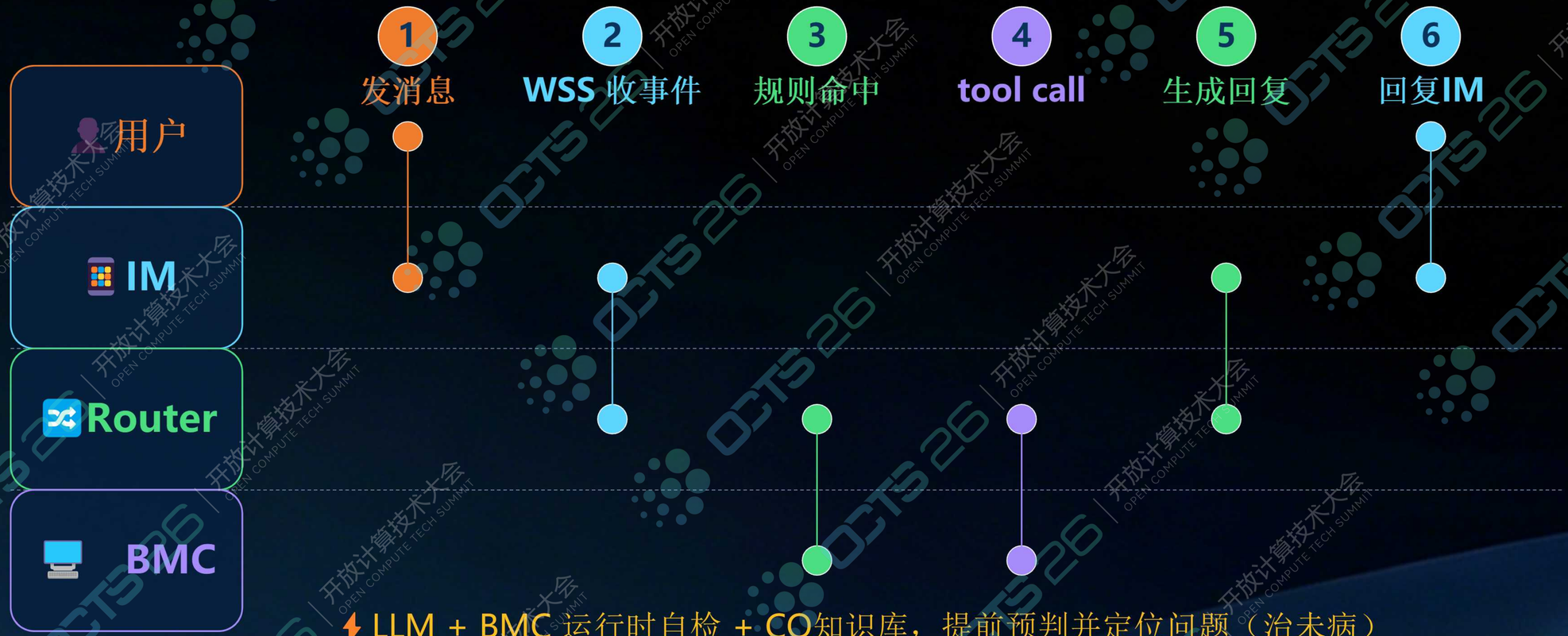
💡 prj-src-mapper · cq-lesson-learn（任意层均可调用）

如果把日志分析和**CQ**知识库

装进 BMC 内部？

→ 下一步：bmc-claw

端到端闭环：IM → BMC 自检



LLM + BMC 运行时自检 + CQ知识库，提前预判并定位问题（治未病）

如果把各个智能体

组成Agent网络？

→ 下一步：A2A协同

A2A协同·全链路故障定位新范式

核心价值·智能并行闭环

跨域排障零门槛

打破技术壁垒，普通运维可完成复杂全栈定位

效率量级跃升

并行排查替代串行分析，定位从「小时级」到「分钟级」

根因精准无误判

多源交叉印证，消除信息盲区，降低误判率

THANKS

THANKS

THANKS

THANKS