

QCon  
全球软件开发大会

# AI x 大前端性能稳定性： 快手亿级DAU下的智能诊断实践

李锐

快手移动端稳定性负责人

📍 北京

**QCon**

全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

📍 北京

**AiCon**

全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

📍 上海

**QCon**

全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月

5月

6月

8月

10月

12月

📍 上海

**AiCon**

全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

📍 深圳

**AiCon**

全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

📍 北京

**AiCon**

全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

# 极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播



参会咨询



查看会议

## 你焦虑吗

AI coding + AI debug 完美闭环，程序员会被替代？



Cursor

100亿美金



Anthropic

1830亿美金



ACM Final

满分夺冠



IMO

金牌水平

## ● 性能稳定性视角

150+

报警事件（年）

2000+

必解决问题（年）

1000万

工程庞大

AI 能释放多少生产力？



## ● 个人介绍



### 李锐

- 快手 / 移动端稳定性负责人
- 2019 年加入快手，曾就职阿里、FaceUnity、华为
- 主导快手移动端稳定性「监控拦截」「排障止损」「应急处置」多领域建设
- 喜欢研究操作系统、编译器、虚拟机等底层技术
- KOOM 开源项目作者

# 目录

01

快手性能稳定性背景（发展历程 | 关键挑战）

02

AI x 性能稳定性介绍（分析判断 | 重点建设 | 如何实施）

03

实践：AI 辅助根因排障（排障本质 | Holmes | 火焰图）

04

实践：AI 加速应急处置（止损工具 | Ekko | 效果分析）

05

总结展望（Agent 开发感悟 | 人机协同 | 认知提升）

01

# 快手性能稳定性背景

发展历程 | 关键挑战

## 快手稳定性发展历程





# 背景

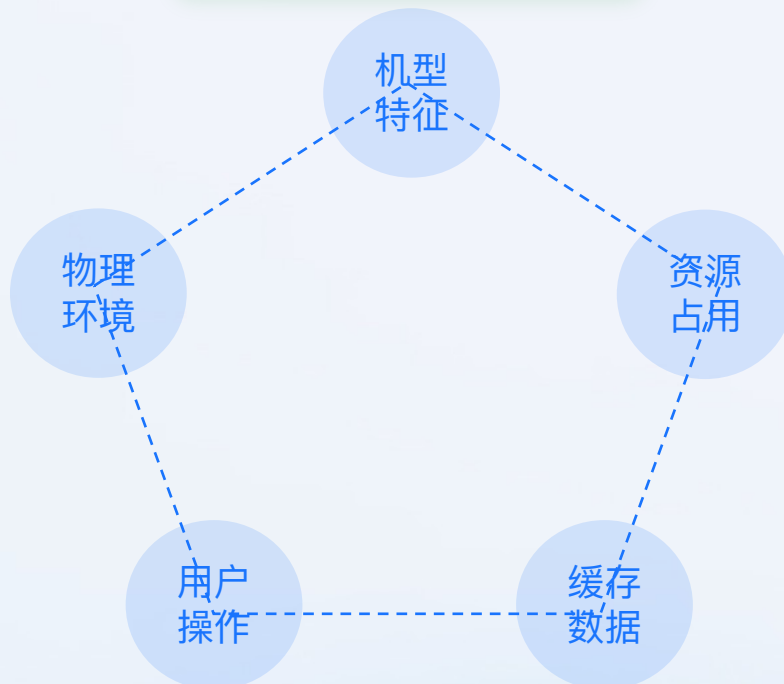
## 2025，性能稳定性仍是大前端的关键挑战

### 业务复杂



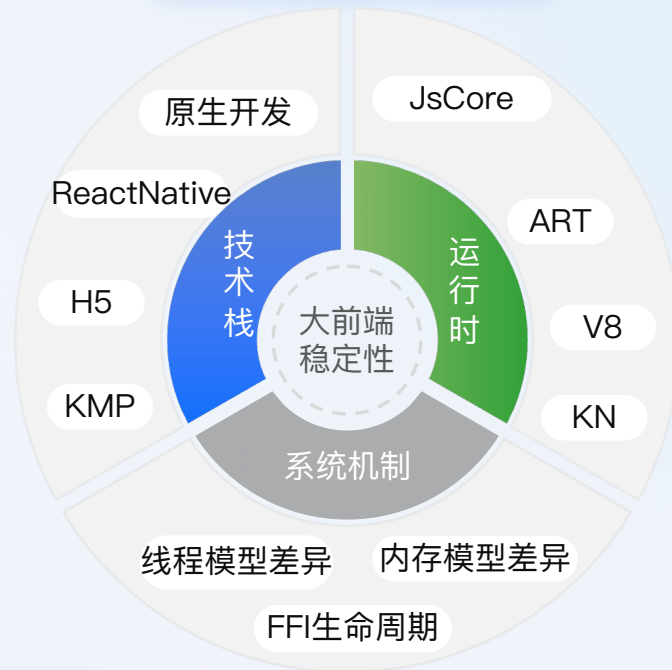
跨团队协作，迭代速度

### 终端环境复杂



执行不可控，不可穷举

### 技术栈复杂



触及底层，疑难杂症

02

# AI x 性能稳定性介绍

分析判断 | 重点建设 | 如何实施

# AI 带来的机遇 – 分析判断



判断

AI x 性能稳定性：团队产出放大器，反哺性能稳定性提升



# AI 带来的机遇 – 稳定性体系化建设现状

## 技术体系

### 拦截左移



### 监控排障

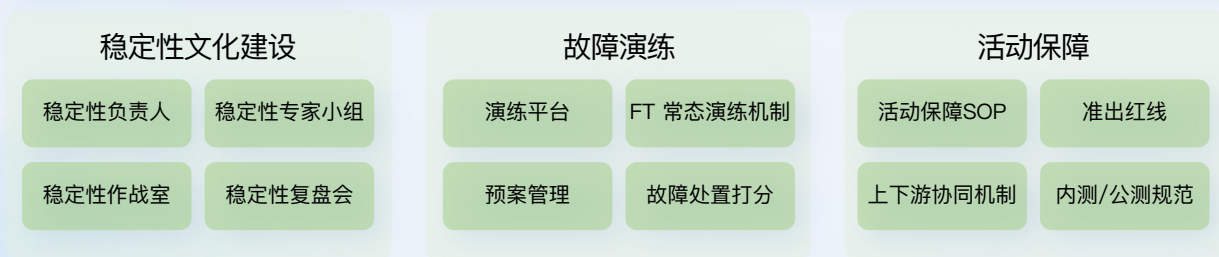


### 应急处置

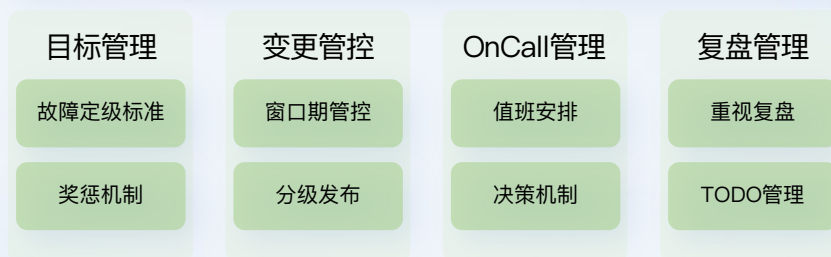


## 运营管理体系

### 运营体系



### 管理体系



AI 发力点如何聚焦？

# AI 带来的机遇 – 重点建设领域





# ● 如何实施：性能稳定性 Agent 架构介绍



# 02 实践：AI 辅助根因排障

排障本质 | Holmes | 火焰图

# AI 辅助根因排障：从一例 NPE 谈起

## 案例：NPE 问题都很简单？

Java异常堆栈跟踪

复制 展开全部

❗ java.lang.NullPointerException: Attempt to read from field 'int android.view.View.mViewFlags' on a null object reference

1. android.view.ViewGroup.dispatchGetDisplayList(ViewGroup.java:4385)

2. android.view.View.updateDisplayListIfDirty(View.java:19280)

3. android.view.ViewGroup.recreateChildDisplayList(ViewGroup.java:4413)

...

4. android.view.ThreadedRenderer.updateViewTreeDisplayList(ThreadedRenderer.java:686)

5. android.view.ThreadedRenderer.updateRootDisplayList(ThreadedRenderer.java:692)

6. android.view.ThreadedRenderer.draw(ThreadedRenderer.java:801)

7. android.view.ViewRootImpl.draw(ViewRootImpl.java:3559)

活动发布，屡次出现

问题难度 ★★★★★

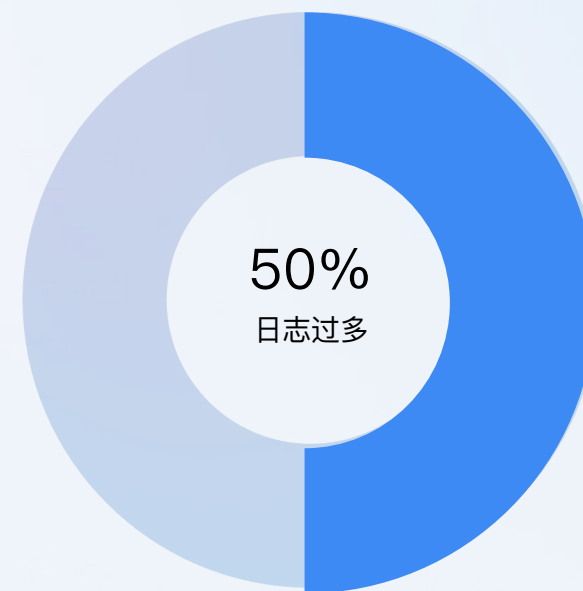
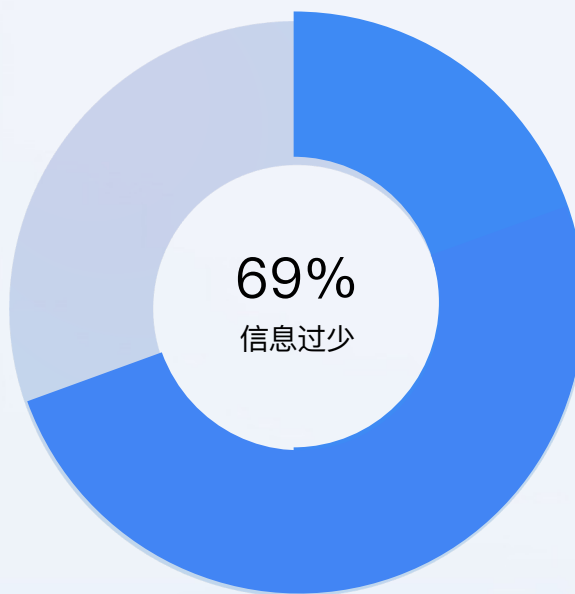
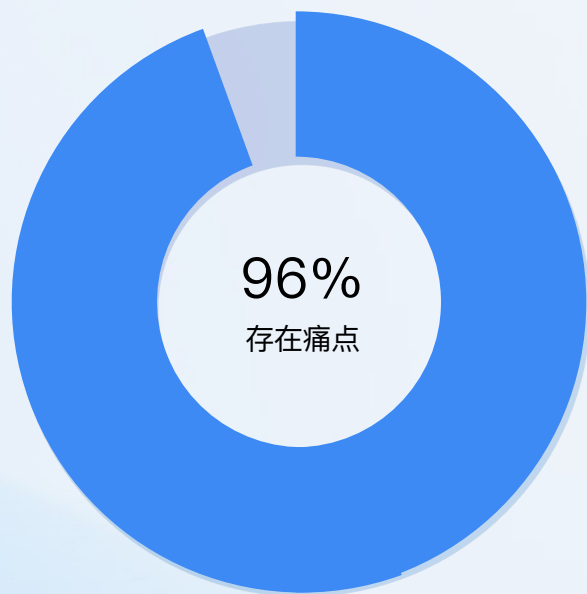
- **堆栈特征**：纯系统堆栈崩溃，没有一行业务代码逻辑。
- **现场缺失**：触发NPE滞后，无异步更新UI的第一现场。
- **复现困难**：在特殊场景下概率性触发，线下调试困难。
- **历史频发**：影响活动放量。



2025, AI 能秒掉这个 NPE 问题吗

# AI 辅助根因排障：调研数据

- 96%认为在日常崩溃排障中存在痛点；
- 69%认为崩溃现场信息过少，偶现问题无法复现，问题无法排查；
- 50%认为日志过多，无法快速找到问题关联日志，问题无法排查。



\* 调研方法：问卷 样本量：80 调研时间：25/09

• 软件开发者花费35–50%的时间在验证和调试软件上，调试、测试和验证的成本估计占软件开发项目总预算的50–75%  
来源：[ACM Queue – The Debugging Mindset \(2017\)](#)



## AI 时代编程范式（性能稳定性视角）

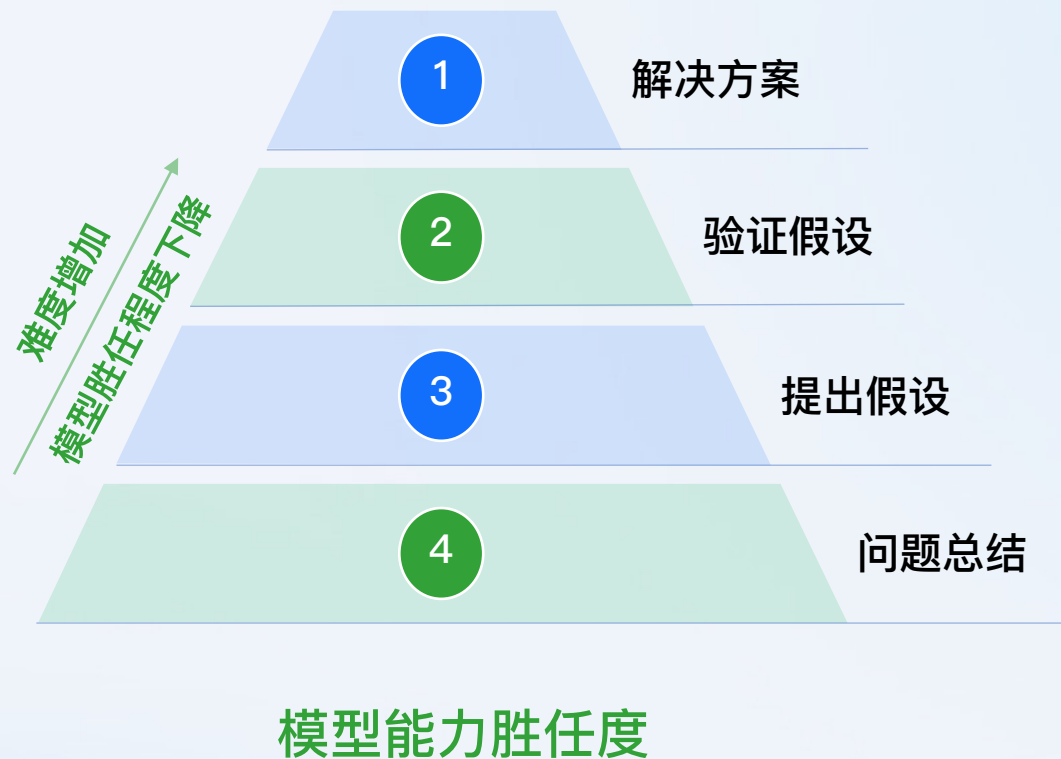
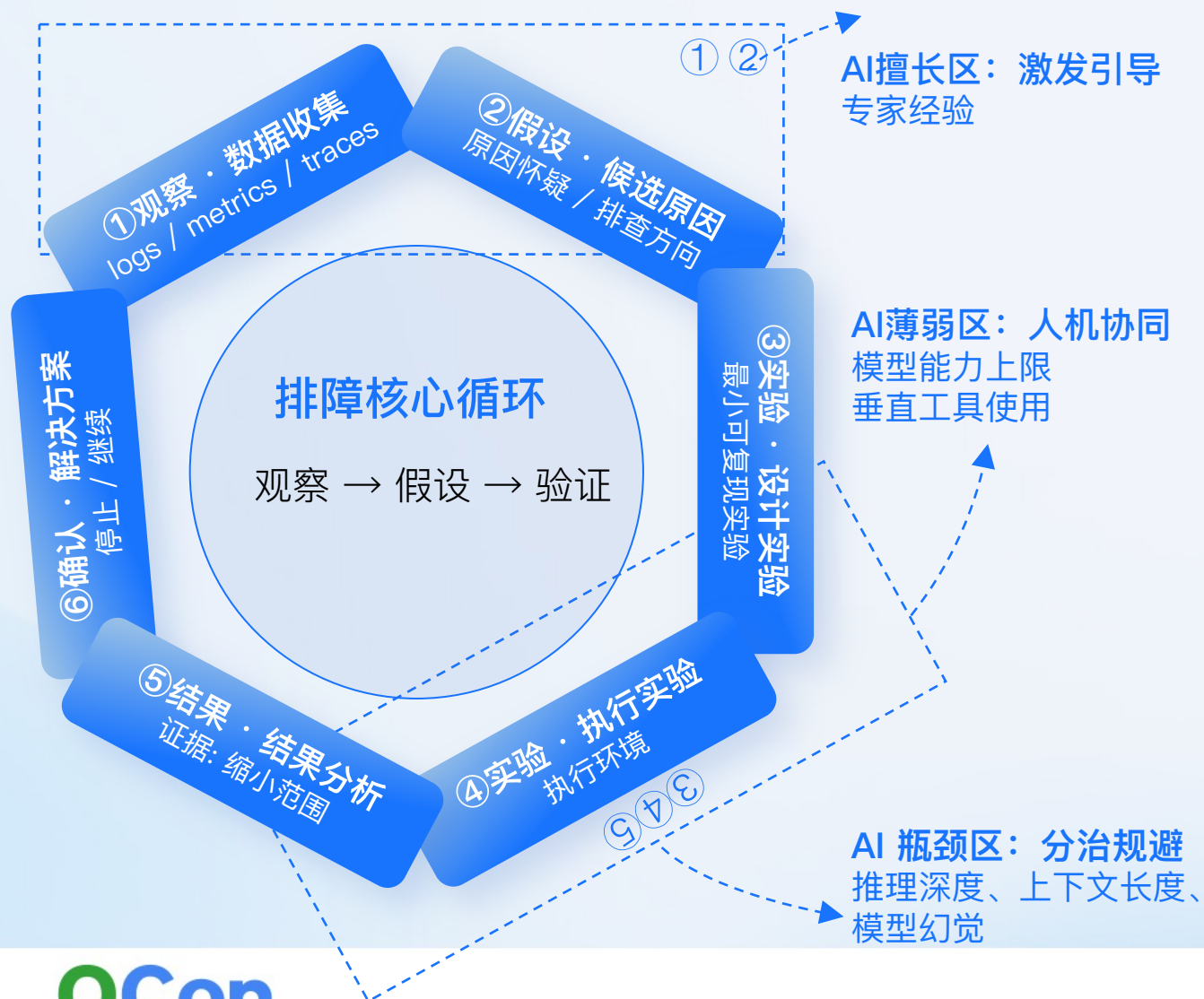
Talk is cheap, show me the code.



核心复杂度

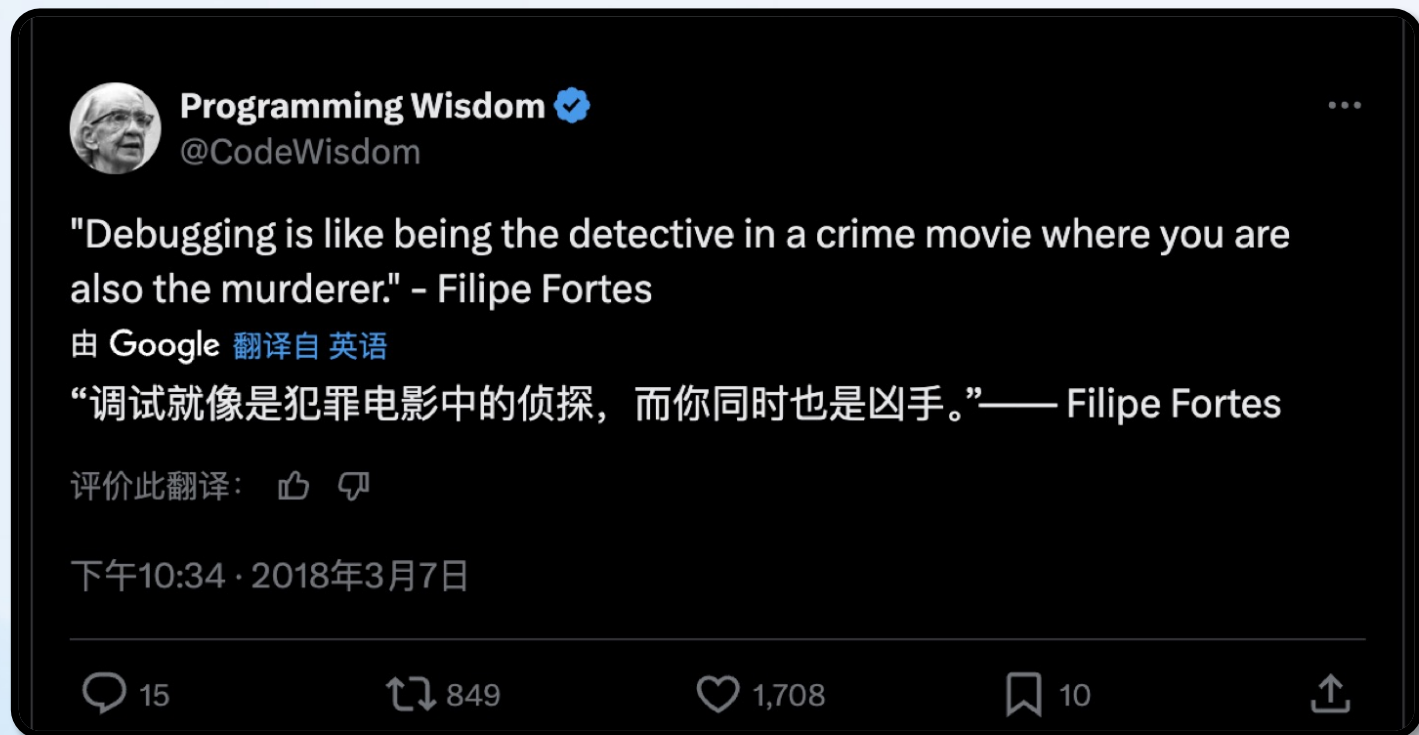
Code is cheap, show me the fix (bugfree).

# AI 辅助根因排障：排障本质 & AI 能力分析



参考: MIT 6.031 课程-debugging  
<https://web.mit.edu/6.031/www/fa17/classes/13-debugging/>

# AI 辅助根因排障：破案 vs. 排障



共同点：逆向推理的认知活动

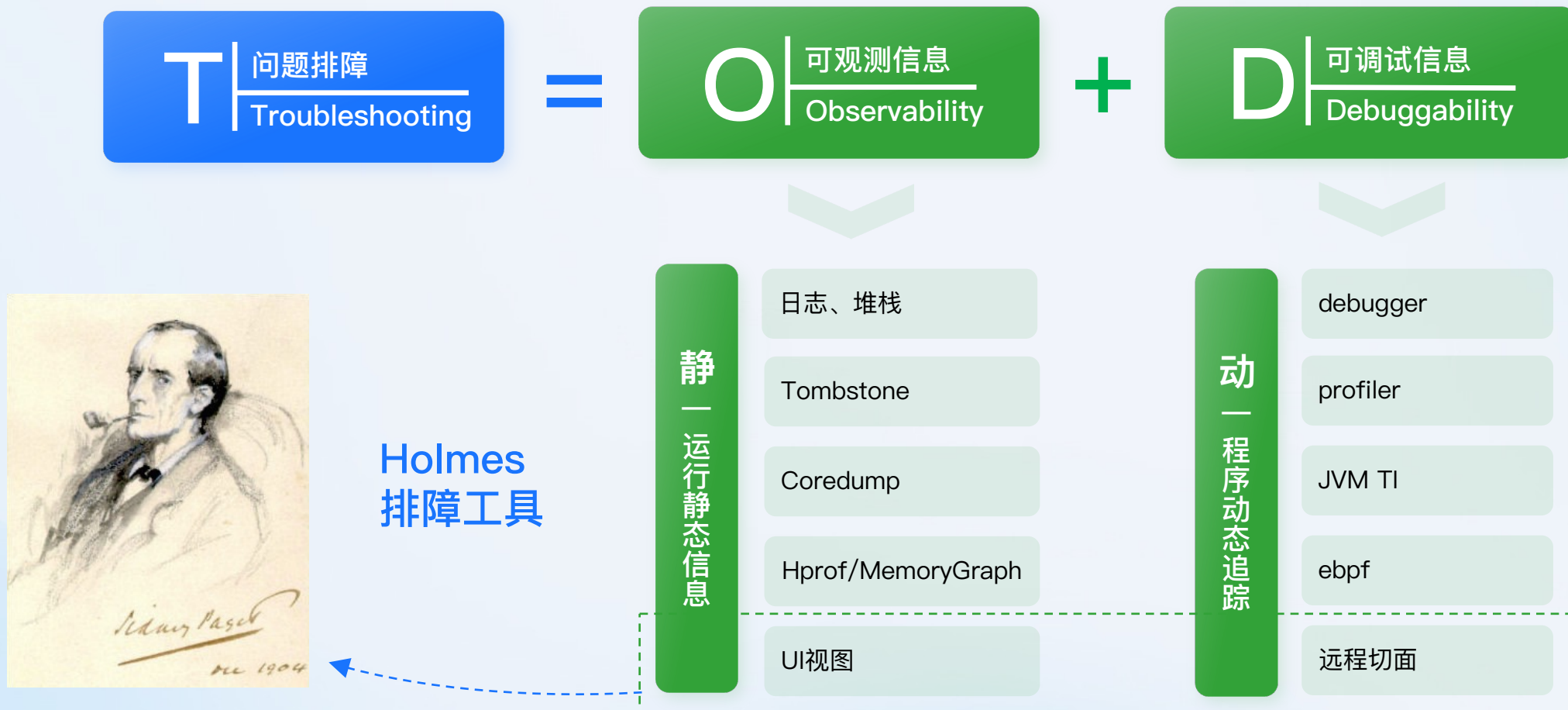
## 破案（侦查）

- 分析线索：物证、指纹、证人陈述
- 推测动机：犯罪意图、作案方式
- 给出嫌疑犯：潜在的犯罪者

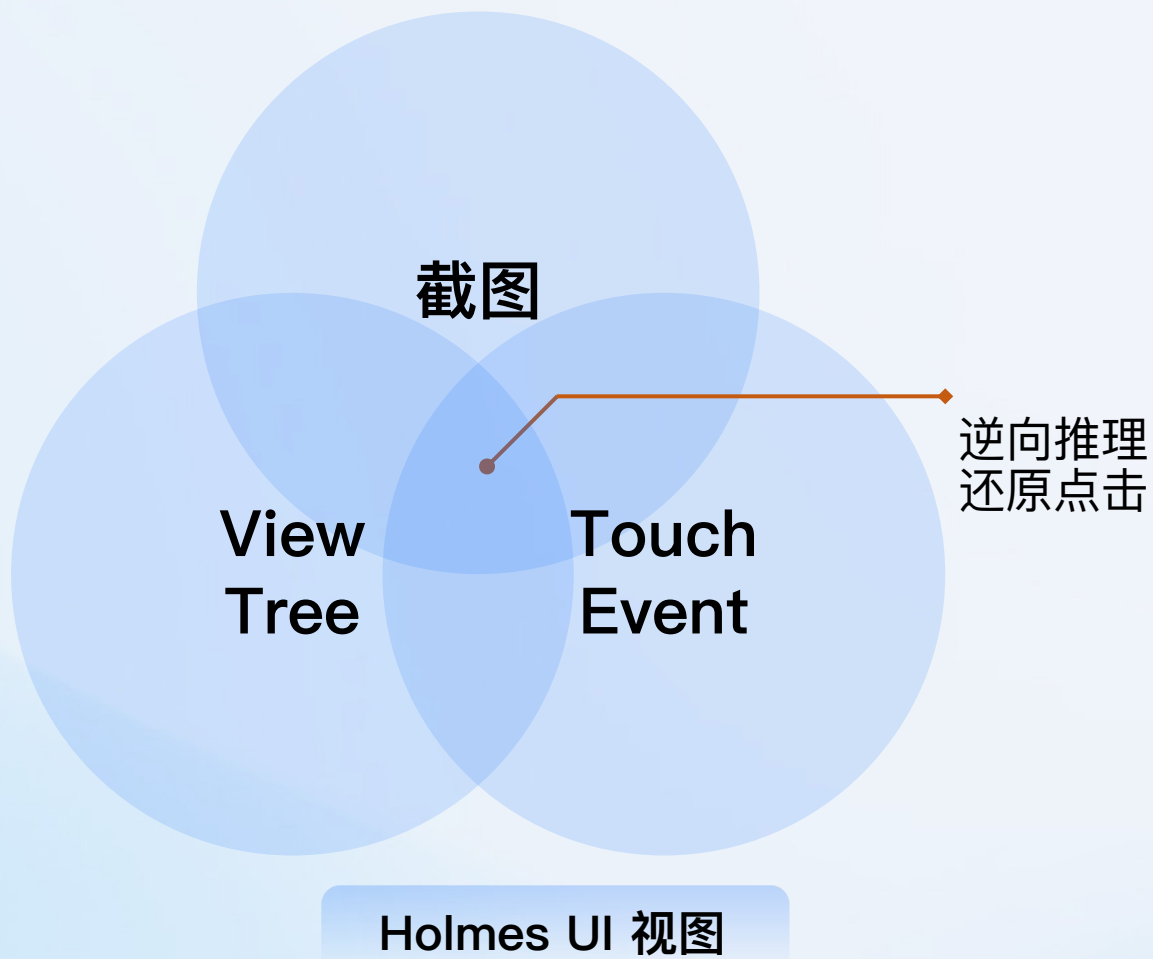
## 排障（debug）

- 分析线索：代码、日志、堆栈、反馈
- 推测动机：程序原意、执行偏差、异常
- 给出嫌疑犯：潜在的Bug引入者

# AI 辅助根因排障：Holmes 介绍



# AI 辅助根因排障: Holmes UI 视图



```
"isDirty": false,
"children": [
  {
    "cla": "class android.widget.FrameLayout@e7ca443",
    "res-id": "app:id/left_container",
    "id": 2131315645,
    "rect": "l-0 t-0 r-80 b-52",
    "measure": "w-80 h-52 state-0",
    "mRecreateDisplayList": false,
    "mViewFlags": 402653328,
    "mViewFlags.desc": "",
    "mPrivateFlags": -2130669392,
    "mPrivateFlags.desc": "HAS_BOUNDS_DRAWN",
    "visibility": "V",
    "pressed": false,
    "isDirty": false,
    "children": [
      {
        "cla": "class android.widget.FrameLayout@3f08ac0",
        "res-id": "app:id/left_fans_group_container",
        "id": 2131316271,
        "rect": "l-0 t-0 r-0 b-0",
        "measure": "w-0 h-0 state-0",
        "mRecreateDisplayList": false,
        "mViewFlags": 402653336,
        "mViewFlags.desc": "GONE",
        "mPrivateFlags": -2130702176,
        "mPrivateFlags.desc": "DRAWN",
        "visibility": "G",
        "pressed": false,
```

# AI 辅助根因排障：Holmes UI 视图

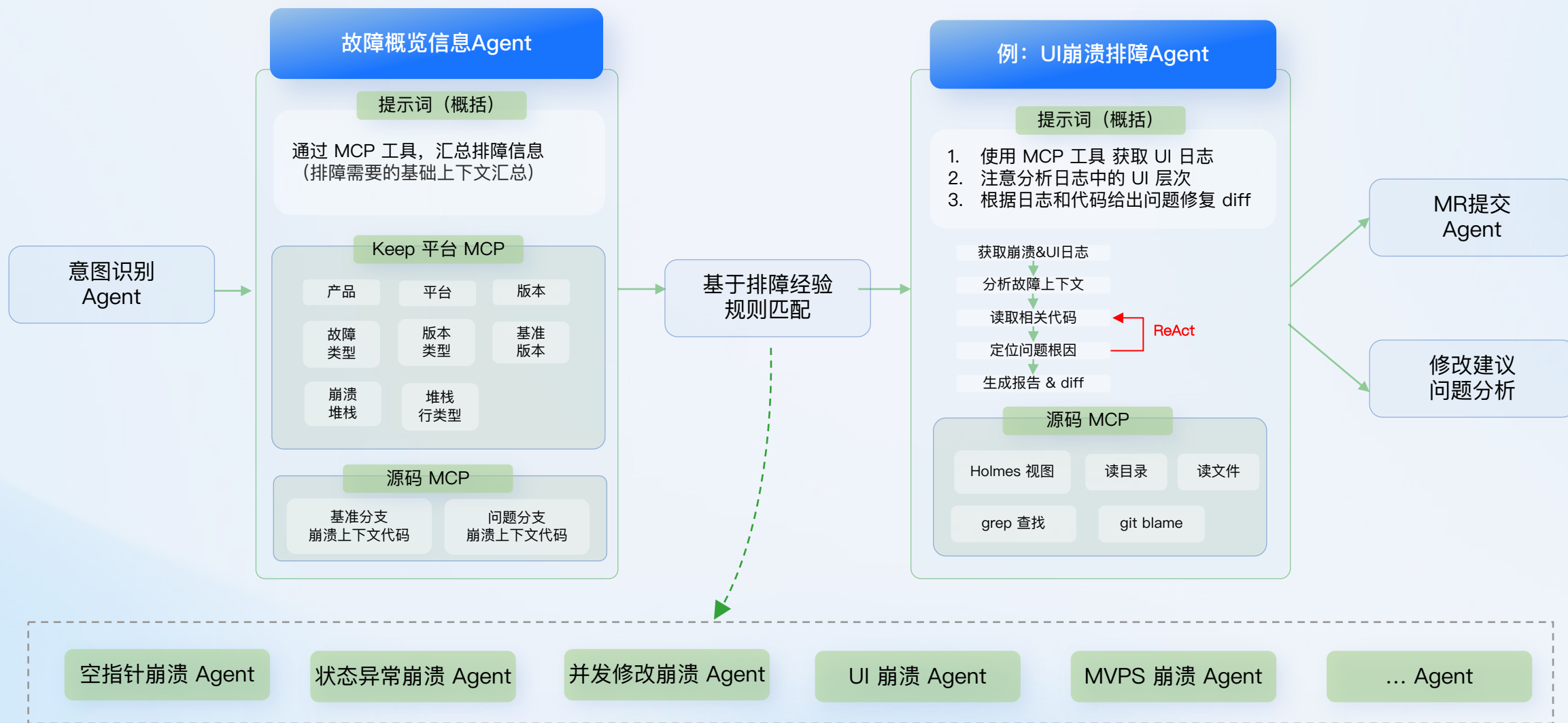






这么多信息，记不过来？

# AI x 根因排障：如何实施



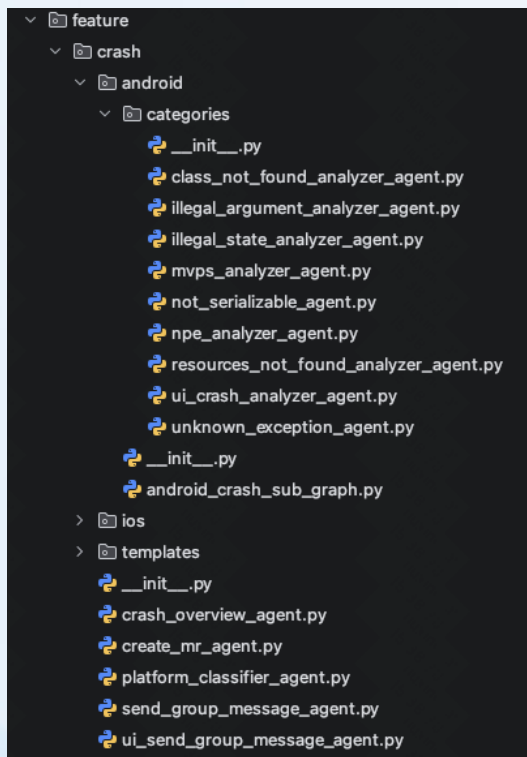


**“Without the experience of actually doing it, you never get three dimensional.”**

不实干，学到的永远只是皮毛。

来源：乔布斯 1992 年 MIT 演讲

# AI x 根因排障：如何提升准确率



```
基于 Key 查询 (例如: INJECTKEY) :  
{  
  "regex": "[InjectInjectOptionalInjectRefInjectOptionalRefInjectOptionalPrimitive]<[>]>?\\[[\\]\\]*INJECTKEY[\\]\\]*\\]",  
  "regex_path": "xxs-features/xx/xx/xx.kt",  
  "ref": "13.4.30.42809",  
  "line_context": 2  
}  
  
基于类型查询 (例如: ILoginPage) :  
{  
  "regex": "[InjectInjectOptionalInjectRefInjectOptionalRefInjectOptionalPrimitive]<[>]>?\\[[\\]\\]*ILogPage::class.java[\\]\\]*\\]",  
  "regex_path": "xxs-features/xx/xx/xx.kt",  
  "ref": "13.4.30.42809",  
  "line_context": 2  
}
```

## 上下文信息不完备 (欠定)

### 模型输入

达成目标信息不足    目标过于抽象/太大

高熵/随机采样

输出：模糊/幻觉/不确定解

## 上下文噪声污染 低信噪比 (超定)

### 模型输入

多职责/目标    无关信息过多

注意力稀释/误导

输出：偏题/被误导的解

## 上下文信息完备 高信噪比 (适中)

### 模型输入

单一职责    直接相关信息  
Few-shot    量化标准

低熵/逻辑推理

输出：确定性解



# AI x 根因排障：效果展示

## AI 助手意图识别

### Kim IM 智能助手使用指南

您好！我是 Kim IM 智能助手，可以帮您快速处理故障相关问题。

#### 使用方法：

请在消息中包含 keep 链接，并说明您的需求。

我可以帮您：

#### 修复故障

- 示例："帮我修复这个崩溃 [keep链接]"
- 示例："创建故障MR [keep链接]"
- 支持关键词：修复崩溃、修复故障、fix、修bug、解决崩溃

#### 止损故障

- 示例："生成气垫配置 [keep链接]"
- 示例："帮我止损这个故障 [keep链接]"
- 支持关键词：安全气垫、气垫配置、止损、降级

#### 根因分析

- 示例："分析这个崩溃的原因 [keep链接]"
- 示例："为什么会崩溃 [keep链接]"
- 支持关键词：崩溃分析、主要原因、根因分析、分析原因

#### 温馨提示

- 请确保消息中包含 keep 链接
- 使用上述关键词可以更准确地识别您的需求

## AI 问题排查建议

### 二、根因推断

- **堆栈关联分析：**堆栈信息显示崩溃发生在 `aemon_PlayList_algo_statistic_thread` (位于 `kflv.c:3946-3947`)，并由 `aemon_algo_cond_wait` (位于 `kflv.c:446`) 线程等待/唤醒逻辑触发，线程创建调用为 `aemon_SDL_RunThread` (位于 `kpplayer_adapter.cc:356`)。结合【LIVE\_WATCH 场景占比极高】的特征，初步判断为直播播放内核 (KFlv playlist) 在统计/自适应算法线程中发生的空指针解引用导致的 `EXC_BAD_ACCESS` (`KERN_INVALID_ADDRESS`)。
- **代码逻辑推测：**关键代码为 `kflv.c:3947`：

```
if (*(playlist->video_state) == NULL || tag_queue->abort_request || playlist->read_abort_request) { ... }
```

。此处直接解引用 `playlist->video_state` 再判空，若 `playlist->video_state == NULL` 将发生 NPE。且 `algo_cond_wait` 的实现为：先 `SDL_LockMutex`，`SDL_CondWaitTimeout` 返回后又立刻 `SDL_UnlockMutex`，导致调用点在检查 `playlist->video_state` 时并未持有互斥锁，存在竞态：其他线程可能在此间将 `playlist->video_state` 置空或释放，触发空指针解引用。结合【直播观看场景】的高并发与频繁切流/停止操作（如 ABR 切换、读取消等），该竞态更易出现。

### 三、问题定位

- **首要排查方向：**优先审查 `kflv.c` 中 `aemon_PlayList_algo_statistic_thread` (3946-3949 行) 与 `algo_cond_wait` (446 行) 相关的锁与判空逻辑，梳理 `playlist->video_state` 的生命周期与并发访问时序。确认在 `stop/read_abort/abort_request` 等路径是否可能将 `playlist->video_state` 置空，并与本线程无锁读发生竞态。
- **具体建议：**
  - i. 将判空顺序改为先判断指针是否为 NULL：

```
if (playlist->video_state == NULL || *(playlist->video_state) == NULL || tag_queue->abort_request || playlist->read_abort_request) { ... }
```
  - ii. 调整同步策略：确保在访问 `playlist->video_state` 时已持有与其写路径一致的互斥锁；考虑让 `algo_cond_wait` 返回时保持持锁语义（典型 `cond wait` 语义：调用方在持锁状态下继续后续检查），或在调用点显式加锁保护随后访问的共享状态。
  - iii. 排查停止/切流路径中 `video_state` 释放/置空时机，必要时使用原子指针或 RCU/引用计数，避免悬挂指针。

## AI 问题修复自动提交



泰山 BOT 08-25 22:49:10

🔧 修复上下滑场景NasaSideslipPlayerPresenter模块因inject(SERIAL\_EPISODE\_SHARE\_DATA)导致的IAE问题「泰山AI」

🔗 Keep 详情 🔗 MR #126042

#### 问题分析：

- 在 `NasaSideslipPlayerPresenter` 和 `SlideAutoPlaySerialOptPresenter` 中，`SERIAL_EPISODE_SHARE_DATA` 被强制注入，但在某些场景下该数据可能未被提供，导致 `InjectDelegate` 抛出 `IllegalArgumentException`，引发应用崩溃。

#### 修复方案：

- 将 `inject(AccessIds.SERIAL_EPISODE_SHARE_DATA)` 修改为 `injectOptional(AccessIds.SERIAL_EPISODE_SHARE_DATA)`，将强制注入改为可选注入，以允许 `mSlidePlayEpisodeSerialShareData` 为空，从而避免在数据缺失时发生崩溃。

```
--- a/ks-features/ft-feed/detail/src/com/yxcorp/gifshow/detail/sideslip/NasaSideslip
+++ b/ks-features/ft-feed/detail/src/com/yxcorp/gifshow/detail/sideslip/NasaSideslip
@@ -783,7 +783,7 @@

 override fun doInject() {
     super.doInject()
-    mSlidePlayEpisodeSerialShareData = inject(AccessIds.SERIAL_EPISODE_SHARE_DATA)
+    mSlidePlayEpisodeSerialShareData = injectOptional(AccessIds.SERIAL_EPISODE_SHARE_DATA)
     mPhoto = inject(QPhoto::class.java)
     mSlidePlayGlobalCache = inject(SlidePlayGlobalCache::class.java)
     mSwipeToProfileFeedMovement = inject(SwipeToProfileFeedMovement::class.java)
```

TraceID: 4f1610ba | ajian\_0822\_2(d0140c8f)

完美👍

有帮助👍

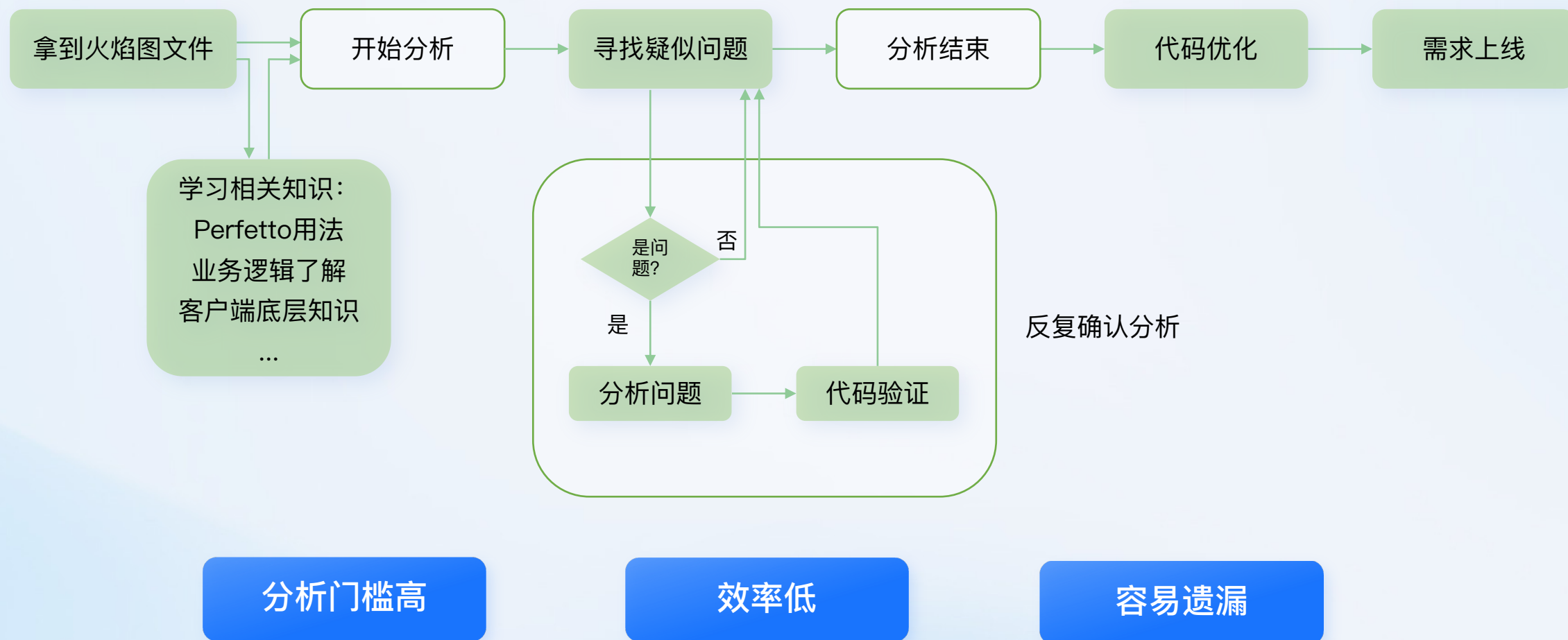
失望👎







# AI x 根因排障：人工火焰图痛点



# AI x 根因排障：火焰图原理架构

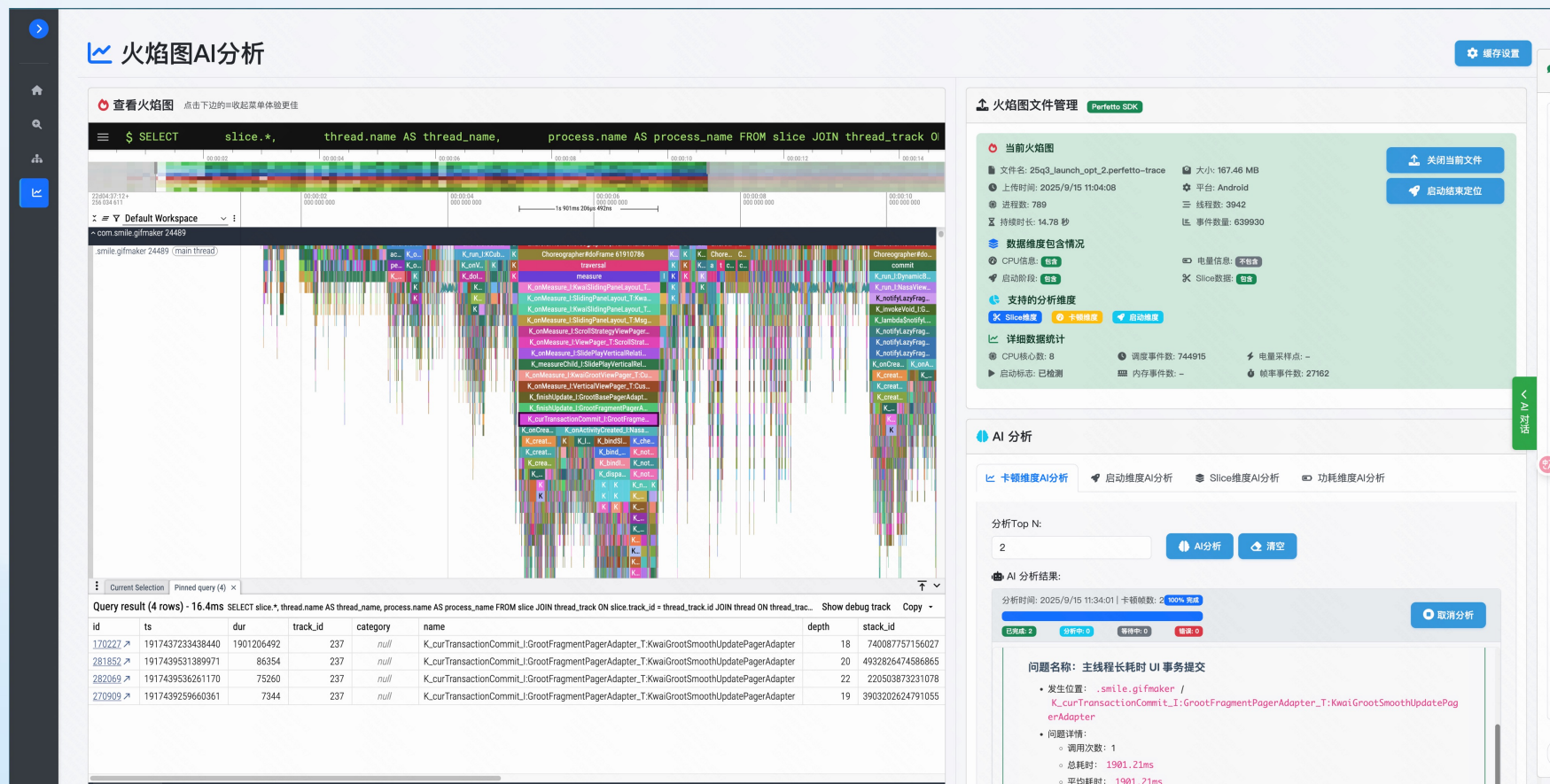


## 关键思路 (Key Ideas)

- 数据预处理分析 (Data preprocessing analysis)
- 源码关联分析 (Source code correlation analysis)



# AI x 火焰图: 效果展示



## 典型性能问题

- 主线程耗时
- 频繁调用
- GC
- 锁竞争

03

# 实践：AI 加速应急处置

止损工具 | Ekko | 效果分析

# AI 加速故障应急处置

案例：[iOS 26 你的 property 崩了吗？](#)

## Objective-C Runtime

### New Features

- Concurrent mutation of nonatomic properties in Objective-C will now sometimes produce more actionable crashes. Synthesized setters will briefly store the sentinel value `0x400000000000bad0` (0xbad0 on 32-bit watchOS) which may be read by another thread accessing the property unsafely. A crash on this sentinel value indicates a thread safety issue with the property it came from. (148109501)

影响几十个旧App版本，如何快速止损

\* 官方文档：<https://developer.apple.com/documentation/xcode-release-notes/xcode-26-release-notes>



# AI 加速故障应急处置

## 止损工具的痛点

01

### 商店更新

覆盖率低，用户流失风险。

02

### 变更回滚

依赖归因成功率，且外部变更变更不可回滚，活动特性发布类不可回滚。

03

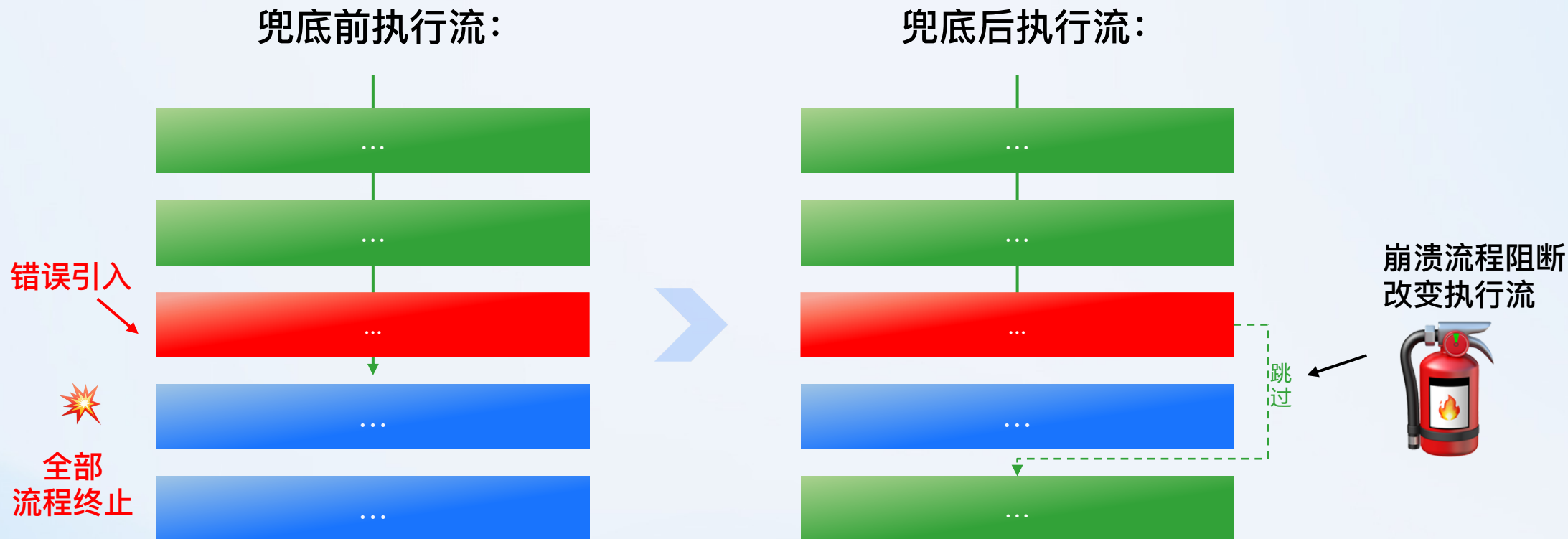
### 根因修复

时效性差，需先定位问题根因，发布流程长，性能损耗。

核心：时效性要求高，不能依赖根因定位

# AI 加速故障应急处置

## Ekko：修改程序执行流



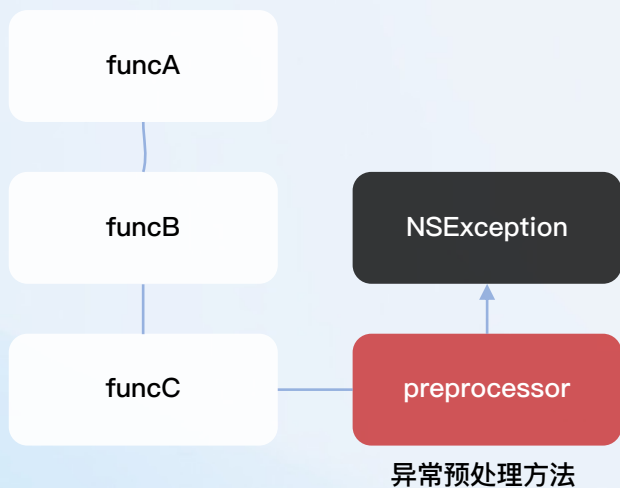
Ekko: 让程序运行也能有R技能，时光倒流，恢复正常状态

# 故障应急体系建设

## Ekko 崩溃阻断：覆盖所有崩溃类型

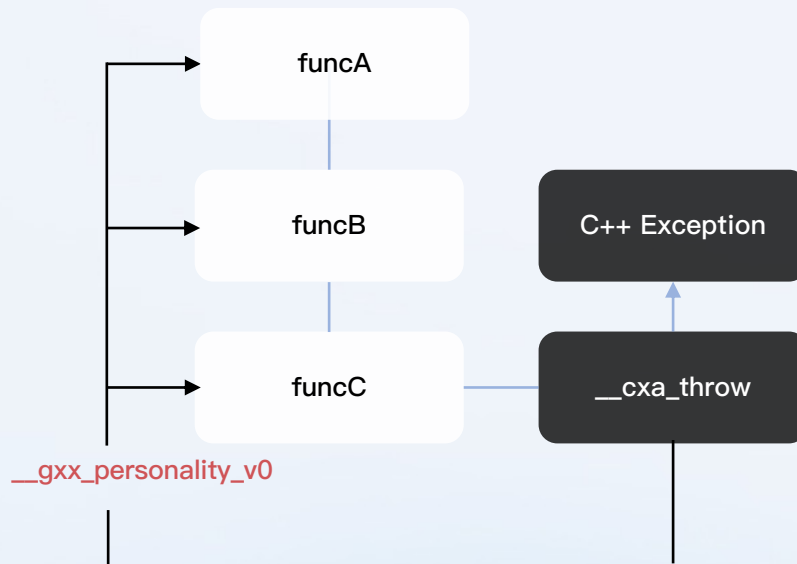
### NSException

libobjc提供的 exception\_preprocessor



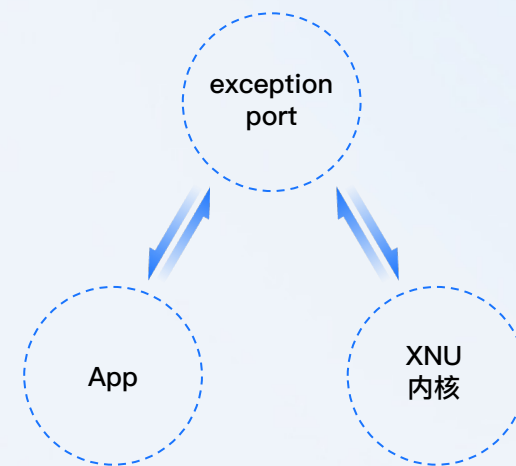
### C++ Exception

主可执行文件的 personality routine



### Mach

基于 task exception port 应答，请求恢复崩溃线程的执行。



\* 注意：iOS sinkhole异常陷阱机制

# 故障应急体系建设

## Ekko 崩溃恢复：类型无关通用性强

### 核心流程

异常发生之后，App 闪退之前，通过动态修改程序执行流返回至上层函数后继续向下执行。

### 实现原理

**地址恢复：**计算并修改目标兜底函数的 PC LR

**上下文重建：**恢复兜底函数的 callee-saved registers 以及“伪造”正在等待 callee 的返回值

### 技术挑战

PAC签名；异步 stack unwind；兼容 unwind info 缺失



# AI 加速故障应急处置：安全气垫发布

1.兜底决策

2.生成配置

3.1白名单

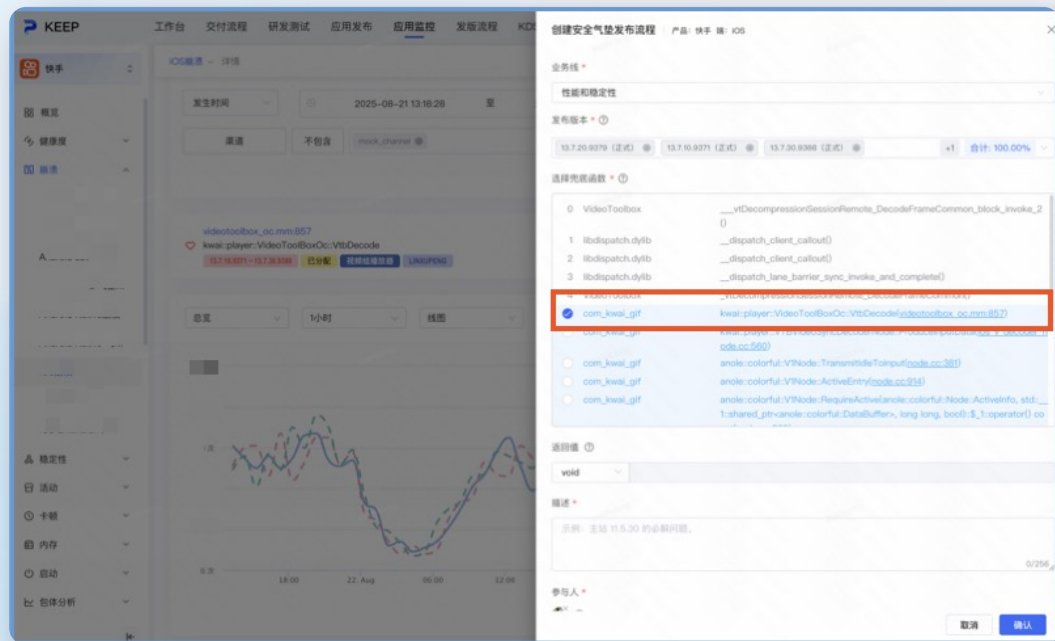
3.2审核

3.3灰度

3.4全量

4.数据观测

## AI 加速配置决策



## AI 加速流水线发布



### 效果说明

发生异常时跳过扩展参数拼装，继续后续请求流程，仅在抛出崩溃时生效，正常路径不受影响。

### 选择原因

1. [AndroidAnalyzer] 开始分析Android崩溃数据...

1. 解析异常链：当前堆栈未包含“Caused by”段落，按最外层异常分析。遵循优先策略，选择第一个 is\_app\_stack=true 的堆栈作为候选：index=0  
AwardAdDataUtils.interceptBusinessExtToServer(AwardAdDataUtils.kt:584)。

2. 源码定位 (dev.content)：在582-586行看到对 impExtMap 的遍历：

- 582: val impExtMap = strToMap(mImpExtData)
- 583: impExtMap.forEach {
- 584: val value = it.value
- 585: if (!params.containsKey(it.key) && value?.isNotBlank() == true) {
- 586: params[it.key] = value

根据崩溃标题 ClassCastException: Double cannot be cast to String, 推断 impExtMap 的值可能为非字符串 (如 Double)，而在 585 行调用 value?.isNotBlank() (通常是 String/CharSequence 的扩展) 导致在内部发生类型强转从 Any 到 String 的 ClassCastException, 触发崩溃。

3. 禁用规则检查：

- 锁操作：方法体未见 synchronized/显式锁(lock()/unlock()), 无死锁风险 ✓。
- 循环内方法调用：上层 NeoMixedRequest\$fetchData\$1.run(NeoMixedRequest.kt:51)仅一次调用该方法，未在循环中反复调用 ✓。候选方法自身包含 forEach 遍历，但规则禁止的是“被循环频繁调用”的方法；本方法不是在外循环中被反复调用，符合要求。

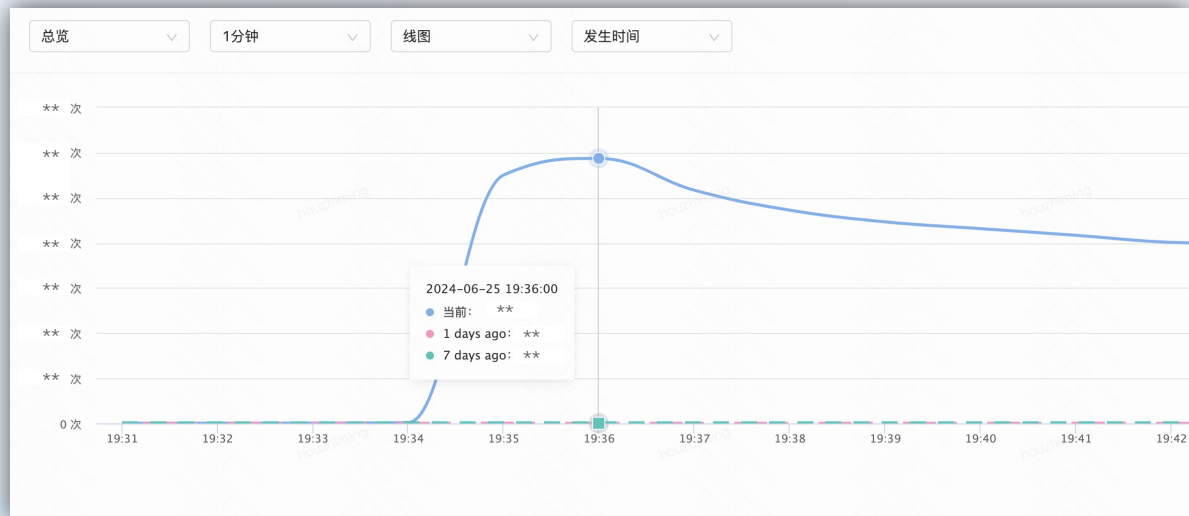
4. 上层一层调用分析 (NeoMixedRequest.kt 49-53)：

- 49: val adSession: AdSession = param.toAdSession()
- 50: AwardAdDataUtils.businessCustomConfigToServer(...)



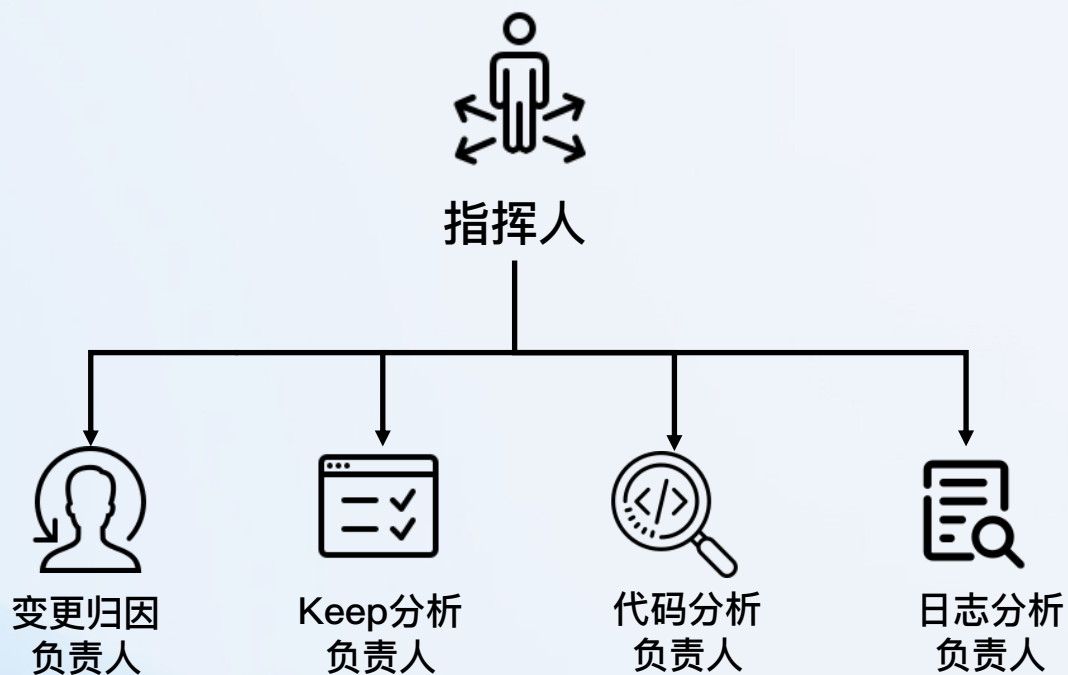
# AI 加速故障应急处置

案例：突发黑天鹅事件，原因竟是...



\*万级崩溃，肾上腺素飙升，问题迟迟得不到止损

# AI x 应急处置：分析判断



依赖人工驱动，易错易漏



AI 驱动

# AI x 应急处置：Agent 流程编排



# AI x 应急处置：效果展示

故障触发

异常识别

根因推理

排障建议

人机协同

## 应急处置 Checklist

请根据团队制定的SOP流程进行应急处置，将勾选已完成或无需处理的CheckBox，并点击「保存标记结果」按钮上传处置进度。供其他响应人员实时了解进展。

你的应急角色为：日志分析负责人、直接干系人

|                                |                              |                               |
|--------------------------------|------------------------------|-------------------------------|
| 优先从keep看板下载用户日志进行分析            | <input type="checkbox"/> 已处理 | <input type="checkbox"/> 无需处理 |
| 同时使用小灵通等工具进行日志回捞               | <input type="checkbox"/> 已处理 | <input type="checkbox"/> 无需处理 |
| 确保日志的进程id和发生崩溃的进程id一致          | <input type="checkbox"/> 已处理 | <input type="checkbox"/> 无需处理 |
| 优先分析崩溃发生时间附近的日志                | <input type="checkbox"/> 已处理 | <input type="checkbox"/> 无需处理 |
| 可疑点及时同步给决策指挥人、更新到Katrix，并交叉验证  | <input type="checkbox"/> 已处理 | <input type="checkbox"/> 无需处理 |
| 根据代码分析负责人提供的排查信息，及时调整分析日志的重点方向 | <input type="checkbox"/> 已处理 | <input type="checkbox"/> 无需处理 |
| 提醒大家验证怀疑点，避免关键信息被遗漏            | <input type="checkbox"/> 已处理 | <input type="checkbox"/> 无需处理 |

⚠ 注意：如需取消请到平台操作，此处取消不生效。[应急单详情>>](#)

💡 提醒：该卡片不会自动刷新，如需获取最新进度，请点击「获取最新状态」按钮

## 应急处置多维度信息诊断

AI模型AllocateTensors导致的Native崩溃

多维度信息诊断

keep: [keep 链接](#)

一、异常聚集特征

- 核心总结：此崩溃高度集中于Android 35系统版本(96.7%)、mt6985 CPU平台(95.0%)、单一App版本13.8.30.44348(100%)，呈现明显的硬件/系统层面兼容性问题特征。
- 特征分析：
  - 系统版本：Android 35占比为 **96.7%**（基准数据中占比较低）。
  - CPU平台：mt6985占比为 **95.0%**（基准数据中mt6985未在前5名）。
  - App版本：13.8.30.44348占比为 **100%**（单一版本集中）。
  - 当前页面：FEATURED\_PAGE占比为 **55.0%**（基准数据中为31.8%）。

二、根因推断

- 堆栈关联分析：堆栈信息显示崩溃发生在 `AllocateTensors`（位于 `libbflite_static_mtk.so`）。结合【Android 35系统占比极高】的核心特征，初步推断崩溃与MediaTek平台的TensorFlow Lite在Android 35系统上的兼容性问题强相关。
- 代码逻辑推测：代码中TensorFlow Lite的Neuron Adapter在mt6985芯片上进行模型编译和优化时，可能未对Android 35系统中的NNAPI(Neural Networks API)变更进行必要的兼容性适配。结合【CPU平台mt6985高度集中】的特征，推测该问题特定于MediaTek的神经网络加速器驱动在新系统版本下的行为变化。

三、问题定位

- 首要排查方向：建议优先审查 `libbflite_static_mtk.so` 中 `AllocateTensors` 周边的代码逻辑。
- 具体建议：重点关注与 **App v13.8.30.44348** 版本相关的TensorFlow Lite集成代码提交记录，并务必在 **Android 35** 系统且配备mt6985芯片的设备上进行复现和调试。

### 搜索业务的详情页渲染模块未知文件导致的长卡顿

#### 多维度信息诊断

keep: [keep 链接](#)

#### 一、异常聚集特征

二、根因推断

三、问题定位

# 04 总结展望

Agent 开发 | 人机协同



# AI x 性能稳定性: Agent 开发感悟

01

## 思维切换

### Thinking in LLM

- 概率自回归 vs. 图灵机
- 能力边界，擅长区、薄弱区、瓶颈区；什么时候用模型、什么时候用代码程序、什么时候用工具

02

## 识别瓶颈

### 释放模型天花板

- 提示词无法提升模型认知上限（权重无变化），但是好的提示词可以激发模型表现
- 模型推理深度有限，提前基于专家知识做好步骤拆解

03

## 认识重要性

### 评测体系 = 上下文工程

- 调试：提示词 vs. 调试程序
- 合理设计架构，降低Agent能力回归评估成本（时间、钱）

## AI 展望：回顾最初的焦虑



冷静：正确认识 AI，不神话



乐观：积极拥抱 AI，不错过浪潮

# AI x 性能稳定性展望：人机协同

## 人类弱化

### 生产力释放

- “体力型”排障
- “死记硬背型”排障

## 人类强化

### 思考力

- 辨别因果、验证结果
- 系统性思考、创造性实验
- 深度业务理解、战略决策

Human in/on the Loop, 人类不可或缺

星辰大海，刚刚启航

QCon  
全球软件开发大会

# THANKS

大模型正在重新定义软件

Large Language Model Is Redefining The Software



李锐

浙江 杭州



扫一扫上面的二维码图案，加我为朋友。