QCon+ 案例研习社 哈啰出行

中型企业快速搭建性能监控体系

赵宇 / 哈啰平台前端 / 前端专家

哈啰出行

个人介绍

- 15–18年在有赞，先后做过架构、批发无线负责人、电商移动负责人；
- 18年创业，搭建20人技术团队，完成前后端基建，支撑业务从0到1；
- 19年加入哈啰，现任平台产品部前端Leader，团队负责公司业务中台和部分基建建设。

哈啰出行

背景 -> 痛点

公司业务量高速增长
业务种类快速横向扩张

迭代后性能衰减
性能问题影响到了用户体验
排查性能问题耗时+难以定量，阻塞业务迭代

哈啰出行

几个问题

- 为什么要先做性能监控系统?
- 为什么不能用工具模拟测试替代?
- 性能监控系统应该包含哪些东西?
- 需要关注哪些指标?

哈啰出行

性能优化流程

收集优化点：

报警触达
页面定位
数据收集
...

性能优化：

代码优化
打包优化
渲染优化
业务优化
...

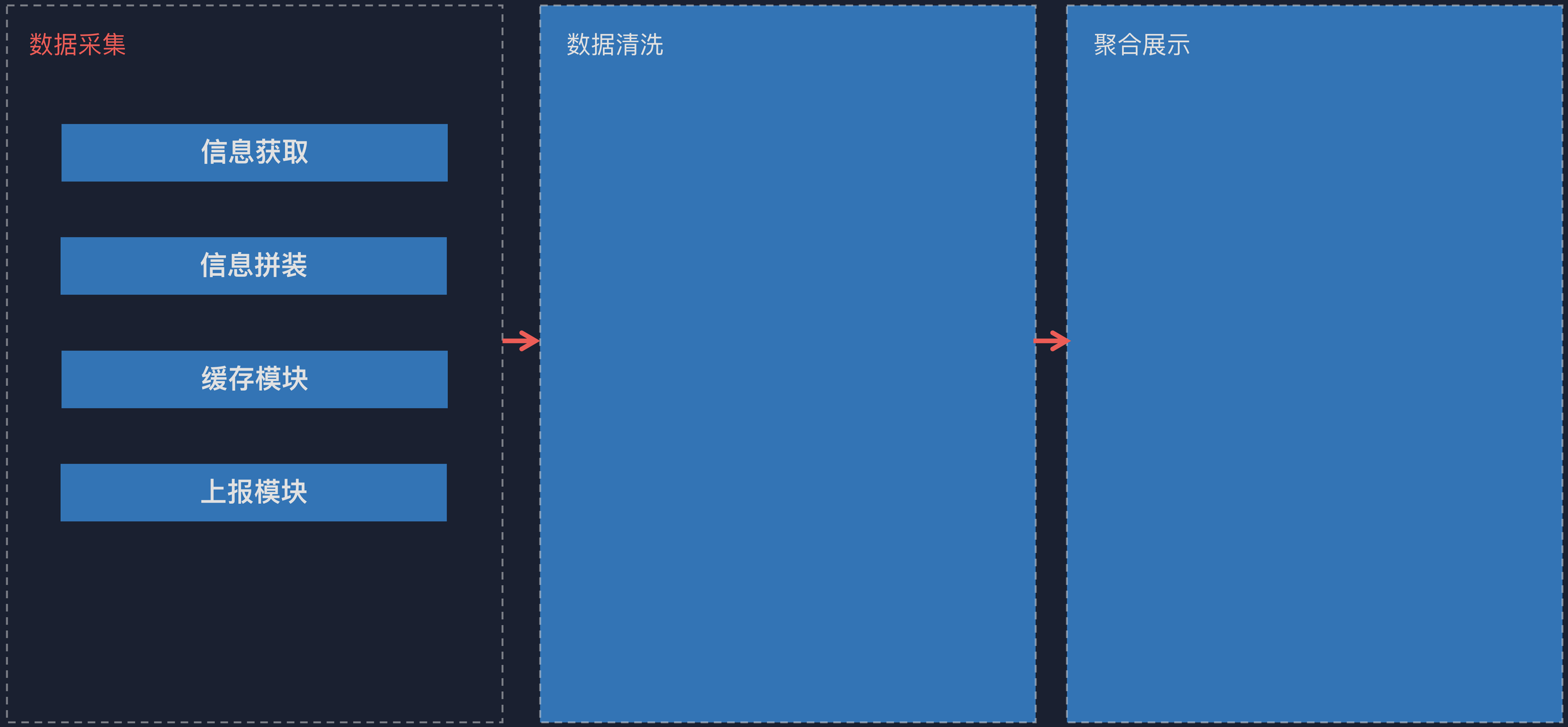
效果回收：

优化效果数据
同比对比
持续跟进数据
...

哈啰出行

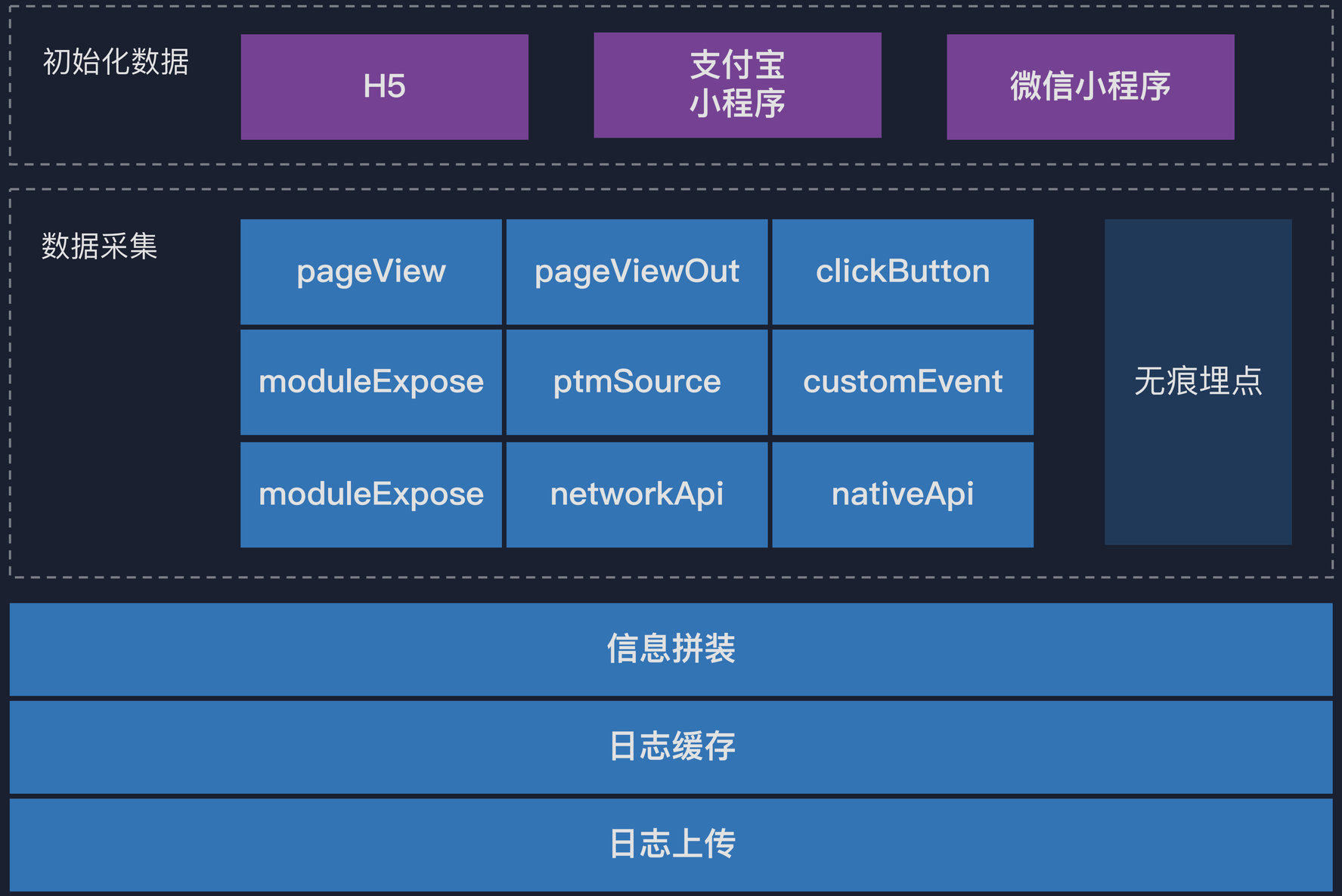
系统流程

性能监控 – 整体系统



哈啰出行

数据采集 – SDK架构



数据采集 – 性能信息

- 异常捕获
- 页面加载时间
  - 白屏时间
  - 页面加载时长
  - 可操作时间
- 帧率
- 网络情况
  - 网络请求
  - 静态资源加载
- 原生方法调用情况

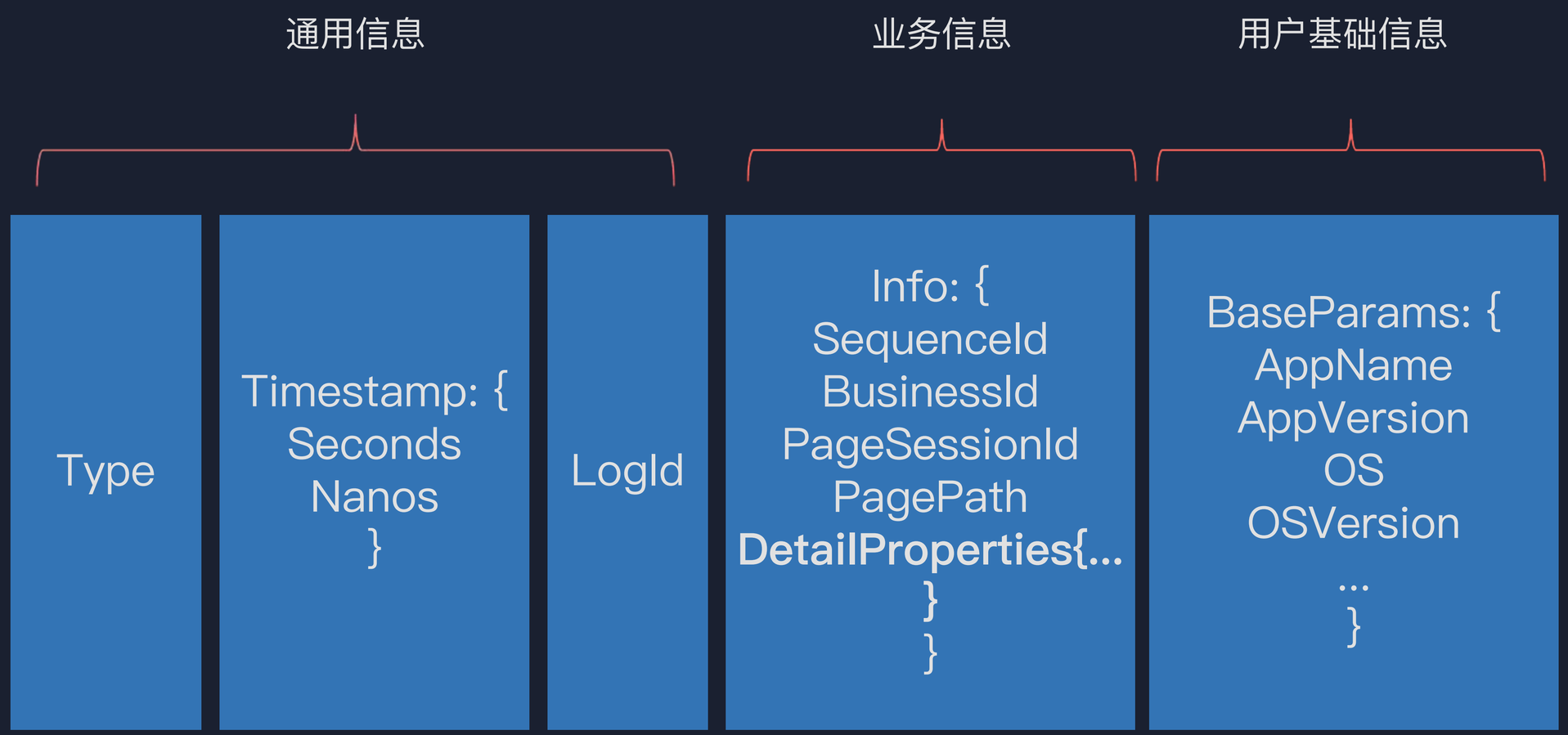
异常捕获

- 一般异常：监听 window.onerror
- Promise 未捕获异常：window.addEventListener('unhandledrejection'...
- Script Error：Access–Control–Allow–Origin + 标签 crossorigin="anonymous"

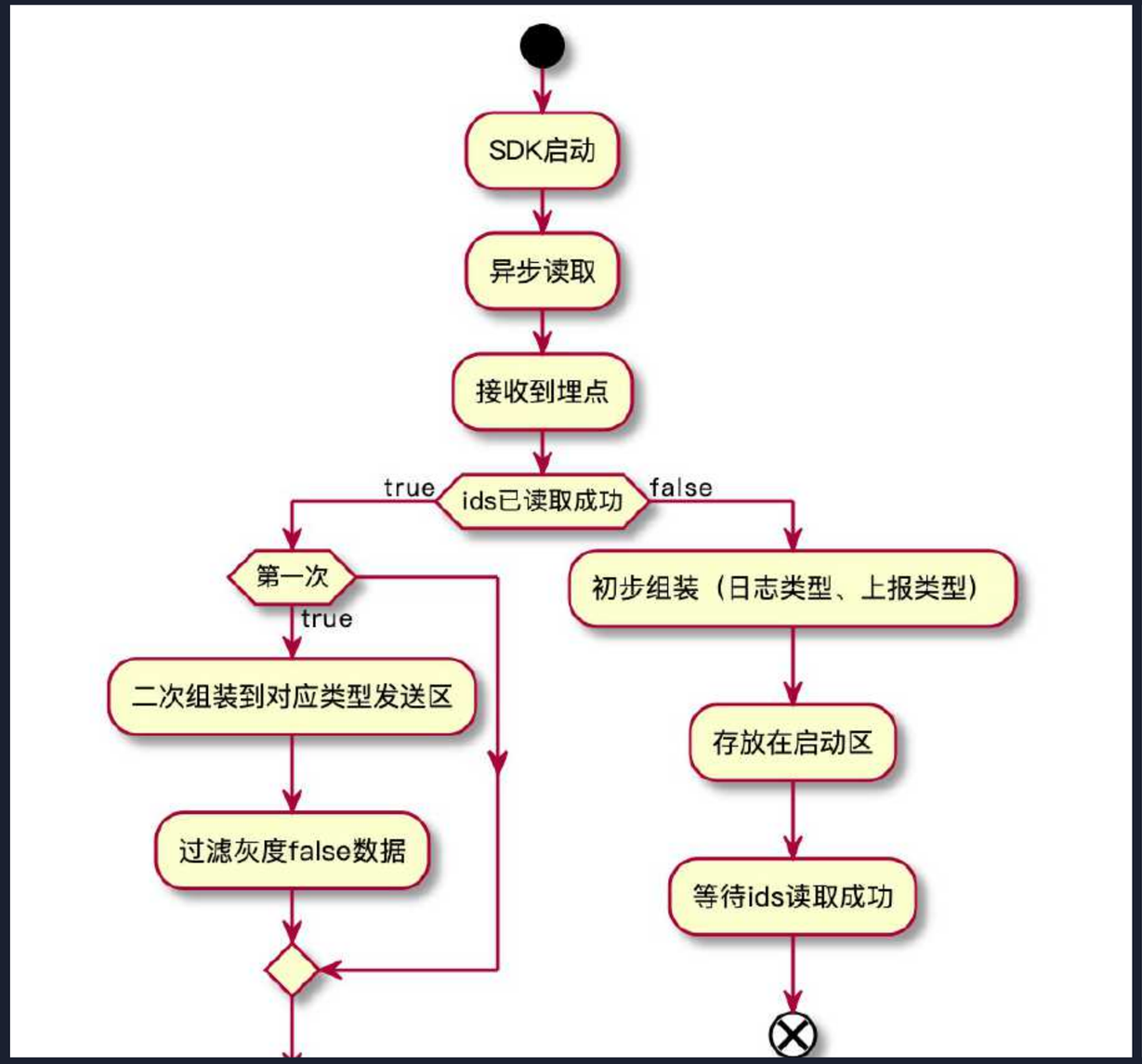
页面加载时间

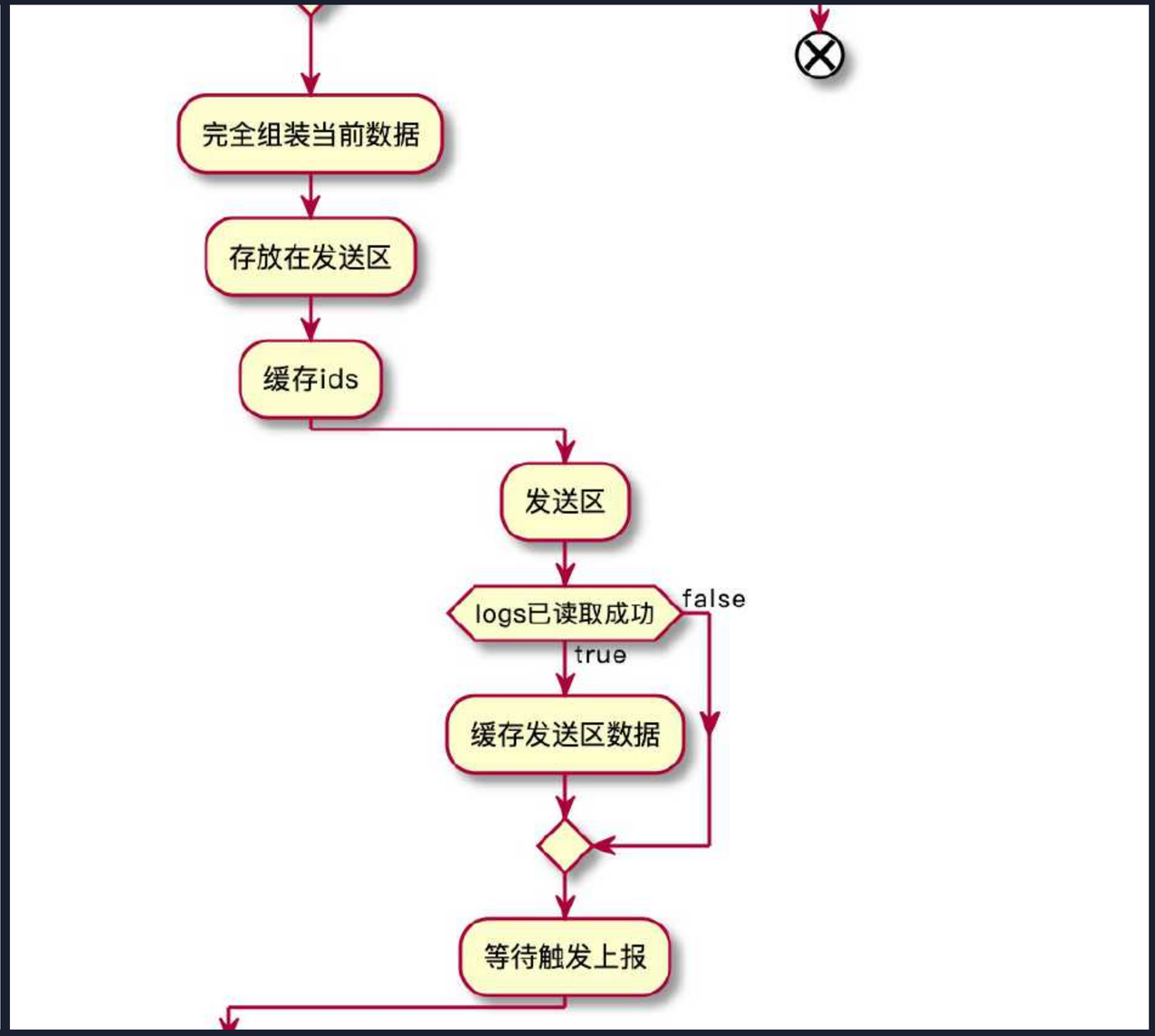
- 白屏时间：重要性得分不为0的页面渲染的时间
- 首屏时间：重要性得分最高的页面渲染的时间
- 页面加载时长：页面所有资源加载完成时间（window.onload）
- 可操作加载时间：Domready + 特定 JS 标记完成时间

数据采集 – 数据拼装

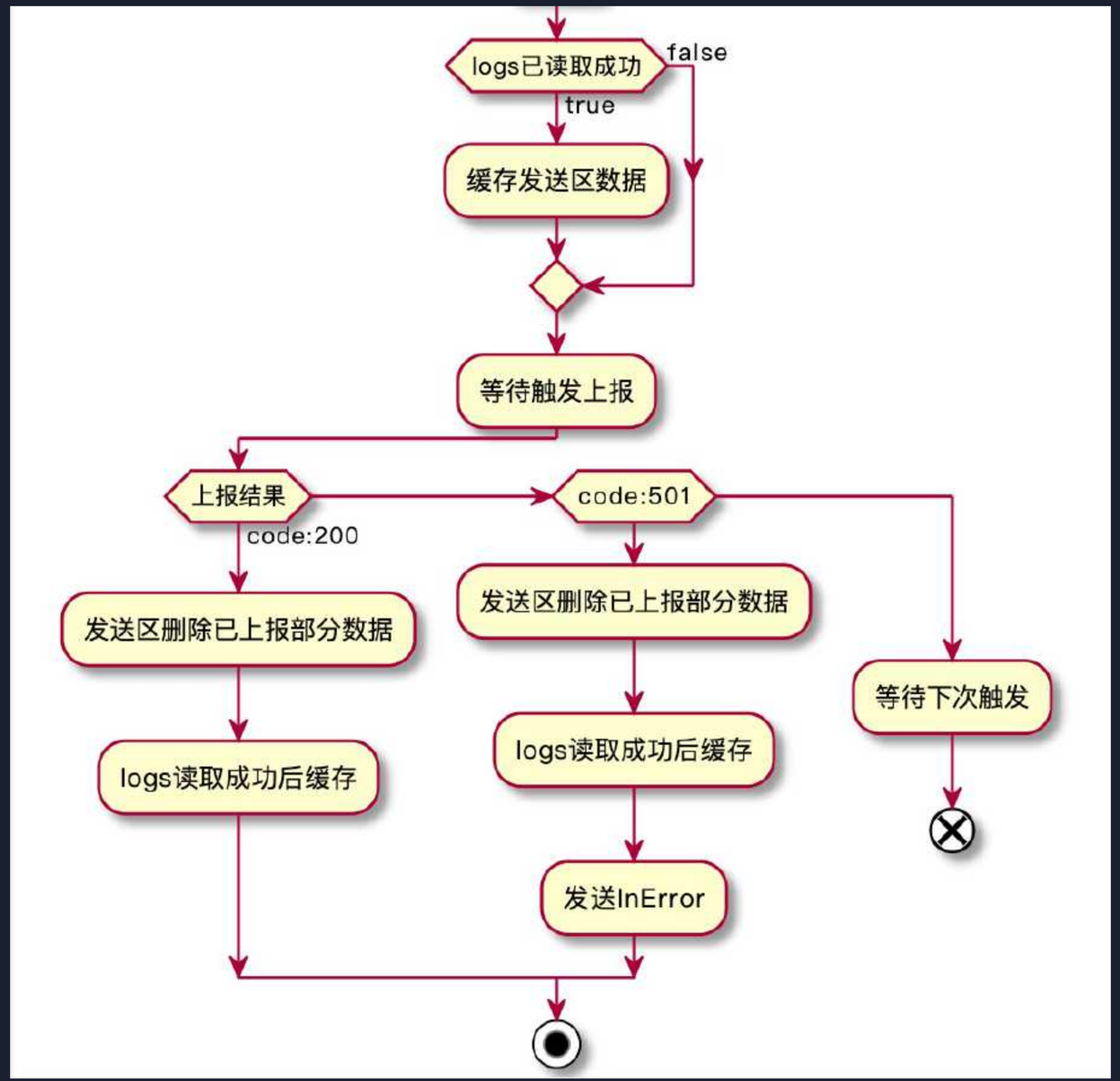

通用信息
业务信息
用户基础信息
Type
Timestamp: {
Seconds
Nanos
}
LogId
Info: {
SequenceId
BusinessId
PageSessionId
PagePath
DetailProperties{...
}
}
BaseParams: {
AppName
AppVersion
OS
OSVersion
...
}

数据采集 – 缓存&上报


SDK启动
异步读取
接收到埋点
ids已读取成功
true
false
第一次
true
二次组装到对应类型发送区
过滤灰度false数据
初步组装（日志类型、上报类型）
存放在启动区
等待ids读取成功


完全组装当前数据
存放在发送区
缓存ids
发送区
logs已读取成功
false
true
缓存发送区数据
等待触发上报

数据采集 – 缓存&上报

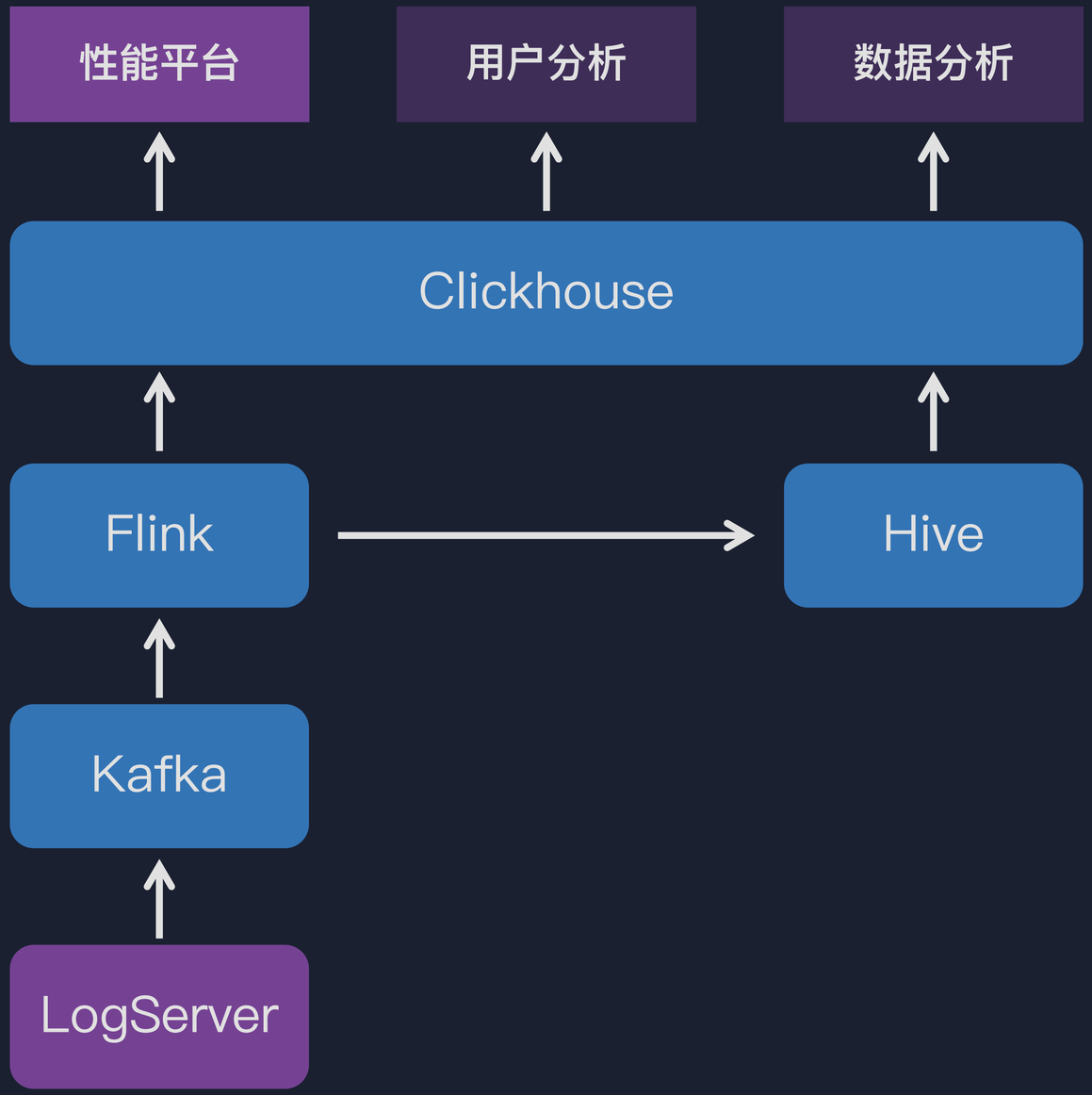

logs已读取成功
false
true
缓存发送区数据
等待触发上报
上报结果
code:501
code:200
发送区删除已上报部分数据
logs读取成功后缓存
发送区删除已上报部分数据
logs读取成功后缓存
发送InError
等待下次触发

- 每条数据写入 Storage 缓存后，判断日志数据数大于等于 logMaxCount，满足则上传
- 通过初始化时开启的定时为 maxTimeLog 定时器，定时上传
- 应用启动后，读取 localStorage 值到 Storage 中并触发上报，失败就等待下一次上传时机

哈啰出行

性能监控 – 整体系统

数据清洗 – 清洗流程



- 为什么要用 Flink + ClickHouse?
  - 速度快，性能类数据，取数计算1S内完成
  - 方案成熟，部署 + 运维简单
  - 关注指标支持 sql 描述，方便后续加工呈现
- 为什么要加 Hive?
  - 对 Flink 和 ClickHouse 做小时级别的容错

哈啰出行

性能监控 – 整体系统

数据采集

数据清洗

聚合展示

展示平台

展示服务

定制处理/清洗

哈啰出行

性能平台 – 业务架构


扁鹊
Dashboard
实时
概览
每日报告
分析工具
性能优化工具
日志查询
告警
健康分
页面报告
实时
AppBusinessTravelH5
不限业务线
Today so far
扁鹊帮助文档
赵宇
今日pv
今日uv
今日
今日
网络异常率
今日网络错误率
网络性能
今日50线
今日90线
首屏平均时间
今日

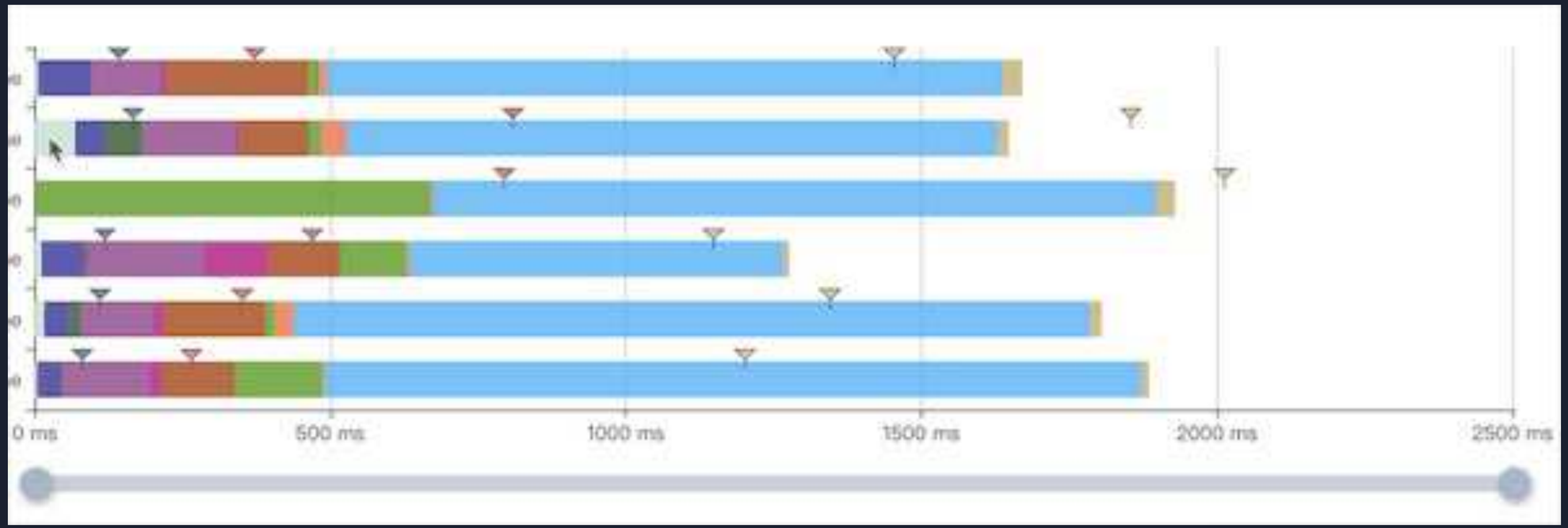
哈啰出行

难点&问题

- 信息采集采样率多少合适
- 随着接入业务方增多和业务增长，日志数据量增大，查询速度慢
- 采集信息丢失?

哈啰出行

性能优化分析



- 建连前
- 资源请求
- html文档+资源下载
- 解析DOM

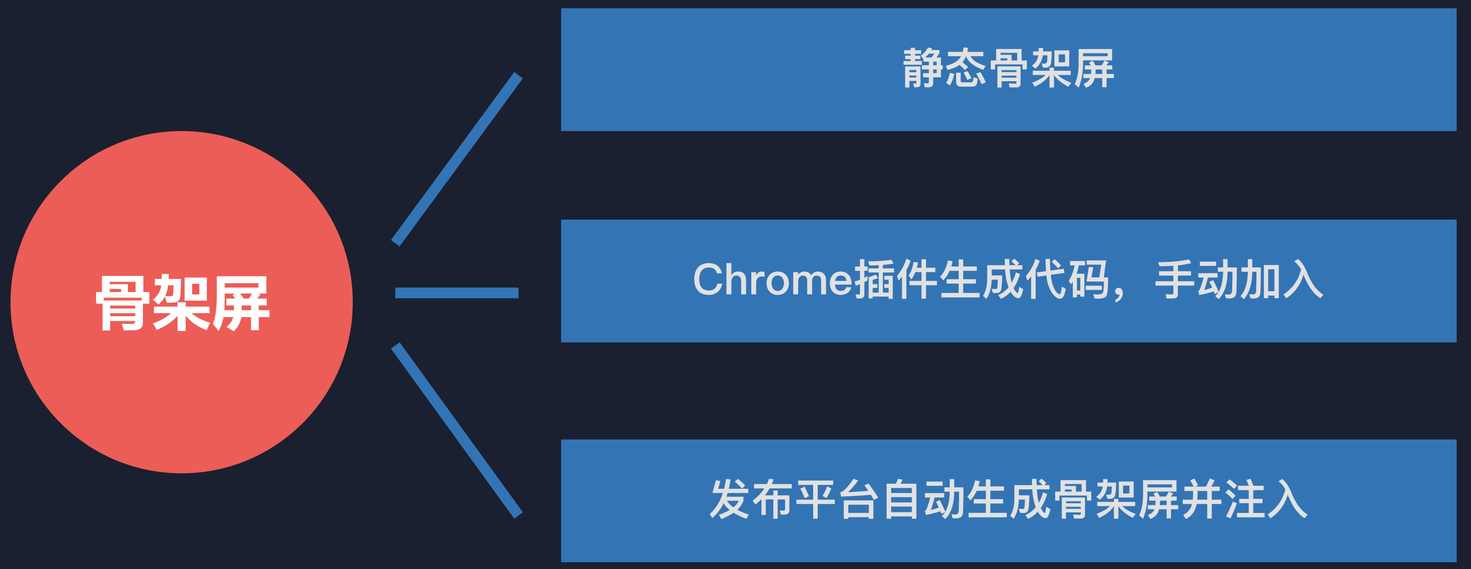
- DNS预获取
- CDN+OSS
- Webpack打包优化、离线包
- 预渲染、SSR、ESR、NSR

哈啰出行

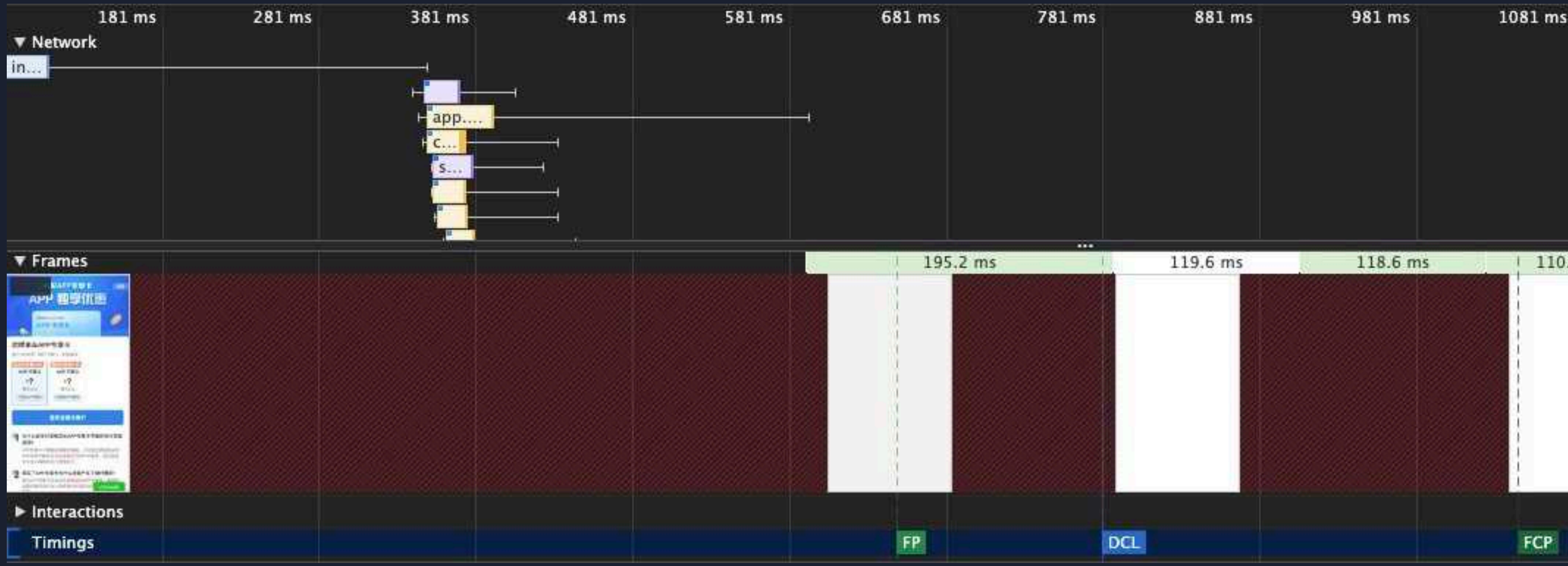
性能优化实践

- 骨架屏
- 预渲染
- 离线包

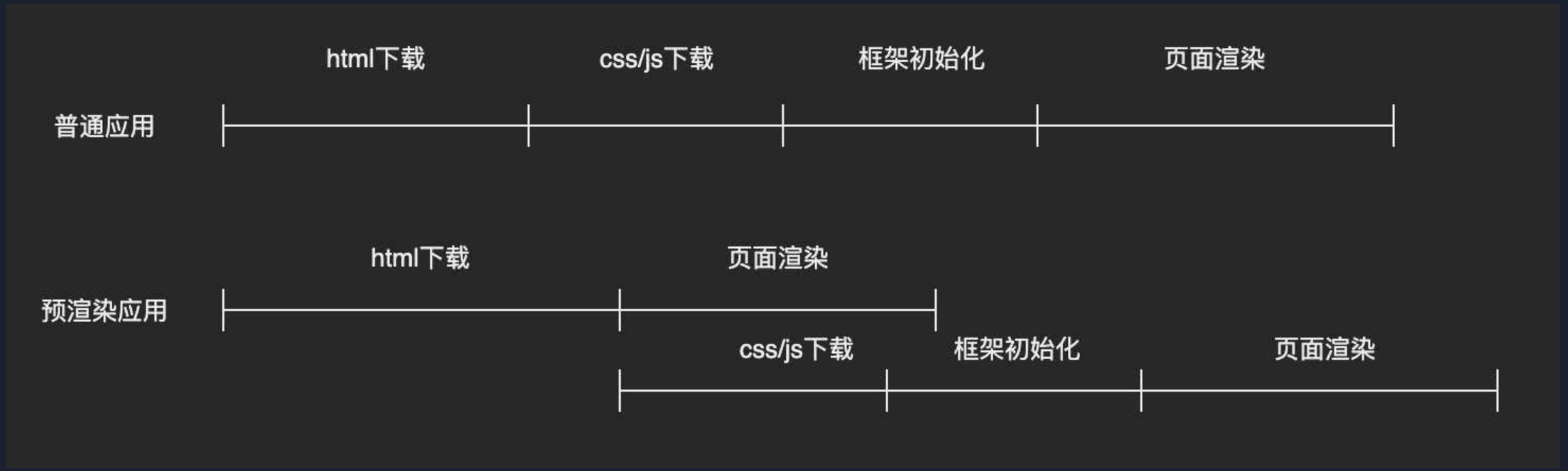
性能优化 – 骨架屏



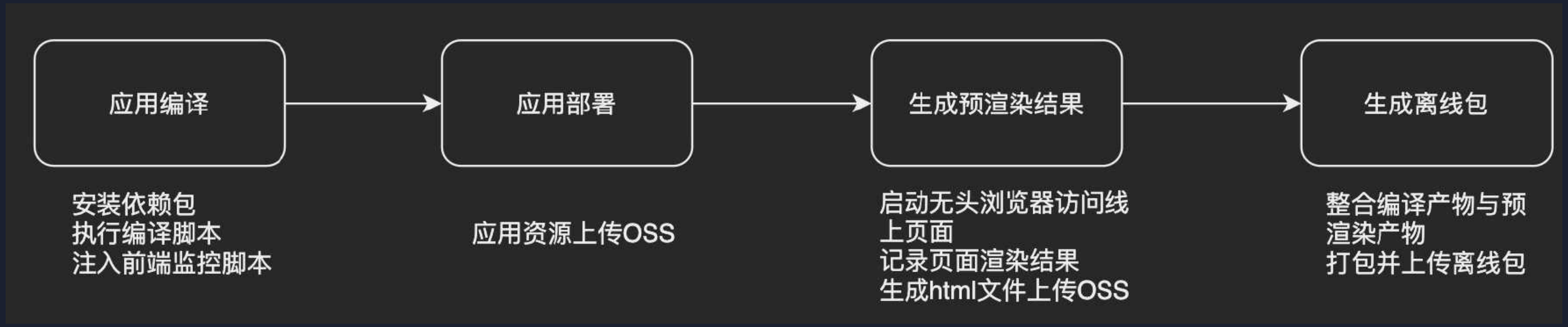
未启用预渲染：首屏时间1200ms， 有较长白屏时间

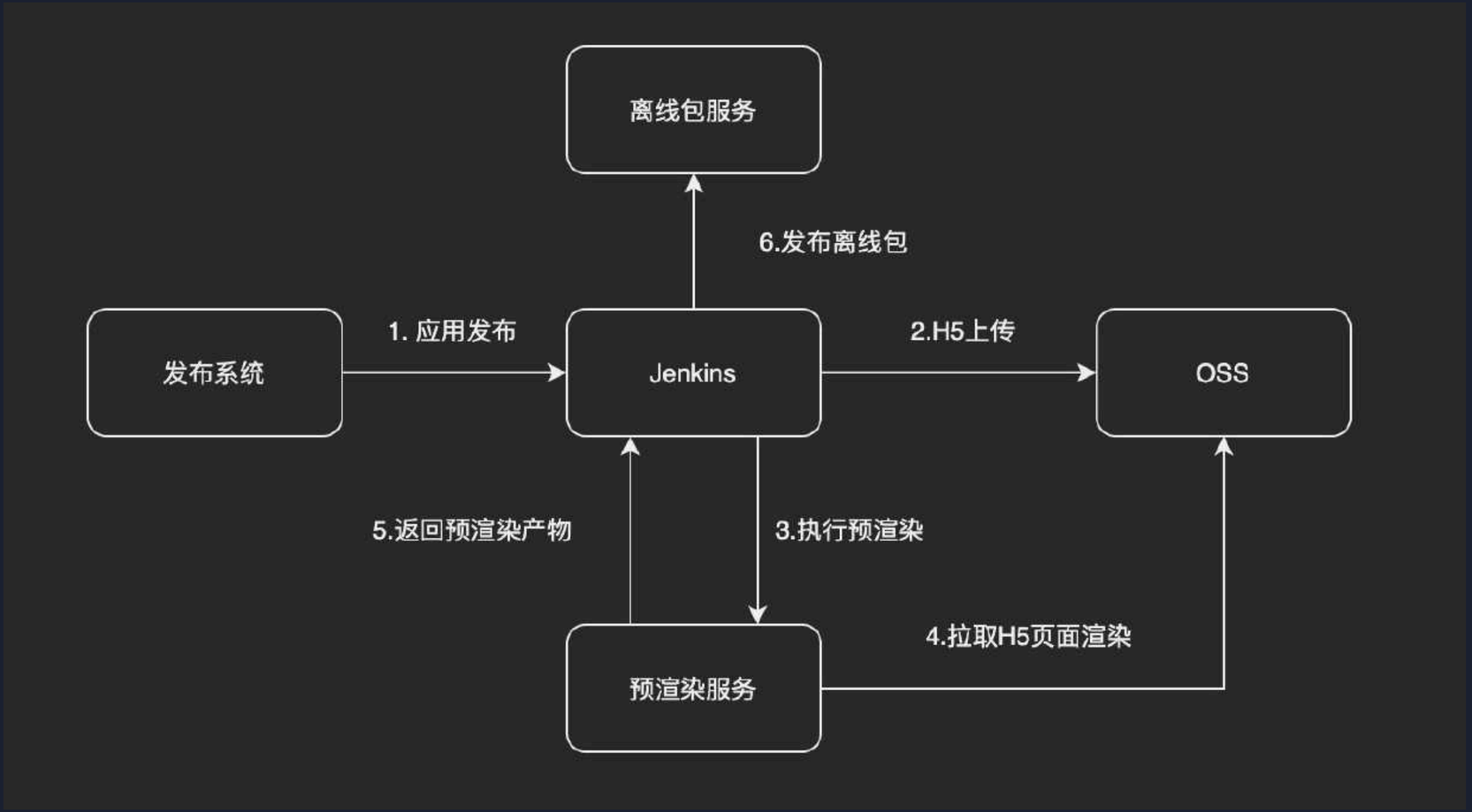


性能优化 – 预渲染


html下载
css/js下载
框架初始化
页面渲染
普通应用
html下载
页面渲染
预渲染应用
css/js下载
框架初始化
页面渲染

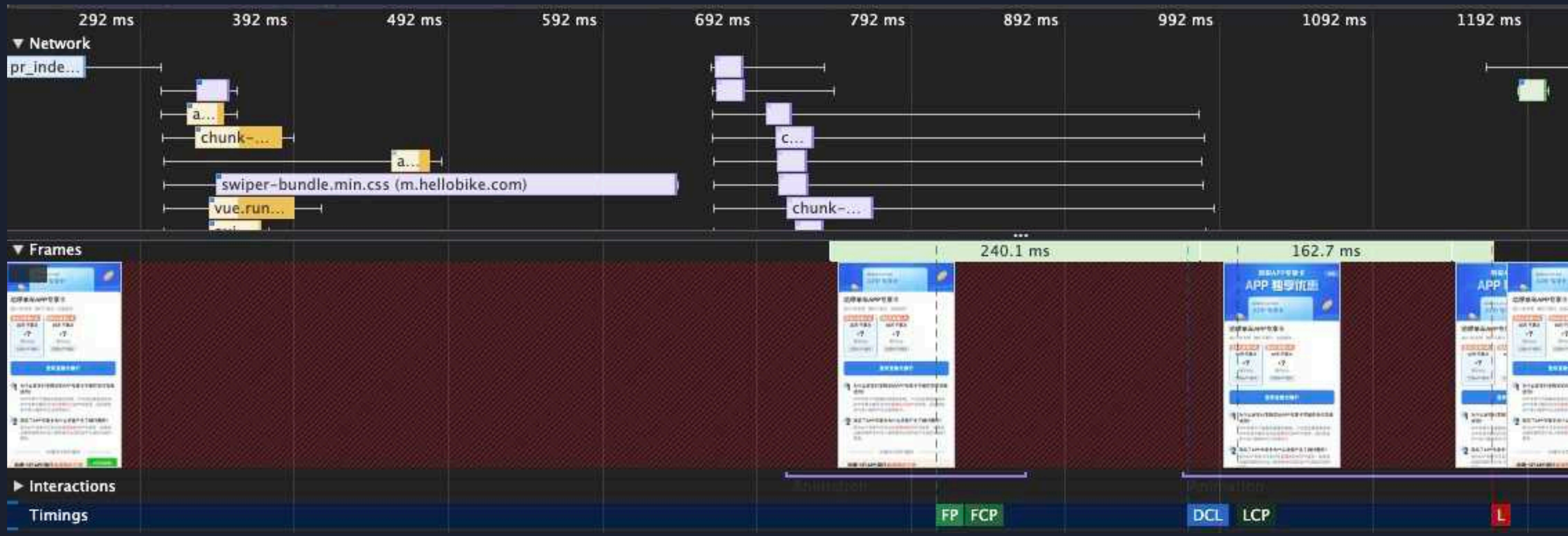
性能优化 – 预渲染

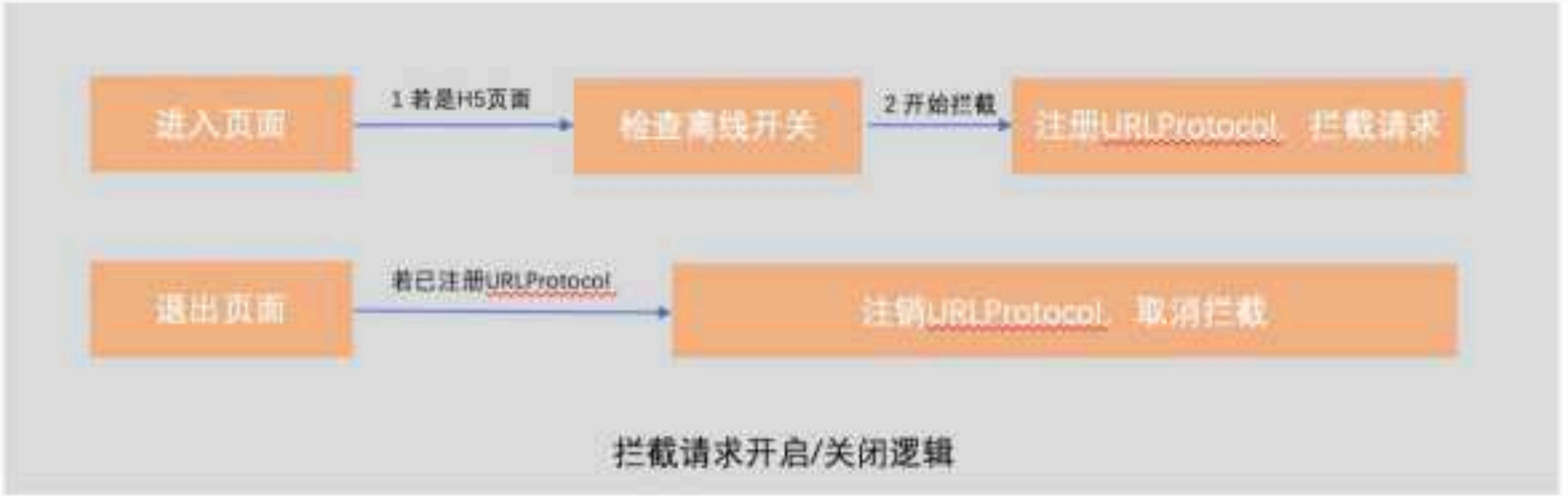

应用编译
应用部署
生成预渲染结果
生成离线包
安装依赖包
执行编译脚本
注入前端监控脚本
应用资源上传OSS
启动无头浏览器访问线上页面
记录页面渲染结果
生成html文件上传OSS
整合编译产物与预渲染产物
打包并上传离线包



- 流程和系统间的交互简单
- 可以基于发布系统做降级和质量治理
- 提升效果明显
- 没有额外负载压力和系统复杂度成本

启用预渲染：首屏时间760ms，无白屏

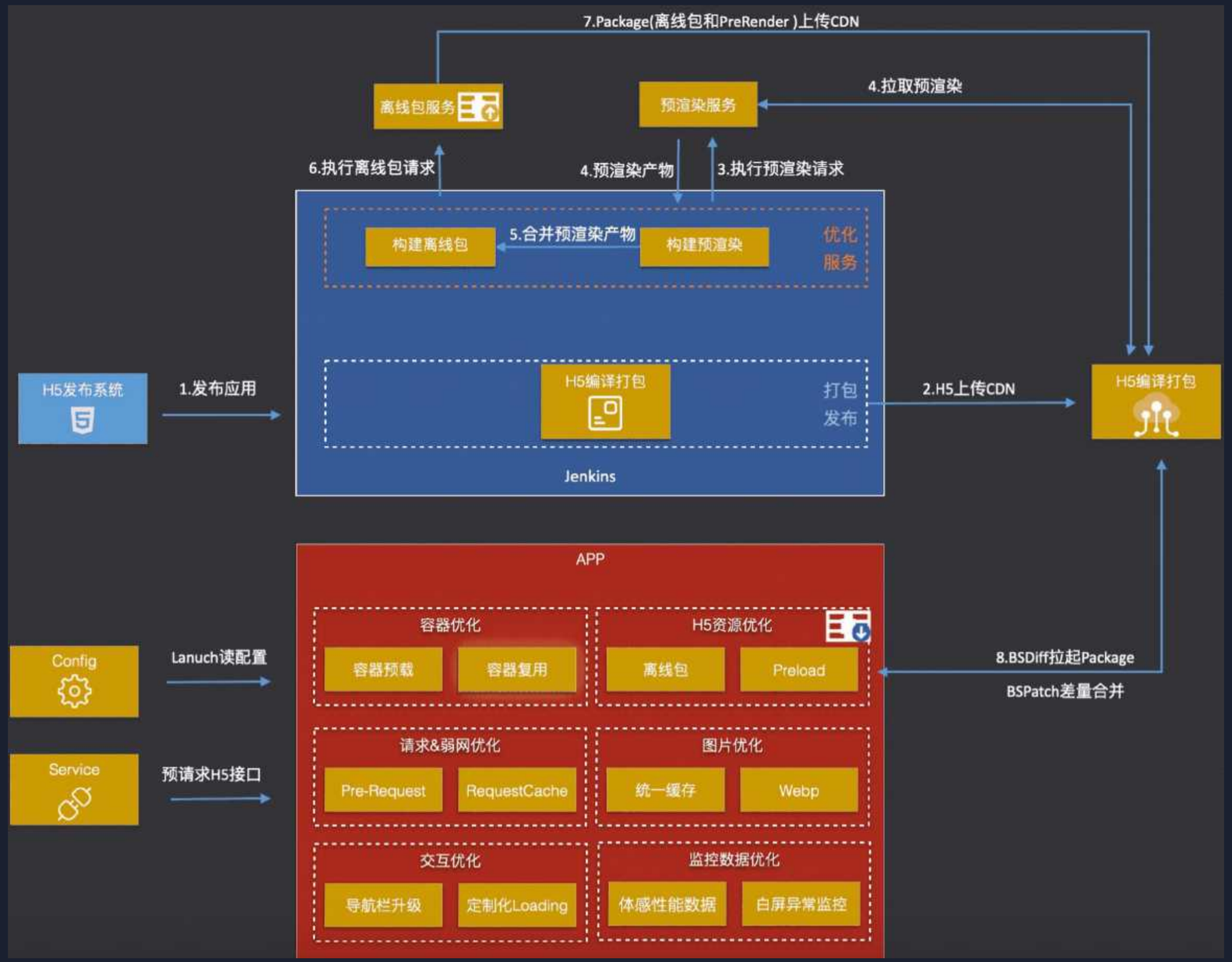




拦截请求开启/关闭逻辑

离线包资源使用逻辑

性能优化 – 离线包


7.Package(离线包和PreRender)上传CDN
离线包服务
预渲染服务
4.拉取预渲染
6.执行离线包请求
4.预渲染产物
3.执行预渲染请求
构建离线包
5.合并预渲染产物
构建预渲染
优化
服务
H5发布系统
1.发布应用
H5编译打包
打包
发布
2.H5上传CDN
H5编译打包
Jenkins
APP
容器优化
容器预载
容器复用
H5资源优化
离线包
Preload
Config
Lanuch读配置
8.BSDiff拉起Package
BSPatch差量合并
请求&弱网优化
Pre-Request
RequestCache
图片优化
统一缓存
Webp
Service
预请求H5接口
交互优化
导航栏升级
定制化Loading
监控数据优化
体感性能数据
白屏异常监控

哈啰出行

总结

- 引发性能问题的原因通常都不是单方面的，可以借助性能采集监控体系更好挖掘性能需求，跟进性能优化效果
- 前端大部分优化都逃不开三板斧: 并发、缓存、压缩资源
- 善用浏览器和客户端侧提供的各种能力，流程优化有时候强过细节优化

THANKS

QCon+ 案例研习社