[SCA] Construct end-to-end performance data indicators tech solution

Revision history

Version	Revision date	Revisor	Revision content
v1	2022.07.15	honggang	First draft
v1.1	2022.07.18	honggang	Improve summary and design

Ownership

Product Manager	*Honggang Xiong*
Native Dev	Honggang Xiong
QA	*Honggang Xiong*

Resources

Transify	*https://transify.seagroup.com*
Git Repo(Optional)	*https://git.garena.com/shopee/ssz-client/android/supplychain/fms-android*
Project Schedule Page (Optional)	*https://confluence.shopee.io/pages/viewpage.action?spaceKey=SPSC&title=%5BSCA%5D+Construct+end-to-end+performance+data+indicators+roadmap*

Summary

Background

上季度 SSC Android 端各项目陆续构建了各自的核心业务指标数据，可以在监控上查看各 App 的各项自定义指标，了解项目的大致运行情况，但整体还存在以下问题：

- 排查效率较低。指标粒度较粗，发现问题后较难快速定位问题点。
- 链路（调用链）追踪不完整。同一个业务链路的指标之间相互独立，缺乏拓扑结构，难以还原调用链路，也就无法对具体业务进行追踪。

在已有宏观指标上，结合链路追踪思想，加入初步的追踪能力，构建更细致的性能数据指标，可以有效提升我们的问题排查效率，同时有助于挖掘并优化流程痛点。

Goal

基于以上背景，我们有以下目标：

- 结合链路追踪思想，搭建一套简单的全链路数据追踪系统，用于端到端性能数据指标的收集与展示。
- 构建 Driver App、WMS、In-Station 等 App 的性能数据指标，针对高频场景（如 Camera 扫描、PDA 扫描）建立全生命周期时间分布。

Abbreviations

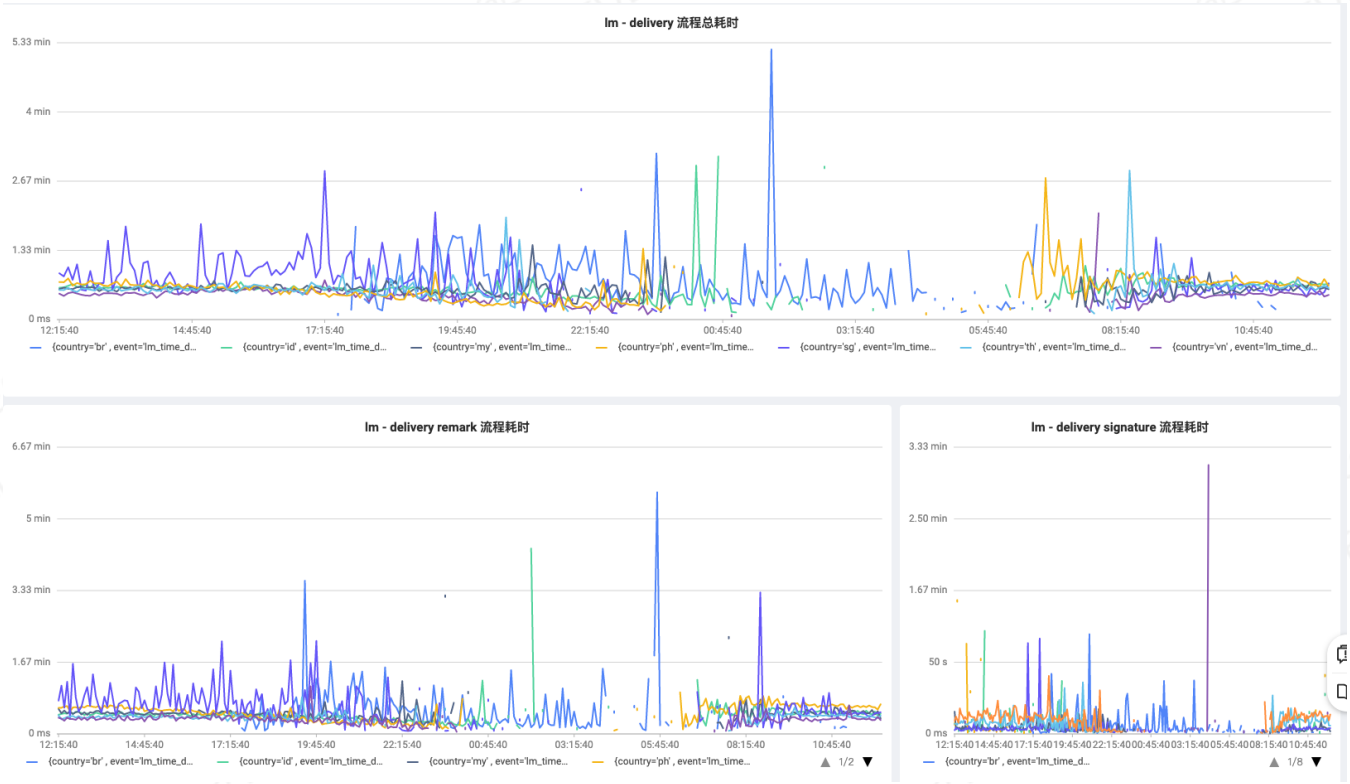
ABB	Full name	Description
MDAP	Multi-Dimension Analysis Platform	多维度分析平台
TMS	Tracking Management System	埋点管理系统

Overall Design

Brand new

2.1 已有设计及其限制

以 Driver App 尾程 delivery 流程为例，delivery 总流程包含凭证（remark）子流程、签名（signature）子流程以及其他支线子流程，已有的统计看板如下：



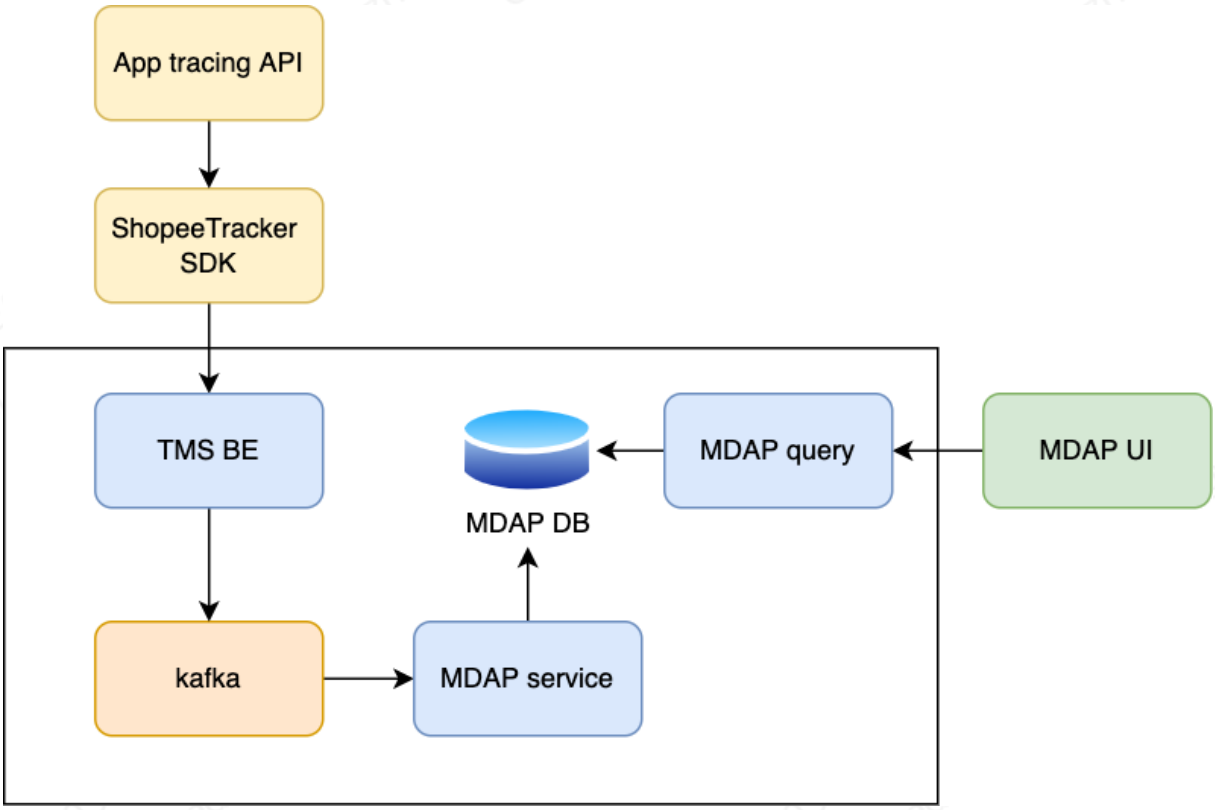
从看板中我们得到各流程在不同时间段不同市场的平均耗时，但更具体的细节我们无法得知，体现在以下两方面：

- 无法精确得知各子流程耗时对整体的影响。假设delivery 总流程平均耗时为 Tn，凭证子流程平均耗时为 T1，签名子流程平均耗时为 T2，其他流程平均耗时为 Tu，由于凭证子流程、签名子流程都是可跳过的，所以还需要假设凭证子流程触发概率为 P1，签名子流程触发概率为 P2，则有：Tn = T1 * P1 + T2 * P2 + Tu。从上图数据我们无法计算出 P1、P2，也就无法精确得知各子流程耗时对整体的影响。
- 各子流程的相对起始时间未知，难以察觉子流程之间可能存在的隐藏耗时操作。

针对已有设计存在的限制，我们希望可以对一个具体的业务流程进行追踪，即流程下的子流程之间可以建立关联关系，各子流程都有相对起始时间，如此我们可以建立更细致的性能数据指标。

2.2 整体设计

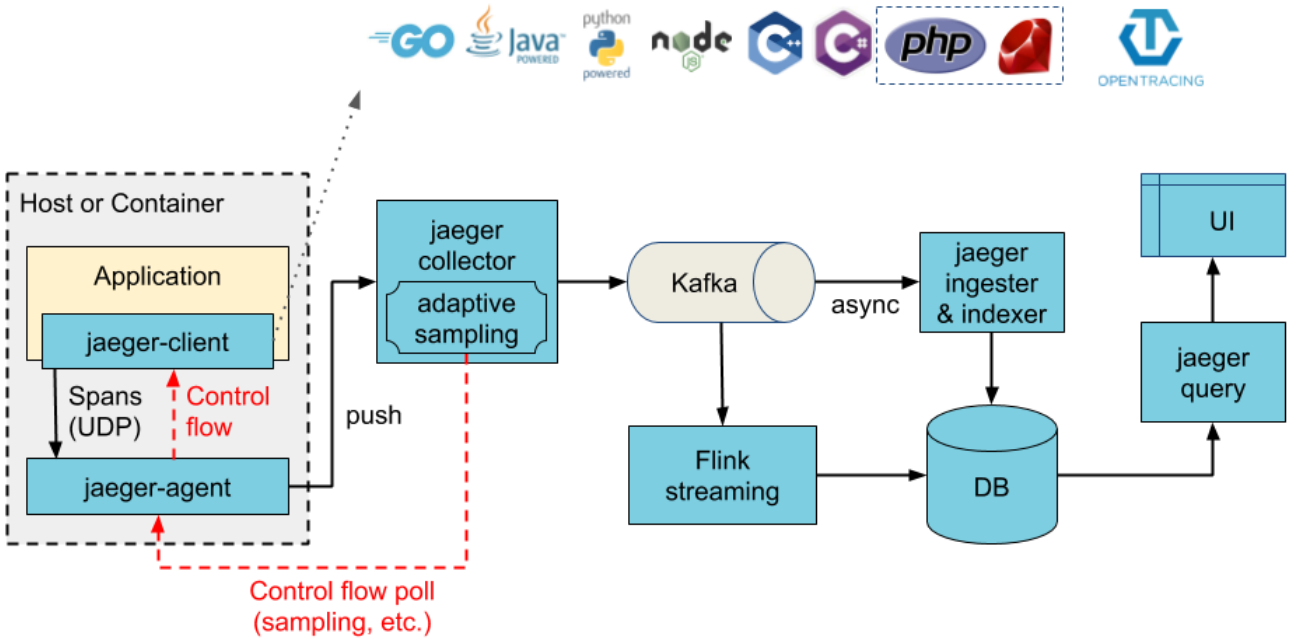
我们的目的是搭建一套端到端性能数据指标的收集与展示框架，可以结合链路追踪思想，利用 TMS 与 MDAP 平台，来搭建一套简单的全链路数据追踪系统，整体框架如下：



框架各环节都已成熟稳定，只要设计一套 Tracing API，进行一定的数据转换与还原，即可达到简单的追踪效果。即使后续需要接入分布式追踪平台，通过切换 Tracing API 实现即可。

2.3 设计来源 - OpenTracing

整体设计受 OpenTracing（分布式追踪系统）启发，看下业界比较优秀的分布式追踪系统 Jiager 的架构：



分析一下 Jaeger 的架构图，Jaeger 主要由以下几部分组成：

- jaeger client：为不同语言实现了符合 OpenTracing 标准的 SDK。应用程序通过 API 写入数据，client library 把 trace 记录按照应用程序指定的采样策略传递给 jaeger-agent。
- jaeger-agent：它是一个监听在 UDP 端口上用以接收 span 数据的网络守护进程，它会将数据批量发送给 collector。它被设计成一个基础组件，部署到所有的宿主机上。jaeger-agent 将 client library 和 collector 解耦，为 client library 屏蔽了路由和发现 collector 的细节。
- jaeger-collector：接收 jaeger-agent 发送来的数据，然后将数据写入后端存储。jaeger-collector 被设计成无状态的组件，因此可以同时运行任意数量的 jaeger-collector。
- Data Store：后端存储被设计成一个可插拔的组件，支持将数据写入 Cassandra、Elastic Search。
- jaeger-query：接收查询请求，然后从后端存储系统中检索 trace 并通过 UI 进行展示。jaeger-query 是无状态的，我们可以启动多个实例，把它们部署在 Nginx 这样的负载均衡器后面。

目前公司已经有基于 Jaeger 的分布式追踪平台，但目前都是服务端接入。就我们的目标来说，接入分布式追踪平台有点费时费力，先搭建一套简单的追踪系统优化业务问题才是当务之急。

Detailed Design

Brand new

3.1 搭建端到端性能数据指标的收集、存储、展示框架

3.1.1 数据模型

对 OpenTracing 的数据模型进行精简，主要包含以下几个概念：

- Trace：可以理解为一个完整的调用链路（流程），包含一个 Root Span 和 0 到多个子 Span。Trace 由其包含的 Spans 隐式定义，没有单独的类对象。
- Span：一个命名的子链路，表示链路中的一部分。
- SpanContext：伴随调用链路的跟踪信息，包括通过网络或消息总线在服务间传递的信息，这里仅包含 traceId 和 spanId。
- Tracer：负责启动一个新 Span。

只保留 Span 间的父子关系。对于以下 Trace：

```
       [Span A]  (the root span)
           |
    +------+------+
    |             |
[Span B]      [Span C] (Span C is a `ChildOf` Span A)
    |             |
[Span D]      +---+-------+
              |           |
          [Span E]    [Span F]
```

使用时间轴更容易可视化跟踪，如下所示：

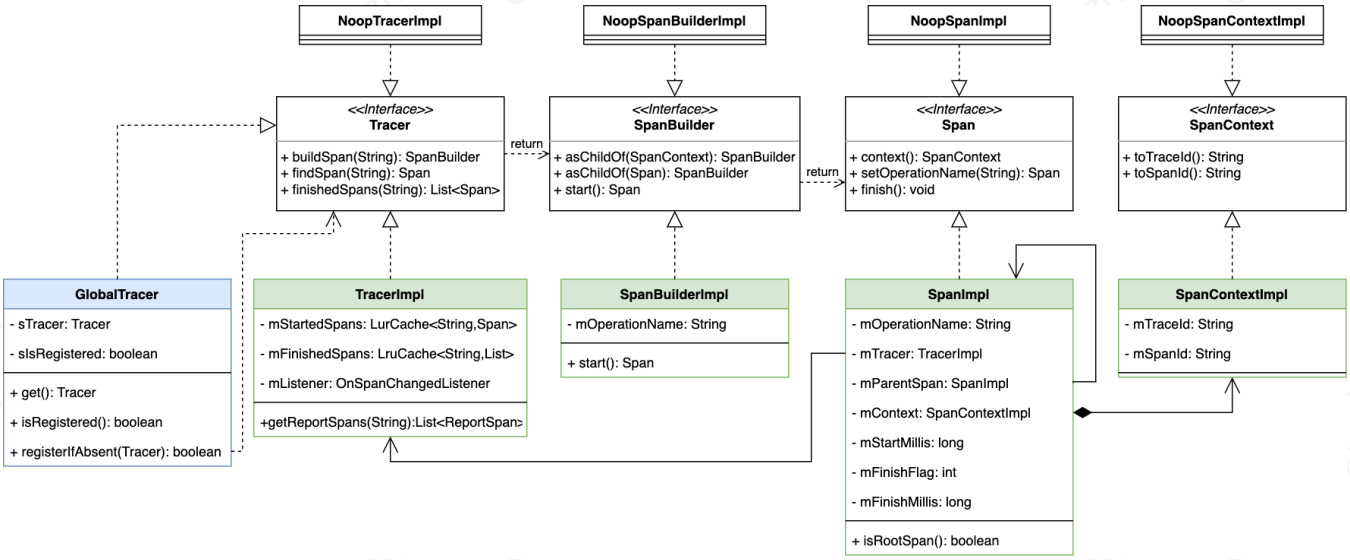
```
--|-------|-------|-------|-------|-------|-------|-------|-> time

 [Span A···················································]
   [Span B·················································]
      [Span D··········································]
    [Span C····························································]
         [Span E·······]        [Span F··]
```

每个 Span 都封装了以下属性：

- 操作名称
- 开始时间戳
- 结束时间戳
- 一个 SpanContext（当前只包含 traceId 和 spanId，保留以便后续扩展）
- 父 Span 引用，如为空则代表当前 Span 为 Root Span

3.1.2 Tracing SDK

基于上节的数据模型，设计一套精简的 Tracing SDK，类图如下：



3.1.3 数据收集与流转

通过 Tracing SDK API 收集追踪信息，缓存于内存中，每个 Trace 结束时通过 converter 将 Span 列表转换成 ShopeeTracker SDK 支持的数据格式，并将该 Trace 从内存缓存中移除，整体数据存储与流转见整体框架。

3.1.4 数据还原展示

假设 pickup 签名流程执行时序如下：

```
 [ all                                                  ]
 |·upload_pic···········|                               |
 |·upload_pop···········|                               |
 |·pop··|                                               |
 |      |·pop2··|                                       |
 |              |·pop3··|                               |
 |·upload_signature·|                                   |
 |·driver·|                                             |
 |        |·seller·|                                    |
 |                       |·call_api···········|         |
 |                                            |·save_result·|
-------------------------------------------------------------> time
```

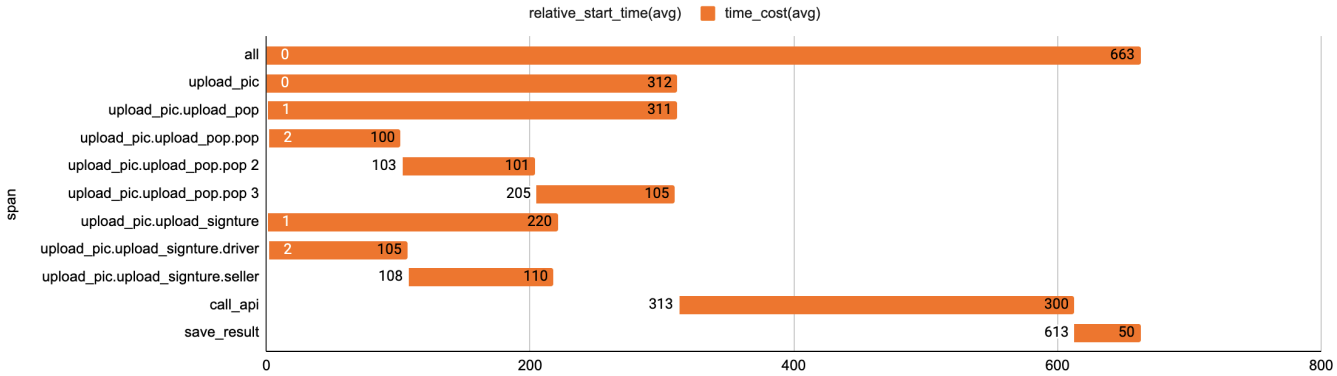
经过 Tracing 系统上报后，其在 MDAP 平台上的数据可能如下：

trace	span	event(count)	relative_start_time(avg)	time_cost(avg)	weighting(sum)	deviceid(distinct)
pickup_sign	all	10	0	663	10	10
pickup_sign	upload_pic	10	0	312	10	10
pickup_sign	upload_pic.upload_pop	10	1	311	10	10
pickup_sign	upload_pic.upload_pop.pop	10	2	100	10	10
pickup_sign	upload_pic.upload_pop.pop 2	10	103	101	10	10
pickup_sign	upload_pic.upload_pop.pop 3	10	205	105	10	10
pickup_sign	upload_pic.upload_signture	10	1	220	10	10
pickup_sign	upload_pic.upload_signture.driver	10	2	105	10	10
pickup_sign	upload_pic.upload_signture.seller	10	108	110	10	10
pickup_sign	call_api	10	313	300	10	10
pickup_sign	save_result	10	613	50	10	10

原始数据本身已经有一定的可读性，细读可以获悉所有信息，经过简单数据还原，可以更加直观的查看实际分布。有以下两种还原方式：

1. 简单还原：导出 MDAP 中的数据至 Excel，生成堆积条形图，即可查看简单的时间分布图（MDAP 默认为字母序，可能还需要手动摆放 Span 位置）

模拟时间分布图



2. 精细还原：导出 MDAP 中的数据至 Excel，运行脚本生成精细的时间分布图（有一定的开发工作）

3.2 业务用例

3.2.1 Camera 扫描全生命周期时间分布

见 [SCA] Performance data indicators business case - Camera Scan

3.2.2 PDA 扫描全生命周期时间分布

见 [SCA] Performance data indicators business case - PDA Scan

3.2.3 Driver APP 端到端性能数据

见 [SCA] Performance data indicators business case - Driver App

3.2.4 OPS APP 端到端性能数据

见 [SCA] Performance data indicators business case - In-station

3.2.5 WMS APP 端到端性能数据

见 [SCA] Performance data indicators business case - WMS

Interface Design

4.1 数据收集接口设计

Tracer 接口设计：

```
public interface Tracer extends Closeable {

    SpanBuilder buildSpan(String operationName);

    @Override
    void close();

    interface SpanBuilder {

        SpanBuilder asChildOf(Span parent);

                SpanBuilder asRoot();

        Span start();
    }
}
```

Span 接口设计：

```
public interface Span {

        String getTraceId();

         String getSpanId();

        void finish();

    boolean isFinished();

    String getOperationName();
}
```

收集接口用法如下：

```
    Tracer tracer = ...;
    Span span = tracer.buildSpan("someWork").start();
    Span childSpan = trace.buildSpan("someChildWork").asChildOf(span).start();
    childSpan.finish();
    span.finish();
```

4.2 数据上报接口设计

4.2.1 指标数据 metric、dimension 设计

通过总结 Driver App 当前以及未来可能新增的追踪事务，提取了一套通用的 metric、dimension。

通用 metric：

metric Name	Path	Aggregation method	Description
trace(count)	data.data.trace	count	事件发生数量
time_cost	data.extra.time_cost	avg	事件平均耗时，单位 ms
relative_start_time	data.extra.rs_time	avg	时间平均相对起始时间，单位 ms
weighting	data.extra.weighting	sum	事件权重，代表事件影响的实体数，对于抽样事件也可以据此估算出样本总数
deviceid	deviceid	distinct	事件涉及的独立用户数，可以借此计算出用户平均指标数据

通用 dimension：

dimension Name	Path	Nullable	Description
country	country	Non Null	市场维度
trace	data.data.trace	Non Null	事件 Trace name(不同于 metric 中的 trace(count))
span	data.data.span	Non Null	事件 Span name
user_type	data.extra.ut	Nullable	用户类型，可以用来做 AB test (有上报未启用)
app_verison	app_verison	Non Null	App 版本
time_interval	data.extra.time_interval	Non Null	整个 trace 的耗时区间 tag，用来做整体流程分析
self_time_interval	data.extra.self_time_interval	Non Null	span 自身的耗时区间 tag，用来做具体流程分析

4.2.2 通用数据上报接口设计

数据通过 Shopee tracker sdk 进行上报，上报 url 为 https://c-api-bit.{env.}shopeemobile.com/{cid}/pf，上报数据结构如下：

```
[
    {
        "app_id": 57,
        "app_version": "5.5.2",
        "brand": "Xiaomi",
        "country": "id",
        "data": {
            "data": {
                "trace":"lm_home_scan",
                "span": "call_api"
            },
            "data_type": "5704",
            "extra": {
                "weighting": 1,
                "time_cost": 1000,
                "rs_time": 0,
                "ut": "lm",
                "time_interval": "2-3",
                "self_time_interval": "1-2"
             }
        },
        "deviceid": "0e82debb3c18c453",
        "event_id": "e7c05981-0ac0-4ed8-9244-c482177f7d33",
        "event_timestamp": 1651828996123,
        "model": "M2011K2C",
        "network_type": 1,
        "os": "android",
        "os_version": "31",
        "platform": "android",
        "rn_version": "",
        "type": 5701,
        "userid": 511,
        "version": "v1"
    }
]
```

Data track

The data you wanna track in Firebase or Lumos or any other data platform

同 3.2 中新增的性能数据

Monitoring

The matrix of the data you report, and the rule for alarm

同 3.2 中新增的性能数据，暂无需告警

MileStones

M1 : 数据指标技术方案

Content :

1. 端到端性能数据指标整体框架、PRD - 0715

2. Camera 扫描数据指标完整性构建方案 - 0722
3. PDA 扫描数据指标完整性构建方案 - 0722
4. Driver、WMS、In-Station 性能数据指标完整性构建方案 - 0729

Effort:

1. 0729 完成各方案设计

M2 : Camera 扫描优化技术方案

Content :

1. Camera 扫描组件优化
2. Camera 解码流程优化

Effort:

1. 7.29 完成扫描组件优化技术方案，8.17 完成解码流程优化技术方案

M3 : 数据指标功能实现

Content :

1. Camera 扫描数据指标完整性构建
2. PDA 扫描数据指标完整性构建
3. Driver、WMS、In-Station 性能数据指标完整性构建

Effort:

1. 8月初完成开发自测，8.10 上线 Camera、PDA 扫描数据指标，9.9 前上线各项目性能数据指标

M4 : Camera 扫描优化实现

Content :

1. Camera 扫描组件优化
2. Camera 解码流程优化

Effort:

1. 8月底完成开发自测，9.14 Go-Live

Tickets

Task content	PIC	JIRA Ticket

zhi.xiao@shopee.com / 2023-03-02 17:43:42 CONFIDENTIAL © Sea Limited