一、实现前的共识

当前凤凰的全局埋点需要是基于业务的，不是基于技术上的方法调用日志记录。
因此采用服务端SDK打点方式埋点，在业务上敏感的关键流程上记录信息，最终收集这些信息并分组展示。

二、一些概念

1、名词

- TraceStep 追踪步骤（分一级步骤、二级步骤、三级步骤）
  - stepNo 步骤号（每个步骤都是经过产品经理定义的，都是独一无二的）
  - stepName 步骤名称（界面展示的链路步骤名称、后期可做国际化）
  - parentStepNo 父步骤名称（因为步骤有层级，所以得关联）
- TraceSubject 链路主体（作业单、任务、机器人动作）
  - subjectNo 链路主体号（作业单号、任务号、动作消息ID）
  - parentNo 链路主体的父主体号（任务的作业单号、动作的任务号）
- TraceLog 链路主体的链路日志（每个步骤埋点产生的一条日志数据）
  - logContent 步骤日志的内容或长或短，实际实现的时候单独存储了

2、日志收集类别

- 接收请求（只记录时间和报文）
- 跨服务请求
  - 请求时间和请求报文
  - 响应时间和响应报文
- 逻辑执行（大多有耗时记录）
  - 开始时间和参数信息
  - 结束时间和结果信息
- 节点记录（时间和内容）

三、埋点集成的方式

1、依赖引入

SDK的形式引入到工程中

```
<dependency>
    <groupId>com.kc.phoenix</groupId>
    <artifactId>phoenix-trace-client</artifactId>
    <version>3.2.3</version>
</dependency>
```

2、收集日志的代码

收集哪里日志在代码行中后端埋点

```
TL.subject("BM_SI_166132480842800026")
  .parent("SIRack_166132508128900027") // 至少保证上报一次
  .step(RmsTraceStep.JOB_ACCEPT)
  .success(true)
  .contentType("REQUEST_JSON|RESPONSE_JSON|GENERAL_JSON|GENERAL_STRING")
  .contentBody("{}")
  .timeType("REQUEST_TIME|RESPONSE_TIME|TIMECOST_DURATION|TIMECOST_RANGE|INVOKE_START_TIME|INVOKE_END_TIME")
  .timeValue(1661505856182)
  .end();
```

> *当timeType为TIMECOST_RANGE的时候左41位存开始时间、后23位存间隔时间*
> *2038-12-31 23:59:59: 2,177,423,999,000*
>
> *Long64位，41位最大：2,199,023,255,552*
> *Long64位，23位最大：8388,608（2.33个小时）*

3、收集日志之·接收请求·

```
TL.subject("BM_SI_166132480842800026")
  .step(RmsTraceStep.JOB_ACCEPT)
  .contentType("REQUEST_JSON")
  .contentBody("{}")
  .timeType("REQUEST_TIME")
  .timeValue(1661505856182)
  .end();
```

各服务可封装后变更写法：

```
PTL.acceptJob("BM_SI_166132480842800026")
  .acceptTime(1661505856182)
  .jobContent("{}")
  .end();
```

4、收集日志之·数据下发·

```
TL.subject("BM_SI_166132480842800026")
  .step(RssTraceStep.JOB_SEND)
  .contentType("REQUEST_JSON")
  .contentBody("{}")
  .timeType("REQUEST_TIME")
  .timeValue(1661505856182)
  .end();
jobService.submitJob({});
TL.subject("BM_SI_166132480842800026")
  .step(RssTraceStep.JOB_SENT)
  .contentType("RESPONSE_JSON")
  .contentBody("{}")
  .timeType("RESPONSE_TIME")
  .timeValue(1661505856282)
  .end();
```

各服务可封装后变更写法：

```
PTL.jobSubmitStart("BM_SI_166132480842800026")
  .requestTime(1661505856182)
  .requestBody("{}")
  .end();
jobService.submitJob({});
PTL.jobSubmitFinish("BM_SI_166132480842800026")
```

```
    .responseTime(1661505856282)
    .responseBody("{}")
    .end();
```

四、各模块链路步骤定义

每个链路步骤都有一个唯一的编码号，由于整形存储占用空间小，因此stepNo采用整形存储，且使用对每位的值制定规则来区分各个链路步骤；设计上非叶子结点的显示顺序都是固定，叶子结点的显示顺序由实际日志的时间来决定。

1、phoenix-rms

1)、任务链路步骤10位长度值划分

第几位	值	描述	备注
1–2位	01	rms模块	
3–4位	00	区分任务固定00	job(00)和action(>00)
5位	0~9	一级步骤区分	任务接收(1)和任务处理(2)
6位	0~9	二级步骤	
7位	0~9	三级步骤	请求路径规划、下发动作（分类少）
8–10位	00	预留、暂不使用、都为0	

2)、任务链路步骤stepNo定义

一级步骤	二级步骤	三级步骤	stepNo
(1) 任务接收	———————	———————	0100–1–00000
(1) 任务接收	(11) RMS接收到任务请求	———————	0100–1–1–0000
(1) 任务接收	(11) RMS接收到任务请求	(111) 接收任务下发	0100–1–1–1000
(1) 任务接收	(11) RMS接收到任务请求	(112) 接收任务取消	0100–1–1–2000
(1) 任务接收	(11) RMS接收到任务请求	(112) 接收任务暂停	0100–1–1–3000
(1) 任务接收	(11) RMS接收到任务请求	(112) 接收任务恢复	0100–1–1–4000
(1) 任务接收	(12) 机器人状态检查	———————	0100–1–20000
(1) 任务接收	(12) 机器人状态检查	(121) 当前是否可以作业	0100–1–2–1000
(1) 任务接收	(13) 落库	———————	0100–1–30000

(1) 任务接收	(13) 落库	(0131) 任务保存	0100–1–3–1000
(2) 任务处理	–––––––	–––––––	0100–2–00000
(2) 任务处理	(21) 选取任务	–––––––	0100–2–10000
(2) 任务处理	(21) 选取任务	(211) 选取好的任务	0100–2–1–1000
(2) 任务处理	(22) 任务执行	–––––––	0100–2–20000
(2) 任务处理	(22) 任务执行	(221) 选取第一个动作	0100–2–2–1000
(2) 任务处理	(22) 任务执行	(221) 恢复执行动作	0100–2–2–2000
(2) 任务处理	(22) 任务执行	(222) 选取下一个动作	0100–2–2–3000
(2) 任务处理	(22) 任务执行	(222) 所有动作执行完毕	0100–2–2–4000
(2) 任务处理	(23) 任务完成	–––––––	0100–3–10000
(2) 任务处理	(23) 任务完成	(231) 处理完成开始	0100–2–3–1000
(2) 任务处理	(23) 任务完成	(232) 任务状态更新	0100–2–3–2000
(2) 任务处理	(23) 任务完成	(233) 任务事件上报	0100–2–3–3000
(2) 任务处理	(23) 任务完成	(234) 开始下个任务	0100–2–3–4000

3)、动作链路步骤10位长度值划分

第几位	值	描述	备注
1–2位	01	rms模块	
3位	0~9	区分各种驱动插件机型	general(0)、carrier(1)、forklift(2)、lithium(3)
4–6位	000~999	区分任务模版	不使用000，支持999个任务模版，系统定制的任务模版使用第0xx位，剩下900个
7–8位	00	二级步骤	请求路径规划、下发动作（分类少）
9–10位	00	三级步骤	中间事件、动作结果上报（分类少）

4)、动作链路步骤stepNo定义

> *以通用机型general(10)和通用移动MOVE(001)举例*
> *又比如潜伏式carrier(11)和通用充电CHARGE(002)*
>
> *动作的步骤定义和作业单或任务不同；链路排查工具的左侧在动作这块展示的是动作名称，*
> *而作业单和任务都展示的是一级节点，因此动作需要把名称定义出来方便展示区分（充当了一级节点），*
> *便有了每个动作模版配合一套动作链路步骤的定义。*

一级步骤	二级步骤	三级步骤	stepNo
(001) 机器人动作	-------	-------	01-0-001-0000
(001) 机器人动作	(01) 开始处理	-------	01-0-001-01-00
(001) 机器人动作	(01) 开始处理	(01) 待执行的动作	01-0-001-01-01
(001) 机器人动作	(01) 开始处理	(02) 待恢复的动作	01-0-001-01-02
(001) 机器人动作	(02) 请求路径规划	-------	01-0-001-02-00
(001) 机器人动作	(02) 请求路径规划	(01) RMS >> RTS(request)	01-0-001-02-01
(001) 机器人动作	(02) 请求路径规划	(02) RMS >> RTS(response)	01-0-001-02-02
(001) 机器人动作	(03) 下发动作	-------	01-0-001-03-00
(001) 机器人动作	(03) 下发动作	(01) RMS >> QSH(request)	01-0-001-03-01
(001) 机器人动作	(03) 下发动作	(02) RMS >> QSH(response)	01-0-001-03-02
(001) 机器人动作	(04) 执行动作	-------	01-0-001-04-00
(001) 机器人动作	(04) 执行动作	(01) 中间事件	01-0-001-04-01
(001) 机器人动作	(04) 执行动作	(02) 动作结果上报	01-0-001-04-02
(001) 机器人动作	(04) 执行动作	(03) 外设交互	01-0-001-04-03
(001) 机器人动作	(05) 动作结束	-------	01-0-001-05-00
(001) 机器人动作	(05) 动作结束	(01) 处理结束开始	01-0-001-05-01
(001) 机器人动作	(05) 动作结束	(02) 保存动作统计	01-0-001-05-02

(001) 机器人动作	(05) 动作结束	(03) 动作状态更新	01–0–001–05–03
(001) 机器人动作	(05) 动作结束	(04) 动作事件上报	01–0–001–05–04
(001) 机器人动作	(05) 动作结束	(05) 开始下个动作	01–0–001–05–04

2、phoenix-rss-carrier

1)、链路步骤10位长度值划分

第几位	值	描述	备注
1–2位	02	rss模块	
3–4位	01	carrier模块	比如：02为forklift
5–6位	01	一级步骤区分	01:接收作业单，02:分车，03:下发任务
7位	0~9	二级步骤	
8位	0~9	三级步骤	
9–10位	00	预留、暂不使用	

2)、链路步骤stepNo定义

一级步骤	二级步骤	三级步骤	stepNo
(01) 接收作业单	-------	-------	0201–01–0000
(01) 接收作业单	(011) 接口平台接收下发请求	-------	0201–01–1–000
(01) 接收作业单	(011) 接口平台接收下发请求	(0111) Interface接收	0201–01–1–1–00
(01) 接收作业单	(012) 接口平台转发请求至RSS	-------	0201–01–2–000
(01) 接收作业单	(012) 接口平台转发请求至RSS	(0121) Interface >> RSS(request)	0201–01–2–1–00
(01) 接收作业单	(012) 接口平台转发请求至RSS	(0122) Interface >> RSS(response)	0201–01–2–2–00
(01) 接收作业单	(013) RSS接收到接口平台请求	-------	0201–01–3–000
(01) 接收作业单	(013) RSS接收到接口平台请求	(0131) RSS接收	0201–01–3–1–00
(01) 接收作业单	(014) 落库	-------	0201–01–4–000
(01)	(014)	(0141)	

接收作业单	落库	RSS内部处理	0201–01–4–1–00
(02) 分车	–––––––	–––––––	0201–02–0000
(02) 分车	(021) 作业单–分车	–––––––	0201–02–1–000
(02) 分车	(021) 作业单–分车	(0211) 分车结果	0201–02–1–1–00
(03) 下发任务	–––––––	–––––––	0201–03–0000
(03) 下发任务	(031) 作业单–下发任务	–––––––	0201–03–1–000
(03) 下发任务	(031) 作业单–下发任务	(0311) RSS >> RMS(request)	0201–03–1–1–00
(03) 下发任务	(031) 作业单–下发任务	(0311) RSS >> RMS(response)	0201–03–1–2–00
(04) 作业单完成	–––––––	–––––––	0201–04–0000
(04) 作业单完成	(041) 作业单状态处理	–––––––	0201–04–1–000
(04) 作业单完成	(041) 作业单状态处理	(0411) RSS接收事件	0201–04–1–1–00
(04) 作业单完成	(041) 作业单状态处理	(0412) 更新货架位置	0201–04–1–2–00
(04) 作业单完成	(041) 作业单状态处理	(0413) 更新作业单状态	0201–04–1–3–00
(04) 作业单完成	(041) 作业单状态处理	(0414) 上报MQ事件给上游	0201–04–1–4–00
(04) 作业单完成	(041) 作业单状态处理	(0415) 作业单处理完成	0201–04–1–5–00
(04) 作业单完成	(042) 接口平台上报上游	–––––––	0201–04–2–000
(04) 作业单完成	(042) 接口平台上报上游	(0421) 接口平台接收MQ消息	0201–04–2–1–00
(04) 作业单完成	(042) 接口平台上报上游	(0422) 接口平台上报消息	0201–04–2–2–00

五、一些关键类

链路主体类型

> *当前接入的只有这三个类型*

```
public enum TraceSubjectType {
    ORDER, JOB, ACTION
}
```

定义链路步骤接口

```
public interface TraceStep {
    // 步骤层级（一级步骤、二级步骤）
    byte getHierarchy();
    // 步骤标记序号（只要不重复就行）
    int getstepNo();
    // 步骤显示名称（默认展示）
    String getDisplayName();
    // 步骤国际化编码
    String getI18nCode();
    // 步骤描述（RSS >> RMS）
    String getDescription();
}
```

定义RMS链路步骤

```
public class JobTraceStep {
    public static final TraceStep JOB_ACCEPT = StepDefinition.JOB_ACCEPT_2REQUEST_3PROCESS.traceStep;
    public static final TraceStep CHECK_ROBOT = StepDefinition.JOB_ACCEPT_2CHECK_ROBOT_3VALIDATE.traceStep;
    public static final TraceStep JOB_PERSIST = StepDefinition.JOB_ACCEPT_2PERSIST_3PROCESS.traceStep;
    public static final TraceStep CHOOSE_JOB = StepDefinition.JOB_HANDLE_2CHOOSE_3PROCESS.traceStep;
    public static final TraceStep PROCESS_JOB = StepDefinition.JOB_HANDLE_2EXECUTE_3PROCESS.traceStep;

    public static List<TraceStep> getAllTraceSteps() {
        return StreamUtils.mapToList(Arrays.asList(StepDefinition.values()), TraceStepDefinition::getTraceStep);
    }

    private enum StepDefinition implements TraceStepDefinition {
        JOB_ACCEPT(root(
                "0100-1-00000", "任务接收", "trace.rms.job.accept")),

        JOB_ACCEPT_2REQUEST(h2(JOB_ACCEPT,
                "0100-1-10000", "RMS接收到任务请求", "trace.rms.job.accept.request")),
        JOB_ACCEPT_2REQUEST_3PROCESS(h2(JOB_ACCEPT_2REQUEST,
                "0100-1-1-1000", "接收任务请求", "trace.rms.job.accept.request")),

        JOB_ACCEPT_2CHECK_ROBOT(h2(JOB_ACCEPT,
                "0100-1-20000", "机器人状态检查", "trace.rms.job.accept.request")),
        JOB_ACCEPT_2CHECK_ROBOT_3VALIDATE(h2(JOB_ACCEPT_2CHECK_ROBOT,
                "0100-1-2-1000", "当前是否可以作业", "trace.rms.job.accept.request")),

        JOB_ACCEPT_2PERSIST(h2(JOB_ACCEPT,
                "0100-1-30000", "落库", "trace.rms.job.accept.request")),
        JOB_ACCEPT_2PERSIST_3PROCESS(h2(JOB_ACCEPT_2PERSIST,
                "0100-1-3-1000", "RMS内部处理", "trace.rms.job.accept.request")),
```

```
        JOB_HANDLE(root("0100-2-00000", "任务处理", "trace.rms.job.handle")),

        JOB_HANDLE_2CHOOSE(h2(JOB_HANDLE,
                "0100-2-10000", "选取任务", "trace.rms.job.handle")),
        JOB_HANDLE_2CHOOSE_3PROCESS(h2(JOB_HANDLE_2CHOOSE,
                "0100-2-1-1000", "RMS内部处理", "trace.rms.job.handle")),

        JOB_HANDLE_2EXECUTE(h2(JOB_HANDLE,
                "0100-2-20000", "任务执行", "trace.rms.job.handle")),
        JOB_HANDLE_2EXECUTE_3PROCESS(h2(JOB_HANDLE_2EXECUTE,
                "0100-2-2-1000", "RMS内部处理", "trace.rms.job.handle"));

        private final TraceStep traceStep;

        StepDefinition(TraceStep traceStep) {
            this.traceStep = traceStep;
        }

        public TraceStep getTraceStep() {
            return traceStep;
        }
    }
}
```

六、链路日志的上报

1、前置：服务启动上报链路步骤

- 服务启动上报：所有TraceStep配置

2、日志上报流程

埋点日志上报：

- Client端TL记录日志(collector)
- Client端TraceLogStub上报(gRPC)
- Server端TraceLogStub接收(gRPC)
- Basic服务(落库)

3、特殊的基础日志上报逻辑

- 服务接受请求得记录请求的数据（单独的叶子结点的链路步骤）
  - 作业单在接口平台的请求被接受
  - 作业单在RSS的请求被接受
  - 任务在RMS的请求被接受
- 服务间远程调用得记录（单独的链路步骤且下挂两个叶子结点的链路步骤）
  - 作业单在接口平台被转发的请求数据和返回数据
  - 任务在RSS中被下发到RMS的请求数据和返回数据
  - 动作在RMS中被下发到QSH的请求数据和返回数据
  - 动作在RMS请求路径规划的请求数据和返回数据

比如处理路径规划的日志埋点

```
private PlanPathResponseBO requestPathPlan(String msgId, PlanPathRequestBO pathRequest) {
    PlanPathResponseBO pathResponse;
    ATL.pathPlanRequest(msgId, pathRequest);
    try {
        pathResponse = trafficRequestService.requestPathPlanForRms(pathRequest);
        ATL.pathPlanResponseSuccess(msgId, pathRequest);
        return pathResponse;
    } catch (Throwable ex) {
        ATL.pathPlanResponseFailure(msgId, ex.getMessage());
        throw ex;
    }
}
```

七、全链路需要的一些表

1、trace_step

```
CREATE TABLE `basic_trace_step` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `create_time` datetime(3) NOT NULL COMMENT '创建时间',
  `update_time` datetime(3) NOT NULL COMMENT '更新时间',
  `subject_type` varchar(255) NOT NULL COMMENT '链路步骤主体类型',
  `step_no` int(11) NOT NULL COMMENT '链路步骤号',
  `parent_step_no` int(11) NOT NULL COMMENT '链路步骤父步骤号',
  `default_name` varchar(255) DEFAULT NULL COMMENT '链路默认名称',
  `i18n_code` varchar(255) DEFAULT NULL COMMENT '链路名称国际化编码',
  `hierarchy` tinyint(4) DEFAULT NULL COMMENT '链路层级',
  `description` varchar(255) DEFAULT NULL COMMENT '链路描述',
  PRIMARY KEY (`id`),
  UNIQUE KEY `idx_step_no` (`step_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='链路步骤定义';
```

2、trace_subject

```
当前链路主体（subjectNo）是什么，链路主体的父主体又是什么（parentNo），链路主体的类型又是什么（subjectType）

一、构建是链路主体的层级关系，这个层级关系只有依赖的层级关系，而不会决定页面上面如果展示（页面展示由stepNo的层级决定）
二、构建完的关系数据对应的搜索框，任意的链路主体数据（上游作业单号、系统作业单号、任务号、动作标识）传进来都能构建出所有的日志数据
```

> *数据量不大、查询索引使用*

```
CREATE TABLE `basic_trace_subject` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `create_time` datetime(3) NOT NULL COMMENT '创建时间',
  `subject_type` varchar(255) NOT NULL COMMENT '主体类型',
  `subject_no` varchar(255) NOT NULL COMMENT '链路主体号',
  `parent_no` varchar(255) NOT NULL COMMENT '链路主体父主体号',
  PRIMARY KEY (`id`),
```

```
  KEY `idx_subject_no` (`subject_no`),
  KEY `idx_parent_no` (`parent_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='链路主体记录';
```

比如：

id	subjectType	subjectNo	parentNo
1	ORDER	Console_rack_move1662551745786	null
2	ORDER	SIRack_166132508128900027	Console_rack_move1662551745786
3	JOB	BM_SI_166132480842800026	SIRack_166132508128900027
4	ACTION	6c6f485c235811ed89070242ac1b000e	BM_SI_166132480842800026

3、trace_log

> *数据量大，单条数据量小（timeValue可存起始时间+时间段）*

```
CREATE TABLE `basic_trace_log` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `create_time` datetime(6) NOT NULL COMMENT '创建时间',
  `subject_id` bigint(20) NOT NULL COMMENT '链路主体ID',
  `step_no` int(11) NOT NULL COMMENT '链路步骤号',
  `is_success` bit(1) DEFAULT NULL COMMENT '链路步骤是否执行成功',
  `time_type` tinyint(4) DEFAULT NULL COMMENT '链路步骤日志时间类型',
  `time_value` bigint(20) DEFAULT NULL COMMENT '链路步骤日志时间值',
  `content_type` tinyint(4) DEFAULT NULL COMMENT '链路步骤日志内容类型',
  `content_id` bigint(20) DEFAULT NULL COMMENT '链路步骤日志内容值',
  PRIMARY KEY (`id`),
  KEY `idx_create_time` (`create_time`),
  KEY `idx_subject_id` (`subject_id`),
  KEY `idx_step_no` (`step_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='链路步骤日志记录';
```

4、trace_log_content

> *数据量大，单条数据存储大，查询靠主键*

```
CREATE TABLE `basic_trace_log_content` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `create_time` datetime(6) NOT NULL COMMENT '创建时间',
  `content` text COMMENT '链路步骤日志内容',
  PRIMARY KEY (`id`),
  KEY `idx_create_time` (`create_time`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='链路步骤日志内容';
```

八、查询逻辑

1、查询所有的trace_subject

```
select id from trace_subject where subject_no = :subject_no; // 判断是否存在
select id as subject_id, subject_no from trace_subject where parent_no = '';
```

2、查询每个subject对应的trace_log

```
select * from trace_log where subject_id = :subject_id;
```

3、查询每个trace_log对应的长（短）内容

```
select id, content from trace_content where id in (...);
```

九、API格式

1、API定义

根据链路主体号查询步骤日志（全步骤的日志树）

```
GET /api/basic/warehouse/{warehouseId}/trace/subject/{subjectNo}/step-log-tree
```

2、返回数据格式

```
{
    // 由于多主体在返回的时候没有嵌套，因此多主体的关系用列表来描述
    relations: [{
        subjectType: "ORDER",
        subjectNo: "SIRack_166356690921200003",
        children: ["BM_SI_166356691023500004","BM_SI_166356694576300005"]
    }],
    // 多主体的链路日志
    subjects:[{
        subjectType: "ORDER",
        subjectItems: [{
            subjectId: 2341,
            subjectNo: 'SIRack_166356690921200003',
            subjectLogs: [{
                // 一级链路步骤
                stepNo: 201010000,
                stepName: "接收作业单",
                // 一级链路步骤的总耗时
                timeRange: {
                  startTime: "2022-09-19 13:55:09.213",
                  endTime: "2022-09-19 13:55:09.348",
                  duration: 135
                },
```

```
                children: [{
                    stepNo: 201011000,
                    stepName: "接口平台接收下发请求",
                    displayMode: "COMMON",
                    children: [{
                        stepNo: 201011100,
                        stepName: "Interface接收",
                        // 链路叶子结点的日志（链路步骤需要显示的日志数据）
                        stepLog: {
                          contentType: "RECEIVE_JSON",
                          contentId: 14340,
                          success: true,
                          timeType: "RECEIVE_TIME",
                          timeValue: "2022-09-19 13:55:09.213"
                        }
                    }],
                    timeRange: {
                      startTime: "2022-09-19 13:55:09.213",
                      endTime: "2022-09-19 13:55:09.348",
                      duration: 135
                    }
                }]
            }],
            // 单个链路主体的总耗时
            timeRange: {
              startTime: "2022-09-19 13:55:09.213",
              endTime: "2022-09-19 13:55:46.070",
              duration: 36857
            }
        }]
    }, {
        subjectType: "JOB",
        subjectItems: [{},{}]
    } {
        subjectType: "ACTION",
        subjectItems: [{},{}]
    }],
    // 全链路耗时
    timeRange: {
        startTime: "2022-09-19 13:55:09.213",
        endTime: "2022-09-19 13:55:49.335",
        duration: 40122
    }
}
```

十、一些问题

各subject的关联关系

- 作业单和任务的关联关系（通过上报解决）
- 多任务多动作的数据返回及关联关系
- 怎么通过上游作业单号查询到作业单链路
- 界面作业单和任务的前后关系

显示问题

- 非叶子节点的耗时（增加timeRange字段）
- 叶子节点时间和内容类型的文本显示（前端自己处理国际化展示）
- 分车链路日志分页（后端全部返回，前端发现children的条目大于10个就分页）
- 服务间调用日志合并（增加ContentDisplayMode类型）

数据更新

- 废弃的链路步骤怎么办