

# Apache Doris 在趣丸科技 实时数仓的应用实践

高瑞林

趣丸科技 数据部数据仓库架构师

## 自我介绍

### 高瑞林

现任趣丸科技数据仓库架构师

#### 个人简介：

曾就职于百度，基于 Doris 拥有广告投放，游戏社交，在线教育等领域的数据仓库架构经验，目前负责趣丸科技实时数仓的架构





# 目录

1. 趣丸科技实时数仓发展历程
2. 实时数仓架构设计
3. 实时数仓场景化实战
4. 总结收益与展望

# 1 趣丸科技实时数仓发展历程



1.1 公司业务介绍



电子竞技



TT电竞

王者荣耀分部、英雄联盟分部  
英雄联盟手游分部、和平精英分部

趣丸科技核心业务



兴趣社交



TT语音



麦可



TTchat



唱鸭



技术研发



人工智能  
联合实验室



码上跃见  
技术品牌



## 1.2 Doris 使用场景



2022年上半年引入 Doris



DORIS

主集群规模

- 数十个节点

- 近百个实时任务导入数据
- 单表数据量最大增量超百亿/天
- 数百T存储

运营活动

- 运营标签
- 营销活动效果分析

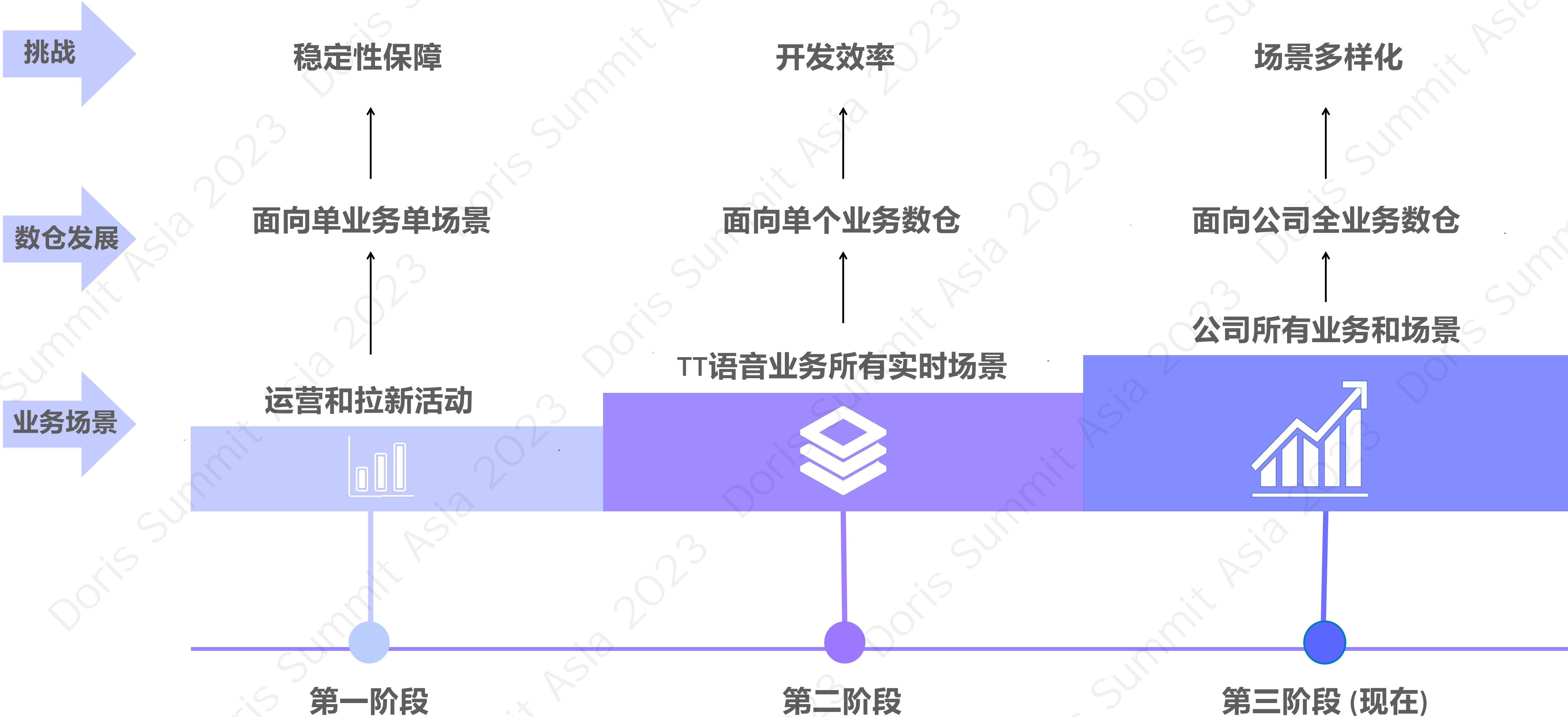
业务报表

- 数据驾驶舱
- 多维度分析报表
- 分钟趋势报表

算法

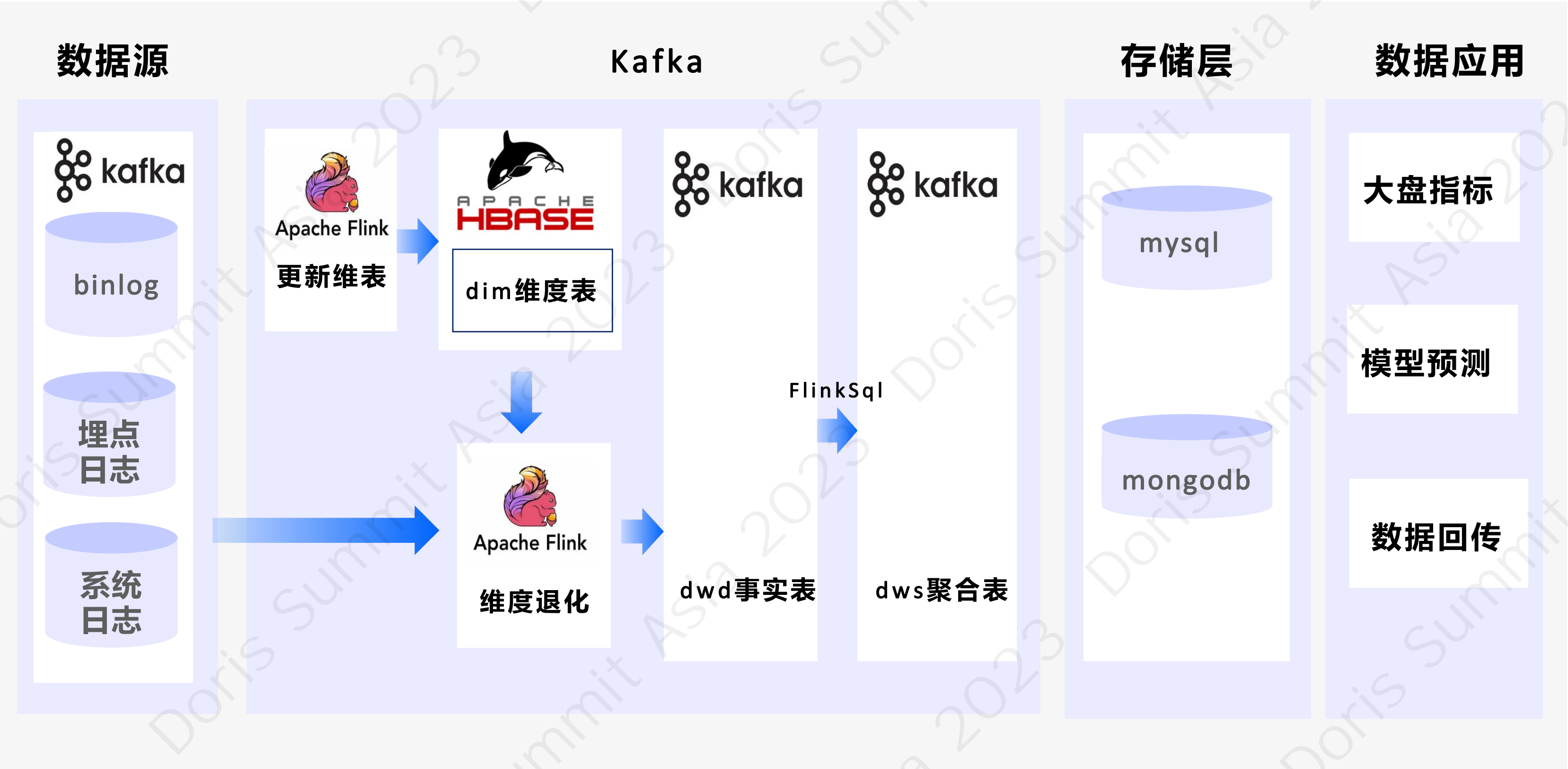
- 数据预测
- 模型训练

# 1.3 趣丸科技实时数仓的发展历程





# 1.4 实时数仓架构的演变 (V1.0)



## 架构设计

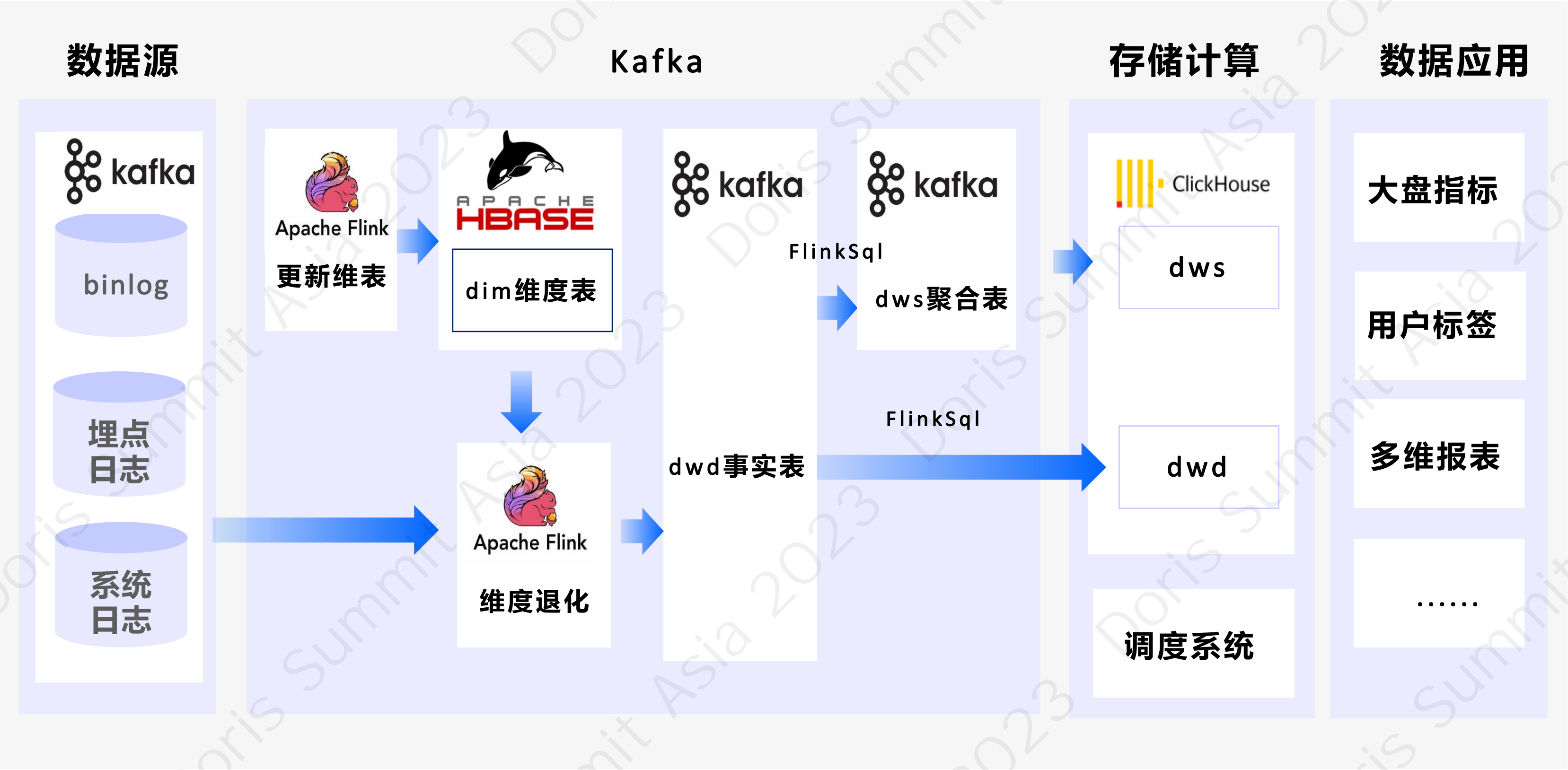
- 基于Flink+Hbase的架构

## 遇到的主要问题

- 问题排查难度高：体现在基于Flink将指标提前聚合好，结果存储在Mysql/mongodb等，问题排查需要从Kafka里面拉取明细数据；
- 数据质量问题，多流join都在hbase，经常出现数据延迟导致结果计算错误；



# 1.5 实时数仓架构的演变 (v2.0)



## 架构设计

- 基于Flink+Clickhouse
- 高时效性场景通过Flink计算
- 准实时的将明细数据写到Clickhouse, 通过分钟级调度计算;

## 遇到的主要问题

- Clickhouse的本身问题:
  - 对标准SQL支持有限, 语法不易理解;
  - 多维报表的查询效率慢, 高峰情况下容易超时;
  - 支持直接订阅Kafka, 但是无法保障数据不丢失, 不重复
  - 实时更新删除的能力较弱, 集群运维成本高;

## 1.6 应用场景面临的问题

应用场景	时效性要求	业务的关注点	数据特点	面临的问题
运营标签	根据计算复杂度 复杂标签：准实时 简单标签：实时	关注用户在特定业务场景下的标签数据，用于运营策略的调整	涉及实体：用户和设备 统计指标：人数，人次，金额，时间 统计周期：近n分钟，近n小时，近n天，历史累计	基于现有的架构开发成本高
实时报表	根据指标级别 核心指标：实时 其他指标：准实时	关注广告投放后各渠道的获客指标，消费指标，及端内关键行为指标	维度灵活：通过各种维度组合看数据 准确性：数据准确性要求高	多维度分析场景下，使用Clickhouse通常需要join维度表，查询较慢，使用高峰期容易超时
算法预测	核心模型：实时 其他模型：准实时	通过模型预测未来数据的变化趋势	表多且数据量大：通常使用的特征数据每天都在数亿级	任务性能瓶颈问题，时效性无法满足业务



# 1.7 Doris VS Clickhouse 性能测试

**基础条件：** 集群1FE3BE， duplicate表， 数据量7亿左右， 集群规模华为云服务器32core256G SSD 1T

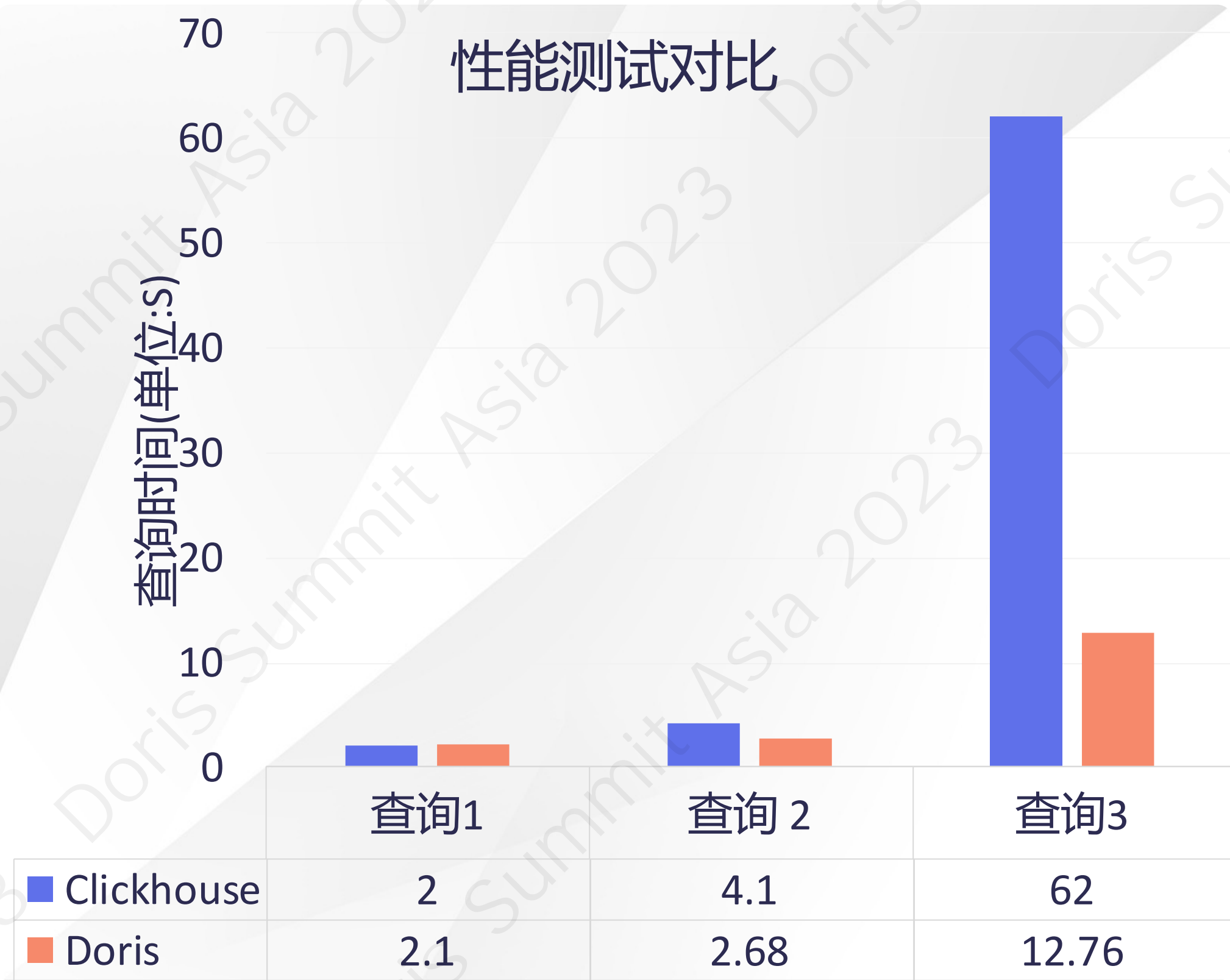
**查询3的示例：**

```
MySQL [tmp]> select
->   a.app_name, a.user_id, b.province
-> from
->   (
->     -- 7亿左右数据
->     select
->       app_name
->       ,user_id
->     from
->       test_distribute_by_uid2
->     where server_timestamp between '2022-11-07 00:00:00' and '2022-11-08 00:00:00'
->   ) a
-> left join
->   (
->     -- 7亿左右数据
->     select
->       app_name
->       ,user_id
->       ,province
->     from
->       test_distribute_by_uid2
->     where server_timestamp between '2022-11-07 00:00:00' and '2022-11-08 00:00:00'
->   ) b on b.user_id = a.user_id and b.app_name = a.app_name
-> limit 10;
```

app_name	user_id	province
		江苏省
		江苏省
		江苏省
		江苏省
		江苏省
		江苏省
		江苏省
		江苏省
		江苏省
10 rows in set (12.76 sec)		

**测试场景：**

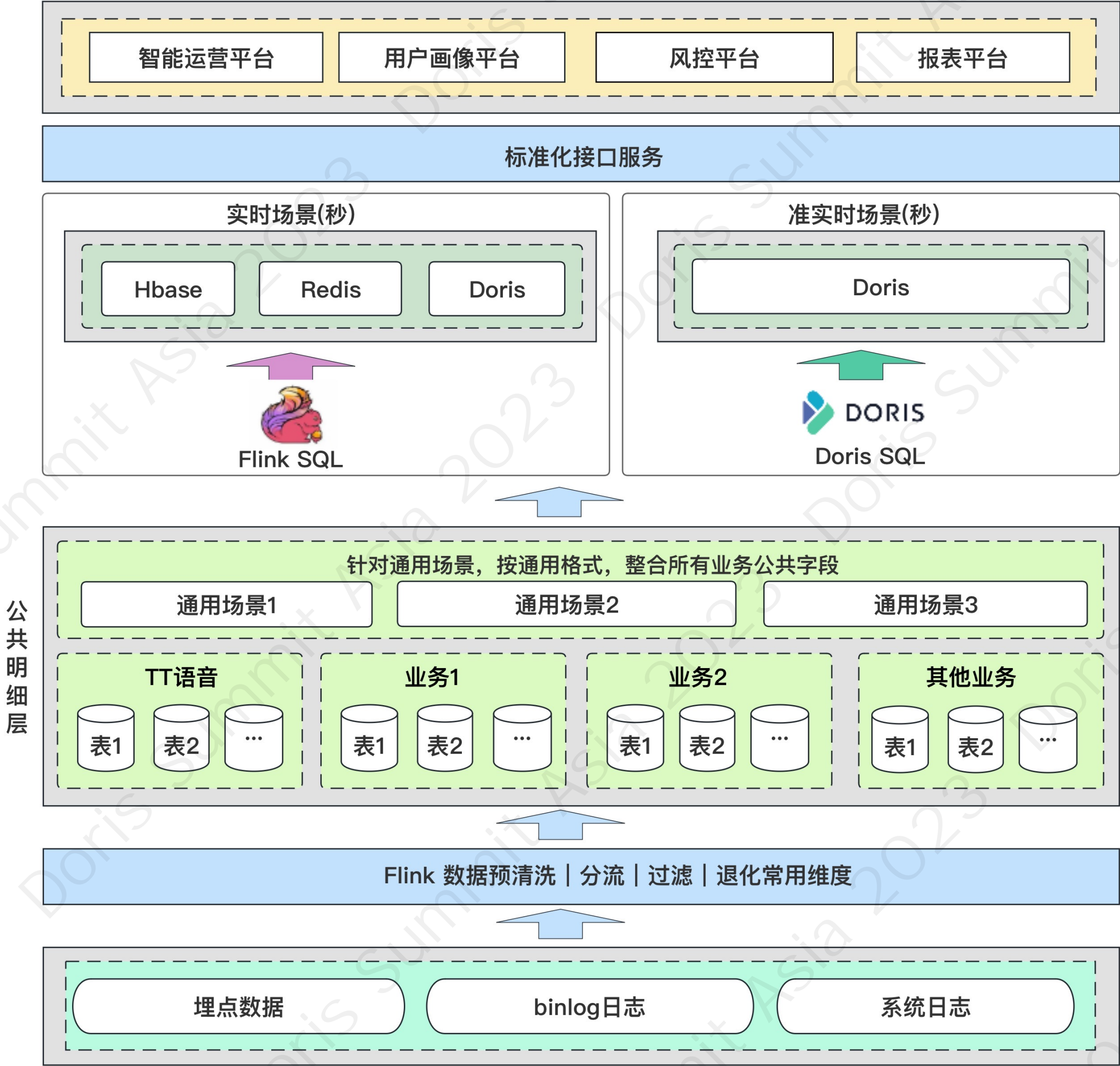
- 查询1： 单表(7亿)分组聚合
- 查询2： 大表(7亿) Join 小表(400w)
- 查询3： 大表(7亿) Join 大表(7亿) 笛卡尔积



## 2 实时数仓的架构设计



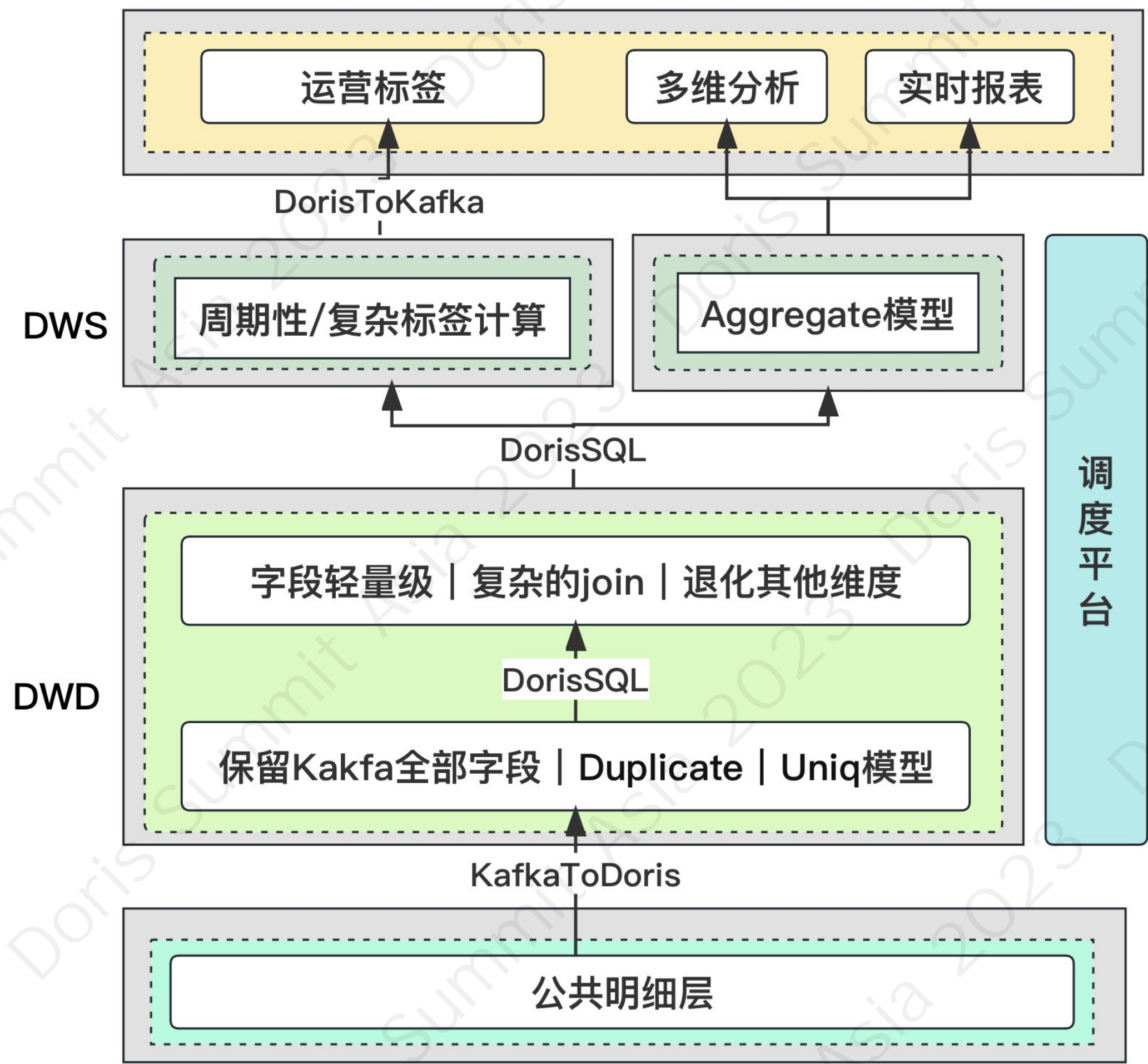
## 2.1 数仓架构设计方案



### 架构设计

- 公共明细层:
  - 通过 FlinkSQL 对各业务的原始数据做清洗, 形成各业务统一标准的 dwd 层数据, 供下游使用;
  - 针对通用场景, 整个所有业务通用字段, 便于下游使用
- 针对时效性高场景:
  - 基于清洗后的 DWD, 通过 FlinkSQL 计算后, 写入对应的存储介质, 供下游使用;
- 针对准实时场景:
  - 基于清洗后的 DWD, 通过 DorisSQL, 分钟级计算, 写入 Doris 后, 供下游使用

## 2.2 Doris 在实时场景的应用方案



### 架构设计

#### DWD

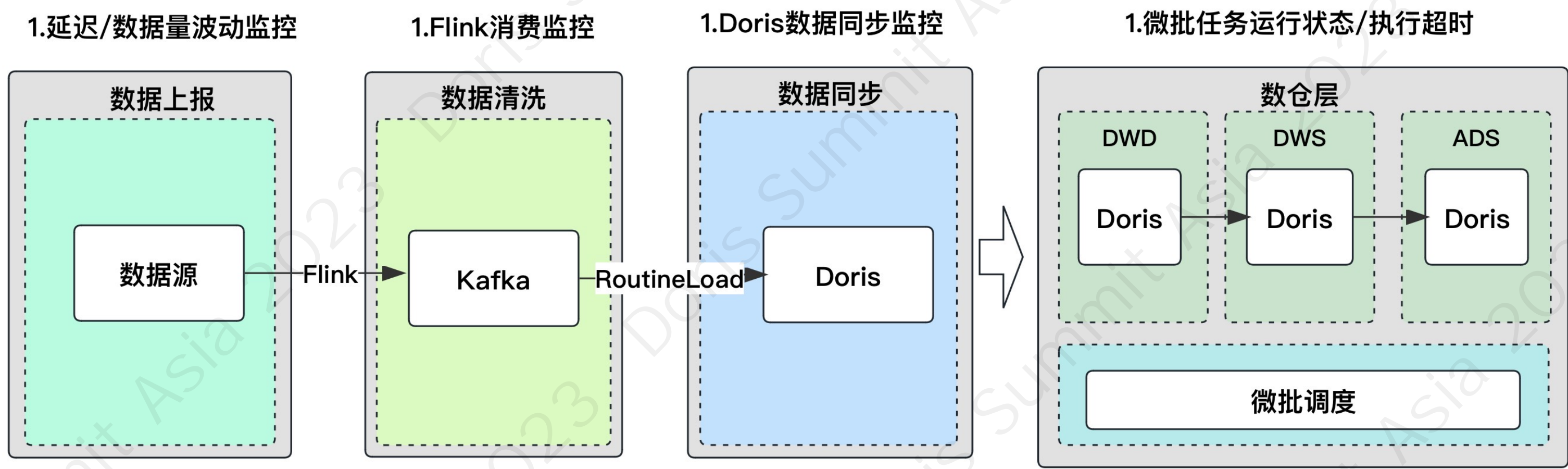
- 基于清洗后的明细数据，通过FlinkSQL写入Doris，作为Doris的数据源，通过Doris的明细模型，尽可能保留全部字段，方便新增字段时，历史数据的回溯；
- 基于Doris的第一层明细数据，针对复杂的join场景做维度退化，最终字段设计尽可能轻量级；

#### DWS

- 复杂标签的计算，可在Doris计算后，同步至Kafka供下游使用；
- 基于Doris的聚合模型，对指标进行预聚合，供上层报表或分析使用；



## 2.3 数据监控方案



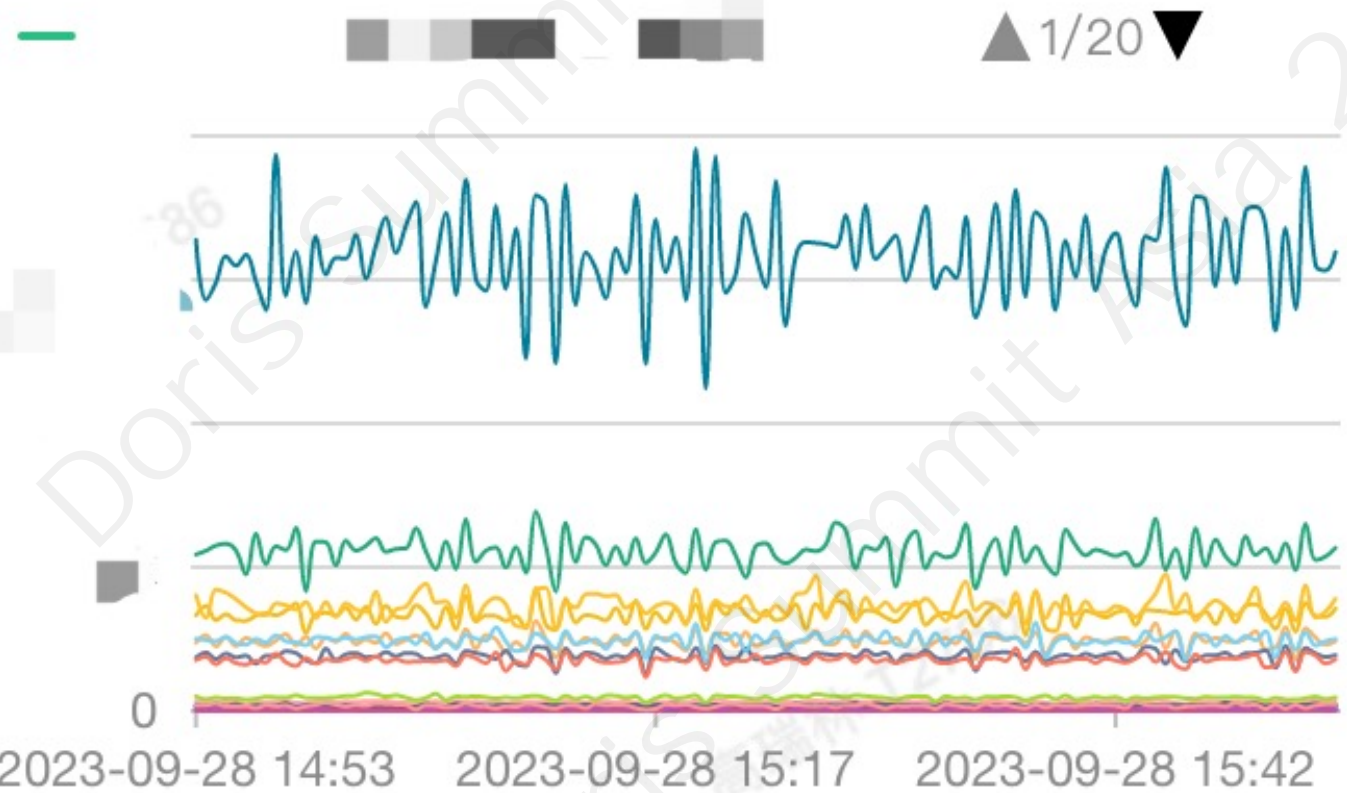
### 监控方式

• 数据链路

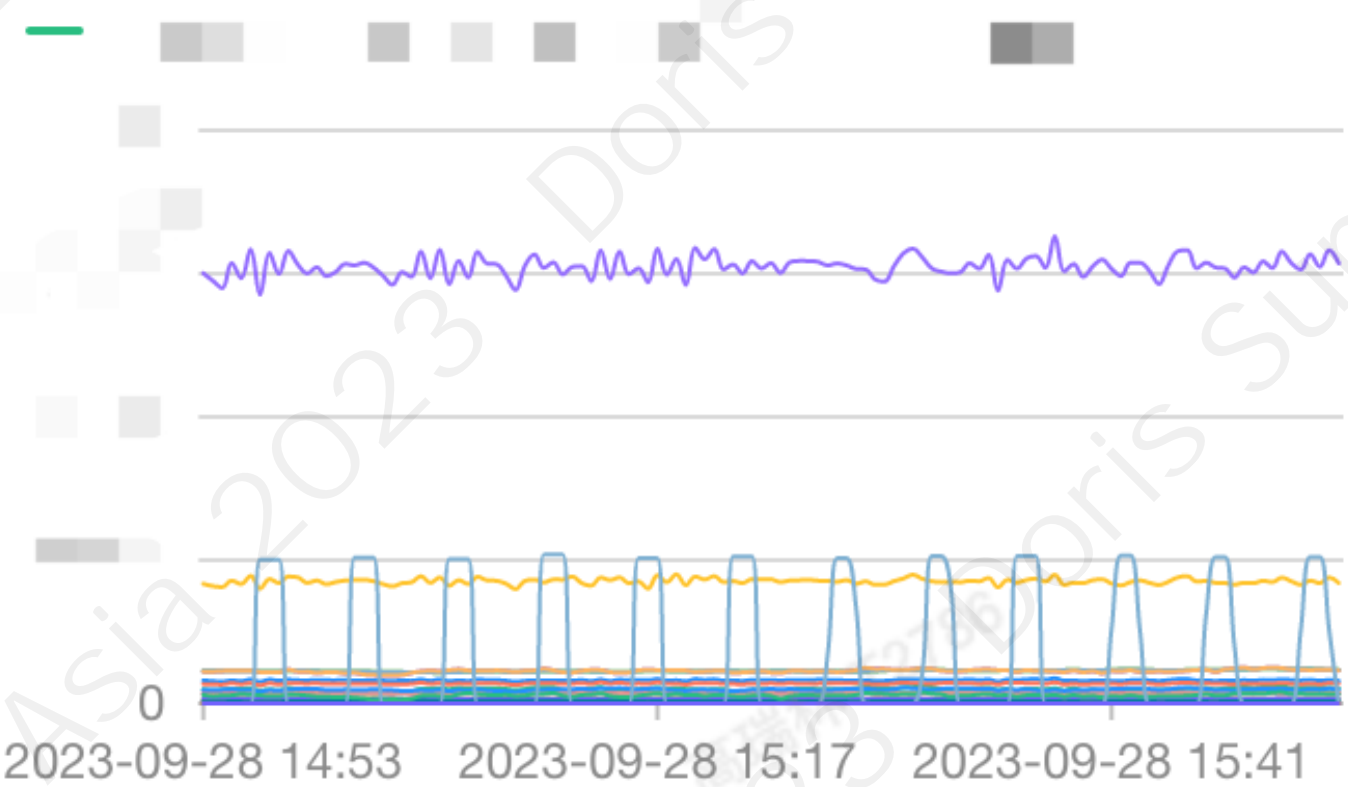
- 数据上报
- 数据清洗
- 数据同步
- 数仓层

由于涉及链路长，暂时没法对端到端做完整监控，只针对全链路的各个环节做保障

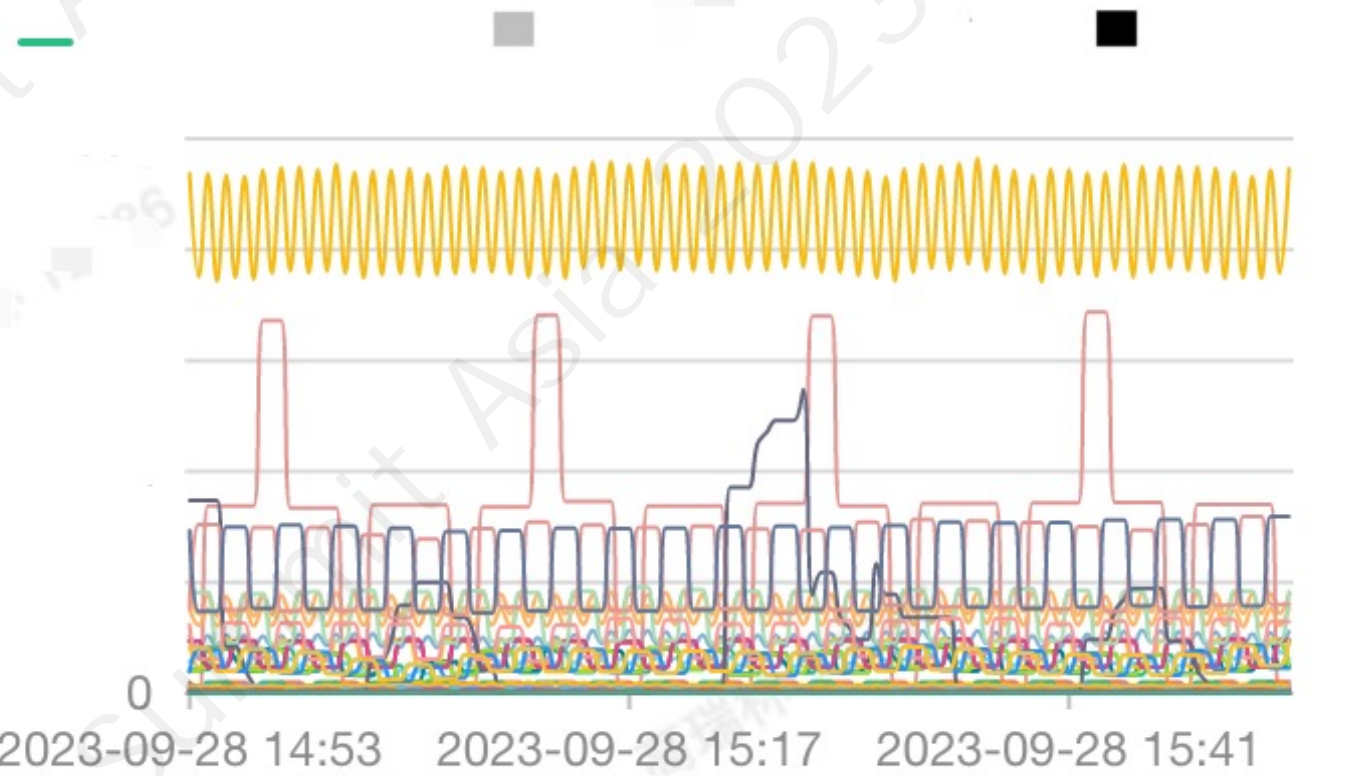
数据上报：埋点(单位：条/秒)



生产阶段：流入速率(单位：条/秒)



消费阶段：消费速率(单位：条/秒)





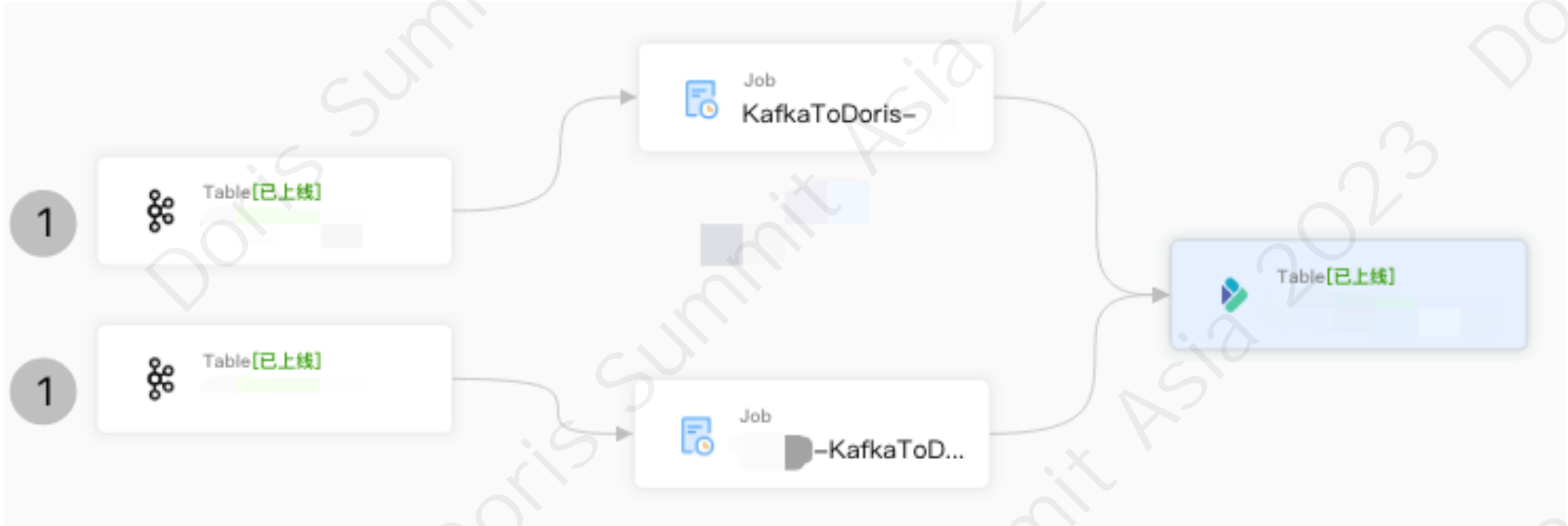
## 2.4 内部平台对 Doris 支撑

### Doris元数据管理

- 通过提前注册Doris和Kafka的元数据，任务的开发只需要关注逻辑的实现



### Doris任务血缘



### 基于社区 Doris 的 Flink connector 的改良

- 新增条件下推，doris开源官方的在实现flink-connector-doris的说明不实现Filter下推和Projection下推

#### 改造前：

```
alter table `doris.xx.test_aggregate_table`
set (
  'sink.label-prefix' = 'xxx',
  'sink.properties.format' = 'json',
  -- 读取 Doris 表的列名列表，多列之间使用逗号分隔
  'doris.read.field' = 'row1,row2,row3',
  -- 过滤读取数据的表达式
  'doris.filter.query' = 'update_time>=if('${lastSucceedScheduledTime}' = '',
    '2023-10-21 00:00:30','${lastSucceedScheduledTime}')'
);

INSERT INTO `kafka.`
select
  row1,
  row2,
  row3,
  ....
from
  `doris.xx.test_aggregate_table`;
```

#### 改造后：数据的Filter不需要通过参数配置

```
alter table `doris.xx.test_aggregate_table`
set (
  'sink.label-prefix' = 'xxx',
  'sink.properties.format' = 'json'
);

INSERT INTO `kafka.`
select
  row1,
  row2,
  row3,
  ....
from
  `doris.xx.test_aggregate_table`
where
  update_time>=if('${lastSucceedScheduledTime}' = '', '2023-10-21 00:00:30', '${lastSucceedScheduledTime}');
```



# 3 实时数仓场景化实战

# 场景一：实时报表维度变更问题

## 业务场景

- 广告投放场景下：分省份，城市，性别维度，统计用户数

```
CREATE TABLE IF NOT EXISTS tmp.test_aggregate_table
(
  -- 维度列
  data_date DATE NOT NULL COMMENT '日期',
  age VARCHAR(20) NOT NULL COMMENT '年龄',
  province VARCHAR(20) NOT NULL COMMENT '省份',
  city VARCHAR(20) NOT NULL COMMENT '城市',
  -- 指标列
  user_cnt BIGINT REPLACE NOT NULL default '0' COMMENT '用户数'
) -- 采用doris的AGGREGATE聚合模型
AGGREGATE KEY(data_date,channel_id,province,city)
PARTITION BY RANGE(`data_date`) ()
DISTRIBUTED BY HASH(`data_date`) BUCKETS 3
PROPERTIES(
  "dynamic_partition.enable" = "true",
  "dynamic_partition.time_unit" = "DAY");
```

## 遇到的问题

- 在doris中指定了维度列和指标列，AGGREGATE模型设定了key之后，维度发生变更会导致汇总后结果重复。

用户id	年龄	省份	城市	更新时间
A	未知	广东省	广州市	10:30:20
B	18	湖北省	武汉市	10:30:20

聚合后

年龄	省份	城市	用户数
未知	广东省	广州市	1
18	湖北省	武汉市	1

10分钟后用户填写了年龄

由未知变更为25

用户id	年龄	省份	城市	更新时间
A	25	广东省	广州市	10:40:18

聚合后

年龄	省份	城市	用户数
未知	广东省	广州市	1
25	广东省	广州市	1
18	湖北省	武汉市	1

按省份汇总查看

省份	城市	用户数
广东省	广州市	2
湖北省	武汉市	1



# 场景一：实时报表维度变更问题

## 解决方式

- 将维度列age字段类型改成REPLACE
- 改成REPLACE后，当维度发生变更时，之前当数据也会被替换掉，从而保障数据准确性
- 缺点：无法根据聚合字段创建索引，会降低一定的查询性能。

```
CREATE TABLE IF NOT EXISTS tmp.test_aggregate_table_v2
(
  -- 维度列
  data_date DATE NOT NULL COMMENT '日期',
  province VARCHAR(20) NOT NULL COMMENT '省份',
  city VARCHAR(20) NOT NULL COMMENT '城市',
  -- 指标
  age VARCHAR(20) REPLACE NOT NULL COMMENT '性别',
  device_cnt BIGINT REPLACE NOT NULL default '0' COMMENT '设备数'
)
AGGREGATE KEY(data_date,province,city)
PARTITION BY RANGE(`data_date`) ()
DISTRIBUTED BY HASH(`data_date`) BUCKETS 3
PROPERTIES(
  "dynamic_partition.enable" = "true",
  "dynamic_partition.time_unit" = "DAY");
```

用户id	年龄	省份	城市	更新时间
A	未知	广东省	广州市	10:30:20
B	18	湖北省	武汉市	10:30:20

聚合后

年龄	省份	城市	用户数
未知	广东省	广州市	1
18	湖北省	武汉市	1

10分钟后用户填写了年龄

由未知变更为25

用户id	年龄	省份	城市	更新时间
A	25	广东省	广州市	10:40:18

聚合后

年龄	省份	城市	用户数
25	广东省	广州市	1
18	湖北省	武汉市	1

按省份汇总查看

省份	城市	用户数
广东省	广州市	1
湖北省	武汉市	1

场景二：基于 Doris 的指标和标签的计算

业务场景

没有 Doris 之前，基于 FlinkSql 的计算方式

基于 Doris 的解决方案

指标的计算场景

- 分钟级指标；
- 当天分小时指标；
- 当天累计指标；

```
-- 基于FlinkSql的滚动窗口
-- 计算1分钟内的注册用户
SELECT
  date_format(window_start, 'yyyy-MM-dd') as data_date,
  window_start,
  window_end,
  -- 每分钟注册用户
  COUNT(distinct user_id) AS reg_uv
FROM
  -- 窗口大小1分钟，步长1分钟
  TABLE(
    TUMBLE(
      TABLE kafka_user_reg_data,
      DESCRIPTOR(ts),
      INTERVAL '60' SECOND
    )
  )
GROUP BY
  window_start,
  window_end
```


```
-- 基于FlinkSql的滑动窗口
-- 每隔1分钟计算1小时内注册用户
SELECT
  date_format(window_start, 'yyyy-MM-dd') as data_date,
  window_start,
  window_end,
  -- 每隔1分钟计算1小时内注册用户数
  COUNT(distinct user_id) AS reg_uv
FROM
  -- 窗口大小1小时，步长1分钟
  TABLE(
    HOP(
      TABLE kafka_user_reg_data,
      DESCRIPTOR(ts),
      INTERVAL '60' SECOND,
      INTERVAL '1' HOUR
    )
  )
GROUP BY
  window_start,
  window_end
```

```
-- 基于FlinkSql的渐进式窗口
-- 每隔1分钟计算今天0点累计至今的注册用户数
SELECT
  date_format(window_start, 'yyyy-MM-dd') as data_date,
  window_start,
  window_end,
  -- 每隔1分钟计算今天内注册用户数
  COUNT(distinct user_id) AS reg_uv
FROM
  -- 最小窗口大小1分钟，最大窗口大小1天，从每天0点开始，24:00:00结束，第二天新开一个窗口。步长1分钟
  TABLE(
    CUMULATE(
      TABLE kafka_user_reg_data,
      DESCRIPTOR(ts),
      INTERVAL '60' SECOND,
      INTERVAL '1' DAY
    )
  )
GROUP BY
  window_start,
  window_end
```

```
CREATE TABLE IF NOT EXISTS tmp.test_bitmap_table
(
  -- 维度列
  data_date      DATE      NOT NULL COMMENT '日期',
  hour           VARCHAR(20) NOT NULL COMMENT '小时',
  minute         VARCHAR(20) NOT NULL COMMENT '分钟',
  user_cnt       bitmap    bitmap_union COMMENT '用户数'
) -- 采用doris的AGGREGATE聚合模型
AGGREGATE KEY(data_date, hour, minute)
PARTITION BY RANGE(`data_date`) ()
DISTRIBUTED BY HASH(`data_date`) BUCKETS 3
PROPERTIES(
  "dynamic_partition.enable" = "true",
  "dynamic_partition.time_unit" = "DAY"
);
```

- 采用AGGREGATE模型，通过bitmap计算uv

```
-- 数据写入Doris
insert into tmp.test_bitmap_table
select
  data_date,
  hour(event_time) as hour,
  minute(event_time) as minute,
  to_bitmap(user_id) as user_id
from
  user_reg_data
where
  data_date='2023-09-18'
```



```
-- 数据查询
select
  data_date, -- 天
  hour, -- 小时
  minute, -- 分钟
  sum(bitmap_count(user_cnt)) -- 用户数
from
  tmp.test_bitmap_table
group by
  minute,
  hour,
  data_date
```



# 场景二：基于 Doris 的指标和标签的计算

## 业务场景

- 标签的计算场景
  - 统计型规则(注册的用户当天累计消费次数大于y次);
  - 序列型规则(A用户注册后x秒内发生了消费)
  - 混合型规则(A用户注册后x秒内发生了充值金额且累计大于y元)

## 基于 Doris 的解决方案

用户id	次数	消费金额	消费时间
A	1	10.0	10:30:20
B	1	20.0	10:31:18

聚合后

用户id	次数	金额	首次消费时间	最近消费时间
A	1	10.0	10:30:20	10:30:20
B	1	20.0	10:31:18	10:31:18

用户id	次数	消费金额	消费时间
A	1	15.0	10:35:25
B	1	30.0	10:36:20

聚合后

用户id	次数	金额	首次消费时间	最近消费时间
A	2	25.0	10:30:20	10:35:25
B	2	50.0	10:31:18	10:36:20

```
CREATE TABLE IF NOT EXISTS tmp_user_consume
(
  data_date      DATE      NOT NULL    comment '日期',
  user_id        VARCHAR(20) NOT NULL   comment '用户id',
  amount_pv      bigint    sum  comment '消费次数',
  amount         bigint    sum  comment '消费金额',
  first_consume_time datetime min      comment '当天首次消费时间',
  last_consume_time datetime max      comment '最近一次消费时间'
)
-- 用户消费表
-- 使用doris的AGGREGATE模型,对amount做sum,采用min和max分别聚合首次和末次时间
AGGREGATE KEY(data_date,user_id)
PARTITION BY RANGE(`data_date`)( )
DISTRIBUTED BY HASH(`data_date`) BUCKETS 3
PROPERTIES(
  "dynamic_partition.enable" = "true",
  "dynamic_partition.time_unit" = "DAY",
)
```

```
-- 注册的用户当天累计消费次数大于y次
select
  a.user_id
from
(
  select
    user_id,
    reg_time
  from
    tmp_user_reg -- 注册表
)a
join
(
  select
    user_id,          -- 用户id
    amount_pv         -- 用户当天累计消费次数
  from
    tmp_user_consume -- 消费表
)b
on(a.user_id=b.user_id)
where
  amount_pv>y -- 累计消费次数大于y次;
```

```
-- A用户注册后x秒内发生了消费
select
  a.user_id
from
(
  select
    user_id,
    reg_time
  from
    tmp_user_reg -- 注册表
)a
join
(
  select
    user_id,          -- 用户id
    amount,           -- 消费金额
    first_consume_time -- 首次消费时间
  from
    tmp_user_consume -- 消费表
)b
on(a.user_id=b.user_id)
where
  timediff(reg_time,first_consume_time)>x -- 注册后x秒
  and amount>0 -- 表示发生过消费行为
```

```
-- A用户注册后x秒内发生了充值金额且累计大于y元
select
  a.user_id
from
(
  select
    user_id,
    reg_time
  from
    tmp_user_reg -- 注册表
)a
join
(
  select
    user_id,          -- 用户id
    amount,           -- 累计消费金额
    first_consume_time -- 首次消费时间
  from
    tmp_user_consume -- 消费表
)b
on(a.user_id=b.user_id)
where
  timediff(reg_time,first_consume_time)>x -- 注册后x秒
  and amount>y -- 表示发生过消费行为
```

## 4 总结收益与展望



## 4.1 总结收益

业务场景	场景特点与诉求	面临的问题	Doris的优势
运营标签	涉及实体：用户和设备 统计指标：人数，人次，金额，时间 统计周期：近n分钟，近n小时，近n天，历史累计	开发成本高	<ul style="list-style-type: none"><li>Aggregate模型通过预聚合的方式，提前聚合好部分标签结果，让标签开发简单高效，相比之前平均开发效率提升30%，部分标签提升60%以上</li></ul>
实时报表	维度灵活：通过各种维度组合看数据 准确性：数据准确性要求高	多维度分析场景下，使用Clickhouse通常需要join维度表，查询较慢，容易超时	<ul style="list-style-type: none"><li>相比Clickhouse，Doris不仅支持大宽表模型，还支持星型和雪花模型，多维度分析场景下，支持分布式Join，报表95分位查询性能相比之前平均提升35%以上；</li><li>数据写入有事务，保证了数据写入的“不丢不重”，因之前CK导入数据重复问题造成的数据质量问题相比之前下降40%；</li></ul>
算法推荐	数据量大：通常使用的数据每天都在数亿级	任务性能瓶颈问题，时效性无法满足业务	<ul style="list-style-type: none"><li>MPP的架构，能够支持算法场景下，复杂的Sql查询，相同情况下，对比Clickhouse查询性能平均提升25%。</li></ul>

## 4.2 展望未来

- 深入 Doris 在 OLAP 分析场景下的应用及 Doris 的调优；
- 引入数据湖相关技术，探索基于 Doris 在湖仓一体方面的应用；





获取更多社区动态与最佳实践

### Apache Doris 官方平台:

- Apache Doris 官网: [doris.apache.org](https://doris.apache.org)
- Apache Doris GitHub: [github.com/apache/doris/](https://github.com/apache/doris/)

### 获取更多峰会资料:

- Doris Summit 峰会官网: [doris-summit.org.cn](https://doris-summit.org.cn)
- Doris Summit 峰会回放: <https://space.bilibili.com/1196172099/channel/collectiondetail?sid=1824324>