知乎 有问题 就会有答案 | DataFun.

基于 DORIS 的知乎 DMP 系统的架构与实践

侯容 知乎用户理解&数据赋能研发 Leader

目录 CONTENT

01 背景
DMP 业务
DMP 业务流程
DMP 画像特征
DMP 功能梳理

02 架构与实现
DMP 架构
DMP 平台功能盘点 – 业务向
DMP 平台功能盘点 – 基础向
特征数据链路及存储
人群定向流程

03 难点及解决方案
人群定向性能优化 - 第一阶段
人群定向性能优化 - 第二阶段

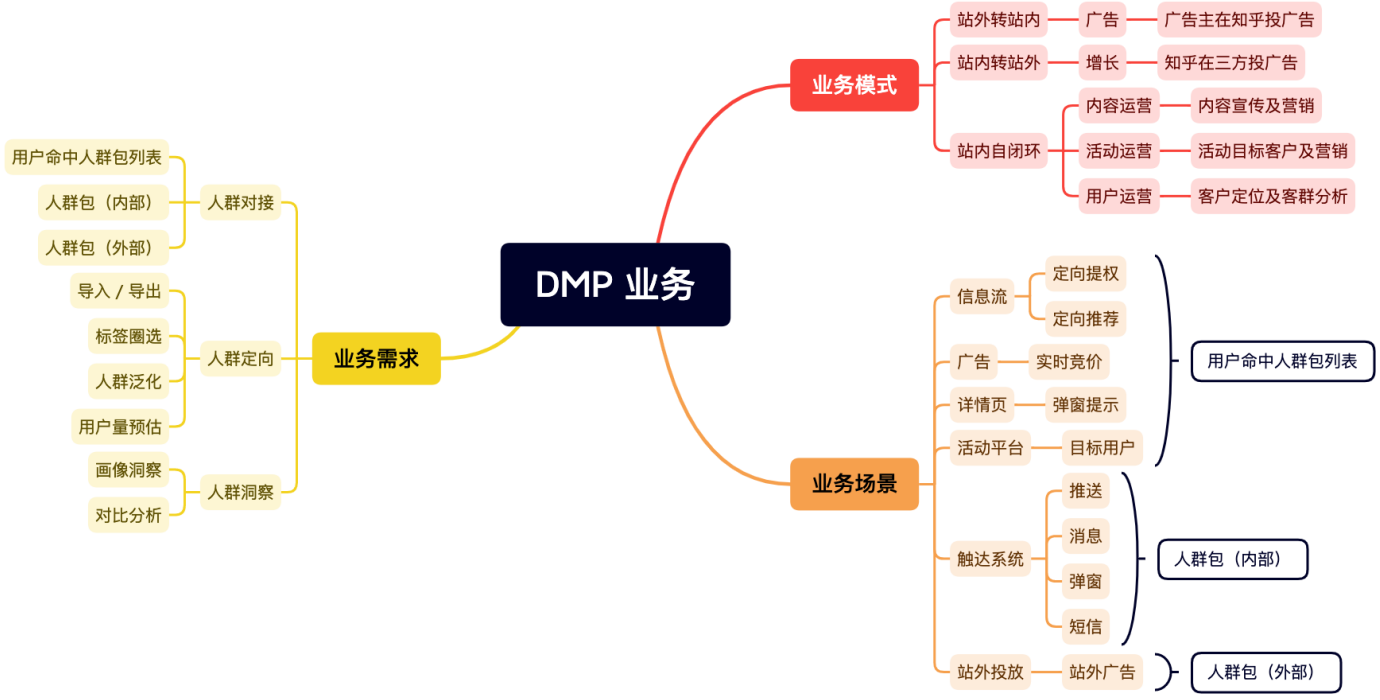
04 未来展望
业务向
技术向

知乎 有问题 就会有答案 | DataFun.

01 背景

DMP 业务

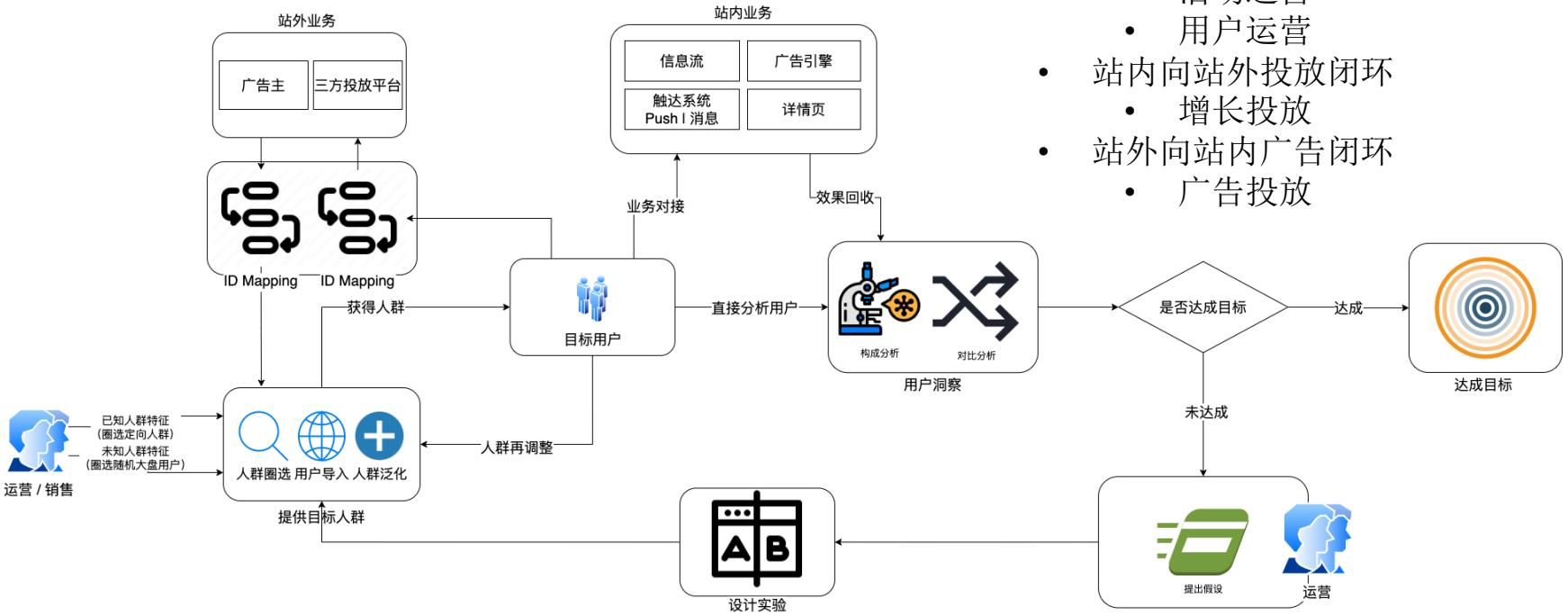
知乎业务中存在哪些问题需要解决？为什么要建立 DMP 平台来解决这些问题？



DMP 业务流程

当前这些业务的运营流程是怎样的？DMP 如何与业务结合并赋能？

- 站内运营自闭环
  - 内容运营
  - 活动运营
  - 用户运营
- 站内向站外投放闭环
  - 增长投放
- 站外向站内广告闭环
  - 广告投放

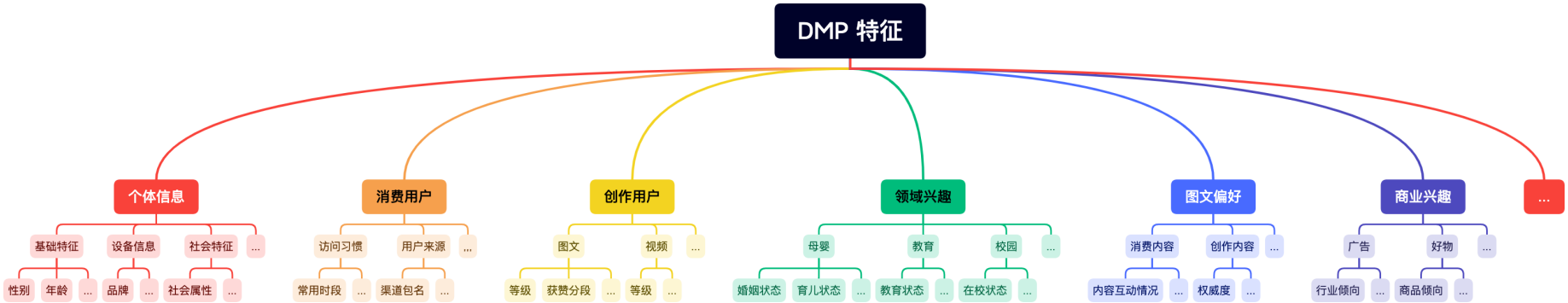


DMP 画像特征

当前有哪些画像特征？这些特征是如何分层分类的？量级如何？

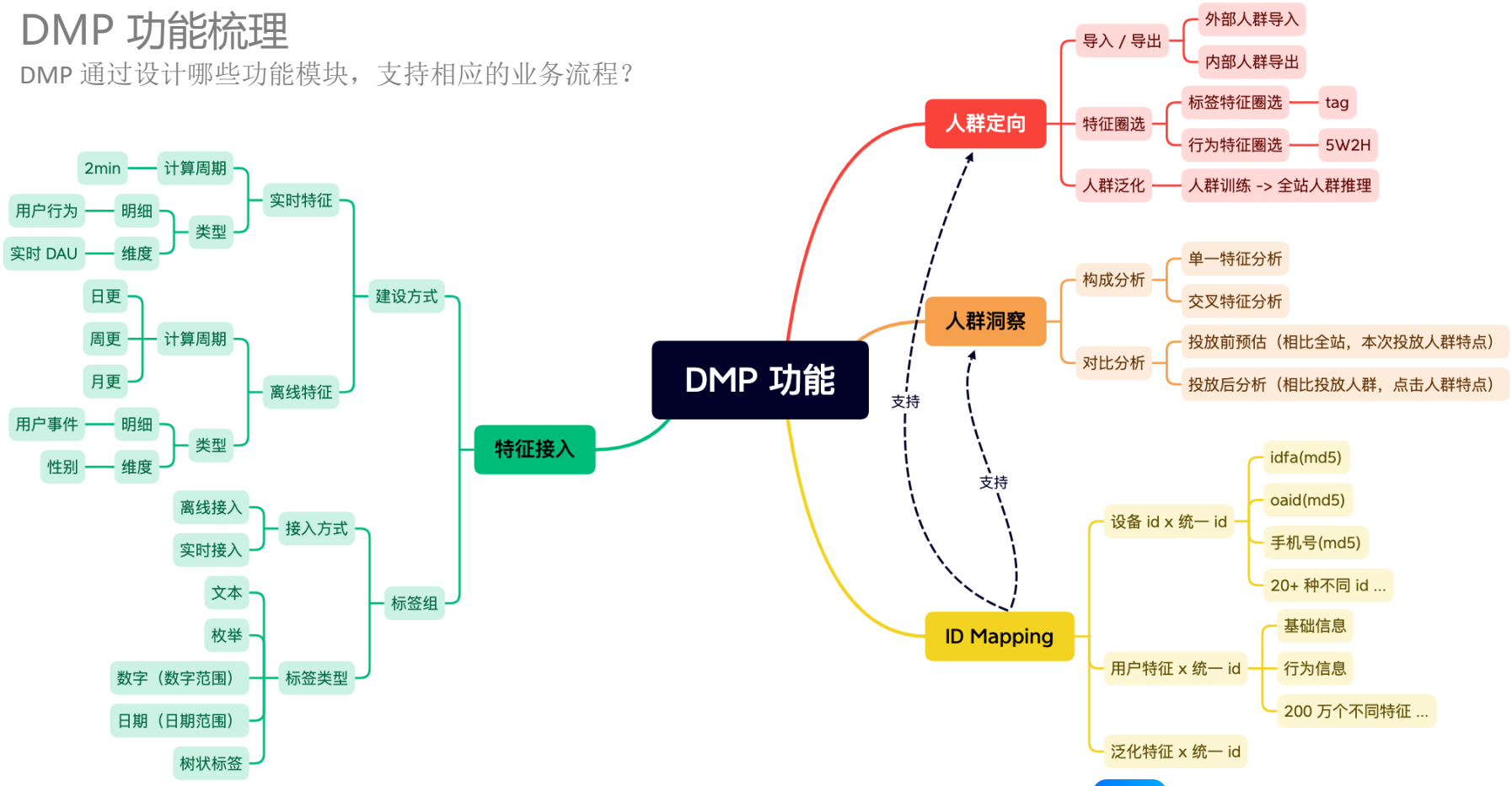
3 层级特征分类

- 一级分类 – 8 组
- 二级分类 – 40 组
- 标签组 – 120 个
  - 性别、手机品牌、话题兴趣…
- 标签 – 250 万
  - 男|女、 HUAWEI|Apple、对影视内容感兴趣程度高…



DMP 功能梳理

DMP 通过设计哪些功能模块，支持相应的业务流程？



知乎 有问题 就会有答案 | DataFun.

02

架构与实现

DMP 架构

DMP 通过设计怎样的架构来降低实现业务功能的复杂度？

拆分后，不同模块的设计重心

- 对外模块
  - DMP 接口：高稳定性、高并发高吞吐
  - DMP 前台：操作简单，低运营使用成本
  - DMP 后台：日常开发工作配置化，降低开发成本
- 业务模块
  - 人群圈选：可扩展。新增特征 0 成本，新增规则低成本。
  - 人群洞察：可扩展。新增特征 0 成本，新增洞察方式低成本。
  - 人群泛化：可扩展。新增泛化方式低成本。
  - 特征生产：扩展成本低。原子特征低成本生产，派生特征通过后台可配置
  - ID Mapping：屏蔽 ID 打通逻辑
  - 计算任务运维：屏蔽机器资源和任务依赖的逻辑
  - 存储：可扩展可持续，不因业务成长而导致成本大幅增加

DMP 平台功能盘点

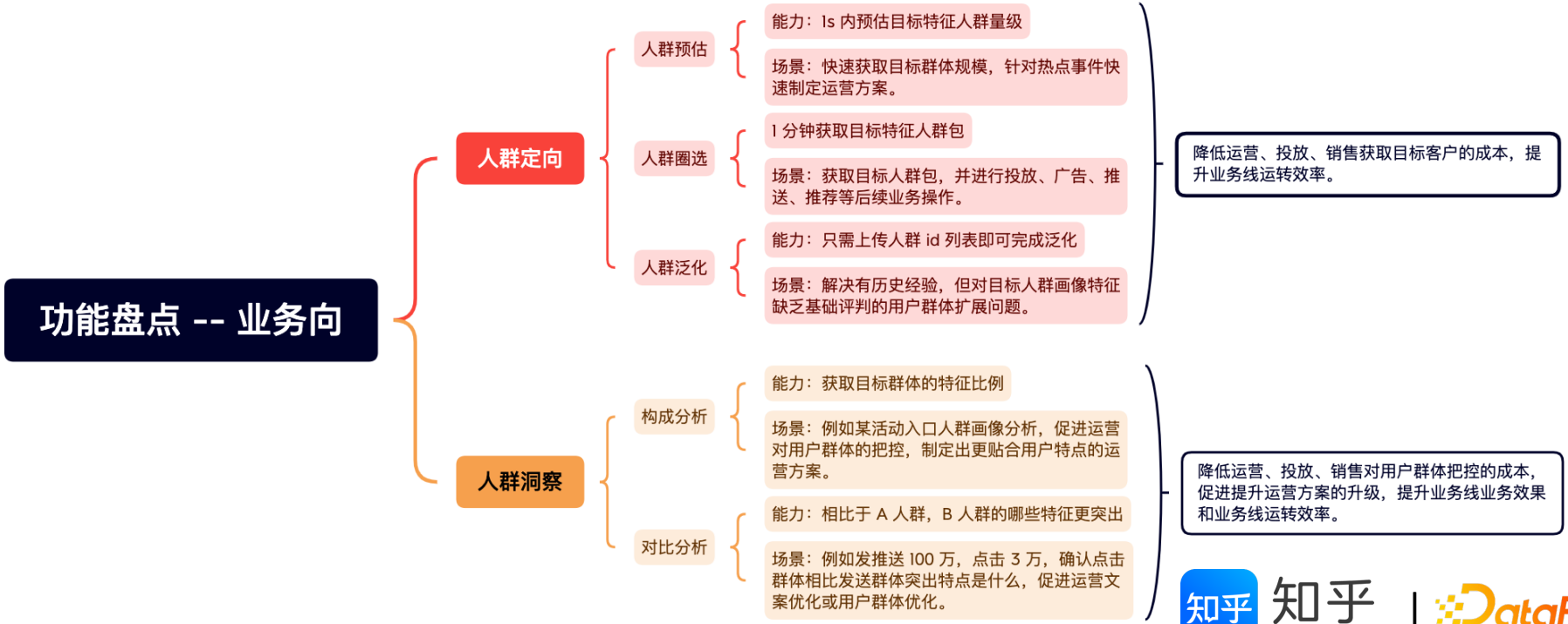
业务向

DMP 上线至今支持了

- 5+ 万人群定向
- 400+ 次人群洞察
- 60+ 次人群泛化

数据量级

- 120 个标签组
- 250 万个标签
- 1100 亿条用户 x 标签的数据

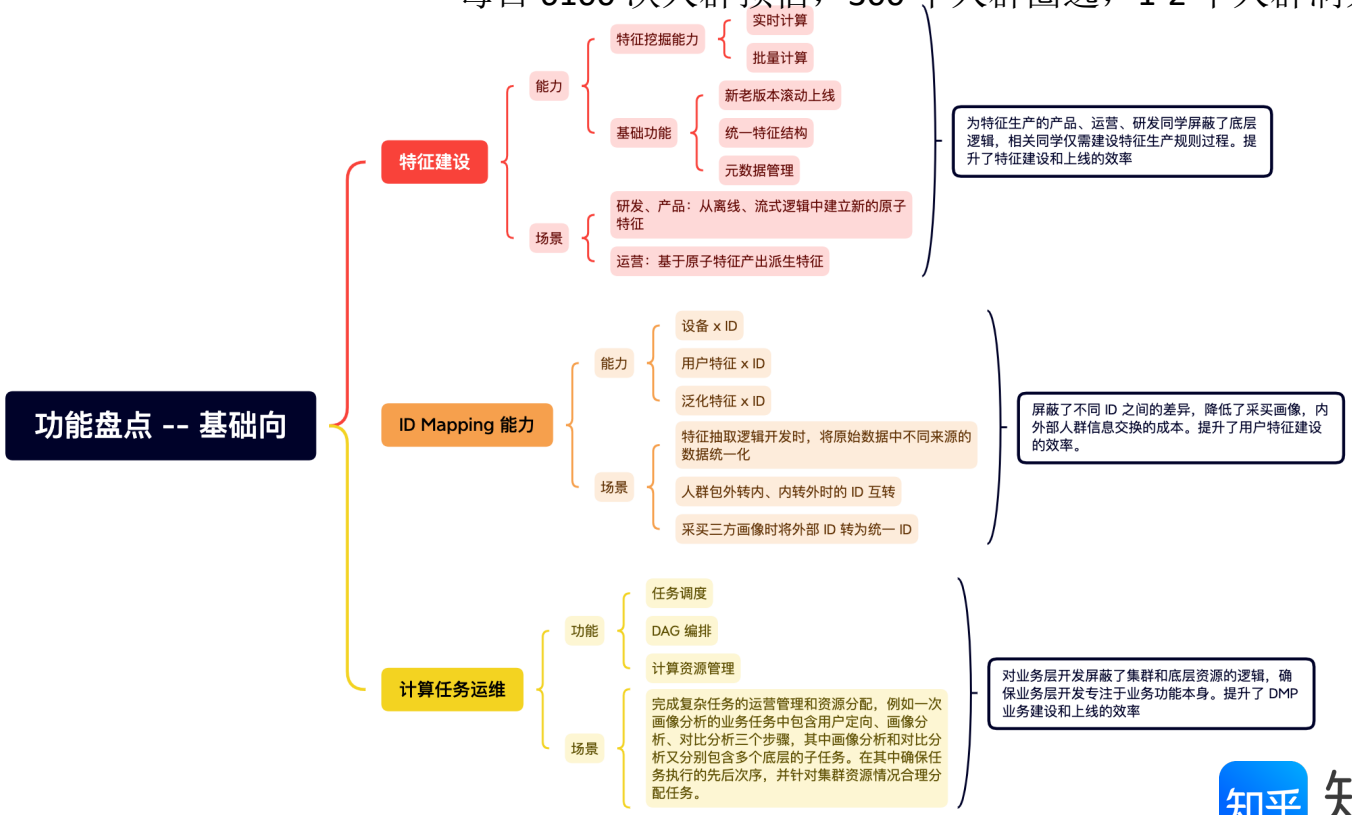


DMP 平台功能盘点

基础向

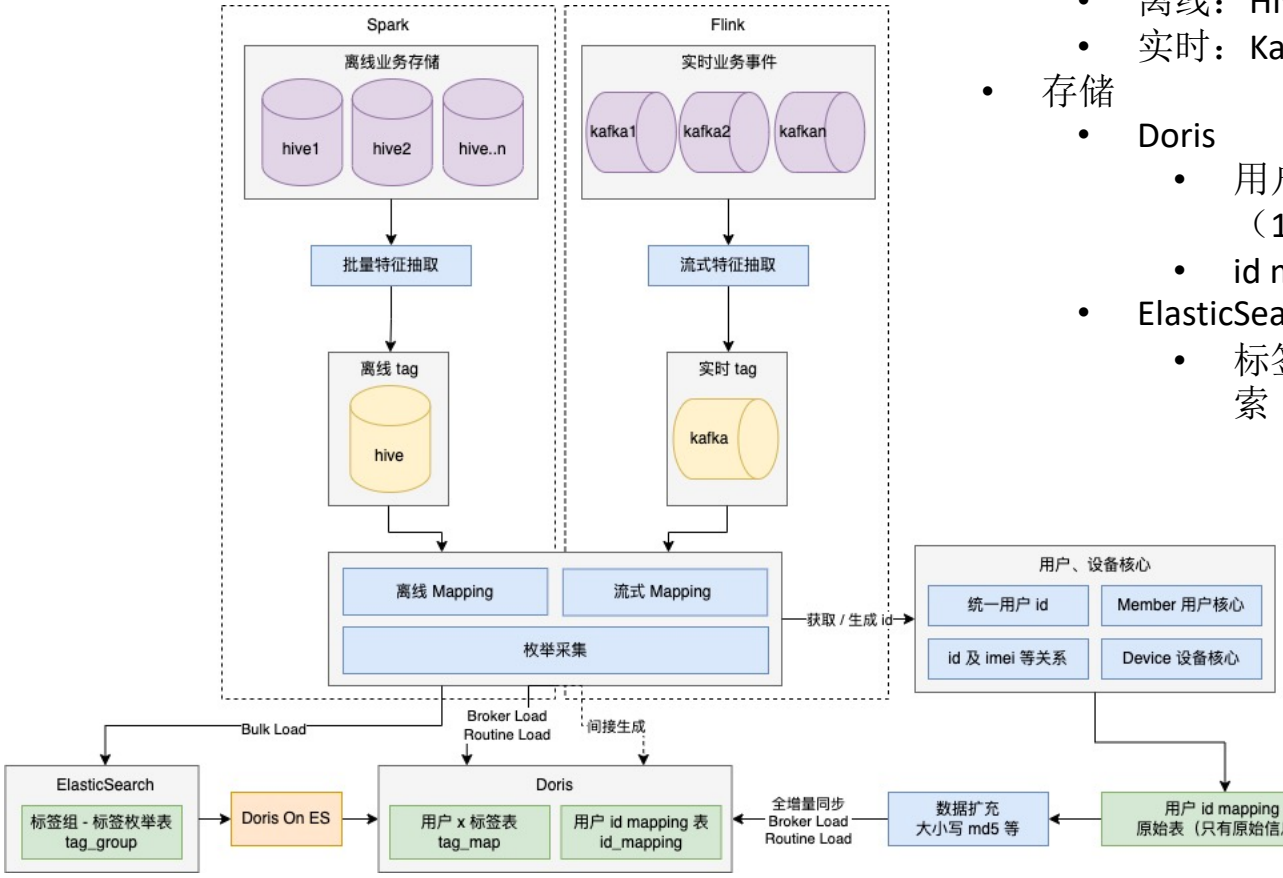
数据量级

- 每日 2.x TB 共 5 日 11 TB（离线、实时）特征（Doris）
- 120 个离线生产任务和 5 个实时生产任务
- 每日 6100 次人群预估，300 个人群圈选，1-2 个人群洞察，1 个人群泛化任务



特征数据链路及存储

DMP 的批量、流式特征如何建设并落地到相应的存储？



数据量级

- 特征链路
  - 离线：Hive -> 特征抽取 -> 离线标签 ->
  - 实时：Kafka -> 特征抽取 -> 实时标签 ->
- 存储
  - Doris
    - 用户 x 标签：用户有哪些标签（1100 亿）
    - id mapping：id 转化宽表（8.5 亿）
  - ElasticSearch
    - 标签枚举表：标签中文信息及搜索（250 万）

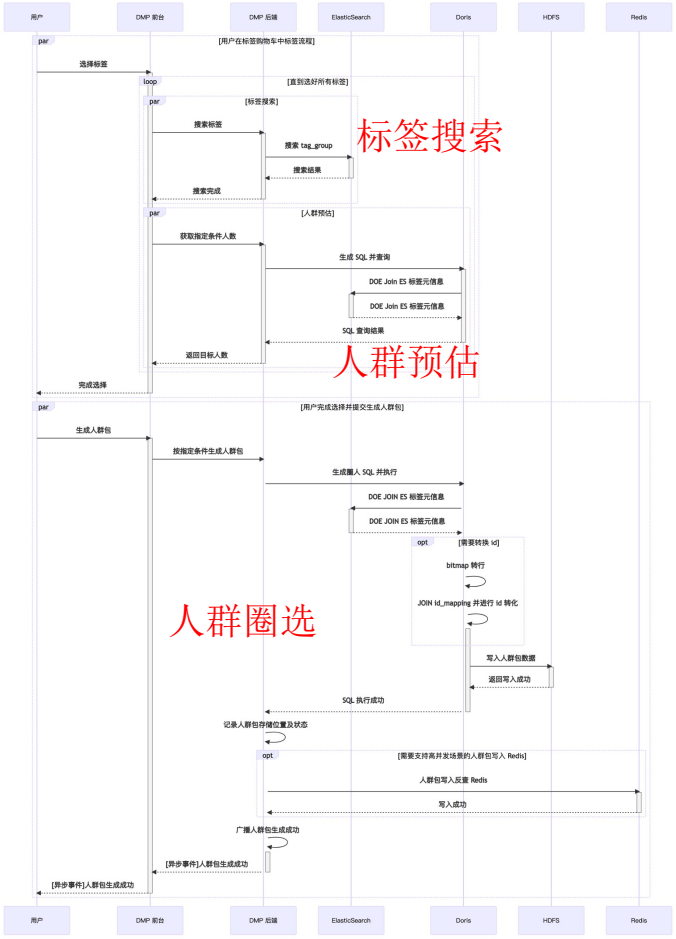
人群定向流程

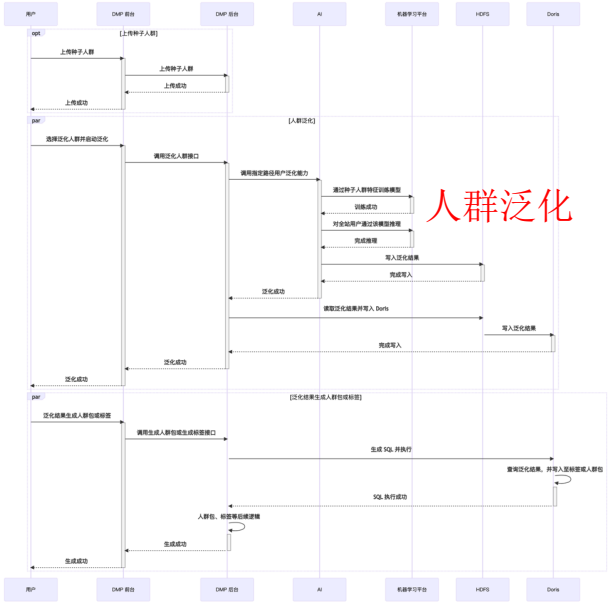
人群定向分哪几个过程？怎么做的？

人群定向流程很多，以下说几种典型的：

1. 标签加购物车 -> 圈选。
2. 传种子人群 -> 泛化。
3. 历史效果人群 -> 泛化 -> 叠加本次运营特点 -> 圈选。
4. 历史效果人群 -> 洞察 -> 重新生成标签关系 -> 圈选 -> 叠加历史正向人群 -> 泛化 -> 限制分发条件 -> 圈选。
5. 等等

对标签、历史人群进行组合、泛化、再限制条件再圈选、洞察，最后再调整等等



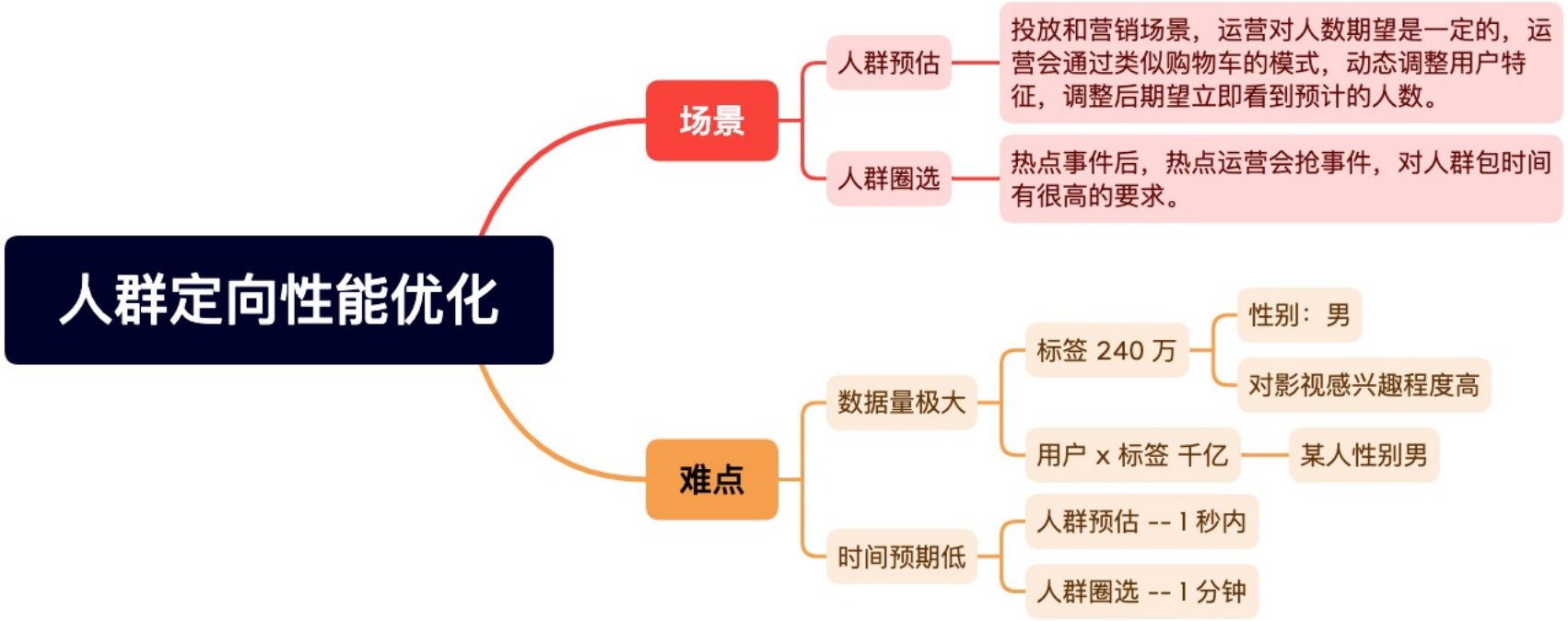


03

难点及解决方案

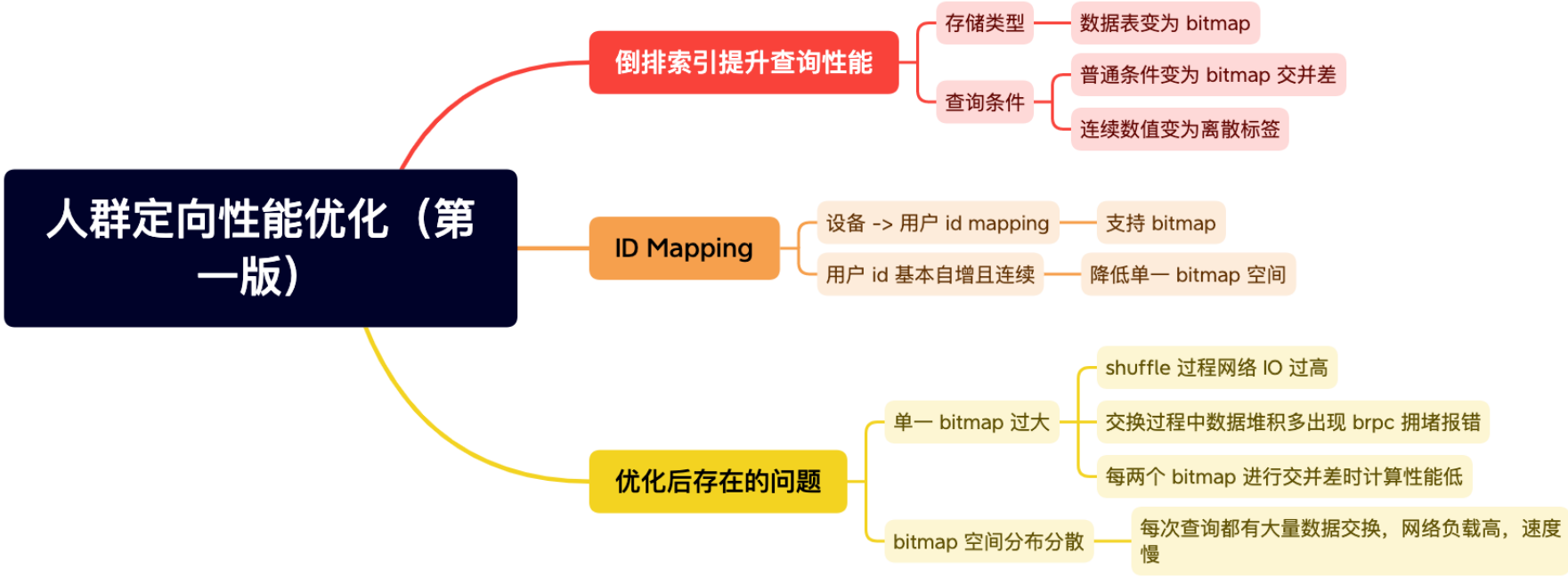
人群定向性能优化

背景和难点



人群定向性能优化

第一阶段



人群定向性能优化

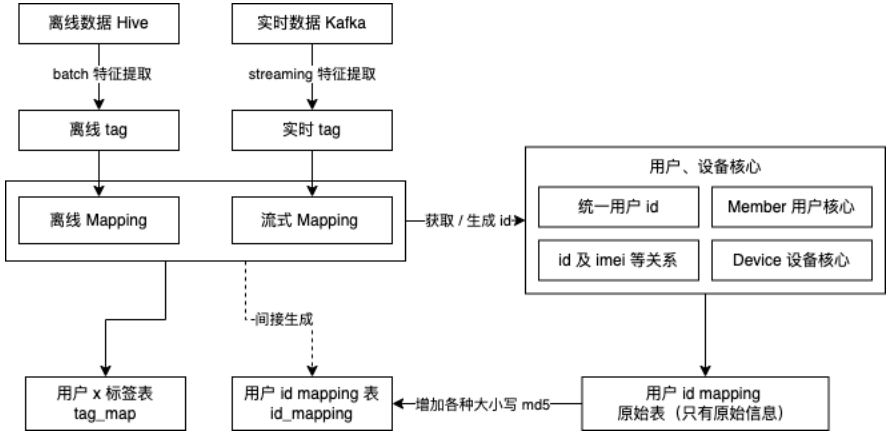
第一阶段 – 倒排索引及 id mapping

倒排索引

Field	Type	Null	Key	Default	Extra
partition_sign	VARCHAR(128)	No	true	NULL	
tag_group	BIGINT	No	true	NULL	
tag_value_id	BIGINT	No	true	NULL	
confidence	TINYINT	No	true	100	
members	BITMAP	No	false		BITMAP_UNION

- partition_sign 分区标识（日期、群组等）
- tag_group、tag_value_id 标签组和标签值 id
- condidence 置信度区间 50 – 55、55 – 60 …
- members 该特征用户 bitmap

ID Mapping



1. 特征提取，生成标签
2. 通过用户、设备等基础设施新增、获取一个统一用户 id
3. 通过统一 id 和其他信息的关联结果生成 id_mapping 表

人群定向性能优化

第一阶段 – 查询逻辑变更

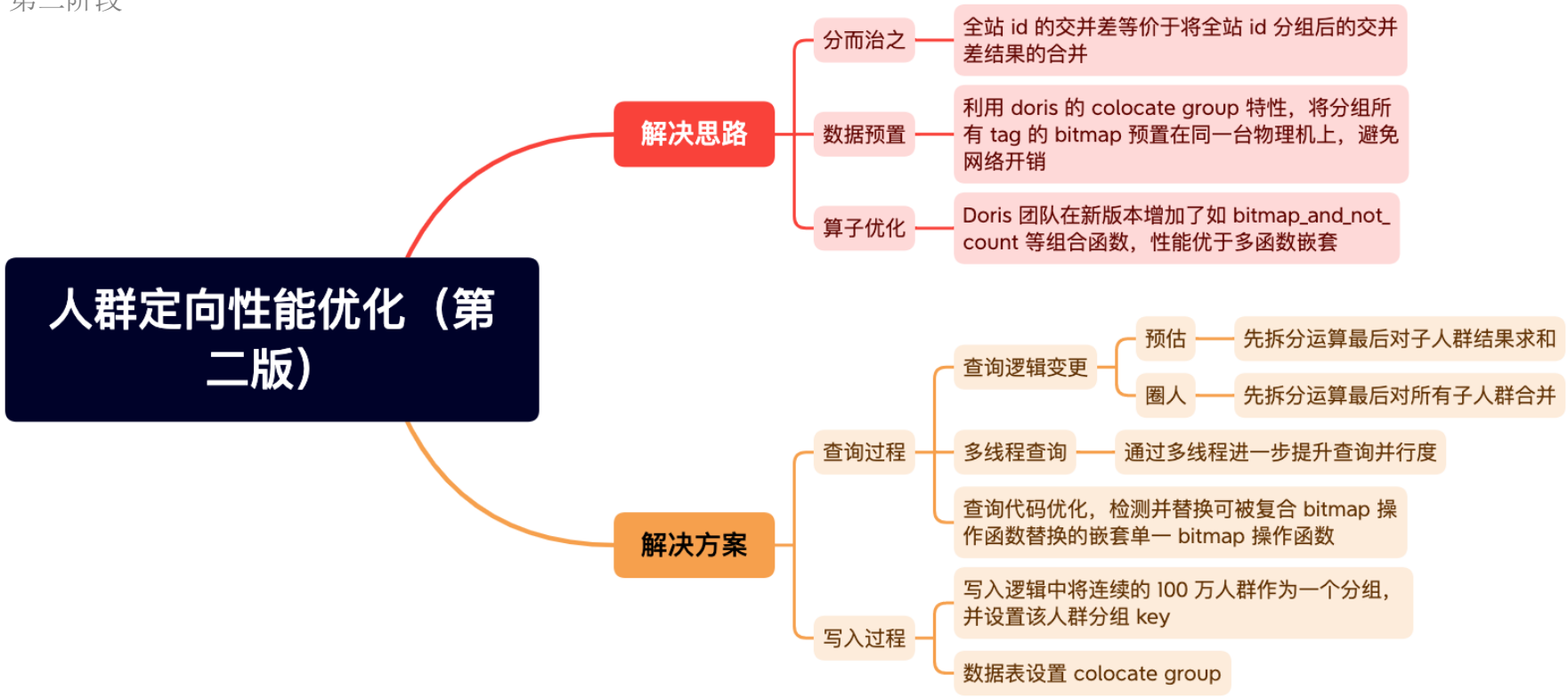
- 过滤条件从 where 条件中的 and、or、not 替换为查询聚合函数的 bitmap_and 等。
- 取用户方式从 id 列表转化为 id bitmap 结果

```
var filter Filter
switch tp {
case "and":
        filter = &AndFilter{BaseFiler{Type: tp, Filters: fs}}
case "or":
        filter = &OrFilter{BaseFiler{Type: tp, Filters: fs}}
case "not":
        filter = &NotFilter{BaseFiler{Type: tp, Filters: fs}}
case "tag":
        filter = &TagFilter{}
        err = utils.JSONUnmarshal(*bytes, &filter)
        if err != nil {
                return nil, err
        }
        err = GroupIsValid(filter.(*TagFilter))
        if err != nil {
                return nil, err
        }
        err = SortTagFiler(filter.(*TagFilter))
        if err != nil {
                return nil, err
        }
case "real_time_tag":
        flt := &TagFilter{}
```

```
func (filter AndFilter) GetSQL() (string, error) {
        if len(filter.Filters) == 1 {
                return filter.Filters[0].GetSQL()
        }
        if len(filter.Filters) == 2 {
                f1, err := filter.Filters[0].GetSQL()
                if err != nil {
                        return "", err
                }
                f2, err := filter.Filters[1].GetSQL()
                if err != nil {
                        return "", err
                }
                return fmt.Sprintf("select f1.bucket, bitmap_and(f1.members, f2.members) as members from (%s) f1,(%s) f2
        }
        return filter.getMultiAndSQL()
}
```

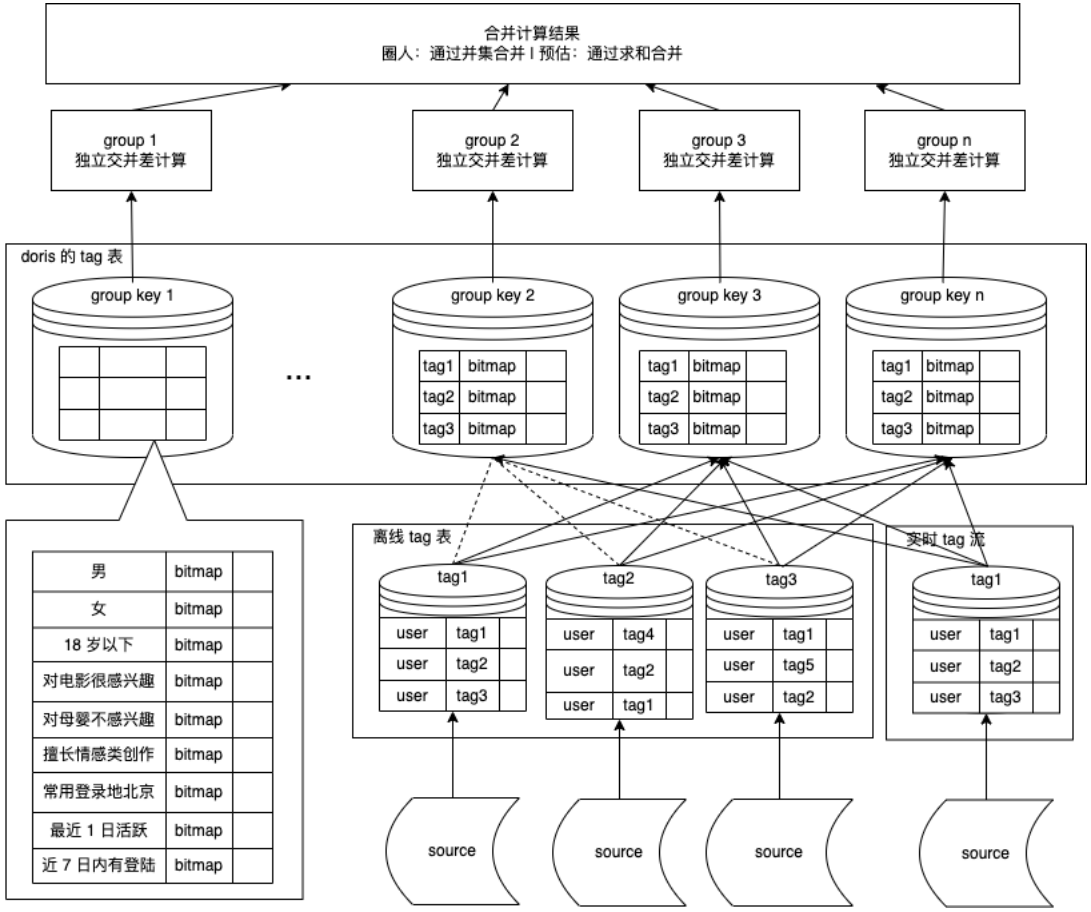
人群定向性能优化

第二阶段



人群定向性能优化

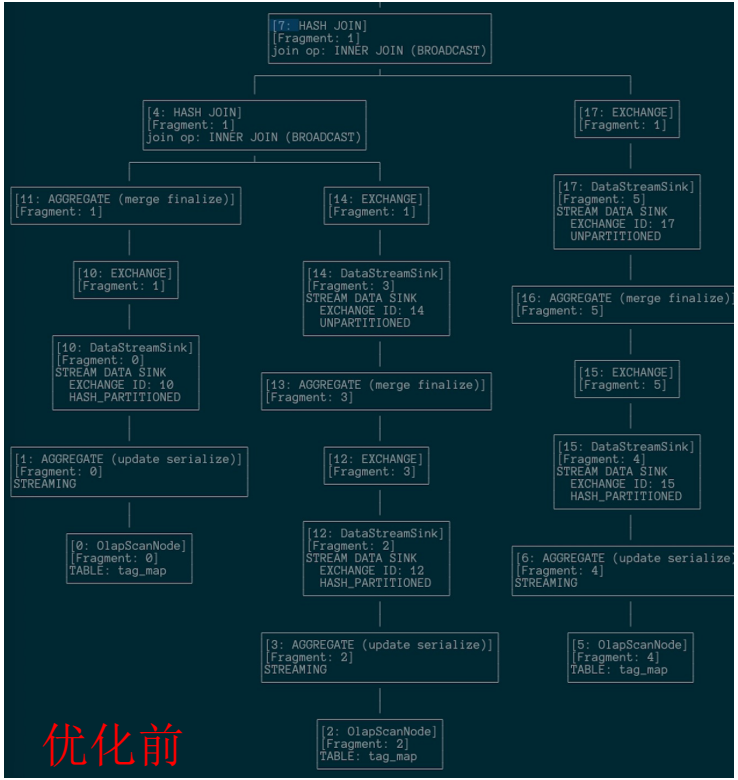
第二阶段 – 分而治之



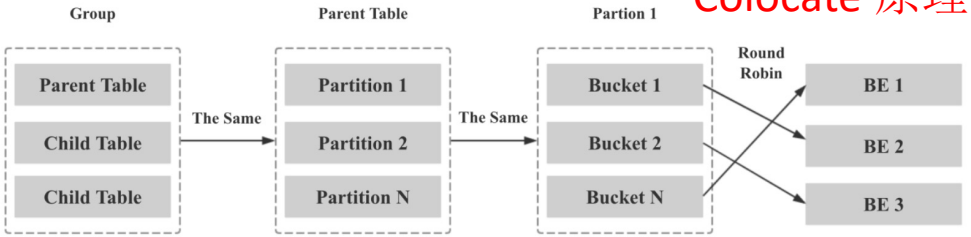
- 将连续一块的用户 id 的不同 tag 的数据，都增加统一的 group 字段进行分组。
- 在 group 内完成交并差后，最后进行数据汇总。
- 同时开启多线程模式，提升每组的计算效率。

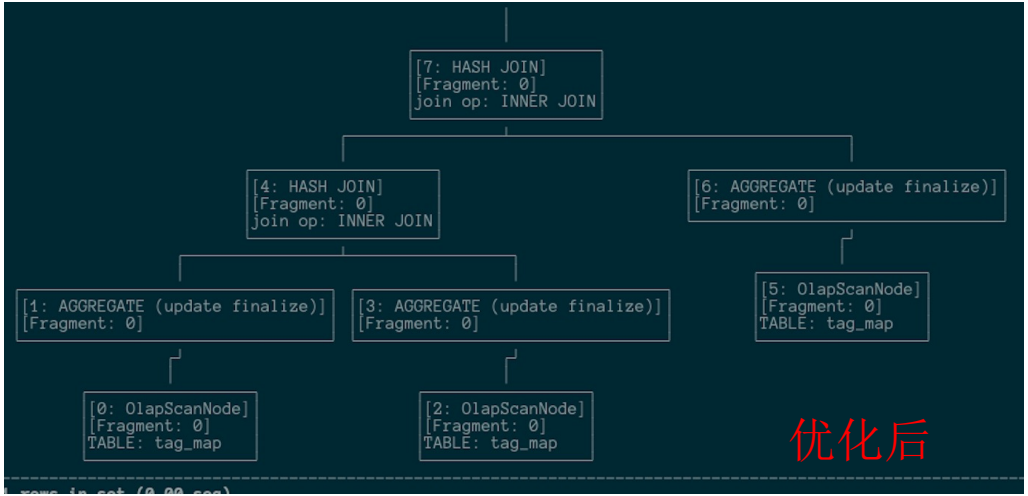
人群定向性能优化

第二阶段 – 数据预置 colocate join



Colocate 原理





知乎 有问题 就会有答案 | DataFun.

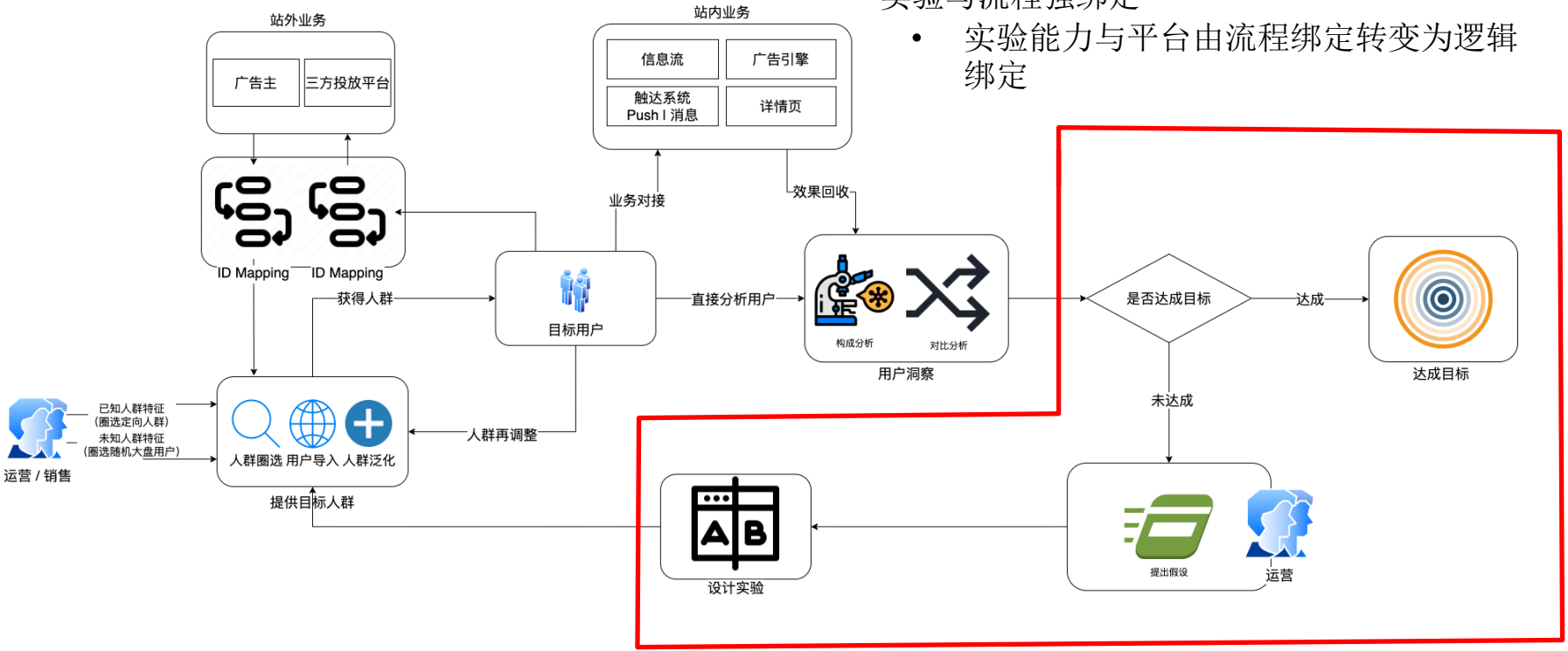
04

未来及展望

未来及展望

业务向

- 目标指向能力提升
  - 目标结果与平台由松耦合转变为强绑定
- 实验与流程强绑定
  - 实验能力与平台由流程绑定转变为逻辑绑定



未来及展望

技术向

- 提升查询效率
  - 自动探测 SQL 复杂查询条件预先合并成一个派生特征的 bitmap，预测和圈人时对复杂条件 SQL 重写为派生特征
- 提升导入速度
  - Spark 直接写 Doris Tablet 文件，并挂在到 FE

回顾

01 背景

DMP 业务
DMP 业务流程
DMP 画像特征
DMP 功能梳理

02 架构与实现

DMP 架构
DMP 平台功能盘点 – 业务向
DMP 平台功能盘点 – 基础向
特征数据链路及存储
人群定向流程

03 难点及解决方案

人群定向性能优化 - 第一阶段
人群定向性能优化 - 第二阶段

04 未来展望

业务向
技术向

非常感谢您的观看

知乎 有问题 就会有答案 | DataFun.