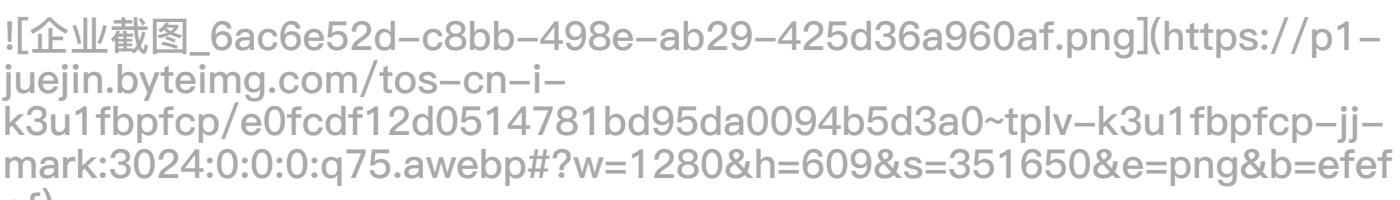
画像系统架构演进

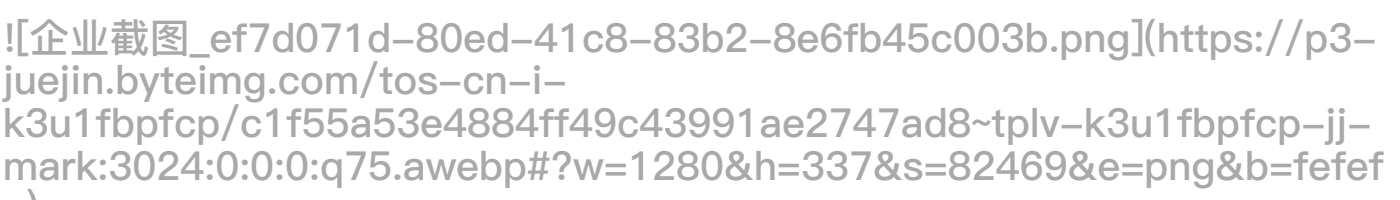
1. 系统概述

1.1 画像是什么

用户画像是指根据用户的属性、用户偏好、生活习惯、用户行为等信息而抽象出来的**标签化用户模型**。通俗来说就是给用户打“标签”，而标签是通过对用户信息分析而来的高度精炼的特征标识。

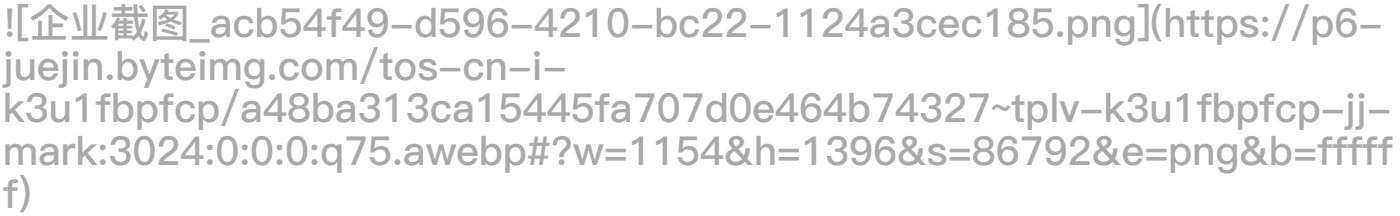


1.2 标签构建使用流程



1. 后台管理标签元数据、权限
2. 后端根据元数据信息作数据清洗出标签数据
3. 业务应用依据这份标签数据去实现各种需求

1.3 数据清洗流程



1. 大数据每天会清洗出T-1标签数据，落到中间表
2. 后端消费实时行为数据，清洗到当日及昨日临时表
3. 后端根据数据更新情况，实时统计出标签数据，最终落到应用表

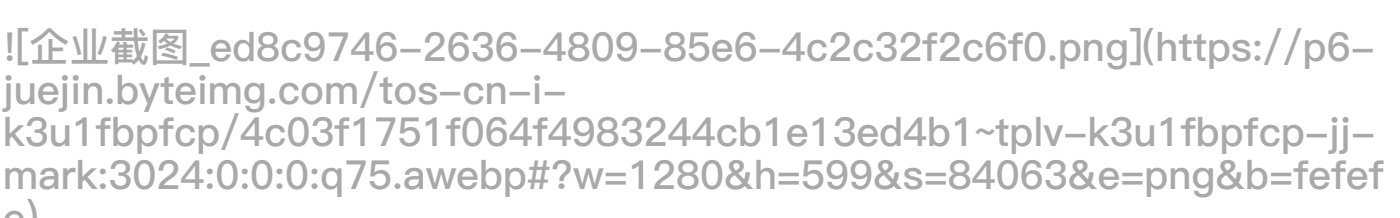
大家可能会奇怪我们为什么设计那么多张表，之所以这么设计，主要是我们考虑到了以下问题点：

1. 离线数据和实时数据会相互覆盖。如果都混用一张表，那数据就不准了
2. 离线T-1数据并不是0点立刻更新完成。意味着T-1中间表里的数据有可能是截止到昨天的，也有可能是截止到前天的，我们实时标签的计算需要兼顾这两种情况

2. 架构演进

2.1 旧版架构

2.1.1 流程

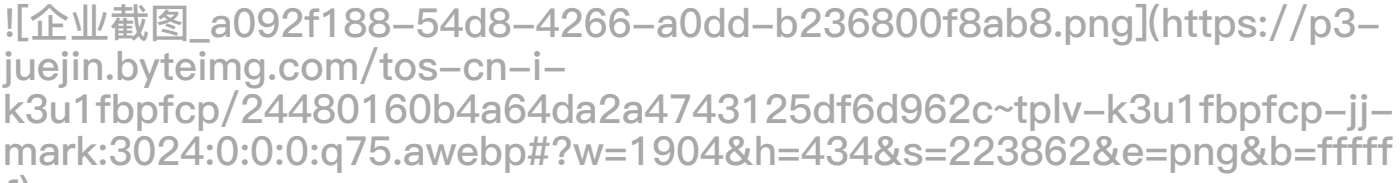


1. T-1数据写入ES
2. 实时数据结合历史数据
3. 处理好数据后写入ES
4. B端后台直接查询ES
5. C端接口直接查询ES

2.1.2 旧版架构痛点

1. B、C端未分离，导致系统经常出现稳定性问题
* 大数据离线&实时脚本、业务接口直接读写ES
* 离线作业启动的时候负载较高，需快速响应的业务超时情况较多
2. 业务发展过程中出现较多需联表查询的场景，ES难以支持

* ES联表查询父子索引的映射难以维护
* ES联表查询性能差负载高，会影响到在线业务



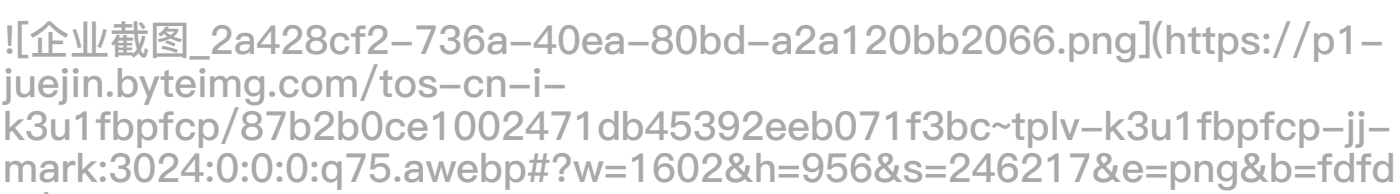
有段时间我们一整天都能收到业务请求超时的告警，特别是跑离线T-1数据的时候请求超时数激增，导致我们的非核心业务请求被迫重试/降级，核心业务也另寻他法不敢调用。另外，难以支持联表查询的问题也让我们不得不在业务代码层面不断造轮子来实现业务的诉求。虽有临时解决方案，但也浪费了很多开发资源，不是长久之计，所以我们迫切需要解决这些问题。由于ES扩容成本很高，我们首先需要找到新组件来替代ES。

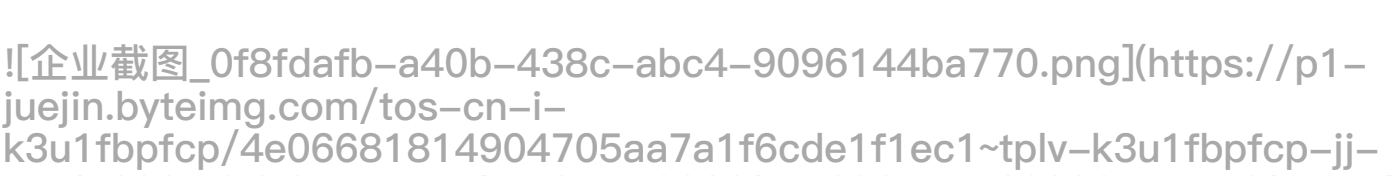
经过充分调研和实验，我们最终选定了ClickHouse/StarRocks + Hbase/Lindorm，接下来给大家介绍它们的特性。

2.2 ClickHouse/StarRocks特性介绍

2.2.1 优势

1. 列式存储，性能卓越
2. 类SQL语法，维护成本低
3. RoaringBitmap实现高效压缩存储
4. 丰富的位图函数，计算效率高
5. 支持丰富的外部表，扩展性好



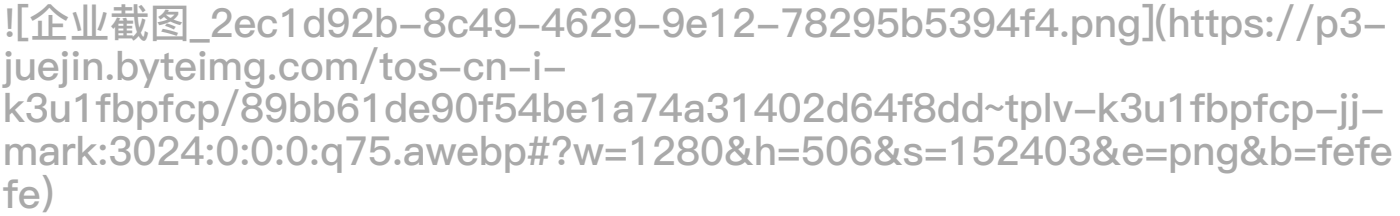


2.2.2 不足

1. 不支持高并发读写
2. 稳定性难以保障

2.3 Hhase/Lindorm特性介绍

2.3.1 优势



1. 非关系型数据库
2. 大容量存储
3. 支持热扩展
4. 高可靠性
5. 高性能读写

2.3.2 不足

1. 不支持复杂查询
2. 不支持反查

2.4 新版架构

纵观这两类组件的优劣势，容易发现它们的优势是基本互补的。我们B端的圈人/特征分析诉求可以用ClickHouse/StarRocks组件来承载，C端的业务查询可以使用Hbase/Lindorm，于是基于这两类组件的新架构就设计出来了。

2.4.1 流程

juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/a0438f7a8be744189acd9cc4f580dc2e~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=967&h=720&s=82861&e=png&b=fefefe)

1. 实时数据程序读取hbase的中间表
2. 离线/实时数据写入hbase
3. 离线/实时数据写入CH/SR
4. B端请求CH/SR做人群包圈选/特征分析
5. B端将人群包缓存写入hbase
6. C端直接读hbase获取用户标签/判断是否命中人群
7. C端从redis缓存获取用户标签/判断是否命中人群
8. redis从hbase同步缓存过期的数据

其中redis可以存储人群包缓存和实时性要求不高的标签缓存，纯内存模型理论上对比hbase查询速度也会更快。但考虑到多了一个组件的成本以及hbase的查询速率也满足我们的要求，所以我们暂未使用。

2.4.2 新版架构优势

1. 改造整个底层的数据流完成适配，实现B、C端分离，提高系统稳定性
2. 结合SR强大的bitmap及联表功能，完美支持了一系列复杂联表查询场景
3. hb替代es点查，P99从原200ms优化成10ms，实现10倍以上性能提升

![企业截图_00815249-76d8-4a77-9754-88353bfa2d07.png](https://p6-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/0ec816095a664b3e831904045537ec17~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1584&h=396&s=212294&e=png&b=1b1e20)

![企业截图_46763a53-f500-495b-a415-8dcdb0eba8b6.png](https://p1-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/4bbf3366dd7f4e47b0e600df87b357e5~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1584&h=250&s=86677&e=png&b=1d2025)

2.5 高可用建设

![企业截图_c68325bf-1174-4a51-bc87-26469a9b30b8.png](https://p3-juejin.byteimg.com/tos-cn-i-

k3u1fbpfcp/37e7863e1f3f40eea74fc2d3995f2de3~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=864&h=646&s=47599&e=png&b=ffffff)

1. BC端分离，只需实现C端
2. DTS成本高，应用实现双写
3. 写入用专属写域名区分（不切换）
4. 读取用一样的读域名（切换）

得益于BC端分离的架构，我们只需实现C端，也就是Hbase的高可用，这也一定程度上减少了冗余部署的成本。考虑到无法接受TB级数据DTS同步的带宽成本以及数据重做的时间成本，我们实现了一套基于双云web端架构的画像应用侧双写高可用架构。双云分别用不同的写域名，但用同一个读域名。当单云故障的时候，读域名切换到另一朵云即可。

3. 结语

本文先带大家了解了画像的定义，然后概述了我们画像系统标签的构建以及数据清洗流程，接着给大家展现了业务发展过程中我们旧版架构所暴露出来的问题，最后介绍了我们是如何选型新组件及设计新版架构来解决这些问题并实现高可用的。

画像系统目前已应用到我们全平台存量业务板块，基于高扩展性的架构设计，也持续有新项目外部数据接入这套系统。其在提升技术需求交付效率，还有赋能业务增长方面都发挥了至关重要的作用。当然，在设计并落地新架构的过程中，我们遇到了各种各样的问题，在不断攻克这些问题的过程中，我们也感受到了个人能力提升所带来的成就以及团队合作的乐趣。

此文章来自于37手游——利金明

原文链接: https://juejin.cn/post/7379626552689442866