**美团点评基于 Flink 的实时数仓建设实践**

| | | | |
|---|---|---|---|
| **笔记本:** | Flink | | |
| **创建时间:** | 2019/8/20 15:06 | **更新时间:** | 2019/8/20 15:07 |

# 引言

近些年，企业对数据服务实时化服务的需求日益增多。本文整理了常见实时数据组件的性能特点和适用场景，介绍了美团如何通过 Flink 引擎构建实时数据仓库，从而提供高效、稳健的实时数据服务。此前我们美团技术博客发布过一篇文章《流计算框架 Flink 与 Storm 的性能对比》，对 Flink 和 Storm 俩个引擎的计算性能进行了比较。本文主要阐述使用 Flink 在实际数据生产上的经验。

## 实时平台初期架构

在实时数据系统建设初期，由于对实时数据的需求较少，形成不了完整的数据体系。我们采用的是“一路到底”的开发模式：通过在实时计算平台上部署 Storm 作业处理实时数据队列来提取数据指标，直接推送到实时应用服务中。

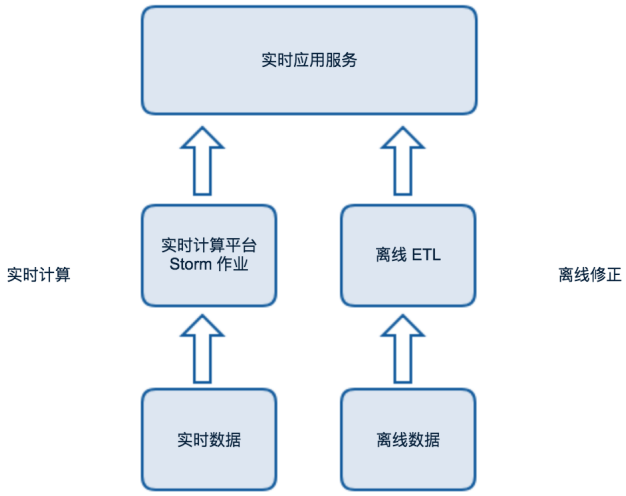


图1 初期实时数据架构

但是，随着产品和业务人员对实时数据需求的不断增多，新的挑战也随之发生。

1. 数据指标越来越多，“烟囱式”的开发导致代码耦合问题严重。
2. 需求越来越多，有的需要明细数据，有的需要 OLAP 分析。单一的开发模式难以应付多种需求。
3. 缺少完善的监控系统，无法在对业务产生影响之前发现并修复问题。

## 实时数据仓库的构建

为解决以上问题，我们根据生产离线数据的经验，选择使用分层设计方案来建设实时数据仓库，其分层架构如下图所示：

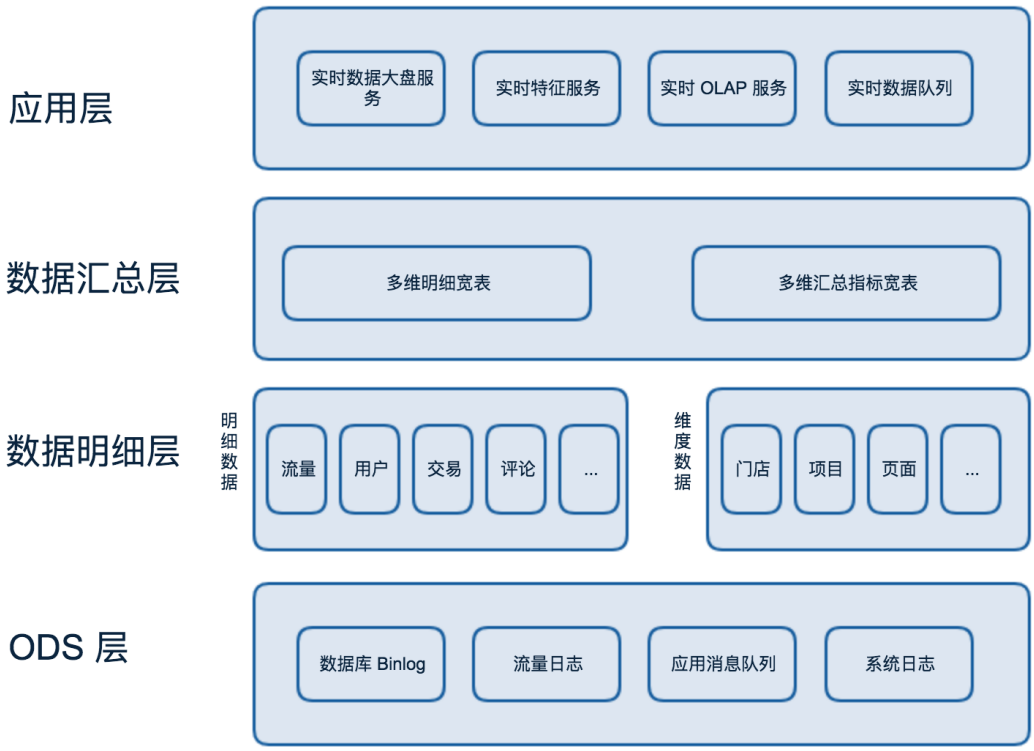


图2 实时数仓数据分层架构

**该方案由以下四层构成：**

1. ODS 层：Binlog 和流量日志以及各业务实时队列。
2. 数据明细层：业务领域整合提取事实数据，离线全量和实时变化数据构建实时维度数据。
3. 数据汇总层：使用宽表模型对明细数据补充维度数据，对共性指标进行汇总。
4. App 层：为了具体需求而构建的应用层，通过 RPC 框架对外提供服务。

通过多层设计我们可以将处理数据的流程沉淀在各层完成。比如在数据明细层统一完成数据的过滤、清洗、规范、脱敏流程；在数据汇总层加工共性的多维指标汇总数据。提高了代码的复用率和整体生产效率。同时各层级处理的任务类型相似，可以采用统一的技术方案优化性能，使数仓技术架构更简洁。

# 技术选型

## 1.存储引擎的调研

实时数仓在设计中不同于离线数仓在各层级使用同种储存方案，比如都存储在 Hive 、DB 中的策略。首先对中间过程的表，采用将结构化的数据通过消息队列存储和高速 KV 存储混合的方案。实时计算引擎可以通过监听消息消费消息队列内的数据，进行实时计算。而在高速 KV 存储上的数据则可以用于快速关联计算，比如维度数据。 其次在应用层上，针对数据使用特点配置存储方案直接写入。避免了离线数仓应用层同步数据流程带来的处理延迟。 为了解决不同类型的实时数据需求，合理的设计各层级存储方案，我们调研了美团内部使用比较广泛的几种存储方案。

存储方案列表如下：

| 方案 | 优势 | 劣势 |
| --- | --- | --- |
| MySQL | 1. 具有完备的事务功能，可以对数据进行更新。2. 支持 SQL，开发成本低。 | 1. 横向扩展成本大，存储容易成为瓶颈； 2. 实时数据的更新和查询频率都很高，线上单个实时应用请求就有 1000+ QPS；使用 MySQL 成本太高。 |
| Elasticsearch | 1. 吞吐量大，单个机器可以支持 2500+ QPS，并且集群可以快速横向扩展。2. Term 查询时响应速度很快，单个机器在 2000+ QPS时，查询延迟在 20 ms以内。 | 1. 没有原生的 SQL 支持，查询 DSL 有一定的学习门槛；2. 进行聚合运算时性能下降明显。 |
| Druid | 1. 支持超大数据量，通过 Kafka 获取实时数据时，单个作业可支持 6W+ QPS；2. 可以在数据导入时通过预计算对数据进行汇总，减少的数据存储。提高了实际处理数据的效率；3. 有很多开源 OLAP 分析框架。实现如 Superset。 | 1. 预聚合导致无法支持明细的查询；2. 无法支持 Join 操作；3. Append-only 不支持数据的修改。只能以 Segment 为单位进行替换。 |
| Cellar | 1. 支持超大数据量，采用内存加分布式存储的架构，存储性价比很高；2. 吞吐性能好，经测试处理 3W+ QPS 读写请求时，平均延迟在 1ms左右；通过异步读写线上最高支持 10W+ QPS。 | 1. 接口仅支持 KV，Map，List 以及原子加减等；2. 单个 Key 值不得超过 1KB ，而 Value 的值超过 100KB 时则性能下降明显。 |

根据不同业务场景，实时数仓各个模型层次使用的存储方案大致如下：

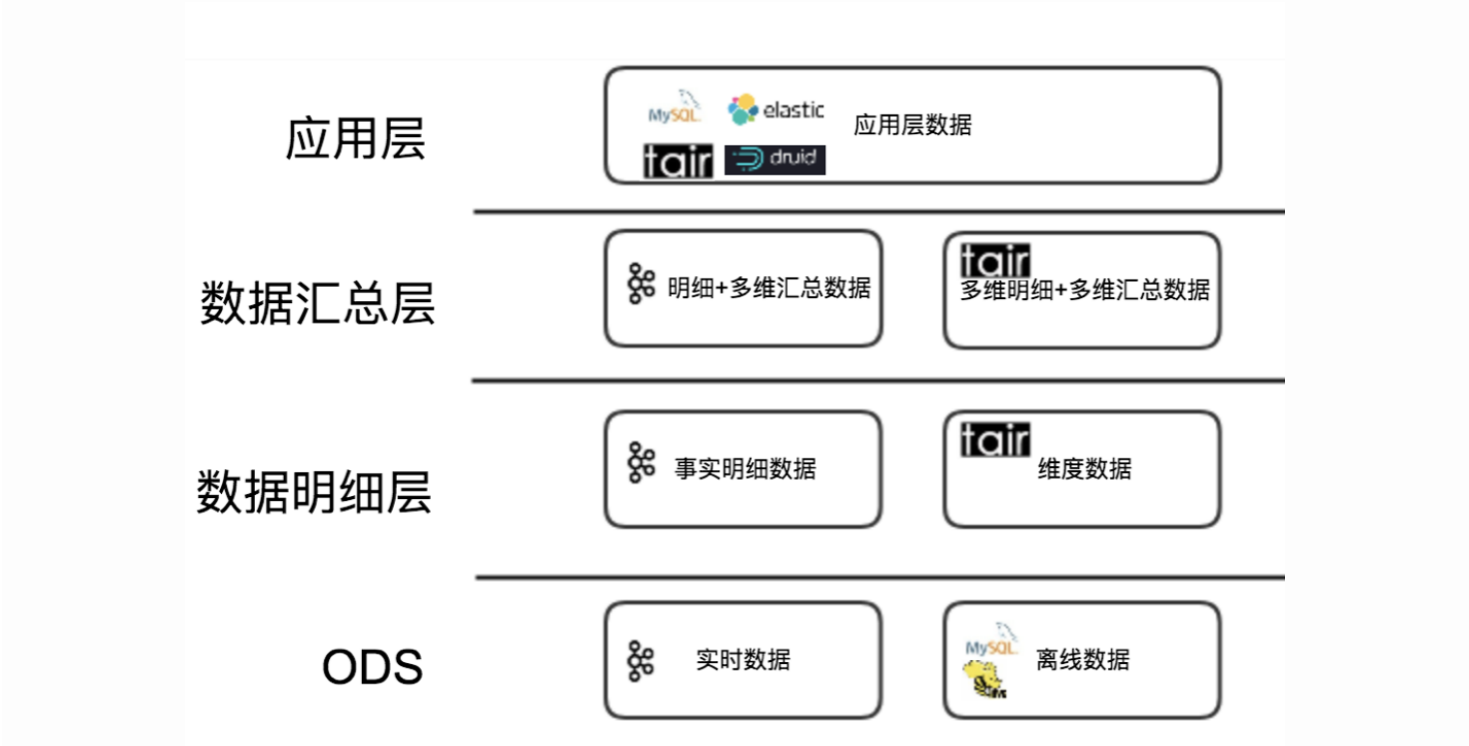


图3 实时数仓存储分层架构

1. **数据明细层** 对于维度数据部分场景下关联的频率可达 10w+ TPS，我们选择 Cellar（美团内部存储系统） 作为存储，封装维度服务为实时数仓提供维度数据。
2. **数据汇总层** 对于通用的汇总指标，需要进行历史数据关联的数据，采用和维度数据一样的方案通过 Cellar 作为存储，用服务的方式进行关联操作。
3. **数据应用层** 应用层设计相对复杂，再对比了几种不同存储方案后。我们制定了以数据读写频率 1000 QPS 为分界的判断依据。对于读写平均频率高于 1000 QPS 但查询不太复杂的实时应用，比如商户实时的经营数据。采用 Cellar 为存储，提供实时数据服务。对于一些查询复杂的和需要明细列表的应用，使用 Elasticsearch 作为存储则更为合适。而一些查询频率低，比如一些内部运营的数据。 Druid 通过实时处理消息构建索引，并通过预聚合可以快速的提供实时数据 OLAP 分析功能。对于一些历史版本的数据产品进行实时化改造时，也可以使用 MySQL 存储便于产品迭代。

## 2.计算引擎的调研

在实时平台建设初期我们使用 Storm 引擎来进行实时数据处理。Storm 引擎虽然在灵活性和性能上都表现不错。但是由于 API 过于底层，在数据开发过程中需要对一些常用的数据操作进行功能实现。比如表关联、聚合等，产生了很多额外的开发工作，不仅引入了很多外部依赖比如缓存，而且实际使用时性能也不是很理想。同时 Storm 内的数据对象 Tuple 支持的功能也很简单，通常需要将其转换为 Java 对象来处理。对于这种基于代码定义的数据模型，通常我们只能通过文档来进行维护。不仅需要额外的维护工作，同时在增改字段时也很麻烦。综合来看使用 Storm 引擎构建实时数仓难度较大。我们需要一个新的实时处理方案，要能够实现：

1. 提供高级 API，支持常见的数据操作比如关联聚合，最好是能支持 SQL。
2. 具有状态管理和自动支持久化方案，减少对存储的依赖。
3. 便于接入元数据服务，避免通过代码管理数据结构。
4. 处理性能至少要和 Storm 一致。

我们对主要的实时计算引擎进行了技术调研。总结了各类引擎特性如下表所示：

**实时计算方案列表如下：**

| 项目/引擎 | Storm | Flink | spark-treaming |
| --- | --- | --- | --- |
| API | 灵活的底层 API 和具有事务保证的 Trident API | 流 API 和更加适合数据开发的 Table API 和 Flink SQL 支持 | 流 API 和 Structured-Streaming API 同时也可以使用更适合数据开发的 Spark SQL |
| 容错机制 | ACK 机制 | State 分布式快照保存点 | RDD 保存点 |
| 状态管理 | Trident State状态管理 | Key State 和 Operator State两种 State 可以使用，支持多种持久化方案 | 有 UpdateStateByKey 等 API 进行带状态的变更，支持多种持久化方案 |
| 处理模式 | 单条流式处理 | 单条流式处理 | Mic batch处理 |
| 延迟 | 毫秒级 | 毫秒级 | 秒级 |

| | | | |
|---|---|---|---|
| 语义保障 | At Least Once，Exactly Once | Exactly Once，At Least Once | At Least Once |

从调研结果来看，Flink 和 Spark Streaming 的 API 、容错机制与状态持久化机制都可以解决一部分我们目前使用 Storm 中遇到的问题。但 Flink 在数据延迟上和 Storm 更接近，对现有应用影响最小。而且在公司内部的测试中 Flink 的吞吐性能对比 Storm 有十倍左右提升。综合考量我们选定 Flink 引擎作为实时数仓的开发引擎。

更加引起我们注意的是，Flink 的 Table 抽象和 SQL 支持。虽然使用 Strom 引擎也可以处理结构化数据。但毕竟依旧是基于消息的处理 API ，在代码层层面上不能完全享受操作结构化数据的便利。而 Flink 不仅支持了大量常用的 SQL 语句，基本覆盖了我们的开发场景。而且 Flink 的 Table 可以通过 TableSchema 进行管理，支持丰富的数据类型和数据结构以及数据源。可以很容易的和现有的元数据管理系统或配置管理系统结合。通过下图我们可以清晰的看出 Storm 和 Flink 在开发统过程中的区别。

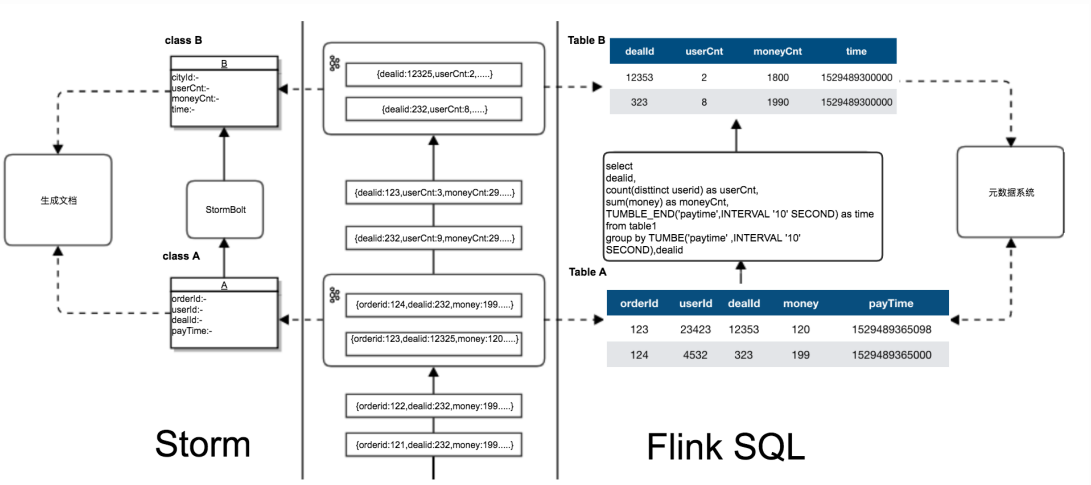


图4 Flink - Storm 对比图

在使用 Storm 开发时处理逻辑与实现需要固化在 Bolt 的代码。Flink 则可以通过 SQL 进行开发，代码可读性更高，逻辑的实现由开源框架来保证可靠高效，对特定场景的优化只要修改 Flink SQL 优化器功能实现即可，而不影响逻辑代码。使我们可以把更多的精力放到到数据开发中，而不是逻辑的实现。当需要离线数据和实时数据口径统一的场景时，我们只需对离线口径的 SQL 脚本稍加改造即可，极大地提高了开发效率。同时对比图中 Flink 和 Storm 使用的数据模型，Storm 需要通过一个 Java 的 Class 去定义数据结构，Flink Table 则可以通过元数据来定义。可以很好的和数据开发中的元数据，数据治理等系统结合，提高开发效率。

# Flink使用心得

在利用 Flink-Table 构建实时数据仓库过程中。我们针对一些构建数据仓库的常用操作，比如数据指标的维度扩充，数据按主题关联，以及数据的聚合运算通过 Flink 来实现总结了一些使用心得。

## 1. 维度扩充

数据指标的维度扩充，我们采用的是通过维度服务获取维度信息。虽然基于 Cellar 的维度服务通常的响应延迟可以在 1ms 以下。但是为了进一步优化 Flink 的吞吐，我们对维度数据的关联全部采用了异步接口访问的方式，避免了使用 RPC 调用影响数据吞吐。 对于一些数据量很大的流，比如流量日志数据量在 10W 条/秒这个量级。在关联 UDF 的时候内置了缓存机制，可以根据命中率和时间对缓存进行淘汰，配合用关联的 Key 值进行分区，显著减少了对外部服务的请求次数，有效的减少了处理延迟和对外部系统的压力。

## 2. 数据关联

数据主题合并，本质上就是多个数据源的关联，简单的来说就是 Join 操作。Flink 的 Table 是建立在无限流这个概念上的。在进行 Join 操作时并不能像离线数据一样对两个完整的表进行关联。采用的是在窗口时间内对数据进行关联的方案，相当于从两个数据流中各自截取一段时间的数据进行 Join 操作。有点类似于离线数据通过限制分区来进行关联。同时需要注意 Flink 关联表时必须有至少一个“等于”关联条件，因为等号两边的值会用来分组。

由于 Flink 会缓存窗口内的全部数据来进行关联，缓存的数据量和关联的窗口大小成正比。因此 Flink 的关联查询，更适合处理一些可以通过业务规则限制关联数据时间范围的场景。比如关联下单用户购买之前 30 分钟内的浏览日志。过大的窗口不仅会消耗更多的内存，同时会产生更大的 Checkpoint ，导致吞吐下降或 Checkpoint 超时。在实际生产中可以使用 RocksDB 和启用增量保存点模式，减少 Checkpoint 过程对吞吐产生影响。对于一些需要关联窗口期很长的场景，比如关联的数据可能是几天以前的数据。对于这些历史数据，我们可以将其理解为是一种已经固定不变的”维度“ 。可以将需要被关联的历史数据采用和维度数据一致的处理方法：”缓存 + 离线”数据方式存储，用接口的方式进行关联。另外需要注意 Flink 对多表关联是直接顺序链接的，因此需要注意先进行结果集小的关联。

## 3. 聚合运算

使用聚合运算时，Flink 对常见的聚合运算如求和、极值、均值等都有支持。美中不足的是对于 Distinct 的支持，Flink-1.6 之前的采用的方案是通过先对去重字段进行分组再聚合实现。对于需要对多个字段去重聚合的场景，只能分别计算再进行关联处理效率很低。为此我们开发了自定义的 UDAF，实现了 MapView 精确去重、BloomFilter 非精确去重、 HyperLogLog 超低内存去重方案应对各种实时去重场景。但是在使用自定义的 UDAF 时，需要注意 RocksDBStateBackend 模式对于较大的 Key 进行更新操作时序列化和反序列化耗时很多。可以考虑使用 FsStateBackend 模式替代。另外要注意的一点 Flink 框架在计算比如 Rank 这样的分析函数时，需要缓存每个分组窗口下的全部数据才能进行排序，会消耗大量内存。建议在这种场景下优先转换为 TopN 的逻辑，看是否可以解决需求。

下图展示一个完整的使用 Flink 引擎生产一张实时数据表的过程：

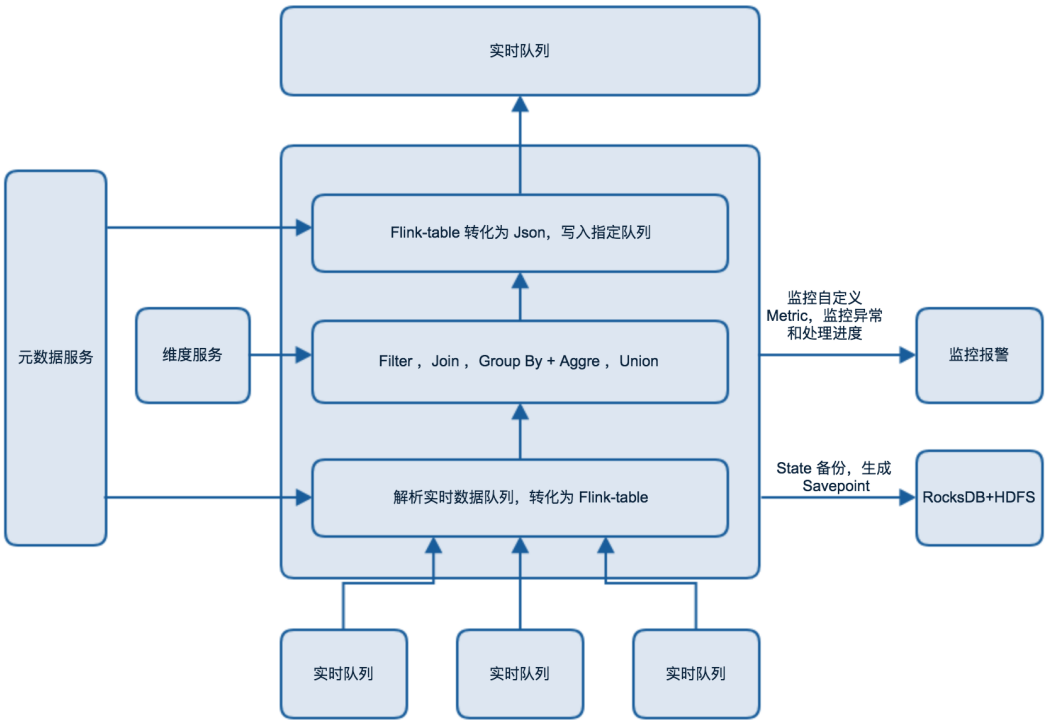


图5 实时计算流程图

## 实时数仓成果

通过使用实时数仓代替原有流程，我们将数据生产中的各个流程抽象到实时数仓的各层当中。实现了全部实时数据应用的数据源统一，保证了应用数据指标、维度的口径的一致。在几次数据口径发生修改的场景中，我们通过对仓库明细和汇总进行改造，在完全不用修改应用代码的情况下就完成全部应用的口径切换。在开发过程中通过严格的把控数据分层、主题域划分、内容组织标准规范和命名规则。使数据开发的链路更为清晰，减少了代码的耦合。再配合上使用 Flink SQL 进行开发，代码加简洁。单个作业的代码量从平均 300+ 行的 JAVA 代码 ，缩减到几十行的 SQL 脚本。项目的开发时长也大幅减短，一人日开发多个实时数据指标情况也不少见。

除此以外我们通过针对数仓各层级工作内容的不同特点，可以进行针对性的性能优化和参数配置。比如 ODS 层主要进行数据的解析、过滤等操作，不需要 RPC 调用和聚合运算。 我们针对数据解析过程进行优化，减少不必要的 JSON 字段解析，并使用更高效的 JSON 包。在资源分配上，单个 CPU 只配置 1GB 的内存即可满需求。而汇总层主要则主要进行聚合与关联运算，可以通过优化聚合算法、内外存共同运算来提高性能、减少成本。资源配置上也会分配更多的内存，避免内存溢出。通过这些优化手段，虽然相比原有流程实时数仓的生产链路更长，但数据延迟并没有明显增加。同时实时数据应用所使用的计算资源也有明显减少。

## 展望

我们的目标是将实时仓库建设成可以和离线仓库数据准确性，一致性媲美的数据系统。为商家，业务人员以及美团用户提供及时可靠的数据服务。同时作为到餐实时数据的统一出口，为集团其他业务部门助力。未来我们将更加关注在数据可靠性和实时数据指标管理。建立完善的数据监控，数据血缘检测，交叉检查机制。及时对异常数据或数据延迟进行监控和预警。同时优化开发流程，降低开发实时数据学习成本。让更多有实时数据需求的人，可以自己动手解决问题。

# 参考文献

流计算框架 Flink 与 Storm 的性能对比

# 作者简介

- 伟伦，美团到店餐饮技术部实时数据负责人，2017年加入美团，长期从事数据平台、实时数据计算、数据架构方面的开发工作。在使用 Flink 进行实时数据生产和提高生产效率上，有一些心得和产出。同时也积极推广 Flink 在实时数据处理中的实战经验。