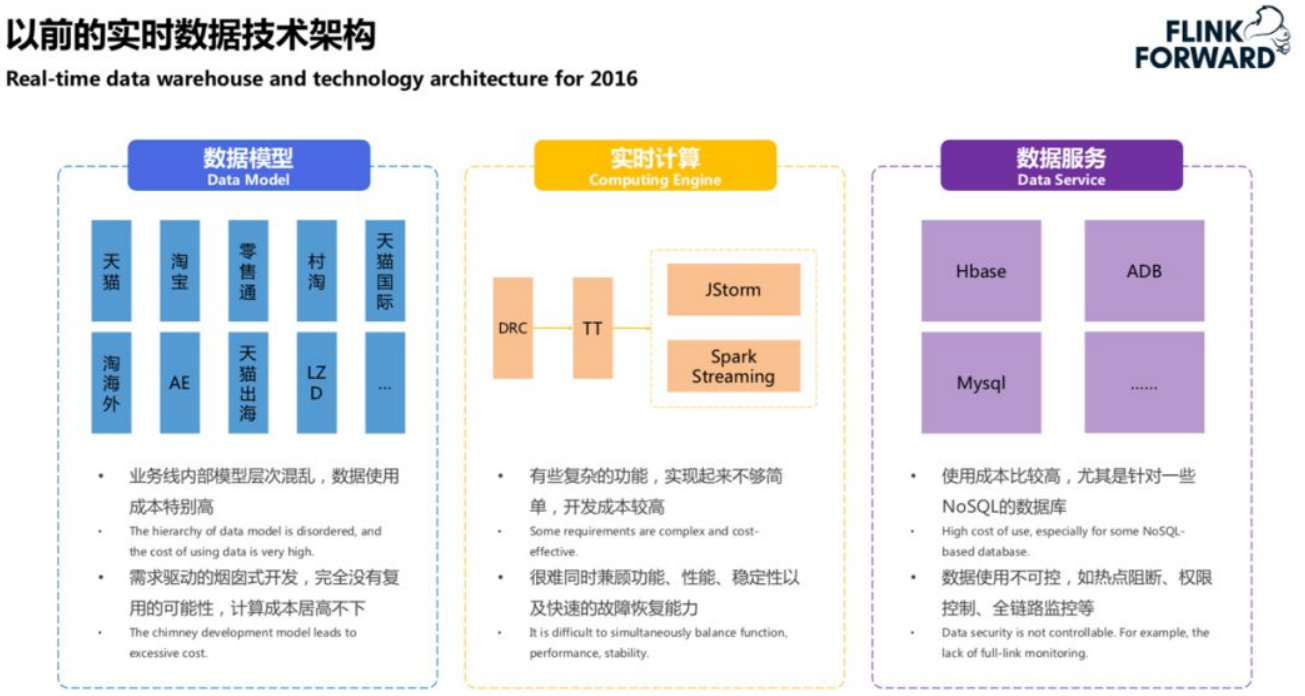
菜鸟供应链实时数仓的架构演进及应用场景

首先从三个方面简要介绍一下菜鸟早在 2016 年采用的实时数据技术架构：数据模型、实时计算和数据服务。

- **数据模型**。菜鸟最初使用的是需求驱动的、纵向烟囱式的开发模式，计算成本高且完全没有复用的可能性，同时也会导致数据一致性的问题；整个数据模型没有分层，业务线内部模型层次混乱，使得数据使用成本特别高。
- **实时计算**。该部分使用的是阿里的 JStorm 和 Spark Streaming，大多数情况下，二者可以满足实时计算的需求，但是对于有些复杂的功能，如物流和供应链场景，实现起来不够简单，开发成本较高；同时很难兼顾功能、性能、稳定性以及快速的故障恢复能力。
- **数据服务**。数据主要存储在 Hbase、MySQL 和 ADB 等不同类型的数据库中，然而对于很多运营人员来说，查询数据库的频率并不高，但使用数据库的成本较高，尤其针对一些 NoSQL 的数据库；也存在数据使用不可控，如热点阻断、权限控制以及全链路监控等问题。



针对以上问题，菜鸟在 2017 年对数据技术架构进行了一次比较大的升级改造，以下将详细介绍。

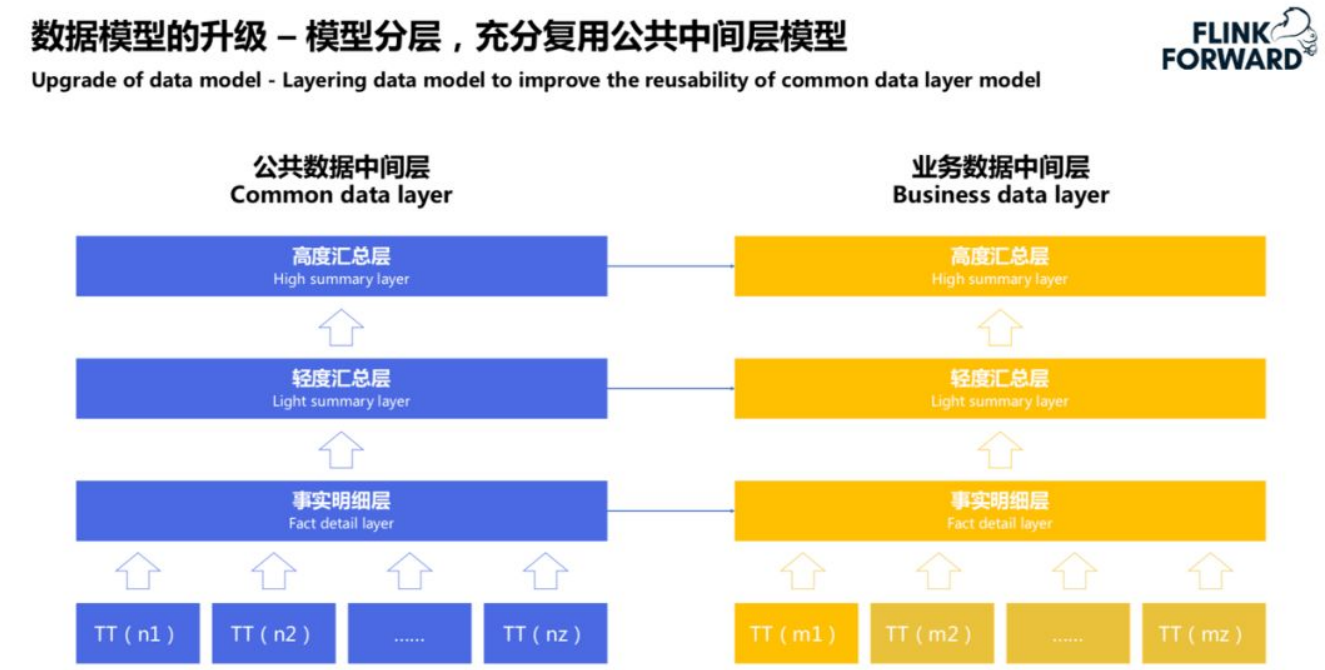
数据模型升级

数据模型的升级主要是模型分层，充分复用公共中间层模型。之前的模式是从数据源 TT（如 Kafka）中抽取数据并进行加工，产生一层式的表结构。新版本的数据模型进行了分层：

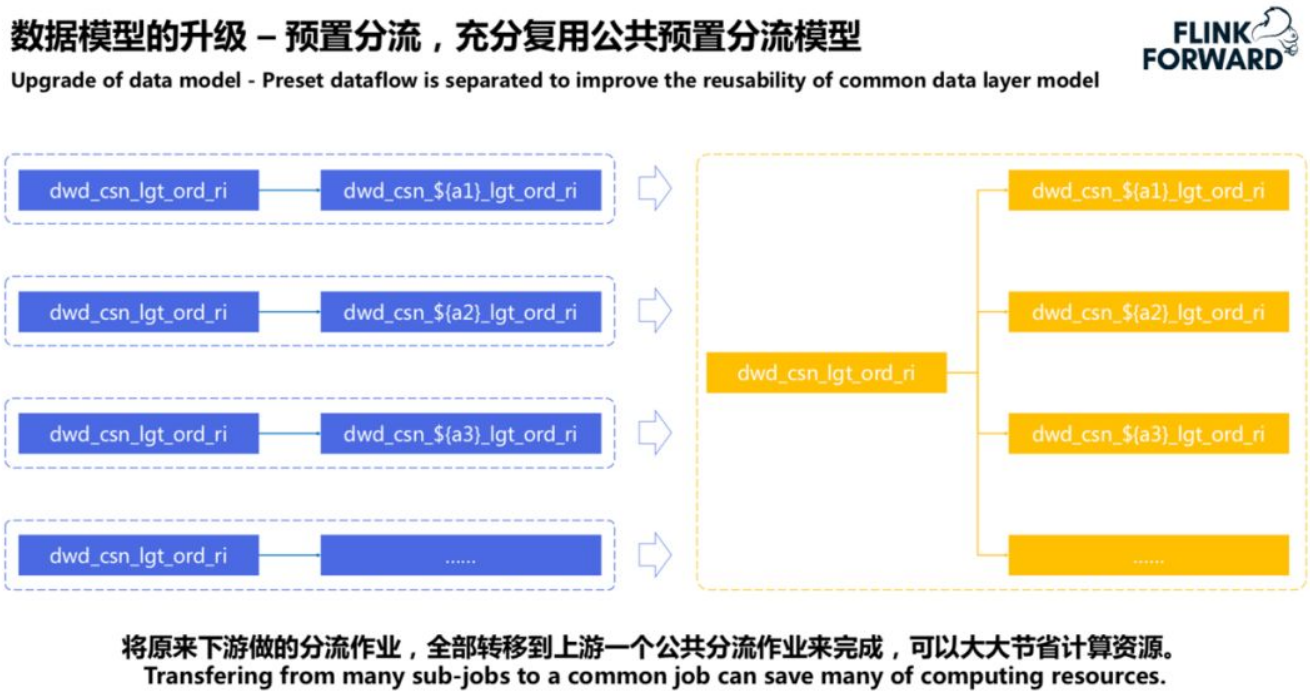
- 第一层是**数据采集**，支持多种数据库中的数据采集，同时将采集到的数据放入消息中间件中；

- 第二层是**事实明细层**，基于TT的实时消息产生事实明细表，然后再写入TT的消息中间件中，通过发布订阅的方式汇总到第三、四层，分别是轻度汇总层和高度汇总层。
- 第三层**轻度汇总层**适合数据维度、指标信息比较多的情况，如大促统计分析的场景，该层的数据一般存入阿里自研的 ADB 数据库中，用户可以根据自己的需求筛选出目标指标进行聚合；
- 而第四层**高度汇总层**则沉淀了一些公共粒度的指标，并将其写入 Hbase 中，支持大屏的实时数据显示场景，如媒体大屏、物流大屏等。

原本采用的开发模式各个业务线独立开发，不同业务线之间不考虑共性的问题，但物流场景中，很多功能需求其实是类似的，这样往往会造成资源的浪费，针对该问题进行的改造首先是抽象出横向的公共数据中间层（左侧蓝色），然后各个业务线在此基础上分流自己的业务数据中间层（右侧黄色）。



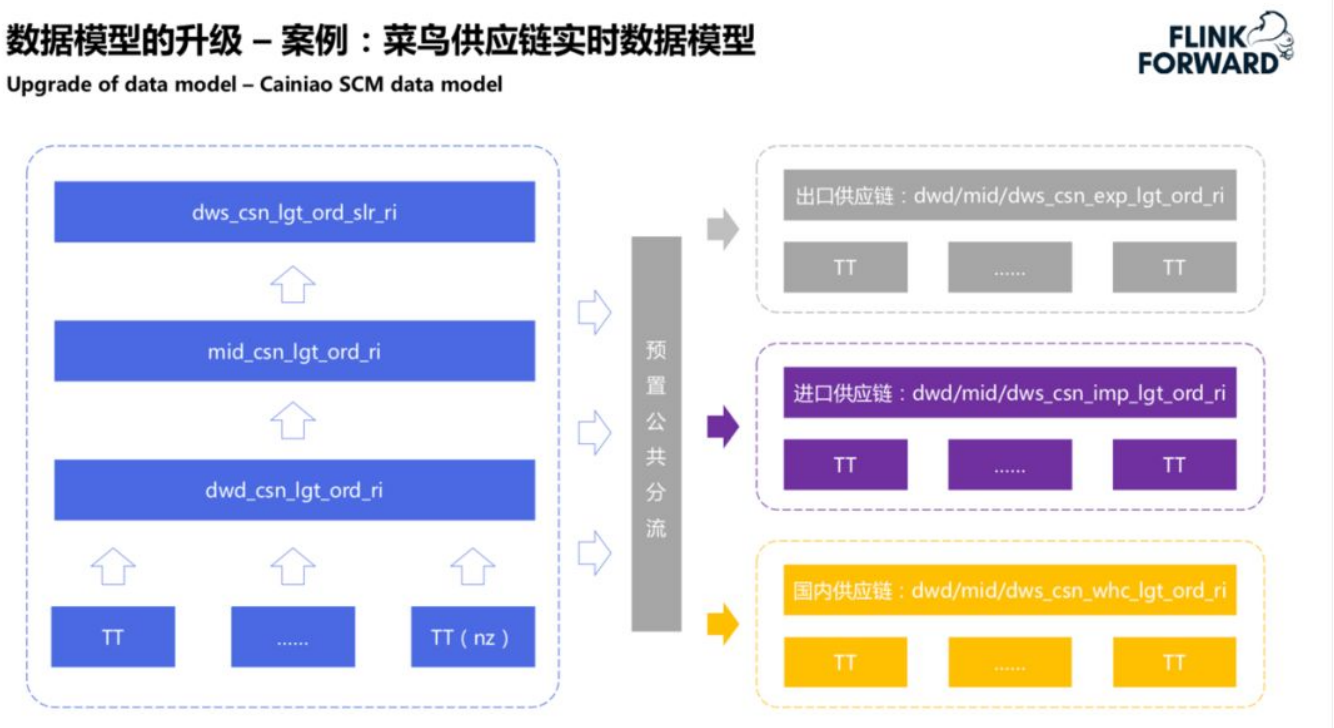
前面介绍的业务线分流由预置的公共分流任务来实现，即将原来下游做的分流作业，全部转移到上游的一个公共分流作业来完成，充分复用公共预置分流模型，大大节省计算资源。



下面介绍一个数据模型升级的具体案例—菜鸟供应链实时数据模型。

- 下图左侧是前面介绍的公共数据中间层，包括整个菜鸟横向的物流订单、大盘物流详情和公共粒度的一些数据，在此基础上菜鸟实现了预置公共分流，从物流订单、物流详情中拆分出个性化业务线的公共数据中间层，包括国内供应链、进口供应链以及出口供应链等。
- 基于已经分流出来的公共逻辑，再加上业务线个性化TT的消息，产出各业务线的业务数据中间层。
- 以进口供应链为例，其可能从公共业务线中分流出物流订单和物流详情，但是海关信息、干线信息等都在自己的业务线进口供应链的TT中，基于这些信息会产生该业务线的业务数据中间层。

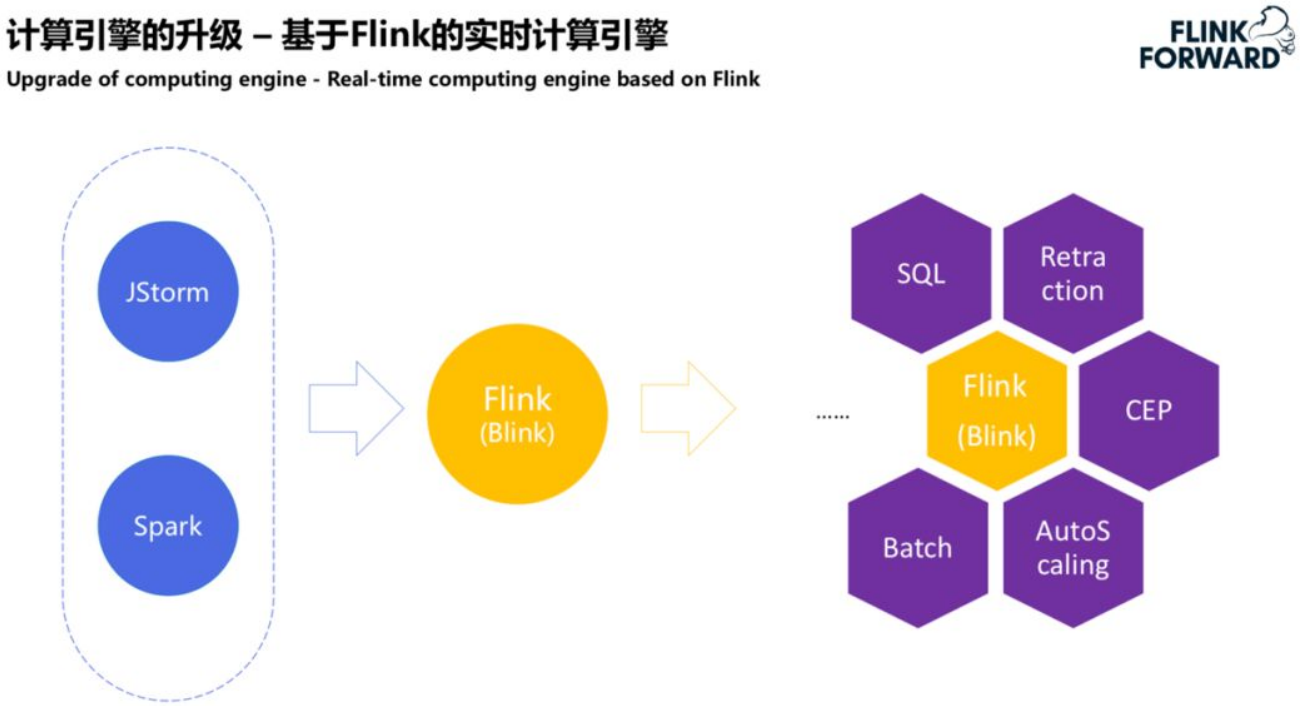
借助前面所述的设计理念，再加上实时的模型设计规范和实时的开发规范，大大提升了数据模型的易用性。



计算引擎升级

菜鸟最初的计算引擎采用的是阿里内部研发的 JStorm 和 Spark Streaming，可以满足大多数场景的需求，但针对一些复杂的场景，如供应链、物流等，会存在一些问题。因此，菜鸟在 2017 年全面升级为基于 Flink 的实时计算引擎。当时选择 Flink 的主要原因是：

- Flink 提供的很多功能非常适用于解决供应链场景下的需求，菜鸟内部提炼了一套 Flink 的 SQL 语法，简单易用且标准化，大大提升了开发效率。
- 此外，Flink 内置的基于 state 的 Retraction 的机制可以很好地支持供应链场景下的取消订单、换配需求的实现；
- 后来推出的 CEP 功能使得物流、供应链中实时超时统计需求的实现变得更加简单；
- AutoScaling 等自动优化的方案可以使得菜鸟省去了一些资源配置等方面的复杂性和成本；
- 半智能功能如批流混合等也较好地满足菜鸟业务的实际需求。



下面介绍三个与计算引擎升级相关的案例。

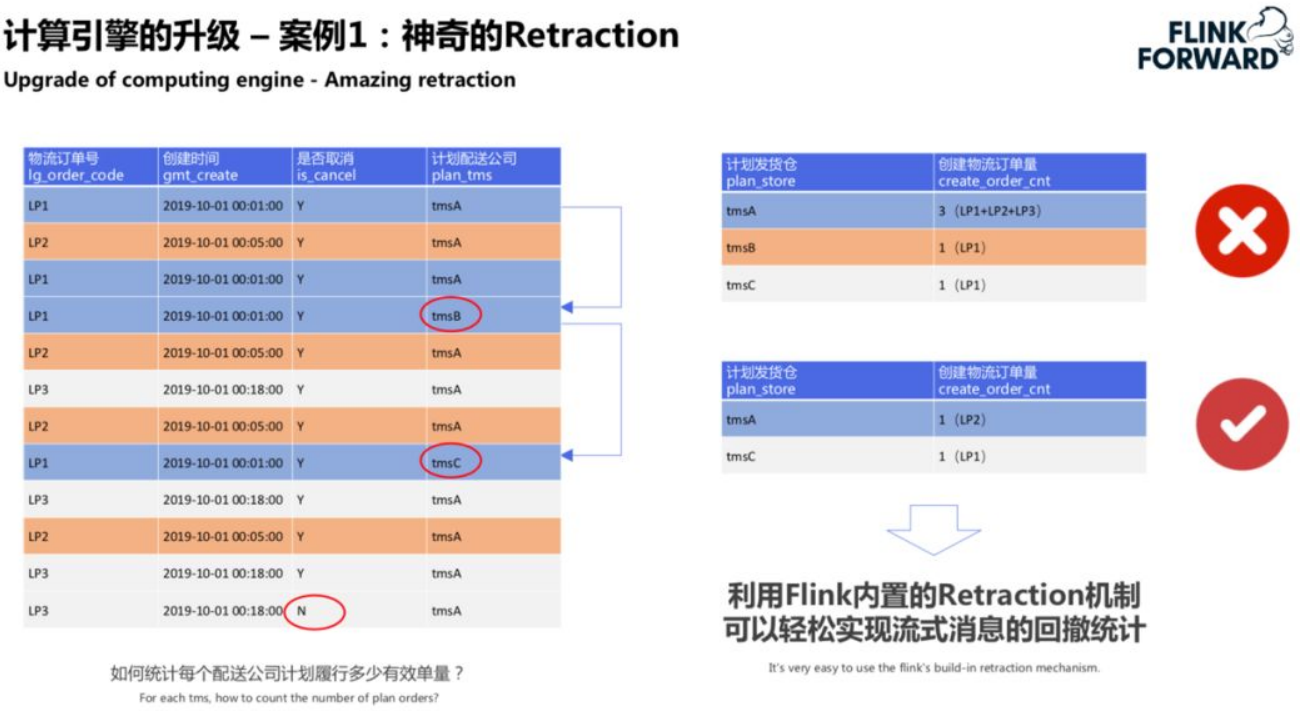
案例 1：基于 state 的 Retraction

下图左侧是一个物流订单表，包含了四列数据，即物流订单号、创建时间、是否取消和计划配送公司。假设有一个需求是统计某个配送公司计划履行的有效单量是多少，该需求看起来简单，实际实现过程中有有一些问题需要注意。

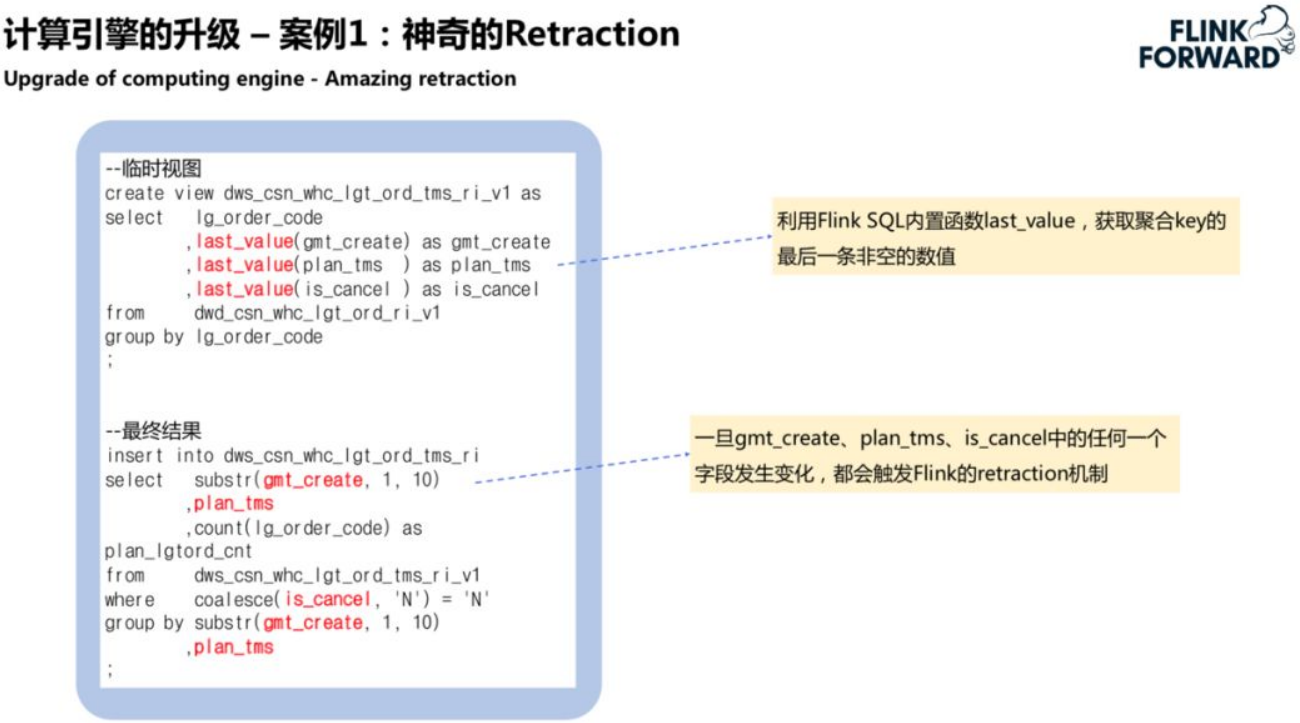
- 一个问题是针对表中 LP3 订单，在开始的时候是有效的（18 分的时候“是否取消”应该是 N，表写错），然而最后该订单却被取消了（最后一行“是否取消”应该是Y，表写错）,这种情况该订单被视为无效订单，统计的时候不应该考虑在内。
- 另外，配送公司的转变也需要注意，LP1 订单在 1 分钟的时候计划配送公司还是 tmsA，而之后计划配送公

司变成了 tmsB 和 tmsC，按照离线的计算方式（如 Storm 或增量）会得出右上角的结果，tmsA、tmsB 和 tmsC 与 LP1 订单相关的记录都会被统计，事实上 tmsA 和 tmsB 都未配送该订单，因此该结果实际上是错误的，正确的结果应该如图右下角表格所示。

针对该场景，Flink 内置提供了基于 state 的 Retraction 机制，可以帮助轻松实现流式消息的回撤统计。



下图展示了 Retraction 机制的伪代码实现。第一步是利用 Flink SQL 内置行数 last_value，获取聚合 key 的最后一条非空的数值，针对上述表中的 LP1 订单，使用 last_value 得到的结果是 tmsC，是符合预期的结果。需要强调的一点是，左侧使用 last_value 统计的字段 gmt_create、plan_tms、is_cancel，一旦其中的任何一个字段发生变化，都会发生出发 Flink 的 Retraction 机制。

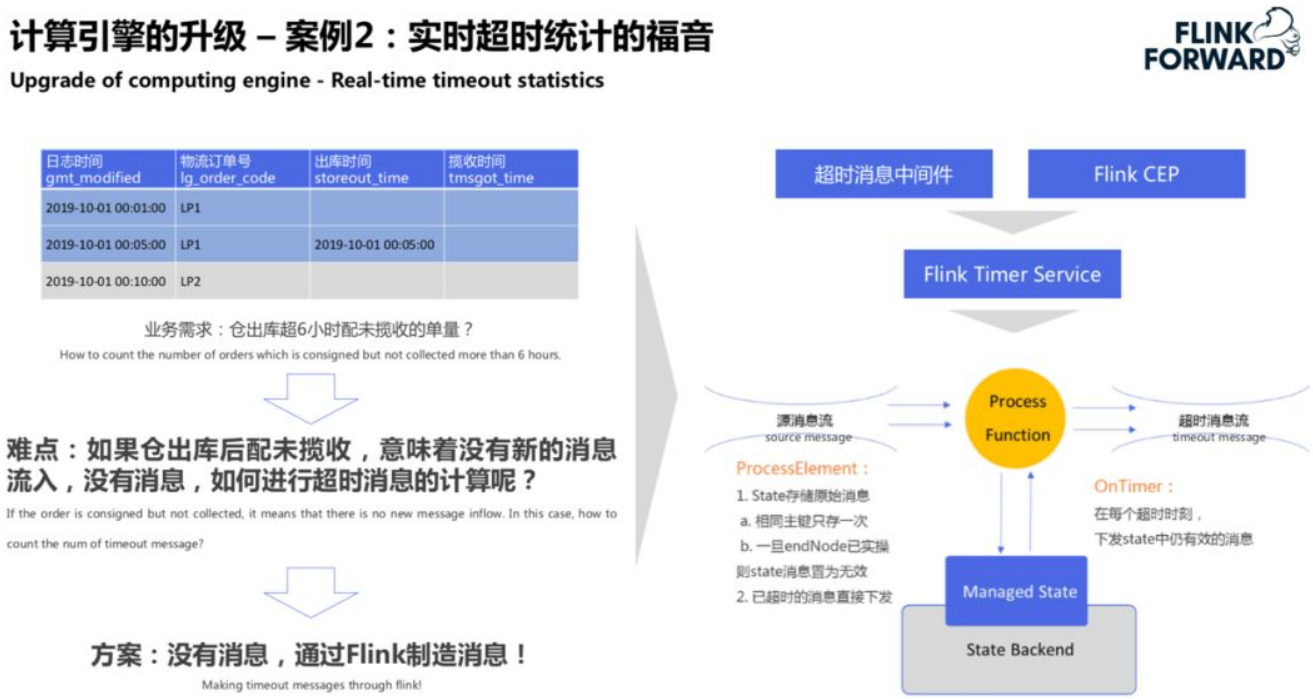


案例 2：超时统计

物流是菜鸟中比较常见的业务场景，物流业务中经常会有实时超时统计的需求，比如统计仓出库超过六个小时未被揽收的单量。

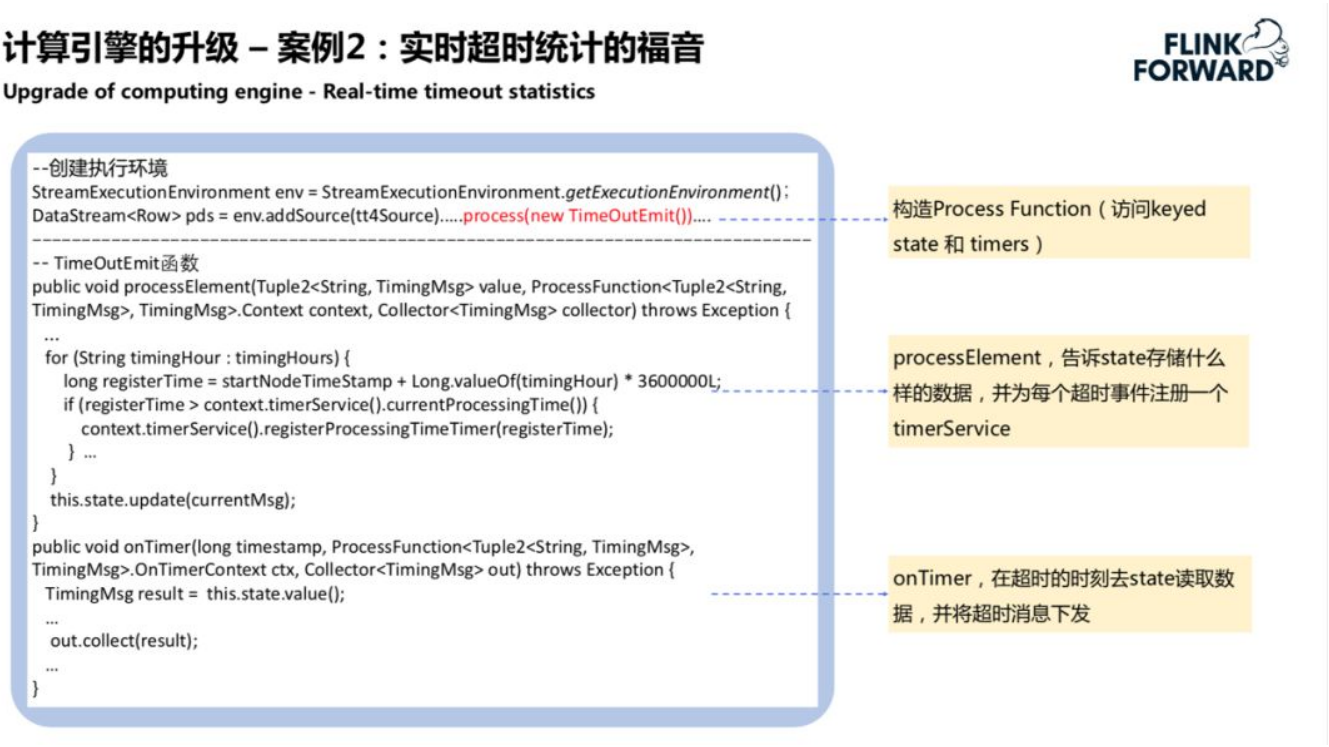
- 用到的数据表如下图左侧所示，其中包含日志时间、物流订单号、出库时间和揽收时间。该需求如果在离线的小时表或天表中比较好实现，但是在实时的场景下，其实现面临一定的挑战。
- 因为如果仓出库后未被揽收，意味着没有新的消息流入，如果没有消息就没有办法进行超时消息的计算。
- 为了解决该问题，菜鸟从 2017 年初就开始了一系列的探索，发现一些消息中间件（如 Kafka）和 Flink CEP 等本身会提供超时消息下发的功能，引入消息中间件的维护成本比较高，而 Flink CEP 的应用会出现回传不准确的问题。

针对上述需求，菜鸟选择了 Flink Timer Service 来进行实现。具体来讲，菜鸟对 Flink 底层的 ProcessFunction 中的 ProcessElement 函数进行了改写,该函数中，由 Flink 的 state 存储原始消息，相同的主键只存一次，一旦 endNode 已实操，则 state 消息置为无效，已超时的消息直接下发。此外，重写编写一个 OnTimer 函数，主要负责在每个超时的时刻读取 state 消息，然后下发 state 中仍然有效的消息，基于下游和正常游的关联操作便可以统计出超时消息的单量。



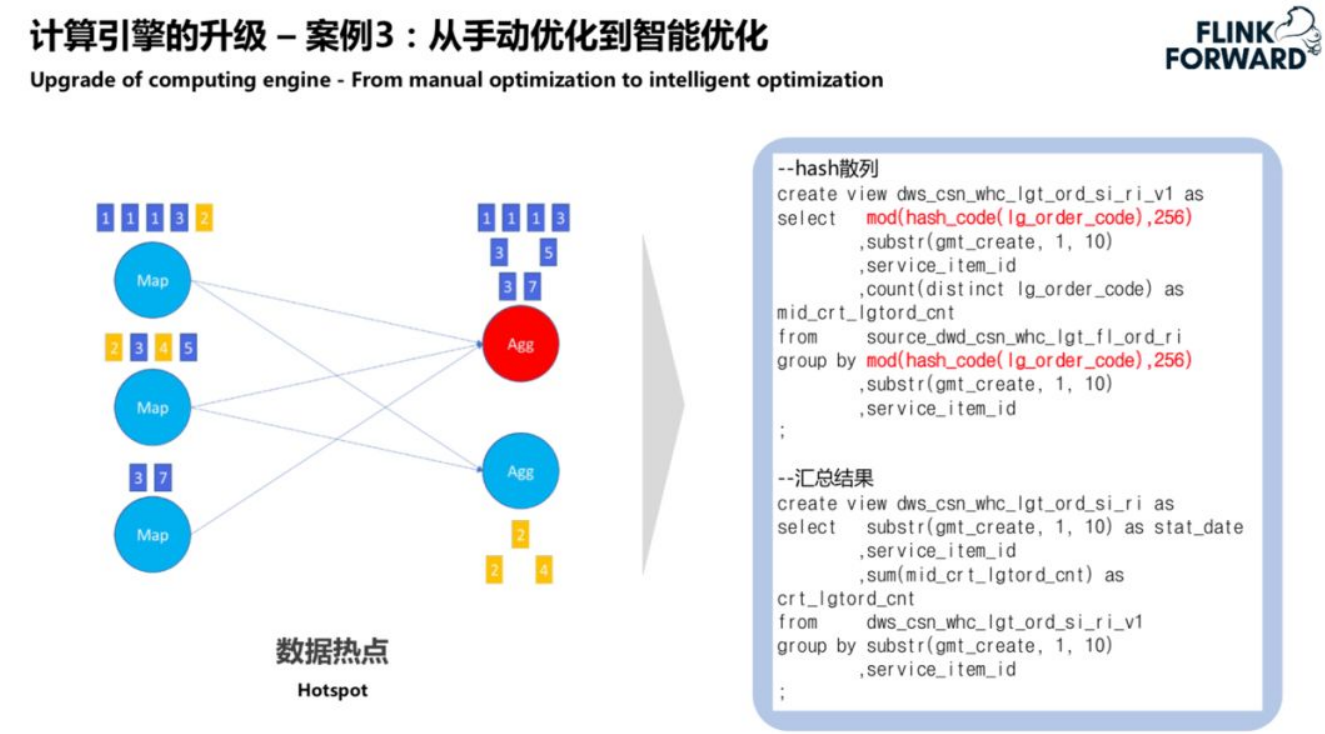
使用 Flink Timer Service 进行超时统计的伪代码实现如下图所示。

- 首先需要创建执行环境，构造 Process Function（访问 keyed state 和 times）；
- 其次是 processElement 函数的编写，主要用于告诉 state 存储什么样的数据，并为每个超时消息注册一个 timerService，代码中 timingHour 存储超时时间，比如前面的提到六小时，
- 然后启动 timerService；
- 最后是 onTimer 函数的编写，作用是在超时的时刻读取 state 的数据，并将超时消息下发。



案例 3：从手动优化到智能优化

实时数仓中会经常遇到数据热点和数据清洗的问题。下图左侧展示了数据热点的流程，蓝色部分 Map 阶段经过 Shuffle 后，转到红色部分 Agg，此时便会出现数据热点。针对该问题，菜鸟最初的解决方案的伪代码实现如下图右侧所示。假设对 lg_order-code 进行清洗，首先会对其进行 hash 散列操作，然后针对散列的结果进行二次聚合，这样便可以在一定程度上减轻倾斜度，因为可能会多一个 Agg 的操作。



菜鸟内部目前使用的 Flink 最新版本提供了解决数据热点问题的智能化特性：

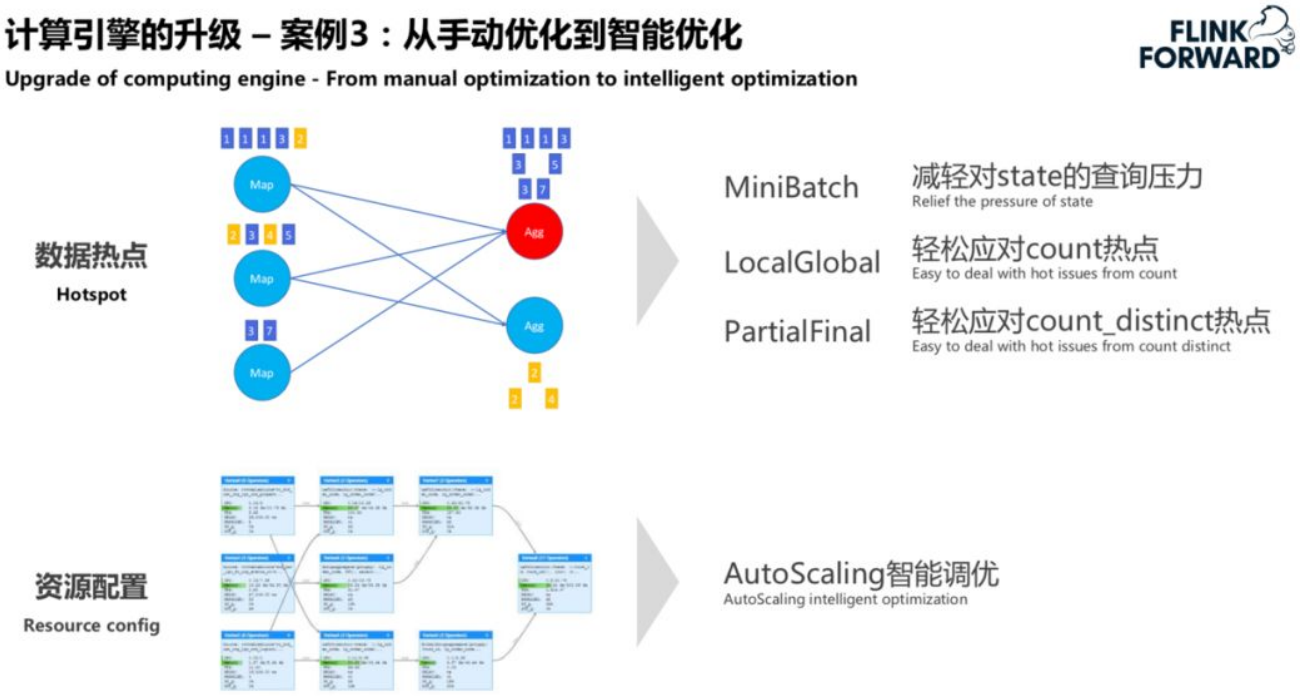
- **MiniBatch**。原来每进来一条数据，就需要去 state 中查询并写入，该功能可以将数据进行聚合后再写入 st

ate 或从 state 中读取，从而减轻对 state 的查询压力。

- **LocalGlobal**。类似于 Hive 中 Map 阶段的聚合，通过该参数可以实现数据读取阶段的聚合，轻松应对 count 热点。
- **PartialFinal**。面对更复杂的场景，比如 count_distinct 的热点，使用该参数可以轻松应对，实现两次聚合，类似于 Hive 中的两次 Reduce 操作。

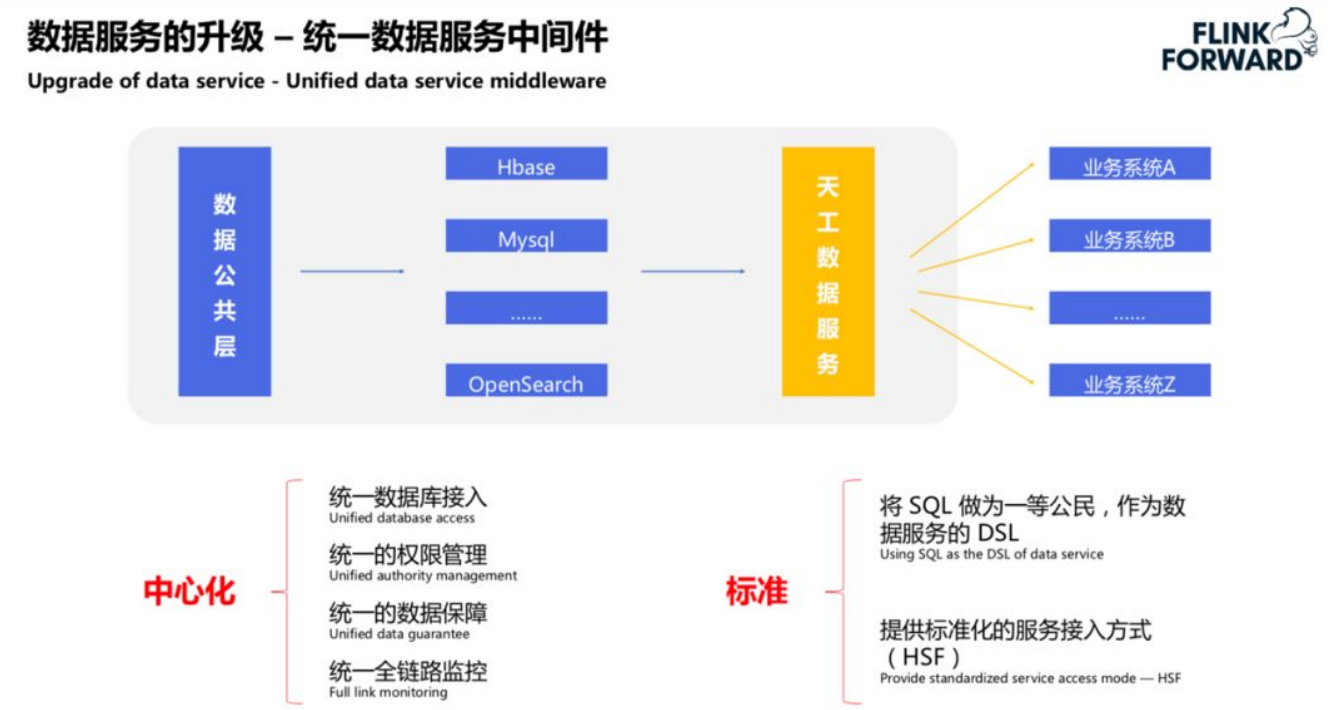
智能化功能支持的另一个场景是资源配置。在进行实时 ETL 过程中，首先要定义 DDL，然后编写 SQL，之后需要进行资源配置。针对资源配置问题，菜鸟之前的方案是对每一个节点进行配置，包括并发量、是否会涉及消息乱序操作、CPU、内存等，一方面配置过程非常复杂，另一方面无法提前预知某些节点的资源消耗量。Flink目前提供了较好的优化方案来解决该问题：

- **大促场景**：该场景下，菜鸟会提前预估该场景下的 QPS，会将其配置到作业中并重启。重启后 Flink 会自动进行压测，测试该 QPS 每个节点所需要的资源。
- **日常场景**：日常场景的 QPS 峰值可能远远小于大促场景，此时逐一配置 QPS 依然会很复杂。为此 Flink 提供了 AutoScaling 智能调优的功能，除了可以支持大促场景下提前设置 QPS 并压测获取所需资源，还可以根据上游下发的 QPS 的数据自动预估需要的资源。大大简化了资源配置的复杂度，使得开发人员可以更好地关注业务逻辑本身的开发。



数据服务升级

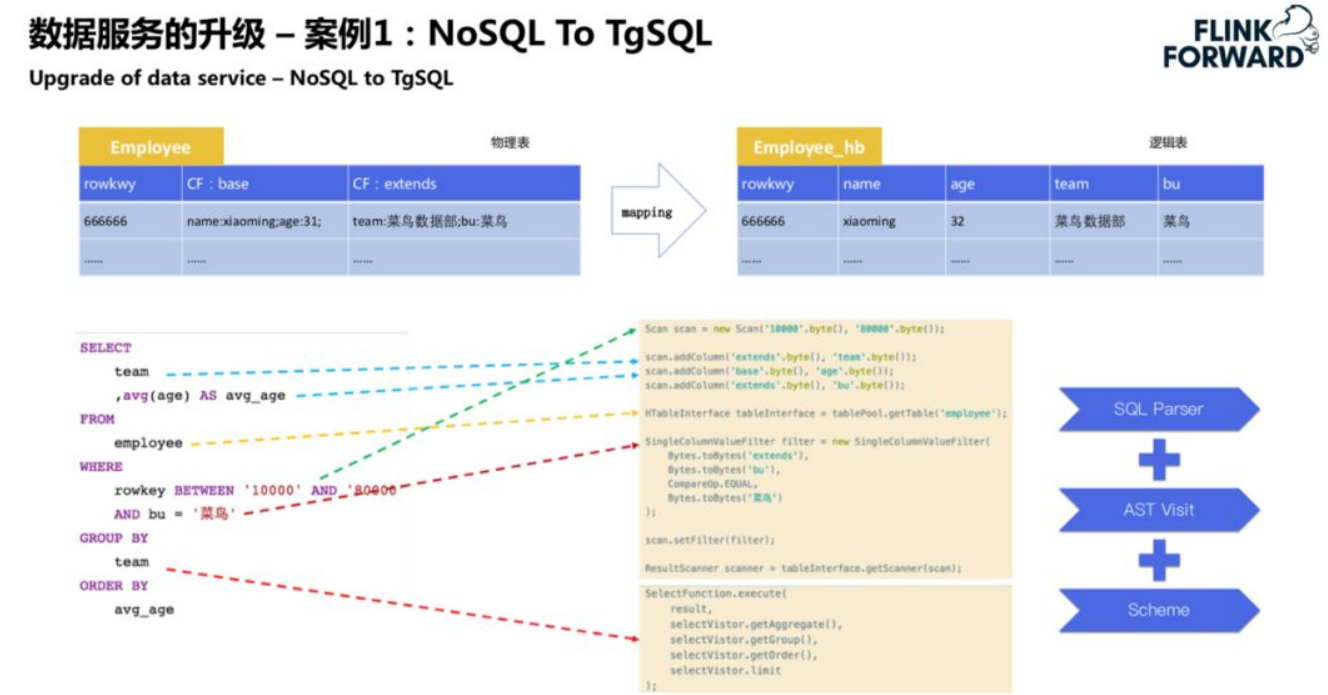
菜鸟在做数仓的过程中也会提供开发一系列的数据产品来提供数据服务，原来是采用 Java Web 提供多种连接 DB 的方式。但是实际应用过程中，经常用到的数据库无非是 Hbase、MySQL 和 OpenSearch 等，因此后来菜鸟联合数据服务团队建立了一个统一的数据服务中间件“天工数据服务”。它可以提供统一的数据库接入、统一的权限管理、统一的数据保障以及统一的全链路监控等中心化的功能，将 SQL 作为一等公民，作为数据服务的 DSL，提供标准化的服务接入方式（HSF）。



作为菜鸟数据服务的践行者，天工还提供了很多贴近业务的功能。接下来通过具体的案例场景来介绍。

案例 1：NoSQL to TgSQL

Hbase 等 NoSQL 类型的数据库，对于运营人员来讲编写代码是比较困难的，这种情况下其急需一套标准的语法。为了解决该问题，天工提供了 TgSQL，用于标准化 NoSQL 的转换。下图展示了转换的过程，Employee 转换成一个二维表，这里只是逻辑转换而非物理转换。天工中间件会解析 SQL，并在后台自动转换成查询的语言对数据进行查询。

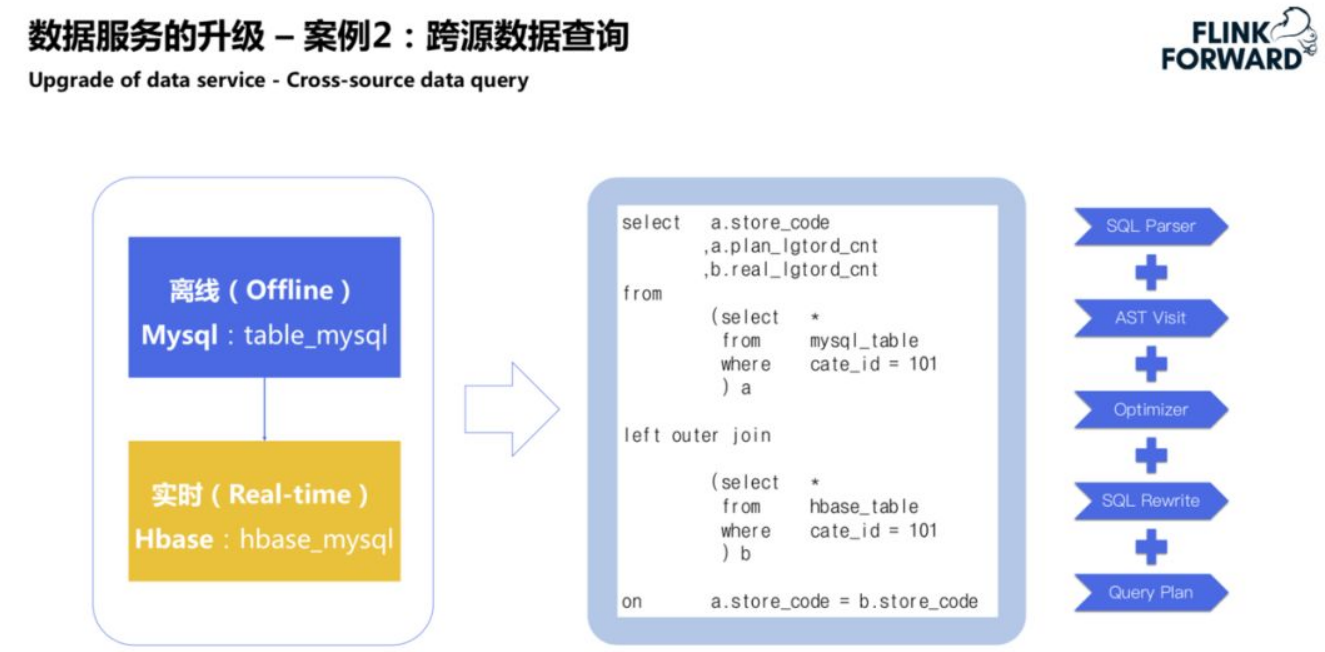


案例 2：跨源数据查询

菜鸟在开发数据产品的过程中，会经常遇到实时和离线分不开的情况。比如菜鸟每年都会统计 KPI 的实时完成率，计算方式是已经完成的单量与计划单量之间的比值，该计算依赖的数据源有两部分：

- 一部分是已经计划好的离线 KPI 表；
- 另一部分是已经计算好的写入 Hbase 的实时表。

原本的实现方案是通过 Java 取两次接口，然后在前端进行加减乘除的计算操作。针对该问题，菜鸟提供了标准的 SQL，用针对跨数据源的查询，如 MySQL 离线表和 Hbase 实时表，用户只需要按照标准 SQL 的方式来写，通过升级的数据服务进行解析，再从对应的数据库中进行数据的查询操作。



案例3：服务保障升级

菜鸟最初对于服务的保障比较缺失，一个任务发布后并不确定是否有问题，有些问题直到用户反馈的时候才能发现。另外，当并发量比较大的时候，也没有办法及时地做限流和主备切换等应对措施。

为此，天工的中间件提供了数据保障功能，除了主备切换，还包括主备双活、动态负载、热点服务阻断以及白名单限流等功能。

- 对于主备切换，前面提到的左右两侧分别是物理表和逻辑表的场景中，一个逻辑表可以映射成主备链路，当主链路出现问题时，可以一键切换到备链上；
- 此外，大促期间一些非常重要的业务，如大屏业务、内部统计分析等，会通过主备链路同时进行操作，此时完全读写其中一个库不合适，所期望的两条链路均有流量，而天工则实现了主备双活的功能支持，即将大流量切到主链，小流量切到备链；
- 当主链上受到其中一个任务影响时，该任务会被移到备链上；对于比较复杂、执行较慢的查询，会对整个任务的性能造成影响，此时会对这种类型的热点服务进行阻断。

其他技术工具的探索和创新

除了数据模型、计算引擎和数据服务，菜鸟还在其他方面进行了探索和创新，包括实时压测、过程监控、资源管理和数据质量等。

实时压测在大促期间比较常用，通过实时压测来模拟大促期间的流量，测试特定的 QPS 下任务是否可以成功执行。原本的做法是重启备链上的作业，然后将备链作业的 source 改为压测的 source，sink 改为压测 source 的动作，这种方案在任务特别多的时候实现起来非常复杂。为此，阿里云团队开发了实时压测的工具，可以做到一次启动所有的需要的压测的作业，并自动生成压测的 source 和 sink，执行自动压测，生成压测报告。

采用 Flink 后，还实现了作业过程监控的功能，包括延迟监控和告警监控，比如超过特定的时间无响应会进行告警，TPS、资源预警等。

菜鸟实时数仓未来发展与思考

菜鸟目前在实时数仓方面更多的是基于 Flink 进行一系列功能的开发，未来的发展方向计划向批流混合以及 AI 方向演进。

（1）Flink 提供了 batch 的功能后，菜鸟很多中小型的表分析不再导入到 Hbase 中， 而是在定义 source 的时候直接将 MaxCompute 的离线维表读到内存中，直接去做关联，如此一来很多操作不需要再进行数据同步的工作。

（2）针对一些物流的场景，如果链路比较长，尤其是双十一支付的订单，在十一月十七号可能还存在未签收的情况，这时候如果发现作业中有一个错误，如果重启的话，作业的 state 将会丢失，再加之整个上游的 source 在 TT 中只允许保存三天，使得该问题的解决变得更加困难。

- 菜鸟之后发现 Flink 提供的 batch 功能可以很好地解决该问题，具体来讲是定义 TT 的 source，作为三天的实时场景的应用，TT 数据写到离线数据库进行历史数据备份，如果存在重启的情况，会读取并整合离线的数据，即使 Flink 的 state 丢失，因为离线数据的加入，也会生成新的 state，从而不必担心双十一的订单如果在十七号签收之前重启导致无法获取十一号的订单信息。
- 当然，在上述问题的解决上，菜鸟也踩了很多的小坑。其中的一个是整合实时数据和离线数据的时候，数据乱序的问题。菜鸟实现了一系列的 UDF 来应对该问题，比如实时数据和离线数据的读取优先级设置。

（3）针对日志型的业务场景，比如曝光、网站流量等，其一条日志下来后，基本不会再发生变化。菜鸟目前在考虑将所有解析的工作交给 Flink 来处理，然后再写入到 batch 中，从而无需在 MaxCompute 的 ODPS 中进行批处理的操作。

（4）在智能化方面，前面提到的数据倾斜隐患的规避、资源的优化等，都用到了 Flink 提供的智能化功能。

- 菜鸟也期望在实时 ETL 过程中的一些场景中，比如去重，也使用 Flink 相应的智能化解决方案来进行优化

。

- 此外，在数据服务保障上，如主备切换等，目前仍然依赖人工对数据库进行监控，菜鸟也期望 Flink 之后能提供全链路实时保障的策略。
- 最后是业务场景的智能化，阿里 Alink 对于业务智能化的支持也是之后探索的方向。

菜鸟实时数仓 – 未来发展与思考

Upgrade of data service - Future development and thinking

FLINK FORWARD