Apache Flink 在唯品会的实践

本文来自于王新春在2018年7月29日 Flink China社区线下 Meetup·上海站的分享。王新春目前在唯品会负责实时平台相关内容，主要包括实时计算框架和提供实时基础数据，以及机器学习平台的工作。之前在美团点评，也是负责大数据平台工作。他已经在大数据实时处理方向积累了丰富的工作经验。。

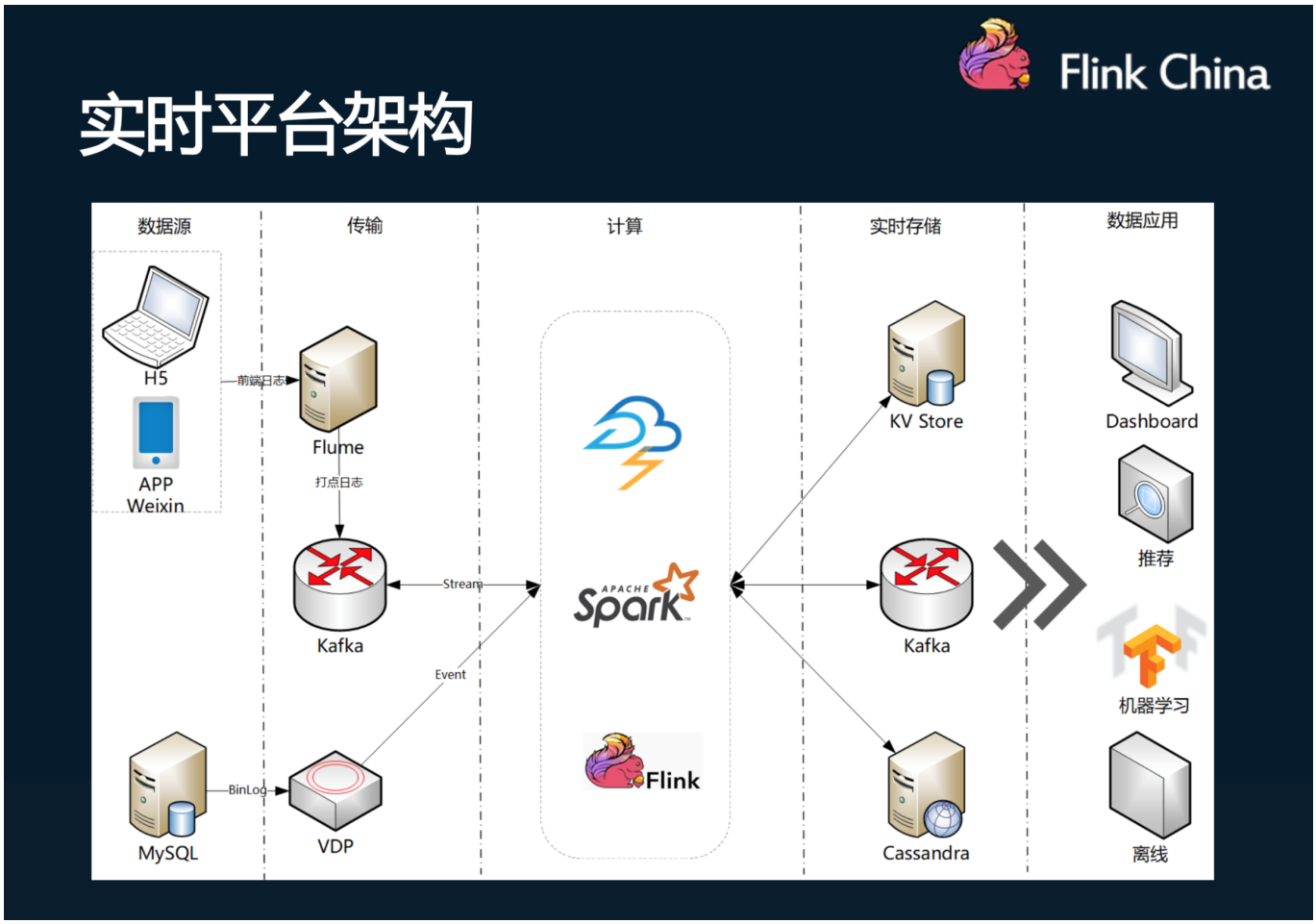
本文主要内容如下：

- 唯品会实时平台现状
- Flink在唯品会的实践
- Flink On K8S
- 后续规划

唯品会实时平台现状

目前在唯品会实时平台并不是一个统一的计算框架，而是包括Storm，Spark，Flink在内的三个主要计算框架。由于历史原因，当前在Storm平台上的job数量是最多的，但是从去年开始，业务重心逐渐切换到Flink上面，所以今年在Flink上面的应用数量有了大幅增加。

实时平台的核心业务包含八大部分：实时推荐作为电商的重点业务，包含多个实时特征；大促看板，包含各种维度的统计指标（例如：各种维度的订单、UV、转化率、漏斗等），供领导层、运营、产品决策使用；实时数据清洗，从用户埋点收集来数据，进行实时清洗和关联，为下游的各个业务提供更好的数据；此外还有互联网金融、安全风控、与友商比价等业务，以及Logview、Mercury、Titan作为内部服务的监控系统、VDRC实时数据同步系统等。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

实时平台的职责主要包括实时计算平台和实时基础数据。实时计算平台在Storm、Spark、Flink等计算框架的基础上，为监控、稳定性提供了保障，为业务开发提供了数据的输入与输出。实时基础数据包含对上游埋点的定义和规范化，对用户行为数据、MySQL的Binlog日志等数据进行清洗、打宽等处理，为下游提供质量保证的数据。

在架构设计上，包括两大数据源。一种是在App、微信、H5等应用上的埋点数据，原始数据收集后发送到在kafka中；另一种是线上实时数据的MySQL Binlog日志。数据在计算框架里面做清洗关联，把原始的数据通过实时ETL为下游的业务应用（包括离线宽表等）提供更易于使用的数据。

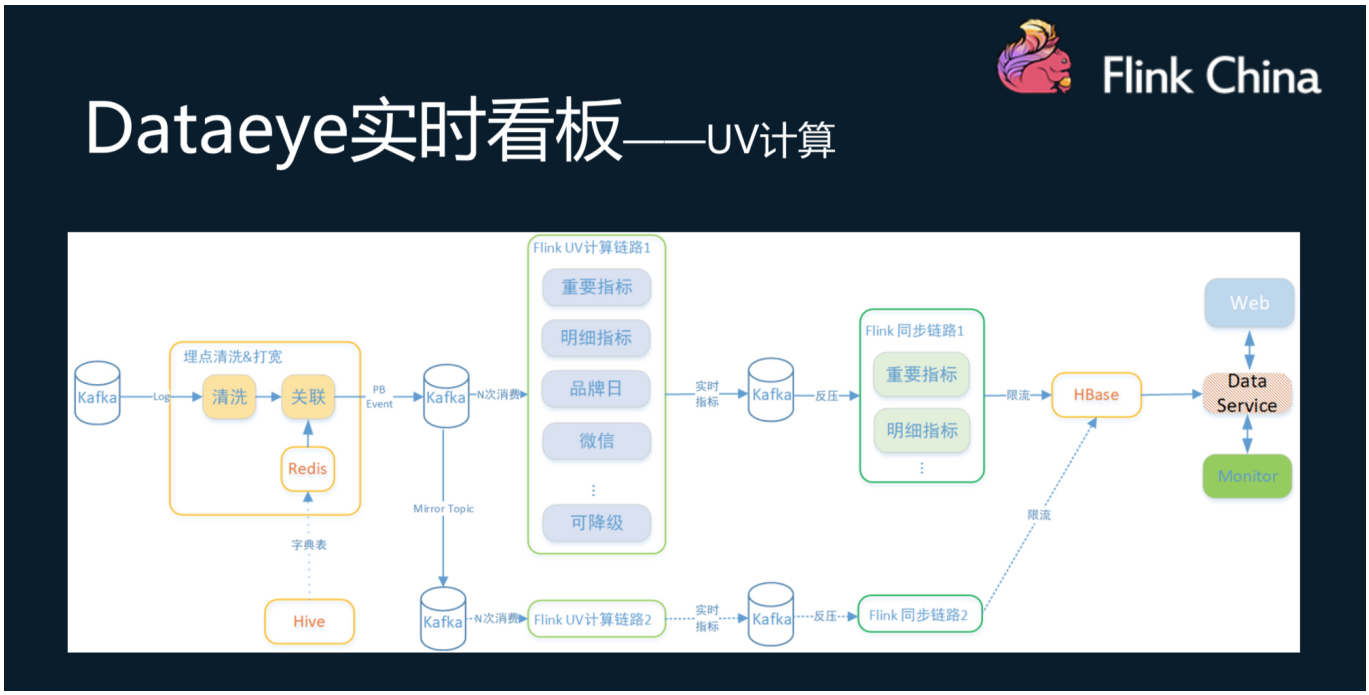
如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

Flink在唯品会的实践

场景一：Dataeye实时看板

Dataeye实时看板是支持需要对所有的埋点数据、订单数据等进行实时计算时，具有数据量大的特点，并且需要统计的维度有很多，例如全站、二级平台、部类、档期、人群、活动、时间维度等，提高了计算的复杂程度，统计的数据输出指标每秒钟可以达到几十万。

以UV计算为例，首先对Kafka内的埋点数据进行清洗，然后与Redis数据进行关联，关联好的数据写入Kafka中；后续Flink计算任务消费Kafka的关联数据。通常任务的计算结果的量也很大（由于计算维度和指标特别多，可以达到上千万），数据输出通过也是通过Kafka作为缓冲，最终使用同步任务同步到HBase中，作为实时数据展示。同步任务会对写入HBase的数据限流和同类型的指标合并，保护HBase。与此同时还有另一路计算方案作为容灾。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

在以Storm进行计算引擎中进行计算时，需要使用Redis作为中间状态的存储，而切换到Flink后，Flink自身具备状态存储，节省了存储空间；由于不需要访问Redis，也提升了性能，整体资源消耗降低到了原来的1/3。

在将计算任务从Storm逐步迁移到Flink的过程中，对两路方案先后进行迁移，同时将计算任务和同步任务分离，缓解了数据写入HBase的压力。

切换到Flink后也需要对一些问题进行追踪和改进。对于FlinkKafkaConsumer，由于业务原因对kafka中的Aotu Commit进行修改，以及对offset的设定，需要自己实现支持kafka集群切换的功能。对不带window的state数据需要手动清理。还有计算框架的通病——数据倾斜问题需要处理。同时对于同步任务追数问题，Storm可以从Redis中取值，Flink只能等待。

场景二：Kafka数据落地HDFS

之前都是通过Spark Streaming的方式去实现，现在正在逐步切换到Flink上面，通过OrcBucketingTableSink将埋点数据落地到HDFS上的Hive表中。在Flink处理中单Task Write可达到3.5K/s左右，使用Flink后资源消耗降低了90%，同时将延迟30s降低到了3s以内。目前还在做Flink对Spark Bucket Table的支持。

场景三：实时的ETL

对于ETL处理工作而言，存在的一个痛点就是字典表存储在HDFS中，并且是不断变化的，而实时的数据流需要与字典表进行join。字典表的变化是由离线批处理任务引起的，目前的做法是使用ContinuousFileMonitoringFunction和ContinuousFileReaderOperator定时监听HDFS数据变化，不

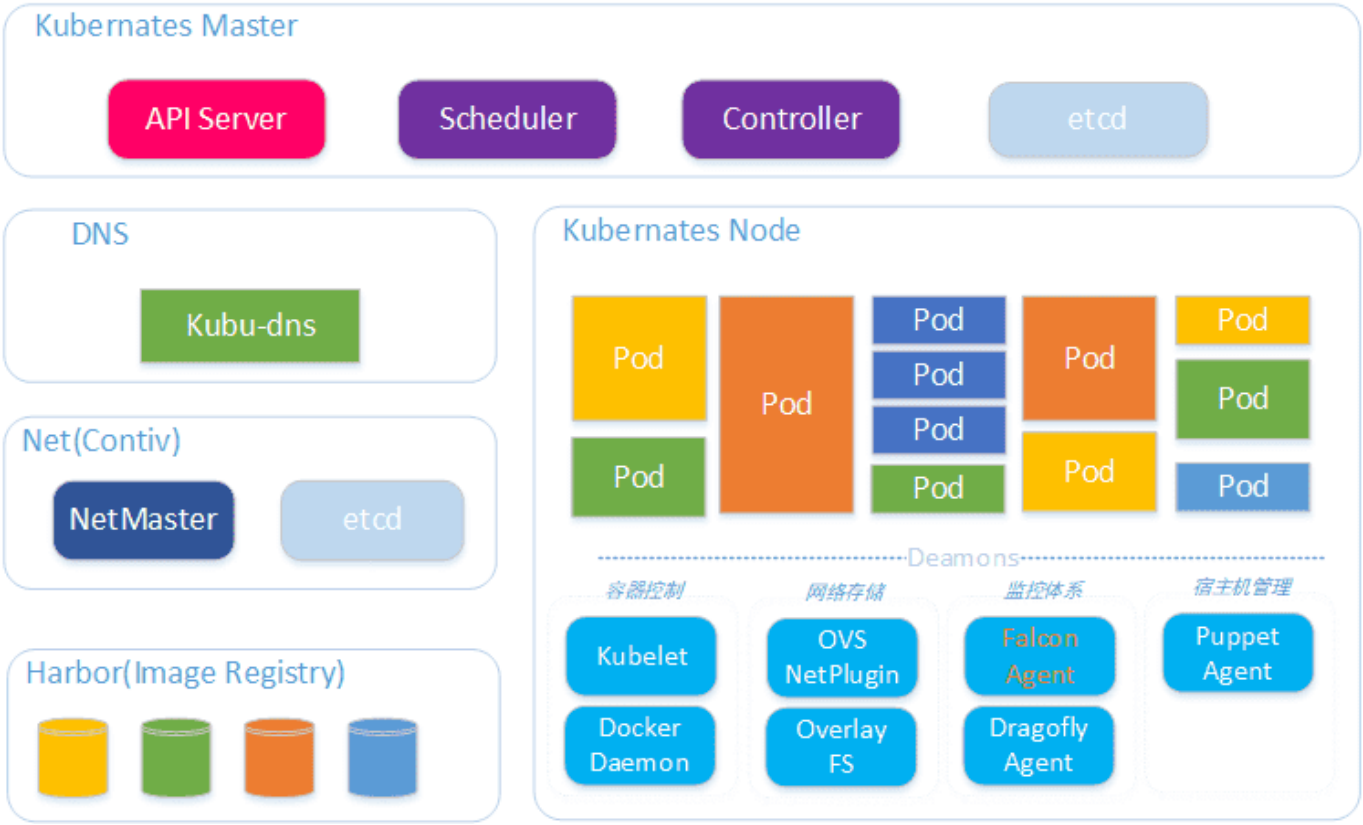
断地将新数据刷入，使用最新的数据去做join实时数据。

我们计划做更加通用的方式，去支持Hive表和Stream的join，实现Hive表数据变化之后，数据自动推送的效果。

Flink On K8S

在唯品会内部有一些不同的计算框架，有实时计算的，有机器学习的，还有离线计算的，所以需要一个统一的底层框架来进行管理，因此将Flink迁移到了K8S上。

在K8S上使用了思科的网络组件，每个docker容器都有独立的ip，对外也是可见的。实时平台的融合器整体架构如下图所示。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

唯品会在K8S上的实现方案与Flink社区提供的方案差异还是很大的。唯品会使用K8S StatefulSet模式部署，内部实现了cluster相关的一些接口。一个job对应一个mini cluster，并且支持HA。对于Flink来说，使用StatefulSet的最大的原因是pod的hostname是有序的；这样潜在的好处有：

- hostname为-0和-1的pod可以直接指定为jobmanager；可以使用一个statefulset启动一个cluster，而deployment必须2个；Jobmanager和TaskManager分别独立的deployment。
- pod由于各种原因fail后，由于StatefulSet重新拉起的pod的hostname不变，集群recover

的速度理论上可以比deployment更快（deployment每次主机名随机）。

镜像的docker entrypoint脚本里面需要设置的环境变量设置说明：

环境变量名称	参数	示例内容	说明
JOB_MANGER_HOSTS	StatefulSet.name-0,StatefulSet.name-1	flink-cluster-0,flink-cluster-1	JM的主机名，短主机名；可以不用FQDN
FLINK_CLUSTER_IDENT	namespace/StatefulSet.name	default/flink-cluster	用来做zk ha设置和hdfs checkpiont的根目录
TASK_MANAGER_NUMBER_OF_TASK_SLOTS	containers.resources.cpu.limits	2	TM的slot数量，根据resources.cpu.limits来设置
FLINK_ZK_QUORUM	env:FLINK_ZK_QUORUM	10.198.199.112:2181	HA ZK的地址
JOB_MANAGER_HEAP_MB	env:JOB_MANAGER_HEAP_MB value:containers.resources.memory.limit -1024	4096	JM的Heap大小，由于存在堆外内存，需要小于container.resources.memory.limits；否则容易OOM kill
TASK_MANAGER_HEAP_MB	env:TASK_MANAGER_HEAP_MB value: containers.resources.memory.limit -1024	4096	TM的Heap大小，由于存在Netty的堆外内存，需要小于container.resources.memory.limits；否则容易OOM kill

对应Flink集群所依赖的HDFS等其他配置，则通过创建configmap来管理和维护。

kubectl create configmap hdfs-conf --from-file=hdfs-site.xml --from-file=core-site.xml

后续计划

当前实时系统，机器学习平台要处理的数据分布在各种数据存储组件中，如Kafka、Redis、Tair和HDFS等，如何方便高效的访问，处理，共享这些数据是一个很大的挑战，对于当前的数据访问和解析常常需要耗费很多的精力，主要的痛点包括：

对于Kafka，Redis，Tair中的binary（PB/Avro等格式）数据，使用者无法快速直接的了解数据的schema与数据内容，采集数据内容及与写入者的沟通成本很高。

由于缺少独立的统一数据系统服务，对Kafka，Redis，Tair等中的binary数据访问需要依赖写入者提供的信息，如proto生成类，数据格式wiki定义等，维护成本高，容易出错。

缺乏relational schema使得使用者无法直接基于更高效易用的SQL或LINQ层API开发业务。

无法通过一个独立的服务方便的发布和共享数据。

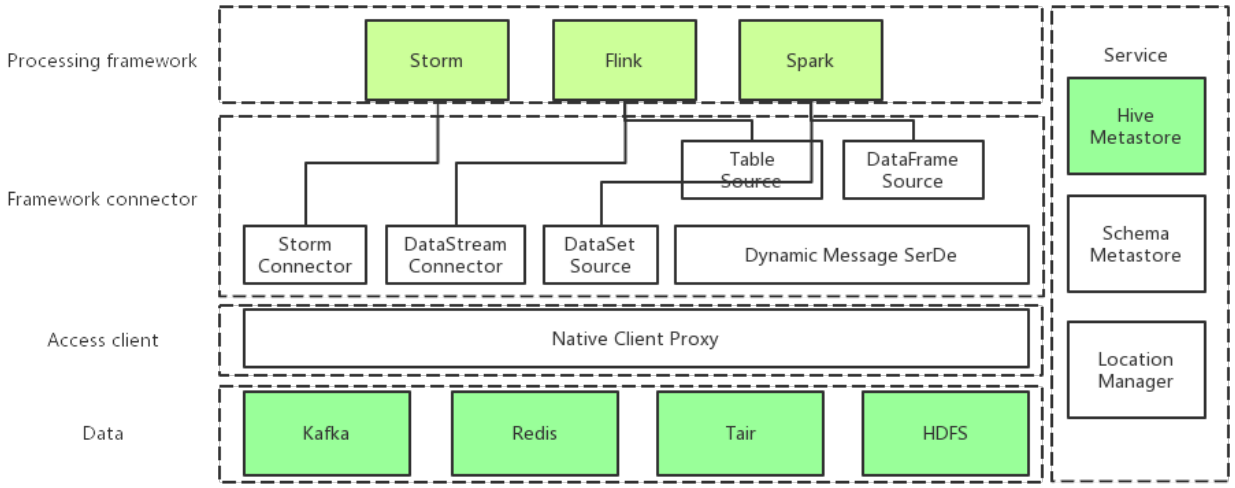
实时数据无法直接提供给Batch SQL引擎使用。

此外，对于当前大部分的数据源的访问也缺少审计，权限管理，访问监控，跟踪等特性。

UDM(统一数据管理系统)包括Location Manager, Schema Metastore以及Client Proxy等模块，主要的功能包括：

- 提供从名字到地址的映射服务，使用者通过抽象名字而不是具体地址访问数据。
- 用户可以方便的通过Web GUI界面方便的查看数据Schema，探查数据内容。
- 提供支持审计，监控，溯源等附加功能的Client API Proxy。
- 在Spark/Flink/Storm等框架中，以最适合使用的形式提供这些数据源的封装。

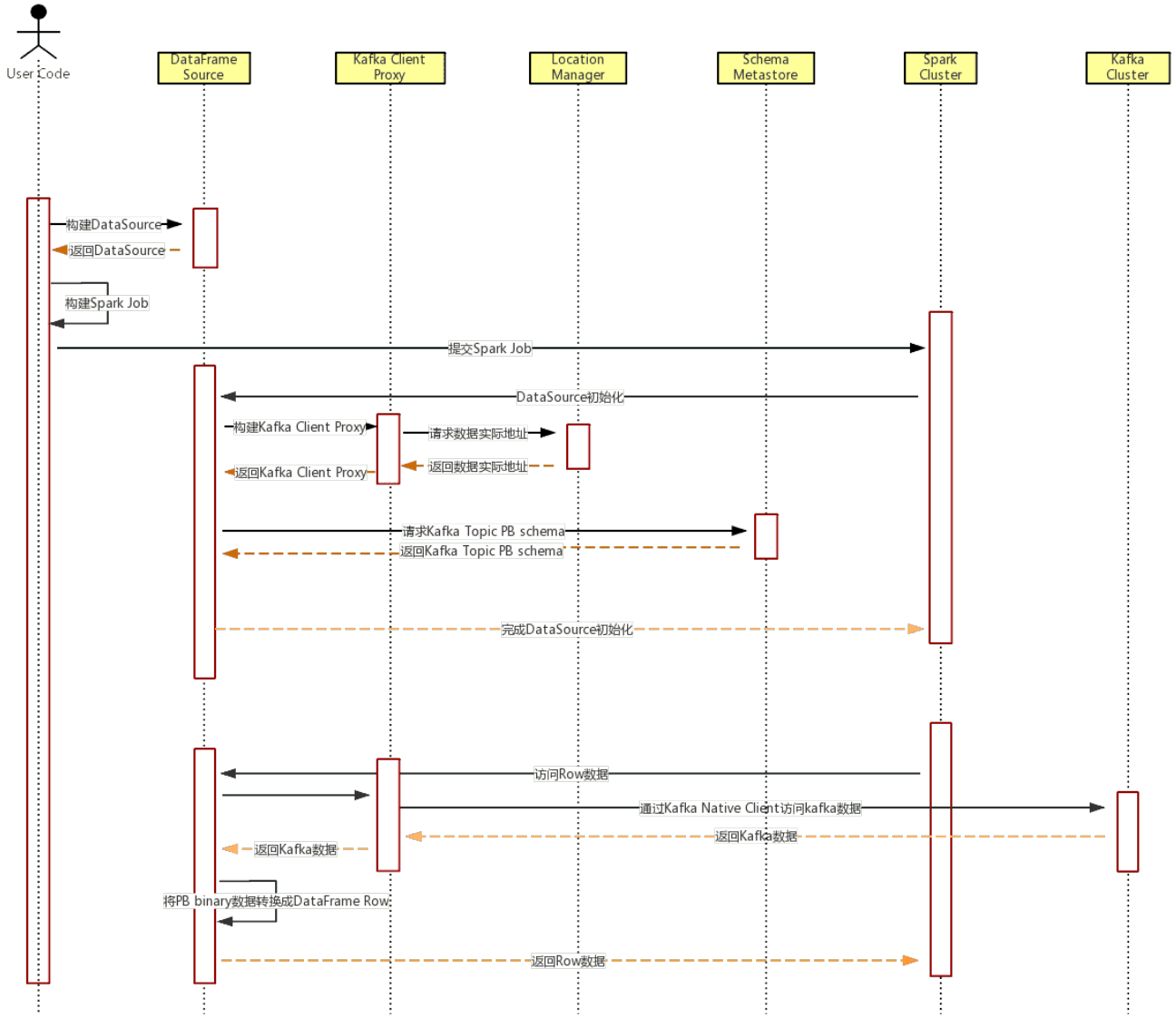
UDM的整体架构如下图所示：



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

UDM的使用者包括实时，机器学习以及离线平台中数据的生产者和使用者。在使用Sql API或Table API的时候，首先完成Schema的注册，之后使用Sql进行开发，降低了开发代码量。

以Spark访问Kafka PB数据的时序图来说明UDM的内部流程：



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

在Flink中，使用UDMExternalCatalog来打通Flink计算框架和UDM之间的桥梁，通过实现External Catalog的各个接口，以及实现各自数据源的TableSourceFactory，完成Schema和接入管控等各项功能。

本文 PPT 下载

本文的 PPT 可以到 《Flink China社区线下 Meetup·上海站 PPT 资料分享》 里面进行下载。