法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

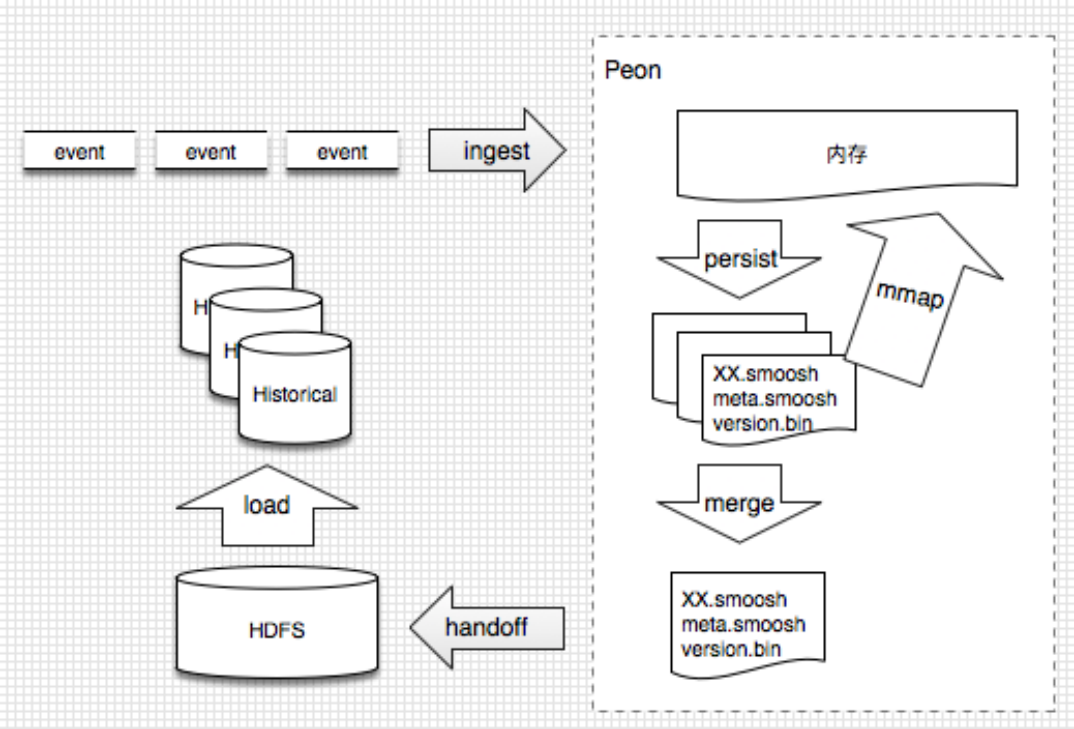
Druid数据存储与写入

目录

- 数据存储格式
- 实时写入方式
- 离线写入方式

数据存储

- OnheapIncrementalIndex 实时聚合写入时间
- OffheapIncrementalIndex 负责实时节点本地索引文件的加载
- Segment 最终持久化索引



Druid基本概念与特性-Segment

Segment

- Druid将索引数据保存到Segment文件中,Segment文件根据时间进行分片。Segment中会保存维度、指标以及索引信息。
- 将行式数据转换为列式存储结构: 按需加载，提高查询速度
- 有三种类型的数据结构:
  - timestamp列, long数组
  - 指标列: int数组或float数组
  - 维度列: 支持过滤和分组。使用压缩的BitMap索引。

Druid基本概念与特性-Segment

1. 字典，将列的所有值进行编码

```
{
  “北京”：0,
  “上海”：1
}
```

2. 列的数据, 要保存的是每一行中这一列的值, 使用字典中的编码

[0, 1, 0 ，0 ，1]

3. Bitmaps – 对于列的每一个不同的值都有

“北京”：[1, 0, 1, 1, 0]

“上海”：[0, 1, 0, 0, 1]

在最坏情况下前面两种会随着数据量的大小而线性增长.

而BitMap的大小则等于数据量大小 * 列的个数.

Druid基本概念与特性-Segment

Segment文件有下面几个文件组成

0000.smoosh factory.json meta.smoosh version.bin

其中.smoosh文件可能有多个，以编号命名。

.smooh文件有多个文件组成，meta.smoosh记录元数据，用于切分文件

```
v1,2147483647,1 # 版本v1, maxChunkSize, numChunks;首行一定是3个字段，后面每一行都是4个字段
__time,0,0,421  # 列名, fileNum, startOffset, endOffset
area,0,8425,33571
busynum,0,421,2898
drivernum,0,2898,5375
idlenum,0,5375,7921
index.drd,0,33571,33736
metadata.drd,0,33736,34151
product_id,0,7921,8425
```

Druid基本概念与特性-Segment

__time 0-421	busynum 421-2898	drivernum 2898-5375	idlenum 5375-7921	product_id 7921-8425	area 8425-33571	index.drd 33571-33736	metadata.drd- 33736-34151

index.drd文件包含：

segment version，columns，dimensions，interval start，interval end，bitmap

metadata.drd文件包含：

Aggregators，timestampSpec，queryGranularity，rollup，container

数据摄入

- 流式数据源与静态数据源
- 实时数据流摄入
  - Standalone Realtime Node(Streaming pull)
  - Indexing-service + Tranquility(Streaming push)
  - KafkaIndex-indexing-service
- 离线数据摄入
  - 以索引服务方式摄入
  - 以MR任务方式摄入

数据摄入

流式数据源

指持续不断地产生数据的数据源。如消息队列、日志文件等。

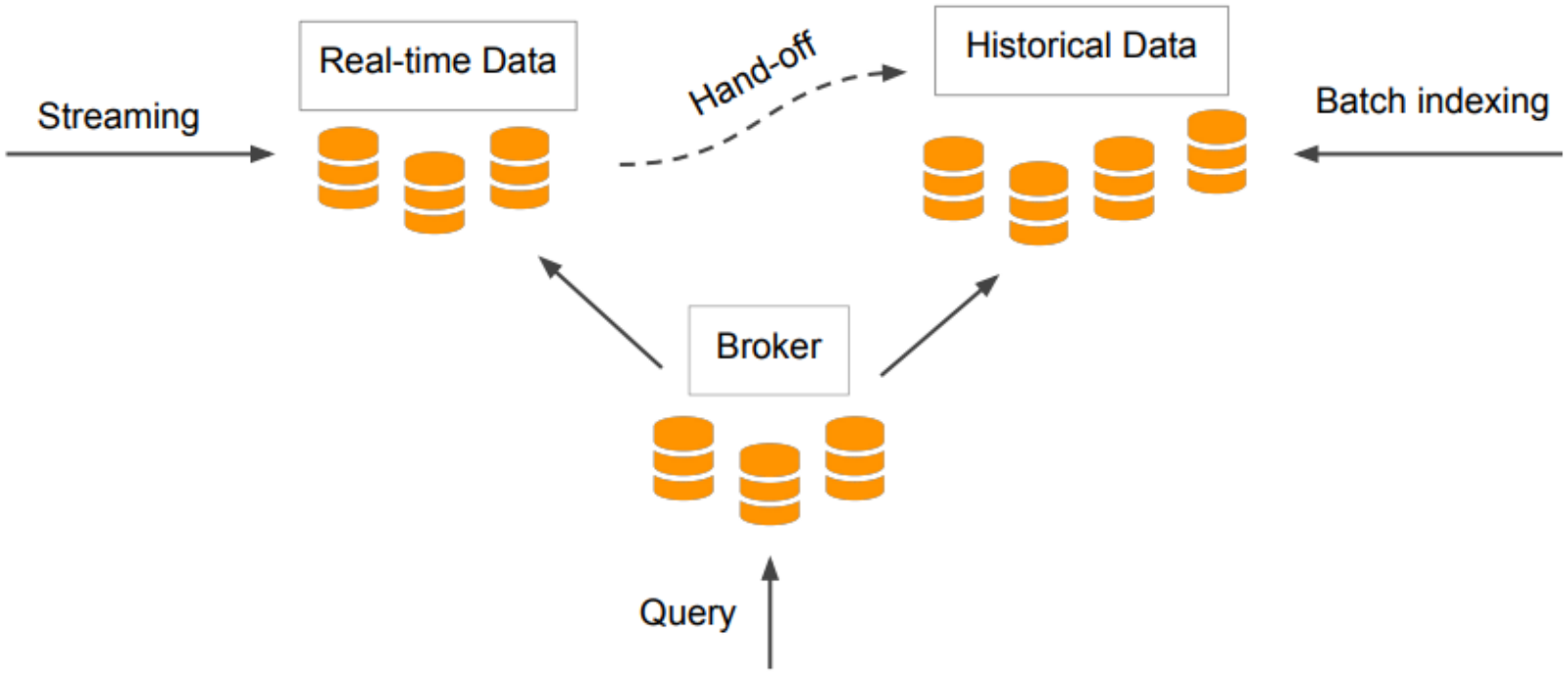
需求：流式导入，一边产生数据，一边倒入。

静态数据源

已经产生完毕、不会有新数据产生的数据源。如文件系统中的文件等。

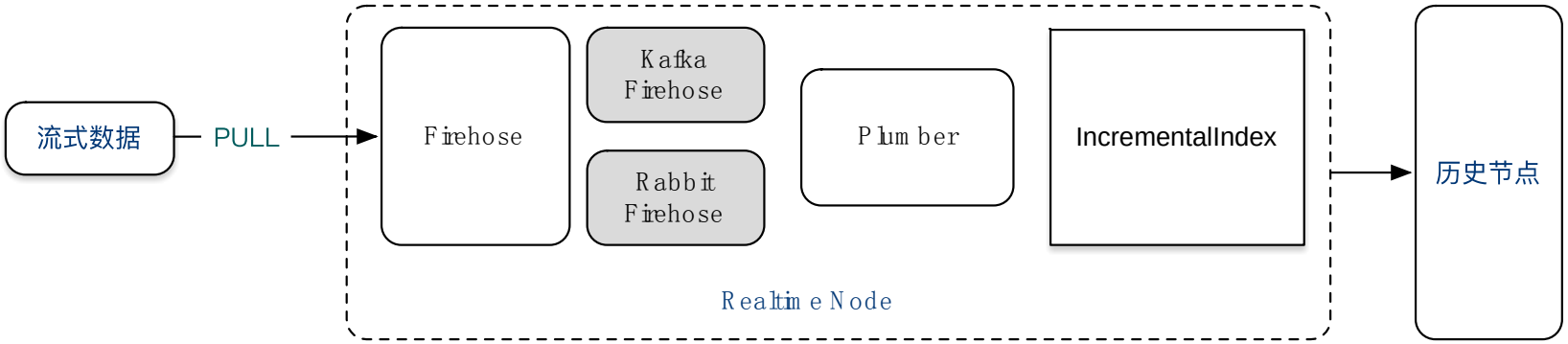
需求：批量导入，周期性的定时导入。

数据摄入


Streaming
Real-time Data
Hand-off
Historical Data
Batch indexing
Broker
Query

数据摄入-Standalone Realtime Node

- Firehose为数据源头的抽象，适配多种上游数据源
- Plumber为从数据源头取数构建增量索引的抽象



数据摄入-Standalone Realtime Node

KafkaFirehose ioConfig

```
"ioConfig": {
    "type": "realtime",
    "firehose": {
        "type": "kafka-0.8",
        "consumerProps": {
            "zookeeper.connect": "localhost:2181",
            "zookeeper.connection.timeout.ms": "15000",
            "zookeeper.session.timeout.ms": "15000",
            "zookeeper.sync.time.ms": "5000",
            "group.id": "druid-example",
            "fetch.message.max.bytes": "1048586",
            "auto.offset.reset": "largest",
            "auto.commit.enable": "false"
        },
        "feed": "wikipedia"
    },
    "plumber": {
        "type": "realtime"
    }
}
```

数据摄入-Standalone Realtime Node

缺点

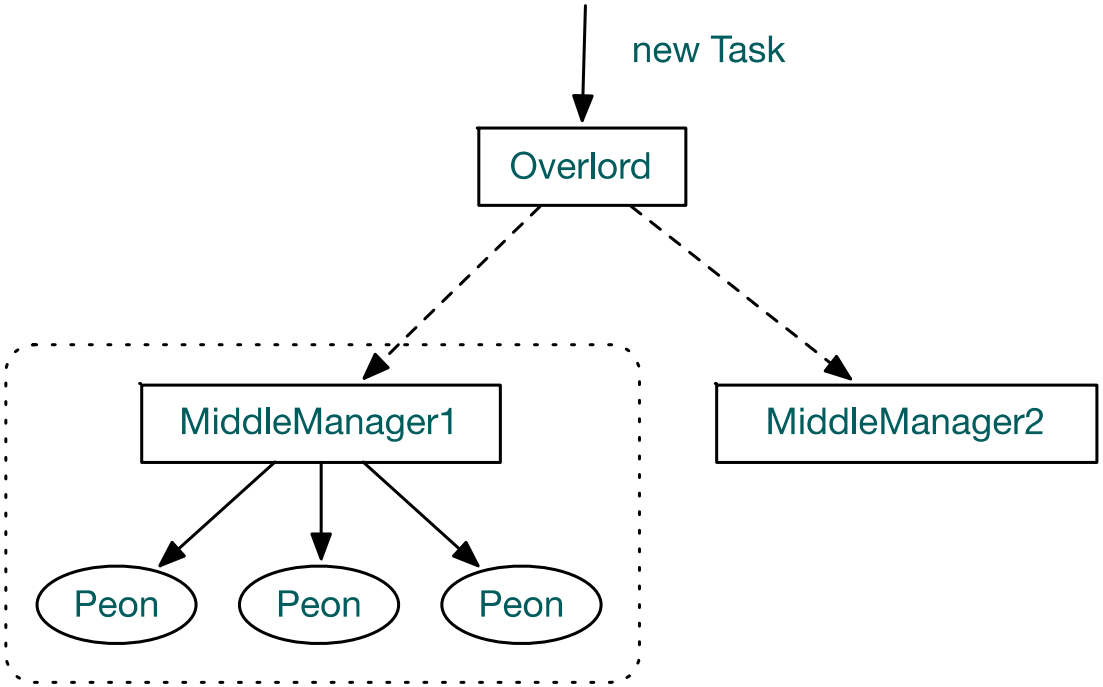
•数据消费任务为单机模式，任务失败后无法恢复

•KafkaFirehose使用Kafka高阶API，多任务消费数据时，难以保证副本任务消费相同的数据

•数据schema变更，需要重启实时节点

数据摄入-Indexing-service

- Overlord接受任务，并按照一定策略分配任务给MiddleManager
- MiddleManager领取任务并通过子进程的方式启动Peon进程（实时任务/实时节点）



数据摄入-Indexing-service

Indexing Service

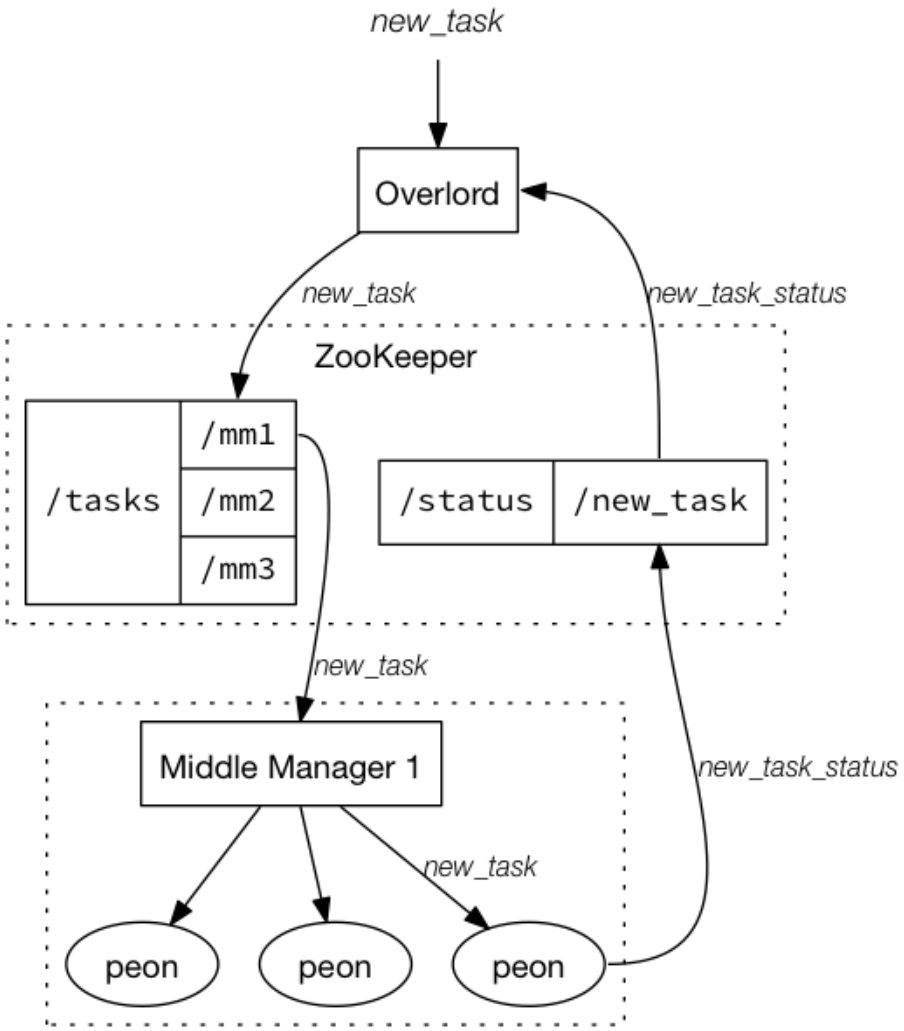
任务管理

•Overlord接受任务

•通过zk将任务信息分配给MM

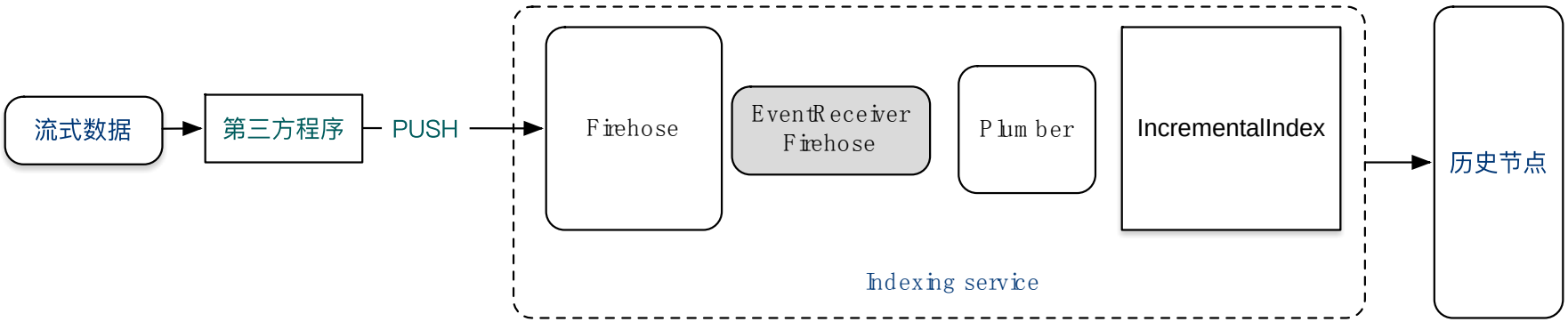
•MM领取任务，创建Peon进程

•通过zk发布任务状态



数据摄入-Indexing-service

- 基于Http API启动实时任务/实时节点，更加灵活
- 使用EventReceiver Firehose，来解决副本任务消费数据一致性的问题



数据摄入-Indexing-service

- ClippedFirehoseFactory 创建指定时间interval的Firehose
- TimedShutoffFirehoseFactory 创建在指定时间关闭的Firehose
- EventReceiverFirehoseFactory 创建提供HTTP接口接受数据的Firehose

```
"firehose": {
    "type": "clipped",
    "delegate": {
        "type": "timed",
        "delegate": {
            "type": "receiver",
            "serviceName": "firehose:druid:overlord:n2:112-014-0000-0000",
            "bufferSize": 100000
        },
        "shutoffTime": "2018-04-02T15:20:00.000Z"
    },
    "interval": "2018-04-02T14:00:00.000Z/2018-04-02T15:00:00.000Z"
}
```

数据摄入-Indexing-service

- ClippedFirehoseFactory 创建指定时间interval的Firehose
- TimedShutoffFirehoseFactory 创建在指定时间关闭的Firehose

思考

•一个实时任务消费的数据是指定时间段之内的

1. 如果数据迟到，它所属的实时任务已经结束，那这条数据没有可能再被摄入，只能抛弃
2. 一个时间段内的Segment在该段时间的实时任务结束后再不会变，后续任务不会有数据追加之类的操作，所以每一个任务可以根据时间直接为自己消费到的数据生成SegmentID

•实时任务自己结束

数据摄入-Indexing-service

问题：

•实时节点的启动是由Overlord调度的，具体的IP和端口号是随机的，数据发送程序如何知道？

•实时任务需要周期性的结束，将消费到的数据推送的DeepStorage，实时任务的定时结束、启动、启动的任务个数、副本数这些如何管理？

数据摄入- Tranquility

Tranquility

可以视为Druid的客户端:

•可以作为Jar包，依赖到其他程序中使用，典型的可以嵌入到其他流计算框架中使用，如Flink、Spark-streaming、Samza等

•可以作为独立的Java应用部署

数据摄入- Tranquility

- 管理任务生命周期
  - 实时任务定时提交
  - 任务副本与任务数
- 实时节点服务发现
- 消费Kafka数据，通过HTTP服务推送到实时节点上

```
"properties" : {
  "task.partitions" : "1",
  "task.replicants" : "1",
  "topicPattern" : "druid_test"
}
```

```
"properties" : {
  "zookeeper.connect" : [redacted],
  "druid.discovery.curator.path" : "/druid/discovery",
  "druid.selectors.indexing.serviceName" : "druid/overlord",
  "commit.periodMillis" : "15000",
  "consumer.numThreads" : "2",
  "kafka.zookeeper.connect" : "[redacted]",
  "kafka.group.id" : "tranquility-druid-test"
}
```

数据摄入- Tranquility

缺点

•任务失败后无法恢复，如果所有副本任务都失败，那么还是会丢失数据

•使用Kafka高阶API，Tranquility消费数据提交Offset与将数据进入增量索引难以在同一个事务中完成，不能保证exactly-once消费

•数据迟到容忍窗口与任务时长挂钩，无法做到容忍较长时间的数据迟到，更不用说不丢弃知道数据

由于这些缺点，使用Streaming-push的方式应用应当使用Lamda架构，利用离线数据修正实时写入数据

数据摄入-Kafka-Indexing-service

- 引入了Supervisor，用于管理实时任务的生命周期，包括任务的启动、停止、副本管理、失败任务恢复
- 由实时任务主动消费Kafka，无需维护数据推送程序
- 使用Kafka低阶API，自己保存Kafka offset，提升数据可靠性
- 可以不丢弃延时数据

数据摄入-Kafka-Indexing-service

- Supervisor与DataSource一一对应，实现为Overlord中的常驻一个线程

Supervisor工作周期主要步骤

1. 从元数据存储中发现任务（from druid_tasks where active=true）
2. 检查任务执行时长，如果有任务已经到期，则触发该任务的停止流程
3. 创建新的任务，包括新诞生的任务以及补足副本数的任务

每个工作周期都会检查同一个taskGroup下的任务副本数，副本数不足将会启动一个该任务。实现了失败任务的重启恢复。

TIPS: taskGroup：一组消费相同数据任务

数据摄入-Kafka-Indexing-service

- 如何支持不丢迟到数据？

- 一个实时任务不在只生成一个时间范围的Segment，而是根据消费到数据的事件时间，确定自己应当属于那个Segment，如果自己所属的Segment已经存在（有之前的实时任务生成），则新建该时段Segment分片，将自己写入该新的Segment分片中。这样就实现了迟到数据的追加。

数据摄入-Kafka-Indexing-service

- 如何支持不丢迟到数据？

- 依赖Overlord提供的SegmentAllocateAPI（事件时间->SegmentId）
- 先尝试从druid_pendingSegments中获取SegmentId
- 如果没有获取到，则新建SegmentId，并将其插入druid_pendingSegments
- 实时任务结束后，该Segment写入druid_segment表

- 因为Overlord可以获知全局Segment的生成情况，所以可以生成已存在Segment的新分片，达到数据追加的效果。

- 该API调用需要依赖实时任务Segment生成状态，而该状态由消费到的数据构造，所以要求每个Segment消费的数据的顺序一致！

数据摄入-Kafka-Indexing-service

- 如何保证副本任务消费的数据一致？

- 考虑如下情况：

Task1和Task2副本任务：

Task1消费到 (p0, o1), (p0, o2), (p1, o1), (p1, o2)

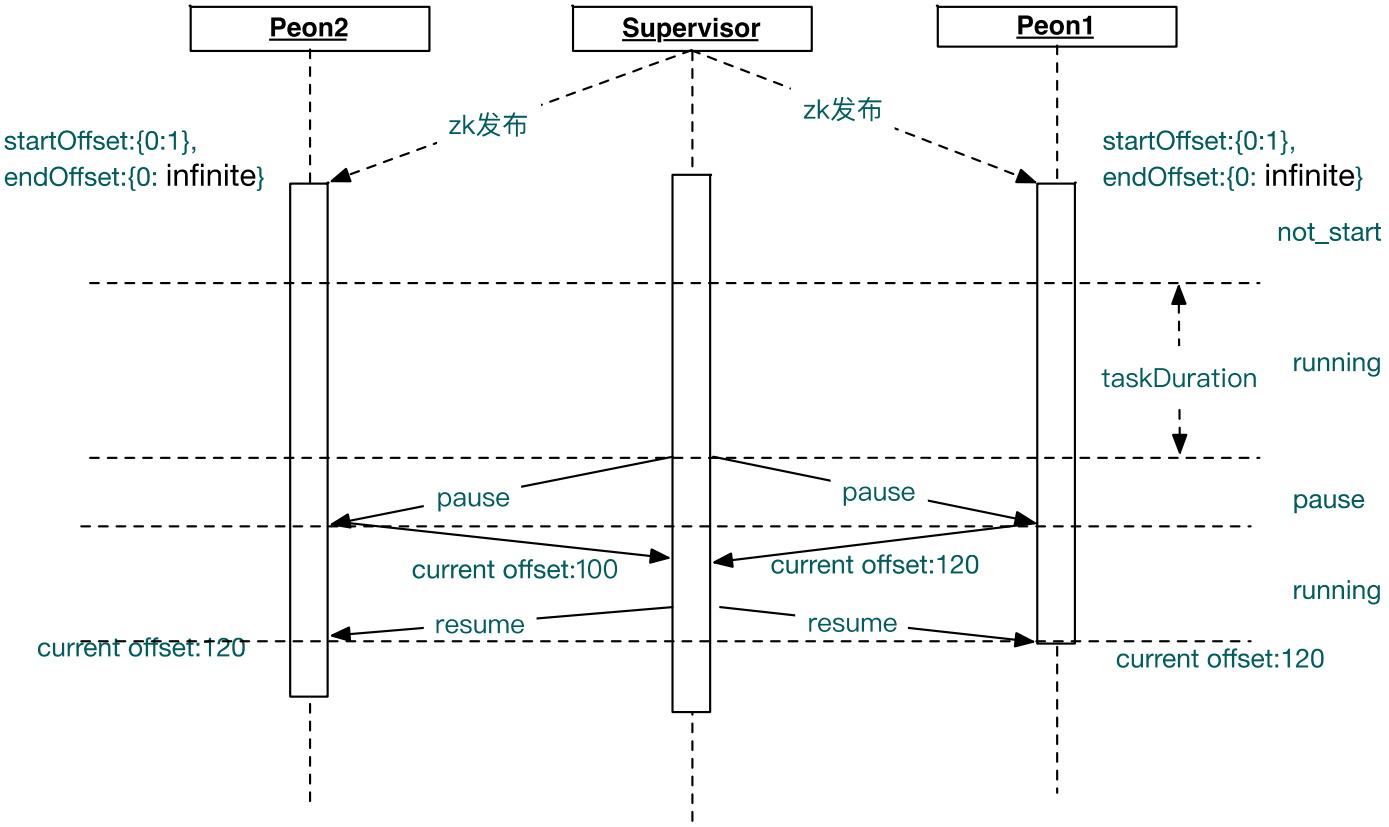
Task2消费到 (p0, o1), (p1, o1), (p0, o2), (p1, o2)

如果maxRowsPerSegment为2，那么Task1与Task2将会生成不同的两个Segment。

所以，一个Segment必须与Topic的partition对应！

数据摄入-Kafka-Indexing-service

- 如何保证副本任务消费的数据一致？
- 任务结束时数据如何对齐？



数据摄入-Kafka-Indexing-service

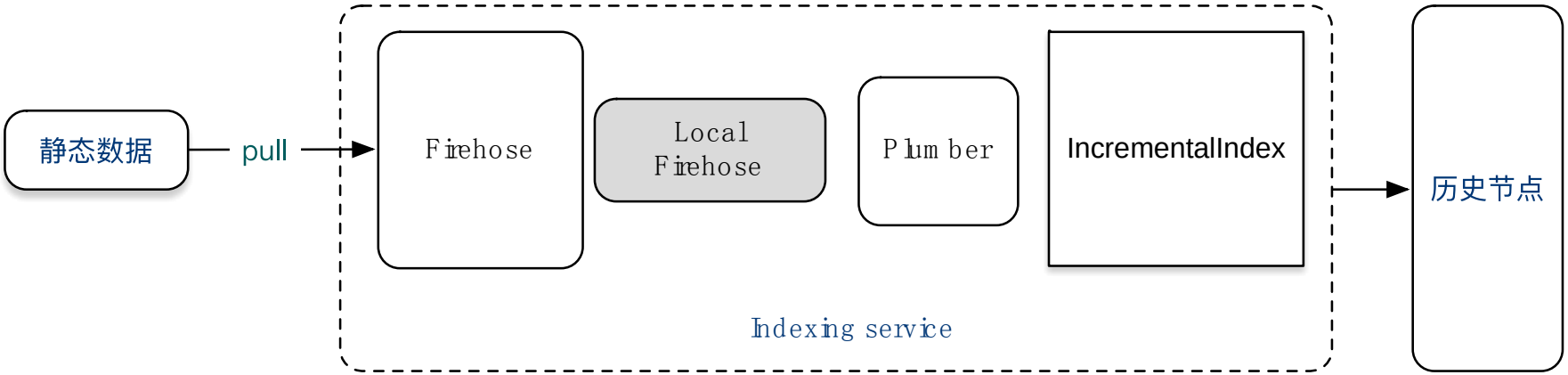
缺点

•由于实时任务数据消费依赖Overlord服务，所以Overlord单机性能将会成为集群规模的瓶颈

•由于Segment与Kafka topic的partition关联，容易造成元数据过度膨胀，引发性能问题。

数据摄入-以索引方式离线导入

- 启动Index task，使用Local Firehose对接静态文件



速度较慢，不适合生产使用

数据摄入-以MR任务离线导入

- 通过HTTP请求提交任务到Overlord
- Overlord在MM上启动Peon进程（index_hadoop任务）
- Peon进程提交MR任务到Yarn
- MR生成Druid Segment，并写到指定HDFS目录
- Peon将Segment信息写入Druid mysql中

应用

- 批量导入离线数据，可支持TestFile、ORC、Parquet文件类型
- 输入源为Druid DataSource时，可用于Segment Merge、Schema Reindex

数据摄入-以MR任务离线导入

- 注意到Index_hadoop任务数据索引工作在Yarn上完成，Peon进程本身只负责任务提交和元数据修改，所以消耗的资源应当比正常Peon进程小得多，所以可以尝试将index_hadoop任务分配到Peon进程资源配置很少的MiddleManager上。

- 可以编写JavaScript类型的MiddleManager select strategy
    - equalDistribution
    - fillCapacity

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**