基于 SeaTunnel 接入 Doris

实现对 Doris 高效的数据导入与导出

刘广东
科大讯飞软件开发工程师

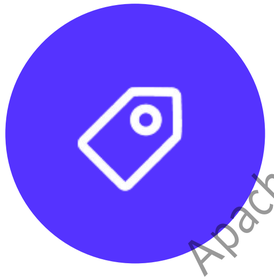
目录

SeaTunnel 介绍

下一代数据集成平台、数据集成（同步）一站式解决方案

简单易用

通过简单的配置和命令即可创建同步任务和运行同步任务



同步过程可监控，指标可量化

同步过程中自动统计任务读取写入的数据量，性能指标，数据延时等信息

丰富的数据生态

国内外数据库消息队列
云存储、云组件
数据湖、仓
SaaS服务
支持用户自定义数据源

全场景支持

支持所有数据集成场景：离线、实时、全量、增量、CDC、CDC整库同步、DDL变更、动态加表...

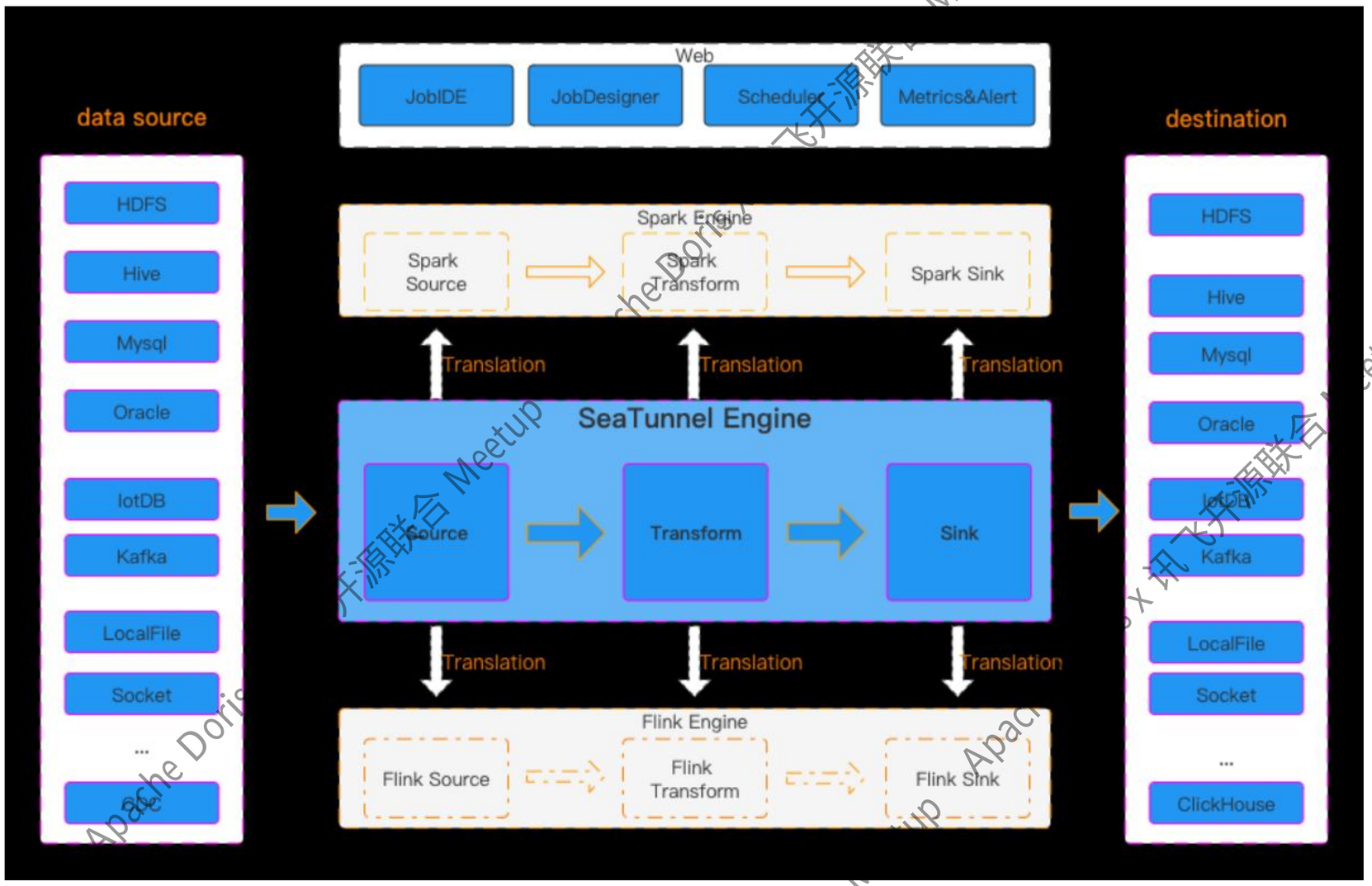
数据一致性保证

数据不丢失、不重复、精确处理一次。
支持断点续传

资源使用少

内存优化
CPU线程优化
多表同步数据库连接共享

SeaTunnel 整体架构



01 与引擎无关的连接器 API

02 连接器翻译层

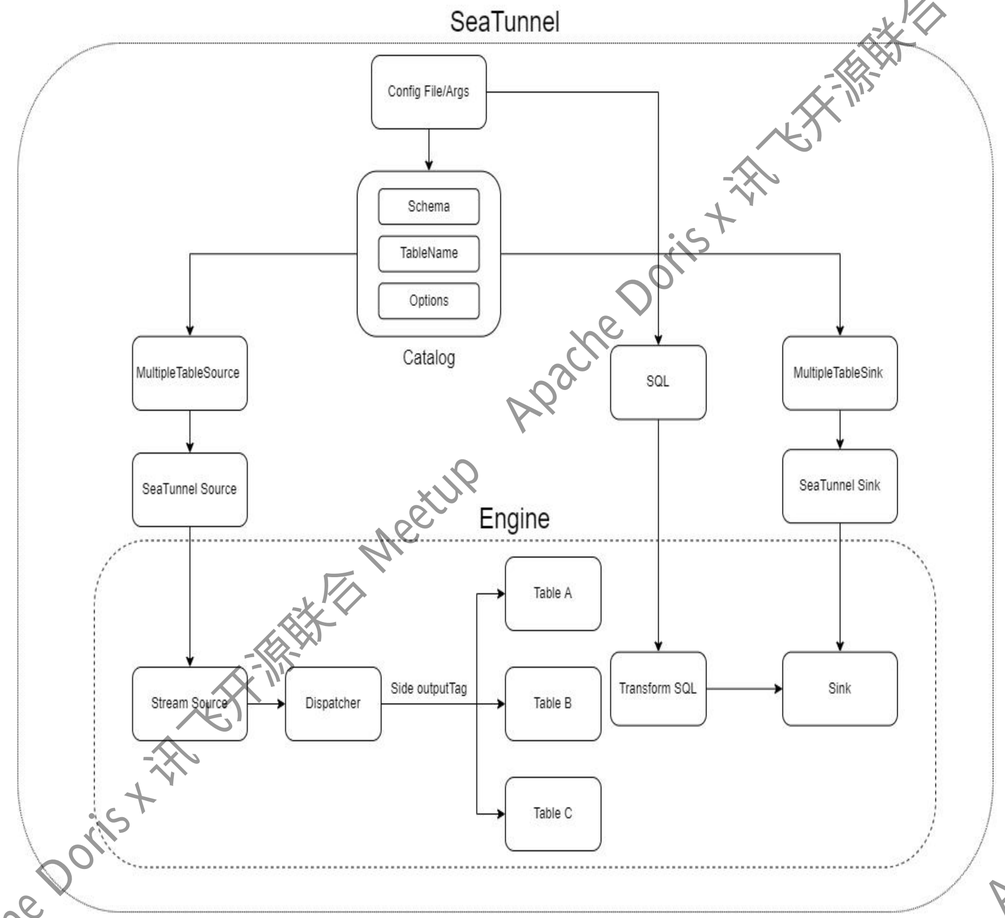
03 Source connector

04 Transform connector

05 Sink connector

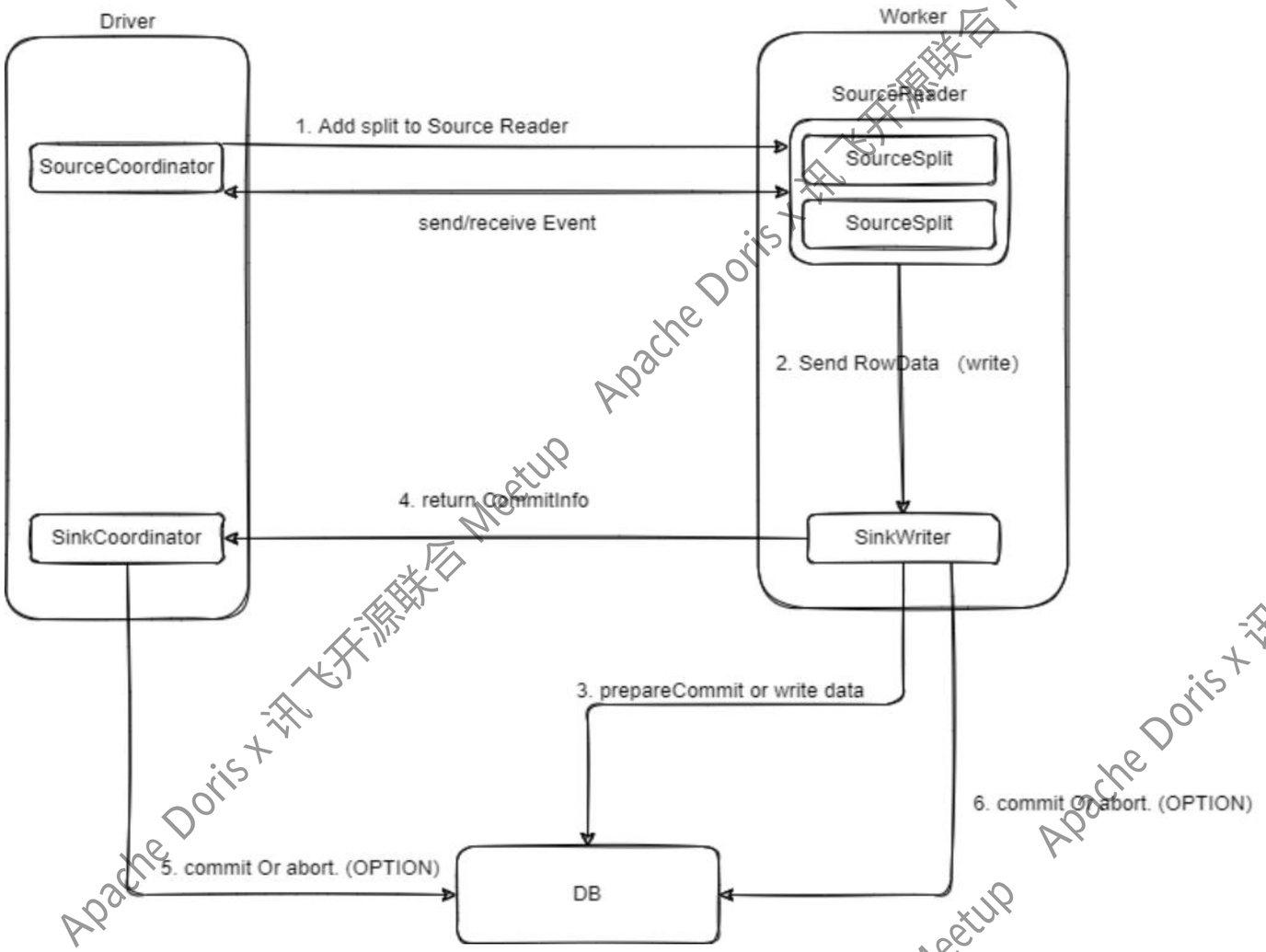
06 多引擎支持

SeaTunnel 架构流程



1. 从配置文件或 Web 等方式获取任务参数；
2. 通过参数从 Catalog 中解析得到 Table Schema、Option 等信息；
3. 以 SPI 方式拉起 SeaTunnel 的 Connector，并注入 Table 信息等；
4. 将 SeaTunnel 的 Connector 翻译为引擎内部的 Connector；
5. 通过 Source-Transform-Sink 完成任务的执行。

SeaTunnel 运行流程



1. SourceCoordinator 负责发现 Split 以及协调 SourceReader；

2. SourceReader 进行数据的实际读取，将数据发送到 Transform 转换后，流转到 SinkWriter；

3. SinkWriter 进行数据的实际写入，或者预提交后将提交信息发送给 SinkCoordinator；

4. SinkAggregatedCommitter 负责协调 SinkWriter，进行正式提交或触发中止；

5. SinkWriter 进行最终的提交或中止。

目录

Source Connector 特性&接口

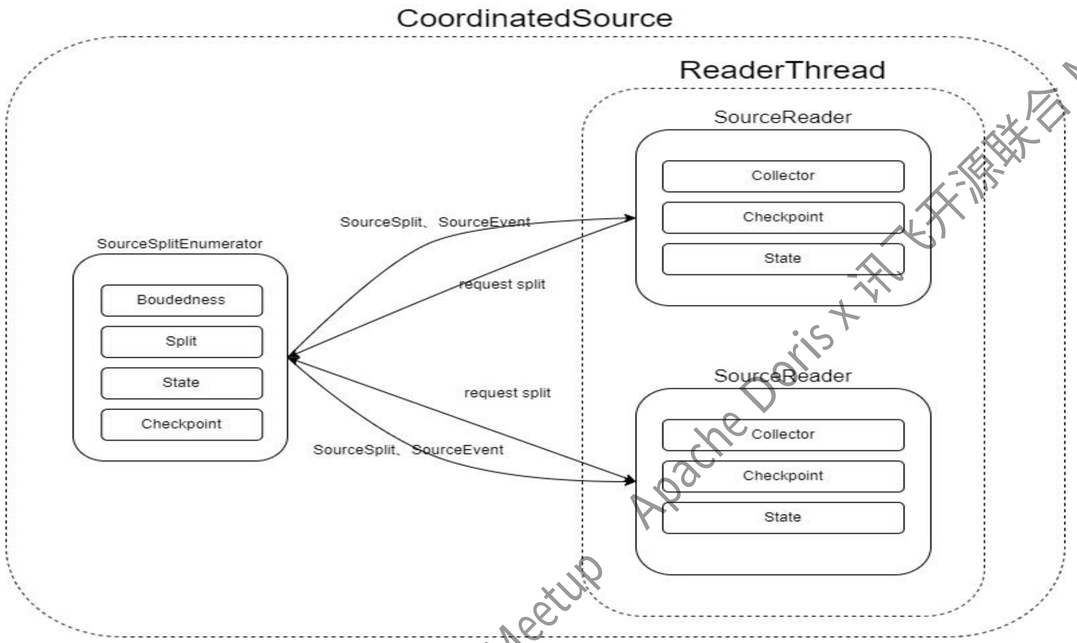
Source 特性

- 支持并行读取，框架管理线程资源。（e.g.消费特定的多分区Kafka 主题）
- 支持split的动态发现。（例如，使用正则表达式使用 Kafka 主题组）
- 支持协调多个reader的工作。（例如Netfilx DBLog CDC无锁算法）
- 支持状态恢复

Source 顶级公共接口

- Boundedness：识别源数据是否有界
- SourceSplitEnumerator：发现split并将它们分配给 SourceReaders
- SourceSplit：所有split类型的接口
- SourceReader：从splits中读取记录
- State：用于存储源状态信息

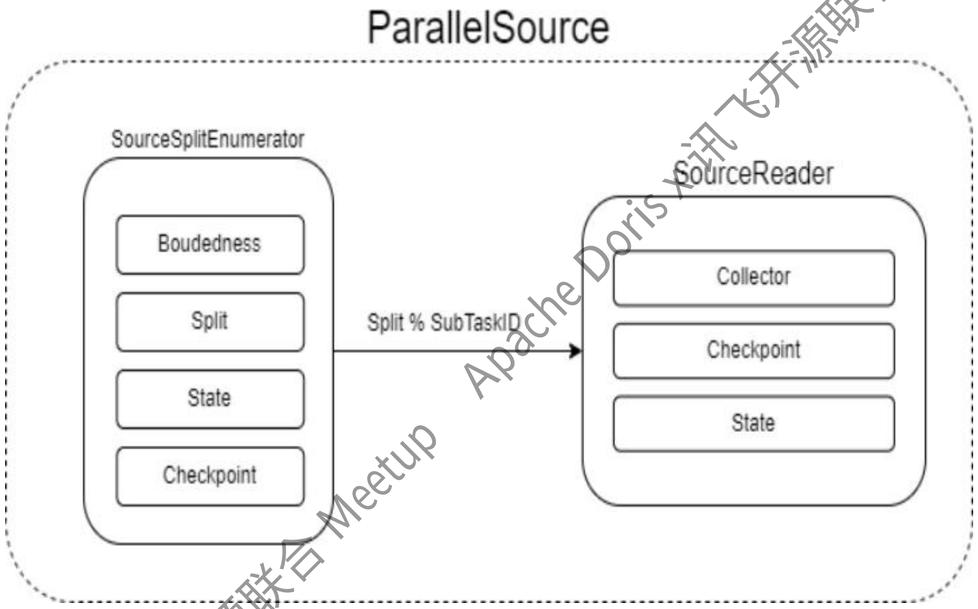
CoordinatedSource & ParallelSource



1. 需要管理 Source Reader 的线程资源；
2. 发现 split 并将其分配给 SourceReaders；
3. 引擎中只能有一个实例（单实例多线程读取）。

特点

1. **简单。**
2. **不能友好地使用引擎资源。**



1. 需要实现一个由 Split ID和 SourceReader ID（SubTask ID）组成的 hash 算法，保证 Split 不会在多个 source reader 中重复读取；
2. 当需要动态获取 Split 时，请确保 SourceSplitEnumerator 始终运行。

特点

1. **需要实现分配算法，对无法切分的数据源不够友好。**
2. **可以将 reader 分发到不同的引擎实例中充分利用引擎资源。**

SeaTunnel Sink 特性概览

结合 Source 支持 Exactly-Once 语义

Sink Write

接收上游数据
写入目标端

State 存储

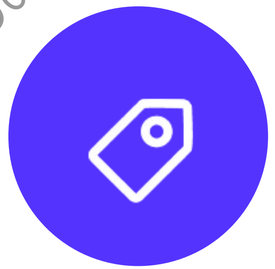
支持状态存储，
HDFS存储状态，
支持基于状态
重启

事务保证

支持两阶段提交，
结合Checkpoint
机制，保证Sink
数据只写一次

Committer

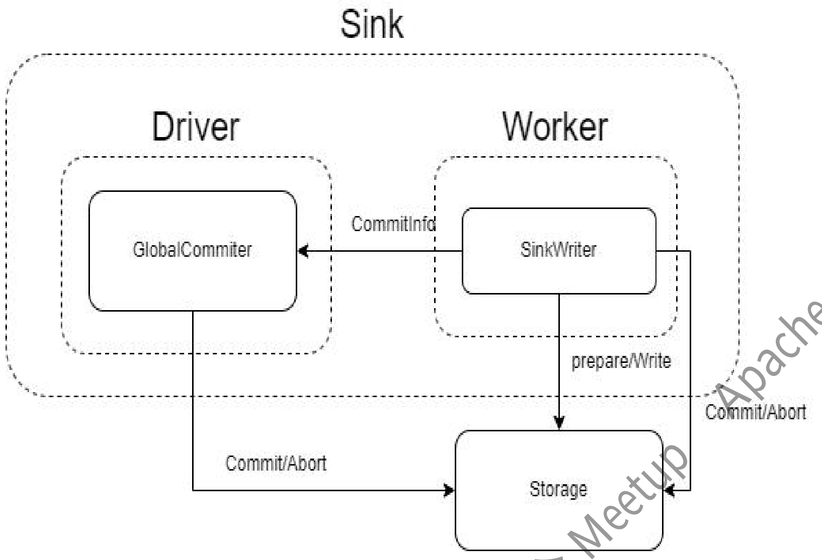
支持 Task 独立
提交事务

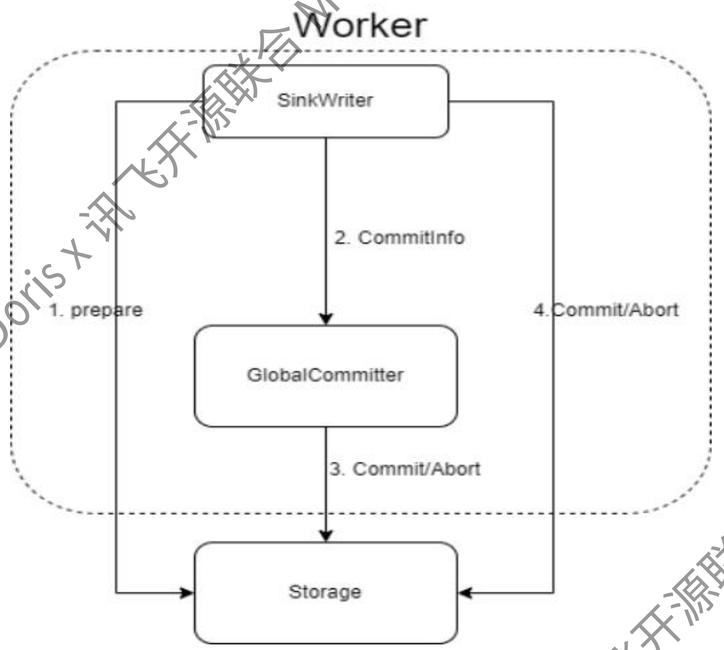


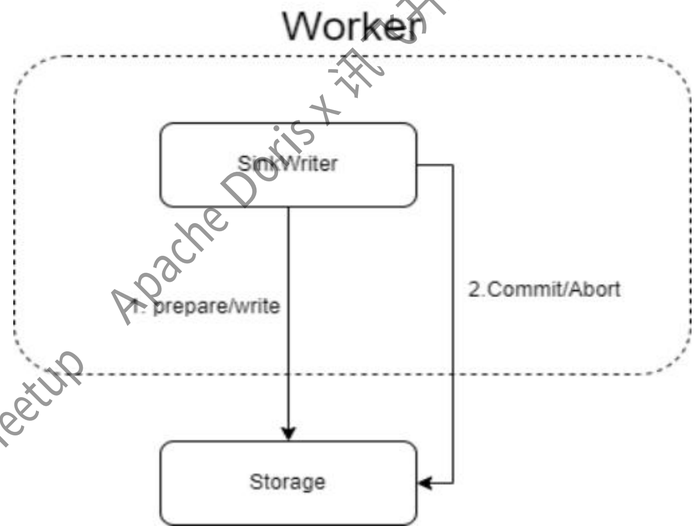
聚合提交

支持 Sink Task
聚合提交

SeaTunnel Sink Connector commit







GlobalCommit Run In Driver

- Global Commiter 运行在 Driver 端
- SinkWriter 运行在 Worker 端
- Flink version >= 1.12
- Spark version >= 2.3

GlobalCommit Run In Worker

- Global Commiter 运行在 Worker 端
- SinkWriter 运行在 Worker 端
- Flink version <= 1.11
- Spark 不适用

Commit In Worker

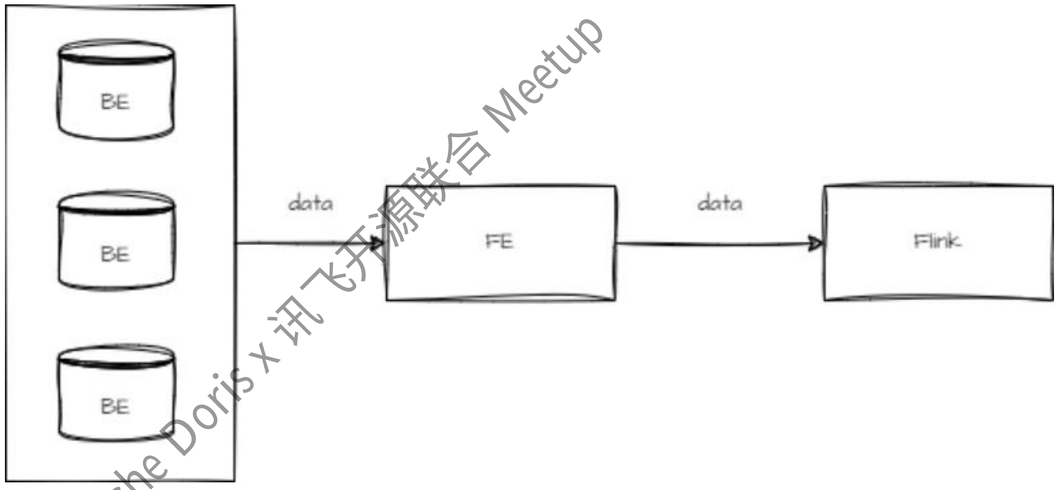
- 不支持 GlobalCommit，在 worker 端 commit
- 每个 Task 单独 Commit 操作
- Flink all versions
- Spark 不适用

目录

Doris 读取数据方法比较

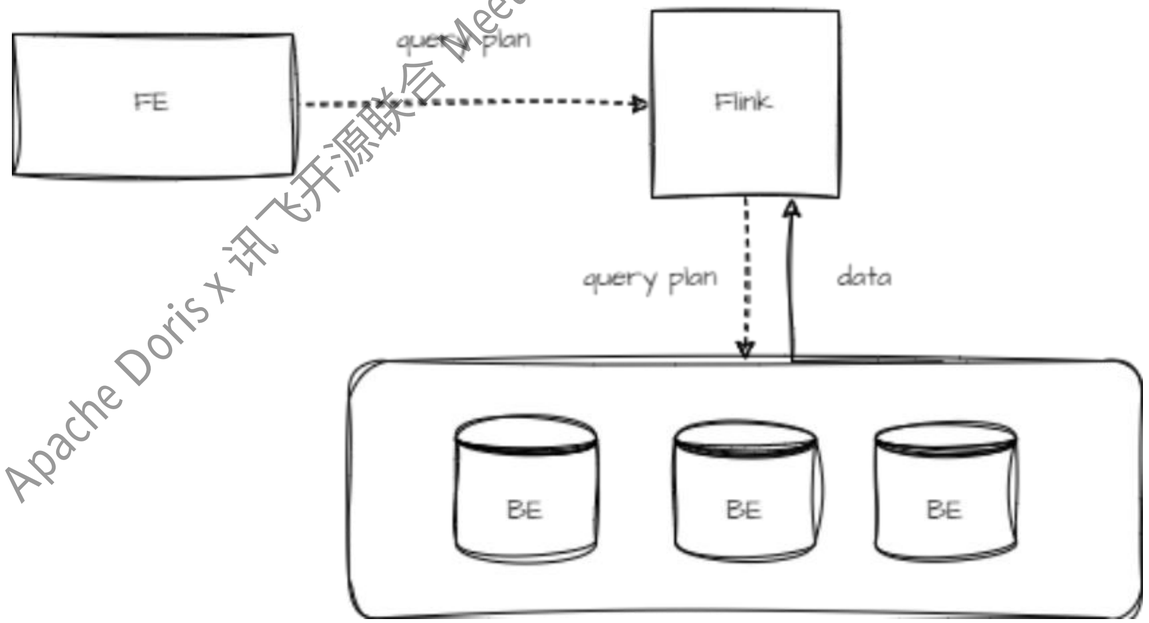
- **Jdbc 方式读取**

仅能从 FE 单点上串行读取数据，数据读取效率较低。



- **查询计划读取**

先从 FE 节点获取查询计划 (Query Plan)，然后将获取到的查询计划作为参数，下发至 BE 节点，最后获取 BE 节点返回的数据。



并行读取：获取数据分布

- **FE 提供获取对单表的查询计划 API**

POST /api/{table_name}/_query_plan

{

“sql” :” select f1,f2 from {table_name}”

}

```
@VisibleForTesting  1 usage   bingquanzhao *
static QueryPlan getQueryPlan(String response, Logger logger) throws DorisConnectorException {
    ObjectMapper mapper = new ObjectMapper();
    QueryPlan queryPlan;
    try {
        queryPlan = mapper.readValue(response, QueryPlan.class);
    } catch (JsonParseException e) {
        String errMsg = "Doris FE's response is not a json. res: " + response;
        logger.error(errMsg, e);
        throw new DorisConnectorException(
                DorisConnectorErrorCode.REST_SERVICE_FAILED, errMsg, e);
    }
    return queryPlan;
}
```

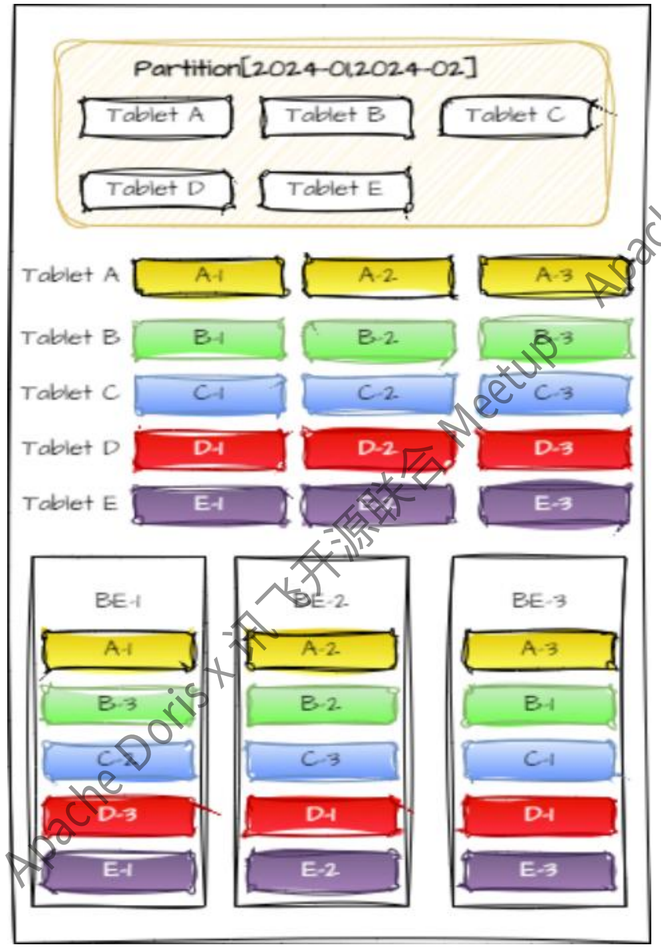
- **查询计划结果的数据结构**

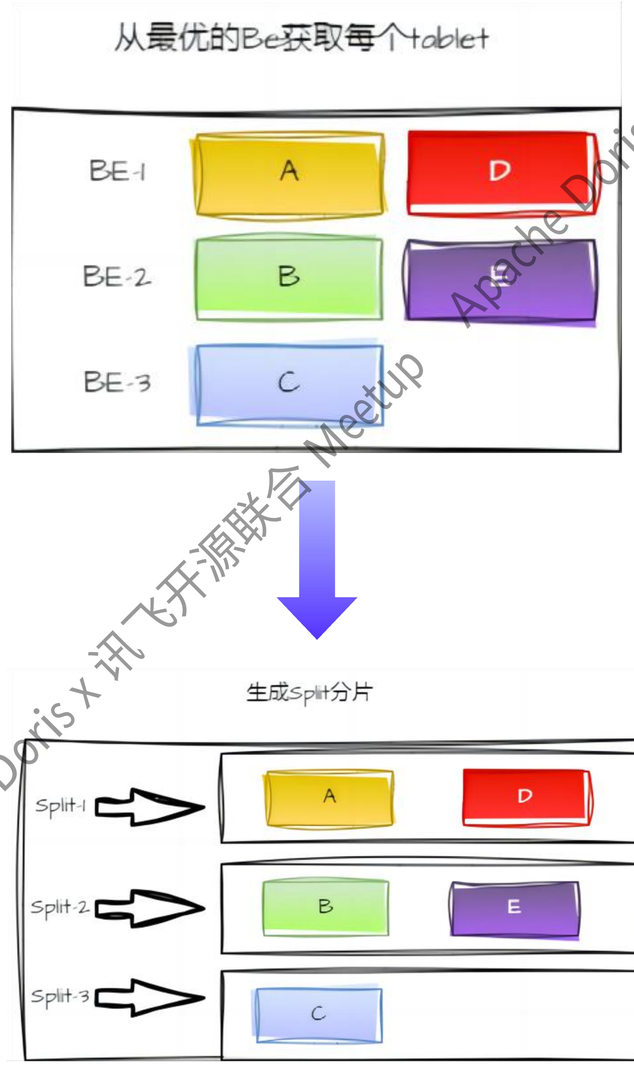
```
@Data  6 usages   bingquanzhao *
public class QueryPlan {
    private int status;
    private String opaqued_query_plan;
    private Map<String, Tablet> partitions;
}
```

```
public class Tablet {  5 usages   bingquanzhao *
    private List<String> routings;  no usages
    private int version;  no usages
    private long versionHash;  no usages
    private long schemaHash;  no usages

}
```

并行读取：读取split

- Doris 表A的 tablet 和副本分布





并行读取：分配 split 到 reader

- 配置并行度

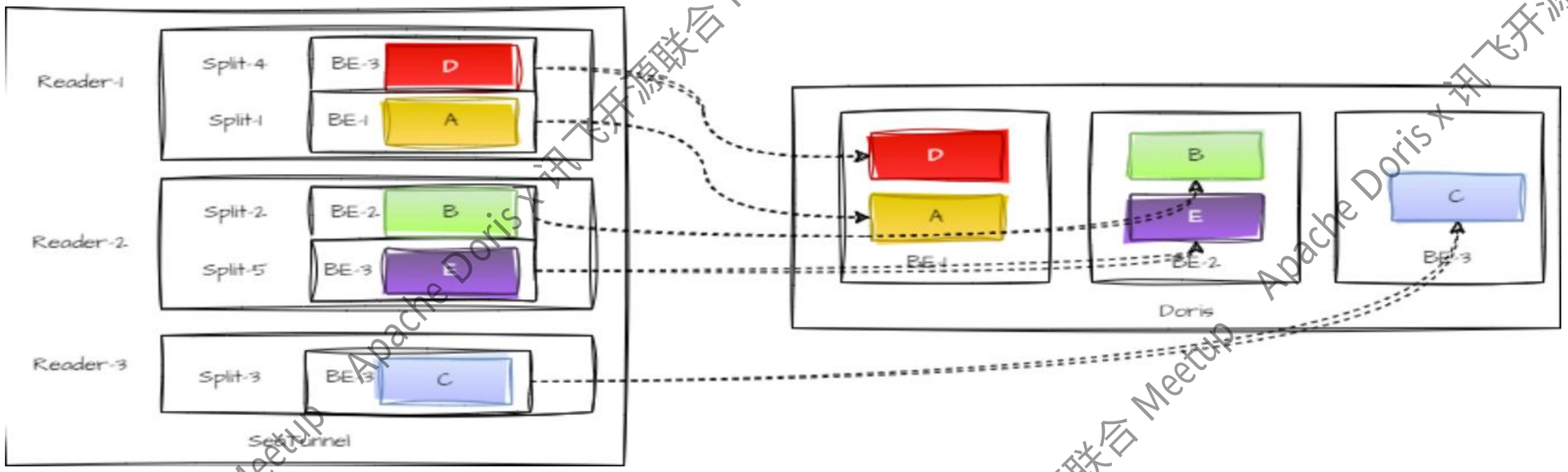
```
env {
  parallelism = 3
  job.mode = "STREAMING"
  checkpoint.interval = 5000
}

source {
   Doris {}
}

transform {
}

sink {
  Doris {}
}
```

- 分配 split 到 reader

```
private static int getSplitOwner(String tp, int numReaders) {
    return (tp.hashCode() & Integer.MAX_VALUE) % numReaders;
}
```

split	reader
split-1	reader-1
split-2	reader-1
split-3	reader-2
split-4	reader-2
split-5	reader-3

并行读取：reader 读取数据



- **BE 提供 Thrift 读取（**https://github.com/apache/doris/blob/master/gensrc/thrift/DorisExternalService.thrift**）**

```
// scan service expose ability of scanning data ability to other compute system
service TDorisExternalService {
    // doris will build  a scan context for this session, context_id returned if success
    TScanOpenResult open_scanner(1: TScanOpenParams params);

    // return the batch_size of data
    TScanBatchResult get_next(1: TScanNextBatchParams params);

    // release the context resource associated with the context_id
    TScanCloseResult close_scanner(1: TScanCloseParams params);
}
```

并行读取：状态恢复

- 将 reader 未读取的 split 存储在 state

```
@AllArgsConstructor  9 usages  bingquanzhao
@Getter
public class DorisSourceState implements Serializable {
    private static final long serialVersionUID = 812677654882088818L;  no usages
    private boolean shouldEnumerate;
    private Map<Integer, List<DorisSourceSplit>> pendingSplit;
}
```

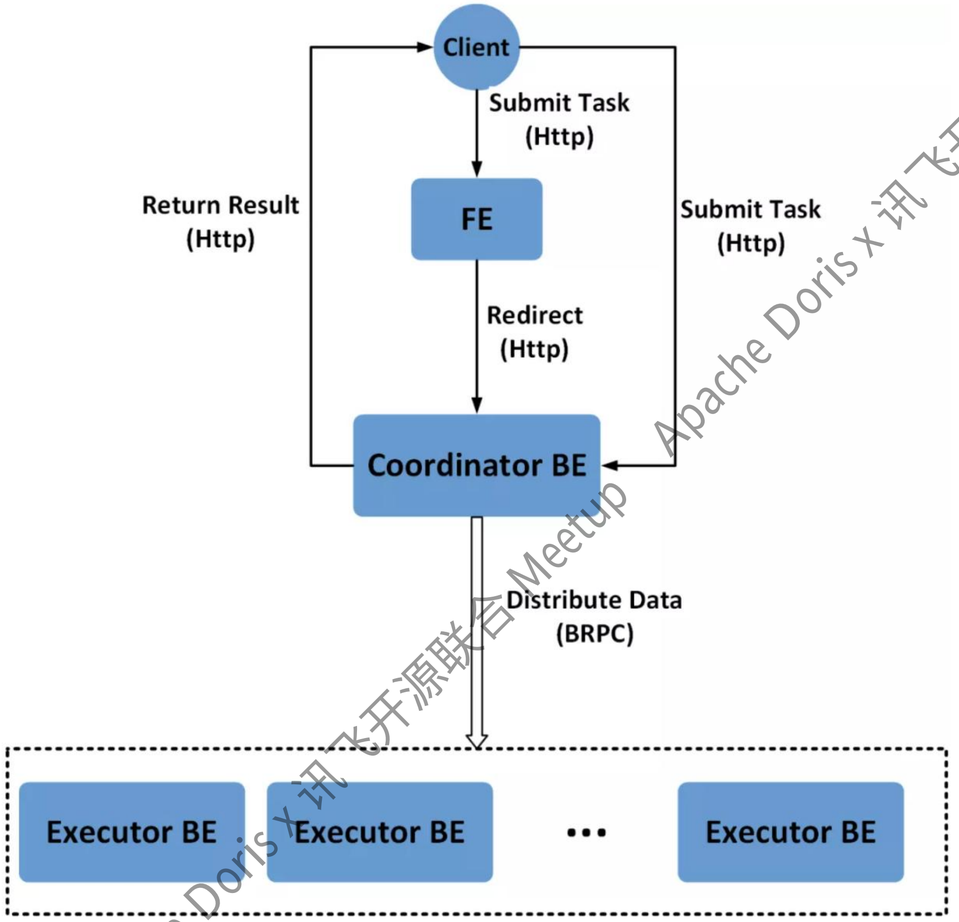
```
    @Override  bingquanzhao
    public DorisSourceState snapshotState(long checkpointId) {
        synchronized (stateLock) {
            return new DorisSourceState(shouldEnumerate, pendingSplit);
        }
    }
```

- reader 在读取时主动从 split 队列消费分片

```
    @Override  bingquanzhao
    public void pollNext(Collector<SeaTunnelRow> output) throws Exception {
        synchronized (output.getCheckpointLock()) {
            DorisSourceSplit nextSplit = splitsQueue.poll();
            if (nextSplit != null) {
                PartitionDefinition partition = nextSplit.getPartitionDefinition();
                valueReader = new DorisValueReader(partition, dorisConfig, seaTunnelRowType);
                while (valueReader.hasNext()) {
                    SeaTunnelRow record = valueReader.next();
                    output.collect(record);
                }
            }
            if (Boundedness.BOUNDED.equals(context.getBoundedness())
                    && noMoreSplits
                    && splitsQueue.isEmpty()) {
                // signal to the source that we have reached the end of the data.
                log.info("Closed the bounded Doris source");
                context.signalNoMoreElement();
            }
        }
    }
```

目录

Doris StreamLoad 原理



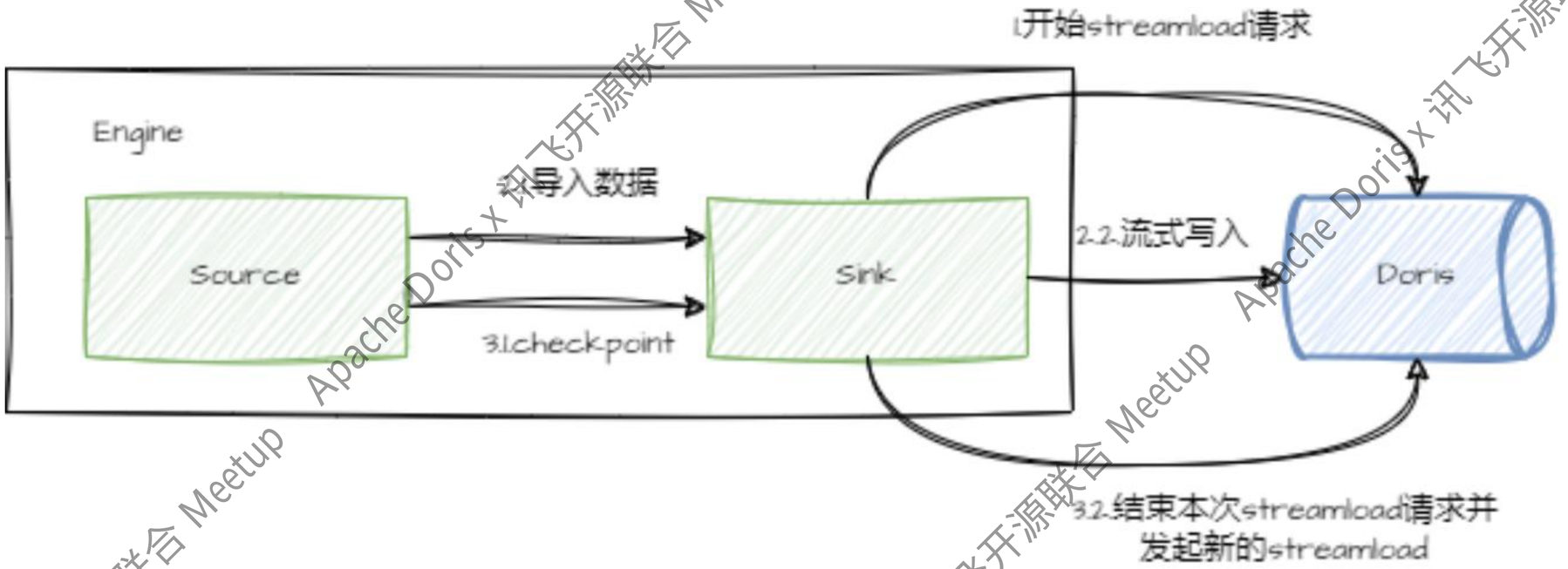
- 用户将 Stream Load 的Http 请求提交给 FE

- FE 会通过 Http 重定向（Redirect）将数据导入请求转发给任意一个 BE 节点，该 BE 节点将作为本次 Stream Load 任务的 Coordinator（也可直接提交 Http 请求到 BE）

> 向FE发送事务请求
> 从FE获取导入执行计划
> 接收实时数据
> 分发数据到其他Executor BE节点
> 数据导入结束后返回结果给用户

- Executor BE 节点负责将数据写入存储层

Doris sink 流式写入

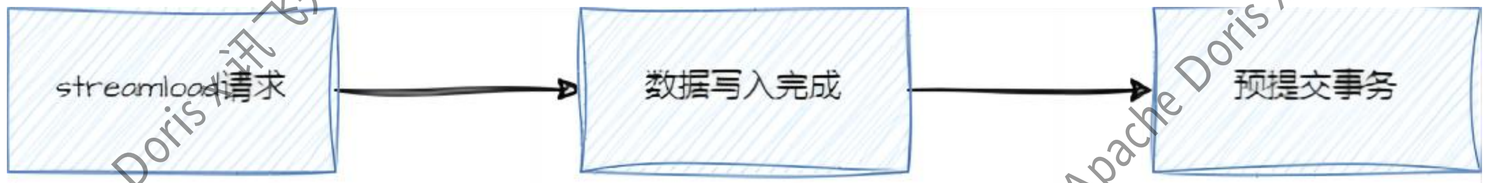


1. 引擎任务启动后，会异步发起一个 Stream Load 的 Http 请求。
2. 接收到实时数据后，通过 Http 的分块传输编码（Chunked transfer encoding）机制持续向 Doris 传输数据。
3. 在 Checkpoint 时结束 Http 请求，完成本次 Stream Load 写入，同时异步发起下一次 Stream Load 请求。

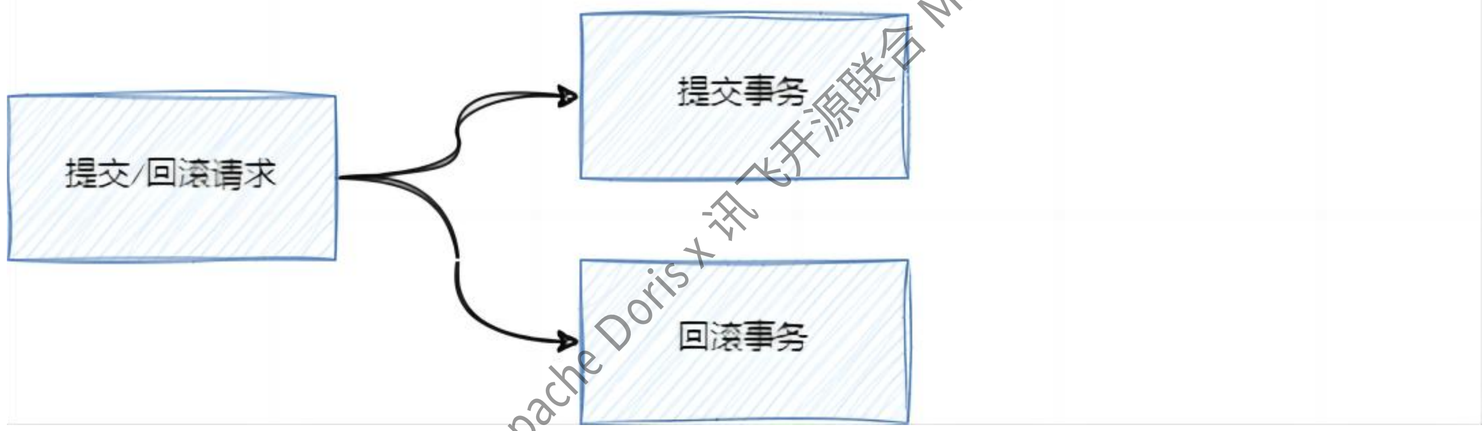
Doris sink Exactly-Once

Streamload 通过两阶段提交保证 Exactly-Once

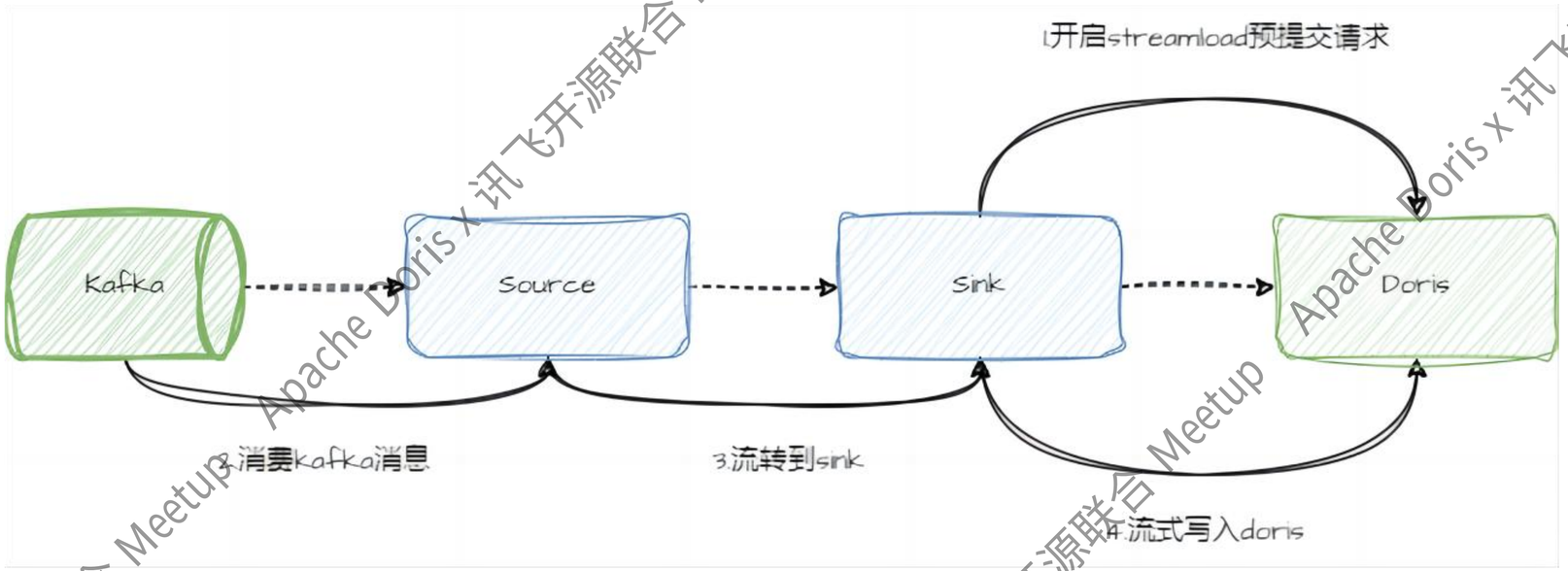
- **第一阶段**



- **第二阶段**

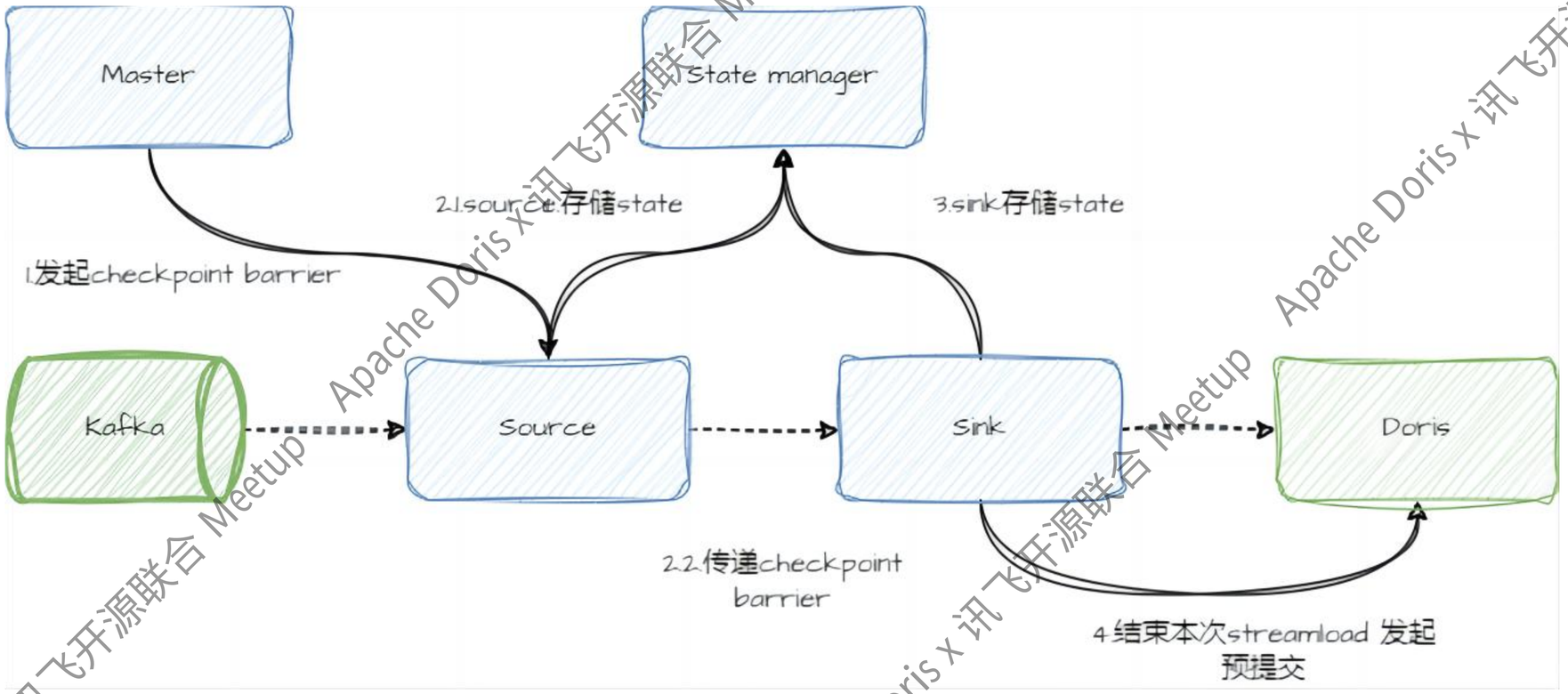


Exactly-Once step1：开启事务



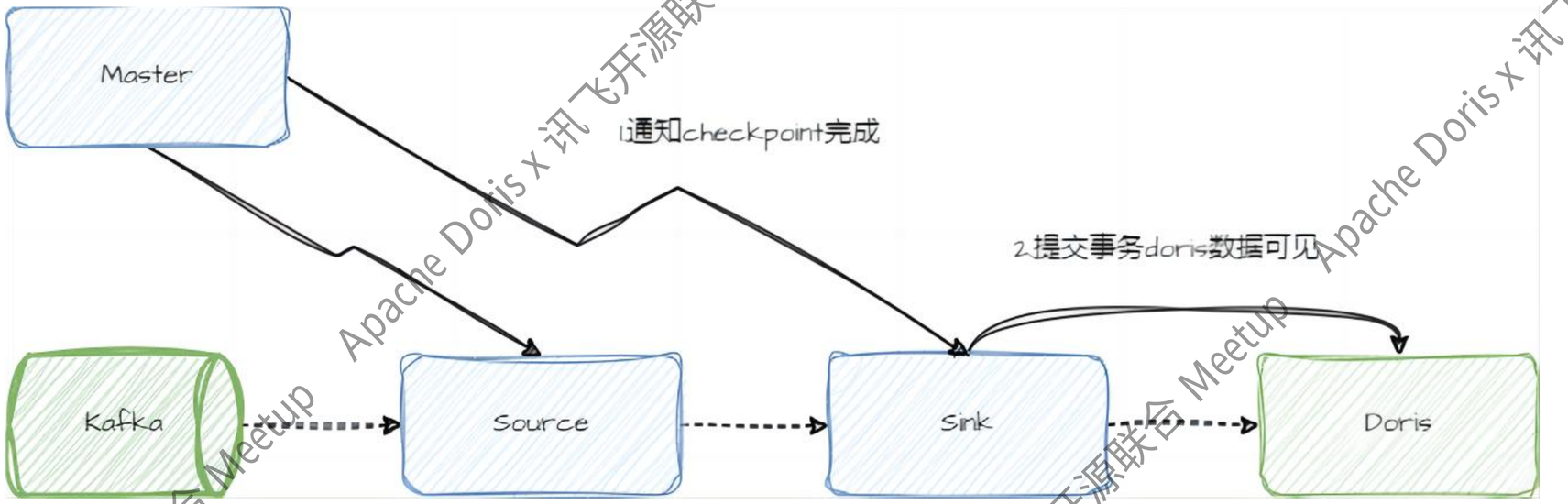
引擎任务启动时，会发起一个 Stream Load 的 PreCommit 请求，此时会先开启一个事务，同时会通过 Http 的 Chunked 机制将数据持续发送到 Doris。

Exactly-Once step2：预提交



在 Checkpoint 时，结束数据写入，同时完成Http请求，并且将事务状态设置为预提交(preCommitted)，此时数据已经写入BE，对用户不可见。

Exactly-Once step3：提交事务



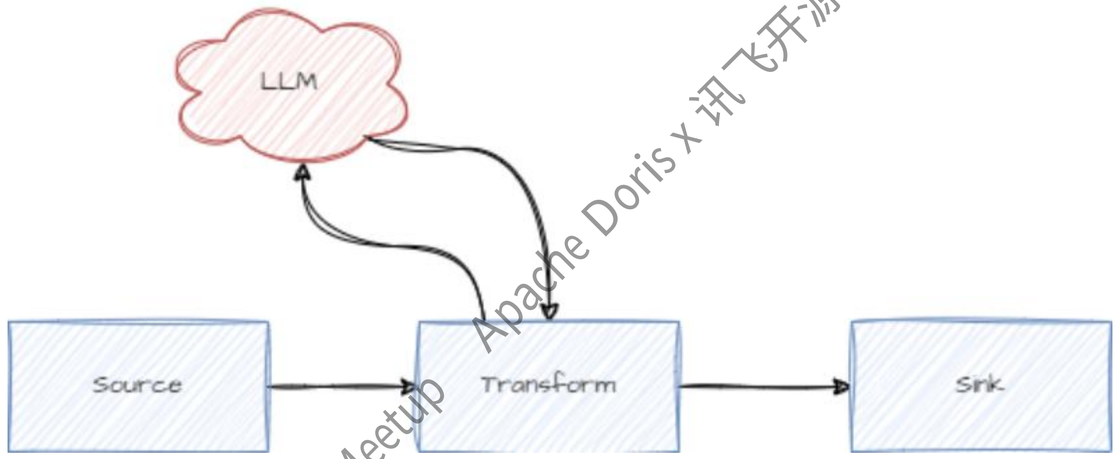
Checkpoint 完成后，发起 Commit 请求，并且将事务状态设置为提交（Committed），完成后数据对用户可见。

目录

与 AI 的结合

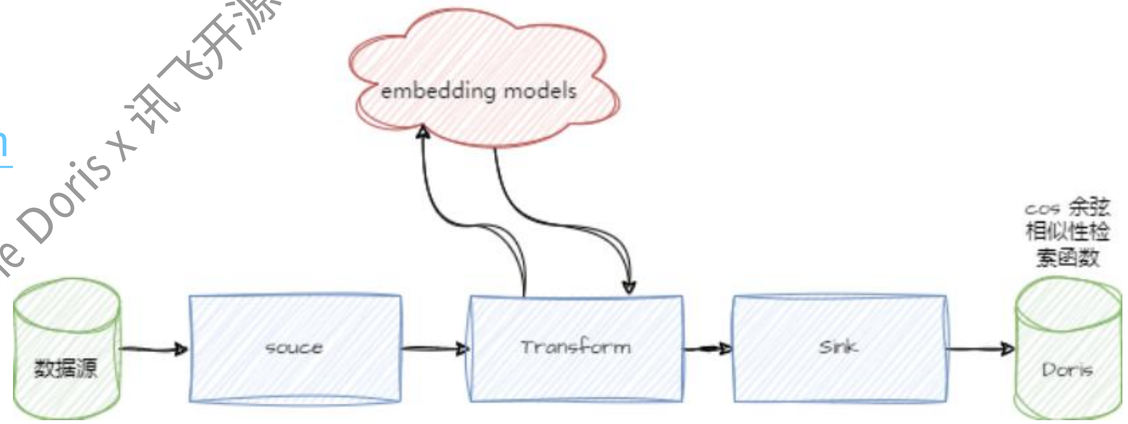
- **SeaTunnel 借助 LLM 自动实现数据打标签**

  https://github.com/apache/seatunnel/pull/7303



- **Doris 作为向量数据库与Langchain结合使用**

  https://python.langchain.com/v0.2/docs/integrations/vectorstores/apache_doris/



Thanks !