

深入解读 Flink CDC 增量快照框架

徐榜江 (雪尽)

Flink CDC Maintainer & Apache Flink Committer



目录

01 Flink CDC 简介

02 Flink CDC 增量快照算法

03 Flink CDC 增量快照框架

04 社区发展规划

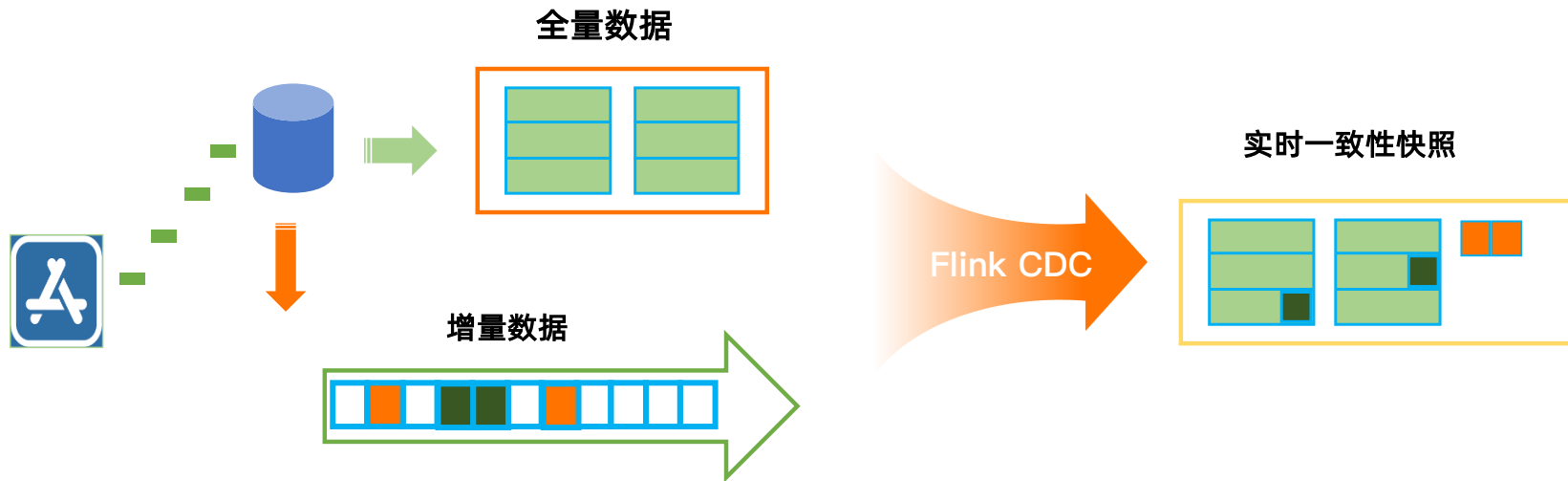
01

Flink CDC 简介

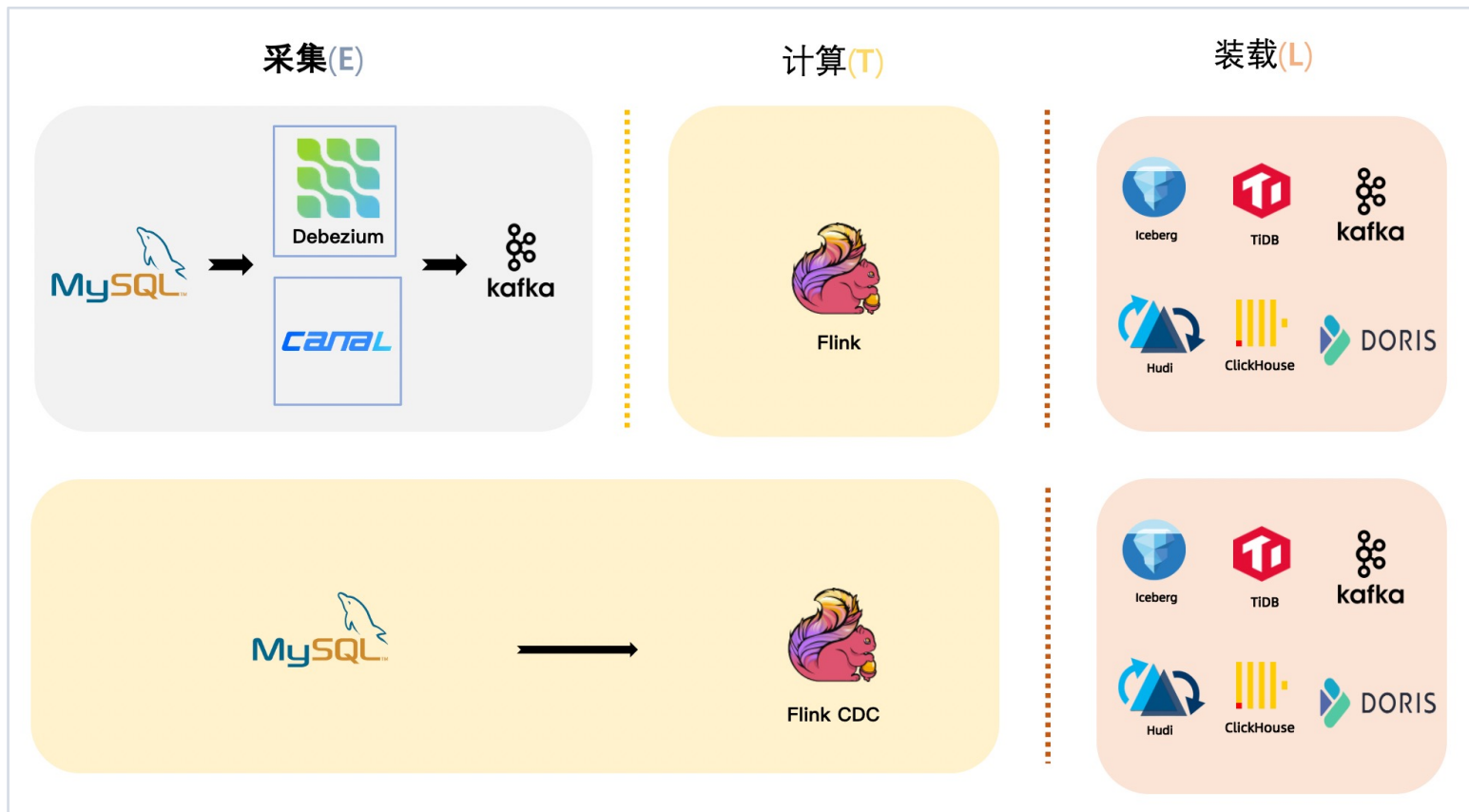


Flink CDC 技术

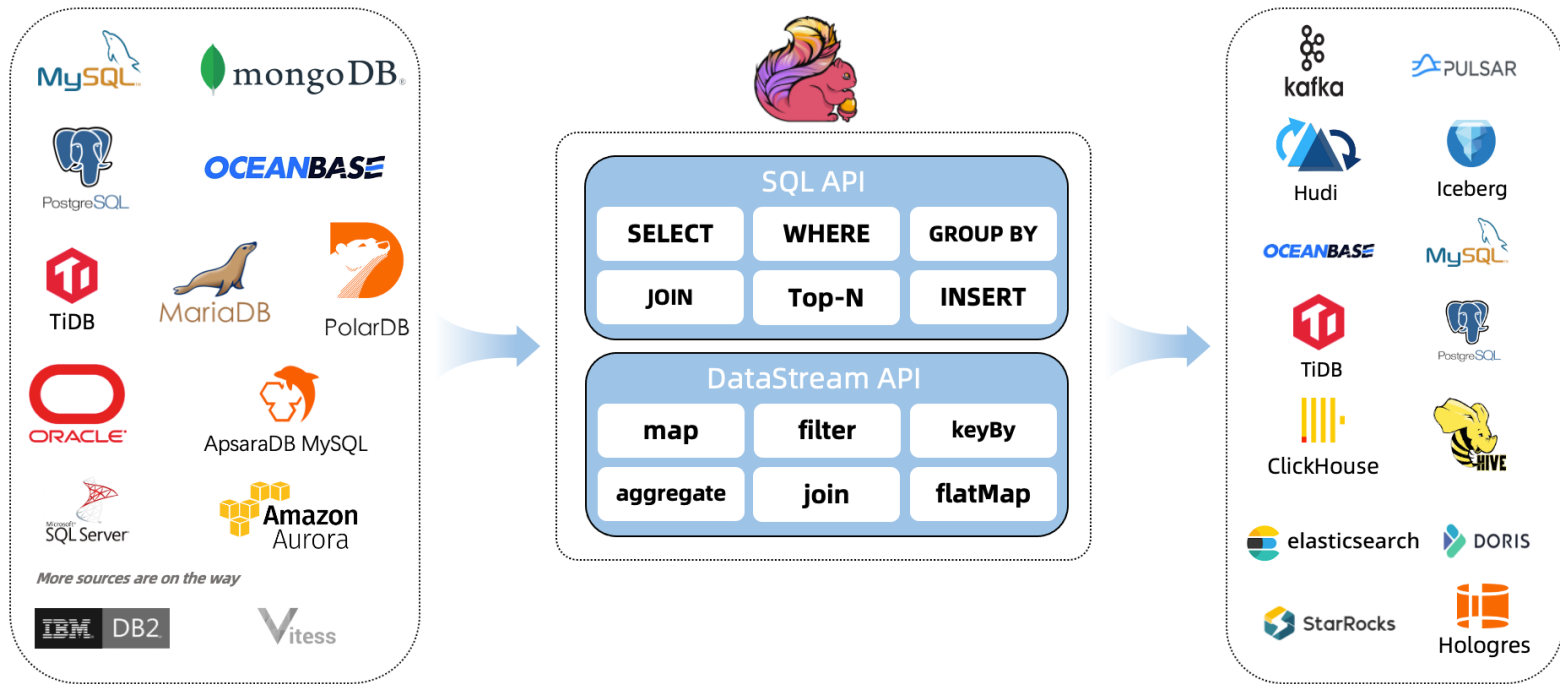
Flink CDC 是基于数据库的日志CDC (Change Data Capture)技术, 实现了全量和增量的一体化读取能力, 借助 Flink 优秀的管道能力和丰富的上下游生态, 支持实时捕获、加工多种数据库的变更并输出到下游。



Flink CDC 技术



Flink CDC 技术



02

Flink CDC 增量快照算法



Flink CDC 增量快照算法

Flink CDC 1.0 痛点

➤ 一致性通过加锁保证

Debezium 在保证数据一致性时，需要对读取的库或表加锁，全局锁可能导致数据库hang住，表级锁会锁住加锁的表(无法更新)，**DBA 一般不给锁权限。**

➤ 不支持水平扩展

Flink CDC 1.0 只支持单并发，在全量阶段读取阶段，如果表非常大(亿级别)，读取时间都在 小时 级别

➤ 全量读取阶段不支持 checkpoint

Flink CDC 1.0 在全量读取阶段是不支持 checkpoint 的，因此会存在一个问题：当我们同步全量数据时，假设需要 5 个 小时，当我们同步了 4 小时的时候作业失败，这时候就需要重新开始，重新读取 5 个小时。

Flink CDC 增量快照算法

Flink CDC 1.0 锁分析

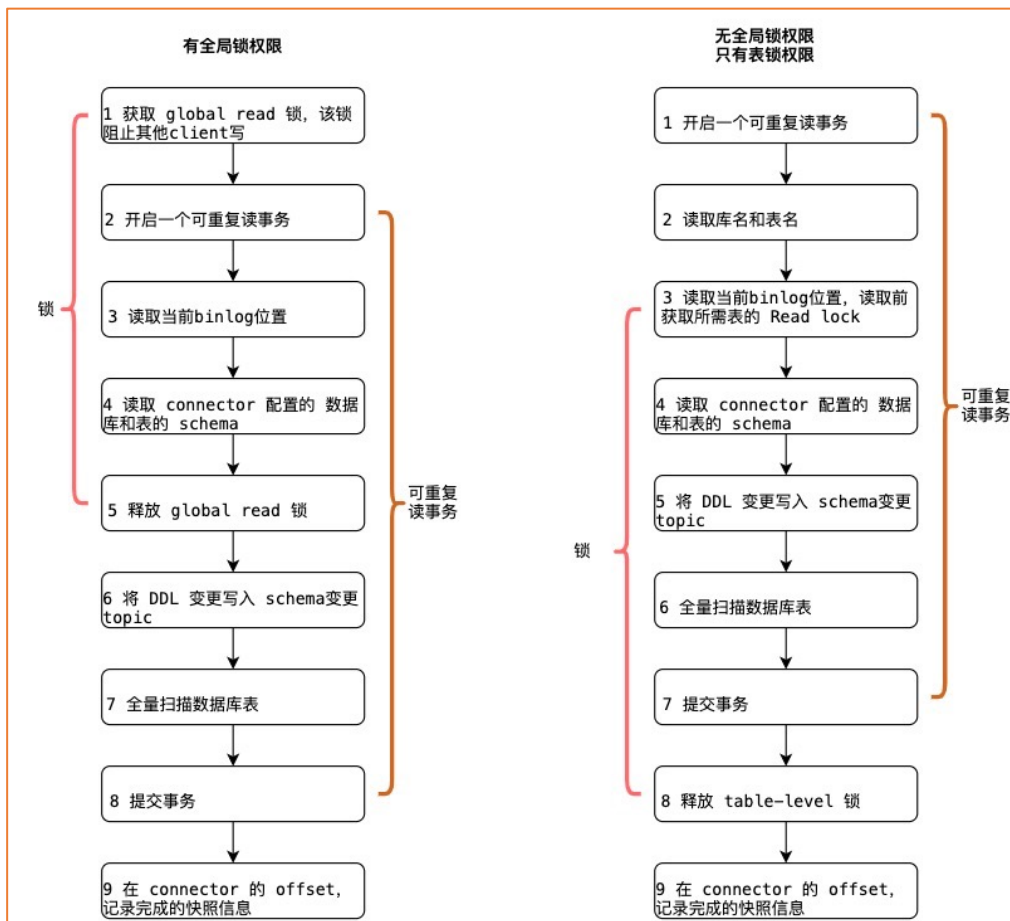
Flink CDC 1.0 底层封装了 Debezium, Debezium 同步一张表分为两个阶段：

- 全量阶段：查询当前表中所有记录
- 增量阶段：从 binlog 消费变更数据

加锁发生在全量阶段，目的是为了确定增量阶段的初始位点，保证增量 + 全量实现一条不多，一条不少，保证数据一致性（exactly-once 语义）

Flink CDC 增量快照算法

Flink CDC 1.0 锁分析



Flink CDC 增量快照算法

Flink CDC 1.0 锁分析

```
MySQL> FLUSH TABLES WITH READ LOCK
```

- 该命令等待所有正在进行的 update 完成，同时阻止所有新来的 update。
- 该命令执行成功前必须等待所有正在运行的 select 完成，所有等待执行的 update 会等待的更久。更坏的情况是，在等待正在运行 select 完成时，DB 实际上处于不可用状态，即使是新加入的 SELECT 也会被阻止，这是 MySQL Query Cache 机制。
- 该命令阻止其他事务 commit。

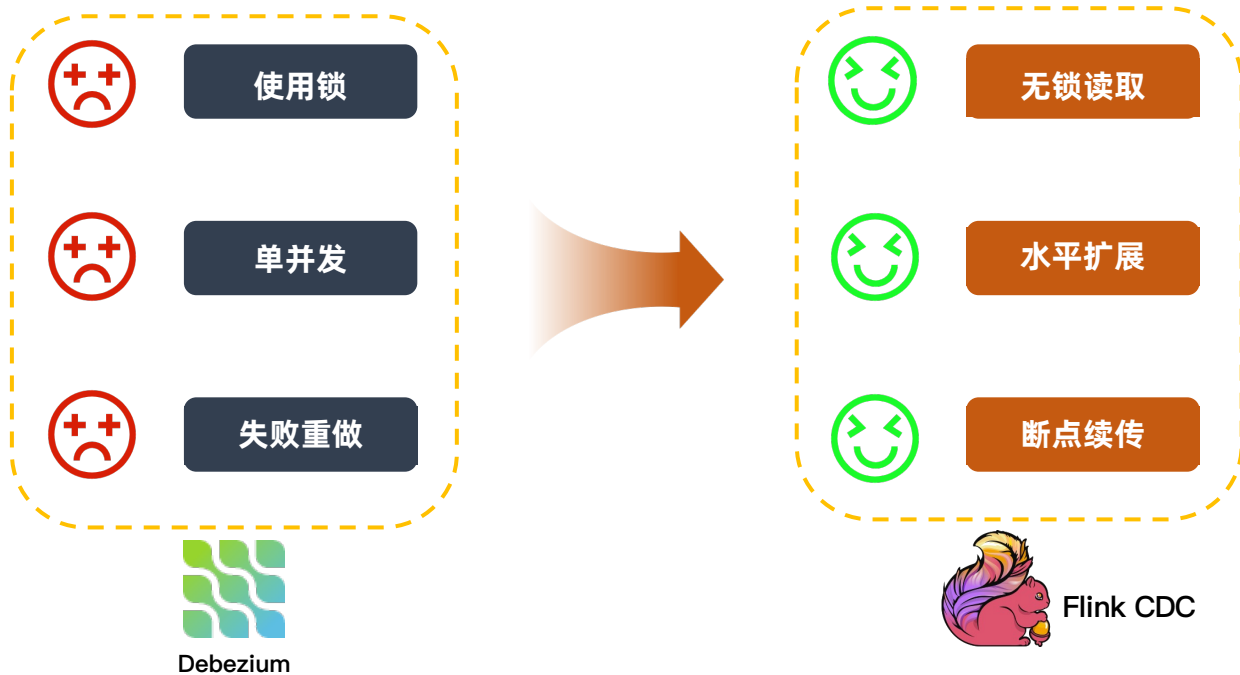
结论：加锁时间是不确定的，极端情况会 hang 住数据库

Percona 文章：

<https://www.percona.com/blog/2014/03/11/introducing-backup-locks-percona-server-2/>

Flink CDC 增量快照算法

设计目标



Flink CDC 增量快照算法

设计方案

- 无锁算法
- 并行读取
- 断点续传



DBLog 论文算法变种[1], 全程无锁

FLIP-27 Source 实现[2], 架构优雅

[1] : <https://arxiv.org/pdf/2010.12597v1.pdf>

[2] : <https://cwiki.apache.org/confluence/display/FLINK/FLIP-27%3A+Refactor+Source+Interface>

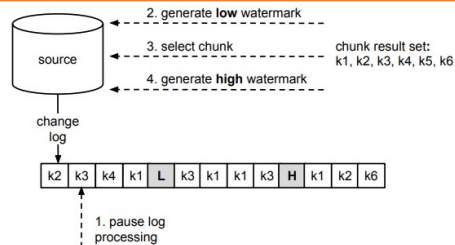
Flink CDC 增量快照算法

DBLog 算法原理

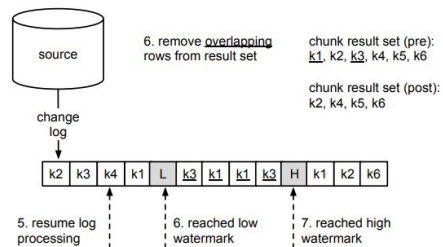
Algorithm 1: Watermark-based Chunk Selection

Input: table

```
(1) pause log event processing
    lw := uuid(), hw := uuid()
(2) update watermark table set value = lw
(3) chunk := select next chunk from table
(4) update watermark table set value = hw
(5) resume log event processing
    inwindow := false
    // other steps of event processing loop
    while true do
        e := next event from changelog
        if not inwindow then
            if e is not watermark then
                append e to outputbuffer
            else if e is watermark with value lw then
                inwindow := true
        else
            if e is not watermark then
                if chunk contains e.key then
                    remove e.key from chunk
                    append e to outputbuffer
                else if e is watermark with value hw then
                    for each row in chunk do
                        append row to outputbuffer
(6)
(7)
    // other steps of event processing loop
    ...
```

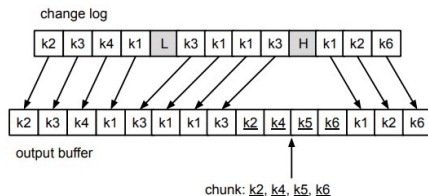


(a) steps 1-4



(b) steps 5-7

Figure 3: Watermark-based Chunk Selection



Flink CDC 增量快照算法

Chunk 划分

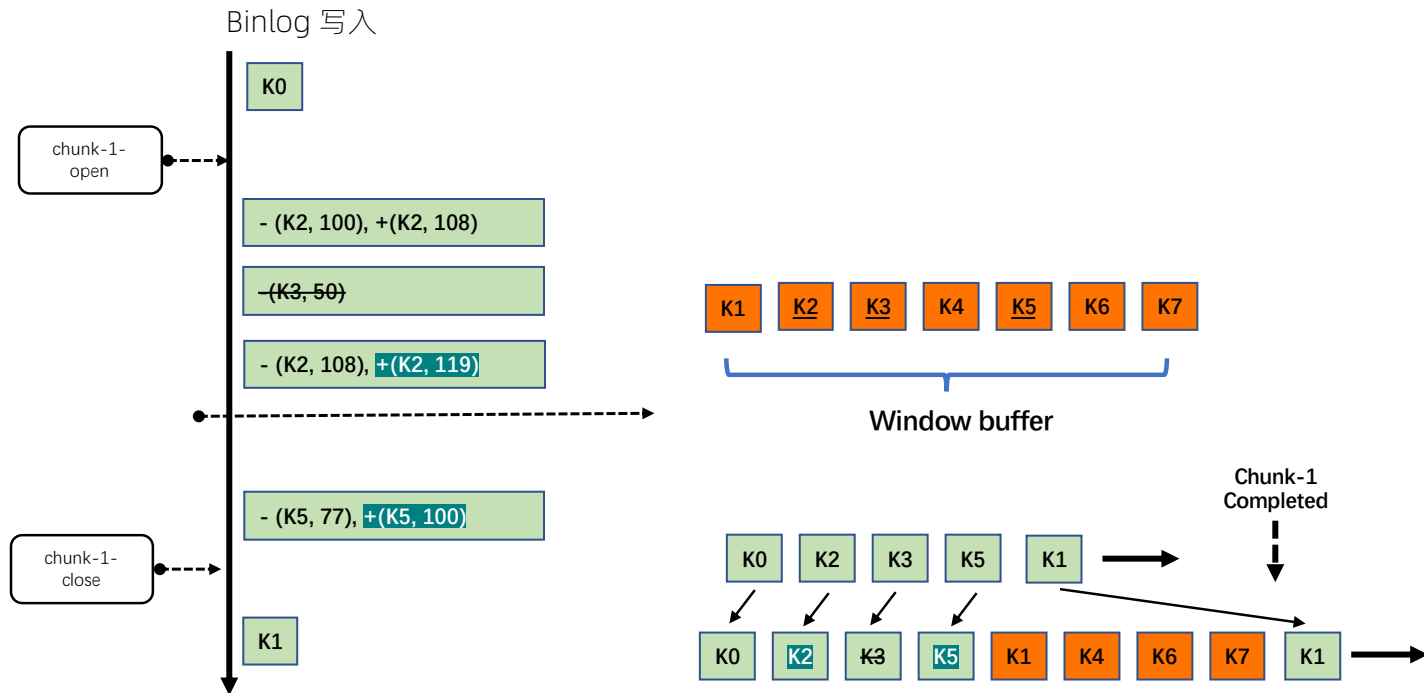
ID(PK)	C1	C2	
K1	A		} Chunk-0
K2	B		
K3	C		} Chunk-1
...	..		
K100	X		} Chunk-14
K101	Y		
K103	X		
K104	Z		
			} Chunk-15

- select max(id) from T1
=> **K104**
- select max(id) from T1 where id <= K104 order by id asc limit 3
=> **K3**
- select max(id) from T1 where id > K101 and id <= K104 order by id asc limit 3
=> **K104**

Chunk-ID	key range
Chunk-0	(null, k1)
Chunk-1	[k1, k3)
....	
Chunk-14	[k101, K104)
Chunk-15	[K104, null)

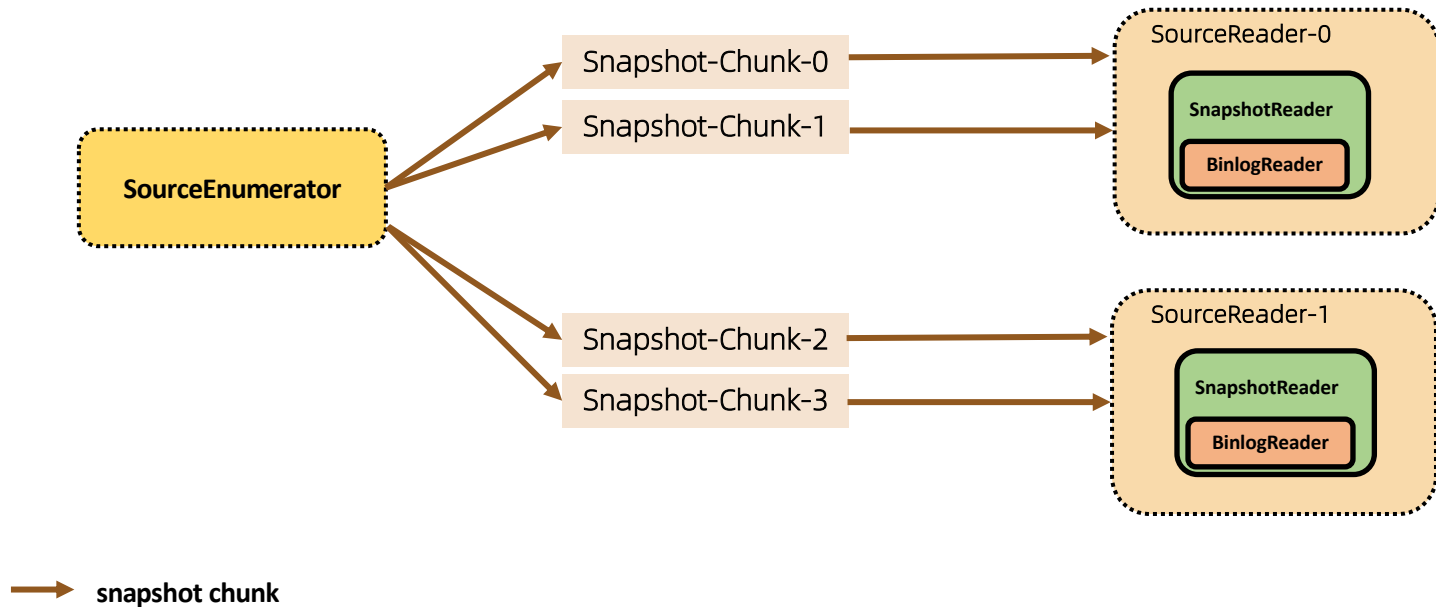
Flink CDC 增量快照算法

Chunk 读取



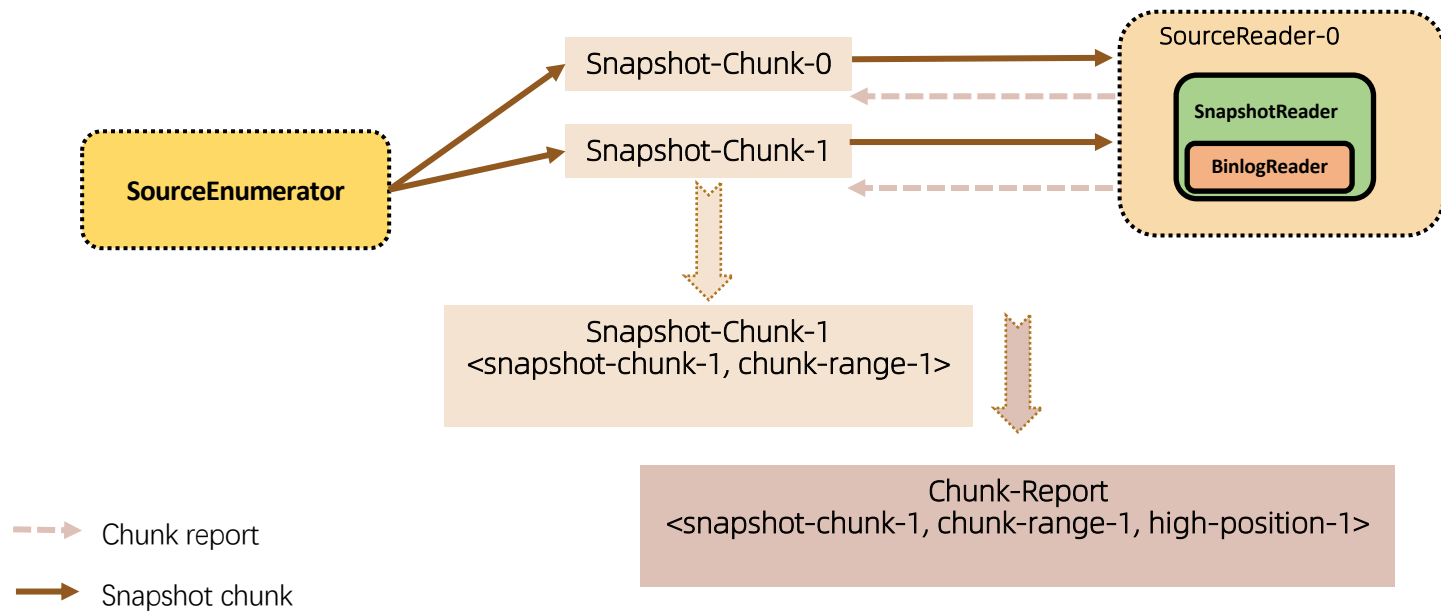
Flink CDC 增量快照算法

Chunk 分发: snapshot chunk



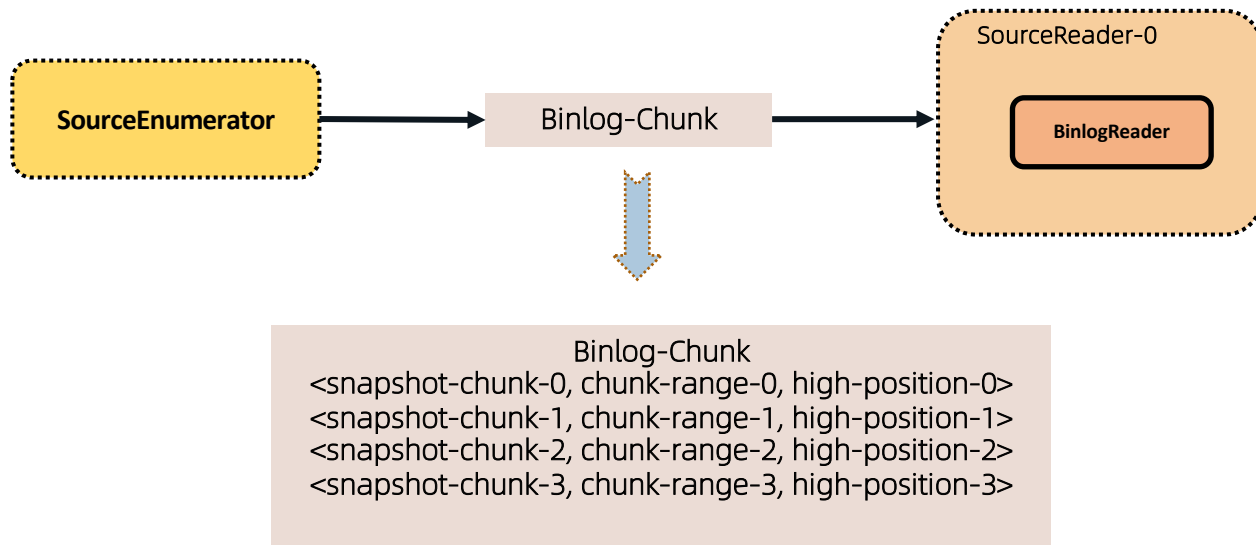
Flink CDC 增量快照算法

Chunk 汇报



Flink CDC 增量快照算法

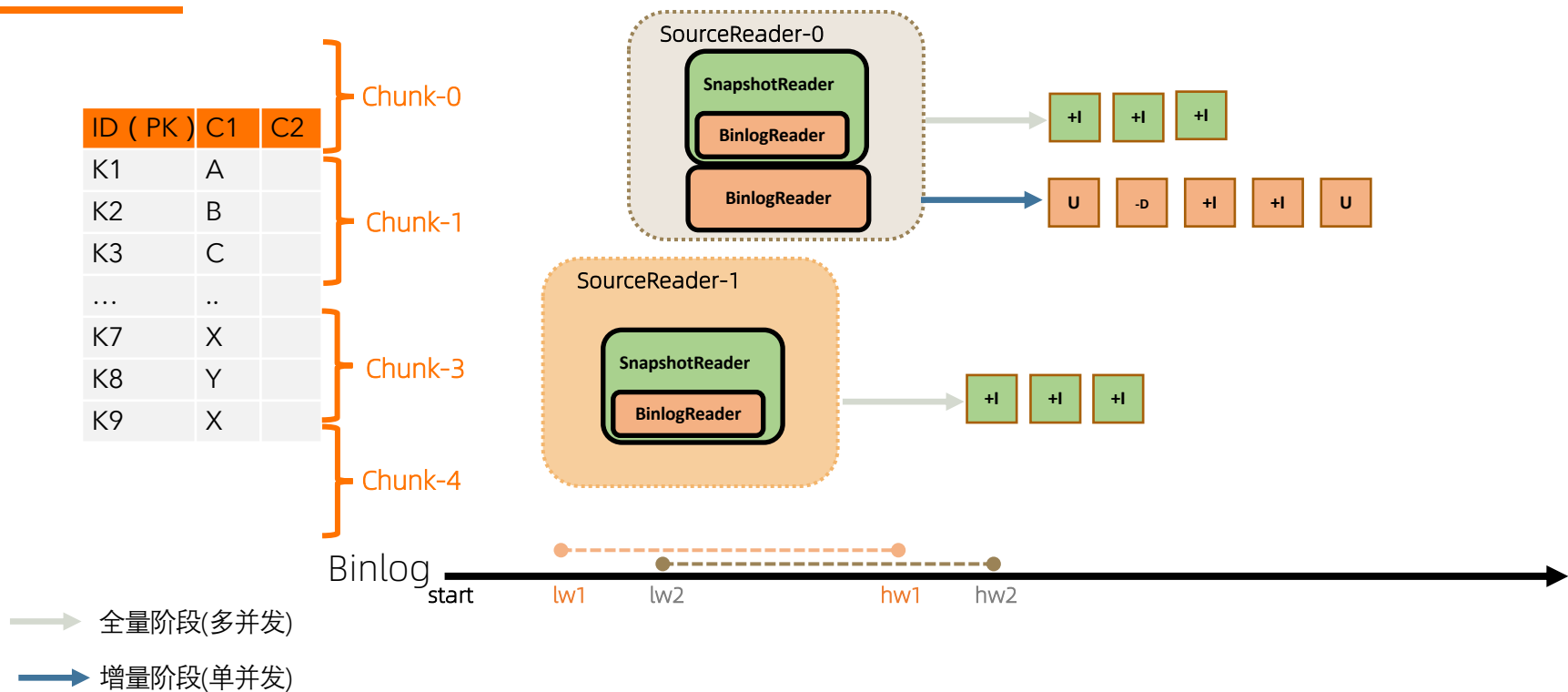
Chunk 分发: binlog chunk



→ binlog chunk

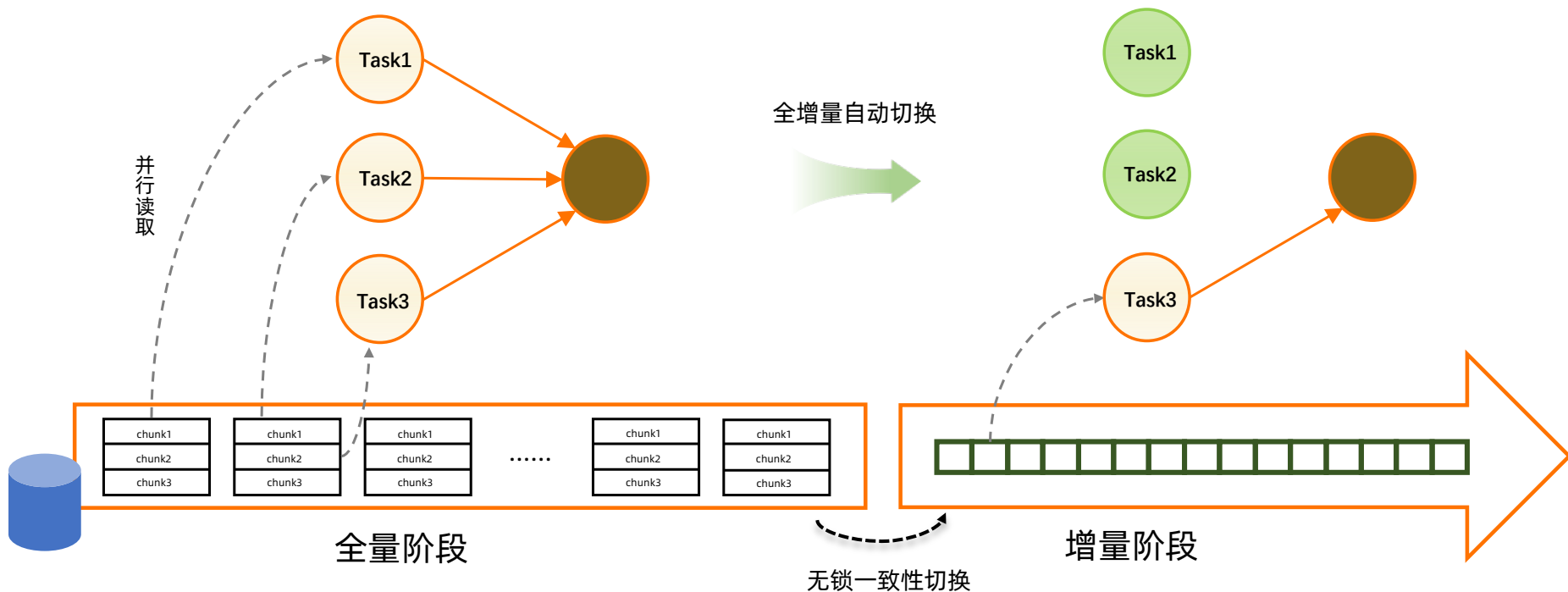
Flink CDC 增量快照算法

算法整体流程



Flink CDC 增量快照算法

并行读取+无锁切换+断点续传 + 全增量一体化



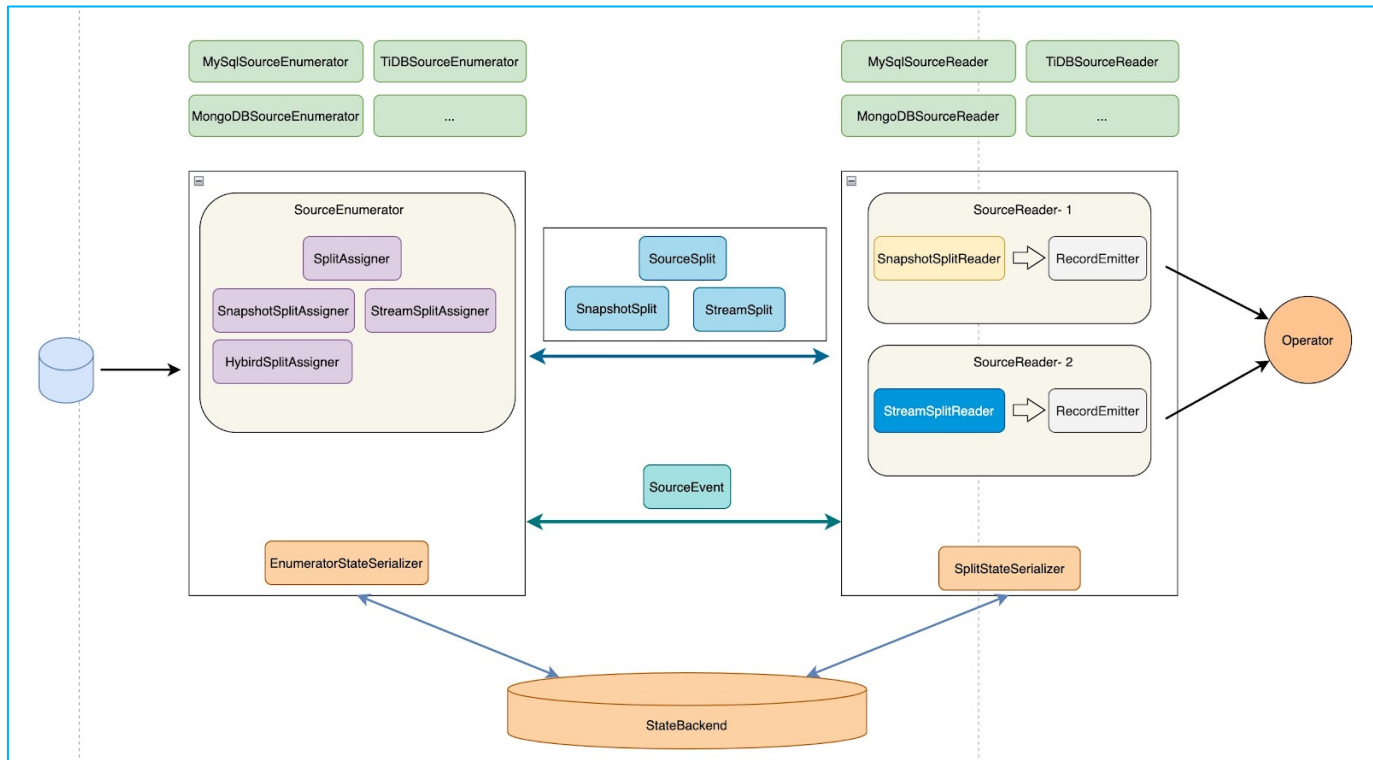
03

Flink CDC 增量快照框架



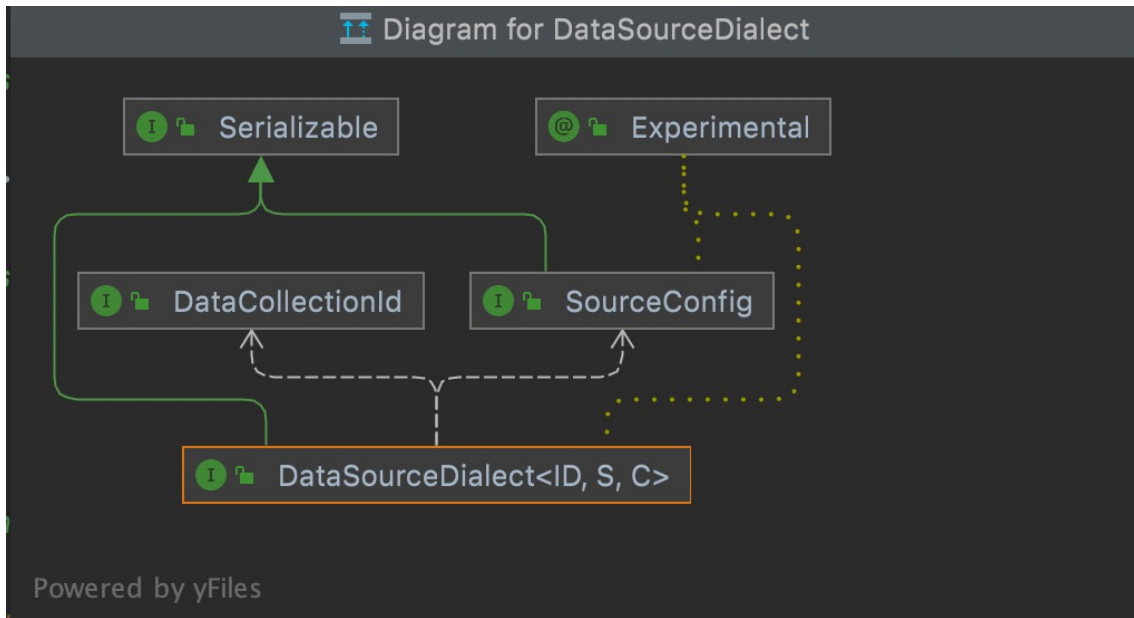
Flink CDC 增量快照框架

框架设计



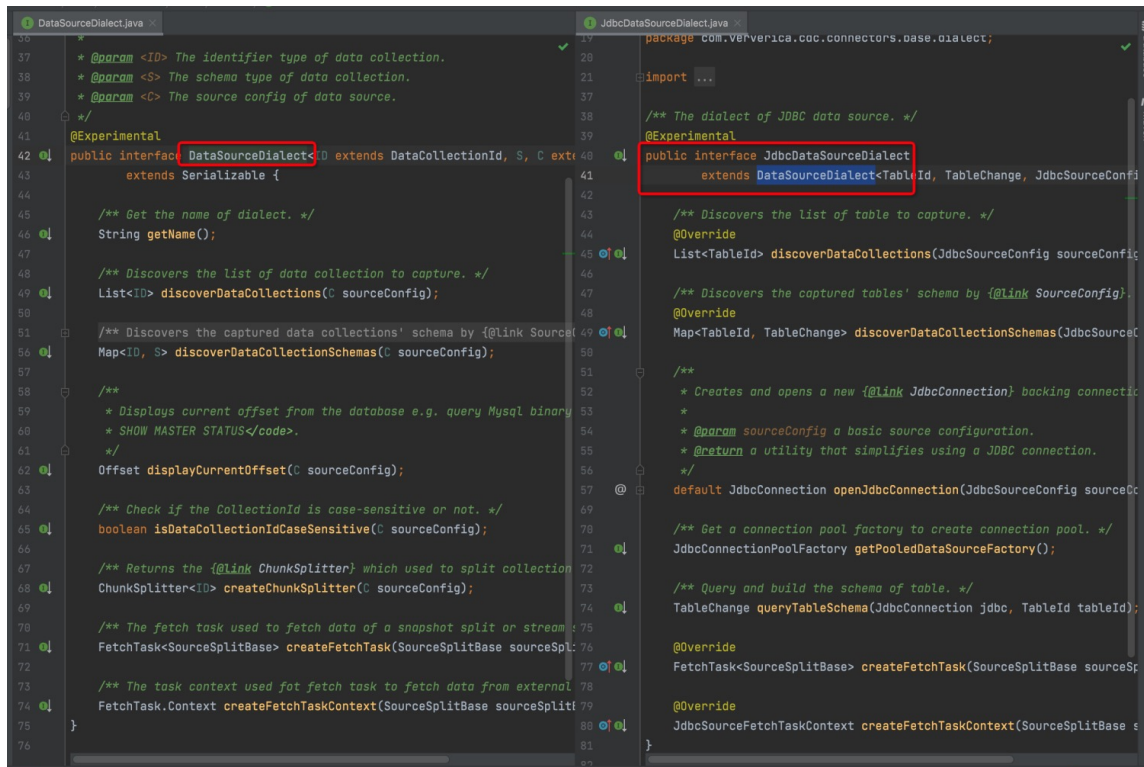
Flink CDC 增量快照框架

核心 API : DataSourceDialect



Flink CDC 增量快照框架

核心 API : JdbcDataSourceDialect

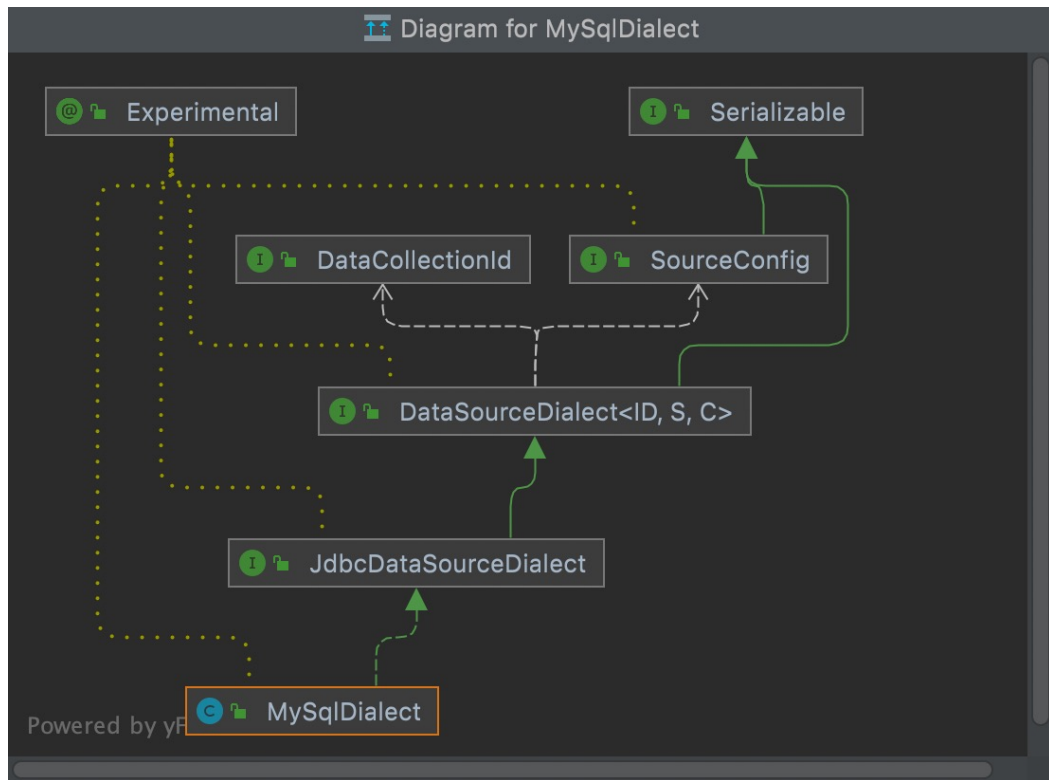


```
DataSourceDialect.java
35  *
36  * @param <ID> The identifier type of data collection.
37  * @param <S> The schema type of data collection.
38  * @param <C> The source config of data source.
39  */
40
41 @Experimental
42 public interface DataSourceDialect<ID extends DataCollectionId, S, C extends Serializable>
43     extends Serializable {
44
45     /** Get the name of dialect. */
46     String getName();
47
48     /** Discovers the list of data collection to capture. */
49     List<ID> discoverDataCollections(C sourceConfig);
50
51     /** Discovers the captured data collections' schema by {@link SourceConfig}. */
52     Map<ID, S> discoverDataCollectionSchemas(C sourceConfig);
53
54     /**
55      * Displays current offset from the database e.g. query Mysql binary
56      * SHOW MASTER STATUS</code>.
57      */
58     Offset displayCurrentOffset(C sourceConfig);
59
60     /** Check if the CollectionId is case-sensitive or not. */
61     boolean isDataCollectionIdCaseSensitive(C sourceConfig);
62
63     /** Returns the {@link ChunkSplitter} which used to split collection */
64     ChunkSplitter<ID> createChunkSplitter(C sourceConfig);
65
66     /** The fetch task used to fetch data of a snapshot split or stream */
67     FetchTask<SourceSplitBase> createFetchTask(SourceSplitBase sourceSplitBase);
68
69     /** The task context used for fetch task to fetch data from external */
70     FetchTask.Context createFetchTaskContext(SourceSplitBase sourceSplitBase);
71 }

JdbcDataSourceDialect.java
17 package com.ververica.cdc.connectors.base.dialect;
18
19 import ...
20
21 /** The dialect of JDBC data source. */
22 @Experimental
23 public interface JdbcDataSourceDialect
24     extends DataSourceDialect<TableId, TableChange, JdbcSourceConfig> {
25
26     /** Discovers the list of table to capture. */
27     @Override
28     List<TableId> discoverDataCollections(JdbcSourceConfig sourceConfig);
29
30     /** Discovers the captured tables' schema by {@link SourceConfig}. */
31     @Override
32     Map<TableId, TableChange> discoverDataCollectionSchemas(JdbcSourceConfig sourceConfig);
33
34     /**
35      * Creates and opens a new {@link JdbcConnection} backing connection.
36      *
37      * @param sourceConfig a basic source configuration.
38      * @return a utility that simplifies using a JDBC connection.
39      */
40     @Default JdbcConnection openJdbcConnection(JdbcSourceConfig sourceConfig);
41
42     /** Get a connection pool factory to create connection pool. */
43     JdbcConnectionPoolFactory getPooledDataSourceFactory();
44
45     /** Query and build the schema of table. */
46     TableChange queryTableSchema(JdbcConnection jdbc, TableId tableId);
47
48     @Override
49     FetchTask<SourceSplitBase> createFetchTask(SourceSplitBase sourceSplitBase);
50
51     @Override
52     JdbcSourceFetchTaskContext createFetchTaskContext(SourceSplitBase sourceSplitBase);
53 }
```

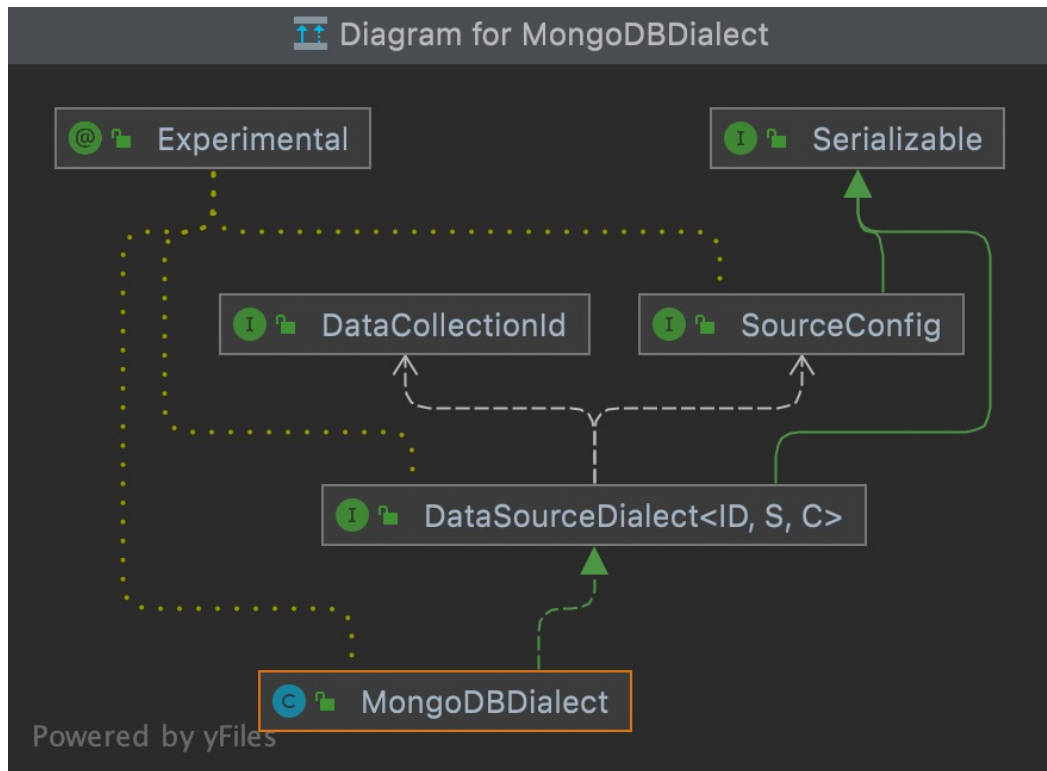
Flink CDC 增量快照框架

JDBC 数据源接入



Flink CDC 增量快照框架

普通数据源接入



Flink CDC 增量快照框架

社区 Connector 接入

- MySQL CDC : [MySQL 接入代码](#)
- MongoDB CDC : [MongoDB 接入PR](#)
- Oracle CDC : [Oracle 接入 PR](#)
- TDSQL CDC : [TDSQL 接入PR](#)

04

社区发展规划



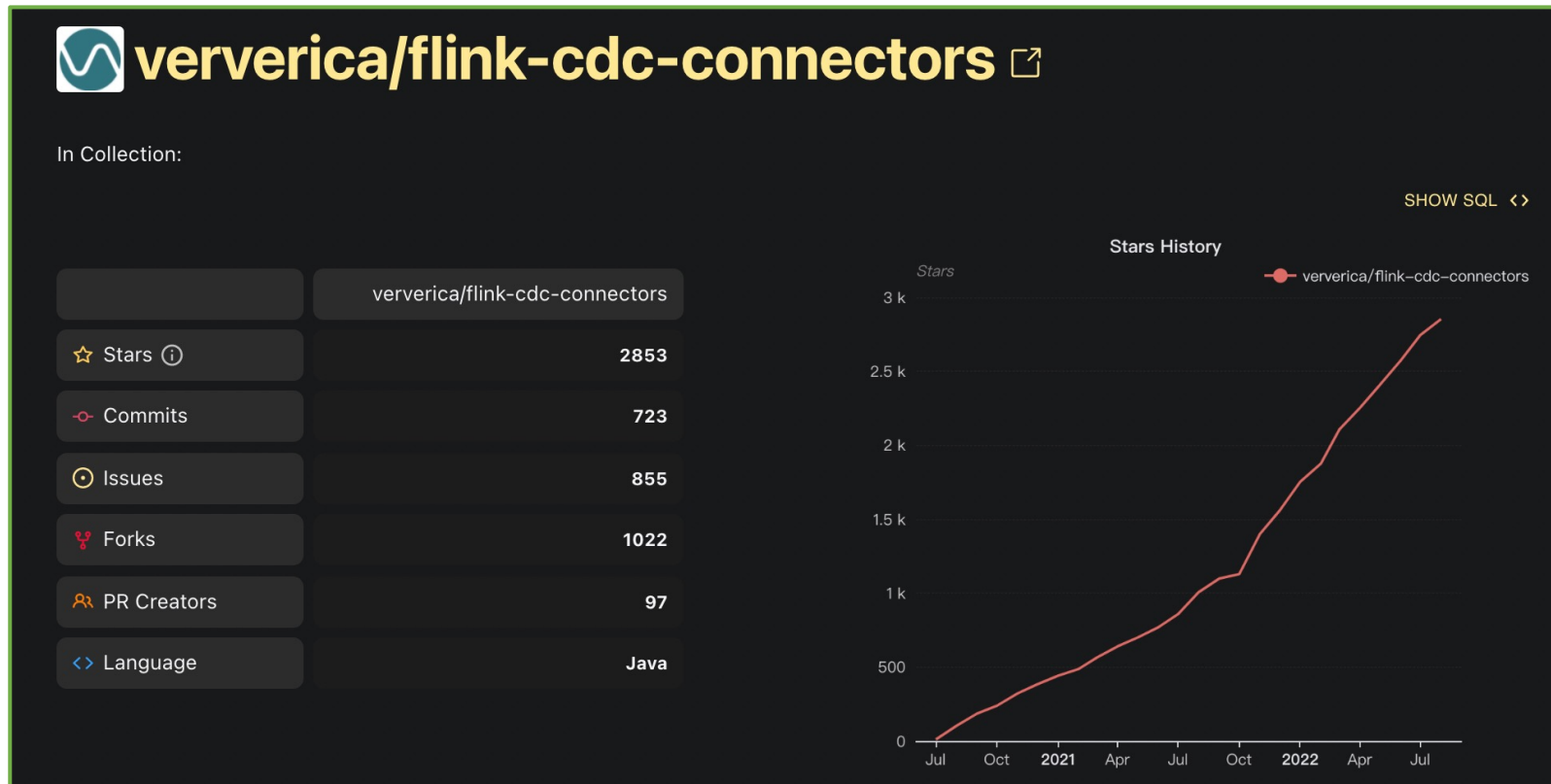
Flink CDC 发展规划

社区历程



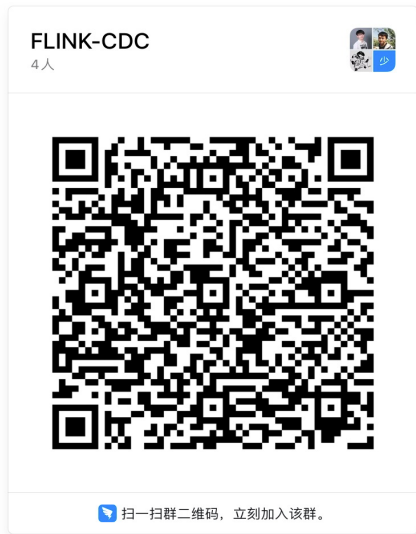
Flink CDC 发展规划

社区指标



Flink CDC 发展规划

社群发展



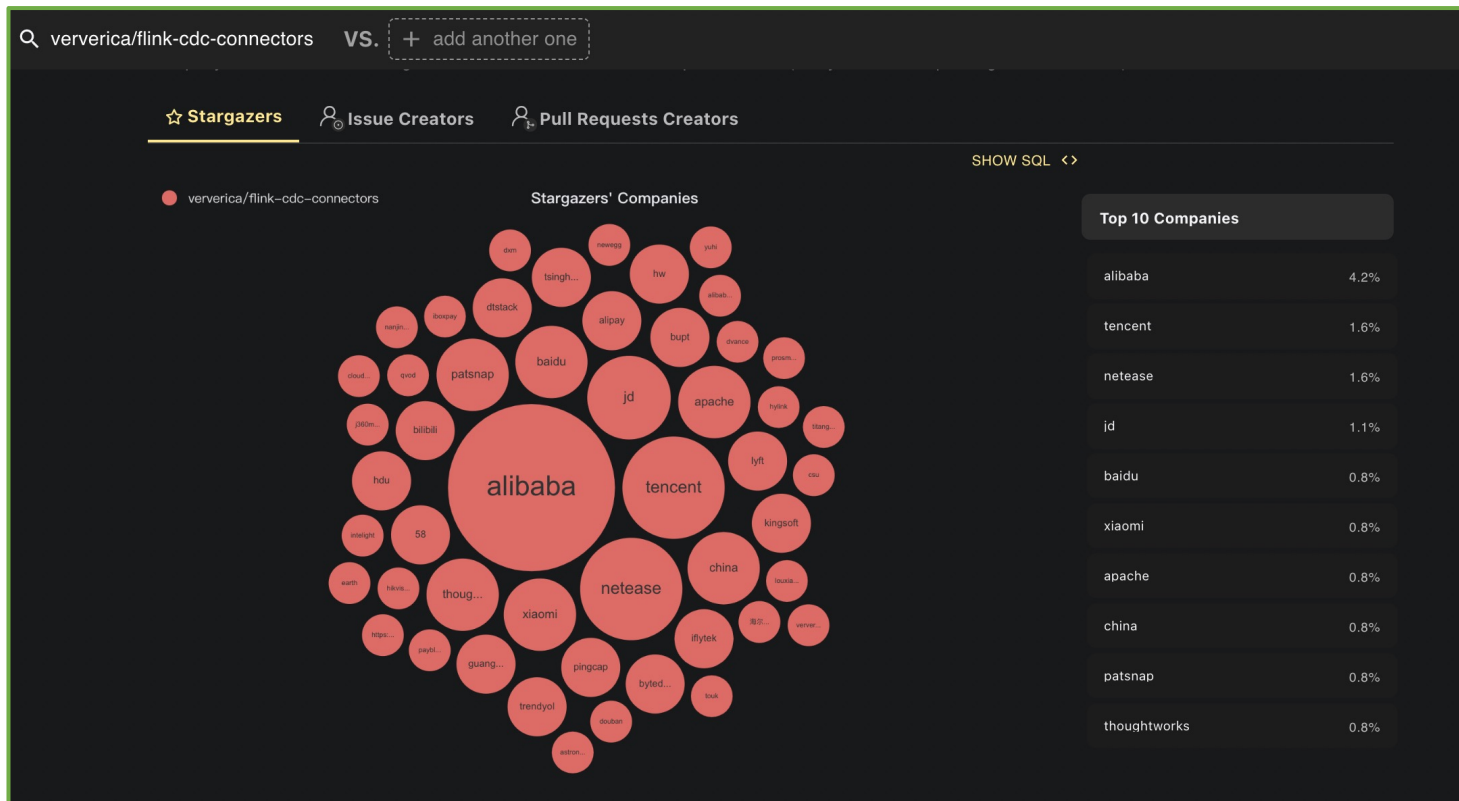
21年7月，我们在北京的 Flink Meetup
建立了 Flink CDC 社区群(4人)



22年8月，Flink CDC 社区群已有
6000+ 开发者和用户

Flink CDC 发展规划

社区成员



未来规划

框架完善

增量快照框架推广

Schema Evolution
/整库同步

生态集成

更多DB

更多版本

数据湖集成

端到端方案

易用性

开箱即用

文档/教程



阿里云实时计算Flink版

实时计算Flink版是阿里云基于 Apache Flink 构建的企业级、高性能实时大数据处理系统，由 Apache Flink 创始团队官方出品，提供全托管Serverless Flink服务，100%兼容开源，并提供扩展性商业功能增值，完整的上下游连接器，即开即用，按需付费。阿里云专家团队持续性内核优化，提供更低成本、更高性能、更安全稳定的实时计算服务。



实时计算Flink版官网



实时计算Flink版交流钉群

非常感谢您的观看

 Alibaba Cloud |  DataFun.

