**尚硅谷大数据技术之Flink-CDC**

(作者：尚硅谷大数据研发组)

版本：V 2.0

# 第1章 CDC简介

## 1.1 什么是CDC

CDC是Change Data Capture(变更数据获取)的简称。核心思想是，监测并捕获数据库的变动（包括数据或数据表的插入、更新以及删除等），将这些变更按发生的顺序完整记录下来，写入到消息中间件中以供其他服务进行订阅及消费。

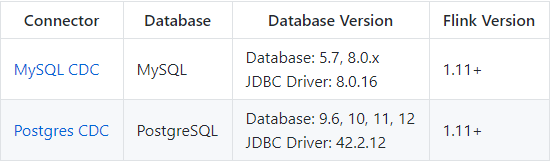
## 1.2 CDC的种类

CDC主要分为基于查询和基于Binlog两种方式，我们主要了解一下这两种之间的区别：

|  |  |  |
| --- | --- | --- |
|  | 基于查询的CDC | 基于Binlog的CDC |
| 开源产品 | Sqoop、Kafka JDBC Source | Canal、Maxwell、Debezium |
| 执行模式 | Batch | Streaming |
| 是否可以捕获所有数据变化 | 否 | 是 |
| 延迟性 | 高延迟 | 低延迟 |
| 是否增加数据库压力 | 是 | 否 |

## 1.3 Flink-CDC

Flink社区开发了 flink-cdc-connectors 组件，这是一个可以直接从 MySQL、PostgreSQL 等数据库直接读取全量数据和增量变更数据的 source 组件。目前也已开源，开源地址：<https://github.com/ververica/flink-cdc-connectors>



# 第2章 FlinkCDC案例实操

## 2.1 DataStream方式的应用

### 2.1.1 导入依赖

<dependencies>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-java</artifactId>

<version>1.12.0</version>

</dependency>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-streaming-java\_2.12</artifactId>

<version>1.12.0</version>

</dependency>

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-clients\_2.12</artifactId>

<version>1.12.0</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>3.1.3</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.49</version>

</dependency>

<dependency>

<groupId>com.alibaba.ververica</groupId>

<artifactId>flink-connector-mysql-cdc</artifactId>

<version>1.2.0</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.75</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-assembly-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

### 2.1.2 编写代码

import com.alibaba.ververica.cdc.connectors.mysql.MySQLSource;

import com.alibaba.ververica.cdc.debezium.DebeziumSourceFunction;

import com.alibaba.ververica.cdc.debezium.StringDebeziumDeserializationSchema;

import org.apache.flink.api.common.restartstrategy.RestartStrategies;

import org.apache.flink.runtime.state.filesystem.FsStateBackend;

import org.apache.flink.streaming.api.CheckpointingMode;

import org.apache.flink.streaming.api.datastream.DataStreamSource;

import org.apache.flink.streaming.api.environment.CheckpointConfig;

import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

import java.util.Properties;

public class FlinkCDC {

public static void main(String[] args) throws Exception {

//1.创建执行环境

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

env.setParallelism(1);

//2.Flink-CDC将读取binlog的位置信息以状态的方式保存在CK,如果想要做到断点续传,需要从Checkpoint或者Savepoint启动程序

//2.1 开启Checkpoint,每隔5秒钟做一次CK

env.enableCheckpointing(5000L);

//2.2 指定CK的一致性语义

env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY\_ONCE);

//2.3 设置任务关闭的时候保留最后一次CK数据

env.getCheckpointConfig().enableExternalizedCheckpoints(CheckpointConfig.ExternalizedCheckpointCleanup.RETAIN\_ON\_CANCELLATION);

//2.4 指定从CK自动重启策略

env.setRestartStrategy(RestartStrategies.fixedDelayRestart(3, 2000L));

//2.5 设置状态后端

env.setStateBackend(new FsStateBackend("hdfs://hadoop102:8020/flinkCDC"));

//2.6 设置访问HDFS的用户名

System.setProperty("HADOOP\_USER\_NAME", "atguigu");

//3.创建Flink-MySQL-CDC的Source

//initial (default): Performs an initial snapshot on the monitored database tables upon first startup, and continue to read the latest binlog.

//latest-offset: Never to perform snapshot on the monitored database tables upon first startup, just read from the end of the binlog which means only have the changes since the connector was started.

//timestamp: Never to perform snapshot on the monitored database tables upon first startup, and directly read binlog from the specified timestamp. The consumer will traverse the binlog from the beginning and ignore change events whose timestamp is smaller than the specified timestamp.

//specific-offset: Never to perform snapshot on the monitored database tables upon first startup, and directly read binlog from the specified offset.

DebeziumSourceFunction<String> mysqlSource = MySQLSource.<String>builder()

.hostname("hadoop102")

.port(3306)

.username("root")

.password("000000")

.databaseList("gmall-flink")

.tableList("gmall-flink.z\_user\_info") //可选配置项,如果不指定该参数,则会读取上一个配置下的所有表的数据，注意：指定的时候需要使用"db.table"的方式

.startupOptions(StartupOptions.initial())

.deserializer(new StringDebeziumDeserializationSchema())

.build();

//4.使用CDC Source从MySQL读取数据

DataStreamSource<String> mysqlDS = env.addSource(mysqlSource);

//5.打印数据

mysqlDS.print();

//6.执行任务

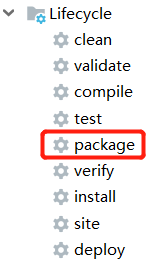
env.execute();

}

}

### 2.1.3 案例测试

1）打包并上传至Linux



2）开启MySQL Binlog并重启MySQL

3）启动Flink集群

[atguigu@hadoop102 flink-standalone]$ bin/start-cluster.sh

4）启动HDFS集群

[atguigu@hadoop102 flink-standalone]$ start-dfs.sh

5）启动程序

[atguigu@hadoop102 flink-standalone]$ bin/flink run -c com.atguigu.FlinkCDC flink-1.0-SNAPSHOT-jar-with-dependencies.jar

6）在MySQL的gmall-flink.z\_user\_info表中添加、修改或者删除数据

7）给当前的Flink程序创建Savepoint

[atguigu@hadoop102 flink-standalone]$ bin/flink savepoint JobId hdfs://hadoop102:8020/flink/save

8）关闭程序以后从Savepoint重启程序

[atguigu@hadoop102 flink-standalone]$ bin/flink run -s hdfs://hadoop102:8020/flink/save/... -c com.atguigu.FlinkCDC flink-1.0-SNAPSHOT-jar-with-dependencies.jar

## 2.2 FlinkSQL方式的应用

### 2.2.1 添加依赖

<dependency>

<groupId>org.apache.flink</groupId>

<artifactId>flink-table-planner-blink\_2.12</artifactId>

<version>1.12.0</version>

</dependency>

### 2.2.2 代码实现

import org.apache.flink.api.common.restartstrategy.RestartStrategies;

import org.apache.flink.runtime.state.filesystem.FsStateBackend;

import org.apache.flink.streaming.api.CheckpointingMode;

import org.apache.flink.streaming.api.environment.CheckpointConfig;

import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;

public class FlinkSQL\_CDC {

public static void main(String[] args) throws Exception {

//1.创建执行环境

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

env.setParallelism(1);

StreamTableEnvironment tableEnv = StreamTableEnvironment.create(env);

//2.创建Flink-MySQL-CDC的Source

tableEnv.executeSql("CREATE TABLE user\_info (" +

" id INT," +

" name STRING," +

" phone\_num STRING" +

") WITH (" +

" 'connector' = 'mysql-cdc'," +

" 'hostname' = 'hadoop102'," +

" 'port' = '3306'," +

" 'username' = 'root'," +

" 'password' = '000000'," +

" 'database-name' = 'gmall-flink'," +

" 'table-name' = 'z\_user\_info'" +

")");

tableEnv.executeSql("select \* from user\_info").print();

env.execute();

}

}

## 2.3 自定义反序列化器

### 2.3.1 代码实现

import com.alibaba.fastjson.JSONObject;

import com.alibaba.ververica.cdc.connectors.mysql.MySQLSource;

import com.alibaba.ververica.cdc.debezium.DebeziumDeserializationSchema;

import com.alibaba.ververica.cdc.debezium.DebeziumSourceFunction;

import io.debezium.data.Envelope;

import org.apache.flink.api.common.restartstrategy.RestartStrategies;

import org.apache.flink.api.common.typeinfo.TypeInformation;

import org.apache.flink.runtime.state.filesystem.FsStateBackend;

import org.apache.flink.streaming.api.CheckpointingMode;

import org.apache.flink.streaming.api.datastream.DataStreamSource;

import org.apache.flink.streaming.api.environment.CheckpointConfig;

import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

import org.apache.flink.util.Collector;

import org.apache.kafka.connect.data.Field;

import org.apache.kafka.connect.data.Struct;

import org.apache.kafka.connect.source.SourceRecord;

import java.util.Properties;

public class Flink\_CDCWithCustomerSchema {

public static void main(String[] args) throws Exception {

//1.创建执行环境

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

env.setParallelism(1);

//2.创建Flink-MySQL-CDC的Source

Properties properties = new Properties();

//initial (default): Performs an initial snapshot on the monitored database tables upon first startup, and continue to read the latest binlog.

//latest-offset: Never to perform snapshot on the monitored database tables upon first startup, just read from the end of the binlog which means only have the changes since the connector was started.

//timestamp: Never to perform snapshot on the monitored database tables upon first startup, and directly read binlog from the specified timestamp. The consumer will traverse the binlog from the beginning and ignore change events whose timestamp is smaller than the specified timestamp.

//specific-offset: Never to perform snapshot on the monitored database tables upon first startup, and directly read binlog from the specified offset.

DebeziumSourceFunction<String> mysqlSource = MySQLSource.<String>builder()

.hostname("hadoop102")

.port(3306)

.username("root")

.password("000000")

.databaseList("gmall-flink")

.tableList("gmall-flink.z\_user\_info") //可选配置项,如果不指定该参数,则会读取上一个配置下的所有表的数据,注意：指定的时候需要使用"db.table"的方式

.startupOptions(StartupOptions.initial())

.deserializer(new DebeziumDeserializationSchema<String>() { //自定义数据解析器

@Override

public void deserialize(SourceRecord sourceRecord, Collector<String> collector) throws Exception {

//获取主题信息,包含着数据库和表名 mysql\_binlog\_source.gmall-flink.z\_user\_info

String topic = sourceRecord.topic();

String[] arr = topic.split("\\.");

String db = arr[1];

String tableName = arr[2];

//获取操作类型 READ DELETE UPDATE CREATE

Envelope.Operation operation = Envelope.operationFor(sourceRecord);

//获取值信息并转换为Struct类型

Struct value = (Struct) sourceRecord.value();

//获取变化后的数据

Struct after = value.getStruct("after");

//创建JSON对象用于存储数据信息

JSONObject data = new JSONObject();

for (Field field : after.schema().fields()) {

Object o = after.get(field);

data.put(field.name(), o);

}

//创建JSON对象用于封装最终返回值数据信息

JSONObject result = new JSONObject();

result.put("operation", operation.toString().toLowerCase());

result.put("data", data);

result.put("database", db);

result.put("table", tableName);

//发送数据至下游

collector.collect(result.toJSONString());

}

@Override

public TypeInformation<String> getProducedType() {

return TypeInformation.of(String.class);

}

})

.build();

//3.使用CDC Source从MySQL读取数据

DataStreamSource<String> mysqlDS = env.addSource(mysqlSource);

//4.打印数据

mysqlDS.print();

//5.执行任务

env.execute();

}

}