尚硅谷大数据技术之Flink-CDC

(作者：尚硅谷大数据研发部)

版本：V1.0

# CDC简介

## 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。

## 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：

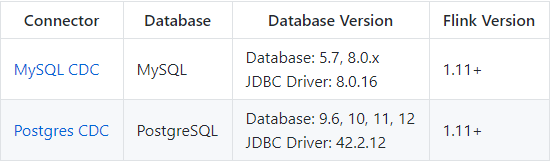
|  |  |  |
| --- | --- | --- |
|  | 基于查询的CDC | 基于Binlog的CDC |
| 开源产品 | Sqoop、Kafka JDBC Source | Canal、Maxwell、Debezium |
| 执行模式 | Batch | Streaming |
| 是否可以捕获所有数据变化 | 否 | 是 |
| 延迟性 | 高延迟 | 低延迟 |
| 是否增加数据库压力 | 是 | 否 |

## 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，

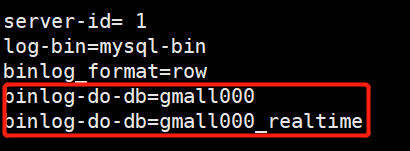
<http://www.dreamwu.com/post-1594.html>

开源地址：<https://github.com/ververica/flink-cdc-connectors>



# FlinkCDC案例实操

## 开启MySQL Binlog并重启MySQL



## DataStream方式的应用

### 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**>5.1.48</**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**>

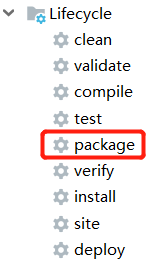
### 编写代码

**package** com.atguigu.cdc;

**import** com.alibaba.ververica.cdc.connectors.mysql.MySQLSource;  
**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** org.apache.flink.streaming.api.functions.source.SourceFunction;  
  
**import** java.util.Properties;  
  
*/\*\*  
 \* Author: Felix  
 \* Desc: Flink-CDC-DS  
 \*/***public class** FlinkCDC\_01\_DS {  
 **public static void** main(String[] args) **throws** Exception {  
 *//****TODO 1.准备流处理环境*** StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setParallelism(1);  
  
 *//****TODO 2.开启检查点 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 ,并指定CK的一致性语义* env.enableCheckpointing(5000L, CheckpointingMode.***EXACTLY\_ONCE***);  
 *//2.2 设置超时时间为1分钟* env.getCheckpointConfig().setCheckpointTimeout(60000);  
 *//2.3 指定从CK自动重启策略* env.setRestartStrategy(RestartStrategies.*fixedDelayRestart*(2,2000L));  
 *//2.4 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***);  
 *//2.5 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");  
  
 //****TODO 3.创建Flink-MySQL-CDC的Source*** Properties props = **new** Properties();  
 props.setProperty(**"scan.startup.mode"**,**"initial"**);  
 SourceFunction<String> sourceFunction = MySQLSource.<String>*builder*()  
 .hostname(**"hadoop102"**)  
 .port(3306)  
 .username(**"root"**)  
 .password(**"123456"**)  
 .databaseList(**"gmall000\_realtime"**)  
 *///可选配置项,如果不指定该参数,则会读取上一个配置中指定的数据库下的所有表的数据  
 //注意：指定的时候需要使用"db.table"的方式* .tableList(**"gmall000\_realtime.t\_user"**)  
 .debeziumProperties(props)  
 .deserializer(**new** StringDebeziumDeserializationSchema()).build();  
  
 *//****TODO 4.使用CDC Source从MySQL读取数据*** DataStreamSource<String> mysqlDS = env.addSource(sourceFunction);  
   
 *//****TODO 5.打印输出*** mysqlDS.print();  
  
 *//****TODO 6.执行任务*** env.execute();  
 }  
}

### 案例测试

1. 打包并上传至Linux



1. 启动HDFS集群

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

1. 启动Flink集群

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

1. 启动程序

[atguigu@hadoop102 flink-local]$ bin/flink run -m hadoop102:8081 -c com.atguigu.cdc.FlinkCDC\_01\_DS ./gmall000-flink-cdc.jar

1. 观察taskManager日志，会从头读取表数据
2. 给当前的Flink程序创建Savepoint

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

1. 在WebUI中cancelJob
2. 在MySQL的**gmall000\_realtime.t\_user**表中添加、修改或者删除数据
3. 从Savepoint重启程序

[atguigu@hadoop102 flink-standalone]$ bin/flink run -s hdfs://hadoop102:8020/flink/save/... -c com.atguigu.cdc.FlinkCDC\_01\_DS ./gmall000-flink-cdc.jar

1. 观察taskManager日志，会从检查点读取表数据

## FlinkSQL方式的应用

### 添加依赖

<**dependency**>

<**groupId**>org.apache.flink</**groupId**>  
 <**artifactId**>flink-table-planner-blink\_2.12</**artifactId**>  
 <**version**>1.12.0</**version**>  
</**dependency**>

### 代码实现

**package** com.atguigu.cdc;

**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
**import** org.apache.flink.table.api.bridge.java.StreamTableEnvironment;  
  
*/\*\*  
 \* Author: Felix  
 \* Desc: Flink-CDC-SQL  
 \*/***public class** FlinkCDC\_02\_SQL {  
 **public static void** main(String[] args) **throws** Exception {  
 *//****TODO 1.准备环境*** *//1.1流处理环境* StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setParallelism(1);  
 *//1.2 表执行环境* StreamTableEnvironment tableEnv = StreamTableEnvironment.*create*(env);  
  
 *//****TODO 2.创建动态表*** tableEnv.executeSql(**"CREATE TABLE user\_info ("** +  
 **" id INT,"** +  
 **" name STRING,"** +  
 **" age INT"** +  
 **") WITH ("** +  
 **" 'connector' = 'mysql-cdc',"** +  
 **" 'hostname' = 'hadoop102',"** +  
 **" 'port' = '3306',"** +  
 **" 'username' = 'root',"** +  
 **" 'password' = '123456',"** +  
 **" 'database-name' = 'gmall000\_realtime',"** +  
 **" 'table-name' = 't\_user'"** +  
 **")"**);  
  
 tableEnv.executeSql(**"select \* from user\_info"**).print();  
  
 *//****TODO 6.执行任务*** env.execute();  
 }  
}

## 自定义反序列化器

### 代码实现

**package** com.atguigu.cdc;

**import** com.alibaba.fastjson.JSONObject;  
**import** com.alibaba.ververica.cdc.connectors.mysql.MySQLSource;  
**import** com.alibaba.ververica.cdc.debezium.DebeziumDeserializationSchema;  
**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.streaming.api.functions.source.SourceFunction;  
**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;  
  
*/\*\*  
 \* Author: Felix  
 \* Desc: 自定义序列化方式  
 \*/***public class** FlinkCDC\_03\_CustomSchema {  
 **public static void** main(String[] args) **throws** Exception {  
 *//****TODO 1.准备流处理环境*** StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setParallelism(1);  
  
 *//****TODO 2.开启检查点 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 ,并指定CK的一致性语义* env.enableCheckpointing(5000L, CheckpointingMode.***EXACTLY\_ONCE***);  
 *//2.2 设置超时时间为1分钟* env.getCheckpointConfig().setCheckpointTimeout(60000);  
 *//2.3 指定从CK自动重启策略* env.setRestartStrategy(RestartStrategies.*fixedDelayRestart*(2, 2000L));  
 *//2.4 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***);  
 *//2.5 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");  
  
 //****TODO 3.创建Flink-MySQL-CDC的Source*** Properties props = **new** Properties();  
 props.setProperty(**"scan.startup.mode"**, **"initial"**);  
 SourceFunction<String> sourceFunction = MySQLSource.<String>*builder*()  
 .hostname(**"hadoop102"**)  
 .port(3306)  
 .username(**"root"**)  
 .password(**"123456"**)  
 .databaseList(**"gmall000\_realtime"**)  
 *///可选配置项,如果不指定该参数,则会读取上一个配置中指定的数据库下的所有表的数据  
 //注意：指定的时候需要使用"db.table"的方式* .tableList(**"gmall000\_realtime.t\_user"**)  
 .debeziumProperties(props)  
 .deserializer(**new** MySchema())  
 .build();  
  
 *//****TODO 4.使用CDC Source从MySQL读取数据*** DataStreamSource<String> mysqlDS = env.addSource(sourceFunction);  
  
 *//****TODO 5.打印输出*** mysqlDS.print();  
  
 *//****TODO 6.执行任务*** env.execute();  
 }  
  
 **public static class** MySchema **implements** DebeziumDeserializationSchema<String> {  
 @Override  
 **public void** deserialize(SourceRecord sourceRecord, Collector<String> collector) **throws** Exception {  
 *//mysql\_binlog\_source.gmall000\_realtime.t\_user* String topic = sourceRecord.topic();  
 String[] topicArr = topic.split(**"\\."**);  
 String dbName = topicArr[1];  
 String tableName = topicArr[2];  
  
 *//获取操作类型* Envelope.Operation operation = Envelope.*operationFor*(sourceRecord);  
  
 *//获取变更的数据 value=Struct{after=Struct{id=3,name=ww11,age=55666}* Struct valueStruct = (Struct) sourceRecord.value();  
 Struct afterStruct = valueStruct.getStruct(**"after"**);  
 *//将变更数据封装为一个json对象* JSONObject dataJsonObj = **new** JSONObject();  
  
 **if** (afterStruct != **null**) {  
 **for** (Field field : afterStruct.schema().fields()) {  
 Object o = afterStruct.get(field);  
 dataJsonObj.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"**, dataJsonObj);  
 result.put(**"database"**, dbName);  
 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**);  
 }  
 }  
}