

题目2：RocketMQ-streams Demo 开发

在新版本的RocketMQ（即RocketMQ5.0版本）中进行了架构重塑，新增或者修改了超过 60% 的代码，但是对 4.0 的所有功能以及整体架构进行了无缝兼容，且没有引入任何外部依赖。而且其中非常重要的一点是，RocketMQ 兼容了开源 Flink 生态。RocketMQ 直接实现了 Flink 的基础功能或者算子，并首创性地兼容了 Flink/Blink SQL 标准以及 UDF/UDAF/UDTF。

功能上 Flink 的基础功能或者算子如 Source、Reduce、Sink、map、flatmap、split、select、window、union rocketmq-streams 都做了实现，当然也包括一些基础的核心特点：Exactly-once，灵活窗口（滚动，滑动，会话），双流 Join，高吞吐、低延迟、高性能等。



rocketmq-streams 的设计之初主要是为了解决用户在使用 RocketMQ 过程中面临的一些轻量级计算的问题，在实时计算领域，社区一直以来的目标都是积极的与 Flink、Spark 等实时计算引擎通过生态合作的方式满足用户诉求。因此，rocketmq-streams 并非要做一个和 Flink 同质的大数据计算引擎，这个引擎场景很明确，主要是满足大数据量 -> 高过滤 -> 轻窗口计算的场景，重点要打造轻量化、高性能和低成本等优势。

接下来，将要部分算子进行简单的分析。

1. union算子：在做数据同步的时候，多个相同结构的源，通过这个union变成一个流，然后执行 process及sink。

示例代码如下，从RocketMQ两个topic接收canal解析的binlog数据，然后简单转成String后，合并成一个流。

```

1 public class UnionDemo {
2     private static Logger logger = Logger.getLogger(UnionDemo.class);
3
4     public static void main(String[] args) {
5         try {
6             // 1.初始化两个数据源
7             StreamExecutionEnvironment env1 = StreamExecutionEnvironment.getExecutionEnvironment();
8
9             Properties consumerProps1 = new Properties();
10            consumerProps1.setProperty(RocketMQConfig.NAME_SERVER_ADDR, Constant.SOURCE_NAME_SERVER_ADDR);
11            consumerProps1.setProperty(RocketMQConfig.CONSUMER_GROUP, "flink_demo");
12            consumerProps1.setProperty(RocketMQConfig.CONSUMER_TOPIC, "BinLogFromCanal");
13
14            Properties consumerProps2 = new Properties();
15            consumerProps2.setProperty(RocketMQConfig.NAME_SERVER_ADDR, Constant.SOURCE_NAME_SERVER_ADDR);
16            consumerProps2.setProperty(RocketMQConfig.CONSUMER_GROUP, "flink_demo");
17            consumerProps2.setProperty(RocketMQConfig.CONSUMER_TOPIC, "MsgFromOds");
18
19            // 2.初始化数据源,对数据源进行映射,过滤,根据表名分成多个侧数据流
20            DataStream<String> dataStream1 = env1
21                .addSource(new RocketMQSource(
22                    new SimpleKeyValueDeserializationSchema(Constant.MQ_CONSTANT_ID, Constant.MQ_CONSTANT_ADDRESS),
23                    consumerProps1))
24                .name("source1").setParallelism(1)
25                .map(new MapFunction<Map<String, String>, String>() {
26                    @Override
27                    public String map(Map<String, String> value) throws Exception
28                    {
29                        StringBuffer str = new StringBuffer("source1:");
30                        BinLogMsgEntity msgEntity = JSON.parseObject(value.get(Constant.MQ_CONSTANT_ADDRESS),
31                            new TypeReference<BinLogMsgEntity>() {});
32                        str.append(msgEntity.getDatabase());
33                        return str.toString();
34                    }
35                });
36            DataStream<String> dataStream2 = env1
37                .addSource(new RocketMQSource(

```

```

38         new SimpleKeyValueDeserializationSchema(Constant.MQ_CONSTANT_
39 ID, Constant.MQ_CONSTANT_ADDRESS),
40         consumerProps2))
41         .name("source2").setParallelism(1)
42         .map(new MapFunction<Map<String, String>, String>() {
43             @Override
44             public String map(Map<String, String> value) throws Exception
45 {
46                 StringBuffer str = new StringBuffer("source2:");
47                 BinLogMsgEntity msgEntity = JSON.parseObject(value.get(Co
48 nstant.MQ_CONSTANT_ADDRESS),
49                     new TypeReference<BinLogMsgEntity>() {}));
50                 str.append(msgEntity.getDatabase());
51                 return str.toString();
52             }
53         });

54         dataStream2.print();
55         dataStream1.print();
56         //3.将两个数据流合并
57         DataStream<String> unionStream = dataStream2.union(dataStream1).map(
58             new MapFunction<String, String>() {
59                 @Override
60                 public String map(String value) throws Exception {
61                     String str = value.replaceAll("source1", "unionSource").repla
62 ceAll("source2", "unionSource");
63                     return str;
64                 }
65             });
66         //4.打印
67         unionStream.print();
68         //执行数据流
69         env1.execute("geekplus_dws_etl_job1");
70     } catch (Exception e) {
71         e.printStackTrace();
72         logger.error("error:" + e.getMessage());
73     }
74 }
75 }

```

2. Reduce算子：将新流入的数据，和最后一个reduce计算后的值进行计算，生成一个新值。

```

1 public class ReduceDemo {
2     private static Logger logger = Logger.getLogger(ReduceDemo.class);
3
4     public static void main(String[] args) {
5         try {
6             // 1.初始化数据源
7             StreamExecutionEnvironment env1 = StreamExecutionEnvironment.getExecutionEnvironment();
8
9             Properties consumerProps1 = new Properties();
10            consumerProps1.setProperty(RocketMQConfig.NAME_SERVER_ADDR, Constant.SOURCE_NAME_SERVER_ADDR);
11            consumerProps1.setProperty(RocketMQConfig.CONSUMER_GROUP, "flink_demo");
12            consumerProps1.setProperty(RocketMQConfig.CONSUMER_TOPIC, "BinLogFromCanal");
13
14            // 2.初始化数据源,对数据源进行映射,过滤,根据表名分成多个侧数据流
15            DataStream<BinLogMsgEntity> dataStream1 = env1
16                .addSource(new RocketMQSource(
17                    new SimpleKeyValueDeserializationSchema(Constant.MQ_CONSTANT_ID, Constant.MQ_CONSTANT_ADDRESS),
18                    consumerProps1))
19                .name("source1").setParallelism(1)
20                .map(new MapFunction<Map<String, String>, BinLogMsgEntity>() {
21                    @Override
22                    public BinLogMsgEntity map(Map<String, String> value) throws
23                        Exception {
24                        BinLogMsgEntity msgEntity = JSON.parseObject(value.get(Constant.MQ_CONSTANT_ADDRESS),
25                            new TypeReference<BinLogMsgEntity>() {});
26                        return msgEntity;
27                    }
28                })
29                .filter(new FilterFunction<BinLogMsgEntity>() {
30                    @Override
31                    public boolean filter(BinLogMsgEntity value) throws Exception
32                    {
33                        if(null == value.getData() ) {
34                            return false;
35                        }else {
36                            return true;
37                        }
38                    }
39                })
40                .keyBy("table").timeWindow(Time.seconds(5))

```

```

37         .reduce(new ReduceFunction<BinLogMsgEntity>() {
38
39             @Override
40             public BinLogMsgEntity reduce(BinLogMsgEntity value1, BinLogM
41 sgEntity value2) throws Exception {
42                 System.out.println("-----redceFunction:tab
43 lie1:{" + value1.getTable() + "},tablie2:{" + value2.getTable() + "}. "
44                 + "size1:{" + value1.getData().size() + "},size2:{" + value
45 2.getData().size() + "}. "
46                 + "type1:{" + value1.getType() + "},type2:{" + value2.getTy
47 pe() + "}" + "-----");
48                 return value1.getData().size() > value2.getData().size() ?
49 value1 : value2 ;
50             });
51
52         dataStream1.print();
53
54         // 执行数据流
55         env1.execute("geekplus_dws_etl_job1");
56     } catch (Exception e) {
57         e.printStackTrace();
58         logger.error("error:" + e.getMessage());
59     }
60 }
61 }

```

3. split算子与select算子：split算子用于将一个数据流拆分成两个或多个数据流，但是，我们如何拿到切出来的那个数据流，这就需要用到select算子。

```

1 public class SplitDemo {
2     private static Logger logger = Logger.getLogger(SplitDemo.class);
3
4
5     public static void main(String[] args) {
6         try {
7             // 1.加载数据源参数
8             StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
9             Properties consumerProps = new Properties();
10            consumerProps.setProperty(RocketMQConfig.NAME_SERVER_ADDR, Constant.SOURCE_NAME_SERVER_ADDR);
11            consumerProps.setProperty(RocketMQConfig.CONSUMER_GROUP, "flink_demo"
12 );
13            consumerProps.setProperty(RocketMQConfig.CONSUMER_TOPIC, "BinLogFromC
14 anal");
15
16            // 2.初始化数据源,对数据源进行映射,过滤,根据表名分割成多个数据流
17            SplitStream<BinLogMsgEntity> splitStream = env
18                .addSource(new RocketMQSource(
19                    new SimpleKeyValueDeserializationSchema(Constant.MQ_CONSTANT_
20 ID, Constant.MQ_CONSTANT_ADDRESS),
21                    consumerProps))
22                .name(Constant.FLINK_SOURCE_NAME).setParallelism(1)
23                .map(new MapFunction<Map<String, String>, BinLogMsgEntity>() {
24                    @Override
25                    public BinLogMsgEntity map(Map<String, String> value) throws
26 Exception {
27                        BinLogMsgEntity msgEntity = JSON.parseObject(value.get(Co
28 nstant.MQ_CONSTANT_ADDRESS),
29                            new TypeReference<BinLogMsgEntity>() {});
30                        return msgEntity;
31                    }
32                }).split(new OutputSelector<BinLogMsgEntity>() {
33
34                    @Override
35                    public Iterable<String> select(BinLogMsgEntity value) {
36                        List<String> output = new ArrayList<String>();
37                        if(value.getTable().equals("out_order")) {
38                            output.add("out_order");
39                        }else if(value.getTable().equals("out_order_details")) {
40                            output.add("out_order_details");
41                        }else {
42                            output.add("other");
43                        }
44                        return output;
45                    }
46                });
47        } catch (Exception e) {
48            logger.error("SplitStream error", e);
49        }
50    }
51 }

```

```

41         });
42     });

43     DataStream<String> outOrderStream = splitStream.select("out_order").m
44 ap(new MapFunction<BinLogMsgEntity, String>() {
45
46         @Override
47         public String map(BinLogMsgEntity value) throws Exception {
48             return "out_order:"+value.getEs();
49         });
50     DataStream<String> outOrderDetailStream = splitStream.select("out_ord
51 er_details").map(new MapFunction<BinLogMsgEntity, String>() {
52
53         @Override
54         public String map(BinLogMsgEntity value) throws Exception {
55             return "out_order_details:"+value.getEs();
56         });
57     DataStream<String> otherStream = splitStream.select("other").map(new
58 MapFunction<BinLogMsgEntity, String>() {
59
60         @Override
61         public String map(BinLogMsgEntity value) throws Exception {
62             return "other:"+value.getTable()+"——"+value.getEs();
63         });
64     // 4.对分割出来的数据流进行打印
65     outOrderStream.print();
66     outOrderDetailStream.print();
67     otherStream.print();
68
69     env.execute("geekplus_dws_etl_job");
70 } catch (Exception e) {
71     e.printStackTrace();
72     logger.error("error:" + e.getMessage());
73 }
74 }
75 }

```

