

Flink Table & SQL 用户自定义函数: UDF、UDAF、UDTF

本文总结Flink Table & SQL中的用户自定义函数: UDF、UDAF、UDTF。

1. UDF: 自定义标量函数(User Defined Scalar Function)。一行输入一行输出。
2. UDAF: 自定义聚合函数。多行输入一行输出。
3. UDTF: 自定义表函数。一行输入多行输出或一行输入多列输出。

测试数据

```
1 // 某个用户在某个时刻浏览了某个商品, 以及商品的价值
2 // eventTime: 北京时间, 方便测试。如下, 乱序数据:
3 {"userID": "user_5", "eventTime": "2019-12-01 10:02:00", "eventType": "b
4 rowse", "productID": "product_5", "productPrice": 20}
5 {"userID": "user_4", "eventTime": "2019-12-01 10:02:02", "eventType": "b
6 rowse", "productID": "product_5", "productPrice": 20}
7 {"userID": "user_5", "eventTime": "2019-12-01 10:02:06", "eventType": "b
8 rowse", "productID": "product_5", "productPrice": 20}
9 {"userID": "user_4", "eventTime": "2019-12-01 10:02:10", "eventType": "b
10 rowse", "productID": "product_5", "productPrice": 20}
11 {"userID": "user_5", "eventTime": "2019-12-01 10:02:06", "eventType": "b
12 rowse", "productID": "product_5", "productPrice": 20}
13 {"userID": "user_5", "eventTime": "2019-12-01 10:02:06", "eventType": "b
rowse", "productID": "product_5", "productPrice": 20}
{"userID": "user_4", "eventTime": "2019-12-01 10:02:12", "eventType": "b
rowse", "productID": "product_5", "productPrice": 20}
{"userID": "user_5", "eventTime": "2019-12-01 10:02:06", "eventType": "b
rowse", "productID": "product_5", "productPrice": 20}
{"userID": "user_5", "eventTime": "2019-12-01 10:02:06", "eventType": "b
rowse", "productID": "product_5", "productPrice": 20}
{"userID": "user_4", "eventTime": "2019-12-01 10:02:15", "eventType": "b
rowse", "productID": "product_5", "productPrice": 20}
{"userID": "user_4", "eventTime": "2019-12-01 10:02:16", "eventType": "b
rowse", "productID": "product_5", "productPrice": 20}
```

UDF时间转换

UDF需要继承 `ScalarFunction` 抽象类，主要实现eval方法。

自定义UDF，实现将Flink Window Start/End Timestamp类型时间转换为指定时区时间。

示例

```
1 package com.bigdata.flink.tableSqlUDF.udf;
2
3 import com.alibaba.fastjson.JSON;
4 import com.bigdata.flink.beans.table.UserBrowseLog;
5 import lombok.extern.slf4j.Slf4j;
6 import org.apache.flink.api.common.serialization.SimpleStringSchema;
7 import org.apache.flink.api.java.utils.ParameterTool;
8 import org.apache.flink.streaming.api.TimeCharacteristic;
9 import org.apache.flink.streaming.api.datastream.DataStream;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11
12 import org.apache.flink.streaming.api.functions.ProcessFunction;
13 import org.apache.flink.streaming.api.functions.timestamps.BoundedOutOfOrderTimestampExtractor;
14 import org.apache.flink.streaming.api.windowing.time.Time;
15 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;
16 import org.apache.flink.table.api.EnvironmentSettings;
17 import org.apache.flink.table.api.Table;
18 import org.apache.flink.table.api.java.StreamTableEnvironment;
19 import org.apache.flink.table.functions.ScalarFunction;
20 import org.apache.flink.types.Row;
21 import org.apache.flink.util.Collector;
22
23
24 import java.sql.Timestamp;
25 import java.time.*;
26 import java.time.format.DateTimeFormatter;
27 import java.util.Properties;
28
29
30 /**
31  * Summary:
32  * UDF
33  */
34 @Slf4j
35 public class Test {
36     public static void main(String[] args) throws Exception{
37
38         //args=new String[]{"--application","flink/src/main/java/com/bigdata/flink/tableSqlUDF/application.properties"};
39
40
41         //1、解析命令行参数
42         ParameterTool fromArgs = ParameterTool.fromArgs(args);
```

```

43     ParameterTool parameterTool = ParameterTool.fromPropertiesFile(f
44     romArgs.getRequired("application"));
45     String kafkaBootstrapServers = parameterTool.getRequired("kafkaB
46     ootstrapServers");
47     String browseTopic = parameterTool.getRequired("browseTopic");
48     String browseTopicGroupID = parameterTool.getRequired("browseTop
49     icGroupID");
50
51     //2、设置运行环境
52     EnvironmentSettings settings = EnvironmentSettings.newInstance()
53     .inStreamingMode().useBlinkPlanner().build();
54     StreamExecutionEnvironment streamEnv = StreamExecutionEnvironmen
55     t.getExecutionEnvironment();
56     streamEnv.setStreamTimeCharacteristic(TimeCharacteristic.EventTi
57     me);
58     StreamTableEnvironment tableEnv = StreamTableEnvironment.create(
59     streamEnv, settings);
60     streamEnv.setParallelism(1);
61
62     //3、注册Kafka数据源
63     Properties browseProperties = new Properties();
64     browseProperties.put("bootstrap.servers", kafkaBootstrapServers);
65     browseProperties.put("group.id", browseTopicGroupID);
66     DataStream<UserBrowseLog> browseStream=streamEnv
67     .addSource(new FlinkKafkaConsumer010<>(browseTopic, new
68     SimpleStringSchema(), browseProperties))
69     .process(new BrowseKafkaProcessFunction())
70     .assignTimestampsAndWatermarks(new BrowseBoundedOutOfOrd
71     ernessTimestampExtractor(Time.seconds(5)));
72
73     // 增加一个额外的字段rowtime为事件时间属性
74     tableEnv.registerDataStream("source_kafka",browseStream,"userID,
75     eventTime,eventTimeTimestamp,eventType,productId,productPrice,rowtime.ro
76     wtime");
77
78     //4、注册UDF
79     //日期转换函数：将Flink Window Start/End Timestamp转换为指定时区时间(
80     默认转换为北京时间)
81     tableEnv.registerFunction("UDFTimestampConverter", new UDFTimest
82     ampConverter());
83
84     //5、运行SQL
85     //基于事件时间，maxOutOfOrderness为5秒，滚动窗口，计算10秒内每个商品被浏
86     览的PV
87     String sql = ""
88     + "      select "
89     + "          UDFTimestampConverter(TUMBLE_START(rowti
90     me, INTERVAL '10' SECOND),'YYYY-MM-dd HH:mm:ss') as window_start, "
91     + "          UDFTimestampConverter(TUMBLE_END(rowtime

```

```

92 , INTERVAL '10' SECOND), 'YYYY-MM-dd HH:mm:ss', '+08:00') as window_end, "
93         + "                productID, "
94         + "                count(1) as browsePV"
95         + "        from source_kafka "
96         + "        group by productID, TUMBLE(rowtime, INTERVAL '10'
97 SECOND)";
98
99     Table table = tableEnv.sqlQuery(sql);
100    tableEnv.toAppendStream(table, Row.class).print();
101
102    //6、开始执行
103    tableEnv.execute(Test.class.getSimpleName());
104
105
106 }
107
108 /**
109  * 自定义UDF
110  */
111 public static class UDFTimestampConverter extends ScalarFunction{
112
113     /**
114      * 默认转换为北京时间
115      * @param timestamp Flink Timestamp 格式时间
116      * @param format 目标格式,如"YYYY-MM-dd HH:mm:ss"
117      * @return 目标时区的时间
118      */
119     public String eval(Timestamp timestamp, String format){
120
121         LocalDateTime noZoneDateTime = timestamp.toLocalDateTime();
122         ZonedDateTime utcZoneDateTime = ZonedDateTime.of(noZoneDateT
123 ime, ZoneId.of("UTC"));
124
125         ZonedDateTime targetZoneDateTime = utcZoneDateTime.withZoneS
126 ameInstant(ZoneId.of("+08:00"));
127
128         return targetZoneDateTime.format(DateTimeFormatter.ofPattern
129 (format));
130     }
131
132     /**
133      * 转换为指定时区时间
134      * @param timestamp Flink Timestamp 格式时间
135      * @param format 目标格式,如"YYYY-MM-dd HH:mm:ss"
136      * @param zoneOffset 目标时区偏移量
137      * @return 目标时区的时间
138      */
139     public String eval(Timestamp timestamp, String format, String zone
140 Offset){

```

```

141
142         LocalDateTime noZoneDateTime = timestamp.toLocalDateTime();
143         ZonedDateTime utcZoneDateTime = ZonedDateTime.of(noZoneDateT
144 ime, ZoneId.of("UTC"));
145
146         ZonedDateTime targetZoneDateTime = utcZoneDateTime.withZoneS
147 ameInstant(ZoneId.of(zoneOffset));
148
149         return targetZoneDateTime.format(DateTimeFormatter.ofPattern
150 (format));
151     }
152 }
153
154
155
156
157 /**
158  * 解析Kafka数据
159  */
160 static class BrowseKafkaProcessFunction extends ProcessFunction<Stri
161 ng, UserBrowseLog> {
162     @Override
163     public void processElement(String value, Context ctx, Collector<
164 UserBrowseLog> out) throws Exception {
165         try {
166
167             UserBrowseLog log = JSON.parseObject(value, UserBrowseLo
168 g.class);
169
170             // 增加一个long类型的时间戳
171             // 指定eventTime为yyyy-MM-dd HH:mm:ss格式的北京时间
172             DateTimeFormatter format = DateTimeFormatter.ofPattern("
173 yyyy-MM-dd HH:mm:ss");
174             OffsetDateTime eventTime = LocalDateTime.parse(log.getEv
175 entTime(), format).atOffset(ZoneOffset.of("+08:00"));
176             // 转换成毫秒时间戳
177             long eventTimeTimestamp = eventTime.toInstant().toEpochM
178 illi();
179             log.setEventTimeTimestamp(eventTimeTimestamp);
180
181             out.collect(log);
182         } catch (Exception ex) {
183             log.error("解析Kafka数据异常...", ex);
184         }
185     }
186 }
187
188 /**
189  * 提取时间戳生成水印

```

```

    */
    static class BrowseBoundedOutOfOrdernessTimestampExtractor extends B
oundedOutOfOrdernessTimestampExtractor<UserBrowseLog> {

        BrowseBoundedOutOfOrdernessTimestampExtractor(Time maxOutOfOrder
ness) {
            super(maxOutOfOrderness);
        }

        @Override
        public long extractTimestamp(UserBrowseLog element) {
            return element.getEventTimeTimestamp();
        }
    }
}

```

结果

```

1 | 2019-12-01 10:02:00,2019-12-01 10:02:10,product_5,7

```

UDAF求Sum

UDAF，自定义聚合函数，需要继承 `AggregateFunction` 抽象类，实现一系列方法。`AggregateFunction` 抽象类如下：

```

1 | abstract class AggregateFunction<T, ACC> extends UserDefinedAggregateFunc
2 | tion<T, ACC>
3 | T: UDAF输出的结果类型
   | ACC: UDAF存放中间结果的类型

```

最基本的UDAF至少需要实现如下三个方法：

- `createAccumulator`：UDAF是聚合操作，需要定义一个存放中间结果的数据结构(即Accumulator)。一般，在这里，初始化时，定义这个Accumulator
- `accumulate`：定义如何根据输入更新Accumulator
- `getValue`：定义如何返回Accumulator中存储的中间结果作为UDAF的最终结果

除了三个基本方法外，在一些特殊的场景，可能还需要以下三个方法：

- `retract`：和accumulate操作相反,定义如何Restract，即减少Accumulator中的值
- `merge`：定义如何merge多个Accumulator

- `resetAccumulator`: 定义如何重置Accumulator

示例

```
1 package com.bigdata.flink.tableSqlUDF.udaf;
2
3 import com.alibaba.fastjson.JSON;
4 import com.bigdata.flink.beans.table.UserBrowseLog;
5 import lombok.extern.slf4j.Slf4j;
6 import org.apache.flink.api.common.serialization.SimpleStringSchema;
7 import org.apache.flink.api.java.utils.ParameterTool;
8 import org.apache.flink.streaming.api.TimeCharacteristic;
9 import org.apache.flink.streaming.api.datastream.DataStream;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11
12 import org.apache.flink.streaming.api.functions.ProcessFunction;
13 import org.apache.flink.streaming.api.functions.timestamps.BoundedOutOfOrderTimestampExtractor;
14 import org.apache.flink.streaming.api.windowing.time.Time;
15 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;
16 import org.apache.flink.table.api.EnvironmentSettings;
17 import org.apache.flink.table.api.Table;
18 import org.apache.flink.table.api.java.StreamTableEnvironment;
19 import org.apache.flink.table.functions.AggregateFunction;
20 import org.apache.flink.table.functions.ScalarFunction;
21 import org.apache.flink.types.Row;
22 import org.apache.flink.util.Collector;
23
24
25 import java.sql.Timestamp;
26 import java.time.*;
27 import java.time.format.DateTimeFormatter;
28 import java.util.Properties;
29
30
31 /**
32  * Summary:
33  *   UDAF
34  */
35 @Slf4j
36 public class Test {
37     public static void main(String[] args) throws Exception{
38
39         //args=new String[]{"--application","flink/src/main/java/com/bigdata/flink/tableSqlUDF/application.properties"};
40
41
42         //1、解析命令行参数
43         ParameterTool fromArgs = ParameterTool.fromArgs(args);
44
```

```

45     ParameterTool parameterTool = ParameterTool.fromPropertiesFile(f
46     romArgs.getRequired("application"));
47     String kafkaBootstrapServers = parameterTool.getRequired("kafkaB
48     ootstrapServers");
49     String browseTopic = parameterTool.getRequired("browseTopic");
50     String browseTopicGroupID = parameterTool.getRequired("browseTop
51     icGroupID");
52
53     //2、设置运行环境
54     EnvironmentSettings settings = EnvironmentSettings.newInstance()
55     .inStreamingMode().useBlinkPlanner().build();
56     StreamExecutionEnvironment streamEnv = StreamExecutionEnvironmen
57     t.getExecutionEnvironment();
58     streamEnv.setStreamTimeCharacteristic(TimeCharacteristic.EventTi
59     me);
60     StreamTableEnvironment tableEnv = StreamTableEnvironment.create(
61     streamEnv, settings);
62     streamEnv.setParallelism(1);
63
64     //3、注册Kafka数据源
65     Properties browseProperties = new Properties();
66     browseProperties.put("bootstrap.servers", kafkaBootstrapServers);
67     browseProperties.put("group.id", browseTopicGroupID);
68     DataStream<UserBrowseLog> browseStream=streamEnv
69     .addSource(new FlinkKafkaConsumer010<>(browseTopic, new
70     SimpleStringSchema(), browseProperties))
71     .process(new BrowseKafkaProcessFunction())
72     .assignTimestampsAndWatermarks(new BrowseBoundedOutOfOrd
73     ernessTimestampExtractor(Time.seconds(5)));
74
75     // 增加一个额外的字段rowtime为事件时间属性
76     tableEnv.registerDataStream("source_kafka",browseStream,"userID,
77     eventTime,eventTimeTimestamp,eventType,productID,productPrice,rowtime.ro
78     wtime");
79
80     //4、注册自定义函数
81     //UDF： 时间转换
82     tableEnv.registerFunction("UDFTimestampConverter", new UDFTimest
83     ampConverter());
84     //UDAF： 求Sum
85     tableEnv.registerFunction("UDAFSum", new UDAFSum());
86
87     //5、运行SQL
88     // 基于事件时间，maxOutOfOrderness为5秒，滚动窗口，计算10秒内每个商品被浏览的总价值
89     String sql = ""
90     + "      select "
91     + "          UDFTimestampConverter(TUMBLE_START(rowti
92     me, INTERVAL '10' SECOND),'YYYY-MM-dd HH:mm:ss') as window_start, "

```



```

94         + "                UDFTimestampConverter(TUMBLE_END(rowtime
95 , INTERVAL '10' SECOND), 'YYYY-MM-dd HH:mm:ss', '+08:00') as window_end, "
96         + "                productID, "
97         + "                UDAFSum(productPrice) as sumPrice"
98         + "        from source_kafka "
99         + "        group by productID, TUMBLE(rowtime, INTERVAL '10'
100 SECOND)";
101
102     Table table = tableEnv.sqlQuery(sql);
103     tableEnv.toAppendStream(table, Row.class).print();
104
105     //6、开始执行
106     tableEnv.execute(Test.class.getSimpleName());
107
108
109 }
110
111 /**
112  * 自定义UDF
113  */
114 public static class UDFTimestampConverter extends ScalarFunction{
115
116     /**
117      * 默认转换为北京时间
118      * @param timestamp Flink Timestamp 格式时间
119      * @param format 目标格式, 如"YYYY-MM-dd HH:mm:ss"
120      * @return 目标时区的时间
121      */
122     public String eval(Timestamp timestamp, String format){
123
124         LocalDateTime noZoneDateTime = timestamp.toLocalDateTime();
125         ZonedDateTime utcZoneDateTime = ZonedDateTime.of(noZoneDateT
126 ime, ZoneId.of("UTC"));
127
128         ZonedDateTime targetZoneDateTime = utcZoneDateTime.withZoneS
129 ameInstant(ZoneId.of("+08:00"));
130
131         return targetZoneDateTime.format(DateTimeFormatter.ofPattern
132 (format));
133     }
134
135     /**
136      * 转换为指定时区时间
137      * @param timestamp Flink Timestamp 格式时间
138      * @param format 目标格式, 如"YYYY-MM-dd HH:mm:ss"
139      * @param zoneOffset 目标时区偏移量
140      * @return 目标时区的时间
141      */
142     public String eval(Timestamp timestamp, String format, String zone

```

```

143 Offset){
144
145     LocalDateTime noZoneDateTime = timestamp.toLocalDateTime();
146     ZonedDateTime utcZoneDateTime = ZonedDateTime.of(noZoneDateT
147 ime, ZoneId.of("UTC"));
148
149     ZonedDateTime targetZoneDateTime = utcZoneDateTime.withZoneS
150 ameInstant(ZoneId.of(zoneOffset));
151
152     return targetZoneDateTime.format(DateTimeFormatter.ofPattern
153 (format));
154 }
155 }
156
157
158 /**
159  * 自定义UDAF
160  */
161 public static class UDAFSum extends AggregateFunction<Long, UDAFSum.
162 SumAccumulator>{
163
164     /**
165      * 定义一个Accumulator, 存放聚合的中间结果
166      */
167     public static class SumAccumulator{
168         public long sumPrice;
169     }
170
171     /**
172      * 初始化Accumulator
173      * @return
174      */
175     @Override
176     public SumAccumulator createAccumulator() {
177         SumAccumulator sumAccumulator = new SumAccumulator();
178         sumAccumulator.sumPrice=0;
179         return sumAccumulator;
180     }
181
182     /**
183      * 定义如何根据输入更新Accumulator
184      * @param accumulator Accumulator
185      * @param productPrice 输入
186      */
187     public void accumulate(SumAccumulator accumulator,int productPri
188 ce){
189         accumulator.sumPrice += productPrice;
190     }
191

```

```

192     /**
193      * 返回聚合的最终结果
194      * @param accumulator Accumulator
195      * @return
196      */
197     @Override
198     public Long getValue(SumAccumulator accumulator) {
199         return accumulator.sumPrice;
200     }
201 }
202
203 /**
204  * 解析Kafka数据
205  */
206     static class BrowseKafkaProcessFunction extends ProcessFunction<String, UserBrowseLog> {
207
208         @Override
209         public void processElement(String value, Context ctx, Collector<UserBrowseLog> out) throws Exception {
210             try {
211
212                 UserBrowseLog log = JSON.parseObject(value, UserBrowseLog.class);
213
214                 // 增加一个long类型的时间戳
215                 // 指定eventTime为yyyy-MM-dd HH:mm:ss格式的北京时间
216                 DateTimeFormatter format = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
217                 OffsetDateTime eventTime = LocalDateTime.parse(log.getEventTime(), format).atOffset(ZoneOffset.of("+08:00"));
218                 // 转换成毫秒时间戳
219                 long eventTimeTimestamp = eventTime.toInstant().toEpochMilli();
220
221                 log.setEventTimeTimestamp(eventTimeTimestamp);
222
223                 out.collect(log);
224             } catch (Exception ex) {
225                 log.error("解析Kafka数据异常...", ex);
226             }
227         }
228     }
229 }
230
231 /**
232  * 提取时间戳生成水印
233  */
234     static class BrowseBoundedOutOfOrdernessTimestampExtractor extends BoundedOutOfOrdernessTimestampExtractor<UserBrowseLog> {
235
236         BrowseBoundedOutOfOrdernessTimestampExtractor(Time maxOutOfOrder

```

```

ness) {
    super(maxOutOfOrderness);
}

@Override
public long extractTimestamp(UserBrowseLog element) {
    return element.getEventTimeTimestamp();
}
}
}

```

结果

```

1 | 2019-12-01 10:02:00,2019-12-01 10:02:10,product_5,140

```

UDTF—列转多列

UDTF，自定义表函数，继承 `TableFunction` 抽象类，主要实现 `eval` 方法。 `TableFunction` 抽象类如下：

```

1 | abstract class TableFunction<T> extends UserDefinedFunction
2 | T: 输出的数据类型

```

注意：

1. 如果需要UDTF返回多列，只需要将返回值类型声明为 `Row` 或 `Tuple` 即可。若返回 `Row`，需要重写 `getResultType` 方法，显示声明返回的 `Row` 的字段类型。如下，示例。
2. 在使用UDTF时，需要带上 `LATERAL` 和 `TABLE` 两个关键字。
3. UDTF支持CROSS JOIN和LEFT JOIN。
 1. `CROSS JOIN`：对于左侧表的每一行，右侧UDTF不输出，则这一行不输出。
 2. `LEFT JOIN`：对于左侧表的每一行，右侧UDTF不输出，则这一行会输出，右侧UDTF字段为Null。

示例

```

1 | package com.bigdata.flink.tableSqlUDF.udtf;
2 |
3 | import com.alibaba.fastjson.JSON;

```

```

4  import com.bigdata.flink.beans.table.UserBrowseLog;
5  import lombok.extern.slf4j.Slf4j;
6  import org.apache.flink.api.common.serialization.SimpleStringSchema;
7  import org.apache.flink.api.common.typeinfo.TypeInformation;
8  import org.apache.flink.api.common.typeinfo.Types;
9  import org.apache.flink.api.java.typeutils.RowTypeInfo;
10 import org.apache.flink.api.java.utils.ParameterTool;
11 import org.apache.flink.streaming.api.TimeCharacteristic;
12 import org.apache.flink.streaming.api.datastream.DataStream;
13 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
14
15 import org.apache.flink.streaming.api.functions.ProcessFunction;
16 import org.apache.flink.streaming.api.functions.timestamps.BoundedOutOfOrderTimestampExtractor;
17
18 import org.apache.flink.streaming.api.windowing.time.Time;
19 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;
20 import org.apache.flink.table.api.EnvironmentSettings;
21 import org.apache.flink.table.api.Table;
22 import org.apache.flink.table.api.java.StreamTableEnvironment;
23 import org.apache.flink.table.functions.AggregateFunction;
24 import org.apache.flink.table.functions.ScalarFunction;
25 import org.apache.flink.table.functions.TableFunction;
26 import org.apache.flink.types.Row;
27 import org.apache.flink.util.Collector;
28
29 import java.sql.Timestamp;
30 import java.time.*;
31 import java.time.format.DateTimeFormatter;
32 import java.util.Properties;
33
34
35 /**
36  * Summary:
37  *      UDTF
38  */
39 @Slf4j
40 public class Test {
41     public static void main(String[] args) throws Exception{
42
43         //args=new String[]{"--application","flink/src/main/java/com/bigdata/flink/tableSqlUDF/application.properties"};
44
45
46         //1、解析命令行参数
47         ParameterTool fromArgs = ParameterTool.fromArgs(args);
48         ParameterTool parameterTool = ParameterTool.fromPropertiesFile(fromArgs.getRequired("application"));
49         String kafkaBootstrapServers = parameterTool.getRequired("kafkaBootstrapServers");
50         String browseTopic = parameterTool.getRequired("browseTopic");

```

```

53         String browseTopicGroupID = parameterTool.getRequired("browseTopicGroupID");
54
55
56         //2、设置运行环境
57         EnvironmentSettings settings = EnvironmentSettings.newInstance()
58             .inStreamingMode().useBlinkPlanner().build();
59         StreamExecutionEnvironment streamEnv = StreamExecutionEnvironment.getExecutionEnvironment();
60         streamEnv.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
61
62         StreamTableEnvironment tableEnv = StreamTableEnvironment.create(
63             streamEnv, settings);
64         streamEnv.setParallelism(1);
65
66
67         //3、注册Kafka数据源
68         Properties browseProperties = new Properties();
69         browseProperties.put("bootstrap.servers", kafkaBootstrapServers);
70         browseProperties.put("group.id", browseTopicGroupID);
71         DataStream<UserBrowseLog> browseStream = streamEnv
72             .addSource(new FlinkKafkaConsumer010<>(browseTopic, new
73             SimpleStringSchema(), browseProperties))
74             .process(new BrowseKafkaProcessFunction())
75             .assignTimestampsAndWatermarks(new BrowseBoundedOutOfOrderTimestampExtractor(Time.seconds(5)));
76
77
78         // 增加一个额外的字段rowtime为事件时间属性
79         tableEnv.registerDataStream("source_kafka", browseStream, "userID,
80             eventTime,eventTimeTimestamp,eventType,productId,productPrice,rowtime.rowtime");
81
82
83         //4、注册自定义函数
84         tableEnv.registerFunction("UDTFOneColumnToMultiColumn", new UDTFOneColumnToMultiColumn());
85
86
87         //5、运行SQL
88         String sql = ""
89             + "select "
90             + "        userID,eventTime,eventTimeTimestamp,eventType,productId,productPrice,rowtime,date1,time1 "
91             + "from source_kafka ,"
92             + "lateral table(UDTFOneColumnToMultiColumn(eventTime))
93             as T(date1,time1)";
94
95
96         Table table = tableEnv.sqlQuery(sql);
97         tableEnv.toAppendStream(table, Row.class).print();
98
99         //6、开始执行
100         tableEnv.execute(Test.class.getSimpleName());
101

```

```

    }

    /**
     * 自定义UDTF
     * 将一列变成两列。
     * 如:2019-12-01 10:02:06 转换成date1(2019-12-01)和time1(10:02:06)两列
     */
    public static class UDTFOneColumnToMultiColumn extends TableFunction
    <Row>{
        public void eval(String value) {
            String[] valueSplits = value.split(" ");

            //一行, 两列
            Row row = new Row(2);
            row.setField(0,valueSplits[0]);
            row.setField(1,valueSplits[1]);
            collect(row);
        }
        @Override
        public TypeInformation<Row> getResultType() {
            return new RowTypeInfo(Types.STRING,Types.STRING);
        }
    }

    /**
     * 解析Kafka数据
     */
    static class BrowseKafkaProcessFunction extends ProcessFunction<String, UserBrowseLog> {
        @Override
        public void processElement(String value, Context ctx, Collector<UserBrowseLog> out) throws Exception {
            try {

                UserBrowseLog log = JSON.parseObject(value, UserBrowseLog.class);

                // 增加一个long类型的时间戳
                // 指定eventTime为yyyy-MM-dd HH:mm:ss格式的北京时间
                DateTimeFormatter format = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                OffsetDateTime eventTime = LocalDateTime.parse(log.getEventTime(), format).atOffset(ZoneOffset.of("+08:00"));
                // 转换成毫秒时间戳
                long eventTimeTimestamp = eventTime.toInstant().toEpochMilli();
            }
        }
    }
}

```

```

        log.setTimeTimestamp(eventTimeTimestamp);

        out.collect(log);
    } catch (Exception ex) {
        log.error("解析Kafka数据异常...", ex);
    }
}

/**
 * 提取时间戳生成水印
 */
static class BrowseBoundedOutOfOrdernessTimestampExtractor extends B
oundedOutOfOrdernessTimestampExtractor<UserBrowseLog> {

    BrowseBoundedOutOfOrdernessTimestampExtractor(Time maxOutOfOrder
ness) {
        super(maxOutOfOrderness);
    }

    @Override
    public long extractTimestamp(UserBrowseLog element) {
        return element.getTimeTimestamp();
    }
}
}

```

结果

```

// 最后两列是用UDTF从第二列中解析出来
user_5,2019-12-01 10:02:06,1575165726000,browse,product_5,20,2019-12-01T
02:02:06,2019-12-01,10:02:06
user_5,2019-12-01 10:02:06,1575165726000,browse,product_5,20,2019-12-01T
02:02:06,2019-12-01,10:02:06
user_5,2019-12-01 10:02:06,1575165726000,browse,product_5,20,2019-12-01T
02:02:06,2019-12-01,10:02:06
user_5,2019-12-01 10:02:00,1575165720000,browse,product_5,20,2019-12-01T
02:02:00,2019-12-01,10:02:00
user_4,2019-12-01 10:02:10,1575165730000,browse,product_5,20,2019-12-01T
02:02:10,2019-12-01,10:02:10
user_4,2019-12-01 10:02:12,1575165732000,browse,product_5,20,2019-12-01T
02:02:12,2019-12-01,10:02:12
user_4,2019-12-01 10:02:15,1575165735000,browse,product_5,20,2019-12-01T
02:02:15,2019-12-01,10:02:15
user_4,2019-12-01 10:02:02,1575165722000,browse,product_5,20,2019-12-01T
02:02:02,2019-12-01,10:02:02

```



```
user_5,2019-12-01 10:02:06,1575165726000,browse,product_5,20,2019-12-01T  
02:02:06,2019-12-01,10:02:06  
user_5,2019-12-01 10:02:06,1575165726000,browse,product_5,20,2019-12-01T  
02:02:06,2019-12-01,10:02:06  
user_4,2019-12-01 10:02:16,1575165736000,browse,product_5,20,2019-12-01T  
02:02:16,2019-12-01,10:02:16
```