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列输入多列输出。

测试数据

```
// 某个用户在某个时刻浏览了某个商品，以及商品的价值
// eventTime: 北京时间，方便测试。如下，乱序数据:
{"userID": "user_5", "eventTime": "2019-12-01 10:02:00", "eventType": "b
rowse", "productID": "product_5", "productPrice": 20}
{"userID": "user_4", "eventTime": "2019-12-01 10:02:02", "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 10:02:10", "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 10:02:12", "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

```
package com.bigdata.flink.tableSqlUDF.udf;

import com.alibaba.fastjson.JSON;
import com.bigdata.flink.beans.table.UserBrowseLog;
import lombok.extern.slf4j.Slf4j;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironm
ent;
import org.apache.flink.streaming.api.functions.ProcessFunction;
import org.apache.flink.streaming.api.functions.timestamps.BoundedOutOfOr
dernessTimestampExtractor;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;
import org.apache.flink.table.api.EnvironmentSettings;
import org.apache.flink.table.api.Table;
import org.apache.flink.table.api.java.StreamTableEnvironment;
import org.apache.flink.table.functions.ScalarFunction;
import org.apache.flink.types.Row;
import org.apache.flink.util.Collector;

import java.sql.Timestamp;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.Properties;


/**
 * Summary:
 *  UDF
 */
@Slf4j
public class Test {
    public static void main(String[] args) throws Exception{

        //args=new String[]{"--application","flink/src/main/java/com/bigd
ata/flink/tableSqlUDF/application.properties"};

        //1、解析命令行参数
        ParameterTool fromArgs = ParameterTool.fromArgs(args);
```

```
        ParameterTool parameterTool = ParameterTool.fromPropertiesFile(f
romArgs.getRequired("application"));
        String kafkaBootstrapServers = parameterTool.getRequired("kafkaB
ootstrapServers");
        String browseTopic = parameterTool.getRequired("browseTopic");
        String browseTopicGroupID = parameterTool.getRequired("browseTop
icGroupID");

        //2、设置运行环境
        EnvironmentSettings settings = EnvironmentSettings.newInstance()
.inStreamingMode().useBlinkPlanner().build();
        StreamExecutionEnvironment streamEnv = StreamExecutionEnvironmen
t.getExecutionEnvironment();
        streamEnv.setStreamTimeCharacteristic(TimeCharacteristic.EventTi
me);
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(
streamEnv, settings);
        streamEnv.setParallelism(1);

        //3、注册Kafka数据源
        Properties browseProperties = new Properties();
        browseProperties.put("bootstrap.servers",kafkaBootstrapServers);
        browseProperties.put("group.id",browseTopicGroupID);
        DataStream<UserBrowseLog> browseStream=streamEnv
                .addSource(new FlinkKafkaConsumer010<>(browseTopic, new
SimpleStringSchema(), browseProperties))
                .process(new BrowseKafkaProcessFunction())
                .assignTimestampsAndWatermarks(new BrowseBoundedOutOfOrd
ernessTimestampExtractor(Time.seconds(5)));

        // 增加一个额外的字段rowtime为事件时间属性
        tableEnv.registerDataStream("source_kafka",browseStream,"userID,
eventTime,eventTimeTimestamp,eventType,productID,productPrice,rowtime.ro
wtime");

        //4、注册UDF
        //日期转换函数: 将Flink Window Start/End Timestamp转换为指定时区时间(
默认转换为北京时间)
        tableEnv.registerFunction("UDFTimestampConverter", new UDFTimest
ampConverter());

        //5、运行SQL
        //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PV
        String sql = ""
                + "     select "
                + "             UDFTimestampConverter(TUMBLE_START(rowti
me, INTERVAL '10' SECOND),'YYYY-MM-dd HH:mm:ss') as window_start, "
                + "             UDFTimestampConverter(TUMBLE_END(rowtime
```

```
, INTERVAL '10' SECOND),'YYYY-MM-dd HH:mm:ss','+08:00') as window_end, "
                + "            productID, "
                + "            count(1) as browsePV"
                + "     from source_kafka "
                + "     group by productID,TUMBLE(rowtime, INTERVAL '10'
 SECOND)";

        Table table = tableEnv.sqlQuery(sql);
        tableEnv.toAppendStream(table,Row.class).print();

        //6、开始执行
        tableEnv.execute(Test.class.getSimpleName());


    }

    /**
     * 自定义UDF
     */
    public static class UDFTimestampConverter extends ScalarFunction{

        /**
         * 默认转换为北京时间
         * @param timestamp Flink Timestamp 格式时间
         * @param format 目标格式,如"YYYY-MM-dd HH:mm:ss"
         * @return 目标时区的时间
         */
        public String eval(Timestamp timestamp,String format){

            LocalDateTime noZoneDateTime = timestamp.toLocalDateTime();
            ZonedDateTime utcZoneDateTime = ZonedDateTime.of(noZoneDateT
ime, ZoneId.of("UTC"));

            ZonedDateTime targetZoneDateTime = utcZoneDateTime.withZoneS
ameInstant(ZoneId.of("+08:00"));

            return targetZoneDateTime.format(DateTimeFormatter.ofPattern
(format));
        }

        /**
         * 转换为指定时区时间
         * @param timestamp Flink Timestamp 格式时间
         * @param format 目标格式,如"YYYY-MM-dd HH:mm:ss"
         * @param zoneOffset 目标时区偏移量
         * @return 目标时区的时间
         */
        public String eval(Timestamp timestamp,String format,String zone
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
                // 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.getEv
entTime(), format).atOffset(ZoneOffset.of("+08:00"));
                // 转换成毫秒时间戳
                long eventTimeTimestamp = eventTime.toInstant().toEpochM
illi();
                log.setEventTimeTimestamp(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.getEventTimeTimestamp();
        }
    }
}
```

结果

```
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:

```
abstract class AggregateFunction<T, ACC> extends UserDefinedAggregateFunc
tion<T, ACC>
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

```
package com.bigdata.flink.tableSqlUDF.udaf;

import com.alibaba.fastjson.JSON;
import com.bigdata.flink.beans.table.UserBrowseLog;
import lombok.extern.slf4j.Slf4j;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironm
ent;
import org.apache.flink.streaming.api.functions.ProcessFunction;
import org.apache.flink.streaming.api.functions.timestamps.BoundedOutOfOr
dernessTimestampExtractor;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;
import org.apache.flink.table.api.EnvironmentSettings;
import org.apache.flink.table.api.Table;
import org.apache.flink.table.api.java.StreamTableEnvironment;
import org.apache.flink.table.functions.AggregateFunction;
import org.apache.flink.table.functions.ScalarFunction;
import org.apache.flink.types.Row;
import org.apache.flink.util.Collector;

import java.sql.Timestamp;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.Properties;


/**
 * Summary:
 *   UDAF
 */
@Slf4j
public class Test {
    public static void main(String[] args) throws Exception{

        //args=new String[]{"--application","flink/src/main/java/com/bigd
ata/flink/tableSqlUDF/application.properties"};

        //1、解析命令行参数
        ParameterTool fromArgs = ParameterTool.fromArgs(args);

```

```
        ParameterTool parameterTool = ParameterTool.fromPropertiesFile(f
romArgs.getRequired("application"));
        String kafkaBootstrapServers = parameterTool.getRequired("kafkaB
ootstrapServers");
        String browseTopic = parameterTool.getRequired("browseTopic");
        String browseTopicGroupID = parameterTool.getRequired("browseTop
icGroupID");

        //2、设置运行环境
        EnvironmentSettings settings = EnvironmentSettings.newInstance()
.inStreamingMode().useBlinkPlanner().build();
        StreamExecutionEnvironment streamEnv = StreamExecutionEnvironmen
t.getExecutionEnvironment();
        streamEnv.setStreamTimeCharacteristic(TimeCharacteristic.EventTi
me);
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(
streamEnv, settings);
        streamEnv.setParallelism(1);

        //3、注册Kafka数据源
        Properties browseProperties = new Properties();
        browseProperties.put("bootstrap.servers",kafkaBootstrapServers);
        browseProperties.put("group.id",browseTopicGroupID);
        DataStream<UserBrowseLog> browseStream=streamEnv
                .addSource(new FlinkKafkaConsumer010<>(browseTopic, new
SimpleStringSchema(), browseProperties))
                .process(new BrowseKafkaProcessFunction())
                .assignTimestampsAndWatermarks(new BrowseBoundedOutOfOrd
ernessTimestampExtractor(Time.seconds(5)));

        // 增加一个额外的字段rowtime为事件时间属性
        tableEnv.registerDataStream("source_kafka",browseStream,"userID,
eventTime,eventTimeTimestamp,eventType,productID,productPrice,rowtime.ro
wtime");

        //4、注册自定义函数
        //UDF: 时间转换
        tableEnv.registerFunction("UDFTimestampConverter", new UDFTimest
ampConverter());
        //UDAF: 求Sum
        tableEnv.registerFunction("UDAFSum", new UDAFSum());

        //5、运行SQL
        //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
        String sql = ""
                + "     select "
                + "             UDFTimestampConverter(TUMBLE_START(rowti
me, INTERVAL '10' SECOND),'YYYY-MM-dd HH:mm:ss') as window_start, "
```

```
                + "            UDFTimestampConverter(TUMBLE_END(rowtime
, INTERVAL '10' SECOND),'YYYY-MM-dd HH:mm:ss','+08:00') as window_end, "
                + "            productID, "
                + "            UDAFSum(productPrice) as sumPrice"
                + "    from source_kafka "
                + "    group by productID,TUMBLE(rowtime, INTERVAL '10'
 SECOND)";

        Table table = tableEnv.sqlQuery(sql);
        tableEnv.toAppendStream(table,Row.class).print();

        //6、开始执行
        tableEnv.execute(Test.class.getSimpleName());


    }

    /**
     * 自定义UDF
     */
    public static class UDFTimestampConverter extends ScalarFunction{

        /**
         * 默认转换为北京时间
         * @param timestamp Flink Timestamp 格式时间
         * @param format 目标格式,如"YYYY-MM-dd HH:mm:ss"
         * @return 目标时区的时间
         */
        public String eval(Timestamp timestamp,String format){

            LocalDateTime noZoneDateTime = timestamp.toLocalDateTime();
            ZonedDateTime utcZoneDateTime = ZonedDateTime.of(noZoneDateT
ime, ZoneId.of("UTC"));

            ZonedDateTime targetZoneDateTime = utcZoneDateTime.withZoneS
ameInstant(ZoneId.of("+08:00"));

            return targetZoneDateTime.format(DateTimeFormatter.ofPattern
(format));
        }

        /**
         * 转换为指定时区时间
         * @param timestamp Flink Timestamp 格式时间
         * @param format 目标格式,如"YYYY-MM-dd HH:mm:ss"
         * @param zoneOffset 目标时区偏移量
         * @return 目标时区的时间
         */
        public String eval(Timestamp timestamp,String format,String zone
```

```
Offset){

            LocalDateTime noZoneDateTime = timestamp.toLocalDateTime();
            ZonedDateTime utcZoneDateTime = ZonedDateTime.of(noZoneDateT
ime, ZoneId.of("UTC"));

            ZonedDateTime targetZoneDateTime = utcZoneDateTime.withZoneS
ameInstant(ZoneId.of(zoneOffset));

            return targetZoneDateTime.format(DateTimeFormatter.ofPattern
(format));
        }
    }


    /**
     * 自定义UDAF
     */
    public static class UDAFSum extends AggregateFunction<Long, UDAFSum.
SumAccumulator>{

        /**
         * 定义一个Accumulator，存放聚合的中间结果
         */
        public static class SumAccumulator{
            public long sumPrice;
        }

        /**
         * 初始化Accumulator
         * @return
         */
        @Override
        public SumAccumulator createAccumulator() {
            SumAccumulator sumAccumulator = new SumAccumulator();
            sumAccumulator.sumPrice=0;
            return sumAccumulator;
        }

        /**
         * 定义如何根据输入更新Accumulator
         * @param accumulator  Accumulator
         * @param productPrice 输入
         */
        public void accumulate(SumAccumulator accumulator,int productPri
ce){
            accumulator.sumPrice += productPrice;
        }

```

```
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<Stri
207     ng, UserBrowseLog> {
208             @Override
209             public void processElement(String value, Context ctx, Collector<
210     UserBrowseLog> out) throws Exception {
211                 try {
212 
213                     UserBrowseLog log = JSON.parseObject(value, UserBrowseLo
    g.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.getEv
    entTime(), format).atOffset(ZoneOffset.of("+08:00"));
                    // 转换成毫秒时间戳
                    long eventTimeTimestamp = eventTime.toInstant().toEpochM
    illi();
                    log.setEventTimeTimestamp(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.getEventTimeTimestamp();
            }
        }
    }
```

结果

```
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:

```
abstract class TableFunction<T> extends UserDefinedFunction
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

```
package com.bigdata.flink.tableSqlUDF.udtf;

import com.alibaba.fastjson.JSON;
```

```
import com.bigdata.flink.beans.table.UserBrowseLog;
import lombok.extern.slf4j.Slf4j;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.api.java.typeutils.RowTypeInfo;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironm
ent;
import org.apache.flink.streaming.api.functions.ProcessFunction;
import org.apache.flink.streaming.api.functions.timestamps.BoundedOutOfOr
dernessTimestampExtractor;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;
import org.apache.flink.table.api.EnvironmentSettings;
import org.apache.flink.table.api.Table;
import org.apache.flink.table.api.java.StreamTableEnvironment;
import org.apache.flink.table.functions.AggregateFunction;
import org.apache.flink.table.functions.ScalarFunction;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
import org.apache.flink.util.Collector;

import java.sql.Timestamp;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.Properties;


/**
 * Summary:
 *    UDTF
 */
@Slf4j
public class Test {
    public static void main(String[] args) throws Exception{

        //args=new String[]{"--application","flink/src/main/java/com/bigd
ata/flink/tableSqlUDF/application.properties"};

        //1、解析命令行参数
        ParameterTool fromArgs = ParameterTool.fromArgs(args);
        ParameterTool parameterTool = ParameterTool.fromPropertiesFile(f
romArgs.getRequired("application"));
        String kafkaBootstrapServers = parameterTool.getRequired("kafkaB
ootstrapServers");
        String browseTopic = parameterTool.getRequired("browseTopic");
```

```
        String browseTopicGroupID = parameterTool.getRequired("browseTop
icGroupID");

        //2、设置运行环境
        EnvironmentSettings settings = EnvironmentSettings.newInstance()
.inStreamingMode().useBlinkPlanner().build();
        StreamExecutionEnvironment streamEnv = StreamExecutionEnvironmen
t.getExecutionEnvironment();
        streamEnv.setStreamTimeCharacteristic(TimeCharacteristic.EventTi
me);
        StreamTableEnvironment tableEnv = StreamTableEnvironment.create(
streamEnv, settings);
        streamEnv.setParallelism(1);

        //3、注册Kafka数据源
        Properties browseProperties = new Properties();
        browseProperties.put("bootstrap.servers",kafkaBootstrapServers);
        browseProperties.put("group.id",browseTopicGroupID);
        DataStream<UserBrowseLog> browseStream=streamEnv
                .addSource(new FlinkKafkaConsumer010<>(browseTopic, new
SimpleStringSchema(), browseProperties))
                .process(new BrowseKafkaProcessFunction())
                .assignTimestampsAndWatermarks(new BrowseBoundedOutOfOrd
ernessTimestampExtractor(Time.seconds(5)));

        // 增加一个额外的字段rowtime为事件时间属性
        tableEnv.registerDataStream("source_kafka",browseStream,"userID,
eventTime,eventTimeTimestamp,eventType,productID,productPrice,rowtime.ro
wtime");

        //4、注册自定义函数
        tableEnv.registerFunction("UDTFOneColumnToMultiColumn",new UDTFO
neColumnToMultiColumn());

        //5、运行SQL
        String sql = ""
                + "select "
                + "     userID,eventTime,eventTimeTimestamp,eventType,pr
oductID,productPrice,rowtime,date1,time1 "
                + "from source_kafka ,"
                + "lateral table(UDTFOneColumnToMultiColumn(eventTime))
as T(date1,time1)";

        Table table = tableEnv.sqlQuery(sql);
        tableEnv.toAppendStream(table,Row.class).print();

        //6、开始执行
        tableEnv.execute(Test.class.getSimpleName());

```

```
102
103         }
104
105
106         /**
107          * 自定义UDTF
108          * 将一列变成两列。
109          * 如:2019-12-01 10:02:06 转换成date1(2019-12-01)和time1(10:02:06)两列
110     。
111          */
112         public static class UDTFOneColumnToMultiColumn extends TableFunction
113     <Row>{
114             public void eval(String value) {
115                 String[] valueSplits = value.split(" ");
116
117                 //一行，两列
118                 Row row = new Row(2);
119                 row.setField(0,valueSplits[0]);
120                 row.setField(1,valueSplits[1]);
121                 collect(row);
122             }
123             @Override
124             public TypeInformation<Row> getResultType() {
125                 return new RowTypeInfo(Types.STRING,Types.STRING);
126             }
127         }
128
129         /**
130          * 解析Kafka数据
131          */
132         static class BrowseKafkaProcessFunction extends ProcessFunction<Stri
133     ng, UserBrowseLog> {
134             @Override
135             public void processElement(String value, Context ctx, Collector<
136     UserBrowseLog> out) throws Exception {
137                 try {
138
139                     UserBrowseLog log = JSON.parseObject(value, UserBrowseLo
140     g.class);
141
142                     // 增加一个long类型的时间戳
143                     // 指定eventTime为yyyy-MM-dd HH:mm:ss格式的北京时间
144                     DateTimeFormatter format = DateTimeFormatter.ofPattern("
145     yyyy-MM-dd HH:mm:ss");
146                     OffsetDateTime eventTime = LocalDateTime.parse(log.getEv
147     entTime(), format).atOffset(ZoneOffset.of("+08:00"));
148                     // 转换成毫秒时间戳
149                     long eventTimeTimestamp = eventTime.toInstant().toEpochM
150     illi();
```

```
                log.setEventTimeTimestamp(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.getEventTimeTimestamp();
        }
    }

}
```

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
```