Flink SQL 实战 (5)：使用自定义函数实现关键字过滤统计

Flink SQL 实战 (5)：使用自定义函数实现关键字过滤统计

在上一篇实战博客中使用POJO Schema解析来自 Kafka 的 JSON 数据源并且使用自定义函数处理。

现在我们使用更强大自定义函数处理数据

使用自定义函数实现关键字过滤统计

自定义表函数(UDTF)

与自定义的标量函数相似，自定义表函数将零，一个或多个标量值作为输入参数。 但是，与标量函数相比，它可以返回任意数量的行作为输出，而不是单个值。

为了定义表函数，必须扩展基类TableFunction并实现**评估方法**。 表函数的行为由其评估方法确定。 必须将评估方法声明为公开并命名为eval。 通过实现多个名为eval的方法，可以重载TableFunction。 评估方法的参数类型确定表函数的所有有效参数。 返回表的类型由TableFunction的通用类型确定。 评估方法使用 collect（T）方法发出输出行。

定义一个过滤字符串 记下关键字 的自定义表函数

KyeWordCount.java:

```
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.table.functions.TableFunction;

public class KyeWordCount extends TableFunction<Tuple2<String,Integer>>
{
    private String[] keys;
    public KyeWordCount(String[] keys){
        this.keys=keys;
    }
    public void eval(String in){
        for (String key:keys){
            if (in.contains(key)){
                collect(new Tuple2<String, Integer>(key,1));
            }
        }
```

```
    }
}
```

实现关键字过滤统计：

```
public class UdtfJob {
    public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment streamEnv = StreamExecutionEnvironmen
t.getExecutionEnvironment();
        EnvironmentSettings streamSettings = EnvironmentSettings.newInst
ance().useBlinkPlanner().inStreamingMode().build();
        StreamTableEnvironment streamTabelEnv = StreamTableEnvironment.c
reate(streamEnv, streamSettings);
        KafkaTabelSource kafkaTabelSource = new KafkaTabelSource();
        streamTabelEnv.registerTableSource("kafkaDataStream", kafkaTabel
Source);//使用自定义TableSource
        //注册自定义函数定义三个关键字："KeyWord","WARNING","illegal"
        streamTabelEnv.registerFunction("CountKEY", new KyeWordCount(new
 String[]{"KeyWord","WARNING","illegal"}));
        //编写SQL
        Table wordWithCount = streamTabelEnv.sqlQuery("SELECT key,COUNT(
countv) AS countsum FROM kafkaDataStream LEFT JOIN LATERAL TABLE(CountKE
Y(response)) as T(key, countv) ON TRUE GROUP BY key");
        //直接输出Retract流
        streamTabelEnv.toRetractStream(wordWithCount, Row.class).print()
;
        streamTabelEnv.execute("BLINK STREAMING QUERY");
    }
}
```

测试用Python脚本如下

```
# https://pypi.org/project/kafka-python/
import pickle
import time
import json
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers=['127.0.0.1:9092'],
                         key_serializer=lambda k: pickle.dumps(k),
                         value_serializer=lambda v: pickle.dumps(v))
start_time = time.time()
for i in range(0, 10000):
    print('------{}---------'.format(i))
    producer = KafkaProducer(value_serializer=lambda v: json.dumps(v).en
code('utf-8'),compression_type='gzip')
```

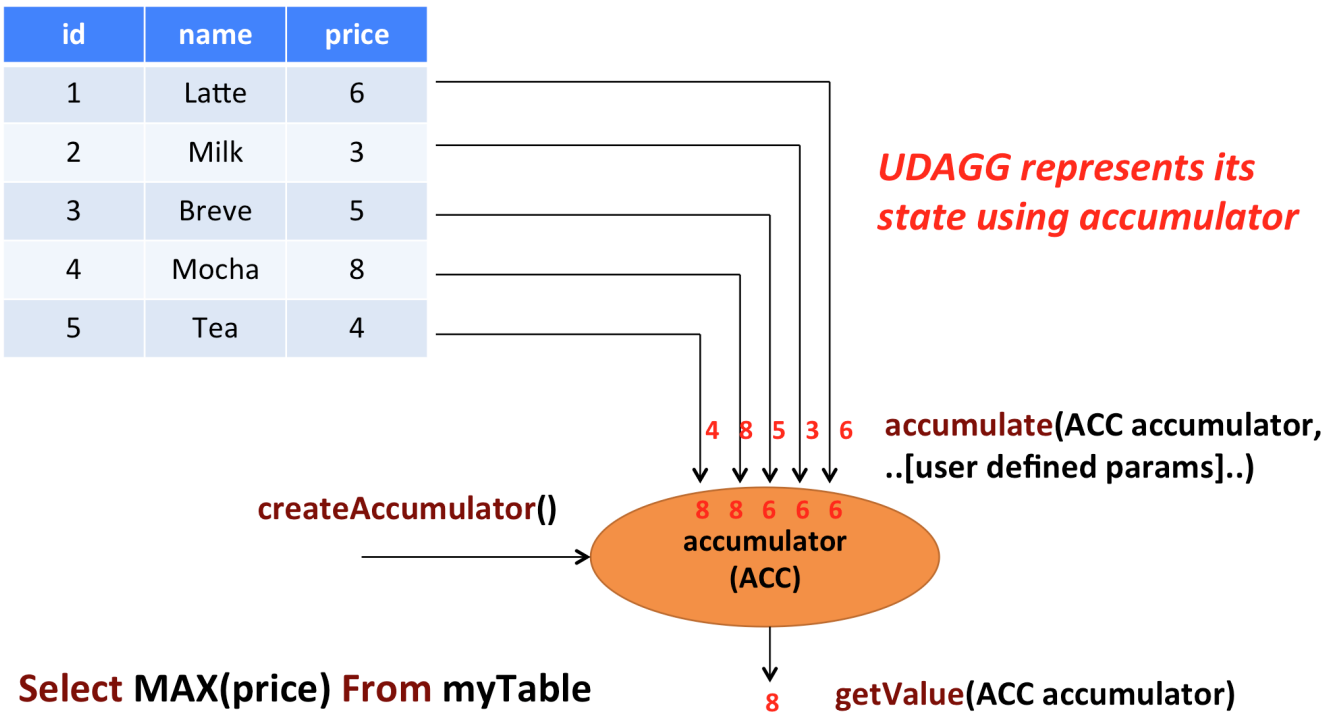
```
    producer.send('test',{"response":"resKeyWordWARNINGillegal","status"
:0,"protocol":"protocol","timestamp":0})
    producer.send('test',{"response":"resKeyWordWARNINGillegal","status"
:1,"protocol":"protocol","timestamp":0})
    producer.send('test',{"response":"resresKeyWordWARNING","status":2,"
protocol":"protocol","timestamp":0})
    producer.send('test',{"response":"resKeyWord","status":3,"protocol":
"protocol","timestamp":0})
    producer.send('test',{"response":"res","status":4,"protocol":"protoc
ol","timestamp":0})
    producer.send('test',{"response":"res","status":5,"protocol":"protoc
ol","timestamp":0})
#    future = producer.send('test', key='num', value=i, partition=0)
# 将缓冲区的全部消息push到broker当中
producer.flush()
producer.close()

end_time = time.time()
time_counts = end_time - start_time
print(time_counts)
```

控制台输出：

```
...
6> (false,KeyWord,157)
3> (false,WARNING,119)
3> (true,WARNING,120)
6> (true,KeyWord,158)
7> (true,illegal,80)
6> (false,KeyWord,158)
6> (true,KeyWord,159)
6> (false,KeyWord,159)
6> (true,KeyWord,160)
...
```

自定义聚合函数

自定义聚合函数(UDAGGs)将一个表聚合为一个标量值。



聚合函数适合用于累计的工作，上面的图显示了聚合的一个示例。假设您有一个包含饮料数据的表。该表由三列组成:id、name和price，共计5行。想象一下，你需要找到所有饮料的最高价格。执行max()聚合。您需要检查5行中的每一行，结果将是单个数值。

用户定义的聚合函数是通过扩展AggregateFunction类来实现的。AggregateFunction的工作原理如下。首先，它需要一个累加器，这个累加器是保存聚合中间结果的数据结构。通过调用AggregateFunction的createAccumulator()方法来创建一个空的累加器。随后，对每个输入行调用该函数的accumulator()方法来更新累加器。处理完所有行之后，将调用函数的getValue()方法来计算并返回最终结果。

**每个AggregateFunction必须使用以下方法： **

- `createAccumulator()`创建一个空的累加器
- `accumulate()`更新累加器
- `getValue()`计算并返回最终结果

除了上述方法之外，还有一些可选方法。虽然其中一些方法允许系统更有效地执行查询，但是对于某些用例是必需的。例如，如果应该在会话组窗口的上下文中应用聚合函数，那么merge()方法是必需的(当观察到连接它们的行时，需要连接两个会话窗口的累加器。

AggregateFunction可选方法

- `retract()` 定义restract:减少Accumulator， 对于在有界窗口上的聚合是必需的。
- `merge()` merge多个Accumulator， 对于许多批处理聚合和会话窗口聚合都是必需的。
- `resetAccumulator()` 重置Accumulator， 对于许多批处理聚合都是必需的。

使用聚合函数聚合最大的status值

编写自定义聚合函数，用于聚合出最大的status

```
public class MaxStatus extends AggregateFunction<Integer,MaxStatus.StatusACC> {
    @Override
    public Integer getValue(StatusACC statusACC) {
        return statusACC.maxStatus;
    }

    @Override
    public StatusACC createAccumulator() {
        return new StatusACC();
    }
    public void accumulate(StatusACC statusACC,int status){
        if (status>statusACC.maxStatus){
            statusACC.maxStatus=status;
        }
    }
    public static class StatusACC{
        public int maxStatus=0;
    }
}
```

mian函数修改注册和SQL就可以使用

```
/**
 *聚合最大的status
 */
streamTabelEnv.registerFunction("maxStatus",new MaxStatus());
Table wordWithCount = streamTabelEnv.sqlQuery("SELECT maxStatus(status) AS maxStatus FROM kafkaDataStream");
```

使用之前的python脚本测试

控制台输出（全部）：

```
5> (false,1)
8> (true,3)
3> (false,0)
4> (true,1)
6> (true,2)
2> (true,0)
2> (true,4)
1> (false,3)
7> (false,2)
3> (false,4)
4> (true,5)
```

除非输入更大的Status，否则控制台不会继续输出新结果

表聚合函数

用户定义的表聚合函数(UDTAGGs)将一个表(具有一个或多个属性的一个或多个行)聚合到具有多行和多列的结果表。

和聚合函数几乎一致，有需求的朋友可以参考官方文档

Table Aggregation Functions

GitHub

项目源码已上传至GitHub

https://github.com/StarPlatinumStudio/Flink-SQL-Practice

我的专栏：Flink SQL原理和实战

To Be Continue=>