零基础学 Flink: CEP 复杂事件处理

> 内容简介：上一篇文章，我们介绍了UDF，可以帮用户自定义函数，从而在使用Flink SQL中，能够得心应手的处理一些数据问题。今天我们来学习一下Flink是如何处理CEP问题的。本文会分为两个部分，概念介绍部分和代码案例部分。那么什么是CEP？CEP即Complex event processing，引用wiki的解释：
> CEP, is event processing that combines data from multiple sources to infer events or patterns that s
>
>

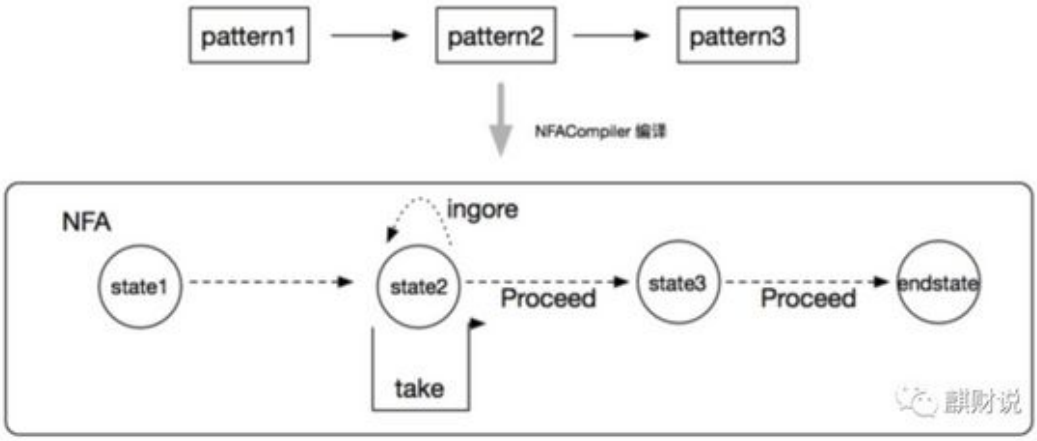
上一篇文章，我们介绍了UDF，可以帮用户自定义函数，从而在使用Flink SQL中，能够得心应手的处理一些数据问题。今天我们来学习一下Flink是如何处理CEP问题的。本文会分为两个部分，概念介绍部分和代码案例部分。

概念介绍

那么什么是CEP？CEP即Complex event processing，引用wiki的解释：

CEP, is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal of complex event processing is to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible.

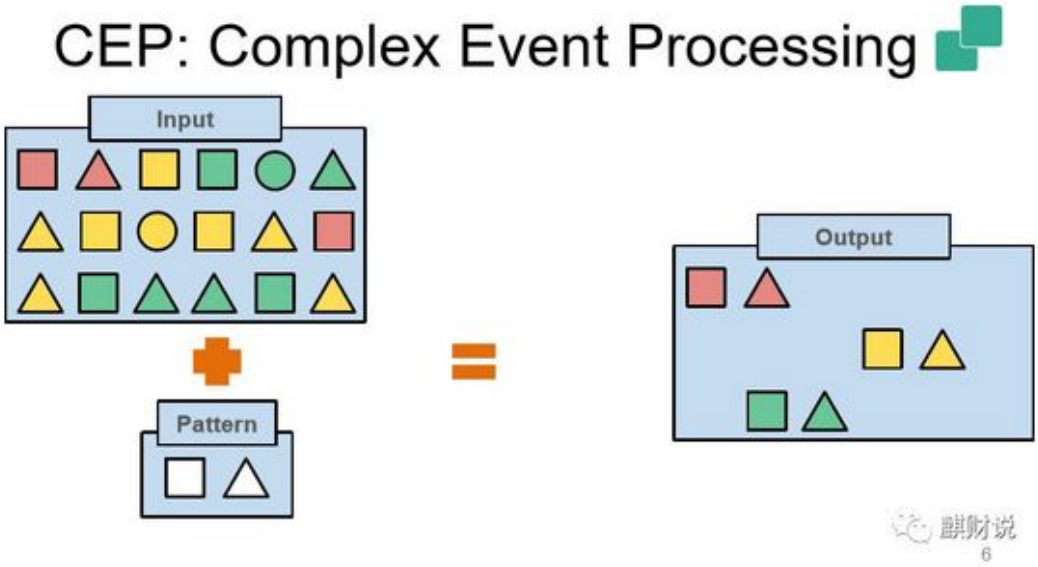
Flink CEP（理论基础《Efficient Pattern Matching over Event Streams 》，对该片论文有兴趣的同学，可以找我索取）是构建在 DataStream API 上的，首先需要用户创建定义一个个pattern，然后通过链表将由前后逻辑关系的pattern串在一起，构成模式匹配的逻辑表达。 然后需要用户利用NFACompiler，将模式进行分拆，创建出NFA(非确定有限自动机)对象，NFA包含了该次模式匹配的各个状态和状态间转换的表达式。 整个示意图就像如下：



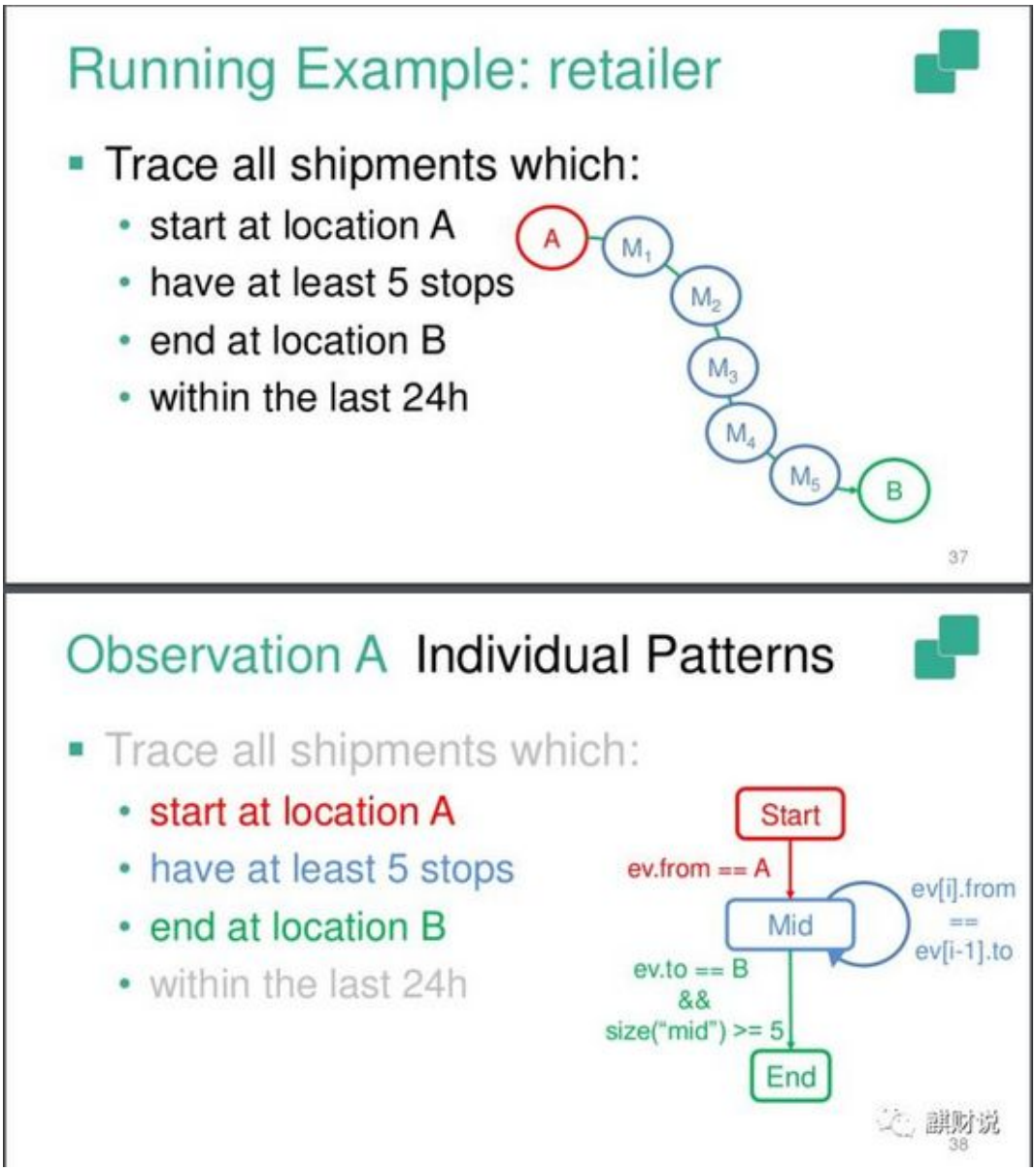
上图中的三个pattern通过编译生成了NFA，NFA包含了四个状态，其中endstate是在编译的时候自动加上的，来作为终止状态。状态间转换是通过箭头表示的状态迁移边(StateTransition)来实现的， 每个状态迁移会涉及到三类状态迁移边，分别是Take、Proceed、Ingore。

- Take: 表示事件匹配成功，将当前状态更新到新状态，并前进到“下一个”状态；
- Procceed: 当事件来到的时候，当前状态不发生变化，在状态转换图中事件直接“前进”到下一个目标状态；
- IGNORE: 当事件来到的时候，如果匹配不成功，忽略当前事件，当前状态不发生任何变化。

说了这么多，CEP到底能解决什么问题？简单总结如下图：



可能你会觉得我再逗你，这不就是实现了一个过滤么，其实不然，我们再看下面的例子



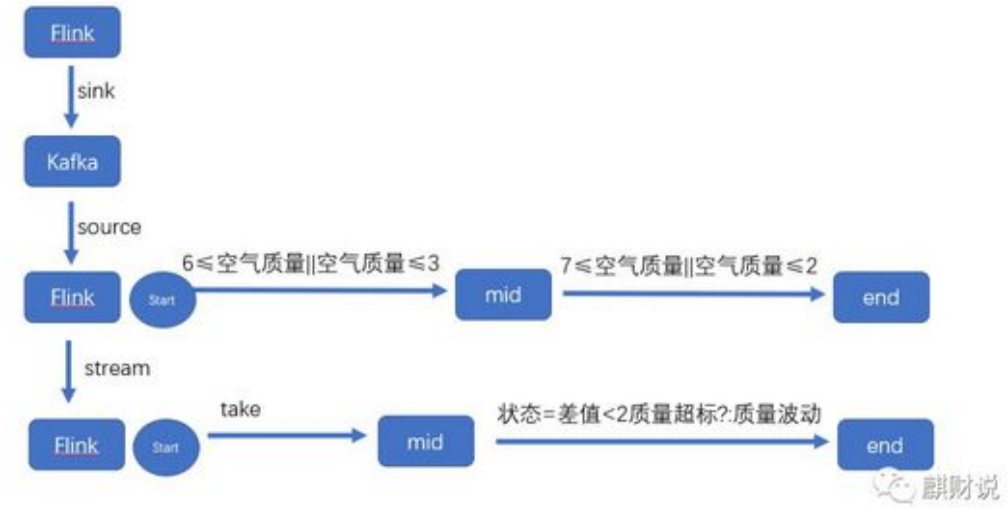
filter算子可以实现对数据的过滤，那么CEP除了对数据过滤，还可以实现一个流程的计算操作。比如我们可以计算从A到B在24个小时内，经历5个节点的数据。

代码案例

首先我们来介绍一下规则（假设规则）：

假设一个数据流，持续写入各地空气质量信息，如果某地连续两次空气质量超过6和7或是小于3和2，就认为其控制质量异常，将记录这条预警，并且将记录再进行处理，如果前后两次样本差值的绝对值小于2，则认为是空气质量超标，否则是空气异常波动。

下图是代码本次的代码流程。先启动flink执行sink将模拟数据写到kafka，然后再启动一个flink消费kafka的数据，并进行CEP。



首先我们定义空气质量对象，包括ID，城市，空气质量，记录时间和时间戳。同时模拟了一个记录发生器（createOne）来创建模拟数据。

```
import java.io.Serializable;
import java.util.Date;
import java.util.Random;
public class AirQualityRecoder implements Serializable {
    private String id;
    private String city;
    private Integer airQuality;
    private Date emmit;
    private Long et;
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public Integer getAirQuality() {
        return airQuality;
    }

    public void setAirQuality(Integer airQuality) {
        this.airQuality = airQuality;
    }

    public Date getEmmit() {
```

```
        return emmit;
    }

    public void setEmmit(Date emmit) {
        this.emmit = emmit;
    }

    public Long getEt() {
        return et;
    }

    public void setEt(Long et) {
        this.et = et;
    }

    public AirQualityRecoder() {

    }

    public AirQualityRecoder(String city, Integer airQuality, Date emmit, Long e
t) {
        this.city = city;
        this.airQuality = airQuality;
        this.emmit = emmit;
        this.et = et;
    }

    @Override
    public String toString() {
        return "AirQualityRecoder{" +
                "id='" + id + '\'' +
                ", city='" + city + '\'' +
                ", airQuality=" + airQuality +
                ", emmit=" + emmit +
                ", et=" + et +
                '}';
    }

    public static AirQualityRecoder createOne(){
        try {
            Thread.sleep(new Random().nextInt(3000));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String[] citys = new String[]{"天津","北京","上海","西安","深圳","广州"};
        AirQualityRecoder aqv = new AirQualityRecoder();
        Random r = new Random();
        aqv.setCity(citys[r.nextInt(6)]);
        aqv.setId(aqv.getCity());
        aqv.setAirQuality(r.nextInt(10));
        aqv.setEmmit(new Date());
        aqv.setEt(System.currentTimeMillis());
        return aqv;
```

```
    }
}
```

接下来，写sink，这里包含两个内部类，SimpleGenerator 用于创建模拟数据，SimpleAirQualityRecoderSchema 用于sink数据，这里主要实现数据的序列化，与反序列化，以及定义元数据类型。 *这里，直接将对象以二进制形式写出去了，生产环境还是不建议这么做* 。

```
package wang.datahub.cep;
import org.apache.flink.api.common.serialization.DeserializationSchema;
import org.apache.flink.api.common.serialization.SerializationSchema;
import org.apache.flink.api.common.typeinfo.TypeHint;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.datastream.DataStreamSink;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.source.SourceFunction;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer010;
import wang.datahub.cep.event.AirQualityRecoder;
//import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer09;
import java.io.*;
import java.util.HashMap;
import java.util.Map;
public class WriteIntoKafka {
    public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecution
Environment();
        Map prop = new HashMap();
        prop.put("bootstrap.servers", "localhost:9092");
        prop.put("topic", "test1");
        ParameterTool parameterTool = ParameterTool.fromMap(prop);
        DataStream<AirQualityRecoder> messageStream = env.addSource(new SimpleGe
nerator());
        DataStreamSink<AirQualityRecoder> airQualityVODataStreamSink = messageSt
ream.addSink(new FlinkKafkaProducer010<>(parameterTool.getRequired("bootstrap.se
rvers"),
                parameterTool.getRequired("topic"),
                new SimpleAirQualityRecoderSchema()));
        messageStream.print();
        env.execute("write to kafka !!!");
    }

    public static class SimpleGenerator implements SourceFunction<AirQualityReco
der>{
        private static final long serialVersionUID = 1L;
        boolean running = true;
        @Override
        public void run(SourceContext<AirQualityRecoder> ctx) throws Exception {
            while(running) {
                ctx.collect(AirQualityRecoder.createOne());
            }
```

```
        }

        @Override
        public void cancel() {
            running = false;
        }

    }

    public static class SimpleAirQualityRecoderSchema implements DeserializationSchema<AirQualityRecoder>, SerializationSchema<AirQualityRecoder>{
        @Override
        public AirQualityRecoder deserialize(byte[] message) throws IOException{
            //System.out.println(new String(message));
            ByteArrayInputStream bi = new ByteArrayInputStream(message);
            ObjectInputStream oi = new ObjectInputStream(bi);
            AirQualityRecoder obj = null;
            try {
                obj = (AirQualityRecoder)oi.readObject();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
            bi.close();
            oi.close();
            return obj;
        }

        @Override
        public boolean isEndOfStream(AirQualityRecoder nextElement) {
            return false;
        }

        @Override
        public byte[] serialize(AirQualityRecoder element) {
            byte[] bytes = null;
            try {

                ByteArrayOutputStream bo = new ByteArrayOutputStream();
                ObjectOutputStream oo = new ObjectOutputStream(bo);
                oo.writeObject(element);
                bytes = bo.toByteArray();
                bo.close();
                oo.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return bytes;
        }

        @Override
        public TypeInformation<AirQualityRecoder> getProducedType() {
            return TypeInformation.of(new TypeHint<AirQualityRecoder>(){});
        }
```

```
    }

}
```

在讲解CEP之前，还是先定义两个POJO，来做数据存储，

一个用于存放前后数据的对比记录

```
package wang.datahub.cep.event;
public class AirWarningRecoder {
    private String city;
    private AirQualityRecoder first;
    private AirQualityRecoder second;
    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public AirQualityRecoder getFirst() {
        return first;
    }

    public void setFirst(AirQualityRecoder first) {
        this.first = first;
    }

    public AirQualityRecoder getSecond() {
        return second;
    }

    public void setSecond(AirQualityRecoder second) {
        this.second = second;
    }

    public AirWarningRecoder(AirQualityRecoder first, AirQualityRecoder second)
{
        this.first = first;
        this.second = second;
    }

    @Override
    public String toString() {
        return "AirWarningRecoder{" +
                "city='" + city + '\'' +
                ", first=" + first +
                ", second=" + second +
                '}';
    }
```

```
    public AirWarningRecoder(String city, AirQualityRecoder first, AirQualityRec
oder second) {
        this.city = city;
        this.first = first;
        this.second = second;
    }

}
```

另一个用于存放，经过空气预警类型。

```
package wang.datahub.cep.event;
public class AirWarningTypeRecoder {
    private String city;
    private String wtype;
    private Integer first;
    private Integer second;
    @Override
    public String toString() {
        return "AirWarningTypeRecoder{" +
                "city='" + city + '\'' +
                ", wtype='" + wtype + '\'' +
                ", first=" + first +
                ", second=" + second +
                '}';
    }

    public Integer getFirst() {
        return first;
    }

    public void setFirst(Integer first) {
        this.first = first;
    }

    public Integer getSecond() {
        return second;
    }

    public void setSecond(Integer second) {
        this.second = second;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }
```

```
    public String getWtype() {
        return wtype;
    }

    public void setWtype(String wtype) {
        this.wtype = wtype;
    }
}
```

下面就是具体的CEP细节流程，首先我们需要定义Pattern，用于识别预警数据，第二个Pattern则没做操作，直接将数据交个下一个处理步骤。

然后将pattern和数据流注册给CEP，再对起进行select和map操作

```
package wang.datahub.cep;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.cep.CEP;
import org.apache.flink.cep.PatternStream;
import org.apache.flink.cep.pattern.Pattern;
import org.apache.flink.cep.pattern.conditions.IterativeCondition;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer010;
import wang.datahub.cep.event.AirQualityRecoder;
import wang.datahub.cep.event.AirWarningRecoder;
import wang.datahub.cep.event.AirWarningTypeRecoder;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
public class CepApp {
    public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecution
Environment();
        Map properties= new HashMap();
        properties.put("bootstrap.servers", "localhost:9092");
        properties.put("group.id", "test");
        properties.put("enable.auto.commit", "true");
        properties.put("auto.commit.interval.ms", "1000");
        properties.put("auto.offset.reset", "earliest");
        properties.put("session.timeout.ms", "30000");
//        properties.put("key.deserializer", "org.apache.kafka.common.serializat
ion.StringDeserializer");
//        properties.put("value.deserializer", "org.apache.kafka.common.serializ
ation.StringDeserializer");
        properties.put("topic", "test1");
        ParameterTool parameterTool = ParameterTool.fromMap(properties);
        FlinkKafkaConsumer010 consumer010 = new FlinkKafkaConsumer010(
                parameterTool.getRequired("topic"), new WriteIntoKafka.SimpleAir
QualityRecoderSchema(), parameterTool.getProperties());
        DataStream<AirQualityRecoder> aqrStream = env
```

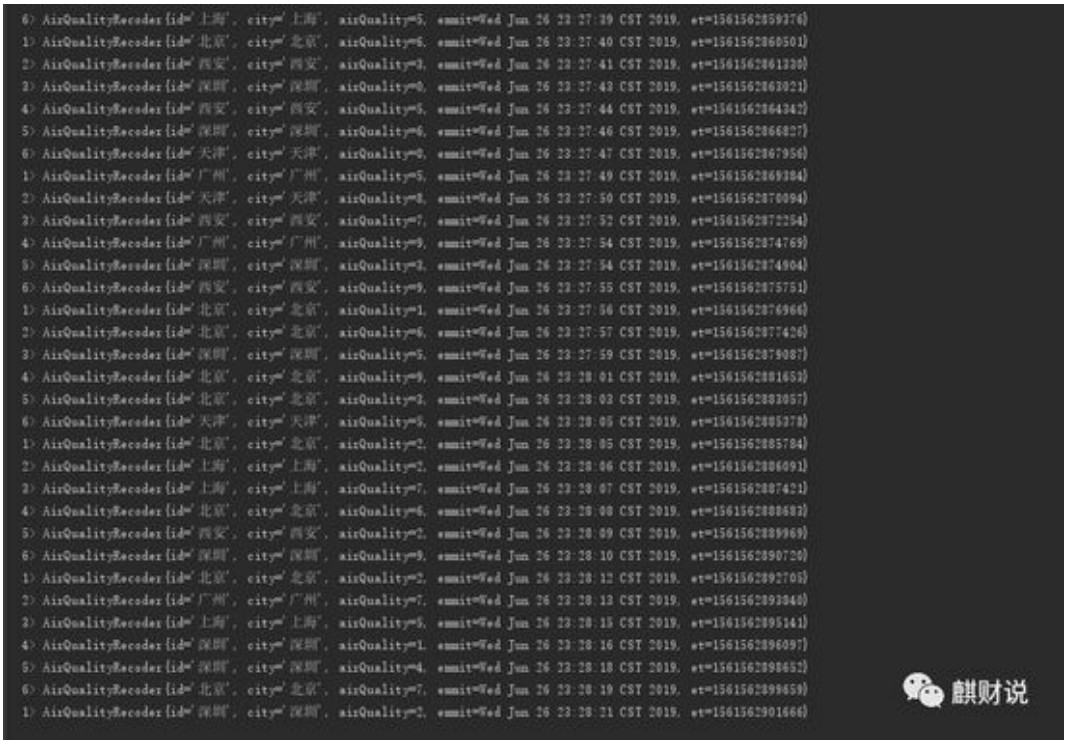
```
                .addSource(consumer010);
        Pattern<AirQualityRecoder, ?> warningPattern = Pattern.<AirQualityRecode
r>begin("first")
                .subtype(AirQualityRecoder.class)
                .where(new IterativeCondition<AirQualityRecoder>(){
                    @Override
                    public boolean filter(AirQualityRecoder value, Context<AirQu
alityRecoder> ctx) throws Exception {
                        return value.getAirQuality() >= 6;
                    }
                }).or(new IterativeCondition<AirQualityRecoder>(){
                    @Override
                    public boolean filter(AirQualityRecoder value, Context<AirQu
alityRecoder> ctx) throws Exception {
                        return value.getAirQuality() <= 3;
                    }
                })

                .next("second")
                .where(new IterativeCondition<AirQualityRecoder>(){
                    @Override
                    public boolean filter(AirQualityRecoder value, Context<AirQu
alityRecoder> ctx) throws Exception {
                        return value.getAirQuality() >= 7;
                    }
                }).or(new IterativeCondition<AirQualityRecoder>(){
                    @Override
                    public boolean filter(AirQualityRecoder value, Context<AirQu
alityRecoder> ctx) throws Exception {
                        return value.getAirQuality() <= 2;
                    }
                })
                .within(Time.seconds(60))
                ;
        PatternStream<AirQualityRecoder> warningPatternStream = CEP.pattern(
                aqrStream.keyBy("city"),//"city"
                warningPattern);
        DataStream<AirWarningRecoder> warnings = warningPatternStream.select(
                (Map<String, List<AirQualityRecoder>> pattern) -> {
                    AirQualityRecoder first = (AirQualityRecoder) pattern.get("f
irst").get(0);
                    AirQualityRecoder second = (AirQualityRecoder) pattern.get("
second").get(0);
                    return new AirWarningRecoder(first.getCity(),first,second);
                }
        );
        Pattern<AirWarningRecoder, ?> typePattern = Pattern.<AirWarningRecoder>b
egin("pass")
                .subtype(AirWarningRecoder.class);
        PatternStream<AirWarningRecoder> typePatternStream = CEP.pattern(
                warnings.keyBy(AirWarningRecoder::getCity),
                typePattern
        );
        DataStream<AirWarningTypeRecoder> awt = typePatternStream.select(
```

```
                (Map<String, List<AirWarningRecoder>> pattern) -> {
                    AirWarningRecoder awr = (AirWarningRecoder) pattern.get("pas
s").get(0);
                    AirWarningTypeRecoder awtr = new AirWarningTypeRecoder();
                    awtr.setCity(awr.getCity());
                    awtr.setFirst(awr.getFirst().getAirQuality());
                    awtr.setSecond(awr.getSecond().getAirQuality());
                    int res = Math.abs(awtr.getFirst()-awtr.getSecond());
                    if(res <=2){
                        awtr.setWtype("质量超标");
                    }else{
                        awtr.setWtype("波动较大");
                    }

                    return awtr;
                }
        );
        warnings.print();
        awt.print();
        env.execute("cep run!!!");
    }
}
```

生产数据截图



计算结果

好了，CEP就说的这，只是一些个人理解，如果对你有帮助，那是我莫大的荣幸，也请大家帮我勘正谬误。也欢迎大家与我交流。

参考:

【1】：http://aitozi.com/flink-cep-paper.html

【2】：https://zhuanlan.zhihu.com/p/37310327

【3】：https://www.jianshu.com/p/cee45f817757

以上就是本文的全部内容，希望本文的内容对大家的学习或者工作能带来一定的帮助，也希望大家多多支持