# Flink CEP 空气质量监控案例

转载至 about 云 http://www.aboutyun.com/thread-27487-1-1.html

实际业务场景代码会比这个复制，但是类似.....

```
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.getExecutionEnvironment();
        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.serialization.StringDeserializer");
//        properties.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        properties.put("topic", "test1");
        ParameterTool parameterTool = ParameterTool.fromMap(properties);
        FlinkKafkaConsumer010 consumer010 = new FlinkKafkaConsumer010(
                parameterTool.getRequired("topic"), new
WriteIntoKafka.SimpleAirQualityRecoderSchema(), parameterTool.getProperties());
        DataStream<AirQualityRecoder> aqrStream = env
                .addSource(consumer010);
        Pattern<AirQualityRecoder, ?> warningPattern = Pattern.
<AirQualityRecoder>begin("first")
                .subtype(AirQualityRecoder.class)
                .where(new IterativeCondition<AirQualityRecoder>(){
                    @Override
```

```
                public boolean filter(AirQualityRecoder value,
Context<AirQualityRecoder> ctx) throws Exception {
                    return value.getAirQuality() >= 6;
                }
            }).or(new IterativeCondition<AirQualityRecoder>(){
                @Override
                public boolean filter(AirQualityRecoder value,
Context<AirQualityRecoder> ctx) throws Exception {
                    return value.getAirQuality() <= 3;
                }
            })

            .next("second")
            .where(new IterativeCondition<AirQualityRecoder>(){
                @Override
                public boolean filter(AirQualityRecoder value,
Context<AirQualityRecoder> ctx) throws Exception {
                    return value.getAirQuality() >= 7;
                }
            }).or(new IterativeCondition<AirQualityRecoder>(){
                @Override
                public boolean filter(AirQualityRecoder value,
Context<AirQualityRecoder> 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("first").get(0);
                AirQualityRecoder second = (AirQualityRecoder)
pattern.get("second").get(0);
                return new AirWarningRecoder(first.getCity(),first,second);
            }
        );
        Pattern<AirWarningRecoder, ?> typePattern = Pattern.<AirWarningRecoder>begin("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("pass").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!!!");
    }
}
```