Flink CEP基础学习与使用03-标准案例代码书写

这里主要是案例，后续要结合实际业务做更复杂的场景，比如条件是否可以从redis里面获取？？？

完整案例：

```
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.cep.CEP;
import org.apache.flink.cep.PatternFlatSelectFunction;
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.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.IngestionTimeExtractor;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.util.Collector;


import java.util.List;
import java.util.Map;


public class CEPMonitoring {
    private static final double TEMPERATURE_THRESHOLD = 100;

    private static final int MAX_RACK_ID = 10;
    private static final long PAUSE = 100;
    private static final double TEMPERATURE_RATIO = 0.5;
    private static final double POWER_STD = 10;
    private static final double POWER_MEAN = 100;
    private static final double TEMP_STD = 20;
    private static final double TEMP_MEAN = 80;

    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();

        // Use ingestion time => TimeCharacteristic == EventTime + IngestionTimeExtractor
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

        // Input stream of monitoring events
        DataStream<MonitoringEvent> inputEventStream = env
                .addSource(new MonitoringEventSource(
                        MAX_RACK_ID,
                        PAUSE,
                        TEMPERATURE_RATIO,
```

```
                    POWER_STD,
                    POWER_MEAN,
                    TEMP_STD,
                    TEMP_MEAN))
                .assignTimestampsAndWatermarks(new IngestionTimeExtractor<>());

        // Warning pattern: Two consecutive temperature events whose temperature is higher than the given threshold
        // appearing within a time interval of 10 seconds
        Pattern<MonitoringEvent, ?> warningPattern = Pattern.<MonitoringEvent>begin("first")
                .subtype(TemperatureEvent.class)
                .where(new IterativeCondition<TemperatureEvent>() {
                    private static final long serialVersionUID = -6301755149429716724L;

                    @Override
                    public boolean filter(TemperatureEvent value, Context<TemperatureEvent> ctx) throws Exception {
                        return value.getTemperature() >= TEMPERATURE_THRESHOLD;
                    }
                })
                .next("second")
                .subtype(TemperatureEvent.class)
                .where(new IterativeCondition<TemperatureEvent>() {
                    private static final long serialVersionUID = 2392863109523984059L;

                    @Override
                    public boolean filter(TemperatureEvent value, Context<TemperatureEvent> ctx) throws Exception {
                        return value.getTemperature() >= TEMPERATURE_THRESHOLD;
                    }
                })
                .within(Time.seconds(10));

        // Create a pattern stream from our warning pattern
        PatternStream<MonitoringEvent> tempPatternStream = CEP.pattern(
                inputEventStream.keyBy("rackID"),
                warningPattern);

        // Generate temperature warnings for each matched warning pattern
        DataStream<TemperatureWarning> warnings = tempPatternStream.select(
            (Map<String, List<MonitoringEvent>> pattern) -> {
                TemperatureEvent first = (TemperatureEvent) pattern.get("first").get(0);
                TemperatureEvent second = (TemperatureEvent) pattern.get("second").get(0);

                return new TemperatureWarning(first.getRackID(), (first.getTemperature() + second.getTemperature()) / 2);
            }
        );

        // Alert pattern: Two consecutive temperature warnings appearing within a time interval of 20 seconds
        Pattern<TemperatureWarning, ?> alertPattern = Pattern.<TemperatureWarning>begin("first")
                .next("second")
                .within(Time.seconds(20));

```

```
        // Create a pattern stream from our alert pattern
        PatternStream<TemperatureWarning> alertPatternStream = CEP.pattern(
                warnings.keyBy("rackID"),
                alertPattern);

        // Generate a temperature alert only if the second temperature warning's average temperature is higher than
        // first warning's temperature
        DataStream<TemperatureAlert> alerts = alertPatternStream.flatSelect(
            (Map<String, List<TemperatureWarning>> pattern, Collector<TemperatureAlert> out) -> {
                TemperatureWarning first = pattern.get("first").get(0);
                TemperatureWarning second = pattern.get("second").get(0);

                if (first.getAverageTemperature() < second.getAverageTemperature()) {
                    out.collect(new TemperatureAlert(first.getRackID()));
                }
            },
            TypeInformation.of(TemperatureAlert.class));

        // Print the warning and alert events to stdout
        warnings.print();
        alerts.print();

        env.execute("CEP monitoring job");
    }
}
```

完整scala 代码案例：

```
import org.apache.flink.cep.scala.CEP
import org.apache.flink.cep.scala.pattern.Pattern
import org.apache.flink.streaming.api.scala._
import scala.collection.Map

object FlinkCEPSelect {

  case class Stock(volume: Int, price: Int)


  def main(args: Array[String]): Unit = {
    val env = StreamExecutionEnvironment.getExecutionEnvironment

    // 数据源
    val data: DataStream[Stock] = env.fromElements(
      Stock(100, 990),
      Stock(110, 980),
      Stock(120, 970),
      Stock(130, 800),
      Stock(140, 700),
      Stock(150, 600)
    ).setParallelism(1)

    // 定义模式
    val pattern = Pattern.begin[Stock]("start").where(_.volume > 110)
      .next("middle").subtype(classOf[Stock])
      .where((value, ctx) => {
```

```scala
        val startSum: Int = ctx.getEventsForPattern("start").map(_.price).sum
        val count = ctx.getEventsForPattern("start").size
        if (count > 0) {
          val sum = ctx.getEventsForPattern("middle").map(_.price).sum
          value.price > (sum + startSum) / count
        } else {
          value.price > startSum
        }
      }).oneOrMore
      .followedBy("end").subtype(classOf[Stock])
      .where((value, ctx) => {
        val count = ctx.getEventsForPattern("middle").size
        if (count > 0) {
          val stock = ctx.getEventsForPattern("middle").toList.apply(count - 1)
          value.volume < stock.volume * 0.8
        } else {
          false
        }

      })

    // 拿到结果
    val dataStream = CEP.pattern(data, pattern)

    // 将事件拼接在一起输出
    val result = dataStream.select((pat: Map[String, Iterable[Stock]]) => {
      val startEvent = pat.get("start").get.head
      val middleEvent = pat.get("middle").get.toList
      val endEvent = pat.get("end").get.head
      val startEventList: List[Stock] = List(startEvent)
      val endEventList: List[Stock] = List(endEvent)

      // 拼接起来输出
      val rs = startEventList ::: middleEvent ::: endEventList

      rs
    })
    result.print().setParallelism(1)

    env.execute()

  }


}
```