Flink CEP 订单案例

1，下面代码要注意的点：

1）函数可能会报错，需要引入scala Map

```
//todo 需要导入scala Map
import scala.collection.Map
val complexResult = patternStream.select(orderTimeoutOutput) {
//todo 统计超时的数据
  (pattern: Map[String, Iterable[OrderEvent]], timestamp: Long) => {
    val createOrder = pattern.get("begin")
    OrderResult(createOrder.get.iterator.next().orderId, "timeout",createOrder.get.iterator.next().
  }
} {
```


2）以为我们的条件是next 严格相邻，所以第一条数据会被过滤掉不满足条件，它不会出现在测流，就是被过滤掉了，如果想要测流拿到不满足的数据，正常应该用followBy或者其他模式;

```
  OrderEvent(1, "create", 1558430842,9001),
    OrderEvent(1, "create", 1558430813,9002), //模拟不在时间内
  OrderEvent(1, "create", 1558430813,9002),
  OrderEvent(1, "pay", 1558430844,9003),
  OrderEvent(2, "create", 1558430843,9004),
  OrderEvent(2, "pay", 1558430844,9005)
)).assignAscendingTimestamps(_.eventTime * 1000)
```


2，具体代码

```
1  package com.coder.flink.core.a_bilibili
2
3  import org.apache.flink.cep.scala.CEP
4  import org.apache.flink.cep.scala.pattern.Pattern
5  import org.apache.flink.streaming.api.TimeCharacteristic
6  import org.apache.flink.streaming.api.scala.{StreamExecutionEnvironment, _}
7  import org.apache.flink.streaming.api.windowing.time.Time
8
9  /**
10   * 订单实时预警
11   */
12
13
14 object OrderStreamingDemo {
15
```

```scala
  case class OrderEvent(orderId: Long, eventType: String, eventTime: Long,id : Long)

  case class OrderResult(orderId: Long, eventType: String,id : Long)

  def main(args: Array[String]): Unit = {

    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setParallelism(1)
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)

    val orderEventStream = env.fromCollection(List(
      OrderEvent(1, "create", 1558430842,9001),
//      OrderEvent(1, "create", 1558430813,9002), //模拟不在时间内
      OrderEvent(1, "create", 1558430813,9002),
      OrderEvent(1, "pay", 1558430844,9003),
      OrderEvent(2, "create", 1558430843,9004),
      OrderEvent(2, "pay", 1558430844,9005)
    )).assignAscendingTimestamps(_.eventTime * 1000)

    // 定义一个带匹配时间窗口的模式
    val orderPayPattern = Pattern.begin[OrderEvent]("begin")
      .where(_.eventType == "create")
      .next("next")
      .where(_.eventType == "pay")
      .within(Time.seconds(15))

    // 定义一个输出标签
    val orderTimeoutOutput = OutputTag[OrderResult]("orderTimeout")
    // 订单事件流根据 orderId 分流，然后在每一条流中匹配出定义好的模式
    val patternStream = CEP.pattern(orderEventStream.keyBy("orderId"), orderPayPattern)

    //todo 需要导入scala Map
    import scala.collection.Map
    val complexResult = patternStream.select(orderTimeoutOutput) {
    //todo 统计超时的数据
      (pattern: Map[String, Iterable[OrderEvent]], timestamp: Long) => {
        val createOrder = pattern.get("begin")
        OrderResult(createOrder.get.iterator.next().orderId,
"timeout",createOrder.get.iterator.next().id)
      }
    } {
      // 检测到定义好的模式序列时，就会调用这个函数
      (pattern: Map[String, Iterable[OrderEvent]]) => {
        val payOrder = pattern.get("next")
        val createOrder = pattern.get("begin")
        OrderResult(payOrder.get.iterator.next().orderId,
"success",createOrder.get.iterator.next().id)
      }
    }
    // 拿到同一输出标签中的 timeout 匹配结果（流） 订单超时
    val timeoutResult = complexResult.getSideOutput(orderTimeoutOutput)

//        complexResult.print()
    timeoutResult.print()

    env.execute("Order Timeout Detect Job")
```

```
  }


}
```