Flink双流处理:实时对账实现

更多内容详见:https://github.com/pierre94/flink-notes

一、基础概念

主要是两种处理模式:

- Connect/Join
- Union

二、双流处理的方法

Connect

`DataStream,DataStream → ConnectedStreams`

连接两个保持他们类型的数据流，两个数据流被Connect之后，只是被放在了一个同一个流中，内部依然保持各自的数据和形式不发生任何变化，两个流相互独立。

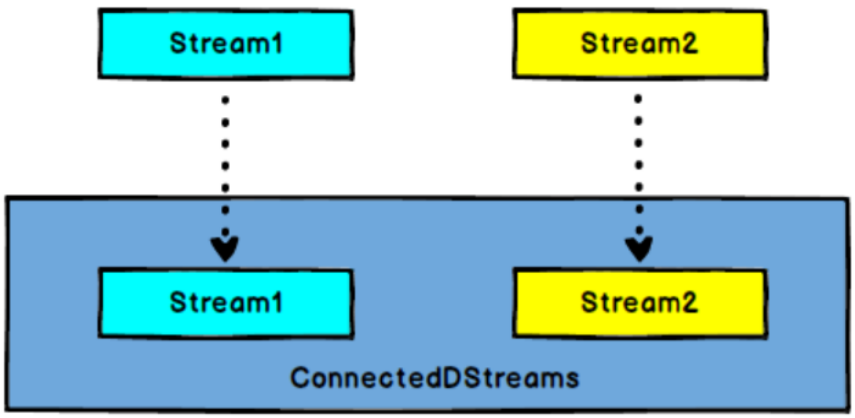

图 Connect 算子

Connect后使用CoProcessFunction、CoMap、CoFlatMap、KeyedCoProcessFunction等API 对两个流分别处理。如CoMap:

```
val warning = high.map( sensorData => (sensorData.id,
sensorData.temperature) )
val connected = warning.connect(low)

val coMap = connected.map(
warningData => (warningData._1, warningData._2, "warning"),
lowData => (lowData.id, "healthy")
)
```

(`ConnectedStreams → DataStream`功能与 map 一样，对 ConnectedStreams 中的每一个流分别进行 map 和 flatMap 处理。)

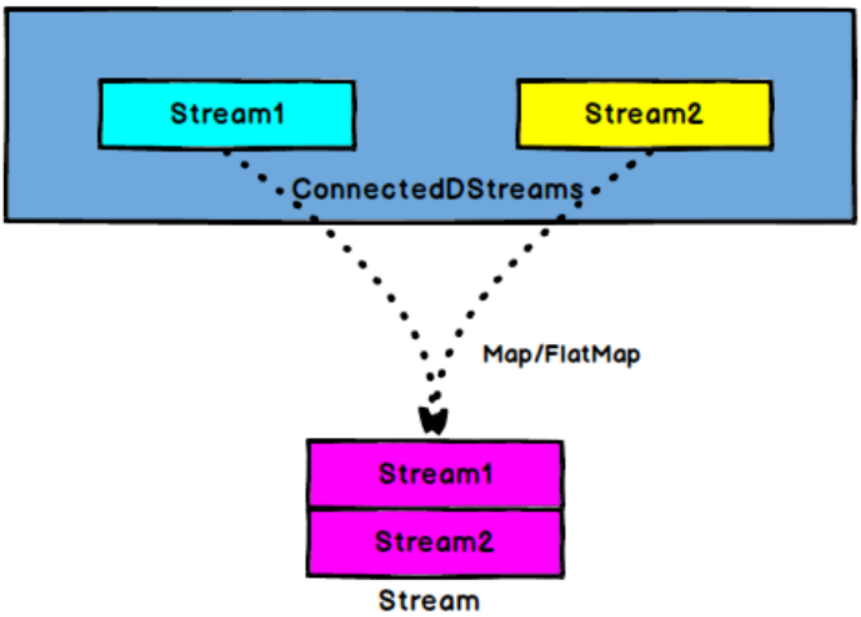

图 CoMap/CoFlatMap

> 疑问，既然两个流内部独立，那Connect 后有什么意义呢?

Connect后的两条流可以共享状态，在对账等场景具有重大意义!

Union

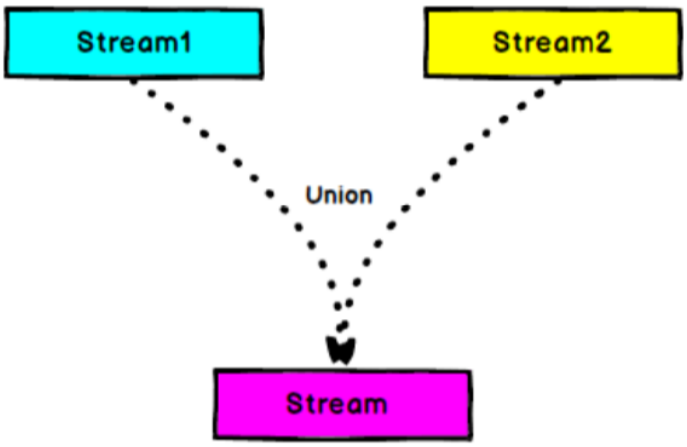

图 Union

`DataStream → DataStream`:对两个或者两个以上的 DataStream 进行 union 操作，产生一个包含所有 DataStream 元素的新 DataStream。

```
val unionStream: DataStream[StartUpLog] =
appStoreStream.union(otherStream) unionStream.print("union:::")
```

注意:Union 可以操作多个流，而Connect只能对两个流操作

Join

Join是基于Connect更高层的一个实现，结合Window实现。

相关知识点比较多，详细文档见: https://ci.apache.org/projects/flink/flink-docs-release-1.10/dev/stream/operators/joining.html

三、实战:实时对账实现

需求描述

有两个时间Event1、Event2,第一个字段是时间id，第二个字段是时间戳，需要对两者进行实时对账。当其中一个事件缺失、延迟时要告警出来。

需求分析

类似之前的订单超时告警需求。之前数据源是一个流，我们在function里面进行一些改写。这里我们分别使用Event1和Event2两个流进行Connect处理。

```
// 事件1
case class Event1(id: Long, eventTime: Long)
// 事件2
case class Event2(id: Long, eventTime: Long)
// 输出结果
case class Result(id: Long, warnings: String)
```

代码实现

> scala实现

涉及知识点:

- 双流Connect
- 使用OutputTag侧输出
- KeyedCoProcessFunction(processElement1、processElement2)使用
- ValueState使用
- 定时器onTimer使用

启动两个TCP服务:

```
nc -lh 9999
nc -lk 9998
```

> 注意:nc启动的是服务端、flink启动的是客户端

```
import java.text.SimpleDateFormat

import org.apache.flink.api.common.state.{ValueState,
ValueStateDescriptor}
import org.apache.flink.streaming.api.TimeCharacteristic
import org.apache.flink.streaming.api.functions.co.KeyedCoProcessFunction
import org.apache.flink.streaming.api.scala.{StreamExecutionEnvironment,
_}
import org.apache.flink.util.Collector

object CoTest {
  val simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy:HH:mm:ss")
  val txErrorOutputTag = new OutputTag[Result]("txErrorOutputTag")


  def main(args: Array[String]): Unit = {
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
    env.setParallelism(1)

    val event1Stream = env.socketTextStream("127.0.0.1", 9999)
      .map(data => {
        val dataArray = data.split(",")
        Event1(dataArray(0).trim.toLong,
simpleDateFormat.parse(dataArray(1).trim).getTime)
```

```
      }).assignAscendingTimestamps(_.eventTime * 1000L)
      .keyBy(_.id)

    val event2Stream = env.socketTextStream("127.0.0.1", 9998)
      .map(data => {
        val dataArray = data.split(",")
        Event2(dataArray(0).trim.toLong,
simpleDateFormat.parse(dataArray(1).trim).getTime)
      }).assignAscendingTimestamps(_.eventTime * 1000L)
      .keyBy(_.id)

    val coStream = event1Stream.connect(event2Stream)
      .process(new CoTestProcess())

    //    union 必须是同一条类型的流
    //    val unionStream = event1Stream.union(event2Stream)
    //    unionStream.print()

    coStream.print("ok")
    coStream.getSideOutput(txErrorOutputTag).print("txError")

    env.execute("union test")
  }

  //共享状态
  class CoTestProcess() extends KeyedCoProcessFunction[Long,Event1,
Event2, Result] {
    lazy val event1State: ValueState[Boolean]
    = getRuntimeContext.getState(new ValueStateDescriptor[Boolean]
("event1-state", classOf[Boolean]))

    lazy val event2State: ValueState[Boolean]
    = getRuntimeContext.getState(new ValueStateDescriptor[Boolean]
("event2-state", classOf[Boolean]))


    override def processElement1(value: Event1, ctx:
KeyedCoProcessFunction[Long, Event1, Event2, Result]#Context, out:
Collector[Result]): Unit = {
      if (event2State.value()) {
        event2State.clear()
        out.collect(Result(value.id, "ok"))
      } else {
```

```
        event1State.update(true)
        //等待一分钟
        ctx.timerService().registerEventTimeTimer(value.eventTime + 1000L
* 60)
      }
    }

    override def processElement2(value: Event2, ctx:
KeyedCoProcessFunction[Long, Event1, Event2, Result]#Context, out:
Collector[Result]): Unit = {
      if (event1State.value()) {
        event1State.clear()
        out.collect(Result(value.id, "ok"))
      } else {
        event2State.update(true)
        ctx.timerService().registerEventTimeTimer(value.eventTime + 1000L
* 60)
      }
    }

    override def onTimer(timestamp: Long, ctx:
KeyedCoProcessFunction[Long, Event1, Event2, Result]#OnTimerContext, out:
Collector[Result]): Unit = {
      if(event1State.value()){
        ctx.output(txErrorOutputTag,Result(ctx.getCurrentKey,s"no
event2,timestamp:$timestamp"))
        event1State.clear()
      }else if(event2State.value()){
        ctx.output(txErrorOutputTag,Result(ctx.getCurrentKey,s"no
event1,timestamp:$timestamp"))
        event2State.clear()
      }
    }
  }

}
```

《Flink状态编程: 订单超时告警》：
https://blog.csdn.net/u013128262/article/details/104648592

《github:Flink学习笔记》：
https://github.com/pierre94/flink-notes