**Flink 单并行度内使用多线程来提高作业性能**

之前在这篇文章中提到的在算子内部多线程处理数据，我看案例中没有使用到flink自带的一些对象，比如ctx和out，现在我将待处理的数据和这两个flink自己的对象一同放到缓冲队列中，后面由线程不断去队列中获取数据，再进行一些处理，处理完之后，通过ctx和out输出不同的侧边流和主流，我运行时报错了，不知道是不是因为这两个对象的原因。而且我这个算子计算完数据之后，还需要发送到下个算子。

错误信息：

java.lang.NullPointerException at

org.apache.flink.streaming.api.operators.co.CoBroadcastWithNonKeyedOperator$ReadOnlyContextImpl.output(CoBroadcastWithNonKeyedOperator.java:214)

at cn.com.bonc.core.operation.RuleHandleRunnable.run(RuleHandleRunnable.scala:36)

at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)

at java.util.concurrent.FutureTask.run(FutureTask.java:266)

at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(ScheduledThreadPoolExecutor.java:180)

at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)

at java.lang.Thread.run(Thread.java:748)

缓冲队列类型代码：

/\*\*  
 \* 处理信息，包括实时数据、规则、ctx、out对象等  
 \*  
 \* @author wzq  
 \* @date 2019-11-27  
 \*\*/  
case class HandleMessage(  
 var jedis: JedisCluster,  
 var warningRule: util.Map[String, Rule],  
 var value: Tuple2[String, String],  
 var addPointNameValue: String,  
 var faultStandbyRule: Tuple2[util.Set[String], util.Map[String, util.Map[String, util.List[FaultStandbyRule]]]],  
 var out: Collector[(String, String)],  
 var ctx: BroadcastProcessFunction[Tuple2[String, String], Tuple2[String, Any], (String, String)]#ReadOnlyContext  
 ) {  
}

自定义线程类：

class RuleHandleRunnable extends Runnable {  
 override def run(): Unit = {  
 while (true) {  
 val handleMessage = pendingData.take()  
 val out = handleMessage.out  
 val ctx = handleMessage.ctx  
 val value = handleMessage.value  
 try {  
 val handleResult = RuleHandle.ruleHandle(handleMessage.jedis, handleMessage.warningRule, value.f0, handleMessage.addPointNameValue, handleMessage.faultStandbyRule)  
 if (Objects.equals(handleResult.tag, "C")) {  
 //主流中存放处理之后的C类型数据  
 out.collect((value.f0, handleResult.fullData))  
 } else if (Objects.equals(handleResult.tag, "A")) {  
 //该侧边流存放处理之后的A类型数据  
 ctx.output(fullDataASide, handleResult.fullData)  
 } else if (Objects.equals(handleResult.tag, "B")) {  
 //该侧边流存放处理之后的B类型数据  
 ctx.output(fullDataBSide, (value.f0, handleResult.fullData))  
 }  
 val iter: util.Iterator[String] = handleResult.warningData.iterator()  
 while (iter.hasNext) {  
 //该侧边流存放报警数据  
 ctx.output(warningSide, iter.next())  
 }  
 val faultStandbyData = handleResult.faultStandbyData  
 if (faultStandbyData != null) {  
 //该侧边流存放故障待机判断数据  
 ctx.output(faultStandbySide, faultStandbyData)  
 }  
 } catch {  
 case e: Throwable => MysqlUtils.insertInto("规则判断处理", e, "实时数据", value.f0, value.f1)  
 }  
 }  
 }  
}

process算子处理实时数据关键代码：

override def processElement(value: Tuple2[String, String], ctx: BroadcastProcessFunction[Tuple2[String, String], Tuple2[String, Any], (String, String)]#ReadOnlyContext, out: Collector[(String, String)]): Unit = {  
 //1. 处理测点名称  
 var addPointNameValue: String = null  
 if (pointInfo != null) {  
 try {  
 addPointNameValue = RuleHandle.pointNameHandle(pointInfo, value.f0, value.f1)  
 } catch {  
 case e: Throwable => MysqlUtils.insertInto("测点名称处理", e, "实时数据", value.f0, value.f1)  
 }  
 }  
 //2. 处理报警和故障待机判断  
 if (warningRule != null && addPointNameValue != null) {  
 //报警和故障待机判断，均需要使用到redis，查询redis耗时较多，因此将该处理交由线程具体处理  
 pendingData.put(HandleMessage(jedis, warningRule, value, addPointNameValue, faultStandbyRule, out, ctx))  
 }  
}

通过debug调试，当自定义线程运行到ctx.output时会报错。