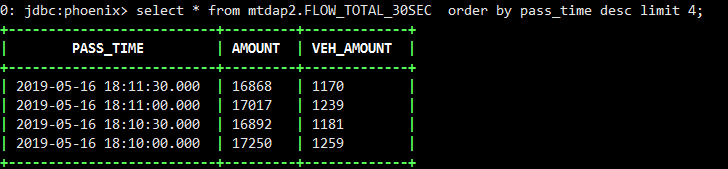
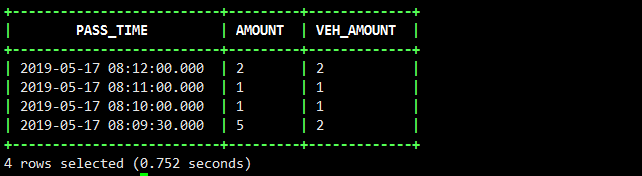
# Flink 常见异常整理

## 1:实时计算结果运行一段时间不正确

刚发布时是正常的:

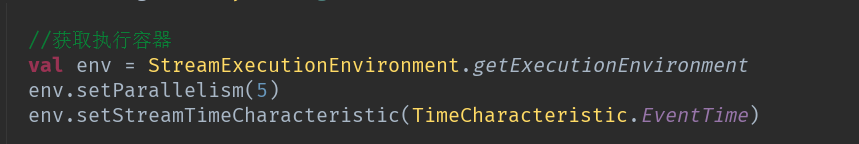


运行一段时间出现:



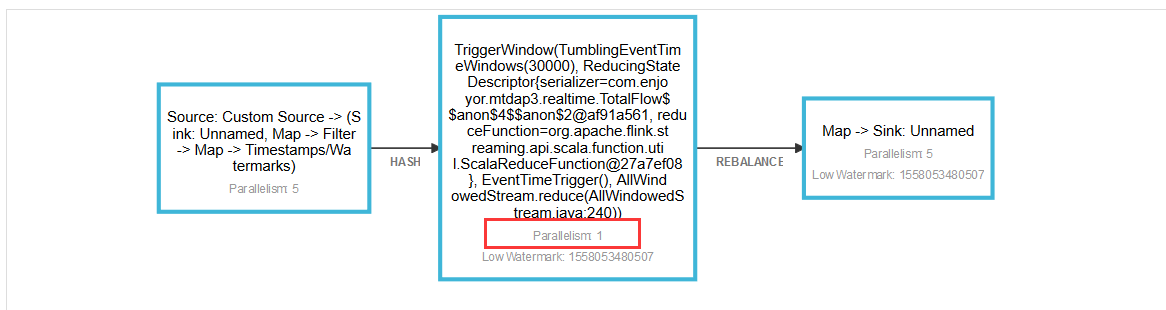
### 思路:资源不够

代码中设置并发度5



效果立竿见影,但第二天发现 问题仍然存在,只会异常出现时间延长了

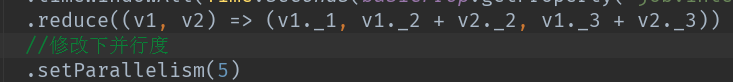
查看运行状况



接收和输出的都是5个slot 但是主要用作计算的算子还是使用1个并行度

Api的优先级: 设置parallelism有多中方式，优先级为api>env>p>file

尝试在代码中给Reduce算子设置并行度5



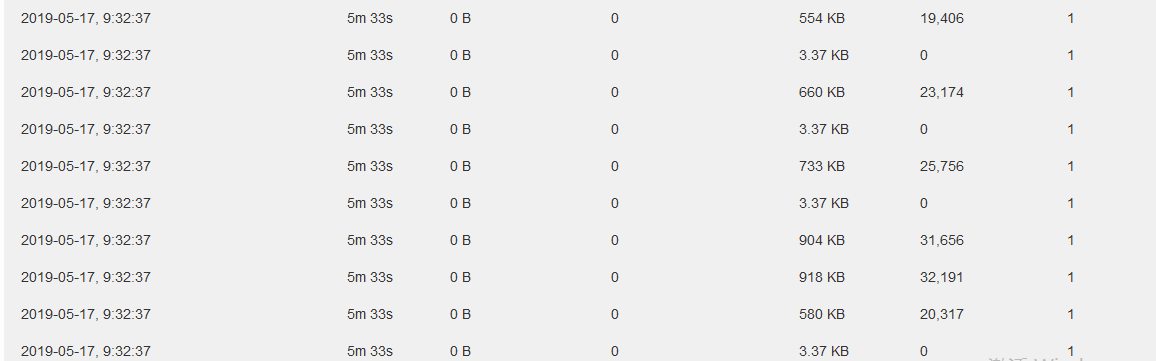
重新发布下:



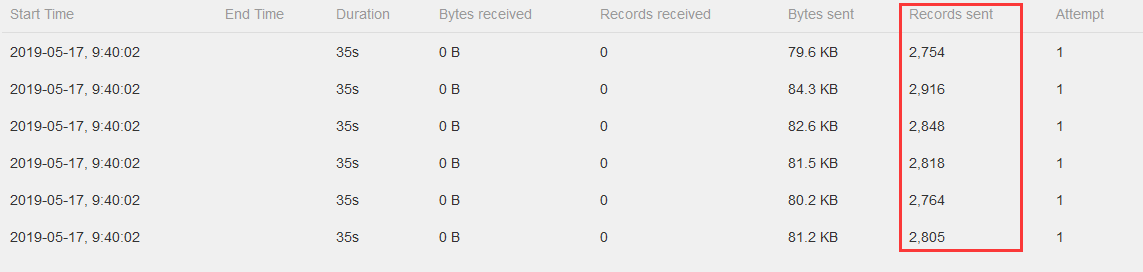
原因是flink的reduce算子为非并行度算子 只能设置为1

加大资源,设置并行度为10 ,指定keyBy(0) 按照过车时间分流

发布后查看Subtasks,发现数据倾斜,有4个并行度没有用上,占用4个slot

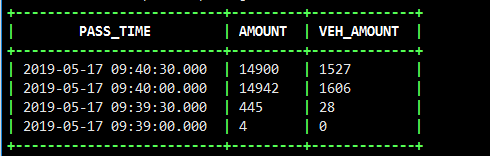


并行度设置为6,重新发布下,没有数据倾斜的现象



停掉的时候,这些计算量小的是怎么产生的?

上面两个是最新发布计算产生的,下面两个是关闭的时候?或者那时候已经计算异常了



监控一下数据异常的时间

### 代码优化

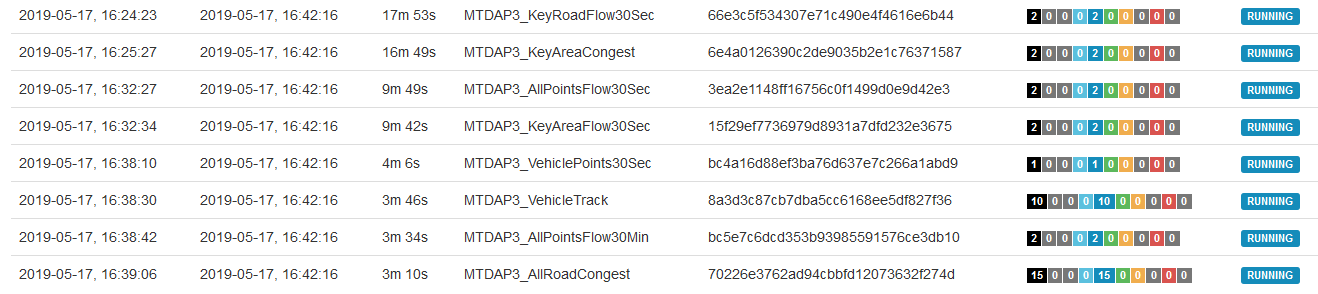
代码中将设置并行度去掉 同意使用-p 的方式指定并行度 同时使用keyBy算子按照过车时间对过车数据数据流进行分流操作

## 2:重点区域拥堵(redis满了)

将Redis的key同步后,实时程序出现:



下面这几个都有问题:



发现原因: org.apache.kafka.common.errors.TimeoutException: Timeout expired while fetching topic metadata

原因是kafka磁盘满了,一共500G 实时计算一天写40G数据

## 3. 也是kafka磁盘满导致问题

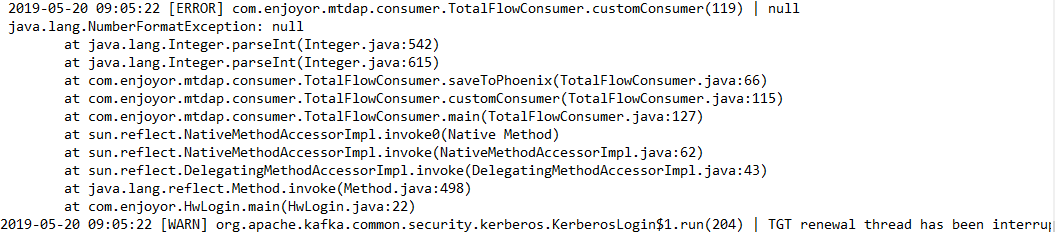
导致30033 ms has passed since batch creation plus linger time

程序运行一天后 直接挂掉



原因是kafka磁盘满了,一共500G 实时计算一天写40G数据

## 4.全流量消费者程序 发布报错



原因是全路段拥堵 的target发布到了mtdap-totalflow的consumer 上去

导致字段不一样

重新发布后,发现问题仍然存在,原因是消费组仍然接着消费之前offset的数据,从最新地方消费就好了

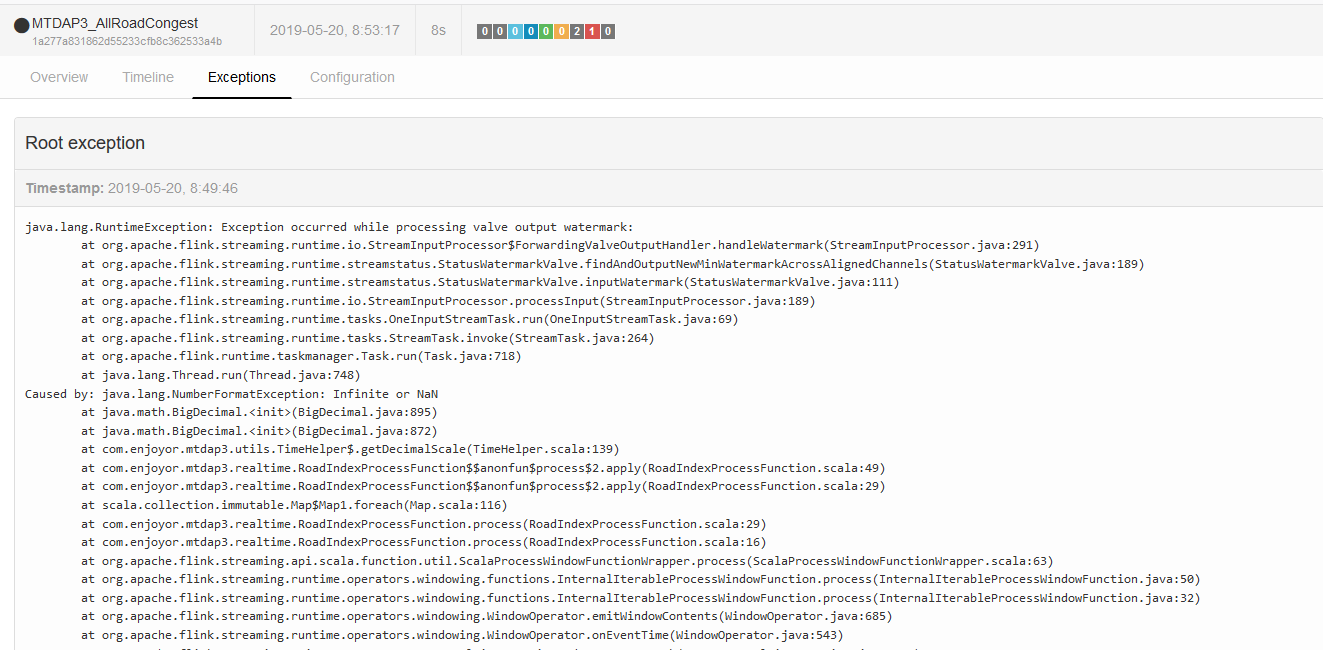
6.车辆轨迹计算 2小时后作业重启

抛; Lost connection to task manager 'NCJJ06/192.169.101.6:32391'. This indicates that the remote task manager was lost.

增加slot的数量,之前是5个slot 现在给定了18个,已经平稳运行13小时,继续跟踪下

## 5.全路段拥堵 发布不断重启

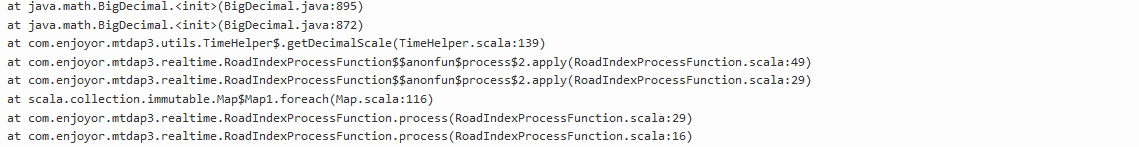
发布后程序一直在重启,依赖的key (23w条路段)



抛:处理输出水印时发生异常

java.lang.RuntimeException: Exception occurred while processing valve output watermark:

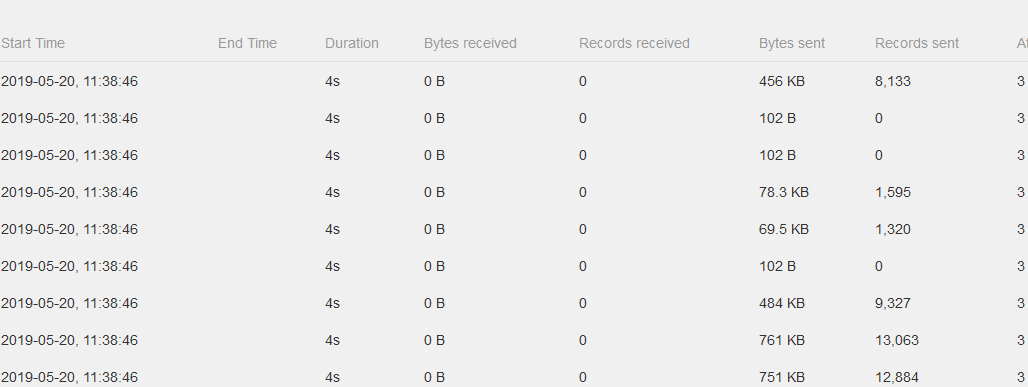
Caused by: java.lang.NumberFormatException: Infinite or NaN

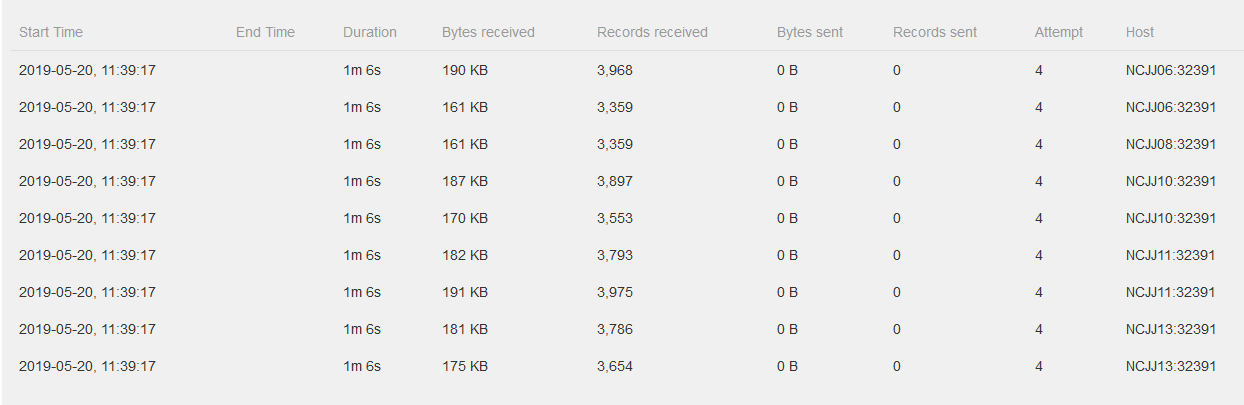
计算出的数据严重滞后

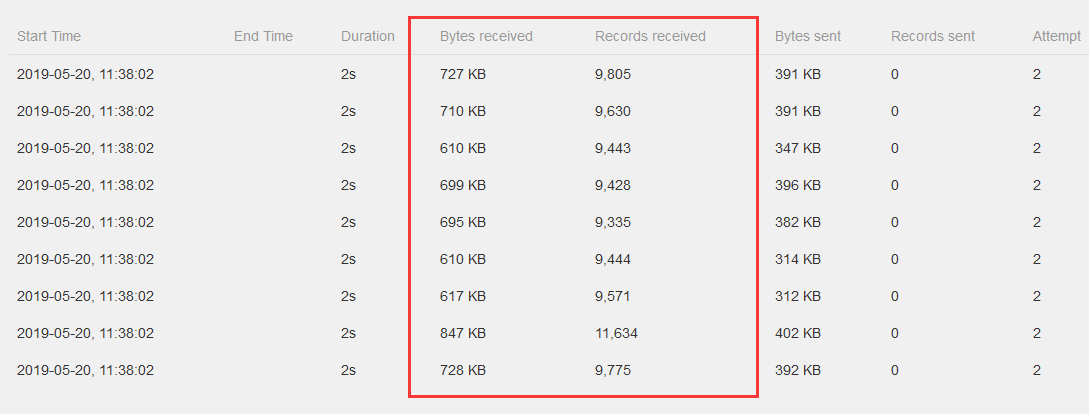
Infinite or NaN:无法格式化的数字，此数字要么不是个数字，要么是无穷大的数字，从而导致 BigDecimal 无法对其解析。

打印输出看下,发现频繁连接redis

路段23w 加点资源看看:





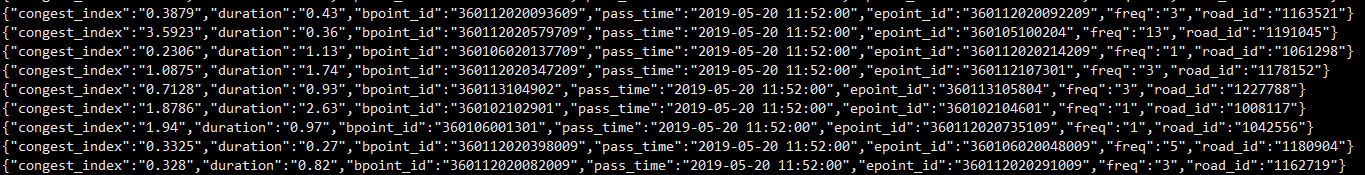


不是自愿的问题,代码中出现异常的脏数据,导致程序出错 能多执行一会,但是还是重启

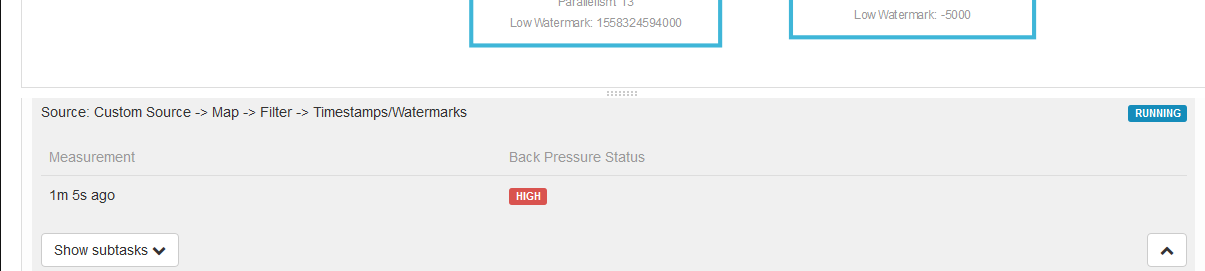
给定13个;

页面上可以出来了,但是程序一直重启???(跟踪下)

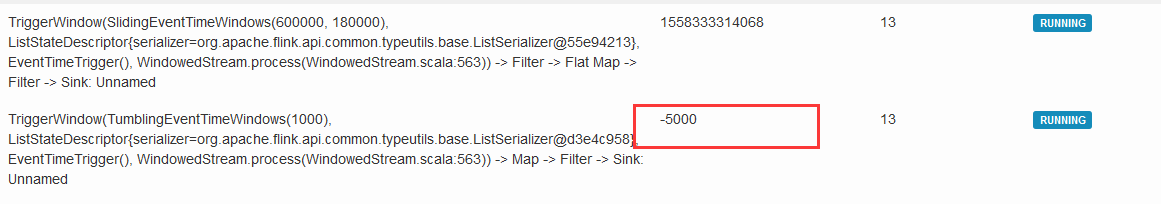
下午查看消费者,仍然在算1点多的数据:



背压太高:



水印输出有错:

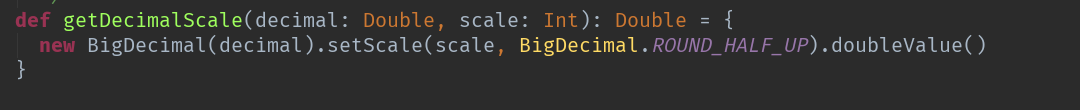


推测原因:

计算的路段太多,九江2w+ 南昌23w+,导致计算量太大

一直重启导致实时计算每次消费少量数据就挂掉,重启从checkpoint恢复后接着消费,出错挂掉重启 导致数据越积越多导致背压高

定位出错程序:



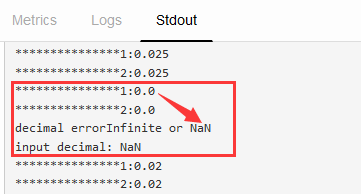
这段抛Infinite or NaN:

无法格式化的数字，此数字要么不是个数字，要么是无穷大的数字，从而导致 BigDecimal 无法对其解析。

跟CPU 没有关系:是代码中有问题

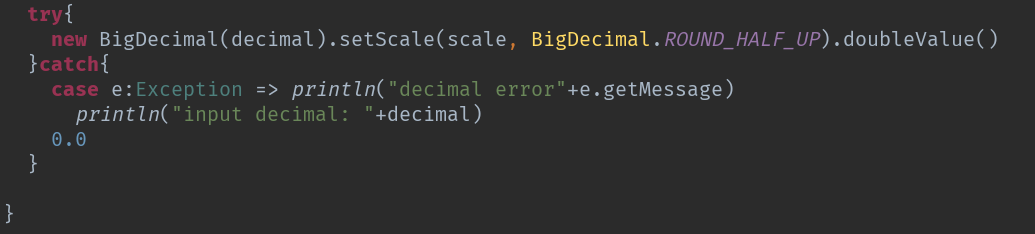
一个cpu内核分为3个slot 计算分配 50个内核 150个slot 每个程序10个

打印日志:



### 代码中更改:

捕获异常,NaN:not a number 0.0/0.0会出现NaN



重启redis服务,解决实时计算程序重启时重连Redis导致的redis最大连接数10000满了

跟踪下

### **问题原因**:

离线计算的全路段逻辑没处理好,出现路段bpoint为null, 之前程序没有这方面处理,直接调用工具类计算出现异常 挂掉,挂掉后触发Flink程序的未保证实时计算的稳定性的重启机制 不断重启(每次重启会连接redis,最终导致redis的最大连接数满了)

同时,挂掉后 导致下一次连接接着从kafka的topic消费 数据积压太多,产生反压现象,将解决问题的方向带偏,没有注意到导致程序出错到的核心问题

## 6: 连接redis命令行 出现Connection reset by peer

JedisDataException: ERR max number of clients reached

原因是Flink程序报错 重启 每一次都得连接 redis 达到最大连接数

端口不会断掉吗?之前的连接没有释放吗 ?

### 解决:重启redis服务

实时计算不断去redis拿数据的任务由于访问不了redis,重启 一次性拿到内存的没事

重启后查看redis的最大连接数:10000

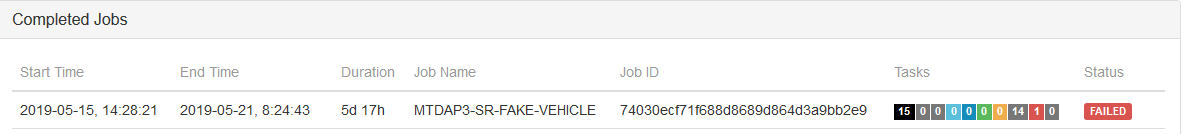


更改最大连接数,但是fusioninsight上不能直接修改

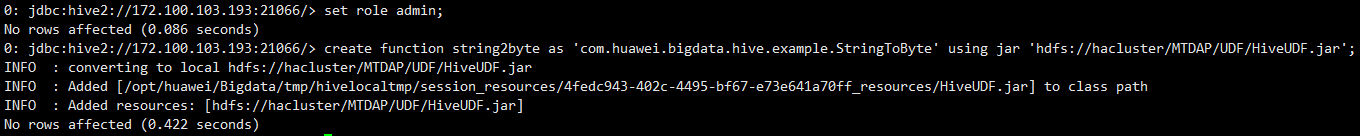


## 7: flink Connection reset by peer(原因未知)

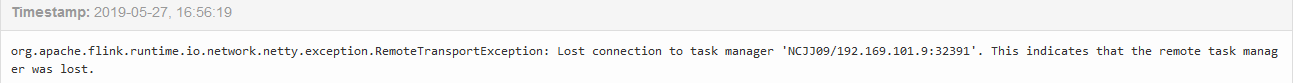
Lost connection to task manager 'host-redis01/\*.\*.\*.\*:port'. This indicates that the remote task manager was lost.



二次识别套牌车运行5天后挂掉

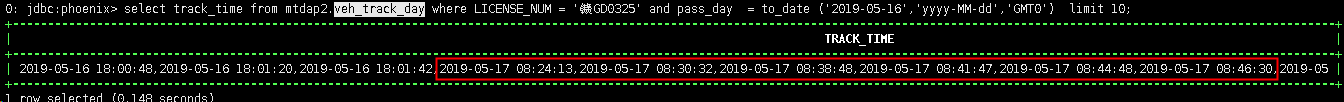


全路段拥堵,给定30个slot 还是会重启



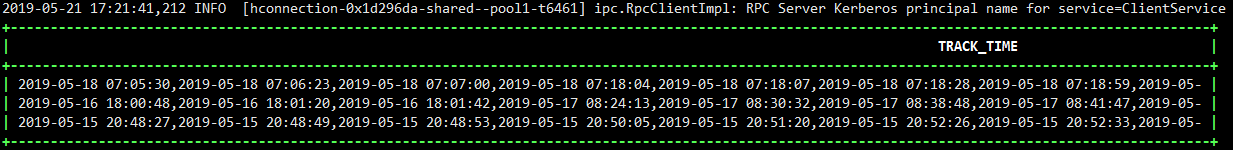
计算会滞后,导致

## 8:车辆轨迹 结果中过车时间异常,水位线不触发



每天一条轨迹,但是查询16号的轨迹,在过车时间出现17号

查看表:



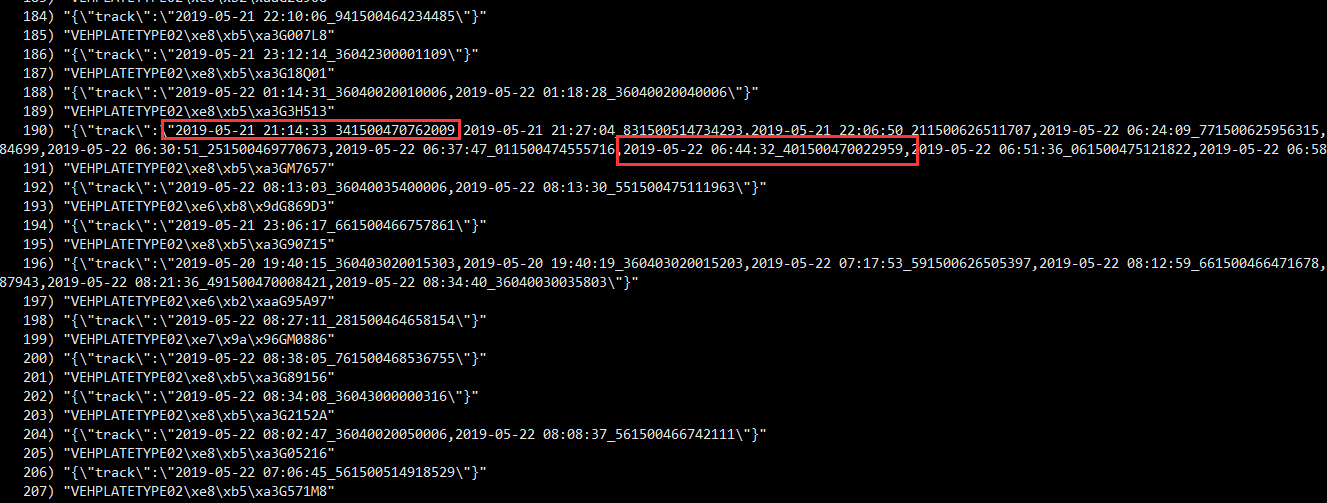
16号跟17号的轨迹是连在一起的

推测原因:中间的结果缓存是存到redis的 17号 redis 中还是16号的 没过期,但是设置的是有过期时间的 ,但是可能没过期(之前也是出现过这种情况的)

redis中作为临时存储的key: mtdap3\_vehicle\_track 设置的是午夜过期

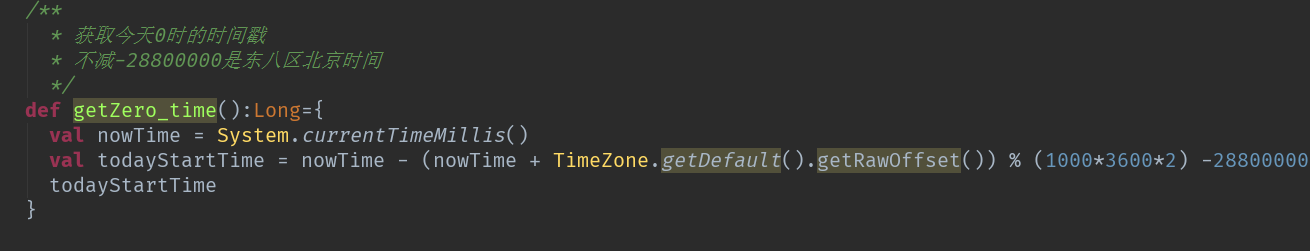
实时计算16号挂掉 redis中保存的还是16号的过车数据,17号实时计算执行 没有删掉过车轨迹的key 所以消费者跑完16号的跑17号,将过车轨迹追加到16号的后面 ,18号没有重复是因为key设置的是午夜自动过期,18号计算新产生一个过车轨迹key

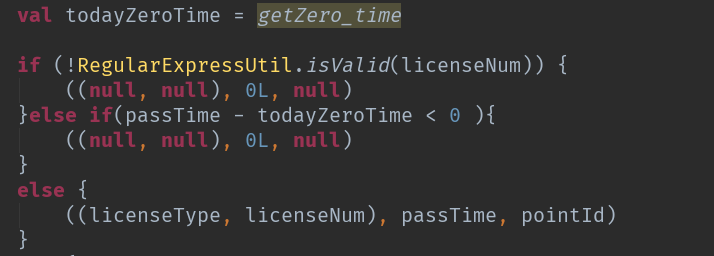
每天的key是正常过期的,但是传入的过车时间有可能是昨天的数据或前几天的数据,导致轨迹排序时,将这些加到过车轨迹的前面



在代码中将不是今天的过车数据过滤掉

将今天零点的之前的数据剔除掉

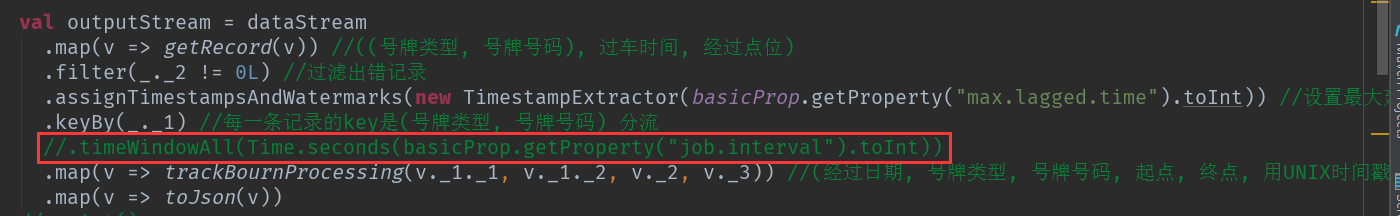




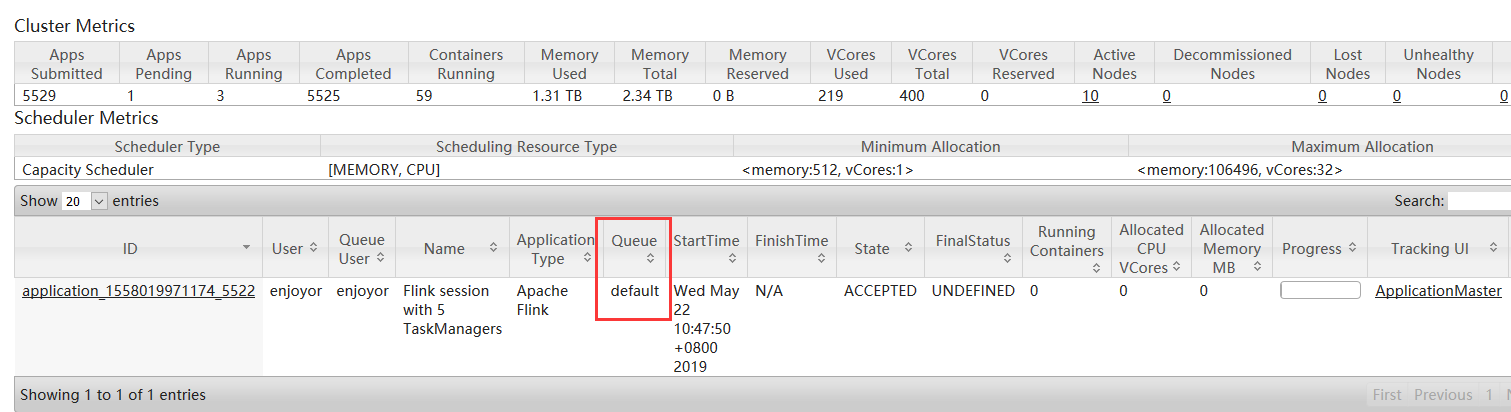
结果应该正常了,跟踪下

疑问: 当实时数据乱序到达时,水印应该会将事件时间延迟太久的这部分抛弃 不应该计算进去的???

水印没生效的原因:车辆轨迹没有使用窗口

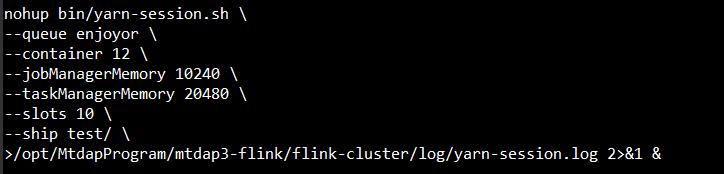


## 9:yarn发布失败



原因:default租户的资源已经用完了 租户分成default和enjoyor 如果不进行修改 就会使用默认的default资源队列,从页面看 default已经用了一半了 所以发布的任务一直等待

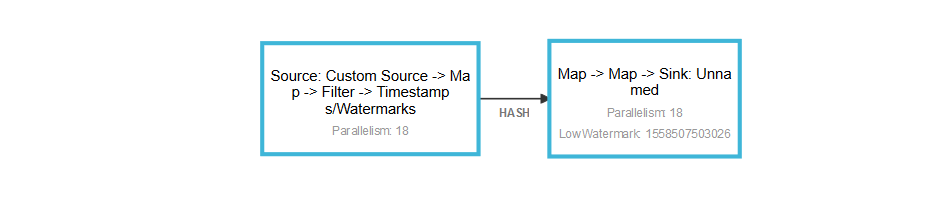
发布命令使用enjoyor租户



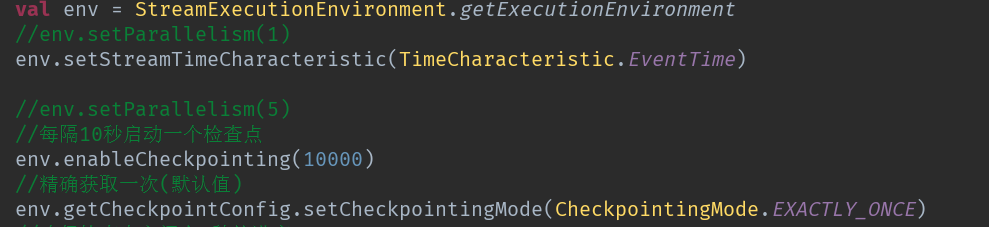
## 10.Flink车辆轨迹计算无窗口(未明白)

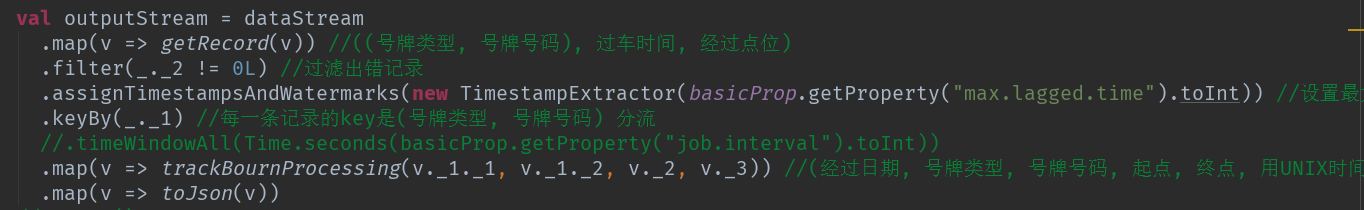
窗口是实时计算统计间隔的 程序中并没有使用窗口 底层是怎样算的 ?

Flink页面监控:并没有使用窗口



代码中





页面上先搞懂吧