对于集群这种无流量的作业，不在处理数据，需要通知到集群负责人，然后负责人去查看作业情况，并联系作业的负责人，看该作业是否可以下线，从而可以节省集群的资源。这样的作业通常有如下特征：

- 消费的 Kafka Topic 最近没有新的数据进来

要解决的问题主要是找到作业的 Source 算子，然后判断 Source 算子每个并行度是否在处理数据。

## 找到作业的所有 Source 算子

通过接口 `https://xxx.com/application_1575453055442_7402/jobs/bf8137982df2f809dc9c7f8cce67e415` 得知有哪些 vertices，返回数据如下：


```
{
    "jid": "bf8137982df2f809dc9c7f8cce67e415",
    "name": "apm etl job",
    "isStoppable": false,
    "state": "RUNNING",
    "start-time": 1594269192570,
    "end-time": -1,
    "duration": 11914144,
    "now": 1594281106714,
    "timestamps": {
        "CREATED": 1594269192570,
        "FINISHED": 0,
        "CANCELLING": 0,
        "RECONCILING": 0,
        "RUNNING": 1594278146775,
        "FAILING": 0,
        "CANCELED": 0,
        "FAILED": 0,
        "RESTARTING": 1594278122499,
        "SUSPENDED": 0
    },
    "vertices": [{
        "id": "bc764cd8ddf7a0cff126f51c16239658",
        "name": "Source: Custom Source",
        "parallelism": 16,
        "status": "RUNNING",
        "start-time": 1594278146815,
```

```
        "end-time": -1,
        "duration": 2959899,
        "tasks": {
            "FINISHED": 0,
            "CREATED": 0,
            "RUNNING": 16,
            "DEPLOYING": 0,
            "SCHEDULED": 0,
            "FAILED": 0,
            "CANCELING": 0,
            "CANCELED": 0,
            "RECONCILING": 0
        },
        "metrics": {
            "read-bytes": 0,
            "read-bytes-complete": true,
            "write-bytes": 149748954074,
            "write-bytes-complete": true,
            "read-records": 0,
            "read-records-complete": true,
            "write-records": 105065790,
            "write-records-complete": true
        }
    }, {
        "id": "20ba6b65f97481d5570070de90e4e791",
        "name": "Map -> Filter -> Map -> Filter -> Map -> Filter -> Filter -> (Filter -> Map -> Filter -> Sink: JDBCAppendTableSink(*), Filter -> Map -> Filter -> Sink: JDBCAppendTableSink(*), Filter -> Map -> Filter -> Sink: JDBCAppendTableSink(*), Filter -> Map -> Filter -> Sink: JDBCAppendTableSink(*))",
        "parallelism": 36,
        "status": "RUNNING",
        "start-time": 1594278146816,
        "end-time": -1,
        "duration": 2959898,
        "tasks": {
            "FINISHED": 0,
            "CREATED": 0,
            "RUNNING": 36,
            "DEPLOYING": 0,
            "SCHEDULED": 0,
            "FAILED": 0,
            "CANCELING": 0,
            "CANCELED": 0,
            "RECONCILING": 0
        },
        "metrics": {
            "read-bytes": 149757082709,
            "read-bytes-complete": true,
            "write-bytes": 0,
            "write-bytes-complete": true,
            "read-records": 105064278,
            "read-records-complete": true,
            "write-records": 0,
            "write-records-complete": true
        }
    }],
    "status-counts": {
        "FINISHED": 0,
```

```
        "CREATED": 0,
        "RUNNING": 2,
        "DEPLOYING": 0,
        "SCHEDULED": 0,
        "FAILED": 0,
        "CANCELING": 0,
        "CANCELED": 0,
        "RECONCILING": 0
    },
    "plan": {
        "jid": "bf8137982df2f809dc9c7f8cce67e415",
        "name": "apm etl job",
        "nodes": [{
            "id": "20ba6b65f97481d5570070de90e4e791",
            "parallelism": 36,
            "operator": "",
            "operator_strategy": "",
            "description": "Map -&gt; Filter -&gt; Map -&gt; Filter -&gt; Map -&gt; Filter -&gt; Filter -&gt; (Filter -&gt; Map -&gt; Filter -&gt; Sink: JDBCAppendTableSink(*), Filter -&gt; Map -&gt; Filter -&gt; Sink: JDBCAppendTableSink(*), Filter -&gt; Map -&gt; Filter -&gt; Sink: JDBCAppendTableSink(*), Filter -&gt; Map -&gt; Filter -&gt; Sink: JDBCAppendTableSink(*))",
            "inputs": [{
                "num": 0,
                "id": "bc764cd8ddf7a0cff126f51c16239658",
                "ship_strategy": "REBALANCE",
                "exchange": "pipelined_bounded"
            }],
            "optimizer_properties": {}
        }, {
            "id": "bc764cd8ddf7a0cff126f51c16239658",
            "parallelism": 16,
            "operator": "",
            "operator_strategy": "",
            "description": "Source: Custom Source",
            "optimizer_properties": {}
        }]
    }
}
```


拿到 vertices 数组，获取里面每个 vertice 的 name，查看是否有含 `Source` 的 vertice，如果有则查找这个 Source 的流数据，并拿到 Source 的并行度，然后通过下面的接口去判断 Source 的每个并行度是否在处理数据。

## 判断 Source 算子每个并行度是否在处理数据

举个例子，获取 Source 算子的所有并行度的流出速度：

```
https://xxx.com/proxy/application_1575453055442_7402/jobs/bf8137982df2f809dc9c7f8cce67e415/vertices/bc764cd8ddf7a0cff126f51c16239658/metrics?get=0.Source__Custom_Source.numRecordsOutPerSecond,1.Source__Custom_Source.numRecordsOutPerSecondt
```

返回值：

```
[
    {
        "id": "1.Source__Custom_Source.numRecordsOutPerSecond",
        "value": "2235.2"
    },
    {
        "id": "0.Source__Custom_Source.numRecordsOutPerSecond",
        "value": "2230.5833333333335"
    }
]
```

举个例子，获取 Source 算子的所有并行度的流出数据量：

```
https://xxx.com/proxy/application_1575453055442_7402/jobs/bf8137982df2f809dc9c7f8cce67e415/vertices/bc764cd8ddf7a0cff126f51c16239658/metrics?get=0.Source__Custom_Source.numRecordsOut,1.Source__Custom_Source.numRecordsOut
```

返回值：

```
[
    {
        "id": "0.Source__Custom_Source.numRecordsOut",
        "value": "6846032"
    },
    {
        "id": "1.Source__Custom_Source.numRecordsOut",
        "value": "6849504"
    }
]
```

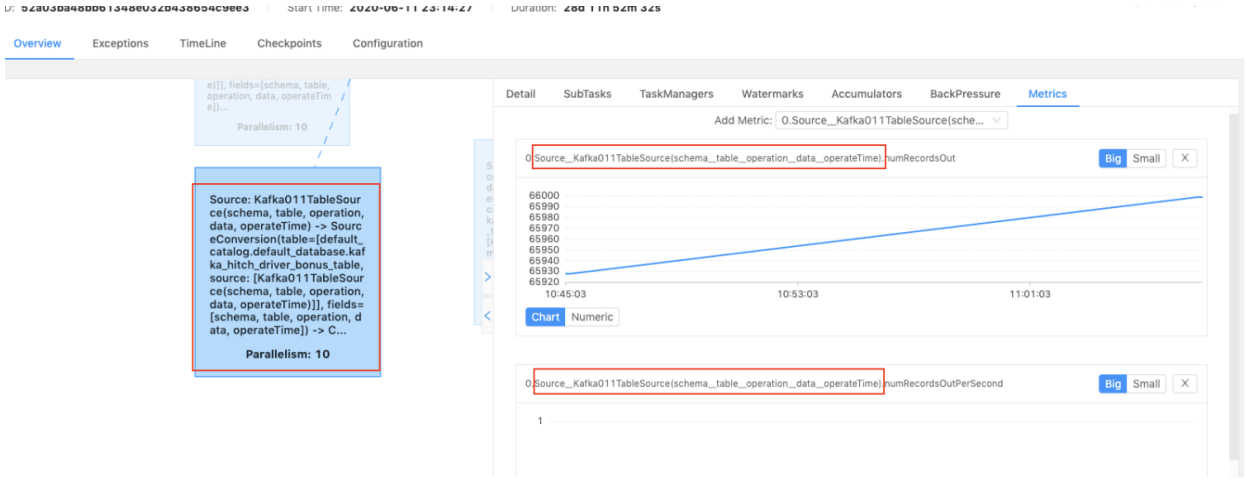
这里可以自己根据拿到的作业的所有 Source 算子的并行度数量，然后去分别构造拼接请求参数，生成最终的完整请求参数，将获取的返回值通过 JsonPath 去获取对应的 value 值进行统计求和，可以分别求出整个作业流入数据量。

但是对于有些 SQL 作业会比较复杂，并不是 `Source__Custom_Source` ，比如如下面图片所示的这种作业，它的完全就和前面的那些不一致，vertex name 如下：

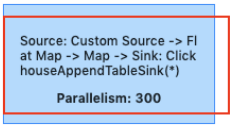
```
Source: Kafka011TableSource(schema, table, operation, data, operateTime) -> SourceConversion(table=[default_catalog.default_database.kafka_hitch_driver_bonus_table, source: [Kafka011TableSource(schema, table, operation, data, operateTime)]], fields=[schema, table, operation, data, operateTime]) -> Calc(select=[data.pax_journey_guid AS pax_journey_guid, data.award_amount AS award_amount], where=[((data.award_status = 2) AND NOT(data.is_delete))])
```

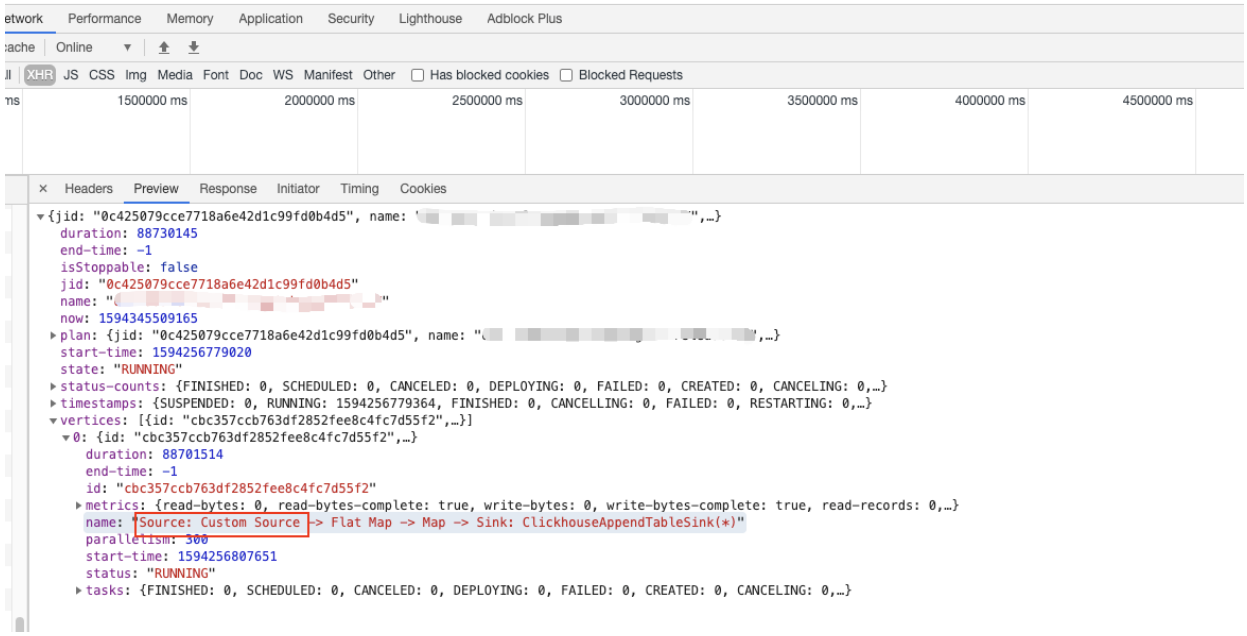
请求要的参数如下：

```
0.Source__Kafka011TableSource(schema__table__operation__data__operateTime).numRecordsOutPerSecond` 和 `0.Source__Kafka011TableSource(schema__table__operation__data__operateTime).numRecordsOut
```




还有的作业是算子会 chain 在一起的，




它的 vertex 的 name 是很长一串的，比如是 `Source: Custom Source -> Flat Map -> Map -> Sink: ClickhouseAppendTableSink(*)` ，而真正请求查看 Source 算子流出的流量的时候是使用的 `Source__Custom_Source` ，所以这里还需要做一个转换，先将 vertex name 通过 `->` 进行分割，判断长度，取数组第一个值，然后再将 `,` 和 空格进行替换成 `_` ，从而可以拿到我们最后真正请求的值了，这种切割和转换的操作应该可以适应所有的作业场景。

最后可以通过 Rest API 遍历访问集群中所有作业的任何时刻的流速和历史处理过去的数据量，从而可以对作业的流速和流量就行监控，如果对于长期无流量的作业则可以直接安排下线（在公司上线了这个功能后，立马找到 16 个这样无流量的作业，其中有个作业竟然运行 184 天，这个作业长期占用了 56GB 的内存资源）；对于流速很慢，并行度设置很高的作业也可以通知对应的负责人进行并行度的调整，有了这个信息其实还可以做更多有趣的事情，比如做作业动态扩缩容，其实都是可以的。

另外，除了可以通过 REST API 可以达到这种目的，另外还可以通过 Flink Metrcs Reporter 出去的数据数据也可以进行同样的操作，里面也有 metric 指标是描述上面这个流速和流量的。