

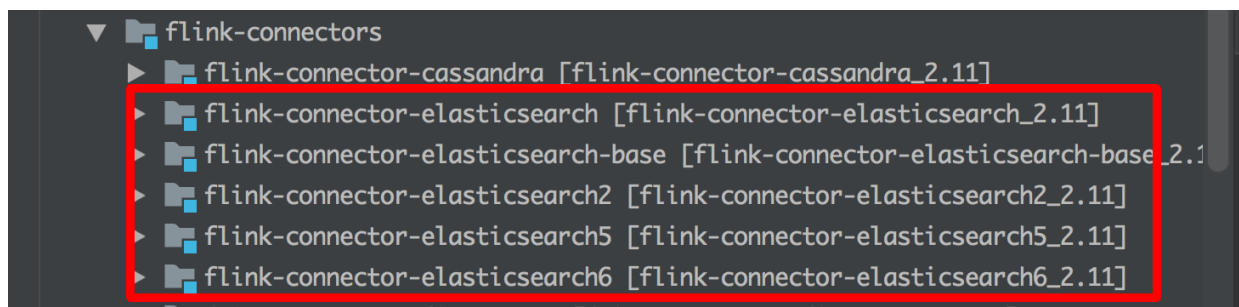
前言

在上一节 [Flink 读取 Kafka 数据处理后批量写入到 MySQL](#) 的总结中，我讲到我们生产环境中的数据量会很大，而对于 MySQL 是承受不了这种数据量写入，所以可以考虑换其他的存储中间件，比如 ES、HBase 等。不过到底存储在哪里，还是得看我们的业务需求和自己公司的情况，选择最适合自己的才是最好的。

我们这一节就是讲解一下如何 Flink 如何读取 Kafka 数据处理后写入到 ElasticSearch?

准备条件

因为在文章 [Flink 环境准备](#) 我已经讲过 ElasticSearch 的安装，这里就不做过多的重复，需要注意的一点就是 Flink 的 ElasticSearch Connector 是区分版本号的。



所以添加依赖的时候要区分一下，根据你安装的 ElasticSearch 来选择不一样的版本依赖，另外就是不同版本的 ElasticSearch 还会导致下面的数据写入到 ElasticSearch 中出现一些不同，我们这里使用的版本是 ElasticSearch6，如果你使用的是其他的版本可以参考官网的实现。

添加依赖

```
1 <dependency>
2   <groupId>org.apache.flink</groupId>
3   <artifactId>flink-connector-elasticsearch6_${scala.binary.version}</artifactId>
4   <version>${flink.version}</version>
5 </dependency>
```

上面这依赖版本号请自己根据使用的版本对应改变下。

下面所有的代码都没有把 import 引入到这里来，如果需要查看更详细的代码，请查看我的 GitHub 仓库地址：

<https://github.com/zhisheng17/flink-learning/tree/master/flink-learning-connectors/flink-learning-connectors-es6>

这个 module 含有本文的所有代码实现，后期可能因为代码做了部分重构，所以如果有代码改变很正常，请直接查看全部项目代码。

ElasticSearchSinkUtil 工具类

这个工具类是自己封装的，getEsAddresses 方法将传入的配置文件 es 地址解析出来，可以是域名方式，也可以是 ip + port 形式。addSink 方法是利用了 Flink 自带的 ElasticsearchSink 来封装了一层，传入了一些必要的调优参数和 es 配置参数，下面章节还会再讲些其他的配置。

ElasticSearchSinkUtil.java

```
1 public class ElasticSearchSinkUtil {
2
3     /**
4      * es sink
5      *
6      * @param hosts es hosts
7      * @param bulkFlushMaxActions bulk flush size
8      * @param parallelism 并行数
9      * @param data 数据
10     * @param func
11     * @param <T>
12     */
13     public static <T> void addSink(List<HttpHost> hosts, int
bulkFlushMaxActions, int parallelism,
14                                     SingleOutputStreamOperator<T> data,
ElasticsearchSinkFunction<T> func) {
15         ElasticsearchSink.Builder<T> esSinkBuilder = new
ElasticsearchSink.Builder<>>(hosts, func);
16         esSinkBuilder.setBulkFlushMaxActions(bulkFlushMaxActions);
17         data.addSink(esSinkBuilder.build()).setParallelism(parallelism);
18     }
19
20     /**
21      * 解析配置文件的 es hosts
22      *
23      * @param hosts
24      * @return
25      * @throws MalformedURLException
26      */
27     public static List<HttpHost> getEsAddresses(String hosts) throws
MalformedURLException {
28         String[] hostList = hosts.split(",");
29         List<HttpHost> addresses = new ArrayList<>();
30         for (String host : hostList) {
31             if (host.startsWith("http")) {
32                 URL url = new URL(host);
33                 addresses.add(new HttpHost(url.getHost(), url.getPort()));
34             } else {
35                 String[] parts = host.split(":", 2);
36                 if (parts.length > 1) {
37                     addresses.add(new HttpHost(parts[0],
Integer.parseInt(parts[1])));
38                 } else {
```

```

39         throw new MalformedURLException("invalid elasticsearch
hosts format");
40     }
41 }
42 }
43     return addresses;
44 }
45 }

```

Main 启动类

Main.java

```

1 public class Main {
2     public static void main(String[] args) throws Exception {
3         //获取所有参数
4         final ParameterTool parameterTool =
5 ExecutionEnvUtil.createParameterTool(args);
6         //准备好环境
7         StreamExecutionEnvironment env =
8 ExecutionEnvUtil.prepare(parameterTool);
9         //从kafka读取数据
10        DataStreamSource<Metrics> data = KafkaConfigUtil.buildSource(env);
11
12        //从配置文件中读取 es 的地址
13        List<HttpHost> esAddresses =
14 ElasticSearchSinkUtil.getEsAddresses(parameterTool.get(ELASTICSEARCH_HOSTS
15 ));
16        //从配置文件中读取 bulk flush size, 代表一次批处理的数量, 这个可是性能调优参
17        数, 特别提醒
18        int bulkSize =
19 parameterTool.getInt(ELASTICSEARCH_BULK_FLUSH_MAX_ACTIONS, 40);
20        //从配置文件中读取并行 sink 数, 这个也是性能调优参数, 特别提醒, 这样才能够更快
21        的消费, 防止 kafka 数据堆积
22        int sinkParallelism =
23 parameterTool.getInt(STREAM_SINK_PARALLELISM, 5);
24
25        //自己再自带的 es sink 上一层封装了下
26        ElasticSearchSinkUtil.addSink(esAddresses, bulkSize,
27 sinkParallelism, data,
28 (Metrics metric, RuntimeContext runtimeContext,
29 RequestIndexer requestIndexer) -> {
30            requestIndexer.add(Requests.indexRequest()
31                .index(ZHISHENG + "_" + metric.getName())
32                .type(ZHISHENG) //es type
33                .source(GsonUtil.toJSONBytes(metric),
34 XContentType.JSON));
35            });
36        env.execute("flink learning connectors es6");
37    }
38 }

```

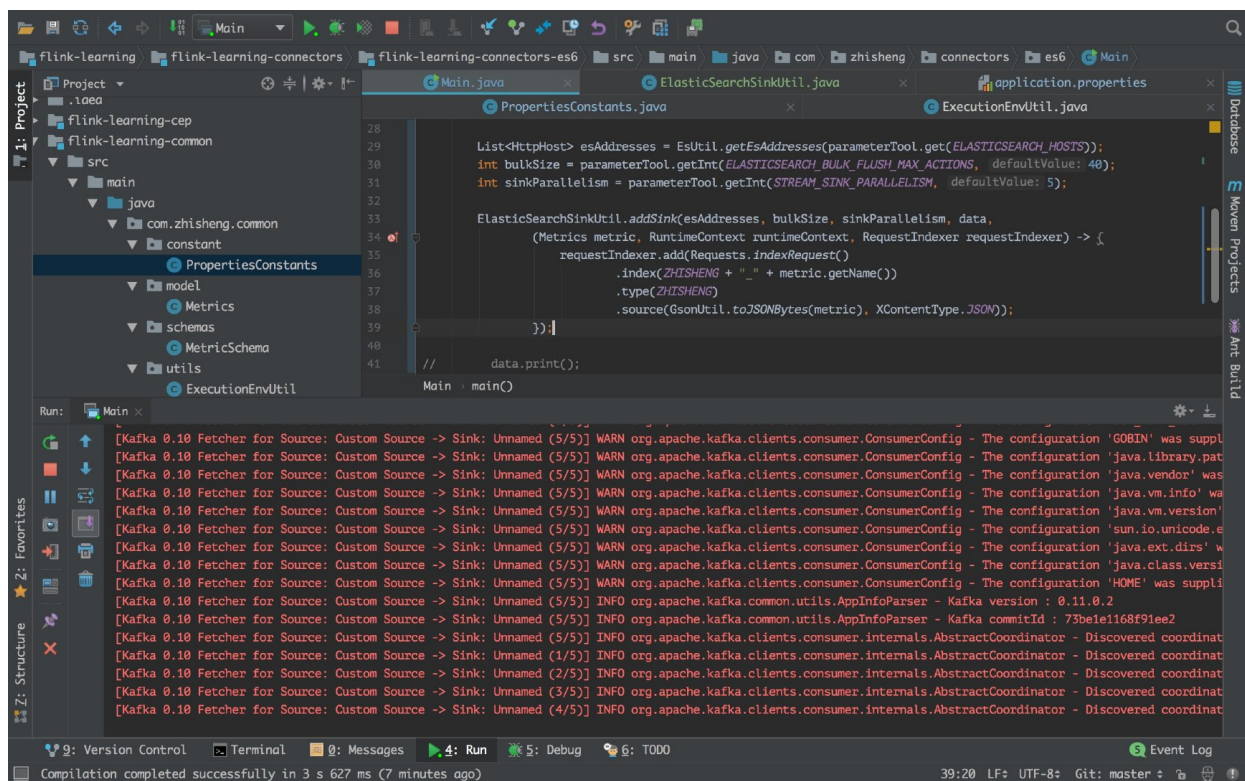
配置文件

配置都支持集群模式填写，注意用 `,` 分隔！

```
1 kafka.brokers=localhost:9092
2 kafka.group.id=zhisheng-metrics-group-test
3 kafka.zookeeper.connect=localhost:2181
4 metrics.topic=zhisheng-metrics
5 stream.parallelism=5
6 stream.checkpoint.interval=1000
7 stream.checkpoint.enable=false
8 elasticsearch.hosts=localhost:9200
9 elasticsearch.bulk.flush.max.actions=40
10 stream.sink.parallelism=5
```

运行结果

执行 Main 类的 main 方法，我们的程序是只打印 Flink 的日志，没有打印存入的日志（因为我们这里没有打日志）：



所以看起来不知道我们的 Sink 是否有用，数据是否从 Kafka 读取出来后存入到 ES 了。

你可以查看下本地起的 ES 终端或者服务器的 ES 日志就可以看到效果了。

ES 日志如下：


```

2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_metaserver_container][0]
: segment writing can't keep up
2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_metaserver_container][3]
: segment writing can't keep up
2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_docker_container_cpu][0]
: segment writing can't keep up
2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_docker_container_cpu][3]
: segment writing can't keep up
2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_docker_container_mem][0]
: segment writing can't keep up
2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_docker_container_mem][3]
: segment writing can't keep up
2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_docker_container_mem][1]
: segment writing can't keep up
2019-01-01T21:53:30,276][INFO ][o.e.i.IndexingMemoryController] [node-1] now throttling indexing for shard [[zhisheng_docker_container_mem][4]
: segment writing can't keep up
2019-01-01T21:53:31,461][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_metaserver_container][0]
]
2019-01-01T21:53:31,461][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_proctstat][2]]
2019-01-01T21:53:31,461][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_proctstat][0]]
2019-01-01T21:53:31,461][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_metaserver_container][3]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_docker_container_cpu][0]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_docker_container_mem][0]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_proctstat][3]]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_proctstat][4]]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_metaserver_container][1]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_docker_container_mem][4]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_metaserver_container][2]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_docker_container_mem][1]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_docker_container_mem][3]
]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_docker_container_cpu][3]
]
2019-01-01T21:53:31,463][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_proctstat][1]]
2019-01-01T21:53:31,463][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_metaserver_container][4]
]

```

上图是我本地 Mac 电脑终端的 ES 日志，可以看到我们的索引了。

如果还不放心，你也可以在你的电脑装个 Kibana，然后更加的直观查看下 ES 的索引情况（或者直接敲 ES 的命令）。

我们用 Kibana 查看存入 ES 的索引如下：

← → ↺

localhost:5601/app/kibana#/dev_tools/console?_g=()

kibana

Discover

Visualize

Dashboard

Timeline

APM

Dev Tools

Monitoring

Management

Console

Search Profiler

Grok Debugger

1 GET /_cat/indices?v

2

3 GET /_cat/health?v

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

1 health status index

2 yellow open

3 yellow open

4 yellow open

5 yellow open

6 yellow open

7 yellow open

8 yellow open

9 yellow open

10 yellow open

11 yellow open

12 yellow open

13 yellow open

14 yellow open

15 yellow open

16 yellow open

17 yellow open

18 yellow open

19 yellow open

store.size pri store.size

4.6mb

114.6mb

1.5mb

7.5mb

200.6mb

2.7mb

372.3kb

25.5mb

4.3mb

139.6mb

2.4mb

8.3mb

37.6mb

7.6mb

28mb

52.7mb

83mb

653.4mb

zhisheng_docker

zhisheng_docker_container_blkio

zhisheng_proctstat_lookup

zhisheng_mem

zhisheng_docker_container_mem

zhisheng_netstat

zhisheng_docker_container_health

zhisheng_ntpq

zhisheng_system

zhisheng_metaserver_container

zhisheng_swap

zhisheng_diskio

zhisheng_disk

zhisheng_status_page

zhisheng_cpu

zhisheng_docker_container_status

zhisheng_docker_container_net

zhisheng_docker_container_cpu

uid

0Lmwv1MnT6KFWvLNyG3WSQ

Isxum5ERRb6F7_bpApqvHQ

06R1BamuQsq021zj6mFTra

Is_Ck5H1Qr2yrSpDRyDhrQ

c4NssURVTC64vDfql5iL0A

XjJ4J-gRTTaU1cs5Fv2jzA

e2-R3ADXSNaLUEI0ADcYQA

Xj_sU7knQDyuKjt1SM5aMw

NkEQFyejTxeelaRu_CSMJA

9meE1wN4QH-TAVX_zANmoQ

z4NXsJoeRuahFLizclm7Xg

W8C5xPjJTQqxyJx2F9DeQ

u6h6YPpZRYGyZJ3RmJJ0Zg

tbzCarW0TNuRoEfupi4v6g

gYcnI1SXQ2G8DhXAZCaiYg

i1nsP3gLTeHkdG4c-0Z0Q

10RR-48BT3ujaGUtILr4IQ

RzuXy4qGSXmthkLCFsCayQ

pri

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

5

rep

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

docs.count

25609

255654

12853

12852

264832

12844

245

192125

38540

265098

25642

28597

193921

28302

115528

264821

248891

2383311

docs.deleted

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

程序执行了一会，存入 ES 的数据量就很大了。

扩展配置

上面代码已经可以实现你的大部分场景了，但是如果你的业务场景需要保证数据的完整性（不能出现丢数据的情况），那么就需要添加一些重试策略，因为在我们的生产环境中，很有可能会因为某些组件不稳定性导致各种问题，所以这里我们就要在数据存入失败的时候做重试操作，这个 Flink 自带的 es sink 就支持了，常用的失败重试配置有：

- 1、bulk.flush.backoff.enable 用来表示是否开启重试机制
- 2、bulk.flush.backoff.type 重试策略，有两种：EXPONENTIAL 指数型（表示多次重试之间的时间间隔按照指数方式进行增长）、CONSTANT 常数型（表示多次重试之间的时间间隔为固定常数）
- 3、bulk.flush.backoff.delay 进行重试的时间间隔
- 4、bulk.flush.backoff.retries 失败重试的次数
- 5、bulk.flush.max.actions：批量写入时的最大写入条数
- 6、bulk.flush.max.size.mb：批量写入时的最大数据量
- 7、bulk.flush.interval.ms：批量写入的时间间隔，配置后则会按照该时间间隔严格执行，无视上面的两个批量写入配置

看下，就是如下这些配置了，如果你需要的话，可以在这个地方配置扩充。

```
* es sink
*
* @param hosts es hosts
* @param bulkFlushMaxActions bulk flush size
* @param parallelism 并行数
* @param data 数据
* @param func
* @param <T>
*/
public static <T> void addSink(List<HttpHost> hosts, int bulkFlushMaxActions, int parallelism,
    SingleOutputStreamOperator<T> data, ElasticsearchSinkFunction<T> func) {
    ElasticsearchSink.Builder<T> esSinkBuilder = new ElasticsearchSink.Builder<>(hosts, func);
    esSinkBuilder.setBulkFlushMaxActions(bulkFlushMaxActions);
    esSinkBuilder.set
        data.add(m
    }
    /**
ElasticSearchSin
        setBulkFlushBackoff(boolean enabled) void
        setBulkFlushBackoffDelay(long delayMillis) void
        setBulkFlushBackoffRetries(int maxRetries) void
        setBulkFlushBackoffType(FlushBackoffType flushBackoffType) void
        setBulkFlushInterval(long intervalMillis) void
        setBulkFlushMaxActions(int numMaxActions) void
        setBulkFlushMaxSizeMb(int maxSizeMb) void
        setFailureHandler(ActionRequestFailureHandler failureHandler) void - Auto-commit
        setRestClientFactory(RestClientFactory restClientFactory) void - Auto-commit
    e -> Sink: Unname
    e -> Sink: Unname
    e -> Sink: Unname
    Press ^, to choose the selected (or first) suggestion and insert a dot afterwards >> π
```

FailureHandler 失败处理器

写入 ES 的时候会有这些情况会导致写入 ES 失败。

- 1、ES 集群队列满了，报如下错误：

```
1 | 12:08:07.326 [I/O dispatcher 13] ERROR o.a.f.s.c.e.ElasticsearchSinkBase -  
Failed Elasticsearch item request: ElasticsearchException[Elasticsearch  
exception [type=es_rejected_execution_exception, reason=rejected execution  
of org.elasticsearch.transport.TransportService$7@566c9379 on  
EsThreadPoolExecutor[name = node-1/write, queue capacity = 200,  
org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@f00b373[Runni  
ng, pool size = 4, active threads = 4, queued tasks = 200, completed tasks  
= 6277]]]]
```

是这样的，我电脑安装的 ES 队列容量默认应该是 200，我没有修改过。我这里如果配置的 bulk flush size * 并发 Sink 数量 这个值如果大于这个 queue capacity，那么就很容易导致出现这种因为 ES 队列满了而写入失败。

当然这里你也可以通过调大点 es 的队列。参考：<https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-threadpool.html>

2、ES 集群某个节点挂了

这个就不用说了，肯定写入失败的。跟过源码可以发现 RestClient 类里的 performRequestAsync 方法一开始会随机的从集群中的某个节点进行写入数据，如果这台机器掉线，会进行重试在其他的机器上写入，那么当时写入的这台机器的请求就需要进行失败重试，否则就会把数据丢失！


```
main java org elasticsearch cluster routing allocation DiskThresholdMonitor
RestController.java PathTrie.java ActionModule.java
DiskThresholdMonitor.java DiskThresholdSettings.java

61 }
62
63 /**
64  * Warn about the given disk usage if the low or high watermark has been passed
65  */
66 @ private void warnAboutDiskIfNeeded(DiskUsage usage) {
67     // Check absolute disk values
68     if (usage.getFreeBytes() < diskThresholdSettings.getFreeBytesThresholdFloodStage().getBytes()) {
69         logger.warn( message: "flood stage disk watermark [{}] exceeded on {}, all indices on this node will be marked read-only",
70             diskThresholdSettings.getFreeBytesThresholdFloodStage(), usage);
71     } else if (usage.getFreeBytes() < diskThresholdSettings.getFreeBytesThresholdHigh().getBytes()) {
72         logger.warn( message: "high disk watermark [{}] exceeded on {}, shards will be relocated away from this node",
73             diskThresholdSettings.getFreeBytesThresholdHigh(), usage);
74     } else if (usage.getFreeBytes() < diskThresholdSettings.getFreeBytesThresholdLow().getBytes()) {
75         logger.info( message: "low disk watermark [{}] exceeded on {}, replicas will not be assigned to this node",
76             diskThresholdSettings.getFreeBytesThresholdLow(), usage);
77     }
78
79     // Check percentage disk values
80     if (usage.getFreeDiskAsPercentage() < diskThresholdSettings.getFreeDiskThresholdFloodStage()) {
81         logger.warn( message: "flood stage disk watermark [{}] exceeded on {}, all indices on this node will be marked read-only",
82             Strings.format1Decimals( value: 100.0 - diskThresholdSettings.getFreeDiskThresholdFloodStage(), suffix: "%"), usage);
83     } else if (usage.getFreeDiskAsPercentage() < diskThresholdSettings.getFreeDiskThresholdHigh()) {
84         logger.warn( message: "high disk watermark [{}] exceeded on {}, shards will be relocated away from this node",
85             Strings.format1Decimals( value: 100.0 - diskThresholdSettings.getFreeDiskThresholdHigh(), suffix: "%"), usage);
86     } else if (usage.getFreeDiskAsPercentage() < diskThresholdSettings.getFreeDiskThresholdLow()) {
87         logger.info( message: "low disk watermark [{}] exceeded on {}, replicas will not be assigned to this node",
88             Strings.format1Decimals( value: 100.0 - diskThresholdSettings.getFreeDiskThresholdLow(), suffix: "%"), usage);
89     }
90 }
91
92
93
```

```
216
217 private void setLowWatermark(String lowWatermark) {
218     // Watermark is expressed in terms of used data, but we need "free" data watermark
219     this.lowWatermarkRaw = lowWatermark;
220     this.freeDiskThresholdLow = 100.0 - thresholdPercentageFromWatermark(lowWatermark);
221     this.freeBytesThresholdLow = thresholdBytesFromWatermark(lowWatermark,
222         CLUSTER_ROUTING_ALLOCATION_LOW_DISK_WATERMARK_SETTING.getKey()); //默认 85%
223 }
224
```

如果你想继续让 ES 写入的话就需要去重新配一下 ES 让它继续写入，或者你也可以清空些不必要的
数据腾出磁盘空间来。

解决方法

```
1 | DataStream<String> input = ...;
2 |
3 | input.addSink(new ElasticsearchSink<> (
4 |     config, transportAddresses,
5 |     new ElasticsearchSinkFunction<String>() {...},
6 |     new ActionRequestFailureHandler() {
7 |         @Override
8 |         void onFailure(ActionRequest action,
9 |             Throwable failure,
10 |             int restStatusCode,
11 |             RequestIndexer indexer) throw Throwable {
12 |
13 |             if (ExceptionUtils.containsThrowable(failure,
14 |                 EsRejectedExecutionException.class)) {
15 |                 //队列满了，重新添加用于索引的 document
16 |             }
17 |         }
18 |     })
```

```

15         indexer.add(action);
16     } else if (ExceptionUtils.containsThrowable(failure,
ElasticsearchParseException.class)) {
17         // 对于有问题的 document, 删除该请求, 没有额外的错误处理逻辑
18     } else {
19         //对于抛出其他的异常错误, 直接就当成 sink 失败, 向外抛出异常, 你也可以抛出自定义的异常
20         throw failure;
21     }
22 }
23 });

```

如果仅仅只是想做失败重试, 也可以直接使用官方提供的默认的 `RetryRejectedExecutionFailureHandler`, 该处理器会对 `EsRejectedExecutionException` 导致到失败写入做重试处理。如果你没有设置失败处理器 (failure handler), 那么就会使用默认的 `NoOpFailureHandler` 来简单处理所有的异常。

总结

本文写了如何利用 Flink 的 ElasticSearch connector 将 Kafka 中的数据读取并存储到 ElasticSearch 中, 讲了如何封装自带的 Sink, 还有一些扩展配置以及 FailureHandler 情况下 (这个问题可是线上很容易遇到的) 要怎么处理。

Github 代码仓库

<https://github.com/zhisheng17/flink-learning/tree/master/flink-learning-connectors/flink-learning-connectors-es6>