toc: true title: 《从0到1学习Flink》—— Flink 写入数据到 ElasticSearch date: 2018-12-30 tags:

- Flink
- 大数据
- 流式计算
- ElasticSearch

## 前言

前面 FLink 的文章中我们已经介绍了说 Flink 已经有很多自带的 Connector。

1、《从0到1学习Flink》—— Data Source 介绍

2、《从0到1学习Flink》—— Data Sink 介绍

其中包括了 Source 和 Sink 的，后面我也讲了下如何自定义自己的 Source 和 Sink。

那么今天要做的事情是啥呢？就是介绍一下 Flink 自带的 ElasticSearch Connector，我们今天就用他来做 Sink，将 Kafka 中的数据经过 Flink 处理后然后存储到 ElasticSearch。

# 准备

安装 ElasticSearch，这里就忽略，自己找我以前的文章，建议安装 ElasticSearch 6.0 版本以上的，毕竟要跟上时代的节奏。

下面就讲解一下生产环境中如何使用 Elasticsearch Sink 以及一些注意点，及其内部实现机制。

# Elasticsearch Sink

## 添加依赖

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-
elasticsearch6_${scala.binary.version}</artifactId>
    <version>${flink.version}</version>
</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，当然越写到后面自己可能会做一些抽象，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，下面文章还会再讲些其他的配置。

ElasticSearchSinkUtil.java

```
public class ElasticSearchSinkUtil {

    /**
     * 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);

data.addSink(esSinkBuilder.build()).setParallelism(parallelism);
    }

    /**
     * 解析配置文件的 es hosts
     *
     * @param hosts
     * @return
     * @throws MalformedURLException
     */
    public static List<HttpHost> getEsAddresses(String hosts)
throws MalformedURLException {
        String[] hostList = hosts.split(",");
        List<HttpHost> addresses = new ArrayList<>();
        for (String host : hostList) {
            if (host.startsWith("http")) {
                URL url = new URL(host);
                addresses.add(new HttpHost(url.getHost(),
url.getPort()));
```

```
            } else {
                String[] parts = host.split(":", 2);
                if (parts.length > 1) {
                    addresses.add(new HttpHost(parts[0],
Integer.parseInt(parts[1])));
                } else {
                    throw new MalformedURLException("invalid
elasticsearch hosts format");
                }
            }
        }
        return addresses;
    }
}
```

## Main 启动类

Main.java

```
public class Main {

    public static void main(String[] args) throws Exception {
        //获取所有参数
        final ParameterTool parameterTool =
ExecutionEnvUtil.createParameterTool(args);
        //准备好环境
        StreamExecutionEnvironment env =
ExecutionEnvUtil.prepare(parameterTool);
        //从kafka读取数据
        DataStreamSource<Metrics> data =
KafkaConfigUtil.buildSource(env);

        //从配置文件中读取 es 的地址
        List<HttpHost> esAddresses =
ElasticSearchSinkUtil.getEsAddresses(parameterTool.get(ELASTICSEARC
H_HOSTS));
        //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
能调优参数，特别提醒
        int bulkSize =
parameterTool.getInt(ELASTICSEARCH_BULK_FLUSH_MAX_ACTIONS, 40);
        //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
能够更快的消费，防止 kafka 数据堆积
        int sinkParallelism =
parameterTool.getInt(STREAM_SINK_PARALLELISM, 5);

        //自己再自带的 es sink 上一层封装了下
        ElasticSearchSinkUtil.addSink(esAddresses, bulkSize,
sinkParallelism, data,
                (Metrics metric, RuntimeContext runtimeContext,
RequestIndexer requestIndexer) -> {
                    requestIndexer.add(Requests.indexRequest()
                            .index(ZHISHENG + "_" +
metric.getName())  //es 索引名
                            .type(ZHISHENG) //es type
                            .source(GsonUtil.toJSONBytes(metric),
XContentType.JSON));
                });
        env.execute("flink learning connectors es6");
    }
}
```

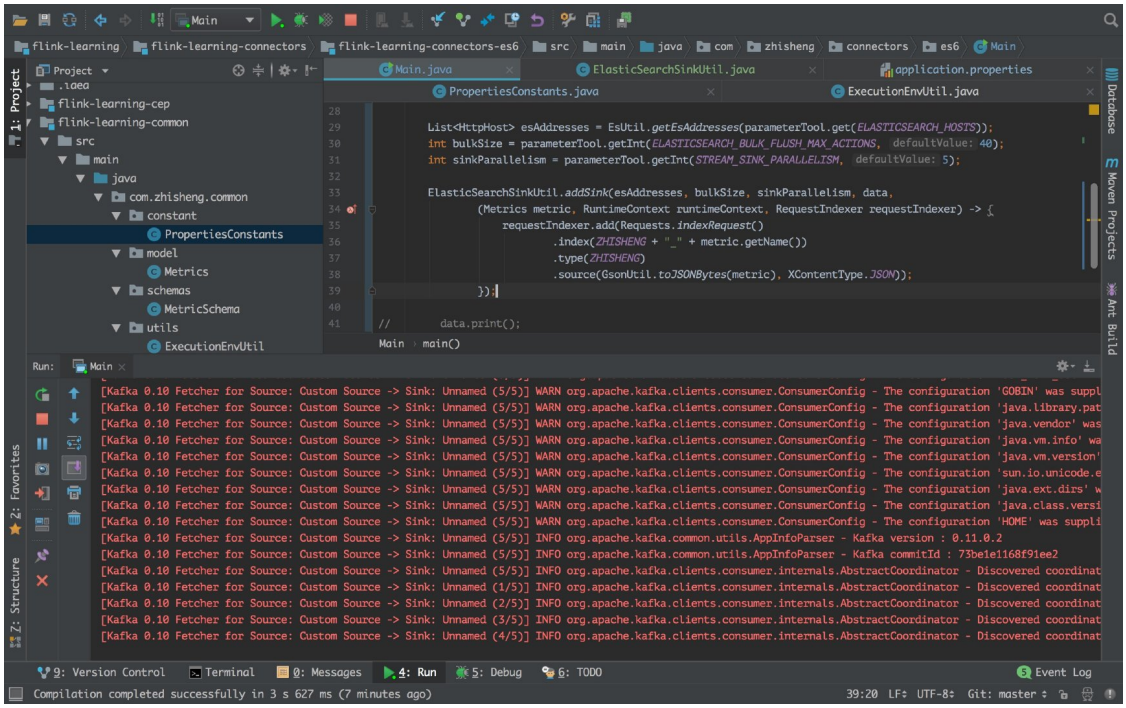
## 配置文件

配置都支持集群模式填写，注意用 , 分隔!

```
kafka.brokers=localhost:9092
kafka.group.id=zhisheng-metrics-group-test
kafka.zookeeper.connect=localhost:2181
metrics.topic=zhisheng-metrics
stream.parallelism=5
stream.checkpoint.interval=1000
stream.checkpoint.enable=false
elasticsearch.hosts=localhost:9200
elasticsearch.bulk.flush.max.actions=40
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。

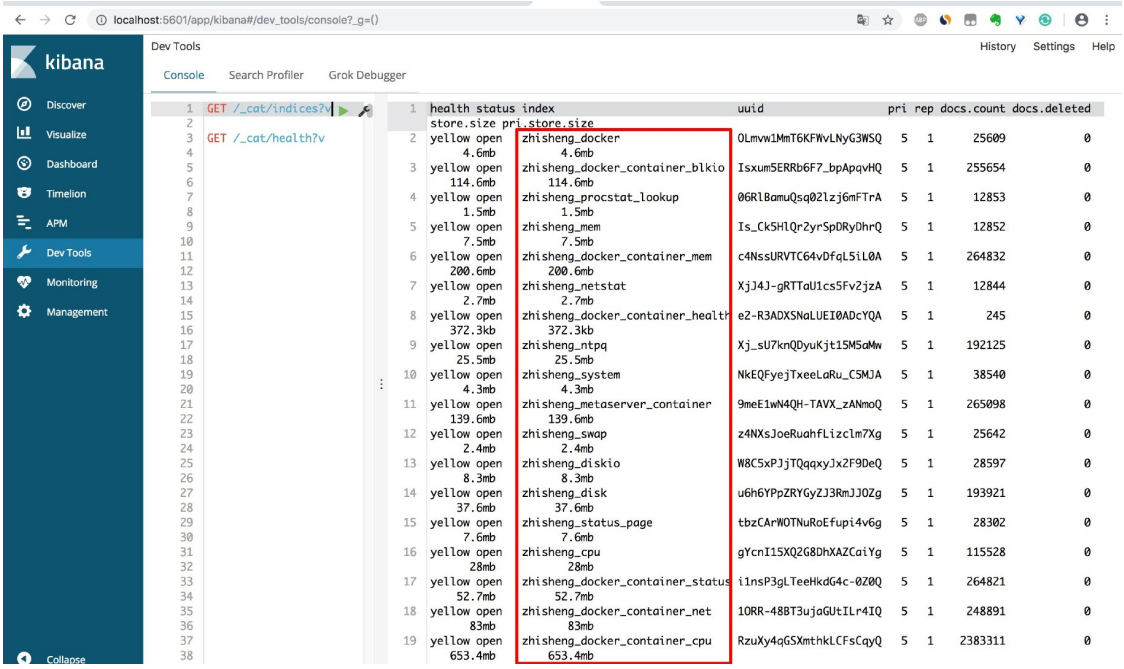
es 日志如下：

```
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_procstat][2]]
2019-01-01T21:53:31,461][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_procstat][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_procstat][3]]
2019-01-01T21:53:31,462][INFO ][o.e.i.IndexingMemoryController] [node-1] stop throttling indexing for shard [[zhisheng_procstat][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_procstat][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：



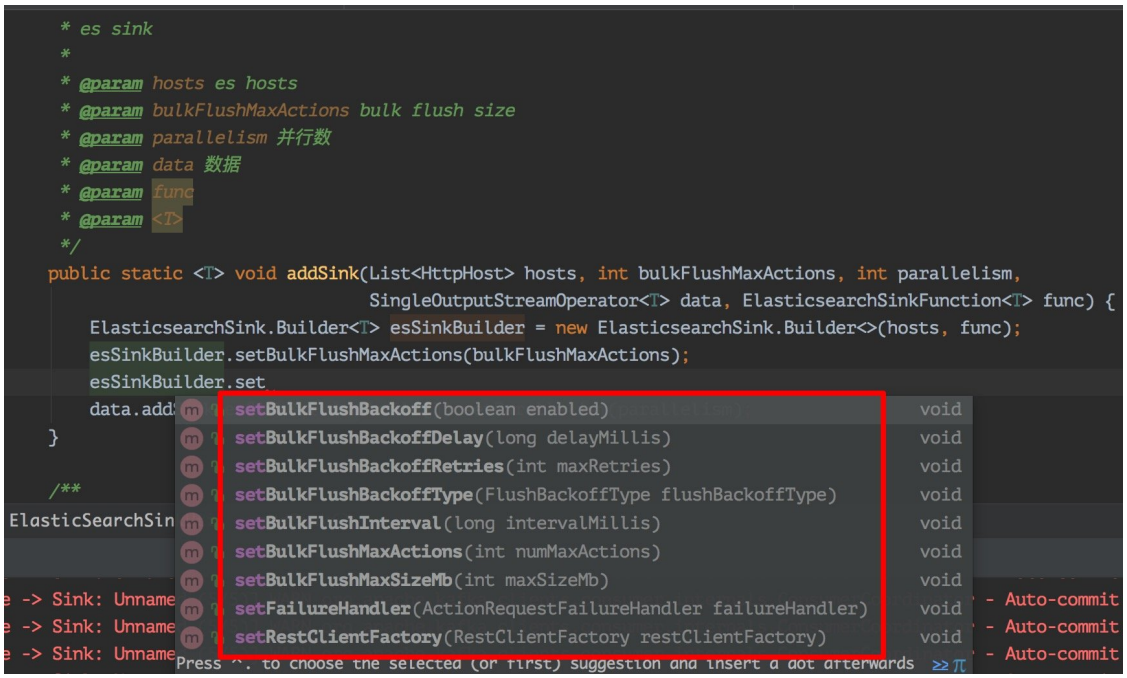

程序执行了一会，存入 es 的数据量就很大了。

## 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，这里 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
```

看下啦，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了。




## 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

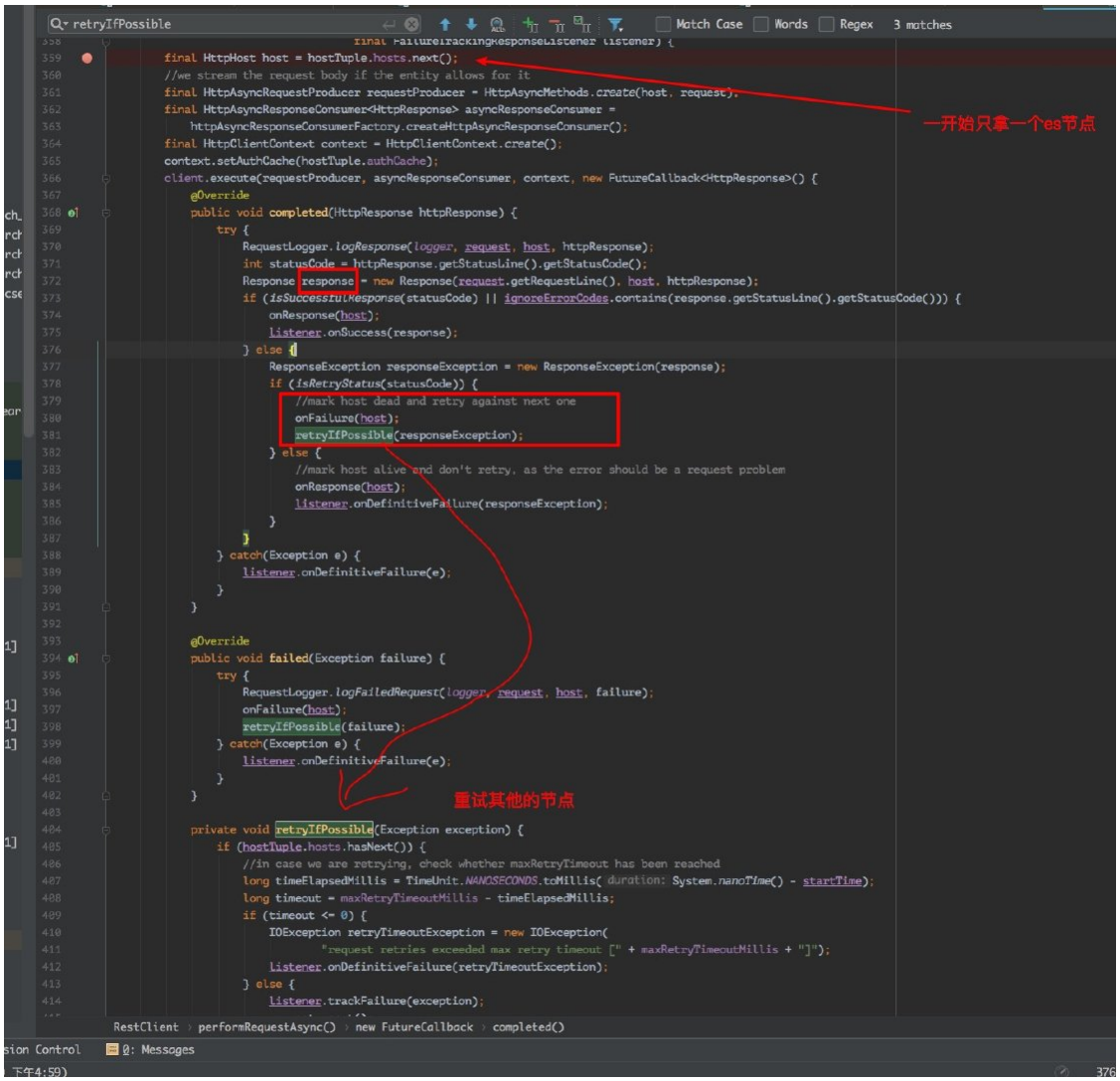
```
12:08:07.326 [I/O dispatcher 13] ERROR
o.a.f.s.c.e.ElasticsearchSinkBase - Failed Elasticsearch item
request: ElasticsearchException[Elasticsearch exception
[type=es_rejected_execution_exception, reason=rejected execution of
org.elasticsearch.transport.TransportService$7@566c9379 on
EsThreadPoolExecutor[name = node-1/write, queue capacity = 200,
org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@f00b3
73[Running, 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

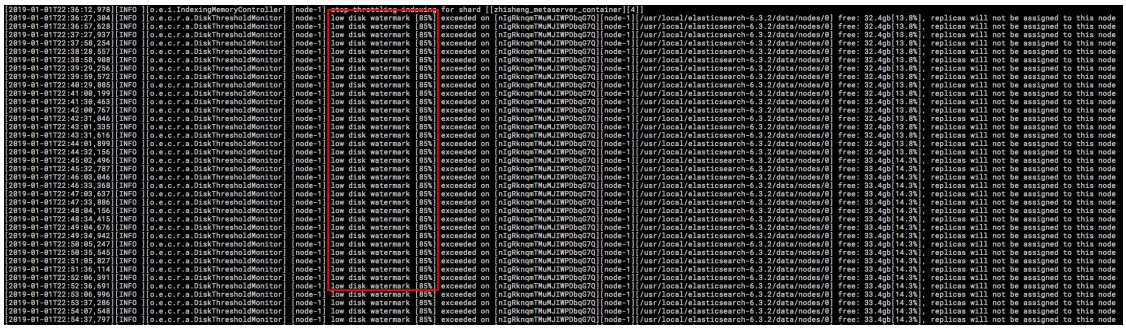
2、ES 集群某个节点挂了

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！

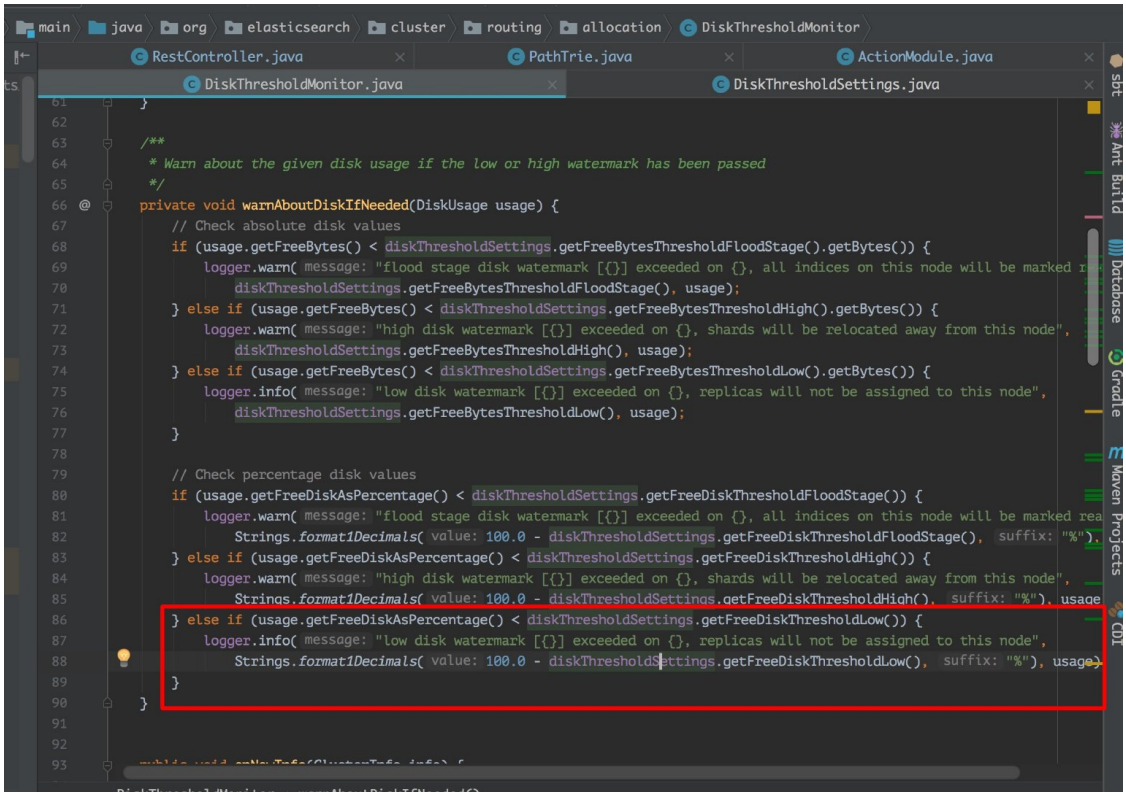



3、ES 集群某个节点的磁盘满了

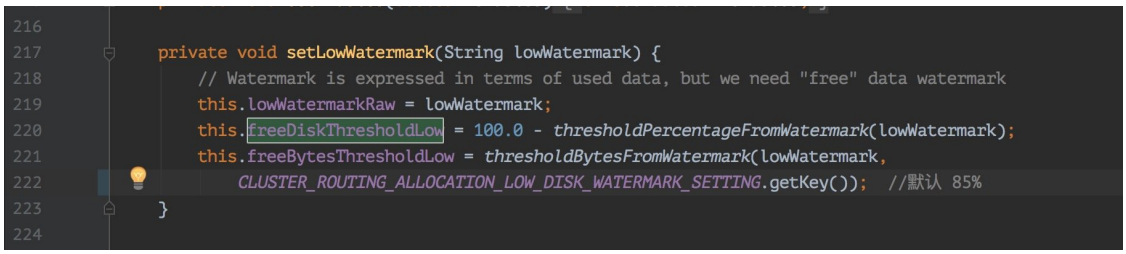
这里说的磁盘满了，并不是磁盘真的就没有一点剩余空间的，是 es 会在写入的时候检查磁盘的使用情况，在 85% 的时候会打印日志警告。



这里我看了下源码如下图：



```
    }

    /**
     * Warn about the given disk usage if the low or high watermark has been passed
     */
    private void warnAboutDiskIfNeeded(DiskUsage usage) {
        // Check absolute disk values
        if (usage.getFreeBytes() < diskThresholdSettings.getFreeBytesThresholdFloodStage().getBytes()) {
            logger.warn( message: "flood stage disk watermark [{}] exceeded on {}, all indices on this node will be marked r
                diskThresholdSettings.getFreeBytesThresholdFloodStage(), usage);
        } else if (usage.getFreeBytes() < diskThresholdSettings.getFreeBytesThresholdHigh().getBytes()) {
            logger.warn( message: "high disk watermark [{}] exceeded on {}, shards will be relocated away from this node",
                diskThresholdSettings.getFreeBytesThresholdHigh(), usage);
        } else if (usage.getFreeBytes() < diskThresholdSettings.getFreeBytesThresholdLow().getBytes()) {
            logger.info( message: "low disk watermark [{}] exceeded on {}, replicas will not be assigned to this node",
                diskThresholdSettings.getFreeBytesThresholdLow(), usage);
        }

        // Check percentage disk values
        if (usage.getFreeDiskAsPercentage() < diskThresholdSettings.getFreeDiskThresholdFloodStage()) {
            logger.warn( message: "flood stage disk watermark [{}] exceeded on {}, all indices on this node will be marked rea
                Strings.format1Decimals( value: 100.0 - diskThresholdSettings.getFreeDiskThresholdFloodStage(), suffix: "%"),
        } else if (usage.getFreeDiskAsPercentage() < diskThresholdSettings.getFreeDiskThresholdHigh()) {
            logger.warn( message: "high disk watermark [{}] exceeded on {}, shards will be relocated away from this node",
                Strings.format1Decimals( value: 100.0 - diskThresholdSettings.getFreeDiskThresholdHigh(), suffix: "%"), usage
        } else if (usage.getFreeDiskAsPercentage() < diskThresholdSettings.getFreeDiskThresholdLow()) {
            logger.info( message: "low disk watermark [{}] exceeded on {}, replicas will not be assigned to this node",
                Strings.format1Decimals( value: 100.0 - diskThresholdSettings.getFreeDiskThresholdLow(), suffix: "%"), usage)
        }
    }
```



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

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

```
DataStream<String> input = ...;

input.addSink(new ElasticsearchSink<>(
    config, transportAddresses,
    new ElasticsearchSinkFunction<String>() {...},
    new ActionRequestFailureHandler() {
        @Override
        void onFailure(ActionRequest action,
                Throwable failure,
                int restStatusCode,
                RequestIndexer indexer) throw Throwable {

            if (ExceptionUtils.containsThrowable(failure,
EsRejectedExecutionException.class)) {
                // full queue; re-add document for indexing
                indexer.add(action);
            } else if (ExceptionUtils.containsThrowable(failure,
ElasticsearchParseException.class)) {
                // malformed document; simply drop request without
failing sink
            } else {
                // for all other failures, fail the sink
                // here the failure is simply rethrown, but users
can also choose to throw custom exceptions
                throw failure;
            }
        }
}));
```

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 Flink connector es，将 Kafka 中的数据读取并存储到 ElasticSearch 中，文中讲了如何封装自带的 sink，然后一些扩展配置以及 FailureHandler 情况下要怎么处理。（这个问题可是线上很容易遇到的）

## 关注我

转载请务必注明原创地址为：http://www.54tianzhisheng.cn/2018/12/30/Flink-ElasticSearch-Sink/

另外我自己整理了些 Flink 的学习资料，目前已经全部放到微信公众号了。你可以加我的微信：**zhisheng_tian**，然后回复关键字：**Flink** 即可无条件获取到。

## 相关文章

1、《从0到1学习Flink》—— Apache Flink 介绍

2、《从0到1学习Flink》—— Mac 上搭建 Flink 1.6.0 环境并构建运行简单程序入门

3、《从0到1学习Flink》—— Flink 配置文件详解

4、《从0到1学习Flink》—— Data Source 介绍

5、《从0到1学习Flink》—— 如何自定义 Data Source ？

6、《从0到1学习Flink》—— Data Sink 介绍

7、《从0到1学习Flink》—— 如何自定义 Data Sink ？

8、《从0到1学习Flink》—— Flink Data transformation(转换)

9、《从0到1学习Flink》—— 介绍Flink中的Stream Windows

10、《从0到1学习Flink》—— Flink 中的几种 Time 详解

11、《从0到1学习Flink》—— Flink 写入数据到 ElasticSearch