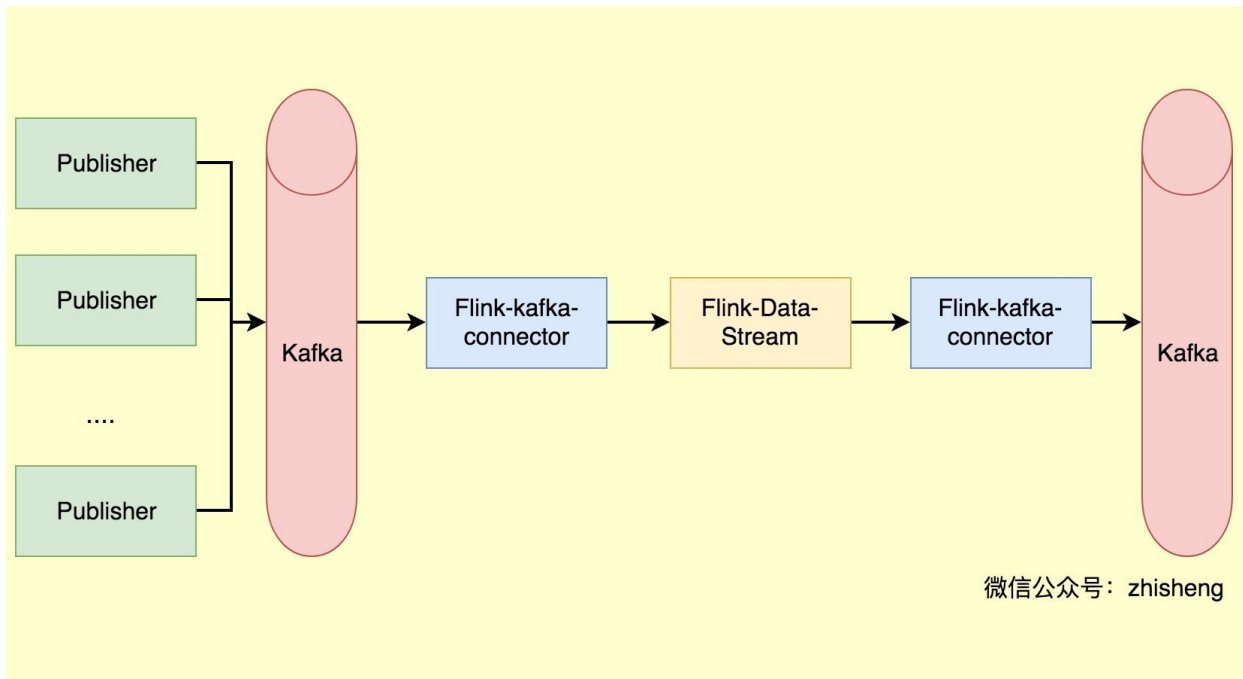


## 前言

在前面文章 [Flink 常用的 Source 和 Sink Connectors 介绍](#) 中我讲解了 Flink 中的 Data Source 和 Data Sink，然后介绍了 Flink 中自带的一些 Source 和 Sink 的 Connector。本篇文章我将讲解一下几乎用的最多的 Connector —— Kafka，带大家利用 Kafka Connector 读取 Kafka 数据，做一些计算操作后然后又通过 Kafka Connector 写入到 kafka 消息队列去。



## Kafka Source Connector 上手

### 准备工作

我们先来看下 Flink 从 Kafka topic 中获取数据的 demo，首先你需要安装好了 Flink 和 Kafka。

如果你已经安装好了 Flink 和 Kafka，那么接下来运行启动 Flink、Zookeeper、Kafka 就行了。

```
flink-console.sh jobmanager.sh mesos-taskmanager.sh sql-client.sh start-scala-shell.sh
r-quorum.sh zookeeper.sh
zhisheng@zhisheng > /usr/local/Cellar/apache-flink/1.6.0/libexec/bin > ./start-cluster.sh
Starting cluster.
Starting standalone-session daemon on host zhisheng.
Starting taskexecutor daemon on host zhisheng.
zhisheng@zhisheng > /usr/local/Cellar/apache-flink/1.6.0/libexec/bin > ls
config.sh jobmanager.sh mesos-appmaster-job.sh pyflink.sh start-zookeeper-quorum.sh zookeeper.sh
flink mesos-appmaster.sh sql-client.sh standalone-job.sh stop-cluster.sh
flink-console.sh mesos-taskmanager.sh start-cluster.sh stop-zookeeper-quorum.sh
flink-daemon.sh pyflink-stream.sh start-scala-shell.sh taskmanager.sh
historyserver.sh yarn-session.sh
zhisheng@zhisheng > /usr/local/Cellar/apache-flink/1.6.0/libexec/bin >
zhisheng@zhisheng > /usr/local/Cellar/apache-flink/1.6.0/libexec/bin >
```

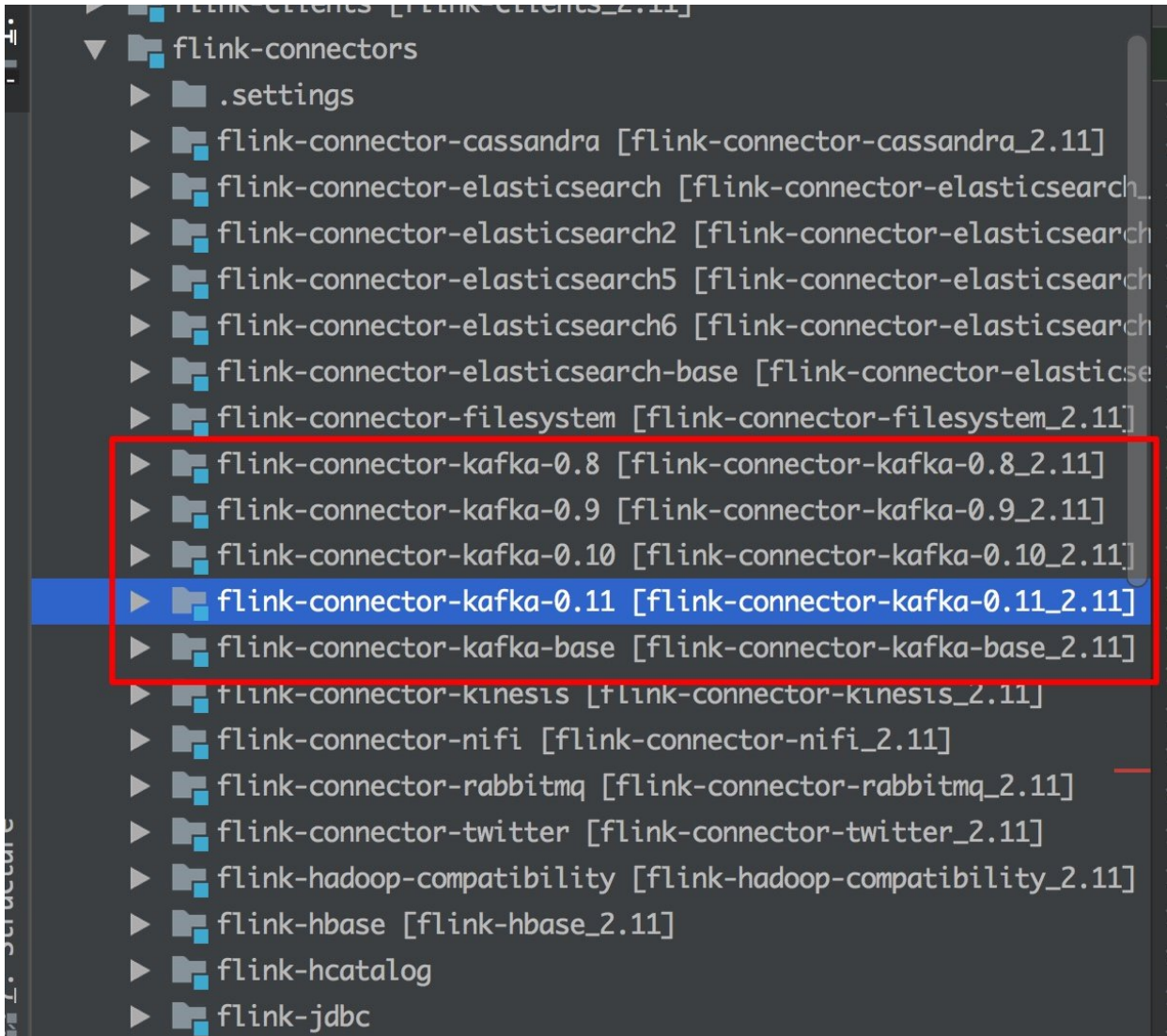
```

given to allow fail-over.
* zhisheng@zhisheng /usr/local/kafka_2.11-1.1.0 ls
LICENSE NOTICE bin config libs logs run.sh site-docs
zhisheng@zhisheng /usr/local/kafka_2.11-1.1.0
zhisheng@zhisheng /usr/local/kafka_2.11-1.1.0 cat run.sh
#!/bin/bash
bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
# bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic metrics
zhisheng@zhisheng /usr/local/kafka_2.11-1.1.0

```

好了，都启动了！

## Maven 依赖



Flink 里面支持 Kafka 0.8、0.9、0.10、0.11 以及更高版本，我们采用哪个版本的 Maven 依赖需要根据我们自己安装的 Kafka 版本。因为之前我们安装的 Kafka 是 0.11 版本，所以这里我们选择的 Kafka Connector 为 `flink-connector-kafka-0.11_2.11`。

```

1 <dependency>
2   <groupId>org.apache.flink</groupId>
3   <artifactId>flink-connector-kafka-0.11_2.11</artifactId>
4   <version>${flink.version}</version>
5 </dependency>

```

另外你还要添加的依赖有：

```

1  <!--flink java-->
2  <dependency>
3      <groupId>org.apache.flink</groupId>
4      <artifactId>flink-java</artifactId>
5      <version>${flink.version}</version>
6      <scope>provided</scope>
7  </dependency>
8  <dependency>
9      <groupId>org.apache.flink</groupId>
10     <artifactId>flink-streaming-java_${scala.binary.version}</artifactId>
11     <version>${flink.version}</version>
12     <scope>provided</scope>
13 </dependency>
14
15 <!--log-->
16 <dependency>
17     <groupId>org.slf4j</groupId>
18     <artifactId>slf4j-log4j12</artifactId>
19     <version>1.7.7</version>
20     <scope>runtime</scope>
21 </dependency>
22 <dependency>
23     <groupId>log4j</groupId>
24     <artifactId>log4j</artifactId>
25     <version>1.2.17</version>
26     <scope>runtime</scope>
27 </dependency>
28
29 <!--alibaba fastjson-->
30 <dependency>
31     <groupId>com.alibaba</groupId>
32     <artifactId>fastjson</artifactId>
33     <version>1.2.51</version>
34 </dependency>

```

## 测试发送数据到 kafka topic

实体类, Metric.java

```

1  package com.zhisheng.flink.model;
2
3  import java.util.Map;
4
5  /**
6   * blog: http://www.54tianzhisheng.cn/
7   */
8  @Data
9  @AllArgsConstructor
10 @NoArgsConstructor
11 public class Metric {
12     public String name;
13     public long timestamp;
14     public Map<String, Object> fields;
15     public Map<String, String> tags;
16 }

```

## 往 kafka 中写数据工具类：KafkaUtils.java

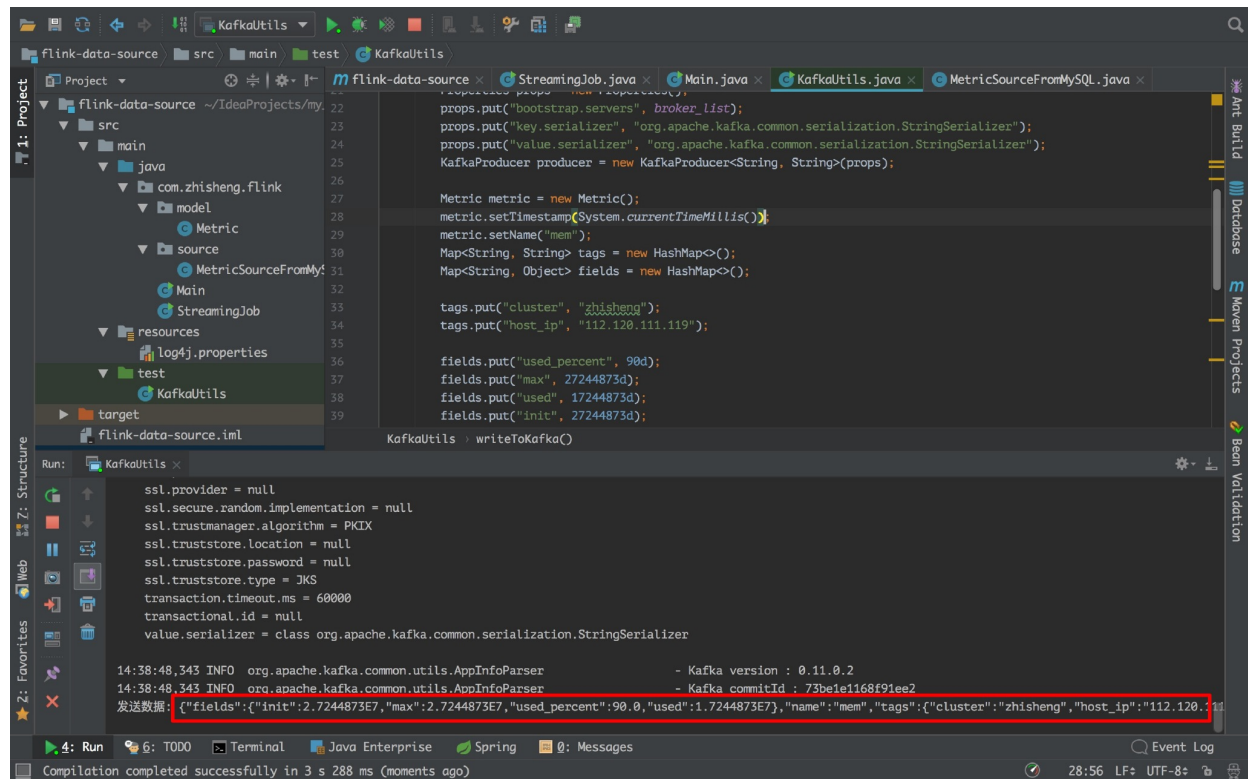
```
1  import com.alibaba.fastjson.JSON;
2  import com.zhisheng.flink.model.Metric;
3  import org.apache.kafka.clients.producer.KafkaProducer;
4  import org.apache.kafka.clients.producer.ProducerRecord;
5
6  import java.util.HashMap;
7  import java.util.Map;
8  import java.util.Properties;
9
10 /**
11  * 往kafka中写数据
12  * 可以使用这个main函数进行测试一下
13  * blog: http://www.54tianzhisheng.cn/
14  */
15 public class KafkaUtils {
16     public static final String broker_list = "localhost:9092";
17     public static final String topic = "metric"; // kafka topic, Flink 程
序中需要和这个统一
18
19     public static void writeToKafka() throws InterruptedException {
20         Properties props = new Properties();
21         props.put("bootstrap.servers", broker_list);
22         props.put("key.serializer",
23 "org.apache.kafka.common.serialization.StringSerializer"); //key 序列化
24         props.put("value.serializer",
25 "org.apache.kafka.common.serialization.StringSerializer"); //value 序列化
26         KafkaProducer producer = new KafkaProducer<String, String>(props);
27
28         Metric metric = new Metric();
29         metric.setTimestamp(System.currentTimeMillis());
30         metric.setName("mem");
31         Map<String, String> tags = new HashMap<>();
32         Map<String, Object> fields = new HashMap<>();
33
34         tags.put("cluster", "zhisheng");
35         tags.put("host_ip", "101.147.022.106");
36
37         fields.put("used_percent", 90d);
38         fields.put("max", 27244873d);
39         fields.put("used", 17244873d);
40         fields.put("init", 27244873d);
41
42         metric.setTags(tags);
43         metric.setFields(fields);
44
45         ProducerRecord record = new ProducerRecord<String, String>(topic,
46 null, null, JSON.toJSONString(metric));
47         producer.send(record);
48         System.out.println("发送数据: " + JSON.toJSONString(metric));
49
50         producer.flush();
51     }
52
53     public static void main(String[] args) throws InterruptedException {
54         while (true) {
55             Thread.sleep(300);
56         }
57     }
58 }
```

```

53         writeToKafka();
54     }
55 }
56 }

```

运行:



如果出现如上图标记的，即代表能够不断往 kafka 发送数据的。

## Flink 程序

### Main.java

```

1 package com.zhisheng.flink;
2
3 import org.apache.flink.api.common.serialization.SimpleStringSchema;
4 import org.apache.flink.streaming.api.datastream.DataStreamSource;
5 import
6 org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer011;
8
9 import java.util.Properties;
10
11 /**
12  * blog: http://www.54tianzhisheng.cn/
13  */
14 public class Main {
15     public static void main(String[] args) throws Exception {
16         final StreamExecutionEnvironment env =
17         StreamExecutionEnvironment.getExecutionEnvironment();
18     }
19 }

```

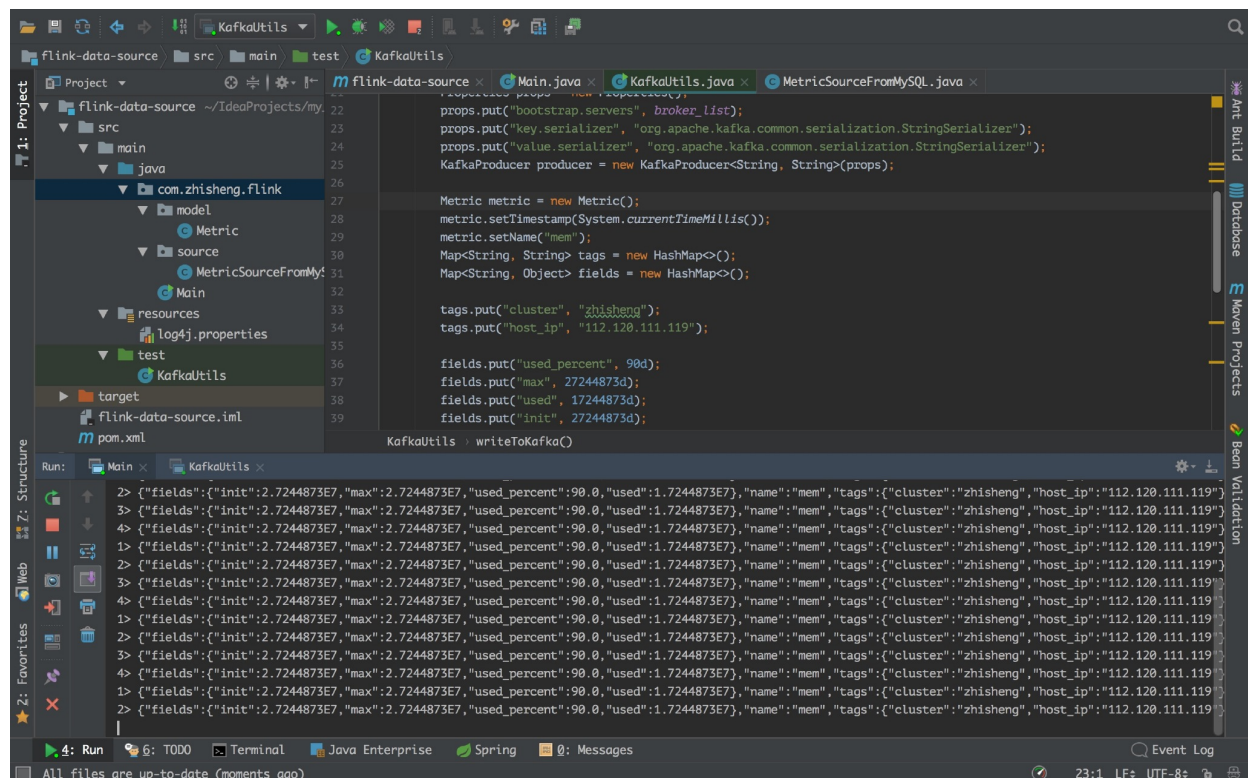


```

17     Properties props = new Properties();
18     props.put("bootstrap.servers", "localhost:9092");
19     props.put("zookeeper.connect", "localhost:2181");
20     props.put("group.id", "metric-group");
21     props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer"); //key 反序列化
22     props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
23     props.put("auto.offset.reset", "latest"); //value 反序列化
24
25     DataStreamSource<String> dataStreamSource = env.addSource(new
FlinkKafkaConsumer011<> (
26         "metric", //kafka topic
27         new SimpleStringSchema(), // String 序列化
28         props)).setParallelism(1);
29
30     dataStreamSource.print(); //把从 kafka 读取到的数据打印在控制台
31
32     env.execute("Flink add data source");
33 }
34 }

```

运行起来：



看到没程序，Flink 程序控制台能够源源不断地打印数据呢。

## 代码分析

使用 FlinkKafkaConsumer011 时传入了三个参数。

- Kafka topic: 这个代表了 Flink 要消费的是 Kafka 哪个 Topic, 如果你要同时消费多个 Topic 的话, 那么你可以传入一个 Topic List 进去, 另外也支持正则表达式匹配 Topic
- 序列化: 上面代码我们使用的是 SimpleStringSchema
- 配置属性: 将 Kafka 等的一些配置传入

```
@PublicEvolving
public class FlinkKafkaConsumer011<T> extends FlinkKafkaConsumer010<T> {
    private static final long serialVersionUID = 2324564345203409112L;

    public FlinkKafkaConsumer011(String topic, DeserializationSchema<T> valueDeserializer, Properties props) {
        this(Collections.singletonList(topic), valueDeserializer, props);
    }

    public FlinkKafkaConsumer011(String topic, KeyedDeserializationSchema<T> deserializer, Properties props) {
        this(Collections.singletonList(topic), deserializer, props);
    }

    public FlinkKafkaConsumer011(List<String> topics, DeserializationSchema<T> deserializer, Properties props) {
        this((List)topics, (KeyedDeserializationSchema)(new KeyedDeserializationSchemaWrapper(deserializer)), props);
    }

    public FlinkKafkaConsumer011(List<String> topics, KeyedDeserializationSchema<T> deserializer, Properties props) {
        super(topics, deserializer, props);
    }

    @PublicEvolving
    public FlinkKafkaConsumer011(Pattern subscriptionPattern, DeserializationSchema<T> valueDeserializer, Properties props) {
        this((Pattern)subscriptionPattern, (KeyedDeserializationSchema)(new KeyedDeserializationSchemaWrapper(valueDeserializer)), props);
    }

    @PublicEvolving
    public FlinkKafkaConsumer011(Pattern subscriptionPattern, KeyedDeserializationSchema<T> deserializer, Properties props) {
        super(subscriptionPattern, deserializer, props);
    }
}
```

多个 Topic

正则表达式匹配 Topic

zhisheng

这里我们接着演示把其他 Kafka 集群中 topic 数据原样写入到自己本地起的 Kafka 中去。

## 配置文件

```
1 kafka.brokers=xxx:9092,xxx:9092,xxx:9092
2 kafka.group.id=metrics-group-test
3 kafka.zookeeper.connect=xxx:2181
4 metrics.topic=xxx
5 stream.parallelism=5
6 kafka.sink.brokers=localhost:9092
7 kafka.sink.topic=metric-test
8 stream.checkpoint.interval=1000
9 stream.checkpoint.enable=false
10 stream.sink.parallelism=5
```

目前我们先看下本地 Kafka 是否有这个 metric-test topic 呢? 需要执行下这个命令:

```
1 bin/kafka-topics.sh --list --zookeeper localhost:2181
```

```

--exclude-dir=.svn kafka
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 kill -9 73584
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 kill -9 73585
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 ps -ef | grep kafka
501 54274 53922 0 2:49下午 ttys008 0:00.00 grep --color=auto --exclude-dir=.bzip --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.
--exclude-dir=.svn kafka
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 ls
LICENSE NOTICE bin config libs logs run.sh site-docs
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181

```

可以看到本地的 Kafka 是没有任何 topic 的，如果等下我们的程序运行起来后，再次执行这个命令出现 metric-test topic，那么证明我的程序确实起作用了，已经将其他集群的 Kafka 数据写入到本地 Kafka 了。

## 程序代码

Main.java

```

1 public class Main {
2     public static void main(String[] args) throws Exception{
3         final ParameterTool parameterTool =
4         ExecutionEnvUtil.createParameterTool(args);
5         StreamExecutionEnvironment env =
6         ExecutionEnvUtil.prepare(parameterTool);
7         DataSource<Metrics> data = KafkaConfigUtil.buildSource(env);
8
9         data.addSink(new FlinkKafkaProducer011<Metrics>(
10             parameterTool.get("kafka.sink.brokers"),
11             parameterTool.get("kafka.sink.topic"),
12             new MetricSchema()
13         ).name("flink-connectors-kafka"));
14
15         .setParallelism(parameterTool.getInt("stream.sink.parallelism"));
16
17         env.execute("flink learning connectors kafka");
18     }
19 }

```

## 运行结果

启动程序，查看运行结果，不段执行上面命令，查看是否有新的 topic 出来：



```

zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0

```

执行命令可以查看该 topic 的信息：

```

1 | bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic metric-
  | test

```

```

zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --list --zookeeper localhost:2181
metric-test
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic metric-test
Topic:metric-test PartitionCount:1 ReplicationFactor:1 Configs:
  Topic: metric-test Partition: 0 Leader: 0 Replicas: 0 Isr: 0
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic metric-test
Topic:metric-test PartitionCount:1 ReplicationFactor:1 Configs:
  Topic: metric-test Partition: 0 Leader: 0 Replicas: 0 Isr: 0
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0 bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic metric-test
Topic:metric-test PartitionCount:1 ReplicationFactor:1 Configs:
  Topic: metric-test Partition: 0 Leader: 0 Replicas: 0 Isr: 0
zhisheng@zhisheng > /usr/local/kafka_2.11-1.1.0

```

## Flink Kafka Producer 分析

上面代码我们使用 Flink Kafka Producer 只传了三个参数：brokerList、topicId、serializationSchema（序列化）

```
flink-connector-kafka-0.11-2.11-1.6.2.jar  org  apache  flink  streaming  connectors  kafka  FlinkKafkaProducer011
Main.java x FlinkKafkaProducer011.class x
Decompiled .class file, bytecode version: 52.0 (Java 8) Download Sources Choose Sources...
93 private boolean logFailuresOnly;
94 private FlinkKafkaProducer011.Semantic semantic;
95 @Nullable
96 private transient Callback callback;
97 @Nullable
98 private transient volatile Exception asyncException;
99 private final AtomicLong pendingRecords;
100 private final Map<String, KafkaMetricMutttableWrapper> previouslyCreatedMetrics;
101
102 @ public FlinkKafkaProducer011(String brokerList, String topicId, SerializationSchema<IN> serializationSchema) {...}
105
106 @ public FlinkKafkaProducer011(String topicId, SerializationSchema<IN> serializationSchema, Properties producerConfig) {...}
109
110 @ public FlinkKafkaProducer011(String topicId, SerializationSchema<IN> serializationSchema, Properties producerConfig, Optional<FlinkKafkaPartitioner<IN>> customPa
113
114 @ public FlinkKafkaProducer011(String brokerList, String topicId, KeyedSerializationSchema<IN> serializationSchema) {...}
117
118 @ public FlinkKafkaProducer011(String topicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig) {...}
121
122 @ public FlinkKafkaProducer011(String topicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig, FlinkKafkaProducer011.Semantic semantic
125
126 @ public FlinkKafkaProducer011(String defaultTopicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig, Optional<FlinkKafkaPartitioner<
129
130 @ public FlinkKafkaProducer011(String defaultTopicId, KeyedSerializationSchema<IN> serializationSchema, Properties producerConfig, Optional<FlinkKafkaPartitioner<
187
188 public void setWriterTimestampToKafka(boolean writeTimestampToKafka) {
189     this.writeTimestampToKafka = writeTimestampToKafka;
190 }
191
192 public void setLogFailuresOnly(boolean logFailuresOnly) { this.logFailuresOnly = logFailuresOnly; }
```

其实也可以传入多个参数进去，现在有的参数用的是默认参数，因为这个内容比较多，后面可以抽出一篇文章单独来讲。

## 总结

本篇文章写了 Flink 读取其他 Kafka 集群的数据，然后写入到本地的 Kafka 上。我在 Flink 这层没做什么数据转换，只是原样的将数据转发了下。如果你们有其他的需求，是可以在 Flink 这层将数据进行各种转换操作的，比如这篇文章中的一些转换：[1Flink 数据转换必须熟悉的算子 \(Operator\)](#)，然后将转换后的数据发到 Kafka 上去。

## Github 代码仓库

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