Flink Connector 分享

董亭亭 · 快手/ 高级开发工程师

Apache Flink Community China

CONTENT

目录 >>

01

Flink Streaming Connectors

Apache Flink

Streaming Connectors

预定义的Source和Sink

Bundled Connectors

Apache Bahir中的连接器

Async I/O

预定义的Source和Sink

基于文件的Source

readTextFile(path)

readFile(fileInputFormat, path)

基于文件的Sink

writeAsText

writeAsCsv

基于Socket

socketTextStream

基于Socket的Sink

writeToSocket

基于Collections、Iterators

fromCollection、fromElements

标准输出、标准错误

Print、printToError

Bundled Connectors

- Apache Kafka (source/sink)
- Apache Cassandra (sink)
- Amazon Kinesis Streams (source/sink)
- Elasticsearch (sink)
- Hadoop FileSystem (sink)
- RabbitMQ (source/sink)
- Apache NiFi (source/sink)
- Twitter Streaming API (source)

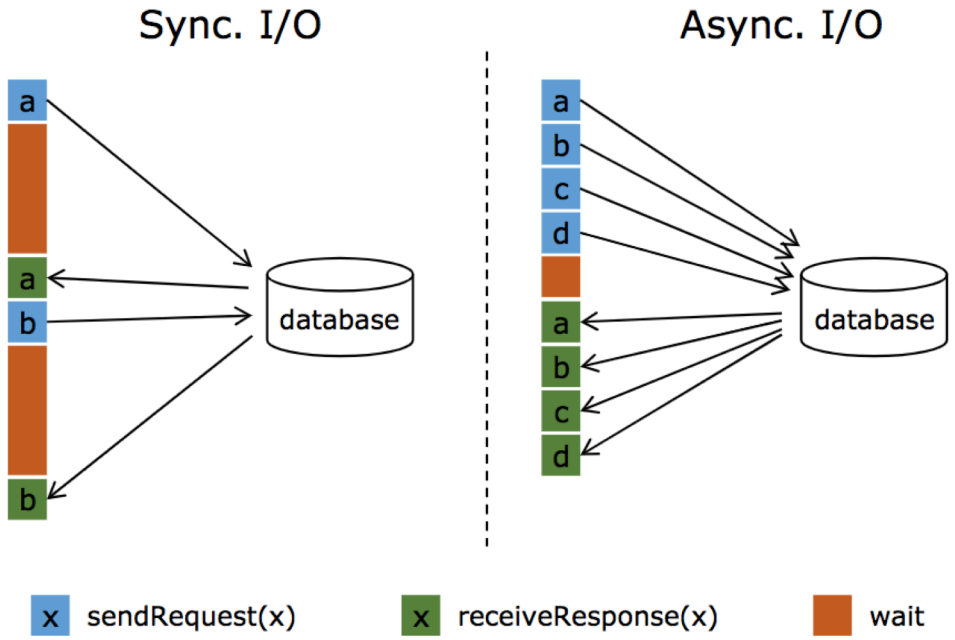
➢ **注意：**以上流connector是Flink 项目的一部分，但是不包括在二进制发布包中！

Apache Bahir中的连接器

- Apache ActiveMQ(source/sink)
- Apache Flume(sink)
- Redis(sink)
- Akka(sink)
- Netty(Source)

Async I/O

- 使用connector并不是数据输入输出Flink的唯一方式。
- 在Map、FlatMap中使用Async I/O方式读取外部数据库等。



https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/stream/asyncio.html

02

Flink Kafka Connector

Apache Flink

Flink Kafka Connector

Flink Kafka Consumer

- 反序列化数据
- 消费起始位置设置
- Topic和Partition动态发现
- Commit Offset方式
- Timestamp Extraction/Watermark 生成

Flink Kafka Producer

- Producer分区
- 容错

Flink Kafka Example

```
public class FlinkKafkaExample {
  public static void main(String[] args) throws Exception{
    {
      final ParameterTool params = ParameterTool.fromArgs(args);

      // set up the execution environment
      final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
      env.getConfig().setGlobalJobParameters(params);
      env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
      env.enableCheckpointing( interval: 60*1000, CheckpointingMode.EXACTLY_ONCE);

      String sourceTopic = "topic1";
      String bootStrapServers = "localhost:9092";                                    source
      Properties properties = getConsumerProperties(bootStrapServers, groupId: "test-dtt-gi");
      FlinkKafkaConsumer010<String> consumer = new FlinkKafkaConsumer010<~>(sourceTopic, new SimpleStringSchema(), properties);
      //consumer.setStartFromEarliest();
      //consumer.setStartFromLatest();
      //consumer.setStartFromTimestamp(1561281792000L);
      //consumer.setStartFromGroupOffsets() //default

      String sinkTopic = "topic2";                                                   sink
      Properties prop = getProducerProperties(bootStrapServers);
      FlinkKafkaProducer010<String> producer = new FlinkKafkaProducer010<~>(sinkTopic, new SimpleStringSchema(), prop);

      env.addSource(consumer) DataStreamSource<String>
        .map(new MapFunction<String, Tuple2<Long, String>>() {
          @Override
          public Tuple2<Long, String> map(String s) throws Exception {...}
        }) SingleOutputStreamOperator<Tuple2<Long, String>>
        .filter(k -> k!=null) SingleOutputStreamOperator<Tuple2<Long, String>>
        .assignTimestampsAndWatermarks(new BoundedOutOfOrdernessTimestampExtractor<Tuple2<Long, String>>(Time.seconds(5)) {
          @Override
          public long extractTimestamp(Tuple2<Long, String> element) { return element.f0; }
        }) SingleOutputStreamOperator<Tuple2<Long, String>>
        .map(k -> k.toString()) SingleOutputStreamOperator<String>
        .addSink(producer);

      env.execute( jobName: "FlinkKafkaExample");
    }
  }
```

Flink Kafka Consumer–反序列化数据

- 将kafka中二进制数据转化为具体的java、scala 对象
- DeserializationSchema ， T deserialize(byte[] message)
- KeyedDeserializationSchema ， T deserialize(byte[] messageKey, byte[] message, String topic, int partition, long offset)：对于访问kafka key/value

常用

- SimpleStringSchema： 按字符串方式进行序列化、反序列化
- TypeInformationSerializationSchema：基于Flink的TypeInformation来创建schema
- JsonDeserializationSchema：使用jackson反序列化json格式消息，并返回ObjectNode，可以使用.get(“property”)方法来访问字段。

Flink Kafka Consumer–消费起始位置

- **setStartFromGroupOffsets(默认)**
  - 从Kafka记录的group.id 的位置开始读取，如果没有根据auto.offset.reset设置
- **setStartFromEarliest**
  - 从kafka最早的位置读取
- **setStartFromLatest**
  - 从kafka最新数据开始读取
- **setStartFromTimestamp(long)**
  - 从时间戳大于或等于指定时间戳的位置开始读取
- **setStartFromSpecificOffsets**
  - 从指定的分区的offset位置开始读取，如指定的offsets中不存某个分区，该分区从group offset位置开始读取。

Flink Kafka Consumer–消费起始位置

```
public static void main(String[] args) throws Exception{
  {
    final ParameterTool params = ParameterTool.fromArgs(args);

    // set up the execution environment
    final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
    env.getConfig().setGlobalJobParameters(params);
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
    env.enableCheckpointing( interval: 60*1000, CheckpointingMode.EXACTLY_ONCE);

    String sourceTopic = "topic1";
    String bootStrapServers = "localhost:9092";
    Properties properties = getConsumerProperties(bootStrapServers,  groupId: "test-gid");
    FlinkKafkaConsumer010<String> consumer = new FlinkKafkaConsumer010<~>(sourceTopic, new SimpleStringSchema(), properties);
    consumer.setStartFromEarliest();
    consumer.setStartFromLatest();
    consumer.setStartFromTimestamp(1561281792000L);
    consumer.setStartFromGroupOffsets();

    Map<KafkaTopicPartition, Long> specificStartOffsets = new HashMap<>();
    specificStartOffsets.put(new KafkaTopicPartition(sourceTopic, partition: 0), 0L);
    specificStartOffsets.put(new KafkaTopicPartition(sourceTopic, partition: 1), 1L);
    consumer.setStartFromSpecificOffsets(specificStartOffsets);
```

- **注意：**作业故障从checkpoint自动恢复，以及手动做savepoint时，消费的位置从保存状态中恢复，与该配置无关！！！

Flink Kafka Consumer–topic partition自动发现

原理：内部单独的线程获取kafka meta信息进行更新

flink.partition–discovery.interval–millis:发现时间间隔。默认false，设置非负值开启。

分区发现

- 消费的Source kafka topic进行了partition 扩容
- 新发现的分区，从earliest位置开始读取

Topic发现

- 支持正则表达式描述topic名字

```
String sourceTopic = "topic1";
String bootStrapServers = "localhost:9092";
Properties properties = getConsumerProperties(bootStrapServers, groupId: "test-gid");
//FlinkKafkaConsumer010<String> consumer = new FlinkKafkaConsumer010<String>(sourceTopic, new SimpleStringSchema(), propert
Pattern topicPattern = java.util.regex.Pattern.compile("topic[0-9]");
FlinkKafkaConsumer010<String> consumer = new FlinkKafkaConsumer010<~>(topicPattern, new SimpleStringSchema(), properties);
```

Flink Kafka Consumer–commit offset方式

Checkpoint关闭

- 依赖kafka客户端的auto commit 定期提交offset。
- 需设置enable.auto.commit， auto.commit.interval.ms 参数到consumer properties

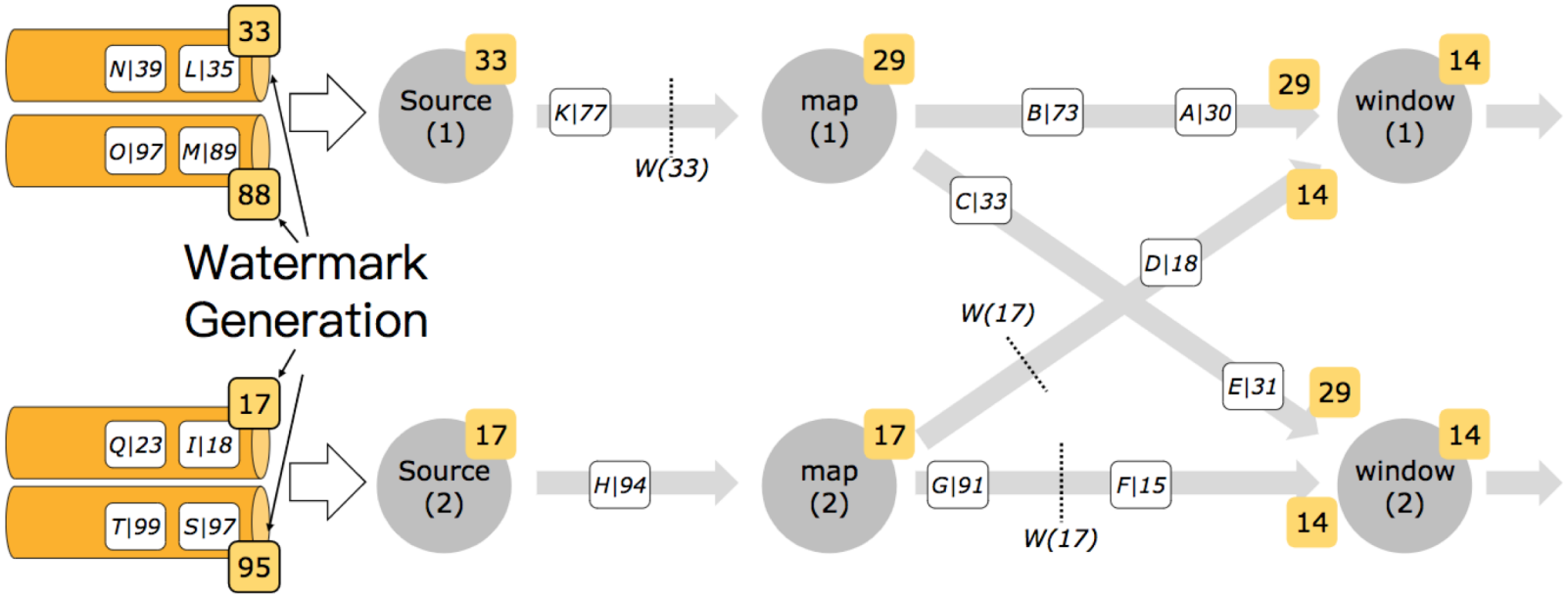
Checkpoint开启

- Offset自己在checkpoint state中管理和容错。 提交kafka 仅作为外部监视消费进度。
- 通过setCommitOffsetsOnCheckpoints控制，Checkpoint成功之后，是否提交offset到kafka

Flink Kafka Consumer–时戳提取/水位生成

per Kafka Partition watermark

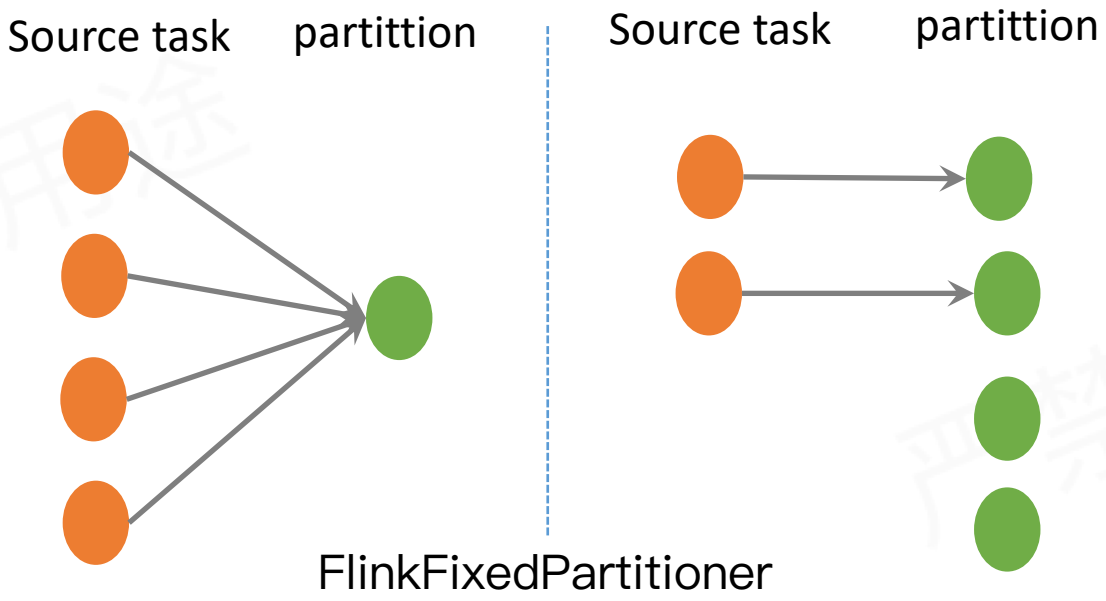
- assignTimestampsAndWatermarks，每个partition一个assigner，水位为多个partition对齐后值
- 不在kafka source后生成watermark，会出现扔掉部分数据情况。



Flink Kafka Producer–Producer分区

Producer 分区

- FlinkFixedPartitioner（默认）：parallelInstanceId % partitions.length
- Partittioner设置为null：round-robin kafka partitioner，维持过多链接
- custom partitioner：自定义分区



Flink Kafka Producer–Producer容错

Kafka 0.9 and 0.10

- setLogFailuresOnly：默认false。写失败时，是否只打印失败log，不抛异常
- setFlushOnCheckpoint：默认true。checkpoint时保证数据写到kafka。
- at–least–once语义：setLogFailuresOnly : flase + setFlushOnCheckpoint : true

Kafka 0.11

- FlinkKafkaProducer011，两阶段提交Sink结合kafka事务，可以保证端到端精准一次。

https://www.ververica.com/blog/end–to–end–exactly–once–processing–apache–flink–apache–kafka

03

Q&A

THANKS

Apache Flink 中文学习网站：ververica.cn