> 本文由 简悦 SimpRead 转码， 原文地址 kaiwu.lagou.com

在上一课时中我们提过在实时计算的场景下，绝大多数的数据源都是消息系统，而 Kafka 从众多的消息中间件中脱颖而出，主要是因为**高吞吐**、**低延迟**的特点；同时也讲了 Flink 作为生产者像 Kafka 写入数据的方式和代码实现。这一课时我们将从以下几个方面介绍 Flink 消费 Kafka 中的数据方式和源码实现。

Flink 如何消费 Kafka

Flink 在和 Kafka 对接的过程中，跟 Kafka 的版本是强相关的。上一课时也提到了，我们在使用 Kafka 连接器时需要引用相对应的 Jar 包依赖，对于某些连接器比如 Kafka 是有版本要求的，一定要去官方网站找到对应的依赖版本。

我们本地的 Kafka 版本是 2.1.0，所以需要对应的类是 FlinkKafkaConsumer。首先需要在 pom.xml 中引入 jar 包依赖：

```
<dependency>


<groupId>org.apache.flink</groupId>


<artifactId>flink-connector-kafka_2.11</artifactId>


<version>1.10.0</version>


</dependency>
```

下面将对 Flink 消费 Kafka 数据的方式进行分类讲解。

消费单个 Topic

上一课时我们在本地搭建了 Kafka 环境，并且手动创建了名为 test 的 Topic，然后向名为 test 的 Topic 中写入了数据。

那么现在我们要消费这个 Topic 中的数据，该怎么做呢？

```
public static void main(String[] args) throws Exception {


    StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();



env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);


    env.enableCheckpointing(5000);
```

```
    Properties properties = new Properties();

    properties.setProperty("bootstrap.servers", "127.0.0.1:9092");

    properties.setProperty("group.id", "group_test");

    FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>("test", new
SimpleStringSchema(), properties);

    consumer.setStartFromEarliest();

    env.addSource(consumer).flatMap(new FlatMapFunction<String, String>() {

@Override

public void flatMap(String value, Collector<String> out) throws Exception {

            System.out.println(value);

        }

    });

    env.execute("start consumer...");

}
```

在设置消费 Kafka 中的数据时，可以显示地指定从某个 Topic 的每一个 Partition 中进行消费。

消费多个 Topic

我们的业务中会有这样的情况，同样的数据根据类型不同发送到了不同的 Topic 中，比如线上的订单数据根据来源不同分别发往移动端和 PC 端两个 Topic 中。但是我们不想把同样的代码复制一份，需重新指定一个 Topic 进行消费，这时候应该怎么办呢？

```
Properties properties = new Properties();

properties.setProperty("bootstrap.servers", "127.0.0.1:9092");

properties.setProperty("group.id", "group_test");
```

```
FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>("test", new
SimpleStringSchema(), properties);


ArrayList<String> topics = new ArrayList<>();


        topics.add("test_A");


        topics.add("test_B");


        FlinkKafkaConsumer<Tuple2<String, String>> consumer = new
FlinkKafkaConsumer<>(topics, new SimpleStringSchema(), properties);


...
```

我们可以传入一个 list 来解决消费多个 Topic 的问题，如果用户需要区分两个 Topic 中的数据，那么需要在发往 Kafka 中数据新增一个字段，用来区分来源。

消息序列化

我们在上述消费 Kafka 消息时，都默认指定了消息的序列化方式，即 SimpleStringSchema。这里需要注意的是，在我们使用 SimpleStringSchema 的时候，返回的结果中只有原数据，没有 topic、parition 等信息，这时候可以自定义序列化的方式来实现自定义返回数据的结构。

```
public class CustomDeSerializationSchema implements
KafkaDeserializationSchema<ConsumerRecord<String, String>> {


@Override


public boolean isEndOfStream(ConsumerRecord<String, String> nextElement) {


return false;


    }


@Override


public ConsumerRecord<String, String> deserialize(ConsumerRecord<byte[], byte[]>
record) throws Exception {


return new ConsumerRecord<String, String>(


                record.topic(),
```

```
                record.partition(),

                record.offset(),

new String(record.key()),

new String(record.value())

        );

    }

@Override

public TypeInformation<ConsumerRecord<String, String>> getProducedType() {

return TypeInformation.of(new TypeHint<ConsumerRecord<String, String>>(){});

    }

}
```

这里自定义了 CustomDeSerializationSchema 信息，就可以直接使用了。

Parition 和 Topic 动态发现

在很多场景下，随着业务的扩展，我们需要对 Kafka 的分区进行扩展，为了防止新增的分区没有被及时发现导致数据丢失，消费者必须要感知 Partition 的动态变化，可以使用 FlinkKafkaConsumer 的动态分区发现实现。

我们只需要指定下面的配置，即可打开动态分区发现功能：每隔 10ms 会动态获取 Topic 的元数据，对于新增的 Partition 会自动从最早的位点开始消费数据。

```
properties.setProperty(FlinkKafkaConsumerBase.KEY_PARTITION_DISCOVERY_INTERVAL_MILLIS, "10");
```

如果业务场景需要我们动态地发现 Topic，可以指定 Topic 的正则表达式：

```
FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>
(Pattern.compile("^test_([A-Za-z0-9]*)$"), new SimpleStringSchema(),
properties);
```

Flink 消费 Kafka 设置 offset 的方法

Flink 消费 Kafka 需要指定消费的 offset，也就是**偏移量**。Flink 读取 Kafka 的消息有五种消费方式：

- 指定 Topic 和 Partition
- 从最早位点开始消费
- 从指定时间点开始消费
- 从最新的数据开始消费
- 从上次消费位点开始消费

```
* Flink从指定的topic和parition中指定的offset开始

*/

Map<KafkaTopicPartition, Long> offsets = new HashedMap();

offsets.put(new KafkaTopicPartition("test", 0), 10000L);

offsets.put(new KafkaTopicPartition("test", 1), 20000L);

offsets.put(new KafkaTopicPartition("test", 2), 30000L);

consumer.setStartFromSpecificOffsets(offsets);

* Flink从topic中最早的offset消费

*/

consumer.setStartFromEarliest();

* Flink从topic中指定的时间点开始消费

*/

consumer.setStartFromTimestamp(1559801580000l);

* Flink从topic中最新的数据开始消费

*/

consumer.setStartFromLatest();
```