Kafka Streams

Kafka 一直被认为是一个强大的消息中间件，它实现了高吞吐、高可用和低延时的消息传输能力，这让它成为流式处理系统中完美的数据来源。目前通用的一些流式处理框架如 Apache Spark、Apache Flink、Apache Storm 等都可以将 Kafka 作为可靠的数据来源。但遗憾的是，在 0.10.x 版本之前，Kafka 还并不具备任何数据处理的能力，但在此之后，Kafka Streams 应运而生。

Kafka Streams 是一个用于处理和分析数据的客户端库。它先把存储在 Kafka 中的数据进行处理和分析，然后将最终所得的数据结果回写到 Kafka 或发送到外部系统。它建立在一些非常重要的流式处理概念之上，例如适当区分事件时间和处理时间、窗口支持，以及应用程序状态的简单（高效）管理。同时，它也基于 Kafka 中的许多概念，例如通过划分主题进行扩展。此外，由于这个原因，它作为一个轻量级的库可以集成到应用程序中。这个应用程序可以根据需要独立运行、在应用程序服务器中运行、作为 Docker 容器，或者通过资源管理器（如 Mesos）进行操作。

Kafka Streams 直接解决了流式处理中的很多问题：

- 毫秒级延迟的逐个事件处理。
- 有状态的处理，包括连接（join）和聚合类操作。
- 提供了必要的流处理原语，包括高级流处理 DSL 和低级处理器 API。高级流处理 DSL 提供了常用流处理变换操作，低级处理器 API 支持客户端自定义处理器并与状态仓库交互。
- 使用类似 DataFlow 的模型对无序数据进行窗口化处理。
- 具有快速故障切换的分布式处理和容错能力。
- 无停机滚动部署。

单词统计是流式处理领域中最常见的示例，这里我们同样使用它来演示一下 Kafka Streams 的用法。在 Kafka 的代码中就包含了一个单词统计的示例程序，即 org.apache.kafka.streams.examples.wordcount.WordCountDemo，这个示例中以硬编码的形式用到了两个主题：streams-plaintext-input 和 streams-wordcount-output。为了能够使示例程序正常运行，我们需要预先准备好这两个主题，这两个主题的详细信息如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost: 2181/stream --
describe --topic streams-wordcount-
output,streams-plaintext-input
Topic:streams-plaintext-input   PartitionCount:1
ReplicationFactor:1 Configs:
    Topic: streams-plaintext-input  Partition: 0
Leader: 0   Replicas: 0 Isr: 0
Topic:streams-wordcount-output  PartitionCount:1
ReplicationFactor:1 Configs:
    Topic: streams-wordcount-output Partition: 0
Leader: 0   Replicas: 0 Isr: 0
```

之后我们就可以运行 WordCountDemo 这个示例了：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-run-
class.sh

org.apache.kafka.streams.examples.wordcount.WordC
ountDemo
```

这个示例程序将从主题 streams-plaintext-input 中读取消息，然后对读取的消息执行单词统计，并将结果持续写入主题 streams-wordcount-output。

之后打开一个 shell 终端，并启动一个生产者来为主题 streams-plaintext-input 输入一些单词，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic streams-plaintext-input
>
```

之后再打开另一个 shell 终端，并启动一个消费者来消费主题 streams-wordcount -output 中的消息，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic streams-wordcount-output --property print.key=true --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
```

现在我们往主题 streams-plaintext-input 中输入 hello kafka streams：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic streams-plaintext-input
>hello kafka streams
```

通过 WordCountDemo 处理之后会在消费端看到如下的结果：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-console-
consumer.sh --bootstrap-server localhost:9092 --
topic streams-wordcount-output --property
print.key=true --property
value.deserializer=org.apache.kafka.common.serial
ization.LongDeserializer
hello 1
kafka 1
streams 1
```

输出结果中的第一列是消息的 key，这里表示被计数的单词，第二列是消息的 value，这里表示该单词的最新计数。

现在继续往主题 streams-plaintext-input 中输入 I love kafka streams，然后会在消费端看到有新的消息输出：

```
I 1
love 1
kafka 2
streams 2
```

最后2行打印的 kafka 2和 streams 2表示计数已经从1递增到2。每当向输入主题（streams-plaintext-input）中写入更多的单词时，将观察到新的消息被添加到输出主题（streams-wordcount-output）中，表示由 WordCount 应用程序计算出的最新计数。

下面我们通过 WordCountDemo 程序来了解一下Kafka Streams的开发方式，WordCountDemo 程序如代码清单30-1所示，对应的 Maven 依赖如下所示。

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>2.0.0</version>
</dependency>
```

```
//代码清单30-1 单词统计示例
package
org.apache.kafka.streams.examples.wordcount;

import
org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Produced;
import java.util.Arrays;
import java.util.Locale;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;

public class WordCountDemo {
    public static void main(String[] args) {
        Properties props = new Properties();
①

props.put(StreamsConfig.APPLICATION_ID_CONFIG,
                  "streams-wordcount");

props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
                  "localhost:9092");
```

```
props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING
_CONFIG, 0);

props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_C
ONFIG,

Serdes.String().getClass().getName());

props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS
_CONFIG,

Serdes.String().getClass().getName());

        StreamsBuilder builder = new
StreamsBuilder();                          ②
        KStream<String, String> source = builder
                .stream("streams-plaintext-
input");                                   ③
        KTable<String, Long> counts = source
            .flatMapValues(value ->
Arrays.asList(

value.toLowerCase(Locale.getDefault())
                                .split(" ")))
            .groupBy((key, value) -> value)
            .count();
④

        counts.toStream().to("streams-wordcount-
output",
                Produced.with(Serdes.String(),
Serdes.Long()));                ⑤
```

```
        final KafkaStreams streams =
                new KafkaStreams(builder.build(),
props);                                    ⑥
        final CountDownLatch latch = new
CountDownLatch(1);
        Runtime.getRuntime().addShutdownHook(
                new Thread("streams-wordcount-
shutdown-hook") {
            @Override
            public void run() {
                streams.close();
⑦
                latch.countDown();
            }
        });

        try {
            streams.start();
⑧
            latch.await();
        } catch (Throwable e) {
            System.exit(1);
        }
        System.exit(0);
    }
}
```

第①行用于构建 Kafka Streams 的配置。每个 Kafka Streams 应用程序必须要有一个 application.id（StreamsConfig.APPLICATION_ID_CONFIG），这个 applicationId 用于协调应用实例，也用于命名内部的本地存储和相关主题。在整个 Kafka 集群中，applicationId 必须是唯一的。

bootstrap.servers 参数配置的是 Kafka 集群的地址，这个参数也是必需的。default.key.serde 和 default.value.serde 分别用来设置消息的 key 和 value 的序列化器。

第②行创建了一个 KStreamBuilder 实例，在第③行中通过调用 KStreamBuilder 实例的 stream() 方法创建了一个 KStream 实例，并设定了输入主题 streams-plaintext-input。

之后在第④行中执行具体的单词统计逻辑。注意这里引入了 KStream 和 KTable 的概念，它们是 Kafka Streams 的两种基本抽象。两者的区别在于：KStream 是一个由键值对构成的抽象记录流，每个键值对是一个独立单元，即使相同的key也不会被覆盖，类似数据库的插入操作；KTable 可以理解成一个基于表主键的日志更新流，相同 key 的每条记录只保存最新的一条记录，类似数据库中基于主键的更新。

无论记录流（用 KStream 定义），还是更新日志流（用 KTable 定义），都可以从一个或多个 Kafka 主题数据源来创建。一个 KStream 可以与另一个 KStream 或 KTable 进行 Join 操作，或者聚合成一个 KTable。同样，一个 KTable 也可以转换成一个 KStream。KStream 和 KTable 都提供了一系列转换操作，每个转换操作都可以转化为一个 KStream 或 KTable 对象，将这些转换操作连接在一起就构成了一个处理器拓扑。

第⑤行中调用 toStream().to() 来将单词统计的结果写入输出主题 streams-wordcount-output。注意计算结果中的消息的 key 是 String 类型，而 value 是 Long 类型，这一点在代码中有所呈现。

最终在第⑥和第⑧行中基于拓扑和配置来订阅一个 KafkaStreams 对象，并启动 Kafka Streams 引擎。整体上而言，Kafka Streams 的程序简单易用，用户只需关心流处理转换的具体逻辑而不需要关心底层的存储等细节内容。

本节只是简单地介绍一下 Kafka Streams，让读者对 Kafka Streams 有一个大致的概念。目前流式处理领域还是 Apache Spark 和 Apache Flink 的天下，其中 Apache Spark 的市场份额占有率最大，在后面我们会详细介绍 Apache Spark（包括 Spark Streaming 和 Structured Streaming），以及它和 Kafka 的整合应用。

从第27节到这里我们主要介绍 Kafka 现有的几个应用工具，对一般用户而言，这些应用工具已经足够应对大多数的场景。不过，我们还可以利用 Kafka 现有的特性和功能来扩展一些高级应用，比如延时（迟）队列、重试队列等，读者可以在[《图解Kafka之核心原理》(https://juejin.im/book/5c7d270ff265da2d89634e9e)](https://juejin.im/book/5c7d270ff265da2d89634e9e)中查阅相关的内容。