除了定义的topology，开发者还将需要在运行它之前在StreamsConfig配置他们的应用程序，Kafka Stream配置的完整列表可以在[这里](http://kafka.apache.org/documentation.html" \l "streamsconfigs)找到。

Kafka Streams中指定配置和生产者、消费者客户端类似，通常，你创建一个java.util.Properties，设置必要的参数，并通过Properties实例构建一个StreamsConfig实例。

import java.util.Properties;

import org.apache.kafka.streams.StreamsConfig;

Properties settings = new Properties();

// Set a few key parameters

settings.put(StreamsConfig.APPLICATION\_ID\_CONFIG, "my-first-streams-application");

settings.put(StreamsConfig.BOOTSTRAP\_SERVERS\_CONFIG, "kafka-broker1:9092");

settings.put(StreamsConfig.ZOOKEEPER\_CONNECT\_CONFIG, "zookeeper1:2181");

// Any further settings

settings.put(... , ...);

// Create an instance of StreamsConfig from the Properties instance

StreamsConfig config = new StreamsConfig(settings);

除了Kafka Streams自己配置参数，你也可以为Kafka内部的消费者和生产者指定参数。根据你应用的需要。类似于Streams设置，你可以通过StreamsConfig设置任何消费者和/或生产者配置。请注意，一些消费者和生产者配置参数使用相同的参数名。例如，用于配置TCP缓冲的send.buffer.bytes或receive.buffer.bytes。用于控制客户端请求重试的request.timeout.ms和retry.backoff.ms。如果需要为消费者和生产者设置不同的值，可以使用consumer.或producer.作为参数名称的前缀。

Properties settings = new Properties();

// Example of a "normal" setting for Kafka Streams

settings.put(StreamsConfig.BOOTSTRAP\_SERVERS\_CONFIG, "kafka-broker-01:9092");

// Customize the Kafka consumer settings

streamsSettings.put(ConsumerConfig.SESSION\_TIMEOUT\_MS\_CONFIG, 60000);

// Customize a common client setting for both consumer and producer

settings.put(CommonClientConfigs.RETRY\_BACKOFF\_MS\_CONFIG, 100L);

// Customize different values for consumer and producer

settings.put("consumer." + ConsumerConfig.RECEIVE\_BUFFER\_CONFIG, 1024 \* 1024);

settings.put("producer." + ProducerConfig.RECEIVE\_BUFFER\_CONFIG, 64 \* 1024);

// Alternatively, you can use

settings.put(StreamsConfig.consumerPrefix(ConsumerConfig.RECEIVE\_BUFFER\_CONFIG), 1024 \* 1024);

settings.put(StremasConfig.producerConfig(ProducerConfig.RECEIVE\_BUFFER\_CONFIG), 64 \* 1024);

你可以在应用程序代码中的任何地方使用Kafka Streams，常见的是在应用程序的main（）方法中使用。

首先，先创建一个KafkaStreams实例，其中构造函数的第一个参数用于定义一个topology builder（Streams DSL的KStreamBuilder，或Processor API的TopologyBuilder）。第二个参数是上面提到的StreamsConfig的实例。

import org.apache.kafka.streams.KafkaStreams;

import org.apache.kafka.streams.StreamsConfig;

import org.apache.kafka.streams.kstream.KStreamBuilder;

import org.apache.kafka.streams.processor.TopologyBuilder;

// Use the builders to define the actual processing topology, e.g. to specify

// from which input topics to read, which stream operations (filter, map, etc.)

// should be called, and so on.

KStreamBuilder builder = ...; // when using the Kafka Streams DSL

//

// OR

//

TopologyBuilder builder = ...; // when using the Processor API

// Use the configuration to tell your application where the Kafka cluster is,

// which serializers/deserializers to use by default, to specify security settings,

// and so on.

StreamsConfig config = ...;

KafkaStreams streams = new KafkaStreams(builder, config);

在这点上，内部结果已经初始化，但是处理还没有开始。你必须通过调用start()方法启动kafka Streams线程：

// Start the Kafka Streams instance

streams.start();

捕获任何意外的异常，设置java.lang.Thread.UncaughtExceptionHandler。每当流线程由于意外终止时，将调用此处理程序。

streams.setUncaughtExceptionHandler(new Thread.UncaughtExceptionHandler() {

public uncaughtException(Thread t, throwable e) {

// here you should examine the exception and perform an appropriate action!

}

);

close()方法结束程序。

// Stop the Kafka Streams instance

streams.close();

现在，运行你的应用程序，像其他的Java应用程序一样（Kafka Sterams没有任何特殊的要求）。同样，你也可以打包成jar，通过以下方式运行：

# Start the application in class `com.example.MyStreamsApp`

# from the fat jar named `path-to-app-fatjar.jar`.

$ java -cp path-to-app-fatjar.jar com.example.MyStreamsApp

当应用程序实例开始运行时，定义的处理器拓扑将被初始化成1个或多个流任务，可以由实例内的流线程并行的执行。如果处理器拓扑定义了状态仓库，则这些状态仓库在初始化流任务期间（重新）构建。这一点要理解，当如上所诉的启动你的应用程序时，实际上Kafka Streams认为你发布了一个实例。现实场景中，更常见的是你的应用程序有多个实例并行运行（如，其他的JVM中或别的机器上）。在这种情况下，Kafka Streams会将任务从现有的实例中分配给刚刚启动的新实例。有关详细的信息，请参阅流分区和任务和线程模型。

作者：半兽人  
链接：http://www.orchome.com/335  
来源：OrcHome  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。