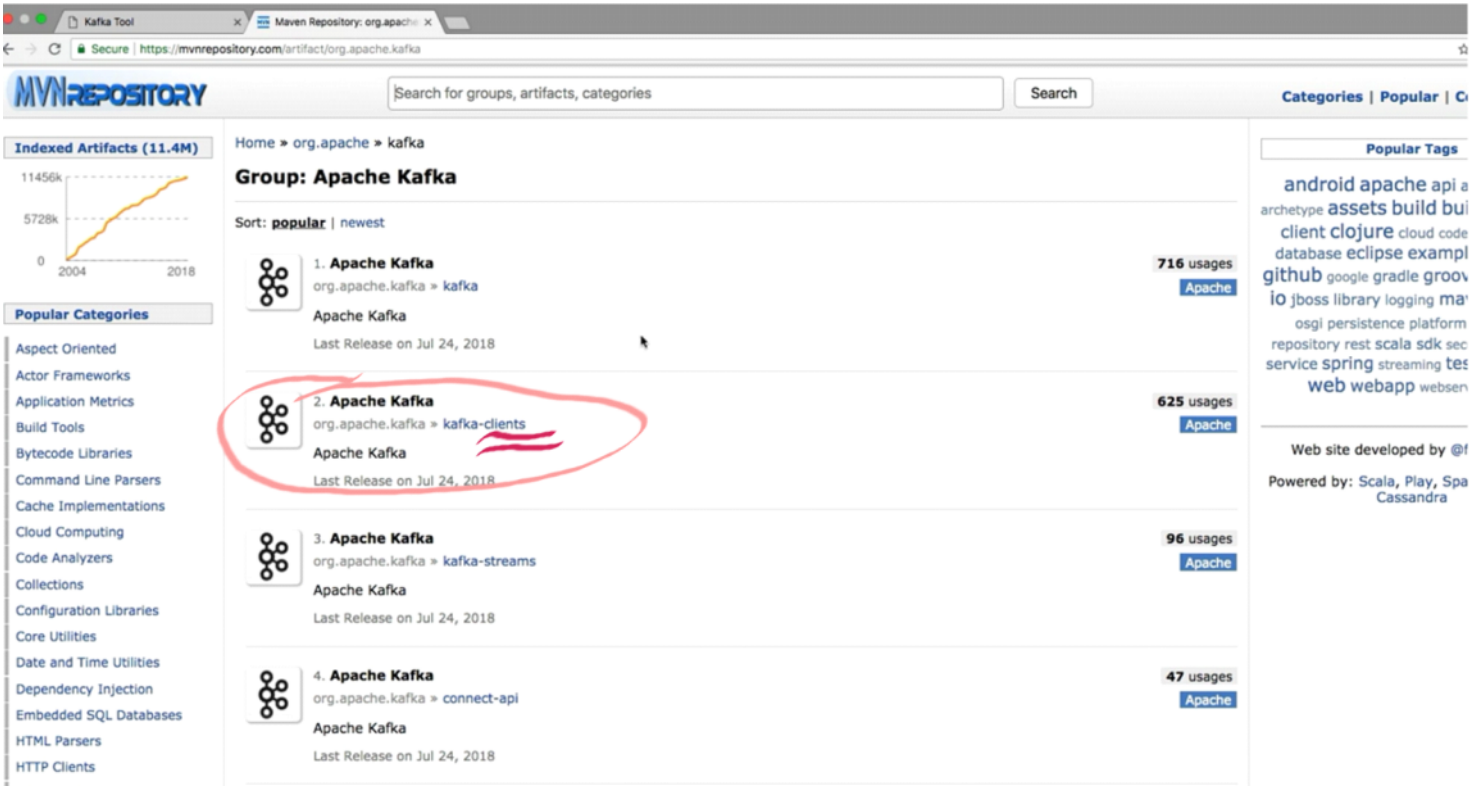
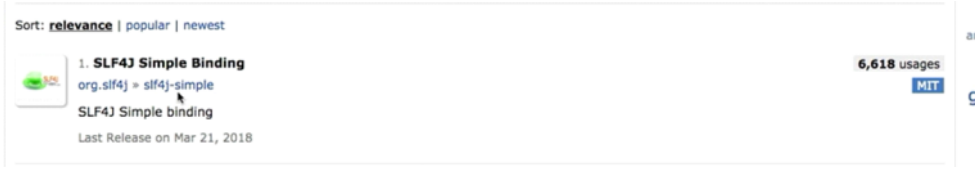
producer 1

29 ноября 2020 г. 21:12





пример продьюсера 1

```
/.../

package org.apache.kafka.clients.producer;

import ...

public class ProducerConfig extends AbstractConfig {
    private static final ConfigDef CONFIG;
    public static final String BOOTSTRAP_SERVERS_CONFIG = "bootstrap.servers";
```

```java
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;

import java.util.Properties;

public class ProducerDemo {

    public static void main(String[] args) {

        String bootstrapServers = "127.0.0.1:9092";

        // create Producer properties
        Properties properties = new Properties();
        properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        properties.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

        // create the producer
        KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

        // create a producer record
        ProducerRecord<String, String> record =
                new ProducerRecord<>( topic: "first_topic",  value: "hello world");

        // send data
        producer.send(record);

        // flush data
        producer.flush();
        // flush and close producer
        producer.close();
```

пример продьюсера 2 (с калбаком)

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

    final Logger logger = LoggerFactory.getLogger(ProducerDemoWithCallback.class);

    String bootstrapServers = "127.0.0.1:9092";

    // create Producer properties
    Properties properties = new Properties();
    properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
    properties.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
    properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

    // create the producer
    KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

    // create a producer record
    ProducerRecord<String, String> record =
            new ProducerRecord<>( topic: "first_topic",  value: "hello world");

    // send data - asynchronous
    producer.send(record, new Callback() {
        public void onCompletion(RecordMetadata recordMetadata, Exception e) {
            // executes every time a record is successfully sent or an exception is thrown
            if (e != null){
                // the record was successfully sent
                logger.info("Received new metadata. \n" +
                            "Topic:" + recordMetadata.topic() + "\n" +
                            "Partition:" + recordMetadata.partition() + "\n" +
                            "Offset: " + recordMetadata.offset() + "\n" +
                            "Timestamp" + recordMetadata.timestamp());
            } else {
                logger.error("Error while producing", e);
            }
```

пример продьюсера 3

Listing 2.3 SimpleProducer example

```java
Properties properties = new Properties();
properties.put("bootstrap.servers", "localhost:9092");
properties.put("key.serializer", "org.apache.kafka.common.serialization.Strin
    gSerializer");
properties.put("value.serializer",
➥ "org.apache.kafka.common.serialization.StringSerializer");
properties.put("acks", "1");
properties.put("retries", "3");
properties.put("compression.type", "snappy");
properties.put("partitioner.class",
➥ PurchaseKeyPartitioner.class.getName());

PurchaseKey key = new PurchaseKey("12334568", new Date());

try(Producer<PurchaseKey, String> producer =
➥ new KafkaProducer<>(properties)) {
    ProducerRecord<PurchaseKey, String>  record =
➥ new ProducerRecord<>("transactions", key, "{\"item\":\"book\",
    \"price\":10.99}");

  Callback callback = (metadata, exception) -> {
            if (exception != null) {
                System.out.println("Encountered exception "
➥ + exception);
            }
        };

  Future<RecordMetadata> sendFuture =
➥ producer.send(record, callback);
}
```

Properties for configuring a producer

Creates the KafkaProducer

Instantiates the ProducerRecord

Builds a callback

Sends the record and sets the returned Future to a variable

при отправке сообщения при заполненном буфере генератора вы можете столкнуться с блокировкой на методе Producer.send

- Генераторы Kafka являются потокобезопасными. Отправка данных в Kafka производится асинхронно — возврат из метода Producer.send

происходит сразу же после помещения генератором записи во внутренний буфер. Этот буфер отправляет записи пакетами. В зависимости от ваших настроек при отправке сообщения при заполненном буфере генератора вы можете столкнуться с блокировкой.

Future дает возможность извлечь результаты асинхронных операций отложенным образом, вместо того чтобы ждать их завершения

- В листинге 2.3 метод Producer.send возвращает объект Future, который представляет собой результат выполнения асинхронной операции. Что важнее, Future дает возможность извлечь результаты асинхронных операций отложенным образом, вместо того чтобы ждать их завершения. Более подробную информацию о фьючерсах вы можете найти в документации Java, в разделе «Интерфейс Future<V>»: http:// mng.bz/0JK2.

пример 4

Listing 3.3 IProducer.java

```
package nile;

import java.util.Properties;

import org.apache.kafka.clients.producer.*;

public interface IProducer {

  public void process(String message);

  public static void write(KafkaProducer<String, String> producer,
    String topic, String message) {
    ProducerRecord<String, String> pr = new ProducerRecord(
      topic, message);
    producer.send(pr);
  }

  public static Properties createConfig(String servers) {
    Properties props = new Properties();
    props.put("bootstrap.servers", servers);
    props.put("acks", "all");
    props.put("retries", 0);
    props.put("batch.size", 1000);
    props.put("linger.ms", 1);
    props.put("key.serializer",
      "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer",
      "org.apache.kafka.common.serialization.StringSerializer");
    return props;
  }
}
```

Our abstract process method, for concrete implementations of IProducer to instantiate

A static helper to write a record to a Kafka topic

A static helper to configure a Kafka producer

Listing 3.4 PassthruProducer.java

```
package nile;

import org.apache.kafka.clients.producer.*;

public class PassthruProducer implements IProducer {

  private final KafkaProducer<String, String> producer;
  private final String topic;

  public PassthruProducer(String servers, String topic) {
    this.producer = new KafkaProducer(
      IProducer.createConfig(servers));
    this.topic = topic;
  }

  public void process(String message) {
    IProducer.write(this.producer, this.topic, message);
  }
}
```

Use the IProducer interface's createConfig function to configure the producer.

Write each supplied record out to the specified Kafka topic.

Listing 3.6 FullProducer.java

```
package nile;

import com.fasterxml.jackson.databind.*;
import com.fasterxml.jackson.databind.node.ObjectNode;

import java.net.InetAddress;
import org.apache.kafka.clients.producer.*;
```



```
import com.maxmind.geoip2.*;
import com.maxmind.geoip2.model.*

public class FullProducer implements IProducer {

  private final KafkaProducer<String, String> producer;
  private final String goodTopic;
  private final String badTopic;
  private final DatabaseReader maxmind;

  protected static final ObjectMapper MAPPER = new ObjectMapper();

  public FullProducer(String servers, String goodTopic,
    String badTopic, DatabaseReader maxmind) {
    this.producer = new KafkaProducer(
      IProducer.createConfig(servers));
    this.goodTopic = goodTopic;
    this.badTopic = badTopic;
    this.maxmind = maxmind;
  }

  public void process(String message) {

    try {
      JsonNode root = MAPPER.readTree(message);
      JsonNode ipNode = root.path("shopper").path("ipAddress");
      if (ipNode.isMissingNode()) {
        IProducer.write(this.producer, this.badTopic,
          "{\"error\": \"shopper.ipAddress missing\"}");
      } else {
        InetAddress ip = InetAddress.getByName(ipNode.textValue());
        CityResponse resp = maxmind.city(ip);
        ((ObjectNode)root).with("shopper").put(
          "country", resp.getCountry().getName());
        ((ObjectNode)root).with("shopper").put(
          "city", resp.getCity().getName());
        IProducer.write(this.producer, this.goodTopic,
          MAPPER.writeValueAsString(root));
      }
    } catch (Exception e) {
      IProducer.write(this.producer, this.badTopic, "{\"error\": \"" +
        e.getClass().getSimpleName() + ": " + e.getMessage() + "\"}");
    }
  }
}
```

Constructor takes good and bad Kafka topics for writing, plus the MaxMind geo-IP lookup service

Retrieves the ipAddress from the shopper object within the incoming event

Looks up the shopper's location based on shopper's IP address

Adds the shopper's country and city to the event

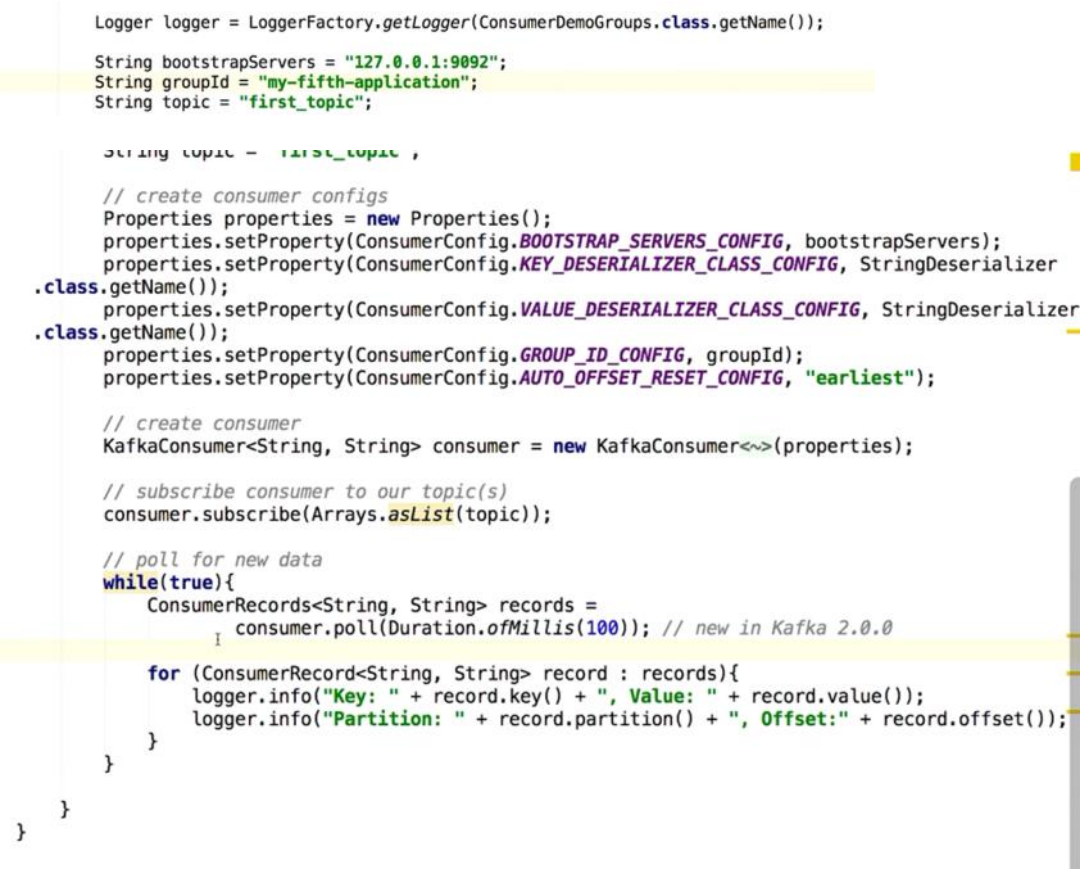
Writes the now-enriched event out to our "good" Kafka topic

In case of validation or processing failure, writes error message out to "bad" Kafka topic

consumer 1

29 ноября 2020 г. 21:45

пример консьюмера 1



```
        Logger logger = LoggerFactory.getLogger(ConsumerDemoGroups.class.getName());

        String bootstrapServers = "127.0.0.1:9092";
        String groupId = "my-fifth-application";
        String topic = "first_topic";

        // create consumer configs
        Properties properties = new Properties();
        properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer
.class.getName());
        properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer
.class.getName());
        properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        properties.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        // create consumer
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);

        // subscribe consumer to our topic(s)
        consumer.subscribe(Arrays.asList(topic));

        // poll for new data
        while(true){
            ConsumerRecords<String, String> records =
                    consumer.poll(Duration.ofMillis(100)); // new in Kafka 2.0.0

            for (ConsumerRecord<String, String> record : records){
                logger.info("Key: " + record.key() + ", Value: " + record.value());
                logger.info("Partition: " + record.partition() + ", Offset:" + record.offset());
            }
        }

    }
}
```

пример консьюмера 2 (с тредами)

```
package kafka.tutorial1;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.errors.WakeupException;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
```

```
public class ConsumerDemoWithThread {

    public static void main(String[] args) {
        new ConsumerDemoWithThread().run();
    }

    private ConsumerDemoWithThread() {

    }

    private void run() {
        Logger logger = LoggerFactory.getLogger(ConsumerDemoWithThread.class.getName());

        String bootstrapServers = "127.0.0.1:9092";
        String groupId = "my-sixth-application";
        String topic = "first_topic";

        // latch for dealing with multiple threads
        CountDownLatch latch = new CountDownLatch(1);

        // create the consumer runnable
        logger.info("Creating the consumer thread");
        Runnable myConsumerRunnable = new ConsumerRunnable(
                bootstrapServers,
                groupId,
                topic,
                latch
        );

        // start the thread
        Thread myThread = new Thread(myConsumerRunnable);
        myThread.start();

        // add a shutdown hook
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            logger.info("Caught shutdown hook");
            ((ConsumerRunnable) myConsumerRunnable).shutdown();
            try {
                latch.await();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            logger.info("Application has exited");
        }

        ));

        try {
            latch.await();
        } catch (InterruptedException e) {
            logger.error("Application got interrupted", e);
        } finally {
            logger.info("Application is closing");
        }
    }

    public class ConsumerRunnable implements Runnable {

        private CountDownLatch latch;
```

```
    private KafkaConsumer<String, String> consumer;
    private Logger logger = LoggerFactory.getLogger(ConsumerRunnable.class.getName());

    public ConsumerRunnable(String bootstrapServers,
                            String groupId,
                            String topic,
                            CountDownLatch latch) {
      this.latch = latch;

      // create consumer configs
      Properties properties = new Properties();
      properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
      properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
      properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
      properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
      properties.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

      // create consumer
      consumer = new KafkaConsumer<String, String>(properties);
      // subscribe consumer to our topic(s)
      consumer.subscribe(Arrays.asList(topic));
    }

    @Override
    public void run() {
      // poll for new data
      try {
        while (true) {
          ConsumerRecords<String, String> records =
              consumer.poll(Duration.ofMillis(100)); // new in Kafka 2.0.0

          for (ConsumerRecord<String, String> record : records) {
            logger.info("Key: " + record.key() + ", Value: " + record.value());
            logger.info("Partition: " + record.partition() + ", Offset:" + record.offset());
          }
        }
      } catch (WakeupException e) {
        logger.info("Received shutdown signal!");
      } finally {
        consumer.close();
        // tell our main code we're done with the consumer
        latch.countDown();
      }
    }

    public void shutdown() {
      // the wakeup() method is a special method to interrupt consumer.poll()
      // it will throw the exception WakeUpException
      consumer.wakeup();
    }
  }
}
```

пример3 отмотать офсет и поискать сообщение (прочтем 5 сообщений начиная с офсета==15)

```
String bootstrapServers = "127.0.0.1:9092";
        String topic = "first_topic";

        // create consumer configs
        Properties properties = new Properties();
        properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        properties.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        // create consumer
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(properties);

        // assign and seek are mostly used to replay data or fetch a specific message

        // assign
        TopicPartition partitionToReadFrom = new TopicPartition(topic, 0);
        long offsetToReadFrom = 15L;
        consumer.assign(Arrays.asList(partitionToReadFrom));

        // seek
        consumer.seek(partitionToReadFrom, offsetToReadFrom);

        int numberOfMessagesToRead = 5;
        boolean keepOnReading = true;
        int numberOfMessagesReadSoFar = 0;

        // poll for new data
        while(keepOnReading){
           ConsumerRecords<String, String> records =
                 consumer.poll(Duration.ofMillis(100)); // new in Kafka 2.0.0

           for (ConsumerRecord<String, String> record : records){
              numberOfMessagesReadSoFar += 1;
              logger.info("Key: " + record.key() + ", Value: " + record.value());
              logger.info("Partition: " + record.partition() + ", Offset:" + record.offset());
              if (numberOfMessagesReadSoFar >= numberOfMessagesToRead){
                 keepOnReading = false; // to exit the while loop
                 break; // to exit the for loop
              }
           }
        }

        logger.info("Exiting the application");
```

пример 4

Listing 3.2 Consumer.java

```
package nile;

import java.util.*;

import org.apache.kafka.clients.consumer.*;

public class Consumer {

  private final KafkaConsumer<String, String> consumer;
  private final String topic;

  public Consumer(String servers, String groupId, String topic) {
    this.consumer = new KafkaConsumer<String, String>(
      createConfig(servers, groupId));
    this.topic = topic;
  }

  public void run(IProducer producer) {
    this.consumer.subscribe(Arrays.asList(this.topic));
    while (true) {
      ConsumerRecords<String, String> records = consumer.poll(100);
      for (ConsumerRecord<String, String> record : records) {
        producer.process(record.value());
      }
    }
  }

  private static Properties createConfig(String servers, String groupId) {
    Properties props = new Properties();
    props.put("bootstrap.servers", servers);
    props.put("group.id", groupId);
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "1000");
    props.put("auto.offset.reset", "earliest");
    props.put("session.timeout.ms", "30000");
    props.put("key.deserializer",
      "org.apache.kafka.common.serialization.StringDeserializer");
    props.put("value.deserializer",
      "org.apache.kafka.common.serialization.StringDeserializer");
    return props;
  }
}
```

Our Kafka consumer will read Kafka records for which the key and value are both strings.

Subscribe our consumer to the given Kafka topic.

Looping forever, fetch records from the Kafka topic.

Feed each record's value to the process method of our producer.

Identify this consumer as belonging to a specific consumer group.

client confluent (aka microservice)

15 декабря 2020 г. 12:06

```
package clients;
import java.time.Duration;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Properties;
import com.fasterxml.jackson.databind.JsonNode;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.kafka.common.serialization.StringSerializer;
import org.apache.kafka.connect.json.JsonDeserializer;
public class Sample {
    public static void main(String[] args){
        System.out.println("*** Starting Microservice ***");

        KafkaConsumer<String, JsonNode> consumer = getConsumer();
        KafkaProducer<String, DoorStatusChanged> producer = getProducer();
        try {
            consumer.subscribe(Arrays.asList("TRAM_DOOR_STATUS"));
            HashMap<String, Integer> cache = new HashMap<>();
            while (true) {
                ConsumerRecords<String, JsonNode> records = consumer.poll(Duration.ofMillis(100));
                for (ConsumerRecord<String, JsonNode> record : records) {
                    System.out.printf("offset = %d, key = %s, value = %s\n", record.offset(), record.key(),
                            record.value());
                    JsonNode node = record.value();
                    String operator = node.get("OPER").asText();
                    String designation = node.get("DESI").asText();
                    String vehicleNo = node.get("VEH").asText();
                    int doorStatus = node.get("DRST").asInt();
                    // assume combination of operator and vehicleNo is unique
                    String key = operator + "|" + vehicleNo;
                    boolean hasChanged = true;
                    if (cache.containsKey(key)) {
                        int prevDoorStatus = cache.get(key);
                        hasChanged = prevDoorStatus != doorStatus;
                    }
                    if (hasChanged) {
                        publishEvent(producer, operator, designation, vehicleNo, doorStatus);
                    }
                    cache.put(key, doorStatus);
                }
            }
        } finally {
            System.out.println("*** Ending Microservice ***");
            consumer.close();
        }
    }
    private static KafkaConsumer<String, JsonNode> getConsumer() {
        Properties settings = new Properties();
        settings.put(ConsumerConfig.GROUP_ID_CONFIG, "tram-door-status");
        settings.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka:9092");
        settings.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        settings.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        settings.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);
        KafkaConsumer<String, JsonNode> consumer = new KafkaConsumer<>(settings);
        return consumer;
    }
    private static KafkaProducer<String, DoorStatusChanged> getProducer(){
        Properties settings = new Properties();
        settings.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka:9092");
        settings.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        settings.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
        return new KafkaProducer<>(settings);
    }
    private static void publishEvent(KafkaProducer<String,DoorStatusChanged> producer, String operator, String designation, String vehicleNo, int
 doorStatus){
        String type = doorStatus == 0 ? "DOOR_CLOSED" : "DOOR_OPENED";
        DoorStatusChanged event = new DoorStatusChanged(operator, designation, vehicleNo, type);
        ProducerRecord<String, DoorStatusChanged> record = new ProducerRecord<>("tram-door-status-changed",operator,event);
        producer.send(record);
    }
}
```

пример1

Listing 2.5 `ThreadedConsumerExample` example

```
public void startConsuming() {
        executorService = Executors.newFixedThreadPool(numberPartitions);
        Properties properties = getConsumerProps();

        for (int i = 0; i < numberPartitions; i++) {
            Runnable consumerThread = getConsumerThread(properties);
            executorService.submit(consumerThread);
        }
    }

    private Runnable getConsumerThread(Properties properties) {
        return () -> {


           Consumer<String, String> consumer = null;
           try {
               consumer = new KafkaConsumer<>(properties);
               consumer.subscribe(Collections.singletonList(
➥ "test-topic"));
               while (!doneConsuming) {
                   ConsumerRecords<String, String> records =
➥ consumer.poll(5000);
                   for (ConsumerRecord<String, String> record : records) {
                       String message = String.format("Consumed: key =
➥ %s value = %s with offset = %d partition = %d",
                               record.key(), record.value(),
                               record.offset(), record.partition());
                       System.out.println(message);
                   }

               }
           } catch (Exception e) {
               e.printStackTrace();
           } finally {
               if (consumer != null) {
                   consumer.close();
               }
           }
       };
    }
```

Builds a consumer thread

Subscribes to the topic

Polls for 5 seconds

Prints a formatted message

Closes the consumer—will leak resources otherwise

consumer confluent

15 декабря 2020 г. 14:50

```java
package clients;
import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
public class VehiclePositionConsumer {
    public static void main(String[] args) {
        System.out.println("*** Starting VP Consumer ***");

        Properties settings = new Properties();
        settings.put(ConsumerConfig.GROUP_ID_CONFIG, "vp-consumer");
        settings.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka:9092");
        settings.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        settings.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        settings.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(settings);
        try {
            consumer.subscribe(Arrays.asList("vehicle-positions"));
            while (true) {
                ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
                for (ConsumerRecord<String, String> record : records)
                    System.out.printf("offset = %d, key = %s, value = %s\n", record.offset(), record.key(), record.value());
            }
        }
        finally{
            System.out.println("*** Ending VP Consumer ***");
            consumer.close();
        }
    }
}
```

Spring Cloud Streams

26 января 2021 г. 20:32

Spring Cloud offers Spring Cloud Streams[50] for implementing applications for the processing ofdata streams. This library supports messaging systems such as Kafka (see also chapter 11), RabbitMQ (see above), and Redis[51]. Spring Cloud Streams builds on these technologies and extends them with concepts such as streams, and therefore goes beyond just simplifying the use of the technology's APIs.

streams app

31 января 2021 г. 21:04

Listing 3.4 Hello World: the Yelling App

```
public class KafkaStreamsYellingApp {

 public static void main(String[] args) {


  Properties props = new Properties();

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


  StreamsConfig streamingConfig = new StreamsConfig(props);

  Serde<String> stringSerde = Serdes.String();

  StreamsBuilder builder = new StreamsBuilder();

  KStream<String, String> simpleFirstStream = builder.stream("src-topic",
  ➥ Consumed.with(stringSerde, stringSerde));

  KStream<String, String> upperCasedStream =
  ➥ simpleFirstStream.mapValues(String::toUpperCase);

  upperCasedStream.to( "out-topic",
  ➥ Produced.with(stringSerde, stringSerde));

  KafkaStreams kafkaStreams = new KafkaStreams(builder.build(),streamsConfig);

  kafkaStreams.start();
  Thread.sleep(35000);
  LOG.info("Shutting down the Yelling APP now");
  kafkaStreams.close();

  }
}
```

Creates the StreamsConfig with the given properties

Properties for configuring the Kafka Streams program

Creates the Serdes used to serialize/ deserialize keys and values

Creates the StreamsBuilder instance used to construct the processor topology

Creates the actual stream with a source topic to read from (the parent node in the graph)

A processor using a Java 8 method handle (the first child node in the graph)

Writes the transformed output to another topic (the sink node in the graph)

Kicks off the Kafka Streams threads