/home/español/java/¿Cómo uso múltiples consumidores en Kafka?

¿Cómo uso múltiples consumidores en Kafka?

Soy un nuevo estudiante que estudia Kafka y me he encontrado con algunos problemas fundamentales al comprender a múltiples consumidores con los que los artículos, documentaciones, etc. no han sido demasiado útiles hasta ahora.

Una cosa que he intentado hacer es escribir mi propio productor y consumidor de alto nivel Kafka y ejecutarlos simultáneamente, publicando 100 mensajes simples a un tema y haciendo que mi consumidor los recupere. Logré hacerlo con éxito, pero cuando trato de presentar un segundo consumidor para que consuma del mismo tema en el que se acaban de publicar los mensajes, no recibe ningún mensaje.

Entendí que para cada tema, podría tener consumidores de grupos de consumidores separados y cada uno de estos grupos de consumidores obtendría una copia completa de los mensajes producidos para algún tema. ¿Es esto correcto? Si no es así, ¿cuál sería la forma correcta para mí de configurar múltiples consumidores? Esta es la clase de consumidor que he escrito hasta ahora:

```
public class AlternateConsumer extends Thread {
    private final KafkaConsumer<Integer, String> consumer;
    private final String topic;
    private final Boolean isAsync = false;

    public AlternateConsumer(String topic, String consumerGroup) {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", "localhost:9092");
        properties.put("group.id", consumerGroup);
        properties.put("partition.assignment.strategy", "roundrobin");
        properties.put("enable.auto.commit", "true");
        properties.put("auto.commit.interval.ms", "1000");
```

it-swarm.dev

```
        properties.put("session.timeout.ms", "30000");
        properties.put("key.deserializer", "org.Apache.kafka.common.serialization.IntegerDe
serializer");
        properties.put("value.deserializer", "org.Apache.kafka.common.serialization.StringD
eserializer");

        consumer = new KafkaConsumer<Integer, String>(properties);
        consumer.subscribe(topic);
        this.topic = topic;
    }


    public void run() {
        while (true) {
            ConsumerRecords<Integer, String> records = consumer.poll(0);
            for (ConsumerRecord<Integer, String> record : records) {
                System.out.println("We received message: " + record.value() + " from topic:
" + record.topic());
            }
        }

    }
}
```

Además, noté que originalmente estaba probando el consumo anterior para un tema 'prueba' con solo una partición. Cuando agregué otro consumidor a un grupo de consumidores existente que decía 'testGroup', esto desencadenó un Kafka reequilibrio que ralentizó la latencia de mi consumo en una cantidad significativa, en la magnitud de segundos. Pensé que este era un problema con el reequilibrio ya que solo tenía una única partición, pero cuando creé un nuevo tema 'particiones múltiples' con, digamos, 6 particiones, surgieron problemas similares en los que agregar más consumidores al mismo grupo de consumidores causó problemas de latencia. He mirado a mi alrededor y la gente me dice que debería usar un consumidor multiproceso. ¿Alguien puede arrojar luz sobre eso?

java bigdata apache-kafka

29 Jeff Gong 17 jun. 2015

Creo que su problema radica en la propiedad auto.offset.reset. Cuando un

nuevo consumidor lee desde una partición y no hay un desplazamiento comprometido anterior, la propiedad auto.offset.reset se usa para decidir cuál debe ser el desplazamiento inicial. Si lo configura en "mayor" (predeterminado), comenzará a leer el último (último) mensaje. Si lo configura en "más pequeño", obtendrá el primer mensaje disponible.

Entonces agregue:

```
properties.put("auto.offset.reset", "smallest");
```

e intenta de nuevo.

*** editar ***

"más pequeño" y "más grande" fueron obsoletos hace un tiempo. Debe usar "más temprano" o "más reciente" ahora. Cualquier pregunta, consulte el docs

20 Chris Gerken 17 jun. 2015

Si desea que varios consumidores consuman los mismos mensajes (como una transmisión), puede generarlos con un grupo de consumidores diferente y también configurar auto.offset.reset al más pequeño en la configuración del consumidor. Si desea que varios consumidores terminen de consumir en paralelo (divida el trabajo entre ellos), debe crear un número de particiones> = número de consumidores. Una partición solo puede ser consumida por un proceso de consumidor como máximo Pero un consumidor puede consumir más de una partición.

7 user1119541 11 may. 2016

En la documentación aquí dice: "si proporciona más hilos que particiones sobre el tema, algunos hilos nunca verán un mensaje". ¿Puedes agregar particiones a tu tema? Tengo el recuento de hilos de mi grupo de consumidores igual al número de particiones en mi tema, y cada hilo recibe mensajes.

Aquí está mi configuración de tema:

```
buffalo-macbook10:kafka_2.10-0.8.2.1 aakture$ bin/kafka-topics.sh --describe --zookeeper
 localhost:2181 --topic recent-wins
Topic:recent-wins    PartitionCount:3    ReplicationFactor:1 Configs:
Topic: recent-wins  Partition: 0    Leader: 0   Replicas: 0 Isr: 0

Topic: recent-wins  Partition: 1    Leader: 0   Replicas: 0 Isr: 0
Topic: recent-wins  Partition: 2    Leader: 0   Replicas: 0 Isr: 0
```

Y mi consumidor:

```
package com.cie.dispatcher.services;

import com.cie.dispatcher.model.WinNotification;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.inject.Inject;
import io.dropwizard.lifecycle.Managed;
import kafka.consumer.ConsumerConfig;
import kafka.consumer.ConsumerIterator;
import kafka.consumer.KafkaStream;
import kafka.javaapi.consumer.ConsumerConnector;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import Java.util.HashMap;
import Java.util.List;
import Java.util.Map;
import Java.util.Properties;
import Java.util.concurrent.ExecutorService;
import Java.util.concurrent.Executors;
import Java.util.concurrent.TimeUnit;

/**
 * This will create three threads, assign them to a "group" and listen for  notifications
on a topic.
 * Current setup is to have three partitions in Kafka, so we need a thread per partition
 (as recommended by
 * the kafka folks). This implements the dropwizard Managed interface, so it can be start
ed and stopped by the
 * lifecycle manager in dropwizard.
 *   <p />
 * Created by aakture on 6/15/15.
 */
public class KafkaTopicListener implements Managed {
private static final Logger LOG = LoggerFactory.getLogger(KafkaTopicListener.class);
private final ConsumerConnector consumer;
private final String topic;
private ExecutorService executor;
private int threadCount;
```

```
    private WinNotificationWorkflow winNotificationWorkflow;
    private ObjectMapper objectMapper;

    @Inject
    public KafkaTopicListener(String a_zookeeper,
                              String a_groupId, String a_topic,
                              int threadCount,
                              WinNotificationWorkflow winNotificationWorkflow,
                              ObjectMapper objectMapper) {
        consumer = kafka.consumer.Consumer.createJavaConsumerConnector(
                createConsumerConfig(a_zookeeper, a_groupId));
        this.topic = a_topic;
        this.threadCount = threadCount;
        this.winNotificationWorkflow = winNotificationWorkflow;
        this.objectMapper = objectMapper;
    }

    /**
     * Creates the config for a connection
     *
     * @param zookeeper the Host:port for zookeeper, "localhost:2181" for example.
     * @param groupId   the group id to use for the consumer group. Can be anything, it's use
    d by kafka to organize the consumer threads.
     * @return the config props
     */
    private static ConsumerConfig createConsumerConfig(String zookeeper, String groupId) {
        Properties props = new Properties();
        props.put("zookeeper.connect", zookeeper);
        props.put("group.id", groupId);
        props.put("zookeeper.session.timeout.ms", "400");
        props.put("zookeeper.sync.time.ms", "200");
        props.put("auto.commit.interval.ms", "1000");

        return new ConsumerConfig(props);
    }

    public void stop() {
        if (consumer != null) consumer.shutdown();
        if (executor != null) executor.shutdown();
        try {
            if (!executor.awaitTermination(5000, TimeUnit.MILLISECONDS)) {
                LOG.info("Timed out waiting for consumer threads to shut down, exiting unclea
    nly");
            }
        } catch (InterruptedException e) {
            LOG.info("Interrupted during shutdown, exiting uncleanly");
        }
        LOG.info("{} shutdown successfully", this.getClass().getName());
    }
    /**
```

```
,
 * Starts the listener
 */
public void start() {
    Map<String, Integer> topicCountMap = new HashMap<>();
    topicCountMap.put(topic, new Integer(threadCount));

    Map<String, List<KafkaStream<byte[], byte[]>>> consumerMap = consumer.createMessageSt
reams(topicCountMap);
    List<KafkaStream<byte[], byte[]>> streams = consumerMap.get(topic);
    executor = Executors.newFixedThreadPool(threadCount);
    int threadNumber = 0;
    for (final KafkaStream stream : streams) {
        executor.submit(new ListenerThread(stream, threadNumber));
        threadNumber++;
    }
}

private class ListenerThread implements Runnable {
    private KafkaStream m_stream;
    private int m_threadNumber;

    public ListenerThread(KafkaStream a_stream, int a_threadNumber) {
        m_threadNumber = a_threadNumber;
        m_stream = a_stream;
    }

    public void run() {
        try {
            String message = null;
            LOG.info("started listener thread: {}", m_threadNumber);
            ConsumerIterator<byte[], byte[]> it = m_stream.iterator();
            while (it.hasNext()) {
                try {
                    message = new String(it.next().message());
                    LOG.info("receive message by " + m_threadNumber + " : " + message);
                    WinNotification winNotification = objectMapper.readValue(message, Win
Notification.class);
                    winNotificationWorkflow.process(winNotification);
                } catch (Exception ex) {
                    LOG.error("error processing queue for message: " + message, ex);
                }
            }
            LOG.info("Shutting down listener thread: " + m_threadNumber);
        } catch (Exception ex) {
            LOG.error("error:", ex);
        }
    }
  }
}
```

5 Alper Akture 17 jun. 2015

¿Qué determina la compensación del consumidor de Kafka?

Cómo eliminar un tema en Apache kafka

Java, cómo obtener varios mensajes en un tema en Apache kafka

¿Cómo puedo enviar mensajes grandes con Kafka (más de 15 MB)?

¿Cómo verificar si Kafka Servidor se está ejecutando?

¿Es posible agregar particiones a un tema existente en Kafka 0.8.2

¿Cómo puedo obtener el último desplazamiento de un tema kafka?

Cómo obtener una lista de temas del servidor kafka en Java

¿Cómo almacena Kafka las compensaciones para cada tema?

Error de tiempo de espera de operación en la consola cqlsh de cassandra

zookeeper no es una opción reconocida cuando se ejecuta kafka-console-consumer.sh

¿Cuándo el cliente Apache Kafka lanza una excepción "Batch Expired"?

kafka consigue el recuento de particiones para un tema

Cómo convertir un archivo csv a parquet

Error al crear Kafka tema: - factor de replicación mayor que los corredores disponibles

Kafka - No se puede enviar un mensaje a un servidor remoto usando Java

Kafka - Implementación de cola retrasada usando consumidor de alto nivel

kafka: Compensación de fallos con excepción retornable. Debe volver a intentar cometer compensaciones

¿Cómo ver el archivo Apache Parquet en Windows?

¿Por qué es más rápido procesar una matriz ordenada que una matriz sin clasificar?

Content dated before 2011-04-08 (UTC) is licensed under CC BY-SA 2.5. Content dated from 2011-04-08 up to but not including 2018-05-02 (UTC) is licensed under CC BY-SA 3.0. Content dated on or after 2018-05-02 (UTC) is licensed under CC BY-SA 4.0.

Privacy