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

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

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

import java.util.Map;

public class CountorInterceptor implements ProducerInterceptor<String, String> {

private int success;

private int error;

@Override

public ProducerRecord<String, String> onSend(ProducerRecord<String, String> record) {

return record;

}

@Override

public void onAcknowledgement(RecordMetadata metadata, Exception exception) {

if (metadata != null){

success++;

}else {

error++;

}

}

@Override

public void close() {

System.out.println("success:" + success + " error:" + error);

}

@Override

public void configure(Map<String, ?> configs) {

}

}

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

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

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

import java.util.Map;

public class TimestampInterceptor implements ProducerInterceptor<String, String> {

@Override

public ProducerRecord<String, String> onSend(ProducerRecord<String, String> record) {

String value = record.value();

String new\_value = System.currentTimeMillis() + ":" + value;

ProducerRecord<String, String> new\_record =

new ProducerRecord<String, String>(record.topic(),record.partition(),record.key(),new\_value);

return new\_record;

}

@Override

public void onAcknowledgement(RecordMetadata metadata, Exception exception) {

}

@Override

public void close() {

}

@Override

public void configure(Map<String, ?> configs) {

}

}

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

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Properties;

public class ProducerWithCallBack {

public static void main(String[] args) {

Properties props = new Properties();

//kafka的地址

props.put("bootstrap.servers", "hadoop102:9092,hadoop103:9092,hadoop104:9092");

//ack=-1

props.put("acks", "all");

//重试次数

props.put("retries", 2);

//基于大小的批处理

props.put(ProducerConfig.BATCH\_SIZE\_CONFIG, 16384);

//基于时间的批处理

props.put("linger.ms", 1);

//客户端的缓存大小

props.put("buffer.memory", 33554432);

//kv的序列化

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put(

ProducerConfig.INTERCEPTOR\_CLASSES\_CONFIG,

Arrays.asList("com.atguigu.kafka1111.TimestampInterceptor","com.atguigu.kafka1111.CountorInterceptor"));

//props.put(ProducerConfig.PARTITIONER\_CLASS\_CONFIG, "com.atguigu.kafka1111.CustomPartitioner");

KafkaProducer<String, String> producer = new KafkaProducer<>(props);

for (int i = 0; i< 10; i++){

ProducerRecord<String, String> record =

new ProducerRecord<>("first", "" + i,i + "test");

producer.send(record, new Callback() {

@Override

public void onCompletion(RecordMetadata metadata, Exception exception) {

if (metadata != null){

System.out.println("topic:" + metadata.topic() +

" partition:" + metadata.partition() +

" offset:" + metadata.offset());

}

}

});

}

producer.close();

}

}

import kafka.api.FetchRequest;

import kafka.api.FetchRequestBuilder;

import kafka.cluster.BrokerEndPoint;

import kafka.javaapi.\*;

import kafka.javaapi.consumer.SimpleConsumer;

import kafka.javaapi.message.ByteBufferMessageSet;

import kafka.message.Message;

import kafka.message.MessageAndOffset;

import java.nio.ByteBuffer;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class LowerConsumer {

public static void main(String[] args) {

//1.集群

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

list.add("hadoop102");

list.add("hadoop103");

list.add("hadoop104");

//2.主题

String topic = "first";

//3.分区

int partition = 0;

//4.offset

long offset = 30;

//5.获取leader

String leader = getLeader(list, topic, partition);

//6.连接leader，获取数据

getData(leader, topic, partition, offset);

}

private static void getData(String leader, String topic, int partition, long offset) {

//1.创建SimpleConsumer对象

SimpleConsumer consumer = new SimpleConsumer(leader, 9092,

2000,

1024 \* 1024 \* 2,

"getData");

//2.发送请求

//3.构建请求对象

FetchRequestBuilder builder = new FetchRequestBuilder();

FetchRequestBuilder requestBuilder = builder.addFetch(topic, partition, offset, 1024 \* 1024);

FetchRequest fetchRequest = requestBuilder.build();

//4.获取响应

FetchResponse fetchResponse = consumer.fetch(fetchRequest);

//5.解析响应

ByteBufferMessageSet messageAndOffsets = fetchResponse.messageSet(topic, partition);

//6.遍历

for (MessageAndOffset messageAndOffset : messageAndOffsets) {

long message\_offset = messageAndOffset.offset();

Message message = messageAndOffset.message();

//7.解析message

ByteBuffer byteBuffer = message.payload();

byte[] bytes = new byte[byteBuffer.limit()];

byteBuffer.get(bytes);

//8.获取数据

System.out.println("offset:" + message\_offset + " value:" + new String(bytes));

}

}

private static String getLeader(ArrayList<String> list, String topic, int partition) {

//1.循环发送请求，获取leader

for (String host : list) {

//2.创建SimpleConsumer对象

SimpleConsumer consumer = new SimpleConsumer(

host,

9092,

2000,

1024 \* 1024,

"getLeader");

//3.发送获取leader的请求

//4.构造请求

TopicMetadataRequest request = new TopicMetadataRequest(Arrays.asList(topic));

//5.获取响应

TopicMetadataResponse response = consumer.send(request);

//6.解析响应

List<TopicMetadata> topicsMetadata = response.topicsMetadata();

//7.遍历topicsMetadata

for (TopicMetadata topicMetadata : topicsMetadata) {

List<PartitionMetadata> partitionsMetadata = topicMetadata.partitionsMetadata();

//8.遍历partitionsMetadata

for (PartitionMetadata partitionMetadata : partitionsMetadata) {

//9.判断

if (partition == partitionMetadata.partitionId()){

BrokerEndPoint endPoint = partitionMetadata.leader();

return endPoint.host();

}

}

}

}

return null;

}

}

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

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

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

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

import java.util.Properties;

public class CustomProducer {

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

Properties props = new Properties();

//kafka的地址

props.put("bootstrap.servers", "hadoop102:9092,hadoop103:9092,hadoop104:9092");

//ack=-1

props.put("acks", "all");

//重试次数

props.put("retries", 2);

//基于大小的批处理

props.put(ProducerConfig.BATCH\_SIZE\_CONFIG, 16384);

//基于时间的批处理

props.put("linger.ms", 1);

//客户端的缓存大小

props.put("buffer.memory", 33554432);

//kv的序列化

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<String, String> producer = new KafkaProducer<>(props);

for (int i =0; i < 10;i++){

ProducerRecord<String, String> record =

new ProducerRecord<String, String>("first","" + i,i + "test");

producer.send(record);

}

//Thread.sleep(100);

producer.close();

}

}

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

import org.apache.kafka.common.Cluster;

import java.util.Map;

public class CustomPartitioner implements Partitioner {

@Override

public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster cluster) {

return 0;

}

@Override

public void close() {

}

/\*\*

\* 获取和添加属性到配置文件

\* @param configs

\*/

@Override

public void configure(Map<String, ?> configs) {

}

}