## 根据时间戳消费或指定时间内的数据

1. 使用场景：
2. 从当前时间往前推2个小时开始接着消费,消费完这2个小时断开
3. 从当前时间往前推2个小时开始接着持续消费
4. 指定个时间戳开始持续消费
5. 指定个开始时间与结束时间消费,消费完断开
6. 使用KafkaConsumer.offsetsForTimes要确认集群已开启log.message.timestamp.type参数，并且clien要使用0.10.\*的客户端发送数据，数据格式和0.9不同了

**consumer.subscribe(topicA);**

**consumer.poll(100);//正常订阅topic和poll消息**

**Set<TopicPartition> assignments = consumer.assignment();//获取consumer所分配的分区信息**

**Map<TopicPartition, Long> query = new HashMap<>();//构造offsetsForTimes参数，通过时间戳找到offset**

**for (TopicPartition topicPartition : assignments) {**

**System.out.println(topicPartition);**

**query.put(topicPartition, 1550804131000L);**

**}**

**Map<TopicPartition, OffsetAndTimestamp> result = consumer.offsetsForTimes(query);**

**for (Map.Entry<TopicPartition, OffsetAndTimestamp> entry : result.entrySet()) {**

**System.out.println(entry);**

**consumer.seek(entry.getKey(), entry.getValue().offset());//每个topic的partition都seek到执行的offset**

**}**

**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.OffsetAndTimestamp;  
**import** org.apache.kafka.common.PartitionInfo;  
**import** org.apache.kafka.common.TopicPartition;  
  
**import** java.text.DateFormat;  
**import** java.text.SimpleDateFormat;  
**import** java.util.\*;  
  
**public class** TimestampConsumer {  
   
 **public static void** main(String[] args) {  
   
 Properties props = **new** Properties();  
 props.put(**"bootstrap.servers"**, **"hadoop102:9092,hadoop103:9092,hadoop104:9092"**);  
 props.put(**"group.id"**, **"atguigu"**);  
 props.put(**"key.deserializer"**, **"org.apache.kafka.common.serialization.StringDeserializer"**);  
 props.put(**"value.deserializer"**, **"org.apache.kafka.common.serialization.StringDeserializer"**);  
 KafkaConsumer<String, String> consumer = **new** KafkaConsumer<>(props);  
 String topic = **"first"**;  
   
 **try** {  
 *// 获取topic的partition信息* List<PartitionInfo> partitionInfos = consumer.partitionsFor(topic);  
 List<TopicPartition> topicPartitions = **new** ArrayList<>();  
   
 Map<TopicPartition, Long> timestampsToSearch = **new** HashMap<>();  
 DateFormat df = **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**);  
 Date now = **new** Date();  
 **long** nowTime = now.getTime();  
 System.***out***.println(**"当前时间: "** + df.format(now));  
 **long** fetchDataTime = nowTime - 1000 \* 60 \* 60 \* 24 \* 120; *// 计算30分钟之前的时间戳* **for**(PartitionInfo partitionInfo : partitionInfos) {  
 topicPartitions.add(**new** TopicPartition(partitionInfo.topic(), partitionInfo.partition()));  
 timestampsToSearch.put(**new** TopicPartition(partitionInfo.topic(), partitionInfo.partition()), fetchDataTime);  
 }  
   
 consumer.assign(topicPartitions);  
   
 *// 获取每个partition一个小时之前的偏移量* Map<TopicPartition, OffsetAndTimestamp> map = consumer.offsetsForTimes(timestampsToSearch);  
   
 OffsetAndTimestamp offsetTimestamp = **null**;  
 System.***out***.println(**"开始设置各分区初始偏移量..."**);  
 **for**(Map.Entry<TopicPartition, OffsetAndTimestamp> entry : map.entrySet()) {  
 *// 如果设置的查询偏移量的时间点大于最大的索引记录时间，那么value就为空* offsetTimestamp = entry.getValue();  
 **if**(offsetTimestamp != **null**) {  
 **int** partition = entry.getKey().partition();  
 **long** timestamp = offsetTimestamp.timestamp();  
 **long** offset = offsetTimestamp.offset();  
 System.***out***.println(**"partition = "** + partition +   
 **", time = "** + df.format(**new** Date(timestamp))+   
 **", offset = "** + offset);  
 *// 设置读取消息的偏移量* consumer.seek(entry.getKey(), offset);  
 }  
 }  
 System.***out***.println(**"设置各分区初始偏移量结束..."**);  
   
 **while**(**true**) {  
 ConsumerRecords<String, String> records = consumer.poll(1000);  
 **for** (ConsumerRecord<String, String> record : records) {  
 System.***out***.println(**"partition = "** + record.partition() + **", offset = "** + record.offset());  
 }  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 consumer.close();  
 }  
 }  
}

package com.atguigu.kafka;

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.OffsetAndTimestamp;

import org.apache.kafka.common.PartitionInfo;

import org.apache.kafka.common.TopicPartition;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.\*;

public class TimestampConsumer {

public static void main(String[] args) {

Properties props = new Properties();

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

props.put("group.id", "first");

props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

String topic = "first";

try {

// 获取topic的partition信息

List<PartitionInfo> partitionInfos = consumer.partitionsFor(topic);

List<TopicPartition> topicPartitions = new ArrayList<>();

Map<TopicPartition, Long> timestampsToSearch = new HashMap<>();

DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

Date now = new Date();

long nowTime = now.getTime();

System.out.println("当前时间: " + df.format(now));

//long fetchDataTime = nowTime - 1000 \* 60 \* 60 \* 24 \* 120; // 计算30分钟之前的时间戳

long fetchDataTime = nowTime - 1000 \* 60 \* 60\*30; // 计算30分钟之前的时间戳

for(PartitionInfo partitionInfo : partitionInfos) {

topicPartitions.add(new TopicPartition(partitionInfo.topic(), partitionInfo.partition()));

timestampsToSearch.put(new TopicPartition(partitionInfo.topic(), partitionInfo.partition()), fetchDataTime);

}

consumer.assign(topicPartitions);

// 获取每个partition一个小时之前的偏移量

Map<TopicPartition, OffsetAndTimestamp> map = consumer.offsetsForTimes(timestampsToSearch);

OffsetAndTimestamp offsetTimestamp = null;

System.out.println("开始设置各分区初始偏移量...");

for(Map.Entry<TopicPartition, OffsetAndTimestamp> entry : map.entrySet()) {

// 如果设置的查询偏移量的时间点大于最大的索引记录时间，那么value就为空

offsetTimestamp = entry.getValue();

if(offsetTimestamp != null) {

int partition = entry.getKey().partition();

long timestamp = offsetTimestamp.timestamp();

long offset = offsetTimestamp.offset();

System.out.println("partition = " + partition +

", time = " + df.format(new Date(timestamp))+

", offset = " + offset);

// 设置读取消息的偏移量

consumer.seek(entry.getKey(), offset);

}

}

System.out.println("设置各分区初始偏移量结束...");

while(true) {

ConsumerRecords<String, String> records = consumer.poll(1000);

for (ConsumerRecord<String, String> record : records) {

System.out.println("partition = " + record.partition() + ", offset = " + record.offset());

}

}

} catch (Exception e) {

e.printStackTrace();

} finally {

consumer.close();

}

}

}