生产者优化3

消费和处理解耦
一个或多个消费者线程来做所有的数据消费，把ConsumerRecords实例存到一个被多个处理线程或线程池
消费的阻塞队列
　好处：不限制消费和处理的线程，让一个消费者来满足多个处理线程，避免了线程数被分区数所限制
　理解：(因为 不解耦的情况下，消费和处理在一起，offset提交的原因，消费线程被分区数限制，多的线程都是空转。而解耦了，处理线程完全不受限制，消费线程仍然限制)
　坏处：顺序是一个问题， 多个处理线程顺序无法保证，先从阻塞队列获得的数据 可能比后面获得的数据处理时间晚
　坏处： 手动提交offset变得很难，可能数据丢失和重复消费

```
package com.bupt.comsumer;
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 sun.applet.Main;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.Properties;
import java.util.concurrent.LinkedBlockingQueue;

public class MyTestThread2 {
    public static void main(String[] args) throws InterruptedException {
        KafkaConsumer<String, String> consumer;
        Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,"hadoop101:9092");
        //消费者组
        properties.put(ConsumerConfig.GROUP_ID_CONFIG,"test");
        //开启手动提交
        properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
        //自动提交的延时
        properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
        //key value的反序列化
        //properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,"earliest");

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,"org.apache.kafka.common.serialization.StringDeserializer");

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,"org.apache.kafka.common.serialization.StringDeserializer");
        consumer = new KafkaConsumer<String, String>(properties);
        consumer.subscribe(Arrays.asList("first"));
        ConsumerRecords<String, String> records ;



        LinkedBlockingQueue<ConsumerRecords<String, String>> list = new LinkedBlockingQueue();
        new Thread(new MyThread4(list),"bb").start();

        new Thread(new MyThread4(list),"aa").start();

        while (true){

            records = consumer.poll(1000);

            list.put(records);
             consumer.commitAsync();

        }



    }
}

class MyThread4 implements Runnable {

    LinkedBlockingQueue<ConsumerRecords<String, String>> list ;

    public MyThread4 (LinkedBlockingQueue<ConsumerRecords<String, String>> list)
{
        this.list = list;
    }

    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName()+"I come in!!");
        while (true) {
            ConsumerRecords<String, String> consumerRecords;
            try {
                consumerRecords = list.take();
                for (ConsumerRecord<String, String> consumerRecord :
consumerRecords) {
                    System.out.println(Thread.currentThread().getName()
                            +"消费了:" + consumerRecord.value()
                            +"  分区："+consumerRecord.partition()
                            +"偏移量是:" + consumerRecord.offset()
                    );
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }


        }
    }
}
```

利用producer生产了20条数据

测试结果

```
aaI come in!!
```

```
bbI come in!!
aa消费了:hello yk 0  分区：1偏移量是:242
aa消费了:hello yk 2  分区：1偏移量是:243
aa消费了:hello yk 4  分区：1偏移量是:244
aa消费了:hello yk 6  分区：1偏移量是:245
aa消费了:hello yk 8  分区：1偏移量是:246
aa消费了:hello yk 10  分区：1偏移量是:247
aa消费了:hello yk 12  分区：1偏移量是:248
aa消费了:hello yk 14  分区：1偏移量是:249
aa消费了:hello yk 16  分区：1偏移量是:250
aa消费了:hello yk 18  分区：1偏移量是:251
bb消费了:hello yk 1  分区：0偏移量是:268
bb消费了:hello yk 3  分区：0偏移量是:269
bb消费了:hello yk 5  分区：0偏移量是:270
bb消费了:hello yk 7  分区：0偏移量是:271
bb消费了:hello yk 9  分区：0偏移量是:272
bb消费了:hello yk 11  分区：0偏移量是:273
bb消费了:hello yk 13  分区：0偏移量是:274
bb消费了:hello yk 15  分区：0偏移量是:275
bb消费了:hello yk 17  分区：0偏移量是:276
bb消费了:hello yk 19  分区：0偏移量是:277
```