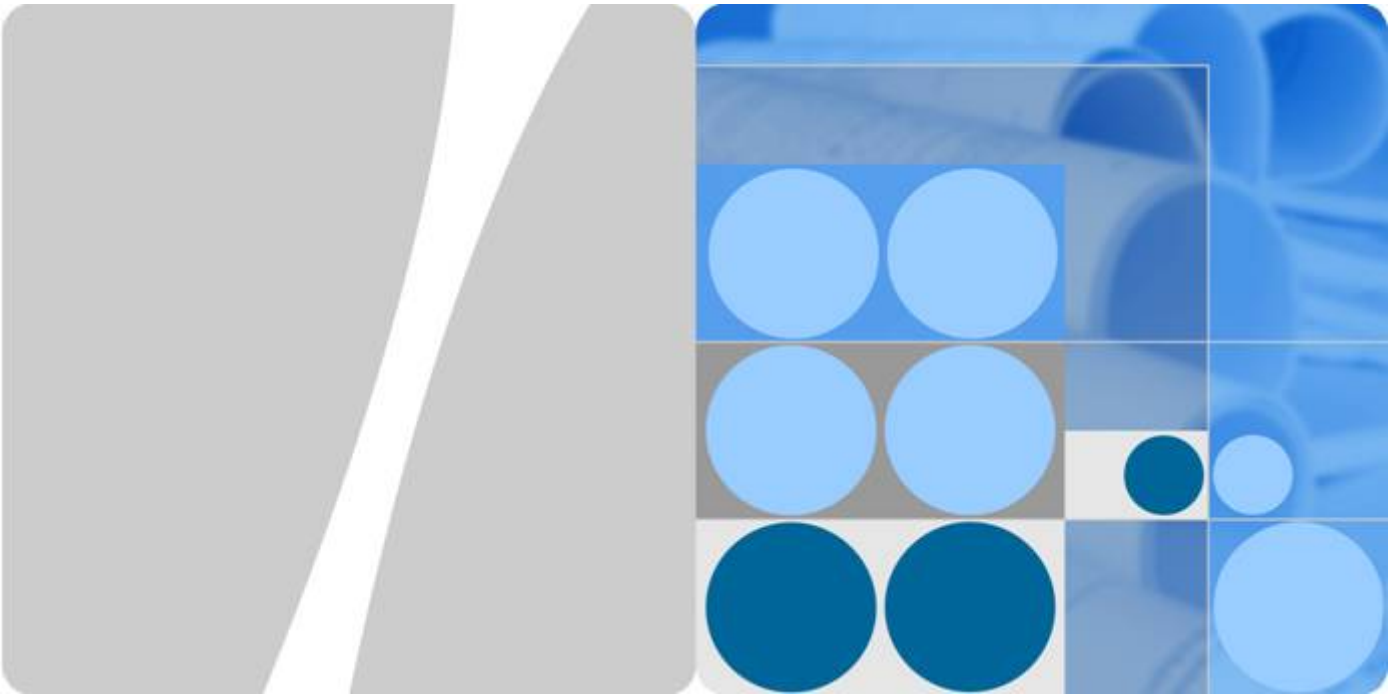


分布式消息服务

最佳实践

文档版本　01

发布日期　2019-01-04

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

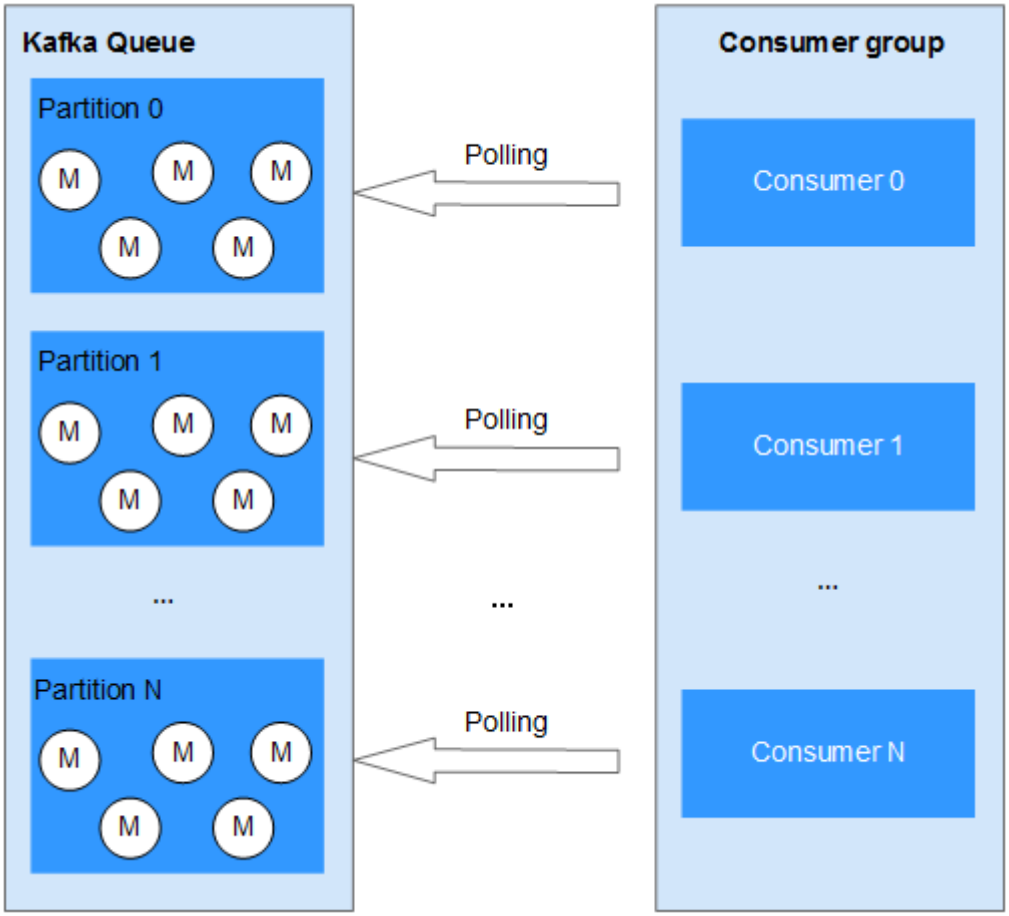
1 DMS Kafka 消费者 poll 的优化

场景介绍

在DMS提供的原生Kafka SDK中，消费者可以自定义拉取消息的时长，如果需要长时间的拉取消息，只需要把poll(long)方法的参数设置合适的值即可。但是这样的长连接可能会对客户端和服务端造成一定的压力，特别是分区数较多且每个消费者开启多个线程的情况下。

如图1-1所示，Kafka队列含有多个分区，消费组中有多个消费者同时进行消费，每个线程均为长连接。当队列中消息较少或者没有时，连接不断开，所有消费者不间断地拉取消息，这样造成了一定的资源浪费。

图 1-1 Kafka 消费者多线程消费模式

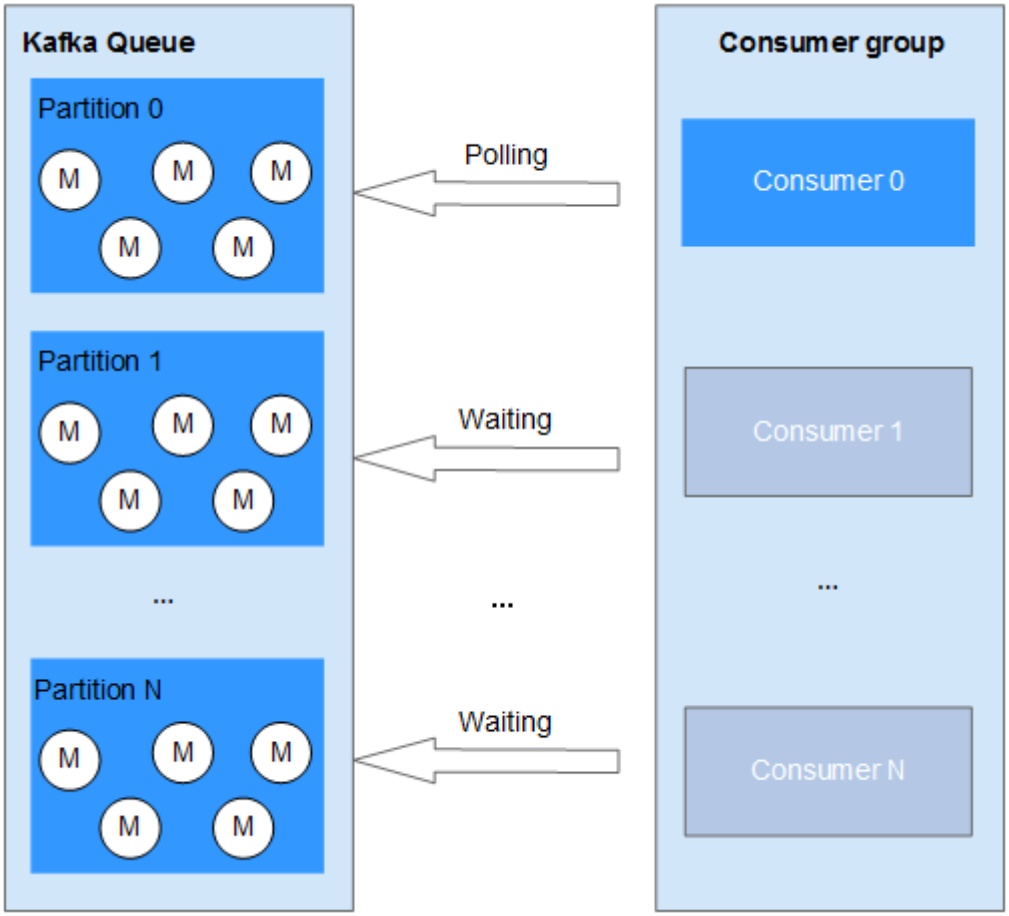


优化方案

在开了多个线程同时访问的情况下，如果队列里已经没有消息了，其实不需要所有的线程都在poll，只需要有一个线程poll各分区的消息就足够了，当在polling的线程发现队列中有消息，可以唤醒其他线程一起消费消息，以达到快速响应的目的。如图1-2所示。

这种方案适用于对消费消息的实时性要求不高的应用场景。如果要求准实时消费消息，则建议保持所有消费者处于活跃状态。

图 1-2 优化后的多线程消费方案



说明

消费者（Consumer）和消息分区（Partition）并不强制数量相等，Kafka的poll(long)方法帮助实现获取消息、分区平衡、消费者与Kafka broker节点间的心跳检测等功能。

因此在对消费消息的实时性要求不高场景下，当消息数量不多的时候，可以选择让一部分消费者处于wait状态。

代码示例

须知

以下仅贴出与消费者线程唤醒与睡眠相关代码，如需运行整个demo，请先下载完整的**代码示例包**，同时参考**DMS开发指南**进行部署和运行。

消费消息代码示例如下：

```
package com.huawei.dms.kafka;

import java.io.IOException;
import java.util.Arrays;
import java.util.Collection;
import java.util.Iterator;
import java.util.Properties;
```



```
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
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.TopicPartition;
import org.apache.log4j.Logger;

public class DmsKafkaConsumeDemo
{
    private static Logger logger = Logger.getLogger(DmsKafkaProduceDemo.class);

    public static void WorkerFunc(int workerId, KafkaConsumer<String, String> kafkaConsumer) throws
IOException
    {
        Properties consumerConfig = Config.getConsumerConfig();
        RecordReceiver receiver = new RecordReceiver(workerId, kafkaConsumer,
consumerConfig.getProperty("topic"));
        while (true)
        {
            ConsumerRecords<String, String> records = receiver.receiveMessage();
            Iterator<ConsumerRecord<String, String>> iter = records.iterator();
            while (iter.hasNext())
            {
                ConsumerRecord<String, String> cr = iter.next();
                System.out.println("Thread" + workerId + " recievedrecords" + cr.value());
                logger.info("Thread" + workerId + " recievedrecords" + cr.value());

            }

        }
    }

    public static KafkaConsumer<String, String> getConsumer() throws IOException
    {
        Properties consumerConfig = Config.getConsumerConfig();

        consumerConfig.put("ssl.truststore.location", Config.getTrustStorePath());
        System.setProperty("java.security.auth.login.config", Config.getSaslConfig());

        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(consumerConfig);
        kafkaConsumer.subscribe(Arrays.asList(consumerConfig.getProperty("topic")),
                new ConsumerRebalanceListener()
                {
                    @Override
                    public void onPartitionsRevoked(Collection<TopicPartition> arg0)
                    {

                    }

                    @Override
                    public void onPartitionsAssigned(Collection<TopicPartition> tps)
                    {

                    }
                });
        return kafkaConsumer;
    }

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

        //创建当前消费组的consumer
        final KafkaConsumer<String, String> consumer1 = getConsumer();
        Thread thread1 = new Thread(new Runnable()
        {
            public void run()
            {
                try
                {
```

```
            WorkerFunc(1, consumer1);
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
});
final KafkaConsumer<String, String> consumer2 = getConsumer();

Thread thread2 = new Thread(new Runnable()
{
    public void run()
    {
        try
        {
            WorkerFunc(2, consumer2);
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
});
final KafkaConsumer<String, String> consumer3 = getConsumer();

Thread thread3 = new Thread(new Runnable()
{
    public void run()
    {
        try
        {
            WorkerFunc(3, consumer3);
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
});

//启动线程
thread1.start();
thread2.start();
thread3.start();

try
{
    Thread.sleep(5000);
}
catch (InterruptedException e)
{
    e.printStackTrace();
}
//线程加入
try
{
    thread1.join();
    thread2.join();
    thread3.join();
}
catch (InterruptedException e)
{
    e.printStackTrace();
}
```



```
    }
}
```

消费者线程管理示例代码

示例仅提供简单的设计思路，开发者可结合实际场景优化线程休眠和唤醒机制。

```
package com.huawei.dms.kafka;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.log4j.Logger;

public class RecordReceiver
{
    private static Logger logger = Logger.getLogger(DmsKafkaProduceDemo.class);

    //polling的间隔时间
    public static final int WAIT_SECONDS = 10 * 1000;

    protected static final Map<String, Object> sLockObjMap = new HashMap<String, Object>();

    protected static Map<String, Boolean> sPollingMap = new ConcurrentHashMap<String, Boolean>();

    protected Object lockObj;

    protected String topicName;

    protected KafkaConsumer<String, String> kafkaConsumer;

    protected int workerId;

    public RecordReceiver(int id, KafkaConsumer<String, String> kafkaConsumer, String queue)
    {
        this.kafkaConsumer = kafkaConsumer;
        this.topicName = queue;
        this.workerId = id;

        synchronized (sLockObjMap)
        {
            lockObj = sLockObjMap.get(topicName);
            if (lockObj == null)
            {
                lockObj = new Object();
                sLockObjMap.put(topicName, lockObj);
            }
        }
    }

    public boolean setPolling()
    {
        synchronized (lockObj)
        {
            Boolean ret = sPollingMap.get(topicName);
            if (ret == null || !ret)
            {
                sPollingMap.put(topicName, true);
                return true;
            }
            return false;
        }
    }

    //唤醒全部线程
    public void clearPolling()
```

```
{
    synchronized (lockObj)
    {
        sPollingMap.put(topicName, false);
        lockObj.notifyAll();
        System.out.println("Everyone WakeUp and Work!");
        logger.info("Everyone WakeUp and Work!");
    }
}

public ConsumerRecords<String, String> receiveMessage()
{
    boolean polling = false;
    while (true)
    {
        //检查线程的poll状态，必要时休眠
        synchronized (lockObj)
        {
            Boolean p = sPollingMap.get(topicName);
            if (p != null && p)
            {
                try
                {
                    System.out.println("Thread" + workerId + " Have a nice sleep!");
                    logger.info("Thread" + workerId +" Have a nice sleep!");
                    polling = false;
                    lockObj.wait();
                }
                catch (InterruptedException e)
                {
                    System.out.println("MessageReceiver Interrupted! topicName is " + topicName);
                    logger.error("MessageReceiver Interrupted! topicName is "+topicName);

                    return null;
                }
            }
        }

        //开始消费，必要时唤醒其他线程消费
        try
        {
            ConsumerRecords<String, String> Records = null;
            if (!polling)
            {
                Records = kafkaConsumer.poll(100);
                if (Records.count() == 0)
                {
                    polling = true;
                    continue;
                }
            }
            else
            {
                if (setPolling())
                {
                    System.out.println("Thread" + workerId + " Polling!");
                    logger.info("Thread " + workerId + " Polling!");
                }
                else
                {
                    continue;
                }
                do
                {
                    System.out.println("Thread" + workerId + " KEEP Poll records!");
                    logger.info("Thread" + workerId + " KEEP Poll records!");
                    try
                    {
                        Records = kafkaConsumer.poll(WAIT_SECONDS);
```

```
                    }
                    catch (Exception e)
                    {
                        System.out.println("Exception Happened when polling records: " + e);
                        logger.error("Exception Happened when polling records: " + e);

                    }
                } while (Records.count()==0);
                clearPolling();
            }
            //消息确认
            kafkaConsumer.commitSync();
            return Records;
        }
        catch (Exception e)
        {
            System.out.println("Exception Happened when poll records: " + e);
            logger.error("Exception Happened when poll records: " + e);
        }
    }
  }
}
```

说明

topicName配置为“队列名称”或者“Kafka Topic”。

示例代码运行结果

```
[2018-01-25 22:40:51,841] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:40:51,841] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:40:52,122] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:40:52,169] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,169] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,216] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,325] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:40:52,325] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:40:54,947] INFO Thread1 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:40:54,979] INFO Thread3 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:32,347] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:42,353] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:47,816] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:47,847] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:47,925] INFO Thread 3 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:47,925] INFO Thread1 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:47,925] INFO Thread3 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:47,957] INFO Thread2 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:48,472] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:48,503] INFO Thread3 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,518] INFO Thread1 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
```



```
[2018-01-25 22:41:48,550] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,597] INFO Thread1 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,659] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:48,659] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:48,675] INFO Thread3 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,675] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:48,706] INFO Thread 1 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:48,706] INFO Thread1 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
```

2 利用消息标签实现消息过滤

场景介绍

在实际场景中，一个队列存储的消息，可以包含不同实际用途。如果这些消息不加区分，消费者每次消费都会按顺序拉取消息，直到完成对所有消息的消费。

如果消费者只对某一类型的消息感兴趣，那么将所有消息都消费一遍势必影响消费者处理效率。

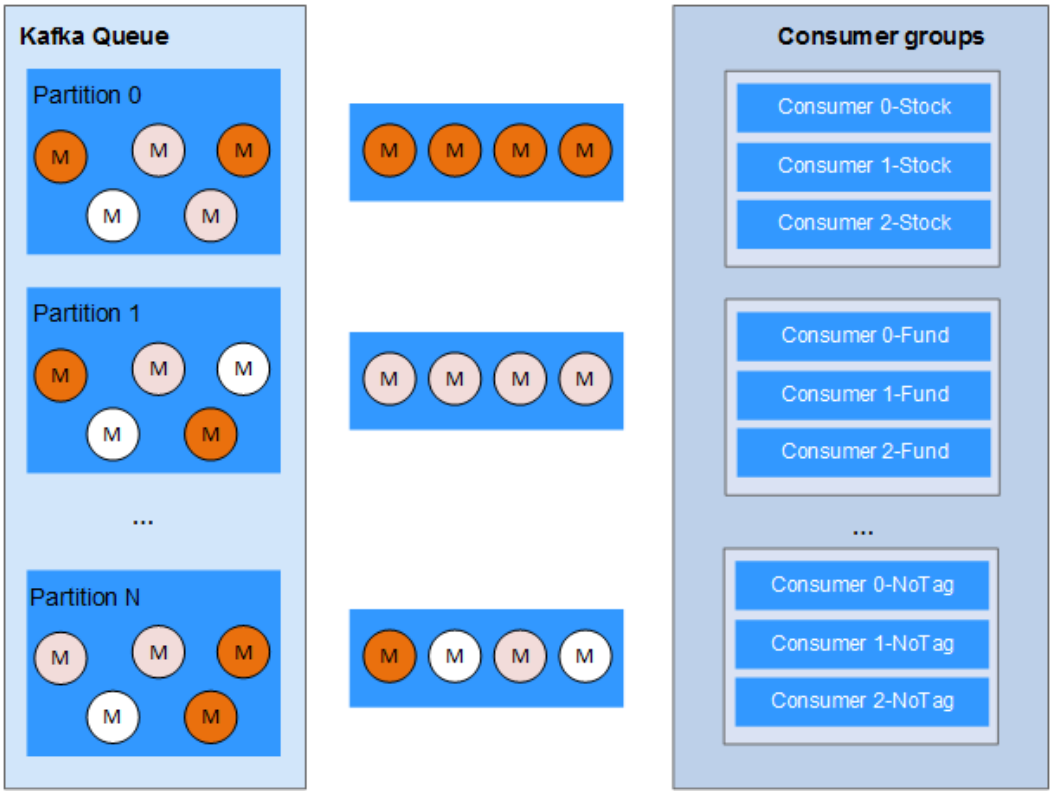
优化方案

DMS服务提供消息标签的能力，支持生产者为每条消息提供一个或多个标签（tag），消费者则根据标签（tag）的内容来过滤消息，确保每个消费者最终只会消费到它感兴趣的消息类型。

以金融交易场景为例，在一种交易中可能会产生多钟类型的消息，如股票（stock），基金（fund），贷款（loan）等。这些消息会通过交易（business）主题来传递给不同的处理系统，如股票系统，基金系统，贷款系统，实时分析系统等。然而基金系统只关心基金类型的消息，而实时分析系统可能需要获取到所有类型的消息。

在生产消息时，生产者对每条消息加上标签（tag），消费者在拉取消息时决定是否仅获取带有某标签（tag）的消息，从而提高消息消费效率。如**图2-1**所示：

图 2-1 增加标签（tag）的消息消费示意图



须知

DMS普通队列与FIFO队列支持消息标签（Tag）功能，Kafka队列不支持。

代码示例

须知

以下仅贴出与消息标签相关代码，如需运行整个demo，请先下载完整的**代码示例包**，同时参考**DMS开发指南**进行部署和运行。

示例提供了基于Http Restful接口的代码，具体API接口可参考帮助中心**分布式消息服务接口参考**。

消息标签设计示例代码

```
package com.cloud.dms;

import java.net.URL;
import java.util.Properties;
import com.cloud.dms.access.AccessServiceUtils;

public class DMSHttpClient
{
    private static String endpointUrl = "";

    private static String region = "";
```

```
private static String serviceName = "dms";

private static String aKey = "";

private static String sKey = "";

private static String projectId = "546e52331ea74cd49722fda4fb23bf55";

private static String queueId = "39cd8dcb-b901-43b4-9ea1-48730e9adc58";

private static String queueGroupId = "g-ae8ed05f-464c-452c-9e37-d3bdd081000d";

/*
 * Read Configure File And Initialize Variables
 */
static
{
    URL configPath = ClassLoader.getSystemResource("dms-service-config.properties");
    Properties prop = AccessServiceUtils.getPropsFromFile(configPath.getFile());
    region = prop.getProperty(Constants.DMS_SERVICE_REGION);
    aKey = prop.getProperty(Constants.DMS_SERVICE_AK);
    sKey = prop.getProperty(Constants.DMS_SERVICE_SK);
    endpointUrl = prop.getProperty(Constants.DMS_SERVICE_ENDPOINT_URL);
    if (endpointUrl.endsWith("/"))
    {
        endpointUrl = endpointUrl + "v1.0/";
    }
    else
    {
        endpointUrl = endpointUrl + "/v1.0/";
    }
    projectId = prop.getProperty(Constants.DMS_SERVICE_PROJECT_ID);
}

public static void main(String[] args)
{
    runAllApiMethods();
}

public static void runAllApiMethods()
{
    MsgAttri msg = new MsgAttri();
    msg.setaKey(aKey);
    msg.setEndpointUrl(endpointUrl);
    msg.setProjectId(projectId);
    msg.setQueueId(queueId);
    msg.setsKey(sKey);
    msg.setRegion(region);
    msg.setServiceName(serviceName);
    msg.setMsgLimit("10");
    msg.setGroupId(queueGroupId);
    /**
     * 构造生产者和四种消费者，设置各自感兴趣的tag
     */
    MsgProducer msgProducer = new MsgProducer(msg);
    MsgConsumer stock = new MsgConsumer(msg, "stock");
    MsgConsumer fund = new MsgConsumer(msg, "fund");
    MsgConsumer loan = new MsgConsumer(msg, "loan");
    MsgConsumer all = new MsgConsumer(msg, null);
    /**
     * 创建线程，模拟生产和消费行为，设置线程的名字，便于区分
     */
    Thread producer = new Thread(msgProducer);
    Thread stockThread = new Thread(stock);
    Thread fundThread = new Thread(fund);
    Thread loanThread = new Thread(loan);
    Thread alls = new Thread(all);
    producer.setName("producer");
```



```
        stockThread.setName("stock");
        fundThread.setName("fund");
        loanThread.setName("loan");
        alls.setName("Analysis");
        /**
         * 启动线程
         */
        producer.start();
        stockThread.start();
        fundThread.start();
//        loanThread.start();
//        alls.start();
    }
}
```

生产消息示例

```
package com.cloud.dms;

import static com.cloud.dms.ApiUtils.constructTempMessages;
import static com.cloud.dms.ApiUtils.sendMessages;

import java.util.concurrent.TimeUnit;

public class MsgProducer implements Runnable{
    private MsgAttri msgAttri;

    public MsgProducer(MsgAttri msg) {
        this.msgAttri = msg;
    }

    public void run() {
        while (true)
        {
            /**
            * 模拟生产者，构造消息，JSON中包含stock， fund， loan三种消息
            **/
            String messages = constructTempMessages(null);
            sendMessages(messages, this.msgAttri);
            try
            {
                TimeUnit.SECONDS.sleep(1);
            }
            catch (InterruptedException e) {
            }
        }
    }
}
```

消费消息示例代码

```
package com.cloud.dms;

import static com.cloud.dms.ApiUtils.acknowledgeMessages;
import static com.cloud.dms.ApiUtils.consumeMessages;
import static com.cloud.dms.ApiUtils.parseHandlerIds;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class MsgConsumer implements Runnable{

    private MsgAttri msgAttri;

    private String tag;
```



```
/**
*构造函数，获取到消费者感兴趣的tag
**/
public MsgConsumer(MsgAttri msg, String tag) {
  this.tag = tag;
  this.msgAttri = msg;
}

public void run() {
  while (true)
  {
    /**
    *消费消息，获取该tag下的消息
    **/
    ResponseMessage consumeMessagesResMsg = consumeMessages(msgAttri, tag);
    /**
    *解析消息
    **/
    if (consumeMessagesResMsg.getStatusCode() == 200)
    {
      List<String> msgStrings = ApiUtils.decodeMsg(consumeMessagesResMsg);
      /**
      *模拟处理消息，打印该tag下的消息
      **/
      for (String s : msgStrings)
      {
        System.out.println("Thread--"+ Thread.currentThread().getName() + "--Message Body is: "+ s);
      }
      /**
      * 消费确认
      **/
      ArrayList<String> handlerIds = parseHandlerIds(consumeMessagesResMsg);
      if (handlerIds.size() > 0)
      {
        acknowledgeMessages(handlerIds, msgAttri);
      }
    }
    else
    {
      System.out.println("Http Response Code is: "
          + consumeMessagesResMsg.getStatusCode() + "\n Http Body is: "
          + consumeMessagesResMsg.getBody());

    }
    try
    {
      TimeUnit.SECONDS.sleep(2);
    }
    catch (InterruptedException e)
    {
    }
  }
 }
}
```

示例运行结果如下

Loan线程由于指定了tag（loan），因此只能消费到loan标签的消息：

```
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607269605","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607270813","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607272069","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607277819","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607280210","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607281437","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607282671","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607311902","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607314401","tags":["loan"]}
Thread--loan--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607315635","tags":["loan"]}

{"success":10,"fail":0}
```

Fund和stock线程也同理只能消费到各自指定的消息：

```
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607341145","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607342397","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607343916","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607345143","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607346370","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607435714","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607437984","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607439235","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607440469","tags":["stock"]}
Thread--stock--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607451488","tags":["stock"]}
{"success":10,"fail":0}
```

而Analysi线程没有指定Tag，可以消费到Topic里的所有消息：

```
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607021008","tags":["loan"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607153683","tags":["stock"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607153683","tags":["fund"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607153683","tags":["loan"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607156342","tags":["stock"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607156342","tags":["fund"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607156342","tags":["loan"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607157801","tags":["stock"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607157801","tags":["fund"]}
Thread--Analysis--Message Body is: {"attributes":{"name":"ignore","type":"string.start","value":"test"},"body":"msg_body_1513607157801","tags":["loan"]}
{"success":10,"fail":0}
```

3 如何提高消息处理效率

消息发送和消费的可靠性必须由DMS服务和生产者以及消费者协同工作才能保证。同时开发者需要尽量合理使用DMS消息队列，以提高消息发送和消息消费的效率与准确性。

对使用DMS服务的生产者和消费者有如下的使用建议：

重视消息生产与消费的确认过程

消息生产（发送）

发送消息后，生产者需要根据DMS的返回信息确认消息是否发送成功，如果返回失败需要重新发送。

每次生产消息，生产者都需要等待消息发送API的应答信号，以确认消息是否成功发送。在消息传递过程中，如果发生异常，生产者没有接收到发送成功的信号，生产者自己决策是否需要重复发送消息。如果接收到发送成功的信号，则表明该消息已经被DMS可靠存储。

消息消费

消息消费时，消费者需要确认消息是否已被成功消费。

生产的消息被依次存储在DMS的存储介质中。消费时依次获取DMS中存储的消息。消费者获取消息后，进行消费并记录消费成功或失败的状态，并将消费状态提交到DMS，由DMS决定消费下一批消息或回滚重新消费消息。

在消费过程中，如果出现异常，没有提交消费确认，该批消息会在后续的消费请求中再次被获取。

消息生产与消费的幂等传递

DMS设计了一系列可靠性保障措施，确保消息不丢失。例如使用消息同步存储机制防止系统与服务器层面的异常重启或者掉电，使用消息确认（ACK）机制解决消息传输过程中遇到的异常。

考虑到网络异常等极端情况，用户除了做好消息生产与消费的确认，还需要配合DMS完成消息发送与消费的重复传输设计。

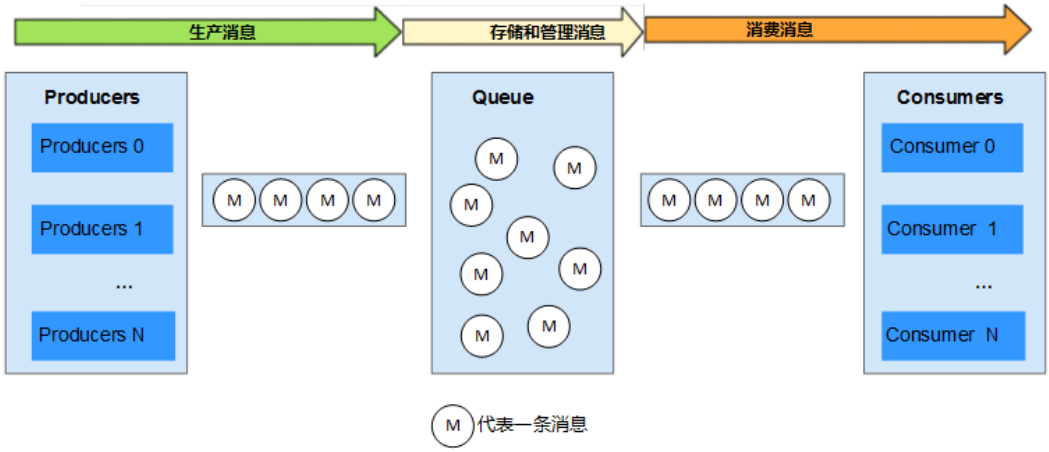
- 当无法确认消息是否已发送成功，生产者需要将消息重复发送给DMS。
- 当重复收到已处理过的消息，消费者需要告诉DMS消费成功且保证不重复处理。

消息可以批量生产和消费

为提高消息发送和消息消费效率，推荐使用批量消息发送和消费。通常，默认消息消费为批量消费，而消息发送尽可能采用批量发送。同时批量方式可有效减少API调用次数，减少服务使用费用。

如下面两张示意图对比所示，消息批量生产与消费，可以减少API调用次数，节约资源。

图 3-1 消息批量生产（发送）与消费

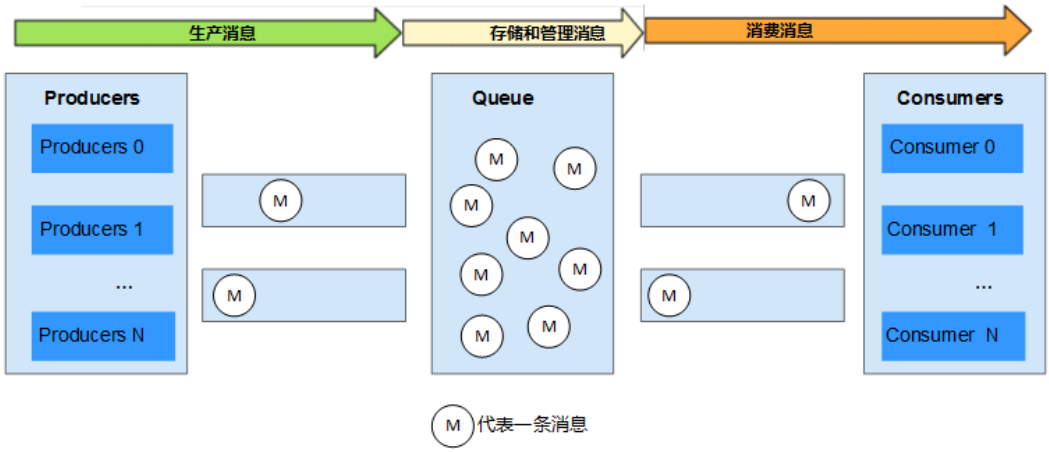


须知

批量发送消息时，单次不能超过10条消息，总大小不能超过512KB。

批量生产（发送）消息可以灵活使用，在消息并发多的时候，批量发送，并发少时，单条发送。这样能够在减少调用次数的同时保证消息发送的实时性。

图 3-2 消息逐条生产（发送）与消费



此外，批量消费消息时，消费者应按照接收的顺序对消息进行处理、确认，当对某一条消息处理失败时，不再需要继续处理本批消息中的后续消息，直接对已正确处理的消息进行确认即可。

巧用消费组协助运维

用户使用DMS服务作为消息管理系统，查看队列的消息内容对于定位问题与调试服务是至关重要的。

当消息的生产和消费过程中遇到疑难问题时，通过创建不同消费组可以帮助定位分析问题或调试服务对接。用户可以创建一个新的消费组，对队列中的消息进行消费并分析消费过程，这样不会影响其他服务对消息的处理。

4 Kafka 客户端参数配置建议

Kafka客户端的配置参数很多，以下提供Producer和Consumer几个常用参数配置。

表 4-1 Producer 参数

参数	默认值	推荐值	说明
acks	1	高可靠：all 高吞吐：1	收到Server端确认信号个数，表示procuder需要收到多少个这样的确认信号，算消息发送成功。acks参数代表了数据备份的可用性。常用选项： acks=**0**：表示producer不需要等待任何确认收到的信息，副本将立即加到socket buffer并认为已经发送。没有任何保障可以保证此种情况下server已经成功接收数据，同时重试配置不会发生作用（因为客户端不知道是否失败）回馈的offset会总是设置为-1。 acks=**1**：这意味着至少要等待leader已经成功将数据写入本地log，但是并没有等待所有follower是否成功写入。如果follower没有成功备份数据，而此时leader又无法提供服务，则消息会丢失。 acks=**all**：这意味着leader需要等待所有备份都成功写入日志，只有任何一个备份存活，数据都不会丢失。
retries	0	结合实际业务调整	客户端发送消息的重试次数。值大于0时，这些数据发送失败后，客户端会重新发送。 注意，这些重试与客户端接收到发送错误时的重试没有什么不同。允许重试将潜在的改变数据的顺序，如果这两个消息记录都是发送到同一个partition，则第一个消息失败第二个发送成功，则第二条消息会比第一条消息出现要早。
request.timeout.ms	30000	结合实际业务调整	设置一个请求最大等待时间，超过这个时间则会抛Timeout异常。 超时时间如果设置大一些，如120000（120秒），高并发的场景中，能减少发送失败的情况。

参数	默认值	推荐值	说明
block.on.buffer.full	TRUE	TRUE	TRUE表示当我们内存用尽时，停止接收新消息记录或者抛出错误。 默认情况下，这个设置为TRUE。然而某些阻塞可能不值得期待，因此立即抛出错误更好。如果设置为false，则producer抛出一个异常错误：BufferExhaustedException
batch.size	16384	262144	默认的批量处理消息字节数上限。producer将试图批处理消息记录，以减少请求次数。这将改善client与server之间的性能。不会试图处理大于这个字节数的消息字节数。 发送到brokers的请求将包含多个批量处理，其中会包含对每个partition的一个请求。 较小的批量处理数值比较少用，并且可能降低吞吐量（0则会仅用批量处理）。较大的批量处理数值将会浪费更多内存空间，这样就需要分配特定批量处理数值的内存大小。
buffer.memory	33554432	67108864	producer可以用来缓存数据的内存大小。如果数据产生速度大于向broker发送的速度，producer会阻塞或者抛出异常，以“block.on.buffer.full”来表明。 这项设置将和producer能够使用的总内存相关，但并不是一个硬性的限制，因为不是producer使用的所有内存都是用于缓存。一些额外的内存会用于压缩（如果引入压缩机制），同样还有一些用于维护请求。

表 4-2 Consumer 参数

参数	默认值	推荐值	说明
auto.commit.enable	TRUE	FALSE	如果为真，consumer所fetch的消息的offset将会自动的同步到zookeeper。这项提交的offset将在进程无法提供服务时，由新的consumer使用。 约束： 设置为false后，需要先成功消费再提交，这样可以避免消息丢失。
auto.offset.reset	latest	earliest	没有初始化offset或者offset被删除时，可以设置以下值： earliest：自动复位offset为最早 latest：自动复位offset为最新 none：如果没有发现offset则向消费者抛出异常 anything else：向消费者抛出异常。

参数	默认值	推荐值	说明
connections.max.idle.ms	600000	30000	空连接的超时时间，设置为30000可以在网络异常场景下减少请求卡顿的时间。

5 Kafka 客户端使用规范

consumer 使用规范

1. consumer的owner线程需确保不会异常退出，避免客户端无法发起消费请求，阻塞消费。
2. 确保处理完消息后再做消息commit，避免业务消息处理失败，无法重新拉取处理失败的消息。
3. consumer不能频繁加入和退出group，频繁加入和退出，会导致consumer频繁做rebalance，阻塞消费。
4. consumer数量不能超过topic分区数，否则会有consumer拉取不到消息。
5. consumer需周期poll，维持和server的心跳，避免心跳超时，导致consumer频繁加入和退出，阻塞消费。
6. consumer拉取的消息本地缓存应有大小限制，避免OOM（Out of Memory）。
7. consumer session设置为30秒，session.timeout.ms=30000。
8. Kafka不能保证消费重复的消息，业务侧需保证消息处理的幂等性。
9. 消费线程退出要调用consumer的close方法，避免同一个组的其他消费者阻塞sesstion.timeout.ms的时间。

producer 使用规范

1. 同步复制客户端需要配合使用：ack=all
2. 配置发送失败重试：retries=3
3. 发送优化：linger.ms=0
4. 生产端的JVM内存要足够，避免内存不足导致发送阻塞

topic 使用规范

配置要求：推荐3副本，同步复制，最小同步副本数为2，且同步副本数不能等于topic副本数，否则宕机1个副本会导致无法生产消息。

创建方式：支持选择是否开启kafka自动创建Topic的开关。选择开启后，表示生产或消费一个未创建的Topic时，会自动创建一个包含3个分区和3个副本的Topic。

单topic最大分区数建议为20。

topic副本数为3（当前版本限制，不可调整）。

其他建议

连接数限制：3000

消息大小：不能超过10MB

使用sasl_ssl协议访问Kafka：确保DNS具有反向解析能力，或者在hosts文件配置kafka所有节点ip和主机名映射，避免Kafka client做反向解析，阻塞连接建立。

磁盘容量申请超过业务量 * 副本数的2倍，即保留磁盘空闲50%左右。

业务进程JVM内存使用确保无频繁FGC，否则会阻塞消息的生产和消费。