ALIBABA CLOUD

阿里云

消息队列 MQ

高级特性

文档版本：20210224

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.消息重试

本文介绍

消息队列Rocket MQ版

的消息重试机制和配置方式。

顺序消息的重试

对于顺序消息，当消费者消费消息失败后，

消息队列Rocket MQ版

会自动不断地进行消息重试（每次间隔时间为1秒），这时，应用会出现消息消费被阻塞的情况。因此，建议您使用顺序消息时，务必保证应用能够及时监控并处理消费失败的情况，避免阻塞现象的发生。

无序消息的重试

对于无序消息（普通、定时、延时、事务消息），当消费者消费消息失败时，您可以通过设置返回状态达到消息重试的结果。

无序消息的重试只针对集群消费方式生效；广播方式不提供失败重试特性，即消费失败后，失败消息不再重试，继续消费新的消息。

> **注意** 以下内容都只针对无序消息生效。

重试次数

消息队列Rocket MQ版

默认允许每条消息最多重试16次，每次重试的间隔时间如下。

第几次重试	与上次重试的间隔时间	第几次重试	与上次重试的间隔时间
1	10秒	9	7分钟
2	30秒	10	8分钟
3	1分钟	11	9分钟
4	2分钟	12	10分钟
5	3分钟	13	20分钟
6	4分钟	14	30分钟
7	5分钟	15	1小时
8	6分钟	16	2小时

如果消息重试16次后仍然失败，消息将不再投递。如果严格按照上述重试时间间隔计算，某条消息在一直消费失败的前提下，将会在接下来的4小时46分钟之内进行16次重试，超过这个时间范围消息将不再重试投递。

> **注意** 一条消息无论重试多少次，这些重试消息的Message ID不会改变。

配置方式

- 消费失败后，重试配置方式

  集群消费方式下，消息消费失败后期望消息重试，需要在消息监听器接口的实现中明确进行配置（三种方式任选一种）：

  - 方式1：返回Action.ReconsumeLater（推荐）
  - 方式2：返回Null
  - 方式3：抛出异常

  示例代码

```
public class MessageListenerImpl implements MessageListener {
  @Override
  public Action consume(Message message, ConsumeContext context) {
    //消息处理逻辑抛出异常，消息将重试。
    doConsumeMessage(message);
    //方式1：返回Action.ReconsumeLater，消息将重试。
    return Action.ReconsumeLater;
    //方式2：返回null，消息将重试。
    return null;
    //方式3：直接抛出异常，消息将重试。
    throw new RuntimeException("Consumer Message exception");
  }
}
```

- 消费失败后，无需重试的配置方式

  集群消费方式下，消息失败后期望消息不重试，需要捕获消费逻辑中可能抛出的异常，最终返回Action.CommitMessage，此后这条消息将不会再重试。

  示例代码

```
public class MessageListenerImpl implements MessageListener {
  @Override
  public Action consume(Message message, ConsumeContext context) {
    try {
      doConsumeMessage(message);
    } catch (Throwable e) {
      //捕获消费逻辑中的所有异常，并返回Action.CommitMessage;
      return Action.CommitMessage;
    }
    //消息处理正常，直接返回Action.CommitMessage;
    return Action.CommitMessage;
  }
}
```

- 自定义消息最大重试次数

  > **说明** 自定义
  >
  > 消息队列RocketMQ版
  >
  > 的客户端日志配置，请升级TCP Java SDK版本到1.2.2及以上。

  消息队列RocketMQ版

  允许Consumer启动的时候设置最大重试次数，重试时间间隔将按照以下策略：

  - 最大重试次数小于等于16次，则重试时间间隔同上表描述。
  - 最大重试次数大于16次，超过16次的重试时间间隔均为每次2小时。

  配置方式如下：

  ```
  Properties properties = new Properties();
  //配置对应Group ID的最大消息重试次数为20次，最大重试次数为字符串类型。
  properties.put(PropertyKeyConst.MaxReconsumeTimes,"20");
  Consumer consumer =ONSFactory.createConsumer(properties);
  ```

  > **注意**
  >
  > - 消息最大重试次数的设置对相同Group ID下的所有Consumer实例有效。
  > - 如果只对相同Group ID下两个Consumer实例中的其中一个设置了MaxReconsumeTimes，那么该配置对两个Consumer实例均生效。
  > - 配置采用覆盖的方式生效，即最后启动的Consumer实例会覆盖之前的启动实例的配置。

获取消息重试次数

消费者收到消息后，可按照以下方式获取消息的重试次数：

```
public class MessageListenerImpl implements MessageListener {
  @Override
  public Action consume(Message message, ConsumeContext context) {
    //获取消息的重试次数。
    System.out.println(message.getReconsumeTimes());
    return Action.CommitMessage;
  }
}
```

2.消息过滤

本文描述

消息队列RocketMQ版

的消费者如何根据Tag在

消息队列RocketMQ版

服务端完成消息过滤，以确保消费者最终只消费到其关注的消息类型。

Tag，即消息标签，用于对某个Topic下的消息进行分类。

消息队列RocketMQ版

的生产者在发送消息时，已经指定消息的Tag，消费者需根据已经指定的Tag来进行订阅。

场景示例

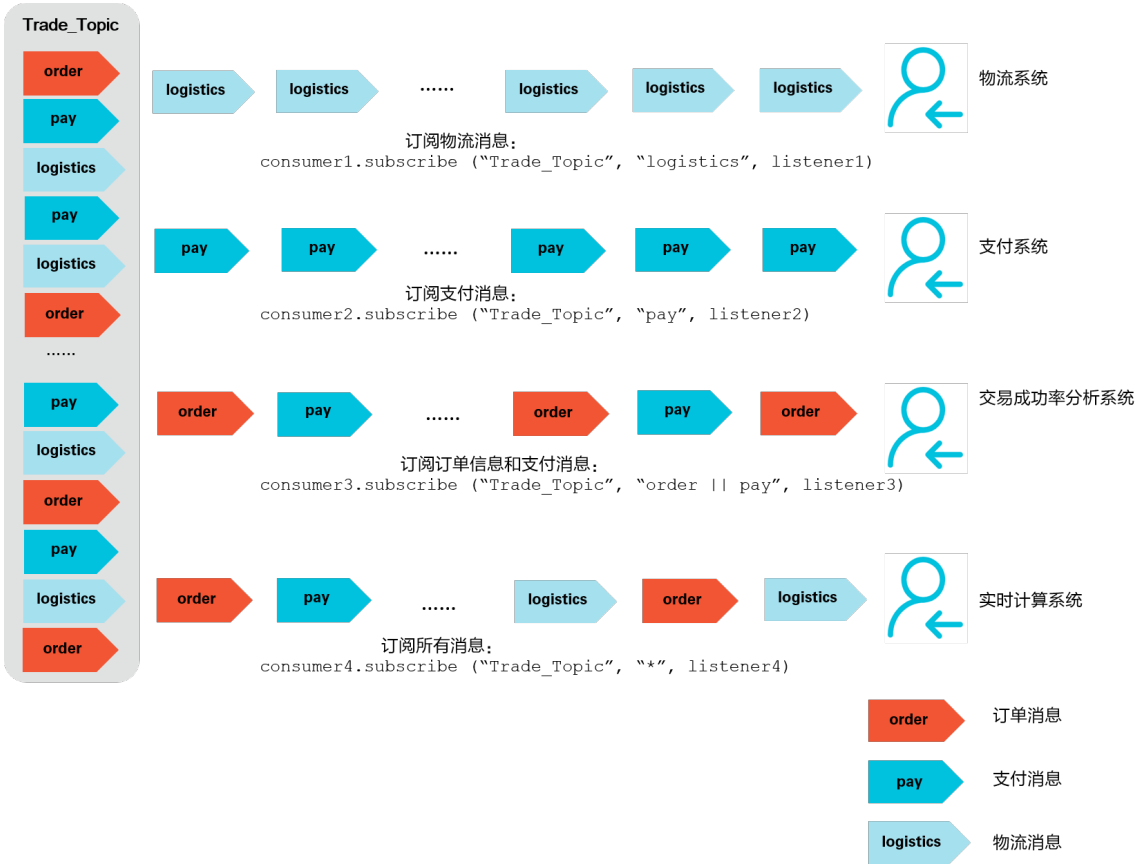
以下图电商交易场景为例，从客户下单到收到商品这一过程会生产一系列消息，以以下消息为例：

- 订单消息
- 支付消息
- 物流消息

这些消息会发送到**Trade_Topic**Topic中，被各个不同的系统所订阅，以以下系统为例：

- 支付系统：只需订阅支付消息。
- 物流系统：只需订阅物流消息。
- 交易成功率分析系统：需订阅订单和支付消息。
- 实时计算系统：需要订阅所有和交易相关的消息。

过滤示意图如下所示。



示例代码

- 发送消息

  发送消息时，每条消息必须指明Tag。

```
Message msg = new Message("MQ_TOPIC","TagA","Hello MQ".getBytes());
```

- 订阅所有Tag

  消费者如需订阅某Topic下所有类型的消息，Tag用星号（*）表示。

```
consumer.subscribe("MQ_TOPIC", "*", new MessageListener() {
  public Action consume(Message message, ConsumeContext context) {
    System.out.println(message.getMsgID());
    return Action.CommitMessage;
  }
});
```

- 订阅单个Tag

  消费者如需订阅某Topic下某一种类型的消息，请明确标明Tag。

```
consumer.subscribe("MQ_TOPIC", "TagA", new MessageListener() {
  public Action consume(Message message, ConsumeContext context) {
    System.out.println(message.getMsgID());
    return Action.CommitMessage;
  }
});
```

- 订阅多个Tag

  消费者如需订阅某Topic下多种类型的消息，请在多个Tag之间用两个竖线（||）分隔。

```
consumer.subscribe("MQ_TOPIC", "TagA||TagB", new MessageListener() {
  public Action consume(Message message, ConsumeContext context) {
    System.out.println(message.getMsgID());
    return Action.CommitMessage;
  }
});
```

- 错误示例

  同一个消费者多次订阅某个Topic下的Tag，以最后一次订阅的Tag为准。

```
//如下错误代码中，Consumer只能订阅到MQ_TOPIC下TagB的消息，而不能订阅TagA的消息。
consumer.subscribe("MQ_TOPIC", "TagA", new MessageListener() {
  public Action consume(Message message, ConsumeContext context) {
    System.out.println(message.getMsgID());
    return Action.CommitMessage;
  }
});
consumer.subscribe("MQ_TOPIC", "TagB", new MessageListener() {
  public Action consume(Message message, ConsumeContext context) {
    System.out.println(message.getMsgID());
    return Action.CommitMessage;
  }
});
```

更多信息

- 同一个Group ID下的消费者实例与Topic的订阅关系需保持一致，更多信息，请参见订阅关系一致。
- 合理使用Topic和Tag来过滤消息可以让业务更清晰，更多信息，请参见Topic与Tag最佳实践。

3.Exactly-Once投递语义

本文主要介绍

消息队列RocketMQ版

的Exactly-Once投递语义的概念和典型使用场景，以便您理解如何使得消息只被消费端处理且仅处理一次。

什么是Exactly-Once投递语义

Exactly-Once是指发送到消息系统的消息只能被消费端处理且仅处理一次，即使生产端重试消息发送导致某消息重复投递，该消息在消费端也只被消费一次。

Exactly-Once语义是消息系统和流式计算系统中消息流转的最理想状态，但是在业界并没有太多理想的实现。因为真正意义上的Exactly-Once依赖消息系统的服务端、消息系统的客户端和用户消费逻辑这三者状态的协调。例如，当您的消费端完成一条消息的消费处理后出现异常宕机，而消费端重启后由于消费的位点没有同步到消息系统的服务端，该消息有可能被重复消费。

业界对于Exactly-Once投递语义存在很大的争议，很多人会拿出“FLP不可能理论”或者其他一致性定律对此议题进行否定，但事实上，特定场景的Exactly-Once语义实现并不是非常复杂，只是因为通常大家没有精确的描述问题的本质。

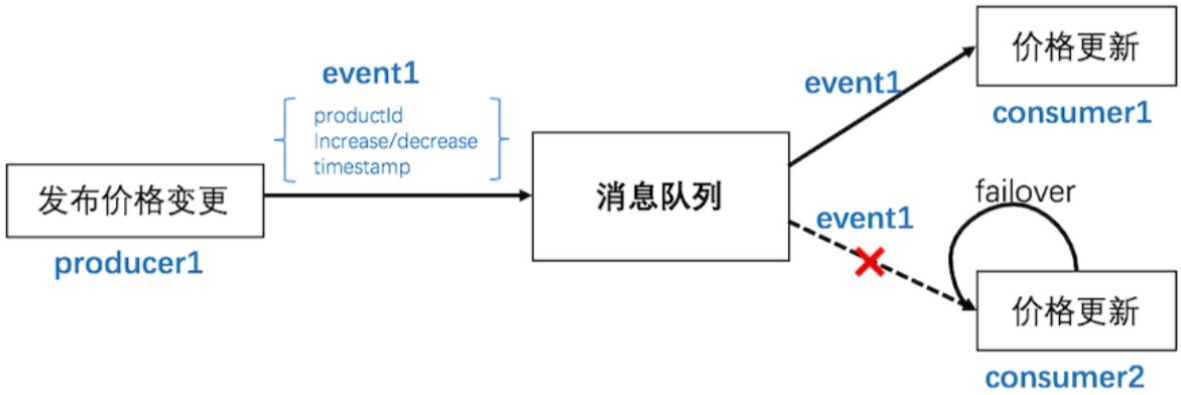
如果您要实现一条消息的消费结果只能在业务系统中生效一次，您需要解决的只是如何保证同一条消息的消费幂等问题。

消息队列RocketMQ版

的Exactly-Once语义就是解决业务中最常见的一条消息的消费结果（消息在消费端计算处理的结果）在数据库系统中有且仅生效一次的问题。

典型使用场景

在电商系统中，上游实时计算模块发布商品价格变更的信息，异步通知到下游商品管理模块进行价格变更。此时，需要保证每一条信息的消费幂等，即重复的价格变更信息只会生效一次，这样便不会发生价格多次重复修改的情况，确保实现了消息消费的幂等。



更多信息

如何使用Exactly-Once投递语义的具体步骤，请参见使用Exactly-Once投递语义收发消息。

4.集群消费和广播消费

本文介绍

消息队列Rocket MQ版

的集群消费和广播消费的基本概念、适用场景以及注意事项。

消息队列Rocket MQ版

是基于发布或订阅模型的消息系统。消费者，即消息的订阅方订阅关注的Topic，以获取并消费消息。由于消费者应用一般是分布式系统，以集群方式部署，因此

消息队列Rocket MQ版

约定以下概念：

- 集群：使用相同Group ID的消费者属于同一个集群。同一个集群下的消费者消费逻辑必须完全一致（包括Tag的使用）。更多信息，请参见订阅关系一致。
- 集群消费：当使用集群消费模式时，

  消息队列Rocket MQ版

  认为任意一条消息只需要被集群内的任意一个消费者处理即可。
- 广播消费：当使用广播消费模式时，

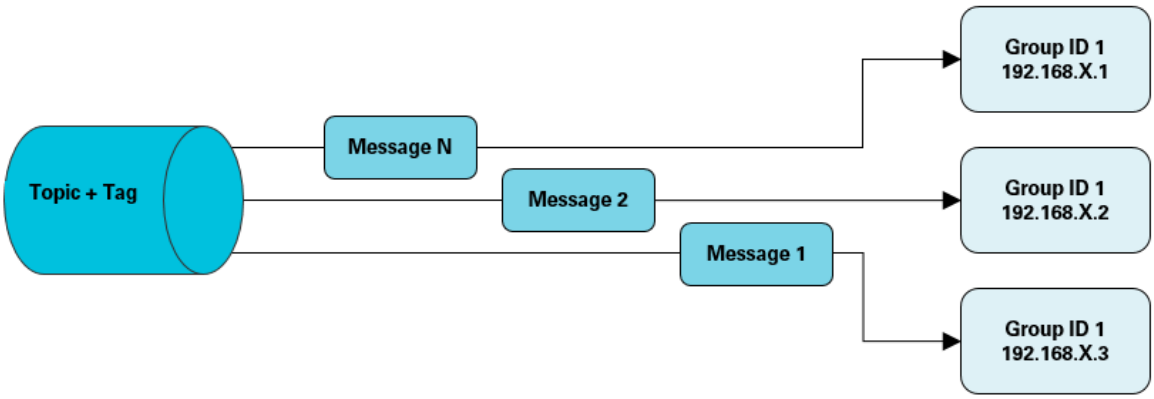
  消息队列Rocket MQ版

  会将每条消息推送给集群内所有注册过的消费者，保证消息至少被每个消费者消费一次。

集群消费模式

- 适用场景

  适用于消费端集群化部署，每条消息只需要被处理一次的场景。此外，由于消费进度在服务端维护，可靠性更高。具体消费示例如下图所示。

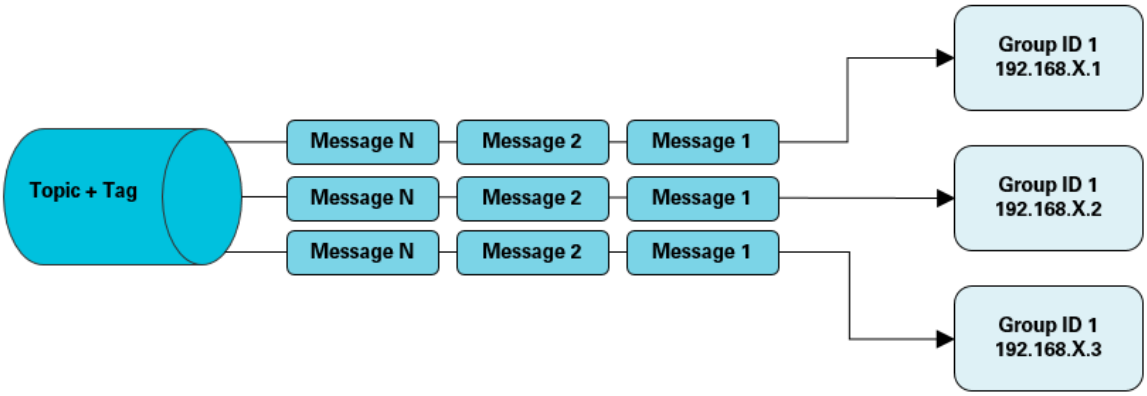


- 注意事项
  - 集群消费模式下，每一条消息都只会被分发到一台机器上处理。如果需要被集群下的每一台机器都处理，请使用广播模式。
  - 集群消费模式下，不保证每一次失败重投的消息路由到同一台机器上。

广播消费模式

- 适用场景

适用于消费端集群化部署，每条消息需要被集群下的每个消费者处理的场景。具体消费示例如下图所示。



- 注意事项
  - 广播消费模式下不支持顺序消息。
  - 广播消费模式下不支持重置消费位点。
  - 每条消息都需要被相同订阅逻辑的多台机器处理。
  - 消费进度在客户端维护，出现重复消费的概率稍大于集群模式。
  - 广播模式下，

    消息队列RocketMQ版

    保证每条消息至少被每台客户端消费一次，但是并不会重投消费失败的消息，因此业务方需要关注消费失败的情况。
  - 广播模式下，客户端每一次重启都会从最新消息消费。客户端在被停止期间发送至服务端的消息将会被自动跳过，请谨慎选择。
  - 广播模式下，每条消息都会被大量的客户端重复处理，因此推荐尽可能使用集群模式。
  - 广播模式下服务端不维护消费进度，所以

    消息队列RocketMQ版

    控制台不支持消息堆积查询、消息堆积报警和订阅关系查询功能。

更多信息

集群消费模式和广播消费模式的具体配置方法，请参见以下文档：

- TCP协议
  - Java：订阅消息
  - .NET：订阅消息
  - C/C++：订阅消息
- HTTP协议

  HTTP协议目前仅支持集群消费，详情请参见HTTP SDK版本说明。

5.批量消费

如需提高消息的处理效率，或降低下游资源的API调用频率，您可使用批量消费功能。本文介绍批量消费的定义、优势与场景、使用限制和示例代码等信息。

什么是批量消费

- 定义

  批量消费是

  消息队列RocketMQ版

  通过Push消费者提供的、将消息分批次消费的功能。

  > **说明** 根据消息获取方式，
  >
  > 消息队列RocketMQ版
  >
  > 提供Push和Pull两种类型的消费者，更多信息，请参见名词解释。

- 功能原理

  批量消费主要分为以下两个阶段：

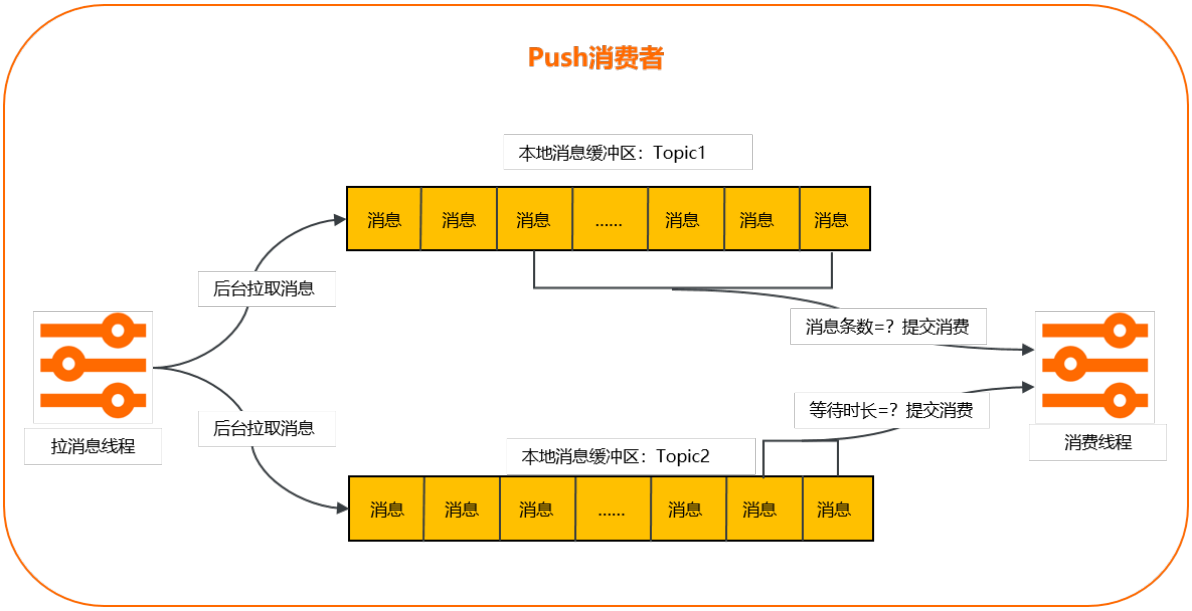
  i. 消息从生产者发布至

  消息队列RocketMQ版

  后，Push消费者中的拉消息线程通过长轮询将消息拉到后台缓存。

  ii. Push消费者根据缓存情况是否满足任一批量条件，判断是否将消息提交给消费线程完成消费。

  具体示意图如下所示。



使用限制

- 请确保您使用的SDK是商业版TCP Java SDK，且版本在1.8.7.3.Final或以上，详细的版本说明和获取方式，请参见商业版TCP Java SDK版本说明。

- 支持一次提交最多1024条消息，支持攒批等待最多450秒。

功能优势及场景示例

批量消费的功能优势和场景示例说明如下：

- 优势一：提高消息的吞吐能力和处理效率

  场景示例：上游订单系统和下游Elasticsearch系统间通过

  消息队列RocketMQ版

  解耦，Elasticsearch消费订单系统的10条日志消息，每一条消息对于Elasticsearch系统而言都是一次RPC请求，假设一次RPC请求耗时10毫秒，那么不使用批量消费的耗时为10×10=100毫秒；理想状态下，使用批量消费的耗时可缩短至10毫秒，因为10条消息合并为一次消费，大大提高消息的处理效率。

- 优势二：降低下游资源的API调用频率

  场景示例：给数据库中插入数据，每更新一条数据执行一次插入任务，如果数据更新较频繁，可能会对数据库造成较大压力。此时，您可以设置每10条数据批量插入一次或每5秒执行一次插入任务，降低系统运行压力。

示例代码

批量消费的示例代码如下所示。

```
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.batch.BatchConsumer;
import com.aliyun.openservices.ons.api.batch.BatchMessageListener;
import java.util.List;
import java.util.Properties;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.tcp.example.MqConfig;
public class SimpleBatchConsumer {
  public static void main(String[] args) {
    Properties consumerProperties = new Properties();
    consumerProperties.setProperty(PropertyKeyConst.GROUP_ID, MqConfig.GROUP_ID);
    consumerProperties.setProperty(PropertyKeyConst.AccessKey, MqConfig.ACCESS_KEY);
    consumerProperties.setProperty(PropertyKeyConst.SecretKey, MqConfig.SECRET_KEY);
    consumerProperties.setProperty(PropertyKeyConst.NAMESRV_ADDR, MqConfig.NAMESRV_ADDR);
    // 设置批量消费最大消息数量，当指定Topic的消息数量已经攒够128条，SDK立即执行回调进行消费。默认值：32，取值范围：1~1024。
    consumerProperties.setProperty(PropertyKeyConst.ConsumeMessageBatchMaxSize, String.valueOf(128));
    // 设置批量消费最大等待时长，当等待时间达到10秒，SDK立即执行回调进行消费。默认值：0，取值范围：0~450，单位：秒。
    consumerProperties.setProperty(PropertyKeyConst.BatchConsumeMaxAwaitDurationInSeconds, Strin
```

```
g.valueOf(10));
    BatchConsumer batchConsumer = ONSFactory.createBatchConsumer(consumerProperties);
    batchConsumer.subscribe(MqConfig.TOPIC, MqConfig.TAG, new BatchMessageListener() {
      @Override
      public Action consume(final List<Message> messages, ConsumeContext context) {
        System.out.printf("Batch-size: %d\n", messages.size());
        // 批量消息处理。
        return Action.CommitMessage;
      }
    });
    //启动batchConsumer。
    batchConsumer.start();
    System.out.println("Consumer start success.");
    //等待固定时间防止进程退出。
    try {
      Thread.sleep(200000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

参数描述如下表所示。

参数名	参数类型	是否必选	描述
ConsumeMessageBatchMaxSize	String	否 **说明** 如未指定参数值，则使用默认值。	批量消费的最大消息数量，缓存的消息数量达到参数设置的值，Push消费者SDK会将缓存的消息统一提交给消费线程，实现批量消费。取值范围：[1, 1024]，默认值：32，单位：条。
BatchConsumeMaxAwaitDurationInSeconds	String		批量消费的最大等待时长，等待时长达到参数设置的值，会将缓存的消息统一推送给消费者进行批量消费。取值范围：[1, 1024]，默认值：0，单位：秒。

说明

- 具体的示例代码，请以消息队列RocketMQ版代码库为准。
- 更多参数信息，请参见接口和参数说明。

最佳实践

请合理设置ConsumeMessageBatchMaxSize和BatchConsumeMaxAwaitDurationInSeconds参数的取值，只要达到任一参数设置的批量条件，即会触发提交批量消费。例如ConsumeMessageBatchMaxSize设置为128，BatchConsumeMaxAwaitDurationInSeconds设置为1，1秒内虽然没有积攒到128条消息，仍然会触发批量消费，此时返回的Batch-size会小于128。

此外，为了获得更好的批量消费效果，强烈推荐您实现消息幂等，保证消息有且仅被处理1次。幂等处理的具体信息，请参见消费幂等。

更多信息

商业版TCP Java SDK订阅消息