ALIBABA CLOUD

Alibaba Cloud

消息队列 MQ

最佳实践

文档版本：20220511

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.Topic与Tag最佳实践

在
消息队列RocketMQ版
中，Topic与Tag都是业务上用来归类的标识，区分在于Topic是一级分类，而Tag可以理解为是二级分类。
您可通过本文了解如何搭配使用Topic和Tag来实现消息过滤。

背景信息

Topic和Tag的定义如下：

Topic
消息主题，通过Topic对不同的业务消息进行分类。

Tag
消息标签，用来进一步区分某个Topic下的消息分类，消息从生产者发出即带上的属性。

Topic和Tag的关系如下图所示。

适用场景

您可能会有这样的疑问：到底什么时候该用Topic，什么时候该用Tag?

建议您从以下几个方面进行判断：

- 消息类型是否一致：如普通消息、事务消息、定时（延时）消息、顺序消息，不同的消息类型使用不同的Topic，无法通过Tag进行区分。
- 业务是否相关联：没有直接关联的消息，如淘宝交易消息，京东物流消息使用不同的Topic进行区分；而同样是天猫交易消息，电器类订单、女装类订单、化妆品类订单的消息可以用Tag进行区分。
- 消息优先级是否一致：如同样是物流消息，盒马必须小时内送达，天猫超市24小时内送达，淘宝物流则相对会慢一些，不同优先级的消息用不同的Topic进行区分。
- 消息量级是否相当：有些业务消息虽然量小但是实时性要求高，如果跟某些万亿量级的消息使用同一个Topic，则有可能会因为过长的等待时间而“饿死”，此时需要将不同量级的消息进行拆分，使用不同的Topic。

总的来说，针对消息分类，您可以选择创建多个Topic，或者在同一个Topic下创建多个Tag。但通常情况下，不同的Topic之间的消息没有必然的联系，而Tag则用来区分同一个Topic下相互关联的消息，例如全集和子集的关系、流程先后的关系。

场景示例

以天猫交易平台为例，订单消息和支付消息属于不同业务类型的消息，分别创建**Topic_Order**和**Topic_Pay**，其中订单消息根据商品品类以不同的Tag再进行细分，例如电器类、男装类、女装类、化妆品类等被各个不同的系统所接收。

通过合理的使用Topic和Tag，可以让业务结构清晰，更可以提高效率。

如何通过Tag实现消息过滤，请参见消息过滤。

更多信息

订阅关系一致。

2.消费幂等

为了防止消息重复消费导致业务处理异常，
消息队列RocketMQ版
的消费者在接收到消息后，有必要根据业务上的唯一Key对消息做幂等处理。本文介绍消息幂等的概念、适用场景以及处理方法。

什么是消息幂等

当出现消费者对某条消息重复消费的情况时，重复消费的结果与消费一次的结果是相同的，并且多次消费并未对业务系统产生任何负面影响，那么这个消费者的处理过程就是幂等的。

例如，在支付场景下，消费者消费扣款消息，对一笔订单执行扣款操作，扣款金额为100美元。如果因网络不稳定等原因导致扣款消息重复投递，消费者重复消费了该扣款消息，但最终的业务结果是只扣款一次，扣费100美元，且用户的扣款记录中对应的订单只有一条扣款流水，不会多次扣除费用。那么这次扣款操作是符合要求的，整个消费过程实现了消费幂等。

适用场景

在互联网应用中，尤其在网络不稳定的情况下，

消息队列RocketMQ版
的消息有可能会出现重复。如果消息重复会影响您的业务处理，请对消息做幂等处理。
消息重复的场景如下：

- 发送时消息重复
  当一条消息已被成功发送到服务端并完成持久化，此时出现了网络闪断或者客户端宕机，导致服务端对客户端应答失败。 如果此时生产者意识到消息发送失败并尝试再次发送消息，消费者后续会收到两条内容相同但Message ID不同的消息。
- 投递时消息重复
  消息消费的场景下，消息已投递到消费者并完成业务处理，当客户端给服务端反馈应答的时候网络闪断。为了保证消息至少被消费一次，
  消息队列RocketMQ版
  的服务端将在网络恢复后再次尝试投递之前已被处理过的消息，消费者后续会收到两条内容相同并且Message ID也相同的消息。
- 负载均衡时消息重复（包括但不限于网络抖动、Broker重启以及消费者应用重启）
  当
  消息队列RocketMQ版
  的Broker或客户端重启、扩容或缩容时，会触发Rebalance，此时消费者可能会收到少量重复消息。

处理方法

因为不同的Message ID对应的消息内容可能相同，有可能出现冲突（重复）的情况，所以真正安全的幂等处理，不建议以Message ID作为处理依据。最好的方式是以业务唯一标识作为幂等处理的关键依据，而业务的唯一标识可以通过消息Key设置。

以支付场景为例，可以将消息的Key设置为订单号，作为幂等处理的依据。具体代码示例如下：

```
Message message = new Message();
message.setKey("ORDERID_100");
SendResult sendResult = producer.send(message);
```

消费者收到消息时可以根据消息的Key，即订单号来实现消息幂等：

```
consumer.subscribe("ons_test", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        String key = message.getKey()
        // 根据业务唯一标识的Key做幂等处理。
    }
});
```

3.订阅关系一致

订阅关系一致指的是同一个消费者Group ID下所有Consumer实例所订阅的Topic、Tag必须完全一致。如果订阅关系不一致，消息消费的逻辑就会混乱，甚至导致消息丢失。本文提供订阅关系一致的正确示例代码以及订阅关系不一致的可能原因，帮助您顺畅地订阅消息。

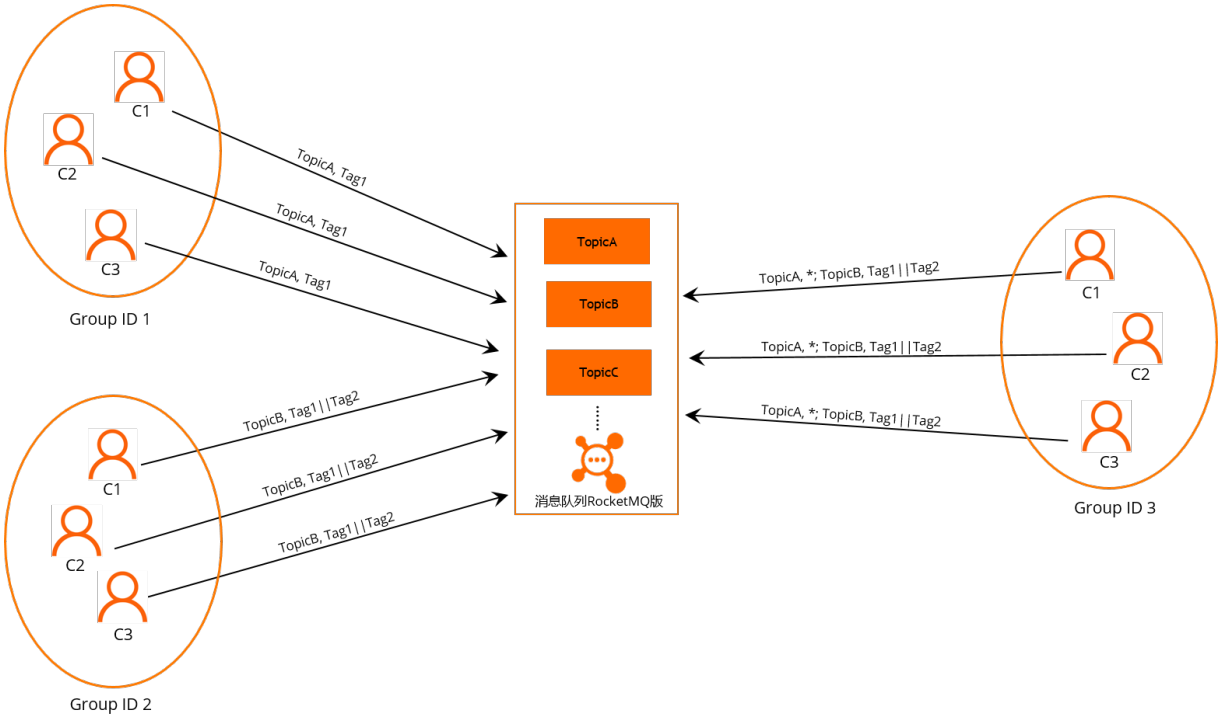
背景信息

消息队列RocketMQ版
里的一个消费者Group ID代表一个Consumer实例群组。对于大多数分布式应用来说，一个消费者Group ID下通常会挂载多个Consumer实例。
由于

消息队列RocketMQ版
的订阅关系主要由Topic+Tag共同组成，因此，保持订阅关系一致意味着同一个消费者Group ID下所有的Consumer实例需在以下方面均保持一致：

- 订阅的Topic必须一致，例如：Consumer1订阅TopicA和TopicB，Consumer2也必须订阅TopicA和TopicB，不能只订阅TopicA、只订阅TopicB或订阅TopicA和TopicC。
- 订阅的同一个Topic中的Tag必须一致，包括Tag的数量和Tag的顺序，例如：Consumer1订阅TopicB且Tag为Tag1||Tag2，Consumer2订阅TopicB的Tag也必须是Tag1||Tag2，不能只订阅Tag1、只订阅Tag2或者订阅Tag2||Tag1。

正确的订阅关系如下，多个Group ID分别订阅了不同的Topic，但是同一个Group ID下的多个Consumer实例C1、C2、C3订阅的Topic和Tag都一致。

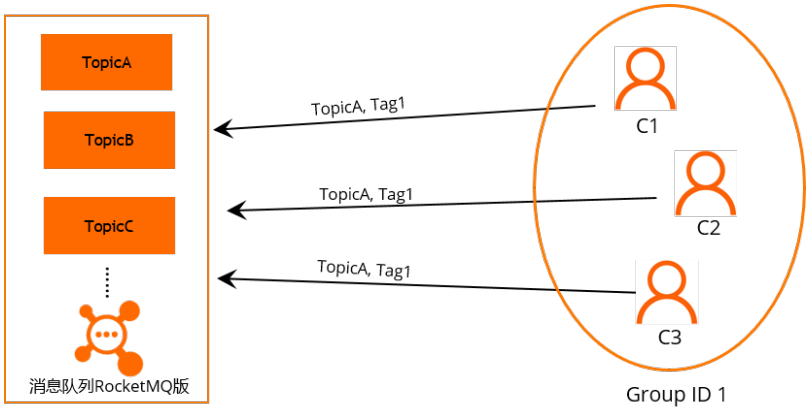


注意

消息队列RocketMQ版
支持使用TCP协议和HTTP协议的SDK客户端收发消息，除了保证同一Group ID下的Consumer实例订阅关系一致，还必须保证订阅消息的Group ID的协议版本和SDK的协议版本一致，例如，使用TCP协议的SDK收发消息，订阅消息时也必须使用创建的TCP协议的Group ID，否则会导致消息消费失败。

正确订阅关系一：订阅一个Topic且订阅一个Tag

如下图所示，同一Group ID下的三个Consumer实例C1、C2和C3分别都订阅了TopicA，且订阅TopicA的Tag也都是Tag1，符合订阅关系一致原则。



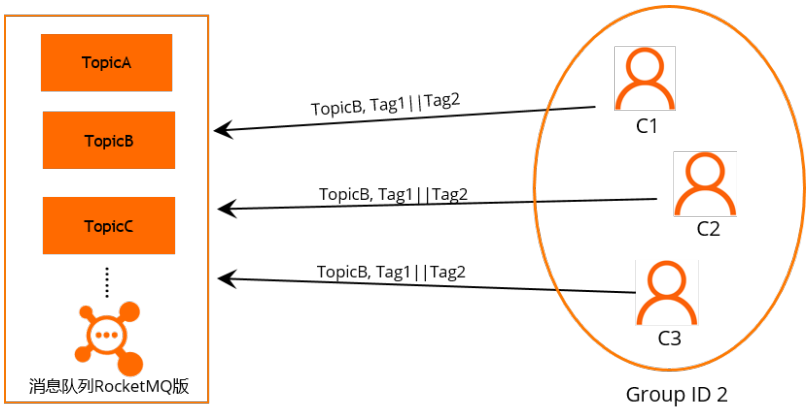
正确示例代码一

C1、C2、C3的订阅关系一致，即C1、C2、C3订阅消息的代码必须完全一致，代码示例如下：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "Tag1", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

正确订阅关系二：订阅一个Topic且订阅多个Tag

如下图所示，同一Group ID下的三个Consumer实例C1、C2和C3分别都订阅了TopicB，订阅TopicB的Tag也都是Tag1和Tag2，表示订阅TopicB中所有Tag为Tag1或Tag2的消息，且顺序一致都是Tag1||Tag2，符合订阅关系一致性原则。



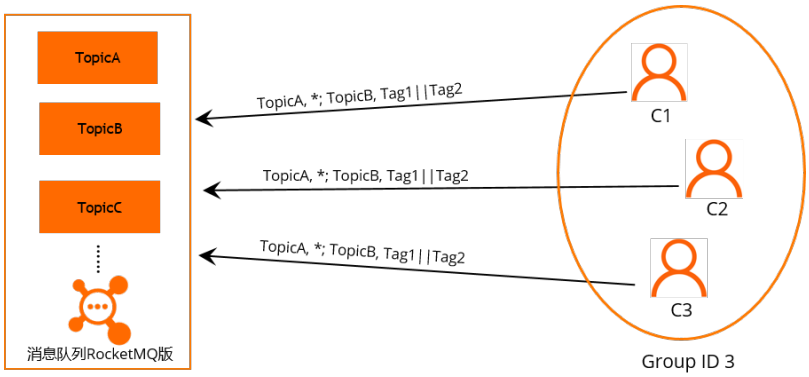
正确示例代码二

C1、C2、C3的订阅关系一致，即C1、C2、C3订阅消息的代码必须完全一致，代码示例如下：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicB", "Tag1||Tag2", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

正确订阅关系三：订阅多个Topic且订阅多个Tag

如下图所示，同一Group ID下的三个Consumer实例C1、C2和C3分别都订阅了TopicA和TopicB，且订阅的TopicA都未指定Tag，即订阅TopicA中的所有消息，订阅的TopicB的Tag都是Tag1和Tag2，表示订阅TopicB中所有Tag为Tag1或Tag2的消息，且顺序一致都是Tag1‖Tag2，符合订阅关系一致原则。



正确示例代码三

C1、C2、C3的订阅关系一致，即C1、C2、C3订阅消息的代码必须完全一致，代码示例如下：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
consumer.subscribe("TopicB", "Tag1||Tag2", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

查看订阅关系一致性

您可在消息

消息队列RocketMQ版

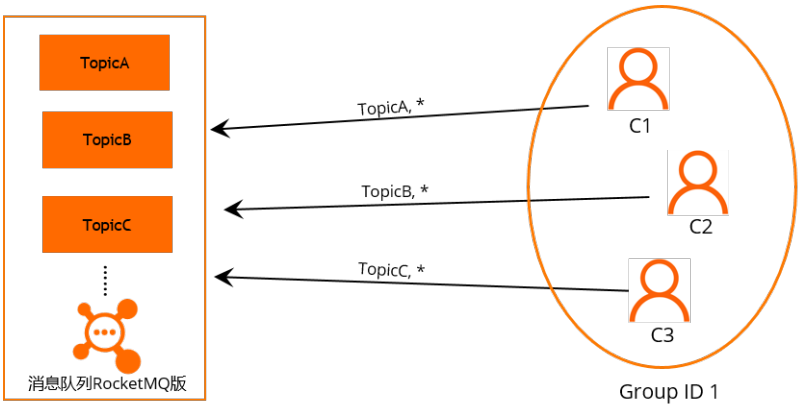
控制台**Group 详情**页面查看指定Group的订阅关系是否一致。具体操作，请参见查看消费者状态。若查询结果不一致，请参见常见订阅关系不一致问题排查Consumer实例的消费代码。

常见订阅关系不一致问题

使用

消息队列RocketMQ版
收发消息时，Consumer收到的消息不符合预期并且在
消息队列RocketMQ版
控制台查看到订阅关系不一致，则Consumer实例可能存在以下问题：

- **错误示例一：同一Group ID下的Consumer实例订阅的Topic不同。**
  如下图所示，同一Group ID下的三个Consumer实例C1、C2和C3分别订阅了TopicA、TopicB和TopicC，订阅的Topic不一致，不符合订阅关系一致性原则。



错误示例代码一

- Consumer实例1-1：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

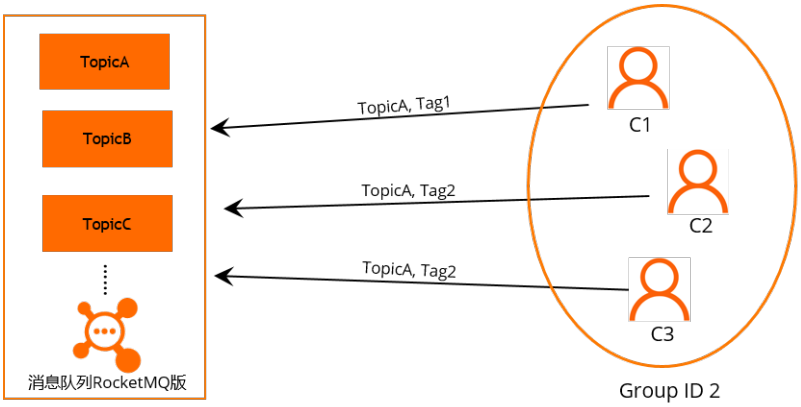
- Consumer实例1-2：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicC", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- Consumer实例1-3：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicB", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- **错误示例二：同一Group ID下的Consumer实例订阅的Topic相同，但订阅Topic的Tag不同。**
  如下图所示，同一Group ID下的三个Consumer实例C1、C2和C3分别都订阅了TopicA，但是C1订阅TopicA的Tag为Tag1，C2和C3订阅的TopicA的Tag为Tag2，订阅同一Topic的Tag不一致，不符合订阅关系一致性原则。



错误示例代码二

- Consumer实例2-1：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "Tag1", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

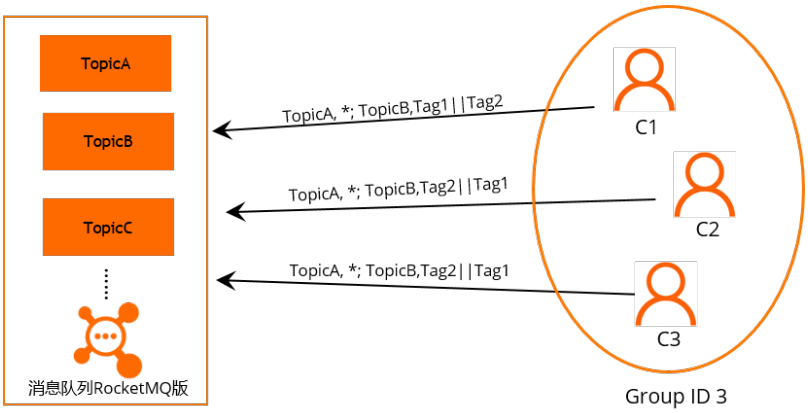
- Consumer实例2-2：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "Tag2", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- Consumer实例2-3：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "Tag2", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- **错误示例三：同一Group ID下的Consumer实例订阅的Topic及Topic的Tag都相同，但订阅的Tag顺序不同。**
  如下图所示，同一Group ID下的三个Consumer实例C1、C2和C3分别都订阅了TopicA和TopicB，并且订阅的TopicA都没有指定Tag，订阅TopicB的Tag都是Tag1和Tag2，但是C1订阅TopicB的Tag为Tag1||Tag2，C2和C3订阅的Tag为Tag2||Tag1，顺序不一致，不符合订阅关系一致性原则。



错误示例代码三

- Consumer实例3-1：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
consumer.subscribe("TopicB", "Tag1||Tag2", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- Consumer实例3-2：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "*", new MessageListener() {
   public Action consume(Message message, ConsumeContext context) {
       System.out.println(message.getMsgID());
       return Action.CommitMessage;
   }
});
consumer.subscribe("TopicB", "Tag2||Tag1", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- Consumer实例3-3：

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicA", "*", new MessageListener() {
   public Action consume(Message message, ConsumeContext context) {
       System.out.println(message.getMsgID());
       return Action.CommitMessage;
   }
});
consumer.subscribe("TopicB", "Tag2||Tag1", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

4.消息堆积和延迟问题

本文主要介绍
消息队列RocketMQ版
TCP协议的Java客户端使用过程中，经常会出现的消息堆积和消息延迟的问题。通过了解
消息队列RocketMQ版
客户端的消费原理和消息堆积的主要原因，帮助您可以在业务部署前更好的规划资源和配置，或在运维过程中及时调整业务逻辑，避免因消息堆积和延迟影响业务运行。

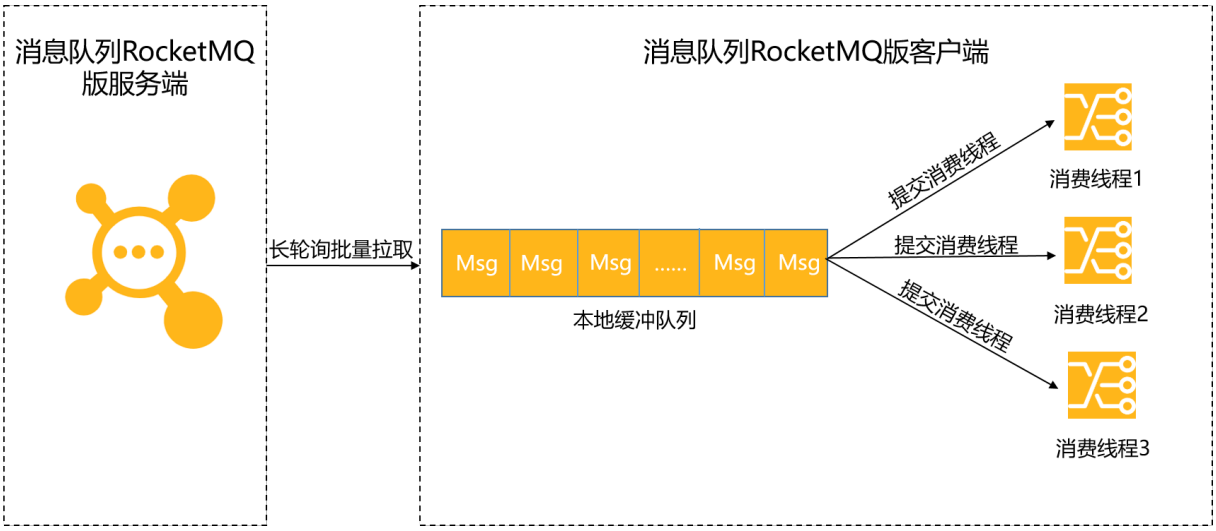
背景信息

消息处理流程中，如果客户端的消费速度跟不上服务端的发送速度，未处理的消息会越来越多，这部分消息就被称为堆积消息。消息出现堆积进而会造成消息消费延迟。以下场景需要重点关注消息堆积和延迟的问题：

- 业务系统上下游能力不匹配造成的持续堆积，且无法自行恢复。
- 业务系统对消息的消费实时性要求较高，即使是短暂的堆积造成的消息延迟也无法接受。

客户端消费原理

消息队列RocketMQ版
TCP协议客户端的消费流程如下图所示。



SDK客户端使用Push模式消费消息时，分为以下两个阶段：

- 阶段一：获取消息，SDK客户端通过长轮询批量拉取的方式从
  消息队列RocketMQ版
  服务端获取消息，将拉取到的消息缓存到本地缓冲队列中。
  SDK获取消息的方式为批量拉取，常见内网环境下都会有很高的吞吐量，例如：1个单线程单分区的低规格机器（4C8GB）可以达到几万TPS，如果是多个分区可以达到几十万TPS。所以这一阶段一般不会成为消息堆积的瓶颈。
- 阶段二：提交消费线程，SDK客户端将本地缓存的消息提交到消费线程中，使用业务消费逻辑进行处理。此时客户端的消费能力就完全依赖于业务逻辑的复杂度（消费耗时）和消费逻辑并发度了。如果业务处理逻辑复杂，处理单条消息耗时都较长，则整体的消息吞吐量肯定不会高，此时就会导致客户端本地缓冲队列达到上限，停止从服务端拉取消息。

通过以上客户端消费原理可以看出，消息堆积的主要瓶颈在于本地客户端的消费能力，即消费耗时和消费并发度。想要避免和解决消息堆积问题，必须合理的控制消费耗时和消息并发度，其中消费耗时的优先级高于消费并发度，必须先保证消费耗时的合理性，再考虑消费并发度问题。

消费耗时

影响消费耗时的消费逻辑主要分为CPU内存计算和外部I/O操作，通常情况下代码中如果没有复杂的递归和循环的话，内部计算耗时相对外部I/O操作来说几乎可以忽略。外部I/O操作通常包括如下业务逻辑：

- 读写外部数据库，例如MySQL数据库读写。
- 读写外部缓存等系统，例如Redis读写。
- 下游系统调用，例如Dubbo调用或者下游HTTP接口调用。

这类外部调用的逻辑和系统容量您需要提前梳理，掌握每个调用操作预期的耗时，这样才能判断消费逻辑中I/O操作的耗时是否合理。通常消费堆积都是由于这些下游系统出现了服务异常、容量限制导致的消费耗时增加。
例如：某业务消费逻辑中需要写一条数据到数据库，单次消费耗时为1 ms，平时消息量小未出现异常。业务侧进行大促活动时，写数据库TPS爆发式增长，并很快达到数据库容量限制，导致消费单条消息的耗时增加到100 ms，业务侧可以明显感受到消费速度大幅下跌。此时仅通过调整

消息队列RocketMQ版
SDK的消费并发度并不能解决问题，需要对数据库容量进行升配才能从根本上提高客户端消费能力。

消费并发度

消息队列RocketMQ版
消费消息的并发度计算方法如下表所示。

消息类型	消费并发度
普通消息	单节点线程数*节点数量
定时和延时消息	
事务消息	
顺序消息	Min（单节点线程数*节点数量，分区数）

客户端消费并发度由单节点线程数和节点数量共同决定，一般情况下需要优先调整单节点的线程数，若单机硬件资源达到上限，则必须通过扩容节点来提高消费并发度。

> **说明** 顺序消息的消费并发度还受Topic中分区个数的限制，具体分区数，请联系阿里云技术支持根据业务情况进行评估。

单节点的并发度需要谨慎设置，不能盲目直接调大线程数，设置过大的线程数反而会带来大量的线程切换的开销。理想环境下单节点的最优线程数计算模型如下：

- 单机vCPU核数为C。
- 线程切换耗时忽略不计，I/O操作不消耗CPU。
- 线程有足够消息等待处理，且内存充足。
- 逻辑中CPU计算耗时为T1，外部I/O操作为T2。

则单个线程能达到的TPS为1/（T1+T2），如果CPU使用率达到理想状态100%，那么单机达到最大能力时需要设置C*（T1+T2）/T1个线程。

> **注意** 这里计算的最大线程数仅仅是在理想环境下得到的理论数据，实际应用环境中建议逐步调大线程数并观察效果再进行调整。

如何避免消息堆积和延迟

为了避免在业务使用时出现非预期的消息堆积和延迟问题，您需要在前期设计阶段对整个业务逻辑进行完善的排查和梳理。整理出正常业务运行场景下的性能基线，才能在故障场景下迅速定位到阻塞点。其中最重要的就是梳理消息的消费耗时和消息消费的并发度。

- 梳理消息的消费耗时
  通过压测获取消息的消费耗时，并对耗时较高的操作的代码逻辑进行分析。查询消费耗时，请参见获取消息消费耗时。梳理消息的消费耗时需要关注以下信息：
  - 消息消费逻辑的计算复杂度是否过高，代码是否存在无限循环和递归等缺陷。
  - 消息消费逻辑中的I/O操作（如：外部调用、读写存储等）是否是必须的，能否用本地缓存等方案规避。
  - 消费逻辑中的复杂耗时的操作是否可以做异步化处理，如果可以是否会造成逻辑错乱（消费完成但异步操作未完成）。
- 设置消息的消费并发度
  i. 逐步调大线程的单个节点的线程数，并观测节点的系统指标，得到单个节点最优的消费线程数和消息吞吐量。
  ii. 得到单个节点的最优线程数和消息吞吐量后，根据上下游链路的流量峰值计算出需要设置的节点数，节点数=流量峰值/单线程消息吞吐量。

如何解决消息堆积和延迟问题

想要快速避免消息堆积和延迟给业务带来的影响，您可以通过

消息队列RocketMQ版
提供的监控报警功能，设置告警规则提前预警消息堆积问题，或通过业务埋点，触发报警事件，及时监控到消息堆积问题并进行处理。设置报警规则，请参见监控报警。

> **说明** 配置消息堆积告警规则时，请根据业务情况合理设置阈值。若阈值设置过小可能会造成频繁报警；阈值设置过大则不能及时收到报警并处理问题。

若收到消息堆积报警，处理方法，请参见如何处理消息堆积。

5.开源RocketMQ迁移上云

5.1. 开源RocketMQ迁移上云概述

和开源RocketMQ相比，阿里云
消息队列RocketMQ版
具有更高的稳定性、安全性及更完善的运维体系。您可以将开源RocketMQ集群迁移到
消息队列RocketMQ版
上以获得更好的业务体验，本文介绍开源RocketMQ集群迁移到
消息队列RocketMQ版
的原理和操作流程。

迁移原理

对于消息队列来说，如果要实现集群迁移，只需消费完旧集群的消息即可。由于Producer和Consumer都是集群化的，您可以通过一台一台操作的方式实现上层业务无感知。

迁移优势

和开源RocketMQ相比，
消息队列RocketMQ版
具有以下优势：

- **高稳定性**：
  消息队列RocketMQ版
  作为阿里巴巴双十一官方指定消息产品，支撑阿里巴巴集团所有的消息服务，历经十余年高可用与高可靠的严苛考验，具有更高的稳定性。
- **高性能**：历年双11购物狂欢节零点千万级TPS、万亿级数据洪峰，创造了全球最大的业务消息并发以及流转纪录（日志类消息除外）；在始终保证高性能前提下，支持亿级消息堆积，不影响集群的正常服务。
- **丰富的消息类型**：提供丰富的消息类型，满足各种严苛场景下的高级特性需求，当前支持的消息类型涵盖普通消息、顺序消息（全局顺序和分区顺序）、分布式事务消息、定时消息、延时消息。
- **完善的运维体系**：
  消息队列RocketMQ版
  支持消息查询、全链路消息轨迹查询以及消息回溯等功能，帮助您快速发现和处理系统问题，提高运维效率。
- **安全访问控制**：以消息主题、订阅组的粒度，对每一条消息的收、发请求都进行严格的访问控制，确保消息的安全性；全面支持阿里云RAM主子账号、黑白名单、STS等功能，支持TLS传输加密协议、阿里云VPC访问等。

迁移操作流程

开源RocketMQ迁移到

消息队列RocketMQ版
的操作流程如下图所示：

步骤一：创建迁移任务

在

消息队列RocketMQ版
控制台创建迁移任务，将开源RocketMQ导出的元数据文件导入至

消息队列RocketMQ版

。

步骤二：迁移评估

从技术和成本方面分别评估迁移上云的条件。

步骤三：迁移元数据

将Topic和Group的元数据迁移至云上的

消息队列RocketMQ版
实例中。

步骤四：迁移消息服务

分批将消息生产者集群和消费者集群的节点连接到云上的

消息队列RocketMQ版
实例，完成消息收发链路的平滑迁移。

5.2. 步骤一：创建迁移任务

本文介绍如何在
消息队列RocketMQ版
控制台创建迁移上云任务。

背景信息

消息队列RocketMQ版
提供元数据导出工具，支持将开源RocketMQ集群的元数据导出为一个JSON文件，您在创建迁移上云任务前需要按照准备工作中的操作导出开源RocketMQ的元数据文件，再将该文件上传至
消息队列RocketMQ版
控制台。

准备工作

使用元数据导出工具导出开源RocketMQ的元数据文件。

根据您的开源RcoketMQ服务器的网络类型，

消息队列RocketMQ版
提供以下两种方式导出开源RcoketMQ的元数据。

- **导出数据方法一：** 下载脚本上传到RocketMQ服务器运行，运行以下命令。

  ```
  ./bin/export.sh
  ```

  操作流程：下载脚本→上传至服务器→执行 `./bin/export.sh`

- **导出数据方法二：** 如果RocketMQ服务器可访问公网，直接运行以下命令。

  ```
  /bin/bash -c "$(curl -fsSL https://ons-migration.oss-cn-hangzhou.aliyuncs.com/export.sh)"
  ```

1. 访问rocketmq-for-export.tar.gz地址下载元数据导出工具。
2. 将下载好的工具上传至开源RocketMQ机器。
3. 在工具所在目录下执行 `./bin/export.sh` 命令运行元数据导出工具。
4. 按提示依次输入要连接的开源RocketMQ服务器地址、需要下载元数据的集群名称以及导出文件的保存路径。

```
//输入开源RocketMQ的服务器地址及端口号，端口固定取值为9876。
Enter name server address list: example.net:9876
//输入需要导出元数据的开源RocketMQ集群名称。
Choose a cluster to export: DefaultCluster
//输入导出的元数据文件的存放路径，不输入则文件被存放至默认路径/tmp/rocketmq/config。
Enter file path to export [default /tmp/rocketmq/export]:
```

返回如下内容说明导出元数据成功：

```
[INFO] The RocketMQ metadata has been export to the file:/tmp/rocketmq/config/rocketmq-metadata-export.json
```

1. 在开源RocketMQ的机器上，执行以下命令一键下载并运行元数据导出工具。

> **说明** 命令中的https://ons-migration.oss-cn-hangzhou.aliyuncs.com为脚本的存放路径，请勿修改。

```
/bin/bash -c "$(curl -fsSL https://ons-migration.oss-cn-hangzhou.aliyuncs.com/export.sh)"
```

2. 按提示依次输入要连接的开源RocketMQ服务器地址、需要下载元数据的集群名称以及导出文件的保存路径。

```
//输入开源RocketMQ的服务器地址及端口号，端口固定取值为9876。
Enter name server address list: example.net:9876
//输入需要导出元数据的开源RocketMQ集群名称。
Choose a cluster to export: DefaultCluster
//输入导出的元数据文件的存放路径，不输入则文件被存放至默认路径/tmp/rocketmq/config。
Enter file path to export [default /tmp/rocketmq/export]:
```

返回如下内容说明导出元数据成功：

```
[INFO] The RocketMQ metadata has been export to the file:/tmp/rocketmq/config/rocketmq-metadata-export.json
```

方法一操作步骤

方法二操作步骤

创建迁移任务

1. 登录
   消息队列RocketMQ版
   控制台。
2. 在左侧导航栏单击**迁移上云**。
3. 在顶部菜单栏，选择地域，如**华东1（杭州）**。
4. 在**迁移上云**页面左上角单击**创建任务**。
5. 在**创建任务**配置向导页面，完成以下操作并单击**下一步**。

i. 在**任务名称**文本框输入迁移上云的任务名称。

> **说明** 任务名称长度限制为3~64个字符，只能包含中文、英文、数字、短划线（-）和下划线（_）。

ii. 单击**元数据**参数右侧的**点击上传元数据文件**按钮，选择提前导出的JSON格式的元数据文件。元数据导出操作，请参见准备工作。

后续步骤

步骤二：迁移评估

5.3. 步骤二：迁移评估

进行元数据迁移操作前，您可以分别从技术层面和成本层面对迁移任务进行评估。技术评估帮助您判断当前开源RocketMQ实例的客户端等环境信息是否符合要求、明确迁移前和迁移后的功能支持情况；成本评估可以帮助您提前计算成本选择合适的目标实例。

前提条件

步骤一：创建迁移任务

操作步骤

1. 登录消息队列RocketMQ版控制台。
2. 在左侧导航栏单击**迁移上云**。
3. 在**迁移上云**任务列表中选择指定的任务，在其**操作**列单击**详情**。
4. 在**迁移评估**配置向导中，单击**技术评估**页签，仔细阅读并选中所有评估项。
   - 单击**下一步**直接完成迁移评估。
   - 执行步骤5继续进行成本评估。

   技术评估项具体内容和说明如下。技术评估项说明

评估项	内容	说明
客户端版本确认	Java SDK	如果您的开源RocketMQ客户端使用的是Java SDK，请确保您使用的客户端的SDK版本为4.9.0及以上。
	C++ SDK	如果您的开源RocketMQ客户端使用的是C++ SDK，请确保您的SDK版本为2.0.1及以上。
	spring-cloud-alibaba	如果您的开源RocketMQ客户端使用的是spring-cloud-alibaba，请确保您的SDK升级为最新版本。
	定时消息	云上支持定时时间秒级精度，建议使用云上使用方式实现。更多信息，请参见定时和延时消息。

评估项	内容	说明
功能确认	Pull消费接口情况	仅企业铂金版支持Pull消费方式，标准版不支持，如果使用Pull方式消费消息，请提前准备消息队列RocketMQ版的企业铂金版实例作为迁移的目标实例。更多信息，请参见订阅消息。
云上Quotas限制	消息大小	■ 普通和顺序消息：4 MB ■ 事务和定时或延时消息：64 KB **说明** 其中，所有消息类型的消息属性大小均不能超过16 KB。
	消息保留时长	最大保留时长为3天。
	定时/延时消息的延时时长	最大延时时长为40天。

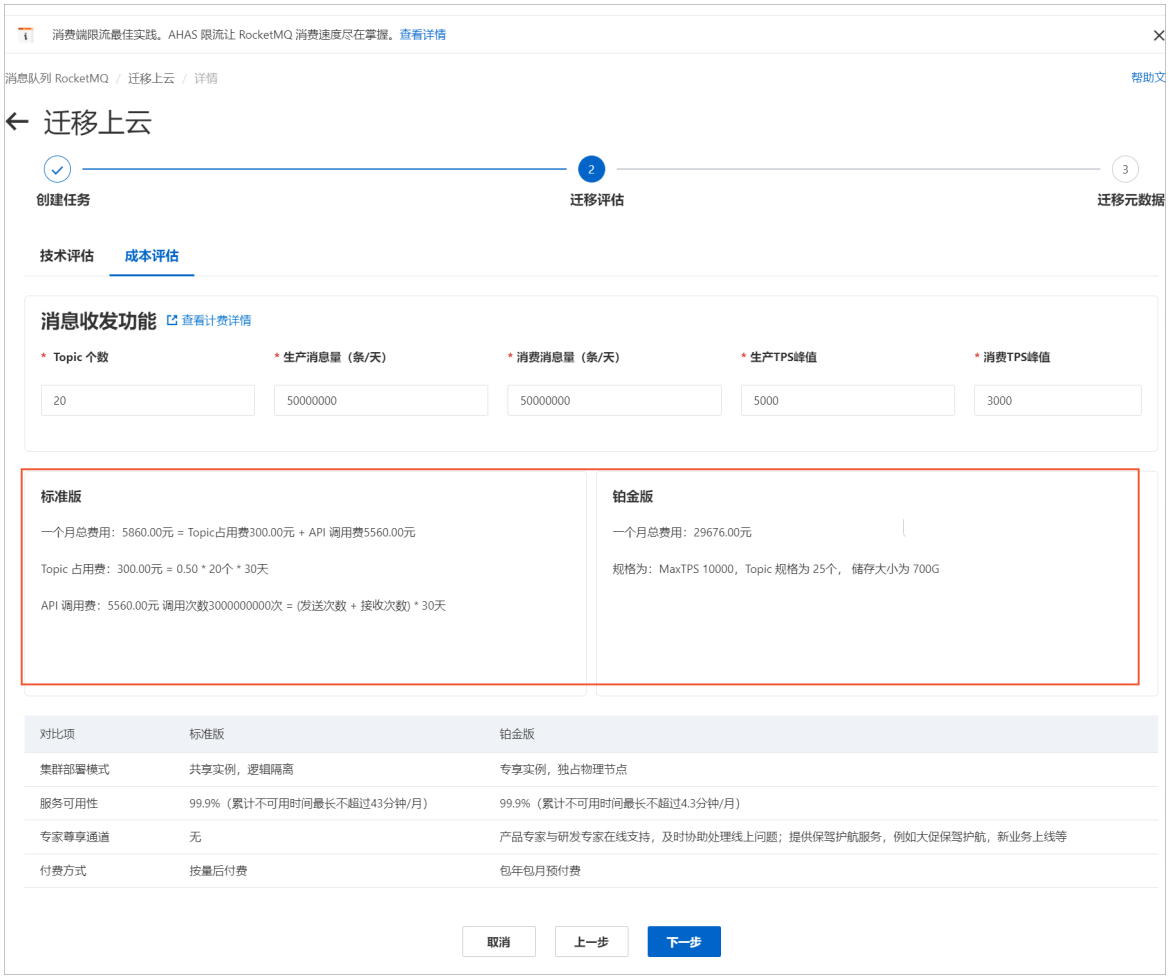
5. （可选）在**迁移评估**配置向导的**成本评估**页签中，按照预估的消息业务量输入消息收发数据，然后单击**下一步**完成成本评估。

   需要输入的消息收发数据如下：

   - **Topic 个数**：您需要创建的Topic个数。
   - **生产消息量（条/天）**：一天的消息生产总量。
   - **消费消息量（条/天）**：一天的消息消费总量。
   - **生产TPS峰值**：每秒钟消息发送条数的最大值。
   - **消费TPS峰值**：每秒钟消息消费条数的最大值。

   系统会自动计算并显示标准版实例和企业铂金版实例的费用成本，您可以根据实例的预估费用和功能差异选择合适的实例类型和规格。实例的功能差异，更多信息，请参见实例规格。

   说明 该流程中显示的成本费用仅为预估的消息收发费用，实际生产环境中产生的费用请以最终的费用账单为准。



后续步骤

步骤三：迁移元数据

5.4. 步骤三：迁移元数据

本文介绍如何在
消息队列RocketMQ版
控制台将开源RocketMQ的元数据迁移到
消息队列RocketMQ版
实例。

背景信息

元数据迁移指的是将开源RocketMQ的Topic和Group的配置信息迁移到

消息队列RocketMQ版
上，并不会迁移Topic中的消息数据。

消息队列RocketMQ版
提供元数据导出工具，支持将开源RocketMQ的元数据导出为一份JSON文件，然后通过
消息队列RocketMQ版
控制台导入至
消息队列RocketMQ版
上的目标实例。
消息队列RocketMQ版

会根据元数据文件中的配置信息在目标实例中创建对应的Topic和Group，
消息队列RocketMQ版
创建的Topic和Group的名称及数量和开源RocketMQ集群下的一致。

前提条件

- 已导入需要迁移的开源RocketMQ集群的元数据文件并创建好迁移任务。具体操作，请参见步骤一：创建迁移任务。
- 已完成迁移评估。具体操作，请参见步骤二：迁移评估。
- 根据评估结果已准备好一个符合要求的
  消息队列RocketMQ版
  实例作为迁移的目标实例。若未创建实例，请参见实例管理创建。

迁移元数据

1. 登录
   消息队列RocketMQ版
   控制台。
2. 在左侧导航栏单击**迁移上云**。
3. 在顶部菜单栏，选择地域，如**华东1（杭州）**。
4. 在**迁移上云**任务列表中选择指定的任务，在其**操作**列单击**详情**。
5. 在**迁移元数据**配置向导页面的**目标实例**下拉菜单中，选择已创建好的
   消息队列RocketMQ版
   实例作为元数据导入的目标实例，并单击**确认**。
   此时所有的Group元数据已经在后台完成自动导入，界面只显示所有Topic的资源列表，您需要完成所有Topic类型的订正及导入操作后才能查看所有资源的导入结果。
6. 完成所有Topic消息类型的订正和导入操作，然后单击页面下方的**确认**。
   i. 在资源列表中选择指定的Topic资源，在其**消息类型**列的下拉菜单中选择Topic类型，然后在其**操作**列单击**确认并导入**；您也可以选中多个Topic，确认完所有选中Topic的类型后，单击页面左下角的**批量确认并导入**。
   ii. 在弹出的**提示**对话框中单击**确认**。

   资源列表中将显示所有Topic和Group的信息及迁移结果。您可以在资源列表上方的**迁移详情**区域查看所有资源的迁移结果总览。您也可以根据资源名称、资源类型或执行结果进行过滤，查看指定资源的迁移结果。

结果验证

若迁移成功，则目标

消息队列RocketMQ版
实例下会创建好对应的Topic和Group。您可以在**Topic 管理**和**Group 管理**页面的资源列表中查看。

1. 登录
   消息队列RocketMQ版
   控制台。
2. 在左侧导航栏，单击**实例列表**。
3. 在顶部菜单栏，选择地域，如**华东1（杭州）**。
4. 在**实例列表**页面，选择指定的实例并单击实例名称或在其**操作**列单击**详情**。
5. 在左侧导航栏单击**Topic 管理**，在Topic列表中查看是否已存在对应Topic。
6. 在左侧导航栏单击**Group 管理**，在Group列表中查看是否已存在对应的Group。

失败原因说明

若迁移失败，资源列表中会显示迁移失败的可能原因。

可能原因	说明
您指定的Topic已存在。	目标实例下已存在和待创建Topic名称相同的Topic。
您请求创建的Group ID已存在，请确认。	目标实例下已存在和待创建Group ID相同的Group。
指定的实例不存在。	请在**实例列表**页面查看目标实例是否存在。
实例权限验证失败。请确认实例权限信息后再试。	请确认目标实例的所属权限。
指定的实例不在服务中。	请在**实例列表**页面查看目标实例的状态信息。
无法创建 Topic，该实例的 Topic 数量已达容量上限，请前往铂金版实例管理页面进行规格升级。	目标实例下的Topic数量已达到最大上限，请删除不再使用的Topic，或对该实例进行升配操作。
实例没有相关权限。	请检查实例的所属权限和授权策略。
Topic不能和Group重名。	Topic名称和Group ID不能重名。
Group不能和Topic重名。	Topic名称和Group ID不能重名。
您指定的 Group ID 属于其他实例。	其他实例下已存在和待创建Group ID相同的Group。
创建 Group ID 失败。请稍后再试。	创建Group ID失败，请稍后重试或提交工单联系技术支持人员。
创建Topic失败。	Topic创建失败，请稍后重试或提交工单联系技术支持人员。
系统处理错误，请联系系统管理员。	系统错误，请提交工单联系技术支持人员。
参数验证失败。	参数缺失或者传入值非法，请提交工单联系技术支持人员。

后续步骤

步骤四：迁移消息服务

5.5. 步骤四：迁移消息服务

本文介绍如何将消息收发服务从开源RocketMQ迁移到阿里云的
消息队列RocketMQ版
上，实现业务上云的平滑迁移。

背景信息

迁移消息服务指的是在元数据迁移完成后，通过手动修改生产者集群和消费者集群的接入信息，将生产者集群和消费者集群从开源RocketMQ切换到阿里云

消息队列RocketMQ版
的实例上，最终实现所有的消息收发业务都在
消息队列RocketMQ版
实例上进行。

迁移消息服务只迁移消息生产和消费链路，并不会迁移开源RocketMQ上的消息数据。

本文主要介绍使用双读双写、分批发布方案迁移消息服务的方法。迁移过程中，生产者集群和消费者集群可并行在开源RocketMQ集群和

消息队列RocketMQ版
集群上同时生产或消费消息，不会因迁移产生数据积压，业务可平滑过渡。具体方案和操作流程，请参见下文中的迁移消息服务。

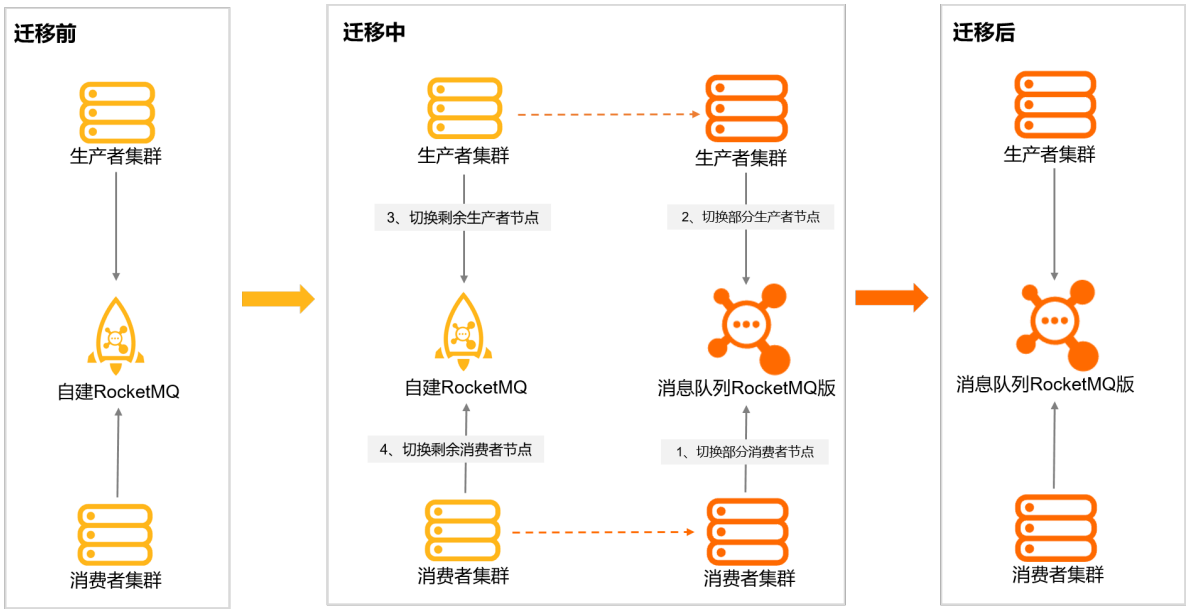
前提条件

已将开源RocketMQ的元数据迁移到

消息队列RocketMQ版
实例上，具体操作，请参见步骤三：迁移元数据。

迁移消息服务

消息服务切换流程和步骤如下所示：



说明 以下操作中，切换消费者集群节点或生产者集群节点的接入信息，指的是将生产者或消费者客户端的接入信息修改为要迁移的目标
消息队列RocketMQ版
实例的接入信息。包括
消息队列RocketMQ版
实例的**接入点**、实例所属账号的AccessKey和AccessKey Secret。

- 获取**接入点**，请参见创建资源。
- 获取AccessKey和AccessKey Secret，请参见获取AK、SK。

1. 切换消费者集群中部分节点的接入信息，将这部分消费者接入到云上的
   消息队列RocketMQ版
   。
   切换的这部分消费者将消费
   消息队列RocketMQ版
   集群中的消息，剩余消费者继续消费开源RocketMQ集群中的消息。
2. 切换生产者集群中部分节点的接入信息，将这部分生产者接入到云上

消息队列RocketMQ版
。
切换的这部分生产者将发送消息到
消息队列RocketMQ版
集群中；剩余的生产者还是将消息发送到开源RocketMQ集群中。

3. 将剩余的生产者全部接入到
消息队列RocketMQ版
上。
此时所有消息将全部被发送到云上的
消息队列RocketMQ版
集群中。
4. 将剩余的消费者全部接入到
消息队列RocketMQ版
上。

> **注意** 切换剩余消费者之前，请确保开源RocketMQ中的消息已全部消费完，否则可能会导致消费遗漏。您可以通过查看开源RocketMQ中的消息堆积量来判断消息是否消费完成。

此时所有的生产者和消费者都迁移到
消息队列RocketMQ版
集群上，所有的消息收发都在
消息队列RocketMQ版
集群中完成。

6.最佳实践常见问题

6.1. 如何处理消息堆积

Condition

在使用
消息队列Rocket MQ版
实例时收到消息堆积告警，登录
消息队列Rocket MQ版
控制台后发现了下列现象：

- 在**Group 详情**页面，看到Group ID的**实时消息堆积量**的值高于预期。
- 导航栏中选择**消息轨迹**，单击**创建查询任务**，选择按**按 Message ID 查询**，输入对应的信息，发现部分消息已发送至Broker节点，但未投递给下游消费者。

Cause

消息队列Rocket MQ版
的消息发送至Broker节点后，配置了Group ID的客户端根据当前的消费位点，从Broker节点拉取部分消息到本地进行消费。一般情况下，客户端从Broker节点拉取消息的过程不会导致消息堆积，主要是客户端本地消费过程中，由于消费耗时过长或消费并发度较小等原因，导致客户端消费能力不足，出现消息堆积的问题。具体的消费原理和消息堆积原因请参见消息堆积和延迟问题。

Remedy

若出现消息堆积，可参考以下措施进行定位和处理。

操作步骤

1. 判断消息堆积在
   消息队列Rocket MQ版
   服务端还是客户端。

   查看客户端本地日志文件 `ons.log` ，搜索是否出现如下信息：

   ```
   the cached message count exceeds the threshold
   ```

   - 出现相关日志信息，说明客户端本地缓冲队列已满，消息堆积在客户端，请执行步骤2。
   - 若未出现相关日志，说明消息堆积不在客户端，若出现这种特殊情况，请直接提交工单联系阿里云技术支持。
2. 确认消息的消费耗时是否合理。
   - 若查看到消费耗时较长，则需要查看客户端堆栈信息排查具体业务逻辑，请执行步骤3。
   - 若查看到消费耗时正常，则有可能是因为消费并发度不够导致消息堆积，需要逐步调大消费线程或扩容节点来解决。

   消息的消费耗时可以通过以下方式查看：

   - 登录
     消息队列Rocket MQ版
     控制台查看消息的消费轨迹，在**消费者**区域中可以看到单条消息的**消费耗时**。具体操作，请参见查询消息轨迹。

- 登录
  消息队列RocketMQ版
  控制台查看消费者状态，在客户端连接信息中查看**业务处理时间**，获取消费耗时的平均值。具体操作，请参见查看消费者状态。

- 使用阿里云ARMS等其他监控产品做业务埋点采集消息的消费耗时。

3. 查看客户端堆栈信息。只需要关注线程名为ConsumeMessageThread的线程，这些都是业务消费消息的逻辑。可参见Java官方文档判断线程的状态并根据具体问题修改业务逻辑。

   客户端堆栈信息可以通过以下方式获取：

   - 登录
     消息队列RocketMQ版
     控制台查看消费者状态，在客户端连接信息中查看**Java客户端堆栈信息**。具体操作，请参见查看消费者状态。
   - 使用Jstack工具打印堆栈信息。
     a. 请参见查看消费者状态获取消息堆积的消费者实例所对应的宿主机IP地址，并登录该宿主机。
     b. 执行以下任意命令，查看并记录Java进程的PID。

     ```
     ps -ef
     |grep javajps -lm
     ```

     c. 执行以下命令，查看堆栈信息。

     ```
     jstack -l pid > /tmp/pid.jstack
     ```

     d. 执行以下命令，查看 `ConsumeMessageThread` 的信息。

     ```
     cat /tmp/pid.jstack|grep ConsumeMessageThread -A 10 --color
     ```

   常见的异常堆栈信息如下：

   - 示例一：空闲无堆积的堆栈。
     消费空闲情况下消费线程都会处于WAITING状态等待从消费任务队列里中获取消息。

Java 客户端堆栈信息

Thread	Stack
ConsumeMessageThread_7	TID: 53 STATE: WAITING sun.misc.Unsafe.park(Native Method) java.util.concurrent.locks.LockSupport.park(LockSupport.java:175) java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039) java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442) java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1074) java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1134) java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) java.lang.Thread.run(Thread.java:748)

- 示例二：消费逻辑有抢锁休眠等待等情况。
  消费线程阻塞在内部的一个睡眠等待上，导致消费缓慢。

Thread	Stack
ConsumeMessageThread_16	TID: 51 STATE: TIMED_WAITING java.lang.Thread.sleep(Native Method) mqtest.DelayTest$1.consume(DelayTest.java:51) com.aliyun.openservices.ons.api.impl.rocketmq.ConsumerImpl$MessageListenerImpl.consumeMessage(ConsumerImpl.java:101) com.aliyun.openservices.shade.com.alibaba.rocketmq.client.impl.consumer.ConsumeMessageConcurrentlyService$ConsumeRequest.run(ConsumeMessageConcurrentlyService.java:415) java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) java.util.concurrent.FutureTask.run(FutureTask.java:266) java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) java.lang.Thread.run(Thread.java:748)

- 示例三：消费逻辑操作数据库等外部存储卡住。
  消费线程阻塞在外部的HTTP调用上，导致消费缓慢。

ConsumeMessageThread_3	TID: 54 STATE: RUNNABLE java.lang.ClassLoader.loadClass(ClassLoader.java:404) sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:349) java.lang.ClassLoader.loadClass(ClassLoader.java:357) org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:174) org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:158) org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:149) org.apache.http.impl.conn.PoolingHttpClientConnectionManager.<init>(PoolingHttpClientConnectionManager.java:125) refactor.base.Tools.getHttpsClient(Tools.java:138) refactor.base.Tools.httpsPost(Tools.java:257) mqtest.DelayTest$1.consume(DelayTest.java:58) com.aliyun.openservices.ons.api.impl.rocketmq.ConsumerImpl$MessageListenerImpl.consumeMessage(ConsumerImpl.java:101) com.aliyun.openservices.shade.com.alibaba.rocketmq.client.impl.consumer.ConsumeMessageConcurrentlyService$ConsumeRequest.run(ConsumeMessageConcurrentlyService.java:415) java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) java.util.concurrent.FutureTask.run(FutureTask.java:266) java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) java.lang.Thread.run(Thread.java:748)

4. 针对某些特殊业务场景，如果消息堆积已经影响到业务运行，且堆积的消息本身可以跳过不消费，您可以通过重置消费位点跳过这些堆积的消息从最新位点开始消费，快速恢复业务。具体操作，请参见重置消费位点。