

分布式消息服务 RocketMQ 版

## 最佳实践

文档版本

01

发布日期

2024-09-14



版权所有 © 华为云计算技术有限公司 2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



HUAWEI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云计算技术有限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，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

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 通过消息幂等实现消息去重.....            | 1  |
| 2 通过 Topic 和 Tag 实现消息分类.....   | 3  |
| 3 实现订阅关系一致.....                | 5  |
| 4 消息堆积处理建议.....                | 8  |
| 5 DMS for RocketMQ 安全使用建议..... | 11 |

# 1 通过消息幂等实现消息去重

## 方案概述

在RocketMQ的业务处理过程中，如果消息重发了多次，消费者端对该重复消息消费多次与消费一次的结果是相同的，多次消费并没有对业务产生负面影响，那么这个消息处理过程是幂等的。消息幂等保证了无论消息被重复投递多少次，最终的处理结果都是一致的，避免了因消息重复而对业务产生影响。

例如在支付场景下，用户购买商品后进行支付，由于网络不稳定导致用户收到多次扣款请求，导致重复扣款。但实际上扣款业务只应进行一次，商家也只应产生一条订单流水。这时候使用消息幂等就可以避免这个问题。

在实际应用中，导致消息重复的原因有网络闪断、客户端故障等，且可能发生在消息生产阶段，也可能发生在消息消费阶段。因此，可以将消息重复的场景分为以下两类：

- 生产者发送消息时发生消息重复：  
生产者发送消息时，消息成功发送至服务端。如果此时发生网络闪断，导致生产者未收到服务端的响应，此时生产者会认为消息发送失败，因此尝试重新发送消息至服务端。当消息重新发送成功后，在服务端中就会存在两条内容相同的消息，最终消费者会消费到两条内容一样的重复消息。
- 消费者消费消息时发生消息重复：  
消费者消费消息时，服务端将消息投递至消费者并完成业务处理。如果此时发生网络闪断，导致服务端未收到消费者的响应，此时服务端会认为消息投递失败。为了保证消息至少被消费一次，服务端会尝试投递之前已被处理过的消息，最终消费者会消费到两条内容一样的重复消息。

## 实施方法

从上面的消费重复场景可以看到，不同Message ID的消息可能有相同的消息内容，因此Message ID无法作为消息的唯一标识符。RocketMQ可以为消息设置Key，把业务的唯一标识作为消息的唯一标识，从而实现消息的幂等。为消息设置Key的示例代码如下：

```
Message message = new Message();
message.setKey("Order_id"); // 设置消息的Key，可以使用业务的唯一标识作为Key，例如订单号等。
SentResult sendResult = mqProducer.send(message);
```

生产者发送消息时，消息已经设置了唯一的Key，在消费者消费消息时，可以根据消息的Key进行幂等处理。消费者通过getKeys()能够读取到消息的唯一标识（如订单号等），业务逻辑围绕该唯一标识进行幂等处理即可。

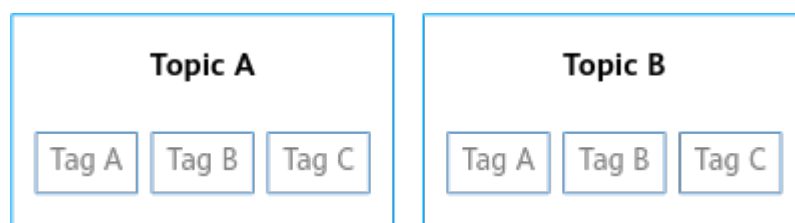
# 2 通过 Topic 和 Tag 实现消息分类

## 方案概述

Topic是消息关联的基础逻辑单元，消息的生产与消费围绕着Topic进行。每个Topic包含若干条消息，每条消息只能属于一个Topic。

Tag是消息的标签，用于在同一Topic下区分不同类型的消息。来自同一业务单元的消息，可以根据不同业务目的在同一Topic下设置不同的标签。标签能够有效保持代码的清晰度和连贯性，并优化RocketMQ提供的查询系统。消费者可以根据Tag实现对不同Topic的不同消费逻辑，实现更好的扩展性。

Topic是消息的一级分类，Tag是消息的二级分类，关系如下图。



## 应用场景

在实际业务中，通过合理使用Topic和Tag，可以使业务结构更清晰，提高效率。可以根据如下几方面判断Topic和Tag的具体使用场景：

- 根据消息类型判断：RocketMQ的消息可分为普通消息、顺序消息、定时/延迟消息、事务消息，不同类型的消息需要用不同的Topic来区分，无法通过Tag区分。
- 根据消息优先级判断：在业务中紧急程度高和紧急程度一般的消息用不同的Topic来区分，方便后续业务处理。
- 根据业务关联性判断：业务逻辑上不相关的消息用不同Topic来区分，业务逻辑上强关联的消息发送到同一Topic下，并用Tag进行子类型或流程先后关系的区分。

## 实施方法

以物流运输场景为例，普通货物订单消息和生鲜货物订单消息属于不同类型的消息，分别创建Topic\_Common和Topic\_Fresh。在不同消息类型中，以不同的Tag划分不同的订单目的省份。

- Topic: Topic\_Common

- Tag = Province\_A
- Tag = Province\_B
- Topic: Topic\_Fresh
  - Tag = Province\_A
  - Tag = Province\_B

以生产发往Province A的普通货物订单消息为例，代码示例如下：

```
Message msg = new Message("Topic_Common", "Province_A" /* Tag */, ("Order_id " +  
i).getBytes(RemotingHelper.DEFAULT_CHARSET));
```

以订阅发往Province A和Province B的生鲜货物订单消息为例，代码示例如下：

```
consumer.subscribe("Topic_Fresh", "Province_A || Province_B");
```

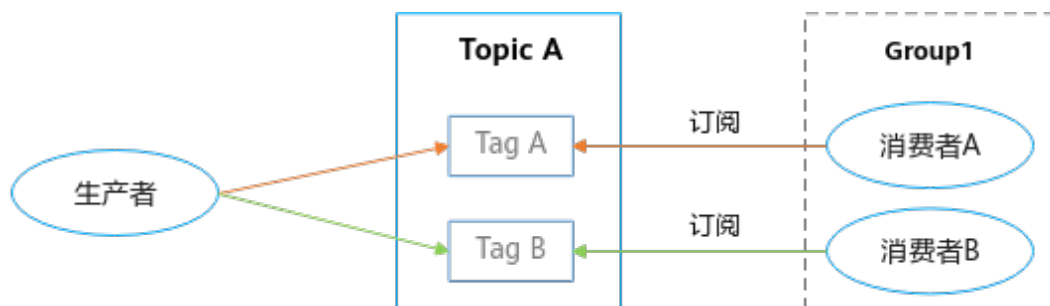
## 不同消费者消费不同 Tag

在实际使用场景中，可能会遇到不同的消费者消费同一Topic的不同Tag消息。对于同一个Topic的不同Tag，如果RocketMQ消费者设置了相同的消费组，会导致消息消费混乱。

例如Topic A下有Tag A和Tag B，消费者A订阅了Tag A的消息，消费者B订阅了Tag B的消息。

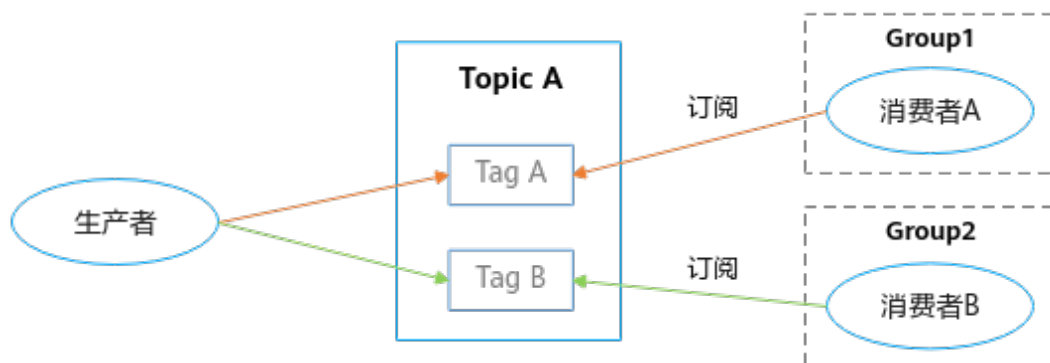
如果消费者A和消费者B设置了相同的消费组，当生产者发送Tag A的消息时，Tag A的消息会均匀发送给消费者A和消费者B。由于消费者B没有订阅Tag A的消息，会把Tag A消息过滤掉，从而导致部分Tag A消息未被消费。

图 2-1 错误的消费组设置



这种情况下，把消费者A和消费者B设置不同的消费组，即可解决消费混乱的问题。

图 2-2 正确的消费者组设置



# 3 实现订阅关系一致

## 方案概述

订阅关系一致指的是同一个消费组下所有消费者所订阅的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。

### 消费原理

RocketMQ为每个Topic划分了消息队列（Queue），队列数越大消费的并发度越大。一个消费组表示一个消费者群组，在分布式应用场景下，同一个消费组中的多个消费者共同完成Topic所有Queue的消费。Queue的分配以消费组为单位，会均匀分配给消费组下的消费者，而不会在意该消费者是否订阅了当前Topic。一个消费者只会分配到Topic中的某几个Queue，而一个Queue只会分配给一个消费者。

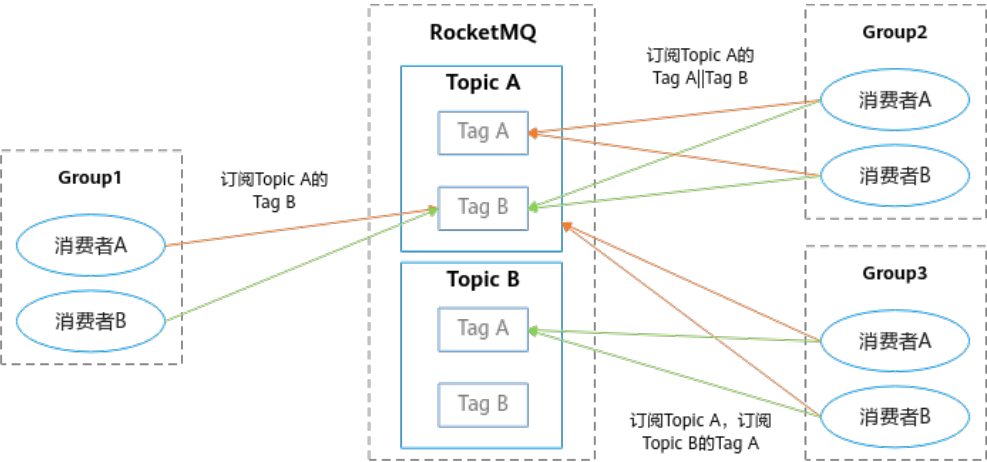
### 正确的订阅关系

在分布式应用场景下，一个消费组中所有的消费者拥有一个相同的消费组ID，他们需要订阅相同的Topic和Tag，保持订阅关系一致，才能保证消息消费逻辑正确，消息不丢失。

- 同一个消费组的消费者必须订阅同一个Topic。例如，消费组Group1中有消费者A和消费者B，消费者A订阅了Topic A和Topic B，则消费者B也必须订阅Topic A和Topic B，不能只订阅Topic A或只订阅Topic B，或者额外订阅Topic C。
- 同一个消费组的消费者订阅同一个Topic下的Tag必须一致，包括Tag的数量和顺序。例如，消费组Group2中有消费者A和消费者B，消费者A订阅Topic A且Tag为Tag A||Tag B，则消费者B订阅Topic A时Tag也必须为Tag A||Tag B，不能只订阅Tag A或只订阅Tag B或订阅Tag B||Tag A。



图 3-1 正确的订阅关系一致设置

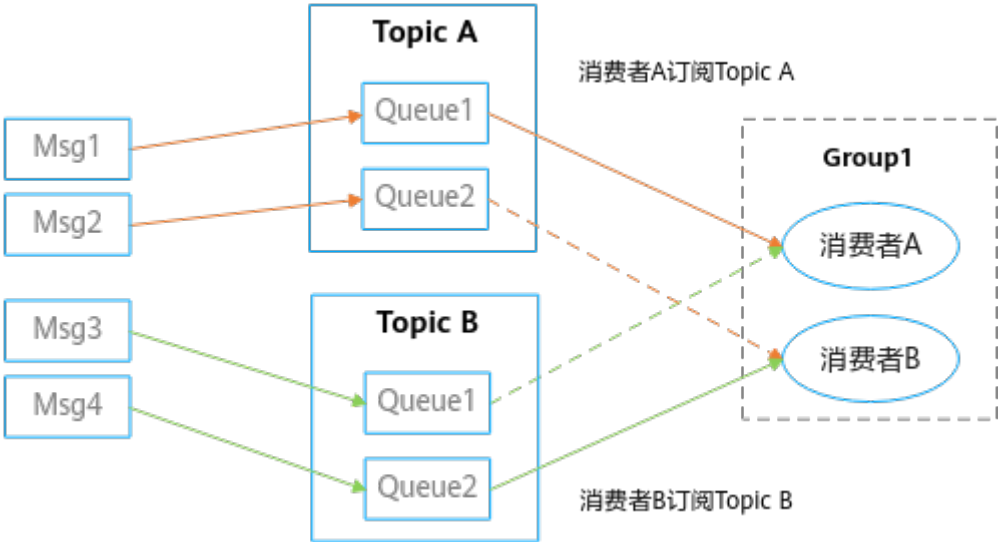


订阅关系一致性保证了同一个消费组中消费消息的正常运行，避免消息逻辑混乱和消息丢失。在实际使用中，生产者端要做好消息的分类，便于消费者可以使用Tag进行消息的准确订阅。而在消费者端，则要保证订阅关系的一致性。

错误的订阅关系

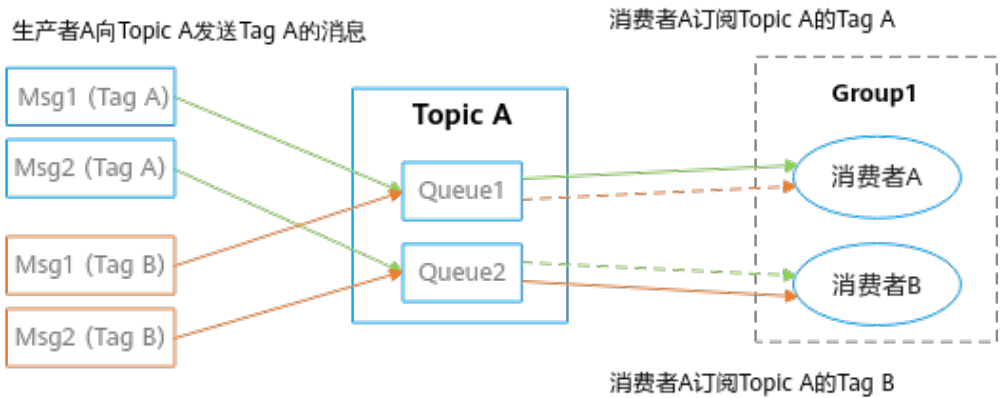
- 同一消费组下的消费者订阅了不同的Topic  
例如，消费组Group1下有消费者A和消费者B，消费者A订阅了Topic A，消费者B订阅了Topic B。当生产者向Topic A发送消息时，消息会按Queue均匀发送给消费者A和消费者B。由于消费者B没有订阅Topic A的消息，会把Topic A消息过滤掉（即图3-2中Topic A的Queue2中的消息会被消费者B过滤），从而导致部分Topic A消息未被消费。

图 3-2 错误的 Topic 订阅



- 同一消费组下的消费者订阅了相同Topic下不同的Tag  
例如，消费组Group1下有消费者A和消费者B，消费者A订阅了Topic A的Tag A，消费者B订阅了Topic A的Tag B。当生产者A向Topic A发送Tag A的消息时，Tag A消息会按Queue均匀发送给消费者A和消费者B。由于消费者B没有订阅Tag A的消息，会把Tag A消息过滤掉（即图3-3中Queue2中的Tag A消息会被消费者B过滤），从而导致部分Tag A消息未被消费。

图 3-3 错误的 Tag 订阅



实施方法

- 订阅一个Topic且订阅一个Tag**

同一消费组Group1中的消费者Consumer1、Consumer2和Consumer3都订阅了Topic\_A，且都订阅了Topic\_A的Tag\_A，符合订阅关系一致原则。Consumer1、Consumer2、Consumer3的订阅关系一致，即Consumer1、Consumer2、Consumer3订阅消息的代码必须完全一致，代码示例如下：

```
DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("Group1");
consumer.subscribe("Topic_A", "Tag_A");
```
- 订阅一个Topic且订阅多个Tag**

同一消费组Group1中的消费者Consumer1、Consumer2和Consumer3都订阅了Topic\_A，且都订阅了Topic\_A的Tag\_A和Tag\_B（即订阅Topic\_A中所有Tag为Tag\_A或Tag\_B的消息），顺序都是Tag\_A||Tag\_B，符合订阅关系一致性原则。Consumer1、Consumer2、Consumer3的订阅关系一致，即Consumer1、Consumer2、Consumer3订阅消息的代码必须完全一致，代码示例如下：

```
DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("Group1");
consumer.subscribe("Topic_A", "Tag_A||Tag_B");
```
- 订阅多个Topic且订阅多个Tag**

同一消费组Group1中的消费者Consumer1、Consumer2和Consumer3都订阅了Topic\_A和Topic\_B，且订阅Topic\_A都未指定Tag（即订阅Topic\_A中的所有消息），订阅Topic\_B的Tag都是Tag\_A和Tag\_B（即订阅Topic\_B中所有Tag为Tag\_A或Tag\_B的消息），顺序都是Tag\_A||Tag\_B，符合订阅关系一致性原则。Consumer1、Consumer2、Consumer3的订阅关系一致，即Consumer1、Consumer2、Consumer3订阅消息的代码必须完全一致，代码示例如下：

```
DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("Group1");
consumer.subscribe("Topic_A", "*");
consumer.subscribe("Topic_B", "Tag_A||Tag_B");
```

# 4 消息堆积处理建议

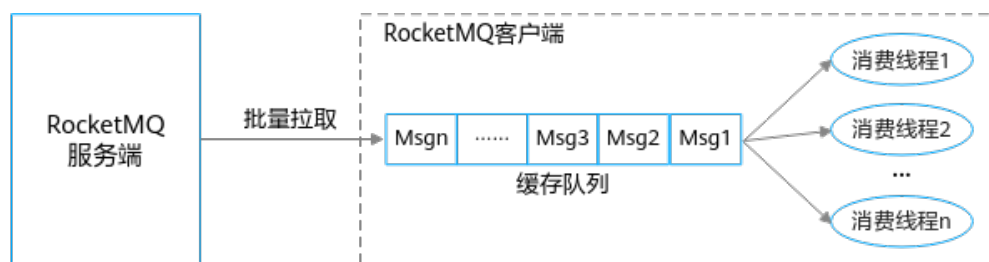
## 方案概述

在RocketMQ的实际业务中，消息堆积是比较常见的问题。在消息处理过程中，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。消息没有被及时消费而产生消息堆积，从而会造成消息消费延迟。对于消息消费实时性要求较高的业务系统，即使是消息堆积造成的短暂消息延迟也无法接受。造成消息堆积的原因有以下两个：

- 消息没有及时被消费，生产者生产消息的速度快于消费者消费消息的速度，从而产生消息积压且无法自行恢复。
- 业务系统本身逻辑耗费时间较长，导致消息消费效率较低。

### 消息消费过程

图 4-1 消息消费的过程



一个完整的消息消费过程主要分为2个阶段：

- 消息拉取  
客户端通过批量拉取的方式从服务端获取消息，将拉取到的消息缓存到本地缓存队列中。对于拉取式消费，在内网环境下的吞吐量很高，因此消息拉取阶段一般不会引起消息堆积。
- 消息消费  
客户端将本地缓存的消息提交到消费线程中，提供给业务消费逻辑进行消息处理，待消息处理完成后获取处理结果。此阶段的消费能力依赖于消息的消费耗时和消费并发度。如果由于业务处理逻辑复杂等原因，导致处理单条消息的耗时较长，就会影响整体的消息吞吐量。而消息吞吐量低会导致客户端本地缓存队列达到上限，从而停止从服务端拉取消息，引起消息堆积。

所以，消息堆积的主要瓶颈在于客户端的消费能力，而消费能力由消费耗时和消费并发度决定。消费耗时的优先级要高于消费并发度，应在保证消费耗时合理性的前提下，再考虑消费并发度问题。

消费耗时

影响消息处理时长的主要因素是业务处理的代码逻辑，而代码逻辑中会影响处理时长的主要有两种代码类型：CPU内部计算型代码和外部I/O操作型代码。如果代码中没有复杂的递归和循环处理，CPU内部计算耗时相对于外部I/O操作耗时来说几乎可以忽略，因此应关注外部I/O操作型代码的消息处理效率。

外部IO操作型代码主要有以下业务操作：

- 读写外部数据库，例如对远程MySQL数据库读写。
- 读写外部缓存系统，例如对远程Redis读写。
- 下游系统调用，例如Dubbo的RPC远程调用，Spring Cloud对下游系统的HTTP接口调用。

提前梳理下游系统的调用逻辑，掌握每个调用操作的预期耗时，有助于判断业务逻辑中I/O操作的耗时是否合理。通常消息堆积都是由于下游系统出现了服务异常或容量限制，从而导致消费耗时增加。而服务异常，并不仅仅是系统出现报错，也可能是更加隐蔽的问题，比如网络带宽问题。

消费并发度

客户端的消费并发度由单客户端线程数和客户端数量决定。单客户端线程数是指单个客户端所包含的线程数量，客户端数量是指消费组所包含的客户端（消费者）数量。对于普通消息、定时/延时消息、事务消息及顺序消息的消费并发度计算方法如下：

| 消息类型    | 消费并发度                   |
|---------|-------------------------|
| 普通消息    | 单客户端线程数 * 客户端数          |
| 定时/延时消息 |                         |
| 事务消息    |                         |
| 顺序消息    | Min(单客户端线程数 * 客户端数，队列数) |

单客户端线程数的调整需谨慎，不能盲目调大线程数，如果设置过大的线程数反而会带来大量的线程切换开销。

理想环境下单客户端的最优线程数计算模型为： $C * (T1+T2) / T1$ 。

其中，C为单机vCPU核数，T1为业务逻辑的CPU计算耗时，T2为外部I/O操作耗时，另外线程切换耗时忽略不计，I/O操作不消耗CPU，线程需有足够消息等待处理且内存充足。

此处最大线程数的计算模型是在理想环境下得到的，在实际应用中建议逐步调大线程数，在观察效果后再进行调整。

实施方法

为了避免在实际业务中出现非预期的消息堆积问题，需要在业务系统的设计阶段梳理业务逻辑的消费耗时和设置消费并发度。

- **梳理消费耗时**

通过压测获取消息的消费耗时，并对耗时较高的操作代码逻辑进行分析和优化。  
梳理消息的消费耗时需要注意以下几点：

- 消息消费逻辑的计算复杂度是否过高，代码是否存在复杂的递归和循环处理。
- 消息消费逻辑中的I/O操作是否是必须，是否可以使用本地缓存等方案规避。
- 消息消费逻辑中的复杂耗时操作是否可以做异步化处理。

- **设置消费并发度**

对于消费并发度的计算，可以按如下方法进行处理：

- a. 根据公式计算出线程数的理想值，然后选取一个比理想值小的数据作为线程数起始值。逐步调大单个客户端的线程数，并观测客户端的系统指标，得到单个客户端的最优消费线程数和消息吞吐量。
- b. 根据上下游链路的流量峰值，计算出需要设置的客户端数量：客户端数=流量峰值/单客户端消息吞吐量。

# 5 DMS for RocketMQ 安全使用建议

安全性是华为云与您的共同责任。华为云负责云服务自身的安全，提供安全的云；作为租户，您需要合理使用云服务提供的安全能力对数据进行保护，安全地使用云。详情请参见[责任共担](#)。

本文提供了如何安全使用DMS for RocketMQ的最佳实践，旨在为提高整体安全能力提供可操作的规范性指导。根据该指导文档您可以持续评估DMS for RocketMQ资源的安全状态，更好的组合使用DMS for RocketMQ提供的多种安全能力，提高对DMS for RocketMQ资源的整体安全防御能力，保护存储在DMS for RocketMQ内的数据不泄露、不被篡改，以及数据在传输过程中不泄露、不被篡改。

本文从以下几个维度给出建议，您可以评估DMS for RocketMQ的使用情况，并根据业务需要在本指导的基础上进行安全配置。

- [通过访问控制，保护数据安全性](#)
- [通过SSL链路传输加密方式访问DMS for RocketMQ](#)
- [不存储敏感数据](#)
- [构建数据的恢复和容灾能力](#)
- [审计是否存在异常数据访问](#)
- [使用最新版本SDK获得更好的操作体验和更强的安全能力](#)

## 通过访问控制，保护数据安全性

1. **建议对不同角色的IAM用户仅设置最小权限，避免权限过大导致数据泄露或被误操作。**  
为了更好的进行权限隔离和管理，建议您配置独立的IAM管理员，授予IAM管理员IAM策略的管理权限。IAM管理员可以根据您业务的实际诉求创建不同的用户组，用户组对应不同的数据访问场景，通过将用户添加到用户组并将IAM策略绑定到对应用户组，IAM管理员可以为不同职能部门的员工按照最小权限原则授予不同的数据访问权限，详情请参见[权限管理](#)。
2. **建议配置安全组访问控制，保护您的数据不被异常读取和操作。**  
参照[表5-1](#)配置安全组的入方向规则限制，控制连接实例的网络范围，避免DMS for RocketMQ暴露给不可信的第三方。

表 5-1 安全组规则

| 实例版本                   | 方向  | 协议  | 端口          | 源地址   | 说明                    |
|------------------------|-----|-----|-------------|---|-----------------------|
| • 4 . 8 . 0<br>• 5 . x | 入方向 | TCP | 8100        | 访问DMS for RocketMQ实例的客户端地址，应避免设置为0.0.0.0/0。 | 使用TCP协议，通过内网访问实例的端口。  |
| • 4 . 8 . 0<br>• 5 . x | 入方向 | TCP | 8200        |   | 使用TCP协议，通过公网访问实例的端口。  |
| • 4 . 8 . 0<br>• 5 . x | 入方向 | TCP | 10100-10199 |   | 使用TCP协议，访问业务节点的端口。    |
| 5.x                    | 入方向 | TCP | 8080        |   | 使用gRPC协议，通过内网访问实例的端口。 |
| 5.x                    | 入方向 | TCP | 8081        |   | 使用gRPC协议，通过公网访问实例的端口。 |

3. 建议将访问DMS for RocketMQ实例方式设置为密码访问（即开启ACL访问控制），防止未经认证的客户端误操作实例。  
您可以选择如下任意一种方法开启ACL访问控制。
  - 在“购买实例”页面开启ACL访问控制，具体可参考[购买RocketMQ实例](#)。
  - 购买实例后，在实例的“基本信息”页面中开启ACL访问控制，具体可参考[查看和修改RocketMQ实例基本信息](#)。
4. 开启敏感操作多因子认证保护您的数据不被误删。

DMS for RocketMQ支持敏感操作保护，开启后执行删除实例等敏感操作时，系统会进行身份验证，进一步对数据的高危操作进行控制，保证DMS for RocketMQ数据的安全性。详情请参见[敏感操作](#)。

## 通过 SSL 链路传输加密方式访问 DMS for RocketMQ

为了确保数据传输过程中不被窃取和破坏，建议使用SSL链路传输加密方式（即开启SSL）访问DMS for RocketMQ。

您需要在“购买实例”页面选择SSL，具体可参考[购买RocketMQ实例](#)。

## 不存储敏感数据

DMS for RocketMQ服务暂不支持数据加密，避免将敏感数据存入消息队列。

## 构建数据的恢复和容灾能力

预先构建数据的容灾和恢复能力，可以有效避免异常数据处理场景下数据被误删、被破坏的问题。

- 建议不要在生产环境使用DMS for RocketMQ 5.x单机实例。
- 建议使用跨可用区复制构建数据容灾能力。

DMS for RocketMQ集群实例支持跨可用区部署，选择多可用区部署实例时，可达到跨可用区复制数据容灾能力。

## 审计是否存在异常数据访问

1. **开启云审计服务，记录DMS for RocketMQ的所有访问操作，便于后期审查。**

云审计服务（Cloud Trace Service，CTS），是华为云安全解决方案中专业的日志审计服务，提供对各种云资源操作记录的收集、存储和查询功能，可用于支撑安全分析、合规审计、资源跟踪和问题定位等常见应用场景。

您开通云审计服务并创建和配置追踪器后，CTS可记录DMS for RocketMQ的管理事件和数据事件用于审计。相关文档请参见[查看RocketMQ审计日志](#)。

2. **使用云监控服务对安全事件进行实时监控和告警。**

为使您更好地掌握DMS for RocketMQ实例状态，华为云提供了云监控服务（Cloud Eye）。您可使用该服务监控自己的DMS for RocketMQ实例，执行自动实时监控、告警和通知操作，帮助您实时掌握DMS for RocketMQ实例中所产生的请求、流量等信息。

云监控服务不需要开通，会在用户创建DMS for RocketMQ实例后自动启动。相关文档请参见[RocketMQ支持的监控指标](#)、[配置RocketMQ监控告警](#)。

## 使用最新版本 SDK 获得更好的操作体验和更强的安全能力

建议您升级SDK并使用最新版本，从客户侧对您的数据和DMS for RocketMQ使用过程提供更好的保护。最新版本SDK在各语言对应界面下载，请参见[RocketMQ SDK](#)。