分布式消息服务RocketMQ

目录

目录

最佳实践

开发指南

性能白皮书

API参考

SDK参考

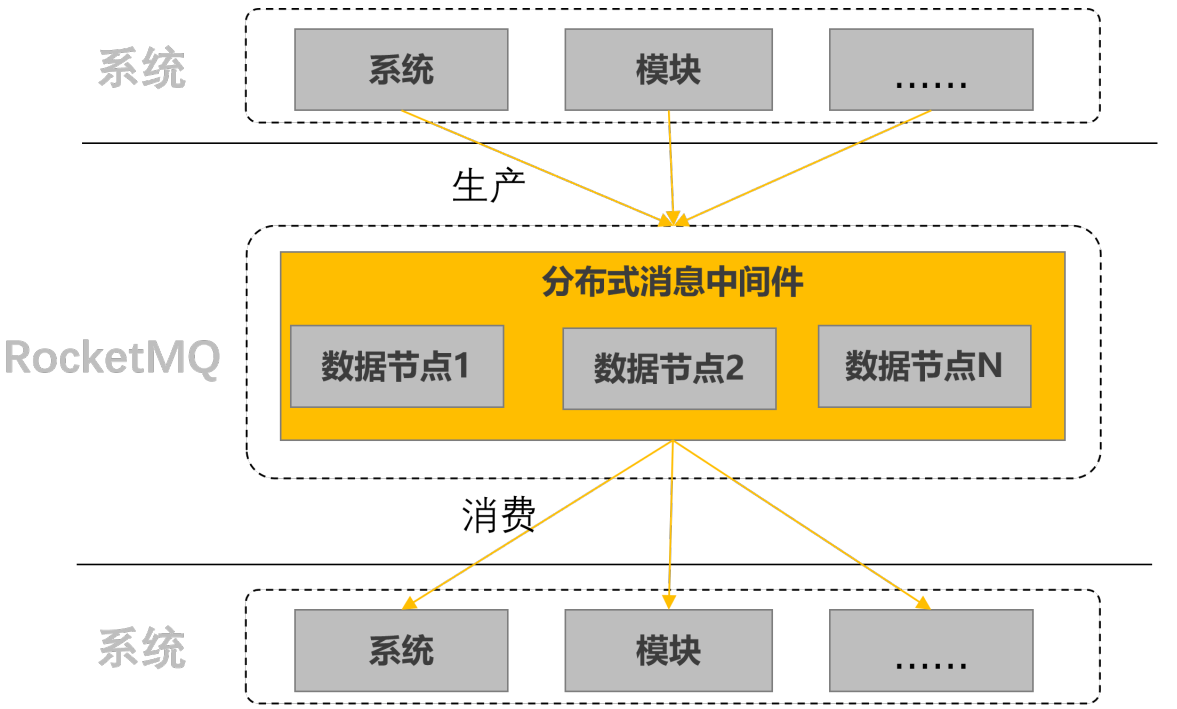
常见问题

目录

产品定义

分布式消息服务RocketMQ是一款低成本、高可靠、高性能的消息中间件产品，兼容开源RocketMQ客户端，提供高效可靠的消息传递服务，解决分布式应用系统之间的消息数据通信难题，用于系统间的解耦，用户只需专注业务，无需部署运维，适用于电商、金融、政企等多样业务场景。

产品示意图



分布式消息服务RocketMQ发布订阅基本流程如下：

1、producer生产者连接nameserver，产生数据放入不同的Topic；

2、对于RocketMQ，一个Topic可以分布在各个Broker上，我们可以把一个Topic分布在一个Broker上的子集定义为一个Topic分片；

3、将Topic分片再切分为若干等分，其中的一份就是一个Queue。每个Topic分片等分的Queue的数量可以不同，由用户在创建Topic时指定；

4、consumer消费者连接nameserver，根据broker分配的Queue来消费数据。

核心组件

分布式消息服务RocketMQ主要由Producer、Broker、Consumer三部分组成，其中Producer负责生产消息，Consumer负责消费消息，Broker负责存储消息。Broker在实际部署过程中对应1台或者多台服务器，每个Broker可以存储多个Topic的消息，每个Topic的消息也可以分片存储于不同的Broker。Message Queue用于存储消息的物理地址，每个Topic中的消息地址存储于多个Message Queue中。ConsumerGroup由多个Consumer 实例构成。更多信息请参见产品架构。

开源对比

相较于开源自建RocketMQ，分布式消息服务RocketMQ在自动化部署、运维监控、增强能力、延迟消息/定时消息、ACL访问控制等方面更具优势。更多信息请参见开源对比。

支持的消息类型

分布式消息服务RocketMQ支持的消息类型包括普通消息、顺序消息、事务消息与延时消息。

- 普通消息：RocketMQ中无特性的消息，普通消息主要包含同步消息和异步消息两种。
- 顺序消息：指消费消息的顺序要同发送消息的顺序一致，在RocketMQ中，主要有两种有序消息：全局有序消息和局部有序消息（又叫普通有序消息、分区有序消息）。
- 事务消息：提供类似X/Open XA的分布式事务功能来确保业务发送方和MQ消息的最终一致性，其本质是通过半消息(prepare消息和commit消息)的方式把分布式事务放在MQ端来处理。
- 延时消息：生产者将消息发送到消息队列RocketMQ服务端，设计消费时延，在预设的时间后才可以被消费者消费。

更多信息请参见功能特性。

功能特性

分布式消息服务RocketMQ的功能特性主要体现在以下几个方面：

访问接口

支持通过API调用，创建队列、查询消息监控指标、查询消息内容等。

队列能力

支持多种消息类型，包括普通队列（高并发场景）、FIFO有序队列（顺序信息场景）、严格有序队列。

消息能力

支持消息过滤、消息复用、消息重试、消息回溯、消息数据主动删除以及消息广播等能力。

安全防护

提供云审计进行租户管理操作的记录。

运维监控

提供主题、订阅组、生产者、消费者、队列的管理；同时支持集群、主题、队列多维度指标监控。

更多信息请参见功能特性。

应用场景

分布式消息服务RocketMQ适用于电商、金融、政企等多样业务场景，通常用于业务的分布式系统异步通信、数据同步和交换以及削峰填谷等场景。更多信息请参见应用场景。

使用限制

分布式消息服务RocketMQ对Topic、Group等对象信息进行限制，使用时注意不要超过限制，以免程序出现异常。更多信息请参见使用限制。

产品优势

分布式消息服务RocketMQ的产品优势主要包括以下几个方面：

堆积并发

支持亿级消息堆积，在海量消息堆积下队列性能影响很小，并可通过队列数扩展，提升系统并发能力。

分布式消息服务RocketMQ具有较强的消息堆积能力，主要以下几个方面实现和保障。

- 消息存储机制：RocketMQ使用基于磁盘的消息存储机制，可以持久化大量的消息到磁盘中，避免消息丢失。
- 消息发送速度：RocketMQ具有高吞吐量和低延迟的特点，可以快速处理大量的消息发送请求。
- 消息消费速度：RocketMQ支持水平扩展和负载均衡机制，可以实现高并发的消息消费能力。
- 存储容量：由于RocketMQ使用磁盘存储消息，因此它的存储容量可以非常大。可以通过添加更多的磁盘来扩展存储容量。

分布式消息服务RocketMQ具有很高的并发能力，RocketMQ的并发能力主要体现于以下几个方面：

- 集群模式：RocketMQ支持将消息队列集群化部署，通过增加Broker节点和消费者实例，可以提高整体系统的并发处理能力。每个Broker节点和消费者实例都可以处理独立的消息流，从而实现并发处理。
- 分区模式：RocketMQ支持将消息划分为多个分区，每个分区可以由不同的Broker节点处理。通过使用分区模式，可以将消息负载均衡到不同的节点上，从而提高并发处理能力。
- 异步发送模式：RocketMQ提供了异步发送消息的方式，发送者将消息发送给Broker后立即返回，无需等待确认。这种方式可以提高发送消息的并发能力，同时也会带来一定的消息丢失的风险。
- 消费者线程数：RocketMQ的消费者可以通过增加线程数来提高并发处理能力。每个线程可以独立地从Broker节点拉取消息并进行处理。需要注意的是，过多的线程数可能会对系统的性能产生负面影响，需要根据实际情况进行调优。

灵活及时

队列处理能力支持按需自动扩展，及时且方便地完成系统扩展，消息投递时间可至毫秒级，从而保证消息及时性。分布式消息服务RocketMQ具有很高的灵活性，可以满足各种不同的业务需求。主要体现：

- 支持多种消息模型：RocketMQ支持多种消息模型，包括消息队列模型和发布/订阅模型。在消息队列模型中，消息发送方将消息发送到一个队列，消息接收方从队列中读取消息。这种模型适用于顺序消息和事务性消息等场景。而在发布/订阅模型中，消息发送方将消息发布到一个主题，所有订阅该主题的消费者都会收到消息。这种模型适用于实时通知、数据分发等场景。
- 支持灵活的消息过滤机制：RocketMQ可以通过对消息的属性进行过滤，只有满足条件的消息才会被消费者接收。这样可以实现消息的动态路由和选择性消费，提升系统的灵活性和效率。
- 支持消息延迟发送和定时消费：RocketMQ可以设置消息的延迟时间，使消息在指定的时间后才能被消费者接收。这对于实现定时任务和延迟处理非常有用。
- 良好的可扩展性：RocketMQ采用了分布式架构，并且支持主从复制和消息分区机制。可以根据业务需求，动态扩展消息生产者、消息消费者和消息存储节点的数量，以满足大规模消息处理和高并发访问的要求。

高可靠

集群节点采用主备模式，具有主备故障自动切换功能；并且提供对消息的持久化能力，多副本冗余；提供消息数据自动删除功能。分布式消息服务RocketMQ具有高可靠性的特点，以下是RocketMQ实现高可靠性的关键特性：

- 主从复制：RocketMQ采用了主从复制的架构，在生产者发送消息时，消息会首先写入主节点，并异步复制到多个从节点。这样即使主节点发生故障，从节点也能够接管并继续提供服务。

- 可靠消息存储：RocketMQ使用Write Ahead Log (WAL)技术来保证消息的可靠存储。在消息写入之前，会先将消息写入磁盘的顺序文件中，然后再写入内存。当RocketMQ重启时，可以通过检查磁盘上的文件来恢复之前未被消费的消息。
- 消息可重复存储：RocketMQ使用消息的唯一ID来确保消息的幂等性。如果一条消息因为网络问题或其他原因发送失败，RocketMQ可以根据消息的ID判断是否已经成功发送过，并避免消息的重复发送。
- 容灾备份：RocketMQ支持Broker集群模式和多数据中心的部署方式，可以将消息数据进行容灾备份。当某个Broker节点发生故障时，其他节点可以继续提供服务，确保系统的可用性。
- 高可用性设计：RocketMQ采用了多个Broker节点组成的集群，并通过主从复制和故障切换来实现高可用性。当某个Broker节点发生故障时，其他节点会自动接管其工作，保证消息的正常处理。

运维管理

提供多维度指标监控（队列级别）；支持消息查询、消息回溯以及消息数据过期自动删除。

功能特性

消息生产

- 消息压缩：将较大的消息进行压缩后发送到服务端，有效利用带宽。
- 延迟消息：设计消费时延，消息发送到服务端后，过了预设时间才可以被消费。
- 事务消息：根据预设的事务，事务消息可保证分布式系统之间的数据最终一致。

消息消费

- 有序消费：支持普通有序消息和严格有序消息两种方式。
- 集群消费：一个主题可被一个或多个消费者组消费，消费者组中消费者实例可平均分摊消费信息。
- 消费位置设置：支持设置消费组首次启动消费的位置，包括队列头、队列尾及由客户端指定。
- 消息回溯：支持按时间回溯消费进度，将订阅组在某主题上的消费进度重置到过去或者未来。

完善的运维能力

- 应用用户管理：集群租户隔离，应用接入集群权限管理。
- 主题管理：支持对实例下的主题进行管理，执行创建删除等操作。
- 订阅组管理：支持对实例下的订阅组进行管理。
- 生产者和消费者管理：用户可查看当前实例下的生产者和消费者信息，并实时更新。
- 消息查询：按消息ID、消息逻辑偏移量、消息key。
- 完善的运维功能，节点状态检测、启停；实例状态检测、启停；SLA监控等。

顺序消息

顺序消息是指消费消息的顺序要同发送消息的顺序一致，在 RocketMQ 中，主要有两种有序消息：全局有序消息和局部有序消息（又称普通有序消息、分区有序消息）。

普通有序消息：在正常情况下可以保证完全的顺序消息，但是一旦发生通信异常造成Broker 重启，队列总数发生变化，哈希取模后定位的队列会变化，因此会产生短暂的消息顺序不一致。如果业务能容忍在集群异常情况（如某个 Broker 宕机或者重启）下消息短暂的乱序，使用普通顺序方式比较合适。

严格有序消息：无论正常异常情况都能保证顺序，但是牺牲了分布式Failover 特性，即 Broker 集群中只要有一台机器不可用，则整个集群都不可用（或者影响hash 值对应队列的使用），服务可用性大大降低。如果服务器部署为同步双写模式，此缺陷可通过备机自动切换为主避免，不过仍然会存在几分钟的服务不可用。

事务消息

消息队列 RocketMQ 提供类似 X/Open XA 的分布事务功能。半事务消息发送后，根据预设的事务进行判断，满足事务的消息将会被服务端确认，不满足的事务的消息不会被服务端接收，从而实现在分布式场景下保障消息生产和本地事务的最终一致性。

半消息： 暂不能投递的消息，发送方已经将消息成功发送到了消息队列服务端，但是服务端未收到生产者对该消息的二次确认，此时该消息被标记成“暂不能投递”状态，处于该种状态下的消息即半消息。

消息回查： 由于网络闪断、生产者应用重启等原因，导致某条事务消息的二次确认丢失，消息队列 RocketMQ服务端通过扫描发现某条消息长期处于“半消息”时，需要主动向消息生产者询问该消息的最终状态（Commit 或是 Rollback），该过程即消息回查。

延时消息

生产者将消息发送到消息队列RocketMQ服务端，设计消费时延，在预设的时间后才可以被消费者消费。发送延时消息时需要设定一个延时时间长度，消息将从当前发送时间点开始延迟固定时间之后才开始投递，实现分布式场景的延时调度触发效果。

广播消费/集群消费

广播消费：在广播消费模式下，一条消息被多个Consumer消费，即使这些Consumer属于同一个ConsumerGroup，消息也会被ConsumerGroup中的每个Consumer都消费一次，广播消费中的ConsumerGroup 概念可以认为在消息划分方面无意义。

集群消费：一个Topic可以被一个或多个ConsumerGroup消费，每个ConsumerGroup有自己独立的消费进度，消费进度是保存在服务端的。 一个ConsumerGroup中的消费者实例可以平均分摊消费消息，做到负载均衡。例如某个Topic有9条消息，其中一个ConsumerGroup有3个不同的消费者实例（可能是3个进程，或者3台机器），那么每个实例只消费其中的3条消息。在此消费模式下，可以做到Point-To-Point的消费，也可以做到JMS里面广播消费，能满足绝大部分场景，推荐使用此消费模式。

消息重试

对于有序消息：有序消息不能跳跃签收，当消费者消费消息失败后，消息队列RocketMQ会自动不断进行消息重试 （每次间隔时间为1秒），此时应用会出现消息消费被阻塞的情况。因此建议使用有序消息时，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。

对于无序消息：消息队列RocketMQ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
1	10秒
2	30秒
3	1分钟
4	2分钟
5	3分钟
6	4分钟
7	5分钟
8	6分钟
9	7分钟

第几次重试	与上次重试的间隔时间
10	8分钟
11	9分钟
12	10分钟
13	20分钟
14	30分钟
15	1小时
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。

消息过滤

消费者订阅了某个Topic后，消息队列RocketMQ会将该主题中的所有消息投递给消费者。若消费者只需要关注部分消息，可通过设置过滤条件在消息队列RocketMQ版服务端进行过滤，只获取到需要关注的消息子集，避免接收到大量无效的消息。

消息过滤主要通过以下几个关键流程实现：

- 生产者：生产者在初始化消息时预先为消息设置一些属性和标签，用于后续消费时指定过滤目标。
- 消费者：消费者在初始化及后续消费流程中向服务端上报需要订阅指定主题的哪些消息，即过滤条件。
- 服务端：消息队列RocketMQ服务端根据消费者上报的过滤条件的表达式进行匹配，将符合条件的消息投递给消费者进行消费。

消息队列RocketMQ支持两种过滤方式：

- 通过Tag进行过滤：生产者在发送消息时，设置消息的Tag标签，消费者通过 Tag标签指定需要消费的信息。
- 通过SQL属性过滤：通过生产者为消息设置的属性（Key）及属性值（Value）进行匹配。生产者在发送消息时可设置多个属性，消费者订阅时可设置SQL语法的过滤表达式过滤多个属性。

应用场景

行业应用

RocketMQ在多个行业中都有广泛的应用。以下是一些典型的行业应用场景：

1. 电子商务：在电子商务行业中，RocketMQ可以用于订单处理、库存管理、支付通知等异步通信和事件驱动的场景。它可以实现订单的可靠传递和处理，同时支持高并发和高可扩展性的需求。
2. 金融服务：在金融服务行业中，RocketMQ可以用于实时交易通知、资金结算、风险控制等关键业务场景。它可以确保交易消息的可靠传递和顺序处理，同时支持高吞吐量和低延迟的要求。
3. 物流与供应链：在物流与供应链行业中，RocketMQ可以用于实时物流跟踪、订单状态更新、库存管理等消息通知和事件驱动的场景。它可以确保供应链各个环节的信息同步和协调，提高物流效率和准确性。
4. 社交媒体：在社交媒体行业中，RocketMQ可以用于实时消息推送、用户关系管理、活动通知等场景。它可以支持大规模用户同时在线的需求，确保消息的低延迟和高可靠性。

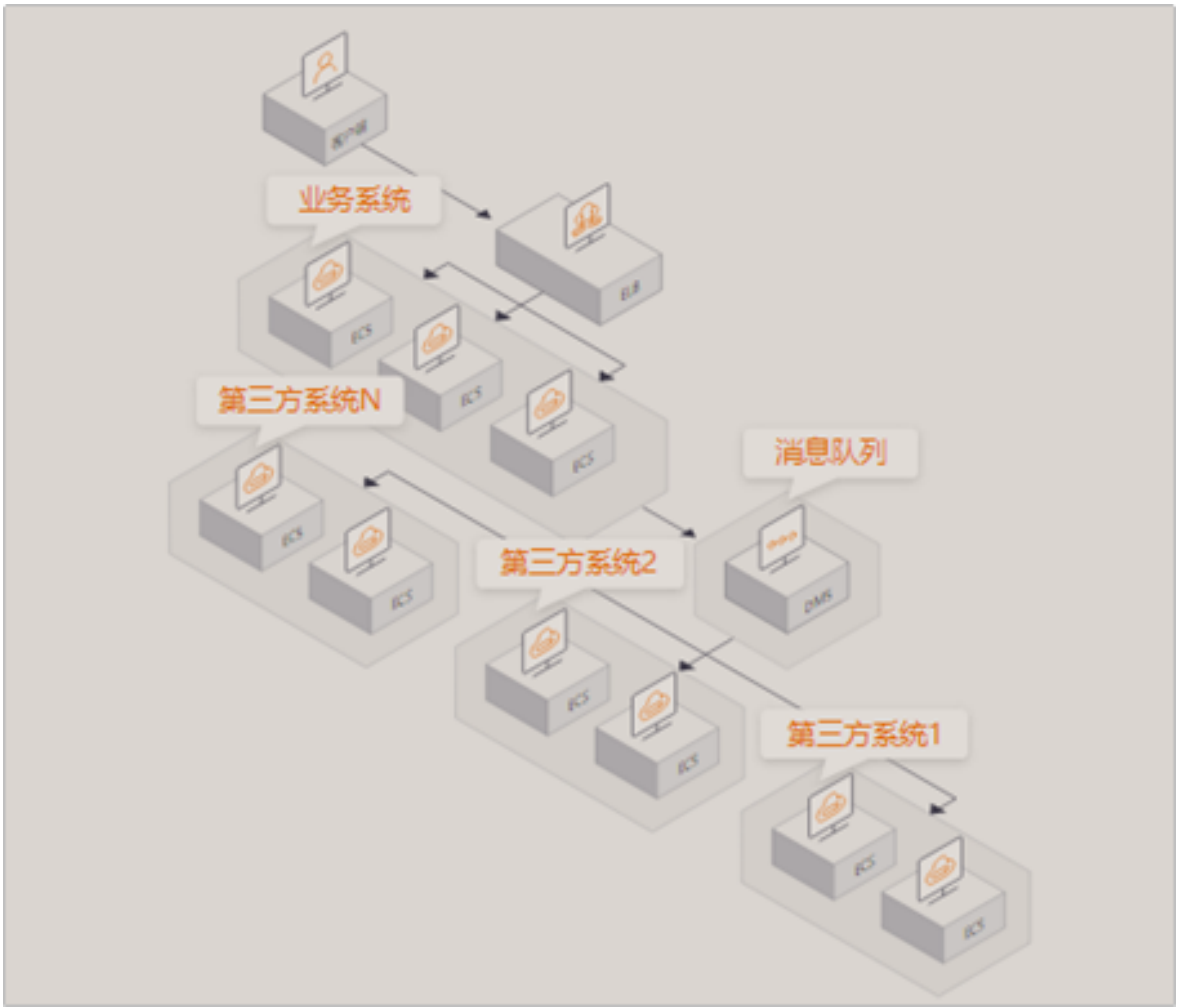
5. 物联网：在物联网行业中，RocketMQ可以用于设备状态监控、告警通知、数据采集等场景。它可以处理大量的设备消息，并支持设备之间的实时通信和协作。

这些是一些典型的RocketMQ行业应用场景，由于RocketMQ的高性能和可靠性，它在更多行业中也有广泛的应用。

分布式消息服务RocketMQ主要适用于以下几种业务场景：

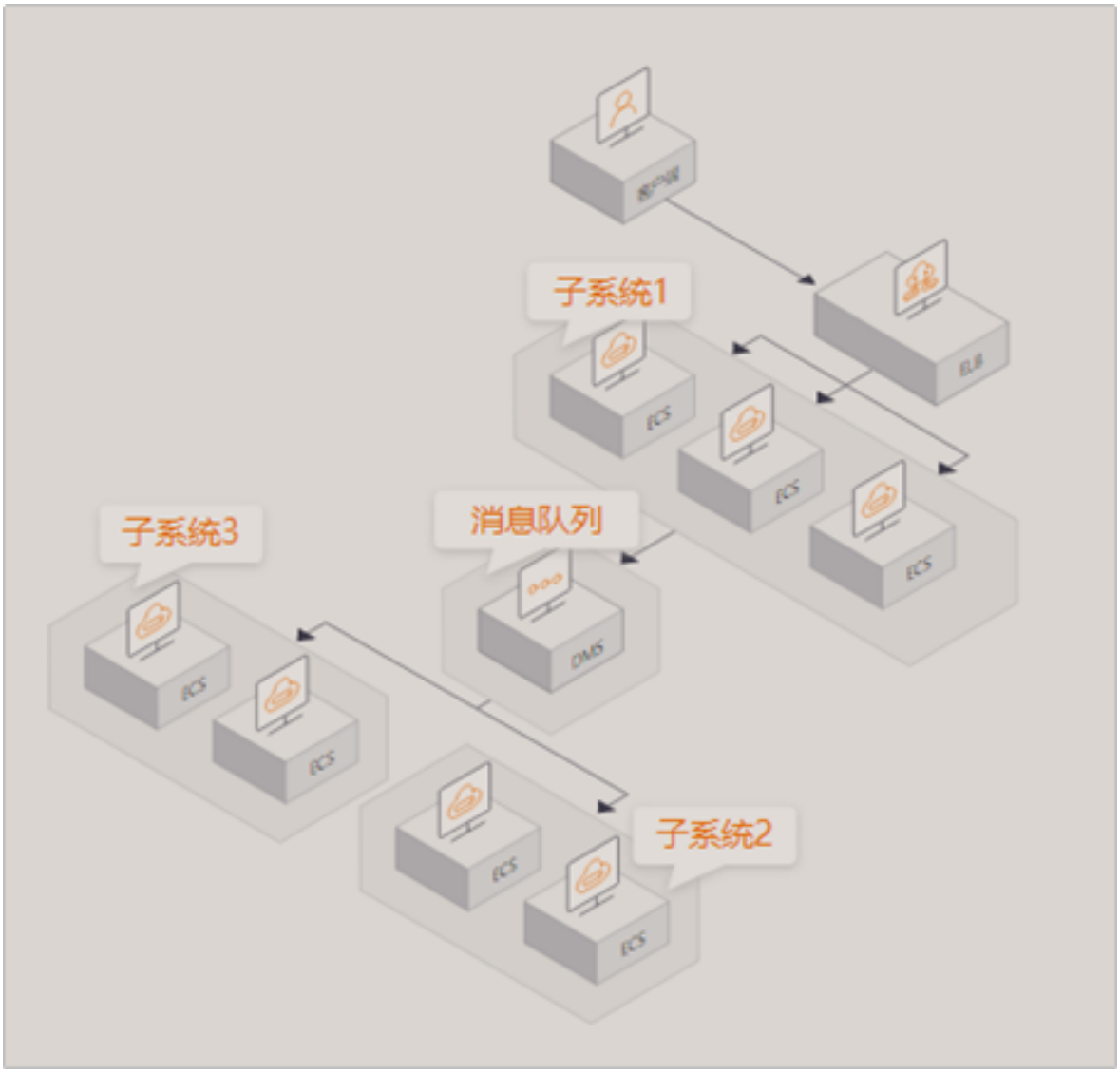
分布式系统异步通信场景

在单体或传统大型单机应用场景中，系统涉及模块众多，接口调用错综繁杂，业务流程耦合导致系统对用户请求响应慢，可以通过将拆分子系统，并用消息队列作为子系统间的异步通信通道进行系统解耦，提升整个系统的响应速度。



数据同步和交换场景

在大中型分布式系统中，各个子系统数据需最终保持一致，比如金融业务场景，需要有可靠消息传递，能保证业务的连续性。分布式消息队列可用于子系统间的高可靠数据传递，实现两者之间数据同步和交换，降低实现难度和成本，并提供数据通道帮助触发其他的业务流程。



削峰填谷场景

在电子商务系统或大型网站，比如大型电商场景，通常会涉及到订单、支付和通知等等场景的业务处理，系统上下游处理能力存在差异，当处理能力高的系统上游突发请求超过系统下游处理能力时，系统对外呈现的服务能力为0。此时可以通过队列服务堆积请求消息，对请求消息实现削峰填谷，错峰处理，避免下游因突发流量崩溃。

产品规格

（1）以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3 节点。

注意

- 通用型规格已调整为白名单特性，如需了解该规格参数请联系技术支持。
- ctgmq引擎已调整为白名单特性，如需了解该引擎请联系技术支持。
- 单机版实例面向用户体验和业务测试场景，无法保证性能和高可用。如果需要在生产环境使用RocketMQ实例，建议购买集群版实例。

天翼云分布式消息服务RocketMQ版产品规格由以下五个维度定义：

- 资源规格：定义使用的弹性云服务器的规格类型。
- 代理个数：即Broker数量，定义实例的规模，天翼云分布式消息RocketMQ每个Broker由一个Master节点（主节点）和一个Slaver节点（备节点）组成，详细见产品架构。
- 存储容量：定义单个代理可以保存的存储容量。
- 单个代理TPS：定义单个代理的TPS性能。
- 单个代理消费组数上限：定义单个代理可以创建的消费组数量。

产品简介

天翼云分布式消息服务RocketMQ支持的产品规格如下所示：

说明

TPS（Transaction per second）是指每秒可以生产消息和消费消息的总次数，可以理解为对应规格每秒生产消息和消费消息的总吞吐量。

Intel计算增强型

资源规格	单个代理存储容量（GB/代理）	单个代理TPS	单个代理消费组数上限
rocketmq.2u4g.cluster	200～20000	15000	2000
rocketmq.4u8g.cluster	200～20000	20000	4000
rocketmq.8u16g.cluster	200～20000	25000	8000
rocketmq.12u24g.cluster	200～20000	28000	12000
rocketmq.16u32g.cluster	200～20000	30000	16000
rocketmq.24u48g.cluster	200～20000	30000	16000
rocketmq.32u64g.cluster	200～20000	30000	16000
rocketmq.48u96g.cluster	200～20000	30000	16000
rocketmq.64u128g.cluster	200～20000	30000	16000

海光计算增强型

资源规格	单个代理存储容量（GB/代理）	单个代理TPS	单个代理消费组数上限
rocketmq.hg.2u4g.cluster	200～20000	10000	2000
rocketmq.hg.4u8g.cluster	200～20000	15000	4000
rocketmq.hg.8u16g.cluster	200～20000	18000	8000
rocketmq.hg.16u32g.cluster	200～20000	21000	16000
rocketmq.hg.32u64g.cluster	200～20000	21000	16000

鲲鹏计算增强型

资源规格	单个代理存储容量（GB/代理）	单个代理TPS	单个代理消费组数上限
rocketmq.kp.2u4g.cluster	200～20000	12000	2000
rocketmq.kp.4u8g.cluster	200～20000	18000	4000
rocketmq.kp.8u16g.cluster	200～20000	21000	8000
rocketmq.kp.16u32g.cluster	200～20000	24000	16000

资源规格	单个代理存储容量（GB/代理）	单个代理TPS	单个代理消费组数上限
rocketmq.kp.32u64g.cluster	200～20000	24000	16000

（2）以下适用于南京3、上海7、重庆2、乌鲁木齐27、石家庄20、内蒙6、晋中、北京5 节点。

天翼云分布式消息服务RocketMQ版产品规格由以下三个维度定义：

- 存储容量：定义实例可以保存的存储数量。
- TPS：定义实例的TPS性能。
- Topic数上限：定义实例可以创建的Topic数量。

天翼云分布式消息服务RocketMQ支持的产品规格如下所示：

资源规格	存储容量（GB）	TPS数上限（条/秒）	Topic上限
基础版	200GB	TPS:5000	50
中级版	500GB	TPS:10000	200
高级版	1000GB	TPS:20000	500

开源对比

分布式消息服务RocketMQ基于开源产品RocketMQ进行问题修复与优化，并自主研发，实现低成本、高可靠、高性能和具备监控运维能力的中间件产品。

使用开源RocketMQ遇到的问题

- 功能不完善：功能比较单一，针对不同应用场景无法有效支持，如消息查询，数据自动删除策略等。
- 可维护性差：缺乏配套监控运维能力，难以迅速发现解决如消息堆积、队列堵塞等问题。
- 可靠性较低：消息服务不提供主备切换能力，存在单点故障，无法保证服务高可用。

分布式消息服务RocketMQ改进点

高可用、高可靠改进：

- 实现自动主备切换、自动拉起功能，保证服务高可用。
- 实现消息删除策略，按不同的场景优先保证服务可用性或者数据安全性。

可维护性改进：

- 实现按生产者、消费者、数据节点、队列4种维度的运行状态监控，方便快速发现问题。
- 实现可视化的监控、配置、管理界面#实现自动化测试，以快速迭代

新增功能：

- 新增消息查询，做到可查可追踪
- 重新封装SDK，简化应用使用，并提供按hash算法实现消息局部有序生产消费。

开源自建对比项

对比项	开源自建	分布式消息服务RocketMQ
自动化部署	不支持自动化部署，需要专业人员自行部署、运维	全托管PaaS，免机器资源一键自动化部署。开箱即用，按需使用，支持弹性扩容
运维监控	缺乏配套的监控运维能力	提供多维度的数据可视化监控，快速定位、处理问题
增强能力	功能单一，无法支持多种场景需求	功能丰富，支持消息重试、消息查询、消息轨迹查询、数据自动清理等定制化功能
延迟消息/定时消息	仅支持18个固定延迟时长，最长延迟2小时	支持秒级的任意延迟时长，最长延迟40天
ACL访问控制	访问控制配置复杂	灵活配置，一键生效

分布式消息产品选型

特性	Kafka	RabbitMQ	RocketMQ
功能	支持功能较少，不支持延迟发送，消息重试等功能	功能丰富，支持多个队列种类（优先级队列、延迟队列、死信队列镜像队列等），提供丰富的策略分配	功能完善，支持事务消息、定时消息、事务消息等
单机吞吐量	十万级	万级	几万级
稳定性	队列/分区多时性能不稳定	消息堆积时，性能不稳定	队列较多、消息堆积时性能保持稳定

特性	Kafka	RabbitMQ	RocketMQ
可用性	非常高（分布式）具有主备故障自动切换	较高，基于主从架构实现高可用性	非常高(分布式)具有主备故障自动切换
选型建议	性能要求高，数据量大，适合产生大量数据的互联网服务的数据收集业务，如日志采集处理、需对接大数据应用等，kafka是首选。	数据量少，吞吐量需求不大；数据可靠性要求较高，对功能丰富性极高	可靠性要求很高且性能要求较高的场景以及业务削峰场景，如电商、订单处理等。

使用限制

参数限制

RocketMQ是一个高性能、高可靠、可伸缩的分布式消息队列系统，但也有一些使用限制需要注意。以下是一些常见的RocketMQ使用限制：

限制项	限制说明
topic名字	限制2到64个字符，超过限制会导致创建主题失败，用户创建主题只能包含大小写字母数字以及_和-符号。
group名字	限制2到64个字符，超过限制会导致创建订阅组失败，用户创建订阅组只能包含大小写字母数字以及_和-符号。
AccessKey	高级版引擎在角色管理中创建AccessKey只能包含大小写字母数组以及_符号，长度限制必须大于6个字符小于64个字符。
SecretKey	高级版引擎角色管理创建SecretKey必须包含大小写字母数字以及以下特殊符号：!@#$%，长度限制必须大于8位小于64个字符。
延时消息的发送时间点	最大支持40天的延时时间点，超过40天将发送失败。
消息大小	普通消息和顺序消息大小限制4MB，延时消息消息大小限制16KB，超过限制会导致消息发送失败。
消息存储时长	消息存储时长限制默认为7天，超过最长存储时间的消息会被滚删除。

资源配额

限制项	限制说明
单地域实例数	实例总数不超过100个。
单代理TPS	由购买的实例规格决定，具体限制值，请参见实例规格限制。
单代理Topic数量	由购买的实例规格决定，具体限制值，请参见实例规格限制。
单代理消费组数量	由购买的实例规格决定，具体限制值，请参见实例规格限制。

安全方案

安全价值

RocketMQ的安全对用户有以下几个重要价值：

1. 保护数据安全：RocketMQ的安全机制可以保护消息的机密性和完整性，防止敏感数据泄露或被篡改。这对于处理包含个人信息、商业机密等敏感数据的应用程序非常重要。
2. 防止未经授权的访问：RocketMQ的访问控制功能可以限制对消息队列的访问权限，只有具有相应权限的用户才能发送和消费消息。这可以防止未经授权的用户访问和操作消息队列，保护系统的安全性。
3. 合规性要求：对于一些行业和法规要求较高的场景，如金融、医疗等，RocketMQ的安全特性可以帮助用户满足合规性要求，确保数据的安全和合规性。
4. 提供安全审计功能：RocketMQ的安全审计功能可以记录和追踪对消息队列的操作，包括发送、消费、订阅等。这可以帮助用户监控和检测潜在的安全风险，及时发现和应对安全事件。
5. 增强用户信任：通过提供安全性能和功能，RocketMQ可以增强用户对系统的信任感。用户可以放心地使用RocketMQ来处理重要的消息传输和处理任务，而不必担心数据的安全问题。

综上所述，RocketMQ的安全性对用户来说具有重要的价值，可以保护数据安全，防止未经授权的访问，满足合规性要求，提供安全审计功能，并增强用户对系统的信任感。

身份认证

- **CTIAM**

  统一身份认证（Identity and Access Management， 简称：CTIAM）是提供用户进行权限管理的基础服务，可以帮助您安全的控制天翼云服务和资源的访问及操作权限，包括：用户身份认证、权限分配、访问控制等功能。具体介绍请参考统一身份认证-产品介绍。

  您可以创建IAM用户，并为其设置关联分布式消息服务RocketMQ实例权限，该用户就可以通过用户名和密码访问授权的实例资源。具体请参见统一身份认证-快速入门-创建IAM用户 。

访问控制

- **权限控制**

  购买分布式消息服务RocketMQ实例之后，您可以使用CTIAM为企业中的员工设置不同的访问权限，以达到不同员工之间的权限隔离，通过CTIAM进行精细的权限管理。

- **VPC和子网**

  虚拟私有云（Virtual Private Cloud，VPC）为分布式消息服务RocketMQ构建隔离、私密的虚拟网络环境，提升数据库的安全性，并简化用户的网络部署。您可以完全掌控自己的专有网络，VPC丰富的功能帮助您灵活管理云上网络，包括创建子网、设置安全组和网络ACL、管理路由表、申请弹性公网IP和带宽等。通过子网与其他网络隔离，独享网络资源，提高网络安全性。具体内容请参见虚拟私有云-用户指南-创建虚拟私有云和子网 。

- **安全组**

  安全组是一个逻辑上的分组，可以为同一个虚拟私有云内具有相同安全保护需求并相互信任的RocketMQ实例提供相同的访问策略。您可以通过为数据库实例设置安全组，开通需访问RocketMQ实例的IP地址和端口，来保证保障其运行环境的安全性和稳定性。具体请参见修改实例安全组

数据保护技术

RocketMQ提供了多种数据保护技术，以确保数据在传输和存储过程中的机密性和完整性。

1. 跨AZ容灾：根据数据和服务的不同可靠性需求，您有多种选择。您可以选择在一个可用区（即单个机房）内部署RocketMQ实例，或者选择跨多个可用区（即同城灾备）进行部署。
2. 副本冗余：通过副本冗余机制，RocketMQ可以提供高可用性和容错性，确保消息的可靠传输和持久存储。即使在节点故障或网络异常的情况下，RocketMQ仍能保证消息的可用性和一致性，提供稳定可靠的消息传输服务。
3. 数据持久化：通过数据持久化机制，RocketMQ可以将消息可靠地存储在磁盘上，并在需要时进行读取和恢复。这样可以确保消息的持久性、可靠性和一致性，提供稳定可靠的消息传输和存储服务。

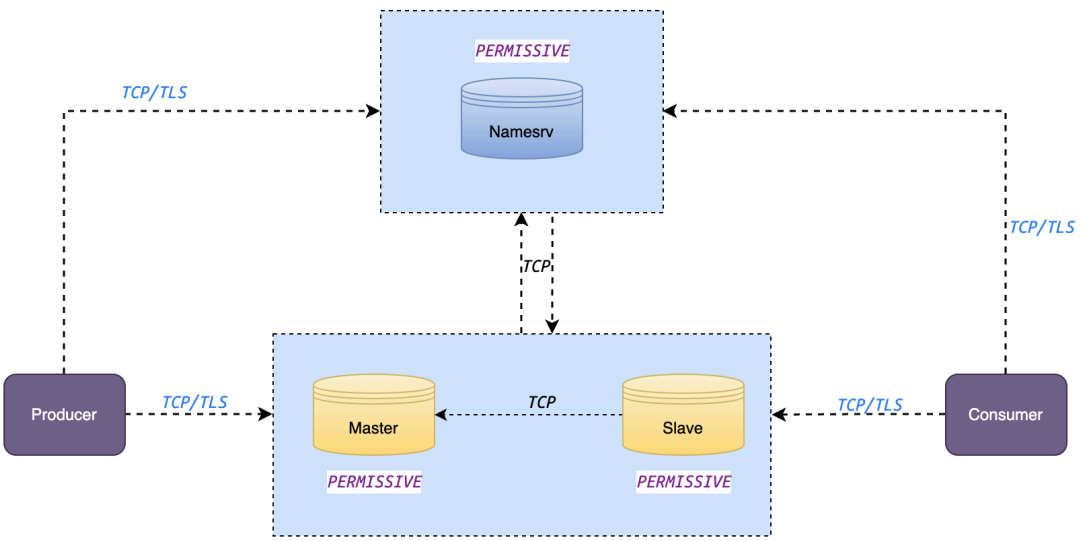
服务韧性

RocketMQ服务的韧性是指其在面对各种故障和异常情况时能够保持可用性和可靠性的能力。以下是保障RocketMQ服务韧性的关键方面：

1. AZ内实例容灾：在一个可用区内部署多个RocketMQ实例，以提供高可用性和容灾能力。当一个实例发生故障或不可用时，其他实例可以接管工作，确保消息服务的连续性。
2. 数据容灾：在RocketMQ中，数据容灾是指保护消息数据免受损失的能力。RocketMQ支持配置多个副本，将消息数据复制到不同的节点上。这样即使某个节点发生故障，其他副本仍然可以提供数据服务，确保消息数据的可用性。

支持TLS传输加密

客户端可以设置TLS或非TLS连接。如下图所示：



权限控制

权限控制主要为 RocketMQ 提供 Topic 资源级别的高级访问控制功能。用户在使用RocketMQ权限控制时，可以在Client客户端注入用户名和密码参数实现签名，服务端通过权限控制参数实现各个资源的权限管理和校验。

Topic资源访问权限控制

对RocketMQ的Topic资源访问权限控制定义主要如下表所示，分为以下四种

权限	含义
DENY	拒绝
ANY	PUB 或者 SUB 权限
PUB	发送权限
SUB	订阅权限

名词解释

Broker

消息中转角色，负责存储消息，转发消息，一般也称Server。在 JMS规范中称为 Provider。RocketMQ一般在多个服务器部署broker集群，从而达到分布式、高可用、可横向扩展的目的。

Name Server

Name Server是一个几乎无状态节点，可集群部署，节点之间无同步信息。它主要提供broker注册、Topic路由管理等功能。

Topic

在RocketMQ中，topic类似于JMS规范中的队列，所有消息都是存放在不同的topic中，生产者与消费者都以topic名字进行生产与消费。(注：RocketMQ的topic并不是jms规范中广播消费topic的概念)。

一个Topic可以存在多个broker之中，这样topic就可以分布在不同的broker从而达到分布式的目的。

一个Topic下面，可以有多个队列，可以理解成分区，Topic的消息是放在不同队列下的。

Queue

在RocketMQ中，queue是存放数据的最小单位，queue是存在于topic下面的。在RocketMQ中，queue不同于JMS规范中的队列，可以理解为topic的分区。

生产组

一类Producer的集合名称，这类Producer通常发送一类消息，且发送逻辑一致，一般由业务系统负责产生消息。

消费组

一类Consumer的集合名称，这类Consumer通常消费一类消息，且消费逻辑一致，一般是后台系统负责异步消费。消费进度由存储在消费组上。

消费者实例

一个消费者实例代表消费组的一员，不同的消费者用不同的实例名字创建。

生产者实例

一个生产者实例代表生产者的一员，不同的生产者用不同的实例名字创建。

PUSH消费

Consumer的一种，应用通常向Consumer对象注册一个Listener接口，一旦收到消息，Consumer对象立刻回调Listener接口方法。在RocketMQ中，客户端会自动起线程消费消息，线程数是5-64，意味着，listener里面的方法，会被多线程执行。客户端内部可以根据堆积量进行调整，使用者不需要新启、管理消费线程。并有流控机制，当客户端缓存一定量消费不及时，会停止推送新消息。

PULL消费

Consumer的一种，应用通常主动调用Consumer的拉消息方法从Broker拉消息，主动权由应用控制，但实时性取决于应用主动拉取的频率。在PULL消费中，线程由应用自主决定。

广播消费

注意：使用消费模式，在很多使用场景都会带来影响或限制，在RocketMQ中，应尽量避免使用此消费模式。在RocketMQ中，消费者有两种不同的方式消费topic中的消息，其中一种是广播消费。在广播消费模式下，一条消息被多个Consumer消费，即使这些Consumer属于同一个Consumer Group，消息也会被Consumer Group中的每个Consumer都消费一次，广播消费中的Consumer Group概念可以认为在消息划分方面无意义。V1.x版本由于广播消费的消费进度，是保存在客户端的，对于很多使用场景会带来影响，在RocketMQ中，并不推荐使用此消费模式。

集群消费

一个Topic可以被一个或多个Consumer Group消费，每个Consumer Group有自己独立的消费进度，消费进度是保存在服务端的。一个Consumer Group中的消费者实例可以平均分摊消费消息，做到负载均衡。例如某个Topic有9条消息，其中一个Consumer Group有3个不同的消费者实例（可能是3个进程，或者3台机器），那么每个实例只消费其中的3条消息。在此消费模式下，可以做到Point-To-Point的消费，也可以做到JMS里面广播消费，能满足绝大部分场景，推荐使用此消费模式。

有序消息

消费消息的顺序要同发送消息的顺序一致，在RocketMQ中，主要有两种有序消息。

普通有序消息

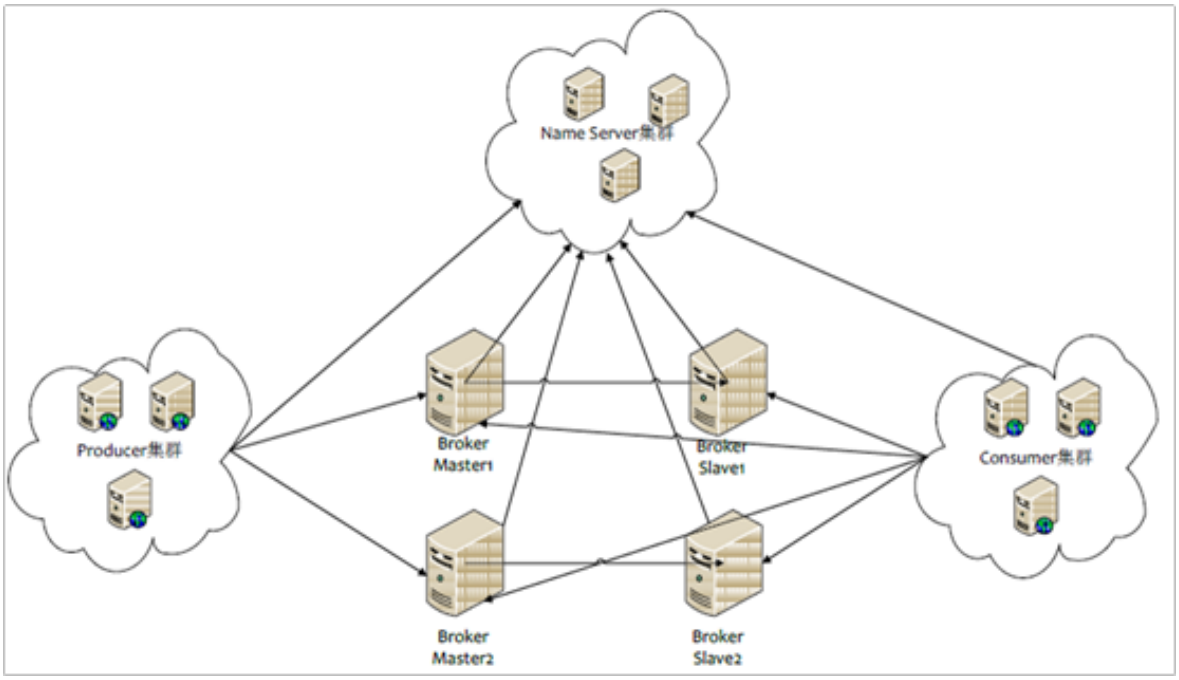
有序消息的一种，在正常情况下可以保证完全的顺序消息，但是一旦发生通信异常，Broker重启，由于队列总数发生变化，哈希取模后定位的队列会变化，产生短暂的消息顺序不一致。如果业务能容忍在集群异常情况（如某个Broker宕机或者重启）下，消息短暂的乱序，使用普通顺序方式比较合适。

严格有序消息

有序消息的一种。无论正常异常情况都能保证顺序，但是牺牲了分布式Failover特性，即Broker集群中只要有一台机器不可用，则整个集群都不可用（或者影响hash值对应队列的使用），服务可用性大大降低。如果服务器部署为同步双写模式，此缺陷可通过备机自动切换为主避免，不过仍然会存在几分钟的服务不可用。若业务能容忍短暂乱序，推荐普通有序消费。

产品架构

系统部署架构如下图所示：



Name Server

Name Server是一个几乎无状态节点，一般集群部署（2个节点或以上），节点之间无同步信息。它主要提供broker注册、Topic路由管理等功能。

Broker

分布式消息中间件核心组件，提供消息生产、消费，主从同步、数据刷盘等核心功能。可以横向扩展、在线扩容以提高集群性能。每个Broker由一个Master节点（主节点）和一个Slaver节点（备节点）组成，与Name Server集群的所有节点建立长连接，并定时注册Topic等信息。

Producer

生产者，一般为应用调用API进行消息生产。Producer 与Name Server集群中的其中一个节点（随机选择）建立长连接，定期从Name Server取Topic路由信息，并向提供Topic服务的Master建立长连接，且定时向Master发送心跳。Producer 完全无状态，可集群部署。

Consumer

消费者，一般为应用调用API进行消息消费。Consumer与Producer一样，与一个Name Server建立长连接并取Topic路由信息。Consumer与提供Topic服务的Master建立长连接，进行消息消费。

与其他服务关系

虚拟私有云(CT-VPC，Virtual Private Cloud)

虚拟私有云为分布式消息服务RocketMQ提供一个逻辑隔离的区域，构建一个安全可靠、 可配置和管理的虚拟网络环境。更多信息请参见虚拟私有云。

弹性云主机（CT-ECS，Elastic Cloud Server）

分布式消息服务RocketMQ订购后，默认按照用户选择的实例规格开通弹性云主机，云主机由 CPU、内存、镜像、云硬盘组成，同时结合VPC、安全组、数据多副本保存等能力，打造一个既高效又可靠安全的计算环境，确保分布式消息服务RocketMQ持久稳定运行。更多信息请参见弹性云主机。

云硬盘（CT-EVS，Elastic Volume Service）

分布式消息服务RocketMQ订购后，默认按照用户选择的存储大小开通云硬盘。云硬盘是一种可弹性扩展的块存储设备，可以为分布式消息服务RocketMQ提供高性能、高可靠的块存储服务。更多信息请参见云硬盘。

产品资费

新资费

说明

分布式消息服务RocketMQ新资费涉及的产品实例支持4.9版本引擎，支持X86和ARM计算CPU架构类型的计算增强型主机，提供集群和单机两种版本规格实例，集群版可选1-16代理数量，单机版默认为1节点。

目前在 华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3 资源池开放订购。

上述资源池实例新购和续订可享受1年83折，2年7折，3年5折优惠。

价格计算公式

分布式消息服务RocketMQ费用由实例费用和存储费用两部分组成，两者单价如下表所示，计费公式为：

- 实例费用=实例规格单价 * 总节点数（NameServer集群固定3节点+代理数量 * 2节点），单机版总节点数量为1。
- 存储费用=存储类型单价 * 代理节点数量（代理数量*2节点） * 单节点存储空间GB大小，单机版代理节点数量为1。

实例规格单价

Intel计算增强型

规格名称	实例单价（单个节点）	
	按需标准价格(元/小时)	包月标准价格(元/月)
rocketmq.2u4g.cluster	0.98	441
rocketmq.4u8g.cluster	2.24	1008
rocketmq.8u16g.cluster	4.86	2187
rocketmq.12u24g.cluster	7.38	3321
rocketmq.16u32g.cluster	9	4050
rocketmq.24u48g.cluster	15.12	6804
rocketmq.32u64g.cluster	20.16	9072
rocketmq.48u96g.cluster	30.24	13608
rocketmq.64u128g.cluster	40.32	18144

海光计算增强型

规格名称	实例单价（单个节点）	
	按需标准价格(元/小时)	包月标准价格(元/月)
rocketmq.hg.2u4g.cluster	1.2152	546.84

rocketmq.hg.4u8g.cluster	2.7776	1249.92
rocketmq.hg.8u16g.cluster	5.3568	2410.56
rocketmq.hg.16u32g.cluster	11.16	5022
rocketmq.hg.32u64g.cluster	24.9984	11249.28

鲲鹏计算增强型

规格名称	实例单价（单个节点）	
	按需标准价格(元/小时)	包月标准价格(元/月)
rocketmq.kp.2u4g.cluster	1.3916	626.22
rocketmq.kp.4u8g.cluster	3.1808	1431.36
rocketmq.kp.8u16g.cluster	6.9012	3105.54
rocketmq.kp.16u32g.cluster	12.78	5751
rocketmq.kp.32u64g.cluster	28.6272	12882.24

存储类型单价

磁盘类型	标准资费（按月） 元/GB/月	标准资费（按天） 元/GB/天	标准资费（按年） 元/GB/年	标准资费（按需） 元/GB/小时
高IO（SAS）	0.4	0.013151	4.8	0.000900
超高IO（SSD）	1.2	0.039452	14.4	0.001700

旧资费

说明

分布式消息服务RocketMQ旧资费根据实例规格分为基础版、中级版和高级版，按照不同版本收费。

目前在 上海7、南京3、乌鲁木齐27、北京5、晋中、内蒙6 资源池开放订购。

实例资费

实例类型	标准资费（元/月）	按需标准资费（元/小时）	实例说明
基础版	1546	3.23	TPS:5000条/秒， Topic数上限：50，存储空间：200GB
中级版	2276	4.75	TPS:10000条/秒， Topic数上限：200，存储空间：500GB
高级版	3792	7.90	TPS:20000条/秒， Topic数上限：500，存储空间：1000GB

计费说明

计费项

4.0版本

以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3 节点。

分布式消息服务RocketMQ费用计算分为两部分，一部分为实例费用，一部分为存储空间费用。

计费项	含义	适用的计费模式
实例费用	用户选择的选择的实例规格*节点数计费，实例规格单价请参考产品资费，节点数为Name Server集群固定3节点+Broker（代理）*2节点。	按需/包周期
存储空间费用	用户选择的实例存储空间计费（每个实例规格您都可以选择高IO、超高IO等多种不同的云硬盘类型以满足您的业务需求）。	按需/包周期

3.0版本

以下适用于南京3、上海7、重庆2、乌鲁木齐27、石家庄20、内蒙6、晋中、北京5 节点。

分布式消息服务RocketMQ按照消息队列的规格版本可分为基础版、中级版和高级版。

计费项：消息队列的规格版本。

计费模式

目前天翼云分布式消息服务RocketMQ提供包周期（包年/包月）、按需2种计费模式供您灵活选择，使用越久越便宜。

下表列出两种模式的区别：

计费模式	包年/包月	按需计费
付费方式	预付费按照订单的购买周期结算。	后付费按照云服务器实际使用时长计费。
计费周期	按订单的购买周期计费。	按小时结算。
实例升级	支持扩容，工单施工完生效，但是施工过程中服务不可用；不支持缩容	支持扩容，工单施工完生效，但是施工过程中服务不可用；不支持缩容
更改计费模式	支持变更为按需资源。	支持变更为包周期资源。注意：目前仅部分资源池支持，支持资源池清单见旧资费-分布式消息服务RocketMQ-计费说明-产品资费 - 天翼云。
变更规格	支持变更实例规格。	支持变更实例规格。
适用场景	适用于可预估资源使用周期的场景，价格比按需计费模式更优惠。对于长期使用者，推荐该方式。	适用于消息资源需求波动的场景，可以随时开通，随时删除。

- 包周期（包年/包月）：天翼云提供包月和包年的购买模式。这种购买方式相对于按需付费则能够提供更大的折扣，对于长期使用者，推荐该方式。包周期计费按照订单的购买周期来进行结算。
- 按需计费：这种购买方式比较灵活，可以即开即停，支持秒级计费。实例从“开通”开启计费到“删除”结束计费，按实际购买时长（精确到秒）计费。

计费模式变更

天翼云分布式消息服务RocketMQ目前支持计费模式变更。

续费、到期与欠费

到期前续费

手动续订：对于包年/包月订购的分布式缓存服务，用户在资源到期前进行续费操作，可以延长原有资源到期时间，避免资源到期后冻结或超过保留期后被系统回收。详细操作请参考费用中心-续订管理-手动续订。

自动续订：自动续订仅针对采用包月、包年计费模式的资源，详细操作请参考费用中心-续订管理-自动续订。

到期处理

到期后，分布式消息服务RocketMQ进入保留期，您将不能正常访问及使用天翼云分布式消息服务RocketMQ服务，已开通的实例资源将予以保留。

- 若您在到期后15天内续费，自资源续订解冻开始，计算新的服务有效期，按照新的服务有效期计算费用；
- 若到期15天后您仍未续费，RocketMQ实例资源将被释放

欠费原因

在按需计费的模式下帐号的余额不足。

按需欠费资源冻结规则

欠费后，资源进入保留期，您将不能正常访问及使用分布式消息服务RocketMQ，已开通的实例资源将予以保留。

- 若您在保留期内充值，充值后系统会自动扣减欠费金额。
- 若保留期到期您仍未充值，RocketMQ实例资源将被释放。

查看欠费账单

欠费后，可以查看欠费详情或者消费账单，具体请参见文档费用中心-账单管理-查看消费账单。

充值

为防止相关资源不被停止或者释放，请及时进行充值，为避免帐号将进入欠费状态，需要在约定时间内支付欠款，详细操作请参考费用中心-资金管理-余额充值。

退订

针对天翼云分布式消息服务RocketMQ退订操作，具体请参考退订流程。

卡券使用

天翼云分布式消息服务RocketMQ产品卡券使用遵循天翼云统一规则。

计费说明

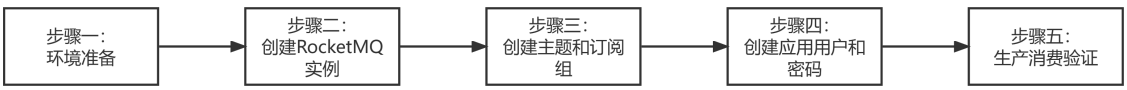
- 优惠券使用
- 代金券使用

卡券使用常见问题：卡券管理-费用中心-常见问题 - 天翼云。

入门指引

本章节将为您介绍分布式消息服务RocketMQ入门的基本流程，主要包括环境准备、控制台创建RocketMQ实例、创建Topic以及生成消费验证，帮助您快速上手RocketMQ。

操作流程



步骤说明

操作流程如下：

1.环境准备

创建RocketMQ实例先要准备好虚拟私有云、子网和安全组。

2.创建RocketMQ实例

在订购分布式消息RocketMQ填写和确认实例名称、引擎类型、计费模式等信息，确认费用后点击下一步，等待开通流程结果通知成功后完成创建实例。

3.创建主题和订阅组

开通实例后，在控制台相关页面按照指引创建主题和订阅组，用于发送和接收消息。

4.创建应用用户和分配权限

创建应用用户并分配主题和订阅组权限，实现各应用共享或者独占消息队列的效果。

5.生产消费验证

以上工作完成后，通过控制台拨测功能生成生产和消费测试数据，用户应用也可按照规范接入RocketMQ，发送、消费消息。

环境准备

概述

在创建天翼云分布式消息服务RocketMQ实例之前，您需要做一些准备工作。

首先，您需要设置一个虚拟私有云（VPC），这是一个隔离的网络环境，用于托管RocketMQ实例。

接下来，您需要创建一个子网，它是VPC内部的一个子网络，用于划分不同的部分和区域。

最后，您需要配置一个安全组，用于控制入站和出站的流量规则，以保证RocketMQ实例的安全性。

每个分布式消息服务RocketMQ实例都会被部署在特定的VPC中，并与特定的子网和安全组相关联。这种方式可以让您自主配置和管理RocketMQ实例的网络环境，并提供安全保护策略。如果您已经有了现成的VPC、子网和安全组，可以重复使用它们，无需重复创建。这样可以节省时间和资源，并确保一致性和可靠性。

VPC和子网

VPC和子网可重复使用，您也可以使用不同的VPC和子网来配置RocketMQ实例，您可根据实际需要进行配置。在创建VPC和子网时应注意如下要求：

- VPC与使用的天翼云分布式消息服务RocketMQ服务应在相同的区域。
- 如无特殊需求，创建VPC和子网的配置参数使用默认配置即可。

创建VPC和子网的操作请参考虚拟私有云-创建VPC、子网搭建私有网络。

若需要在已有VPC上创建和使用新的子网，请参考虚拟私有云-子网管理-创建子网。

要注意的是：

- 如果用户需要创建ipv4实例，则VPC子网只需配置ipv4子网。

- 如果用户需要创建ipv6实例，则vpc子网只需配置ipv4/ipv6双栈子网。

安全组

安全组可重复使用，您也可以根据实际情况使用不同的安全组，请根据实际需要进行配置。

创建安全组的操作指导，请参考虚拟私有云-创建安全组。

若需要为安全组添加规则，请参考虚拟私有云-安全组-添加安全组规则。

弹性云主机

用户若需要自己客户应用接入RocketMQ发送、消费消息，需先购买弹性云主机并确保和RocketMQ实例在同一VPC下。创建操作说明请参见创建弹性云主机。

其他工具

下载安装工具Eclipse3.6.0以上版本 或者IntelliJ，JDK 1.8.111以上版本。

生产消费验证涉及的SDK如下：

- rocketmq引擎版本：推荐使用的社区版Java SDK版本为4.9.3，请访问Apache RocketMQ官网下载。
- ctgmq引擎版本（已调整白名单特性）：点击ctgmq-SDK直接下载。

创建RocketMQ实例

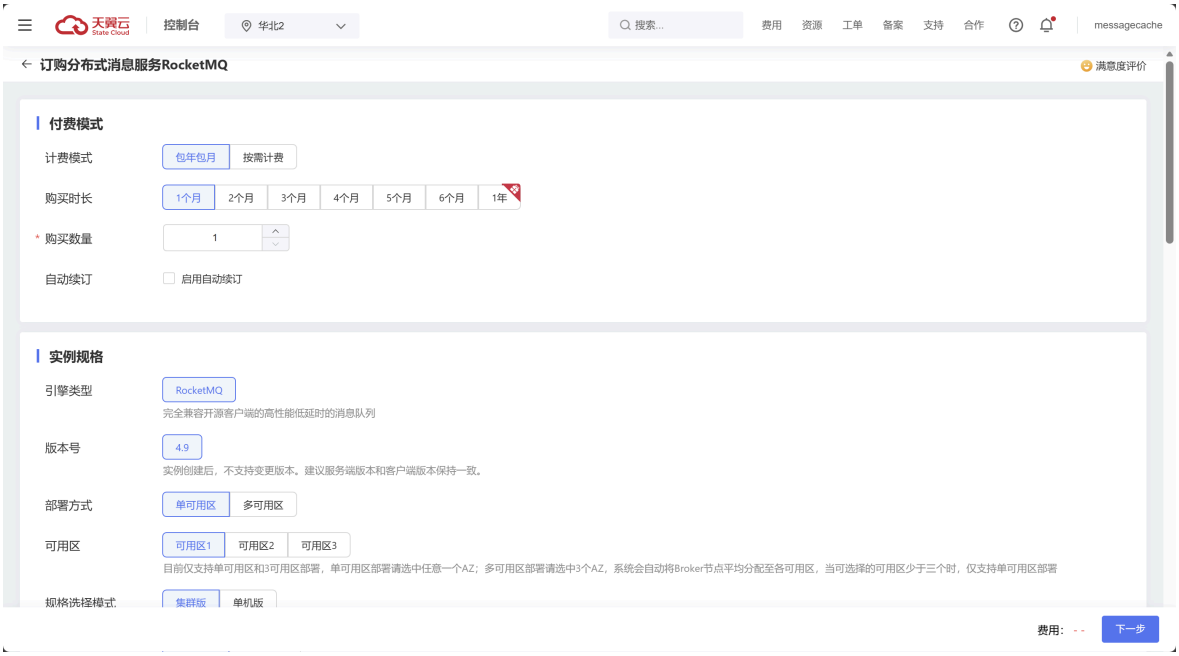
实例介绍

RocketMQ实例订购支持用户自定义规格和自定义特性，可根据业务需要可定制相应规格的RocketMQ实例。在新的资源池节点上，还支持选择主机类型和存储规格等丰富用户选项。

操作步骤

1、在产品详情页点击立即开通按钮，或者进入消息管理控制台创建实例，进入订购分布式消息RocketMQ页面。

2、点击订购实例进入相应页面，左上角选择目标资源池。



1）选择计费模式：包年包月/按需计费，两种模式说明参见计费模式。

2）购买时长按照计费模式选择变化：

- 计费模式为包年包月，可选择购买时长1-6个月、1年。该模式提供自动续期功能。
- 计费模式为按需计费，则该选项隐藏无需选择。

3）引擎类型目前默认选择RocketMQ引擎，完全兼容开源客户端的高性能低延时的消息队列。ctgmq引擎已调整为白名单特性，如需了解该引擎请联系技术支持。

4）版本号默认4.9，实例创建后，不支持变更版本。建议服务端版本和客户端版本保持一致。

5）部署方式有单可用区和多可用区两个选项，目前仅支持单可用区和3可用区部署，单可用区部署请选中任意一个AZ；多可用区部署请选中3个AZ，系统会自动将Broker节点平均分配至各可用区。

6）提供集群版和单机版两种规格选择模式，单机版实例面向用户体验和业务测试场景，无法保证性能和高可用。如果需要在生产环境使用RocketMQ实例，建议购买集群版实例。

7）CPU架构支持X86计算和ARM计算两类。

8）选择X86计算，可以选择计算增强型（默认Intel）和海光-计算增强型主机；选择ARM计算，可以选择鲲鹏-计算增强型主机。通用型规格已调整为白名单特性，如需了解该规格参数请联系技术支持。

9）代理规格：针对不同主机类型，RocketMQ提供不同性能表现的规格参数。

10）代理数量：选择单机版该选项不展示，选择集群版此选择支持1-16代理选择。

11）选择存储空间，包括磁盘类型和空间。

- 磁盘类型提供高IO/超高IO两类。高IO：适用于主流的高性能、高可靠应用场景。超高IO：适用于超高IOPS、超大带宽需求的读写密集型应用场景。了解更多磁盘类型说明参见云硬盘规格。
- 磁盘空间以100G起步，可以以100倍数增加磁盘空间。

12）选择已有虚拟私有云，若无虚拟私有云，点击创建跳转到虚拟私有云页面新增，了解更多内容参见虚拟私有云。

13）选择已有子网，若无子网，点击创建跳转到子网页面新增。

14）选择已有安全组，若无安全组，点击创建跳转到安全组页面新增。

15）填写实例名称，系统默认实例名称，用户可直接修改。

16）选择企业项目，创建新的企业项目可以到IAM配置。

17）创建实例标签，标签由区分大小写的键值对组成，您最多可以设置 20 个标签

3、 填写完上述信息后，单击“下一步”，进入费用确认页面。

4、 确认实例信息无误后，提交请求。

5、 在实例列表页面，查看RocketMQ实例是否创建成功。创建实例大约需要3到15分钟，此时实例状态为“创建中”。

创建主题和订阅组

背景信息

在实例创建完成后，需要创建主题和订阅组来进行消息实例的日常功能运转。

- 主题：在RocketMQ中，主题（Topic）是消息发布的逻辑分组。它类似于一个消息的分类或者标签，帮助用户将不同类型的消息进行归类和管理。通常情况下，一个主题可以包含多个消息生产者和多个消息消费者。通过使用主题，RocketMQ能够实现高效的消息发布和订阅机制，帮助用户更好地管理和组织消息。
- 订阅组：订阅组是 RocketMQ 中的一个重要概念，用于实现消息的发布与订阅模式。一个订阅组可以包含多个消费者实例，这些实例共同消费同一个主题下的消息。当消息被发送到主题时，订阅组中的每个消费者实例会按照一定规则来均衡地接收消息，并进行相应的处理。订阅组是 RocketMQ 提供的一种灵活且可扩展的方式，用于实现消息的发布与订阅模式，并保证消息在消费者之间的均衡分配和可靠处理。
- 集群：RocketMQ集群是一种由多个节点（或者称为Broker）组成的分布式消息中间件系统。每个节点都具有相同的功能并且可以处理和存储消息。通过将消息分发到不同的节点，RocketMQ集群能够实现高可用性和可伸缩性。通过使用RocketMQ集群，可以实现消息传输的并行处理、容错性和高可用性，满足高并发场景下的消息传递需求。

创建主题

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、 进入主题管理菜单，点击【新建主题】按钮

5、 在弹出的新建主题页面，填写如下字段信息

1）默认展示当前集群名称，不可修改。

2）选择主题所在的Broker，按照实例创建时候选择的主备节点对数列出每个broker，可复选。

3）填写主题名称，名字限制2到64个字符，超过限制会导致创建主题失败，用户创建主题只能包含大小写字母数字以及_和-符号。

4）按照实际需求填写主题备注。

5）填写每个Broker分区数，分区数必须大于0，小于等于8，创建严格顺序队列时，设置分区数为1，且只能选择一个broker。

6）选择生产模式，RocketMQ是一个开源的分布式消息中间件，它支持两种消息生产模式：有序和无序。

- 有序消息生产模式（Ordered Message）是指按照特定规则将消息发送到相同的Message Queue中，并且确保消息在消费者端按照相同的顺序进行消费。这种模式适用于那些需要严格按照消息顺序进行处理的场景，比如订单处理、流程审批等。
- 无序消息生产模式（Unordered Message）是指消息发送到不同的Message Queue中，每个Queue都是独立的。消费者可以并行地从多个Queue中消费消息，而无需关心消息的顺序。这种模式适用于那些不需要严格按照消息顺序处理的场景，比如日志收集、异步通知等。

需要注意的是，无论是有序还是无序消息生产模式，RocketMQ都提供了高可靠性的消息传输和存储，并支持水平扩展和高吞吐量的特性。根据具体的业务需求，选择适合的消息生产模式能够更好地满足应用的要求。

7）选择主题的读写权限，支持读写、只读、只写3类权限。

6、 完成主题信息填写后，保存确认即可新增主题。

7、 若希望批量创建主题，可点击【批量创建】按钮

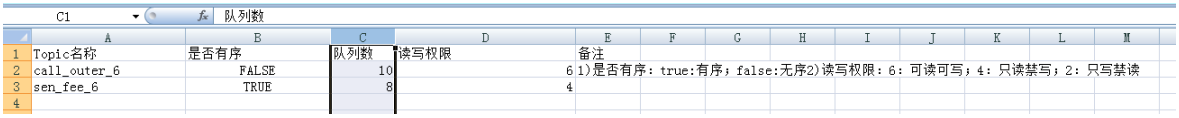
- 批量创建

注意：输入的主题名不要带空格等特殊字符。

通过上传csv文件,批量创建主题。格式：点击【主题模板】按钮下载。

- 主题模板

批量上传主题的模板，必须使用模板，才能够上传成功，模板格式如下：



	A	B	C	D	E
1	Topic名称	是否有序	队列数	读写权限	备注
2	call_outer_6	FALSE	10	6	1)是否有序：true:有序；false:无序2)读写权限：6：可读可写；4：只读禁写；2：只写禁读
3	sen_fee_6	TRUE	8	4	
4					

创建订阅组

1、进入订阅管理菜单，点击【新建】按钮

2、 在弹出的新建订阅组窗口填写相应字段。

1）默认展示当前集群名称，不可修改。

2）填写订阅组名称，名字限制2到64个字符，超过限制会导致创建订阅组失败，用户创建订阅组只能包含大小写字母数字以及_和-符号。

3）按照实际需求填写主题备注。

4）选择订阅组所在的Broker，按照实例创建时候选择的主备节点对数列出每个broker，可复选。

5）选择是否开启消费，默认开启消费。

3、 完成订阅组信息填写后，保存确认即可新增订阅组。

4、 若希望批量创建订阅组，可点击【批量创建】按钮

- 批量创建

注意：注意输入的订阅组名不要带空格等特殊字符

通过csv格式模板上传，批量创建订阅组。格式：点击【订阅组模板】按钮下载。

- 订阅组模板

指批量上传订阅组的模板，必须使用模板，才能够上传成功，模板格式如下：

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	订阅组名	消费机制	消费起始位	topic权限	备注								
2	test_subgroup_e_1	1	1	*	1)消费机制：1： 由客户端保存消费进度,原生v1版本；2：由服务端通过BDB保存消费进度，V2版								
3	test_subgroup_f_1	2	2	topicA									
4													

创建应用用户和密码

背景信息

新建消息实例后，必须新建应用用户，然后应用才能在此消息实例上发送、消费消息。

- 应用用户：指MQ客户端，连接服务器生产消费时，需要进行权限校验，所以MQ客户端的用户，称为应用用户；除了用户密码的校验，还可以为用户指定topic，代表该用户只能生产消费指定的topic，其他topic不能生产消费。

操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、进入应用用户管理（旧模式）或者角色控制（新模式）菜单。

（1）旧模式维护用户及权限

以下适用于上海6、北京4、内蒙5、西安3、重庆2、拉萨3、南京3、雄安2、晋中、郴州2、成都4、杭州2、上海7、西安4、福州3、泉州1、芜湖2、北京5、中卫2、贵州3、九江、内蒙6、武汉4、佛山3、福州4、昆明2、保定、乌鲁木齐27、辽阳1节点。

1）点击【新建用户】按钮

2）进入用户列表界面，新增用户

3）弹出框填写用户字段

- 默认展示租户名，不可修改。
- 选择集群名称，填写应用用户名，请输入大于6位字符，只能输入大小写字母，下划线，数字。
- 填写用户密码，请输入大于8位字符，需要包含数字大小写字母以及特殊符号(!@#$%^&*)。
- 按照实际需求填写描述。

4）设置用户主题或订阅组权限

点击“主题权限”或“订阅组”，可以设置该用户的主题或订阅组发布或订阅权限：

5）选择主题名称及对应权限，PUB代表生产权限 SUB代表消费权限，DENY代表无任何权限，用|符号相连即表示两种权限都有 如PUB|SUB。

6）选择订阅组名称及对应权限，权限说明同上。

（2）新模式维护用户及权限

以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3节点。

1）点击【新建用户】按钮

2）弹出框填写用户字段

- 填写应用用户名，请输入大于6位字符,只能输入大小写字母，下划线，数字。
- 填写密钥，请输入大于8位字符,需要包含数字大小写字母以及特殊符号(!@#$%^&*)。
- 选择默认主题权限，PUB代表生产权限 SUB代表消费权限，DENY代表无任何权限，用|符号相连即表示两种权限都有 如PUB|SUB。
- 选择默认订阅组权限，权限说明同上。

生产消费验证

背景信息

RocketMQ的生产消费验证是指在使用RocketMQ进行消息生产和消费时的验证过程。具体而言，验证包括以下几个方面：

- 生产者验证：RocketMQ提供了丰富的生产者API，开发人员可以使用这些API将消息发送到RocketMQ的消息队列中。在验证阶段，可以通过发送消息并检查返回结果来确保消息成功发送到Broker节点。此外，生产者还应该验证消息的顺序性、事务性以及可靠性等方面。
- 消费者验证：RocketMQ的消费者可以订阅特定的消息主题，从而消费这些主题下的消息。在验证阶段，消费者应该能够正确地从Broker节点拉取消息并进行消费处理。消费者还可以验证消息的顺序性、重试机制以及消息过滤等功能。

操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

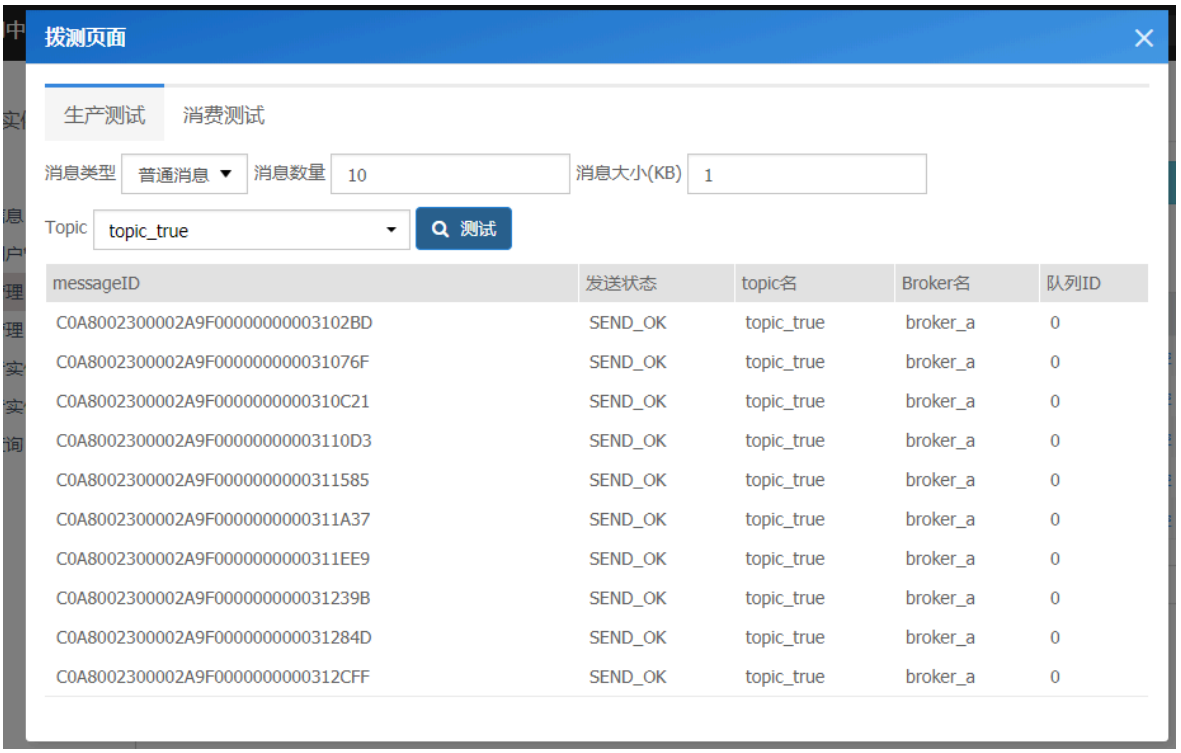
2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

进入实例列表，点击【管理】按钮进入管理菜单。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、 进入主题管理菜单，点击【拨测】按钮，进行生产消费的拨测验证，验证开通的消息实例和主题。

1）生产测试拨测：



- 选择消息类型，默认普通消息。
- 填写需要产生的测试消息数量，以及每条消息的大小，默认每条消息1KB，建议不超过4MB(4096KB)。
- 选择已建的消息主题，若无选项，请新增主题，详见上文创建主题和订阅组。
- 点击【测试】按钮，按照已填写规格及数量产生测试消息数据，展示消息数据的信息，包括消息ID(messageID)、发送状态、主题名（topic名）、Broker名、队列ID。

拨测功能涉及消息发送状态码，以下是RocketMQ消息发送状态码及其说明：

✧ SEND_OK（发送成功）：表示消息成功发送到了消息服务器。

◇ FLUSH_DISK_TIMEOUT（刷新磁盘超时）：表示消息已经成功发送到消息服务器，但是刷新到磁盘上超时。这可能会导致消息服务器在宕机后，尚未持久化到磁盘上的数据丢失。

◇ FLUSH_SLAVE_TIMEOUT（刷新从服务器超时）：表示消息已经成功发送到消息服务器，但是刷新到从服务器上超时。这可能会导致主从同步不一致。

◇ SLAVE_NOT_AVAILABLE（从服务器不可用）：表示消息已经成功发送到消息服务器，但是从服务器不可用。这可能是由于网络故障或从服务器宕机引起的。

◇ UNKNOWN_ERROR（未知错误）：表示发送消息时遇到了未知的错误。一般情况下建议重试发送消息。

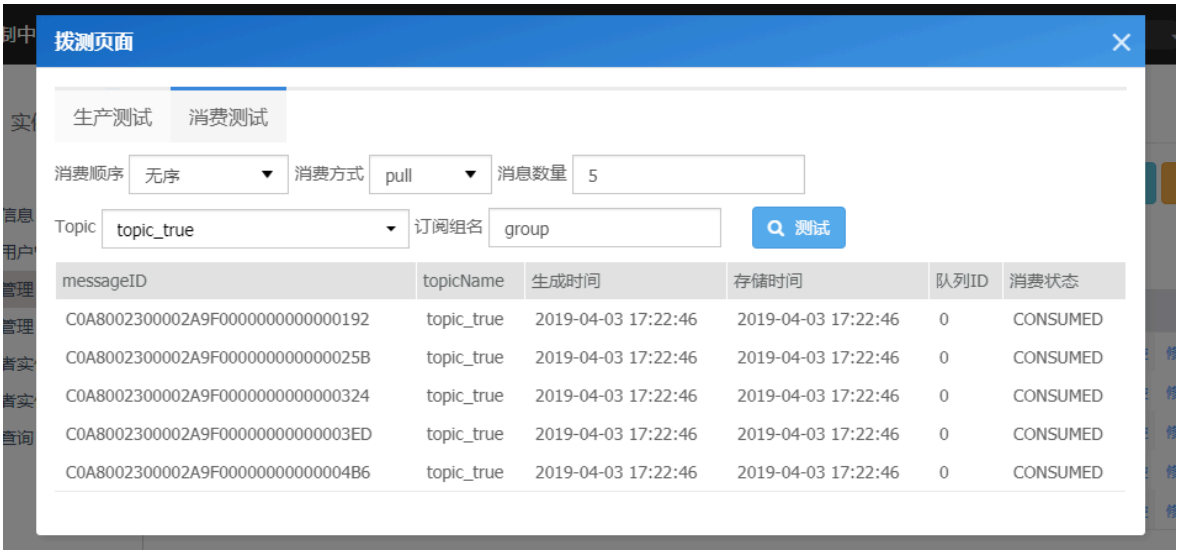
◇ MESSAGE_SIZE_EXCEEDED（消息大小超过限制）：表示消息的大小超过了消息服务器的限制。需要检查消息的大小是否合适。

◇ PRODUCE_THROTTLE（消息生产被限流）：表示消息生产者的频率超出了消息服务器的限制。这可能是由于消息发送频率过高引起的。

◇ SERVICE_NOT_AVAILABLE（服务不可用）：表示消息服务器不可用。这可能是由于网络故障或者消息服务器宕机引起的。

请注意，以上状态码仅适用于RocketMQ消息发送阶段，并且并不代表消息是否成功被消费者接收。同时，这些状态码也可能因版本变化而有所不同，建议查阅官方文档获取最新信息。

2）消费测试拨测：



- 选择消息顺序，下拉选择无序/有序，默认选项为无序。

RocketMQ是一种开源的分布式消息中间件，它支持有序消息和无序消息。

◇ 有序消息是指消息的消费顺序与发送顺序完全一致。在某些业务场景下，消息的处理需要保证顺序性，例如订单的处理或者任务的执行。RocketMQ提供了有序消息的支持，通过指定消息的顺序属性或使用消息队列的分区机制，可以确保消息按照指定的顺序进行消费。

◇ 无序消息则是指消息的消费顺序与发送顺序无关。无序消息的特点是高吞吐量和低延迟，适用于一些不要求严格顺序的业务场景，如日志收集等。

在RocketMQ中，有序消息和无序消息的实现方式略有不同。有序消息需要借助MessageQueue的分区机制和消费者端的顺序消息消费来实现。而无序消息则是通过消息的发送和接收的并发处理来实现的。

总的来说，RocketMQ既支持有序消息也支持无序消息，根据业务需求选择合适的消息类型来满足业务的要求。

- 选择消费方式，目前仅提供pull方式。值得注意的是，RocketMQ还提供了推送（push）方式的消费模式，其中消息队列服务器会主动将消息推送给消费者。但在当前仅限于pull方式的消费模式。
- 填写消费数量。
- 下拉选择选择已建的消息主题和订阅组，若无选项，请新增主题和订阅组，详见上文创建主题和订阅组。
- 点击【测试】按钮，按照已填写规格及数量产生消费数据，展示消息数据的信息，包括消息ID(messageID)、主题名称（topicName）、生成时间、存储时间、队列ID、消费状态。

拨测功能涉及消息消费状态码，RocketMQ消费状态码是指在消息消费过程中，对消费结果进行标识的状态码。以下是常见的RocketMQ消费状态码：

✧ CONSUME_SUCCESS（消费成功）：表示消息成功被消费。

✧ RECONSUME_LATER（稍后重试）：表示消费失败，需要稍后再次进行消费。

✧ CONSUME_FAILURE（消费失败）：表示消息消费出现异常或失败。

✧ SLAVE_NOT_AVAILABLE（从节点不可用）：表示消费者无法访问从节点来消费消息。

✧ NO_MATCHED_MESSAGE（无匹配的消息）：表示当前没有匹配的消息需要消费。

✧ OFFSET_ILLEGAL（偏移量非法）：表示消费的偏移量参数不合法。

✧ BROKER_TIMEOUT（Broker超时）：表示由于Broker超时导致消费失败。

5、 用户应用按照规范接入RocketMQ，发送、消费消息。

1）生产者示例API

以下适用于南京3、上海7、重庆2、乌鲁木齐27、保定、石家庄20、内蒙6、晋中、北京5节点。

--ctgmq引擎版本，SDK下载方式详见环境准备-其他工具章节。

```
package com.ctg.guide;
import com.ctg.mq.api.CTGMQFactory;
import com.ctg.mq.api.IMQProducer;
import com.ctg.mq.api.PropertyKeyConst;
import com.ctg.mq.api.bean.MQMessage;
import com.ctg.mq.api.bean.MQSendResult;
import com.ctg.mq.api.exception.MQException;
import com.ctg.mq.api.exception.MQProducerException;
import java.util.Properties;
/**
 * Producer，发送消息
 */
public class Producer {
   public static void main(String[] args) throws InterruptedException, MQException {
Properties properties = new Properties();
properties.setProperty(PropertyKeyConst.ProducerGroupName, "producer_group");
properties.setProperty(PropertyKeyConst.NamesrvAddr,
 "10.50.208.1:9876;10.50.208.2:9876;10.50.208.3:9876");
```

```
properties.setProperty(PropertyKeyConst.NamesrvAuthID, "app4test");
properties.setProperty(PropertyKeyConst.NamesrvAuthPwd, "******"); properties.setProperty(
PropertyKeyConst.ClusterName, "defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID, "defaultMQTenantID");
    IMQProducer producer = CTGMQFactory.createProducer(properties);//建议应用启动时创建
    int connectResult = producer.connect();
    if(connectResult != 0){
      return;
    }
    for (int i = 0; i < 10; i++) {
      try {
        MQMessage message = new MQMessage(
            "test_topic_1", // topic
            "ORDER_KEY_"+i, // key
            "ORDER_TAG", //tag
            ("HELLO ORDER BODY" + i).getBytes()// body
        );
        MQSendResult sendResult = producer.send(message);
        //System.out.println(sendResult);
        //TODO
      } catch (MQProducerException e) {
        //TODO
      }
    }
    producer.close();//建议应用关闭时关闭
  }
```

以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3节点。

--rocketmq引擎版本，SDK下载方式详见环境准备-其他工具章节。

```
importorg.apache.rocketmq.client.exception.MQClientException;
importorg.apache.rocketmq.client.producer.DefaultMQProducer;
importorg.apache.rocketmq.client.producer.SendResult;
importorg.apache.rocketmq.common.message.Message;
importorg.apache.rocketmq.remoting.common.RemotingHelper;
importorg.apache.rocketmq.acl.common.AclClientRPCHook;


publicclassProducer{
publicstaticvoidmain(String[] args)throwsMQClientException,InterruptedException{

AclClientRPCHook rpcHook =newAclClientRPCHook(
newSessionCredentials(ACCESS_KEY, SECRET_KEY));
DefaultMQProducer producer =newDefaultMQProducer("ProducerGroupName", rpcHook);
// 填入元数据地址
    producer.setNamesrvAddr("192.168.0.1:9876");
```

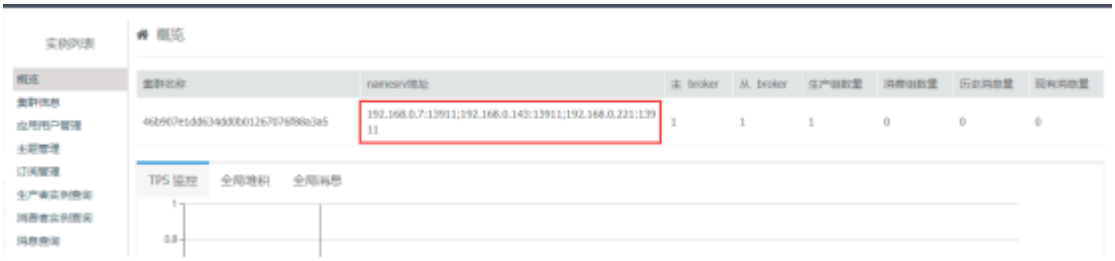
```
//producer.setUseTLS(true);   //如果需要开启SSL，请增加此行代码
      producer.start();

for(int i =0; i <128; i++)
try{
{
Message msg =newMessage("TopicTest",
"TagA",
"OrderID188",
"Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
SendResult sendResult = producer.send(msg);
System.out.printf("%s%n", sendResult);
}

}catch(Exception e){
          e.printStackTrace();
}

      producer.shutdown();
}
}
```

示例参数说明：

Namesrv地址

```
properties.setProperty(PropertyKeyConst.NamesrvAddr,    "
10.50.208.1:9021;10.50.208.2:9021;10.50.208.3:9021");↓
```

Namesrv地址可从控制台查看，多个地址按分号分隔：



应用用户和密码

```
properties.setProperty(PropertyKeyConst.NamesrvAuthID,
"app4test");↓
properties.setProperty(PropertyKeyConst.NamesrvAuthPwd,
"*******");
```

这个应用用户和密码就是控制台创建的应用用户和密码。

租户id和集群名

```
properties.setProperty(PropertyKeyConst.ClusterName,
"defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID,
"defaultMQTenantID");
```

集群名和租户id可以从应用用户管理查询：

生产组

```
properties.setProperty(PropertyKeyConst.ProducerGroupNa
me, "producer_group");
```

生产组名不需要提前创建，只需创建生产者时候配置，服务端会自动创建。建议按业务规划好生产组名，严禁按随机方式生成生产组名。

6.消费者示例API

以下适用于南京3、上海7、重庆2、乌鲁木齐27、保定、石家庄20、内蒙6、晋中、北京5节点。

--ctgmq引擎版本，SDK下载方式详见环境准备-其他工具章节。

```
package com.ctg.guide;

import com.ctg.mq.api.enums.MQConsumeFromWhere;
import com.ctg.mq.api.CTGMQFactory;
import com.ctg.mq.api.IMQPushConsumer;
import com.ctg.mq.api.PropertyKeyConst;
import com.ctg.mq.api.bean.MQResult;
import com.ctg.mq.api.listener.ConsumerTopicListener;
import com.ctg.mq.api.listener.ConsumerTopicStatus;

import java.util.List;
import java.util.Properties;

public class PushConsumer {

    public static void main(String[] args) throws Exception {
        final Properties properties = new Properties();
        properties.setProperty(PropertyKeyConst.ConsumerGroupName, "test_consumer_1");
```

```
    properties.setProperty(PropertyKeyConst.NamesrvAddr,
"10.50.208.1:9876;10.50.208.2:9876;10.50.208.3:9876");
    properties.setProperty(PropertyKeyConst.NamesrvAuthID, "app4test");
    properties.setProperty(PropertyKeyConst.NamesrvAuthPwd, "******"); properties.setPrope
rty(PropertyKeyConst.ClusterName, "defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID, "defaultMQTenantID");


    IMQPushConsumer consumer = CTGMQFactory.createPushConsumer(properties);
    int connectResult = consumer.connect();
    if (connectResult != 0) {
      return;
    }
    consumer.listenTopic("test_topic_1", null, new ConsumerTopicListener() {
      @Override
      public ConsumerTopicStatus onMessage(List<MQResult> mqResultList) {
        //mqResultList 默认为1，可通过批量消费数量设置
        for(MQResult result : mqResultList) {
          //TODO
          System.out.println(result);
        }
        return ConsumerTopicStatus.CONSUME_SUCCESS;//对消息批量确认(成功)
        //return ConsumerTopicStatus.RECONSUME_LATER;//对消息批量确认(失败)
      }
    });
  }
}
```

以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3节点。

--rocketmq引擎版本，SDK下载方式详见环境准备-其他工具章节。

```
importorg.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
importorg.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
importorg.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
importorg.apache.rocketmq.client.exception.MQClientException;

publicclassPushConsumer{

publicstaticvoidmain(String[] args)throwsException{
AclClientRPCHook rpcHook =newAclClientRPCHook(
newSessionCredentials(ACCESS_KEY, SECRET_KEY));
DefaultMQPushConsumer consumer =newDefaultMQPushConsumer(rpcHook);
  consumer.setConsumerGroup("ConsumerGroupName");
// 填入元数据地址
    consumer.setNamesrvAddr("192.168.0.1:9876");
//consumer.setUseTLS(true);  //如果需要开启SSL，请增加此行代码
```

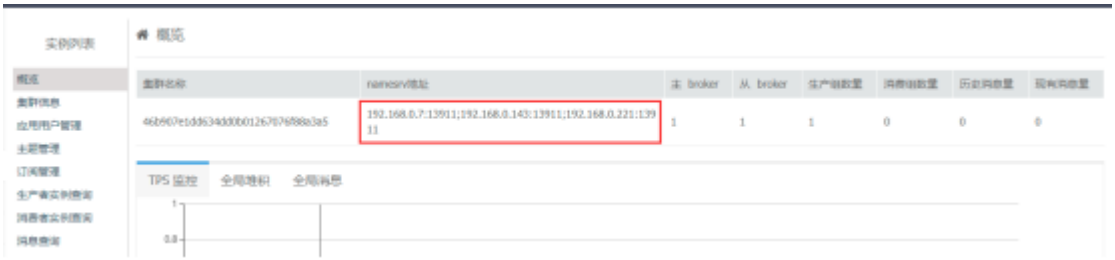
```
        consumer.subscribe("TopicTest", "*");
        consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().getName(), msgs);
returnConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
        consumer.start();
System.out.printf("Consumer Started.%n");
}
}
```

示例参数说明：

Namesrv地址

```
properties.setProperty(PropertyKeyConst.NamesrvAddr,    "
10.50.208.1:9021;10.50.208.2:9021;10.50.208.3:9021");↓
```

Namesrv地址可从控制台查看，多个地址按分号分隔。



应用用户和密码

```
properties.setProperty(PropertyKeyConst.NamesrvAuthID,
"app4test");↓
properties.setProperty(PropertyKeyConst.NamesrvAuthPwd,
"******");
```

这个应用用户和密码就是控制台创建的应用用户和密码。

租户id和集群名

```
properties.setProperty(PropertyKeyConst.ClusterName,
"defaultMQBrokerCluster");↵
properties.setProperty(PropertyKeyConst.TenantID,
"defaultMQTenantID");↵
```

集群名和租户id可以从应用用户管理查询。

订阅组

```
properties.setProperty(PropertyKeyConst.ConsumerGroupName, "test_consumer_1");↓
```

订阅组名需要在控制台提前创建好。

创建实例

订购前准备

在创建天翼云分布式消息服务RocketMQ实例之前，您需要做一些准备工作。

首先，您需要设置一个虚拟私有云（VPC），这是一个隔离的网络环境，用于托管RocketMQ实例。

接下来，您需要创建一个子网，它是VPC内部的一个子网络，用于划分不同的部分和区域。

最后，您需要配置一个安全组，用于控制入站和出站的流量规则，以保证RocketMQ实例的安全性。

每个分布式消息服务RocketMQ实例都会被部署在特定的VPC中，并与特定的子网和安全组相关联。这种方式可以让您自主配置和管理RocketMQ实例的网络环境，并提供安全保护策略。如果您已经有了现成的VPC、子网和安全组，可以重复使用它们，无需重复创建。这样可以节省时间和资源，并确保一致性和可靠性。

VPC和子网

VPC和子网可重复使用，您也可以使用不同的VPC和子网来配置RocketMQ实例，您可根据实际需要进行配置。在创建VPC和子网时应注意如下要求：

- VPC与使用的天翼云分布式消息服务RocketMQ服务应在相同的区域。
- 如无特殊需求，创建VPC和子网的配置参数使用默认配置即可。

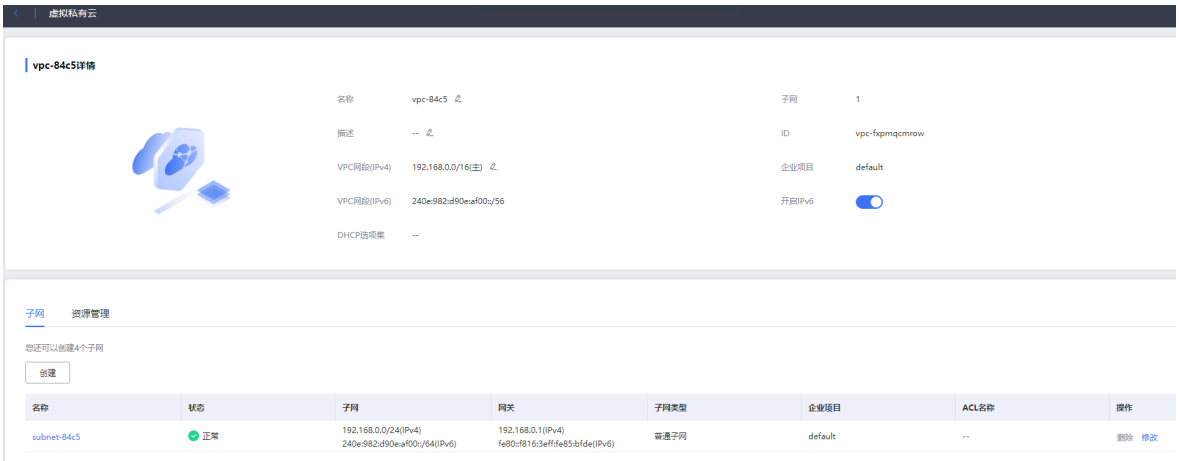
创建VPC和子网的操作请参考虚拟私有云-创建VPC、子网搭建私有网络。

若需要在已有VPC上创建和使用新的子网，请参考虚拟私有云-子网管理-创建子网。

要注意的是：

- 如果用户需要创建ipv4实例，则VPC子网只需配置ipv4子网。

- 如果用户需要创建ipv6实例，则vpc子网只需配置ipv4/ipv6双栈子网。



安全组

安全组可重复使用，您也可以根据实际情况使用不同的安全组，请根据实际需要进行配置。

创建安全组的操作指导，请参考虚拟私有云-创建安全组。

若需要为安全组添加规则，请参考虚拟私有云-安全组-添加安全组规则。

弹性云主机

用户若需要自己客户应用接入RocketMQ发送、消费消息，需先购买弹性云主机并确保和RocketMQ实例在同一VPC下。创建操作说明请参见创建弹性云主机。

订购实例

实例介绍

RocketMQ实例订购支持用户自定义规格和自定义特性，采用物理隔离的方式部署。租户独占RocketMQ实例，可根据业务需要可定制相应规格的RocketMQ实例。在新的资源池节点上，还支持选择主机类型和存储规格等丰富用户选项。

操作步骤

1、在产品详情页点击立即开通按钮，或者进入消息管理控制台创建实例，进入订购分布式消息RocketMQ页面。

2、点击创建实例进入对应页面，左上角选择目标资源池。

1）填写实例名称，系统默认实例名称，用户可直接修改。

2）选择引擎类型，目前默认选择RocketMQ引擎，完全兼容开源客户端的高性能低延时的消息队列。

3）选择计费模式：包年包月/按需计费，两种模式说明参见计费模式。

4）购买时长按照计费模式选择变化：

- 计费模式为包年包月，可选择购买时长1-6个月、1年。该模式提供自动续期功能，勾选后可以自动续期购买时长：1-6个月、1年。
- 计费模式为按需计费，则该选项隐藏无需选择。

5）部署方式有单可用区和多可用区两个选项，目前仅支持单可用区和3可用区部署，单可用区部署请选中任意一个AZ；多可用区部署请选中3个AZ，系统会自动将Broker节点平均分配至各可用区。

6）设置主备节点对数，可输入1~16。主备节点通常是指主题（Topic）的生产者和消费者之间的角色切换。当主节点发生故障或不可用时，备节点可以自动接管并继续提供服务。这种主备节点的设计可以确保系统的稳定性和可靠性。

7）主机类型为计算增强型。计算增强型云主机独享宿主机的CPU资源，实例间无CPU争抢，并且没有进行资源超配，同时搭载全新网络加速引擎，实现接近物理服务器的强劲稳定性能。

8）选择实例规格，分布式消息服务RocketMQ提供计算增强型2C4G、4C8G等一系列规格，各规格详细说明参见弹性云主机规格。

9）选择存储空间，包括磁盘类型和空间。

- 磁盘类型提供高IO/超高IO。高IO：适用于主流的高性能、高可靠应用场景。超高IO：适用于超高IOPS、超大带宽需求的读写密集型应用场景。了解更多磁盘类型说明参见云硬盘规格。
- 磁盘空间以100G起步，可以以100倍数增加磁盘空间。

10）选择已有虚拟私有云，若无虚拟私有云，点击创建跳转到虚拟私有云页面新增，了解更多内容参见虚拟私有云。

11）选择已有子网，若无子网，点击创建跳转到子网页面新增。

12）选择已有安全组，若无安全组，点击创建跳转到安全组页面新增。

3、 填写完上述信息后，单击“下一步”，进入费用确认页面。

4、 确认实例信息无误后，提交请求。

5、 在实例列表页面，查看RocketMQ实例是否创建成功。创建实例大约需要3到15分钟，此时实例状态为“创建中”。

实例管理

查看实例

场景说明

RocketMQ实例是指在RocketMQ消息队列系统中的一个独立运行环境，它包含了独立的配置信息、Topic和消费者组等。一个RocketMQ实例可以独立运行，具有自己的存储和处理能力，能够接收、存储和传递消息。每个RocketMQ实例可以配置多个Topic，每个Topi可以有多个消费者组进行消费。通过配置实例可以实现不同业务场景之间的隔离和并行处理。

创建实例成功后，登录管理台，首页展示该用户可以管理的实例列表，如下图所示。用户可以对该实例列表进行管理。

实例列表

1、 进入控制台查看已购买的实例列表，若列表为空，可点击右上角【创建实例】进入购买页面，创建实例详情见具体操作步骤。

2、 支持按照实例ID和实例名称查询，下拉选择查询字段，输入查询内容，点击【查询】按钮即可展示需要的实例数据。同时提供刷新按钮，人工同步后台最新的实例数据。

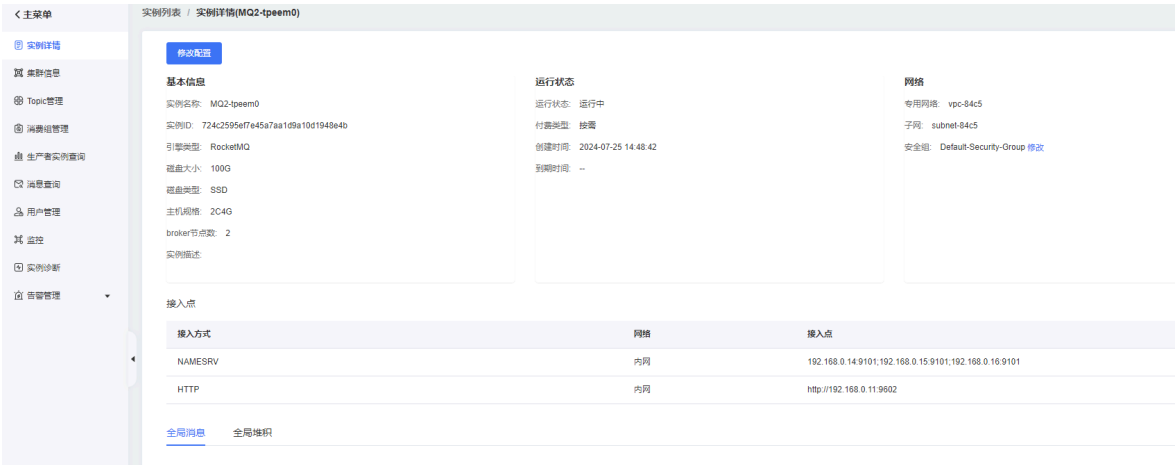
3、 查看实例基本信息，包括实例名称、实例ID、状态、引擎、计费模式、创建时间、到期时间。其中实例状态说明如下表所示：

状态	说明
运行中	RocketMQ实例正常运行状态。在这个状态的实例可以运行您的业务。
已关闭	RocketMQ实例处于故障的状态。
变更中	RocketMQ实例正在进行规格变更操作。
变更失败	RocketMQ实例处于规格变更失败状态。
暂停	RocketMQ实例处于已冻结状态，用户可以在“更多”中续费开启冻结的RocketMQ实例。
注销	RocketMQ实例已经过期并关闭，需要重新购买实例。

实例详情

1、 实例列表点击【管理】按钮，进入实例列表页面。

2、 查看实例详情信息，包括基本信息、运行状态、网络、接入点和全局监控5个部分。



3、各部分参数及内容说明如下表所示。

参数	说明
实例名称	RocketMQ实例名称是指在RocketMQ中创建的一个特定实例的名称。它可以用来唯一标识和区分不同的RocketMQ实例。根据RocketMQ的设计，实例名称通常由字母、数字和下划线组成，并且长度通常不超过255个字符。实例名称应该具有描述性，以便在多个RocketMQ实例存在时进行区分和管理。
实例ID	RocketMQ实例ID是指在RocketMQ中创建的一个实例的唯一标识符。它是由系统自动生成的，用于区分不同的RocketMQ实例。实例ID可以用来管理和操作RocketMQ实例，例如创建、删除和修改实例等。实例ID通常由一串数字和字母组成，并且在RocketMQ集群中必须保持唯一性。

参数	说明
引擎类型	提供RocketMQ和CTGMQ两类引擎。
磁盘大小	购买实例选择的磁盘大小，通常为100G的整数倍。
磁盘类型	购买实例选择的磁盘类型，默认为SAS，也可选择更高的IO磁盘。
主机规格	购买实例选择的主机规格，主要展示CPU核数和内存大小，更多主机规格请参见产品规格。
broker节点数	RocketMQ的broker节点数是指在一个RocketMQ集群中扮演broker角色的节点数量。每个broker节点负责存储消息、处理消息的传输和消费者的请求等功能。
实例描述	用户根据需要可填写实例备注。
运行状态	实例当前运行状态，各状态描述见上文表格实例状态说明。
付费类型	创建实例时选择的付费类型，有包周期/按需两个选项。
创建时间	实例创建时间。
到期时间	包周期的实例到期时间，按需付费类型不展示。
专用网络	创建实例时绑定的VPC名称，详见订购前准备。
子网	创建实例时绑定的子网名称，详见订购前准备。
安全组	创建实例时绑定的安全组名称，详见订购前准备。
客户端协议	RocketMQ客户端协议是指在使用RocketMQ消息队列的应用程序与RocketMQ服务端之间进行通信和交互的协议规范。
网络	VPC专用网络。
接入点	接入点具体VPC内网IP。
TPS监控	标识该集群所有broker下，所有topic的生产、消费tps(2min)的时间变化曲线
流量监控	标识该集群下，所有topic的消费堆积情况，列表按倒叙排列了堆积量最大的前20个topic的堆积情况。
全局堆积	全局堆积是指消息在整个消息队列系统中的积压情况。
全局消息	标识该集群，所有topic的生产情况，按现有消息量倒叙排列。

修改实例

场景说明

RocketMQ的修改实例的场景描述如下：

在使用RocketMQ时，可能会遇到需要修改实例的场景，例如：

- 业务变更：当业务需求发生变化，需要修改RocketMQ实例的名称以反映新的业务逻辑时，可以进行实例名称的修改。例如，当新增或调整了某个业务模块，需要将其对应的实例名称修改为更准确的名称，以便于管理和监控。

- 消息处理速度优化：当消息的处理速度较慢或不稳定时，可以通过调整消息保存时长参数来优化消息的处理效率。例如，可以缩短消息的保存时长，减少消息在系统中的停留时间，以提高消息的实时性和处理速度。
- 安全配置修改：当需要修改RocketMQ实例的安全配置时，可以进行相应的修改。

需要注意的是，在进行实例的修改时，应该谨慎操作，并根据实际需求和系统情况进行调整。同时，修改实例时应该遵循RocketMQ的最佳实践和建议，以确保系统的稳定性和性能。

操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

进入实例列表，在实例名称点击修改按钮，直接修改实例名称字段。

3、 点击【管理】按钮进入管理菜单。点击修改配置，在弹出窗口修改实例描述和消息保存时长等信息。

注意

- 实例描述长度限制为0~256个字符。
- 您可以调整消息保存时长参数，实现消息容量消耗的控制。安全生产和稳定性保障限制，消息需要至少保留24小时。 要完整支持延时消息的功能，消息需要至少保留48小时。

4、 点击修改安全组，选择变更用户安全组。

连接实例

场景说明

RocketMQ提供了两种连接：

- 生产者连接：生产者通过调用RocketMQ提供的API，与RocketMQ代理（Broker）建立连接。生产者连接主要用于发送消息到RocketMQ。
- 消费者连接：消费者通过调用RocketMQ提供的API，与RocketMQ代理（Broker）建立连接。消费者连接主要用于从RocketMQ订阅消息并进行消费。

RocketMQ采用基于TCP/IP的通信协议，使用长连接方式进行连接。在建立连接之前，需要配置正确的RocketMQ服务端地址和端口，并使用相应的身份验证信息（如AccessKey和SecretKey）进行认证。

建立连接后，RocketMQ客户端可以通过连接发送消息到RocketMQ集群，并从集群中接收消息进行消费。连接的建立和维护是RocketMQ消息传递的基础，确保了消息的可靠传递和高效处理。

操作步骤

首先需要在RocketMQ控制台实例详情记录下集群的接入点信息，目前支持namesrv的内网访问，将该地址作为客户端程序namesrv地址的参数，具体生产消费消息的客户端示例代码请参考快速入门-生产消费验证。

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

进入实例列表，点击【管理】按钮进入管理菜单。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、 进入主题管理菜单，点击【拨测】按钮，进行生产消费的拨测验证，验证开通的消息实例和主题。

5、 用户应用按照规范接入RocketMQ，发送、消费消息。

扩容

场景描述

当需要处理大量消息时，RocketMQ实例的扩容是一种常见的解决方案。扩容可以增加RocketMQ集群的吞吐量、存储能力和高可用性。分布式消息服务RocketMQ提供三类扩容方案，分别为节点、规格和磁盘扩容，更好满足用户不同场景下的扩容需求。

- 节点扩容：指向RocketMQ集群中添加更多的节点以增加系统的吞吐量和可靠性。通过扩容，可以将消息的发送和消费负载分摊到更多的节点上，从而提高系统的并发处理能力。
- 规格扩容：指通过增加RocketMQ的资源配置来提升系统的处理能力和性能。
- 磁盘扩容：指增加磁盘的存储容量，以满足更多消息的存储需求。

操作步骤

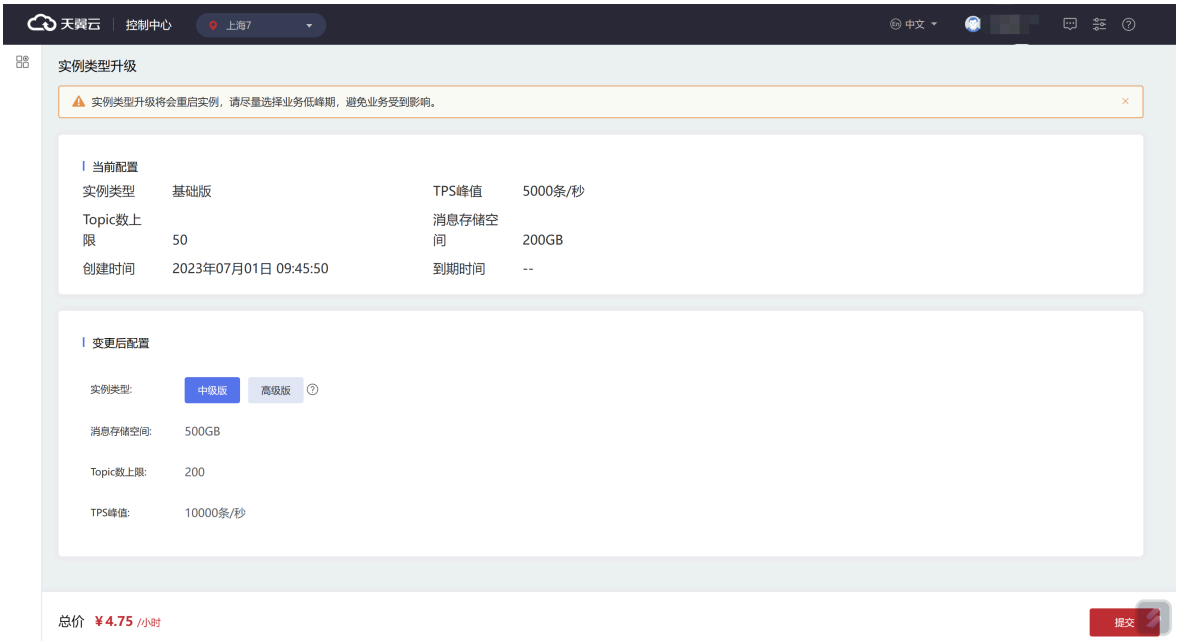
以下适用于南京3、上海7、重庆2、乌鲁木齐27、保定、石家庄20、内蒙6、晋中、北京5 节点。

扩容

1、登录RocketMQ消息控制台，可以看到当前租户下面的实例列表。

2、点击需要变更实例栏 **>** 更多 **>** 扩容。

3、进入到扩容页面，选择你需要扩容的规格即可。



注意：实例类型升级将会重启实例，请尽量选择业务低峰期，避免业务受到影响。

以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3 节点。

节点扩容

1、登录RocketMQ消息控制台，可以看到当前租户下面的实例列表。

2、 点击需要变更实例栏 **>** 更多 **>** 节点扩容。

3、 选择代理数量。

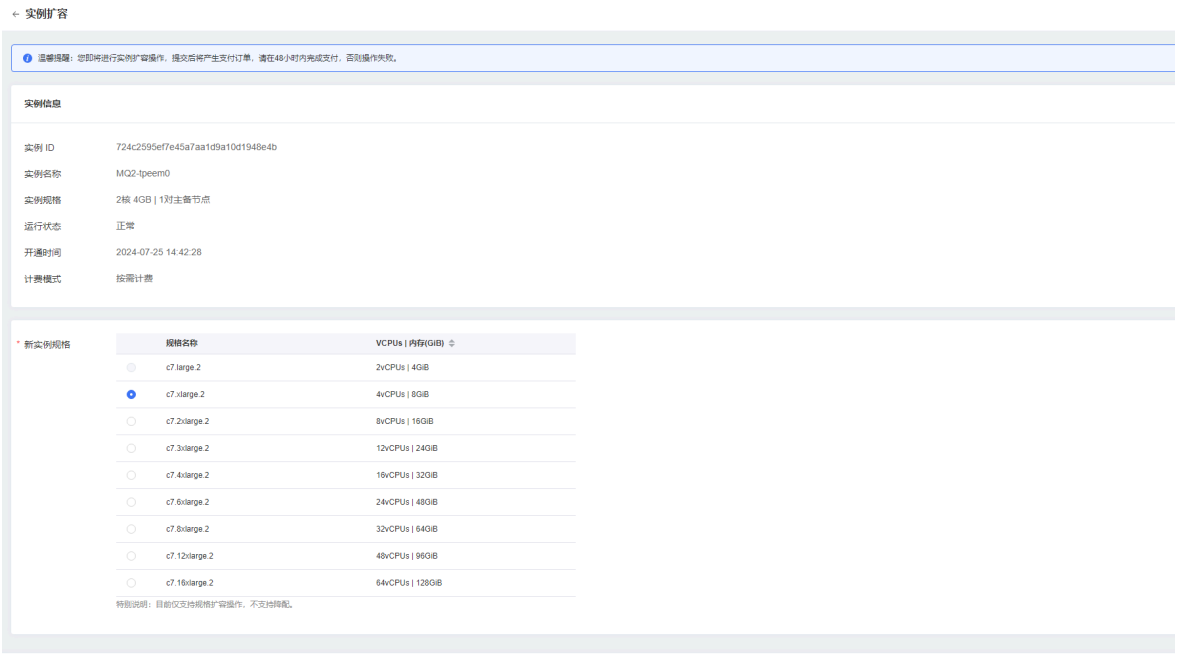
注意：目前仅支持规格扩容操作，不支持降配

规格扩容

1、登录RocketMQ消息控制台，可以看到当前租户下面的实例列表。

2、 点击需要变更实例栏 **>** 更多 **>** 规格扩容。

3、 选择新实例规格。



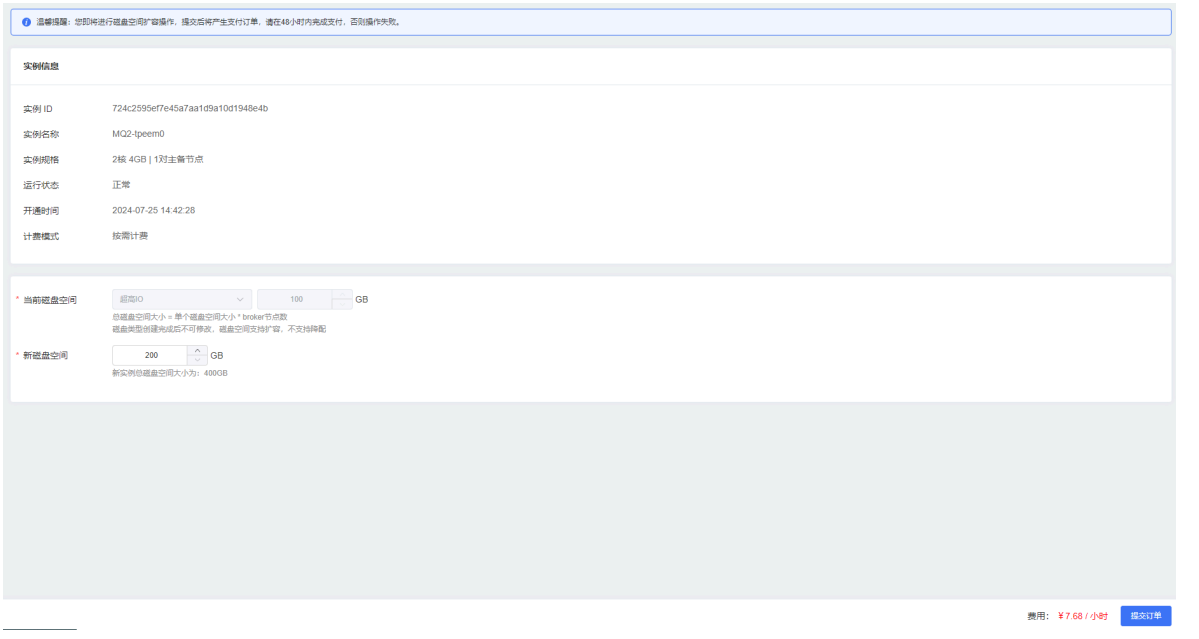
注意：目前仅支持规格扩容操作，不支持降配。

磁盘扩容

1、登录RocketMQ消息控制台，可以看到当前租户下面的实例列表。

2、 点击需要变更实例栏 **>** 更多 **>** 规格扩容。

3、 填写新磁盘空间。



注意：磁盘类型创建完成后不可修改，磁盘空间支持扩容，不支持降配。

按需转包周期

本节适用于南京3、重庆2、晋中、上海7、北京5、内蒙6、石家庄20 节点。

场景描述

RocketMQ的按需转包周期的场景描述如下：

在使用RocketMQ时，可能会遇到需要设置按需转包周期的场景，例如：

- 消息积压处理：当RocketMQ中的消息积压较多时，可能会导致消息的消费速度跟不上消息的生产速度，进而影响系统的性能和稳定性。为了解决这个问题，可以设置按需转包周期，即将一定数量的消息打包成一个批次进行消费，以提高消费的效率和吞吐量。
- 业务流量波动：在某些业务场景下，业务流量可能会出现波动，即某个时间段内的消息产生速度较快，而另一个时间段内的消息产生速度较慢。为了更好地适应业务流量的波动，可以设置按需转包周期，以根据实际的消息产生情况进行灵活的批量消费。
- 系统资源优化：当RocketMQ的消费者资源有限时，可以通过设置按需转包周期来优化系统的资源利用。通过将一定数量的消息打包成一个批次进行消费，可以减少消费者的竞争和上下文切换，提高系统的并发处理能力。
- 消息处理延迟优化：在某些场景下，对消息的实时性要求较低，可以通过设置按需转包周期来优化消息的处理延迟。将一定数量的消息打包成一个批次进行消费，可以减少消息的处理次数，从而降低消息的处理延迟。

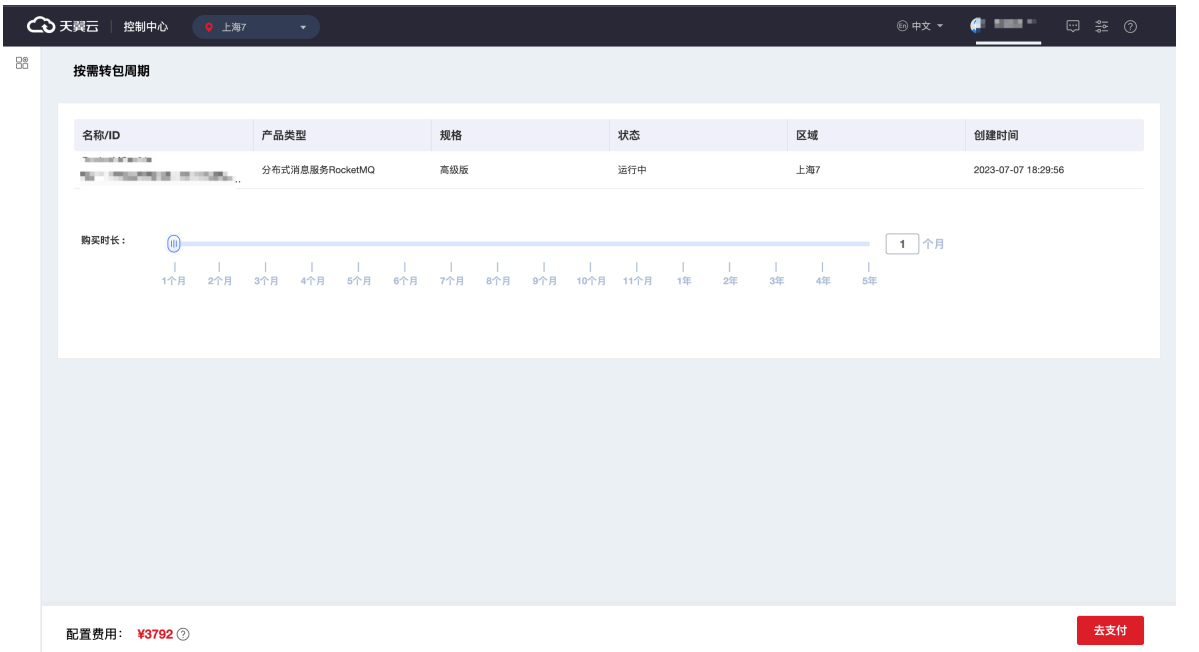
需要注意的是，在设置按需转包周期时，应根据实际业务需求和系统情况进行调整。同时，应考虑消息的重要性、消费者的处理能力、系统的资源限制等因素，以确保系统的稳定性和性能。

操作步骤

1、登录RocketMQ消息控制台，可以看到当前租户下面的实例列表。

2、点击需要变更实例栏 **>** 更多 **>** 按需转包周期。

3、进入到按需转包周期页面，在弹出来的确认窗口选择续订时长，点击确认即可。



续订和退订

场景描述

分布式消息服务RocketMQ为用户提供全面周到的服务，支持用户续订和退订的需求。

- 续订：针对包周期消息实例服务，用户可在到期前进行服务周期延长操作，即续订操作。
- 退订：用户如不需要继续使用该分布式消息服务RocketMQ实例，可进行删除实例操作，即退订操作。

续订

1、登录RocketMQ消息控制台，可以看到当前租户下面的实例列表。

2、点击需要变更实例栏 **>** 更多 **>** 续订。

3、 进入到续订页面，在弹出来的确认窗口选择续订时长，点击确认即可。

退订

1、登录RocketMQ消息控制台，可以看到当前租户下面的实例列表。

2、点击需要变更实例栏 **>** 更多 **>** 退订。

3、进入到退订页面，在弹出来的确认窗口点击确认即可。

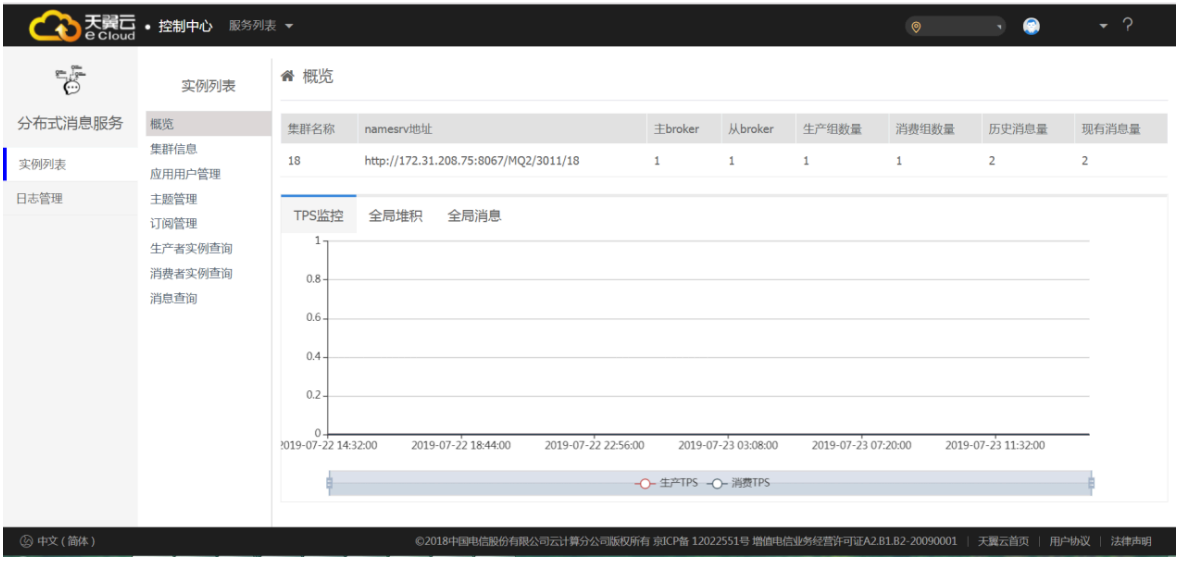
注意：

- 退订的实例处于冻结状态，请务必在实例退订前停止全部的应用。
- 在申请退订前，请做好数据备份工作，退订后数据将保留15个自然日，15天后相关数据将不予保留，且不会进行备份，务必谨慎操作。

概览

本节适用于南京3、上海7、重庆2、乌鲁木齐27、保定、石家庄20、内蒙6、晋中、北京5 节点。

概览界面如下图：



表格：显示的是当前集群的统计详情，里面包含该集群的主从broker节点数量，生产组和消费组的数量，以及整个集群累计存储消息量。

各字段释义如下表所示：

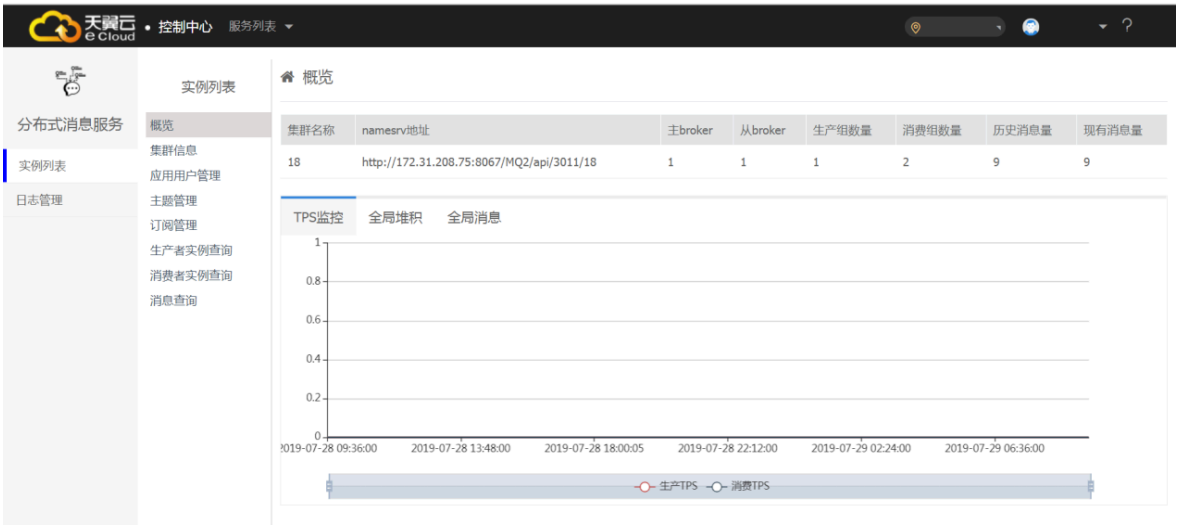
字段名称	说明
集群名称	RocketMQ集群是指将多个RocketMQ Broker节点组合在一起形成的一个集群系统。
nameserv地址	Namesrv地址概念指的是在使用RocketMQ时，客户端需要连接到Namesrv来获取Broker节点的路由信息，以便能够发送和接收消息。
主broker	主Broker（Master Broker）是指在一个Broker集群中充当主要角色的节点。
从broker	从Broker（Slave Broker）是指作为主Broker的备份节点，用于实现消息的冗余和故障转移。
生产组数量	展示该实例下生产组数量。
消费组数量	展示该实例下消费组数量。
历史消息量	展示该实例下历史消息量。
现有消息量	展示该实例下现有消息量。

TPS监控

RocketMQ的TPS监控指的是对RocketMQ消息中间件每秒处理的消息数量进行监控和统计。TPS即每秒事务数（Transactions Per Second），用于衡量系统在单位时间内能够处理的事务数量。

RocketMQ TPS监控可以帮助用户了解消息中间件的处理能力和性能状况，在系统运行期间，通过监控数据可以及时发现潜在的性能问题或瓶颈，以便进行调优或扩展。一般来说，吞吐量越高，表示系统的处理能力越强。

分布式消息服务RocketMQ标识该集群所有broker下，所有topic的生产、消费tps(2min)的时间变化曲线。

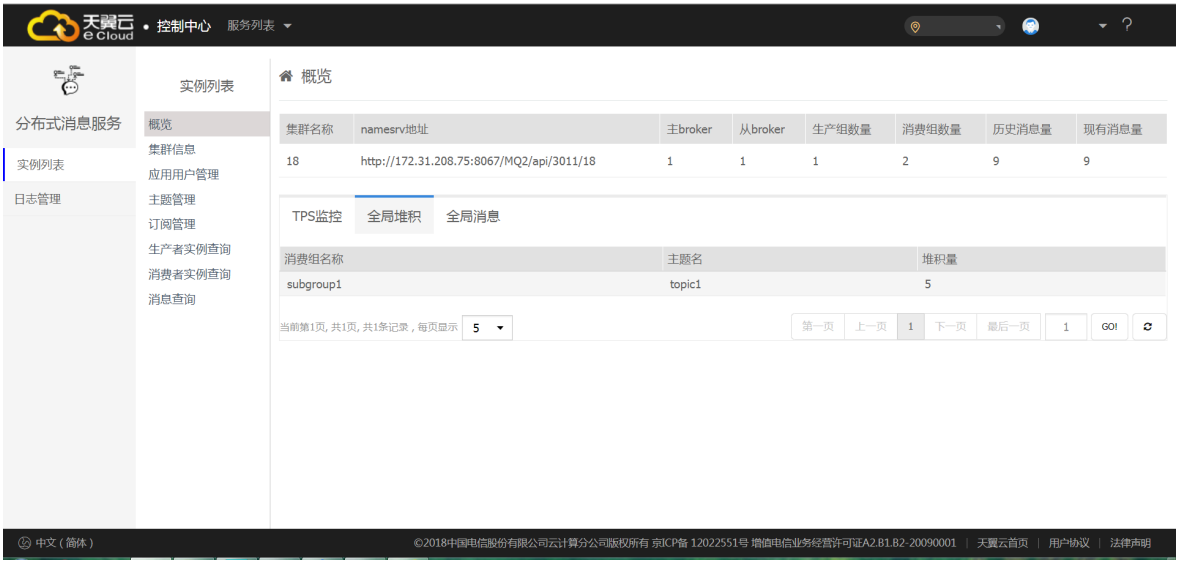


全局堆积

在使用RocketMQ时，可能会遇到消息堆积的情况，即未能及时处理和消费的消息大量积压在消息队列中。

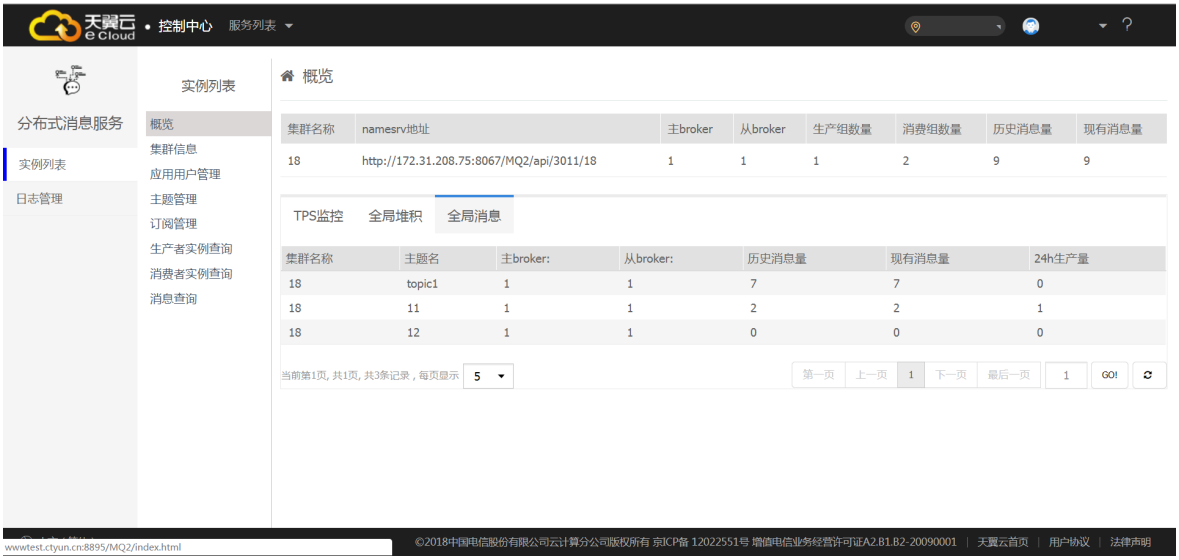
造成消息堆积的原因可能是消费者处理消息的速度不够快，或者消费者出现故障导致无法正常消费消息。当消息堆积发生时，可能会导致系统的稳定性下降，甚至影响整个应用的正常运行。

分布式消息服务RocketMQ标识该集群下，所有topic的消费堆积情况，列表按倒叙排列了堆积量最大的前20个topic的堆积情况。



全局消息

分布式消息服务RocketMQ标识该集群，所有topic的生产情况，按现有消息量倒叙排列。



集群信息

场景描述

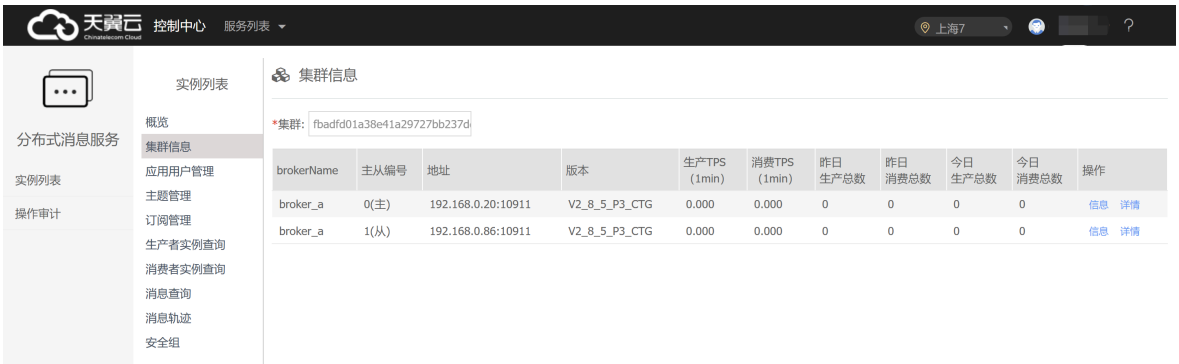
RocketMQ集群信息是指描述RocketMQ集群的配置和状态的信息。它包括集群的名称、角色、节点信息、主题信息等。通过集群信息，可以管理和监控RocketMQ集群的运行情况，包括节点的状态、消息的发送和消费情况，以及负载均衡等。集群信息对于配置、监控和管理RocketMQ集群非常重要，能够确保集群的正常运行和高可用性。

- 配置管理：集群信息可以用于管理RocketMQ集群的配置，包括节点的IP地址、端口号、角色（如Master、Slave）、存储路径等。通过集群信息，可以方便地查看集群配置，协助保障集群的正常运行。
- 监控和管理：集群信息可以用于监控RocketMQ集群的状态和健康状况。通过集群信息，可以查看集群的运行状态、消息的发送和消费情况，以及节点的负载情况等。这对于集群的管理和故障排查非常重要。
- 负载均衡：集群信息可以用于实现RocketMQ的负载均衡。通过集群信息，可以根据节点的负载情况，动态地调整消息的分布和消费的负载，保证集群的高可用性和性能。
- 容灾和高可用性：集群信息可以用于实现RocketMQ的容灾和高可用性。通过配置主备节点和多副本模式，可以保证消息的可靠传递和数据的一致性。集群信息可以用于监控主备节点的状态，并在主节点故障时自动切换到备节点，保证集群的高可用性。

集群列表

分布式消息服务RocketMQ集群信息界面展示如下图：

以下适用于南京3、上海7、重庆2、乌鲁木齐27、保定、石家庄20、内蒙6、晋中、北京5 节点。

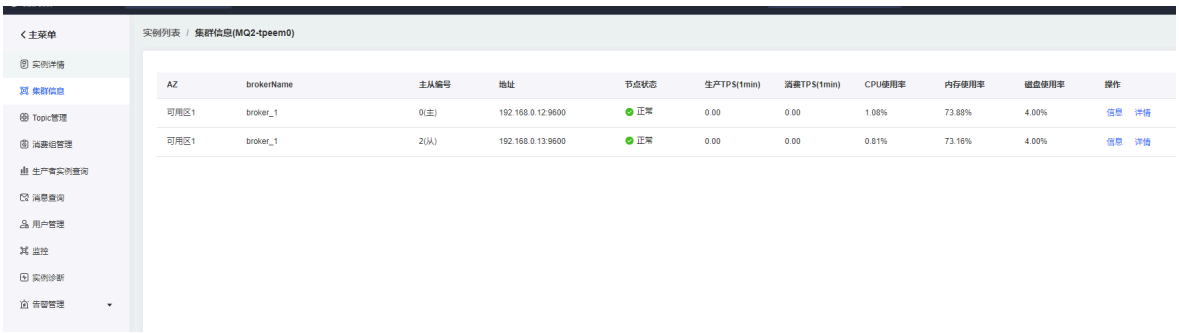


列表展示该集群每个broker的统计指标概况：

其中关键指标

- 昨天生产总数为前一天24小时内生产消息总数
- 昨天消费总数为前一天24小时内消费消息总数
- 今日生产总数为当前生产消息总量减去当天点集群内的生产消息总数
- 今日消费总数为当前消费消息总量减去当天点集群内的消费消息总数

以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7、呼和浩特3 节点。



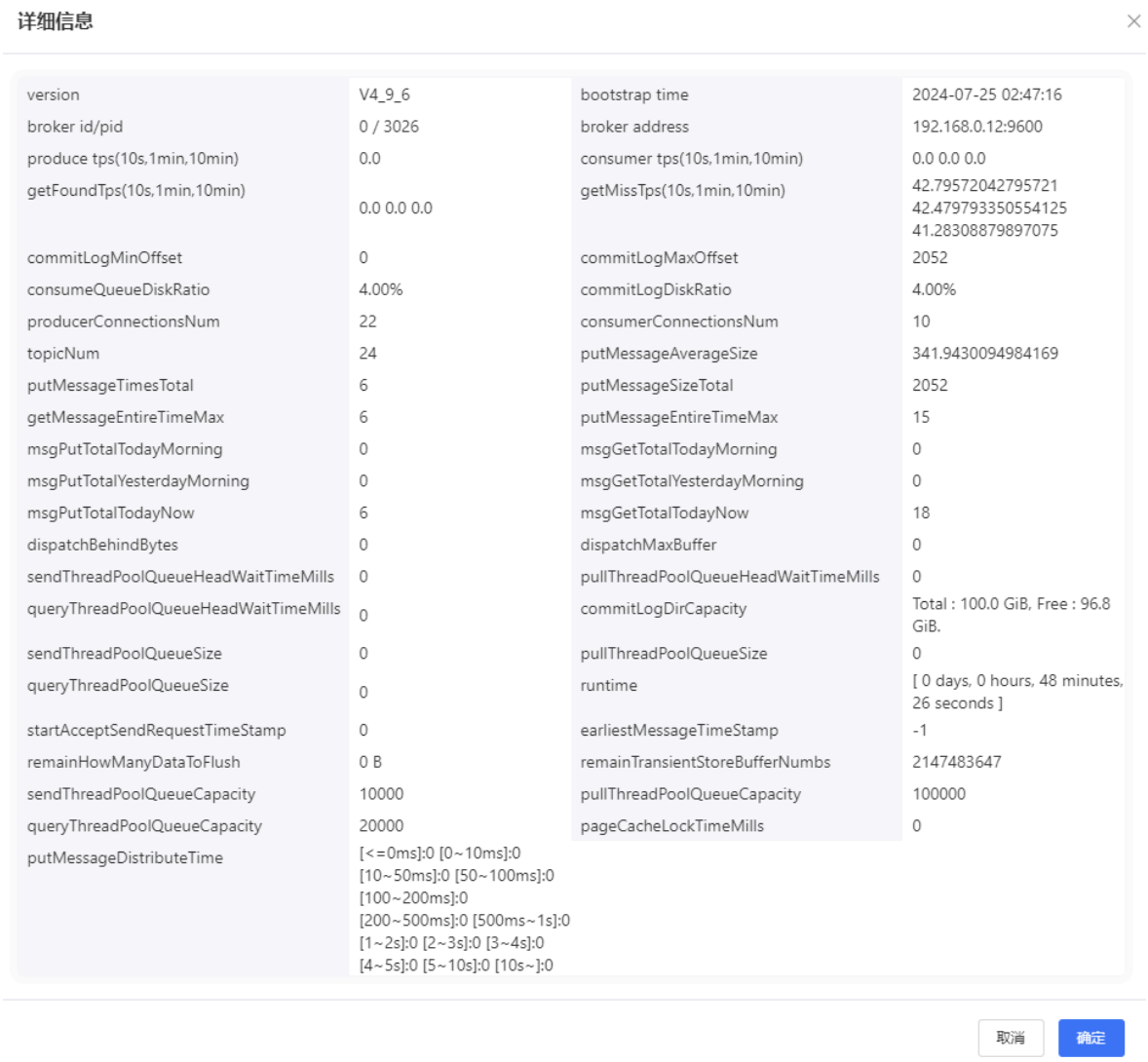
列表展示该集群每个broker的统计指标概况：

其中关键指标

- 生产TPS（1min），即生产TPS（每分钟事务数），是指RocketMQ消息生产者在一分钟内发送的消息数量。TPS是衡量系统性能的重要指标之一，它反映了系统处理能力的强弱。对于RocketMQ来说，生产TPS代表了消息生产者的发送速度和系统的处理能力。较高的生产TPS意味着系统能够处理更多的消息请求，具有更高的吞吐量。监控和优化生产TPS可以帮助提高系统的性能和可扩展性。
- 消费TPS（1min），即消费TPS（每分钟事务数），是指RocketMQ消息消费者在一分钟内成功消费的消息数量。消费TPS反映了系统消费消息的速度和处理能力。较高的消费TPS表示系统能够高效地处理大量的消息，具有更高的吞吐量。监控和优化消费TPS可以帮助提高系统的消费能力和性能。

Broker信息

展示该集群具体broker信息，包括该broker的详细指标，如下图：



其中关键指标为：

- commitLogMaxOffset为当前brokercommitLog最大的物理偏移。通过commitLogMaxOffset，RocketMQ可以追踪和管理消息的存储位置。当有新的消息写入时，RocketMQ会将消息追加到Commit Log文件的末尾，并更新commitLogMaxOffset的值。消费者在消费消息时，可以根据commitLogMaxOffset来确定从哪个偏移量开始消费消息。
- consumeQueueDiskRatio为消费队列存储的文件占用的磁盘空间比例。通过配置consumeQueueDiskRatio，可以在保证消费队列的性能的同时，控制磁盘空间的占用。较小的consumeQueueDiskRatio值可以提高消费队列的读写性能，但会增加内存的使用。较大的consumeQueueDiskRatio值可以降低内存的使用，但可能会降低消费队列的读写性能。根据实际需求，可以根据系统的内存和磁盘资源情况来调整consumeQueueDiskRatio的值，以获得更好的性能和资源利用率。
- putMessageDistributeTime为消息写入commitLog的耗时分布。通过配置putMessageDistributeTime，可以了解消息从发送到最终被消费的整体时间。这对于监控和优化消息传递的性能和延迟非常有用。注

意，putMessageDistributeTime是一个估计值，实际的消息传递时间可能会受到网络状况、消费者处理能力等多种因素的影响。因此，在配置putMessageDistributeTime时，需要根据实际情况进行调整，并结合其他指标进行综合分析。

Broker详情

标签管理

标签可以识别资源，您可以将作用相同的分布式消息服务RocketMQ资源通过标签进行归类，便于搜索和资源聚合。本文介绍标签的使用说明以及标签的相关操作。

使用说明

- 标签都由一对键值对（Key-Value）组成。
- 资源的任一标签的标签键（Key）必须唯一。

  例如，实例先添加了type:order标签，后续如需添加type:pay标签，需先删除type:pay标签。

添加标签

1. 登录分布式消息服务RocketMQ控制台，在左侧导航栏单击实例列表 。
2. 在顶部菜单栏，选择地域，如华东1。

   在该地域的所有实例都展示在实例列表页面。
3. 在实例列表页面，找到您想要添加标签的实例，在其标签列，将光标移动至 图标，在弹出的消息框中，单击 + 添加 。

   
4. 在编辑标签面板，按要求输入标签键和标签值 ，然后单击确定 。

根据标签过滤资源

说明

本文介绍如何通过标签过滤实例列表。

1. 在实例列表页面，单击筛选标签 。

2. 分别在标签键和标签值下的列表中选择对应的值，然后单击搜索，列表中将只显示绑定了对应标签的实例。

编辑和删除标签

1. 在实例列表页面，找到您想要编辑或删除标签的实例，在其标签列，将光标放至 图标，在弹出的消息框中，单击编辑。

2. 在编辑标签面板，按需执行相应操作：
   - 编辑标签：修改标签键和标签值输入框中的取值，或在新的一行中输入新的标签键值对，然后单击确定 。
   - 删除标签：在您需删除的标签键值对右侧，单击删除按钮，然后单击确定 。

回收站管理

分布式消息服务RocketMQ实例在被删除后，会被临时存入回收站中，此时实例中的数据尚未被彻底删除，在保留天数内支持从回收站中恢复此实例。超过保留天数的实例会被彻底删除，无法恢复。

约束与限制

- 回收站中的实例不会收取实例的费用，但是会收取存储空间的费用。
- 包年/包月的实例退订后会存入回收站中，此时不会收取实例的费用，但是收取存储空间的费用。
- 回收站中的按需实例，不支持实例恢复。

回收站管理

1. 登录分布式消息服务RocketMQ管理控制台。
2. 在左侧导航栏选择“回收站”，进入“回收站”页面。

恢复RocketMQ实例

1. 登录分布式消息服务RocketMQ管理控制台。
2. 在左侧导航栏选择“回收站”，进入“回收站”页面。
3. 在待恢复RocketMQ实例所在行，单击“恢复”。

删除回收站中的实例

1. 登录分布式消息服务RocketMQ管理控制台。
2. 在左侧导航栏选择“回收站”，进入“回收站”页面。
3. 在待删除RocketMQ实例所在行，单击“销毁”。

注意

回收站中的RocketMQ实例删除后，实例中原有的数据将被删除，且没有备份，请谨慎操作。

用户管理

创建用户

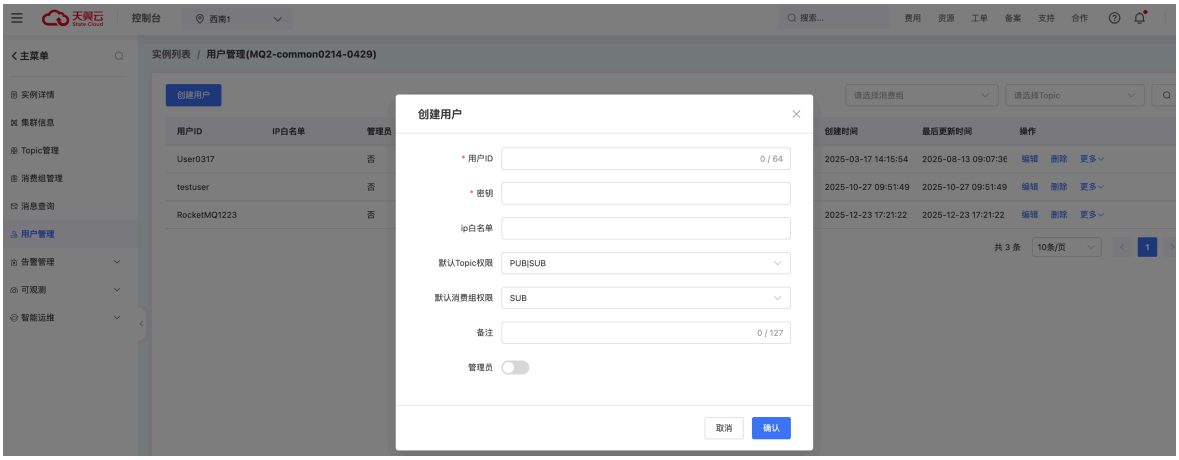
1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击左上角地域选择对应资源池。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、 进入用户管理菜单，点击【创建用户】按钮

5、 在弹出的创建用户页面，填写如下字段信息，点击确认保存。



1、用户ID：必填，7-64位字符,只能输入大小写字母，下划线，数字。

2、密钥：必填，8～26个字符，必须同时包含大写字母、小写字母、数字和英文格式特殊符号(@%^*_+!$-=.)中的至少三种类型且不能包含空格。

3、IP白名单：非必填。

4、默认Topic权限

- DENY：拒绝所有操作。
- PUB：仅允许发布消息到Topic。
- PUB|SUB：允许发布+订阅。
- SUB：仅允许订阅Topic消息。

5、默认消费组权限

- DENY：拒绝所有操作。
- SUB：允许订阅Topic并关联该消费组。

6、备注：按需备注该用户的说明。

7、管理员：有Topic和消费组所有权限。

Topic管理

查看Topic

场景描述

RocketMQ中的Topic是消息发布和订阅的核心概念之一。主题可以理解为消息的分类或者标签，用于将一组相关的消息进行归类和管理。

- 在RocketMQ中，消息发送者通过指定主题来发布消息，而消息消费者则通过订阅主题来接收消息。一个主题可以有多个生产者和多个消费者。生产者将消息发送到主题，而消费者从主题中订阅消息。
- 主题的创建和管理是由RocketMQ的管理员或者应用程序开发者负责的。创建主题时，需要指定主题的名称，并可以设置一些属性，如主题的存储策略、消息的过期时间等。主题的名称应该具有一定的描述性，能够清楚地表达所属消息的含义或用途。
- 主题在RocketMQ中具有一定的灵活性和扩展性。可以根据业务需求创建多个主题，每个主题用于处理不同类型或不同业务场景下的消息。通过合理地设计和管理主题，可以实现消息的高效传递和灵活的消息路由。

总之，主题是RocketMQ中用于分类和管理消息的重要概念，通过主题可以实现消息的发布和订阅，帮助开发者更好地组织和管理消息系统。

Topic列表

1、 进入Topic管理菜单查看Topic列表，若列表为空则新建Topic，创建Topic详情见具体操作步骤。

2、 支持按照集群、Broker和Topic查询，下拉选择或者输入查询内容，点击【查询】按钮即可展示需要的Topic数据。

3、 查看Topic基本信息，包括Topic名、brokerName、读队列数量、写队列数量、权限、是否有序。其中Topic权限包括读写、只读、只写3类权限。

Topic详情

展示当前Topic的统计指标、队列分布信息、消费组、生产组的情况。

1、队列信息

2、消费组

标识正在连接该topic的消费组列表。

消费方式分为两种：

- CONSUME_PASSIVELY为push消费模式
- CONSUME_ACTIVELY为pull消费模式

路由

展示topic的队列分布情况，队列分布在哪些broker上

RocketMQ的队列分布情况是根据Broker的配置和主题的配置来确定的。在RocketMQ中，每个主题可以有多个队列，每个队列可以有多个Broker来提供服务。

队列的分布情况可以通过以下几个因素来确定：

- Broker的配置：在RocketMQ中，每个Broker都有一个唯一的名称，可以通过名称来识别和配置Broker。当创建主题时，可以指定消息队列的数量和分布策略。分布策略可以是固定的，也可以是根据一定规则动态分配的。
- 主题的配置：在创建主题时，可以指定队列的数量和分布策略。分布策略可以是按照固定数量进行分配，也可以是根据一定规则进行动态分配。例如，可以将消息队列均匀地分布在不同的Broker上，也可以根据消息的属性将消息路由到不同的队列。
- 负载均衡策略：RocketMQ提供了多种负载均衡策略，用于在Broker集群中均衡地分布消息队列。负载均衡策略可以根据Broker的负载情况、网络延迟等因素来进行动态调整，以确保消息队列的均衡分布和高效处理。

总的来说，RocketMQ的队列分布情况是根据Broker的配置、主题的配置和负载均衡策略来确定的。通过合理的配置和使用负载均衡策略，可以实现消息队列的均衡分布和高效处理，提高消息系统的性能和可扩展性。

导出

将勾选的topic，导出其配置及路由情况，为excel文件，导出格式如下：

	A	B	C	D	E	F	G
1	topic名称	queue数量	读写权限	broker名称	broker IP	集群名称	是否有序
2	huxl_topic234223	8	4	brokeryhh	10.142.90.28:8245	testyhhcluster2	false
3	testTopic3	16	6	brokeryhh	10.142.90.28:8245	testyhhcluster2	false
4	huxl_topic1213	10	2	brokeryhh	10.142.90.28:8245	testyhhcluster2	false
5	testTopic1	4	6	brokeryhh	10.142.90.28:8245	testyhhcluster2	false
6	testTopic2	16	6	brokeryhh	10.142.90.28:8245	testyhhcluster2	false
7	zipkin	4	6	brokeryhh	10.142.90.28:8245	testyhhcluster2	false
8	huxl_topic23413	10	2	brokeryhh	10.142.90.28:8245	testyhhcluster2	false
9	huxl_topic1123	8	4	brokeryhh	10.142.90.28:8245	testyhhcluster2	true
10							
11							

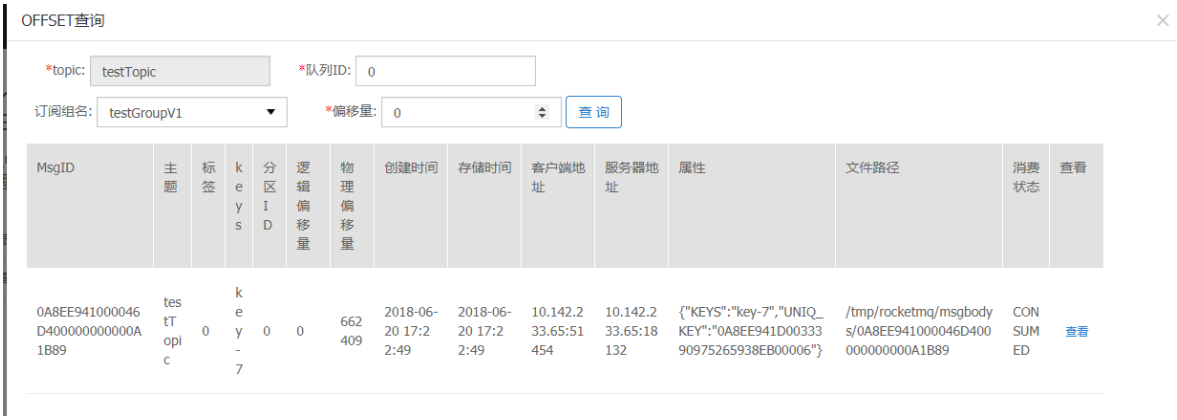
备注：

如果勾选了topic，则导出勾选的topic;如果没有勾选，则导出该集群，选定broker下的所有topic信息。

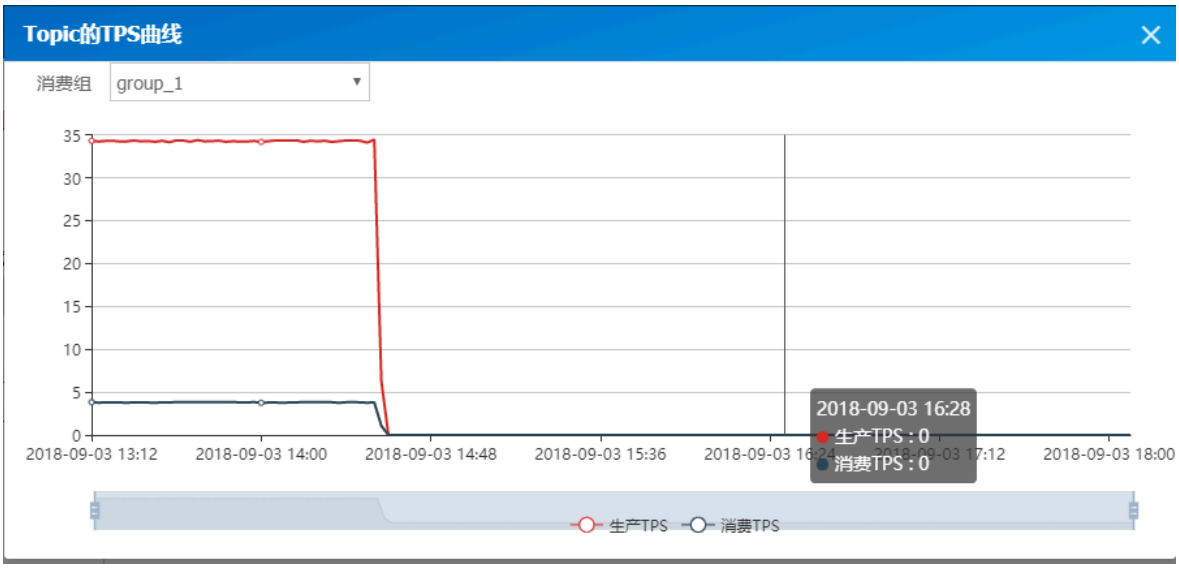
堆积量

展示该topic被哪些消费组消费，及该消费组对应的未消费的消息数量统计；

通过“offset查询”，可以查找队列相应偏移量的消息；



“TPS监控”：显示指定topic的生产、消费的tps趋势图。



创建Topic

场景描述

RocketMQ的Topic是消息的逻辑分类单位，用于将消息进行分组和管理。创建Topic的场景可以根据具体业务需求来确定，以下是一些常见的场景描述：

- 消息发布与订阅：当需要实现消息发布与订阅模式时，可以创建一个主题来管理相关的消息。发布者可以将消息发送到该主题，而订阅者可以订阅该主题以接收感兴趣的消息。
- 事件驱动架构：在事件驱动的架构中，不同的模块之间通过事件进行通信。每个事件可以对应一个主题，模块可以将事件发送到相应的主题，其他模块可以订阅该主题以接收事件通知。
- 日志收集与分析：当需要收集大量的日志数据并进行分析时，可以创建一个主题来管理日志消息。日志产生者可以将日志消息发送到该主题，而日志消费者可以订阅该主题以进行实时分析或存储。
- 异步处理：当需要将某些操作异步处理时，可以创建一个主题来管理相关的异步消息。操作发起者可以将异步消息发送到该主题，而异步处理器可以订阅该主题以进行后续的异步处理。
- 分布式事务消息：在分布式系统中，当需要实现分布式事务消息时，可以创建一个主题来管理相关的事务消息。事务发起者可以将事务消息发送到该主题，而事务消费者可以订阅该主题以进行事务的处理和确认。

总的来说，创建主题的场景可以根据具体的业务需求来确定。主题可以帮助将消息进行逻辑分类和管理，实现不同的消息传递模式和业务场景

新建Topic

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、 进入Topic管理菜单，点击【创建Topic】按钮

5、 在弹出的创建Topic页面，填写如下字段信息

1）默认展示当前集群名称，不可修改。

2）选择Topic所在的Broker，按照实例创建时候选择的主备节点对数列出每个broker，可复选。

3）填写Topic名称，名字限制2到64个字符，超过限制会导致创建Topic失败，用户创建Topic只能包含大小写字母数字以及_和-符号。

4）按照实际需求填写Topic备注。

5）填写每个Broker分区数，分区数必须大于0，小于等于8，创建严格顺序队列时，设置分区数为1，且只能选择一个broker。

6）选择生产模式，RocketMQ是一个开源的分布式消息中间件，它支持两种消息生产模式：有序和无序。

- 有序消息生产模式（Ordered Message）是指按照特定规则将消息发送到相同的Message Queue中，并且确保消息在消费者端按照相同的顺序进行消费。这种模式适用于那些需要严格按照消息顺序进行处理的场景，比如订单处理、流程审批等。
- 无序消息生产模式（Unordered Message）是指消息发送到不同的Message Queue中，每个Queue都是独立的。消费者可以并行地从多个Queue中消费消息，而无需关心消息的顺序。这种模式适用于那些不需要严格按照消息顺序处理的场景，比如日志收集、异步通知等。

需要注意的是，无论是有序还是无序消息生产模式，RocketMQ都提供了高可靠性的消息传输和存储，并支持水平扩展和高吞吐量的特性。根据具体的业务需求，选择适合的消息生产模式能够更好地满足应用的要求。

7）选择Topic的读写权限，支持读写、只读、只写3类权限。

6、 完成Topic信息填写后，保存确认即可新增主题。

7、 若希望批量创建Topic，可点击【导入Topic】按钮

- 导入Topic

注意：输入的Topic名不要带空格等特殊字符。

通过上传csv文件,批量创建主题。格式：点击【下载模板】按钮下载。

- Topic模板

批量上传Topic的模板，必须使用模板，才能够上传成功，模板格式如下：

TopicName	QueueNums	Perm	MessageType
Topic1	4	6	NORMAL
Topic2	1	6	FIFO
Topic3	4	6	DELAY
Topic4	4	6	TRANSACTION

修改Topic

场景描述

RocketMQ中的Topic一旦创建后，通常是不允许直接修改的。因为Topic的配置信息会对消息的发送和消费产生影响，直接修改Topic可能导致消息的发送和消费出现问题。

然而，有时候可能会遇到需要修改Topic配置的场景，以下是一些可能的场景描述：

- 扩展Topic的分区数量：当Topic的消息量增加，原有的分区数量可能无法满足需求时，可以考虑扩展Topic的分区数量。通过修改Topic的配置，增加分区数量，可以提高消息的并发处理能力和负载均衡性能。
- 修改Topic的生产模式，可能会对消息的发送和消费产生影响，生产模式决定了消息发送的方式和策略，直接修改可能会导致消息发送和消费的异常。
- 修改Topic的读写权限，需要考虑到消费者和生产者的配置，以确保它们能够正确地读写Topic。
- 修改Topic备注，用户按照业务需要修改注意备注。

需要注意的是，修改Topic的配置可能会对消息的发送和消费产生影响，因此在进行修改之前，需要谨慎评估和测试，确保不会对现有的消息系统造成不可逆的影响。在实际操作中，建议在创建Topic时就根据业务需求进行充分的规划和配置，避免频繁修改Topic的配置。

操作步骤

1、进入Topic管理菜单，在Topic列表点击【编辑】按钮，进入编辑Topic窗口。

2、Topic修改时，不能修改集群、broker、topic名称；可以修改分区数、读写权限、备注；

删除Topic

场景描述

在RocketMQ中，删除Topic是一个比较重要且敏感的操作，需要谨慎处理。一般来说，删除Topic的场景有以下几种情况：

- 业务不再需要该Topic：当某个Topic对应的业务已经结束或不再需要时，可以考虑删除该Topic，以释放资源和减少管理工作。
- Topic配置错误或不合适：如果创建Topic时配置错误或者配置不合适，可以考虑删除该Topic，并重新创建一个正确的Topic。
- 数据归档或清理：在某些情况下，可能需要对Topic中的数据进行归档或清理，以释放存储空间。在归档或清理之前，需要先将Topic中的消息备份或迁移到其他地方，确保数据的完整性和可恢复性。

无论是哪种场景，删除Topic都需要注意以下几点：

- 确保Topic中的消息已经被正确处理和消费：在删除Topic之前，需要确保Topic中的消息已经被正确地处理和消费，以避免数据丢失或处理中断。
- 停止生产者和消费者对该Topic的操作：在删除Topic之前，需要停止生产者和消费者对该Topic的操作，以避免数据冲突或丢失。
- 确保删除操作的权限和安全性：删除Topic通常需要管理员或具有相应权限的用户来执行，确保只有授权人员可以进行删除操作。
- 慎重操作，备份重要数据：在删除Topic之前，建议备份Topic中的重要数据，以防止误操作或数据丢失。

总之，删除Topic是一个敏感操作，需要在慎重考虑和评估后进行，以确保不会对业务和数据产生不可逆的影响。

操作步骤

1、进入Topic管理菜单。

2、在Topic管理菜单选择将要删除的主题，在更多下拉框选择删除，即可完成删除

注意事项：

- 删除主题前必须确保该主题对应的生产消费实例已经全部停止。
- 删除主题后消息数据会发生丢失。

拨测

场景描述

在进行消息发送与接收验证的过程中，我们需要确保RocketMQ的生产者能够将消息成功发送到指定的Topic，同时消费者能够接收到这些消息。通过验证，可以确认生产者发送的消息能够准确无误地到达消费者，这是消息队列最基本也是最重要的功能要求。在开发和测试阶段，通过消息发送与接收验证可以及时发现并修复可能存在的问题，避免在实际应用中出现故障。

操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

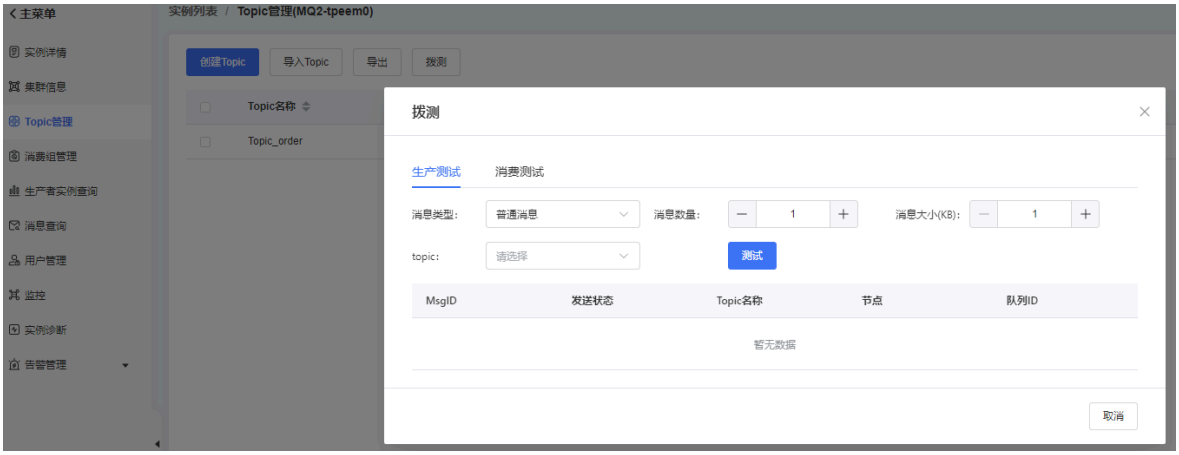
2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

进入实例列表，点击【管理】按钮进入管理菜单。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、 进入Topic管理菜单，点击【拨测】按钮，进行生产消费的拨测验证，验证开通的消息实例和Topic。

1）生产测试拨测：



- 选择消息类型，默认普通消息。
- 填写需要产生的测试消息数量，以及每条消息的大小，默认每条消息1KB，建议不超过4MB(4096KB)。
- 选择已建的消息Topic，若无选项，请新增Topic，详见上文创建Topic和订阅组。
- 点击【测试】按钮，按照已填写规格及数量产生测试消息数据，展示消息数据的信息，包括消息ID(messageID)、发送状态、主题名（topic名）、Broker名、队列ID。

拨测功能涉及消息发送状态码，以下是RocketMQ消息发送状态码及其说明：

✧ SEND_OK（发送成功）：表示消息成功发送到了消息服务器。

✧ FLUSH_DISK_TIMEOUT（刷新磁盘超时）：表示消息已经成功发送到消息服务器，但是刷新到磁盘上超时。这可能会导致消息服务器在宕机后，尚未持久化到磁盘上的数据丢失。

✧ FLUSH_SLAVE_TIMEOUT（刷新从服务器超时）：表示消息已经成功发送到消息服务器，但是刷新到从服务器上超时。这可能会导致主从同步不一致。

✧ SLAVE_NOT_AVAILABLE（从服务器不可用）：表示消息已经成功发送到消息服务器，但是从服务器不可用。这可能是由于网络故障或从服务器宕机引起的。

✧ UNKNOWN_ERROR（未知错误）：表示发送消息时遇到了未知的错误。一般情况下建议重试发送消息。

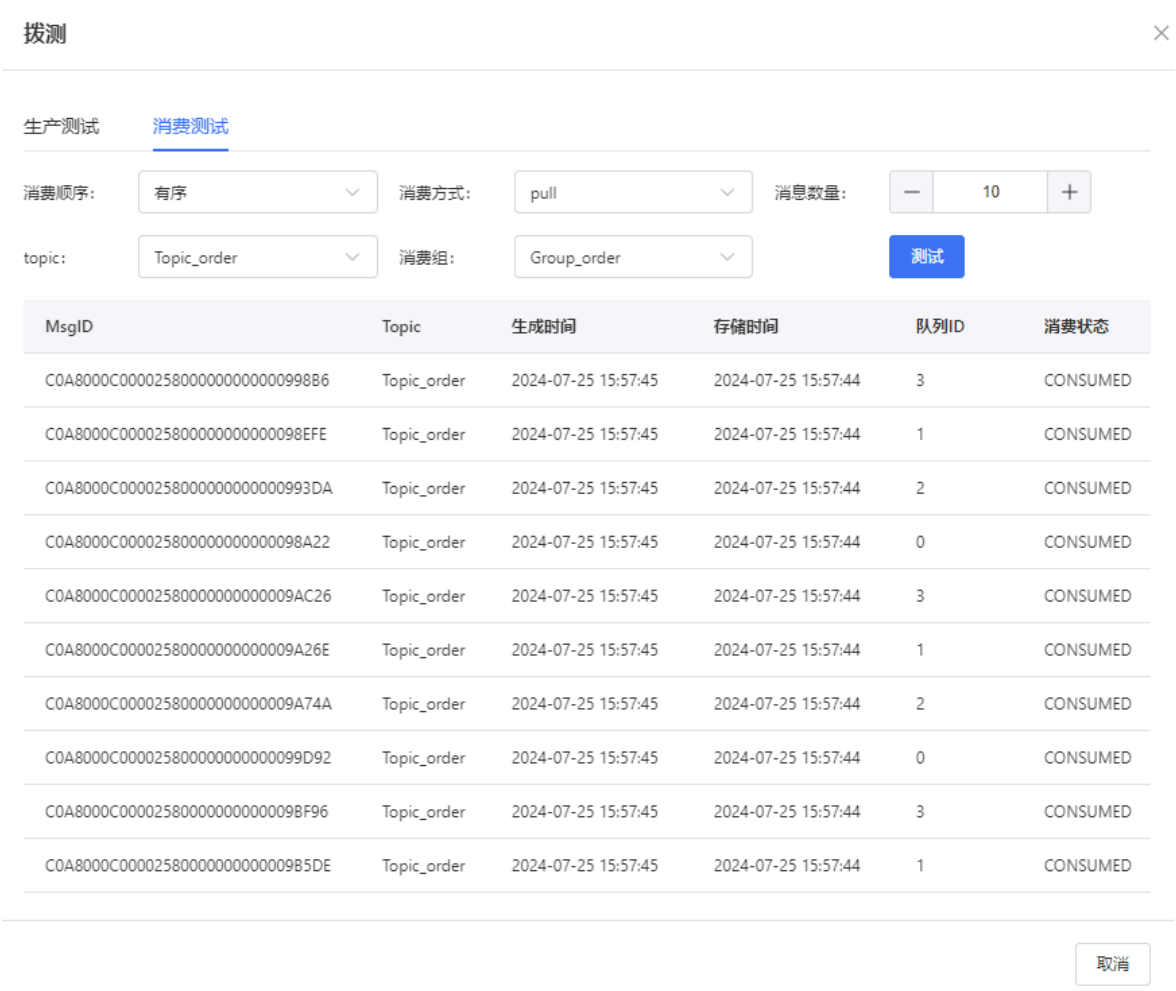
✧ MESSAGE_SIZE_EXCEEDED（消息大小超过限制）：表示消息的大小超过了消息服务器的限制。需要检查消息的大小是否合适。

✧ PRODUCE_THROTTLE（消息生产被限流）：表示消息生产者的频率超出了消息服务器的限制。这可能是由于消息发送频率过高引起的。

◇ SERVICE_NOT_AVAILABLE（服务不可用）：表示消息服务器不可用。这可能是由于网络故障或者消息服务器宕机引起的。

请注意，以上状态码仅适用于RocketMQ消息发送阶段，并且并不代表消息是否成功被消费者接收。同时，这些状态码也可能因版本变化而有所不同，建议查阅官方文档获取最新信息。

2）消费测试拨测：



- 选择消息顺序，下拉选择无序/有序，默认选项为无序。

RocketMQ是一种开源的分布式消息中间件，它支持有序消息和无序消息。

◇ 有序消息是指消息的消费顺序与发送顺序完全一致。在某些业务场景下，消息的处理需要保证顺序性，例如订单的处理或者任务的执行。RocketMQ提供了有序消息的支持，通过指定消息的顺序属性或使用消息队列的分区机制，可以确保消息按照指定的顺序进行消费。

◇ 无序消息则是指消息的消费顺序与发送顺序无关。无序消息的特点是高吞吐量和低延迟，适用于一些不要求严格顺序的业务场景，如日志收集等。

在RocketMQ中，有序消息和无序消息的实现方式略有不同。有序消息需要借助MessageQueue的分区机制和消费者端的顺序消息消费来实现。而无序消息则是通过消息的发送和接收的并发处理来实现的。

总的来说，RocketMQ既支持有序消息也支持无序消息，根据业务需求选择合适的消息类型来满足业务的要求。

- 选择消费方式，目前仅提供pull方式。值得注意的是，RocketMQ还提供了推送（push）方式的消费模式，其中消息队列服务器会主动将消息推送给消费者。但在当前仅限于pull方式的消费模式。
- 填写消费数量。
- 下拉选择选择已建的消息主题和订阅组，若无选项，请新增主题和订阅组，详见上文创建主题和订阅组。
- 点击【测试】按钮，按照已填写规格及数量产生消费数据，展示消息数据的信息，包括消息ID(messageID)、主题名称（topicName）、生成时间、存储时间、队列ID、消费状态。

拨测功能涉及消息消费状态码，RocketMQ消费状态码是指在消息消费过程中，对消费结果进行标识的状态码。以下是常见的RocketMQ消费状态码：

✧ CONSUME_SUCCESS（消费成功）：表示消息成功被消费。

✧ RECONSUME_LATER（稍后重试）：表示消费失败，需要稍后再次进行消费。

✧ CONSUME_FAILURE（消费失败）：表示消息消费出现异常或失败。

✧ SLAVE_NOT_AVAILABLE（从节点不可用）：表示消费者无法访问从节点来消费消息。

✧ NO_MATCHED_MESSAGE（无匹配的消息）：表示当前没有匹配的消息需要消费。

✧ OFFSET_ILLEGAL（偏移量非法）：表示消费的偏移量参数不合法。

✧ BROKER_TIMEOUT（Broker超时）：表示由于Broker超时导致消费失败。

重置消费位置

场景描述

在RocketMQ中，重置消费位置是一个常见的场景，用于重新设置消费者对消息队列的消费位置。以下是一些常见的RocketMQ重置消费位置的场景描述：

- 消费者异常停止后的重启：当消费者由于某种原因异常停止后，重新启动时可能需要重置消费位置。例如，消费者在处理消息时发生故障导致停止，重新启动后可以选择从上次消费的位置继续消费，或者从最早的消息开始重新消费。
- 消费者消费进度错误或需要重新处理：在某些情况下，消费者的消费进度可能出现错误，或者需要重新处理之前未正确处理的消息。此时，可以选择重置消费位置，将消费者的消费进度重置到指定的位置，从而重新消费或重新处理消息。
- 消费者切换消费模式：当消费者从集群模式切换到广播模式，或者从广播模式切换到集群模式时，可能需要重置消费位置。因为在不同的消费模式下，消费者对消息队列的消费位置和进度管理方式是不同的。
- 消费者消费速度过慢导致消息堆积：当消费者的消费速度过慢，无法及时处理消息时，可能会导致消息堆积。为了加快消息的消费速度，可以选择重置消费位置，将消费者的消费进度重置到最新的消息位置，从而快速消费积压的消息。

在进行消费位置重置时，需要注意以下几点：

- 确定重置的目标位置：在重置消费位置之前，需要确定重置的目标位置，可以选择从最早的消息位置开始消费，或者从指定的消息位置开始消费。
- 谨慎操作：重置消费位置是一个敏感操作，需要谨慎处理。在重置消费位置之前，建议备份相关数据，以防止数据丢失或处理错误。

- 考虑消费者组和消息模式：在重置消费位置时，需要考虑消费者组和消息模式的影响。不同的消费者组和消息模式可能对消费位置重置有不同的要求和限制。

总之，重置消费位置是一个常见的操作，可以帮助解决消费者异常停止、消费进度错误或重新处理消息等情况。但在进行操作时，需要谨慎评估和处理，确保不会对业务和数据产生不可逆的影响。

操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

进入实例列表，点击【管理】按钮进入管理菜单。

3、 进入实例列表，点击【管理】按钮进入管理菜单。

4、 进入Topic管理菜单，点击【重置消费】按钮，弹出窗口选择消费组名称。

5、 选择重置时间点，点击确定即可重置消费位置。

注意：当将消费时间重置到历史时间T1,则T1之前生产的消息，都变成已消费，T1之后生产的消息，都变成未消费；将消费时间重置到未来时间T2，则存量的所有消息都变成已消费。

备注：

- 重置时间：是指消息生产的时间
- 不选时间：所有消息都变成未消费
- 未来时间：所有消息都变成已消费
- 某时间点：该时间之前变成已消费，改时间之后未消费；
- v1消费模式必须关闭消费客户端

Topic导入/导出

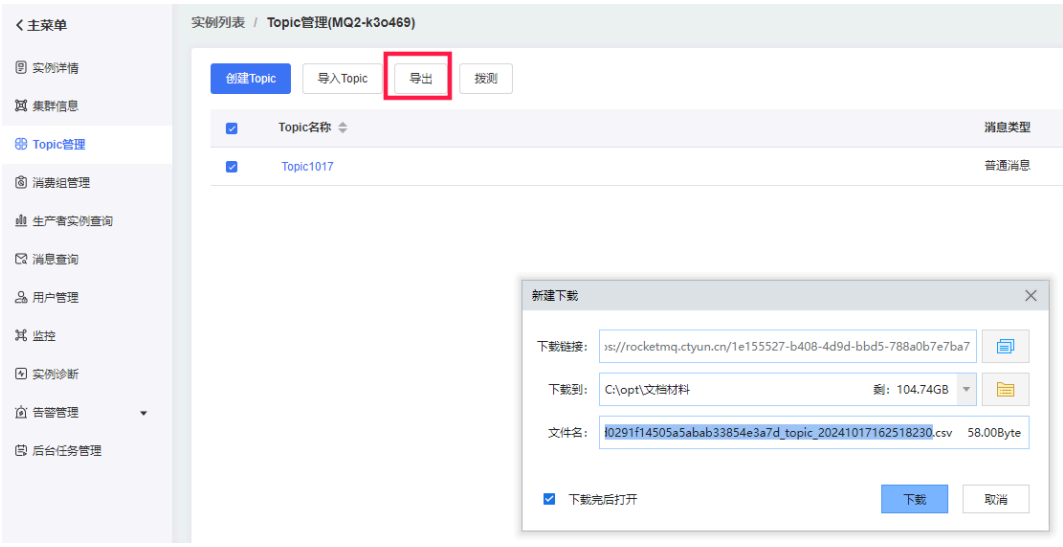
分布式消息服务RocketMQ支持将指定实例的Topic列表信息导出，再导入至其他的RocketMQ实例，实现Topic的跨实例迁移。您可以在更换实例实例时，使用Topic导入/导出功能快速批量创建出相同的Topic。

Topic导出

1. 登录分布式消息服务RocketMQ，在左侧导航栏单击 实例列表 。

2. 在顶部菜单栏选择地域，如 华东1，然后在实例列表中，单击目标实例名称。
3. 在左侧导航栏单击 Topic 管理 。
4. 在Topic 管理页面选择需导出的Topic，点击导出。

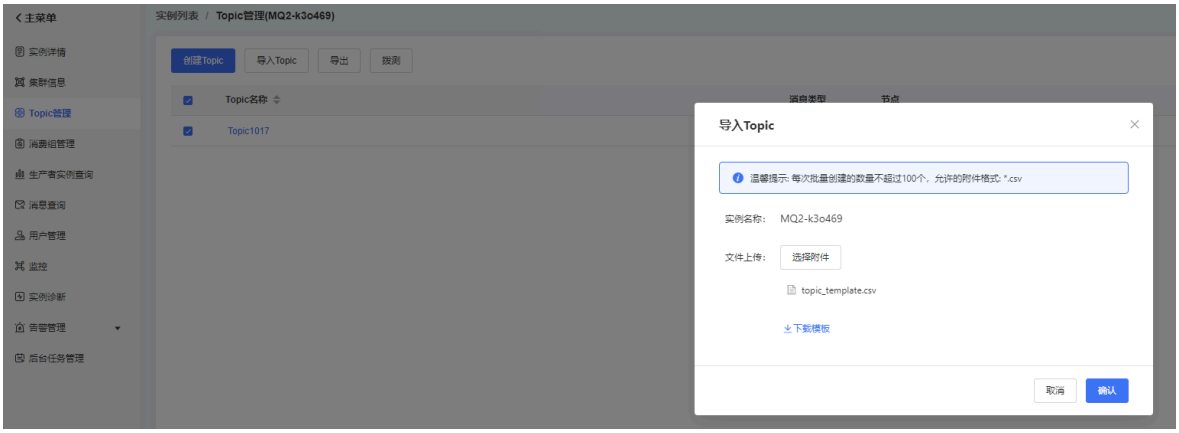
   系统会自动将该实例下的Topic资源列表数据导出并保存为.csv文件。



Topic导入

您可以将已导出的Topic列表直接导入至目标实例中，也可以根据实际需求更新列表内容再导入Topic信息。

1. 登录分布式消息队列RocketMQ控制台，在左侧导航栏单击 实例列表 。
2. 在顶部菜单栏选择地域，如 华东1 ，然后在实例列表中，单击目标实例名称。
3. 在左侧导航栏单击 Topic 管理 。
4. 在Topic 管理页面右上角单击导入Topic。
5. 选择在本地保存的Topic资源列表文件，然后单击打开，点击确认 。
6. 导入完成后，导入的Topic会出现在Topic列表中。



消费组管理

查看消费组

场景描述

在RocketMQ中，消费组（Consumer Group）是一种逻辑上的概念，用于标识一组消费者（Consumer）的集合，这些消费者共同消费同一个主题（Topic）的消息。

消费组的概念允许多个消费者同时消费同一个主题的消息，从而实现消息的负载均衡和高可用性。当一个消息发送到RocketMQ中的某个主题时，RocketMQ会将该消息推送给订阅了该Topic的所有消费组中的消费者。

每个消费组可以包含一个或多个消费者，这些消费者可以在同一个应用程序中或不同的应用程序中运行。消费者可以以集群模式（Cluster Mode）或广播模式（Broadcast Mode）进行消费。

在集群模式下，消费组中的消费者以消费者组的形式协同工作，每个消息只会被消费组中的一个消费者消费。RocketMQ会根据消费者的负载均衡策略，将消息分发给不同的消费者，实现消息的负载均衡和高可用性。

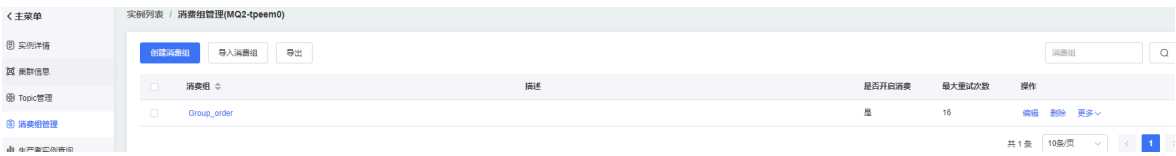
在广播模式下，消费组中的每个消费者都会独立消费消息，每个消息会被消费组中的所有消费者都消费一次。广播模式适用于需要所有消费者都处理同一份消息的场景。

消费组的概念使得RocketMQ能够支持水平扩展和容错能力。通过增加消费者数量或部署多个消费者实例，可以提高消费能力和可用性。

需要注意的是，每个消费组在RocketMQ中必须具有唯一的名称，以便区分不同的消费组。同时，每个消费组可以订阅一个或多个Topic，以满足不同业务场景的需求。

消费组列表

1、 进入消费组管理菜单查看消费组列表，若列表为空则新建消费组，创建消费组详情见具体操作步骤。



2、 支持按照集群、Broker和订阅组名查询，下拉选择或者输入查询内容，点击【查询】按钮即可展示需要的订阅组数据。

3、 查看订阅组基本信息，包括订阅组名、BrokerName、BrokerID、起始消费位置、消费机制、是否开启消费，可点击【开启消费】和【关闭消费】进行切换。

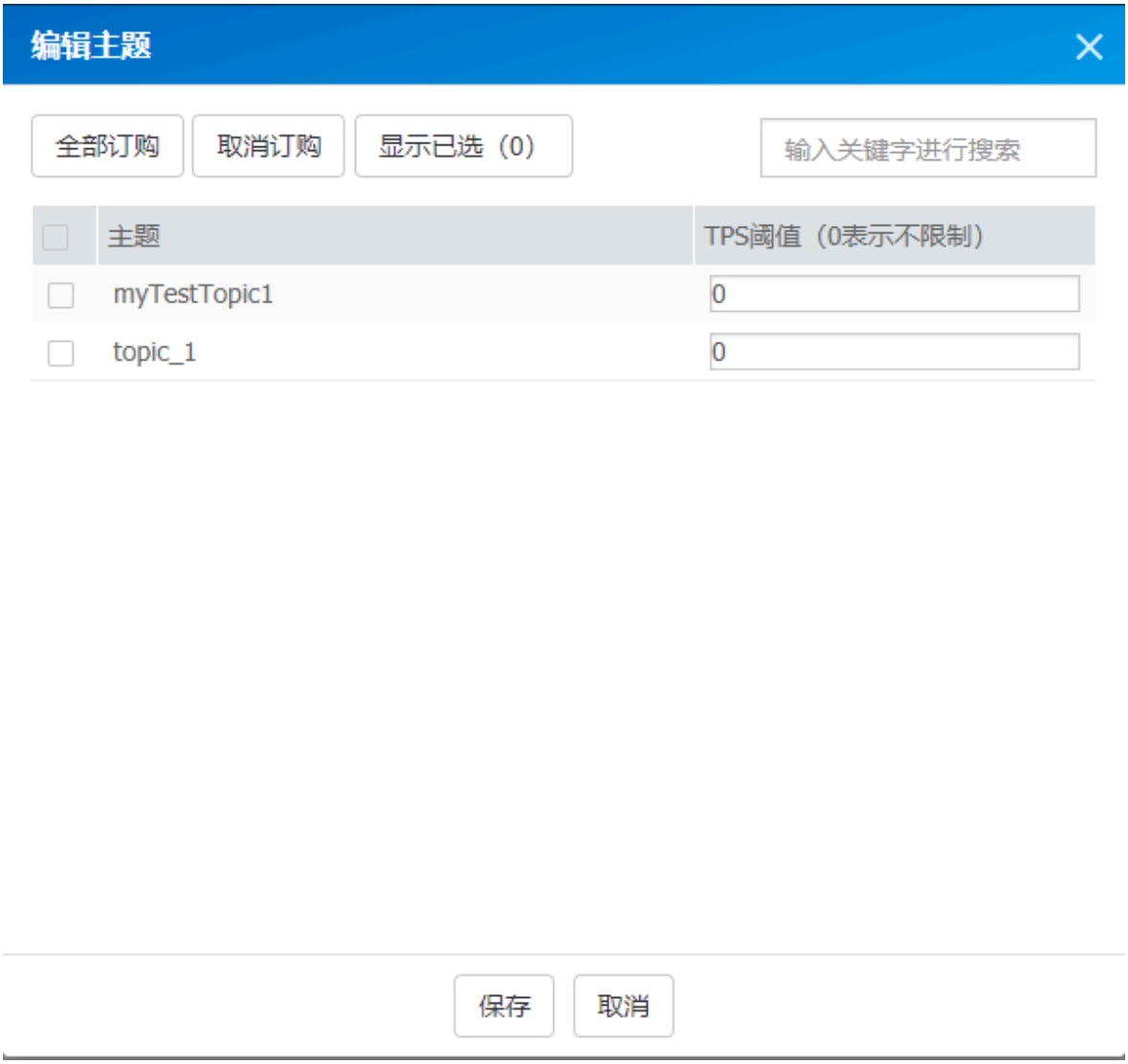
Topic列表

“Topic列表”：查看该消费组可消费的topic列表，及消费TPS限制情况。

1、 支持输入关键字进行主题数据搜索。

2、 勾选需要消费的主题，保存确认。

3、 也可选择全部订购或者取消订购。



下载

将勾选的订阅组，导出其配置及路由情况，为excel文件，导出格式如下：

A	B	C	D	E	F	G
订阅组名	起始消费位置	消费机制	集群名称	Broker组名	是否开启	Broker IP
CID_ONS-HTTP-PROXY	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
CID_ONSAPI_PULL	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
CID_ONSAPI_PERMISSION	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
testsub4	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
testsub2	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
CID_ONSAPI_OWNER	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
group_huxl_11	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
testsub3	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245
testSub1	由客户端指定	1.X版本消费机制	testyhhcluster2	brokeryhh	true	10.142.90.28:8245

备注：

如果勾选了订阅组，则导出勾选的订阅组;如果没有勾选，则导出该集群，选定broker下的所有订阅组信息。

创建消费组

场景描述

在RocketMQ中，创建消费组的场景可以有以下几种情况：

- 消息负载均衡：当一个Topic的消息量较大时，可以创建多个消费者，并将它们加入同一个消费组。RocketMQ会根据消费者的负载均衡策略，将消息均匀地分发给消费组中的消费者，从而实现消息的负载均衡。
- 高可用性：为了提高消费者的可用性和容错能力，可以创建多个消费者，并将它们加入同一个消费组。当其中一个消费者发生故障或停止运行时，RocketMQ会将该消费者的消息重新分发给消费组中的其他消费者，从而实现高可用性。
- 消费者水平扩展：当需要提高消费能力时，可以创建多个消费者，并将它们加入同一个消费组。RocketMQ会根据消费者的数量和负载均衡策略，将消息均匀地分发给消费组中的消费者，从而提高消费能力。
- 消费者组管理：通过创建消费组，可以对一组具有相同消费逻辑的消费者进行管理。例如，可以对相同业务逻辑的消费者进行分组，并为每个消费组设置不同的消费策略、消费进度等参数。

需要注意的是，创建消费组时需要为其指定一个唯一的名称，以便在RocketMQ中进行标识和管理。同时，消费组可以消费一个或多个Topic，以满足不同业务场景的需求。

总之，通过创建消费组，可以实现消息的负载均衡、高可用性和消费者水平扩展，同时方便对消费者组进行管理和配置。

新增消费组

1、进入消费组管理菜单，点击【创建消费组】按钮

2、 在弹出的新建消费组窗口填写相应字段。

1）默认展示当前集群名称，不可修改。

2）填写消费组名称，名字限制2到64个字符，超过限制会导致创建消费组失败，用户创建消费组只能包含大小写字母数字以及_和-符号。

3）按照实际需求填写消费组备注。

4）选择消费组所在的Broker，按照实例创建时候选择的主备节点对数列出每个broker，可复选。

5）选择是否开启消费，默认开启消费。

3、 完成消费组信息填写后，保存确认即可新增消费组。

4、 若希望批量创建消费组，可点击【导入消费组】按钮

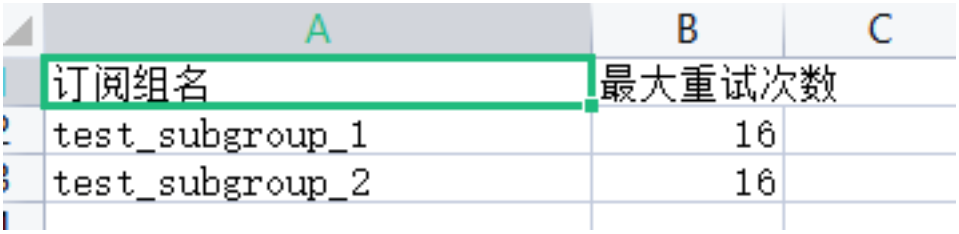
- 批量创建

注意：注意输入的订阅组名不要带空格等特殊字符

通过csv格式模板上传，批量创建订阅组。格式：点击【订阅组模板】按钮下载。

- 消费组模板

指批量上传消费组的模板，必须使用模板，才能够上传成功，模板格式如下：



	A	B	C
	订阅组名	最大重试次数	
2	test_subgroup_1	16	
3	test_subgroup_2	16	

修改消费组

场景描述

在RocketMQ中，修改订阅组的场景可以有以下几种情况：

- 增加或减少消费者：当需要增加或减少订阅组中的消费者数量时，可以通过修改订阅组来实现。例如，当消息量增加时，可以增加消费者数量以提高消费能力；当消费者数量过多时，可以减少消费者数量以降低资源消耗。
- 修改消费策略：订阅组中的消费者可以采用不同的消费策略，如集群模式或广播模式。集群模式下，每个消息只会被订阅组中的一个消费者消费；广播模式下，每个消息会被订阅组中的所有消费者都消费一次。通过修改订阅组，可以更改消费策略以满足不同的业务需求。
- 调整消费进度：订阅组中的消费者可以通过消费进度来记录已经消费的消息位置。当需要重新消费某些消息或调整消费的起始位置时，可以修改订阅组中各个消费者的消费进度。
- 修改消费者参数：订阅组中的消费者可以设置一些参数，如消费线程数、消息拉取策略等。通过修改订阅组，可以对消费者的参数进行调整，以优化消费性能和资源利用。

需要注意的是，修改订阅组时需要确保订阅组的唯一性，避免与其他订阅组冲突。同时，修改订阅组可能会影响消息的分发和消费进度，需要谨慎操作，避免消息丢失或重复消费的问题。

总之，通过修改订阅组，可以灵活调整消费者数量、消费策略、消费进度和消费者参数，以满足不同的业务需求和优化消费性能。

操作步骤

1、 进入消费组管理菜单，在消费组列表点击【编辑】按钮，进入修改消费组窗口。

2、 编辑信息，确认保存。

删除消费组

场景描述

在RocketMQ中，删除消费组的场景可以有以下几种情况：

- 业务变更：当某个业务不再需要使用消费组时，可以删除该消费组。例如，某个业务已经停止使用RocketMQ作为消息中间件，或者该业务的订阅逻辑发生了变化，不再需要使用该消费组。
- 重构订阅逻辑：当订阅逻辑发生较大变化时，可能需要删除原有的消费组，并重新创建一个新的消费组。例如，原有的消费组无法满足新的业务需求，或者需要重新设计消费组的消费者数量、消费策略等配置。
- 清理无用消费组：在RocketMQ中，可能会存在一些无用的消费组，这些订阅组可能是之前测试或开发过程中创建的，但在实际生产环境中已经不再使用。为了清理无用的消费组，可以进行删除操作，以减少资源占用和管理复杂度。

需要注意的是，在删除消费组之前，需要确保该消费组没有任何未处理的消息。如果消费组中还存在未消费的消息，删除消费组可能会导致消息丢失。因此，在删除消费组之前，建议先停止该消费组的消费者，并确保所有消息都已经被消费或者转移到其他消费组。

总之，删除消费组的场景包括业务变更、重构订阅逻辑和清理无用消费组。在删除消费组之前，需要确保没有未处理的消息，并谨慎操作，以避免消息丢失和其他问题的发生。

操作步骤

1、进入消费组管理菜单。

2、在消费组管理菜单选择将要删除的消费组，点击删除按钮，即可完成删除。

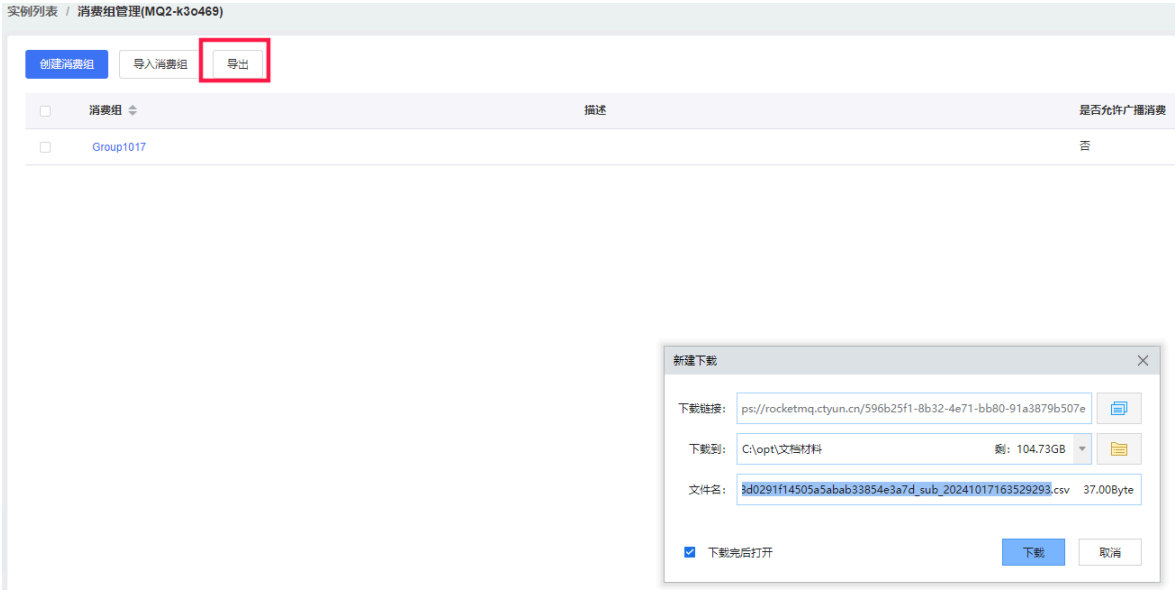
消费组导入/导出

分布式消息服务RocketMQ支持将指定实例的消费组列表信息导出，再导入至其他的RocketMQ实例，实现消费组的跨实例迁移。您可以在更换实例实例时，使用消费组导入/导出功能快速批量创建出相同的消费组。

消费组导出

1. 登录分布式消息服务RocketMQ，在左侧导航栏单击 实例列表。
2. 在顶部菜单栏选择地域，如华东1，然后在实例列表中，单击目标实例名称。
3. 在左侧导航栏单击消费组管理。
4. 在消费组管理页面选择需导出的消费组，点击导出。

   系统会自动将该实例下的消费组资源列表数据导出并保存为 .csv文件。



消费组导入

您可以将已导出的消费组列表直接导入至目标实例中，也可以根据实际需求更新列表内容再导入消费组信息。

1. 登录分布式消息队列RocketMQ控制台，在左侧导航栏单击 实例列表。
2. 在顶部菜单栏选择地域，如华东1，然后在实例列表中，单击目标实例名称。
3. 在左侧导航栏单击消费组管理。
4. 在消费组管理页面右上角单击导入消费组。
5. 选择在本地保存的消费组资源列表文件，然后单击打开，点击确认。

6. 导入完成后，导入的消费组会出现在消费组列表中。

查看订阅关系

在分布式消息服务RocketMQ控制台，您可实时查看消费组和Topic之间的订阅关系，即某个Topic被哪些消费组订阅了，以及某个消费组订阅了哪些Topic。

前提条件

消费组处于在线状态。

什么是订阅关系

分布式消息服务RocketMQ里的一个消费组代表一个Consumer实例群组。对于大多数分布式应用来说，一个消费组下通常会挂载多个Consumer实例。分布式消息服务RocketMQ的订阅关系主要由Topic+Tag共同组成，即一个消费组下所有的Consumer实例订阅的Topic以及这些Topic中的过滤规则Tag。

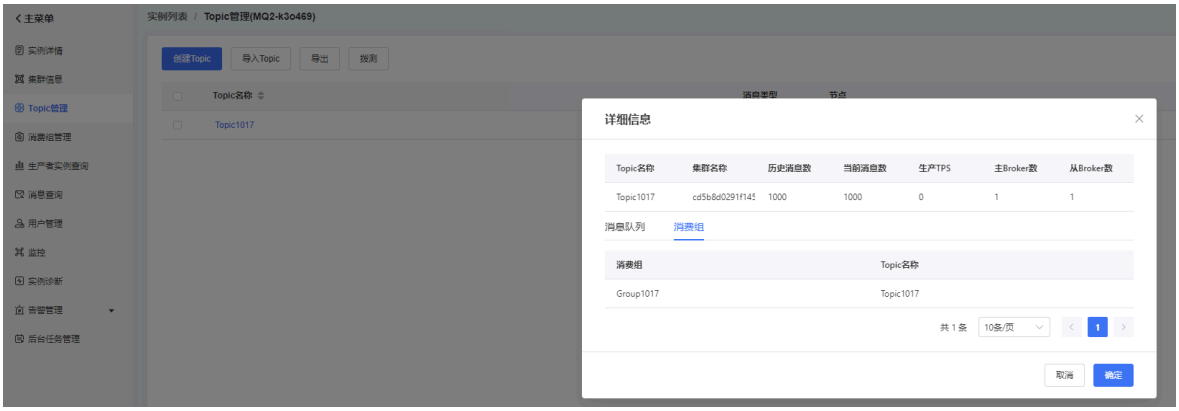
同一个消费组下所有的Consumer实例需保持订阅关系一致，否则，消息消费的逻辑就会混乱，甚至导致消息丢失。

查看Topic被哪些消费组订阅

1. 登录分布式消息服务RocketMQ控制台，在左侧导航栏单击实例列表。
2. 在顶部菜单栏选择地域，如华东1，然后在实例列表中，单击目标实例名称。

3. 在左侧导航栏单击Topic管理 ，然后在Topic列表单击目标Topic名称。

   您可查看到该Topic的所有在线消费组。示例如下。



查看消费组订阅的Topic

1. 登录分布式消息服务RocketMQ控制台，在左侧导航栏单击 实例列表。
2. 在顶部菜单栏选择地域，如华东1，然后在实例列表中，单击目标实例名称。
3. 在左侧导航栏单击消费组管理 ，然后在消费组列表单击目标消费组名称。您可查看到该消费组的订阅关系，示例如下。

用户权限管理

场景描述

RocketMQ提供了用户管理和权限管理机制，用于确保消息队列的安全性和访问控制。另外RocketMQ还支持角色的概念，可以将多个权限组合成一个角色，并将角色分配给用户。通过角色管理，可以简化权限管理的复杂性，提高管理效率。

通过用户管理和权限管理机制，RocketMQ可以实现对消息队列的细粒度访问控制，确保只有授权的用户能够读取、写入和订阅消息，并提供了灵活的角色管理功能，方便管理员进行权限分配和管理。这些机制可以帮助用户保护消息队列的安全性，防止未经授权的访问和操作。

应用用户管理

本章节适用于南京3、上海7、重庆2、乌鲁木齐27、保定、石家庄20、内蒙6、晋中、北京5 节点。

新建消息实例后，必须在此菜单新建应用用户，然后应用才能在此消息实例上发送、消费消息。此处的用户名即为管控openapi中的accessKey，加密后的密码即为管控openapi中的secretKey。

应用用户：指MQ客户端，连接服务器生产消费时，需要进行权限校验，所以MQ客户端的用户，称为应用用户；

除了用户密码的校验，还可以为用户指定topic，代表该用户只能生产消费，指定的topic，其他topic不能生产消费。

1、点击【新建用户】按钮

2）进入用户列表界面，新增用户

3）弹出框填写用户字段

- 默认展示租户名，不可修改。
- 选择集群名称，填写应用用户名，请输入大于6位字符，只能输入大小写字母，下划线，数字。
- 填写用户密码，请输入大于8位字符，需要包含数字大小写字母以及特殊符号(!@#$%^&*)。
- 按照实际需求填写描述。

4）设置用户主题或订阅组权限

点击“主题权限”或“订阅组”，可以设置该用户的主题或订阅组发布或订阅权限：

5）选择主题名称及对应权限，PUB代表生产权限 SUB代表消费权限，DENY代表无任何权限，用|符号相连即表示两种权限都有 如PUB|SUB。

6）选择订阅组名称及对应权限，权限说明同上。

（2）新模式维护用户及权限

角色控制

以下适用于华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7 节点。

1）点击【新建用户】按钮

2）弹出框填写用户字段

- 填写应用用户名，请输入大于6位字符,只能输入大小写字母，下划线，数字。
- 填写密钥，请输入大于8位字符,需要包含数字大小写字母以及特殊符号(!@#$%^&*)。
- 选择默认主题权限，PUB代表生产权限 SUB代表消费权限，DENY代表无任何权限，用|符号相连即表示两种权限都有 如PUB|SUB。
- 选择默认订阅组权限，权限说明同上。

权限管理

主子账号

本章介绍如何使用主子账号体系管理子账号权限

概述

分布式消息服务RocketMQ已接入主子账号体系，可区分两种账号权限，实现主账号对子账号的数据权限管理与功能权限管理，支持系统策略和自定义策略的授权方式。本章介绍如何使用主子账号体系管理子账号权限。

背景

1. 主账号 ：用户在天翼云注册后自动创建，该账号对其所拥有的资源具有完全的访问权限，可以重置用户密码、分配用户权限等。如果需要多人共同使用天翼云资源，由于账号是付费主体为了确保账号安全，建议创建子用户来进行日常管理工作。
2. 子账号 ：主账号认证为企业账号后，在天翼云用户中心页面创建出来的账号。子账号的用户名、密码统一由主账号创建管理。子账号同样可以登录访问天翼云控制台，登录入口与主账号相同，受主账号赋予的权限限制。

3. 企业项目： 将云资源、企业成员按项目进行管理，通过企业项目将云资源、带有权限的用户组绑定到一起，用户使用项目内云资源的权限受用户组的授权限制。

注意

一个实例只能归属一个企业项目（可变更），一个子账号可以同时在多个企业项目中。

4. 策略： 是描述一组权限集的语言，它可以精确地描述被授权的资源集和操作集，通过策略，用户可以自由搭配需要授予的权限集。通过给用户组授予策略，用户组中的用户就能获得策略中定义的权限。
5. 系统策略： 系统预置的常用权限集，主要针对不同云服务的只读权限或管理员权限，比如对组件的只读权限、普通用户权限和管理员权限等等；系统策略只能用于授权，不能编辑和修改。
6. 数据权限： 看到的数据不一样。主账号看到所有实例，子账号只能看到所属项目中的实例。
7. 功能权限： 主账号可以进行所有控制台操作，子账号对单个组件实例拥有的操作权限由主账号授权。
8. 功能权限授权： 给子账号在企业项目A下增加一个策略，即代表该子账号对企业项目A下的实例拥有了策略中定义的权限，策略以外的操作会被禁止。

数据权限控制

数据权限的权限码为统一值instance-list，策略中包含instance-list的权限码才具备查看实例的权限。如果在用户组直接授权包含instance-list的策略，则该用户组的子用户能看到所有实例。如果在企业项目中的用户组授权包含instance-list的策略，则该用户组的子用户只能看到该企业项目下的实例。

操作步骤：

1. 进入IAM管理页面：

- 登录天翼云官网，鼠标悬浮至右上角个人信息，点击个人信息进入账号中心页面。
- 在账号中心页面中，点击统一认证服务进入IAM管理页面。

2. 创建用户组：

- 在IAM管理页面中，点击左侧菜单栏中用户组进入用户组管理页面。
- 在用户组管理页面中，点击创建用户组按钮，在弹出框中填写用户组名称与描述，点击确定即可。

3. 创建子用户：

- 在IAM管理页面中，点击左侧菜单栏中用户进入用户管理页面。
- 在用户管理页面中，点击创建用户按钮，进入创建页面。
- 在创建页面中，首先需要配置用户基本信息。填写用户名称、手机号、邮箱和密码等信息，点击下一步加入用户组。
- 在加入用户组页面，选择对应的用户组，点击添加按钮加入已选用户组，点击下一步即可。

4. 创建企业项目：

- 在IAM管理页面中，点击左侧菜单栏中企业项目进入企业项目管理页面。
- 在企业项目管理页面中，点击创建企业项目按钮，在弹出框中填写企业项目名称与描述，点击确定即可。

5. 在企业项目中添加用户组：

- 在企业项目管理页面中，点击企业项目的查看用户组按钮进入企业项目的用户组管理页面。
- 在企业项目的用户组管理页面中，点击设置用户组按钮，弹出设置用户组弹框。
- 在弹出框中，选择用户组再点击>按钮加入已选用户组，最后点击确定按钮即可。

6. 对企业项目中的用户组设置策略：

- 在企业项目管理页面中，点击企业项目的查看用户组按钮进入企业项目的用户组管理页面。
- 在企业项目的用户组管理页面中，点击用户组的设置策略按钮弹出设置策略弹框。
- 在弹出框中选择对应策略，点击>按钮加入已选策略，最后点击确定按钮即可。

功能权限控制

主账号对子账号的功能权限控制是通过给用户组授权策略实现的，策略包括系统策略和自定义策略。策略中可定义“允许”的操作和“拒绝”的操作，“拒绝”的优先级大于“允许”。

操作步骤：

1. 进入IAM管理页面：

- 登录天翼云官网，鼠标悬浮至右上角个人信息，点击“个人信息”进入账号中心页面。
- 在账号中心页面中，点击“统一认证服务”进入IAM管理页面。

2. 创建用户组：

- 在IAM管理页面中，点击左侧菜单栏中“用户组”进入用户组管理页面。
- 在用户组管理页面中，点击“创建用户组”按钮，在弹出框中填写用户组名称与描述，点击确定即可。

3. 创建子用户：

- 在IAM管理页面中，点击左侧菜单栏中“用户”进入用户管理页面。
- 在用户管理页面中，点击“创建用户”按钮，进入创建页面。
- 在创建页面中，首先需要配置用户基本信息。填写用户名称、手机号、邮箱和密码等信息，点击下一步加入用户组。
- 在加入用户组页面，选择对应的用户组，点击“添加”按钮加入已选用户组，点击下一步即可。

4. 授权：

- 在IAM管理页面中，点击左侧菜单栏中“用户组”进入用户组管理页面。
- 在用户组管理页面中，点击对应用户组的“授权”按钮，进入授权页面。
- 在授权页面中，勾选对应策略，点击下一步进入设置授权范围页面。
- 在设置授权范围页面中，选择授权范围，点击“确定”按钮就可完成授权。

管理消息

消息查询

场景描述

RocketMQ查询消息信息的作用如下：

● 监控消息状态：通过查询消息信息，可以实时监控消息的状态，包括消息是否已被消费、消费进度、重试次数等。这有助于及时发现消息消费异常或延迟等问题，以便进行及时处理和调整。

● 故障排查与追踪：通过查询消息信息，可以帮助定位消息消费失败的原因，如消费者异常、网络故障等。同时，还可以追踪消息的消费路径，了解消息从生产到消费的流程，方便排查故障和进行问题定位。

● 统计与分析：通过查询消息信息，可以进行消息的统计和分析，如消息的发送量、消费量、消费延迟等。这有助于了解系统的消息处理情况，评估系统的性能和稳定性，以便进行相应的优化和改进。

● 数据同步与恢复：通过查询消息信息，可以了解消息的发送时间、内容和关键字等信息，方便进行数据的同步和恢复。当系统发生故障或数据丢失时，可以通过查询消息信息来恢复数据，并确保数据的一致性和完整性。

综上所述，通过查询RocketMQ中的消息信息，可以实现消息的监控、故障排查、统计分析以及数据同步与恢复等功能，为系统的稳定运行和数据管理提供了重要的支持。

操作步骤

1、 进入管理控制台消息查询菜单。

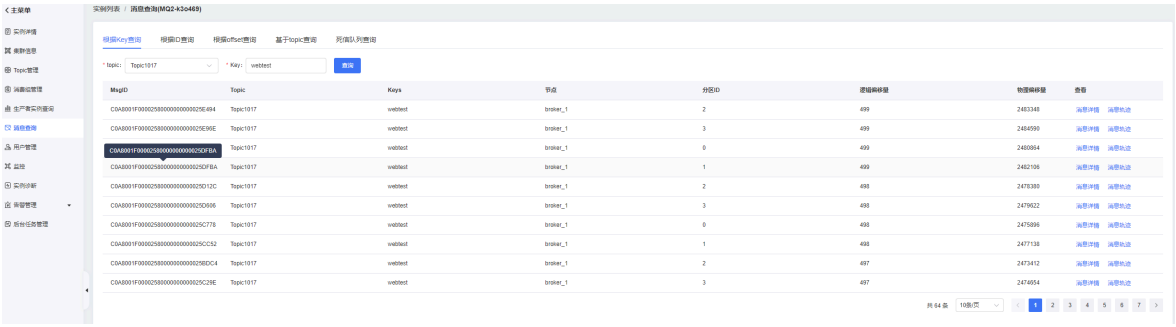
2、 下拉选择集群名称和broker名称。

3、 提供五种查询消息的方式：按key，按ID，按偏移量，基于Topic查询，死信队列查询。

根据Key查询

根据消息的key查询消息列表，key要求尽可能全局唯一。

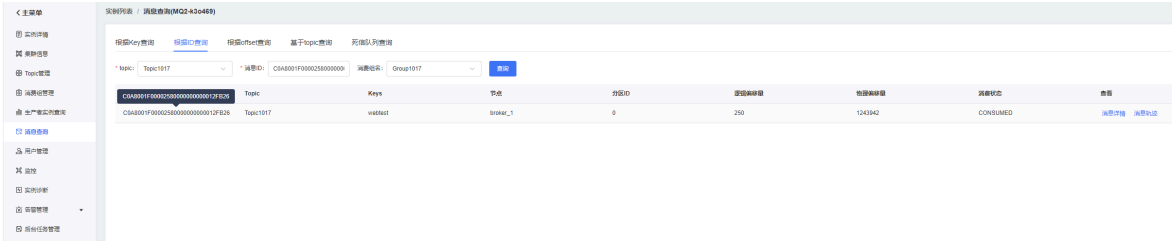
点击“查看”，可以查询该消息的包体内容。



根据ID查询

根据消息ID查询唯一消息，选择消费组后，可以查询到，该消息是否被该消费组消费过，查看“消费状态”。

点击“查看”，可以查询该消息的包体内容。



消费状态标志含义：

（1）To-consume：未消费。

（2）Consumed：已签收。

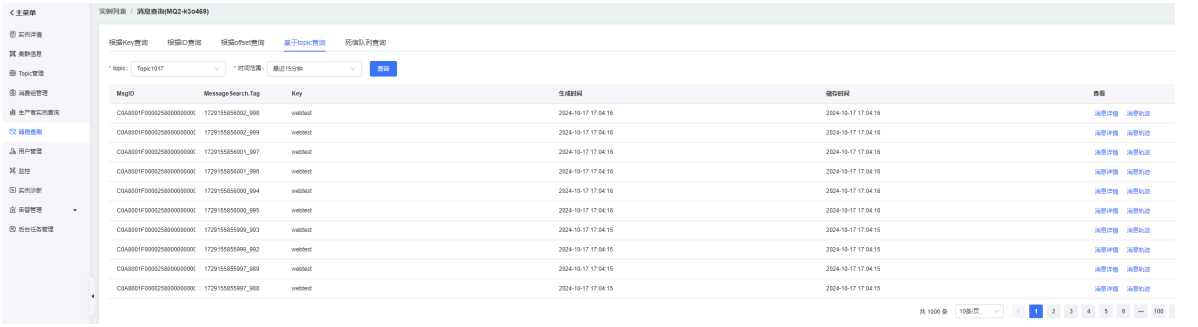
（3）Consuming：已拉取，未签收。

根据offset查询

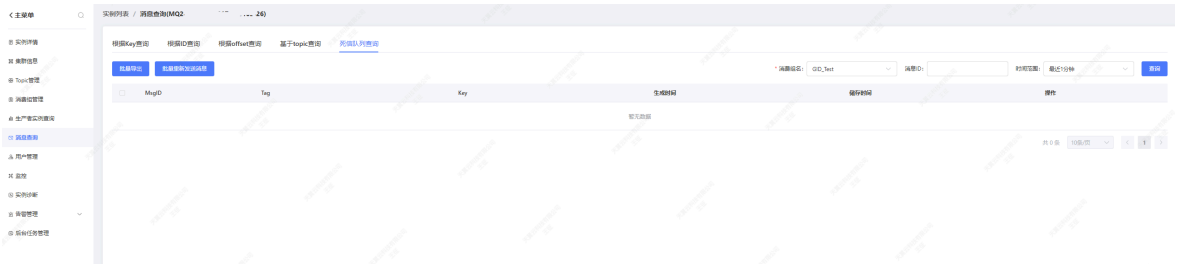
根据指定队列指定偏移量查询唯一消息，选择订阅组后，可以查询到，该消息是否被该订阅组消费过，查看“消费状态”。

点击“查看”，可以查询该消息的包体内容。

基于Topic查询



死信队列查询



消息轨迹

场景描述

消息轨迹记录了一条消息从生产端到消息队列RocketMQ服务端，再到消费端的整个过程，包括各阶段的时间、执行状态等。

使用说明

客户端SDK默认不开启消息轨迹功能，需要在消息收发时主动开启消息轨迹，以Java为例，如下所示：

- 生产者启用消息轨迹：

```
DefaultMQProducer producer =new
DefaultMQProducer("ProducerGroupName", new
AclClientRPCHook(newSessionCredentials("控制台角色AK", "控制台角色CK")), true, null);
```

- 消费者启用消息轨迹：

```
DefaultMQPushConsumer consumer =new
DefaultMQPushConsumer("ConsumerGroupName",new
AclClientRPCHook(
        new SessionCredentials(
           "控制台角色AK",
           "控制台角色CK")
    ),newAllocateMessageQueueAveragely(),true,null);
```

消息轨迹的查询方式

1. 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。
2. 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。
3. 进入实例列表，点击【管理】按钮进入管理菜单。
4. 根据查询结果，查看消息轨迹。

消息重试

消费者出现异常，分布式消息服务RocketMQ会根据消费重试策略重新投递该消息进行故障恢复。本文主要介绍分布式消息服务RocketMQ中消息重试与使用方法。

功能介绍

当消息第一次被消费者消费后，没有得到正常的回应，或者用户要求服务端重投，分布式消息服务RocketMQ会通过消费重试机制自动重新投递该消息，直到该消息被成功消费，当重试达到一定次数后，消息仍未被成功消费，则会停止重试，将消息投递到死信队列中。

当消息进入到死信队列中，表示分布式消息服务RocketMQ集群已经无法自动处理这批消息，一般这时就需要人为介入来处理这批消息。客户可以通过编写专门的客户端来订阅死信Topic，处理这批之前处理失败的消息。

消息重试次数

- 默认值：16次。
- 最大限制：不超过1000次。

最大重试次数由消费者分组的元数据控制，修改方式，请参见修改最大重试次数。

例如，最大重试次数为3次，则该消息最多可被投递4次，1次为原始消息，3次为重试投递次数。

说明

只有当消费模式为集群消费模式时，Broker 才会自动进行重试，广播消费模式下不会进行重试。

出现以下三种情况会按照消费失败处理并会发起重试：

1、消费者返回 ConsumeResult.FAILURE。

2、消费者返回 null。

3、消费者主动/被动抛出异常。

死信消息

对于消费失败且重试后依然失败的消息，分布式消息服务RocketMQ不会立即丢弃，而是将消息转发至指定的队列中，即死信队列，这些消息即为死信消息。当消费失败的原因排查并解决后，您可以重新投递这些死信消息，让消费者重新消费；若您暂时无法处理这些死信消息，为避免到期后死信消息被删除，您也可以先将死信消息导出进行保存。

特性说明

死信消息具有如下特性：

- 不会再被消费者正常消费。

  有效期与正常消息相同，默认保留168小时。超过默认的168小时后，会被自动删除。

死信队列具有如下特性：

- 一个死信队列对应一个消费组，而不是对应单个消费者实例。

  如果一个消费组未产生死信消息，分布式消息服务RocketMQ不会为其创建相应的死信队列。

  一个死信队列包含了对应消费组产生的所有死信消息，不论该消息属于哪个Topic。

查询死信消息的方式

登录天翼云分布式消息服务RocketMQ控制台。

单击RocketMQ实例的名称，进入实例详情页面。

在左侧导航栏，单击“消息查询 > 死信队列查询”，进入“死信队列”页面。

选择消费组名、消息ID、时间范围查询死信消息。

RocketMQ提供的查询死信消息的方式。

查询方式	查询条件	说明
按Group查询	Group+时间段	根据消费组和时间范围，批量获取符合条件的所有死信消息，查询量大，不易匹配。
按消息ID查询	Group+消息ID	根据消费组和消息可以精确定位任意一条死信消息，获取死信消息的属性。

生产者实例查询

场景描述

RocketMQ显示当前在线的生产者实例有以下作用：

- 监控生产者状态：通过显示当前在线的生产者实例，可以实时监控生产者的状态。可以了解生产者的连接情况、运行状态和发送速率等信息，帮助及时发现生产者异常或故障，以便进行及时处理和调整。
- 故障排查与追踪：通过显示当前在线的生产者实例，可以帮助定位消息发送失败的原因。可以查看每个生产者实例的运行情况，包括发送的消息数量、发送延迟等信息，方便排查故障和进行问题定位。

综上所述，显示当前在线的生产者实例可以帮助监控生产者状态、故障排查等，为系统的稳定运行和性能调优提供重要支持。

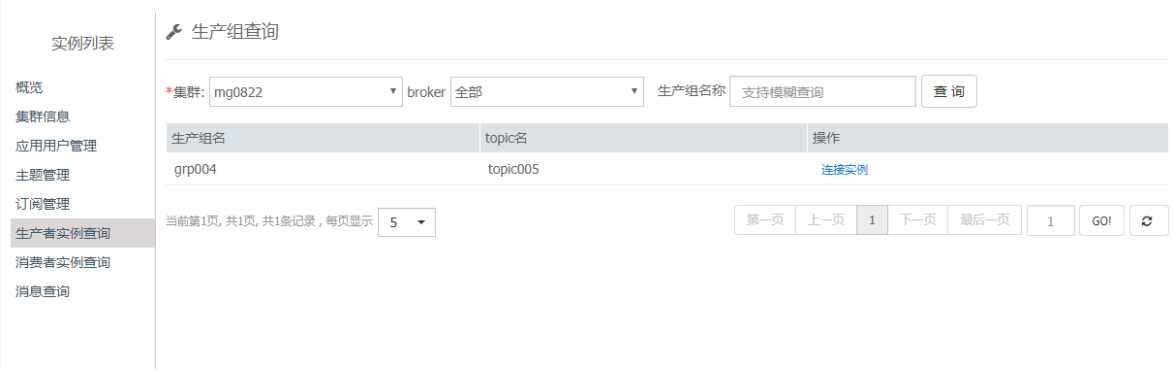
操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

进入实例列表，点击【管理】按钮进入管理菜单。

3、 进入生产者实例查询菜单，列出了指定集群，指定broker下的生产组和关联的topic的情况。



列表展示了指定集群和指定broker下的生产组和关联topic的情况。

连接实例

显示该生产组，当前在线的生产者实例列表。

消费者实例查询

场景描述

RocketMQ显示当前在线的消费者实例有以下作用：

- 监控消费者状态：通过显示当前在线的消费者实例，可以实时监控消费者的状态。可以了解消费者的连接情况、消费进度和消费速率等信息，帮助及时发现消费者异常或故障，以便进行及时处理和调整。
- 故障排查与追踪：通过显示当前在线的消费者实例，可以帮助定位消息消费失败的原因。可以查看每个消费者实例的消费情况，包括消费的消息数量、消费延迟等信息，方便排查故障和进行问题定位。

综上所述，显示当前在线的消费者实例可以帮助监控消费者状态、故障排查等，为系统的稳定运行提供重要支持。

操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

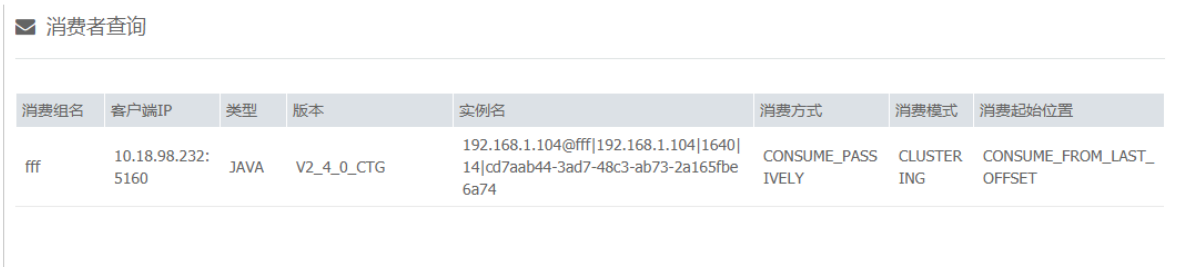
进入实例列表，点击【管理】按钮进入管理菜单。

3、 进入消费者实例查询菜单，列出了指定集群和broker下的消费组消费的情况。

列表展示了指定集群和指定broker下的消费组消费的情况。

连接实例

显示该消费组，当前在线的消费者实例列表。



堆积量

显示该消费组消费指定topic时，还有多少未消费。

监控与告警

监控指标说明

生产者指标

指标名称	指标说明	单位
生产TPS	统计Topic的消息生产速率，计算方式：1分钟内的最大值	条/秒
生产消息量峰值	统计消息生产速率的最大值	条/秒
累计生产消息量	统计所选时间段内所选topic累计生产的消息总量	条

消费者指标

指标名称	指标说明	单位
处理中消息量	计算选定的topic和group当前消费者客户端正在消费但是还没有返回消费成功响应到服务端的消息数。	条
已就绪消息量	计算选定的topic和group当前在服务端已经就绪可以被消费消费的消息总量，这部分消息消费者客户端还没有开始消费。	条
堆积消息量	计算选定topic和group当前消息堆积总量，包括处理中消息和已就绪消息。	条
已就绪消息排队时间	计算选定的topic和group最早一条就绪消息的就绪时间和当前时间差，数值面板展示取选定时间段内统计的最大值展示，曲线面板展示选定时间范围的序列值，当该订阅组没有在线时，该值不显示。该指标可以观测还未被处理的消息的延迟时间大小，适用于对消息延时时间比较敏感的业务场景。	毫秒，但随着数值增大会自适应变换单位
消息消费速率	计算选定topic和Group消费消息的速率。	条/秒
消费者速率峰值	计算所选定topic以及group的消息消费速率的最大值。	条/秒
消费堆积量	包含上面的堆积消息量，处理中消息量，已就绪消息量，以曲线的形式展示。	条
消息消费处理耗时	计算所选topic以及group消费时，从消息开始被消费到消费完成的处理耗时。	毫秒

指标名称	指标说明	单位
消息生产速率 top20 topics	展示消息生产速率最高的前20个Topic生产速率曲线。	条/秒
消息消费速率 top20 groups	展示消息消费速率最高的前20个group消费速率曲线。	条/秒
已就绪消息量 top20 Groups	计算已就绪消息量最大的前20个Group。	条
已就绪消息排队时间 top20 Groups	计算已就绪消息量最大的前20个Group。	条
堆积消息量（包含已就绪消息以及处理中消息）top20 group	统计堆积的消息量最多的前20个Group。	条
处理中消息量 top20 Groups	计算处理中的消息量最多的前20个Group。	条
消费处理耗时 top20 Groups	计算消费处理耗时最长的前20个Group。	毫秒

节点指标

指标名称	指标说明
限流次数	计算触发限流次数，包括生产限流、消费限流
磁盘读写速率	计算磁盘读写速率，单位：Byte/s
磁盘使用率	计算磁盘使用率
磁盘IOPS	计算磁盘单位时间内的IO次数，单位：次/s

集群监控

场景描述

RocketMQ集群监控的场景描述如下：

- 集群状态监控：通过监控RocketMQ集群的状态，可以实时了解集群的运行情况。可以监控集群中各个角色（如NameServer、Broker、Producer、Consumer）的状态，包括连接数、运行状态、消息发送和消费速率等信息，以便及时发现集群中的异常或故障。
- 故障排查与处理：通过集群监控，可以帮助快速排查和处理故障。当集群中出现异常情况时，可以通过监控数据进行故障定位，找出导致故障的原因，并及时采取相应的措施进行处理，以保证集群的稳定运行。
- 负载均衡与调优：集群监控可以帮助进行负载均衡和性能调优。可以监控集群中各个节点的负载情况，包括消息发送和消费的速率、资源利用率等信息，根据实际情况进行资源调配和负载均衡，以确保集群的高性能和稳定性。
- 容量规划和预测：通过集群监控数据，可以进行容量规划和预测。可以监控集群中的消息存储量、消息堆积情况等信息，根据历史数据和趋势进行容量规划，以便提前做好资源扩展和调整的准备。
- 性能优化和系统调优：通过集群监控，可以分析和评估系统的性能瓶颈和瓶颈原因，以便进行系统调优。可以监控集群中各个节点的性能指标，如处理延迟、吞吐量等，找出性能瓶颈并进行相应的优化，以提升系统的性能和效率。

综上所述，RocketMQ集群监控可以帮助实时监控集群状态、故障排查、负载均衡和性能优化等，为保证集群的稳定运行和提升系统性能提供重要支持。

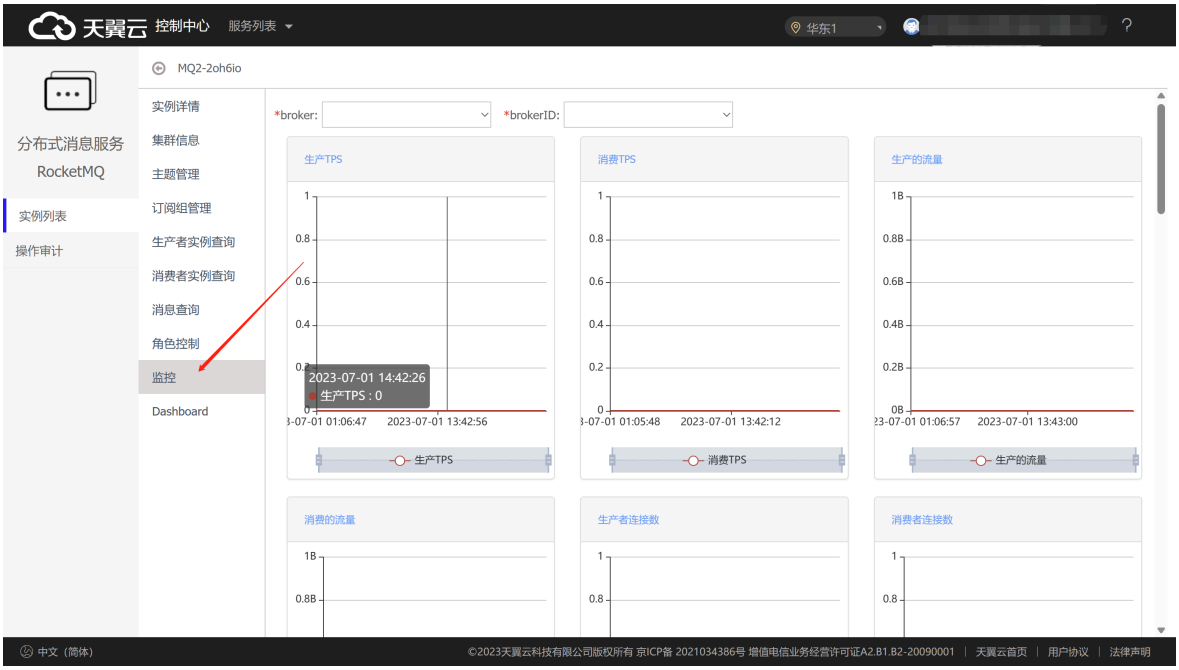
操作步骤

1、 天翼云官网点击控制中心，选择产品分布式消息服务RocketMQ。

2、 登录分布式消息服务RocketMQ控制台，点击右上角地域选择对应资源池。

进入实例列表，点击【管理】按钮进入管理菜单。

3、 进入监控菜单，下拉选择broker和brokerID，查看集群的监控信息：



仪表盘

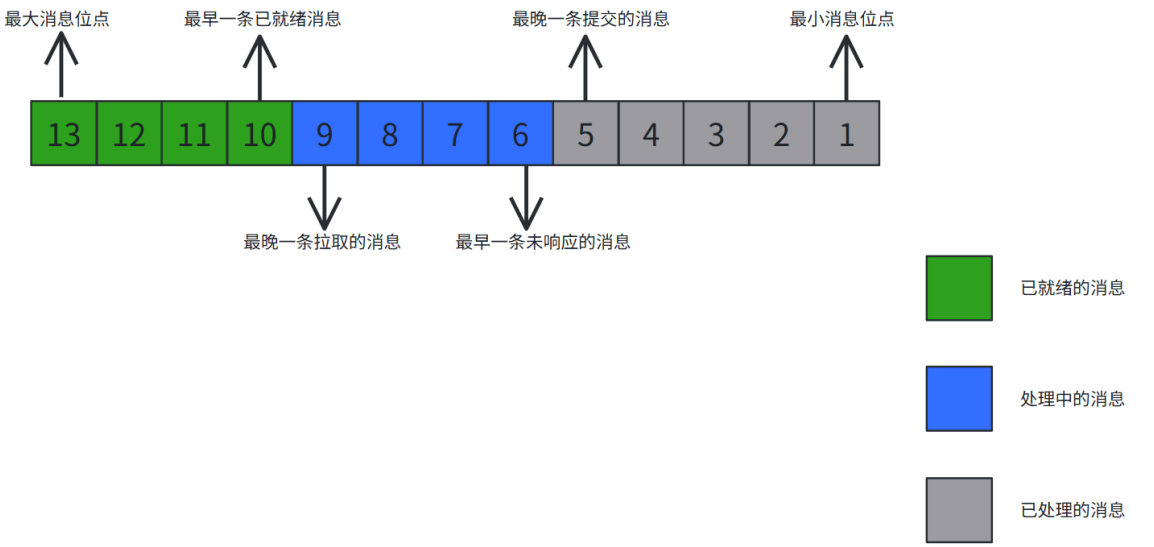
应用场景

- 场景一：线上消息消费有异常，消息不能及时被处理，需要及时收到报警，并快速定位问题。
- 场景二：线上某些订单状态有异常，需要排查对应的消息链路环节是否正常发送消息。
- 场景三：需要分析消息流量变化趋势、流量分布特点或消息体量，进而进行业务趋势分析规划。
- 场景四：需要查看和分析应用上下游依赖拓扑情况，进行架构升级优化或改造。

相关概念

查看仪表盘指标前，您需要了解以下涉及消息堆积的指标概念。

如下图所示，表示指定主题的某一队列中各消息的状态



上图表示指定主题的某一队列中各消息的状态，分布式消息服务RocketMQ将处于不同处理阶段的消息数量和耗时进行统计，这些指标可直接反映队列中消息的处理速率和堆积情况，通过观察这些指标可初步判断业务的消费是否异常。具体的指标含义和计算公式如下：

分类	指标	定义	计算公式
消息数量指标	处理中的消息（inflight messages）	在消费者客户端正在处理，但客户端还未返回消费结果的消息。	最晚一条拉取消息的位点－最晚一条提交消息的位点
消息数量指标	已就绪的消息（ready messages）	消息在云消息队列 RocketMQ 版服务端已就绪，对消费者可见可被消费的消息。	最大消息位点－最晚一条拉取消息的位点
消息数量指标	消息堆积量（consumer lag）	所有未处理完成的消息量。	处理中消息量＋已就绪消息量
消息耗时指标	已就绪消息的就绪时间（ready time）	普通消息、顺序消息：消息存储到服务端的时间。 定时/延时消息：消息定时或延时结束的时间。* 事务消息：事务提交的时间。	不涉及
消息耗时指标	已就绪消息的排队时间（ready message queue time）	最早一条就绪消息的就绪时间和当前时刻的时间差。该时间反映消费者拉取消息的及时性。	当前时间－最早一条就绪消息的就绪时间
消息耗时指标	消费处理滞后时间（comsumer lag time）	最早一条未消费完成的消息的就绪时间和当前时刻的时间差。该时间反映消费者完成消息处理的及时性。	当前时间－最早一条未提交消息的就绪时间

生产者

指标项	说明
消息生产速率	统计Topic的消息生产速率和消息生产时的API调用速率。单位：*消息速率：条/秒* API调用速率：次/秒
累计生产消息量	统计指定实例下生产的消息总量。单位：条。

消费者

指标项	说明
平均消费成功率	统计指定实例下所有消息的消费成功率。
堆积消息量（已就绪+处理中）	统计指定实例下的消息堆积总量，包括已就绪的消息和处理中的消息。单位：条。
处理中消息量	统计在消费者客户端正在处理但客户端还未返回消费成功响应的消息的数量。单位：条。
已就绪消息量	统计在云消息队列 RocketMQ 版服务端已就绪，可以被消费者消费的消息数量。指标反映还未被消费者开始处理的消息规模。单位：条。
已就绪消息排队时间	统计最早一条就绪消息的就绪时间和当前时间差。该指标反映了还未被处理的消息的延迟时间大小，对于时间敏感的业务来说是非常重要的度量指标。总览中的指标值表示指定实例下的已就绪消息排队时间的平均值；具体图表中的指标值表示指定Group订阅指定某Topic下的已就绪消息排队时长。单位：毫秒。
消息消费速率	统计Group消费消息的速率。单位：条/秒
消息消费速率峰值	统计消息消费速率的最大值。单位：条/秒
累计消费消息量	统计指定实例下所有消费的消息总量。单位：条。
消费堆积量	统计Group消费消息时的消息堆积量，包括已就绪消息和处理中消息。单位：条。

实例Top 20概览

指标项	说明
消息生产速率 top20 Topics	统计消息生产速率最快的前20个Topic。单位：条/秒。
消息消费速率 top20 GroupIDs	统计消息消费速率最快的前20个Group。单位：条/秒。
已就绪消息量 top20 GroupIDs	统计已就绪消息量最多的前20个Group。单位：条。
已就绪消息排队时间 top20 GroupIDs	统计已就绪消息排队时间最长的前20个Group。单位：毫秒。
堆积消息量（已就绪消息+处理中消息）top20 GroupIDs	统计堆积的消息量最多的前20个Group。单位：条。
处理中消息量 top20 GroupIDs	统计处理中的消息量最多的前20个Group。单位：条。

仪表盘常见问题

如何理解实例的TPS均值和TPS Max值？

- TPS均值＝1分钟内总的请求次数/60秒
- TPS Max值：以1分钟为统计周期，每秒采样一次TPS值，统计结果取这60个采样值的最大值。

具体示例如下：

假设某实例在1分钟内生产60条消息（均为普通消息、每条4 KB大小），则实例的生产速率是 60 条/分钟。

实例TPS均值 = 60 次/ 60 秒 = 1 次/秒

实例TPS Max值计算如下：

- 如果这60条消息在第1秒发送完成，则该实例在这1分钟内每秒的TPS分别为60、0、0.....0。

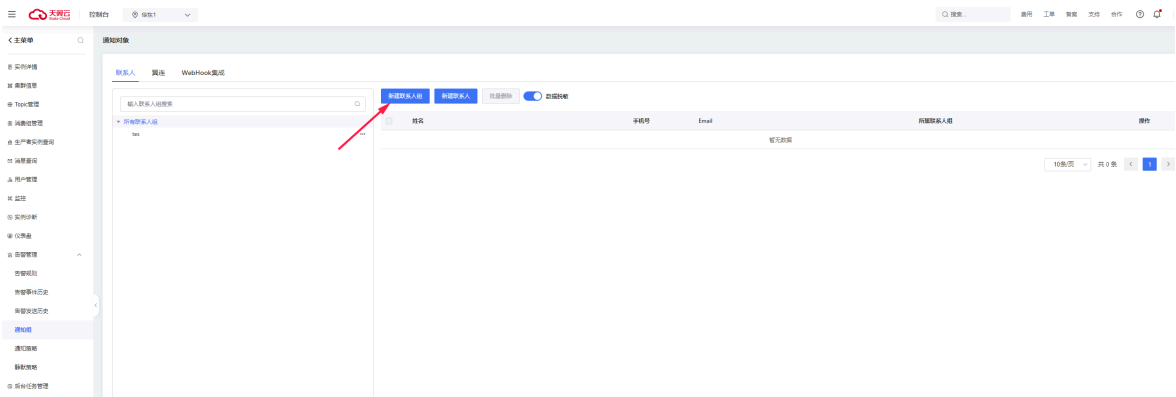
  实例TPS Max值=60次/秒。
- 如果这60条消息在第1秒发送了40条，第2秒发送了20条，则该实例在这1分钟内每秒的TPS分别为40、20、0、0.....0。

  实例TPS Max值=40次/秒。

配置告警

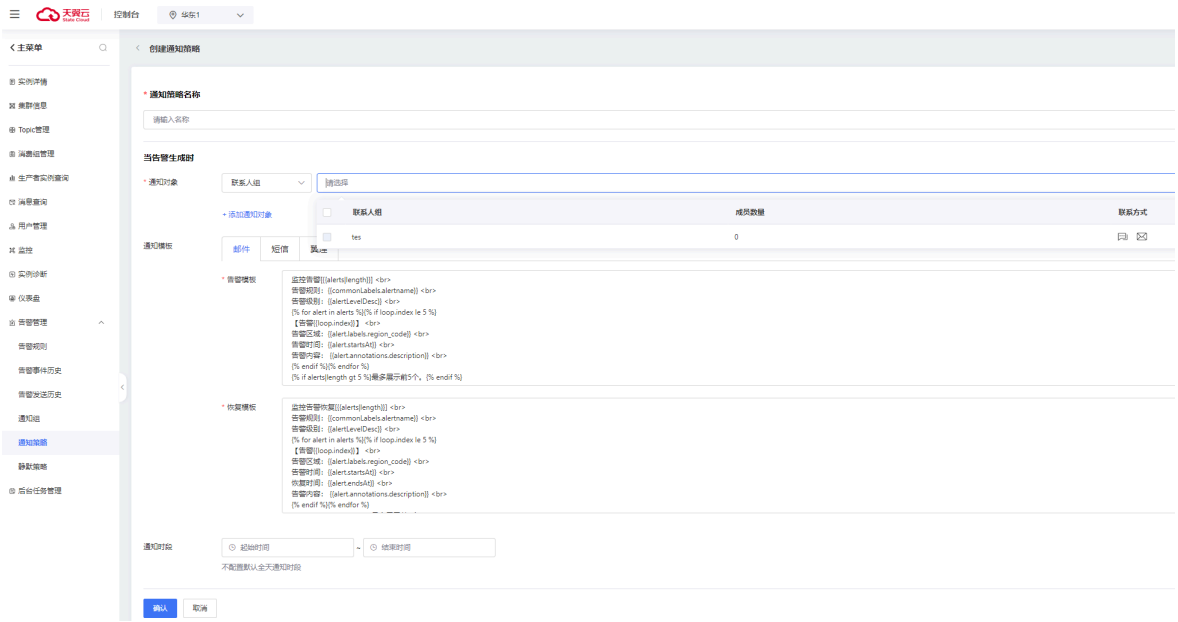
创建联系人组

进入分布式消息服务RocketMQ管理控制台，在实例列表页点击管理进入实例详情页，左侧菜单点击告警管理点击通知组菜单，右侧点击新建联系人组，输入联系人信息信息，点击保存



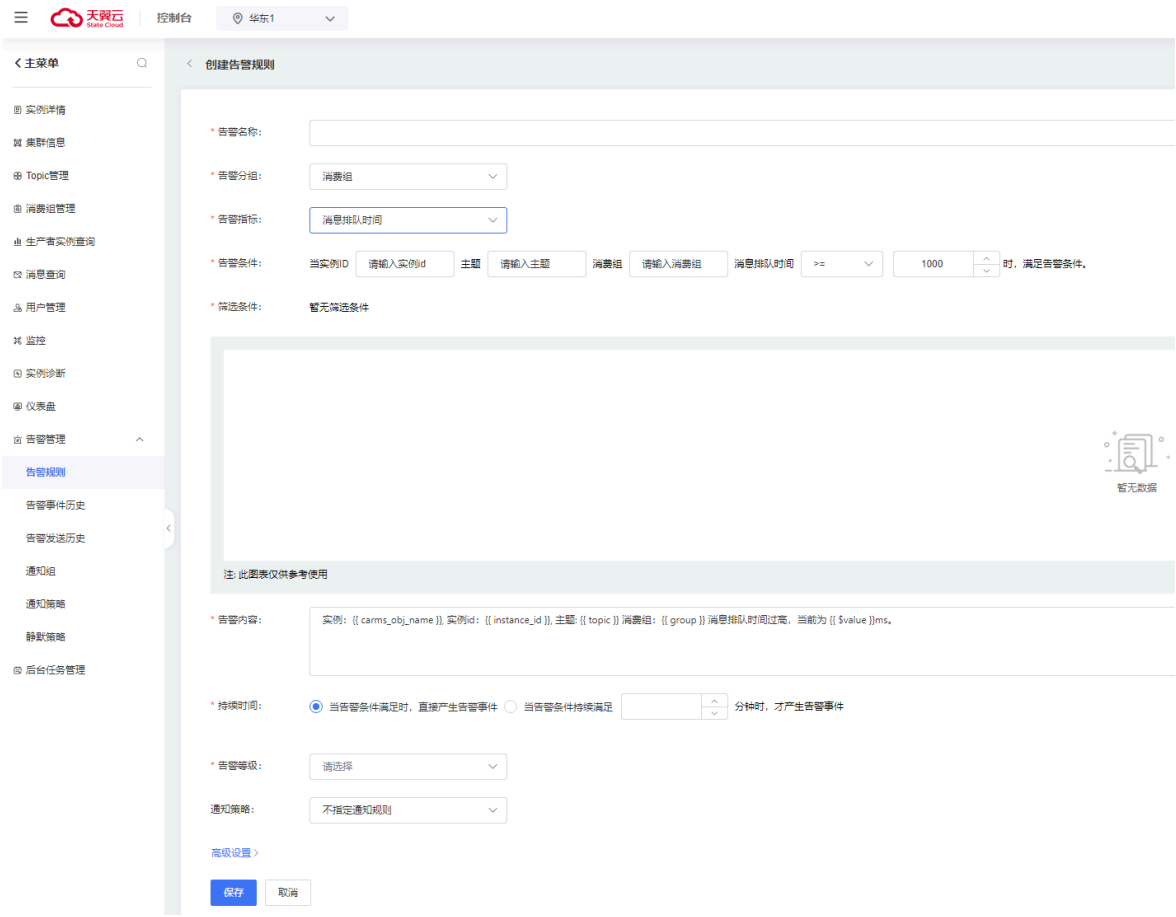
创建通知策略

进入分布式消息服务RocketMQ管理控制台，在实例列表页点击管理进入实例详情页，左侧菜单点击告警管理点击通知策略菜单，右侧点击创建通知策略，输入信息，点击确认



创建告警规则

进入分布式消息服务RocketMQ管理控制台，在实例列表页点击管理进入实例详情页，左侧菜单点击告警管理点击告警规则菜单，右侧点击创建告警规则，选择告警指标，输入告警条件、告警内容等信息，点击保存



云审计服务支持的关键操作

云审计服务支持的RocketMQ操作列表

本页面主要介绍分布式消息服务RocketMQ对接的云审计服务使用和查看方法。

操作场景

本服务现已对接天翼云云审计服务，云审计服务提供对各种云资源操作的记录和查询功能，用于支撑合规审计、安全分析、操作追踪和问题定位等场景，同时提供事件跟踪功能，将操作日志转储至对象存储实现永久保存。

云审计可提供的功能服务具体如下：

- 记录审计日志：支持用户通过管理控制台或API接口发起的操作，以及各服务内部自触发的操作。
- 审计日志查询：支持在管理控制台对7天内操作记录按照事件类型、事件来源、资源类型、筛选类型、操作用户和事件级别等多个维度进行组合查询。

使用限制

- 云审计服务本身免费，包括时间记录以及7天内时间的存储和检索。

- 用户通过云审计能查询到多久前的操作事件：7天。
- 用户操作后多久可以通过云审计查询到数据：5分钟。
- 其它限制请参考使用限制-云审计。

操作步骤

1. 开通云审计服务。

参见开通云审计服务-云审计。

2. 查看云审计事件。

参见查看审计事件-云审计。

3. 在事件列表中，选择事件来源为“容器与中间件”，资源类型选择“分布式消息服务Kafka”，上方时间选择需要筛选的时间段。点击查询即可。

4. 在审计事件右侧点击详情，可以看到更详细的事件信息。

更多云审计相关使用说明和常见问题请参考用户指南、常见问题。

关键操作列表

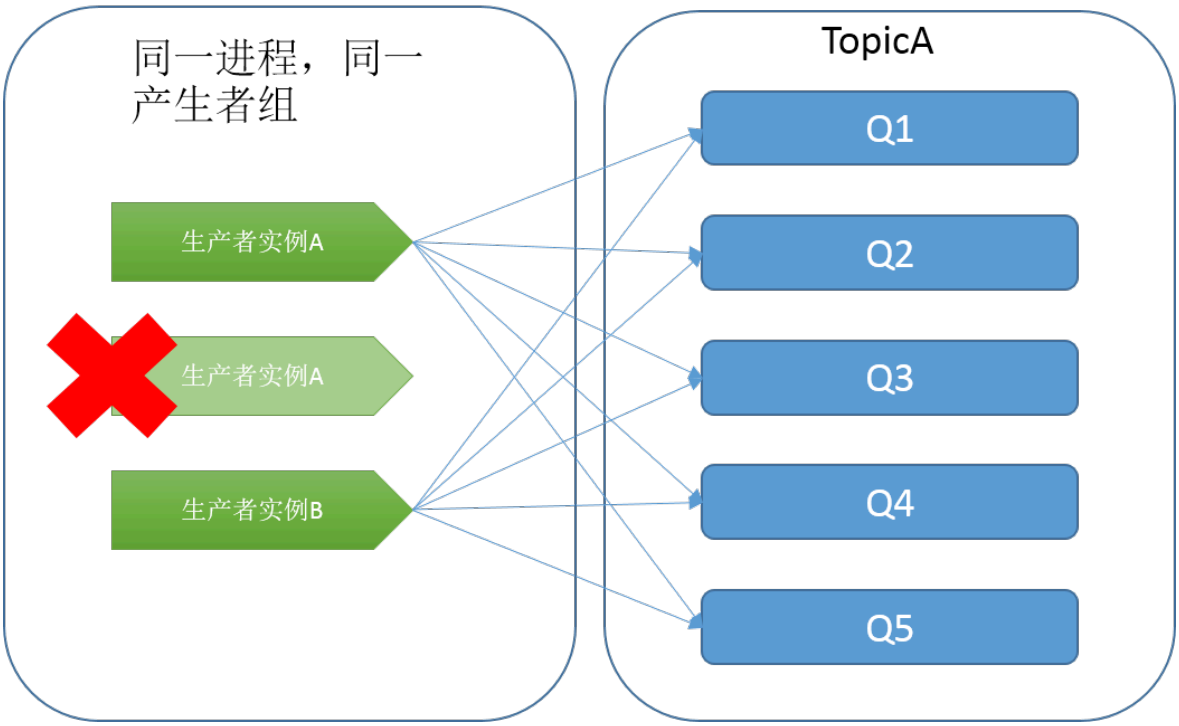
操作事件	读写类型
重启实例节点	写事件
创建消息路由任务	写事件
删除消息路由任务	写事件
暂停消息路由任务	写事件
恢复消息路由任务	写事件
重发死信消息	写事件
更新Topic的读写模式	写事件
创建Acl用户	写事件
删除用户	写事件
更新用户	写事件
用户更新Topic权限	写事件
创建Connector任务	写事件
删除Connector任务	写事件
暂停Connector任务	写事件
恢复Connector任务	写事件
创建预付费实例	写事件
创建后付费实例	写事件
磁盘扩容	写事件
节点扩容	写事件
升规格	写事件

操作事件	读写类型
续订	写事件
退订	写事件

生产者

应用进程、生产组、生产实例的关系

同一进程内，同一生产组不能有相同的实例



实例的创建和销毁

1.使用CTGFactory进行创建Producer实例【强制规范】。

2.Prouder是重量级的实例，每次创建、销毁都会消耗系统资源，建议系统启动的时候创建，系统退出的时候关闭，禁止每次发送消息创建新的实例【强制规范】。

客户端参数建议

- 生产者参数：

常量字段	说明
namesrv	地址必要，例如：192.168.10.10:9876;192.168.10.11:9876
namesrv	用户名必填
namesrv	密码必填
ClusterName	客户端订阅broker的集群名，根据实际情况设置
TenantID	租户ID，根据实际情况设置

常量字段	说明
生产组名称	生产者必填
VipChannelEnabled	true或者false，默认是true，生产和消费端口分开，如果是false，那就共用一个端口
EnalbeCipher	true或者false，默认是false，客户端字段填明文密码，如果设置为true，字段填md5密文

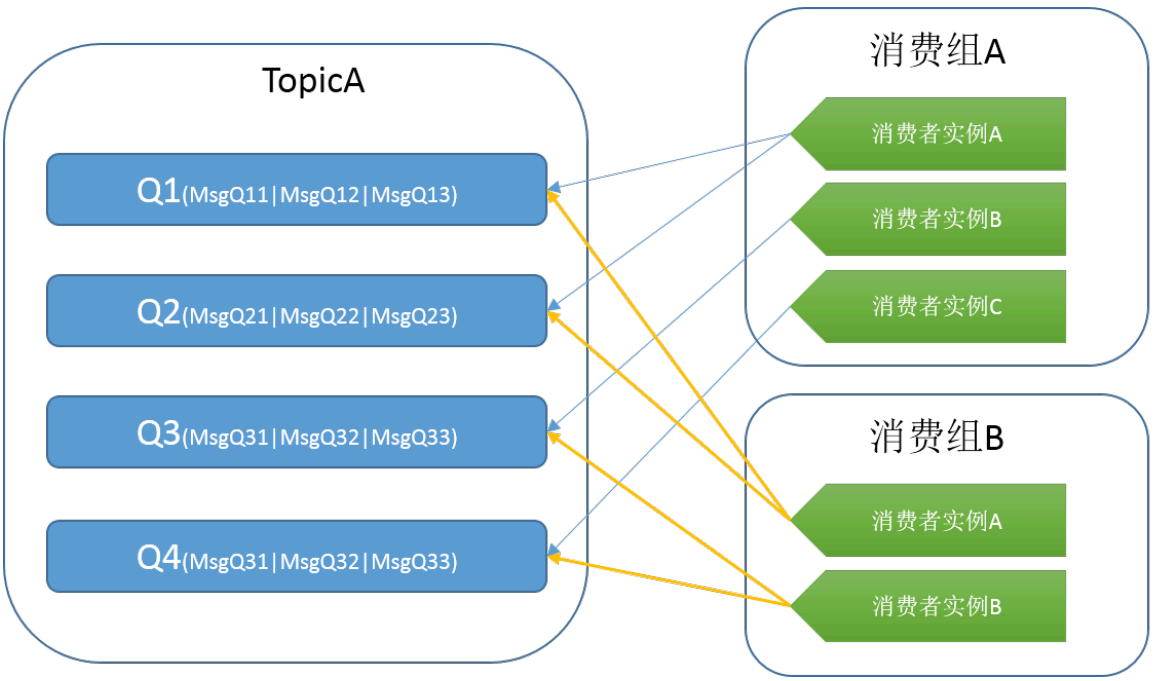
- 消费者参数：

常量字段	说明
从哪个位置开始消费	建议设置为：ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET.name()
消费组名称	消费者必要
最大消费线程数	针对push模式，默认64
最小消费线程数	针对push模式，默认20
消费超时时间	1.默认不设或设置<=0：表示不做超时处理，应用自己处理签收2.设置>0：表示一旦超过此消费超时时间（应用未做签收），客户端将自动签收失败，消息进入重试队列。
namesrv	地址必要，例如：192.168.10.10:9876;192.168.10.11:9876
namesrv	用户名必填
namesrv	密码必填
ClusterName	客户端订阅broker的集群名，默认defaultMQBrokerCluster
TenantID	租户ID，默认defaultMQTenantID

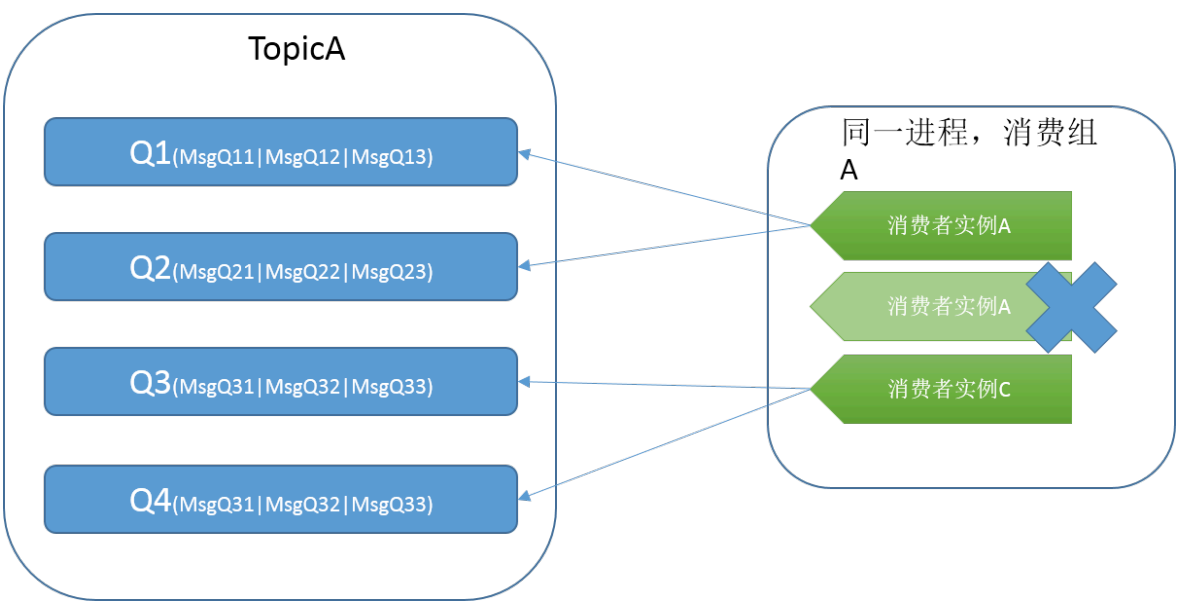
消费者

应用进程、消费组、消费者实例的关系

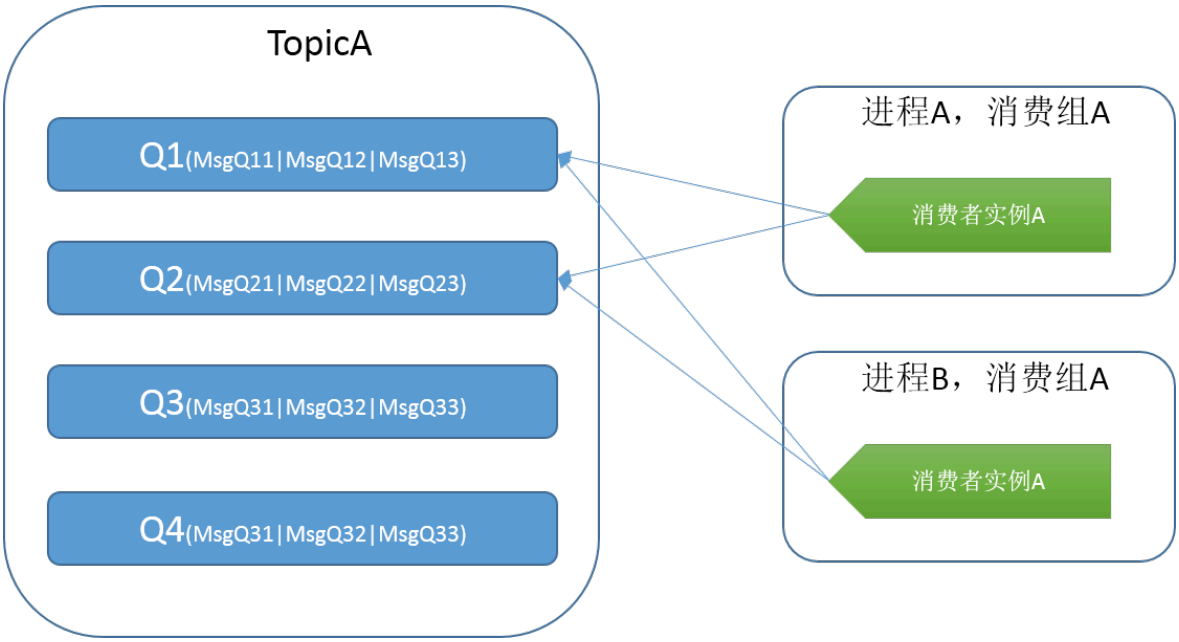
消费组可以有多个消费者实例。



同一进程，同一个消费组不允许有相同的消费者实例。



不同进程，不能创建相同消费者实例，可能产生两个实例分配到相同的队列，部分队列却没有消费的情况。



不同进程，不能创建相同消费者实例，如上图中的实例A，两个实例均连到Q1跟Q2，但Q3与Q4并无消费。

应用在创建消费者实例时，指定消费者实例名，应用需要保证不同的进程间，同一消费组不能有相同的实例名。或者应用在创建消费者实例时，不指定实例名，RocketMQ会创建唯一的实例名(JAVA SDK)，规则是：groupName|ip|pid|线程id|uuid。

同组Consumer订阅关系一致

RocketMQ 里的一个 Consumer Group 代表一个 Consumer 群组。对于大多数分布式应用来说，一个 Consumer Group 下通常会有多个 Consumer 实例。订阅关系一致指的是同一个 Consumer Group 下所有 Consumer 实例的处理逻辑必须完全一致，一旦订阅关系不一致，消息消费的逻辑就会混乱，甚至导致消息丢失。

由于 RocketMQ 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 Consumer group 下所有的实例需在以下两方面均保持一致：

1.订阅的 Topic 必须一致。

2.订阅的 Topic 中的 Tag 必须一致。

实例的创建和销毁

1.使用CTGFactory进行创建实例【强制规范】。

2.Consumer是重量级的实例，每次创建、销毁都会消耗系统资源，建议系统 启动的时候创建，系统退出的时候关闭，禁止每次消费消息都创建新的实例【强制规范】。

多消费组消费

一个Topic，可以使用多个消费组消费消息，每个消费组将在服务端独立保存进度。

集群消费意味着多个消费者均衡消费Topic的消息，由于经常不同的程序由多个开发者进行研发和调试，如果使用同一个消费组，在调试过程中存在被其他程序消费者消费的可能，因此应该尽量避免多种类型的应用程序使用同一个消费组。

消费位置设置

消费位置重置可通过控制台进行按时间重置，客户端必须离线。

堆积量

不建议高堆积量的消费，为了预防出现高堆积的情况，建议：

1.边生产边消费，如果消费速度跟不上，增加消费者。

2.消费者一直在线，不要等生产了一段时间再开启消费者，这样会造成消费的堆积。

消费并行度

1. 同一个 ConsumerGroup 下，可通过增加 Consumer 实例数量来提高消费的并行度（注意超过主题队列数的 部分Consumer 实例因为分配不到队列而无效），从而提升消费吞吐。
2. 增加单个消费节点的线程数，通过修改Consumer 的参数 consumeThreadMin、consumeThreadMax，增加并发线程个数实现更高的并发度。
3. 批量方式消费，如果业务流程支持批量方式消费处理消息，可设置 consumer的 consumeMessageBatchMaxSize 返回消息个数参数，该参数默认为 1，即一次只消费一条消息。通过调大该参数的值，则可以很大程度上提高消费的吞吐量。

topic、queue的规划

在RocketMQ中，队列数直接影响到消费者实例数的上限，同一消费组消费者实例数的上限=队列数，需要集群消费的情况，需考虑队列数的设置。

在RocketMQ中，队列能分布到不同的Broker上，是RocketMQ分布式的基础。Queue分布在Broker中，则能使用Broker的资源，包括存储、IO等，一般情况下，分布在某个Broker上的Queue比例越大，则占用此Broker的资源越多，Topic中的Queue分布到的Broker数量越多，则性能越好、存储越大。若Broker的所在机器性能不同，可以通过调整Queue数量，达到资源调优的目的，在应用设计时，需要充分利用上述特性。

在Push消费模式中，API会默认为每个队列预拉取消息1000条，若队列数过大、或者单条消息包体过大，则需要考虑设置减少预拉数量，防止预拉消息过大导致内存溢出。

Java客户端Pull和Push的选择：Java客户端必须使用Push Consumer

- 使用Pull可以实现的所有场景，均可使用Push实现，并且更简单。
- Push其实是长轮询的Pull（依然是由客户端发起），在客户端通过配置参数是可以实现流控的，并不会出现服务端的流量压垮客户端的情况。

- Push封装了拉取消息，分发给消费线程的线程模型，非流控的情况下，由后台线程主动拉取消息，并缓存在本地，消费线程池有空闲线程时，分发给消费线程，在有堆积量的情况下，可以保证消费线程一直工作，性能更高（备注：Pull只提供了拉取消息的功能，并且何时去拉取，拉取时机，这些都需要应用去控制；分发给消费线程的逻辑需要应用封装，除了增加应用工作量外，还可能有不稳定、性能问题等）。
- Push经过多个大型项目的长时间的使用(比如物联网，使用能台，多个试点省份)，更成熟稳定。
- Push会自动订阅重试队列，不需要再次调用拉取重试队列的API来取得重试队列的消息（备注：Pull需要另外调用API拉取重试队列的消息）。
- Pull是一种遗留的消费模式（兼容早期的API），新开发的应用，或者未上线的应用，都要求使用Push消费模式。

有序消费和无序消费的选择

在业务场景允许的情况下，优先选择无序消息，或者在业务能变通的情况下，将有序消息转化为无序消息。

无序消息的优点：

- 生产者可以使用多进程、多线程往同一个Topic发送消息，性能更好。
- 消费者可以使用多进程、多线程同时消费，性能较好。
- 可以充分使用集群的Failover特点，无须依赖自动主备切换（切换过程服务会中断），包括：
  - 当集群中某一Broker节点故障时，不影响业务消息生产，消息将failover发送到其它节点；
  - 当集群中某一Broker节点故障时，不影响其它节点数据消费，故障恢复后即可消费。
- 能动态地扩容。

有序消息的缺点：

- 对于有序消息，当节点故障时，Queue数不会变化，生产与消费都会出现异常，直到故障节点恢复。
- 对于有序消息，需要将所有消息消费完，并且停止客户端，才能扩容。

消费幂等

RocketMQ无法避免消息重复，原因主要有以下几点：

- 签收的偏移量是定时（每5秒/次）同步到服务端的。
- 为保证消息不丢失，SDK每次提交的总是队列未签收的最小偏移量（比如无序消费，offset为1，2，3，4，5的消息，1，3，4，5消费并已签收，2未签收，签收时最后提交的偏移量将会是2，如果此时，客户端重启，会从2这个位置开始消费）。
- 有网络交互就不能确保每一次的交互数据都是送达的，为保证数据不丢失就要进行重试，有重试就存在重复的可能。

如果业务对消费重复非常敏感，务必要注意，建议可以采用以下两种方式处理：

- 业务层面做去重处理，可以根据msgId；如果key字段为业务唯一字段，也可采用key去重。
- 业务逻辑实现消费幂等，即多次处理同一消息，对业务的影响是幂等的。

最佳实践

业务消息设计：Topic与Tag

Topic与Tag释义

1）Topic：消息主题，通过 Topic 对不同的业务消息进行分类。

2）Tag：消息标签，用来进一步区分某个Topic下的消息分类，消息队列 RocketMQ允许消费者按照Tag对消息进行过滤，确保消费者最终只消费到他关注的消息类型。

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 可以说是二级分类。

适用场景

什么时候该用topic，什么时候该用tag，可以参考下面的一些考虑进行权衡

考虑消息类型：如普通消息、顺序消息，事务消息、定时（延时）消息，不同消息类型是无法通过tag区分的，这种情况就需要我们创建不同的topic。

业务关联性：如果是不同业务之间没有直关联的消息，建议按照Topic进行区分；而同一个业务只是子类型不一样的消息可以用Tag进行区分。

消息优先级：不同的业务场景可能会导致消费端对于消息的优先级需求不同，有的紧急，有的相对来说对于延时的接收程度更大，不同优先级的消息用不同的Topic进行区分。

消息量级：如果量小但延时要求高的消息，跟超大量级（如万亿）的消息使用同一个Topic，则有可能排队时间过长导致延时无法接受，所以不同量级的消息不要使用不同的tag，需要用不同的Topic。

总结起来就是，在消息分类实践中，有创建多个Topic，以及在同一个Topic下创建多个Tag两种常见做法。一般来说，不同的Topic之间的消息不产生直接业务上的关联，而同一个topic下相互之间产生联系的消息可以选择用tag来区分，一般是相同业务下的不同板块不同类型。

同组Consumer订阅关系一致

RocketMQ里的一个Consumer Group代表一个Consumer群组。对于大多数分布式应用来说，一个Consumer Group下通常会有多个Consumer实例。订阅关系一致指的是同一个Consumer Group下所有Consumer实例的处理逻辑必须完全一致，一旦订阅关系不一致，消息消费的逻辑就会混乱，甚至导致消息丢失。

背景信息

RocketMQ 中一个消费者代表一个Consumer实例群组。在大多数场景中，一个消费者组下面包含多个Consumer实例。

由于分布式消息服务RocketMQ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实例订阅关系的一致性大概包括下面几个方面：

同一个消费组订阅的Topic必须一致，例如：在同一个消费组下，ConsumerA订阅Topic1和Topic2，ConsumerB也必须订阅Topic1和Topic2，只订阅Topic1、只订阅Topic2或订阅Topic2和Topic3都是不允许的。

同一个消费者订阅的同一个Topic的场景下Tag必须一致，包括Tag的数量和T顺序，例如：ConsumerA订阅Topic1的Tag配置为Tag1||Tag2，ConsumerB订阅Topic1的Tag也必须是Tag1||Tag2，只订阅Tag1、只订阅Tag2或者订阅Tag2||Tag1都是不允许的。

正确的订阅关系如下，多个不同的topic可以被多个消费组订阅，但是同一个消费组下的多个Consumer实例订阅Topic和Tag都必须一致。

代码示例

- 订阅一个Topic、一个Tag

同一个消费组下面的全部消费者实例均订阅一个topic，且均配置同一个tag这种是符合订阅关系一致性原则的。

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic,"Tag1");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

- 订阅一个topic多个tag

每个消费者订阅消息的代码必须一致

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic,"Tag1||Tag2");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

- 订阅多个topic且订阅多个tag

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic1,"Tag1");
consumer.subscribe(topic2,"Tag1 | Tag2");
consumer.subscribe(topic3,"*");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

常见的订阅关系不一致情况

如果Rocketmq实例消费到的消息不符合预期，可以检查一下消费者逻辑是否存在订阅关系不一致的情况

下面列举几种常见的错误示例

同一个消费组下订阅的topic不一致

消费者实例1的代码：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic1,"*");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

消费者实例2的代码：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic2,"*");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
```

```
//      do something
}
```

消费者实例3的代码：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic3,"*");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

同一个消费组的消费实例订阅的topic相同但订阅的tag不一致

消费者实例1的代码：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic,"Tag1");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

消费者实例2的代码：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic,"Tag2");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

消费者实例3的代码：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic,"Tag2");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

同一个消费组下全部消费者实例订阅的topic以及tag都一致但订阅tag的顺序不一致

消费者实例1的代码

：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic,"Tag1||Tag2");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

消费者实例2的代码：

```
consumer.setConsumerGroup("group1");
```

```
consumer.subscribe(topic,"Tag2||Tag1");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

消费者实例3的代码：

```
consumer.setConsumerGroup("group1");
consumer.subscribe(topic,"Tag2||Tag1");
consumer.registerMessageListener((MessageListenerConcurrently)(msgs, context)->{
//      do something
}
```

概述

如何获取RocketMQ实例连接信息请参阅收集连接信息。

开发指南详细介绍Java、Go和Python客户端访问分布式消息服务RocketMQ的示例代码，具体如表1所示。

表1 示例代码

客户端语言	示例代码
Java	收发普通消息收发顺序消息收发事务消息发送定时/延时消息
Go（TCP协议）	收发普通消息 收发顺序消息收发事务消息 发送定时/延时消息
Python（TCP协议）	收发普通消息 收发顺序消息收发事务消息 发送定时/延时消息

收集连接信息

- 实例连接地址和端口

  实例创建后，从RocketMQ实例控制台的“实例详情”页面获取，在客户端配置时，可将地址都配上。

- Topic名称

  从RocketMQ实例控制台的“主题管理”页签中获取Topic名称。

- 订阅组名称

  从RocketMQ实例控制台的“订阅组管理”页签中获取消费组名称。

- 用户名和用户密钥

  从RocketMQ实例控制台的“角色控制”页面获取用户名，在用户详情页获取用户密钥。

Java

收发普通消息

本章节介绍普通消息的收发方法和示例代码。其中，普通消息发送方式分为同步发送、异步发送、单向发送。

- 同步发送：同步发送是指消息发送方发出一条消息后，会在收到服务端同步响应之后才发下一条消息的通讯方式。
- 异步发送：异步发送是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式。
- 单向发送：发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发。
- 收发消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

准备环境

开源的Java客户端支持连接分布式消息服务RocketMQ版，推荐使用的客户端版本为4.9.7。

通过以下任意一种方式引入依赖：

1. 使用Maven方式引入依赖。

```
<dependencies>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.9.7</version>
  </dependency>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-acl</artifactId>
    <version>4.9.7</version>
  </dependency>
</dependencies>
```

2. 点击下载依赖JAR包：rocketmq-all-4.9.7-bin-release.zip

同步发送

同步发送是最常用的方式，是指消息发送方发出一条消息后，会在收到服务端同步响应之后才发下一条消息的通讯方式，可靠的同步传输被广泛应用于各种场景，如重要的通知消息、短消息通知等。

参考如下示例代码

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class ProducerNormalExample {
  private static RPCHook getAclRPCHook() {
```

```
        return new AclClientRPCHook(new SessionCredentials(
            "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
            "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
        ));
    }
    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook());
       // 填入控制台获取NAMESRV接入点地址
        producer.setNamesrvAddr("XXX:xxx");
        //producer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
        producer.start();
        for (int i = 0; i < 128; i++)
            try {
                {
                    Message msg = new Message("YOUR TOPIC",
                        "TagA",
                        "Hello RocketMQ".getBytes(RemotingHelper.DEFAULT_CHARSET));
                    SendResult sendResult = producer.send(msg);
                    System.out.println(sendResult);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        producer.shutdown();
    }
}
```

异步发送

消息发送方在发送了一条消息后，不需要等待服务端响应即可发送第二条消息，发送方通过回调接口接收服务端响应，并处理响应结果。异步发送一般用于链路耗时较长，对响应时间较为敏感的业务场景。例如，视频上传后通知启动转码服务，转码完成后通知推送转码结果等。

参考如下示例代码

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendCallback;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class AsyncProducerNormalExample {
    private static RPCHook getAclRPCHook() {
```

```
        return new AclClientRPCHook(new SessionCredentials("accessKey", // 分布式消息服务
RocketMQ控制台用户管理菜单中创建的用户ID
            "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
        ));
    }
    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook());
        // 填入控制台获取NAMESRV接入点地址
        producer.setNamesrvAddr("XXX:xxx");
        //producer.setUseTLS(true); //如果需要开启SSL，请增加此行代码
        producer.start();
        int messageCount = 10;
        final CountDownLatch countDownLatch = new CountDownLatch(messageCount);
        for (int i = 0; i < messageCount; i++) {
            try {
                Message msg = new Message("YOUR TOPIC",
                    "TagA", // 设置消息的TAG，若无可设置为空
                    "Hello RocketMQ".getBytes(RemotingHelper.DEFAULT_CHARSET));
                producer.send(msg, new SendCallback() {
                    @Override
                    public void onSuccess(SendResult sendResult) {
                        countDownLatch.countDown();
                        System.out.println("send message success. msgId= " + sendResult.getMsgId());
                    }
                    @Override
                    public void onException(Throwable e) {
                        countDownLatch.countDown();
                        // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补
偿处理。
                        System.out.println("send message failed.");
                    }
                });
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        countDownLatch.await(5, TimeUnit.SECONDS);
        producer.shutdown();
    }
}
```

单向发送

发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发，即只发送请求不等待应答。此方式发送消息的过程耗时非常短，一般在微秒级别。适用于某些耗时非常短，但对可靠性要求并不高的场景，例如日志收集。

参考如下示例代码

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class OnewayProducerNormalExample {
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials(
                "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
                "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
        ));
    }
    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook());
        // 填入控制台获取NAMESRV接入点地址
        producer.setNamesrvAddr("XXX:xxx");
        //producer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
        producer.start();
        for (int i = 0; i < 10; i++) {
            try {
                Message msg = new Message("TopicTest",
                        "TagA", // 设置消息的TAG，若无可设置为空
                        "Hello RocketMQ".getBytes(RemotingHelper.DEFAULT_CHARSET));
                producer.sendOneway(msg);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        producer.shutdown();
    }
}
```

订阅普通消息

参考如下示例代码

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.remoting.RPCHook;
public class ConsumerNormalExample {
    private static RPCHook getAclRPCHook() {
```

```
        return new AclClientRPCHook(new SessionCredentials(
            "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
            "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
        ));
    }
    public static void main(String[] args) throws Exception {
/*
* 创建Consumer，如果想开启消息轨迹，可以按照如下方式创建：
* DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
 AllocateMessageQueueAveragely(), true, null);
 */
        DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
            AllocateMessageQueueAveragely());
        // 填入控制台NAMESRV接入点地址
        consumer.setNamesrvAddr("XXX:xxx");
        // consumer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
        /*
         * 如果想要消费指定TAG的消息，可以按照如下方式订阅：*为订阅所有的TAG
         * pushConsumer.subscribe(TOPIC_NAME, "Tag1");
         */
        consumer.subscribe("TopicTest", "*");
        consumer.registerMessageListener((MessageListenerConcurrently) (msgs, context) -> {
            System.out.printf("Receive New Messages: %s %n", msgs);
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        consumer.start();
        System.out.println("Consumer Started.");
    }
}
```

收发顺序消息

顺序消息是分布式消息服务RocketMQ版提供的一种严格按照顺序来发布和消费的消息类型。

顺序消息分为全局顺序消息和分区顺序消息：

- 全局顺序消息：对于指定的一个Topic，将队列数量设置为1，这个队列内所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和订阅。
- 分区顺序消息：对于指定的一个Topic，同一个队列内的消息按照严格的FIFO顺序进行发布和订阅。生产者指定分区选择算法，保证需要按顺序消费的消息被分配到同一个队列。

全局顺序消息和分区顺序消息的区别仅为队列数量不同，代码没有区别。

收发消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

准备环境

开源的Java客户端支持连接分布式消息服务RocketMQ版，推荐使用的客户端版本为4.9.7。

通过以下任意一种方式引入依赖：

1. 使用Maven方式引入依赖。

```
<dependencies>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.9.7</version>
  </dependency>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-acl</artifactId>
    <version>4.9.7</version>
  </dependency>
</dependencies>
```

2. 点击下载依赖JAR包：rocketmq-all-4.9.7-bin-release.zip

发送顺序消息

参考如下示例代码

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class ProducerFifoExample {
  private static RPCHook getAclRPCHook() {
    return new AclClientRPCHook(new SessionCredentials(
      "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
      "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
    ));
  }
  public static void main(String[] args) throws Exception {
    /*
     * 创建Producer，如果想开启消息轨迹，可以按照如下方式创建：
     * DefaultMQProducer producer
 = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook(), true, null);
     */
    DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook());
    // 填入控制台NAMESRV接入点地址
    producer.setNamesrvAddr("XXX:xxx");
    //producer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
    producer.start();
    for (int i = 0; i < 128; i++) {
      try {
```

```
            int orderId = i % 10;
            Message msg = new Message("TopicTest",
               "YOUR MESSAGE TAG",
               "Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
            SendResult sendResult = producer.send(msg, (mqs, msgl, arg) -> {
               // 选择适合自己的分区选择算法，保证同一个参数得到的结果相同。
               Integer id = (Integer) arg;
               int index = id % mqs.size();
               return mqs.get(index);
            }, orderId);
            System.out.printf("%s%n", sendResult);
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
      producer.shutdown();
   }
}
```

注意

上述代码中，相同id的消息需要保证顺序，不同id的消息不需要保证顺序，所以在分区选择算法中以“id/队列个数的余数”作为消息发送的队列。

订阅顺序消息

参考如下示例代码

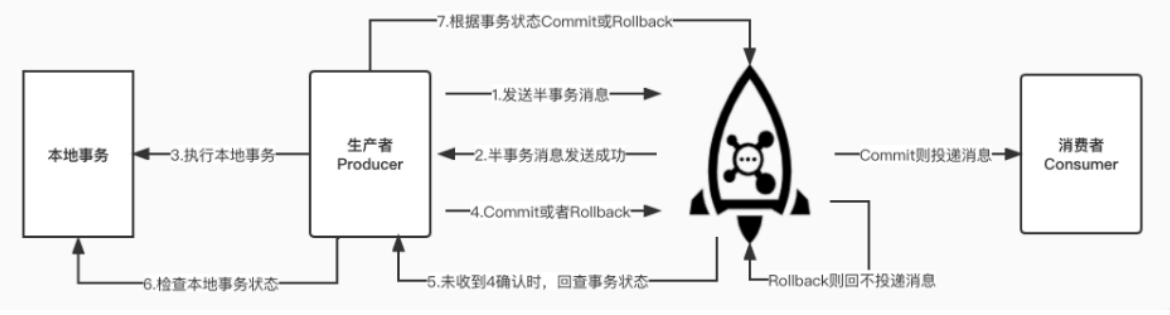
```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeOrderlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerOrderly;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.remoting.RPCHook;

public class ConsumerFifoExample {
   private static RPCHook getAclRPCHook() {
      return new AclClientRPCHook(new SessionCredentials(
            "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
            "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
      ));
   }

   public static void main(String[] args) throws Exception {
  /*
* 创建Consumer，如果想开启消息轨迹，可以按照如下方式创建：
* DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
```

```
 AllocateMessageQueueAveragely(), true, null);
 */
     DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
          AllocateMessageQueueAveragely());
     // 填入控制台NAMESRV接入点地址
     consumer.setNamesrvAddr("XXX:xxx");
     // consumer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
     /*
      * 如果想要消费指定TAG的消息，可以按照如下方式订阅：* 为订阅所有的TAG
      * pushConsumer.subscribe(TOPIC_NAME, "Tag1");
      */
     consumer.subscribe("TopicTest", "*");
     consumer.registerMessageListener((MessageListenerOrderly) (msgs, context) -> {
       System.out.printf("Receive New Messages: %s %n", msgs);
       return ConsumeOrderlyStatus.SUCCESS;
     });
     consumer.start();
     System.out.println("Consumer Started.");
   }
}
```

收发事务消息

分布式消息服务RocketMQ版的事务消息支持在业务逻辑与发送消息之间提供事务保证，通过两阶段的方式提供对事务消息的支持，事务消息交互流程如图1所示。

图1 事务消息交互流程



事务消息生产者首先发送半消息，然后执行本地事务。如果执行成功，则发送事务提交，否则发送事务回滚。服务端在一段时间后如果一直收不到提交或回滚，则发起回查，生产者在收到回查后重新发送事务提交或回滚。消息只有在提交之后才投递给消费者，消费者对回滚的消息不可见。

收发事务消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

准备环境

开源的Java客户端支持连接分布式消息服务RocketMQ版，推荐使用的客户端版本为4.9.7。

通过以下任意一种方式引入依赖：

1. 使用Maven方式引入依赖

```
<dependencies>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.9.7</version>
  </dependency>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-acl</artifactId>
    <version>4.9.7</version>
  </dependency>
</dependencies>
```

2. 点击下载依赖JAR包：rocketmq-all-4.9.7-bin-release.zip

发送事务消息

参考如下示例代码

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.producer.LocalTransactionState;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.client.producer.TransactionListener;
import org.apache.rocketmq.client.producer.TransactionMQProducer;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.common.message.MessageExt;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class ProducerTransactionExample {
  private static RPCHook getAclRPCHook() {
    return new AclClientRPCHook(new SessionCredentials(
      "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
      "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
    ));
  }
  public static void main(String[] args) throws Exception {
    // 执行本地事务和事务回查的接口
    TransactionListener transactionListener = new TransactionListener() {
      @Override
      public LocalTransactionState executeLocalTransaction(Message message, Object o) {
        System.out.println("开始执行本地事务： " + message);
        return LocalTransactionState.COMMIT_MESSAGE;
      }
      @Override
      public LocalTransactionState checkLocalTransaction(MessageExt messageExt) {
        System.out.println("收到事务消息的回查请求, MsgId: " + messageExt.getMsgId());
```

```
            return LocalTransactionState.COMMIT_MESSAGE;
        }
    };
    /*
     * 创建Producer，如果想开启消息轨迹，可以按照如下方式创建：
     * TransactionMQProducer producer = new TransactionMQProducer(null,
"YOUR GROUP ID", getAclRPCHook(), true, null);
     */
    TransactionMQProducer producer = new Transact
ionMQProducer("YOUR GROUP ID", getAclRPCHook());
    // 填入控制台NAMESRV接入点地址
    producer.setNamesrvAddr("XXX:xxx");
    //producer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
    producer.setTransactionListener(transactionListener);
    producer.start();
    Message msg = new Message("YOUR TOPIC",
      "TagA",
      "Hello RocketMQ".getBytes(RemotingHelper.DEFAULT_CHARSET));
    SendResult sendResult = producer.sendMessageInTransaction(msg, null);
    System.out.printf("%s%n", sendResult);
  }
}
```

事务消息生产者需要实现执行本地事务和事务回查的接口，其中executeLocalTransaction方法在发送完半事务消息后被调用，checkLocalTransaction方法在收到事务回查时调用，两者被调用时可返回如下三个事务状态。

- LocalTransactionState.COMMIT_MESSAGE：提交事务，允许事务消息投递到消费者。

- LocalTransactionState.ROLLBACK_MESSAGE：回滚事务，消息将被丢弃不允许消费。

- LocalTransactionState.UNKNOW：无法判断状态，服务端会向生产者再次回查该消息的状态。

订阅事务消息

参考如下示例代码

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.remoting.RPCHook;
public class ConsumerTransactionExample {
  private static RPCHook getAclRPCHook() {
    return new AclClientRPCHook(new SessionCredentials(
      "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
```

```
            "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
        ));
    }
    public static void main(String[] args) throws Exception {
  /*
* 创建Consumer，如果想开启消息轨迹，可以按照如下方式创建：
* DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
  AllocateMessageQueueAveragely(), true, null);
  */
        DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
          AllocateMessageQueueAveragely());
        // 填入控制台NAMESRV接入点地址
        consumer.setNamesrvAddr("XXX:xxx");
        // consumer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
        /*
         * 如果想要消费指定TAG的消息，可以按照如下方式订阅：* 为订阅所有的TAG
         * pushConsumer.subscribe(TOPIC_NAME, "Tag1");
         */
        consumer.subscribe("TopicTest", "*");
        consumer.registerMessageListener((MessageListenerConcurrently) (msgs, context) -> {
            System.out.printf("Receive New Messages: %s %n", msgs);
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        consumer.start();
        System.out.println("Consumer Started.");
    }
}
```

收发定时/延时消息

分布式消息服务RocketMQ版支持任意时间的定时消息，最大推迟时间可达到40天。

定时消息即生产者生产消息到分布式消息服务RocketMQ版后，消息不会立即被消费，而是延迟到设定的时间点后才会发送给消费者进行消费。

发送定时消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

适用场景

定时消息适用于以下场景：

- 消息对应的业务逻辑有时间窗口要求，如电商交易中超时未支付关闭订单的场景。在订单创建时发送一条定时消息，5分钟以后投递给消费者，消费者收到此消息后需要判断对应订单是否完成支付，如果未完成支付，则关闭订单。如果已完成，则忽略。
- 通过消息触发定时任务的场景，如在某些固定时间点向用户发送提醒消息。

注意

- 定时消息的最大延迟时间为40天，延迟超过40天的消息将会发送失败。

- 定时消息的定时时间如果被设置成当前时间戳之前的某个时刻，消息将立刻投递给消费者。
- 定时消息的精度有1s~2s的延迟误差
- 无法确保定时消息仅投递一次，定时消息可能会重复投递。
- 定时消息的定时时间是服务端开始向消费端投递的时间。如果消费者当前有消息堆积，那么定时消息会排在堆积消息后面，将不能严格按照配置的时间进行投递。
- 由于客户端和服务端可能存在时间差，消息的实际投递时间与客户端设置的投递时间之间可能存在偏差，以服务端时间为准。
- 设置定时消息的投递时间后，依然受消息老化时间限制，默认消息过期时间为7天。例如，设置定时消息5天后才能被消费，如果第5天后一直没被消费，那么这条消息将在第12天被删除。
- 定时消息将占用普通消息约3倍的存储空间，大量使用定时消息时需要注意存储空间占用。

准备环境

开源的Java客户端支持连接分布式消息服务RocketMQ版，推荐使用的客户端版本为4.9.7。

通过以下任意一种方式引入依赖：

1. 使用Maven方式引入依赖。

```
<dependencies>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.9.7</version>
  </dependency>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-acl</artifactId>
    <version>4.9.7</version>
  </dependency>
</dependencies>
```

2. 点击下载依赖JAR包：rocketmq-all-4.9.7-bin-release.zip

发送定时/延时消息

发送定时/延时消息的示例代码如下：

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;

public class ProducerTimerDelayExample {
  private static RPCHook getAclRPCHook() {
    return new AclClientRPCHook(new SessionCredentials(
        "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
```

```
            "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
        ));
    }

    public static void main(String[] args) throws Exception {
        /*
         * 创建Producer，如果想开启消息轨迹，可以按照如下方式创建：
         * DefaultMQProducer producer
 = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook(), true, null);
         */
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook());
        // 填入控制台NAMESRV接入点地址
        producer.setNamesrvAddr("XXX:xxx");
        //producer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
        producer.start();
        for (int i = 0; i < 128; i++) {
            try {
                Message msg = new Message("TopicTest",
                        "YOUR MESSAGE TAG",
                        "Hello RocketMQ".getBytes(RemotingHelper.DEFAULT_CHARSET));
                /*
                 * 发送延时消息，需要设置延时时间，单位毫秒（ms），消息将在指定延时时间后投递，
例如消息将在3秒后投递。
                 */
                long delayTime = System.currentTimeMillis() + 3000;
                msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(delayTime));
                /*
                 * 若需要发送定时消息，则需要设置定时时间，消息将在指定时间进行投递，例如消息将
在2022-10-10 10:10:00投递。
                 * 定时时间格式为：yyyy-MM-dd HH:mm:ss，若设置的时间戳在当前时间之前，则消息将被
立即投递给Consumer。
                 * long timeStamp = new SimpleDateFormat("yyyy-MM-
dd HH:mm:ss").parse("2022-10-10 10:10:00").getTime();
                 * msg.putUserProperty("__STARTDELIVERTIME",String.valueOf(timeStamp));
                 */
                SendResult sendResult = producer.send(msg);
                System.out.printf("%s%n", sendResult);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        producer.shutdown();
    }
}
```

消费定时/延时消息

消费定时/延时消息的示例代码如下

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.remoting.RPCHook;
public class ConsumerDelayExample {
  private static RPCHook getAclRPCHook() {
    return new AclClientRPCHook(new SessionCredentials(
      "accessKey", // 分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
      "accessSecret" // 分布式消息服务RocketMQ控制台用户管理菜单中创建的密钥
    ));
  }
  public static void main(String[] args) throws Exception {
   /*
  * 创建Consumer，如果想开启消息轨迹，可以按照如下方式创建：
  * DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
    AllocateMessageQueueAveragely(), true, null);
    */
     DefaultMQPushConsumer consumer = new DefaultM
QPushConsumer("YOUR GROUP ID", getAclRPCHook(), new
       AllocateMessageQueueAveragely());
    // 填入控制台NAMESRV接入点地址
    consumer.setNamesrvAddr("XXX:xxx");
    // consumer.setUseTLS(true); // 如果需要开启SSL，请增加此行代码
    /*
     * 如果想要消费指定TAG的消息，可以按照如下方式订阅：* 为订阅所有的TAG
     * pushConsumer.subscribe(TOPIC_NAME, "Tag1");
     */
    consumer.subscribe("TopicTest", "*");
    consumer.registerMessageListener((MessageListenerConcurrently) (msgs, context) -> {
      System.out.printf("Receive New Messages: %s %n", msgs);
      return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    });
    consumer.start();
    System.out.println("Consumer Started.");
  }
}
```

消费限流

在分布式消息服务RocketMQ中，消费者消费消息时，可能会出现消费过快导致下游业务来不及处理的情况，进而影响系统的稳定性。本章节介绍在消费端进行限流的示例代码，以保障系统的稳定。

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

import com.google.common.util.concurrent.RateLimiter;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.consumer.ConsumeFromWhere;
import org.apache.rocketmq.common.message.MessageExt;

public class ConsumerLimitExample {

  public static void main(String[] args) throws InterruptedException, MQClientException {
    DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("GroupTest");
    consumer.subscribe("TopicTest", "*");
    consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
    RateLimiter rateLimiter = RateLimiter.create(200);
    consumer.registerMessageListener(new MessageListenerConcurrently() {

      @Override
      public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeC
oncurrentlyContext context) {
        if (!rateLimiter.tryAcquire(msgs.size(),3, TimeUnit.SECONDS)) {
          return ConsumeConcurrentlyStatus.RECONSUME_LATER;
        }
        System.out.printf("%s Receive New Messages: %s %n", Thread.currentT
hread().getName(), msgs);
        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
      }
    });
    consumer.start();
    System.out.printf("Consumer Started.%n");
  }
}
```

Python

收发普通消息

本章节介绍普通消息的收发方法和示例代码。其中，普通消息发送方式分为同步发送、异步发送、单向发送。

- 同步发送：同步发送是指消息发送方发出一条消息后，会在收到服务端同步响应之后才发下一条消息的通讯方式。
- 异步发送：异步发送是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式。
- 单向发送：发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发。
- 收发消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

准备环境

1. 在命令行输入python，检查是否已安装Python。得到如下回显，说明Python已安装。

```
PS C:\> python
Python 3.9.9 (tags/v3.9.9:ccb0e6a, Nov 15 2021, 18:08:50) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

如果未安装Python，请使用以下命令安装：

```
pip install rocketmq-client-python
```

2. 安装librocketmq库和rocketmq-client-python。

说明

建议下载rocketmq-client-cpp-2.2.0，获取librocketmq库。

3. 将librocketmq.so添加到系统动态库搜索路径。

a. 查找librocketmq.so的路径。

```
find / -name librocketmq.so
```

b. 将librocketmq.so添加到系统动态库搜索路径。

```
ln -s /查找到的librocketmq.so路径/librocketmq.so /usr/lib
sudo ldconfig
```

以下示例代码中的参数说明如下，请参考收集连接信息获取参数值。

- GROUP：表示消费组名称。
- ENDPOINT：表示实例连接地址和端口。
- TOPIC：表示Topic名称。

发送消息

参考如下示例代码。

```
from rocketmq.client import Producer, Message
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
producer_group = "${GROUP}" # 生产者组group
# 创建并启动生产者实例
producer = Producer(producer_group)
```

```
producer.set_name_server_address(endpoint)
producer.set_session_credentials(access_key, access_secret, "")
producer.start()
msg = Message(topic)
msg.set_body("Hello RocketMQ")
msg.set_keys("") # 消息key
msg.set_tags("") # 消息tag
ret = producer.send_sync(msg)
print(ret.status, ret.msg_id, ret.offset)
# 关闭生产者实例，释放资源
producer.shutdown()
```

订阅消息

参考如下示例代码。

```
import time
from rocketmq.client import PushConsumer, ConsumeStatus
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
group = "${GROUP}" # 填写订阅组group，在管理控制台创建
def callback(msg):
  print(msg.id, msg.body)
  return ConsumeStatus.CONSUME_SUCCESS
consumer = PushConsumer(group)
consumer.set_name_server_address(endpoint)
consumer.set_session_credentials(access_key, access_secret, "")
consumer.subscribe(topic, callback)
consumer.start()
while True:
  time.sleep(3600)
consumer.shutdown()
```

收发顺序消息

顺序消息是分布式消息服务RocketMQ版提供的一种严格按照顺序来发布和消费的消息类型。

顺序消息分为全局顺序消息和分区顺序消息：

- 全局顺序消息：对于指定的一个Topic，将队列数量设置为1，这个队列内所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和订阅。
- 分区顺序消息：对于指定的一个Topic，同一个队列内的消息按照严格的FIFO顺序进行发布和订阅。生产者指定分区选择算法，保证需要按顺序消费的消息被分配到同一个队列。

全局顺序消息和分区顺序消息的区别仅为队列数量不同，代码没有区别。

收发消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

准备环境

1. 在命令行输入python，检查是否已安装Python。得到如下回显，说明Python已安装。

```
PS C:\> python
Python 3.9.9 (tags/v3.9.9:ccb0e6a, Nov 15 2021, 18:08:50) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

如果未安装Python，请使用以下命令安装：

```
pip install rocketmq-client-python
```

2. 安装librocketmq库和rocketmq-client-python。

说明

建议下载rocketmq-client-cpp-2.2.0，获取librocketmq库。

3. 将librocketmq.so添加到系统动态库搜索路径。

a. 查找librocketmq.so的路径。

```
find / -name librocketmq.so
```

b. 将librocketmq.so添加到系统动态库搜索路径。

```
ln -s /查找到的librocketmq.so路径/librocketmq.so /usr/lib
sudo ldconfig
```

以下示例代码中的参数说明如下，请参考收集连接信息获取参数值。

- GROUP：表示消费组名称。
- ENDPOINT：表示实例连接地址和端口。
- TOPIC：表示Topic名称。

发送消息

参考如下示例代码。

```
from rocketmq.client import Producer, Message
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
producer_group = "${GROUP}" # 生产者组group
# 创建并启动生产者实例
producer = Producer(producer_group)
producer.set_name_server_address(endpoint)
producer.set_session_credentials(access_key, access_secret, "")
producer.start()
msg = Message(topic)
msg.set_body("Hello RocketMQ")
```

```
msg.set_keys("") # 消息key
msg.set_tags("") # 消息tag
sharding_key = "key" # 指定消息投递的sharding key
# 根据Sharding Key，发送顺序消息，
ret = producer.send_orderly_with_sharding_key(msg, sharding_key)
print(ret.status, ret.msg_id, ret.offset)
# 关闭生产者实例，释放资源
producer.shutdown()
```

订阅消息

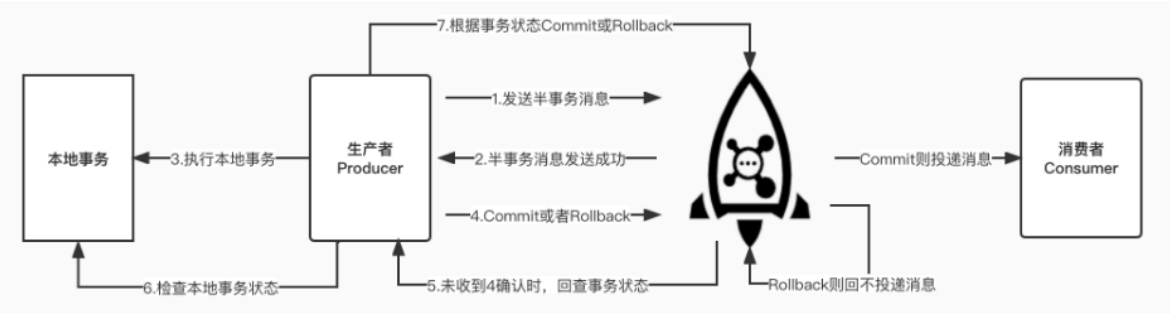
参考如下示例代码。

```
import time
from rocketmq.client import PushConsumer, ConsumeStatus
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
group = "${GROUP}" # 填写订阅组group，在管理控制台创建
def callback(msg):
  print(msg.id, msg.body)
  return ConsumeStatus.CONSUME_SUCCESS
consumer = PushConsumer(group, orderly=True) # 指定消费者为顺序消费类型
consumer.set_name_server_address(endpoint)
consumer.set_session_credentials(access_key, access_secret, "")
consumer.subscribe(topic, callback)
consumer.start()
while True:
  time.sleep(3600)
consumer.shutdown()
```

收发事务消息

分布式消息服务RocketMQ版的事务消息支持在业务逻辑与发送消息之间提供事务保证，通过两阶段的方式提供对事务消息的支持，事务消息交互流程如图1所示。

图1 事务消息交互流程



事务消息生产者首先发送半消息，然后执行本地事务。如果执行成功，则发送事务提交，否则发送事务回滚。服务端在一段时间后如果一直收不到提交或回滚，则发起回查，生产者在收到回查后重新发送事务提交或回滚。消息只有在提交之后才投递给消费者，消费者对回滚的消息不可见。

收发事务消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

准备环境

1. 在命令行输入python，检查是否已安装Python。得到如下回显，说明Python已安装。

```
PS C:\> python
Python 3.9.9 (tags/v3.9.9:ccb0e6a, Nov 15 2021, 18:08:50) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

   如果未安装Python，请使用以下命令安装：

```
pip install rocketmq-client-python
```

2. 安装librocketmq库和rocketmq-client-python。

   说明

   建议下载rocketmq-client-cpp-2.2.0，获取librocketmq库。

3. 将librocketmq.so添加到系统动态库搜索路径。

   a. 查找librocketmq.so的路径。

```
find / -name librocketmq.so
```

   b. 将librocketmq.so添加到系统动态库搜索路径。

```
ln -s /查找到的librocketmq.so路径/librocketmq.so /usr/lib
sudo ldconfig
```

以下示例代码中的参数说明如下，请参考收集连接信息获取参数值。

- GROUP：表示消费组名称。
- ENDPOINT：表示实例连接地址和端口。
- TOPIC：表示Topic名称。

发送消息

参考如下示例代码。

```
import time
```

```
from rocketmq.client import TransactionMQProducer, Message, TransactionStatus
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
producer_group = "${GROUP}" # 生产者组group
def transaction_checker_callback(msg, user_args):
  return TransactionStatus.COMMIT
def transaction_local_execute(msg, user_args):
  return TransactionStatus.UNKNOWN
producer = TransactionMQProducer(producer_group, transaction_checker_callback)
producer.set_name_server_address(endpoint)
producer.set_session_credentials(access_key, access_secret, "")
producer.start()
msg = Message(topic)
msg.set_body("Hello RocketMQ")
ret = producer.send_message_in_transaction(msg, transaction_local_execute, None)
print(ret.status, ret.msg_id, ret.offset)
while True:
  time.sleep(3600)
```

订阅消息

参考如下示例代码。

```
import time
from rocketmq.client import PushConsumer, ConsumeStatus
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
group = "${GROUP}" # 填写订阅组group，在管理控制台创建
def callback(msg):
  print(msg.id, msg.body)
  return ConsumeStatus.CONSUME_SUCCESS
consumer = PushConsumer(group)
consumer.set_name_server_address(endpoint)
consumer.set_session_credentials(access_key, access_secret, "")
consumer.subscribe(topic, callback)
consumer.start()
while True:
  time.sleep(3600)
consumer.shutdown()
```

收发定时/延时消息

分布式消息服务RocketMQ版支持任意时间的定时消息，最大推迟时间可达到40天。

定时消息即生产者生产消息到分布式消息服务RocketMQ版后，消息不会立即被消费，而是延迟到设定的时间点后才会发送给消费者进行消费。

收发消息前，请参考收集连接信息收集RocketMQ所需的连接信息。

准备环境

1. 在命令行输入python，检查是否已安装Python。得到如下回显，说明Python已安装。

```
PS C:\> python
Python 3.9.9 (tags/v3.9.9:ccb0e6a, Nov 15 2021, 18:08:50) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

如果未安装Python，请使用以下命令安装：

```
pip install rocketmq-client-python
```

2. 安装librocketmq库和rocketmq-client-python。

说明

建议下载rocketmq-client-cpp-2.2.0，获取librocketmq库。

3. 将librocketmq.so添加到系统动态库搜索路径。

a. 查找librocketmq.so的路径。

```
find / -name librocketmq.so
```

b. 将librocketmq.so添加到系统动态库搜索路径。

```
ln -s /查找到的librocketmq.so路径/librocketmq.so /usr/lib
sudo ldconfig
```

以下示例代码中的参数说明如下，请参考收集连接信息获取参数值。

- GROUP：表示消费组名称。
- ENDPOINT：表示实例连接地址和端口。
- TOPIC：表示Topic名称。

发送消息

参考如下示例代码。

```
import datetime
from rocketmq.client import Producer, Message
from rocketmq.exceptions import RocketMQException
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
```

```
producer_group = "${GROUP}" # 生产者组group
# 初始化生产者
producer = Producer(producer_group)
producer.set_namesrv_addr(endpoint)
producer.set_session_credentials(access_key, access_secret, "")
# 启动生产者
try:
    producer.start()
except RocketMQException as e:
    print('start producer error:', e)
    exit(1)
msg = Message(topic)
msg.set_body("Hello RocketMQ")
delay_time = 10
# 发送任意延迟消息，时间单位为毫秒，如下所示：消息将在10s后投递
delay_timestamp = int((datetime.datetime.now() + datetime.timedelta(
seconds=delay_time)).timestamp() * 1000)
msg.set_property('__STARTDELIVERTIME', str(delay_timestamp))
# 发送消息
try:
    result = producer.send_sync(msg)
    print('send result:', result)
except RocketMQException as e:
    print('send message error:', e)
    producer.shutdown()
    exit(1)
# 关闭生产者实例，释放资源
producer.shutdown()
```

订阅消息

参考如下示例代码。

```
import time
from rocketmq.client import PushConsumer, ConsumeStatus
endpoint = "${ENDPOINT}" # 填写分布式消息服务RocketMQ控制台Namesrv接入点
access_key = "${ACCESS_KEY}" # 填写AccessKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户ID
access_secret = "${SECRET_KEY}" # 填写SecretKey 在分布式消息服务RocketMQ控制台用户管理菜单中创建的用户密钥
topic = "${TOPIC}" # 填写Topic，在管理控制台创建
group = "${GROUP}" # 填写订阅组group，在管理控制台创建
def callback(msg):
    print(msg.id, msg.body)
    return ConsumeStatus.CONSUME_SUCCESS
consumer = PushConsumer(group)
consumer.set_name_server_address(endpoint)
```

```
consumer.set_session_credentials(access_key, access_secret, "")
consumer.subscribe(topic, callback)
consumer.start()
while True:
  time.sleep(3600)
consumer.shutdown()
```

RocketMQ性能白皮书

测试场景

本章节主要测试RocketMQ不同产品规格在发送1KB大小的消息，实例的网络入流量、网络出流量、消息生产速率、消息消费速率、CPU核均负载和内存使用率。

测试环境

购买实例

名称	规格	存储空间
rocketmq-01	c7.xlarge.2 4C8G	超高I/O 500GB
rocketmq-02	c7.2xlarge.2 8C16G	超高I/O 500GB
rocketmq-03	c7.4xlarge.2 16C32G	超高I/O 500GB

控制台创建主题

名称	权限	关联代理	队列个数
topic-01	发布+订阅	broker-1	8

控制台创建消费组

名称	关联代理	最大重试次数	是否允许以广播模式消费
group-01	broker-1	16	否

测试工具准备

购买1台ECS服务器（区域、可用区、虚拟私有云、子网、安全组与RocketMQ实例保持一致，Linux系统），安装jdk，并配置环境变量

```
tar -zxvf jdk-8u131-linux-x64.tar.gz

vi /etc/profile

#追加以下内容
export JAVA_HOME=/root/jdk1.8.0_131
export PATH=$JAVA_HOME/bin:$PATH

#生效配置
source /etc/profile
```

下载测试工具

wget https://dist.apache.org/repos/dist/release/rocketmq/5.1.4/rocketmq-all-5.1.4-bin-release.zip

解压测试工具

unzip rocketmq-all-5.1.4-bin-release.zip

测试命令

消费命令

sh consumer.sh -n "${namesrv接入地址}" -t ${Topic名称} -g ${消费组名称}

- namesrv接入地址：购买RocketMQ实例后，获取实例的namesrv接入地址
- Topic名称：创建Topic时设置的Topic名称
- 消费组名称：创建消费组时设置的消费组名称

生产命令

sh producer.sh -n "${namesrv接入地址}" -t ${Topic名称} -s 1024 -w ${生产者线程数}

- namesrv接入地址：购买RocketMQ实例后，获取实例的namesrv接入地址
- Topic名称：创建Topic时设置的Topic名称
- 生产者线程数：测试s7.xlarge.4，s7.2xlarge.4，s7.4xlarge.4，c7.xlarge.22时，生产者线程数输入256；测试c7.2xlarge.2，c7.4xlarge.2时，生产者线程数输入640。

关闭命令

#关闭生产者
sh shutdown.sh producer
#关闭消费者
sh shutdown.sh consumer

测试结果

增强型云主机

性能指标	c7.xlarge.2 4C8G	c7.2xlarge.2 8C16G	c7.4xlarge.2 16C32G
消息生产速率	72216 个/秒	125168 个/秒	128226 个/秒
消息消费速率	72142 个/秒	125080 个/秒	128316 个/秒
网络入流量	97.12 MB/S	169.42 MB/S	173.52MB/S
网络出流量	196.39 MB/S	342.25 MB/S	348.51 MB/S
CPU使用率	76%	61%	38%
CPU核均负载	1.43	0.825	0.423

性能指标	c7.xlarge.2 4C8G	c7.2xlarge.2 8C16G	c7.4xlarge.2 16C32G
内存使用率	55%	51%	34%

API使用说明

天翼云OpenAPI门户提供了产品的API 文档、API调试、SDK中心等。

关于用户如何使用分布式消息服务RocketMQ产品API的详细介绍，请参见使用API。您可以在OpenAPI门户可以了解到具体的调用前必知、API概览、如何调用API以及具体的API的接口详细说明。

说明

分布式消息服务RocketMQ提供两种版本接口供用户调用

- 4.0版本：适用于 华东1、华北2、西南1、华南2、上海36、青岛20、长沙42、南昌5、武汉41、杭州7、西南2-贵州、太原4、郑州5、西安7 资源池存量实例调用。
- 3.0版本：适用于 南京3、上海7、重庆2、乌鲁木齐27、保定、石家庄20、内蒙6、晋中、北京5 资源池存量实例调用，部分资源池已不再提供订购入口。

API

2022-04-06

生命周期管理

续订查价

接口功能介绍

续订查价

接口约束

无

URI

POST /v3/instance/queryPriceForRenew

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		

参数	是否必填	参数类型	说明	示例	下级对象
cycleCnt	是	Integer	续费周期，取值范围：值需大于零，不超过384个月		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Array of Objects	价格详情数据列表		data
表 data				

参数	参数类型	说明	示例	下级对象
totalPrice	String	总价格	2445.0	
subOrderPrices	Array of Objects	子订单价格列表		subOrderPrice
finalPrice	String	最终价格	2445.0	
表 subOrderPrice				

参数	参数类型	说明	示例	下级对象
totalPrice	String	子订单总价格	2445.0	
serviceTag	String	服务标签	PAAS	
finalPrice	String	子订单最终价格	2445.0	
orderItemPrices	Array of Objects	订单项价格列表		orderItemPrice
表 orderItemPrice				

参数	参数类型	说明	示例	下级对象
itemId	String	项目ID	f796ba566ee34d818a60fd02411ba178	
totalPrice	String	项目总价格	2205.0	

参数	参数类型	说明	示例	下级对象
finalPrice	String	项目最终价格	2205.0	
resourceType	String	资源类型	PAAS_DMQ	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/queryPriceForRenew

请求头header

无

请求体body


```
{
  "prodInstId": "853f872ddcab4a5c9661c5b5486c91a5",
  "cycleCnt": 1
}
```


响应示例

响应成功示例


```
{
  "message": "success",
  "returnObj": {
    "data": [
      {
        "totalPrice": "2445.0",
        "subOrderPrices": [
          {
            "totalPrice": "2445.0",
            "serviceTag": "PAAS",
            "finalPrice": "2445.0",
            "orderItemPrices": [
              {
                "itemId": "f796ba566ee34d818a60fd02411ba178",
                "totalPrice": "2205.0",
                "finalPrice": "2205.0",
                "resourceType": "PAAS_DMQ"
              },
              {
                "itemId": "c511b35f2ebc4646b94b744386cedbd2",
                "totalPrice": "120.0",
                "finalPrice": "120.0",
```

```
                    "resourceType": "PAAS_DMQ_EBS"
                },
                {
                    "itemId": "c511b35f2ebc4646b94b744386cedbd2",
                    "totalPrice": "120.0",
                    "finalPrice": "120.0",
                    "resourceType": "PAAS_DMQ_EBS"
                }
            ]
        }
    ],
    "finalPrice": "2445.0"
  }
 ]
},
"statusCode": 800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

规格变更查价

接口功能介绍

规格变更查价

接口约束

无

URI

POST /v3/instance/queryPriceForSpecExtend

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
cpuNum	是	Integer	扩容后的cpu核数		
memSize	是	Integer	扩容后的内存大小，单位G		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	String	返回对象。此参数所包含的参数请见“响应示例'里面的注释		
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	价格详情数据		data
表 data				

参数	参数类型	说明	示例	下级对象
totalPrice	String	总价格	11.38	
subOrderPrices	Array of Objects	子订单价格列表（子订单价格项）		subOrderPrice
finalPrice	String	最终价格	11.38	
表 subOrderPrice				

参数	参数类型	说明	示例	下级对象
totalPrice	String	子订单总价格	11.38	
serviceTag	String	服务标签	PAAS	
finalPrice	String	子订单最终价格	11.38	
orderItemPrices	Array of Objects	订单项价格列表		orderItemPrice
表 orderItemPrice				

参数	参数类型	说明	示例	下级对象
itemId	String	项目id	64bb1216bbf711ed96fe34800d471001	
totalPrice	String	项目总价格	11.2	
finalPrice	String	项目最终价格	11.2	
resourceType	String	资源类型（PAAS_DMQ为rocketmq实例，PAAS_DMQ_EBS为云硬盘）	PAAS_DMQ	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/queryPriceForSpecExtend

请求头header

无

请求体body

```
{
  "prodInstId": "string",
  "cpuNum": 4,
  "memSize": 8
}
```

响应示例

响应成功示例

```
{
    "message": "success",
    "returnObj": {
        "data": {
            "totalPrice": "11.38",//总价格
            "subOrderPrices": [
                {
                    "totalPrice": "11.38",//总价格
                    "serviceTag": "PAAS",//服务标签
                    "finalPrice": "11.38",//最终价格
                    "orderItemPrices": [
                        {
                            "itemId": "64bb1216bbf711ed96fe34800d471001",//项目id
                            "totalPrice": "11.2",//总价格
                            "finalPrice": "11.2",//最终价格
```

```
                    "resourceType": "PAAS_DMQ"//资源类型，PAAS_DMQ为rokcetmq实例，PAAS_DMQ_EBS为云硬盘
                },
                {
                    "itemId": "48ab7eedadaa4b00a8e556a06e2076fa",
                    "totalPrice": "0.09",
                    "finalPrice": "0.09",
                    "resourceType": "PAAS_DMQ_EBS"
                },
                {
                    "itemId": "48ab7eedadaa4b00a8e556a06e2076fa",
                    "totalPrice": "0.09",
                    "finalPrice": "0.09",
                    "resourceType": "PAAS_DMQ_EBS"
                }
            ]
        }
    ],//子订单价格项
    "finalPrice": "11.38"//最终价格
  }
 },
 "statusCode": 800
}
响应失败示例
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

查询产品规格

接口功能介绍

查询产品规格

接口约束

无

URI

GET /v3/instance/queryProd

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Array of Objects	规格详情列表		FlavorDetail
表 FlavorDetail				

参数	参数类型	说明	示例	下级对象
flavorID	String	规格id	9b4b5e39-db25-f2c8-3914-76881ee77d5c	
specName	String	规格名称	rocketmq.2u4g.cluster	
flavorType	String	规格类型	C7	
flavorName	String	规格类型名称	计算增强型	
cpuNum	Integer	cpu核数	2	
memSize	Integer	内存大小，单位G	4	
multiQueue	Integer	多队列数	2	
pps	Integer	网络最大收发包能力(万PPS)	40	
bandwidthBase	Float	基准带宽(Gbps)	0.8	
bandwidthMax	Integer	最大带宽(Gbps)	4	
cpuArch	String	cpu架构（x86、arm）	x86	
series	String	系列	C	

参数	参数类型	说明	示例	下级对象
azList	Array of Strings	支持的az名称列表	["cn-huadong1-jsnj1A-public-ctcloud", "cn-huadong1-jsnj3A-public-ctcloud", "cn-huadong1-jsnj2A-public-ctcloud"]	
skuProdId	String	产品id	201101	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/queryProd

请求头header

无

请求体body

无

响应示例

响应成功示例

```
{
    "message": "success",
    "returnObj": {
        "data": [
            {
                "flavorID": "9b4b5e39-db25-f2c8-3914-76881ee77d5c",//规格id
                "specName": "rocketmq.2u4g.cluster",//规格名称
                "flavorType": "C7",//规格类型
                "flavorName": "计算增强型",//规格类型名称
                "cpuNum": 2,//cpu核数
                "memSize": 4,//内存大小，单位G
                "multiQueue": 2,//多队列数
                "pps": 40,//网络最大收发包能力(万PPS)
                "bandwidthBase": 0.8,//基准带宽(Gbps)
                "bandwidthMax": 4,//最大带宽(Gbps)
                "cpuArch": "x86",//cpu架构 （x86、arm）
                "series": "C",//系列
                "azList": [
                    "cn-huadong1-jsnj1A-public-ctcloud",
```

```
                        "cn-huadong1-jsnj3A-public-ctcloud",
                        "cn-huadong1-jsnj2A-public-ctcloud"
                    ],//支持的az名称列表
                    "skuProdId": "201101"//产品id
                }
            ]
        },
        "statusCode": 800
}
```


响应失败示例


```
{
   "returnObj": {},
   "message": "...",
   "error":"ROCKETMQ_1201",
   "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

开通查价

接口功能介绍

开通查价

接口约束

无

URI

POST /v3/instance/queryPrice

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
cycleType	是	String	计费模式101:按需 3:按月订购	101	

参数	是否必填	参数类型	说明	示例	下级对象
cycleCnt	否	String	cycleType=3时，必填；订购周期，取值范围：值需大于零，不超过384个月		
specName	是	String	规格名	rocketmq.2u4g.cluster	
nodeNum	是	Integer	节点数	2	
diskType	是	String	存储类型 SAS、SSD	SSD	
diskSize	是	Integer	节点存储空间大小,单位G	100	

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
itemId	String	订单项ID	f796ba566ee34d818a60fd02411ba178	
totalPrice	String	订单项总价格	2205.0	
finalPrice	String	订单项最终价格	2205.0	
resourceType	String	资源类型	PAAS_DMQ	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/queryPrice

请求头header

无

请求体body


```
{
    "cycleType": "3",
    "cycleCnt": "1",
    "engineType": "ctgmq-advance",
    "diskSize": 100,
    "diskType": "SSD",
    "nodeNum": 2,
    "specName":"rocketmq.2u4g.cluster"
}
```


响应示例

响应成功案例


```
{
    "message": "success",
    "returnObj": {
        "data": {
            "totalPrice": "2445.0",
            "subOrderPrices": [
                {
                    "totalPrice": "2445.0",
                    "serviceTag": "PAAS",
                    "finalPrice": "2445.0",
                    "orderItemPrices": [
                        {
                            "itemId": "f796ba566ee34d818a60fd02411ba178",
                            "totalPrice": "2205.0",
                            "finalPrice": "2205.0",
                            "resourceType": "PAAS_DMQ"
                        },
                        {
                            "itemId": "c511b35f2ebc4646b94b744386cedbd2",
                            "totalPrice": "120.0",
                            "finalPrice": "120.0",
                            "resourceType": "PAAS_DMQ_EBS"
                        },
                        {
                            "itemId": "c511b35f2ebc4646b94b744386cedbd2",
                            "totalPrice": "120.0",
                            "finalPrice": "120.0",
                            "resourceType": "PAAS_DMQ_EBS"
                        }
```

```
                    ]
                }
            ],
            "finalPrice": "2445.0"
        }
    },
    "statusCode": 800
}
```


响应失败案例


```
{
 "statusCode": 900,
 "message": "engineType not valid, valid perms is ctgmq-advance",
 "returnObj": null,
 "error": "ROCKETMQ_0500"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

磁盘扩容查价

接口功能介绍

磁盘扩容查价

接口约束

无

URI

POST /v3/instance/queryPriceForDiskExtend

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
diskExtendSize	是	Integer	每个节点扩容后的存储空间，单位G		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
totalPrice	String	子订单总价格	5.8	
serviceTag	String	服务标签	PAAS	
finalPrice	String	子订单最终价格	5.8	
orderItemPrices	Array of Objects	订单子项价格列表		OrderItemPrice
表 OrderItemPrice				

参数	参数类型	说明	示例	下级对象
itemId	String	项目id	f796ba566ee34d818a60fd02411ba178	
totalPrice	String	项目总价格	4.9	
finalPrice	String	项目最终价格	4.9	
resourceType	String	资源类型，PAAS_DMQ为rocketmq实例，PAAS_DMQ_EBS为云硬盘	PAAS_DMQ	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/queryPriceForDiskExtend

请求头header

无

请求体body

```
{
  "prodInstId":"string",
  "diskExtendSize":300
}
```

响应示例

响应成功示例

```
{
    "message":"success",
    "returnObj":{
        "data":{
            "totalPrice":"5.8",
            "subOrderPrices":[
                {
                    "totalPrice":"5.8",
                    "serviceTag":"PAAS",
                    "finalPrice":"5.8",
                    "orderItemPrices":[
                        {
                            "itemId":"f796ba566ee34d818a60fd02411ba178",//项目id
                            "totalPrice":"4.9",//总价格
                            "finalPrice":"4.9",//最终价格
                            "resourceType":"PAAS_DMQ"// 资源类型，PAAS_DMQ为rokcetmq实
例，PAAS_DMQ_EBS为云硬盘
                        },
                        {
                            "itemId":"48ab7eedadaa4b00a8e556a06e2076fa",
                            "totalPrice":"0.45",
                            "finalPrice":"0.45",
                            "resourceType":"PAAS_DMQ_EBS"
                        },
                        {
                            "itemId":"48ab7eedadaa4b00a8e556a06e2076fa",
                            "totalPrice":"0.45",
                            "finalPrice":"0.45",
                            "resourceType":"PAAS_DMQ_EBS"
                        }
                    ]
                }
            ],//子订单价格项
            "finalPrice":"5.8" // 最终价格
        }
    },
    "statusCode":800
}
```

响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

规格变更

接口功能介绍

变更实例节点机器规格

接口约束

无

URI

POST /v3/instance/specExtend

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id		
cpuNum	是	Integer	扩容后的cpu核数	1000	
memSize	是	Integer	扩容后的内存大小，单位G		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		

参数	参数类型	说明	示例	下级对象
returnObj	String	返回对象		returnObj
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	返回的订单相关数据		data
表 data				

参数	参数类型	说明	示例	下级对象
submitted	Boolean	是否已经提交	true	
newOrderId	String	订单id	34fec217c5504bc7b8ae9cb8c63ef135	
newOrderNo	String	订单号	20230216102127451788	
totalPrice	String	总价格	3057.0	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/specExtend

请求头header

无

请求体body

```
{
  "prodInstId": "string",
  "cpuNum": 2,
  "memSize": 4
}
```

响应示例

响应成功示例

```
{
    "returnObj": {
        "data": {
            "submitted": true, // 是否已经提交
            "newOrderId": "34fec217c5504bc7b8ae9cb8c63ef135", //订单id
            "newOrderNo": "20230216102127451788", //订单号
            "totalPrice": "3057.0"//总价格
```

```
            }
        },
        "message": "success",
        "statusCode": "800"
}
```


响应失败示例


```
{
   "returnObj": {},
   "message": "...",
   "error":"201",
   "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

磁盘扩容

接口功能介绍

实例磁盘扩容

接口约束

无

URI

POST /v3/instance/diskExtend

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id		
diskExtendSize	是	String	每个节点扩容后的存储空间，单位G		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功："800"，失败："900"。	800	
message	String	接口调用状态描述。成功时为"success"，失败时为具体失败信息	success	
returnObj	Object	核心返回对象。成功时包含订单数据，失败时为空对象		returnObj
error	String	错误码。仅失败时返回，描述具体错误信息	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	订单核心数据。仅接口调用成功时返回		OrderData
表 OrderData				

参数	参数类型	说明	示例	下级对象
submitted	Boolean	订单是否提交成功标识	true	
newOrderId	String	系统生成的订单唯一标识ID	09838a19f1474f94a8ef0a9a5d9c9ed3	
newOrderNo	String	系统生成的业务订单编号	20230216102712059731	
totalPrice	String	订单总价格（单位：元）	348.5	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/diskExtend

请求头header

无

请求体body

```
{
  "prodInstId": "string",
  "diskExtendSize": 300
}
```

响应示例

响应成功示例

```
{
    "returnObj": {
        "data": {
            "submitted": true,// 是否已经提交
            "newOrderId": "09838a19f1474f94a8ef0a9a5d9c9ed3",//订单id
            "newOrderNo": "20230216102712059731",//订单号
            "totalPrice": "348.5"//总价格
        }
    },
    "message": "success",
    "statusCode": "800"
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"ROCKETMQ_1201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

节点扩容

接口功能介绍

实例节点扩容

接口约束

无

URI

POST /v3/instance/nodeExtend

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id		
extendNodeNum	是	Integer	扩容后实例的节点数量，对应取值为等于代理数*2，范围为[4,32]		

响应参数

无

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/nodeExtend

请求头header

regionId:bb9fdb42056f11eda1610242ac110002

请求体body

```
{
   "prodInstId": "789395f32e454c57b2a88c186c68e5d2",
   "extendNodeNum": 4
}
```

响应示例

```
{
 "returnObj": {
   "data": {
     "submitted": true,
     "newOrderId": "09838a19f1474f94a8ef0a9a5d9c9ed3",
     "newOrderNo": "20230216102712059731",
     "totalPrice": "348.5"
   }
 },
```

"message": "success",
"statusCode": "800"
}

状态码

请参考 状态码

错误码

请参考 错误码

退订查价
接口功能介绍

退订查价

接口约束

无

URI

POST /v3/instance/queryPriceForUnsubscribe

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	响应对象		data
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 data				

参数	参数类型	说明	示例	下级对象
paidPrice	String	支付金额		
refundPrice	String	退款金额		

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/queryPriceForUnsubscribe

请求头header

regionId: bb9fdb42056f11eda1610242ac110002

请求体body

```
{
 "prodInstId": "789395f32e454c57b2a88c186c68e5d2",
}
```

响应示例

```
{
   "message": "success",
   "returnObj": {
      "data": {
         "paidPrice": "32060.00",
         "refundPrice": "31901.21"
      }
   },
   "statusCode": 800
}
```

状态码

请参考 状态码

错误码

请参考 错误码

退订

接口功能介绍

退订

接口约束

无

URI

POST /v3/instance/unsubscribeInst

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	响应数据对象		data
表 data				

参数	参数类型	说明	示例	下级对象
errorMessage	String	错误信息描述		
batchOrderPlacementResults	Array of Objects	批量下单结果列表		batchOrderPlacementResult
表 batchOrderPlacementResult				

参数	参数类型	说明	示例	下级对象
errorMessage	String	错误信息描述		

参数	参数类型	说明	示例	下级对象
submitted	Boolean	是否提交成功		
orderPlacedEvents	Array of Objects	下单事件列表		orderPlacedEvent
表 orderPlacedEvent				

参数	参数类型	说明	示例	下级对象
errorMessage	String	错误信息描述		
submitted	String	是否提交成功		
newOrderId	String	订单id		
newOrderNo	String	订单编号		
totalPrice	Double	订单价格		

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/unsubscribeInst

请求头header

无

请求体body

```
{"prodInstId":"string"}
```

响应示例

响应成功示例

```
{
  "statusCode":800,
  "returnObj":{
    "errorMessage":"",
    "batchOrderPlacementResults":[
      {
        "errorMessage":"",
        "submitted":true,
        "orderPlacedEvents":[
          {
            "errorMessage":"",
            "submitted":true,
            "newOrderId":"96ec06154feb4da89bf4262f88ada76e",
            "newOrderNo":"20180404165441398225",
            "totalPrice":100
```

```
            }
          ]
        },
        {
          "errorMessage": "",
          "submitted": true,
          "orderPlacedEvents": [
            {
              "errorMessage": "",
              "submitted": true,
              "newOrderId": "38159a060c2f4618a762f0982596dac8",
              "newOrderNo": "20180404165441476438",
              "totalPrice": 100
            }
          ]
        },
        {
          "errorMessage": "",
          "submitted": true,
          "orderPlacedEvents": [
            {
              "errorMessage": "",
              "submitted": true,
              "newOrderId": "cfa120962c5447b3b4b66f9136087295",
              "newOrderNo": "20180404165441568372",
              "totalPrice": 100
            }
          ]
        }
      ]
    },
    "message": "success"
}
```


响应失败示例


```
{
    "returnObj": {},
    "message": "...",
    "error":"201",
    "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

节点扩容查价
接口功能介绍

节点扩容查价

接口约束

无

URI

POST /v3/instance/queryPriceForNodeExtend

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
extendNodeNum	是	Integer	扩容后的节点数		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	价格详情数据		data
表 data				

参数	参数类型	说明	示例	下级对象
totalPrice	String	总价格	7.22	
subOrderPrices	Array of Objects	子订单价格列表		subOrderPrice
finalPrice	String	最终价格	7.22	
表 subOrderPrice				

参数	参数类型	说明	示例	下级对象
totalPrice	String	子订单总价格	7.22	
serviceTag	String	服务标签	PAAS	
finalPrice	String	子订单最终价格	7.22	
orderItemPrices	Array of Objects	订单项价格列表		orderItemPrice
表 orderItemPrice				

参数	参数类型	说明	示例	下级对象
itemId	String	项目id	f796ba566ee34d818a60fd02411ba178	
totalPrice	String	项目总价格	6.86	
finalPrice	String	项目最终价格	6.86	
resourceType	String	资源类型（PAAS_DMQ为rocketmq实例，PAAS_DMQ_EBS为云硬盘）	PAAS_DMQ	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/queryPriceForNodeExtend

请求头header

无

请求体body

```
{
  "prodInstId": "string",
  "extendNodeNum": 4
}
```

响应示例

响应成功示例

```
{
    "message": "success",
    "returnObj": {
        "data": {
            "totalPrice": "7.22",
            "subOrderPrices": [
                {
                    "totalPrice": "7.22", //总价格
                    "serviceTag": "PAAS", //服务标签
                    "finalPrice": "7.22", //最终价格
                    "orderItemPrices": [
                        {
                            "itemId": "f796ba566ee34d818a60fd02411ba178", //项目id
                            "totalPrice": "6.86", //总价格
                            "finalPrice": "6.86", //最终价格
                            "resourceType": "PAAS_DMQ"// 资源类型，PAAS_DMQ为rokcetmq实
例，PAAS_DMQ_EBS为云硬盘
                        },
                        {
                            "itemId": "48ab7eedadaa4b00a8e556a06e2076fa",
                            "totalPrice": "0.09",
                            "finalPrice": "0.09",
                            "resourceType": "PAAS_DMQ_EBS"
                        },
                        {
                            "itemId": "48ab7eedadaa4b00a8e556a06e2076fa",
                            "totalPrice": "0.09",
                            "finalPrice": "0.09",
                            "resourceType": "PAAS_DMQ_EBS"
                        },
                        {
                            "itemId": "48ab7eedadaa4b00a8e556a06e2076fa",
                            "totalPrice": "0.09",
                            "finalPrice": "0.09",
                            "resourceType": "PAAS_DMQ_EBS"
                        },
                        {
                            "itemId": "48ab7eedadaa4b00a8e556a06e2076fa",
                            "totalPrice": "0.09",
                            "finalPrice": "0.09",
                            "resourceType": "PAAS_DMQ_EBS"
                        }
                    ]
                }
```

```
            ],//子订单价格项
            "finalPrice": "7.22"//最终价格
        }
    },
    "statusCode": 800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

创建实例

接口功能介绍

开通RocketMQ实例

接口约束

无

URI

POST /v3/instance/createPostPayOrder

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
specName	是	String	规格名称，通过规格接口获取支持的规格列表	rocketmq.2u4g.cluster	

参数	是否必填	参数类型	说明	示例	下级对象
billMode	否	Integer	计费模式取值范围:1为预付费，即包年包月，包周期；2为后付费，即按需；默认值为按需。	2	
cycleCnt	否	Integer	billMode为1时生效且为必选值，表示预付费的订购周期时长，单位：月。取值范围：1，2，3，4，5，6，12，24，36	1	
autoRenew	否	Boolean	预付费实例是否自动续订取值范围：true：自动续订。false：不自动续订。默认不自动续订。	false	
autoRenewCycleCount	否	Integer	autoRenew为true时生效且为必选值，表示自动续订的周期时长，单位：月。取值范围：1，2，3，4，5，6，12，24，36	1	
nodeNum	是	Integer	broker的节点数，值等于代理数*2，取值范围为[1，32]，单机版传1	2	
diskType	是	String	存储类型 SAS、SSD、FAST-SSD	SSD	
diskSize	是	Integer	节点存储空间大小,单位G	100	
azInfo	是	String	az信息通过规格接口查询，支持单可用区和三可用区		
vpcId	是	String	私有云ID		
securityGroupId	是	String	安全组ID		
subnetId	是	String	子网ID		
clusterName	是	String	实例名称	MQ-openAPI	

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	String	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	订单核心数据。仅接口调用成功时返回		OrderData
表 OrderData				

参数	参数类型	说明	示例	下级对象
submitted	Boolean	订单是否提交成功标识	true	
newOrderId	String	系统生成的订单唯一标识ID	9a439b4646784662a0672cc3df35b164	
newOrderNo	String	系统生成的业务订单编号	20250716144120097418	
totalPrice	String	订单总价格。按需返回	null	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/createPostPayOrder

请求头header

regionId:bb9fdb42056f11eda1610242ac110002

请求体body

```
{
 "specName":"rocketmq.4u8g.cluster",
 "billMode":1,
 "cycleCnt":1,
```

```
 "engineType": "ctgmq-advance",
 "nodeNum": 2,
 "diskType": "SAS",
 "diskSize": 100,
 "azInfo":"[{\"azName\":\"cn-xinan1-1A\",\"azId\":337}]",
 "vpcId": "vpc-7reh61pveb",
 "subnetId": "subnet-abug44gdbf",
 "securityGroupId": "sg-o0mrfucmol",
 "clusterName":"MQtest_API"
}
```


响应示例


```
{
 "message": "success",
 "returnObj": {
  "data": {
   "submitted": true, //是否已经提交
   "newOrderId": "9a439b4646784662a0672cc3df35b164", // 订单id
   "newOrderNo": "20250716144120097418", // 订单号
   "totalPrice": null// 总价格，按需该字段为null
  }
 },
 "statusCode": 800
}
```


状态码

请参考 状态码

错误码

请参考 错误码

实例管理

更新实例名称
接口功能介绍

更新实例名称

接口约束

无

URI

POST /v3/instance/updateName

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
instanceName	是	String	新的实例名称		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/updateName

请求头header

无

请求体body

```
{
  "prodInstId":"mq_test",
  "instanceName":"70687660456281088"
}
```

响应示例

响应成功示例

```
{
  "returnObj":{
```

```
  },
  "message":"success",
  "statusCode":800
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

查询实例列表

接口功能介绍

实例列表

接口约束

无

URI

GET /v3/instance/list

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功："800"，失败："900"。	800	

参数	参数类型	说明	示例	下级对象
message	String	接口调用状态描述。成功时为"success"，失败时为具体失败信息	success	
returnObj	Object	核心返回对象。成功时包含MQ实例列表数据，失败时为空对象		returnObj
error	String	错误码。仅失败时返回，描述具体错误信息	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
prodInstList	Array of Objects	MQ实例列表数据。仅接口调用成功时返回		ProdInstInfo
表 ProdInstInfo				

参数	参数类型	说明	示例	下级对象
prodInstId	String	实例id	091081ef b5c34cb78a0fcbcd9e253515	
billMode	String	计费模式 1：包周期；2：按需	2	
prodInstName	String	MQ实例名称	MQ2-ok9xtk	
runningState	String	实例运行状态	运行中	
state	Integer	实例状态编码 1.运行中 2.已过期 3.已注销 4.已退订 5.扩容中 6.开通中 7.已取消 8.缩容中 9.重启中 10.网络变更中 11.运维恢复 12.运维停止 13.异常中 15.已欠费 -1.变更 101.开通失败	1	
prodType	String	产品类型	单节点、1M1S	
machineSpec	String	机器规格名	rocketmq.2u4g.single、rocketmq.4u8g.cluster	
topicLimit	String	Topic数量限制	200	
diskSpace	String	磁盘空间大小	100G	
netName	String	网络名称（VPC名称）	vpc-48fe-victor、redis-1	

参数	参数类型	说明	示例	下级对象
subnet	String	子网名称	subnet-913b、redis-subnet-01	
securityGroup	String	安全组ID	51170、44115	
modTime	String	最后修改时间（UTC格式）	2025-10-31T18:28:14.000+00:00	
deployType	String	部署类型编码 1-虚机部署	1	
diskType	String	磁盘类型	SSD	
diskIsEncrypt	Boolean	磁盘是否加密	false	
fileReservedTime	Integer	文件保留时间（单位：小时）	168	
clusterProperties	Object	集群配置属性		ClusterProperties
engineType	String	引擎类型	ctgmq-advance	
labels	Array of Objects	标签列表		LabelInfo
vip	String	实例VIP地址	192.168.23.30、192.168.1.129	
clusterType	Integer	集群类型编码 1-单机版 2-集群版	1	
version	String	版本号	5.x、4.9.6	
nodeSize	Integer	broker节点数量	1	
outerProjectName	String	企业项目名称	default	
vpcId	String	VPC ID	18414、17372	
enableIpv6	Boolean	是否启用IPv6	false	
resources	Array of Strings	资源列表	[]	
expandAccess	Array of Strings	可扩展权限列表 MQ2_INST_VM_EXPAND-规格扩容；MQ2_INST_DISK_EXPAND-磁盘扩容；MQ2_INST_NODE_EXPAND-节点扩容	["MQ2_INST_VM_EXPAND", "MQ2_INST_DISK_EXPAND"]	
shrinkAccess	Array of Strings	可缩容权限列表 MQ2_INST_VM_SHRINK-规格缩容 当前只支持规格缩容	["MQ2_INST_VM_SHRINK"]	
crtTime	String	创建时间（UTC格式）	2025-10-31T09:49:24.000+00:00	
表 ClusterProperties				

参数	参数类型	说明	示例	下级对象
enableTpsLimit	String	是否启用TPS限制。0：未启用，1：已启用	0	
deleteWhen	String	清理时间配置	4	
tpsLimit	String	集群级TPS限制值	2500、5000	
maxQueuesPerTopic	String	每个Topic最大队列数	8	
表 LabelInfo				

参数	参数类型	说明	示例	下级对象
labelId	String	标签唯一标识ID	c2f03247 41f349feb2e557e3f09cce7f	
key	String	标签键	RY-key-11856	
value	String	标签值	RY-val-11856	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/list

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
    "message": "success",
    "returnObj": {
        "prodInstList": [
            {
                "prodInstId": "091081efb5c34cb78a0fcbcd9e253515",
                "billMode": "2",
                "prodInstName": "MQ2-ok9xtk",
                "runningState": "运行中",
                "state": 1,
                "prodType": "单节点",
                "machineSpec": "rocketmq.2u4g.single",
```

```
        "topicLimit": "200",
        "diskSpace": "100G",
        "netName": "vpc-48fe-victor",
        "subnet": "subnet-913b",
        "securityGroup": "51170",
        "modTime": "2025-10-31T18:28:14.000+00:00",
        "deployType": "1",
        "diskType": "SSD",
        "diskIsEncrypt": false,
        "fileReservedTime": 168,
        "clusterProperties": {
            "enableTpsLimit": "0",
            "deleteWhen": "4",
            "tpsLimit": "2500",
            "maxQueuesPerTopic": "8"
        },
        "engineType": "ctgmq-advance",
        "labels": [],
        "vip": "192.168.23.30",
        "clusterType": 1,
        "version": "5.x",
        "nodeSize": 1,
        "outerProjectName": "default",
        "vpcId": "18414",
        "enableIpv6": false,
        "resources": [],
        "expandAccess": [
            "MQ2_INST_VM_EXPAND",
            "MQ2_INST_DISK_EXPAND"
        ],
        "shrinkAccess": [
            "MQ2_INST_VM_SHRINK"
        ],
        "crtTime": "2025-10-31T09:49:24.000+00:00"
    },
    {
        "prodInstId": "079ba066b4fc4bb694696d14d27ddb63",
        "billMode": "2",
        "prodInstName": "MQ2-7wrfcp",
        "runningState": "运行中",
        "state": 1,
        "prodType": "1M1S",
        "machineSpec": "rocketmq.4u8g.cluster",
        "topicLimit": "200",
        "diskSpace": "100G",
        "netName": "redis-1",
```

```
                    "subnet": "redis-subnet-01",
                    "securityGroup": "44115",
                    "modTime": "2025-11-12T03:46:16.000+00:00",
                    "deployType": "1",
                    "diskType": "SSD",
                    "diskIsEncrypt": false,
                    "fileReservedTime": 168,
                    "clusterProperties": {
                        "enableTpsLimit": "0",
                        "deleteWhen": "4",
                        "tpsLimit": "5000",
                        "maxQueuesPerTopic": "8"
                    },
                    "engineType": "ctgmq-advance",
                    "labels": [],
                    "vip": "192.168.1.129",
                    "clusterType": 2,
                    "version": "4.9.6",
                    "nodeSize": 2,
                    "outerProjectName": "default",
                    "vpcId": "17372",
                    "enableIpv6": false,
                    "resources": [],
                    "expandAccess": [
                        "MQ2_INST_VM_EXPAND",
                        "MQ2_INST_DISK_EXPAND",
                        "MQ2_INST_NODE_EXPAND"
                    ],
                    "shrinkAccess": [
                        "MQ2_INST_VM_SHRINK"
                    ],
                    "crtTime": "2025-11-12T03:46:13.000+00:00"
                }
            ]
        },
        "statusCode": 800
}
```


响应失败示例


```
{
    "returnObj": {},
    "message": "...",
    "error":"201",
    "statusCode": 900
}
```

状态码

请参考 状态码

错误码

请参考 错误码

查询实例基本信息
接口功能介绍

查询实例基本信息

接口约束

无

URI

GET /v3/instance/baseInfo

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id	079ba066b4fc4bb694696d14d27ddb63	

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功："800"，失败："900"。	800	
message	String	接口调用状态描述。成功时为"success"，失败时为具体失败信息	success	
returnObj	Object	核心返回对象		returnObj
error	String	错误码。仅失败时返回，描述具体错误信息	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
instanceBaseInfo	Object	MQ实例基础信息详情。仅接口调用成功时返回		InstanceBaseInfo
表 InstanceBaseInfo				

参数	参数类型	说明	示例	下级对象
prodInstId	String	实例id	079ba066b4fc4bb694696d14d27ddb63	
billMode	String	计费模式 1：包周期；2：按需	2	
prodInstName	String	MQ实例名称	MQ2-7wrfcp	
runningState	String	实例运行状态	运行中	
prodType	String	产品类型	1M1S	
machineSpec	String	机器规格	rocketmq.4u8g.cluster	
tpsLimit	String	TPS限制值	5000	
topicLimit	String	Topic数量限制。按需返回，可能为null	null	
diskSpace	String	磁盘空间大小	100G	
netName	String	网络名称（VPC名称）	redis-1	
subnet	String	子网名称	redis-subnet-01	
securityGroup	String	安全组ID	44115	
expTime	String	过期时间。按需返回，可能为null	null	
nameServer	String	NameServer地址列表，多个地址以分号分隔	192.168.1.134:9101;192.168.1.132:9101;192.168.1.130:9101	
crtTime	String	创建时间（UTC格式）	2025-11-12T03:46:13.000+00:00	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/instance/baseInfo

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
    "message": "success",
    "returnObj": {
        "instanceBaseInfo": {
            "prodInstId": "079ba066b4fc4bb694696d14d27ddb63",
            "billMode": "2",
            "prodInstName": "MQ2-7wrfcp",
            "runningState": "运行中",
            "prodType": "1M1S",
            "machineSpec": "rocketmq.4u8g.cluster",
            "tpsLimit": "5000",
            "topicLimit": null,
            "diskSpace": "100G",
            "netName": "redis-1",
            "subnet": "redis-subnet-01",
            "securityGroup": "44115",
            "expTime": null,
            "nameServer": "192.168.1.134:9101;192.168.1.132:9101;192.168.1.130:9101",
            "crtTime": "2025-11-12T03:46:13.000+00:00"
        }
    },
    "statusCode": 800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": 900
}
```


状态码

请参考 状态码

错误码

请参考 错误码

Topic管理

查询主题一段时间内消息写入tps信息

接口功能介绍

查询主题一段时间内消息写入tps信息

接口约束

无

URI

GET /v3/topic/inputTps

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topicName	是	String	Topic名字		
brokerName	是	String	Broker名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	主题TPS统计数据		data
表 data				

参数	参数类型	说明	示例	下级对象
prodInstId	String	实例id	mq_test	
topicName	String	主题名称	test	
tpsVoList	Array of Objects	TPS统计列表		TpsVo

参数	参数类型	说明	示例	下级对象
表 TpsVo				

参数	参数类型	说明	示例	下级对象
tps	Double	消息吞吐量（条/秒）	0.3333333333333333	
crtTime	Long	统计时间戳（毫秒）	1661937180392	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/topic/inputTps?prodInstId=70687660456281088&topicName=test&brokerName=

请求头header

无

请求体body

无

响应示例


```
{
  "returnObj":{
    "data":{
      "prodInstId":"mq_test",
      "topicName":"test",
      "tpsVoList":[
        {
          "tps":0,
          "crtTime":1661937180392
        },
        {
          "tps":0.3,
          "crtTime":1661937200405
        },
        {
          "tps":0.3333333333333333,
          "crtTime":1661937220359
        }
      ]
    }
  },
  "message":"success",
```

"statusCode":800
}

响应失败示例

{
"returnObj": {},
"message": "...",
"error":"201",
"statusCode": "900"
}

状态码

请参考 状态码

错误码

请参考 错误码

获取Topic列表信息

接口功能介绍

获取Topic列表信息

接口约束

无

URI

GET /v3/topic/list

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topicName	否	String	topic名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		

参数	参数类型	说明	示例	下级对象
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
rows	Array of Objects	主题信息列表		rows
total	Integer	主题总数量	2	
表 rows				

参数	参数类型	说明	示例	下级对象
topicName	String	主题名称	topic001	
readQueueNums	Integer	读队列数量	4	
writeQueueNums	Integer	写队列数量	4	
perm	Integer	读写权限 2	6	
topicFilterType	String	主题过滤标志		
topicSysFlag	Integer	主题系统标识		
0 - 表示非系统主题				
1 - 系统内部主题	0			
order	Boolean	是否为顺序消息	false	
remark	String	备注		
clusterName	String	集群名		
brokerName	String	集群名称		
brokerId	String	代理id	0	
messageType	String	消息类型		
NORMAL-普通消息	NORMAL			

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/topic/list?prodInstId=70687660456281088

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "returnObj":{
    "total":1,
    "rows":[
      {
        "topicName":"test",
        "readQueueNums":4,
        "writeQueueNums":4,
        "perm":6,
        "topicSysFlag":0,
        "order":false,
        "clusterName":"mq_test",
        "brokerName":"mq_test_broker_1",
        "brokerId":0
      }
    ]
  },
  "message":"success",
  "statusCode":800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

删除主题
接口功能介绍

删除主题

接口约束

无

URI

POST /v3/topic/delete

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id	d7ed2dbad63843f4bcb3b9dc0955a617	
topicName	是	String	主题名称	topicTest	

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功："800"，失败："900"。	800	
message	String	描述状态。		
returnObj	Object	返回对象		
error	String	错误码，描述错误信息。		

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/topic/delete

请求头header

无

请求体body

```
{
    "topicName": "topicTest",
    "prodInstId": "d7ed2dbad63843f4bcb3b9dc0955a617"
}
```

响应示例

响应成功示例

```
{
  "returnObj":{},
  "message":"success",
  "statusCode":800
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

查看Topic的订阅信息

接口功能介绍

查看Topic的订阅信息

接口约束

无

URI

GET /v3/topic/subDetail

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topicName	是	String	topic名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象。		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	主题订阅信息		data
表 data				

参数	参数类型	说明	示例	下级对象
subscriptionDataList	Array of Objects	订阅数据列表		subscriptionData
topic	String	主题名称	test	
prodInstId	String	实例id	mq_test	
表 subscriptionData				

参数	参数类型	说明	示例	下级对象
consumerGroup	String	消费者组	group	
subString	String	订阅表达式	*	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/topic/subDetail?prodInstId=70687660456281088&topicName=test

请求头header

无

请求体body

无

响应示例

响应成功示例

```
{
  "returnObj":{
    "data":{
      "subscriptionDataList":[
        {
          "consumerGroup":"group",
          "subString":"*"
        }
      ],
      "topic":"test",
      "prodInstId":"mq_test"
    }
  },
  "message":"success",
  "statusCode":800
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

配置Topic的读写模式
接口功能介绍

配置Topic的读写模式

接口约束

无

URI

POST /v3/topic/update

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topic	是	String	主题名称		
perm	是	Integer	设置该Topic的读写模式。取值说明如下：6：同时支持读写;4：禁写;2：禁读		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/topic/update

请求头header

无

请求体body

```
{
      "perm": 4,
      "topic": "topicTest",
      "prodInstId": "d7ed2dbad63843f4bcb3b9dc0955a617"
}
```

响应示例

响应成功示例

```
{
   "returnObj":{},
   "message":"success",
   "statusCode":800
}
```

响应失败示例

```
{
   "returnObj": {},
   "message": "...",
   "error":"201",
   "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

查询Topic状态v3

接口功能介绍

查询Topic状态

接口约束

无

URI

GET /v3/topic/status

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topicName	是	String	topic名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败："900"	800	
message	String	描述状态	success	
returnObj	Object	返回对象		returnObj
error	String	错误码，描述错误信息	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	响应数据对象		data
表 data				

参数	参数类型	说明	示例	下级对象
perm	Integer	权限标识	6	
totalCount	Integer	总记录数	27	
lastTimeStamp	Long	最新时间戳	1661857434599	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/topic/status?prodInstId=70687660456281088&topicName=test

请求头header

无

请求体body

无

响应示例

响应成功示例

```
{
  "returnObj":{
    "data":{
      "perm":6,
      "totalCount":27,
      "lastTimeStamp":1661857434599
    }
  },
  "message":"success",
  "statusCode":800
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

创建主题

接口功能介绍

创建主题

接口约束

无

URI

POST /v3/topic/create

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id	d7ed2dba d63843f4bcb3b9dc0955a617	
brokerNameList	是	Array of Strings	关联的Broker节点名称列表，需填写实际部署的Broker节点名称，不可为空数组	["broker_1"]	
writeQueueNums	是	Integer	主题的写队列数量，用于控制消息写入的并发能力，需为正整数	4	
readQueueNums	是	Integer	主题的读队列数量，需与写队列数量保持一致，确保消息消费的负载均衡，需为正整数	4	
order	是	Boolean	标识主题是否为顺序消息队列，true表示顺序队列，false表示普通队列	false	
perm	是	Integer	主题的权限控制值，固定可选值为2（只读）、4（只写）、6（读写），默认推荐6	6	
allowdConsumerGroups	是	Array of Strings	允许订阅该主题的消费者组列表，空数组表示不限制消费者组订阅	[]	
topicName	是	String	主题名称，需符合MQ主题命名规范（字母、数字、下划线组合，长度1-64字符），不可重复	test1	

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功："800"，失败："900"。	800	

参数	参数类型	说明	示例	下级对象
message	String	描述状态。		
returnObj	Object	返回对象。		
error	String	错误码，描述错误信息。		

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/topic/create

请求头header

无

请求体body

```
{
    "readQueueNums": 4,
    "perm": 6,
    "writeQueueNums": 4,
    "topicName": "topicTest",
    "prodInstId": "d7ed2dbad63843f4bcb3b9dc0955a617",
    "brokerNameList": [
        "broker_1"
    ],
    "order": false
}
```

响应示例

响应成功示例

```
{
  "returnObj":{},
  "message":"success",
  "statusCode":800
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

订阅组管理

配置订阅组读取权限
接口功能介绍

配置订阅组读取权限

接口约束

无

URI

POST /v3/group/updatePerm

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名字		
brokerNameList	是	Array of Strings	brokerNameList		
enable	是	Boolean	是否开启		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象		
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/group/updatePerm

请求头header

无

请求体body

```
{
  "prodInstId":"mq_test",
  "groupName":"group",
  "brokerNameList":[
    "mq_test_broker_1"
  ],
  "enable":true
}
```

响应示例

响应成功示例

```
{
  "returnObj":{
  },
  "message":"success",
  "statusCode":800
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

查询订阅组一段时间内消费tps信息
接口功能介绍

查询订阅组一段时间内消费tps信息

接口约束

无

URI

GET /v3/group/outputTps

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topicName	是	String	topic名字		
brokerName	是	String	Broker名字		
groupName	是	String	订阅组名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	主题TPS统计数据		data
表 data				

参数	参数类型	说明	示例	下级对象
prodInstId	String	实例id	mq_test	

参数	参数类型	说明	示例	下级对象
topicName	String	主题名称	test	
groupName	String	消费组名称	group	
tpsVoList	Array of Objects	TPS统计列表		TpsVo
表 TpsVo				

参数	参数类型	说明	示例	下级对象
tps	Double	消息吞吐量（条/秒）	0.3333333333333333	
crtTime	Long	统计时间戳（毫秒）	1661937180392	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/group/outputTps?prodInstId=70687660456281088&topicName=test&brokerName

请求头header

无

请求体body

无

响应示例


```
{
  "returnObj":{
    "data":{
      "prodInstId":"mq_test",
      "topicName":"test",
      "groupName":"group",
      "tpsVoList":[
        {
          "tps":0,
          "crtTime":1661945340329
        },
        {
          "tps":0,
          "crtTime":1661945360495
        },
        {
          "tps":0,
          "crtTime":1661945380388
```

```
            }
          ]
        }
    },
    "message":"success",
    "statusCode":800
}
```


响应失败示例


```
{
    "returnObj": {},
    "message": "...",
    "error":"...",
    "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

查询订阅组的消息消费堆积
接口功能介绍

查询订阅组的消息消费堆积

接口约束

无

URI

GET /v3/consumer/accumulate

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名字		
topicName	是	String	Topic名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
total	Integer	总记录数。	1	
rows	Array of Objects	消费组消息统计信息列表。		rows
表 rows				

参数	参数类型	说明	示例	下级对象
diff	Integer	差值	0	
total	Integer	消息总数	27	
consumedTotal	Integer	已消费消息总数	27	
retryNums	Integer	重试次数	0	
groupName	String	消费组名称	group	
brokerName	String	Broker名称	mq_test_broker_1	
topicName	String	主题名称	test	
outTps	Integer	出消息TPS	0	
outTotalMsgToday	Integer	今日出消息总数	0	
offSetDiff	Integer	偏移量差值	0	
notAckMsgNum	Integer	未确认消息数	0	
notAckMsgTime	Integer	未确认消息时长	0	
notConsumeMsgTime	Integer	未消费消息时长	0	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/consumer/accumulate?prodInstId=70687660456281088&groupName=group&topic

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "returnObj":{
    "total":1,
    "rows":[
      {
        "diff":0,
        "total":27,
        "consumedTotal":27,
        "retryNums":0,
        "groupName":"group",
        "brokerName":"mq_test_broker_1",
        "topicName":"test",
        "outTps":0,
        "outTotalMsgToday":0,
        "offSetDiff":0,
        "notAckMsgNum":0,
        "notAckMsgTime":0,
        "notConsumeMsgTime":0
      }
    ]
  },
  "message":"success",
  "statusCode":800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

查询订阅组详细状态信息
接口功能介绍

查询订阅组详细状态信息

接口约束

无

URI

GET /v3/consumer/status

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名字		
clientId	是	String	客户端ID		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	消费组详情数据		data
表 data				

参数	参数类型	说明	示例	下级对象
properties	Object	消费组配置属性		properties
subscriptionSet	Array of Objects	订阅关系集合		subscriptionSet
mqTable	Object	消息队列信息表		mqTable
statusTable	Object	消费状态统计表		statusTable
jstack	String	线程堆栈信息	null	
表 properties				

参数	参数类型	说明	示例	下级对象
maxReconsumeTimes	String	最大重试次数	-1	
adjustTh readPoolNumsThreshold	String	线程池数量调整阈值	100000	
unitMode	String	单元模式开关	false	
consumeTimeoutInSec	String	消费超时时间（秒）	-1	
timeoutStrategy	String	超时处理策略	ALARM_AND_RETRY	
consumerGroup	String	消费组名称	group	
messageModel	String	消息模型	CLUSTERING	
allocate MessageQueueStrategy	String	消息队列分配策略	org.apache.rocketmq. client.consumer. rebalance.Allocate MessageQueueAveragely@ 14978a04	
pullThre sholdSizeForTopic	String	主题拉取大小阈值	-1	
suspendC urrentQueueTimeMillis	String	队列挂起时间（毫秒）	1000	
pullThre sholdSizeForQueue	String	队列拉取大小阈值	100	
PROP_CLIENT_VERSION	String	客户端版本	V2_8_3_CTG	
offsetStore	String	偏移量存储实现类	org.apache.rocketmq. client.consumer. store.RemoteBr okerOffsetStore@ 64668c91	
consumeC oncurrentlyMaxSpan	String	并发消费最大跨度	2000	

参数	参数类型	说明	示例	下级对象
postSubscriptionWhenPull	String	拉取时提交订阅关系开关	false	
consumeTimestamp	String	消费时间戳	20220831110532	
PROP_CONSUME_TYPE	String	消费类型	CONSUME_PASSIVELY	
consumeMessageBatchMaxSize	String	批量消费最大消息数	1	
defaultMQPushConsumerImpl	String	推送消费者实现类	org.apache.rocketmq.client.impl.consumer.DefaultMQPushConsumerImpl@6470d9c8	
PROP_THREADPOOL_CORE_SIZE	String	线程池核心大小	20	
pullInterval	String	拉取间隔（毫秒）	0	
pullThresholdForQueue	String	队列拉取阈值	1000	
pullThresholdForTopic	String	主题拉取阈值	-1	
consumeFromWhere	String	消费起始位置	CONSUME_FROM_FIRST_OFFSET	
PROP_NAMESERVER_ADDR	String	名称服务地址	192.168.71.188:8411;	
pullBatchSize	String	拉取批次大小	32	
consumeThreadMin	String	最小消费线程数	20	
PROP_CONSUMER_START_TIMESTAMP	String	消费者启动时间戳	1661916932150	
consumeThreadMax	String	最大消费线程数	64	
consumeTimeout2	String	消费超时时间2（秒）	15	
subscription	String	订阅关系	{}	
PROP_CONSUMEORDERLY	String	顺序消费开关	false	
messageListener	String	消息监听器实现类	com.ctg.mq.api.impl.MQConsumerImpl$ConsumerTopicListenerImpl@677658a0	
表 subscriptionSet				

参数	参数类型	说明	示例	下级对象
classFilterMode	Boolean	类过滤模式开关	false	

参数	参数类型	说明	示例	下级对象
topic	String	订阅主题	test	
subString	String	订阅表达式	*	
tagsSet	Array of Strings	标签集合	[]	
codeSet	Array of Strings	代码集合	[]	
subVersion	Long	订阅版本号	1661916951326	
expressionType	String	表达式类型	null	
filterClassSource	String	过滤类源码	null	
表 mqTable				

参数	参数类型	说明	示例	下级对象
commitOffset	Long	提交偏移量	6	
cachedMsgMinOffset	Long	缓存消息最小偏移量	0	
cachedMsgMaxOffset	Long	缓存消息最大偏移量	0	
cachedMsgCount	Integer	缓存消息数量	0	
cachedMsgSizeInMiB	Integer	缓存消息大小（MiB）	0	
transact ionMsgMinOffset	Long	事务消息最小偏移量	0	
transact ionMsgMaxOffset	Long	事务消息最大偏移量	0	
transactionMsgCount	Integer	事务消息数量	0	
locked	Boolean	是否锁定	false	
tryUnlockTimes	Integer	解锁尝试次数	0	
lastLockTimestamp	Long	最后锁定时间戳	1661916951284	
droped	Boolean	是否删除	false	
lastPullTimestamp	Long	最后拉取时间戳	1661929388688	
lastCons umeTimestamp	Long	最后消费时间戳	1661916951284	
表 statusTable				

参数	参数类型	说明	示例	下级对象
pullRT	Integer	拉取响应时间（毫秒）	0	
pullTPS	Integer	拉取TPS	0	
consumeRT	Integer	消费响应时间（毫秒）	0	
consumeOKTPS	Integer	消费成功TPS	0	

参数	参数类型	说明	示例	下级对象
consumeFailedTPS	Integer	消费失败TPS	0	
consumeFailedMsgs	Integer	消费失败消息数	0	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/consumer/status?prodInstId=70687660456281088&groupName=group&clientId=5.148.141@test_instance

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "returnObj":{
    "data":{
      "properties":{
        "maxReconsumeTimes":"-1",
        "adjustThreadPoolNumsThreshold":"100000",
        "unitMode":"false",
        "consumeTimeoutInSec":"-1",
        "timeoutStrategy":"ALARM_AND_RETRY",
        "consumerGroup":"group",
        "messageModel":"CLUSTERING",
        "allocateMessageQueueStrategy":"org.apache.rocketmq.client.consumer.rebalance.
AllocateMessageQueueAveragely@14978a04",
        "pullThresholdSizeForTopic":"-1",
        "suspendCurrentQueueTimeMillis":"1000",
        "pullThresholdSizeForQueue":"100",
        "PROP_CLIENT_VERSION":"V2_8_3_CTG",
        "offsetStore":"org.apache.rocketmq.client.consumer.store.RemoteBrokerOffsetStore@
64668c91",
        "consumeConcurrentlyMaxSpan":"2000",
        "postSubscriptionWhenPull":"false",
        "consumeTimestamp":"20220831110532",
        "PROP_CONSUME_TYPE":"CONSUME_PASSIVELY",
```

```
        "consumeMessageBatchMaxSize":"1",
        "defaultMQPushConsumerImpl":"org.apache.rocketmq.client.impl.consumer.DefaultM
QPushConsumerImpl@6470d9c8",
        "PROP_THREADPOOL_CORE_SIZE":"20",
        "pullInterval":"0",
        "pullThresholdForQueue":"1000",
        "pullThresholdForTopic":"-1",
        "consumeFromWhere":"CONSUME_FROM_FIRST_OFFSET",
        "PROP_NAMESERVER_ADDR":"192.168.71.188:8411;",
        "pullBatchSize":"32",
        "consumeThreadMin":"20",
        "PROP_CONSUMER_START_TIMESTAMP":"1661916932150",
        "consumeThreadMax":"64",
        "consumeTimeout2":"15",
        "subscription":"{}",
        "PROP_CONSUMEORDERLY":"false",
        "messageListener":"com.ctg.mq.api.impl.MQConsumerImpl$ConsumerTopicListenerImpl@
677658a0"
      },
      "subscriptionSet":[
        {
          "classFilterMode":false,
          "topic":"%RETRY%group",
          "subString":"*",
          "tagsSet":[
          ],
          "codeSet":[
          ],
          "subVersion":1661916951262,
          "expressionType":null,
          "filterClassSource":null
        },
        {
          "classFilterMode":false,
          "topic":"test",
          "subString":"*",
          "tagsSet":[
          ],
          "codeSet":[
          ],
          "subVersion":1661916951326,
          "expressionType":null,
          "filterClassSource":null
        }
      ],
      "mqTable":{
```

```
"MessageQueue [topic=%RETRY%group, brokerName=mq_test_broker_1, queueId=0]":{
  "commitOffset":0,
  "cachedMsgMinOffset":0,
  "cachedMsgMaxOffset":0,
  "cachedMsgCount":0,
  "cachedMsgSizeInMiB":0,
  "transactionMsgMinOffset":0,
  "transactionMsgMaxOffset":0,
  "transactionMsgCount":0,
  "locked":false,
  "tryUnlockTimes":0,
  "lastLockTimestamp":1661916951245,
  "droped":false,
  "lastPullTimestamp":1661929387746,
  "lastConsumeTimestamp":1661916951245
},
"MessageQueue [topic=test, brokerName=mq_test_broker_1, queueId=0]":{
  "commitOffset":6,
  "cachedMsgMinOffset":0,
  "cachedMsgMaxOffset":0,
  "cachedMsgCount":0,
  "cachedMsgSizeInMiB":0,
  "transactionMsgMinOffset":0,
  "transactionMsgMaxOffset":0,
  "transactionMsgCount":0,
  "locked":false,
  "tryUnlockTimes":0,
  "lastLockTimestamp":1661916951284,
  "droped":false,
  "lastPullTimestamp":1661929388688,
  "lastConsumeTimestamp":1661916951284
},
"MessageQueue [topic=test, brokerName=mq_test_broker_1, queueId=1]":{
  "commitOffset":6,
  "cachedMsgMinOffset":0,
  "cachedMsgMaxOffset":0,
  "cachedMsgCount":0,
  "cachedMsgSizeInMiB":0,
  "transactionMsgMinOffset":0,
  "transactionMsgMaxOffset":0,
  "transactionMsgCount":0,
  "locked":false,
  "tryUnlockTimes":0,
  "lastLockTimestamp":1661916951307,
  "droped":false,
  "lastPullTimestamp":1661929388580,
```

```
      "lastConsumeTimestamp":1661916951307
    },
    "MessageQueue [topic=test, brokerName=mq_test_broker_1, queueId=2]":{
      "commitOffset":7,
      "cachedMsgMinOffset":0,
      "cachedMsgMaxOffset":0,
      "cachedMsgCount":0,
      "cachedMsgSizeInMiB":0,
      "transactionMsgMinOffset":0,
      "transactionMsgMaxOffset":0,
      "transactionMsgCount":0,
      "locked":false,
      "tryUnlockTimes":0,
      "lastLockTimestamp":1661916951295,
      "droped":false,
      "lastPullTimestamp":1661929388691,
      "lastConsumeTimestamp":1661916951295
    },
    "MessageQueue [topic=test, brokerName=mq_test_broker_1, queueId=3]":{
      "commitOffset":8,
      "cachedMsgMinOffset":0,
      "cachedMsgMaxOffset":0,
      "cachedMsgCount":0,
      "cachedMsgSizeInMiB":0,
      "transactionMsgMinOffset":0,
      "transactionMsgMaxOffset":0,
      "transactionMsgCount":0,
      "locked":false,
      "tryUnlockTimes":0,
      "lastLockTimestamp":1661916951317,
      "droped":false,
      "lastPullTimestamp":1661929388688,
      "lastConsumeTimestamp":1661916951317
    }
  },
  "statusTable":{
    "%RETRY%group":{
      "pullRT":0,
      "pullTPS":0,
      "consumeRT":0,
      "consumeOKTPS":0,
      "consumeFailedTPS":0,
      "consumeFailedMsgs":0
    },
    "test":{
      "pullRT":0,
```

```
            "pullTPS":0,
            "consumeRT":0,
            "consumeOKTPS":0,
            "consumeFailedTPS":0,
            "consumeFailedMsgs":0
          }
        },
        "jstack":null
      }
    },
    "message":"success",
    "statusCode":800
}
```


响应失败示例


```
{
    "returnObj": {},
    "message": "...",
    "error":"201",
    "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

创建订阅组

接口功能介绍

创建消费组

接口约束

无

URI

POST /v3/group/create

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id	mq_test	
brokerNameList	是	Array of Strings	Broker名称列表	["broker_1"]	
subscrip tionGroupConfig	是	Object	订阅组配置信息		subscrip tionGroupConfig
表 subscrip tionGroupConfig					

参数	是否必填	参数类型	说明	示例	下级对象
consumeEnable	是	Boolean	是否允许消费	true	
firstCon sumeMechanism	是	String	首次消费机制	1	
groupName	是	String	订阅组名称	group2	
pullMechanism	是	String	拉取机制	1	

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功："800"，失败："900"。	800	
message	String	描述状态。		
returnObj	Object	返回对象。		
error	String	错误码，描述错误信息。		

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/group/create

请求头header

无

请求体body

```
{
    "subscriptionGroupConfig": {
        "firstConsumeMechanism": 1,
        "groupName": "groupTest",
        "consumeEnable": true,
```

```
        "pullMechanism": 1
    },
    "prodInstId": "d7ed2dbad63843f4bcb3b9dc0955a617",
    "brokerNameList": [
        "broker_1"
    ]
}
```

响应示例

响应成功示例

```
{
    "message": "success",
    "returnObj": {},
    "statusCode": 800
}
```

响应失败示例

```
{
    "error": "ROCKETMQ_2002",
    "message": "Subscription Group[groupTest] is existed in broker[broker_1]",
    "returnObj": {},
    "statusCode": 900
}
```

状态码

请参考 状态码

错误码

请参考 错误码

查看订阅组订阅信息

接口功能介绍

查看订阅组订阅信息

接口约束

无

URI

GET /v3/group/subDetail

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	消费组名		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败："900"	800	
message	String	描述状态	success	
returnObj	Object	返回对象		returnObj
error	String	错误码，描述错误信息	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	订阅相关数据对象		data
表 data				

参数	参数类型	说明	示例	下级对象
prodInstId	String	实例id	d7ed2dba d63843f4bcb3b9dc0955a617	
groupName	String	消费组名称	groupTest	
subscriptionDataMap	Object	订阅数据映射表（没有消费组实例在线时为空）		

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/group/subDetail?prodInstId=70687660456281088&groupName=test

请求头header

无

请求体body

1

响应示例

响应成功示例


```
{
    "message": "success",
    "returnObj": {
        "data": {
            "prodInstId": "d7ed2dbad63843f4bcb3b9dc0955a617",
            "groupName": "groupTest",
            "subscriptionDataMap": {}
        }
    },
    "statusCode": 800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

查询主题可重置的时间范围

接口功能介绍

查询主题可重置的时间范围

接口约束

无

URI

GET /v3/consumer/timeSpan

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topicName	是	String	Topic名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	响应数据详情		data
表 data				

参数	参数类型	说明	示例	下级对象
tenantId	String	租户ID	2147483647	
clusterName	String	集群名称	mq_test	
topicName	String	主题名称	test	
minTimeStamp	Long	最小时间戳 单位:毫秒	1660548910643	
maxTimeStamp	Long	最大时间戳 单位:毫秒	1661857434591	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/consumer/timeSpan?prodInstId=70687660456281088&topicName=test

请求头header

无

请求体body

无

响应示例

响应成功示例

```
{
  "returnObj":{
    "data":{
      "tenantId":"2147483647",
      "clusterName":"mq_test",
      "topicName":"test",
      "minTimeStamp":1660548910643,
      "maxTimeStamp":1661857434591
    }
  },
  "message":"success",
  "statusCode":800
}
```

响应失败示例

```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

删除订阅组

接口功能介绍

删除消费组

接口约束

无

URI

POST /v3/group/delete

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id	d7ed2dbad63843f4bcb3b9dc0955a617	
groupName	是	String	消费组名称	groupTest	

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功："800"，失败："900"。	800	
message	String	描述状态。		
returnObj	Object	返回对象		
error	String	错误码，描述错误信息。		

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/group/delete

请求头header

无

请求体body

```
{
    "groupName": "groupTest",
    "prodInstId": "d7ed2dbad63843f4bcb3b9dc0955a617"
}
```

响应示例

响应成功示例

```
{
  "returnObj":{},
  "message":"success",
  "statusCode":800
```

```
}
```

响应失败示例

```
{
   "returnObj": {},
   "message": "...",
   "error":"201",
   "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

重置订阅组消费进度到指定时间戳

接口功能介绍

重置订阅组消费进度到指定时间戳

接口约束

无

URI

POST /v3/consumer/resetOffset

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名字		
topicName	是	String	Topic名字		
resetTime	是	Long	要重置的毫秒时间戳		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	String	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
total	Integer	总记录数	4	
diffOffset	Integer	偏移量差值	5	
rows	Array of Objects	队列偏移量详情列表		rows
表 rows				

参数	参数类型	说明	示例	下级对象
brokerName	String	Broker名称	mq_test_broker_1	
queueId	Integer	队列ID	0	
brokerOffset	Integer	Broker端偏移量	6	
consumerOffset	Integer	消费者端偏移量	6	
timestampOffset	Integer	时间戳偏移量	5	
rollbackOffset	Integer	回滚偏移量	5	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/consumer/resetOffset

请求头header

无

请求体body

```
{
  "prodInstId":"mq_test",
```

```
  "groupName":"group",
  "topicName":"test",
  "resetTime":1661843664000
}
```

响应示例

响应成功示例

```
{
  "returnObj":{
    "total":4,
    "diffOffset":5,
    "rows":[
      {
        "brokerName":"mq_test_broker_1",
        "queueId":0,
        "brokerOffset":6,
        "consumerOffset":6,
        "timestampOffset":5,
        "rollbackOffset":5
      },
      {
        "brokerName":"mq_test_broker_1",
        "queueId":1,
        "brokerOffset":6,
        "consumerOffset":6,
        "timestampOffset":5,
        "rollbackOffset":5
      },
      {
        "brokerName":"mq_test_broker_1",
        "queueId":2,
        "brokerOffset":7,
        "consumerOffset":7,
        "timestampOffset":6,
        "rollbackOffset":6
      },
      {
        "brokerName":"mq_test_broker_1",
        "queueId":3,
        "brokerOffset":8,
        "consumerOffset":8,
        "timestampOffset":6,
        "rollbackOffset":6
      }
    ]
  },
```

```
  "message":"success",
  "statusCode":800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

查询订阅组列表

接口功能介绍

查询订阅组列表

接口约束

无

URI

GET /v3/group/list

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		

参数	参数类型	说明	示例	下级对象
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
rows	Array of Objects	消费组信息列表		rows
total	Integer	总记录数。	1	
表 rows				

参数	参数类型	说明	示例	下级对象
groupName	String	消费组名称	Group0317	
consumeEnable	Boolean	是否允许消费	true	
consumeFromMinEnable	Boolean	是否从最小位点消费	true	
consumeBroadcastEnable	Boolean	是否允许广播消费	false	
retryQueueNums	Integer	重试队列数量	1	
retryMaxTimes	Integer	最大重试次数	16	
brokerId	Integer	Broker ID	0	
whichBrokerWhenConsumeSlowly	Integer	消费慢时指定的Broker ID	1	
notifyConsumerIdsChangedEnable	Boolean	是否开启消费者ID变更通知	true	
pullMechanism	Integer	拉取机制	1	
firstConsumeMechanism	Integer	首次消费机制	1	
subscribeMap	Object	订阅关系		
remark	String	备注		
brokerName	String	Broker名称	broker_1	
type	Integer	类型	0	
allowdAllTopics	Boolean	是否允许所有主题	false	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/group/list?prodInstId=70687660456281088

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "message": "success",
  "returnObj": {
    "rows": [{
      "groupName": "Group0317",
      "consumeEnable": true,
      "consumeFromMinEnable": true,
      "consumeBroadcastEnable": false,
      "retryQueueNums": 1,
      "retryMaxTimes": 16,
      "brokerId": 0,
      "whichBrokerWhenConsumeSlowly": 1,
      "notifyConsumerIdsChangedEnable": true,
      "pullMechanism": 1,
      "firstConsumeMechanism": 1,
      "subscribeMap": {},
      "remark": null,
      "namesrvAddr": null,
      "clusterName": null,
      "brokerName": "broker_1",
      "type": 0,
      "allowdAllTopics": false
    }],
    "total": 1
  },
  "statusCode": 800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
```

"statusCode": "900"
}

状态码

请参考 状态码

错误码

请参考 错误码

查询订阅组当前客户端的连接情况
接口功能介绍

查询订阅组当前客户端的连接情况

接口约束

无

URI

GET /v3/consumer/connection

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	响应数据详情		data
表 data				

参数	参数类型	说明	示例	下级对象
group	Object	消费组详情		group
表 group				

参数	参数类型	说明	示例	下级对象
connectionSet	Array of Objects	客户端连接集合		connectionSet
subscriptionTable	Object	订阅关系表		subscriptionTable
consumeType	String	消费类型	CONSUME_PASSIVELY	
messageModel	String	消费方式 CLUSTERING-集群消费;BROADCASTING-广播消费	CLUSTERING	
consumeFromWhere	String	消费起始位置	CONSUME_FROM_FIRST_OFFSET	
表 connectionSet				

参数	参数类型	说明	示例	下级对象
clientId	String	客户端ID	5.5.148.141@test_instance	
clientAddr	String	客户端地址	192.168.71.1:52369	
language	String	客户端开发语言	JAVA	
version	Integer	客户端版本	297	
表 subscriptionTable				

参数	参数类型	说明	示例	下级对象
classFilterMode	Boolean	类过滤模式开关	false	
topic	String	订阅主题	test	
subString	String	订阅表达式	*	
tagsSet	Array of Strings	标签集合	[]	
codeSet	Array of Strings	代码集合	[]	
subVersion	Long	订阅版本号	1661916951326	
expressionType	String	表达式类型	null	
filterClassSource	String	过滤类源码	null	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/consumer/connection?prodInstId=70687660456281088&groupName=group

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "returnObj":{
    "data":{
      "group":{
        "connectionSet":[
          {
            "clientId":"5.5.148.141@test_instance",
            "clientAddr":"192.168.71.1:52369",
            "language":"JAVA",
            "version":297
          }
        ],
        "subscriptionTable":{
          "%RETRY%group":{
            "classFilterMode":false,
            "topic":"%RETRY%group",
            "subString":"*",
            "tagsSet":[
            ],
            "codeSet":[
            ],
            "subVersion":1661916951262,
            "expressionType":null,
            "filterClassSource":null
          },
          "test":{
            "classFilterMode":false,
            "topic":"test",
            "subString":"*",
```

```
                "tagsSet":[
                ],
                "codeSet":[
                ],
                "subVersion":1661916951326,
                "expressionType":null,
                "filterClassSource":null
            }
        },
        "consumeType":"CONSUME_PASSIVELY",
        "messageModel":"CLUSTERING",
        "consumeFromWhere":"CONSUME_FROM_FIRST_OFFSET"
      }
    }
  },
  "message":"success",
  "statusCode":800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

死信队列管理

通过传入MessageID查询指定的死信消息

接口功能介绍

通过传入MessageID查询指定的死信消息

接口约束

无

URI

GET /v3/dlq/getById

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名字		
msgId	是	String	消息ID		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题		
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	返回具体数据		data
表 data				

参数	参数类型	说明	示例	下级对象
queueId	Integer	队列ID	0	
storeSize	Integer	存储大小,单位B	280	
queueOffset	Integer	队列偏移量	0	
sysFlag	Integer	系统标记	0	
bornTimestamp	Long	消息生成时间戳	1661947215767	
bornHost	String	消息生成主机地址	192.168.71.1:64222	
storeTimestamp	Long	消息存储时间戳	1661947567107	
storeHost	String	消息存储主机地址	192.168.71.188:8422	

参数	参数类型	说明	示例	下级对象
msgId	String	消息ID	C0A847BC 000020E600000000000794D8	
commitLogOffset	Long	提交日志偏移量	496856	
bodyCRC	Integer	消息体CRC校验值	1335308324	
reconsumeTimes	Integer	重试次数	4	
prepared TransactionOffset	Long	预处理事务偏移量	0	
topic	String	主题名称	%DLQ%group1	
properties	Object	消息属性		properties
messageBody	String	消息体内容	GZ2BfSrWuAinKs	
表 properties				

参数	参数类型	说明	示例	下级对象
REAL_TOPIC	String	真实主题名称	%DLQ%group1	
ORIGIN_MESSAGE_ID	String	原始消息ID	C0A847BC 000020E600000000000109F2	
RETRY_TOPIC	String	重试主题名称	test1	
UNIQ_KEY	String	唯一标识键	0505948D 1C9C18B4AAC29EC9A7950000	
WAIT	String	等待标记	false	
DELAY	String	延迟级别	5	
REAL_QID	String	真实队列ID	0	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/dlq/getById?prodInstId=70687660456281088&groupName=group&msgId=C0A847B
000020E600000000000794D8

请求头header

无

请求体body

1

响应示例

响应成功示例


```
{
  "returnObj":{
    "data":{
      "queueId":0,
      "storeSize":280,
      "queueOffset":0,
      "sysFlag":0,
      "bornTimestamp":1661947215767,
      "bornHost":"192.168.71.1:64222",
      "storeTimestamp":1661947567107,
      "storeHost":"192.168.71.188:8422",
      "msgId":"C0A847BC000020E600000000000794D8",
      "commitLogOffset":496856,
      "bodyCRC":1335308324,
      "reconsumeTimes":4,
      "preparedTransactionOffset":0,
      "topic":"%DLQ%group1",
      "properties":{
        "REAL_TOPIC":"%DLQ%group1",
        "ORIGIN_MESSAGE_ID":"C0A847BC000020E600000000000109F2",
        "RETRY_TOPIC":"test1",
        "UNIQ_KEY":"0505948D1C9C18B4AAC29EC9A7950000",
        "WAIT":"false",
        "DELAY":"5",
        "REAL_QID":"0"
      },
      "messageBody":"GZ2BfSrWuAinKs"
    }
  },
  "message":"success",
  "statusCode":800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

查询指定时间段内订阅组内存在的死信消息
接口功能介绍

查询指定时间段内订阅组内存在的死信消息

接口约束

无

URI

GET /v3/dlq/getByTime

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名称		
beginTime	是	Long	开始时间的毫秒时间戳		
endTime	是	Long	结束时间的毫秒时间戳		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
total	Integer	总记录数。	1	
rows	Array of Objects	消息详情列表。		rows
表 rows				

参数	参数类型	说明	示例	下级对象
queueId	Integer	队列ID。	0	
storeSize	Integer	存储大小，单位B	280	
queueOffset	Integer	队列偏移量。	0	
sysFlag	Integer	系统标记。	0	
bornTimestamp	Long	消息生成时间戳。	1661947215767	
bornHost	String	消息生成主机地址。	192.168.71.1:64222	
storeTimestamp	Long	消息存储时间戳。	1661947567107	
storeHost	String	消息存储主机地址。	192.168.71.188:8422	
msgId	String	消息ID。	C0A847BC 000020E600000000000794D8	
commitLogOffset	Long	提交日志偏移量。	496856	
bodyCRC	Integer	消息体CRC校验值。	1335308324	
reconsumeTimes	Integer	重试次数。	4	
prepared TransactionOffset	Long	预处理事务偏移量。	0	
topic	String	主题名称。	%DLQ%group1	
properties	Object	消息属性。		properties
messageBody	String	消息体内容。	GZ2BfSrWuAinKs	
表 properties				

参数	参数类型	说明	示例	下级对象
MIN_OFFSET	String	最小偏移量。	0	
REAL_TOPIC	String	真实主题名称。	%DLQ%group1	
ORIGIN_MESSAGE_ID	String	原始消息ID。	C0A847BC 000020E600000000000109F2	
RETRY_TOPIC	String	重试主题名称。	test1	
MAX_OFFSET	String	最大偏移量。	1	
UNIQ_KEY	String	唯一标识键。	0505948D 1C9C18B4AAC29EC9A7950000	
WAIT	String	等待标记。	false	

参数	参数类型	说明	示例	下级对象
DELAY	String	延迟级别。	5	
REAL_QID	String	真实队列ID。	0	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/dlq/getByTime?prodInstId=70687660456281088&groupName=group&beginTime=1

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "returnObj":{
    "total":1,
    "rows":[
      {
        "queueId":0,
        "storeSize":280,
        "queueOffset":0,
        "sysFlag":0,
        "bornTimestamp":1661947215767,
        "bornHost":"192.168.71.1:64222",
        "storeTimestamp":1661947567107,
        "storeHost":"192.168.71.188:8422",
        "msgId":"C0A847BC000020E600000000000794D8",
        "commitLogOffset":496856,
        "bodyCRC":1335308324,
        "reconsumeTimes":4,
        "preparedTransactionOffset":0,
        "topic":"%DLQ%group1",
        "properties":{
          "MIN_OFFSET":"0",
          "REAL_TOPIC":"%DLQ%group1",
          "ORIGIN_MESSAGE_ID":"C0A847BC000020E600000000000109F2",
          "RETRY_TOPIC":"test1",
```

```
            "MAX_OFFSET":"1",
            "UNIQ_KEY":"0505948D1C9C18B4AAC29EC9A7950000",
            "WAIT":"false",
            "DELAY":"5",
            "REAL_QID":"0"
          },
          "messageBody":"GZ2BfSrWuAinKs"
        }
      ]
    },
    "message":"success",
    "statusCode":800
}
```


响应失败示例


```
{
    "returnObj": {},
    "message": "...",
     “error” :"201",
    "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

重发死信队列

接口功能介绍

重发死信队列

接口约束

无

URI

POST /v3/dlq/resend

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
groupId	是	String	订阅组名称		
msgId	是	String	消息ID		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/dlq/resend

请求头header

无

请求体body

```
{
"prodInstId":"mq_test",
"groupId": "T1",
   "msgId": "xxx"
}
```

响应示例

响应成功示例

```
{
   "returnObj":{
   },
   "message":"success",
   "statusCode":800
}
```

响应失败示例

```
{
   "returnObj": {},
   "message": "...",
   "error":"201",
   "statusCode": "900"
}
```

状态码

请参考 状态码

错误码

请参考 错误码

消息管理

查看消息消费结果

接口功能介绍

查看消息消费结果

接口约束

无

URI

GET /v3/message/trace

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
msgId	是	String	消息ID		
prodInstId	是	String	实例ID		
groupName	是	String	订阅组名字		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		

参数	参数类型	说明	示例	下级对象
message	String	描述状态		
returnObj	Object	返回对象		data
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 data				

参数	参数类型	说明	示例	下级对象
msgId	String	消息ID	C0A847BC000020E600000000000003FEE	
clusterName	String	集群名称	mq_test	
consumeStatusList	Array of Objects	消费状态列表		consumeStatusList
表 consumeStatusList				

参数	参数类型	说明	示例	下级对象
object1	String	消费组名称	group	
object2	String	消费状态	TO_CONSUME	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/message/trace?msgId= C0A847BC000020E600000000000045FE&prodInstId=70687660456281088&groupName=group

请求头header

无

请求体body

无

响应示例

响应成功示例

```
{
  "returnObj":{
    "data":{
      "msgId":"C0A847BC000020E600000000000003FEE",
      "clusterName":"mq_test",
```

```
        "consumeStatusList":[
          {
            "object1":"group",
            "object2":"TO_CONSUME"
          }
        ]
      }
    },
    "message":"success",
    "statusCode":800
}
```


响应失败示例


```
{
    "returnObj": {},
    "message": "...",
    "error":"201",
    "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

根据MessageKey查询消息

接口功能介绍

根据MessageKey查询消息

接口约束

无

URI

GET /v3/message/queryByKey

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
topicName	是	String	主题名字		
key	是	String	用于查询的消息KEY值		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
total	Integer	总记录数。	1	
rows	Array of Objects	消息详情列表。		rows
表 rows				

参数	参数类型	说明	示例	下级对象
queueId	Integer	队列ID。	3	
storeSize	Integer	存储大小。	485	
queueOffset	Integer	队列偏移量。	7	
sysFlag	Integer	系统标记。	0	
bornTimestamp	Long	消息生成时间戳。	1661857434591	
bornHost	String	消息生成主机地址。	192.168.71.1:55492	
storeTimestamp	Long	消息存储时间戳。	1661857434599	
storeHost	String	消息存储主机地址。	192.168.71.188:8422	
msgId	String	消息ID。	C0A847BC 000020E600000000000045FE	
commitLogOffset	Long	提交日志偏移量。	17918	
bodyCRC	Integer	消息体CRC校验值。	979262990	
reconsumeTimes	Integer	重试次数。	0	
prepared TransactionOffset	Long	预处理事务偏移量。	0	

参数	参数类型	说明	示例	下级对象
topic	String	主题名称。	test	
properties	Map of String	消息属性。		properties
messageBody	String	消息体内容。		
status	String	消息状态。	null	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/message/queryByKey?key=test_key&prodInstId=70687660456281088&groupName

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "returnObj":{
    "total":1,
    "rows":[
      {
        "queueId":3,
        "storeSize":485,
        "queueOffset":7,
        "sysFlag":0,
        "bornTimestamp":1661857434591,
        "bornHost":"192.168.71.1:55492",
        "storeTimestamp":1661857434599,
        "storeHost":"192.168.71.188:8422",
        "msgId":"C0A847BC000020E600000000000045FE",
        "commitLogOffset":17918,
        "bodyCRC":979262990,
        "reconsumeTimes":0,
        "preparedTransactionOffset":0,
        "topic":"test",
        "properties":{
          "KEYS":"test_key",
```

```
            "UNIQ_KEY":"050562443C7018B4AAC2996FB3DE0000"
          },
          "messageBody":"D8SQN8Vl
Dr1ZD4r4KVtse9Zczg5A9FA2nT7vu7LwLZ2w6oQnnXSmARpRJgHA8VJ6akEtHcm470kilKhE4pGgTKoXf5fF0kpzLc3ek01da
          "status":null
        }
      ]
    },
    "message":"success",
    "statusCode":800
}
```


响应失败示例


```
{
    "returnObj": {},
    "message": "...",
    "error":"201",
    "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

向指定的消费者推送消息

接口功能介绍

向指定的消费者推送消息

接口约束

无

URI

POST /v3/message/push

路径参数 无

Query参数 无

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		

参数	是否必填	参数类型	说明	示例	下级对象
msgId	是	String	消息的offsetMsgId		
groupName	是	String	订阅组名字		
clientId	是	String	客户端ID		

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900	800	
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	

枚举参数

无

请求示例

请求url

无

请求头header

无

请求体body

```
{
  "prodInstId":"mq_test",
  "msgId":"C0A847BC000020E60000000000003FEE",
  "groupName":"group",
  "clientId":"5.5.148.141@test_instance"
}
```

响应示例

响应成功示例

```
{
  "returnObj":{

  },
```

```
  "message":"success",
  "statusCode":800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

查询指定时间段内Topic的消息

接口功能介绍

查询指定时间段内Topic的消息

接口约束

无

URI

GET /v3/message/queryByTime

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例id		
topicName	是	String	主题名字		
beginTime	是	Long	开始时间的毫秒时间戳		
endTime	是	Long	结束时间的毫秒时间戳		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	200000002368	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	

枚举参数

无

请求示例

请求url

https://
[endpoint].ctapi.ctyun.cn/v3/message/queryByTime? topicName=test_key&prodInstId=70687660456281088&b

请求头header

无

请求体body

、

响应示例

响应成功示例


```
{
  "returnObj":{
    "total":1,
    "rows":[
      {
        "queueId":0,
        "storeSize":967,
        "queueOffset":4,
        "sysFlag":0,
        "bornTimestamp":1660564149558,
        "bornHost":"192.168.71.1:60905",
        "storeTimestamp":1660564149566,
        "storeHost":"192.168.71.188:8422",
        "msgId":"C0A847BC000020E60000000000003C27",
        "commitLogOffset":15399,
```

```
            "bodyCRC":1729222146,
            "reconsumeTimes":0,
            "preparedTransactionOffset":0,
            "topic":"test",
            "properties":{
               "MIN_OFFSET":"0",
               "MAX_OFFSET":"6",
               "UNIQ_KEY":"0505CD4343C018B4AAC24C59BD360002"
            },
            "messageBody":"mmfyJPna
5ArnW0KcqP5QQErMYmfdYMVSn699eWZqLFk07AjzBN2eTK0cEtcicngBvKMegX760wddTRqkd0xVMgLLNdqTG41UwLKJt1a4N
            "status":null
         }
      ]
   },
   "message":"success",
   "statusCode":800
}
```


响应失败示例


```
{
   "returnObj": {},
   "message": "...",
   "error":"201",
   "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

通过Msgld查询消息

接口功能介绍

根据msgId查询消息详情

接口约束

无

URI

GET /v3/message/queryByMsgId

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
msgId	是	String	消息ID		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败："900"	800	
message	String	描述状态	success	
returnObj	Object	返回对象		returnObj
error	String	错误码，描述错误信息	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	消息详情数据对象		data
表 data				

参数	参数类型	说明	示例	下级对象
brokerName	String	Broker名称	broker_1	
queueId	Integer	队列ID	0	
storeSize	Integer	存储大小	1240	
queueOffset	Integer	队列偏移量	0	
sysFlag	Integer	系统标识	0	
bornTimestamp	Long	消息生成时间戳	1763433803612	
bornHost	String	消息生成主机地址及端口	192.168.1.138:63894	
storeTimestamp	Long	消息存储时间戳	1763433803618	
storeHost	String	消息存储主机地址及端口	192.168.1.140:9600	
msgId	String	消息唯一标识	C0A8018C00002580000000257833D8FF	
commitLogOffset	Long	提交日志偏移量	160930453759	
bodyCRC	Integer	消息体CRC校验值	183478142	

参数	参数类型	说明	示例	下级对象
reconsumeTimes	Integer	重新消费次数	0	
prepared TransactionOffset	Integer	预处理事务偏移量	0	
topic	String	消息主题	topicTest	
properties	Object	消息属性集合		properties
messageBody	String	消息体内容	WebTestTools	
status	String	消息状态	TO_CONSUME	
messageBodyPath	String	消息体路径	null	
bodySize	Integer	消息体大小	1024	
表 properties				

参数	参数类型	说明	示例	下级对象
KEYS	String	keys	webtest	
UNIQ_KEY	String	消息唯一键	21000DF6 1BD505EF04B559D9275C0000	
CLUSTER	String	集群ID	d7ed2dba d63843f4bcb3b9dc0955a617	
TAGS	String	消息标签	1763433803606_0	
WAIT	String	是否等待标识	true	
DELAY	String	延迟时间	0	
BUYER_ID	String	买家ID	null	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/message/queryByMsgId?prodInstId=70687660456281088&msgId=xxx

请求头header

无

请求体body

无

响应示例

响应成功示例

```
{
    "message": "success",
    "returnObj": {
        "data": {
            "brokerName": "broker_1",
            "queueId": 0,
            "storeSize": 1240,
            "queueOffset": 0,
            "sysFlag": 0,
            "bornTimestamp": 1763433803612,
            "bornHost": "192.168.1.138:63894",
            "storeTimestamp": 1763433803618,
            "storeHost": "192.168.1.140:9600",
            "msgId": "C0A8018C00002580000000257833D8FF",
            "commitLogOffset": 160930453759,
            "bodyCRC": 183478142,
            "reconsumeTimes": 0,
            "preparedTransactionOffset": 0,
            "topic": "topicTest",
            "properties": {
                "KEYS": "webtest",
                "UNIQ_KEY": "21000DF61BD505EF04B559D9275C0000",
                "CLUSTER": "d7ed2dbad63843f4bcb3b9dc0955a617",
                "TAGS": "1763433803606_0",
                "WAIT": "true",
                "DELAY": "0",
                "BUYER_ID": null
            },
            "messageBody": "WebTestTools",
            "status": "TO_CONSUME",
            "messageBodyPath": null,
            "bodySize": 1024
        }
    },
    "statusCode": 800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

根据msgId查询消息消息轨迹
接口功能介绍

根据msgId查询消息消息轨迹

接口约束

无

URI

GET /v3/message/getTrace

路径参数 无

Query参数

参数	是否必填	参数类型	说明	示例	下级对象
prodInstId	是	String	实例ID		
msgId	是	String	消息ID		

请求参数

请求头header参数

参数	是否必填	参数类型	说明	示例	下级对象
regionId	是	String	资源池编码	100054c0416811e9a6690242ac110002	

请求体body参数 无

响应参数

参数	参数类型	说明	示例	下级对象
statusCode	String	接口系统层面状态码。成功：800，失败：900		
message	String	描述状态		
returnObj	Object	返回对象。此参数所包含的参数请见“响应示例'里面的注释		returnObj
error	String	错误码，只有非成功才有这个字段，方便快速定位问题	201	
表 returnObj				

参数	参数类型	说明	示例	下级对象
data	Object	消息生产消费轨迹数据对象		data

参数	参数类型	说明	示例	下级对象
表 data				

参数	参数类型	说明	示例	下级对象
topic	String	消息主题名	topicTest	
tags	String	标签		
keys	String	消息索引值		
groupName	String	生产者名		
msgType	String	消息类型		
msgId	String	消息id		
costTime	String	耗时		
status	String	发送状态	SEND_SUCCESS	
pubTime	String	发送时间戳		
arrivedServerTime	String	消息到达服务端时间戳		
clientHost	String	客户端主机地址		
subTraceList	Array of Objects	消息堆积列表		subTrace
表 subTrace				

参数	参数类型	说明	示例	下级对象
groupName	String	消费组名		
traceLogList	Array of Objects	消费轨迹日志列表		traceLog
表 traceLog				

参数	参数类型	说明	示例	下级对象
msgId	String	消息id		
clientHost	String	客户端主机地址		
costTime	String	消费耗时（毫秒）		
subTime	String	消费时间戳		
status	String	消费状态 CONSUME_SUCCESS-消费成功;RECONSUME_LATER-稍后重试	CONSUME_SUCCESS	

枚举参数

无

请求示例

请求url

https://[endpoint].ctapi.ctyun.cn/v3/message/getTrace?prodInstId=70687660456281088&msgId=xxx

请求头header

无

请求体body

无

响应示例

响应成功示例


```
{
  "returnObj":{
    {
  "message": "success",
  "data": {
        "topic": "TopicTest1234",
        "keys": "OrderID188",
        "tags": "TagA",
        "groupName": "ProducerGroupName",
        "msgType": "NORMAL",
        "msgId": "C0A8016400002AA00000000000000000",
        "costTime": 51,
        "status": "SEND_SUCCESS",
        "pubTime": 1693231023742,
        "arrivedServerTime": 0,
        "clientHost": "192.168.1.100",
        "subTraceList": [
          {
            "groupName": "CID_JODIE_1",
            "traceLogList": [
              {
                "msgId": "C0A8016400002AA00000000000000000",
                "clientHost": "192.168.1.100",
                "costTime": 4,
                "subTime": 1693232037955,
                "status": "CONSUME_SUCCESS"
              }
            ]
          }
        ]
    }
  "statusCode": 800
}
```

```
  },
  "message":"success",
  "statusCode":800
}
```


响应失败示例


```
{
  "returnObj": {},
  "message": "...",
  "error":"201",
  "statusCode": "900"
}
```


状态码

请参考 状态码

错误码

请参考 错误码

SDK概述

本文介绍了分布式消息服务RocketMQ版提供的SDK语言版本。

SDK列表

下表提供了分布式消息服务RocketMQ版支持的SDK列表。

编程语言	参考文档
Java	开发指南-Java
Python	开发指南-Python
Go	开发指南-Go
C++	SDK参考-C++
.NET	SDK参考-.NET
PHP	SDK参考-PHP

RocketMQ C++ SDK

概述

本文介绍使用 RocketMQ C++（4.x） 客户端 SDK，访问分布式消息服务RocketMQ，帮助您更好地理解消息收发的完整过程。

前置条件

安装gcc-c++ 4.8.2 及以上版本，需支持C++11。

安装cmake 2.8.0及以上版本。

安装automake 1.11.1及以上版本。

安装autoconf 2.65及以上版本。

安装libtool 2.2.6 及以上版本。

环境准备

需要在客户端环境安装 RocketMQ-Client-CPP 库，根据官方文档进行安装即可： 安装 CPP 动态库，推荐使用 master 分支构建。

在项目中引入 RocketMQ-Client-CPP 相关头文件及动态库，详见实例代码头文件。

使用g++命令获得可执行文件，如：

```
g++ -o xxxProducer xxxProducer.cpp -lrocketmq -lpthread -lz -ldl -lrt
```

使用 C++ SDK 收发普通消息

发送普通消息

```
#include <iostream>
#include <chrono>
#include <thread>
#include <string>
#include "rocketmq/DefaultMQProducer.h"
using namespace std;
using namespace rocketmq;

int main(){
  DefaultMQProducer producer("group_name");
  //填写分布式消息服务RocketMQ版的接入点
  producer.setNamesrvAddr("your access point");
  //填写分布式消息服务RocketMQ版的ak、sk
  producer.setSessionCredentials("ak", "sk", "channel");
  producer.start();
  int count = 64;
  for (int i = 0; i < count; ++i)
  {
    //填入主题名、tag名、消息body
    MQMessage msg("topic_name", "TAG", "msg content");
    try
    {
      SendResult sendResult = producer.send(msg);
      std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID: "
 << sendResult.getMsgId() << std::endl;
      this_thread::sleep_for(chrono::seconds(1));
    }
    catch (MQException e)
    {
      std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() << std::endl;
    }
  }
  std::cout << "Send " << count << " messages OK, costs" << std::endl;
  producer.shutdown();
  return 0;
}
```

收取普通消息

```
#include <iostream>
#include <thread>
#include "rocketmq/DefaultMQPushConsumer.h"
using namespace rocketmq;
```

```
class ConcurrentMessageListener : public MessageListenerConcurrently
{
public:
  ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs)
  {
    for (auto item = msgs.begin(); item != msgs.end(); item++)
    {
      std::cout << "Received Message Topic:" << item->getTopic() << ", MsgId:"
 << item->getMsgId() << std::endl;
    }
    return CONSUME_SUCCESS;
  }
};

int main(int argc, char *argv[]){

  DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer("consumer_group");
  consumer->setNamesrvAddr("your access point");
  consumer->setSessionCredentials("ak", "sk");
  ConcurrentMessageListener *messageListener = new ConcurrentMessageListener();
  consumer->subscribe("topic_name", "tag");
  consumer->registerMessageListener(messageListener);
  consumer->start();
  std::this_thread::sleep_for(std::chrono::seconds(60));
  consumer->shutdown();
  return 0;
}
```

使用C++客户端收发顺序消息

简介

顺序消息分为两类，全局顺序消息和分区顺序消息，通过队列数区分。

全局顺序： 对于指定的一个 Topic，所有消息的生产和消费需要遵循一定的顺序，消息的消费顺序必须和生产顺序一致，即需要严格的先入先出 FIFO的顺序进行发布和消费。

分区顺序： 对于指定的一个 Topic，其中每一个分区的消息生产与消费是有序的，同一个队列内的消息按照严格的 FIFO 顺序进行发布和订阅。消息投递到哪一个分区由消息的 Sharding Key 来进行区分。在 SDK 中可以通过指定 Sharding Key 和回调函数来控制消息投递到哪个分区。

发送顺序消息

```
#include <iostream>
#include <chrono>
#include <thread>
#include "rocketmq/DefaultMQProducer.h"
using namespace std;
```

```
using namespace rocketmq;

class DefineSelectMessageQueue : public MessageQueueSelector
{
public:
  MQMessageQueue select(const std::vector<MQMessageQueue> &mqs, const MQMessage &msg, void
 *arg)
  {
    //若希望全局有序，请修改对应index
    int orderId = *static_cast<int *>(arg);
    int index = orderId % mqs.size();
    return mqs[index];
  }
};

int main(){
  DefaultMQProducer producer("group_name");
  //填写分布式消息服务RocketMQ版的接入点
  producer.setNamesrvAddr("your access point");
  //填写分布式消息服务RocketMQ版的ak、sk
  producer.setSessionCredentials("ak", "sk", "channel");
  producer.start();

  DefineSelectMessageQueue *queueSelector = new DefineSelectMessageQueue();
  int count = 64;
  for (int i = 0; i < count; ++i)
  {
    MQMessage msg("you_topic_name", "TAG", "msg content");
    try
    {
      SendResult sendResult = producer.send(msg, queueSelector, &i, 3, false);
      std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID: "
 << sendResult.getMsgId() << std::endl;
      this_thread::sleep_for(chrono::seconds(1));
    }
    catch (MQException e)
    {
      std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() << std::endl;
    }
  }
  std::cout << "Send " << count << " messages OK, costs" << std::endl;
  producer.shutdown();
  return 0;
}
```

消费顺序消息

```
#include <iostream>
#include <thread>
#include "rocketmq/DefaultMQPushConsumer.h"
using namespace rocketmq;

class OrderlyMessageListener : public MessageListenerOrderly
{
public:
  ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs)
  {
    for (auto item = msgs.begin(); item != msgs.end(); item++)
    {
      std::cout << "Received Message Topic:" << item->getTopic() << ", MsgId:"
 << item->getMsgId() << std::endl;
    }
    return CONSUME_SUCCESS;
  }
};

int main(int argc, char *argv[]){
  DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer("GID_group");
  consumer->setNamesrvAddr("your access point");
  consumer->setSessionCredentials("ak", "sk", "VOLC");
  OrderlyMessageListener *messageListener = new OrderlyMessageListener();
  consumer->subscribe("topic_name", "tag");
  consumer->registerMessageListener(messageListener);
  consumer->start();
  std::this_thread::sleep_for(std::chrono::seconds(60));
  consumer->shutdown();  return 0;
}
```

使用C++客户端收发事务消息

简介

业务侧通过 sendMessageInTransaction 发送消息到 RocketMQ 服务端。

业务侧通过 executeLocalTransaction 执行本地事务。

实现业务查询事务执行是否成功的接口 checkLocalTransaction。

使用C++客户端发送事务消息

```
#include <iostream>
#include <chrono>
#include <thread>
```

```
#include "rocketmq/TransactionMQProducer.h"
#include "rocketmq/MQClientException.h"
#include "rocketmq/TransactionListener.h"

using namespace std;
using namespace rocketmq;

class DefineTransactionListener : public TransactionListener
{
public:
  LocalTransactionState executeLocalTransaction(const MQMessage &msg, void *arg)
  {
    /*
      执行本地事务
      1.成功返回COMMIT_MESSAGE
      2.失败返回ROLLBACK_MESSAGE
      3.不确定返回UNKNOWN。服务端会触发定时任务回查状态
    */
    std::cout << "Execute Local Transaction,Received Message Topic:" << msg.getTopic()
        << ", transactionId:" << msg.getTransactionId() << std::endl;
    return UNKNOWN;
  }

  LocalTransactionState checkLocalTransaction(const MQMessageExt &msg)
  {
    /*
      回查本地事务执行情况
      1.成功返回COMMIT_MESSAGE
      2.失败返回ROLLBACK_MESSAGE
      3.不确定返回UNKNOWN。则等待下次定时任务回查。
    */
    std::cout << "Check Local Transaction,Received Message Topic:" << msg.getTopic()
        << ", MsgId:" << msg.getMsgId() << std::endl;
    return COMMIT_MESSAGE;
  }
};

int main(){
  TransactionMQProducer producer("producer_group_name");
  producer.setNamesrvAddr("accesspoint");
  producer.setSessionCredentials("ak", "sk", "channel");
  DefineTransactionListener *exampleTransactionListener = new DefineTransactionListener();
  producer.setTransactionListener(exampleTransactionListener);
  producer.start();
  int count = 3;
  for (int i = 0; i < count; ++i)
  {
```

```
    MQMessage msg("TRANSACTION TOPIC", "TAG", "Transaction content");
    try
    {
      SendResult sendResult = producer.sendMessageInTransaction(msg, &i);
      std::cout << "SendResult:" << sendResult.getSendStatus()
            << ", Message ID:" << sendResult.getMsgId()
            << std::endl;
      this_thread::sleep_for(chrono::seconds(1));
    }
    catch (MQException e)
    {
      std::cout << "ErrorCode:" << e.GetError() << " Exception:" << e.what() << std::endl;
    }
  }
  std::cout << "Send " << count << " messages OK " << endl;  std::cout <<
"Wait for local transaction check..... " << std::endl;
  for (int i = 0; i < 6; ++i)
  {
    this_thread::sleep_for(chrono::seconds(10));
    std::cout << "Running " << i * 10 + 10 << " Seconds....." << std::endl;
  }
  producer.shutdown();
  return 0;
}
```

使用C++客户端消费事务消息

和消费普通消息一致，请参考对应部分。

使用C++客户端收发延时消息

使用C++客户端发送延时消息

```
#include <iostream>
#include <chrono>
#include <thread>
#include <string>
#include "rocketmq/DefaultMQProducer.h"

using namespace std;
using namespace rocketmq;

int main(){
  DefaultMQProducer producer("producer_group_name");
  producer.setNamesrvAddr("accesspoint");
  producer.setSessionCredentials("ak", "sk", "volc");
  producer.start();
```

```
    int count = 64;
    for (int i = 0; i < count; ++i)
    {
        MQMessage msg("you topic name", "TAG", "msg content");
        // messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h
        msg.setDelayTimeLevel(5);
        try
        {
            SendResult sendResult = producer.send(msg);
            std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID: "
 << sendResult.getMsgId() << std::endl;
            this_thread::sleep_for(chrono::seconds(1));
        }
        catch (MQException e)
        {
            std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() << std::endl;
        }
    }
    std::cout << "Send " << count << " messages OK, costs" << std::endl;
    producer.shutdown();
    return 0;
}
```

使用C++客户端消费延时消息

和消费普通消息一致，请参考对应部分。

RocketMQ .NET SDK

说明

分布式消息服务RocketMQ兼容了社区版 HTTP SDK，您可以使用社区版 HTTP SDK接入分布式消息服务RocketMQ。

前提条件

1. 下载社区 C# SDK到本地并解压。
2. 使用Visual Studio打开sln文件导入工程。

发送普通消息

```
using System;
using Aliyun.MQ;
using Aliyun.MQ.Model;
namespace MQ.Sample
{
    public class Producer
    {
        // 填写分布式消息服务RocketMQ控制台HTTP接入点
        private const string _endpoint = "${HTTP_ENDPOINT}";
```

```
    // 填写AccessKey，在管理控制台创建
    private const string _accessKeyId = "${ACCESS_KEY}";
    // 填写SecretKey 在管理控制台创建
    private const string _secretAccessKey = "${SECRET_KEY}";
    // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
    private const string _topicName = "${TOPIC}";
    // Topic所属实例ID，默认实例为空
    private const string _instanceId = "${INSTANCE_ID}";
    private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId,
_secretAccessKey, _endpoint);

    static MQProducer producer = _client.GetProducer(_instanceId, _topicName);
    static void Main(string[] args)
    {
      try
      {
        // 循环发送4条消息。
        for (int i = 0; i < 4; i++)
        {
          TopicMessage sendMsg;
          // 消息内容。
          sendMsg = new TopicMessage("hello mq");
          // 设置消息的自定义属性。
          sendMsg.PutProperty("a", i.ToString());
          // 设置消息的Key。
          sendMsg.MessageKey = "MessageKey";

          TopicMessage result = producer.PublishMessage(sendMsg);
          Console.WriteLine("publish message success:" + result);
        }
      }
      catch (Exception ex)
      {
        Console.Write(ex);
      }
    }
  }
}
```

消费普通消息

```
using System;using System.Collections.
Generic;using System.Threading;using Aliyun.MQ.Model;using Aliyun.MQ.Model.Exp;
 namespace MQ.Sample{
 public class Consumer
  {
    // 填写分布式消息服务RocketMQ控制台HTTP接入点
    private const string _endpoint = "${HTTP_ENDPOINT}";
```

```
    // 填写AccessKey，在管理控制台创建
    private const string _accessKeyId = "${ACCESS_KEY}";
    // 填写SecretKey 在管理控制台创建
    private const string _secretAccessKey = "${SECRET_KEY}";
    // 所属的 Topic
    private const string _topicName = "${TOPIC}";
    // Topic所属实例ID，默认实例为空
    private const string _instanceId = "${INSTANCE_ID}";
    // 您在控制台创建的 Consumer ID(Group ID)
    private const string _groupId = "${GROUP_ID}";

    private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccessKey,
_endpoint);
    static MQConsumer consumer = _client.GetConsumer(_instanceId, _topicName,
_groupId, null);

    static void Main(string[] args)
    {
        // 在当前线程循环消费消息，建议是多开个几个线程并发消费消息
        while (true)
        {
            try
            {
                // 长轮询消费消息
                // 长轮询表示如果topic没有消息则请求会在服务端挂住3s，3s内如果有消息可以消费则
立即返回
                List<Message> messages = null;

                try
                {
                    messages = consumer.ConsumeMessage(
                        3, // 一次最多消费3条(最多可设置为16条)
                        3 // 长轮询时间3秒（最多可设置为30秒）
                    );
                }
                catch (Exception exp1)
                {
                    if (exp1 is MessageNotExistException)
                    {
                        Console.WriteLine(Thread.CurrentThread.Name + " No new message, " + ((MessageN
otExistException)exp1).RequestId);
                        continue;
                    }
                    Console.WriteLine(exp1);
                    Thread.Sleep(2000);
                }
```

```
            if (messages == null)
            {
                continue;
            }

            List<string> handlers = new List<>();
            Console.WriteLine(Thread.CurrentThread.Name + " Receive Messages:");
            // 处理业务逻辑
            foreach (Message message in messages)
            {
                Console.WriteLine(message);
                handlers.Add(message.ReceiptHandle);
            }
            // Message.nextConsumeTime前若不确认消息消费成功，则消息会重复消费
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样
            try
            {
                consumer.AckMessage(handlers);
                Console.WriteLine("Ack message success:");
                foreach (string handle in handlers)
                {
                    Console.Write("\t" + handle);
                }
                Console.WriteLine();
            }
            catch (Exception exp2)
            {
                // 某些消息的句柄可能超时了会导致确认不成功
                if (exp2 is AckMessageException)
                {
                    AckMessageException ackExp = (AckMessageException)exp2;
                    Console.WriteLine("Ack message fail, RequestId:" + ackExp.RequestId);
                    foreach (AckMessageErrorItem errorItem in ackExp.ErrorItems)
                    {
                        Console.WriteLine("\tErrorHandle:" + errorItem.ReceiptHandle +
", ErrorCode:" + errorItem.ErrorCode + ", ErrorMsg:" + errorItem.ErrorMessage);
                    }
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex);
            Thread.Sleep(2000);
        }
    }
  }
}
```

RocketMQ PHP SDK

说明

分布式消息服务RocketMQ兼容了社区版 HTTP SDK，您可以使用社区版 HTTP SDK接入分布式消息服务RocketMQ。

前提条件：

1. 在PHP安装目录下的composer.json文件中加入社区PHP SDK 依赖。
2. 使用Composer安装依赖。

```
composer install
```

发送普通消息

```
<?phprequire "vendor/autoload.php";use MQ\Model\TopicMessage;use MQ\MQClient;class NormalPr
oducerExample{
    private $client;
    private $producer;

    public function __construct()
    {
        $this->client = new MQClient(
        // 填写分布式消息服务RocketMQ控制台HTTP接入点
            "${HTTP_ENDPOINT}",
            // 填写AccessKey，在管理控制台创建
            "${ACCESS_KEY}",
            // 填写SecretKey 在管理控制台创建
            "${SECRET_KEY}"
        );

        // 所属的 Topic
        $topic = "${TOPIC}";
        // Topic所属实例ID，默认实例为空NULL
        $instanceId = "${INSTANCE_ID}";

        $this->producer = $this->client->getProducer($instanceId, $topic);
    }

    public function run()
    {
        try {
            for ($i = 1; $i <= 4; $i++) {
                $publishMessage = new TopicMessage(
                    "Hello RocketMQ" // 消息内容
                );

                // 设置属性
                $publishMessage->putProperty("a", $i);
```

```
            // 设置消息KEY
            $publishMessage->setMessageKey("MessageKey");

            $result = $this->producer->publishMessage($publishMessage);
            print "Send mq message success. msgId is:" . $result->getMessageId() . ", bodyMD5 is:" .
$result->getMessageBodyMD5() . "\n";
        }
    } catch (\Exception $e) {
        print_r($e->getMessage() . "\n");
    }
}}$instance = new NormalProducerExample();$instance->run();?>
```

消费普通消息

```
<?phprequire "vendor/autoload.php";use MQ\MQClient;class ConsumerExample{
    private $client;
    private $consumer;

    public function __construct()
    {
        $this->client = new MQClient(
            // 填写分布式消息服务RocketMQ控制台HTTP接入点
            "${HTTP_ENDPOINT}",
            // 填写AccessKey，在管理控制台创建
            "${ACCESS_KEY}",
            // 填写SecretKey 在管理控制台创建
            "${SECRET_KEY}"
        );

        // 所属的 Topic
        $topic = "${TOPIC}";
        // 您在控制台创建的 Consumer ID(Group ID)
        $groupId = "${GROUP_ID}";
        // Topic所属实例ID，默认实例为空NULL
        $instanceId = "${INSTANCE_ID}";

        $this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId, "TagA");
    }

    public function run()
    {
        // 在当前线程循环消费消息，建议是多开个几个线程并发消费消息
        while (True) {
            try {
                // 长轮询消费消息
                // 长轮询表示如果topic没有消息则请求会在服务端挂住3s，3s内如果有消息可以消费则立
即返回
                $messages = $this->consumer->consumeMessage(
```

```
            3, // 一次最多消费3条(最多可设置为16条)
            3 // 长轮询时间3秒（最多可设置为30秒）
        );
    } catch (\MQ\Exception\MessageResolveException $e) {
        // 当出现消息Body存在不合法字符，无法解析的时候，会抛出此异常。
        // 可以正常解析的消息列表。
        $messages = $e->getPartialResult()->getMessages();
        // 无法正常解析的消息列表。
        $failMessages = $e->getPartialResult()->getFailResolveMessages();

        $receiptHandles = array();
        foreach ($messages as $message) {
            // 处理业务逻辑。
            $receiptHandles[] = $message->getReceiptHandle();
            printf("MsgID %s\n", $message->getMessageId());
        }
        foreach ($failMessages as $failMessage) {
            // 处理存在不合法字符，无法解析的消息。
            $receiptHandles[] = $failMessage->getReceiptHandle();
            printf("Fail To Resolve Message. MsgID %s\n", $failMessage->getMessageId());
        }
        $this->ackMessages($receiptHandles);
        continue;
    } catch (\Exception $e) {
        if ($e instanceof MQ\Exception\MessageNotExistException) {
            // 没有消息可以消费，接着轮询
            printf("No message, contine long polling!RequestId:%s\n", $e->getRequestId());
            continue;
        }



        print_r($e->getMessage() . "\n");

        sleep(3);
        continue;
    }

    print "consume finish, messages:\n";

    // 处理业务逻辑
    $receiptHandles = array();
    foreach ($messages as $message) {
        $receiptHandles[] = $message->getReceiptHandle();
        printf("MessageID:%s TAG:%s BODY:%s \nPublishTime:%d, FirstConsumeTime:%d,
\nConsumedTimes:%d, NextConsumeTime:%d,MessageKey:%s\n",
```

```
            $message->getMessageId(), $message->getMessageTag(),
$message->getMessageBody(),
            $message->getPublishTime(), $message->getFirstConsumeTime(),
$message->getConsumedTimes(), $message->getNextConsumeTime(),
            $message->getMessageKey());
          print_r($message->getProperties());
        }

        // $message->getNextConsumeTime()前若不确认消息消费成功，则消息会重复消费
        // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样
        print_r($receiptHandles);
        try {
          $this->ackMessages($receiptHandles);
        } catch (\Exception $e) {
          if ($e instanceof MQ\Exception\AckMessageException) {
            // 某些消息的句柄可能超时了会导致确认不成功
            printf("Ack Error, RequestId:%s\n", $e->getRequestId());
            foreach ($e->getAckMessageErrorItems() as $errorItem) {
              printf("\tReceiptHandle:%s, ErrorCode:%s, ErrorMsg:%s\n",
$errorItem->getReceiptHandle(), $errorItem->getErrorCode(), $errorItem->getErrorCode());
            }
          }
        }
        print "ack finish\n";
      }

    }

    public function ackMessages($receiptHandles)
    {
      try {
        $this->consumer->ackMessage($receiptHandles);
      } catch (\Exception $e) {
        if ($e instanceof MQ\Exception\AckMessageException) {
          // 某些消息的句柄可能超时，会导致消费确认失败。
          printf("Ack Error, RequestId:%s\n", $e->getRequestId());
          foreach ($e->getAckMessageErrorItems() as $errorItem) {
            printf("\tReceiptHandle:%s, ErrorCode:%s, ErrorMsg:%s\n",
$errorItem->getReceiptHandle(), $errorItem->getErrorCode(), $errorItem->getErrorCode());
          }
        }
      }
    }}$instance = new ConsumerExample();$instance->run();?>
```

常见问题

计费类

如何为购买的消息队列RocketMQ实例续费？

为防止资源到期或者浪费，已经购买包月实例的用户，可执行续费操作，延长 RocketMQ实例的有效期，也可以设置到期自动续费的相关操作。

开通的RocketMQ实例资源包何时生效？

天翼云分布式消息服务RocketMQ资源包在客户支付成功之后，系统经过初始化完成后即可生效使用。

购买类

RocketMQ实例的Topic数量是否有限制？

在天翼云分布式消息服务RocketMQ中，不同规格RocketMQ支持的Topic个数不同，详细请参见产品规格。

RocketMQ包周期实例不支持删除吗？

可以删除。登录RocketMQ控制台，在包周期实例所在行，单击“更多 > 退订”，即可完成实例的删除。

分布式消息服务RocketMQ资源包可以叠加购买吗？

天翼云分布式消息服务RocketMQ支持叠加购买。

产品订购时可选资源池节点不一致？

已上线资源池节点的剩余容量达到一定比例后，为确保老客户权益，将不再面向新客户开放，产品订购时的可选资源节点范围以实际为准。

操作类

服务端异常

地址冲突，在启动broker时，出现地址已在使用错误

```
load config properties file OK, ../conf/2m-noslave/broker-b.properties
java.net.BindException: 地址已在使用
        at sun.nio.ch.Net.bind0(Native Method)
        at sun.nio.ch.Net.bind(Net.java:444)
        at sun.nio.ch.Net.bind(Net.java:436)
        at sun.nio.ch.ServerSocketChannelImpl.bind(ServerSocketChannelImpl.java:214)
        at sun.nio.ch.ServerSocketAdaptor.bind(ServerSocketAdaptor.java:74)
        at io.netty.channel.socket.nio.NioServerSocketChannel.doBind(NioServerSocketChannel.java:125)
        at io.netty.channel.AbstractChannel$AbstractUnsafe.bind(AbstractChannel.java:484)
        at io.netty.channel.DefaultChannelPipeline$HeadContext.bind(DefaultChannelPipeline.java:1080)
        at io.netty.channel.AbstractChannelHandlerContext.invokeBind(AbstractChannelHandlerContext.jav
        at io.netty.channel.AbstractChannelHandlerContext.bind(AbstractChannelHandlerContext.java:415)
        at io.netty.channel.DefaultChannelPipeline.bind(DefaultChannelPipeline.java:903)
        at io.netty.channel.AbstractChannel.bind(AbstractChannel.java:197)
        at io.netty.bootstrap.AbstractBootstrap$2.run(AbstractBootstrap.java:350)
        at io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.ja
        at io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:357)
        at io.netty.util.concurrent.SingleThreadEventExecutor$2.run(SingleThreadEventExecutor.java:116
        at java.lang.Thread.run(Thread.java:745)
```

解决方案：修改配置文件里面的listenPort的值，然后重新启动。

```
brokerClusterName=DefaultCluster
brokerName=broker-a
brokerId=0
deleteWhen=04
fileReservedTime=48
brokerRole=ASYNC_MASTER
flushDiskType=ASYNC_FLUSH
namesrvAddr= 172.31.25.24:9876
brokerIP1=172.31.25.24
listenPort=10911
storePathRootDir=/data2/broker-a-s/store
storePathCommitLog=/data2/broker-a-s/store/commitlog
```

brokerName不匹配，启动出现异常：broker-b does not match the expected group name: broker-a

问题原因：启动其他Master broker服务时，直接将之前使用过的store目录以及bdb目录复制过来，仅仅只是修改了brokerName，导致此问题出现。

解决方案：2.0以后版本brokerName一旦创建启动后就不能改变，否则只能删除store目录才能解决。

service not available

发送消息一定量的时候，出现 create maped file failed, please make sure OS and JDK both 64bit。或者当topic的队列数位1024个的时候，会出现service not available now, maybe disk full，maybe your broker machine memory too small。

解决方案：使用ulimit-a命令查询系统参数，检查open files是否超过655350，max memory size是为否unlimited，若不是，需要重新根据安装手册的步骤，重新调整系统参数。

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 4133973
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 655360
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) unlimited
cpu time               (seconds, -t) unlimited
max user processes              (-u) 16384
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

磁盘空间不足

当磁盘空间大于85%时，会出现“ CODE: 14 DESC: service not available now, maybe disk full, CL: 0.87 CQ: 0.87 INDEX: 0.87, maybe your broker machine memory too small.”的异常。

解决方案：消息中间件有两种策略， 包括数据高安全性与服务高可靠性，分别如下：

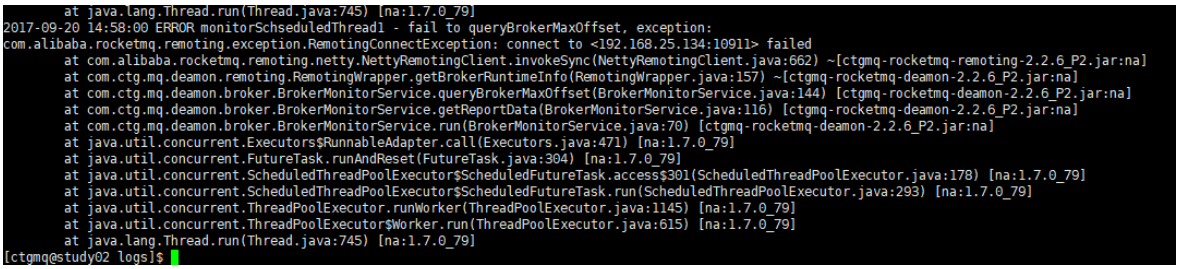
策略	配置	说明
数据高安全性	cleanFileForciblyEnable=false	若磁盘使用率大于85%，有消息生产时或默认凌晨四点，则触发删除过期的消息，若没过期消息则不会被删除
服务高可靠性	cleanFileForciblyEnable=true	若磁盘使用率大于85%，有消息生产时或默认凌晨四点，则触发删除消息（在有效期内的数据将被删除）

若磁盘使用率大于85%，策略为数据高安全性，且无过期文件，可以按实际需求，减少数据保存时间来触发消息删除，腾出磁盘空间。

- 使用updateBrokerConfig命令，修改fileReservedTime属性，此属性为消息保存时间，单位为小时。按需减少保存时间，则可以腾出磁盘空间。
- 主备都需要同时修改。

通过deamon拉起broker时报错

deamon.log日志中报Fail to queryBrokerMaxOffset。



问题原因：

- 配置文件错误。
- 做过主备切换，然后手动干预或重启集群，启动进程的地址和角色与zookeeper中存储的不同，造成启动失败。
- 上次启动失败后未清理错误数据。

解决方法：

- 删除zk中该集群的信息。
- 核对配置文件，确保端口和路径有效。

删除running目录。

- 重新通过自动部署或者手动部署启动进程。

消费进度停滞不前

通过consumerProgress命令查询消费进度某些队列无变化，而客户端正在正常消费。

问题原因：某些队列有消息没有签收，导致服务端消费进度没有后移。

解决方案：通过consumerProgress命令显示的consumer offset找到对应消息，并按如下判断执行：

- 如果是BDB消费模式，重启应用即可或者通过以下api void com.ctg.mq.api.IMQAckHandler.ackMessageSuccess(String msgID)签收卡住的消息即可。
- 如果是原生有序消费模式，重启应用即可。
- 如果是原生无序消费模式，启动一个同消费组的实例，会将该消息签收。

删除topic失败

删除topic时出现topic **** is consuming by consumer ****，或者topic *** is publishing by producer ***异常。

问题原因：删除topic必须没有生产者和消费者正在订阅该topic（与该topic相关的生产者消费者都必须离线），否则会失败。

解决方案：

- 可以通过一下方式查看是否还有客户端连接该topic：管理平台->主题管理->详情->生产组|消费组->连接实例。
- 如果使用命令行删除有序队列，需要使用集群删除，例如：sh mqadmin deleteTopic -n 10.142.90.33:9876 -c mq_cluster -t mytesttopic。

- 如果使用命令行删除无序队列，可以使用broker删除，例如：sh mqadmin deleteTopic -n 10.142.90.33:9876 -b 10.142.90.33:10911 -t mytesttopic。

启动broker时BDB报错

问题原因：可能是迁移了store目录或者更换了broker的组名、地址或端口。

解决方案：删除store目录下的consumeStore目录，重启broker即可解决。

从broker已启动，但clusterList看不到

从broker已启动，但无法加入到集群（clusterList查询不到）。

问题原因：

- 查看/etc/hosts文件，机器名与IP的映射关系是否填写有误。
- 查看防火墙设置（是否有端口未开放），listenPort 到 listenPort+2的端口都 需要开放（如果主broker的listenPort=10911，那么10911、10912、10913都要开放）。

通过命令行创建有序topic，但是web管理台显示的是无序的

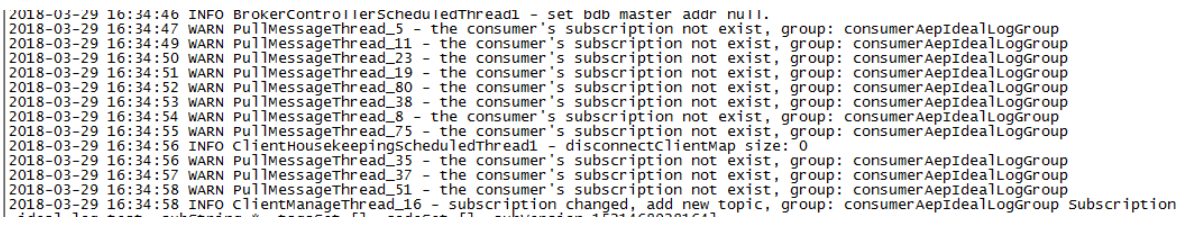
通过updateTopic命令，加-o true创建有序topic，但是web端查询的时候 显示是无序的。

问题原因：集群有多个namesrv，但是创建的时候只填了一个namesrv。

解决方案：创建时加上这个broker集群的所有namesrv，中间用分号分割，例如：sh mqadmin updateTopic -n “10.142.90.30:9876;10.142.90.28:9876”-t crmTopic -o true。

消费者订阅关系不存在

broker.log日志报错：the consumer's subscription not exist, group: consumerAepIdealLogGroup。



问题原因：使用同个订阅组，同时消费不同的topic，订阅关系会被覆盖。

解决方案：不能使用同个订阅组的消费组去订阅不同的topic，如果需要变换订阅关系，请关闭旧消费者。

使用clusterList查询 主TPS不为0，从TPS一直为0

这种情况最大的可能是从同步出错，可以做进一步的确认，查看store.log或者 stoererror.log，一般会看到有持续的报错信息。这种情况可以删除从的store目录，重新进行同步。

注意：部署有高可用模块或者主的brokerRole=ASYNC_MASTER，否则停止从的时候，生产会报错。

解决方案：

- 手工停止从broker（kill pid，注意不要加-9，如果自动拉起broker参数设置为true，则需要先关掉从的deamon）。
- 删除或者备份从的store目录（为保险起见，空间允许的话，可以mv备份，不要直接删除。
- 手工启动broker（sh sh/broker_*.sh）。
- 从broker启动完成后，用clusterList查看，可以看到从的TPS比较高，因为正在同步。

主broker异常恢复

需要走异常恢复流程的一般是consumequeue生成有问题，导致无法拉取消息（注意 有多种情况会导致无法拉取消息，不一定是consumequeue有问题，注意判断）、根据offset查询报错。

异常恢复流程：

1.停止需要恢复这一组broker的主从deamon，主从broker。

2.删除主broker store目录下的checkpoint consumequeue consumeStore index（也可以mv 改下名字来备份）。

3.检查store目录下的abort文存是否存在，如果不存在新建一个（touch abort）。

4.启动主broker，查看store.log，可以看到打印恢复过程的日志，如果没有报错，说明恢复成功。

5.如果commitlog文件比较多，可能恢复时间较长，可以通过查看store.log或者broker端口是否起来判断恢复是否完成。

6.主broker起来后，通过消费或者根据offset查询消息来验证是否恢复成功。

7.如果主broker恢复成功，启动从broker，启动主从deamon。

RPC异常（所以服务端组件均可能出现）

```
2018-11-09 16:36:08 ERROR NettyServerCodecThread_2 - decode exception, 192.168.10.1:53436
io.netty.handler.codec.TooLongFrameException: Adjusted frame length exceeds 16777216: 1195725860 - discarded
        at io.netty.handler.codec.LengthFieldBasedFrameDecoder.fail(LengthFieldBasedFrameDecoder.java:499)
        at io.netty.handler.codec.LengthFieldBasedFrameDecoder.failIfNecessary(LengthFieldBasedFrameDecoder.java:477)
        at io.netty.handler.codec.LengthFieldBasedFrameDecoder.decode(LengthFieldBasedFrameDecoder.java:403)
        at org.apache.rocketmq.remoting.netty.NettyDecoder.decode(NettyDecoder.java:43)
```

问题原因：使用非组件RPC协议访问导致，比如用http协议、或者telnet等，均可导致decode错误。

解决方案：无需解决，应服务端及客户端RPC请求均无影响。

常见问题

应用客户端

连接拒绝Connect faild

```
2015-11-10 12:00:00 WARN BrokerControllerScheduledThread1 - registerBroker Exception, 134.130.134.46:9876
com.alibaba.rocketmq.remoting.exception.RemotingConnectException: connect to <134.130.134.46:9876> failed
        at com.alibaba.rocketmq.remoting.netty.NettyRemotingClient.invokeSync(NettyRemotingClient.java:641) ~[rocketmq-remoting-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.out.BrokerOuterAPI.registerBroker(BrokerOuterAPI.java:153) ~[rocketmq-broker-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.out.BrokerOuterAPI.registerBrokerAll(BrokerOuterAPI.java:193) ~[rocketmq-broker-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.a.a(BrokerController.java:617) [rocketmq-broker-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.a$7.run(BrokerController.java:587) [rocketmq-broker-3.2.6.jar:na]
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:471) [na:1.7.0_79]
        at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:304) [na:1.7.0_79]
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$301(ScheduledThreadPoolExecutor.java:178) [na:1.7.0_79]
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293) [na:1.7.0_79]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145) [na:1.7.0_79]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615) [na:1.7.0_79]
        at java.lang.Thread.run(Thread.java:745) [na:1.7.0_79]
2015-11-10 13:16:33 WARN BrokerControllerScheduledThread1 - registerBroker Exception, 134.130.134.46:9876
com.alibaba.rocketmq.remoting.exception.RemotingTimeoutException: wait response on the channel <134.130.134.46:9876> timeout, 3000(ms)
        at com.alibaba.rocketmq.remoting.netty.NettyRemotingAbstract.invokeSyncImpl(NettyRemotingAbstract.java:375) ~[rocketmq-remoting-3.2.6.jar:na]
        at com.alibaba.rocketmq.remoting.netty.NettyRemotingClient.invokeSync(NettyRemotingClient.java:622) ~[rocketmq-remoting-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.out.BrokerOuterAPI.registerBroker(BrokerOuterAPI.java:153) ~[rocketmq-broker-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.out.BrokerOuterAPI.registerBrokerAll(BrokerOuterAPI.java:193) ~[rocketmq-broker-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.a.a(BrokerController.java:617) [rocketmq-broker-3.2.6.jar:na]
        at com.alibaba.rocketmq.broker.a$7.run(BrokerController.java:587) [rocketmq-broker-3.2.6.jar:na]
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:471) [na:1.7.0_79]
        at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:304) [na:1.7.0_79]
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$301(ScheduledThreadPoolExecutor.java:178) [na:1.7.0_79]
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293) [na:1.7.0_79]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145) [na:1.7.0_79]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615) [na:1.7.0_79]
        at java.lang.Thread.run(Thread.java:745) [na:1.7.0_79]
"broker.log" 632L, 74762C
```

解决方案：当出现这类问题时，检查当前网络并无异常时，并排查下ulimit -a openfiles是否为1024，修改至65535。

超时异常RemotingTimeoutException

服务器端日志出现RemotingTimeoutException：wait response on the channel **<** 10.4.246.198:10911 **>** timeout, 3000ms。

解决方案：这类情况一般由于客户端与服务端通信出现问题，可以ping Ip 以及telnet ip port 来排查这类，同时也要检查防火墙的问题。

找不到路由No route info of this topic

问题可能原因：

- 没创建topic。
- name server填错了。
- 网络问题无法获取路由。

解决方案：

- 在管理台创建topic。
- 检查客户端配置的namesrv的地址是否配错了。
- 检查网络是否正常。

备不可用SLAVE_NOT_AVAILABLE

当生产者发送消息时，出现“status:SLAVE_NOT_AVAILABLE”，说明从节点发生状况。

解决方案：

- 从节点机器出现问题，重启从节点，并查看网络连接。
- 在多网卡情况下，broker配置文件properties中，需增加配置项，例如：brokerIP1=10.4.246.130，brokerIP2=10.4.246.130
- 防止网卡ip读取错误，取不到从节点信息。

消息体大小越界

客户端报此类异常Fail to send message, for: message body size over max message size, max: 524288。

解决方案：

- 检查服务端的最大消息体大小，即启动broker配置文件的maxMessageSize大小，如未配置，默认是512K。
- 检查客户端设置的最大消息体(默认128k)是否小于当前发送的消息体大小。

注意：ROCKETMQ建议消息体在50K或以下(压缩后)。

组名已经创建

当消息生产端/消费端运行时，报错The producer/consumer group has been created。

问题原因：在同一个jvm里面只允许一个producerGroupName被加载一次 （consumerGroupName同理），否则就会报错。

解决方案：

- 如果使用同一个producerGroupName，部署多个实例（起多个进程）。
- 在一个进程里，起多个线程，共用一个Producer对象实例。

Subscription group not exist或者抛出%retry%的topic没有路由信息

问题原因：没有建立消费关系或者没有创建相关订阅组。

解决方案：在管理台或命令行创建对应订阅组。

Messgae already acked, ackMessage failed

```
2016-03-29 18:50:10,878 ERROR  (TransacUtil.java:187): Thread-19 首次签收消息失败
com.ctg.mq.api.exception.MQAckException: message already acked, ackMessage fail, record not existed: consumeQueueOffset=24
        at com.ctg.mq.api.impl.MQAbstractConsumer.ackMessage(MQAbstractConsumer.java:258)
        at com.ctg.mq.api.impl.MQAbstractConsumer.ackMessage(MQAbstractConsumer.java:420)
        at com.dcs.dispatch.EventListener.BillNodeRespHandle(EventListener.java:426)
        at com.dcs.dispatch.EventListener.work(EventListener.java:222)
        at com.dcs.dispatch.EventListener.run(EventListener.java:168)
Caused by: com.alibaba.rocketmq.client.exception.MQBrokerException: CODE: 511  DESC: message already acked, ackMessage fail, record not existed: consumeQueueOffset=24
For more information, please visit the url, https://github.com/alibaba/RocketMQ/issues/48
        at com.alibaba.rocketmq.client.impl.MQClientAPIImpl.ackMessage(MQClientAPIImpl.java:741)
        at com.alibaba.rocketmq.client.impl.consumer.PullAPIWrapper.ackMessage(PullAPIWrapper.java:386)
        at com.ctg.mq.api.impl.consumer.LightMQPullConsumerImpl.ackMessage(LightMQPullConsumerImpl.java:515)
        at com.ctg.mq.api.consumer.LightMQPullConsumer.ackMessage(LightMQPullConsumer.java:345)
        at com.ctg.mq.api.impl.MQPullConsumerImpl.ackMessage(MQPullConsumerImpl.java:515)
        at com.ctg.mq.api.impl.MQAbstractConsumer.ackMessage(MQAbstractConsumer.java:248)
        ... 4 more
```

解决方案：这种异常表明该消息已被签收，直接跳过即可。

重试签收调用次数

ackMessageRetry重试签收是只需要调一次就够了，还是需要调多次。例如：签收失败后，调用重试签收，如果重试签收也失败，是否需要再次调用重试签收，还是会自动重试签收。

现有版本接口不会自动帮你重试签收的， 重试签收失败后，需要自己再次调用重试签收接口。

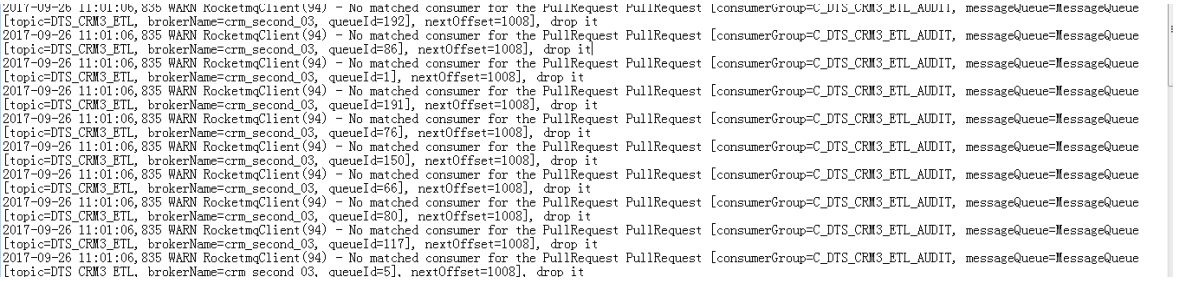
签收时出现不确定异常，如发生超时，或者网络异常时，是需要应用判断消息是否已经签收成功

解决方案：

- 通过管理台“即时查询”模块，查询这消息是否已经签收成功，看结果再做处理。
- 重试签收，如果已经签收会抛已签收异常。主要还是看应用的自己处理。

客户端注册失败

客户端日志报No matched consumer for the PullRequest PullRequest。



问题原因：客户端实例注册失败。

解决方案：检查客户端代码，重启客户端进程。

the consumer message buffer is full, so do flow control

客户端日志出现the consumer message buffer is full, so do flow control

问题原因：push客户端消费过慢，本地缓存队列已满，暂时停止向服务端拉取消息。消费慢的原因可能是网络原因、topic队列数过多、消费者过少，内存过小等。

解决方案：

（1）查看网络是否异常，缓慢。

（2）增加消费者实例。

（3）如果消息不重要，又不方便增加消费者实例，可以减少topic队列数量。

system busy, start flow control for a while

客户端日志出现 [REJECTREQUEST]system busy, start flow control for a while 或者 [PCBUSY_CLEAN_QUEUE]broker busy, start flow control for a while, period in queue。

问题原因：

- 在关闭生产者实例的同时用生产者实例发送消息，连接关闭了netty会拒绝请求。
- 线程少，处理发送请求过慢。

解决方案：

- 应用优化使用流程，禁止在close生产者实例后使用生产者。

- 如果Broker是同步主，那么改成异步主，或者将 sendMessageThreadPoolNums=32且 waitTimeMillsInSendQueue=1000。

消费者消费不到消息如何处理

进入控制台查看订阅管理菜单，检查订阅组是否有消费实例在线，如果不在线检查消费客户端日志是否有连接异常。

检查消费客户端逻辑，是否存在订阅关系不一致的情况。

消费者机器宕机重启是否会造成消息丢失

RocketMQ的消息数据以及订阅信息都是持久化保存的，当消费者下线重新上线后，会Broker持久化的下线前的消费偏移重新开始消费，所以不会发生消息丢失的情况。

订阅消息时是否可以允许消息Tag为空

订阅主题时如果Tag设置为空会导致消费者消费不到消息，如不希望通过Tag进行消息过滤，可以将Tag设置为*，示例如下：

```
consumer.subscribe(topic, "*");
```

客户端连接时出现“signature validate by dauth failed”错误

这种错误的原因一般是由于ACL认证失败，较大的可能是客户端配置的AccessKey和SecretKey出现错误，可以检查下这两项配置是否输入有误。

管理类

RocketMQ实例是否支持扩容？

可以扩容。登录RocketMQ控制台，在包周期实例所在行，单击“更多 > 扩容”，即答完成实例的扩容。

消费的最长保留时间是多久？

一般情况下消息如果未被消费会一直保留，只有被消费后，才会被删除。但是如果设置了过期时间（TTL），则以TTL时间为准。

产品类

顺序消息和普通消息的区别是什么

最大的区别在于是否能保证消息生产和消费的顺序一致。

对于顺序消息，消息均根据ShardingKey进行区块分区，同一分区内的消息消费满足先进先出，保证分区有序，不同分区的消息消费顺序不做要求。

普通消息则没有该项保证，消息消费的顺序跟生产的顺序不一定保证一致性。

RocketMQ集群消费和广播消费区别是什么

使用集群消费模式时，MQ内任意一条消息只需被订阅组集群内的任意一个消费者消费即可。

使用广播消费模式时，MQ内每条消息都会投递到订阅组集群的所有消费者，每条消息至少被每个消费者消费一次。

多个订阅组订阅同一个主题时消息如何被消费

RocketMQ中订阅关系并非是一对一的，一个主题可以被一个或多个订阅组订阅，但不同订阅组之间的消费是互不影响的，它们各自维护自己在当前主题的消费偏移信息，每一条消息都会被订阅该主题的订阅组接收到。

消息消费失败是否会有重试机制

在push消费模式下，RocketMQ在消费者消费消息失败后会通过将消息重新投递到该订阅组的重试队列在一定时间后会被消费者重新消费到，如果多次失败则会多次重复上述的重试过程，超过最大次数之后（创建订阅组时可配置）会将消息投递到死信队列。

消息消费的负载均衡策略是什么

在push消费模式下，RocketMQ的一个Queue同一时间内只会被一个消费者消费，一个主题内的全部Queue会根据负载均衡策略（默认的是根据平均分配原则）分配给订阅组的全部消费者。

消息消费时是否会出现重复消费的情况

RocketMQ可能出现重复消费的情况，比如在Push模式的消费模式下，由于网络等原因可能出现本地消费者实际消费到超过Broker保存的消费偏移的消息，如果此时发生消费者下线重新上线就会拉取到已经消费过消息。建议客户端的逻辑保证消费的幂等性。