消息队列 RocketMQ 版

RocketMQ 5.x

腾讯云

文档目录

RocketMQ 5.x
RocketMQ 5.x 简介
产品概述

最近更新时间：2024-10-14 17:52:01

消息队列 RocketMQ 版 5.x 系列是腾讯云基于最新的 Apache RocketMQ 5.0 版本构建的分布式消息中间件，完全兼容 Apache RocketMQ 5.0 的使用和概念，支持开源社区版本的客户端零改造接入。
消息队列 RocketMQ 版 5.x 系列除了兼具低延迟、高性能、高可靠、万亿级消息容量和灵活可扩展等历史版本的特点之外，充分结合云原生大潮下的基础设施和生态技术，提高资源利用和弹性能力。

产品优势

最近更新时间：2024-10-14 11:34:23

消息队列 RocketMQ 版 5.x 系列除了兼具低延迟、高性能、高可靠、万亿级消息容量和灵活可扩展等历史版本的特点之外，充分结合云原生大潮下的基础设施和生态技术，提高资源利用和弹性能力。
相比于自建的 RocketMQ 和 消息队列 TDMQ RocketMQ 版 4.x 系列， 5.x 系列还具有以下优势：

存储和计算弹性

TDMQ RocketMQ 版 5.x 系列使用了存算分离的架构，充分提高了资源利用率和弹性能力。存储使用按量后付费，客户无需为峰值提前存储资源，按实际使用量进行付费，有效降低客户的实际成本。计算规格支持弹性TPS，客户无需为计划外的峰值预留计算资源，有效降本。

轻量化 SDK

TDMQ RocketMQ 版 5.x 系列兼容开源社区 SDK，享受开源社区迭代带来的红利。5.x 系列的客户端更加轻量，采用了全新的极简的 API 设计，更易被集成和使用，同时 5.x 系列提供了更多语言的 SDK，使得开发者在使用时有更多的技术栈选择。

基础功能增强

TDMQ RocketMQ 版 5.x 系列有了进一步的功能增强，如更加灵活的消息保留时间控制，可以根据集群或者 Topic 粒度设置消息保留时间。消费者组有了更多的白屏化设置，如可以在服务端指定消息重试次数和自由绑定死信队列等功能。

可观测性

TDMQ RocketMQ 版 5.x 系列增加了更加丰富的指标，如消息堆积场景相关指标、关键接口的耗时指标、错误分布指标、存储读写流量等指标。这些指标都得以对接腾讯云的监控和告警功能，同时提供完善的云 API，支持集成自助运维系统。

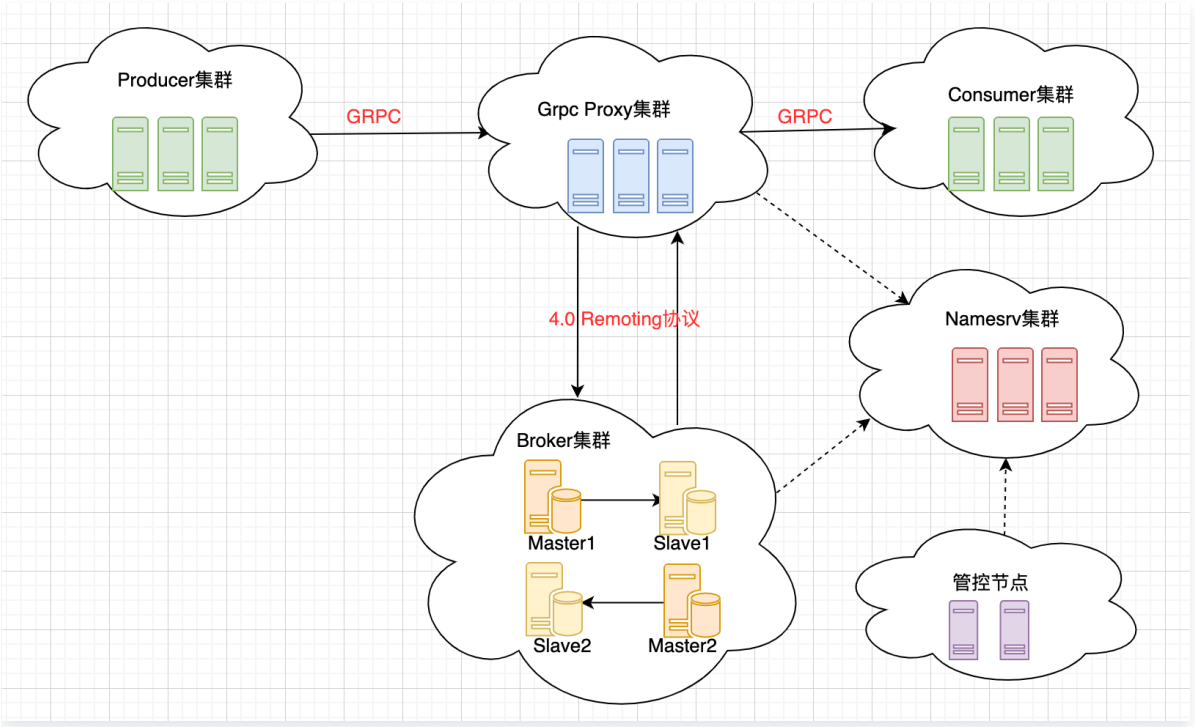
技术架构

最近更新时间：2024-10-14 18:18:31

本文主要介绍消息队列TDMQ RocketMQ 版 5.x 系列的部署架构，方便您更好地理解消息队列 RocketMQ 版的架构原理。

部署架构

消息队列 RocketMQ 版的系统部署架构图如下：



消息队列 TDMQ RocketMQ 版 5.x 系列引入了新的 GRPC 协议和 Proxy 组件，实现了存算分离的架构，对 RocketMQ 的运维和使用都会带来巨大的变化。

其中涉及的各个概念如下：

- Producer 集群： 客户侧应用，负责生产并发送消息。
- Consumer 集群：客户侧应用，负责订阅和消费处理消息。
- Nameserver 集群： 服务端应用，负责路由寻址和 Broker 心跳注册。为保证高可用，默认跨可用区部署。
  - 心跳注册：NameServer 相当于注册中心的角色，各个角色的机器都要定时向 NameServer 上报自己的状态，如果超时未上报，NameServer 会认为某个机器出现故障不可用了，从而将这个机器从可用列表中删除。
  - 路由寻址：每个 NameServer 中都保存着 Broker 集群的整个路由信息和用于客户端查询的队列信息，生产者和消费者通过 NameServer 去获取整个Broker 集群的路由信息，从而进行消息的投递和消费。
- Proxy 集群：全新的弹性无状态代理服务，为保证高可用，默认跨可用区部署。将 4.x 中的 Broker 职责进行拆分，对于客户端协议适配、权限管理、消费管理等计算逻辑进行抽离。
- Broker集群：与 4.x 产品系列相比，在 5.x 系列中，Broker 更专注于存储能力的持续优化。为保证高可用，默认跨可用区部署。

使用限制

最近更新时间：2025-01-25 11:40:32

本文列举了 TDMQ RocketMQ 版 5.x 集群中对一些指标和性能的限制，请您在使用中注意不要超出对应的限制值，避免出现异常。

集群

限制类型	限制
集群名称长度	3-64个字符
集群 TPS 上限	不同集群按照规格进行限制，超出会被限流。TPS 按照消息类型和消息大小进行折算，折算规格见 计费说明 。
公网安全策略	50 条

Topic

限制类型	体验版集群限制	基础版集群限制	专业版集群限制	铂金版集群限制
单集群内 Topic 数量上限	50，死信队列和重试队列对应的 Topic 不占用配额	100-200，按不同的集群规格计算，控制台可以自助提升额度，详细参见 计费说明，死信队列和重试队列对应的 Topic 不占用配额	300-1000，按不同的集群规格计算，控制台可以自助提升额度，详细参见 计费说明，死信队列和重试队列对应的 Topic 不占用配额	1000-10000，按不同的集群规格计算，控制台可以自助提升额度，详细参见 计费说明，死信队列和重试队列对应的 Topic 不占用配额
Topic 名称长度	3-64个字符	3-64个字符	3-64个字符	3-64个字符
单 Topic 最大生产者数量	1000	1000	1000	1000
单 Topic 最大消费者数量	500	500	500	500

Group

限制类型	体验版集群限制	基础版集群限制	专业版集群限制	铂金版集群限制
单集群内 Group 数量上限	500	1000-2000，按不同的集群规格计算，控制台可以自助提升额度，详细参见 计费说明	3000-10000，按不同的集群规格计算，控制台可以自助提升额度，详细参见 计费说明	10000-100000，按不同的集群规格计算，控制台可以自助提升额度，详细参见 计费说明
Group 名称长度	3-64个字符	3-64个字符	3-64个字符	3-64个字符

消息

限制类型	体验版集群限制	基础版集群限制	专业版集群限制	铂金版集群限制
消息最大保留时间	1-3天，默认为 3 天，可在集群维度调整		1-7天，默认为 3 天，可在 Topic 维度调整	
消息最大延时	7天	40天，特殊需求可以通过 工单咨询	40天，特殊需求可以通过 工单咨询	可定制，最长1年
消息大小	4 MB	4 MB	最大 4 MB，特殊需求可以通过 工单咨询	可定制
消费位点重置	默认 3天，和消息保留时间一致			

开源对比

最近更新时间：2025-01-25 11:40:32

TDMQ RocketMQ 版和开源版 RocketMQ 的对比如下：

功能大类	功能项	开源 RocketMQ	TDMQ RocketMQ 版
安全管控	收发消息 ACL 管理	默认 ACL 鉴权方式，鉴权方式单一。	支持生产消息和发送消息的角色细化，支持更加细粒度的权限细化。
	主子账号	不支持。	全面支持腾讯云主子账号，实现 CAM 主子账号及企业内跨账号和授权等服务。
	扩缩容	依赖自建运维团队，自动化、白屏化程度低。	客户无需关注底层机器大小，也无需扩缩容机器，只需要根据业务量购买相应的规格，并支持在不同规格间进行自由升降配。底层资源通过容器化实现了弹性扩缩容，自动化智能运维。
	高可用	需要自行部署高可用架构，增加运维难度。	腾讯云底层实现高可用方案，客户无需关心部署容灾架构。
迁移工具	迁移工具	不支持迁移。	同时支持两种迁移方式：自建元数据导入后改接入点迁移 和 支持无感迁移，白屏化操作，按照不同的 Topic 分阶段进行灰度迁移。
监控告警	资源大盘	简单支持，部分监控指标缺失。	核心指标观测、生产消费报表和细粒度监控，支持公网、集群、topic、group 以及 topic&group 粒度的指标下钻；集群粒度提供资源 Top 排行。
	报警管理	不支持。	消息积压、延迟等多项指标告警，云监控联动。
弹性能力	弹性 TPS	不支持，需要按需扩缩容机器，提高运维复杂度。	专业版和铂金版支持，在正常的流量规格外，支持开启临时的弹性流量支持。
	存储免配额限制	不支持，部署时需要指定磁盘大小，磁盘买大容易导致资源浪费，且云上购买的磁盘不支持缩容；磁盘买小需要频繁进行扩缩容，提高运维难度。	存储按实际使用量进行收费，0.0021 元/小时/G。
稳定性	分布式限流	不支持，存在因为流量过载导致宕机问题。	支持多种限流方式和策略（全局/本地，降级策略，多规则优先级等），能有效避免流量过大引起宕机。
	收发消息配比调整	不支持。	支持在集群维度调整调整整个集群的消息收发占比，并进行分别限流，资源利用更加合理。
消息生命周期	Topic 级别保存时间设置	不支持。	支持，根据 Topic 设置消息保留时间，进一步节约存储成本。
控制台功能优化	客户端堆栈查看	不支持。	支持，并支持堆栈内指定代码搜索。
	按 Tag 查询消息	不支持。	支持。

	查询重试/死信/延时消息。	不支持。	单独根据指定条件查询特定类型的消息。
兼容性	兼容历史低版本 SDK	不完全兼容，部分场景会出现报错。	完全兼容 4.1之后的所有低版本 SDK。

购买指南
计费概述

最近更新时间：2025-03-18 10:18:42

本文主要介绍 TDMQ RocketMQ 5.0 集群的计费组成和计费方式。

计费方式

消息队列 RocketMQ 版提供**包年包月（预付费）**和**按量计费（后付费）**两种计费方式。

- **包年包月**是一种需要先付费才能使用资源的计费方式，主要适用于稳定的业务运行场景。您需要根据实际业务量分析资源的使用需求，一次性支付一个月、多个月或多年的费用。
- **按量计费**是一种根据您所购买的资源规格的实际使用量计费的付费方式，主要适用于测试或者流量峰值不确定的场景。您可以先使用资源后再付费，费用按照小时整点结算。

计费项目

消息队列 RocketMQ 版以集群的形式售卖，计费项目组成如下：

计费项	计费方式	计费规则
存储费用	按量计费	消息队列 RocketMQ 版集群存储费用按照消息存储所占用的存储空间大小和存储时长计费。

计算规格	包年包月
	按量计费
		消息队列 RocketMQ 版集群以消息收发 TPS 作为计算能力，每种版本集群提供不同规格的 TPS，计算费用按照 TPS 规格大小和使用时长计费。TPS 的计算规则分为两个维度：消息类型和消息大小。
		• 消息类型：消息队列 RocketMQ 版有4种消息类型，普通消息、定时和延时消息、事务消息以及顺序消息，其中定时和延时消息、事务消息和顺序消息都为高级特性消息。
		  ○ 普通消息：发送或者消费1条普通消息，无论消息是否发送成功或者消费成功，消息 TPS 都计算为1次/秒。
		  ○ 高级特性消息：发送1条或者消费1条高级特性消息，消息 TPS 都计算为5次/秒。例如1个 Topic 发送2条事务消息1次，消费1条事务消息，则消息收发 TPS 为 2×5+1×5=15次/秒。
		• 消息大小：
		消息大小以4 KB为计量单位，不满4KB按4KB计算。例如，一条18 KB的消息请求，消息发送 TPS 为⌈18/4⌉=5次/秒。
		⌈ ⌉表示向上取整数。

弹性 TPS 费用	按量计费	消息队列 RocketMQ 版集群的每个规格都有 TPS 限制，对于 专业版 和 铂金版的集群，在购买一定的 TPS 规格后，超过计算规格限制的部分，按 TPS 次数进行计费。**弹性 TPS 适用于业务侧偶尔出现少量突发流量的场景，若您的业务流量经常超过计算规格限制，建议您升级集群规格。**
公网带宽费用	包年包月 按量计费	公网带宽开启后，计费方式默认和集群一致，即如果集群为包年包月，则公网带宽也按照包年包月计费。开启公网带宽后将根据使用的公网带宽时长收费，中途可以单独关闭。若未开通公网访问，则不会产生费用。

价格说明

存储费用

消息队列 RocketMQ 版集群存储费用按照消息存储所占用的存储空间大小和存储时长计费，计费方式为按量计费（后付费），计量单位为 “XX 元/GB/小时”，每小时出一次账单，不足1小时，按1小时计算。
存储费用 = 存储空间 × 使用时长 × 存储单价

存储收费类型	价格（元/GB/小时）
存储按小时收费（后付费）	0.0021

计算规格费用

消息队列 RocketMQ 版集群计算费用按照规格大小和使用时长收取费用。计费方式支持包年包月和按量计费两种。

- 包年包月：计量单位为“XX元/月”。
- 按量计费（后付费）：计量单位为 “XX 元/小时”，每小时计算一次费用，不足1小时，按1小时计算；费用结算按天进行，账单会在隔天进行推送并扣费。

计算规格费用=计算规格单价 × 使用时长。
每个计算规格均支持不同的 Topic 免费额度，超出免费额度的部分将收取一定费用，收费政策见下文的 “超规格外 Topic 收费规则” 部分。如您需要的Topic 数量超出了当前集群规格承载的上限，建议您通过 升配 来提升计算集群的规格。
Group 的数量和 Topic 的数量默认均为 10:1，您可以通过调整 Topic 的限额来调整 Group 的限额。

版本	TPS 规格	Topic 免费额度（个）	可增加的 Topic 数量上限（个）	地域：北京、广州、上海、南京、成都、重庆、清远		地域：中国香港、东京、新加坡、中国台北、曼谷、硅谷、雅加达、法兰克福、弗吉尼亚、圣保罗、首尔		地域：深圳金融、上海自动驾驶云	
				按量计费价格（元/小时）	包年包月价格（元/月）	按量计费价格（元/小时）	包年包月价格（元/月）	按量计费价格（元/小时）	包年包月价格（元/月）
体验版	500	50	99	0.8	384	1.04	499.2	1.28	614.4
基础版	1000	100	999	1.67	800	2.17	1040	2.67	1280
	2000	100	999	2.50	1200	3.25	1560	4.00	1920
	3000	100	999	3.33	1600	4.33	2080	5.33	2560
	4000	100	999	4.17	2000	5.42	2600	6.67	3200
	5000	100	999	5.00	2400	6.50	3120	8.00	3840
	6000	200	999	5.83	2800	7.58	3640	9.33	4480
	7000	200	999	6.67	3200	8.67	4160	10.67	5120
	8000	200	999	7.50	3600	9.75	4680	12.00	5760
	9000	200	999	8.33	4000	10.83	5200	13.33	6400
	10000	200	999	9.17	4400	11.92	5720	14.67	7040

专业版	4000	300	999	8.00	3840	10.40	4992	12.80	6144
	6000	300	999	11.38	5460	14.79	7098	18.21	8736
	8000	300	999	14.75	7080	19.18	9204	23.60	11328
	10000	300	999	18.13	8700	23.57	11310	29.01	13920
	15000	325	3000	23.13	11100	30.07	14430	37.01	17760
	20000	350	3000	27.29	13100	35.48	17030	43.66	20960
	25000	375	3000	31.46	15100	40.90	19630	50.34	24160
	30000	400	3000	35.63	17100	46.32	22230	57.01	27360
	35000	425	3000	39.79	19100	51.73	24830	63.66	30560
	40000	450	3000	43.96	21100	57.15	27430	70.34	33760
	45000	475	3000	48.13	23100	62.57	30030	77.01	36960
	50000	500	3000	52.29	25100	67.98	32630	83.66	40160
	55000	550	3000	53.59	25725	69.67	33442.5	85.74	41160
	60000	600	3000	57.50	27600	74.75	35880	92.00	44160
	65000	650	3000	61.41	29475	79.83	38317.5	98.26	47160
	70000	700	3000	65.31	31350	84.90	40755	104.50	50160
	75000	750	3000	69.22	33225	89.99	43192.5	110.75	53160
	80000	800	3000	73.13	35100	95.07	45630	117.01	56160
	85000	850	3000	77.03	36975	100.14	48067.5	123.25	59160
	90000	900	3000	80.94	38850	105.22	50505	129.50	62160
	95000	950	3000	84.84	40725	110.29	52942.5	135.74	65160
	100000	1000	3000	88.75	42600	115.38	55380	142.00	68160

铂金版（独占资源）	10000	1000	10000	32.29	15500	41.98	20150	51.66	24800
	20000	1000	10000	41.67	20000	54.17	26000	66.67	32000
	30000	1000	10000	51.04	24500	66.35	31850	81.66	39200
	40000	1000	10000	60.42	29000	78.55	37700	96.67	46400
	50000	1000	10000	69.79	33500	90.73	43550	111.66	53600
	60000	1100	20000	72.92	35000	94.80	45500	116.67	56000
	70000	1200	20000	81.25	39000	105.63	50700	130.00	62400
	80000	1300	20000	89.58	43000	116.45	55900	143.33	68800
	90000	1400	20000	97.92	47000	127.30	61100	156.67	75200
	100000	1500	20000	106.25	51000	138.13	66300	170.00	81600
	120000	1600	30000	122.92	59000	159.80	76700	196.67	94400
	140000	1700	30000	139.58	67000	181.45	87100	223.33	107200
	160000	1800	40000	152.08	73000	197.70	94900	243.33	116800
	180000	1900	40000	166.67	80000	216.67	104000	266.67	128000
	200000	2000	40000	181.25	87000	235.63	113100	290.00	139200
	250000	2500	40000	217.71	104500	283.02	135850	348.34	167200
	300000	3000	40000	254.17	122000	330.42	158600	406.67	195200
	350000	3500	40000	290.63	139500	377.82	181350	465.01	223200
	400000	4000	40000	327.08	157000	425.20	204100	523.33	251200
	450000	4500	40000	363.54	174500	472.60	226850	581.66	279200
	500000	5000	40000	400.00	192000	520.00	249600	640.00	307200

	600000	6000	99999	447.92	215000	582.30	279500	716.67	344000
	700000	7000	99999	515.63	247500	670.32	321750	825.01	396000
	800000	8000	99999	583.33	280000	758.33	364000	933.33	448000
	900000	9000	99999	651.04	312500	846.35	406250	1,041.66	500000
	1000000	10000	99999	718.75	345000	934.38	448500	1,150.00	552000

弹性 TPS 费用

仅专业版和铂金版的集群支持 “弹性 TPS” 功能并涉及这部分费用。
消息队列 RocketMQ 版集群弹性 TPS 计费方式为按量计费（后付费）。计量单位为 “XX 元/TPS/小时”，每小时出一次账单，不足1小时，按1小时计算。
每小时弹性 TPS 费用=1小时内的最大增量 TPS 值 × 弹性 TPS 单价。
举例说明：某个专业版集群原先的 TPS 规格是 6000，开启弹性后，额外加了 4000 弹性 TPS，此时总 TPS 变为 10000，如果一个小时内，总 TPS 最高到达了 9000，那么当时收费为单价 0.006 * 超出的 3000 TPS * 1小时 = 18 元。

版本	TPS 规格（次/秒）	弹性 TPS 上限（次/秒）	价格（元/TPS/小时，地域：北京、广州、上海、南京、成都、重庆、清远）	价格（元/TPS/小时，地域：中国台北、中国香港、东京、新加坡、曼谷、硅谷、雅加达、法兰克福、弗吉尼亚、圣保罗、首尔）	价格（元/TPS/小时，地域：深圳金融、上海自动驾驶云）
体验版	不支持弹性 TPS 能力	不涉及			
基础版	不支持弹性 TPS 能力				
专业版	4000	2500	0.006	0.0078	0.0096
	6000	4000			
	8000	5000			
	10000	6000			
	15000	9000			
	20000	12000			
	25000	13000			

	30000	15000			
	35000	18000			
	40000	20000			
	45000	22000			
	50000	25000			
	55000				
	60000				
	65000				
	70000				
	75000	30000			
	80000				
	85000				
	90000				
	95000	35000			
	100000	40000			
铂金版	10000	6000	0.012	0.0156	0.0192
	20000	12000			
	30000	18000			
	40000	25000			
	50000	30000			
	60000				
	70000				
	80000	40000			
	90000				
	100000				
	120000				
	140000	50000			
	160000	60000			
	180000				

	200000	80000			
	250000				
	300000	100000			
	350000				
	400000	150000			
	450000				
	500000	200000			
	600000	250000			
	700000				
	800000	300000			
	900000				
	1000000	350000			

计费示例：

假设您的集群属于广州地域，集群类型为专业版，购买的基础消息收发 TPS 规格为10000次/秒，则弹性 TPS 上限为6000次/秒，弹性 TPS 单价为0.006元/TPS/秒。假设在14:00-15:00这一小时内，该集群消息收发情况如下：

- 14:00，集群使用 TPS 峰值为9000次/秒，未使用弹性 TPS。
- 14:01，集群使用 TPS 峰值为9500次/秒，未使用弹性 TPS。
- 14:02，集群使用 TPS 峰值为10500次/秒，则使用弹性 TPS 500次/秒。
- ……，集群使用 TPS 峰值均保持在8000-9000次/秒之间，未使用弹性 TPS。
- 14:58，集群使用 TPS 峰值为12000次/秒，则使用弹性 TPS 2000次/秒。
- 14:59，集群使用 TPS 峰值为11000次/秒，则使用弹性 TPS 1000次/秒。

则这一小时内，取1小时内的最大增量 TPS 值，为2000次/秒，则该小时内产生的弹性 TPS 费用为 2000 × 0.006 = 12 元。

公网带宽费用

公网带宽费用有**包年包月**，**按小时计费**和**按流量计费**共三种计费模式。

> **说明：**
> 为了方便客户更好地进行资源生命周期管理，如果集群的计算规格为包年包月，则公网支持包年包月和按流量计费两种方式；如果集群的计算规格为按小时收费，则公网支持按小时和按流量计费两种方式。

包年包月

带宽范围	价格（元/月）				
	北京、广州、深圳金融、上海、南京、成都、重庆、清远	中国香港、中国台北、硅谷、弗吉尼亚、圣保罗	新加坡、法兰克福	曼谷、东京、首尔	上海自动驾驶云

1Mbps	23	30	23	25	34.5
2Mbps	46	60	46	50	69
3Mbps	71	90	69	75	106.5
4Mbps	96	120	92	100	144
5Mbps	125	150	115	125	187.5
6Mbps及以上（n 为设置的带宽上限）	125 + 80 ×（n − 5）	150 + 100 ×（n − 5）	115 + 80 ×（n − 5）	125 + 80 ×（n − 5）	187.5 + 120 ×（n − 5）

按小时计费

按公网传输速率（单位为 Mbps）计费，每小时结算一次，结算时按实际使用小时数计费。

带宽范围	价格（元/小时）			
	北京、广州、上海、南京、成都、重庆、清远、深圳金融、中国香港、中国台北、东京、曼谷、法兰克福	新加坡、硅谷、弗吉尼亚	上海自动驾驶云	圣保罗、雅加达
1–5Mbps	0.04	0.035	0.06	0.063
6Mbps及以上（n 为设置的带宽上限）	0.14	0.12	0.21	0.25

按流量计费

说明：

- 当前计费流量是出流量，即从负载均衡到公网的流量。
- 为了防止因突然爆发的流量而产生较高的费用，您可以通过指定带宽上限进行限制。若超出此上限，则默认丢包且不计算费用。
- 流量的单位换算进制为1024，即：1TB = 1024GB，1GB = 1024MB。

地域	价格（单位：元/GB）
广州/上海/南京/北京/成都/重庆	1.0
深圳金融/上海金融/北京金融	1.6
上海自动驾驶云	1.2
中国香港/圣保罗/新加坡/雅加达/东京/法兰克福/首尔/曼谷/硅谷/弗吉尼亚	1.3

超规格外 Topic 收费规则

考虑到集群的稳定性和实际使用场景，不同 TPS 规格的集群的 Topic 个数上限有所区别。客户可以在页面上通过升配自助添加 Topic 数量的限额，超出免费限额的部分将按照阶梯收取一定费用。

> **说明：**
> 此处的 Topic 数量仅包含控制台创建的 Topic 数量，死信队列和重试队列对应的 Topic 不占用配额。

包年包月

超规格 Topic 数量阶梯	价格（地域：北京、广州、上海、南京、成都、重庆）	价格（地域：中国香港、新加坡、曼谷、硅谷）	价格（地域：深圳金融、上海自动驾驶云）
0-100	12 元/个/月	15.6 元/个/月	19.2 元/个/月
101-200	10 元/个/月	13 元/个/月	16 元/个/月
201-500	8 元/个/月	10.4 元/个/月	12.8 元/个/月
501-1500	6 元/个/月	7.8 元/个/月	9.6 元/个/月
1501-2000	4 元/个/月	5.2 元/个/月	6.4 元/个/月
2000 以上	2 元/个/月	2.6 元/个/月	3.2 元/个/月

计费示例：
客户 A 购买了一个 5.x 专业版 8000 TPS 的包月预付费集群，用户需要 800 个 Topic。专业版 8000 TPS 的集群默认支持 300 个免费的 Topic 额度，因此超额的 Topic 数量为 500。对照上方的阶梯计费，则额外的 Topic 收费为：12*100+10*100+8*(500-200) =4400 元/月。

按量计费

超规格 Topic 数量阶梯	价格（地域：北京、广州、上海、南京、成都、重庆）	价格（地域：中国香港、新加坡、曼谷、硅谷）	价格（地域：深圳金融、上海自动驾驶云）
0-100	0.025 元/个小时	0.0325 元/个小时	0.04 元/个小时
101-200	0.02 元/个小时	0.026 元/个小时	0.032 元/个小时
201-500	0.016 元/个小时	0.0208 元/个小时	0.0256 元/个小时
501-1500	0.0125 元/个小时	0.01625 元/个小时	0.02 元/个小时
1501-2000	0.008 元/个小时	0.0104 元/个小时	0.0128 元/个小时
2000 以上	0.004 元/个小时	0.0052 元/个小时	0.0064 元/个小时

产品系列

最近更新时间：2024-07-31 16:32:01

消息队列 RocketMQ 版提供不同的版本规格，以满足您不同业务场景与规模的需求，各版本的功能对比差异如下：

项目	体验版	基础版	专业版	铂金版
TPS 规格（次/秒）	500	1000-10000	4000-100000	10000-1000000
消息轨迹	支持			
读写流量配比	可调整			
消息保存时长设置	1-3 天		1-7 天	
消息保存粒度设置	集群粒度		Topic 粒度	
SQL 过滤	支持			
消息重试策略	默认		可定制	
弹性 TPS	不支持		支持	
定时消息时长	7天	40天	40天	可定制，最长1年
消息大小	4MB			可定制，20MB
服务可用性	不保证 SLA	99.95%	99.95%	99.99%
部署架构	资源共享	资源共享，默认跨可用区部署	资源共享，默认跨可用区部署	资源独占，默认跨可用区部署
产品系列转换	可以互相转换升级			不可转换成其他规格
服务支持	7*24小时工单服务			专业服务支持，培训交流，重要活动护航服务，架构咨询和建议

购买方式

最近更新时间：2025-01-03 14:38:02

消息队列 RocketMQ 版集群当前提供**包年包月**和**按量计费**的计费方式，您可按照以下步骤购买服务：

1. 登录 腾讯云 RocketMQ 控制台 。
2. 在左侧导航栏选择 **集群列表**，单击**新建集群**，进入购买页。
3. 在购买页面，选择地域、可用区、集群类型型号、集群规格等信息。
4. 信息填写完成后，单击**立即购买**，根据系统提示步骤完成付款，即购买成功。

退费说明

最近更新时间：2025-01-03 14:38:02

按量计费

按量计费模式下的集群可随时销毁集群，销毁完成后计费将终止。

包年包月

5天无理由自助退还

退款说明

TDMQ RocketMQ 版包年包月模式下的实例产品遵守腾讯云 云服务退货说明，若您在购买集群后有任何不满意，我们支持五天内无理由自助退还，具体规则如下：

- 对于每个主体，自新购实例之日起5天内（含5天），可享受**1**个实例5天无理由退还。退款金额为**购买时花费的全部消耗金额**，包括现金账户金额、收益转入账户金额以及赠送账户金额。
- 如出现疑似异常/恶意退货，腾讯云有权拒绝您的退货申请。

退款金额及途径

购买时使用的代金券不支持退还，购买时使用的非代金券费用按支付方式（现金/赠送金）及支付比例退还到支付方腾讯云账户。

非全额退款

不满足五天内无理由全额退款规则的预付费云产品退货退款，退款说明如下：

退款说明

- 单价和折扣按照系统当前的优惠为准。
- 参与活动购买的产品，如若退款规则与活动规则冲突，以活动规则为准，活动规则中若说明不支持退款，则无法申请退款。
- 推广奖励渠道订单暂不支持自助退款，请提交工单发起退款申请。
- 如出现疑似异常/恶意退货，腾讯云有权否决退货申请。

退款金额及途径

退款金额 = 订单实付金额 – 资源已消耗金额
计算方式基于**使用时长**计算：
已消耗金额 = (已使用时长 ÷ 总时长) × 订单原价 × 当前折扣

> **说明：**
> 使用时长不足1天按1天计算，当前折扣根据已使用时长匹配系统当前折扣。

欠费说明

最近更新时间：2024-10-18 16:32:31

注意事项

- 集群不再使用时请及时销毁，以免继续扣费。
- 集群被销毁/回收后，数据将会被清除且不可找回。

按量计费模式

按量付费的模式下，计费周期为“日”，即计费系统在下一个自然日就您上一个自然日的服务使用进行计量和出具账单，并在您的账户中按账单金额扣除服务费用。
如当前账户余额不足，但是当前的用量在免费额度内，可以继续使用；
如账号金额不足且无欠费不停机特权时，消息队列 RocketMQ 版在24小时内可继续使用且继续扣费，24小时后消息队列 RocketMQ 版将停止服务，无法正常收发消息、控制台和云 API 无法正常调用，并继续产生资源占用费用。
停止服务后，系统对消息队列 RocketMQ 版进行如下处理：

停止服务后的时间	说明
≤ 7天	若充值至余额大于0，计费将继续，用户可重启消息队列 RocketMQ 版。
	若您的账户余额尚未充值到大于0，则无法重启消息队列 RocketMQ 版。
> 7天	若您的账户余额未充值到大于0，按量计费的消息队列 RocketMQ 版资源将被销毁，所有数据将被清理，且不可找回。消息队列 RocketMQ 版资源被销毁时，我们将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

包年包月模式

消息队列 RocketMQ 专享版以包年包月的模式计算费用。

到期预警

包年包月实例在到期前7天开始，系统会自动每隔1天向您推送到期预警消息。预警消息将通过**邮件及短信**的方式通知到腾讯云账户的创建者以及所有协作者。

欠费提醒

包年包月实例到期当天及以后，每隔1天向您推送欠费隔离预警消息。预警消息将通过**邮件及短信**的方式通知到腾讯云账户的创建者以及所有协作者。

欠费处理

- 账户余额充足的情况下，若您已设置了自动续费，设备到期当日会执行自动续费。
- 在您的集群实例到期后7天内，集群禁写不禁读，即已经发送到服务端的消息可以正常消费，但是无法发送新的消息。若您在此期间进行续费，**被续费的集群实例续费周期的起始时间为上一个周期的到期日。**
- 若您的集群实例在到期后7天内未进行续费，集群将会停止服务。届时相关集群资源将被销毁，所有数据将被清理，且不可找回。集群被销毁时，我们将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

注意：

- 请在收到欠费通知后，及时前往控制台 充值中心 进行充值，以免影响您的业务。
- 如您对消费明细有疑问，可以在控制台 资源账单 页面查阅和核对您的消费明细。
- 如您对具体的扣费项有疑问，可以参见 价格说明 了解具体的计费项含义及计费规则。
- 您还可以通过计费中心的余额告警功能，自行设定欠费预警。详细信息请参见 余额预警指引 。

快速入门
资源创建与准备

最近更新时间：2025-02-03 14:02:01

操作场景

在使 SDK 收发消息前，您需要在消息队列 RocketMQ 控制台中创建集群、Topic 等资源，运行客户端时需要配置相关的资源信息。

前提条件

已 注册腾讯云账号 。

操作步骤

步骤1. 新建集群

1. 登录 RocketMQ 控制台 ，进入**集群管理**页面，选择好目标地域。
2. 单击**新建集群**，选择好集群规格，单击**立即购买**，创建一个集群。

3. 在集群列表页面，单击创建好的集群ID，在集群基本信息页面的**接入信息**模块，可以查看到集群的接入点信息。

步骤2. 配置集群权限

1. 单击 步骤1 中创建好的集群的“ID”，进入集群基本信息页面。
2. 在页面上方选择集群权限页签，单击**添加角色**，创建一个角色并为其配置生产和消费权限。

> **说明：**
> 5.x 集群支持在 集群管理 页面关闭 ACL 权限开关。但是为了保证客户的集群安全，在关闭 ACL 开关前会首先校验客户的公网开关是否打开，如果已经打开，需要再关闭公网开关后再关闭 ACL 权限。

步骤3. 创建 Topic

1. 在集群详情页面，选择 Topic 页签，进入 Topic 列表页。
2. 单击**新建**，填写好 Topic 名称，消息类型选择**普通消息**，单击**提交**，创建一个 Topic。

> **说明：**
> 本文以收发普通消息为例进行说明，因此，您参考上述步骤创建的普通消息的Topic，不能用于收发其他类型的消息。

步骤4. 新建 Group

1. 在集群详情页面，选择 **Group** 页签，进入 Group 列表页。
2. 单击**新建**，填写好 Group 名称，其他选项保持默认即可，单击**提交**，创建一个 Group。


基本信息
集群监控
Topic
Group
集群权限
新建 (1 / 500)
请输入关键字进行搜索
Group 名称
监控
消费者信息
最大重试次数
投递顺序性
说明
操作
group1
在线消费者 0 TPS 0 总堆积 0
16
并发投递
重置 offset 编辑 删除
共 1 条
20 条 / 页
1
/ 1 页

使用 5.0 SDK 收发普通消息

最近更新时间：2024-10-14 09:51:41

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **说明**
> 以 Java 客户端为例说明，其他语言客户端请参见 SDK 文档 。

前提条件

- 完成资源创建与准备
- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 Maven 工程为例，在 pom.xml 添加以下依赖：

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.6</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.7</version>
        </dependency>
</dependencies>
```

步骤2. 生产消息

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import org.apache.rocketmq.client.apis.ClientConfiguration;
import org.apache.rocketmq.client.apis.ClientException;
import org.apache.rocketmq.client.apis.ClientServiceProvider;
import org.apache.rocketmq.client.apis.SessionCredentialsProvider;
```

```
import org.apache.rocketmq.client.apis.StaticSessionCredentialsProvider;
import org.apache.rocketmq.client.apis.message.Message;
import org.apache.rocketmq.client.apis.producer.Producer;
import org.apache.rocketmq.client.apis.producer.SendReceipt;
import org.apache.rocketmq.client.java.example.AsyncProducerExample;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class NormalMessageSyncProducer {
    private static final Logger log =
LoggerFactory.getLogger(NormalMessageSyncProducer.class);

    private NormalMessageSyncProducer() {
    }

    public static void main(String[] args) throws ClientException, IOException {
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        // 添加配置的ak和sk
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // 填写腾讯云提供的接入地址
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String topic = "yourNormalTopic";
        // 通常在一个客户端内无需创建过多的生产者。
        final Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            // 设置主题名，此处的设置非必须，但是推荐设置，以便生产者可以在正式发送消息前，预先抓
取消息路由。
            .setTopics(topic)
            // 如生产者未初始化可能会报 M {@link ClientException} 的错误。
            .build();
        // 此处定义消息主体。
        byte[] body = "This is a normal message for Apache
RocketMQ".getBytes(StandardCharsets.UTF_8);
        String tag = "yourMessageTagA";
        final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // 在 topic 下进行的消息二级分类，区别同一个主题内不同的消息。
            .setTag(tag)
            // 消息键，除消息 ID 外可以区别不同消息的其他途径。
            .setKeys("yourMessageKey-1c151062f96e")
```

```
            .setBody(body)
            .build();
        try {
            final SendReceipt sendReceipt = producer.send(message);
            log.info("Send message successfully, messageId={}",
sendReceipt.getMessageId());
        } catch (Throwable t) {
            log.error("Failed to send message", t);
        }
        // 发送完成后，如无别的需要可以关闭生产者客户端。
        producer.close();
    }
}
```

步骤3. 消费消息

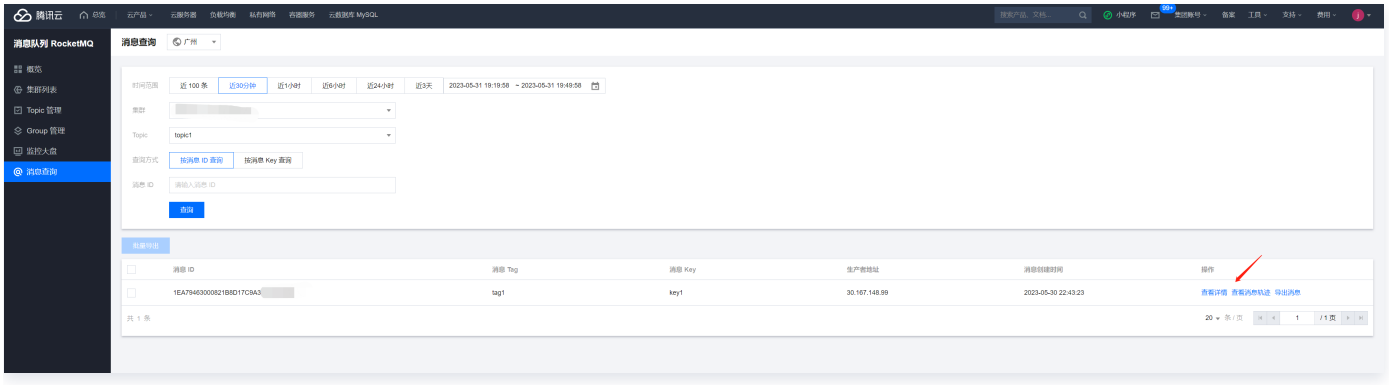
腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种消费模式，分别为 Push Consumer 和 Simple Consumer，以下代码示例以 Push Consumer 为例。

```
import java.io.IOException;
import java.util.Collections;
import org.apache.rocketmq.client.apis.ClientConfiguration;
import org.apache.rocketmq.client.apis.ClientException;
import org.apache.rocketmq.client.apis.ClientServiceProvider;
import org.apache.rocketmq.client.apis.SessionCredentialsProvider;
import org.apache.rocketmq.client.apis.StaticSessionCredentialsProvider;
import org.apache.rocketmq.client.apis.consumer.ConsumeResult;
import org.apache.rocketmq.client.apis.consumer.FilterExpression;
import org.apache.rocketmq.client.apis.consumer.FilterExpressionType;
import org.apache.rocketmq.client.apis.consumer.PushConsumer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class NormalPushConsumer {
    private static final Logger log =
LoggerFactory.getLogger(NormalPushConsumer.class);

    private NormalPushConsumer() {
    }

    public static void main(String[] args) throws ClientException, IOException,
InterruptedException {
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        // 添加配置的 ak 和 sk
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);
```

```
        // 填写腾讯云提供的接入地址
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        // 通常在一个客户端内无需创建过多的消费者。
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // 设置消费者组名称。
            .setConsumerGroup(consumerGroup)
            // 设置消费者订阅名称
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .setMessageListener(messageView -> {
                // 处理消息并返回消息消费结果。
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
        // 生产环境无需阻塞主线程。
        Thread.sleep(Long.MAX_VALUE);
        // 消费完成后，如无别的需要可以关闭消费者客户端。
        pushConsumer.close();
    }
}
```

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情见 消息查询 章节。



使用 4.0 SDK 收发普通消息

最近更新时间：2024-10-14 15:23:01

操作场景

本文以调用 4.0 Java SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **说明**
> 以 Java 客户端为例说明，其他语言客户端请参见 SDK 文档 。

前提条件

- 完成资源创建与准备
- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 Maven 工程为例，在 pom.xml 添加以下依赖：

```
<!-- in your <dependencies> block -->
  <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>
```

步骤2. 生产消息

```
// 实例化消息生产者Producer
   DefaultMQProducer producer = new DefaultMQProducer(
       groupName,
       new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL权限
   );
   // 设置NameServer的地址,地址就是形如xxx.tencenttdmq.com:8080 这样的接入地址。
   producer.setNamesrvAddr(nameserver);
   // 启动Producer实例
   producer.start();
```

```
    for (int i = 0; i < 10; i++) {
            // 创建消息实例，设置topic和消息内容.
            Message msg = new Message(topic_name, ("Hello RocketMQ " +
i).getBytes(RemotingHelper.DEFAULT_CHARSET));
            // 发送消息
            SendResult sendResult = producer.send(msg);
            System.out.printf("%s%n", sendResult);
    }
```

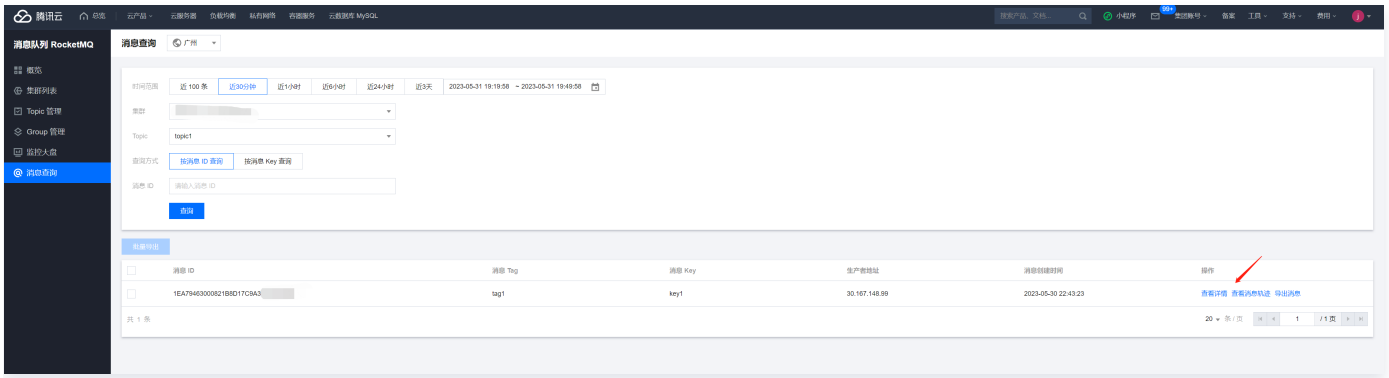
步骤3. 消费消息

以下代码示例以 Push Consumer 为例，其他的可以参考更详细 4.x 的使用文档。

```
// 实例化消费者
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)));
//ACL权限
        // 设置NameServer的地址
pushConsumer.setNamesrvAddr(nameserver);
        // 订阅topic
pushConsumer.subscribe(topic_name, "*");
      // 注册回调实现类来处理从broker拉取回来的消息
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context) -
> {
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
      });
      // 启动消费者实例
pushConsumer.start();
```

步骤4. 查看消息详情

发送完成消息后会得到一个消息ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情参见 消息查询 。



运行 RocketMQ 客户端（可选）

最近更新时间：2024-11-26 15:35:52

操作场景

该任务指导您在购买 RocketMQ 服务后，使用 RocketMQ API。在腾讯云服务器上搭建 RocketMQ 环境后，本地下载并解压 RocketMQ 工具包，并对 RocketMQ API 进行简单测试。

操作步骤

步骤1：安装 JDK 环境

1. 检查 Java 安装。

打开终端，执行如下命令：

```
java -version
```

如果输出 Java 版本号，说明 Java 安装成功；如果没有安装 Java，请 下载安装 Java 软件开发套件（JDK）。

2. 设置 Java 环境。

设置 `JAVA_HOME` 环境变量，并指向您机器上的 Java 安装目录。
以 Java JDK 1.8.0_20 版本为例，操作系统的输出如下：

操作系统	输出
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdkjdk1.8.0_20
Linux	export JAVA_HOME=/usr/local/java-current
Mac OSX	export JAVA_HOME=/Library/Java/Home

将 Java 编译器地址添加到系统路径中：

操作系统	输出
Windows	将字符串“;C:\Program Files\Java\jdk1.8.0_20\bin”添加到系统变量“Path”的末尾
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac OSX	not required

使用上面提到的 `java -version` 命令验证 Java 安装。

步骤2：下载 RocketMQ 工具包

下载并解压 RocketMQ 安装包。（ RocketMQ 安装包官网下载地址 ）

步骤3：RocketMQ API 测试

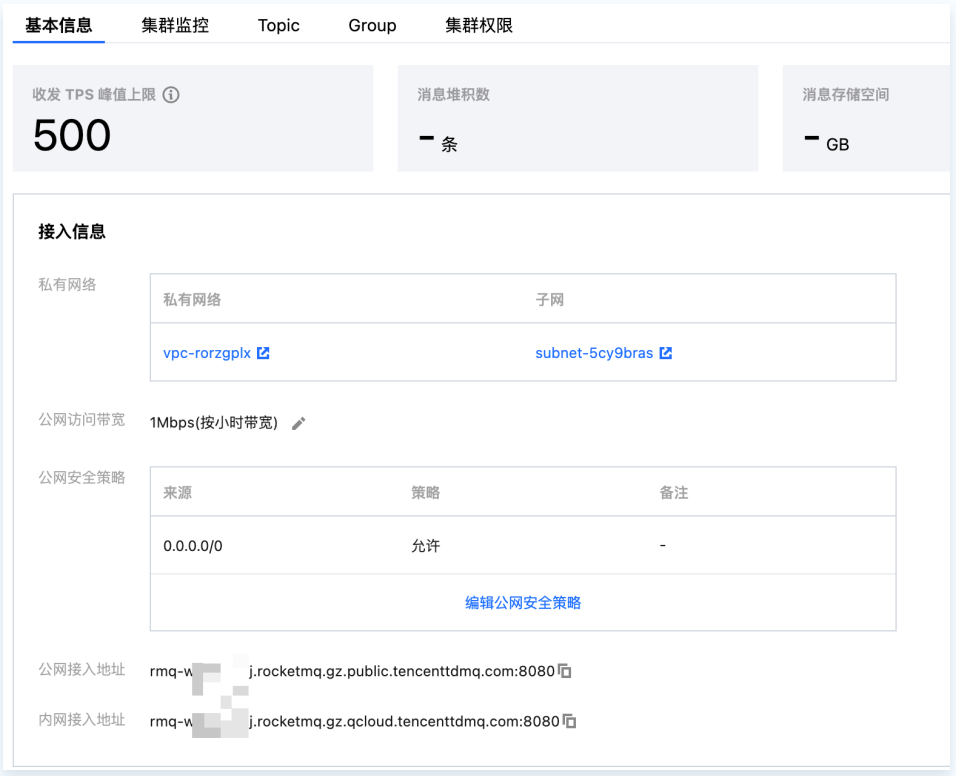
通过 CLI 命令生产和消费消息，去到 `./benchmark` 目录下。单条消息发送，可以使用云上管控台的发送功能。

1. 打开终端启动消费者。

```
sh consumer.sh -t "topic的名称" -n "rocketmq接入地址" -g "group的名称" -a true -ak xxx
-sk yyy
```

说明：

- 将 **rocketmq接入点** 替换成 VPC 网络访问的域名与端口，如果在本地上测试，不能联通 VPC，可以替换成公网的（注意放开对应的安全组），在控制台实例详情页面的**接入方式**模块获取。



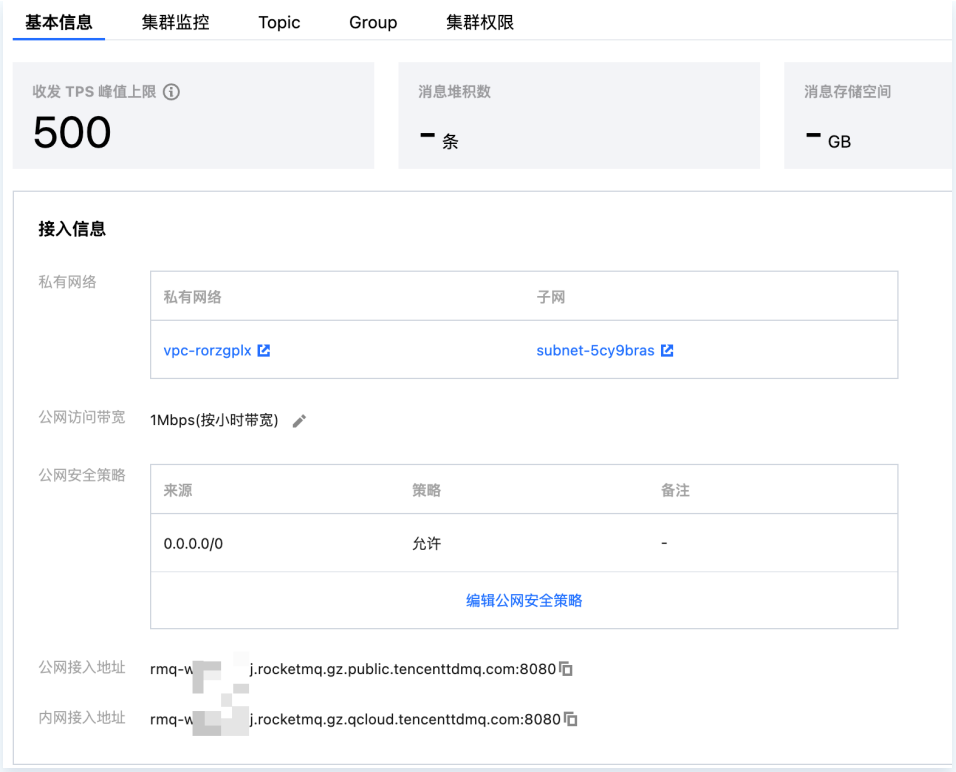
- topic：在控制台 **topic 管理** 页面获取的 topic 的名称。
- group: 在控制台 **group 管理** 页面获取的 group 的名称。

2. 另外开一个终端窗口启动生产者。

```
sh producer.sh -t "topic的名称" -w 1 -s 1024 -n "rocketmq接入地址" -a true -ak xxx -sk
yyy -m true
```

说明：

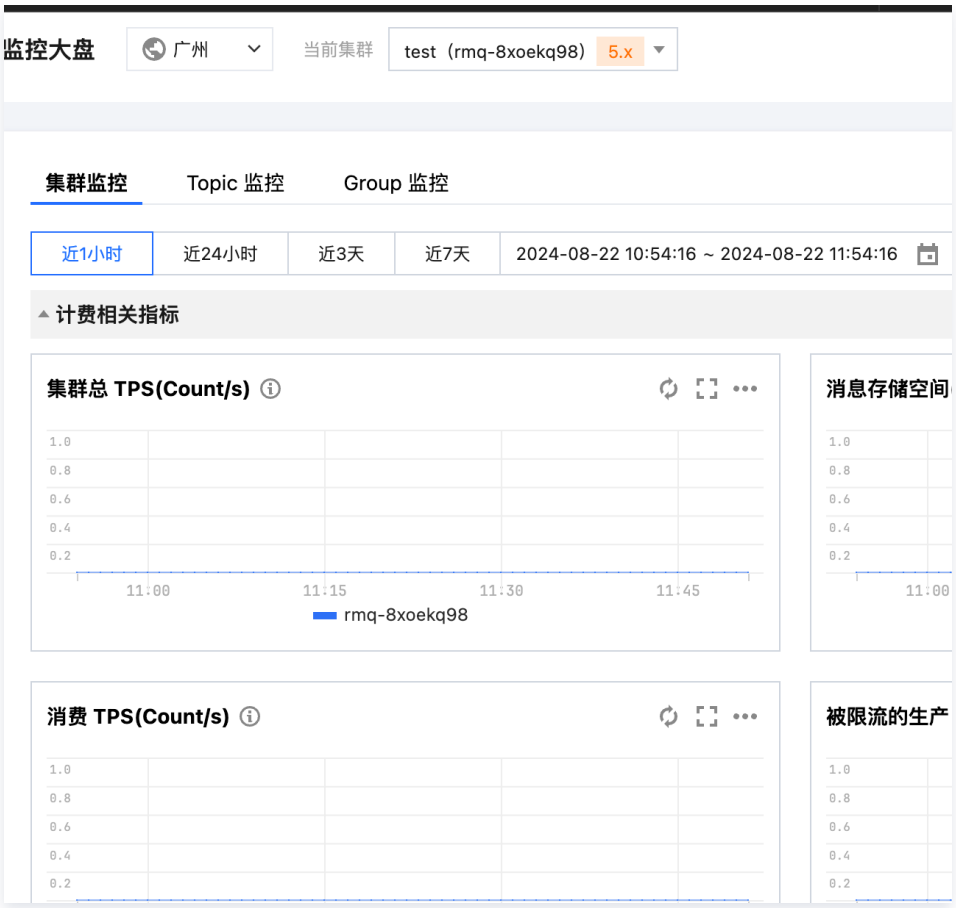
- 将 **rocketmq接入点** 替换成 VPC 网络访问的域名与端口，如果在本地上测试，不能联通 VPC，可以替换成公网的（注意放开对应的安全组），在控制台实例详情页面的**接入方式**模块获取。



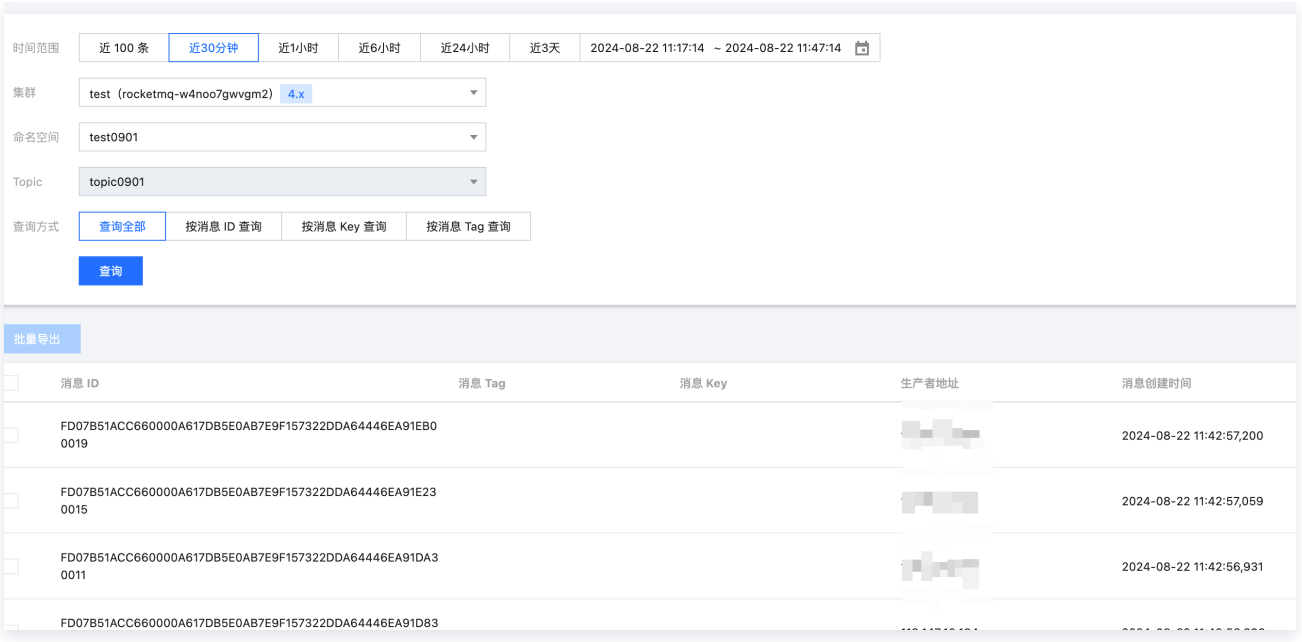
- topic：将 XXXX 替换成 topic 名称，在控制台 **topic 管理**页面获取。

启动后即可看到生产消费正常生产：

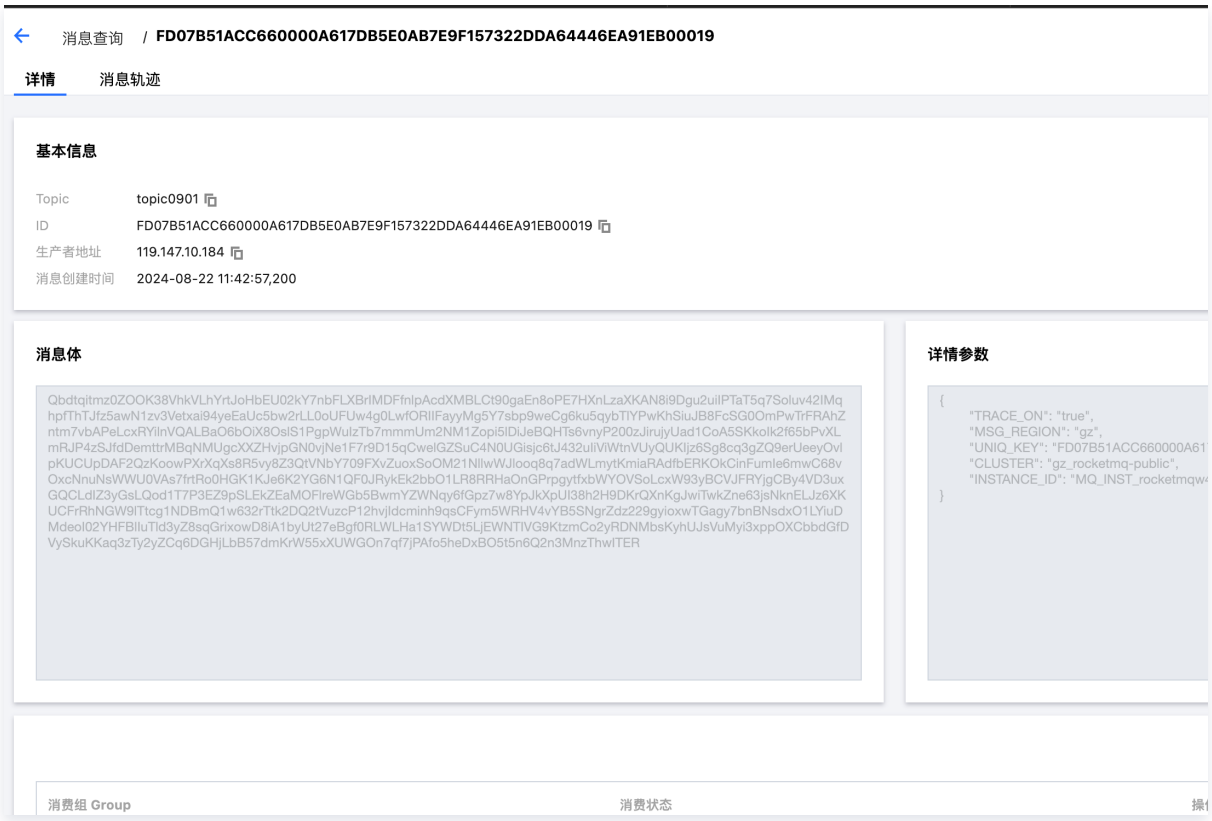
也可以在监控大盘查看监控指标：



3. 在 RocketMQ 控制台消息查询页面，查询刚刚发送的消息内容。



消息详情如下：


消息查询 / FD07B51ACC660000A617DB5E0AB7E9F157322DDA64446EA91EB00019
详情
消息轨迹
基本信息
Topic topic0901
ID FD07B51ACC660000A617DB5E0AB7E9F157322DDA64446EA91EB00019
生产者地址 119.147.10.184
消息创建时间 2024-08-22 11:42:57,200
消息体
详情参数
消费组 Group
消费状态

控制台指南
集群管理
新建集群

最近更新时间：2024-10-14 15:28:31

操作场景

集群是 TDMQ RocketMQ 版中的一个资源维度，不同集群的Topic、Group 等资源完全隔离。每个集群会有集群的资源限制例如 Topic 总数、消息保留时长等。常见的使用方式如：开发测试环境使用一个专门集群，生产环境使用一个专门的集群。

操作步骤

创建集群

1. 登录 RocketMQ 控制台，进入**集群列表**页面。
2. 在**集群列表**页面，选择地域后，单击**新建集群**进入购买页面，设置相关信息。

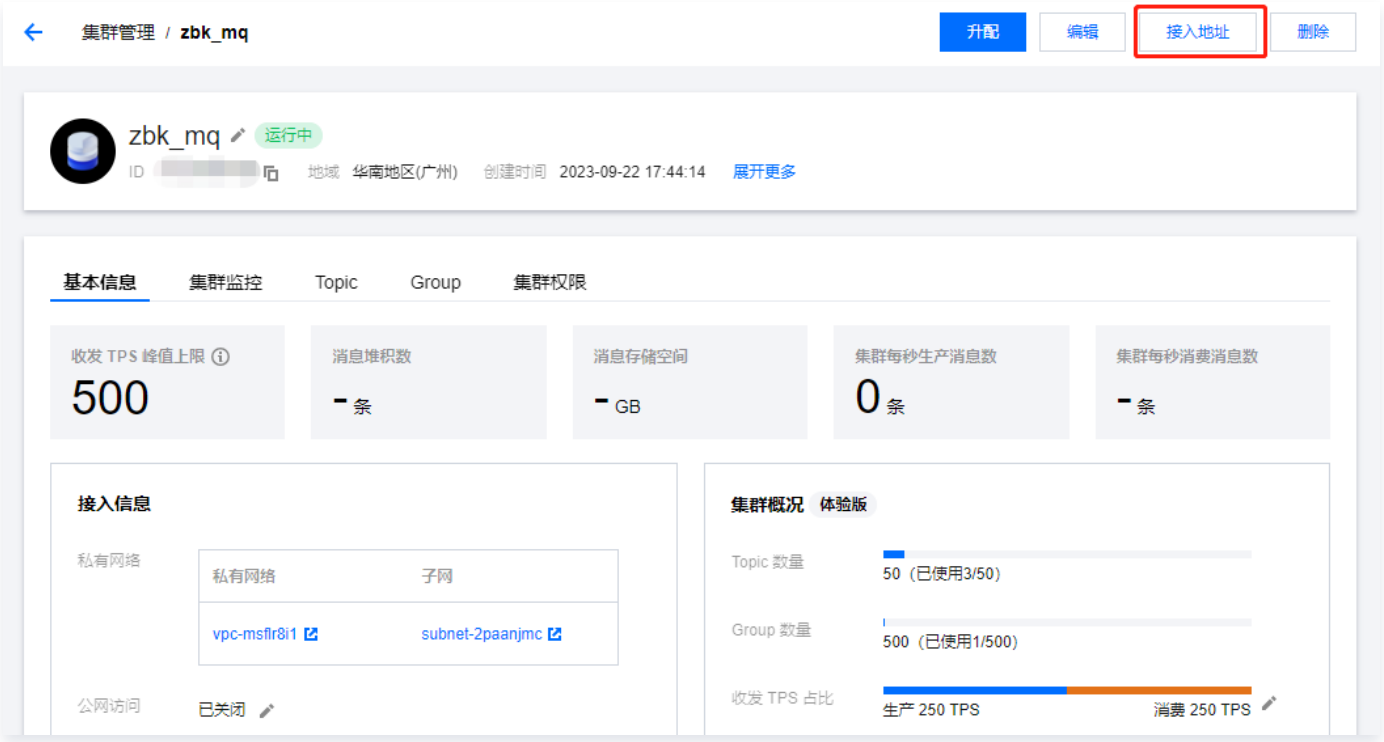
参数	是否必选	说明
集群版本	是	选择5.x
计费模式	是	5.x 架构集群当前支持按量计费和包年包月两种模式。
地域	是	选择与您的业务最靠近的地域，处于不同地域的云产品内网不通，购买后不能更换，请您谨慎选择。例如，广州地域的云服务器无法通过内网访问上海地域的集群。若需要跨地域内网通信，请参见 对等连接 。
集群规格	是	当前支持体验版、标准版、专业版和铂金版，不同集群类型在性能规格和功能上有差异，详见 产品系列 。 • 体验版：适用于在开发初期体验产品功能的企业客户和个人开发者；仅供测试和体验使用，不建议在生产环境使用。 • 基础版：适用于当前规模适中的客户，提供兼容开源社区 RocketMQ 的消息收发等基础能力。 • 专业版：适用于业务规模较大，但是对物理资源隔离性没有特殊要求的企业级客户，支持底层资源共享，支持的TPS规格最高可达15万 TPS。 • 铂金版：适用于对于资源有物理隔离需求的大型企业客户和大规模业务场景，是唯一一个底层资源独占的规格类型，支持的 TPS 规格跨度最大，最高可支持百万 TPS。
TPS 规格	是	TPS 规格包含生产消息和消费消息的总和；单条消息以 4KB 为单位对消息进行折算，特殊类型的消息按照特定比例进行折算，详细规则见 计费概述 。 • 体验版：支持500。 • 基础版：支持1000、2000、4000、6000。 • 专业版：支持4000 - 15万 TPS。 • 铂金版：支持6000 - 100万 TPS.
消息保留时间	是	时间消息保留的时间范围为24-72小时，超过消息保留时间后，无论消息是否已被消费，都会被删除。

私有网络	是	授权将新购集群接入点域名绑定至私有网络（VPC）。
公网访问	否	开启公网带宽后会新增单独的费用，开通后可在集群管理页关闭， 详情参考 公网计费说明 。
集群名称	是	填写集群名称， 3-64个字符，只能包含数字、字母、 “-” 和 “_” 。
标签	否	标签用于从不同维度对资源分类管理。使用方法请参见 使用标签管理资源 。
协议条款	是	勾选我已阅读并同意《消息队列 RocketMQ 版服务条款》。

3. 单击**立即购买**，等待3-5分钟后，完成集群创建。

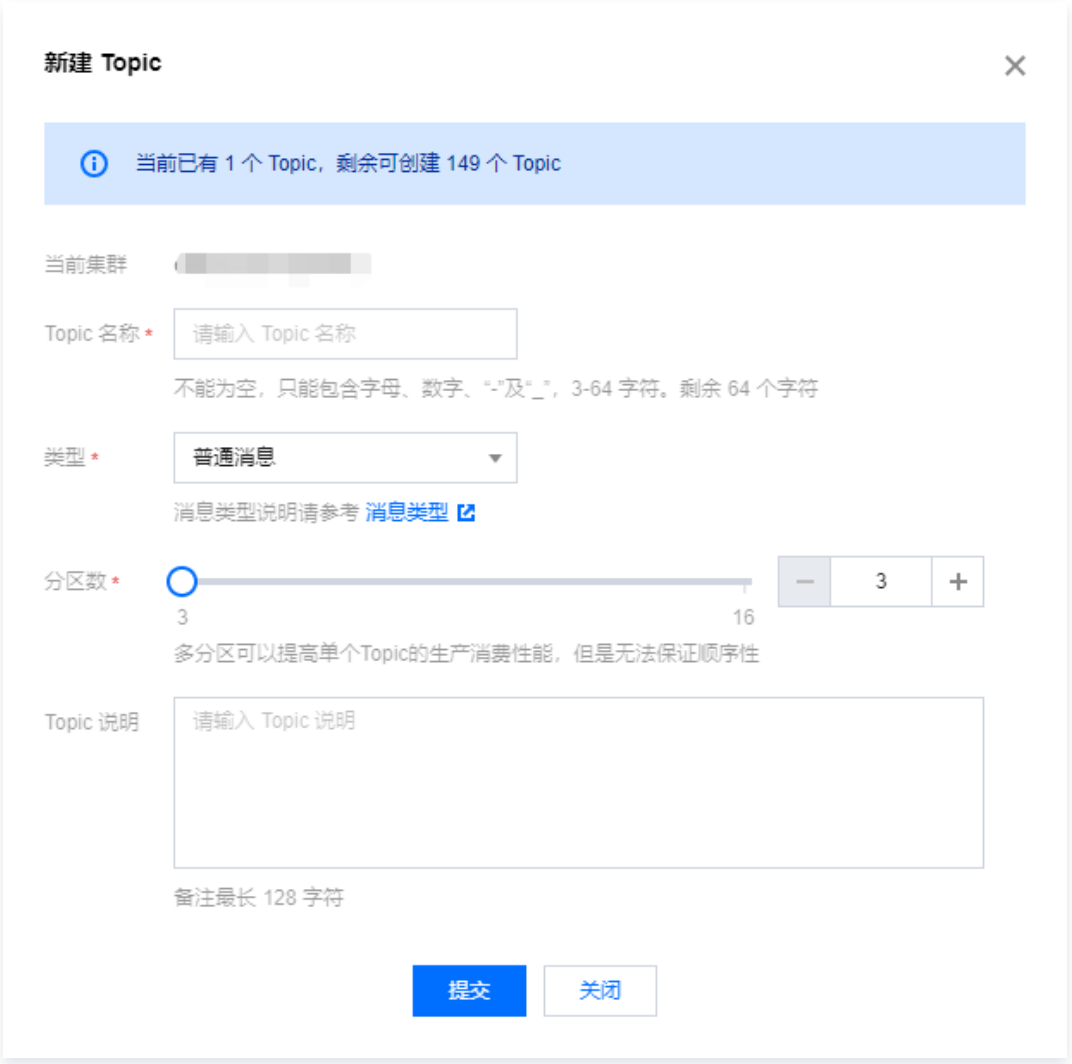
后续步骤：

1. 获取接入地址，得到服务端的连接信息。



2. 在集群中创建 ***角色*** 并授予该集群的生产消费权限。

3. 在集群中***创建 Topic 和 Group***。


新建 Topic
当前已有 1 个 Topic，剩余可创建 149 个 Topic
当前集群
Topic 名称 *
请输入 Topic 名称
不能为空，只能包含字母、数字、"-"及"_"，3-64 字符。剩余 64 个字符
类型 *
普通消息
消息类型说明请参考 消息类型
分区数 *
3
16
3
多分区可以提高单个Topic的生产消费性能，但是无法保证顺序性
Topic 说明
请输入 Topic 说明
备注最长 128 字符
提交
关闭

4. 按照 ***SDK 文档*** 的提示编写 Demo，配置上连接信息，进行消息的生产和消费。

查看集群详情

在**集群列表**页，单击集群的 ID，进入集群详情页面。在详情页中，您可以查询到：

- 集群的基础信息（集群名称/ID、地域、创建时间、说明、资源标签）
- 集群数据统计：展示所选时间范围内当前集群的消息消费速率、消息生产速率、消息堆积数、集群每秒生产流量和集群每秒消费流量。
- 接入信息：展示内网和公网接入点信息。
- 集群概况：展示当前集群内各种资源数量、资源额度使用情况、消息类型分布等。
- 集群资源消耗 Top：展示当前集群中占用主要资源的Group和Topic的排行，包括Group的堆积和死信数量排行、Topic 生产速率和消费速率的排行、Topic 占用存储空间的排行。

升配集群

最近更新时间：2024-10-14 11:34:26

操作场景

如当前的集群规格不满足您的业务需求，您可以在控制台上提升您的集群规格和 TPS 规格。

> **说明**
> - 当前仅支持提升集群规格和 TPS 规格，暂不支持降配。
> - RocketMQ 集群在升配的过程中，无论是提升集群规格还是 TPS 规格，腾讯云都保证升级过程平滑和无感，客户侧的应用无需做停机处理。

操作步骤

升级集群规格有两个入口：

- 入口一：登录 RocketMQ 控制台，在**集群列表**页，单击需要调整规格的集群操作列的 **升配**。
- 入口二：登录 RocketMQ 控制台，在**集群基本信息**页面，右上角点击**升配**。
  - 目标集群规格：仅支持提升集群规格，不支持降低。
  - 目标 TPS 规格：仅支持提升 TPS 规格，不支持降低。

删除集群

最近更新时间：2024-10-14 16:17:01

操作场景

当您不再需要 RocketMQ 集群时，可以在控制台上手动销毁，避免产生额外的费用。

> **注意：**
> 删除后，该集群下的所有配置都会被清空，且无法恢复，请谨慎操作。

操作步骤

1. 登录 RocketMQ 控制台 。
2. 在**集群列表**页，单击想要删除的实例操作列的**更多**，单击**删除**。

3. 在删除的确认弹框中，单击**删除**，即可删除集群。

> **说明：**

为了避免用户的误操作造成集群内部数据（如 topic、group 和角色等）被删除，在删除集群时，控制台会对您集群内的资源进行校验，如果存在资源未删除，如 topic 或者 group 等资源未删除，则无法删除集群。

调整公网带宽

最近更新时间：2025-01-23 10:51:02

操作场景

集群创建完成后，您可以通过调整公网带宽来开启/关闭公网访问、修改公网带宽大小和设置公网安全策略来对用户访问进行限制。

操作步骤

公网带宽的调整有两个入口：

- 入口1：在 集群列表 页，单击操作列的 **更多** > **调整网络带宽**。
- 入口2：在**集群基本信息**页面，在接入信息模块的**公网访问**处点击**编辑**。
  - 公网访问：开启公网带宽后会新增单独的费用，计费价格参见 公网计费说明 。
  - 计费模式：默认公网的计费方式和集群的计费方式一致，即如果集群是包月的计费方式，则公网默认也是包月的计费方式。同时，不管是包年包月还是按小时计费的集群，公网支持调整为按流量计费。
  - 公网带宽：选择要调整的公网带宽大小。如果是按流量计费，则公网带宽为当前集群的公网带宽上限。
  - 公网安全策略：填写允许/拒绝访问的 IP或者 IP 段，不设置安全策略默认禁止所有 IP 访问。单个集群最多支持 50 条公网安全策略，如果新增规则和存量规则重复，将优先匹配最后添加的条目。

角色与授权

最近更新时间：2025-03-27 11:43:22

名词解释

RocketMQ 的“角色”是 RocketMQ 内专有的概念，区别于腾讯云的“角色”，是用户自行在 RocketMQ 内部做权限划分的最小单位，用户可以添加多个角色并为其赋予不同集群下的生产和消费权限。每种角色都有其对应的唯一密钥，用户可以通过在客户端中添加密钥来访问 RocketMQ 进行消息的生产消费。
使用场景如下：

- 用户需要安全地使用 RocketMQ 进行消息的生产消费。
- 用户需要对不同的集群设置不同角色的生产消费权限。

例如：一个公司有 A 部门和 B 部门，A 部门的系统产生交易数据，B 部门的系统根据这些交易数据做数据分析和展示。那么遵循权限最小化原则，可以配置两种角色，A 部门角色只授予往交易系统集群中生产消息的权限，B 部门则只授予消费消息的权限。这样可以很大程度避免由于权限不清带来的数据混乱、业务脏数据等问题。

操作步骤

添加角色并授权

体验版和基础版集群

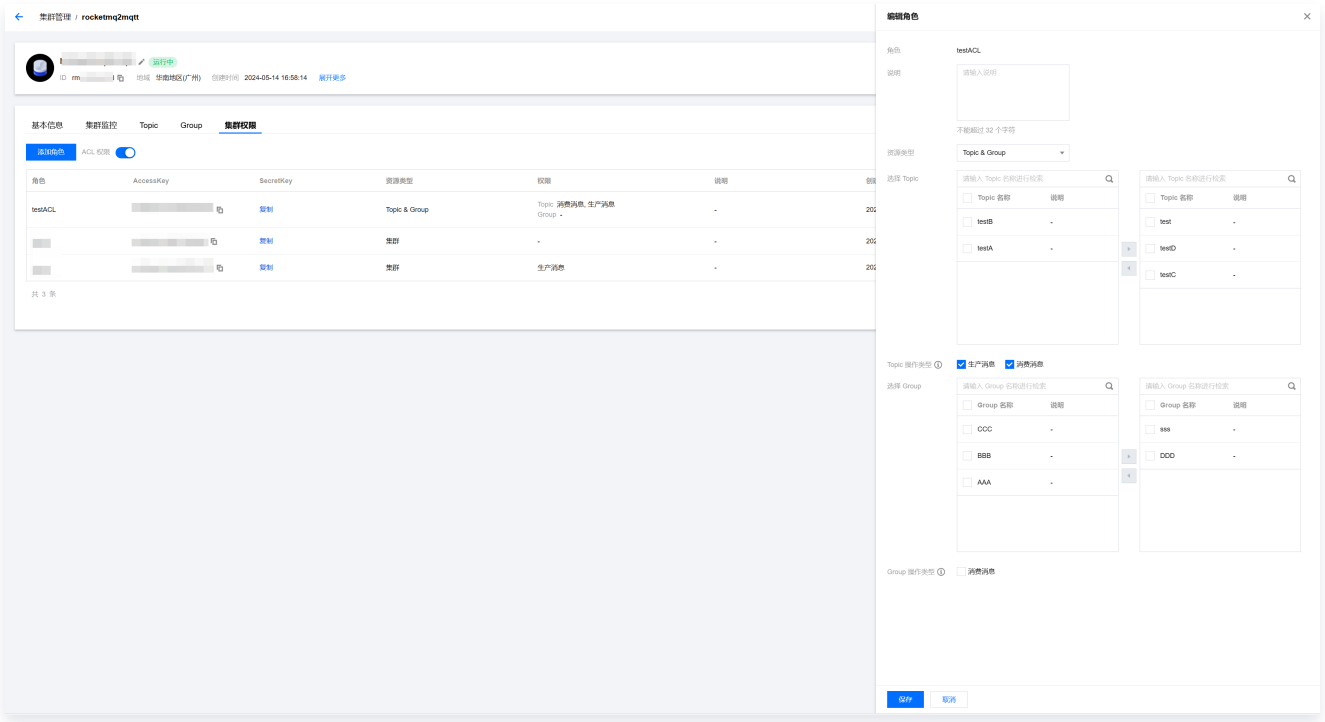
1. 登录 RocketMQ 控制台 。
2. 在左侧导航栏选择**集群列表**，选择地域后，点击要配置角色的集群的“ ID ”，进入基本信息页面。
3. 在页面上方选择**集群权限**页签，单击**添加角色**，输入角色名称，并配置好生产和消费权限。
4. 单击保存，完成角色创建。

专业版和铂金版集群

> **说明：**
> 5.x 集群的 “Topic&Group” 粒度权限功能灰度中，如有需求可以通过工单 联系我们 打开。

1. 登录 RocketMQ 控制台 。
2. 在左侧导航栏选择**集群列表**，选择地域后，点击要配置角色的集群的“ ID ”，进入基本信息页面。
3. 在页面上方选择**集群权限**页签，单击**添加角色**，输入角色名称。专业版集群和铂金版集群支持根据不同的资源粒度来进行权限的细分，当前支持"集群" 和 "Topic&Group" 两种粒度的权限控制。
4. 如果选择 “集群” 粒度，选择生产和消费权限，选择单击保存，完成角色创建。
5. 如果选择 “Topic&Group” 粒度，需要单独针对 Topic 和 Group 配置生产和消费权限，如下图所示。
   - Topic 操作类型分为 生产 和 消费 两类。“生产消息”表示当前用户往选中 Topic 发消息的权限；“消费消息”表示当前用户消费选中 Topic 消息的权限，不勾选表示该用户无论使用任何 Group（即使在 Group 列表配置了消费权限）均无法消费选中的 Topic。
   - Group 操作类型分仅有 消费 选项，表示当前用户消费选中 Topic 的权限，不勾选表示该用户无论使用任何 Group（即使在 Topic 列表配置了消费权限）均无法消费选中的 Topic。

> **说明：**
> 综上所述，如果要保证当前用户使用 GroupA 消费 TopicA 的消息，需要在 Topic 列表选择 TopicA，同时勾选 “消费消息”，同时在 Group 列表选择 GroupA，同时勾选 “消费消息”。



检查权限是否生效

1. 在集群权限列表复制角色密钥。

> **注意：**
> 密钥泄露很可能导致您的数据泄露，请妥善保管您的密钥。

2. 将复制的角色密钥添加到客户端的参数中。如何在客户端代码中添加密钥参数请参考 RocketMQ 的 **代码示例**。
以下给出一种推荐的方式。
2.1 声明 `ACL_SECRET_KEY` 和 `ACL_SECRET_ACCESS` 两个字段，使用各类框架的话建议从配置文件中读取。

```
private static final String ACL_ACCESS_KEY = "eyJr****";
private static final String ACL_SECRET_KEY = "xxx"; /
```

2.2 声明一个静态函数，用于载入一个 RocketMQ Client 的 `RPCHook` 对象。

```
static RPCHook getAclRPCHook() {
    return new AclClientRPCHook(new SessionCredentials(ACL_ACCESS_KEY,
ACL_SECRET_KEY));
}
```

2.3 在创建 RocketMQ `producer` 、`pushConsumer` 或 `pullConsumer` 的时候，引入 `RPCHook` 对象。
以下为创建一个 `producer` 的代码示例：

```
DefaultMQProducer producer = new DefaultMQProducer("rocketmq-mw***|namespace",
"ProducerGroupName", getAclRPCHo
```

3. 运行配置好的客户端访问对应集群中的 Topic 资源，按照刚刚配置的权限进行生产或消费，看是否会产生没有权限的报错信息，如果没有即代表配置成功。

编辑权限

1. 在集群权限列表中，找到需要编辑权限的角色，单击操作列的**编辑**。

2. 在编辑的弹框中，修改权限信息后，单击**保存**。

删除角色

> **注意**
> 删除角色后，原先使用该角色进行生产或消费消息的密钥（ accessKey 和 accessSecret ）将立即失效。请确保当前业务已经没有使用该角色进行消息的生产消费，否则可能会出现客户端无法生产消费而导致的异常。

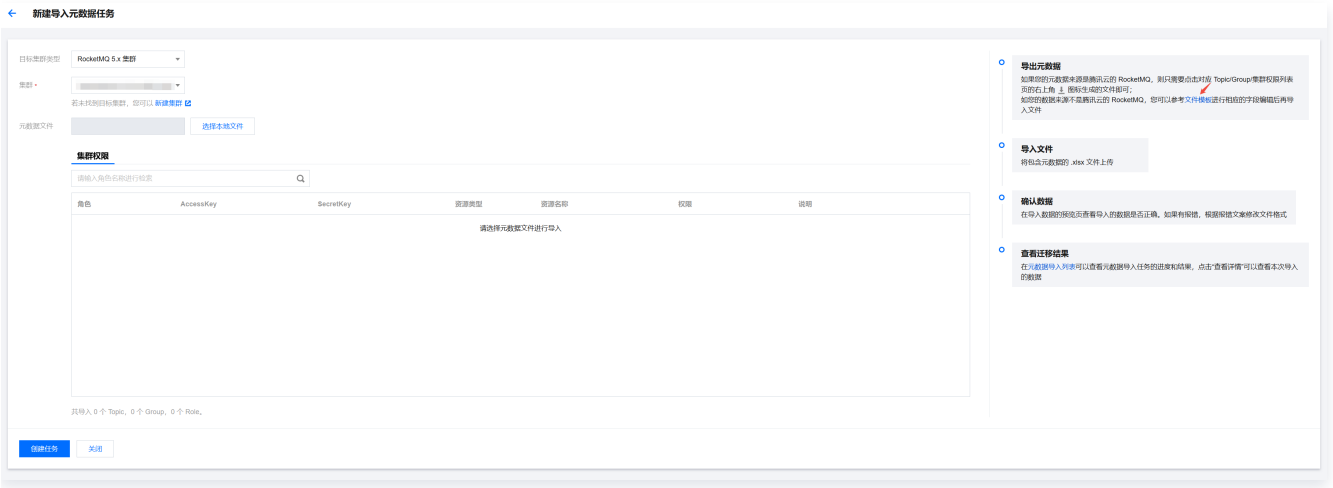
1. 在集群权限列表中，找到需要删除权限的角色，单击操作列的**删除**。

2. 在删除的弹框中，单击**删除**，即可删除该角色。

导入/导出角色

您可以通过角色列表页右上角的 按钮直接导出元数据，元数据的导出格式为 .xlsx 格式的表格文件。

如果您需要将一个集群的角色和权限导入到另一个集群内，在导出元数据后，您可以单击角色列表页右上角的 按钮，将权限数据导入到指定的集群内。如果您的数据来源不是腾讯云的 RocketMQ，可以参考文件模板链接处找到对应的模板，进行相应的字段编辑后再导入文件。



切换集群计费模式

最近更新时间：2024-10-14 18:03:01

操作场景

为了更加方便您使用 RocketMQ，腾讯云开放了 RocketMQ 按量计费和包年包月计费相互转换的功能，本文档介绍在 RocketMQ 控制台进行计费模式切换的操作。

转换规则

- 计费方式转换成功及支付成功后，集群会即刻新的计费模式计费，新集群的起始时间为转换成功时间。
- 集群计费模式转换之后，公网带宽也会自动转换计费模式。
- 计费模式由按量计费转换为包年包月的 RocketMQ 不支持五天内无理由退还。

使用限制

- 仅“运行中”的状态集群支持转计费模式，在“隔离中/发货/异常”状态的集群不支持修改计费模式。

操作步骤

按量计费转包年包月

1. 登录 RocketMQ 控制台 。
2. 在左侧导航栏单击**集群管理**，单击目标集群的操作栏的 **更多** > **转包年包月计费**。
3. 在弹出的按量计费转包年包月窗口中，根据实际需求，设置续费时长以及是否自动续费。

4. 勾选**已阅读并同意计费模式转换规则**，单击**确认**。
5. 根据页面提示，完成支付，即完成转换操作。

包年包月转按量计费

1. 登录 RocketMQ 控制台 。
2. 在左侧导航栏单击**集群管理**，单击目标集群的操作栏的 **更多** > **转按量计费**。
3. 在弹出的窗口中，根据实际需求，设置续费时长以及是否自动续费。

4. 勾选**已阅读并同意计费模式转换规则**，单击**确认**，即完成转换操作。

Topic 管理

最近更新时间：2025-04-17 10:41:01

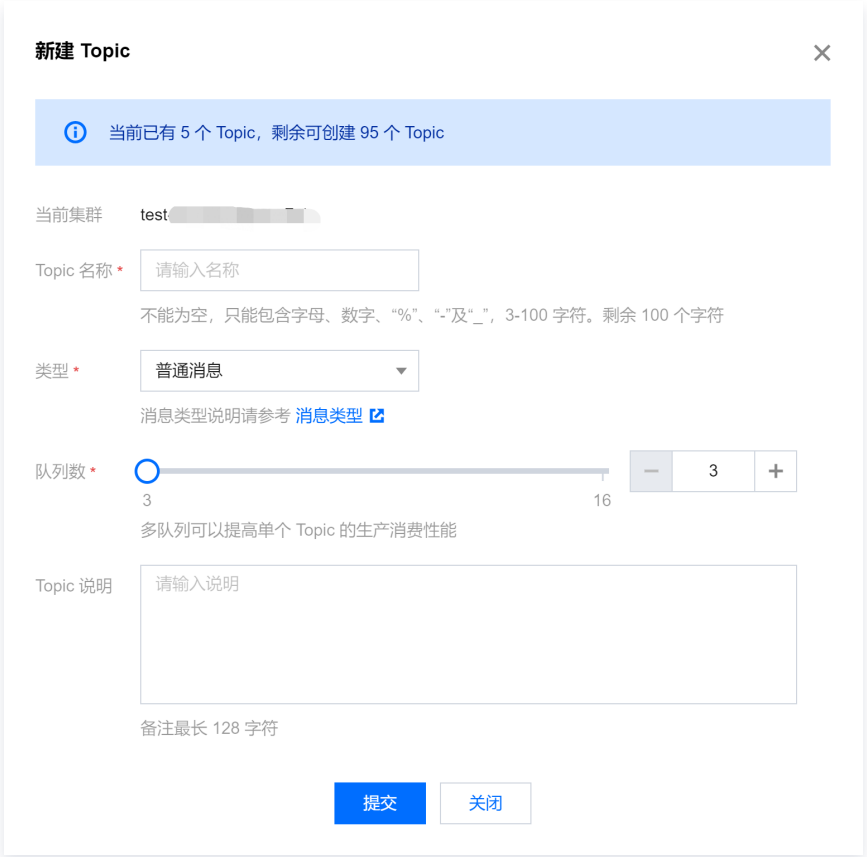
操作场景

Topic 是 TDMQ RocketMQ 版中的核心概念。Topic 通常用来对系统生产的各类消息做一个集中的分类和管理，例如和交易的相关消息可以放在一个名为 “trade” 的 Topic 中，供其他消费者订阅。
在实际应用场景中，一个 Topic 往往代表着一个业务聚合，由开发者根据自身系统设计、数据架构设计来决定如何设计不同的 Topic。
本文档可以指导您使用 TDMQ RocketMQ 版时，利用 Topic 对消息进行分类管理。

操作步骤

创建 Topic

1. 登录 RocketMQ 控制台 。
2. 在左侧导航栏选择 **Topic 管理**页签，选择好地域和集群后，单击**新建**进入创建 Topic 页面。
3. 在新建 Topic 对话框中，填写以下信息。
   - Topic 名称：填写 Topic 名称（创建后不可修改），3-64个字符，只能包含字母、数字、“-”及“_”
   - 类型：选择消息类型，包括：普通、顺序消息、延迟消息和事务消息（关于消息类型的说明，请参见 消息类型 ）。
   - 队列数：选择队列数量，最大支持16队列。多队列可以提高单 Topic 的生产消费性能，但是非顺序消息的场景下无法保证顺序性。
   - Topic 说明：填写 Topic 的说明信息，最长128字符。
4. 单击**提交**，在 Topic 列表中即可看见创建好的 Topic。



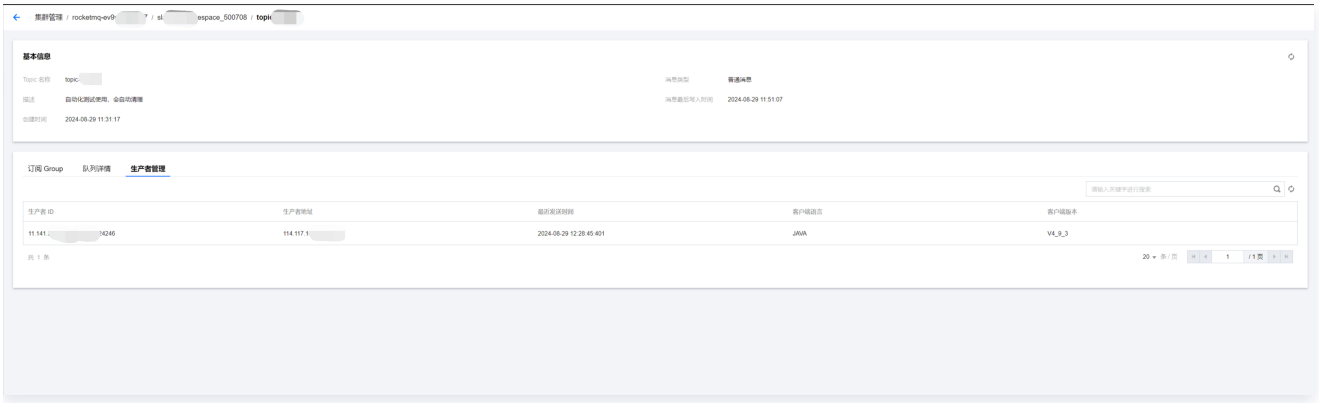
发送测试消息

RocketMQ 控制台支持手动发送消息，在控制台进行相应的操作即可实现消息发送给指定的 Topic。

1. 在 **Topic 管理**列表中，单击目标Topic 操作栏的**发送测试消息**。
2. 在弹窗中输入消息 Key，消息 Tag 和消息内容，单击**发送**。

查看生产者信息

在 Topic 详情页，可以查看当前 Topic 下生产者的相关信息。



查看订阅的 Group

1. 在 **Topic 管理**列表中，单击目标 Topic 的“ID”。
2. 页面跳转到 Group 列表，展示订阅该 Topic 的 Group 信息。

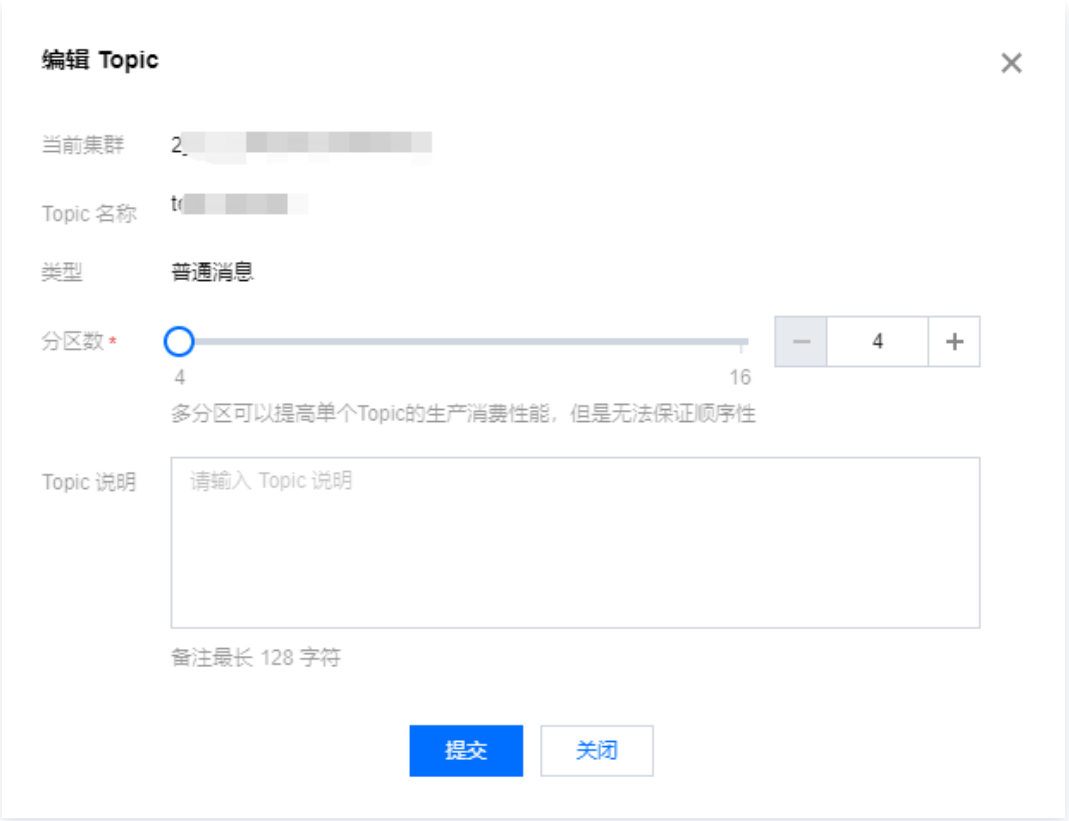
查询 Topic

您可以在 Topic 管理列表页右上角的搜索框中，通过 Topic 名称进行搜索查询，TDMQ RocketMQ 版将会模糊匹配并呈现搜索结果。

编辑 Topic

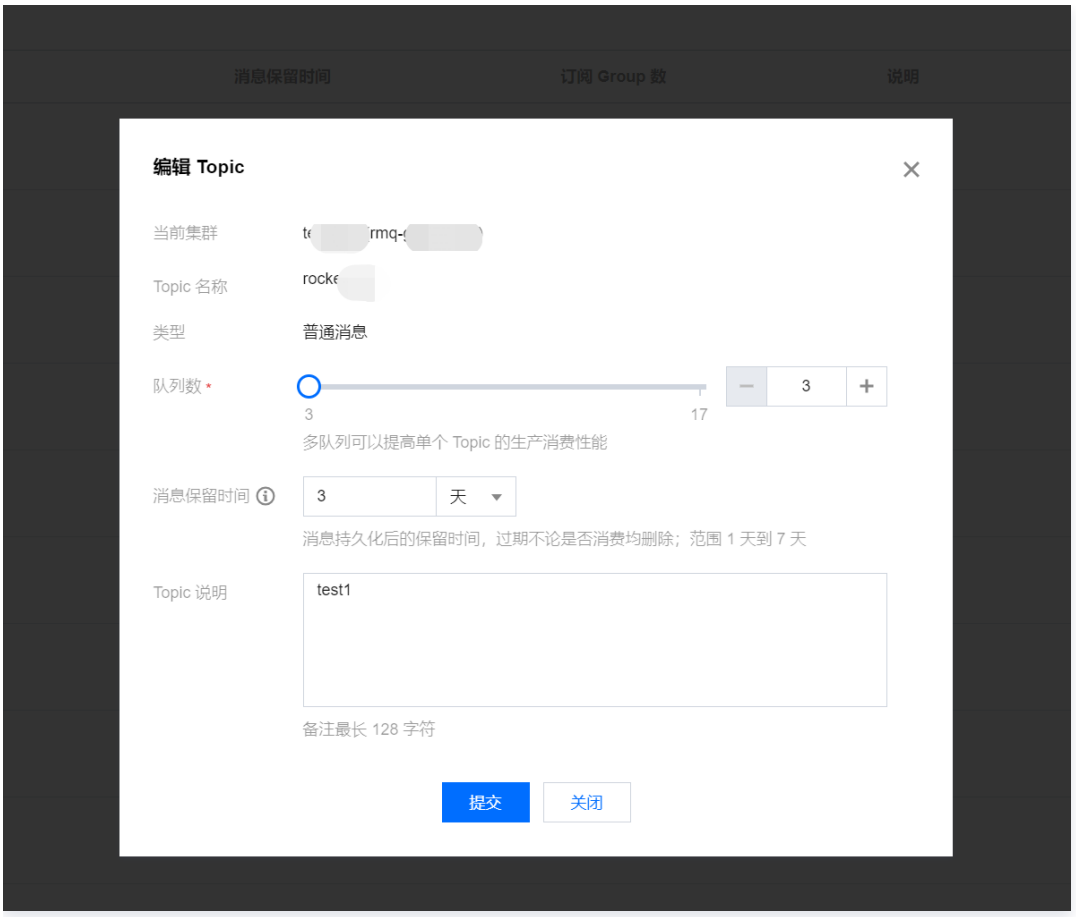
1. 在 Topic 管理列表中，找到需要编辑的 Topic ，单击操作栏中的**编辑**。

2. 在弹出的对话框中可以对 Topic 的说明进行编辑。
3. 单击**提交**即完成对 Topic 的编辑。



为了方便用户根据不同的业务要求调整消息的保留时间，专业版和铂金版集群支持按照 Topic 粒度调整消息的保留时间。在专业版和铂金版集群购买时，默认所有 Topic（主题）的初始化消息保留时间为 3 天，在使用过程中，用户可以根据不同的业务需求进行调整，消息保留时间的范围为 1-7 天，如用户需要更长的消息保留时间，可以通过工单联系调整。

要修改 Topic 的消息保留时间，如下图所示，在 **Topic 管理** 列表中，找到需要编辑的 Topic ，单击操作栏中的**编辑**。



删除 Topic

- **批量删除：**在 **Topic 管理**列表中，勾选所有需要删除的 Topic，单击左上角的**批量删除**，在弹出的提示框中，单击**删除**，完成删除。
- **单个删除：**在 **Topic 管理**列表中，找到需要删除的 Topic，单击操作列的**删除**，在弹出的提示框中，单击**删除**，完成删除。

> **注意：**
> 删除了 Topic 之后也会清除该 Topic 下积累的未消费消息，请谨慎执行。

元数据导入导出

元数据导出

您可以通过 Topic 列表页右上角的 按钮直接导出元数据，元数据的导出格式为 .xlsx 格式的表格文件。

元数据导入

如果您需要将一个集群的 Topic 信息载入到另一个集群内，在导出元数据后，您可以单击 Topic 列表页右上角的 按钮，将 Topic 数据导入到指定的集群内。

Group 管理

最近更新时间：2024-08-01 10:32:41

操作场景

Group 用于标识一类 Consumer，这类 Consumer 通常消费同一类消息，且消息订阅的逻辑一致。
该任务指导您使用消息队列 TDMQ RocketMQ 版时在控制台上创建，删除和查询 Group。

前提条件

- 需要提前创建好对应的命名空间。
- 根据 TDMQ 提供的 SDK 创建好消息的生产者和消费者并正常运行。

操作步骤

创建 Group

5.x 集群

1. 登录 RocketMQ 控制台 。
2. 在左侧导航栏中选择 **Group 管理**，选择地域和目标集群。

3. 单击**新建**进入创建 Group 页面。
4. 填写 Group 相关信息。
   - Group 名称：填写 Group 名称（创建后不可修改），3-64个字符，只能包含字母、数字、“-”及“_”。
   - Group 说明：填写 Group 说明。
   - 最大重试次数：表示消息可以重新被投递的最大次数，超过最大重试次数还没被成功消费，消息将被投递至死信队列或丢弃。如果您使用的是 RocketMQ 4.x 客户端，消息的重试次数以您在客户端内设置消息重试次数为准。如果您使用的是 RocketMQ 5.x 客户端，则消息重试次数以您在当前页面设置的为准。
   - 投递顺序性：服务端将消息投递给消费者消费的顺序，支持顺序投递和并发投递，默认投递方式为并发投递。
   - 开启消费：关闭后 Group 下的所有消费会暂停，重新开启可继续消费。
5. 单击**提交**，完成 Group 创建。

4.x 集群

1. 登录 TDMQ RocketMQ 版控制台，选择地域后，单击目标集群的“ID”进入集群基本信息页面。
2. 单击顶部 **Group** 页签，选择命名空间后，单击**新建**进入创建 Group 页面。
3. 填写 Group 相关信息。

- Group 名称：填写 Group 名称（创建后不可修改），3-64个字符，只能包含字母、数字、“-”及“_”。
- 协议类型：支持 HTTP 和 TCP 协议。
- Group 说明：填写 Group 说明。
- 开启消费：关闭后 Group 下的所有消费会暂停，重新开启可继续消费。
- 开启广播模式：关闭后 Group 下的所有声明为广播模式的消费者会暂停，重新开启可继续消费。

4. 单击**提交**，完成 Group 创建。

> **注意：**
> 为了保障线上集群的稳定性，避免控制台的元数据冗余，腾讯云 TDMQ-RocketMQ 于2023年2月底关闭了 group 自动创建的配置。用户在启动消费者客户端时，需要先在控制台创建对应的 Group。

查看消费者详情

1. 在 **Group** 列表，点击 Group 名称，进入客户端连接列表，可以查看 Group 基本信息及客户端连接列表。

5.x 集群

- Group 名称
- 创建时间
- 投递顺序性：顺序投递或者并发投递

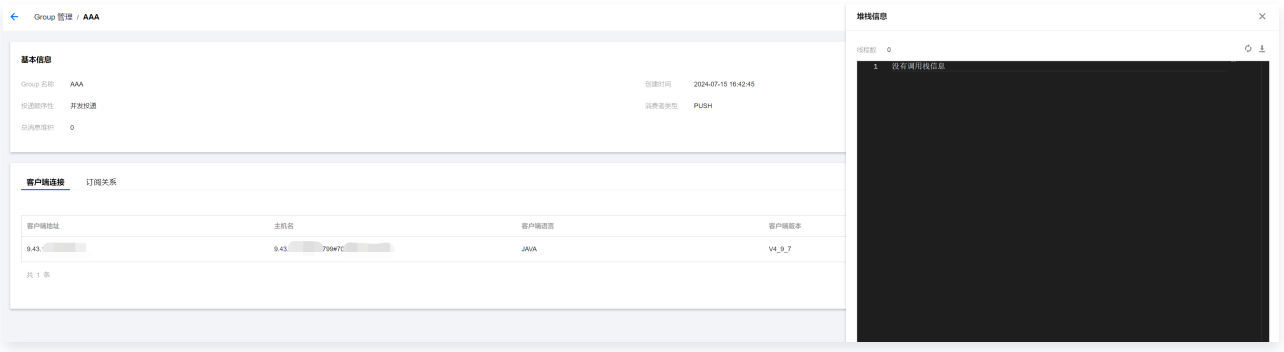
- 消费者类型：PUSH 或者 PULL
- 总消息堆积：消息堆积的总数量

4.x 集群

- 消费模式：集群模式或者广播模式
  - 集群消费：当使用集群消费模式时，任意一条消息只需要被集群内的任意一个消费者处理即可。
  - 广播消费：当使用广播消费模式时，每条消息会被推送给集群内所有注册过的消费者，保证消息至少被每个消费者消费一次。
- 客户端协议：支持 TCP 和 HTTP 协议
- 总消息堆积：消息堆积的总数量
- 消费者类型：ACTIVELY（主动拉消息，Pull 模式）或 PASSIVELY（客户端等待服务端推消息，Push 模式）

2. 单击客户端操作栏的**查看详情**可查看消费者详情。

3. 单击客户端的 **查看堆栈**，则可以查看客户端的堆栈详情和线程数量。



4. 切换到订阅关系页签，可以查看该 Group 订阅的 Topic 列表与订阅属性。

设置 offset

1. 在 Group 列表中，单击目标 Group 操作列的**重置位点**。

2. 在弹窗中，可以选择**从最新位点开始**或者**按从指定时间点开始**设定 Topic 的**消费位移 offset**（即指定该订阅下的消费者从哪里开始消费消息）。
3. 单击**提交**，完成设置。

> **说明：**
> TDMQ-RocketMQ 支持给离线的 Group 重置 offset（消费位点），但目前仅支持 Push 消费模式下的消费者组，否则会出现重置失败的情况。

编辑 Group

1. 在 Group 列表中，单击目标 Group 操作列的**编辑**。

2. 在弹窗中，对 Group 信息进行编辑。
3. 单击**提交**，完成修改。

删除 Group

- **批量删除：**在 Group 列表中，勾选所有需要删除的 Group，单击左上角的**批量删除**，在弹出的提示框中，单击**删除**，完成删除。
- **单个删除：**在 Group 列表中，找到需要删除的 Group，单击操作列的**删除**，在弹出的提示框中，单击**删除**，完成删除。

> **注意：**
> 删除 Group 后，由该 Group 标识的消费者将立即停止接收消息，该 Group 下的所有配置将会被清空，且无法恢复，请您谨慎执行该操作。

元数据导入导出

元数据导出

您可以通过 Group 列表页右上角的 ⬇ 按钮直接导出元数据，元数据的导出格式为 `.xlsx` 格式的表格文件。

元数据导入

如果您需要将一个集群的 Group 信息载入到另一个集群内，在导出元数据后，您可以点击 Group 列表页右上角的 ⬆ 按钮，将 Group 数据导入到指定的命名空间下。

监控告警

最近更新时间：2025-01-25 11:40:32

操作场景

TDMQ RocketMQ 支持监控您账户下创建的资源，包括集群、Topic、Group 等，您可以根据这些监控数据，分析集群的使用情况，针对可能存在的风险及时处理。同时您也可以对监控项设置报警规则，以便数据异常时收到报警消息，及时处理风险，保障系统的稳定运行。

监控指标

TDMQ RocketMQ 版支持的监控指标如下：

指标	单位	备注	指标支持维度
总 TPS	Count/s	生产和消费消息的 API 调用次数之和（按照计费规则进行折算）	集群、Topic
生产 TPS	Count/s	生产消息的 API 调用次数之和（按照计费规则进行折算）	集群、Topic
消费 TPS	Count/s	消费消息的 API 调用次数之和（按照计费规则进行折算）	集群、Topic、Group
消息存储空间	GB	-	集群、Topic
消息堆积条数	Count/s	-	集群、Topic、Group、Topic&Group
被限流的生产 TPS	Count/s	-	集群、Topic
被限流的消费 TPS	Count/s	-	集群、Topic、Group
生产消息条数	Count/s	-	集群、Topic
消费消息条数	Count/s	-	集群、Topic、Group
生产消息流量	MB/s	-	集群、Topic
消费消息流量	MB/s	-	集群、Topic、Group
生产者数量	Count	仅统计在线的生产者客户端	集群、Topic
生产成功率	%	-	集群、Topic
生产消息平均耗时(发送RT)	ms	使用 SDK 调用发送消息接口成功的耗时，即生产消息的 RT，仅 5.x 客户端能采集到该指标	集群、Topic
消息平均大小	Bytes	-	集群、Topic
各类型消息数量变化	Count	-	集群

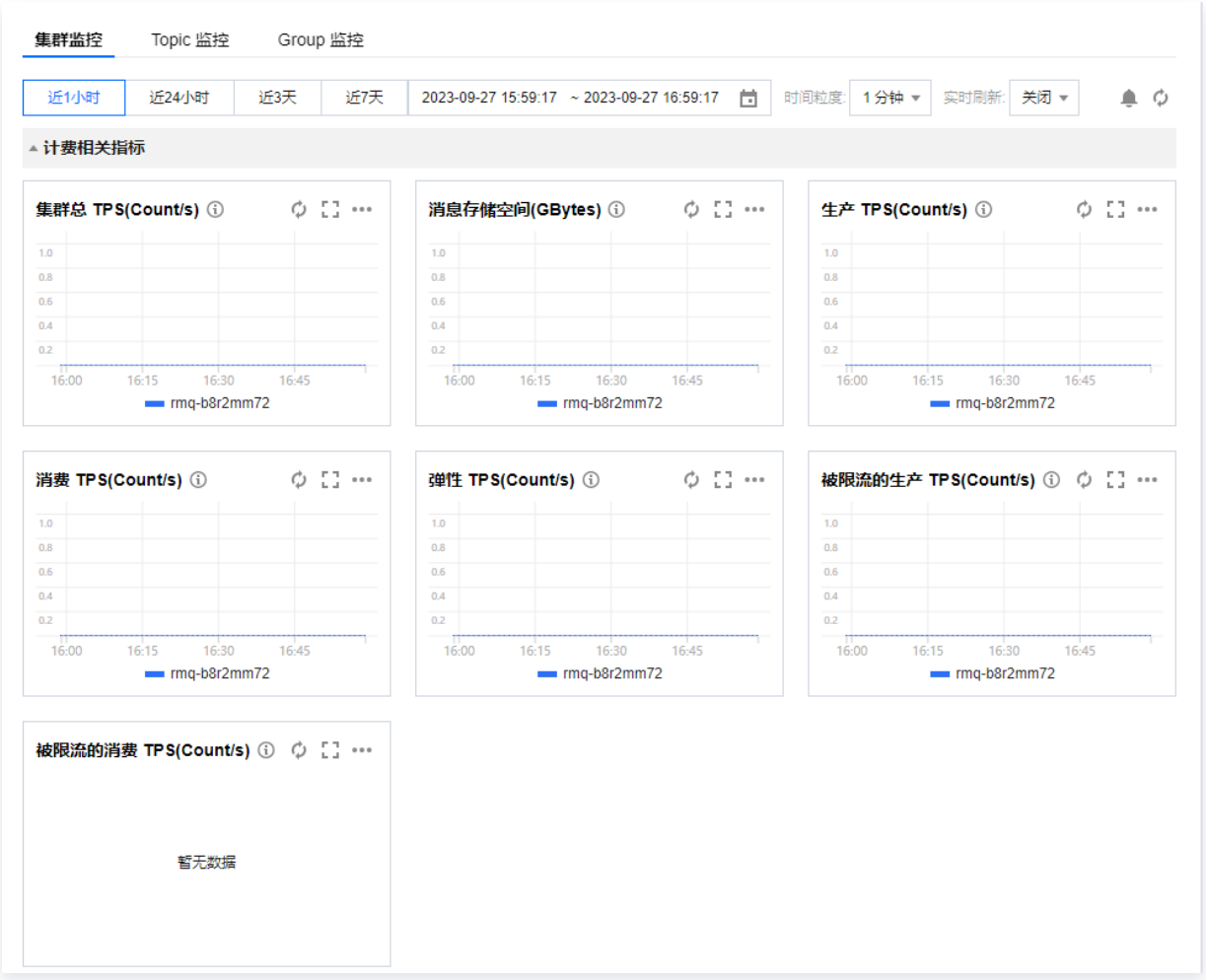
每秒被保存的死信消息条数	Count	每秒新增的状态为 DLQ 的消息数量，表示消息达最大重试次数后依旧消费失败，但是保存到指定 Topic 的消息数	集群、Topic、Group、Topic&Group
已就绪消息的排队时间	ms	最早一条已就绪消息的就绪时间和当前时刻的时间差，反映了消费者拉取消息的及时性	集群、Topic、Group、Topic&Group
处理中的消息数	Count	状态为 Inflight 的消息数量，表示消息在服务端消费，还未返回消费结果的消息数量	集群、Topic、Group、Topic&Group
已就绪消息的排队时间	ms	最早一条已就绪消息的就绪时间和当前时刻的时间差，反映了消费者拉取消息的及时性	集群、Topic、Group、Topic&Group
消费处理滞后时间	ms	最早一条未返回响应的消息的就绪时间和当前时刻的时间差，反映了消费者完成消费消息的及时性	集群、Topic、Group、Topic&Group
本地缓存队列中的平均排队时间	ms	仅展示 5.x 版本的 PushConsumer客户端的数据，SimpleConsumer客户端没有缓存队列	集群、Topic、Group、Topic&Group
消费消息平均耗时(发送RT)	ms	使用 SDK 调用发送消息接口成功的耗时，即生产消息的 RT，仅 5.x 客户端能采集到该指标	集群、Topic、Group、Topic&Group、消费者客户端
重试消息条数	Count	-	Topic、Group、Topic&Group
消费耗时分布	-	不同消费消息耗时范围的热力分布图（仅专业版和铂金版）	集群、Topic、Group、Topic&Group、消费者客户端
生产耗时分布	-	不同生产消息耗时范围的热力分布图（仅专业版和铂金版）	集群、Topic
生产消息大小分布	-	不同消息大小的热力分布图（仅专业版和铂金版）	集群、Topic
公网流出/入流量	MB/s	-	集群
公网流出/入带宽	Mbps/s	-	集群
公网流出/入带宽利用率	%	-	集群
公网丢弃出/入带宽	Mbps/s	-	集群

查看监控数据

1. 登录 RocketMQ 控制台 。
2. 在左侧导航栏单击**监控大盘**，选择好地域和要查看的集群。

3. 在监控页面选择要查看的资源页签，设置好时间范围后，查看对应的监控数据。

图标	说明
时间粒度: 1分钟	单击可调整图表时间粒度，支持1分钟、5分钟和1小时。
实时刷新: 关闭	单击可刷新获取最新的监控数据，支持设置30s、1min和5min时间间隔自动刷新监控数据。
•••	单击可将图表复制到 Dashboard，关于 Dashboard 请参见 什么是 Dashboard 。



在集群监控页面，您可以选择集群内的多个 Topic，查看多个 Topic 的指标对比，如下图所示。

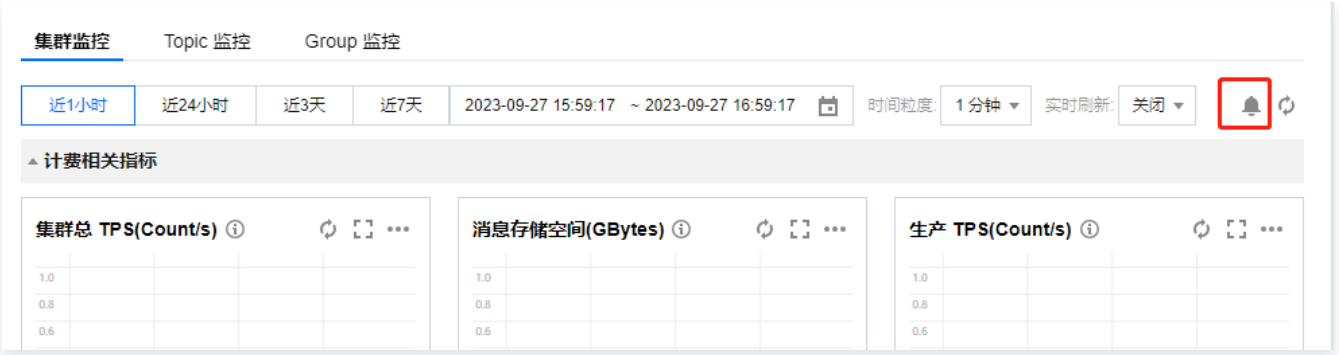
同理，您可以查看某个Topic下，订阅关系内的多个 Group 的相关指标对比；也可以查看某个 Group 下，订阅关系内的多个 Topic 的相关指标对比。

配置告警规则

新建告警规则

您可以为监控指标配置告警规则，当监控指标达到设定的报警阈值时，腾讯云可观测平台可以通过邮件、短信、微信、电话等方式通知您，帮助您及时应对异常情况。

1. 在集群的**监控**页面，单击下图告警按钮跳转至 腾讯云可观测平台控制台 配置告警策略。



2. 在告警策略页面，选择好策略类型和要设置告警的实例，设置好告警规则和告警通知模板。
   - **策略类型**：选择**消息队列 TDMQ/RocketMQ5集群**。
   - **告警对象**：选择需要配置告警策略的 **RocketMQ** 实例。
   - 触发条件：支持**选择模板**和**手动配置**，默认选择手动配置，手动配置参见以下说明，新建模板参见 新建触发条件模板 。

     > **说明：**
     > - 指标：例如“消息生产条数TPS”，选择统计粒度为1分钟，则在1分钟内，消息生产条数TPS连续N个数据点超过阈值，就会触发告警。
     > - 告警频次：例如“每30分钟警告一次”，指每30分钟内，连续多个统计周期指标都超过了阈值，如果有一次告警，30分钟内就不会再次进行告警，直到下一个30分钟，如果指标依然超过阈值，才会再次告警。

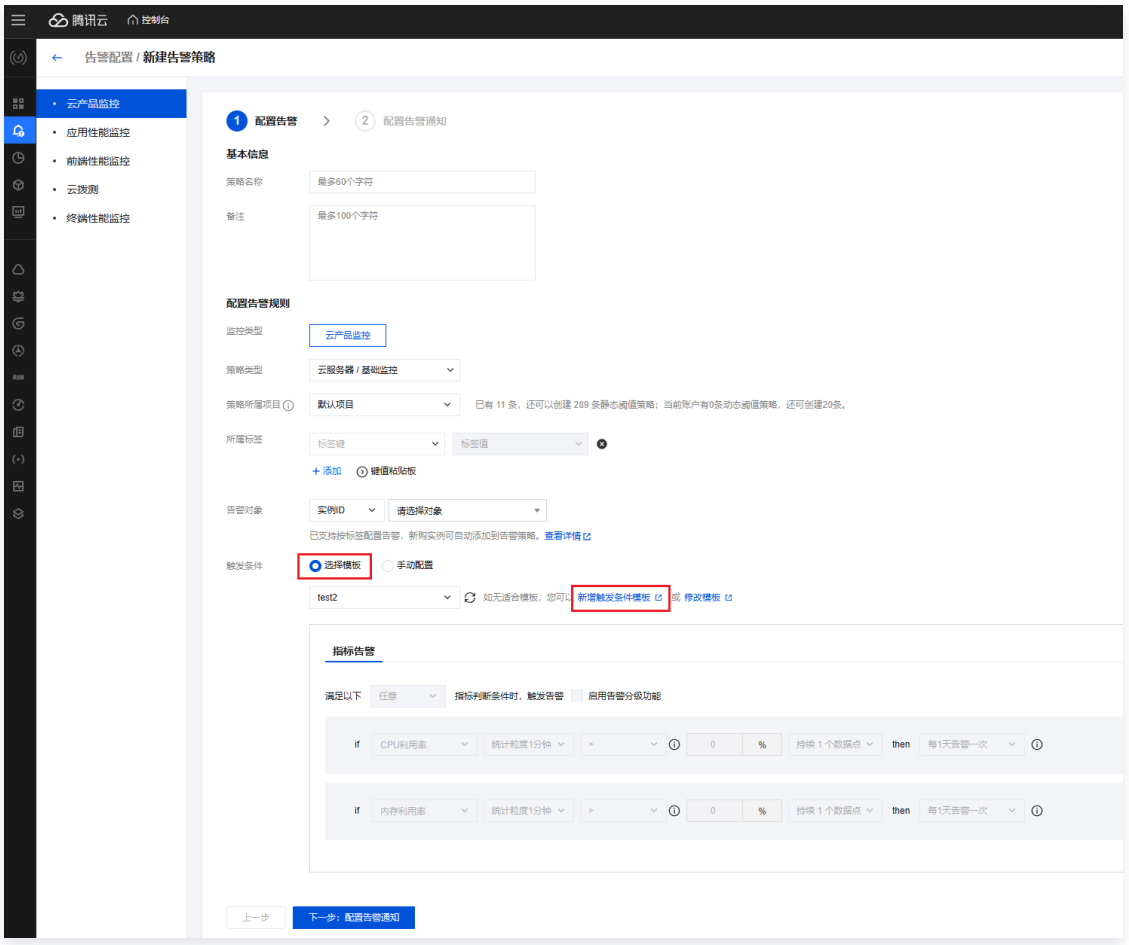
   - **通知模板**：选择通知模板，也可以新建通知模板，设置告警接收对象和接收渠道。
3. 单击**完成**，完成配置。

> **说明：**

> 有关告警的更多信息，请参见 腾讯云可观测平台告警服务 。

新建触发条件模板

1. 登录 腾讯云可观测平台控制台 。
2. 进入**新增触发条件模板**页，有以下两种方式：
   2.1 单击**告警管理 > 告警配置 > 云产品监控 > 告警策略 > 新建策略**，进入**新建配置告警策略**页，在**配置告警规则**栏的**触发条件**项单击**选择模板**，然后单击**新增触发条件模板**。



   2.2 单击**告警管理 > 告警配置 > 云产品监控 > 触发条件模板 > 新建触发条件模板**。

3. 在新建模板页，配置策略类型。

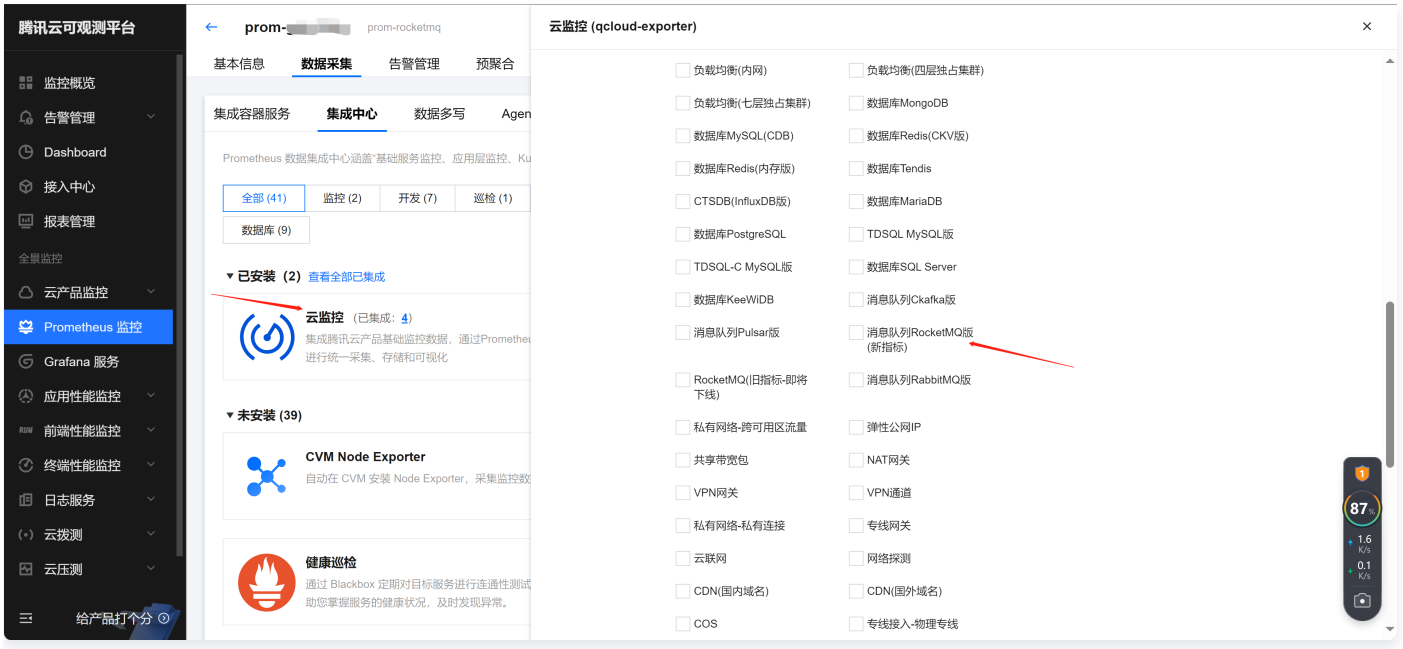
- **策略类型**：选择 **消息队列TDMQ/RocketMQ5/集群**。
- **使用预置触发条件**：勾选此选项，会出现系统建议的告警策略。

4. 确认无误后，单击**保存**。
5. 返回新建告警策略页，就会出现刚配置的告警策略模板。

对接云监控 Prometheus

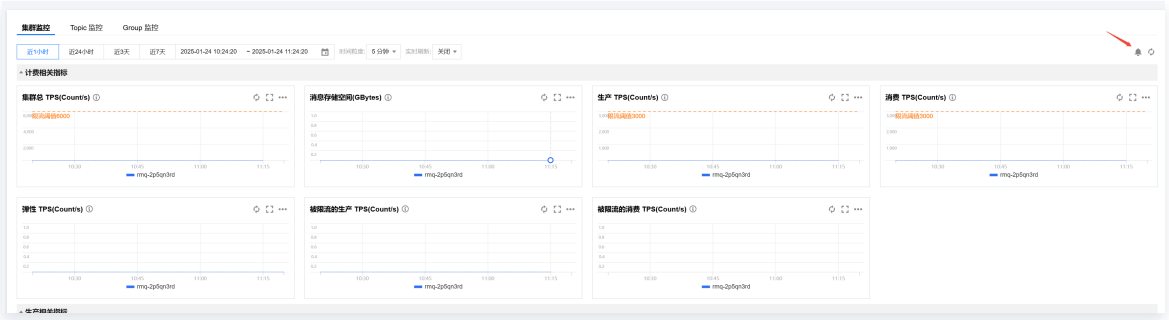
如果您使用了腾讯云可观测平台，并使用了 Prometheus 实例，您可以通过 Prometheus 实例监控腾讯云 RocketMQ。

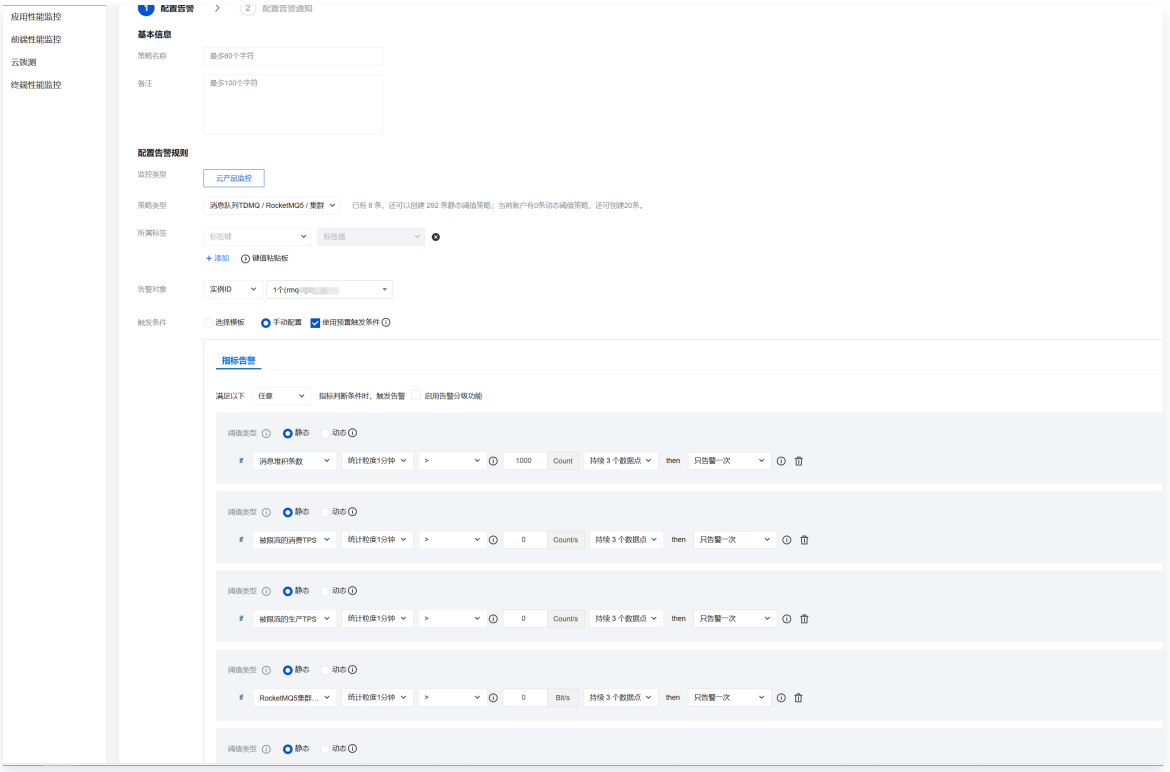
1. 登录 Prometheus 监控控制台，在实例列表中，选择对应的 Prometheus 实例。
2. 进入实例详情页，选择 **数据采集**，再单击 **集成中心**。
3. 选择 **云监控**，可以直接单击 **一键安装**。如果您只需要查看 RocketMQ 的监控数据，如下图所示，您可以在“云产品选择” 处选择 **消息队列RocketMQ版（新指标）**，同时填写其他相关信息，如名称、地域等等。详细操作步骤可以参见 Prometheus 集成中心 。



一键配置告警模板

1. 进入 消息队列 RocketMQ 控制台 。
2. 单击**监控大盘**或者对应资源的监控详情页的 按钮，页面会自动跳转到腾讯云可观测平台的**告警配置**页面，同时配置上对应集群 ID 的相关资源的推荐告警指标。





消息查询
查询普通消息

最近更新时间：2024-08-01 10:32:41

当一条消息从生产者发送到 TDMQ RocketMQ 版服务端，再由消费者进行消费，TDMQ RocketMQ 版会完整记录这条消息中间的流转过程，并以消息轨迹的形式呈现在控制台。
消息轨迹记录了消息从生产端到 TDMQ RocketMQ 版服务端，最后到消费端的整个过程，包括各阶段的时间（精确到微秒）、执行结果、生产者 IP、消费者 IP 等。

- 如果您使用的是 5.0 及以上版本的 gRPC 客户端进行消息的生产和消费，则无需在客户端另行开启轨迹开关。
- 如果您使用的是 4.x 版本的客户端，或者 5.0 以上版本的 Remoting 客户端，则需要在客户端来设置开启消息轨迹功能，具体设置示例如下：

更多关于客户端的说明请参见 社区客户端说明 。

生产者设置

```
DefaultMQProducer producer = new DefaultMQProducer(namespace, groupName,
    // ACL权限
    new AclClientRPCHook(new SessionCredentials(AK, SK)), true, null);
```

Push 消费者设置

```
// 实例化消费者
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(NAMESPACE,groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)),
    new AllocateMessageQueueAveragely(), true, null);
```

Pull 消费者设置

```
DefaultLitePullConsumer pullConsumer = new
DefaultLitePullConsumer(NAMESPACE,groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)));
// 设置NameServer的地址
pullConsumer.setNamesrvAddr(NAMESERVER);
pullConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_LAST_OFFSET);
pullConsumer.setAutoCommit(false);
pullConsumer.setEnableMsgTrace(true);
pullConsumer.setCustomizedTraceTopic(null);
```

Spring Boot Starter 接入（2.2.2版本及以上）

```
package com.lazycece.sbac.rocketmq.messagemodel;

import lombok.extern.slf4j.Slf4j;
```

```
import org.apache.rocketmq.spring.annotation.MessageModel;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;

/**
 * @author lazycece
 * @date 2019/8/21
 */
@Slf4j
@Component
public class MessageModelConsumer {

@Component
    @RocketMQMessageListener(
            topic = "topic-message-model",
            consumerGroup = "message-model-consumer-group",
            enableMsgTrace = true,
            messageModel = MessageModel.CLUSTERING)
    public class ConsumerOne implements RocketMQListener<String> {
        @Override
        public void onMessage(String message) {
            log.info("ConsumerOne: {}", message);
        }
    }

}
```

操作场景

当您需要排查以下问题时，就可以使用 TDMQ RocketMQ 版控制台的消息查询功能，按照时间维度或者根据日志中查到的消息 ID 或消息 Key，来查看具体某条消息的消息内容、消息参数和消息轨迹。

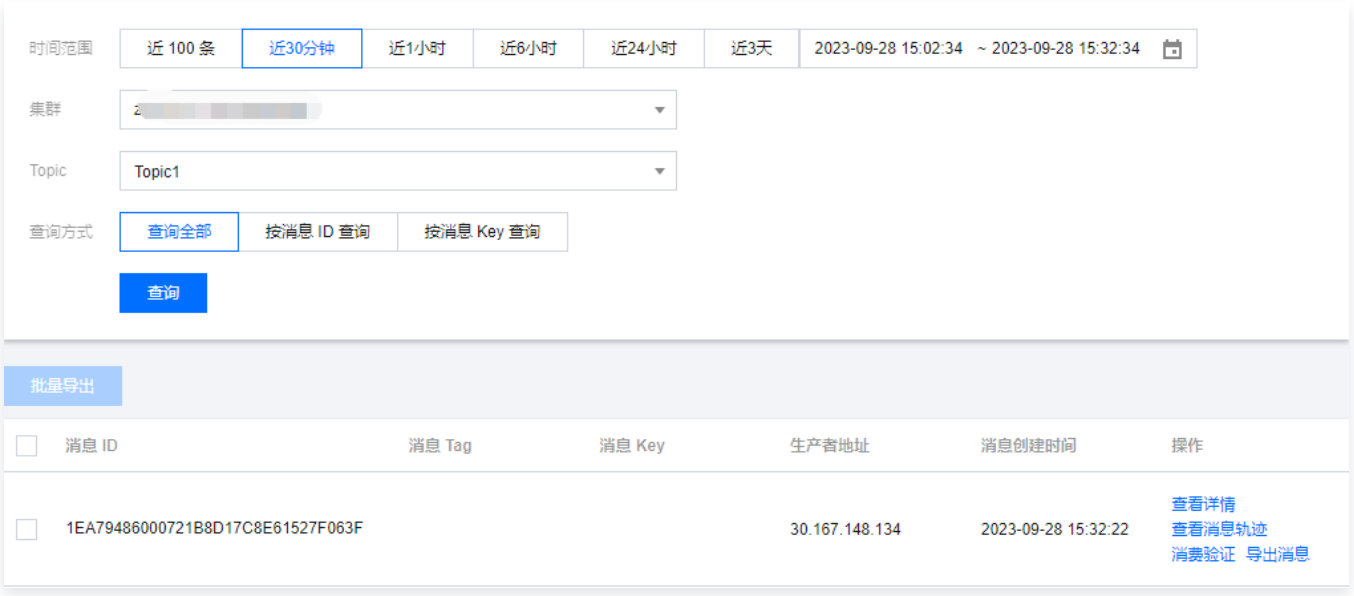
- 查看某条消息的具体内容，具体参数。
- 查看消息由哪个生产 IP 发送，是否发送成功，消息到服务端的具体时间。
- 查看消息是否已持久化。
- 查看消费由哪些消费者消费了，是否消费成功，消息确认消费的具体时间。
- 需要做分布式系统的性能分析，查看 MQ 对相关消息处理的时延。

操作步骤

1. 登录 RocketMQ 控制台，在左侧导航栏单击**消息查询**。
2. 在消息查询页面，选择好地域后根据页面提示输入查询条件。
    - 时间范围：选择需要查询的时间范围，支持近100条（默认按时间顺序展示最近的 100 条消息），近30分钟，近1小时，近6小时，近24小时，近3天和自定义时间范围。
    - 集群：选择需要查询的 Topic 所在的集群。
    - Topic：选择需要查询的 Topic。
    - 查询方式：消息查询功能支持以下查询方式。
        - **查询全部**：该方式适合查询全部消息。

- **按消息 ID 查询：**该方式属于精确查询、速度快、精确匹配。
- **按消息 Key 查询：**该方式属于模糊查询，适用于您没有记录消息 ID 但是设置了消息 Key 的场景。

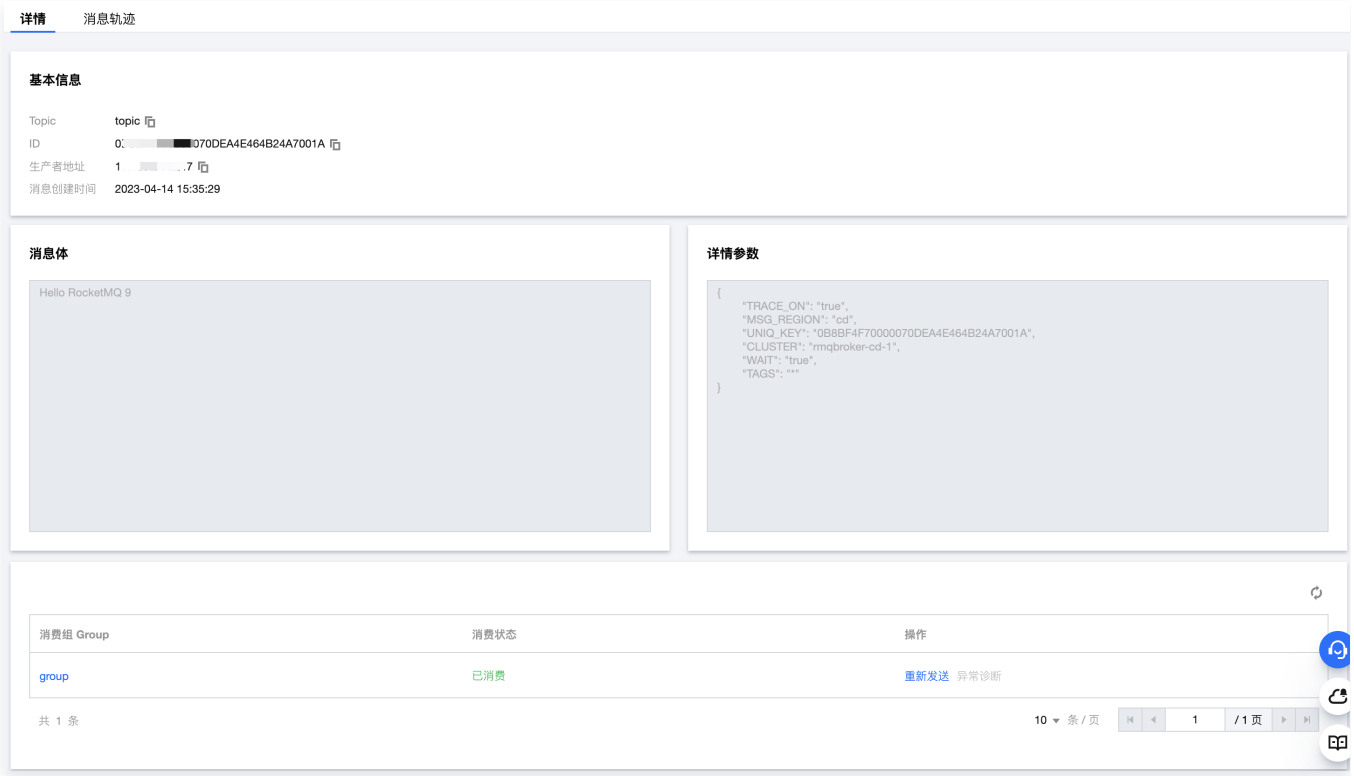
3. 单击**查询**，下方列表会展示所有查询到的结果并分页展示。



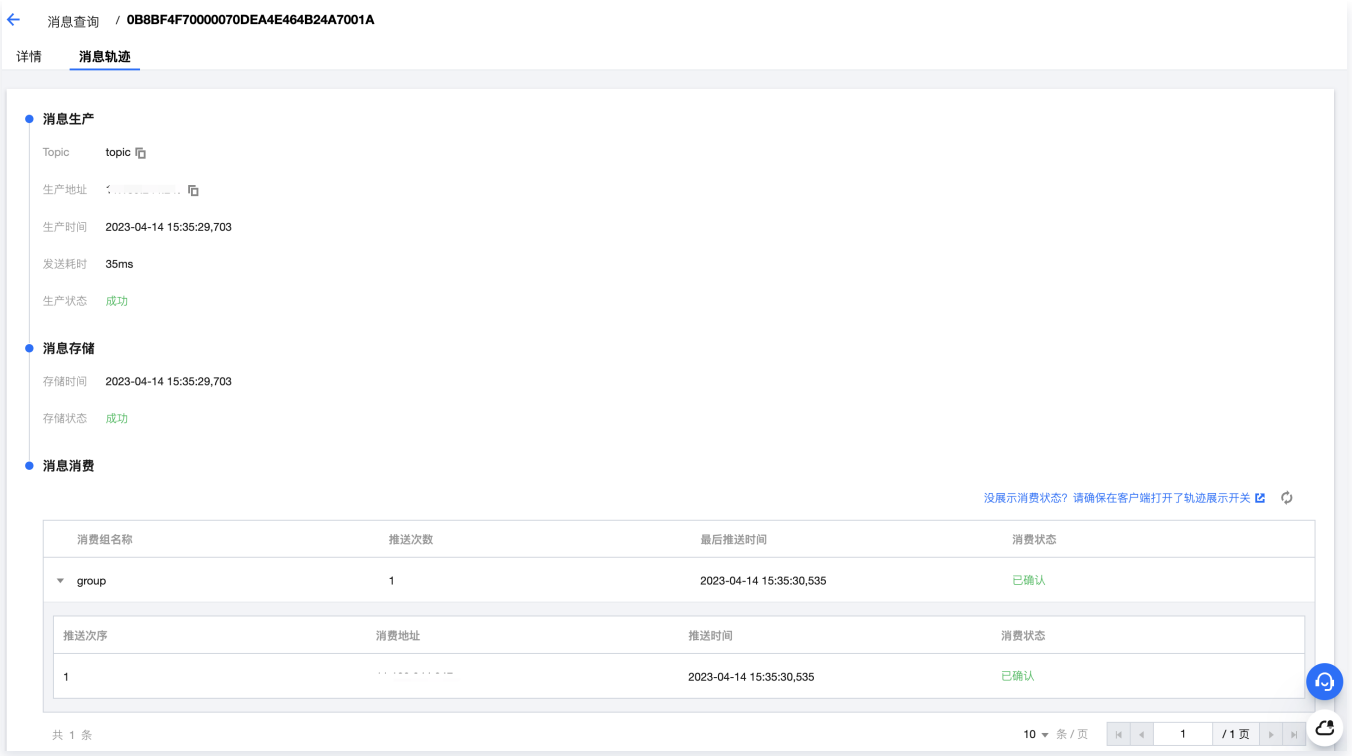
4. 找到您希望查看内容或参数的消息，单击操作列的**查看详情**，即可查看消息的基本信息、内容（消息体）、详情参数和消费状态。

   在消费状态模块，您可以查看到消费该消息的 Group 以及消费状态，同时还可以在操作栏进行如下操作：

   - 重新发送：将消息重新发送到指定的客户端，如消息已消费成功，重新发送该消息可能导致消费重复。
   - 异常诊断：若消费异常，可以查看异常诊断信息。



5. 单击操作列的**查看消息轨迹**，或者在详情页单击 Tab 栏的**消息轨迹**，即可查看该消息的消息轨迹（详细说明请参见 消息轨迹查询结果说明）。



消费验证

在查询到某条消息后，您可以单击操作列的**消费验证**将该条消息发送到指定的客户端来验证该消息。**该功能可能会导致消息重复。**

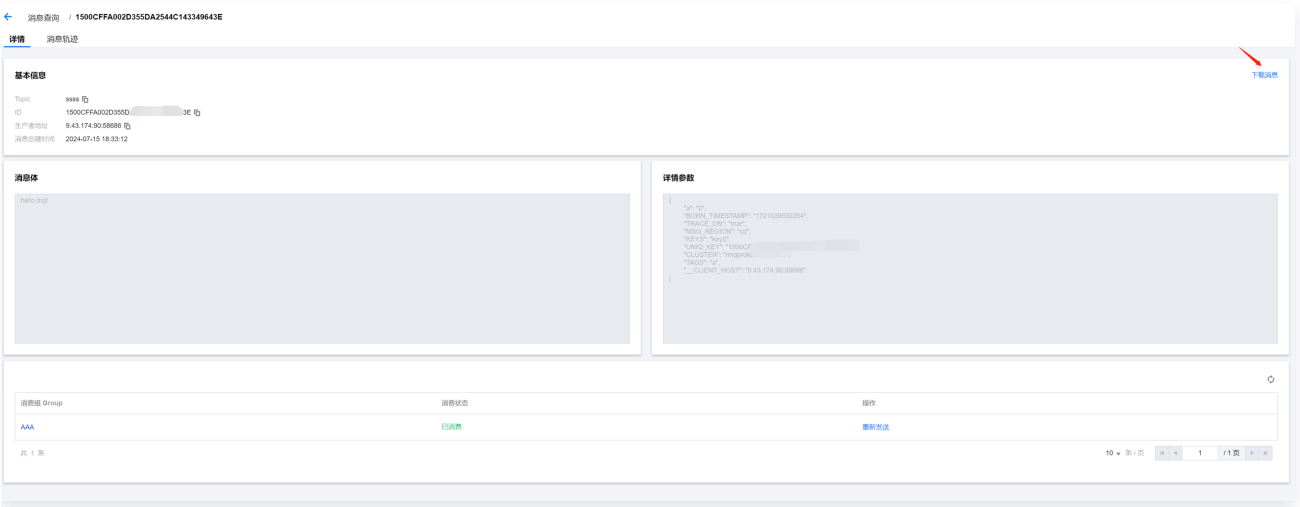
> **说明：**
> - 消费验证功能仅用于验证客户端的消费逻辑是否正常，并不会影响正常的收消息流程，因此消息的消费状态等信息在消费验证后并不会改变。
> - 消费验证功能目前暂不支持发送到 Python 和 C++ 客户端。

导出消息

在查询到某条消息后，您可以单击操作列的**导出消息**将该条消息的消息体，消息 Tag，消息 Key，消息生产时间和消费属性等信息。

下载消息

在消息详情页，您可以点击右上方的 **下载消息** 按钮，可以将单条消息的消息体和消息头保存到本地。



查询重试消息

最近更新时间：2024-11-21 15:29:32

为了给业务处理业务失败，给消息消费失败的情况兜底，保证消息生命周期的完整，RocketMQ 实现了消费失败后重试的策略。
如果您使用的是 RocketMQ 4.x 客户端，消息的重试次数以您在客户端内设置消息重试次数为准。
对于 RocketMQ 5.x 集群，您在创建 Group 时可以设置消息的重试次数，如果您使用的是 5.x 客户端，则重试次数以您在服务端设定的为准；如果您使用的是 4.x 客户端，则重试次数依旧以客户端内设置消息重试次数为准。

操作场景

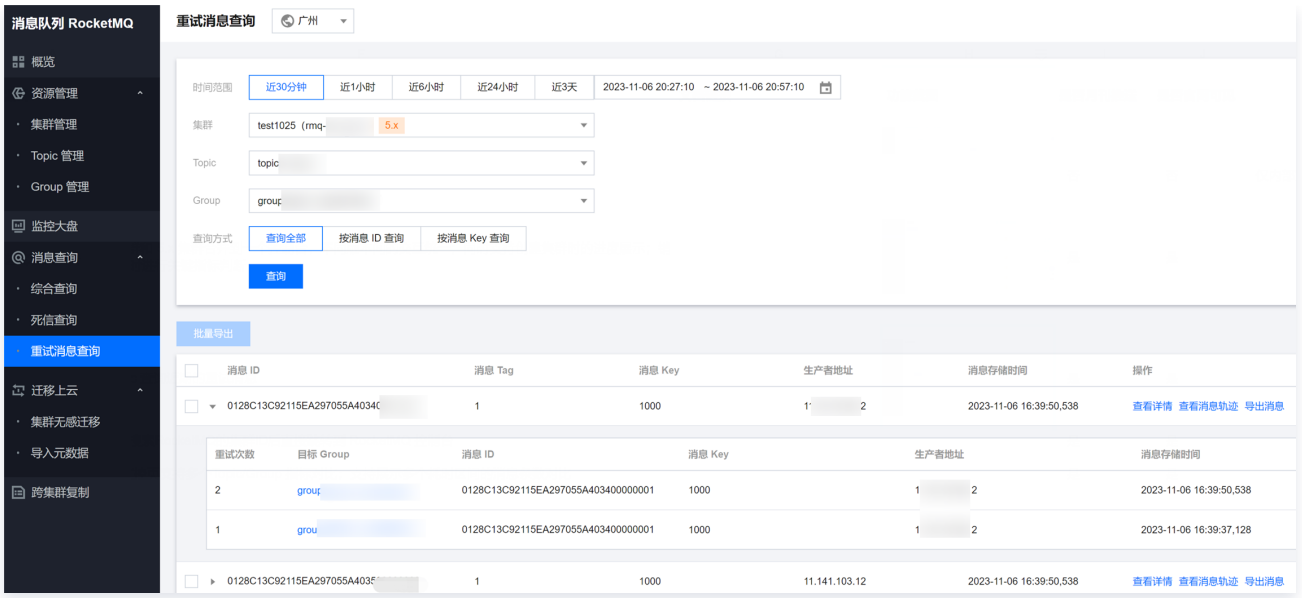
当您需要查看某个 Topic 下是否有重试消息时，您可以在 重试消息查询页 查询消息，并且可以展开查看消息每次重试的时间和生产者地址等信息，并且支持导出消息和查看消息的详细内容，如下图所示。

操作步骤

1. 登录 RocketMQ 控制台，在左侧导航栏单击 重试消息查询页 。
2. 在消息查询页面，选择好地域后根据页面提示输入查询条件。
   - 时间范围：选择需要查询的时间范围，支持近30分钟，近1小时，近6小时，近24小时，近3天和自定义时间范围。
   - 集群：选择需要查询的 Topic 所在的集群。
   - Topic：选择需要查询的 Topic。
   - Group：如果查询的集群为 5.x 集群，则需要选择该 Topic 下订阅的具体 Group。4.x 集群无需填写。
   - 查询方式：消息查询功能支持以下查询方式。
     - **查询全部：**该方式适合在对于重试消息的信息不明确的情况下使用，用于查询当前 Topic 下的全部重试消息。
     - **按消息 ID 查询：**该方式属于精确查询、速度快、精确匹配。
     - **按消息 Key 查询：**该方式属于模糊查询，适用于您没有记录消息 ID 但是设置了消息 Key 的场景。

> **说明：**
> 为了保证查询速度，当您选择 “查询全部” 时，服务端会按照时间先后查询最近的消息，但由于查询时间和展示的限制，可能无法快速定位到您需要排查的消息。建议您使用更加明确的搜索条件，如 消息 ID 和消息 Key 。

3. 单击**查询**，下方列表会展示所有查询到的结果并分页展示。



4. 查询完成后，您可以点击单条消息，查看当前消息的重试情况，如重试的次数和生产者地址等信息。您也可以单击操作栏的其他操作选项。

查询死信消息

最近更新时间：2024-10-14 14:21:42

操作场景

死信队列是一种特殊的消息队列，用于集中处理无法被正常消费的消息的队列。当消息在达到一定重试次数后仍未能被正常消费，TDMQ RocketMQ 版会判定这条消息在当前情况下无法被消费，将其投递至死信队列。
实际场景中，消息可能会由于持续一段时间的服务宕机，网络断连而无法被消费。这种场景下，消息不会被立刻丢弃，死信队列会对这种消息进行较为长期的持久化，用户可以在找到对应解决方案后，创建消费者订阅死信队列来完成对当时无法处理消息的处理。

查询限制

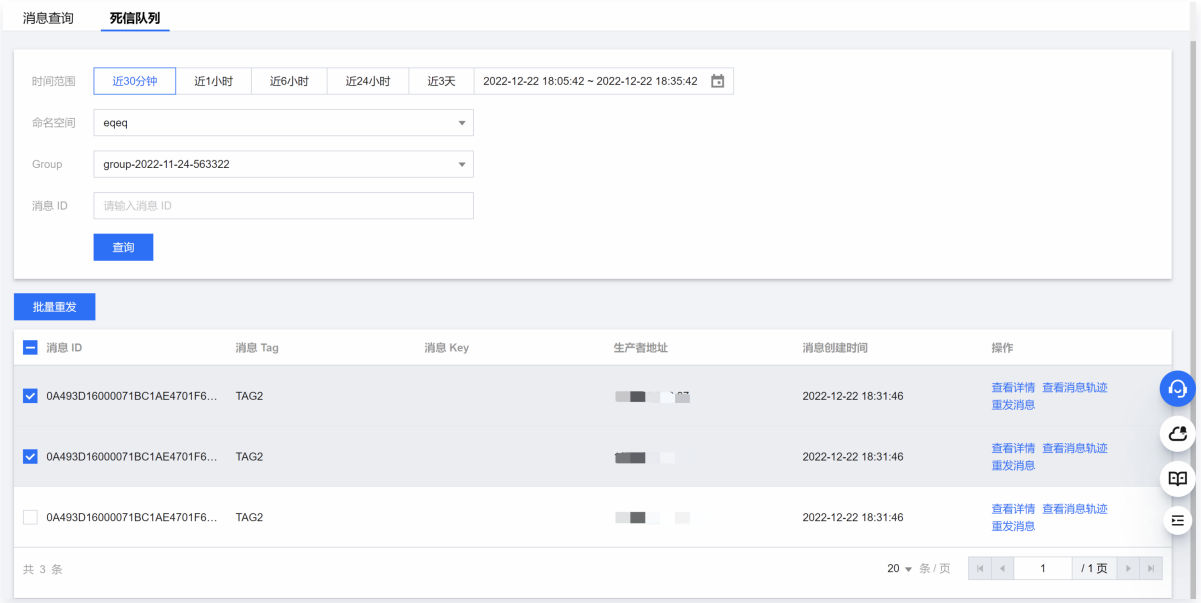
消息查询最多可以查询近3天的消息。

特性说明

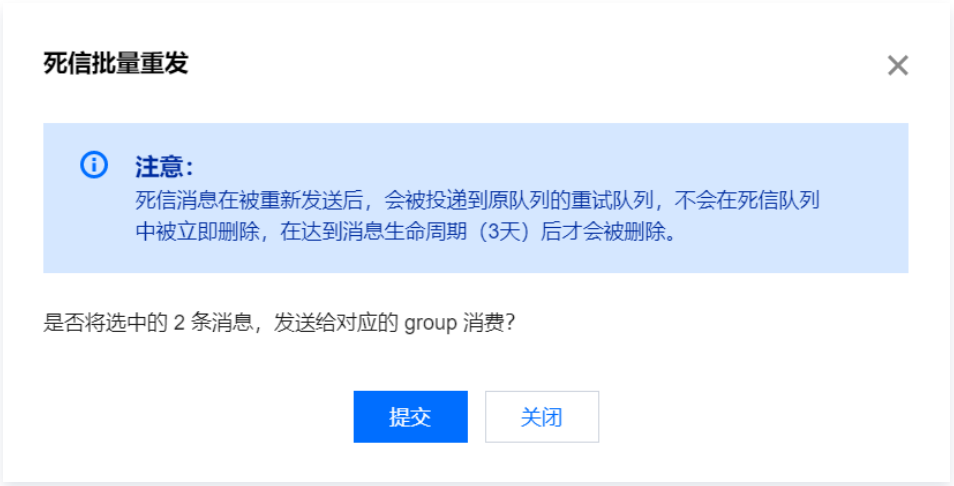
- 当消息被投递到死信队列后，消息不会再被消费者正常消费。消息查询最多可以查询近3天的消息，请尽量在死信消息产生的3天内进行处理，否则消息可能会被删除。
- 一个死信队列包含了对应的一个 Group 中所有 Topic 产生的所有死信消息。如果一个Group 中没有产生死信消息，则不会为其创建死信队列，也查询不到死信消息。

操作步骤

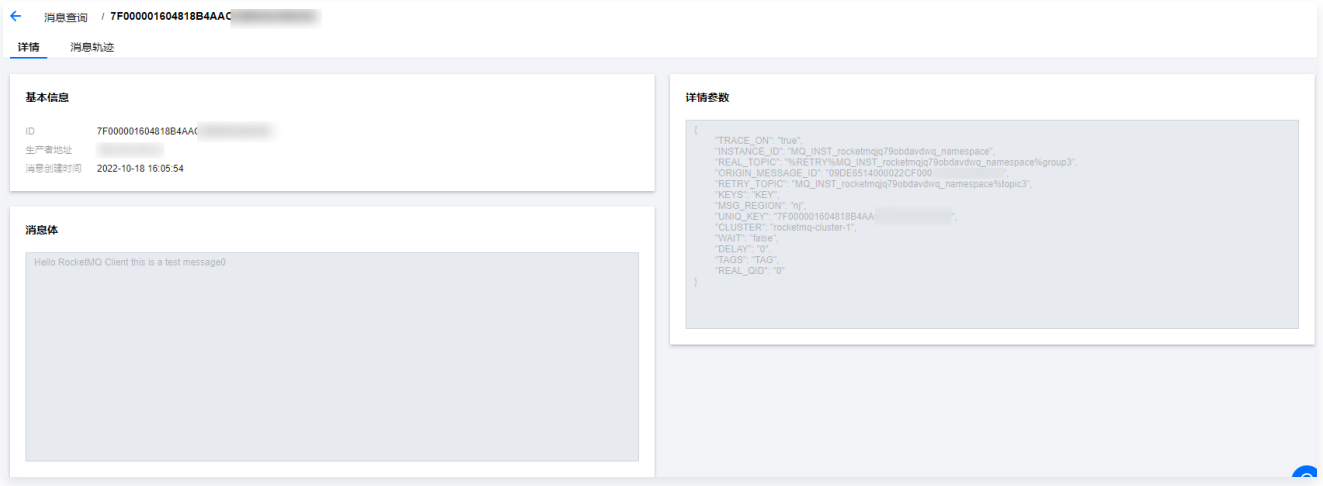
1. 登录 TDMQ RocketMQ 控制台，在左侧导航栏单击**死信查询**。
2. 在消息查询页面，选择好地域后根据页面提示输入查询条件。
    - 时间范围：选择需要查询的时间范围，支持近30分钟，近1小时，近6小时，近24小时，近3天和自定义时间范围。
    - 当前集群：选择需要查询的死信消息所在的集群。
    - Group：选择需要查询的死信消息所在的Group。
    - 消息 ID：非必填。
        - 不填写消息 ID：属于**模糊查询**。根据 Group ID 和死信消息产生的时间范围，批量查询该 Group ID在某段时间内产生的所有死信消息。
        - 填写消息 ID：属于**精确查询**。根据 Group ID与 Message ID 精确定位到任意一条消息。
3. 单击**查询**，下方列表会展示所有查询到的结果并分页展示。



4. 您可以勾选多条死信消息后单击左上角的**批量重发消息**将死信消息批量重新发送到原队列的重试队列，也可以单击某条消息操作列的**重发消息**将单条死信消息重新发送。死信消息在被重新发送后，会被投递到原队列的重试队列，不会在死信队列中被立即删除，在达到消息生命周期（3天）后才会被删除。

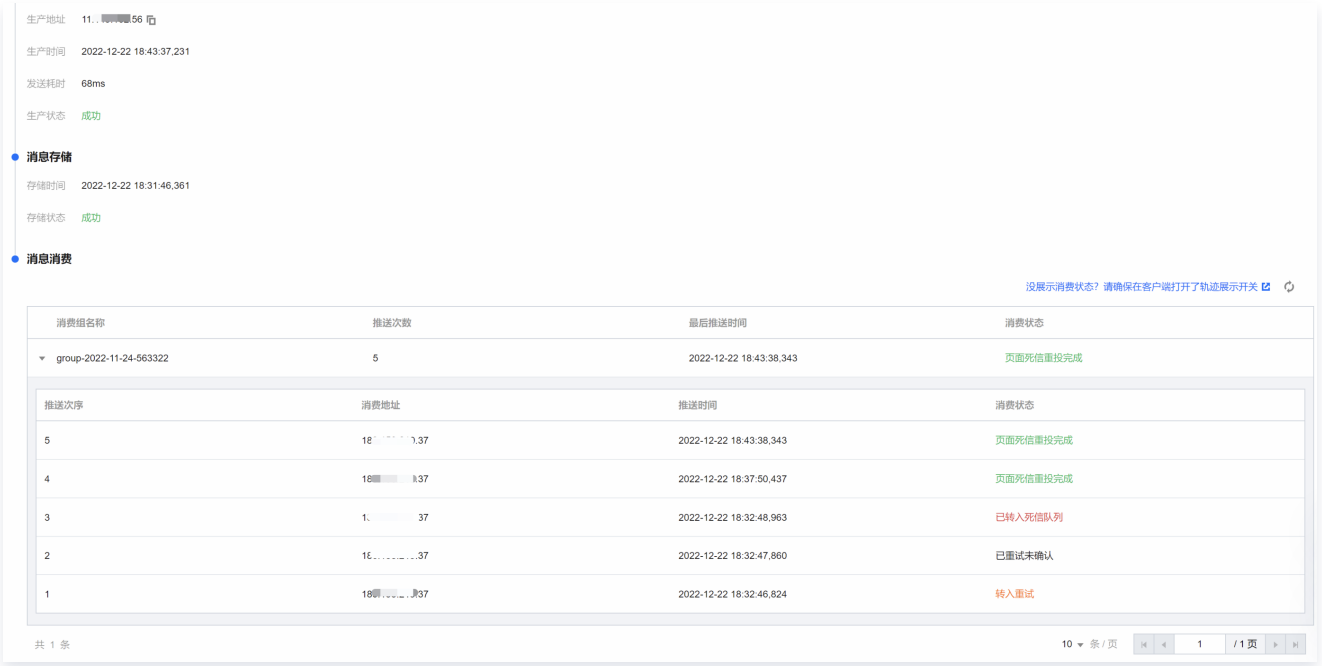


5. 找到您希望查看内容或参数的消息，单击操作列的**查看详情**，即可查看消息的基本信息、内容（消息体）以及参数。



6. 单击操作列的**查看消息轨迹**，或者在详情页单击 Tab 栏的**消息轨迹**，即可查看该消息的消息轨迹（详细说明请参见 消息轨迹查询结果说明）。

可以看到当死信消息被重新投递后，消费状态变成页面死信重投完成。



查询定时和延时消息

最近更新时间：2025-01-10 17:17:42

操作场景

定时/延时消息是 RocketMQ 中比较重要的消息类型。

- **定时消息**：消息在发送至服务端后，实际业务并不希望消费端马上收到这条消息，而是推迟到某个时间点被消费，这类消息统称为定时消息。
- **延时消息**：消息在发送至服务端后，实际业务并不希望消费端马上收到这条消息，而是推迟一段时间后再被消费，这类消息统称为延时消息。

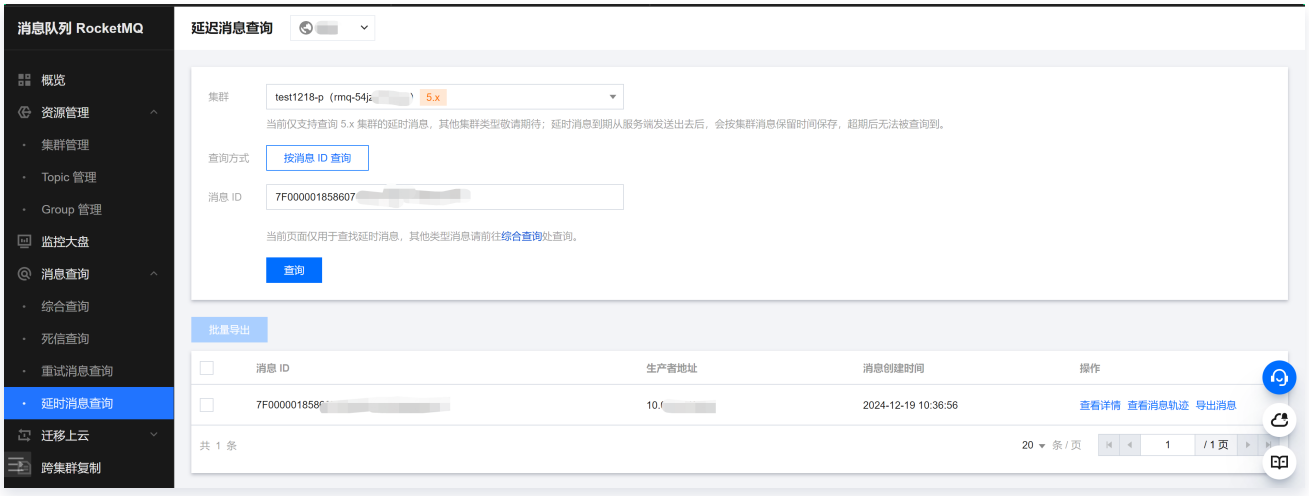
实际上，延时消息可以看成是定时消息的一种特殊用法，其实现的最终效果和定时消息是一致的。

查询限制

由于 RocketMQ 给消息设置有消息保留时间，即消费完成的消息会在服务端保留一段时间（体验版和基础版的时间为 3 天，专业版和铂金版支持调 Topic 维度的时间调整，支持7天及更长时间），在消息的保留时间内的延时和定时消息均可被查询到。
例如：假设某集群于 1月1日1时发送了一条延时10天的延时消息，集群的保留时间为三天，则1月11日1时，消息发送给消费者，假定消费者消费成功，则在1月14日1时前，均可以在当前页面查询到本条消息。

操作步骤

1. 登录 TDMQ RocketMQ 控制台，在左侧导航栏单击**延时消息查询**。
2. 在消息查询页面，选择好地域后根据页面提示输入查询条件。
   - 当前集群：选择需要查询的死信消息所在的集群。
   - 消息 ID：填写 Message ID 精确定位到任意一条消息。
3. 单击**查询**，下方列表会展示所有查询到的结果并分页展示。



4. 完成查询后，可以查看消息的创建和生产时间，和其他的消息查询结果类似，可以对消息进行查看轨迹和导出等操作。

消息轨迹与说明

最近更新时间：2024-10-14 17:29:01

消息轨迹记录了消息从生产端到 TDMQ RocketMQ 版服务端，最后到消费端的整个过程，包括各阶段的时间（精确到微秒）、执行结果、生产者 IP、消费者 IP 等。

前提条件

- 您已经参见 SDK 文档 部署好生产端和消费端服务，并在3天内有消息生产和消费。
- 如果您使用的是 5.0 及以上版本的 gRPC 客户端进行消息的生产和消费，则无需在客户端另行开启轨迹开关。
- 如果您使用的是 4.x 版本的客户端，或者 5.0 以上版本的 Remoting 客户端，则需要在客户端来设置开启消息轨迹功能，具体设置示例如下：
- 更多关于客户端的说明请参见 社区客户端说明 。

生产者设置

```
DefaultMQProducer producer = new DefaultMQProducer(namespace, groupName,
    // ACL权限
    new AclClientRPCHook(new SessionCredentials(AK, SK)), true, null);
```

Push 消费者设置

```
// 实例化消费者
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(NAMESPACE,groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)),
    new AllocateMessageQueueAveragely(), true, null);
```

Pull 消费者设置

```
DefaultLitePullConsumer pullConsumer = new
DefaultLitePullConsumer(NAMESPACE,groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)));
// 设置NameServer的地址
pullConsumer.setNamesrvAddr(NAMESERVER);
pullConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_LAST_OFFSET);
pullConsumer.setAutoCommit(false);
pullConsumer.setEnableMsgTrace(true);
pullConsumer.setCustomizedTraceTopic(null);
```

Spring Boot Starter 接入（2.2.2版本及以上）

```
package com.lazycece.sbac.rocketmq.messagemodel;

import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.annotation.MessageModel;
```

```
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;

/**
 * @author lazycece
 * @date 2019/8/21
 */
@Slf4j
@Component
public class MessageModelConsumer {

@Component
    @RocketMQMessageListener(
            topic = "topic-message-model",
            consumerGroup = "message-model-consumer-group",
            enableMsgTrace = true,
            messageModel = MessageModel.CLUSTERING)
    public class ConsumerOne implements RocketMQListener<String> {
        @Override
        public void onMessage(String message) {
            log.info("ConsumerOne: {}", message);
        }
    }

}
```

操作步骤

1. 登录 TDMQ RocketMQ 控制台，在左侧导航栏单击**消息查询**。
2. 在消息查询页面，选择好地域后根据页面提示输入查询条件。
   - 时间范围：选择需要查询的时间范围，支持近100条（默认按时间顺序展示最近的 100 条消息）、进30分钟、近1小时、近6小时、近24小时、近3天和自定义时间范围。
   - 集群：选择需要查询的 Topic 所在的集群。
   - Topic：选择需要查询的 Topic。
   - 查询方式：消息查询功能支持两种查询方式。
     - **查询全部：**该方式会展示所选时间范围内 Topic 中的全部消息。
     - **按消息 ID 查询：**该方式属于精确查询、速度快、精确匹配。
     - **按消息 Key 查询：**该方式属于模糊查询，适用于您没有记录消息 ID 但是设置了消息 Key 的场景。
3. 单击**查询**，下方列表会展示所有查询到的结果并分页展示。
4. 单击操作列的**查看消息轨迹**，或者在详情页单击 Tab 栏的**消息轨迹**，即可查看该消息的消息轨迹。

消息轨迹查询结果说明

消息轨迹查询出来的结果分为三段：消息生产、消息存储和消息消费。

消息生产

参数	说明
生产地址	对应生产者的地址以及端口。
生产时间	TDMQ RocketMQ 版服务端确认接收到消息的时间，精确到毫秒。
发送耗时	消息从生产端发送到 TDMQ RocketMQ 版服务端的时间消耗，精确到微秒。
生产状态	表示消息生产成功或失败，如果状态为失败一般是消息在发送过程中遇到了头部数据部分丢失，以上几个字段可能会为空值。

消息存储

参数	说明
存储时间	消息被持久化的时间。
存储耗时	消息从被持久化到 TDMQ RocketMQ 版服务端接收到确认信息的时间，精确到毫秒。
存储状态	表示消息持久化成功或失败，如果状态为失败则表明消息未落盘成功，可能由于底层磁盘损坏或无多余容量导致，遇见此类情况需尽快提工单咨询。

消息消费

消息消费是以列表形式呈现的，TDMQ RocketMQ 版支持集群消费和广播消费两种消费模式。
列表中展示的信息说明：

参数	说明
消费组名称	消费组的名称。
消费模式	消费组的消费模式，支持集群消费和广播消费两种模式。
推送次数	TDMQ RocketMQ 版服务端向消费者投递该消息的次数。
最后推送时间	TDMQ RocketMQ 版服务端最后一次向消费者投递该消息的时间。
消费状态	• 已推送未确认：TDMQ RocketMQ 版服务端已向消费者投递消息，未接收到消费者回复的确认消息。 • 已确认：消费者回复确认信息（ACK）到 TDMQ RocketMQ 版服务端，服务端接收到确认信息。 • 转入重试：已超时，服务端仍未接收到确认信息，将再次投递消息。 • 已重试未确认：TDMQ RocketMQ 版服务端已再次向消费者投递消息，未接收到消费者回复的确认消息。 • 已转入死信队列：消息经过一定重试次数后仍未能被正常消费，被投递至死信队列。 **说明：**如果消费模式为广播模式，则消费状态只有**已推送**一种。

单击订阅名称左方的右三角，查看服务端每次推送消息的详情。

参数	说明
推送次序	TDMQ RocketMQ 版服务端第几次向消费者投递该消息。
消费地址	收到消息的消费者地址及端口。

推送时间	TDMQ RocketMQ 版服务端向消费者投递消息的时间。
消费状态	• 已推送未确认：TDMQ RocketMQ 版服务端已向消费者投递消息，未接收到消费者回复的确认消息。 • 已确认：消费者回复确认信息（ACK）到 TDMQ RocketMQ 版服务端，服务端接收到确认信息。 • 转入重试：已超时，服务端仍未接收到确认信息，将再次投递消息。 • 已重试未确认：TDMQ RocketMQ 版服务端已再次向消费者投递消息，未接收到消费者回复的确认消息。 • 已转入死信队列：消息经过一定重试次数后仍未能被正常消费，被投递至死信队列。 • 页面死信重投完成：在死信队列重发页面上，用户已经将死信消息重新投递到原队列的重试队列中。

权限管理
主账号获取访问授权

最近更新时间：2024-10-14 17:52:01

操作背景

由于 RocketMQ 需要访问其他云产品的 API，所以需要授权 RocketMQ 创建服务角色。

前提条件

已注册腾讯云账号

> **说明：**
> 当您注册腾讯云账号后，系统默认为您创建了一个主账号，用于快捷访问腾讯云资源。
> 操作步骤

1. 登录 TDMQ 控制台，在左侧导航栏选择 **RocketMQ > 集群管理**，单击**新建集群**。
2. 在购买集群页面进行网络配置时，选择好私有网络后，勾选**授权将新购集群接入点域名绑定至上述的私有网络（VPC）**时，将会弹出需要您授权的提示弹窗。

3. 单击**前往授权**，进入访问管理控制台，单击**同意授权**，则为 TDMQ RocketMQ 授权服务角色访问您的其他云服务资源。

4. 授权完成后，您可以继续创建 RocketMQ 集群并使用相关服务。

子账号获取访问授权
授予子账号访问权限

最近更新时间：2024-10-14 14:20:52

CAM 基本概念

主账号通过给子账号绑定策略实现授权，策略设置可精确到 [API，资源，用户/用户组，允许/拒绝，条件] 维度。

账号体系

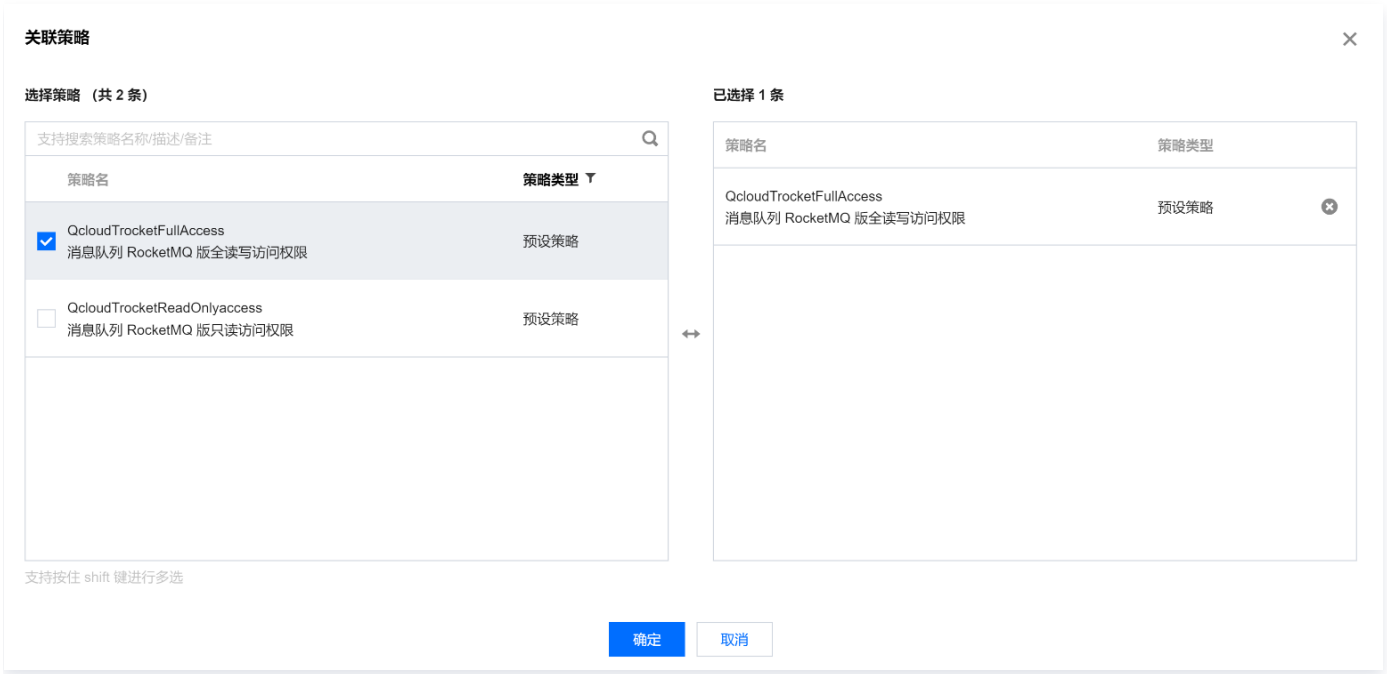
- **主账号**：拥有腾讯云所有资源，可以任意访问其任何资源。
- **子账号**：包括子用户和协作者。
  - **子用户**： 由主账号创建，完全归属于创建该子用户的主账号。
  - **协作者**：本身拥有主账号身份，被添加作为当前主账号的协作者，则为当前主账号的子账号之一，可切换回主账号身份。
- **身份凭证**：包括登录凭证和访问证书两种，**登录凭证**指用户登录名和密码，**访问证书**指云 API 密钥（SecretId 和 SecretKey）。

资源与权限

- **资源**：资源是云服务中被操作的对象，如一个云服务器实例、COS 存储桶、VPC 实例等。
- **权限**：权限是指允许或拒绝某些用户执行某些操作。默认情况下，**主账号拥有其名下所有资源的访问权限**，而**子账号没有主账号下任何资源的访问权限**。
- **策略**：策略是定义和描述一条或多条权限的语法规范。**主账号**通过将**策略关联**到用户/用户组完成授权。

子账号使用 RocketMQ

为了保证子账号能够顺利使用 RocketMQ，主账号需要对子账户进行授权。
主账号登录 访问管理控制台，在子账号列表中找到对应的子账号，单击操作列的**授权**。
RocketMQ 为子账号提供了两种预设策略：QcloudTrocketReadOnlyaccess 和 QcloudTrocketFullAccess，前者仅能查看控制台的相关信息，后者可以在产品控制台进行读写等相关操作。



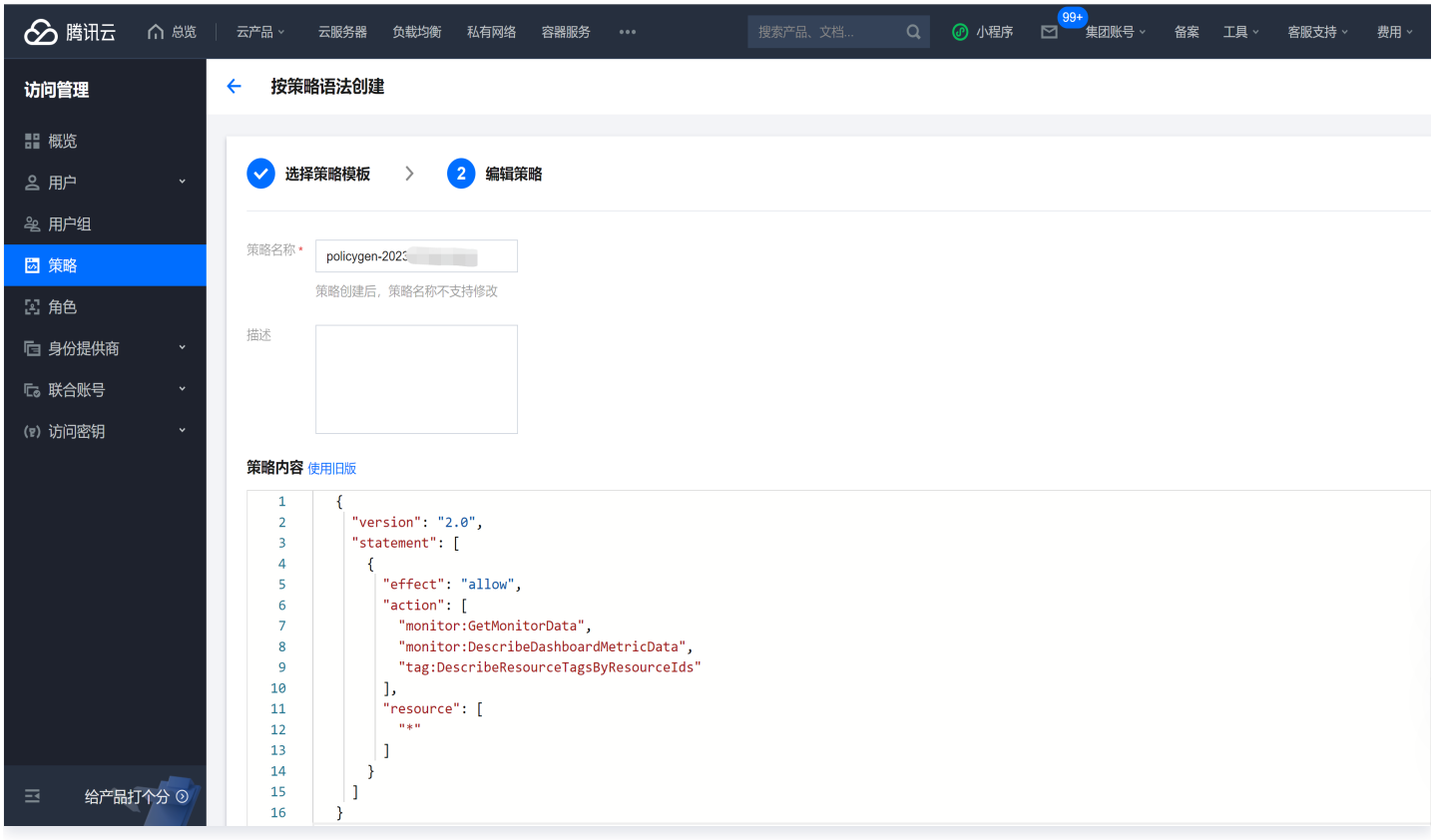
除了以上的预设策略外，为了方便使用，主账号还需要根据实际需要，授予子账号合适的其他云产品调用权限。RocketMQ 使用中涉及到以下云产品的相应接口权限：

云产品	接口名	接口作用	对应在 RocketMQ 中的作用
腾讯云可观测平台（Monitor）	GetMonitorData	查询指标监控数据	查看控制台展示的相应监控指标
腾讯云可观测平台（Monitor）	DescribeDashboardMetricData	查询指标监控数据	查看控制台展示的相应监控指标
资源标签（Tags）	DescribeResourceTagsByResourceIds	查询资源标签	查看集群的资源标签

为了给子账号增加上述权限，主账号还需要在 访问管理控制台 的策略页面，进行新建自定义策略操作。单击按策略语法新建后，选择空白模板，输入以下策略语法：

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "monitor:GetMonitorData",
        "monitor:DescribeDashboardMetricData",
        "tag:DescribeResourceTagsByResourceIds"
      ],
      "resource": [
        "*"
      ]
    }
  ]
```

```
    }
```



创建完成策略后，在操作列，将创建好的策略关联给子账号即可，如下图所示：

相关文档

- 操作级授权
- 资源级授权
- 标签级授权

授予子账号操作级权限

最近更新时间：2024-10-14 14:20:52

操作场景

本文指导您使用腾讯云主账号为子账号进行操作级授权，您可以根据实际需要，为子账号授予不同的读写权限。

操作步骤

授予全量读写权限

> **说明**
> 授予子账号全量读写权限后，子账号将拥有对主账号下**所有资源**的**全读写能力**。

1. 使用主账号登录 访问管理控制台 。
2. 在左侧导航栏，单击**策略**，进入策略管理列表页。
3. 在右侧搜索栏中，输入 **QcloudTrocketFullAccess** 进行搜索。

4. 在搜索结果中，单击 **QcloudTrocketFullAccess** 的**关联用户/组**，选择需要授权的子账号。

5. 单击**确定**完成授权。该策略会显示在用户的策略列表中。

授予只读权限

> **说明**
> 授予子账号只读权限后，子账号将拥有对主账号下**所有资源**的**只读能力**。

1. 使用主账号登录 访问管理控制台 。

2. 在左侧导航栏，单击**策略**，进入策略管理列表页。
3. 在右侧搜索栏中，输入 **QcloudTrocketReadOnlyAccess** 进行搜索。

4. 在搜索结果中，单击 **QcloudTrocketReadOnlyAccess** 的**关联用户/组**，选择需要授权的子账号。

5. 单击**确定**完成授权。该策略会显示在用户的策略列表中。

其他授权方式

- 资源级授权
- 标签级授权

授予子账号资源级权限

最近更新时间：2024-10-14 17:32:05

操作场景

该任务指导您使用主账号给子账号进行资源级授权，得到权限的子账号可以获得对某个资源的控制能力。

操作前提

- 拥有腾讯云主账号，且已经开通腾讯云访问管理服务。
- 主账号下至少有一个子账号，且已根据 子账号获取访问授权 完成授权。
- 至少拥有一个 RocketMQ 实例。

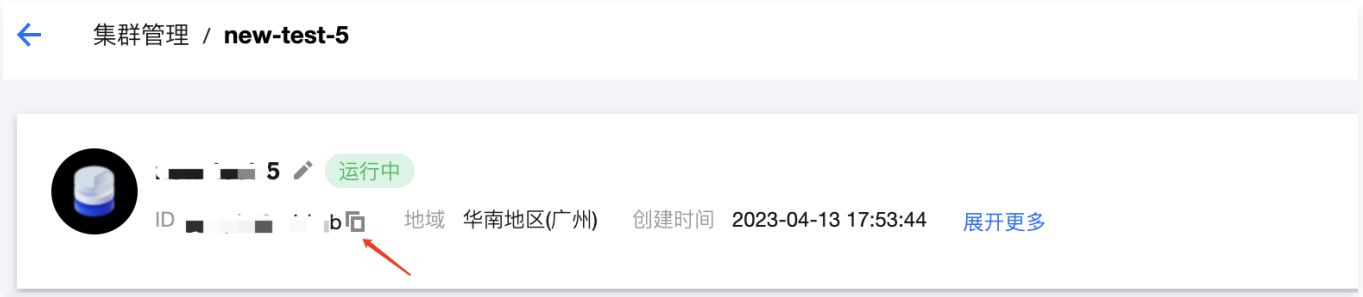
操作步骤

您可通过访问管理控制台的策略功能，将主账号拥有的 RocketMQ 资源授权给子账号，详细 **RocketMQ 资源授权给子账号**操作如下。本示例以授权一个集群资源给子账号为例，其他类型资源操作步骤类似。

步骤1：获取 RocketMQ 集群的资源 ID

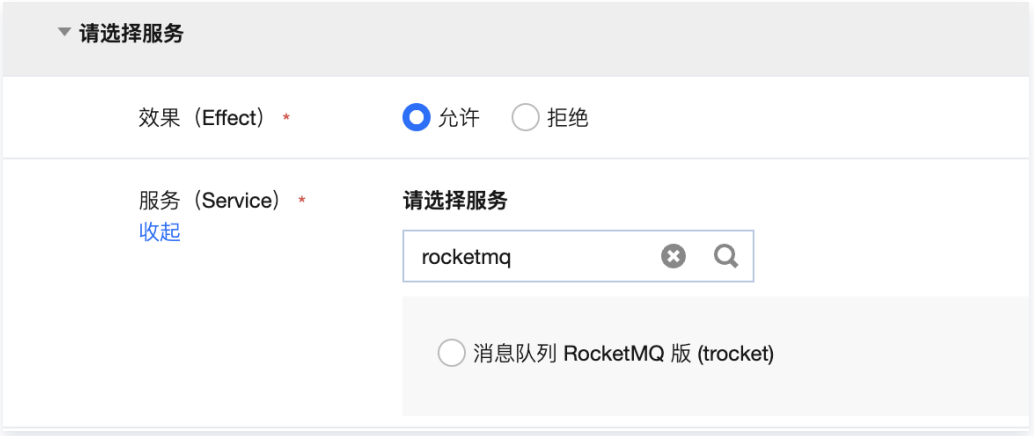
1. 使用**主账号**登录到 消息队列 RocketMQ 版控制台，选择已有的集群实例并单击进入详情页。

2. 在**基本信息**中，字段 ID 即为当前 RocketMQ 集群的 ID。



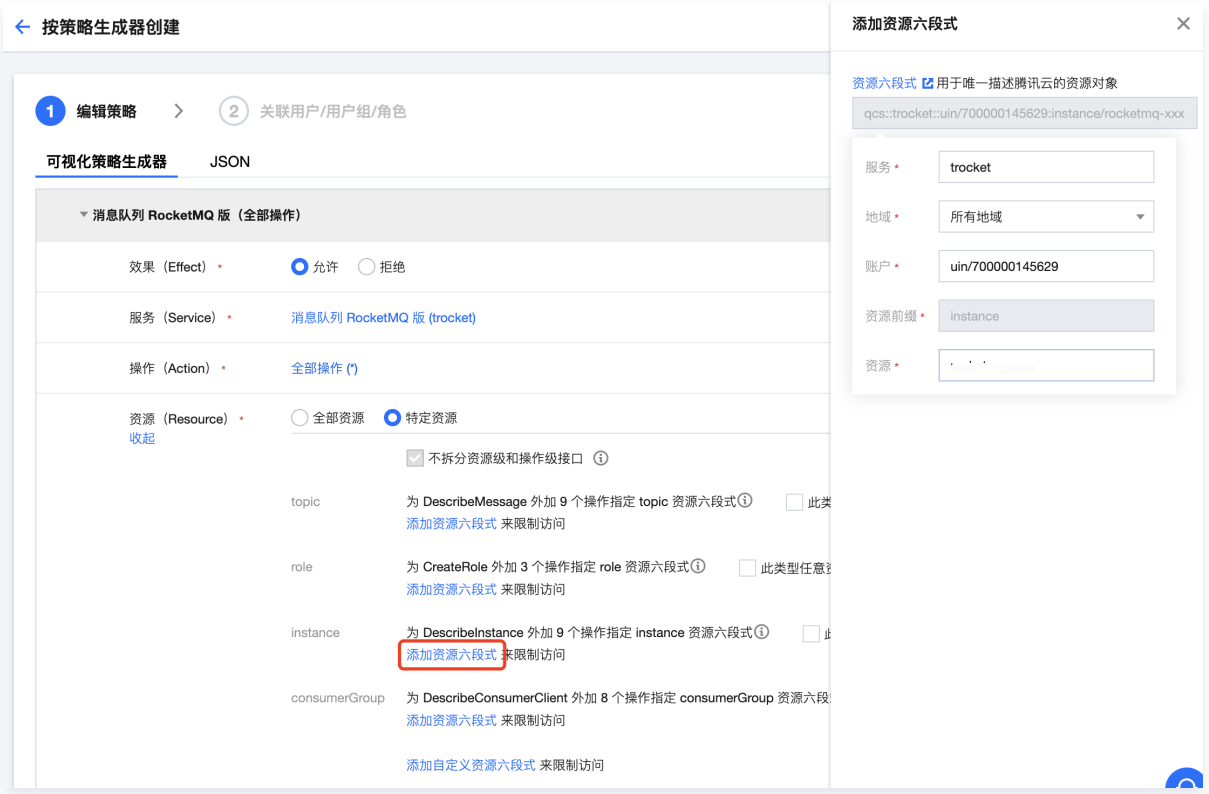
步骤2：新建授权策略

1. 进入 访问管理控制台，单击左侧导航栏的 **策略**。
2. 单击**新建自定义策略**，选择**策略生成器创建**。
3. 在可视化策略生成器中，保持**效果**为**允许**，在**服务**中输入 rocketmq 进行筛选，在结果中选择**消息队列 RocketMQ 版 (trocket)**。



4. 在**操作**中选择**全部操作**，您也可以根据自己的需要选择操作类型。

5. 在**资源**中选择**特定资源**，您可以勾选右侧**此类型任意资源（授权所有该类资源）**，或者并单击**添加资源六段式（授权特定资源）**。
6. 在弹出的侧边对话框中的**资源**中，填入要授权的资源**的 ID**，获取流程可参见 步骤1。



7. 单击下一步，按需填写策略名称。

8. 单击**选择用户**或**选择用户组**，可选择需要授予资源权限的用户或用户组。

9. 单击**完成**，授予资源权限的子账号就拥有了访问相关资源的能力。

其他授权方式

- 操作级授权
- 标签级授权

附录

支持资源级授权的 API 列表

TDMQ 支持资源级授权，您可以指定子账号拥有特定资源的接口权限。
支持资源级授权的接口列表如下：

API名	API描述	资源类型	资源六段式示例
CreateConsumerGroup	创建消费组	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/*
CreateInstance	创建实例	instan	qcs::trocket:${region}:uin/${uin}:instance/*

		ce	
CreateInstanceEndpoint	创建接入点	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
CreateRole	添加角色	role	qcs::trocket:${region}:uin/${uin}:role/${instanceId}/*
CreateTopic	创建主题	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/*
DeleteConsumerGroup	删除消费组	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
DeleteInstance	删除实例	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
DeleteInstanceEndpoint	删除接入点	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
DeleteRole	删除角色	role	qcs::trocket:${region}:uin/${uin}:role/${instanceId}/${role}
DeleteTopic	删除主题	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}
DescribeConsumerClient	查询消费者客户端详情	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
DescribeConsumerClientList	查询消费组下的客户端连接	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
DescribeConsumerGroup	查询消费组详情	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
DescribeConsumerGroupList	查询消费组列表	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
DescribeInstance	查询实例	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
DescribeInstanceList	查询实例列表	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
DescribeInstanceTopUsages	获取实例资源消耗排行	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
DescribeMessage	查询消息	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}

DescribeMessageList	查询消息列表	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}
DescribeMessageTrace	查询消息轨迹	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}
DescribeRoleList	查询角色列表	role	qcs::trocket:${region}:uin/${uin}:role/${instanceId}/${role}
DescribeTopic	查询主题详情	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}
DescribeTopicList	查询主题列表	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}
DescribeTopicListByGroup	根据消费组获取主题列表	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
DescribeTopicStatisticalList	获取指定实例下主题类型和个数	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
ExportMessage	导出消息	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}
ModifyConsumerGroup	修改消费组属性	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
ModifyInstance	修改实例	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
ModifyInstanceEndpoint	修改接入点	instance	qcs::trocket:${region}:uin/${uin}:instance/${instanceId}
ModifyRole	修改角色	role	qcs::trocket:${region}:uin/${uin}:role/${instanceId}/${role}
ResetConsumerGroupOffset	重置消费位点	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerGroup/${instanceId}/${consumerGroup}
SendMessage	发送消息	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}
VerifyMessageConsumption	消息消费验证	topic	qcs::trocket:${region}:uin/${uin}:topic/${instanceId}/${topic}

授予子账号标签级权限

最近更新时间：2024-10-11 10:33:52

操作场景

该任务指导您通过标签的鉴权方式，使用主账号给子账号进行某标签下资源的授权。得到权限的子账号可以获得具有相应标签下资源的控制能力。

操作前提

- 拥有腾讯云主账号，且已经开通腾讯云访问管理服务。
- 主账号下至少有一个子账号，且已根据 子账号获取访问授权 完成授权。
- 至少拥有一个 RocketMQ 集群资源实例。
- 至少拥有一个**标签**，若您没有，可以前往 标签控制台 > **标签列表**进行新建。

操作步骤

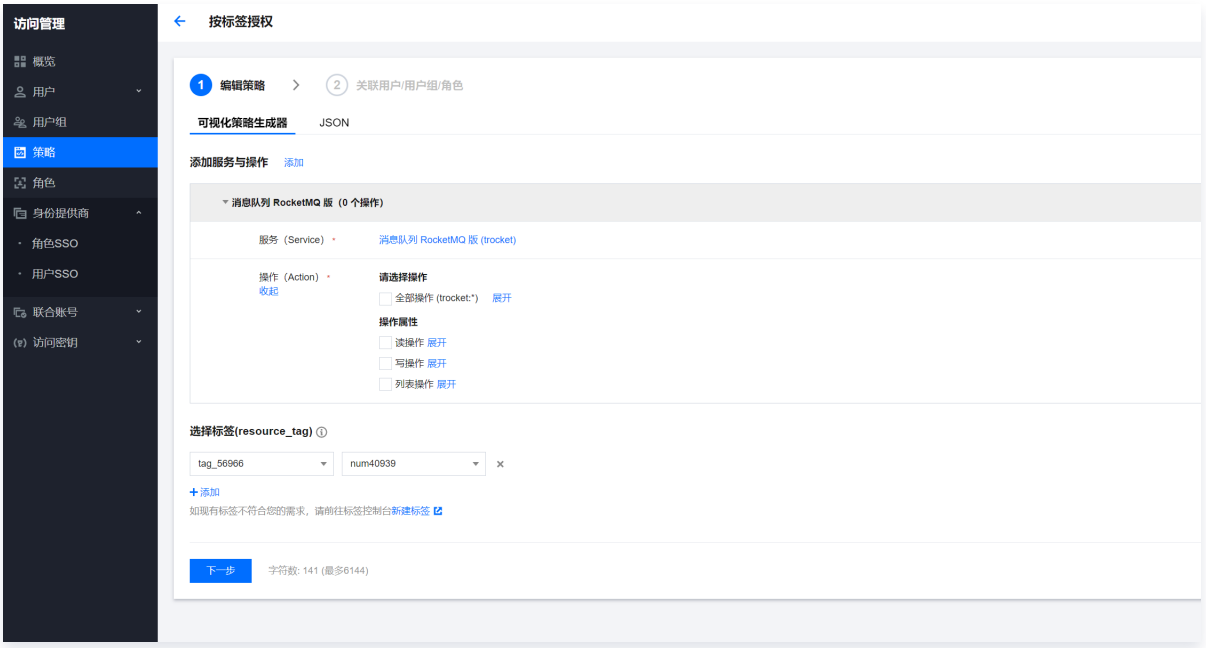
您可通过访问管理控制台的策略功能，将主账号拥有的、已经绑定标签的 RocketMQ 资源，通过**按标签授权**的方式授予子账号这些资源的读写权限，详细**按标签授予资源权限给子账号**的操作如下。

步骤 1：为资源绑定标签

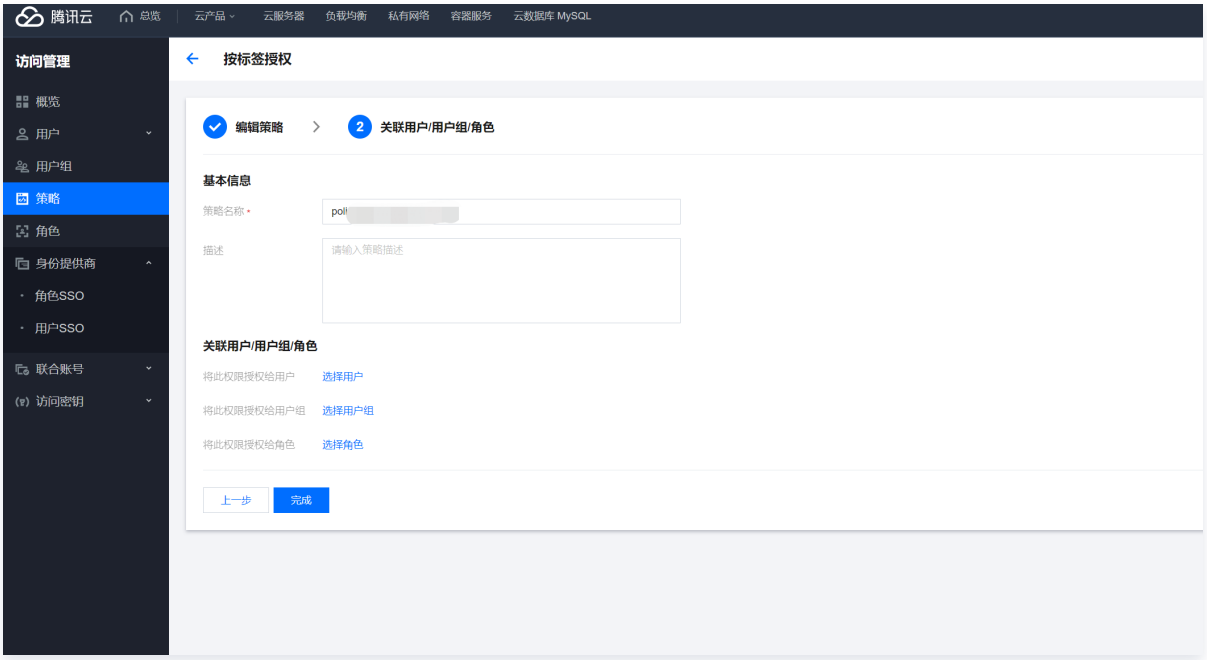
1. 使用**主账号**登录到 消息队列 RocketMQ 控制台，进入集群管理页面。
2. 勾选目标集群，单击左上角的**编辑资源标签**，为集群绑定好资源标签。

步骤 2：按标签授权

1. 进入 访问管理控制台，单击左侧导航栏的 **策略**。
2. 单击**新建自定义策略**，选择**按标签授权**。
3. 在可视化策略生成器中，在**服务**中输入 rocketmq 进行筛选，在结果中选择 **消息队列 RocketMQ 版（trocket）**，在**操作**中选择**全部操作**，您也可以根据需要选择相应的操作。



4. 单击下一步，按需填写策略名称。
5. 单击**选择用户**或**选择用户组**，可选择需要授予资源权限的用户或用户组。

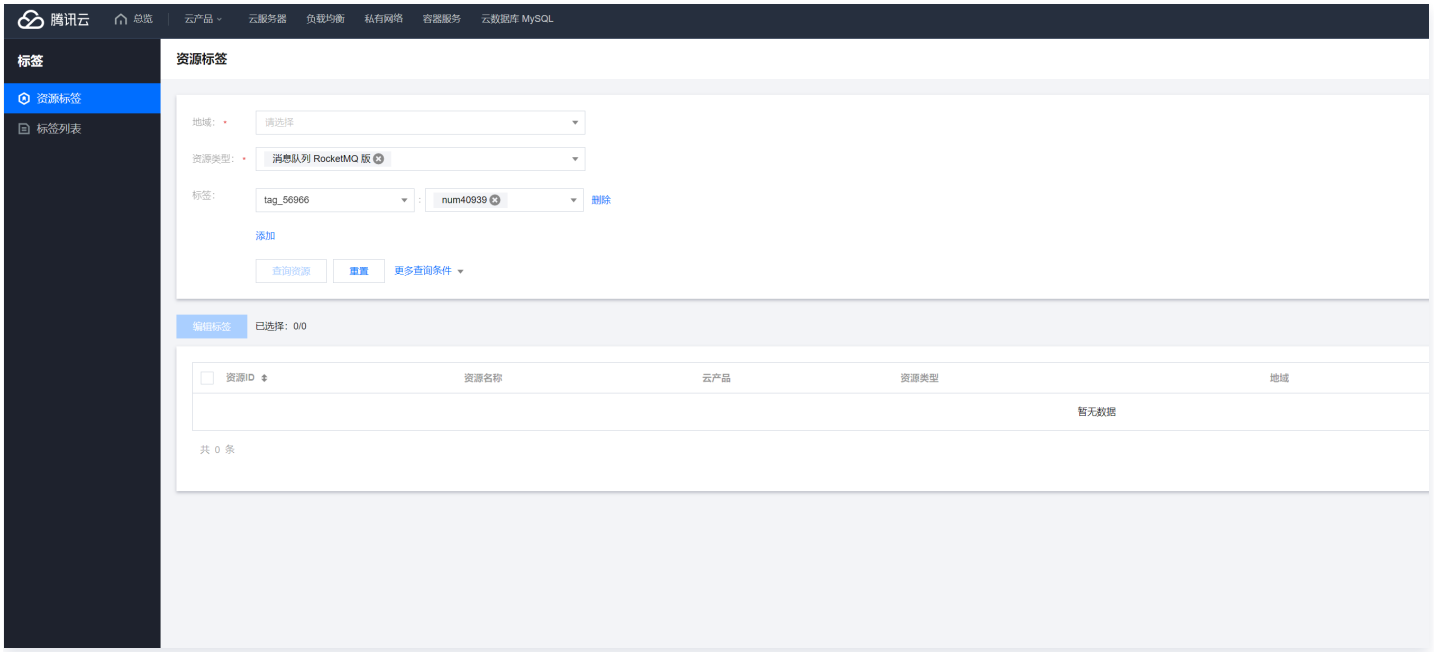


6. 单击**完成**，相关子账号就能够根据策略控制指定标签下的资源。

统一管理资源标签

您也可以在 **标签控制台** 统一管理资源标签，详细操作如下：

1. 登录腾讯云 **标签控制台**。
2. 在左侧导航栏选择资源标签，根据需要选择查询条件，并在 **资源类型** 中选择 **消息队列 RocketMQ 版 > RocketMQ 实例**。
3. 单击**查询资源**。
4. 在结果中勾选需要的 资源，单击**编辑标签**，即可批量进行标签的绑定或解绑操作。



其他授权方式

- 操作级授权
- 资源级授权

标签管理

最近更新时间：2025-03-03 16:19:32

操作场景

标签是腾讯云提供的用于标识云上资源的标记，是一个键-值对（Key-Value）。标签可以帮助您从各种维度（例如业务，用途，负责人等）方便的对 TDMQ RocketMQ 版资源进行分类管理。

> **说明：**
> 腾讯云不会使用您设定的标签，标签仅用于您对 TDMQ RocketMQ 版资源的管理。

使用限制

使用标签时，需注意以下限制条件：

限制类型	限制说明
数量限制	每个云资源允许的最大标签数是50。
标签键限制	• `qcloud`、`tencent`、`project` 开头为系统预留标签键，禁止创建。 • 只能为 数字、字母、`+=.@-`，且标签键长度最大为255个字符。
标签值限制	只能为 空字符串或数字、字母、`+=.@-`，且标签值最大长度为127个字符。

操作方法及案例

案例描述

案例：某公司在腾讯云上拥有6个 TDMQ RocketMQ 版集群，这6个集群的使用部门、业务范围以及负责人的信息如下：

队列 ID	使用部门	业务范围	负责人
rocketmq-qzga74ov5gw1	电商	营销活动	张三
rocketmq-qzga74ov5gw2	电商	营销活动	王五
rocketmq-qzga74ov5gw3	游戏	游戏 A	李四
rocketmq-qzga74ov5gw4	游戏	游戏 B	王五
rocketmq-qzga74ov5gw5	文娱	后期制作	王五
rocketmq-qzga74ov5gw6	文娱	后期制作	张三

以 rocketmq-qzga74ov5gw1 为例，我们可以给该实例添加以下三组标签：

标签键	标签值
dept	ecommerce

business	mkt
owner	zhangsan

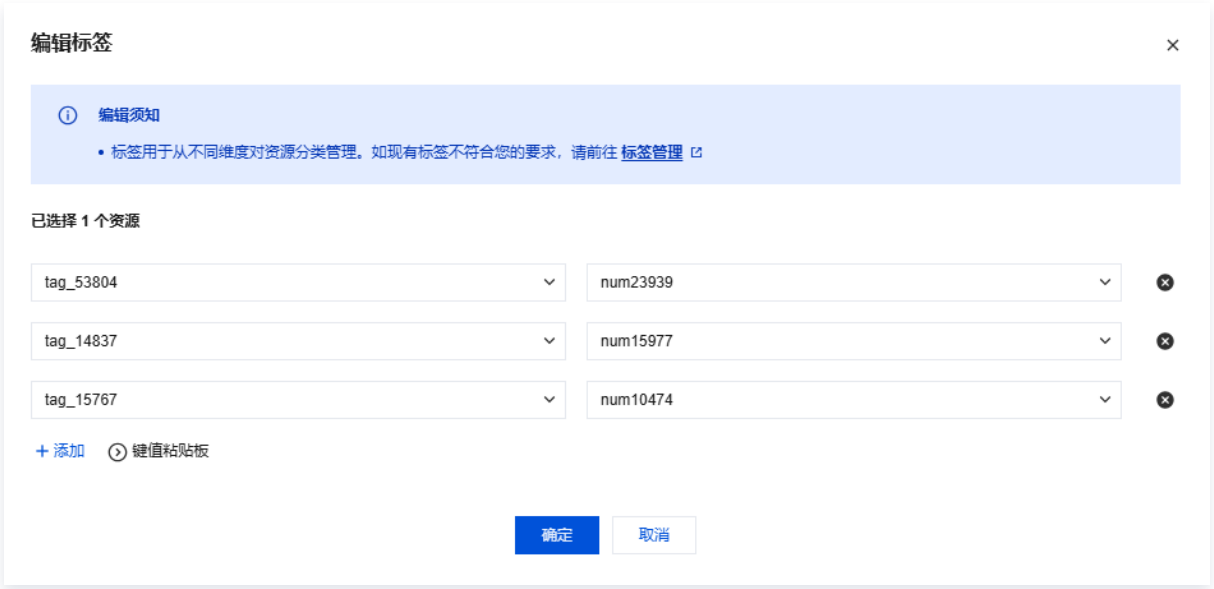
类似的，其他队列资源也可以根据其使用部门、业务范围和负责人的不同设置其对应的标签。

在 TDMQ RocketMQ 版控制台设置标签

以上文场景为例，当您完成标签键和标签值的设计后，可以登录 TDMQ RocketMQ 版控制台进行标签的设置。

1. 登录 RocketMQ 控制台 。
2. 在集群管理列表页面，选择好地域后，勾选需要编辑标签的集群，单击页面上方的**编辑资源标签**。

3. 在弹出的 “编辑标签” 窗口中设置标签。



> **说明：**
> 如果现有标签不符合您的要求，请前往 标签管理 新建标签。

4. 单击**确定**，系统出现修改成功提示，在集群的资源标签栏可查看与之绑定的标签。

通过标签键筛选资源

当您希望筛选出绑定了相应标签的集群时，可通过以下操作进行筛选。

1. 在页面右上方搜索框中，选择**标签**。
2. 在**标签：**后弹出的窗口中选择您要搜索的标签，单击**确定**进行搜索。
   例如：选择 `标签：owner:zhangsan` 可筛选出绑定了标签键 `owner:zhangsan` 的集群。

编辑标签

1. 在集群管理列表页面，选择好地域后，勾选需要编辑标签的集群，单击页面上方的**编辑资源标签**。

说明：

最多支持对20个资源进行标签的批量编辑操作。

2. 在弹出的 “编辑标签” 窗口中，根据实际需求进行添加、修改或者删除标签。

消息跨集群复制

最近更新时间：2025-01-10 17:17:42

操作场景

TDMQ RocketMQ 支持客户在两个集群之间同步消息（同个地域或不同地域间；4.x 集群之间，5.x 集群之间，4.x 和 5.x 集群间，三种情况均支持），您可以按照 Topic 维度，把集群 A 的某个 Topic 的消息复制到集群 B 的某个 Topic。在进行某个 Topic 的消息复制时，RocketMQ 支持按照特定的条件进行过滤（例如 Tag 或者 SQL 表达式），支持复制任务的任意启停，并且支持通过监控查看复制任务的进度和健康程度。

计费规则

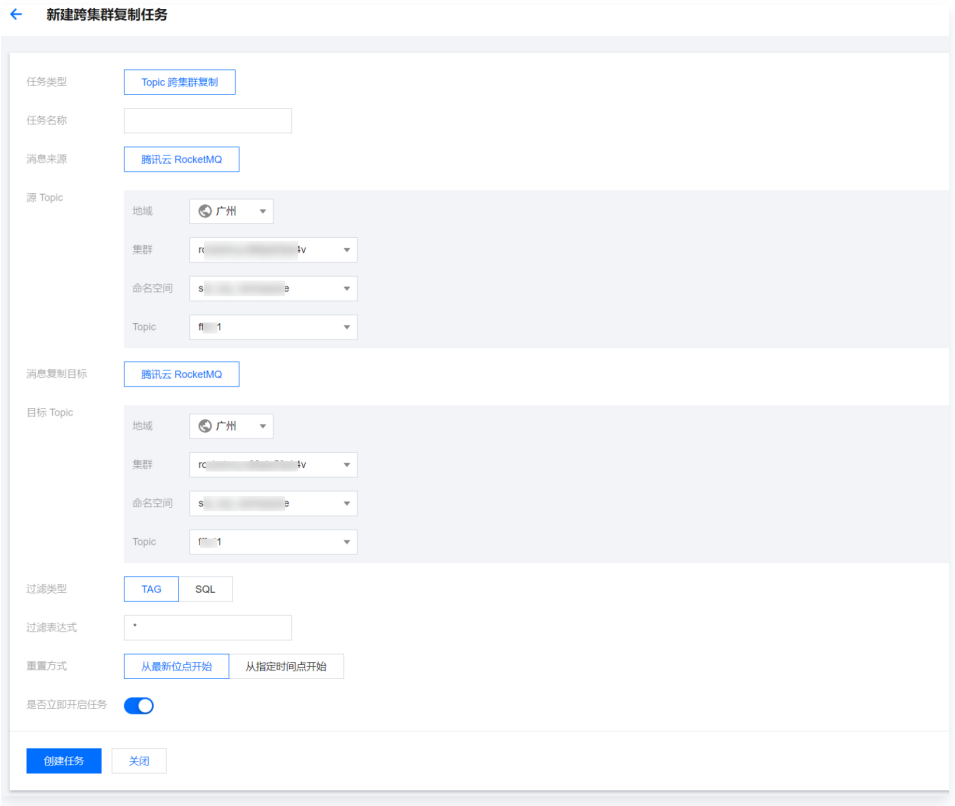
消息跨集群复制功能当前免费；在开始收费前，腾讯云会提前一个月多次通过站内信、短信和邮件等形式通知客户。

操作步骤

创建任务

进入 消息队列 RocketMQ 版控制台，单击左侧导航栏**跨集群复制 > 新建任务**， 按照要求填写以下字段：

- 任务名称：200字符以内，只能包含 中文、数字、字母、“-”和“_”；
- 源 Topic：通过下拉依次选择地域、集群、命名空间和 Topic，如果找不到需要的集群或 Topic 可以在集群列表页进行新建。
- 目标 Topic：通过下拉依次选择地域、集群、命名空间和 Topic，如果找不到需要的集群或 Topic 可以在集群列表页进行新建。
- 过滤类型：支持 TAG 过滤和 SQL 过滤两个方式。
- 复制起始位置：支持从最新的位点开始复制或者指定时间点开始复制。
- 是否立即开启任务：如果打开开关，在任务创建完成后就按照当前任务的配置进行复制。



单击**创建任务**后，会跳转到任务列表页，在任务初始化后即创建完成。

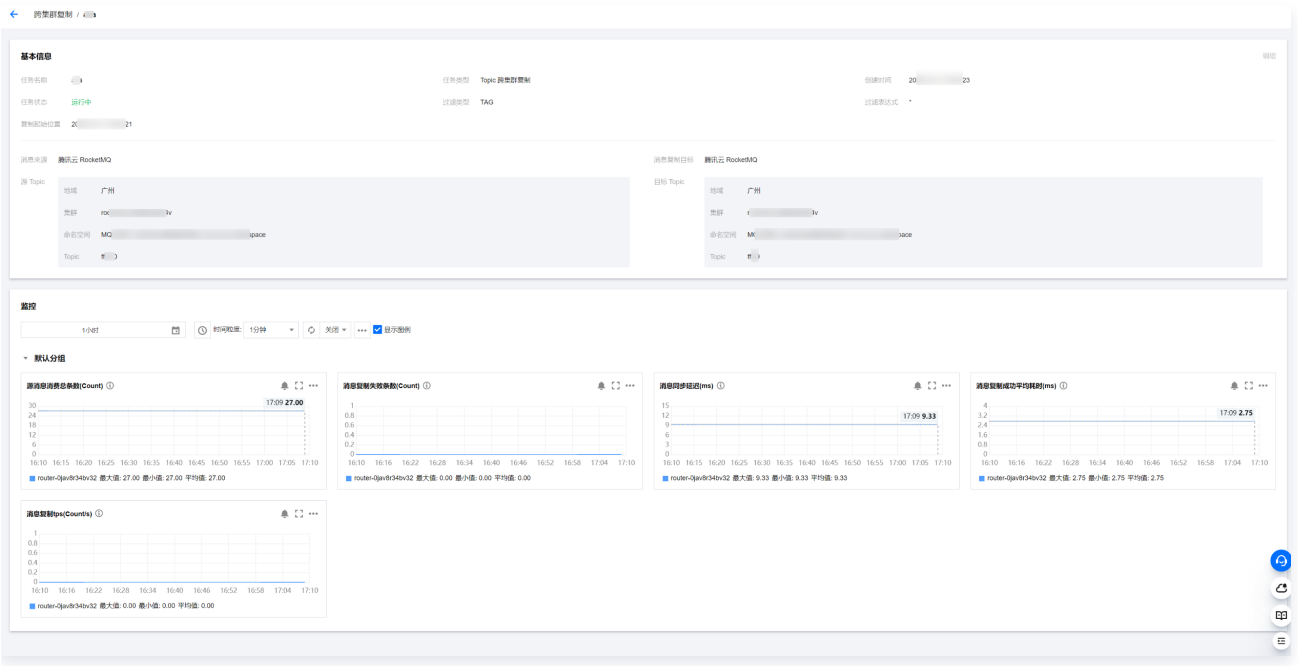
您创建的复制任务是单向的，即如果您创建一个 Topic A 到 Topic B 的复制任务，Topic A 的消息会自动复制到 Topic B；如果您需要双向的复制任务，您需要再次新建一个从 Topic B 到 Topic A 的复制任务。

复制任务

为了方便消息复制任务的创建，您也可以通过复制已有任务的配置快速创建新的任务。
进入消息队列 RocketMQ 版控制台，单击左侧导航栏**跨集群复制**,选择已创建的任务，单击操作列的**复制任务**，创建任务的页面会自动填充复制任务的相同参数，只需要简单修改后即可创建新的任务（目前控制台不得创建完全相同配置的任务）。

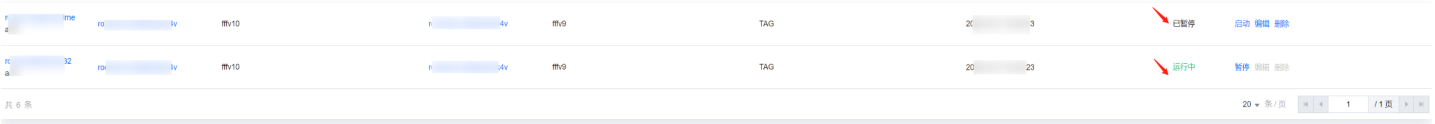
查看任务详情

在任务创建完成后，您可以在任务的列表页看到新增的复制任务，同时可以快速查看任务的状态。单击操作列的**启动/暂停**可以快速的开启和暂停任务。
进行中的任务不能修改配置信息，如果要修改复制任务的配置，请先暂停任务后，单击操作栏的**编辑**，或者进入任务详情页，单击“基本信息”右上角的**编辑**，修改任务的信息。
您可以单击任务名称，进入任务详情页查看任务的详细配置，例如过滤规则和起始时间等等。 在监控部分，您可以查看当前消息复制任务的实时监控，例如复制成功的速率（XX条消息/秒），复制失败的速率，消息复制延时等。



异常处理

正常情况下，状态栏会展示 “运行中” 或者 “已暂停” 的状态；如果状态为 “启动失败”，您需要检查任务运行状态和任务详细配置是否正确，例如 SQL 表达式是否正确等；鼠标悬浮在失败状态上会有具体的失败原因。



如果任务状态失败，您可以单击操作栏的**编辑**，或者进入任务详情页，单击“基本信息”右上角的**编辑**，重新更正任务的信息。

开发指南
消息类型
普通消息

最近更新时间：2024-10-14 11:34:25

普通消息是一种基础的消息类型，由生产投递到指定 Topic 后，被订阅了该 Topic 的消费者所消费。普通消息的 Topic 中无顺序的概念，可以使用多个分区数来提升消息的生产和消费效率，在吞吐量巨大时其性能最好。
普通消息区别于有特性的定时与延迟消息、顺序消息和事务消息。这四种类型的消息对应的 Topic 不能混用，只能用于收发相同类型的消息，例如普通消息的 Topic 只能用于收发普通消息，不能用于收发延迟消息、顺序消息和事务消息。

定时与延迟消息

最近更新时间：2024-10-14 12:43:51

本文主要介绍消息队列 TDMQ RocketMQ 版中定时与延迟消息的概念和使用方式。

相关概念

- **定时消息**：消息在发送至服务端后，实际业务并不希望消费端马上收到这条消息，而是推迟到某个时间点被消费，这类消息统称为定时消息。
- **延时消息**：消息在发送至服务端后，实际业务并不希望消费端马上收到这条消息，而是推迟一段时间后再被消费，这类消息统称为延时消息。

实际上，延时消息可以看成是定时消息的一种特殊用法，其实现的最终效果和定时消息是一致的。

使用方式

开源 Apache RocketMQ 没有提供让用户自由设定延时时间的 API 的，TDMQ RocketMQ 版为了保证向开源 RocketMQ Client 的兼容，设计了一种通过添加 message 的 property 键值对来指定消息发送时间的方法。该方法只要在需要定时发送消息的 `property` 属性中增加 `__STARTDELIVERTIME` 属性值，就能在一定范围内（40天）实现该消息在任意时间的定时发送。延时消息则可以先通过计算得到定时发送的时间点，再以定时消息的形式发送。
下面给出一段具体代码示例来展示如何使用 TDMQ RocketMQ 版的定时和延时消息，查看完整示例 >>
延时消息先通过 `System.currentTimeMillis() + delayTime` 计算得到定时发送的时间点，再以定时消息的方式发送。

4.x 客户端

```
Message msg = new Message("test-topic", ("message
content").getBytes(StandardCharsets.UTF_8));

// 设定消息在10秒之后被发送
long delayTime = System.currentTimeMillis() + 10000;
// 将 __STARTDELIVERTIME 设定到 msg 的属性中
msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(delayTime));

SendResult result = producer.send(msg);
System.out.println("Send delay message: " + result);
```

5.x 客户端

```
Duration messageDelayTime = Duration.ofSeconds(10);
final Message message = provider.newMessageBuilder()
// Set topic for the current message.
.setTopic(topic)
```

```
 // Message secondary classifier of message besides topic.
.setTag(tag)
// Key(s) of the message, another way to mark message besides message id.
.setKeys("yourMessageKey-3ee439f945d7")
// Set expected delivery timestamp of message. 设置延迟消息的时间
.setDeliveryTimestamp(System.currentTimeMillis() + messageDelayTime.toMillis())
.setBody(body)
.build();
```

使用限制

- 使用延时消息时，请确保客户端的机器时钟和服务端的机器时钟（所有地域均为 UTC+8 北京时间）保持一致，否则会有时差。
- 定时和延时消息在精度上会有1秒左右的偏差。
- 关于定时和延时消息的时间范围，根据不同的集群规格，有不同的限制，可以参考产品系列。
- 使用定时消息时，设置的时刻在当前时刻以后才会有定时效果，否则消息将被立即发送给消费者。

顺序消息

最近更新时间：2025-02-28 11:35:22

顺序消息是消息队列 RocketMQ 提供的一种高级消息类型，对于一个指定的Topic，消息严格按照先进先出（FIFO）的原则进行消息发布和消费，即先发送的消息先消费，后发送的消息后消费。
顺序消息适用于对消息发送和消费顺序有严格要求的情况。

使用场景

顺序消息和普通消息的对比如下：

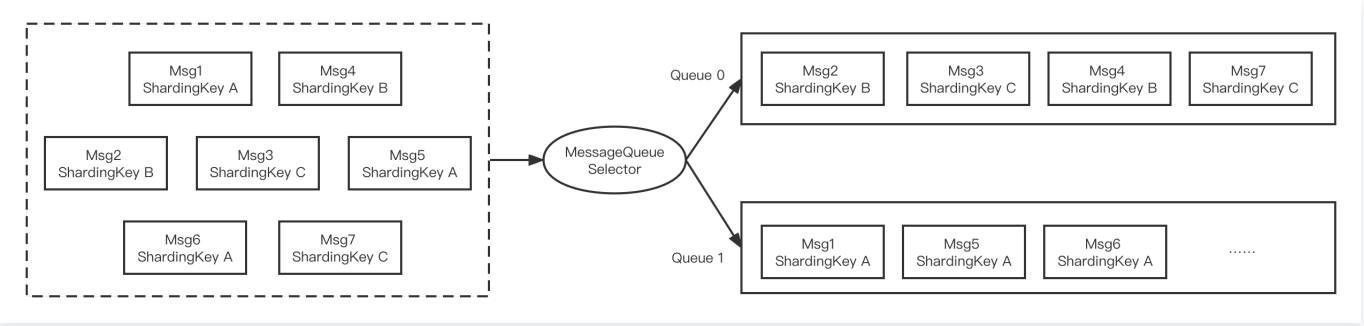
消息类型	消费顺序	性能	适用场景
普通消息	无顺序	高	适用于对吞吐量要求高，且对生产和消费顺序无要求
顺序消息	指定的 Topic 内的消息遵循先入先出（FIFO）规则	一般	吞吐量要求一般，但是要求特定的 Topic 严格地按照 FIFO 原则进行消息发布和消费的场景

对应到具体的业务场景，顺序消息可以被用在以下场景中：

- **订单创建场景：**在一些电商系统中，同一个订单相关的创建订单消息、订单支付消息、订单退款消息、订单物流消息必须严格按照先后顺序来进行生产或者消费，否则消费中传递订单状态会发生紊乱，影响业务的正常进行。因此，该订单的消息必须按照一定的顺序在客户端和消息队列中进行生产和消费，同时消息之间有先后的依赖关系，后一条消息需要依赖于前一条消息的处理结果。
- **日志同步场景：**在有序事件处理或者数据实时增量同步的场景中，顺序消息也能发挥较大的作用，如同步 mysql 的 binlog 日志时，需要保证数据库的操作是有顺序的。
- **金融场景：**在一些撮合交易的场景下，例如某些证券交易，在价格相同的情况下，先出价者优先处理，则需要按照FIFO的方式生产和消费顺序消息。

实现原理

在 RocketMQ 中支持顺序消息的原理如下图所示。我们可以按照某一个标准对消息进行分区（例如图中的ShardingKey），同一个ShardingKey 的消息会被分配到同一个队列中，并按照顺序被消费。



生产顺序消息

顺序消息的代码如下所示：

```
public class Producer {
    public static void main(String[] args) throws UnsupportedEncodingException {
        try {
            // 实例化消息生产者Producer
```

```
            DefaultMQProducer producer = new DefaultMQProducer(groupName,
              // ACL权限
            new AclClientRPCHook(new SessionCredentials(ACCESS_KEY, SECRET_KEY)),
true, null);
            // 设置NameServer的地址
            producer.setNamesrvAddr("rmq-xxx.rocketmq.xxxtencenttdmq.com:8080");
            // 启动Producer实例
            producer.start();

            String[] tags = new String[] {"TagA", "TagB", "TagC", "TagD", "TagE"};
            for (int i = 0; i < 100; i++) {
                int orderId = i % 10;
                Message msg =
                    new Message("TopicTest", tags[i % tags.length], "KEY" + i,
                        ("Hello RocketMQ " +
i).getBytes(RemotingHelper.DEFAULT_CHARSET));
                SendResult sendResult = producer.send(msg, new MessageQueueSelector()
{
                    @Override
                    public MessageQueue select(List<MessageQueue> mqs, Message msg,
Object arg) {
                        Integer id = (Integer) arg;
                        int index = id % mqs.size();
                        return mqs.get(index);
                    }
                }, orderId);

                System.out.printf("%s%n", sendResult);
            }

            producer.shutdown();
        } catch (MQClientException | RemotingException | MQBrokerException |
InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

这里的区别主要是调用了 `SendResult send(Message msg, MessageQueueSelector selector, Object arg)` 方法，`MessageQueueSelector` 是队列选择器，arg 是一个 Java Object 对象，可以传入作为消息发送分区的分类标准。`MessageQueueSelector` 的接口如下：

```
public interface MessageQueueSelector {
    MessageQueue select(final List<MessageQueue> mqs, final Message msg, final Object
arg);
}
```

其中 mqs 是可以发送的队列，msg 是消息，arg 是上述 send 接口中传入的 Object 对象，返回的是该消息需要发送到的队列。上述例子里，是以 orderId 作为分区分类标准，对所有队列个数取余，来对将相同 orderId 的消息发送到同一个队列中。

生产环境中建议选择最细粒度的分区键进行拆分，例如，将订单ID、用户ID作为分区键关键字，可实现同一终端用户的消息按照顺序处理，不同用户的消息无需保证顺序。

> **注意：**
> 为了保证消息的高可用，目前TDMQ RocketMQ版不支持单队列的 “全局顺序消息”（已经创建了全局顺序消息的用户可以正常使用）；如果您想保证全局的顺序性，您可以通过使用一致的 ShardingKey 来实现。

消费顺序消息

顺序消费代码如下：

```
/**
 * Description: 顺序消费者
 */
public class OrderConsumer {
    /**
     * topic名称
     */
    private static final String TOPIC_NAME = "order_topic";

    /**
     * 消费者组名称
     */
    private static final String GROUP_NAME = "group2";
    public static void main(String[] args) throws Exception {
        // 创建消息消费者
        // 实例化消费者
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer(GROUP_NAME,
                new AclClientRPCHook(new SessionCredentials("accessKey",
"secretKey")),
                new AllocateMessageQueueAveragely(), true, null);
        // 设置NameServer的地址
        consumer.setNamesrvAddr("rmq-xxx.rocketmq.xxxtencenttdmq.com:8080");
        consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
        /*
         * 设置Consumer第一次启动是从队列头部开始消费还是队列尾部开始消费<br>
         * 如果非第一次启动，那么按照上次消费的位置继续消费
         */
        consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
        // 订阅topic中的所有的信息
        consumer.subscribe(TOPIC_NAME, "*");
        consumer.registerMessageListener(new MessageListenerOrderly() {
            @Override
            public ConsumeOrderlyStatus consumeMessage(List<MessageExt> msgs,
ConsumeOrderlyContext context) {
                context.setAutoCommit(true);
                for (MessageExt msg : msgs) {
                    // 可以看到每个queue有唯一的consume线程来消费, 订单对每个queue(分区)有序
                    System.out.println("consumeThread=" +
Thread.currentThread().getName() + ", queueId=" + msg.getQueueId() + ", msgId=" +
```

```
msg.getMsgId() + ", content:" + new String(msg.getBody()));
                }
                try {
                    // 模拟业务逻辑处理中...
                    TimeUnit.SECONDS.sleep(1);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return ConsumeOrderlyStatus.SUCCESS;
            }
        });
        consumer.start();
        System.out.println("Consumer Started.");
    }
}
```

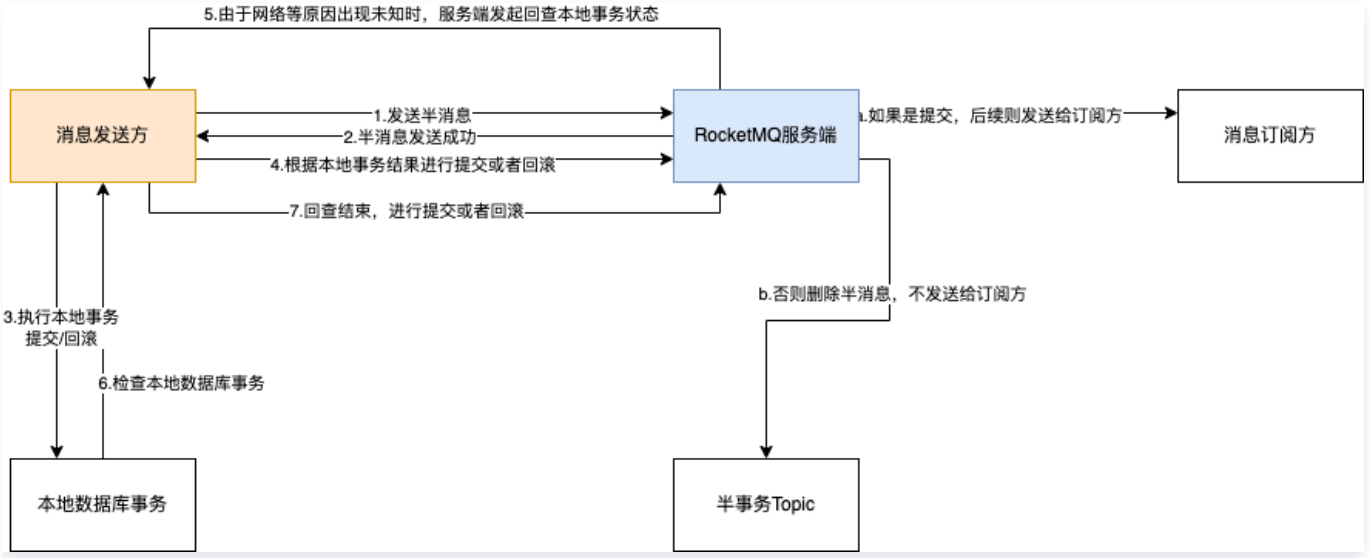
事务消息

最近更新时间：2024-10-10 11:34:25

本文主要介绍消息队列 TDMQ RocketMQ 版中事务消息的概念、技术原理、应用场景和使用方式。

功能介绍

事务消息实现了消息生产者本地事务与消息发送的原子性，保证了消息生产者本地事务处理成功与消息发送成功的最终一致。用户实现类似 X/Open XA 的分布事务功能，通过消息队列 TDMQ RocketMQ 版能达到分布式事务的最终一致。



1. 生产者发送消息到 RocketMQ 中（1）。
2. 服务端收到消息后将消息存储到半消息Topic 中（2）。
3. 当本地事务执行完成（3）。
4. 生产者主动将事务执行结果发送到 RocketMQ 中（4）。
5. 若本地事务执行结果超过一定期限还没反馈，RocketMQ 将执行回查逻辑（5）
6. 生产者收到消息回查后，需要检查对应消息的本地事务执行的最终结果，并反馈（6、7）。事务执行状态有以下三种情况：
   - TransactionStatus.COMMIT 提交事务，消费者可以消费到该消息。
   - TransactionStatus.ROLLBACK 回滚事务，消息被丢弃，消费者不会消费到该消息。
   - TransactionStatus.UN_KNOW 无法判断状态，等待再次发送回查。
7. 当事务执行成功，RocketMQ 将事务消息提交到 real topic，待消费者消费。

应用场景

使用 TDMQ RocketMQ 版的事务消息来处理交易事务，可以大大提升处理效率和性能。计费的交易链路通常比较长，出错或者超时的概率比较高，借助 TDMQ 的自动重推和海量堆积能力来实现事物补偿，支付 Tips 通知和交易流水推送可以通过 TDMQ 来实现最终一致性。

消息过滤

最近更新时间：2024-10-10 14:36:42

本文主要介绍 TDMQ RocketMQ 版中消息过滤的功能、应用场景和使用方式。

功能介绍

消息过滤功能指消息生产者向 Topic 中发送消息时，设置消息属性对消息进行分类，消费者订阅 Topic 时，根据消息属性设置过滤条件对消息进行过滤，只有符合过滤条件的消息才会被投递到消费端进行消费。
消费者订阅 Topic 时若未设置过滤条件，无论消息发送时是否有设置过滤属性，Topic 中的所有消息都将被投递到消费端进行消费。

应用场景

通常，一个 Topic 中存放的是相同业务属性的消息，例如交易流水 Topic 包含了下单流水、支付流水、发货流水等，业务若只想消费者其中一种类别的流水，可在客户端进行过滤，但这种过滤方式会带来带宽的资源浪费。
针对上述场景，TDMQ 提供 Broker 端过滤的方式，用户可在生产消息时设置一个或者多个 Tag 标签，消费时指定 Tag 订阅。

使用方式

TAG 过滤

发送消息

> **说明：**
> 发送消息时，每条消息必须指明 Tag。

```
String tag = "yourMessageTagA";
        final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // Message secondary classifier of message besides topic.
            .setTag(tag)
```

```
            // Key(s) of the message, another way to mark message besides message id.
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            .build();
```

订阅消息

- 订阅所有 Tag：消费者如需订阅某 Topic 下所有类型的消息，Tag 用星号（*）表示。

```
  String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        // In most case, you don't need to create too many consumers, singleton
pattern is recommended.
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
```

- 订阅单个 Tag：消费者如需订阅某 Topic 下某一种类型的消息，请明确标明 Tag。

```
  String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        String tag = "TAGA";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        // In most case, you don't need to create too many consumers, singleton
pattern is recommended.
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
```

```
            })
            .build();
```

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

```
 String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        String tag = "TAGA || TAGB";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        // In most case, you don't need to create too many consumers, singleton
pattern is recommended.
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
```

SQL 过滤

发送消息

发送代码和简单的消息没有区别。主要是在构造消息体的时候，带上自定义属性，允许多个。

```
final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // Message secondary classifier of message besides topic.
            // Key(s) of the message, another way to mark message besides message id.
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            //一些用于sql过滤的信息
            .addProperty("key1", "value1")
            .build();
```

订阅消息

对于消费消息，订阅时需带上相应的 SQL 表达式，其余与普通的消费消息流程无区别。

```
  String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
```

```
        String sql = "key1 IS NOT NULL AND key1='value1'";
        //sql表达式
        FilterExpression filterExpression = new FilterExpression(sql,
FilterExpressionType.SQL92);
        //如果是订阅所有
        //FilterExpression filterExpression = FilterExpression.SUB_ALL;
        // In most case, you don't need to create too many consumers, singleton
pattern is recommended.
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
```

说明

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档 。

消息重试

最近更新时间：2024-11-05 15:01:11

本文主要介绍消息队列 TDMQ RocketMQ 版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，或者用户主动要求服务端重投，TDMQ RocketMQ 版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 TDMQ RocketMQ 版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。您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。

> **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：

- 消费者返回 ConsumeResult.FAILURE。
- 消费者返回 null。
- 消费者主动/被动抛出异常。

重试次数

当消息需要重试时，TDMQ RocketMQ 中配置了如下的 messageDelayLevel 参数来设置重试次数与时间间隔。

```
messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h
```

由于 SDK 中第一次 delayLevel 为3，所以重试次数与重试时间间隔关系如下：

第几次重试	距离上一次重试的时间间隔	第几次重试	距离上一次重试的时间间隔
1	10秒	9	7分钟
2	30秒	10	8分钟
3	1分钟	11	9分钟
4	2分钟	12	10分钟
5	3分钟	13	20分钟
6	4分钟	14	30分钟
7	5分钟	15	1小时
8	6分钟	16	2小时

使用方式

无需特别处理，5.0的 SDK 遵循上述规则。

死信消息

最近更新时间：2025-02-28 11:35:22

本文主要介绍消息队列 TDMQ RocketMQ 版中消息死信队列与使用方法。

功能介绍

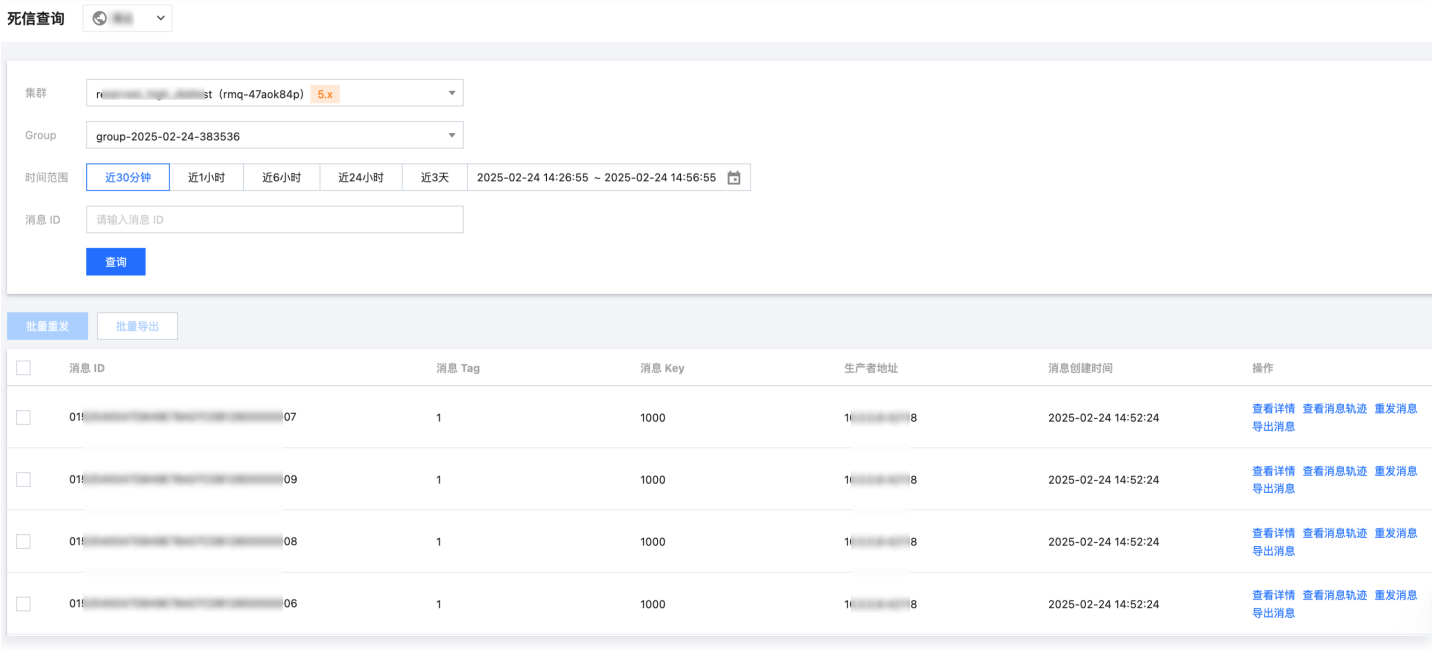
- 当消息第一次被消费者消费后，没有得到正常的回应，或者用户主动要求服务端重投，TDMQ RocketMQ 版会通过消费重试机制自动重新投递该消息，直到该消息被成功消费，当重试达到一定次数后，消息仍未被成功消费，则会停止重试，将消息投递到死信队列中。
- 当消息进入到死信队列中，表示 TDMQ RocketMQ 版已经无法自动处理这批消息，一般这时就需要人为介入来处理这批消息。您可以通过导出消息的方式进行确认，或者在管控台进行指定消息的重发。

特性说明

- 不同于重试队列，会自动消费，死信队列的消息目前需要人工介入。
- 消息有效期依然遵守默认三天删除的规则。
- 死信队列开头 `%DLQ%` ，和消费组一一对应，因此一个死信队列包含了对应 Group ID 的所有死信消息，不论消息属于哪个 Topic。

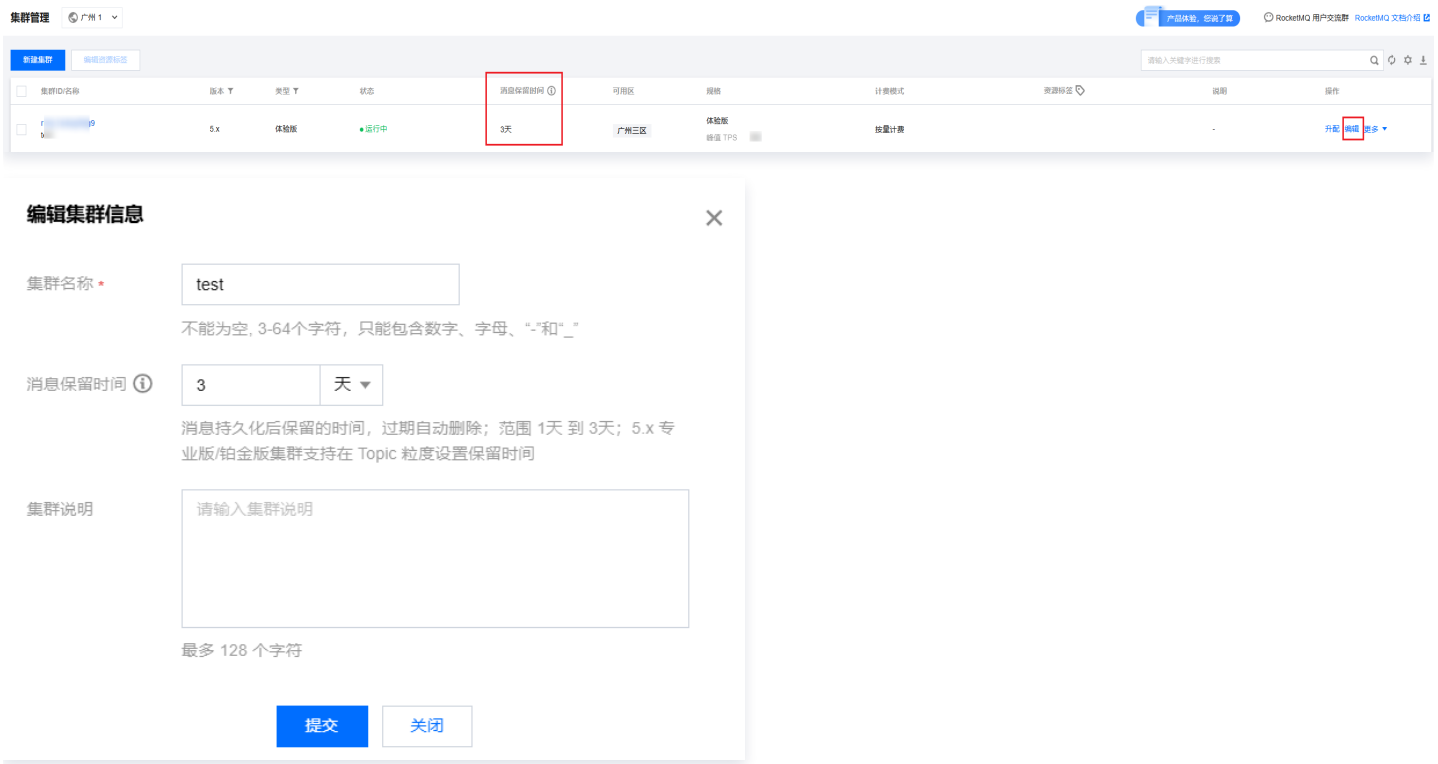
查询死信消息

死信消息，由于不能直接被业务代码消费，所以允许用户进行查询。您可以进入 RocketMQ 控制台，在左侧导航栏选择**死信查询**，并根据不同的查询条件来查询死信消息。



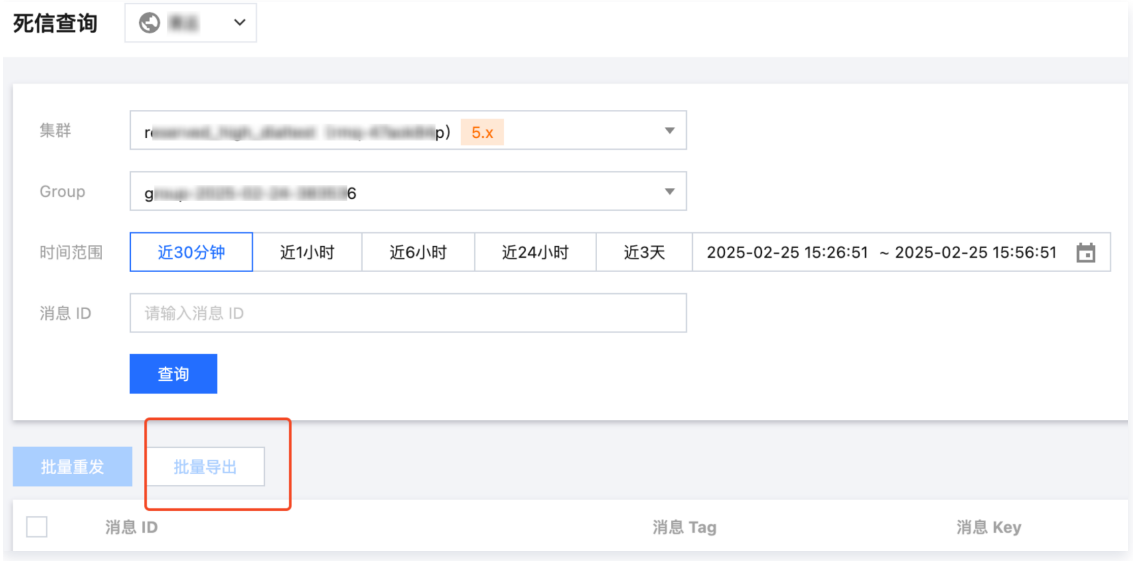
> **说明：**
> 查询时间和普通消息相同，不能超过集群的消息保留时间，默认3天。

您可以进入 RocketMQ 控制台，在左侧导航栏选择**集群管理**，查看集群消息保留时间。您也可以单击操作列**编辑**，进入**编辑集群信息**页来调整**消息保留时间**。



导出死信消息

如果您想要导出所有的死信消息用来单独处理，您可以进入 RocketMQ 控制台，在左侧导航栏选择死信查询，输入条件后单击查询，选择需要导出的信息后单击批量导出。若您想单条导出，您也可以单击操作列的导出消息。



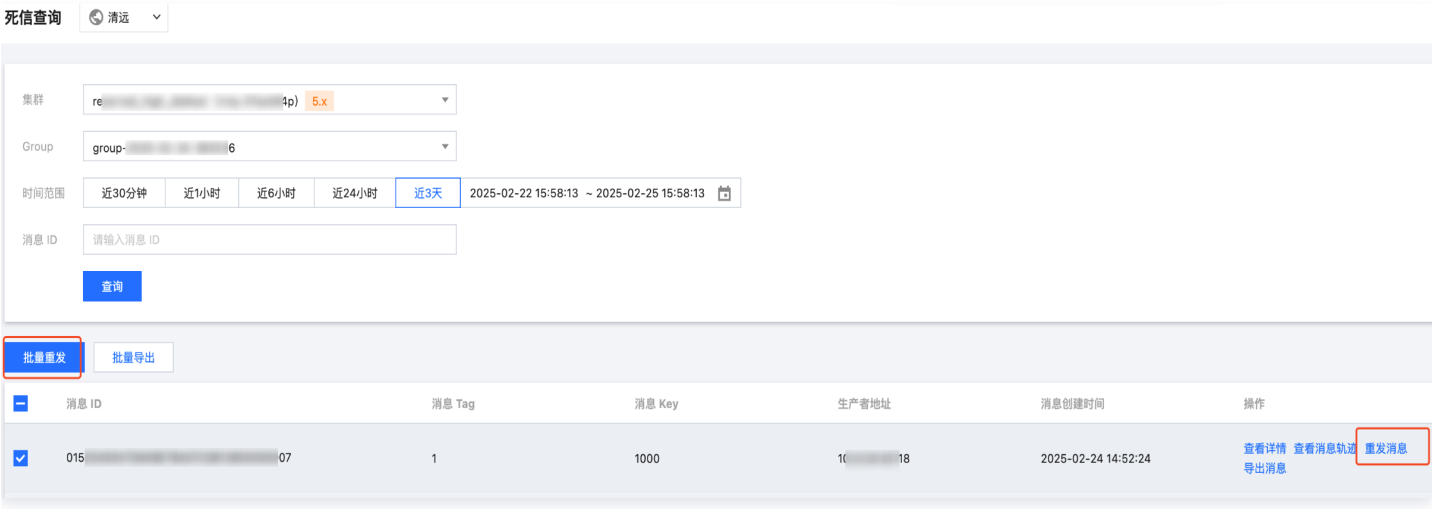
导出的文件包含以下字段：

字段	说明
intanceId	集群 ID
group	消息所属 Group
tags	消息 Tag
keys	消息 Key

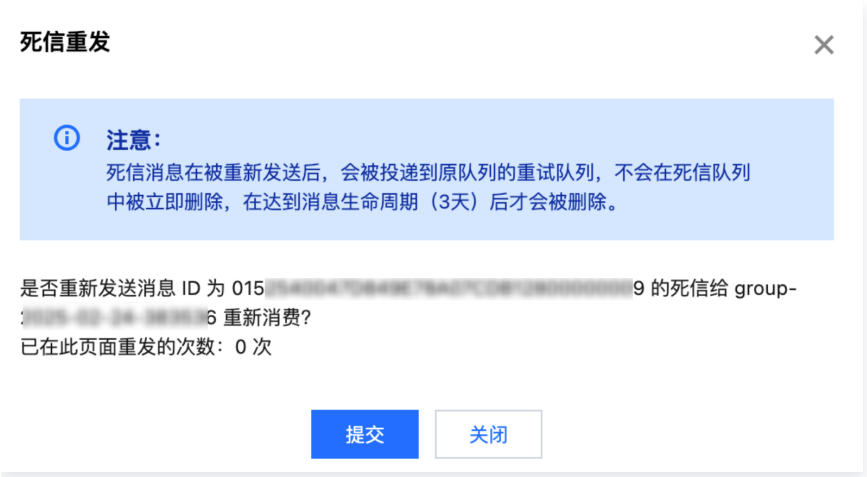
messageId	消息 ID
bornHost	消息生产地址
bornTimestamp	消息生产时间
storeTimestamp	消息存储到服务端的时间
reconsumeTimes	消费失败次数
properties	消费属性
body	消息体
crc	消息体 CRC

死信消息重发

如果您确认代码已经完成修改，您可以处理那些因为之前多次消费失败而最终进入死信队列的消息。具体操作如下：
您可以进入 RocketMQ 控制台，在左侧导航栏选择**死信查询**，输入条件后单击**查询**，选择需要重发的信息后单击**批量重发**。若您想单条重发，您也可以单击**操作**列的**重发消息**。



单击**批量重发**或**重发消息**后，服务端将会把这些消息，写入到原始的 topic 里面，走正常的消费流程进行消费。



消费模式
集群消费模式

最近更新时间：2024-10-11 14:36:42

集群消费模式介绍

当使用集群消费模式时，任意一条消息只需要被同一个订阅组内的任意一个消费者处理即可。

应用场景

适用于每条消息只需要被处理一次的场景。

如何使用

5.0 SDK 默认为集群消费模式，无需特殊设置。

> **说明：**
> 请确保同一个 Group ID 下所有 Consumer 实例的订阅关系保持一致。

广播消费模式

最近更新时间：2024-10-14 14:36:42

广播消费模式介绍

当使用广播消费模式时，每条消息会被推送给集群内所有注册过的消费者，保证消息至少被每个消费者消费一次。

应用场景

适用于每条消息需要被集群下每一个消费者处理的场景。

如何使用

5.0 SDK 已经不支持广播消费，如果需要使用，可以给每个消费者创建一个单独的订阅组实现类似的功能。

> **说明：**
> 请确保同一个 Group ID 下所有 Consumer 实例的订阅关系保持一致。

SDK 文档
兼容性说明

最近更新时间：2025-01-03 14:38:02

RocketMQ 5.0 提供了全新的基于 gRPC 协议的 5.x SDK，新版本 SDK 更加轻量化，多语言支持更好，建议优先使用。同时，消息队列 RocketMQ 版 5.x 系列也支持存量业务继续使用 4.x SDK 访问，兼容性说明如下：

服务端版本	客户端版本		兼容性
5.x	5.x SDK		完全兼容
	4.x SDK	版本 >= 4.9.5	• PushConsumer CONSUME_FROM_TIMESTAMP 暂不生效（控制台可以重置位点）。 • 消费者 setPullBatchSize 最大值为 32。 • 事务消息存在兼容性问题，事务提交或回查可能会失败，暂不建议使用。
		版本 < 4.9.5	• PushConsumer CONSUME_FROM_TIMESTAMP 暂不生效（控制台可以重置位点）。 • 消费者 setPullBatchSize 最大值为 32。 • PullConsumer 消费暂不支持。 • 事务消息存在兼容性问题，事务提交或回查可能会失败，暂不建议使用。

5.x SDK
Java SDK 使用

最近更新时间：2025-01-03 14:38:02

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **说明：**
> 以 Java 客户端为例说明，其他语言客户端请参见 SDK 文档。

前提条件

- 完成资源创建与准备
- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 Maven 工程为例，在 pom.xml 添加以下依赖：

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.6</version>
        </dependency>
</dependencies>
```

> **注意：**
> 如果您是在 Spring Boot 环境下使用，可能会遇到 annotations-api 依赖冲突，这时候，可以增加排除依赖即可。

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.6</version>
            <exclusions>
                <exclusion>
                    <groupId>org.apache.tomcat</groupId>
                    <artifactId>annotations-api</artifactId>
                </exclusion>
```

```
            </exclusions>
        </dependency>
</dependencies>
```

步骤2. 生产消息

```
public class NormalMessageSyncProducer {
    private static final Logger log =
LoggerFactory.getLogger(NormalMessageSyncProducer.class);

    private NormalMessageSyncProducer() {
    }

    public static void main(String[] args) throws ClientException, IOException {
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        // 添加配置的ak和sk
        String accessKey = "yourAccessKey"; //ak，可以在控制台“权限管理”页面获取
        String secretKey = "yourSecretKey"; //sk，可以在控制台“权限管理”页面获取
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // 填写腾讯云提供的接入地址
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8080";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String topic = "yourNormalTopic";
        // In most case, you don't need to create too many producers, singleton
pattern is recommended.
        final Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the topic name(s), which is optional but recommended. It makes
producer could prefetch the topic
            // route before message publishing.
            .setTopics(topic)
            // May throw {@link ClientException} if the producer is not initialized.
            .build();
        // Define your message body.
        byte[] body = "This is a normal message for Apache
RocketMQ".getBytes(StandardCharsets.UTF_8);
        String tag = "yourMessageTagA";
        final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // Message secondary classifier of message besides topic.
            .setTag(tag)
            // Key(s) of the message, another way to mark message besides message id.
```

```
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            .build();
        try {
            final SendReceipt sendReceipt = producer.send(message);
            log.info("Send message successfully, messageId={}",
sendReceipt.getMessageId());
        } catch (Throwable t) {
            log.error("Failed to send message", t);
        }
        // Close the producer when you don't need it anymore.
        producer.close();
    }
}
```

步骤3. 消费消息

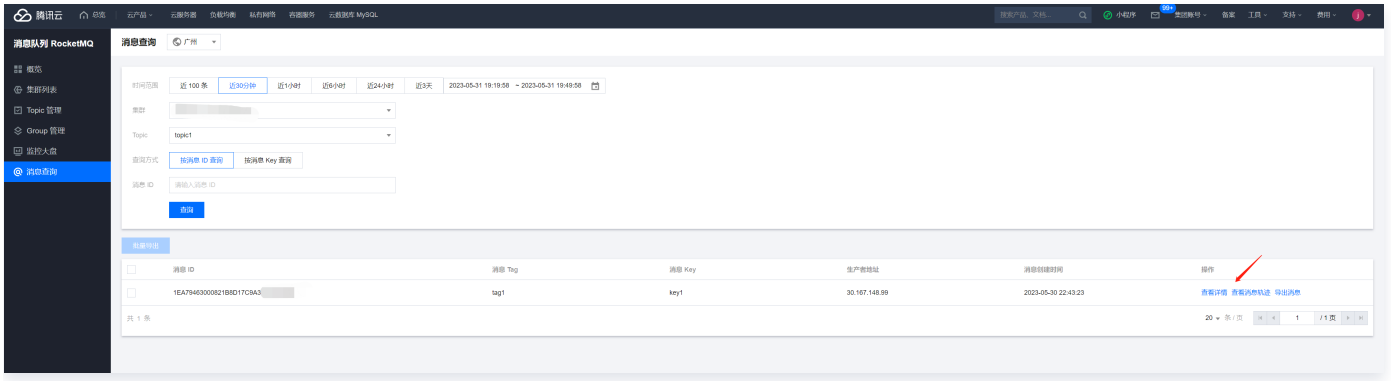
腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种类型的消费者客户端，分别为 Push Consumer 和 Simple Consumer。
以下代码示例以 Push Consumer 为例：

```
public class NormalPushConsumer {
    private static final Logger log =
LoggerFactory.getLogger(NormalPushConsumer.class);

    private NormalPushConsumer() {
    }

    public static void main(String[] args) throws ClientException, IOException,
InterruptedException {
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        // 添加配置的ak和sk
        String accessKey = "yourAccessKey"; //ak，可以在控制台“权限管理”页面获取
        String secretKey = "yourSecretKey"; //sk，可以在控制台“权限管理”页面获取
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // 填写腾讯云提供的接入地址
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8080";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
```

```
        // In most case, you don't need to create too many consumers, singleton
pattern is recommended.
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
        // Block the main thread, no need for production environment.
        Thread.sleep(Long.MAX_VALUE);
        // Close the push consumer when you don't need it anymore.
        pushConsumer.close();
    }
}
```

步骤4. 查看消息详情

发送完成消息后会得到一个消息ID（messageID），开发者可以在**消息查询**页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 。



Go SDK 使用

最近更新时间：2024-07-15 17:55:31

操作场景

本文以调用 Go SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **说明：**
> 以 Go 客户端为例说明，其他语言客户端请参见 SDK 文档。

前提条件

- 完成资源创建与准备
- 安装 1.13 版本以上的 golang
- 下载 Demo

操作步骤

步骤1：安装 Go 依赖库

在 Golang 项目中引入相关依赖，以 `go get` 为例，使用如下命令：

```
go get github.com/apache/rocketmq-clients/golang/v5
```

步骤2：生产消息

```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "strconv"
    "time"

    rmq_client "github.com/apache/rocketmq-clients/golang/v5"
    "github.com/apache/rocketmq-clients/golang/v5/credentials"
)

const (
    Topic = "xxxxxx"
    // Endpoint 填写腾讯云提供的接入地址
    Endpoint = "xxxxxx"
    // AccessKey 添加配置的ak
    AccessKey = "xxxxxx"
    // SecretKey 添加配置的sk
    SecretKey = "xxxxxx"
```

```
)

func main() {
    os.Setenv("mq.consoleAppender.enabled", "true")
    rmq_client.ResetLogger()
    // In most case, you don't need to create many producers, singleton pattern is
more recommended.
    producer, err := rmq_client.NewProducer(&rmq_client.Config{
        Endpoint: Endpoint,
        Credentials: &credentials.SessionCredentials{
            AccessKey:    AccessKey,
            AccessSecret: SecretKey,
        },
    },
        rmq_client.WithTopics(Topic),
    )
    if err != nil {
        log.Fatal(err)
    }
    // start producer
    err = producer.Start()
    if err != nil {
        log.Fatal(err)
    }
    // graceful stop producer
    defer producer.GracefulStop()

    for i := 0; i < 10; i++ {
        // new a message
        msg := &rmq_client.Message{
            Topic: Topic,
            Body:  []byte("this is a message : " + strconv.Itoa(i)),
        }
        // set keys and tag
        msg.SetKeys("a", "b")
        msg.SetTag("ab")
        // send message in sync
        resp, err := producer.Send(context.TODO(), msg)
        if err != nil {
            log.Fatal(err)
        }
        for i := 0; i < len(resp); i++ {
            fmt.Printf("%#v\n", resp[i])
        }
        // wait a moment
        time.Sleep(time.Second * 1)
    }
}
```

步骤3：消费消息

腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种消费模式，分别为 Push Consumer 和 Simple Consumer。

> **说明：**
> 社区版 Golang SDK 目前仅支持 Simple Consumer。

以下代码示例以 Simple Consumer 为例：

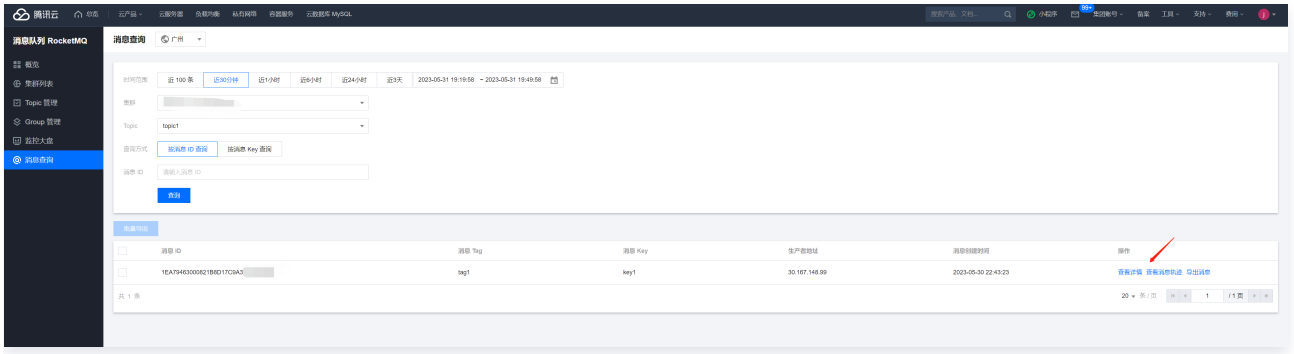
```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "os/signal"
    "sync"
    "syscall"
    "time"

    rmq_client "github.com/apache/rocketmq-clients/golang/v5"
    "github.com/apache/rocketmq-clients/golang/v5/credentials"
)

const (
    Topic         = "xxxxxx"
    ConsumerGroup = "xxxxxx"
    // Endpoint 填写腾讯云提供的接入地址
    Endpoint = "xxxxxx"
    // AccessKey 添加配置的ak
    AccessKey = "xxxxxx"
    // SecretKey 添加配置的sk
    SecretKey = "xxxxxx"
)

var (
    // maximum waiting time for receive func
    awaitDuration = time.Second * 5
    // maximum number of messages received at one time
    maxMessageNum int32 = 16
    // invisibleDuration should > 20s
    invisibleDuration = time.Second * 20
    // receive concurrency
    receiveConcurrency = 6
)

func main() {
    // log to console
    os.Setenv("mq.consoleAppender.enabled", "true")
    rmq_client.ResetLogger()
```

```
    // In most case, you don't need to create many consumers, singleton pattern is
more recommended.
    simpleConsumer, err := rmq_client.NewSimpleConsumer(&rmq_client.Config{
        Endpoint:      Endpoint,
        ConsumerGroup: ConsumerGroup,
        Credentials: &credentials.SessionCredentials{
            AccessKey:    AccessKey,
            AccessSecret: SecretKey,
        },
    },
        rmq_client.WithAwaitDuration(awaitDuration),

rmq_client.WithSubscriptionExpressions(map[string]*rmq_client.FilterExpression{
            Topic: rmq_client.SUB_ALL,
        }),
    )
    if err != nil {
        log.Fatal(err)
    }
    // start simpleConsumer
    err = simpleConsumer.Start()
    if err != nil {
        log.Fatal(err)
    }
    // graceful stop simpleConsumer
    defer func() {
        if r := recover(); r != nil {
            fmt.Println(r)
        }
        _ = simpleConsumer.GracefulStop()
    }()

    fmt.Println("start receive message")

    // Each Receive call will only select one broker queue to pop messages.
    // Enable multiple consumption goroutines to reduce message end-to-end latency.
    ch := make(chan struct{})
    wg := &sync.WaitGroup{}
    for i := 0; i < receiveConcurrency; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            for {
                select {
                case <-ch:
                    return
                default:
                    mvs, err := simpleConsumer.Receive(context.TODO(), maxMessageNum,
invisibleDuration)
                    if err != nil {
                        fmt.Println("receive message error: " + err.Error())
```

```
                    }
                    // ack message
                    for _, mv := range mvs {
                        fmt.Println(mv)
                        if err := simpleConsumer.Ack(context.TODO(), mv); err != nil {
                            fmt.Println("ack message error: " + err.Error())
                        }
                    }
                }
            }
        }()
    }

    exit := make(chan os.Signal, 1)
    signal.Notify(exit, syscall.SIGINT, syscall.SIGTERM)

    // wait for exit
    <-exit
    close(ch)
    wg.Wait()
}
```

步骤4：查看消息详情

发送完成消息后会得到一个消息 ID（messageID），开发者可以在**消息查询**页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 。



C++ SDK 使用

最近更新时间：2024-10-14 14:33:21

操作场景

本文以调用 C++ SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **说明**
> 以 C++ 客户端为例说明，其他语言客户端请参见 SDK 文档。

前提条件

- 完成资源创建与准备
- 安装支持 C++11 的编译套件
- 安装 bazel（5.2.0）或 cmake（3.13 及以上）
- 如果使用 cmake 编译，需要提前安装 grpc，推荐安装 1.46.3 版本，较高版本与 SDK 存在不兼容。
- 下载 Demo

操作步骤

步骤1：安装 C++ SDK

- 安装 SDK。

> **注意：**
> 腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列暂不支持 TLS，需要拉取 补丁 。

步骤2. 生产消息

```
#include <algorithm>
#include <atomic>
#include <iostream>
#include <memory>
#include <random>
#include <string>
#include <system_error>

#include "rocketmq/CredentialsProvider.h"
#include "rocketmq/Logger.h"
#include "rocketmq/Message.h"
#include "rocketmq/Producer.h"

using namespace ROCKETMQ_NAMESPACE;

const std::string &alphaNumeric() {
    static std::string
alpha_numeric("0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ");
```

```
    return alpha_numeric;
}

std::string randomString(std::string::size_type len) {
    std::string result;
    result.reserve(len);
    std::random_device rd;
    std::mt19937 generator(rd());
    std::string source(alphaNumeric());
    std::string::size_type generated = 0;
    while (generated < len) {
        std::shuffle(source.begin(), source.end(), generator);
        std::string::size_type delta = std::min({len - generated, source.length()});
        result.append(source.substr(0, delta));
        generated += delta;
    }
    return result;
}

static const std::string topic = "xxx";
// 填写腾讯云提供的接入地址
static const std::string access_point = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
// 添加配置的ak和sk
static const std::string access_key = "xxx";
static const std::string access_secret = "xxx";
static const uint32_t total = 32;
static const int32_t message_body_size = 128;


int main(int argc, char *argv[]) {
    CredentialsProviderPtr credentials_provider;
    if (!access_key.empty() && !access_secret.empty()) {
        credentials_provider = std::make_shared<StaticCredentialsProvider>(access_key,
access_secret);
    }

    // In most case, you don't need to create too many producers, singletion pattern
is recommended.
    auto producer = Producer::newBuilder()
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)
                                       .withCredentialsProvider(credentials_provider)
                                       .withSsl(false)
                                       .build())
            .withTopics({topic})
            .build();

    std::atomic_bool stopped;
    std::atomic_long count(0);

    auto stats_lambda = [&] {
```

```
        while (!stopped.load(std::memory_order_relaxed)) {
            long cnt = count.load(std::memory_order_relaxed);
            while (count.compare_exchange_weak(cnt, 0)) {
                break;
            }
            std::this_thread::sleep_for(std::chrono::seconds(1));
            std::cout << "QPS: " << cnt << std::endl;
        }
    };

    std::thread stats_thread(stats_lambda);

    std::string body = randomString(message_body_size);

    try {
        for (std::size_t i = 0; i < total; ++i) {
            auto message = Message::newBuilder()
                    .withTopic(topic)
                    .withTag("TagA")
                    .withKeys({"Key-" + std::to_string(i)})
                    .withBody(body)
                    .build();
            std::error_code ec;
            SendReceipt send_receipt = producer.send(std::move(message), ec);
            if (ec) {
                std::cerr << "Failed to publish message to " << topic << ". Cause: "
<< ec.message() << std::endl;
            } else {
                std::cout << "Publish message to " << topic << " OK. Message-ID: " <<
send_receipt.message_id
                          << std::endl;
                count++;
            }
        }
    } catch (...) {
        std::cerr << "Ah...No!!!" << std::endl;
    }
    stopped.store(true, std::memory_order_relaxed);
    if (stats_thread.joinable()) {
        stats_thread.join();
    }

    return EXIT_SUCCESS;
}
```

步骤3. 消费消息

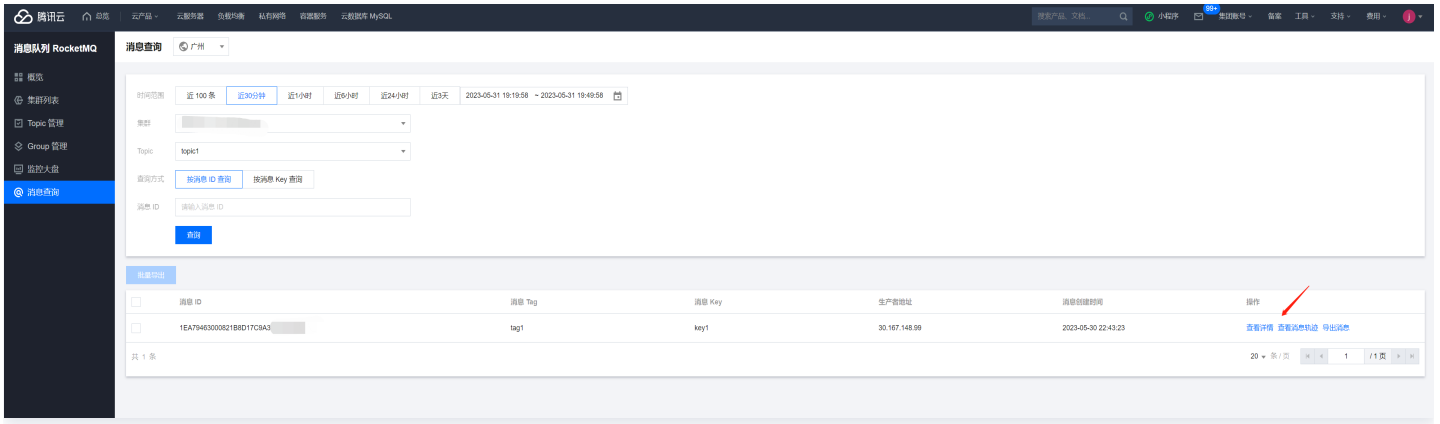
腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种类型的客户端，分别为 Push Consumer 和 Simple Consumer。以下代码示例以 Push Consumer 为例：

```
#include <chrono>
#include <iostream>
#include <mutex>
#include <thread>

#include "rocketmq/Logger.h"
#include "rocketmq/PushConsumer.h"

using namespace ROCKETMQ_NAMESPACE;

static const std::string topic = "xxx";
// 填写腾讯云提供的接入地址
static const std::string access_point = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
// 添加配置的ak和sk
static const std::string access_key = "xxx";
static const std::string access_secret = "xxx";
static const std::string group = "group-xxx";

int main(int argc, char *argv[]) {
    auto &logger = getLogger();
    logger.setConsoleLevel(Level::Info);
    logger.setLevel(Level::Info);
    logger.init();

    std::string tag = "*";

    auto listener = [](const Message &message) {
        std::cout << "Received a message[topic=" << message.topic() << ", MsgId=" <<
message.id() << "]" << std::endl;
        return ConsumeResult::SUCCESS;
    };

    CredentialsProviderPtr credentials_provider;
    if (!access_key.empty() && !access_secret.empty()) {
        credentials_provider = std::make_shared<StaticCredentialsProvider>(access_key,
access_secret);
    }

    // In most case, you don't need to create too many consumers, singletion pattern
is recommended.
    auto push_consumer = PushConsumer::newBuilder()
            .withGroup(group)
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)
                                       .withRequestTimeout(std::chrono::seconds(3))
                                       .withCredentialsProvider(credentials_provider)
                                       .withSsl(false)
                                       .build())
            .withConsumeThreads(4)
            .withListener(listener)
```

```
            .subscribe(topic, tag)
            .build();

    std::this_thread::sleep_for(std::chrono::minutes(30));

    return EXIT_SUCCESS;
}
```

步骤4. 查看消息详情

发送完成消息后会得到一个消息ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 。



C# SDK 使用

最近更新时间：2024-10-11 15:01:21

操作场景

本文以调用 C# SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **说明：**
> 以 C# 客户端为例说明，其他语言客户端请参见对应语言的 SDK 文档。

前提条件

- 完成资源创建与准备
- 安装 DotNet 环境
- 下载 Demo

操作步骤

步骤1：安装 RocketMQ5 sdk 依赖库

在 C# 项目中引入相关依赖，使用如下命令：

```
dotnet add package RocketMQ.Client --version 5.1.0
```

步骤2：生产消息

```
using System.Text;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Org.Apache.Rocketmq;
namespace examples
{
    internal static class ProducerNormalMessageDemo
    {
        static readonly ILoggerFactory factory = LoggerFactory.Create(builder =>
builder.AddConsole());
        static ILogger logger = factory.CreateLogger("Program_Producer");
        internal static async Task QuickStart()
        {
            // Enable the switch if you use .NET Core 3.1 and want to disable TLS/SSL.

AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2UnencryptedSupport",
true);
            const string accessKey = "yourAccessKey";
            const string secretKey = "yourSecretKey";


            // Credential provider is optional for client configuration.
```

```
            var credentialsProvider = new StaticSessionCredentialsProvider(accessKey,
secretKey);
            const string endpoints = "腾讯云官网接入地址:8080";
            var clientConfig = new ClientConfig.Builder()
                .SetEndpoints(endpoints)
                .SetCredentialsProvider(credentialsProvider)
                .Build();


            const string topic = "topicName";
            // In most case, you don't need to create too many producers, single
pattern is recommended.
            // Producer here will be closed automatically.
            var producer = await new Producer.Builder()
                // Set the topic name(s), which is optional but recommended.
                // It makes producer could prefetch the topic route before message
publishing.
                .SetTopics(topic)
                .SetClientConfig(clientConfig)
                .Build();


            // Define your message body.
            var bytes = Encoding.UTF8.GetBytes("foobar");
            const string tag = "yourMessageTagA";
            var message = new Message.Builder()
                .SetTopic(topic)
                .SetBody(bytes)
                .SetTag(tag)
                // You could set multiple keys for the single message actually.
                .SetKeys("yourMessageKey-7044358f98fc")
                .Build();


            var sendReceipt = await producer.Send(message);
            logger.LogInformation($"Send message successfully, messageId=
{sendReceipt.MessageId}");


            // Close the producer if you don't need it anymore.
            await producer.DisposeAsync();


        }
    }
}
```

步骤3：消费消息

腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种消费模式，分别为 Push Consumer 和 Simple Consumer。

> **说明：**
> 社区版 C# SDK 目前仅支持 Simple Consumer。

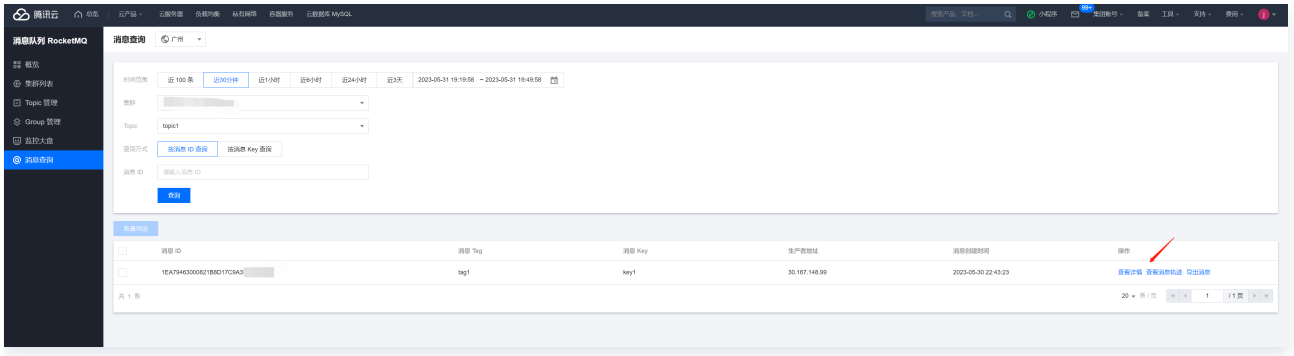
以下代码示例以 Simple Consumer 为例：

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Org.Apache.Rocketmq;


namespace examples
{
    internal static class SimpleConsumerExample
    {
        static readonly ILoggerFactory factory = LoggerFactory.Create(builder =>
builder.AddConsole());
        static ILogger logger = factory.CreateLogger("Program_Consumer");
        internal static async Task QuickStart()
        {
            // Enable the switch if you use .NET Core 3.1 and want to disable TLS/SSL.

AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2UnencryptedSupport",
true);
            const string accessKey = "yourAccessKey";
            const string secretKey = "yourSecretKey";


            // Credential provider is optional for client configuration.
            var credentialsProvider = new StaticSessionCredentialsProvider(accessKey,
secretKey);
            const string endpoints = "foobar.com:8080";
            var clientConfig = new ClientConfig.Builder()
                .SetEndpoints(endpoints)
                .SetCredentialsProvider(credentialsProvider)
                .Build();


            // Add your subscriptions.
            const string consumerGroup = "yourConsumerGroup";
            const string topic = "yourTopic";
            var subscription = new Dictionary<string, FilterExpression>
                { { topic, new FilterExpression("*") } };
            // In most case, you don't need to create too many consumers, single
pattern is recommended.
            var simpleConsumer = await new SimpleConsumer.Builder()
                .SetClientConfig(clientConfig)
                .SetConsumerGroup(consumerGroup)
```

```
                .SetAwaitDuration(TimeSpan.FromSeconds(15))
                .SetSubscriptionExpression(subscription)
                .Build();


            while (true)
            {
                var messageViews = await simpleConsumer.Receive(16,
TimeSpan.FromSeconds(15));
                foreach (var message in messageViews)
                {
                    logger.LogInformation(
                        $"Received a message, topic={message.Topic}, message-id=
{message.MessageId}, body-size={message.Body.Length}");
                    await simpleConsumer.Ack(message);
                    logger.LogInformation($"Message is acknowledged successfully,
message-id={message.MessageId}");
                    // await simpleConsumer.ChangeInvisibleDuration(message,
TimeSpan.FromSeconds(15));
                    // Logger.LogInformation($"Changing message invisible duration
successfully, message=id={message.MessageId}");
                }
            }
            // Close the simple consumer if you don't need it anymore.
            // await simpleConsumer.DisposeAsync();
        }
    }
}
```

步骤4：查看消息详情

发送完成消息后会得到一个消息 ID（messageID），开发者可以在**消息查询**页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 。



4.x SDK
Java SDK 使用

最近更新时间：2024-10-10 10:59:01

操作场景

本文以调用 4.0 Java SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **说明**
> 以 Java 客户端为例说明，其他语言客户端请参见 SDK 文档。

前提条件

- 完成资源创建与准备
- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 Maven 工程为例，在 pom.xml 添加以下依赖：

```
<!-- in your <dependencies> block -->
  <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>
```

步骤2. 生产消息

```
// 实例化消息生产者Producer
   DefaultMQProducer producer = new DefaultMQProducer(
       groupName,
       new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL权限，
ak sk可以在控制台“权限管理”页面获取
   );
   // 设置NameServer的地址，地址就是形如xxx.tencenttdmq.com:8080 这样的接入地址。
```

```
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();

    for (int i = 0; i < 10; i++) {
            // 创建消息实例，设置topic和消息内容.
            Message msg = new Message(topic_name, ("Hello RocketMQ " +
i).getBytes(RemotingHelper.DEFAULT_CHARSET));
            // 发送消息
            SendResult sendResult = producer.send(msg);
            System.out.printf("%s%n", sendResult);
    }
```

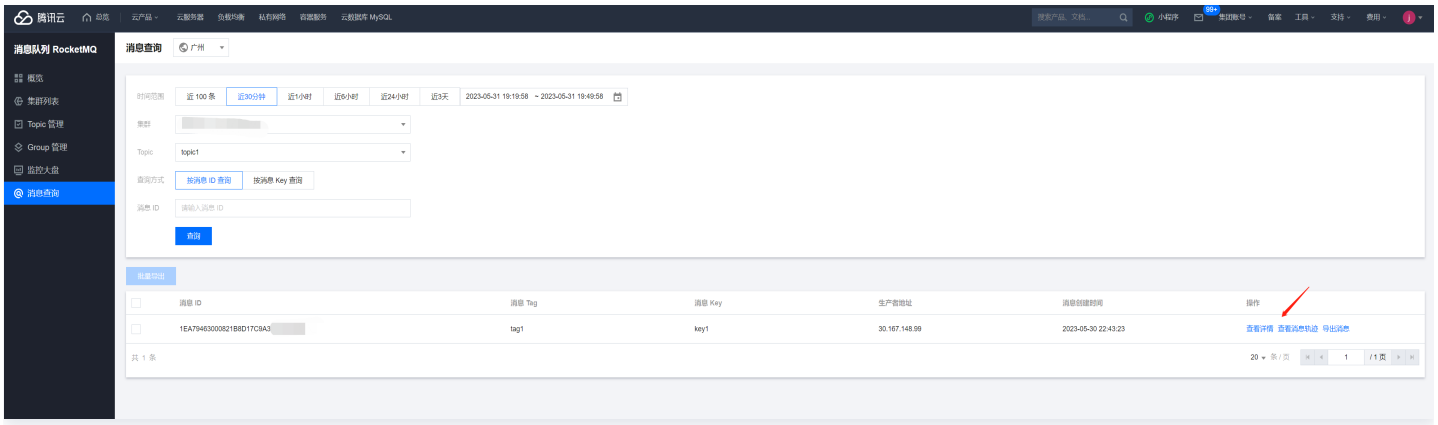
步骤3. 消费消息

以下代码示例以 Push Consumer 为例，其他的可以参考更详细 4.x 的使用文档。

```
// 实例化消费者
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //ACL
权限，ACL权限，ak sk可以在控制台“权限管理”页面获取
        // 设置NameServer的地址,地址就是形如xxx.tencenttdmq.com:8080 这样的接入地址。
pushConsumer.setNamesrvAddr(nameserver);
        // 订阅topic
pushConsumer.subscribe(topic_name, "*");
      // 注册回调实现类来处理从broker拉取回来的消息
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context) ->
{
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
      });
      // 启动消费者实例
pushConsumer.start();
```

步骤4. 查看消息详情

发送完成消息后会得到一个消息ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 章节。


消息队列 RocketMQ
概览
集群列表
Topic 管理
Group 管理
监控大盘
消息查询
消息查询
广州
近 100 条
近30分钟
近1小时
近6小时
近24小时
近3天
2023-05-31 19:19:58 ~ 2023-05-31 19:49:58
Topic
topic1
按消息 ID 查询
按消息 Key 查询
查询
消息 ID
消息 Tag
消息 Key
生产者地址
消息创建时间
操作
tag1
key1
30.167.148.99
2023-05-30 22:43:23
查看详情
查看消息轨迹
导出消息

Go SDK 使用

最近更新时间：2025-02-28 11:35:22

操作场景

本文以调用 Go SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

- 完成资源创建与准备
- 安装 Go
- 下载 Demo

操作步骤

1. 在客户端环境执行如下命令下载 RocketMQ 客户端相关的依赖包。

```
go get github.com/apache/rocketmq-client-go/v2@0e19ee6
```

2. 在对应的方法内创建生产者，如您需要发送普通消息，则在 `syncSendMessage.go` 文件内修改对应的参数。
   延时消息目前支持任意精度的延时，且不受 delay level 的影响。

普通消息

```
// 服务接入地址 （注意：需要在接入地址前面加上 http:// 或 https:// 否则无法解析）
   var serverAddress = "https://rocketmq-xxx.rocketmq.ap-
bj.public.tencenttdmq.com:8080"
   // 授权角色名
   var secretKey = "admin"
   // 授权角色密钥
   var accessKey = "eyJrZXlJZC...."
   // 生产者组名称
   var groupName = "group1"
   // 创建消息生产者
   p, _ := rocketmq.NewProducer(
       // 设置服务地址

producer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddress})),
       // 设置acl权限
       producer.WithCredentials(primitive.Credentials{
           SecretKey: secretKey,
           AccessKey: accessKey,
       }),
       // 设置生产组
       producer.WithGroupName(groupName),

       // 设置发送失败重试次数
       producer.WithRetry(2),
```

```
    )
    // 启动producer
    err := p.Start()
    if err != nil {
        fmt.Printf("start producer error: %s", err.Error())
        os.Exit(1)
    }
```

延时消息

```
// topic名称
    var topicName = "topic1"
    // 生产者组名称
    var groupName = "group1"
    // 创建消息生产者
    p, _ := rocketmq.NewProducer(
        // 设置服务地址

producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"http://rocketmq-
xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080"})),
        // 设置acl权限
        producer.WithCredentials(primitive.Credentials{
            SecretKey: "admin",
            AccessKey: "eyJrZXlJZC......",
        }),
        // 设置生产组
        producer.WithGroupName(groupName),

        // 设置发送失败重试次数
        producer.WithRetry(2),
    )
    // 启动producer
    err := p.Start()
    if err != nil {
        fmt.Printf("start producer error: %s", err.Error())
        os.Exit(1)
    }

    for i := 0; i < 1; i++ {
        msg := primitive.NewMessage(topicName, []byte("Hello RocketMQ Go Client! This
is a delay message."))
        // 设置延迟等级
        // 等级与时间对应关系：
        // 1s、 5s、 10s、 30s、  1m、 2m、 3m、 4m、 5m、 6m、 7m、 8m、 9m、 10m、 20m、
30m、 1h、 2h;
        // 1    2    3     4     5    6   7    8   9   10   11   12  13   14    15
16   17   18
        //如果想用延迟级别，那么设置下面这个方法
```

```
        msg.WithDelayTimeLevel(3)
        //如果想用任意延迟消息，那么设置下面这个方法，WithDelayTimeLevel 就不要设置了，单位为具体的
毫秒，如下则是10s后投递
        delayMills := int64(10 * 1000)
        msg.WithProperty("__STARTDELIVERTIME",
strconv.FormatInt(time.Now().UnixMilli()+delayMills, 10))
        // 发送消息

       res, err := p.SendSync(context.Background(), msg)
        if err != nil {
            fmt.Printf("send message error: %s\n", err)
        } else {
            fmt.Printf("send message success: result=%s\n", res.String())
        }
    }

    // 释放资源
    err = p.Shutdown()
    if err != nil {
        fmt.Printf("shutdown producer error: %s", err.Error())
    }
```

3. 发送消息同上（以同步发送方式为例）。

```
// topic名称
    var topicName = "topic1"
    // 构造消息内容
    msg := &primitive.Message{
        Topic: topicName, // 设置topic名称
        Body:  []byte("Hello RocketMQ Go Client! This is a new message."),
    }
    // 设置tag
    msg.WithTag("TAG")
    // 设置key
    msg.WithKeys([]string{"yourKey"})
    // 发送消息
    res, err := p.SendSync(context.Background(), msg)

    if err != nil {
        fmt.Printf("send message error: %s\n", err)
    } else {
        fmt.Printf("send message success: result=%s\n", res.String())
    }
```

参数	说明
topicName	Topic 名称在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
TAG	消息 TAG 标识。

yourKey	消息业务 key。

资源释放。

```
// 关闭生产者
    err = p.Shutdown()
    if err != nil {
        fmt.Printf("shutdown producer error: %s", err.Error())
    }
```

> **说明：**
> 异步发送、单向发送等，可参见 demo 示例 或参见 rocketmq-client-go 示例 。

4. 创建消费者。

```
// 服务接入地址 （注意：需要在接入地址前面加上 http:// 或 https:// 否则无法解析）
    var serverAddress = "http://rocketmq-xxx.rocketmq.ap-
bj.public.tencenttdmq.com:8080"
    // 授权角色名
    var secretKey = "admin"
    // 授权角色密钥
    var accessKey = "eyJrZXlJZC...."
    // 生产者组名称
    var groupName = "group11"
    // 创建consumer
    c, err := rocketmq.NewPushConsumer(
        // 设置消费者组
        consumer.WithGroupName(groupName),
        // 设置服务地址

consumer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddress})),
        // 设置acl权限
        consumer.WithCredentials(primitive.Credentials{
            SecretKey: secretKey,
            AccessKey: accessKey,
        }),
        // 设置从起始位置开始消费
        consumer.WithConsumeFromWhere(consumer.ConsumeFromFirstOffset),
        // 设置消费模式（默认集群模式）
        consumer.WithConsumerModel(consumer.Clustering),

        //广播消费，设置一下实例名，设置为应用的系统名即可。如果不设置，会使用pid，这会导致重启消费
重复
        consumer.WithInstance("xxxx"),
    )
    if err != nil {
        fmt.Println("init consumer2 error: " + err.Error())
        os.Exit(0)
```

```
    }
```

5. 消费消息。

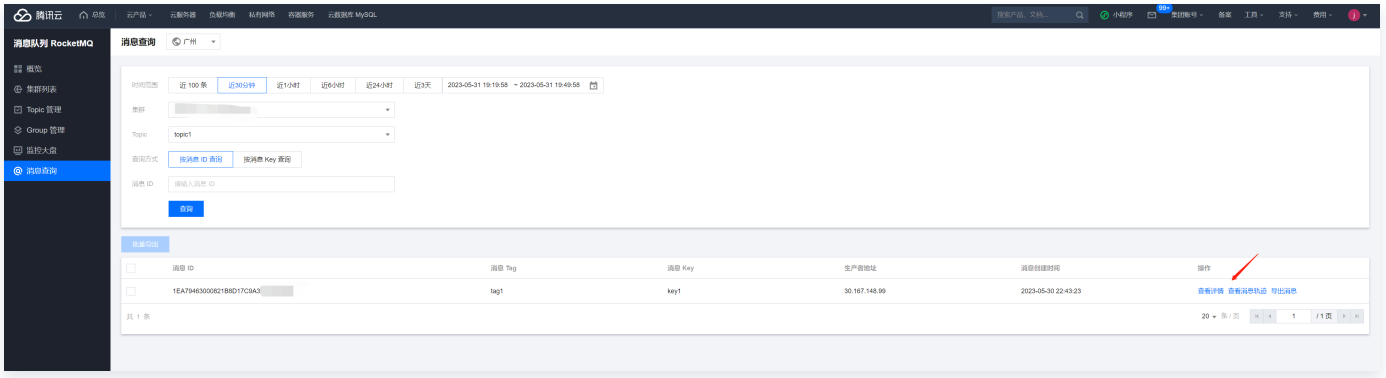
```
// topic名称
    var topicName = "topic1"
    // 设置订阅消息的tag
    selector := consumer.MessageSelector{
        Type:       consumer.TAG,
        Expression: "TagA || TagC",
    }
    // 设置重新消费的延迟级别，共支持18种延迟级别。下面是延迟级别与延迟时间的对应关系
    // 1   2   3    4    5   6   7   8   9   10  11  12  13  14   15   16   17  18
    // 1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m, 30m, 1h, 2h
    delayLevel := 1
    err = c.Subscribe(topicName, selector, func(ctx context.Context,
                                                        msgs
...*primitive.MessageExt) (consumer.ConsumeResult, error) {
        fmt.Printf("subscribe callback len: %d \n", len(msgs))
        // 设置下次消费的延迟级别
        concurrentCtx, _ := primitive.GetConcurrentlyCtx(ctx)
        concurrentCtx.DelayLevelWhenNextConsume = delayLevel // only run when return
consumer.ConsumeRetryLater

        for _, msg := range msgs {
            // 模拟重试3次后消费成功
            if msg.ReconsumeTimes > 3 {
                fmt.Printf("msg ReconsumeTimes > 3. msg: %v", msg)
                return consumer.ConsumeSuccess, nil
            } else {
                fmt.Printf("subscribe callback: %v \n", msg)
            }
        }
        // 模拟消费失败，回复重试
        return consumer.ConsumeRetryLater, nil
    })
    if err != nil {
        fmt.Println(err.Error())
    }
```

参数	说明
topicName	Topic 的名称，在控制台 Topic 页面复制。
Expression	消息 TAG 标识。
delayLevel	设置重新消费的延迟级别，共支持18种延迟级别。

6. 消费消息（消费者消费消息必须在订阅之后）。

```
// 开始消费
    err = c.Start()
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(-1)
    }
    time.Sleep(time.Hour)
    // 资源释放
    err = c.Shutdown()
    if err != nil {
        fmt.Printf("shundown Consumer error: %s", err.Error())
    }
```

7. 查看消费详情。发送完成消息后会得到一个消息ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 章节。



说明：

本文简单介绍了使用 Go 客户端进行简单的收发消息，更多操作可参见 Demo 或 rocketmq-client-go 示例 。

C++ SDK 使用

最近更新时间：2024-10-10 14:36:43

操作场景

本文以调用 C++ SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

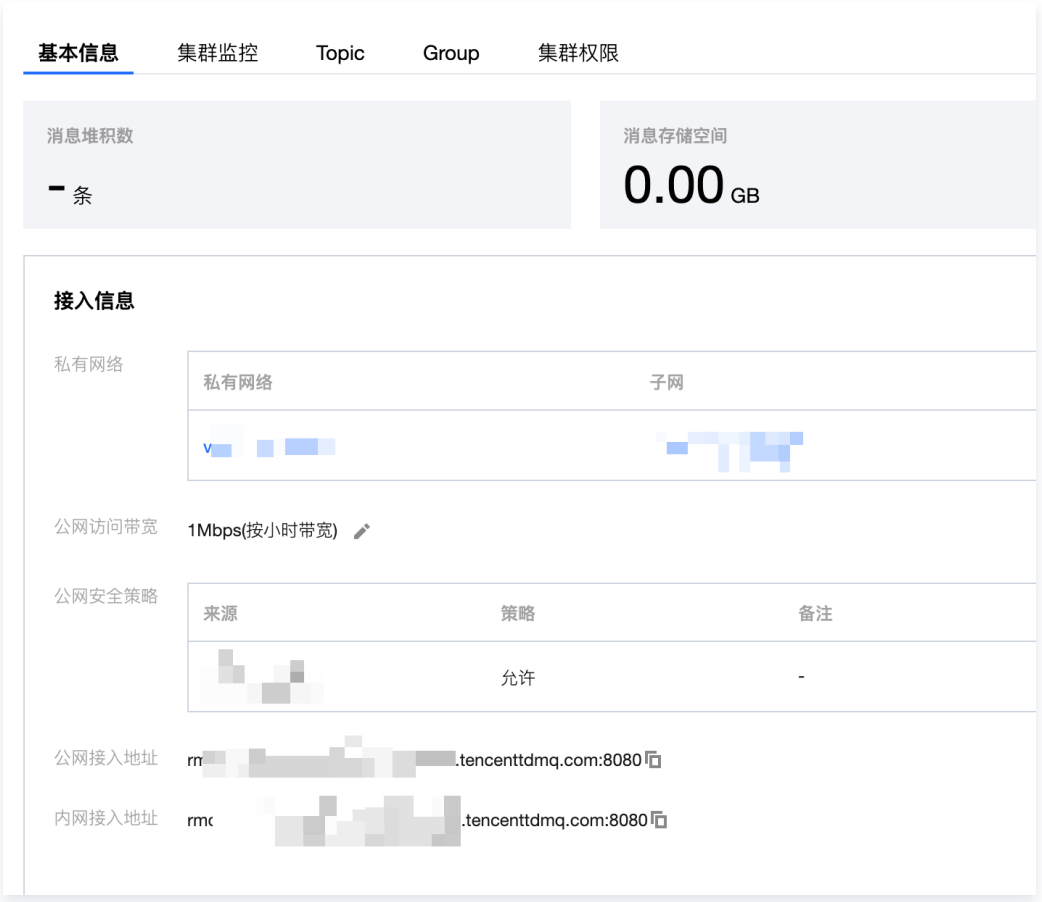
- 安装 GCC
- 下载 Demo

操作步骤

1. 准备环境。
   1.1 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**。
   1.2 在项目中引入 RocketMQ-Client-CPP 相关头文件及动态库。
2. 初始化消息生产者。

```
// 设置生产组名称
DefaultMQProducer producer(groupName);
// 设置服务接入地址
producer.setNamesrvAddr(nameserver);
// 设置用户权限
producer.setSessionCredentials(
    accessKey,  // 角色密钥
    secretKey, // 角色名称
    "");
// 设置命名空间（命名空间全称）
producer.setNameSpace(namespace);
// 请确保参数设置完成在启动之前
producer.start();
```

参数	说明
groupName	生产者组名称，在控制台集群管理中 Group tab 中获取。
nameserver	集群接入地址，在集群基本信息中，根据使用需求，使用不同的内网/公网接入地址。



secretKey	角色名称，在 集群管理 页面复制 accessSecret 复制。
accessKey	角色密钥，在 集群管理 页面复制 accessKey 复制。 

3. 发送消息。

```
// 初始化消息内容
    MQMessage msg(
        topicName,  // topic名称
        TAGS,       // 消息tag
        KEYS,       // 消息业务key
        "Hello cpp client, this is a message."  // 消息内容
    );

    try {
        // 发送消息
        SendResult sendResult = producer.send(msg);
```

```
        std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID: "
<< sendResult.getMsgId()
             << std::endl;
    } catch (MQException e) {
        std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() <<
std::endl;
    }
```

参数	说明
topicName	Topic 的名称，在控制台 topic 页面复制。
TAGS	用来设置消息的 TAG。
KEYS	设置消息业务 key。

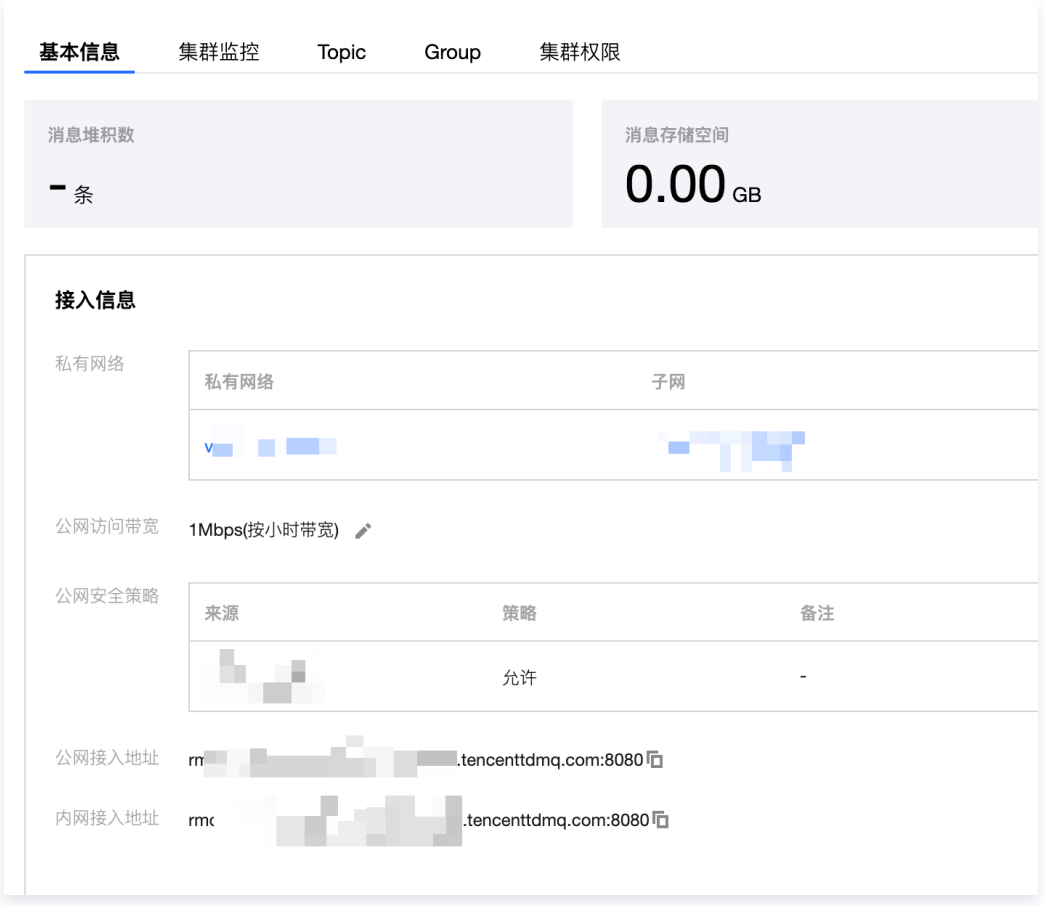
4. 资源释放。

```
// 释放资源
    producer.shutdown();
```

5. 初始化消费者。

```
// 消息监听
    class ExampleMessageListener : public MessageListenerConcurrently {
    public:
        ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
            for (auto item = msgs.begin(); item != msgs.end(); item++) {
                // 业务
                std::cout << "Received Message Topic:" << item->getTopic() << ",
MsgId:" << item->getMsgId() << ", TAGS:"
                          << item->getTags() << ", KEYS:" << item->getKeys() << ",
Body:" << item->getBody() << std::endl;
            }
            // 消费成功返回CONSUME_SUCCESS
            return CONSUME_SUCCESS;
            // 消费失败返回RECONSUME_LATER，该消息将会被重新消费
            // return RECONSUME_LATER;
        }
    };

    // 初始化消费者
    DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer(groupName);
    // 设置服务地址
    consumer->setNamesrvAddr(nameserver);
    // 设置用户权限
    consumer->setSessionCredentials(
        accessKey,
        secretKey,
```

```
    "");
// 设置命名空间
consumer->setNameSpace(namespace);
// 设置实例名称
consumer->setInstanceName("CppClient");

// 请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
ExampleMessageListener *messageListener = new ExampleMessageListener();
// 订阅消息
consumer->subscribe(topicName, TAGS);
// 设置消息监听
consumer->registerMessageListener(messageListener);

// 准备工作完成，必须调用启动函数，才可以正常工作。
consumer->start();
```

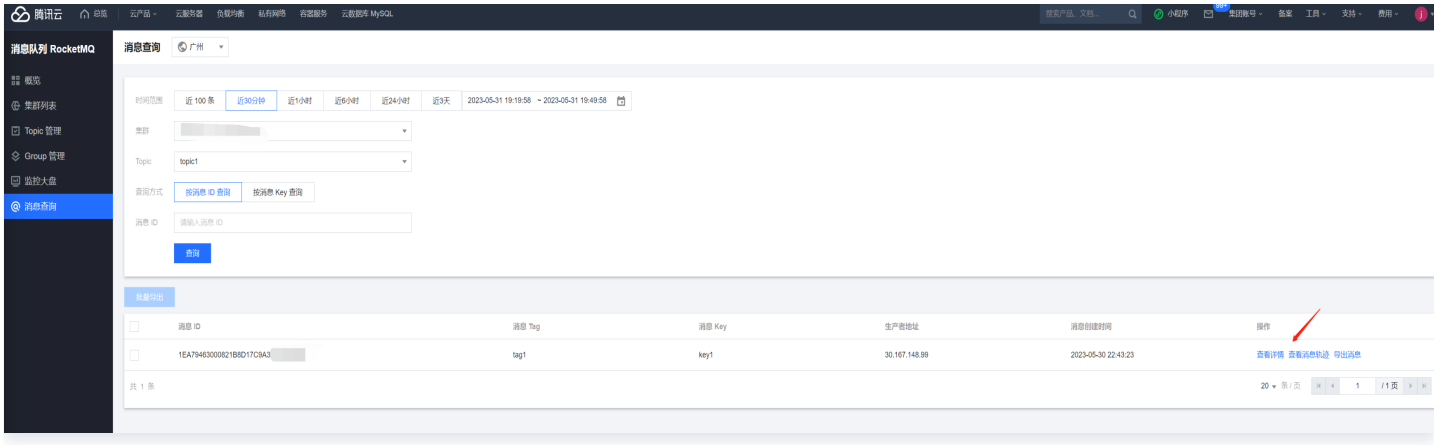
参数	说明
groupName	消费者组名称。在控制台集群管理中 Group 页签中获取。
nameserver	集群接入地址，在集群管理页面操作列的获取接入地址获取。 
secretKey	角色名称，在 集群管理 页面复制 accessSecret 复制。

accessKey	角色密钥，在 集群管理 页面复制 accessKey 复制。 
topicName	Topic 的名称，在控制台 topic 页面复制。
TAGS	用来设置订阅消息的 TAG。

6. 资源释放。

```
// 资源释放
consumer->shutdown();
```

7. 发送完成消息后会得到一个消息ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 。



说明

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 Demo 或 RocketMQ-Client-CPP 示例 。

Python SDK 使用

最近更新时间：2024-11-01 10:20:42

操作场景

本文以调用 Python SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

- 完成资源创建与准备
- 安装 Python
- 安装 pip
- 下载 Demo

操作步骤

步骤1：准备环境

Rocketmq-client Python 基于 rocketmq-client-cpp 进行包装，因此需要先安装 `librocketmq` 。

> **说明：**
> - 目前Python客户端仅支持 Linux 和 macOS 操作系统，暂不支持 Windows 系统。
> - 在使用 Python SDK 时要注意安装的 Python 支持的底层芯片架构类型（x86 或是 ARM），例如使用 `'64bit','ELF'`（即 x86_64 架构）的 Python 版本，则使用 ARM 芯片的 macOS 系统会出现报错。

1. 安装 librocketmq（版本2.0.0及以上）， 安装教程参见 librocketmq 安装 。
2. 执行如下命令安装 rocketmq-client-python。

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

步骤2：生产消息

创建并编译运行生产消息程序。

```
from rocketmq.client import Producer, Message

    # 初始化生产者，并设置生产组信息，group1
    producer = Producer(groupName)
    # 设置服务地址
    producer.set_name_server_address(nameserver)
    # 设置权限（角色名和密钥）
    producer.set_session_credentials(
         accessKey,   # 角色密钥
        secretKey,   # 角色名称
        ''
    )
```

```
# 启动生产者
producer.start()

# 组装消息    topic名称，在控制台 topic 页面复制。
msg = Message(topicName)
# 设置keys，用户自定义，如果没用，就是空。
KEYS = "YourKey"
msg.set_keys(KEYS)
# 设置tags，用户自定义，如果没用，就是空。
TAGS = "yourTag"
msg.set_tags(TAGS)
# 消息内容
msg.set_body('This is a new message.')

# 发送同步消息
ret = producer.send_sync(msg)
print(ret.status, ret.msg_id, ret.offset)
# 资源释放
producer.shutdown()
```

参数	说明
groupName	生产者组名称。在控制台集群管理中 Group tab 中获取。
nameserver	集群接入地址，在集群基本信息中，根据使用需求，使用不同的内网/公网接入地址 

secretKey	角色名称，在 集群管理 页面复制 accessSecret 复制。
accessKey	角色密钥，在 集群管理 页面复制 accessKey 复制。 
topicName	Topic 的名称，在控制台 Topic 页面复制。
TAGS	用来设置消息的 TAG。
KEYS	设置消息业务 key。

当前开源社区的 Python 客户端生产消息存在一定缺陷，导致同个 Topic 的不同队列间负载不均，详情可参见 缺陷详情 。

步骤3：消费消息

创建并编译运行消费消息程序。

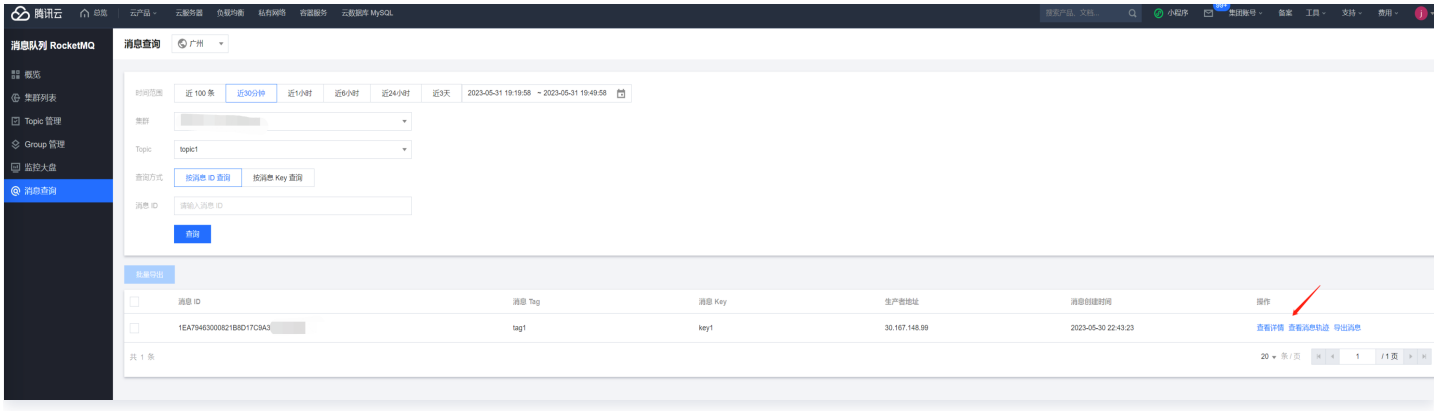
```
import time

    from rocketmq.client import PushConsumer, ConsumeStatus


    # 消息处理回调
    def callback(msg):
        # 模拟业务
        print('Received message. messageId: ', msg.id, ' body: ', msg.body)
        # 消费成功回复CONSUME_SUCCESS
        return ConsumeStatus.CONSUME_SUCCESS
        # 消费成功回复消息状态
        # return ConsumeStatus.RECONSUME_LATER


    # 初始化消费者，并设置消费者组信息
    consumer = PushConsumer(groupName)
    # 设置服务地址
    consumer.set_name_server_address(nameserver)
    # 设置权限（角色名和密钥）
    consumer.set_session_credentials(
        accessKey,     # 角色密钥
        secretKey,    # 角色名称
        ''
    )
    # 订阅topic,默认*订阅，根据需求修改
    TAGS = "*"
    consumer.subscribe(topicName, callback, TAGS)
    print(' [Consumer] Waiting for messages.')
    # 启动消费者
```

```
    consumer.start()

    while True:
        time.sleep(3600)
    # 资源释放
    consumer.shutdown()
```

参数	说明
groupName	消费者 Group 的名称，在控制台 Group 页面复制。
nameserver	同生产地址。
secretKey	同生产消息的获取方式。
accessKey	同生产消息的获取方式。
topicName	Topic 的名称，在控制台 Topic 页面复制。
TAGS	设置订阅消息的tag，默认为"*"，表示订阅所有消息。

步骤4：查看消费详情

发送完成消息后会得到一个消息ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情请参见 消息查询 。



说明

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 Demo 或 RocketMQ-Client-Python示例 。

C# SDK

最近更新时间：2024-12-12 12:08:42

操作场景

本文以调用 C# SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

- 完成资源创建与准备
- 安装 DotNet
- 下载 Demo

操作步骤

步骤1：准备环境

根据自己的 IDE 环境，新建一个项目，然后安装 NewLife.RocketMQ 依赖。

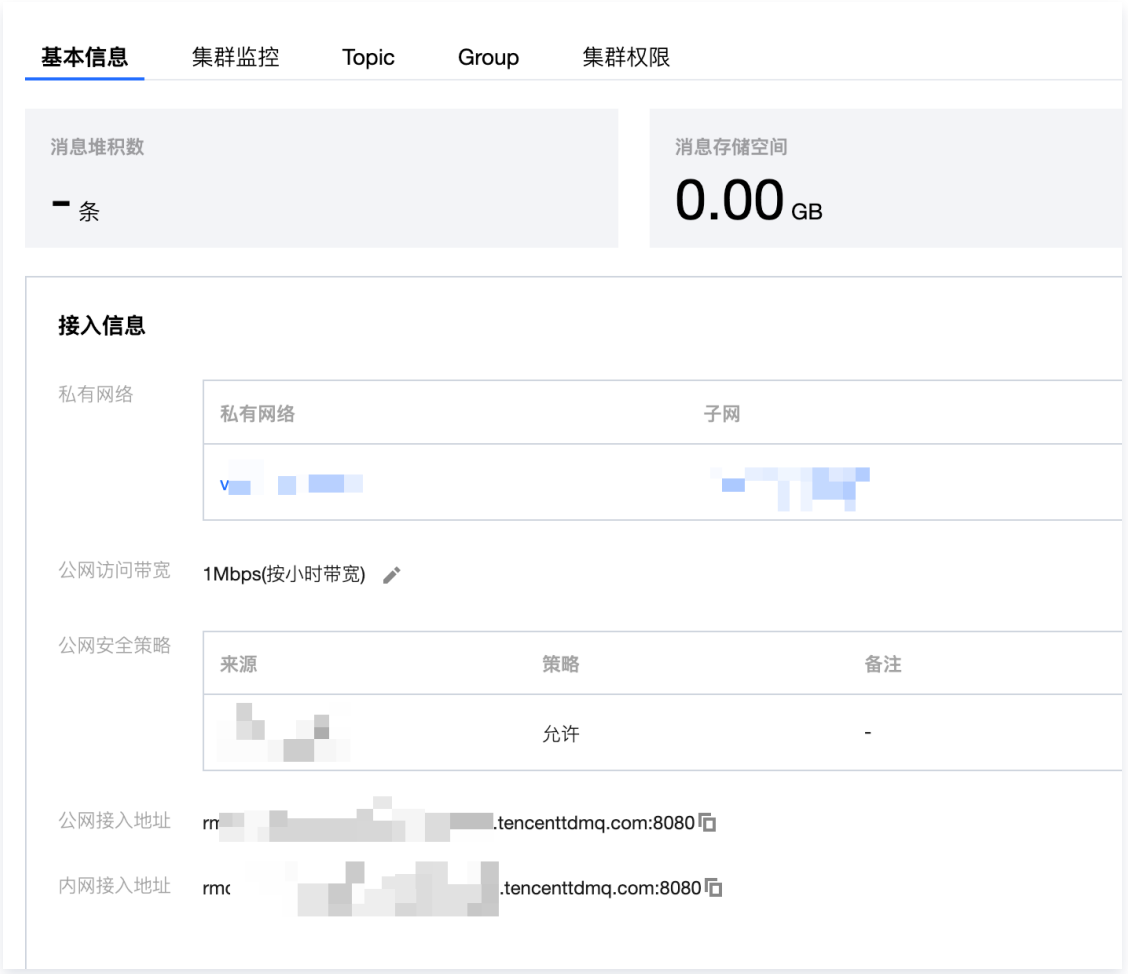
```
dotnet add package NewLife.RocketMQ --version 2.0.2022.325-beta0806
```

步骤2：生产消息

创建并编译运行生产消息程序。

```
//生产者
var mq = new Producer
{
    Topic = "TopicTest1", //不需要拼接namespace等
    NameServerAddress = "127.0.0.1:9876",// 腾讯云页面填写
    Log = XTrace.Log,
    AclOptions = new AclOptions
    {
        AccessKey = "页面的ak",
        SecretKey = "页面的sk",
    },
};
mq.Start();
for (var i = 0; i < 10; i++)
{
    var str = "生产消息测试" + i;
    var sr = mq.Publish(str, "TagA");
}
```

参数	说明
nameserver	集群接入地址，在控制台集群管理页面操作列的获取接入地址获取。新版共享集群与专享集群命名接入点地址在命名空间列表获取。

	
secretKey	角色名称，在 集群权限 页面复制。
accessKey	角色密钥，在 集群权限 页面复制密钥列复制。 
topicName	Topic 的名称，在控制台 Topic 页面复制
TAGS	用来设置消息的 TAG。
KEYS	设置消息业务 key。

当前开源社区的 Python 客户端生产消息存在一定缺陷，导致同个 Topic 的不同队列间负载不均，详情可参见 缺陷详情 。

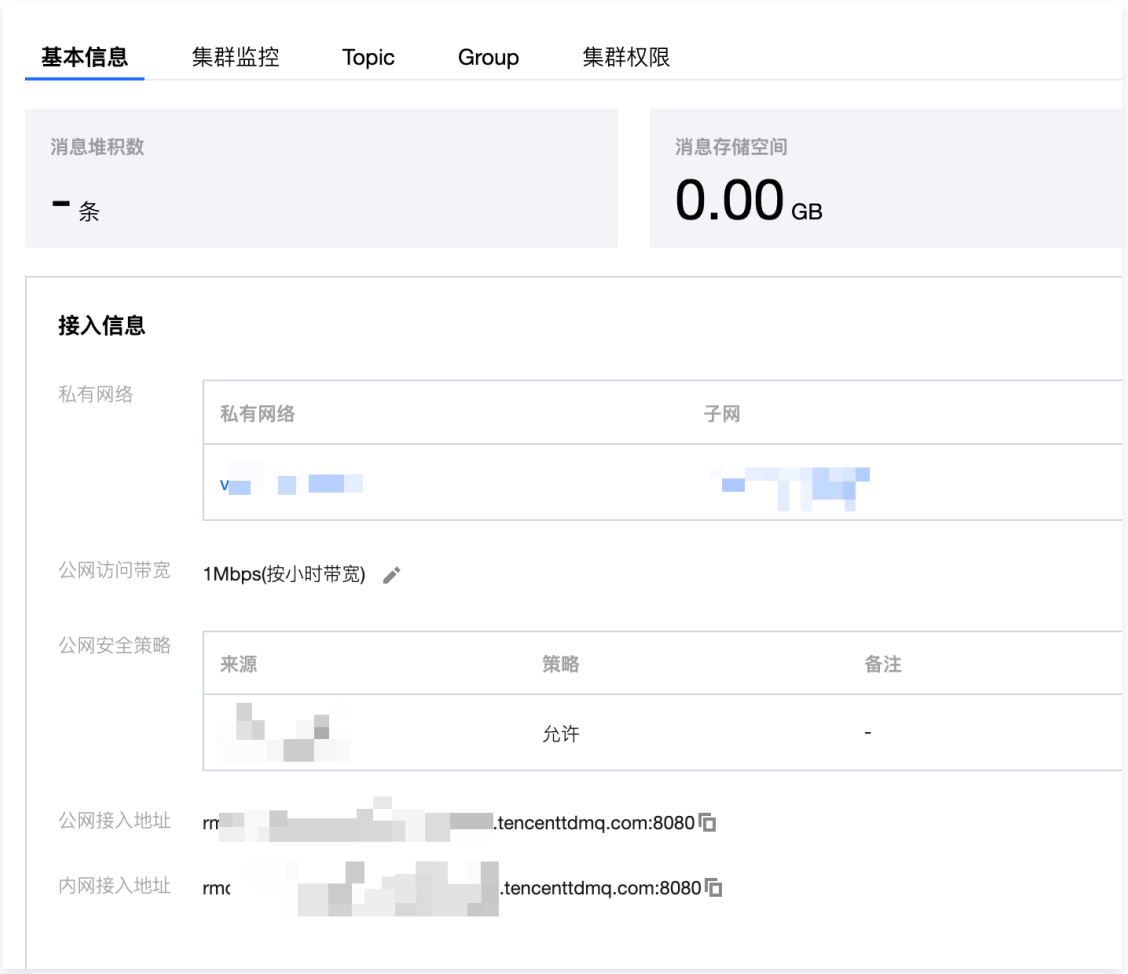
步骤3：消费消息

创建并编译运行消费消息程序。

```
//消费者
var consumer = new Consumer
{
    Topic = "TopicTest1",//对于非通用版集群
    Group = "test",
```

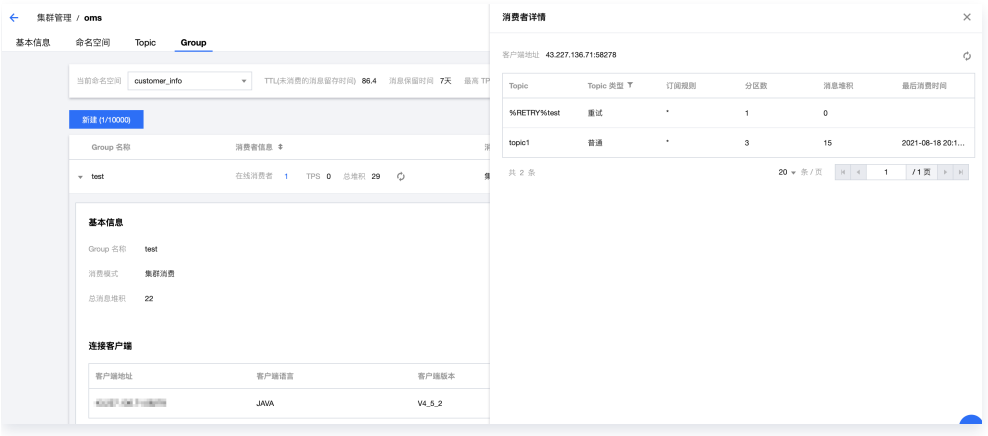
```
    NameServerAddress = "127.0.0.1:9876",
    FromLastOffset = true,
    SkipOverStoredMsgCount = 0,
    BatchSize = 20,
    Log = XTrace.Log,
    AclOptions = new AclOptions
    {
        AccessKey = "页面的ak",
        SecretKey = "页面的sk",
    },
};
consumer.OnConsume = (q, ms) =>
{
    XTrace.WriteLine("[{0}@{1}]收到消息 [{2}]", q.BrokerName, q.QueueId, ms.Length);
    foreach (var item in ms.ToList())
    {
        XTrace.WriteLine($"消息 {item.Keys}，发送时间{item.BornTimestamp.ToDateTime()}，
内容 {item.Body.ToStr()}");
    }
    return true;
};
consumer.Start();
```

参数	说明
groupName	消费者 Group 的名称，在控制台 Group 页面复制
nameserver	集群接入地址，在控制台集群管理页面操作列的获取接入地址获取。新版共享集群与专享集群命名接入点地址在命名空间列表获取。

	
secretKey	角色名称，在 集群权限 页面复制。
accessKey	角色密钥，在 集群权限 页面复制密钥列复制。 
topicName	Topic 的名称，在控制台 Topic 页面复制。
TAGS	设置订阅消息的tag，默认为"*"，表示订阅所有消息

步骤4：查看消费详情

登录 RocketMQ 控制台，在**集群管理** > **Group** 页面，可查看与 Group 连接的客户端列表，单击操作列的**查看详情**，可查看消费者详情。



> **说明**
>
> 上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 。

Spring Cloud Stream 使用

最近更新时间：2024-10-08 11:00:12

操作场景

本文以调用 Spring Cloud Stream 接入为例介绍实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

- 完成资源创建与准备
- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo 或者前往 GitHub 项目

操作步骤

步骤1：引入依赖

在 pom.xml 中引入 spring-cloud-starter-stream-rocketmq 相关依赖。当前建议版本 2021.0.5.0，同时需要排除依赖，使用4.9.7的 SDK。

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
    <version>2021.0.5.0</version>
   <exclusions>
     <exclusion>
         <groupId>org.apache.rocketmq</groupId>
         <artifactId>rocketmq-client</artifactId>
     </exclusion>
     <exclusion>
         <groupId>org.apache.rocketmq</groupId>
         <artifactId>rocketmq-acl</artifactId>
      </exclusion>
   </exclusions>
</dependency>
<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>
```

步骤2：添加配置

在配置文件中增加 RocketMQ 相关配置。

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          # 服务地址全称
          name-server: rmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080
          # 角色名称
          secret-key: admin
          # 角色密钥
          access-key: eyJrZXlJZ...
          # producer group
          group: producerGroup
        bindings:
          # channel名称，与spring.cloud.stream.bindings下的channel名称对应
          Topic-TAG1-Input:
            consumer:
              # 订阅的tag类型，根据消费者实际情况进行配置（默认是订阅所有消息）
              subscription: TAG1
          # channel名称
          Topic-TAG2-Input:
            consumer:
              subscription: TAG2
      bindings:
        # channel名称
        Topic-send-Output:
          # 指定topic，对应创建的topic名称
          destination: TopicTest
          content-type: application/json
        # channel名称
        Topic-TAG1-Input:
          destination: TopicTest
          content-type: application/json
          group: consumer-group1
        # channel名称
        Topic-TAG2-Input:
          destination: TopicTest
          content-type: application/json
          group: consumer-group2
```

注意

配置方面 `2.2.5-RocketMQ-RC1` 与 `2.2.5.RocketMQ.RC2` 的订阅配置项为 `subscription`，其他低版本订阅配置项为 `tags`。

其他版本完整配置项参考如下：

```
spring:
```

```
cloud:
  stream:
    rocketmq:
      bindings:
        # channel名称，与spring.cloud.stream.bindings下的channel名称对应
        Topic-test1:
          consumer:
            # 订阅的tag类型，根据消费者实际情况进行配置（默认是订阅所有消息）
            tags: TAG1
        # channel名称
        Topic-test2:
          consumer:
            tags: TAG2
      binder:
        # 服务地址全称
        name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080
        # 角色名称
        secret-key: admin
        # 角色密钥
        access-key: eyJrZXlJZ...
    bindings:
      # channel名称
      Topic-send:
        # 指定topic,
        destination: topic1
        content-type: application/json
        # 要使用group全称
        group: group1
      # channel名称
      Topic-test1:
        destination: topic1
        content-type: application/json
        group: group1
      # channel名称
      Topic-test2:
        destination: topic1
        content-type: application/json
        group: group2
```

参数	说明
name-server	集群接入地址，在控制台集群管理页面的集群列表操作栏的接入地址处获取。新版共享集群与专享集群命名接入点地址在命名空间列表获取。
secret-key	角色名称，在 集群管理 页面复制 accessSecret 复制。
access-key	角色密钥，在 集群管理 页面复制 accessKey 复制。

	
group	生产者 Group 的名称，在控制台 Group 页面复制。
destination	Topic 的名称，在控制台 topic 页面复制。

步骤3：配置 channel

channel 分为输入和输出，可根据自己的业务进行单独配置。

```
/**
 * 自定义通道 Binder
 */
public interface CustomChannelBinder {

    /**
     * 发送消息(消息生产者)
     * 绑定配置中的channel名称
     */
    @Output("Topic-send-Output")
    MessageChannel sendChannel();


    /**
     * 接收消息1(消费者1)
     * 绑定配置中的channel名称
     */
    @Input("Topic-TAG1-Input")
    MessageChannel testInputChannel1();

    /**
     * 接收消息2(消费者2)
     * 绑定配置中的channel名称
     */
    @Input("Topic-TAG2-Input")
    MessageChannel testInputChannel2();
}
```

步骤4：添加注解

在配置类或启动类上添加相应注解，如果有多个 binder 配置，都要在此注解中进行指定。

```
@EnableBinding({CustomChannelBinder.class})
```

步骤5：发送消息

1. 在要发送消息的类中，注入 `CustomChannelBinder` 。

```
@Autowired
     private CustomChannelBinder channelBinder;
```

2. 发送消息，调用对应的输出流 channel 进行消息发送。

```
Message<String> message = MessageBuilder.withPayload("This is a new
message.").build();
          channelBinder.sendChannel().send(message);
```

步骤6：消费消息

```
@Service
public class StreamConsumer {
    private final Logger logger =
LoggerFactory.getLogger(StreamDemoApplication.class);

    /**
     * 监听channel（配置中的channel 名称）
     *
     * @param messageBody 消息内容
     */
    @StreamListener("Topic-TAG1-Input")
    public void receive(String messageBody) {
        logger.info("Receive1: 通过stream收到消息，messageBody = {}", messageBody);
    }

    /**
     * 监听channel（配置中的channel 名称）
     *
     * @param messageBody 消息内容
     */
    @StreamListener("Topic-TAG2-Input")
    public void receive2(String messageBody) {
        logger.info("Receive2: 通过stream收到消息，messageBody = {}", messageBody);
    }
}
```

步骤7：本地测试

本地启动项目之后，可以从控制台看到启动成功。
浏览器访问 `http://localhost:8080/test-simple` 可以看到发送成功。观察开发 IDE 的输出日志。

```
2023-02-23 19:19:00.441  INFO 21958 --- [nio-8080-exec-1]
c.t.d.s.controller.StreamController      : Send: 通过stream发送消息，messageBody =
GenericMessage [payload={"key":"value"}, headers={id=3f28bc70-da07-b966-a922-
14a17642c9c4, timestamp=1677151140353}]
2023-02-23 19:19:01.138  INFO 21958 --- [nsumer-group1_1]
c.t.d.s.StreamDemoApplication            : Receive1: 通过stream收到消息，messageBody =
{"headers":{"id":"3f28bc70-da07-b966-a922-
14a17642c9c4","timestamp":1677151140353},"payload":{"key":"value"}}
```

可以看到。发送了一条 TAG1 的消息，同时也只有 TAG1 的订阅者收到了消息。

> **说明**
> 具体使用可参见 GitHub Demo 或 Spring cloud stream 官网 。

Spring Boot Starter 使用

最近更新时间：2024-10-10 14:40:42

操作场景

本文以调用 Spring Boot Starter SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

- 完成资源创建与准备
- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo 或者前往 GitHub 项目

操作步骤

步骤1：添加依赖

在 pom.xml 中添加依赖。

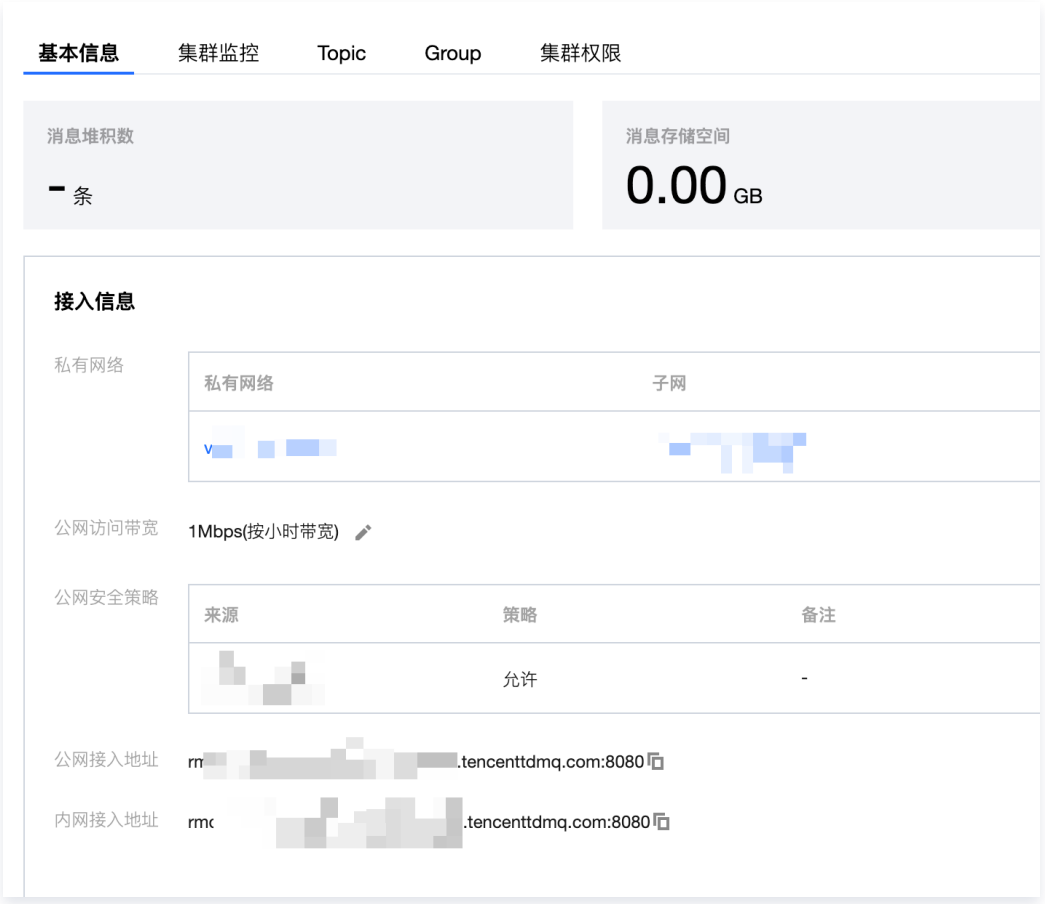
```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.2.2</version>
     <exclusions>
             <exclusion>
                 <groupId>org.apache.rocketmq</groupId>
                 <artifactId>rocketmq-client</artifactId>
             </exclusion>
             <exclusion>
                 <groupId>org.apache.rocketmq</groupId>
                 <artifactId>rocketmq-acl</artifactId>
              </exclusion>
      </exclusions>
</dependency>
 <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>
```

步骤2：准备配置

在配置文件中添加配置信息。

```
server:
     port: 8082

   #rocketmq配置信息
   rocketmq:
     # tdmq-rocketmq服务接入地址
     name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080
     # 生产者配置
     producer:
       # 生产者组名
       group: group111
       # 角色密钥
       access-key: eyJrZXlJZC....
       # 已授权的角色名称
       secret-key: admin
     # 消费者公共配置
     consumer:
       # 角色密钥
       access-key: eyJrZXlJZC....
       # 已授权的角色名称
       secret-key: admin

     # 用户自定义配置

     producer1:
       topic: testdev1
     consumer1:
       group: group111
       topic: testdev1
       subExpression: TAG1
     consumer2:
       group: group222
       topic: testdev1
       subExpression: TAG2
```

参数	说明
name-server	集群接入地址，在集群基本信息中，根据使用需求，使用不同的内网/公网接入地址



group	生产者 Group 的名称，在控制台 Group 页面复制。
secret-key	角色名称，在 集群管理 页面复制 accessSecret 复制。
access-key	角色密钥，在 集群管理 页面复制 accessKey 复制。 
topic	Topic 的名称，在控制台 topic 页面复制。
subExpression	用来设置消息的 TAG。

步骤3：发送消息

1. 在需要发送消息的类中注入 `RcoketMQTemplate` 。

```
@Value("${rocketmq.producer1.topic}")
      private String topic;  // topic名称

      @Autowired
      private RocketMQTemplate rocketMQTemplate;
```

2. 发送消息，消息体可以是自定义对象，也可以是 Message 对象（org.springframework.messaging包中）。

```
SendResult sendResult = rocketMQTemplate.syncSend(destination, message);
/*-----------------------------------------------------------------------*/
rocketMQTemplate.syncSend(destination, MessageBuilder.withPayload(message).build())
```

3. 完整示例如下。

```
/**
* Description: 消息生产者
*/
@Service
public class SendMessage {
// 需要使用topic全称，所以进行topic名称的拼接，也可以自己设置  格式：topic名称
 @Value("${rocketmq.producer1.topic}")
 private String topic;
     @Autowired
 private RocketMQTemplate rocketMQTemplate;
         /**
     * 同步发送
     *
     * @param message 消息内容
     * @param tags    订阅tags
     */
 public void syncSend(String message, String tags) {
         // springboot不支持使用header传递tags，根据要求，需要在topic后进行拼接 formats:
`topicName:tags`，不拼接标识无tag
         String destination = StringUtils.isBlank(tags) ? topic : topic + ":" +
tags;
         SendResult sendResult = rocketMQTemplate.syncSend(destination,
                         MessageBuilder.withPayload(message)
                                         .setHeader(MessageConst.PROPERTY_KEYS,
"yourKey")    // 指定业务key
                                         .build());
         System.out.printf("syncSend1 to topic %s sendResult=%s %n", topic,
sendResult);
  }
}
```

> **说明**
> 该示例为同步发送。异步发送，单向发送等请参见 Demo 或者前往 GitHub 项目 。

步骤4：消费消息

```
@Service
   @RocketMQMessageListener(
           consumerGroup = "${rocketmq.consumer1.group}",   // 消费组，格式：group名称
           // 需要使用topic全称，所以进行topic名称的拼接，也可以自己设置  格式：topic名称
```

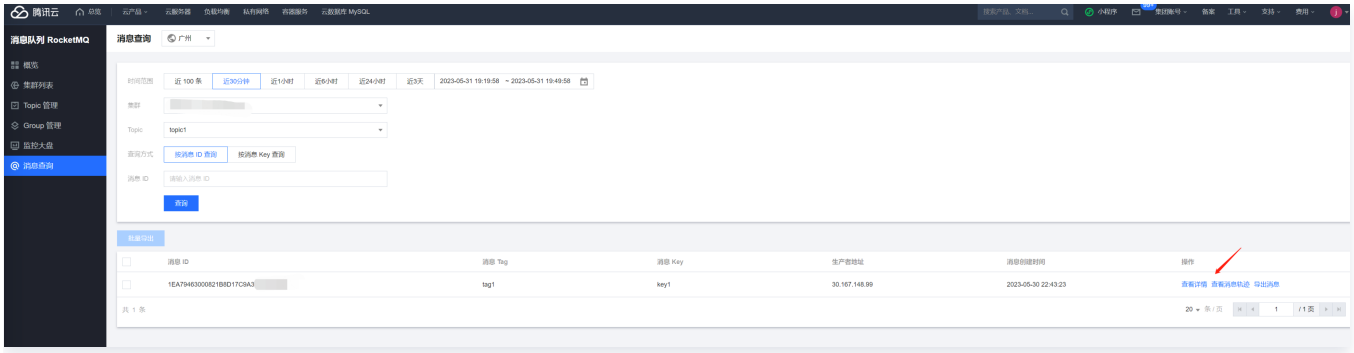
```
        topic = "${rocketmq.consumer1.topic}",
        selectorExpression = "${rocketmq.consumer1.subExpression}" // 订阅表达式，不
配置表示订阅所有消息
    )
    public class MessageConsumer implements RocketMQListener<String> {

        @Override
        public void onMessage(String message) {
            System.out.println("Tag1Consumer receive message: " + message);
        }
    }
```

可根据业务需求配置多个消费者。消费者其他配置可根据具体业务需求进行配置。

> **说明**
> 完整示例参见 下载 Demo 或者前往 GitHub 项目

步骤5：查看消费详情

发送完成消息后会得到一个消息 ID（messageID），开发者可以在 “消息查询” 页面查询刚刚发送的消息，如下图所示；并且可以查看特定消息的详情和轨迹等信息，详情见 消息查询 章节。



实践教程
RocketMQ 常见概念命名规范

最近更新时间：2024-10-14 15:35:11

本文介绍在使用 RocketMQ 过程中常见概念的命名规范以及使用规范。

命名规范

topic

- 不能为空，只能包含字母、数字、“-”及“_”，3-64 字符
- 建议格式：String.format("tp_%s_%s", "系统名","业务名")
- 例如：tp_order_check

tag

- 允许为空，只要是字符串就可以，用来做 topic 下的二级消息过滤
- 建议格式：String.format ("tag_%s","业务动作或类别")
- 例如：tag+业务动作，例如：订单创建的 tag为：tag_create；订单关闭的tag为：tag_close。

keys

- 允许为空，建议设置，只要是字符串或字符串数组都可以，用来在控制台查询消息或查询消息轨迹
- 建议格式：String.format("%s","业务唯一性 ID")
- 例如：订单 ID，交易 ID 或流水号，TraceID 等唯一性 ID。

producer group

- 不能为空，3-64个字符，只能包含字母、数字、“-”及“_”
- 建议格式：String.format("pg_%s_%s", "系统名","业务名")
- 例如：pg_transfer_check

consumer group

- 不能为空，3-64个字符，只能包含字母、数字、“-”及“_”
- 建议格式：String.format("cg_%s_%s", "系统名","业务名")
- 例如：cg_transfer_check

role

- 不能为空，只支持数字、大小写字母、分隔符（"_"、"-"），不能超过 32 个字符
- 不同业务读写权限的标记，建议格式：业务名 + 发送/消费
- 例如：role_order_send, role_order_consume，role_order_all

clientId

clientId 表示一个客户端实例 ID，不同客户端不可重复。clientId 不能在客户端直接设置，instanceName 是 clientId 的可选组成部分，可以通过调整instanceName 修改 clientId。

分类	生成策略
生产者	${当前IP}@${instanceName}
集群消费者	${当前IP}@${instanceName}
广播消费者	${当前IP}@${instanceName}

```
public String buildMQClientId() {
        StringBuilder sb = new StringBuilder();
        sb.append(this.getClientIP());

        sb.append("@");
        sb.append(this.getInstanceName());
        if (!UtilAll.isBlank(this.unitName)) {
            sb.append("@");
            sb.append(this.unitName);
        }

        if (enableStreamRequestType) {
            sb.append("@");
            sb.append(RequestType.STREAM);
        }

        return sb.toString();
    }
```

instanceName

实例名，默认场景不需要用户特殊设置，默认系统会通过以下代码随机生成一个唯一值。

```
public void changeInstanceNameToPID() {
        if (this.instanceName.equals("DEFAULT")) {
            this.instanceName = UtilAll.getPid() + "#" + System.nanoTime();
        }
    }
```

广播消费者在每次启动时，需要保持消费者实例名不变，读取客户端本地的进度文件，需要显式地设置 instanceName，并且广播消费需要保持当前客户端 IP 启动前后不变， 如果容器部署，需要设置固定 pod IP 调度，否则重启期间的广播消息会漏消费。
建议格式：String.format("instance-%s-%s","系统名","业务名")。

使用规范

生产者

- **【强制】一个领域服务只能有一个 topic。**
- **【强制】领域服务发送消息时必须根据业务动作设置 tag。**
- **【强制】在 producer 发送消息时必须设置 keys。**

- 【强制】消息发送成功或者失败要打印消息日志，务必要打印 SendResult 和 key 字段。
- 【建议】消息发送失败后建议将消息存储到 db，然后由定时器类线程进行定时重试，确保消息达到 broker。
- 【建议】对于可靠性要求不高的业务场景可以使用 oneway 消息。
- 【强制】新建生产者时必须指定生产者分组。

消费者

- 【强制】新建消费者时必须指定消费者分组。
- 【强制】消息消费者无法避免消息重复，所以需要业务服务来保证消息消费幂等。
- 【建议】为了提高消费并行度，可以在同一个 ConsumerGroup 下启动多个 Consumer 实例或者通过修改 ConsumeThreadMin 和consumeThreadMax 来提高单个 Consumer 的并行消费能力。
- 【建议】为了增加业务吞吐量，可以通过设置 consumer 的 consumeMessageBatchMaxSize 来批量消费消息。
- 【建议】发生消息堆积时，如果业务对数据要求不高时，可以选择丢弃不重要的消息。
- 【建议】如果消息量较少，建议在消费入口打印消息、消费耗时等，方便后续排查问题。

使用社区版 HTTP SDK 接入 Java SDK 发送与接收普通消息

最近更新时间：2024-10-14 15:05:51

TDMQ RocketMQ 5.x 版兼容了社区版 HTTP SDK 的接入，如果您此前使用的客户端使用了社区版 HTTP SDK，您在切换到 TDMQ RocketMQ 5.x 版后，您无需在客户端进行任何代码改造。

操作场景

如果当前您已使用了 HTTP 协议进行消息的收发，在您的客户端引入开源 HTTP SDK 后，TDMQ RocketMQ 5.x 版支持用户通过内网或公网使用 HTTP 协议接入。
本文以调用 Java SDK 为例介绍通过 HTTP SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **注意：**
> 暂不支持通过使用 HTTP 协议实现事务消息。

前提条件

- 完成资源创建与准备 。
- 安装1.8或以上版本 JDK 。
- 安装2.5或以上版本 Maven 。
- 通过 Maven 方式引入依赖，在 pom.xml 文件中添加对应语言的 SDK 依赖。

重试机制

HTTP 采用固定重试间隔的机制：

重试间隔	最大重试次数
5分钟	可通过修改消费组配置实现自定义最大重试次数，默认 16 次。

> **说明：**
> - 客户端在重试间隔内 ACK 这条消息，表示消费成功，不会重试。
> - 重试间隔到期后客户端仍未 ACK，客户端会重新消费到这条消息。
> - 每次消费的消息句柄只在重试间隔内有效，过期无效。

操作步骤

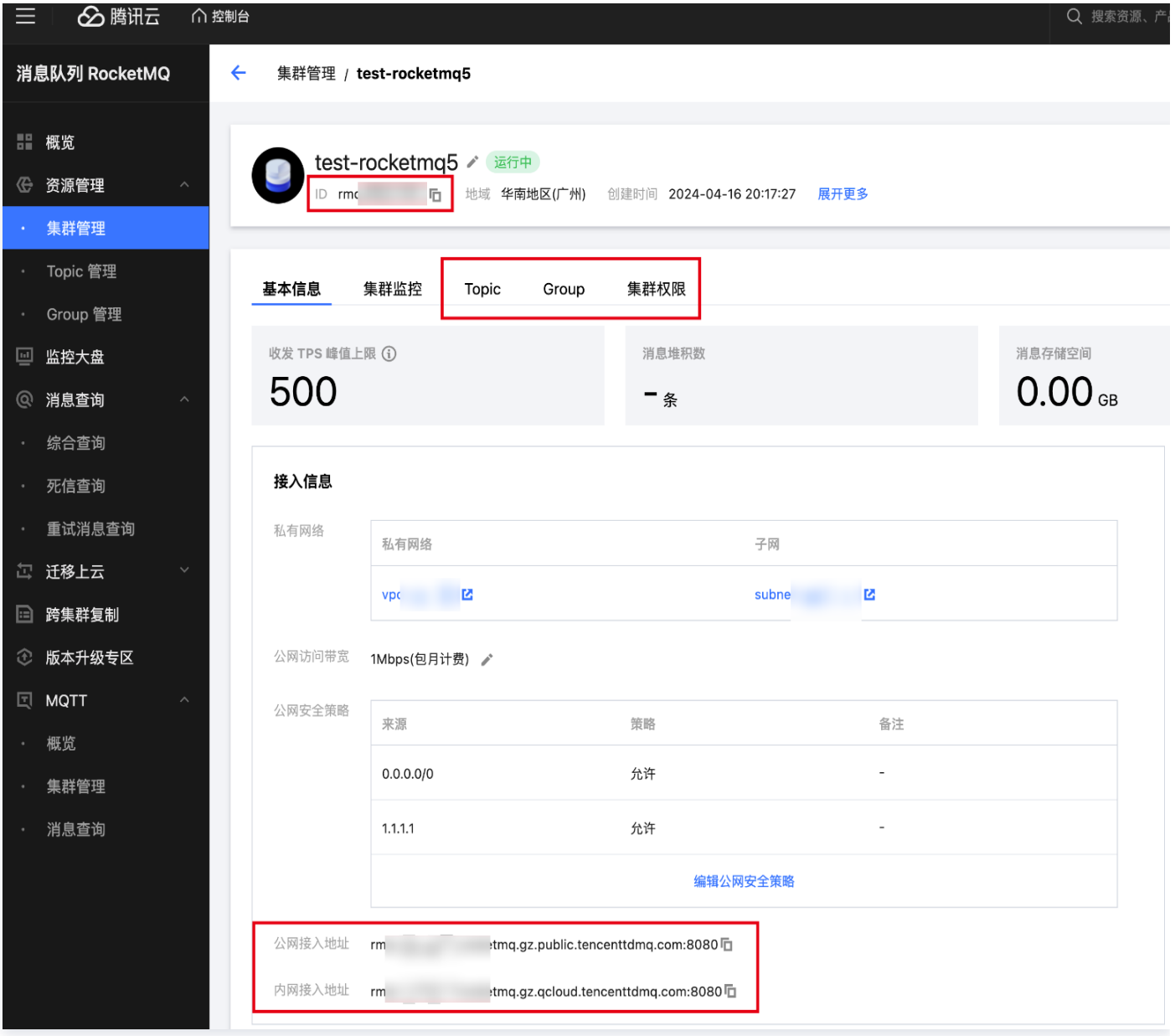
步骤1：安装 Java 依赖库

在 Java 项目中引入社区版 HTTP SDK 依赖。

步骤2：获取参数

1. 登录消息队列 RocketMQ 控制台，选择集群。

2. 复制集群 ID、接入地址等参数。



步骤3：生产消息

创建消息生产者

```
// 获取Client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

// 获取Topic的Producer
MQProducer producer = mqClient.getProducer(instanceId, topicName);
```

参数	说明

endpoint	接入地址，在基本信息页面获取。
accessKey	角色 AccessKey，在集群权限页面获取。
secretKey	角色 SecretKey，在集群权限页面获取。
instanceId	集群 ID。
topicName	主题名称，在 Topic 页面获取。

发送消息

```
try {
    for (int i = 0; i < 10; i++) {
        TopicMessage pubMsg;
        pubMsg = new TopicMessage(
            ("Hello RocketMQ " + i).getBytes(),
            "TAG"
        );
        TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
        System.out.println("Send mq message success. MsgId is: " +
pubResultMsg.getMessageId());
    }
} catch (Throwable e) {
    System.out.println("Send mq message failed.");
    e.printStackTrace();
}
```

参数	说明
TAG	设置消息的 TAG。

步骤3：消费消息

创建消费者

```
    // 获取Client
    MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

    // 获取Topic的Consumer
    MQConsumer consumer = mqClient.getConsumer(instanceId, topicName, groupName,
"TAG");
```

参数	说明
endpoint	接入地址，在基本信息页面获取。
accessKey	角色 AccessKey，在集群权限页面获取。
secretKey	角色 SecretKey，在集群权限页面获取。

instanceId	集群 ID。
topicName	主题名称，在 Topic 页面获取。
groupName	消费组名称，在 Group 页面获取。

订阅消息

```
do {
    List<Message> messages = null;

    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等待，如果有消息可以消费则立即返回
        // 如果对消费延迟比较敏感，强烈建议使用多线程并发拉取消息
        messages = consumer.consumeMessage(
                Integer.parseInt(batchSize),
                Integer.parseInt(waitSeconds)
        );
    } catch (Throwable e) {
        e.printStackTrace();
    }
    if (messages == null || messages.isEmpty()) {
        System.out.println(Thread.currentThread().getName() + ": no new message,
continue!");
        continue;
    }

    for (Message message : messages) {
        System.out.println("Receive message: " + message);
    }

    {
        List<String> handles = new ArrayList<String>();
        for (Message message : messages) {
            handles.add(message.getReceiptHandle());
        }

        try {
            consumer.ackMessage(handles);
        } catch (Throwable e) {
            if (e instanceof AckMessageException) {
                AckMessageException errors = (AckMessageException) e;
                System.out.println("Ack message fail, requestId is:" +
errors.getRequestId() + ", fail handles:");
                if (errors.getErrorMessages() != null) {
                    for (String errorHandle :errors.getErrorMessages().keySet()) {
                        System.out.println("Handle:" + errorHandle + ", ErrorCode:" +
errors.getErrorMessages().get(errorHandle).getErrorCode()
```

```
                                    + ", ErrorMsg:" +
errors.getErrorMessages().get(errorHandle).getErrorMessage());
                        }
                    }
                    continue;
                }
                e.printStackTrace();
            }
        }
    } while (true);
```

参数	说明
batchSize	一次拉取的消息条数，支持最多16条。
waitSeconds	一次拉取的轮询等待时间，支持最长30秒。

发送与接收顺序消息

最近更新时间：2024-10-10 15:05:51

TDMQ RocketMQ 5.x 版兼容了社区版 HTTP SDK 的接入，如果您此前使用的客户端使用了社区版 HTTP SDK，您在切换到 TDMQ RocketMQ 5.x 版后，您无需在客户端进行任何代码改造。

操作场景

如果当前您已使用了 HTTP 协议进行消息的收发，在您的客户端引入开源 HTTP SDK 后，TDMQ RocketMQ 5.x 版支持用户通过内网或公网使用 HTTP 协议接入。
本文以调用 Java SDK 为例介绍通过 HTTP SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **注意：**
> 暂不支持通过使用 HTTP 协议实现事务消息。

前提条件

- 完成资源创建与准备 。
- 安装1.8或以上版本 JDK 。
- 安装2.5或以上版本 Maven 。
- 通过 Maven 方式引入依赖，在 pom.xml 文件中添加对应语言的 SDK 依赖。

重试机制

HTTP 采用固定重试间隔的机制：

重试间隔	最大重试次数
5分钟	可通过修改消费组配置实现自定义最大重试次数，默认 16 次。

> **说明：**
> - 客户端在重试间隔内 ACK 这条消息，表示消费成功，不会重试。
> - 重试间隔到期后客户端仍未 ACK，客户端会重新消费到这条消息。
> - 每次消费的消息句柄只在重试间隔内有效，过期无效。

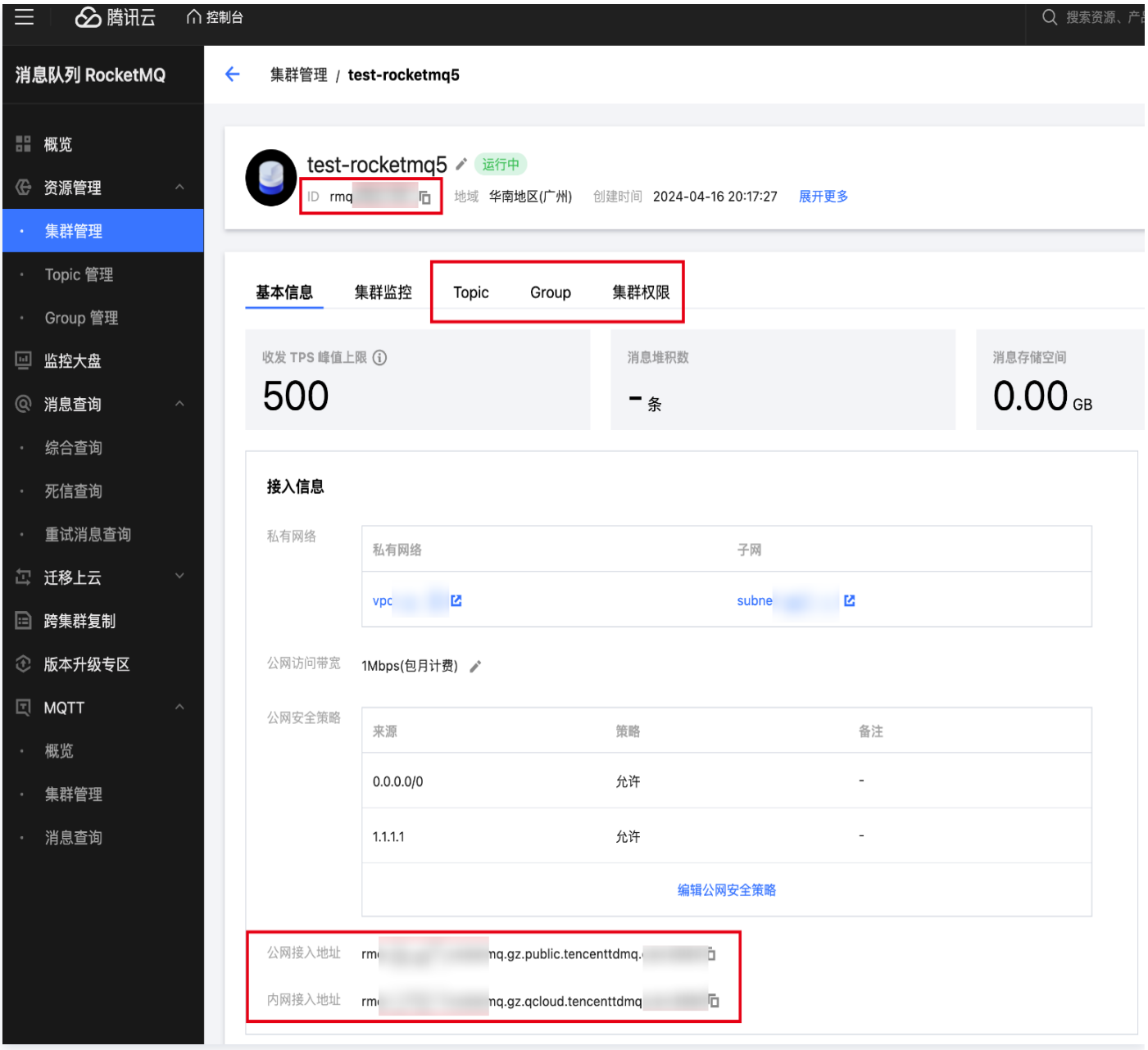
操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入社区版 HTTP SDK 依赖。

步骤2：获取参数

1. 登录消息队列 RocketMQ 控制台，选择集群。
2. 复制集群 ID、接入地址等参数。



步骤2：生产消息

创建消息生产者

```
// 获取Client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

// 获取Topic的Producer
MQProducer producer = mqClient.getProducer(instanceId, topicName);
```

参数	说明

endpoint	接入地址，在基本信息页面获取。
accessKey	角色 AccessKey，在集群权限页面获取。
secretKey	角色 SecretKey，在集群权限页面获取。
instanceId	集群 ID。
topicName	主题名称，在 Topic 页面获取。

发送顺序消息

```
try {
    for (int i = 0; i < 10; i++) {
        TopicMessage pubMsg;
        pubMsg = new TopicMessage(
            ("Hello RocketMQ " + i).getBytes(),
            "TAG"
        );
        // 设置分区顺序消息的 ShardingKey
        pubMsg.setShardingKey(i % 3);
        TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
        System.out.println("Send mq message success. MsgId is: " +
pubResultMsg.getMessageId());
    }
} catch (Throwable e) {
    System.out.println("Send mq message failed.");
    e.printStackTrace();
}
```

参数	说明
TAG	设置消息的 TAG。
ShardingKey	顺序消息的分区字段，相同 ShardingKey 的消息会发送到同一个分区。

步骤3：消费消息

创建消费者

```
    // 获取Client
    MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

    // 获取Topic的Consumer
    MQConsumer consumer = mqClient.getConsumer(instanceId, topicName, groupName,
"TAG");
```

参数	说明
endpoint	接入地址，在基本信息页面获取。

accessKey	角色 AccessKey，在集群权限页面获取。
secretKey	角色 SecretKey，在集群权限页面获取。
instanceId	集群 ID。
topicName	主题名称，在 Topic 页面获取。
groupName	消费组名称，在 Group 页面获取。

订阅消息

```
do {
    List<Message> messages = null;

    try {
        // 长轮询顺序消费消息， 拿到的消息可能是多个分区的， 一个分区的内的消息一定是顺序的
        // 对于顺序消费，如果一个分区内的消息只要有没有被确认消费成功的，则对于这个分区下次还会消费到
相同的消息
        // 对于一个分区，只有所有消息确认消费成功才能消费下一批消息
        // 如果对消费延迟比较敏感，强烈建议使用多线程并发拉取消息
        messages = consumer.consumeMessageOrderly(
                Integer.parseInt(batchSize),
                Integer.parseInt(waitSeconds)
        );
    } catch (Throwable e) {
        e.printStackTrace();
    }
    if (messages == null || messages.isEmpty()) {
        System.out.println(Thread.currentThread().getName() + ": no new message,
continue!");
        continue;
    }

    for (Message message : messages) {
        System.out.println("Receive message: " + message);
    }

    {
        List<String> handles = new ArrayList<String>();
        for (Message message : messages) {
            handles.add(message.getReceiptHandle());
        }

        try {
            consumer.ackMessage(handles);
        } catch (Throwable e) {
            if (e instanceof AckMessageException) {
                AckMessageException errors = (AckMessageException) e;
                System.out.println("Ack message fail, requestId is:" +
errors.getRequestId() + ", fail handles:");
```

```
                if (errors.getErrorMessages() != null) {
                    for (String errorHandle :errors.getErrorMessages().keySet()) {
                        System.out.println("Handle:" + errorHandle + ", ErrorCode:" +
errors.getErrorMessages().get(errorHandle).getErrorCode()
                                + ", ErrorMsg:" +
errors.getErrorMessages().get(errorHandle).getErrorMessage());
                    }
                }
                continue;
            }
            e.printStackTrace();
        }
    }
} while (true);
```

参数	说明
batchSize	一次拉取的消息条数，支持最多16条。
waitSeconds	一次拉取的轮询等待时间，支持最长30秒。

PHP SDK
发送与接收普通消息

最近更新时间：2024-10-14 15:05:51

操作场景

TDMQ RocketMQ 5.x 版支持用户通过内网或公网使用 HTTP 协议接入，并兼容社区的多语言 HTTP SDK 。
本文以调用 PHP SDK 为例介绍通过 HTTP SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

> **注意：**
> 暂不支持通过使用 HTTP 协议实现事务消息。

前提条件

- 完成 资源创建与准备 。
- 安装 PHP。
- 更多示例可以参见开源社区的 Demo 示例 。

重试机制

HTTP 采用固定重试间隔的机制：

重试间隔	最大重试次数
5分钟	可通过修改消费组配置实现自定义最大重试次数，默认 16 次。

> **说明：**
> - 客户端在重试间隔内 ACK 这条消息，表示消费成功，不会重试。
> - 重试间隔到期后客户端仍未 ACK，客户端会重新消费到这条消息。
> - 每次消费的消息句柄只在重试间隔内有效，过期无效。

操作步骤

步骤1：安装 PHP 依赖库

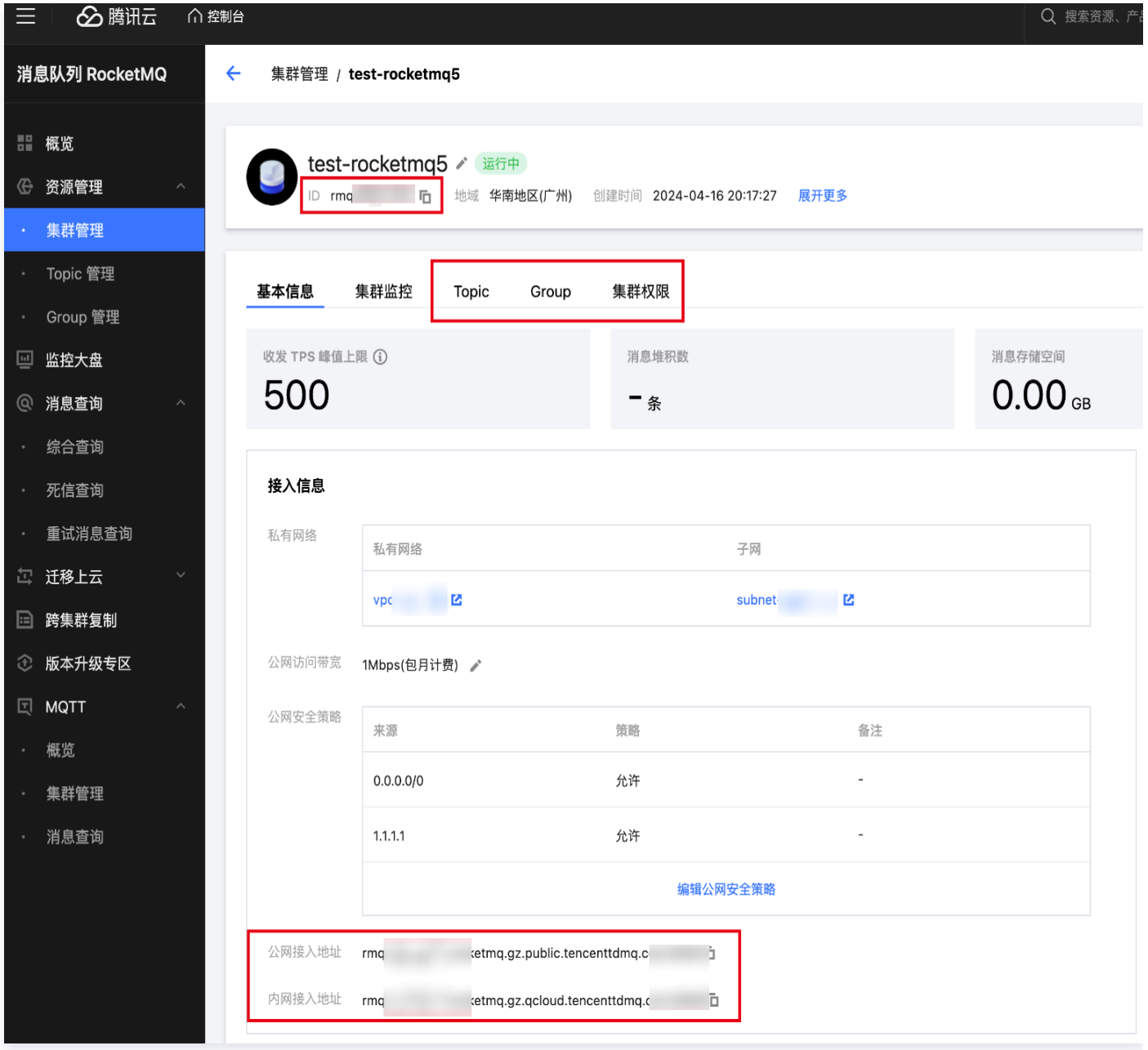
在 PHP 项目中引入相关依赖：

```
{
  "require": {
     "aliyunmq/mq-http-sdk": ">=1.0.4"
  }
}
```

步骤2：获取参数

1. 登录消息队列 RocketMQ 控制台，选择集群。

2. 复制集群 ID、接入地址等参数。



步骤3：生产消息

创建消息生产者

```
private $client;
private $producer;

public function __construct()
 {
     // 获取 client
     $this->client = new MQClient(
         endpoint,
```

```
            accessKey,
            secretKey
        );

        $topic = topicName;
        $instanceId = instanceId;

        // 获取 producer
        $this->producer = $this->client->getProducer($instanceId, $topic);
    }
```

参数	说明
endpoint	接入地址，在基本信息页面获取。
accessKey	角色 AccessKey，在集群权限页面获取。
secretKey	角色 SecretKey，在集群权限页面获取。
instanceId	集群 ID。
topicName	主题名称，在 Topic 页面获取。

发送消息

```
public function run()
    {
        try
        {
            for ($i=0; $i<8; $i++)
            {
                $publishMessage = new TopicMessage(
                    "hello mq!"
                );
                // 设置属性
                $publishMessage->putProperty("a", $i);
                $result = $this->producer->publishMessage($publishMessage);
                print "Send success. msgId is:" . $result->getMessageId() . ", bodyMD5
is:" . $result->getMessageBodyMD5() . "\n";
            }
        } catch (\Exception $e) {
            print_r($e->getMessage() . "\n");
        }
    }
```

步骤3：消费消息

创建消费者

```
    private $client;
```

```
    private $consumer;

    public function __construct()
     {
         // 获取 client
         $this->client = new MQClient(
             endpoint,
             accessKey,
             secretKey
         );

         $topic = topicName;
         $groupId = groupName;
         $instanceId = instanceId;

         // 获取consumer
         $this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId);
     }
```

参数	说明
endpoint	接入地址，在基本信息页面获取。
accessKey	角色 AccessKey，在集群权限页面获取。
secretKey	角色 SecretKey，在集群权限页面获取。
instanceId	集群 ID。
topicName	主题名称，在 Topic 页面获取。
groupName	消费组名称，在 Group 页面获取。

订阅消息

```
public function run()
    {
        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等待，如果有消息可以消费则立即返回
                // 如果对消费延迟比较敏感，强烈建议并发拉取消息
                $messages = $this->consumer->consumeMessage(
                    batchSize,
                    waitSeconds
                );
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\MessageNotExistException) {
                    printf("No message, contine long polling!RequestId:%s\n", $e-
>getRequestId());
                    continue;
```

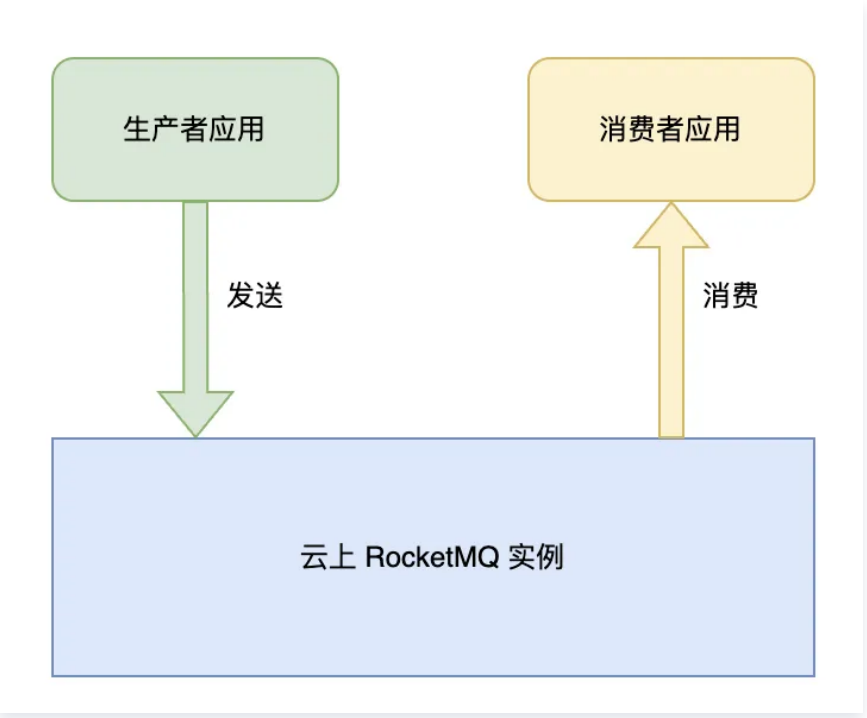
```
                }

                print_r($e->getMessage() . "\n");

                sleep(1);
                continue;
            }

            print "consume finish, messages:\n";

            $receiptHandles = array();
            foreach ($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());
                // 处理业务
            }

            print_r($receiptHandles);
            try {
                $this->consumer->ackMessage($receiptHandles);
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\AckMessageException) {
                    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";
        }
    }
```

参数	说明
batchSize	一次拉取的消息条数，支持最多16条。
waitSeconds	一次拉取的轮询等待时间，支持最长30秒。

RocketMQ 性能压测和容量评估

最近更新时间：2024-10-09 15:34:01

性能测试和容量评估是保证系统整体稳定性的一个基础保证手段，但是如何实施一场有效压测并不是一个简单的事情，我们建议采用以下压测方案，进行压测前，首先应该结合业务场景明确本次压测的目的，然后确定压测具体方案，包括压测对象，压测场景模拟，工具选择以及应该关注哪些指标等，最后分析压测是否达到预期和明确购买合适的规格。



明确压测目的

如上图所示，针对 RocketMQ 的业务场景，一般关注压测发送消息还是压测消费。
如果压测发送消息，主要关注发送速率，耗时和成功率，以及触发峰值限流后应用的表现。
如果压测消费消息，主要关注消费速率，耗时和成功率，失败后的重试策略，以及堆积消息延迟对业务的影响。

分析压测对象

压测发送消息的场景，主要压测对象为 RocketMQ 实例，需要关注 RocketMQ 实例的发送耗时和成功率，以 RocketMQ 5.x 实例为例，实例默认都会开启分布式限流，防止涌入过大流量导致集群被压垮，所以关注限流对业务的影响。
压测消费消息的场景，主要压测对象为下游的消费者应用，需要关注业务下游的消费消息的能力，主要指标是消费处理耗时和消费并发线程数，是否消费超时，是否有异常导致的重试，和是否产生消息堆积。

模拟压测场景

针对发送消息的场景，通常两种方式，第一种可以使用 Apache RocketMQ 开源代码自带的压测脚本来制造压测流量，第二种也可以也入口录制或模拟业务流量，通过业务逻辑代码使用生产者应用进行全链路压测。
通常，我们建议先通过开源代码自带的压测脚本来快速的摸底进行压测，快速拿到 RocketMQ 实例的一些基准指标，做到心里有底，确保RocketMQ 实例本身是达标的，然后再结合业务模型，通过模拟压测业务流量，设置合理的并发数，确保最终在压测结果可以满足业务的需求。
针对消费消息的场景，通常也有两种方式，第一种可以使用 Apache RocketMQ 开源代码自带的压测脚本，订阅压测的 topic，默认收到消息后立即确认消息，来确认 RocketMQ 实例提供的消费能力是足够的。第二种也是通过全链路的压测，由上游消息发送方发送符合消费方业务代码要求的消息格式，确保消费业务代码可以被压测覆盖到，甚至将压测流量进一步传导到下游。

分析压测指标

- 发送指标

  关注发送速率，发送耗时和发送成功率，以及是否触发限流，以及限流后对业务的影响或者消息重试策略。
- 消费指标

  关注消费速率，消费业务耗时，消费延迟和消费成功率，以及消息堆积和延迟对业务的影响。

常见问题分析

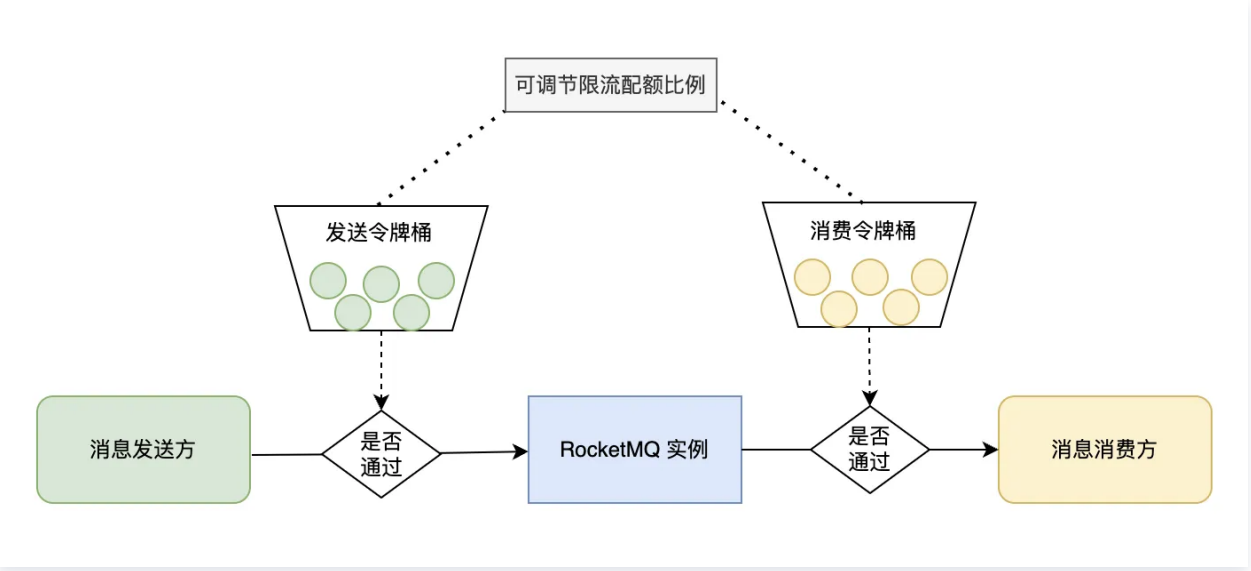
1.如何解决发送速率压不上去?

决定发送速率的核心因素有两个，一个是发送耗时，一个是发送并发度，假如平均发送耗时是 5ms，并发度是 1 的话，则发送速率是 200 TPS；所以如果遇到压测目标不达标的时候，首先确认一下发送耗时，例如是不是网络走了公网，或者经过了代理，到时发送耗时偏高； 如果发送耗时符合预期，则重点排查并发度是否满足，发送方的并行执行的线程数量是否足够，发送方的节点的负载是否正常，更上层是否有锁等因素影响。

2.如何高效模拟流量的压测下游业务方?

经常为了达到好的压测效果，需要压测流量尽可能的接近真实的业务，为了模拟流量，除了全链路压测以外，还可以通过重置消费位点的方式，重放历史消息，来给下游业务方高效的制造流量，这样就不需要上游业务来反复的制造发送流量。

3.如何分析触发限流的原因?



因为 RocketMQ 5.x 实例默认开启了限流，如果压测触发了限流，重点分析以下几个原因：

1. 存在“微突发”的场景，例如我们的监控是分钟级粒度的监控，可能所有的流量集中在前 1 秒发起，实际我们限流令牌窗口是 10秒更新一次窗口，这样就会导致分钟级监控看没有超限流值，实际上10 秒粒度却会被限流。
2. 消息体过大，因为限流是按照 4KB 一条消息折算，例如 100KB 的消息，将会折算为 25 条消息，进行限流，所以限流值和生产消息条数并不是一对一的。
3. 限流比例调整不合适，我们提供了可以调整发送和消费限流配额比例，默认为 5:5 ，最多可以调整为 2:8 或者 8:2，所以如果触发了限流，也要检查一下配置的限流配额比例是否合理。

POP 消费模式的说明

最近更新时间：2024-09-05 17:18:11

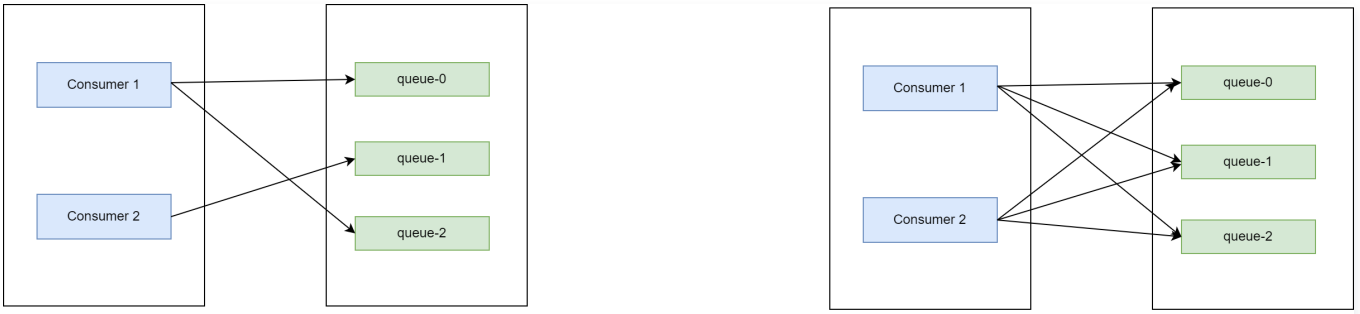
问题背景

RocketMQ 以其高性能，低延迟和抗积压的特性被不少客户和开发者熟知，但是在 RocketMQ 4.x 客户端 SDK 的使用中，不少客户会反馈消费者客户端在实际消费消息过程中，4.x 的客户端（例如常用的 Push Comsumer）在消费的过程中遇到一些问题：

- SDK 承担了太多功能，例如拉消息，负载均衡，消息位点管理和新增客户端时的 Rebalance 等等，这点对多语言开发开发者不友好。
- 队列独占的负载均衡策略容易导致消费瓶颈：Broker 上的每个队列只能分配到相同 Group 的一台消费者客户端上。因此当队列数固定时，单纯增加消费者客户端的数量并不能提升消费性能。假设某个 Topic 共有10个队列，Group 最多有 10 个客户端进行消费（即最多每个客户端消费一个队列）。在业务高峰期，即使客户想增加新的客户端去消费消息，新上线的第11个客户端也无法消费消息。
- 单个客户端异常导致堆积。假设单个客户端因为异常 “hang 机” 时，由于和服务端的心跳没有断开，因此该客户端会被分配到队列进行消费，而此时因为客户端异常实际并不能消费机器，导致异常堆积产生，且因为上一条的原因，单纯加客户端数量并不能解决问题。

解决方案

鉴于以上原因，5.x 推出了 POP 消费模式。
POP 模式下，消费位点由服务端进行管理，因此多个客户端可以消费同一个队列。使用 POP 消费模式的客户端，每个客户端都会从所有的队列去拉消息，因此解决了上述的单个客户端异常和消费瓶颈的问题。
同时，服务端维护消费信息，使得客户端 SDK 更加轻量，方便进行多语言移植。



代码示例

那么我们如何使用 POP 消费模式呢?
需要使用 5.x 的 gRPC SDK，引入相关依赖如下：

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.6</version>
        </dependency>
</dependencies>
```

同时参考开源社区的 DEMO 如下（以 Java 代码为例）：

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.rocketmq.client.java.example;

import java.time.Duration;
import java.util.Collections;
import java.util.List;
import org.apache.rocketmq.client.apis.ClientConfiguration;
import org.apache.rocketmq.client.apis.ClientException;
import org.apache.rocketmq.client.apis.ClientServiceProvider;
import org.apache.rocketmq.client.apis.SessionCredentialsProvider;
import org.apache.rocketmq.client.apis.StaticSessionCredentialsProvider;
import org.apache.rocketmq.client.apis.consumer.FilterExpression;
import org.apache.rocketmq.client.apis.consumer.FilterExpressionType;
import org.apache.rocketmq.client.apis.consumer.SimpleConsumer;
import org.apache.rocketmq.client.apis.message.MessageId;
import org.apache.rocketmq.client.apis.message.MessageView;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SimpleConsumerExample {
    private static final Logger log =
LoggerFactory.getLogger(SimpleConsumerExample.class);

    private SimpleConsumerExample() {
    }

    @SuppressWarnings({"resource", "InfiniteLoopStatement"})
    public static void main(String[] args) throws ClientException {
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        // Credential provider is optional for client configuration.
        String accessKey = "用户ak";
```

```
        String secretKey = "用户sk";
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        String endpoints = "腾讯云页面接入点";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            // On some Windows platforms, you may encounter SSL compatibility issues.
Try turning off the SSL option in
            // client configuration to solve the problem please if SSL is not
essential.
            // .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String consumerGroup = "消费组";
        // 默认消费时间，30s，也就是说，对于拉到的消息，30s内如果消费没完成，该条消息会被别的客户端
重新拉到，
        // 需要用户根据自己的场景配置
        Duration awaitDuration = Duration.ofSeconds(30);
        String tag = "*";
        String topic = "topic名字";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        // In most case, you don't need to create too many consumers, singleton
pattern is recommended.
        SimpleConsumer consumer = provider.newSimpleConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // set await duration for long-polling.
            .setAwaitDuration(awaitDuration)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .build();
        // Max message num for each long polling.
        int maxMessageNum = 16;
        // Set message invisible duration after it is received.
        Duration invisibleDuration = Duration.ofSeconds(15);
        // Receive message, multi-threading is more recommended.
        do {
            final List<MessageView> messages = consumer.receive(maxMessageNum,
invisibleDuration);
            log.info("Received {} message(s)", messages.size());
            for (MessageView message : messages) {
                final MessageId messageId = message.getMessageId();
                try {
                    consumer.ack(message);
                    log.info("Message is acknowledged successfully, messageId={}",
messageId);
                } catch (Throwable t) {
```

```
                    log.error("Message is failed to be acknowledged, messageId={}",
messageId, t);
                }
            }
        } while (true);
        // Close the simple consumer when you don't need it anymore.
        // You could close it manually or add this into the JVM shutdown hook.
        // consumer.close();
    }
}
```

在这种情况下，同一个消费组内，一个消费者将不会再和队列一一绑定，也能最大程度避免之前4.x中单个消费者阻塞导致队列堆积的问题。

RocketMQ 限流说明与实践教程

最近更新时间：2025-02-20 16:38:42

概述

腾讯云消息队列 RocketMQ 版为各类大规模、低延时、高可用性要求的在线业务提供消息服务，客户端通过 SDK 与 RocketMQ 集群建立长连接并进行消息收发，同时消耗集群机器节点的计算、存储、网络带宽等资源。为了提供高质量的消息服务，我们需要控制集群在高并发、大流量情况下的负载水位，以保障系统的稳定性与可靠性。因此，服务端会**根据集群规格限制客户端每秒能够发送和消费的最大折算消息条数**（Transaction Per Second/TPS），具体计算规则可参见 计费概述 - 计算规格 。为了兼具隔离性和灵活性，发送消息与消费消息的 TPS 配额不共享，同时支持自定义配额比例（默认配额比例为1：1 ）。

限流行为说明

腾讯云消息队列 RocketMQ 版采用基于快速失败 (Fail-Fast) 限流策略，即**当客户端请求速率达到上限后，服务端会立即响应错误**。通常在线业务都对响应时间敏感，快速失败可以让客户端感知限流事件并及时介入处理，避免业务消息端到端耗时恶化。
以1000TPS 规格的基础集群为例，假设配额收发 TPS 比例为1:1，相关的触发限流行为说明如下：

说明	发送消息限流	消费消息限流
触发限流情景	所有连接该集群的发送客户端每秒最多可发送折算消息的总和为 500 条，发送速率达到限制后，超限的发送请求会失败。	所有连接该集群的消费客户端每秒最多可消费折算消息的总和为 500 条，消费速率达到限制后，消息的消费延迟会增加。
触发限流时 SDK 日志关键词	Rate of message sending reaches limit, please take a control or upgrade the resource specification。	Rate of message receiving reaches limit, please take a control or upgrade the resource specification。
触发限流时 SDK 重试机制	不同协议的 SDK 处理有差异： • 5.x SDK 会根据指数退避策略进行重试发送，最大重试次数可在初始化 Producer 时自定义，默认值为 2 次；达到最大重试次数仍未成功的发送请求会抛出异常。 • 4.x SDK 直接抛出异常，不会进行重试。	SDK 拉消息线程会自动退避重试。

客户端实践教程

规划集群

腾讯云消息队列 RocketMQ 版集群限流的目的是保障服务稳定可靠，防止在集群高负载时出现服务响应时间变长、请求成功率下降等问题，从而避免业务受损。因此，在您接入腾讯云消息队列 RocketMQ 版时，合理规划集群非常重要，建议您：

- 依据当前规模和未来趋势预测来充分评估业务 TPS， 如果业务流量具有波动特性，应以峰值 TPS 为准。此外，评估时建议您预留一部分 TPS 配额（例如 30%）来应对可能出现的突发流量。
- 对稳定性要求较高的业务，建议您使用多套 RocketMQ 集群加强隔离性。例如，将核心链路（如交易系统）与非核心链路（如日志系统）隔离，以及生产环境与开发测试环境进行隔离等。

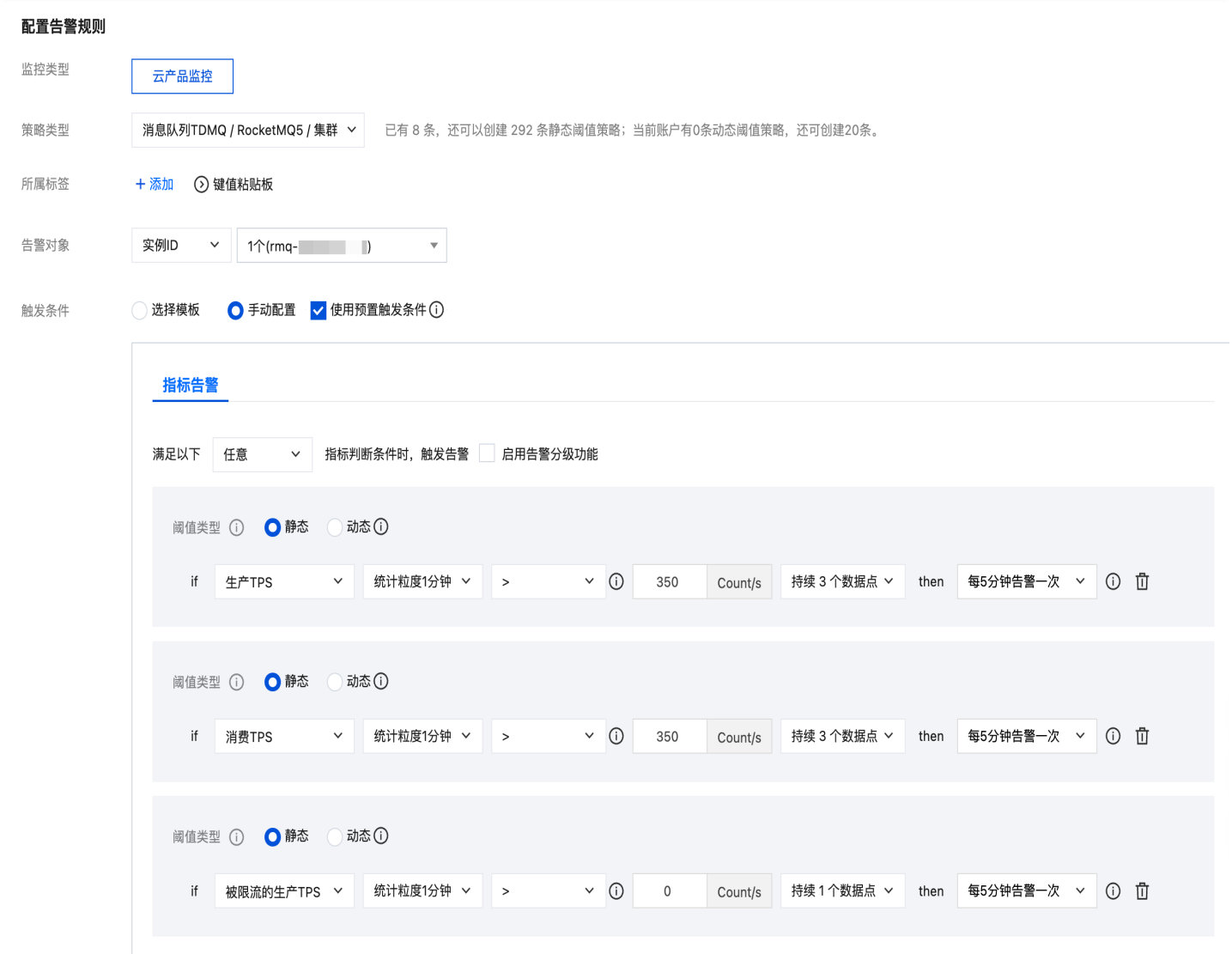
监控负载

您可以利用腾讯云消息队列 RocketMQ 版控制台的监控告警能力实现对集群负载的实时观测，提前发现 TPS 水位风险并及时操作升配，保证资源充足，避免触发限流。具体操作可参见 监控告警，告警策略建议如下：

- 发送和消费 TPS 水位超过容量的 70% 时触发告警，提醒进行升配评估。
- 出现发送限流时触发告警，警告业务发送消息可能失败风险。

示例

以 1000TPS 规格的基础集群为例，TPS 告警策略如下：



TPS 告警配置

代码异常处理

业务代码通过 RocketMQ SDK 发送消息时，需要捕获包括限流错误在内的异常，并保存必要的上下文信息，以便人工介入恢复业务。不同协议的 SDK 重试机制有差异，相关处理示例代码如下：

- 4.x SDK 不会对限流错误进行自动重试，因此业务代码需要捕获异常并进行处理，示例代码如下：

```
import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.exception.MQBrokerException;
import org.apache.rocketmq.client.exception.MQClientException;
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.common.message.MessageClientIDSetter;
import org.apache.rocketmq.logging.InternalLogger;
import org.apache.rocketmq.logging.InternalLoggerFactory;

public class ProducerExample {
    private static final InternalLogger log =
InternalLoggerFactory.getLogger(ProducerExample.class);

    public static void main(
        String[] args) throws MQClientException, InterruptedException,
UnsupportedEncodingException {

        String nameserver = "Your_Nameserver";
        String accessKey = "Your_Access_Key";
        String secretKey = "Your_Secret_Key";
        String topicName = "Your_Topic_Name";
        String producerGroupName = "Your_Producer_Group_Name";

        // 实例化消息生产者 Producer
        DefaultMQProducer producer = new DefaultMQProducer(
            producerGroupName, // 生产者组名字
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) //
ACL权限，accessKey 和 secretKey 可以在控制台集群权限页面获取
        );

        // 设置 Nameserver 地址，可以在控制台集群基本信息页面获取
        producer.setNamesrvAddr(nameserver);

        // 启动 Producer 实例
        producer.start();

        // 创建消息实例，设置 topic 和消息内容
        Message message = new Message(topicName,
"Your_Biz_Body".getBytes(StandardCharsets.UTF_8));

        // 最大尝试发送次数，请根据业务情况设置
        final int maxAttempts = 3;
```

```
        // 重试间隔时间，请根据业务情况设置
        final int retryIntervalMillis = 200;

        // 发送消息
        int attempt = 0;
        do {
            try {
                SendResult sendResult = producer.send(message);
                log.info("Send message successfully, {}", sendResult);
                break;
            } catch (Throwable t) {
                attempt++;
                if (attempt >= maxAttempts) {
                    // 达到最大次数
                    log.warn("Failed to send message finally, run out of attempt
times, attempt={}, maxAttempts={}, msgId={}",
                        attempt, maxAttempts,
MessageClientIDSetter.getUniqID(message), t);
                    // 记录发送失败的消息（或记录到其他业务系统，比如数据库等）
                    log.warn(message.toString());
                    break;
                }
                int waitMillis;
                if (t instanceof MQBrokerException && ((MQBrokerException)
t).getResponseCode() == 215 /* FLOW_CONTROL */) {
                    // 限流异常，采用退避重试
                    waitMillis = (int) Math.pow(2, attempt - 1) *
retryIntervalMillis; // 重试间隔：200ms, 400ms, ......
                } else {
                    // 其他异常
                    waitMillis = retryIntervalMillis;
                }
                log.warn("Failed to send message, will retry after {}ms, attempt={},
maxAttempts={}, msgId={}",
                    waitMillis, attempt, maxAttempts,
MessageClientIDSetter.getUniqID(message), t);
                try {
                    Thread.sleep(waitMillis);
                } catch (InterruptedException ignore) {
                }
            }
        }
        while (true);

        producer.shutdown();
    }
}
```

- 5.x SDK 会对发送异常进行自动重试，业务代码可以自定义最大重试次数，示例代码如下：

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import org.apache.rocketmq.client.apis.ClientConfiguration;
import org.apache.rocketmq.client.apis.ClientException;
import org.apache.rocketmq.client.apis.ClientServiceProvider;
import org.apache.rocketmq.client.apis.SessionCredentialsProvider;
import org.apache.rocketmq.client.apis.StaticSessionCredentialsProvider;
import org.apache.rocketmq.client.apis.message.Message;
import org.apache.rocketmq.client.apis.producer.Producer;
import org.apache.rocketmq.client.apis.producer.SendReceipt;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class ProducerExample {
    private static final Logger log =
LoggerFactory.getLogger(ProducerExample.class);

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

        String nameserver = "Your_Nameserver";
        String accessKey = "Your_Access_Key";
        String secretKey = "Your_Secret_Key";
        String topicName = "Your_Topic_Name";

        // ACL权限, accessKey 和 secretKey 可以在控制台集群权限页面获取
        SessionCredentialsProvider sessionCredentialsProvider = new
StaticSessionCredentialsProvider(accessKey, secretKey);

        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(nameserver) // 设置 NameServer 地址, 可以在控制台集群基本信息页
面获取
            .setCredentialProvider(sessionCredentialsProvider)
            .build();

        // 启动 Producer 实例
        ClientServiceProvider provider = ClientServiceProvider.loadService();
        Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            .setTopics(topicName) // 预声明消息发送的 topic, 建议设置
            .setMaxAttempts(3) // 最大尝试发送次数, 请根据业务情况设置
            .build();

        // 创建消息实例, 设置 topic 和消息内容
        byte[] body = "Your_Biz_Body".getBytes(StandardCharsets.UTF_8);
        final Message message = provider.newMessageBuilder()
            .setTopic(topicName)
            .setBody(body)
            .build();

        try {
```

```
            final SendReceipt sendReceipt = producer.send(message);
            log.info("Send message successfully, messageId={}",
sendReceipt.getMessageId());
        } catch (Throwable t) {
            log.warn("Failed to send message", t);
            // 记录发送失败的消息（或记录到其他业务系统，比如数据库等）
            log.warn(message.toString());
        }

        producer.close();
    }
}
```

常见问题

触发限流后会不会丢消息?

发送消息触发限流后服务端不会存储该条消息，客户端需要捕获异常并做降级处理；消费触发限流后会出现消费延迟，但已经发送成功的消息不会丢。

为什么监控页面的 TPS 比消息条数大?

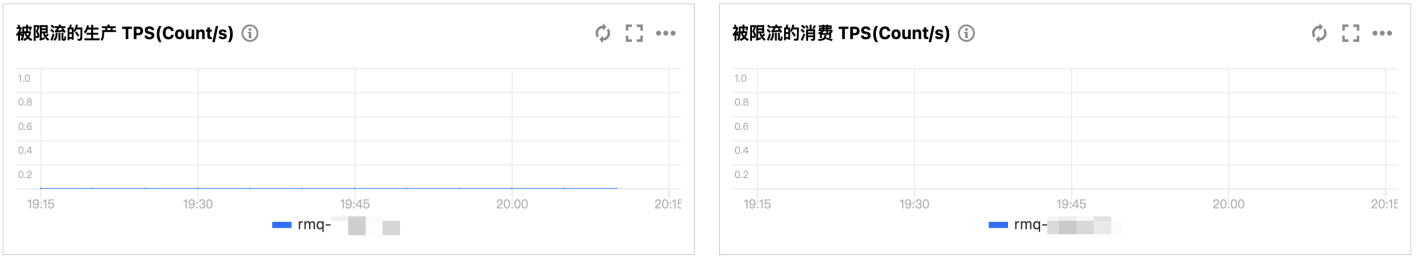
TPS 是折算消息数量，如果业务使用了高级消息（顺序、延迟、事务等）或消息体比较大，那么一条业务消息会被统计为多条折算消息，具体的折算逻辑可参见 计算规格 。此外，消息条数指标统计的是一分钟内的秒级平均值，而 TPS 指标统计的是一分钟内的秒级峰值。

集群偶尔出现短暂的消费被限流，是否有影响?

一般没有影响。在客户端重启、服务端重启、控制台扩容主题队列等操作期间，都有可能因为消费组瞬间堆积而触发短暂的消费限流，通常稳定后很快会恢复。

如何判断集群是否出现了限流?

除了通过识别 SDK 发送接口抛出的异常或 SDK 日志记录的信息外，您还可以查看 腾讯云 RocketMQ 控制台 > 监控大盘 的**被限流的生产 TPS(Count/s)** 和**被限流的消费 TPS(Count/s)**。



客户端风险

最近更新时间：2024-12-10 16:14:22

1.场景 Java的PushConsumer顺序消息特定场景下无法保证

客户端列表

5.0.6及以下在一些特殊场景下可能会遇到，在一些网络异常，超时等的场景下，无法保证顺序。相关问题：[Bug] PushConsumer reentrant pop orderly doesn't work as expected 。

升级客户端操作指南

建议升级到5.0.7，直接升级依赖版本即可。

2.场景 golang 低版本，可能存在路由表被清空

客户端列表

v5.0.1及以下在一些网络超时的场景下，会由于获取不到路由表，进行了清空，相关问题：golang: optimize the logic of obtaining routing tasks 。

升级客户端操作指南

建议升级到v5.1.1-rc1，直接升级依赖版本即可。