消息队列 RocketMQ 版

RocketMQ 4.x

产品文档

腾讯云

文档目录

RocketMQ 4.x
- 产品简介
  - 产品概述
  - 技术架构
  - 基本概念
  - 产品优势
  - 应用场景
  - 使用限制
  - 开源对比
- 购买指南
  - 计费概述
    - 通用集群
    - 专享集群
    - 虚拟集群
  - 产品系列
  - 购买方式
  - 欠费说明
  - 退费说明
  - 账单说明
- 快速入门
  - 快速入门概述
  - 使用 TCP 协议收发消息
    - 资源创建与准备
    - 使用 SDK 收发普通消息
- 操作指南
  - 集群管理
    - 新建集群
    - 升配集群
    - 降配集群
    - 销毁/退还集群
    - 调整公网带宽
  - 命名空间管理
  - Topic 管理
  - Group 管理
  - 角色与鉴权

访问管理 CAM
主账号获取访问授权
子账号获取访问授权
授予子账号访问权限
授予子账号操作级权限
授予子账号资源级权限
授予子账号标签级权限
标签管理
使用标签管理资源
编辑标签
监控告警
集群监控
Group 监控
Topic 监控
配置告警
消息查询
查询消息
查询死信消息
消息轨迹与说明
消息跨集群复制
开发指南
消息类型
普通消息
定时与延时消息
顺序消息
事务消息
消息过滤
消费模式
消息重试
死信队列
SDK 文档
TCP 协议接入
Spring Boot Starter 接入
发送与接收普通消息
发送与接收过滤消息
发送与接收延迟消息
Spring Cloud Stream 接入
Java SDK

发送与接收普通消息
发送与接收延迟消息
发送与接收顺序消息
发送与接收事务消息
发送与接收过滤消息
发送与接收广播消息
C++ SDK
Go SDK
Python SDK
HTTP 协议接入
HTTP 协议接入
Java SDK
发送与接收普通消息
发送与接收顺序消息

RocketMQ 4.x
产品简介
产品概述

最近更新时间：2023-03-01 10:33:25

消息队列 RocketMQ 版（TDMQ for RocketMQ，简称 TDMQ RocketMQ 版）是腾讯云基于 Apache RocketMQ 构建的分布式消息中间件，完全兼容 Apache RocketMQ 的各个组件与概念，支持开源社区版本的客户端零改造接入。消息队列 RocketMQ 版具有低延迟、高性能、高可靠、万亿级消息容量和灵活可扩展等特点，可为分布式应用系统提供异步解耦和削峰填谷的能力，同时也具备互联网应用所需的海量消息堆积、高吞吐、可靠重试等特性。

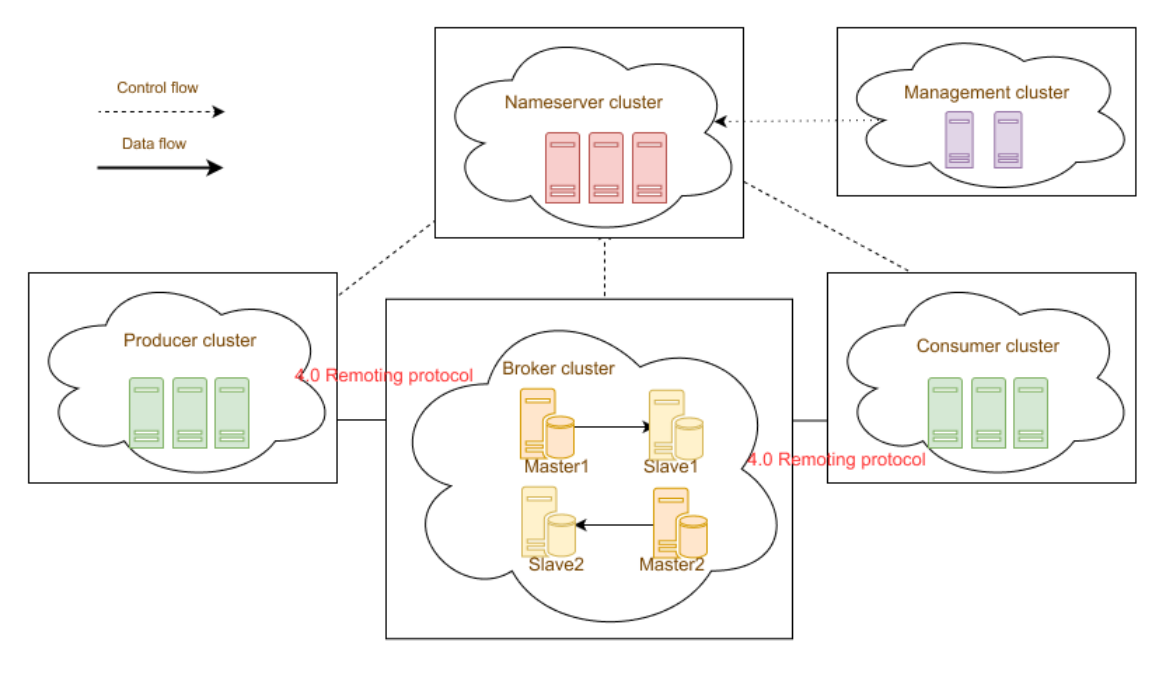
技术架构

最近更新时间：2023-04-14 16:49:58

本文主要介绍消息队列 RocketMQ 版的部署架构，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：



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。

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。

Broker集群：服务端应用，负责接收，存储，投递消息，支持主从多副本模式，从节点可选部署，实际现网公有云上数据高可靠直接依赖云盘三副本。

管控集群： 服务端应用，可视化的管控控制台，负责运维整个集群，例如源数据的收发和管理等。

消息队列 RocketMQ 版相比于自建开源 RocketMQ 所具备的优势请参见开源对比。

基本概念

最近更新时间：2023-10-19 10:38:35

本文列举了 TDMQ RocketMQ 版中常见的概念与定义。

消息（Message）

消息系统所传输信息的物理载体，生产和消费数据的最小单位。生产者将业务数据的负载和拓展属性包装成消息发送到服务端，服务端按照相关语义将消息投递到消费端进行消费。

主题（Topic）

Topic 表示一类消息的集合，每个主题包含若干消息，是 RocketMQ 进行消息订阅的基本单位。

消息标签（MessageTag）

为消息设置的标签，用于将同一个 Topic 下区分不同类型的消息，可以理解为 Topic 是消息的一级分类，Tag 是消息的二级分类。

消息队列（MessageQueue）

存储消息的物理实体，一个 Topic 可以包含多个 Queue，Queue 也叫消息分区，一个 Queue 中的消息只能被一个消费者组中的一个消费者消费，一个 Queue 中的消息不允许同一个消费者组中的多个消费者同时消费。

消息位点（MessageQueueOffset）

消息是按到达 RocketMQ 服务端的先后顺序存储在指定主题的多个队列中，每条消息在队列中都有一个唯一的 Long 类型坐标，这个坐标被定义为消息位点。

消费位点（ConsumerOffset）

一条消息被某个消费者消费完成后不会立即从队列中删除， RocketMQ 会基于每个消费者分组记录消费过的最新一条消息的位点，即消费位点。

消息索引（MessageKey）

消息索引是 RocketMQ 提供的面向消息的索引属性。通过设置的消息索引可以快速查找到对应的消息内容。

生产者（Producer）

生产者是 RocketMQ 系统中用来构建并传输消息到服务端的运行实体。生产者通常被集成在业务系统中，将业务消息按照要求封装成消息并发送至服务端。

消费者（Consumer）

消费者是 RocketMQ 中用来接收并处理消息的运行实体。消费者通常被集成在业务系统中，从服务端获取消息，并将消息转化成业务可理解的信息，供业务逻辑处理。

分组（Group）

可分为生产者组和消费者组：
生产者组：同一类 Producer 的集合，这类 Producer 发送同一类消息且发送逻辑一致。如果发送的是事务消息，且生产者发送后崩溃，则 Broker 服务器会联系同一个生产者组的其他生产者实例以提交或者回溯消费。
消费者组：同一类 Consumer 的集合，这类 Consumer 通常消费同一类消息且消费逻辑一致。消费者组使得在消息消费方面实现了负载均衡和容错。消费者组的消费者实例必须订阅完全相同的 Topic。

消息类型（MessageType）

按照消息传输特性的不同而定义的分类，用于类型管理和安全校验。RocketMQ 支持的消息类型有普通消息、顺序消息、事务消息和定时/延时消息。

普通消息

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。

顺序消息

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。

重试队列

重试队列是一种为了确保消息被正常消费而设计的队列。当某些消息第一次被消费者消费后，没有得到正常的回应，则会进入重试队列，当重试达到一定次数后，停止重试，投递到死信队列中。
由于实际场景中，可能会存在的一些临时短暂的问题（如网络抖动，服务重启等）导致消息无法及时被处理，但短暂时间过后又恢复正常。这种场景下，重试队列的重试机制就可以很好解决此类问题。

死信队列

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。当消息在重试队列中达到一定重试次数后仍未能被正常消费，TDMQ 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。

集群消费

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。适用于每条消息只需要被处理一次的场景。

广播消费

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。适用于每条消息需要被集群下每一个消费者处理的场景。

消息过滤

消费者可以通过订阅指定消息标签（Tag）对消息进行过滤，确保最终只接收被过滤后的消息合集。过滤规则的计算和匹配在 RocketMQ 的服务端完成。

重置消费位点

以时间轴为坐标，在消息持久化存储的时间范围内，重新设置消费者分组对已订阅主题的消费进度，设置完成后消费者将接收设定时间点之后，由生产者发送到 RocketMQ 服务端的消息。

消息轨迹

在一条消息从生产者发出到消费者接收并处理过程中，由各个相关节点的时间、地点等数据汇聚而成的完整链路信息。通过消息轨迹，您能清晰定位消息从生产者发出，经由 RocketMQ 服务端，投递给消费者的完整链路，方便定位排查问题。

消息堆积

生产者已经将消息发送到 RocketMQ 的服务端，但由于消费者的消费能力有限，未能在短时间内将所有消息正确消费掉，此时在服务端保存着未被消费的消息，该状态即消息堆积。一分钟统计一次，如果消息已经过了保留时间（默认3天），那么这部分消息是不会再计算在堆积里面了（因为服务端已经进行了删除）。

产品优势

最近更新时间：2024-01-18 09:45:43

兼容开源

TDMQ RocketMQ 版兼容开源 RocketMQ 4.3.0及以上版本，支持开源 Java、C/C++、Go 等多语言客户端接入。

资源隔离

多层级的资源结构，不仅基于命名空间做了虚拟隔离，也可以在集群维度做物理隔离。支持在命名空间维度为客户端配置权限校验，区分不同环境的客户端，方便灵活。

丰富的消息类型

支持普通消息、顺序消息、延时消息和事务消息等多种消息类型，支持消息重试和死信机制，满足各类业务场景。

高性能

单机最高可支持上万级别的生产消费吞吐，分布式架构，无状态服务，可以横向扩容来增强整个集群的吞吐。

可观测性

控制台提供丰富的监控指标，消息轨迹可视化展示，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。

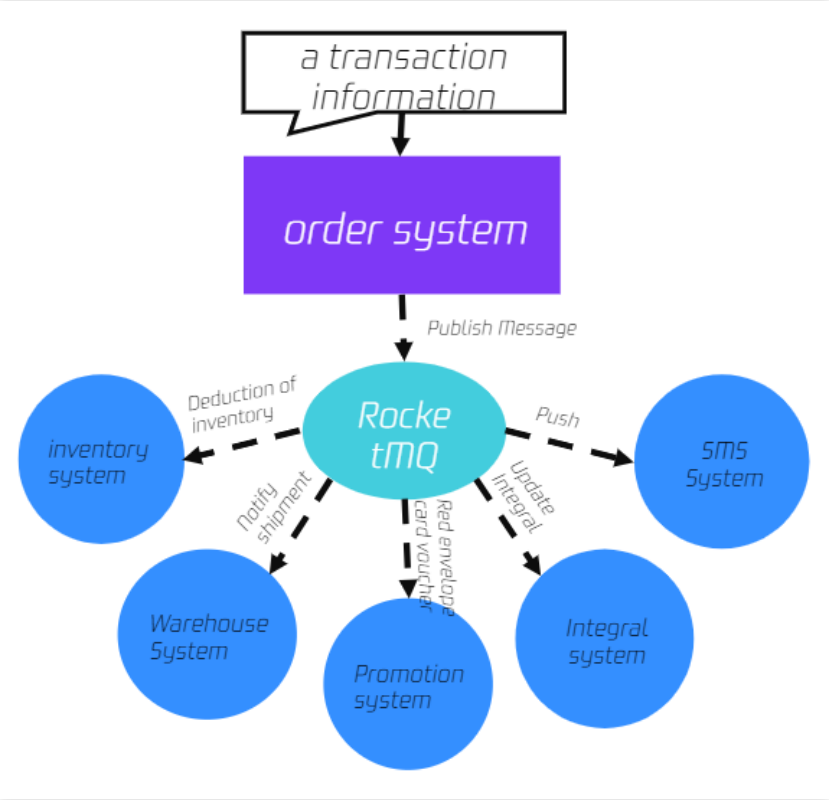
应用场景

最近更新时间：2023-09-12 16:01:48

TDMQ RocketMQ 版是基于 Apache RocketMQ 构建的分布式消息中间件，应用于分布式系统或组件之间的消息通讯，具备海量消息堆积、低延迟、高吞吐、高可靠、事务强一致性等特性，满足异步解耦、削峰填谷、顺序收发、分布式事务一致性、日志同步等场景需求。

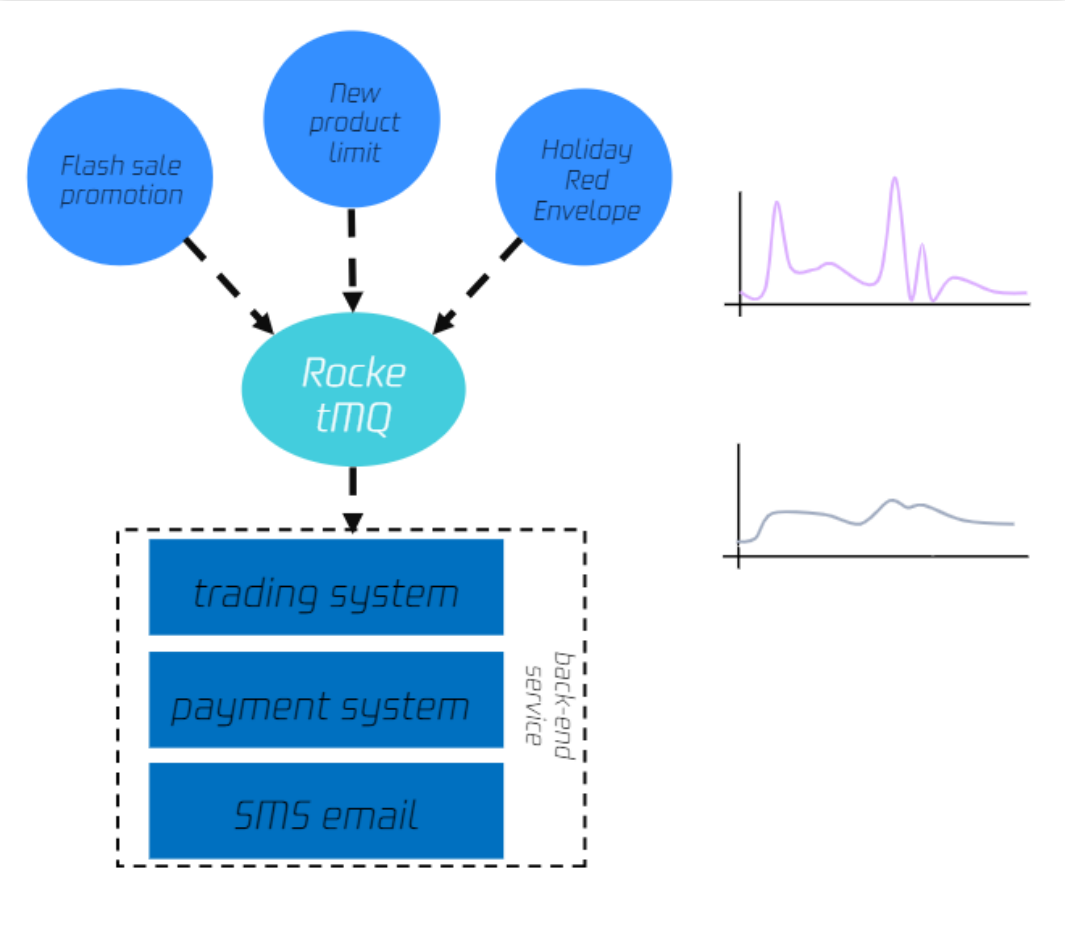
异步解耦

交易引擎作为腾讯计费最核心的系统，每笔交易订单数据需要被几十个下游业务系统关注，包括库存系统、仓储系统、促销系统、积分系统等，多个系统对消息的处理逻辑不一致，单个系统不可能去适配每一个关联业务。此时，TDMQ RocketMQ 版可解除多个业务系统之间的耦合度，减少系统之间影响，提升核心业务响应速度和健壮性。



削峰填谷

企业不定时举办的一些营销活动，新品发布上线，节日抢红包等，往往都会带来临时性的流量洪峰，这对后端的各个应用系统考验是十分巨大的，如果直接采用扩容方式应对又会带来一定的资源浪费。RocketMQ 可以应对突发性的流量洪峰，在峰值时堆积消息，而在峰值过去后下游系统慢慢消费消息，解决上下游处理能力不匹配，提升系统可用性。



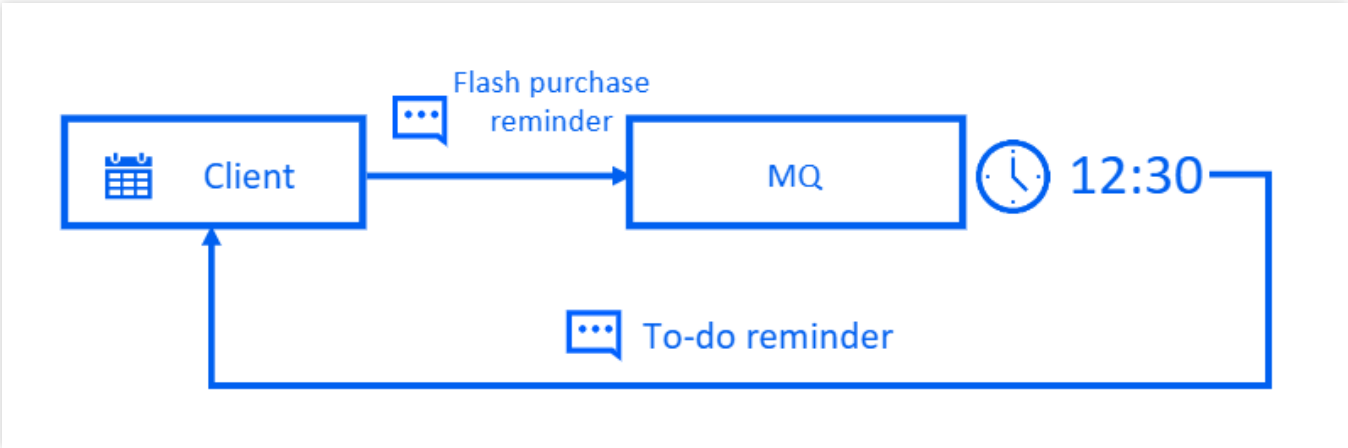
订阅通知

消息队列 RocketMQ 版提供的定时、延迟等能力，满足需要订阅通知的电商场景。

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。

关于定时与延迟消息的详细内容，请参见 定时与延迟消息。



分布式事务一致性

RocketMQ 提供分布式事务消息，使应用之间松耦合，可靠传输与多副本技术能确保消息不丢失，At-Least-Once 特性确保数据最终一致性。

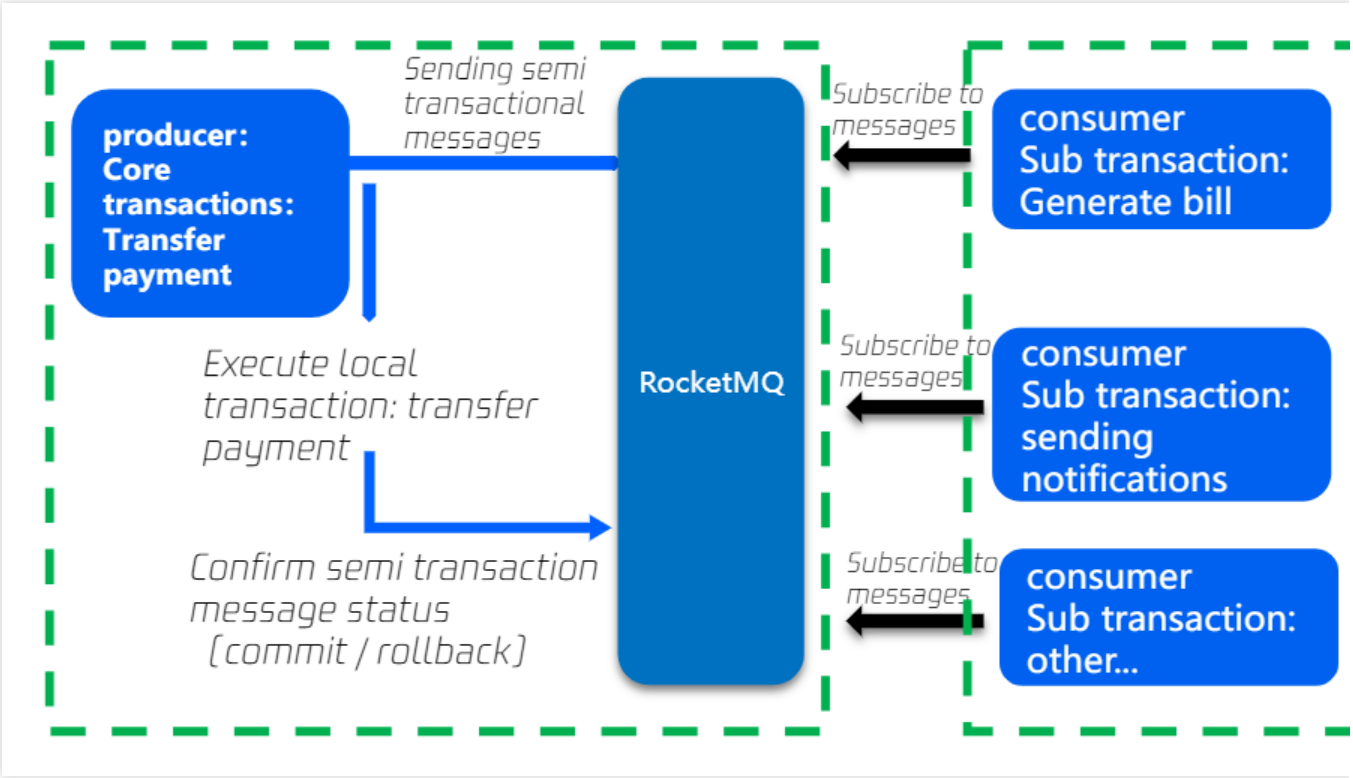
支付系统作为生产者，与消息队列，组成一个事务，保障本地事务和消息发送的一致性。

下游业务系统（账单、通知、其它），作为消费者，并行处理。

消息支持可靠重试，保证数据最终一致性。

使用消息队列 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。计费系统的交易链路通常比较长，出错或者超时的概率比较高，这时会借助 TDMQ 的自动重推和海量堆积能力来实现事务补偿，同时支付 Tips 通知和交易流水推送可以通过 RocketMQ 来实现最终一致性。

关于事务消息的详细内容，请参见 事务消息。



顺序收发

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。顺序消息常用于以下业务场景：

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，比如某些证券交易，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。

关于顺序消息的详细介绍，请参见 顺序消息。

分布式模缓存同步

企业举办打折促销活动时，产品种类繁多并且价格频繁变动，用户访问较多次商品价格查询，缓存服务器网卡满载，影响页面的打开速度。使用 TDMQ RocketMQ 版的广播消费模式，那么这条消息会被所有节点消费一次，相当于把价格信息同步到需要的每台机器上，取代缓存的作用。

使用限制

最近更新时间：2023-09-12 17:53:17

本文列举了 TDMQ RocketMQ 版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	虚拟集群限制	专享集群限制
单地域内集群数量上限	10个	不限制
集群名称长度	3-64个字符	3-64个字符
集群 TPS 上限	4000	4000以上，按不同的节点规格计算
单集群带宽上限（生产+消费）	40Mbps	80Mbps以上，按不同的节点规格计算

命名空间

限制类型	虚拟集群限制	专享集群限制
单集群内命名空间数量上限	10个	10个
命名空间名称长度	3-32个字符	3-32个字符

Topic

限制类型	虚拟集群限制	专享集群限制
单集群内 Topic 数量上限	150	200-500，按不同的集群规格计算
Topic 名称长度	3-64个字符	3-64个字符
单 Topic 最大生产者数量	1000	1000
单 Topic 最大消费者数量	500	500

Group

限制类型	虚拟集群限制	专享集群限制
单集群内 Group 数量上限	1500	2000-5000，按不同的集群规格计算

Group 名称长度	3-64个字符	3-64个字符

消息

限制类型	虚拟集群限制	专享集群限制
消息最大保留时间	3天	3天
消息最大延时	40天	40天
消息大小	4 MB	4 MB
消费位点重置	3天	3天

开源对比

最近更新时间：2024-01-18 10:01:54

TDMQ RocketMQ 版与开源 RocketMQ 的性能对比详情如下：

功能大类	功能项	腾讯云 TDMQ RocketMQ 版	开源 RocketMQ
基础功能	定时消息	优化（精准秒级），支持任意延时刻度	有限支持（指定延迟级别）
	可视化管理能力	支持对集群、topic、group 等进行可视化管理和详细信息浏览，包括订阅关系、消费者状态等。	简单支持，易用性一般，控制台不区分 topic 类型
可用性	弹性伸缩	客户无需关注部署和扩容，无需人工配置，节点注册等操作完全自动化和白屏化；用户可以根据需要随时横向扩容节点数量、增加存储磁盘大小、垂直升级单节点配置	依赖自建运维团队，自动化、白屏化程度低
	高可靠	数据三副本，容器化秒级自动重启，保证宕机时容量和数据都不受损	支持同步复制和异步复制，需要自行设计部署方案和参数，主从同步方案不支持自动切主
	跨 AZ 高可用部署	支持跨 AZ 高可用部署，避免机房级故障	支持但较为繁琐，需要自行设计部署方案和参数
可观测性	资源大盘	核心指标观测、生产消费报表和细粒度监控	简单支持，部分监控指标缺失
	报警管理	消息积压、延迟告警，云监控联动	不支持
安全管控	租户命名空间隔离	控制台可视化支持	不支持，命名空间 bug 较多，无法实现真正隔离
	主子账号管理	全面支持腾讯云主子账号，实现 CAM 主子账号及企业间跨账号的授权服务。	不支持
迁移工具	开源迁移工具	脚本化一键迁移，可以无缝从开源 RocketMQ 迁移	-

购买指南
计费概述
通用集群

最近更新时间：2024-05-14 17:01:17

本文主要介绍 TDMQ RocketMQ 版通用集群的计费方式、计费组成等信息。

开服地域

通用集群目前支持的地域列表如下：

地域	取值
广州	ap-guangzhou
上海	ap-shanghai
南京	ap-nanjing
北京	ap-beijing
新加坡	ap-singapore
弗吉尼亚	na-ashburn
硅谷	na-siliconvalley
中国香港	ap-hongkong
上海金融	ap-shanghai-fsi
上海自动驾驶云	ap-shanghai-fsi
清远	ap-qingyuan
重庆	ap-chongqing

如以上地域不满足您的需求，您可以 提交工单 申请开新的地域和可用区。

计费方式

项目	计费模式	计费说明
集群实例	包年包月-预付费	您在购买通用集群时，系统会根据您选择的集群规格和购买时长计算出费用账单，您需要先结清账单，才能开始使用包年包月资源。此种计费方式适用于业务流量峰值在不同时间段比较平稳，且长期使用的场景。
公网带宽	按小时带宽-后付费	根据使用的公网带宽时长收费，付费模式为后付费，每小时结算一次。适用于业务流量峰值在不同时间段比较平稳，且仅短期使用的场景。

集群实例价格

计费项目

选购 TDMQ RocketMQ 版通用集群的总费用计算方式如下：

集群整体价格 = 计算配置价格 + 存储配置价格。

计费项	说明
计算配置	计算的服务费用，按 TPS 计费，可以自定义 TPS 范围（步长 为4000TPS），计算配置价格根据节点数量呈线性变化。
存储配置	主要为存储的服务费用，可以自定义存储空间，存储配置价格根据存储大小呈线性变化。

价格说明

具体价格以 购买页 配置计算展出的费用为准，本小节主要介绍通用集群的规格。

计算配置

TPS 规格包含生产消息和消费消息的总和，**超出集群 TPS 规格的流量将被严格限流**。

TPS 的计算规则和虚拟集群的 “API 调用次数” 一致：

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。

发送或者消费1条普通消息都计算为1 TPS，发送1条消费1条高级特性消息（如延时消息，事务消息等）都计算为 5 TPS。例如1个 Topic 发送2条事务消息1次，消费1条事务消息，则 TPS 为 2×5+1×5=15 。

消息大小：单条消息大小的上限为4MB，以4 KB为计量单位，不满4KB按4KB计算。

规格类型	单集群最低 TPS 规格（4KB计算）	单集群最高 TPS 规格（4KB计算）	选择步长（TPS）
通用集群	8000	80000	4000

说明：

如当前规格的节点数量限制不满足您的业务量需求，您可以 提交工单 申请支持。

存储配置

存储费用 = 存储空间 × 存储单价。

通用集群提供起步存储 200GB，您可以根据您的业务需求，选择更高的存储空间。

计费项	价格（地域：北京、广州、南京、上海、清远、重庆）	价格（地域：中国香港、弗吉尼亚、新加坡、硅谷）	价格（地域：上海金融、上海自动驾驶云）
存储费用（美元/G/月）	0.1381	0.1796	0.2210

公网带宽价格

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。

地域	价格（单位：美元/Mbps/小时）	
	1Mbps - 5Mbps	6 Mbps及以上
广州/上海/南京/北京/上海金融/重庆/清远/上海自动驾驶云/中国香港	0.0055	0.0193
新加坡/弗吉尼亚/硅谷	0.0048	0.0166

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。

包年包月

超规格 Topic 数量阶梯	价格（美元/个/月，地域：北京、广州、南京、上海、清远、重庆）	价格（美元/个/月，地域：中国香港、弗吉尼亚、新加坡、硅谷）	价格（美元/个/月，地域：上海金融、上海自动驾驶云）
0-100	1.6575	2.1547	2.3519
101-200	1.3812	1.7956	2.2099

201-500	1.1050	1.4365	1.7680
501-1500	0.8287	1.0773	1.3260
1501-2000	0.5525	0.7182	0.8840
2000 以上	0.2762	0.3591	0.4420

专享集群

最近更新时间：2024-04-26 16:39:48

本文主要介绍 TDMQ RocketMQ 版专享集群的计费方式、计费组成等信息。

开服地域

专享集群目前支持的地域列表如下：

地域	取值
广州	ap-guangzhou
上海	ap-shanghai
南京	ap-nanjing
北京	ap-beijing
新加坡	ap-singapore
弗吉尼亚	na-ashburn
硅谷	na-siliconvalley
中国香港	ap-hongkong

如以上地域不满足您的需求，您可以 提交工单 申请开新的地域和可用区。

计费方式

项目	计费模式	计费说明
集群实例	包年包月-预付费	您在购买专享集群时，系统会根据您选择的集群规格和购买时长计算出费用账单，您需要先结清账单，才能开始使用包年包月资源。此种计费方式适用于业务流量峰值在不同时间段比较平稳，且长期使用的场景。
公网带宽	按小时带宽-后付费	根据使用的公网带宽时长收费，付费模式为后付费，每小时结算一次。适用于业务流量峰值在不同时间段比较平稳，且仅短期使用的场景。

集群实例价格

计费项目

选购 TDMQ RocketMQ 版专享集群的总费用计算方式如下：
集群整体价格 = 起步配置价格 + 计算配置价格 + 存储配置价格 = 起步配置价格 + 单节点价格 × 节点数量 + 存储单价 × 存储大小。

计费项	说明
起步配置	新建集群的基础服务费用（如集群管理、消息管理、可观测性、高可用等），是固定费用，不会随着集群规模的增加而增加。
计算配置	主要为计算的服务费用，根据需求提供多种节点规格，每种规格的单节点价格固定，有起步节点数和最高节点数范围限制，计算配置价格根据节点数量呈线性变化。
存储配置	主要为存储的服务费用，可以自定义存储空间（步长是100GB），存储配置价格根据存储大小呈线性变化。

价格说明

具体价格以 购买页 配置计算展出的费用为准，本小节介绍不同规格的专享集群规格性能差异。

说明

如当前规格的节点数量限制不满足您的业务量需求，您可以 提交工单 申请支持。

起步配置

如下规格性能为我们设定的一个参考基准值，实际性能不会低于此规格，以用户压测的数据为准，且不会产生规格外突发弹性能力费用。
TPS 的计算规则和虚拟集群的 “API 调用次数” 一致：
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。
发送或者消费1条普通消息都计算为1 TPS，发送1条消费1条高级特性消息（如延时消息，事务消息等）都计算为 5 TPS。例如1个 Topic 发送2条事务消息1次，消费1条事务消息，则 TPS 为 2×5+1×5=15 。
消息大小：单条消息大小的上限为4MB，以4 KB为计量单位，不满4KB按4KB计算。

规格类型	单节点规格
基础型	TPS（生产+消费）：2000
标准型	TPS（生产+消费）：5000
高阶I型	TPS（生产+消费）：10000
高阶II型	TPS（生产+消费）：18000

计算配置

计算配置费用 = 节点价格 × 节点数量

集群性能规格说明：**集群整体性能 = 节点性能 × 节点数量，**在节点范围内呈线性变化。

规格类型	起步节点数（个）	最高节点数（个）	单节点起步TPS规格（4KB计算）
基础型	2	10	2000
标准型	2	10	5000
高阶I型	2	20	10000
高阶II型	2	20	18000

存储配置

存储费用 = 存储空间 × 存储单价。

专享集群各版本单节点均提供起步存储 200GB，您可以根据您的业务需求，选择更高的存储空间。选择步长为 100GB。

公网带宽价格

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。

地域	价格（单位：美元/Mbps/小时）	
	1Mbps - 5Mbps	6 Mbps及以上
广州/上海/南京/北京	0.0055	0.0194
新加坡	0.0048	0.0166

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。

包年包月

超规格 Topic 数量阶梯	价格（地域：北京、广	价格（地域：中国香	价格（地域：上海金

	州、上海、南京、重庆）	港、新加坡、弗吉尼亚、硅谷）	融、上海自动驾驶专区）
0-100	1.6598 美元/个/月	2.1577 美元/个/月	2.6556 美元/个/月
101-200	1.3831 美元/个/月	1.7808 美元/个/月	2.1918 美元/个/月
201-500	1.1065 美元/个/月	1.4385 美元/个/月	1.7704 美元/个/月
501-1500	0.8299 美元/个/月	1.0788 美元/个/月	1.3278 美元/个/月
1501-2000	0.5479 美元/个/月	0.7192 美元/个/月	0.8852 美元/个/月
2000 以上	0.2766 美元/个/月	0.3596 美元/个/月	0.4429 美元/个/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、重庆）	价格（地域：中国香港、新加坡、弗吉尼亚、硅谷）	价格（地域：上海金融、上海自动驾驶专区）
0-100	0.0035 美元/个小时	0.0045 美元/个小时	0.0055 美元/个小时
101-200	0.0028 美元/个小时	0.0036 美元/个小时	0.0044 美元/个小时
201-500	0.0022 美元/个小时	0.0029 美元/个小时	0.0035 美元/个小时
501-1500	0.0017 美元/个小时	0.0022 美元/个小时	0.0028 美元/个小时
1501-2000	0.0011 美元/个小时	0.0014 美元/个小时	0.0018 美元/个小时
2000 以上	0.0006 美元/个小时	0.0007 美元/个小时	0.0009 美元/个小时

各规格 Topic 限额

节点规格	Topic 限额
基础型	250 * 节点数
标准型	300 * 节点数
高阶I型	350 * 节点数
高阶II型	400 * 节点数

虚拟集群

最近更新时间：2023-09-12 16:05:09

TDMQ RocketMQ 共享版已于 2022年12月28日结束公测，全面商业化并开始计费。本文介绍 TDMQ RocketMQ 共享版（虚拟集群）的计费方式、计费组成等信息。

开服地域

虚拟集群目前支持的地域列表如下：

地域	取值
广州	ap-guangzhou
上海	ap-shanghai
上海金融	ap-shanghai-fsi
北京	ap-beijing
南京	ap-nanjing
北京金融	ap-beijing-fsi
中国香港	ap-hongkong
新加坡	ap-singapore
硅谷	na-siliconvalley
法兰克福	eu-frankfurt
首尔	ap-seoul
孟买	ap-mumbai
弗吉尼亚	na-ashburn
雅加达	ap-jakarta

计费方式

TDMQ RocketMQ 虚拟集群计费方式为**按量付费（后付费）**，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

TDMQ RocketMQ 版虚拟集群以集群的形式售卖，在按量计费模式下的计费公式如下：
总费用 = API 调用次数费用 + Topic 资源占用费用 =（发送消息 API 调用次数 + 消费消息 API 调用次数）× API 调用次数单价 + Topic 个数 × 天数 × Topic 单价。

计费项	计费规则
API 调用次数费用	API 调用次数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条普通消息都计算为1次 API 调用，无论消息是否发送成功或者消费成功，只要发起 API 调用都会进行计费。 高级特性消息：发送1条消费1条高级特性消息都计算为5次 API 调用。例如1个 Topic 发送2条事务消息1次，消费1条事务消息，则 API 调用次数为2×5+1×5=15次。 消息大小： 单条消息大小的上限为4MB，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，将以⌈18/4⌉=5次 API 调用次数计费。**⌈ ⌉表示向上取整数。**
Topic 资源占用费用	Topic 计费单价会根据每个 Topic 当天的产生 API 调用次数阶梯变化，每个 Topic 每天都会被收取一次资源占用费，每天的 Topic 资源占用费为所有 Topic 当天产生费用的总和。**无论 Topic 当天是否有进行消息收发，都会产生一次计费。**

价格说明

免费额度

每个主账户每个月拥有 2000 万次 API 免费调用额度。
说明：
该免费额度为新品优惠期间暂时提供的免费额度，正式收费前将以公告、短信、站内信、邮箱等方式进行通知。

API 调用次数价格

计费阶梯	调用次数（亿次/月）	公有云单价（美元/百万次）		
		大陆内地	境外	金融区
第一阶梯	0~10	0.26	0.33	0.41
第二阶梯	10~50	0.21	0.13	0.29
第三阶梯	50~500	0.17	0.23	0.29
第四阶梯	500以上	0.14	0.19	0.23

说明：
以上阶梯按照腾讯云账号（UIN）维度、按账期（月度）累计 API 调用次数。
计费示例
假设您的实例属于广州地域，每天消息收发情况如下：
生产5千万条普通消息，消息消费次数为7千万次（包含消息投递失败重试的次数），每条消息大小为20KB。
生产3千万条事务消息，消息消费次数为3千万次，每条消息大小为每条消息大小为4KB。
生产1千万条延时消息，消息消费次数为1千万次，每条消息大小为每条消息大小为2KB。
则1天产生的 API 调用次数为（5千万＋7千万）×⌈20/4⌉＋（3千万＋3千万）× 5 ×⌈4/4⌉＋（1千万＋1千万）× 5 ×⌈2/4⌉= 10亿次。

若9月每天都产生10亿次 API 调用次数，则产生的 API 调用费用如下：
9月1号，API 调用次数为10亿，当月累计的 API 调用次数为10亿，属于第一阶梯范围，单价为 0.26美元，收取费用 10 × 0.26 × 100 = 260美元 (100 即 10亿=100万*100，按百万次调用收费)。
9月2号，API 调用次数为10亿，当月累计的 API 调用次数为20亿，属于第二阶梯范围，单价为 0.21美元，收取费用 10 × 0.21 × 100 = 210美元 (100 即 10亿=100万*100，按百万次调用收费)。
9月3号，API 调用次数为10亿，当月累计的 API 调用次数为30亿，属于第二阶梯范围，单价为 0.21美元，收取费用 10 × 0.21 × 100 = 210美元 (100 即 10亿=100万*100，按百万次调用收费)。
9月4号，……以此类推。
10月1号将从0开始累计 API 调用次数。

Topic 资源占用费用

Topic 计费单价会根据每个 Topic 当天的产生 API 调用次数阶梯变化，每天的 Topic 资源占用费为所有 Topic 当天产生费用的总和。Topic 创建不满一天按一天计算。
说明：
由于计费周期从0点开始计算，如您当天删除主题资源后，资源占用费当天还是会收取，因此第二天的账单上还是会有体现；之后将不再收取任何费用。

计费阶梯	调用次数（万次/个/日）	公有云单价（美元/个/日）		
		大陆内地	境外	金融区
第一阶梯	0~100	0.26	0.33	0.41
第二阶梯	100~1000	0.13	0.17	0.21
第三阶梯	1000以上	0	0	0

计费示例

假设您的实例属于广州地域，您一共创建了若3个 Topic。

Topic 1 当天 API 调用次数20万，属于第一阶梯范围，则当天收取资源占用费0.26美元。

Topic 2当天 API 调用次数200万，属于第二阶梯范围，则当天收取资源占用费0.13美元。

Topic 3当天 API 调用次数50万，属于第一阶梯范围，则当天收取资源占用费0.26美元。

则当天收取的 Topic 资源占用费总和 = 0.26 + 0.13 + 0.26 = 0.65美元。

产品系列

最近更新时间：2024-05-14 16:56:26

TDMQ RocketMQ 版按照售卖形式分为专享集群、通用集群和虚拟集群，三个版本的对比差异如下：

功能	专享集群	通用集群	虚拟集群
版本兼容	兼容开源版 4.9 及以下版本	兼容开源版 4.9 及以下版本	兼容开源版 4.9 及以下版本
实例类型	物理资源隔离	物理资源隔离	资源逻辑隔离，底层物理资源共享
计费模式	提供**包年包月**计费模式，具体价格见 购买页	提供**包年包月**计费模式，具体价格见 购买页	提供**按量计费**模式，具体价格见 购买页
TPS 范围	按照不同的节点规格按需自由购买	8000-80000 TPS	适用于 4000 TPS 以下
扩容	自由度高，可以单独扩容节点数量、节点规格（后续支持）和存储空间	支持在 TPS 范围内调整 TPS，对应着底层节点数量的变化，超出 TPS 规格外的流量会被严格限流	不支持扩容
Broker 修复升级周期	快速升级	快速升级	共享集群，周期较长
可用性	99.99%	99.99%	99.95%
高可用能力	支持同地域自定义多可用区部署，提升容灾能力	支持同地域自定义多可用区部署，提升容灾能力	不支持同城跨可用区部署
技术服务	提供参数优化咨询服务，可以针对部分特殊的业务场景定制化参数配置，您可以 提交工单 申请	提供参数优化咨询服务，可以针对部分特殊的业务场景定制化参数配置，您可以 提交工单 申请	基础的故障处理和问题修复
护航能力	提供保驾护航服务，如产品升级、新业务上线、大促营销活动等，保障业务平稳运行	提供保驾护航服务，如产品升级、新业务上线、大促营销活动等，保障业务平稳运行	不支持

购买方式

最近更新时间：2024-01-18 10:02:42

TDMQ RocketMQ 版专享集群采用**包年包月-预付费**的计费方式，您可按照以下步骤购买服务：

1. 登录腾讯云 TDMQ 控制台。
2. 在左侧导航栏选择 **rocketmq > 集群管理**，单击**新建**，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。

欠费说明

最近更新时间：2023-07-21 15:19:17

注意：

如果您是腾讯云合作伙伴的客户，账户欠费下的产品资源处理规则以您与合作伙伴约定的协议为准。

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

消息队列 RocketMQ 共享版以按量付费的模式计算费用，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，集群可以正常使用。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。

退费说明

最近更新时间：2023-03-31 11:31:37

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

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。

账单说明

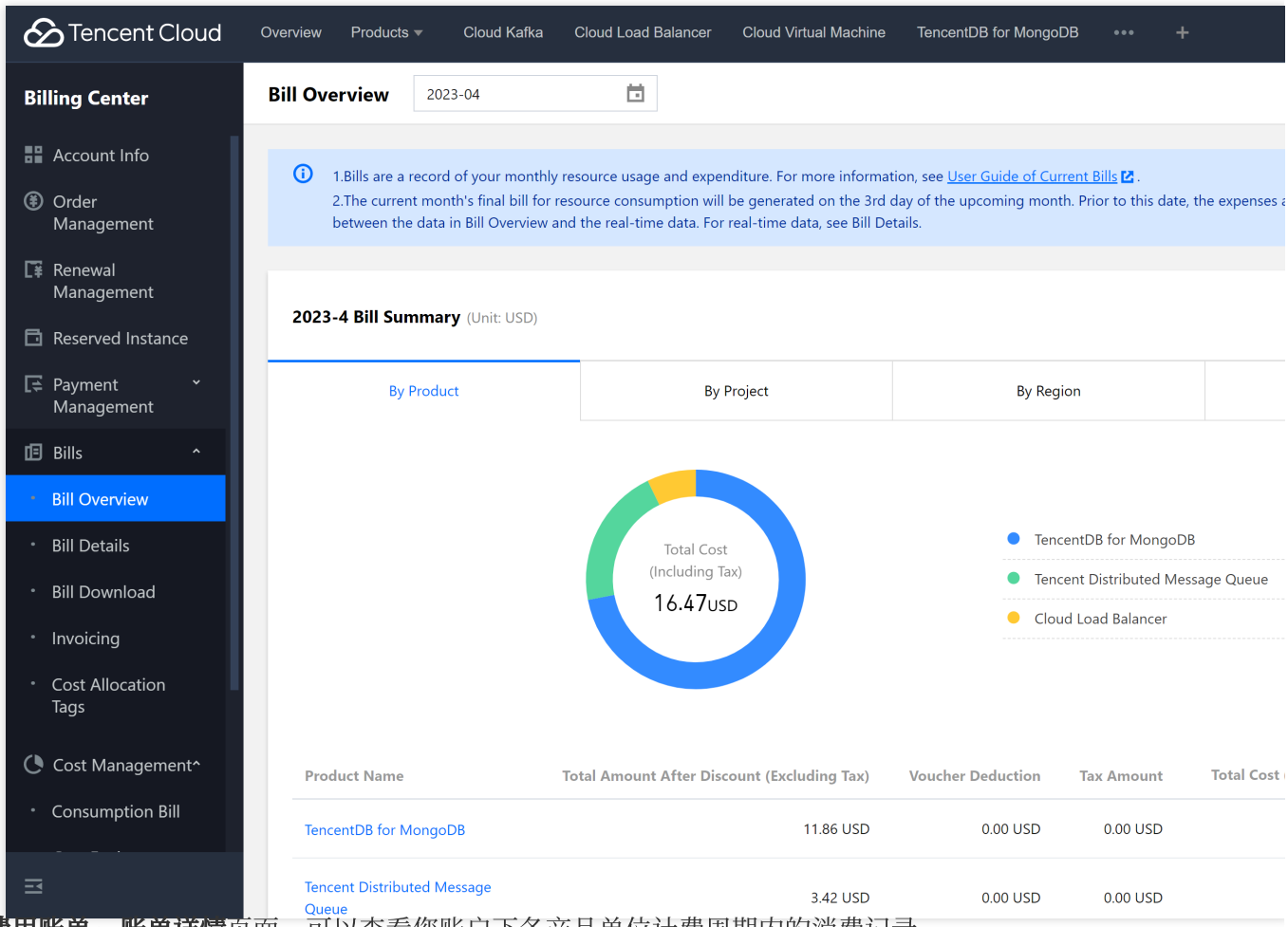
最近更新时间：2023-04-12 11:21:15

操作场景

若您对消息队列 RocketMQ 版的扣费情况产生疑问，可以前往费用中心查看消费明细。

操作步骤

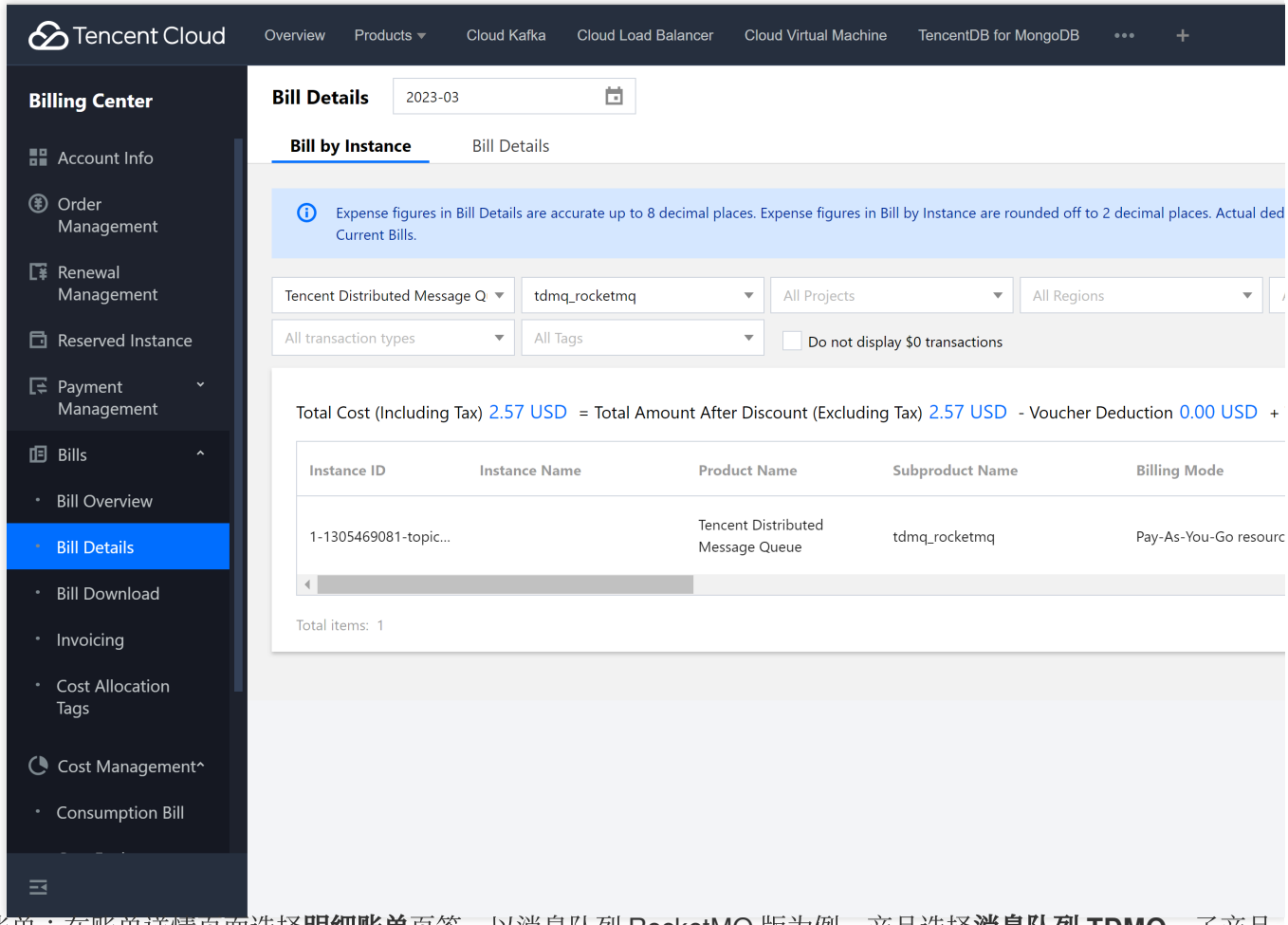
1. 登录 TDMQ 控制台。
2. 在页面右上方，将鼠标悬浮至**费用**，在下拉框中点击**费用账单**，进入费用中心。
3. 在**费用账单** > **账单概览**页面，可以查看您账户下所有产品的消费概览。



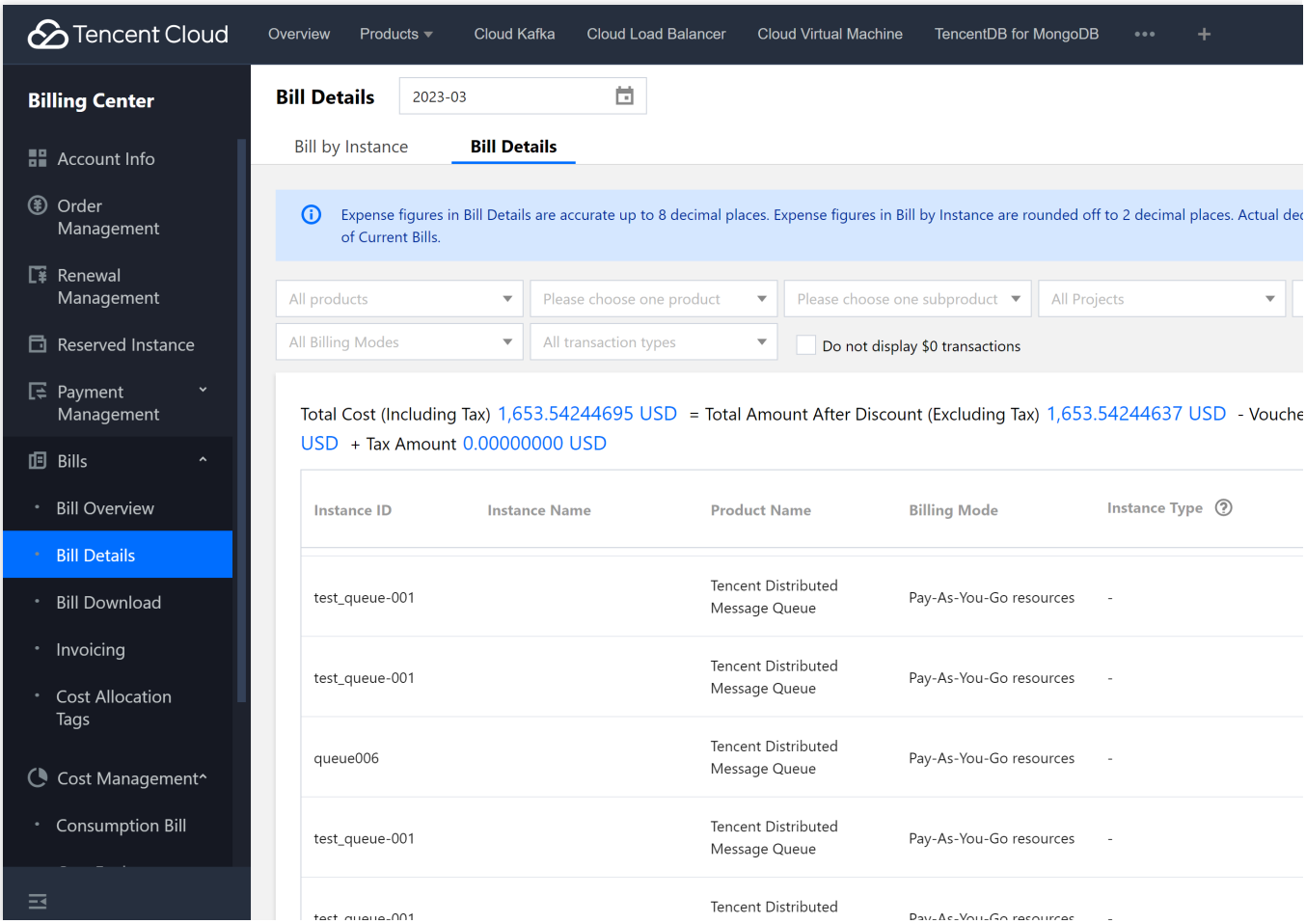
4. 在**费用账单** > **账单详情**页面，可以查看您账户下各产品单位计费周期内的消费记录。

说明：

资源 ID 账单在次月1号出账，资源ID账单数据可能有延迟，查询结果仅供参考；实时扣费数据请您查看明细账单。
资源 ID 账单：在账单详情页面选择**资源 ID 账**单页签，以消息队列 RocketMQ 版为例，产品选择**消息队列 TDMQ**，子产品选择 **消息队列TDMQ RocketMQ版，**可以查看消息队列 RocketMQ 版的各资源实例的消费明细。



明细账单：在账单详情页面选择**明细账单**页签，以消息队列 RocketMQ 版为例，产品选择**消息队列 TDMQ**，子产品选择 **消息队列TDMQ RocketMQ版，**可以查看每个应用在计费周期内 Topic 资源占用和 API 调用次数费用。



快速入门
快速入门概述

最近更新时间：2023-04-14 16:54:48

消息队列 RocketMQ 版支持使用 TCP 协议和 HTTP 协议的多语言客户端 SDK 收发消息，本文主要介绍使用这两种协议的 SDK 收发普通消息的操作流程。

使用说明

消息队列 RocketMQ 版支持的消息类型有普通消息、定时与延时消息、顺序消息和事务消息四种，本文以介绍收发普通消息为例，其他类型的消息介绍请参见 消息类型。

说明：

不同消息类型的 Topic 不能混用，因此您创建的普通消息的 Topic，不能用于收发其他类型的消息。

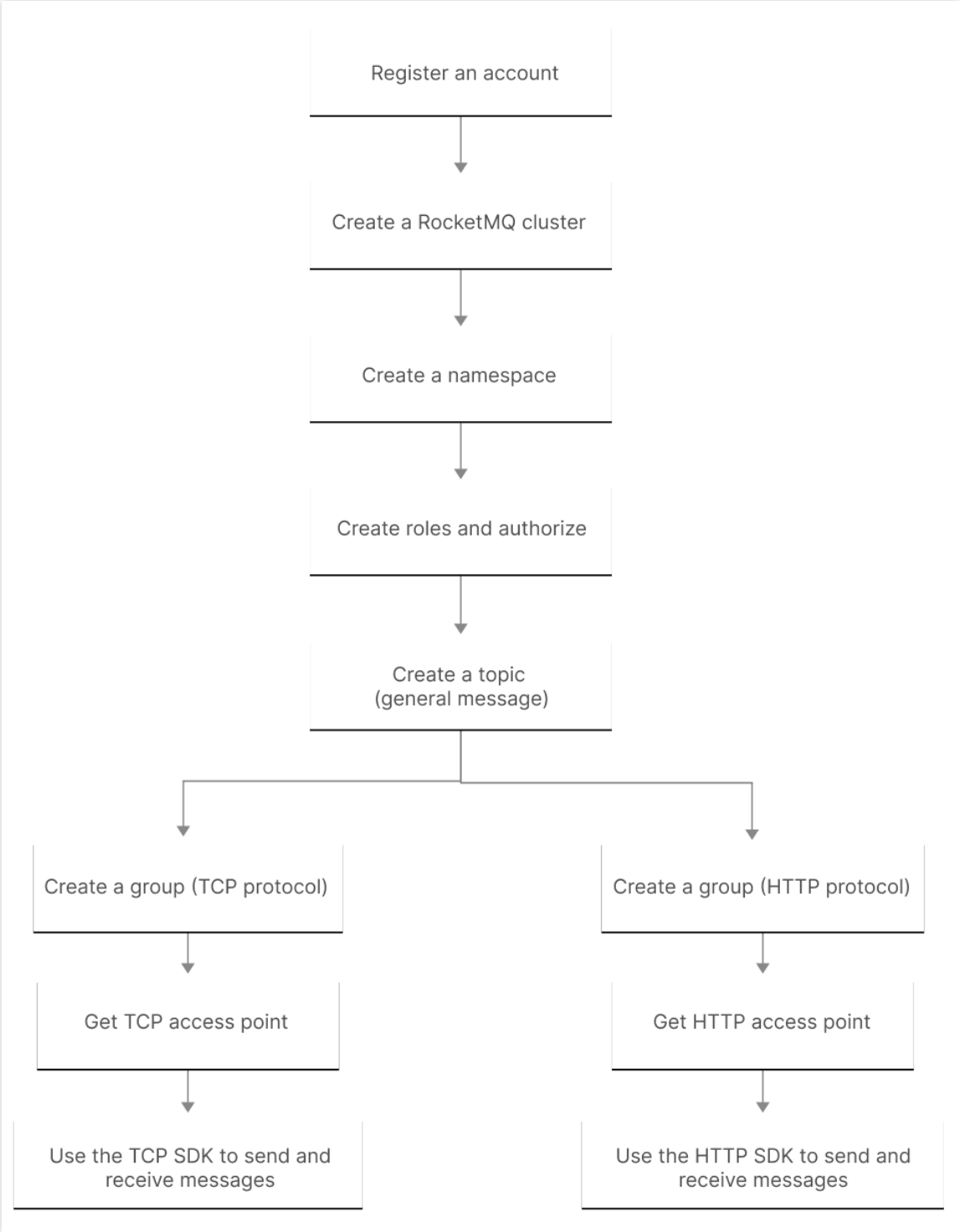
消息队列 RocketMQ 版支持 TCP 协议和 HTTP 协议接入，因此，建议您分别为两种协议创建对应类型的 Group，若多个消费者使用同一个 Group 消费消息，其中部分消费者使用 TCP 协议，部分消费者使用 HTTP 协议，可能会导致消费失败、部分消息重复或丢失。

TCP 协议和 HTTP 协议均支持公网和 VPC 网络接入地址，生产环境默认推荐 VPC 网络，公网接入地址默认不开通，如需开通公网访问，虚拟集群可以提交工单申请，专享集群可以通过调整公网带宽来开启/关闭公网访问。建议公网访问只在测试、调试等不影响生产环境的场景下使用。

说明：

TCP 协议和 HTTP 协议在各地域均可提供支持，若您当前的实例所在地域未支持 HTTP 协议且有使用需求，可以 提交工单申请。

使用流程


Register an account
Create a RocketMQ cluster
Create a namespace
Create roles and authorize
Create a topic
(general message)
Create a group (TCP protocol)
Get TCP access point
Use the TCP SDK to send and receive messages
Create a group (HTTP protocol)
Get HTTP access point
Use the HTTP SDK to send and receive messages

使用 TCP 协议收发消息
资源创建与准备

最近更新时间：2023-09-12 16:08:35

操作场景

在使用 TCP 协议的 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。

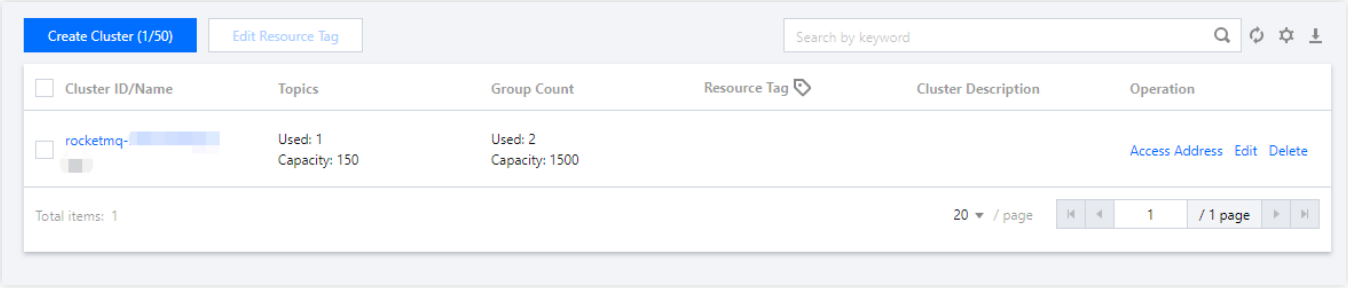
前提条件

已 注册腾讯云账号。

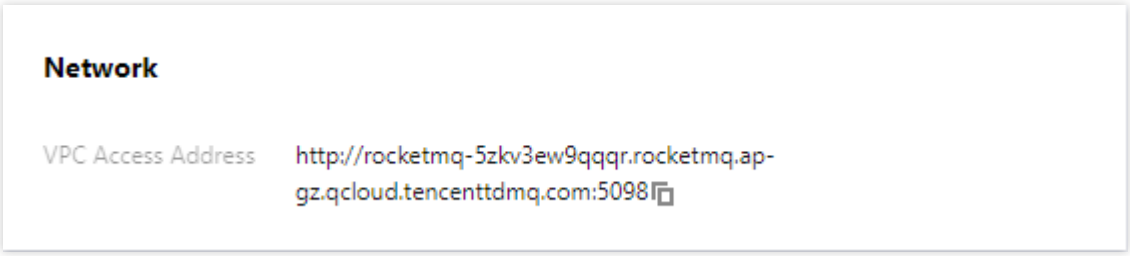
操作步骤

步骤1：新建集群

1. 登录 TDMQ 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**，集群类型可以选择**虚拟集群**，填写好集群名称和说明，单击**确定**，创建一个集群。

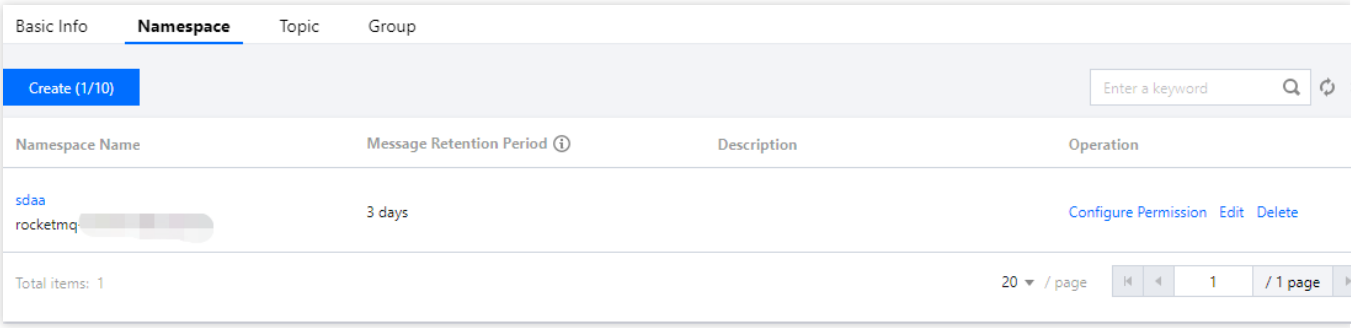


3. 在集群列表页面，单击创建好的集群ID，在集群基本信息页面的网络模块，可以查看到集群的接入点信息。



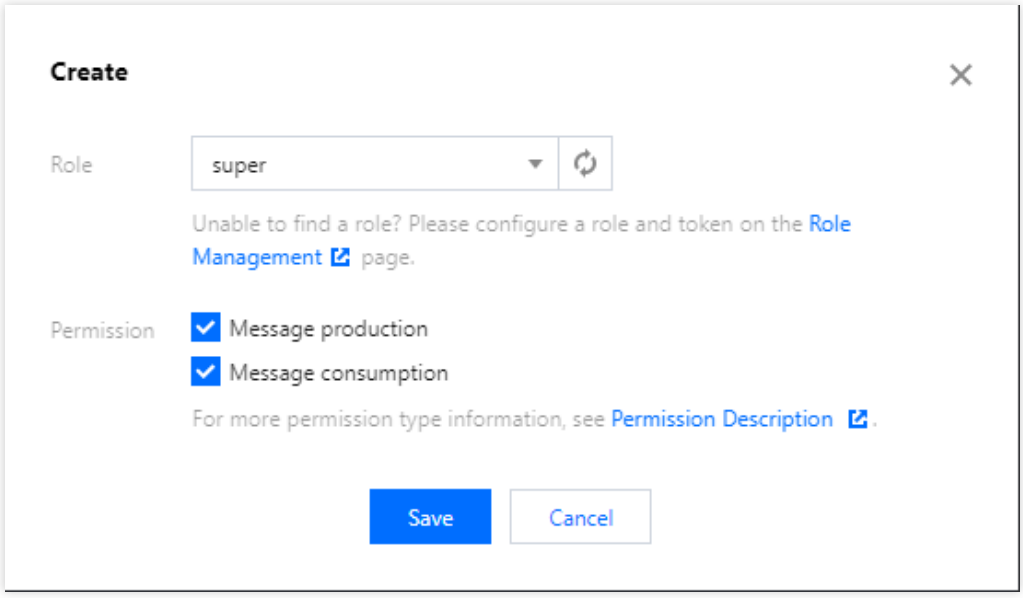
步骤2：创建命名空间

1. 在集群列表页面，单击**步骤1**创建好的集群的“ID”，进入集群基本信息页。
2. 选择顶部的**命名空间**页签，单击**新建**，设置好命名空间名称和描述信息，创建一个命名空间。



步骤3：创建角色并授权

1. 在左侧导航栏选择**角色管理**，单击**新建**，创建一个角色。
2. 在**集群管理**页面，单击**步骤1**创建好的集群的“ID”，进入**集群**详情页面。
3. 在页面上方选择**命名空间**页签，找到刚刚创建的命名空间，单击操作栏的**配置权限**。
4. 在**配置权限**页面，单击**添加角色**，为刚刚创建的角色添加生产消费权限。

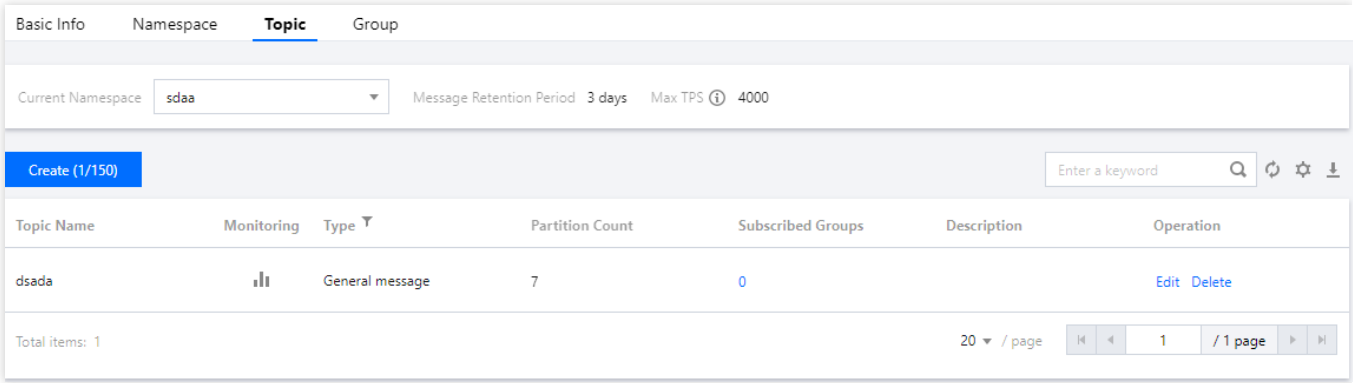


步骤4：创建 Topic

1. 在命名空间列表页，选择顶部的 **Topic** 页签，进入 Topic 列表页。
2. 选择 步骤3 创建好的命名空间，单击**新建**，填写好 Topic 名称，消息类型选择**普通消息**，单击**确定**，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。

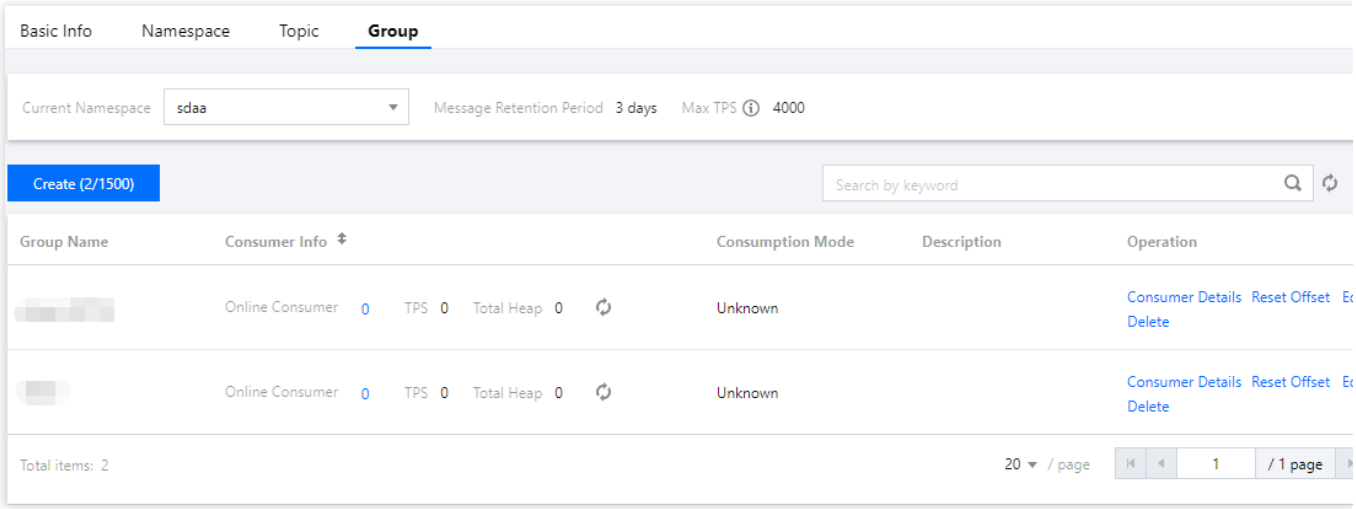


步骤5：创建 Group

1. 在 Topic 列表页，选择顶部的 **Group** 页签，进入 Group 列表页。
2. 选择刚刚创建好的命名空间，单击**新建**，填写好 Group 名称，协议类型选择 **TCP**，单击**确定**，创建一个 Group。

说明：

消息队列 RocketMQ 版支持 HTTP 协议和 TCP 协议，因此，建议您分别为两种协议创建对应类型的 Group，若多个消费者使用同一个 Group 消费消息，其中部分消费者使用 TCP 协议，部分消费者使用 HTTP 协议，可能会导致消费失败、部分消息重复或丢失。



使用 SDK 收发普通消息

最近更新时间：2023-03-28 10:15:36

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

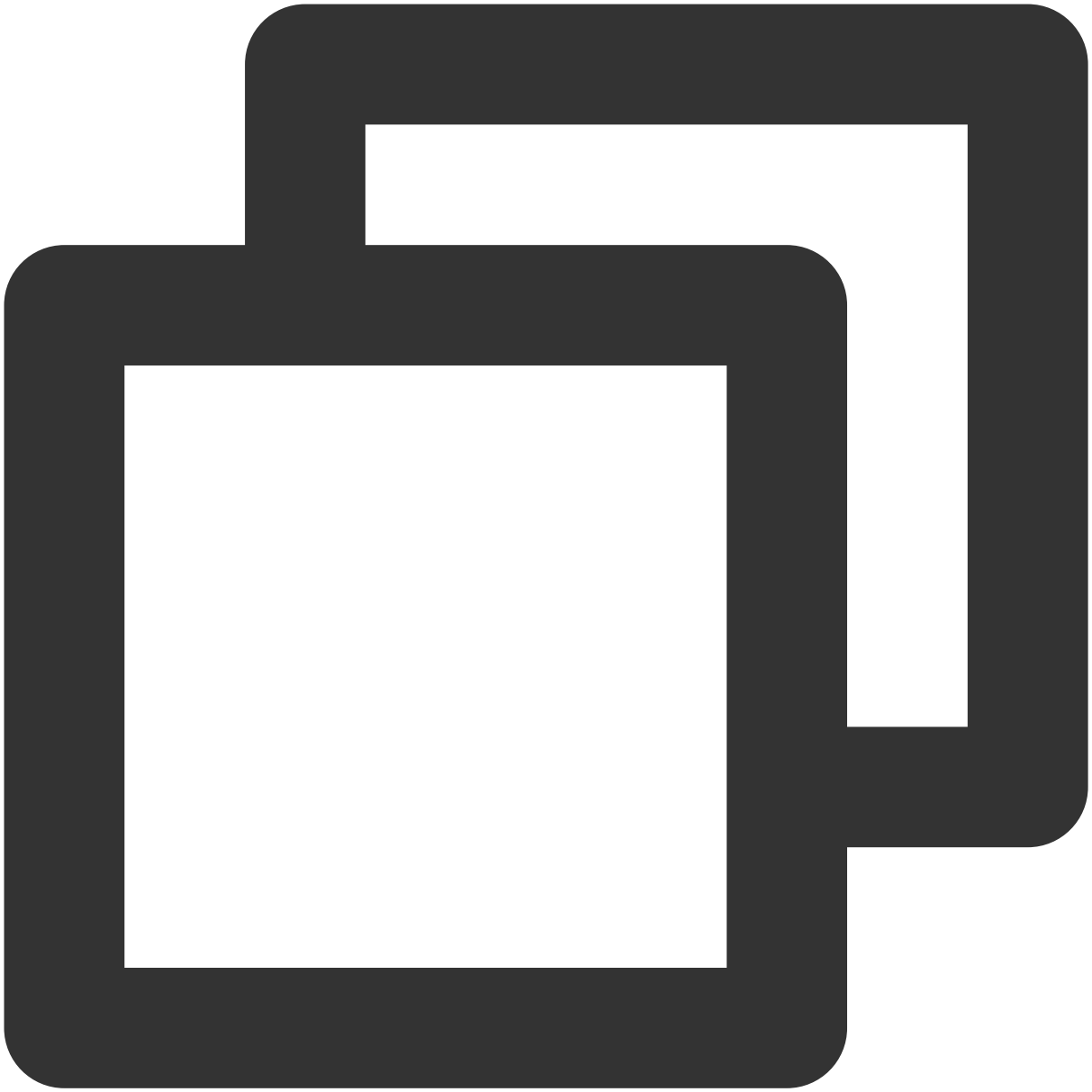
操作步骤

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：

说明

依赖版本要求 ≥ 4.9.3。

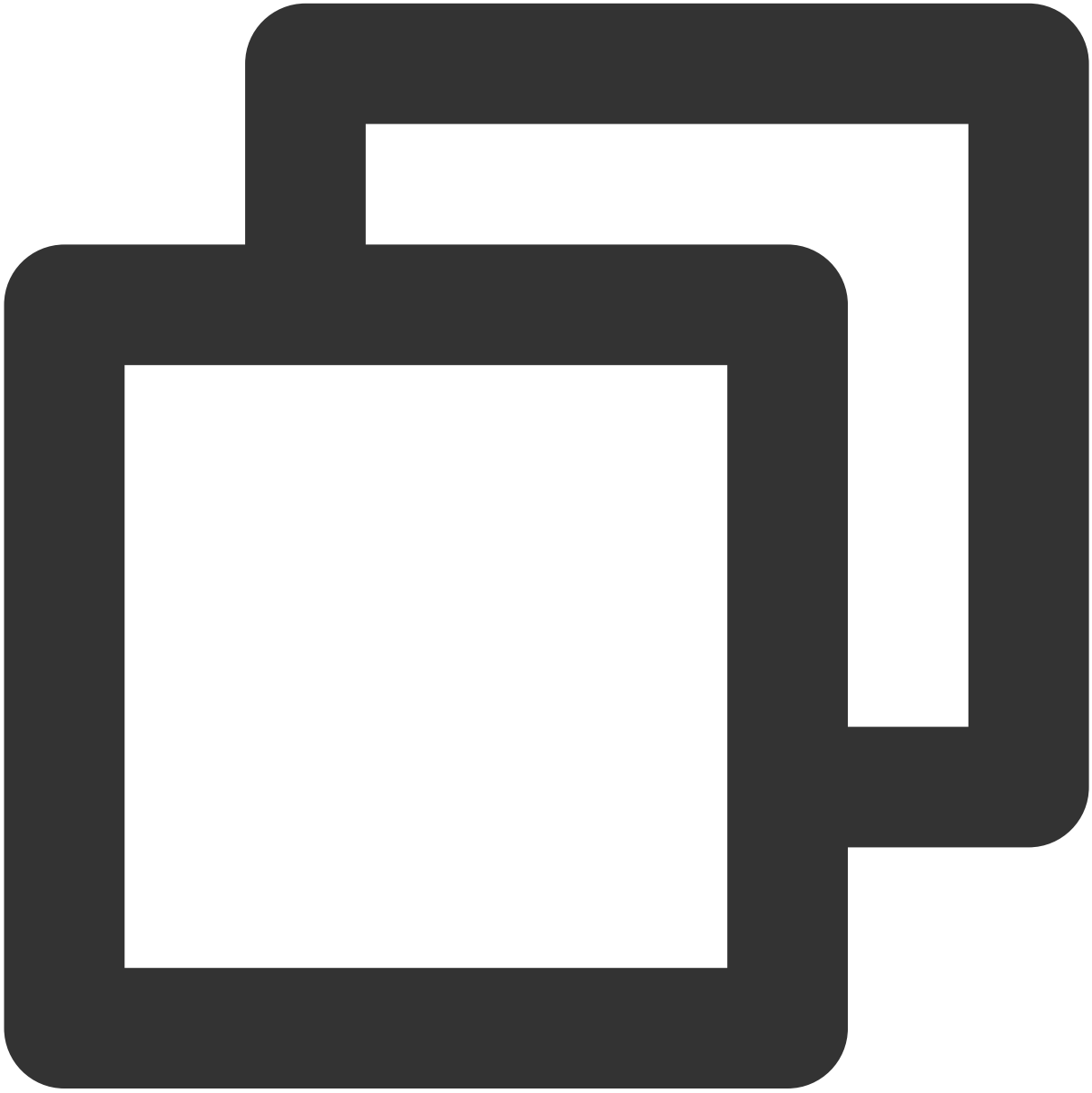


```
<!-- in your <dependencies> block -->
<dependency>
     <groupId>org.apache.rocketmq</groupId>
     <artifactId>rocketmq-client</artifactId>
     <version>4.9.3</version>
</dependency>

<dependency>
     <groupId>org.apache.rocketmq</groupId>
     <artifactId>rocketmq-acl</artifactId>
     <version>4.9.3</version>
```

```
</dependency>
```

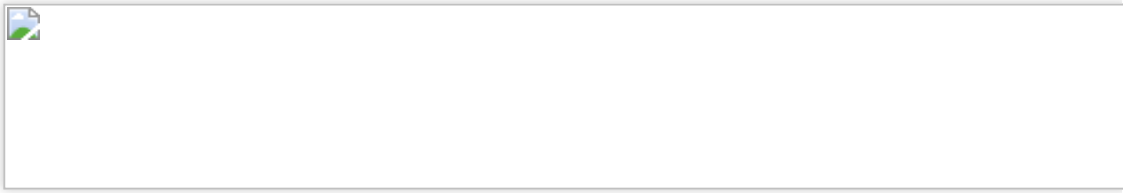
步骤2：生产消息

1. 创建消息生产者



```
// 实例化消息生产者Producer
DefaultMQProducer producer = new DefaultMQProducer(
     namespace,
     groupName,
     new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))
```

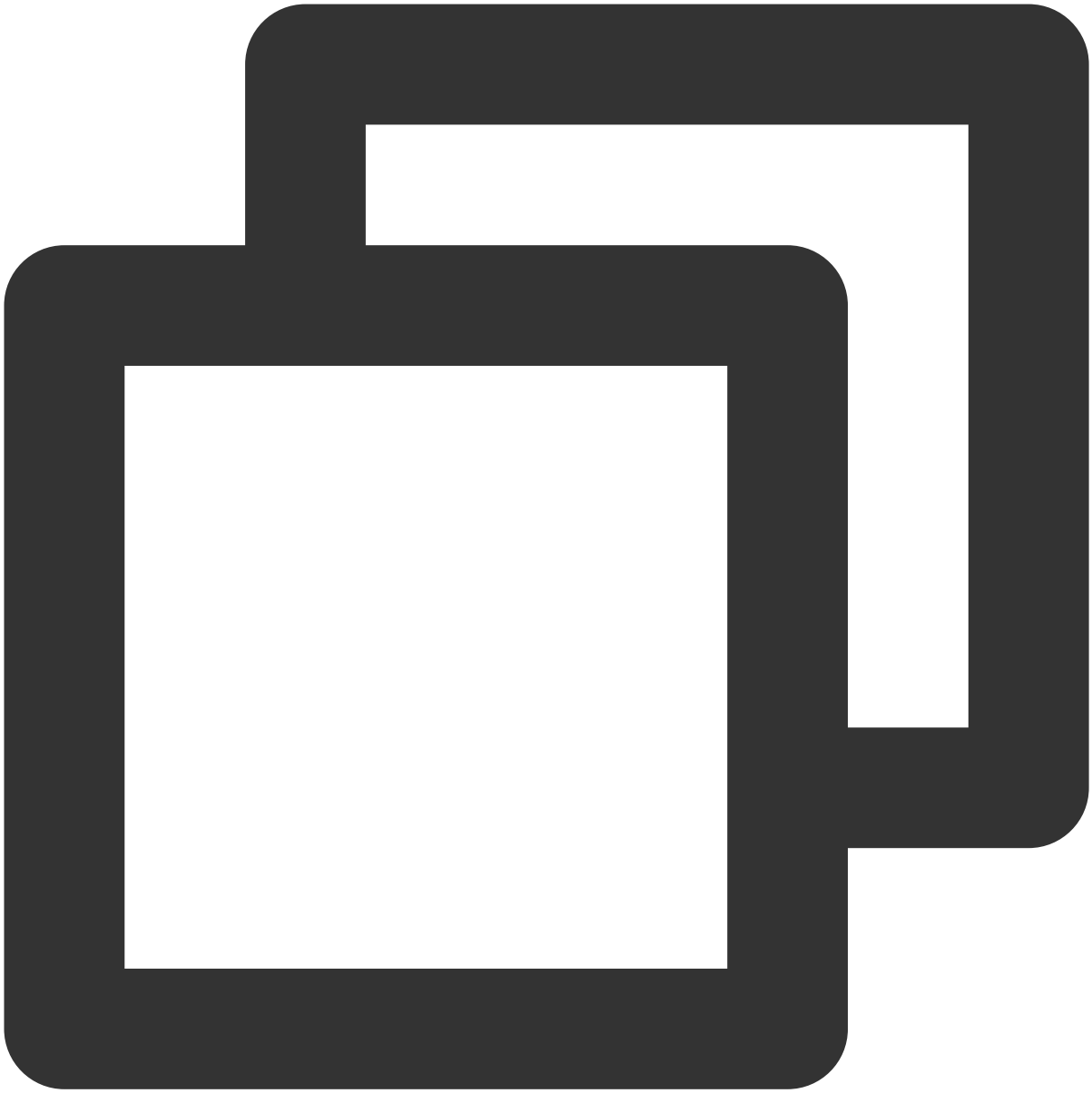
```
        // ACL权限
);
// 设置NameServer的地址
producer.setNamesrvAddr(nameserver);
// 启动Producer实例
producer.start();
```

参数	说明
namespace	命名空间的名称，在控制台命名空间页面复制。 
groupName	生产者组名称，在控制台集群管理中 Group 页签中复制。
nameserver	集群接入地址，在控制台集群基本信息页面的**网络**模块复制。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

2. 发送消息

发送消息由多种方式，同步发送、异步发送、单向发送等。

同步发送

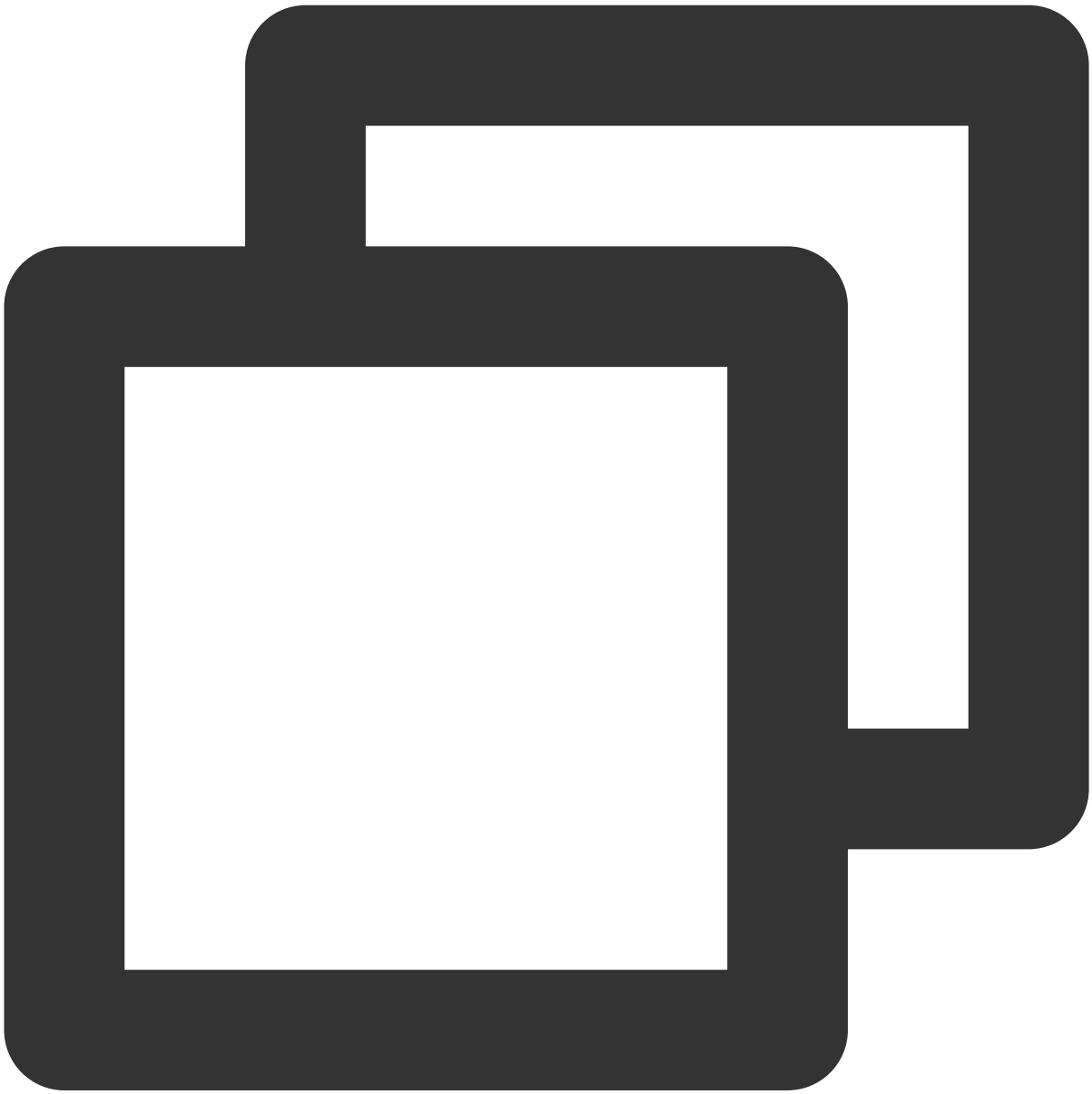


```
for (int i = 0; i < 10; i++) {
     // 创建消息实例，设置topic和消息内容
     Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getBytes(
     // 发送消息
     SendResult sendResult = producer.send(msg);
     System.out.printf("%s%n", sendResult);
}
```

参数	说明

topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
TAG	用来设置消息的 TAG。

异步发送

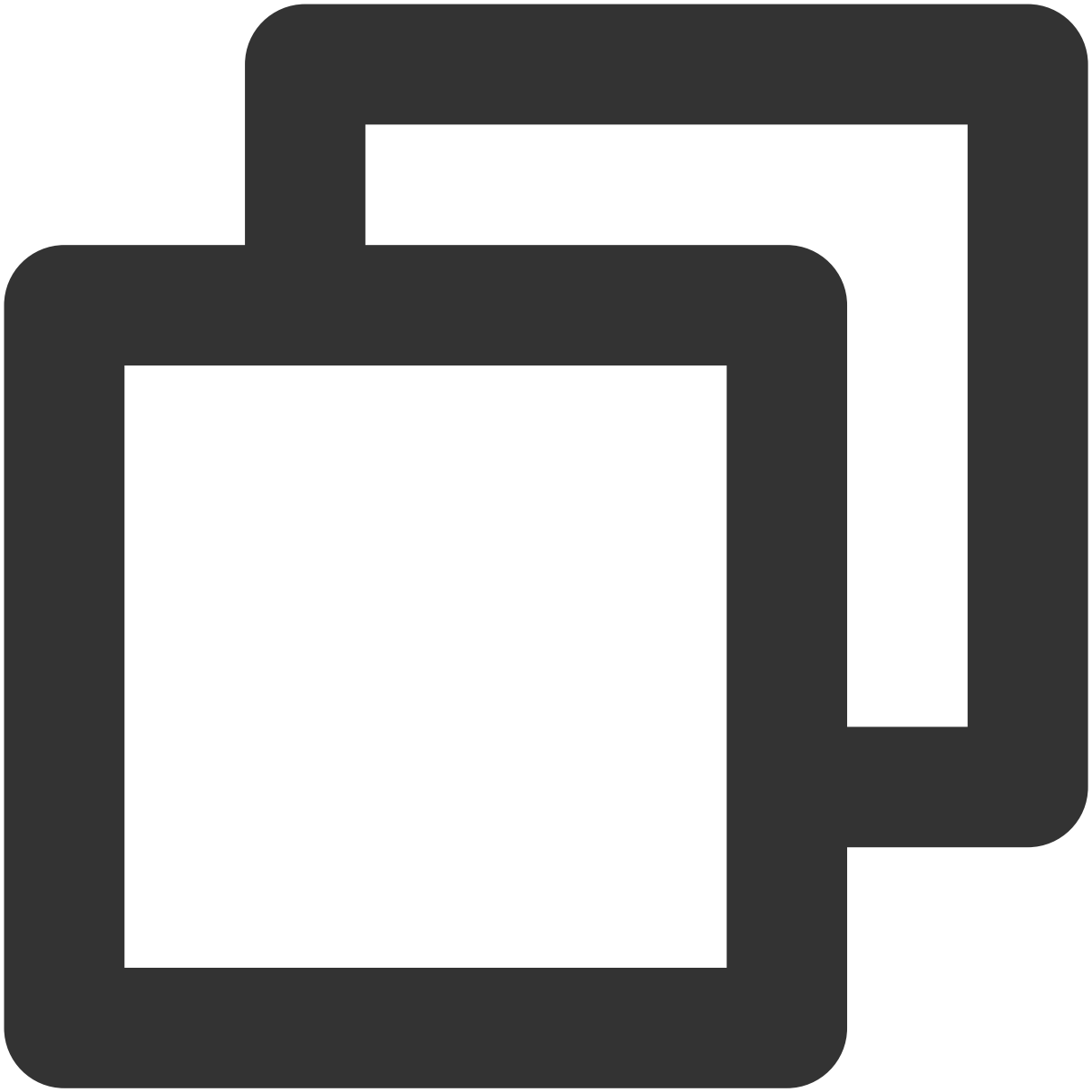


```
// 设置发送失败后不重试
producer.setRetryTimesWhenSendAsyncFailed(0);
// 设置发送消息的数量
int messageCount = 10;
final CountDownLatch countDownLatch = new CountDownLatch(messageCount);
```

```
for (int i = 0; i < messageCount; i++) {
    try {
        final int index = i;
        // 创建消息实体，设置topic和消息内容
        Message msg = new Message(topic_name, "TAG", ("Hello rocketMq " + inde
        producer.send(msg, new SendCallback() {
            @Override
            public void onSuccess(SendResult sendResult) {
                // 消息发送成功逻辑
                countDownLatch.countDown();
                System.out.printf("%-10d OK %s %n", index, sendResult.
            }

            @Override
            public void onException(Throwable e) {
                // 消息发送失败逻辑
                countDownLatch.countDown();
                System.out.printf("%-10d Exception %s %n", index, e);
                e.printStackTrace();
            }
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}
countDownLatch.await(5, TimeUnit.SECONDS);
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
TAG	用来设置消息的 TAG。

单向发送



```
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, "TAG", ("Hello RocketMQ " + i).getBytes(
    // 发送单向消息
    producer.sendOneway(msg);
}
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。

TAG	用来设置消息的 TAG。

说明

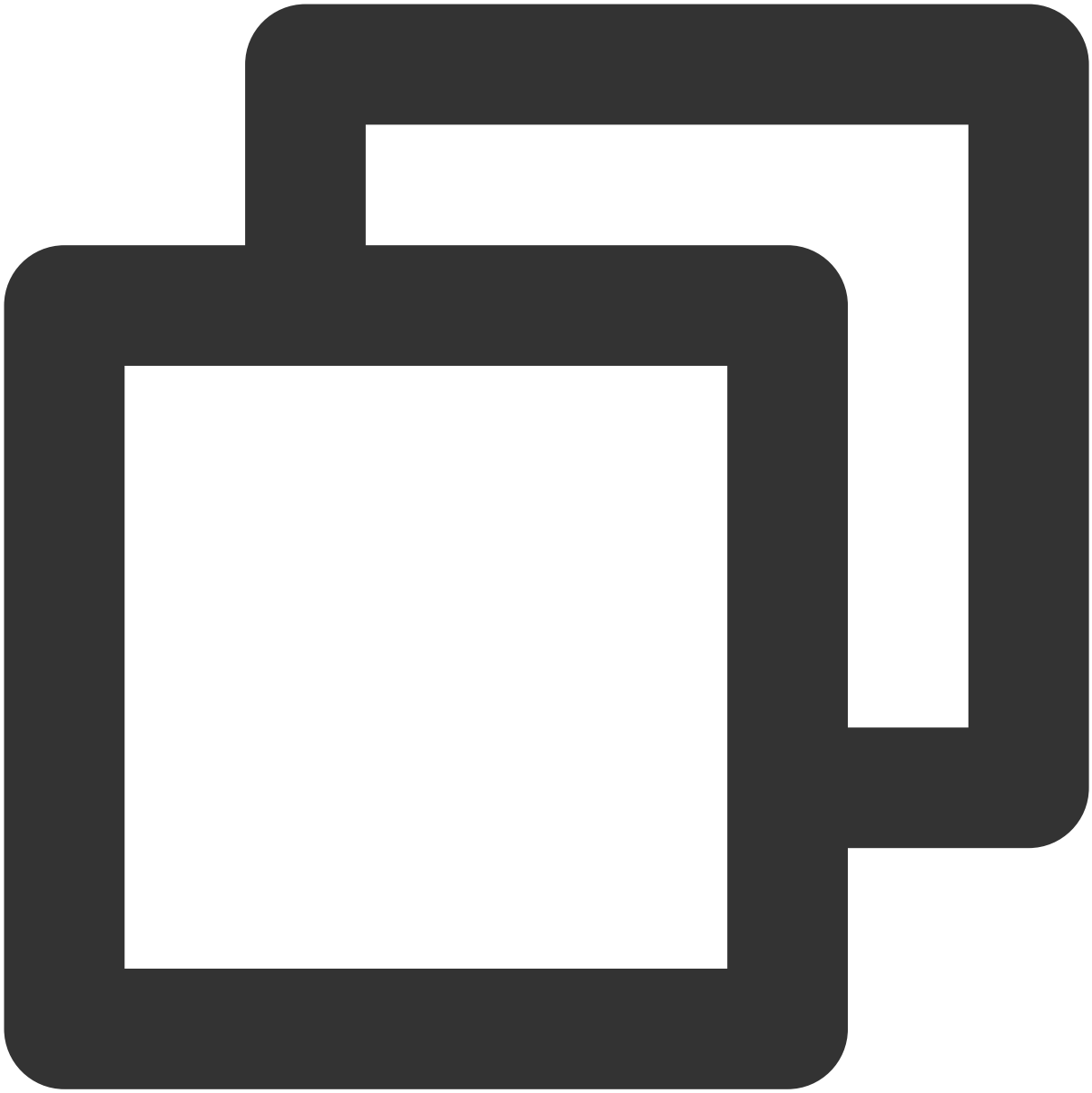
批量发送及其他情况可参见 Demo 或 社区文档。

步骤3：消费消息

1. 创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。

push 消费者

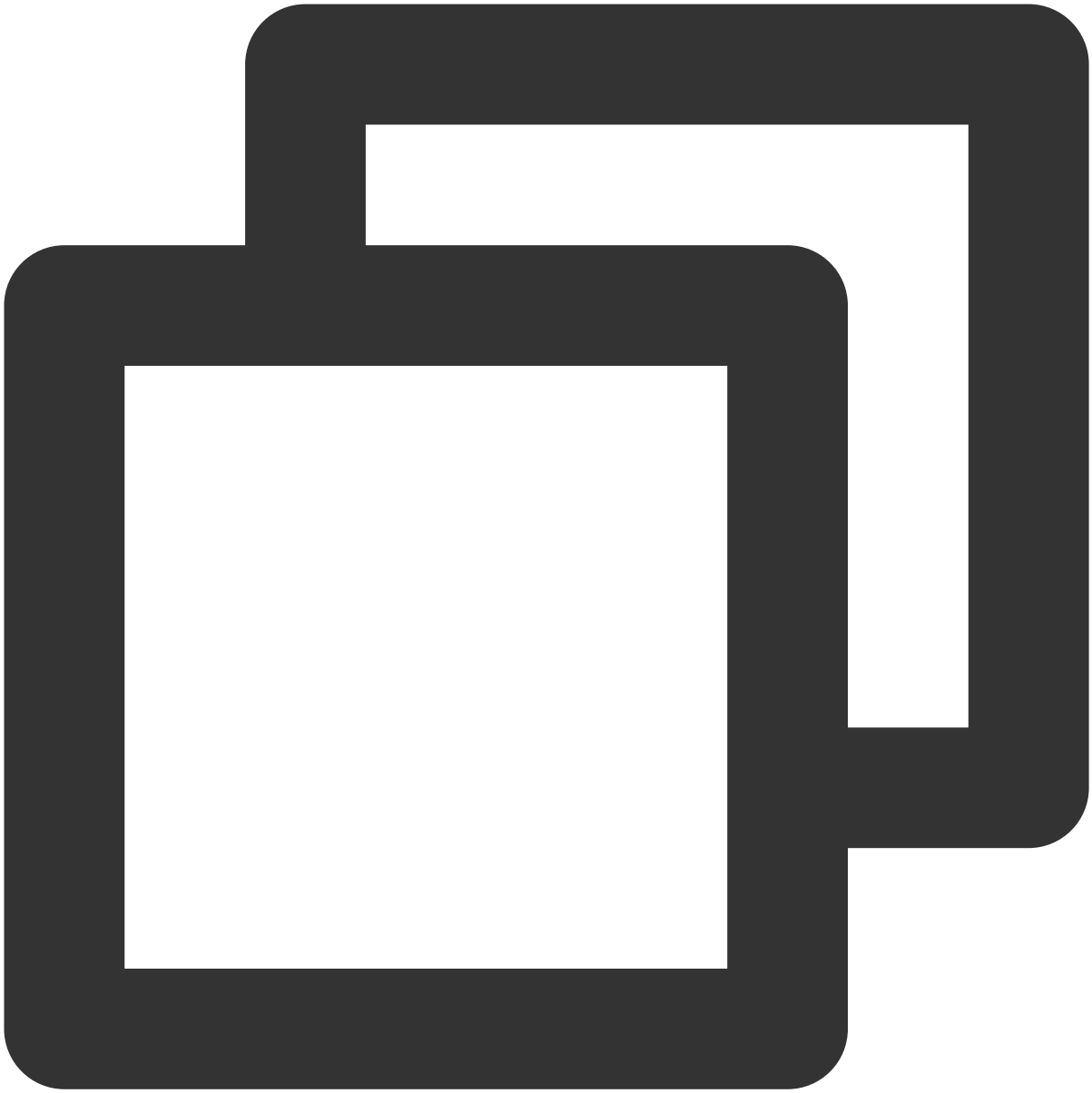


```
// 实例化消费者
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
     namespace,
     groupName,
     new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //ACL权限
// 设置NameServer的地址
pushConsumer.setNamesrvAddr(nameserver);
```

参数	说明

namespace	命名空间的名称，在控制台命名空间页面复制。 
groupName	生产者组名称，在控制台集群管理中 Group 页签中复制。
nameserver	集群接入地址，在控制台集群基本信息页面的**网络**模块复制。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

pull 消费者



```
// 实例化消费者
DefaultLitePullConsumer pullConsumer = new DefaultLitePullConsumer(
     namespace,
     groupName,
     new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)));
// 设置NameServer的地址
pullConsumer.setNamesrvAddr(nameserver);
// 设置从第一个偏移量开始消费
pullConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
```

参数	说明
namespace	命名空间的名称，在控制台命名空间页面复制，格式是集群 ID + |+ 命名空间。
groupName	生产者组名称，在控制台集群管理中 Group 页签中复制。
nameserver	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。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

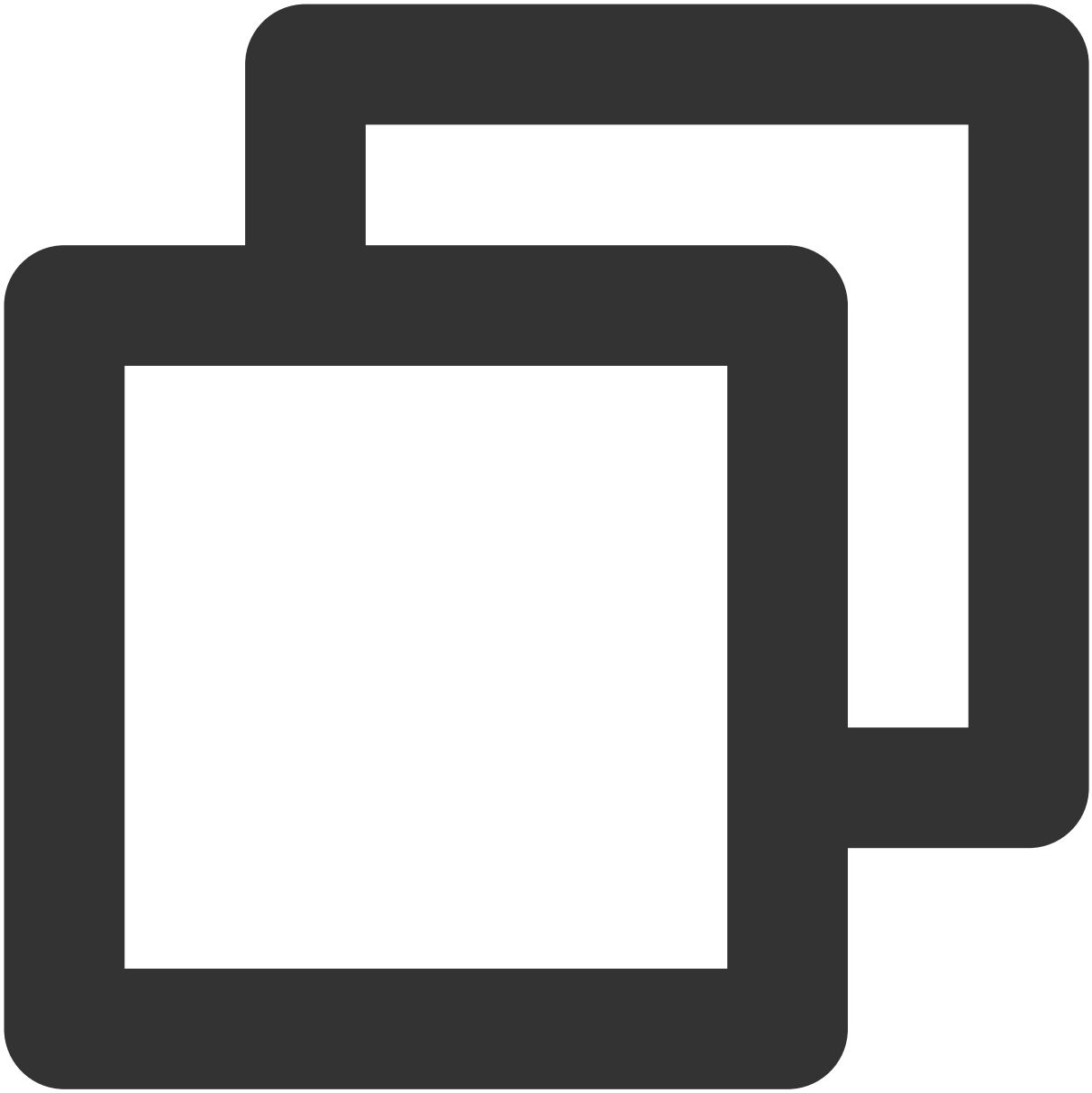
说明

更多消费类型可参见 Demo 或 RocketMQ 官方文档 。

2. 订阅消息

根据消费模式不同，订阅方式也有所区别。

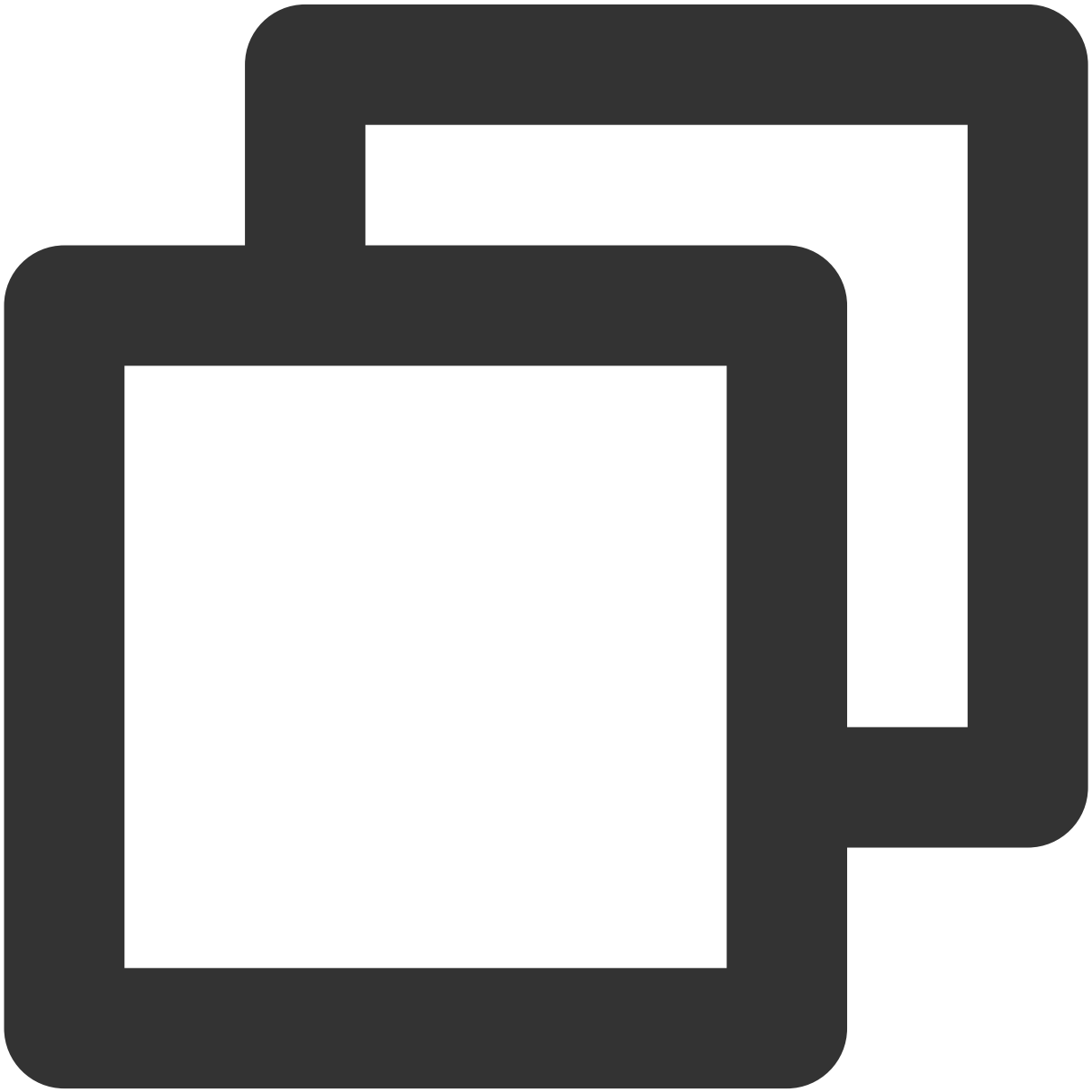
push 订阅



```
// 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().get
     // 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();
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

pull 订阅



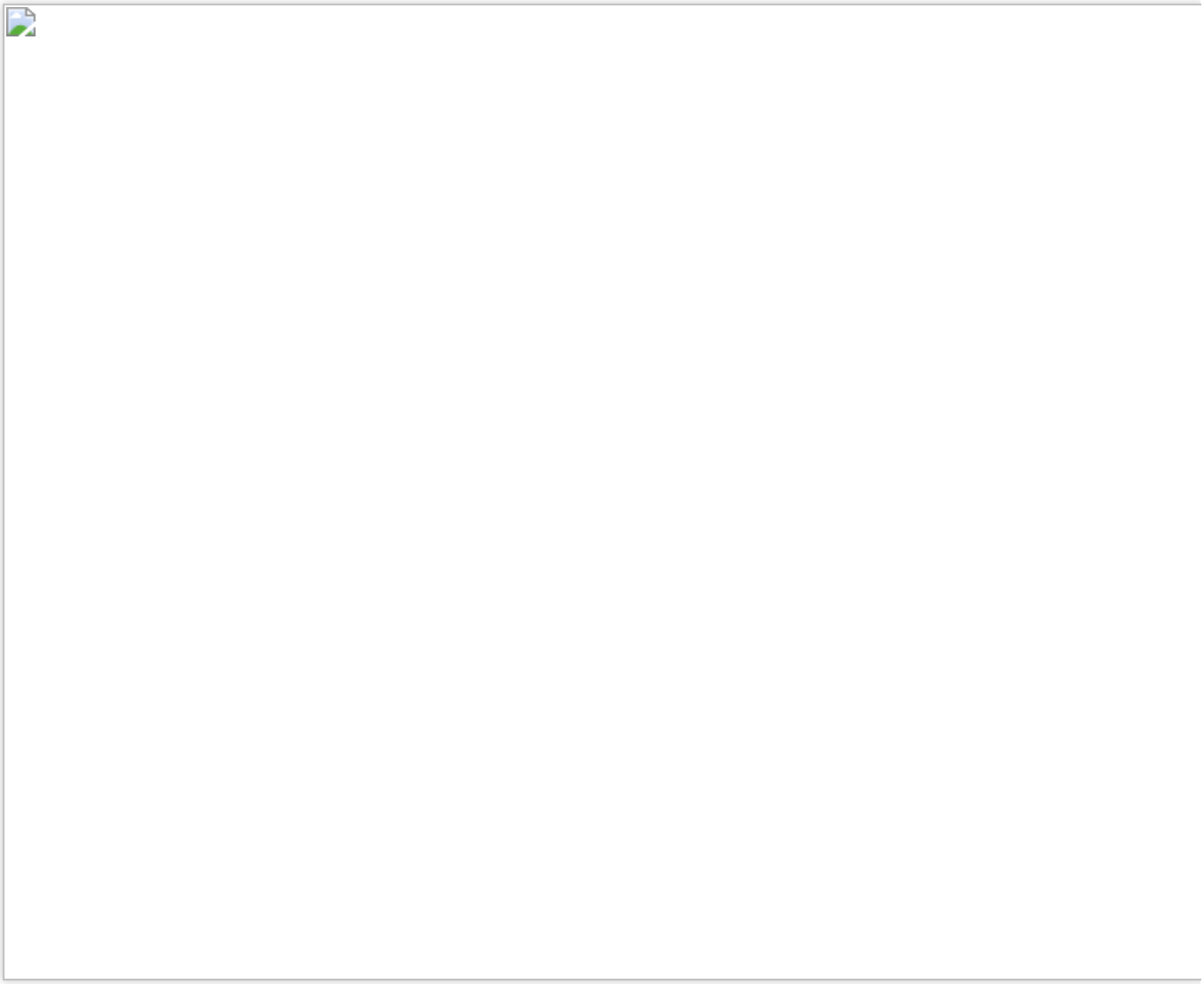
```
// 订阅topic
```

```
pullConsumer.subscribe(topic_name, "*");
// 启动消费者实例
pullConsumer.start();
try {
     System.out.printf("Consumer Started.%n");
     while (true) {
             // 拉取消息
             List<MessageExt> messageExts = pullConsumer.poll();
             System.out.printf("%s%n", messageExts);
     }
} finally {
     pullConsumer.shutdown();
}
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4：查看消费详情

登录 TDMQ 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 Demo 或 社区文档。

操作指南

集群管理

新建集群

最近更新时间：2024-05-14 16:47:50

升配集群

最近更新时间：2024-05-14 16:49:49

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节点规格、节点数量和存储规格。

说明

当前仅专享集群和通用集群支持升级集群规格：专享集群支持新增节点数量，升级节点的规格及存储的规格；通用集群支持升级 TPS 范围和存储规格。

集群在升配的过程中，腾讯云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

1. 在**集群管理**列表页，单击操作列的**升配**。
2. 选择目标节点规格后，单击**确认调整**。

专享集群

通用集群

目标节点数量：根据您的业务需求调整节点数量。

单节点存储规格：调整单节点存储后，新的存储规格对集群内所有的节点都生效。

Topic 个数：每个规格会提供一定额度的免费 Topic，如当前额度不满足您的要求，您可以自助添加 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。

目标 TPS 规格：根据您的业务需求调整 TPS。

Topic 个数：每个规格会提供一定额度的免费 Topic，如当前额度不满足您的要求，您可以自助添加 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。

Group 个数：默认Topic 和 Group 的比例为 1 比 10 。Group 限额不收取费用，您可以通过调整 Topic 的个数来进行调整。

降配集群

最近更新时间：2024-05-14 16:50:31

在购买专享集群或者通用集群后，您可以根据业务量和集群使用情况来调整专享集群的节点规格。

说明：

当前仅专享集群和通用集群支持降低集群规格：

专享集群：由于底层组件的限制，目前存储规格不支持降配。为了避免在降配时的数据丢失，目前仅支持修改集群的节点规格，暂不支持修改集群的节点数量。

通用集群：由于底层组件的限制，目前存储规格不支持降配。仅支持降低集群 TPS 规格。

操作步骤

1. 在**集群管理**列表页，单击操作列的**降配**。
2. 选择目标集群规格后，单击**确认调整**。


降配
降配后会导致集群负载能力下降；您可以通过控制台的监控和配置告警等方式来查看当前集群的运行情况。
当前配置
节点规格
存储规格
到期时间
基础型 | 2 节点
410GB
2024-08-18 20:03:34
目标节点规格
基础型
标准型
高阶I型
高阶II型
目标节点数
2
单节点存储规格
205 G
Topic 个数
504
个
当前集群的 Topic 免费限额为 500，额外购买 Topic 需要支付一定费用，价格请参考;计费说明
Group 个数
5040
个
降配后费用
当前费用仅供展示，因为涉及到折扣/代金券的计算等，实际金额以支付页/退款页面为准。
确认调整
取消

销毁/退还集群

最近更新时间：2024-05-14 16:51:16

用户不再需要 RocketMQ 集群时，可以销毁并释放该集群。

操作步骤

虚拟集群

专享集群&通用集群

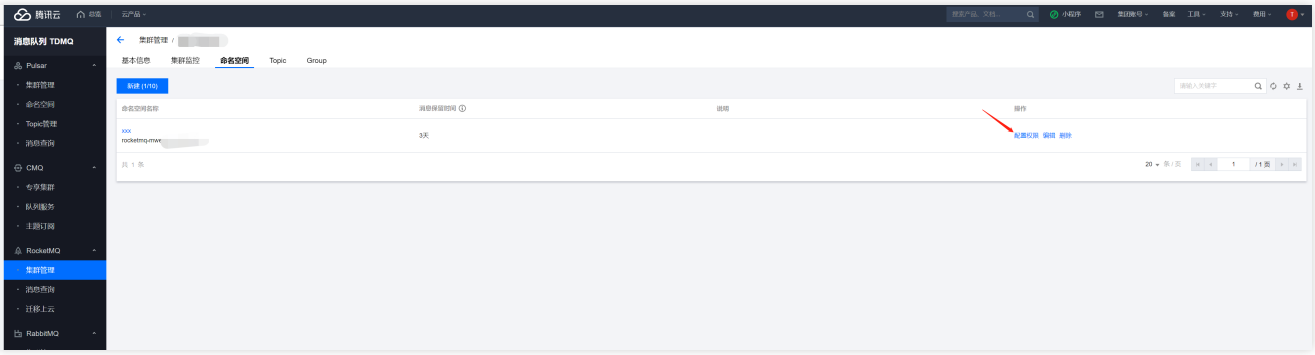
如果想删掉创建的虚拟集群，可以通过以下步骤操作：

1. 在**集群管理**列表页，单击操作列的**删除**。
2. 在删除的确认弹框中，单击**删除**，即可删除集群。 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。

说明：

为了避免用户的误操作造成集群内部数据（例如 Topic 和 Group 等），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，如命名空间，Topic 或者 Group 未删除，则无法删除集群。

在删除命名空间时，请先删除命名空间内的 Topic 和 Group，目前均已支持控制台批量删除。删除命名空间前，还需要删除命名空间的绑定角色。在**命名空间**列表页，单击**配置权限**。



进入权限配置页面后，单击**删除**后，解除两者的绑定关系，之后可以再次前往命名空间页，删除命名空间。

如果想删掉创建的专享/通用集群，可以通过以下步骤操作：

1. 在**集群管理**列表页，单击操作列的 **销毁/退还** 按钮。
2. 在 **退还集群** 页面，确认集群信息，如果当前集群存在消息堆积，会在当前页面进行提醒。

3. 确认完成后，进入 **下一步**。进入下一步后会展示退款明细和金额。

4. 确认无误后，集群将进入“隔离中” 的状态，“隔离中” 的集群无法进行消息的生产和消费，进入“隔离中”的状态超过7天后的集群，将会被自动销毁。

5. 对于处于“隔离中” 状态的集群，您如果要彻底删除该集群，您可以继续点击操作列的 **销毁** 按钮。

6. 如果您在集群的隔离期内希望恢复集群的使用，您可以在隔离期内点击操作列的 **恢复** 按钮，即可恢复使用期间的元数据和配置。

调整公网带宽

最近更新时间：2024-05-14 16:52:02

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。

说明：

当前仅专享集群和通用集群支持调整带宽。

操作步骤

公网带宽的调整有两个入口：

入口一：在集群管理列表页，单击操作列的 **更多 > 调整网络带宽**。

入口二：在集群基本信息页面，在网络模块的**公网访问带宽**处点击编辑按钮。

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。

公网带宽：选择要调整的公网带宽大小。

公网安全策略：填写允许访问的 IP，不设置安全策略默认禁止所有 IP 访问。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。

命名空间管理

最近更新时间：2023-03-14 15:22:45

操作场景

命名空间是 TDMQ RocketMQ 版中的一个资源管理概念。用户不同的业务场景一般都可以通过命名空间做隔离，并且针对不同的业务场景设置专门的配置，例如消息保留时间。不同命名空间之间的 Topic 相互隔离，订阅相互隔离，角色权限相互隔离。

本文档指导您使用消息队列 TDMQ RocketMQ 版时，创建多个命名空间，以便在同一个集群下将 TDMQ RocketMQ 版应用于不同的场景。

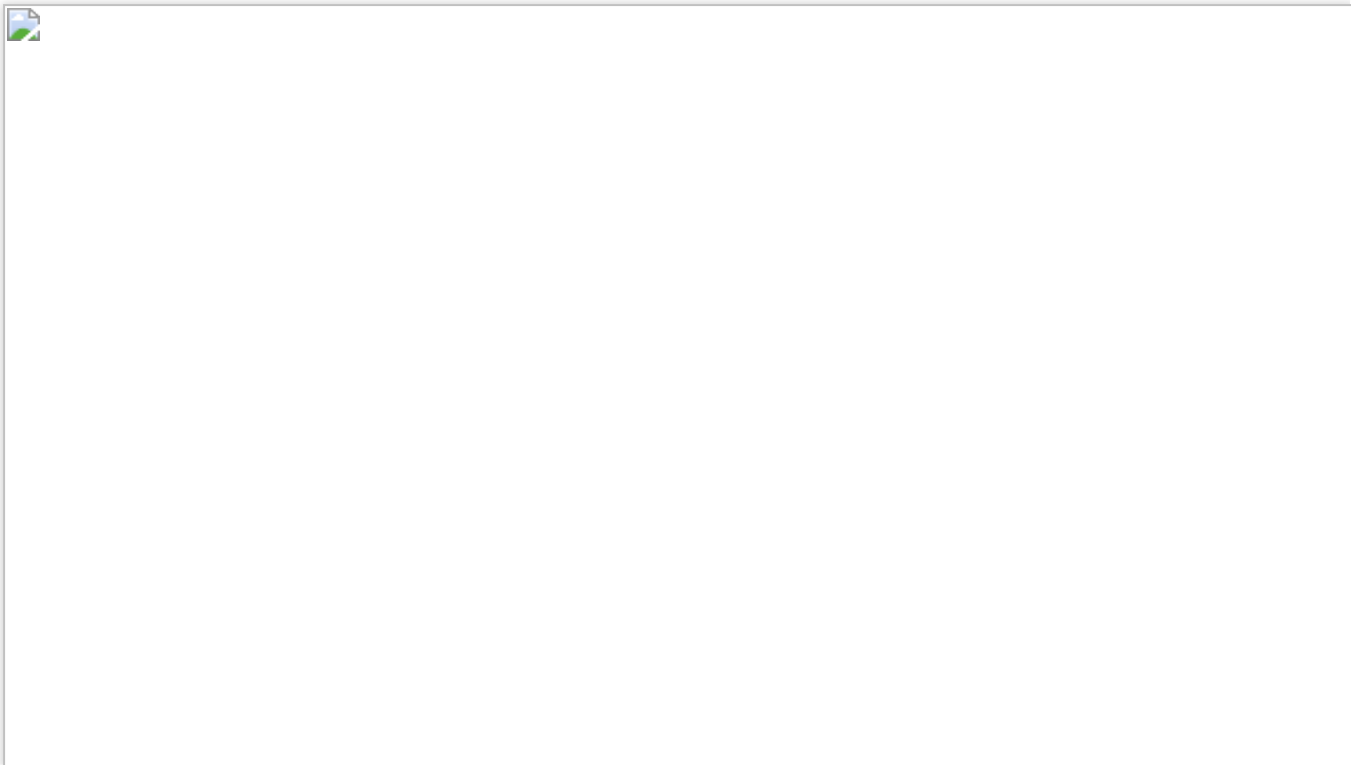
说明

同一个命名空间下的 Topic 和 Group 的名称唯一。

操作步骤

创建命名空间

1. 登录 TDMQ 控制台，选择地域后，单击目标集群的 “ID”，进入基本信息页面。
2. 在页面上方选择**命名空间**页签，单击**新建**进入创建命名空间页面。
3. 在新建命名空间对话框，设置命名空间的相关属性配置。



命名空间名称：设置命名空间的名称（创建后不可修改），3-32个字符，只能包含字母、数字及'_'
消息保留时间：设置未消费消息的保留时间，消息成功生产后，会持续保存一段时间，无论是否被消费，到时间则自动删除，范围：0秒-3天
命名空间描述：命名空间的备注说明
4. 单击**保存**完成所在集群命名空间的创建。

说明

每个集群下最多可以创建10个命名空间。
后续步骤：接下来就可以在该命名空间中 创建 Topic 进行消息的生产和消费。

获取接入地址

在**命名空间**列表页，在 **VPC 内网接入地址**和**公网接入地址**栏可获取命名空间的 TCP 协议接入地址，HTTP 协议的接入点请前往集群基本信息页查看。

修改命名空间

如果需要重新修改编辑，可以通过以下步骤操作：

1. 在**命名空间**列表页，单击操作列的**编辑**，进入编辑页面。
2. 修改消息 TTL、消息保留时间或说明，单击**保存**完成修改。

删除命名空间

如果想删掉创建的命名空间，可以通过以下步骤操作：

1. 在**命名空间**列表页，单击操作列的**删除**。
2. 在删除的确认弹框中，单击**确认**，即可删除命名空间。

注意

当命名空间内有 Topic 时，该命名空间不可被删除。

Topic 管理

最近更新时间：2024-05-14 16:42:06

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。

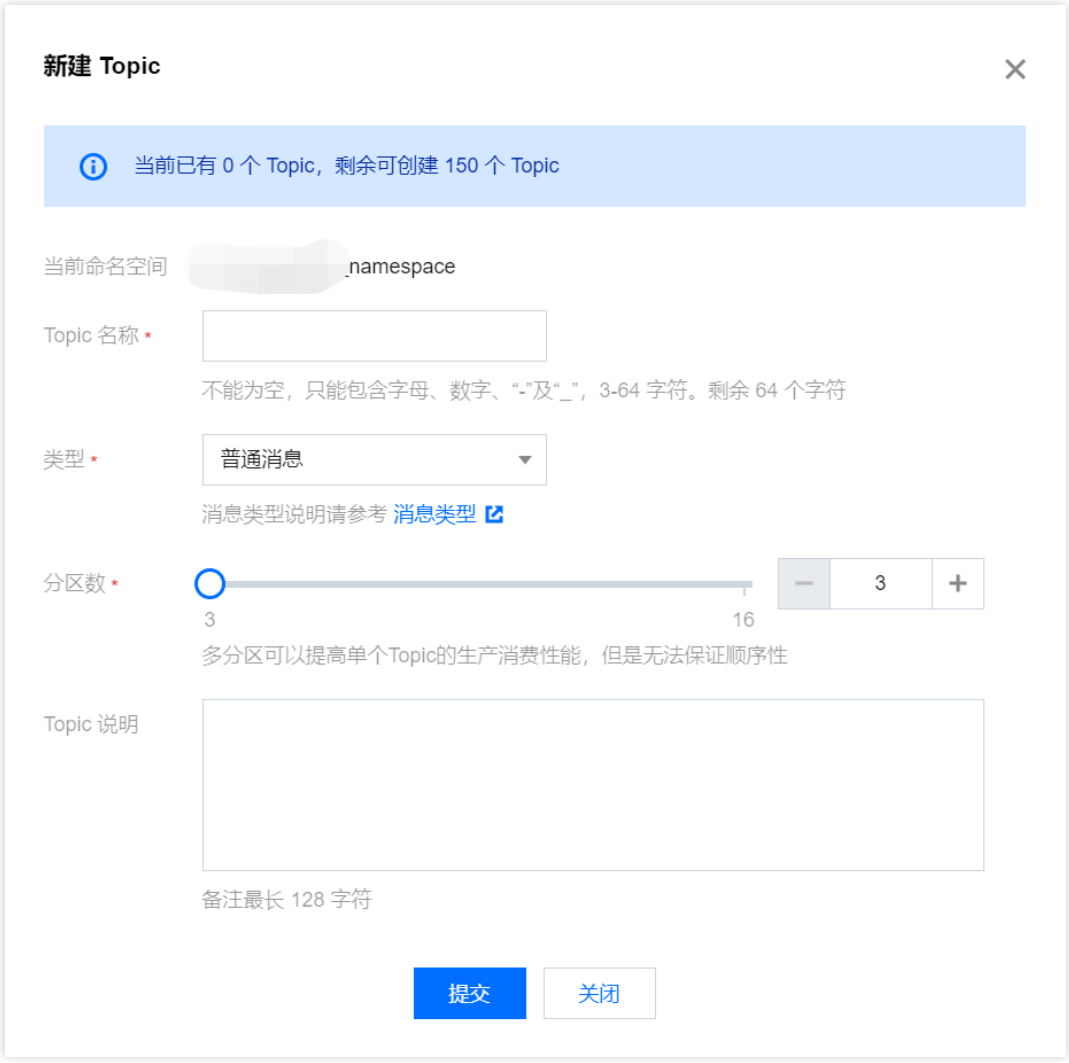
前提条件

已创建好对应的命名空间。

操作步骤

创建 Topic

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 **Topic** 页签，选择命名空间后，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）。
分区数：选择分区数量，最大支持16分区。多分区可以提高单 Topic 的生产消费性能，但是无法保证顺序性。
说明：填写 Topic 的说明信息
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

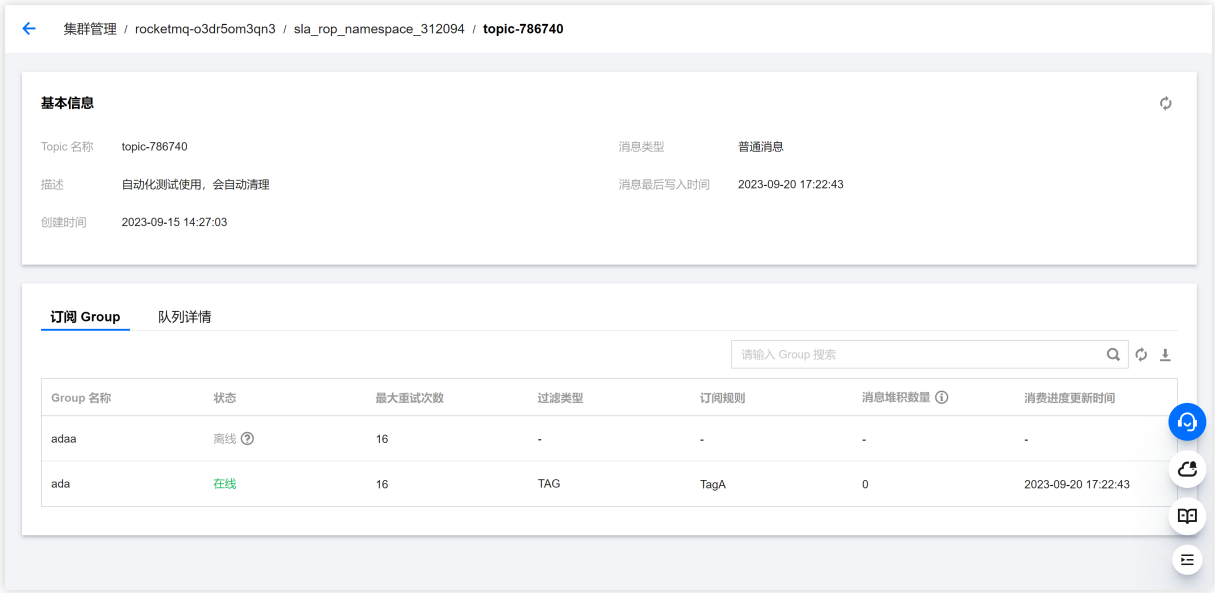
TDMQ RocketMQ 版控制台支持手动发送消息，在控制台进行相应的操作即可实现消息发送给指定的 Topic。
1. 在 Topic 列表中，单击目标Topic 操作栏的**发送测试消息**。
2. 在弹窗中输入消息 Key，消息 Tag 和消息内容，单击**提交**。

查看 Topic 详情

您可以进入 Topic 详情页查看当前 Topic 的详细信息。

订阅的 Group

1. 在 Topic 列表中，单击 Topic 名称或者操作列的 **Topic 详情** 进入Topic详情页。
2. 在 Topic 详情页，您可以查看订阅了该 Topic 的 Group，以及订阅规则等信息。

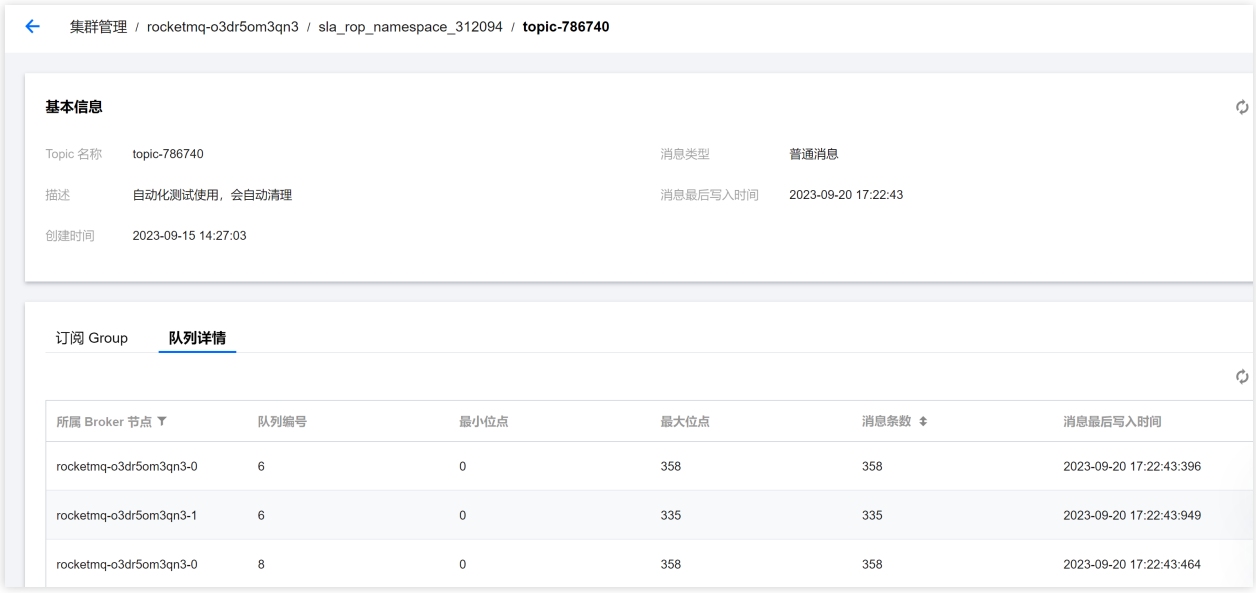


Topic 队列详情

说明：

仅专享集群支持查看 Topic 队列详情。

在 Topic 详情页，可以查看当前 Topic 下分布在各个 Broker 节点上的队列情况。



查询 Topic

您可以在 Topic 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 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 参数进行编辑。
3. 单击**提交**即完成对 Topic 的编辑。

删除 Topic

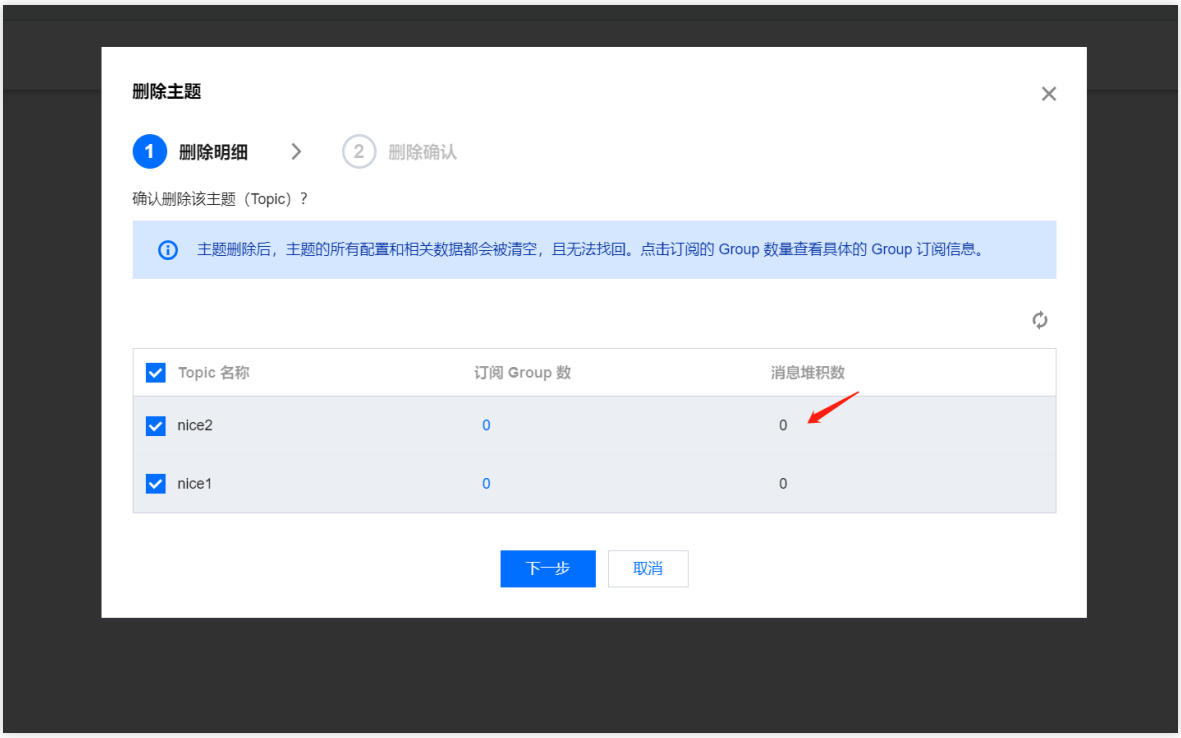
批量删除：在 Topic 列表中，勾选所有需要删除的 Topic，单击左上角的**批量删除**，在弹出的提示框中，单击**删除**，完成删除。

单个删除：在 Topic 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。

在删除单个或者批量删除 Topic 时，控制台会对当前 Topic 的数据进行检验，如下图所示。



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，您可以击 Topic 列表页右上角的

按钮，将 Topic 数据导入到指定的命名空间下。

Group 管理

最近更新时间：2023-05-16 11:07:52

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。

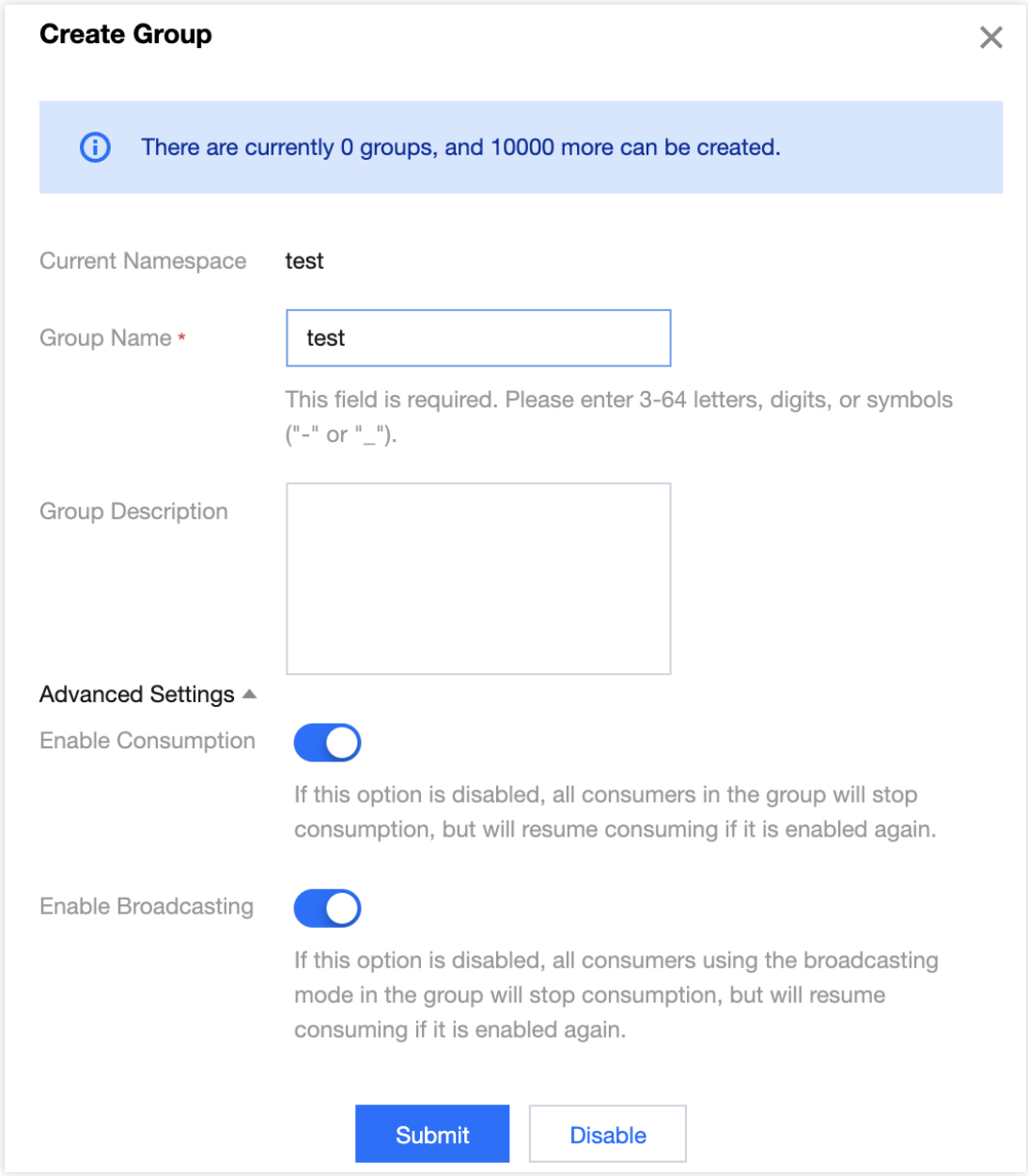
前提条件

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

1. 登录 TDMQ 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

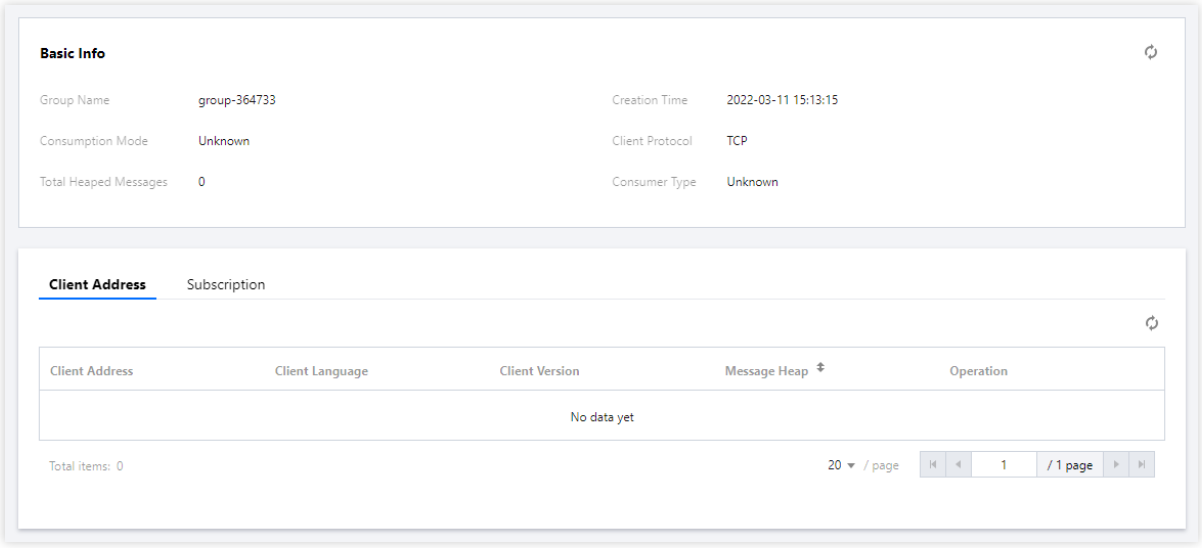
4. 单击**提交**，完成 Group 创建。

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，单击操作列的**消费者详情**，进入订阅列表。
2. 展开列表后可以看到 Group 的基本信息，客户端连接和订阅关系。



基本信息

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。

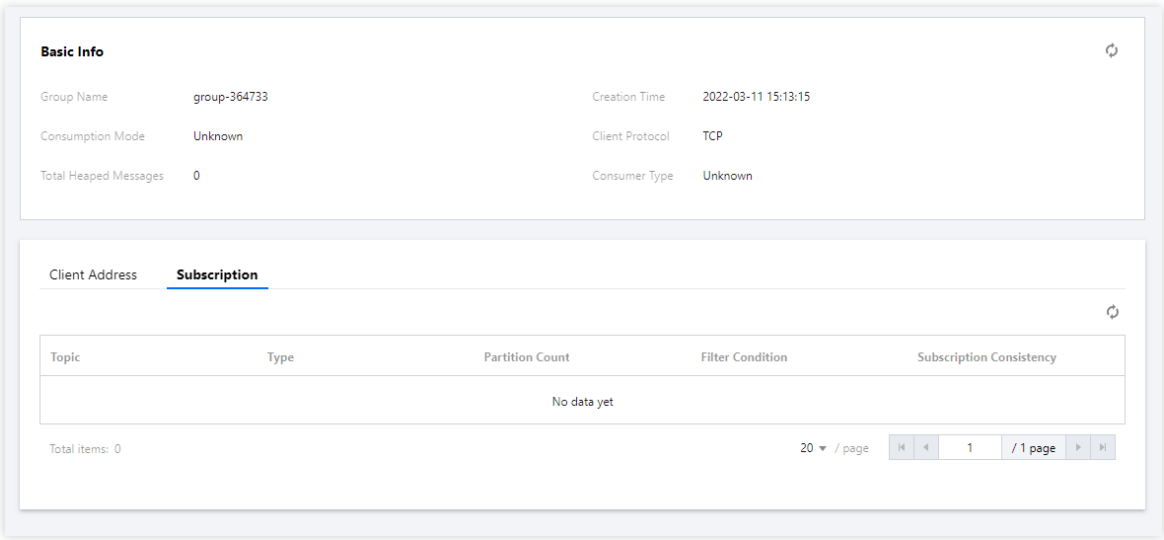
广播消费：当使用广播消费模式时，每条消息会被推送给集群内所有注册过的消费者，保证消息至少被每个消费者消费一次。

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 或 PASSIVELY

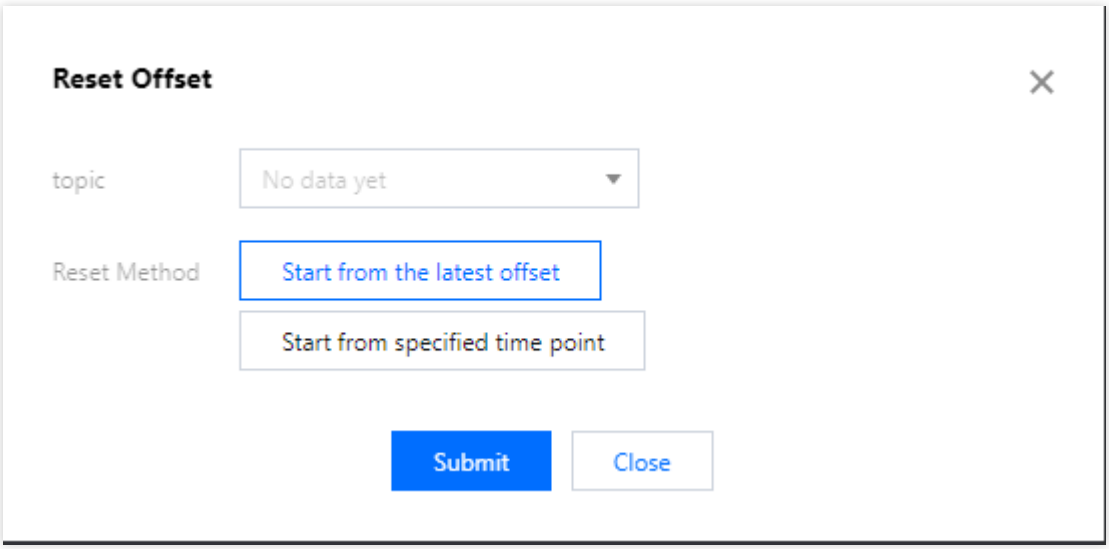
客户端连接：连接的客户端列表，单击客户端操作栏的**查看详情**可查看消费者详情。



订阅关系：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 offset**。
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**提交**，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**提交**，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。

角色与鉴权

最近更新时间：2023-05-16 10:39:23

名词解释

角色（role）：TDMQ 的“角色”是 TDMQ 内专有的概念，区别于腾讯云的“角色”，是用户自行在 TDMQ 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命名空间下的生产和消费权限。
密钥（token）：TDMQ 的“密钥”是一种鉴权工具，用户可以通过在客户端中添加密钥来访问 TDMQ 进行消息的生产消费。密钥和角色一一对应，每种角色都有其对应的唯一密钥。
TDMQ RocketMQ 版继承了这些概念，在 TDMQ RocketMQ 版中，开源 client 中定义的 `ACL_ACCESS_KEY` 对应 TDMQ 的密钥（token）。

使用场景

用户需要安全地使用 TDMQ 进行消息的生产消费。
用户需要对不同的命名空间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命名空间中生产消息的权限，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

新增角色

1. 登录 TDMQ 控制台，在左侧导航栏单击 角色管理，进入角色管理页面。
2. 在角色管理页面顶部选择需要设置的 RocketMQ 集群，单击**新建**进入新建角色页面。
3. 在新建角色页面，填写角色名称和说明：
角色：最长为32个字符，支持数字、大小写字母和分隔符（"_","-"）。
说明（选填）：不得超过100个字符。
4. 单击**提交**，完成当前集群命名空间的创建。

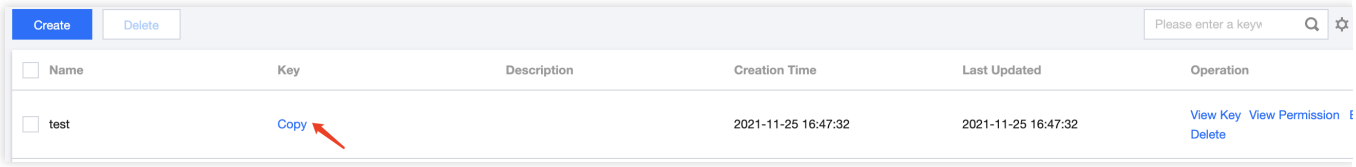
角色授权

1. 在 TDMQ 控制台的 角色管理 中，找到新建的角色，通过以下任一种方式复制角色密钥：

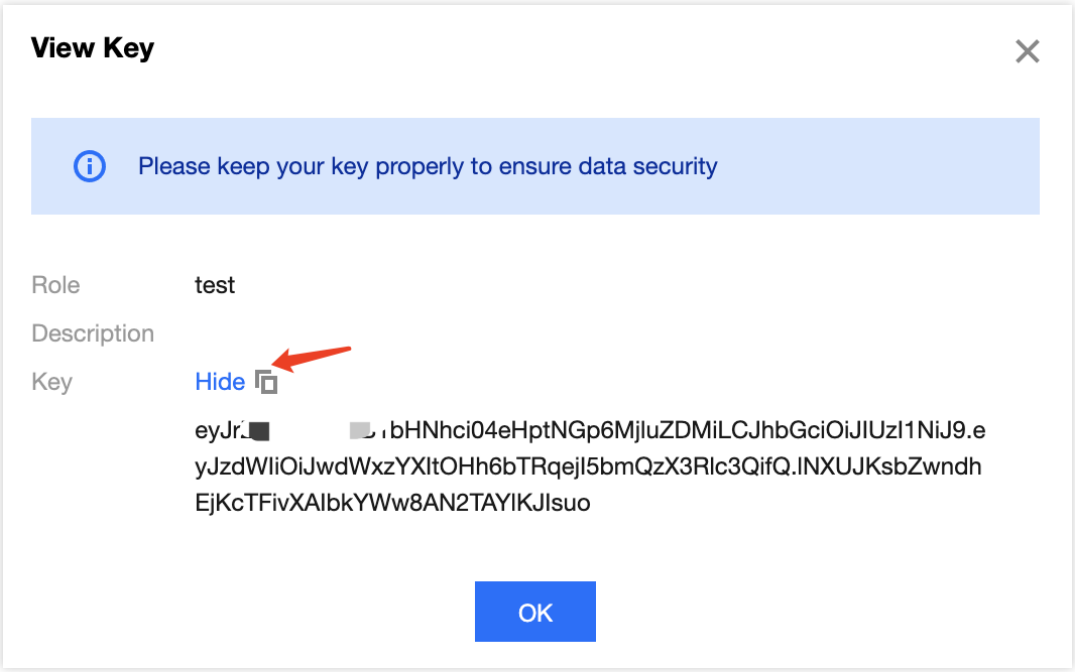
方式一：密钥列复制

方式二：操作列查看并复制

单击密钥列的**复制**。



单击操作列的**查看密钥**，在查看密钥弹框中单击复制图标。

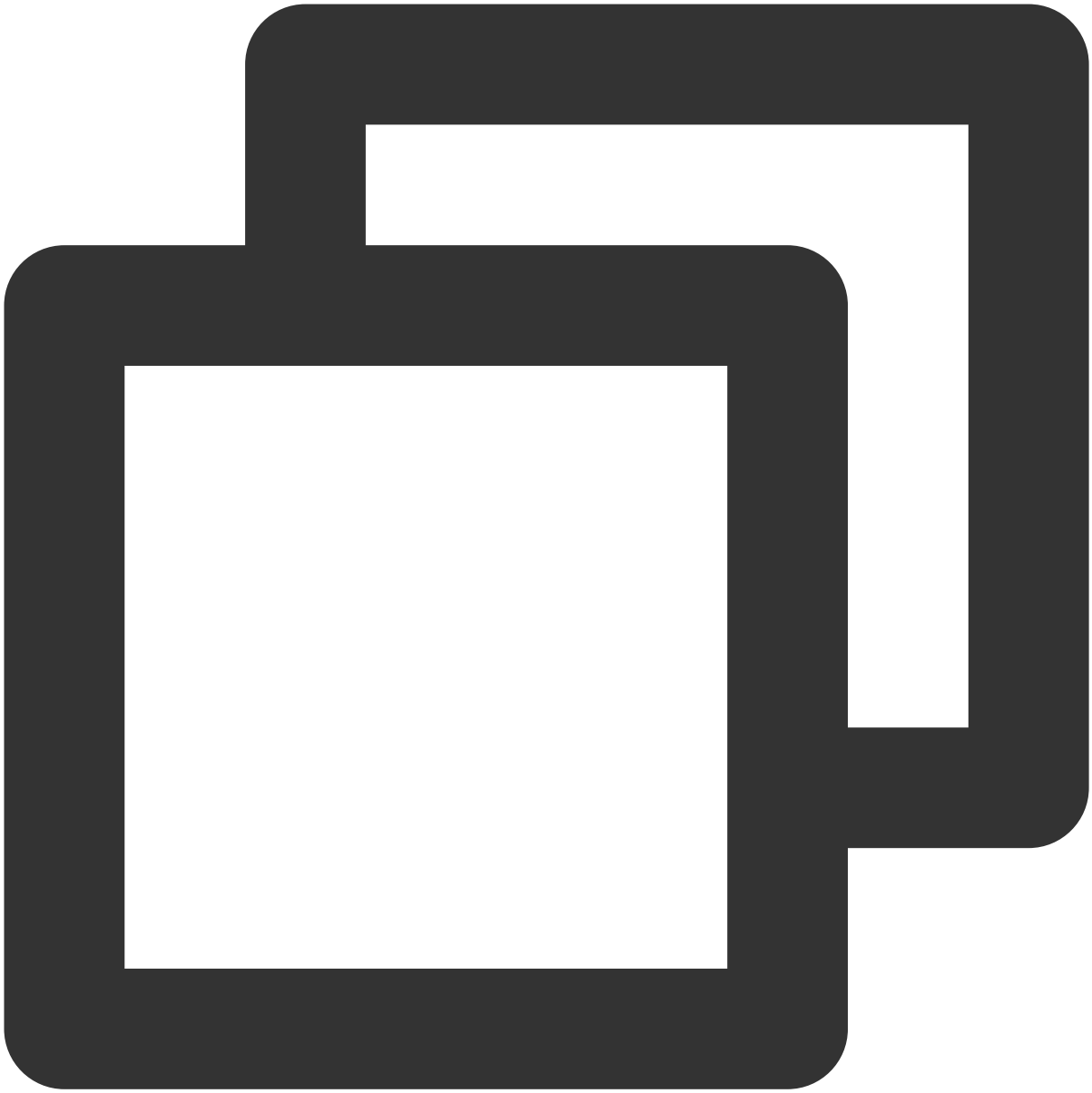


注意

密钥泄露很可能导致您的数据泄露，请妥善保管您的密钥。

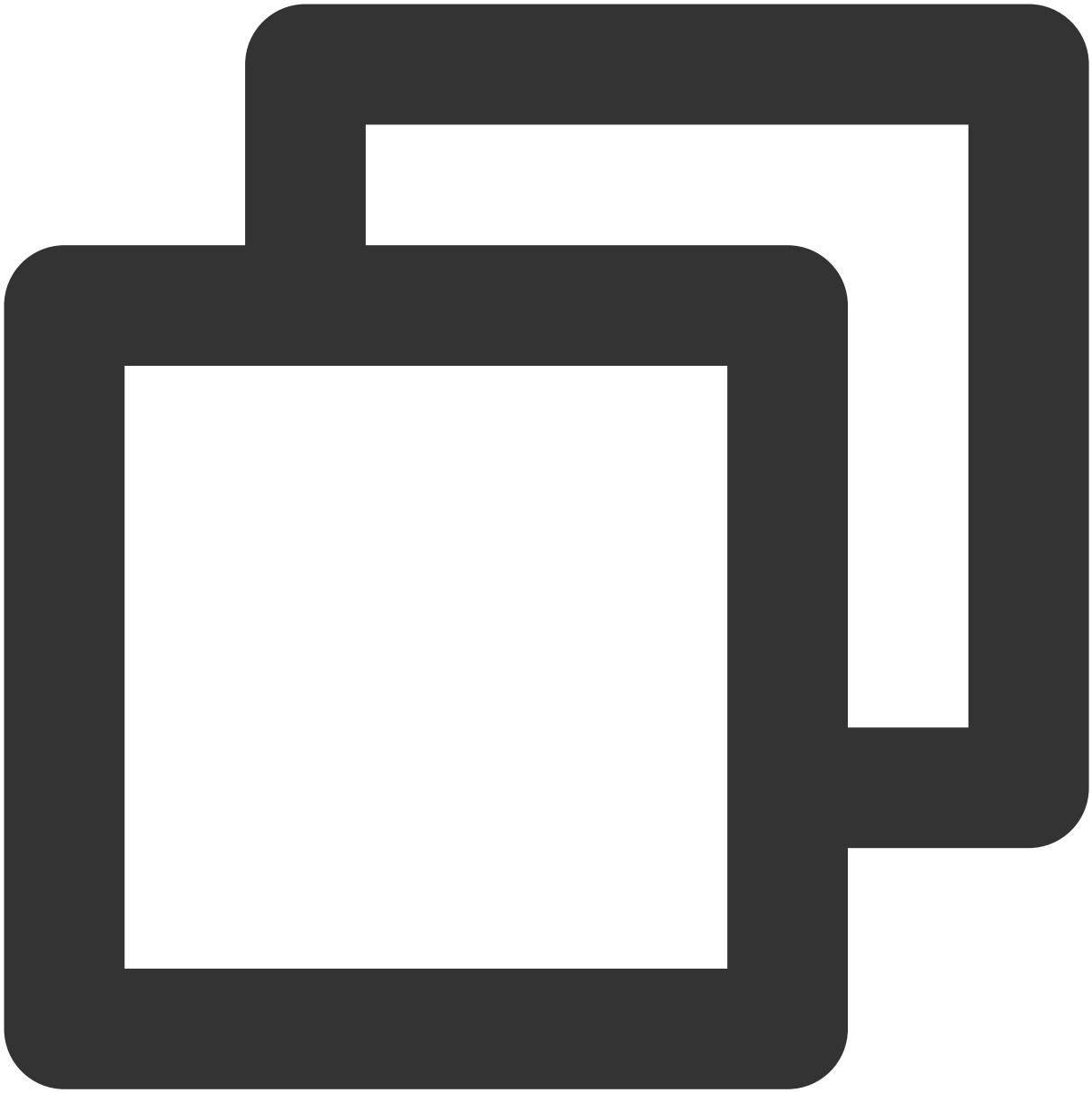
2. 将复制的角色密钥添加到客户端的参数中。如何在客户端代码中添加密钥参数请参考 RocketMQ 的 代码示例。以下给出一种推荐的方式。

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 = "bigdata"; //使用上图页面的角色名称即可
```

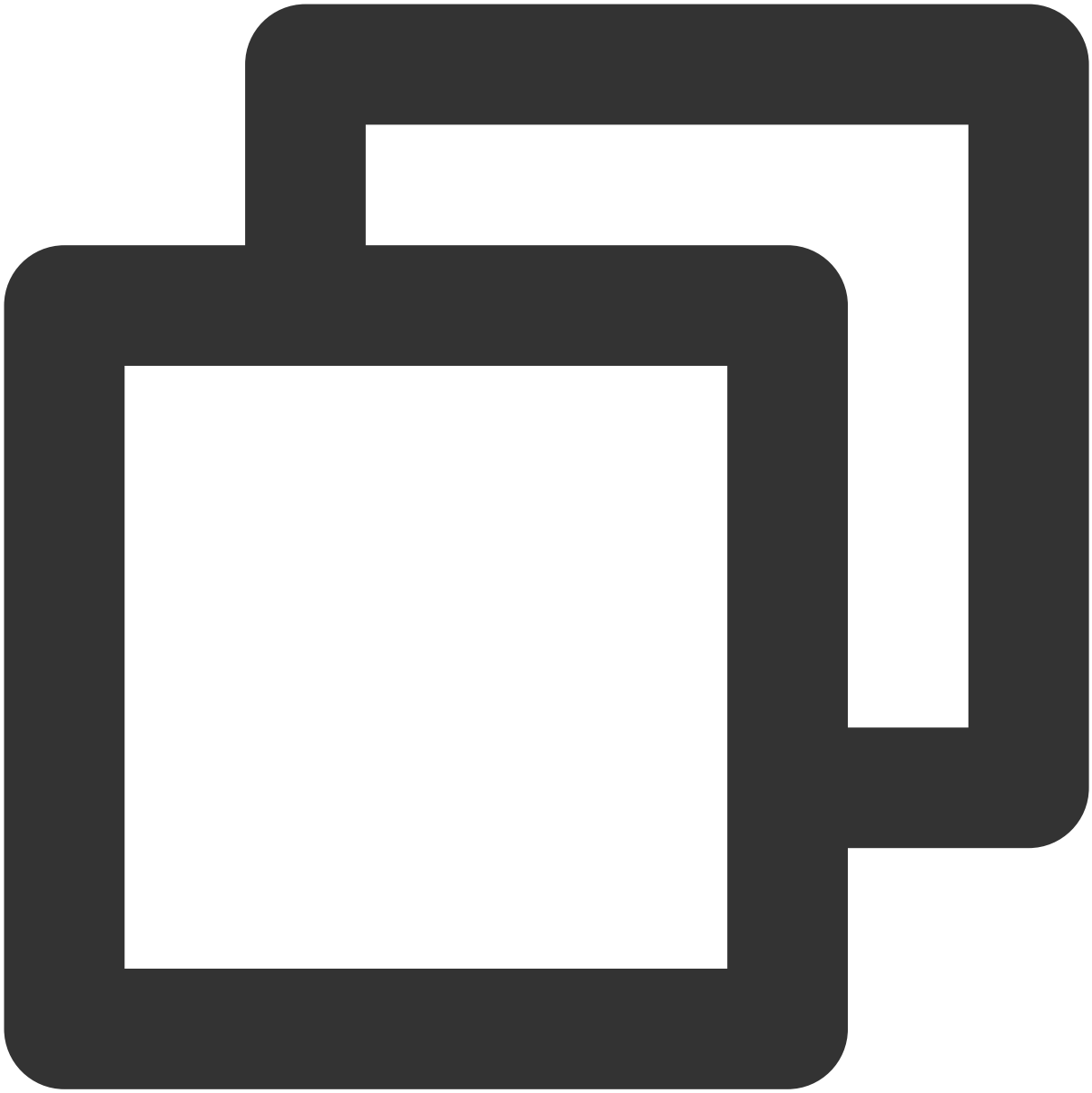
2.2 声明一个静态函数，用于载入一个 RocketMQ Client 的 `RPCHook` 对象



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

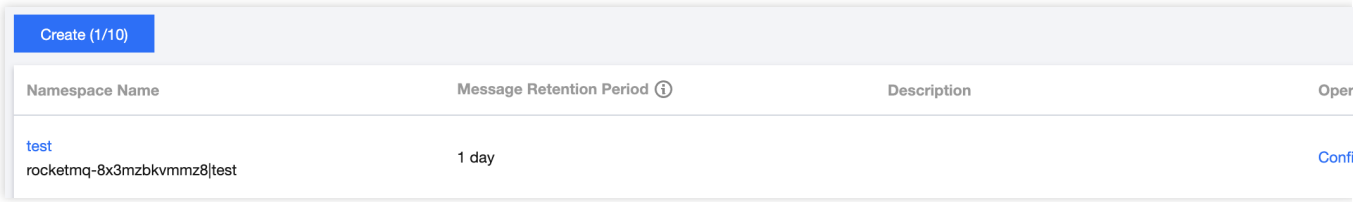
2.3 在创建 RocketMQ `producer` 、 `pushConsumer` 或 `pullConsumer` 的时候，引入 `RPCHook` 对象

以下为创建一个 `producer` 的代码示例：

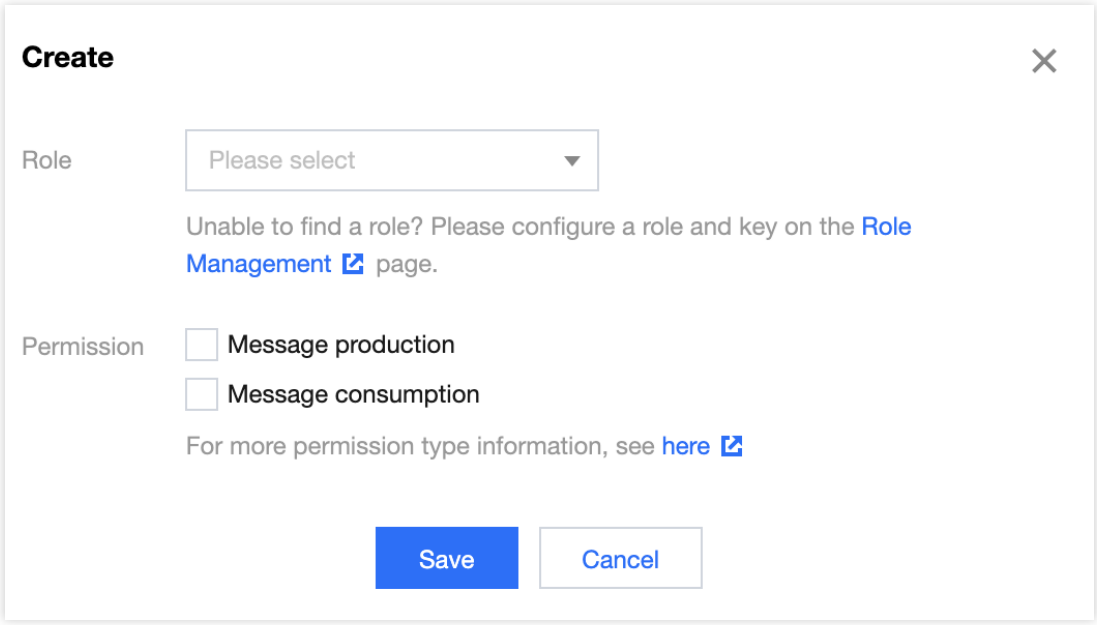


```
DefaultMQProducer producer = new DefaultMQProducer("rocketmq-mw***|namespace", "Pro
```

3. 在 TDMQ RocketMQ 控制台选择之前设定了角色的集群后，切换到**命名空间**，选择需要配置生产消费权限的一个命名空间，单击操作列的**配置权限**。



4. 单击**添加角色**，在下拉列表中找到刚刚新创建的角色，勾选上需要的权限，单击**保存**。



5. 检查权限是否生效。

您可以运行配置好的客户端访问对应命名空间中的 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 TDMQ RocketMQ 控制台的**命名空间**中，找到需要配置生产消费权限的一个命名空间，单击操作列的**配置权限**，进入配置权限列表。
2. 在配置权限列表中，找到需要编辑权限的角色，单击操作列的**编辑**。
3. 在编辑的弹框中，修改权限信息后，单击**保存**。

删除权限

注意

删除权限为危险操作，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再进行此项操作，否则可能会出现客户端无法生产消费而导致的异常。

当角色还有配置在各命名空间中的权限时，不可删除。

1. 在 TDMQ RocketMQ 控制台的**命名空间**中，找到需要配置生产消费权限的一个命名空间，单击操作列的**配置权限**，进入配置权限列表。

2. 在配置权限列表中，找到需要删除权限的角色，单击操作列的**删除**。
3. 在删除的弹框中，单击**确认**，即可删除该权限。

访问管理 CAM
主账号获取访问授权

最近更新时间：2023-09-12 16:44:43

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

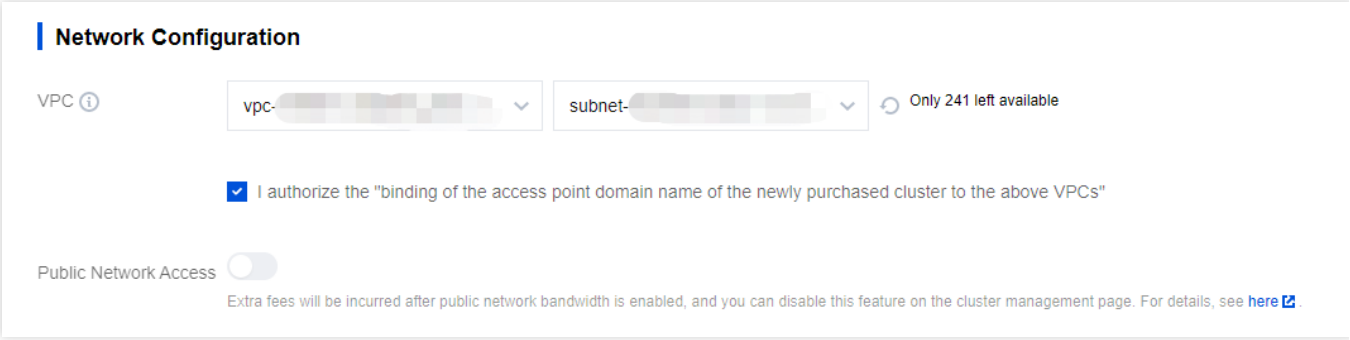
已注册腾讯云账号

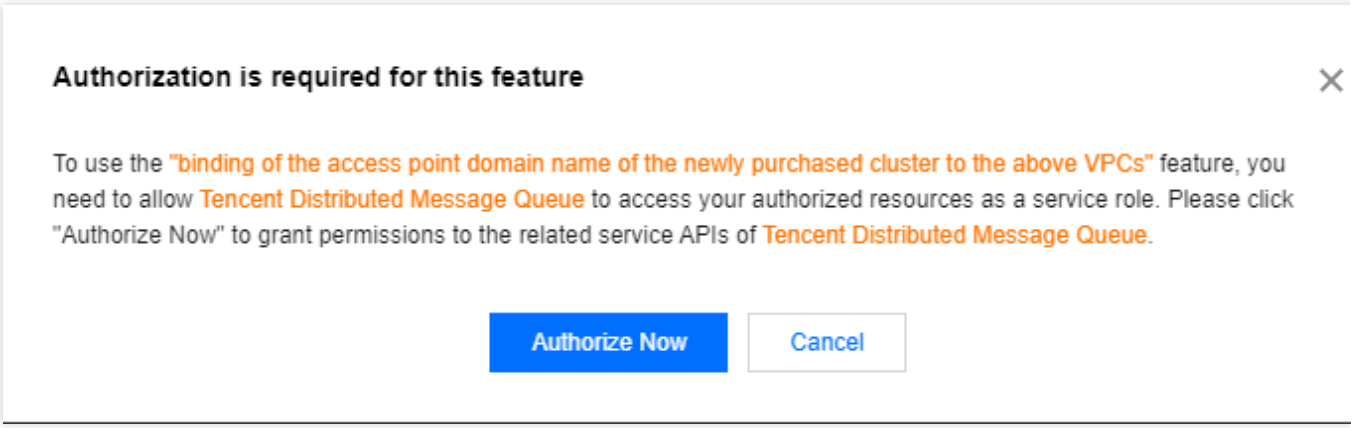
说明：

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。

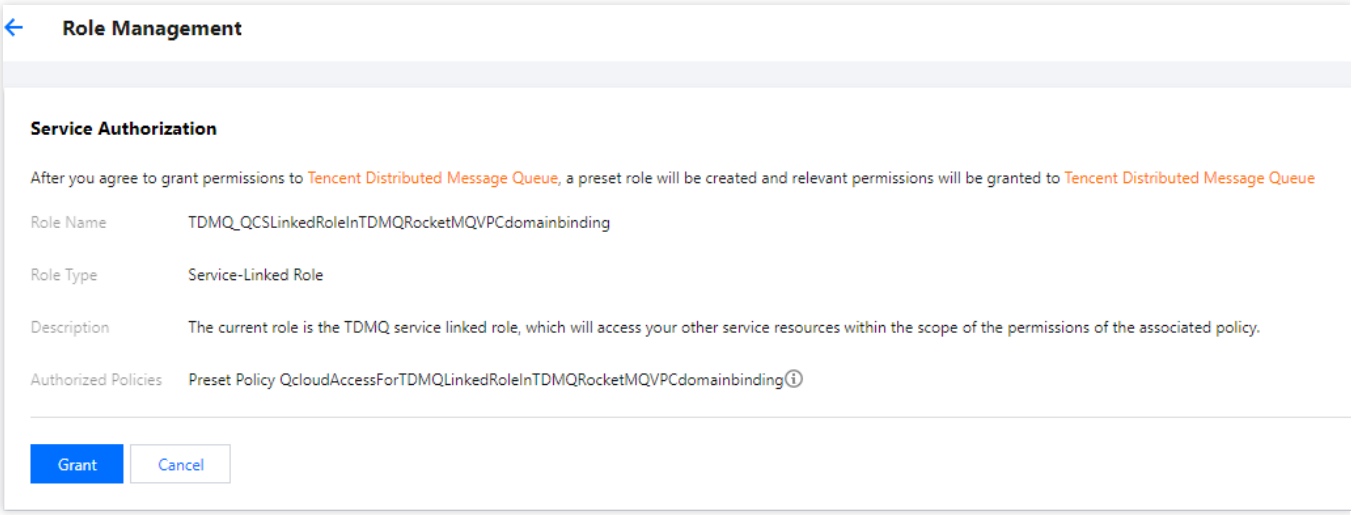
操作步骤

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. 点击**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：2023-10-19 10:45:22

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

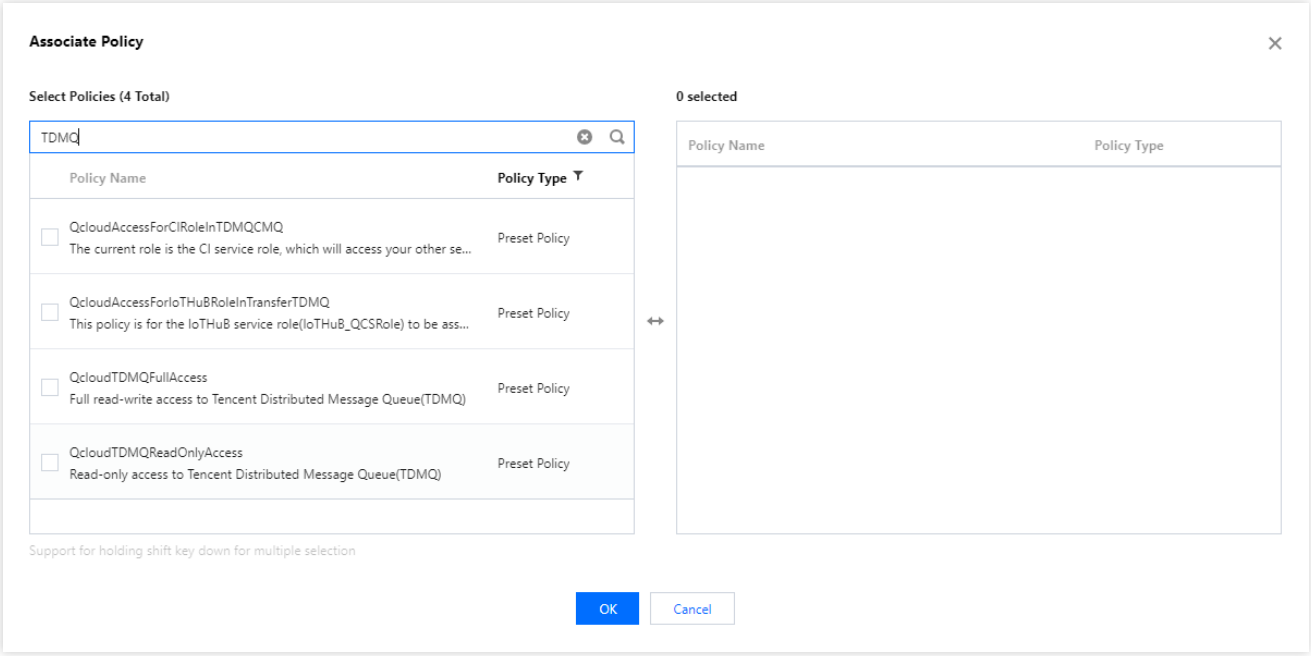
主账号：拥有腾讯云所有资源，可以任意访问其任何资源。
子账号：包括子用户和协作者。
子用户： 由主账号创建，完全归属于创建该子用户的主账号。
协作者：本身拥有主账号身份，被添加作为当前主账号的协作者，则为当前主账号的子账号之一，可切换回主账号身份。
身份凭证：包括登录凭证和访问证书两种，**登录凭证**指用户登录名和密码，**访问证书**指云 API 密钥（SecretId 和 SecretKey）。

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

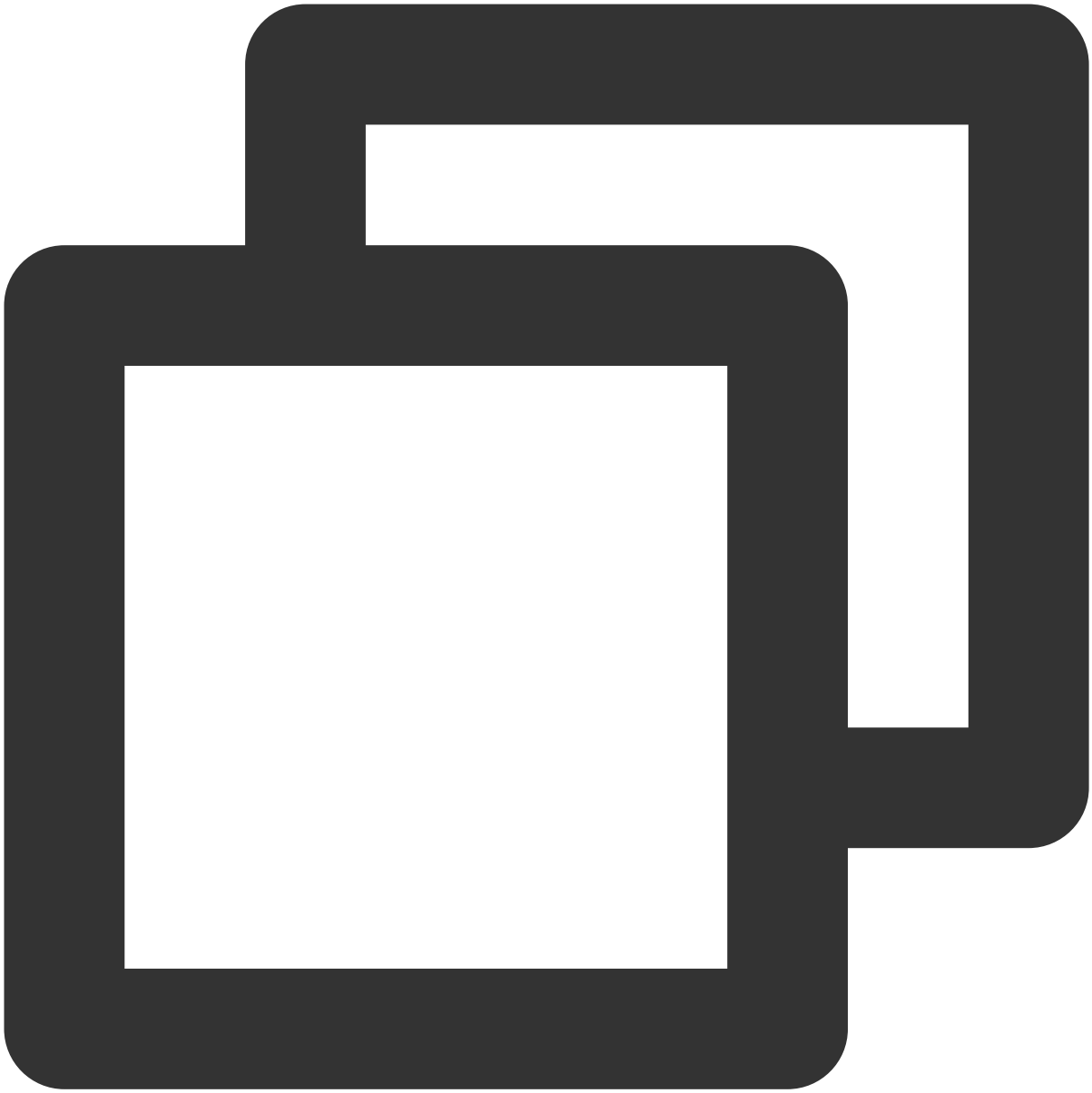
为了保证子账号能够顺利使用 RocketMQ，主账号需要对子账户进行授权。
主账号登录 访问管理控制台，在子账号列表中找到对应的子账号，单击操作列的**授权**。
RocketMQ 为子账号提供了两种预设策略：QcloudTDMQReadOnlyAccess 和 QcloudTDMQFullAccess，前者仅能查看控制台的相关信息，后者可以在产品控制台进行读写等相关操作。



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：

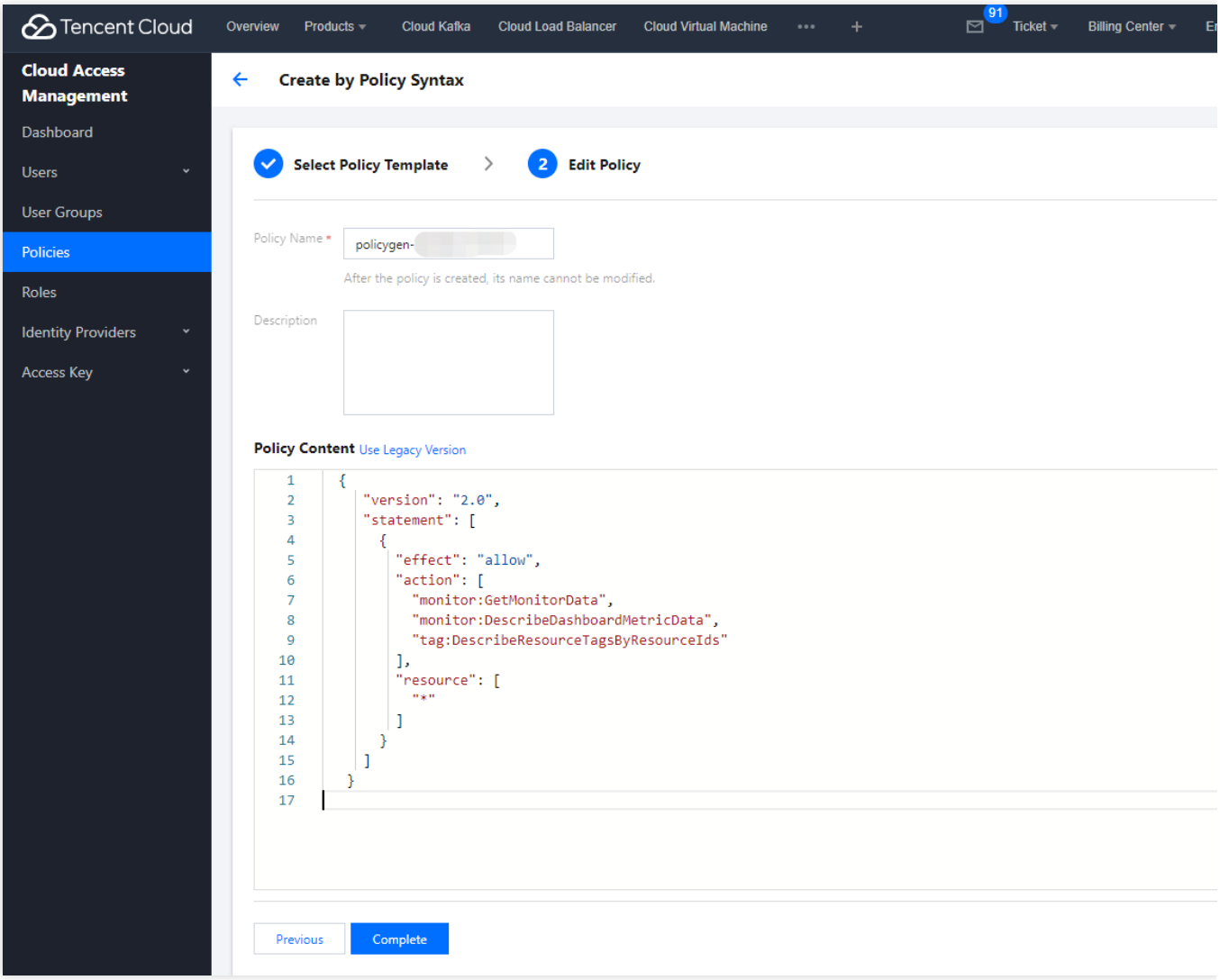
云产品	接口名	接口作用	对应在 RocketMQ 中的作用
腾讯云可观测平台（Monitor）	GetMonitorData	查询指标监控数据	查看控制台展示的相应监控指标
腾讯云可观测平台（Monitor）	DescribeDashboardMetricData	查询指标监控数据	查看控制台展示的相应监控指标
资源标签（Tags）	DescribeResourceTagsByResourceIds	查询资源标签	查看集群的资源标签

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：

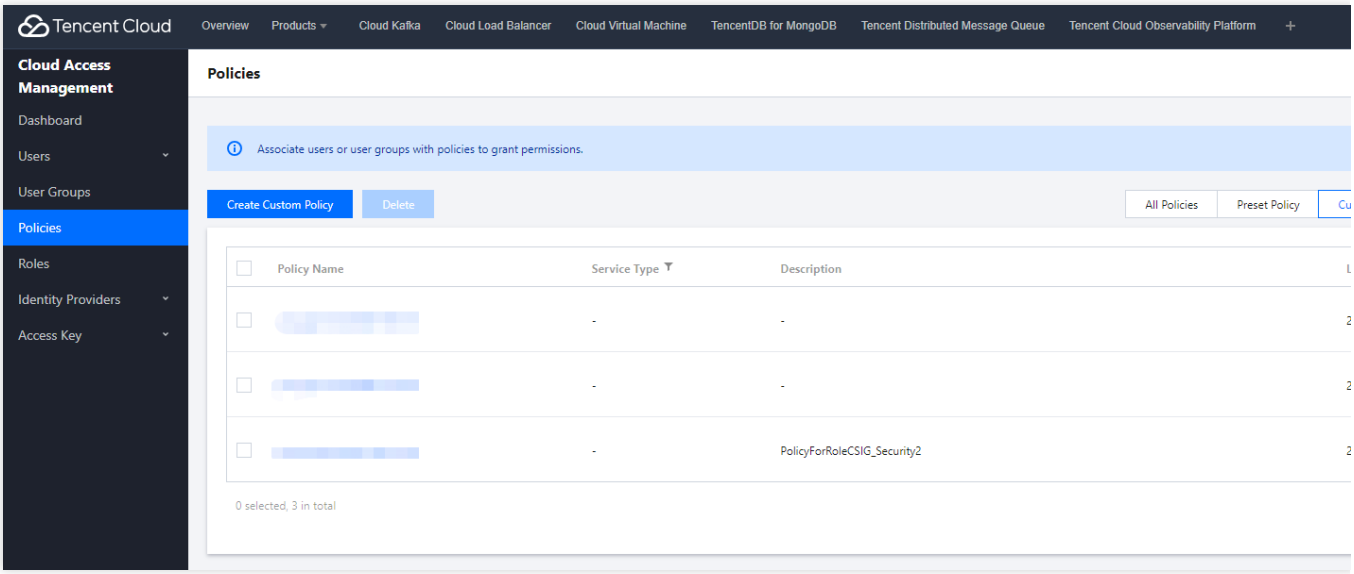


```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "monitor:GetMonitorData",
        "monitor:DescribeDashboardMetricData",
        "tag:DescribeResourceTagsByResourceIds"
      ],
      "resource": [
```

```
            "*"
        ]
    }
  ]
}
```



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

为子账号授权的相关操作可参见以下文档：

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：2023-09-12 16:48:58

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。

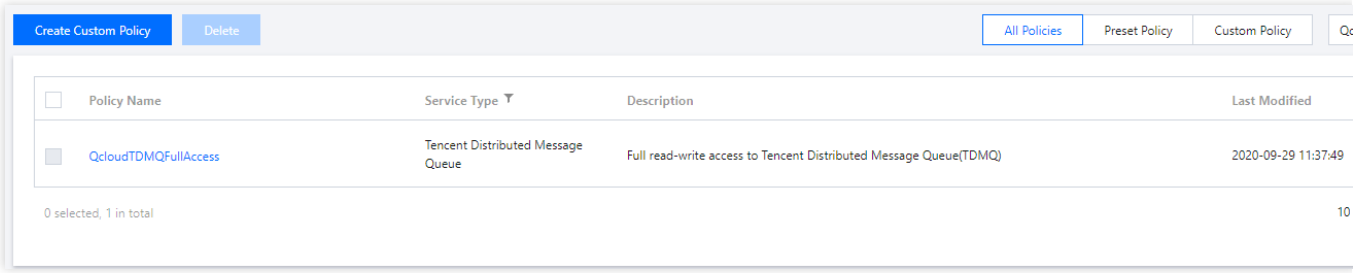
操作步骤

授予全量读写权限

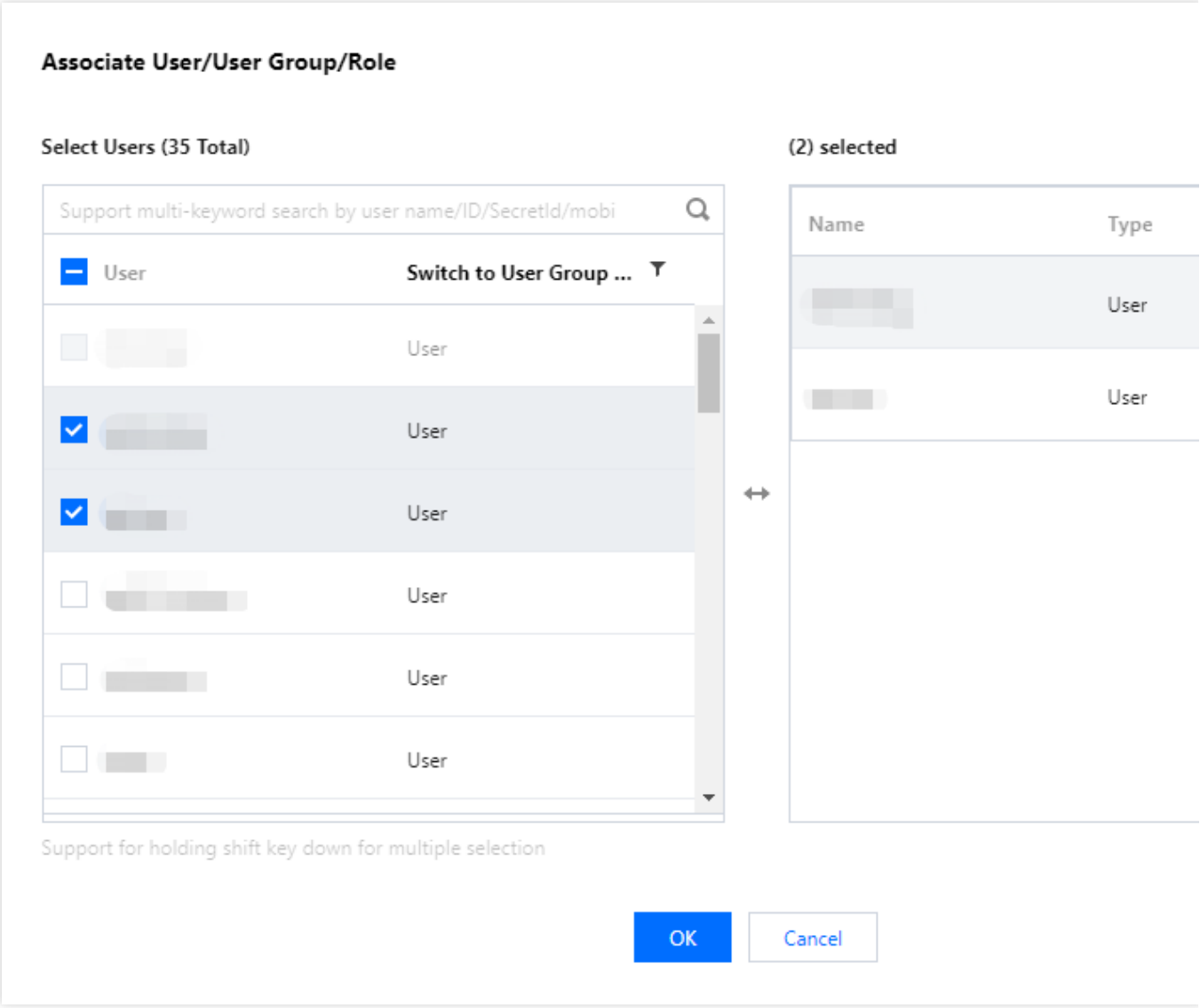
说明

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**。

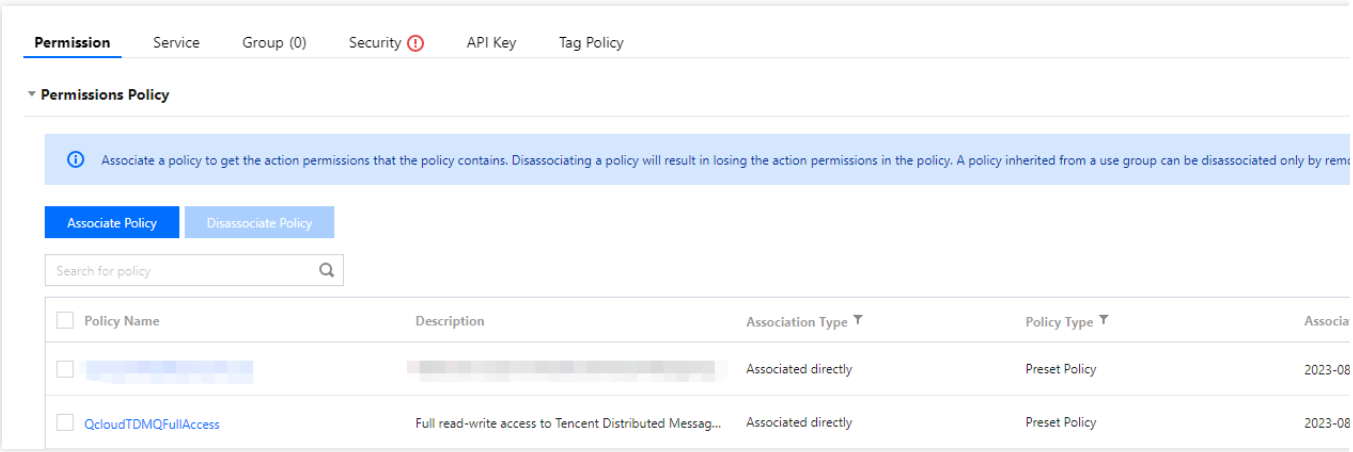
1. 使用主账号登录 访问管理控制台。
2. 在左侧导航栏，单击**策略**，进入策略管理列表页。
3. 在右侧搜索栏中，输入 **QcloudTDMQFullAccess** 进行搜索。



4. 在搜索结果中，单击 **QcloudTDMQFullAccess** 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。

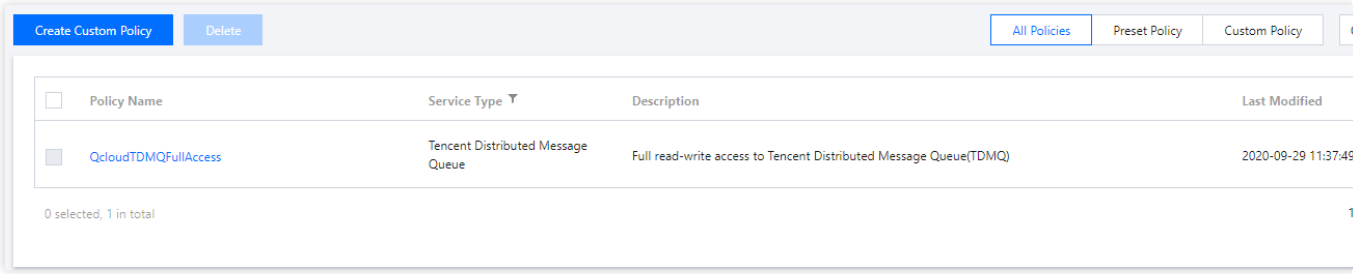


授予只读权限

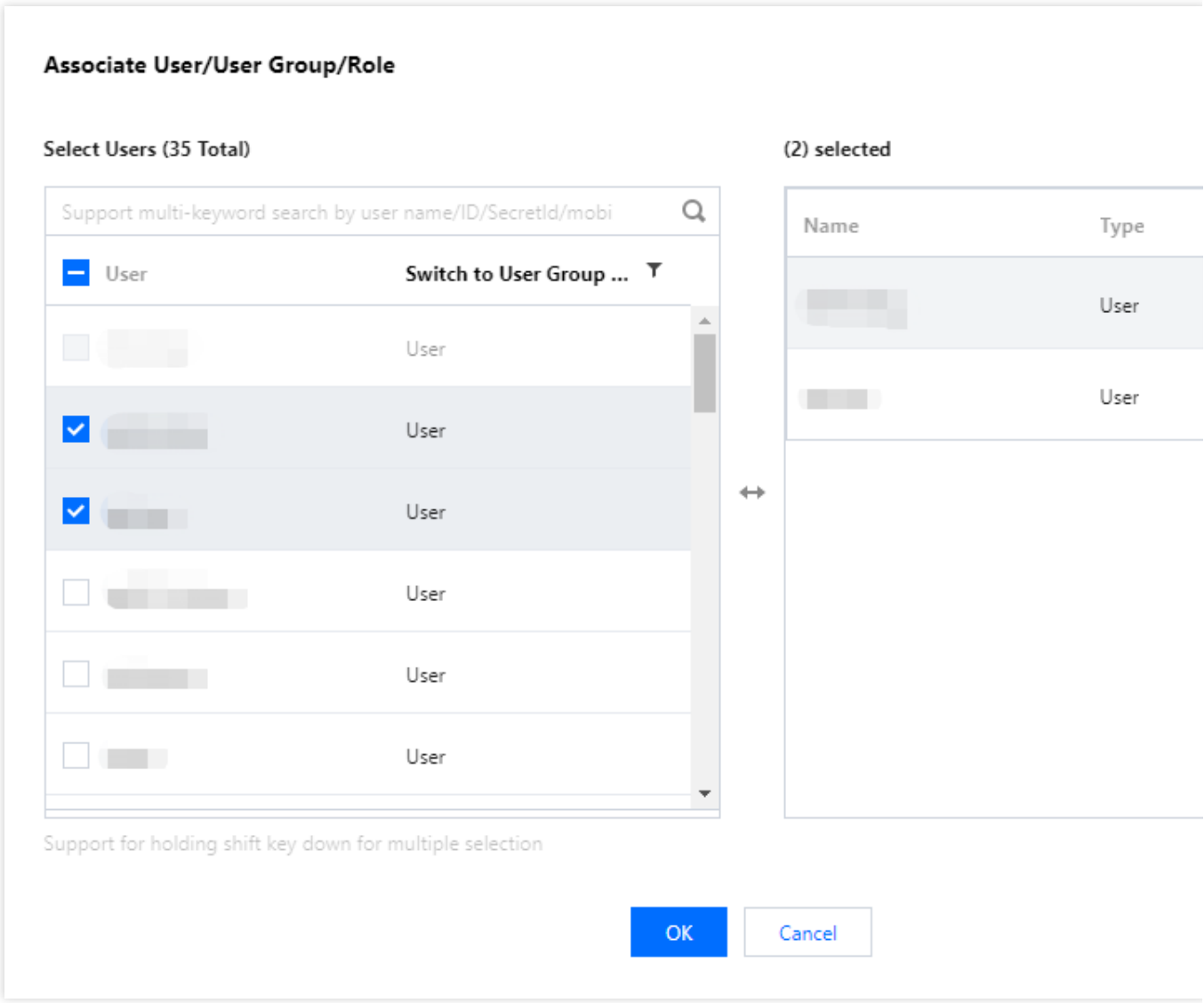
说明

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**。

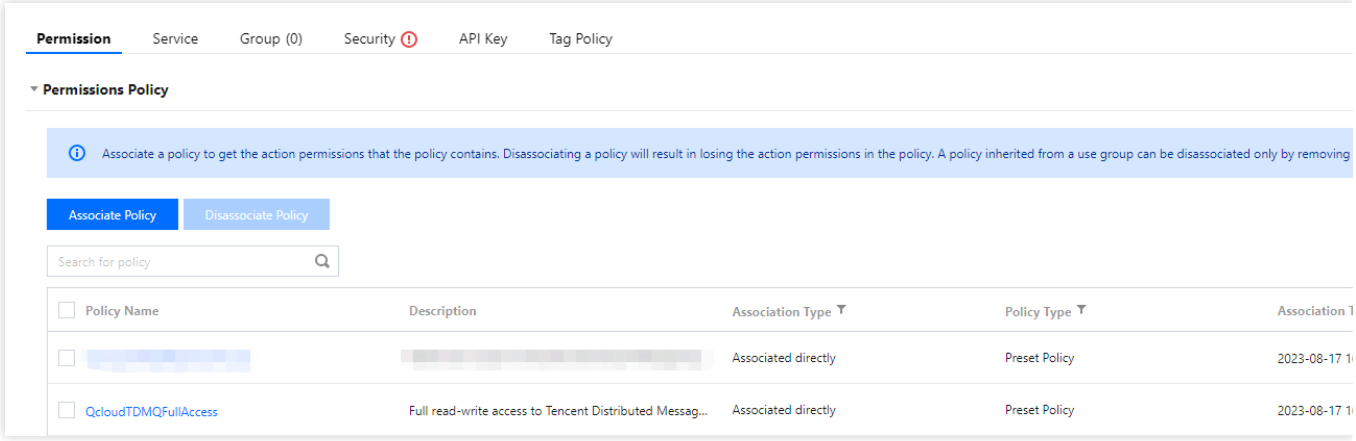
1. 使用主账号登录 访问管理控制台。
2. 在左侧导航栏，单击**策略**，进入策略管理列表页。
3. 在右侧搜索栏中，输入 **QcloudTDMQReadOnlyAccess** 进行搜索。



4. 在搜索结果中，单击 **QcloudTDMQReadOnlyAccess** 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：2023-09-12 17:00:17

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

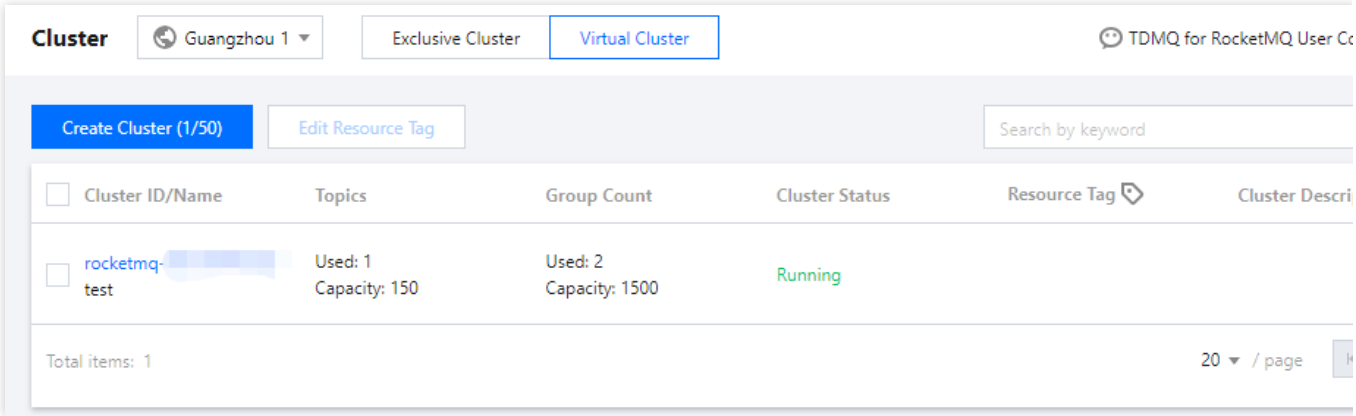
拥有腾讯云主账号，且已经开通腾讯云访问管理服务。
主账号下至少有一个子账号，且已根据 子账号获取访问授权 完成授权。
至少拥有一个 RocketMQ 实例。

操作步骤

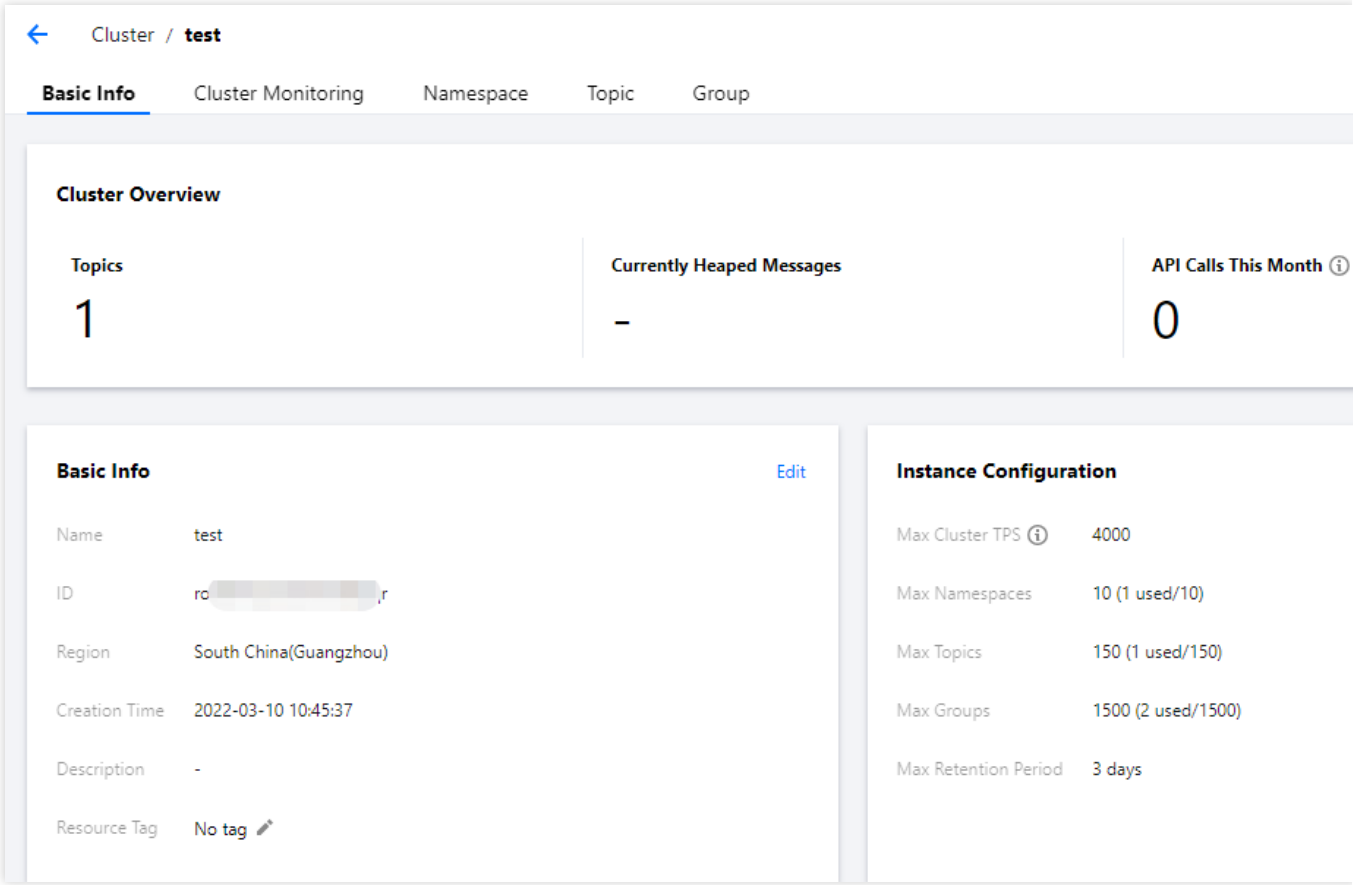
您可通过访问管理控制台的策略功能，将主账号拥有的 RocketMQ 资源授权给子账号，详细 **RocketMQ 资源授权给子账号**操作如下。本示例以授权一个集群资源给子账号为例，其他类型资源操作步骤类似。

步骤1：获取 RocketMQ 集群的 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。

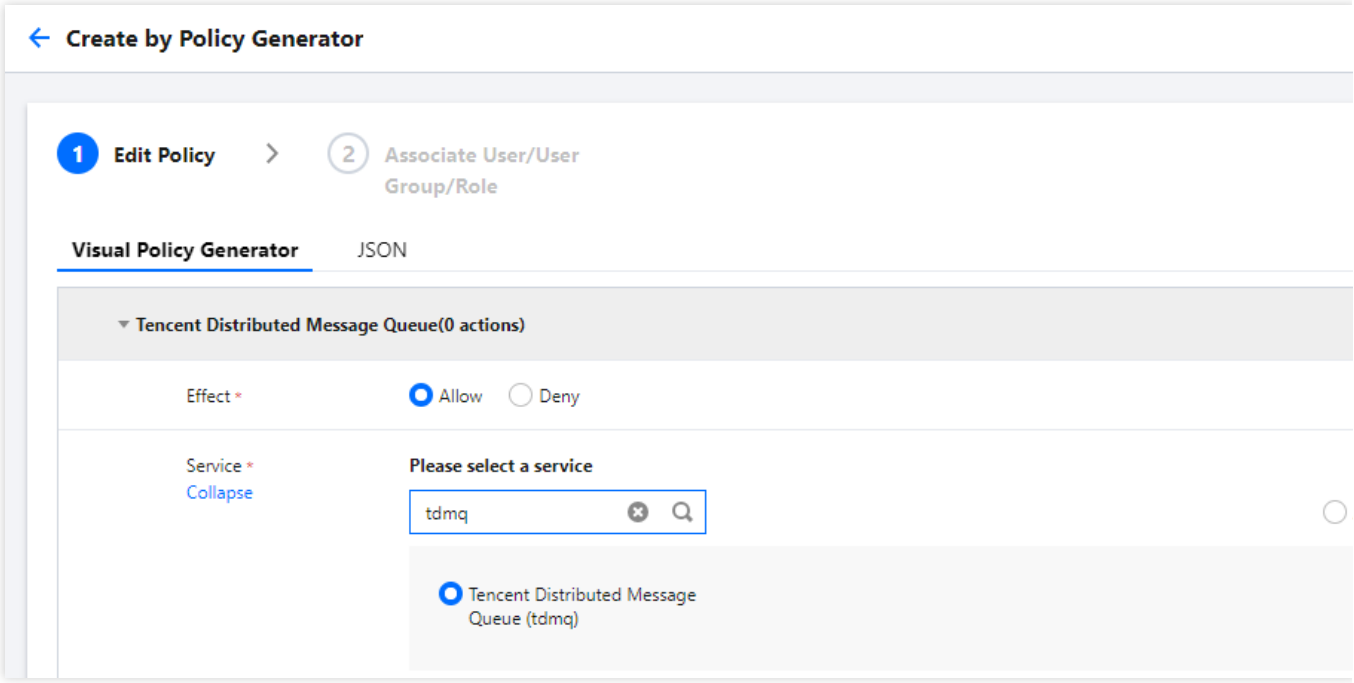


2. 在**基本信息**中，字段 **ID** 即为当前 RocketMQ 集群的 ID。



步骤2：新建授权策略

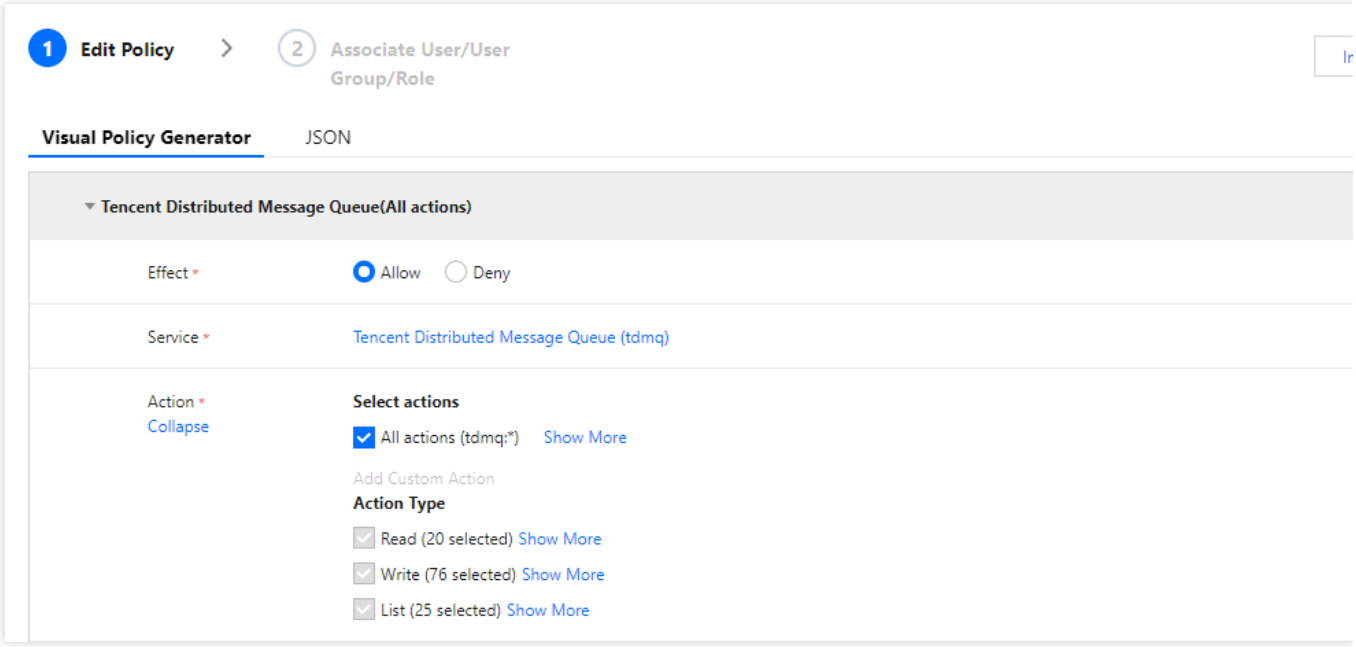
1. 进入访问管理控制台，单击左侧导航栏的 **策略**。
2. 单击**新建自定义策略**，选择**策略生成器创建**。
3. 在可视化策略生成器中，保持**效果**为**允许**，在**服务**中输入 TDMQ 进行筛选，在结果中选择**消息队列TDMQ (tdmq)**。



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。

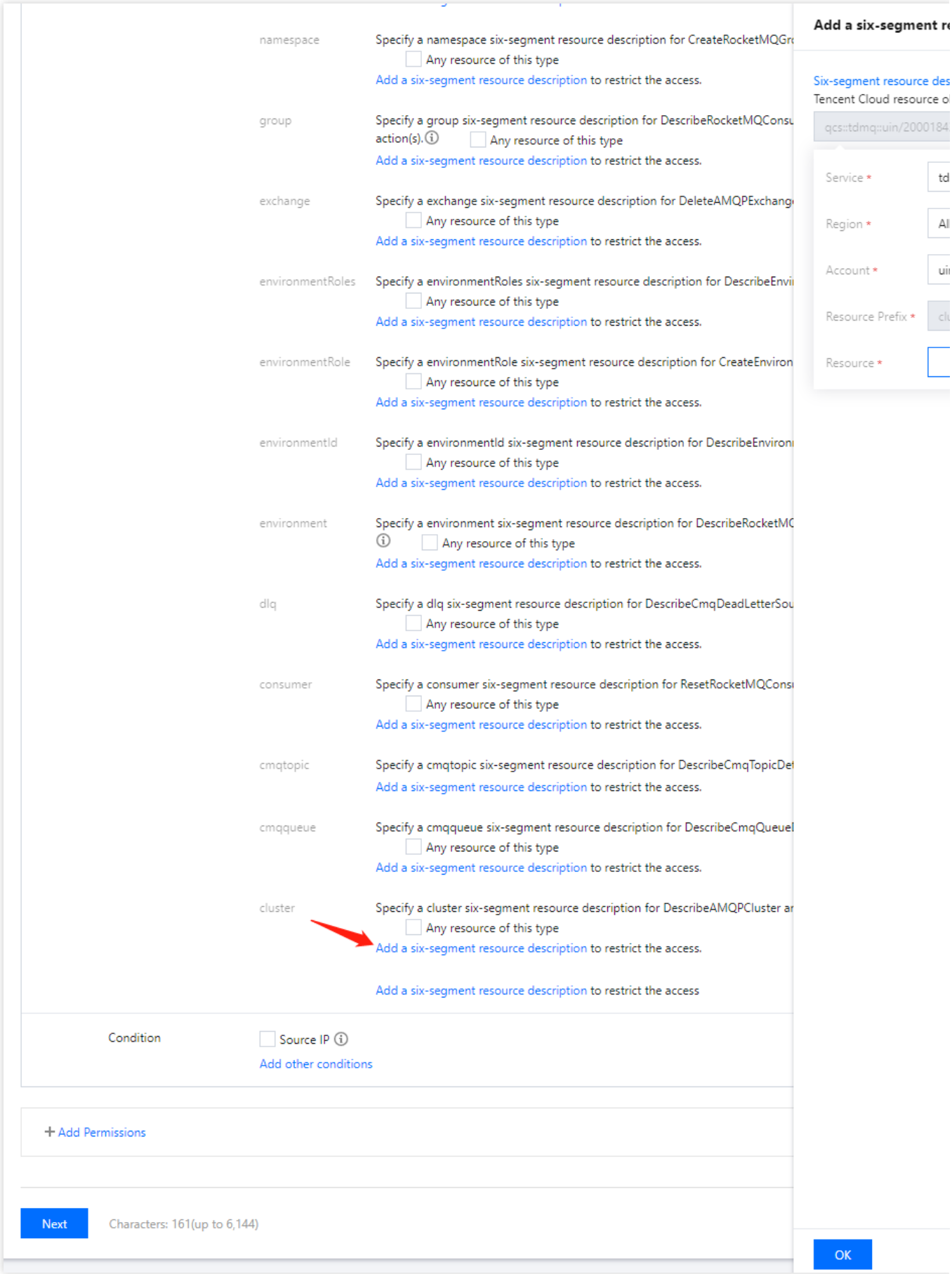
说明：

部分接口暂时不支持资源鉴权，以控制台页面展示为准。支持资源级授权的 API 列表可以参考文档 支持资源级授权的接口列表。

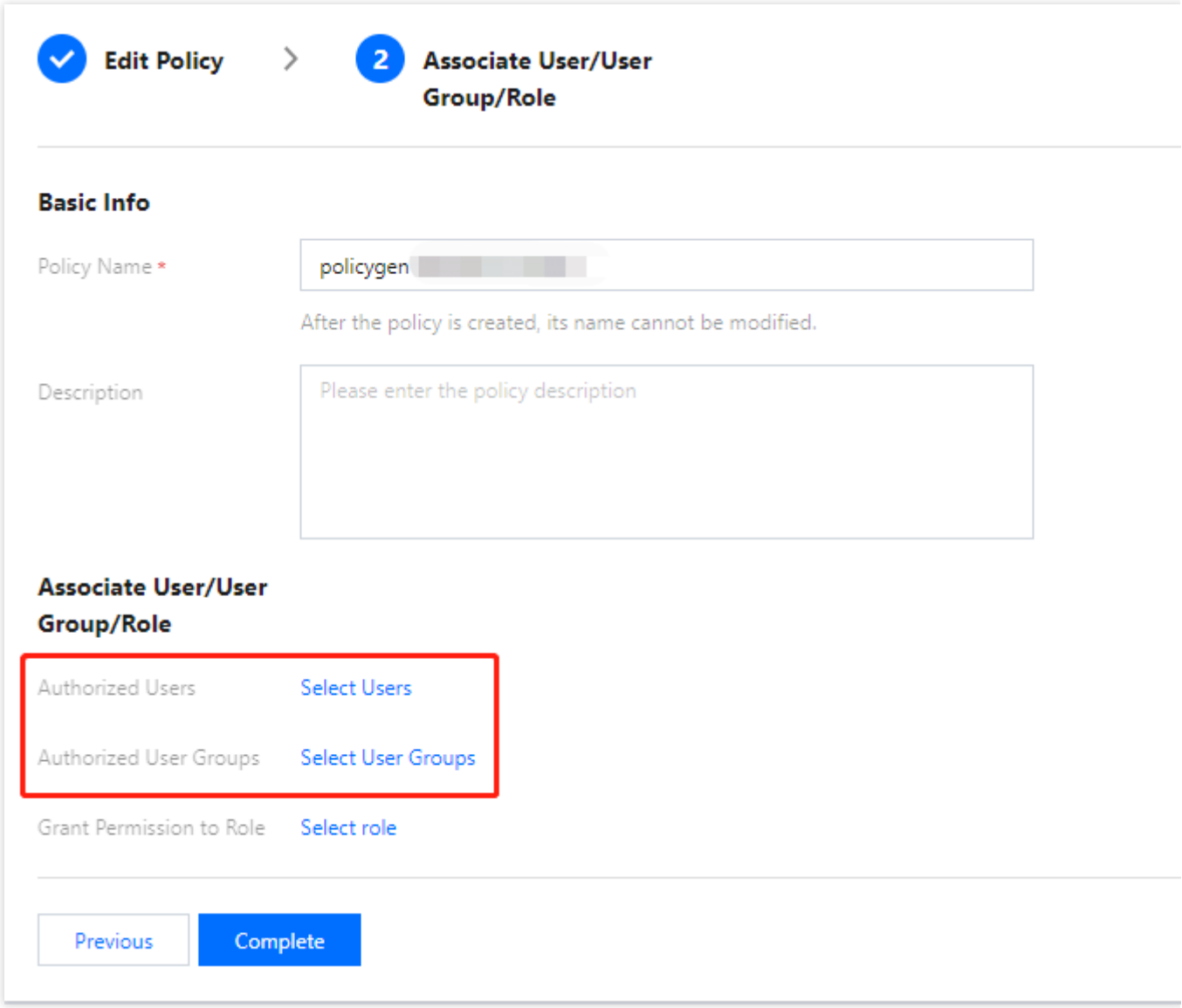


5. 在**资源**中选择**特定资源**，找到 **cluster** 资源类型，您可以勾选右侧**此类型任意资源（授权所有集群资源）**，或者并单击**添加资源六段式（授权特定集群资源）**。

6. 在弹出的侧边对话框中的**资源**中，填入**集群的 ID**，获取流程可参见 步骤1。


namespace
Specify a namespace six-segment resource description for CreateRocketMQGr
Any resource of this type
Add a six-segment resource description to restrict the access.
group
Specify a group six-segment resource description for DescribeRocketMQConsu
action(s).
Any resource of this type
Add a six-segment resource description to restrict the access.
exchange
Specify a exchange six-segment resource description for DeleteAMQPExchang
Any resource of this type
Add a six-segment resource description to restrict the access.
environmentRoles
Specify a environmentRoles six-segment resource description for DescribeEnvi
Any resource of this type
Add a six-segment resource description to restrict the access.
environmentRole
Specify a environmentRole six-segment resource description for CreateEnviron
Any resource of this type
Add a six-segment resource description to restrict the access.
environmentId
Specify a environmentId six-segment resource description for DescribeEnviron
Any resource of this type
Add a six-segment resource description to restrict the access.
environment
Specify a environment six-segment resource description for DescribeRocketMQ
Any resource of this type
Add a six-segment resource description to restrict the access.
dlq
Specify a dlq six-segment resource description for DescribeCmqDeadLetterSou
Any resource of this type
Add a six-segment resource description to restrict the access.
consumer
Specify a consumer six-segment resource description for ResetRocketMQCons
Any resource of this type
Add a six-segment resource description to restrict the access.
cmqtopic
Specify a cmqtopic six-segment resource description for DescribeCmqTopicDet
Add a six-segment resource description to restrict the access.
cmqqueue
Specify a cmqqueue six-segment resource description for DescribeCmqQueueI
Any resource of this type
Add a six-segment resource description to restrict the access.
cluster
Specify a cluster six-segment resource description for DescribeAMQPCluster ar
Any resource of this type
Add a six-segment resource description to restrict the access.
Add a six-segment resource description to restrict the access
Condition
Source IP
Add other conditions
+ Add Permissions
Next
Characters: 161(up to 6,144)
Add a six-segment r
Six-segment resource des
Tencent Cloud resource o
qcs::tdmq::uin/2000184
Service *
Region *
Account *
Resource Prefix *
Resource *
OK

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
ResetRocketMQConsumerOffSet	重置 RocketMQ 消费位点	consumer	qcs::tdmq:${region}:uin/${uin}:co
DescribeRocketMQClusters	获取 RocketMQ 集群列表	cluster	qcs::tdmq:${region}:uin/${uin}:cl
DeleteRocketMQCluster	删除 RocketMQ 集群	cluster	qcs::tdmq:${region}:uin/${uin}:cl
DescribeRocketMQCluster	获取单个 RocketMQ 集群信息	cluster	qcs::tdmq:${region}:uin/${uin}:cl
CreateRocketMQNamespace	创建 RocketMQ 命名空间	cluster	qcs::tdmq:${region}:uin/${uin}:cl
ModifyRocketMQNamespace	更新 RocketMQ 命名空间	namespace	qcs::tdmq:${region}:uin/${uin}:na
DeleteRocketMQNamespace	删除 RocketMQ 命名空间	namespace	qcs::tdmq:${region}:uin/${uin}:na
CreateRocketMQGroup	创建 RocketMQ 消费组	namespace	qcs::tdmq:${region}:uin/${uin}:na
ModifyRocketMQGroup	更新 RocketMQ 消费组信息	group	qcs::tdmq:${region}:uin/${uin}:gr
DescribeRocketMQGroups	获取 RocketMQ 消费组列表	group	qcs::tdmq:${region}:uin/${uin}:gr

DeleteRocketMQGroup	删除 RocketMQ 消费组	group	qcs::tdmq:${region}:uin/${uin}:gr
CreateRocketMQTopic	创建 RocketMQ 主题	namespace	qcs::tdmq:${region}:uin/${uin}:na
ModifyRocketMQTopic	更新 RocketMQ 主题信息	topic	qcs::tdmq:${region}:uin/${uin}:to
DeleteRocketMQTopic	删除 RocketMQ 主题	topic	qcs::tdmq:${region}:uin/${uin}:to
DescribeRocketMQTopics	获取 RocketMQ 主题列表	topic	qcs::tdmq:${region}:uin/${uin}:to
DescribeRocketMQTopicsByGroup	获取指定消费组下订阅的主题列表	topic	qcs::tdmq:${region}:uin/${uin}:to
DescribeRocketMQConsumerConnections	获取指定消费组下当前客户端的连接情况	group	qcs::tdmq:${region}:uin/${uin}:gr
DescribeRocketMQConsumerConnectionDetail	获取在线消费端详情	group	qcs::tdmq:${region}:uin/${uin}:gr
ModifyRocketMQCluster	修改 RocketMQ 集群信息	cluster	qcs::tdmq:${region}:uin/${uin}:cl

不支持资源级授权的 API 列表

API 名	API 描述	资源六段式
CreateRocketMQCluster	创建 RocketMQ 集群	*

授予子账号标签级权限

最近更新时间：2023-09-12 17:01:29

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

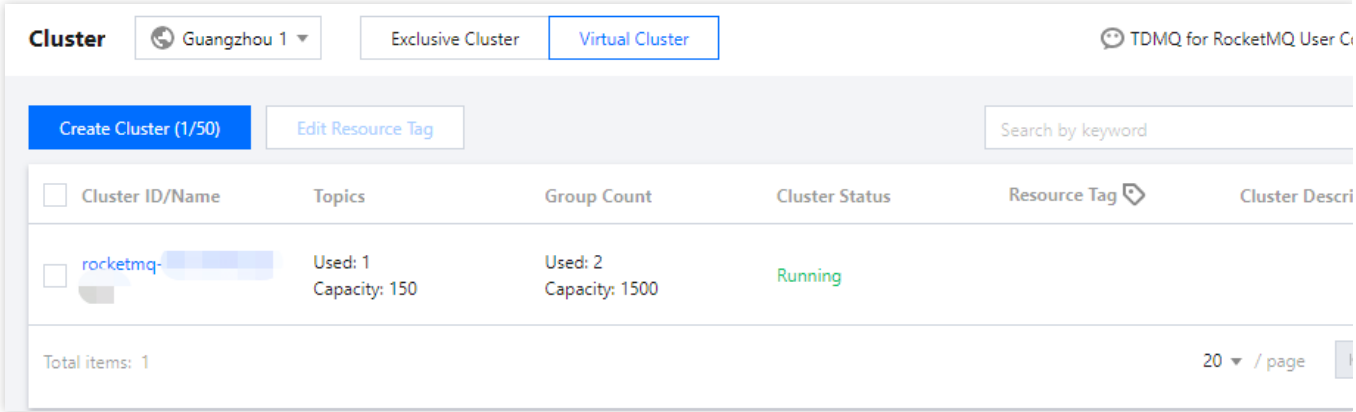
拥有腾讯云主账号，且已经开通腾讯云访问管理服务。
主账号下至少有一个子账号，且已根据 子账号获取访问授权 完成授权。
至少拥有一个 RocketMQ 集群资源实例。
至少拥有一个**标签**，若您没有，可以前往 标签控制台 > **标签列表**进行新建。

操作步骤

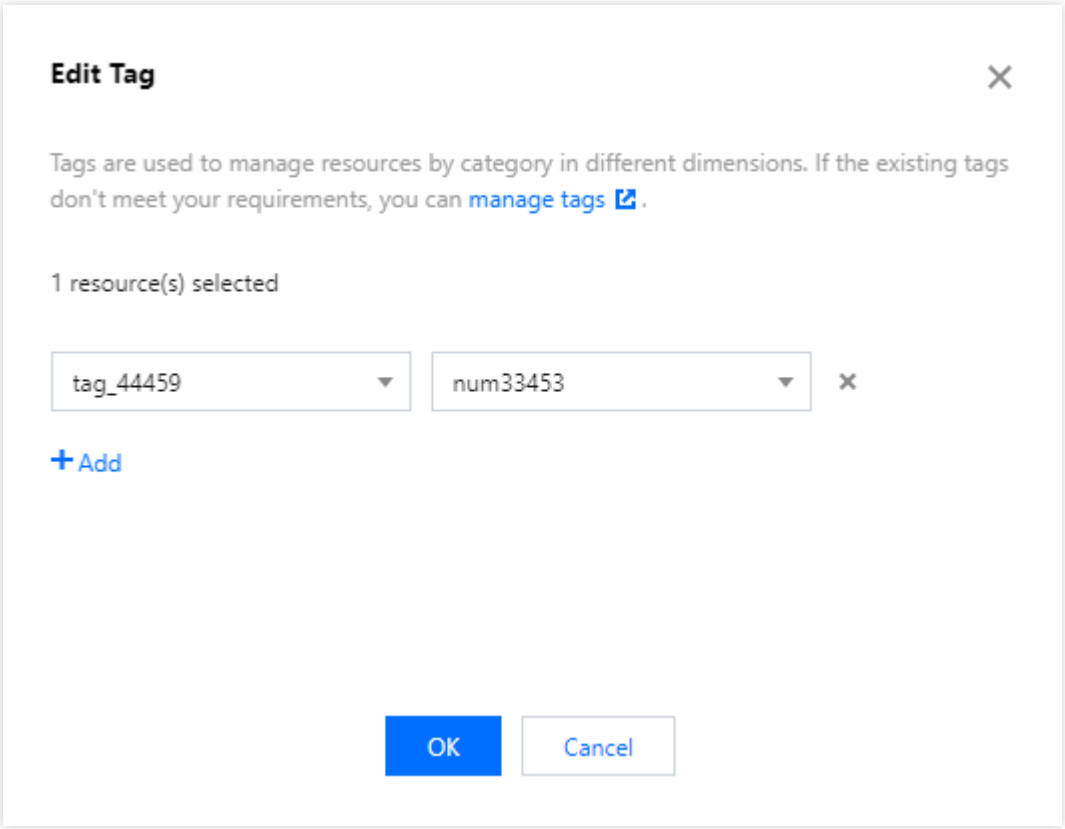
您可通过访问管理控制台的策略功能，将主账号拥有的、已经绑定标签的 RocketMQ 资源，通过**按标签授权**的方式授予子账号这些资源的读写权限，详细**按标签授予资源权限给子账号**的操作如下。

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。

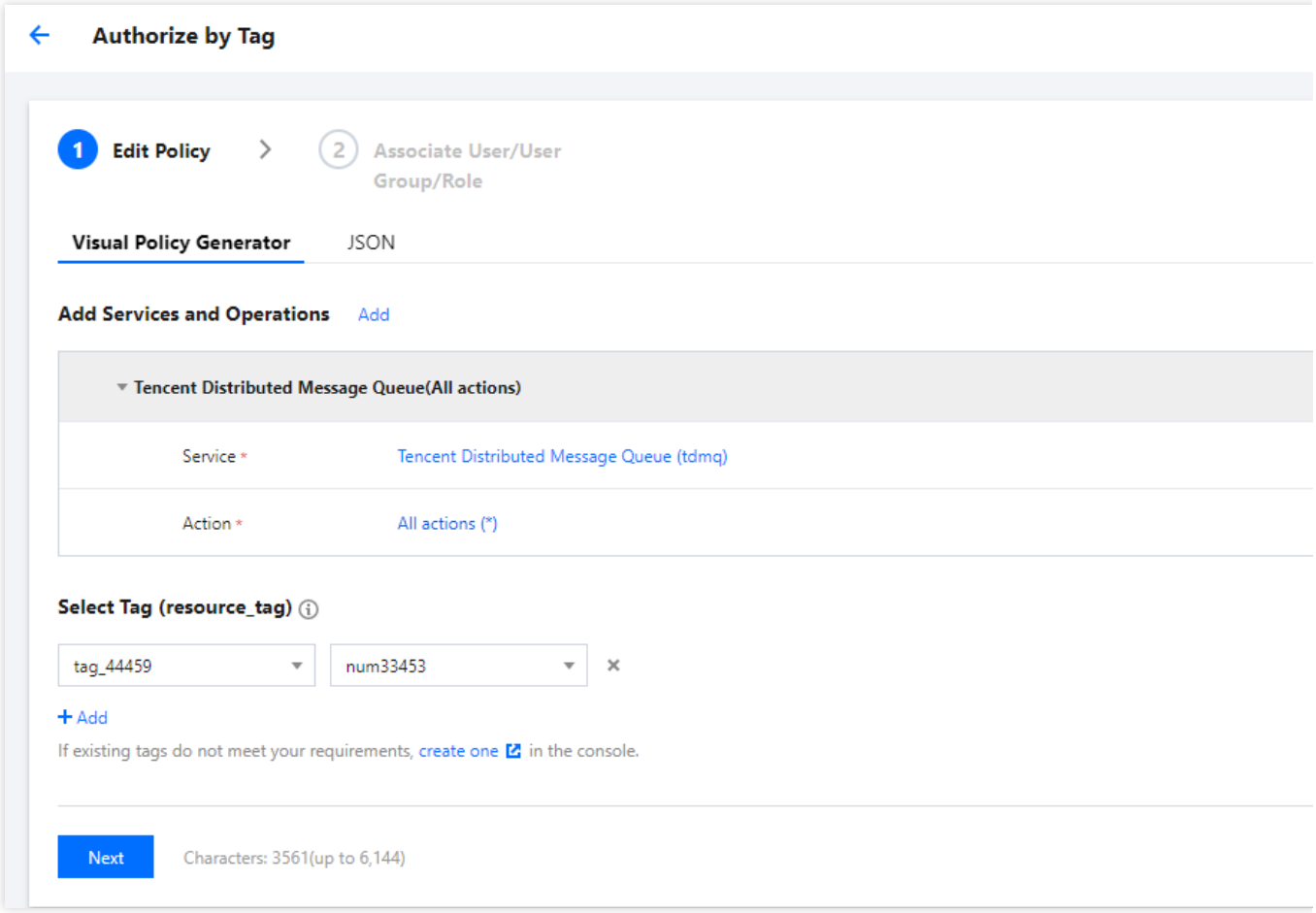


步骤 2：按标签授权

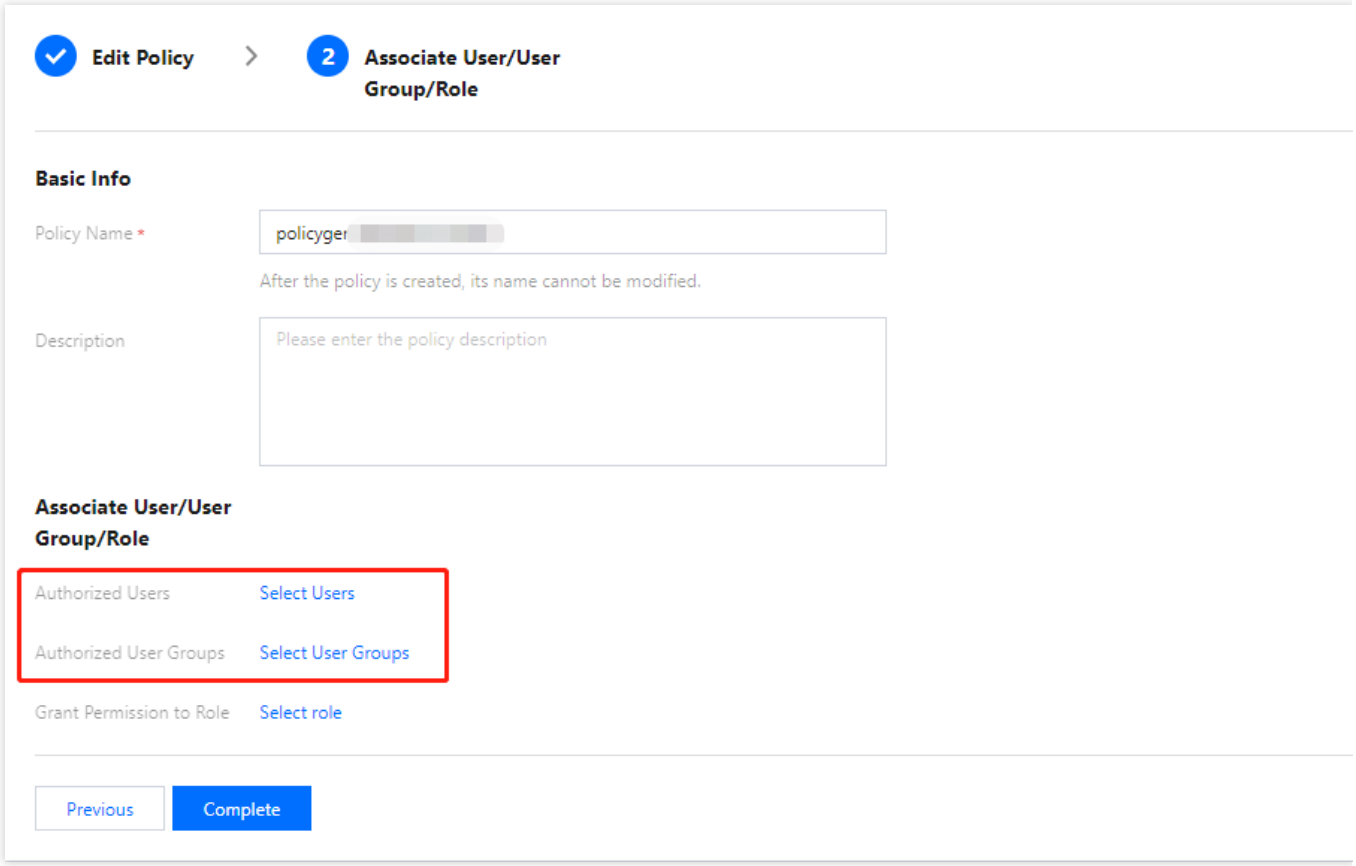
1. 进入访问管理控制台，单击左侧导航栏的 **策略**。
2. 单击**新建自定义策略**，选择**按标签授权**。
3. 在可视化策略生成器中，在**服务**中输入 tdmq 进行筛选，在结果中选择**消息队列TDMQ（tdmq）**，在**操作**中选择**全部操作**，您也可以根据需要选择相应的操作。

说明：

部分接口暂时不支持标签鉴权，以控制台页面展示为准。



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。

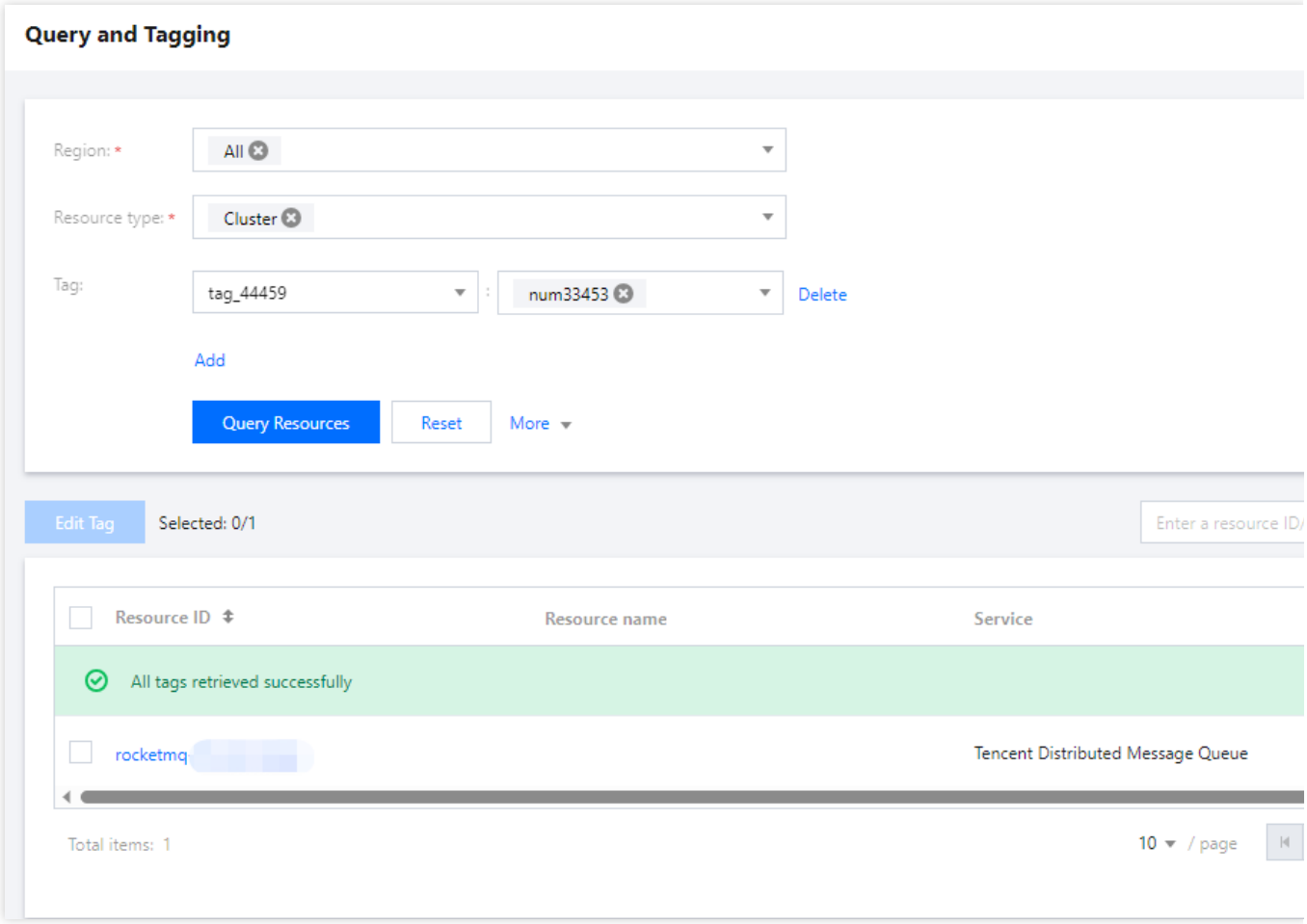


6. 单击**完成**，相关子账号就能够根据策略控制指定标签下的资源。

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 TDMQ > 集群**。
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
使用标签管理资源

最近更新时间：2024-01-18 10:05:43

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	黎四

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 ：

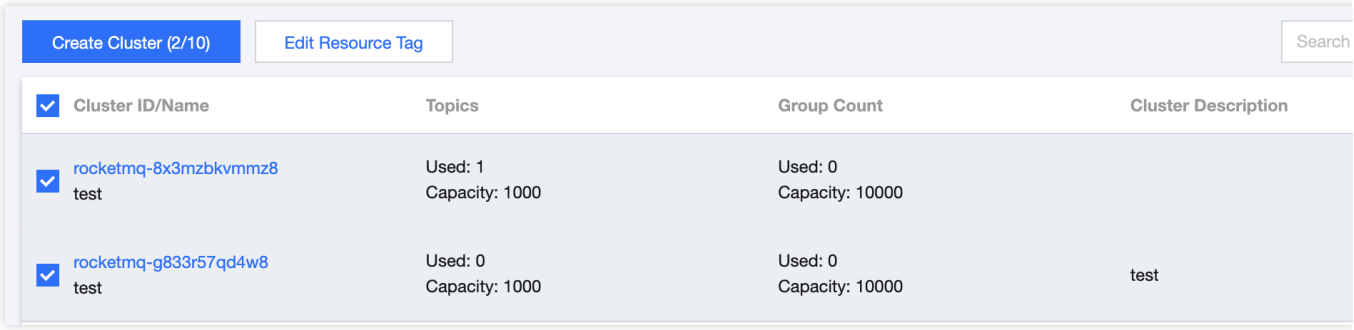
标签键	标签值
dept	ecommerce
business	mkt
owner	zhangsan

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

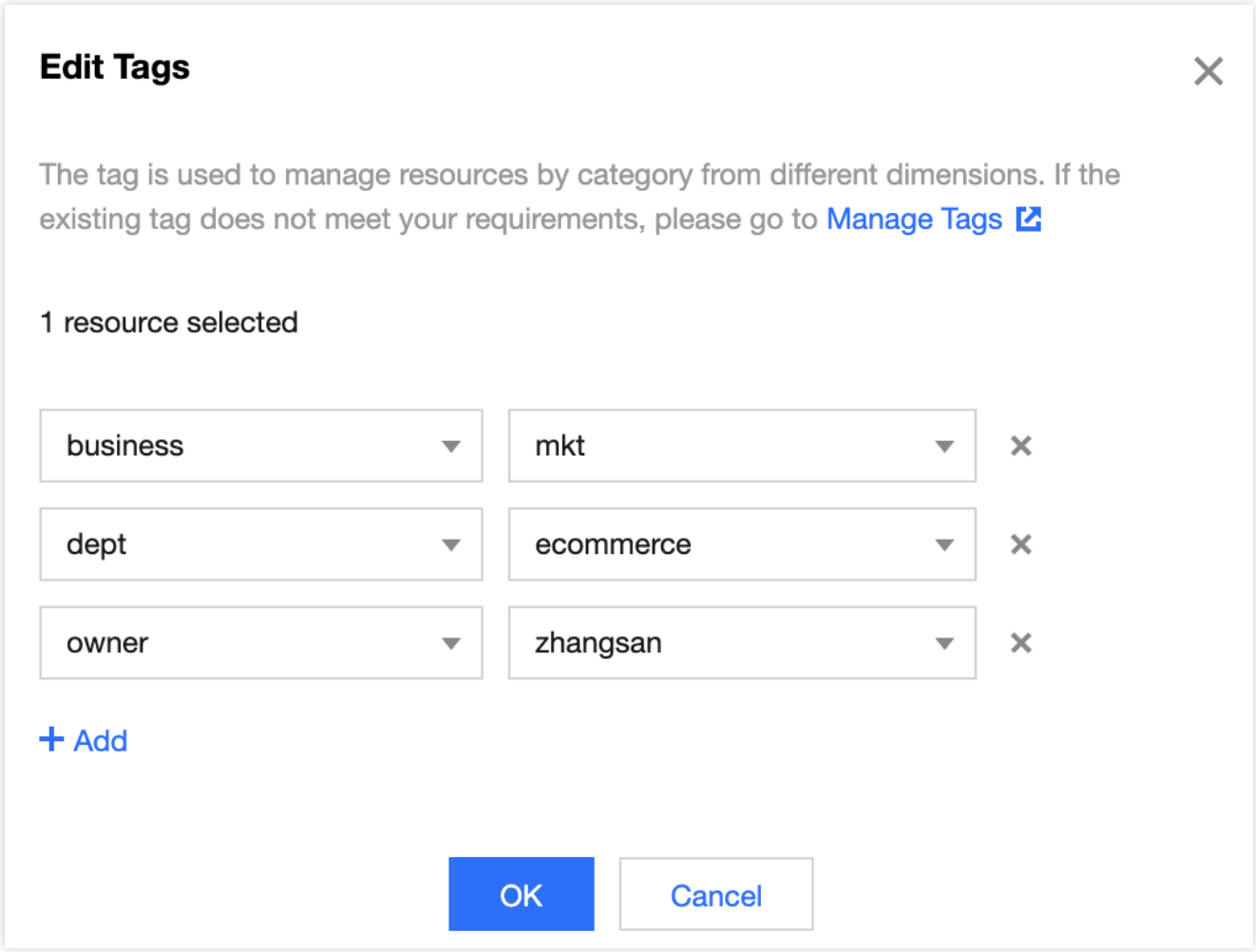
以上文场景为例，当您完成标签键和标签值的设计后，可以登录 TDMQ RocketMQ 版控制台进行标签的设置。

1. 登录 TDMQ RocketMQ 版控制台。
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。

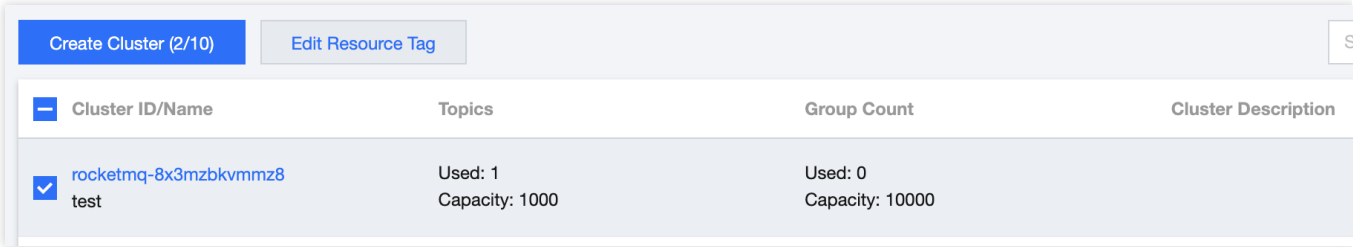
例如：为 rocketmq-qzga74ov5gw1 的集群添加三组标签。



说明：

如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。

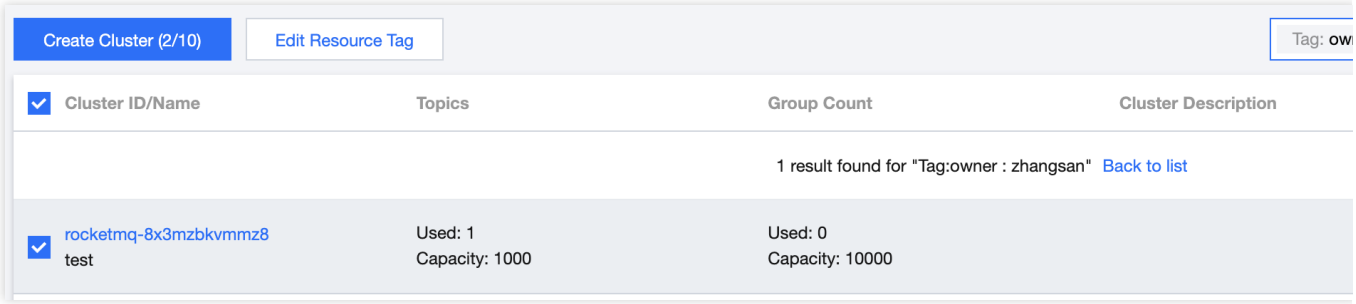


通过标签键筛选资源

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。


Create Cluster (2/10)
Edit Resource Tag
Tag: ow
Cluster ID/Name
Topics
Group Count
Cluster Description
1 result found for "Tag:owner : zhangsan"
Back to list
rocketmq-8x3mzbkvmmz8
test
Used: 1
Capacity: 1000
Used: 0
Capacity: 10000

编辑标签

最近更新时间：2024-01-18 10:06:16

操作场景

本文档指导您对资源进行编辑标签的操作。

使用限制

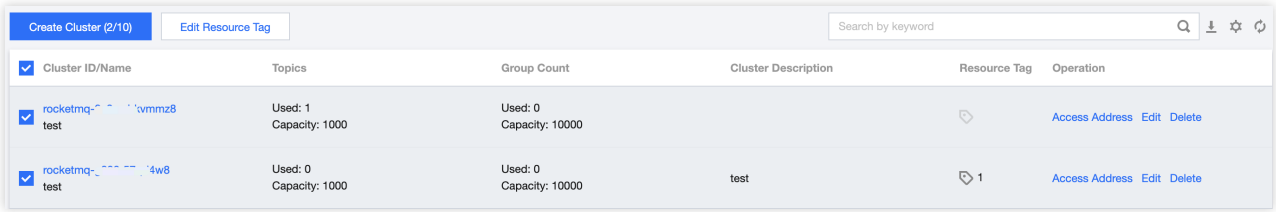
关于标签的使用限制，请参见 使用标签管理资源 - 使用限制。

前提条件

已登录 TDMQ RocketMQ 版控制台。

操作步骤

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。

操作案例

关于如何使用标签，请参见 使用标签管理资源。

监控告警
集群监控

最近更新时间：2023-09-12 17:53:17

操作场景

TDMQ RocketMQ 版支持集群监控功能，包括集群的生产消费指标、存储指标和消费者组指标等，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：

分类	单位	指标
生产消费	MBytes	集群每秒生产流量
	Count/s	消息生产速率
	MBytes	集群每秒消费流量
	Count/s	消息消费速率
	Count	消息堆积数
计费相关（仅虚拟集群展示）	Count/s	集群每秒被限流次数
	Count/s	集群消费消息每秒 API 调用次数
	Count/s	集群生产消息每秒 API 调用次数
存储（仅专享集群展示）	MBytes	磁盘可用空间
	%	磁盘使用比例
节点（仅专享集群展示）	%	节点负载（当前节点的实际TPS/ 购买页支持的 TPS 规格 * 100%）
消费者组	Count	在线消费者数

	Count/s	消息消费速率
	MBytes	每秒消费流量
	Count	消息堆积数

查看监控数据

1. 登录 TDMQ RocketMQ 版控制台。
2. 在左侧导航栏选择集群管理，选择好地域后，单击需要查看的集群的“ID”，进入集群详情页。
3. 在集群详情页顶部，选择**集群监控**页签，进入监控页面。
4. 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。

Group 监控

最近更新时间：2023-10-19 10:52:21

操作场景

本文介绍如何在 TDMQ RocketMQ 版控制台查看已创建 Group 的监控信息。

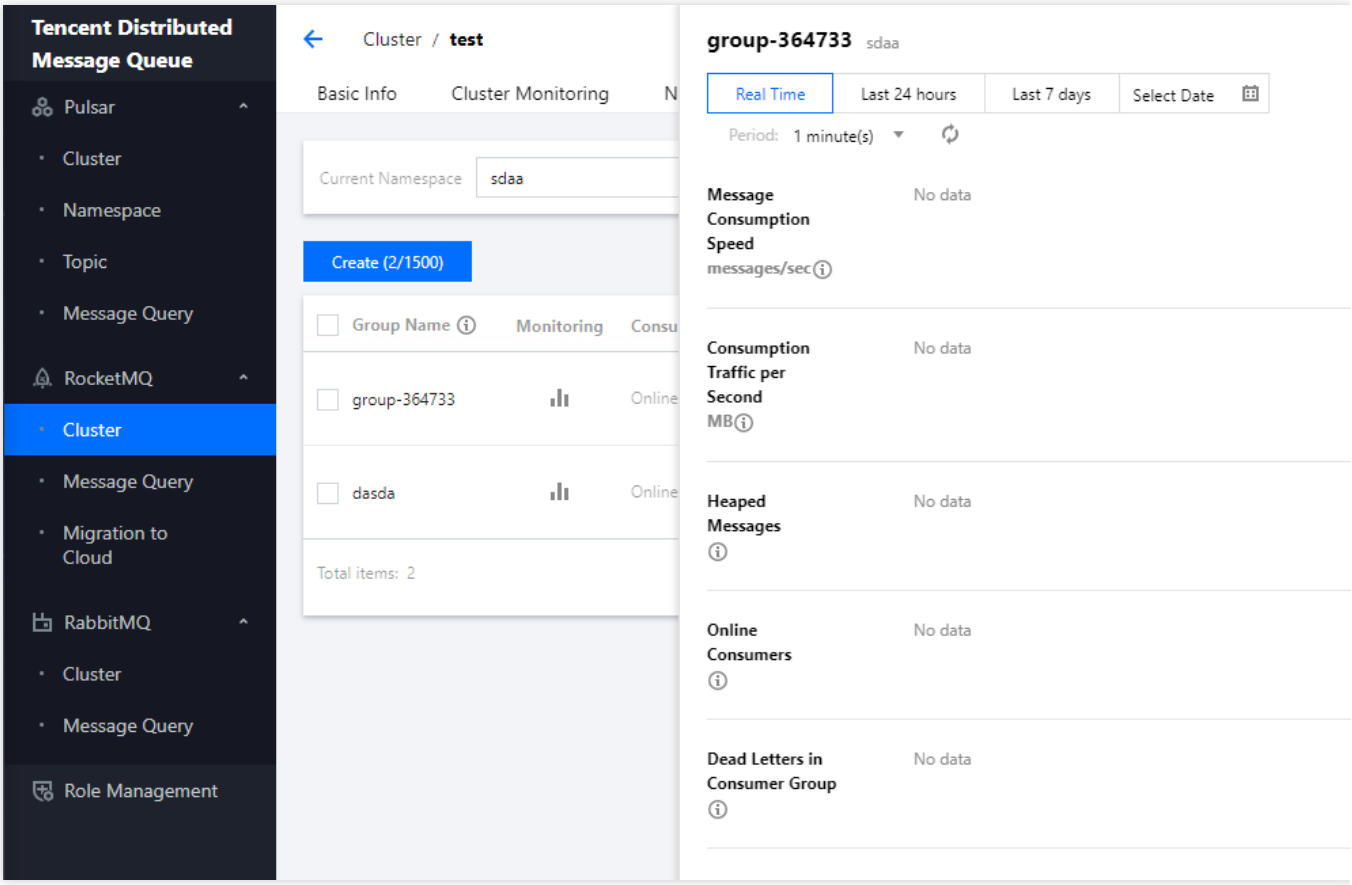
监控指标

监控指标	说明
消息消费速率（条/秒）	在所选时间范围内，该 Group 中所有消费者某一秒内消费消息的数量。
每秒消费流量（MB）	在所选时间范围内，该 Group 中所有消费者某一秒内消费的消息的数据量大小。
消息堆积数（条）	在所选时间范围内某一秒，该 Group 积压的未消费消息数量。
在线消费者数（个）	在所选时间范围内某一秒，该 Group 中在线的消费者总数。
消费组死信数量（条）	在所选时间范围内，该 Group 下所有消费者中死信消息的数量。

操作步骤

1. 登录 TDMQ 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. 选择顶部 Group 页签，在 Group 列表中，单击目标 Group 监控栏的

图标。
3. 设置好时间范围和时间粒度后，查看对应的 Group 监控数据。
4. 在监控页面，用户可以切换 Group 订阅的 Topic 的细分数据，如下图所示，可以查看某个 Group 订阅的一个特定 Topic 的消费速率和流量。


Tencent Distributed Message Queue
Pulsar
Cluster
Namespace
Topic
Message Query
RocketMQ
Cluster
Message Query
Migration to Cloud
RabbitMQ
Cluster
Message Query
Role Management
Cluster / test
Basic Info
Cluster Monitoring
Current Namespace
sdaa
Create (2/1500)
Group Name
Monitoring
group-364733
Online
dasda
Online
Total items: 2
group-364733
sdaa
Real Time
Last 24 hours
Last 7 days
Select Date
Period: 1 minute(s)
Message Consumption Speed
messages/sec
No data
Consumption Traffic per Second
MB
No data
Heaped Messages
No data
Online Consumers
No data
Dead Letters in Consumer Group
No data

Topic 监控

最近更新时间：2023-10-19 11:00:38

操作场景

本文介绍如何在 TDMQ RocketMQ 版控制台查看已创建 Topic 的监控信息。

监控指标

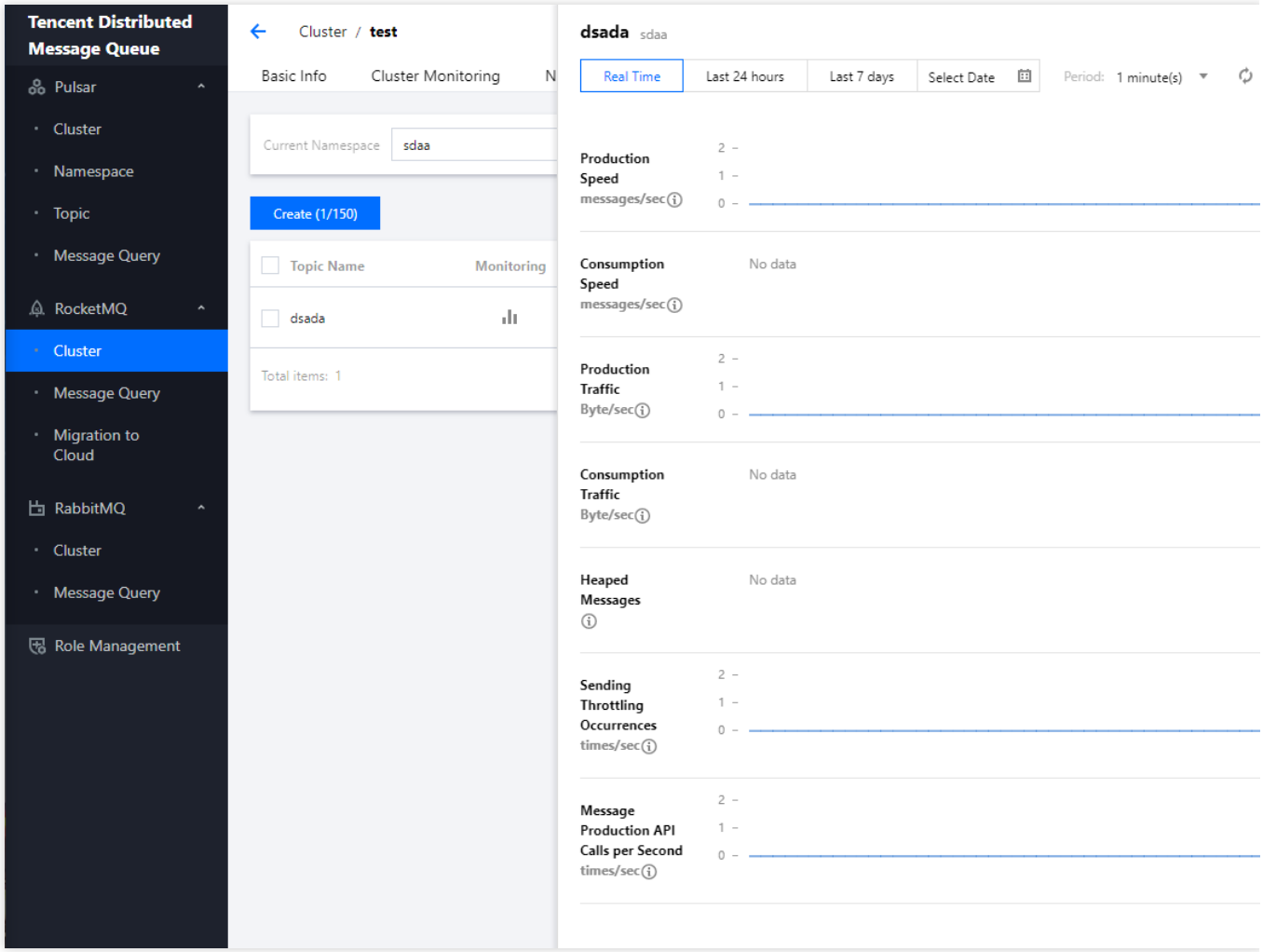
监控指标	说明
生产速率（条/秒）	在所选时间范围中，本 Topic 下所有生产者某一秒内生产消息的数量。
消费速率（条/秒）	在所选时间范围中，本 Topic 下所有消费者某一秒内消费消息的数量。
生产流量（字节/秒）	在所选时间范围内某一秒，本 Topic 下所有生产者生产的消息数据量大小。
消费流量（字节/秒）	在所选时间范围内某一秒，本 Topic 下所有消费者消费的消息数据量大小。
消息积压数量（条）	在所选时间范围内某一秒，本 Topic 下积压的消息数量。
发送被限流次数（次/秒）	在所选时间范围内某一秒，本 Topic 生产消息被限流的次数。
API 调用次数（次/秒）	仅虚拟集群展示，在所选时间范围内某一秒，本 Topic 生产消息时的 TPS （按照计费的规则进行折算的 API 调用次数）。

操作步骤

1. 登录 TDMQ 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. 选择顶部 **Topic** 页签，在 Topic 列表中，单击目标Topic 监控栏的

图标。

3. 设置好时间范围后，查看对应的 Topic 监控数据。


Tencent Distributed Message Queue
Pulsar
Cluster
Namespace
Topic
Message Query
RocketMQ
Cluster
Message Query
Migration to Cloud
RabbitMQ
Cluster
Message Query
Role Management
Cluster / test
Basic Info
Cluster Monitoring
Current Namespace
sdaa
Create (1/150)
Topic Name
Monitoring
dsada
Total items: 1
dsada sdaa
Real Time
Last 24 hours
Last 7 days
Select Date
Period: 1 minute(s)
Production Speed messages/sec
Consumption Speed messages/sec
No data
Production Traffic Byte/sec
Consumption Traffic Byte/sec
No data
Heaped Messages
No data
Sending Throttling Occurrences times/sec
Message Production API Calls per Second times/sec

配置告警

最近更新时间：2023-05-16 10:55:32

操作场景

腾讯云默认为所有用户提供腾讯云可观测平台功能，无需用户手动开通。用户在使用了腾讯云某个产品后，腾讯云可观测平台才可以开始收集监控数据。
TDMQ RocketMQ 版支持监控您账户下创建的资源，帮助您实时掌握资源状态。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设置好的通知方式及时通知您，帮助您及时应对异常情况。

操作步骤

配置告警规则

创建的告警会将一定周期内监控的指标与给定阈值的情况进行比对，从而判断是否需要触发相关通知。当 TDMQ RocketMQ 版状态改变而导致告警触发后，您可以及时进行相应的预防或补救措施，合理地创建告警能帮助您提高应用程序的健壮性和可靠性。

注意

请务必对实例配置告警，防止因突发流量或者到达规格限制而导致的异常。

1. 登录 腾讯云可观测平台控制台。
2. 在左侧导航栏选择**告警配置** > **告警策略**，单击**新建**。
3. 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/RocketMQ/主题告警**。

告警对象：选择需要配置告警策略的 TDMQ RocketMQ 版资源。

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“消息堆积量”，选择统计粒度为1分钟，则在1分钟内，消息堆积量连续N个数据点超过阈值，就会出发告警。

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。

4. 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. 在左侧导航栏中，单击**触发条件模板**，进入触发条件列表页面。
3. 在触发条件模板页单击**新建**。
4. 在新建模板页，配置策略类型。

策略类型：选择**消息队列 TDMQ-RocketMQ-主题告警**。

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。

5. 确认无误后，单击**保存**。
6. 返回新建告警策略页，单击**刷新**，则会显示刚配置的告警策略模板。

告警指标维度说明

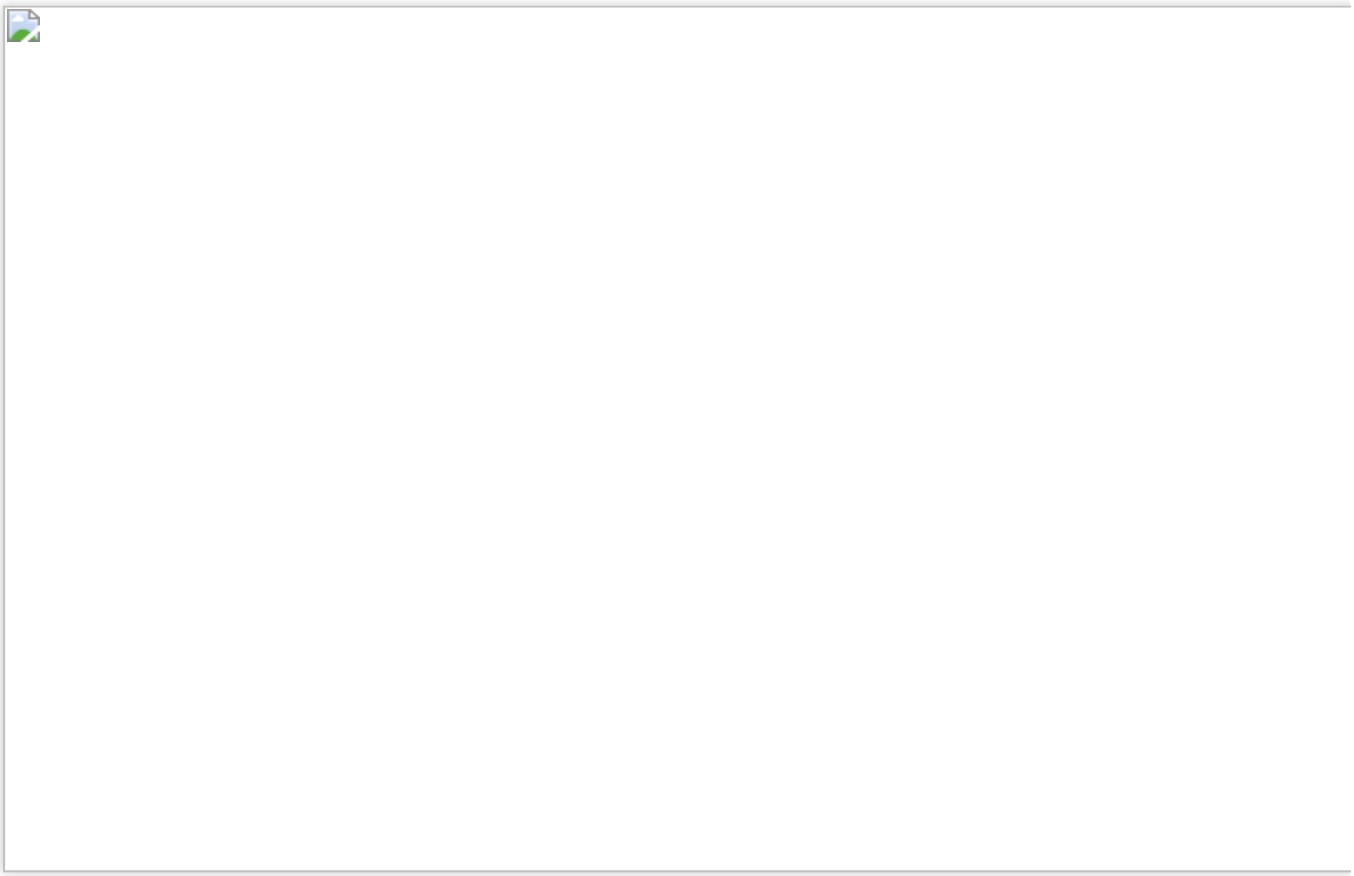
目前 RocketMQ 的指标分为以下几个维度，用户可以根据需要选择相应的指标进行监控和告警的配置：

指标维度	指标说明
集群	整个集群维度的数据聚合，如集群的生产/消费速率，集群的生产/消费流量，消息堆积数量和被限流的次数
存储（仅专享集群）	当前专享集群剩余的可用的存储空间，以及存储已使用的占比
节点（仅专享集群）	当前集群的各个计算节点的负载情况
主题（Topic）	所选中的主题的生产/消费速率，生产/消费流量，消息堆积数量
消费者组（Group）	所选中的消费者组的在线消费者（客户端）的数量，消费消息的速率和流量，当前 Group 的堆积和死信数量等
主题下的消费者组	将上述的 Group 的指标按照所订阅的主题进行拆分，展示订阅了某个主题的 group 的消费速率和流量以及堆积数等

消息查询
查询消息

最近更新时间：2023-03-28 10:15:36

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。



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/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。

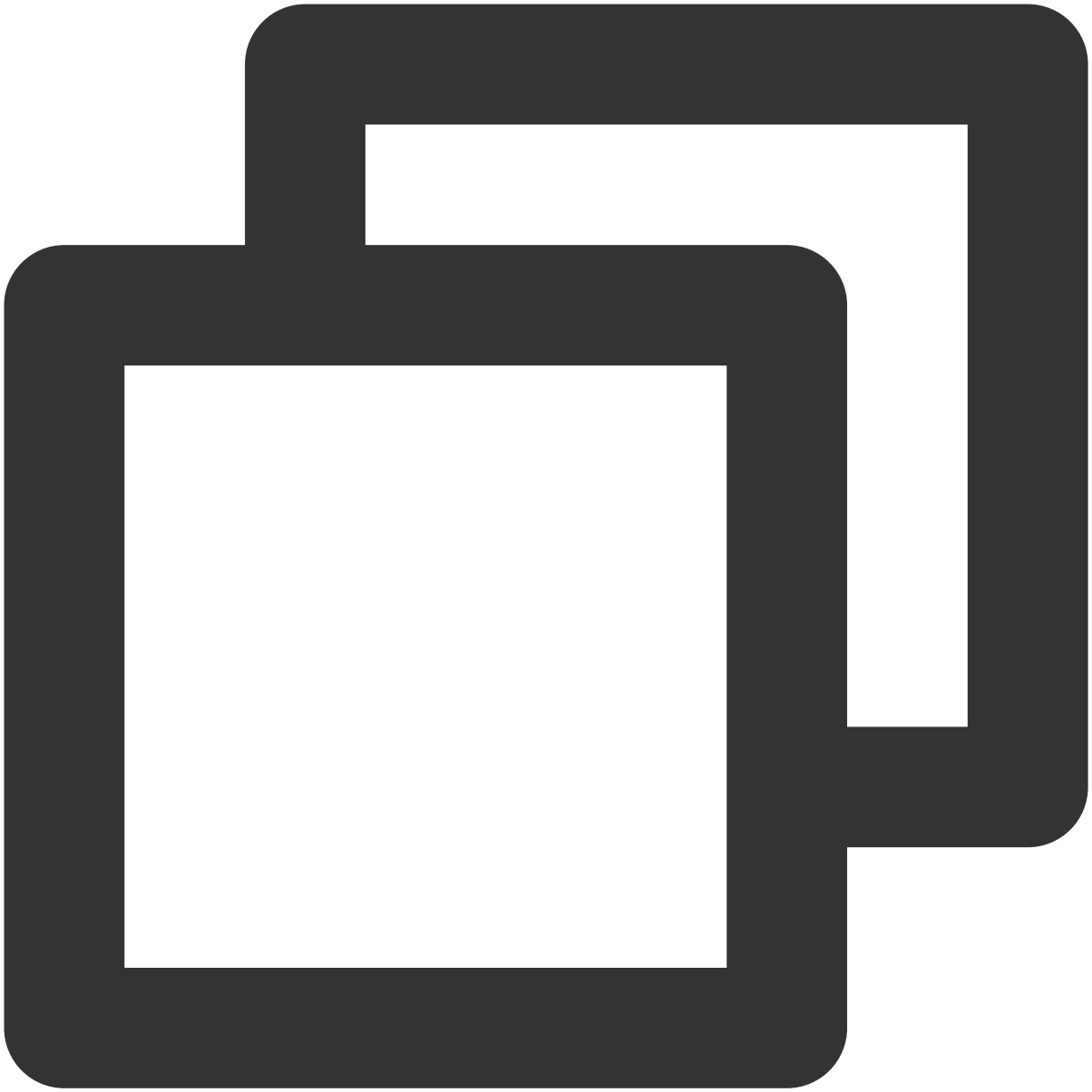
查看消费由哪些消费者消费了，是否消费成功，消息确认消费的具体时间。
需要做分布式系统的性能分析，查看 MQ 对相关消息处理的时延。

查询限制

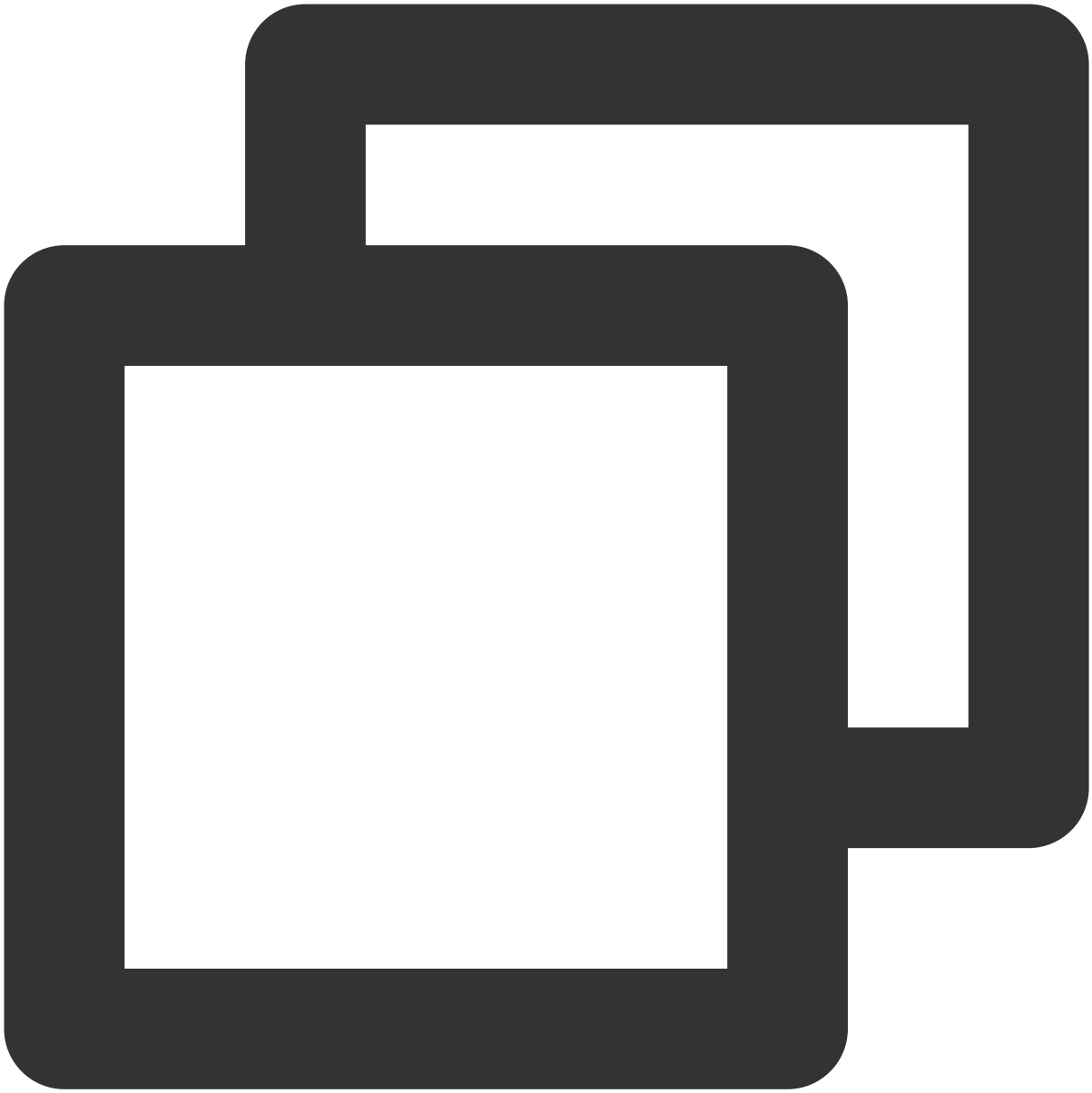
消息查询最多可以查询近3天的消息。

前提条件

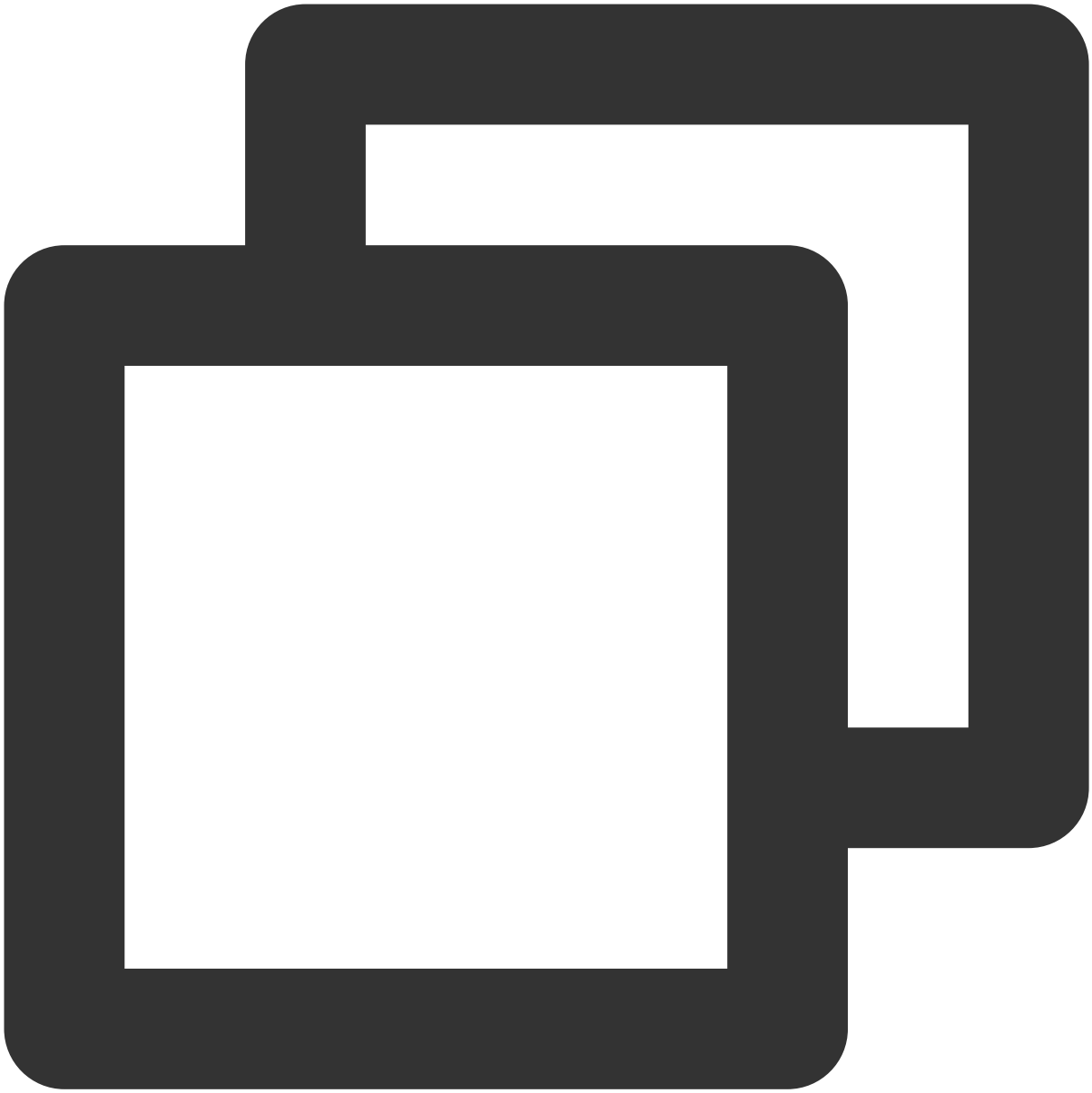
已经参考 SDK 文档 部署好生产端和消费端服务，并在3天内有消息生产和消费。
2022年8月8日部分地区上线新的虚拟集群 RocketMQ 服务，由于新版的集群与旧版集群的消息轨迹实现不同，所以在使用上有些许差异。旧版集群消息轨迹由服务端实现，客户端无需关注消息轨迹相关设置，新版集群需要客户端进行来设置开启消息轨迹功能，具体设置示例如下：
生产者设置
push 消费者设置
pull 消费者设置



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

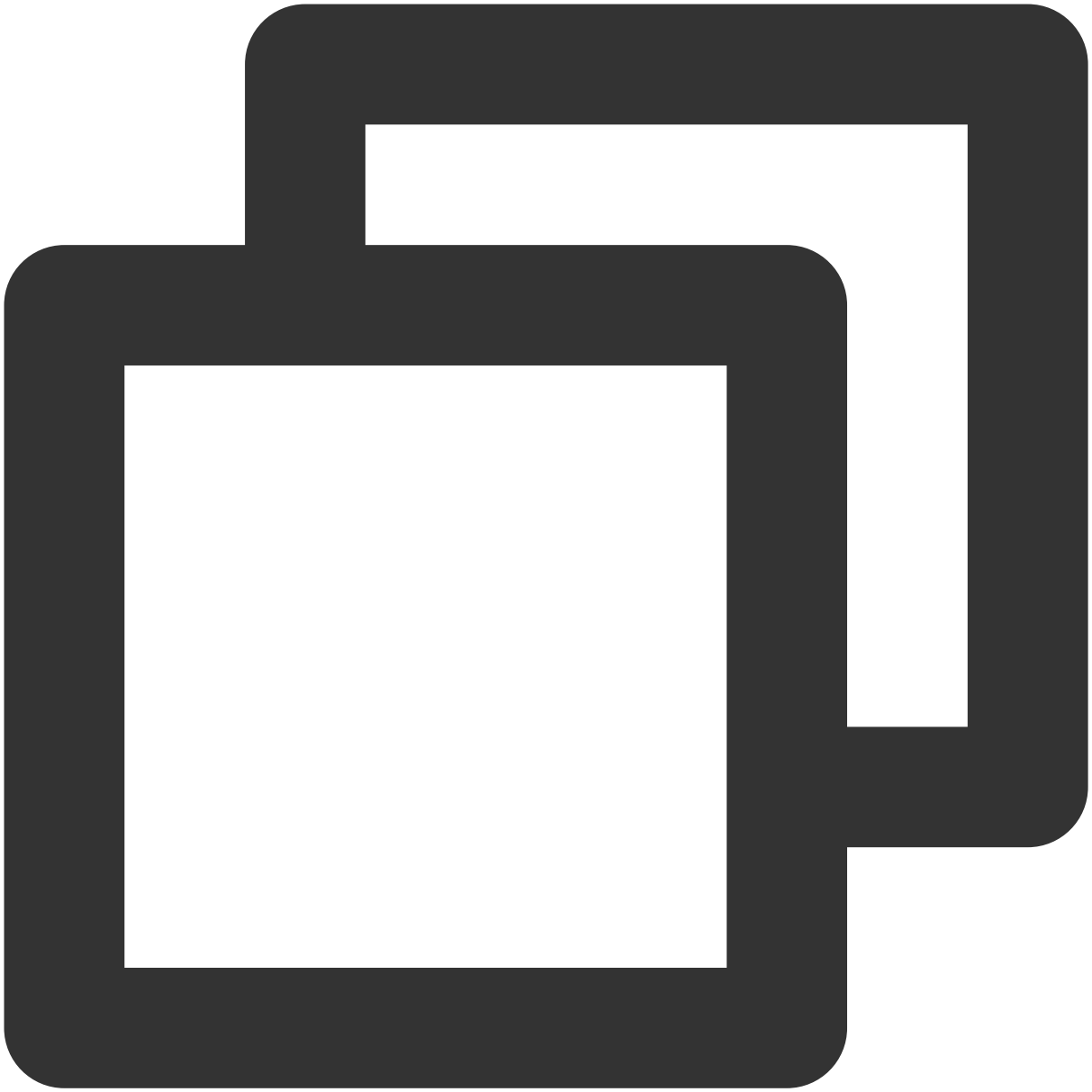


```
// 实例化消费者
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(NAMESPACE,groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)),
    new AllocateMessageQueueAveragely(), true, null);
```



```
DefaultLitePullConsumer pullConsumer = new DefaultLitePullConsumer(NAMESPACE,groupN
    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);
```

如果是使用 Spring Boot Starter 接入（2.2.2版本及以上），具体的代码参考：



```
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），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 Topic 所在的集群。

命名空间：选择需要查询的 Topic 所在的命名空间。

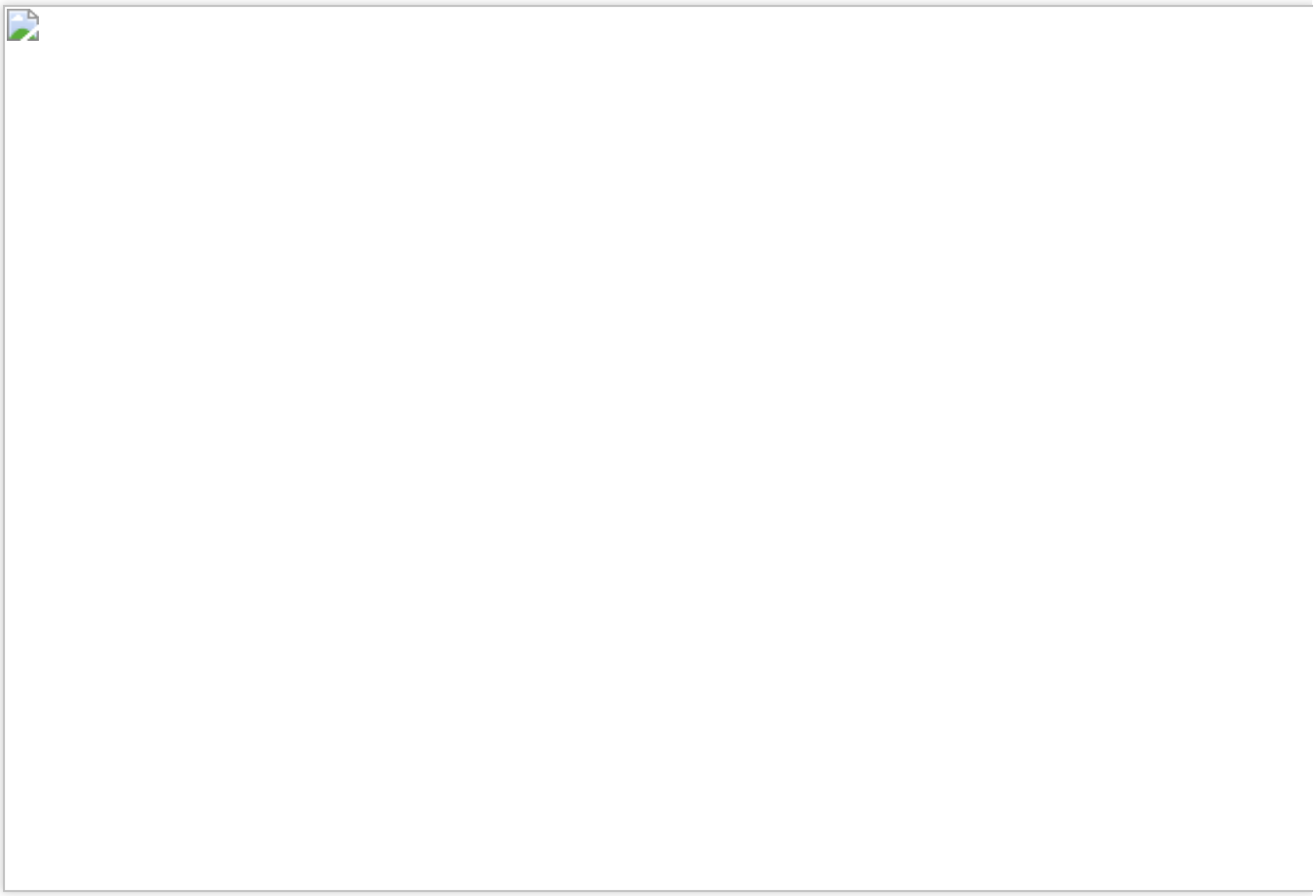
Topic：选择需要查询的 Topic。

查询方式：消息查询功能支持两种查询方式。

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。

说明：

消费模式为 “广播消费” 的 Group 下如果没有在线的客户端，则不会在上图的详情页展示消费信息和异常诊断信息；如果您需要查看消费状态，可以前往 “消息轨迹” 页进行查看。

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

当前仅专享集群支持消费验证功能。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。

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。
同时，您可以批量选择当前页面内的消息，批量导出多条消息。将消息导出到本地后，用户可以根据需要对消息进行一些处理，如复制消息体或者进行时间排序等操作。

查询死信消息

最近更新时间：2023-03-14 15:28:23

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**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。
命名空间：选择需要查询的死信消息所在的命名空间。
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。

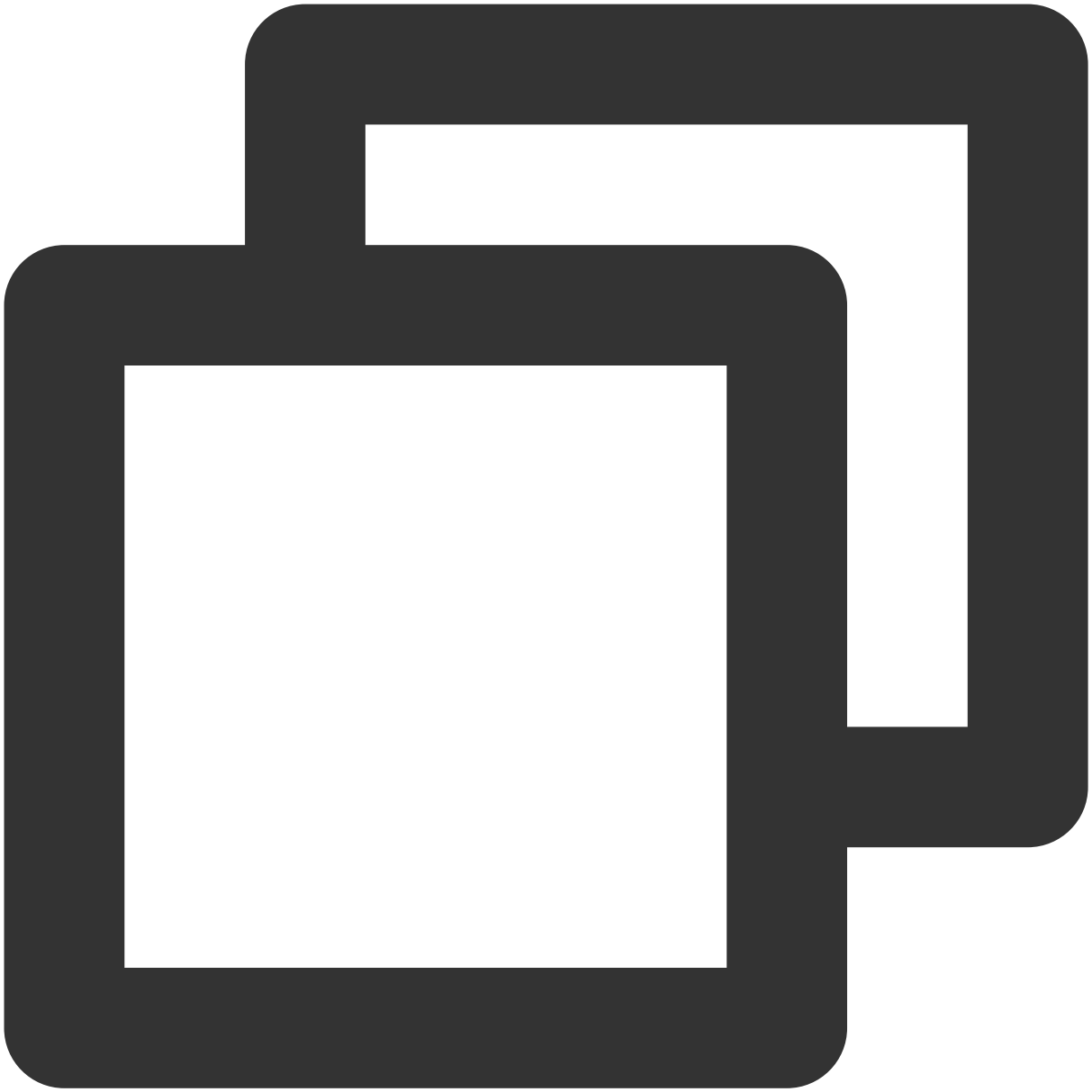
消息轨迹与说明

最近更新时间：2023-03-14 15:29:18

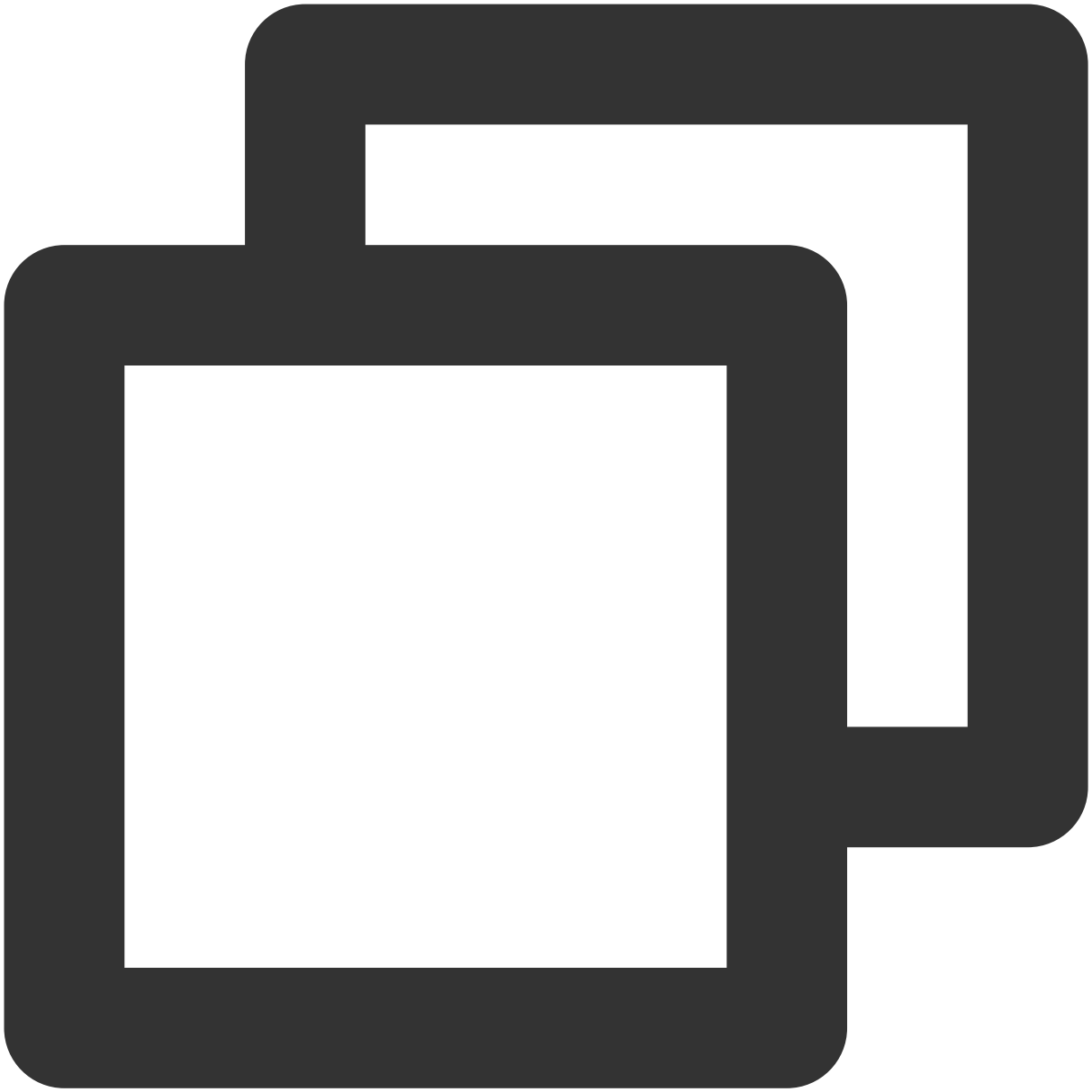
消息轨迹记录了消息从生产端到 TDMQ RocketMQ 版服务端，最后到消费端的整个过程，包括各阶段的时间（精确到微秒）、执行结果、生产者 IP、消费者 IP 等。

前提条件

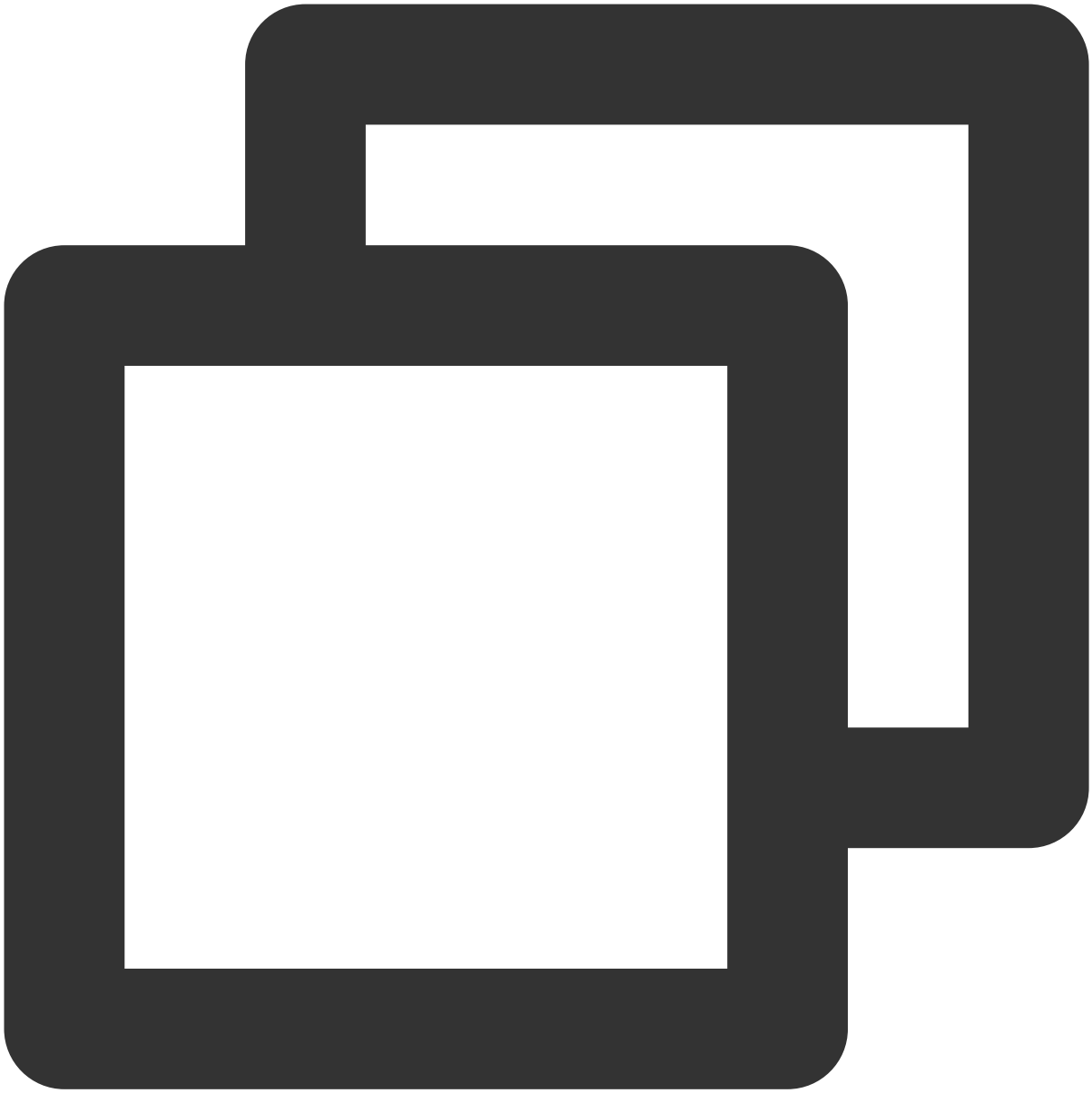
已经参考 SDK 文档 部署好生产端和消费端服务，并在3天内有消息生产和消费。
需要客户端进行来设置开启消息轨迹功能，具体设置示例如下：
生产者设置
push 消费者设置
pull 消费者设置



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

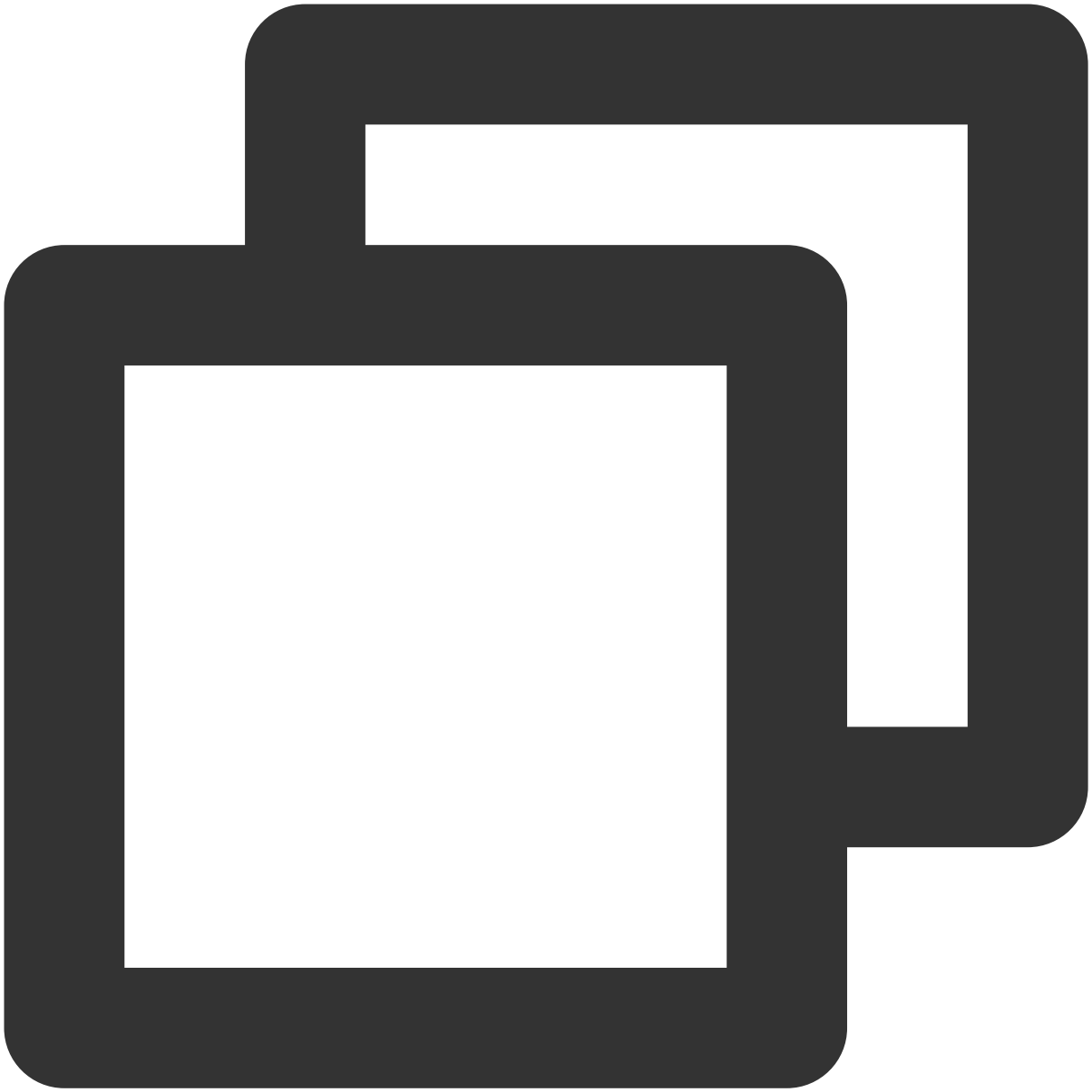


```
// 实例化消费者
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(NAMESPACE,groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)),
    new AllocateMessageQueueAveragely(), true, null);
```



```
DefaultLitePullConsumer pullConsumer = new DefaultLitePullConsumer(NAMESPACE,groupN
    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);
```

如果是使用 Spring Boot Starter 接入（2.2.2版本及以上），具体的代码参考：



```
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 Topic 所在的集群。

命名空间：选择需要查询的 Topic 所在的命名空间。

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。

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，详细说明请参见 集群消费和广播消费。
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。

消息跨集群复制

最近更新时间：2024-05-14 17:36:16

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），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（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。

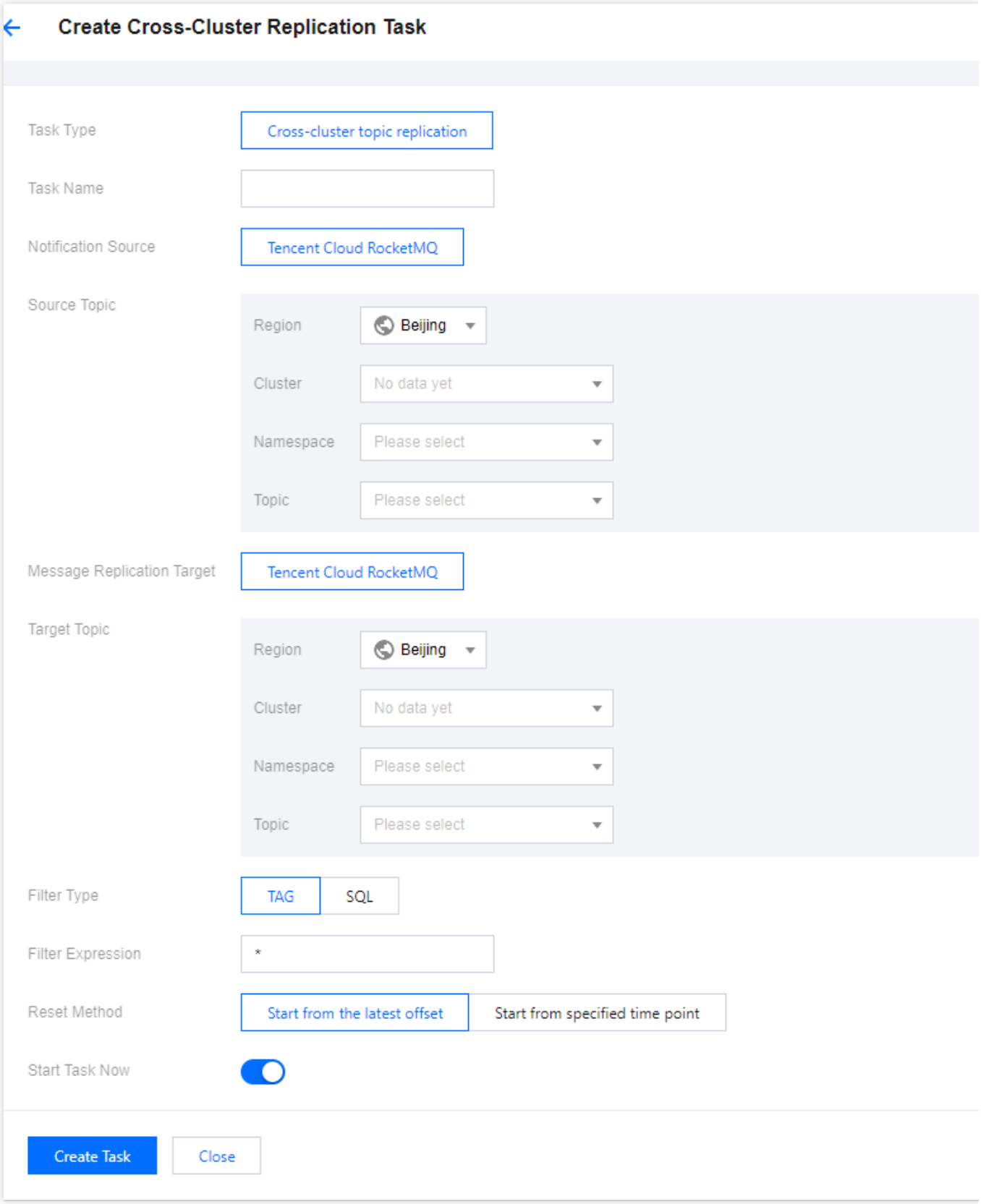
计费规则

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**跨集群复制**页面，单击页面上方的**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。

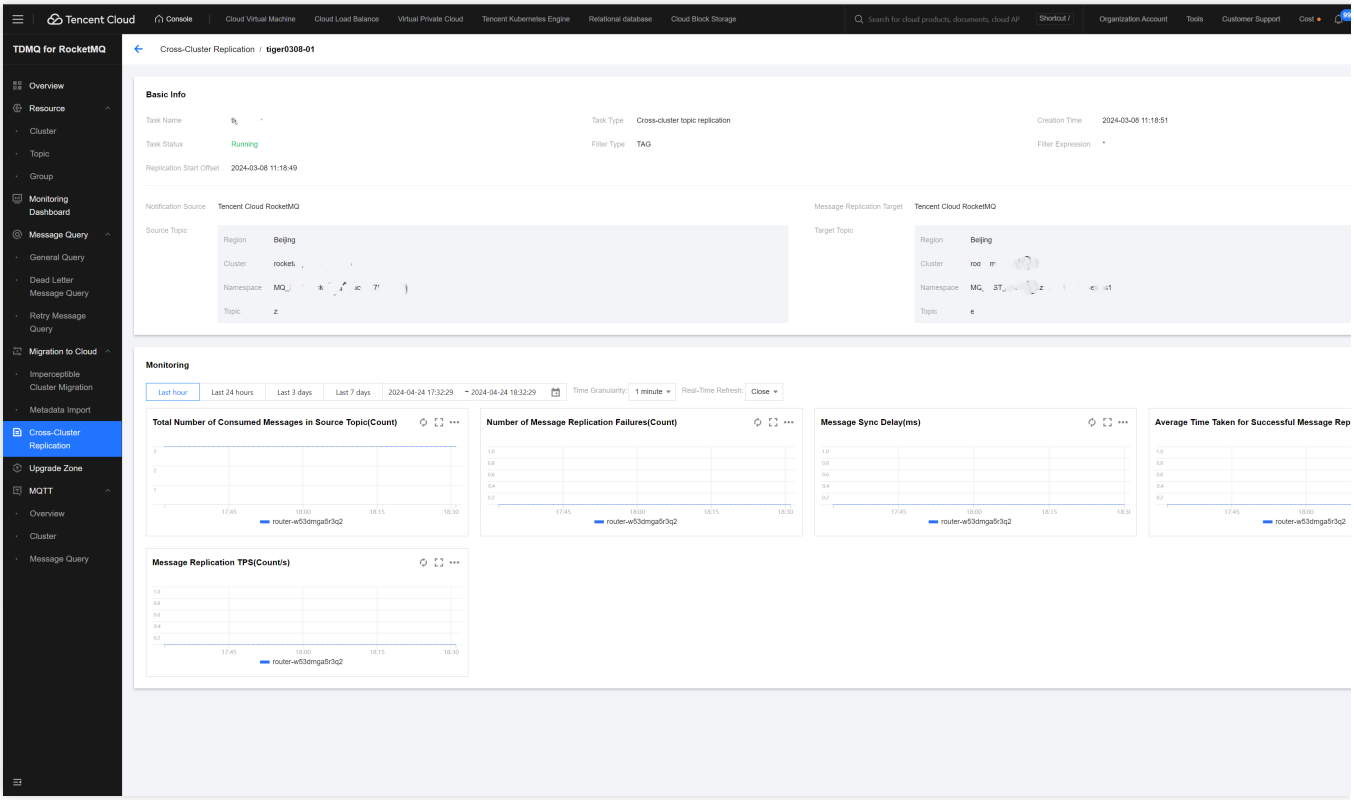
您创建的复制任务是单向的，即如果您创建一个 Topic A 到 Topic B 的复制任务，Topic A 的消息会自动复制到 Topic B；如果您需要双向的复制任务，您需要再次新建一个从 Topic B 到 Topic A 的复制任务。

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。

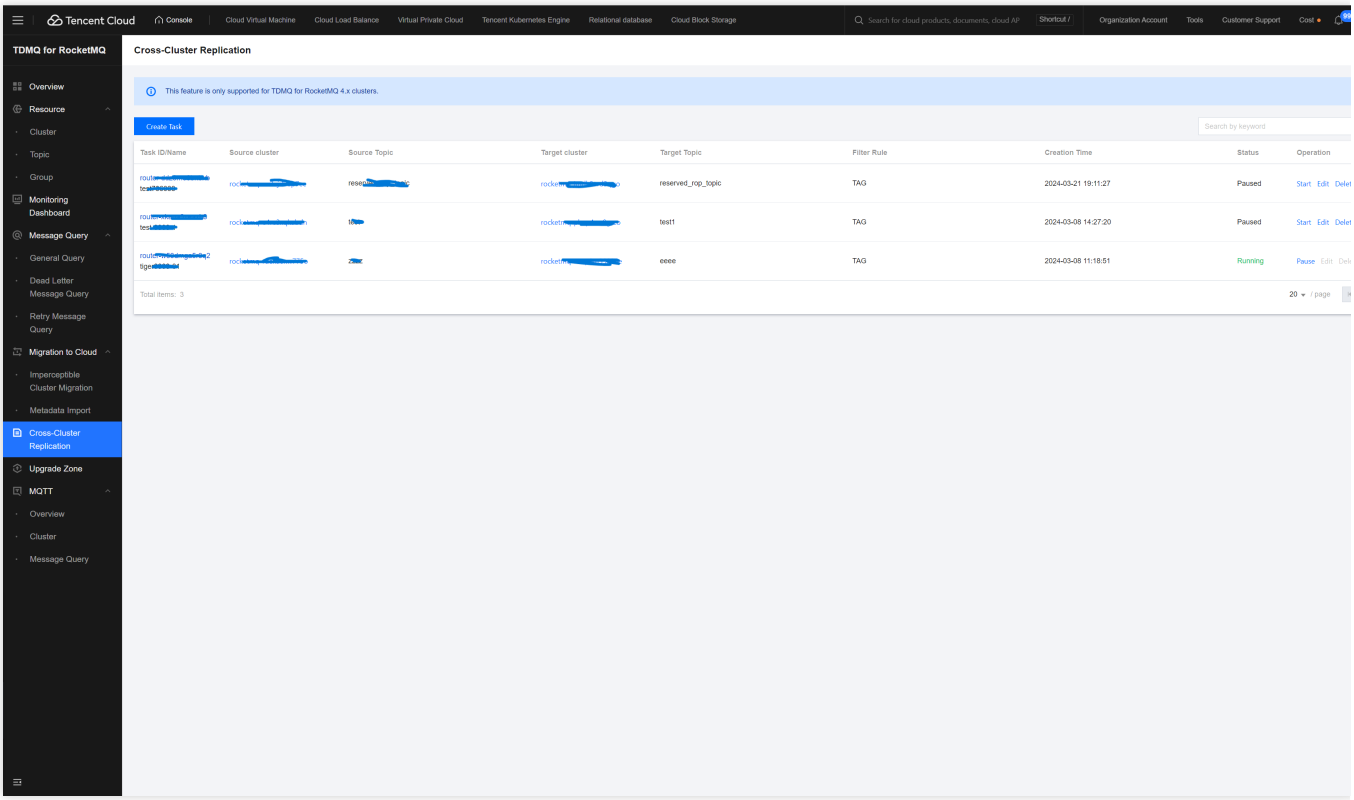
进行中的任务不能修改配置信息，如果要修改复制任务的配置，请先暂停任务后，单击操作栏的**编辑**，或者进入任务详情页，单击“基本信息”右上角的**编辑**，修改任务的信息。

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，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，比如复制成功的速率（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，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：2024-01-18 10:07:59

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。

定时与延时消息

最近更新时间：2023-09-12 16:10:04

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

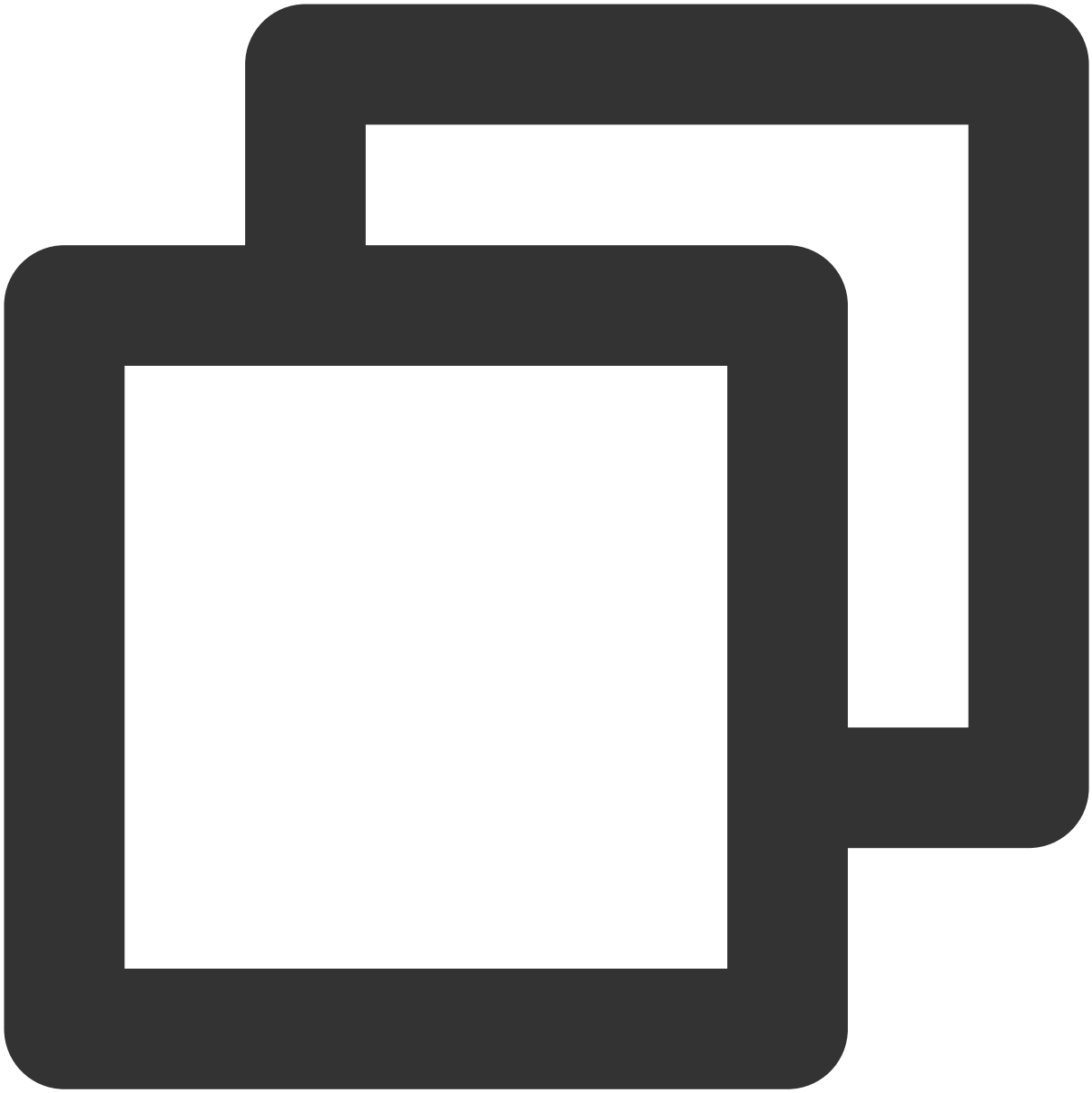
定时消息：消息在发送至服务端，实际业务并不希望消费端马上收到这条消息，而是推迟到某个时间点被消费，这类消息统称为定时消息。
延时消息：消息在发送至服务端，实际业务并不希望消费端马上收到这条消息，而是推迟一段时间后再被消费，这类消息统称为延时消息。
实际上，延时消息可以看成是定时消息的一种特殊用法，其实现的最终效果和定时消息是一致的。

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 >>

定时消息

定时消息直接在 message 发送前写入标准毫秒化的时间戳到 `__STARTDELIVERTIME` 属性即可。

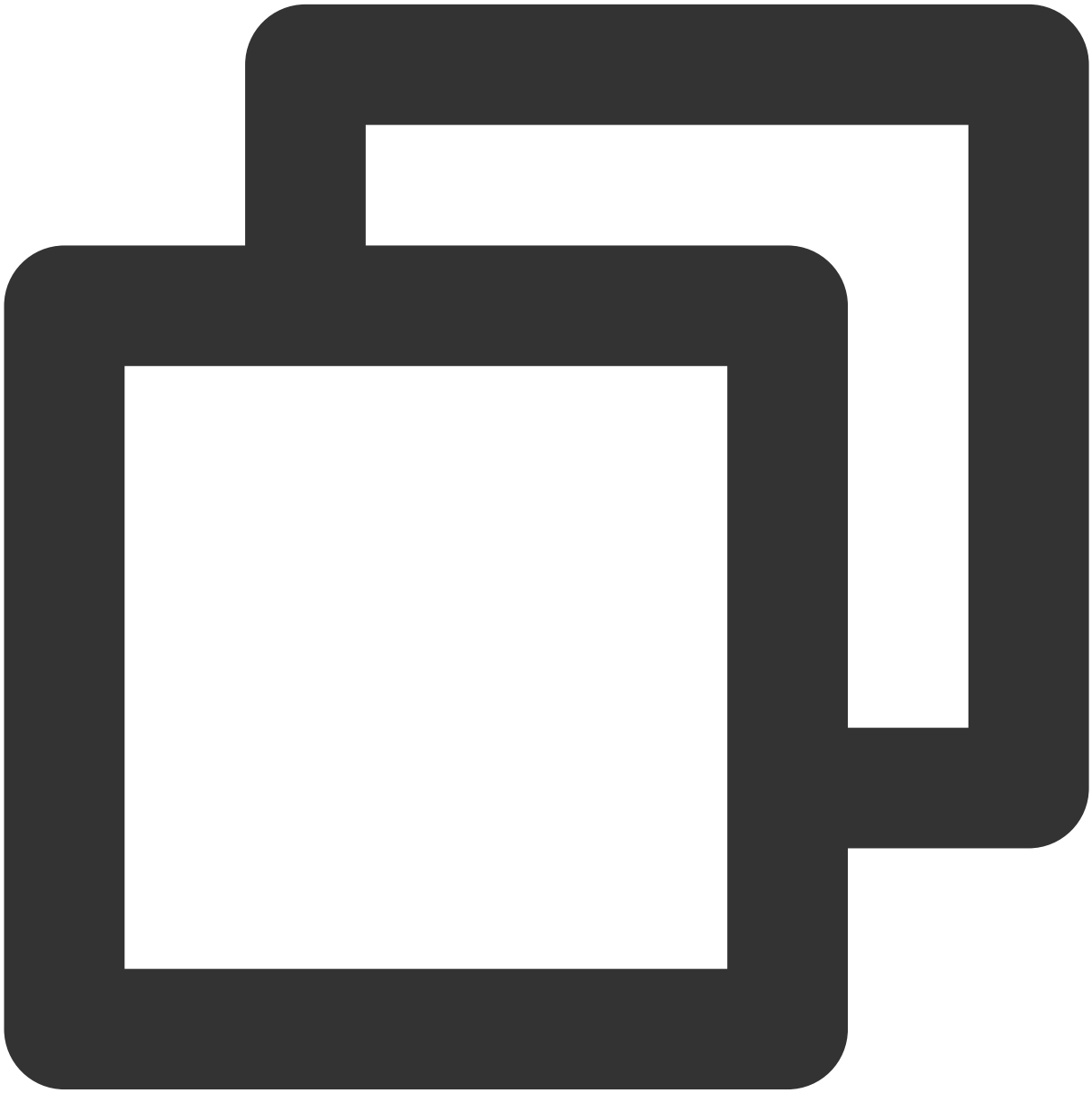


```
Message msg = new Message("test-topic", ("message content").getBytes(StandardCharse
// 设定消息在 2021-10-01 00:00:00 被发送
try {
    long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2021-10-01
    //将 __STARTDELIVERTIME 设定到 msg 的属性中
    msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(timeStamp));
    SendResult result = producer.send(msg);
    System.out.println("Send delay message: " + result);
} catch (ParseException e) {
    //TODO 添加对 Timestamp 解析失败的处理方法
    e.printStackTrace();
```

```
}
```

延时消息

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。



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

// 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);
```

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，最大均为40天。
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：2023-09-12 16:31:51

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：

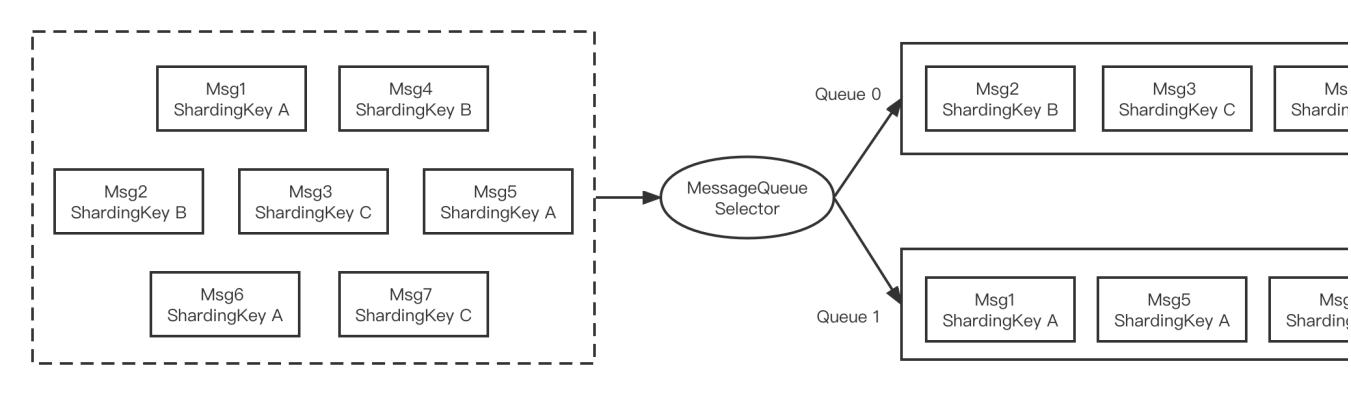
消息类型	消费顺序	性能	适用场景
普通消息	无顺序	高	适用于对吞吐量要求高，且对生产和消费顺序无要求
顺序消息	指定的 Topic 内的消息遵循先入先出（FIFO）规则	一般	吞吐量要求一般，但是要求特定的 Topic 严格地按照 FIFO 原则进行消息发布和消费的场景

对应到具体的业务场景，顺序消息可以被用在以下场景中：

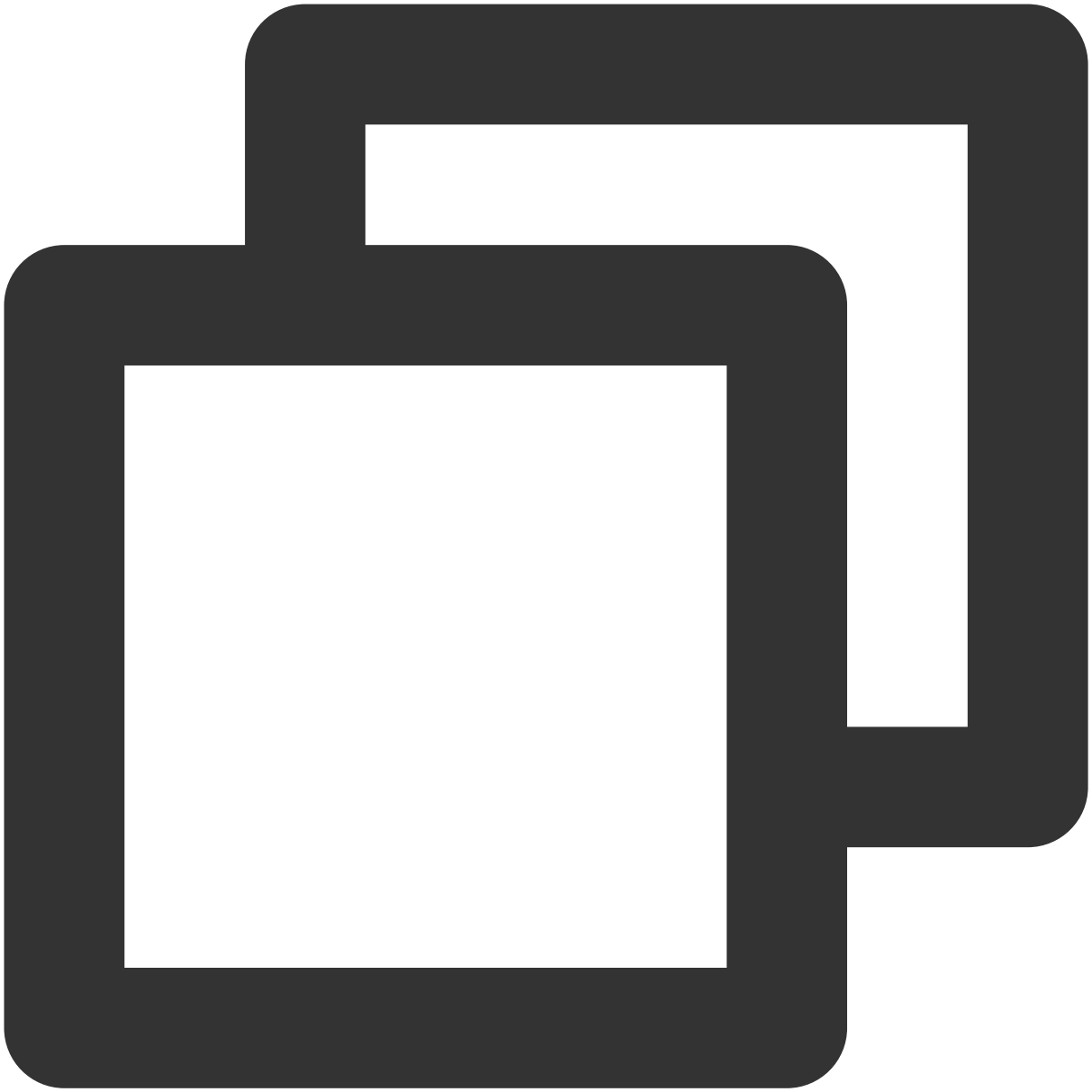
订单创建场景：在一些电商系统中，同一个订单相关的创建订单消息、订单支付消息、订单退款消息、订单物流消息必须严格按照先后顺序来进行生产或者消费，否者消费中传递订单状态会发生紊乱，影响业务的正常进行。因此，该订单的消息必须按照一定的顺序在客户端和消息队列中进行生产和消费，同时消息之间有先后的依赖关系，后一条消息需要依赖于前一条消息的处理结果。
日志同步场景：在有序事件处理或者数据实时增量同步的场景中，顺序消息也能发挥较大的作用，如同步 mysql 的 binlog 日志时，需要保证数据库的操作是有顺序的。
金融场景：在一些撮合交易的场景下，比如某些证券交易，在价格相同的情况下，先出价者优先处理，则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



顺序消息的代码如下所示：

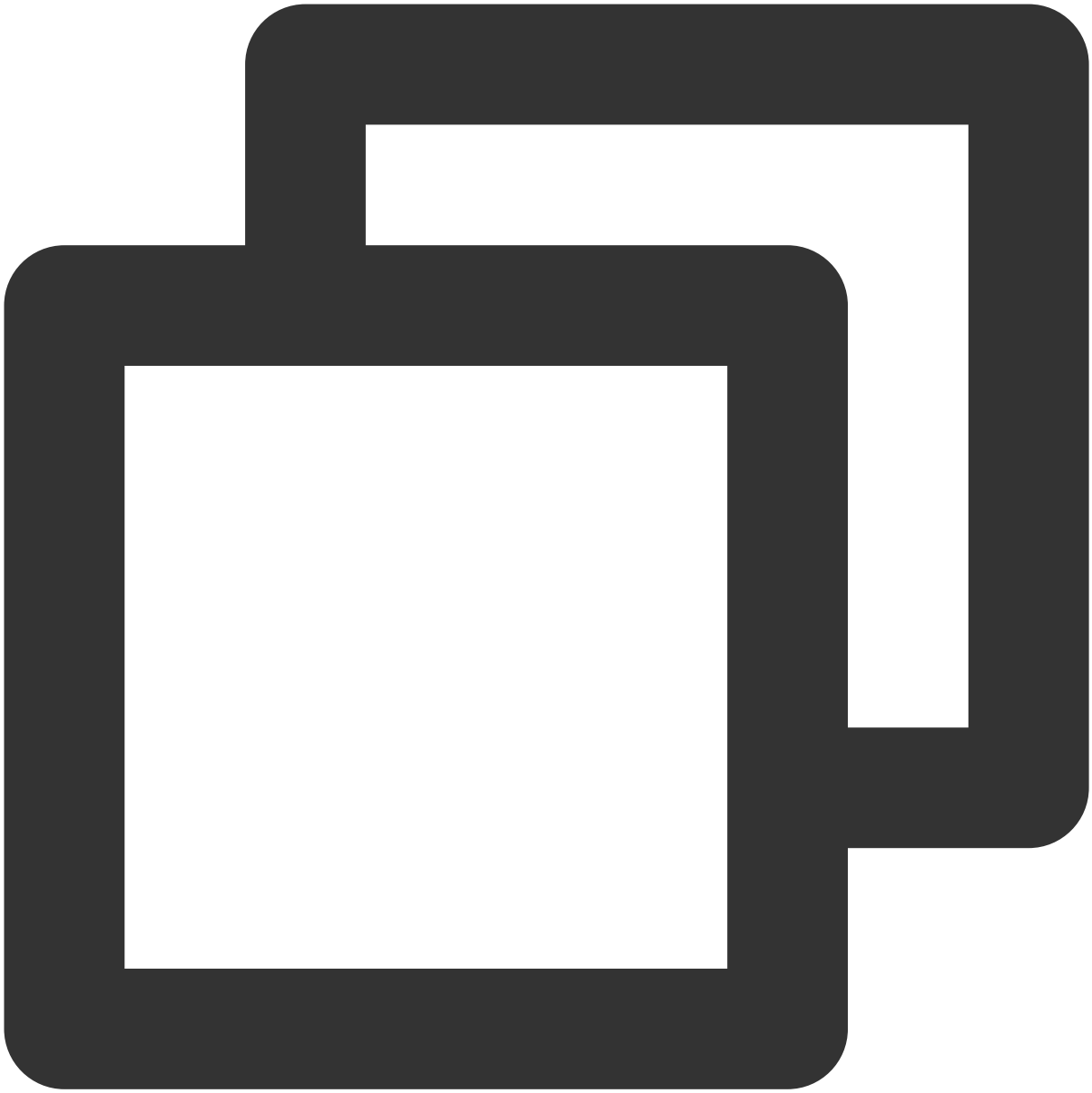


```
public class Producer {
    public static void main(String[] args) throws UnsupportedEncodingException {
        try {
            DefaultMQProducer producer = new DefaultMQProducer("please_rename_uniqu
            producer.start();

            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_CHA
                SendResult sendResult = producer.send(msg, new MessageQueueSelector
                    @Override
                    public MessageQueue select(List<MessageQueue> mqs, Message msg,
                        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 | Interr
            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 Obje
}
```

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。

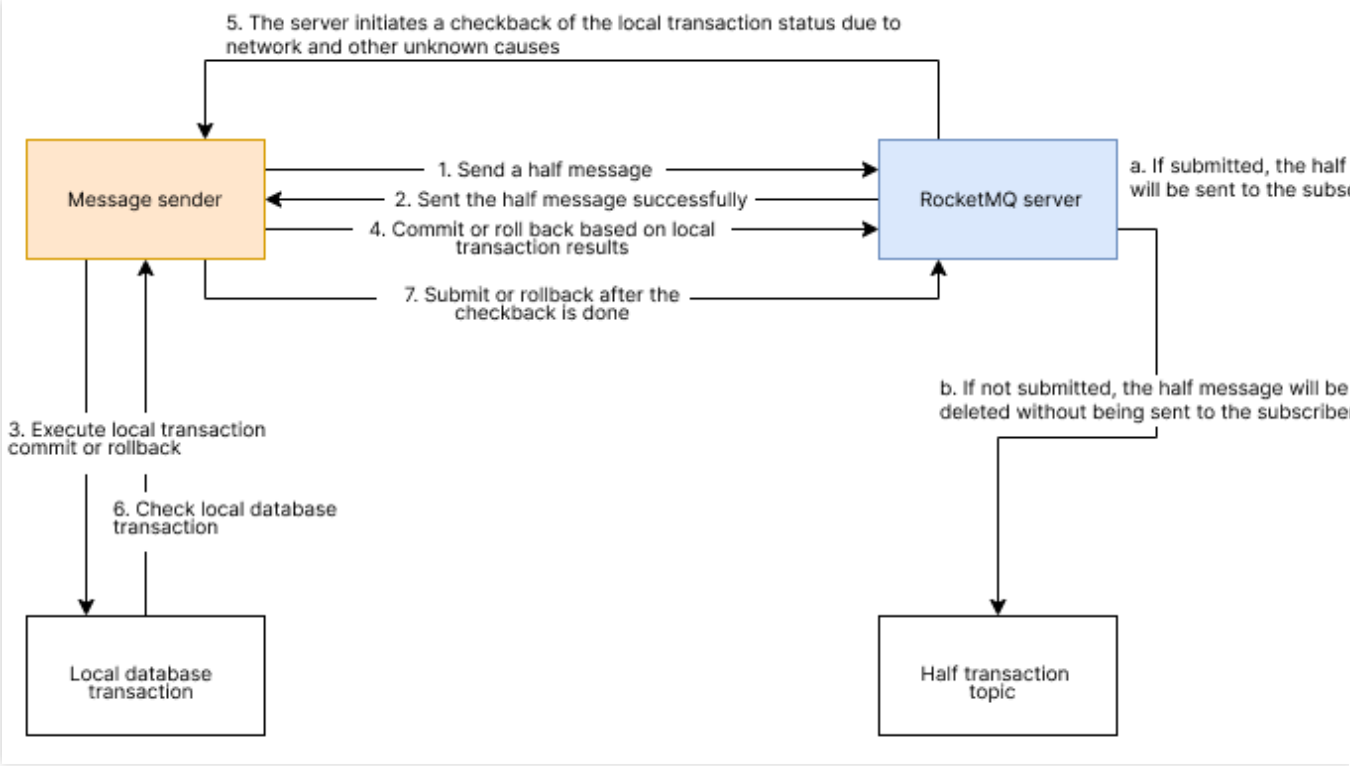
事务消息

最近更新时间：2023-04-14 16:59:04

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（a）。

应用场景

使用 TDMQ RocketMQ 版的事务消息来处理交易事务，可以极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：2023-10-19 11:02:36

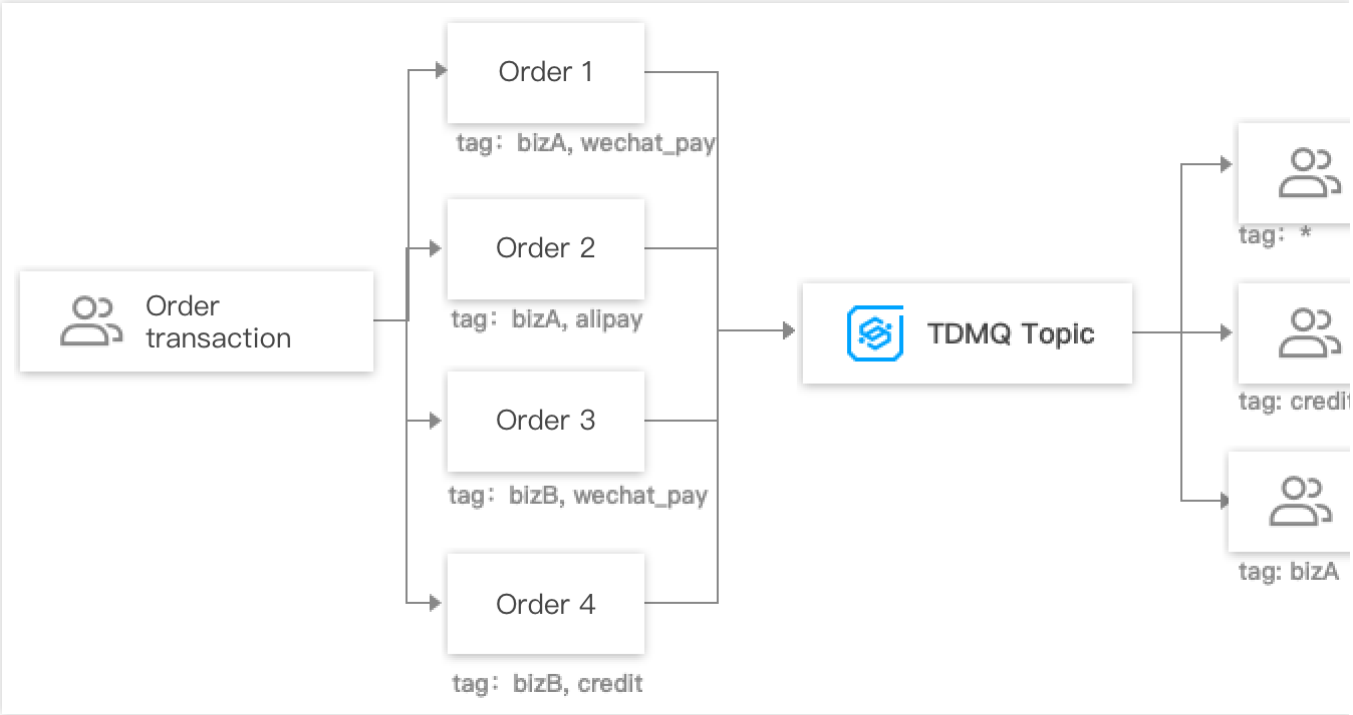
本文主要介绍 TDMQ RocketMQ 版中消息过滤的功能、应用场景和使用方式。

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。

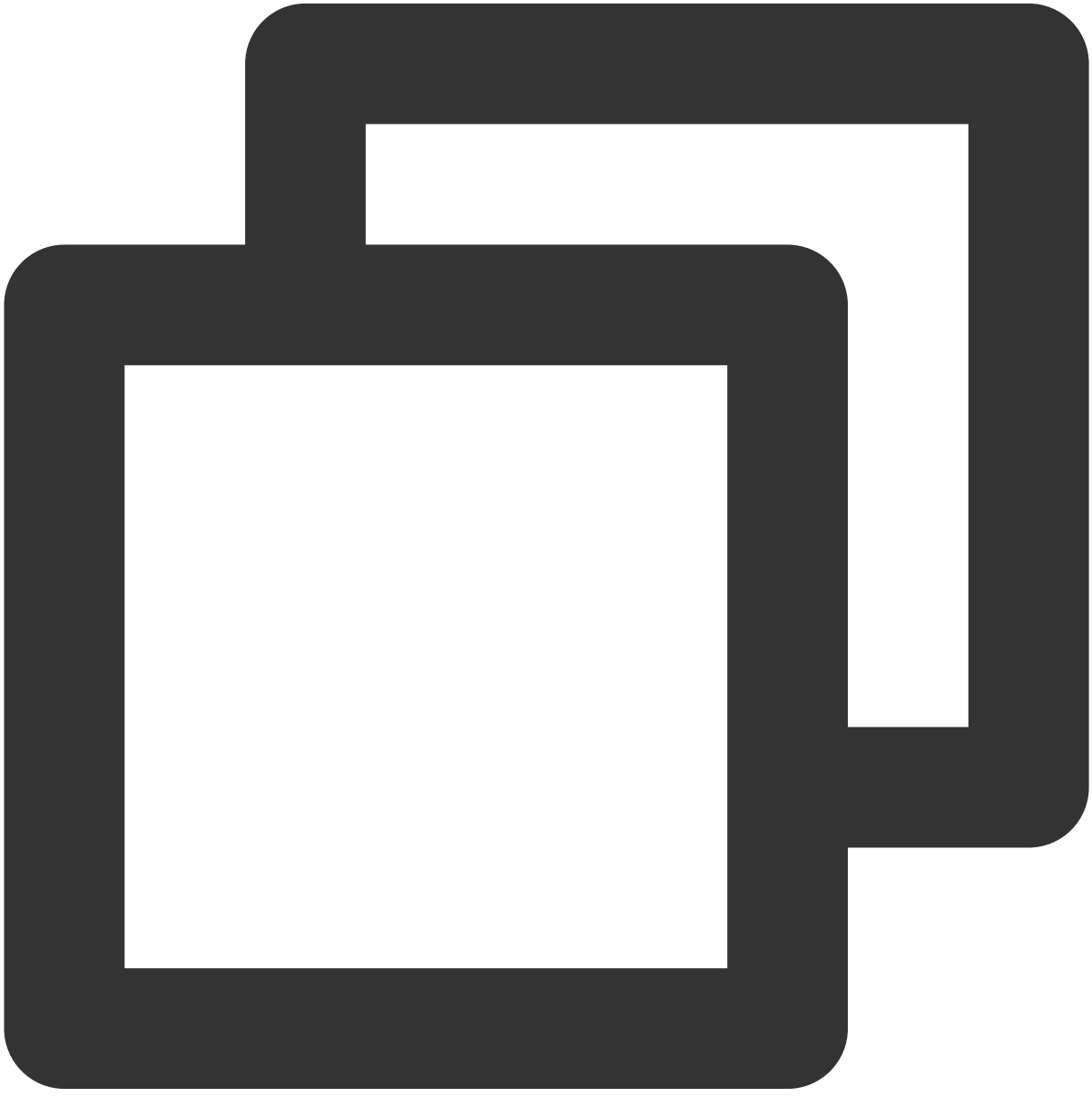


使用方式

TAG 过滤

发送消息

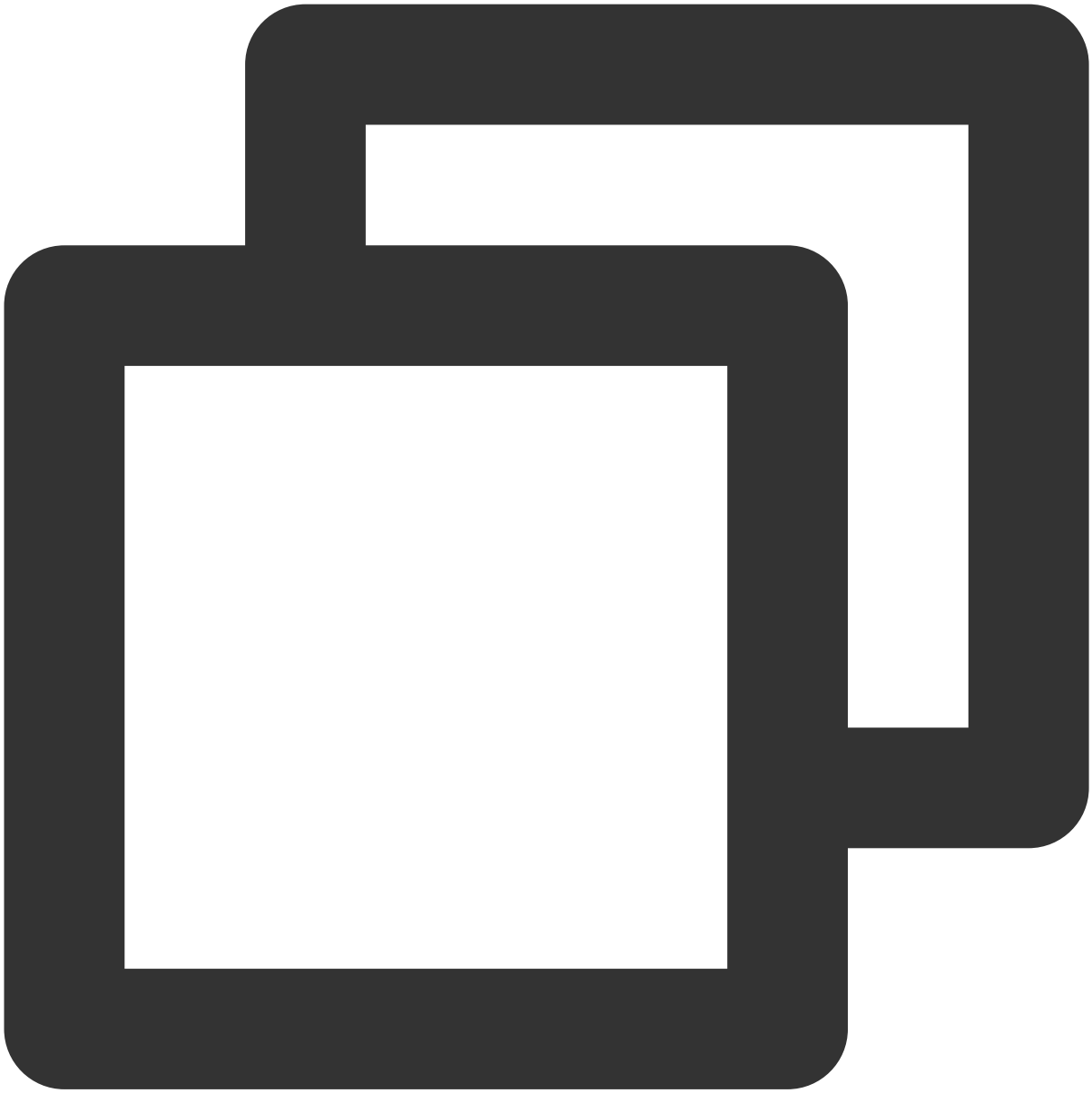
发送消息时，每条消息必须指明 Tag。



```
Message msg = new Message("TOPIC","TagA","Hello world".getBytes());
```

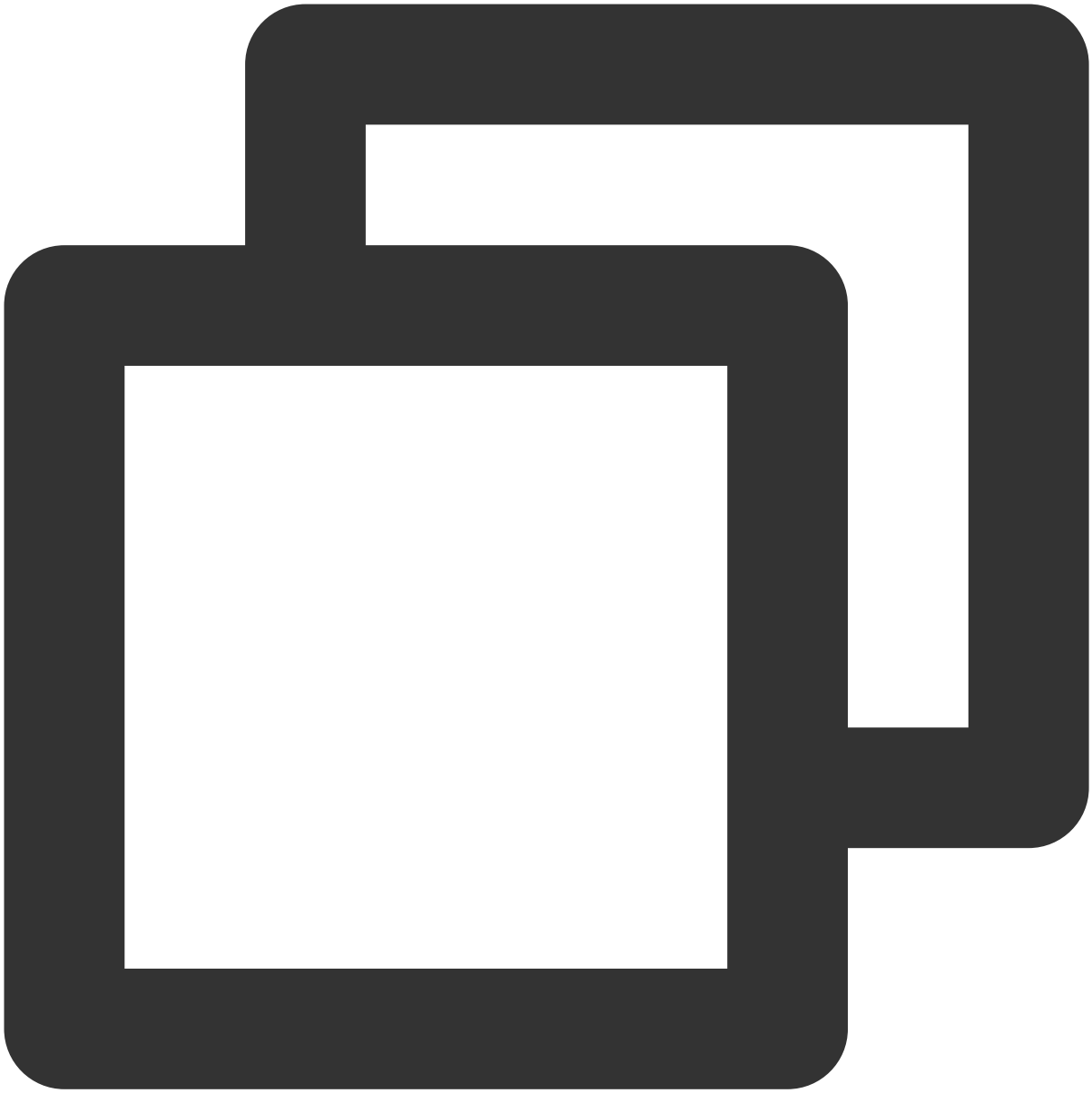
订阅消息

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。



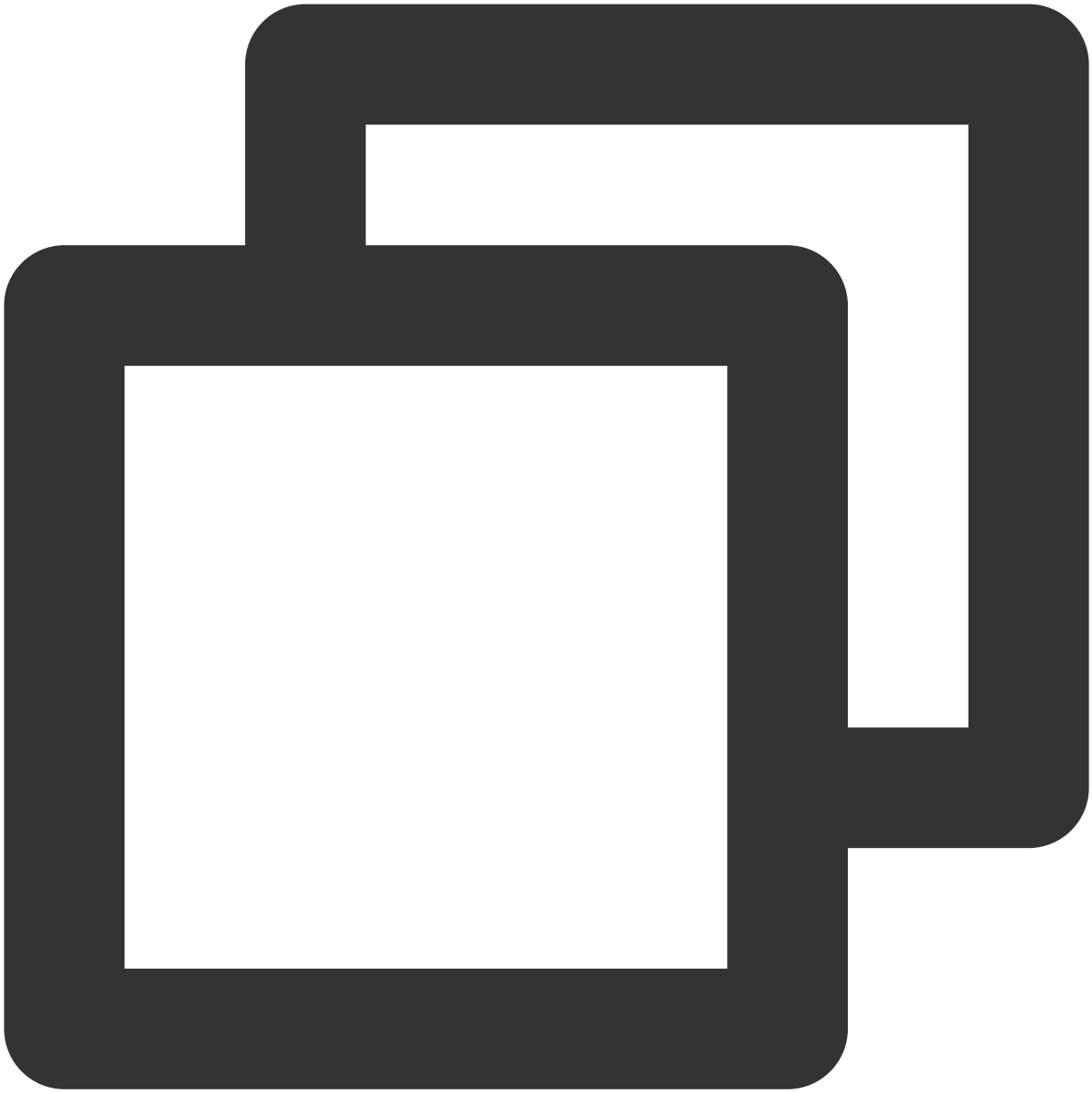
```
consumer.subscribe("TOPIC", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

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



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

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

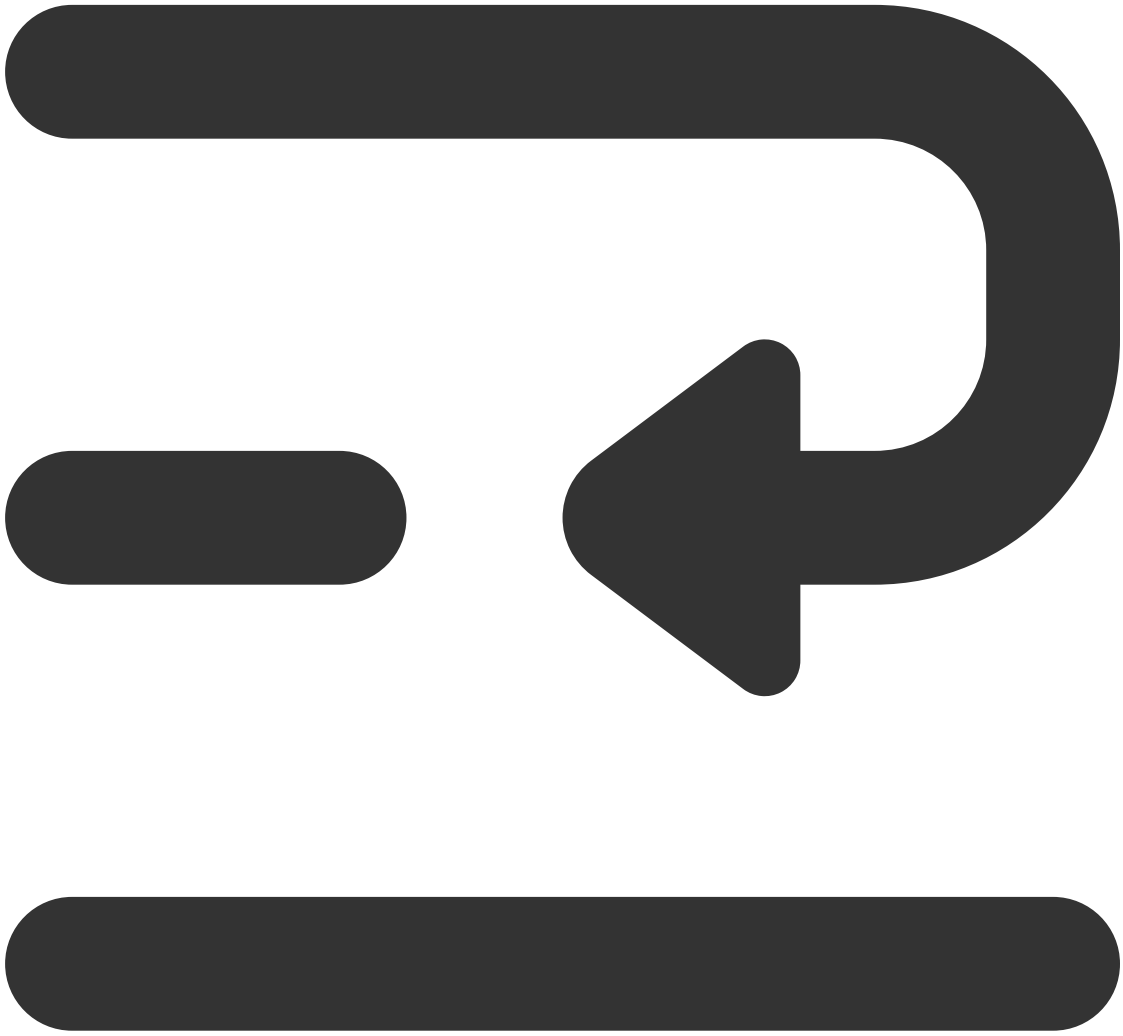


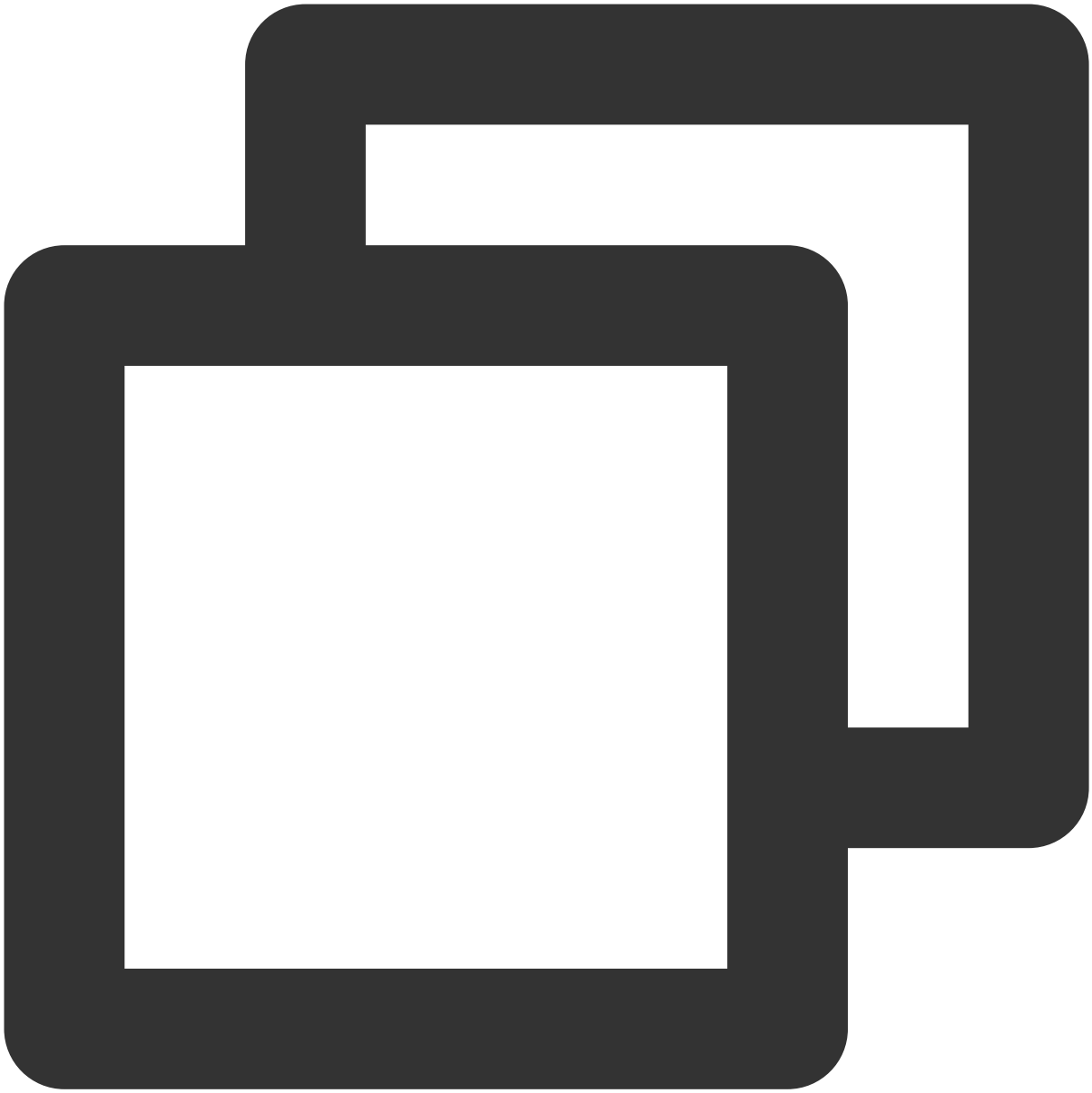
```
consumer.subscribe("TOPIC", "TagA||TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

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。

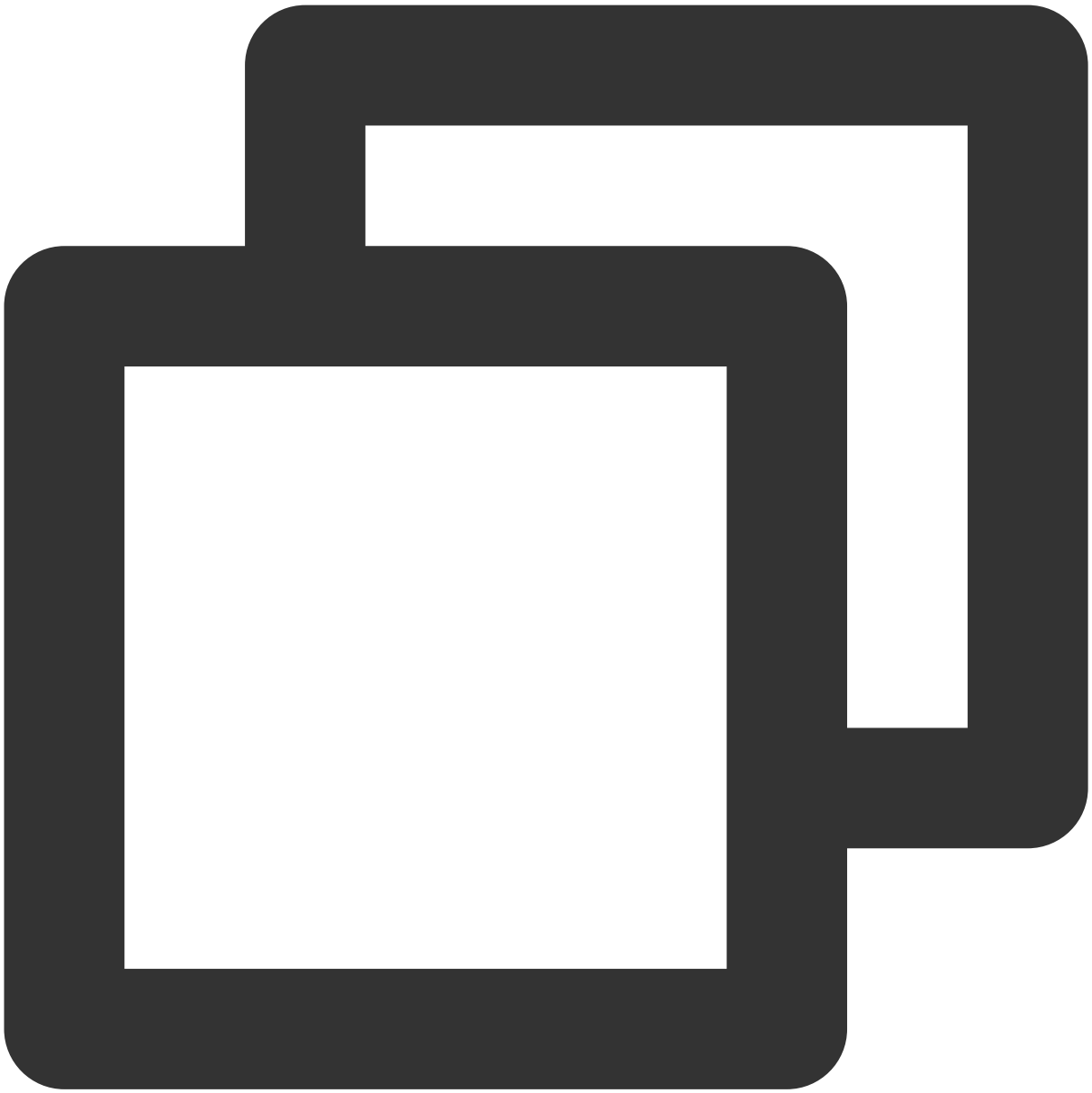




```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME,"Hello RocketMQ.".getBytes(StandardCharset
    msg.putUserProperty("key1","value1");
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

订阅消息

对于消费消息，主要是订阅的时候，带上对应的SQL表达式，其他的和普通的消费消息流程没有区别。



```
//订阅所有消息
pushConsumer.subscribe(TOPIC_NAME, MessageSelector.bySql("True"));

// 订阅topic 订阅单个key的sql，最常用
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND

//订阅多个属性
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND
```

```
// 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(
        // 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();
```

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 。

消费模式

最近更新时间：2023-09-12 16:38:00

本文主要介绍消息队列 TDMQ RocketMQ 版中集群消费和广播消费的相关功能和应用场景。

功能介绍

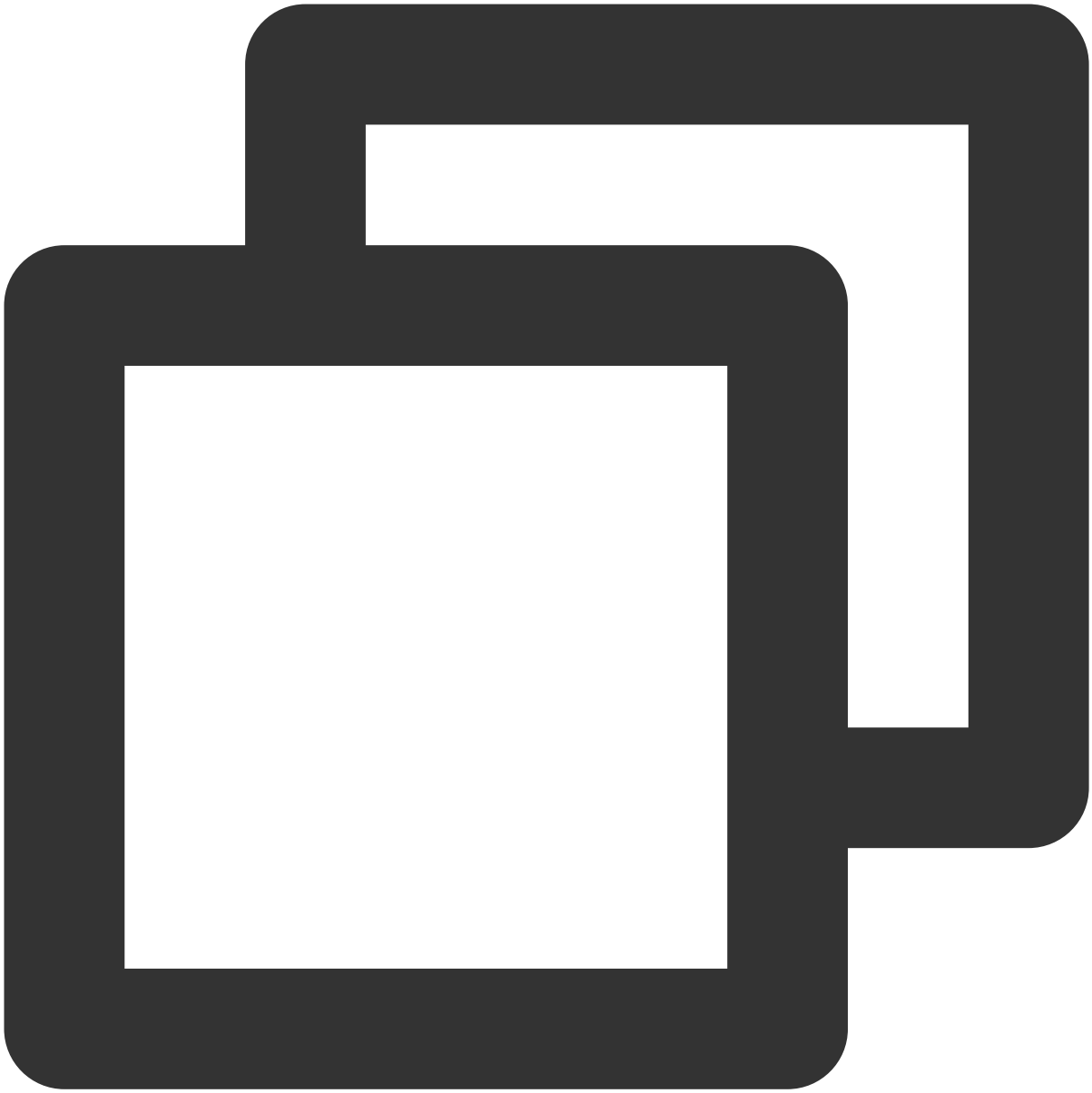
集群消费：当使用集群消费模式时，任意一条消息只需要被集群内的任意一个消费者处理即可。
广播消费：当使用广播消费模式时，每条消息会被推送给集群内所有注册过的消费者，保证消息至少被每个消费者消费一次。

应用场景

集群消费：适用于每条消息只需要被处理一次的场景。
广播消费：适用于每条消息需要被集群下每一个消费者处理的场景。

代码示例

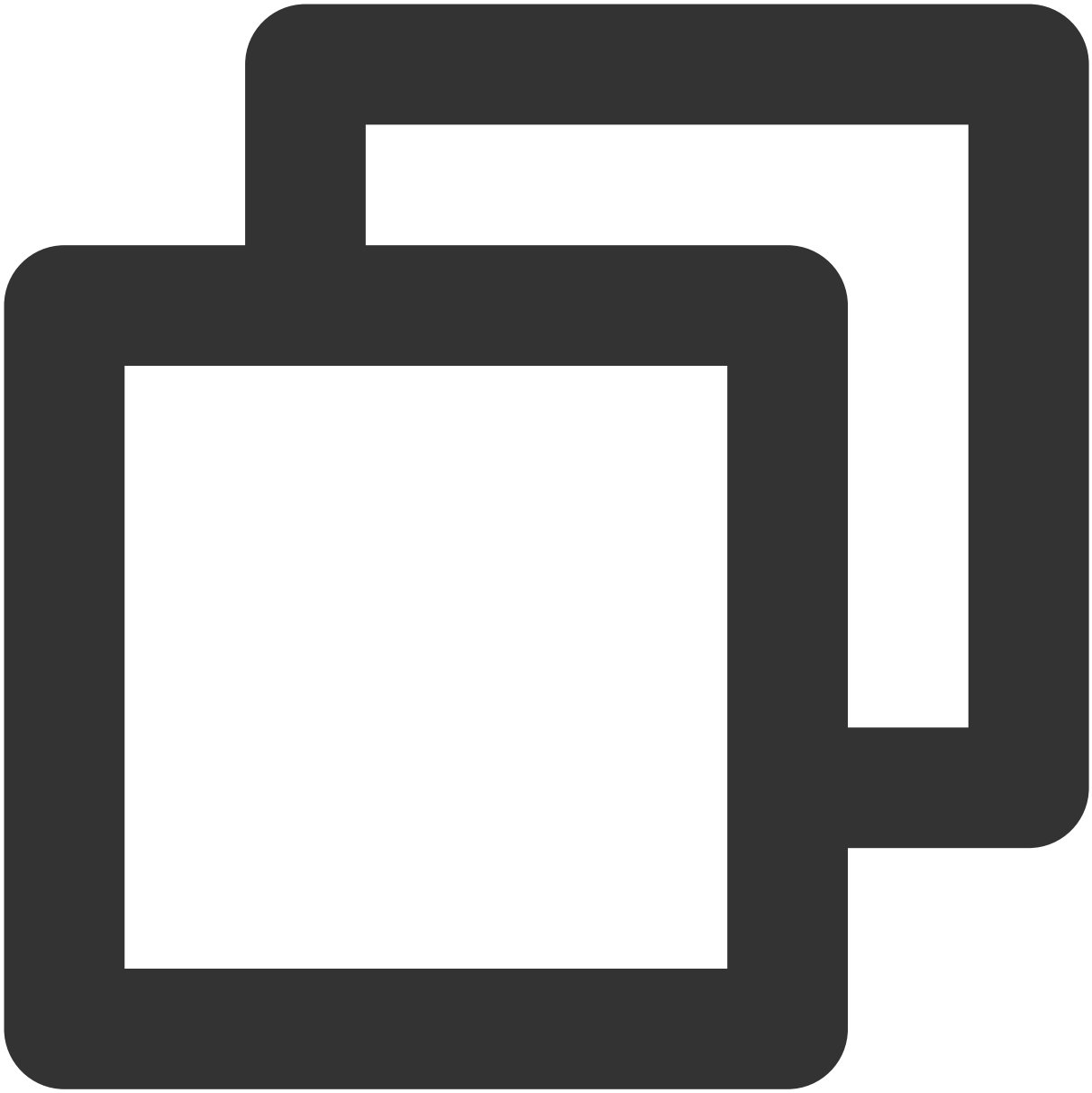
集群订阅
同一个 Group ID 所标识的所有 Consumer 平均分摊消费消息。例如某个 Topic 有9条消息，一个 Group ID 有3个 Consumer 实例，那么在集群消费模式下每个实例平均分摊，只消费其中的3条消息。



```
// 集群订阅方式设置（不设置的情况下，默认为集群订阅方式）。
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
```

广播订阅

同一个 Group ID 所标识的所有 Consumer 都会各自消费某条消息一次。例如某个 Topic 有9条消息，一个 Group ID 有3个 Consumer 实例，那么在广播消费模式下每个实例都会各自消费9条消息。



```
// 广播订阅方式设置。
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING);
```

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。

消息重试

最近更新时间：2023-09-12 16:40:42

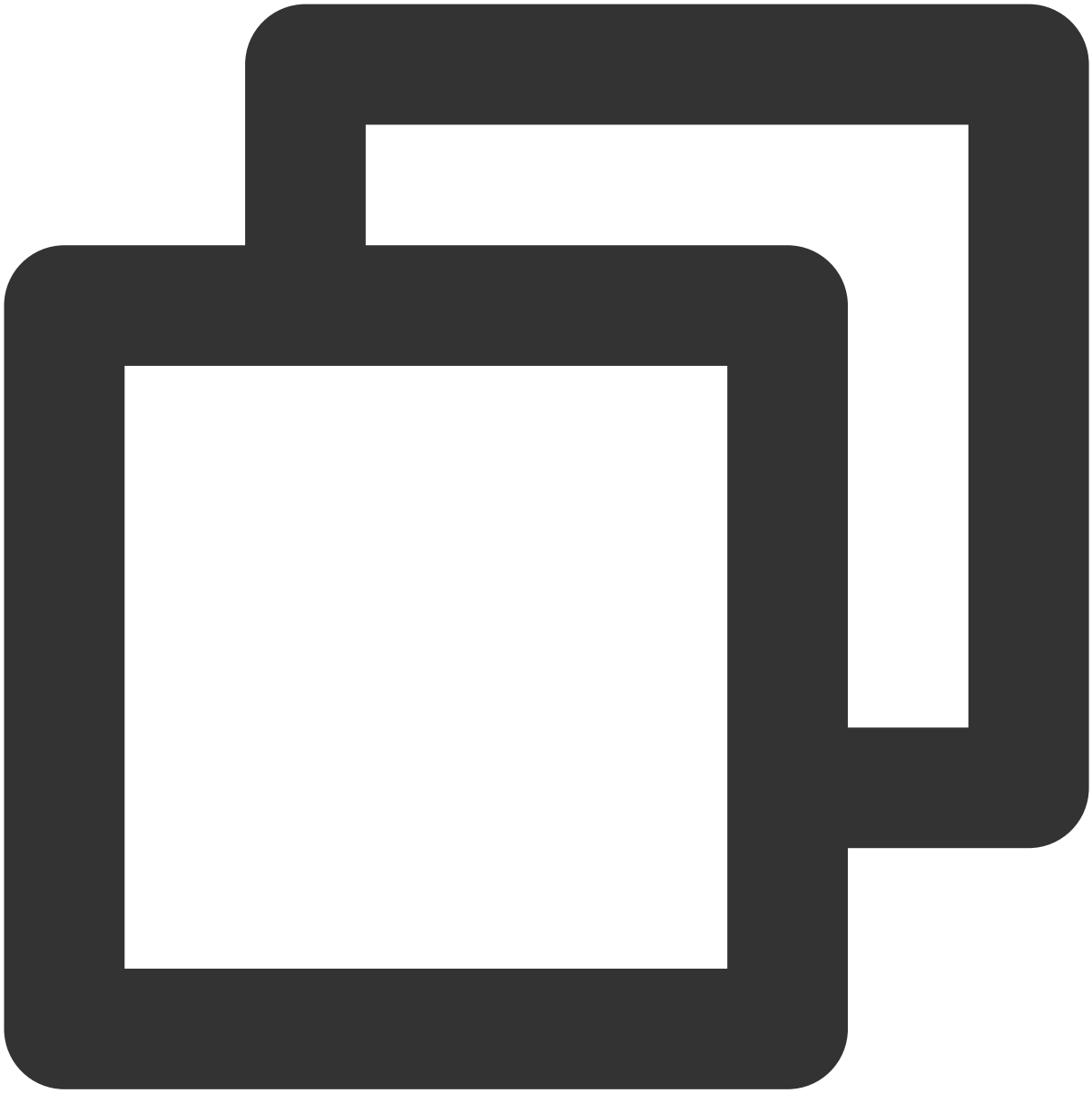
本文主要介绍消息队列 TDMQ RocketMQ 版中消息重试与使用方法。

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：
1. 消费者返回 ConsumeConcurrentlyStatus.RECONSUME_LATER。
2. 消费者返回 null。
3. 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
```

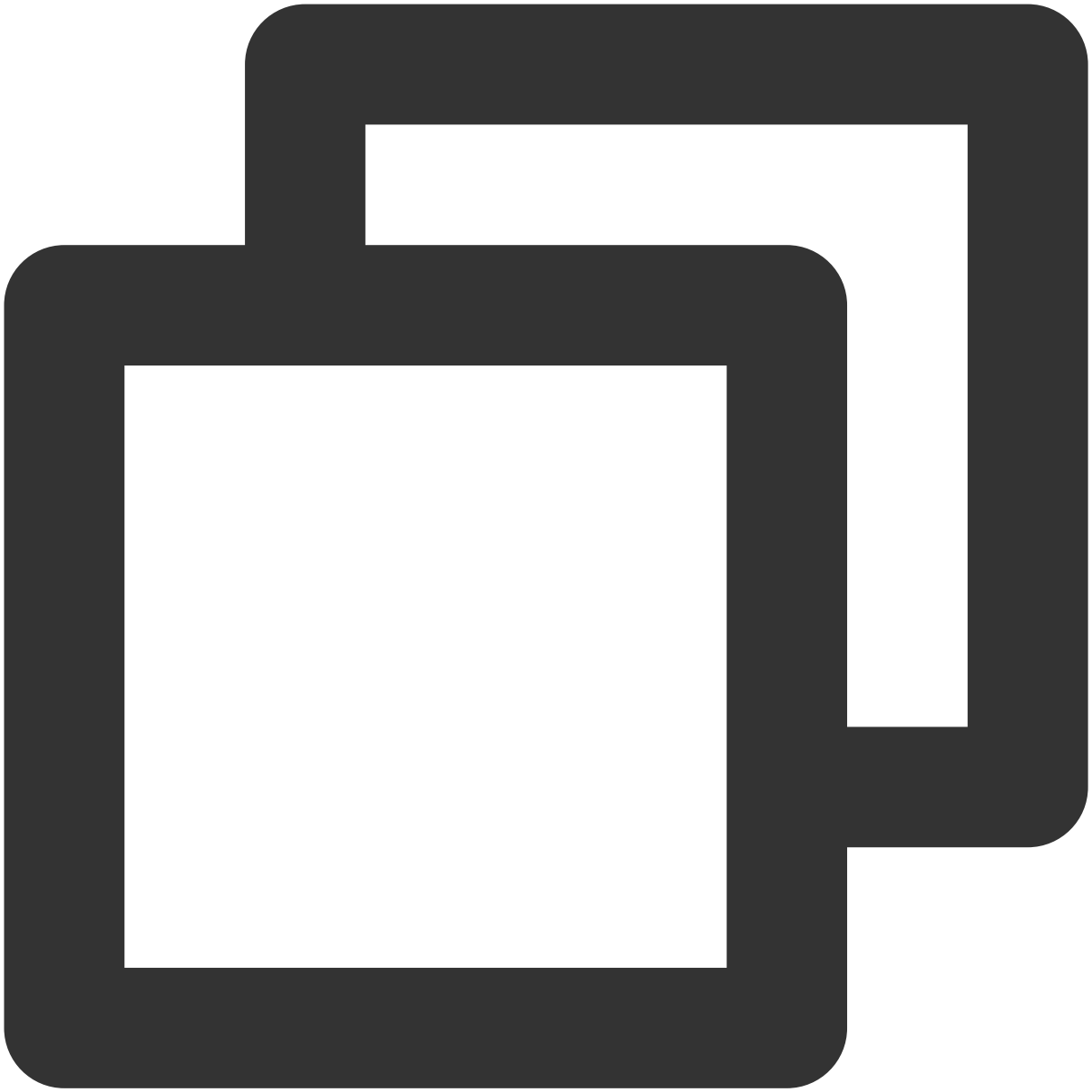
重试次数与重试时间间隔关系如下：

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	1秒	10	6分钟
2	5秒	11	7分钟
3	10秒	12	8分钟

4	30秒	13	9分钟
5	1分钟	14	10分钟
6	2分钟	15	20分钟
7	3分钟	16	30分钟
8	4分钟	17	1小时
9	5分钟	18	2小时

使用方式

如果用户需要自行调整这个重试次数。可以通过设置 consumer 的参数来决定。



```
pushConsumer.setMaxReconsumeTimes(3);
```

死信队列

最近更新时间：2023-09-12 16:41:57

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。

特性说明

不同于重试队列，会自动消费，死信队列的消息需要有新的代码处理，或者人工介入
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。

SDK 文档
TCP 协议接入
Spring Boot Starter 接入
发送与接收普通消息

最近更新时间：2023-10-19 11:11:07

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。

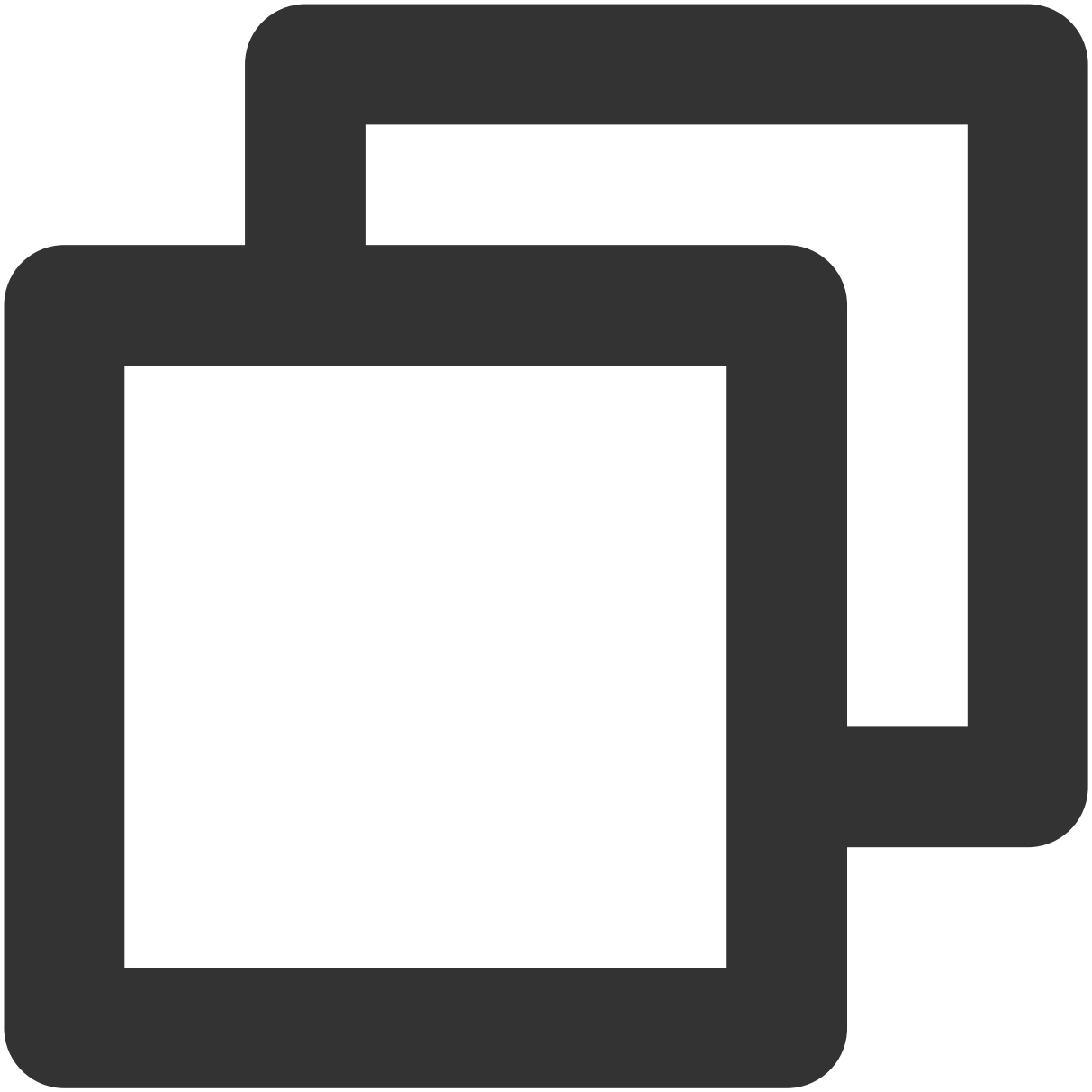
前提条件

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

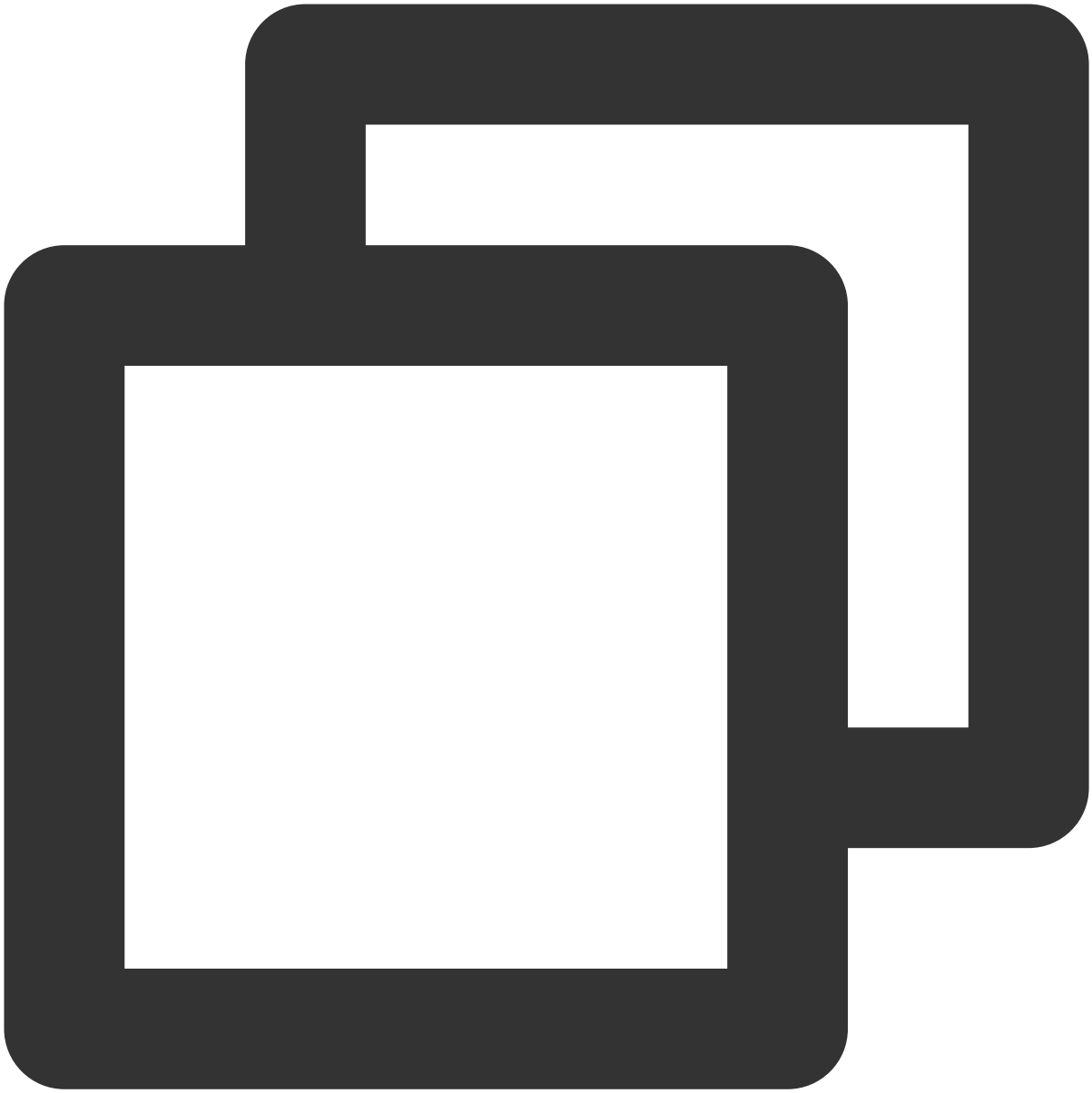
在 pom.xml 中添加依赖。



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

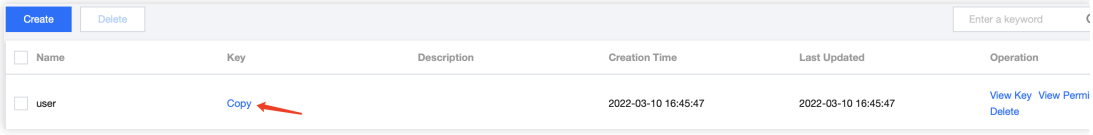
步骤2：准备配置

在配置文件中添加配置信息。



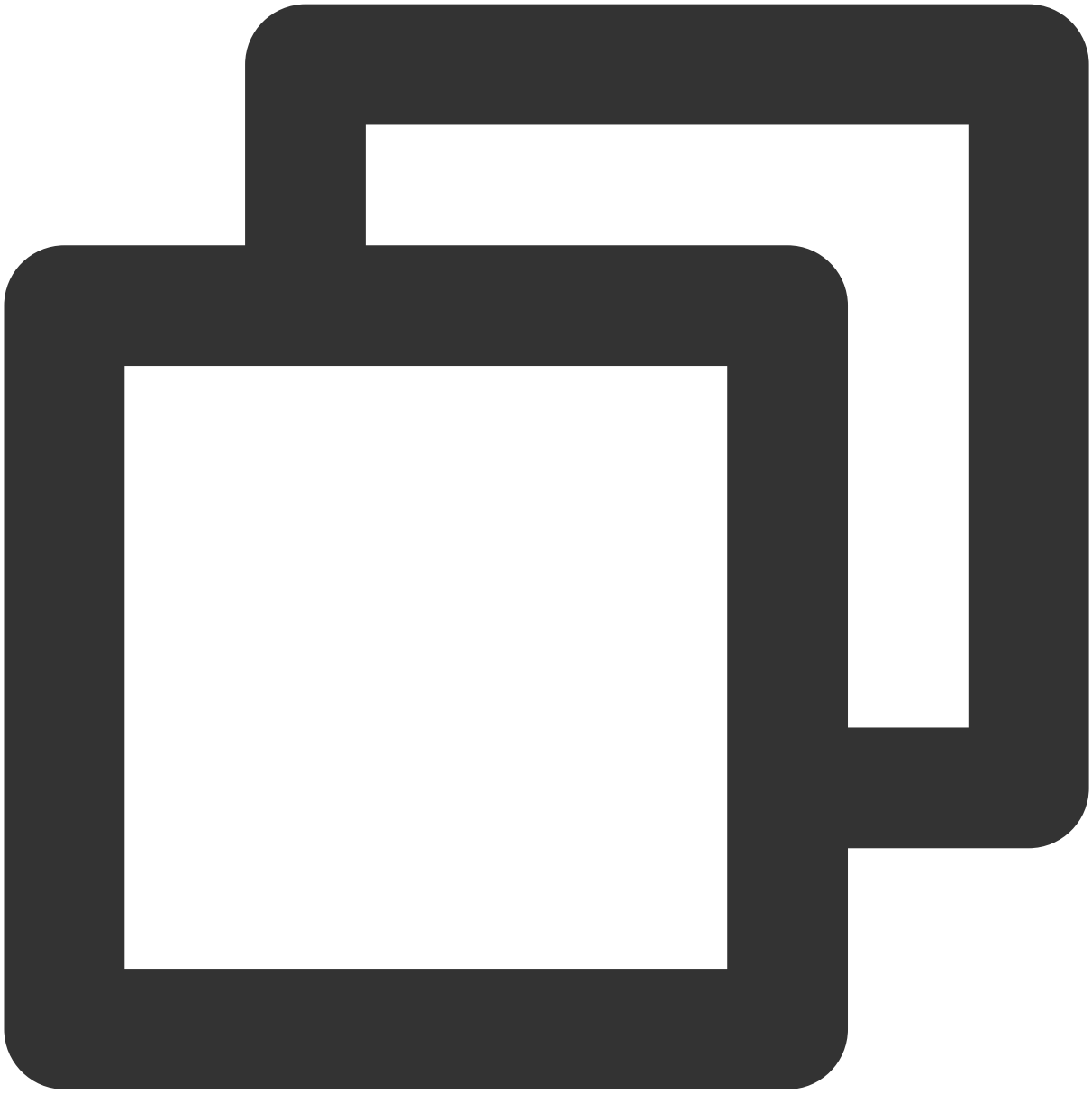
```
server:
  port: 8082

#rocketmq配置信息
rocketmq:
  # tdmq-rocketmq服务接入地址
  name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
  # 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

  # 自定义配置，根据业务进行配置
  namespace: rocketmq-xxx|namespace1
  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 名接入点地址在命名空间列表获取。
group	生产者 Group 的名称，在控制台 Group 页面复制。
secret-key	角色名称，在 角色管理 页面复制。
access-key	角色密钥，在 角色管理 页面复制密钥列复制。 
namespace	命名空间的名称，在控制台命名空间页面复制。
topic	Topic 的名称，在控制台 topic 页面复制。
subExpression	用来设置消息的 TAG。

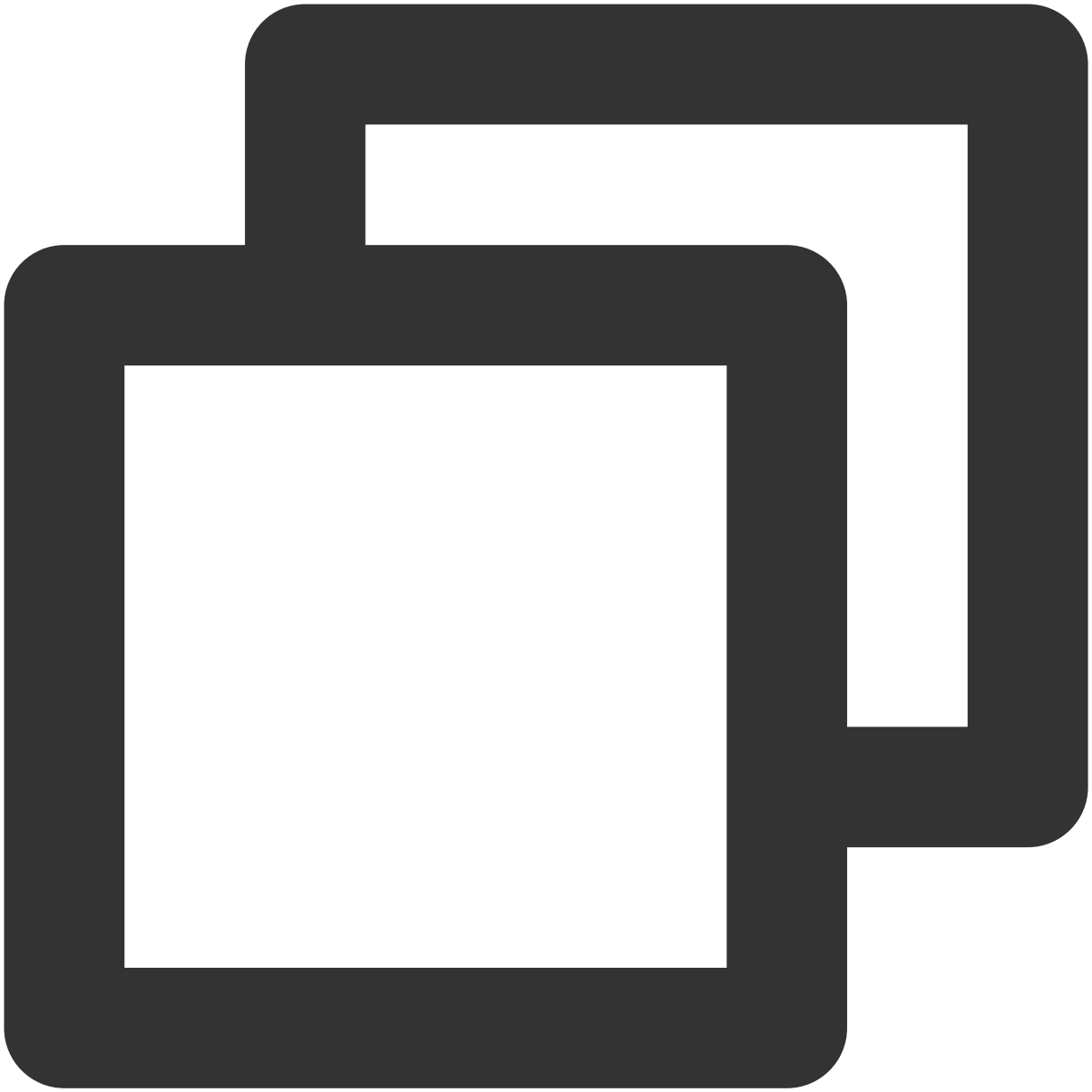
步骤3：发送消息

1. 在需要发送消息的类中注入 `RcoketMQTemplate` 。



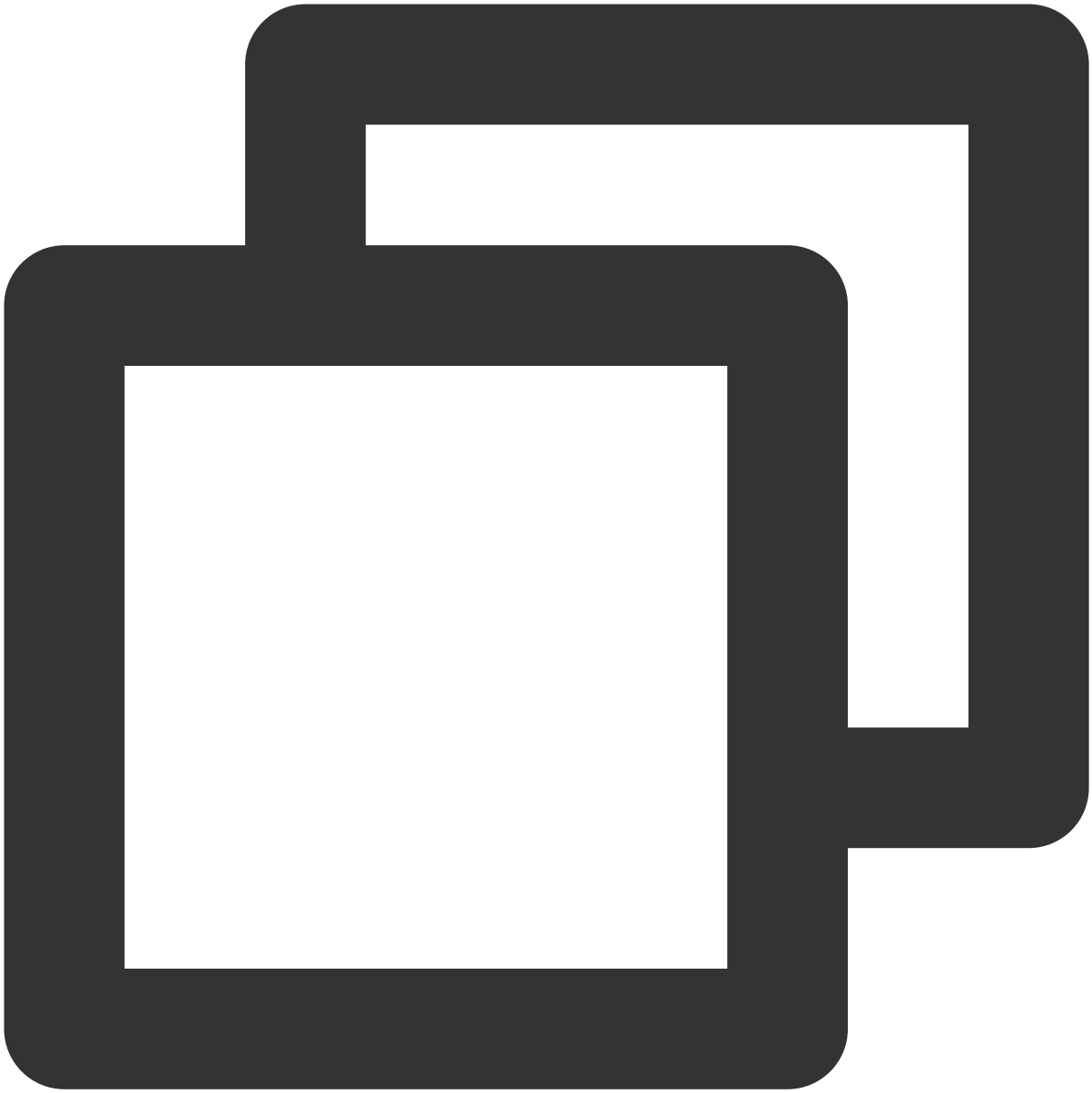
```
@Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
      private String topic;  // topic名称 (需要使用topic全称，所以在这里对topic名称进行拼接

      @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。



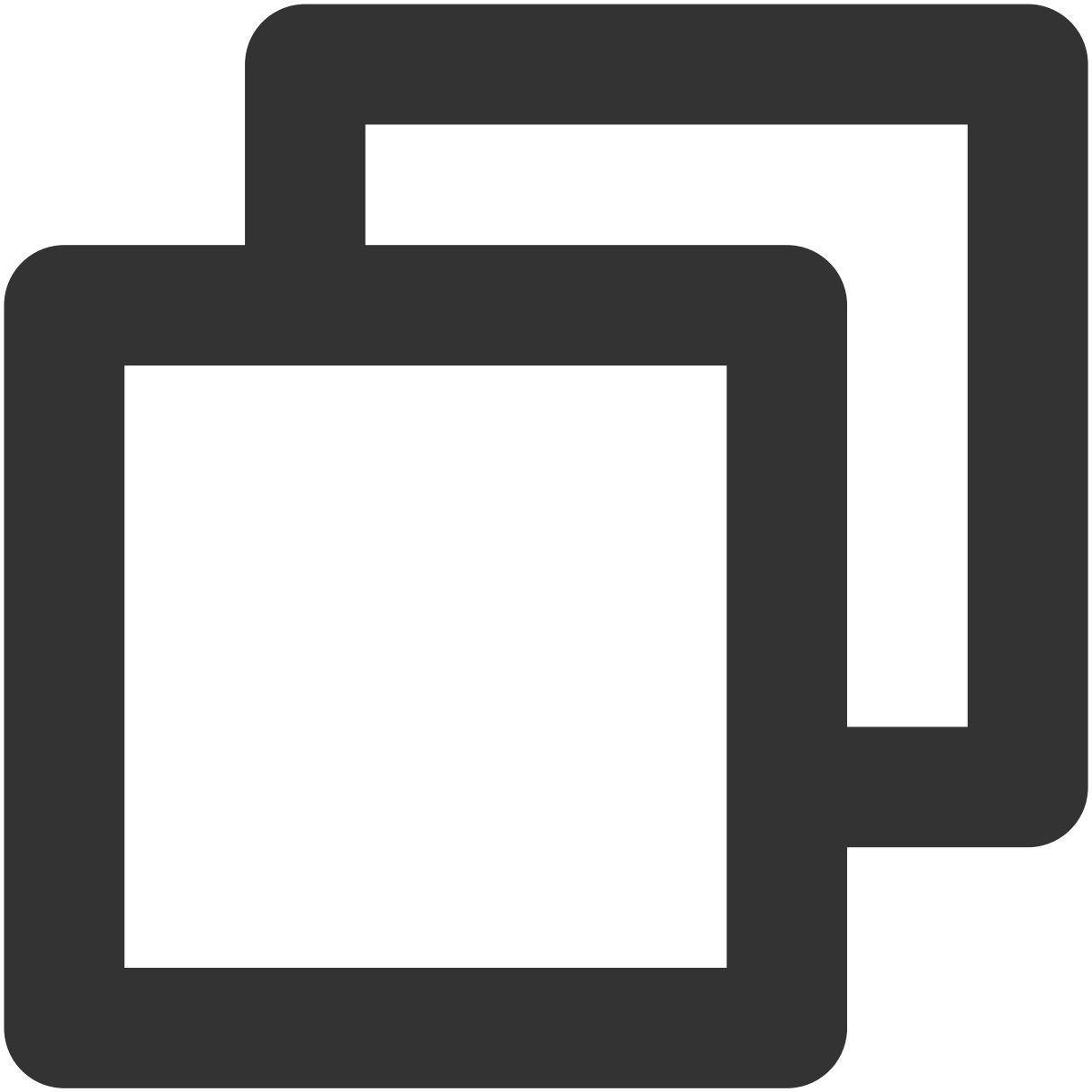
```
/**
* Description: 消息生产者
*/
@Service
public class SendMessage {
// 需要使用topic全称，所以进行topic名称的拼接，也可以自己设置　格式：namespace全称%topic名称
 @Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
 private String topic;
     @Autowired
 private RocketMQTemplate rocketMQTemplate;
         /**
```

```
     * 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: `
            String destination = StringUtils.isBlank(tags) ? topic : topic + ":" + tag
            SendResult sendResult = rocketMQTemplate.syncSend(destination,
                            MessageBuilder.withPayload(message)
                                            .setHeader(MessageConst.PROPERTY_KEYS, "yo
                                            .build());
            System.out.printf("syncSend1 to topic %s sendResult=%s %n", topic, sendRes
    }
}
```

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 = "${rocketmq.namespace}%${rocketmq.consumer1.group}",  //
           // 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：namespace全称
           topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
           selectorExpression = "${rocketmq.consumer1.subExpression}" // 订阅表达式,
   )
   public class MessageConsumer implements RocketMQListener<String> {

       @Override
       public void onMessage(String message) {
```

```
            System.out.println("Tag1Consumer receive message：" + message);
        }
    }
```

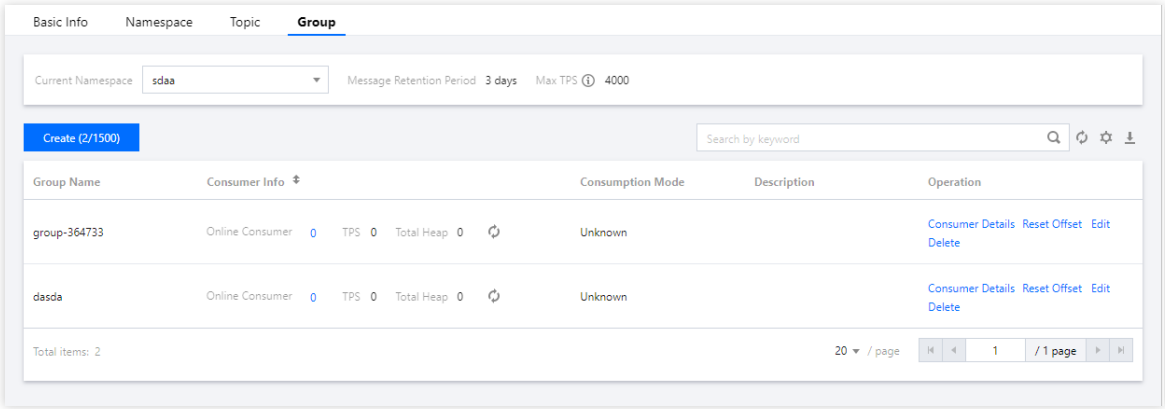
可根据业务需求配置多个消费者。消费者其他配置可根据具体业务需求进行配置。

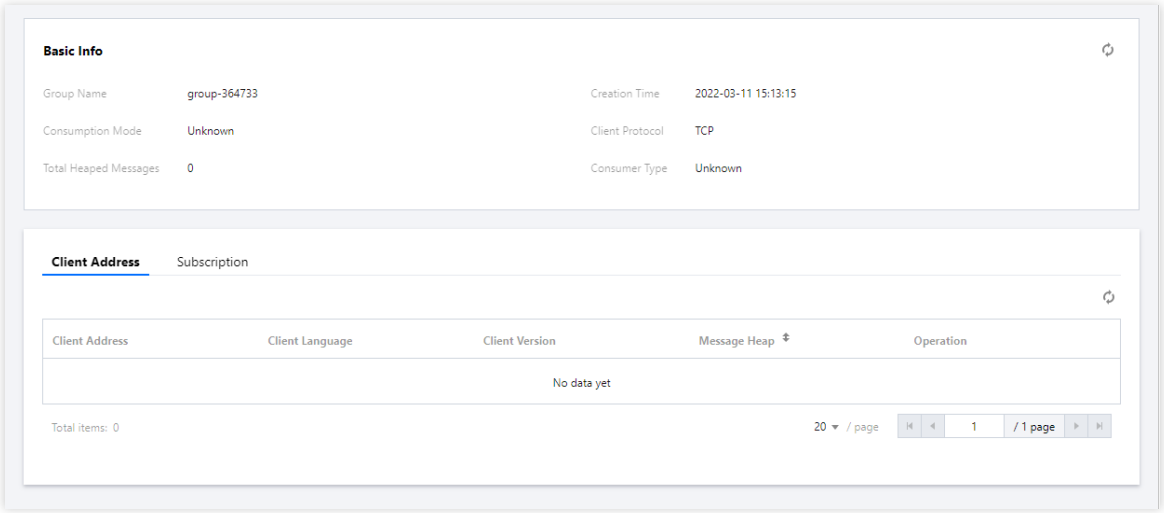
说明

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

登录 TDMQ 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。





发送与接收过滤消息

最近更新时间：2023-04-12 11:40:09

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过滤收发的完整过程。

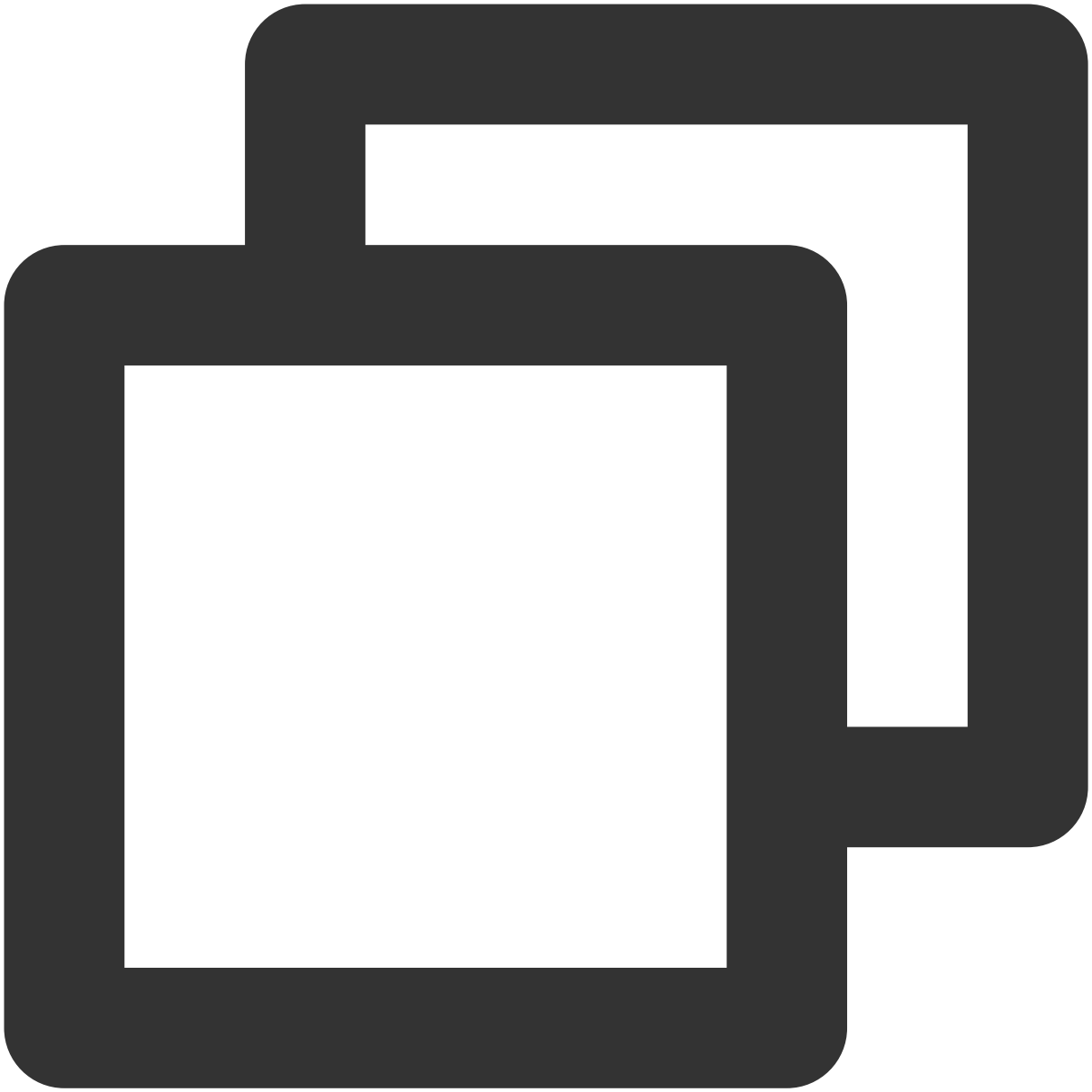
前提条件

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
已经了解发送与接收普通消息
下载 Demo或者前往GitHub 项目

操作步骤

发送消息

和普通发送消息相同，发送时，只需要将 rocketMQTemplate 的发送 Topic 拼接上 对应的 tag 即可。

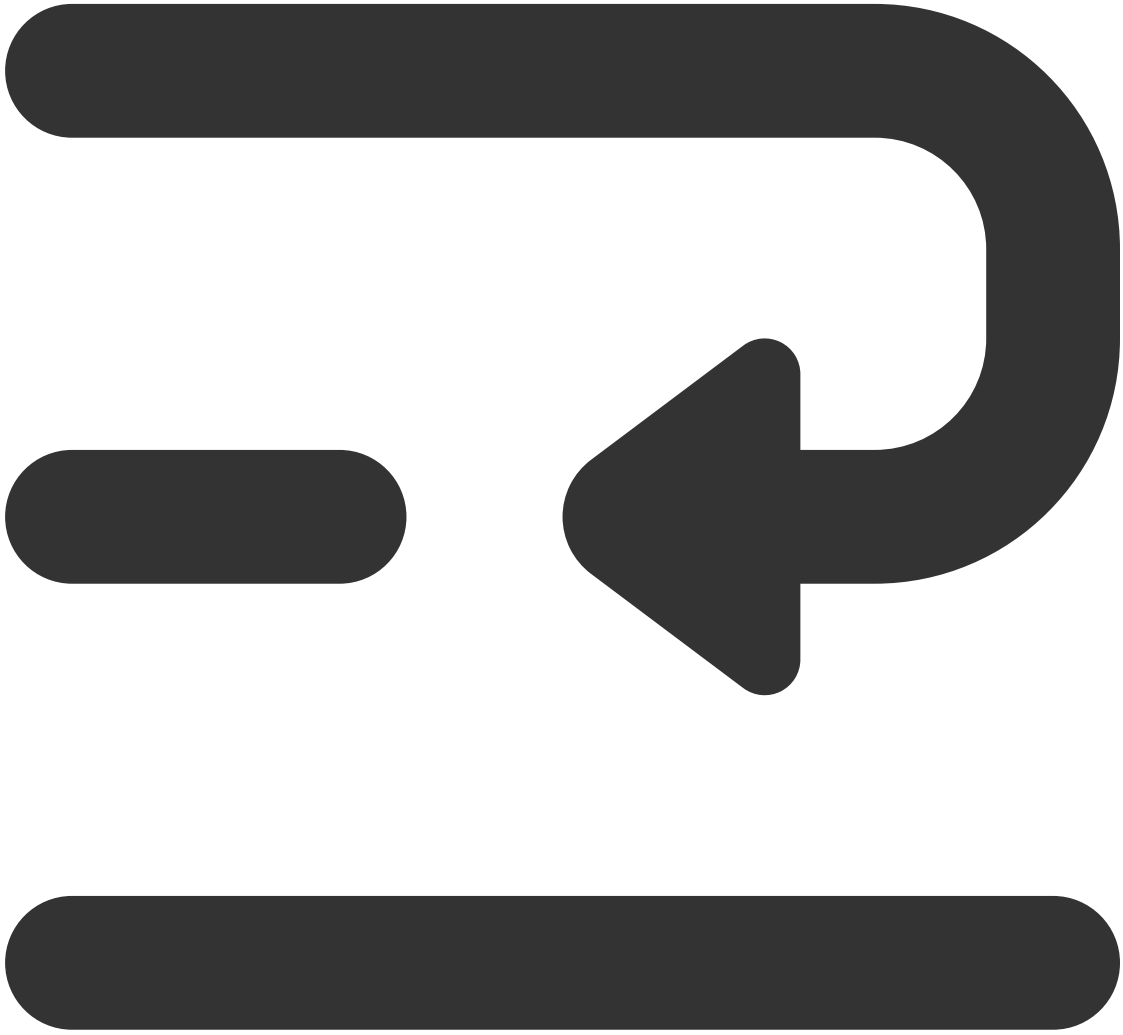


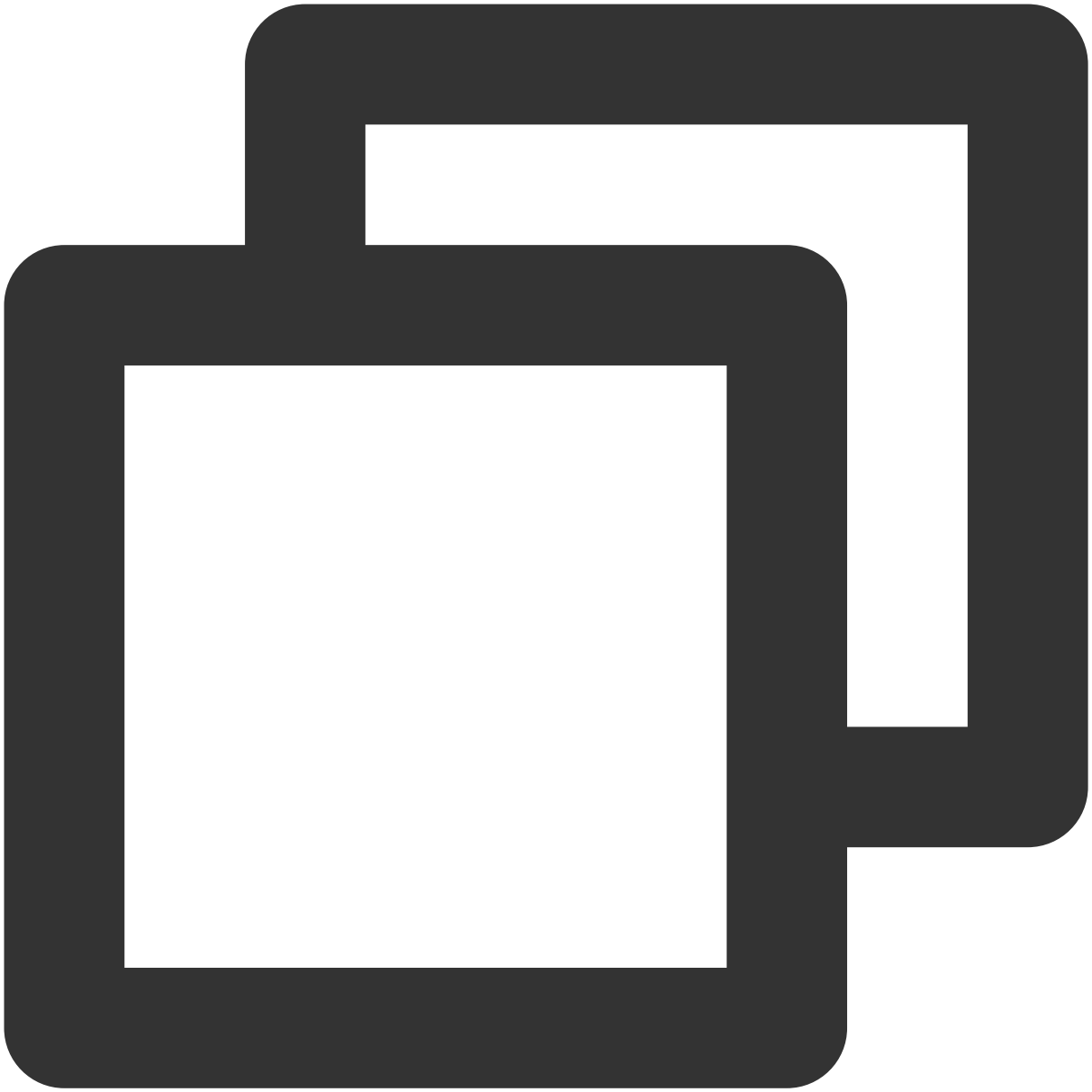
```
// springboot不支持使用header传递tags，根据要求，需要在topic后进行拼接 formats: topicName
        String destination = StringUtils.isBlank(tags) ? topic : topic + ":" + tags
        // object消息类型
        SendResult sendResult = rocketMQTemplate.syncSend(destination,
                MessageBuilder.withPayload(message)
                        .setHeader(MessageConst.PROPERTY_KEYS, "yourKey")    // 指定业
                        .build());
        System.out.printf("syncSend1 to topic %s sendResult=%s %n", topic, sendResu
```

例如 topic 是 TopicTest，tag是 TAG1, 那么调用 rocketMQTemplate 方法的第一个参数将是 TopicTest:TAG1

消费消息

设置 selectorExpression 字段为对应的过滤tag即可。如下代码中，将 rocketmq.consumer1.subExpression 设置为 TAG1 就可以消费 TAG1 的消息





```
@Service
@RocketMQMessageListener(
        consumerGroup = "${rocketmq.namespace}%${rocketmq.consumer1.group}",  // 消
        // 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：namespace全称%to
        topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
        selectorExpression = "${rocketmq.consumer1.subExpression}" // 订阅表达式，不配
)
public class Tag1Consumer implements RocketMQListener<String> {

    @Override
    public void onMessage(String message) {
```

```
            System.out.println("Tag1Consumer receive message：" + message);
        }
    }
```

发送与接收延迟消息

最近更新时间：2023-04-12 11:41:29

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，帮助您更好地理解延迟消息收发的完整过程。

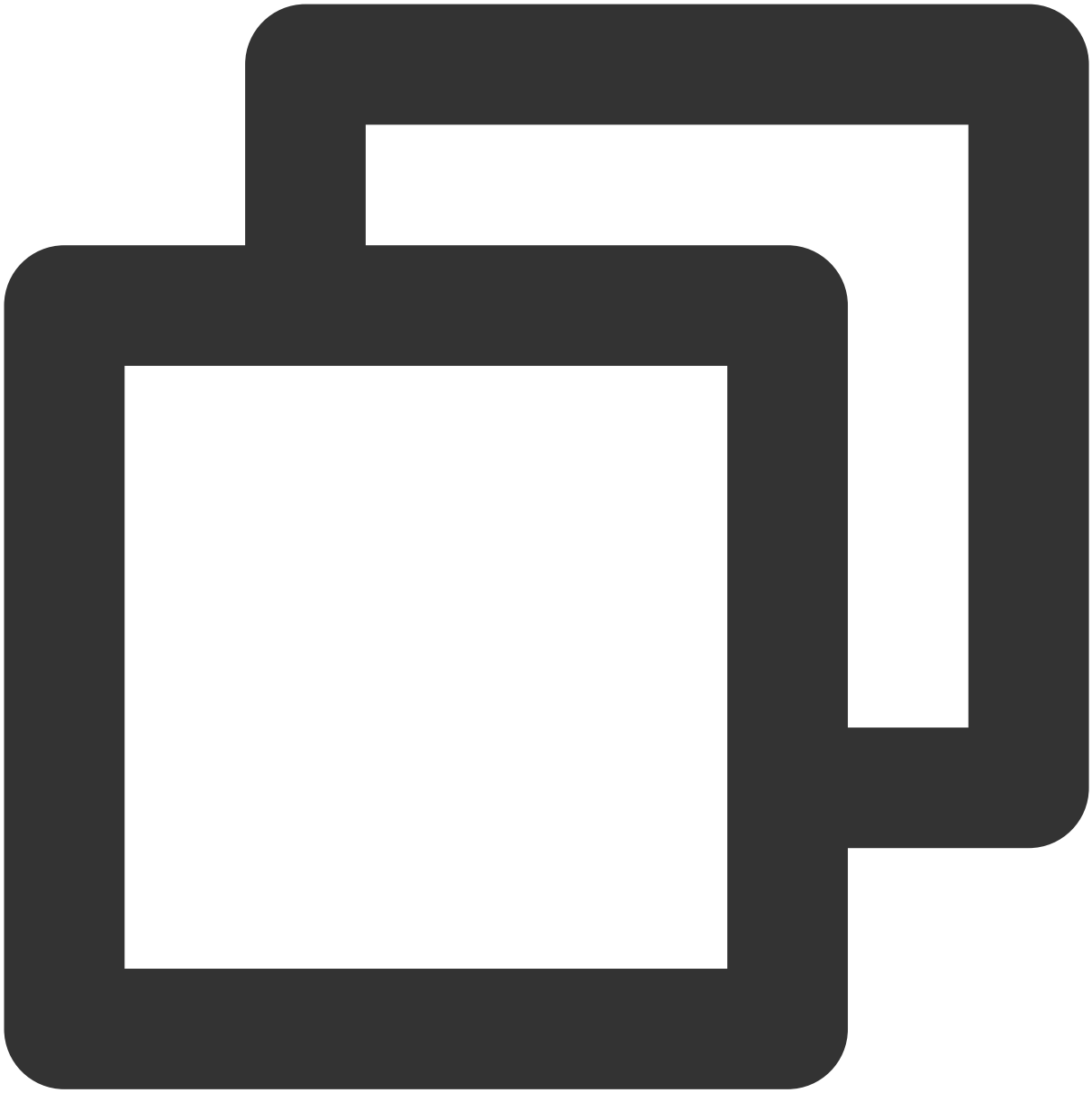
前提条件

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
已经了解发送与接收普通消息
下载 Demo或者前往GitHub 项目

操作步骤

发送消息

和普通发送消息相同，在调用发送方法的时候，传递对应的延迟级别即可。



```
SendResult sendResult = rocketMQTemplate.syncSend(
                destination,
                MessageBuilder.withPayload(message).build(),
                5000,
                delayLevel);
```

延迟等级与时间对应关系

其他延迟级别和具体延迟时间的对应关系如下：

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；

消费消息

和普通消息相同，无需特殊处理。

Spring Cloud Stream 接入

最近更新时间：2023-09-12 17:53:17

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。

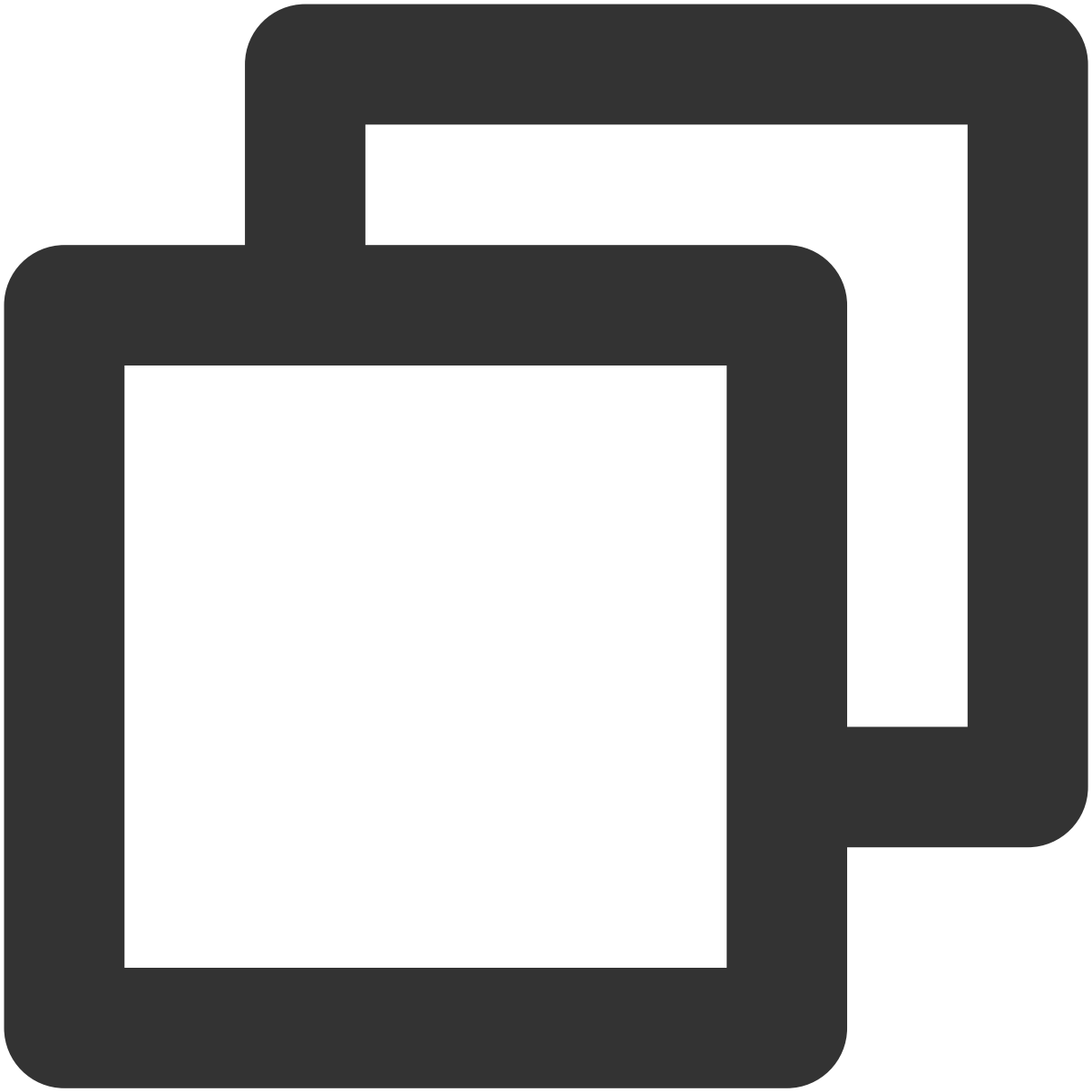
前提条件

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

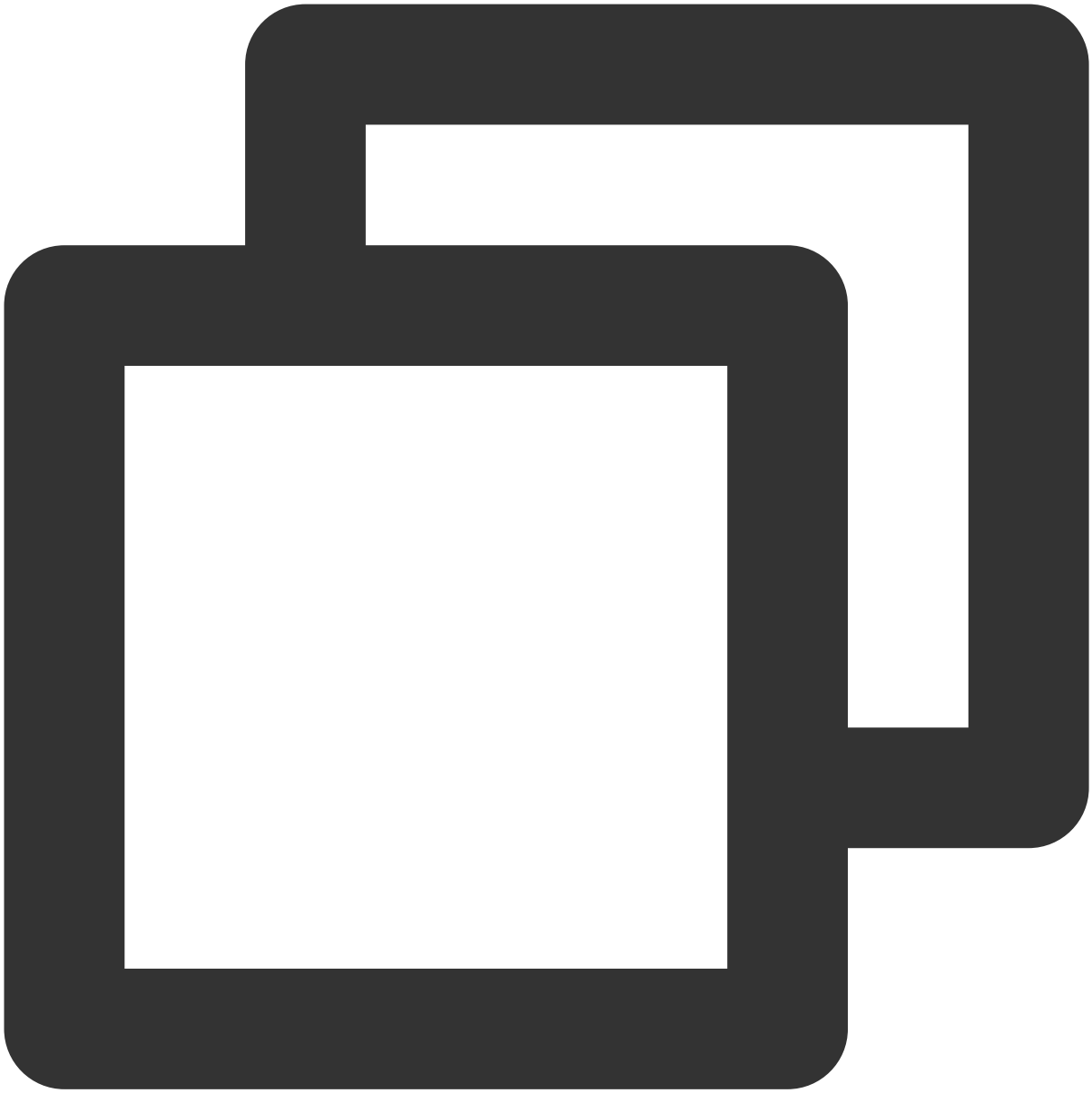
在 pom.xml 中引入 spring-cloud-starter-stream-rocketmq 相关依赖。当前建议版本 2021.0.4.0



```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
    <version>2021.0.4.0</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: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
          # 角色名称
          secret-key: admin
          # 角色密钥
          access-key: eyJrZXlJZ...
```

```
        # namespace全称
        namespace: rocketmq-xxx|namespace1
        # 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

1. 目前只有 `2.2.5-RocketMQ-RC1` 与 `2.2.5.RocketMQ.RC2` 及以上版本支持 **namespace** 配置，如使用别的版本需要对 topic 和 group 名称进行拼接。

格式如下：

rocketmq-pngrpmk94d5o|stream%topic (格式为：namespace全称%topic名称)

rocketmq-pngrpmk94d5o|stream%group (格式为：namespace全称%group名称)

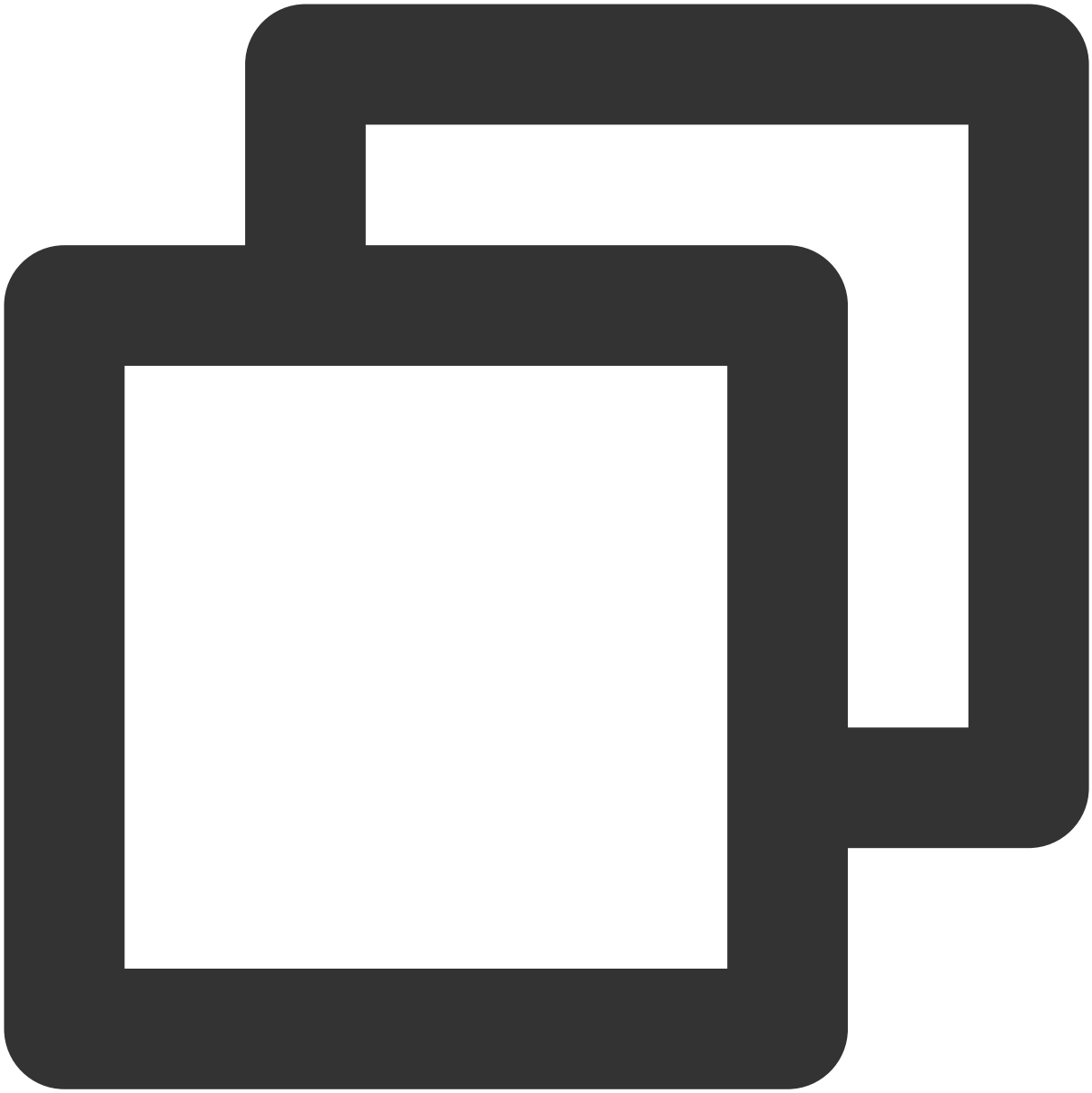
新的共享版与专享版格式如下：

MQ_INST_rocketmqpj79obd2ew7v_test%topic (格式为：namespace全称%topic名称)

MQ_INST_rocketmqpj79obd2ew7v_test%group (格式为：namespace全称%group名称)

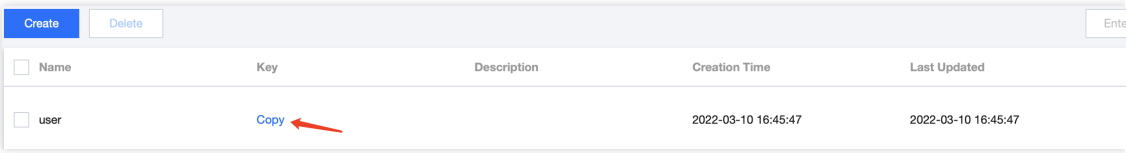
2. 配置方面 `2.2.5-RocketMQ-RC1` 与 `2.2.5.RocketMQ.RC2` 的订阅配置项为 `subscription` ，其他低版本订阅配置项为 `tags` 。

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
```

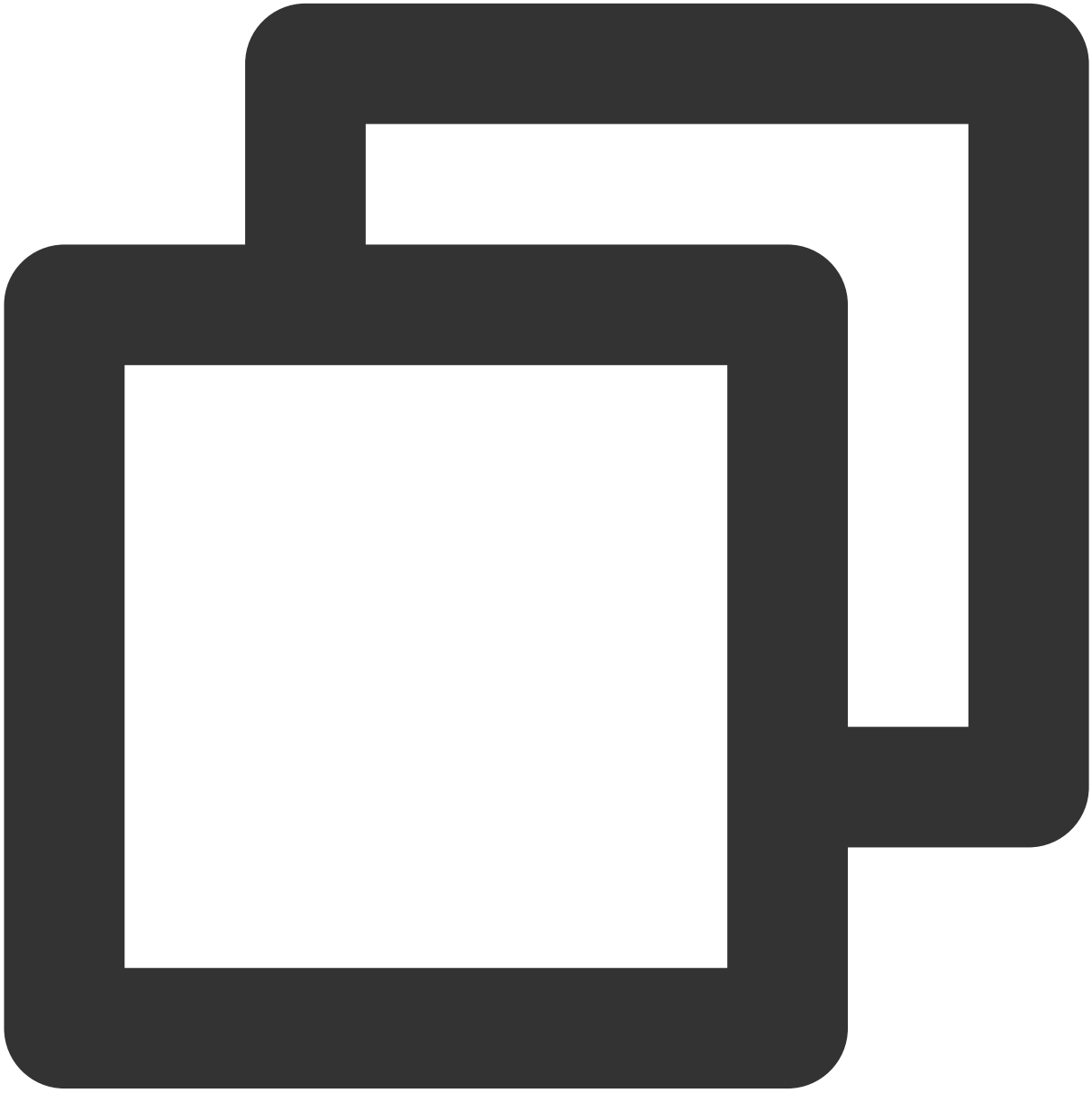
```
        Topic-test2:
          consumer:
            tags: TAG2
      binder:
        # 服务地址全称
        name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
        # 角色名称
        secret-key: admin
        # 角色密钥
        access-key: eyJrZXlJZ...
    bindings:
      # channel名称
      Topic-send:
        # 指定topic，对应创建的topic全称，格式为：集群id|namespace名称%topic名称
        destination: rocketmq-xxx|stream%topic1
        content-type: application/json
        # 要使用group全称称，格式为：集群id|namespace名称%group名称
        group: rocketmq-xxx|stream%group1
      # channel名称
      Topic-test1:
        destination: rocketmq-xxx|stream%topic1
        content-type: application/json
        group: rocketmq-xxx|stream%group1
      # channel名称
      Topic-test2:
        destination: rocketmq-xxx|stream%topic1
        content-type: application/json
        group: rocketmq-xxx|stream%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在命名空间列表获取。
secret-key	角色名称，在 角色管理 页面复制。
access-key	角色密钥，在 角色管理 页面复制密钥列复制。 
namespace	命名空间的名称，在控制台命名空间页面复制。

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。

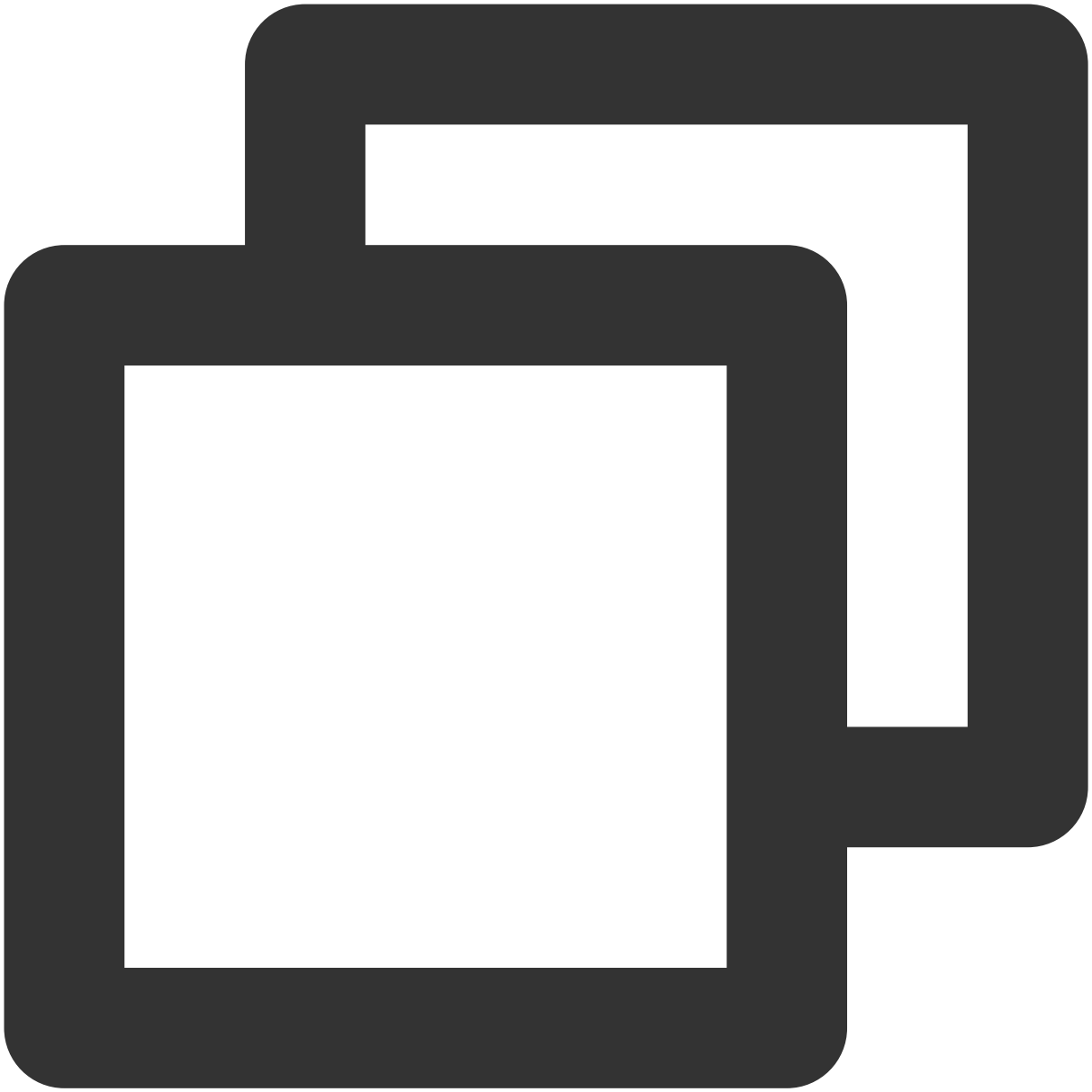


```
/**
 * 自定义通道 Binder
 */
```

```
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

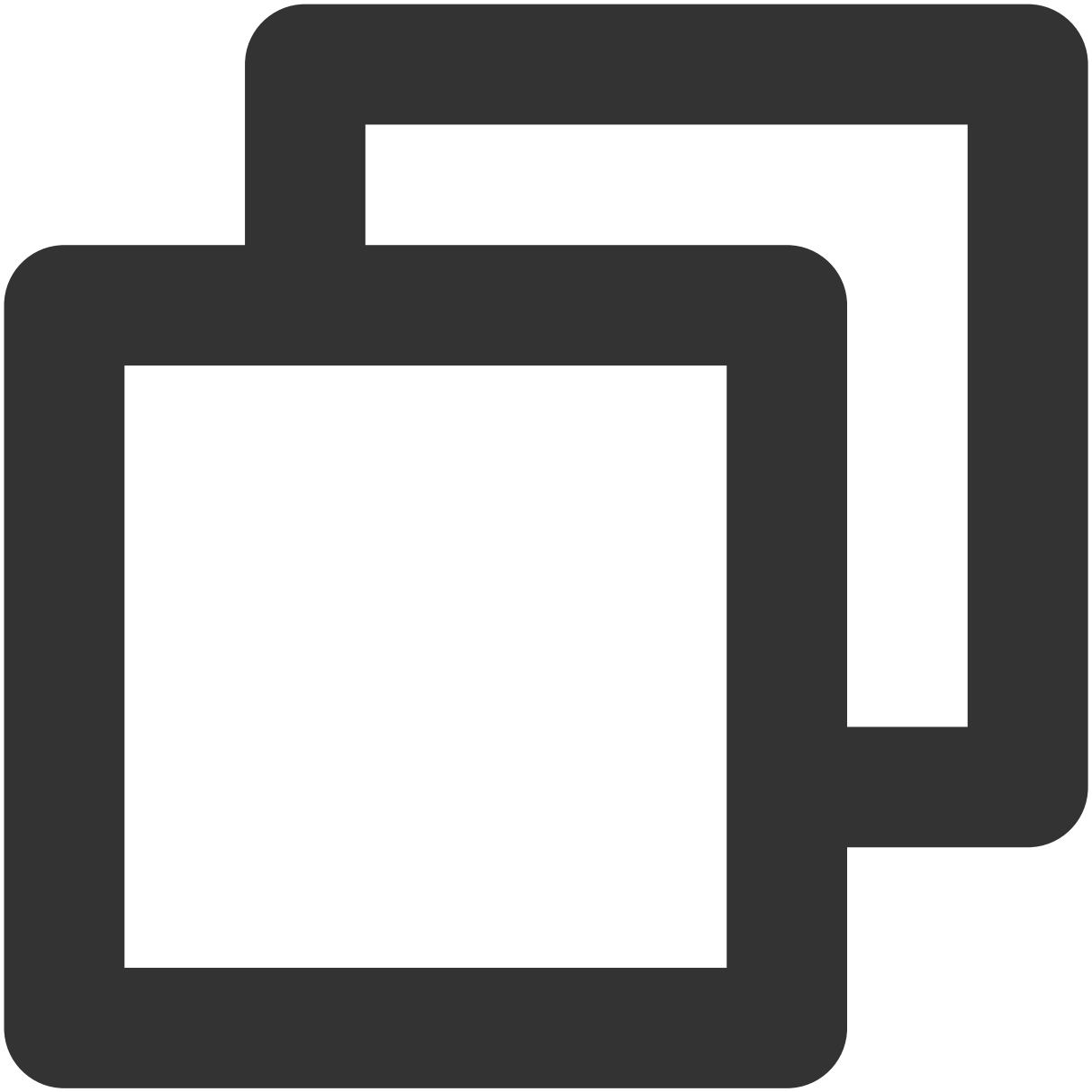
在配置类或启动类上添加相应注解，如果有多个 binder 配置，都要在此注解中进行指定。



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

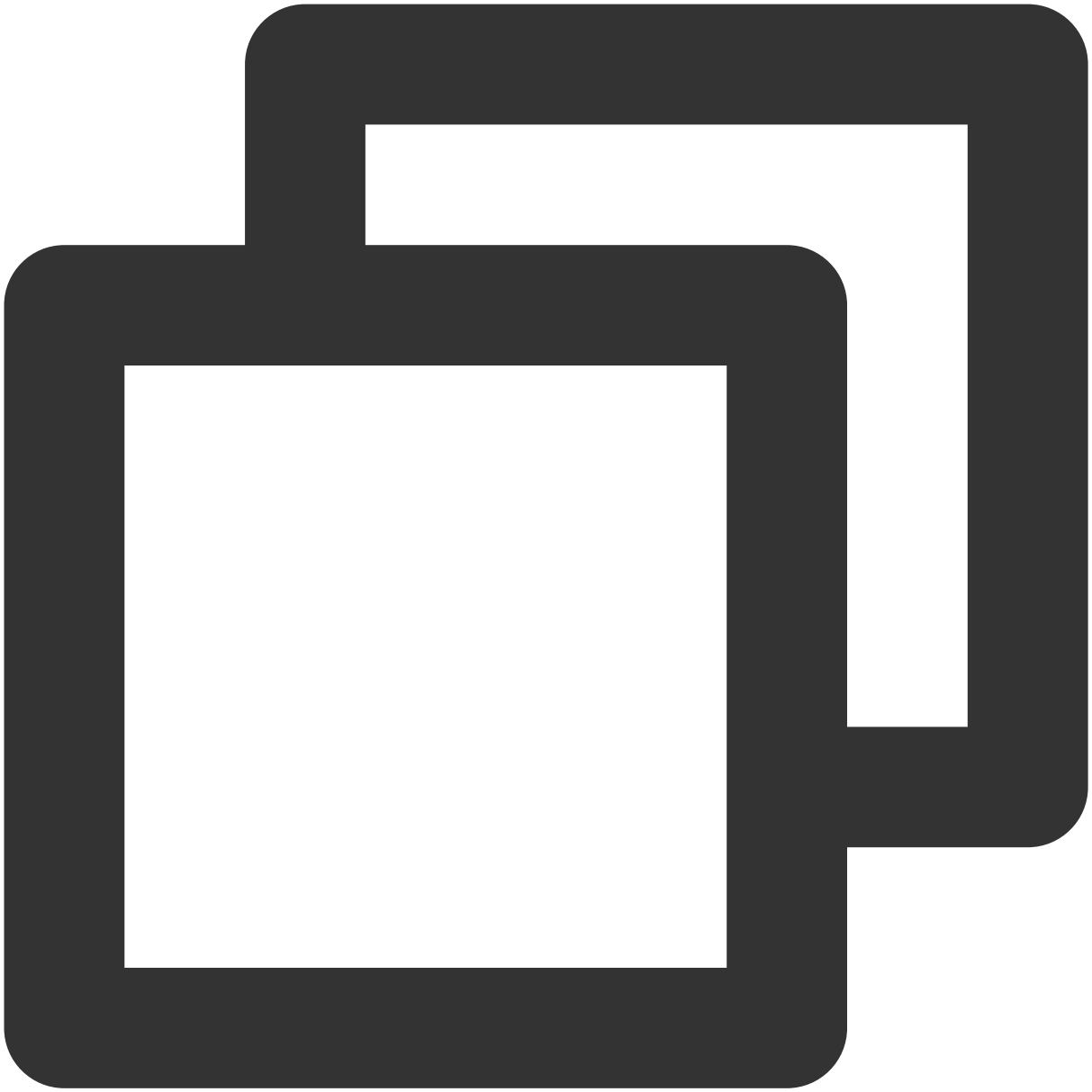
步骤5：发送消息

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



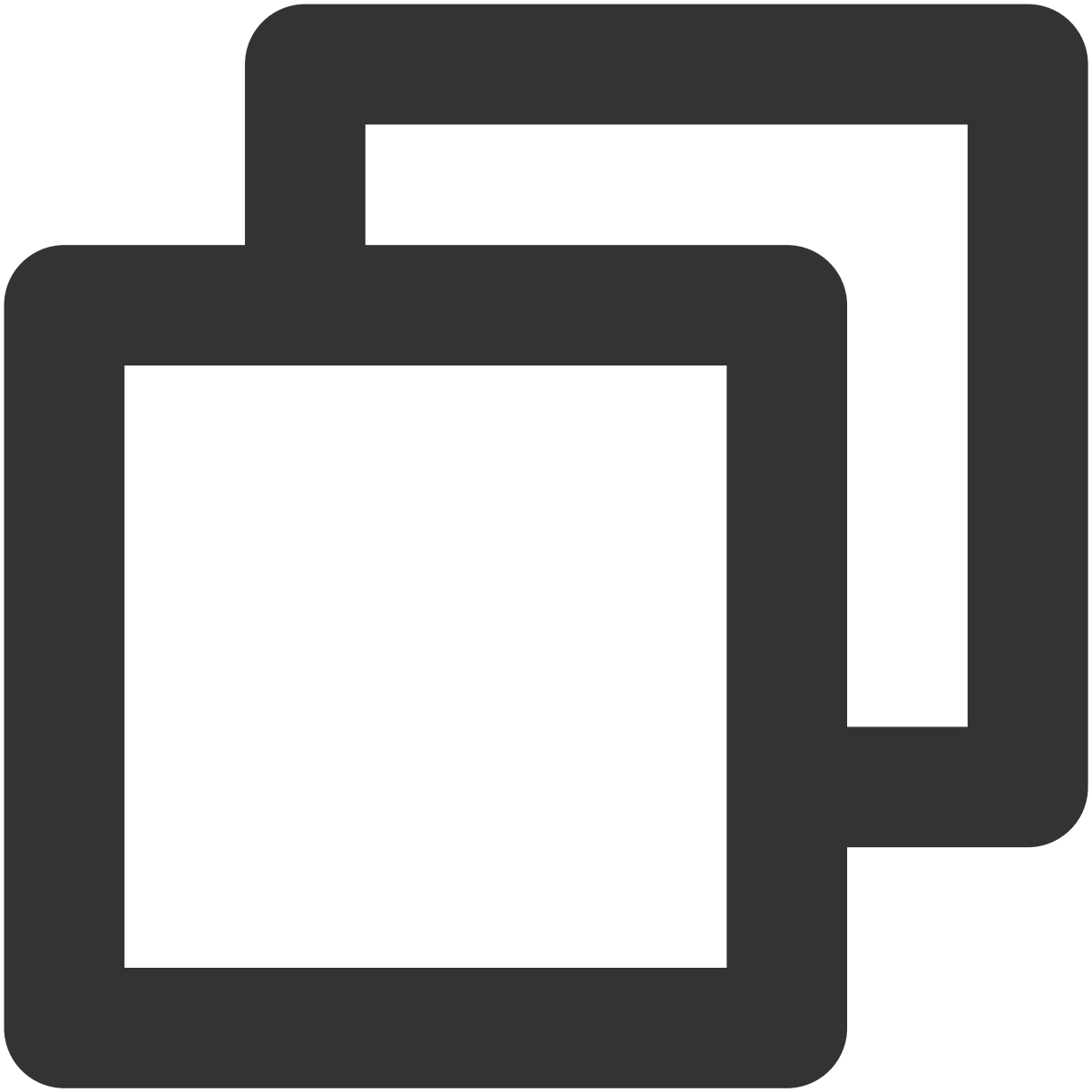
```
@Autowired
     private CustomChannelBinder channelBinder;
```

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.").buil
          channelBinder.sendChannel().send(message);
```

步骤6：消费消息



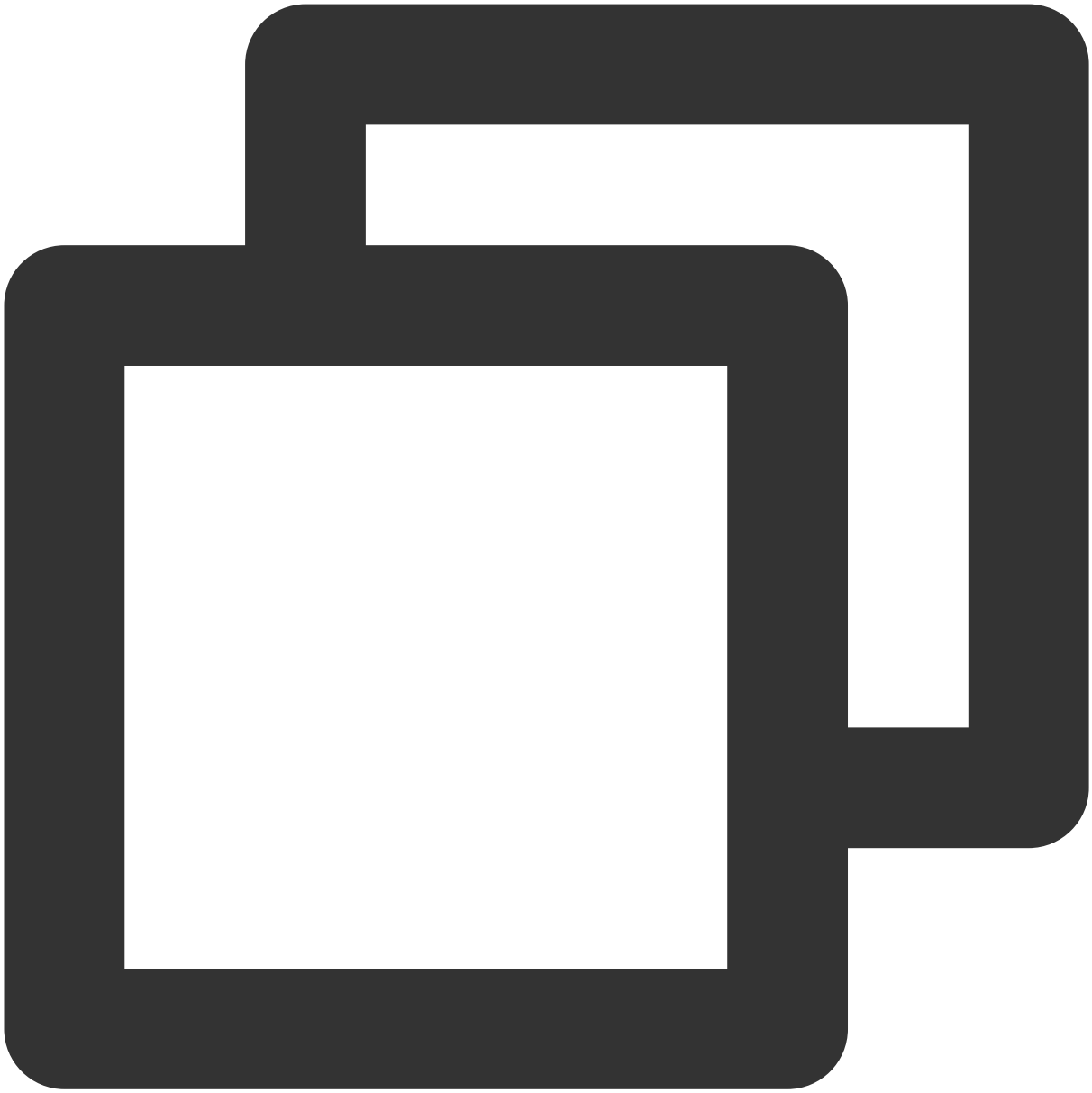
```
@Service
public class StreamConsumer {
    private final Logger logger = LoggerFactory.getLogger(StreamDemoApplication.cla

    /**
     * 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.Stream
2023-02-23 19:19:01.138  INFO 21958 --- [nsumer-group1_1] c.t.d.s.StreamDemoApplica
```

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。

Java SDK
发送与接收普通消息

最近更新时间：2023-11-24 14:22:39

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。

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

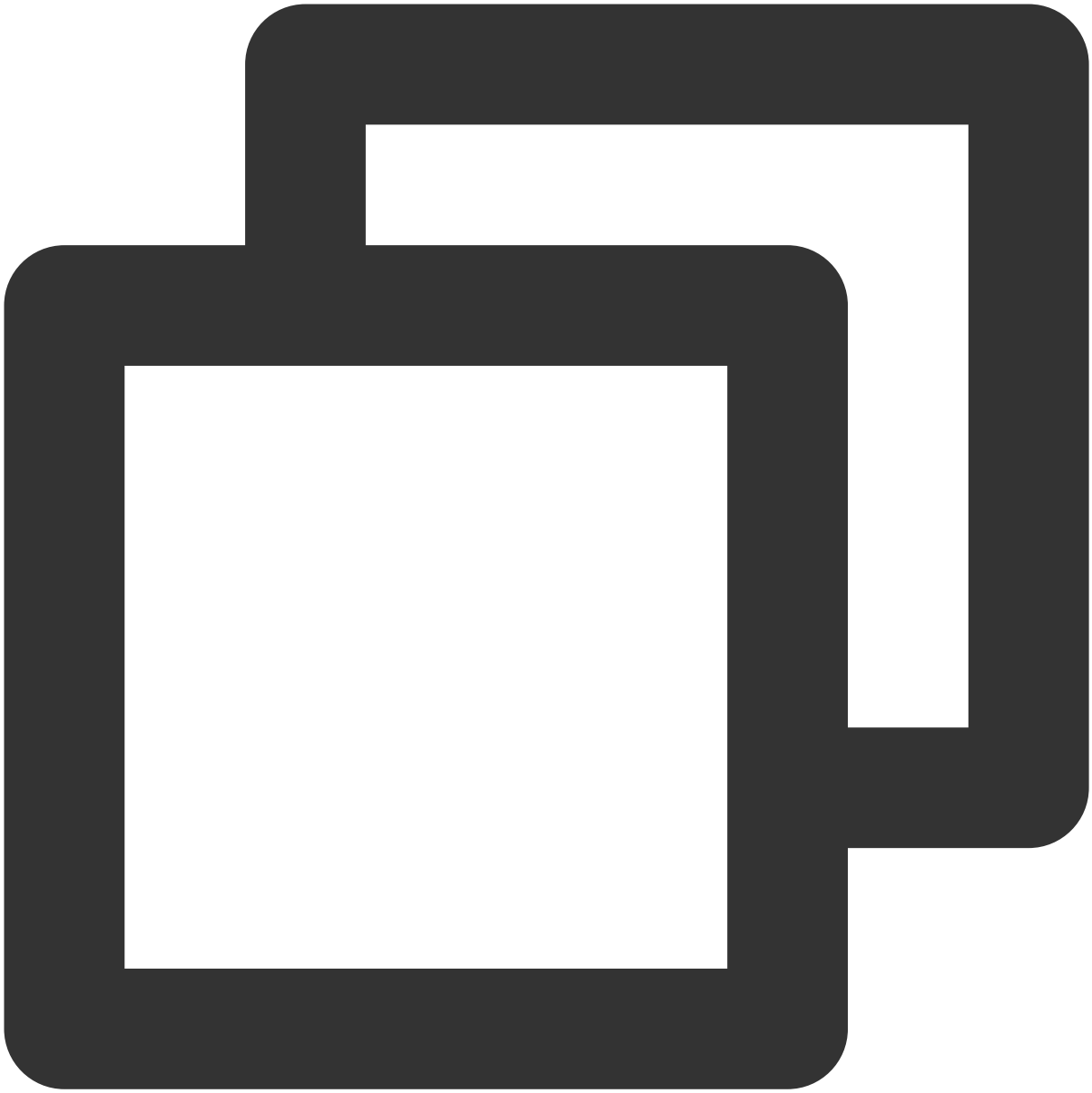
操作步骤

步骤1：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：

说明

依赖版本要求 ≥ 4.9.3, 当前建议为4.9.4

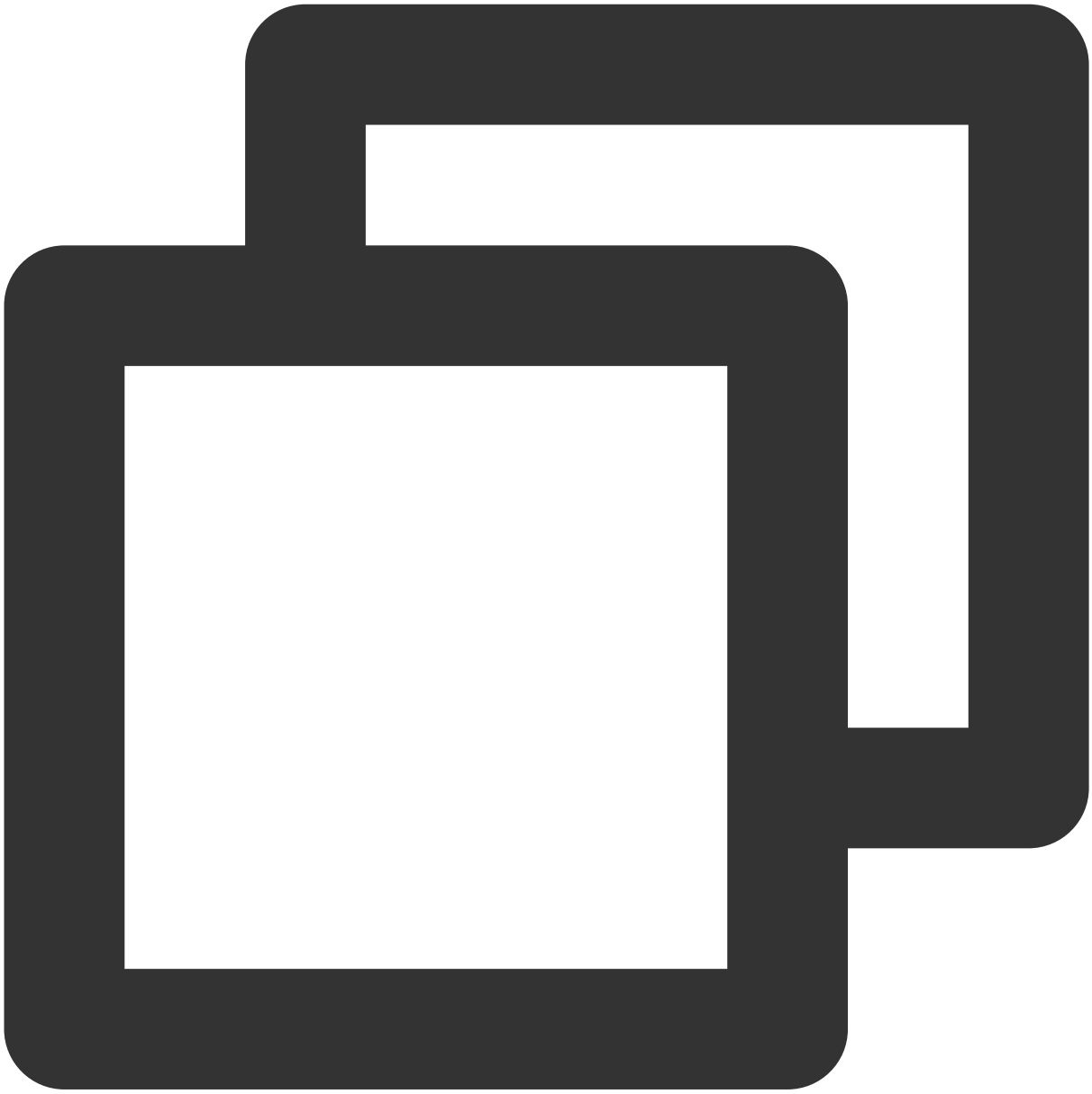


```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.4</version>
  </dependency>

  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-acl</artifactId>
      <version>4.9.4</version>
```

```
    </dependency>
```

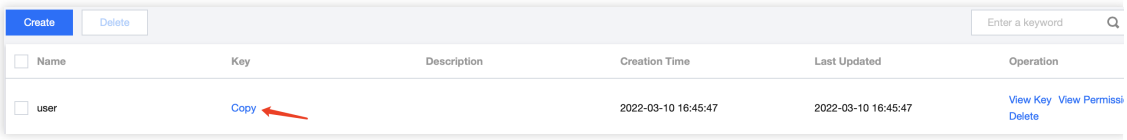
步骤2：生产消息

创建消息生产者



```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL权限
    );
```

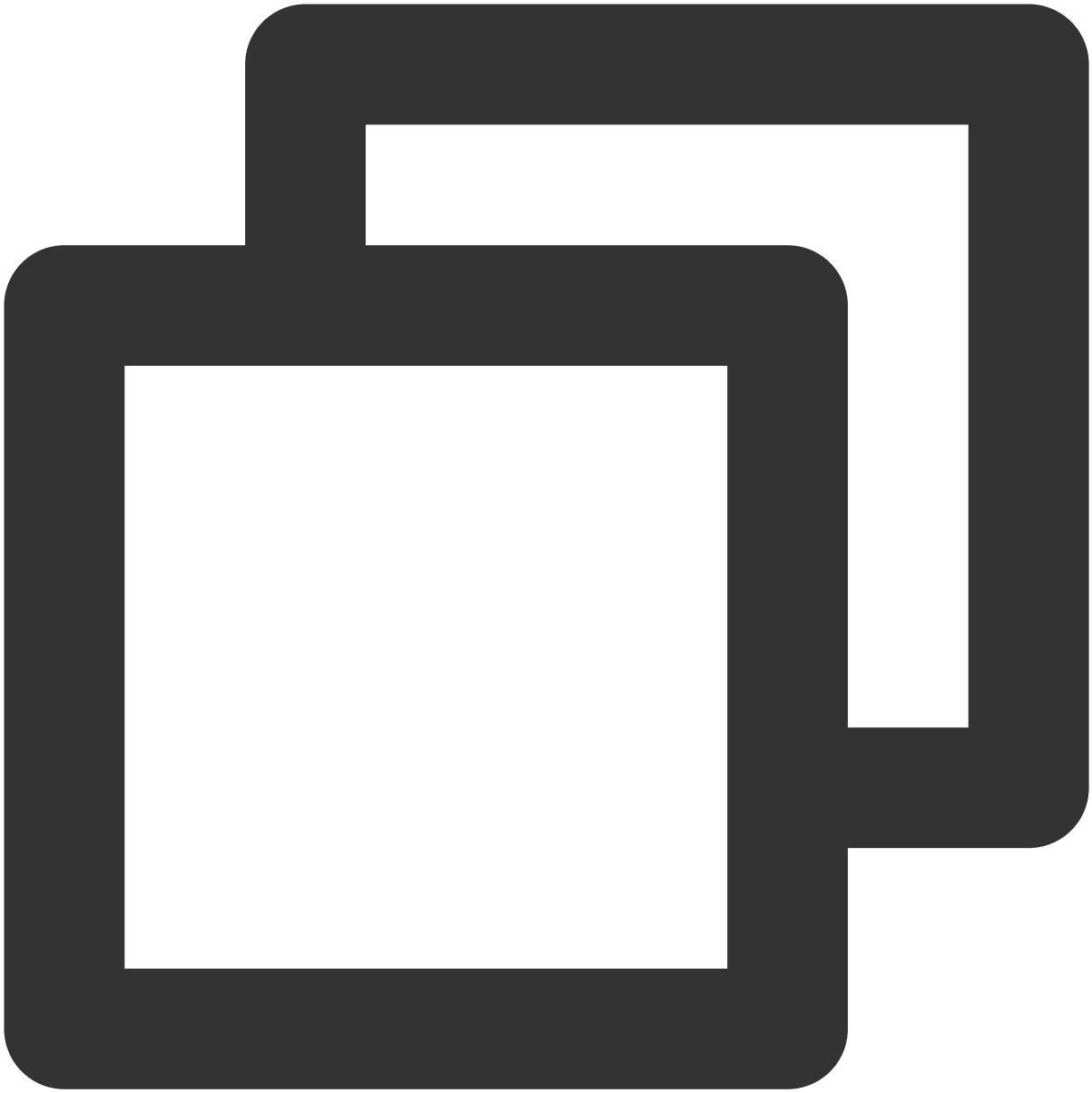
```
// 设置NameServer的地址
producer.setNamesrvAddr(nameserver);
// 启动Producer实例
producer.start();
```

参数	说明
groupName	生产者组名称，建议使用对应的 topic 名字
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 
secretKey	角色名称，在 角色管理 页面复制。
nameserver	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接入点地址在命名空间列表获取。

发送消息

发送消息有多种方式：同步发送、异步发送、单向发送等。

同步发送

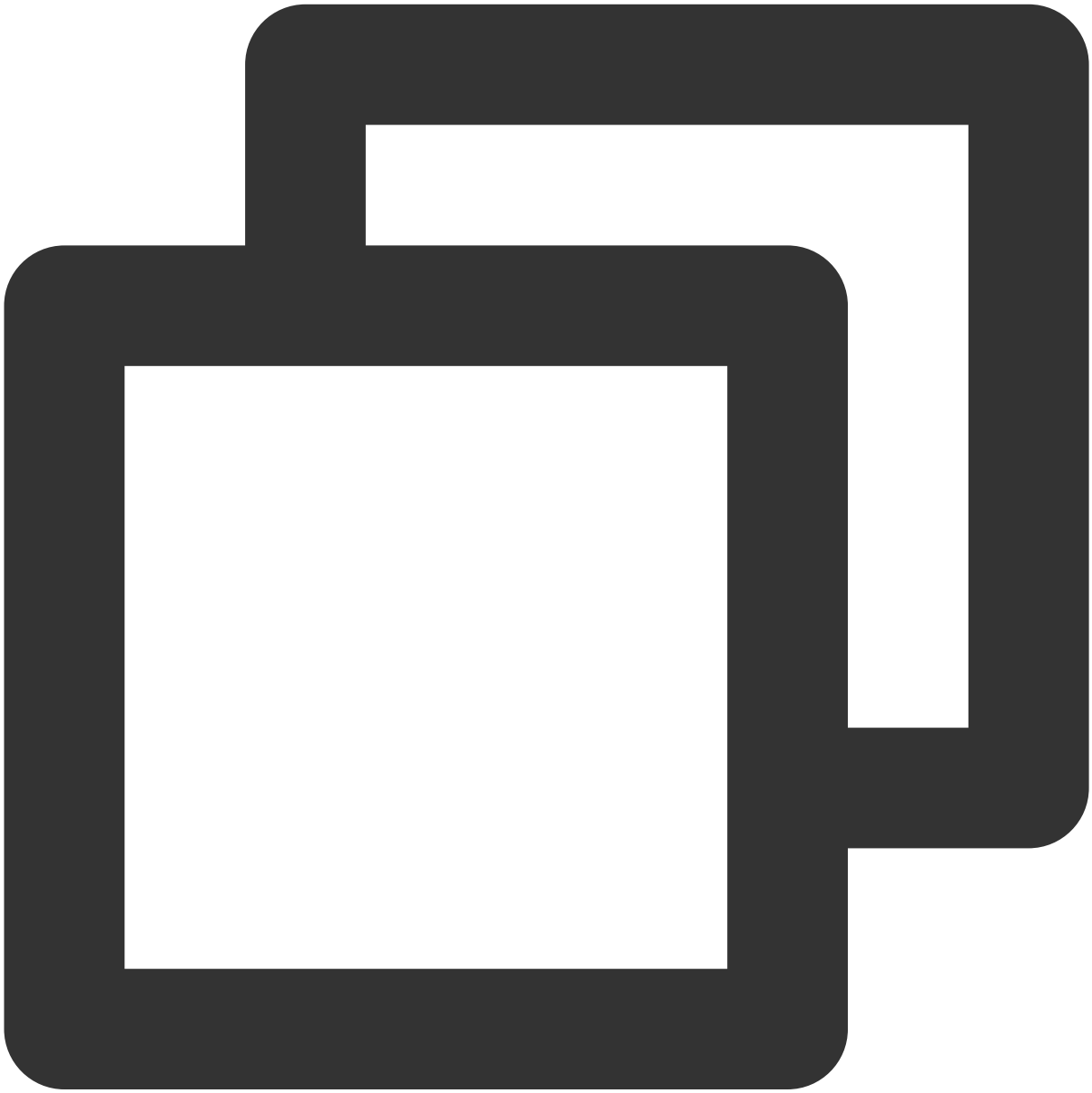


```
for (int i = 0; i < 10; i++) {
            // 创建消息实例，设置topic和消息内容
            Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getB
            // 发送消息
            SendResult sendResult = producer.send(msg);
            System.out.printf("%s%n", sendResult);
        }
```

参数	说明

topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
TAG	用来设置消息的 TAG。

异步发送

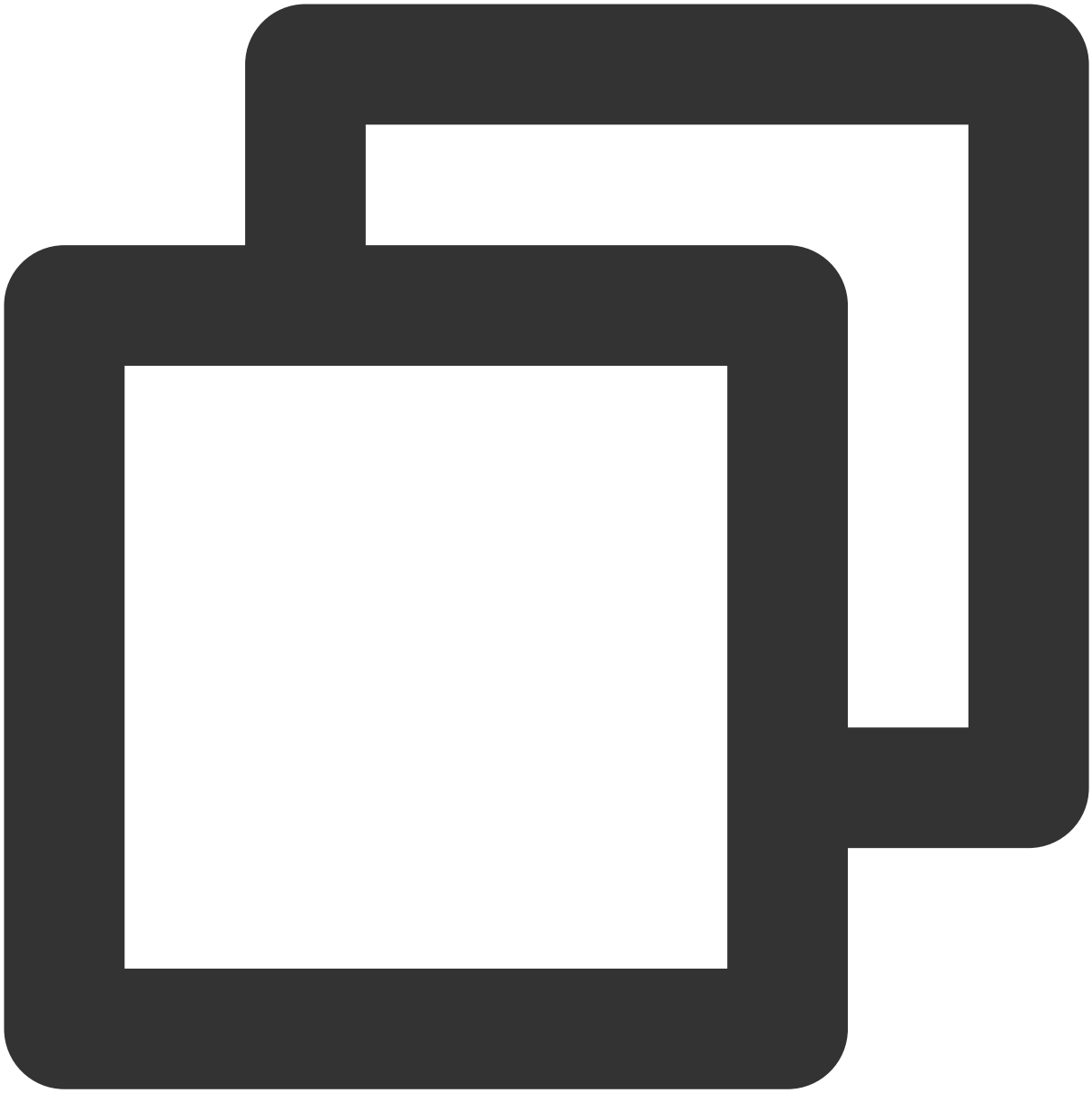


```
// 设置发送失败后不重试
        producer.setRetryTimesWhenSendAsyncFailed(0);
        // 设置发送消息的数量
        int messageCount = 10;
        final CountDownLatch countDownLatch = new CountDownLatch(messageCount);
```

```
        for (int i = 0; i < messageCount; i++) {
            try {
                final int index = i;
                // 创建消息实体，设置topic和消息内容
                Message msg = new Message(topic_name, "TAG", ("Hello rocketMq " + ind
                producer.send(msg, new SendCallback() {
                    @Override
                    public void onSuccess(SendResult sendResult) {
                        // 消息发送成功逻辑
                        countDownLatch.countDown();
                        System.out.printf("%-10d OK %s %n", index, sendResult.getMsgI
                    }

                    @Override
                    public void onException(Throwable e) {
                        // 消息发送失败逻辑
                        countDownLatch.countDown();
                        System.out.printf("%-10d Exception %s %n", index, e);
                        e.printStackTrace();
                    }
                });
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        countDownLatch.await(5, TimeUnit.SECONDS);
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
TAG	用来设置消息的 TAG。

单向发送



```
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, "TAG", ("Hello RocketMQ " + i).getBy
          // 发送单向消息
          producer.sendOneway(msg);
      }
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。

TAG	用来设置消息的 TAG。

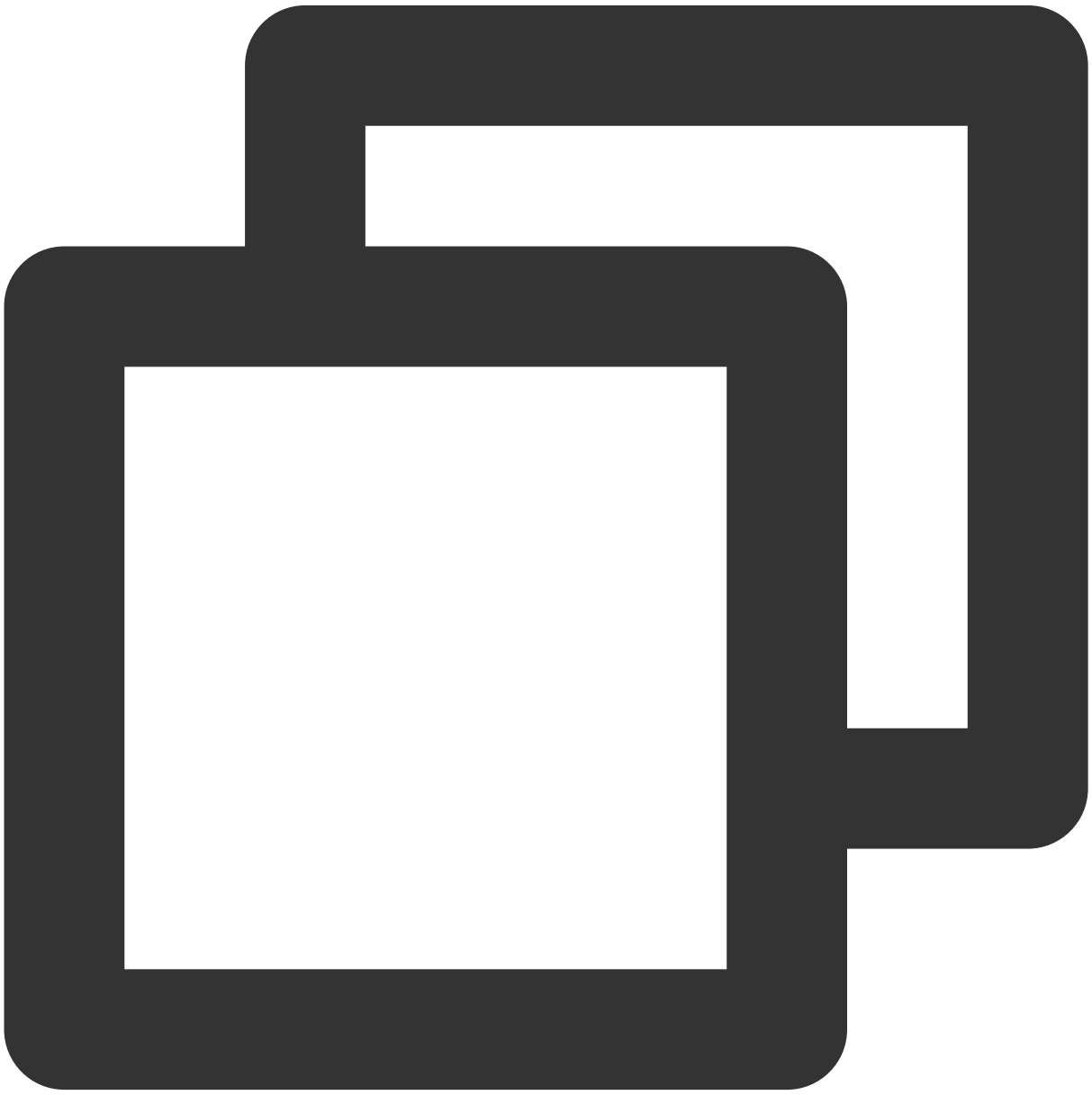
说明

批量发送及其他情况可参见 GitHub Demo 或 RocketMQ 官方文档 。

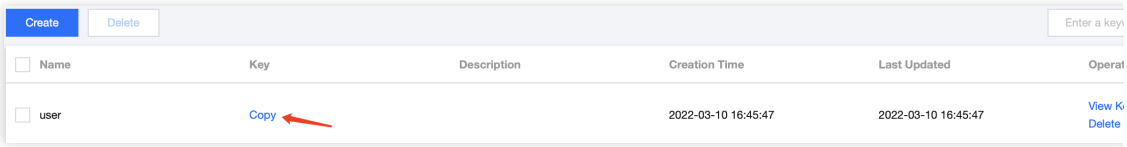
步骤3：消费消息

创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式

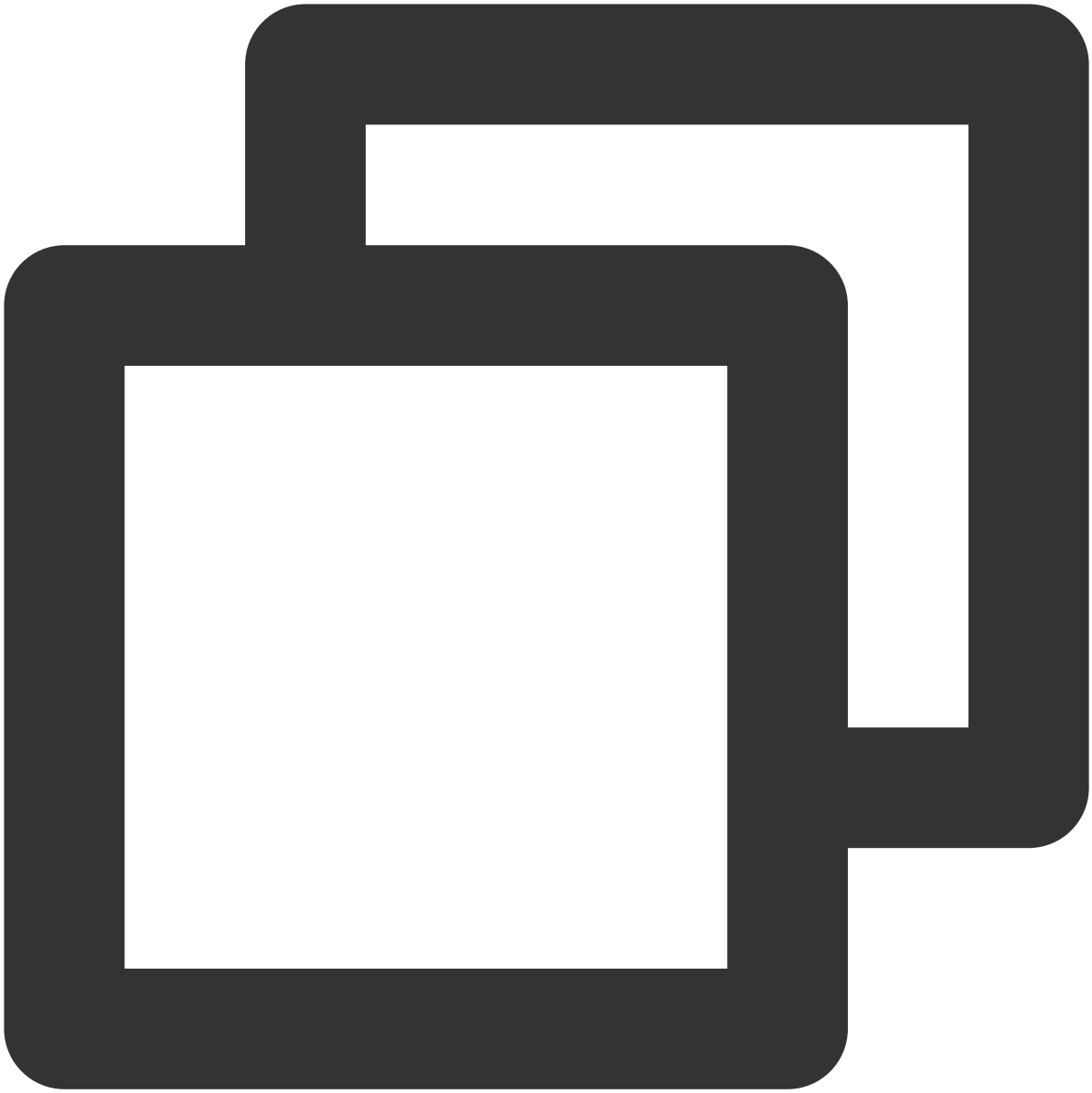


```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

参数	说明
groupName	生产者组名称，在控制台集群管理中Group 页签中复制。
nameserver	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地址在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

订阅消息

根据消费模式不同，订阅方式也有所区别。

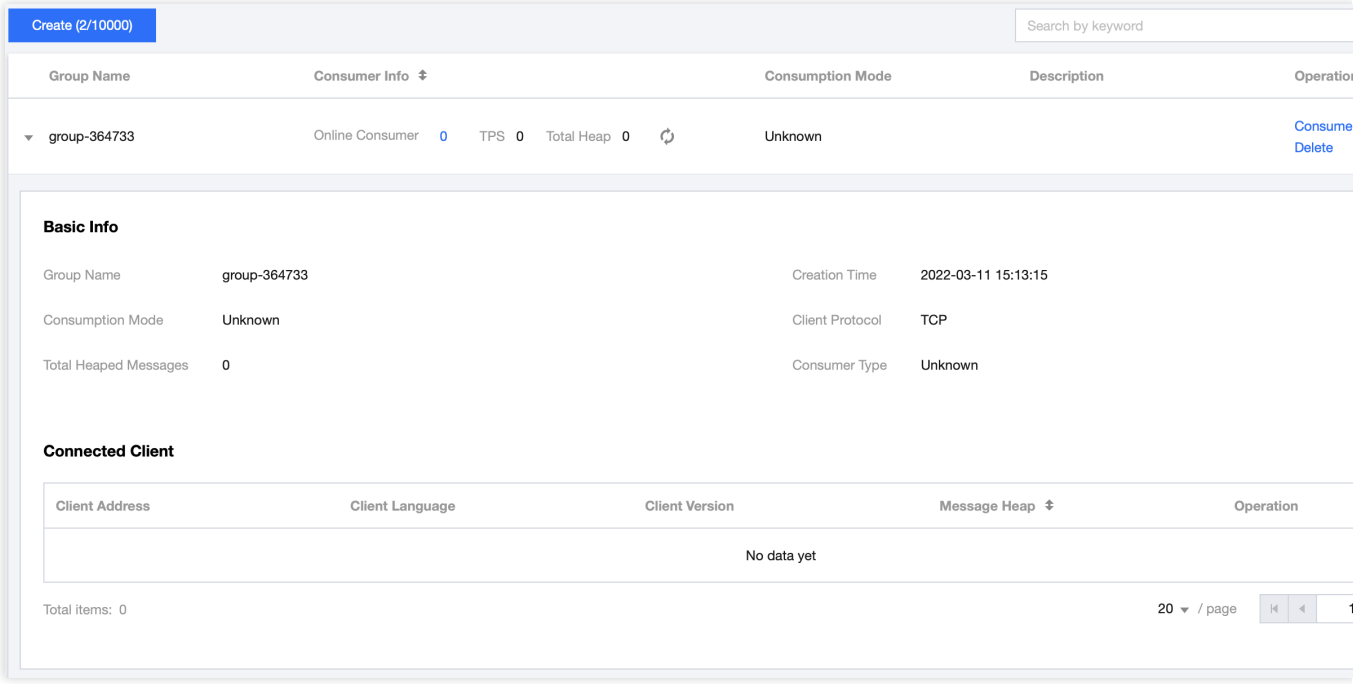


```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
            // 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();
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4：查看消费详情

登录 TDMQ 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**查看消费者详情**，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 或 RocketMQ 官方文档 。

发送与接收延迟消息

最近更新时间：2023-11-24 14:28:48

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现定时消息收发的操作过程。

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

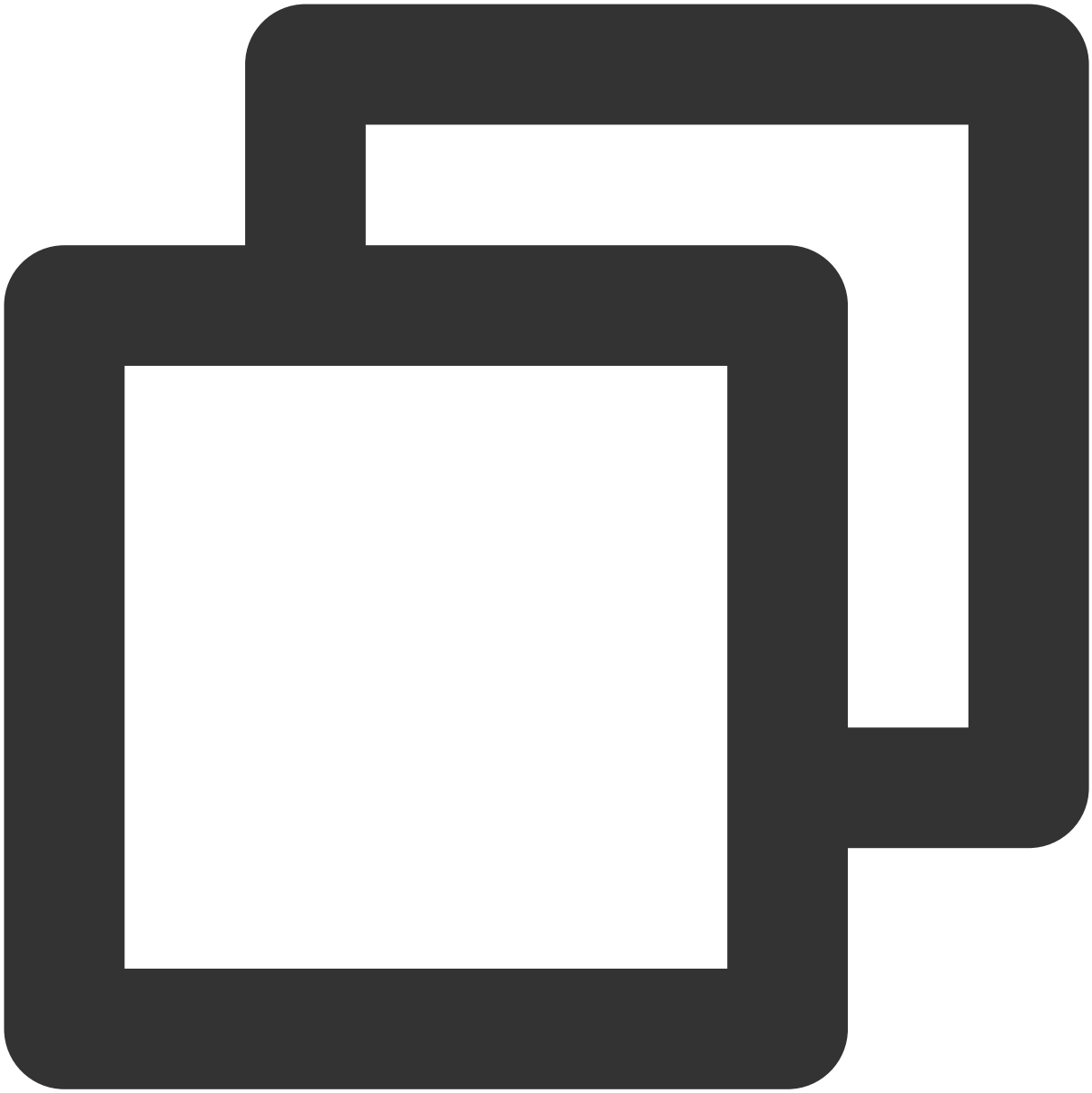
操作步骤

步骤1：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：

说明

依赖版本要求 ≥ 4.9.3, 当前建议为4.9.4

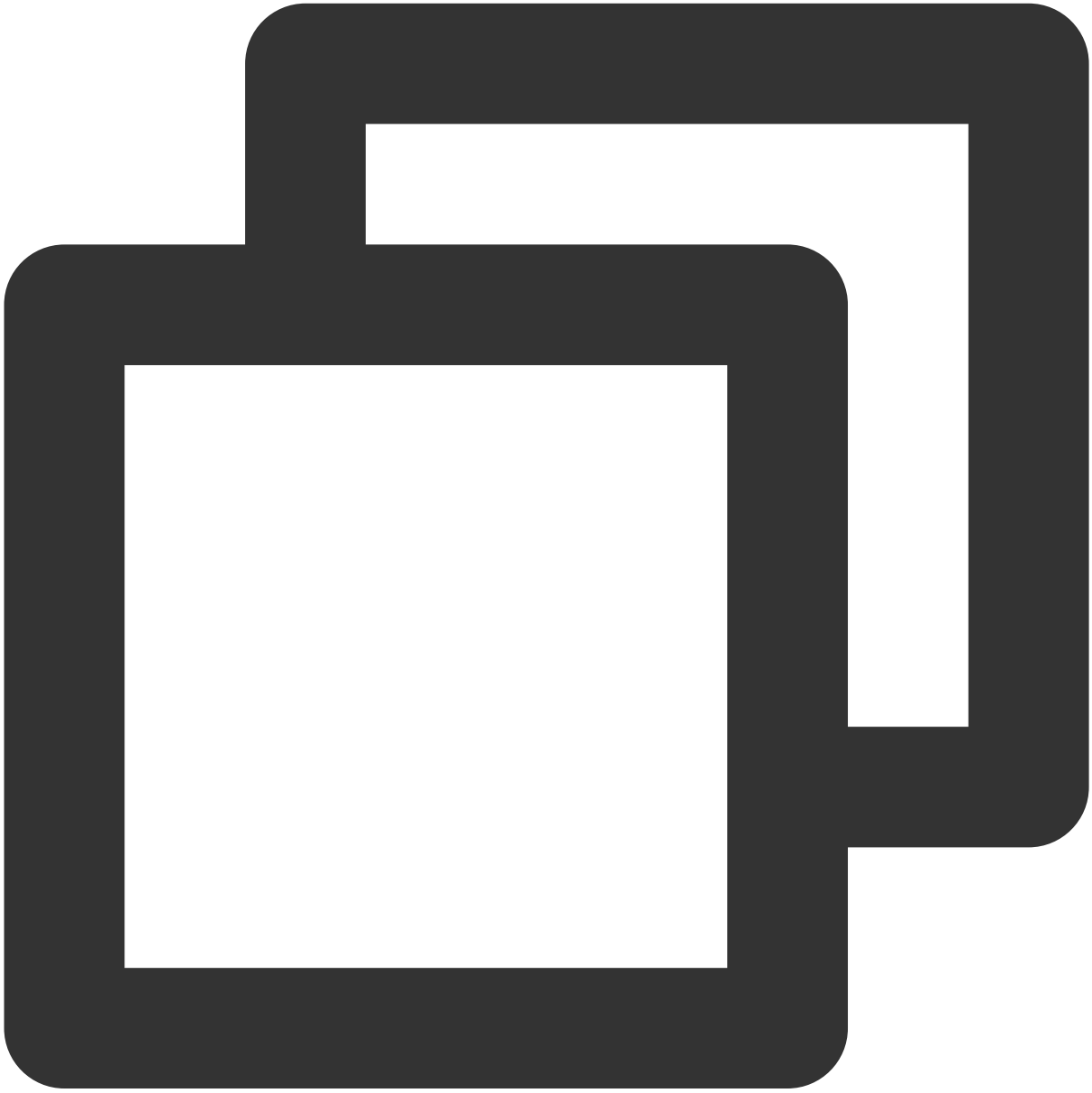


```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.4</version>
  </dependency>

  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-acl</artifactId>
      <version>4.9.4</version>
```

```
    </dependency>
```

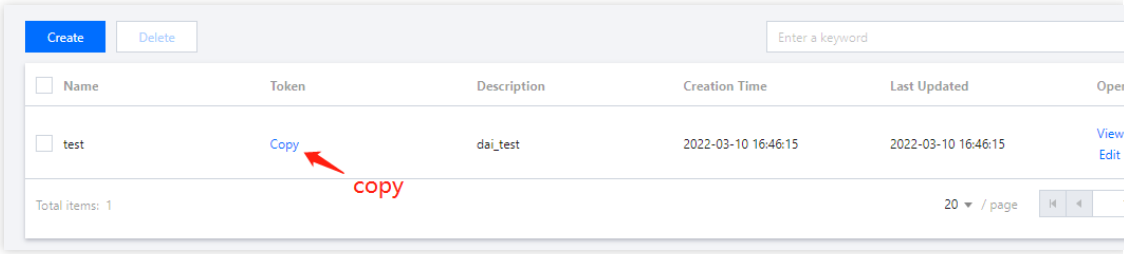
步骤2：生产消息

创建消息生产者



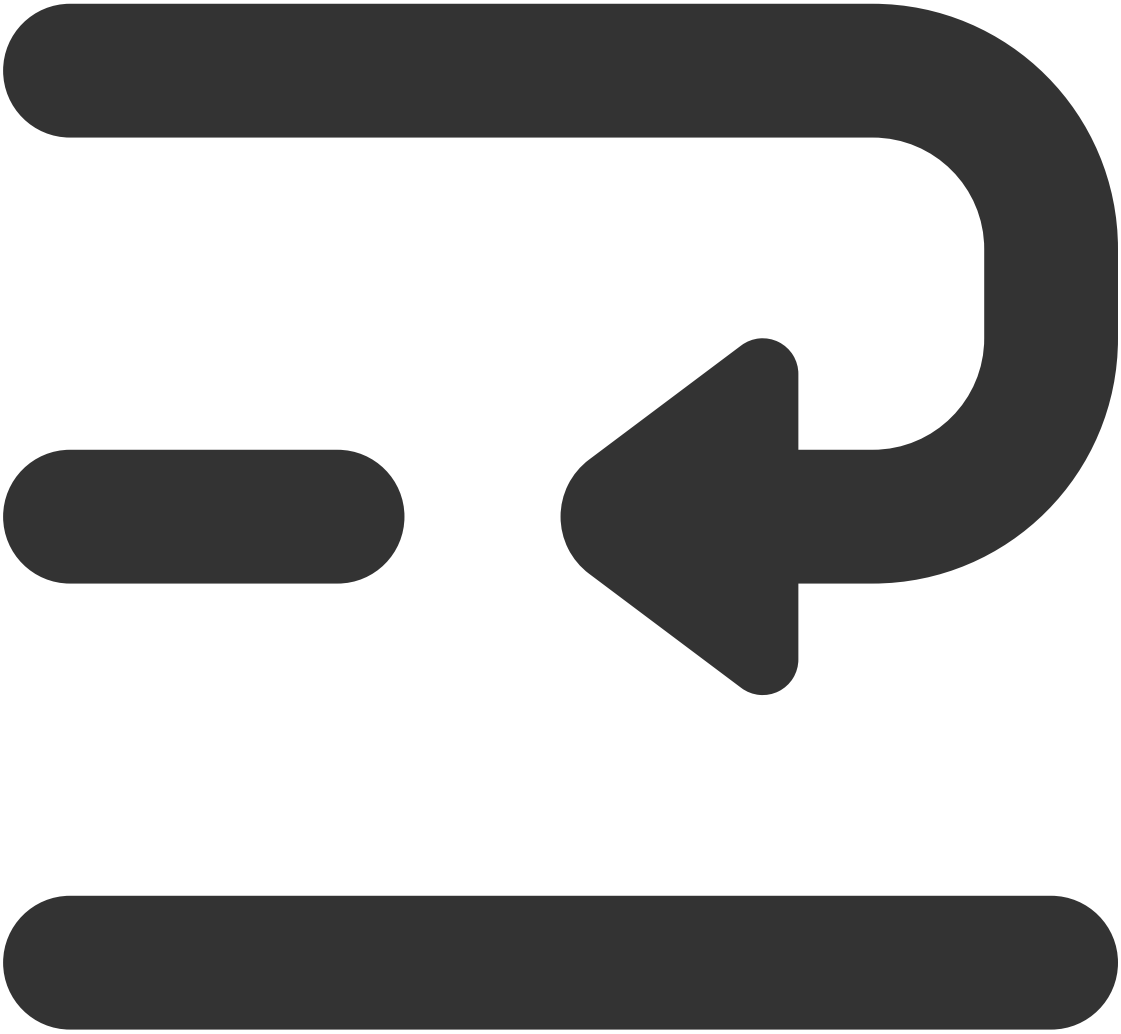
```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL权限
    );
```

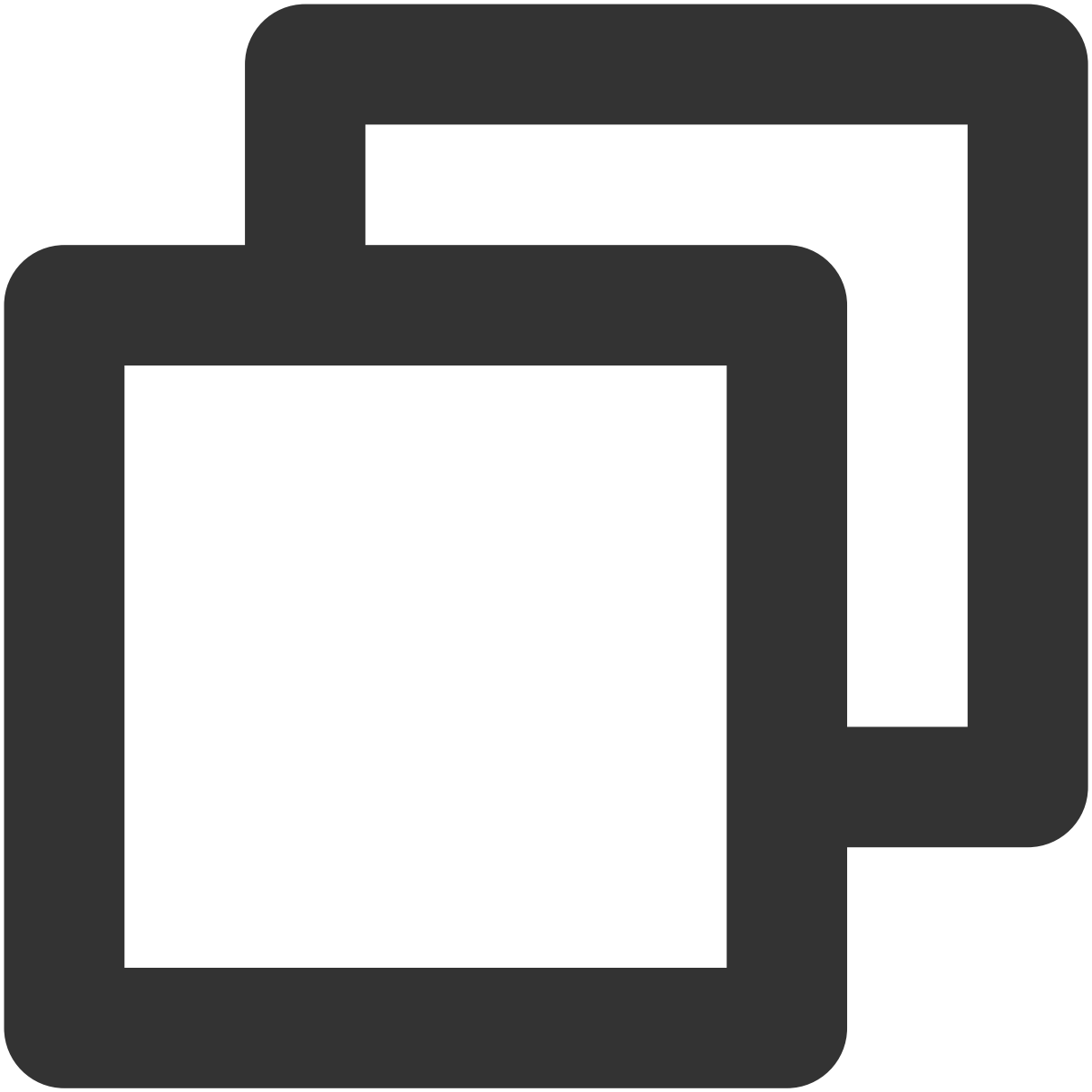
```
    // 设置NameServer的地址
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();
```

参数	说明
groupName	生产者组名称，建议使用对应的topic名字
nameserver	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在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

发送消息

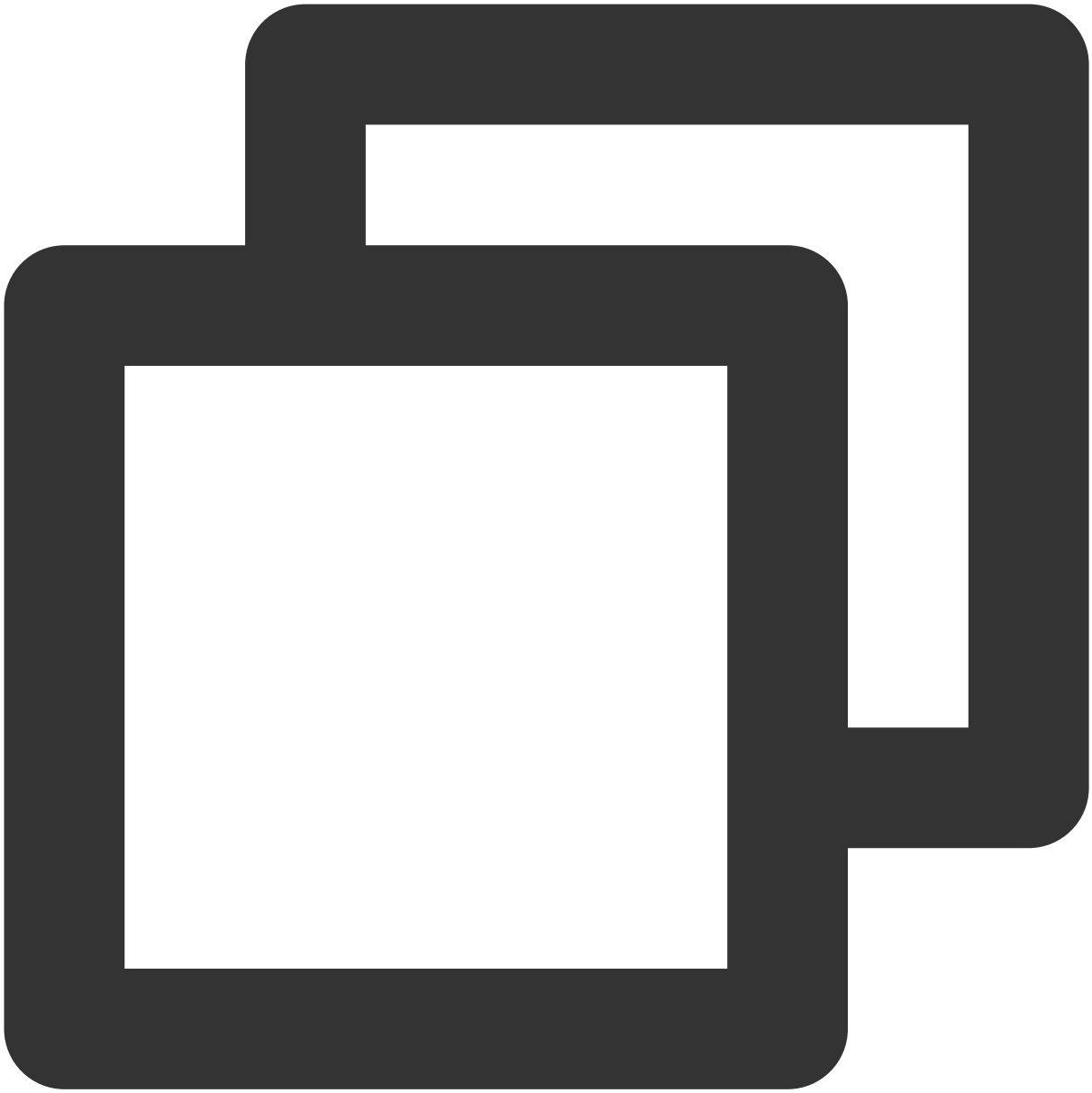
固定延迟级别的消息





```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message " + i).getB
    // 设置消息延迟等级
    message.setDelayTimeLevel(5);
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

任意延迟时间的消息



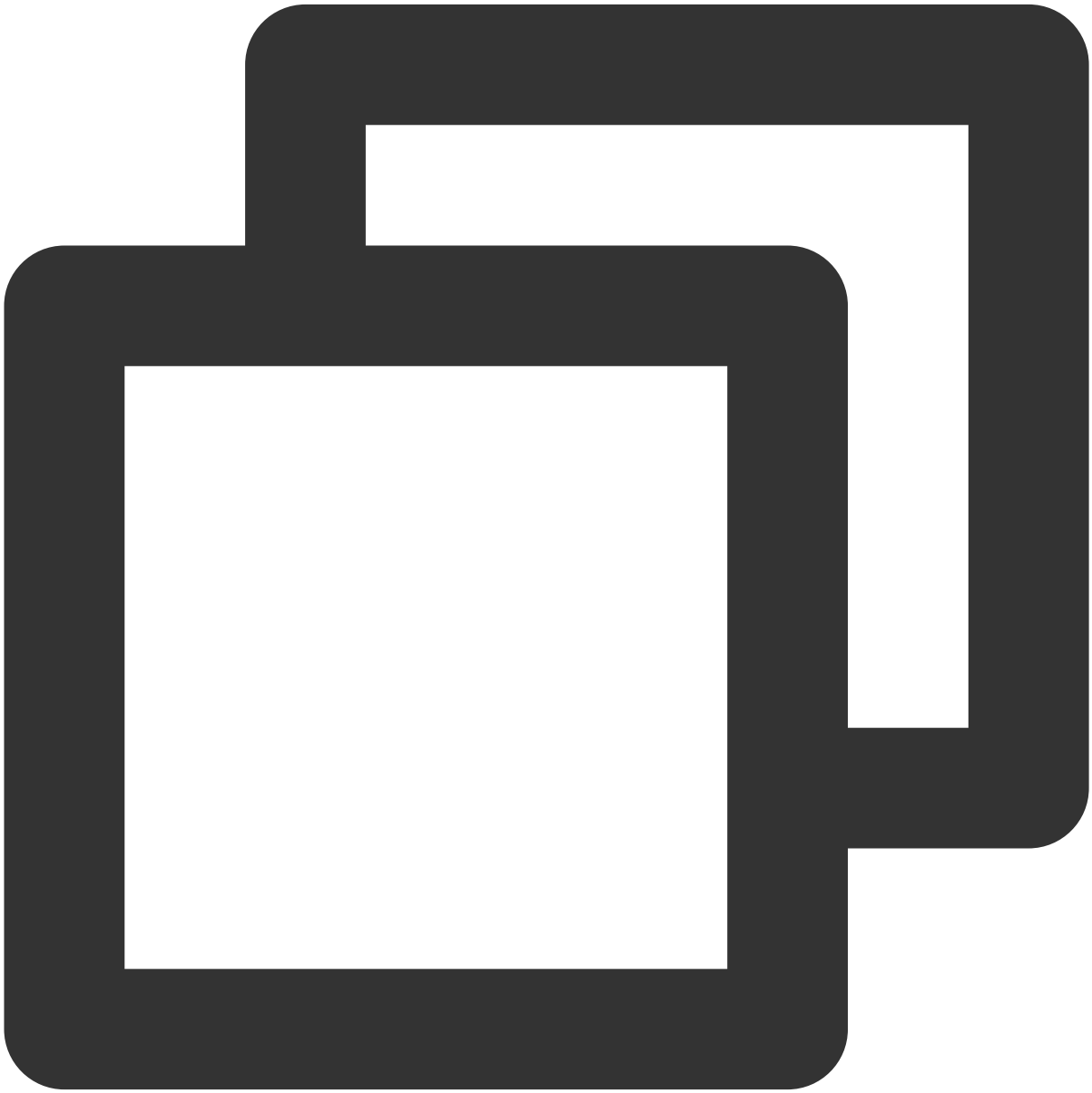
```
int totalMessagesToSend = 1;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello timer message " + i).getBytes
    // 设置发送消息的时间
    long timeStamp = System.currentTimeMillis() + 30000;
    // 若需要发送定时消息，则需要设置定时时间，消息将在指定时间进行投递，例如消息将在2022-08-08
    // 若设置的时间戳在当前时间之前，则消息将被立即投递给Consumer。
    //将 __STARTDELIVERTIME 设定到 msg 的属性中
    message.putUserProperty("__STARTDELIVERTIME", String.valueOf(timeStamp));
```

```
        // 发送消息
        SendResult sendResult = producer.send(message);
        System.out.println("sendResult = " + sendResult);
    }
```

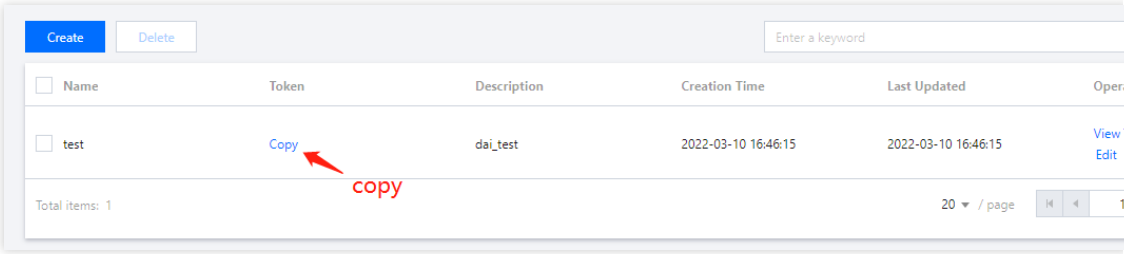
步骤3：消费消息

创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式

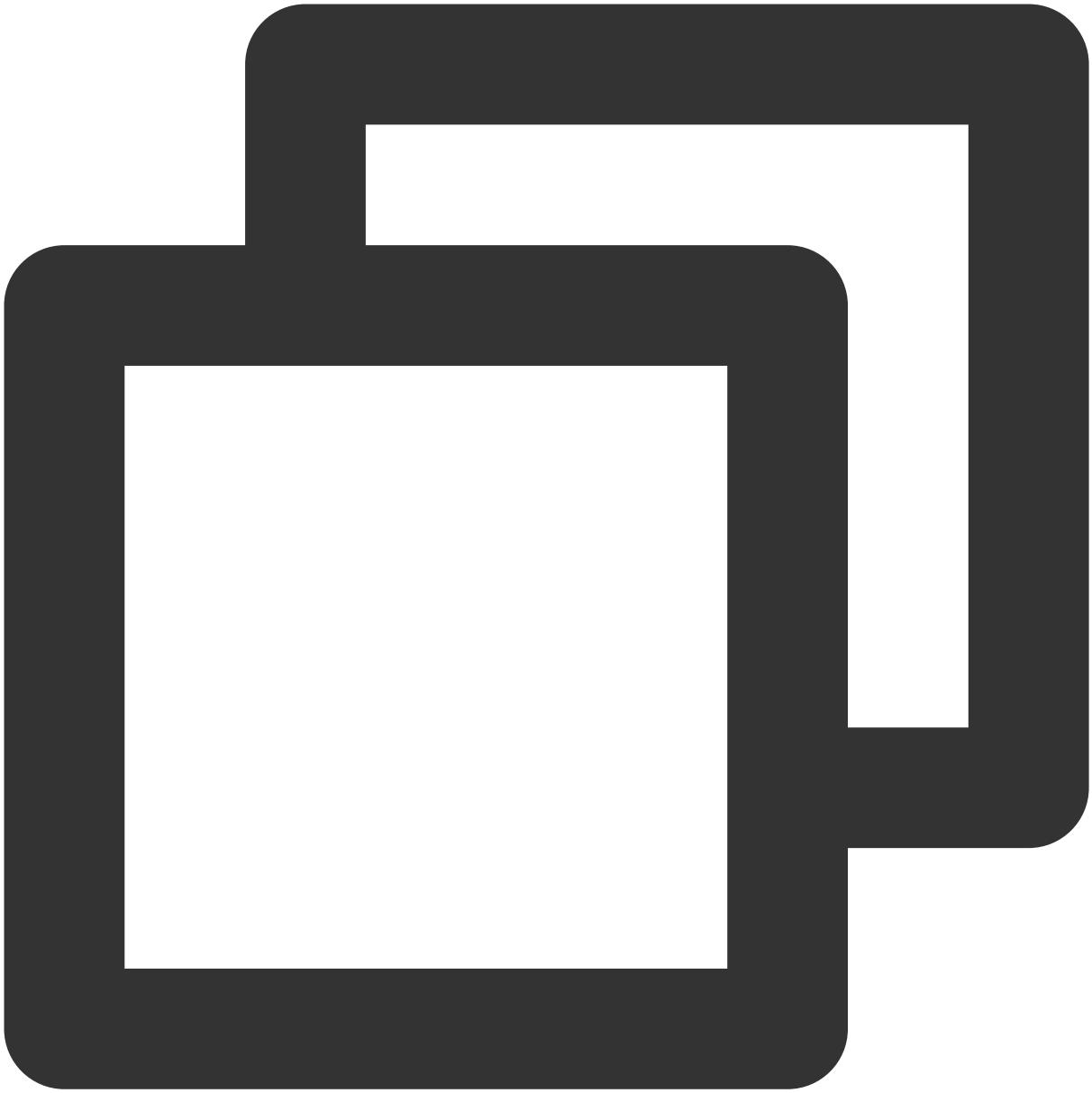


```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

参数	说明
groupName	生产者组名称，在控制台集群管理中Group 页签中复制。
nameserver	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在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

订阅消息

根据消费模式不同，订阅方式也有所区别。

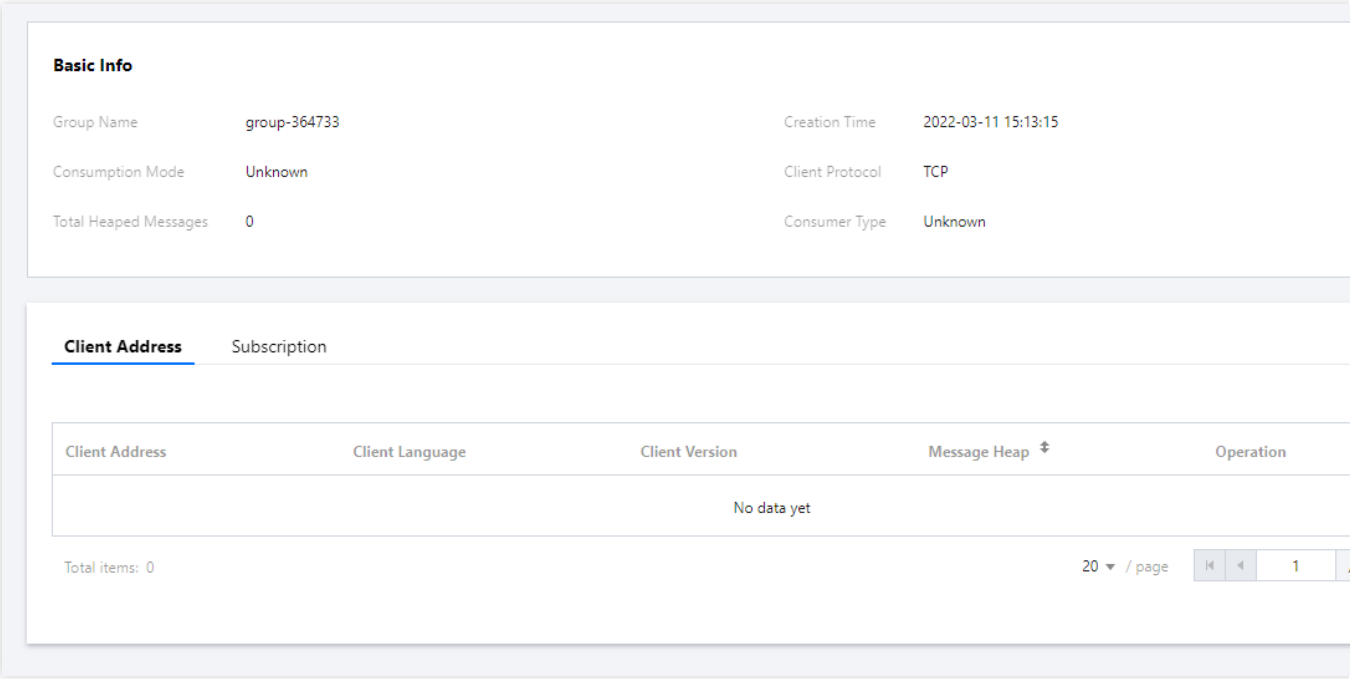


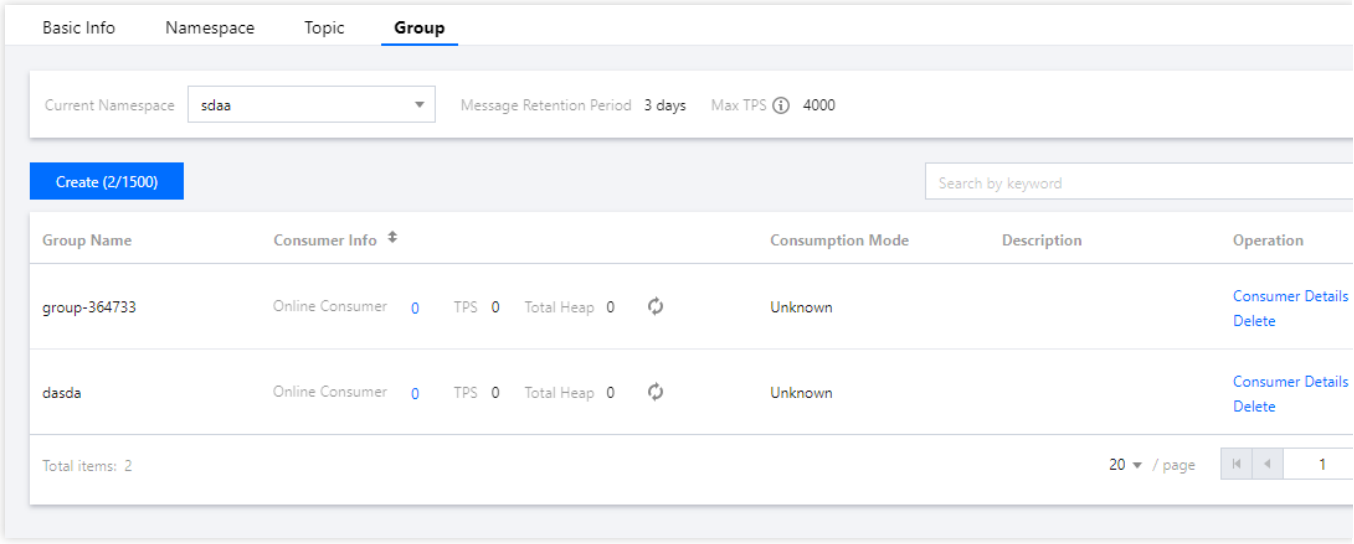
```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
            // 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();
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4：查看消费详情

登录 TDMQ 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**查看消费者详情**，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 或 RocketMQ 官方文档 。

发送与接收顺序消息

最近更新时间：2023-11-24 14:29:33

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现定时消息收发的操作过程。

前提条件

完成资源创建与准备（如果是全局顺序消息，需要创建单队列topic）
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

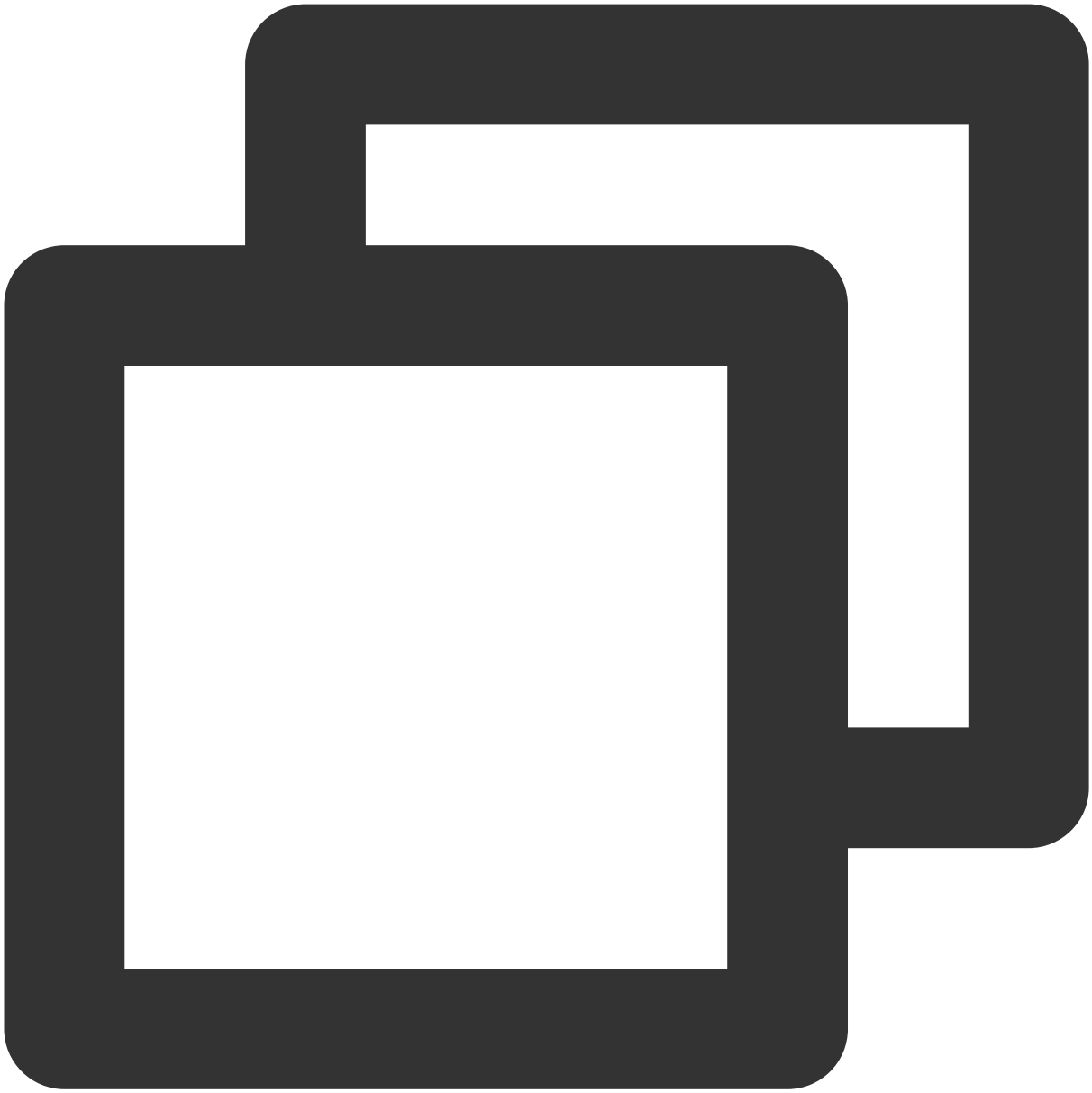
操作步骤

步骤1：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：

说明

依赖版本要求 ≥ 4.9.3, 当前建议为4.9.4

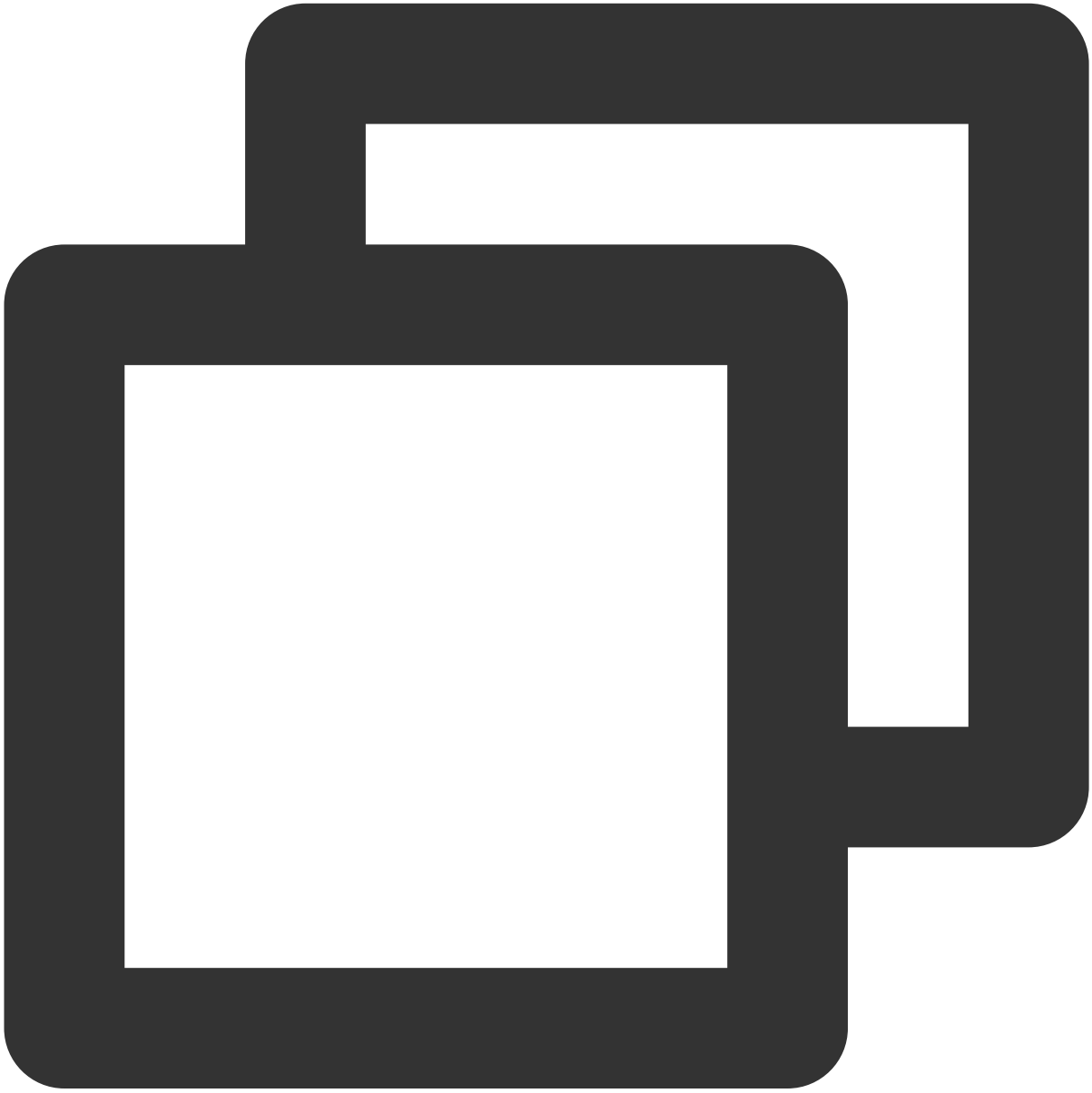


```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.4</version>
  </dependency>

  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-acl</artifactId>
      <version>4.9.4</version>
```

```
    </dependency>
```

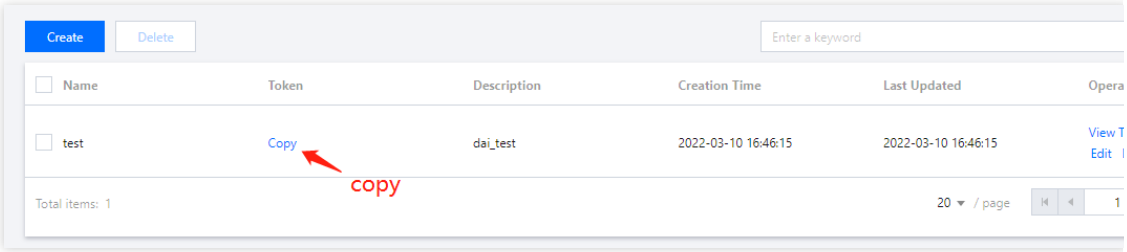
步骤2：生产消息

创建消息生产者



```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL权限
    );
```

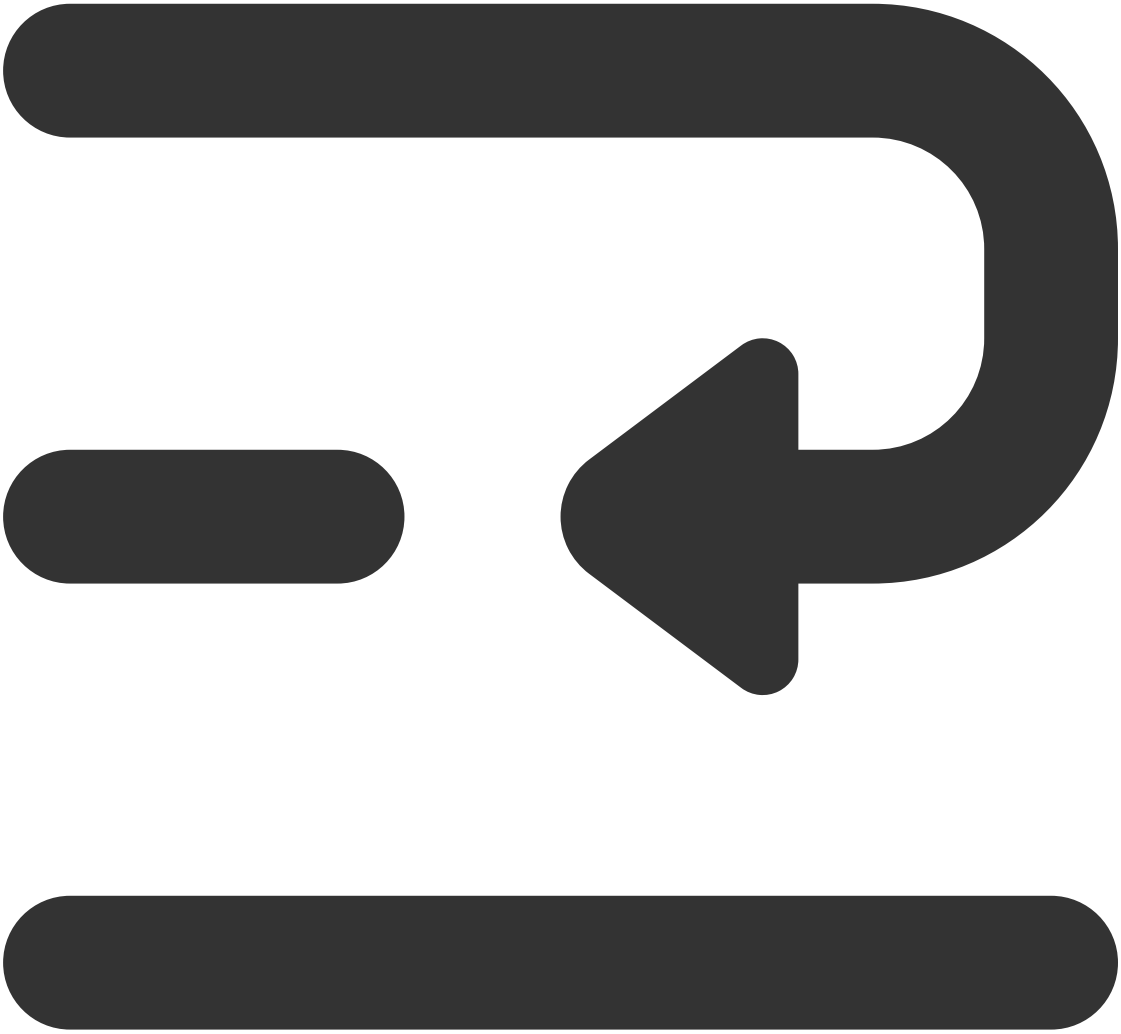
```
    // 设置NameServer的地址
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();
```

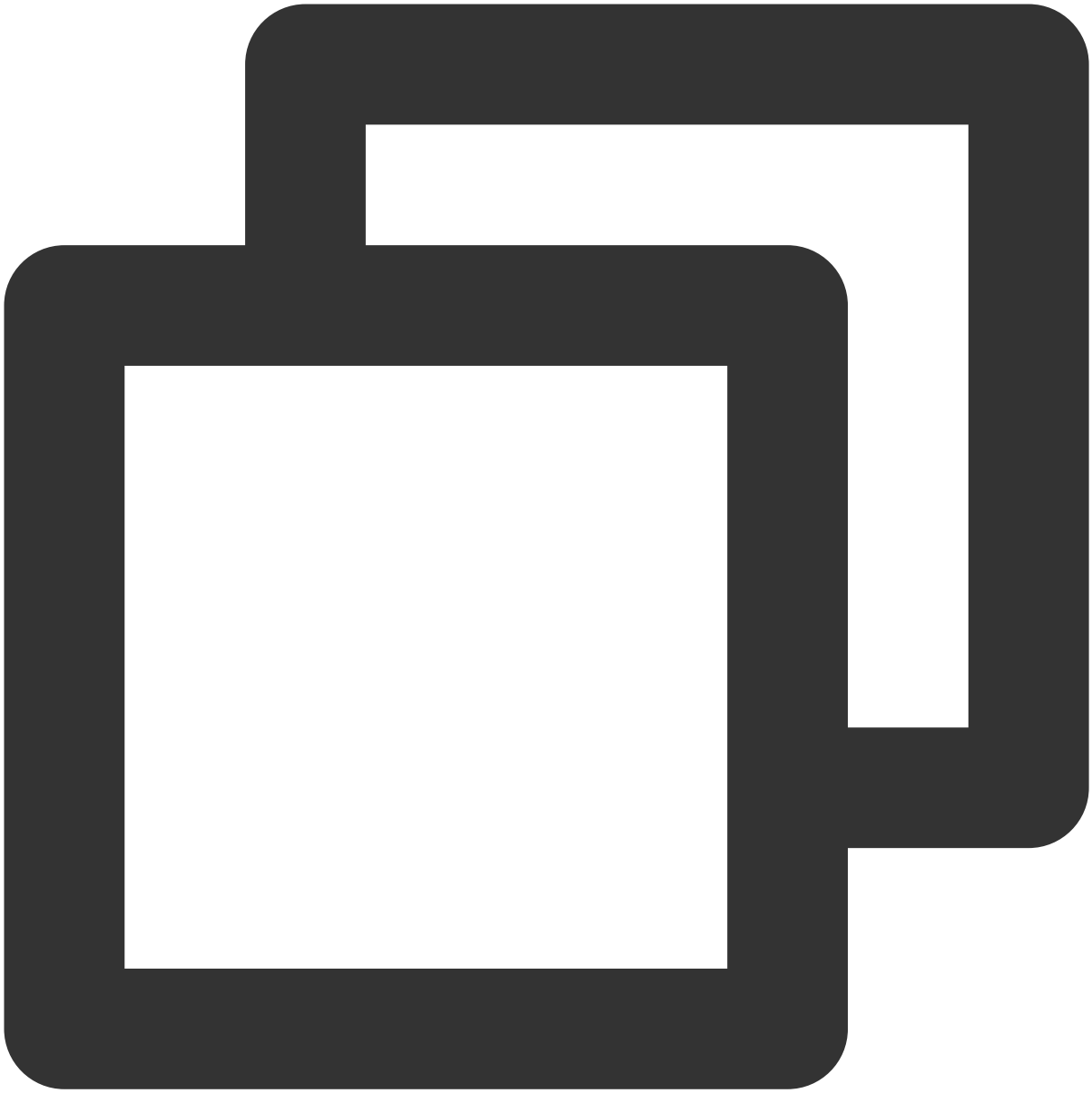
参数	说明
groupName	生产者组名称，建议使用对应的topic名字
nameserver	集群接入地址，在控制台集群管理页面的集群列表操作栏的接入地址处获取。新版共享集群与专享在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

发送消息

全局顺序消息

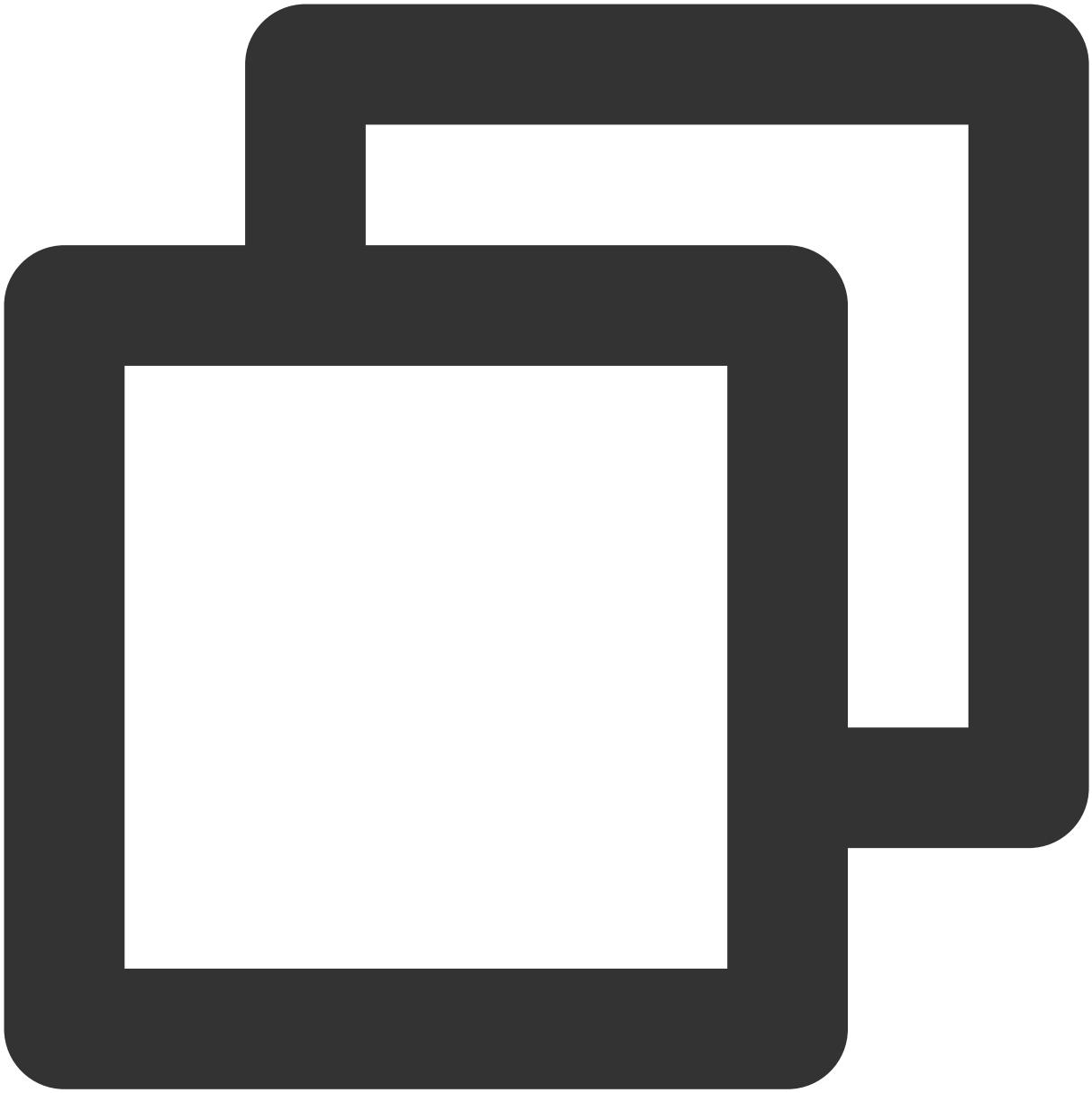
发送代码和简单的消息没有区别





```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message " + i).getB
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

分区有序的消息



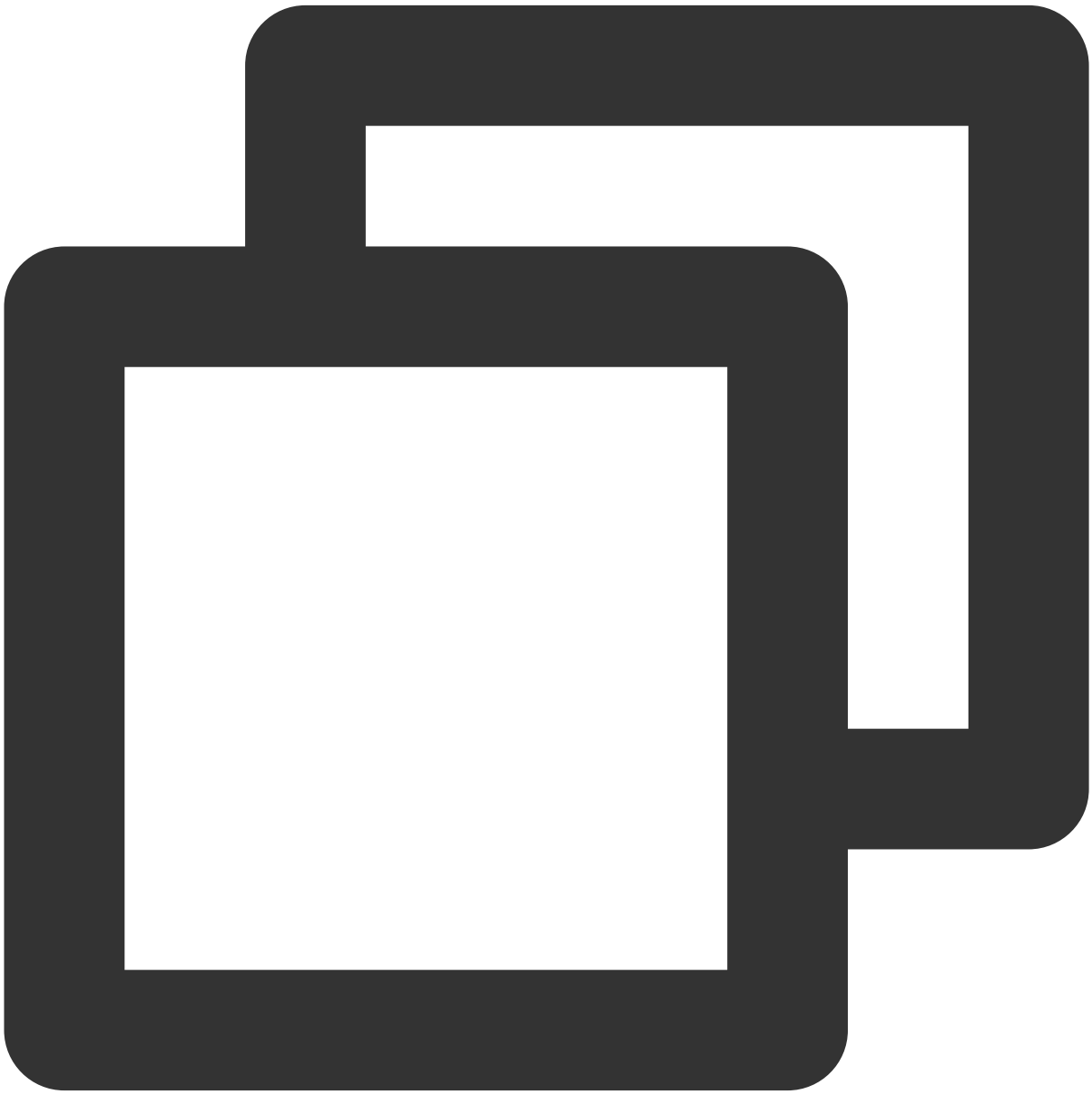
```
for (int i = 0; i < 3; i++) {
    int orderId = i % 3;
    // 构造消息示例
    Message msg = new Message(TOPIC_NAME, "your tag", "KEY" + i,("Hello RocketMQ "
    SendResult sendResult = producer.send(msg, new MessageQueueSelector() {
      @Override
      public MessageQueue select(List<MessageQueue> mqs, Message msg1, Object arg)
          Integer id = (Integer) arg;
          int index = id % mqs.size();
          return mqs.get(index);
      }
```

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

步骤3：消费消息

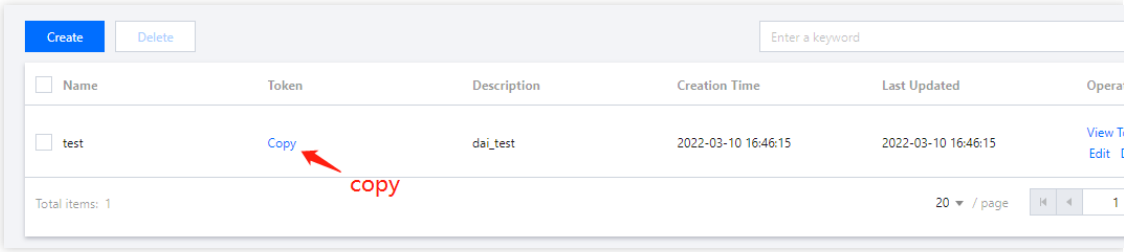
创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式



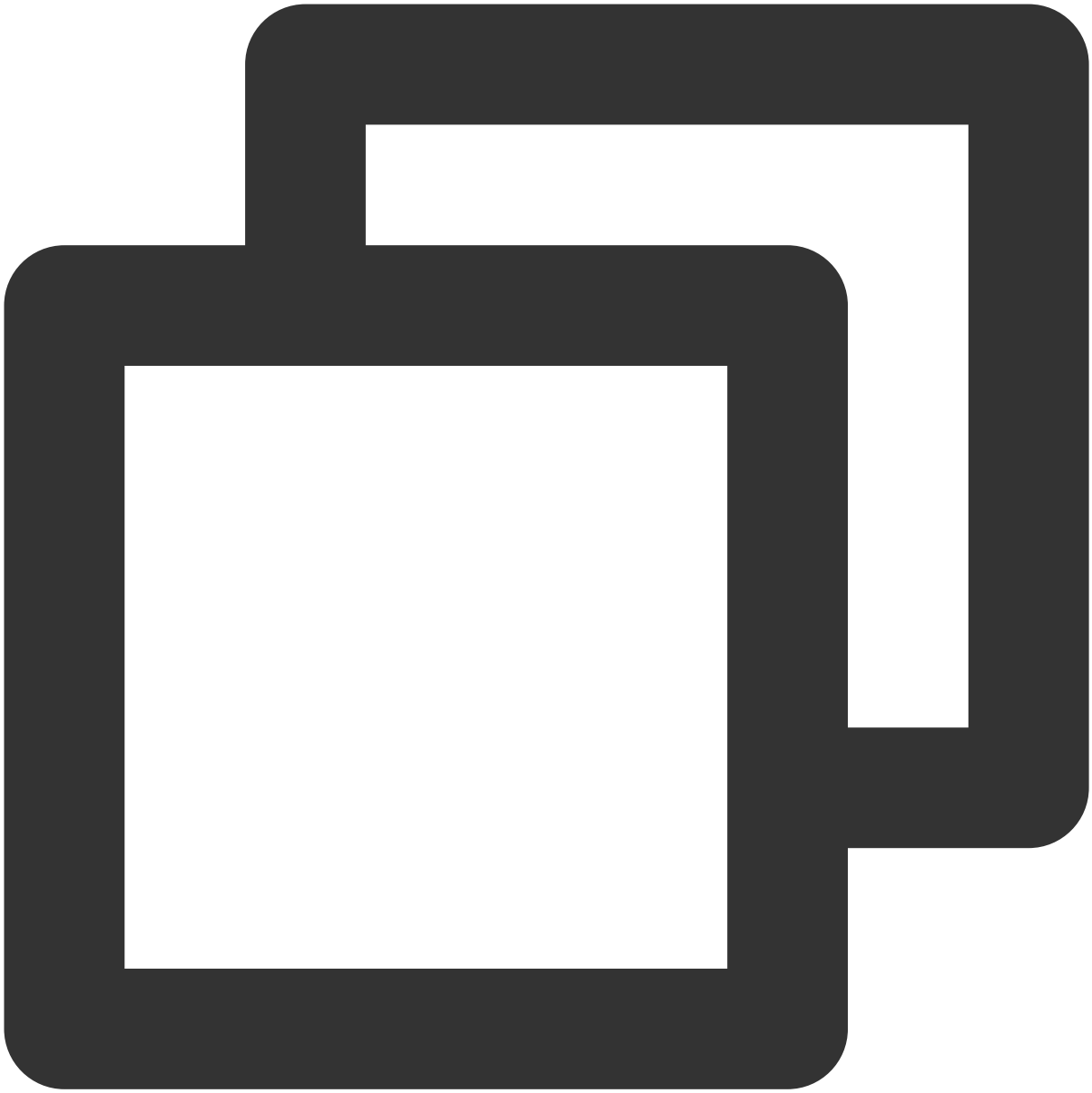
```
// 实例化消费者
```

```
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

参数	说明
groupName	生产者组名称，在控制台集群管理中Group 页签中复制。
nameserver	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在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

订阅消息

根据消费模式不同，订阅方式也有所区别。

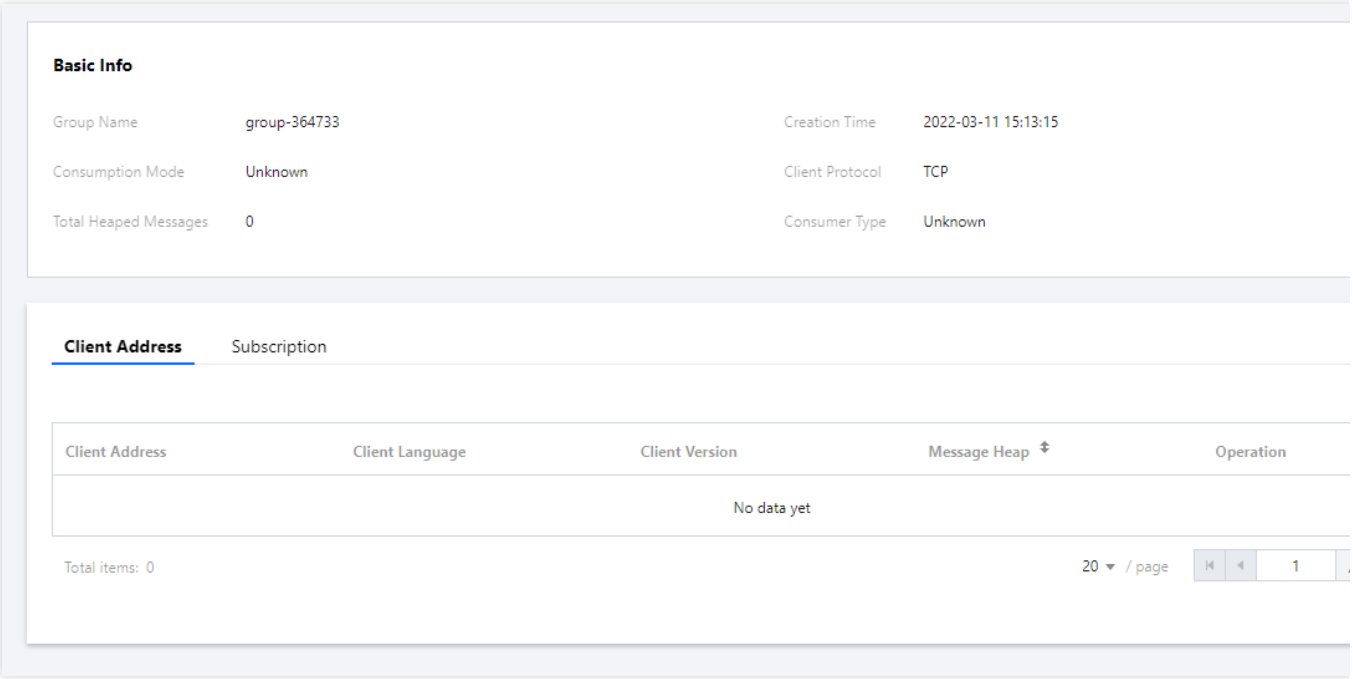


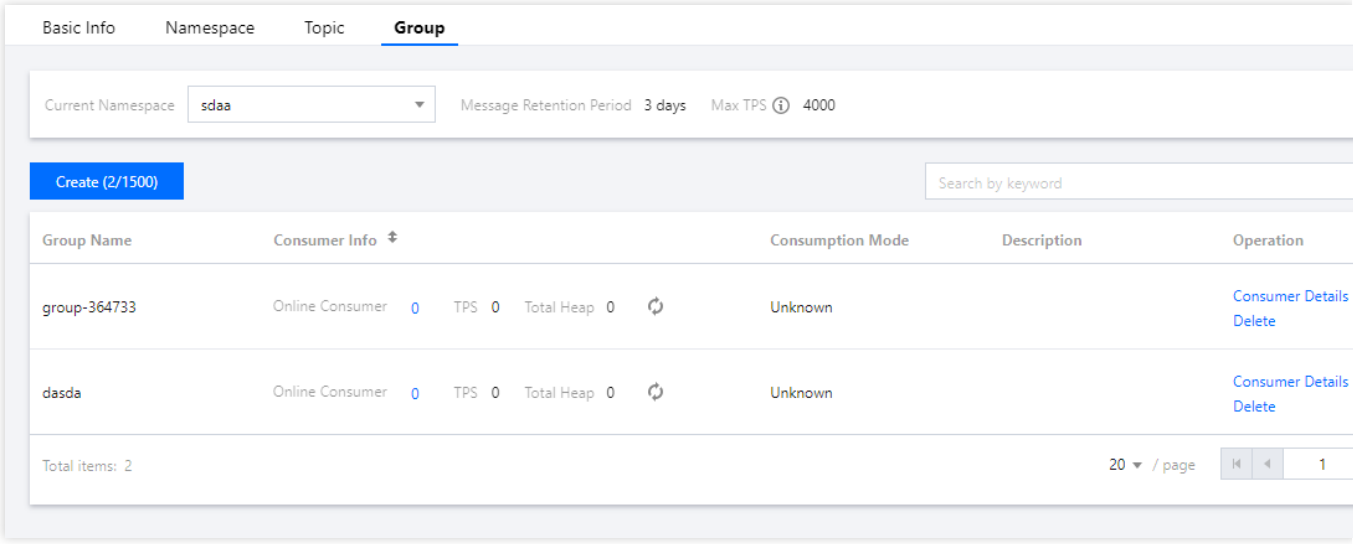
```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
            // 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();
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4：查看消费详情

登录 TDMQ 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**查看消费者详情**，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 或 RocketMQ 官方文档 。

发送与接收事务消息

最近更新时间：2023-11-24 14:30:01

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现事务消息收发的操作过程。

前提条件

完成资源创建与准备（如果是全局顺序消息，需要创建单队列topic）
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

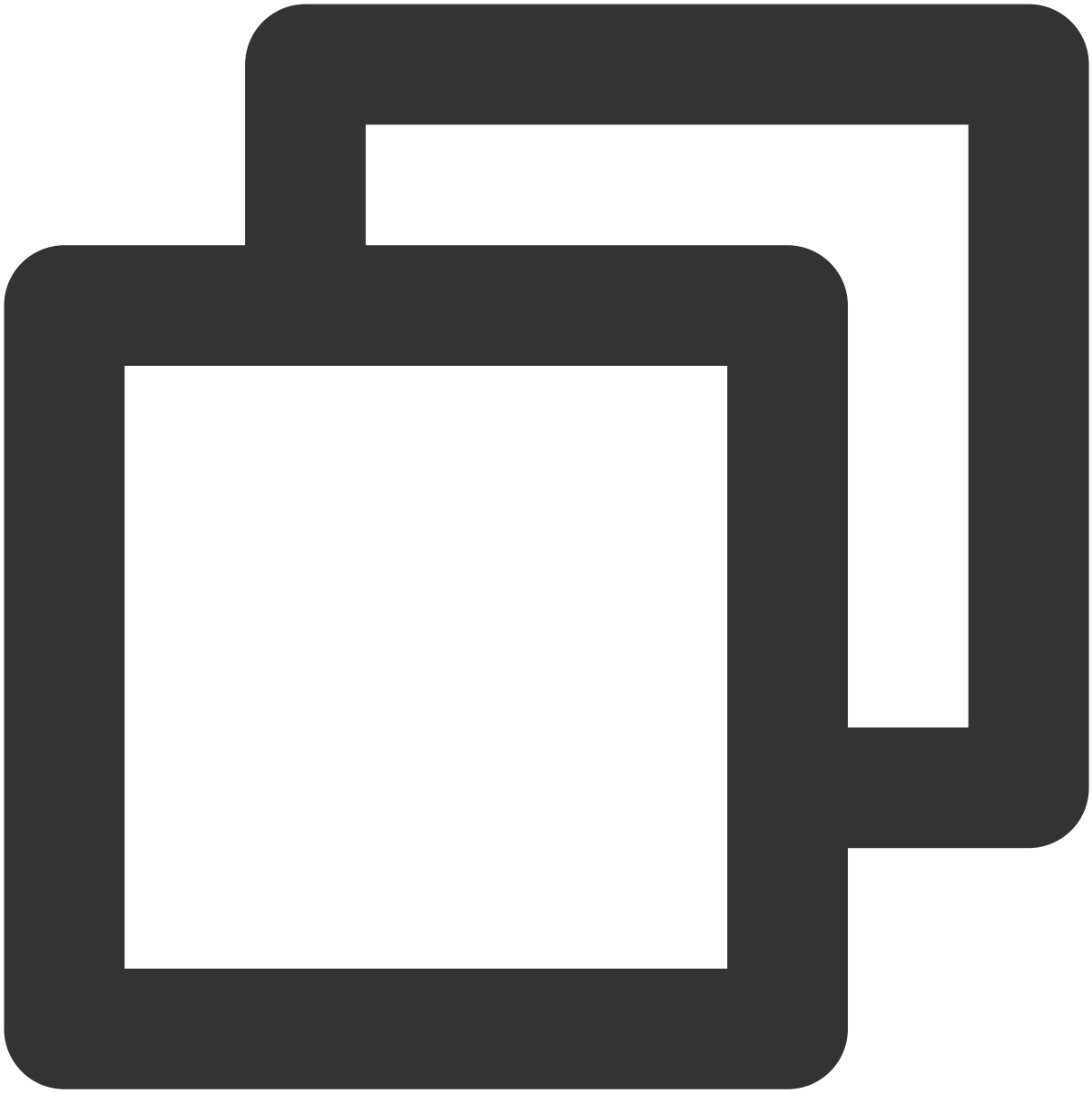
操作步骤

步骤1：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：

说明

依赖版本要求 ≥ 4.9.3, 当前建议为4.9.4

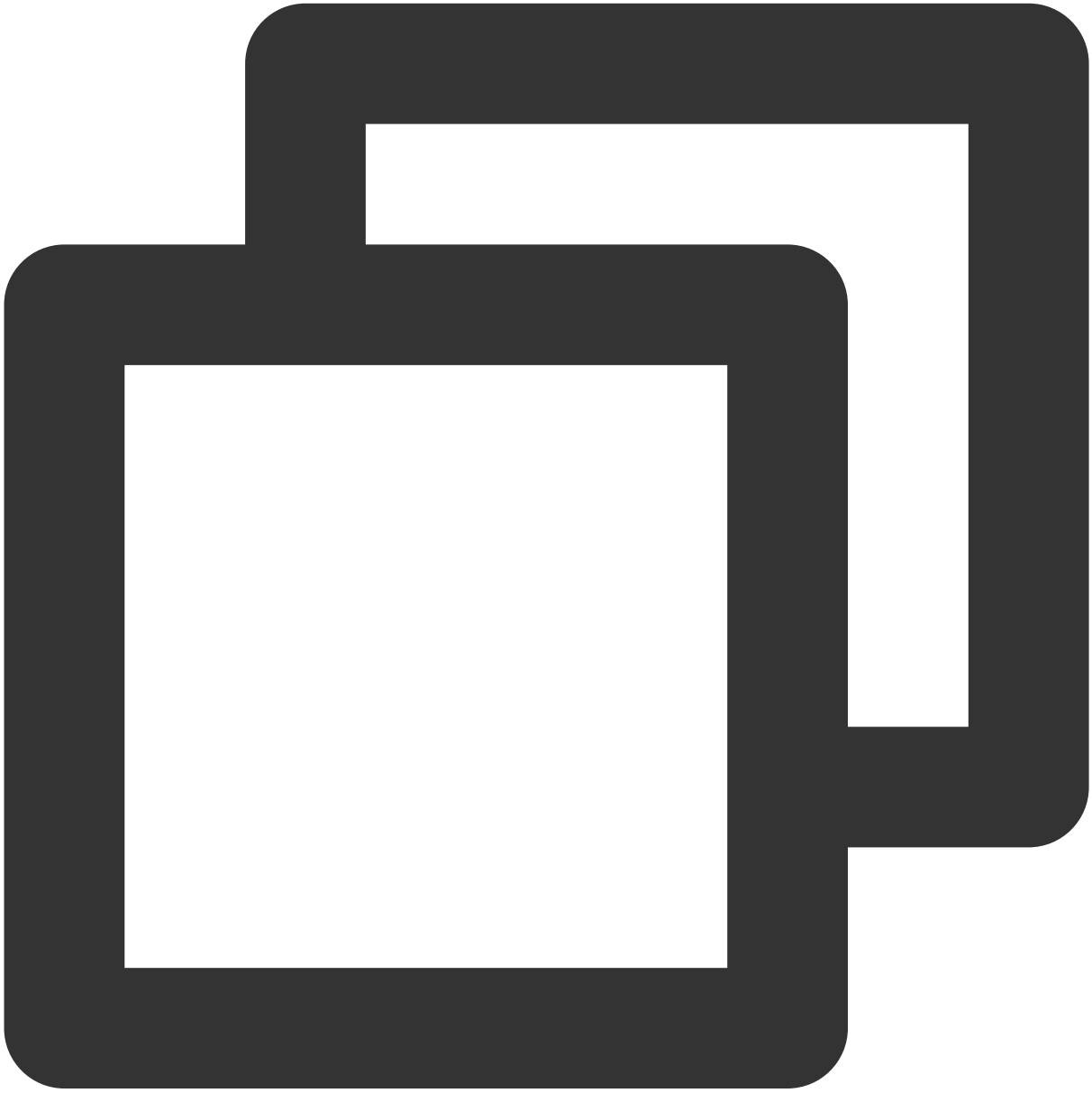


```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.4</version>
  </dependency>

  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-acl</artifactId>
      <version>4.9.4</version>
```

```
    </dependency>
```

步骤2：生产消息

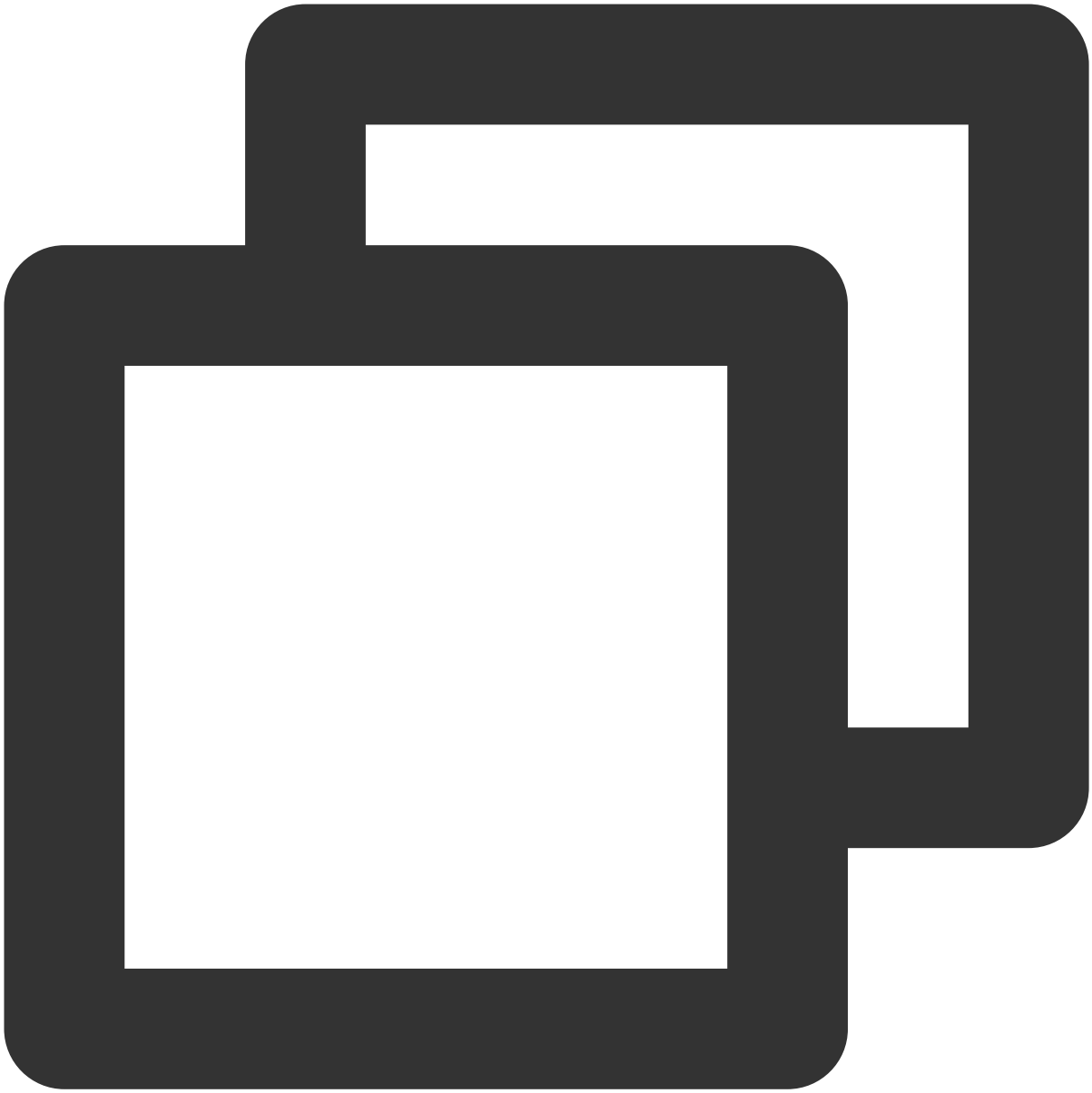
实现 TransactionListener



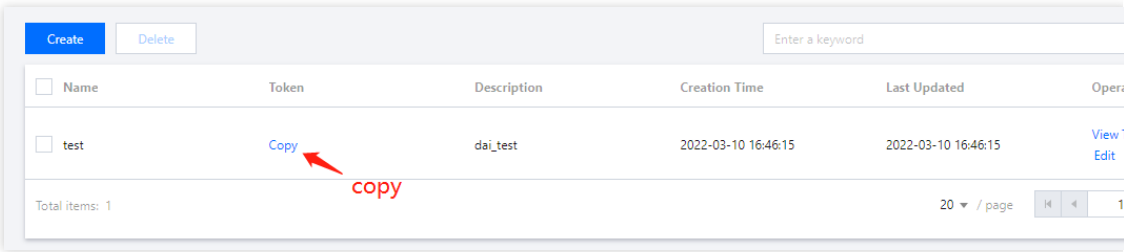
```
public class TransactionListenerImpl implements TransactionListener {

    //半消息发送成功后，回调该方法执行本地事务
    @Override
    public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
```

```
        //这里执行数据库事务，如果成功，就返回成功，否则返回未知，或者回滚，等待回查
        return LocalTransactionState.UNKNOW;
    }
    //回查本地事务
    @Override
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {
        //这里查询本地db的数据状态，然后决定是否是否提交
        return LocalTransactionState.COMMIT_MESSAGE;
    }

}
```

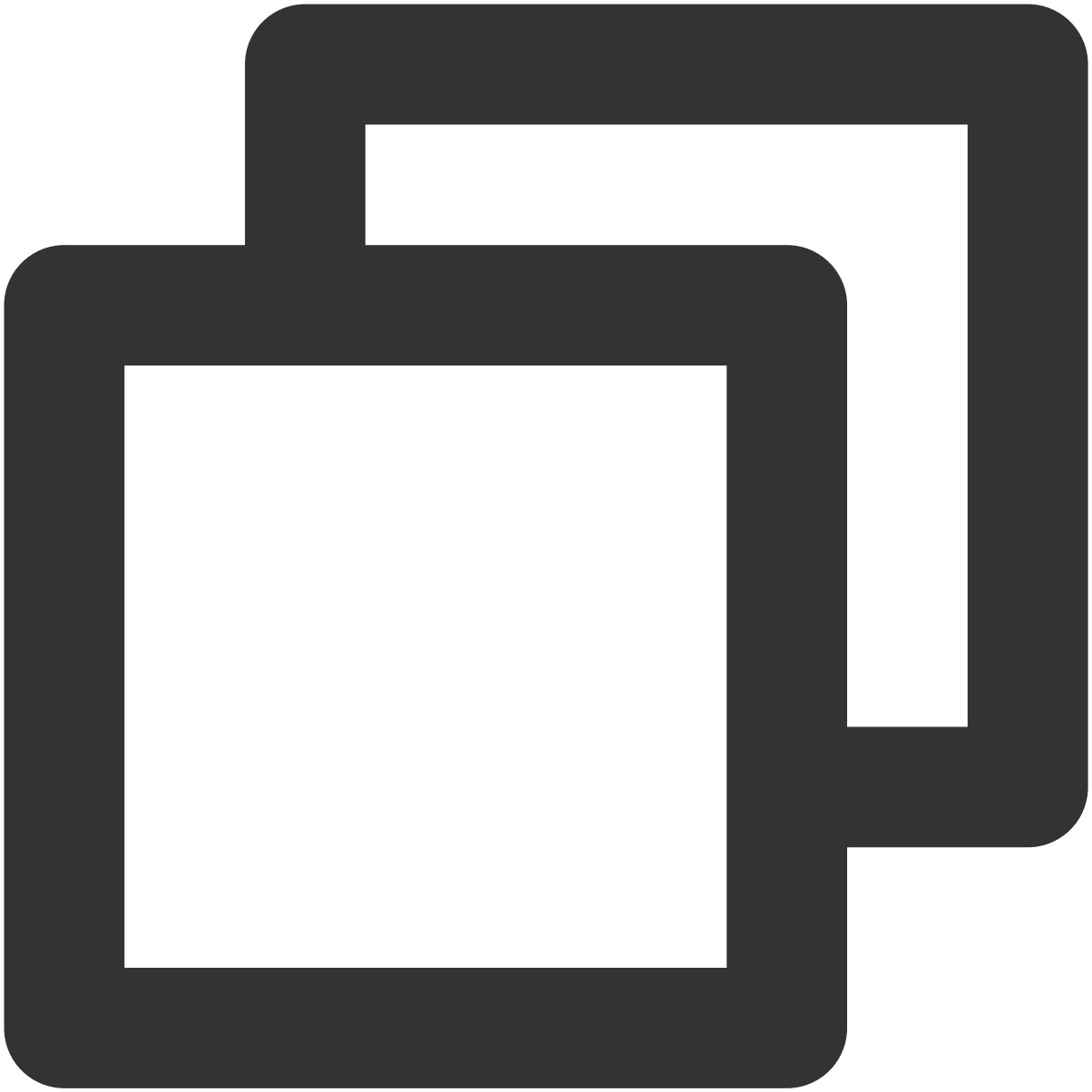
创建消息生产者



```
//需要用户实现一个TransactionListener 实例,
TransactionListener transactionListener = new TransactionListenerImpl();
// 实例化事务消息生产者
ProducerTransactionMQProducer producer = new TransactionMQProducer("transaction_gro
// ACL权限
new AclClientRPCHook(new SessionCredentials(ClientCreater.ACCESS_KEY, ClientCreater
// 设置NameServer的地址
producer.setNamesrvAddr(ClientCreater.NAMESERVER);
producer.setTransactionListener(transactionListener);
producer.start();
```

参数	说明
groupName	生产者组名称，建议使用对应的topic名字
nameserver	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在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

发送消息

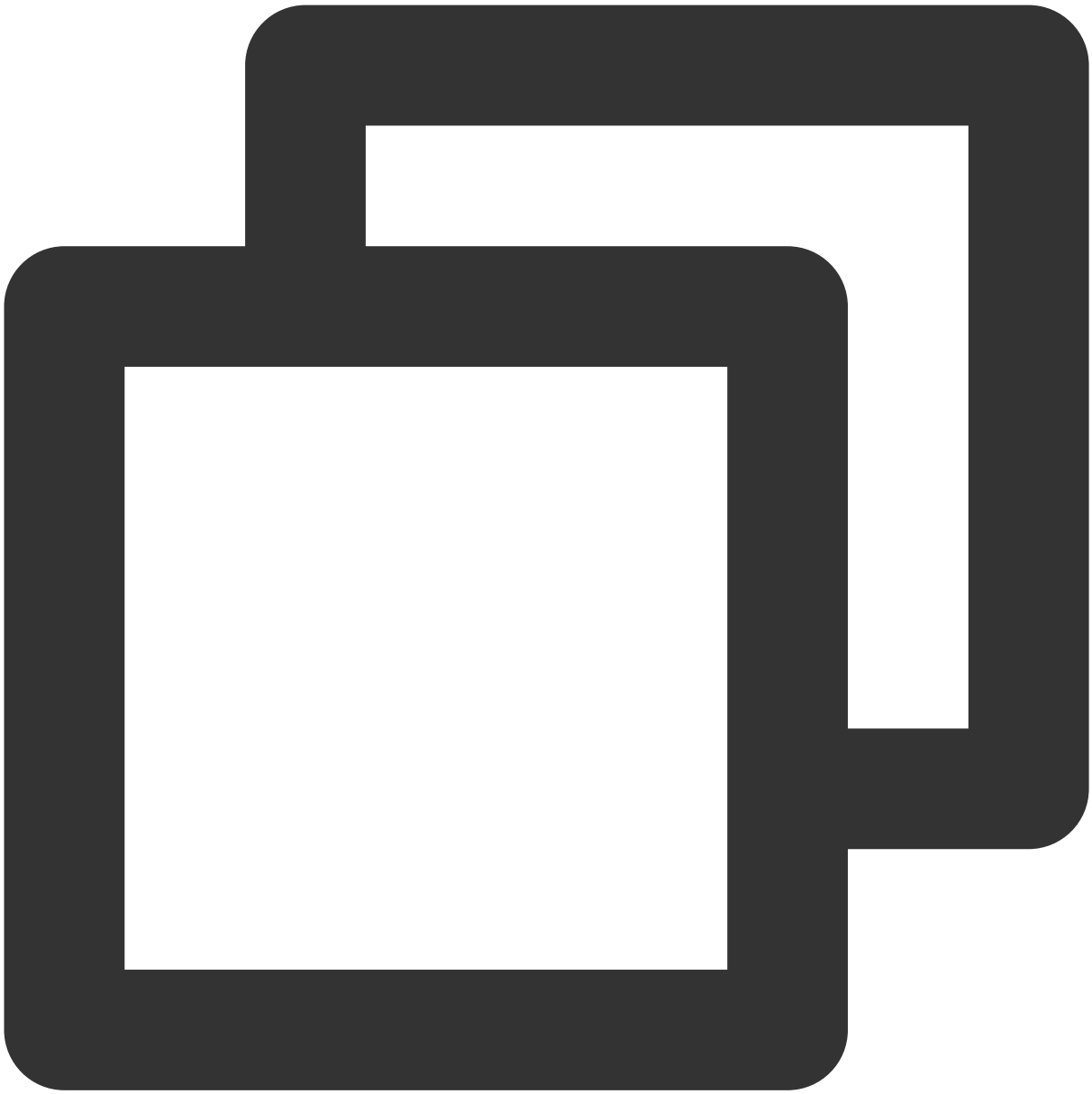


```
for (int i = 0; i < 3; i++) {
    // 构造消息示例
    Message msg = new Message(TOPIC_NAME, "your tag", "KEY" + i,("Hello RocketMQ "
    SendResult sendResult = producer.sendMessageInTransaction(msg,null);
     System.out.printf("%s%n", sendResult);
}
```

步骤3：消费消息

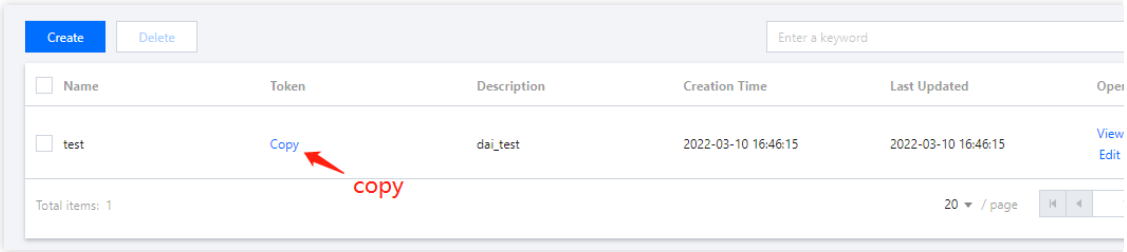
创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式



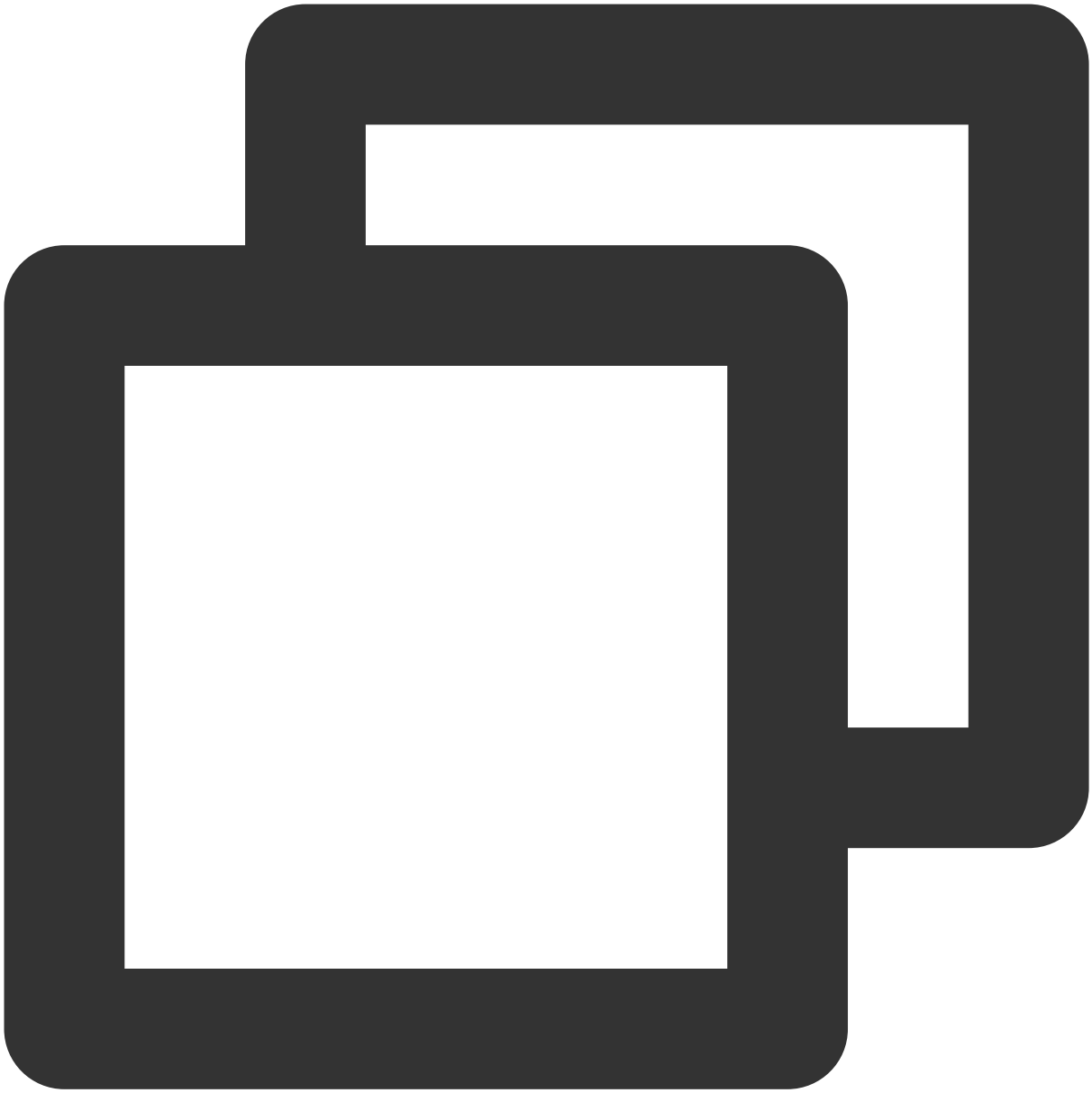
```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, c
           // 消息处理逻辑
           System.out.printf("%s Receive transaction messages: %s %n", Thread.curre
```

```
        // 标记该消息已经被成功消费
        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
```

参数	说明
groupName	生产者组名称，在控制台集群管理中Group 页签中复制。
nameserver	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在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

订阅消息

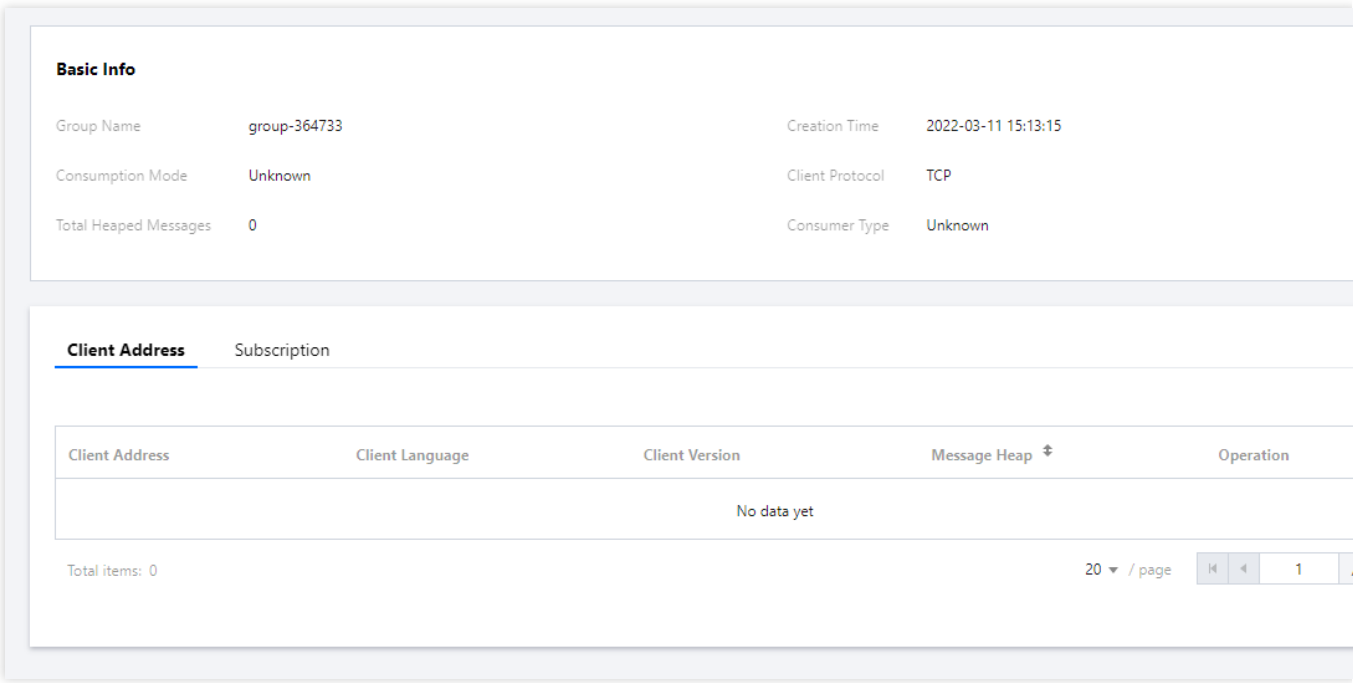
根据消费模式不同，订阅方式也有所区别。

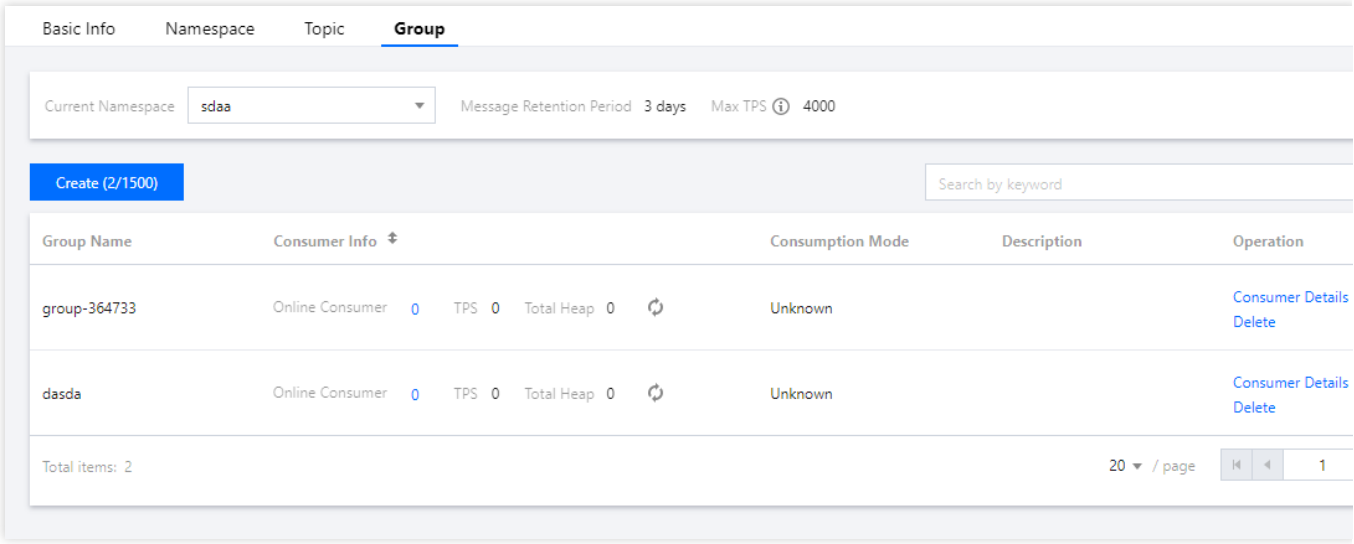


```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
            // 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：查看消费详情

登录 TDMQ 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**查看消费者详情**，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 或 RocketMQ 官方文档 。

发送与接收过滤消息

最近更新时间：2023-03-28 10:15:40

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现过滤消息收发的操作过程。目前支持两种，分别是Tag 和 SQL 方式。

前提条件

完成资源创建与准备（如果是全局顺序消息，需要创建单队列 topic）
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
参考普通消息的发送和接收

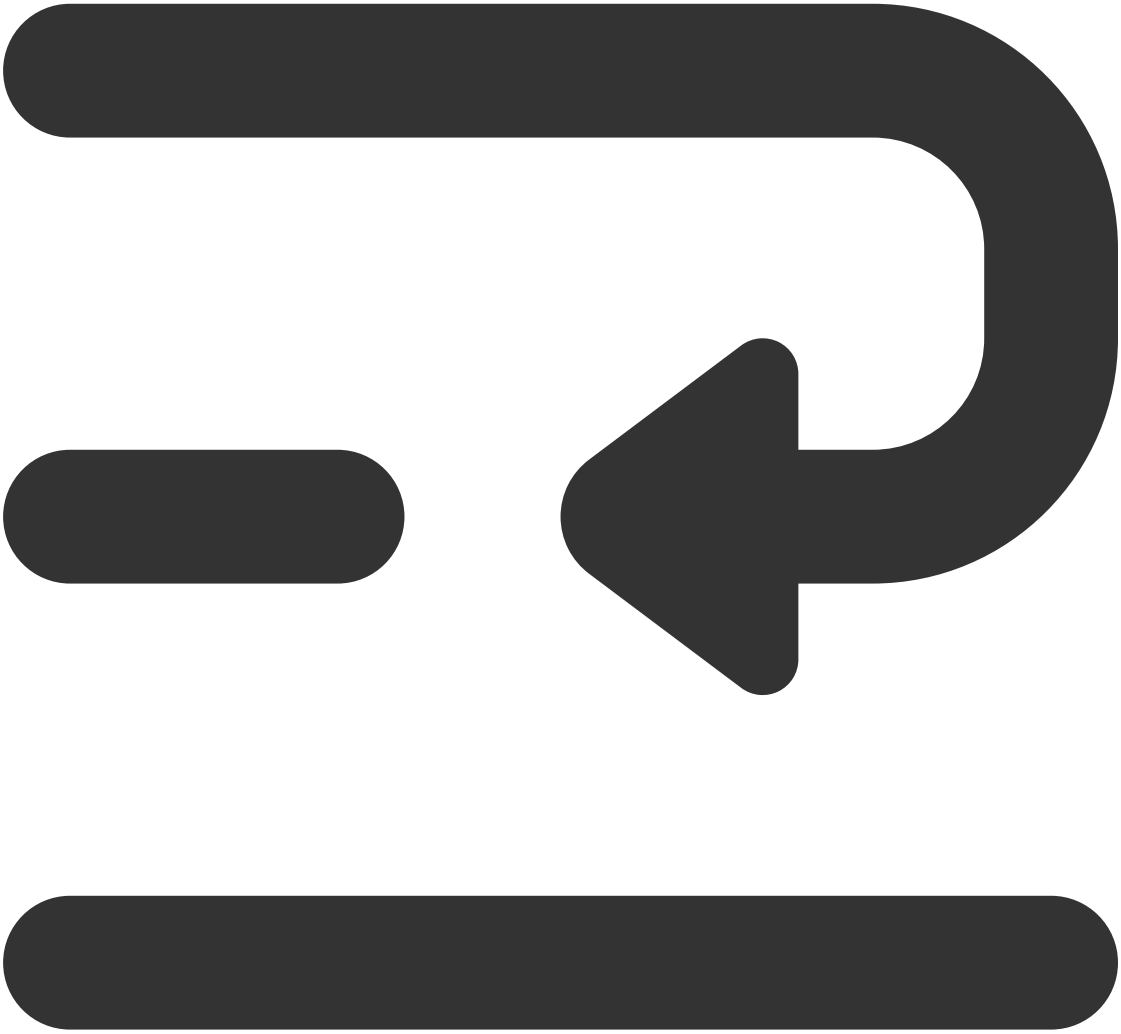
TAG使用步骤

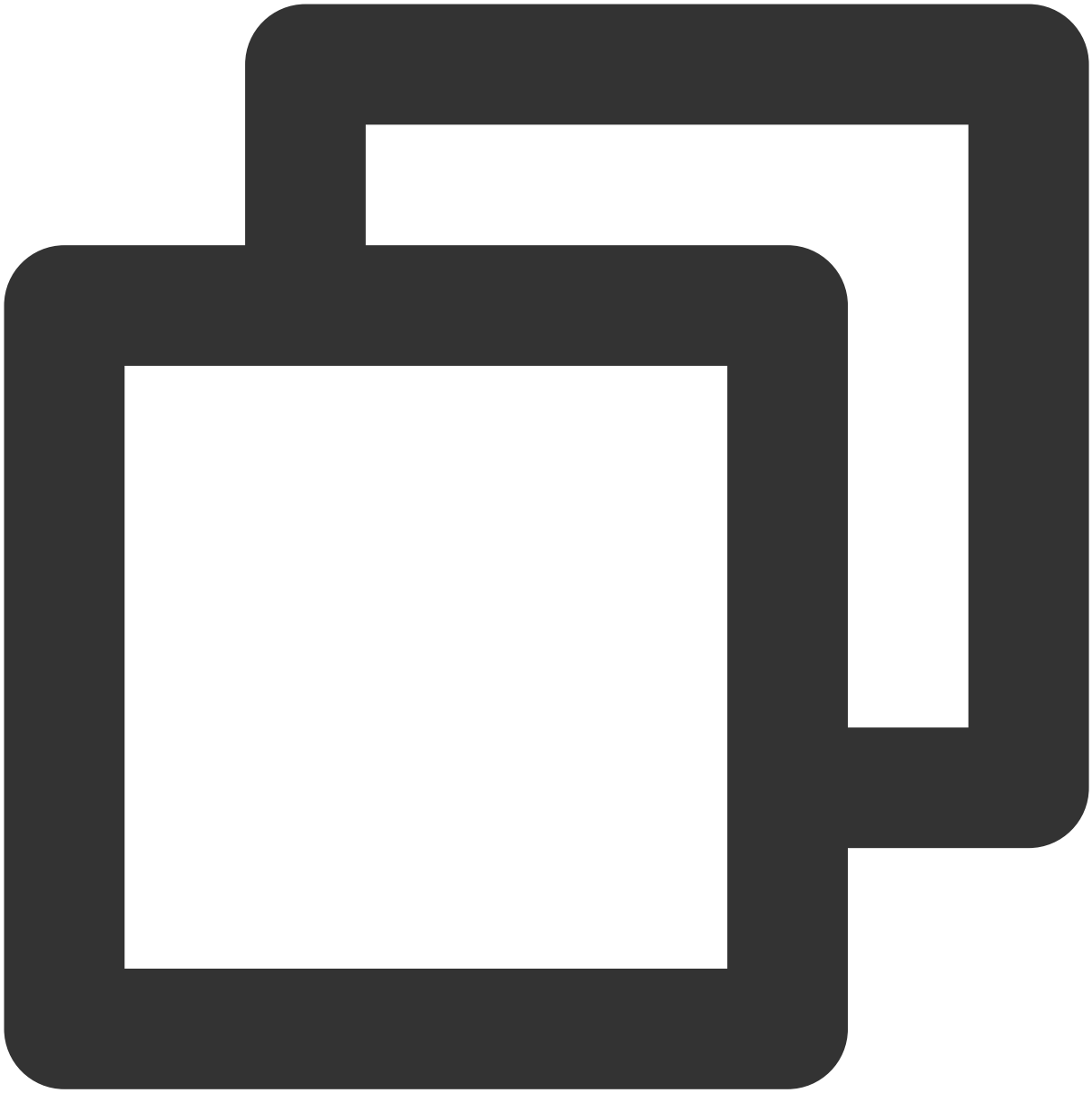
创建生产者消费者的基础代码不在赘述，可以参考普通消息的发送和接收即可。
对于生产消息，主要是在构造消息体的时候，带上 TAG。
对于消费消息，主要是订阅的时候，带上单个 TAG 或者 * 或者多个 TAG 表达式。

步骤1：生产消息

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 TAG，仅允许一个。

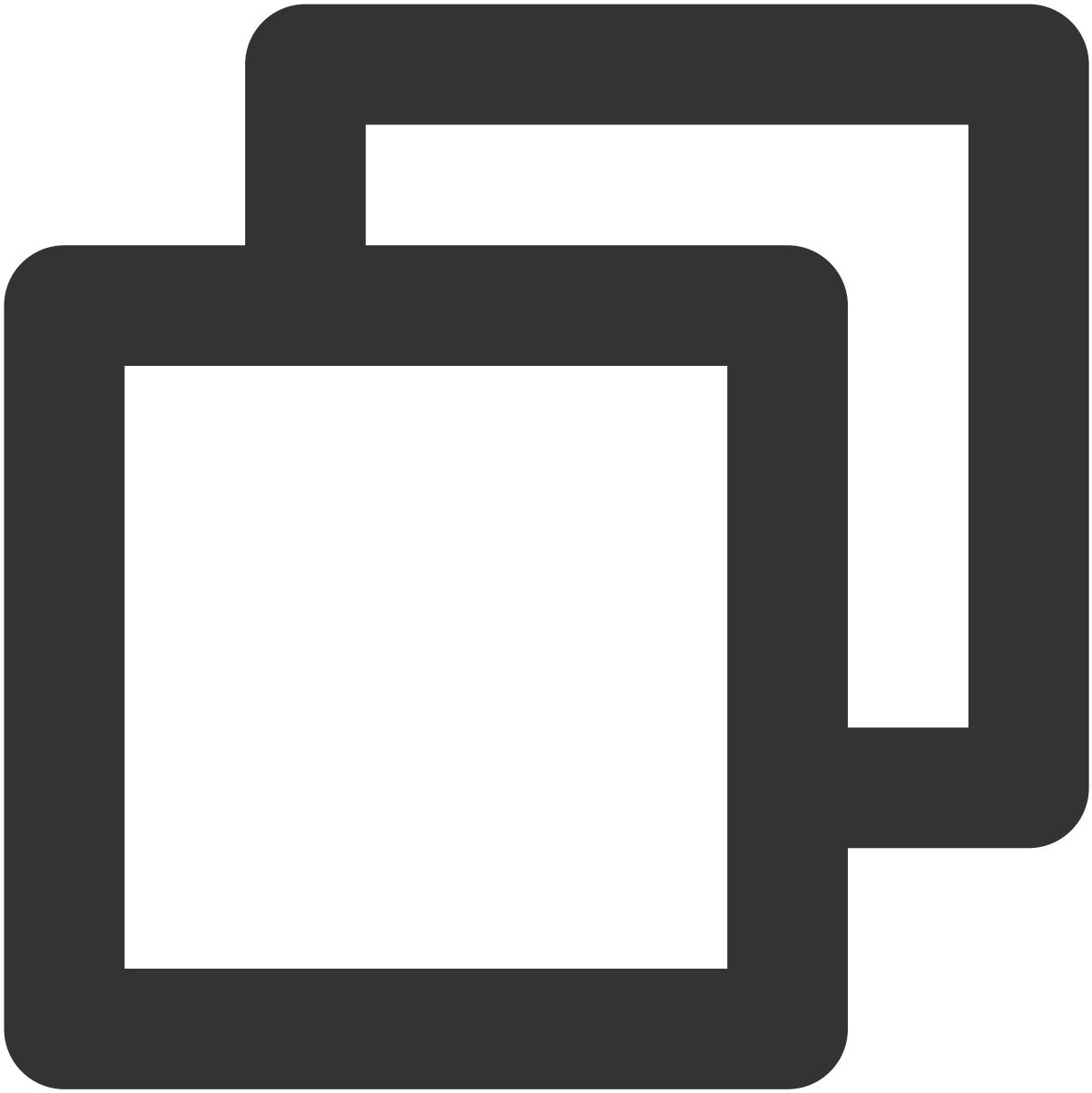




```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME, "Tag1", "Hello RocketMQ.".getBytes(Standa
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

步骤2：消费消息

订阅消息



```
// 订阅topic 订阅所有的TAG
pushConsumer.subscribe(topic_name, "*");

//订阅指定的TAG
//pushConsumer.subscribe(TOPIC_NAME, "Tag1");

//订阅多个TAG
//pushConsumer.subscribe(TOPIC_NAME, "Tag1||Tag2");
```

```
// 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(
            // 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();
```

参数	说明
topic_name	在控制台集群管理中 Topic 页签中复制具体 Topic 名称。
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

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。

SQL 使用步骤

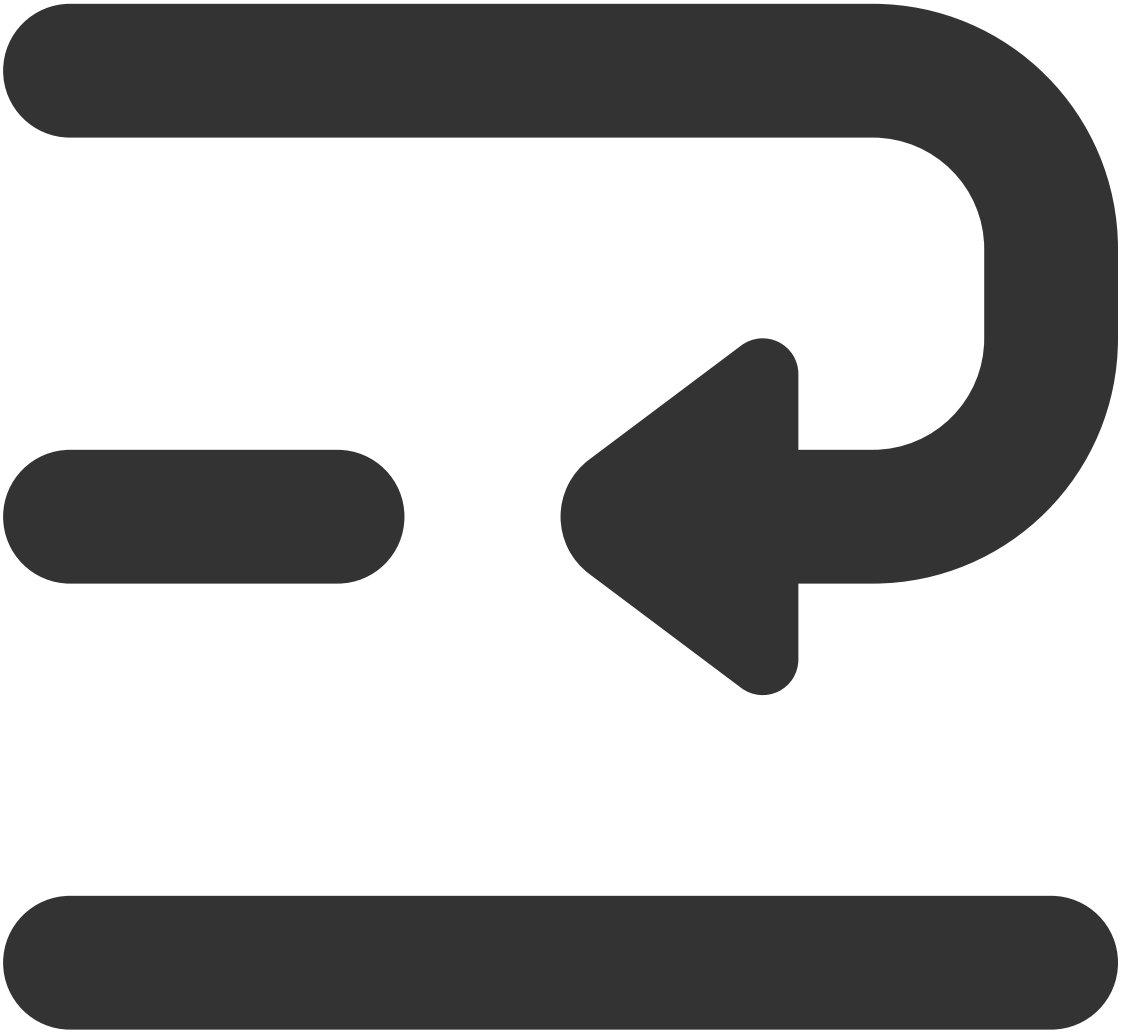
创建生产者消费者的基础代码不在赘述，可以参考普通消息的发送和接收即可。

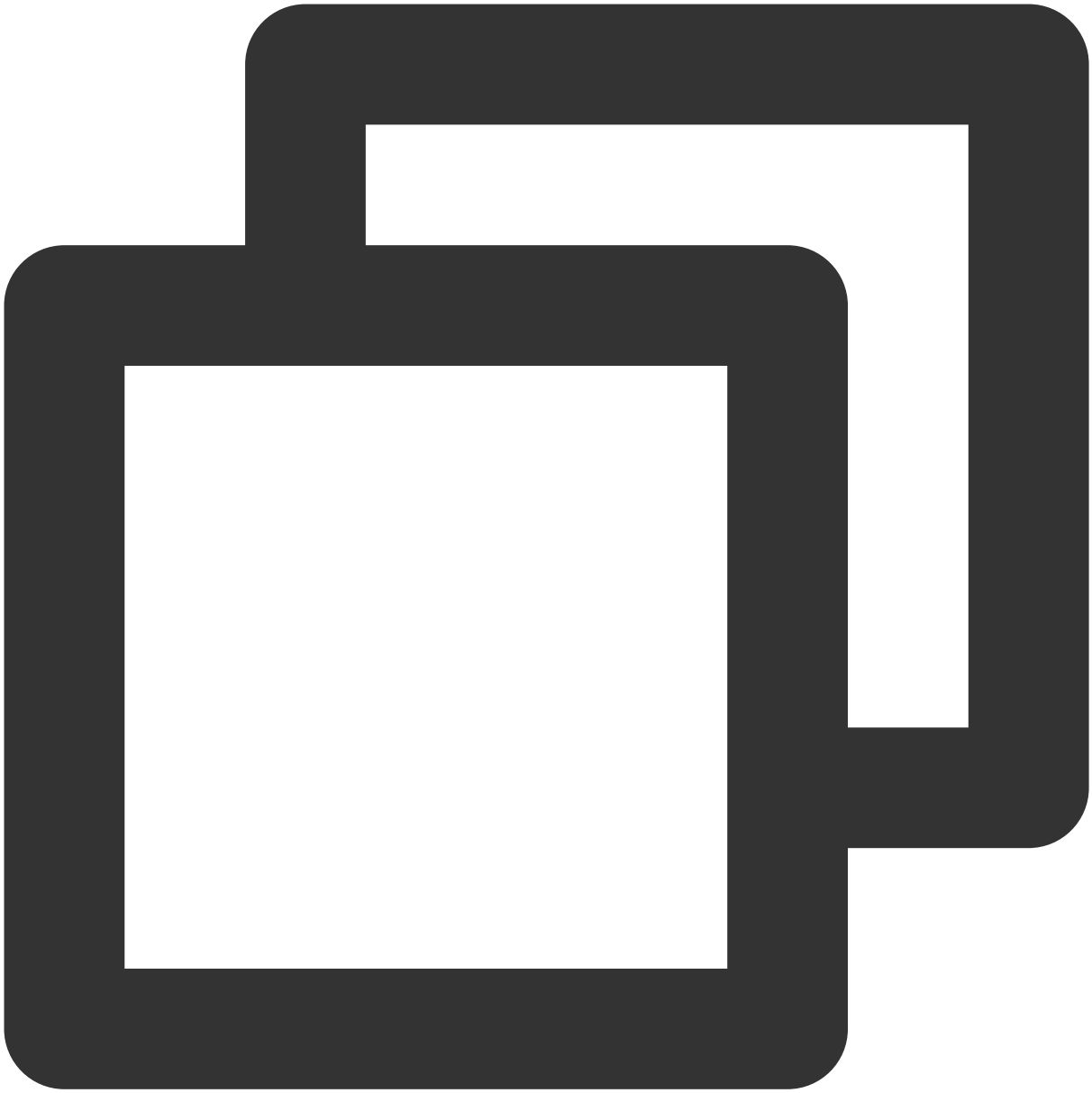
对于生产消息，主要是在构造消息体的时候，带上用户自定义的 properties。

对于消费消息，主要是订阅的时候，带上对应的 SQL 表达式。

步骤1：生产消息

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。

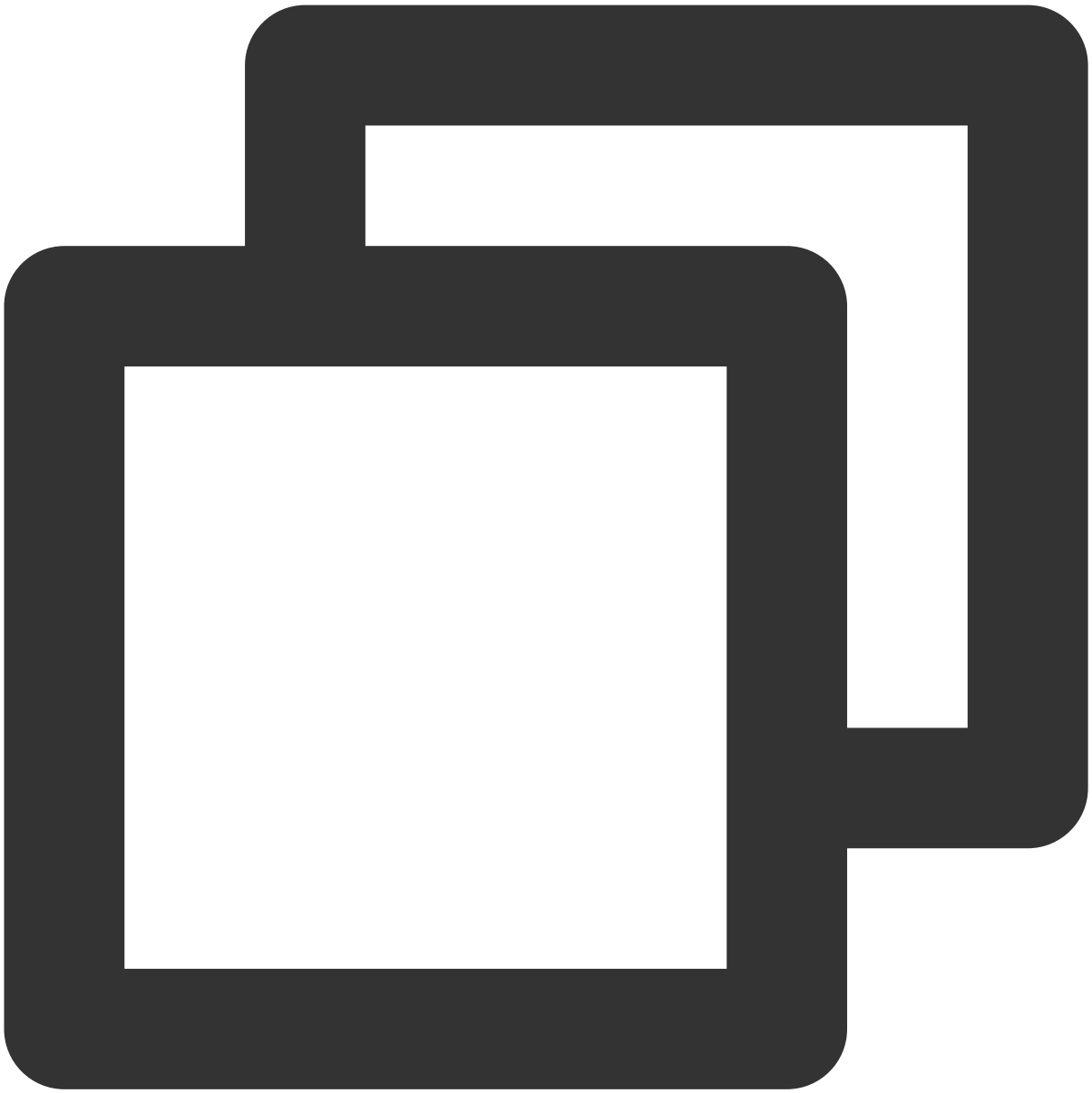




```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME,"Hello RocketMQ.".getBytes(StandardCharset
    msg.putUserProperty("key1","value1");
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

步骤2：消费消息

对于消费消息，主要是订阅的时候，带上对应的 SQL 表达式，其他的和普通的没有区别。



```
pushConsumer.subscribe(TOPIC_NAME, MessageSelector.bySql("True"));

// 订阅topic 订阅单个key的sql，最常用
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND

//订阅多个属性
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND

// 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(
            // 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();
```

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 。

发送与接收广播消息

最近更新时间：2023-11-24 14:31:24

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现广播消息收发的操作过程。

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

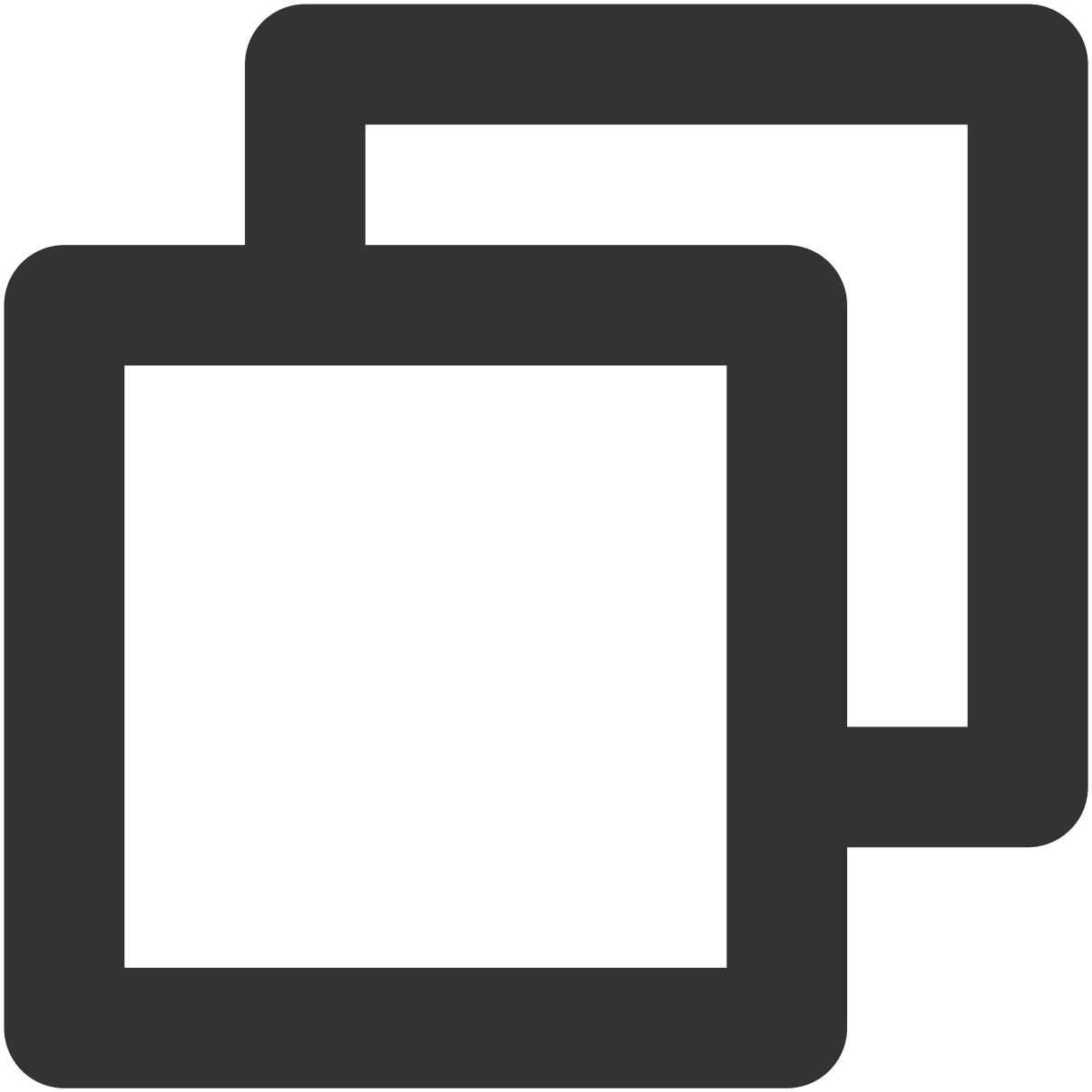
操作步骤

步骤1：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：

说明

依赖版本要求 ≥ 4.9.3, 当前建议为4.9.4

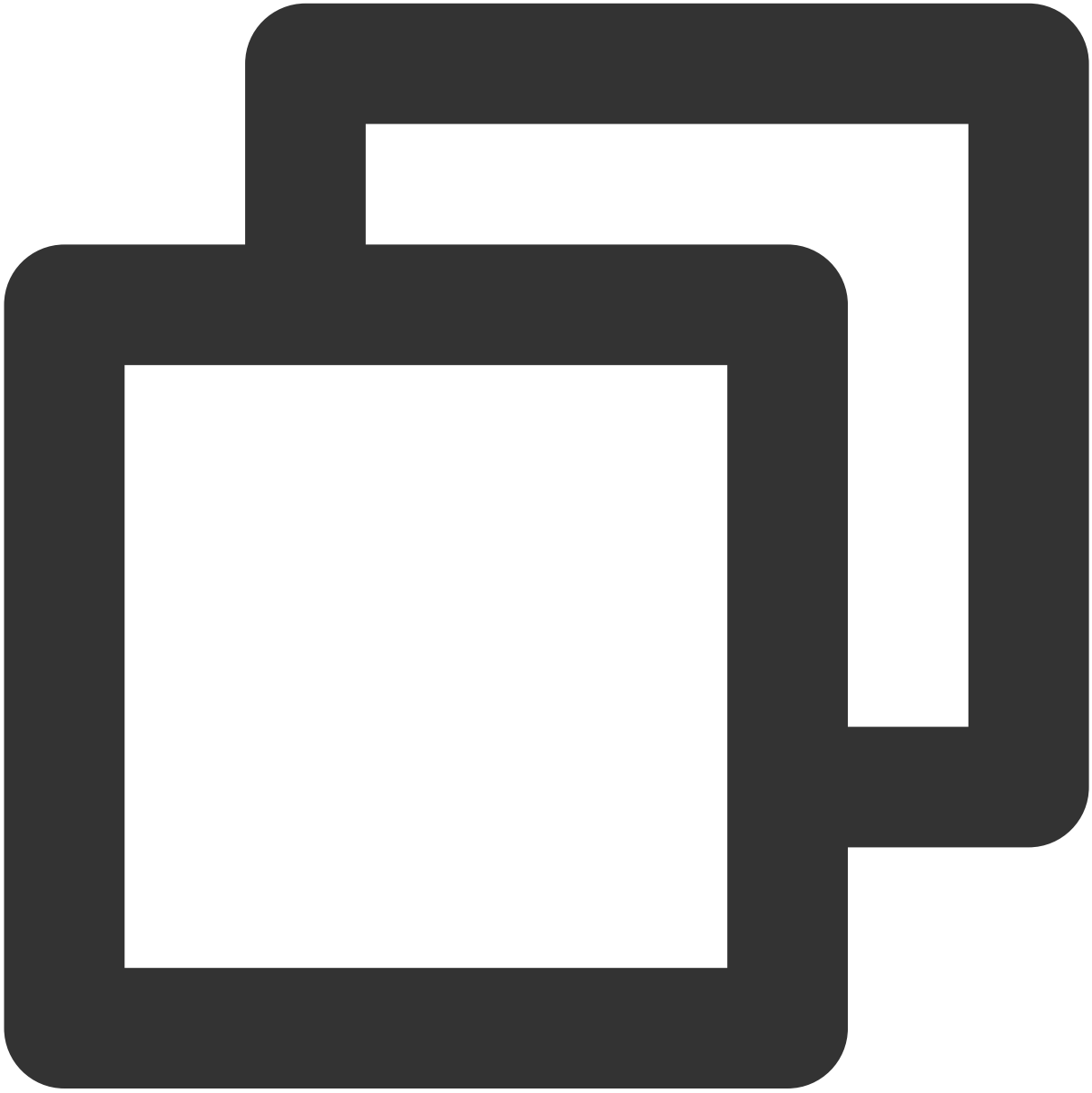


```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.4</version>
  </dependency>

  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-acl</artifactId>
      <version>4.9.4</version>
```

```
    </dependency>
```

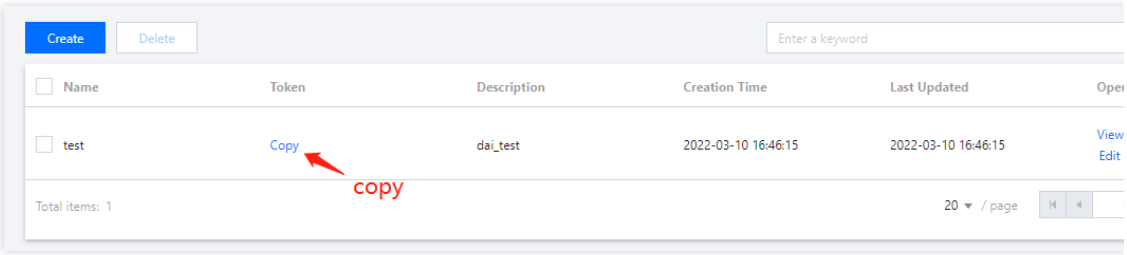
步骤2：生产消息

创建消息生产者



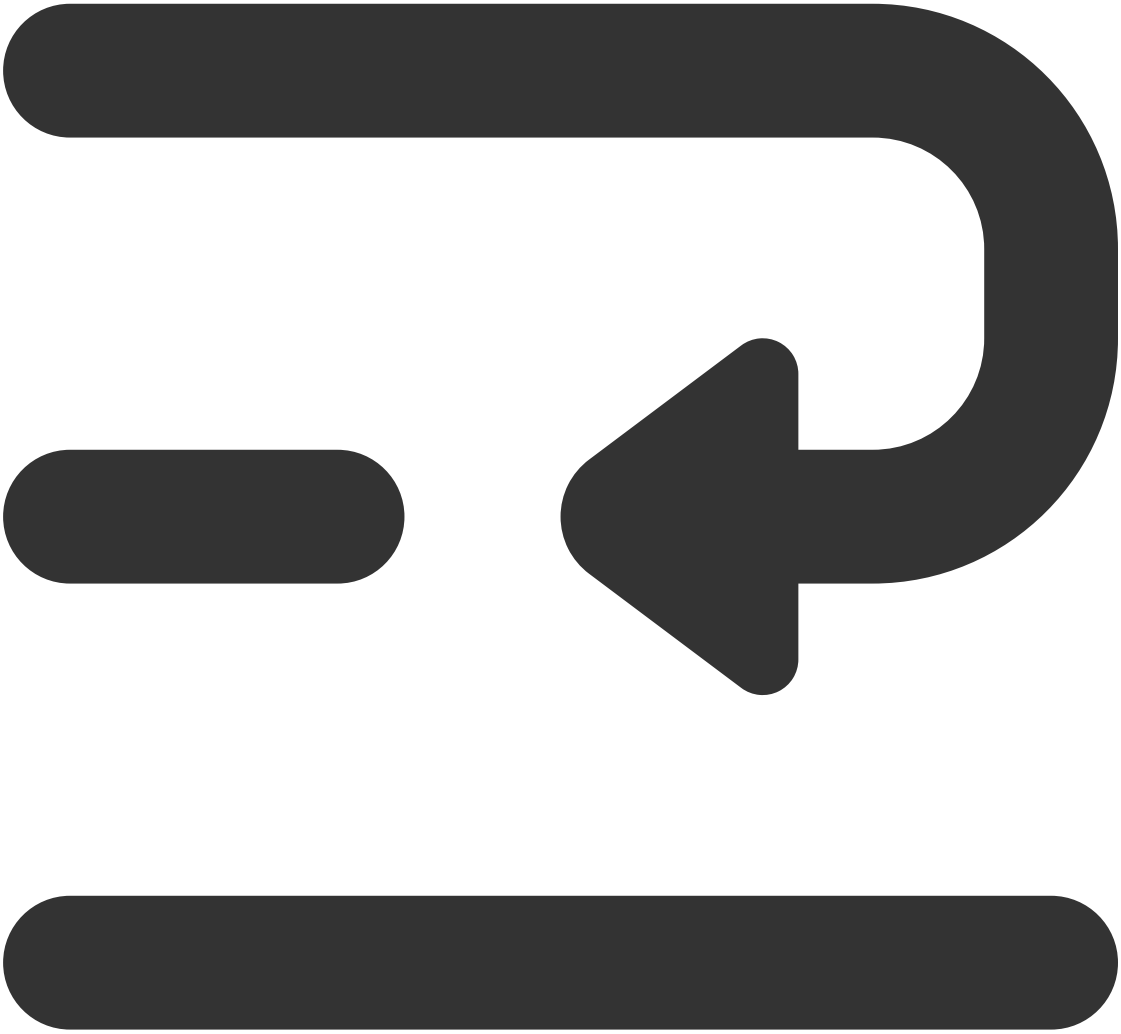
```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL权限
    );
```

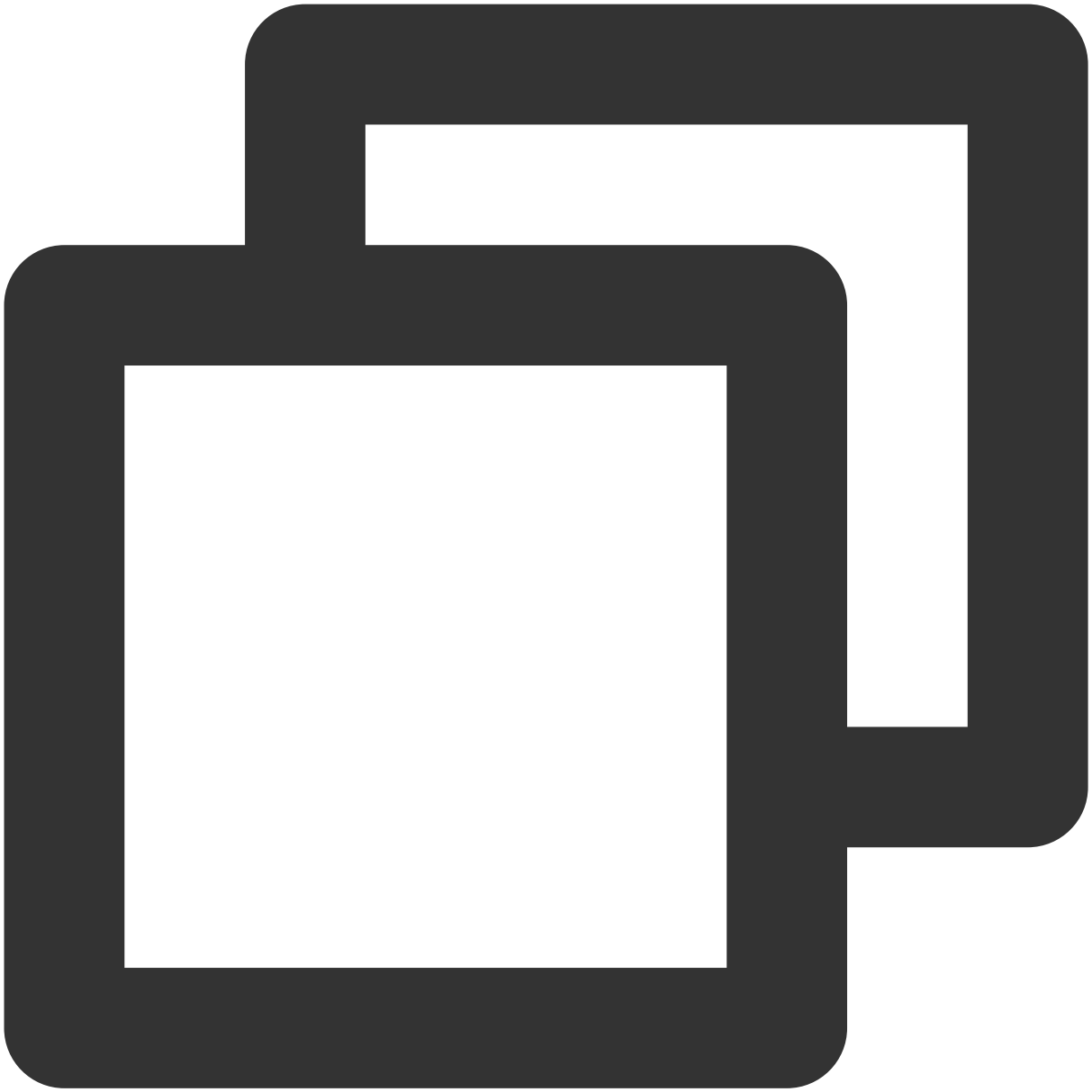
```
    // 设置NameServer的地址
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();
```

参数	说明
groupName	生产者组名称，建议使用对应的topic名字
nameserver	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在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

发送消息

和普通的消息没有区别，广播消息主要表达的是消费方的行为。



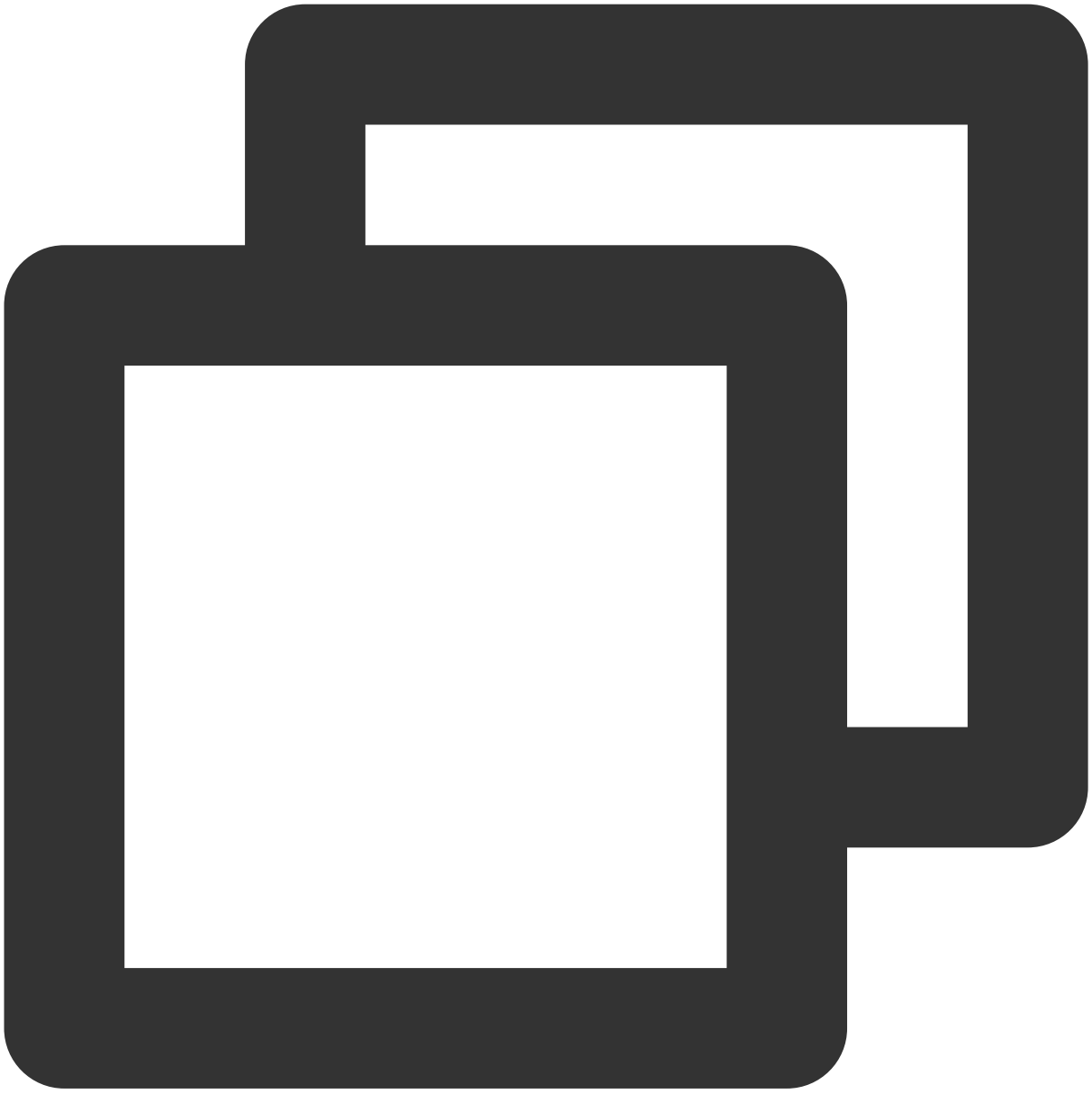


```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message " + i).getB
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

步骤3：消费消息

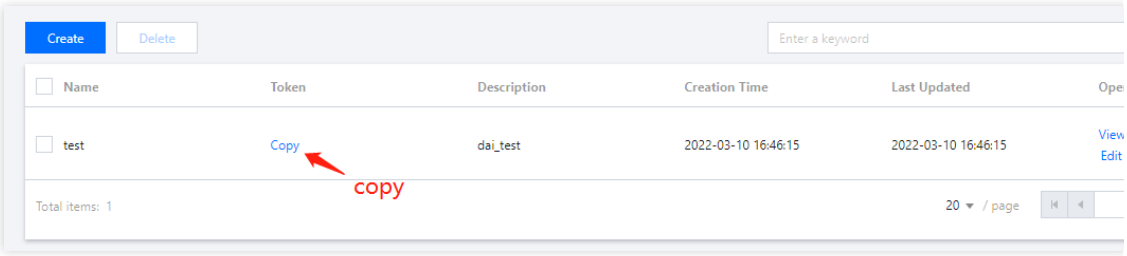
创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式



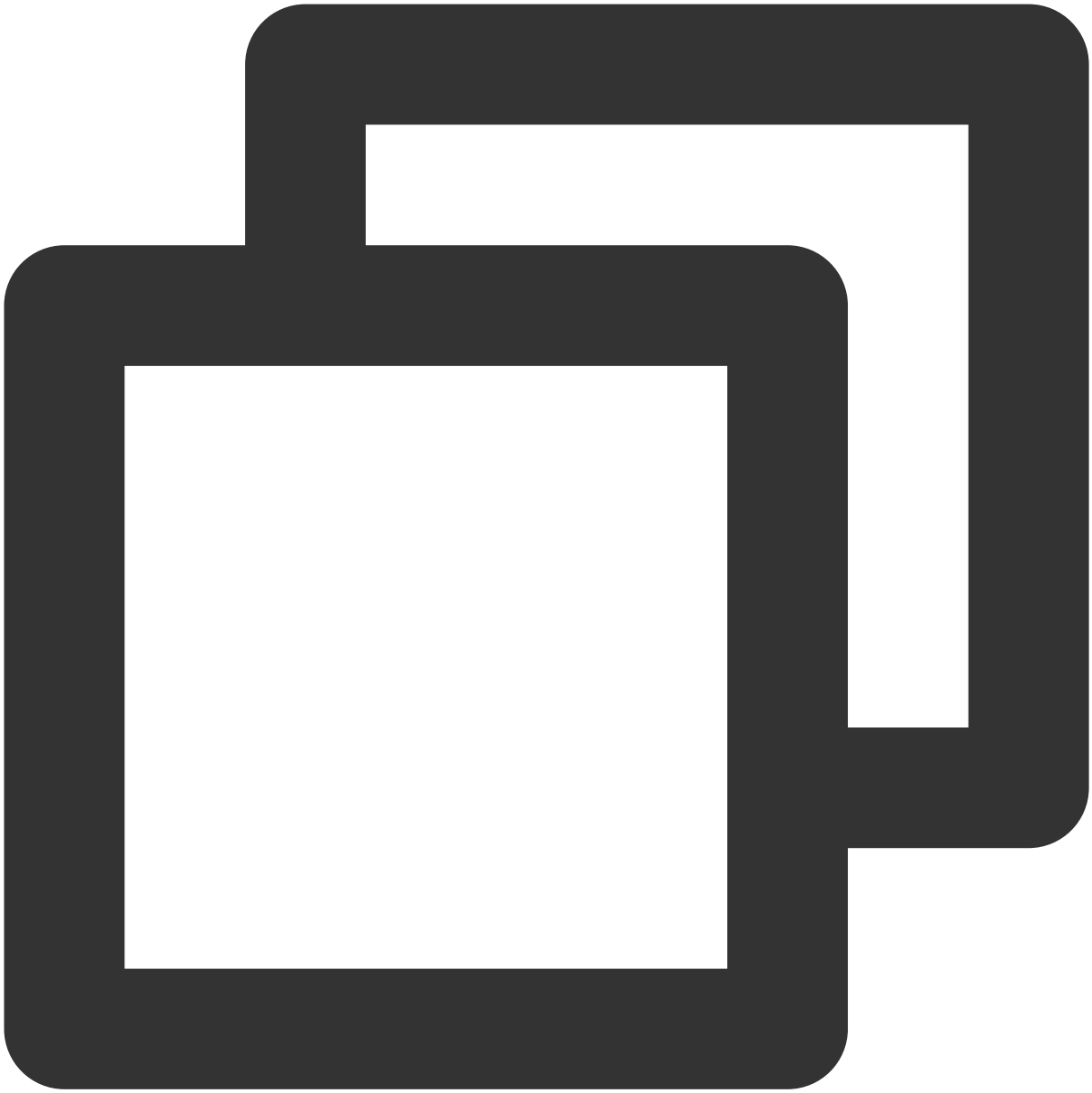
```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

参数	说明

groupName	生产者组名称，在控制台集群管理中Group 页签中复制。
nameserver	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在命名空间列表获取。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

订阅消息

主要增加消费模式的设置。

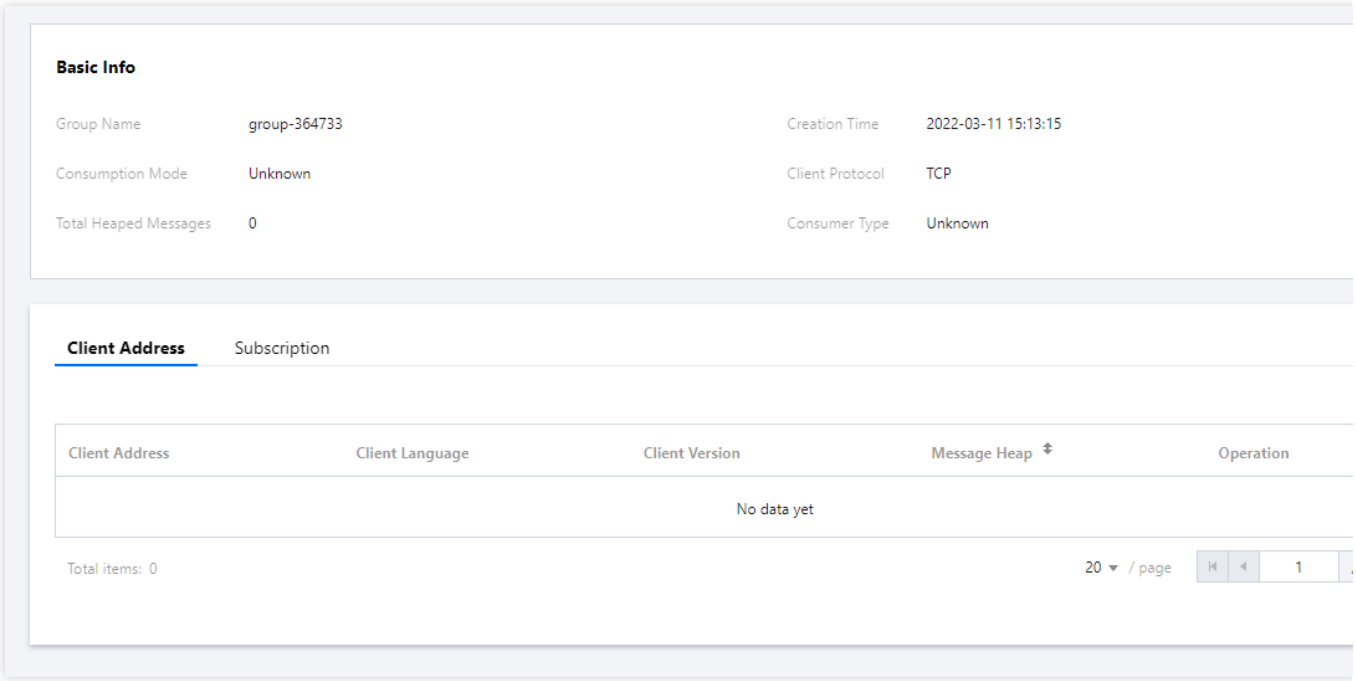


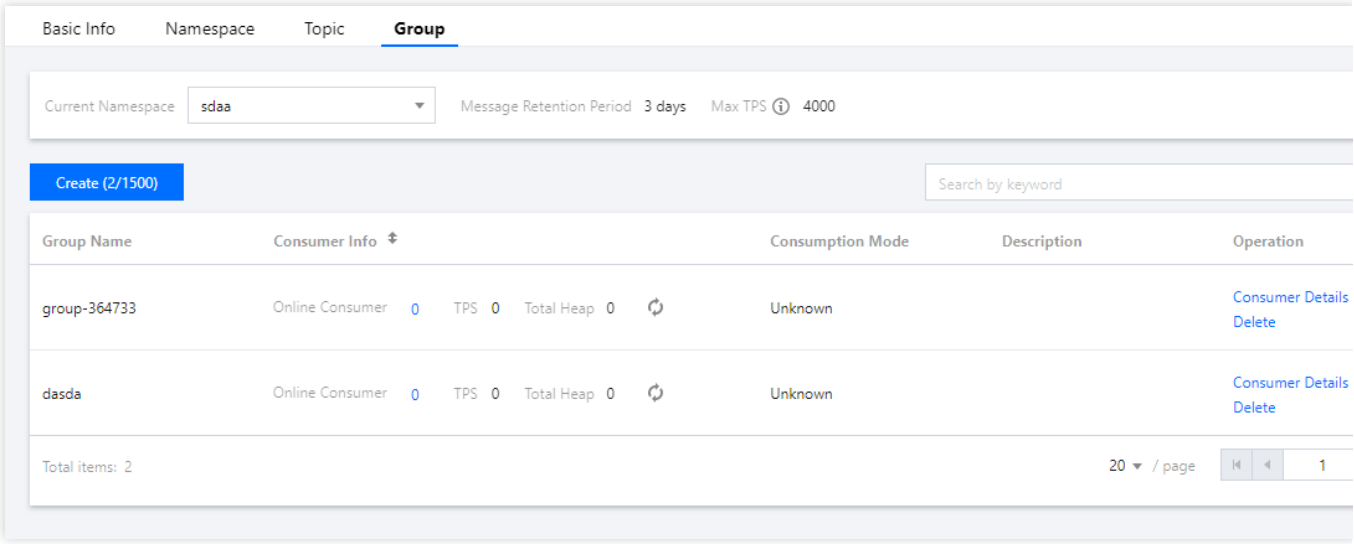
```
//设置广播消费模式
pushConsumer.setMessageModel(MessageModel.BROADCASTING);
// 订阅topic
pushConsumer.subscribe(topic_name, "*");
// 注册回调实现类来处理从broker拉取回来的消息
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
    // 消息处理逻辑
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().getNa
    // 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：查看消费详情

登录 TDMQ 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**查看消费者详情**，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 或 RocketMQ 官方文档 。

C++ SDK

最近更新时间：2023-11-24 14:32:37

操作场景

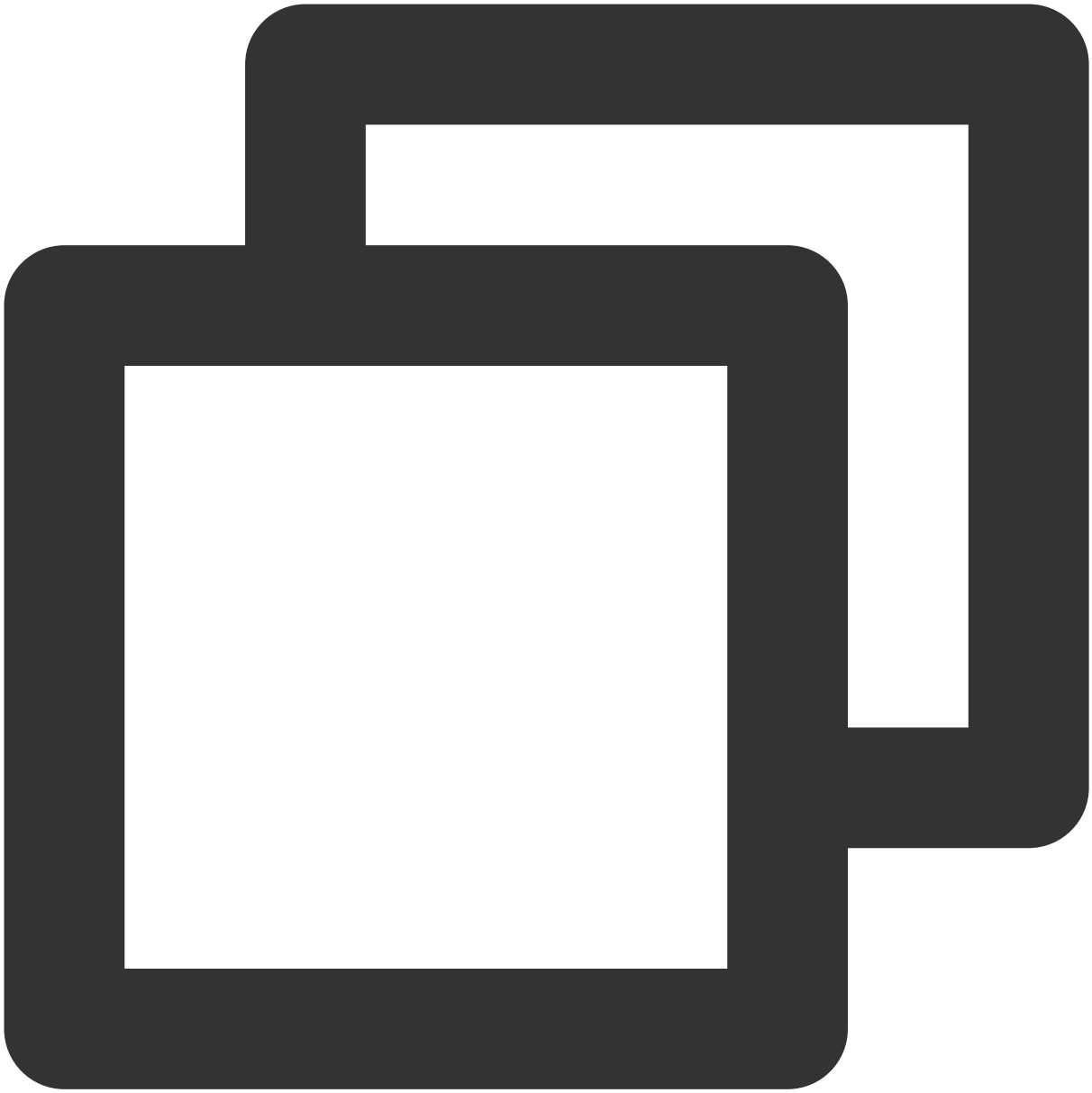
本文以调用 C++ SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

安装 GCC
下载 Demo

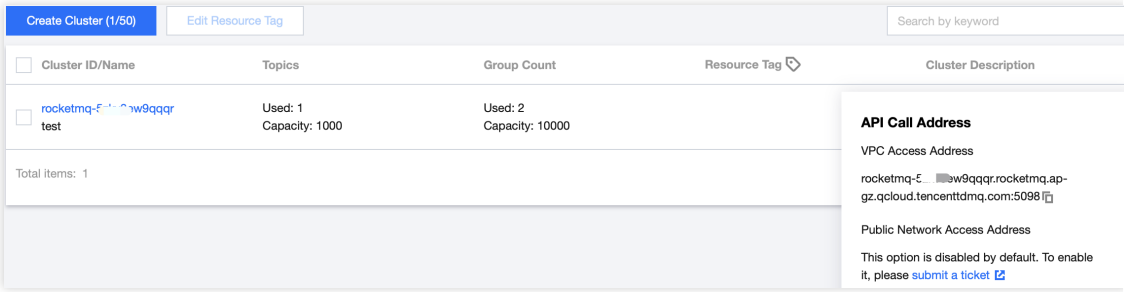
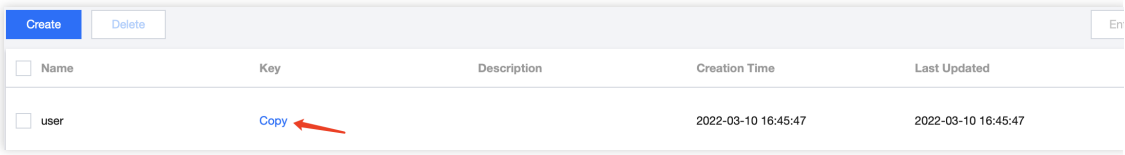
操作步骤

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。

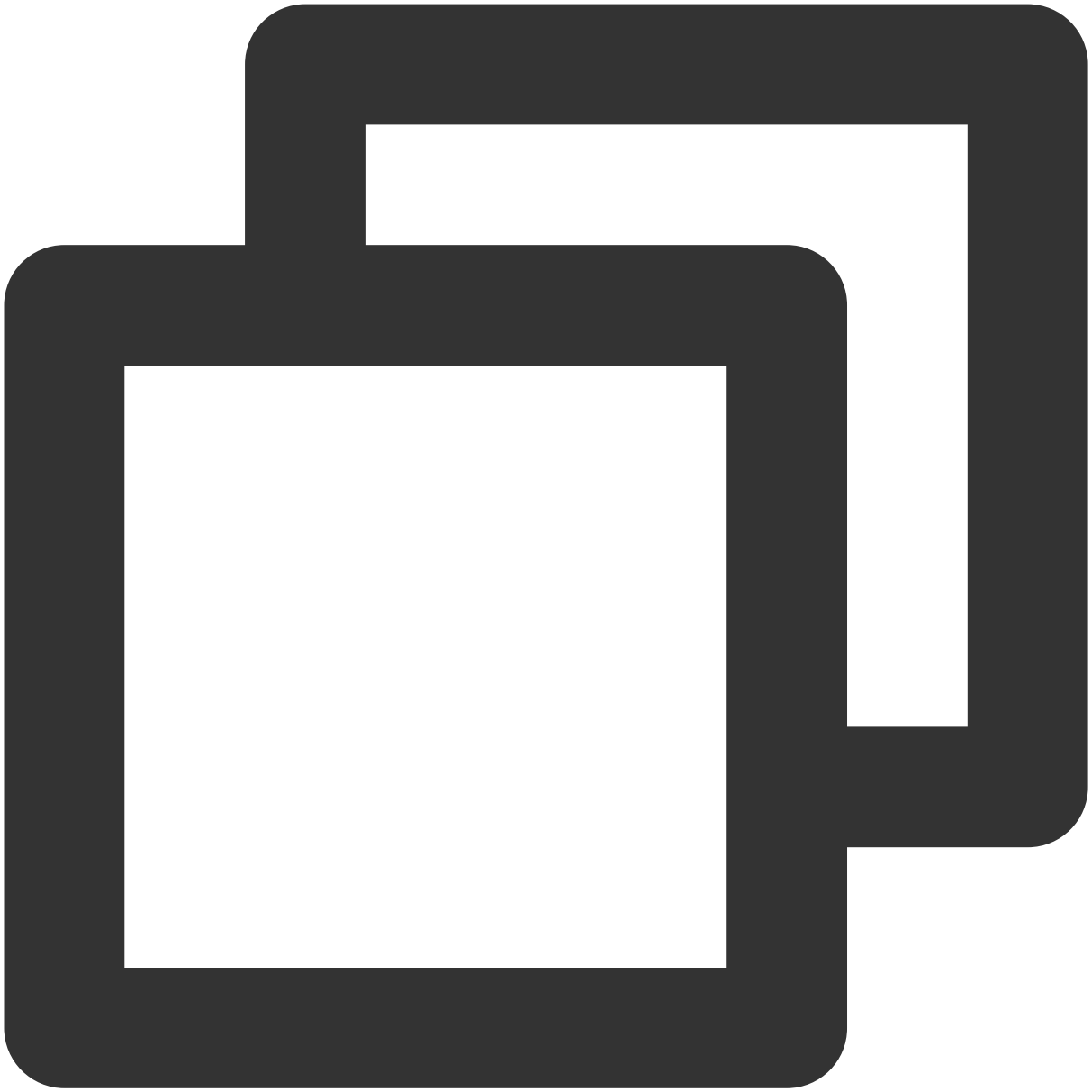


```
// 设置生产组名称
DefaultMQProducer producer(groupName);
// 设置服务接入地址
producer.setNamesrvAddr(nameserver);
// 设置用户权限
producer.setSessionCredentials(
    accessKey,  // 角色密钥
    secretKey, // 角色名称
    "");
// 设置命名空间(命名空间全称)
producer.setNameSpace(namespace);
```

```
// 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空间列表获取。 
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 
namespace	命名空间的名称，在控制台命名空间页面复制。

3. 发送消息。

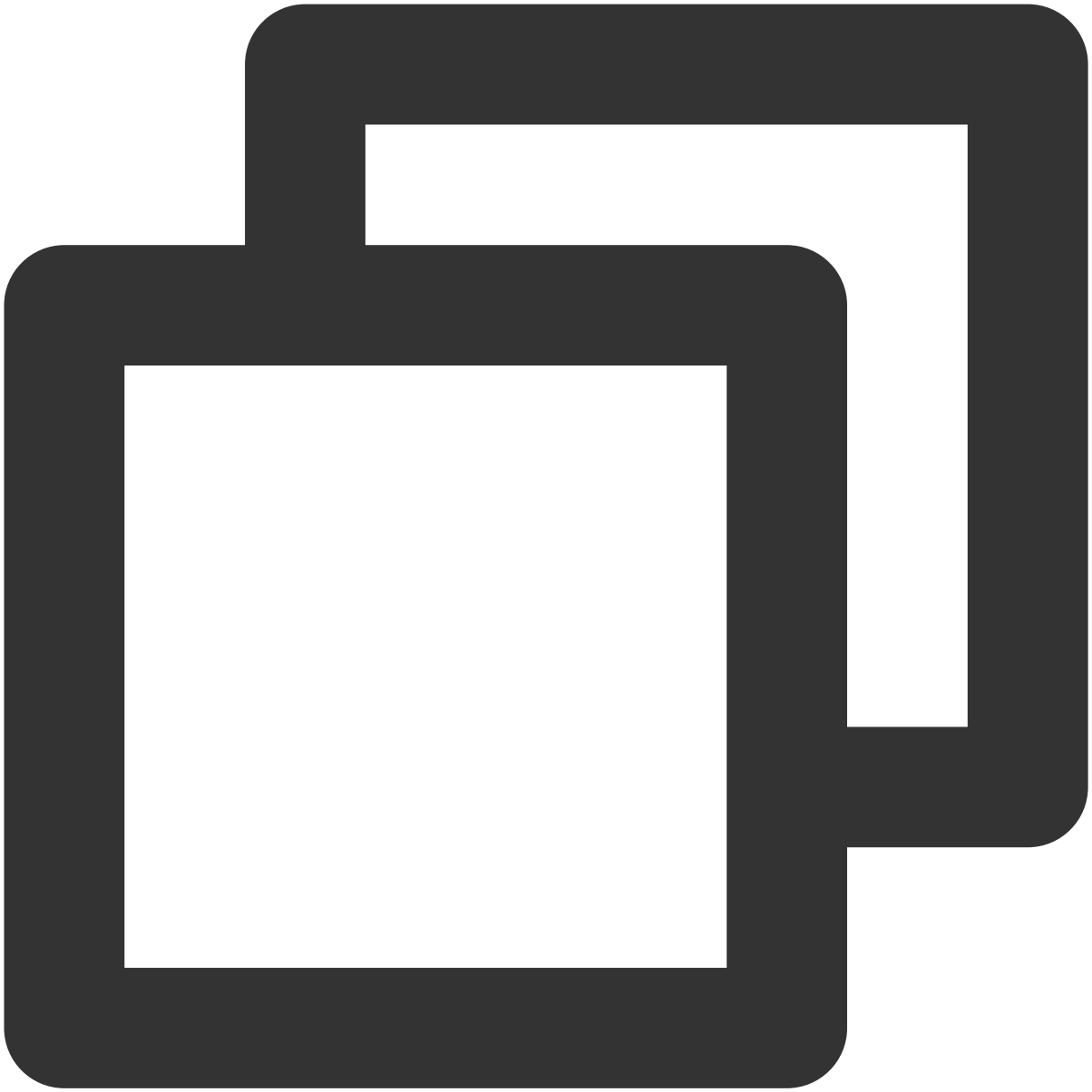


```
// 初始化消息内容
    MQMessage msg(
        topicName,  // topic名称
        TAGS,       // 消息tag
        KEYS,       // 消息业务key
        "Hello cpp client, this is a message."  // 消息内容
    );

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

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

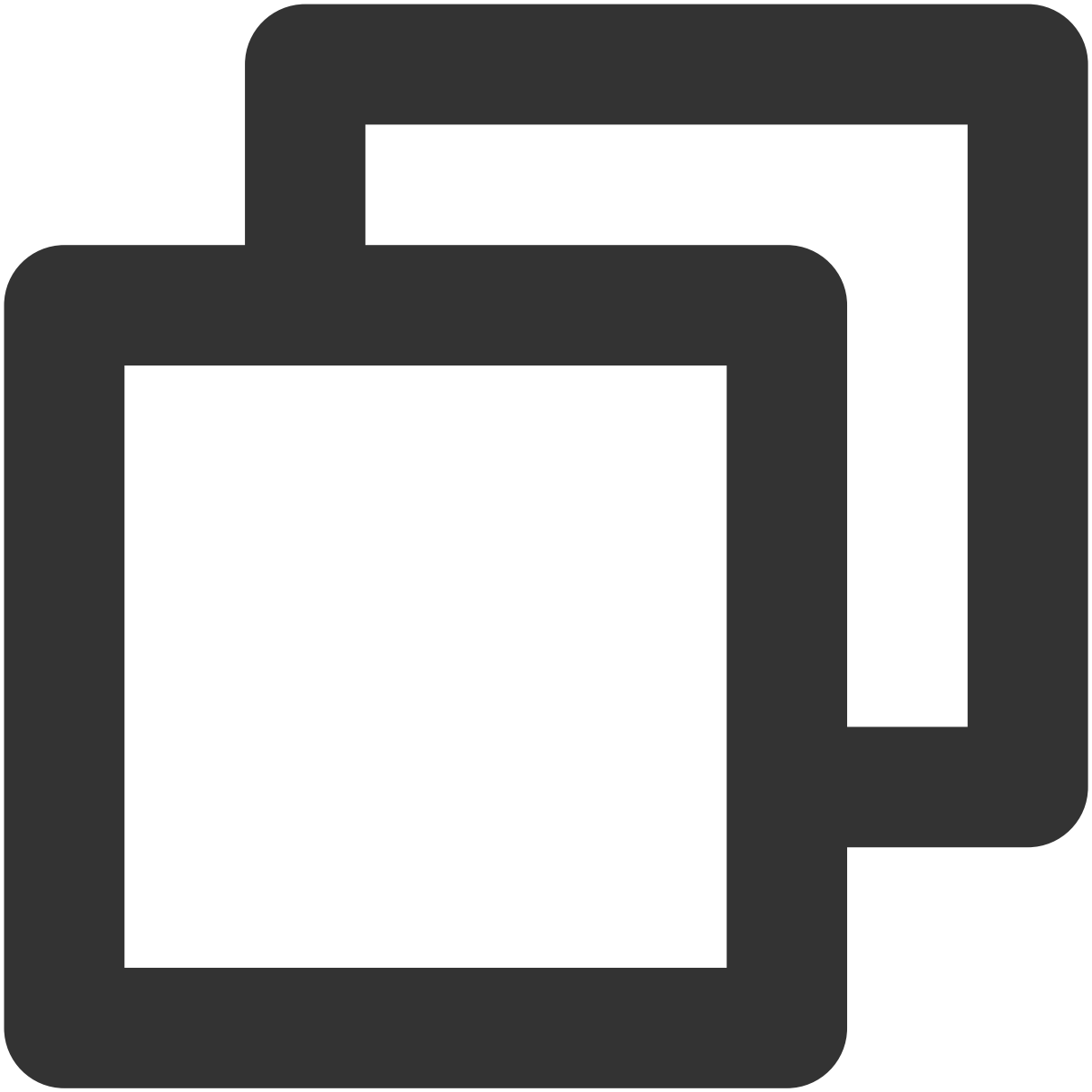
参数	说明
topicName	Topic 的名称，在控制台 topic 页面复制。
TAGS	用来设置消息的 TAG。
KEYS	设置消息业务 key。

4. 资源释放。



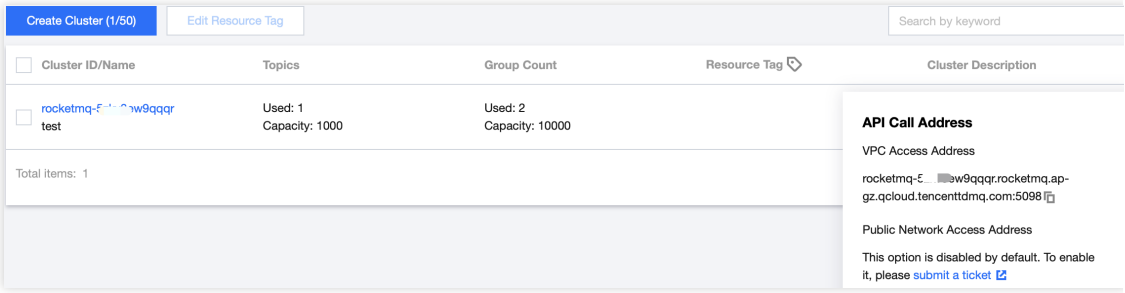
```
// 释放资源
   producer.shutdown();
```

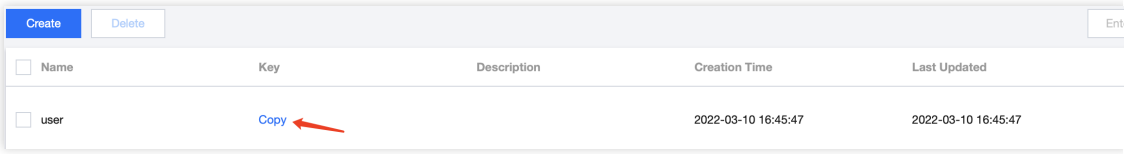
5. 初始化消费者。



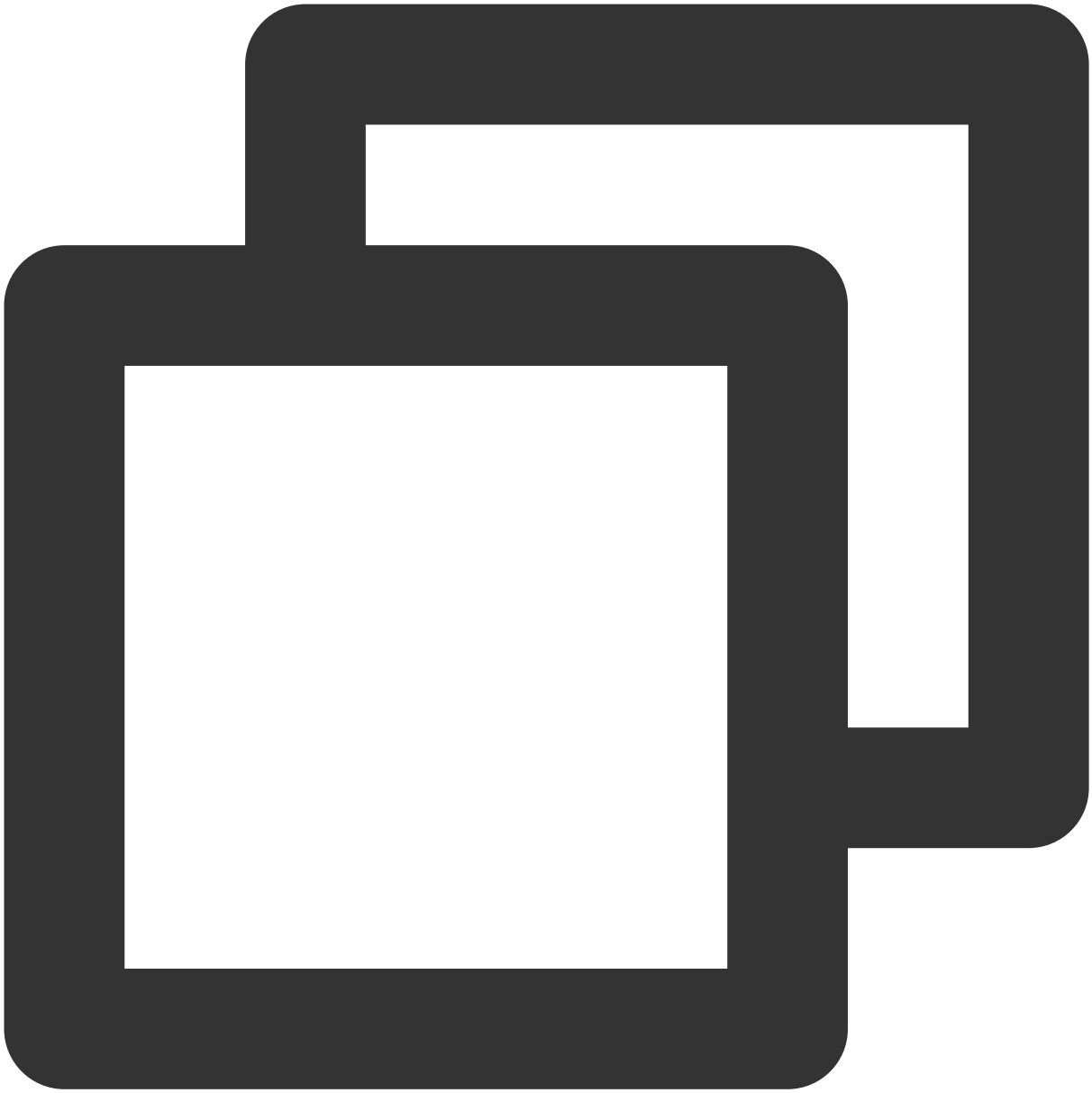
```
// 消息监听
   class ExampleMessageListener : public MessageListenerConcurrently {
   public:
       ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
           for (auto item = msgs.begin(); item != msgs.end(); item++) {
               // 业务
               std::cout << "Received Message Topic:" << item->getTopic() << ", Msg
                         << item->getTags() << ", KEYS:" << item->getKeys() << ", B
           }
           // 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。新版共享集群与专享集群命名接入点获取。 
secretKey	角色名称，在 角色管理 页面复制。

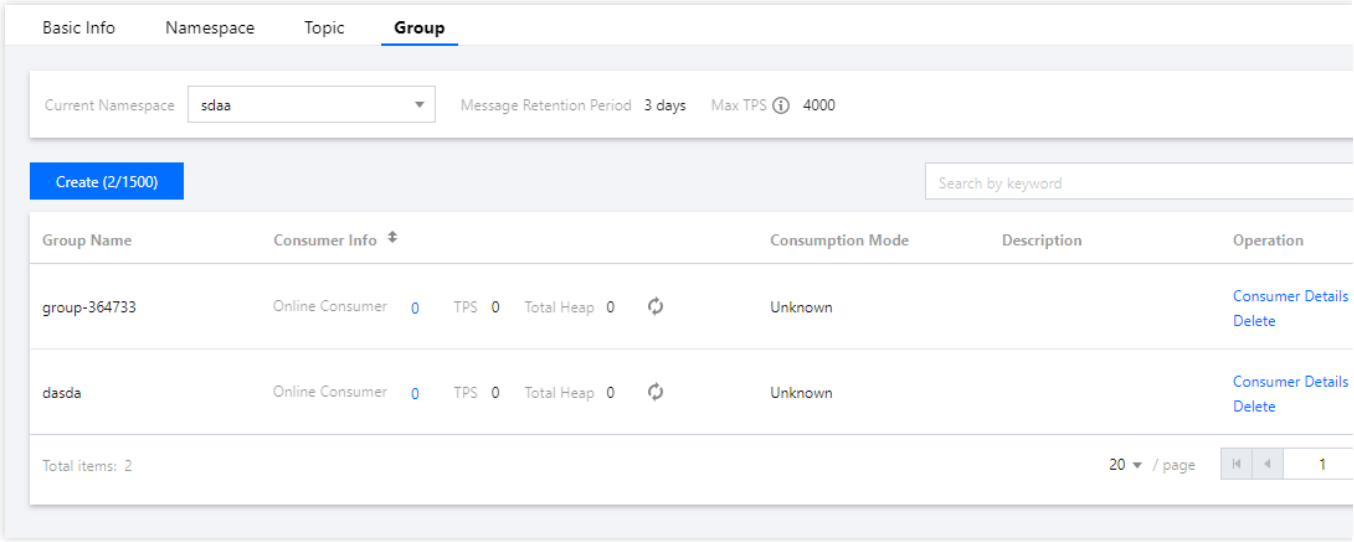
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 
namespace	命名空间的名称，在控制台命名空间页面复制。
topicName	Topic 的名称，在控制台 topic 页面复制。
TAGS	用来设置订阅消息的 TAG。

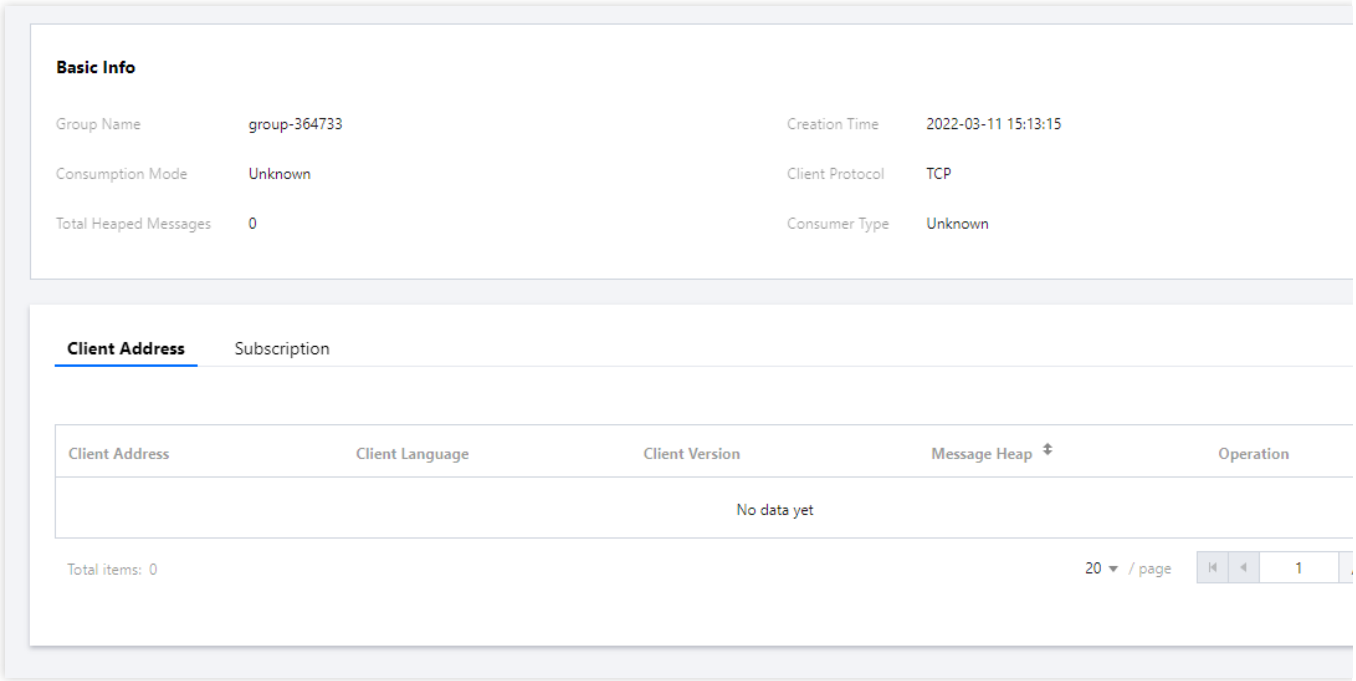
6. 资源释放。



```
// 资源释放
consumer->shutdown();
```

7. 查看消费详情。登录 TDMQ 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**查看消费者详情**，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 Demo 或 RocketMQ-Client-CPP 示例 。

Go SDK

最近更新时间：2023-11-24 14:33:34

操作场景

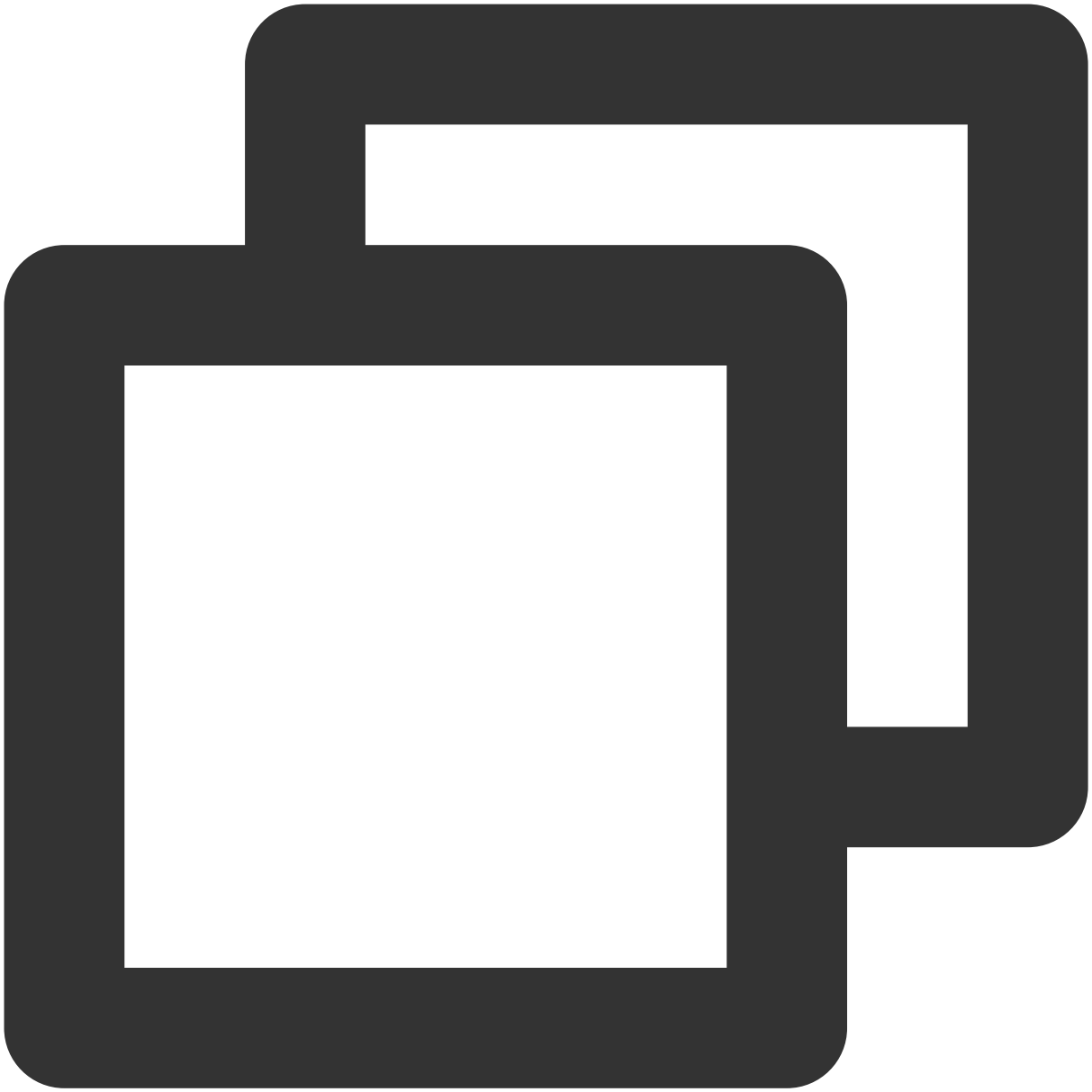
本文以调用 Go SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

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。



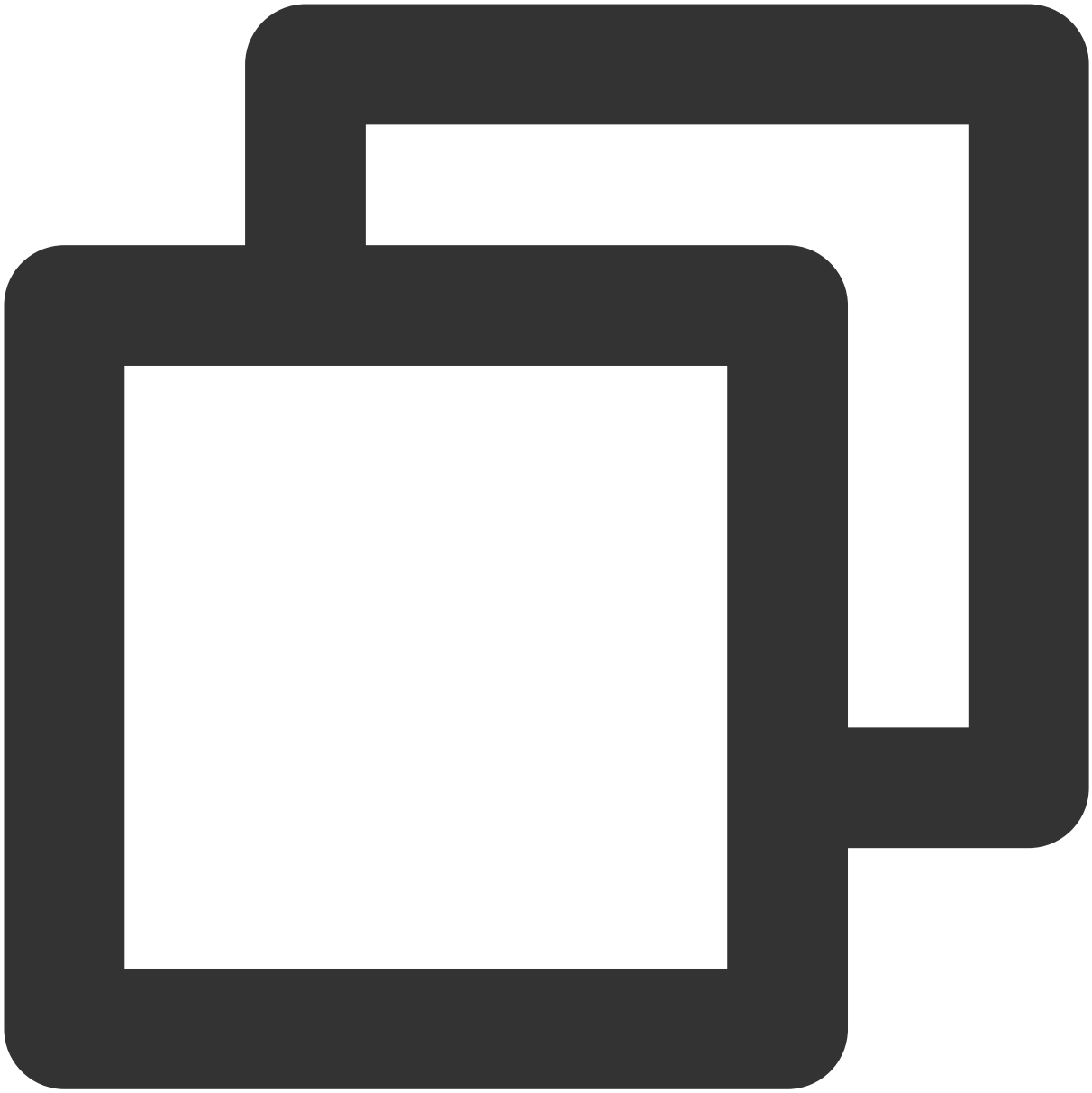
```
go get github.com/apache/rocketmq-client-go/v2
```

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。

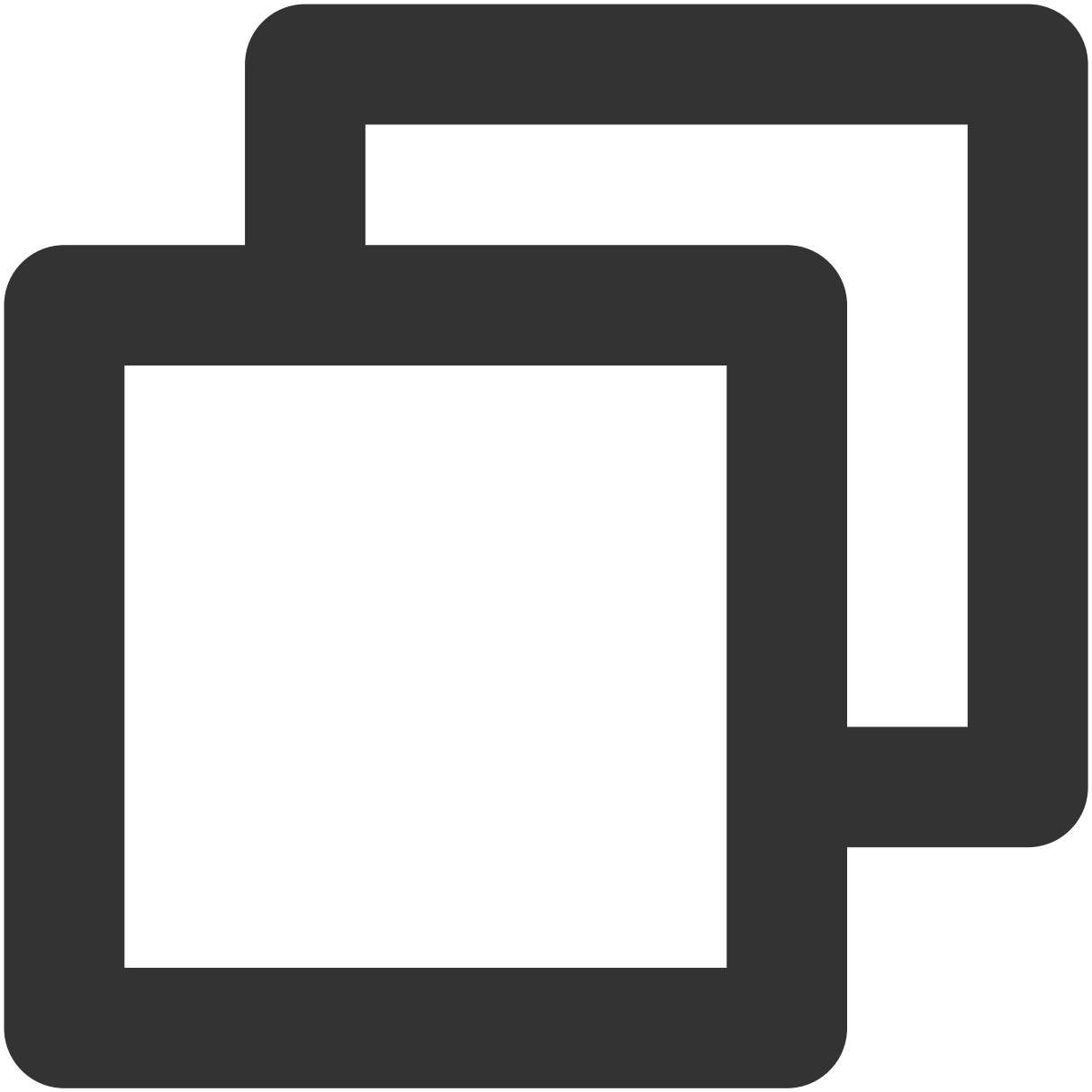
普通消息

延时消息



```
// 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:
    // 授权角色名
    var secretKey = "admin"
    // 授权角色密钥
    var accessKey = "eyJrZXlJZC...."
    // 命名空间全称
    var nameSpace = "MQ_INST_rocketmqem4xxxx"
    // 生产者组名称
    var groupName = "group1"
    // 创建消息生产者
```

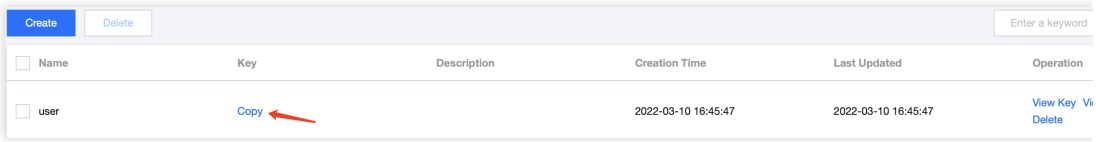
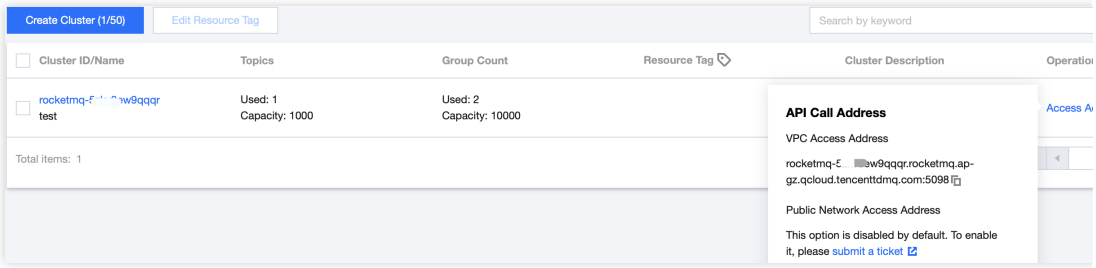
```
    p, _ := rocketmq.NewProducer(
        // 设置服务地址
        producer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddr
        // 设置acl权限
        producer.WithCredentials(primitive.Credentials{
            SecretKey: secretKey,
            AccessKey: accessKey,
        }),
        // 设置生产组
        producer.WithGroupName(groupName),
        // 设置命名空间名称
        producer.WithNamespace(nameSpace),
        // 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)
    }
```



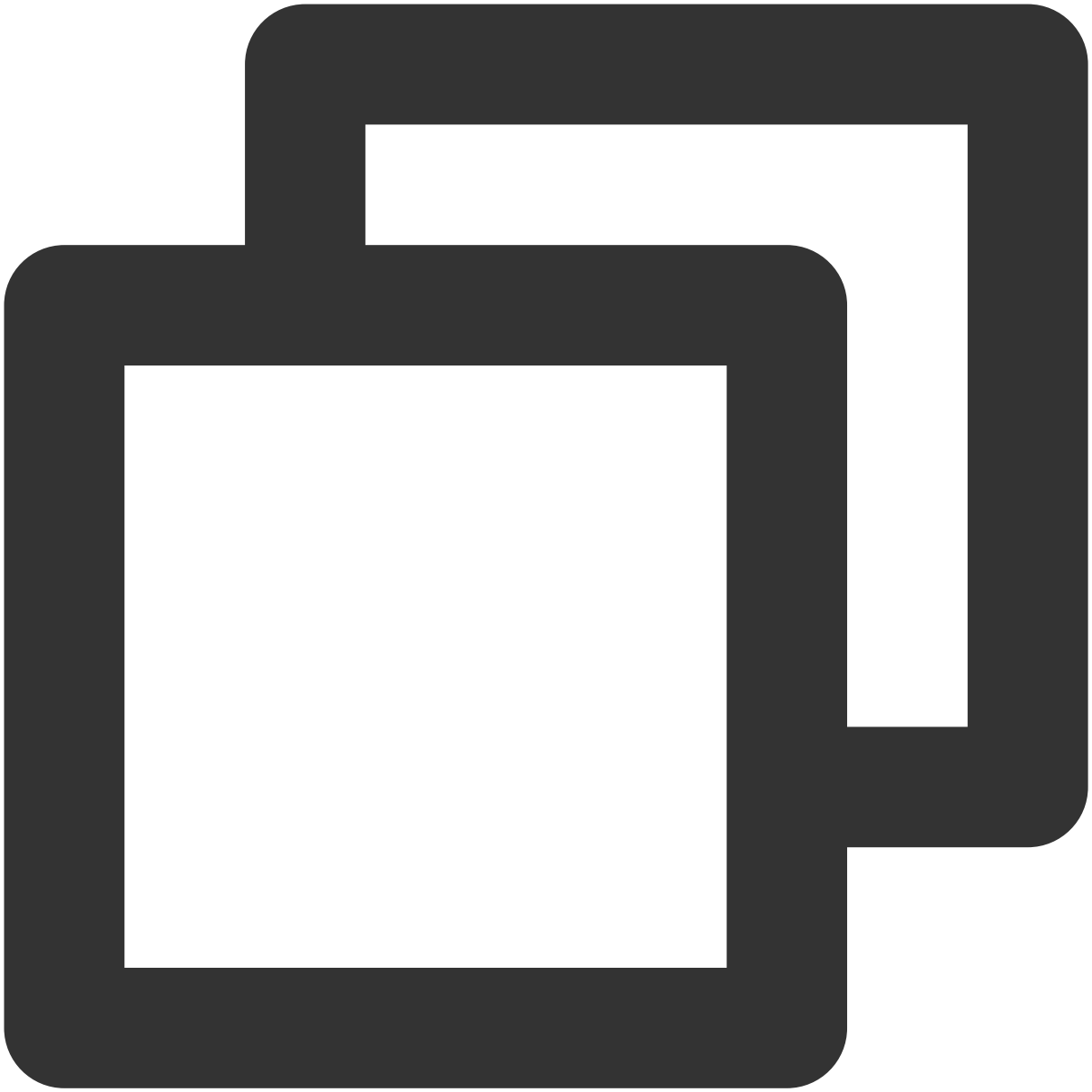
```
// topic名称
        var topicName = "topic1"
        // 生产者组名称
        var groupName = "group1"
        // 创建消息生产者
        p, _ := rocketmq.NewProducer(
                // 设置服务地址
                producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"
                // 设置acl权限
                producer.WithCredentials(primitive.Credentials{
```

```
                        SecretKey: "admin",
                        AccessKey: "eyJrZXlJZC......",
                },),
                // 设置生产组
                producer.WithGroupName(groupName),
                // 设置命名空间名称
                producer.WithNamespace("rocketmq-xxx|namespace_go"),
                // 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 Cl
                // 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、
                // 1    2    3     4     5    6   7    8   9   10   11   12  13   1
                //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
                delayMills := int64(10 * 1000)
                msg.WithProperty("__STARTDELIVERTIME", strconv.FormatInt(time.Now()
                // 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())
        }
```

参数	说明

secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 
nameSpace	命名空间的名称，在控制台命名空间页面复制。
serverAddress	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 入点地址在命名空间列表获取。 (注意：需要在接入地址前面加上 http:// 或 https:// 否则无法解析 
groupName	生产者组名称，在控制台 Group 页面复制。

3. 发送消息同上（以同步发送方式为例）。

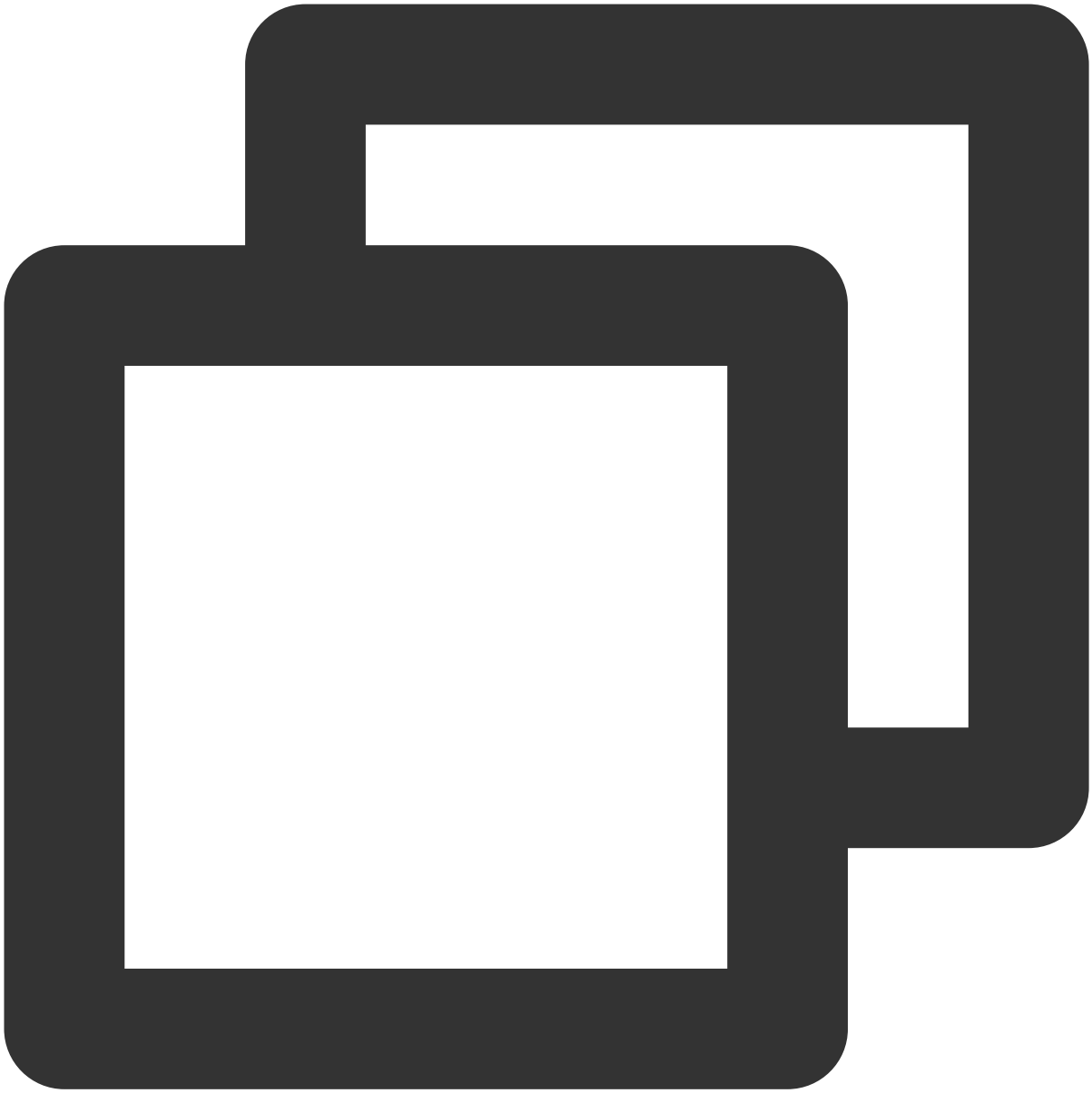


```
// topic名称
    var topicName = "topic1"
    // 构造消息内容
    msg := &primitive.Message{
        Topic: topicName, // 设置topic名称
        Body:  []byte("Hello RocketMQ Go Client! This is a new message."),
    }
    // 设置tag
    msg.WithTag("TAG")
    // 设置key
    msg.WithKeys([]string{"yourKey"})
```

```
    // 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。

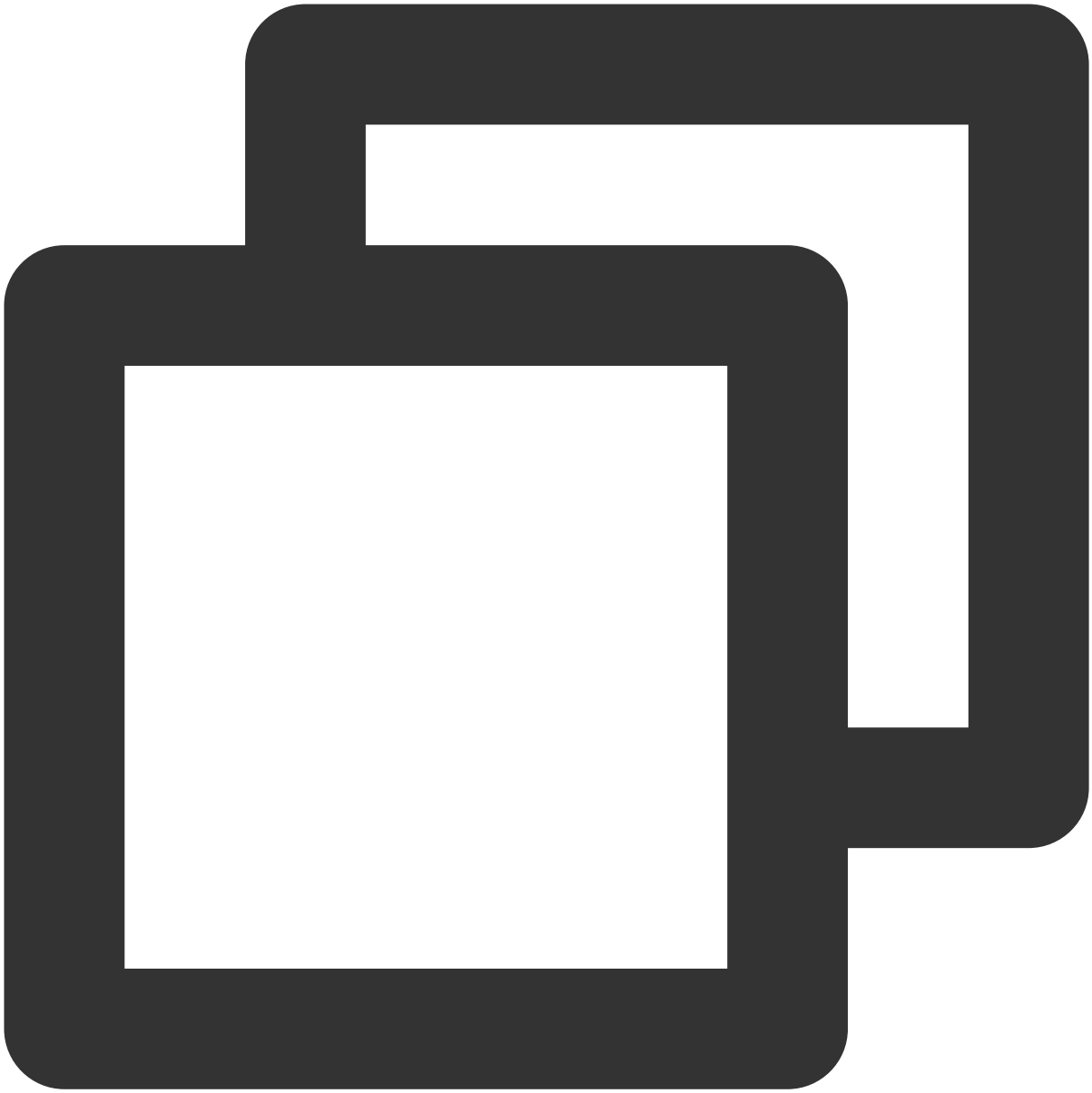


```
// 关闭生产者
    err = p.Shutdown()
    if err != nil {
        fmt.Printf("shutdown producer error: %s", err.Error())
    }
```

说明

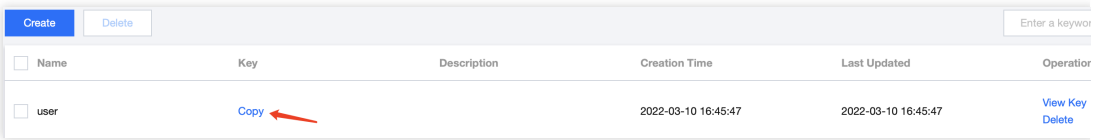
异步发送、单向发送等，可参见 demo 示例 或参见 rocketmq-client-go 示例。

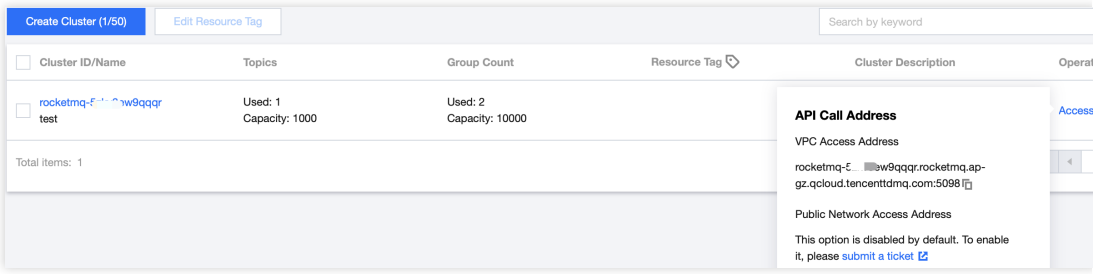
4. 创建消费者。



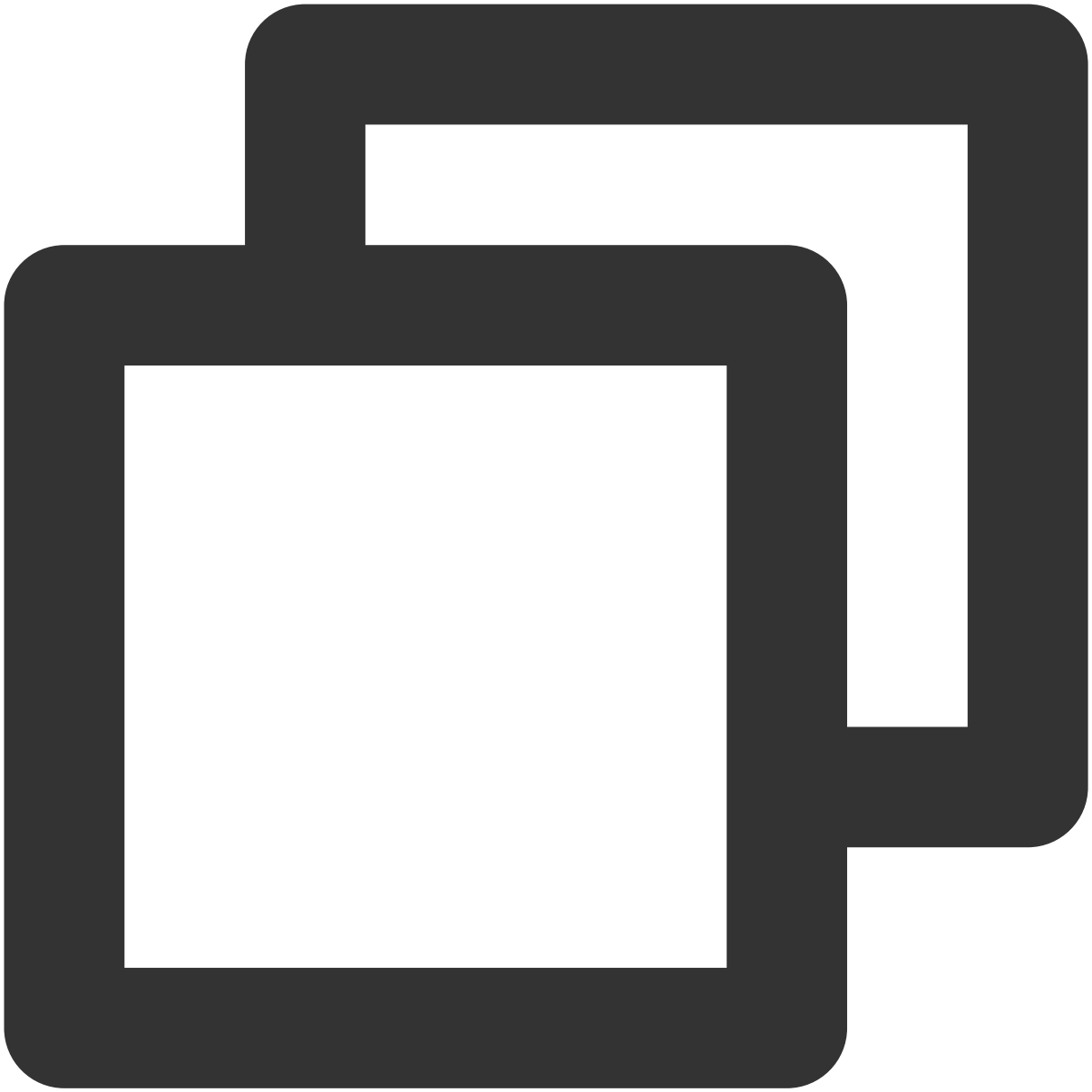
```
// 服务接入地址  （注意：需要在接入地址前面加上  http:// 或 https:// 否则无法解析）
    var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:
    // 授权角色名
    var secretKey = "admin"
    // 授权角色密钥
    var accessKey = "eyJrZXlJZC...."
    // 命名空间全称
    var nameSpace = "rocketmq-xxx|namespace_go"
    // 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{serverAddr
        // 设置acl权限
        consumer.WithCredentials(primitive.Credentials{
            SecretKey: secretKey,
            AccessKey: accessKey,
        }),
        // 设置命名空间名称
        consumer.WithNamespace(nameSpace),
        // 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
        consumer.WithInstance("xxxx"),
    )
    if err != nil {
        fmt.Println("init consumer2 error: " + err.Error())
        os.Exit(0)
    }
```

参数	说明
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 
nameSpace	命名空间全称在控制台集群管理中 Topic 页签中页面复制，格式是集群 ID +｜+命名空间。
serverAddress	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 入点地址在命名空间列表获取。 (注意：需要在接入地址前面加上 http:// 或 https:// 否则无法解析

	
groupName	生产者组名称，在控制台 Group 页面复制。

5. 消费消息。

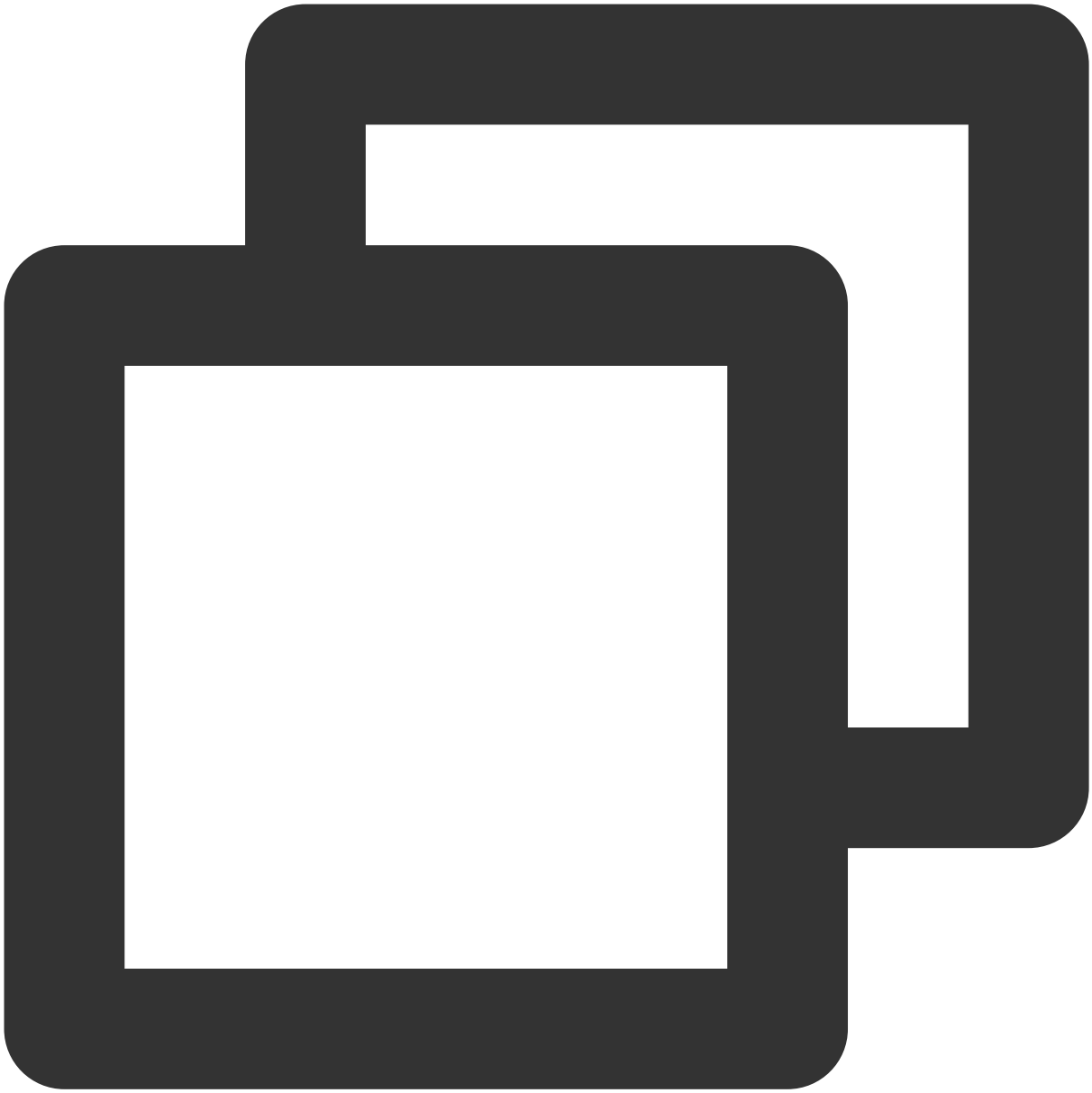


```
// topic名称
    var topicName = "topic1"
    // 设置订阅消息的tag
    selector := consumer.MessageSelector{
        Type:       consumer.TAG,
        Expression: "TagA || TagC",
    }
    // 设置重新消费的延迟级别，共支持18种延迟级别。下面是延迟级别与延迟时间的对应关系
    // 1   2   3    4    5   6   7   8   9   10  11  12  13  14   15   16   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
        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

        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。

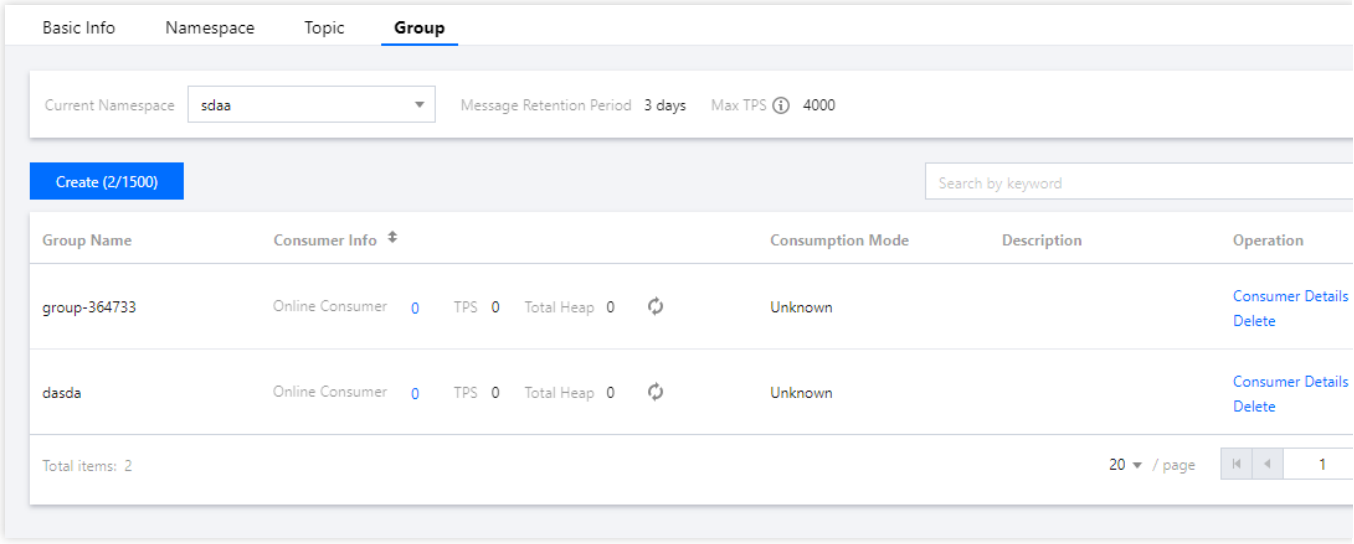
6. 消费消息 （消费者消费消息必须在订阅之后）。

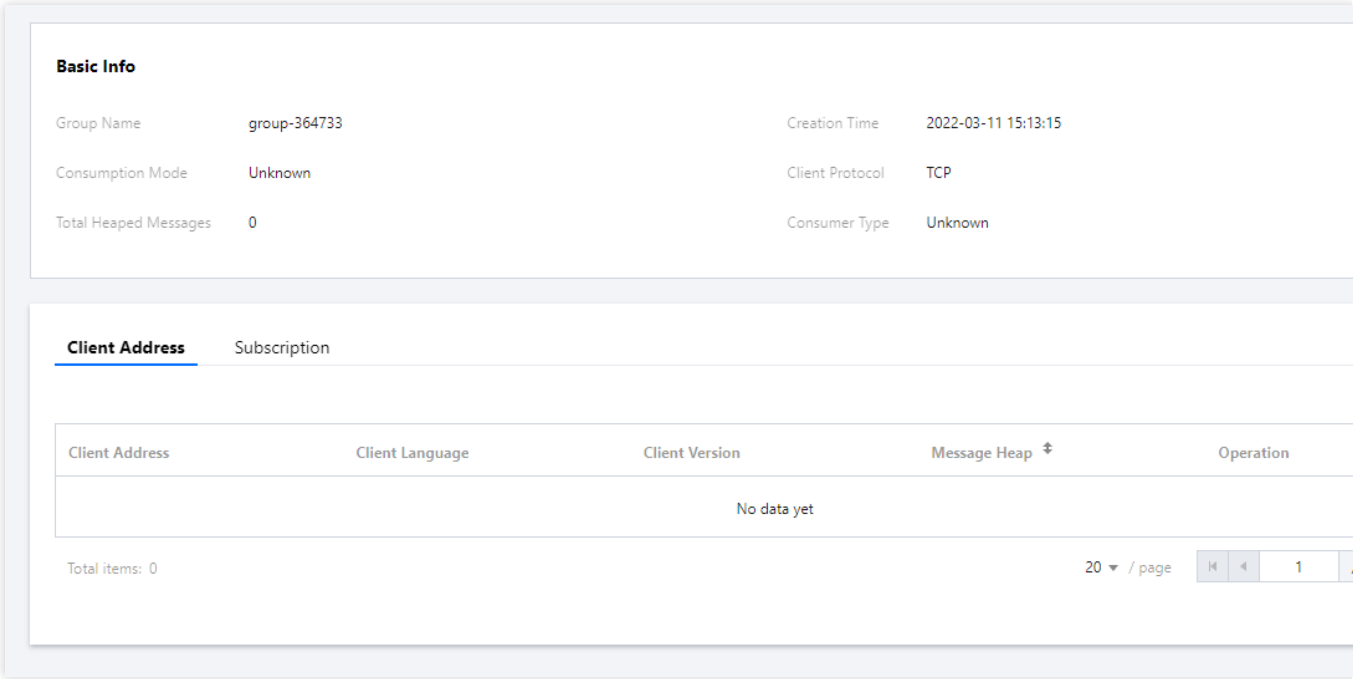


```
// 开始消费
    err = c.Start()
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(-1)
    }
    time.Sleep(time.Hour)
    // 资源释放
    err = c.Shutdown()
    if err != nil {
        fmt.Printf("shundown Consumer error: %s", err.Error())
```

```
    }
```

7. 查看消费详情。登录 TDMQ 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**查看消费者详情**，可查看消费者详情。





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。

Python SDK

最近更新时间：2023-11-24 14:33:56

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

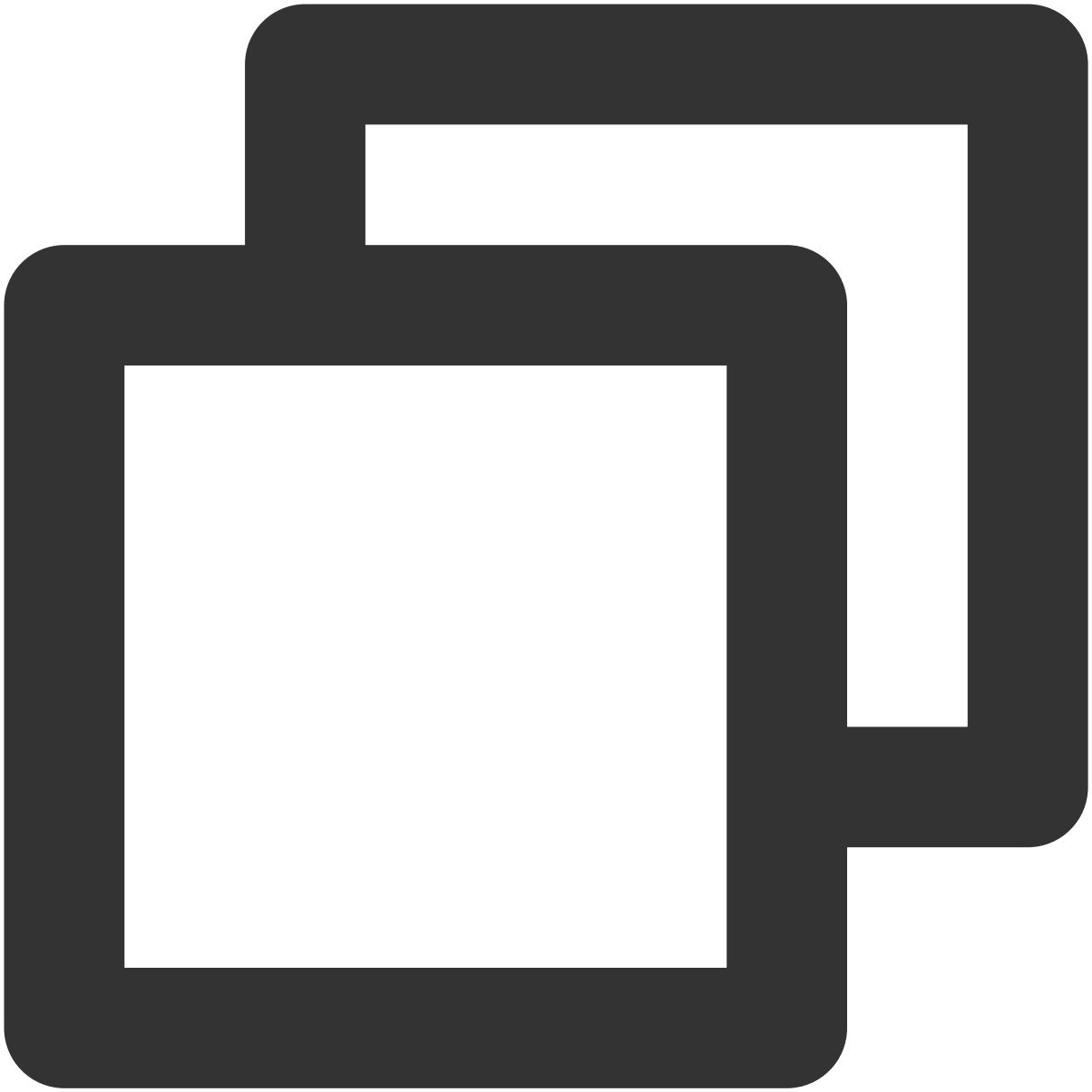
步骤1：准备环境

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。

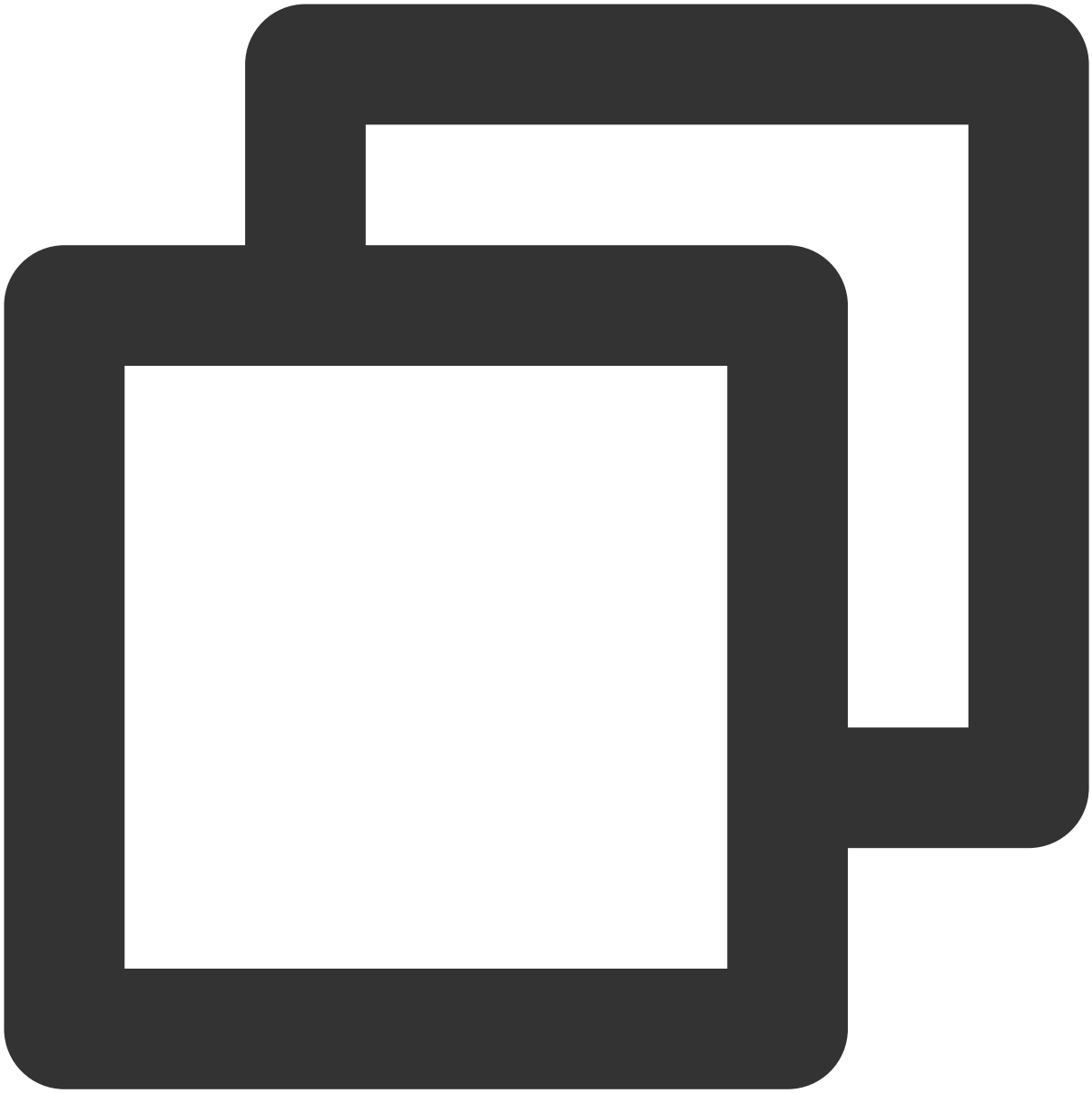
1. 安装 librocketmq （版本2.0.0及以上）， 安装教程参见 librocketmq 安装。
2. 执行如下命令安装 rocketmq-client-python。



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

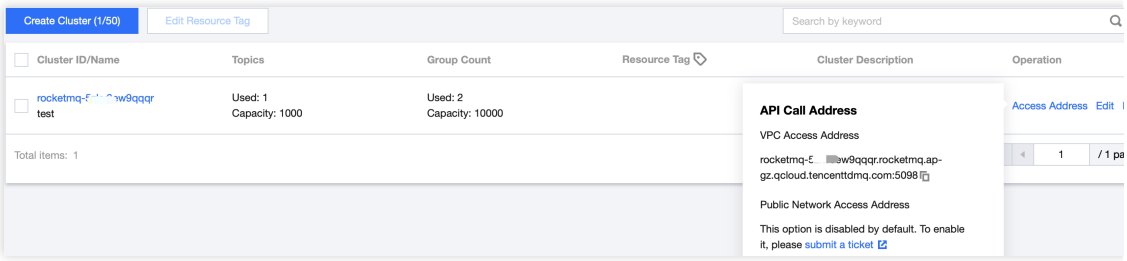
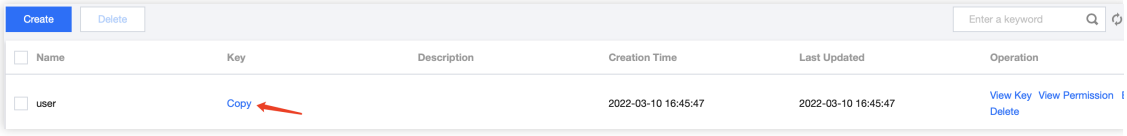
步骤2：生产消息

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，组名称使用全称，例：rocketmq-xxx|namespace_python%group
    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
msg.set_keys(TAGS)
# 设置tags
msg.set_tags(KEYS)
# 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()
```

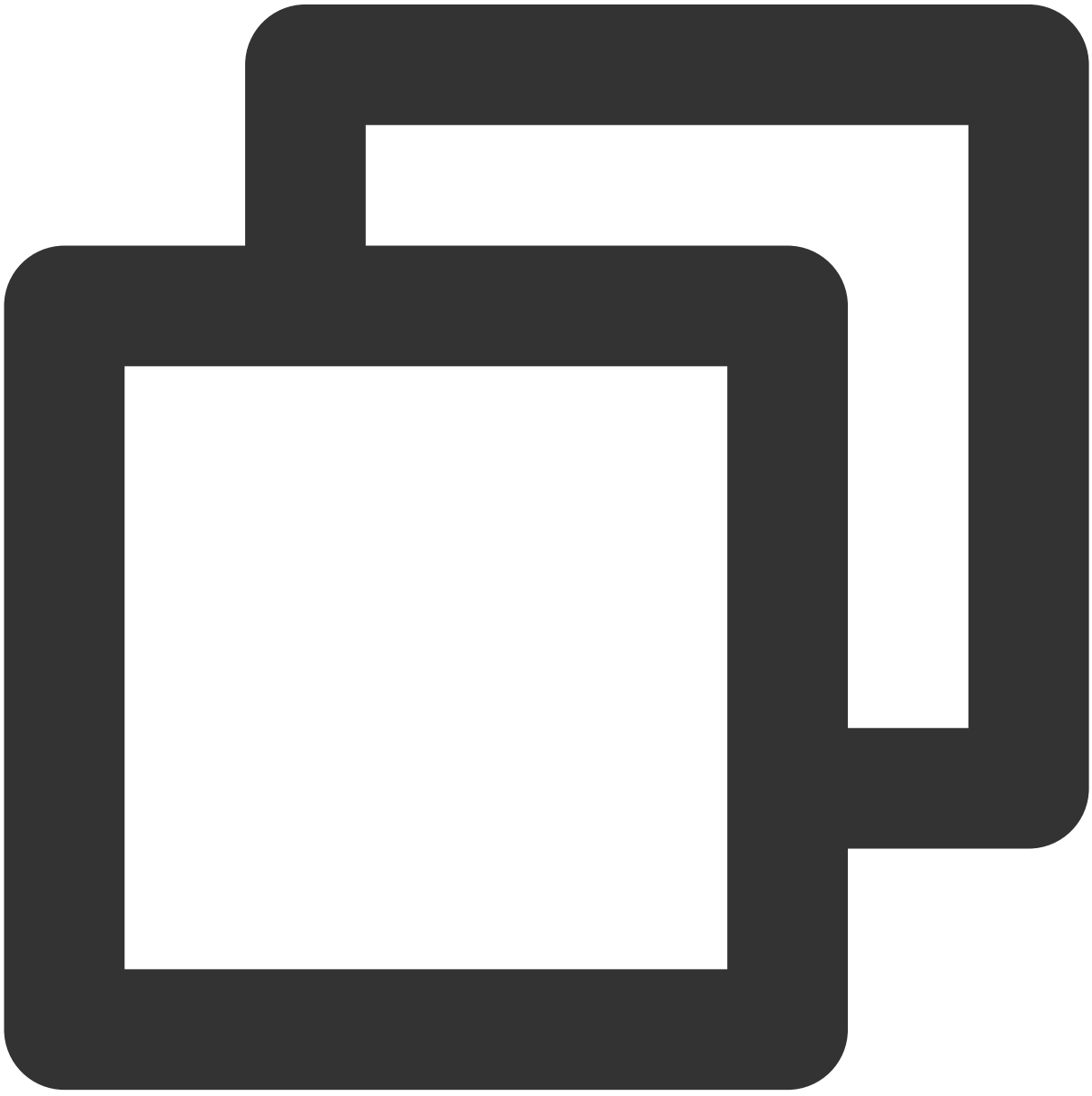
参数	说明
groupName	生产者组名称。在控制台集群管理中 Group tab 中获取。
nameserver	集群接入地址，在控制台集群管理页面操作列的获取接入地址获取。新版共享集群与专享集群命名地址在命名空间列表获取。 
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 
topicName	Topic 的名称，在控制台 Topic 页面复制。

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。

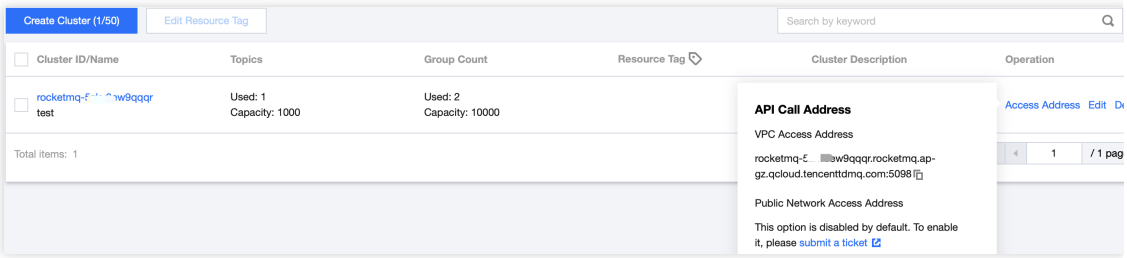
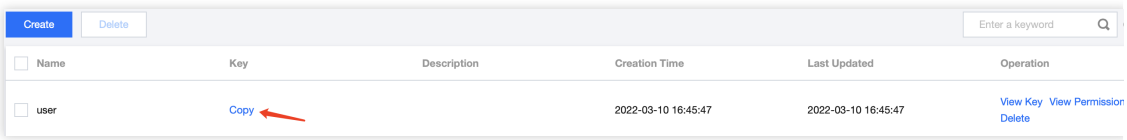
步骤3：消费消息

创建并编译运行消费消息程序。



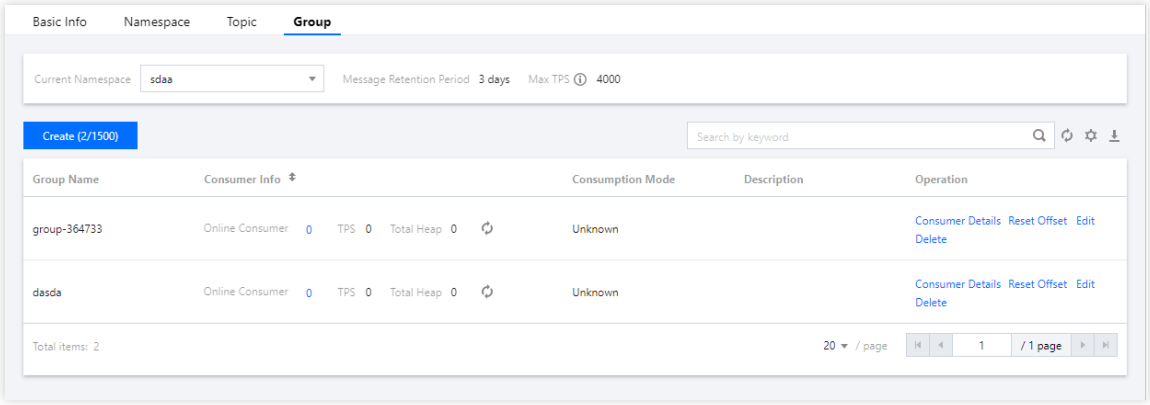
```
import time

    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
    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()
```

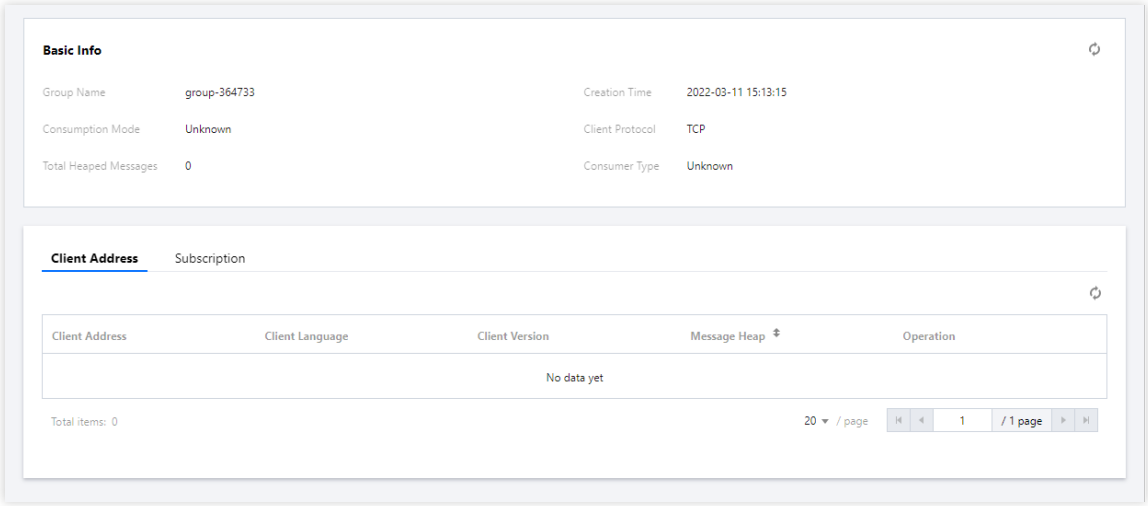
参数	说明
groupName	消费者 Group 的名称，在控制台 Group 页面复制。
nameserver	集群接入地址，在控制台集群管理页面操作列的获取接入地址获取。新版共享集群与专享集群命名地址在命名空间列表获取。

	
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 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 TDMQ 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**查看消费者详情**，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 Demo 或 RocketMQ-Client-Python示例。

HTTP 协议接入

最近更新时间：2023-05-16 11:07:52

操作场景

TDMQ RocketMQ 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 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 协议实现事务消息和顺序消息。
在创建 group（消费组）时需要制定类型（TCP 或者 HTTP，详情请参见 创建 Group 说明 ），因此，同一个 group（消费组）不支持 TCP 和 HTTP 客户端同时消费。

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
更多示例可以参考开源社区的 Demo 示例

重试机制

通过 HTTP 消费的每条消息，会有一个 5 分钟的**不可见时间。**
若客户端在不可见时间内 ACK 这条消息，则表示消费成功，不会重试。
若不可见时间到期后客户端仍未 ACK，则该条消息会重新可见，即接下来客户端会重新消费到这条消息。
需要注意，一次消费的不可见时间到期后消息句柄会失效，无法再对其进行 ACK。

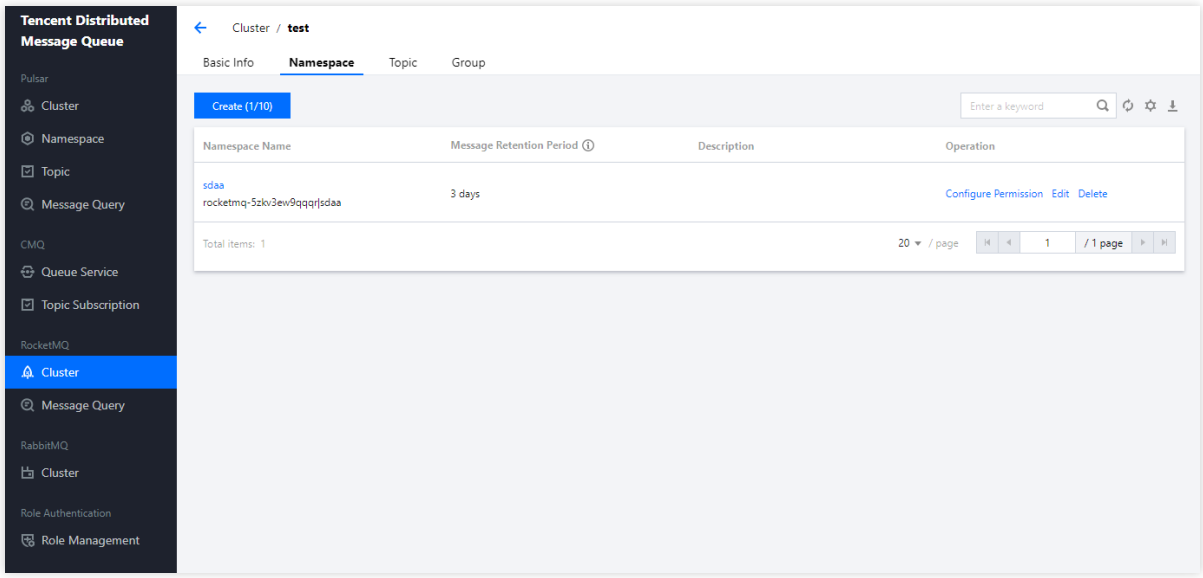
操作步骤

步骤1：引入依赖

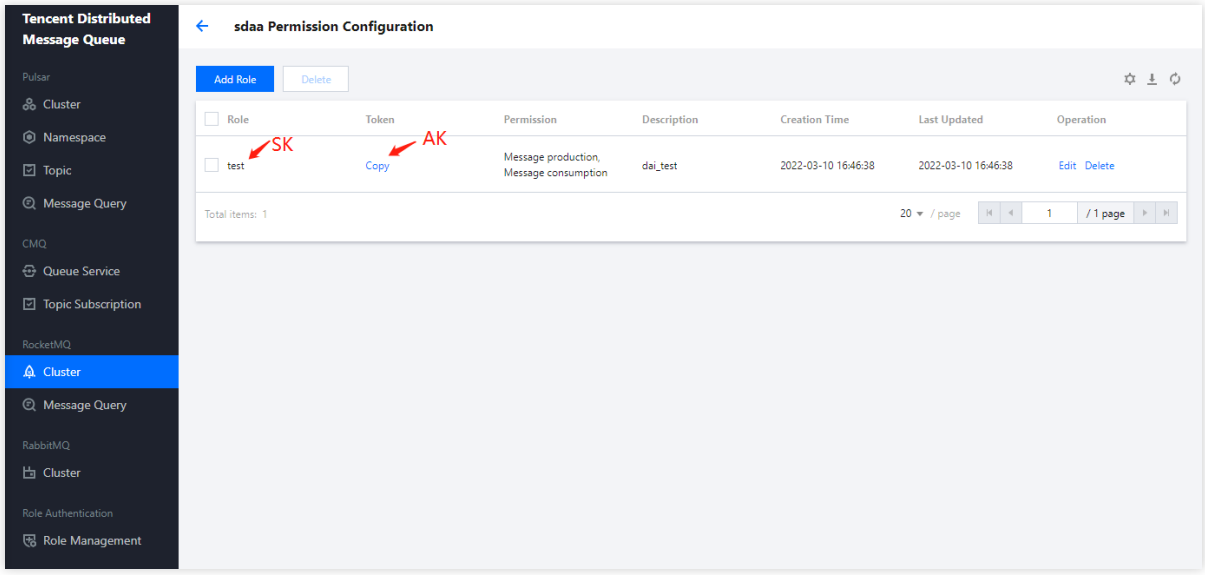
在工程的 pom.xml 文件中引入对应语言的 SDK 依赖。

步骤2：获取参数

1. 登录 TDMQ 控制台，选择所在的集群，点击集群名进入集群详情页。
2. 如下图所示，选择顶部的 “命名空间” 页签，单击右侧的 **配置权限** 进入权限配置页面，如当前角色列表为空，可以单击 **新建**，新建一个角色，详细描述请参见 完成资源创建与准备。



3. 在页面上复制对应的 AK 和 SK，以备在接下来的步骤中使用。

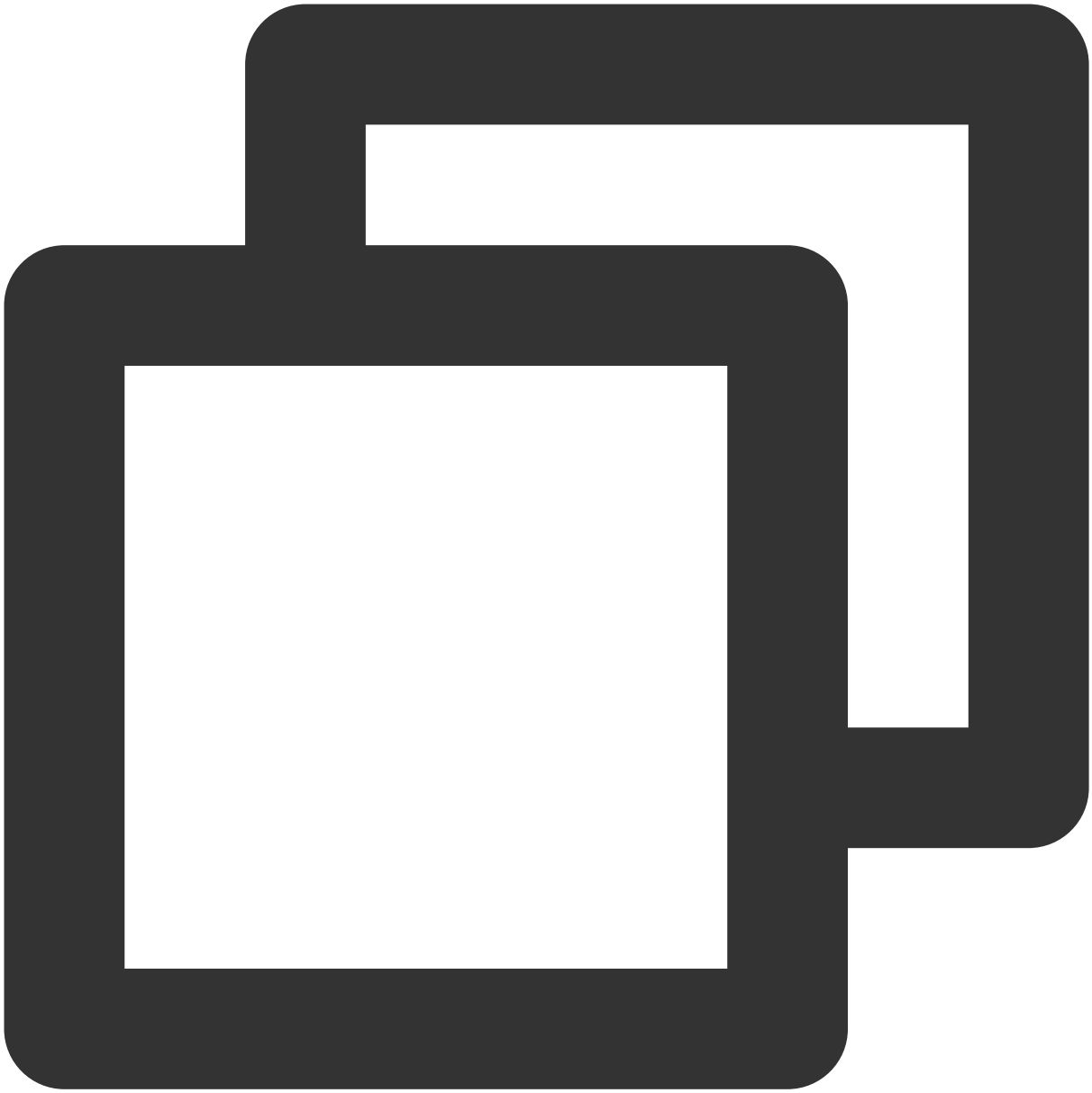


步骤3：生产者客户端初始化

JAVA

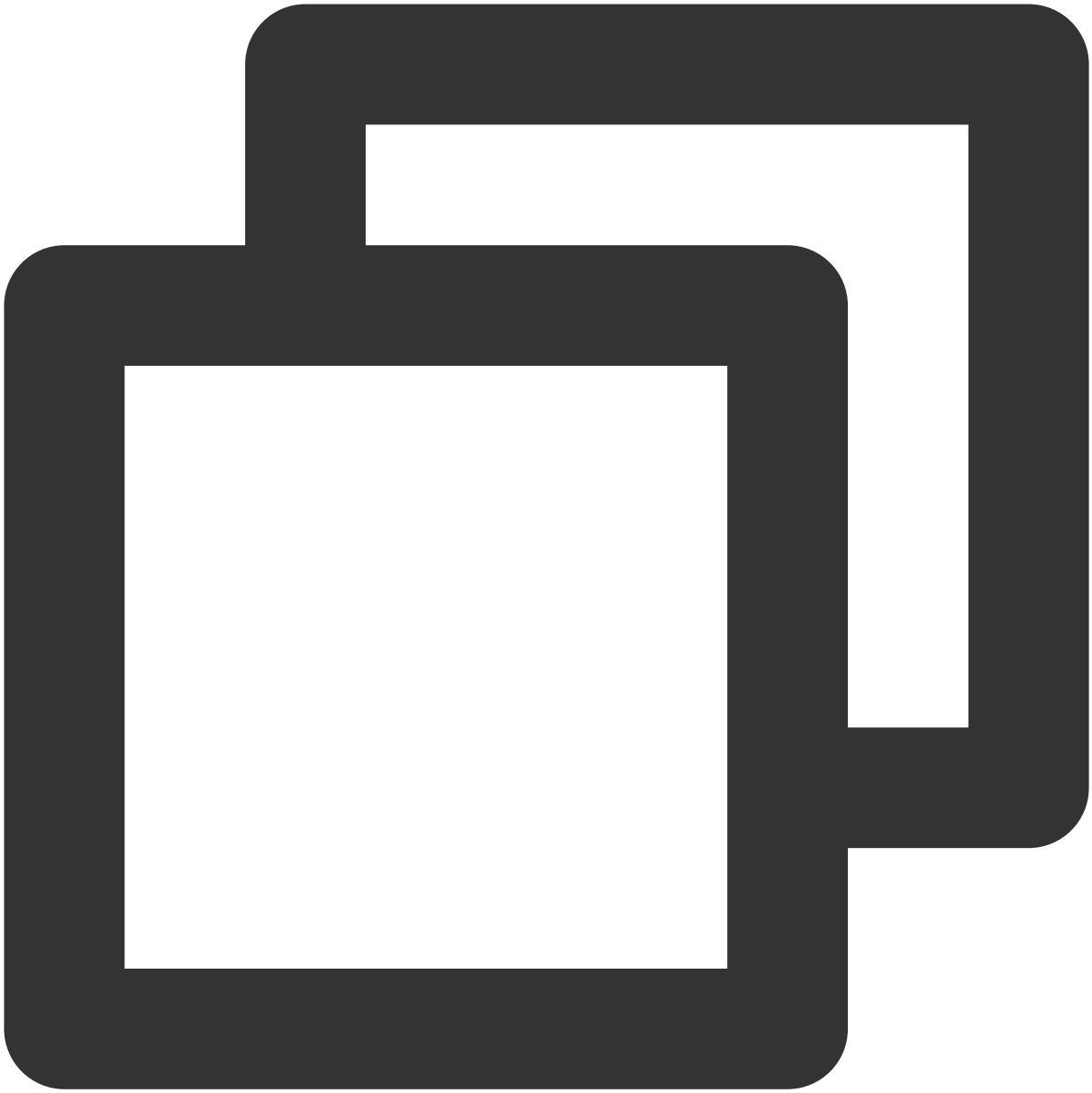
PHP

NodeJS



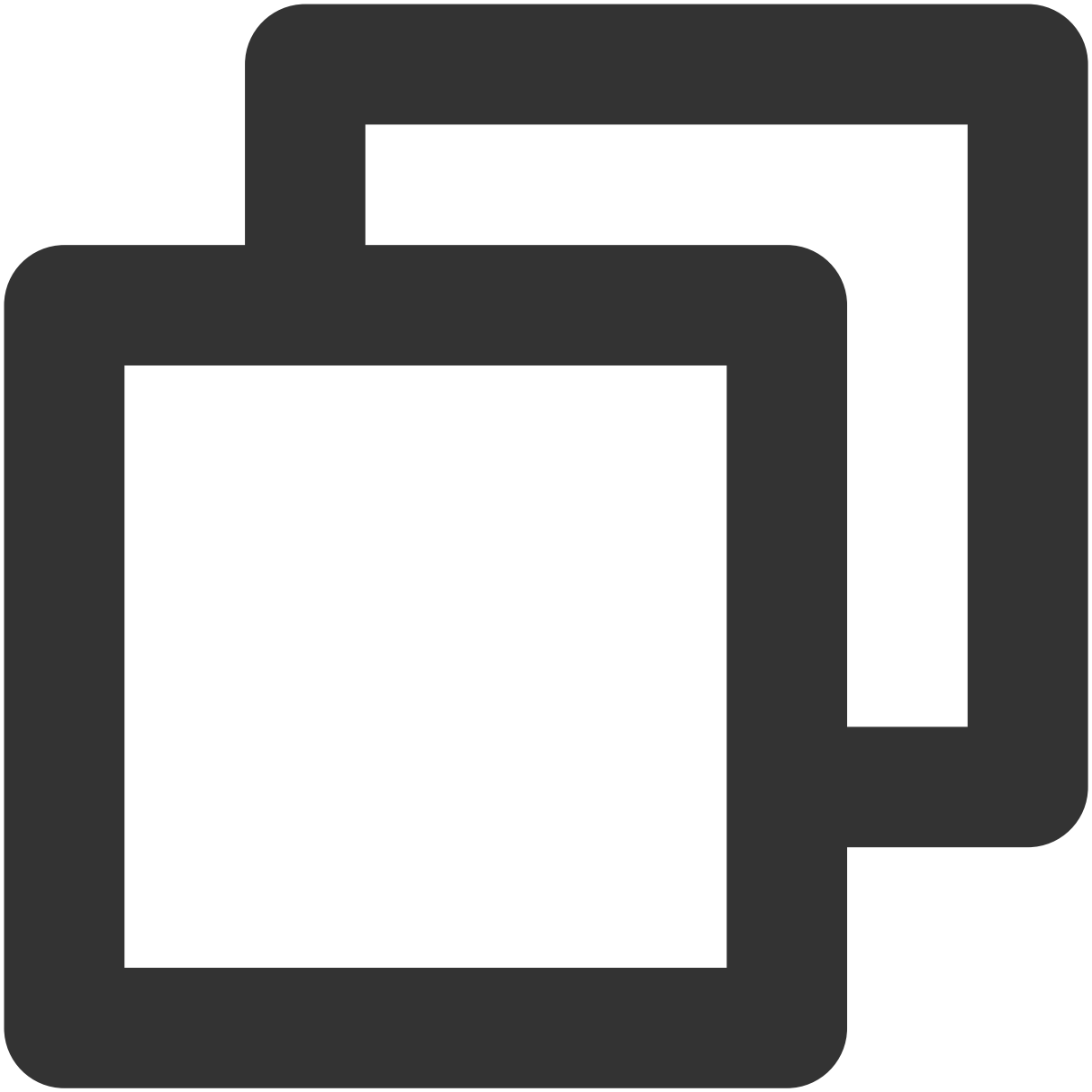
```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQProducer;

public class Producer {

    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // HTTP接入点
                "${HTTP_ENDPOINT}",
                // TDMQ RocketmQ access key, 从腾讯云控制台创建获取
                "${ACCESS_KEY}",
```

```
                // TDMQ RocketmQ 的角色名，从腾讯云控制台创建获取
                "${SECRET_KEY}"
        );

        // 消息发送的Topic，从腾讯云TDMQ控制台创建获取，必传参数
        final String topic = "${TOPIC}";
        // Topic所属的命名空间，从腾讯云TDMQ控制台创建获取，必传参数
        final String instanceId = "${INSTANCE_ID}";

        // 创建生产者
        MQProducer producer = mqClient.getProducer(instanceId, topic);

        // 发送消息

        mqClient.close();
    }
}
```



```
require "vendor/autoload.php";

use MQ\\MQClient;

class ProducerTest
{
    private $client;
    private $producer;

    public function __construct()
    {
```

```
        $this->client = new MQClient(
            // HTTP接入点
            "${HTTP_ENDPOINT}",
            // TDMQ RocketmQ access key, 从腾讯云控制台创建获取
            "${ACCESS_KEY}",
            // TDMQ RocketmQ 的角色名, 从腾讯云控制台创建获取
            "${SECRET_KEY}"
        );

        // 消息发送的Topic, 从腾讯云TDMQ控制台创建获取, 必传参数
        $topic = "${TOPIC}";
        // Topic所属的命名空间, 从腾讯云TDMQ控制台创建获取, 必传参数
        $instanceId = "${INSTANCE_ID}";

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

    public function run()
    {
        // 发送消息
    }
}


$instance = new ProducerTest();
$instance->run();
```



```
const {
  MQClient,
  MessageProperties
} = require('@aliyunmq/mq-http-sdk');

// 设置HTTP接入域名
const endpoint = "{Endpoint}";
// AccessKey
const accessKeyId = "{Accesskey}";
// SecretKey
const accessKeySecret = "rop";
```

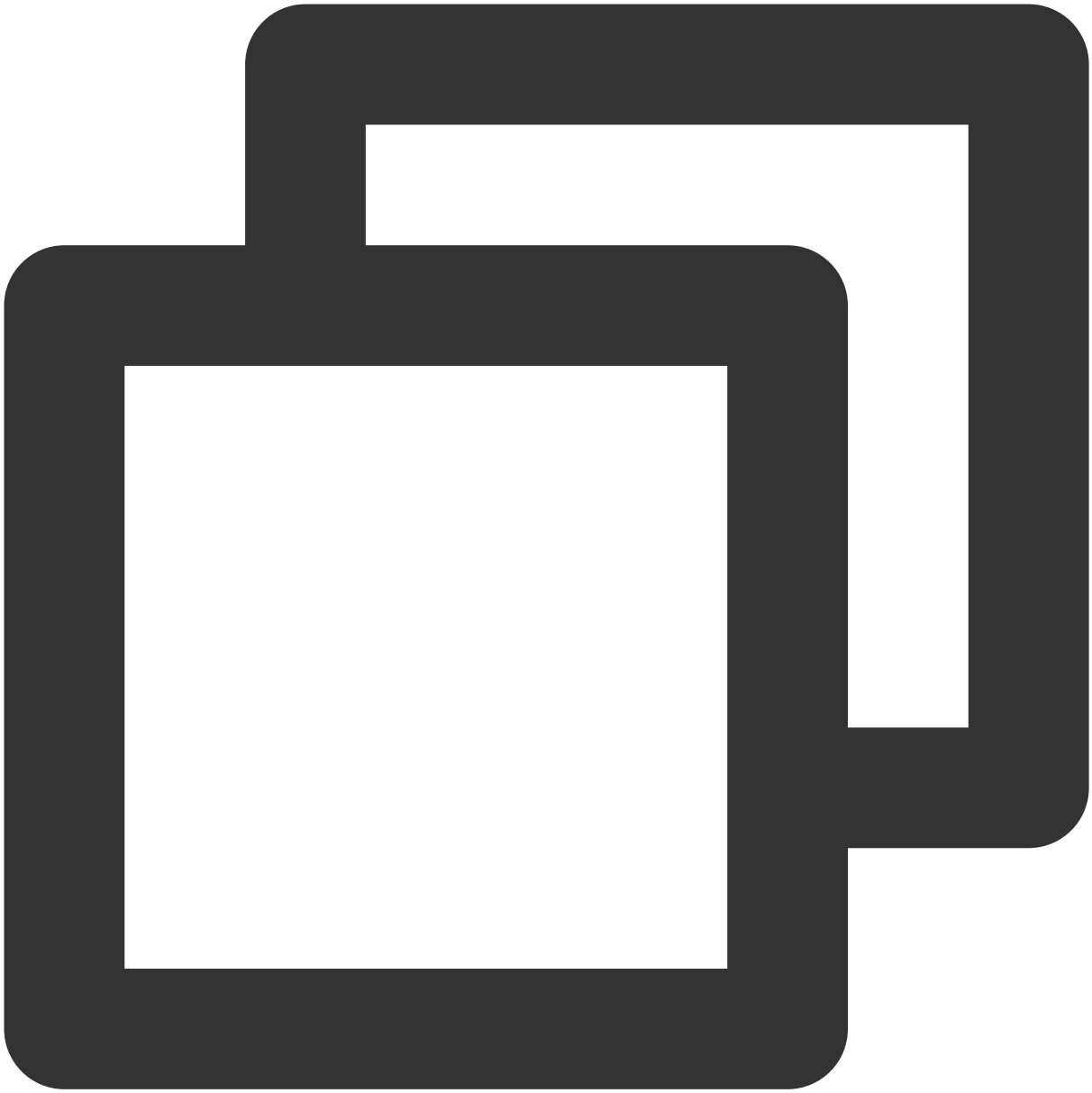
```
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);

// 所属的 Topic
const topic = "TopicA";
// Topic所属实例ID
const instanceId = "MQ_INST_xxxxx";

const producer = client.getProducer(instanceId, topic);

(async function(){
  try {
    // 循环发送4条消息
    for(var i = 0; i < 4; i++) {
      let res;
      if (i % 2 == 0) {
        msgProps = new MessageProperties();
        // 设置属性
        msgProps.putProperty("key", i);
        // 设置KEY
        msgProps.messageKey("MessageKey");
        res = await producer.publishMessage("hello mq.", "", msgProps);
      } else {
        msgProps = new MessageProperties();
        // 设置属性
        msgProps.putProperty("key", i);
        // 定时消息，定时时间为10s后
        msgProps.startDeliverTime(Date.now() + 10 * 1000);
        res = await producer.publishMessage("hello mq. timer msg!", "TagA", msgProp
      }
      console.log("Publish message: MessageID:%s,BodyMD5:%s", res.body.MessageId, r
    }

  } catch(e) {
    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理
    console.log(e)
  }
})();
```

步骤4：消费者客户端初始化

JAVA

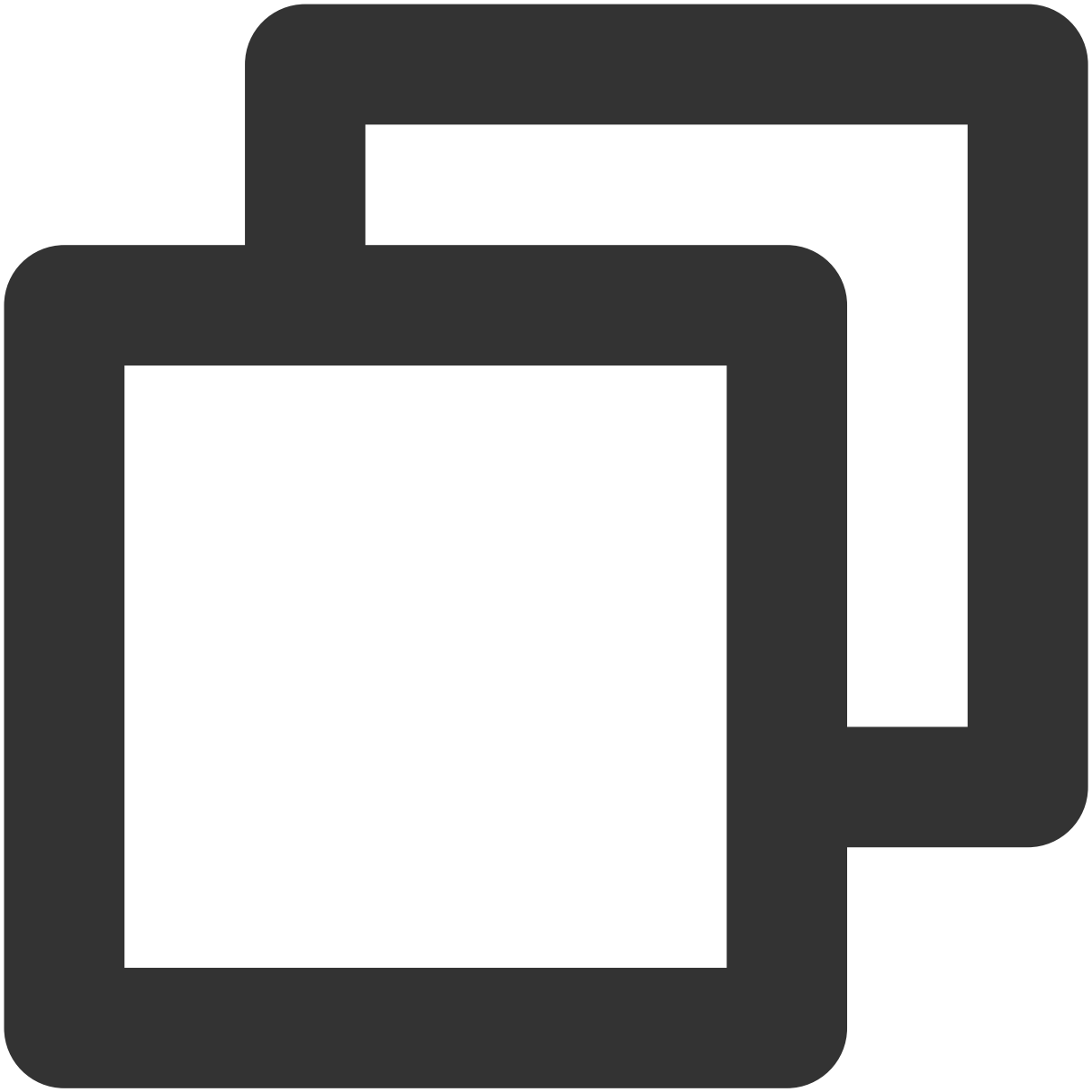
PHP

NodeJS



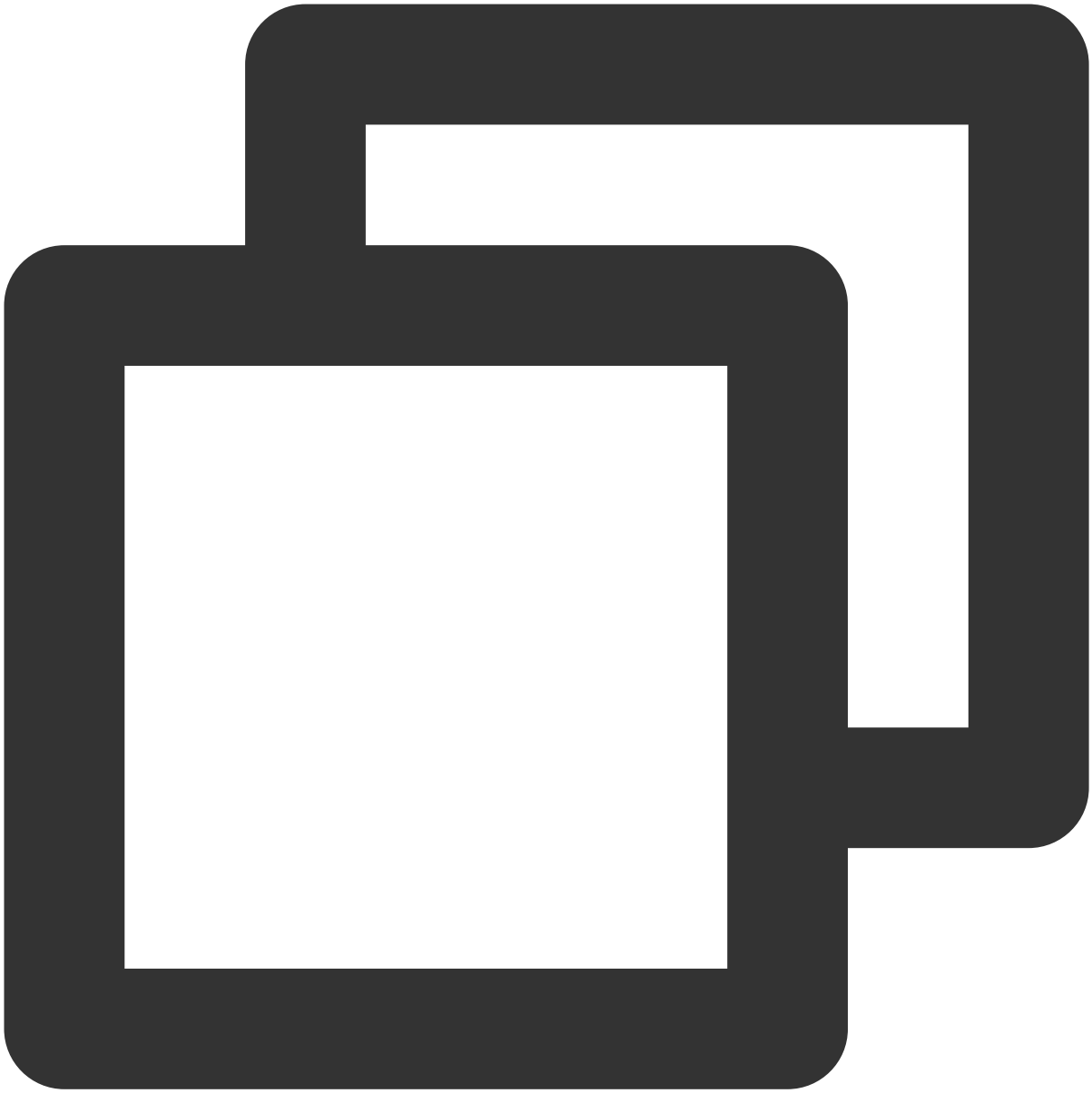
```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQConsumer;

public class Consumer {

    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // HTTP接入点
                "${HTTP_ENDPOINT}",
                // TDMQ RocketmQ access key, 从腾讯云控制台创建获取
                "${ACCESS_KEY}",
```

```
                // TDMQ RocketmQ 的角色名，从腾讯云控制台创建获取
                "${SECRET_KEY}"
        );

        // 消费的Topic，从腾讯云TDMQ控制台创建获取，必传参数
        final String topic = "${TOPIC}";
        // 消费组名称，从腾讯云TDMQ控制台创建获取，必传参数
        final String groupId = "${GROUP_ID}";
        // Topic所属的命名空间，从腾讯云TDMQ控制台创建获取，必传参数
        final String instanceId = "${INSTANCE_ID}";

        final MQConsumer consumer = mqClient.getConsumer(instanceId, topic, groupId

        do {
            // 消费消息
        } while (true);
    }
}
```



```
require "vendor/autoload.php";

use MQ\\MQClient;

class ConsumerTest
{
    private $client;
    private $consumer;

    public function __construct()
    {
```

```
        $this->client = new MQClient(
            // HTTP接入点
            "${HTTP_ENDPOINT}",
            // TDMQ RocketmQ access key, 从腾讯云控制台创建获取
            "${ACCESS_KEY}",
            // TDMQ RocketmQ 的角色名, 从腾讯云控制台创建获取
            "${SECRET_KEY}"
        );

        // 消费的Topic, 从腾讯云TDMQ控制台创建获取, 必传参数
        $topic = "${TOPIC}";
        // 消费组名称, 从腾讯云TDMQ控制台创建获取, 必传参数
        $groupId = "${GROUP_ID}";
        // Topic所属的命名空间, 从腾讯云TDMQ控制台创建获取, 必传参数
        $instanceId = "${INSTANCE_ID}";

        $this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId)
    }

    public function run()
    {
        while (True) {
            // 消费消息
        }
    }
}


$instance = new ConsumerTest();
$instance->run();
```



```
const {
  MQClient
} = require('@aliyunmq/mq-http-sdk');

// 设置HTTP接入域名
const endpoint = "{Endpoint}";
// AccessKey
const accessKeyId = "{Accesskey}";
// SecretKey
const accessKeySecret = "rop";
```

```
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);

// 所属的 Topic
const topic = "TopicA";
// Topic所属实例ID
const instanceId = "MQ_INST_xxxxx";
// 您在控制台创建的 Consumer Group
const groupId = "GID_xxx";

const consumer = client.getConsumer(instanceId, topic, groupId);

(async function(){
  // 循环消费消息
  while(true) {
    try {
      // 长轮询消费消息
      // 长轮询表示如果topic没有消息则请求会在服务端挂住3s，3s内如果有消息可以消费则立即返回
      res = await consumer.consumeMessage(
          3, // 一次最多消费3条(最多可设置为16条)
          3 // 长轮询时间3秒（最多可设置为30秒）
          );

      if (res.code == 200) {
        // 消费消息，处理业务逻辑
        console.log("Consume Messages, requestId:%s", res.requestId);
        const handles = res.body.map((message) => {
          console.log("\\tMessageId:%s,Tag:%s,PublishTime:%d,NextConsumeTime:%d,Fir
            ",Props:%j,MessageKey:%s,Prop-A:%s",
              message.MessageId, message.MessageTag, message.PublishTime, message.N
              message.MessageBody,message.Properties,message.MessageKey,message.Pro
          return message.ReceiptHandle;
        });

        // message.NextConsumeTime前若不确认消息消费成功，则消息会重复消费
        // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样
        res = await consumer.ackMessage(handles);
        if (res.code != 204) {
          // 某些消息的句柄可能超时了会导致确认不成功
          console.log("Ack Message Fail:");
          const failHandles = res.body.map((error)=>{
            console.log("\\tErrorHandle:%s, Code:%s, Reason:%s\\n", error.ReceiptHa
            return error.ReceiptHandle;
          });
          handles.forEach((handle)=>{
            if (failHandles.indexOf(handle) < 0) {
              console.log("\\tSucHandle:%s\\n", handle);
            }
```

```
            });
          } else {
            // 消息确认消费成功
            console.log("Ack Message suc, RequestId:%s\\n\\t", res.requestId, handles
          }
        }
      } catch(e) {
        if (e.Code.indexOf("MessageNotExist") > -1) {
          // 没有消息，则继续长轮询服务器
          console.log("Consume Message: no new message, RequestId:%s, Code:%s", e.Req
        } else {
          console.log(e);
        }
      }
    }
})();
```

HTTP 协议接入
Java SDK
发送与接收普通消息

最近更新时间：2023-09-13 11:36:47

操作场景

TDMQ RocketMQ 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 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。
在创建 Group（消费组）时需要制定类型（TCP 或者 HTTP，详情请参见 创建 Group 说明 ），因此，同一个 Group（消费组）不支持 TCP 和 HTTP 客户端同时消费。

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
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，暂不支持自定义。

消息类型	重试间隔	最大重试次数
普通消息	5分钟	288
顺序消息	1分钟	288

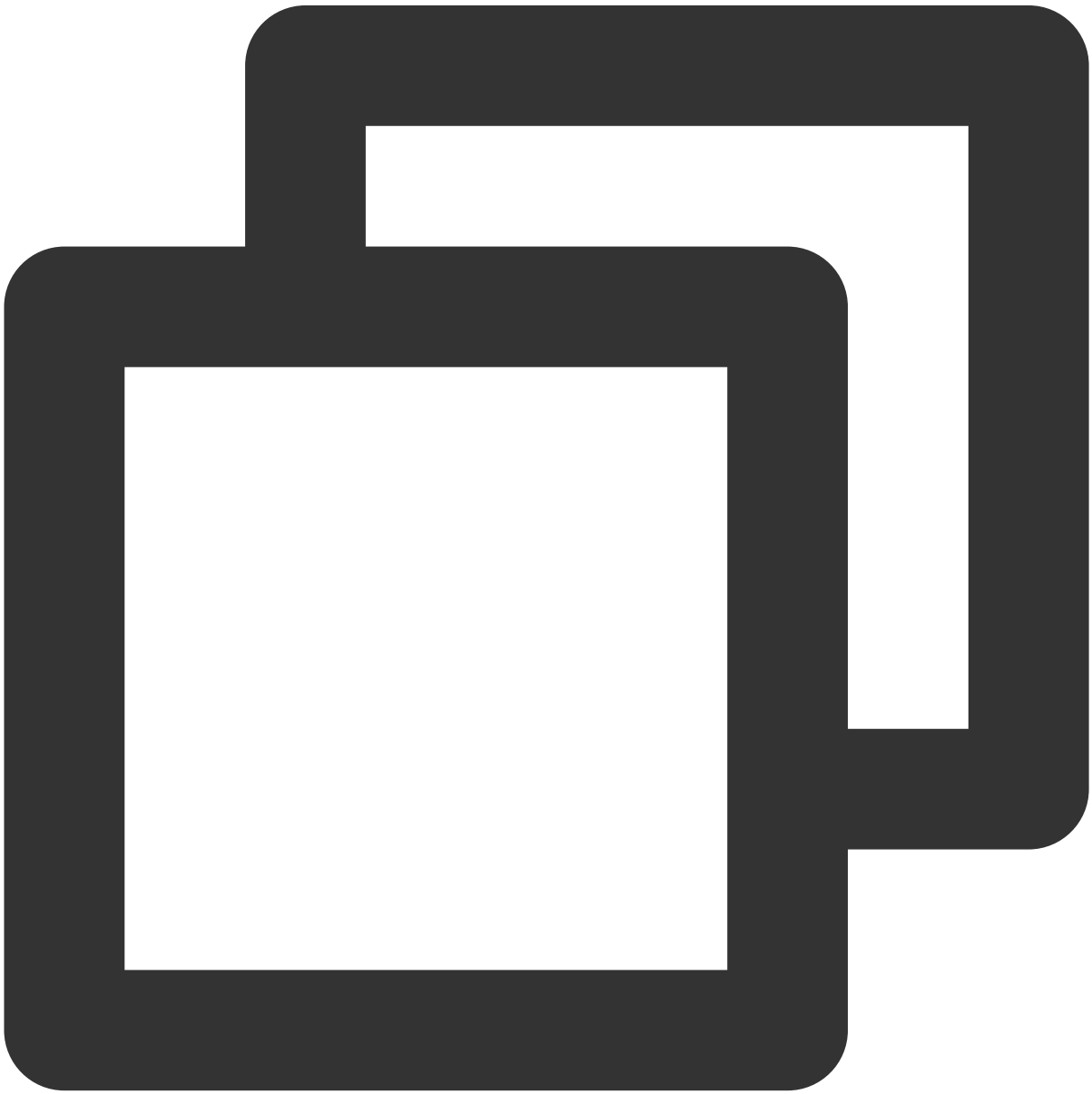
说明：

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

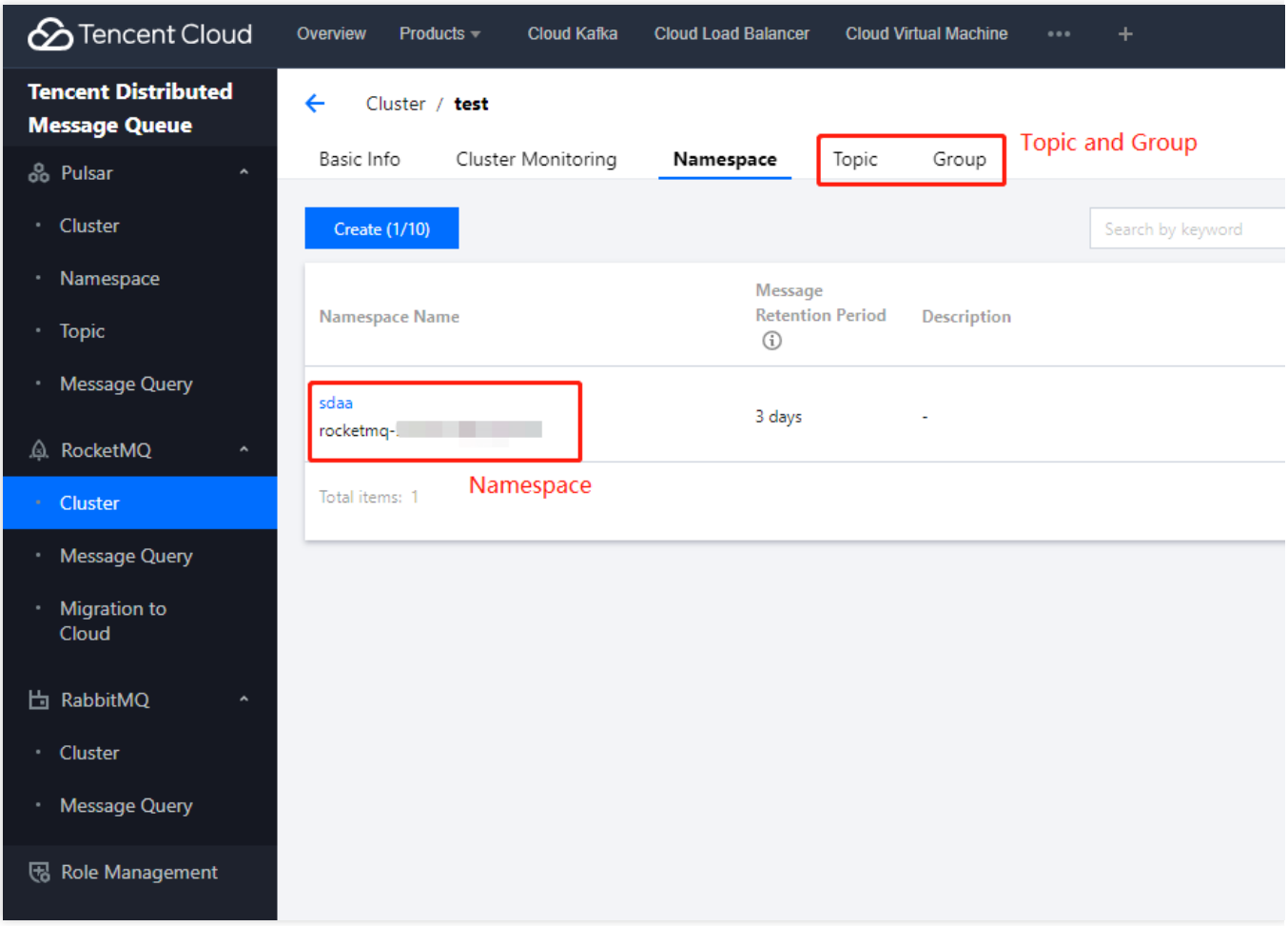
在 Java 项目中引入相关依赖，以 maven 工程为例，在 pom.xml 添加以下依赖：



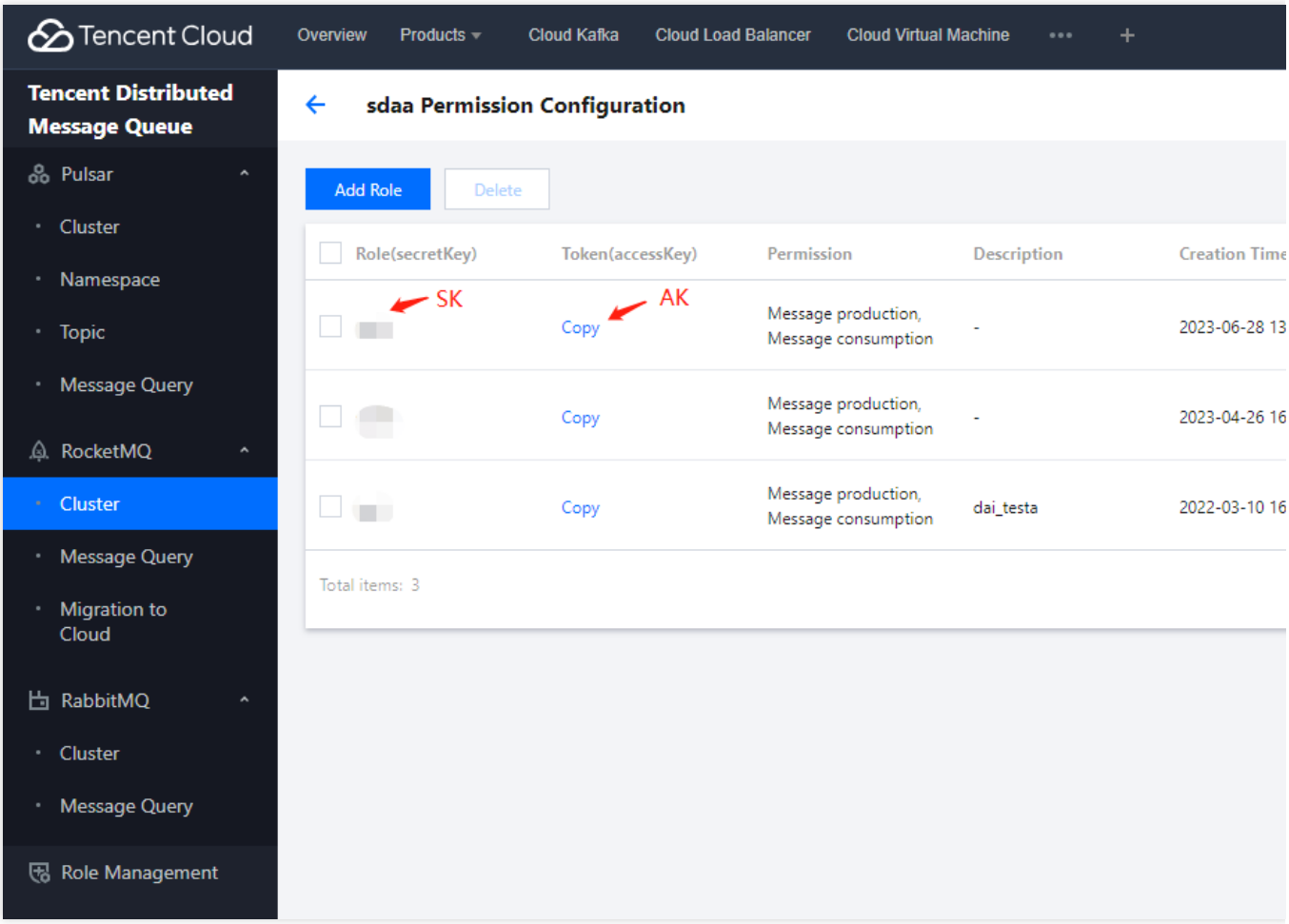
```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>com.aliyun.mq</groupId>
      <artifactId>mq-http-sdk</artifactId>
      <version>1.0.3</version>
  </dependency>
```

步骤2：获取参数

1. 登录 TDMQ 控制台，选择所在的集群，单击集群名进入集群详情页。
2. 如下图所示，选择顶部的**命名空间**页签，单击右侧的**配置权限**进入权限配置页面，如当前角色列表为空，可以单击**新建**，新建一个角色，详细描述请参见 完成资源创建与准备。

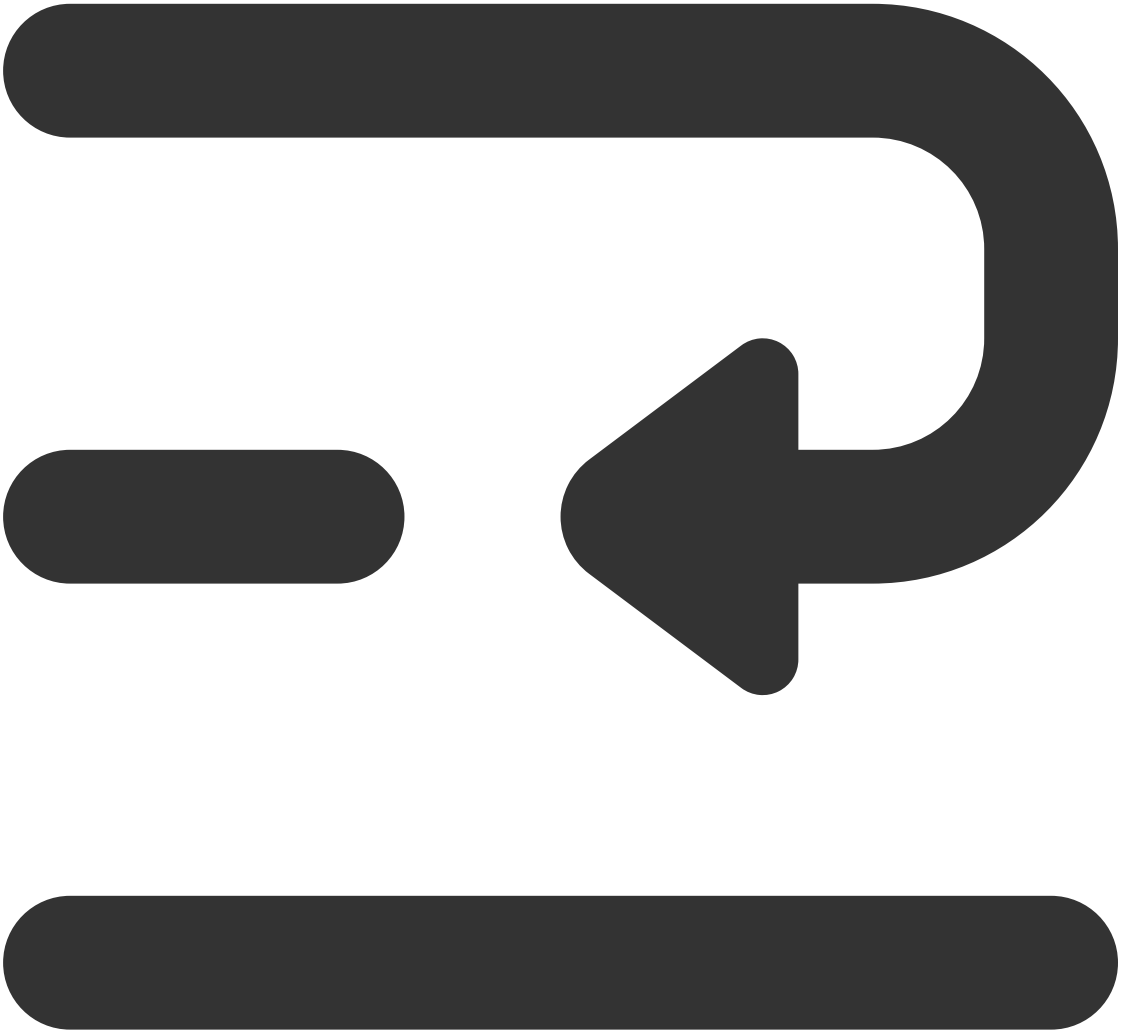


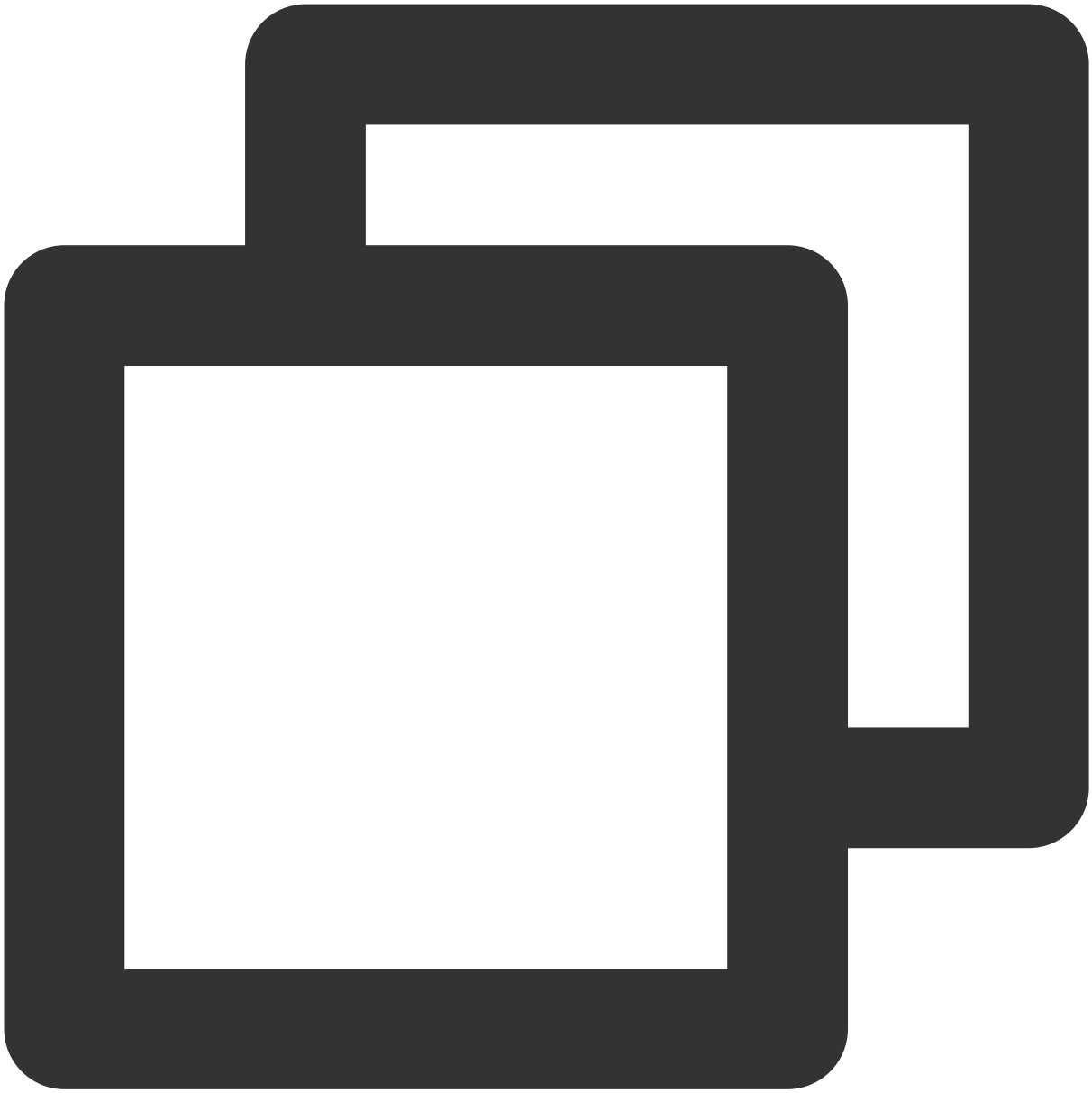
3. 在页面上复制对应的 AK 和 SK，以备在接下来的步骤中使用。



步骤3：生产消息

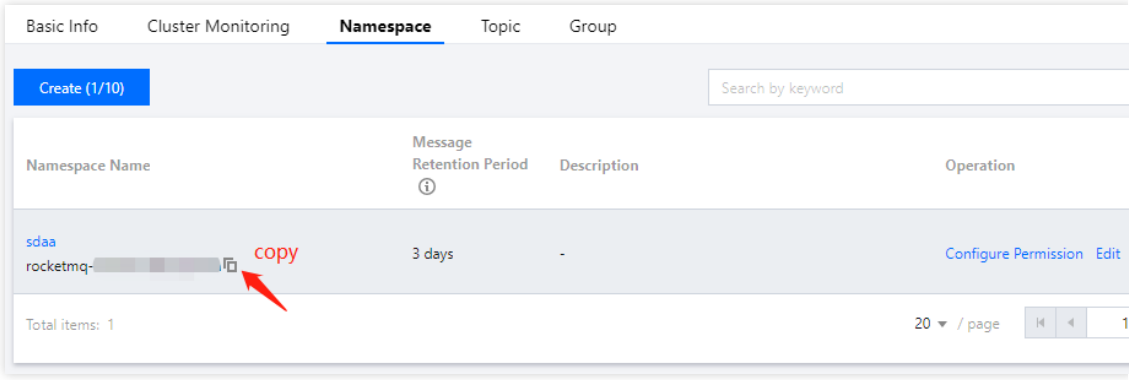
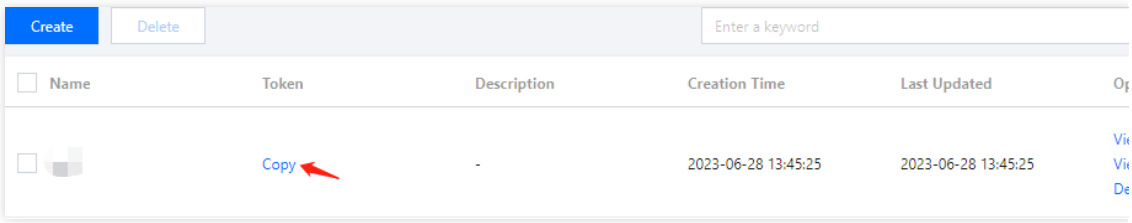
创建消息生产者



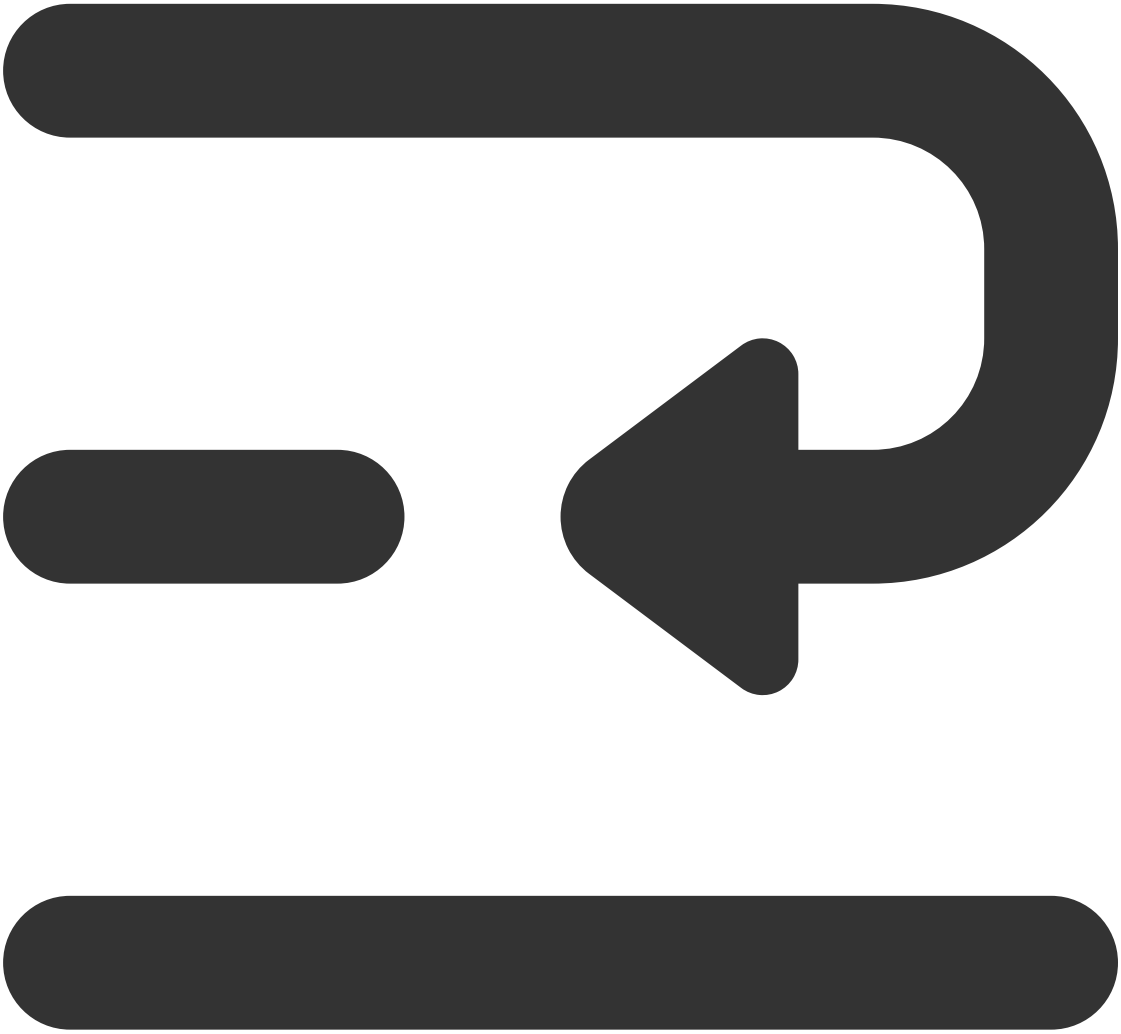


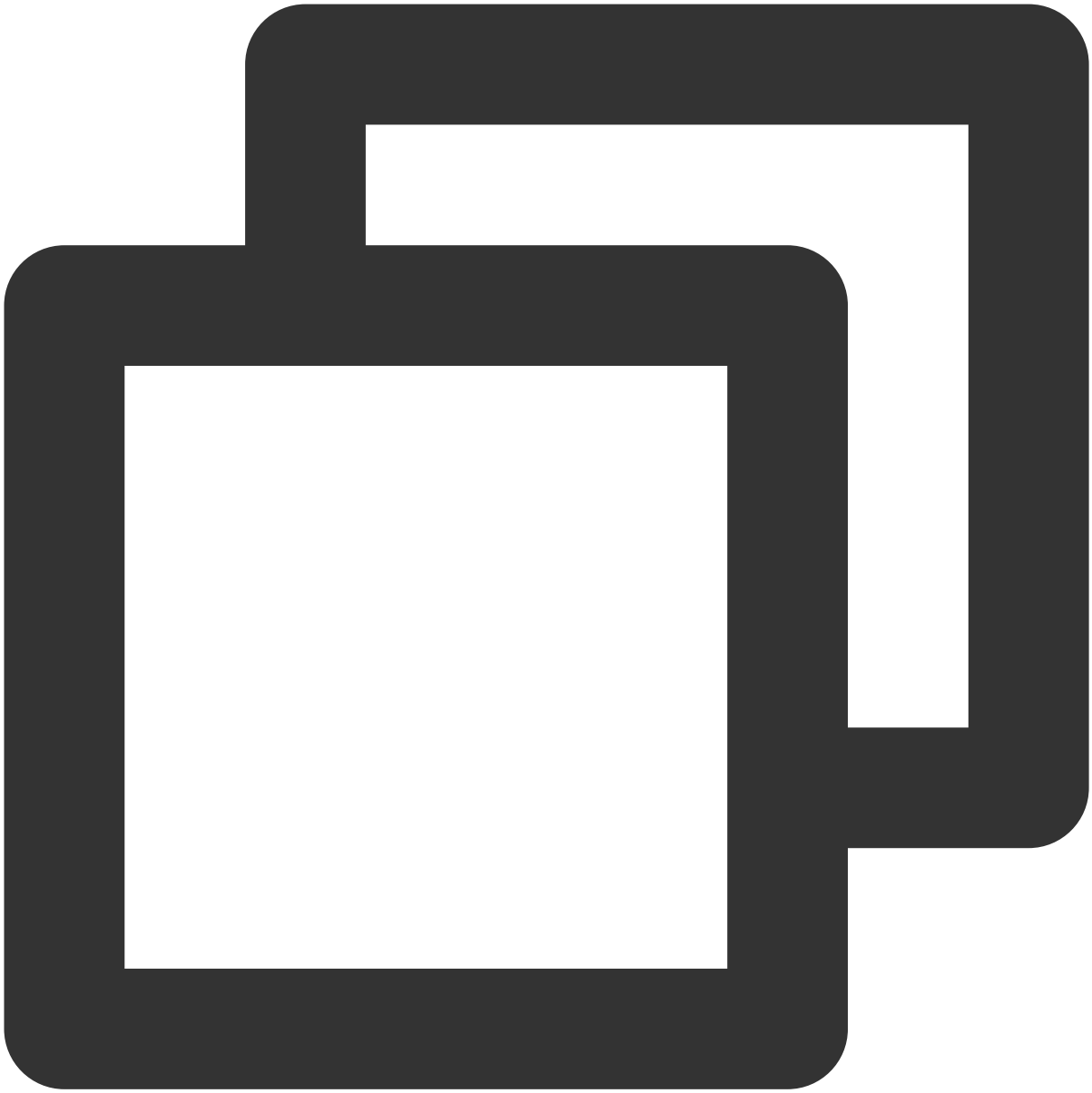
```
// 获取Client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

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

参数	说明
topicName	主题名称，在控制台集群管理中 Topic 页签中复制。
namespace	命名空间名称，在控制台集群管理中 Namespace 页签中复制。

	
endpoint	集群 HTTP 协议接入地址，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。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

发送消息





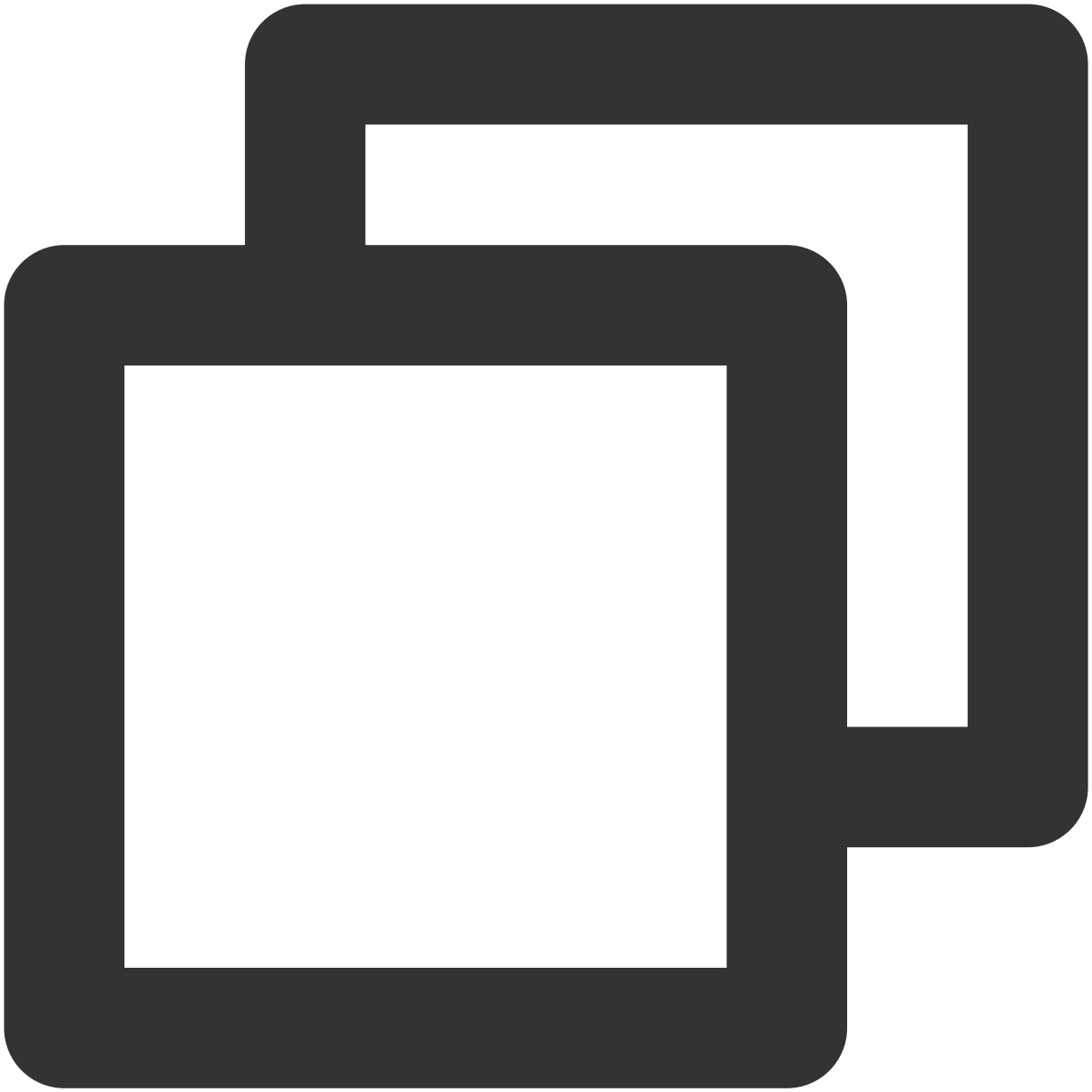
```
try {
    for (int i = 0; i < 10; i++) {
        TopicMessage pubMsg;
        pubMsg = new TopicMessage(
            ("Hello RocketMQ " + i).getBytes(),
            "TAG"
        );
        TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
        System.out.println("Send mq message success.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

```
    System.out.println("Send mq message failed.");
    e.printStackTrace();
}
```

参数	说明
TAG	设置消息的 TAG。

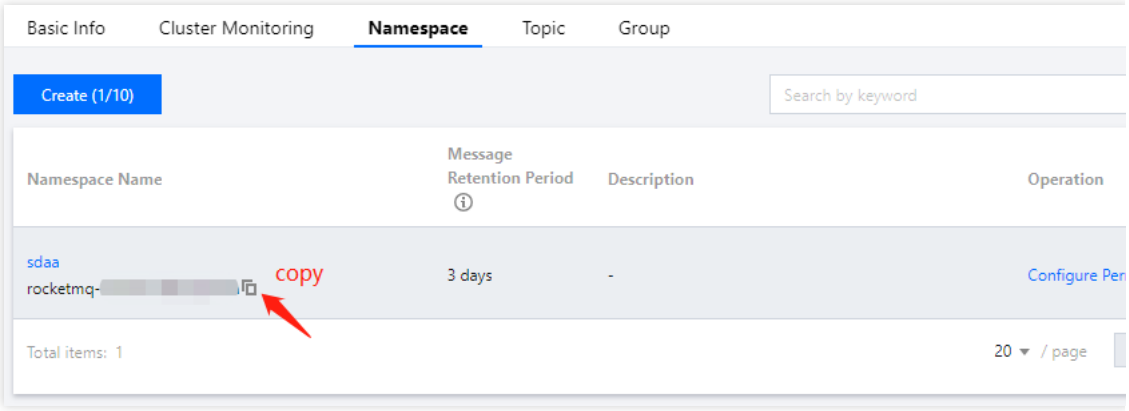
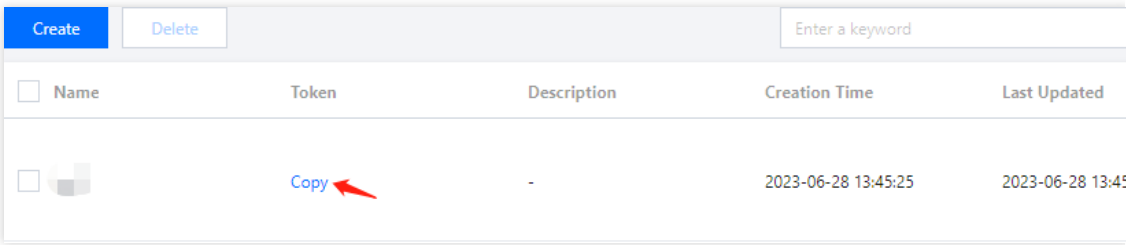
步骤4：消费消息

创建消费者

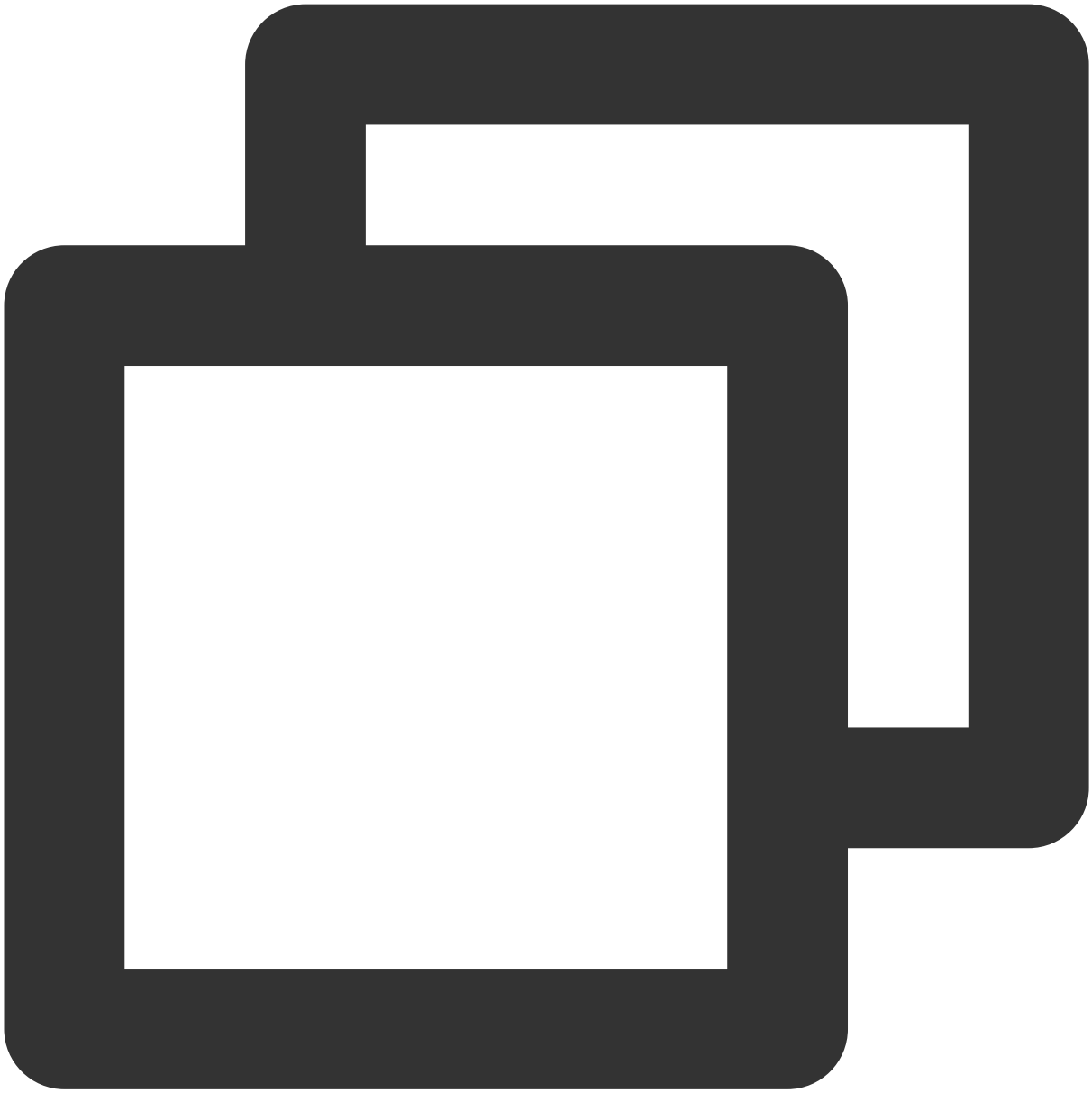


```
// 获取Client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

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

参数	说明
topicName	主题名称，在控制台集群管理中 Topic 页签中复制。
groupName	生产者组名称，在控制台集群管理中 Group 页签中复制。

namespace	命名空间名称，在控制台集群管理中 Namespace 页签中复制。 
TAG	订阅的标签。
endpoint	集群 HTTP 协议接入地址，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。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

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
        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, co
            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.getRe
                    if (errors.getErrorMessages() != null) {
                        for (String errorHandle :errors.getErrorMessages().keySet()) {
                            System.out.println("Handle:" + errorHandle + ", ErrorCode:"
                                    + ", ErrorMsg:" + errors.getErrorMessages().get(err
                        }
                    }
                    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：2023-09-13 11:38:17

操作场景

TDMQ RocketMQ 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 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。
在创建 Group（消费组）时需要制定类型（TCP 或者 HTTP，详情请参见 创建 Group 说明 ），因此，同一个 Group（消费组）不支持 TCP 和 HTTP 客户端同时消费。

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
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，暂不支持自定义。

消息类型	重试间隔	最大重试次数
普通消息	5分钟	288
顺序消息	1分钟	288

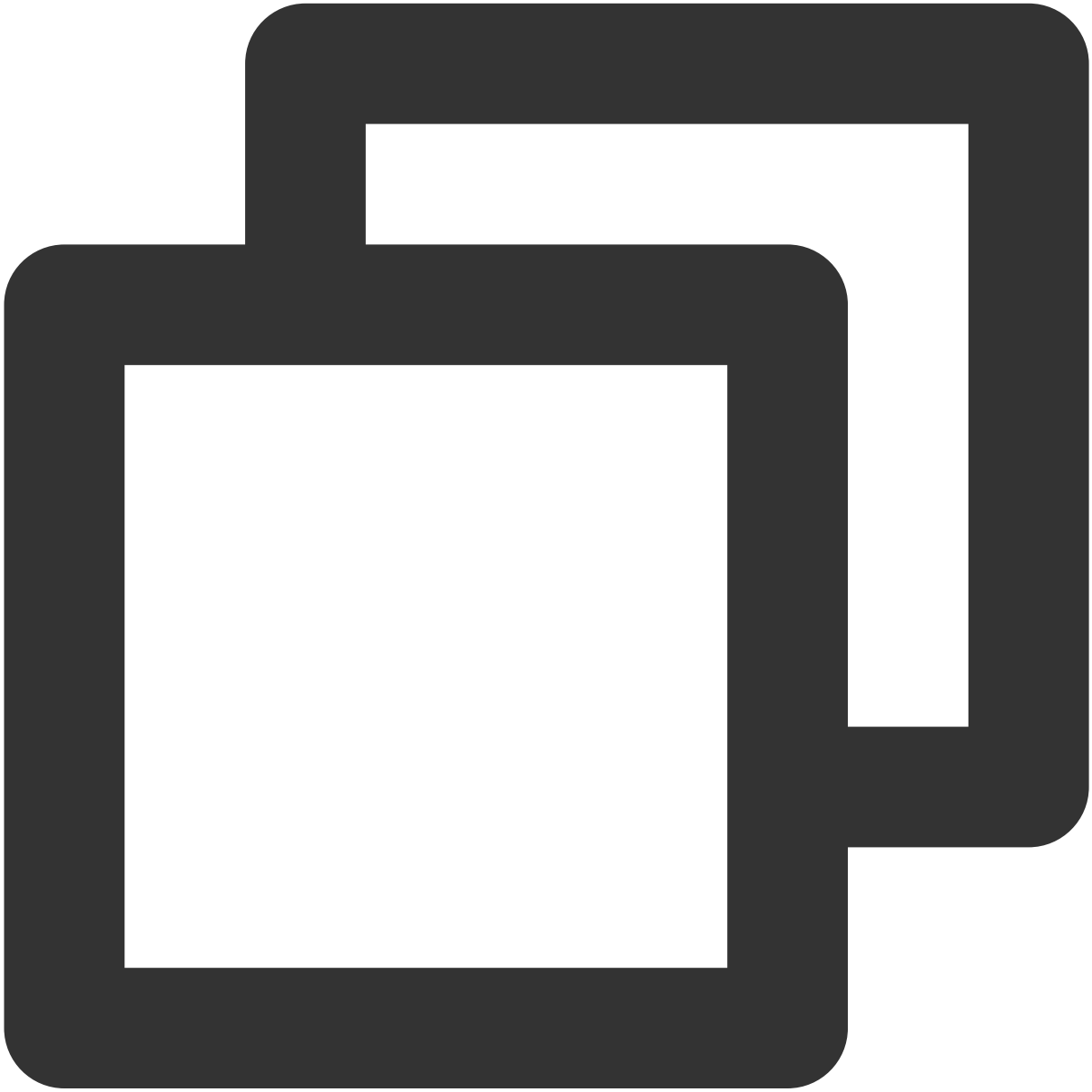
说明：

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：

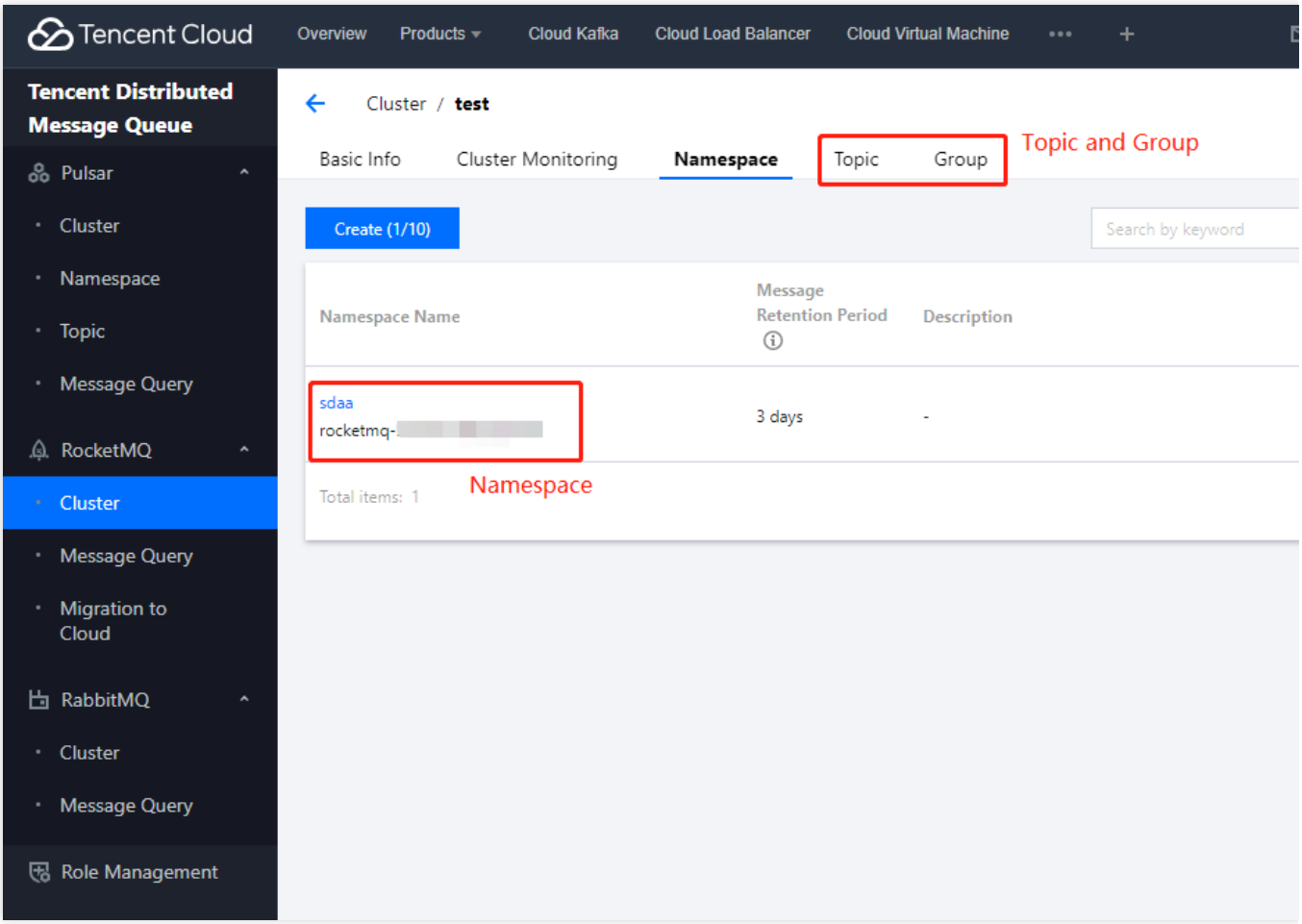


```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>com.aliyun.mq</groupId>
      <artifactId>mq-http-sdk</artifactId>
      <version>1.0.3</version>
```

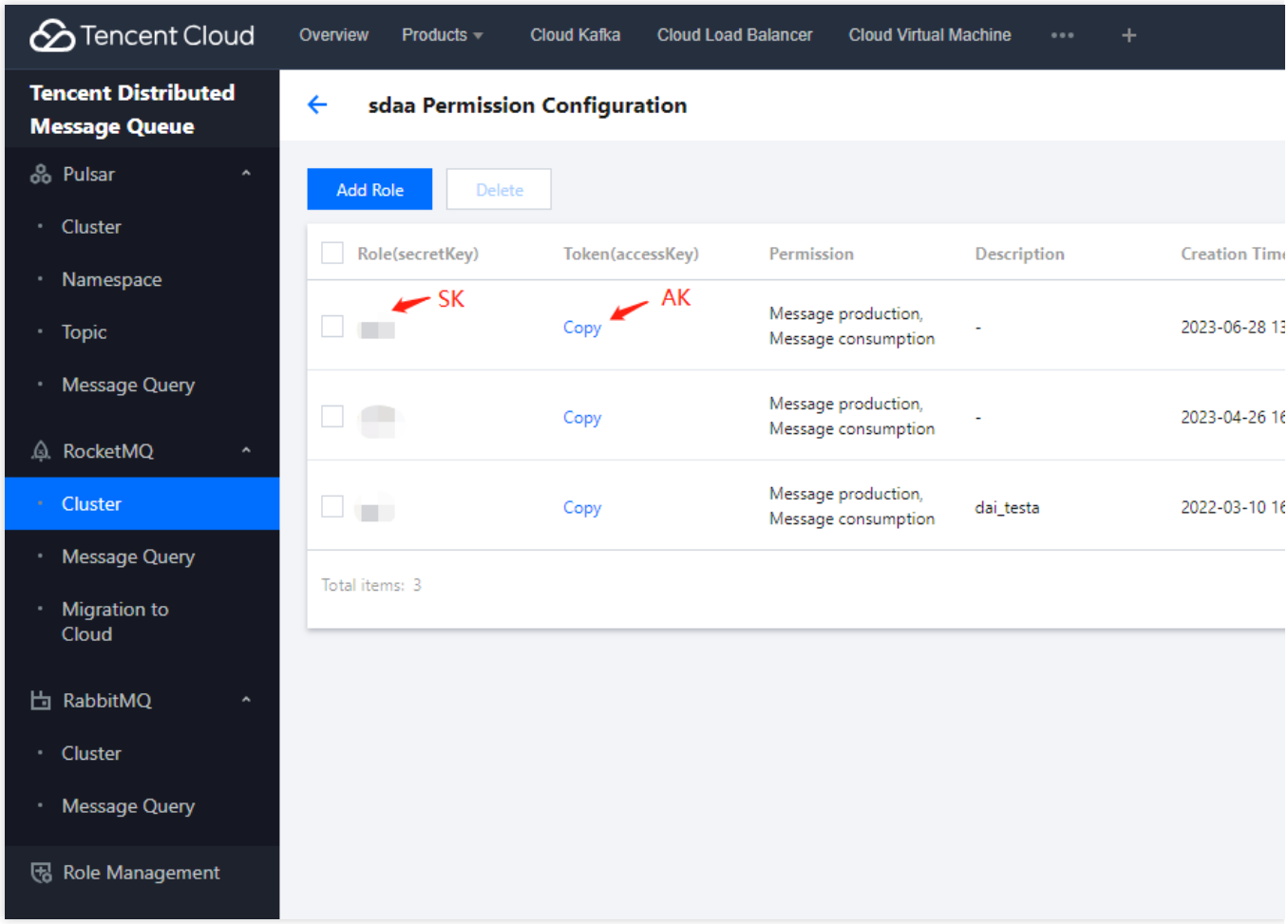
```
    </dependency>
```

步骤2：获取参数

1. 登录 TDMQ 控制台，选择所在的集群，单击集群名进入集群详情页。
2. 如下图所示，选择顶部的**命名空间**页签，单击右侧的**配置权限**进入权限配置页面，如当前角色列表为空，可以单击**新建**，新建一个角色，详细描述请参见 完成资源创建与准备。

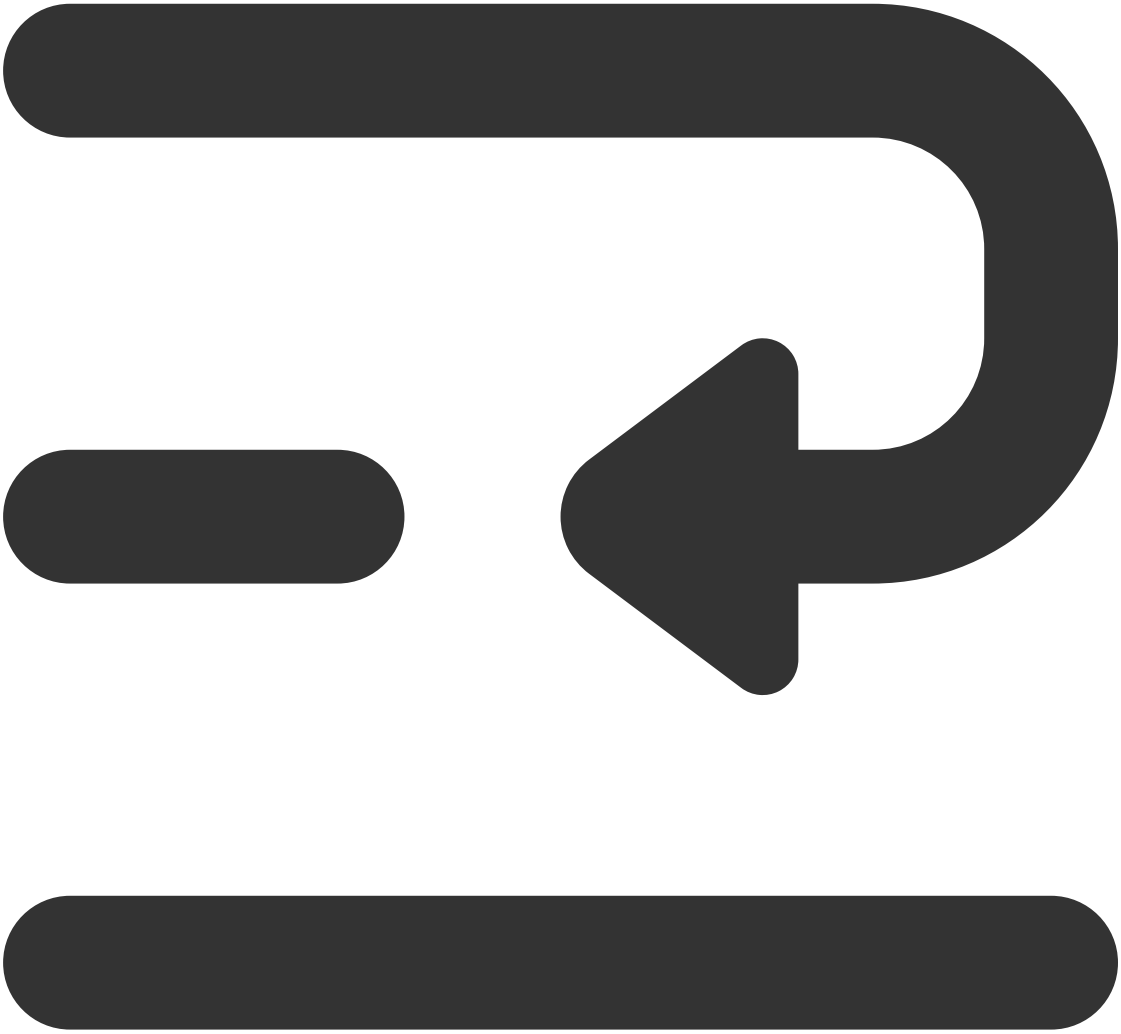


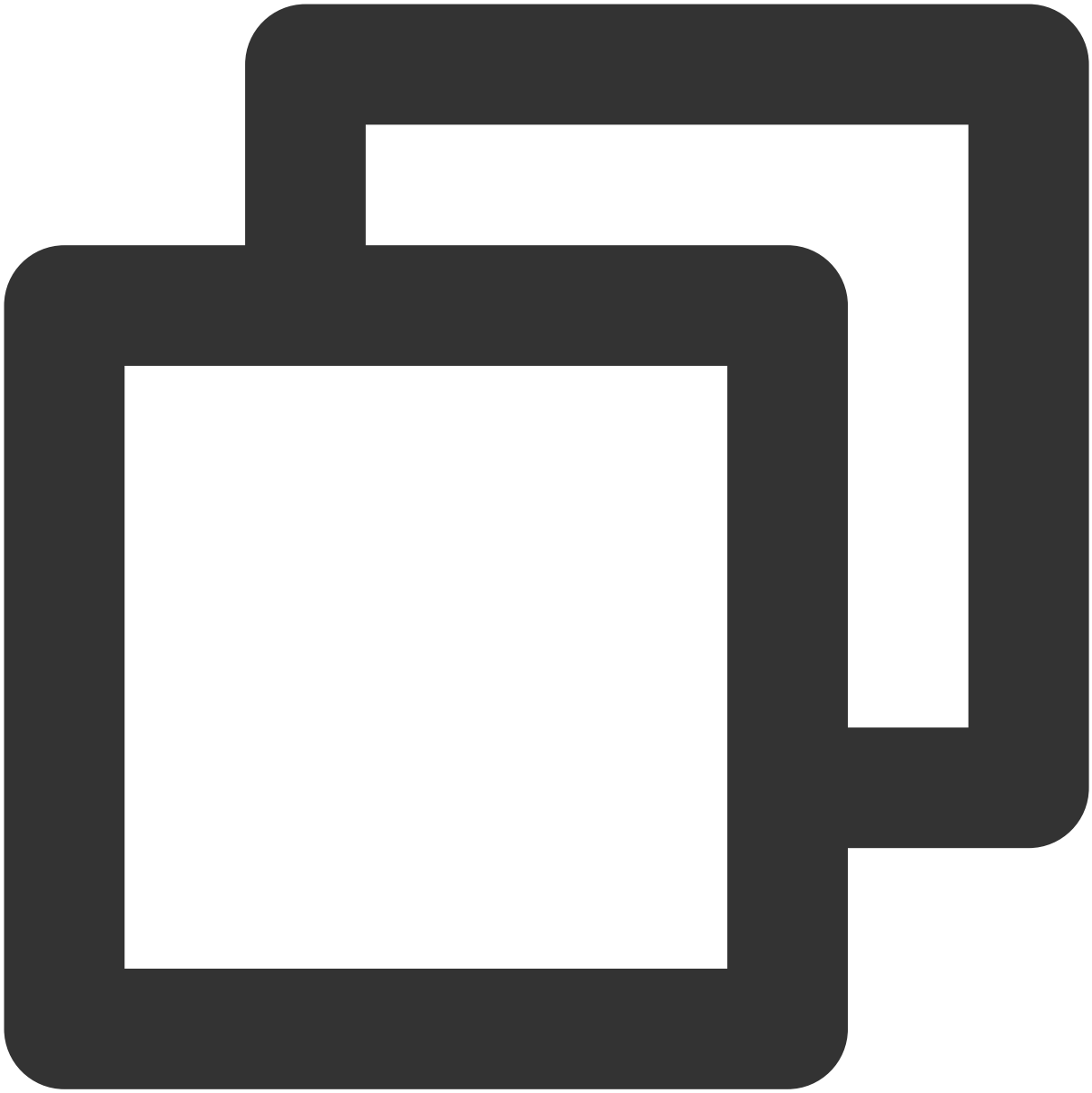
3. 在页面上复制对应的 AK 和 SK，以备在接下来的步骤中使用。



步骤3：生产消息

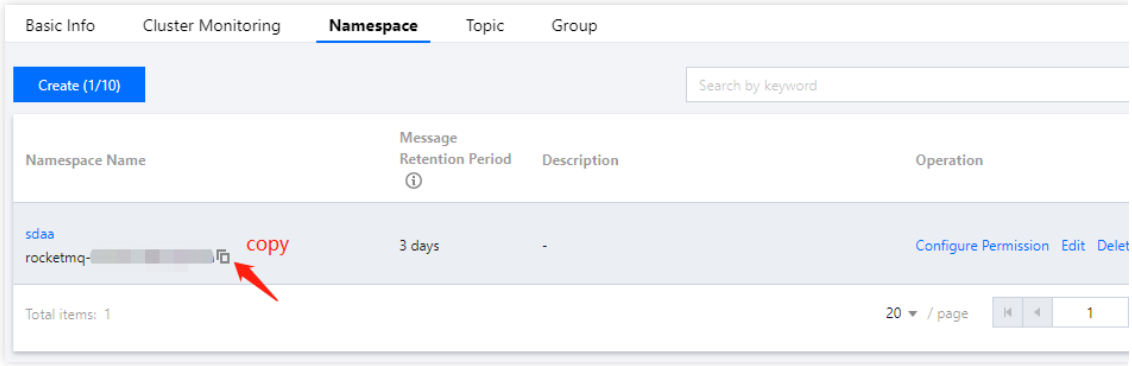
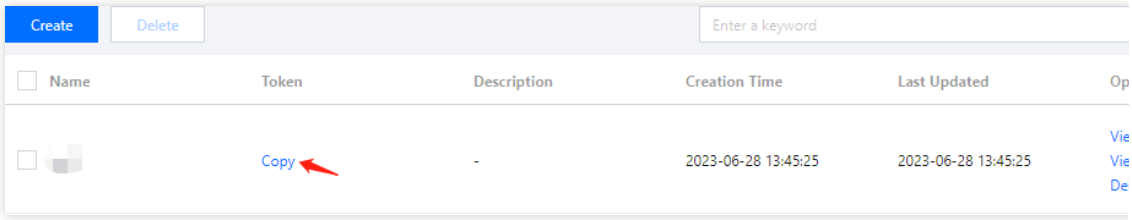
创建消息生产者



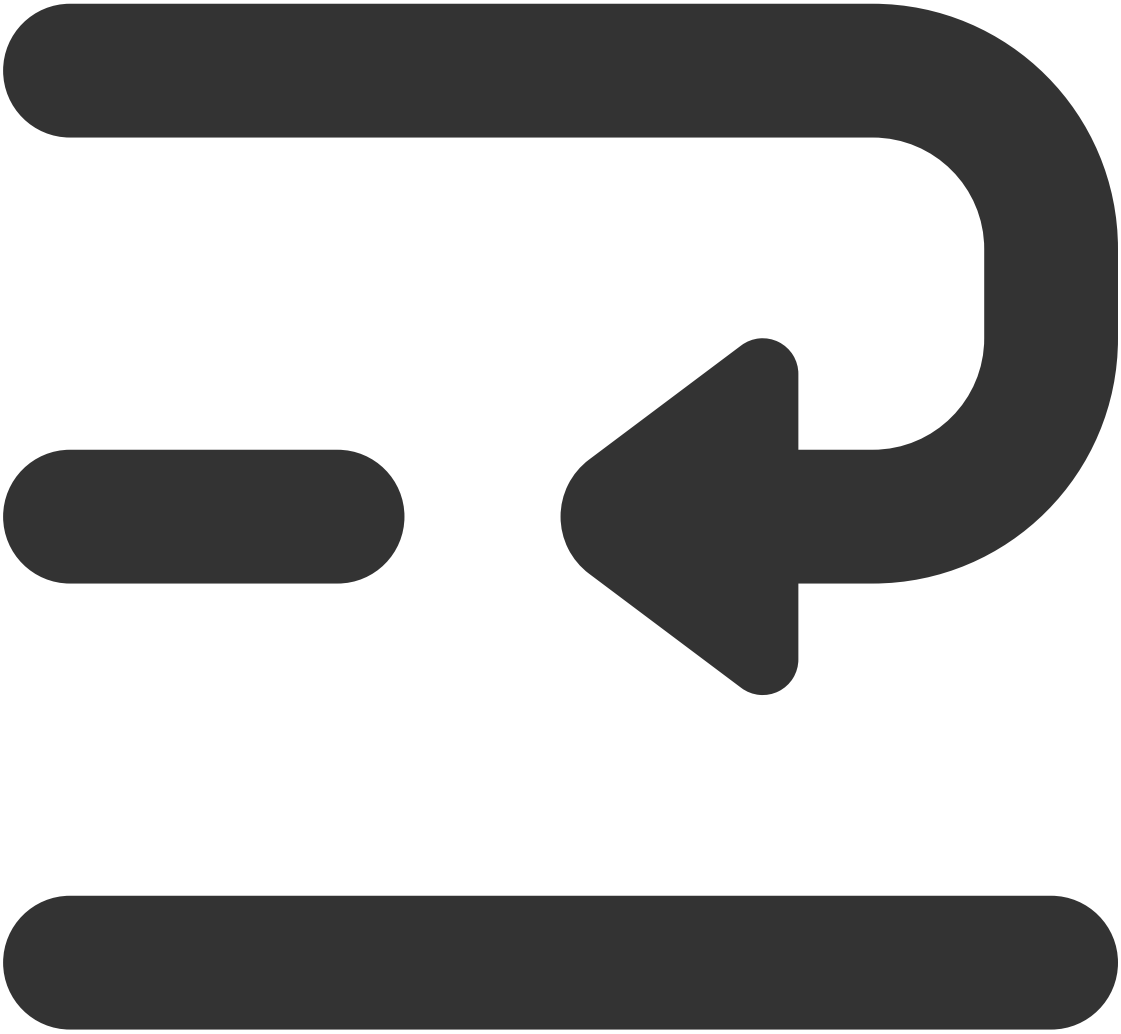


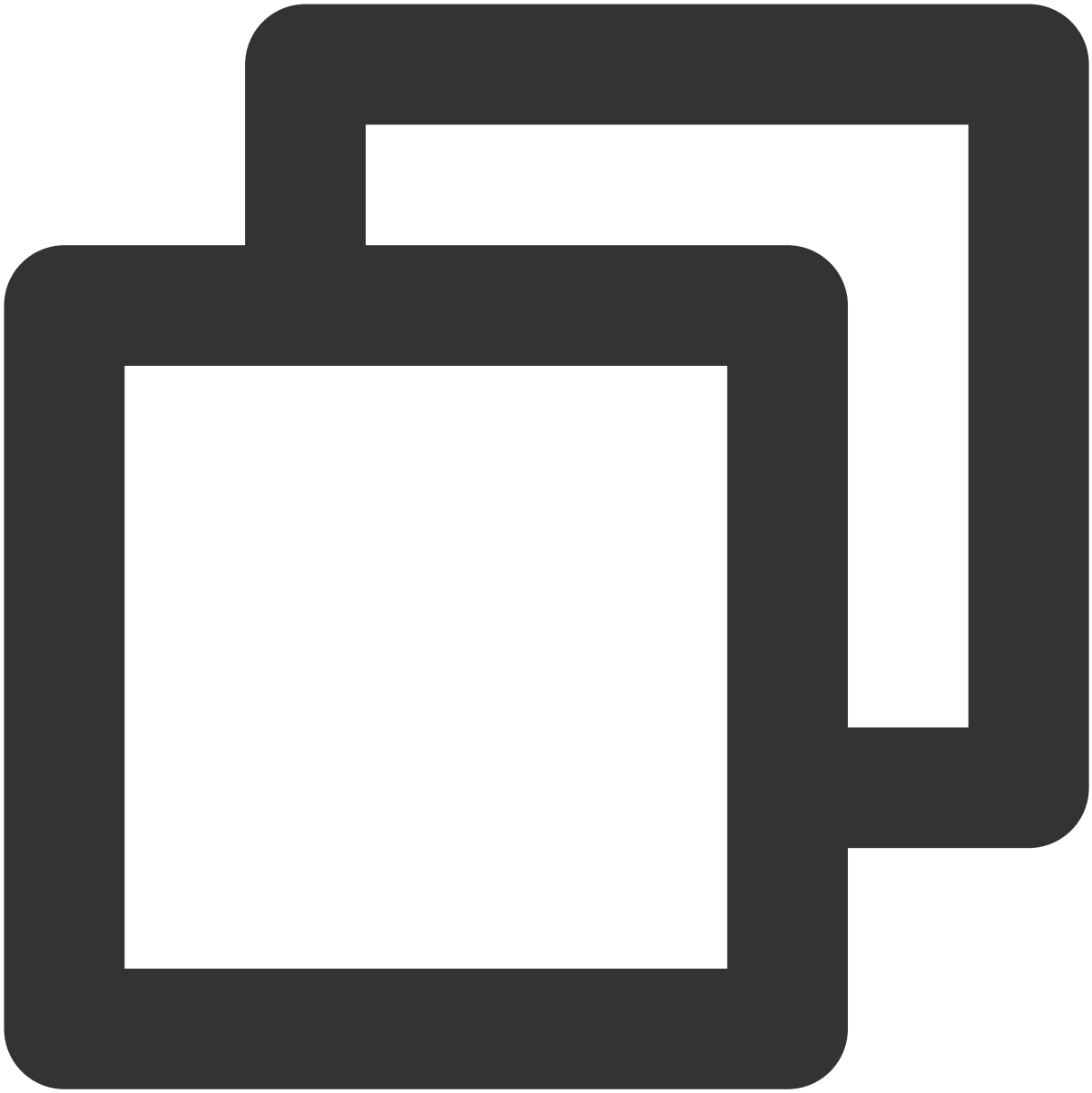
```
// 获取Client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

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

参数	说明
topicName	主题名称，在控制台集群管理中 Topic 页签中复制。
namespace	命名空间名称，在控制台集群管理中 Namespace 页签中复制。

	
endpoint	集群 HTTP 协议接入地址，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。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

发送顺序消息





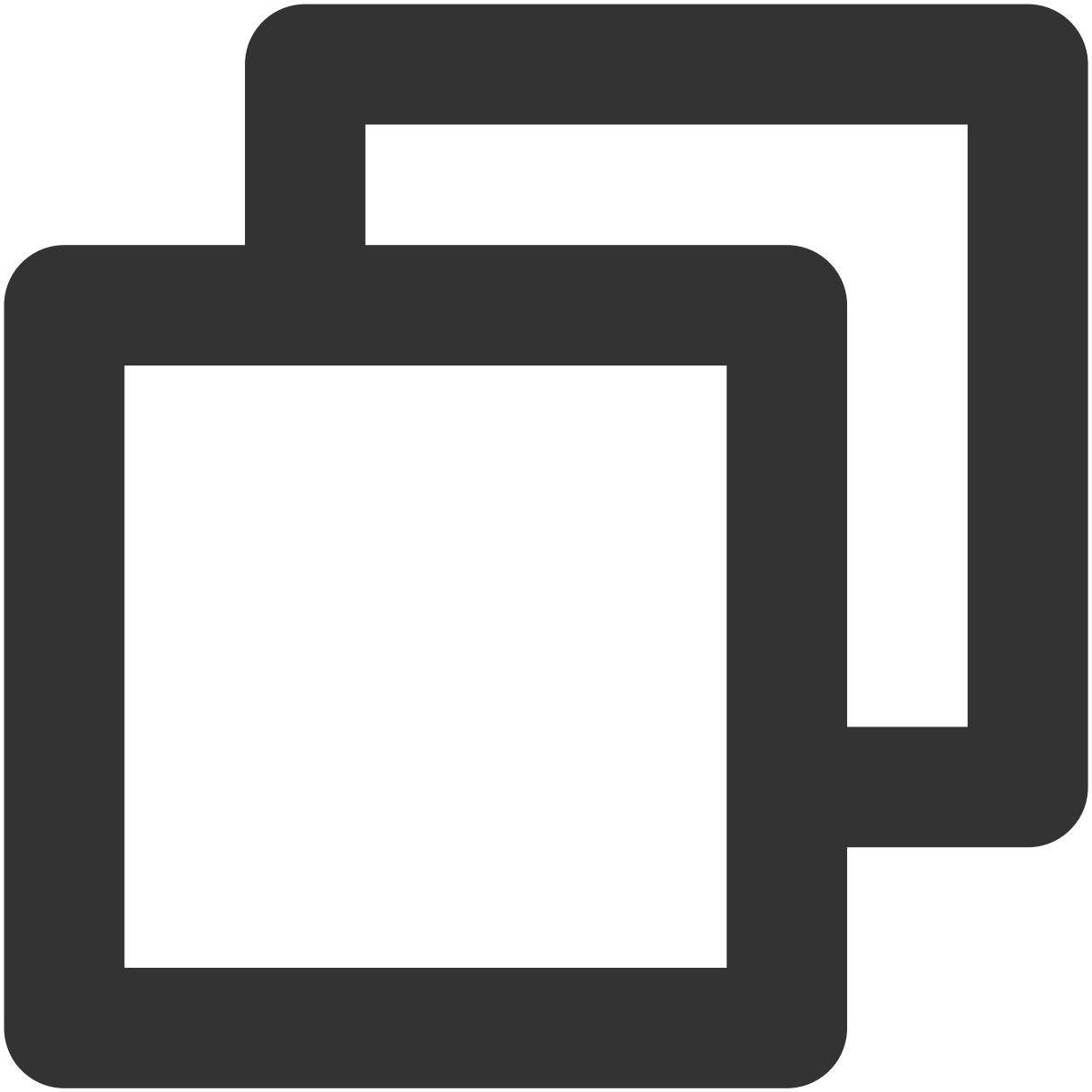
```
try {
    for (int i = 0; i < 10; i++) {
        TopicMessage pubMsg;
        pubMsg = new TopicMessage(
            ("Hello RocketMQ " + i).getBytes(),
            "TAG"
        );
        // 设置分区顺序消息的 ShardingKey
        pubMsg.setShardingKey(i % 3);
        TopicMessage pubResultMsg = producer.publishMessage(pubMsg);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

```
    }
} 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。

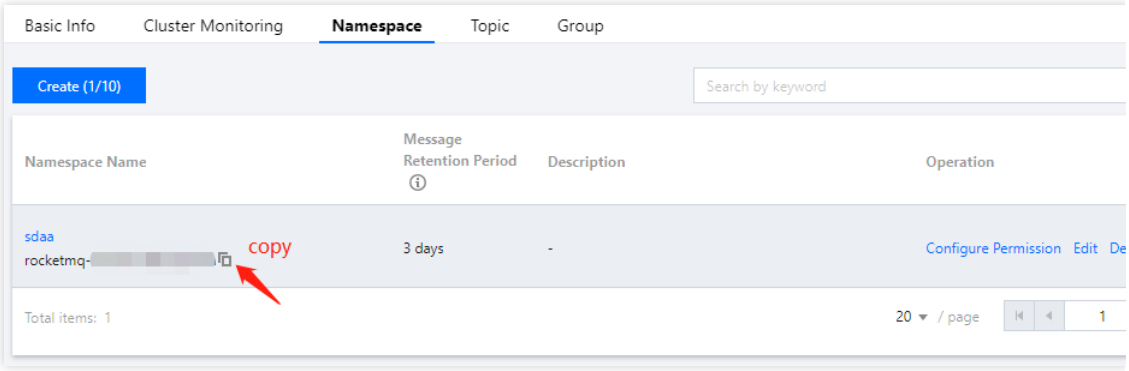
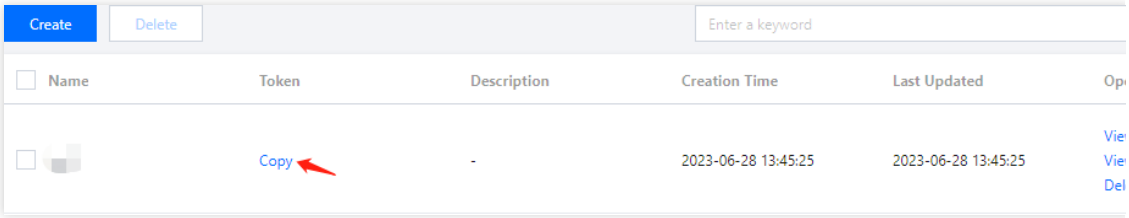
步骤4：消费消息

创建消费者

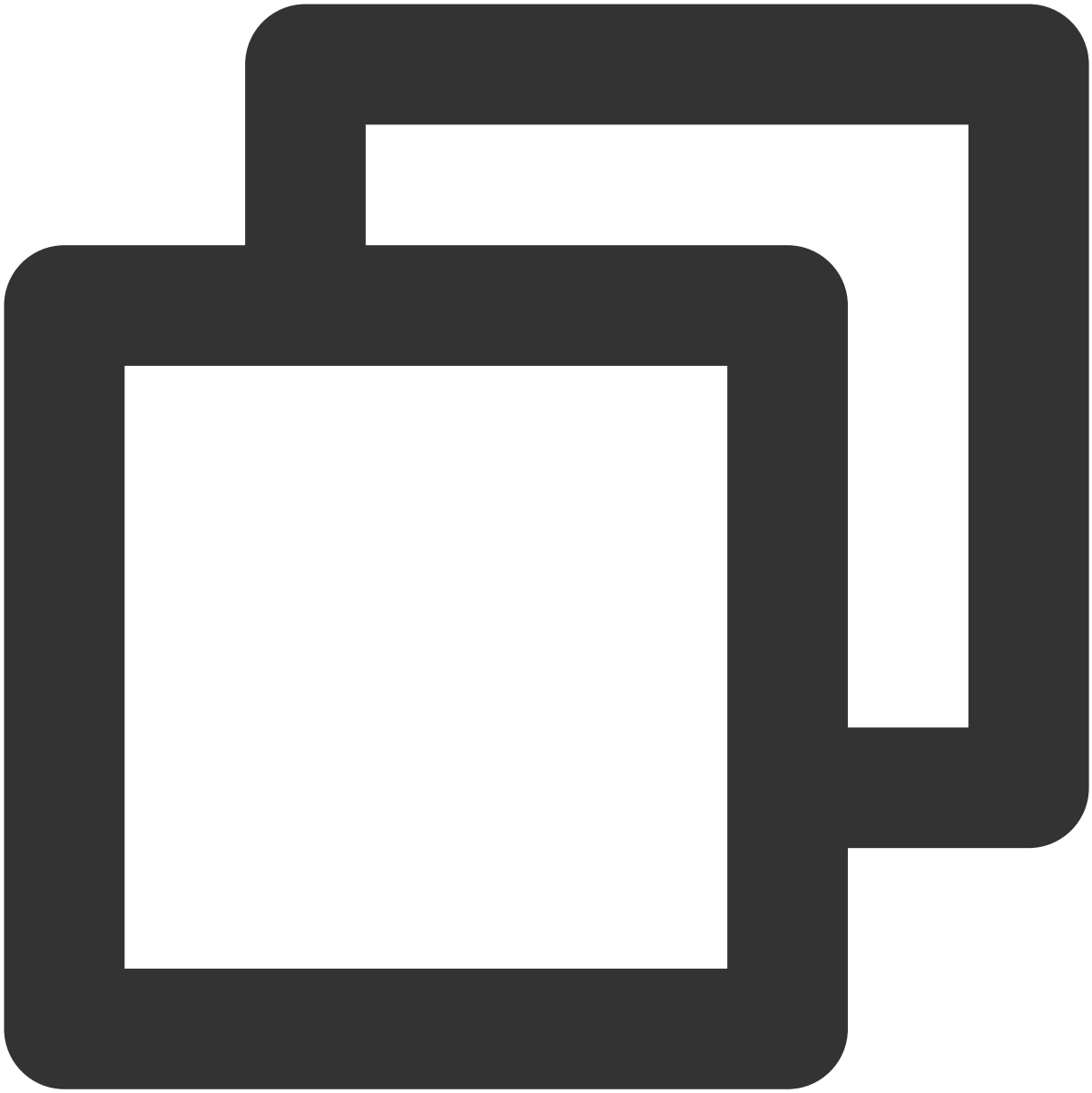


```
// 获取Client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

// 获取Topic的Consumer
MQProducer consumer = mqClient.getConsumer(namespace, topicName, groupName, "TAG
```

参数	说明
topicName	主题名称，在控制台集群管理中 Topic 页签中复制。
groupName	生产者组名称，在控制台集群管理中 Group 页签中复制。

namespace	命名空间名称，在控制台集群管理中 Namespace 页签中复制。 
TAG	订阅的标签。
endpoint	集群 HTTP 协议接入地址，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。
secretKey	角色名称，在 角色管理 页面复制。
accessKey	角色密钥，在 角色管理 页面复制密钥列复制。 

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
        // 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
        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, co
        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.getRe
                if (errors.getErrorMessages() != null) {
                    for (String errorHandle :errors.getErrorMessages().keySet()) {
                        System.out.println("Handle:" + errorHandle + ", ErrorCode:"
                                + ", ErrorMsg:" + errors.getErrorMessages().get(err
                    }
                }
                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。