消息队列 Pulsar 版

产品文档



腾讯云TCE

目录

消息队列 Pulsar 版 4
- 产品简介 4
  - 产品概述 4
  - 产品优势 5
  - 应用场景 6
  - 功能特性 9
    - 订阅模式 9
    - 定时和延时消息 11
    - 消息重试和死信机制 14
    - 消息标签过滤 18
  - 使用限制 21
- 快速入门 23
  - 资源创建与准备 23
  - 下载并运行Demo 25
- 操作指南 30
  - 集群管理 30
  - 命名空间 33
  - Topic 管理 35
  - 订阅管理 39
  - VPC 接入 41
  - JWT 鉴权配置 43
  - 消息查询与轨迹 44
  - 角色与鉴权 48
  - 配置告警 52
- 最佳实践 54
  - 使用 TLS 传输加密 54
  - 消息幂等性 59
  - 消息压缩 62
- 通用参考 66
  - 消息类型 66
  - 重试队列和死信队列 67
- 常见问题 68
  - 概念相关 68
- SDK文档 69
  - SDK概览 69
  - SDK 升级说明 70
  - Go SDK 73
  - TCP协议（Pulsar社区版） 80
    - Spring Boot Starter 接入 80
    - Java SDK 85
    - Go SDK 89
    - C++ SDK 93
    - Python SDK 97
    - Node.js SDK 101
- 开发指南 102
  - 原理解析 102
    - Pulsar Topic和分区 102
    - 消息副本与存储机制 106
    - 消息存储原理与ID规则 111
  - 使用实践 114
    - 订阅模式 114
    - 定时和延时消息 123
    - 消息标签过滤 126
    - 消息重试与死信机制 132
    - 客户端连接与生产消费者 137
- API文档 143
  - 分布式消息队列（tdmq） 143
    - 版本（2020-02-17） 143
      - API 概览 143
      - 调用方式 146
        - 接口签名v1 146
        - 接口签名v3 153
        - 请求结构 162
        - 返回结果 163
        - 公共参数 166
      - CMQ管理相关接口 168
        - 回溯cmq队列 168
      - 主题相关接口 170
        - 新增主题 170
        - 删除主题 173
        - 查询主题列表 175
        - 修改主题 178
      - 其他接口 180
        - 删除角色 180
        - 获取租户VPC绑定关系 182
        - 发送消息 184
      - 命名空间相关接口 186
        - 创建命名空间 186
        - 删除命名空间 189
        - 获取命名空间属性 191
        - 获取命名空间角色列表 194
        - 获取命名空间列表 196
        - 修改命名空间属性 198
      - 消息相关接口 200
        - 确认消息 200
        - 接收消息 202
        - 批量发送消息 204
        - 发送单条消息 207
      - 环境角色授权相关接口 209

- 创建环境角色授权 209
- 创建角色 211
- 删除环境角色授权 213
- 获取角色列表 215
- 修改环境角色授权 217
- 角色修改 219
- 修改资源对应的项目（包含新增） 221

生产消费相关接口 223
- 创建订阅关系 223
- 删除订阅关系 226
- 消费订阅列表 228
- 消息回溯 231

集群相关接口 233
- 创建集群 233
- 删除集群 235
- 获取专享集群列表 237
- 获取集群详情 239
- 获取集群列表 241
- 更新集群信息 243

数据结构 245

错误码 268

产品简介

产品概述

消息队列 Pulsar 版（TDMQ for Apache Pulsar，简称 TDMQ Pulsar 版）是一款基于 Apache 顶级开源项目 Pulsar 的金融级分布式消息中间件，其计算与存储分离的架构设计，使得它具备极好的云原生和 Serverless 特性，用户按量使用，无需关心底层资源。
TDMQ Pulsar 版拥有原生 Java 、 C++、Python、GO 等多种 SDK，可为分布式应用系统提供异步解耦和削峰填谷的能力，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。
TDMQ Pulsar版 目前已应用在计费绝大部分场景，包括支付主路径、实时对账、实时监控、大数据实时分析等方面。

主要特性

- 具备高一致、高可靠、高并发特性
- 采用服务和存储分离架构，支持水平动态扩容
- 支持百万级消息主题
- 非常低的消息发布和端到端的延迟
- 支持多种订阅模式: 独占（exclusive）、共享（shared）、灾备（failover）
- 一个 Serverless 的轻量级计算框架 Functions 提供了原生的流数据处理
- 支持多集群，能够无缝的基于地理位置进行跨集群的备份

产品优势

数据强一致

TDMQ Pulsar 版采用 BookKeeper 一致性协议 实现数据强一致性（类似 RAFT 算法），将消息数据备份写到不同物理机上，并且要求是同步刷盘。当某台物理机出故障时，后台数据复制机制能够对数据快速迁移，保证用户数据备份可用。

高性能低延迟

TDMQ Pulsar 版能够高效支持百万级消息生产和消费，海量消息堆积且消息堆积容量不设上限，支撑了计费所有场景；性能方面，单集群 QPS 超过10万，同时在时耗方面有保护机制来保证低延迟，帮助您轻松满足业务性能需求。

百万级 Topic

TDMQ Pulsar 版计算与存储架构的分离设计，使得 TDMQ Pulsar 版可以轻松支持百万级消息主题。相比于市场上其他 MQ 产品，整个集群不会因为 Topic 数量增加而导致性能急剧下降。

丰富的消息类型

TDMQ Pulsar 版提供丰富的消息类型，涵盖普通消息、顺序消息（全局顺序 / 分区顺序）、定时消息，满足各种严苛场景下的高级特性需求。

消费者数量无限制

不同于 Kafka 的消息消费模式，TDMQ Pulsar 版的消费者数量不受限于 Topic 的分区个数，并且会按照一定的算法均衡每个消费者的消息量，业务可按需启动对应的消费者数量。

多协议接入

TDMQ Pulsar 版的 API 支持 Java、C++、Go 等多语言，并且支持 HTTP 协议，可扩展更多语言的接入，另外还支持开源RocketMQ、RabbitMQ客户端的接入。如果用户只是利用消息队列的基础功能进行消息的生产和消费，可以不用修改代码就完成到 TDMQ Pulsar 版的迁移。

隔离控制

提供按租户对 Topic 进行隔离的机制，同时可精确管控各个租户的生产和消费速率，保证租户之间互不影响，消息的处理不会出现资源竞争的现象。

应用场景

异步解耦

交易引擎作为计费最核心的系统，每笔交易订单数据需要被几十个下游业务系统关注，包括物品批价、道具发货、积分、流计算分析等，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，消息队列 TDMQ Pulsar 版可实现高效的异步通信和应用解耦，确保主站业务的连续性。



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。消息队列 TDMQ Pulsar 版此时便可以承担一个缓冲器的角色，将上游突增的请求集中收集，下游可以根据自己的实际处理能力来消费请求消息。



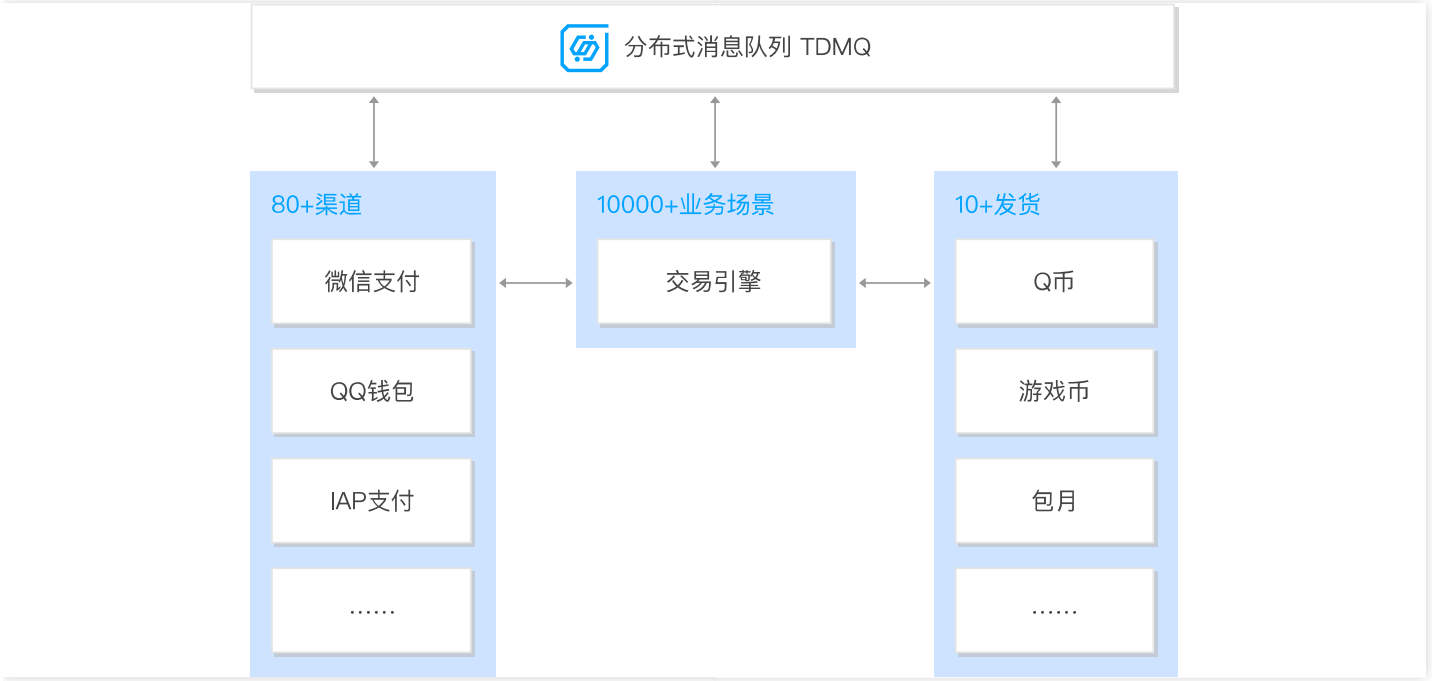
顺序收发

顺序消息的应用出现在业务场景中。例如王者荣耀的皮肤道具购买与发放，过程中的订单创建、支付、退款等流程都是严格按照顺序执行的，与先进先出（First In First Out，FIFO）原理类似，消息队列 TDMQ Pulsar 版提供一种专门应对这种情形的顺序消息功能，即保证消息 FIFO。



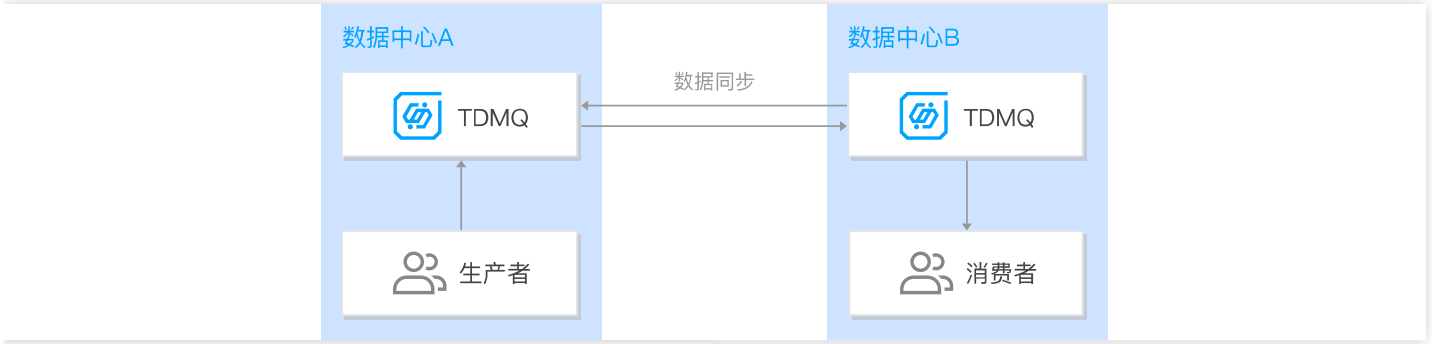
分布式事务一致性

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 Pulsar 版的自动重推和海量堆积能力来实现事物补偿，以及支付 Tips 通知和交易流水推送可以通过 TDMQ Pulsar 版来实现最终一致性。



数据同步

如果有多个数据中心存在，需要在多个数据中心之间消费，那么 TDMQ Pulsar 版可以非常方便实现数据中心之间的同步。



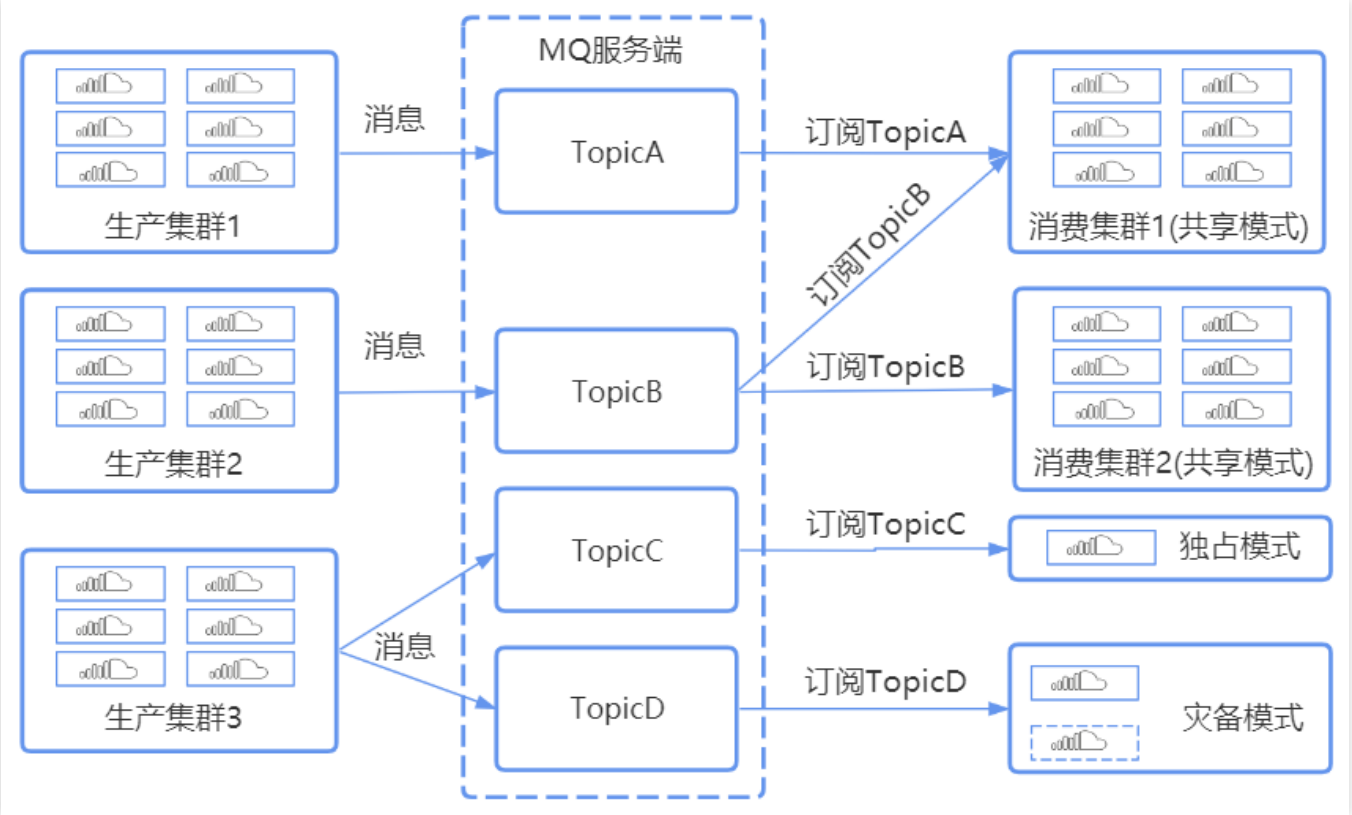
大数据分析

数据在"流动"中产生价值，传统数据分析大多是基于批量计算模型，而无法做到实时的数据分析，利用 TDMQ Pulsar 版与流式计算引擎相结合，可以很方便地实现业务数据的实时分析。

功能特性
订阅模式

为了适用不同场景的需求， TDMQ Pulsar 版提供多种订阅方式。订阅可以灵活组合出很多可能性：

- 如果您想实现传统的 “发布-订阅消息”形式 ，可以让每个消费者都有一个唯一的订阅名称（独占）。
- 如果您想实现传统的“消息队列” 形式，可以使多个消费者使用同一个的订阅名称（共享、灾备）。
- 如果您想同时实现以上两点，可以让一些消费者使用独占方式，剩余消费者使用其他方式。



独占模式（Exclusive）

如果两个及以上的消费者尝试以同样方式去订阅主题，消费者将会收到错误，适用于全局有序消费的场景。

```
 Consumer<byte[]> consumer1 = client.newConsumer()
            .subscriptionType(SubscriptionType.Exclusive)
            .topic(topic)
            .subscriptionName(groupName)
            .subscribe();
//consumer1启动成功
 Consumer<byte[]> consumer2 = client.newConsumer()
            .subscriptionType(SubscriptionType.Exclusive)
            .topic(topic)
```

```
            .subscriptionName(groupName)
            .subscribe();
    //consumer2启动失败
```

灾备模式（Failover）

consumer 将会按字典顺序排序，第一个 consumer 被初始化为唯一接受消息的消费者。

```
    Consumer<byte[]> consumer1 = client.newConsumer()
            .subscriptionType(SubscriptionType.Failover)
            .topic(topic)
            .subscriptionName(groupName)
            .subscribe();
    //consumer1启动成功
    Consumer<byte[]> consumer2 = client.newConsumer()
            .subscriptionType(SubscriptionType.Failover)
            .topic(topic)
            .subscriptionName(groupName)
            .subscribe();
    //consumer2启动成功
```

当 master consumer 断开时，所有的消息（未被确认和后续进入的）将会被分发给队列中的下一个 consumer。

共享模式（Shared）

消息通过 round robin 轮询机制（也可以自定义）分发给不同的消费者，并且每个消息仅会被分发给一个消费者。当消费者断开连接，所有被发送给他，但没有被确认的消息将被重新安排，分发给其它存活的消费者。

```
    Consumer<byte[]> consumer = client.newConsumer()
            .subscriptionType(SubscriptionType.Shared)
            .topic(topic)
            .subscriptionName(groupName)
            .subscribe();
```

定时和延时消息

相关概念

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

适用场景

如果系统是一个单体架构，则通过业务代码自己实现延时或利用第三方组件实现基本没有差别；一旦架构复杂起来，形成了一个大型分布式系统，有几十上百个微服务，这时通过应用自己实现定时逻辑会带来各种问题。一旦运行着延时程序的某个节点出现问题，整个延时的逻辑都会受到影响。
针对以上问题，利用延时消息的特性投递到消息队列里，便是一个较好的解决方案，能统一计算延时时间，同时重试和死信机制确保消息不丢失。
具体场景的示例如下：

- 微信红包发出后，生产端发送一条延时24小时的消息，到了24小时消费端程序收到消息，进行用户是否已经领走红包的判断，如果没有则退还到原账户。
- 小程序下单某商品后，后台存放一条延时30分钟的消息，到时间之后消费端收到消息触发对支付结果的判断，如果没有支付就取消订单，这样就实现了超过30分钟未完成支付就取消订单的逻辑。
- 微信上用户将某条信息设置待办后，也可以通过发送一条定时消息，服务端到点收到这条定时消息，对用户进行待办项提醒。

使用方式

在 TDMQ Pulsar 版的 SDK 中提供了专门的 API 来实现定时消息和延时消息。

- 对于定时消息，您需要提供一个消息发送的时刻。
- 对于延时消息，您需要提供一个时间长度作为延时的时长。

定时消息

定时消息通过生产者 producer 的 deliverAt() 方法实现，代码示例如下：

```
String value = "message content";
```

```
try {
      //需要先将显式的时间转换为 Timestamp
    long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2020-11-11 00:00:0
0").getTime();
    //通过调用 producer 的 deliverAt 方法来实现定时消息
      MessageId msgId = producer.newMessage()
            .value(value.getBytes())
            .deliverAt(timeStamp)
            .send();
} catch (ParseException e) {
      //TODO 添加对 Timestamp 解析失败的处理方法
      e.printStackTrace();
}
```

> 注意：
>
> 定时消息的时间范围为当前时间开始计算，864000秒（10天）以内的任意时刻。如10月1日12:00开始，最长可以设置到10月11日12:00。

延时消息

延时消息通过生产者 produce 的 deliverAfter() 方法实现，代码示例如下：

```
String value = "message content";

//需要指定延时的时长
long delayTime = 10L;
//通过调用 producer 的 deliverAfter 方法来实现定时消息
MessageId msgId = producer.newMessage()
   .value(value.getBytes())
   .deliverAfter(delayTime, TimeUnit.SECONDS) //单位可以自由选择
   .send();
```

> 注意：
>
> 延时消息的时长取值范围为0 - 864000秒（0秒 - 10天）。如10月1日12:00开始，最长可以设置864000秒。如果设置的时间超过这个时间，则直接按864000秒计算，到时会直接投递。

使用说明和限制

- 使用定时和延时两种类型的消息时，请确保客户端的机器时钟和服务端的机器时钟保持一致，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。
- 定时和延时消息单个 Topic 下同时存在的最大数量为10万条，超过这个并发数服务端会限制生产端继续生产这两种消息。为预防此类现象，请留意控制台上关于定时和延时消息指标的监控。

- 关于定时和延时消息的时间范围，最大均为10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。
- 设定定时时间后，从定时的时间点开始计算消息最长保留时间，例如定时到3天后发送，消息最长保留7天，则到了第10天仍未被消费时，消息会被删除。延时消息同理。

消息重试和死信机制

重试 Topic 是一种为了确保消息被正常消费而设计的 Topic 。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 Topic 中，当重试达到一定次数后，停止重试，投递到死信 Topic 中。
当消息进入到死信队列中，表示 TDMQ Pulsar 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。

自动重试

相关概念

重试 Topic：一个重试 Topic 是对应一个订阅名（一个订阅者组的唯一标识）的，以 Topic 形式存在于 TDMQ Pulsar 版中。当您新建了一个订阅后，会自动创建一个 “[订阅名]-retry” 的 Topic，该 Topic 会自主实现消息重试的机制。

实现原理

您创建的消费者使用某个订阅名以共享模式订阅了一个 Topic 后，如果开启了 enableRetry 属性，就会自动订阅这个订阅名对应的重试队列。

> 说明：
>
> 仅共享模式支持自动化重试和死信机制，独占和灾备模式不支持。

这里以 Java 语言客户端为例，在 topic1 创建了一个 sub1 的订阅，客户端使用 sub1 订阅名订阅了 topic1 并开启了 enableRetry ，如下所示：

```
Consumer consumer = client.newConsumer()
    .topic("persistent://1******30/default/topic1")
    .subscriptionType(SubscriptionType.Shared)//仅共享消费模式支持重试和死信
    .enableRetry(true)
    .subscriptionName("sub1")
    .subscribe();
```

此时， topic1 对 sub1 的订阅就形成了带有重试机制的投递模式， sub1 会自动订阅之前在新建订阅时自动创建的 sub1-retry （可以在控制台 Topic 列表中找到）。当 topic1 中的消息投递第一次未收到消费端 ACK 时，这条消息就会被自动投递到重试 Topic sub1-retry ，并且由于 consumer 自动订阅了这个主题，后续这条消息会在一定的 重试规则下重新被消费。当达到最大重试次数后仍失败，消息会被投递到对应的死信队列 sub1-dlq ，等待人工处理。

自定义参数设置

TDMQ Pulsar 版会默认配置一套重试和死信参数，具体如下：

- 指定重试次数为16次（失败16次后，第17次会投递到死信队列）
- 指定重试队列为 [订阅名]-retry
- 指定死信队列为 [订阅名]-dlq
  如果希望自定义配置这些参数，可以使用 deadLetterPolicy API 进行配置，代码如下：

```
Consumer<byte[]> consumer = pulsarClient.newConsumer()
    .topic(topic)
    .subscriptionName("sub1")
    .subscriptionType(SubscriptionType.Shared)
    .enableRetry(true)//开启重试消费
    .deadLetterPolicy(DeadLetterPolicy.builder()
        .maxRedeliverCount(maxRedeliveryCount)//可以指定最大重试次数
        .retryLetterTopic("persistent://my-property/my-ns/sub1-retry")//可以指定重试队列
        .deadLetterTopic("persistent://my-property/my-ns/sub1-dlq")//可以指定死信队列
        .build())
    .subscribe();
```

重试规则

重试规则由 reconsumerLater API 实现，有三种模式：

```
//指定任意延迟时间
consumer.reconsumeLater(msg, 1000L, TimeUnit.MILLISECONDS);
//指定延迟等级
consumer.reconsumeLater(msg, 1);
//等级递增
consumer.reconsumeLater(msg);
```

- 第一种：指定任意延迟时间。第二个参数填写延迟时间，第三个参数指定时间单位。延迟时间和延时消息的取值范围一致，范围在1 - 864000（单位：秒）。
- 第二种：指定任意延迟等级。实现效果和第一种基本一致，更方便统一管理分布式系统中的延时时长，延迟等级说明如下：
  i. reconsumeLater(msg, 1) 中的第二个参数即为消息等级
  ii. 默认 MESSAGE_DELAYLEVEL = "1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h" ，这个常数决定了每级对应的延时时间，例如1级对应1s，3级对应10s。如果默认值不符合实际业务需求，用户可以重新自定义。
- 第三种：等级递增。实现的效果不同于以上两种，为退避式的重试，即第一次失败后重试间隔为1秒，第二次失败后重试间隔为5秒，以此类推，次数越多，间隔时间越长。具体时间间隔同样由第二种中介绍的 MESSAGE_DELAYLEVEL 决定。
  这种重试机制往往在业务场景中有更实际的应用，如果消费失败，一般服务不会立刻恢复，使用这种渐进式的重试方式更为合理。

重试消息的消息属性

一条重试消息会给消息带上如下 property。

```
{
  REAL_TOPIC="persistent://my-property/my-ns/test,
  ORIGIN_MESSAGE_ID=314:28:-1,
  RETRY_TOPIC=persistent://my-property/my-ns/my-subscription-retry,
  RECONSUMETIMES=16
}
```

- REAL_TOPIC ：原 Topic
- ORIGIN_MESSAGE_ID ：最初生产的消息 ID
- RETRY_TOPIC ：重试 Topic
- RECONSUMETIMES ：代表该消息重试的次数

重试消息的消息 ID 流转

消息 ID 流转过程如下所示，您可以借助此规则对相关日志进行分析。

```
原始消费： msgid=1:1:0:1
第一次重试： msgid=2:1:-1
第二次重试： msgid=2:2:-1
第三次重试： msgid=2:3:-1
.....
第16次重试： msgid=2:16:0:1
第17次写入死信队列： msgid=3:1:-1
```

完整代码示例

以下为借助 TDMQ Pulsar 版实现完整消息重试机制的代码示例，供开发者参考。

订阅主题

```
Consumer<byte[]> consumer1 = client.newConsumer()
        .topic("topic")
        .subscriptionName("my-subscription")
        .subscriptionType(SubscriptionType.Shared)
        .enableRetry(true)//开启重试消费
        //.deadLetterPolicy(DeadLetterPolicy.builder()
        //        .maxRedeliverCount(maxRedeliveryCount)
        //        .retryLetterTopic("persistent://my-property/my-ns/my-subscription-retry")//可以指定重试队
列
        //        .deadLetterTopic("persistent://my-property/my-ns/my-subscription-dlq")//可以指定死信队
列
        //        .build())
```

```
        .subscribe();
```

执行消费

```
while (true) {
    Message msg = consumer.receive();
    try {
        // Do something with the message
        System.out.printf("Message received: %s", new String(msg.getData()));
        // Acknowledge the message so that it can be deleted by the message broker
        consumer.acknowledge(msg);
    } catch (Exception e) {
        // select reconsume policy
        consumer.reconsumeLater(msg, 1000L, TimeUnit.MILLISECONDS);
        //consumer.reconsumeLater(msg, 1);
        //consumer.reconsumeLater(msg);
    }
}
```

主动重试

当消费者在某个时间没有成功消费某条消息，如果想重新消费到这条消息时，消费者可以发送一条取消确认消息到 TDMQ Pulsar 版服务端，TDMQ Pulsar 版 会将这条消息重新发给消费者。 这种方式重试时不会产生新的消息，所以也不能自定义重试间隔。以下为主动重试的 Java 代码示例：

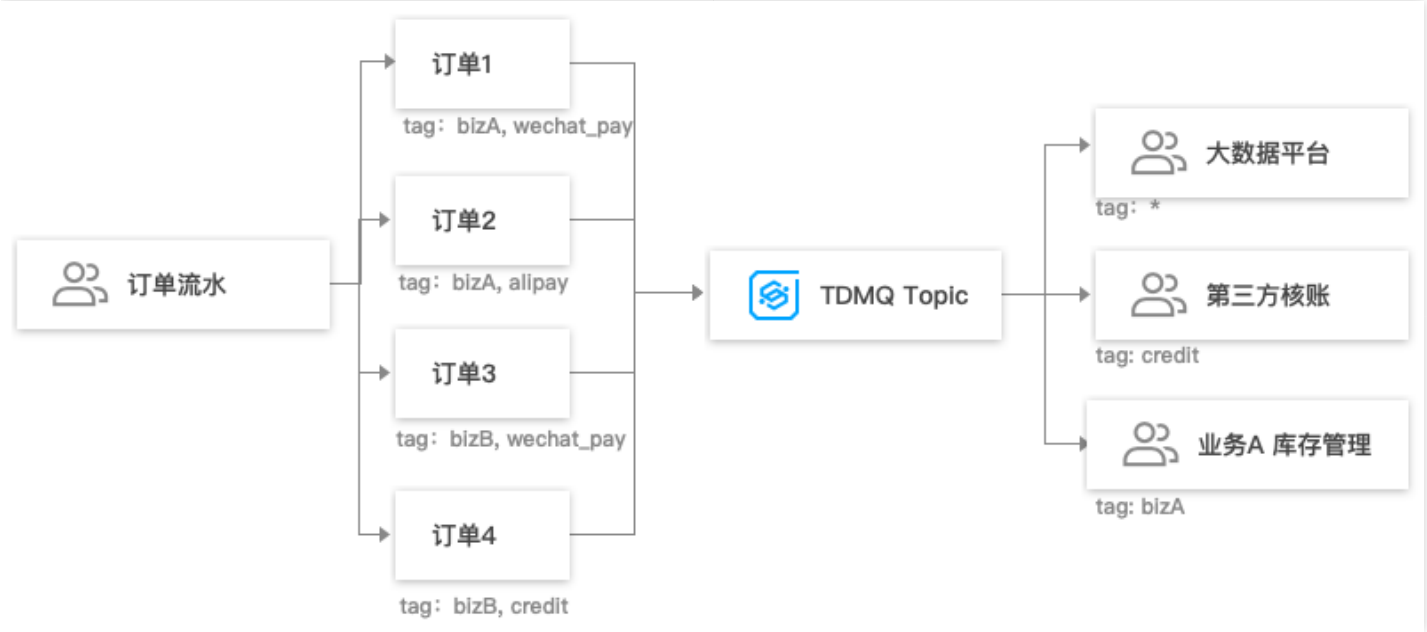
```
while (true) {
    Message msg = consumer.receive();
    try {
        // Do something with the message
        System.out.printf("Message received: %s", new String(msg.getData()));
        // Acknowledge the message so that it can be deleted by the message broker
        consumer.acknowledge(msg);
    } catch (Exception e) {
        // Message failed to process, redeliver later
        consumer.negativeAcknowledge(msg);
    }
    consumer.negativeAcknowledge(msg);
}
```

消息标签过滤

> 说明：
>
> 为了更好地和 Pulsar 开源社区统一，自2021年4月30日起，云平台版 SDK 将停止功能更新，TDMQ Pulsar 版推荐您使用社区版本的 SDK，该功能仅限存量客户使用。

场景说明

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 Pulsar 版提供 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。



> 说明：
>
> 消息标签过滤的功能仅限云平台版 SDK 使用，社区版 SDK 暂不支持。

生产带标签的消息

TDMQ Pulsar 版的消息构造函数中，声明了一个 tags 字段，开发者可以为消息添加一个由字符串组成的 tags，如下所示：

```
TypedMessageBuilder<T> tags(String... tags);
```

TDMQ Pulsar 版的客户端提供了专门为消息附加标签属性的 API，以下为 Java 代码示例：

```
producer.newMessage().value("my-message".getBytes())
    .tags("TagA", "TagB", "TagC")
    .send();
```

消费时按标签过滤

开发者在创建 TDMQ Pulsar 版的消费客户端时，可以通过以下三种方式，声明消费者接收消息时按照哪种规则去过滤带有标签的消息：

方式一：按特定格式指定标签的组合

按特定格式指定一个标签的组合时，接口声明如下：

```
ConsumerBuilder<T> topicByTag(String topicName, String tag);
```

代码示例如下：

```
consumer = client.newConsumer()
        .topicByTag(topic, "TagA || TagB")//指定多个 Tag
        .subscriptionName("my-subscription")
        .subscriptionType(SubscriptionType.Shared)
        .subscribe();
```

> 注意：
>
> 指定多个标签组合后，只要组合中有一个标签匹配，则该消息就会被投递到这个消费者。多个标签之间的过滤规则为“或”的关系。

方式二：使用正则表达式过滤标签

使用正则表达式过滤标签时，接口声明如下：

```
ConsumerBuilder<T> topicByTagsPattern(String topicName, String tagsPattern);
```

代码示例如下：

```
consumer = client.newConsumer()
        .topicByTagsPattern(topic, "Tag.*")//正则表达式
        .subscriptionName("my-subscription")
```

```
        .subscriptionType(SubscriptionType.Shared)
        .subscribe();
```

方式三：通过键值对批量指定标签

通过一个键值对批量指定标签过滤时，接口声明如下：

```
ConsumerBuilder<T> topicsAndTags(Map<String,String > topicsAndTags);
```

代码示例如下：

```
Map<String,String > map = new HashMap<>();
map.put(topic1, "TagB");
map.put(topic2, "TagA");
Consumer consumer = client.newConsumer()
    .topicsAndTags(map)
    .subscriptionName("my-subscription")
    .subscriptionType(SubscriptionType.Shared)
    .subscribe();
```

通过通配符实现不过滤

除此之外，还可以通过通配符 * 来接收所有消息，代码示例如下：

```
consumer = client.newConsumer()
        .topic(topic, "*") //订阅所有
        .subscriptionName("my-subscription")
        .subscriptionType(SubscriptionType.Shared)
        .subscribe();
```

使用限制

本文列举了消息队列 TDMQ Pulsar 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	限制说明
单地域内集群数量上限	5个
集群名称长度	不能超过64个字符
最大存储容量	100GB

命名空间

限制类型	限制说明
单集群内命名空间数量上限	100个
单命名空间 TPS 上限	10000
单命名空间带宽上限（生产 + 消费）	400Mbps

Topic

限制类型	限制说明
单集群内 Topic 数量上限	1000
Topic 名称长度	不能超过64个字符
单 Topic 最大生产者数量	1000
单 Topic 最大消费者数量	500

消息

限制类型	限制说明
消息最大保留时间	15天
消息最大延时	10天
消息大小	5MB

限制类型	限制说明
消费位点重置	15天
最大已接收未确认消息数量	5000条

快速入门
资源创建与准备

操作场景

该任务指导您通过 TDMQ Pulsar 版控制台创建集群和 Topic 等资源，了解运行一个客户端之前，控制台所需要进行的操作。

操作步骤

步骤一：新建集群并配置网络

1. 登录 TDMQ Pulsar 版控制台，进入【集群管理】页面，选择目标地域。
2. 单击【新建集群】，创建一个集群。
3. 单击操作栏的**配置接入点，**新建一个VPC网络接入点。



步骤二：创建命名空间

在 TDMQ Pulsar 版控制台的【命名空间】页面，选择地域和刚刚创建好的集群，单击【新建】，创建一个命名空间。



步骤三：创建角色并授权

1. 在 TDMQ Pulsar 版控制台的【角色管理】页面，选择地域和刚刚创建好的集群，单击【新建】进入新建角色页面。
2. 填写角色名称和说明，勾选刚刚创建好的命名空间，并为其分配生产消费权限。
3. 单击【保存】完成角色创建。



步骤四：创建 Topic 和订阅关系

1. 在【Topic管理】页面，选择目标地域、当前集群和命名空间，单击【新建】，创建一个Topic。
2. 单击操作列的【新增订阅】，为刚刚新建好的 Topic 创建一个订阅关系。
3. 单击操作列的【更多】>【查看订阅】，可看到刚刚创建好的订阅。



下载并运行Demo

操作场景

该任务指导您在购买 TDMQ Pulsar 版服务后，下载 Demo 并进行简单的测试，了解运行一个客户端的操作步骤。

前提条件

已购买云服务器。

操作步骤

1. 下载 Demo，并配置相关参数。
   添加 Maven 依赖
   按照 [Pulsar 官方文档] 添加 Maven 依赖。

```
<!-- in your <properties> block -->
<pulsar.version>2.7.1</pulsar.version>
<!-- in your <dependencies> block -->
<dependency>
  <groupId>org.apache.pulsar</groupId>
  <artifactId>pulsar-client</artifactId>
  <version>${pulsar.version}</version>
</dependency>
```

创建 Client

```
// 一个Pulsar client对应一个客户端链接

// 原则上一个进程一个client，尽量避免重复创建，消耗资源

PulsarClient client = PulsarClient.builder()
      //替换成集群接入地址，位于【集群管理】页面接入地址
      .serviceUrl("http://****")
      //替换成角色密钥，位于【角色管理】页面
      .authentication(AuthenticationFactory.token("eyJr****"))
      .build();

System.out.println(">> pulsar client created.");
```

- serviceUrl 即接入地址，可以在控制台【集群管理】接入点页面查看并复制。
- token 即角色的密钥，角色密钥可以在【角色管理】中复制。

> 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。

创建消费者进程

```
Consumer<byte[]> consumer = client.newConsumer()
        .topic("persistent://pulsar-****")//topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
        .subscriptionName("****")//需要现在控制台或者通过控制台API创建好一个订阅，此处填写订阅名
        .subscriptionType(SubscriptionType.Exclusive)//声明消费模式为exclusive（独占）模式
        .subscriptionInitialPosition(SubscriptionInitialPosition.Earliest)//配置从最早开始消费，否则可能会消费不到历史消息
        .subscribe();
    System.out.println(">> pulsar consumer created.");
```

- Topic 名称需要填入完整路径，即“ persistent://clusterid/namespace/Topic ”， clusterid/namespace/topic 的部分可以从控制台上【Topic管理】页面直接复制。



- subscriptionName需要写入订阅名，可在【消费管理】界面查看。
  创建生产者进程

```
Producer<byte[]> producer = client.newProducer()
        .topic("persistent://pulsar-****")//topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
        .create();
    System.out.println(">> pulsar producer created.");
```

Topic 名称需要填入完整路径，即“ persistent://clusterid/namespace/Topic ”，clusterid/namespace/topic 的部分可以从控制台上【Topic管理】页面直接复制。
生产消息

```
for (int i = 0; i < 1000; i++) {
        String value = "my-sync-message-" + i;
        MessageId msgId = producer.newMessage().value(value.getBytes()).send();//发送消息
        System.out.println("deliver msg " + msgId + ",value:" + value);
```

```
        }
        producer.close();//关闭生产进程
```

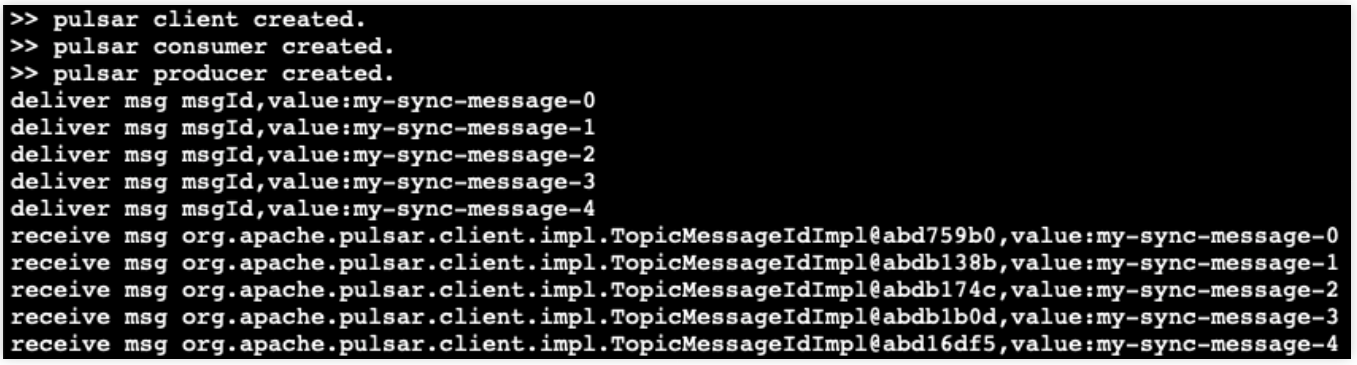
消费消息

```
for (int i = 0; i < 1000; i++) {
        Message<byte[]> msg = consumer.receive();//接收当前offset对应的一条消息
        String msgId = msg.getMessageId().toString();
        String value = new String(msg.getValue());
        System.out.println("receive msg " + msgId + ",value:" + value);
        consumer.acknowledge(msg);//接收到之后必须要ack，否则offset会一直停留在当前消息，无法继续消费
    }
```

2. 在 pom.xml 所在目录执行命令 mvn clean package ，或者通过IDE自带的功能打包整个工程，在target目录下生成一个可运行的jar文件。
3. 运行成功后将 jar 文件上传到云服务器。
4. 登录云服务器，进入到刚刚上传jar文件所在的目录，可看到文件已上传到云服务器。



5. 执行命令 java -jar tdmq-demo-1.0.0.jar，运行 Demo，可查看运行日志。

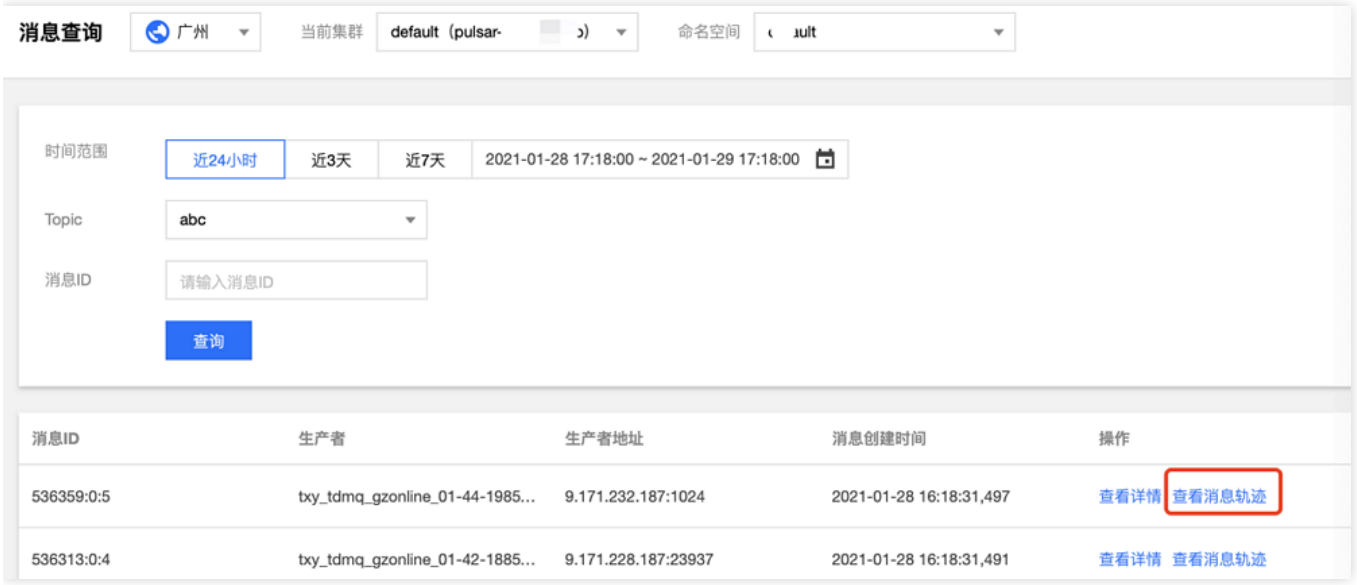


6. 登录 TDMQ Pulsar 版控制台，依次点击【Topic管理】>【Topic名称】进入消费管理页面，点开订阅名下方右三角号，可查看生产消费记录。

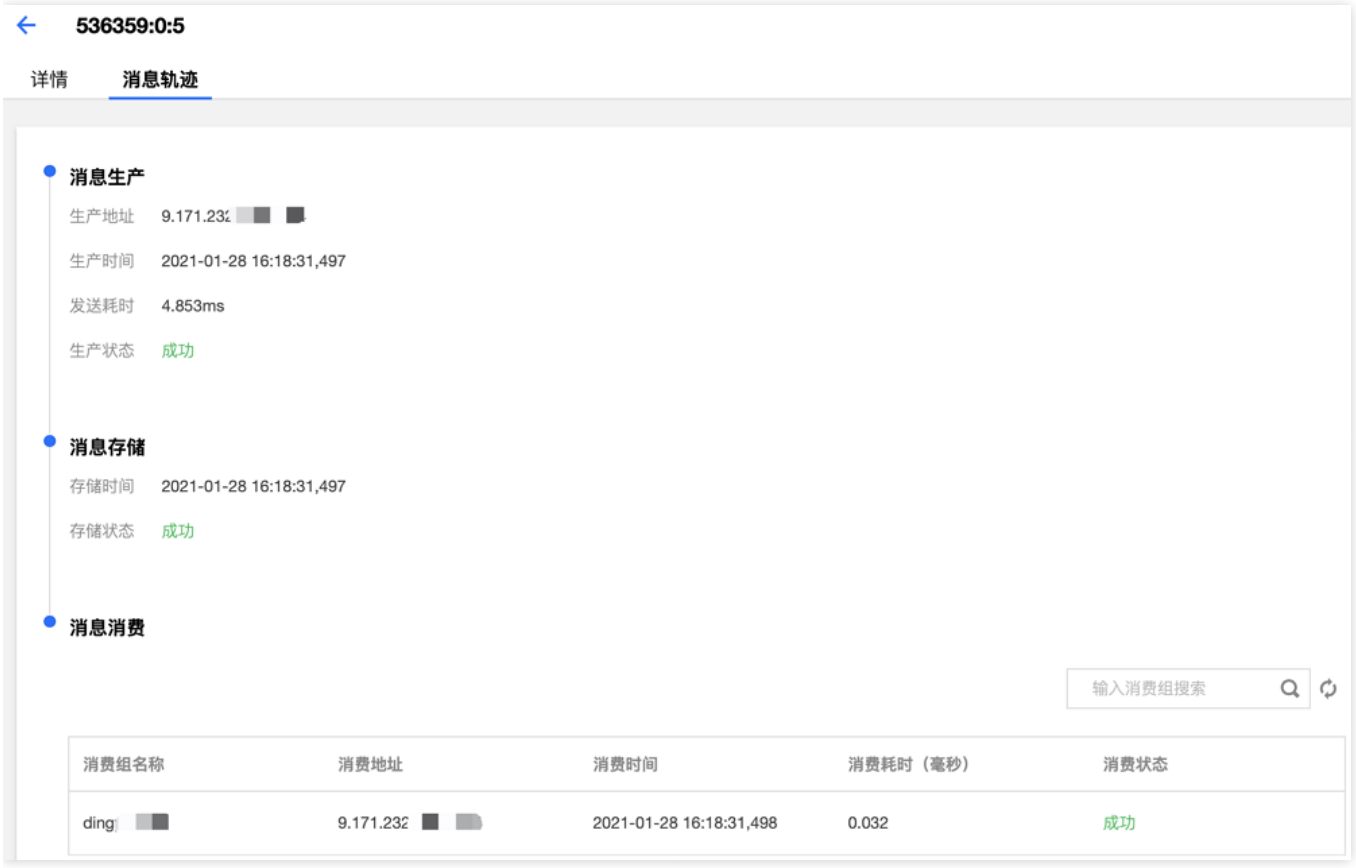
消费进度

分区ID	消息数量	已消费	消费速率(条/秒)	消费带宽(字节/秒)	进度差
0	0	0	0	0	0
1	1	1	0	0	0
2	1	1	0	0	0
3	2	2	0	0	0
4	0	0	0	0	0

7. 进入【消息查询】页面，可查看 Demo 运行后的消息轨迹。



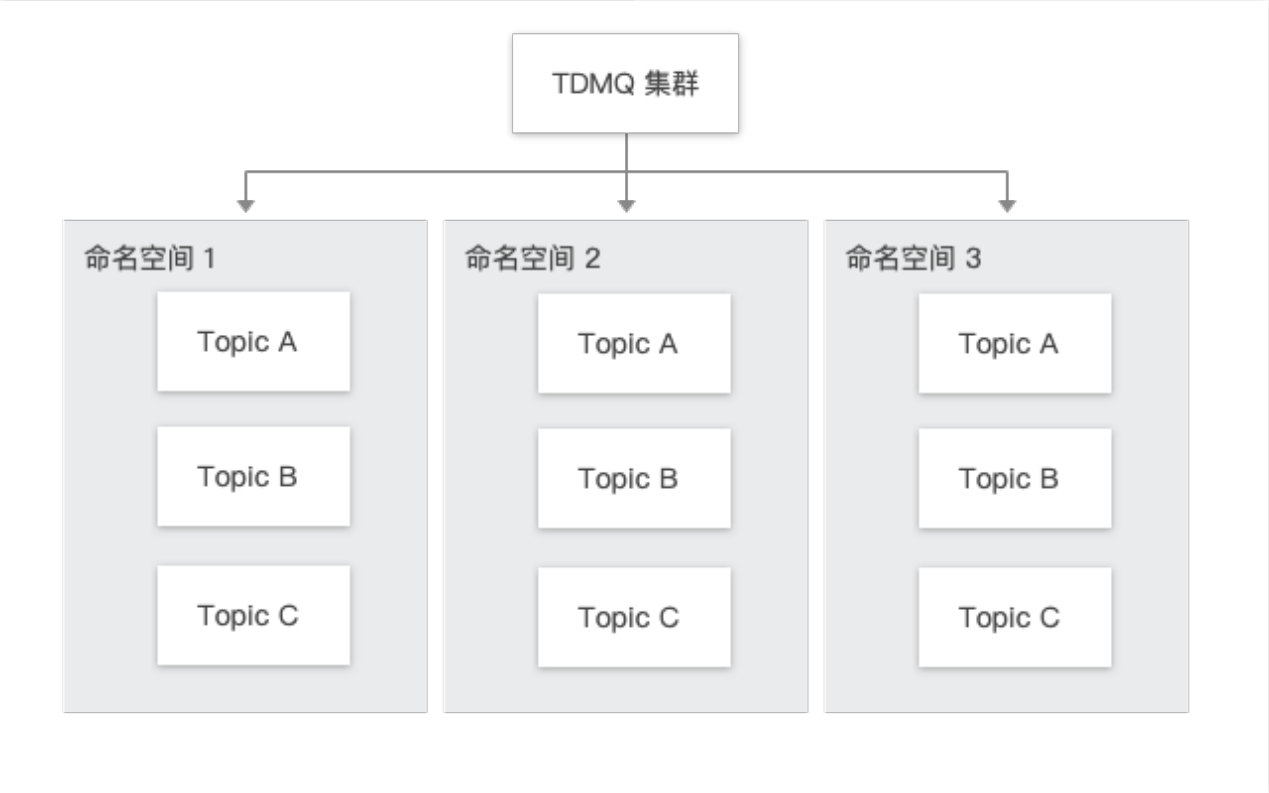
消息轨迹如下：


536359:0:5
详情
消息轨迹
消息生产
生产地址 9.171.23
生产时间 2021-01-28 16:18:31,497
发送耗时 4.853ms
生产状态 成功
消息存储
存储时间 2021-01-28 16:18:31,497
存储状态 成功
消息消费
输入消费组搜索
消费组名称
消费地址
消费时间
消费耗时（毫秒）
消费状态
ding
9.171.232
2021-01-28 16:18:31,498
0.032
成功

操作指南
集群管理

操作场景

集群是 TDMQ Pulsar 版中的一个资源维度，不同集群的命名空间、Topic、角色权限等完全隔离。每个集群会有集群的资源限制例如 Topic 总数、消息保留时长等。常见的使用方式如：开发测试环境使用一个专门集群，生产环境使用一个专门的集群。
TDMQ Pulsar 版资源层次关系



操作步骤

创建集群

1. 登录 TDMQ Pulsar 版控制台，进入【集群管理】页面。
2. 在【集群管理】页面，选择地域后，单击【新建集群】进入新建集群对话框。
3. 在新建集群对话框，选择需要创建的集群类型并设置集群的相关属性：
    - 集群名称：设置环境的名称（创建后不可修改），不能超过64个字符，只能包含字母、数字、“-”及“_”。
    - 说明：环境设置的备注说明，不能超过200个字符。

4. 单击【确定】完成所在地域环境的创建。
   后续步骤：
5. 获取集群接入地址。
6. 在集群中创建命名空间。
7. 在集群中创建角色，并授予该命名空间的生产消费权限。
8. 在命名空间中创建 Topic。
9. 编写 Demo，配置上链接信息和密钥，进行消息的生产和消费。

查看集群详情

在【集群管理】列表页，单击目标集群操作列的查看详情，进入集群详情页面。
在详情页中，您可以查询到：

- 集群的基础信息：集群名称/ID、版本、地域、创建时间、说明。
- 集群的配置：

集群配置	配置说明
最高 QPS	集群的 QPS 上限，如果超出此上限会对请求进行限流处理（生产 QPS + 消费 QPS）
命名空间最大数量	命名空间可创建的最大数量
Topic 最大数量	Topic 可创建的最大数量
消息保留时间	消息保留可配置的最长时间，在命名空间级别可以配置比这个更短的时间
最大存储容量	消息堆积消耗磁盘的最大容量，超过会禁止新消息的生产（正常情况下，消息堆积不应该太大，如发生此类情况请检查业务是否在正常消费消息）



获取接入地址

在集群管理列表页，单击目标集群操作列的配置接入点，在接入点列表页面可以获取集群接入地址。



删除集群

如果想删掉创建的集群，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。

命名空间

操作场景

命名空间是 TDMQ Pulsar 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 Pulsar 版时，创建多个命名空间，以便在同一个集群下将 TDMQ Pulsar 版应用于不同的场景。

> 说明：
>
> 同一个命名空间下的 Topic 和订阅的名称唯一。

操作步骤

创建命名空间

1. 登录TDMQ Pulsar 版控制台，进入【命名空间】页面。
2. 在【命名空间】页面，选择地域后，单击【新建】进入新建命名空间页面。
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），只能包含字母、数字、“-”及“_”。
- 消息 TTL：设置消息保留的时间，单位可以选择秒、分钟、小时、天，取值范围：60秒 - 15天。
- 消息保留策略：选择消费即删除或持久化保留消息。
- 说明：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。
   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了。

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【确定】，即可删除命名空间。

> 说明：
>
> 当命名空间内有 Topic 时，该命名空间不可被删除。

Topic 管理

操作场景

Topic 是 TDMQ Pulsar 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 Pulsar 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 Pulsar 版控制台，在左侧导航栏单击【Topic 管理】。
2. 在 Topic 管理页面，单击【新建】，弹出新建 Topic 的对话框。
3. 在新建 Topic 对话框中，填写以下信息：
    - Topic 名称：最多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、全局顺序、局部顺序、事务。
    - 分区数：全局顺序只有1个分区，其他为1-128个分区
    - 说明：填写 Topic 的说明信息，不超过128字符
4. 单击【保存】，在 Topic 列表中即可看见创建好的 Topic。



查询 Topic

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

编辑 Topic

1. 在 **【Topic 管理】**中，找到需要编辑的 Topic ，单击操作栏中的【编辑】。
2. 在弹出的对话框中可以对 Topic 的分区数（全局顺序型消息只有1个分区，不可编辑）和说明进行编辑。
3. 单击【提交】即完成对 Topic 的编辑。

发送消息

TDMQ Pulsar 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 管理】**中，找到需要编辑的 Topic ，单击操作列中的【发送消息】。

2. 在弹出的对话框中输入消息内容。消息长度不超过64KB。



3. 单击【提交】，完成消息的发送。消息发送后即可被 Topic 下的任意订阅者消费。

新增订阅

TDMQ Pulsar 版控制台支持手动创建订阅，在控制台进行相应的操作后即可完成订阅的创建。

1. 在【**Topic 管理】**中，找到需要创建订阅的 Topic ，单击操作列中的【新增订阅】。
2. 在弹出的对话框中输入订阅的名称和说明。
    - 订阅名称：只能包含字母、数字、“-”及“_”
    - 自动创建重试&死信队列：可以选择是否创建重试和死信 Topic
    - 说明：不超过128字符



3. 单击【提交】完成创建。
   创建后可通过单击操作列的【查看订阅】，查看订阅了该 Topic 的订阅，即可在列表中看到刚刚创建的订阅。

> 说明：
>
> - 如果没有选择自动创建重试和死信 Topic，TDMQ Pulsar 版会自动帮用户创建好一个重试队列和死信队列，以两个新的 Topic 呈现于 Topic 列表，分别以 “订阅名”+“retry” 和 “订阅名”+“dlq” 命名。
> - 关于重试队列和死信队列的概念和用法请参考 重试队列和死信队列 文档。

删除 Topic

> 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。

1. 在【**Topic 管理】**中，找到需要删除的 Topic ，单击操作列中的【删除】，或者勾选多个 Topic 之后单击 Topic 列表顶部的【删除】。
2. 在弹出的提示框中，单击【提交】，完成删除。

订阅管理

操作场景

在 TDMQ Pulsar 版控制台中，订阅代表一个具体的消费者以及其对某个 Topic 的订阅关系。当一个消费者订阅了某个 Topic 之后，则该 Topic 下的消息均可以被其消费。一个订阅可以订阅多个 Topic ，例如用户在一个 Topic 下创建了一个订阅后，其不仅会订阅当前的 Topic，还会订阅系统自动创建的重试队列 Topic。
本文档可以指导您使用消息队列 TDMQ Pulsar 版时，如何利用订阅管理对一个 Topic 下的订阅。

前提条件

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

操作步骤

查看订阅详情

1. 登录 TDMQ Pulsar 版控制台，在左侧导航栏中单击【Topic 管理】。
2. 在 Topic 管理列表页中，找到需要管理订阅的 Topic，单击操作列的【查看订阅】，进入订阅列表。
3. 在订阅列表中，一级列表可以看到订阅了当前 Topic 的所有订阅，二级展开后可以看到每个订阅的消费连接实例以及每个分片的消费进度。



设置 offset

1. 在订阅列表中，单击操作列的【offset设置】，按时间维度手动设定每个订阅的消费位移 offset（即指定该订阅下的

消费者从哪个时间点开始消费消息）。

2. 单击【提交】，完成设置。



重建重试/死信队列

由于用户可以手动删除 Topic，所以当用户删除了重试/死信队列的 Topic 后，如果希望重新让系统生成这两种队列，可以通过订阅中的重建重试/死信队列操作来进行重建。

删除订阅

> 注意：
>
> 在某个 Topic 下删除了某个订阅之后，若该订阅还订阅了其他 Topic（包括系统创建的重试/死信队列），则这个订阅还会在其他 Topic 下存在，并不会被真正删除。

1. 在订阅列表中，找到需要删除的订阅，单击操作列的【删除】，或者勾选多个订阅之后单击订阅列表顶部的【删除】。
2. 在弹出的提示框中，单击【提交】，完成删除。

VPC 接入

操作场景

本文档可以指导您使用 TDMQ Pulsar 版时，将自己当前私有网络 VPC 中的资源和 TDMQ Pulsar 版开通互访，以保证您部署的生产者/消费者客户端能正常和 TDMQ Pulsar 版通信。

前提条件

已在云平台上有购买云服务器 CVM 或者容器资源，并配置了私有网络 VPC。

操作步骤

1. 登录 TDMQ Pulsar 版控制台，进入集群管理页面，选择目标集群。
2. 单击操作列的配置接入点，进入集群的接入点配置页面。



3. 单击新建，在新建 VPC 接入点对话框中，选择 VPC、子网，填写备注。
   - VPC：选择您部署生产者或消费者所在的 VPC 网络
   - 子网：根据您的 IP 分配方式选择对应的子网
   - 备注（选填）：填写备注信息，不超过128个字符
4. 单击提交，即可完成 VPC 网络的接入。



5. 配置安全组策略。确保测试程序所在的安全组放行 TCP:6000-7000。

> 说明
>
> 您可以在接入点列表中看到已创建的接入点记录，其中有您需要在客户端配置的参数（路由 ID 和地址），详细介绍请参见 SDK 文档。

JWT 鉴权配置

操作场景

TDMQ Pulsar 版提供和原生 Pulsar 一样的 JWT 鉴权方式，用户可以通过在客户端参数中配置 Token 的方式来访问对应的TDMQ Pulsar 版资源。关于如何配置不同角色 Token 与 TDMQ Pulsar 版资源的关系，需要在控制台上进行操作，详细步骤请参考 角色与权限。

本文主要讲述如何在 TDMQ Pulsar 版客户端中配置 JWT 鉴权，以方便您安全地使用TDMQ Pulsar 版的 Client 对接 TDMQ Pulsar 版进行消息的生产消费（您可以在创建 Client 的时候添加密钥）。

鉴权配置

Java 客户端

在 Java 客户端中配置 JWT 鉴权：

```
PulsarClient client = PulsarClient.builder()
    .serviceUrl("pulsar://*.*.*.*:6000/")
    .authentication(AuthenticationFactory.token("eyJh****"))
    .listenerName("custom:1********0/vpc-******/subnet-********")//custom:+路由ID
    .build();
```

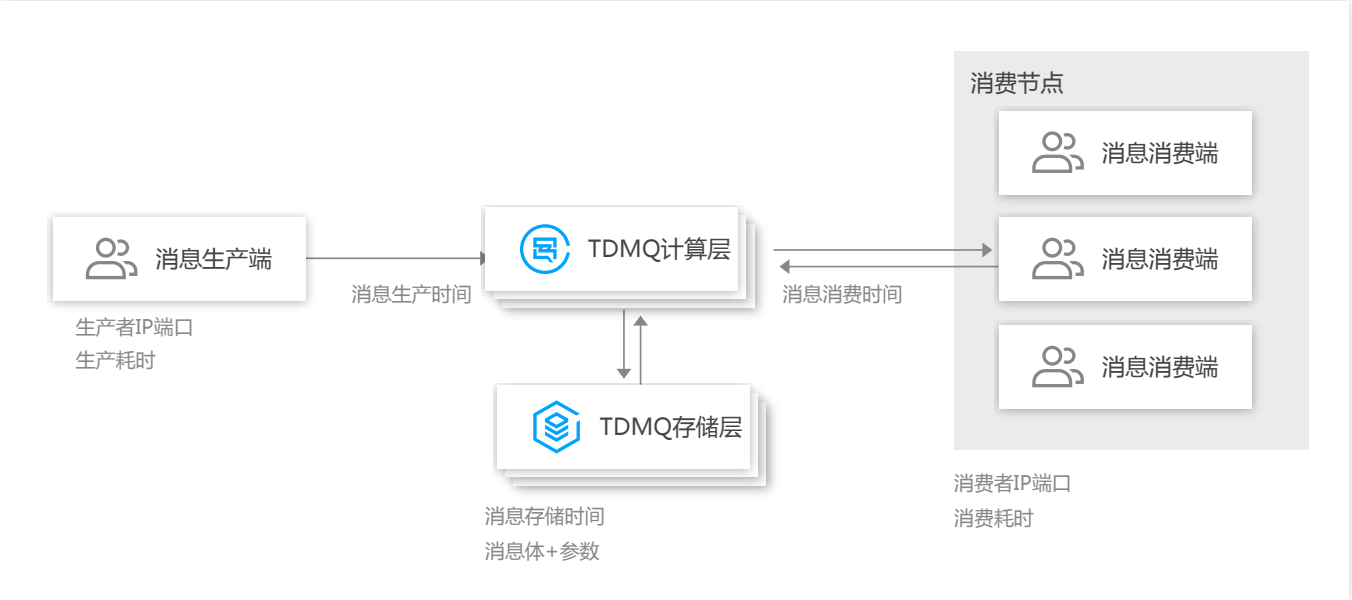
Go 客户端

在 Go 客户端中配置 JWT 鉴权：

```
client, err := NewClient(ClientOptions{
    URL:              "pulsar://*.*.*.*:6000",
    Authentication: NewAuthenticationToken("eyJh****"),
    ListenerName:    "custom:1300*****0/vpc-******/subnet-********",
})
```

消息查询与轨迹

当一条消息从生产者发送到 TDMQ Pulsar 版服务端，再由消费者进行消费，TDMQ Pulsar 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 Pulsar 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 Pulsar 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，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

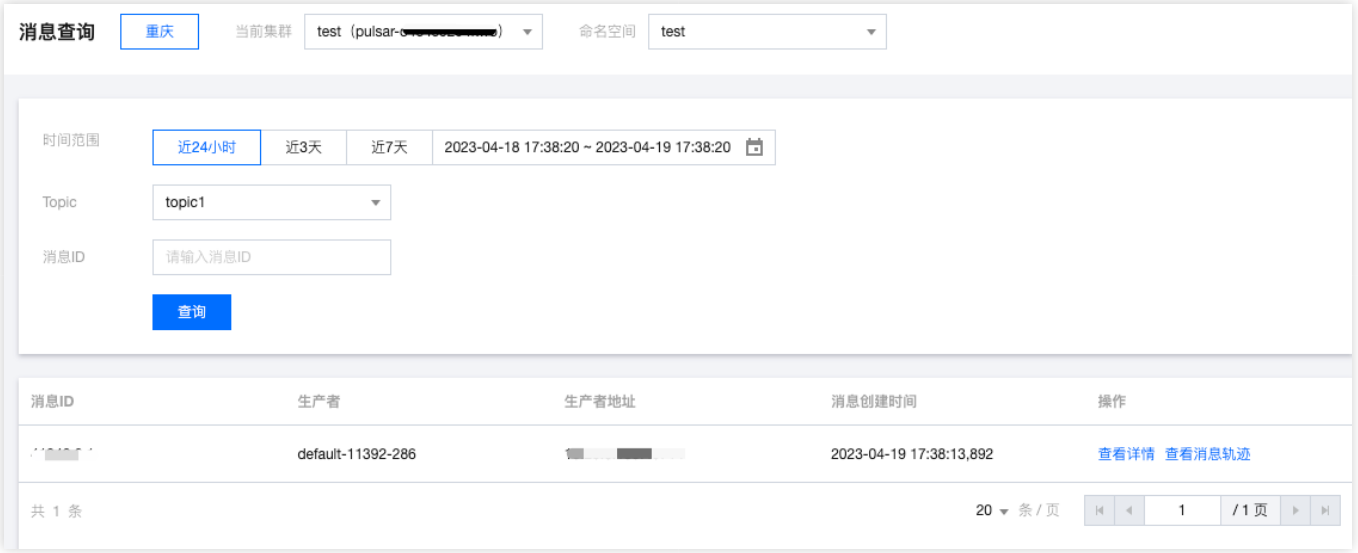
- 消息查询最多可以查询近7天的消息。
- 一次性最多可以查询10000条消息。

前提条件

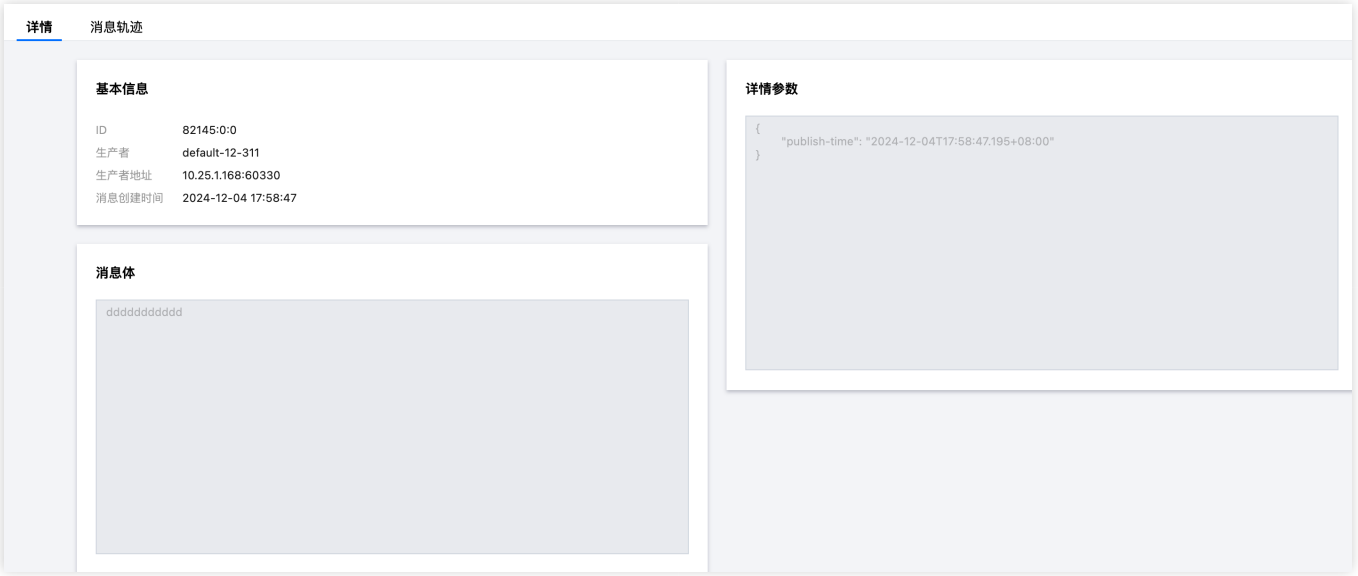
已经参考【SDK 文档】部署好生产端和消费端服务，并在7天内有消息生产和消费。

操作步骤

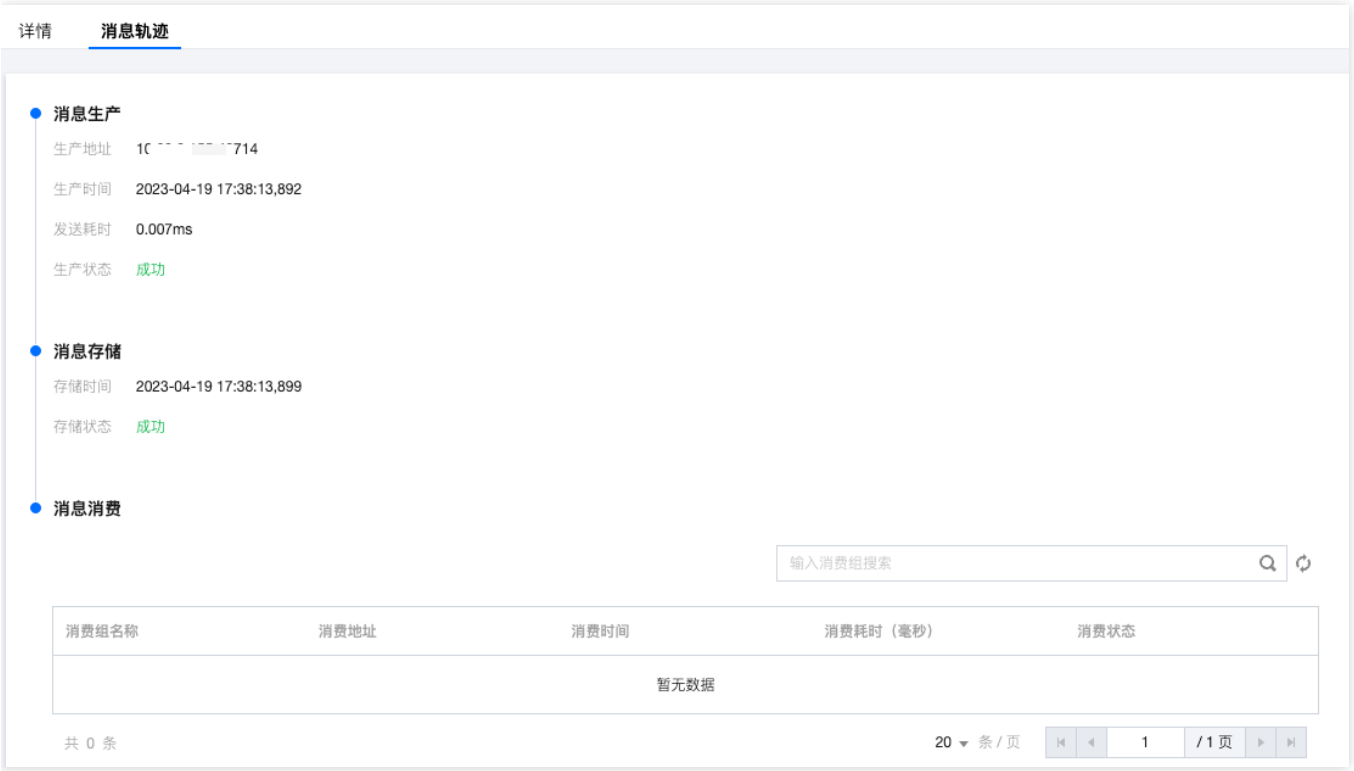
1. 登录 TDMQ Pulsar 版控制台，在左侧导航栏单击【消息查询】。
2. 在消息查询页面，首先选择地域和环境，再选择需要查询的时间范围，如果您知道对应的消息 ID，也可以输入消息 ID 进行精准查询。
3. 单击【查询】，下方列表会展示所有查询到的结果并分页展示。



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）以及参数。



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（详细说明请参考消息轨迹查询结果说明）。



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 Pulsar 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 Pulsar 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（由于当前 TDMQ Pulsar 版统一为强一致模式，消息持久化后才会被确认，所以该时间与生产时间一致，高性能模式下此时间会与生产时间不一致）。

参数	说明
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 Pulsar 版支持多订阅模式，一条消息可能会被多个订阅中的多个消费者消费。
列表中展示的信息说明：

参数	说明
消费组名称	即订阅名称。
消费地址	收到消息的消费者地址及端口。
消费时间	消费者回复确认信息（ACK）到 TDMQ Pulsar 版服务端，服务端接收到确认信息的时间。
消费耗时	消息从服务端投递到消费者，直到服务端收到消费者的确认信息（ACK）经过的时长，精确到微秒。
消费状态	消费者消费成功或者失败，消费者如果返回了取消确认的信息（NACK），这个字段会显示失败。

角色与鉴权

名词解释

- 角色（role）：TDMQ Pulsar 版的“角色”是 TDMQ Pulsar 版内专有的概念，是用户自行在 TDMQ Pulsar 版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 Pulsar 版的“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 Pulsar 版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。

使用场景

- 用户需要安全地使用 TDMQ Pulsar 版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

新增角色并授权

1. 登录 TDMQ Pulsar 版控制台，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，选择地域和当前集群后，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。
   - 所有权限：选择需要授权的命名空间，并配置生产消费权限。
4. 单击【保存】，完成当前集群命名空间的创建。

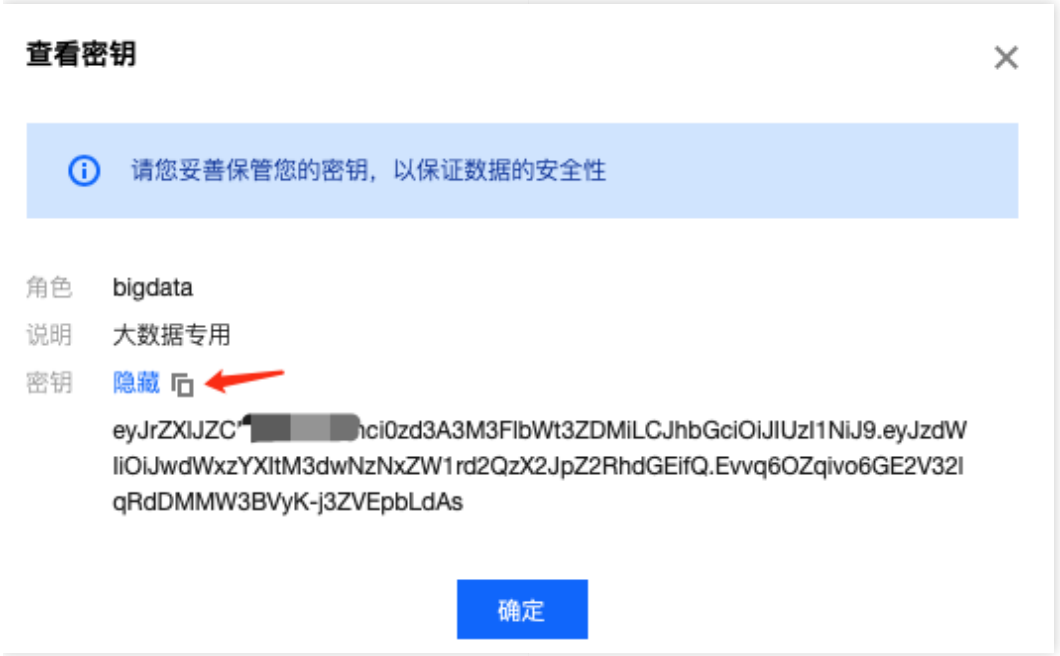


检查授权是否生效

1. 在 TDMQ Pulsar 版控制台的【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
   单击密钥列的【复制】。



方式二：【操作】列查看并复制
单击操作列的【查看密钥】，在查看密钥弹框中单击复制图标。



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 [JWT 鉴权配置]。

> 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。

3. 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 Pulsar 版控制台的【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 Pulsar 版控制台的【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。

配置告警

操作场景

云平台默认为所有用户提供云监控功能，无需用户手动开通。用户在使用了云平台某个产品后，云监控才可以开始收集监控数据。
TDMQ Pulsar 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，云监控可以通过邮件、短信、微信、电话等方式通知您，帮助您及时应对异常情况。

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 Pulsar 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. 登录云监控控制台。
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。

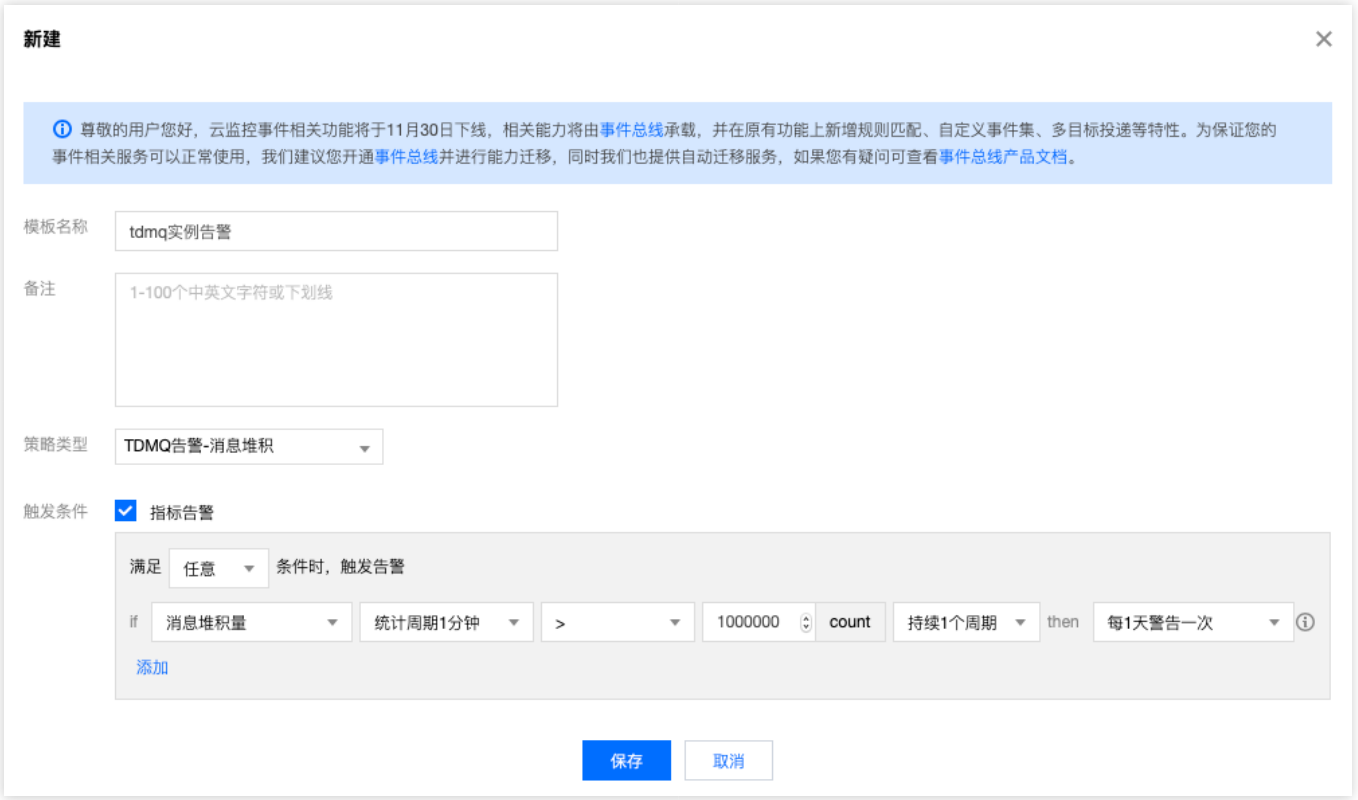
- 策略类型：选择 TDMQ告警。
- 告警对象：选择需要配置告警策略的 TDMQ Pulsar 版资源。
- 触发条件：支持选择模板和手动配置，默认选择手动配置，手动配置参见以下说明，新建模板参见 新建触发条件模版。

  > 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。

4. 通知模板：选择通知模版，也可以新建通知模版，设置告警接收对象和接收渠道。
5. 单击完成，完成配置。
   有关告警的更多信息，请参考 云监控告警服务。

新建触发条件模板

1. 登录云监控控制台。
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TDMQ 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。



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。

最佳实践
使用 TLS 传输加密

使用 TLS 传输加密

TLS

TDMQ支持与Apache Pulsar兼容，因此可实现TLS传输加密的配置。默认情况下，TDMQ客户端以纯文本形式与服务通信。这意味着所有数据都以明文形式发送。您可以使用 TLS 加密此流量，以保护流量免受中间人攻击者的窥探。

> 请注意，启用 TLS 可能会因加密开销而影响性能。

TLS概述

TLS 是一种形式。使用由公钥和私钥组成的密钥对可以执行加密。公钥加密消息，私钥解密消息。

要使用 TLS 传输加密，您需要两种密钥对，服务器密钥对和证书颁发机构。

您可以使用第三种密钥对，客户端密钥对，用于客户端身份验证。

您应该将证书颁发机构的私钥存储在非常安全的位置（完全加密、断开连接、有气隙的计算机）。至于证书颁发机构的公钥，信任证书，您可以自由共享。

对于客户端和服务器密钥对，管理员首先生成私钥和证书请求，然后使用证书颁发机构私钥对证书请求进行签名，最后生成证书。此证书是服务器/客户端密钥对的公钥。

对于 TLS 传输加密，客户端可以使用信任证书来验证服务器是否具有在客户端与服务器通信时证书颁发机构签署的密钥对。中间人攻击者无权访问证书颁发机构，因此他们无法使用这样的密钥对创建服务器。

对于 TLS 身份验证，服务器使用信任证书来验证客户端是否具有证书颁发机构签署的密钥对。然后将客户端证书的通用名称用作客户端的角色令牌（请参阅概述）。

创建 TLS证书

请按照以下指南设置证书颁发机构。。

认证机构

1. 为 CA 创建证书。您可以使用 CA 对代理和客户端证书进行签名。这确保了每一方都会信任其他方。您应该将 CA 存储在非常安全的位置（理想情况下完全与网络断开连接、隔离并完全加密）。
2. 输入以下命令为您的 CA 创建一个目录，并将openssl 配置文件放在该目录中。您可能需要在配置文件中修改公司名称和部门的默认答案。将 CA 目录的位置导出到环境变量 CA_HOME。配置文件使用此环境变量来查找 CA 需要的其余文件和目录。

```
mkdir my-ca
cd my-ca
wget https://raw.githubusercontent.com/apache/pulsar-site/main/site2/website/static/examples/openssl.cnf
export CA_HOME=$(pwd)
```

1. 输入以下命令以创建必要的目录、密钥和证书。

```
mkdir certs crl newcerts private
chmod 700 private/
touch index.txt
echo 1000 > serial
openssl genrsa -aes256 -out private/ca.key.pem 4096
# You need enter a password in the command above
chmod 400 private/ca.key.pem
openssl req -config openssl.cnf -key private/ca.key.pem \
    -new -x509 -days 7300 -sha256 -extensions v3_ca \
    -out certs/ca.cert.pem
# You must enter the same password in the previous openssl command
chmod 444 certs/ca.cert.pem
```

小费

macOS 上的默认设置 openssl 不适用于上述命令。 openssl 您必须通过 Homebrew升级：

```
brew install openssl
export PATH="/usr/local/Cellar/openssl@3/3.0.1/bin:$PATH"
```

未来版本 3.0.1 可能会发生变化。使用 brew install 命令输出中的实际路径。

1. 在您回答问题提示后，CA 相关文件将存储在该 ./my-ca 目录中。在该目录中：

- certs/ca.cert.pem 是公共证书。此公共证书旨在分发给所有相关方。
- private/ca.key.pem 是私钥。只有在为经纪人或客户签署新证书时才需要它，并且必须安全地保护此私钥。

服务器证书

创建 CA 证书后，您可以创建证书请求并使用 CA 对其进行签名。

以下命令会询问您几个问题，然后创建证书。当您被要求提供公用名时，您应该与代理的主机名匹配。您还可以使用通配符来匹配一组代理主机名，例如 *.broker.usw.example.com . 这确保了多台机器可以重用相同的证书。

小费

有时匹配主机名是不可能的或没有意义的，例如当您使用随机主机名创建代理时，或者您计划通过其 IP 连接到主机时。在这些情况下，您应该将客户端配置为禁用 TLS 主机名验证。。

1. 输入以下命令生成密钥。

```
openssl genrsa -out broker.key.pem 2048
```

代理希望密钥为PKCS 8格式，因此输入以下命令进行转换。

```
openssl pkcs8 -topk8 -inform PEM -outform PEM \
    -in broker.key.pem -out broker.key-pk8.pem -nocrypt
```

1. 输入以下命令以生成证书请求。

```
openssl req -config openssl.cnf \
  -key broker.key.pem -new -sha256 -out broker.csr.pem
```

1. 通过输入以下命令，使用证书颁发机构对其进行签名。

```
openssl ca -config openssl.cnf -extensions server_cert \
  -days 1000 -notext -md sha256 \
  -in broker.csr.pem -out broker.cert.pem
```

此时，您有一个证书 broker.cert.pem 和一个密钥 ， broker.key-pk8.pem 您可以使用它们 ca.cert.pem 来为您的代理和代理节点配置 TLS 传输加密。

配置代理

要将 Pulsar代理配置为使用 TLS 传输加密，您需要对 broker.conf 位于Pulsar 安装 conf 目录中的进行一些更改。

将这些值添加到配置文件（在必要时替换适当的证书路径）：

```
brokerServicePortTls=6651
webServicePortTls=8081
tlsRequireTrustedClientCertOnConnect=true
tlsCertificateFilePath=/path/to/broker.cert.pem
tlsKeyFilePath=/path/to/broker.key-pk8.pem
tlsTrustCertsFilePath=/path/to/ca.cert.pem
```

> 您可以在代理配置中找到文件中可用参数的完整列表 conf/broker.conf ，以及这些参数的默认值
>
> TLS 协议版本和密码

您可以将代理（和代理）配置为需要特定的 TLS 协议版本和密码以进行 TLS 协商。您可以使用 TLS 协议版本和密码来阻止客户端请求可能存在弱点的降级 TLS 协议版本或密码。

TLS 协议版本和密码属性都可以采用多个值，以逗号分隔。协议版本和密码的可能值取决于您使用的 TLS 提供程序。如果 OpenSSL 可用，则 Pulsar 使用 OpenSSL，但如果 OpenSSL 不可用，则 Pulsar 默认回到 JDK 实现。

```
tlsProtocols=TLSv1.3,TLSv1.2
tlsCiphers=TLS_DH_RSA_WITH_AES_256_GCM_SHA384,TLS_DH_RSA_WITH_AES_256_CBC_SHA
```

OpenSSL 目前支持 TLSv1.1，TLSv1.2 和 TLSv1.3 为协议版本。您可以从 openssl ciphers 命令获取支持的密码列表，即 openssl ciphers -tls1_3 .

代理配置

代理需要在两个方向上配置 TLS，客户端连接代理，代理连接代理。

```
# For clients connecting to the proxy
tlsEnabledInProxy=true
tlsCertificateFilePath=/path/to/broker.cert.pem
tlsKeyFilePath=/path/to/broker.key-pk8.pem
tlsTrustCertsFilePath=/path/to/ca.cert.pem

# For the proxy to connect to brokers
tlsEnabledWithBroker=true
brokerClientTrustCertsFilePath=/path/to/ca.cert.pem
```

客户端配置

启用 TLS 传输加密时，您需要将客户端配置为使用 https:// Web 服务 URL 的端口 8443 和 pulsar+ssl:// 代理服务 URL 的端口 6651。

由于您上面生成的服务器证书不属于任何默认信任链，您还需要指定信任证书的路径（推荐），或者告诉客户端允许不受信任的服务器证书。

主机名验证

主机名验证是 TLS 安全功能，如果“CommonName”与主机名连接的主机名不匹配，客户端可以拒绝连接到服务器。默认情况下，Pulsar 客户端禁用主机名验证，因为它要求每个代理都有一个 DNS 记录和一个唯一的证书。

此外，由于管理员对证书颁发机构拥有完全控制权，因此不良行为者不太可能发起中间人攻击。“allowInsecureConnection”允许客户端连接到其证书未经批准的 CA 签名的服务器。客户端默认禁用“allowInsecureConnection”，您应该在生产环境中始终禁用“allowInsecureConnection”。只要您禁用“allowInsecureConnection”，中间人攻击就要求攻击者有权访问 CA。

您可能想要启用主机名验证的一种情况是，您在 VIP 后面有多个代理节点，并且 VIP 具有 DNS 记录，例如 pulsar.mycompany.com。在这种情况下，您可以使用 pulsar.mycompany.com 作为“CommonName”生成 TLS 证书，然后在客户端上启用主机名验证。

消息幂等性

应用的幂等是在分布式系统设计时必须要考虑的一个关键点。如果对幂等没有额外的考虑，那么在业务出现处理失败的情况时，可能出现重复消费相同的消息，从而导致出现不符合业务预期的情况。为了避免上述异常，消息队列的消费者在接收到消息后，有必要根据业务上的唯一 Key 对消息做幂等处理。

什么是消息幂等

定义

当业务多次消费到同一条消息的结果与消费一次的结果是相同的，同时多次消费同一条消息并未对业务系统产生任何负面影响，那么这个消费者的处理过程就是幂等的。

场景示例

举个例子，在银行支付系统的场景下，当消费端消费到扣款消息后，系统将对订单执行扣款操作，扣款金额为1元。如果因由于网络不稳定等一些原因导致扣款消息再次被消费到，如果最终的业务结果是只扣款一次，扣费1元，且用户的扣款记录中对应的订单只有一条扣款流水，并没有多次被多次扣费，那么这次扣款操作是符合要求的，整个消费过程实现了消费幂等。

适用场景

发送消息引起消息重复场景

生产者在发送一条消息后，服务端接收成功并完成持久化，如果此时出现了网络闪断或者客户端重启等异常导致服务端对客户端应答失败，此时生产者由于没有收到服务端的确认消息，从而尝试再次发送消息，这时会导致后续消费者会收到两条内容相同但 Message ID 不同的消息。

消费消息引起消息重复场景

消费端消费到了消息并完成业务处理，当消费端给服务端返回 ACK 的时候，此时网络发生异常。当消费端再次来消

费消息时，会再次消费到已被处理过的消息，消费者收到了两条内容相同并且 Message ID 也相同的消息。

处理方案

根据以上两种场景可以看出消息重复会出现以下两种情况：

- 可能出现不同的 Message ID 对应的消息内容相同。
- 可能是相同的 Message ID 同时消息内容相同。
  所以不建议以 Message ID 作为处理依据，建议以业务的唯一标识作为幂等处理的依据。例如支付场景可以将订单号，作为幂等处理的依据。消费到消息后，取出业务中的订单号，业务根据订单号进行判断是否被处理。

代码示例

```
public static class Order {
    public String orderId;
    public String orderData;
}
```

生产端

```
Producer<Order> producer = client.newProducer(Schema.AVRO(Order.class)).create();
producer.newMessage().value(new Order("orderid-12345678", "orderData")).send();
```

消费端

```
Consumer<Order> consumer = client.newConsumer(Schema.AVRO(Order.class)).subscribe();
Order order = consumer.receive().getValue();
String key = order.orderId;
```

获取到业务的唯一标识 orderId 后，对其进行去重操作。

常见的去重方式

利用数据库进行去重

业务上的幂等操作可以添加一个过滤的数据库，例如设置一个去重表，也可以在数据库中通过唯一索引来去重。

举一个例子，现在要根据订单流转的消息在数据库中写一张订单 Log 表，可以把订单 ID 和修改时间戳做一个唯一索引进行约束。
当消费端消费消息出现相同内容的消息，会多次去订单 Log 表中进行写入，由于添加了唯一索引，除了第一条之外，后面的都会失败，这就从业务上保证了幂等，即使消费多次，也不会影响最终的数据结果。

设置全局唯一消息 ID 或者任务 ID

调用链 ID 也可以应用在这里。在消息生产的时候向每条消息中添加一个唯一 ID，消息被消费后，在缓存中设置一个 Key 为对应的唯一 ID，代表数据已经被消费，当消费端去消费时，就可以根据这条记录，来判断是否已经处理过。

消息压缩

背景描述

由于 Pulsar 限制消息最大为 5MB，消息体过大将会导致消息发送失败。这时需要客户端端需要将大消息进行压缩，以支持20M大小的消息体发送。

Pulsar 大消息处理

Pulsar 的消息最大限制默认是 5MB，Producer发送的消息大小超过5MB会导致消息发送失败。如果客户端发送单条消息大小超过该限制，我们可以采用如下两种方式来处理：

- Chunk Message：Pulsar 提供 Chunk Message 功能，开启 Chunk 机制时，客户端能够自动对大消息进行拆分，并保证消息的完整性，在 Consumer 能自动整合消息。
- 压缩消息：主要是对消息数据中相同字符序列进行替换，来压缩消息的大小。Pulsar支持 LZ4、ZLIB、ZSTD、SNAPPY四种压缩算法
  我们这里推荐压缩消息对大消息进行处理。

压缩算法分析比较

算法介绍

- LZ4
  LZ4是一种无损数据压缩算法，可以提供极快的压缩和解压缩速度，对于 CPU 占用少。
- ZLIB
  ZLIB 压缩算法是一种常用的无损数据压缩技术，它可以有效地减少收发数据的大小，从而提高网络传输效率和网络容量。ZLIB 压缩算法是基于 Lempel-Ziv 压缩算法的一种变体，可以将原始数据压缩到原来的一半大小以下，并且支持压缩和解压缩操作。
- ZSTD
  ZSTD 压缩算法是一种 Huffman 编码的压缩算法，是 LZ77 的一种变种，可以针对不同数据进行有效压缩。它是一种实时编码算法，在处理大数据时可以更快速、更高效地压缩数据。相比其他压缩算法，ZSTD在提高数据压缩率的同时兼顾压缩速度。
- SNAPPY

SNAPPY 压缩是一种无损压缩技术，它依赖于 LZ77 原理来实现压缩效果。SNAPPY 压缩的核心原理是：只要数据流找到两个字符串之间的重复，就会用一组更短的代码来表示这个字符串，这样就可以减少数据流的大小。

算法对比

压缩算法	压缩比	压缩速度	解压速度
ZLIB 1.2.11 -1	2.743	110MB/S	400MB/S
LZ4 1.8.1	2.101	750MB/S	3700MB/S
ZSTD 1.3.4-1	2.877	470MB/S	1380MB/S
SNAPPY 1.1.4	2.091	530MB/S	1800MB/S

- 吞吐量：LZ4 > SNAPPY > ZSTD > ZLIB
- 压缩比：ZSTD > ZLIB > LZ4 > SNAPPY
- 物理资源方面，SNAPPY算法占用的网络带宽最多，ZSTD算法占用的网络带宽最少

各压缩算法测试

测试结果

注意，以下测试结果仅供参考。压缩效果，需要根据具体消息体内容来验证。

消息大小	消息	压缩算法	topic监控消息大小	客户端消息压缩耗时	消息发送耗时
5M	随机消息体	LZ4(阀值5M)	9.95M	31ms	0.049ms
		ZLIB	7.26M	31ms	0.038ms
		ZSTD	8.20M	31ms	0.039ms
		SNAPPY(阀值5M)	9.70M	33ms	0.046ms
6M	随机消息体	ZLIB(阀值6M)	8.71M	35ms	0.044ms
		ZSTD(阀值6M)	9.84M	35ms	0.046ms

消息大小	消息	压缩算法	topic监控消息大小	客户端消息压缩耗时	消息发送耗时
20M	相同消息体	LZ4	0.16M	41ms	0.006ms
		ZLIB	0.20M	42ms	0.006ms
		ZSTD	0.01M	42ms	0.003ms
		SNAPPY	2.47M	41ms	0.021ms
40M	相同消息体	LZ4	0.32M	123ms	0.008ms
		ZLIB	0.39M	122ms	0.008ms
		ZSTD	0.01M	124ms	0.004ms
		SNAPPY	4.95M	123ms	0.036ms
80M	相同消息体	LZ4	0.63M	241ms	0.009ms
		ZLIB	0.39M	244ms	0.01ms
		ZSTD	0.01M	243ms	0.004ms
		SNAPPY(阀值80M)	9.9M	243ms	0.056ms
160M	相同消息体	LZ4	1.26M	484ms	0.013ms
		ZLIB	1.56M	479ms	0.016ms
		ZSTD	0.03M	481ms	0.004ms
320M	相同消息体	LZ4	2.5M	1035ms	0.03ms
		ZLIB	3.1M	1008ms	0.027ms
		ZSTD	0.03M	949ms	0.004ms
585M	相同消息体	LZ4	4.59M	1705ms	0.027ms
		ZLIB	5.67M	1733ms	0.03ms
		ZSTD	0.11M	1722ms	0.006ms

总结：

1、在纯随机数据流中，四种压缩算法压缩效率都不是很高。消息大小超过 5MB，四种压缩算法都无法将消息压缩到 5MB 以内；

2、在重复数据较多的数据流中，四种压缩算法可以实现很高的压缩速率，其中LZ4、ZLIB、ZSTD压缩算法可以实现

将600MB内的消息压缩到5MB以内。

消息压缩 Demo 及使用测试

Demo 代码地址

使用方式

生产端调用参数

java -jar tdmq-sdk-demo-1.0-SNAPSHOT-jar-with-dependencies.jar pulsar://xxxx:6650 eyJrZXlJZCI6ImRlZmF1bHRfa2V5SWQiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzdXBlcnVzZXIifQ.dYcCfp4XrdWRKdKaWylobY-_xEExfRCi1pMvNyZXbqU pulsar-78ra8ownxb7d/BigMSGSpace/BigMSGTopic subname 1 500 0 1 20480 1 0

消费端调用参数：

java -jar tdmq-sdk-demo-1.0-SNAPSHOT-jar-with-dendencies.jar pulsar://xxxx:6650 eyJrZXlJZCI6ImRlZmF1bHRfa2V5SWQiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzdXBlcnVzZXIifQ.dYcCfp4XrdWRKdKaWylobY-_xEExfRCi1pMvNyZXbqU pulsar-92d7w2mjwmv9/BigMessSpace/BigMessTopic subname 1 500 1

通用参考
消息类型

在消息队列中，根据消息的特性及使用场景，可以将消息作如下分类：

消息类型	消费顺序	性能	适用场景
普通消息	无顺序	最好	吞吐量巨大，且对生产和消费顺序无要求
局部顺序消息	同一分区下所有消息遵循先入先出（FIFO）规则	较好	吞吐量较大，同一分区内有序，不同分区内无序
全局顺序消息	同一 Topic 下所有消息遵循先入先出（FIFO）规则	一般	吞吐量一般，全局有序，单分区
死信消息	-	-	无法正常消费的消息

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。

局部顺序消息

局部顺序消息相较于普通消息类型，多了一个局部有顺序的特性。即同一个分区下，其消费者在消费消息的时候，严格按照生产者投递到该分区的顺序进行消费。局部顺序消息在保证了一定顺序性的同时，保留了分区机制提升性能。但局部顺序消息不能保证不同分区之间的顺序。

全局顺序消息

全局顺序消息最大的特性就在于，严格保证消息是按照生产者投递的顺序来消费的。所以其使用的是单分区来处理消息，用户不可自定义分区数，相比前两种消息类型，这种类型消息的性能较低。

死信消息

死信消息是指无法被正常消费的消息。 TDMQ Pulsar 版会在创建新的订阅（消费者确定了与某个 Topic 的订阅关系）时自动创建一个死信队列用于处理这种消息。

重试队列和死信队列

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。

常见问题
概念相关

什么是消息队列 TDMQ Pulsar 版?

消息队列 Pulsar 版（TDMQ for Apache Pulsar，简称 TDMQ Pulsar 版）基于 Apache 开源项目 Pulsar 的金融级分布式消息中间件，是一款具备高一致、高可靠、高并发的消息队列。

TDMQ Pulsar 版拥有原生 Java 、 C++、Python、Go 多种 API， 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。详细说明请参考 [产品概述]。

如何使用 TDMQ Pulsar 版？

您可以参考 [快速入门]和 [操作指南] 文档，快速上手并使用 TDMQ Pulsar 版。

TDMQ Pulsar 版有哪些应用场景？

TDMQ Pulsar 版可应用在异步解耦、分布式事务的数据一致性、ETL 流程、消息的顺序收发等场景。详细说明请参考 [应用场景] 和 [最佳实践] 文档。

TDMQ Pulsar 版相对于其他消息队列产品有哪些优势？

相对于 Kafka 和 RabbitMQ，TDMQ Pulsar 版提供了更好的扩展性、伸缩性，以及独有的统一的兼具流和排队功能的消息传递模型。
TDMQ Pulsar 版能为使用者提供统一的消息传递平台来提高运营效率，删除冗余系统并减少硬件和软件成本。详细了解可查看 [产品优势] 。

SDK文档
SDK概览

消息队列 TDMQ Pulsar 版现支持 TDMQ 版 SDK 和 Pulsar 社区版 SDK。以下是消息队列 TDMQ Pulsar 版所支持的多语言 SDK：

> 说明：
>
> - 为了更好地和 Pulsar 开源社区统一，自2021年4月30日起，云平台版 SDK 将停止功能更新，TDMQ Pulsar 版推荐您使用社区版本的 SDK。
> - 如果您已使用了云平台版 SDK，并且确定未使用下文中列出的 额外功能，可以直接替换成社区版的 SDK。

协议类型	SDK 语言
TCP 协议（Pulsar 社区版）	Go SDK
	C++ SDK
	Python SDK
	Node.js SDK
TCP 协议（云平台版，仅限存量客户使用）	Go SDK
	Java SDK

云平台版 SDK 额外支持的功能

功能	说明
标签过滤	可以为消息附上 Tag 属性，消息支持绑定多个标签，订阅者可按照标签筛选决定是否处理消息。这样可以利用标签优化业务架构，节省 Topic 资源消耗。
支持退避式延时消息	对于失败超时重试场景，并不需要在短时间内大量重试，因为很可能还是失败，依次扩大时间间隔进行重试是比较合理的。

SDK 升级说明

> 说明：
>
> TDMQ Pulsar 版内测版本将于2020年9月17日上午09:00~12:00进行停服升级，升级后需要用户做少量改造重新接入。
>
> 升级之后，我们会完整帮您迁移数据，包括您创建的环境和 Topic 以及 VPC 接入点等数据，您无需担心基础数据的丢失。

操作场景

本文主要介绍新版 TDMQ Pulsar 版 SDK 升级后如何快速恢复您原先业务代码的接入。

操作步骤

步骤1：创建角色并授权

1. 登录 【TDMQ Pulsar 版控制台】，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，选择地域后，单击【新建】进行新建角色。
   详细操作可参考 【角色与鉴权-新增角色】 文档。



3. 在左侧导航栏单击【环境管理】，在所需环境上单击【配置权限】，配置刚刚创建的角色的权限。

4. 详细操作可参考 [角色与鉴权-编辑角色] 文档。



步骤2：配置角色密钥

Java 客户端

当前由于 Pulsar 官方只有 Java 客户端提供了带有 listenerName 参数的最新版本，TDMQ 此次更新需要依赖于 listenerName 参数，所以截止目前只有 Java 客户端可以直接从官网下载。

1. 按照 [Java SDK 下载方式] 或者 Pulsar 官方文档 更新 Maven 依赖。在此我们仍推荐您按前者的指引使用云平台官方提供的SDK。
2. 前往 TDMQ 控制台【角色管理】，找到刚刚创建的角色，单击复制密钥。
3. 在创建 Client 的代码中加入刚刚复制的密钥，并添加 listenerName 参数。

```
PulsarClient client = PulsarClient.builder()
        .serviceUrl("pulsar://*.*.*.*:6000/")
        .listenerName("custom:1300*****0/vpc-******/subnet-********")
        .authentication(AuthenticationFactory.token("eyJh****"))
        .build();
```

> 注意：
>
> listenerName 即“custom:”拼接原先的路由 ID（NetModel），路由 ID 可以在控制台【环境管理】接入点查看并复制。

Go 客户端

Go 客户端 Pulsar 官方目前还未更新最新适配的客户端，在官方适配之前需要下载云平台提供的 SDK。

1. 下载 SDK。

```
$ go get -u "github.com/TencentCloud/tdmq-go-client@v0.3.0-beta.2"
```

2. 在代码中重新导入。

```
import "github.com/TencentCloud/tdmq-go-client/pulsar"
```

3. 前往 TDMQ Pulsar 版控制台【角色管理】，找到刚刚创建的角色，单击复制密钥。

4. 在创建 Client 的代码中加入刚刚复制的密钥，并添加 ListenerName 参数.

```
client, err := pulsar.NewClient(pulsar.ClientOptions{
        URL:              "pulsar://*.*.*.*:6000",
        ListenerName:    "custom:1300*****0/vpc-******/subnet-********",
        Authentication:  pulsar.NewAuthenticationToken(),
    })
```

> 注意：
>
> listenerName 即“custom:”拼接原先的路由 ID（ NetModel ），路由 ID 可以在控制台【环境管理】接入点查看并复制。

步骤3：部署客户端

在修改完客户端代码后，您需要重新将客户端部署到原先的环境进行验证和生产。

Go SDK

操作场景

TDMQ Pulsar 版提供了 Go 语言的 SDK 来调用服务，进行消息队列的生产和消费。
本文主要介绍 Go SDK 的使用方式，提供 Demo 工程的环境配置、下载、代码编写及运行示例，帮助工程师快速搭建 TDMQ Pulsar 版测试工程。

前提条件

- 已经在本地安装 Golang 开发环境 。
- 已经准备好 Go 1.11+ 的部署环境（云服务器或其他云资源），且该环境所在的 VPC 已接入TDMQ Pulsar 版（参考 [VPC 接入指南] ）。
- 已获取调用地址（URL）和路由 ID（NetModel）。
- 这两个参数均可以在【集群管理】的接入点列表中获取。请根据客户端部署的云服务器或其他资源所在的私有网络选择正确的接入点来复制参数信息，否则会有无法连接的问题。



- 已参考 【角色与鉴权】文档配置好了角色与权限，并获取到了对应角色的密钥（Token）。

操作步骤

准备 Demo 环境

1. 安装 IDE
   您可以 [安装 GoLand] 或其它的 Go IDE 运行这个 Demo，直接通过 go run 执行也可以。
2. 配置 GCC 环境
   因为现在的 SDK 依赖了 CGO 的库，所以需要本地配置64位 GCC，可以通过 [MinGW] 来安装。
3. 打开命令控制台，运行以下命令：

```
go get -u "github.com/TencentCloud/tdmq-go-client@v0.3.0-beta.3"
```

如果国内网络环境下载比较慢，可以通过配置 [Go Proxy] 来解决。
如果处于无法连接外网的环境下，需要先行下载依赖文件 压缩包，将压缩包里的文件放在 %GOPATH/pkg/mod/cache/download 文件夹下即可， %GOPATH 可通过如下指令获取：

```
# linux
go env | grep GOPATH

# Windows
go env | findstr GOPATH
```

> 注意：
>
> 目前 Pulsar 官方尚未更新最新适配的客户端（目前官方的 Go 语言客户端在 client 构造中，没有用于声明路由方式的 ListenerName 字段），云平台已将适配后的 Go 客户端提交至社区，即将在下个版本发布，在官方适配之前您需要先使用云平台提供的 SDK。

创建 Demo工程

1. 使用 IDE 创建一个新工程，在文件夹中创建 go.mod 文件并编辑如下：

```
module example/godemo

go 1.12

require github.com/TencentCloud/tdmq-go-client v0.3.0-beta.3
```

上述 v0.3.0-beta.3 是 Go SDK 的版本，云上资源环境中下载的依赖文件压缩包也需要是同样的版本。
2. 创建 producer.go 和 consumer.go 测试 Demo 文件。

- producer.go 代码内容如下，其中 ListenerName 即 custom: 拼接路由 ID（NetModel），路由 ID 可以在控制台【[集群管理]】的接入点列表查看并复制， NewAuthenticationToken 即角色密钥，可以在【[角色管理]】页面复制。
  ::: go
  package main

```
import (
"context"
"github.com/TencentCloud/tdmq-go-client/pulsar"
"log"
"strconv"
)

func main() {

    client, err := pulsar.NewClient(pulsar.ClientOptions{
        URL:              "pulsar://*.*.*.*:6000",
```

```
        ListenerName:   "custom:1300*****0/vpc-******/subnet-********",
        Authentication: pulsar.NewAuthenticationToken("eyJh****"),
    })
    if err != nil {
        log.Fatal(err)
    }
    defer client.Close()

    producer, err := client.CreateProducer(pulsar.ProducerOptions{
        DisableBatching: true,
        Topic:          "persistent://pulsar-****/namespace/topic-1",
    })
    if err != nil {
        log.Fatal(err)
    }
    defer producer.Close()

    ctx := context.Background()

    for j := 0; j < 10; j++ {
        if msgId, err := producer.Send(ctx, &pulsar.ProducerMessage{
            Payload: []byte("Hello " + strconv.Itoa(j)),
        }); err != nil {
            log.Fatal(err)
        } else {
            log.Println("Published message: ", msgId)
        }
    }

}
```

:::

其中 Topic 名称需要填入完整路径，即`persistent://pulsar-****/namespace/Topic`的组合，其中`pulsar-****/namespace/topic`的部分可以从控制台【[Topic管理]】页面直接复制。



- consumer.go 的代码内容如下：

::: go

```
package main

import (
"context"
"fmt"
"github.com/TencentCloud/tdmq-go-client/pulsar"
"log"
)

func main() {

    client, err := pulsar.NewClient(pulsar.ClientOptions{
        URL:            "pulsar://10.*.*.*:6000",//更换为接入点地址
        ListenerName:    "custom:1300*****0/vpc-******/subnet-********",
        Authentication: pulsar.NewAuthenticationToken("eyJh****"),
    })
    if err != nil {
        log.Fatal(err)
    }
    defer client.Close()

    consumer, err := client.Subscribe(pulsar.ConsumerOptions{
        Topics:            []string{"persistent://pulsar-****/namespace/topic-1"},
        SubscriptionName: "my-sub",
        Type:              pulsar.Shared,
    })
    if err != nil {
        log.Fatal(err)
    }
    defer consumer.Close()

    for ; ; {
        msg, err := consumer.Receive(context.Background())
        if err != nil {
            log.Fatal(err)
        }
        fmt.Printf("Received message msgId: %#v -- content: '%s' -- topic : '%v'\n",
            msg.ID(), string(msg.Payload()), msg.Topic())

        consumer.Ack(msg)
    }

}
```

:::

测试验证

您需要先将 Demo 代码打包上传到云服务器上，且确认该云服务器所在的私有网络 VPC 和 TDMQ 中配置的接入点吻合。

接下来进入 Demo 测试，先执行 go run 指令启动 consumer.go ，命令如下：

```
go run consumer.go
```

再类似启动 producer.go，观察控制台消息：
在 producer.go 运行的控制台可以看到有10条消息发送成功：

```
2020/02/20 20:20:20 Published message:  &{581 0 0 0 <nil>  <nil>}
2020/02/20 20:20:20 Published message:  &{581 1 0 0 <nil>  <nil>}
2020/02/20 20:20:20 Published message:  &{581 2 0 0 <nil>  <nil>}
2020/02/20 20:20:20 Published message:  &{581 3 0 0 <nil>  <nil>}
2020/02/20 20:20:21 Published message:  &{581 4 0 0 <nil>  <nil>}
2020/02/20 20:20:21 Published message:  &{581 5 0 0 <nil>  <nil>}
2020/02/20 20:20:21 Published message:  &{581 6 0 0 <nil>  <nil>}
2020/02/20 20:20:21 Published message:  &{581 7 0 0 <nil>  <nil>}
2020/02/20 20:20:21 Published message:  &{581 8 0 0 <nil>  <nil>}
2020/02/20 20:20:22 Published message:  &{581 9 0 0 <nil>  <nil>}
```

在 consumer.go 运行的控制台可以看到消息被成功接收并打印出来：

```
Received message msgId: &pulsar.messageID{ledgerID:581, entryID:0, batchIdx:0, partitionIdx:0, tracker:(*pulsar.ackTracker)(nil), consumer:(*pulsar.partitionConsumer)(0xc000198000)} -- content: 'Hello 0' -- topic : 'persistent://pulsar-****/namespace/topic-1'

Received message msgId: &pulsar.messageID{ledgerID:581, entryID:1, batchIdx:0, partitionIdx:0, tracker:(*pulsar.ackTracker)(nil), consumer:(*pulsar.partitionConsumer)(0xc000198000)} -- content: 'Hello 1' -- topic : 'persistent://pulsar-****/namespace/topic-1'

Received message msgId: &pulsar.messageID{ledgerID:581, entryID:2, batchIdx:0, partitionIdx:0, tracker:(*pulsar.ackTracker)(nil), consumer:(*pulsar.partitionConsumer)(0xc000198000)} -- content: 'Hello 2' -- topic : 'persistent://pulsar-****/namespace/topic-1'

Received message msgId: &pulsar.messageID{ledgerID:581, entryID:3, batchIdx:0, partitionIdx:0, tracker:(*pulsar.ackTracker)(nil), consumer:(*pulsar.partitionConsumer)(0xc000198000)} -- content: 'Hello 3' -- topic : 'persistent://pulsar-****/namespace/topic-1'

...//后续省略
```

则 Go 版本的 SDK Demo 运行成功。

Tag 功能

为了配置 Go SDK 的 Tag 支持，需要在消费者订阅的时候配置相应的 Tag 参数，如下：
::: go
consumer, err := client.Subscribe(pulsar.ConsumerOptions{
Topics: []string{"persistent://pulsar-/namespace/topic-1"},
SubscriptionName: "my-sub",
Type: pulsar.Shared,
TagMapTopicNames: map**[**string]string{"persistent://pulsar-/namespace/topic-1":"a||b"},

```
})
if err != nil {
log.Fatal(err)
}
defer consumer.Close()
```

不同的 Tag 之间用“||”符号分隔，此时创建的 consumer 就会只消费 Tag 中包含 a 或 b 的消息，如下创建 producer 并发送消息：

```go
// 创建 Producer 对象
producer, err := client.CreateProducer(pulsar.ProducerOptions{
Topic: "persistent://clusterid/namespace/topic-1",
})
if err != nil {
log.Fatal(err)
}
defer producer.Close()
// 发送消息
for j := 0; j < 10; j++ {
if msgId, err := producer.Send(ctx, &pulsar.ProducerMessage{
Payload: []byte("Hello " + strconv.Itoa(j)),
Tags: []string{"a","b"},
}); err != nil {
log.Fatal(err)
} else {
log.Println("Published message: ", msgId)
}
}
```

可以用 consumer 来接收消息，完成消费。

消息延迟重试

有时我们接收到一条消息时希望能在可控的延迟时间之后再次消费这个消息，这个功能现在也集成在 SDK 中，首先我们需要在创建 consumer 时配置相应的参数：

```go
consumer, err := client.Subscribe(pulsar.ConsumerOptions{
Topics: []string{"persistent://pulsar-****/namespace/topic-1"},
SubscriptionName: "my-sub",
Type: pulsar.Shared,
//EnableRetry 设为 true 是必须的，否则默认关闭 Retry 功能
```

```
EnableRetry: true,
//DelayLevelUtil 不是必须配置的，系统会有缺省值
DelayLevelUtil: pulsar.NewDelayLevelUtil("1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m"),
})
if err != nil {
log.Fatal(err)
}
defer consumer.Close()
```

在消息重试的时候，我们提供了两个接口，分别是异步和同步的方式来重试，示例如下：

```go
//同步的方式
err = consumer.ReconsumeLater(msg,pulsar.NewReconsumeOptionsWithLevel(2))
if err != nil{
log.Fatal(err)
}
//异步的方式，提供了一个回调方法，会在 Retry 消息发送出去后进行调用
consumer.ReconsumeLaterAsync(msg, pulsar.NewReconsumeOptionsWithLevel(2), func(id pulsar.MessageID,
message *pulsar.ProducerMessage, err error) {
if err != nil {
fmt.Printf("Error %v when send retry msg", err)
} else {
fmt.Printf("Retry message send success with id : %v", id)
}
})
```

TCP协议（Pulsar社区版）
Spring Boot Starter 接入

操作场景

本文以 Spring Boot Starter 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

操作步骤

步骤1：添加依赖

在项目中引入 Pulsar Starter 相关依赖。

```
<dependency>
    <groupId>io.github.majusko</groupId>
    <artifactId>pulsar-java-spring-boot-starter</artifactId>
    <version>1.0.7</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.projectreactor/reactor-core -->
<dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-core</artifactId>
    <version>3.4.11</version>
</dependency>
```

步骤2：准备配置

在配置文件中 添加 Pulsar 相关配置信息。

```
pulsar:
  # 命名空间名称
  namespace: namespace_java
  # 服务接入地址
  service-url: http://pulsar-xxx.tdmq.ap-gz.public.tencenttdmq.com:8080
  # 授权角色密钥
  token-auth-value: eyJrZXlJZC....
  # 集群 ID
  tenant: pulsar-xxx
```

参数	说明
namespace	命名空间名称，在控制台 命名空间 管理页面中复制。
service-url	集群接入地址，可以在控制台 集群管理 页面查看并复制。 
token-auth-value	角色密钥，在 角色管理 页面复制密钥列复制。 
tenant	集群 ID，在控制台 集群管理 页面中获取。

步骤3：生产消息

1. 生产者工厂配置。

```
@Configuration
public class ProducerConfiguration {
        @Bean
    public ProducerFactory producerFactory() {
            return new ProducerFactory()
                        // topic1 使用String类型生产者
                        .addProducer("topic1", String.class)
                        // topic2 使用byte[]类型(默认类型)生产者
                        .addProducer("topic2")
                        // topic3 使用MyMessage类型生产者 (自定义消息类型)
                        .addProducer("topic4", MyMessage.class);
    }
}.
```

2. 注入生产者。

```
@Autowired
private PulsarTemplate<byte[]> defaultProducer;  // byte[]类型生产者

@Autowired
private PulsarTemplate<String> stringProducer;   // String类型生产者

@Autowired
private PulsarTemplate<MyMessage> customProducer;  // MyMessage类型生产者 (自定义消息类型)
```

3. 发送消息。

```
// 发送String类型的消息
stringProducer.send("topic1", "Hello pulsar client.");

// 发送MyMessage类型消息 (自定义消息类型)
MyMessage myMessage = new MyMessage();
myMessage.setData("Hello client, this is a custom message.");
myMessage.setSendDate(new Date());
customProducer.send("topic4", myMessage);

// 发送byte[]类型消息
defaultProducer.send("topic2", ("Hello pulsar client, this is a order message" + i + ".").getBytes(StandardCharsets.UTF_
8));
```

> 注意：
>
> - 发送消息的 Topic 是在生产者配置中已经声明的 Topic。
> - PulsarTemplate 类型应与发送消息的类型一致。
> - 发送消息到指定 Topic 时，消息类型需要与生产者工厂配置中的 Topic 绑定的消息类型对应。

步骤4：消费消息

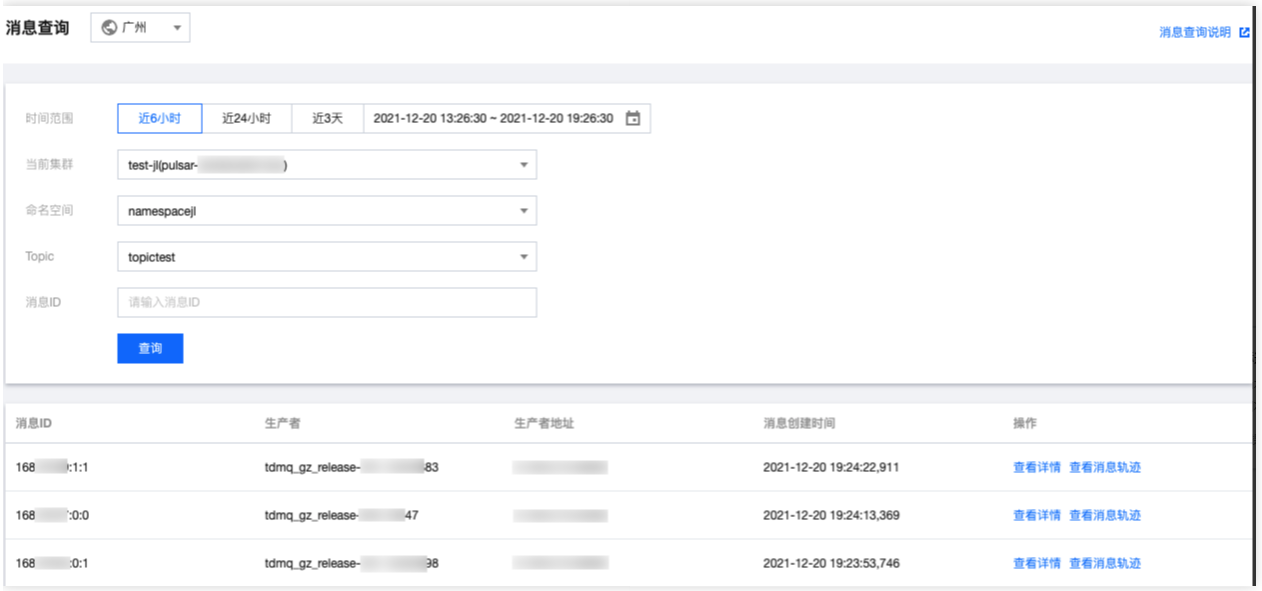
消费者配置。

```
@PulsarConsumer(topic = "topic1",  // 订阅topic名称
```

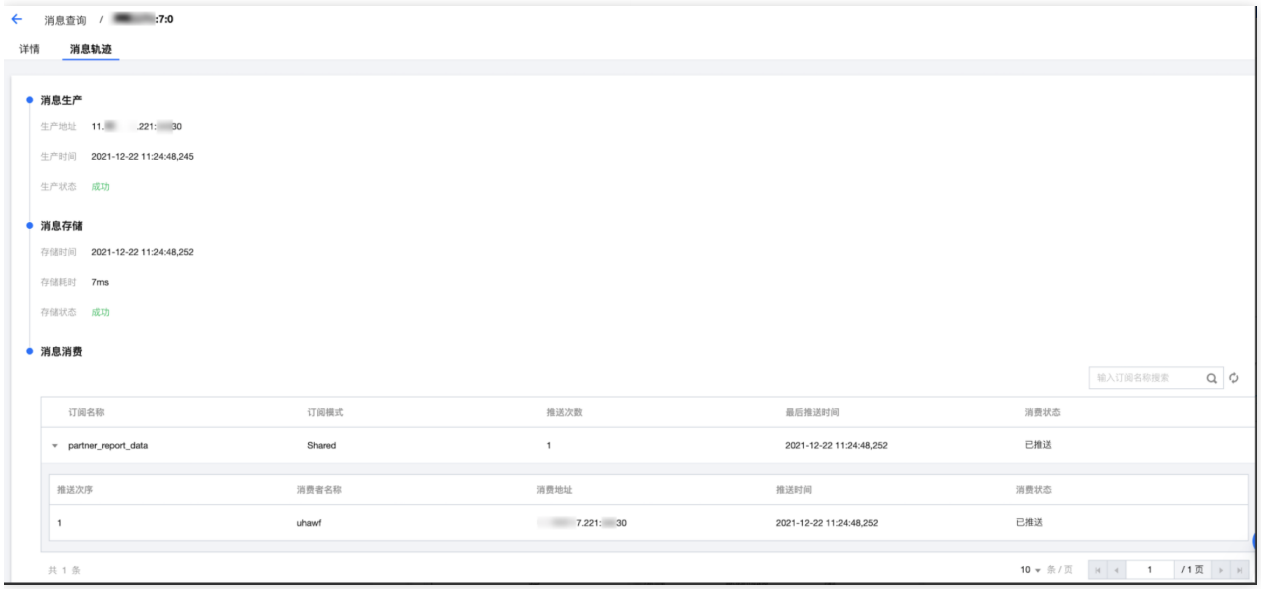
```
            subscriptionName = "sub_topic1", // 订阅名称
            clazz = String.class, // 消息类型，需要与生产者保持一致，绑定后不能修改类型
            serialization = Serialization.JSON, // 序列化方式
            subscriptionType = SubscriptionType.Shared, // 订阅模式，默认为独占模式
            consumerName = "firstTopicConsumer", // 消费者名称
            maxRedeliverCount = 3, // 最大重试次数
            deadLetterTopic = "sub_topic1-DLQ" // 死信topic名称
           )
public void topicConsume(String msg) {
    // TODO process your message
    System.out.println("Received a new message. content: [" + msg + "]");
    // 如果消费失败，请抛出异常，这样消息会进入重试队列，之后可以重新消费，直到达到最大重试次数之后，进入死信队列。前提是要创建重试和死信topic
}
```

步骤5：查询消息

登录控制台，进入 消息查询 页面，可查看 Demo 运行后的消息轨迹。



消息轨迹如下：



> 说明：
>
> 以上是基于 Springboot Starter 方式对 Pulsar 简单使用的配置。详细使用可参见 Demo 或 Starter 文档。

Java SDK

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

操作步骤

1. 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：

```
<dependency>
    <groupId>org.apache.pulsar</groupId>
    <artifactId>pulsar-client</artifactId>
    <version>2.9.4</version>
</dependency>
```

2. 创建 Pulsar 客户端。

```
PulsarClient pulsarClient = PulsarClient.builder()
            // 服务接入地址
            .serviceUrl("pulsar://" + SERVICE_URL)
            // 路由ID
            .listenerName("custom:" + ROUTERID)
            // 授权角色密钥
        .authentication(AuthenticationFactory.token(AUTHENTICATION)).build();
```

参数	说明
SERVICE_URL	集群接入地址，可以在控制台 集群管理 接入点页面查看并复制。 
AUTHENTICATION	角色密钥，在 角色管理 页面复制密钥列复制。

参数	说明
	
ROUTERID	

3. 创建生产者。

```
// 构建生产者
Producer<byte[]> producer = pulsarClient.newProducer()
      // 禁用掉batch功能
      .enableBatching(false)
      // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
      .topic(TOPICNAME).create();
```

> 说明：
>
> 参数 TOPICNAME 需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。

4. 发送消息。

```
//发送消息
MessageId msgId = producer.newMessage()
   // 消息内容
   .value("this is a new message.".getBytes(StandardCharsets.UTF_8))
   .send();
```

5. 资源释放。

```
// 关闭生产者
producer.close();
// 关闭客户端
pulsarClient.close();
```

6. 创建消费者。

```
// 构建byte[]类型（默认类型）的消费者
Consumer<byte[]> consumer = pulsarClient.newConsumer()
   // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
```

```
    .topic(TOPICNAME)
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName(SUBSCRIPTIONNAME)
    // 声明消费模式为share（共享）模式
    .subscriptionType(SubscriptionType.Shared)
    // 配置从最早开始消费，否则可能会消费不到历史消息
    .subscriptionInitialPosition(SubscriptionInitialPosition.Earliest)
    // 订阅
    .subscribe();
```

说明：

- 参数 SUBSCRIPTIONNAME 需要写入订阅名，可在消费管理界面查看。
- 参数 TOPICNAME 需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。



7. 消费消息。

```
// 接收当前offset对应的一条消息
Message<byte[]> msg = consumer.receive();
MessageId msgId = msg.getMessageId();
String value = new String(msg.getValue());
System.out.println("receive msg " + msgId + ",value:" + value);
// 接收到之后必须要ack，否则offset会一直停留在当前消息，导致消息积压
consumer.acknowledge(msg);
```

8. 使用监听器进行消费。

```
// 消息监听器
MessageListener<byte[]> myMessageListener = (consumer, msg) -> {
    try {
        System.out.println("Message received: " + new String(msg.getData()));
        // 回复ack
        consumer.acknowledge(msg);
    } catch (Exception e) {
        // 消费失败，回复nack
        consumer.negativeAcknowledge(msg);
    }
};
pulsarClient.newConsumer()
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    .topic("persistent://pulsar-mmqwr5xx9n7g/sdk_java/topic1")
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName("sub_topic1")
```

```
// 声明消费模式为exclusive（独占）模式
.subscriptionType(SubscriptionType.Exclusive)
// 设置监听器
.messageListener(myMessageListener)
// 配置从最早开始消费，否则可能会消费不到历史消息
.subscriptionInitialPosition(SubscriptionInitialPosition.Earliest)
.subscribe();
```

9. 登录 TDMQ Pulsar 版控制台，依次点击 Topic 管理 > Topic 名称进入消费管理页面，点开订阅名下方右三角号，可查看生产消费记录。



> 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 Pulsar 官方文档。

Go SDK

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

操作步骤

1. 在客户端环境引入 pulsar-client-go 库。
2. 在客户端环境执行如下命令下载 Pulsar 客户端相关的依赖包。

```
go get github.com/apache/pulsar-client-go/pulsar@v0.9.0
```

2. 安装完成后，即可通过以下代码引用到您的 Go 工程文件中。

```
import "github.com/apache/pulsar-client-go/pulsar"
```

2. 创建 Pulsar Client。

```
// 创建pulsar客户端
client, err := pulsar.NewClient(pulsar.ClientOptions{
    // 服务接入地址
    URL: serviceUrl,
    // 授权角色密钥
    Authentication:    pulsar.NewAuthenticationToken(authentication),
    ListenerName: RouterID
    OperationTimeout:  30 * time.Second,
    ConnectionTimeout: 30 * time.Second,
})

if err != nil {
    log.Fatalf("Could not instantiate Pulsar client: %v", err)
}

defer client.Close()
```

参数	说明
serviceUrl	集群接入地址，可以在控制台 集群管理 接入点页面查看并复制。

参数	说明
	
Authentication	角色密钥，在 角色管理 页面复制密钥列复制。 
RouterID	

3. 创建生产者。

```
// 使用客户端创建生产者
producer, err := client.CreateProducer(pulsar.ProducerOptions{
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
    Topic: "persistent://pulsar-mmqwr5xx9n7g/sdk_go/topic1",
})

if err != nil {
    log.Fatal(err)
}
defer producer.Close()
```

> 说明：
>
> Topic 名称需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。

4. 发送消息。

```
// 发送消息
_, err = producer.Send(context.Background(), &pulsar.ProducerMessage{
    // 消息内容
    Payload: []byte("hello go client, this is a message."),
```

```
    // 业务key
    Key: "yourKey",
    // 业务参数
    Properties: map[string]string{"key": "value"},
})
```

5. 创建消费者。

```
// 使用客户端创建消费者
consumer, err := client.Subscribe(pulsar.ConsumerOptions{
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
    Topic:            "persistent://pulsar-mmqwr5xx9n7g/sdk_go/topic1",
    // 订阅名称
    SubscriptionName: "topic1_sub",
    // 订阅模式
    Type:             pulsar.Shared,
})
if err != nil {
    log.Fatal(err)
}
defer consumer.Close()
```

说明：

- subscriptionName 需要写入订阅名，可在消费管理界面查看。
- Topic 名称需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。



6. 消费消息。

```
// 获取消息
msg, err := consumer.Receive(context.Background())
if err != nil {
    log.Fatal(err)
}
// 模拟业务处理
fmt.Printf("Received message msgId: %#v -- content: '%s'\n",
        msg.ID(), string(msg.Payload()))

// 消费成功，回复ack，消费失败根据业务需要选择回复nack或ReconsumeLater
consumer.Ack(msg)
```

7. 登录 TDMQ Pulsar 版控制台，依次点击 Topic 管理 > Topic 名称进入消费管理页面，点开订阅名下方右三角号，可查看生产消费记

录。



> 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 Pulsar 官方文档。

C++ SDK

操作场景

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

前提条件

- 完成资源创建与准备
- 安装 GCC

操作步骤

1. 准备环境。
   i. 在客户端环境安装 Pulsar C++ client，安装过程可参考官方教程 Pulsar C++ client。
   ii. 在项目中引入 Pulsar C++ client 相关头文件及动态库。
2. 创建客户端。

```
// 客户端配置信息
ClientConfiguration config;
// 设置授权角色密钥
AuthenticationPtr auth = pulsar::AuthToken::createWithToken(AUTHENTICATION);
std::string routerID = ROUTERID;
config.setAuth(auth);
config.setListenerName(routerID);
// 创建客户端
Client client(SERVICE_URL, config);
```

参数	说明
SERVICE_URL	集群接入地址，可以在控制台集群管理接入点页面查看并复制。 
AUTHENTICATION	角色密钥，在角色管理页面复制密钥列复制。 

参数	说明
ROUTERID	

3. 创建生产者。

```
// 生产者配置
ProducerConfiguration producerConf;
producerConf.setBlockIfQueueFull(true);
producerConf.setSendTimeout(5000);
// 生产者
Producer producer;
// 创建生产者
Result result = client.createProducer(
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
    "persistent://pulsar-xxx/sdk_cpp/topic1",
    producerConf,
    producer);
if (result != ResultOk) {
    std::cout << "Error creating producer: " << result << std::endl;
    return -1;
}
```

> 说明：
>
> Topic 名称需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。

4. 发送消息。

```
// 消息内容
std::string content = "hello cpp client, this is a msg";
// 构建消息对象
Message msg = MessageBuilder().setContent(content)
    .setPartitionKey("mykey")  // 业务key
    .setProperty("x", "1")  // 设置消息参数
    .build();
// 发送消息
Result result = producer.send(msg);
if (result != ResultOk) {
    // 发送失败
    std::cout << "The message " << content << " could not be sent, received code: " << result << std::endl;
} else {
    // 发送成功
    std::cout << "The message " << content << " sent successfully" << std::endl;
}
```

5. 创建消费者。

```
// 消费者配置信息
ConsumerConfiguration consumerConfiguration;
consumerConfiguration.setSubscriptionInitialPosition(pulsar::InitialPositionEarliest);
// 消费者
Consumer consumer;
// 订阅topic
Result result = client.subscribe(
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
    "persistent://pulsar-xxx/sdk_cpp/topic1",
    // 订阅名称
    "sub_topic1",
    consumerConfiguration,
    consumer);

if (result != ResultOk) {
    std::cout << "Failed to subscribe: " << result << std::endl;
    return -1;
}
```

说明：

- subscriptionName 需要写入订阅名，可在消费管理界面查看。
- Topic 名称需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。



6. 消费消息。

```
Message msg;
// 获取消息
consumer.receive(msg);
// 模拟业务
std::cout << "Received: " << msg << "  with payload '" << msg.getDataAsString() << "'" << std::endl;
// 回复ack
consumer.acknowledge(msg);
// 消费失败回复nack, 消息将会重新投递
// consumer.negativeAcknowledge(msg);
```

7. 登录 TDMQ Pulsar 版控制台，依次点击 Topic 管理 > Topic 名称进入消费管理页面，点开订阅名下方右三角号，可查看生产消费记录。



> 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 Pulsar 官方文档。

Python SDK

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

操作步骤

1. 准备环境。在客户端环境安装 pulsar-client 库，可以使用 pip 进行安装，也可以使用其他方式，参见 Pulsar Python client。

```
pip install pulsar-client==2.8.1
```

2. 创建客户端。

```
# 创建客户端
client = pulsar.Client(
   authentication=pulsar.AuthenticationToken(
      # 已授权角色密钥
      AUTHENTICATION),
   # 服务接入地址
   service_url=SERVICE_URL,
   # 接入点路由ID
   listener_name=ROUTERID)
```

参数	说明
SERVICE_URL	集群接入地址，可以在控制台集群管理接入点页面查看并复制。 
AUTHENTICATION	角色密钥，在角色管理页面复制密钥列复制。

参数	说明
	
ROUTERID	

3. 创建生产者。

```
# 创建生产者
producer = client.create_producer(
    # topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    topic='pulsar-xxx/sdk_python/topic1'
)
```

> 说明：
>
> Topic 名称需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。

4. 发送消息。

```
# 发送消息
producer.send(
    # 消息内容
    'Hello python client, this is a msg.'.encode('utf-8'),
    # 消息参数
    properties={'k': 'v'},
    # 业务key
    partition_key='yourKey'
)
```

还可以使用异步方式发送消息。

```
# 异步发送回调
def send_callback(send_result, msg_id):
    print('Message published: result:{}  msg_id:{}'.format(send_result, msg_id))

# 发送消息
producer.send_async(
    # 消息内容
    'Hello python client, this is a async msg.'.encode('utf-8'),
    # 异步回调
    callback=send_callback,
```

```
    # 消息配置
    properties={'k': 'v'},
    # 业务key
    partition_key='yourKey'
)
```

5. 创建消费者。

```
# 订阅消息
consumer = client.subscribe(
    # topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    topic='pulsar-xxx/sdk_python/topic1',
    # 订阅名称
    subscription_name='sub_topic1'
)
```

说明：

- subscriptionName 需要写入订阅名，可在消费管理界面查看。
- Topic 名称需要填入完整路径，即 persistent://clusterid/namespace/Topic，clusterid/namespace/topic 的部分可以从控制台上 Topic管理 页面直接复制。



6. 消费消息。

```
# 获取消息
msg = consumer.receive()
try:
    # 模拟业务
    print("Received message '{}' id='{}'".format(msg.data(), msg.message_id()))
    # 消费成功，回复ack
    consumer.acknowledge(msg)
except:
    # 消费失败，消息将会重新投递
    consumer.negative_acknowledge(msg)
```

7. 登录 TDMQ Pulsar 版控制台，依次点击 Topic 管理 > Topic 名称进入消费管理页面，点开订阅名下方右三角号，可查看生产消费记录。



> 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 Pulsar 官方文档。

Node.js SDK

操作场景

TDMQ Pulsar 版2.7.1及以上版本的集群已支持 Pulsar 社区版 Node.js SDK。本文介绍如何使用 Pulsar 社区版 Node.js SDK 完成接入。

前提条件

- 获取接点地址
- 在 TDMQ Pulsar 版控制台【集群管理】页面复制接入地址。
- 获取密钥
- 已参考 【角色与鉴权】 文档配置好了角色与权限，并获取到了对应角色的密钥（Token）。

操作步骤

1. 按照 Pulsar 官方文档在您客户端所在的环境中安装 Node.js Client。

```
$ npm install pulsar-client
```

2. 在创建 Node.js Client 的代码中，配置准备好的接入地址和密钥。

```
const Pulsar = require("pulsar-client");

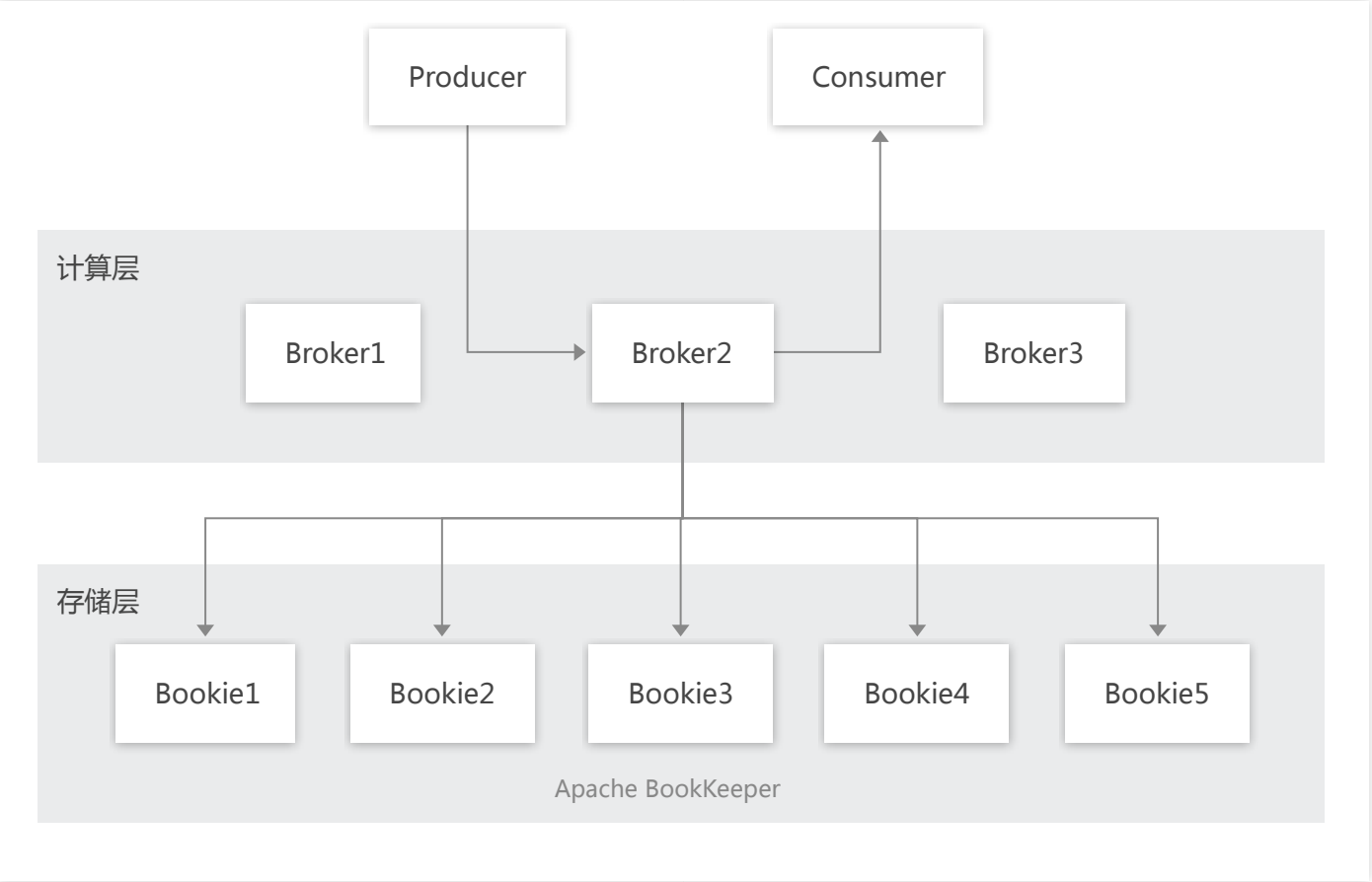
(async () => {
  const client = new Pulsar.Client({
    serviceUrl: "http://*",   //更换为接入地址（控制台集群管理页完整复制）
    authentication: new Pulsar.AuthenticationToken({
      token: "eyJh**",       //更换为密钥
    }),
  });
  await client.close();
})();
```

关于 Pulsar 社区版 Node.js SDK 各种功能的使用方式，请参考Pulsar 官方文档。

开发指南
原理解析
Pulsar Topic和分区

Apache Pulsar 架构

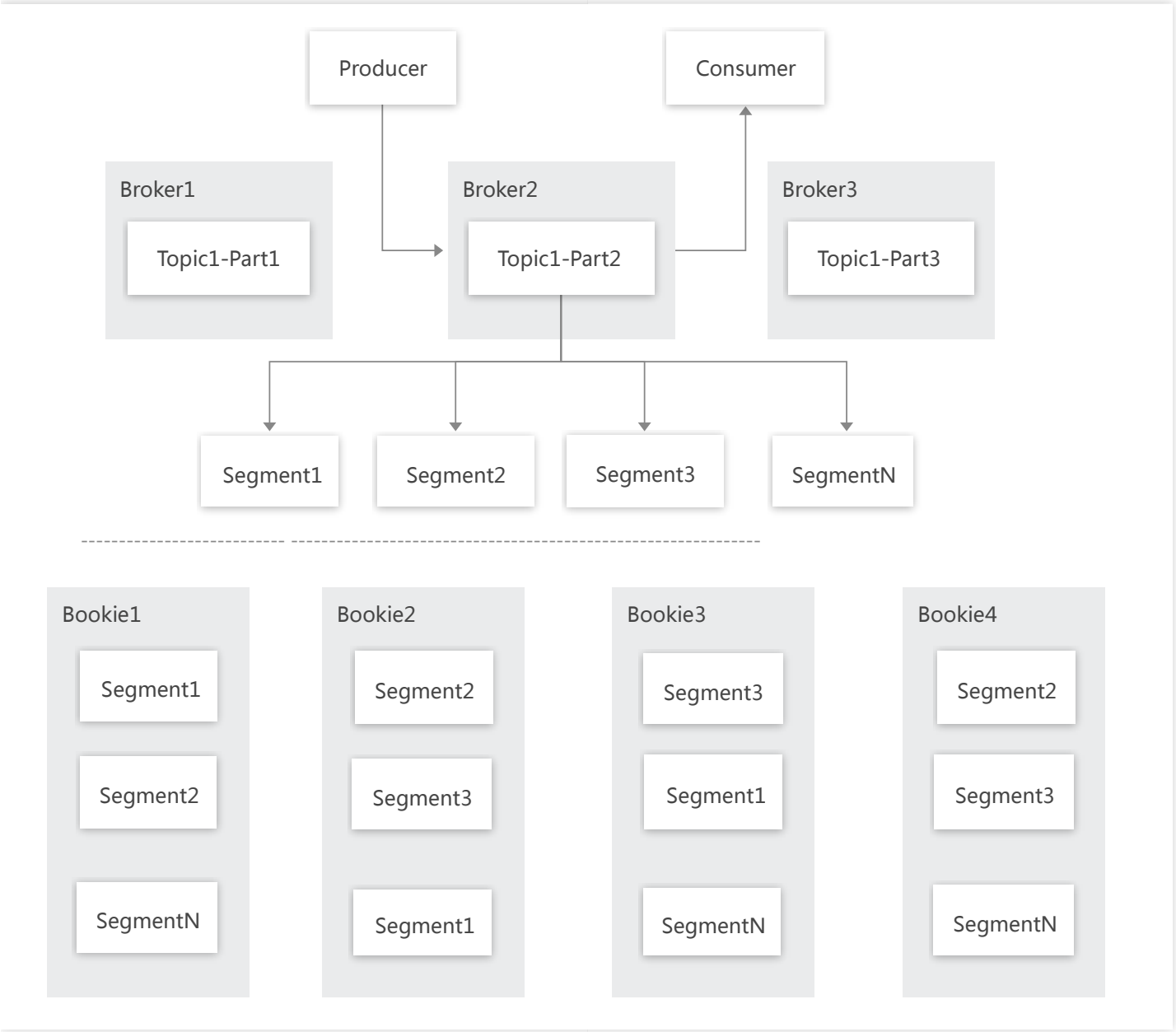
Apache Pulsar 是一个发布-订阅模型的消息系统，由 Broker、Apache BookKeeper、Producer、Consumer 等组件组成。



- Producer ： 消息的生产者，负责发布消息到 Topic。
- Consumer：消息的消费者，负责从 Topic 订阅消息。
- Broker：无状态服务层，负责接收和传递消息，集群负载均衡等工作，Broker 不会持久化保存元数据，因此可以快速的上、下线。
- Apache BookKeeper：有状态持久层，由一组 Bookie 存储节点组成，可以持久化地存储消息。
  Apache Pulsar 在架构设计上采用了计算与存储分离的模式，消息发布和订阅相关的计算逻辑在 Broker 中完成，数据存储在 Apache BookKeeper 集群的 Bookie 节点上。

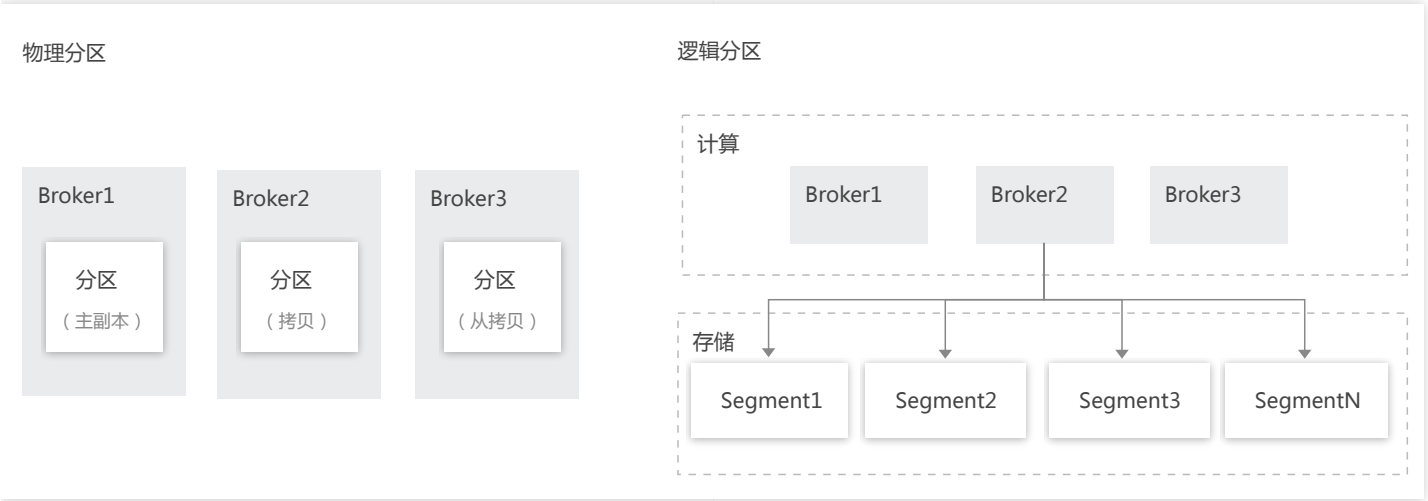
Topic 与分区

Topic（主题）是某一种分类的名字，消息在 Topic 中可以被存储和发布。生产者往 Topic 中写消息，消费者从 Topic 中读消息。
Pulsar 的 Topic 分为 Partitioned Topic 和 Non-Partitioned Topic 两类，Non-Partitioned Topic 可以理解为一个分区数为1的 Topic。实际上在 Pulsar 中，Topic 是一个虚拟的概念，创建一个3分区的 Topic，实际上是创建了3个“分区Topic”，发给这个 Topic 的消息会被发往这个 Topic 对应的多个 “分区Topic”。
例如：生产者发送消息给一个分区数为3，名为 my-topic 的 Topic，在数据流向上是均匀或者按一定规则（如果指定了key）发送给了 my-topic-partition-0 、 my-topic-partition-1 和 my-topic-partition-2 三个“分区 Topic”。
分区 Topic 做数据持久化时，分区是逻辑上的概念，实际存储的单位是分片（Segment）的。
如下图所示，分区 Topic1-Part2 的数据由N个 Segment 组成， 每个 Segment 均匀分布并存储在 Apache BookKeeper 群集中的多个 Bookie 节点中， 每个 Segment 具有3个副本。



物理分区与逻辑分区

逻辑分区和物理分区对比如下：



物理分区：计算与存储耦合，容错需要拷贝物理分区，扩容需要迁移物理分区来达到负载均衡。
逻辑分区：物理“分片”，计算层与存储层隔离，这种结构使得 Apache Pulsar 具备以下优点。

- Broker 和 Bookie 相互独立，方便实现独立的扩展以及独立的容错。
- Broker 无状态，便于快速上、下线，更加适合于云原生场景。
- 分区存储不受限于单个节点存储容量。
- 分区数据分布均匀，单个分区数据量突出不会使整个集群出现木桶效应。
- 存储不足扩容时，能迅速利用新增节点平摊存储负载。

消息副本与存储机制

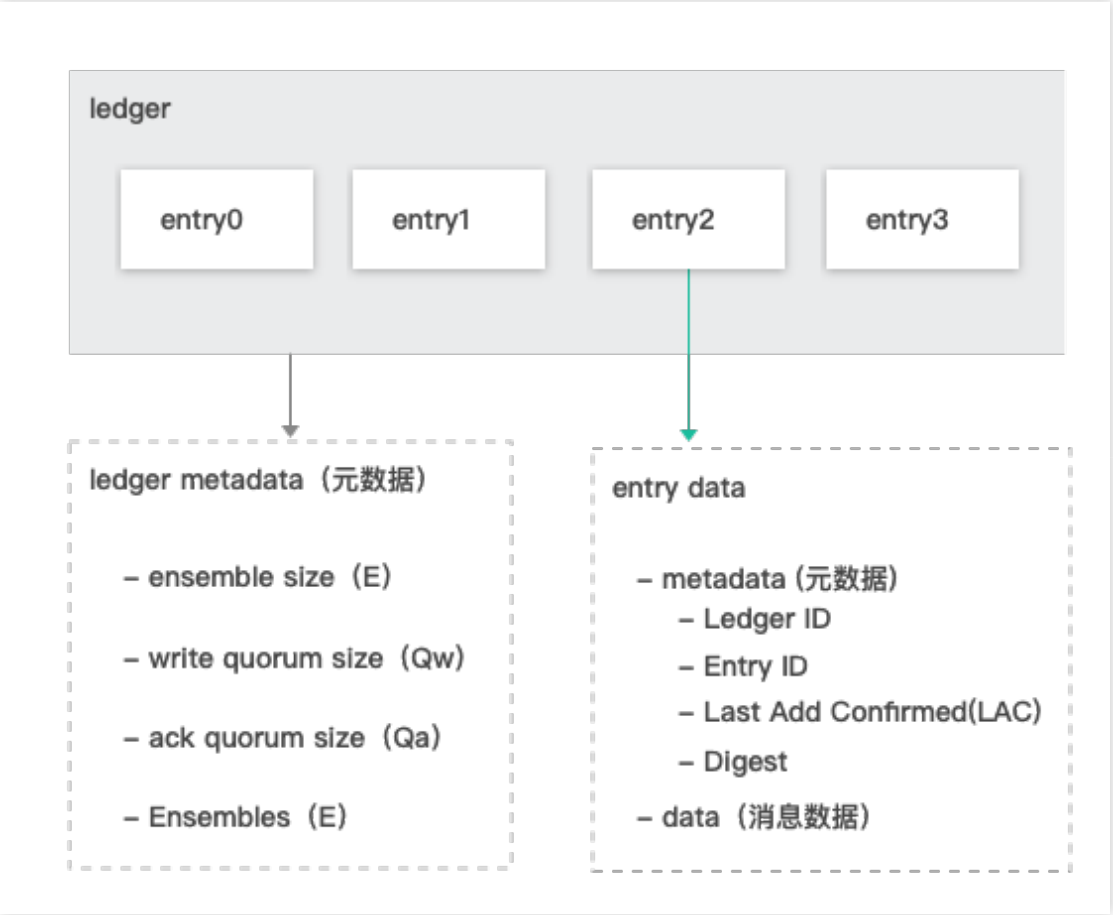
消息元数据组成

Pulsar 中每个分区 Topic 的消息数据以 ledger 的形式存储在 BookKeeper 集群的 bookie 存储节点上，每个 ledger 包含一组 entry，而 bookie 只会按照 entry 维度进行写入、查找、获取。

> 注意：
>
> 批量生产消息的情况下，一个 entry 中可能包含多条消息，所以 entry 和消息并不一定是一一对应的。

Ledger 和 entry 分别对应不同的元数据。

- ledger 的元数据存储在 zk 上。
- entry 除了消息数据部分之外，还包含元数据，entry 的数据存储在 bookie 存储节点上。



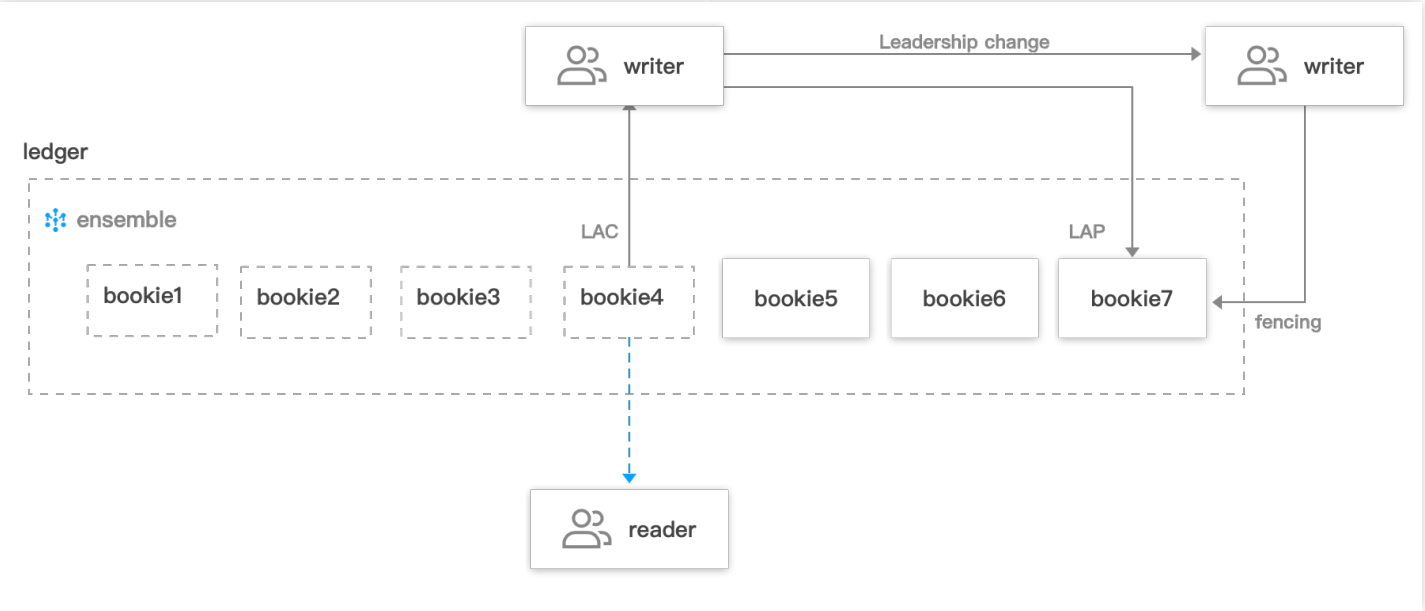
类型	参数	参数说明	数据存放位置

ledger	ensemble size（E）	每个 ledger 选用的 bookie 节点的个数	元数据存储在 zk 上
	write quorum size（Qw）	每个 entry 需要向多少个 bookie 发送写入请求	
	ack quorum size（Qa）	收到多少个写入确认后，即可认为写入成功	
	Ensembles（E）	使用的 ensemble 列表，形式为<entry id,="" ensembles=""> 元组 key（entry id）：使用这个 ensembles 列表开始时的 entry id value（ensembles）：ledger 选用的 bookie ip 列表，每个 value 中包含 ensemble size（E）个 IP 每个 ledger 可能包含多个 ensemble 列表，同一时刻每个 ledger 最多只有一个 ensembles 列表在使用	
Entry	Ledger ID	entry 所在的 ledger id	数据存储在 bookie 存储节点上
	Entry ID	当前 entry id	
	Last Add Confirmed	创建当前 entry 的时候，已知最新的写入确认的 entry id	
	Digest	CRC	

每个 ledger 在创建的时候，会在现有的 BookKeeper 集群中的可写状态的 bookie 候选节点列表中，选用 ensemble size 对应个数的 bookie 节点，如果没有足够的候选节点则会抛出 BKNotEnoughBookiesExceptio 异常。选出候选节点后，将这些信息组成 <entry id, ensembles> 元组，存储到 ledger 的元数据里的 ensembles 中。

消息副本机制

消息写入流程



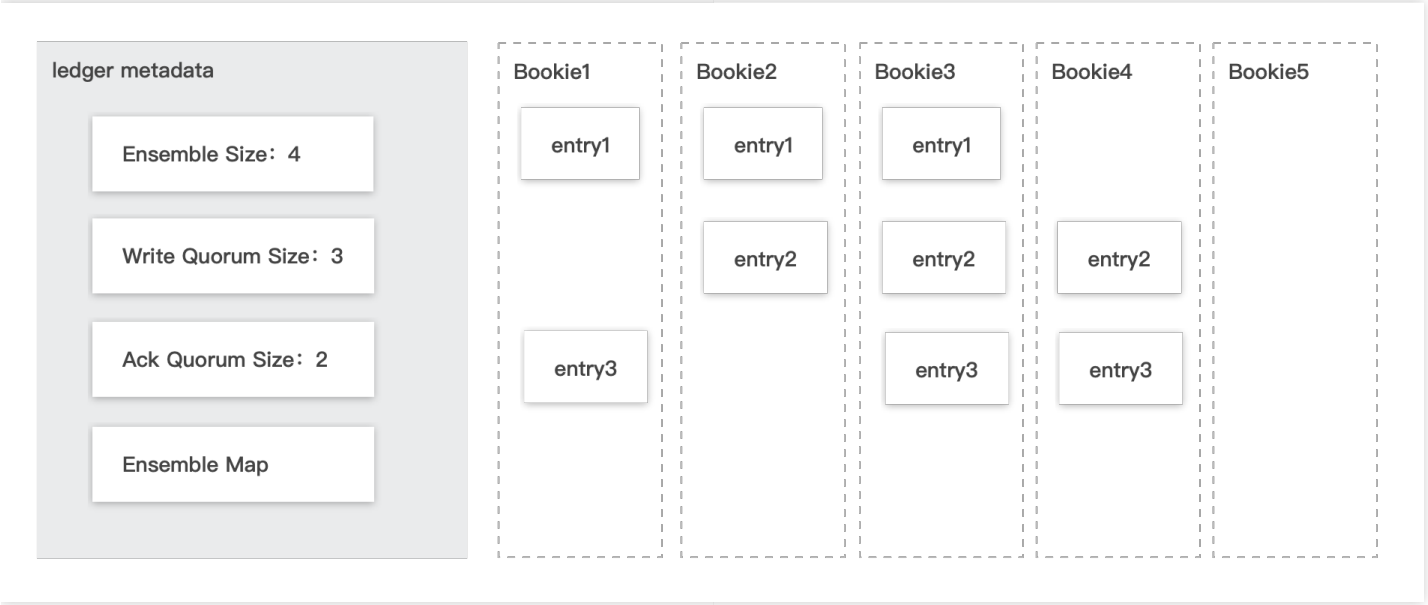
客户端在写入消息时，每个 entry 会向 ledger 当前使用的 ensemble 列表中的 Qw 个 bookie 节点发送写入请求，当收到 Qa 个写确认后，即认为当前消息写入存储成功。同时会通过 LAC（LastAddConfirmed）和 LAP（lastAddPushed）分别标识当前推送的位置和已经收到存储确认的位置。

每个正在推送的 entry 中的 LAC 元数据值，为当前时刻创建发送 entry 请求时，已经收到最新的确认位置值。LAC 所在位置及之前的消息对读客户端是可见的。

同时，pulsar 通过 fencing 机制，来避免同时有多个客户端对同一个 ledger 进行写操作。这里主要适用于一个 topic/partition 的归属关系从一个 broker 变迁到另一个 broker 的场景。

消息副本分布

每个 entry 写入时，会根据当前消息的 entry id 和当前使用的 ensembles 列表的开始 entry id（即key值），计算出在当前 entry 需要使用 ensemble 列表中由哪组 Qw 个 bookie 节点进行写入。之后，broker 会向这些 bookie 节点发送写请求，当收到 Qa 个写确认后，即认为当前消息写入存储成功。这时至少能够保证 Qa 个消息的副本个数。



如上图所示，ledger 选用了4个 bookie 节点（bookie1-4 这4个节点），每次写入3个节点，当收到2个写入确认即代表消息存储成功。当前 ledger 选中的 ensemble 从 entry 1开始，使用 bookie1、bookie2、bookie3 进行写入，写入 entry 2的时候选用 bookie2、bookie3、bookie4写入，而 entry 3 则会根据计算结果，写入 bookie3、bookie4、bookie1。

消息恢复机制

Pulsar 的 BookKeeper 集群中的每个 bookie 在启动的时候，默认自动开启 recovery 的服务，这个服务会进行如下几个事情：

1. auditorElector 审计选举。
2. replicationWorker 复制任务。
3. deathWatcher 宕机监控。
   BookKeeper 集群中的每个 bookie 节点，会通过 zookeeper 的临时节点机制进行选主，主 bookie 主要处理如下几个事情：
4. 负责监控 bookie 节点的变化。
5. 到 zk 上面标记出宕机的 bookie 上面的 ledger 为 Underreplicated 状态。
6. 检查所有的 ledger 的副本数（默认一周一个周期）。
7. Entry 副本数检查（默认未开启）。
   其中 ledger 中的数据是按照 Fragment 维度进行恢复的（每个 Fragment 对应 ledger 下的一组 ensemble 列表，如果一个 ledger 下有多个 ensemble 列表，则需要处理多个 Fragment）。
   在进行恢复时，首先要判断出当前的 ledger 中的哪几个 Fragment 中的哪些存储节点需要用新的候选节点进行替换和恢复数据。当 Fragment 中关联的部分 bookie 节点上面没有对应的 entry 数据（默认是按照首、尾 entry 是否存在判断），则这个 bookie 节点需要被替换，当前的这个 Fragment 需要进行数据恢复。
   Fragment 的数据用新的 bookie 节点进行数据恢复完毕后，更新 ledger 的元数据中当前 Fragment 对应的

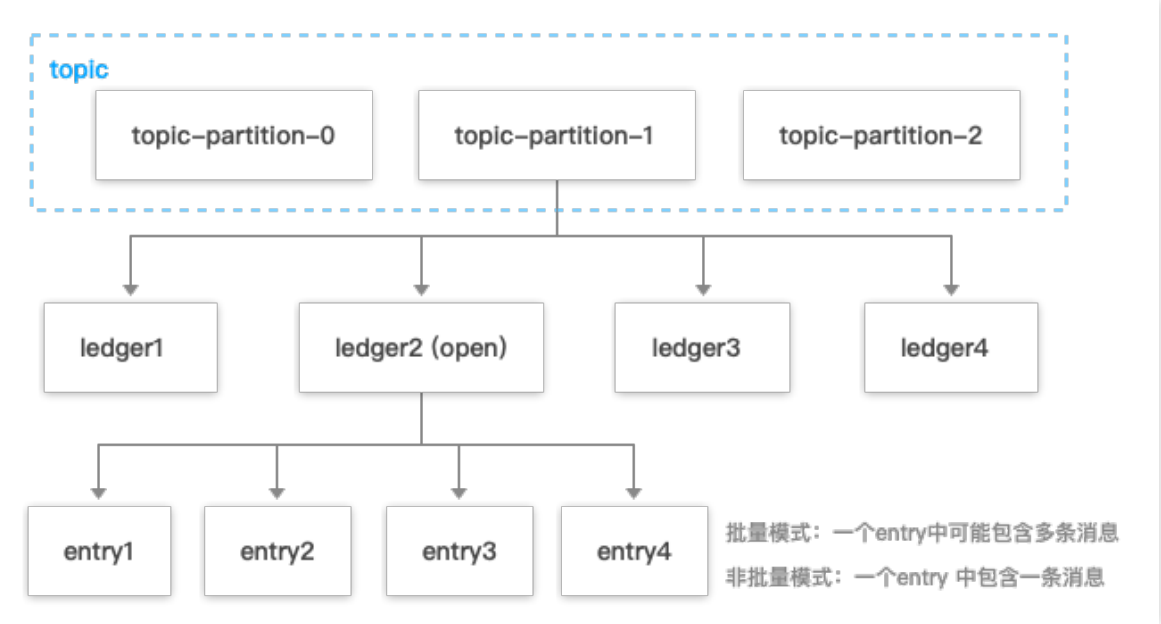
ensemble 列表的原数据。
经过此过程，因 bookie 节点宕机引起的数据副本数减少的场景，数据的副本数会逐步的恢复成 Qw（后台指定的副本数，TDMQ 默认3副本）个。

消息存储原理与ID规则

消息 ID 生成规则

在 Pulsar 中，每条消息都有自己的 ID（即 MessageID），MessageID 由四部分组成： ledgerId:entryID:partition-index:batch-index 。其中：

- partition-index：指分区的编号，在非分区 topic 的时候为 -1。
- batch-index：在非批量消息的时候为 -1。
  消息 ID 的生成规则由 Pulsar 的消息存储机制决定，Pulsar 中消息存储原理图如下：



如上图所示，在 Pulsar中，一个 Topic 的每一个分区会对应一系列的 ledger，其中只有一个 ledger 处于 open 状态即可写状态，而每个 ledger 只会存储与之对应的分区下的消息。
Pulsar 在存储消息时，会先找到当前分区使用的 ledger ，然后生成当前消息对应的 entry ID，entry ID 在同一个 ledger 内是递增的。每个 ledger 存在的时长或保存的 entry 个数超过阈值后会进行切换，新的消息会存储到同一个 partition 中的下一个 ledger 中。

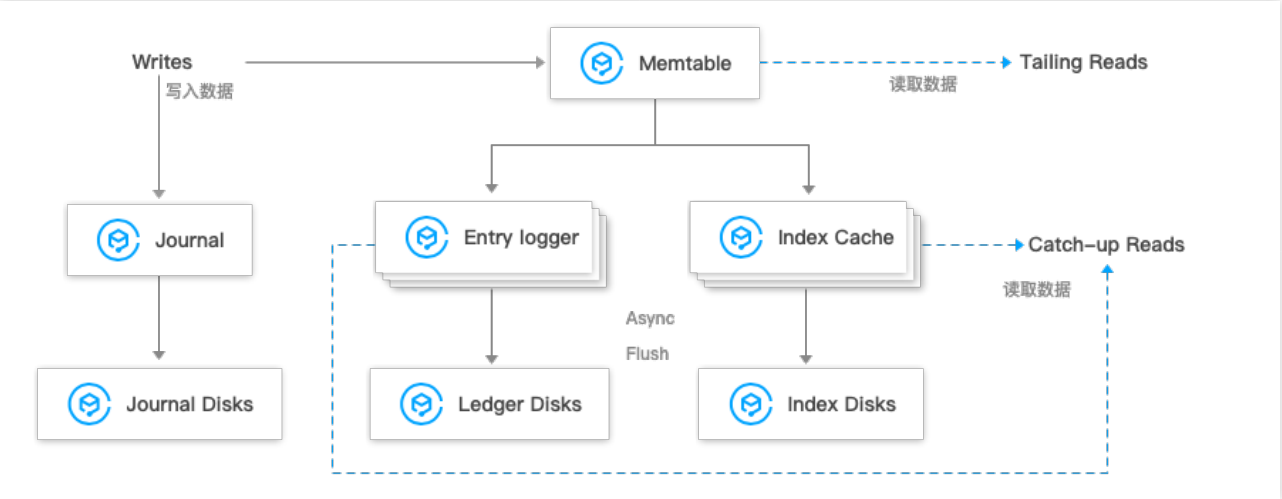
- 批量生产消息情况下，一个 entry 中可能包含多条消息。
- 非批量生产的情况下，一个 entry 中包含一条消息（producer 端可以配置这个参数，默认是批量的）。
  Ledger 只是一个逻辑概念，是数据的一种逻辑组装维度，并没有对应的实体。而 bookie 只会按照 entry 维度进行写入、查找、获取。

分片机制详解：Legder 和 Entry

Pulsar 中的消息数据以 ledger 的形式存储在 BookKeeper 集群的 bookie 存储节点上。Ledger 是一个只追加的数据结构，并且只有一个写入器，这个写入器负责多个 bookie 的写入。Ledger 的条目会被复制到多个 bookie 中，同时会写入相关的数据来保

证数据的一致性。
BookKeeper 需要保存的数据包括：

- Journals
  - journals 文件里存储了 BookKeeper 的事务日志，在任何针对 ledger 的更新发生前，都会先将这个更新的描述信息持久化到这个 journal 文件中。
  - BookKeeper 提供有单独的 sync 线程根据当前 journal 文件的大小来作 journal 文件的 rolling。
- EntryLogFile
  - 存储真正数据的文件，来自不同 ledger 的 entry 数据先缓存在内存buffer中，然后批量flush到EntryLogFile中。
  - 默认情况下，所有ledger的数据都是聚合然后顺序写入到同一个EntryLog文件中，避免磁盘随机写。
- Index 文件
  - 所有 Ledger 的 entry 数据都写入相同的 EntryLog 文件中，为了加速数据读取，会作 ledgerId + entryId 到文件 offset 的映射，这个映射会缓存在内存中，称为 IndexCache。
  - IndexCache 容量达到上限时，会被 sync 线程 flush 到磁盘中。
    三类数据文件的读写交互如下图：



Entry 数据写入

1. 数据首先会同时写入 Journal（写入 Journal 的数据会实时落到磁盘）和 Memtable（读写缓存）。
2. 写入 Memtable 之后，对写入请求进行响应。
3. Memtable 写满之后，会 flush 到 Entry Logger 和 Index cache，Entry Logger 中保存数据，Index cache 中保存数据的索引信息，
4. 后台线程将 Entry Logger 和 Index cache 数据落到磁盘。
   Entry 数据读取

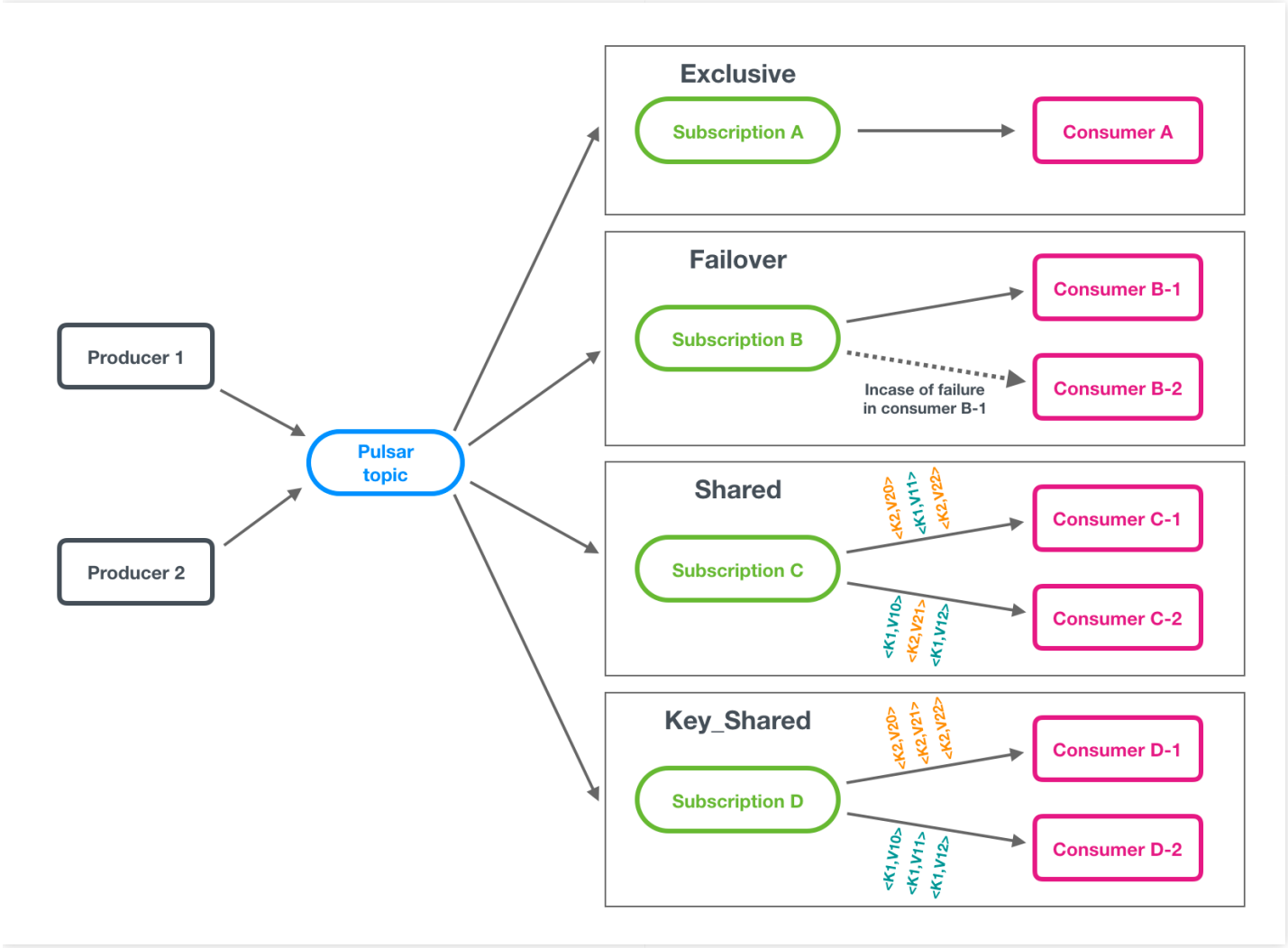
- Tailing read 请求：直接从 Memtable 中读取 Entry。
- Catch-up read（滞后消费）请求：先读取 Index信息，然后索引从 Entry Logger 文件读取 Entry。
  数据一致性保证：LastLogMark
- 写入的 EntryLog 和 Index 都是先缓存在内存中，再根据一定的条件周期性的 flush 到磁盘，这就造成了从内存到持久化到磁盘的时间间隔，如果在这间隔内 BookKeeper 进程崩溃，在重启后，我们需要根据 journal 文件内容来恢复，这个 LastLogMark 就记录了从 journal 中什么位置开始恢复。
- 它其实是存在内存中，当 IndexCache 被 flush 到磁盘后其值会被更新，LastLogMark 也会周期性持久化到磁盘文件，供

Bookkeeper 进程启动时读取来从 journal 中恢复。

- LastLogMark 一旦被持久化到磁盘，即意味着在其之前的 Index 和 EntryLog 都已经被持久化到了磁盘，那么 journal 在这 LastLogMark 之前的数据都可以被清除了。

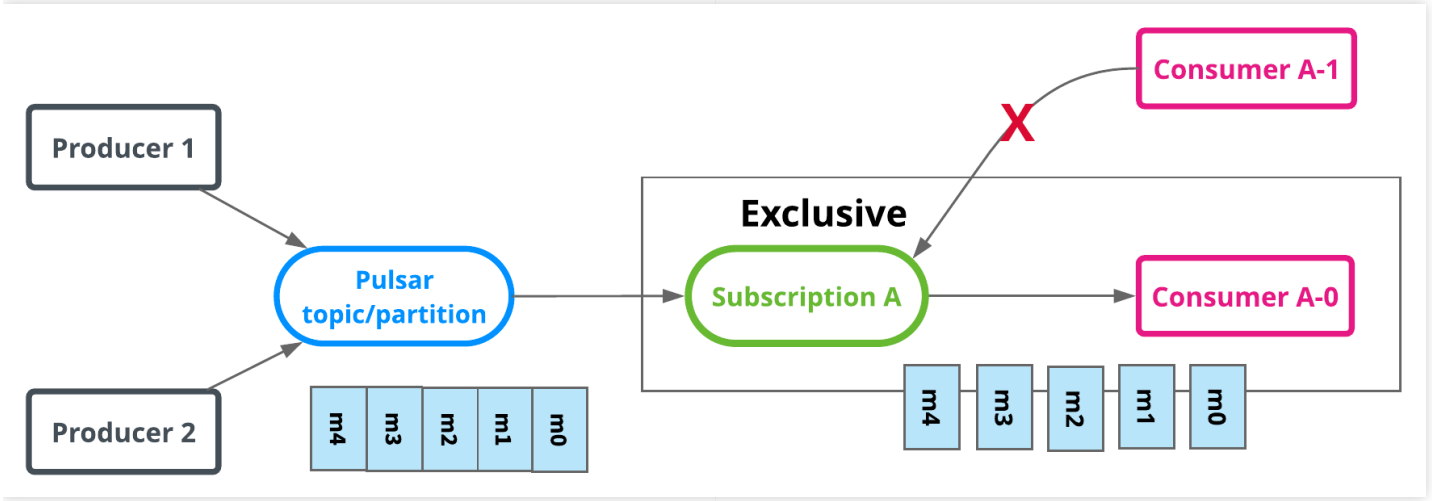
使用实践
订阅模式

为了适用不同场景的需求，Pulsar 支持四种订阅模式：Exclusive、Shared、Failover、Key_Shared。



独占模式（Exclusive）

Exclusive 独占模式（默认模式）：一个 Subscription 只能与一个 Consumer 关联，只有这个 Consumer 可以接收到 Topic 的全部消息，如果该 Consumer 出现故障了就会停止消费。
Exclusive 订阅模式下，同一个 Subscription 里只有一个 Consumer 能消费 Topic，如果多个 Consumer 订阅则会报错，适用于全局有序消费的场景。



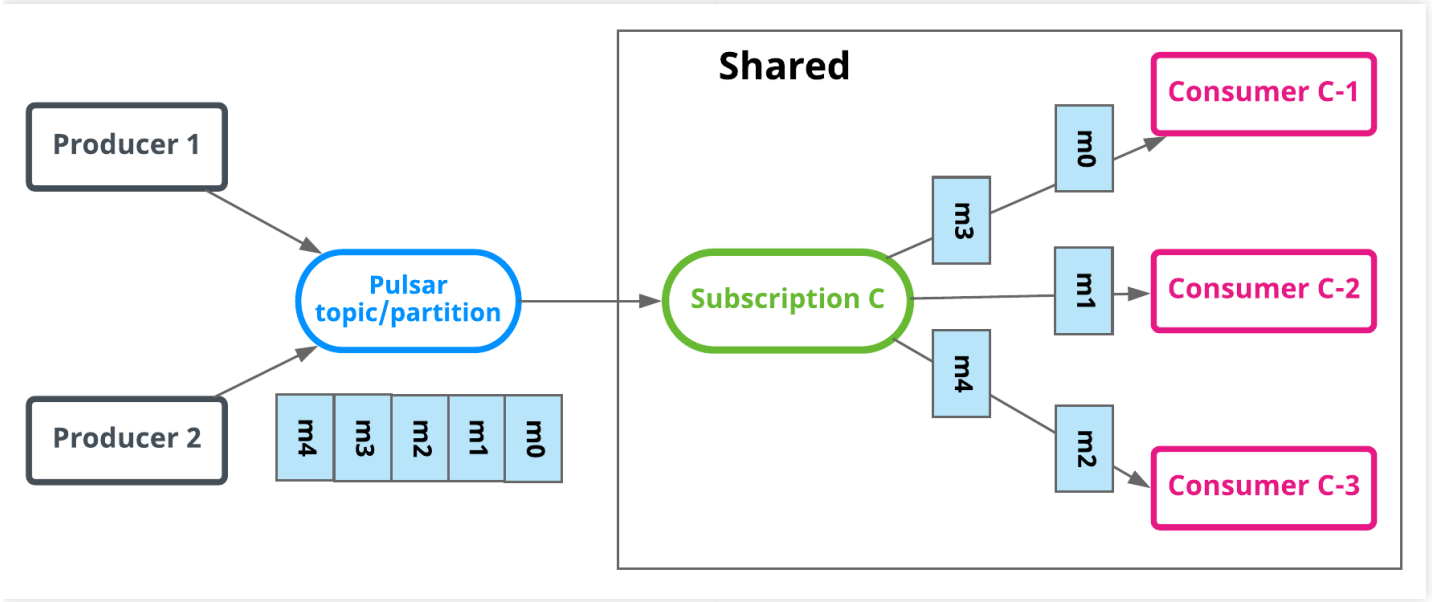

```
// 构建消费者
Consumer<byte[]> consumer = pulsarClient.newConsumer()
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    .topic("persistent://pulsar-xxx/sdk_java/topic1")
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName("sub_topic1")
    // 声明消费模式为exclusive（独占）模式
    .subscriptionType(SubscriptionType.Exclusive)
    .subscribe();
```

启动多个消费者将收到错误信息如下图所示：

```
                                        \java.exe ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.                                  for further details.
>> pulsar client created.
Exception in thread "main" org.apache.pulsar.client.api.PulsarClientException$ConsumerBusyException: Failed to subscribe persistent
Exclusive consumer is already connected
    at org.apache.pulsar.client.api.PulsarClientException.unwrap(PulsarClientException.java:946)
    at org.apache.pulsar.client.impl.ConsumerBuilderImpl.subscribe(ConsumerBuilderImpl.java:102)
    at com.tencent.cloud.tdmq.pulsar.simple.ListenerConsumer.main(ListenerConsumer.java:41)
```

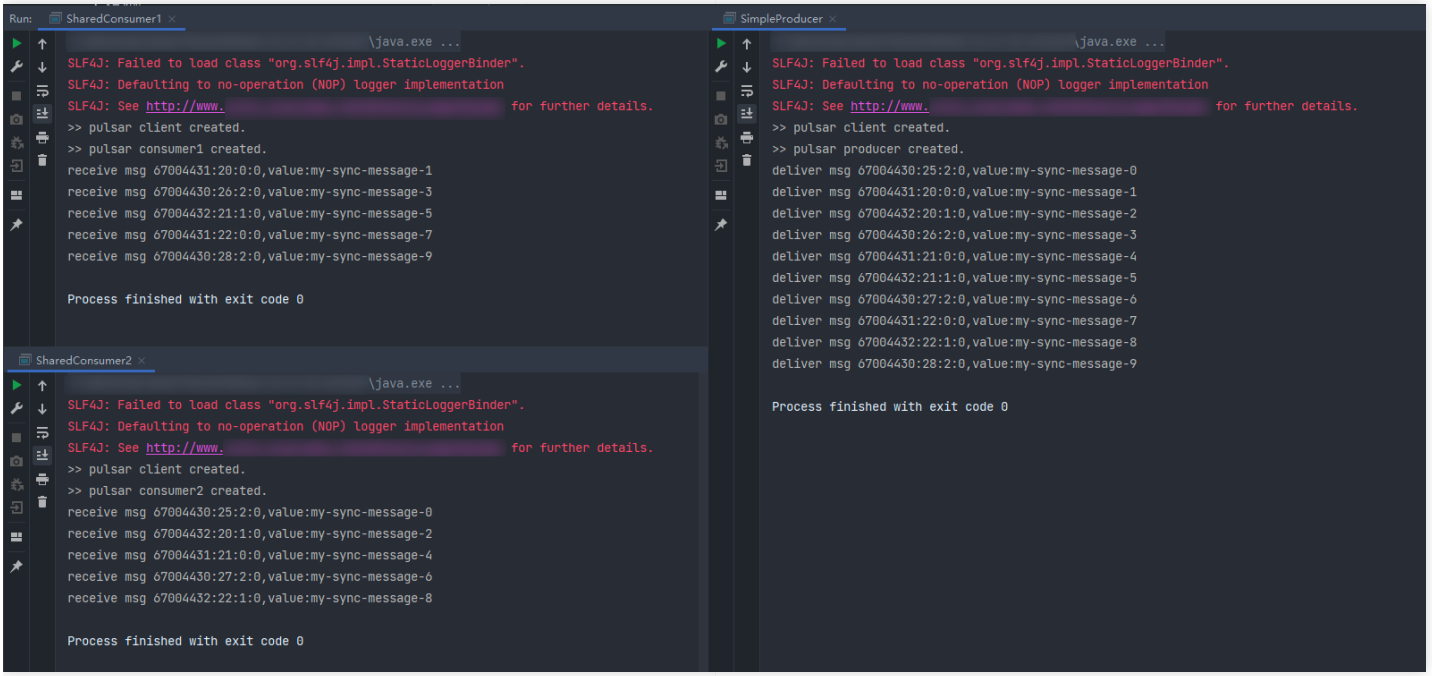
共享模式（Shared）

消息通过 round robin 轮询机制（也可以自定义）分发给不同的消费者，并且每个消息仅会被分发给一个消费者。当消费者断开连接，所有被发送给他，但没有被确认的消息将被重新安排，分发给其它存活的消费者。



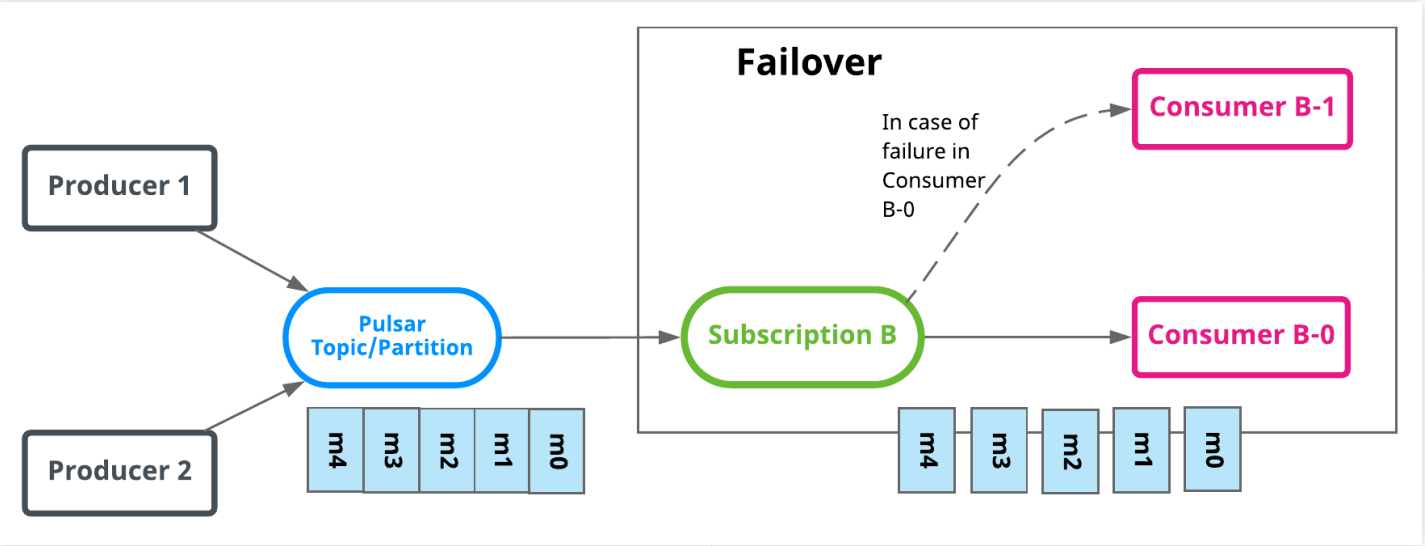

```
// 构建消费者
Consumer<byte[]> consumer = pulsarClient.newConsumer()
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    .topic("persistent://pulsar-xxx/sdk_java/topic1")
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName("sub_topic1")
    // 声明消费模式为 Shared（共享）模式
    .subscriptionType(SubscriptionType.Shared)
    .subscribe();
```

多个 Shared 模式消费者如下图所示：



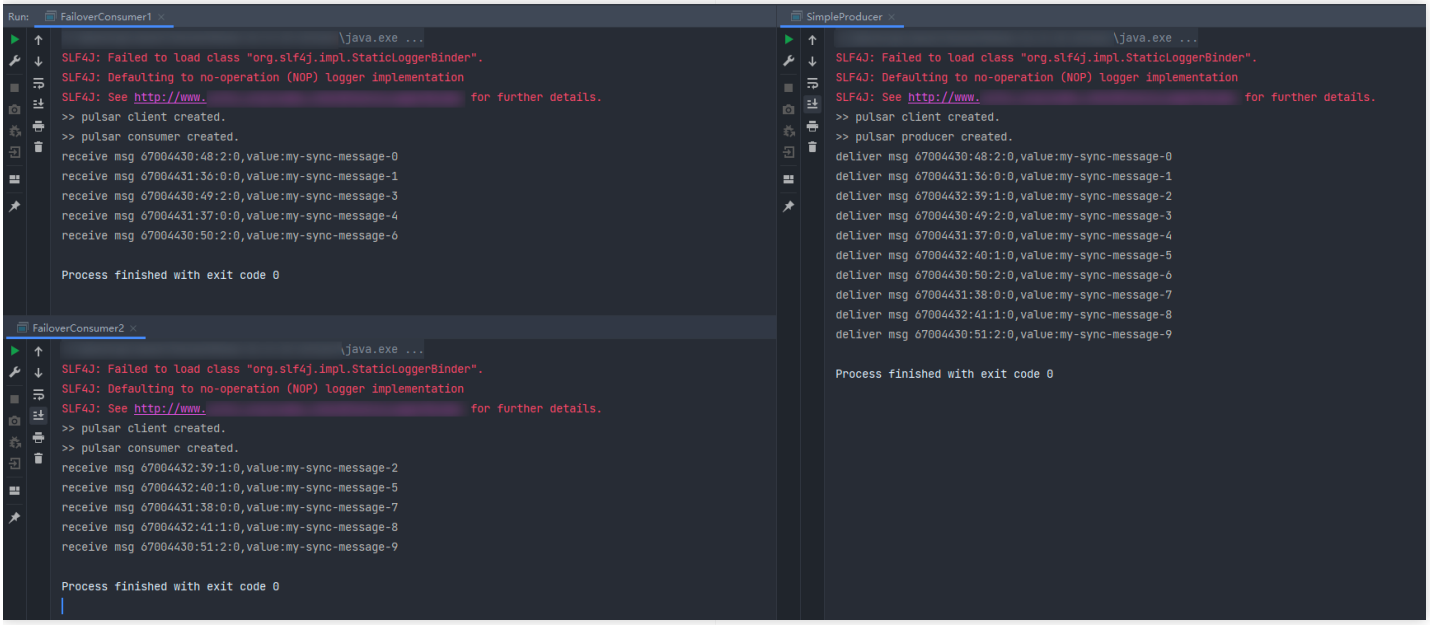
灾备模式（Failover）

当存在多个 consumer 时，将会按字典顺序排序，第一个 consumer 被初始化为唯一接受消息的消费者。当第一个 consumer 断开时，所有的消息（未被确认和后续进入的）将会被分发给队列中的下一个 consumer。



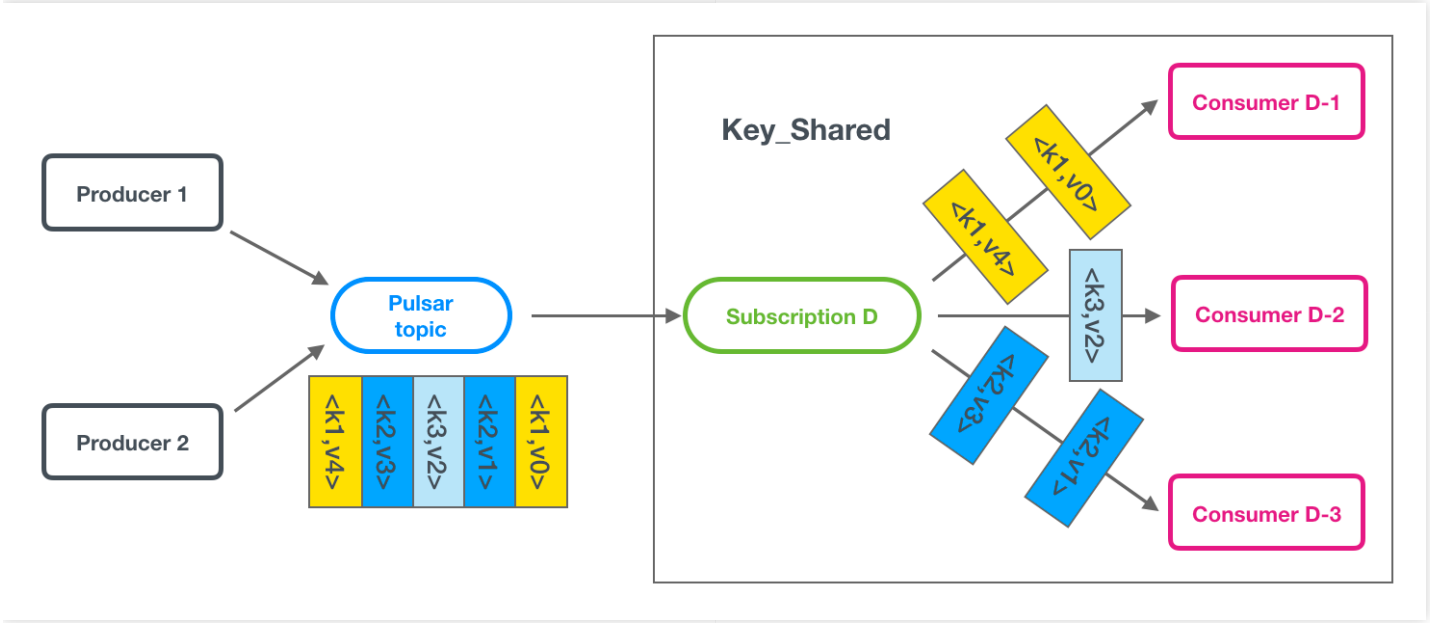

```
// 构建消费者
Consumer<byte[]> consumer = pulsarClient.newConsumer()
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    .topic("persistent://pulsar-xxx/sdk_java/topic1")
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName("sub_topic1")
    // 声明消费模式为灾备模式
    .subscriptionType(SubscriptionType.Failover)
    .subscribe();
```

多个 Failover 模式消费者如下图所示：



KEY 共享模式（Key_Shared）

当存在多个 Consumer 时，将根据消息的 Key 进行分发，Key 相同的消息只会被分发到同一个消费者。



注意：

- Key_Shared 本身在使用上存在一定的限制条件，由于其工程实现复杂度较高，在社区版本迭代中，不断有对 Key_Shared 的功能进行改进以及优化，整体稳定性相较 Exclusive,Failover 和 Shared 这三种订阅类型偏弱。如果上述三种订阅类型能满足业务需要，可以优先选用上述三种订阅类型。
- 专业集群可以保证相同 KEY 的消息按顺序投递；虚拟集群无法保障消息投递顺序。

Key_Shared 使用建议

什么时候才考虑用 Key_Shared 订阅模式

如是普通的生产消费场景，建议直接选用 Shared 模式即可。
若需要让相同 Key 的消息分给同一个消费者，这个时候 Shared 订阅模式无法满足用户需求。有两种方式可以选择：

- 选择 Key_Shared 订阅模式。
- 通过多分区主题 + Failover 订阅模式实现。

什么场景下适合用 Key_Shared 订阅

- Key 数量多且每个 Key 的消息分布相对均匀
- 消费处理速度快，无消息堆积的情况
  如果在生产过程中不能保证上面的两个条件同时满足，建议用 【多分区主题 + Failover 订阅】

代码示例

Key_Shared 订阅示例

默认情况下，Pulsar 在生产消息时是开启 Batch 功能的，Pulsar 的 Batch 消息解析是在 Consumer 侧处理的。所以在 Broker 侧一个 Batch 消息是被当作一条 Entry 处理的，所以对于 Key_shared 的基于消息 Key 有序订阅类型来说，是没办法处理这种 Case 的，因为不同 Key 的消息有可能被打包到同一个 Batch 中。针对这种情况在创建 Producer 时有如下两种规避方式：

1. 禁用 Batch。

```
// 构建生产者
Producer<byte[]> producer pulsarClient.newProducer()
        .topic(topic)
        .enableBatching(false)
        .create();
// 发送消息时设置key
MessageId msgId = producer.newMessage()
 // 消息内容
 .value(value.getBytes(StandardCharsets.UTF_8))
 // 在此处设置key，key相同的消息只会被分发到同一个消费者。
 .key("youKey1")
 .send();
```

2. 使用 key_based batch 类型。

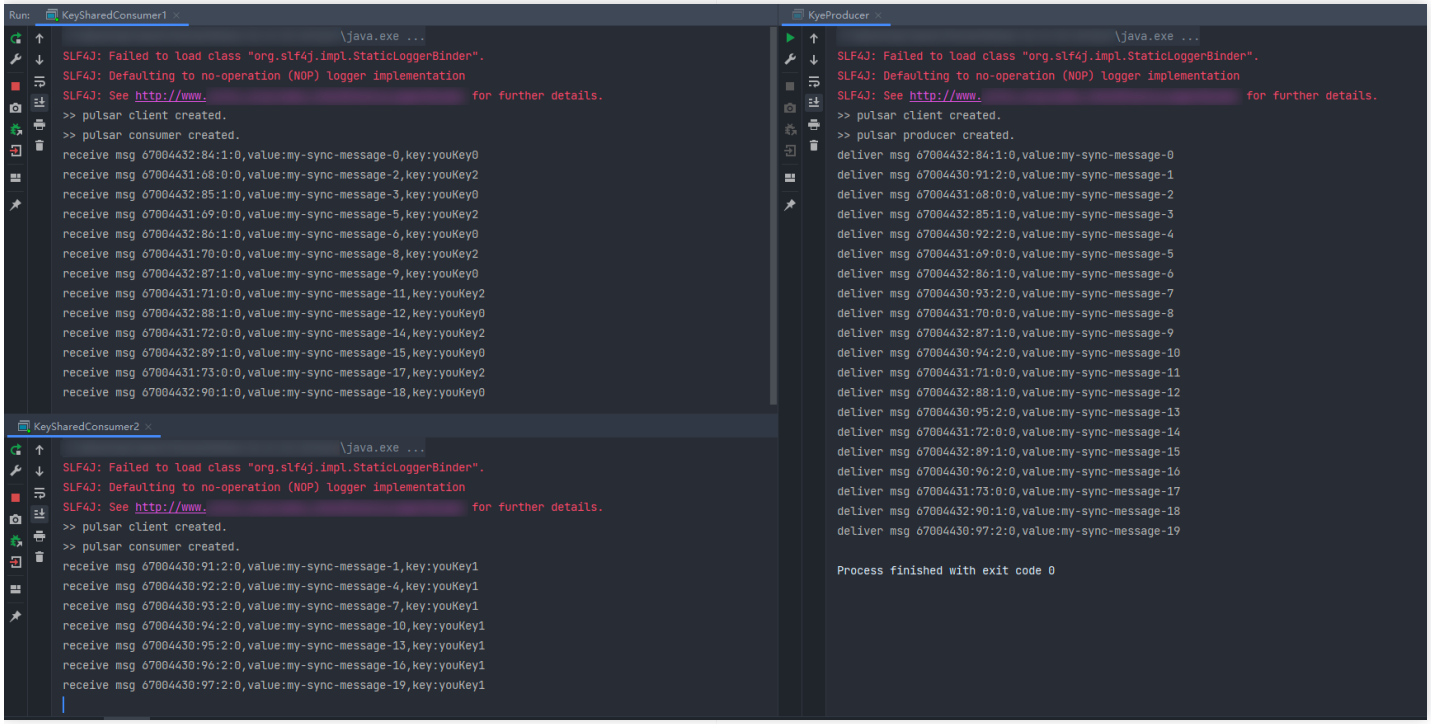
```
// 构建生产者
Producer<byte[]> producer = pulsarClient.newProducer()
        .topic(topic)
```

```
            .enableBatching(true)
            .batcherBuilder(BatcherBuilder.KEY_BASED)
            .create();
    // 发送消息时设置key
    MessageId msgId = producer.newMessage()
     // 消息内容
     .value(value.getBytes(StandardCharsets.UTF_8))
     // 在此处设置key，key相同的消息只会被分发到同一个消费者。
     .key("youKey1")
     .send();
```

消费者代码示例：

```
    // 构建消费者
    Consumer<byte[]> consumer = pulsarClient.newConsumer()
       // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
       .topic("persistent://pulsar-xxx/sdk_java/topic1")
       // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
       .subscriptionName("sub_topic1")
       // 声明消费模式为 Key_Shared（Key 共享）模式
       .subscriptionType(SubscriptionType.Key_Shared)
       .subscribe();
```

多个 Key_Shared 模式消费者。



多分区主题 + Failover 订阅示例

注意事项：

- 在该模式下，每个分区同时只会分配给一个消费者实例。若消费者数量多于分区数量，超出数量的消费者无法参与消息，可以通过扩容分区数量不小于消费者数量解决。
- 在设计 Key 的时候尽量保证 Key 分布均匀。
- Failover 模式下不支持延时消息。

1. 生产者代码示例

```
// 构建生产者
Producer<byte[]> producer pulsarClient.newProducer()
        .topic(topic)
        .enableBatching(false)  // 禁用batch
        .create();
// 发送消息时设置key
MessageId msgId = producer.newMessage()
 // 消息内容
 .value(value.getBytes(StandardCharsets.UTF_8))
 // 在此处设置key，key相同的消息会发送到同一个分区中
 .key("youKey1")
 .send();
```

2. 消费者代码示例

```
// 构建消费者
Consumer<byte[]> consumer = pulsarClient.newConsumer()
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    .topic("persistent://pulsar-xxx/sdk_java/topic1")
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName("sub_topic1")
    // 声明消费模式为Failover模式
    .subscriptionType(SubscriptionType.Failover)
    .subscribe();
```

定时和延时消息

相关概念

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

适用场景

如果系统是一个单体架构，则通过业务代码自己实现延时或利用第三方组件实现基本没有差别；一旦架构复杂起来，形成了一个大型分布式系统，有几十上百个微服务，这时通过应用自己实现定时逻辑会带来各种问题。一旦运行着延时程序的某个节点出现问题，整个延时的逻辑都会受到影响。
针对以上问题，利用延时消息的特性投递到消息队列里，便是一个较好的解决方案，能统一计算延时时间，同时重试和死信机制确保消息不丢失。
具体场景的示例如下：

- 微信红包发出后，生产端发送一条延时24小时的消息，到了24小时消费端程序收到消息，进行用户是否已经领走红包的判断，如果没有则退还到原账户。
- 小程序下单某商品后，后台存放一条延时30分钟的消息，到时间之后消费端收到消息触发对支付结果的判断，如果没有支付就取消订单，这样就实现了超过30分钟未完成支付就取消订单的逻辑。
- 微信上用户将某条信息设置待办后，也可以通过发送一条定时消息，服务端到点主动消费这条定时消息，对用户进行待办项提醒。

使用方式

在 TDMQ Pulsar 版的 SDK 中提供了专门的 API 来实现定时消息和延时消息。

- 对于定时消息，您需要提供一个消息发送的时刻。
- 对于延时消息，您需要提供一个时间长度作为延时的时长。

定时消息

定时消息通过生产者 producer 的 deliverAt() 方法实现，代码示例如下：

```
String value = "message content";
```

```
try {
       //需要先将显式的时间转换为 Timestamp
     long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2020-11-11 00:00:0
0").getTime();
     //通过调用 producer 的 deliverAt 方法来实现定时消息
      MessageId msgId = producer.newMessage()
            .value(value.getBytes())
            .deliverAt(timeStamp)
            .send();
} catch (ParseException e) {
       //TODO 添加对 Timestamp 解析失败的处理方法
       e.printStackTrace();
}
```

> 注意
>
> - 定时消息的时间范围为当前时间开始计算，864000秒（10天）以内的任意时刻。如10月1日12:00开始，最长可以设置到10月11日12:00。
> - 定时消息不可以使用 batch 模式发送，请在创建 producer 的时候把 enableBatch 参数设为 false 。
> - 定时消息的消费模式仅支持使用 Shared 模式进行消费，否则会失去定时效果（Key-shared 也不支持）。

延时消息

延时消息通过生产者 produce 的 deliverAfter() 方法实现，代码示例如下：

```
String value = "message content";

//需要指定延时的时长
long delayTime = 10L;
//通过调用 producer 的 deliverAfter 方法来实现定时消息
MessageId msgId = producer.newMessage()
   .value(value.getBytes())
   .deliverAfter(delayTime, TimeUnit.SECONDS) //单位可以自由选择
   .send();
```

> 注意
>
> - 延时消息的时长取值范围为0 - 864000秒（0秒 - 10天）。如10月1日12:00开始，最长可以设置864000秒。
> - 延时消息不可以使用 batch 模式发送，请在创建 producer 的时候把 enableBatch 参数设为 false 。
> - 延时消息的消费模式仅支持使用 Shared 模式进行消费，否则会失去延时效果（Key-shared 也不支持）。

使用说明和限制

- 使用定时或延迟消息时，建议与普通消息使用不同的 Topic 来管理，即定时与延迟消息发送到一个固定的 Topic，普通消息发送到另一个 Topic 中，方便后续的管理与维护，增加稳定性。
- 使用定时和延时两种类型的消息时，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）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。
- 定时和延时消息不支持 batch 模式（批量发送），batch 模式会引起消息堆积，保险起见，请在创建 producer 的时候把 enableBatch 参数设为 false 。
- 定时和延时消息的消费模式仅支持使用 Shared 模式进行消费，否则会失去定时或延时效果（Key-shared 也不支持）。
- 关于定时和延时消息的时间范围，最大均为10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。
- 设定定时时间后，TTL 的时间依旧会从发送消息的时间点开始算消息的最长保留时间；例如定时到2天后发送，消息最长保留（TTL）如果设置为1天的话，则消息在1天后会被删除，这个时候要确保 TTL 的时间要大于延时的时间，即 TTL 设置成大于等于2天，否则 TTL 到期时，消息会被删除。延时消息同理。
- 普通类型 Topic 支持收发定时/延

消息标签过滤

本文主要介绍 TDMQ Pulsar 版中消息标签过滤的功能、应用场景和使用方式。

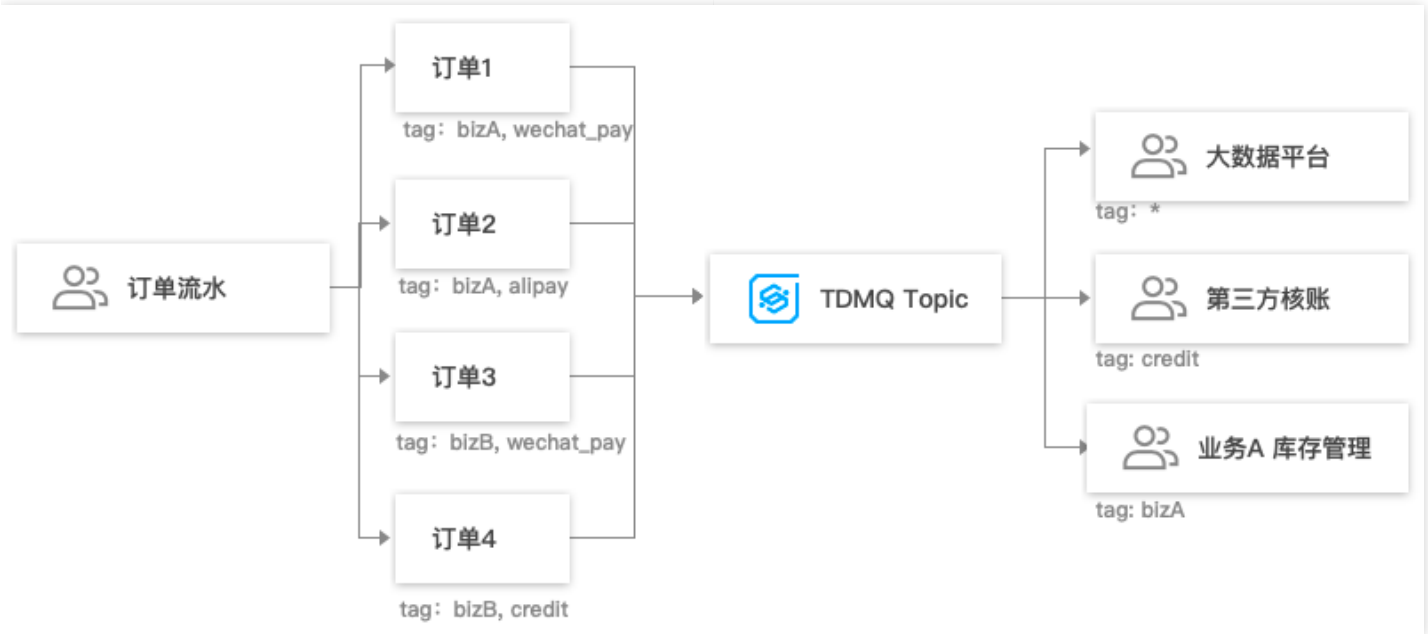
功能介绍

Tag，即消息标签，用于对某个Topic下的消息进行分类。TDMQ Pulsar 版的生产者在发送消息时，指定消息的 Tag，消费者需根据已经指定的 Tag 来进行订阅。
消费者订阅 Topic 时若未设置 Tag，Topic 中的所有消息都将被投递到消费端进行消费。

> 注意
>
> 在一个订阅中，单个消费者可以使用多个 Tag，多个 Tag 之间的关系是「或」，不同消费者使用的 Tag 需要是相同的。

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 Pulsar 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。



使用说明

Tag 消息目前是通过 Properties 的方式传入的，可以通过如下方式获取：
Java:

```
<dependency>
  <groupId>org.apache.pulsar</groupId>
  <artifactId>pulsar-client</artifactId>
  <version>2.10.3</version> <!-- 推荐版本 -->
</dependency>
```

Go：

```
go get -u github.com/apache/pulsar-client-go@master
```

Tag 消息使用限制

- Tag 消息不支持 Batch 功能，Batch 功能默认是开启的。如果要使用 Tag 消息，需要在 Producer 侧禁用掉 batch，具体如下：

```
// 构建生产者
Producer<byte[]> producer = pulsarClient.newProducer()
      // 禁用掉batch功能
      .enableBatching(false)
      // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称
      .topic("persistent://pulsar-xxx/sdk_java/topic2").create();
```

```
producer, err := client.CreateProducer(pulsar.ProducerOptions{
   DisableBatching: true, // 禁用掉batch功能
})
```

- tag 消息的过滤只针对已设置 tag 的消息，未设置 tag 的消息，不在过滤范围内。即未设置 tag 的消息会推送给所有的订阅者。
- 如果要开启 Tag 消息，需要发送消息的时候，在 ProducerMessage 中设置 Properties 字段；同时在创建 Consumer 的时候需要在 ConsumerOptions 中指定 SubscriptionProperties 字段。
- 在 ProducerMessage 中设置 Properties 字段时，其中 key 为 tag 的名字，value 为固定值： TAGS 。
- 在 ConsumerOptions 中指定 SubscriptionProperties 字段时，其中 key 为要订阅的 tag 的名字，value 为 tag 的版本信息，为保留字段，目前没有实质含义，用来做后续功能的扩展，具体如下：
    - 指定单个 tag

```
// 发送消息
MessageId msgId = producer.newMessage()
```

```
    .property("tag1", "TAGS")
    .value(value.getBytes(StandardCharsets.UTF_8))
      .send();

// 订阅相关参数，可用来设置订阅标签(TAG)
HashMap<String, String> subProperties = new HashMap<>();
 subProperties.put("tag1","1");
 // 构建消费者
 Consumer<byte[]> consumer = pulsarClient.newConsumer()
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    .topic("persistent://pulsar-xxxx/sdk_java/topic2")
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName("topic_sub1")
    // 声明消费模式为共享模式
    .subscriptionType(SubscriptionType.Shared)
    // 订阅相关参数，tag订阅等。。
    .subscriptionProperties(subProperties)
     // 配置从最早开始消费，否则可能会消费不到历史消息
     .subscriptionInitialPosition(SubscriptionInitialPosition.Earliest).subscribe();


  // 发送消息
  if msgId, err := producer.Send(ctx, &pulsar.ProducerMessage{
            Payload: []byte(fmt.Sprintf("hello-%d", i)),
    Properties: map[string]string{
        "tag1": "TAGS",
    },
      }); err != nil {
            log.Fatal(err)
    }

// 创建 consumer
    consumer, err := client.Subscribe(pulsar.ConsumerOptions{
            Topic:            "topic-1",
            SubscriptionName: "my-sub",
            SubscriptionProperties: map[string]string{"tag1": "1"},
     })
```

- 指定多个 tag

```
  // 发送消息
    MessageId msgId = producer.newMessage()
      .property("tag1", "TAGS")
      .property("tag2", "TAGS")
     .value(value.getBytes(StandardCharsets.UTF_8))
      .send();
```

```
// 订阅相关参数，可用来设置订阅标签(TAG)
HashMap<String, String> subProperties = new HashMap<>();
subProperties.put("tag1","1");
subProperties.put("tag2","1");
// 构建消费者
Consumer<byte[]> consumer = pulsarClient.newConsumer()
   // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
   .topic("persistent://pulsar-xxxx/sdk_java/topic2")
   // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
   .subscriptionName("topic_sub1")
   // 声明消费模式为共享模式
   .subscriptionType(SubscriptionType.Shared)
   // 订阅相关参数，tag订阅等。。
   .subscriptionProperties(subProperties)
   // 配置从最早开始消费，否则可能会消费不到历史消息
   .subscriptionInitialPosition(SubscriptionInitialPosition.Earliest).subscribe();
```

```
// 创建 producer
if msgId, err := producer.Send(ctx, &pulsar.ProducerMessage{
        Payload: []byte(fmt.Sprintf("hello-%d", i)),
          Properties: map[string]string{
                "tag1": "TAGS",
                "tag2": "TAGS",
        },
   }); err != nil {
        log.Fatal(err)
   }
```

```
// 创建 consumer
   consumer, err := client.Subscribe(pulsar.ConsumerOptions{
          Topic:            "topic-1",
          SubscriptionName: "my-sub",
          SubscriptionProperties: map[string]string{
                "tag1": "1",
                "tag2": "1",
          },
   })
```

- tag 与 properties 混合

```
// 发送消息
MessageId msgId = producer.newMessage()
   .property("tag1", "TAGS")
   .property("tag2", "TAGS")
```

```
        .property("xxx", "yyy")
    .value(value.getBytes(StandardCharsets.UTF_8))
        .send();

// 订阅相关参数，可用来设置订阅标签(TAG)
HashMap<String, String> subProperties = new HashMap<>();
subProperties.put("tag1","1");
subProperties.put("tag2","1");
// 构建消费者
Consumer<byte[]> consumer = pulsarClient.newConsumer()
    // topic完整路径，格式为persistent://集群（租户）ID/命名空间/Topic名称，从【Topic管理】处复制
    .topic("persistent://pulsar-xxxx/sdk_java/topic2")
    // 需要在控制台Topic详情页创建好一个订阅，此处填写订阅名
    .subscriptionName("topic_sub1")
    // 声明消费模式为共享模式
    .subscriptionType(SubscriptionType.Shared)
    // 订阅相关参数，tag订阅等。。
    .subscriptionProperties(subProperties)
    // 配置从最早开始消费，否则可能会消费不到历史消息
    .subscriptionInitialPosition(SubscriptionInitialPosition.Earliest).subscribe();


 // 创建 producer
 if msgId, err := producer.Send(ctx, &pulsar.ProducerMessage{
        Payload: []byte(fmt.Sprintf("hello-%d", i)),
        Properties: map[string]string{
            "tag1": "TAGS",
            "tag2": "TAGS",
            "xxx": "yyy",
        },
    }); err != nil {
        log.Fatal(err)
    }

// 创建 consumer
    consumer, err := client.Subscribe(pulsar.ConsumerOptions{
        Topic:            "topic-1",
        SubscriptionName: "my-sub",
        SubscriptionProperties: map[string]string{
            "tag1": "1",
            "tag2": "1",
        },
    })
```

> 注意
>
> 在 consumer 侧设置 SubscriptionProperties 字段时，一旦设定，这个订阅所处理的 tag 信息是不可变更的。

> 如果需要更换订阅的 tag，可以将当前的订阅先取消订阅

消息重试与死信机制

重试 Topic 是一种为了确保消息被正常消费而设计的 Topic 。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 Topic 中，当重试达到一定次数后，停止重试，投递到死信 Topic 中。
当消息进入到死信队列中，表示 TDMQ Pulsar 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。

自动重试

相关概念

重试 Topic：一个重试 Topic 对应一个订阅名（一个订阅者组的唯一标识），以 Topic 形式存在于 TDMQ Pulsar 版中。当您在控制台新建订阅，并打开自动创建重试&死信队列，系统会自动创建重试 Topic，该 Topic 会自主实现消息重试的机制。
该 Topic 命名为：

- 2.7.2 版本集群：[订阅名]-RETRY
- 2.6.1 版本集群：[订阅名]-retry

实现原理

您创建的消费者使用某个订阅名以共享模式订阅了一个 Topic 后，如果开启了 enableRetry 属性，就会自动订阅这个订阅名对应的重试队列。

> 说明
>
> 1. 仅共享模式（包括 Key 共享）支持自动化重试和死信机制，独占和灾备模式不支持。
> 2. 注意客户端版本需要与集群版本保持一致，客户端才能准确识别自动创建出的重试、死信队列。

这里以 Java 语言客户端为例，在 topic1 创建了一个 sub1 的订阅，客户端使用 sub1 订阅名订阅了 topic1 并开启了 enableRetry ，如下所示：

```
Consumer consumer = client.newConsumer()
    .topic("persistent://1******30/my-ns/topic1")
    .subscriptionType(SubscriptionType.Shared)//仅共享消费模式支持重试和死信
    .enableRetry(true)
    .subscriptionName("sub1")
    .subscribe();
```

此时， topic1 对 sub1 的订阅就形成了带有重试机制的投递模式， sub1 会自动订阅之前在新建订阅时自动创建的重试 Topic 中（可以在控制台 Topic 列表中找到）。当 topic1 中的消息投递第一次未收到消费端 ACK 时，这条消息就会被自动投递到重试 Topic ，并且由于 consumer 自动订阅了这个主题，后续这条消息会在一定的 重试规则 下重新被消费。当达到最大重试次数后仍失败，消息会被投递到对应的死信队列，等待人工处理。

> 说明
>
> 如果是 client 端自动创建的订阅，可以通过控制台上的 Topic 管理 > 更多 > 查看订阅进入消费管理页面手动重建重试和死信队列。



自定义参数设置

TDMQ Pulsar 版会默认配置一套重试和死信参数，具体如下：
2.7.2 版本集群

- 指定重试次数为16次（失败16次后，第17次会投递到死信队列）
- 指定重试队列为 [订阅名]-RETRY
- 指定死信队列为 [订阅名]-DLQ

  2.6.1 版本集群
- 指定重试次数为16次（失败16次后，第17次会投递到死信队列）
- 指定重试队列为 [订阅名]-retry
- 指定死信队列为 [订阅名]-dlq

  如果希望自定义配置这些参数，可以使用 deadLetterPolicy API 进行配置，代码如下：

```
Consumer<byte[]> consumer = pulsarClient.newConsumer()
    .topic("persistent://pulsar-****")
    .subscriptionName("sub1")
    .subscriptionType(SubscriptionType.Shared)
    .enableRetry(true)//开启重试消费
    .deadLetterPolicy(DeadLetterPolicy.builder()
        .maxRedeliverCount(maxRedeliveryCount)//可以指定最大重试次数
        .retryLetterTopic("persistent://my-property/my-ns/sub1-retry")//可以指定重试队列
        .deadLetterTopic("persistent://my-property/my-ns/sub1-dlq")//可以指定死信队列
        .build())
    .subscribe();
```

重试规则

重试规则由 reconsumerLater API 实现，有三种模式：

```
//指定任意延迟时间
```

```
consumer.reconsumeLater(msg, 1000L, TimeUnit.MILLISECONDS);
//指定延迟等级
consumer.reconsumeLater(msg, 1);
//等级递增
consumer.reconsumeLater(msg);
```

- 第一种：指定任意延迟时间。第二个参数填写延迟时间，第三个参数指定时间单位。延迟时间和延时消息的取值范围一致，范围在1 - 864000（单位：秒）。
- 第二种：指定任意延迟等级（仅限存量SDK的用户使用）。实现效果和第一种基本一致，更方便统一管理分布式系统中的延时时长，延迟等级说明如下：
  i. reconsumeLater(msg, 1) 中的第二个参数即为消息等级。
  ii. 默认 MESSAGE_DELAYLEVEL = "1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h" ，这个常数决定了每级对应的延时时间，例如1级对应1s，3级对应10s。如果默认值不符合实际业务需求，用户可以重新自定义。
- 第三种：等级递增（仅限存量 SDK 的用户使用）。实现的效果不同于以上两种，为退避式的重试，即第一次失败后重试间隔为1秒，第二次失败后重试间隔为5秒，以此类推，次数越多，间隔时间越长。具体时间间隔同样由第二种中介绍的 MESSAGE_DELAYLEVEL 决定。
  这种重试机制往往在业务场景中有更实际的应用，如果消费失败，一般服务不会立刻恢复，使用这种渐进式的重试方式更为合理。

> 注意
>
> 如果您使用的是 Pulsar 社区的 SDK，则不支持延迟等级和等级递增两种模式。

重试消息的消息属性

一条重试消息会给消息带上如下 property。

```
{
 REAL_TOPIC="persistent://my-property/my-ns/test,
 ORIGIN_MESSAGE_ID=314:28:-1,
 RETRY_TOPIC="persistent://my-property/my-ns/my-subscription-retry,
 RECONSUMETIMES=16
}
```

- REAL_TOPIC ：原 Topic。
- ORIGIN_MESSAGE_ID ：最初生产的消息 ID。
- RETRY_TOPIC ：重试 Topic。
- RECONSUMETIMES ：代表该消息重试的次数。

重试消息的消息 ID 流转

消息 ID 流转过程如下所示，您可以借助此规则对相关日志进行分析。

```
原始消费： msgid=1:1:0:1
第一次重试： msgid=2:1:-1
```

第二次重试： msgid=2:2:-1
第三次重试： msgid=2:3:-1
.....
第16次重试： msgid=2:16:0:1
第17次写入死信队列： msgid=3:1:-1

完整代码示例

- 重试（-RETRY）Topic 需要在 Consumer 中首先开启该功能（enableRetry(true)），默认为关闭状态。之后需要调用 reconsumeLater() 的接口消息才会被发送到重试 Topic 中。
- 死信（-DLQ）Topic 需要调用 consumer.reconsumeLater()，执行 reconsumeLater 之后原 topic 的那条消息会被 ack，消息转存到 retry topic，重试到达上限后消息转存至死信。Pulsar Client 会自动订阅 retry topic，但是进入死信就不会自动订阅，需要用户自己来订阅。
  以下为借助 TDMQ Pulsar 版实现完整消息重试机制的代码示例，供开发者参考。
  订阅主题

```
Consumer<byte[]> consumer1 = client.newConsumer()
        .topic("persistent://pulsar-****")
        .subscriptionName("my-subscription")
        .subscriptionType(SubscriptionType.Shared)
        .enableRetry(true)//开启重试消费
        //.deadLetterPolicy(DeadLetterPolicy.builder()
        //        .maxRedeliverCount(maxRedeliveryCount)
        //        .retryLetterTopic("persistent://my-property/my-ns/my-subscription-retry")//可以指定重试队列
        //        .deadLetterTopic("persistent://my-property/my-ns/my-subscription-dlq")//可以指定死信队列
        //        .build())
        .subscribe();
```

执行消费

```
while (true) {
    Message msg = consumer.receive();
    try {
        // Do something with the message
        System.out.printf("Message received: %s", new String(msg.getData()));
        // Acknowledge the message so that it can be deleted by the message broker
        consumer.acknowledge(msg);
    } catch (Exception e) {
        // select reconsume policy
        consumer.reconsumeLater(msg, 1000L, TimeUnit.MILLISECONDS);
        //consumer.reconsumeLater(msg, 1);
        //consumer.reconsumeLater(msg);
    }
}
```

主动重试

当消费者在某个时间没有成功消费某条消息，如果想重新消费到这条消息时，消费者可以发送一条取消确认消息到 TDMQ Pulsar 版服务端，TDMQ Pulsar 版会将这条消息重新发给消费者。 这种方式重试时不会产生新的消息，所以也不能自定义重试间隔。
以下为主动重试的 Java 代码示例：

```
while (true) {
    Message msg = consumer.receive();
    try {
        // Do something with the message
        System.out.printf("Message received: %s", new String(msg.getData()));
        // Acknowledge the message so that it can be deleted by the message broker
        consumer.acknowledge(msg);
    } catch (Exception e) {
        // Message failed to process, redeliver later
        consumer.negativeAcknowledge(msg);
    }
}
```

客户端连接与生产消费者

本文主要介绍 TDMQ Pulsar 客户端与连接、客户端与生产/消费者之间的关系，并向开发者介绍客户端合理的使用方式，以便更高效、稳定地使用 TDMQ Pulsar 版的服务。

> 说明：
>
> 核心原则：
>
> - 一个进程一个 PulsarClient 即可。
> - Producer、Consumer 是线程安全的，对于同一个 Topic，可以复用且最好复用。

客户端与连接

TDMQ Pulsar 客户端（以下简称 PulsarClient ）是应用程序连接到 TDMQ Pulsar 版的一个基本单位，一个 PulsarClient 对应一个 TCP 连接。一般来说，用户侧的一个应用程序或者进程对应使用一个 PulsarClient，有多少个应用节点，对应就有多少个 Client 数量。若长时间不使用 TDMQ 服务的应用节点，应回收 Client 以节省资源消耗（当前 TDMQ Pulsar 版的连接上限是单个 Topic 2000个 Client 连接）。

客户端与生产/消费者

一个 Client 下可以创建多个生产和消费者，用于提升生产和消费的速度。比较常见的用法是，一个 Client 下，利用多线程创建多个 Producer 或 Consumer 对象，用于生产消费，不同 Producer 和 Consumer 之间数据相互隔离。当前 TDMQ 对生产/消费者的限制为：

- 单个 Topic 生产者上限1000个。
- 单个 Topic 消费者上限500个。

最佳实践

生产/消费者的数量不一定取决于业务对象，它们是一个可以复用的资源，通过名称作为唯一标识进行区分。

生产者

假设有1000个业务对象在同时生产消息，并不是要创建1000个 Producer，只要是向同一个 Topic 进行投递，每个应用节点可以先统一使用一个 Producer 来进行生产（单例模式），往往单个 Producer 就能吃满单个应用节点的硬件配置。

以下给出一段 Java 消息生产的代码示例。

```
//从配置文件中获取 serviceURL 接入地址、Token 密钥、Topic 全名和 Subscription 名称（均可从控制台复制）
@Value("${tdmq.serviceUrl}")
private String serviceUrl;
@Value("${tdmq.token}")
private String token;
@Value("${tdmq.topic}")
private String topic;

//声明1个 Client 对象、producer 对象
private PulsarClient pulsarClient;
private Producer<String> producer;

//在一段初始化程序中创建好客户端和生产者对象
public void init() throws Exception {
pulsarClient = PulsarClient.builder()
.serviceUrl(serviceUrl)
.authentication(AuthenticationFactory.token(token))
.build();
producer = pulsarClient.newProducer(Schema.STRING)
.topic(topic)
.create();
}
```

在实际生产消息的业务逻辑中直接引用 producer 完成消息的发送。

```
//在实际生产消息的业务逻辑中直接引用，注意 Producer 通过范式声明的 Schema 类型要和传入对象匹配
public void onProduce(Producer<String> producer){
   //添加业务逻辑
   String msg = "my-message";//模拟从业务逻辑拿到消息
   try {
      //TDMQ Pulsar 版默认开启 Schema 校验, 消息对象一定需要和 producer 声明的 Schema 类型匹配
      MessageId messageId = producer.newMessage()
           .key("msgKey")
              .value(msg)
              .send();
      System.out.println("delivered msg " + msgId + ", value:" + value);
   } catch (PulsarClientException e) {
        System.out.println("delivered msg failed, value:" + value);
        e.printStackTrace();
   }
}

   public void onProduceAsync(Producer<String> producer){
```

```
    //添加业务逻辑
    String msg = "my-asnyc-message";//模拟从业务逻辑拿到消息
    //异步发送消息，无线程阻塞，提升发送速率
    CompletableFuture<MessageId> messageIdFuture = producer.newMessage()
         .key("msgKey")
            .value(msg)
            .sendAsync();
    //通过异步回调得知消息发送成功与否
    messageIdFuture.whenComplete(((messageId, throwable) -> {
       if( null != throwable ) {
          System.out.println("delivery failed, value: " + msg );
          //此处可以添加延时重试的逻辑
       } else {
          System.out.println("delivered msg " + messageId + ", value:" + msg);
       }
    }));
}
```

当一个生产者长时间不使用时需要调用 close 方法关闭，以避免占用资源；当一个客户端实例长时间不使用时，同样需要调用 close 方法关闭，以避免连接池被占满。

```
public void destroy(){
    if (producer != null) {
       producer.close();
    }
if (pulsarClient != null) {
    pulsarClient.close();
    }
}
```

消费者

如同生产者，消费者也最好按照单例模式进行使用，单个消费节点只需要一个客户端实例以及一个消费者实例。一般来说，一个消息队列的消费端的性能瓶颈都在于消费者按照自己业务逻辑处理消息的过程，而并非在接收消息的动作上。所以当出现了消费性能不足的时候，先看消费者的网络带宽消耗，如果趋势上看没有达到一个明显的上限，就应该先根据日志以及消息轨迹信息分析自身处理消息的业务逻辑耗时。

> 注意：
>
> - 当使用 Shared 或者 Key-Shared 模式时，消费者数量不一定小于等于分区数。服务端会有一个负责分发消息的模块按照一定的方式（Shared 模式默认是轮询，Key-Shared 则是在同一个 key 内轮询）将消息分发给所有的消费者。
> - 当使用 Shared模式，如果生产侧暂停了生产，则到了末尾一部分消息时，可能会出现消费分布不均的情况。
> - 使用多线程消费，即使复用一个 consumer 对象，消息的顺序也将无法得到保证。

以下给出一个 Java 基于 Spring boot 框架用线程池进行多线程消费的完整代码示例。

```
import org.apache.pulsar.client.api.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

@Service
public class ConsumerService implements Runnable {

    //从配置文件中获取 serviceURL 接入地址、Token 密钥、Topic 全名和 Subscription 名称（均可从控制台复制）
    @Value("${tdmq.serviceUrl}")
    private String serviceUrl;
    @Value("${tdmq.token}")
    private String token;
    @Value("${tdmq.topic}")
    private String topic;
    @Value("${tdmq.subscription}")
    private String subscription;

    private volatile boolean start = false;
    private PulsarClient pulsarClient;
    private Consumer<String> consumer;
    private static final int corePoolSize = 10;
    private static final int maximumPoolSize = 10;

    private ExecutorService executor;
    private static final Logger logger = LoggerFactory.getLogger(ConsumerService.class);

    @PostConstruct
    public void init() throws Exception {
        pulsarClient = PulsarClient.builder()
                .serviceUrl(serviceUrl)
                .authentication(AuthenticationFactory.token(token))
                .build();
        consumer = pulsarClient.newConsumer(Schema.STRING)
```

```
            .topic(topic)
            //.subscriptionType(SubscriptionType.Shared)
            .subscriptionName(subscription)
            .subscribe();
        executor = new ThreadPoolExecutor(corePoolSize, maximumPoolSize, 0, TimeUnit.SECONDS, new
ArrayBlockingQueue<>(100),
            new ThreadPoolExecutor.AbortPolicy());
        start = true;
    }

    @PreDestroy
    public void destroy() throws Exception {
        start = false;
        if (consumer != null) {
            consumer.close();
        }
        if (pulsarClient != null) {
            pulsarClient.close();
        }
        if (executor != null) {
            executor.shutdownNow();
        }
    }

    @Override
    public void run() {
        logger.info("tdmq consumer started...");
        for (int i = 0; i < maximumPoolSize; i++) {
            executor.submit(() -> {
                while (start) {
                    try {
                        Message<String> message = consumer.receive();
                        if (message == null) {
                            continue;
                        }
                        onConsumer(message);
                    } catch (Exception e) {
                        logger.warn("tdmq consumer business error", e);
                    }
                }
            });
        }
        logger.info("tdmq consumer stopped...");
    }

    /**
     * 这里写消费业务逻辑
```

```
     *
     * @param message
     * @return return true: 消息ack  return false: 消息nack
     * @throws Exception 消息nack
     */
    private void onConsumer(Message<String> message) {
        //业务逻辑,延时类操作
        try {
            System.out.println(Thread.currentThread().getName() + " - message receive: " + message.getVa
lue());
            Thread.sleep(1000);//模拟业务逻辑处理
            consumer.acknowledge(message);
            logger.info(Thread.currentThread().getName() + " - message processing succeed : " + messag
e.getValue());
        } catch (Exception exception) {
            consumer.negativeAcknowledge(message);
            logger.error(Thread.currentThread().getName() + " - message processing failed : " + message.g
etValue());
        }
    }
}
```

API文档
分布式消息队列（tdmq）
版本（2020-02-17）
API 概览

API版本

V3

CMQ管理相关接口

接口名称	接口功能
RewindCmqQueue	回溯cmq队列

主题相关接口

接口名称	接口功能
CreateTopic	新增主题
DeleteTopics	删除主题
DescribeTopics	查询主题列表
ModifyTopic	修改主题

其他接口

接口名称	接口功能
DeleteRoles	删除角色
DescribeBindVpcs	获取租户VPC绑定关系

接口名称	接口功能
SendMsg	发送消息

命名空间相关接口

接口名称	接口功能
CreateEnvironment	创建命名空间
DeleteEnvironments	删除命名空间
DescribeEnvironmentAttributes	获取命名空间属性
DescribeEnvironmentRoles	获取命名空间角色列表
DescribeEnvironments	获取命名空间列表
ModifyEnvironmentAttributes	修改命名空间属性

消息相关接口

接口名称	接口功能
AcknowledgeMessage	确认消息
ReceiveMessage	接收消息
SendBatchMessages	批量发送消息
SendMessages	发送单条消息

环境角色授权相关接口

接口名称	接口功能
CreateEnvironmentRole	创建环境角色授权
CreateRole	创建角色
DeleteEnvironmentRoles	删除环境角色授权

接口名称	接口功能
DescribeRoles	获取角色列表
ModifyEnvironmentRole	修改环境角色授权
ModifyRole	角色修改
TpoModifyResourceProject	修改资源对应的项目（包含新增）

生产消费相关接口

接口名称	接口功能
CreateSubscription	创建订阅关系
DeleteSubscriptions	删除订阅关系
DescribeSubscriptions	消费订阅列表
ResetMsgSubOffsetByTimestamp	消息回溯

集群相关接口

接口名称	接口功能
CreateCluster	创建集群
DeleteCluster	删除集群
DescribeBindClusters	获取专享集群列表
DescribeClusterDetail	获取集群详情
DescribeClusters	获取集群列表
ModifyCluster	更新集群信息

调用方式
接口签名v1

TCloudFinanceZone API 会对每个访问请求进行身份验证，即每个请求都需要在公共请求参数中包含签名信息（ Signature ）以验证请求者身份。
签名信息由安全凭证生成，安全凭证包括 SecretId 和 SecretKey；若用户还没有安全凭证，请前往云API密钥页面申请，否则无法调用云API接口。

1. 申请安全凭证

在第一次使用云API之前，请前往云API密钥页面申请安全凭证。
安全凭证包括 SecretId 和 SecretKey：

- SecretId 用于标识 API 调用者身份
- SecretKey 用于加密签名字符串和服务器端验证签名字符串的密钥。
- 用户必须严格保管安全凭证，避免泄露。

申请安全凭证的具体步骤如下：

1. 登录TCloudFinanceZone管理中心控制台。
2. 前往云API密钥的控制台页面
3. 在云API密钥页面，点击【新建】即可以创建一对SecretId/SecretKey

注意：开发商帐号最多可以拥有两对 SecretId / SecretKey。

2. 生成签名串

有了安全凭证SecretId 和 SecretKey后，就可以生成签名串了。以下是生成签名串的详细过程：

假设用户的 SecretId 和 SecretKey 分别是：

- SecretId: AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
- SecretKey: Gu5t9xGARNpq86cd98joQYCN3EXAMPLE

注意：这里只是示例，请根据用户实际申请的 SecretId 和 SecretKey 进行后续操作！

以云服务器查看实例列表(DescribeInstances)请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥Id	AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	shjr
InstanceIds.0	待查询的实例ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

2.1. 对参数排序

首先对所有请求参数按参数名的字典序（ ASCII 码）升序排序。注意：1）只按参数名进行排序，参数值保持对应即可，不参与比大小；2）按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 php 中的 ksort 函数。上述示例参数的排序结果如下:

```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'shjr',
    'SecretId' : 'AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE',
    'Timestamp' : 1465185768,
    'Version': '2017-03-12',
}
```

使用其它程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2.2. 拼接请求字符串

此步骤生成请求字符串。
将把上一步排序好的请求参数格式化成“参数名称”=“参数值”的形式，如对 Action 参数，其参数名称为 "Action" ，参数值为 "DescribeInstances" ，因此格式化后就为 Action=DescribeInstances 。
注意：“参数值”为原始值而非url编码后的值。

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.3. 拼接签名原文字符串

此步骤生成签名原文字符串。
签名原文字符串由以下几个参数构成:

1. 请求方法: 支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机:查看实例列表(DescribeInstances)的请求域名为：cvm.finance.cloud.tencent.com。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径: 当前版本云API的请求路径固定为 / 。
4. 请求字符串: 即上一步生成的请求字符串。

签名原文串的拼接规则为: 请求方法 + 请求主机 +请求路径 + ? + 请求字符串

示例的拼接结果为：

```
GETcvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.4. 生成签名串

此步骤生成签名串。
首先使用 HMAC-SHA1 算法对上一步中获得的签名原文字符串进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。

具体代码如下，以 PHP 语言为例:

```
$secretKey = 'Gu5t9xGARNpq86cd98joQYCN3EXAMPLE';
$srcStr = 'GETcvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12';
$signStr = base64_encode(hash_hmac('sha1', $srcStr, $secretKey, true));
echo $signStr;
```

最终得到的签名串为:

```
EliP9YW3pW28FpsEdkXt/+WcGeI=
```

使用其它程序设计语言开发时，可用上面示例中的原文进行签名验证，得到的签名串与例子中的一致即可。

3. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。

如上一步生成的签名串为 EliP9YW3pW28FpsEdkXt/+WcGeI= ，最终得到的签名串请求参数（Signature）为：EliP9YW3pW28FpsEdkXt%2f%2bWcGeI%3d，它将用于生成最终的请求 URL。

注意：如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。

注意：有些编程语言的 http 库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。

注意：其他参数值也需要进行编码，编码采用 RFC 3986。使用 %XY 对特殊字符例如汉字进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

4. 签名失败

根据实际情况，存在以下签名失败的错误码，请根据实际情况处理

错误代码	错误描述
AuthFailure.SignatureExpire	签名过期
AuthFailure.SecretIdNotFound	密钥不存在
AuthFailure.SignatureFailure	签名错误
AuthFailure.TokenFailure	token 错误
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）

5. 签名演示

在实际调用 API 3.0 时，推荐使用配套的TCloudFinanceZone SDK 3.0 ，SDK 封装了签名的过程，开发时只关注产品提供的具体接口即可。详细信息参见 SDK 中心。当前支持的编程语言有：

- Python
- Java

- PHP
- Go
- Node

为了更清楚的解释签名过程，下面以实际编程语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

最终输出的 url 可能为： https://cvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Signature=EliP9YW3pW28FpsEdkXt%2F%2BWcGeI%3D&Timestamp=1465185768&Version=2017-03-12

注意：由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp 。

注意：在下面的示例中，不同编程语言，甚至同一语言每次执行得到的 url 可能都有所不同，表现为参数的顺序不同，但这并不影响正确性。只要所有参数都在，且签名计算正确即可。

注意：以下代码仅适用于 API 3.0，不能直接用于其他的签名流程，即使是旧版的 API ，由于存在细节差异也会导致签名计算错误，请以对应的实际文档为准。

Java

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Random;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class CloudAPIDemo {
    private final static String CHARSET = "UTF-8";

    public static String sign(String s, String key, String method) throws Exception {
        Mac mac = Mac.getInstance(method);
        SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(CHARSET), mac.getAlgorithm());
        mac.init(secretKeySpec);
        byte[] hash = mac.doFinal(s.getBytes(CHARSET));
        return DatatypeConverter.printBase64Binary(hash);
    }

    public static String getStringToSign(TreeMap<String, Object> params) {
        StringBuilder s2s = new StringBuilder("GETcvm.finance.cloud.tencent.com/?");
```

```
        // 签名时要求对参数进行字典排序，此处用TreeMap保证顺序
        for (String k : params.keySet()) {
            s2s.append(k).append("=").append(params.get(k).toString()).append("&");
        }
        return s2s.toString().substring(0, s2s.length() - 1);
    }

    public static String getUrl(TreeMap<String, Object> params) throws UnsupportedEncodingException
{
        StringBuilder url = new StringBuilder("https://cvm.finance.cloud.tencent.com/?");
        // 实际请求的url中对参数顺序没有要求
        for (String k : params.keySet()) {
            // 需要对请求串进行urlencode，由于key都是英文字母，故此处仅对其value进行urlencode
            url.append(k).append("=").append(URLEncoder.encode(params.get(k).toString(), CHARSET)).app
end("&");
        }
        return url.toString().substring(0, url.length() - 1);
    }

    public static void main(String[] args) throws Exception {
        TreeMap<String, Object> params = new TreeMap<String, Object>(); // TreeMap可以自动排序
        // 实际调用时应当使用随机数，例如：params.put("Nonce", new Random().nextInt(java.lang.Intege
r.MAX_VALUE));
        params.put("Nonce", 11886); // 公共参数
        // 实际调用时应当使用系统当前时间，例如：  params.put("Timestamp", System.currentTimeMillis() /
1000);
        params.put("Timestamp", 1465185768); // 公共参数
        params.put("SecretId", "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"); // 公共参数
        params.put("Action", "DescribeInstances"); // 公共参数
        params.put("Version", "2017-03-12"); // 公共参数
        params.put("Region", "shjr"); // 公共参数
        params.put("Limit", 20); // 业务参数
        params.put("Offset", 0); // 业务参数
        params.put("InstanceIds.0", "ins-09dx96dg"); // 业务参数
        params.put("Signature", sign(getStringToSign(params), "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE
", "HmacSHA1")); // 公共参数
        System.out.println(getUrl(params));
    }
}
```

Python

注意：如果是在 Python 2 环境中运行，需要先安装 requests 依赖包： pip install requests 。

```
# -*- coding: utf8 -*-
import base64
```

```
import hashlib
import hmac
import time

import requests

secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

def get_string_to_sign(method, endpoint, params):
    s = method + endpoint + "/?"
    query_str = "&".join("%s=%s" % (k, params[k]) for k in sorted(params))
    return s + query_str

def sign_str(key, s, method):
    hmac_str = hmac.new(key.encode("utf8"), s.encode("utf8"), method).digest()
    return base64.b64encode(hmac_str)

if __name__ == '__main__':
    endpoint = "cvm.finance.cloud.tencent.com"
    data = {
        'Action' : 'DescribeInstances',
        'InstanceIds.0' : 'ins-09dx96dg',
        'Limit' : 20,
        'Nonce' : 11886,
        'Offset' : 0,
        'Region' : 'shjr',
        'SecretId' : secret_id,
        'Timestamp' : 1465185768, # int(time.time())
        'Version': '2017-03-12'
    }
    s = get_string_to_sign("GET", endpoint, data)
    data["Signature"] = sign_str(secret_key, s, hashlib.sha1)
    print(data["Signature"])
    # 此处会实际调用，成功后可能产生计费
    # resp = requests.get("https://" + endpoint, params=data)
    # print(resp.url)
```

接口签名v3

TCloudFinanceZone API 会对每个访问请求进行身份验证，即每个请求都需要在公共请求参数中包含签名信息（Signature）以验证请求者身份。
签名信息由安全凭证生成，安全凭证包括 SecretId 和 SecretKey；若用户还没有安全凭证，请前往云API密钥页面申请，否则无法调用云API接口。

1. 申请安全凭证

在第一次使用云API之前，请前往云API密钥页面申请安全凭证。
安全凭证包括 SecretId 和 SecretKey：

- SecretId 用于标识 API 调用者身份
- SecretKey 用于加密签名字符串和服务器端验证签名字符串的密钥。
- 用户必须严格保管安全凭证，避免泄露。

申请安全凭证的具体步骤如下：

1. 登录TCloudFinanceZone管理中心控制台。
2. 前往云API密钥的控制台页面
3. 在云API密钥页面，点击【新建】即可以创建一对SecretId/SecretKey

注意：开发商帐号最多可以拥有两对 SecretId / SecretKey。

2. TC3-HMAC-SHA256 签名方法

注意：对于GET方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于POST方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式默认所有业务接口均支持，multipart 格式只有特定业务接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。

下面以云服务器查询广州区实例列表作为例子，分步骤介绍签名的计算过程。我们仅用到了查询实例列表的两个参数：Limit 和 Offset，使用 GET 方法调用。

假设用户的 SecretId 和 SecretKey 分别是：AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE 和 Gu5t9xGARNpq86cd98joQYCN3EXAMPLE

2.1. 拼接规范请求串

按如下格式拼接规范请求串（CanonicalRequest）：

```
CanonicalRequest =
    HTTPRequestMethod + '\n' +
    CanonicalURI + '\n' +
    CanonicalQueryString + '\n' +
    CanonicalHeaders + '\n' +
    SignedHeaders + '\n' +
    HashedRequestPayload
```

- HTTPRequestMethod：HTTP 请求方法（GET、POST ），本示例中为 GET；
- CanonicalURI：URI 参数，API 3.0 固定为正斜杠（/）；
- CanonicalQueryString：发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串，对于 GET 请求，则为 URL 中问号（?）后面的字符串内容，本示例取值为：Limit=10&Offset=0。注意：CanonicalQueryString 需要经过 URL 编码。
- CanonicalHeaders：参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部参与签名以提高自身请求的唯一性和安全性。拼接规则：1）头部 key 和 value 统一转成小写，并去掉首尾空格，按照 key:value\n 格式拼接；2）多个头部，按照头部 key（小写）的字典排序进行拼接。此例中为： content-type:application/x-www-form-urlencoded\nhost:cvm.finance.cloud.tencent.com\n
- SignedHeaders：参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包含的头部内容是一一对应的。content-type 和 host 为必选头部。拼接规则：1）头部 key 统一转成小写；2）多个头部 key（小写）按照字典排序进行拼接，并且以分号（;）分隔。此例中为： content-type;host
- HashedRequestPayload：请求正文的哈希值，计算方法为 Lowercase(HexEncode(Hash.SHA256(RequestPayload)))，对 HTTP 请求整个正文 payload 做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。注意：对于 GET 请求，RequestPayload 固定为空字符串，对于 POST 请求，RequestPayload 即为 HTTP 请求正文 payload。

根据以上规则，示例中得到的规范请求串如下（为了展示清晰，\n 换行符通过另起打印新的一行替代）：

```
GET
/
Limit=10&Offset=0
content-type:application/x-www-form-urlencoded
host:cvm.finance.cloud.tencent.com

content-type;host
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

2.2. 拼接待签名字符串

按如下格式拼接待签名字符串：

```
StringToSign =
    Algorithm + \n +
```

```
RequestTimestamp + \n +
CredentialScope + \n +
HashedCanonicalRequest
```

- Algorithm：签名算法，目前固定为 TC3-HMAC-SHA256；
- RequestTimestamp：请求时间戳，即请求头部的 X-TC-Timestamp 取值，如上示例请求为 1539084154；
- CredentialScope：凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致；service 为产品名，必须与调用的产品域名一致，例如 cvm。如上示例请求，取值为 2018-10-09/cvm/tc3_request；
- HashedCanonicalRequest：前述步骤拼接所得规范请求串的哈希值，计算方法为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。

注意：

> 1. Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。
> 2. Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败（返回签名过期错误）。

根据以上规则，示例中得到的待签名字符串如下（为了展示清晰，\n 换行符通过另起打印新的一行替代）：

```
TC3-HMAC-SHA256
1539084154
2018-10-09/cvm/tc3_request
91c9c192c14460df6c1ffc69e34e6c5e90708de2a6d282cccf957dbf1aa7f3a7
```

2.3. 计算签名

1）计算派生签名密钥，伪代码如下

```
SecretKey = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"
SecretDate = HMAC_SHA256("TC3" + SecretKey, Date)
SecretService = HMAC_SHA256(SecretDate, Service)
SecretSigning = HMAC_SHA256(SecretService, "tc3_request")
```

- SecretKey：原始的 SecretKey；
- Date：即 Credential 中的 Date 字段信息，如上示例，为2018-10-09；
- Service：即 Credential 中的 Service 字段信息，如上示例，为 cvm；

2）计算签名，伪代码如下

```
Signature = HexEncode(HMAC_SHA256(SecretSigning, StringToSign))
```

- SecretSigning：即以上计算得到的派生签名密钥；
- StringToSign：即步骤2计算得到的待签名字符串；

2.4. 拼接 Authorization

按如下格式拼接 Authorization：

```
Authorization =
    Algorithm + ' ' +
    'Credential=' + SecretId + '/' + CredentialScope + ', ' +
    'SignedHeaders=' + SignedHeaders + ', '
    'Signature=' + Signature
```

- Algorithm：签名方法，固定为 TC3-HMAC-SHA256；
- SecretId：密钥对中的 SecretId；
- CredentialScope：见上文，凭证范围；
- SignedHeaders：见上文，参与签名的头部信息；
- Signature：签名值

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=5da7a33f6993f0614b047e5df4582db9e9bf4672ba50567dba16c6ccf174c474
```

最终完整的调用信息如下：

```
https://cvm.finance.cloud.tencent.com/?Limit=10&Offset=0

Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE/2018-10-09/cvm/tc3_request, SignedHeaders=content-type;host, Signature=5da7a33f6993f0614b047e5df4582db9e9bf4672ba50567dba16c6ccf174c474
Content-Type: application/x-www-form-urlencoded
Host: cvm.finance.cloud.tencent.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1539084154
X-TC-Region: shjr
```

3. 签名失败

根据实际情况，存在以下签名失败的错误码，请根据实际情况处理

错误代码	错误描述
AuthFailure.SignatureExpire	签名过期
AuthFailure.SecretIdNotFound	密钥不存在
AuthFailure.SignatureFailure	签名错误
AuthFailure.TokenFailure	token 错误
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）

4. 签名演示

Java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;
import java.util.TimeZone;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.HttpsURLConnection;
import javax.xml.bind.DatatypeConverter;

import org.apache.commons.codec.digest.DigestUtils;

public class CloudAPITC3Demo {
    private final static String CHARSET = "UTF-8";
    private final static String ENDPOINT = "cvm.finance.cloud.tencent.com";
    private final static String PATH = "/";
    private final static String SECRET_ID = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE";
    private final static String SECRET_KEY = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE";
    private final static String CT_X_WWW_FORM_URLENCODED = "application/x-www-form-urlencode
d";
    private final static String CT_JSON = "application/json";
```

```
    private final static String CT_FORM_DATA = "multipart/form-data";

    public static byte[] sign256(byte[] key, String msg) throws Exception {
        Mac mac = Mac.getInstance("HmacSHA256");
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, mac.getAlgorithm());
        mac.init(secretKeySpec);
        return mac.doFinal(msg.getBytes(CHARSET));
    }

    public static void main(String[] args) throws Exception {
        String service = "cvm";
        String host = "cvm.finance.cloud.tencent.com";
        String region = "shjr";
        String action = "DescribeInstances";
        String version = "2017-03-12";
        String algorithm = "TC3-HMAC-SHA256";
        String timestamp = "1539084154";
        //String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        // 注意时区，否则容易出错
        sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
        String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

        // ************* 步骤 1：拼接规范请求串 *************
        String httpRequestMethod = "GET";
        String canonicalUri = "/";
        String canonicalQueryString = "Limit=10&Offset=0";
        String canonicalHeaders = "content-type:application/x-www-form-urlencoded\n" + "host:" + host
+ "\n";
        String signedHeaders = "content-type;host";
        String hashedRequestPayload = DigestUtils.sha256Hex("");
        String canonicalRequest = httpRequestMethod + "\n" + canonicalUri + "\n" + canonicalQueryStri
ng + "\n"
                + canonicalHeaders + "\n" + signedHeaders + "\n" + hashedRequestPayload;
        System.out.println(canonicalRequest);

        // ************* 步骤 2：拼接待签名字符串 *************
        String credentialScope = date + "/" + service + "/" + "tc3_request";
        String hashedCanonicalRequest = DigestUtils.sha256Hex(canonicalRequest.getBytes(CHARSET));
        String stringToSign = algorithm + "\n" + timestamp + "\n" + credentialScope + "\n" + hashedCan
onicalRequest;
        System.out.println(stringToSign);

        // ************* 步骤 3：计算签名 *************
        byte[] secretDate = sign256(("TC3" + SECRET_KEY).getBytes(CHARSET), date);
        byte[] secretService = sign256(secretDate, service);
        byte[] secretSigning = sign256(secretService, "tc3_request");
```

```
        String signature = DatatypeConverter.printHexBinary(sign256(secretSigning, stringToSign)).toLow
erCase();
        System.out.println(signature);

        // ************* 步骤 4：拼接 Authorization *************
        String authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" + credentialScope + ", "
                + "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signature;
        System.out.println(authorization);

        TreeMap<String, String> headers = new TreeMap<String, String>();
        headers.put("Authorization", authorization);
        headers.put("Host", host);
        headers.put("Content-Type", CT_X_WWW_FORM_URLENCODED);
        headers.put("X-TC-Action", action);
        headers.put("X-TC-Timestamp", timestamp);
        headers.put("X-TC-Version", version);
        headers.put("X-TC-Region", region);
    }
}
```

Python

```
# -*- coding: utf-8 -*-
import hashlib, hmac, json, os, sys, time
from datetime import datetime

# 密钥参数
secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

service = "cvm"
host = "cvm.finance.cloud.tencent.com"
endpoint = "https://" + host
region = "shjr"
action = "DescribeInstances"
version = "2017-03-12"
algorithm = "TC3-HMAC-SHA256"
timestamp = 1539084154
date = datetime.utcfromtimestamp(timestamp).strftime("%Y-%m-%d")
params = {"Limit": 10, "Offset": 0}

# ************* 步骤 1：拼接规范请求串 *************
http_request_method = "GET"
canonical_uri = "/"
canonical_querystring = "Limit=10&Offset=0"
ct = "x-www-form-urlencoded"
```

```
payload = ""
if http_request_method == "POST":
    canonical_querystring = ""
    ct = "json"
    payload = json.dumps(params)
canonical_headers = "content-type:application/%s\nhost:%s\n" % (ct, host)
signed_headers = "content-type;host"
hashed_request_payload = hashlib.sha256(payload.encode("utf-8")).hexdigest()
canonical_request = (http_request_method + "\n" +
                     canonical_uri + "\n" +
                     canonical_querystring + "\n" +
                     canonical_headers + "\n" +
                     signed_headers + "\n" +
                     hashed_request_payload)
print(canonical_request)

# ************* 步骤 2：拼接待签名字符串 *************
credential_scope = date + "/" + service + "/" + "tc3_request"
hashed_canonical_request = hashlib.sha256(canonical_request.encode("utf-8")).hexdigest()
string_to_sign = (algorithm + "\n" +
                  str(timestamp) + "\n" +
                  credential_scope + "\n" +
                  hashed_canonical_request)
print(string_to_sign)

# ************* 步骤 3：计算签名 *************
# 计算签名摘要函数
def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
secret_date = sign(("TC3" + secret_key).encode("utf-8"), date)
secret_service = sign(secret_date, service)
secret_signing = sign(secret_service, "tc3_request")
signature = hmac.new(secret_signing, string_to_sign.encode("utf-8"), hashlib.sha256).hexdigest()
print(signature)

# ************* 步骤 4：拼接 Authorization *************
authorization = (algorithm + " " +
                 "Credential=" + secret_id + "/" + credential_scope + ", " +
                 "SignedHeaders=" + signed_headers + ", " +
                 "Signature=" + signature)
print(authorization)

# 公共参数添加到请求头部
headers = {
    "Authorization": authorization,
    "Host": host,
    "Content-Type": "application/%s" % ct,
```

```
    "X-TC-Action": action,
    "X-TC-Timestamp": str(timestamp),
    "X-TC-Version": version,
    "X-TC-Region": region,
}
```

请求结构

1. 服务地址

地域（Region）是指物理的数据中心的地理区域。TCloudFinanceZone交付验证不同地域之间完全隔离，保证不同地域间最大程度的稳定性和容错性。为了降低访问时延、提高下载速度，建议您选择最靠近您客户的地域。

您可以通过 API接口 查询地域列表 查看完成的地域列表。

2. 通信协议

TCloudFinanceZone API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法:

- POST（推荐）
- GET

POST 请求支持的 Content-Type 类型：

- application/json（推荐），必须使用 TC3-HMAC-SHA256 签名方法。
- application/x-www-form-urlencoded，必须使用 HmacSHA1 或 HmacSHA256 签名方法。
- multipart/form-data（仅部分接口支持），必须使用 TC3-HMAC-SHA256 签名方法。

GET 请求的请求包大小不得超过 32 KB。POST 请求使用签名方法为 HmacSHA1、HmacSHA256 时不得超过 1 MB。POST 请求使用签名方法为 TC3-HMAC-SHA256 时支持 10 MB。

4. 字符编码

均使用UTF-8编码。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表 (DescribeInstancesStatus) 2017-03-12 版本为例，若调用成功，其可能的返回如下为：

```
{
    "Response": {
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

- Response 及其内部的 RequestId 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。
- 除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 TotalCount 和 InstanceStatusSet 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 TotalCount 在此情况下的返回值为 0， InstanceStatusSet 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：

```
{
    "Response": {
        "Error": {
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please check your signature is cor
rect."
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

- Error 的出现代表着该请求调用失败。Error 字段连同其内部的 Code 和 Message 字段在调用失败时是必定返回的。
- Code 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

- Message 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。

错误码	错误描述
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	http(s)请求协议错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

公共参数

公共参数是用于标识用户和接口鉴权目的的参数，如非必要，在每个接口单独的接口文档中不再对这些参数进行说明，但每次请求均需要携带这些参数，才能正常发起请求。

签名方法 v3

使用 TC3-HMAC-SHA256 签名方法时，公共参数需要统一放到 HTTP Header 请求头部中，如下：

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702。注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024 其中， - TC3-HMAC-SHA256：签名方法，目前固定取该值； - Credential：签名凭证，AKIDEXAMPLE 是 SecretId；Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致；service为产品名，必须与调用的产品域名一致，例如cvm； - SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部； - Signature：签名摘要。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 v1

使用 HmacSHA1 和 HmacSHA256 签名方法时，公共参数需要统一放到请求串中，如下

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。

参数名称	类型	必选	描述
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在云API密钥上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey ，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见接口鉴权文档。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

地域列表

地域（Region）是指物理的数据中心的地理区域。TCloudFinanceZone交付验证不同地域之间完全隔离，保证不同地域间最大程度的稳定性和容错性。为了降低访问时延、提高下载速度，建议您选择最靠近您客户的地域。

您可以通过 API接口 查询地域列表 查看完成的地域列表。

CMQ管理相关接口
回溯cmq队列

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

回溯cmq队列

默认接口请求频率限制：20次/秒。

接口更新时间：2021-09-15 16:59:19。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：RewindCmqQueue
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
QueueName	是	否	String	队列名字，在单个地域同一帐号下唯一。队列名称是一个不超过64个字符的字符串，必须以字母为首字符，剩余部分可以包含字母、数字和横划线(-)。 示例值： test-queue
StartConsumeTime	是	否	Uint64	设定该时间，则（Batch）receiveMessage接口，会按照生产消息的先后顺序消费该时间戳以后的消息。 示例值： 1582108595000

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

该接口暂无业务逻辑相关的错误码，其他错误码详见公共错误码。

主题相关接口
新增主题

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

新增指定分区、类型的消息主题

默认接口请求频率限制：200次/秒。

接口更新时间：2021-07-19 19:21:10。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateTopic
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： default
TopicName	是	否	String	主题名，不支持中字以及除了短线和下划线外的特殊字符且不超过64个字符。 示例值： test_topic
Partitions	是	否	Uint64	0：非分区topic，无分区；非0：具体分区topic的分区数，最大不允许超过128。 示例值： 2
Remark	否	否	String	备注，128字符以内。 示例值： 2个分区的普通消息队列
TopicType	是	否	Uint64	0： 普通消息； 1 ：全局顺序消息；

参数名称	必选	允许NULL	类型	描述
				2 ：局部顺序消息； 3 ：重试队列； 4 ：死信队列。 示例值： TopicType 和 PulsarTopicType 两者不能同时为空，有一个必选
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-5r59xd4vnx

3. 输出参数

参数名称	类型	描述
EnvironmentId	String	环境（命名空间）名称。 示例值： default
TopicName	String	主题名。 示例值： test_topic
Partitions	Uint64	0：非分区topic，无分区；非0：具体分区topic的分区数。 示例值： 2
Remark	String	备注，128字符以内。 示例值： 2个分区的普通消息队列
TopicType	Uint64	0： 普通消息； 1 ：全局顺序消息； 2 ：局部顺序消息； 3 ：重试队列； 4 ：死信队列； 5 ：事务消息。 示例值： 0
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。

错误码	描述
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
ResourceInUse.Topic	重名，主题已存在。
FailedOperation.CreateTopic	主题创建失败。
MissingParameter.NeedMoreParams	必要参数没有传递。
LimitExceeded.Topics	实例下主题数量超过限制。
InvalidParameterValue	参数取值错误。
FailedOperation	操作失败。
ResourceNotFound.BrokerCluster	服务的集群不存在。
ResourceNotFound.Cluster	集群不存在。

删除主题

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

批量删除topics

默认接口请求频率限制：20次/秒。

接口更新时间：2021-08-04 10:41:16。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteTopics
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
TopicSets	是	否	Array of TopicRecord	主题集合，每次最多删除20个。 示例值：查看
ClusterId	否	否	String	pulsar集群Id。 示例值： pulsar-xk3ne8k2qkp8
EnvironmentId	否	否	String	环境（命名空间）名称。 示例值： devNs
Force	否	否	Bool	是否强制删除，默认为false 示例值： false

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
TopicSets	Array of TopicRecord	被删除的主题数组。 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
ResourceNotFound.Topic	主题不存在。
FailedOperation.DeleteTopics	主题删除失败。
OperationDenied.DefaultEnvironment	默认环境不允许操作。
MissingParameter.NeedMoreParams	必要参数没有传递。
FailedOperation	操作失败。
ResourceNotFound.Cluster	集群不存在。

查询主题列表

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取环境下主题列表

默认接口请求频率限制：20次/秒。

接口更新时间：2023-04-18 15:22:05。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeTopics
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： devNs
TopicName	否	否	String	主题名模糊匹配。 示例值： topic1
Offset	否	否	Uint64	起始下标，不填默认为0。 示例值： 0
Limit	否	否	Uint64	返回数量，不填则默认为10，最大值为20。 示例值： 20
TopicType	否	否	Uint64	topic类型描述： 0：普通消息； 1：全局顺序消息； 2：局部顺序消息； 3：重试队列； 4：死信队列；

参数名称	必选	允许NULL	类型	描述
				5：事务消息。 示例值： 1
ClusterId	否	否	String	Pulsar 集群的ID 示例值： cluster1
Filters	否	否	Array of Filter	* TopicName 按照主题名字查询，精确查询。 类型：String 必选：否 示例值：查看
InternalReqSource	否	否	String	内部业务标识 示例值： rocketmq-xxxxx

3. 输出参数

参数名称	类型	描述
TopicSets	Array of Topic	主题集合数组。 示例值：查看
TotalCount	Uint64	主题数量。 示例值： 1
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
MissingParameter.NeedMoreParams	必要参数没有传递。

错误码	描述
ResourceNotFound.BrokerCluster	服务的集群不存在。
ResourceNotFound.Cluster	集群不存在。

修改主题

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

修改主题备注和分区数

默认接口请求频率限制：50次/秒。

接口更新时间：2021-06-16 18:02:21。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyTopic
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： default
TopicName	是	否	String	主题名。 示例值： test_topic
Partitions	是	否	Uint64	分区数，必须大于或者等于原分区数，若想维持原分区数请输入原数目，修改分区数仅对非全局顺序消息起效果，不允许超过128个分区。 示例值： 3
Remark	否	否	String	备注，128字符以内。 示例值： 修改分区为3
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-5r59xd4vnx

3. 输出参数

参数名称	类型	描述
Partitions	Uint64	分区数 示例值： 3
Remark	String	备注，128字符以内。 示例值： 修改分区为3
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Topic	主题不存在。
FailedOperation.UpdateTopic	主题更新失败。
MissingParameter.NeedMoreParams	必要参数没有传递。
FailedOperation.GetTopicPartitionsFailed	获取主题分区数失败。
FailedOperation	操作失败。

其他接口
删除角色

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

删除角色，支持批量。

默认接口请求频率限制：20次/秒。

接口更新时间：2021-10-08 16:07:08。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteRoles
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
RoleNames	是	否	Array of String	角色名称数组。 示例值： ["test_role_1","test_role_2"]
ClusterId	否	否	String	必填字段，集群Id 示例值： pulsar-xk3ne8k2qkp8

3. 输出参数

参数名称	类型	描述
RoleNames	Array of String	成功删除的角色名称数组。 示例值： ["test_role_1","test_role_2"]

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.Role	角色不存在。
FailedOperation.DeleteRoles	角色删除失败。
ResourceInUse.EnvironmentRole	环境角色已存在。
FailedOperation	操作失败。

获取租户VPC绑定关系

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取租户VPC绑定关系

默认接口请求频率限制：20次/秒。

接口更新时间：2021-01-13 15:38:57。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeBindVpcs
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
Offset	否	否	Uint64	起始下标，不填默认为0。 示例值： 1
Limit	否	否	Uint64	返回数量，不填则默认为10，最大值为20。 示例值： 1
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-xk3ne8k2qkp8

3. 输出参数

参数名称	类型	描述
TotalCount	Uint64	记录数。 示例值： 1

参数名称	类型	描述
VpcSets	Array of VpcBindRecord	Vpc集合。 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.NeedMoreParams	必要参数没有传递。
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.BrokerCluster	服务的集群不存在。

发送消息

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

此接口仅用于测试发生消息，不能作为现网正式生产使用

默认接口请求频率限制：20次/秒。

接口更新时间：2021-06-21 17:11:02。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：SendMsg
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： userNamespace
TopicName	是	否	String	主题名称，如果是分区topic需要指定具体分区，如果没有指定则默认发到0分区，例如：my_topic-partition-0。 示例值： cloud_test
MsgContent	是	否	String	消息内容，不能为空且大小不得大于5242880个byte。 示例值： one_msg
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-k3ex2kxanuol

3. 输出参数

参数名称	类型	描述
Result	Bool	结果。 示例值： true
MsgId	String	消息ID。 示例值： 50169152:0:0:-1
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Topic	主题不存在。
MissingParameter.NeedMoreParams	必要参数没有传递。
FailedOperation.GetTopicPartitionsFailed	获取主题分区数失败。
FailedOperation.SendMsgFailed	发送消息失败。
FailedOperation	操作失败。

命名空间相关接口
创建命名空间

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

用于在用户账户下创建消息队列 Tdmq 命名空间

默认接口请求频率限制：20次/秒。

接口更新时间：2021-07-27 16:20:00。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateEnvironment
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称，不支持中字以及除了短线和下划线外的特殊字符且不超过16个字符。 示例值： devNamespace
MsgTTL	是	否	Uint64	未消费消息过期时间，单位：秒，最小60，最大1296000，（15天）。 示例值： 60
Remark	否	否	String	说明，128个字符以内。 示例值： demo
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-ge3n43v3wj

参数名称	必选	允许NULL	类型	描述
RetentionPolicy	否	否	RetentionPolicy	消息保留策略 示例值：查看

3. 输出参数

参数名称	类型	描述
EnvironmentId	String	环境（命名空间）名称。 示例值： devNamespace
MsgTTL	Uint64	未消费消息过期时间，单位：秒。 示例值： 60
Remark	String	说明，128个字符以内。 示例值： demo
NamespaceId	String	命名空间ID 示例值： namespace-ak7jexmrz2
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
FailedOperation.CreateEnvironment	环境创建失败。
OperationDenied.DefaultEnvironment	默认环境不允许操作。
MissingParameter.NeedMoreParams	必要参数没有传递。
LimitExceeded.Environments	实例下环境数量超过限制。
FailedOperation	操作失败。

错误码	描述
ResourceNotFound.Cluster	集群不存在。
InvalidParameterValue.TTL	无效的消息TTL值。
LimitExceeded.Namespaces	实例下命名空间数量超过限额。
ResourceInUse.Namespace	重名，命名空间已存在。
FailedOperation.CreateNamespace	创建命名空间失败。

删除命名空间

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

批量删除租户下的命名空间

默认接口请求频率限制：20次/秒。

接口更新时间：2023-04-19 18:37:26。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteEnvironments
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentIds	是	否	Array of String	环境（命名空间）数组，每次最多删除20个。 示例值： [test1,test2]
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-ge3n43v3wj

3. 输出参数

参数名称	类型	描述
EnvironmentIds	Array of String	成功删除的环境（命名空间）数组。 示例值： [test1,test2]
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
FailedOperation.DeleteEnvironments	环境删除失败。
OperationDenied.DefaultEnvironment	默认环境不允许操作。
MissingParameter.NeedMoreParams	必要参数没有传递。
InvalidParameter	参数错误。
FailedOperation.RoleInUse	必须先清除关联角色数据才能继续操作。
FailedOperation.TopicInUse	必须先清除关联主题数据才能继续操作。
FailedOperation.DeleteNamespace	删除命名空间失败。
ResourceNotFound.Cluster	集群不存在。

获取命名空间属性

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取指定命名空间的属性

默认接口请求频率限制：20次/秒。

接口更新时间：2021-07-12 16:00:36。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeEnvironmentAttributes
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： default
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-ge3n43v3wj

3. 输出参数

参数名称	类型	描述
MsgTTL	Uint64	未消费消息过期时间，单位：秒，最大1296000（15天）。 示例值： 60

参数名称	类型	描述
RateInByte	Uint64	消费速率限制，单位：byte/秒，0：不限速。 示例值： 0
RateInSize	Uint64	消费速率限制，单位：个数/秒，0：不限速。 示例值： 0
RetentionHours	Uint64	已消费消息保存策略，单位：小时，0：消费完马上删除。 示例值： 0
RetentionSize	Uint64	已消费消息保存策略，单位：G，0：消费完马上删除。 示例值： 0
EnvironmentId	String	环境（命名空间）名称。 示例值： default
Replicas	Uint64	副本数。 示例值： 2
Remark	String	备注。 示例值： demo
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InternalError.GetAttributesFailed	获取属性失败。
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
FailedOperation.GetEnvironmentAttributesFailed	获取环境属性失败。
MissingParameter.NeedMoreParams	必要参数没有传递。
InternalError.BrokerService	Broker服务异常。

获取命名空间角色列表

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取命名空间角色列表

默认接口请求频率限制：20次/秒。

接口更新时间：2021-10-08 16:06:38。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeEnvironmentRoles
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	否	否	String	必填字段，环境（命名空间）名称。 示例值： default
Offset	否	否	Int64	起始下标，不填默认为0。 示例值： 0
Limit	否	否	Int64	返回数量，不填则默认为10，最大值为20。 示例值： 10
ClusterId	否	否	String	必填字段，Pulsar 集群的ID 示例值： pulsar-ge3n43v3wj
RoleName	否	否	String	角色名称 示例值： role
Filters	否	否	Array of Filter	* RoleName 按照角色名进行过滤，精确查询。 类型：String

参数名称	必选	允许NULL	类型	描述
				必选：否 示例值：查看

3. 输出参数

参数名称	类型	描述
TotalCount	Int64	记录数。 示例值： 1
EnvironmentRoleSets	Array of EnvironmentRole	命名空间角色集合。 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.Role	角色不存在。
ResourceNotFound.BrokerCluster	服务的集群不存在。
ResourceNotFound.Cluster	集群不存在。

获取命名空间列表

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取租户下命名空间列表

默认接口请求频率限制：20次/秒。

接口更新时间：2023-04-20 15:35:11。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeEnvironments
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	否	否	String	命名空间名称，模糊搜索。 示例值： default
Offset	否	否	Uint64	起始下标，不填默认为0。 示例值： 0
Limit	否	否	Uint64	返回数量，不填则默认为10，最大值为20。 示例值： 5
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-ge3n43v3wj
Filters	否	否	Array of Filter	* EnvironmentId 按照名称空间进行过滤，精确查询。 类型：String 必选：否 示例值：查看

3. 输出参数

参数名称	类型	描述
TotalCount	Uint64	命名空间记录数。 示例值： 1
EnvironmentSet	Array of Environment	命名空间集合数组。 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.Cluster	集群不存在。
FailedOperation	操作失败。

修改命名空间属性

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

修改指定命名空间的属性值

默认接口请求频率限制：20次/秒。

接口更新时间：2021-07-28 11:01:03。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyEnvironmentAttributes
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	命名空间名称。 示例值： default
MsgTTL	是	否	Uint64	未消费消息过期时间，单位：秒，最大1296000。 示例值： 60
Remark	否	否	String	备注，字符串最长不超过128。 示例值： demo
ClusterId	否	否	String	集群ID 示例值： pulsar-ge3n43v3wj
RetentionPolicy	否	否	RetentionPolicy	消息保留策略 示例值：查看

3. 输出参数

参数名称	类型	描述
EnvironmentId	String	命名空间名称。 示例值： default
MsgTTL	Uint64	未消费消息过期时间，单位：秒。 示例值： 60
Remark	String	备注，字符串最长不超过128。 示例值： demo
NamespaceId	String	命名空间ID 示例值： namespace-ak27qjs68x
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
FailedOperation.UpdateEnvironment	环境更新失败。
OperationDenied.DefaultEnvironment	默认环境不允许操作。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.Cluster	集群不存在。
InvalidParameterValue.TTL	无效的消息TTL值。
FailedOperation.SetTTL	设置消息TTL失败。
ResourceNotFound.Namespace	命名空间不存在。

消息相关接口
确认消息

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

根据提供的 MessageID 确认指定 topic 中的消息

默认接口请求频率限制：1000次/秒。

接口更新时间：2021-06-24 14:51:05。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：AcknowledgeMessage
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
MessageId	是	否	String	用作标识消息的唯一的ID（可从 receiveMessage 的返回值中获得） 示例值： 5:7:-1
AckTopic	是	否	String	Topic 名字（可从 receiveMessage 的返回值中获得）这里尽量需要使用topic的全路径，即：tenant/namespace/topic。如果不指定，默认使用的是：public/default 示例值： persistent://tenant/namespace/my-topic
SubName	是	否	String	订阅者的名字，可以从receiveMessage的返回值中获取到。这里尽量与receiveMessage中的订阅者保持一致，否则没办法正确ack 接收回来的消息。 示例值： my-sub

3. 输出参数

参数名称	类型	描述
ErrorMsg	String	如果为“”，则说明没有错误返回 示例值： "ack error"
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.TopicNotFound	上传的topic name错误。
InvalidParameter.TenantNotFound	上传的 tenant name 错误。

接收消息

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

接收发送到指定 topic 中的消息

默认接口请求频率限制：1000次/秒。

接口更新时间：2021-06-24 14:50:21。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ReceiveMessage
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
Topic	是	否	String	接收消息的topic的名字, 这里尽量需要使用topic的全路径，如果不指定，即：tenant/namespace/topic。默认使用的是：public/default 示例值： persistent://tenant/namespace/my-topic
SubscriptionName	是	否	String	订阅者的名字 示例值： my-sub
ReceiverQueueSize	否	否	Int64	默认值为1000，consumer接收的消息会首先存储到receiverQueueSize这个队列中，用作调优接收消息的速率 示例值： 1000
SubInitialPosition	否	否	String	默认值为：Latest。用作判定consumer初始接收消息的位置，可选参数为：Earliest, Latest 示例值： Latest

3. 输出参数

参数名称	类型	描述
MessageID	String	用作标识消息的唯一主键 示例值： {"Response":{"MessageId":"26365:1475:0"}}
MessagePayload	String	接收消息的内容 示例值： my-payload
AckTopic	String	提供给 Ack 接口，用来Ack哪一个topic中的消息 示例值： my-topic
ErrorMsg	String	返回的错误信息，如果为空，说明没有错误 示例值： sssss
SubName	String	返回订阅者的名字，用来创建 ack consumer时使用 示例值： my-sub
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.TopicNotFound	上传的topic name错误。
InvalidParameter.TenantNotFound	上传的 tenant name 错误。
InvalidParameter.TokenNotFound	没有获取到正确的 token。
FailedOperation.ReceiveTimeout	接收消息超时，请重试。
FailedOperation.ReceiveError	接收消息出错。

批量发送消息

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

批量发送消息

默认接口请求频率限制：1000次/秒。

接口更新时间：2021-06-24 14:46:50。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：SendBatchMessages
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
Topic	是	否	String	消息要发送的topic的名字, 这里尽量需要使用topic的全路径，即：tenant/namespace/topic。如果不指定，默认使用的是：public/default 示例值： persistent://tenant/namespace/my-topic
Payload	是	否	String	需要发送消息的内容 示例值： hello tdmq
StringToken	否	否	String	String 类型的 token，可以不填，系统会自动获取 示例值： abcdefghijklmnopqrstovwxyz
ProducerName	否	否	String	producer 的名字，要求全局是唯一的，如果不设置，系统会自动生成

参数名称	必选	允许NULL	类型	描述
				示例值： producer1
SendTimeout	否	否	Int64	单位：s。消息发送的超时时间。默认值为：30s 示例值： 30
MaxPendingMessages	否	否	Int64	内存中允许缓存的生产消息的最大数量，默认值：1000条 示例值： 1000
BatchingMaxMessages	否	否	Int64	每一个batch中消息的最大数量，默认值：1000条/batch 示例值： 1000
BatchingMaxPublishDelay	否	否	Int64	每一个batch最大等待的时间，超过这个时间，不管是否达到指定的batch中消息的数量和大小，都会将该batch发送出去，默认：10ms 示例值： 10L
BatchingMaxBytes	否	否	Int64	每一个batch中最大允许的消息的大小，默认：128KB 示例值： 128*1024

3. 输出参数

参数名称	类型	描述
MessageId	String	消息的唯一标识 示例值： {"Response":{"MessageId":"26365:1475:0"}}
ErrorMsg	String	错误消息，返回为 ""，代表没有错误 示例值： "error"
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述

错误码	描述
InvalidParameterValue.TopicNotFound	上传的topic name错误。
InvalidParameter.TenantNotFound	上传的 tenant name 错误。
InvalidParameter.TokenNotFound	没有获取到正确的 token。
FailedOperation.CreateProducerError	创建producer出错。
FailedOperation.CreatePulsarClientError	创建TDMQ client的出错。

发送单条消息

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

发送单条消息

默认接口请求频率限制：1000次/秒。

接口更新时间：2021-06-24 14:46:14。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：SendMessages
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
StringToken	否	否	String	Token 是用来做鉴权使用的，可以不填，系统会自动获取 示例值： abcdefghijklmnopqrstovwxyz
Topic	是	否	String	消息要发送的topic的名字, 这里尽量需要使用topic的全路径，即：tenant/namespace/topic。如果不指定，默认使用的是：public/default 示例值： persistent://tenant/namespaces/my-topic
Payload	是	否	String	要发送的消息的内容 示例值： hello tdmq
ProducerName	否	否	String	设置 producer 的名字，要求全局唯一，用户不配置，系统会随机生成 示例值： producer

参数名称	必选	允许NULL	类型	描述
SendTimeout	否	否	Int64	设置消息发送的超时时间，默认为30s 示例值： 30
MaxPendingMessages	否	否	Int64	内存中缓存的最大的生产消息的数量，默认为1000条 示例值： 1000

3. 输出参数

参数名称	类型	描述
MessageId	String	消息的messageID, 是全局唯一的，用来标识消息的元数据信息 示例值： {"Response":{"MessageId":"26365:1475:0"}}
ErrorMsg	String	返回的错误消息，如果返回为“”，说明没有错误 示例值： "error"
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.TopicNotFound	上传的topic name错误。
InvalidParameter.TenantNotFound	上传的 tenant name 错误。
InvalidParameter.TokenNotFound	没有获取到正确的 token。
FailedOperation.CreateProducerError	创建producer出错。
FailedOperation.CreatePulsarClientError	创建TDMQ client的出错。

环境角色授权相关接口
创建环境角色授权

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

创建环境角色授权

默认接口请求频率限制：20次/秒。

接口更新时间：2023-04-19 18:38:36。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateEnvironmentRole
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： devDemo
RoleName	是	否	String	角色名称。 示例值： test_role_1
Permissions	是	否	Array of String	授权项，最多只能包含produce、consume两项的非空字符串数组。 示例值： ["produce"]
ClusterId	否	否	String	必填字段，集群的ID 示例值： pulsar-xk3ne8k2qkp8

3. 输出参数

参数名称	类型	描述
EnvironmentId	String	环境（命名空间）名称。 示例值： devDemo
RoleName	String	角色名称。 示例值： test_role_1
Permissions	Array of String	授权项。 示例值： ["produce"]
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.Role	角色不存在。
ResourceInUse.EnvironmentRole	环境角色已存在。
FailedOperation.CreateEnvironmentRole	创建环境角色失败。
FailedOperation	操作失败。
ResourceNotFound.Cluster	集群不存在。
FailedOperation.UpdateEnvironmentRole	更新环境角色失败。

创建角色

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

创建角色

默认接口请求频率限制：20次/秒。

接口更新时间：2023-04-18 15:54:23。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateRole
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
RoleName	是	否	String	角色名称，不支持中字以及除了短线和下划线外的特殊字符且长度必须大于0且小等于32。 示例值： test_role_123
Remark	否	否	String	备注说明，长度必须大等于0且小等于128。 示例值： devDemo
ClusterId	否	否	String	必填字段，集群Id 示例值： pulsar-xk3ne8k2qkp8
EnvironmentRoleSets	否	否	Array of EnvironmentRoleSet	批量操作环境 示例值：查看

3. 输出参数

参数名称	类型	描述
RoleName	String	角色名称 示例值： test_role_123
Token	String	角色token 示例值： eyJrZXlJZCI6InN1bmdvxxxxx0X3JvbGVfMyJ9.dbHR8m6gc4L4vZUrodhW_O9bDulZQ6lraNswNLtcUcY
Remark	String	备注说明 示例值： devDemo
EnvironmentRoleSets	Array of EnvironmentRoleSet	批量操作环境信息 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceInUse.Role	角色已存在。
FailedOperation.CreateRole	角色创建失败。
FailedOperation	操作失败。
ResourceNotFound.BrokerCluster	服务的集群不存在。
ResourceNotFound.Cluster	集群不存在。
FailedOperation.CreateSecretKey	创建秘钥失败。
FailedOperation.SaveSecretKey	保存秘钥失败。

删除环境角色授权

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

删除环境角色授权。

默认接口请求频率限制：20次/秒。

接口更新时间：2021-10-08 16:09:02。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteEnvironmentRoles
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： devNs
RoleNames	是	否	Array of String	角色名称数组。 示例值： ["test_role"]
ClusterId	否	否	String	必填字段，集群的ID 示例值： pulsar-xk3ne8k2qkp8

3. 输出参数

参数名称	类型	描述
EnvironmentId	String	环境（命名空间）名称。 示例值： devNs

参数名称	类型	描述
RoleNames	Array of String	角色名称数组。 示例值：["test_role"]
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.EnvironmentRole	环境角色不存在。
FailedOperation.DeleteEnvironmentRoles	删除环境角色失败。

获取角色列表

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取角色列表

默认接口请求频率限制：20次/秒。

接口更新时间：2021-10-08 16:05:57。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeRoles
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
RoleName	否	否	String	角色名称，模糊查询 示例值： devRole1
Offset	否	否	Int64	起始下标，不填默认为0。 示例值： 0
Limit	否	否	Int64	返回数量，不填则默认为10，最大值为20。 示例值： 10
ClusterId	否	否	String	必填字段，集群Id 示例值： pulsar-xk3ne8k2qkp8
Filters	否	否	Array of Filter	* RoleName 按照角色名进行过滤，精确查询。 类型：String 必选：否 示例值：查看

3. 输出参数

参数名称	类型	描述
TotalCount	Int64	记录数。 示例值： 4
RoleSets	Array of Role	角色数组。 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.BrokerCluster	服务的集群不存在。
ResourceNotFound.Cluster	集群不存在。

修改环境角色授权

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

修改环境角色授权。

默认接口请求频率限制：20次/秒。

接口更新时间：2021-10-09 10:09:49。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyEnvironmentRole
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： default
RoleName	是	否	String	角色名称。 示例值： test_role
Permissions	是	否	Array of String	授权项，最多只能包含produce、consume两项的非空字符串数组。 示例值： ["produce"]
ClusterId	否	否	String	必填字段，集群的ID 示例值： pulsar-xk3ne8k2qkp8

3. 输出参数

参数名称	类型	描述
EnvironmentId	String	环境（命名空间）名称。 示例值： default
RoleName	String	角色名称。 示例值： test_role
Permissions	Array of String	授权项。 示例值： ["produce","consume"]
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.EnvironmentRole	环境角色不存在。
FailedOperation.UpdateEnvironmentRole	更新环境角色失败。

角色修改

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

角色修改

默认接口请求频率限制：20次/秒。

接口更新时间：2025-10-15 16:57:42。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyRole
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
RoleName	是	否	String	角色名称，不支持中字以及除了短线和下划线外的特殊字符且长度必须大于0且小等于32。 示例值： test_role
Remark	否	否	String	备注说明，长度必须大等于0且小等于128。 示例值： 更新角色
ClusterId	否	否	String	必填字段，集群Id 示例值： pulsar-xk3ne8k2qkp8
EnvironmentRoleSets	否	否	Array of EnvironmentRoleSet	批量操作环境 示例值：查看

参数名称	必选	允许NULL	类型	描述
UnbindAllEnvironment	否	否	Bool	是否全部解绑环境 示例值： false

3. 输出参数

参数名称	类型	描述
RoleName	String	角色名称 示例值： test_role
Remark	String	备注说明 示例值： Remark1
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceNotFound.Role	角色不存在。
FailedOperation.UpdateRole	角色更新失败。
ResourceNotFound.BrokerCluster	服务的集群不存在。

修改资源对应的项目（包含新增）

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

新增或修改资源对应的项目，新增时 oldProjectId 为空

默认接口请求频率限制：20次/秒。

接口更新时间：2021-11-11 22:10:21。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：TpoModifyResourceProject
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
ResourceProjectInfos	是	否	Array of ResourceProjectInfo	ResourceProjectInfo 数组，用于新增或修改资源对应的权限 示例值：查看

3. 输出参数

参数名称	类型	描述
ResourceProjectInfos	Array of ResourceProjectInfo	ResourceProjectInfo 数组，新增或修改成功的资源 示例值：查看

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
FailedOperation	操作失败。
InternalError	内部错误。
InternalError.CloudAPIError	云API错误。

生产消费相关接口
创建订阅关系

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

创建一个主题的订阅关系

默认接口请求频率限制：200次/秒。

接口更新时间：2021-07-27 11:43:25。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateSubscription
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： default
TopicName	是	否	String	主题名称。 示例值： cloud_test
SubscriptionName	是	否	String	订阅者名称，不支持中字以及除了短线和下划线外的特殊字符且不超过150个字符。 示例值： cloud_sub
Remark	否	否	String	备注，128个字符以内。 示例值： 创建订阅关系
IsIdempotent	是	否	Bool	是否幂等创建，若否不允许创建同名的订阅关系。 示例值： true

参数名称	必选	允许NULL	类型	描述
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-5r59xd4vnx
AutoCreatePolicyTopic	否	否	Bool	是否自动创建死信和重试主题，True 表示创建，False表示不创建，默认自动创建死信和重试主题。 示例值： true
PostFixPattern	否	否	String	指定死信和重试主题名称规范，LEGACY表示历史命名规则，COMMUNITY表示Pulsar社区命名规范 示例值： LEGACY

3. 输出参数

参数名称	类型	描述
Result	Bool	创建结果。 示例值： true
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
InternalError.Retry	重试可以成功。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Topic	主题不存在。
ResourceInUse.Subscription	重名，订阅关系已存在。
FailedOperation.GetTopicPartitionsFailed	获取主题分区数失败。

错误码	描述
FailedOperation.CreateSubscription	创建订阅关系失败。
LimitExceeded.Topics	实例下主题数量超过限制。
LimitExceeded.Subscriptions	实例下订阅者数量超过限制。
FailedOperation	操作失败。
ResourceNotFound.BrokerCluster	服务的集群不存在。

删除订阅关系

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

删除订阅关系

默认接口请求频率限制：20次/秒。

接口更新时间：2021-08-04 10:44:23。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteSubscriptions
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
SubscriptionTopicSets	是	否	Array of SubscriptionTopic	订阅关系集合，每次最多删除20个。 示例值：查看
ClusterId	否	否	String	pulsar集群Id。 示例值： pulsar-xk3ne8k2qkp8
EnvironmentId	否	否	String	环境（命名空间）名称。 示例值： devNs
Force	否	否	Bool	是否强制删除，默认为false 示例值： false

3. 输出参数

参数名称	类型	描述
SubscriptionTopicSets	Array of SubscriptionTopic	成功删除的订阅关系数组。 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Topic	主题不存在。
ResourceNotFound.Subscription	订阅关系不存在。
ResourceInUse.Subscription	重名，订阅关系已存在。
MissingParameter.NeedMoreParams	必要参数没有传递。
FailedOperation.GetTopicPartitionsFailed	获取主题分区数失败。
FailedOperation.DeleteSubscriptions	删除订阅关系失败。
ResourceNotFound.BrokerCluster	服务的集群不存在。
ResourceNotFound.Cluster	集群不存在。
OperationDenied.ConsumerRunning	订阅仍在消费中。

消费订阅列表

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

查询指定环境和主题下的订阅者列表

默认接口请求频率限制：20次/秒。

接口更新时间：2021-06-29 16:20:55。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeSubscriptions
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： devNs
TopicName	是	否	String	主题名称。 示例值： devtopic
Offset	否	否	Uint64	起始下标，不填默认为0。 示例值： 0
Limit	否	否	Uint64	返回数量，不填则默认为10，最大值为20。 示例值： 10
SubscriptionName	否	否	String	订阅者名称，模糊匹配。 示例值： devSub

参数名称	必选	允许NULL	类型	描述
Filters	否	否	Array of FilterSubscription	数据过滤条件。 示例值：查看
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-b843bz38z5mz

3. 输出参数

参数名称	类型	描述
SubscriptionSets	Array of Subscription	订阅者集合数组。 示例值：查看
TotalCount	Uint64	数量。 示例值： 1
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Environment	环境不存在。
ResourceNotFound.Topic	主题不存在。
MissingParameter.NeedMoreParams	必要参数没有传递。
FailedOperation	操作失败。
ResourceNotFound.BrokerCluster	服务的集群不存在。
ResourceNotFound.Cluster	集群不存在。

错误码	描述
FailedOperation.DescribeSubscription	查询订阅数据失败。

消息回溯

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

根据时间戳进行消息回溯，精确到毫秒

默认接口请求频率限制：20次/秒。

接口更新时间：2021-06-21 11:31:34。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ResetMsgSubOffsetByTimestamp
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
EnvironmentId	是	否	String	命名空间名称。 示例值： default
TopicName	是	否	String	主题名称。 示例值： topic
Subscription	是	否	String	订阅者名称。 示例值： subscription
ToTimestamp	是	否	Uint64	时间戳，精确到毫秒。 示例值： 1730690396152
ClusterId	否	否	String	Pulsar 集群的ID 示例值： pulsar-ge3n43v3wj

3. 输出参数

参数名称	类型	描述
Result	Bool	结果。 示例值： true
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Topic	主题不存在。
ResourceNotFound.Subscription	订阅关系不存在。
MissingParameter.NeedMoreParams	必要参数没有传递。
FailedOperation.ResetMsgSubOffsetByTimestampFailed	消息回溯设置失败。
FailedOperation	操作失败。
ResourceNotFound.BrokerCluster	服务的集群不存在。

集群相关接口
创建集群

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

创建用户的集群

默认接口请求频率限制：20次/秒。

接口更新时间：2023-04-19 11:29:14。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateCluster
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
ClusterName	是	否	String	集群名称，不支持中字以及除了短线和下划线外的特殊字符且不超过16个字符。 示例值： devName
BindClusterId	否	否	Uint64	用户专享物理集群ID，如果不传，则默认在公共集群上创建用户集群资源。 示例值： sh_cluster_1
Remark	否	否	String	说明，128个字符以内。 示例值： demo
Tags	否	否	Array of Tag	集群的标签列表 示例值：查看
ProjectId	否	否	String	集群对应的项目id 示例值： 0

参数名称	必选	允许NULL	类型	描述
BindClusterName	否	否	String	绑定集群名 示例值： sh_cluster_1

3. 输出参数

参数名称	类型	描述
ClusterId	String	集群ID 示例值： pulsar-ge3n43v3wj
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
FailedOperation	操作失败。
ResourceInUse.Cluster	集群已存在。
InternalError.SystemError	系统错误。
FailedOperation.CreateCluster	创建集群失败。
ResourceUnavailable	资源不可用。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
LimitExceeded.Clusters	实例下集群数量超过限制。
MissingParameter.NeedMoreParams	必要参数没有传递。
ResourceUnavailable.CreateFailed	The resource is abnormal, check your account.

删除集群

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

删除集群

默认接口请求频率限制：20次/秒。

接口更新时间：2021-07-20 14:46:19。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteCluster
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
ClusterId	是	否	String	集群Id，传入需要删除的集群Id。 示例值： pulsar-ge3n43v3wj

3. 输出参数

参数名称	类型	描述
ClusterId	String	集群的ID 示例值： pulsar-ge3n43v3wj
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
OperationDenied.DefaultEnvironment	默认环境不允许操作。
FailedOperation	操作失败。
FailedOperation.NamespaceInUse	必须先清除关联命名空间才能继续操作。
FailedOperation.RoleInUse	必须先清除关联角色数据才能继续操作。
FailedOperation.VpcInUse	必须先清除关联VPC路由数据才能继续操作。
FailedOperation.DeleteCluster	删除集群失败。
ResourceNotFound.Cluster	集群不存在。

获取专享集群列表

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取用户绑定的专享集群列表

默认接口请求频率限制：20次/秒。

接口更新时间：2021-06-09 15:35:05。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeBindClusters
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表

3. 输出参数

参数名称	类型	描述
TotalCount	Int64	专享集群的数量 示例值： 1
ClusterSet	Array of BindCluster	专享集群的列表 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
ResourceUnavailable.SystemUpgrade	系统升级。
UnsupportedOperation	操作不支持。
UnauthorizedOperation	未授权操作。
ResourceUnavailable	资源不可用。
ResourcesSoldOut	资源售罄。
FailedOperation	操作失败。
UnknownParameter	未知参数错误。

获取集群详情

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取集群的详细信息

默认接口请求频率限制：20次/秒。

接口更新时间：2021-06-09 15:51:04。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeClusterDetail
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
ClusterId	是	否	String	集群的ID 示例值： pulsar-ge3n43v3wj

3. 输出参数

参数名称	类型	描述
ClusterSet	Cluster	集群的详细信息 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

该接口暂无业务逻辑相关的错误码，其他错误码详见公共错误码。

获取集群列表

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

获取集群列表

默认接口请求频率限制：20次/秒。

接口更新时间：2023-04-18 17:18:12。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeClusters
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
Offset	否	否	Uint64	起始下标，不填默认为0。 示例值： 0
Limit	否	否	Uint64	返回数量，不填则默认为10，最大值为20。 示例值： 1
ClusterName	否	否	String	Pulsar 集群的名称 示例值： devCluster
ClusterIdList	否	否	Array of String	集群ID列表过滤 示例值： ["cluster001"]

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
TotalCount	Int64	集群列表数量 示例值： 1
ClusterSet	Array of Cluster	集群信息列表 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

该接口暂无业务逻辑相关的错误码，其他错误码详见公共错误码。

更新集群信息

1. 接口描述

接口请求域名： tdmq.api3.finance.cloud.tencent.com。

更新集群信息

默认接口请求频率限制：20次/秒。

接口更新时间：2021-07-20 14:45:34。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyCluster
Version	是	否	String	公共参数，本接口取值：2020-02-17
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
ClusterId	是	否	String	Pulsar 集群的ID，需要更新的集群Id。 示例值： pulsar-ge3n43v3wj
ClusterName	否	否	String	更新后的集群名称。 示例值： devName
Remark	否	否	String	说明信息。 示例值： demo

3. 输出参数

参数名称	类型	描述
ClusterId	String	Pulsar 集群的ID 示例值： pulsar-ge3n43v3wj

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
MissingParameter	缺少参数错误。
FailedOperation	操作失败。
InvalidParameterValue.ClusterNameDuplication	与现有集群名称重复。

数据结构

ConfigDictItemOpt

配置项字典

被如下接口引用：DescribeConfigDictOpt

名称	必选	允许NULL	类型	描述
ItemKey	是	否	String	配置项Key 示例值： clientPort
ItemDefaultValue	是	否	String	配置项默认值 示例值： 2181
ItemDesc	是	否	String	配置项描述 示例值： Item
IsDynamic	是	否	Bool	是否动态生成 示例值： false

AMQPVHost

vhostd信息

被如下接口引用：DescribeAMQPVHosts

名称	必选	允许NULL	类型	描述
VHostId	是	否	String	命名空间名称，3-64个字符，只能包含字母、数字、“-”及“_” 示例值： prod
MsgTtl	是	否	Uint64	未消费消息的保留时间，以毫秒单位，范围60秒到15天 示例值： 100000
Remark	是	是	String	备注 示例值： 备注
CreateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 100000
UpdateTime	是	否	Uint64	更新时间，以毫秒为单位 示例值： 100000
Username	是	否	String	用户名 示例值： xxxxx
Password	是	否	String	密码 示例值： xxxxxx

AMQPClusterDetail

租户AMQP集群详细信息

被如下接口引用：DescribeAMQPClusters

名称	必选	允许NULL	类型	描述
Info	是	否	AMQPClusterInfo	集群基本信息 示例值：查看
Config	是	否	AMQPClusterConfig	集群配置信息 示例值：查看

ConfigModuleInfoOpt

配置模块

被如下接口引用：DescribeModuleListOpt

名称	必选	允许NULL	类型	描述
ModuleName	是	否	String	模块名 示例值： broker
ComponentName	是	否	String	组件名 示例值： BROKER
ConfigName	是	否	String	配置文件名 示例值： broker.conf
IsShell	是	否	Bool	是否脚本文件 示例值： false

RocketMQNamespace

RocketMQ命名空间信息

被如下接口引用：DescribeRocketMQNamespaces

名称	必选	允许NULL	类型	描述
NamespaceId	是	否	String	命名空间名称，3-64个字符，只能包含字母、数字、“-”及“_” 示例值： prod
Ttl	是	否	Uint64	未消费消息的保留时间，以毫秒单位，范围60秒到15天 示例值： 100000
RetentionTime	是	否	Uint64	消息持久化后保留的时间，以毫秒单位 示例值： 10000
Remark	是	是	String	说明 示例值： 备注

ConfigBaseInfoOpt

集群配置基础信息

被如下接口引用：CreateConfigOpt、DescribeAllVersionConfigsOpt、DescribeClusterConfigOpt

名称	必选	允许NULL	类型	描述
ClusterName	是	否	String	集群名称 示例值： tdmq_txy_gz_01
ModuleName	是	否	String	模块名称 示例值： BROKER
ConfigVersion	是	否	String	配置版本 示例值： 1
RootPath	是	是	String	程序路径 示例值： /usr
DataPath	是	是	String	数据路径 示例值： /usr
ConfigPath	是	是	String	配置路径 示例值： /usr

名称	必选	允许NULL	类型	描述
ConfigName	是	是	String	配置文件名 示例值： broker.conf

ConsumerLog

消费日志

被如下接口引用：DescribeMsgTrace

名称	必选	允许NULL	类型	描述
MsgId	是	否	String	消息ID。 示例值： 1:1:1
ConsumerGroup	是	否	String	消费组。 示例值： group
ConsumerName	是	否	String	消费组名称。 示例值： name
ConsumeTime	是	否	String	消费时间。 示例值： 2021-11-02 17:53:00,12,654
ConsumerAddr	是	否	String	消费者客户端地址。 示例值： 1.1.1.1
ConsumeUseTime	是	否	Uint64	消费耗时（毫秒）。 示例值： 12
Status	是	否	String	消费状态。 示例值： 0

RocketMQTopic

RocketMQ主题信息

被如下接口引用：DescribeRocketMQTopics

名称	必选	允许NULL	类型	描述
Name	是	否	String	主题名称 示例值： test_topic
Type	是	否	String	主题的类别，为枚举类型，Normal，GlobalOrder，PartitionedOrder，Transaction，Retry及DeadLetter 示例值： Normal
GroupNum	是	否	Uint64	订阅组数量 示例值： 5
Remark	是	是	String	说明 示例值： 备注信息
PartitionNum	是	否	Uint64	读写分区数 示例值： 3
CreateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 100000
UpdateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 100000

RocketMQClusterDetail

租户RocketMQ集群详细信息

被如下接口引用：DescribeRocketMQClusters

名称	必选	允许NULL	类型	描述
Info	是	否	RocketMQClusterInfo	集群基本信息 示例值：查看
Config	是	否	RocketMQClusterConfig	集群配置信息 示例值：查看

TopicRecord

主题关键信息

被如下接口引用：DeleteTopics

名称	必选	允许NULL	类型	描述
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： devNs
TopicName	是	否	String	主题名称。 示例值： devTopic

Topic

主题实例

被如下接口引用：DescribeTopics

名称	必选	允许NULL	类型	描述
AverageMsgSize	是	是	String	最后一次间隔内发布消息的平均byte大小。 示例值： 10
ConsumerCount	是	是	String	消费者数量。 示例值： 120
LastConfirmedEntry	是	是	String	被记录下来的消息总数。 示例值： 100
LastLedgerCreatedTimestamp	是	是	String	最后一个ledger创建的时间。 示例值： 2024-06-19 14:15:54
MsgRateIn	是	是	String	本地和复制的发布者每秒发布消息的速率。 示例值： 130
MsgRateOut	是	是	String	本地和复制的消费者每秒分发消息的数量之和。 示例值： 140
MsgThroughputIn	是	是	String	本地和复制的发布者每秒发布消息的byte。 示例值： 160
MsgThroughputOut	是	是	String	本地和复制的消费者每秒分发消息的byte。 示例值： 120
NumberOfEntries	是	是	String	被记录下来的消息总数。 示例值： 100
Partitions	是	是	Int64	分区数<=0：topic下无子分区。 示例值： 2
ProducerCount	是	是	String	生产者数量。 示例值： 2

名称	必选	允许NULL	类型	描述
TotalSize	是	是	String	以byte计算的所有消息存储总量。 示例值： 2
SubTopicSets	是	是	Array of PartitionsTopic	分区topic里面的子分区。 示例值：查看
TopicType	是	是	Uint64	topic类型描述： 0：普通消息； 1：全局顺序消息； 2：局部顺序消息； 3：重试队列； 4：死信队列； 5：事务消息。 示例值： 0
EnvironmentId	是	是	String	环境（命名空间）名称。 示例值： devNs
TopicName	是	是	String	主题名称。 示例值： devTopic
Remark	是	是	String	说明，128个字符以内。 示例值： devRemark
CreateTime	是	是	Datetime	创建时间。 示例值： 2024-06-19 14:15:54
UpdateTime	是	是	Datetime	最近修改时间。 示例值： 2024-06-19 14:15:54
ProducerLimit	是	是	String	生产者上限。 示例值： 5000
ConsumerLimit	是	是	String	消费者上限。 示例值： 5000
PulsarTopicType	否	是	Int64	0: 非持久非分区 1: 非持久分区 2: 持久非分区 3: 持久分区 示例值： 1
ReplicationStatus	否	是	String	topic所在集群的消息同步状态OriginReplicationOpen 源集群开启消息同步OriginReplicationClose 源集群关闭消息同步TargetReplicationOpen 目标集群开启消息同步TargetReplicationClose 目标集群关闭消息同步 示例值： success
MsgTTL	否	是	Uint64	未消费消息过期时间，单位：秒 示例值： 60

AgentNodeOpt

代理节点

被如下接口引用：DescribeAgentNodesOpt

名称	必选	允许NULL	类型	描述
ClusterName	是	否	String	集群名称 示例值： tdmq_txy_gz_01
ModuleName	是	否	String	模块名称 示例值： ZOOKEEPER_BOOKIE
AgentId	是	否	String	代理ID 示例值： 10.73.13.46-fqLzhQ6B
AgentIp	是	否	String	代理IP 示例值： 10.73.13.46

名称	必选	允许NULL	类型	描述
ProcId	是	否	Int64	协议ID 示例值： 8451
AgentVersion	是	否	String	代理版本 示例值： 0.0.1-SNAPSHOT
JavaVersion	是	否	String	JDK版本 示例值： 1.8.0_275

AMQPExchange

AMQP Exchange信息

被如下接口引用：DescribeAMQPExchanges

名称	必选	允许NULL	类型	描述
Name	是	否	String	Exchange名称 示例值： test
Type	是	否	String	Exchange的类别，为枚举类型:Direct, Fanout, Topic 示例值： Default
SourceBindedNum	是	否	Uint64	主绑定数 示例值： 1
Remark	是	是	String	说明 示例值： 备注
DestBindedNum	是	否	Uint64	被绑定数 示例值： 1
CreateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 100000
UpdateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 100000
Internal	是	否	Bool	是否为内部Exchange(以amq.前缀开头的) 示例值： false

SendConfigInfoOpt

下发配置结果

被如下接口引用：SendAllModulesConfigOpt、SendConfigOpt

名称	必选	允许NULL	类型	描述
NodeIp	是	否	String	节点ip地址 示例值： 192.168.1.1
ClusterName	是	否	String	集群名 示例值： test-cluster
ModuleName	是	是	String	模块名 示例值： BROKER
ConfigVersion	是	是	String	配置版本 示例值： 1
Status	是	否	Bool	下发状态 示例值： true

Filter

过滤参数

被如下接口引用：DescribeEnvironmentRoles、DescribeEnvironments、DescribeRoles、DescribeTopics

名称	必选	允许NULL	类型	描述
Name	否	否	String	过滤参数的名字 示例值： paramName
Values	否	否	Array of String	数值 示例值： ["a"]

ServerLog

服务方信息

被如下接口引用：DescribeMsgTrace

名称	必选	允许NULL	类型	描述
SaveTime	是	否	String	存储时间。 示例值： 2024-10-08 11:38:48,068
Status	是	否	String	状态。 示例值： success

Environment

命名空间信息

被如下接口引用：DescribeEnvironments

名称	必选	允许NULL	类型	描述
EnvironmentId	是	否	String	命名空间名称 示例值： devNs
Remark	是	否	String	说明 示例值： devRemark
MsgTTL	是	否	Int64	未消费消息过期时间，单位：秒，最大1296000（15天） 示例值： 2
CreateTime	是	否	Datetime	创建时间 示例值： 2020-02-17 00:00:00
UpdateTime	是	否	Datetime	最近修改时间 示例值： 2020-02-17 00:00:00
NamespaceId	是	否	String	命名空间ID 示例值： devName1
NamespaceName	是	否	String	命名空间名称 示例值： devName1
TopicNum	是	是	Int64	Topic数量 示例值： 2
RetentionPolicy	是	是	RetentionPolicy	消息保留策略 示例值：查看

Cluster

集群信息集合

被如下接口引用：DescribeClusterDetail、DescribeClusters

名称	必选	允许NULL	类型	描述
ClusterId	是	否	String	集群Id。 示例值： 1000
ClusterName	是	否	String	集群名称。 示例值： pulsar-47emajijmazk
Remark	是	否	String	说明信息。 示例值： devInfo
EndPointNum	是	否	Int64	接入点数量 示例值： 0
CreateTime	是	否	String	创建时间 示例值： 2021-04-21 19:00:00
Healthy	是	否	Int64	集群是否健康，1表示健康，0表示异常 示例值： 1
HealthyInfo	是	是	String	集群健康信息 示例值： healthy
Status	是	否	Int64	集群状态，0:创建中，1:正常，2:删除中，3:已删除，5:创建失败，6: 删除失败 示例值： 1
MaxNamespaceNum	是	否	Int64	最大命名空间数量 示例值： 100
MaxTopicNum	是	否	Int64	最大Topic数量 示例值： 8000
MaxQps	是	否	Int64	最大QPS 示例值： 1000
MessageRetentionTime	是	否	Int64	消息保留时间 示例值： 100000000
MaxStorageCapacity	是	否	Int64	最大存储容量 示例值： 1024
Version	是	是	String	集群版本 示例值： 2.5.1
PublicEndPoint	是	是	String	公网访问接入点 示例值： http://pulsar-45epqe3j7jvz.sap-8iutas0k.tdmq.ap-gz.qcloud.xxx.com:8080
VpcEndPoint	是	是	String	VPC访问接入点 示例值： http://pulsar-45epqe3j7jvz.sap-8iutas0k.tdmq.ap-gz.qcloud.xxx.com:8080
NamespaceNum	是	是	Int64	命名空间数量 示例值： 2
UsedStorageBudget	是	是	Int64	已使用存储限制，MB为单位 示例值： 0
ProjectId	否	是	String	资源对应项目id 示例值： 251198839
ProjectName	否	是	String	资源对应项目名称 示例值： product01
PayMode	是	否	Int64	计费模型0:按量计费，1：包年包月 示例值： 0

名称	必选	允许NULL	类型	描述
MaxPublishRateInMessages	否	是	Int64	最大生产消息速率，以条数为单位 注意：此字段可能返回 null，表示取不到有效值。 示例值： 1
PclusterName	否	是	String	集群名字 示例值： test-Id
PublicAccessEnabled	否	是	Bool	是否开启公网访问，不填时默认开启 注意：此字段可能返回 null，表示取不到有效值。 示例值： true
MaxDispatchRateInBytes	否	是	Int64	命名空间最大消费带宽，byte为单位 注意：此字段可能返回 null，表示取不到有效值。 示例值： 1
Tags	否	是	Array of Tag	标签 示例值：查看
TopicNum	否	否	Int64	topic数量 示例值： 1
InternalEndPoint	否	是	String	内部接入点 示例值： pulsar://1.1.1.1:6650
MaxDispatchRateInMessages	否	是	Int64	最大推送消息速率，以条数为单位 注意：此字段可能返回 null，表示取不到有效值。 示例值： 1
InternalPulsarEndPoint	否	是	String	内部pulsar-sdk接入地址 示例值： pulsar://tdmq.ap-gz.xxx.com:6650
MaxPublishRateInBytes	否	是	Int64	最大生产消息速率，以字节为单位 注意：此字段可能返回 null，表示取不到有效值。 示例值： 1
MaxMessageDelayInSeconds	否	是	Int64	最长消息延时，以秒为单位 注意：此字段可能返回 null，表示取不到有效值。 示例值： 1
InternalHttpEndPoint	否	是	String	内部http接入地址 示例值： http://tdmq.ap-gz.xxx.com:8080

ConfigTemplateItemOpt

模板配置项

被如下接口引用：CreateTemplateItemsOpt

名称	必选	允许NULL	类型	描述
ItemKey	是	否	String	配置属性 示例值： key
ItemValue	是	否	String	属性值 示例值： value
ItemDesc	否	否	String	配置项备注 示例值： desc
IsDynamic	否	否	Bool	是否动态配置 示例值： true

Tag

标签的key/value的类型

被如下接口引用：CreateCluster、DescribeClusterDetail、DescribeClusters

名称	必选	允许NULL	类型	描述

名称	必选	允许NULL	类型	描述
TagKey	是	否	String	标签的key的值 示例值： test_tag_key
TagValue	是	否	String	标签的Value的值 示例值： test_tag_value1

ConsumerLogs

消费信息

被如下接口引用：DescribeMsgTrace

名称	必选	允许NULL	类型	描述
TotalCount	是	是	Uint64	记录数。 示例值： 1
ConsumerLogSets	是	是	Array of ConsumerLog	消费日志。 示例值：查看

Subscription

订阅者

被如下接口引用：DescribeSubscriptions

名称	必选	允许NULL	类型	描述
TopicName	是	否	String	主题名称。 示例值： devTopic
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： devNs
ConnectedSince	是	是	String	消费者开始连接的时间。 示例值： 2024-09-12T16:01:59.677+08:00
ConsumerAddr	是	是	String	消费者地址。 示例值： 127.0.0.1
ConsumerCount	是	是	String	消费者数量。 示例值： 11
ConsumerName	是	是	String	消费者名称。 示例值： devConsumerName
MsgBacklog	是	是	String	堆积的消息数量。 示例值： 1
MsgRateExpired	是	是	String	于TTL，此订阅下没有被发送而是被丢弃的比例。 示例值： 1
MsgRateOut	是	是	String	消费者每秒分发消息的数量之和。 示例值： 1
MsgThroughputOut	是	是	String	消费者每秒消息的byte。 示例值： 1
SubscriptionName	是	是	String	订阅名称。 示例值： devSubscriptionName
ConsumerSets	是	是	Array of Consumer	消费者集合。 示例值：查看

名称	必选	允许NULL	类型	描述
IsOnline	是	是	Bool	是否在线。 示例值： true
ConsumersScheduleSets	是	是	Array of ConsumersSchedule	消费进度集合。 示例值：查看
Remark	是	是	String	备注。 示例值： devRemark
CreateTime	是	是	Datetime	创建时间。 示例值： 2023-10-05 14:30:00
UpdateTime	是	是	Datetime	最近修改时间。 示例值： 2023-12-05 14:30:00
BlockedSubscriptionOnUnackedMsgs	否	是	Bool	是否由于未 ack 数到达上限而被 block 注意：此字段可能返回 null，表示取不到有效值。 示例值： true
SubType	否	是	String	订阅类型，Exclusive，Shared，Failover， Key_Shared，空或NULL表示未知， 注意：此字段可能返回 null，表示取不到有效值。 示例值： Shared
MaxUnackedMsgNum	否	是	Int64	未 ack 消息数上限 注意：此字段可能返回 null，表示取不到有效值。 示例值： 5000

AMQPClusterRecentStats

AMQP集群近期使用量

被如下接口引用：DescribeAMQPCluster

名称	必选	允许NULL	类型	描述
QueueNum	是	否	Uint64	Queue数量 示例值： 20
ProducedMsgNum	是	否	Uint64	消息生产数 示例值： 11000
AccumulativeMsgNum	是	否	Uint64	消息堆积数 示例值： 22000
ExchangeNum	是	否	Uint64	Exchange数量 示例值： 20

RetentionPolicy

消息保留策略

被如下接口引用：CreateEnvironment、DescribeEnvironments、ModifyEnvironmentAttributes

名称	必选	允许NULL	类型	描述
TimeInMinutes	是	否	Int64	消息保留时长 示例值： 0
SizeInMB	是	否	Int64	消息保留大小 示例值： 0

TemplateItemInfoOpt

模板配置项

被如下接口引用：DescribeTemplateItemOpt

名称	必选	允许NULL	类型	描述
ItemKey	是	否	String	配置属性名 示例值： key
ItemDefaultValue	是	是	String	配置项默认值 示例值： value
ItemValue	是	是	String	模板配置项值 示例值： value
ItemDesc	是	是	String	描述 示例值： desc
IsDynamic	是	否	Bool	是否动态配置，0需要重启，1不需要重启 示例值： 0

AMQPVHostConnection

Vhost客户端连接情况

被如下接口引用：DescribeAMQPVHostConnections

名称	必选	允许NULL	类型	描述
ClientAddr	是	否	String	客户端实例的地址和端口 示例值： 192.168.0.1:5555
Protocol	是	否	String	客户端协议 示例值： AMQP 0.9.1
Security	是	否	Bool	是否支持SSL/TLS 示例值： true
ChannelNum	是	否	Uint64	信道数 示例值： 2
ReceiveRate	是	否	Uint64	接受速率 示例值： 102
SendRate	是	否	Uint64	发送速率 示例值： 97

AMQPQueueConsumer

AMQP队列消费者信息

被如下接口引用：DescribeAMQPQueueConsumers

名称	必选	允许NULL	类型	描述
ClientAddr	是	否	String	消费者实例的地址和端口 示例值： 192.168.0.1:5555
ClientTag	是	否	String	消费者标签 示例值： consumer1
CreateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 100000

MsgLog

消息日志

被如下接口引用：DescribeTopicMsgs

名称	必选	允许NULL	类型	描述
MsgId	否	是	String	消息ID。 示例值： 931087:81:0
ProducerName	否	是	String	生产者名称。 示例值： devProducerName
ProduceTime	否	是	String	生产时间。 示例值： 2024-10-08 08:04:13,809
ProducerAddr	否	是	String	生产客户端地址。 示例值： 10.0.0.120:38168

EnvironmentRole

环境角色集合

被如下接口引用：DescribeEnvironmentRoles

名称	必选	允许NULL	类型	描述
EnvironmentId	是	否	String	环境（命名空间）。 示例值： test_namespace
RoleName	是	否	String	角色名称。 示例值： test_role_name
Permissions	是	否	Array of String	授权项，最多只能包含produce、consume两项的非空字符串数组。 示例值： ["produce"]
RoleDescribe	是	否	String	角色描述。 示例值： 测试角色
CreateTime	是	否	Datetime	创建时间。 示例值： 2023-07-18 17:06:57
UpdateTime	是	否	Datetime	更新时间。 示例值： 2023-07-18 17:06:57

VpcBindRecord

vcp绑定记录

被如下接口引用：DescribeBindVpcs

名称	必选	允许NULL	类型	描述
UniqueVpcId	是	否	String	租户Vpc Id 示例值： vpc-8jjiuayds
UniqueSubnetId	是	否	String	租户Vpc子网Id 示例值： subnet-1i12ofyz
RouterId	是	否	String	路由Id 示例值： 124323
Ip	是	否	String	Vpc的Id 示例值： 10.0.0.120
Port	是	否	Uint64	Vpc的Port 示例值： 8080

名称	必选	允许NULL	类型	描述
Remark	是	是	String	说明，128个字符以内 示例值： devRemark

RocketMQConsumerConnection

在线消费者情况

被如下接口引用：DescribeRocketMQConsumerConnections

名称	必选	允许NULL	类型	描述
ClientId	是	否	String	消费者实例ID 示例值： rocketmq-2p9vx3ax9jxg
ClientAddr	是	否	String	消费者实例的地址和端口 示例值： 192.168.0.1:5555
Language	是	否	String	消费者应用的语言版本 示例值： JAVA
Accumulative	是	否	Int64	消息堆积量 示例值： 100
Version	是	否	String	消费端版本 示例值： V4.9.7

RocketMQGroup

RocketMQ消费组信息

被如下接口引用：DescribeRocketMQGroups

名称	必选	允许NULL	类型	描述
Name	是	否	String	消费组名称 示例值： test_group
ConsumerNum	是	否	Uint64	在线消费者数量 示例值： 2
TPS	是	否	Uint64	消费TPS 示例值： 100
TotalAccumulative	是	否	Int64	总堆积数量 示例值： 380
ConsumptionMode	是	否	Int64	0表示集群消费模式，1表示广播消费模式，-1表示未知 示例值： 0
ReadEnabled	是	否	Bool	是否允许消费 示例值： true
RetryPartitionNum	是	是	Uint64	重试队列分区数 示例值： 2
CreateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 1729478721
UpdateTime	是	否	Uint64	修改时间，以毫秒为单位 示例值： 1729478721
ClientProtocol	是	否	String	客户端协议 示例值： tcp

名称	必选	允许NULL	类型	描述
Remark	是	是	String	说明信息 示例值： 备注信息
ConsumerType	是	是	String	消费者类型，枚举值ACTIVELY, PASSIVELY 示例值： PUSH
BroadcastEnabled	是	否	Bool	是否开启广播消费 示例值： false

FilterSubscription

过滤订阅列表

被如下接口引用：DescribeSubscriptions

名称	必选	允许NULL	类型	描述
ConsumerHasCount	否	否	Bool	是否仅展示包含真实消费者的订阅。 示例值： false
ConsumerHasBacklog	否	否	Bool	是否仅展示消息堆积的订阅。 示例值： false
ConsumerHasExpired	否	否	Bool	是否仅展示存在消息超期丢弃的订阅。 示例值： false
SubscriptionNames	否	否	Array of String	按照订阅名过滤，精确查询。 示例值： false

RocketMQConsumerTopic

消费者详情中的主题信息

被如下接口引用：DescribeRocketMQConsumerConnectionDetail

名称	必选	允许NULL	类型	描述
Topic	是	否	String	主题名称 示例值： test_topic
Type	是	否	String	主题类型，Default表示普通，GlobalOrder表示全局顺序，PartitionedOrder表示局部顺序，Transaction表示事务，Retry表示重试，DeadLetter表示死信 示例值： Normal
PartitionNum	是	否	Uint64	分区数 示例值： 3
Accumulative	是	否	Int64	消息堆积数 示例值： 100
LastConsumptionTime	是	否	Uint64	最后消费时间，以毫秒为单位 示例值： 1697945990000
SubRule	是	是	String	订阅规则 示例值： test_tag

DeliveryInfoOpt

配置下发状态信息

被如下接口引用：DescribeDeliveryStatusOpt

名称	必选	允许NULL	类型	描述

名称	必选	允许NULL	类型	描述
DeliveryIp	是	否	String	代理节点IP 示例值： 10.73.13.97
DeliveryTimes	是	是	Int64	下发时间 示例值： 1
DeliveryStatus	是	否	Int64	下发状态（实际大小1字节） 0 下发中 1 陈工 2失败 示例值： 2
ConfigVersion	是	否	String	版本 示例值： 1
AddTime	是	否	Int64	新增时间 示例值： 1111

BindCluster

用户专享集群信息

被如下接口引用：DescribeBindClusters

名称	必选	允许NULL	类型	描述
ClusterId	是	否	Int64	物理集群的ID 示例值： pulsar-mw4qmgaq298g
ClusterName	是	否	String	物理集群的名称 示例值： devClusterName

Role

角色实例

被如下接口引用：DescribeRoles

名称	必选	允许NULL	类型	描述
RoleName	是	否	String	角色名称。 示例值： test_role_name
Token	是	否	String	角色token值。 示例值： eyJrZXlJZCI6InN1bmdvxxxxx0X3JvbGVfMyJ9.dbHR8m6gc4L4vZUrodhW_O9bDulZQ6lraNswNLtcUcY
Remark	是	否	String	备注说明。 示例值： 测试账号
CreateTime	是	否	Datetime	创建时间。 示例值： 2020-09-22 00:00:00
UpdateTime	是	否	Datetime	更新时间。 示例值： 2020-09-22 00:00:00

SubscriptionTopic

订阅关系

被如下接口引用：DeleteSubscriptions

名称	必选	允许NULL	类型	描述
EnvironmentId	是	否	String	环境（命名空间）名称。 示例值： devNs

名称	必选	允许NULL	类型	描述
TopicName	是	否	String	主题名称。 示例值： devTopic
SubscriptionName	是	否	String	订阅名称。 示例值： devSub

RocketMQClusterRecentStats

RocketMQ近期使用量

被如下接口引用：DescribeRocketMQCluster

名称	必选	允许NULL	类型	描述
TopicNum	是	否	Uint64	Topic数量 示例值： 20
ApiRequestNum	是	否	Uint64	API请求数 示例值： 538000
ProducedMsgNum	是	否	Uint64	消息生产数 示例值： 11000
ConsumedMsgNum	是	否	Uint64	消息消费数 示例值： 11200
AccumulativeMsgNum	是	否	Uint64	消息堆积数 示例值： 22000

Consumer

消费者

被如下接口引用：DescribeSubscriptions

名称	必选	允许NULL	类型	描述
ConnectedSince	是	是	String	消费者开始连接的时间。 示例值： 2024-04-11T23:03:47.328+08:00
ConsumerAddr	是	是	String	消费者地址。 示例值： 30.11.19.46:40940
ConsumerName	是	是	String	消费者名称。 示例值： devName1
ClientVersion	是	是	String	消费者版本。 示例值： 2.7.1
Partition	否	是	Int64	Partition数量 示例值： 0

ConsumersSchedule

消费进度详情

被如下接口引用：DescribeSubscriptions

名称	必选	允许NULL	类型	描述
Partitions	是	是	Uint64	当前分区id。 示例值： 1

名称	必选	允许NULL	类型	描述
NumberOfEntries	是	是	Uint64	消息数量。 示例值： 1000
MsgBacklog	是	是	Uint64	消息积压数量。 示例值： 2
MsgRateOut	是	否	String	消费者每秒分发消息的数量之和。 示例值： 0.0
MsgThroughputOut	是	否	String	消费者每秒消息的byte。 示例值： 0.0
MsgRateExpired	是	是	String	超时丢弃比例。 示例值： 0.0

AMQPClusterInfo

AMQP集群基本信息

被如下接口引用：DescribeAMQPCluster、DescribeAMQPClusters

名称	必选	允许NULL	类型	描述
ClusterId	是	否	String	集群ID 示例值： amqp-xxxxx
ClusterName	是	否	String	集群名称 示例值： test
Region	是	否	String	地域信息 示例值： 某地区
CreateTime	是	否	Uint64	创建时间，毫秒为单位 示例值： 100000000
Remark	是	是	String	集群说明信息 示例值： 备注
PublicEndPoint	是	是	String	公网接入地址 示例值： xxxx
VpcEndPoint	是	是	String	VPC接入地址 示例值： xxxxx

ResourceProjectInfo

新增或修改资源对应的项目，新增时 oldProjectid 为空，修改时oldProjectId 为当前资源对应的 projectId

被如下接口引用：TpoModifyResourceProject

名称	必选	允许NULL	类型	描述
ResourceId	是	否	String	资源id 示例值： pulasr-11123
OldProjectId	否	否	String	旧项目id 示例值： pr-106602ce
NewProjectId	是	否	String	新项目id 示例值： pr-106602ce7

AMQPRouteRelation

AMQP路由关系

被如下接口引用：DescribeAMQPRouteRelations

名称	必选	允许NULL	类型	描述
RouteRelationId	是	否	String	路由关系ID 示例值： test
SourceExchange	是	否	String	源Exchange 示例值： xxx
DestType	是	否	String	目标类型:Queue
DestValue	是	否	String	目标值 示例值： queue1
RoutingKey	是	否	String	绑定key 示例值： xxxc
SourceExchangeType	是	否	String	源路由类型:Direct
CreateTime	是	否	Uint64	创建时间，以毫秒为单位 示例值： 10000
UpdateTime	是	否	Uint64	修改时间，以毫秒为单位 示例值： 修改时间
Remark	是	是	String	说明信息 示例值： 备注

ProducerLog

消息生产信息

被如下接口引用：DescribeMsgTrace

名称	必选	允许NULL	类型	描述
MsgId	是	否	String	消息ID。 示例值： 6054156:22067:16
ProducerName	否	否	String	生产者名称。 示例值： pulsar-7dqvnqwjgzm-2156-9
ProduceTime	是	否	String	消息生产时间。 示例值： 2024-10-08 14:45:52,636
ProduceUseTime	是	否	Uint64	生产耗时（秒）。 示例值： 2
Status	是	否	String	状态。 示例值： success

AMQPClusterConfig

AMQP集群配置

被如下接口引用：DescribeAMQPCluster、DescribeAMQPClusters

名称	必选	允许NULL	类型	描述
MaxTpsPerVHost	是	否	Uint64	单Vhost TPS上限 示例值： 8000
MaxConnNumPerVHost	是	否	Uint64	单Vhost客户端连接数上限 示例值： 8000

名称	必选	允许NULL	类型	描述
MaxVHostNum	是	否	Uint64	最大Vhost数量 示例值： 10
MaxExchangeNum	是	否	Uint64	最大exchange数量 示例值： 100
MaxQueueNum	是	否	Uint64	最大Queue数量 示例值： 100
MaxRetentionTime	是	否	Uint64	消息最大保留时间，以毫秒为单位 示例值： 100000
UsedVHostNum	是	否	Uint64	已使用Vhost数量 示例值： 3
UsedExchangeNum	是	否	Uint64	已使用exchange数量 示例值： 3
UsedQueueNum	是	否	Uint64	已使用queue数量 示例值： 8

RocketMQClusterConfig

RocketMQ集群配置

被如下接口引用：DescribeRocketMQCluster、DescribeRocketMQClusters

名称	必选	允许NULL	类型	描述
MaxTpsPerNamespace	是	否	Uint64	单命名空间TPS上线 示例值： 8000
MaxNamespaceNum	是	否	Uint64	最大命名空间数量 示例值： 10
UsedNamespaceNum	是	否	Uint64	已使用命名空间数量 示例值： 3
MaxTopicNum	是	否	Uint64	最大Topic数量 示例值： 1000
UsedTopicNum	是	否	Uint64	已使用Topic数量 示例值： 8
MaxGroupNum	是	否	Uint64	最大Group数量 示例值： 10000
UsedGroupNum	是	否	Uint64	已使用Group数量 示例值： 10
MaxRetentionTime	是	否	Uint64	消息最大保留时间，以毫秒为单位 示例值： 100000
MaxLatencyTime	是	否	Uint64	消息最长延时，以毫秒为单位 示例值： 100000

PartitionsTopic

分区topic

被如下接口引用：DescribeTopics

名称	必选	允许NULL	类型	描述

名称	必选	允许NULL	类型	描述
AverageMsgSize	是	是	String	最后一次间隔内发布消息的平均byte大小。 示例值： 10
ConsumerCount	是	是	String	消费者数量。 示例值： 10
LastConfirmedEntry	是	是	String	被记录下来的消息总数。 示例值： 10
LastLedgerCreatedTimestamp	是	是	String	最后一个ledger创建的时间。 示例值： 2024-10-08 08:04:13,809
MsgRateIn	是	是	String	本地和复制的发布者每秒发布消息的速率。 示例值： 10
MsgRateOut	是	是	String	本地和复制的消费者每秒分发消息的数量之和。 示例值： 10
MsgThroughputIn	是	是	String	本地和复制的发布者每秒发布消息的byte。 示例值： 10
MsgThroughputOut	是	是	String	本地和复制的消费者每秒分发消息的byte。 示例值： 10
NumberOfEntries	是	是	String	被记录下来的消息总数。 示例值： 10
Partitions	是	是	Int64	子分区id。 示例值： 2
ProducerCount	是	是	String	生产者数量。 示例值： 2
TotalSize	是	是	String	以byte计算的所有消息存储总量。 示例值： 2
TopicType	是	是	Uint64	topic类型描述。 示例值： 0

EnvironmentRoleSet

角色批量绑定命名空间信息

被如下接口引用：CreateRole、ModifyRole

名称	必选	允许NULL	类型	描述
EnvironmentId	是	否	String	命名空间Id 示例值： devNs
Permissions	是	否	Array of String	权限枚举值 示例值： ["produce"]
ClusterId	否	是	String	绑定的集群Id 示例值： pulsar-jmw4o94a3z82
RoleName	否	是	String	角色名字 示例值： role-a

AMQPQueueDetail

AMQP 队列信息

被如下接口引用：DescribeAMQPQueues

名称	必选	允许NULL	类型	描述
Name	是	否	String	Queue名称 示例值： test
Remark	是	是	String	说明 示例值： 备注
DestBindedNum	是	是	Uint64	被绑定数 示例值： 1
CreateTime	是	是	Uint64	创建时间，以毫秒为单位 示例值： 100000
UpdateTime	是	是	Uint64	创建时间，以毫秒为单位 示例值： 100000
OnlineConsumerNum	是	是	Uint64	在线消费者数 示例值： 20
Tps	是	是	Uint64	每秒钟的事务数 示例值： 120
AccumulativeMsgNum	是	是	Uint64	消息堆积数 示例值： 90
AutoDelete	是	是	Bool	是否自动删除 示例值： true
DeadLetterExchange	是	是	String	死信交换机 示例值： test
DeadLetterRoutingKey	是	是	String	死信交换机路由键 示例值： test

ConfigTypeInfoOpt

集群模块配置类型描述

被如下接口引用：DescribeClusterConfigTypeOpt

名称	必选	允许NULL	类型	描述
ComponentName	是	否	String	模块组件名称 示例值： BROKER
ConfigName	是	否	String	模块配置文件名称 示例值： broker.conf
ModuleName	是	否	String	配置模块名 示例值： BROKER

RocketMQClusterInfo

RocketMQ集群基本信息

被如下接口引用：DescribeRocketMQCluster、DescribeRocketMQClusters

名称	必选	允许NULL	类型	描述
ClusterId	是	否	String	集群ID 示例值： rocketmq-4k4orqgq
ClusterName	是	否	String	集群名称 示例值： rocket-order-instance

名称	必选	允许NULL	类型	描述
Region	是	否	String	地域信息 示例值： 某地区
CreateTime	是	否	Uint64	创建时间，毫秒为单位 示例值： 1729478721
Remark	是	是	String	集群说明信息 示例值： info-remark
PublicEndPoint	是	否	String	公网接入地址 示例值： rocketmq.access.public.com:9867
VpcEndPoint	是	否	String	VPC接入地址 示例值： rocketmq.resource.vpc.com:5010

RouteRecord

路由记录

被如下接口引用：DescribeRoute

名称	必选	允许NULL	类型	描述
VpcId	是	是	String	私有网络 示例值： xxx
SubnetId	是	是	String	子网id 示例值： xxx
RouterId	是	否	String	路由id 示例值： pulsar-xxx
Vip	是	否	String	vip地址 示例值： 10.10.10.10
Vport	是	否	Uint64	vport记录 示例值： 6000
Remark	是	是	String	备注信息 示例值： xxx
NetType	是	否	Uint64	路由类型 1: 公网 2：vpc 4：支撑网 示例值： 1
Id	否	否	Uint64	路由Id（int 类型） 示例值： route-ID

错误码

功能说明

如果返回结果中存在 Error 字段，则表示调用 API 接口失败。例如：

```
{
  "Response": {
    "Error": {
      "Code": "AuthFailure.SignatureFailure",
      "Message": "The provided credentials could not be validated. Please check your signature is cor
rect."
    },
    "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
  }
}
```

Error 中的 Code 表示错误码，Message 表示该错误的具体信息。

错误码列表

公共错误码

错误码	说明
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。 请在控制台检查密钥是否已被删除或者禁用，如状态正常，请检查密钥是否填写正确，注意前后不得有空格。
AuthFailure.SignatureExpire	签名过期。Timestamp 和服务器时间相差不得超过五分钟，请检查本地时间是否和标准时间同步。
AuthFailure.SignatureFailure	签名错误。 签名计算错误，请对照调用方式中的接口鉴权文档检查签名计算过程。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。

错误码	说明
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	http(s)请求协议错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

业务错误码

错误码	说明
FailedOperation.DeleteRoles	角色删除失败。
FailedOperation.RoleInUse	必须先清除关联角色数据才能继续操作。
InternalError.CloudAPIError	云API错误。
ResourceUnavailable	资源不可用。

错误码	说明
ResourceNotFound.Msg	找不到指定消息。
ResourceUnavailable.CreateFailed	The resource is abnormal, check your account.
UnauthorizedOperation	未授权操作。
ResourceInUse.BindVpc	vpc绑定关系已存在。
InvalidParameter.TenantNotFound	上传的 tenant name 错误。
FailedOperation.CreateEnvironment	环境创建失败。
FailedOperation.CreateNamespace	创建命名空间失败。
LimitExceeded.Clusters	实例下集群数量超过限制。
ResourceNotFound.Cluster	集群不存在。
ResourceInUse.VpcRoute	vpc路由已存在。
LimitExceeded	超过配额限制。
ResourceUnavailable.SystemUpgrade	系统升级。
OperationDenied.ConsumerRunning	订阅仍在消费中。
InvalidParameter	参数错误。
FailedOperation.GetTopicPartitionsFailed	获取主题分区数失败。
InvalidParameterValue.ClusterNameDuplication	与现有集群名称重复。
FailedOperation.TopicInUse	必须先清除关联主题数据才能继续操作。
ResourceNotFound.EnvironmentRole	环境角色不存在。
LimitExceeded.Subscriptions	实例下订阅者数量超过限制。
FailedOperation.CreateSecretKey	创建秘钥失败。
ResourceInUse.SupportRoute	支撑路由已存在。
FailedOperation.UpdateEnvironmentRole	更新环境角色失败。
ResourceInUse.Topic	重名，主题已存在。
FailedOperation.UpdateEnvironment	环境更新失败。

错误码	说明
ResourceInUse.EnvironmentRole	环境角色已存在。
InternalError.GetAttributesFailed	获取属性失败。
ResourceNotFound.Environment	环境不存在。
ResourceNotFound.Topic	主题不存在。
FailedOperation.CreateBindVpc	创建vpc绑定关系失败。
FailedOperation.ResetMsgSubOffsetByTimestampFailed	消息回溯设置失败。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TTL	无效的消息TTL值。
ResourceNotFound.MsgProduceLog	消息生产日志不存在。
LimitExceeded.Topics	实例下主题数量超过限制。
LimitExceeded.Namespaces	实例下命名空间数量超过限额。
InvalidParameterValue.InvalidParams	参数值不在允许范围内。
InternalError.SystemError	系统错误。
ResourceInUse.Role	角色已存在。
ResourceInUse	资源被占用。
FailedOperation.SaveSecretKey	保存秘钥失败。
ResourceNotFound.Role	角色不存在。
ResourceInUse.Namespace	重名，命名空间已存在。
MissingParameter	缺少参数错误。
AuthFailure.UnauthorizedOperation	CAM鉴权不通过。
ResourceNotFound.Subscription	订阅关系不存在。
InvalidParameterValue.TopicNotFound	上传的topic name错误。
FailedOperation.CreateProducerError	创建producer出错。
InternalError.VpcService	VPC服务异常。
ResourceNotFound.BindVpc	vpc绑定关系不存在。

错误码	说明
InternalError	内部错误。
FailedOperation.SendMsgFailed	发送消息失败。
FailedOperation	操作失败。
FailedOperation.UpdateRole	角色更新失败。
FailedOperation.DeleteTopics	主题删除失败。
InternalError.BrokerService	Broker服务异常。
FailedOperation.SetTTL	设置消息TTL失败。
FailedOperation.DeleteEnvironmentRoles	删除环境角色失败。
OperationDenied.DefaultEnvironment	默认环境不允许操作。
FailedOperation.DeleteSubscriptions	删除订阅关系失败。
FailedOperation.DeleteCluster	删除集群失败。
FailedOperation.DescribeProducers	查询生产者数据失败。
FailedOperation.NamespaceInUse	必须先清除关联命名空间才能继续操作。
FailedOperation.CreateTopic	主题创建失败。
FailedOperation.CreateRole	角色创建失败。
LimitExceeded.Environments	实例下环境数量超过限制。
FailedOperation.DeleteEnvironments	环境删除失败。
FailedOperation.CreatePulsarClientError	创建TDMQ client的出错。
FailedOperation.VpcInUse	必须先清除关联VPC路由数据才能继续操作。
ResourceInUse.Subscription	重名，订阅关系已存在。
ResourceNotFound.Namespace	命名空间不存在。
UnsupportedOperation	操作不支持。
FailedOperation.DeleteNamespace	删除命名空间失败。
FailedOperation.ReceiveTimeout	接收消息超时，请重试。
FailedOperation.CreateEnvironmentRole	创建环境角色失败。

错误码	说明
FailedOperation.CreateCluster	创建集群失败。
InvalidParameterValue.NeedMoreParams	必要参数没有传递。
ResourceInUse.Cluster	集群已存在。
ResourceInsufficient	资源不足。
FailedOperation.DescribeSubscription	查询订阅数据失败。
InvalidParameter.TokenNotFound	没有获取到正确的 token。
ResourceNotFound.BrokerCluster	服务的集群不存在。
MissingParameter.NeedMoreParams	必要参数没有传递。
FailedOperation.ReceiveError	接收消息出错。
ResourceNotFound	资源不存在。
FailedOperation.DeleteBindVpc	删除vpc绑定关系失败。
FailedOperation.GetEnvironmentAttributesFailed	获取环境属性失败。
ResourcesSoldOut	资源售罄。
InternalError.Retry	重试可以成功。
UnknownParameter	未知参数错误。
FailedOperation.CreateSubscription	创建订阅关系失败。
FailedOperation.UpdateTopic	主题更新失败。