

# 消息队列 CMQ 版

## 产品文档



腾讯云TCE

目录

消息队列 CMQ 版 ..... 3

- 用户指南 ..... 3
  - 产品介绍 ..... 3
    - 产品概述 ..... 3
    - 产品功能 ..... 4
    - 产品优势 ..... 5
    - 应用场景 ..... 6
    - 使用限制 ..... 10
    - 基本概念 ..... 12
  - 快速入门 ..... 14
    - 队列模型快速入门 ..... 14
    - 主题模型快速入门 ..... 17
  - 开发指南 ..... 20
    - HTTP Endpoint 订阅 ..... 20
  - 操作指南 ..... 23
    - 主题订阅 ..... 23
      - 主题管理 ..... 23
      - 订阅管理 ..... 25
      - 查看监控 ..... 28
      - 标签键匹配功能说明 ..... 30
      - 路由键匹配功能说明 ..... 33
    - 队列服务 ..... 35
      - 队列管理 ..... 35
      - 查看队列详情 ..... 37
      - 查看监控 ..... 39
      - 消息回溯 ..... 41
      - 死信队列 ..... 45
    - 最佳实践 ..... 47
      - 消息去重 ..... 47
      - 选择 Push 还是 Pull ..... 50
    - 常见问题 ..... 52
  - 开发者指南 ..... 55
    - SDK 文档 ..... 55
      - 概述 ..... 55
      - TCP SDK ..... 56
      - HTTP SDK ..... 60
      - SDK 下载 ..... 65
    - 版本说明书 ..... 66
      - 版本说明书 ..... 66

# 用户指南

## 产品简介

## 产品概述

消息队列 CMQ 版（TDMQ for CMQ，简称 TDMQ CMQ 版）是一种分布式消息队列服务，它具有可靠的、基于消息的异步通信机制，能够将分布式部署的不同应用（或同一应用的不同组件）中的信息传递，存储在可靠有效的消息队列中，防止消息丢失。TDMQ CMQ 版支持多进程同时读写，收发消息互不干扰，无需各应用或组件始终处于运行状态。

TDMQ CMQ 版支持队列模型和主题模型，可用于各类异步通知、远程调用和主题消息分发等场景，常用于订单处理、耗时时间长的事件回调、各运营系统的日志流水等实际业务，同时支持百万级消息堆积数量，保证消息不丢失。

# 产品功能

## 异步通信协议

消息的发送者将消息发送到消息队列后可以立即返回，不用等待接收者的响应。消息会被保存在队列中，直到被接收者取出。消息的发送与处理是完全异步的。

## 提高可靠性

传统模式下消息可能因为长时间等待而导致请求失败。消息队列模式下，如果发送消息时接收者不可用，消息队列会保留消息直到成功传递它。

## 进程解耦

消息队列帮助减少两个进程间的耦合度。只要消息格式不变，即使接收者的接口、位置或者配置改变，也不会给发送者带来任何改变。并且，消息发送者无需知道消息接收者是谁，使得系统设计更清晰；相反的，进程间使用远程过程调用（RPC）或者 socket 连接，当一方接口、IP 或端口改变了，另一方则必须修改请求配置。

## 消息路由

发送者无需与接收者建立直接连接，双方通过消息队列保证消息能够从发送者路由到接收者，甚至对于本来网络不易互通的两个服务，也可以提供消息路由。

## 多终端

用户系统的多个部分可以同时发送或接收消息，TDMQ CMQ 版通过消息状态来进行消息可用性的控制。

## 多样性

每个队列均可独立配置，并非所有队列都要完全相同。在不同业务场景下的队列可以进行个性化的配置，例如一个队列中消息处理时间较长，可以针对队列属性进行优化。



# 产品优势

相比传统开源 MQ 应用，消息队列 CMQ 版具有以下优势：

| 对比项  | 消息队列 CMQ 版                                                                                                                                                                                                                     | 传统开源 MQ 应用                                                                                                                                                                                          |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 高性能  | 兼顾性能与可靠性，单 TDMQ CMQ 版实例 QPS 达到5000                                                                                                                                                                                             | 数据可靠性与性能无法兼顾                                                                                                                                                                                        |
| 高扩展性 | <ul style="list-style-type: none"><li>- 队列数量及队列存储容量可扩展性强</li><li>- 底层系统根据业务规模，自动弹性伸缩，上层业务无感知</li><li>- 高效支持亿级消息收发、推送、堆积，容量不设上限</li></ul>                                                                                       | 队列数量和消息堆积数量有限<br>每个 IDC 机房必须重新部署、购买设备，非常繁琐                                                                                                                                                          |
| 高可靠性 | <ul style="list-style-type: none"><li>- 消息服务每条消息在返回给用户写成功之时就确保数据已被复制3份写到不同物理机上，并且后台数据复制机制能够保证任何一台物理机故障时其上的数据能够快速的做迁移，时刻保证用户数据3份 copy 可用，可靠性达99.999999%</li><li>- 引入改良后的 Raft 一致性算法，保证数据强一致性</li><li>- 业务可用性承诺：99.95%</li></ul> | <ul style="list-style-type: none"><li>- 数据单机或简单主从结构，存在数据单点问题，一旦丢失不可回溯</li><li>- 开源的 replica 算法，在集群新增、删除服务器节点时，会引发全局的数据重新均衡，引起可用性急剧下降</li><li>- 如 Kafka 使用异步刷盘方式，异步 Replication，无法保证数据强一致性</li></ul> |
| 业务安全 | <ul style="list-style-type: none"><li>- 多维度的安全防护和通过接入层的缓存和负载均衡来应对 DDoS 攻击</li><li>- 每个消息服务提供单独命名空间，客户间数据严格隔离</li><li>- 支持 HTTP 访问</li><li>- 支持跨地域的安全消息服务</li></ul>                                                             | <ul style="list-style-type: none"><li>- 安全防护功能有限</li><li>- 考虑到公网的网络威胁，经常无法提供跨地域、跨 IDC 的公网服务</li></ul>                                                                                               |

# 应用场景

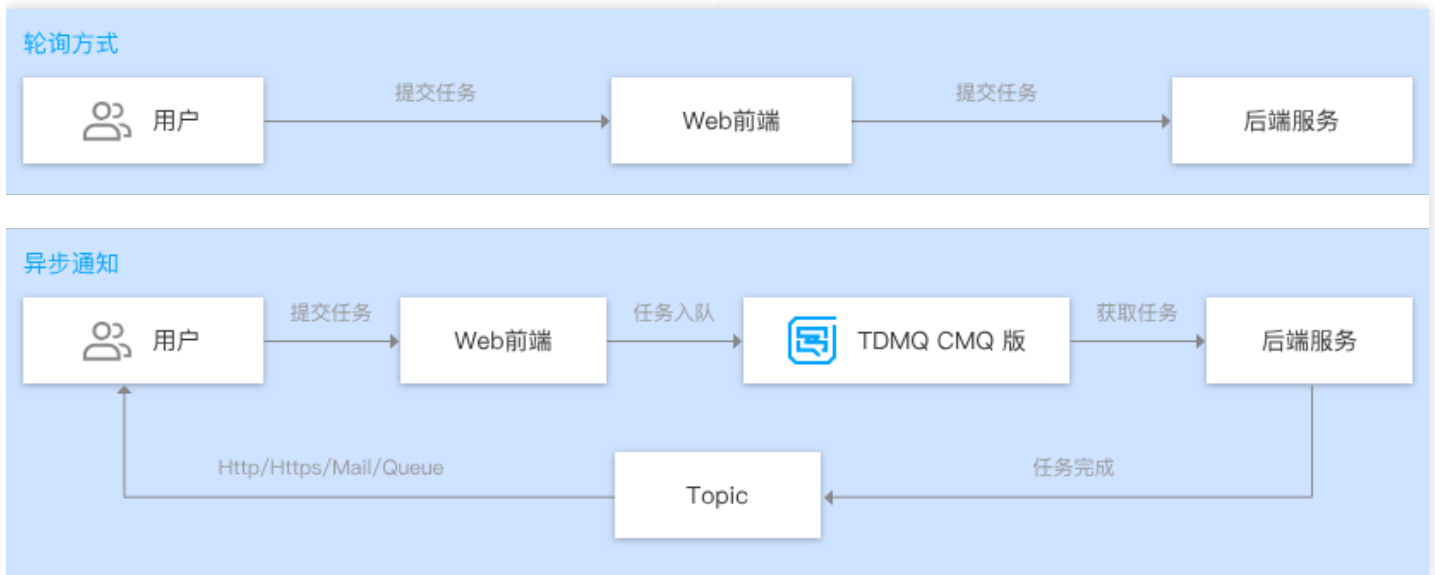
## 一对多生产

系统组件之间或者应用之间通信较多，需要组件或者应用自身维护彼此的网络连接，而且通信的内容不仅一种，实现难度和维护成本较大。使用 TDMQ CMQ 版可以实现生产者向多个订阅者，同时异步投递消息的能力，同时客户端可以通过消息过滤消费其中的某类消息。



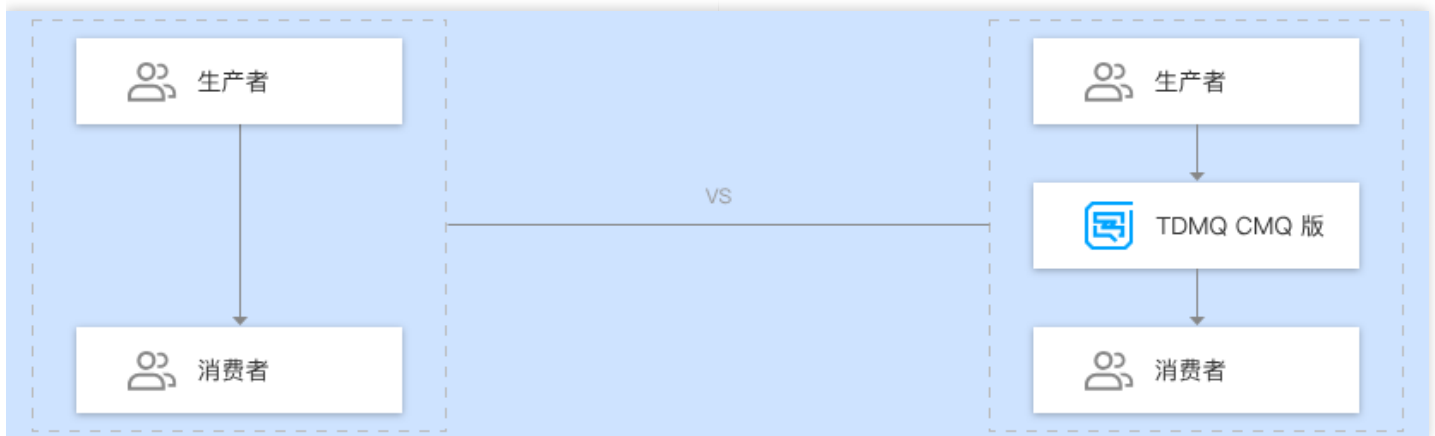
## 异步通知

发送消息时，接收者由于断电、宕机或 CPU 负载过高等原因不可用，消息无法可靠传递。使用 TDMQ CMQ 版，消息会被持久化保存在队列中，直到接收者可用时消息被成功消费。



## 系统解耦

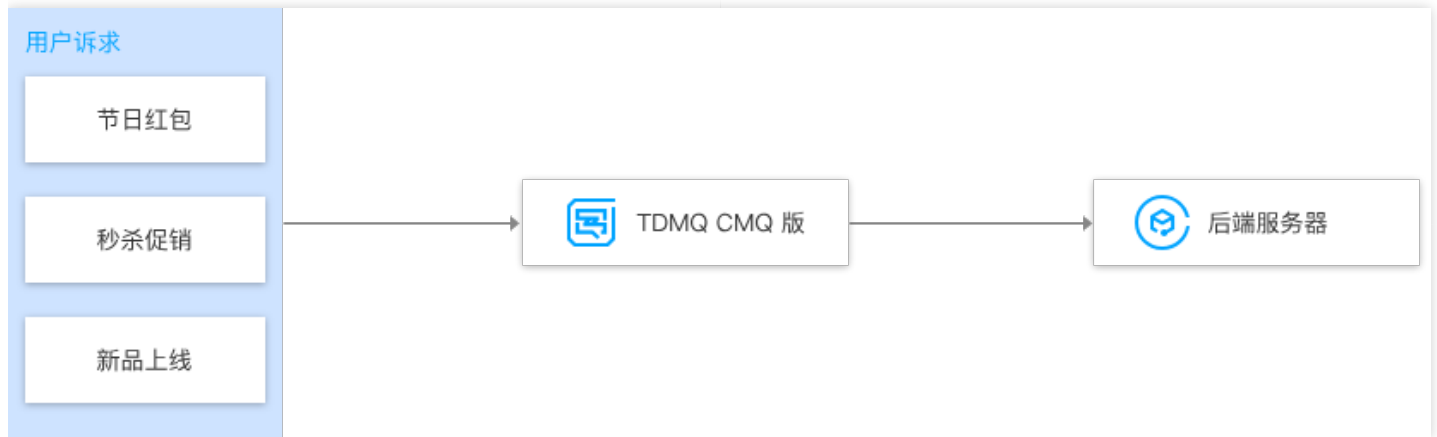
公司业务收到用户提交的内容，将部分数据存储在自身的系统中，并将处理后的数据转发给其他业务应用（如数据分析系统、数据存储系统等），在这种系统组件之间或者应用之间耦合较紧，尤其对依赖的组件可控性较弱的情况下，使用 TDMQ CMQ 版，发送者和接收者对彼此信息无感知，耦合度大大降低。



## 削峰填谷

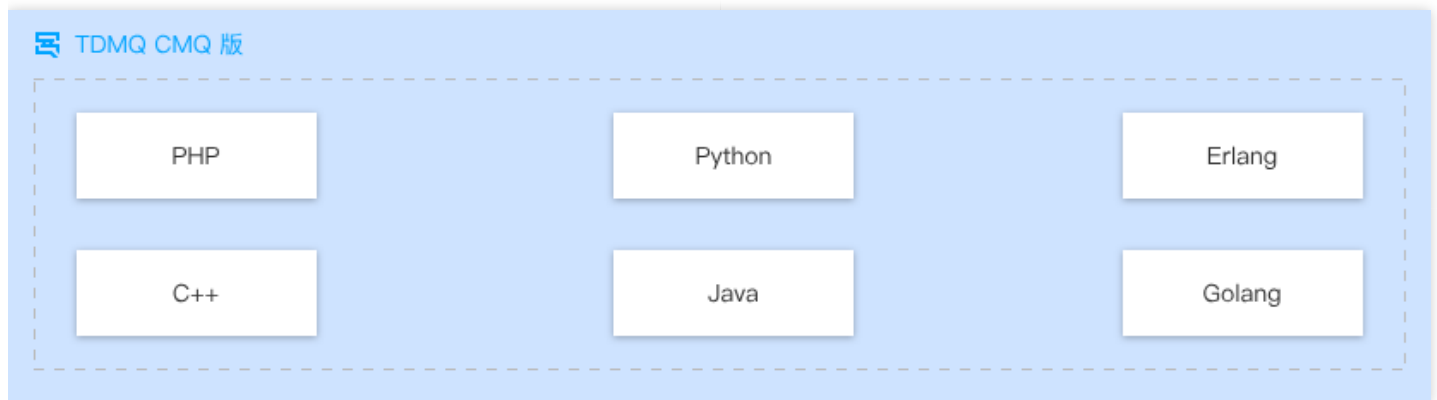
在秒杀或团队抢购活动中，由于用户请求量较大，往往都会带来临时性的流量洪峰，这对后端的各个应用系统考验是十分巨大的，如果直接采用扩容方式应对又会带来一定的资源浪费。TDMQ CMQ 版此时便可以承担一个缓冲器的角

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。



## 屏蔽平台差异

随着电商系统架构逐渐成长，假如订单系统（order\_module）采用 Java 架构，库存系统（inventory\_module）采用 Erlang 架构，而发货系统使用的是 Python 架构，开发人员需要长期维护一些冗余的代码来支持各模块之间通信。引入 TDMQ CMQ 版后，可以屏蔽不同平台，不同编程语言之间的差异。



## 跨用户数据交换

两个服务在网络不能互通或者应用的路由信息（例如 IP 和端口）不确定的情况下需要通信。例如：两个云上的服务在不知道对方地址的情况下需要进行通信，引入 TDMQ CMQ 版后，可以通过约定相同的队列名，一个向队列中发消息，一个从队列中收消息来进行数据交换。



# 使用限制

本文列举了 TDMQ CMQ 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。

## 队列

| 限制类型        | 限制说明                     |
|-------------|--------------------------|
| 单主账号下最大队列数量 | 10000个                   |
| 消息生命周期      | 1分钟 - 15天                |
| 消息接收长轮询等待时间 | 0 - 30秒                  |
| 取出消息隐藏时长    | 1秒 - 12小时                |
| 消息最大长度      | 1MB                      |
| 堆积消息数量上限    | 单个队列的堆积消息上限为1亿条，最小值为1百万条 |
| 生产 QPS 限制   | 5000                     |
| 消费 QPS 限制   | 5000                     |
| 流量限制        | 400Mbps                  |

## 主题

| 限制类型        | 限制说明             |
|-------------|------------------|
| 单主账号下最大主题数量 | 10000个           |
| 消息生命周期      | 默认设置为24小时，暂不允许修改 |
| 消息最大长度      | 1MB              |
| 生产 QPS 限制   | 5000             |
| 消费 QPS 限制   | 5000             |
| 流量限制        | 400Mbps          |

# 订阅者

| 限制类型             | 限制说明  |
|------------------|-------|
| 单个 Topic 订阅者数量上限 | 1000个 |

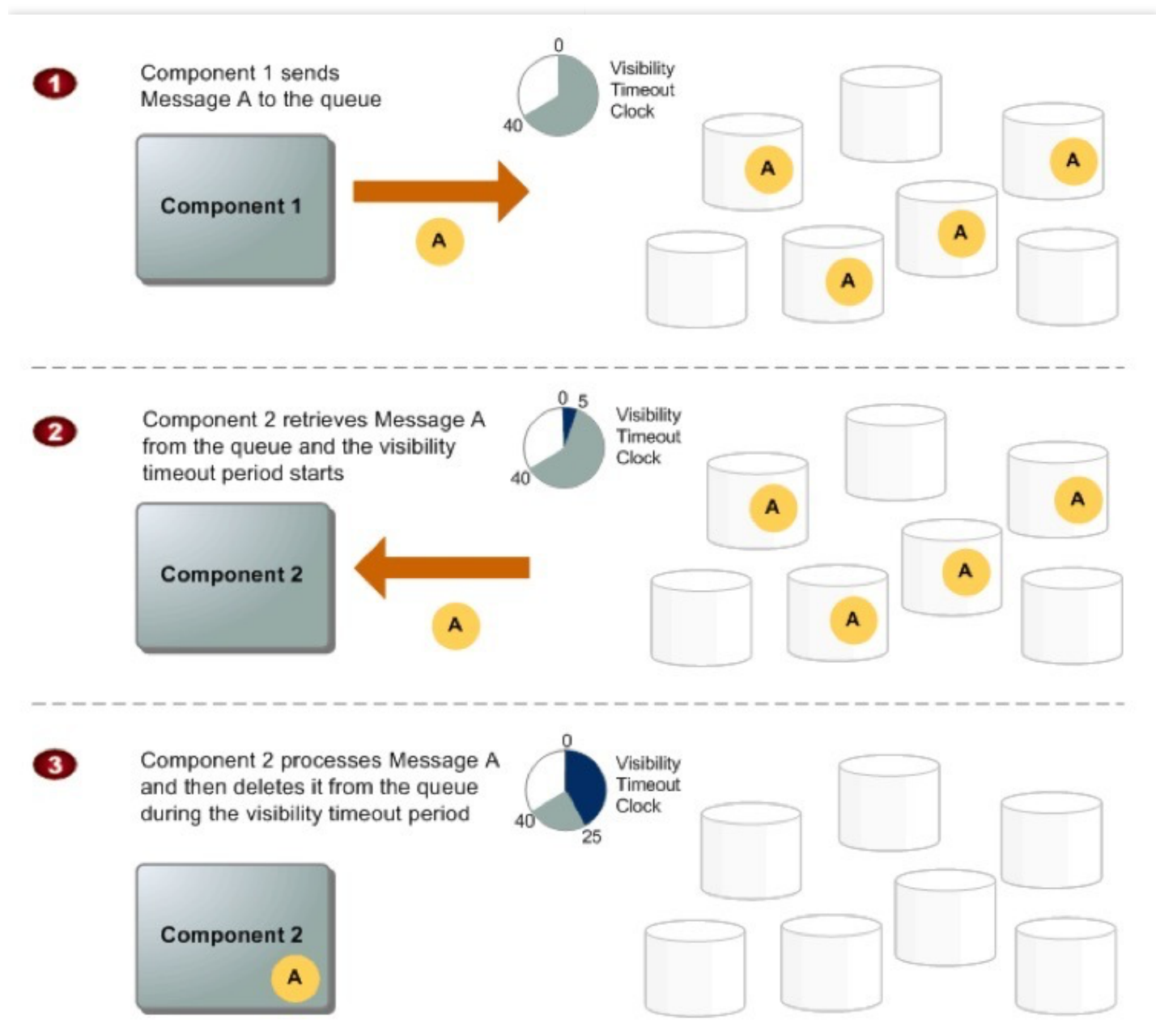
说明：

其中，如果对生产 QPS 限制、消费 QPS 限制和流量限制指标有更高的需求，支持咨询技术支持申请提高其上限

# 基本概念

## 消息生命周期

普通消息被发送到普通消息队列时，初始状态为 Active，当其被取走后在 VisibilityTimeout 的时间内状态为 Inactive，若超过 VisibilityTimeout 时间后消息还未被删除，消息会重新变成 Active 状态；如果在 VisibilityTimeout 时间内被删除，消息状态将变为 Deleted。消息的最长存活时间由创建队列时指定的 MessageRetentionPeriod 属性值决定，超过此时间后消息状态变成 Expired 并将被回收。消费者只能取到处于 Active 状态的消息。这保证了同一条消息不会同时被多次消费，但可被顺序性地多次消费。





- Component 1 将 Message A 发送到一个队列，该消息在 TDMQ CMQ 版服务器间提供多份冗余。
- 当 Component 2 准备好处理消息时，就从队列检索消息，然后 Message A 返回。在 Message A 处理期间，它仍然停留在队列中，在取出消息隐藏时长阶段，其他业务不可获取 Message A。
- Component 2 可从队列删除 Message A，以避免一旦取出消息隐藏时长过期后该消息被再次接受并处理；也可以不删除 Message A，该消息可以被其他业务多次消费。

## 队列和消息标识符

### 队列名称

创建新队列时，用户必须提供在此地域范围内唯一的队列名称，不同地域间队列名称可以重复。消息队列 CMQ 版使用地域和队列名称唯一标识一个队列，每当用户要对队列执行操作时，都需要提供这两个参数。

### 消息 ID

每条消息都会收到一个由系统分配的消息 ID，该 ID 可由 SendMessage 接口请求中返回给用户。此标识符用于识别消息。需要注意的是，删除消息时用户需要消息的接收句柄，而不是消息 ID。消息 ID 有形如 Msg-XXXXXXXX 的样式。

### 接收句柄

每当收到来自队列的消息时，用户都会收到该消息的接收句柄。消息句柄始终与接收消息的操作相关联，与消息本身无关。要删除消息或更改消息属性，用户则必须提供接收句柄，而不是消息 ID。这意味着，必须始终先接收消息，然后才能删除/更改它。

#### 说明：

如果多次接收某条消息，则每次接收该消息时，用户都会获得不同的接收句柄。在请求删除该消息时，您必须提供最近收到的接收句柄，否则可能无法删除该消息。

# 快速入门

## 队列模型快速入门

### 操作场景

本文为您介绍从零开始创建一个队列服务并使用 Java SDK 进行收发消息测试的方法，帮助您快速了解客户端接入 TDMQ CMQ 版所需的基本操作。

### 操作步骤

#### 步骤1：创建队列服务

- 1. 登录 TDMQ CMQ 版控制台。
- 2. 在左侧导航栏选择队列服务，选择地域后，单击新建，配置队列服务相关属性值。

| 属性          | 说明                                                                             | 取值                                                                                           |
|-------------|--------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| 队列名称        | QueueName，为队列的名称。                                                              | 作为资源的唯一标识，调用 API 接口进行操作时，以 Queue name 为准，创建成功后无法修改。不区分大小写，相同字母即会判定为同名，并不能以 -retry 和 -dlq 结尾。 |
| 消息最长未确认时间   | 如果消费客户端在获取到消息后超过此时间仍未进行消息的确认，则服务端会自动确认该消息。                                     | 范围在30秒到12小时                                                                                  |
| 消息接收长轮询等待时间 | PollingWaitSeconds，长轮询等待时，一个消息消费请求只会在取到有效消息或长轮询超时时才返回响应，类似于 Ajax 请求的长轮询。       | 范围在0秒到30秒，推荐设置为3秒，设置过高可能造成消息重复的概率提升。                                                         |
| 取出消息        | 该项为队列的 VisibilityTimeout 属性，每条 Message 都有个默认的VisibilityTimeout，Worker 在接收到消息后， | 范围在1秒到12小时                                                                                   |

|          |                                                                                                  |                                                         |
|----------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| 隐藏<br>时长 | timeout 就开始计时了。如果 Worker 在 timeout 时间内没能处理完 Message，那么消息就有可能被其他 Worker 接收到并处理。                   |                                                         |
| 死信<br>队列 | 死信队列用于处理无法被正常消费的消息。达到最大重试次数后，若消费依然失败，则表明消费者在正常情况下无法正确地消费该消息，此时，MQ 不会立刻将消息丢弃，而是将其发送到该消费者对应的特殊队列中。 | -                                                       |
| 消息<br>回溯 | 若未开启“消息回溯”能力，则消费者已消费，且确认删除的消息，会立即删除，开启该功能时，须指定回溯的“可回溯周期”。                                        | “可回溯周期”的范围，必须小于等于消息的生命周期。建议将回溯周期与消息的生命周期设置为相同的值，便于定位问题。 |

3. 单击提交，在队列服务列表可以看到创建好的队列服务。

## 步骤2：使用 SDK 收发消息

1. 下载 Demo 并解压。
2. 引入 CMQ 客户端相关依赖。

```
<!-- cmq sdk --><!-- cmq sdk -->
<dependency>
  <groupId>com.qcloud</groupId>
  <artifactId>cmq-http-client</artifactId>
  <version>1.0.7</version>
</dependency>

<!-- 云API sdk -->
<dependency>
  <groupId>com.tencentcloudapi</groupId>
  <artifactId>tencentcloud-sdk-java</artifactId>
  <version>3.1.423</version>
</dependency>
```

### 3. 发送消息

```
Account account = new Account(SERVER_ENDPOINT, SECRET_ID, SECRET_KEY);
Queue queue = account.getQueue(queueName);
String msg = "hello client, this is a message. Time:" + new Date();
CmqResponse response = queue.send(msg);
```

| 参数 | 说明 |
|----|----|
|----|----|

|                          |                                                                                                                  |
|--------------------------|------------------------------------------------------------------------------------------------------------------|
| SERVER_ENDPOINT          | API 调用地址，在TDMQ CMQ 版控制台的队列服务 > API 请求地址处复制。                                                                      |
| SECRET_ID、<br>SECRET_KEY | <p>云 API 密钥，在用户与权限列表页面复制。</p>  |
| queueName                | 队列名称，在 TDMQ CMQ 版控制台的队列服务列表页面获取。                                                                                 |

#### 4. 消费消息。

```
Account account = new Account(SERVER_ENDPOINT, SECRET_ID, SECRET_KEY);
Queue queue = account.getQueue(queueName);
Message message = queue.receiveMessage();
// 消费成功，删除消息。未删除的消息，将在一定时间后可重新投递
queue.deleteMessage(message.receiptHandle);
```

|                          |                                                                                                                    |
|--------------------------|--------------------------------------------------------------------------------------------------------------------|
| 参数                       | 说明                                                                                                                 |
| SERVER_ENDPOINT          | API 调用地址，在TDMQ CMQ 版控制台的队列服务 > API 请求地址处复制。                                                                        |
| SECRET_ID、<br>SECRET_KEY | <p>云 API 密钥，在用户与权限列表页面复制。</p>  |
| queueName                | 队列名称，在 TDMQ CMQ 版控制台的队列服务列表页面获取。                                                                                   |

# 主题模型快速入门

## 操作场景

本文为您介绍从零开始创建一个主题并使用 Java SDK 进行收发消息测试的方法，帮助您快速了解客户端接入 TDMQ CMQ 版所需的基本操作。

## 操作步骤

### 步骤1：创建主题

1. 登录 TDMQ CMQ 版 控制台。
2. 在左侧导航栏选择主题订阅，选择好地域，单击新建，填写主题名称。
  - 主题名称：以字母起始，只能包含字母、数字、“-”及“\_”，最大64字符，创建后不能修改，不区分大小写。
  - 消息堆积：未触发推送到订阅者，或订阅者接收失败的消息，暂时堆积到主题中。
  - 消息过滤类型：
    - 标签：CMQ 提供生产、订阅的消息标签匹配能力，可用于消息过滤。
    - 路由匹配：Binding key、Routing key 是组合使用的，完全兼容 rabbitmq topic 匹配模式。发消息时配的 Routing key 是客户端发消息带的。创建订阅关系时配的 Binding key 是 topic 和 订阅者 的绑定关系。
3. 单击提交，在主题订阅列表可以看到创建好的主题。

### 步骤2：创建订阅

主题发布消息有一个前提，即需要有订阅者订阅主题，如果没有订阅者存在，那么主题中的消息不会被投递，此时发布消息这一操作就失去了意义。

1. 在 主题订阅 页面，单击刚刚创建的主题的“ID”，进入主题详情页面。
2. 选择页面上方的订阅者页签，单击新建，填写订阅者相关信息。
  - 订阅者类型
    - Queue 队列服务：订阅者可以填写一个 Queue，使用队列来接收发布的消息。
    - URL地址：订阅者也可以不与 Queue 结合，自己来处理消息。
  - 消息推送格式：支持消息原文或者 JSON 格式。
  - Binding key：Binding key、Routing key是组合使用的，完全兼容rabbitmq topic匹配模式。发消息时配的Routing key是客户端发消息带的。创建订阅关系时配的 Binding key 是 topic 和 订阅者 的绑定关系。
  - 重试策略：主题发布消息之后，会自动将消息推送给订阅，当推送失败时，有两种重试策略：
    - 退避重试：重试3次，间隔时间为10s - 20s之间的一个随机值，超过3次后，该条消息对于该订阅

者丢弃，不会再重试。

- 衰退指数重试：重试176次，总计重试时间为1天，间隔时间依次为： $2^0$ ， $2^1$ ，...，512，512，...，512秒。默认为衰退指数重试策略。

3. 单击提交，在订阅者列表可以看到刚刚创建好的订阅者。

### 步骤3：使用 SDK 收发消息

1. 下载 Demo 并解压。

2. 引入cmq客户端相关依赖。

```
<!-- cmq sdk -->
<dependency>
  <groupId>com.qcloud</groupId>
  <artifactId>cmq-http-client</artifactId>
  <version>1.0.7</version>
</dependency>

<!-- 云API sdk -->
<dependency>
  <groupId>com.tencentcloudapi</groupId>
  <artifactId>tencentcloud-sdk-java</artifactId>
  <version>3.1.423</version>
</dependency>
```

3. 创建Topic对象。

```
Account account = new Account(SERVER_ENDPOINT, SECRET_ID, SECRET_KEY);
Topic topic = account.getTopic(topicName);
```

| 参数                       | 说明                                           |
|--------------------------|----------------------------------------------|
| SERVER_ENDPOINT          | API 调用地址，在 TDMQ CMQ 版控制台的主题订阅 > API 请求地址处复制。 |
| SECRET_ID、<br>SECRET_KEY | 云 API 密钥，在用户与权限列表页面复制。                       |



topicName

主题订阅名称，在 TDMQ CMQ 版控制台 的主题订阅列表页面获取。

#### 4. 发送 TAG 类型消息。

```
String msg = "hello client, this is a message. tag=TAG1. Time:" + new Date();  
List<String> tags = Collections.singletonList("TAG1");  
String messageId = topic.publishMessage(msg, tags, null);
```

#### 5. 发送 route 消息。

```
String msg = "hello client, this is a message. route(abc) Time:" + new Date();  
String messageId = topic.publishMessage(msg, "abc");
```

#### 6. 消费消息，使用订阅者对应的queue进行消费。

```
Account account = new Account(SERVER_ENDPOINT, SECRET_ID, SECRET_KEY);  
Queue queue = account.getQueue(queueName);  
Message message = queue.receiveMessage();  
// 消费成功，删除消息。未删除的消息，将在一定时间后可重新投递  
queue.deleteMessage(message.receiptHandle);
```

说明：

以上是 CMQ 的生产和消费方式的简单介绍，更多操作可参见 [Demo](#)。

# 开发指南

## HTTP Endpoint 订阅

### 投递描述

CMQ 通过发送 POST 请求将主题消息推送到订阅的 HTTP Endpoint 端，消息格式支持两种：JSON 格式和 SIMPLIFIED 精简格式。

- JSON 格式：推送的 HTTP 请求 Body 包含消息的正文和消息的属性信息。Content-type 为 text/plain。
- SIMPLIFIED 格式：推送的 HTTP 请求 Body 即为消息正文，而 msgId 等信息会在 HTTP 请求 Header 中传递给订阅方。

订阅方的 HTTP 服务返回标准的 2xx 响应（如200），代表投递成功，否则为投递失败，并触发重试投递策略；超时不响应，CMQ 也会认为失败，同样触发重试投递策略，检测超时时间约为15秒。

### 投递的 HTTP 请求 Header

| 参数名称              | 描述                |
|-------------------|-------------------|
| x-cmq-request-id  | 此次推送消息的 requestId |
| x-cmq-message-id  | 此次推送消息的 msgId     |
| x-cmq-message-tag | 此次推送消息的消息标签       |

### 投递的 HTTP 请求 Body

在 JSON 格式下，HTTP Body 包含消息的正文和消息的属性信息。

| 参数名称             | 类型     | 描述               |
|------------------|--------|------------------|
| TopicOwner       | String | 被订阅的主题拥有者的 APPID |
| topicName        | String | 主题名称             |
| subscriptionName | String | 订阅名称             |
| msgId            | String | 消息 ID            |
| msgBody          | String | 消息正文             |



|             |     |         |
|-------------|-----|---------|
| publishTime | Int | 消息的发布时间 |
|-------------|-----|---------|

在 SIMPLIFIED 格式下，HTTP Body 即为发布者发布的信息正文。

## 投递的 HTTP 请求响应

订阅方 HTTP 服务正常处理投递消息，返回 2xx 响应；其他响应码或者超时不响应，当作错误，并触发重试投递策略。

## 请求示例

本示例中，订阅的 HTTP Endpoint 为<http://test.com/cgi>。JSON 格式：

```
POST /cgi HTTP/1.1
Host: test.com
Content-Length: 761
Content-Type: text/plain
User-Agent: Qcloud Notification Service Agent
x-cmq-request-id: 2394928734
x-cmq-message-id: 6942316962
x-cmq-message-tag: a, b

{"TopicOwner":100015036,"topicName":"MyTopic","subscriptionName":"mysubscription","msgId":"6942316962","msgBody":"test message","publishTime":11203432}
```

SIMPLIFIED 格式：

```
POST /cgi HTTP/1.1
Host: test.com
Content-Length: 123
Content-Type: text/plain
User-Agent: Qcloud Notification Service Agent
x-cmq-request-id: 2394928734
x-cmq-message-id: 6942316962
x-cmq-message-tag: a, b

test message
```

## 订阅消息示例

如下为订阅消息 Demo，投递消息均为 POST 方法，因此只需重写 do\_POST 方法即可。本示例会打印接收到的 HTTP POST 请求内容，并将 post data json 反序列化，实现遍历打印请求数据。

```
#!/usr/bin/python
from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler
import json
class TestHTTPHandle(BaseHTTPRequestHandler):
    def do_POST(self):
        content_len = int(self.headers.getheader('content-length',0))
        post_body = self.rfile.read(content_len)
        print "receive cmq topic publisher request:"
        print self.headers
        print post_body
        post_data = json.loads(post_body)
        for k,v in post_data.iteritems():
            print "key:%s value:%s" % (k,v)
        #response http status 200
        self.send_response(200)
        self.end_headers()
        self.wfile.write('ok')
def start_server(port):
    http_server = HTTPServer(('0.0.0.0', int(port)),TestHTTPHandle)
    http_server.serve_forever()
if __name__ == '__main__':
    start_server(80)
```

# 操作指南

## 主题订阅

## 主题管理

### 操作场景

主题模型类似设计模式中的发布订阅模式，Topic 相当于发布消息的单位，Topic 下面的订阅者相当于观察者模式。Topic 会把发布的信息主动推送到订阅者。

本文档介绍在使用 TDMQ CMQ 版时，如何创建或删除一个主题。

### 操作步骤

#### 创建主题

1. 登录 TDMQ 控制台，在左侧导航栏选择主题订阅，选择好地域。
2. 在队列列表页面，单击新建，根据页面提示填写信息。

新建主题

主题名 \*

请输入主题名

以字母起始，只能包含字母、数字、“-”及“\_”，最大64字符，创建后不能修改，不区分大小写

消息堆积 ⓘ

已启用

消息过滤类型

☒ 标签 ⓘ

☐ 路由匹配 ⓘ

提交

关闭

- 主题名称：以字母起始，只能包含字母、数字、“-”及“\_”，最大64字符，创建后不能修改，不区分大小写。
- 消息堆积：未触发推送到订阅者，或订阅者接收失败的消息，暂时堆积到主题中。
- 消息过滤类型：
  - 标签：CMQ 提供生产、订阅的消息标签匹配能力，可用于消息过滤。

- 路由匹配：Binding key、Routing key 是组合使用的，完全兼容 rabbitmq topic 匹配模式。发消息时配的 Routing key 是客户端发消息带的。创建订阅关系时配的 Binding key 是 topic 和 订阅者 的绑定关系。

## 删除主题

在 主题 列表，单击目标主题操作列的删除，可删除主题，删除该主题后将不再推送消息。

## 发送消息

1. 在主题列表页面，单击目标主题操作列的发送消息。
2. 填写消息内容和消息过滤标签，单击发送，向消息接收侧发送测试消息。

发送消息

主题名

at\_cmq\_tp404833

主题ID

cmqt-ppdbx

消息过滤类型

标签

消息内容

hello world

消息过滤标签

hello

发送

关闭

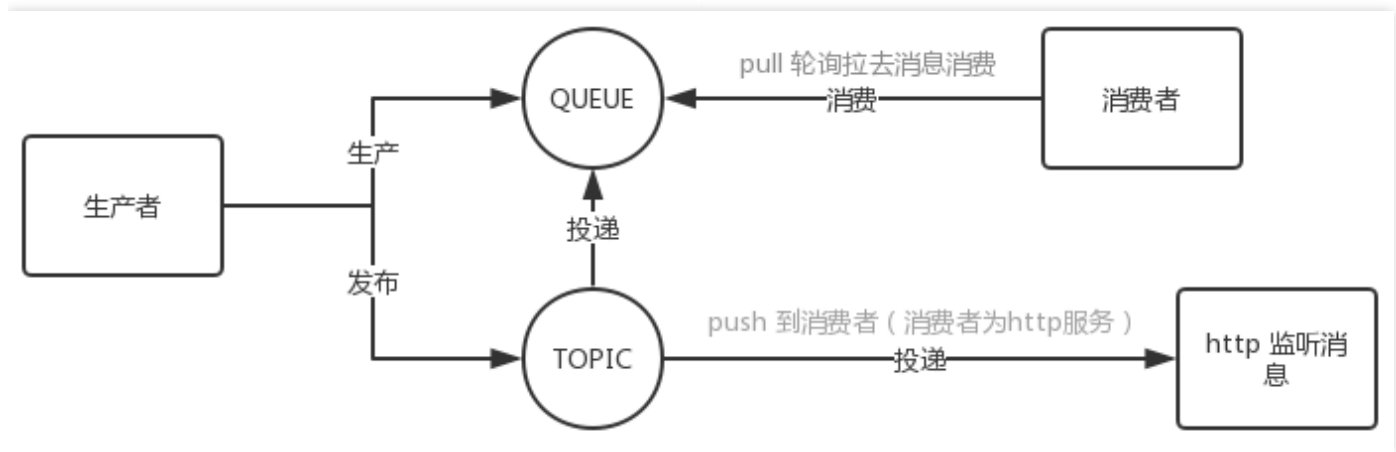
说明：

填写发送的内容，至少1Byte，最大长度受限于设置的队列消息最大长度属性。

# 订阅管理

## 操作场景

主题发布消息有一个前提，即需要有订阅者订阅主题，如果没有订阅者存在，那么主题中的消息不会被投递，此时发布消息这一操作就失去了意义。主题（Topic）向订阅者投递消息的模型如下：



Topic 向订阅者投递消息时，遵循以下原则：

- Topic 会尽最大努力将生产者 publish 的消息，投递（notification）到订阅者。
- 当投递重试多次仍失败后，消息会堆积在 Topic 中，等待下一次投递；若持续失败，将在消息最大生命周期（1 天）结束后，丢弃该消息。

本文档介绍在使用 TDMQ CMQ 版时，如何对一个主题下的订阅进行管理。

## 前提条件

已提前创建好对应主题。

## 操作步骤

### 创建订阅者

1. 在主题列表页面，单击目标主题的“ID”，进入主题详情页面。
2. 在页面上方选择订阅者页签，单击新建，填写订阅者相关信息。

新建订阅

订阅名

请输入订阅名

以字母起始，只能包含字母、数字、“-”及“\_”，最大64字符，创建后不能修改

订阅者属性

订阅者类型

☒ Queue队列服务

☐ HTTP URL地址

☐ HTTPS URL地址

订阅队列

请选择

消息推送格式

消息原文

消息过滤标签

点击输入框开始编辑

重试策略

☐ 退避重试

☒ 衰退指数重试

提交

关闭

- 订阅者类型
  - Queue 队列服务：订阅者可以选择一个 Queue，使用队列来接收发布的消息。
  - URL地址：订阅者也可以不与 Queue 结合，自己来处理消息。
- 添加订阅者标签：添加订阅者时，需增加 FilterTag。增加 FilterTag 后，该订阅者仅能收到带该 FilterTag 的消息，单个订阅者最多可添加5个 tag。只要其中某个 tag 能匹配 Topic 的过滤标签，订阅者即可收到该次 Topic 投递的消息，若消息不带任何标签，则该订阅者无法收到该类型消息。
  - 标签：详细规则参考 标签匹配功能说明。
  - 路由匹配：详细规则请参考 路由键匹配功能说明。
- 重试策略：主题发布消息之后，会自动将消息推送给订阅，当推送失败时，有两种重试策略：
  - 退避重试：重试3次，间隔时间为10 - 20s之间的一个随机值，超过3次后，该条消息对于该订阅者丢弃，不会再重试。
  - 衰退指数重试：重试176次，总计重试时间为1天，间隔时间依次为： $2^0$ ， $2^1$ ，...，512，512，...，512秒。默认为衰退指数重试策略。

3. 单击提交，在订阅者列表可以看到刚刚创建好的订阅者。

## 编辑订阅者

在 订阅者 列表页面，单击操作列的编辑，可修改订阅者属性。

说明：

仅支持修改订阅者的消息过滤标签和重试策略。

## 删除订阅者

在 订阅者 列表页面，单击操作列的删除，可删除订阅者。

# 查看监控

## 操作场景

本文介绍如何在 TDMQ CMQ 版控制台查看已创建主题的监控信息。

## 操作步骤

1. 登录 TDMQ CMQ 版控制台。
2. 在左侧导航栏选择主题订阅，选好地域后，单击目标主题的“ID”，进入主题详情页面。
3. 在主题详情页面，选择页面上方的监控页签。

### 监控指标说明

| 监控指标        | 说明                                   |
|-------------|--------------------------------------|
| 生产速率（条/秒）   | 在所选时间范围中，本 Topic 下所有生产者某一秒内生产消息的数量   |
| 消费速率（条/秒）   | 在所选时间范围中，本 Topic 下所有消费者某一秒内消费消息的数量   |
| 生产流量（字节/秒）  | 在所选时间范围内某一秒，本 Topic 下所有生产者生产的消息数据量大小 |
| 消费流量（字节/秒）  | 在所选时间范围内某一秒，本 Topic 下所有消费者消费的消息数据量大小 |
| 消息总大小（字节）   | 消息当前总数据量大小                           |
| 消息平均大小（字节）  | 消息当前平均数据量大小                          |
| 消息总数量（个）    | 消息当前总数量                              |
| API 调用次数（次） | 该指标用于计算API调用次数费用，并非实际调用次数            |



基本信息

订阅者

监控

实时

近24小时

近7天

选择日期

数据对比

时间粒度: 1分钟

刷新

① 注释: Max、Min和Avg数值统计为当前折线图内所有点的最大值、最小值和平均值

导出数据

|           |                              |           |           |           |                           |
|-----------|------------------------------|-----------|-----------|-----------|---------------------------|
| 生产速率条/秒①  | 无数据                          | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |
| 消费速率条/秒①  | 无数据                          | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |
| 生产流量字节/秒① | 无数据                          | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |
| 消费流量字节/秒① | 无数据                          | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |
| 消息总大小字节①  | 无数据                          | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |
| 消息平均大小字节① | 无数据                          | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |
| 消息总数量个    | 网络异常, 请 <a href="#">重新加载</a> | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |
| API调用次数次① | 无数据                          | Max:<br>- | Min:<br>- | Avg:<br>- | <div>🔍</div> <div>☰</div> |

# 标签键匹配功能说明

TDMQ CMQ 版的 Topic 模式支持订阅筛选标签 (tag) 功能，类似于 RabbitMQ 的 direct\_routing 模式，topic\_pattern 模式在下个迭代提供。由于筛选标签的使用策略较复杂，本文将结合不同场景进行说明。

## 场景说明

### 场景1

有4个订阅者 A、B、C、D，A 消息标签是 apple，B 消息标签是 xiaomi，C 的消息标签是 imac+xiaomi，D 没有设置任何标签。此时，有生产者 publish 到 Topic 100条消息，带的消息标签过滤为：apple、imac、iphone、macbook。此后 Topic 立即投递到 A、B、C、D。

场景分析：

| 订阅者 | 消息标签        | 消息接收说明                                   |
|-----|-------------|------------------------------------------|
| A   | apple       | 由于 apple 能匹配消息过滤标签中的 apple，则能正常收到100条消息。 |
| B   | xiaomi      | 不能收到任何消息。                                |
| C   | imac+xiaomi | 由于其中一个标签 imac 能匹配，则能正常收到100条消息。          |
| D   | -           | 可以收到任何消息。                                |

### 场景2

Topic 有且仅有4个订阅者 A、B、C、D，四个订阅者都没有设置消息 tag。此时，有生产者 publish 到 Topic 100条消息，带的消息标签过滤为：apple、imac、iphone、macbook。此后 Topic 立即投递到 A、B、C、D。

场景分析：订阅者 A、B、C、D 均没有 tag，则投递时不用匹配消息，A、B、C、D 都能收到这100条消息。

### 场景3

Topic 有且仅有1个订阅者 A，A 的订阅标签设置为 xiaomi。此时，有生产者 publish 到 Topic 100条消息，带的消息标签过滤为：apple、imac、iphone、macbook。此后 Topic 立即投递到 A。

场景分析：

- 对于订阅者 A，订阅标签不匹配，则 A 无法收到这100条消息。此时这100条消息立即丢弃，不会堆积在 TDMQ CMQ 版里。
- 生产者 publish 到 Topic 时，消息过滤 tag 只能在 publish 前设置一次，跟消息 ID 绑定，不可修改。

### 场景4

Topic 名为 test1，在12点01分，调用了订阅发布的 API，定义该次操作为“发布 test1”，发布后 Topic 投递到订阅者，给 A、B、C 三个订阅者投递了200条消息。假设结果为：A 有100条消息接收失败，B 有30条消息接收失败，C

的200条消息全部接收成功

场景分析：

- 消息堆积：在 A、B、C 三个订阅者中，假定消息投递失败的总集合为110条（A 有100条消息失败，B 有30条失败，C全部成功，三者之间的失败消息有交错的部分），只要存在任意1个订阅者与某条消息存在关联关系，该消息都不会被立即退订。
- 阻塞策略：以 A 为案例，Topic 投递给 A 200条消息，第101条失败时候，则后面的99条的投递会被阻塞。状态为有100条消息接收失败。
- 重试策略（退避重试）：Topic-test1 会每隔 N 秒给 A 重新投递消息，失败的100条里，按顺序，从第一条重新尝试，若失败三次，直接就丢弃，然后投递下一条消息，失败三次后再次丢弃，按顺序排列。
- 重试策略（衰退指数重试）：以 A 为案例，Topic 会并发投递100条消息（无法保证先后顺序）当订阅者接收的第一条消息失败后，从第一条重新尝试，若失败，继续阻塞。后续新增的投递消息，会继续阻塞。
- 消息生命周期不可延长：假定这110条堆积在 Topic-test1 里的消息，中间无论被重试多少次，生命周期都为1天，从生产者推送到 Topic 的时间点，作为起止时间点，到期后删除
- 重新推送：生产者不断往 Topic 中生产新的消息，客户在12点02分，又调用了订阅发布的 API，加上之前失败堆积的110条消息，假设目前 Topic 中共有110（失败堆积）+100（1分钟时间内新增的消息）=210条消息，再次投递。这时，A、B 的重试策略为衰退指数重试，处于“失联”状态，则210条消息会继续堆积。对于 C，只接收新增的100条消息。

说明：

每一条消息的 ID，作为 key，value 是关联的订阅者、代表每个订阅者消费成功与否。

## 场景5

Topic 名为 test2，在12点01分，调用了订阅发布的 API，定义该次操作为“发布 test2”，给 A、B、C 三个订阅者投递了200条消息。假设结果为：A、B、C 的200条消息全部接收成功。

场景分析：若 Topic-test2，有且仅有 A、B、C 三个订阅者，且都确定消费成功后，Topic 会立即删除200条消息。

## 规律总结

通过以上不同场景，标签匹配规律如下：

| 订阅者 | 订阅者有无 tag | 有无消息 tag | 消息接收说明                                                                    |
|-----|-----------|----------|---------------------------------------------------------------------------|
| A   | 有         | 无        | 订阅者不匹配，不能收到消息。                                                            |
| B   | 无         | 有        | 投递时消息不用匹配，订阅者都能收到消息。                                                      |
| C   | 有         | 有        | 两者匹配的，才能收到消息。支持 N:M 匹配，如消息有10个 tag，订阅者有4个 tag，其中有1个 tag 相互能匹配上，则订阅者能收到消息。 |

|   |   |   |                  |
|---|---|---|------------------|
| D | 无 | 无 | 投递后，所有订阅者都能收到消息。 |
|---|---|---|------------------|

# 路由键匹配功能说明

TDMQ CMQ 版的路由键匹配功能类似于 RabbitMQ 的 exchange queue，可以用于消息过滤，根据不同的条件使订阅者获取不同的消息。创建 Topic 时，可开启路由匹配键。

## 使用说明

Binding key、Routing key 是组合使用的，提供类似于 RabbitMQ 的消息过滤能力。发消息时配的 Routing key 是客户端发消息带的。创建订阅关系时配的 Binding key 是 Topic 和订阅者的绑定关系。

## 使用限制

- Binding key 的数量不超过5个。单个 Binding key 的长度  $\leq 64$  字节，用于表示发送消息的路由路径，最多含有15个“.”，即最多16个词组。
- Routing key 的数量由1个字符串组成。单个 Routing key 的长度  $\leq 64$  字节，用于表示发送消息的路由路径，最多含有15个“.”，即最多16个词组。

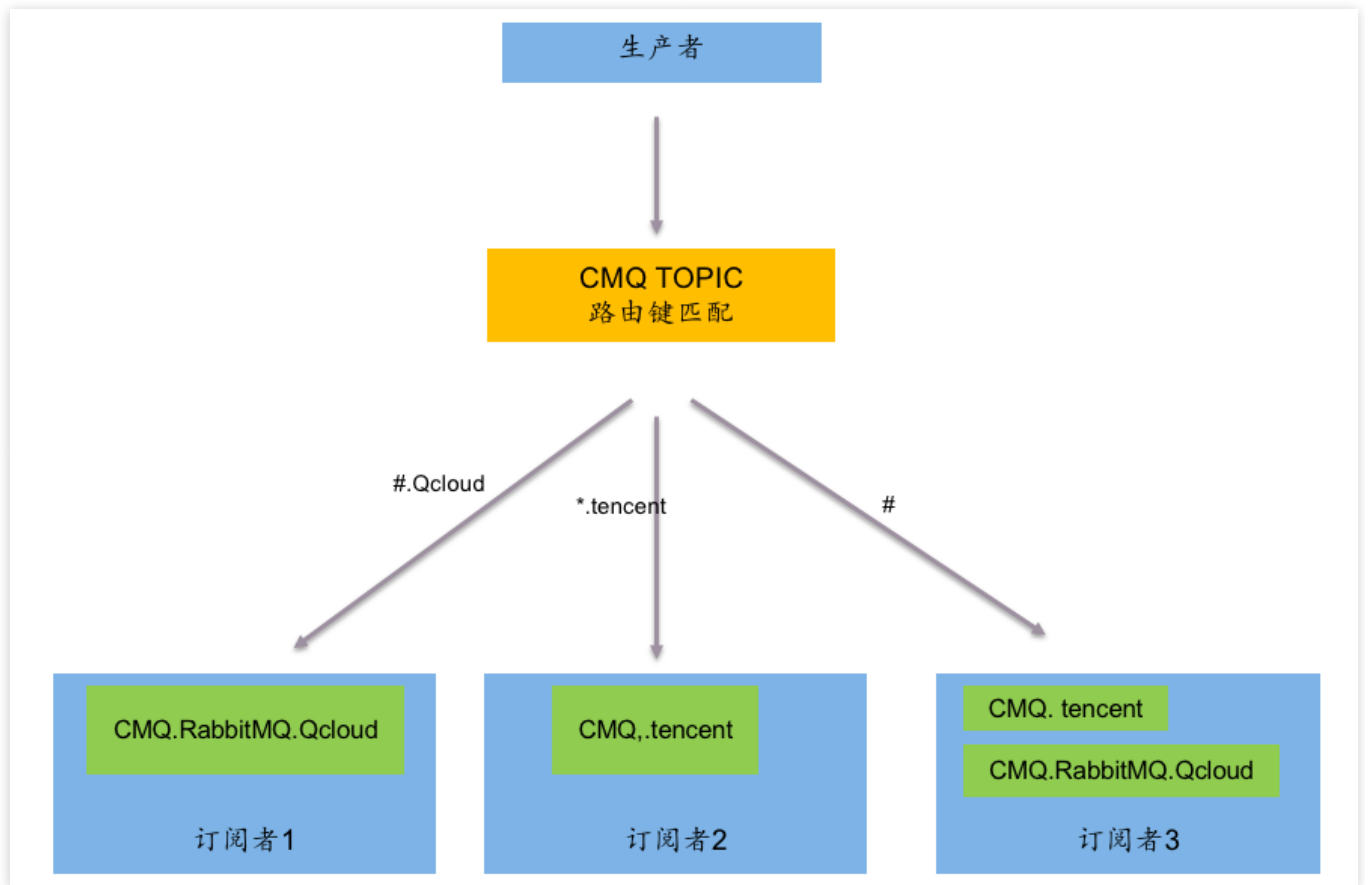
## 通配符说明

\*（星号），可以替代一个单词（一串连续的字母串），不能为空。

#（井号）：可以匹配零个或多个字符。

示例：

- 订阅者是『1.\*.0』，此时消息为『1.任意字符.0』，则订阅者都能收到消息。
- 订阅者是『1.#.0』，此时消息为『1.2.3.4.4.2.2.0』，『1.0』则订阅者都能收到消息（消息中间元素随意）。
- 订阅者是『#』，则所有消息订阅者都能收到。



# 队列服务

## 队列管理

### 操作场景

本文介绍如何在 TDMQ CMQ 版控制台创建队列服务和向消息接受侧发送消息的操作步骤。

### 进入队列列表

登录 TDMQ 控制台，在左侧导航栏选择队列服务，选择好地域。

### 新建队列

- 1. 在队列列表页面，单击新建，设置队列服务基本信息。

| 参数          | 说明                                                                                                                                                                                            |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 队列名称        | Queue name为队列服务的主键，不区分大小写，作为资源的唯一标识，调用API接口进行操作时，以Queue name为准，无法修改                                                                                                                           |
| 消息最长未确认时间   | 范围30秒到12小时，如果消费客户端在获取到消息后超过此时间仍未进行消息的确认，则服务端会自动确认该消息。<br>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小时后发送，消息最长未确认时间如果设置为1小时的话，则消息在1小时后会被删除，则延时消息会丢失。这种情况下要确保消息最长未确认时间要大于延时的时间，否则消息到期前会被删除。 |
| 消息接收长轮询等待时间 | 长轮询等待时，一个消息消费请求只会在取到有效消息或长轮询超时时才返回响应，类似于Ajax请求的长轮询；单位秒，推荐设置为3秒以下，有效值范围为0秒到30秒，设置过高可能造成消息重复的概率提升。                                                                                              |
| 取出消息隐藏时长    | 该项为队列的VisibilityTimeout属性，单位为秒，有效值范围1秒到43200秒，也即1秒到12小时。每条Message都有个默认的VisibilityTimeout，Worker在接收到消息后，timeout就开始计时了。如果Worker在timeout时间内没能处理完Message，那么消息就有可能被其他Worker接收到并处理。                 |

|      |                                                                                                                  |
|------|------------------------------------------------------------------------------------------------------------------|
| 死信队列 | 死信队列用于处理无法被正常消费的消息。达到最大重试次数后，若消费依然失败，则表明消费者在正常情况下无法正确地消费该消息，此时，MQ 不会立刻将消息丢弃，而是将其发送到该消费者对应的特殊队列中                  |
| 消息回溯 | 若未开启“消息回溯”能力，则消费者已消费，且确认删除的消息，会立即删除，开启该功能时，须指定回溯的“可回溯周期”，“可回溯周期”的范围，必须小于等于消息的生命周期。建议将回溯周期与消息的生命周期设置为相同的值，便于定位问题。 |

2. 单击确认，在队列服务列表可以看到创建好的队列服务。

## 发送消息

1. 在队列列表页面，单击目标队列操作列的发送消息。
2. 填写消息内容，单击发送，向消息接收侧发送测试消息。

发送消息

队列名

ABC

消息

hello world!

发送

关闭

说明：

填写发送的内容，至少1Byte，最大长度受限于设置的队列消息最大长度属性。



# 查看队列详情

## 操作场景

本文介绍如何在 TDMQ CMQ 版控制台查看已创建队列服务的详情信息。

## 操作步骤

1. 登录 TDMQ CMQ 版控制台。

2. 在队列列表页面，单击目标队列的“ID”，进入队列详情页面。在详情页中，您可以查询到：

◦ 队列概况

- 可见消息总数：消息被发送到队列后初始状态为可见（Active），该状态下消息可以被消费者消费。
- 不可见消息总数：消息被消费者取走后，转换为不可见（Inactive），如果在经过了 VisibilityTimeout 时间后仍未被消费，则会重新转换为可见（Active）。
- 延迟消息总数：如果无消费者在线，则延时消息总数会统计为0，消费者上线后该数据恢复为真实延时消息数量。
- 近24小时API调用次数

◦ 队列基本信息（名称，ID，地域，创建时间，修改时间）。

◦ 队列属性：单击右上角修改配置可修改队列属性。

| 参数          | 说明                                                                                                                                       |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------|
| 消息最大未确认时间   | 范围30秒到12小时，如果消费客户端在获取到消息后超过此时间仍未进行消息的确认，则服务端会自动确认该消息。<br>说明：如您使用延时消息，要保证消息的延时时间要小于消息最大未确认时间。例如延时到2小时后发送，消息最大未确认时间如果设置为1小时的话，则消息在1小时后会删除。 |
| 消息接收长轮询等待时间 | 长轮询等待时，一个消息消费请求只会在取到有效消息或长轮询超时时才返回响应，类似于Ajax请求的长轮询；单位秒，推荐设置为3秒以下，有效值范围为0秒到30秒，设置过高可能造成消息重复的概率提升。                                         |
| 取出消息隐藏时长    | 长轮询等待时，一个消息消费请求只会在取到有效消息或长轮询超时时才返回响应，类似于Ajax请求的长轮询；单位秒，推荐设置为3秒以下，有效值范围为0秒到30秒，设置过高可能造成消息重复的概率提升。                                         |

|        |                                                                                                  |
|--------|--------------------------------------------------------------------------------------------------|
| 消息最大长度 | 队列的MaxMsgSize属性， 限定允许发送到该队列的消息体的最大长度；单位为KB。                                                      |
| 死信队列   | 死信队列用于处理无法被正常消费的消息。达到最大重试次数后，若消费依然失败，则表明消费者在正常情况下无法正确地消费该消息，此时，MQ 不会立刻将消息丢弃，而是将其发送到该消费者对应的特殊队列中。 |

- 消息回溯：使用消息回溯功能您可在业务成功消费并删除消息后重新消费已删除的消息，详情参见 消息回溯。

队列概况

可见消息总数 ⓘ

0条

不可见消息总数 ⓘ

0条

延时消息总数 ⓘ

0条

近24小时API调用次数

0次

详细指标可以查看[监控](#)页签；对于堆积等重要指标，建议客户前往[配置告警](#)

基本信息

名称

at\_cmq741926

ID

cmqq-vje4nxbvoa5m

地域

重庆

创建时间

2023-10-27 10:50:31

修改时间

2023-10-27 10:50:31

队列属性

消息最大未确认时间 ⓘ

2小时

消息接收长轮询等待时间 ⓘ

0秒

取出消息隐藏时长 ⓘ

30秒

消息最大长度 ⓘ

1024KB

死信队列 ⓘ

未开启

修改配置

消息回溯

ⓘ 消息可回溯范围

1. 最远可回溯时间点 =  $current\_time - rewind\_seconds$ （消息回溯保留周期）

2. 最近可回溯时间点 =  $min\_msg\_time$ （队列最小未消费消息）

3. 最远时间点  $\leq start\_consume\_time \leq$  最近时间点

消息回溯 关闭

消息回溯

版权所有：TCloudFinanceZone

2026/8/26 10:11:28

第 38 页 共 66 页

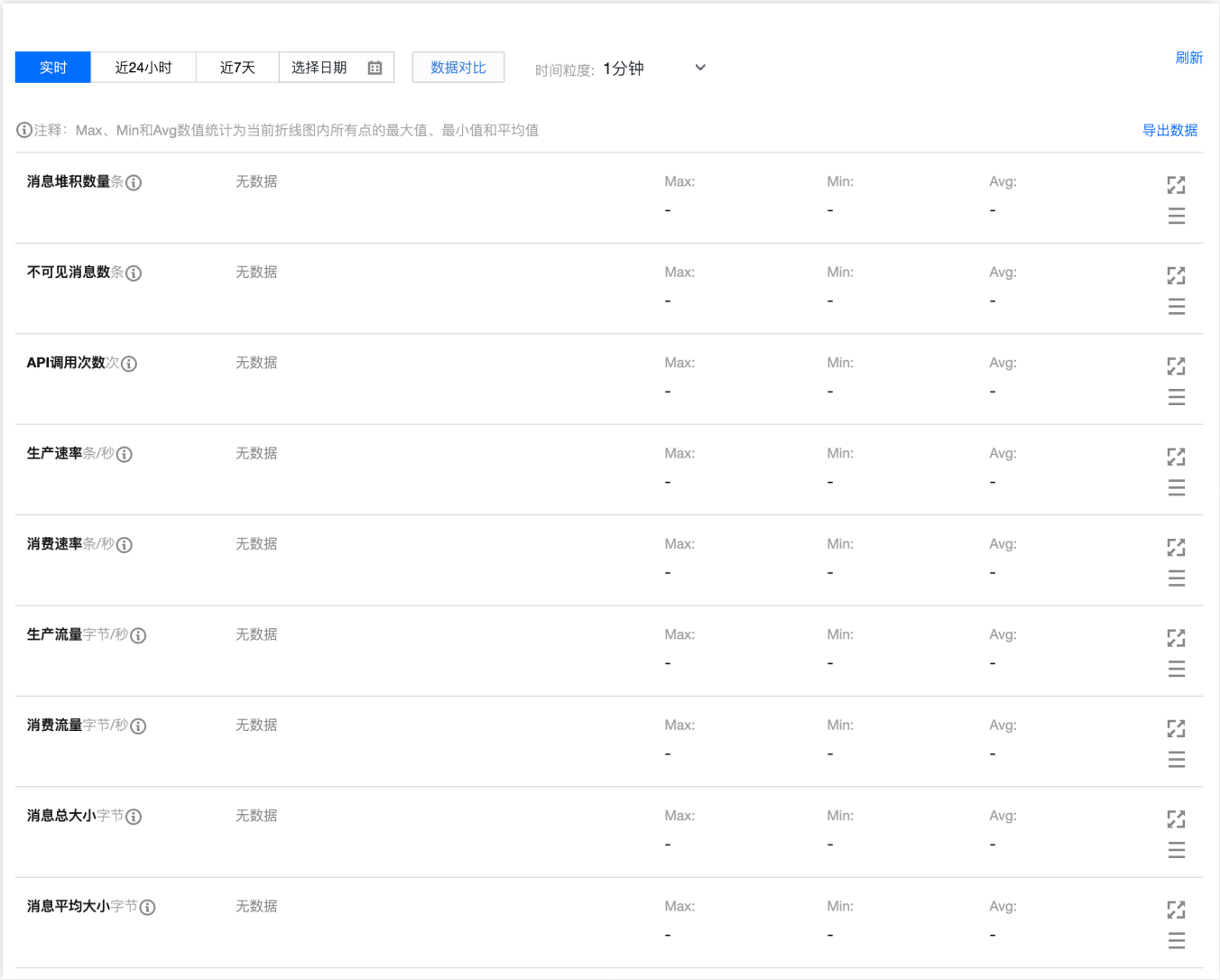
# 查看监控

## 操作场景

本文介绍如何在 TDMQ CMQ 版控制台查看已创建的队列的监控信息。

## 操作步骤

1. 登录 TDMQ CMQ 版控制台。
2. 在左侧导航栏选择队列服务，选好地域后，单击目标队列服务的“ID”，进入队列详情页面。
3. 在消息详情页面，选择页面上方的监控页签，选择好时间范围后可看到该队列的监控数据。



## 监控指标说明

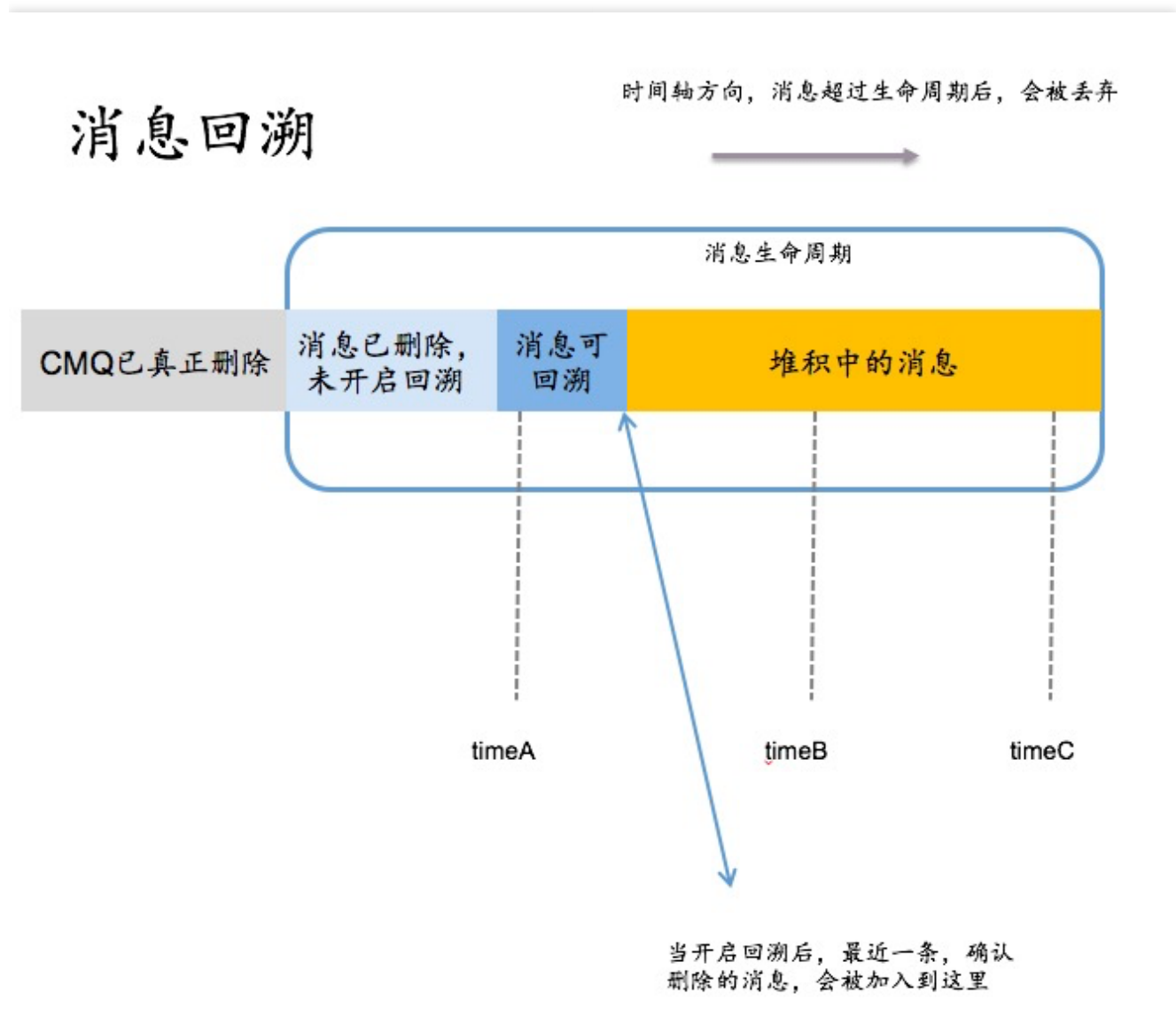
| 监控指标            | 说明                                                                              |
|-----------------|---------------------------------------------------------------------------------|
| 消息堆积数量          | Activemessages，所选时间范围内在该 Queue 中处于 Active 状态的消息总数，为近似值。                         |
| 不可见消息条数         | 消息被消费者取走后，转换为不可见（Inactive），如果在经过了 VisibilityTimeout 时间后仍未被消费，则会重新转换为可见（Active）。 |
| API 调用次数<br>（次） | 该指标用于计算API调用次数费用，并非实际调用次数                                                       |
| 生产速率（条/秒）       | 在所选时间范围中，该队列所有生产者某一秒内生产消息的数量。                                                   |
| 消费速率（条/秒）       | 在所选时间范围中，该队列所有消费者某一秒内消费消息的数量。                                                   |
| 生产流量（字节/秒）      | 在所选时间范围内某一秒，该队列所有生产者生产的消息数据量大小。                                                 |
| 消费流量（字节/秒）      | 在所选时间范围内某一秒，该队列所有消费者消费的消息数据量大小。                                                 |
| 消息总大小（字节）       | 消息当前总数据量大小。                                                                     |
| 消息平均大小<br>（字节）  | 消息当前平均数据量大小。                                                                    |

# 消息回溯

TDMQ CMQ 版提供类似于 Kafka 的消息回溯能力，使用消息回溯功能您可在业务成功消费并删除消息后重新消费已删除的消息。

本文为您介绍 TDMQ CMQ 版消息回溯功能的概念，应用场景和使用方法。

## 功能说明



如上图，消息的生命周期为蓝色框内的片段。开启消息回溯能力后，已被消费者消费且确认删除的消息会进入消息可回溯区域，TDMQ CMQ 版后端还会保存该信息。但消息超过 Queue 的消息生命周期时（假设为1天），达到生命周期后，该消息会自动删除，不可回溯。

## 产品逻辑

- 开启：若未开启消息回溯能力，则消费者已消费且确认删除的消息会立即删除。开启该功能时，须指定回溯的可回溯周期，可回溯周期的范围必须小于等于消息的生命周期。
- 里程碑：根据上一条策略，开启消息回溯后，随着消费者的不断消费及删除，可回溯的消息数量会不断增多。
- 关闭：关闭消息回溯后，消息可回溯区域的消息将被立即删除，且不可回溯。
- 队列属性：消息回溯是 Queue 的属性，可在创建时或在修改配置处进行设置。指定回溯（rewind）的时间点后，所有消费者都会从该时间点的消息往后消费。
- 计费：开启消息回溯能力后，可回溯部分的消息会产生一定的堆积费用，单位价格与消息堆积的费用共同计算。
- 指定回溯时间点：消费者发起回溯消费，需要指定 Queue Name 及具体的回溯时间。且从最远的时间点，往回回溯。时间为 key，不可逆向消费。如图所示，只能从 timeA 到 timeB/timeC 消费，不支持反向消费。
- 指定回溯时间范围：0 - 15天，控制台开启该能力后，删除的消息才可被回溯。建议关键应用，长期开启消息回溯能力。且消息回溯周期，与消息生命周期设成一致。
- 不可指定堆积中的消息回溯：若消息仍在堆积中，未被消费，则无法指定某一个具体的位置进行消费。

## 可回溯范围

消息可回溯范围以消息生产的时间为排序标准，与被删除的先后无关。

1. 最远可回溯时间点 =  $\text{current\_time} - \text{rewind\_seconds}$ （消息回溯保留周期）
2. 最近可回溯时间点 =  $\text{min\_msg\_time}$ （队列最小未消费消息）
3.  $\text{最远时间点} \leq \text{start\_consume\_time} \leq \text{最近时间点}$

## 应用场景

此功能便于核心金融业务做业务对账、业务系统重试等操作。

## 典型案例

这里结合一个场景来描述消息回溯的使用：

假设有 A/B 两个业务，正常的生产消费场景，A 生产消息投递到队列中，B 从队列中消费消息，A/B 这时已经实现互相解耦，双方互不关心。A 只需要生产消息投递即可，B 从队列中拿到消息，然后将消息从队列中删除，接着在本地消费消息。

如果出现了异常，例如 B 业务虽然进行了消费，但是在一段时间内消费情况都出现了异常。这时，已经删除的消息已经被删除，无法重新消费，会对业务造成影响，且需要暂停 B 业务，等开发运维人员修复 Bug 之后才能重新上线 B 业务。而且运维人员也无法实时监控 B 业务的情况，等到发现异常场景，可能已经过去一段时间。

为了防止这种情况出现，A 业务需要关心 B 业务的处理情况，需要对生产消息进行备份，确保 B 业务正常才能删除备份，保证现网正常。

这种情况下，您可以使用 消息回溯 的功能，开发人员对 B 业务进行修复，然后根据 B 业务消费正常的最近一个时间点，将消息回溯到该时间点。这时候，B 业务获取到的消息就会从指定的时间点开始，A 业务完全不用关心 B 业务的异常情况。这里 B 需要注意要对消费进行幂等处理。

## 开启消息回溯功能

在队列详情页面开启消息回溯功能。

### 消息回溯



#### 消息可回溯范围

1. 最远可回溯时间点 = `current_time - rewind_seconds`（消息回溯保留周期）
2. 最近可回溯时间点 = `min_msg_time`（队列最小未消费消息）
3. 最远时间点  $\leq$  `start_consume_time`  $\leq$  最近时间点

消息回溯

关闭

消息回溯

在客户端设置消息回溯相关参数。

```
endpoint="//TDMQ CMQ 版的域名  
secretId="//用户的 ID 和 key  
secretKey = "  
account = Account(endpoint,secretId,secretKey)  
queueName = 'QueueTest'  
my_queue = account.get_queue(queueName)  
queue_meta = QueueMeta()  
queue_meta.rewindSeconds = 43200 //消息允许回溯的时间，单位为秒  
my_queue.create(queue_meta)
```

## 使用消息回溯

```
my_queue.rewindQueue(1488718862) //指定本次消息回溯的时间点，为 Unix 时间戳
```



# 死信队列

本文为您介绍 TDMQ CMQ 版死信队列的基本概念、应用场景和使用方法。

## 功能介绍

死信队列（Dead-Letter-Queue，DLQ）用于处理无法被正常消费的消息。达到最大重试次数后，若消费依然失败，则表明消费者在正常情况下无法正确地消费该消息，此时 TDMQ CMQ 版不会立刻将消息丢弃，而是将其发送到该消费者对应的特殊队列中，这个特殊队列就是 DLQ。对新建的队列，或存量的队列，您都可以启用死信队列。

## 应用场景

- 定位问题：例如，某条消息被多次消费后却未被删除，一般是由于该消息未被正确消费，可能存在问题需要回溯定位。您可以设置最大接收次数，超额后该消息会被淘汰到指定的死信队列，便于后续问题发现。
- 优先级队列：例如，共享单车等 O2O 客户，对访问时延、实时性要求非常高。单车开锁逻辑中，当 TDMQ CMQ 版堆积了1亿条消息时，会优先处理最新生产的部分消息，老的消息淘汰到死信队列，在消费者有能力时再处理。老的消息（如用户扫码开车场景）在等待时已流失了用户，老的消息的价值会偏小，建议优先处理最新消息。

## 使用方法

在 TDMQ CMQ 版控制台 新建队列的时候，您可以开启死信队列功能。

## 死信队列设置

死信队列 



死信队列名称

请选择



死信策略 



最大接收次数



最大未消费时间

最大接收次数

20



次

范围在1到1000

- 死信队列名称：指定某队列为该队列的死信队列。
- 死信策略：触发死信消息的方式。
  - 最大接收次数：将消息发送到死信队列前允许接收该消息的最大次数，支持设定值为1 - 1000次。
  - 最大未消费时间：将消息发送到死信队列前达到了最大未消费时间，允许设置5分钟 - 12小时。

## 使用限制

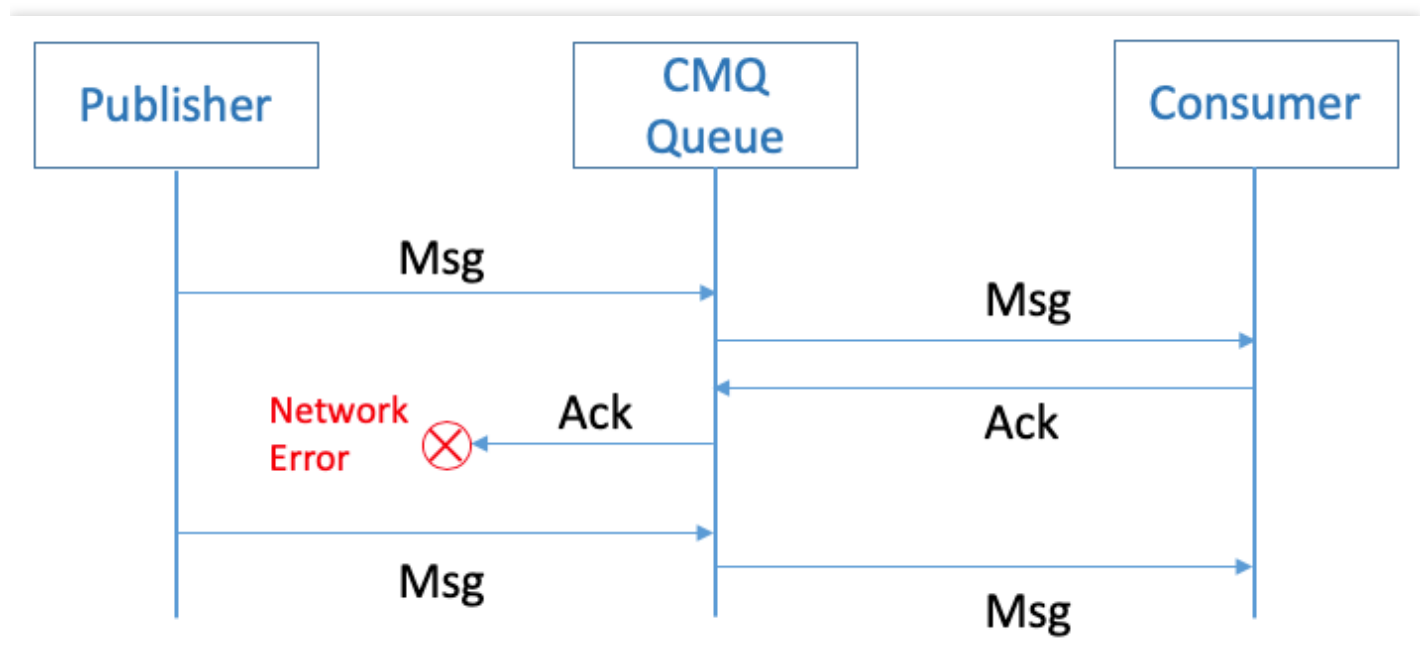
- 一个队列只能绑定一个死信队列。
- 绑定的死信队列必须是同地域、同账号下的队列。
- 如果死信队列已被其他队列绑定，则不能直接删除该死信队列。
- 开启事务消息的队列，不能成为其他队列的死信队列。
- 多个队列（最多6个）可以指定同一个队列作为死信队列，但死信队列本身不能再指定其他队列作为死信队列，避免嵌套。
- 可以在源队列（客户端生产、消费的数据队列）中解绑死信队列，解绑后无其他队列绑定，可删除死信队列。
- 一个已存在的普通队列在 T 时刻被指定为死信队列，若在 T 时刻之前队列中还存在未被消费的消息，需要将 T 时刻之前的消息全部消费完成后才可以触发死信策略。

# 最佳实践

## 消息去重

对于重复消息，最好的方法是消息可重入（消息重复消费对业务无影响）。做不到可重入时，需要在消费端去重。

### 重复消息出现的原因



网络异常、服务器宕机等原因都有可能导致消息丢失。TDMQ CMQ 版为了做到不丢消息、可靠交付，采用了消息生产、消费确认机制。

**\*\*生产消息确认：**\*\*生产者向 TDMQ CMQ 版发送消息后，等待 TDMQ CMQ 版回复确认；TDMQ CMQ 版将消息持久化到磁盘后，向生产者返回确认成功。否则在生产者请求超时、TDMQ CMQ 版返回失败等情况下，生产者需要向 TDMQ CMQ 版重发消息。

**\*\*消费者确认：**\*\*TDMQ CMQ 版向消费者交付消息后，将消息置为不可见；在消息不可见时间内，消费者使用句柄删除消息。如果消息未被删除，且不可见时间超时，消息将重新可见。

由于消息确认机制是“至少一次交付（at least once）”，在网络抖动、生产者/消费者异常等情况下，就会出现生产者重复生产、消费者重复消费的情况。

### 去重方案

要去重，先要识别重复消息。通常的做法是在生产消息时，业务方在消息体中插入去重 key，消费时通过该去重 key 来识别重复消息。去重 key 可以由 <生产者 IP + 线程 ID + 时间戳 + 时间内递增值> 组成的唯一值。

只有一个消费者时，您可以将消费过的去重 key 缓存（如 KV 等），然后每次消费时检查去重 key 是否已消费过。去重 key 缓存可以根据消息最大有效时间来淘汰。TDMQ CMQ 版提供了队列当前最小未消费消息的时间（min\_msg\_time），您可以使用该时间和业务生产消息最大重试时间来确定缓存淘汰时间。存在多个消费者时，去重 key 缓存就需要是分布式的。

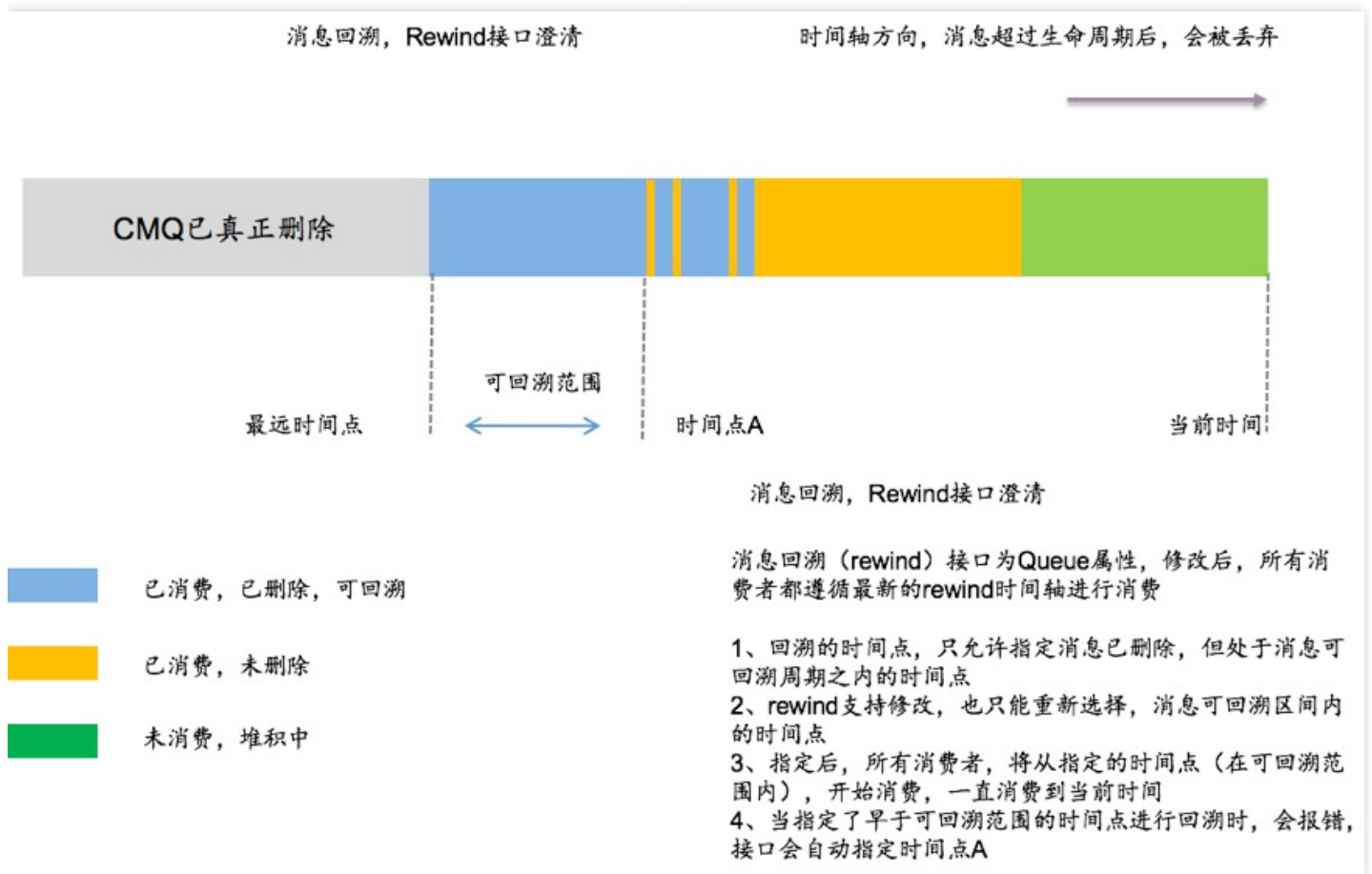
根据消息最大有效时间，计算 key 过期时间点：

$\text{current\_time} + \text{max\_retention\_time} + \text{max\_retry\_time} + \text{max\_network\_time}$ （当前时间）+（最大有效时间）+（最大重试时间）+（最大网络时间）

根据 TDMQ CMQ 版最小未消费时间，计算 key 过期时间点：

$\text{min\_msg\_time} + \text{max\_retry\_time} + \text{max\_network\_time}$ （最小未消费时间）+（最大重试时间）+（最大网络时间）

TDMQ CMQ 版可配置消息最大有效时间为15天，业务可根据实际情况调整。TDMQ CMQ 版队列当前最小未消费消息时间，即下图中最远时间点。该时间之前的消息都已经被删除，之后的消息可能未被删除。



## 举例说明

避免重复提交：

- 场景：A 为生产者，B 为消费者，中间是 TDMQ CMQ 版。A 已完成10元转账操作，且将消息发送给 TDMQ CMQ 版，TDMQ CMQ 版也已成功收到。此时网络闪断或者客户端 A 宕机导致服务端应答给客户端 A 失败。A 会认为发送失败，从而再次生产消息。这会造成重复提交。
- 解决方法：A 在生产消息时，加入 time 时间戳等信息，生成唯一的去重 key。若生产者 A 由于网络问题判断当前发送失败，重试时，去重 key 沿用第一次发送的去重 key。此时消费者 B 可通过去重 key 判断并做去重。该案例也说明了不能使用 TDMQ CMQ 版的 Message ID 进行去重，因为这两条消息有不同的 ID，但却有相同的 body。

注意事项：生产者 A，在发送消息之前，要将去重 key 做持久化（写磁盘等，避免掉电后丢失）。

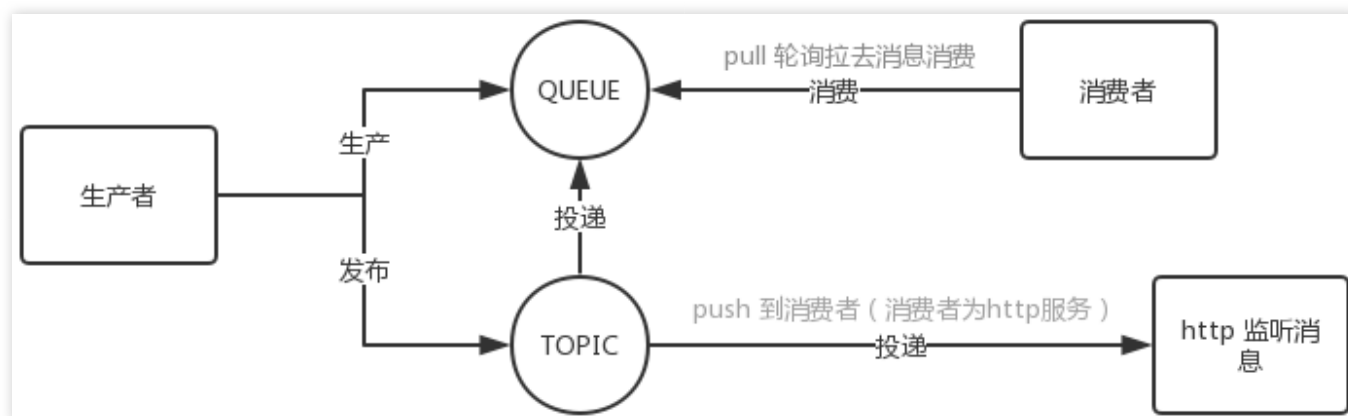
避免多条相同 body 的消息被过滤：

- 场景：A 给 B 转账10元，一共发起5次，每一次提交的 body 内容是一样的。如果消费者粗暴用 body 做去重判断，就会把5次请求，当做1次请求来处理。
- 解决方法：A 在生产消息时，加入 time 时间戳等信息。此时哪怕消息 body 一样，生成的去重 key 都是不同的，这样就满足了多次发送同样内容的需求。

# 选择 Push 还是 Pull

TDMQ CMQ 版支持Pull（队列）和 Push（主题）两种方式：

- Push 模型：当 Producer 发出的消息到达后，服务端马上将这条消息投递给 Consumer。
- Pull 模型：当服务端收到这条消息后什么也不做，只是等着 Consumer 主动到自己这里来读，即 Consumer 这里有一个“拉取”的动作。



本文将结合不同场景，简要分析 Push 模型和 Pull 模型各自存在的利弊。

## 场景1：Producer 速率大于 Consumer 速率

当 Producer 速率大于 Consumer 速率时，有两种可能性：一种是 Producer 本身的效率就要比 Consumer 高（例如，Consumer 端处理消息的业务逻辑可能很复杂，或者涉及到磁盘、网络等 I/O 操作）；另一种是 Consumer 出现故障，导致短时间内无法消费或消费不畅。

Push 方式由于无法得知当前 Consumer 的状态，所以只要有数据产生，便会不断地进行推送，在以上两种情况下时，可能会导致 Consumer 的负载进一步加重，甚至是崩溃（例如生产者 flume 疯狂抓日志，消费者是 HDFS+hadoop，处理效率跟不上）。除非 Consumer 有合适的反馈机制能够让服务端知道自己的状况。

而采取 Pull 的方式问题就简单了许多，由于 Consumer 是主动到服务端拉取数据，此时只需要降低自己访问频率即可。举例：如前端是 flume 等日志收集业务，不断向 TDMQ CMQ 版生产消息，TDMQ CMQ 版向后端投递，后端业务如数据分析等业务，效率可能低于生产者。

## 场景2：强调消息的实时性

采用 Push 的方式时，一旦消息到达，服务端即可马上将其推送给消费端，这种方式的实时性显然是非常好的；而采用 Pull 方式时，为了不给服务端造成压力（尤其是当数据量不足时，不停的轮询显得毫无意义），需要控制好自己轮询的间隔时间，但这必然会给实时性带来一定的影响。

## 场景3：Pull 的长轮询

Pull 模式存在的问题：由于主动权在消费方，消费方无法准确地决定何时去拉取最新的消息。如果一次 Pull 取到消息了还可以继续去 Pull，如果没有 Pull 取到消息则需要等待一段时间再重新 Pull。

由于等待时间很难判定。您可能有很多种动态拉取时间调整算法，可能依然会遇到问题，是否有消息到来不是由消费方决定。也许1分钟内连续到来3000条消息，接下来的几分钟或几个小时内却没有新消息产生。

TDMQ CMQ 版提供了长轮询的优化方法，用以平衡 Pull/Push 模型各自的缺点。基本方式是：消费者如果尝试拉取失败，不是直接 return，而是把连接挂在那里 wait，服务端如果有新的消息到来，把连接拉起，返回最新消息。

## 场景4：部分或全部 Consumer 不在线

在消息系统中，Producer 和 Consumer 是完全解耦的，Producer 发送消息时，并不要求 Consumer 一定要在线，对于 Consumer 也是同样的道理，这也是消息通信区别于 RPC 通信的主要特点；但是对于 Consumer 不在线的情况，却有很多值得讨论的场景。

首先，在 Consumer 偶然宕机或下线时，Producer 的生产是可以不受影响的，Consumer 上线后，可以继续之前的消费，此时消息数据不会丢失；但是如果 Consumer 长期宕机或是由于机器故障无法再次启动，就会出现问题，即服务端是否需要为 Consumer 保留数据，以及保留多久的数据等。

采用 Push 方式时，因为无法预知 Consumer 的宕机或下线是短暂的还是持久的，如果一直为该 Consumer 保留自宕机开始的所有历史消息，那么即便其他所有的 Consumer 都已经消费完成，数据也无法清理掉，随着时间的积累，队列的长度会越来越大，此时无论消息是暂存于内存还是持久化到磁盘上（采用 Push 模型的系统，一般都是将消息队列维护于内存中，以保证推送的性能和实时性，这一点会在后边详细讨论），都将对 TDMQ CMQ 版服务端造成巨大压力，甚至可能影响到其他 Consumer 的正常消费，尤其当消息的生产速率非常快时更是如此；但是如果保留数据，那么等该 Consumer 再次起来时，则要面对丢失数据的问题。

折中的方案是：TDMQ CMQ 版给数据设定一个超时时间，当 Consumer 宕机时间超过这个阈值时，则清理数据；但这个时间阈值也并不太容易确定。

在采用 Pull 模型时，情况会有所改善；服务端不再关心 Consumer 的状态，而是采取“你来了我才服务”的方式，Consumer 是否能够及时消费数据，服务端不会做任何保证（也有超时清理时间）。

## 常见问题

当前队列的消息数小于当前请求的批量消费数量，请求会阻塞吗？

不会阻塞。

目前 SDK 是异步模式吗？

HTTP SDK 所有操作均为同步模式，TCP SDK 接入可选择异步模式。

可以在图形管理界面中查看队列的情况吗？

TDMQ CMQ 版提供可视化控制台，您可以很方便地查看当前队列情况。

TDMQ CMQ 版目前支持 MQTT 协议吗？

TDMQ CMQ 版目前不支持 MQTT 协议。

TDMQ CMQ 版收发消息大小有限制吗？

TDMQ CMQ 版单条消息大小为1MB，单次请求的大小为1MB（指 HTTP post body 大小上限为1MB）。当消息体



大于1KB时，建议使用 post 请求方式，get 方式会被截断触发异常。如果消息超过1MB，您可对消息进行重新拆分。

## TDMQ CMQ 版使用哪种协议访问？

TDMQ CMQ 版使用 HTTP、TCP 协议，TDMQ CMQ 版 SDK 会维护 TCP 长连接。

## TDMQ CMQ 版会进行消息去重吗？

目前 TDMQ CMQ 版不会对消息进行去重。重复消息出现的原因以及去重方案，您可以参见 [消息去重](#)。

## TDMQ CMQ 版消费消息时，pull 的长轮询是如何实现的？

队列有 pollingWaitSeconds 属性，表示默认的队列长轮询时间。消费消息时如果有消息，则直接返回消息；如果当前队列没有消息，则会等待 pollingWaitSeconds 时间，如果在该时间内有消息则返回消息；如果超过了该时间仍然没有消息，则会报没有消息的异常。

这里用户可以根据自己的需求来设置队列的这个属性，而且用户也可以在消费消息的时候指定 pollingWaitSeconds 时间，不用每次消费消息都使用这个默认值。

## 删除消息失败时如何处理？

删除消息失败可能是因为消息句柄超时了。队列属性 visibilityTimeout 表明了消息的可见时间，如果从消费消息到删除消息超过了这个时间，那么消息句柄就会失效，从而导致无法删除消息。

## 调用TDMQ CMQ 版时出现 10250 qps throttling 异常如何解决？

出现该异常是由于 QPS 达到了上限值。QPS 默认值为5000，表示每秒发送（发布）消息的上限为5000 条，消费的消息条数默认为生产的1.1倍。如果对 QPS 有要求，您可以咨询技术支持申请提高 QPS 上限。

## 如果没有需要的 SDK 应该如何操作？

如果没有对应的语言，可以按照文档中的规范自己组包调用 TDMQ CMQ 版服务。

## 如何使 API 密钥只对 TDMQ CMQ 版的接口有用？

API 密钥是全局的密钥，目前 TDMQ CMQ 版已经接入 CAM，您将可以使用 CAM 进行权限控制。

# 开发者指南

## SDK 文档

### 概述

消息队列 CMQ 目前支持 Java、Python、PHP 及 C++ SDK，后续会支持更多语言。也欢迎广大开发者根据 API 说明开发更多语言的 SDK 版本。

由于分配资源和释放资源需要1s 左右的时间，当前消息队列 SDK 在创建及删除队列/主题时会有1s 延迟，建议在程序中增加创建和删除的时间间隔保障调用成功。

# TCP SDK

CMQ 目前已开放基于 TCP 协议的 SDK 调用，支持普通消息、事务消息、延迟消息、异步消息的收发功能。其中，事务消息特性仅通过 TCP 方式实现。

TCP 协议目前支持公网访问以及同地域的私有网络的 CVM 内网访问方式，暂时不支持基础网络的内网访问方式。

本文主要介绍 TCP SDK 使用方式，提供 Demo 工程的安装、下载、配置及运行示例，帮助工程师快速搭建 CMQ 测试工程。

## TCP 协议优势

- 更少的计算资源

HTTP 针对请求鉴权，每次请求都需要签名；TCP 针对链接鉴权，只需要建立连接的时候对链接鉴权，节约客户端计算资源。

- 更安全的客户端线程

TCP 客户端线程安全，多个线程可使用相同的链接，节省链接资源。

- 更高效的传输效率

TCP 传输提高有效数据占比，在相同客户端下，拥有更高的吞吐量和 QPS，相比 HTTP 具有更高效的传输效率。

- 更优的使用体验

- TCP 支持异步接口，支持回调。

## Demo工程使用

### 准备 Demo 环境

#### 1. 安装 IDE

您可以安装 IntelliJ IDEA 或者 Eclipse，本文以 IntelliJ IDEA 为例进行说明。

请在 [下载 IntelliJ IDEA Ultimate 版本](#)，并参考 IntelliJ IDEA 说明进行安装。

#### 2. 下载 Demo 工程

请在 [下载 CMQ 的 Demo 工程](#) 到本地，解压后即可看到本地新增的 cmq-java-tcp-sdk-master 文件夹。

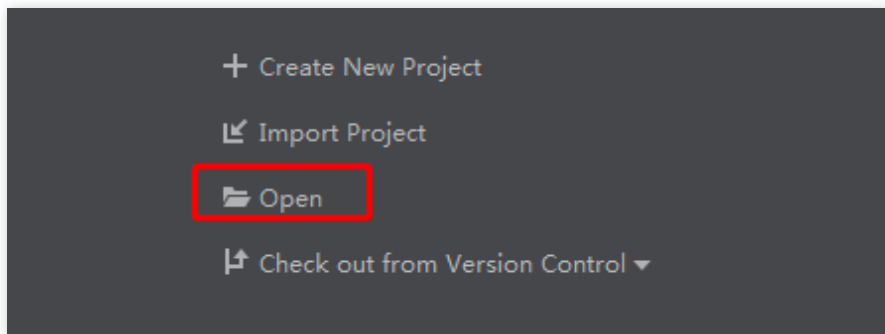
### 配置 Demo 工程

#### 1. 创建资源

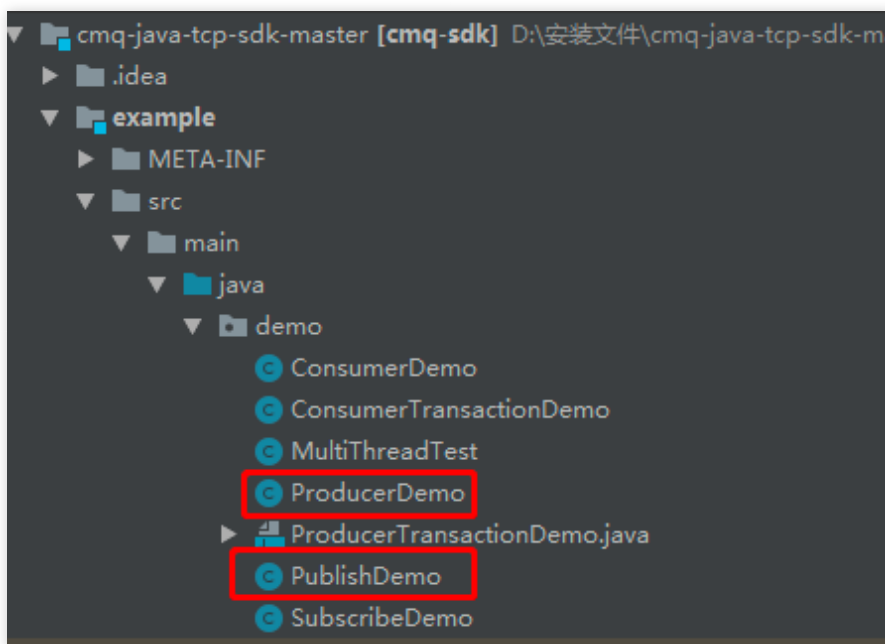
您需要在控制台创建所需消息队列资源，包括 CMQ 队列名、SecretID、SecretKey。

导入 Demo 工程文件

在 IDEA 的开机界面打开文件夹。



打开文件夹后，文件层级关系如下，Demo 工程文件存于 Demo 文件夹下。



## 2. 配置 Demo 参数

修改文件 NameServer 地址、密钥对及消息队列名，NameServer 地址请参考 [NameServer 对照表](#)。

以 ProducerDemo 为例，配置如下：

```
producer.setNameServerAddress("对应的NameSever");  
producer.setSecretID("获取的SecretID")  
producer.setSecretKey("获取的SecretKey")  
String queue = "创建的队列名"
```

具体图示如下：

```

public class ProducerDemo {
    public static void main(String args[]) {

        Producer producer = new Producer();
        // 设置 Name Server地址，在控制台上获取，必须设置
        producer.setNameServerAddress("http://cmq-nameserver-bj.tencentcloudapi.com");
        // 设置SecretId，在控制台上获取，必须设置
        producer.setSecretId("AKIDX1LbH2DN08MWm0dx DVWc");
        // 设置SecretKey，在控制台上获取，必须设置
        producer.setSecretKey("F2jJjAWr5A5v8D8sT");
        // 设置签名方式，可以不设置，默认为SHA1
        producer.setSignMethod(ClientConfig.SIGN_METHOD_SHA256);
        // 设置发送消息失败时，重试的次数，设置为0表示不重试，默认为2
        producer.setRetryTimesWhenSendFailed(3);
        // 设置请求超时时间，默认3000ms
        producer.setRequestTimeoutMS(8000);

        // 消息发往的队列，在控制台创建
        String queue = "subuin";
        try {

```

## 运行 Demo

### 使用队列模型收发消息

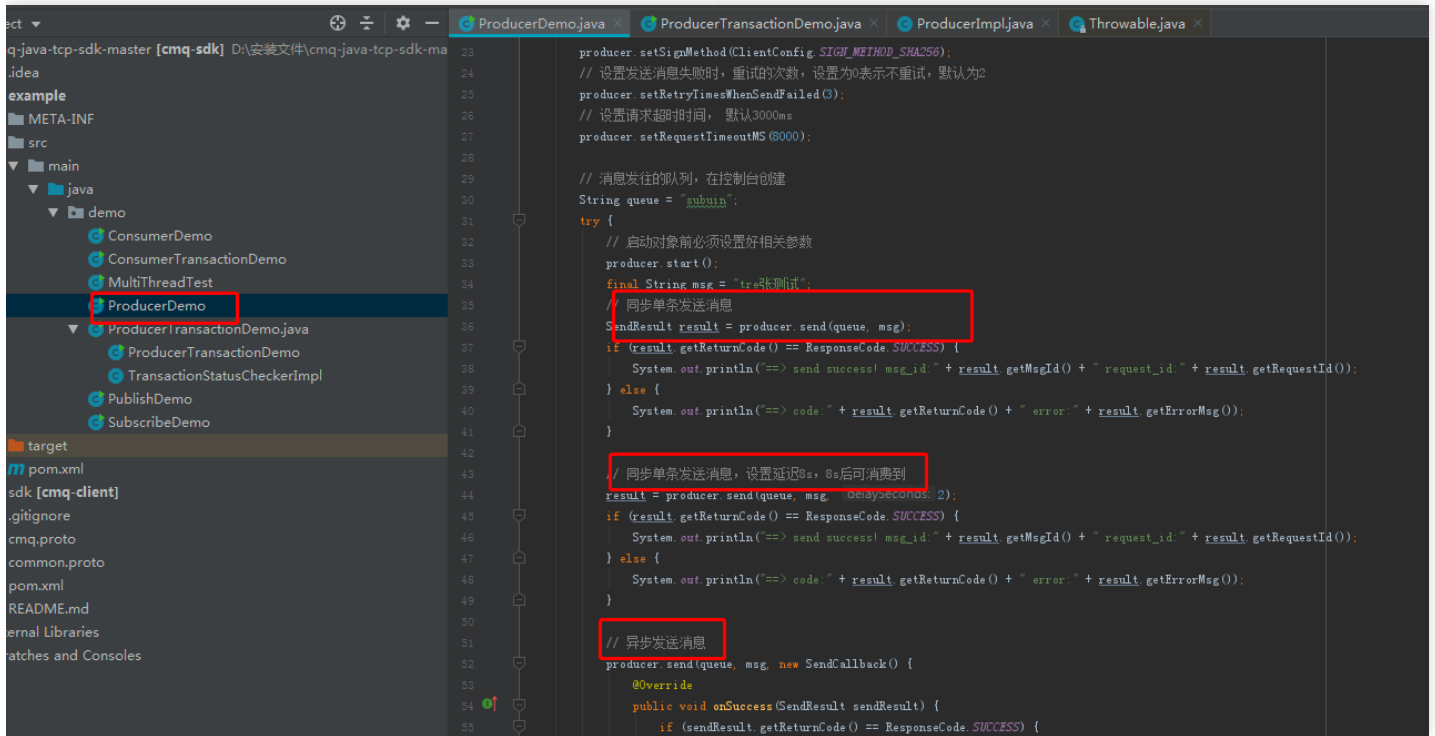
执行文件 ProducerDemo，成功后显示日志如下：

```

2019-06-26 17:26:32,448 INFO NettyClient.java(666) : createChannel: begin to connect remote host[140.143.52.21:12000] asynchronously
2019-06-26 17:26:32,470 INFO NettyClient.java(760) : NETTY CLIENT PIPELINE: CONNECT UNKNOWN => 140.143.52.21:12000
==> send success! msg_id:4222124650659902 request_id:1
==> send success! msg_id:4222124650659903 request_id:3
==> send success! msg_id:4222124650659904 request_id:4
==> send success! msg_id:[4222124650659905, 4222124650659906] request_id:3
==> send success! msg_id:[4222124650659907, 4222124650659908] request_id:6
2019-06-26 17:26:37,890 INFO NettyClient.java(573) : closeChannel: begin close the channel[140.143.52.21:12000] Found: true
2019-06-26 17:26:37,890 INFO NettyClient.java(586) : closeChannel: the channel[140.143.52.21:12000] was removed from channel table
2019-06-26 17:26:37,891 INFO NettyClient.java(775) : NETTY CLIENT PIPELINE: CLOSE 140.143.52.21:12000
2019-06-26 17:26:37,892 INFO MQClientInstance.java(108) : the client instance [10.66.130.82@21656] shutdown OK

```

ProducerDemo 支持普通消息、延时消息、异步消息的发送。



执行文件 ConsumerDemo，可接收消息。

使用主题模型收发消息

1. 运行 PublishDemo 类以主题模型进行消息发送。
2. 运行 SubscriberDemo 类以主题模式进行消息接收。

# HTTP SDK

## Demo工程使用

### 准备 Demo 环境

#### 1. 安装 IDE

您可以安装 IntelliJ IDEA 或者 Eclipse，本文以 IntelliJ IDEA 为例进行说明。

请在 [下载 IntelliJ IDEA Ultimate 版本](#)，并参考 IntelliJ IDEA 说明进行安装。

#### 2. 下载 Demo 工程

请在 [下载 CMQ-HTTP 的 Demo 工程](#) 到本地，解压后即可看到本地新增的 cmq-java-sdk-master 文件夹。

### 配置 Demo 工程

#### 1. 创建资源

您需要在控制台创建所需消息队列资源，包括 CMQ 队列名、SecretID、SecretKey。

#### 2. 导入 Demo 工程文件

在 IDEA 的开机界面打开文件夹。

打开文件夹后，Demo 工程文件存于 /src/main/java/com/qcloud/cmq/example 文件夹下。

#### 3. 配置 Demo 参数

修改文件请求地址、密钥对等。以 Producer 为例，配置如下：

```
String secretId="获取的SecretID"; // 控制台账号访问管理可查看
String secretKey="获取的SecretKey"; // 控制台账号访问管理可查看
String endpoint = "https://cmq-topic-{$region}.api.qcloud.com"; // 控制台获取
String queueName = "test-queue"; // 队列名称
String topicName = "test-topic"; // 主题名称
Account account = new Account(endpoint, secretId, secretKey);
```

### 运行 Demo

#### 使用队列模型收发消息

先运行 Producer 类发送消息，再运行 Consumer 类接受消息。

发送消息代码示例：

```
Queue queue = account.getQueue(queueName);
String msg = "hello!";
String msgId = queue.sendMessage(msg);
System.out.println("==> send success! msg_id:" + msgId);
```



接收/删除消息代码示例：

```
Message message = queue.receiveMessage();
System.out.println("==> receive a message msgId: " + message.msgId);
queue.deleteMessage(message.receiptHandle);
```

使用主题模型收发消息

运行 TopicDemo 类。发布消息示例：

```
Topic topic = account.getTopic(topicName)
String msg = "hello!";
String msgId = topic.publishMessage(msg);
System.out.println("==> publish success! msg_id:" + msgId);
```

订阅者可以填写一个 Queue，使用队列来接收发布的消息。订阅者需要去控制台创建

```
Message message = queue.receiveMessage();
System.out.println("==> receive a message msgId: " + message.msgId);
queue.deleteMessage(message.receiptHandle);
```

新建订阅

订阅名

请输入订阅名

以字母起始，只能包含字母、数字、“-”及“\_”，最大64字符，创建后不能修改

订阅者属性

订阅者类型

Queue队列服务

HTTP URL地址

HTTPS URL地址

订阅队列

请选择

消息推送格式

消息原文

消息过滤标签

点击输入框开始编辑

重试策略

退避重试

衰退指数重试

提交

关闭

Tag消息

Topic 目前支持消息过滤，即消息标签、消息类型，用来区分某个 CMQ 的 Topic 下的消息分类。MQ 允许消费者按照标签对消息进行过滤，确保消费者最终只消费到他关心的消息类型。该功能默认不开启，未开启时，所有消息向所有订阅者发送；增加标签后，订阅者将仅能收到带该标签的信息。消息过滤标签描述了该订阅中消息过滤的标签（标签一致的消息才会被推送）。单个标签不超过16个字符，单个 Message 可最多添加5个标签。

有4个订阅者 A、B、C、D，A 消息标签是 apple，B 消息标签是 xiaomi，C 的消息标签是 imac+xiaomi，D 没有设置任何标签。

此时，有生产者 publish 到 Topic 100条消息，带的消息标签过滤为：apple、imac、iphone、macbook。此后 Topic 立即投递到 A、B、C、D。

场景分析：

|     |      |        |
|-----|------|--------|
| 订阅者 | 消息标签 | 消息接收说明 |
|-----|------|--------|

| 订阅者 | 消息标签        | 消息接收说明                                   |
|-----|-------------|------------------------------------------|
| A   | apple       | 由于 apple 能匹配消息过滤标签中的 apple，则能正常收到100条消息。 |
| B   | xiaomi      | 不能收到任何消息。                                |
| C   | imac+xiaomi | 由于其中一个标签 imac 能匹配，则能正常收到100条消息。          |
| D   | -           | 可以收到任何消息。                                |

通过以上场景，标签匹配规律如下：

| 订阅者 | 订阅者有无 tag | 有无消息 tag | 消息接收说明                                                                    |
|-----|-----------|----------|---------------------------------------------------------------------------|
| A   | 有         | 无        | 订阅者不匹配，不能收到消息。                                                            |
| B   | 无         | 有        | 投递时消息不用匹配，订阅者都能收到消息。                                                      |
| C   | 有         | 有        | 两者匹配的，才能收到消息。支持 N:M 匹配，如消息有10个 tag，订阅者有4个 tag，其中有1个 tag 相互能匹配上，则订阅者能收到消息。 |
| D   | 无         | 无        | 投递后，所有订阅者都能收到消息。                                                          |

```
String msg = "hello world, Time:" + new Date();
List<String> tagList = Collections.singletonList("This-is-A-Tag");
String messageId = topic.publishMessage(msg, tagList, null);
```

## 路由匹配

- CMQ 的路由键匹配功能类似于 rabbitMQ 的 exchange queue，可以用于消息过滤，根据不同的条件使订阅者获取不同的消息。创建 topic 时，可开启【路由匹配键】。
- Binding key、Routing key 是组合使用的，提供类似于 RabbitMQ 的消息过滤能力。发消息时配的 Routing key 是客户端发消息带的。创建订阅关系时配的 Binding key 是 topic 和 订阅者的绑定关系。

## 使用限制

- Binding key 的数量不超过5个。单个 binding key 的长度  $\leq 64$  字节，用于表示发送消息的路由路径，最多含有15个“.”，即最多16个词组。
- Routing key 的数量由1个字符串组成。单个 Routing key 的长度  $\leq 64$  字节，用于表示发送消息的路由路径，最多含有15个“.”，即最多16个词组。

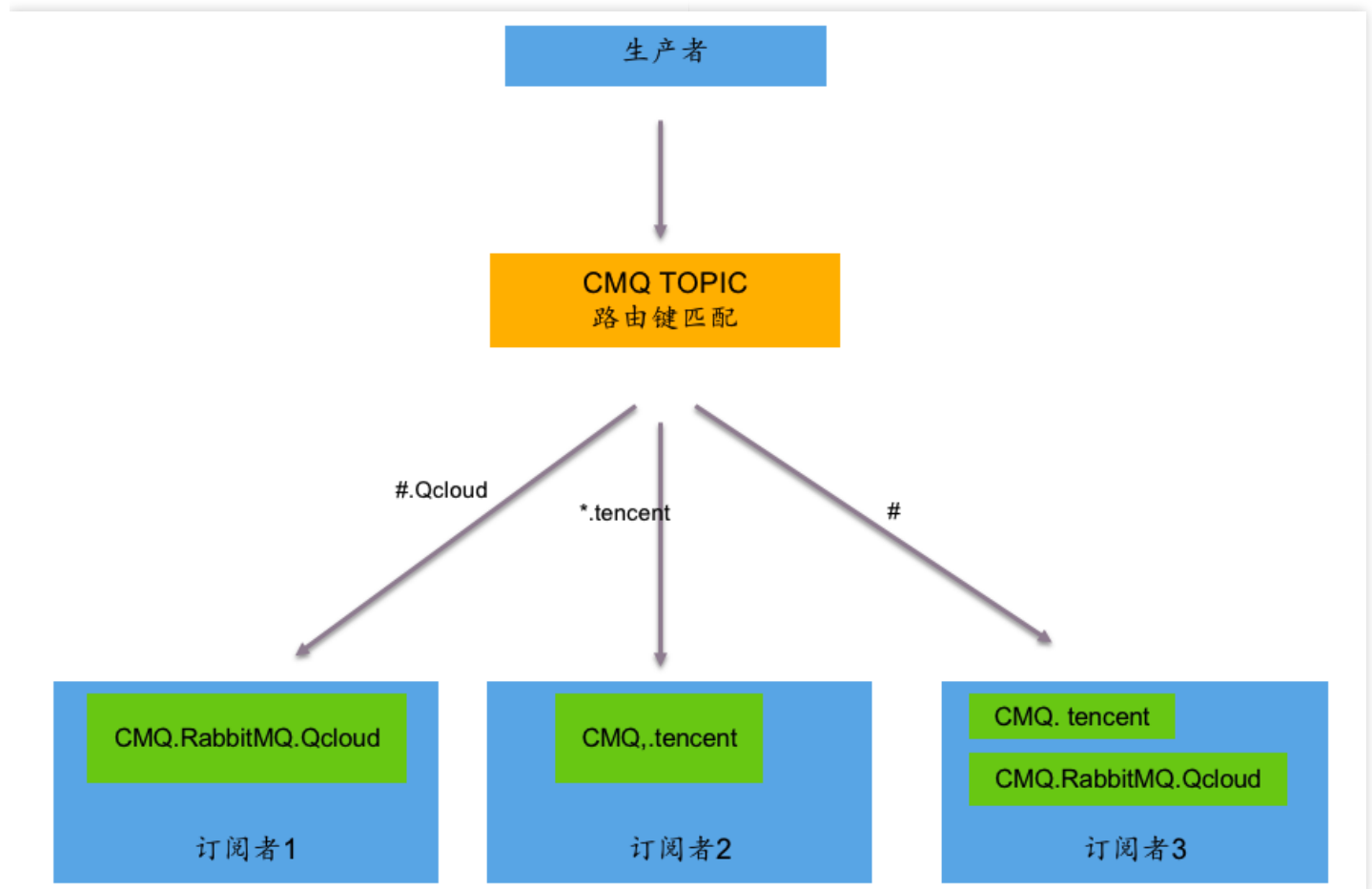
## 通配符说明

- \*（星号），可以替代一个单词（一串连续的字母串），不能为空。

- #（井号）：可以匹配零个或多个字符。

示例：

- 订阅者是『1.\*.0』，此时消息为『1.任意字符.0』，则订阅者都能收到消息。
- 订阅者是『1.#.0』，此时消息为『1.2.3.4.4.2.2.0』，『1.0』则订阅者都能收到消息（消息中间元素随意）。
- 订阅者是『#』，则所有消息订阅者都能收到。



```
String msg = "hello world, Time:" + new Date();  
String routeKey = "first.second.middle.middle2.last";  
String messageId = topic.publishMessage(msg, null, routeKey);
```

# SDK 下载

## 下载地址

不同语言版本的 SDK 下载地址如下：

- [Java SDK](#)
- [Python SDK](#)
- [PHP SDK](#)
- [C++ SDK](#)
- [C# SDK](#)

GitHub 地址

- [Java SDK](#)
- [Python SDK](#)
- [PHP SDK](#)
- [C++ SDK](#)

## 注意事项

使用 SDK 前至少要获取 SecretId、SecretKey 和 endpoint。endpoint 说明如下。

队列模型

请参照下面说明将域名中的 {\$region} 替换成相应地域：

接口请求域名：http://cmq-queue-{\$region}.api.finance.cloud.tencent.com

主题模型

请参照下面说明将域名中的 {\$region} 替换成相应地域：

接口请求域名：http://cmq-topic-{\$region}.api.finance.cloud.tencent.com

说明

需用具体地域替换。公共参数中的 region 值要与域名的 region 值保持一致，如果出现不一致的情况，以域名的 region 值为准，将请求发往域名 region 所指定的地域。

# 版本说明书

# 版本说明书

## 版本说明

| 版本    | 发布时间            | 版本说明                                                                                                                                                        |
|-------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| V 1.0 | 2023 年 10 月16 日 | TDMQ CMQ 版（TDMQ for CMQ）是一种分布式消息队列服务，它具有可靠的、基于消息的异步通信机制，能够将分布式部署的不同应用（或同一应用的不同组件）中的信息传递，存储在可靠有效的消息队列中，防止消息丢失。TDMQ CMQ 版支持多进程同时读写，收发消息互不干扰，无需各应用或组件始终处于运行状态。 |

## 更多资源

关于 TDMQ CMQ 的其他版本，您可以前往文档中心 进行查看。