

TDMQ for CMQ

Getting Started

Product Documentation



Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Getting Started

Getting Started with Queue Model

Getting Started with Topic Model

Getting Started

Getting Started with Queue Model

Last updated : 2024-01-03 10:15:44

Overview

This document describes how to create a queue service from scratch and use the SDK for Java to test message sending and receiving. It helps you quickly understand the basic operations required for client access to TDMQ for CMQ.

Prerequisites

You have [signed up for a Tencent Cloud account](#).

Directions

Step 1. Create a queue service

1. Log in to the [TDMQ for CMQ console](#).
2. Select **Queue Service** on the left sidebar, select the region, click **Create**, and configure the queue service attributes.

Create Queue

Queue Name ⓘ

Please enter the queue name. It c

Resource Tag

Tag key ▼

Tag value

[+ Add](#)

Queue Attributes

Message Lifecycle ⓘ

day ▼

Value range: 60 seconds 15 days

Long Polling Wait Time for Message Receipt ⓘ

0

secc ▼

Value range: 0 seconds 30 seconds

Hidden Duration of Fetched Message ⓘ

30

secc ▼

Value range: 1 second 12 hours

Dead Letter Queue Settings

Dead Letter Queue ⓘ



Message Heap & Rewind Settings

Max Heaped Messages ⓘ

1

Millic ▼

Value range: 1000000 100Million

Message Rewind ⓘ



Time Range ⓘ

1

day ▼

Value range: 1 second 15 days

[Submit](#)[Disable](#)

| Attribute | Description | Value |
|--|---|--|
| Queue Name | It is the `QueueName` attribute of the queue. | It is the unique identifier of a resource and can be used to differentiate API calls. It cannot be changed once the queue is created. It is case-insensitive and cannot end with `-retry` or `-dlq`. |
| Resource Tag | It is optional and can help you easily categorize and manage TDMQ for CMQ resources in many dimensions. For detailed usage, see Managing Resource with Tag . | - |
| Maximum Message Unacknowledged Time | If the consumer client fails to acknowledge a received message within this time period, the server will automatically acknowledge the message. | It ranges from 30 seconds to 12 hours. |
| Long Polling Wait Time for Message Receipt | It is the `PollingWaitSeconds` attribute of the queue. Just like with long polling of Ajax requests, a message consumption request will return a response only after a valid message is fetched or the long-polling time elapses. | It ranges from 0 to 30 seconds. We recommend you set it to 3 seconds. A higher value may cause more duplicated messages. |
| Hidden Duration of Fetched Message | It is the `VisibilityTimeout` attribute of the queue. Each message has a default `VisibilityTimeout`, which starts counting after a worker receives a message. If the worker fails to complete processing the message within the period specified by this attribute, the message will be sent to and processed by another worker. | It ranges from 1 second to 12 hours. |
| Max Invisible Messages | If the client fails to acknowledge messages in a timely manner, an excessive number of invisible messages are generated. The | It is 100,000. If you need to increase the upper limit, contact technical support. |

| | | |
|----------------------------------|--|---|
| | generation of invisible messages will consume the memory; therefore, a capacity upper limit is set for each queue. | |
| Max Capacity for Heaped Messages | Message heap is generally caused by the production speed being greater than the consumption speed or the consumption being blocked. The heap will use disk space; therefore, a capacity upper limit is set for each queue. | It is 10 GB. If you need to increase the upper limit, contact technical support. |
| Dead Letter Queue | A dead letter queue is used to handle messages that cannot be consumed successfully after their retry limit has been reached. Rather than being discarded immediately, such messages will be sent to the consumer's specified dead letter queue. | - |
| Message Rewind | If the message rewind feature is not enabled, a message consumed by a consumer and confirmed for deletion will be deleted immediately. When enabling this feature, you need to specify the rewindable time range. | The rewindable time range must be equal to or shorter than the message lifecycle. We recommend you make it the same as the message lifecycle to facilitate troubleshooting. |
| Rewindable Time Range | If the message rewind feature is enabled, messages confirmed for deletion by the consumer will not be deleted immediately; instead, they will be stored for the maximum time configured here. | It ranges from 1 to 15 days. A higher value may cause higher storage fees. The maximum rewindable time point is the current time minus the configured rewindable time range. Messages cannot be rewound if produced before this time. |
| Rewindable Storage Space | After the message rewind feature is enabled, if the volume of persistently stored messages exceeds this maximum storage space, messages will be deleted by time (the oldest data will be deleted first). | It ranges from 1 to 10 GB. A higher value may cause higher storage fees. |

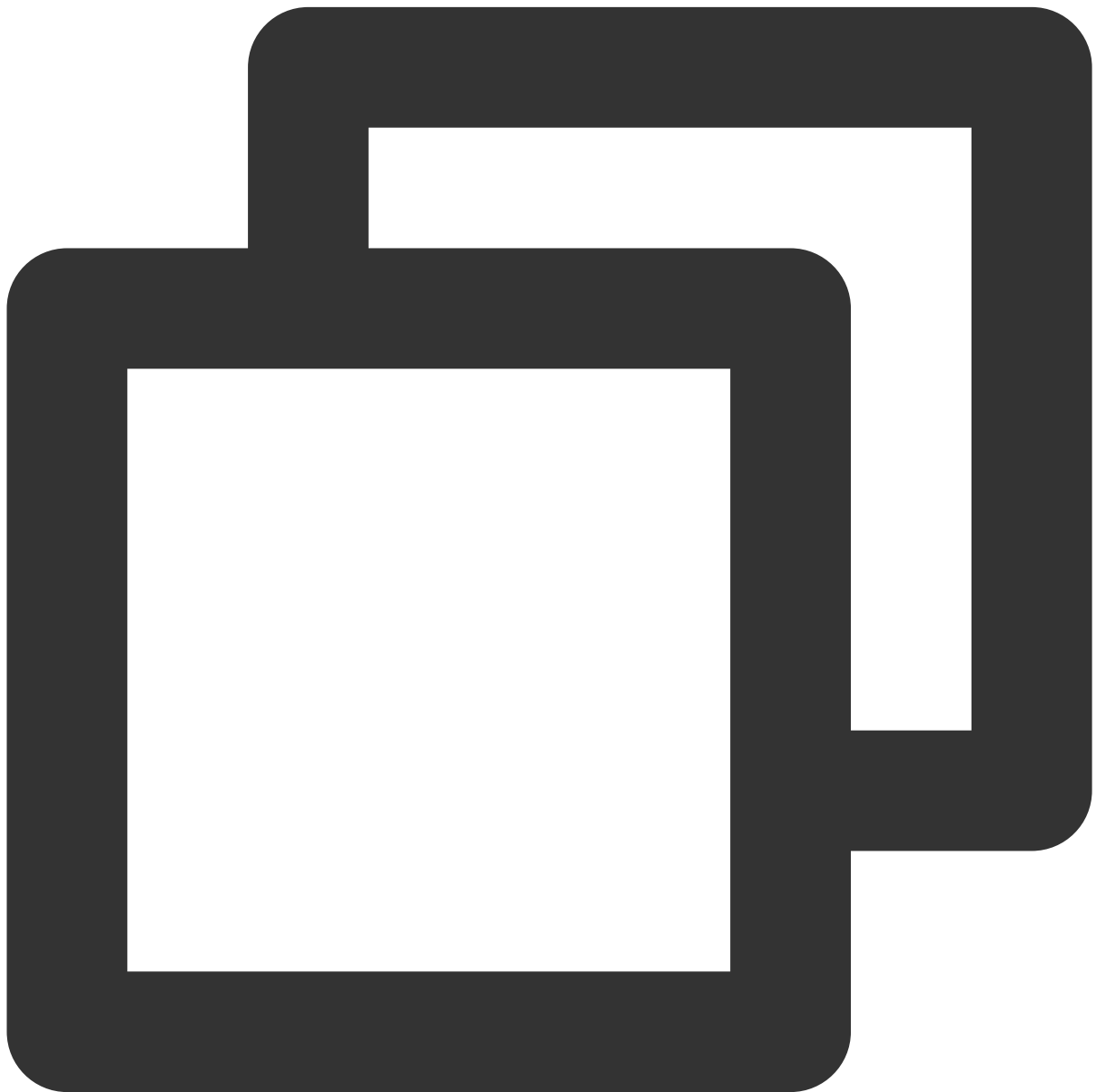
3. Click **Submit**, and you can see the created queue service in the queue service list.

Step 2. Use the SDK to send and receive messages

Note:

The following takes Java as an example. For clients in other languages, see [API Overview](#).

1. [Download the demo](#) and decompress it.
2. Import CMQ client dependencies.



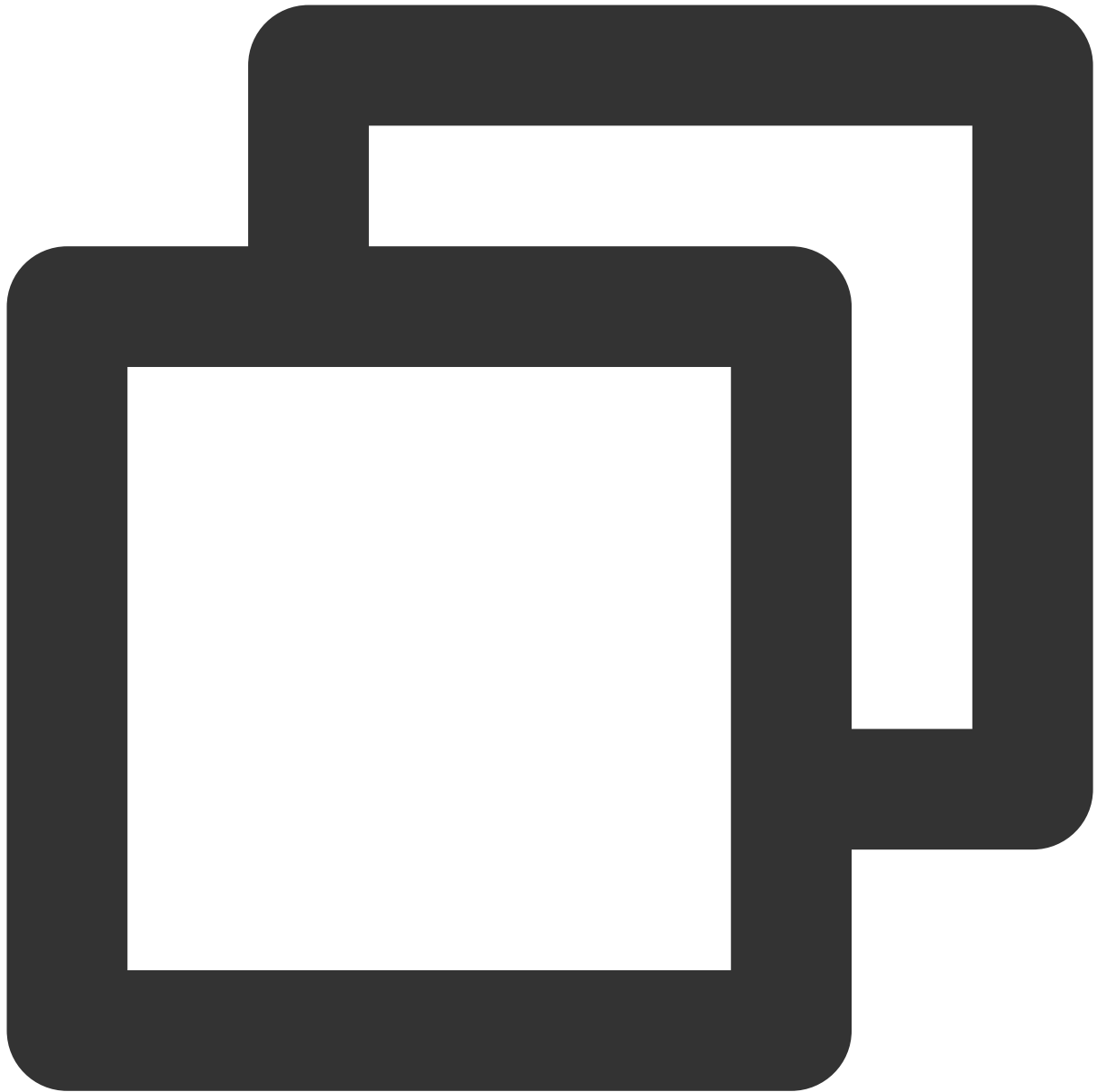
```
<!-- cmq sdk --><!-- cmq sdk -->  
<dependency>
```



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

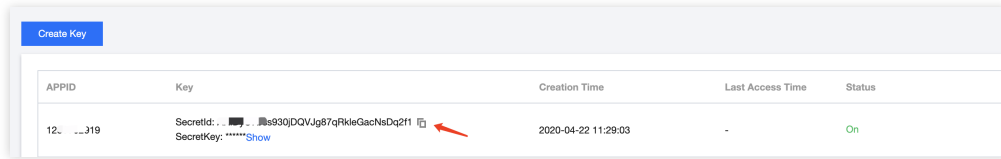
<!-- TencentCloud API SDK -->
<dependency>
  <groupId>com.tencentcloudapi</groupId>
  <artifactId>tencentcloud-sdk-java</artifactId>
  <version>3.1.423</version>
</dependency>
```

3. Send messages.

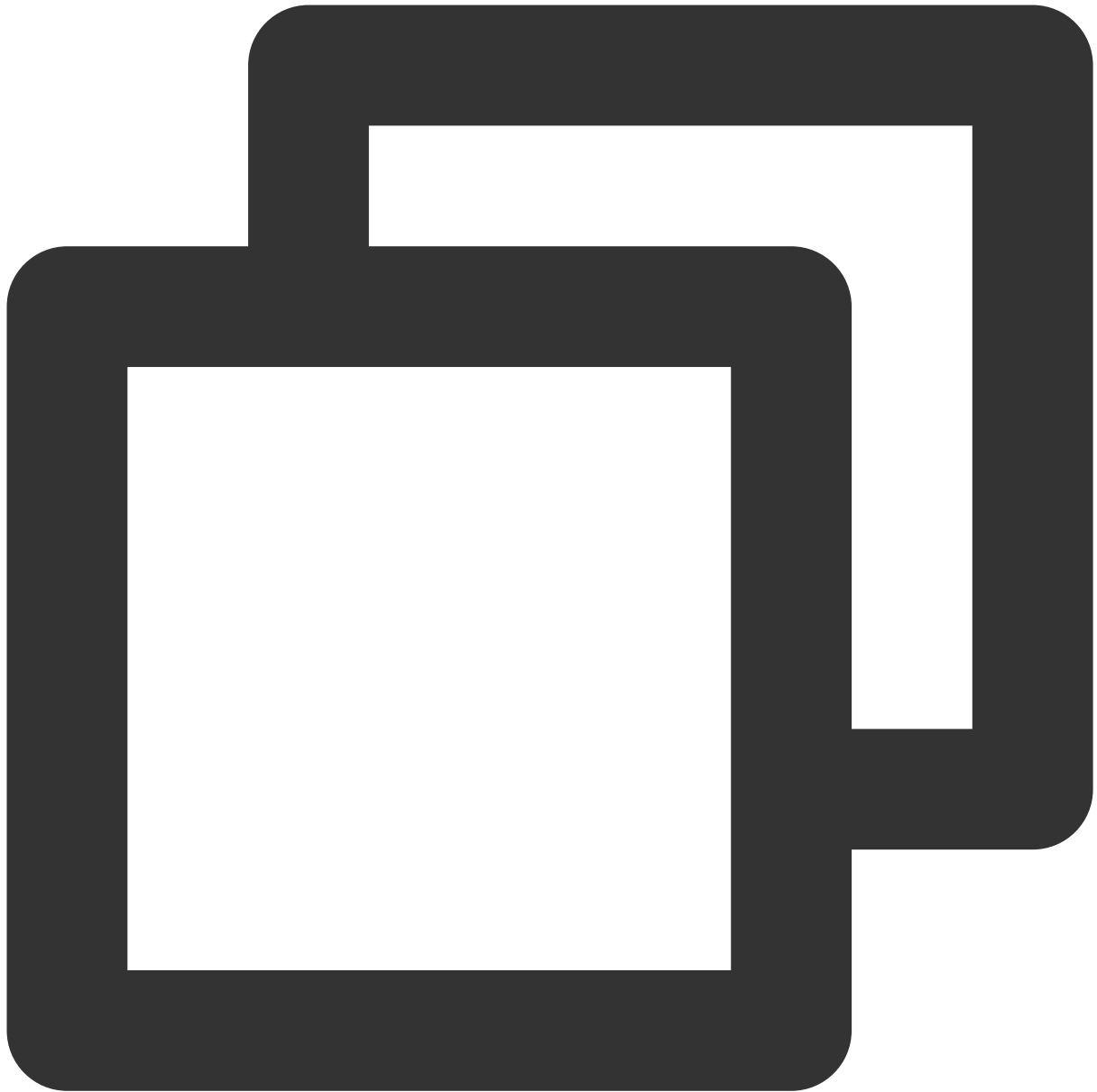


```
Account account = new Account(SERVER_ENDPOINT, SECRET_ID, SECRET_KEY);
Queue queue = account.getQueue(queueName);
String msg = "hello client, this is a message. Time:" + new Date();
CmqResponse response = queue.send(msg);
```

| Parameter | Description |
|-----------------|---|
| SERVER_ENDPOINT | API call address, which can be copied from Queue Service > API Request Address in the console . |

| | |
|--------------------------|---|
| | <div> <div>Note</div> <div> <p>API call address of TDMQ for CMQ:</p> <p>1. Public network address:</p> <p>https://cmq-sh.public.tencenttdmq.com</p> <p>*The URL for calling APIs varies by region</p> <p>2. Private network address:</p> <p>http://sh.mqadapter.cmq.tencentyun.com</p> <p>*The URL for calling APIs varies by region</p> <p>OK</p> </div> </div> |
| SECRET_ID, SECRET_KEY | <p>TencentCloud API key, which can be copied on the Access Key > API Key Management console.</p>  |
| queueName | Queue name, which can be obtained on the Queue Service page in the TDMQ for CMQ c |

4. Consume messages.



```
Account account = new Account (SERVER_ENDPOINT, SECRET_ID, SECRET_KEY);
Queue queue = account.getQueue (queueName);
Message message = queue.receiveMessage();
// Successfully consumed messages are deleted. Retained messages can be delivered
queue.deleteMessage (message.receiptHandle);
```

| Parameter | Description |
|-----------------|---|
| SERVER_ENDPOINT | API call address, which can be copied from Queue Service > API Request Address in the |

[console](#).

Note



API call address of TDMQ for CMQ:

1. Public network address:

<https://cmq-sh.public.tencenttdmq.com>

*The URL for calling APIs varies by region

2. Private network address:

<http://sh.mqadapter.cmq.tencentyun.com>

*The URL for calling APIs varies by region

OK

SECRET_ID,
SECRET_KEY

TencentCloud API key, which can be copied on the Access Key > API Key Management [console](#).

| Create Key | | | | |
|------------|---|---------------------|------------------|--------|
| APPID | Key | Creation Time | Last Access Time | Status |
| 12... | SecretId: ... 930jDQVJg87qRkleGacNsDq2f1 SecretKey: ***** Show | 2020-04-22 11:29:03 | - | On |

queueName

Queue name, which can be obtained on the Queue Service page in the [TDMQ for CMQ console](#).

Note:

Above is a brief introduction to message production and consumption. For more information, see [Demo](#).

Getting Started with Topic Model

Last updated : 2024-01-03 10:15:44

Overview

This document describes how to create a topic and use the SDK for Java to test message sending and receiving, so as to help you quickly understand the basic operations required for client access to TDMQ for CMQ.

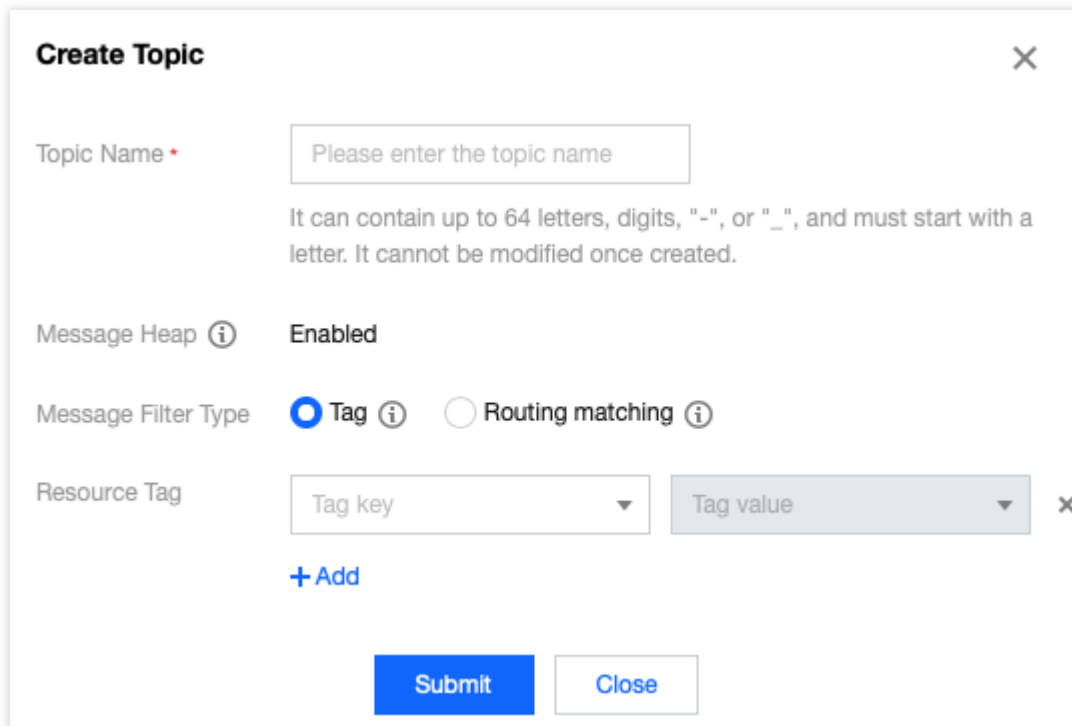
Prerequisite

You have [signed up for a Tencent Cloud account](#).

Directions

Step 1. Create a topic

1. Log in to the [TDMQ for CMQ console](#).
2. Select **Topic Subscription** on the left sidebar, select a region, click **Create**, and enter the topic name.



The 'Create Topic' dialog box contains the following fields and controls:

- Topic Name ***: A text input field with placeholder text 'Please enter the topic name'. Below it, a note states: 'It can contain up to 64 letters, digits, "-", or "_", and must start with a letter. It cannot be modified once created.'
- Message Heap**: A toggle switch labeled 'Enabled'.
- Message Filter Type**: Two radio buttons. 'Tag' is selected (indicated by a blue dot), and 'Routing matching' is unselected.
- Resource Tag**: Two dropdown menus labeled 'Tag key' and 'Tag value', followed by a close icon 'x'.
- + Add**: A blue text link to add more tags.
- Submit** and **Close**: Two buttons at the bottom.

Topic Name: It can contain up to 64 letters, digits, "-", or "_", and must start with a letter. It cannot be modified once created.

Message Heap: If this option is enabled, messages that failed to be pushed to or received by the subscriber will be temporarily heaped in the topic.

Message Filter Type:

Tag: TDMQ for CMQ can match message tags for production and subscription, which can be used for message filtering. For detailed rules, see [Tag Key Matching Feature Description](#).

Routing matching: The binding key and routing key are used together and are fully compatible with the topic match mode of RabbitMQ. The routing key carried when a message is sent is added by the client, and the binding key carried when a subscription is created is the binding relationship between the topic and the subscriber. For detailed rules, see [Routing Key Matching Feature Description](#).

Resource Tag: It is optional and can help you easily categorize and manage TDMQ for CMQ resources in many dimensions. For detailed usage, see [Managing Resource with Tag](#).

3. Click **Submit**, and you can see the created topic in the topic subscription list.

Step 2. Create a subscription

A topic can publish messages only if it is subscribed to by at least one subscriber. If there are no subscribers, messages in the topic will not be delivered, and message publishing will be meaningless.

1. On the [Topic Subscription](#) page, click the ID of the topic you just created to enter the topic details page.
2. Select the **Subscriber** tab at the top, click **Create**, and enter the subscriber information.

Create Subscription

×

Subscription Name

Enter the subscription name

It can contain up to 64 letters, digits, "-", or "_", and must start with a letter. It cannot be modified once created.

Subscriber Attribute

Subscriber Type

☒ Queue service
 ☐ URL

Subscribed Queue

Please select ▼

Message Push Format

Original message

Message Filter Tag

Click on the input box to edit

Retry Policy

☒ Backoff retry
 ☐ Exponential decay retry

Submit

Close

Subscriber Type

Queue service: you can select a queue for the subscriber to receive published messages.

URL: Subscribers can process messages on their own without using queues. For more information, see [Delivering Message](#).

Subscriber Tag: When adding a subscriber, you can add filter tags (FilterTag), so that the subscriber can receive only messages with the specified tags. Up to five tags can be added for one subscriber. As long as a tag matches a topic filter tag, the subscriber can receive messages delivered by the topic. If a message does not have any tag, the subscriber cannot receive it.

Tag: For detailed rules, see [Tag Key Matching Feature Description](#).

Routing matching: For detailed rules, see [Routing Key Matching Feature Description](#).

Retry Policy: After a message is published by a topic, it will automatically be pushed to the subscription. If the push fails, there are two retry policies:

Backoff retry: An attempt will be retried three times at random intervals between 10 and 20 seconds. After three retries, the message will be discarded for the subscriber and will not be retried again.

Exponential decay retry: An attempt will be retried 176 times at exponentially increasing intervals: 2^0 second, 2^1 seconds, ..., 512 seconds, 512 seconds, ..., 512 seconds. The total retry duration is 1 day. This is the default retry policy.

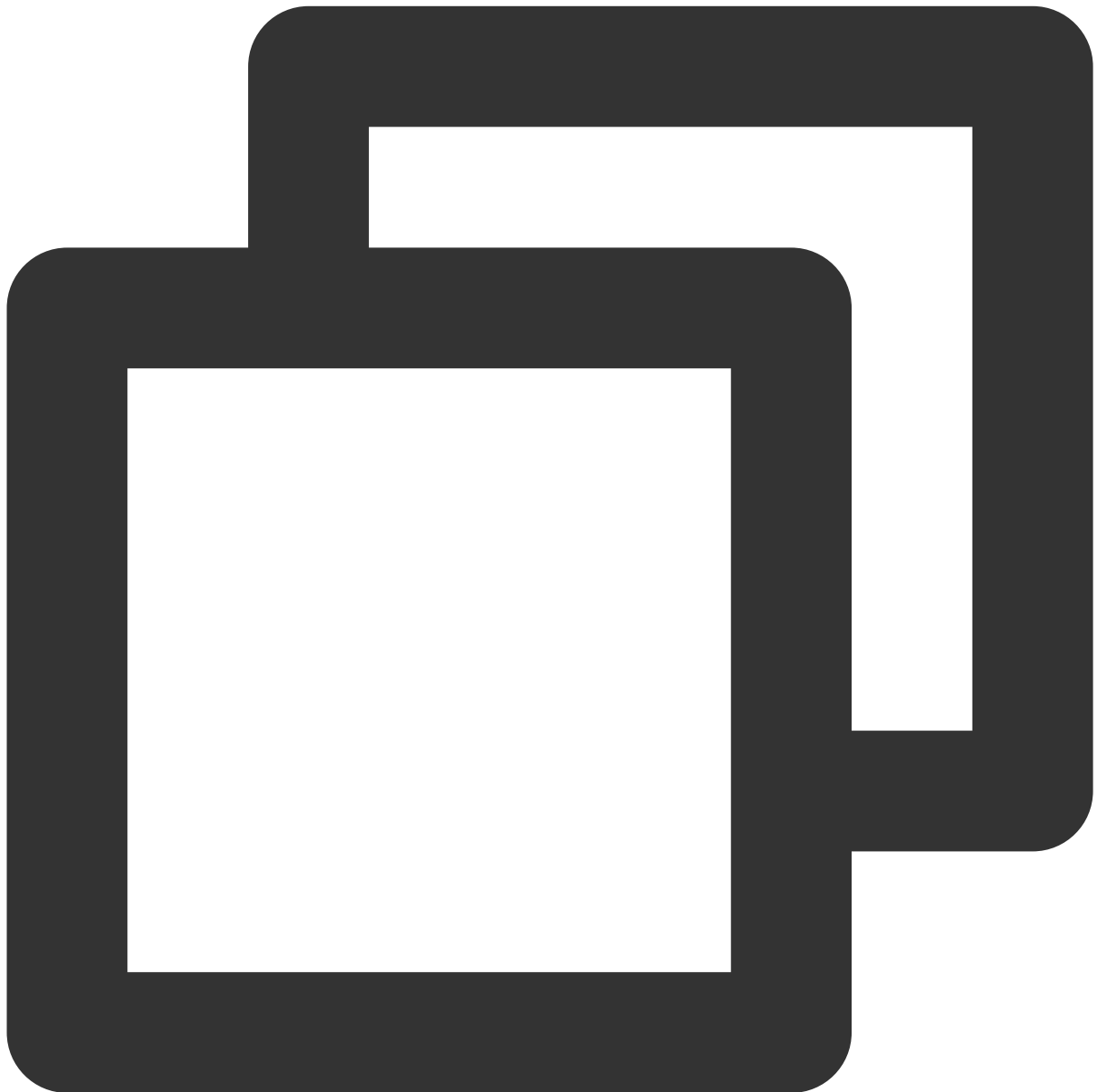
3. Click **Submit**, and you can see the created subscriber in the subscriber list.

Step 3. Use the SDK to send and receive messages

Note:

The following takes Java as an example. For clients in other languages, see the [SDK Documentation](#).

1. [Download the demo](#) and decompress it.
2. Import CMQ client dependencies.

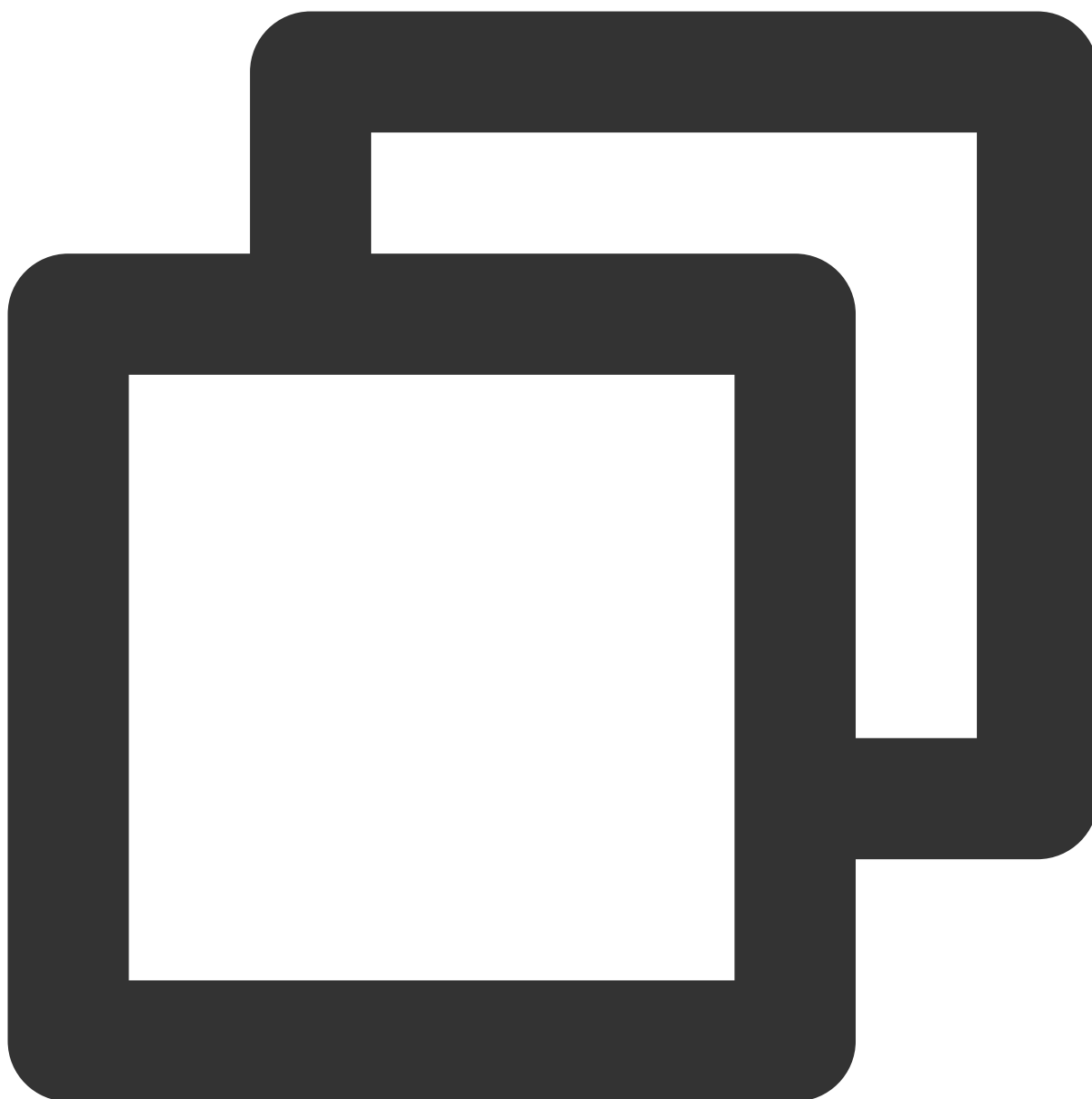


```
<!-- cmq sdk -->
<dependency>
  <groupId>com.qcloud</groupId>
  <artifactId>cmq-http-client</artifactId>
```

```
        <version>1.0.7</version>
    </dependency>

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

3. Create a topic object.



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

| Parameter | Description |
|-----------------|---|
| SERVER_ENDPOINT | API call address, which can be copied from Topic Subscription > API Request Address in TDMQ for CMQ console . |

Note

API call address of TDMQ for CMQ:

1. Public network address:
`https://cloud.tencent.com/public.tencenttdmq.com`
*The URL for calling APIs varies by region
2. Private network address:
`http://gz.n[redacted].cmq.tencentyun.com`
*The URL for calling APIs varies by region

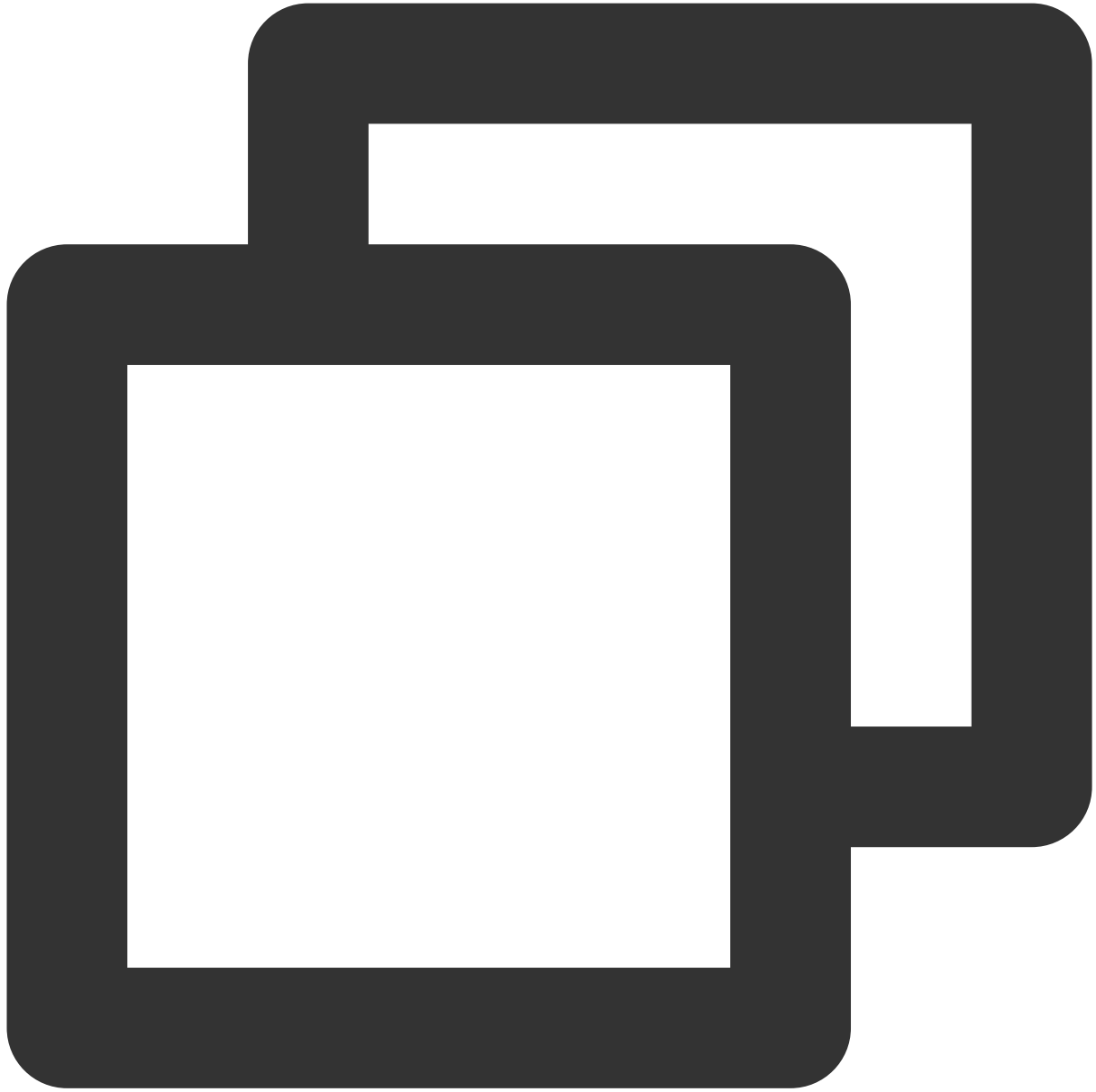
OK

TencentCloud API key, which can be copied on the Access Key > Manage API Key page [CAM console](#).

SECRET_ID,
SECRET_KEY

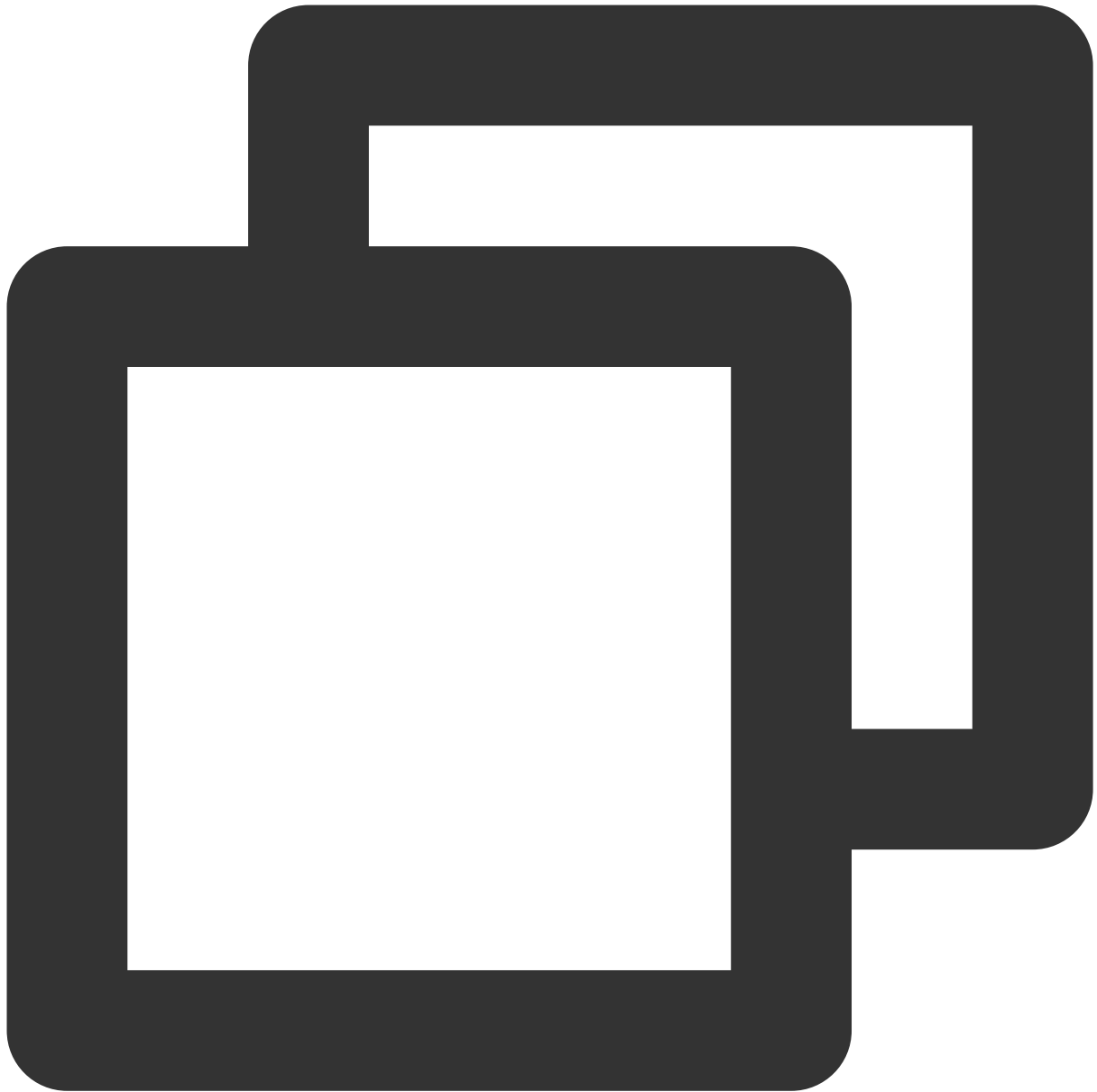
| | |
|-----------|---|
| topicName | Topic subscription name, which can be obtained on the Topic Subscription page in the TI for CMQ console . |
|-----------|---|

4. Send tag messages.



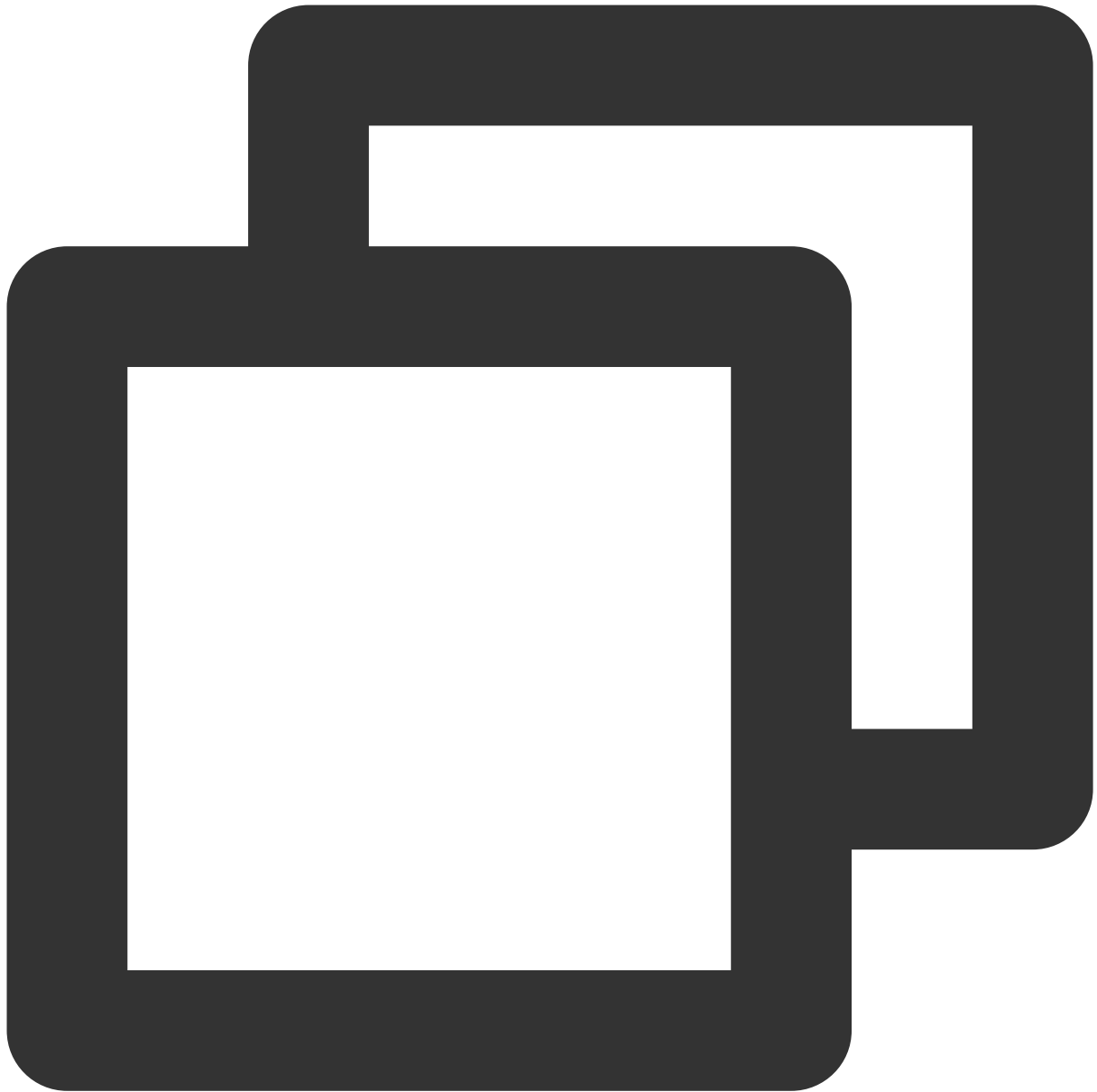
```
String msg = "hello client, this is a message. tag=TAG1. Time:" + new Date();  
List<String> tags = Collections.singletonList("TAG1");  
String messageId = topic.publishMessage(msg, tags, null);
```

5. Send route messages.



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

6. Consume messages in the queue corresponding to the subscriber.



```
Account account = new Account(SERVER_ENDPOINT, SECRET_ID, SECRET_KEY);
Queue queue = account.getQueue(queueName);
Message message = queue.receiveMessage();
// Successfully consumed messages are deleted. Retained messages can be delivered.
queue.deleteMessage(message.receiptHandle);
```

Note:

The above is a brief introduction to the message production and consumption in TDMQ for CMQ. For more information, see [Demo](#).