TDMQ for RocketMQ

SDK Documentation

Product Documentation

Contents

SDK Documentation
Access over TCP
Spring Boot Starter
Sending and Receiving General Messages

Last updated：2023-10-19 11:04:14

Overview

This document describes how to use Spring Boot Starter SDK to send and receive messages and helps you better understand the message sending and receiving processes.

Prerequisites

You have created or prepared the required resources as instructed in Resource Creation and Preparation.

You have installed JDK 1.8 or later.

You have installed Maven 2.5 or later.

You have downloaded the demo or obtained the demo in TencentCloud/rocketmq-demo in GitHub.

Directions

Step 1. Add dependencies

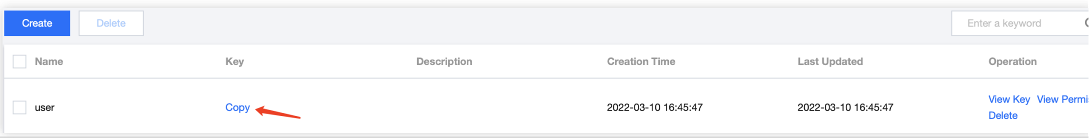
Add dependencies to the `pom.xml` file.

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

Step 2. Prepare configurations

Add configuration information to the configuration file.

```
server:
  port: 8082

# RocketMQ configuration information
rocketmq:
  # Service access address of TDMQ for RocketMQ
  name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
  # Producer configurations
  producer:
    # Producer group name
    group: group111
    # Role token
    access-key: eyJrZXlJZC....
    # Name of the authorized role
    secret-key: admin
  # Common configurations for the consumer
  consumer:
    # Role token
    access-key: eyJrZXlJZC....
    # Name of the authorized role
    secret-key: admin

  # Custom configurations based on business needs
  namespace: rocketmq-xxx|namespace1
  producer1:
    topic: testdev1
  consumer1:
    group: group111
    topic: testdev1
    subExpression: TAG1
  consumer2:
    group: group222
    topic: testdev1
    subExpression: TAG2
```

Parameter	Description
name-server	Cluster access address, which can be obtained from **Access Address** in the **Operation** columr **Cluster** page in the console. The namespace access address can be obtained under the **Names** on the **Cluster** page.
group	Consumer group name, which can be copied under the **Group** tab on the **Cluster** page in the co
secret-key	Role name, which can be copied on the Role Management page.
access-key	Role token, which can be copied in the **Token** column on the Role Management page.  Name | Key | Description | Creation Time | Last Updated | Operation user | Copy | | 2022-03-10 16:45:47 | 2022-03-10 16:45:47 | View Key View Permi Delete
namespace	Namespace name, which can be copied under the **Namespace** tab on the **Cluster** page in the c
topic	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
subExpression	A parameter used to set the message tag.

Step 3. Send messages

1. Inject `RcoketMQTemplate` into the class that needs to send messages.

```
@Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
     private String topic;  // Full topic name, which needs to be concatenated.

     @Autowired
     private RocketMQTemplate rocketMQTemplate;
```

2. Send messages. The message body can be a custom object or a message object that is contained in the package `org.springframework.messaging`.

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

3. Below is a complete sample.

```
/**
* Description: Message producer
*/
@Service
public class SendMessage {
// Use the full name of the topic, which can be either customized or concatenated i
 @Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
 private String topic;
     @Autowired
 private RocketMQTemplate rocketMQTemplate;
         /**
    * Sync sending
    *
    * @param message Message content
    * @param tags    Subscribed tags
    */
 public void syncSend(String message, String tags) {
         // Spring Boot does not support passing tags by using the header. You must
         String destination = StringUtils.isBlank(tags) ? topic : topic + ":" + tag
         SendResult sendResult = rocketMQTemplate.syncSend(destination,
                         MessageBuilder.withPayload(message)
                                         .setHeader(MessageConst.PROPERTY_KEYS, "yo
                                         .build());
         System.out.printf("syncSend1 to topic %s sendResult=%s %n", topic, sendRes
 }
}
```

Note

Above is a sync sending sample. For more information on async sending and one-way sending, see the demo or TencentCloud/rocketmq-demo in GitHub.

Step 4. Consume messages

```
@Service
   @RocketMQMessageListener(
           consumerGroup = "${rocketmq.namespace}%${rocketmq.consumer1.group}",  //
           // Use the full name of the topic, which can be either customized or con
           topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
           selectorExpression = "${rocketmq.consumer1.subExpression}" // Subscripti
   )
   public class MessageConsumer implements RocketMQListener<String> {

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

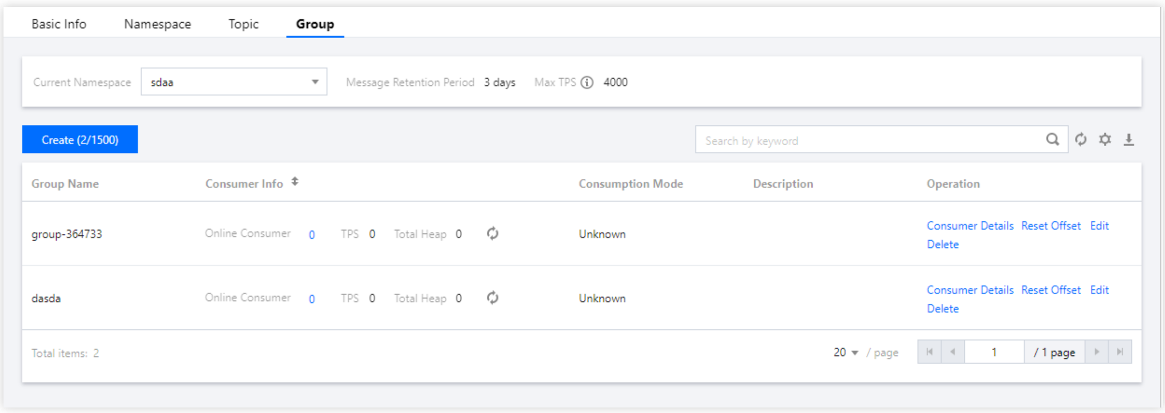
You can configure multiple consumers as needed. The consumer configurations depend on your business requirements.

Note

For a complete sample, download the demo or (https://github.com/TencentCloud/rocketmq-demo/tree/main/java/rocketmq-demo/rocketmq4/src/main/java/com/tencent/demo/rocketmq4/simple!f2025fba6fb266a8503c27ebf173037b) obtain the demo in TencentCloud/rocketmq-demo in GitHub.

Step 5. View consumption details

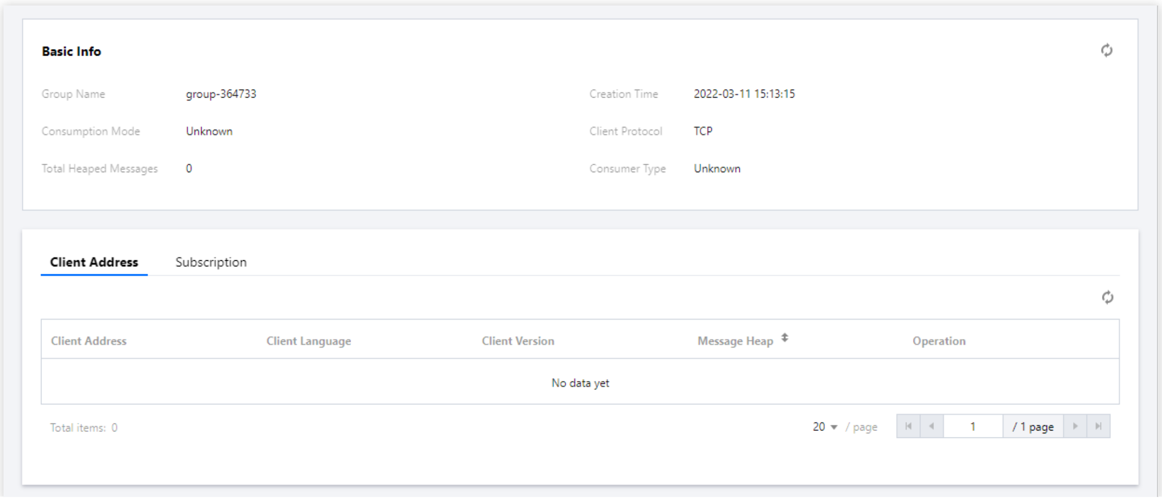
Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the consumer group. Click **Consumer Details** in the **Operation** column to view consumer details.



Current Namespace: sdaa Message Retention Period: 3 days Max TPS: 4000

Group Name	Consumer Info	Consumption Mode	Description	Operation
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown		Consumer Details Reset Offset Edit Delete
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown		Consumer Details Reset Offset Edit Delete

Total items: 2 20 / page 1 / 1 page



Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address	Client Language	Client Version	Message Heap	Operation

No data yet

Total items: 0 20 / page 1 / 1 page

Sending and Receiving Filtered Messages

Last updated：2023-04-12 11:39:41

Overview

This document describes how to use Spring Boot Starter to send and receive messages and helps you better understand the message sending and receiving processes.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have learned about the sending and receiving process of general messages.
You have downloaded the demo here or have downloaded one at the GitHub project.

Directions

Sending a message

This process is the same as that of general messages, but you need to concatenate the topic sent by rocketMQTemplate to corresponding tag.

```
// Spring Boot does not support passing tags by using the header. You must concate
        String destination = StringUtils.isBlank(tags) ? topic : topic + ":" + tags
        // object message type
        SendResult sendResult = rocketMQTemplate.syncSend(destination,
                 MessageBuilder.withPayload(message)
                           .setHeader(MessageConst.PROPERTY_KEYS, "yourKey")    // Spec
                           .build());
        System.out.printf("syncSend1 to topic %s sendResult=%s %n", topic, sendResu
```

For example, topic is `TopicTest` , tag is `TAG1` , then the first parameter to call rocketMQTemplate method will be `TopicTest:TAG1`

Consuming a message

Set the `selectorExpression` field to the corresponding filter tag. In the following code, set `rocketmq.consumer1.subExpression` to `TAG1` to consume the messages of `TAG1` .

```
@Service
@RocketMQMessageListener(
        consumerGroup = "${rocketmq.namespace}%${rocketmq.consumer1.group}",  // Co
        // Use the full name of the topic, which can be either customized or concat
        topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
        selectorExpression = "${rocketmq.consumer1.subExpression}" // Subscription
)
public class Tag1Consumer implements RocketMQListener<String> {

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

Sending and Receiving Delayed Messages

Last updated：2023-04-12 11:41:05

Overview

This document describes how to use Spring Boot Starter to send and receive messages and helps you better understand the message sending and receiving processes.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have learned about the sending and receiving process of general messages.
You have downloaded the demo here or have downloaded one at the GitHub project.

Directions

Sending a message

This process is the same as that of general messages, but you need to pass in the corresponding delay level when calling the sending method.

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

The relationship between the delay level and the delay time

The corresponding relationship between other delay levels and specific delay times is as follows:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1s,	5s,	10s,	30s,	1m,	2m,	3m,	4m,	5m,	6m,	7m,	8m,	9m,	10m,	20m,	30m,	1h,	2h;

Consuming a message

This process is the same as that of general messages. No other actions are required.

Spring Cloud Stream

Last updated：2023-09-12 17:53:17

Overview

This document describes how to use Spring Cloud Stream to send and receive messages and helps you better understand the message sending and receiving processes.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.

You have installed JDK 1.8 or later.

You have installed Maven 2.5 or later.

You have downloaded the demo here or have downloaded one at the GitHub project.

Directions

Step 1. Import dependencies

Import `spring-cloud-starter-stream-rocketmq` -related dependencies in pom.xml. It is recommended to use v2021.0.4.0.

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
    <version>2021.0.4.0</version>
</dependency>
```

Step 2. Add configurations

Add RocketMQ-related configurations to the configuration file.

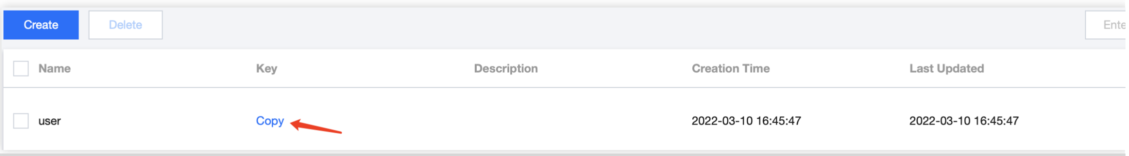
```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          # Full service address
          name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
          # Role name
          secret-key: admin
          # Role token
          access-key: eyJrZXlJZ...
          # Full namespace name
          namespace: rocketmq-xxx|namespace1
          # producer group
          group: producerGroup
        bindings:
          # Channel name, which is the same as the channel name in spring.cloud.str
          Topic-TAG1-Input:
            consumer:
              # Tag type of the subscription, which is configured based on consumer
              subscription: TAG1
          # Channel name
          Topic-TAG2-Input:
            consumer:
              subscription: TAG2
      bindings:
        # Channel name
        Topic-send-Output:
          # Specify a topic, which refers to the one you created
          destination: TopicTest
          content-type: application/json
        # Channel name
        Topic-TAG1-Input:
          destination: TopicTest
          content-type: application/json
          group: consumer-group1
        # Channel name
        Topic-TAG2-Input:
          destination: TopicTest
          content-type: application/json
          group: consumer-group2
```

Note

1. Currently, only `2.2.5-RocketMQ-RC1` and `2.2.5.RocketMQ.RC2` or later versions support **namespace** configuration. If you use other versions, you need to concatenate topic and group names.
The format is as follows:
rocketmq-pngrpmk94d5o|stream%topic (format: namespace name %topic name)
rocketmq-pngrpmk94d5o|stream%group (format: namespace name%group name)
The format for Shared and Exclusive editions is as follows:
MQ_INST_rocketmqpj79obd2ew7v_test%topic (format: namespace name%topic name)
MQ_INST_rocketmqpj79obd2ew7v_test%group (format: namespace name%group name)
2. The subscription configuration item is `subscription` for `2.2.5-RocketMQ-RC1` and `2.2.5.RocketMQ.RC2` and is `tags` for other earlier versions.
The complete configuration items of other versions are as follows:

```
spring:
     cloud:
       stream:
         rocketmq:
           bindings:
             # Channel name, which is the same as the channel name in spring.cloud.
             Topic-test1:
               consumer:
                 # Tag type of the subscription, which is configured based on consu
                 tags: TAG1
             # Channel name
```

```
          Topic-test2:
            consumer:
              tags: TAG2
        binder:
          # Full service address
          name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
          # Role name
          secret-key: admin
          # Role token
          access-key: eyJrZXlJZ...
      bindings:
        # Channel name
        Topic-send:
          # Specify a topic in the format of `cluster ID|namespace name%topic na
          destination: rocketmq-xxx|stream%topic1
          content-type: application/json
          # Name of the group to be used in the format of `cluster ID|namespace
          group: rocketmq-xxx|stream%group1
        # Channel name
        Topic-test1:
          destination: rocketmq-xxx|stream%topic1
          content-type: application/json
          group: rocketmq-xxx|stream%group1
        # Channel name
        Topic-test2:
          destination: rocketmq-xxx|stream%topic1
          content-type: application/json
          group: rocketmq-xxx|stream%group2
```

Parameter	Description
name-server	Cluster access address, which can be copied from **Access Address** in the **Operation** column on t the console. Namespace access addresses in new virtual or exclusive clusters can be copied from t
secret-key	Role name, which can be copied on the Role Management page.
access-key	Role token, which can be copied in the **Token** column on the Role Management page.  Name: user; Key: Copy; Description: ; Creation Time: 2022-03-10 16:45:47; Last Updated: 2022-03-10 16:45:47
namespace	Namespace name, which can be copied on the **Namespace** page in the console.
group	Producer group name, which can be copied under the **Group** tab on the cluster details page.

destination	Topic name, which can be copied on the **Topic** page in the console.

Step 3. Configure channels

You can separately configure input and output channels as needed.

```
/**
 * Custom channel binder
 */
public interface CustomChannelBinder {
    /**
     * (Message producers) send messages
     * Bind the channel name in the configurations
     */
    @Output("Topic-send-Output")
    MessageChannel sendChannel();


    /**
     * (Consumer 1) receives message 1
     * Bind the channel name in the configurations
     */
    @Input("Topic-TAG1-Input")
    MessageChannel testInputChannel1();

    /**
     * (Consumer 2) receives message 2
     * Bind the channel name in the configurations
     */
    @Input("Topic-TAG2-Input")
    MessageChannel testInputChannel2();
}
```

Step 4. Add annotations

Add annotations to the configuration class or startup class. If multiple binders are configured, specify them in the annotations.

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

Step 5. Send messages

1. Inject `CustomChannelBinder` into the class that needs to send messages.

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

2. Use the corresponding output stream channel to send messages.

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

Step 6. Consume messages

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

    /**
     * Listen on the channel configured in the configurations
     *
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG1-Input")
    public void receive(String messageBody) {
        logger.info("Receive1: Messages are received through the stream. messageBod
    }

    /**
     * Listen on the channel configured in the configurations
     *
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG2-Input")
    public void receive2(String messageBody) {
        logger.info("Receive2: Messages are received through the stream. messageBod
    }
}
```

Step 7: Perform local testing

After starting the project locally, you can see from the console that the startup was successful.

You can see that the sending is successful by checking `http://localhost:8080/test-simple` in the browser. Watch the output log of the development IDE.

```
2023-02-23 19:19:00.441  INFO 21958 --- [nio-8080-exec-1] c.t.d.s.controller.Stream
2023-02-23 19:19:01.138  INFO 21958 --- [nsumer-group1_1] c.t.d.s.StreamDemoApplica
```

You can see that a message of TAG1 is sent, and only the subscribers of TAG1 receive the message.

Note

For more information, see GitHub Demo or Spring cloud stream official documentation.

SDK for Java

Sending and Receiving General Messages

Last updated：2023-10-30 10:38:25

Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for Java as an example and helps you better understand the message sending and receiving processes.

Prerequisites

You have created or prepared the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo or obtained the demo in TencentCloud/rocketmq-demo in GitHub.

Directions

Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.

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

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

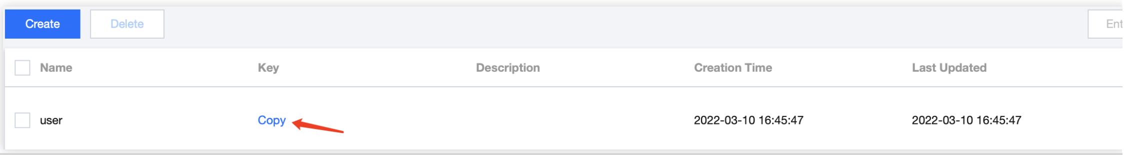
```
    </dependency>
```

Step 2. Produce messages

Creating a message producer

```
// Instantiate the message producer
   DefaultMQProducer producer = new DefaultMQProducer(
       groupName,
       new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
   );
```

```
    // Set the Nameserver address
    producer.setNamesrvAddr(nameserver);
    // Start the producer instance
    producer.start();
```

Parameter	Description
groupName	Producer group name. We recommend that you use the corresponding topic name as the producer
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Ent Name Key Description Creation Time Last Updated user Copy 2022-03-10 16:45:47 2022-03-10 16:45:47
secretKey	Role name, which can be copied on the Role Management page.
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or the console. The namespace access address can be obtained under the **Namespace** tab on the **Cl**

Sending messages

Messages can be sent in the sync, async, or one-way mode.

Sync sending

```
for (int i = 0; i < 10; i++) {
            // Create a message instance and set the topic and message content
            Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getB
            // Send the message
            SendResult sendResult = producer.send(msg);
            System.out.printf("%s%n", sendResult);
       }
```

Parameter	Description

topic_name	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
TAG	A parameter used to set the message tag.

Async sending

```
// Disable retry upon sending failures
      producer.setRetryTimesWhenSendAsyncFailed(0);
      // Set the number of messages to be sent
      int messageCount = 10;
      final CountDownLatch countDownLatch = new CountDownLatch(messageCount);
```

```
        for (int i = 0; i < messageCount; i++) {
            try {
                final int index = i;
                // Create a message instance and set the topic and message content
                Message msg = new Message(topic_name, "TAG", ("Hello rocketMq " + ind
                producer.send(msg, new SendCallback() {
                    @Override
                    public void onSuccess(SendResult sendResult) {
                        // Logic for message sending successes
                        countDownLatch.countDown();
                        System.out.printf("%-10d OK %s %n", index, sendResult.getMsgI
                    }

                    @Override
                    public void onException(Throwable e) {
                        // Logic for message sending failures
                        countDownLatch.countDown();
                        System.out.printf("%-10d Exception %s %n", index, e);
                        e.printStackTrace();
                    }
                });
            } catch (Exception e){
                e.printStackTrace();
            }
        }
        countDownLatch.await(5, TimeUnit.SECONDS);
```

Parameter	Description
topic_name	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
TAG	A parameter used to set the message tag.

One-way sending

```
for (int i = 0; i < 10; i++) {
          // Create a message instance and set the topic and message content
          Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getBy
          // Send one-way messages
          producer.sendOneway(msg);
      }
```

Parameter	Description
topic_name	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.

TAG	A parameter used to set the message tag.

Note

For batch sending and other cases, see TencentCloud/rocketmq-demo in GitHub or the Apache RocketMQ documentation.

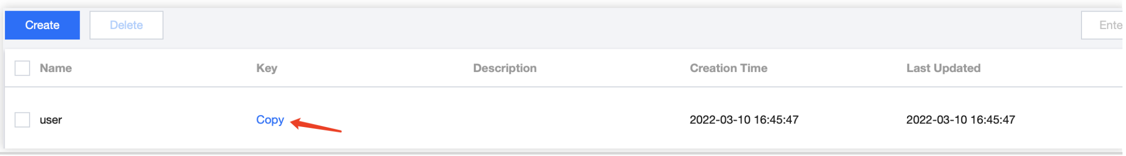
Step 3. Consume messages

Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. The push mode is recommended.

```
// Instantiate the consumer
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // Set the Nameserver address
        pushConsumer.setNamesrvAddr(nameserver);
```

Parameter	Description
groupName	Consumer group name, which can be copied under the **Group** tab on the **Cluster** page in the cons

nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or the console. The namespace access address can be obtained under the **Namespace** tab on the **Cl**
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Ente Name | Key | Description | Creation Time | Last Updated user | Copy | | 2022-03-10 16:45:47 | 2022-03-10 16:45:47

Subscribing to messages

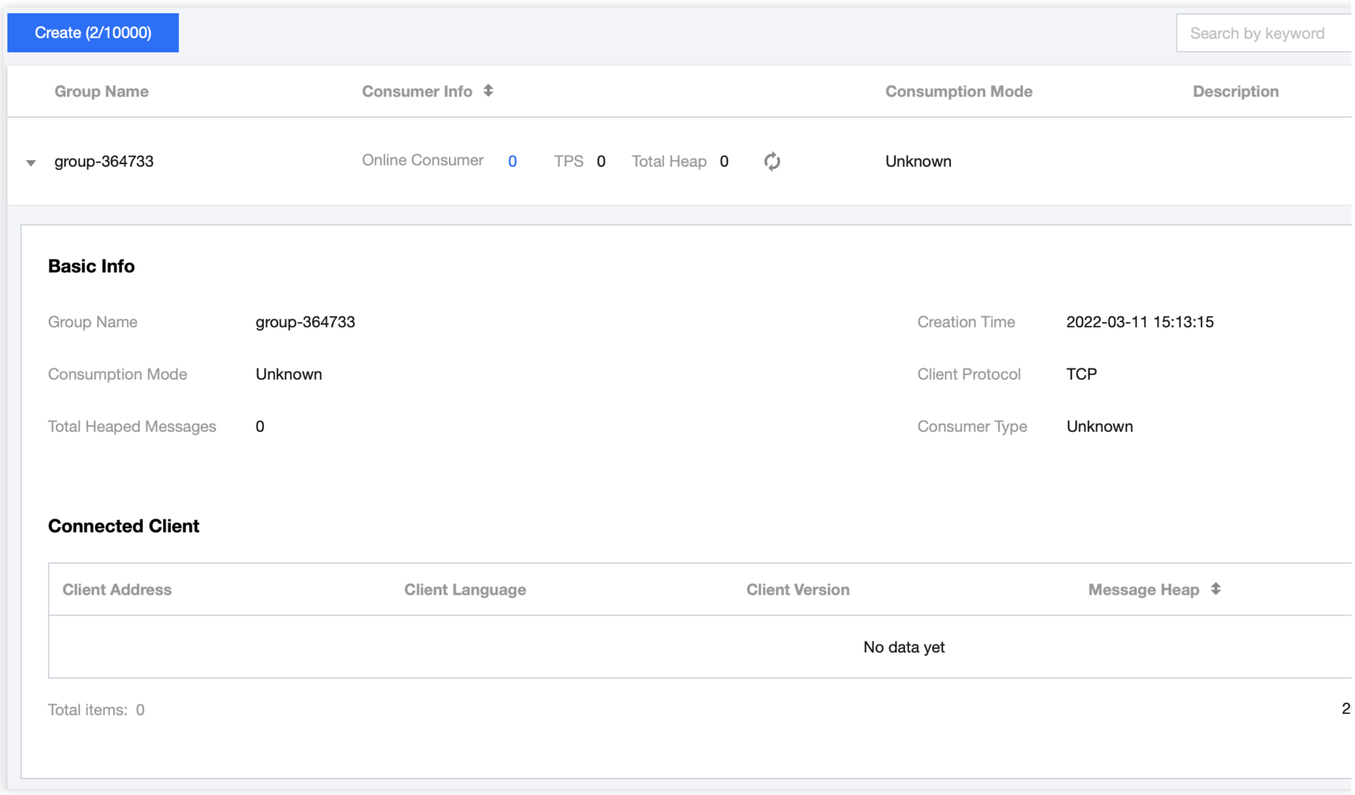
The subscription modes vary by consumption mode.

```
// Subscribe to a topic
        pushConsumer.subscribe(topic_name, "*");
        // Register a callback implementation class to process messages pulled from t
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
            // Message processing logic
            System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
            // Mark the message as being successfully consumed and return the consump
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        // Start the consumer instance
        pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Step 4. View consumption details

Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the consumer group. Click **Consumer Details** in the **Operation** column to view consumer details.



Group Name	Consumer Info	Consumption Mode	Description
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Connected Client

Client Address	Client Language	Client Version	Message Heap
No data yet			

Total items: 0

Note

Above is a brief introduction to message publishing and subscription. For more information, see TencentCloud/rocketmq-demo or the Apache RocketMQ documentation.

Sending and Receiving Delayed Messages

Last updated：2023-05-16 11:07:52

Overview

This document describes how to use open-source SDK to send and receive timed messages by using the SDK for Java as an example.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo here or have downloaded one at the GitHub project.

Directions

Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.

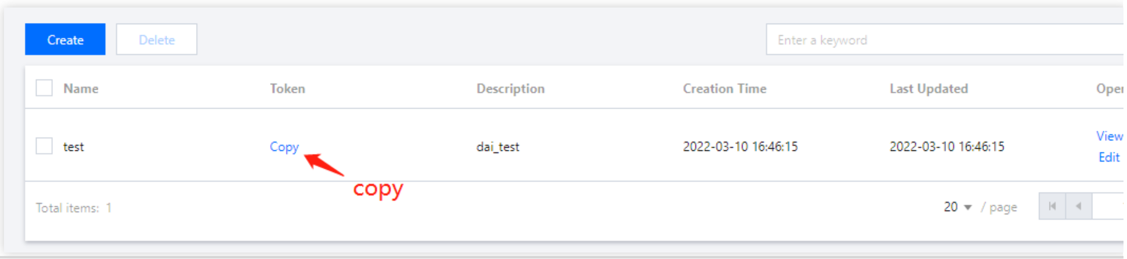
```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.4</version>
  </dependency>

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

```
    </dependency>
```

Step 2. Produce messages

Creating a message producer

```
// Instantiate the message producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
    );
    // Set the Nameserver address
    producer.setNamesrvAddr(nameserver);
    // Start the producer instance
    producer.start();
```

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Name | Token | Description | Creation Time | Last Updated | Oper test | Copy | dai_test | 2022-03-10 16:46:15 | 2022-03-10 16:46:15 | View Edit Total items: 1 20 / page

Sending a message

Messages with fixed delay level

```
}
```

```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message " + i).getB
    // Set message delay level
    message.setDelayTimeLevel(5);
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

Messages with random delay time

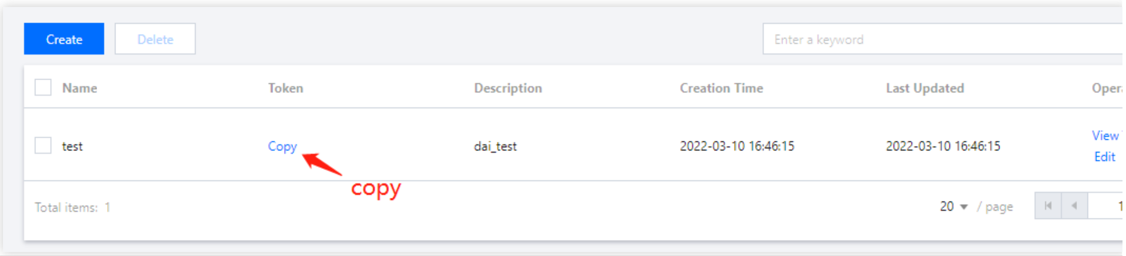
```
int totalMessagesToSend = 1;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello timer message " + i).getBytes
    // Set the time for sending the message
    long timeStamp = System.currentTimeMillis() + 30000;
    // To send a timed message, you need to specify a time for it, and the message
    // If the timestamp is set before the current time, the message will be deliver
    // Set `__STARTDELIVERTIME` into the property of `msg`
    message.putUserProperty("__STARTDELIVERTIME", String.valueOf(timeStamp));
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

Step 3. Consume messages

####Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.

```
// Instantiate the consumer
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // Set the Nameserver address
        pushConsumer.setNamesrvAddr(nameserver);
```

Parameter	Description
groupName	Producer group name, which can be copied under the **Group** tab on the **Cluster** page in the consol
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page. 

Create Delete Enter a keyword

Name	Token	Description	Creation Time	Last Updated	Oper
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View Edit

Total items: 1 20 / page 1

Subscribing to messages

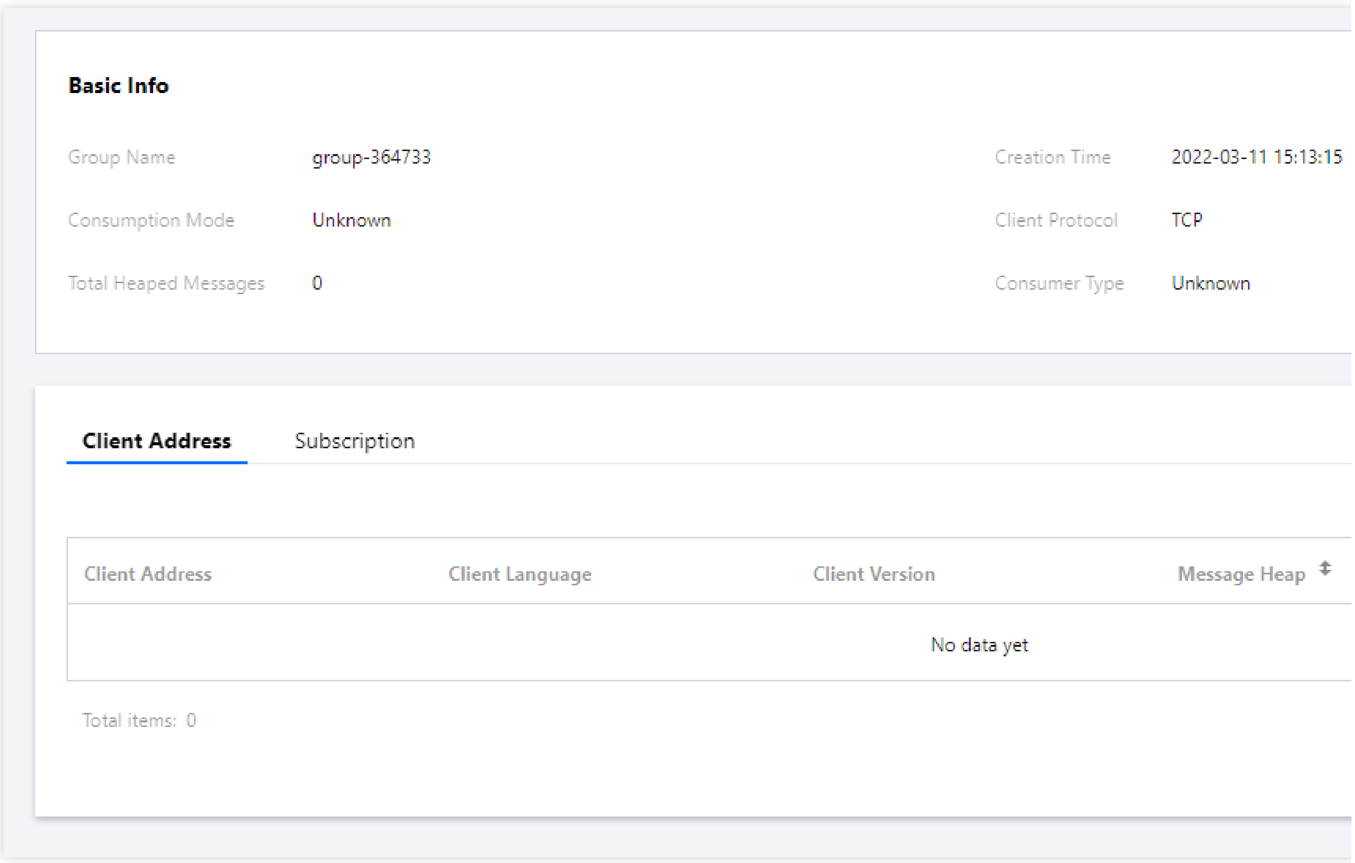
The subscription modes vary by consumption mode.

```
// Subscribe to a topic
      pushConsumer.subscribe(topic_name, "*");
      // Register a callback implementation class to process messages pulled from t
      pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
          // Message processing logic
          System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
          // Mark the message as being successfully consumed and return the consump
          return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
      });
      // Start the consumer instance
      pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Step 4. View consumption details

Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.

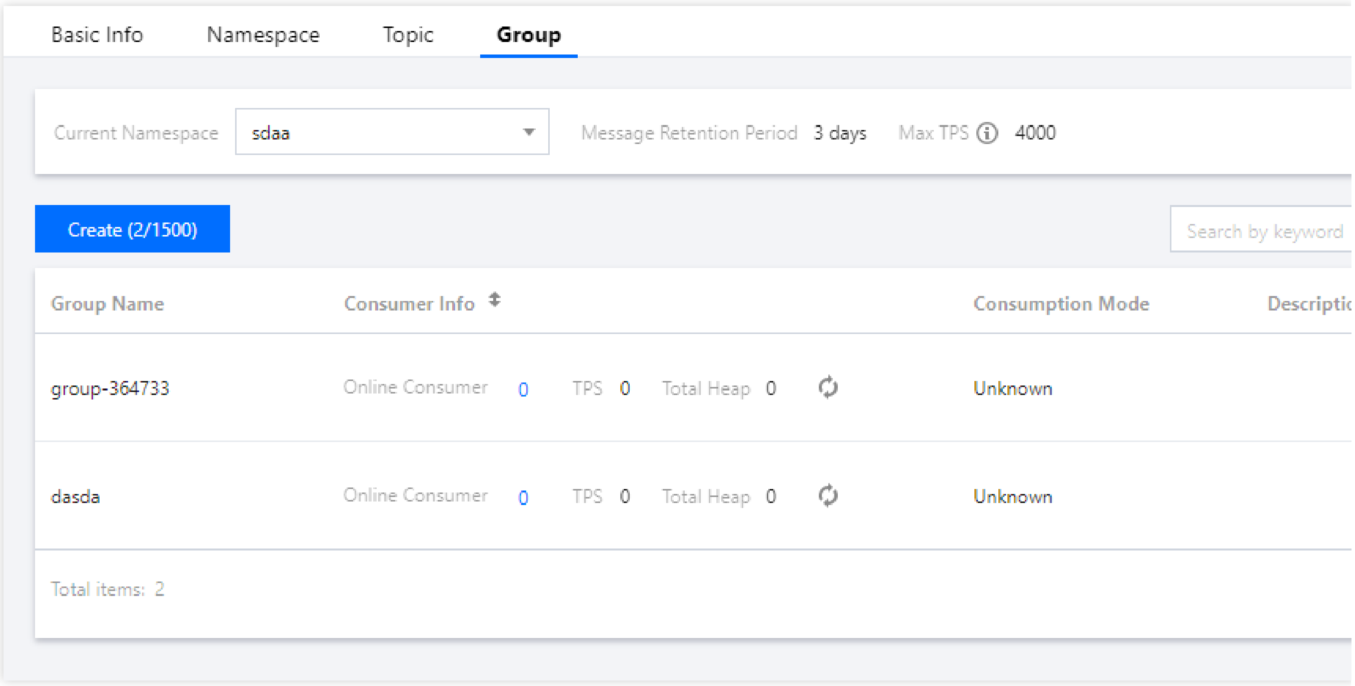


Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address	Client Language	Client Version	Message Heap
No data yet			

Total items: 0



Current Namespace sdaa Message Retention Period 3 days Max TPS 4000

Group Name	Consumer Info	Consumption Mode	Descriptic
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

Total items: 2

Note

Above is a brief introduction to message publishing and subscription. For more information, see Demo or RocketMQ documentation.

Sending and Receiving Sequential Messages

Last updated：2023-05-16 11:07:52

Overview

This document describes how to use open-source SDK to send and receive timed messages by using the SDK for Java as an example.

Prerequisites

You have created the required resources. If it is a globally sequential message, you need to create a single-queue topic. For more information, see Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo here or have downloaded one at the GitHub project.

Directions

Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.
Note
The dependency version must be v4.9.3 or later, preferably v4.9.4.

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

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

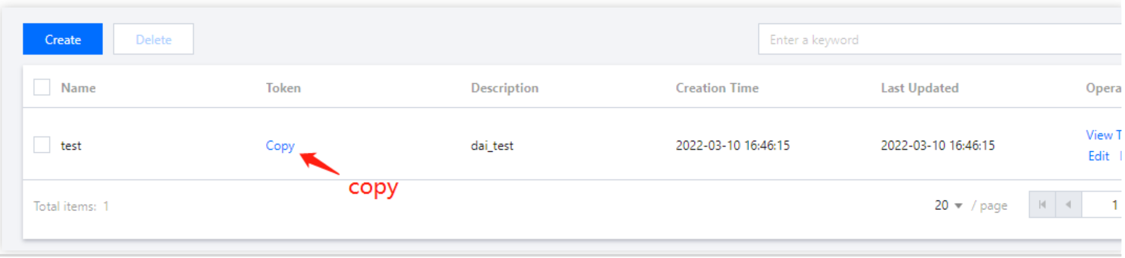
```
    </dependency>
```

Step 2. Produce messages

Creating a message producer

```
// Instantiate the message producer
   DefaultMQProducer producer = new DefaultMQProducer(
       groupName,
       new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
   );
```

```
    // Set the Nameserver address
    producer.setNamesrvAddr(nameserver);
    // Start the producer instance
    producer.start();
```

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste from the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page. 

Name	Token	Description	Creation Time	Last Updated	Opera
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View T Edit

Sending a message

Globally sequential message

This process is the same as that of general messages.

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

Partitionally sequential message

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

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

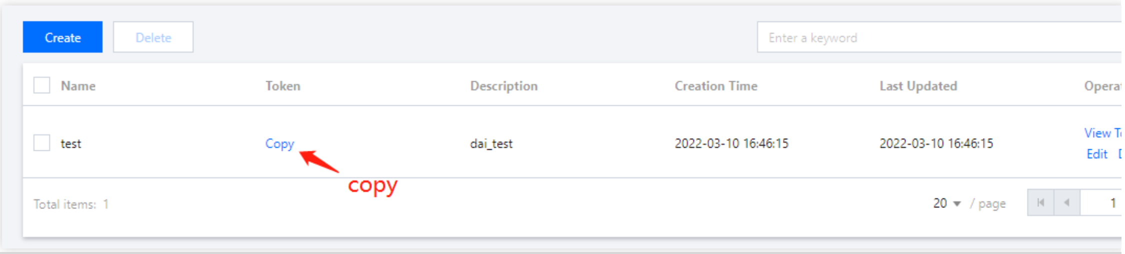
Step 3. Consume messages

####Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.

```
// Instantiate the consumer
```

```
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // Set the Nameserver address
        pushConsumer.setNamesrvAddr(nameserver);
```

Parameter	Description
groupName	Producer group name, which can be copied under the **Group** tab on the **Cluster** page in the consol
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste from the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page. 

Create Delete Enter a keyword

Name	Token	Description	Creation Time	Last Updated	Opera
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View T Edit

Total items: 1 20 / page 1

Subscribing to messages

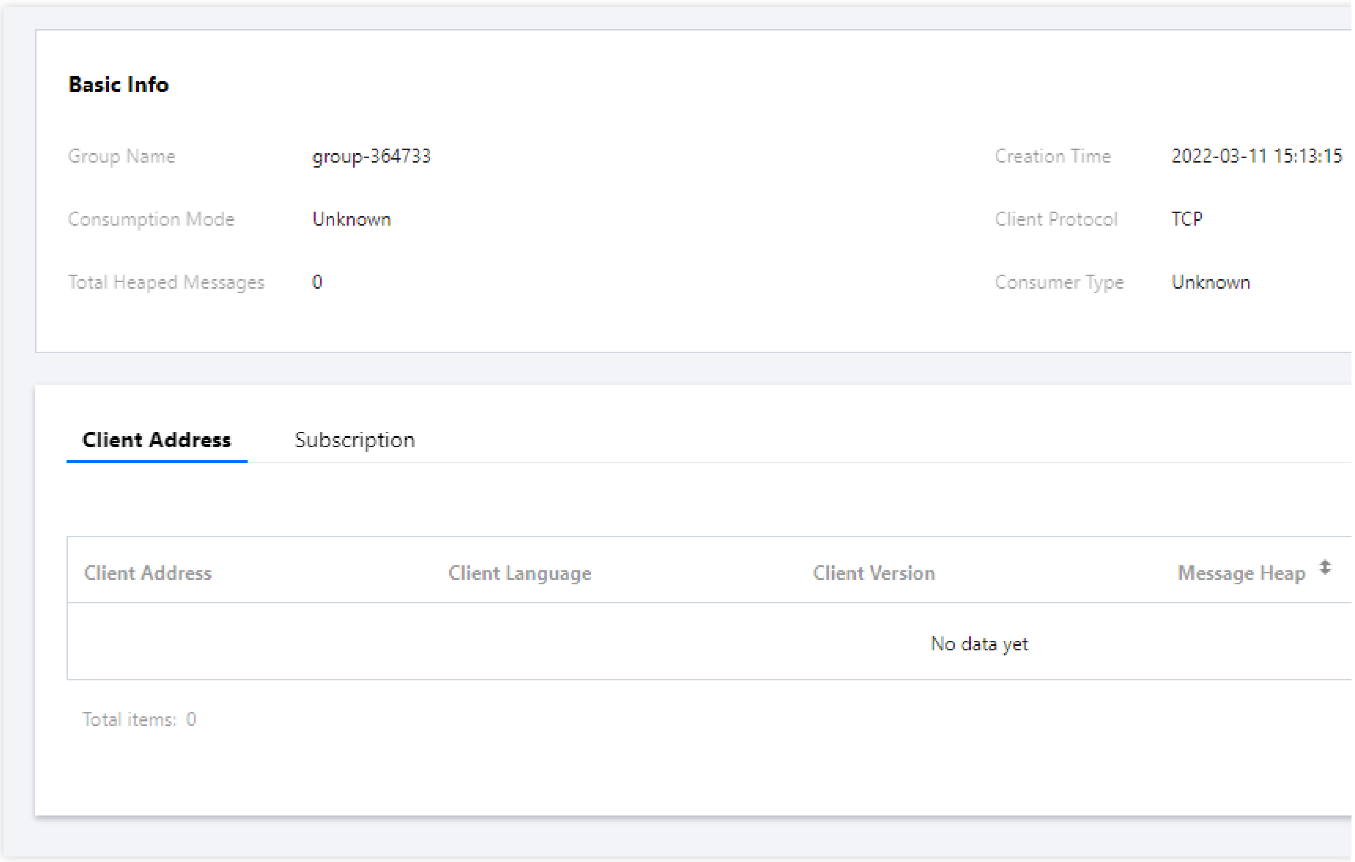
The subscription modes vary by consumption mode.

```
// Subscribe to a topic
      pushConsumer.subscribe(topic_name, "*");
      // Register a callback implementation class to process messages pulled from t
      pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
          // Message processing logic
          System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
          // Mark the message as being successfully consumed and return the consump
          return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
      });
      // Start the consumer instance
      pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Step 4. View consumption details

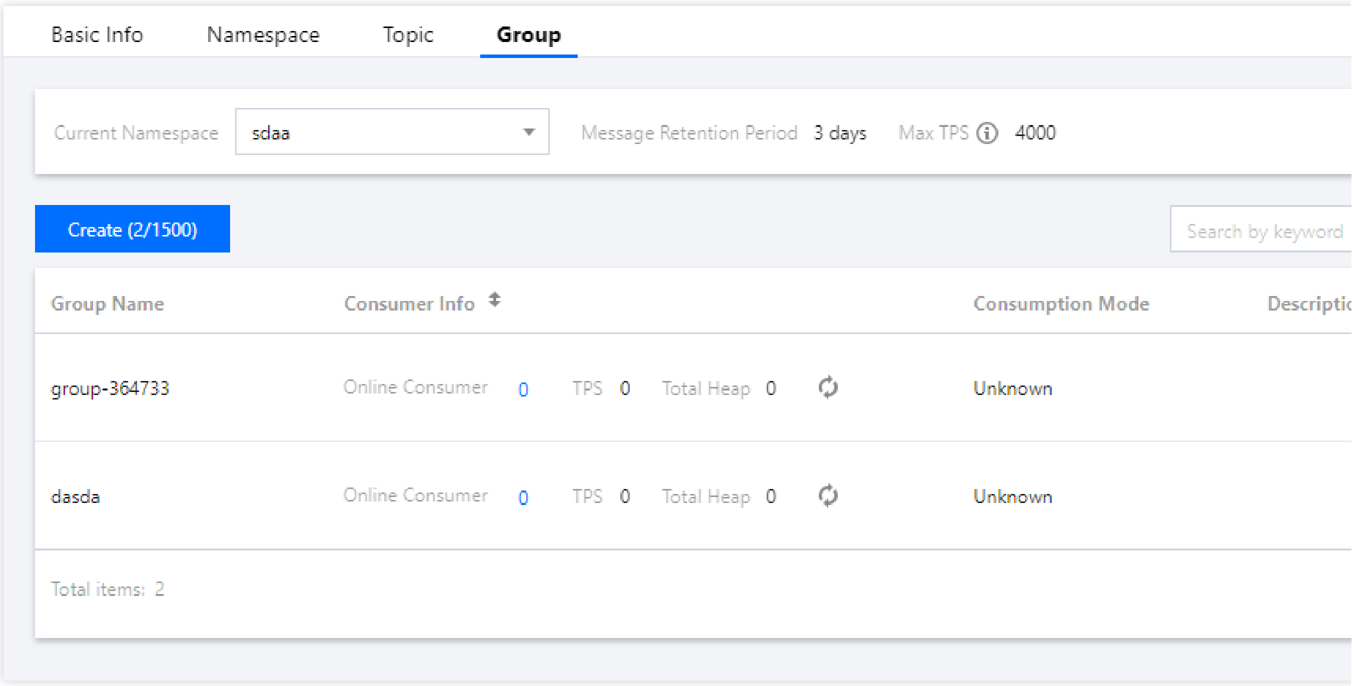
Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.



Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address	Client Language	Client Version	Message Heap
No data yet			

Total items: 0



Basic Info | Namespace | Topic | **Group**

Current Namespace: sdaa | Message Retention Period: 3 days | Max TPS: 4000

Create (2/1500) | Search by keyword

Group Name	Consumer Info	Consumption Mode	Descriptic
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

Total items: 2

Note

Above is a brief introduction to message publishing and subscription. For more information, see Demo or RocketMQ documentation.

Sending and Receiving Transactional Messages

Last updated：2023-05-16 11:07:52

Overview

This document describes how to use open-source SDK to send and receive transactional messages by using the SDK for Java as an example.

Prerequisites

You have created the required resources. If it is a globally sequential message, you need to create a single-queue topic. For more information, see Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo here or have downloaded one at the GitHub project.

Directions

Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.
Note
The dependency version must be v4.9.3 or later, preferably v4.9.4.

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

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

```
    </dependency>
```

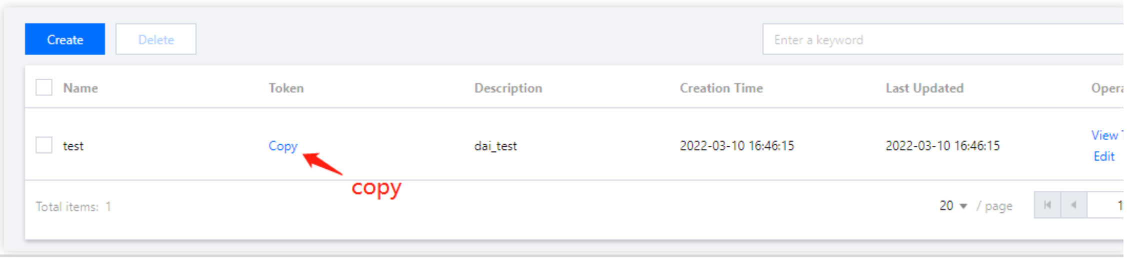
Step 2. Produce messages

Implementing TransactionListener

```
public class TransactionListenerImpl implements TransactionListener {

    // After the half message is sent successfully, call back this method to execut
    @Override
    public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
      // Execute the database transaction here. If the execution is successful, it
      return LocalTransactionState.UNKNOW;
    }
    // Check back local transaction
    @Override
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {
       // Here query the data status of the local database, and then decide whether
       return LocalTransactionState.COMMIT_MESSAGE;
    }

}
```

Creating a message producer

```
//Users need to inplement a TransactionListener instance,
TransactionListener transactionListener = new TransactionListenerImpl();
// Instantiate a transactional message producer
ProducerTransactionMQProducer producer = new TransactionMQProducer("transaction_gro
// ACL permission
new AclClientRPCHook(new SessionCredentials(ClientCreater.ACCESS_KEY, ClientCreater
// Set the Nameserver address
producer.setNamesrvAddr(ClientCreater.NAMESERVER);
producer.setTransactionListener(transactionListener);
producer.start();
```

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste from the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Name / Token / Description / Creation Time / Last Updated / Opera test / Copy / dai_test / 2022-03-10 16:46:15 / 2022-03-10 16:46:15 / View Edit Total items: 1 20 / page 1

Sending a message

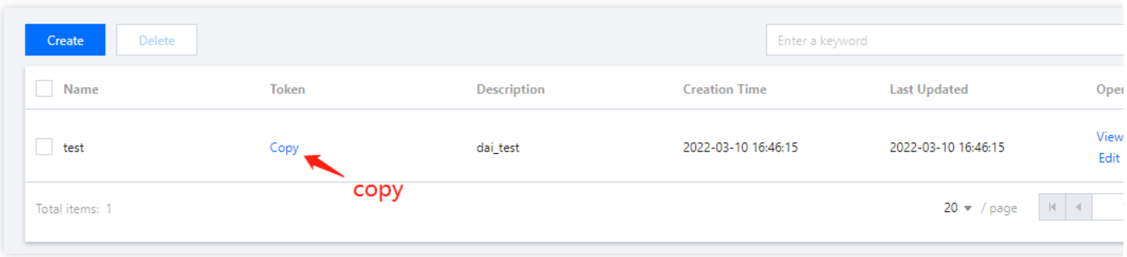
```
for (int i = 0; i < 3; i++) {
    // Construct message instance
    Message msg = new Message(TOPIC_NAME, "your tag", "KEY" + i,("Hello RocketMQ "
    SendResult sendResult = producer.sendMessageInTransaction(msg,null);
     System.out.printf("%s%n", sendResult);
}
```

Step 3. Consume messages

####Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.

```
// Instantiate the consumer
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // Set the Nameserver address
        pushConsumer.setNamesrvAddr(nameserver);
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, c
           // Message processing logic
           System.out.printf("%s Receive transaction messages: %s %n", Thread.curre
           // Mark that the message has been successfully consumed
           return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
           });
```

Parameter	Description
groupName	Producer group name, which can be copied under the **Group** tab on the **Cluster** page in the consol
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page. 

Create Delete Enter a keyword

Name	Token	Description	Creation Time	Last Updated	Oper
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View Edit

Total items: 1 20 / page

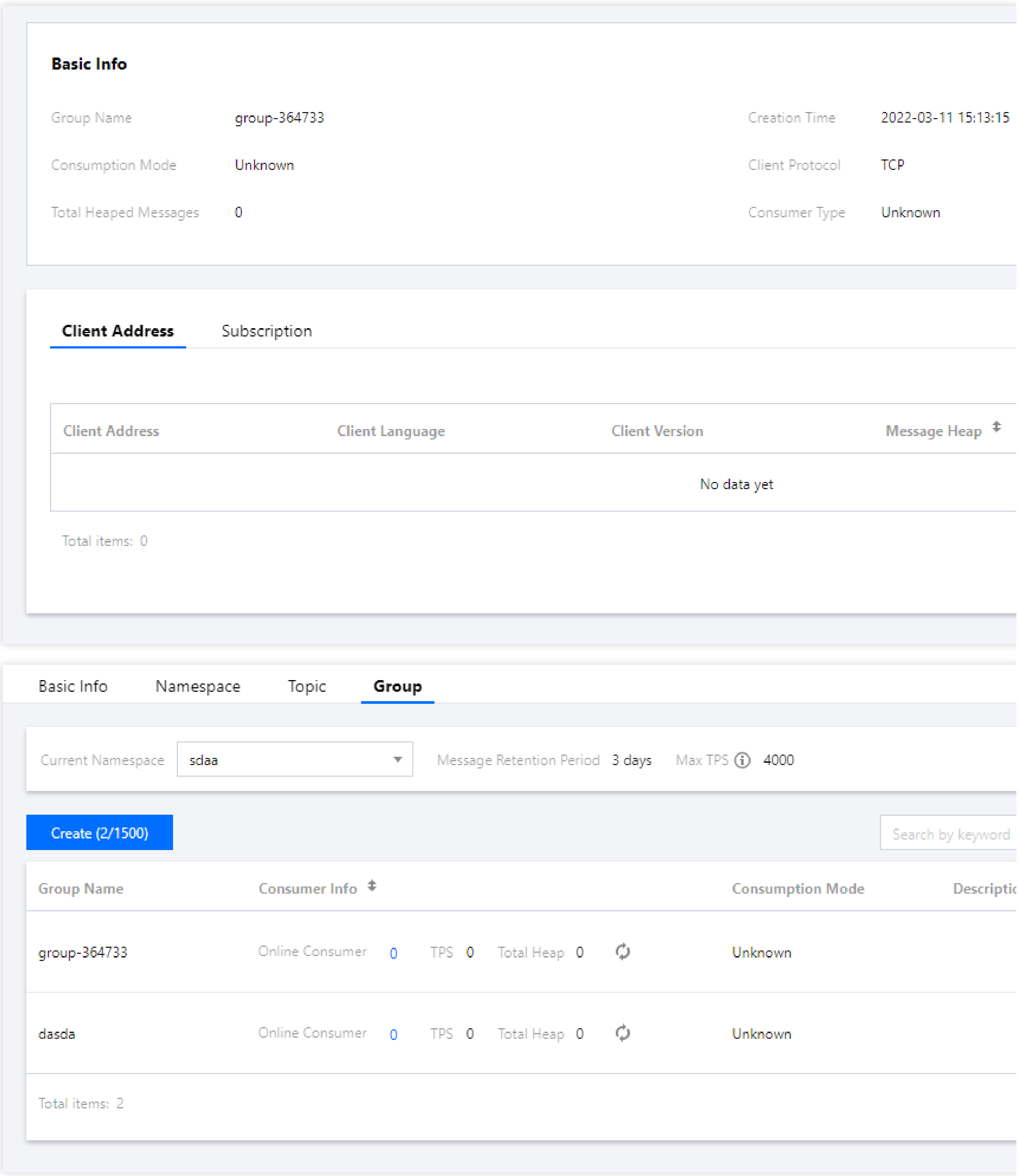
Subscribing to messages

The subscription modes vary by consumption mode.

```
// Subscribe to a topic
        pushConsumer.subscribe(topic_name, "*");
        // Register a callback implementation class to process messages pulled from t
        pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
            // Message processing logic
            System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
            // Mark the message as being successfully consumed and return the consump
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        // Start the consumer instance
        pushConsumer.start();
```

Step 4. View consumption details

Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.



Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address	Client Language	Client Version	Message Heap
		No data yet	

Total items: 0

Current Namespace sdaa Message Retention Period 3 days Max TPS 4000

Group Name	Consumer Info	Consumption Mode	Descriptic
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

Total items: 2

Note

Above is a brief introduction to message publishing and subscription. For more information, see Demo or RocketMQ documentation.

Sending and Receiving Filtered Messages

Last updated：2023-03-28 10:15:45

Overview

This document describes how to use open-source SDK to send and receive filtered messages by using the SDK for Java as an example. You can do so with tags or SQL expressions.

Prerequisites

You have created the required resources. If it is a globally sequential message, you need to create a single-queue topic. For more information, see Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo here or have downloaded one at the GitHub project.
You have learned about the sending and receiving processes of general messages.

Tag-based option

The main code of creating producer and consumer is basically same as that for general messages.
For message production, a message need to be carried with a or more tags when constructing the message body.
For message consumption, a message need to be carried with a tag, an asterisk (*), or multiple tag expressions when being subscribed to.

Step 1. Produce messages

Sending messages

The main code of sending messages is basically same as that for general messages. However, a message is allowed to carry only a tag when constructing the message body.

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

Step 2. Consume messages

Subscribing to messages

```
// Subscribe to all tags when subscribing to a topic
pushConsumer.subscribe(topic_name, "*");

//Subscribe to the specified tags
//pushConsumer.subscribe(TOPIC_NAME, "Tag1");

// Subscribe to multiple tags
//pushConsumer.subscribe(TOPIC_NAME, "Tag1||Tag2");
```

```
// Register a callback implementation class to process messages pulled from the bro
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
          // Message processing logic
          System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
          // Mark the message as being successfully consumed and return the consump
          return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Note

Above is a brief introduction to message publishing and subscription. For more information, see GitHub Demo or official RocketMQ documentation.

SQL expression-based option

The main code of creating producer and consumer is basically same as that for general messages.
For message production, a message need to be carried with user-defined properties when constructing the message body.
For message consumption, a message need to be carried with corresponding SQL expression when being subscribed to.

Step 1. Produce messages

The main code of sending messages is basically same as that for general messages. However, a message is allowed to carry multiple user-defined properties when constructing the message body.

}

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

Step 2. Consume messages

The main code of consuming messages is basically same as that for general messages. However, a message need to be carried with corresponding SQL expression when being subscribed to.

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

// Subscribe to single-key SQL expression when subscribing to a topic
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND

//Subscribe to multiple properties
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND
// Register a callback implementation class to process messages pulled from the bro
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
          // Message processing logic
          System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
          // Mark the message as being successfully consumed and return the consump
          return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

Note

Above is a brief introduction to message publishing and subscription. For more information, see GitHub Demo or official RocketMQ documentation.

Sending and Receiving Broadcast Messages

Last updated：2023-05-16 11:07:52

Overview

This document describes how to use open-source SDK to send and receive broadcast messages by using the SDK for Java as an example.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo here or have downloaded one at the GitHub project.

Directions

Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.

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

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

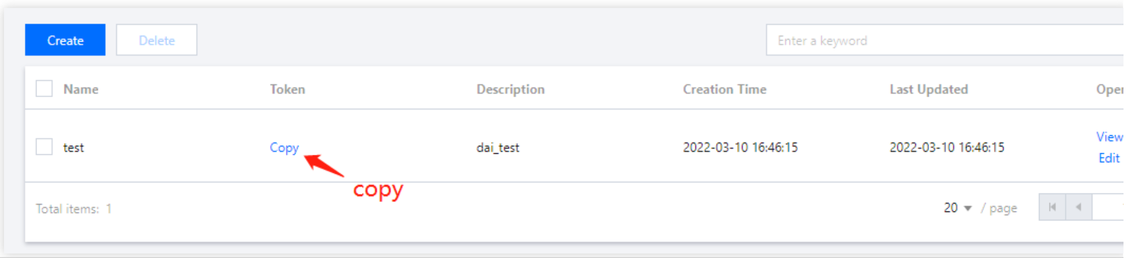
```
    </dependency>
```

Step 2. Produce messages

Creating a message producer

```
// Instantiate the message producer
   DefaultMQProducer producer = new DefaultMQProducer(
       groupName,
       new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
   );
```

```
    // Set the Nameserver address
    producer.setNamesrvAddr(nameserver);
    // Start the producer instance
    producer.start();
```

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Enter a keyword; Name / Token / Description / Creation Time / Last Updated / Oper; test / Copy / dai_test / 2022-03-10 16:46:15 / 2022-03-10 16:46:15 / View Edit; copy; Total items: 1; 20 / page

Sending a message

This process is the same as that of general messages. Broadcast messages reflect the behavior of consumers.

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

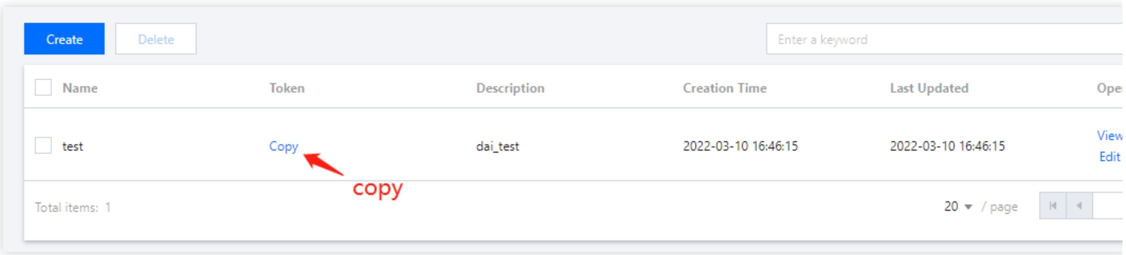
Step 3. Consume messages

####Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.

```
// Instantiate the consumer
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // Set the Nameserver address
        pushConsumer.setNamesrvAddr(nameserver);
```

Parameter	Description

groupName	Producer group name, which can be copied under the **Group** tab on the **Cluster** page in the consol
nameserver	Cluster access address, which can be obtained from **Access Address** in the **Operation** column or **Management** page in the console. Namespace access addresses in new virtual or exclusive cluste the **Namespace** list.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Name | Token | Description | Creation Time | Last Updated | Ope; test | Copy | dai_test | 2022-03-10 16:46:15 | 2022-03-10 16:46:15 | View Edit; Total items: 1

Subscribing to messages

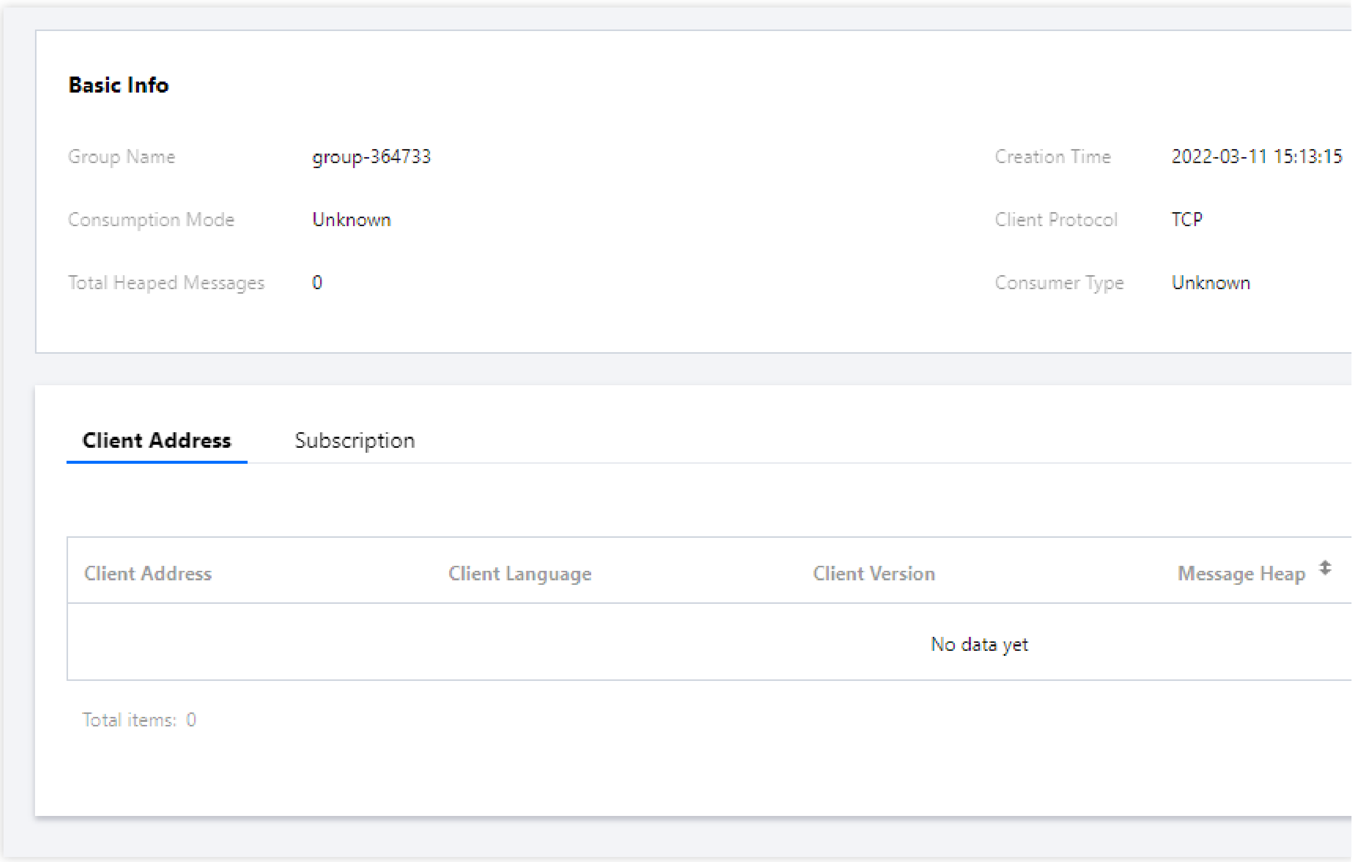
This process requires setting consumption mode.

```
// Set broadcast consumption mode
pushConsumer.setMessageModel(MessageModel.BROADCASTING);
// Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
// Register a callback implementation class to process messages pulled from the bro
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
   // Message processing logic
   System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().getNa
   // Mark the message as being successfully consumed and return the consumption st
   return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
```

```
// Start the consumer instance
pushConsumer.start();
```

Step 4. View consumption details

Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.

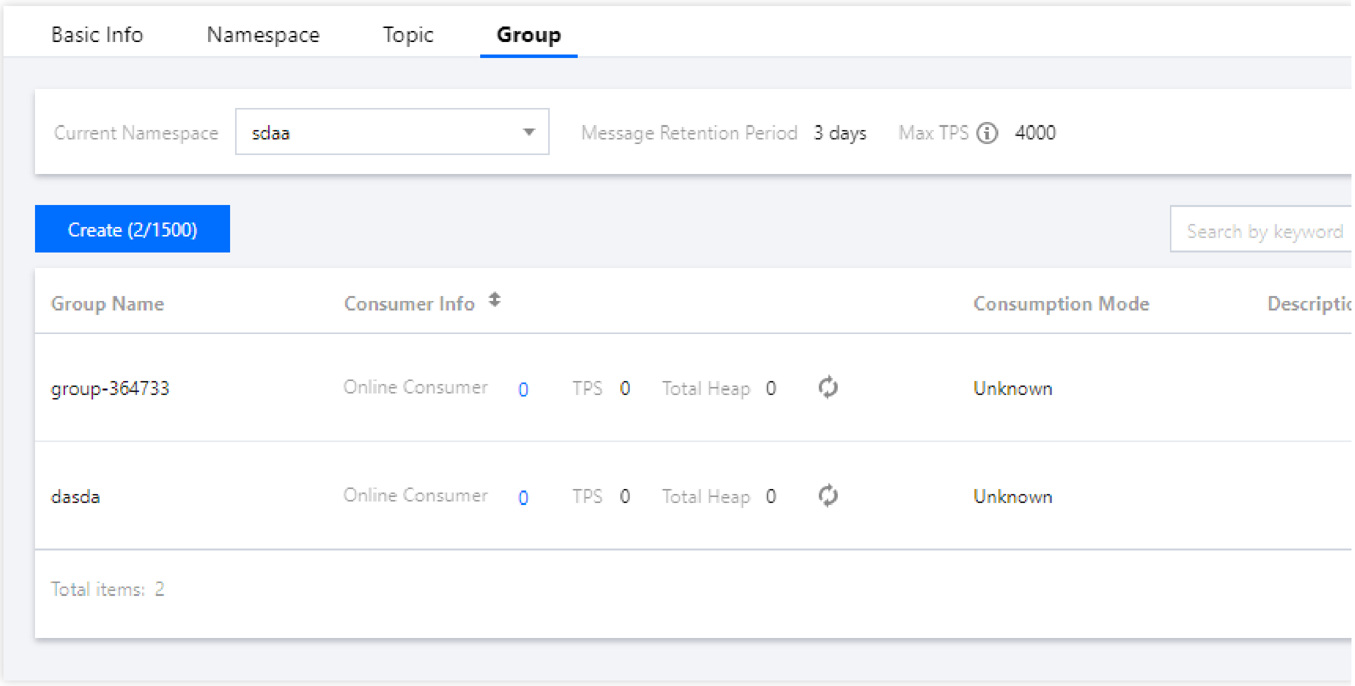


Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address	Client Language	Client Version	Message Heap
No data yet			

Total items: 0



Current Namespace sdaa Message Retention Period 3 days Max TPS 4000

Group Name	Consumer Info	Consumption Mode	Descriptic
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

Total items: 2

Note

Above is a brief introduction to message publishing and subscription. For more information, see Demo or RocketMQ documentation.

SDK for C++

Last updated：2023-05-16 11:07:52

Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for C++ as an example and helps you better understand the message sending and receiving processes.

Prerequisites

You have installed GCC.

You have downloaded the demo.

Directions

1. Prepare the environment.

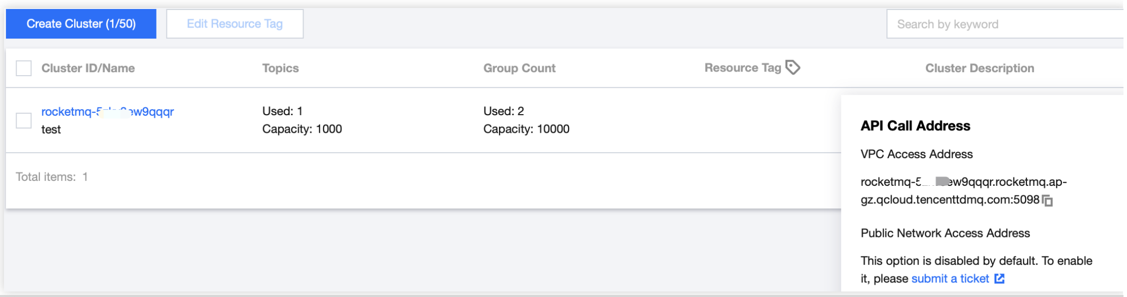
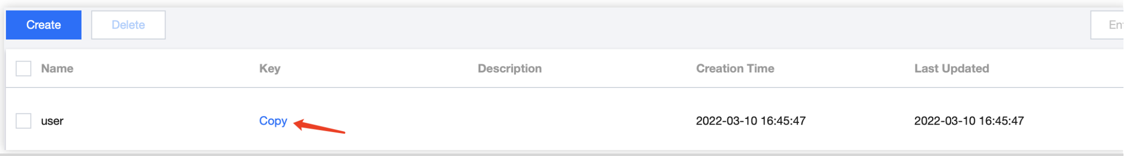
1.1 Install RocketMQ-Client-CPP in the client environment as instructed in the official documentation. **The master branch is recommended**.

1.2 Import the header files and dynamic libraries related to RocketMQ-Client-CPP to the project.

2. Instantiate the message producer.

```
// Set the producer group name
DefaultMQProducer producer(groupName);
// Set the service access address
producer.setNamesrvAddr(nameserver);
// Set user permissions
producer.setSessionCredentials(
    accessKey,  // Role token
    secretKey, // Role name
    "");
// Set the full namespace name
producer.setNameSpace(namespace);
```

```
// Make sure all parameters are configured before the start
producer.start();
```

Parameter	Description
groupName	Producer group name, which can be copied under the **Group** tab on the **Cluster** page in the consol
nameserver	Cluster access address, which can be obtained in the **Operation** column on the **Cluster Managem** Namespace access addresses in new virtual or exclusive clusters can be copied from the **Namesp**  Create Cluster (1/50) · Edit Resource Tag · Search by keyword · Cluster ID/Name · Topics · Group Count · Resource Tag · Cluster Description · rocketmq-[redacted]w9qqqr test · Used: 1 Capacity: 1000 · Used: 2 Capacity: 10000 · Total items: 1 · API Call Address · VPC Access Address · rocketmq-[redacted]w9qqqr.rocketmq.ap-gz.qcloud.tencenttdmq.com:5098 · Public Network Access Address · This option is disabled by default. To enable it, please submit a ticket
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create · Delete · En · Name · Key · Description · Creation Time · Last Updated · user · Copy · 2022-03-10 16:45:47 · 2022-03-10 16:45:47
namespace	Namespace name, which can be copied on the **Namespace** page in the console.

3. Send a message.

```
// Initialize message content
   MQMessage msg(
       topicName,  // Topic name
       TAGS,        // Message tag
       KEYS,        // Message key
       "Hello cpp client, this is a message."   // Message content
   );

   try {
       // Send the message
       SendResult sendResult = producer.send(msg);
```

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

Parameter	Description
topicName	Topic name, which can be copied on the **Topic** page in the console.
TAGS	A parameter used to set the message tag.
KEYS	A parameter used to set the message key.

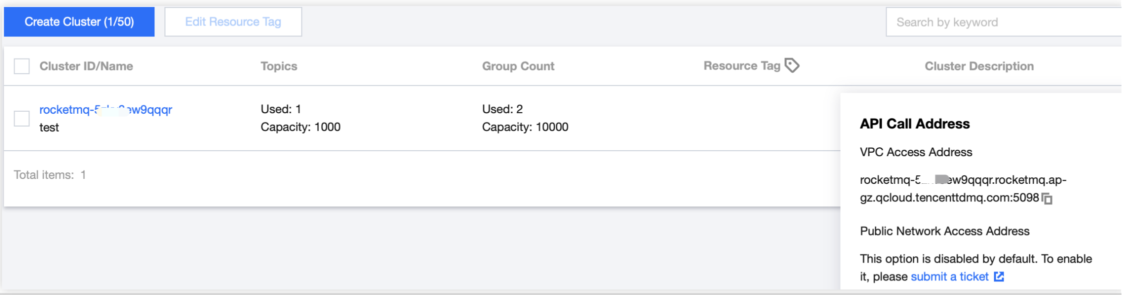
4. Release the resource.

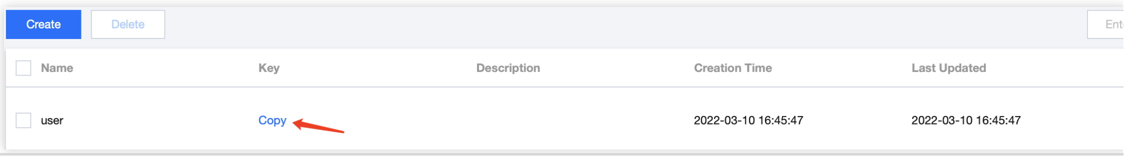
```
// Release resources
   producer.shutdown();
```

5. Initialize the consumer.

```
// Listen on messages
   class ExampleMessageListener : public MessageListenerConcurrently {
   public:
       ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
           for (auto item = msgs.begin(); item != msgs.end(); item++) {
               // Business
               std::cout << "Received Message Topic:" << item->getTopic() << ", Msg
                         << item->getTags() << ", KEYS:" << item->getKeys() << ", B
           }
           // Return CONSUME_SUCCESS if the consumption is successful
           return CONSUME_SUCCESS;
```

```
            // Return RECONSUME_LATER if the consumption failed. The message will be
            // return RECONSUME_LATER;
        }
    };

    // Initialize the consumer
    DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer(groupName);
    // Set the service address
    consumer->setNamesrvAddr(nameserver);
    // Set user permissions
    consumer->setSessionCredentials(
        accessKey,
        secretKey,
        "");
    // Set the namespace
    consumer->setNameSpace(namespace);
    // Set the instance name
    consumer->setInstanceName("CppClient");

    // Register a custom listener function to process the received messages and retu
    ExampleMessageListener *messageListener = new ExampleMessageListener();
    // Subscribe to the message
    consumer->subscribe(topicName, TAGS);
    // Set the message listener
    consumer->registerMessageListener(messageListener);

    // After the preparations, you must call the start function before the consumpti
    consumer->start();
```

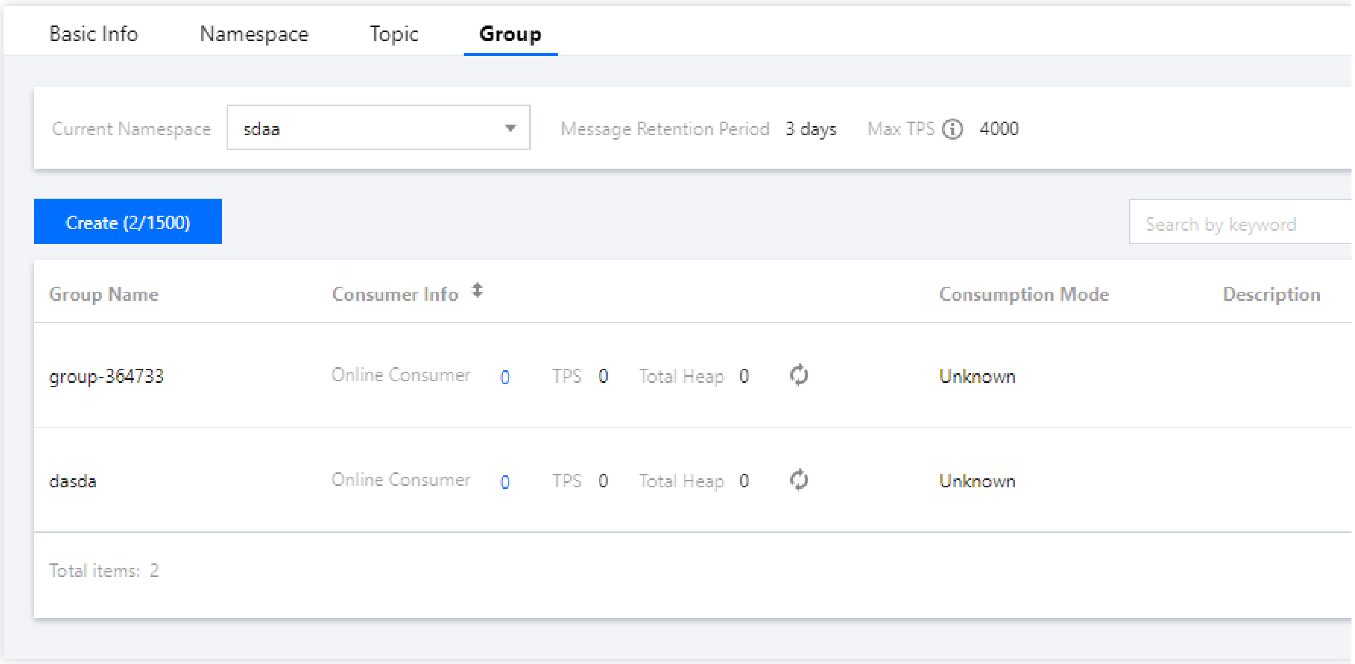
Parameter	Description
groupName	Consumer group name, which can be obtained under the **Group** tab on the cluster details page in th
nameserver	Cluster access address, which can be obtained in the **Operation** column on the **Cluster Managem** Namespace access addresses in new virtual or exclusive clusters can be copied from the **Namespa**  Create Cluster (1/50) Edit Resource Tag Search by keyword Cluster ID/Name Topics Group Count Resource Tag Cluster Description rocketmq-[redacted]w9qqqr test Used: 1 Capacity: 1000 Used: 2 Capacity: 10000 Total items: 1 **API Call Address** VPC Access Address rocketmq-[redacted]ew9qqqr.rocketmq.ap-gz.qcloud.tencenttdmq.com:5098 Public Network Access Address This option is disabled by default. To enable it, please submit a ticket
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.

	 Create Delete Name Key Description Creation Time Last Updated user Copy 2022-03-10 16:45:47 2022-03-10 16:45:47
namespace	Namespace name, which can be copied on the **Namespace** page in the console.
topicName	Topic name, which can be copied on the **Topic** page in the console.
TAGS	A parameter used to set the message tag.

6. Release the resource.

```
// Release resources
consumer->shutdown();
```

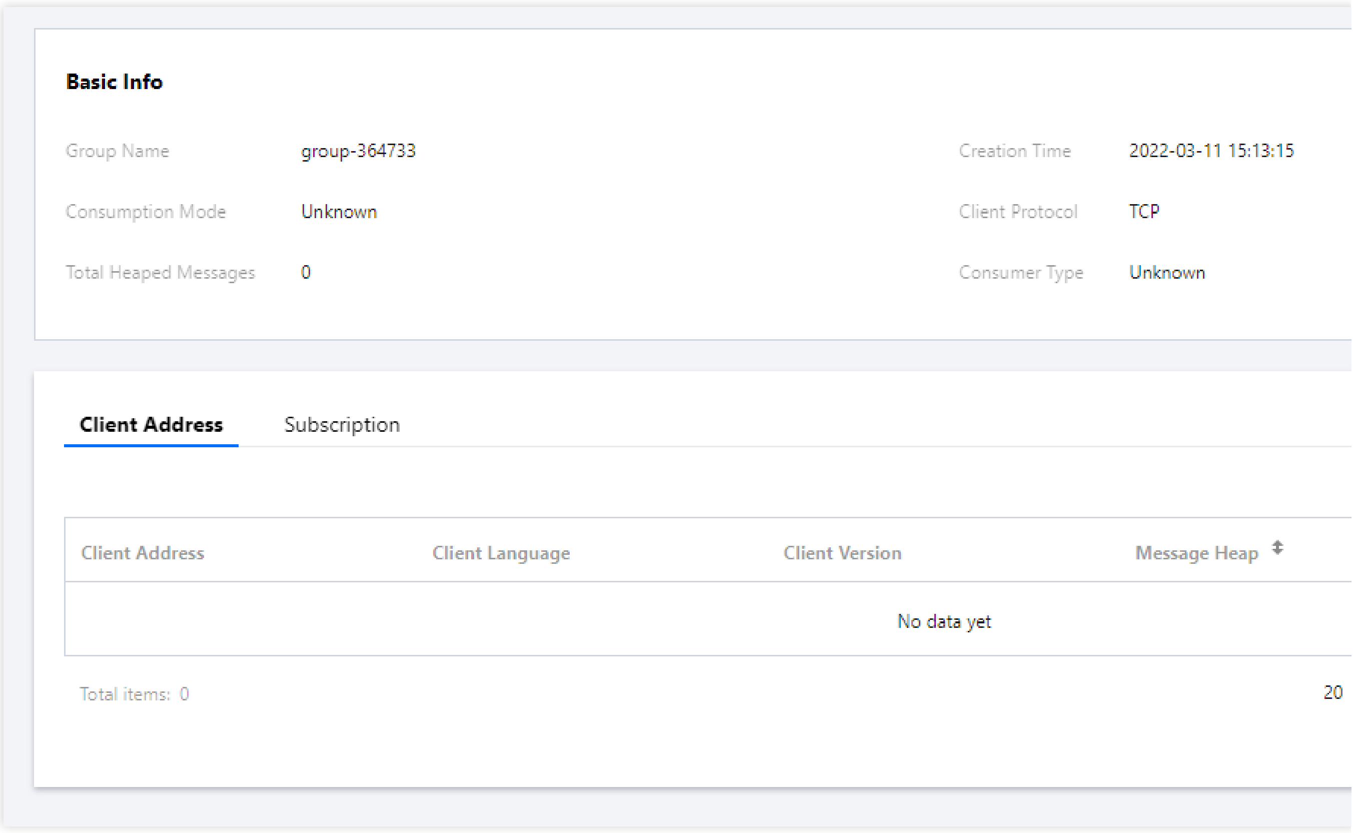
7. View consumer details. Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.



Current Namespace sdaa Message Retention Period 3 days Max TPS 4000

Group Name	Consumer Info	Consumption Mode	Description
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

Total items: 2



Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address	Client Language	Client Version	Message Heap
No data yet			

Total items: 0 20

Note

Above is a brief introduction to message publishing and subscription. Above is a brief introduction to message publishing and For more information, see Demo or RocketMQ-Client-CPP Example.

SDK for Go

Last updated：2023-09-12 17:53:17

Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for Go as an example and helps you better understand the message sending and receiving processes.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.

You have installed Go.

You have downloaded the demo.

Directions

1. Run the following command in the client environment to RocketMQ client dependencies.

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

2. Create a producer in the corresponding method. If you need to send general messages, modify the corresponding parameters in the `syncSendMessage.go` file.

Delayed messages currently support delays of arbitrary precision without being subject to the delay level.

General Message

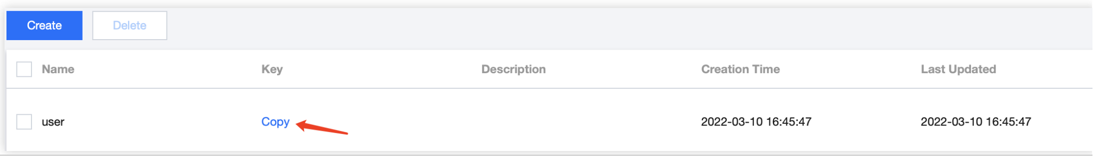
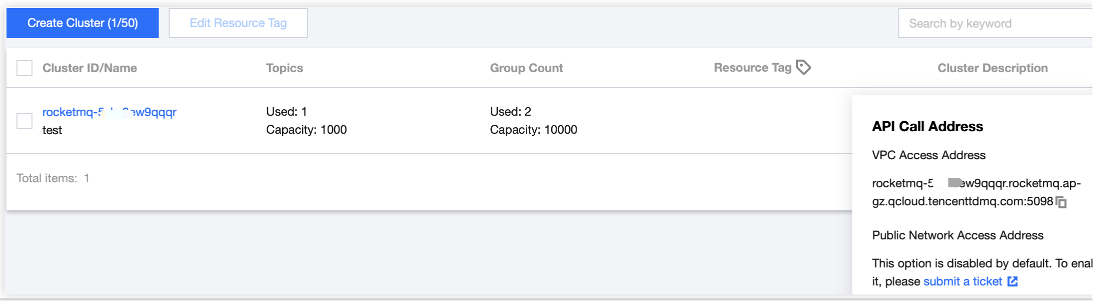
Delayed message

```
// Service access address (Note: Add “http://” or “https://” before the access addr
   var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:
   // Authorize the role name
   var secretKey = "admin"
   // Authorize the role token
   var accessKey = "eyJrZXlJZC...."
   // Full namespace name
   var nameSpace = "MQ_INST_rocketmqem4xxxx"
   // Producer group name
   var groupName = "group1"
   // Create a message producer
```

```
    p, _ := rocketmq.NewProducer(
        // Set the service address
        producer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddr
        // Set ACL permissions
        producer.WithCredentials(primitive.Credentials{
            SecretKey: secretKey,
            AccessKey: accessKey,
        }),
        // Set the producer group
        producer.WithGroupName(groupName),
        // Set the namespace name
        producer.WithNamespace(nameSpace),
        // Set the number of retries upon sending failures
        producer.WithRetry(2),
    )
    // Start the producer
    err := p.Start()
    if err != nil {
        fmt.Printf("start producer error: %s", err.Error())
        os.Exit(1)
    }
```

```
// Topic name
        var topicName = "topic1"
        // Producer group name
        var groupName = "group1"
        // Create a message producer
        p, _ := rocketmq.NewProducer(
                // Set the service address
                producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"
                // Set ACL permissions
                producer.WithCredentials(primitive.Credentials{
                        SecretKey: "admin",
                        AccessKey: "eyJrZXlJZC......",
                }),
                // Set the producer group
                producer.WithGroupName(groupName),
                // Set the namespace name
                producer.WithNamespace("rocketmq-xxx|namespace_go"),
                // Set the number of retries upon sending failures
                producer.WithRetry(2),
        )
        // Start the producer
        err := p.Start()
        if err != nil {
                fmt.Printf("start producer error: %s", err.Error())
                os.Exit(1)
        }

        for i := 0; i < 1; i++ {
                msg := primitive.NewMessage(topicName, []byte("Hello RocketMQ Go Cl
                // Set delay level
                // The relationship between the delay level and the delay time:
                // 1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m,
                // 1    2    3    4    5    6   7    8   9   10   11   12  13   1
                // If you want to use the delay level, then set the following metho

                msg.WithDelayTimeLevel(3)
                // If you want to use any delayed message, then set the following m
                delayMills := int64(10 * 1000)
                msg.WithProperty("__STARTDELIVERTIME", strconv.FormatInt(time.Now()
                // Send the message

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

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

Parameter	Description

secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Name Key Description Creation Time Last Updated user Copy 2022-03-10 16:45:47 2022-03-10 16:45:47
nameSpace	Namespace name, which can be copied on the **Namespace** page in the console.
serverAddress	Cluster access address, which can be copied from **Access Address** in the **Operation** column o console. Namespace access addresses in new virtual or exclusive clusters can be copied from th Add `http://` or `https://` before the access address; otherwise, it cannot be resolved.  Create Cluster (1/50) Edit Resource Tag Search by keyword Cluster ID/Name Topics Group Count Resource Tag Cluster Description rocketmq-[redacted]w9qqqr test Used: 1 Capacity: 1000 Used: 2 Capacity: 10000 Total items: 1 API Call Address VPC Access Address rocketmq-[redacted]ew9qqqr.rocketmq.ap-gz.qcloud.tencenttdmq.com:5098 Public Network Access Address This option is disabled by default. To enal it, please submit a ticket
groupName	Producer group name, which can be copied under the **Group** tab in the console.

3. The process of sending messages (using sync sending as an example) is the same as above.

```
// Topic name
   var topicName = "topic1"
   // Configure message content
   msg := &primitive.Message{
       Topic: topicName, // Set the topic name
       Body:  []byte("Hello RocketMQ Go Client! This is a new message."),
   }
   // Set tags
   msg.WithTag("TAG")
   // Set keys
   msg.WithKeys([]string{"yourKey"})
```

```
    // Send the message
    res, err := p.SendSync(context.Background(), msg)

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

Parameter	Description
topicName	Topic name, which can be copied under the **Topic** tab on the cluster details page in the console.
TAG	Message tag identifier
yourKey	Message business key

Release the resource.

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

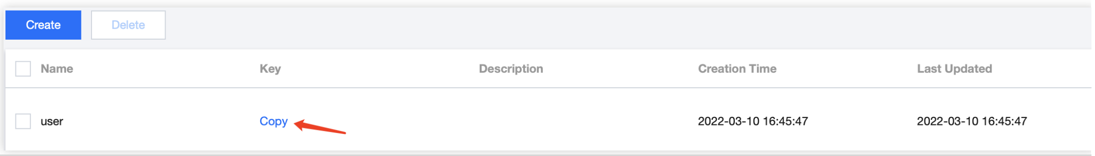
Note

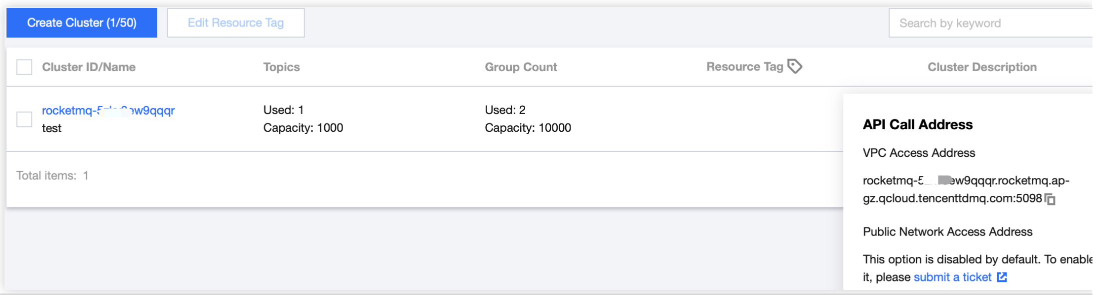
For more information on async sending and one-way sending, see Demo or RocketMQ-Client-Go Example.

4. Create a consumer.

```
// Service access address (Note: Add “http://” or “https://” before the access addr
   var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:
   // Authorize the role name
   var secretKey = "admin"
   // Authorize the role token
   var accessKey = "eyJrZXlJZC...."
   // Full namespace name
   var nameSpace = "rocketmq-xxx|namespace_go"
   // Producer group name
   var groupName = "group11"
   // Create a consumer
```

```
    c, err := rocketmq.NewPushConsumer(
        // Set the consumer group
        consumer.WithGroupName(groupName),
        // Set the service address
        consumer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddr
        // Set ACL permissions
        consumer.WithCredentials(primitive.Credentials{
            SecretKey: secretKey,
            AccessKey: accessKey,
        }),
        // Set the namespace name
        consumer.WithNamespace(nameSpace),
        // Set consumption from the start offset
        consumer.WithConsumeFromWhere(consumer.ConsumeFromFirstOffset),
        // Set the consumption mode (cluster consumption by default)
        consumer.WithConsumerModel(consumer.Clustering),

        //For broadcasting consumption, set the instance name to the system name of
        consumer.WithInstance("xxxx"),
    )
    if err != nil {
        fmt.Println("init consumer2 error: " + err.Error())
        os.Exit(0)
    }
```

Parameter	Description
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Name | Key | Description | Creation Time | Last Updated user | Copy | | 2022-03-10 16:45:47 | 2022-03-10 16:45:47
nameSpace	The full namespace name can be copied under the **Topic** tab on the **Cluster** page in the console cluster ID + | + namespace.
serverAddress	Cluster access address, which can be copied from **Access Address** in the **Operation** column o the console. Namespace access addresses in new virtual or exclusive clusters can be copied fro Note: Add `http://` or `https://` before the access address; otherwise, it cannot be resol

	
groupName	Producer group name, which can be copied under the **Group** tab in the console.

Create Cluster (1/50) | Edit Resource Tag | Search by keyword

Cluster ID/Name	Topics	Group Count	Resource Tag	Cluster Description
rocketmq-[redacted]w9qqqr test	Used: 1 Capacity: 1000	Used: 2 Capacity: 10000		

Total items: 1

API Call Address

VPC Access Address

rocketmq-[redacted]w9qqqr.rocketmq.ap-gz.qcloud.tencenttdmq.com:5098

Public Network Access Address

This option is disabled by default. To enable it, please submit a ticket

5. Consume a message.

```
// Topic name
   var topicName = "topic1"
   // Set the tag of messages that are subscribed to
   selector := consumer.MessageSelector{
       Type:       consumer.TAG,
       Expression: "TagA || TagC",
   }
   // Set the delay level of consumption retry. A total of 18 levels can be set. Be
   // 1   2   3    4    5    6    7    8    9    10   11   12   13   14    15    16    17   18
   // 1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m, 30m, 1h, 2h
   delayLevel := 1
```

```
    err = c.Subscribe(topicName, selector, func(ctx context.Context,
                                                              msgs ...*primitive
        fmt.Printf("subscribe callback len: %d \\n", len(msgs))
        // Set the delay level for the next consumption
        concurrentCtx, _ := primitive.GetConcurrentlyCtx(ctx)
        concurrentCtx.DelayLevelWhenNextConsume = delayLevel // only run when return

        for _, msg := range msgs {
            // Simulate a successful consumption after three retries
            if msg.ReconsumeTimes > 3 {
                fmt.Printf("msg ReconsumeTimes > 3. msg: %v", msg)
                return consumer.ConsumeSuccess, nil
            } else {
                fmt.Printf("subscribe callback: %v \\n", msg)
            }
        }
        // Simulate a consumption failure. Retry is required.
        return consumer.ConsumeRetryLater, nil
    })
    if err != nil {
        fmt.Println(err.Error())
    }
```

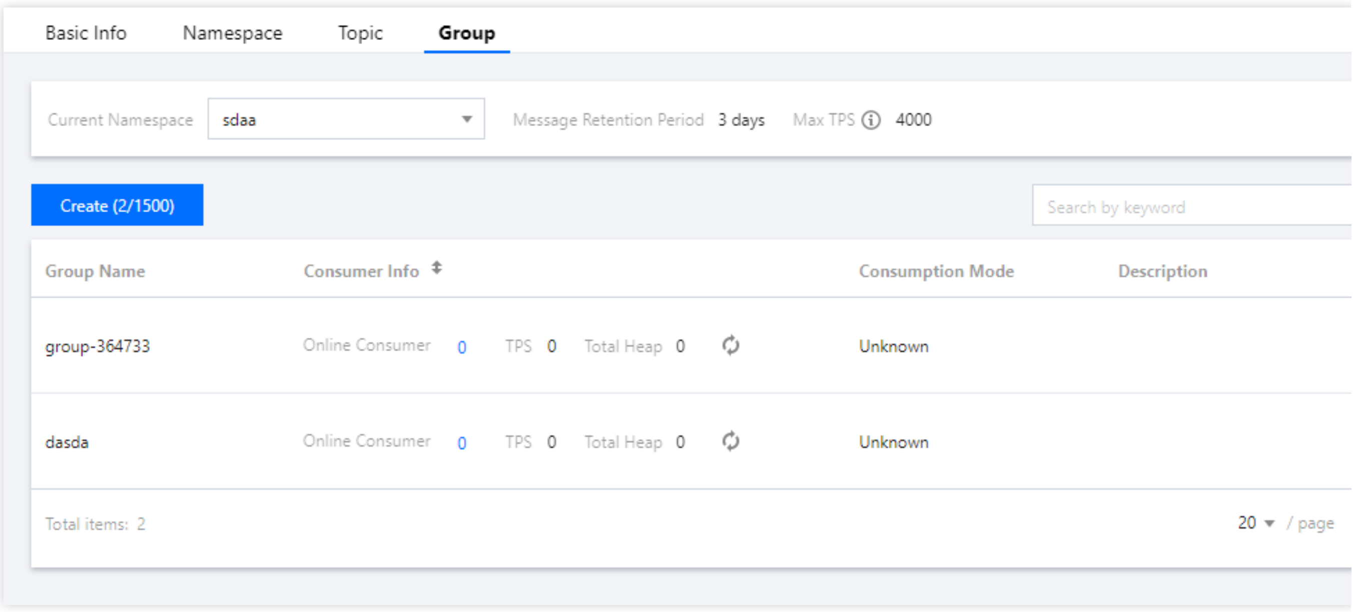
Parameter	Description
topicName	Topic name, which can be copied on the **Topic** page in the console.
Expression	Message tag identifier
delayLevel	A parameter used to set the delay level of consumption retry. A total of 18 delay levels are supported.

6. Consume messages (the consumer can consume messages only after the messages are subscribed to).

```
// Start consumption
   err = c.Start()
   if err != nil {
       fmt.Println(err.Error())
       os.Exit(-1)
   }
   time.Sleep(time.Hour)
   // Release resources
   err = c.Shutdown()
   if err != nil {
       fmt.Printf("shundown Consumer error: %s", err.Error())
```

```
    }
```

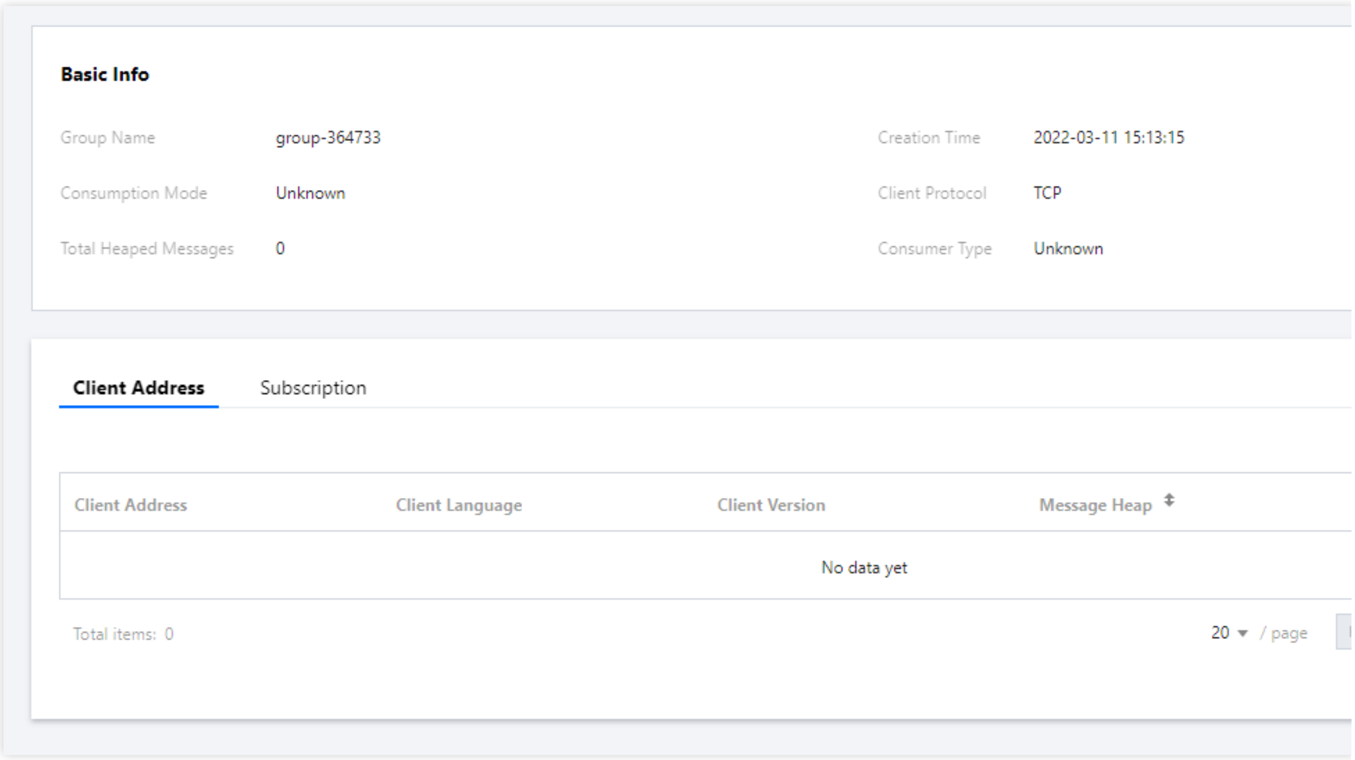
7. View consumption details. Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.



Current Namespace sdaa   Message Retention Period 3 days   Max TPS 4000

Group Name	Consumer Info	Consumption Mode	Description
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

Total items: 2   20 / page



Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address	Client Language	Client Version	Message Heap
No data yet			

Total items: 0   20 / page

Note

Above is a brief introduction to how to send and receive messages with the Go client. For more information, see Demo or Rocketmq-Client-Go Example.

SDK for Python

Last updated：2023-09-12 17:53:17

Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for Python as an example and helps you better understand the message sending and receiving processes.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed Python.
You have installed pip.
You have downloaded the demo.

Directions

Step 1. Prepare the environment

As RocketMQ-client Python is lightweight wrapper around rocketmq-client-cpp, you need to install `librocketmq` first.

Note

Currently, the Python client only supports Linux and macOS operating systems. It doesn't support Windows systems.

1. Install `librocketmq` 2.0.0 or later as instructed in Install librocketmq.
2. Run the following command to install `rocketmq-client-python` .

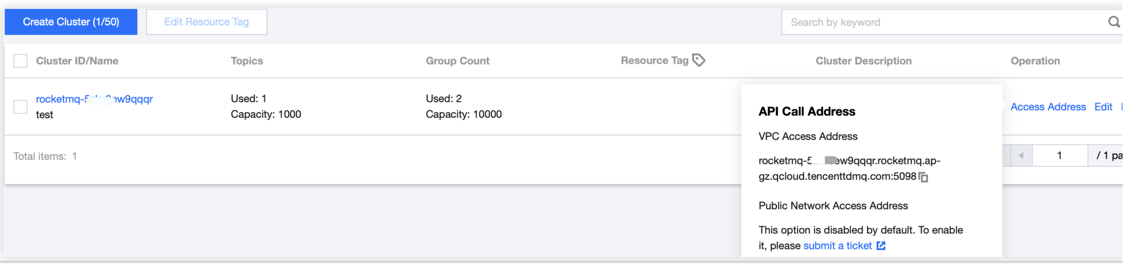
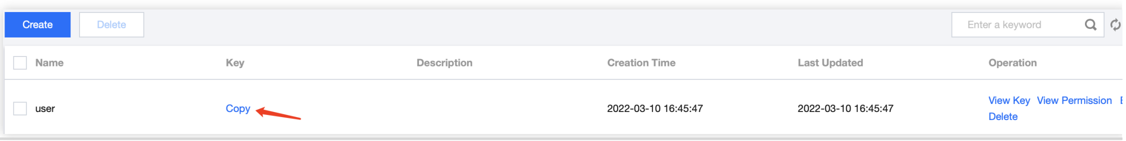
```
pip install rocketmq-client-python
```

Step 2. Produce messages

Create, compile, and run a message production program.

```
from rocketmq.client import Producer, Message

   # Initialize the producer and set the producer group information. Be sure to use
   producer = Producer(groupName)
   # Set the service address
   producer.set_name_server_address(nameserver)
   # Set permissions (role name and token)
   producer.set_session_credentials(
        accessKey,  # Role token
       secretKey,  # Role name
       ''
```

```
    )
    # Start the producer
    producer.start()

    # Assemble messages. The topic name can be copied on the **Topic** page in the c
    msg = Message(topicName)
    # Set keys
    msg.set_keys(TAGS)
    # Set tags
    msg.set_tags(KEYS)
    # Message content
    msg.set_body('This is a new message.')

    # Send messages in sync mode
    ret = producer.send_sync(msg)
    print(ret.status, ret.msg_id, ret.offset)
    # Release resources
    producer.shutdown()
```

Parameter	Description
groupName	Producer group name, which can be obtained under the **Group** tab on the cluster details page in the console.
nameserver	Cluster access address, which can be copied from **Access Address** in the **Operation** column on t **Cluster** page in the console. Namespace access addresses in new virtual or exclusive clusters can copied from the **Namespace** list.  Create Cluster (1/50) Edit Resource Tag Search by keyword Cluster ID/Name Topics Group Count Resource Tag Cluster Description Operation rocketmq-[redacted]w9qqqr test Used: 1 Capacity: 1000 Used: 2 Capacity: 10000 Access Address Edit Total items: 1 1 / 1 pa **API Call Address** VPC Access Address rocketmq-[redacted]ew9qqqr.rocketmq.ap-gz.qcloud.tencenttdmq.com:5098 Public Network Access Address This option is disabled by default. To enable it, please submit a ticket
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Enter a keyword Name Key Description Creation Time Last Updated Operation user Copy 2022-03-10 16:45:47 2022-03-10 16:45:47 View Key View Permission Delete
topicName	Topic name, which can be copied on the **Topic** page in the console.
TAGS	A parameter used to set the message tag.

KEYS	A parameter used to set the message key.

There are certain defects in the message production of the open-source Python client, causing uneven load among different queues of the same Topic. For more information, see [RocketMQ document](https://github.com/apache/rocketmq-client-python/issues /128!cac28b204e4c02765f18ecd741ed1628).

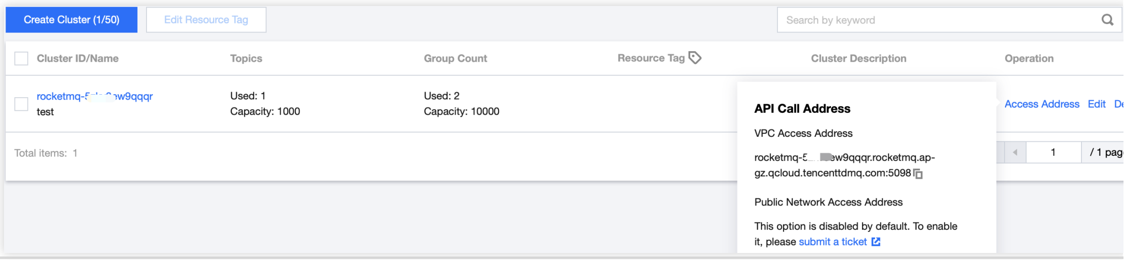
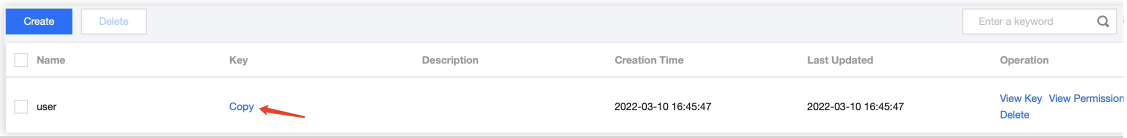
Step 3. Consume messages

Create, compile, and run a message consumption program.

```
import time
```

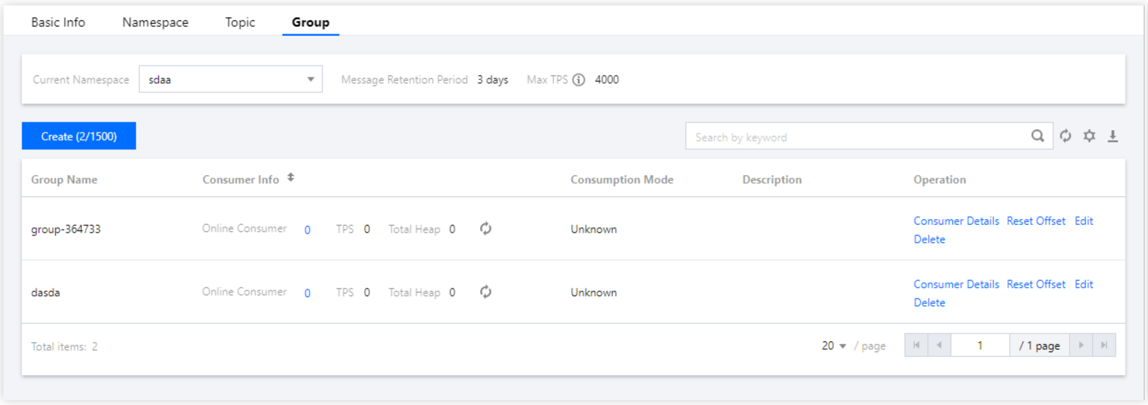
```
from rocketmq.client import PushConsumer, ConsumeStatus


# Message processing callback
def callback(msg):
    # Simulate the business processing logic
    print('Received message. messageId: ', msg.id, ' body: ', msg.body)
    # Return CONSUME_SUCCESS if the consumption is successful
    return ConsumeStatus.CONSUME_SUCCESS
    # Return the consumption status if the consumption is successful
    # return ConsumeStatus.RECONSUME_LATER


# Initialize the consumer and set the consumer group information
consumer = PushConsumer(groupName)
# Set the service address
consumer.set_name_server_address(nameserver)
# Set permissions (role name and token)
consumer.set_session_credentials(
    accessKey,        # Role token
    secretKey,   # Role name
    ''
)
# Subscribe to a topic
consumer.subscribe(topicName, callback, TAGS)
print(' [Consumer] Waiting for messages.')
# Start the consumer
consumer.start()

while True:
    time.sleep(3600)
# Release resources
consumer.shutdown()
```

Parameter	Description
groupName	Consumer group name, which can be copied under the **Group** tab on the cluster details page.
nameserver	Cluster access address, which can be copied from **Access Address** in the **Operation** column on t **Cluster** page in the console. Namespace access addresses in new virtual or exclusive clusters can copied from the **Namespace** list.

	 Create Cluster (1/50) Edit Resource Tag Search by keyword Cluster ID/Name | Topics | Group Count | Resource Tag | Cluster Description | Operation rocketmq-[redacted]w9qqqr test | Used: 1 Capacity: 1000 | Used: 2 Capacity: 10000 | | | Access Address Edit De Total items: 1 1 / 1 pag API Call Address VPC Access Address rocketmq-5[redacted]w9qqqr.rocketmq.ap-gz.qcloud.tencenttdmq.com:5098 Public Network Access Address This option is disabled by default. To enable it, please submit a ticket
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Enter a keyword Name | Key | Description | Creation Time | Last Updated | Operation user | Copy | | 2022-03-10 16:45:47 | 2022-03-10 16:45:47 | View Key View Permission Delete
topicName	Topic name, which can be copied on the **Topic** page in the console.
TAGS	A parameter used to set the tag of messages that are subscribed to. The default value is set to `*`, means subscribing to all messages.

Step 4. View consumption details

Log in to the TDMQ console, go to the **Cluster** > **Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.



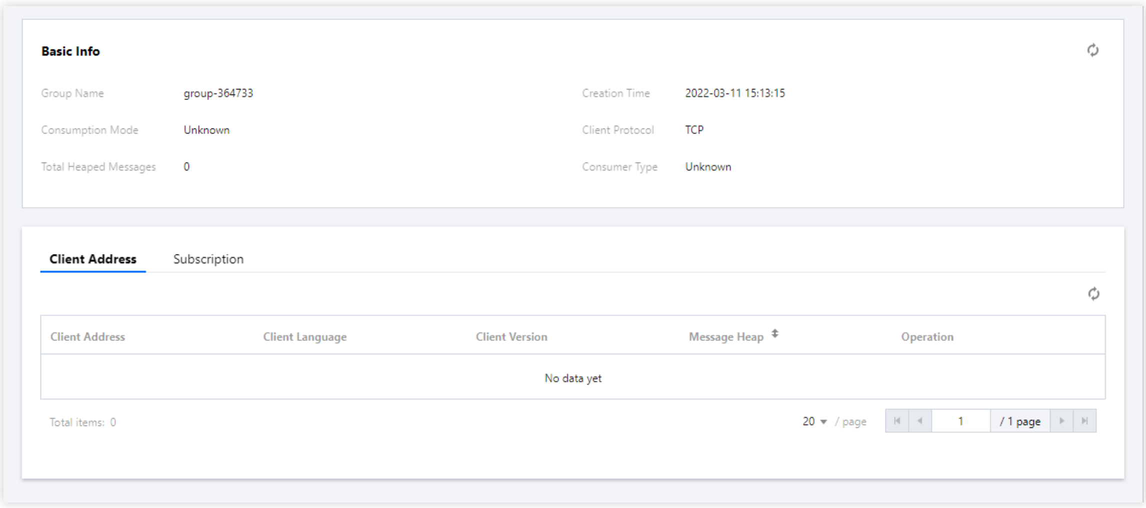
Basic Info Namespace Topic **Group**

Current Namespace sdaa Message Retention Period 3 days Max TPS 4000

Create (2/1500) Search by keyword

Group Name	Consumer Info	Consumption Mode	Description	Operation
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown		Consumer Details Reset Offset Edit Delete
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown		Consumer Details Reset Offset Edit Delete

Total items: 2 20 / page 1 / 1 page



Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

Client Address | Subscription

Client Address	Client Language	Client Version	Message Heap	Operation
No data yet				

Total items: 0

Note

Above is a brief introduction to message publishing and subscription. For more information, see Demo or RocketMQ-Client-Python Sample.

Access over HTTP

Last updated：2023-05-16 11:07:52

Overview

TDMQ for RocketMQ can be accessed over the HTTP protocol from the private or public network. It is compatible with HTTP SDKs for multiple programming languages in the community.
This document describes how to use HTTP SDK to send and receive messages by using the SDK for Java as an example and helps you better understand the message sending and receiving processes.
Note
Currently, transactional message and sequential message cannot be implemented over HTTP.
When creating a consumer group, you need to specify the type (TCP or HTTP, as described in Group Management); therefore, a consumer group does not support simultaneous consumption by TCP and HTTP clients.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have imported dependencies through Maven and added SDK dependencies of the corresponding programming language in the pom.xml file.
For more examples, see the demos in the open-source community.

Retry Mechanism

Every message consumed over HTTP will have an **invisibility time** of 5 minutes.
If the client acknowledges a message within the invisibility time, the consumption is successful and will not be retried.
If the client does not acknowledge a message after the invisibility time elapses, the message will become visible again, that is, the client will consume the message again subsequently.
Note that after the invisibility time of a message elapses during one consumption, the message handler will become invalid, and the message can no longer be acknowledged.

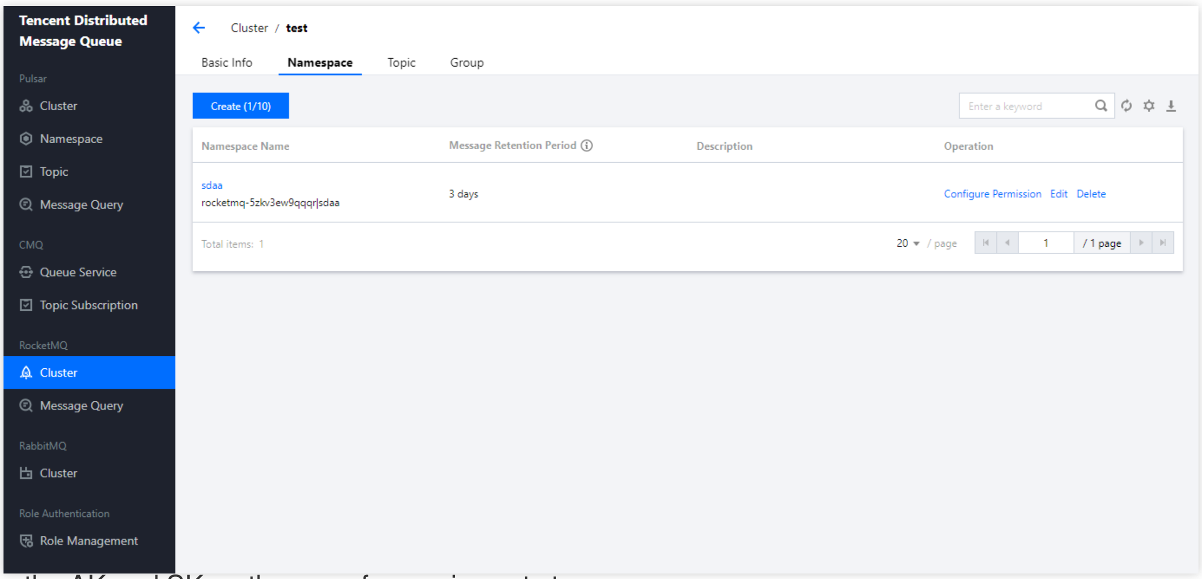
Directions

Step 1. Import dependencies

Import the SDK dependencies of the corresponding programming language into the pom.xml file of the project.

Step 2. Get parameters

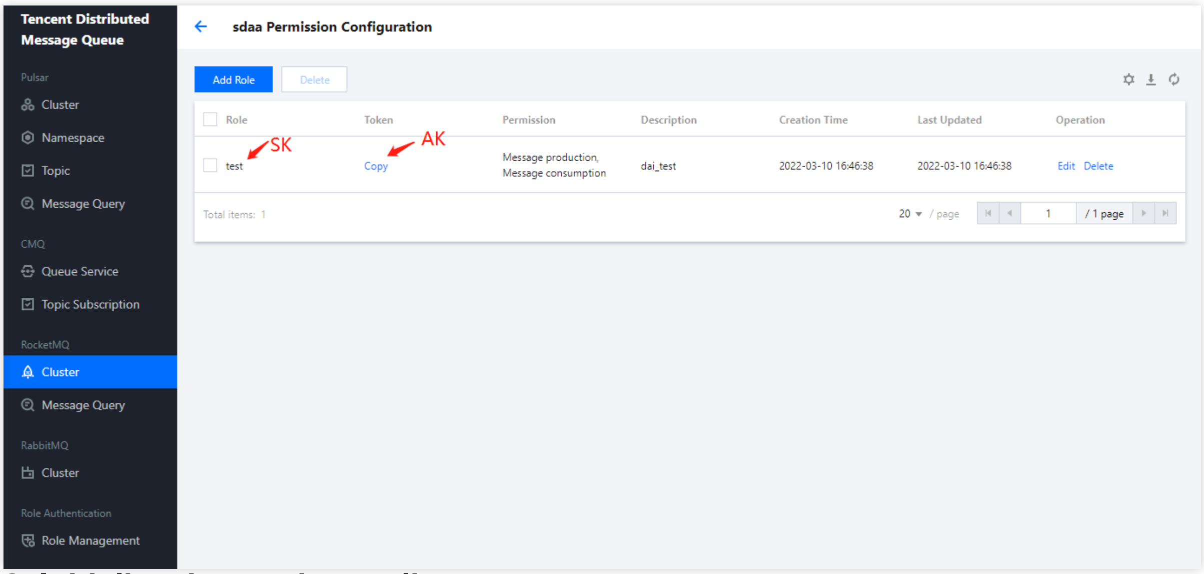
1. Log in to the TDMQ console, select the target cluster, and click the cluster name to enter the cluster details page.
2. Select the **Namespace** tab at the top and click **Configure Permission** on the right to enter the permission configuration page. If the role list is empty, click **Create** to create a role. For more information, see Resource Creation and Preparation.



Namespace Name	Message Retention Period	Description	Operation
sdaa rocketmq-5zkv3ew9qqqr|sdaa	3 days		Configure Permission Edit Delete

Total items: 1　20 / page　1 / 1 page

3. Copy the AK and SK on the page for use in next steps.



Role	Token	Permission	Description	Creation Time	Last Updated	Operation
test	Copy	Message production, Message consumption	dai_test	2022-03-10 16:46:38	2022-03-10 16:46:38	Edit Delete

Total items: 1　20 / page　1 / 1 page

Step 3. Initialize the producer client

JAVA
PHP

NodeJS

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

public class Producer {

    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // HTTP access point
                "${HTTP_ENDPOINT}",
                // Access key, which can be created and obtained in the TDMQ for Ro
                "${ACCESS_KEY}",
                // Role name, which can be created and obtained in the TDMQ for Roc
                "${SECRET_KEY}"
        );

        // The topic used for sending messages, which is required and can be obtain
        final String topic = "${TOPIC}";
        // The namespace of the topic, which is required and can be obtained in the
        final String instanceId = "${INSTANCE_ID}";

        // Create a producer
        MQProducer producer = mqClient.getProducer(instanceId, topic);

        // Send the message

        mqClient.close();
    }
}
```

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

use MQ\\MQClient;

class ProducerTest
{
    private $client;
    private $producer;

    public function __construct()
    {
        $this->client = new MQClient(
            // HTTP access point
            "${HTTP_ENDPOINT}",
            // Access key, which can be created and obtained in the TDMQ for Rocket
            "${ACCESS_KEY}",
            // Role name, which can be created and obtained in the TDMQ for RocketM
            "${SECRET_KEY}"
        );

        // The topic used for sending messages, which is required and can be obtain
        $topic = "${TOPIC}";
        // The namespace of the topic, which is required and can be obtained in the
        $instanceId = "${INSTANCE_ID}";

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

    public function run()
    {
        // Send the message
    }
}


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

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

// Set HTTP access endpoints
const endpoint = "{Endpoint}";
// AccessKey
const accessKeyId = "{Accesskey}";
// SecretKey
const accessKeySecret = "rop";
```

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

// Its Topic
const topic = "TopicA";
// ID of the instance to which the topic belongs
const instanceId = "MQ_INST_xxxxx";

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

(async function(){
  try {
    // Send 4 messages in a loop
    for(var i = 0; i < 4; i++) {
      let res;
      if (i % 2 == 0) {
        msgProps = new MessageProperties();
        // Set attributes
        msgProps.putProperty("key", i);
        // Set keys
        msgProps.messageKey("MessageKey");
        res = await producer.publishMessage("hello mq.", "", msgProps);
      } else {
        msgProps = new MessageProperties();
        // Set attributes
        msgProps.putProperty("key", i);
        // Timed message, with the time being 10s later
        msgProps.startDeliverTime(Date.now() + 10 * 1000);
        res = await producer.publishMessage("hello mq. timer msg!", "TagA", msgProp
      }
      console.log("Publish message: MessageID:%s,BodyMD5:%s", res.body.MessageId, r
    }

  } catch(e) {
    // The message failed to be sent and needs to be retried. You can resend this m
    console.log(e)
  }
```

Step 4. Initialize the consumer client

JAVA

PHP

NodeJS

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

public class Consumer {

    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // HTTP access point
                "${HTTP_ENDPOINT}",
                // Access key, which can be created and obtained in the TDMQ for Ro
                "${ACCESS_KEY}",
                // Role name, which can be created and obtained in the TDMQ for Roc
                "${SECRET_KEY}"
        );

        // The topic used for consuming messages, which is required and can be obta
        final String topic = "${TOPIC}";
        // Consumer group name, which is required and can be obtained in the TDMQ c
        final String groupId = "${GROUP_ID}";
        // The namespace of the topic, which is required and can be obtained in the
        final String instanceId = "${INSTANCE_ID}";

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

        do {
            // Consume a message
        } while (true);
    }
}
```

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

use MQ\\MQClient;

class ConsumerTest
{
    private $client;
    private $consumer;

    public function __construct()
    {
        $this->client = new MQClient(
            // HTTP access point
            "${HTTP_ENDPOINT}",
            // Access key, which can be created and obtained in the TDMQ for Rocket
            "${ACCESS_KEY}",
            // Role name, which can be created and obtained in the TDMQ for RocketM
            "${SECRET_KEY}"
        );

        // The topic used for consuming messages, which is required and can be obta
        $topic = "${TOPIC}";
        // Consumer group name, which is required and can be obtained in the TDMQ c
        $groupId = "${GROUP_ID}";
        // The namespace of the topic, which is required and can be obtained in the
        $instanceId = "${INSTANCE_ID}";

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

    public function run()
    {
        while (True) {
            // Consume a message
        }
    }
}


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

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

// Set HTTP access endpoints
const endpoint = "{Endpoint}";
// AccessKey
const accessKeyId = "{Accesskey}";
// SecretKey
const accessKeySecret = "rop";
```

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

// Its Topic
const topic = "TopicA";
// ID of the instance to which the topic belongs
const instanceId = "MQ_INST_xxxxx";
// The consumer group you created in the console
const groupId = "GID_xxx";

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

(async function(){
  // Consume messages in loop
  while(true) {
    try {
      // long polling of consumption messages
      // Long polling means that if the topic has no messages, the request will han
      res = await consumer.consumeMessage(
          3, // This indicates a maximum of 3 messages can be consumed at a time. U
          3 // Long polling lasts 3 seconds, which can be set up to 30 seconds.
          );

      if (res.code == 200) {
        // Consume messages based on business processing logic
        console.log("Consume Messages, requestId:%s", res.requestId);
        const handles = res.body.map((message) => {
          console.log("\\tMessageId:%s,Tag:%s,PublishTime:%d,NextConsumeTime:%d,Fir
            ",Props:%j,MessageKey:%s,Prop-A:%s",
              message.MessageId, message.MessageTag, message.PublishTime, message.N
              message.MessageBody,message.Properties,message.MessageKey,message.Pro
          return message.ReceiptHandle;
        });

        // If a message is not acked for successful consumption before `message.Nex
        // The message handle has a timestamp that changes each time the same messa
        res = await consumer.ackMessage(handles);
        if (res.code != 204) {
          // The handle of some messages may time out, which will cause the acknowl
          console.log("Ack Message Fail:");
          const failHandles = res.body.map((error)=>{
            console.log("\\tErrorHandle:%s, Code:%s, Reason:%s\\n", error.ReceiptHa
            return error.ReceiptHandle;
          });
          handles.forEach((handle)=>{
            if (failHandles.indexOf(handle) < 0) {
              console.log("\\tSucHandle:%s\\n", handle);
            }
          });
        } else {
          // The message is acked for successful consumption
          console.log("Ack Message suc, RequestId:%s\\n\\t", res.requestId, handles
        }
      }
    } catch(e) {
      if (e.Code.indexOf("MessageNotExist") > -1) {
        // If there is no message, long polling will continue on the server.
        console.log("Consume Message: no new message, RequestId:%s, Code:%s", e.Req
      } else {
        console.log(e);
      }
    }
  }
})();
```

Access over HTTP

SDK for Java

Sending and Receiving General Messages

Last updated：2023-09-13 11:36:59

Overview

TDMQ for RocketMQ can be accessed over the HTTP protocol from the private or public network. It is compatible with HTTP SDKs for multiple programming languages in the community.
This document describes how to use HTTP SDK to send and receive messages by using the SDK for Java as an example and helps you better understand the message sending and receiving processes.
Note
Currently, transactional message cannot be implemented over HTTP.
As a consumer group does not support simultaneous consumption by TCP and HTTP clients, you need to specify the type (TCP or HTTP) when creating a consumer group. For more information, see Group Management.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have imported dependencies through Maven and added SDK dependencies of the corresponding programming language in the pom.xml file.
For more examples, see the demos in the open-source community.

Retry Mechanism

A fixed retry interval is used in HTTP, which can't be customized currently.

Message Type	Retry Interval	Maximum Number of Retries
General Message	5 minutes	288
Sequential message	1 minute	288

Note
If the client acknowledges a message within the retry interval, the message consumption is successful and will not be retried.
If the client doesn’t acknowledge a message after the retry interval has expired, the message will become visible again, and the client will consume it again.
The message handle consumed each time is only valid within the retry interval, and become invalid after that time period.

Directions

Step 1. Install the Java dependent library

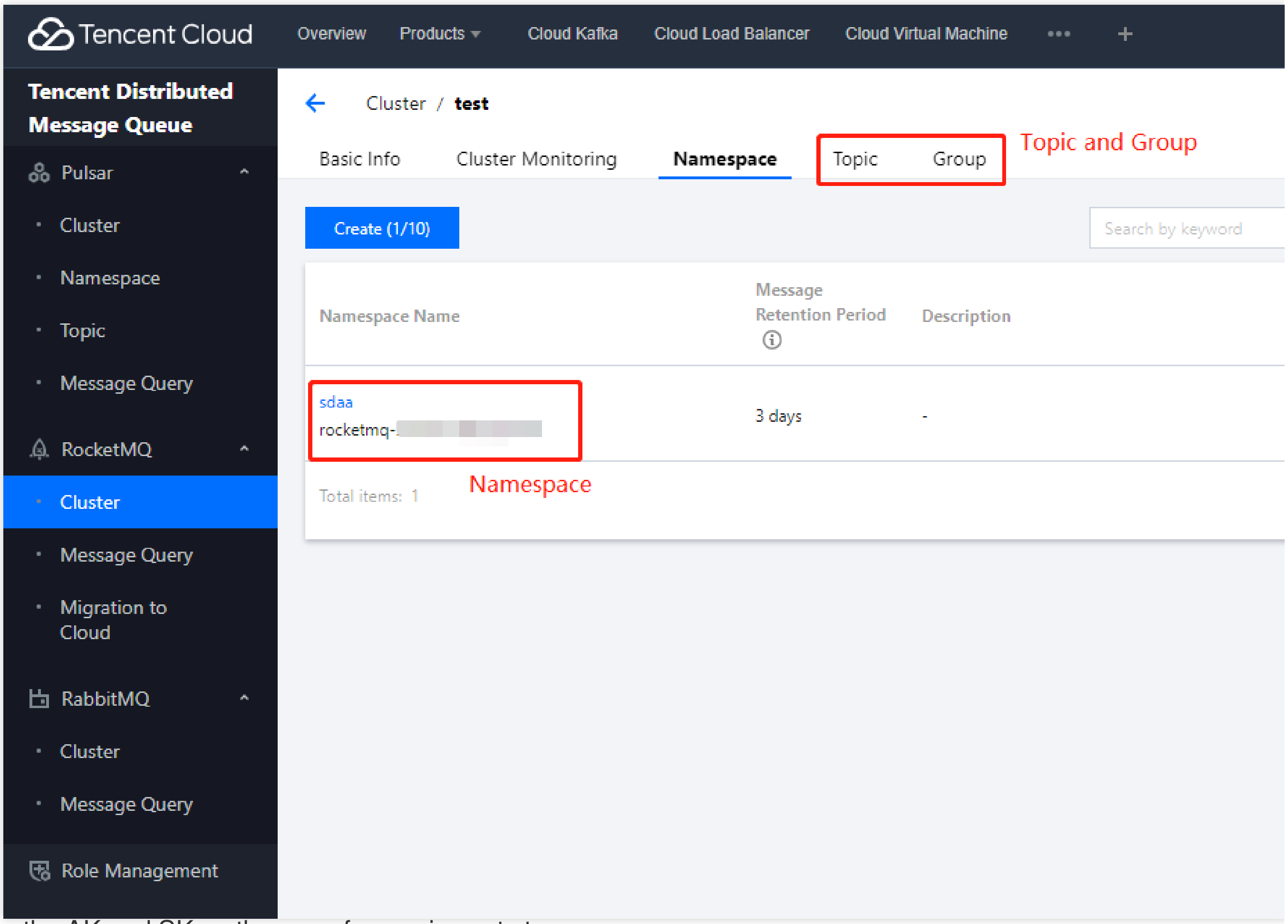
Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

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

Step 2. Get parameters

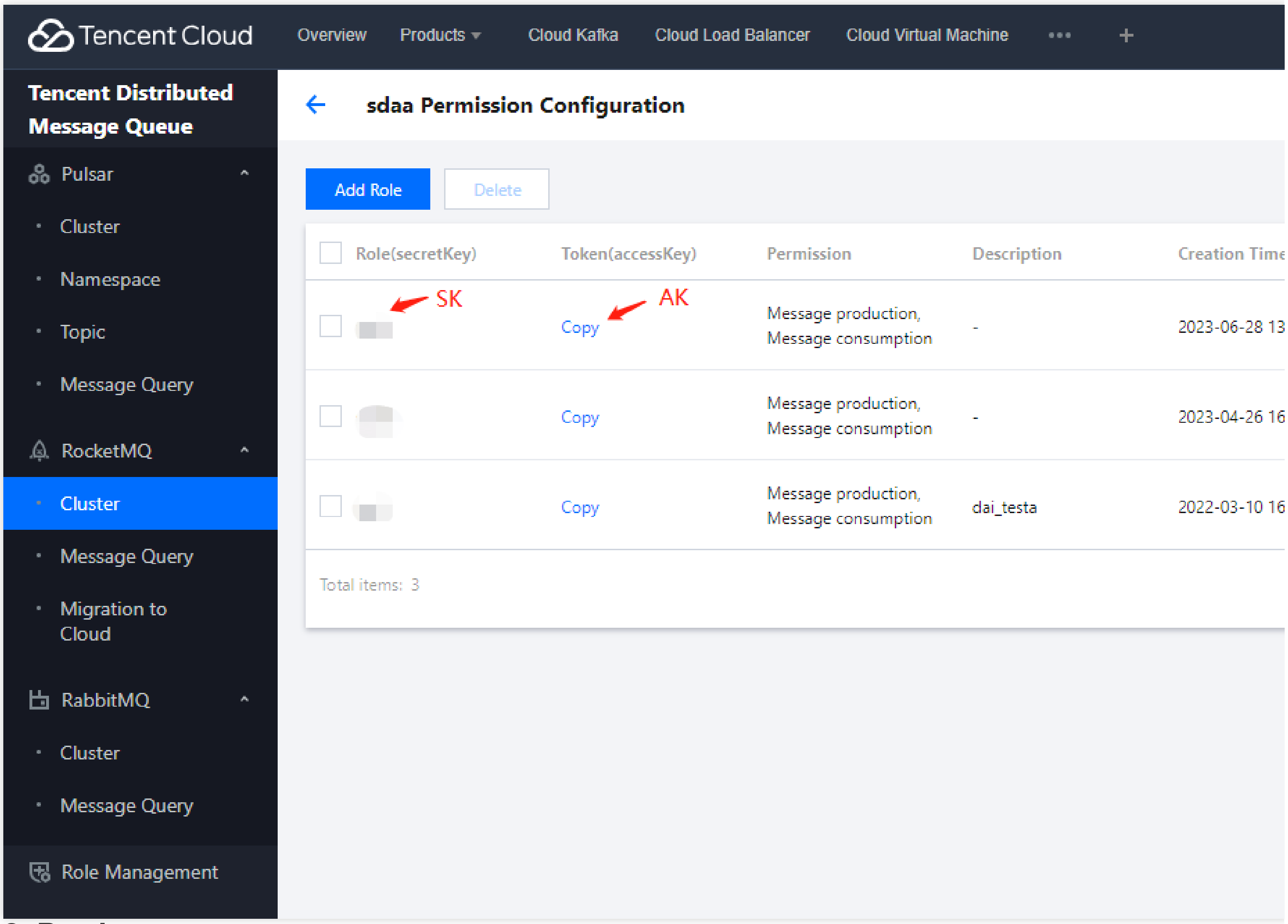
1. Log in to the TDMQ console, select the target cluster, and click the cluster name to enter the cluster details page.

2. Select the **Namespace** tab at the top and click **Configure Permission** on the right to enter the permission configuration page. If the role list is empty, click **Create** to create a role. For more information, see Resource Creation and Preparation.



Namespace Name	Message Retention Period	Description
sdaa rocketmq-[redacted]	3 days	-

3. Copy the AK and SK on the page for use in next steps.



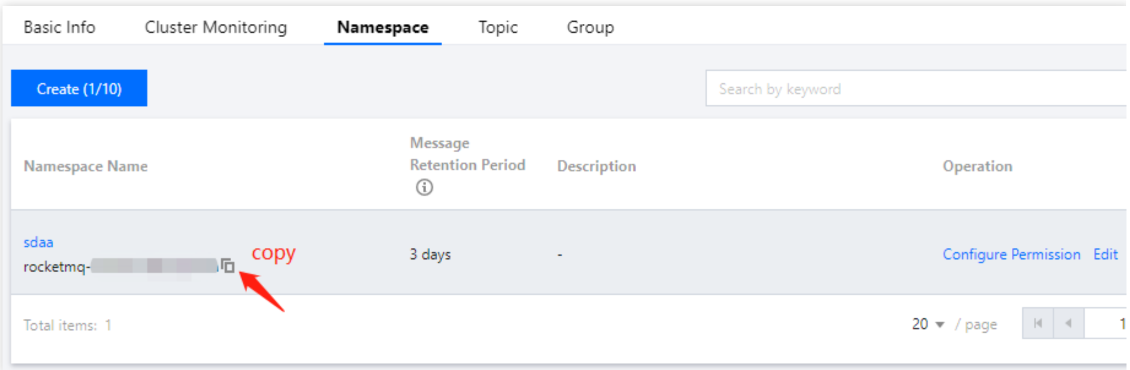
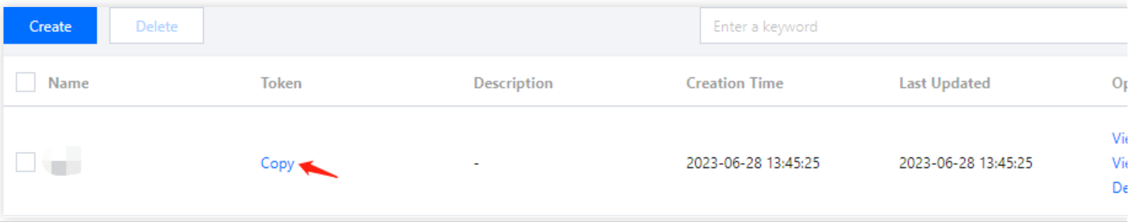
Role(secretKey)	Token(accessKey)	Permission	Description	Creation Time
	Copy	Message production, Message consumption	-	2023-06-28 13
	Copy	Message production, Message consumption	-	2023-04-26 16
	Copy	Message production, Message consumption	dai_testa	2022-03-10 16

Step 3. Produce messages

Creating a message producer

```
    // Get the client
    MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

    // Get the topic producer
    MQProducer producer = mqClient.getProducer(namespace, topicName);
```

Parameter	Description
topicName	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
namespace	Namespace name, which can be copied under the **Namespace** tab on the **Cluster** page in the cons

	 Basic Info Cluster Monitoring **Namespace** Topic Group Create (1/10) Search by keyword Namespace Name | Message Retention Period | Description | Operation sdaa rocketmq-[redacted] | 3 days | - | Configure Permission Edit Total items: 1 20 / page 1
endpoint	Cluster access address over HTTP, which can be obtained from **Access Address** in the **Operatio** **Cluster** page in the console.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Enter a keyword Name | Token | Description | Creation Time | Last Updated | Op [redacted] | Copy | - | 2023-06-28 13:45:25 | 2023-06-28 13:45:25 | Vi Vi De

Sending a message

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

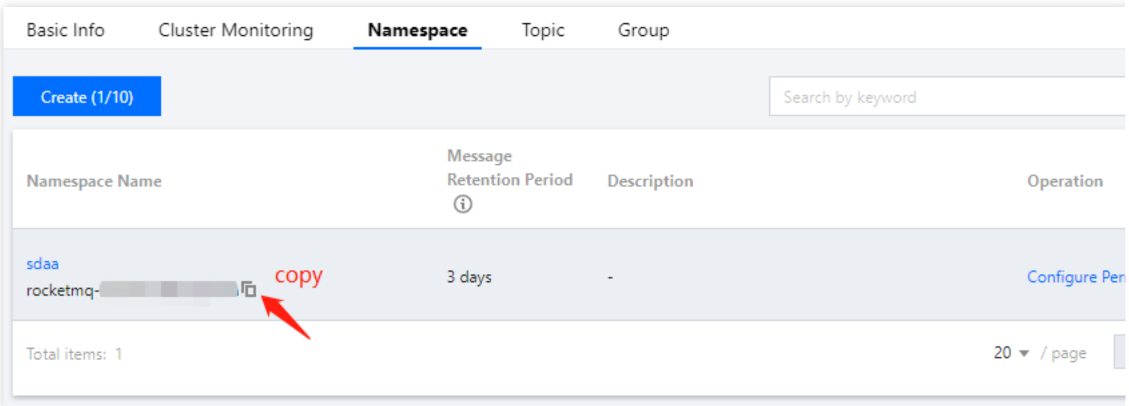
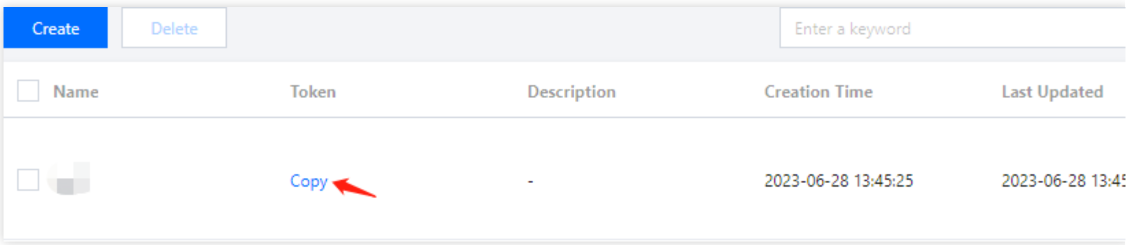
Parameter	Description
TAG	Set the message tag.

Step 4. Consume messages

Creating a consumer

```
    // Get the client
    MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

    // Get the topic consumer
    MQConsumer consumer = mqClient.getConsumer(namespace, topicName, groupName, "TAG
```

Parameter	Description
topicName	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
groupName	Producer group name, which can be copied under the **Group** tab on the **Cluster** page in the consol

namespace	Namespace name, which can be copied under the **Namespace** tab on the **Cluster** page in the cons  Basic Info / Cluster Monitoring / **Namespace** / Topic / Group; Create (1/10); Search by keyword; Namespace Name: sdaa rocketmq-[redacted] copy; Message Retention Period: 3 days; Description: -; Operation: Configure Per; Total items: 1; 20 / page
TAG	Subscribed tag.
endpoint	Cluster access address over HTTP, which can be obtained from **Access Address** in the **Operatio** the console.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create; Delete; Enter a keyword; Name: [redacted]; Token: Copy; Description: -; Creation Time: 2023-06-28 13:45:25; Last Updated: 2023-06-28 13:45

Subscribing to messages

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

    try {
        // long polling of consumption messages
        // Long polling means that if the topic has no messages, the request will w
        messages = consumer.consumeMessage(
                Integer.parseInt(batchSize),
                Integer.parseInt(waitSeconds)
        );
    } catch (Throwable e) {
        e.printStackTrace();
    }
    if (messages == null || messages.isEmpty()) {
        System.out.println(Thread.currentThread().getName() + ": no new message, co
        continue;
    }

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


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

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

Parameter	Description
batchSize	The number of messages pulled at a time. Maximum value: 16.
waitSeconds	The polling waiting time for a message pull. Maximum value: 30 seconds.

Sending and Receiving Sequential Messages

Last updated：2023-09-13 11:37:45

Overview

TDMQ for RocketMQ can be accessed over the HTTP protocol from the private or public network. It is compatible with HTTP SDKs for multiple programming languages in the community.
This document describes how to use HTTP SDK to send and receive messages by using the SDK for Java as an example and helps you better understand the message sending and receiving processes.

Note
Currently, transactional message cannot be implemented over HTTP.
As a consumer group does not support simultaneous consumption by TCP and HTTP clients, you need to specify the type (TCP or HTTP) when creating a consumer group. For more information, see Group Management.

Prerequisites

You have created the required resources as instructed in Resource Creation and Preparation.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have imported dependencies through Maven and added SDK dependencies of the corresponding programming language in the pom.xml file.
For more examples, see the demos in the open-source community.

Retry Mechanism

A fixed retry interval is used in HTTP, which can't be customized currently.

Message Type	Retry Interval	Maximum Number of Retries
General Message	5 minutes	288
Sequential message	1 minute	288

Note
If the client acknowledges a message within the retry interval, the message consumption is successful and will not be retried.

If the client doesn’t acknowledge a message after the retry interval has expired, the message will become visible again, and the client will consume it again.
The message handle consumed each time is only valid within the retry interval, and become invalid after that time period.

Directions

Step 1. Install the Java dependent library

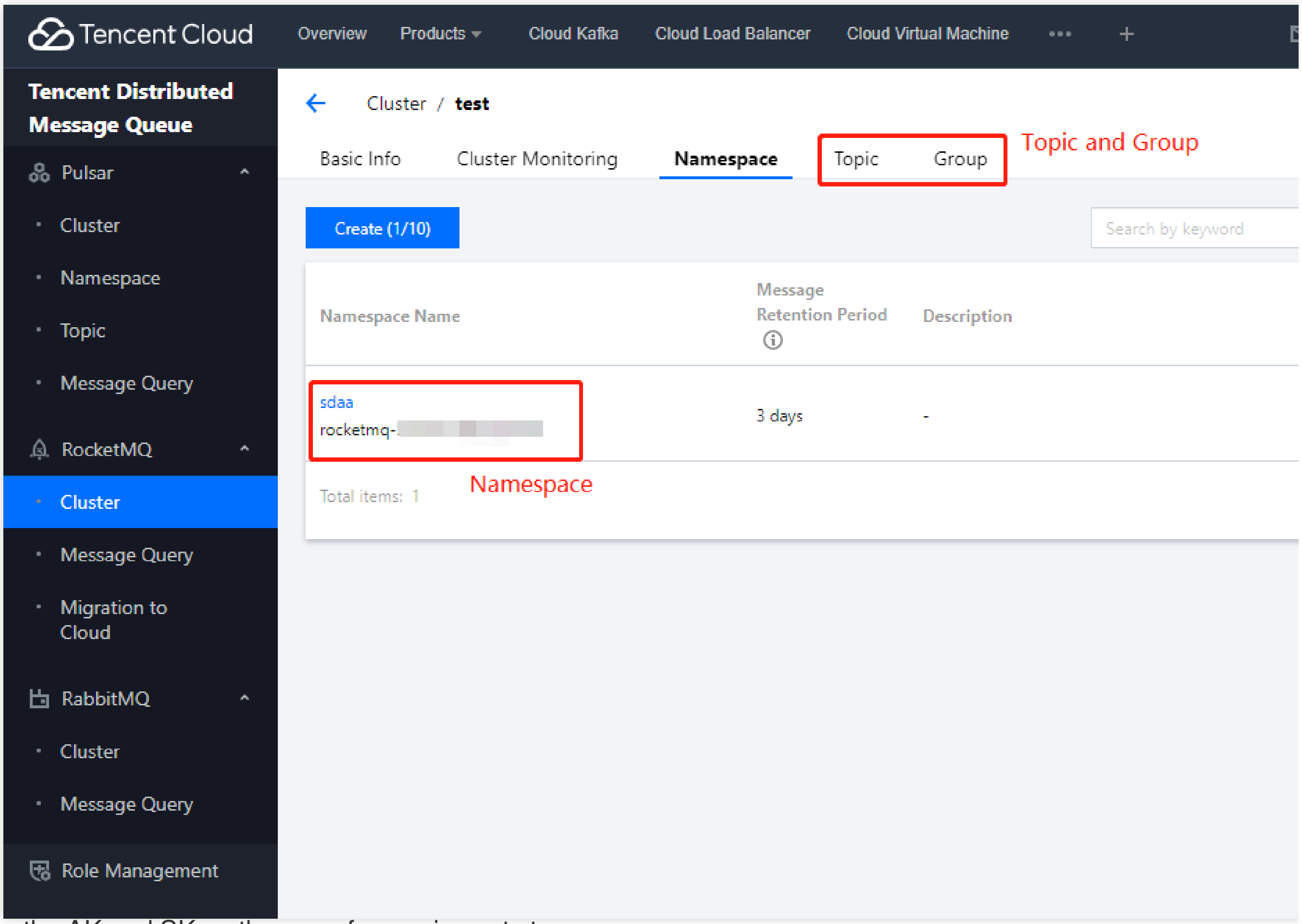
Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

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

Step 2. Get parameters

1. Log in to the TDMQ console, select the target cluster, and click the cluster name to enter the cluster details page.

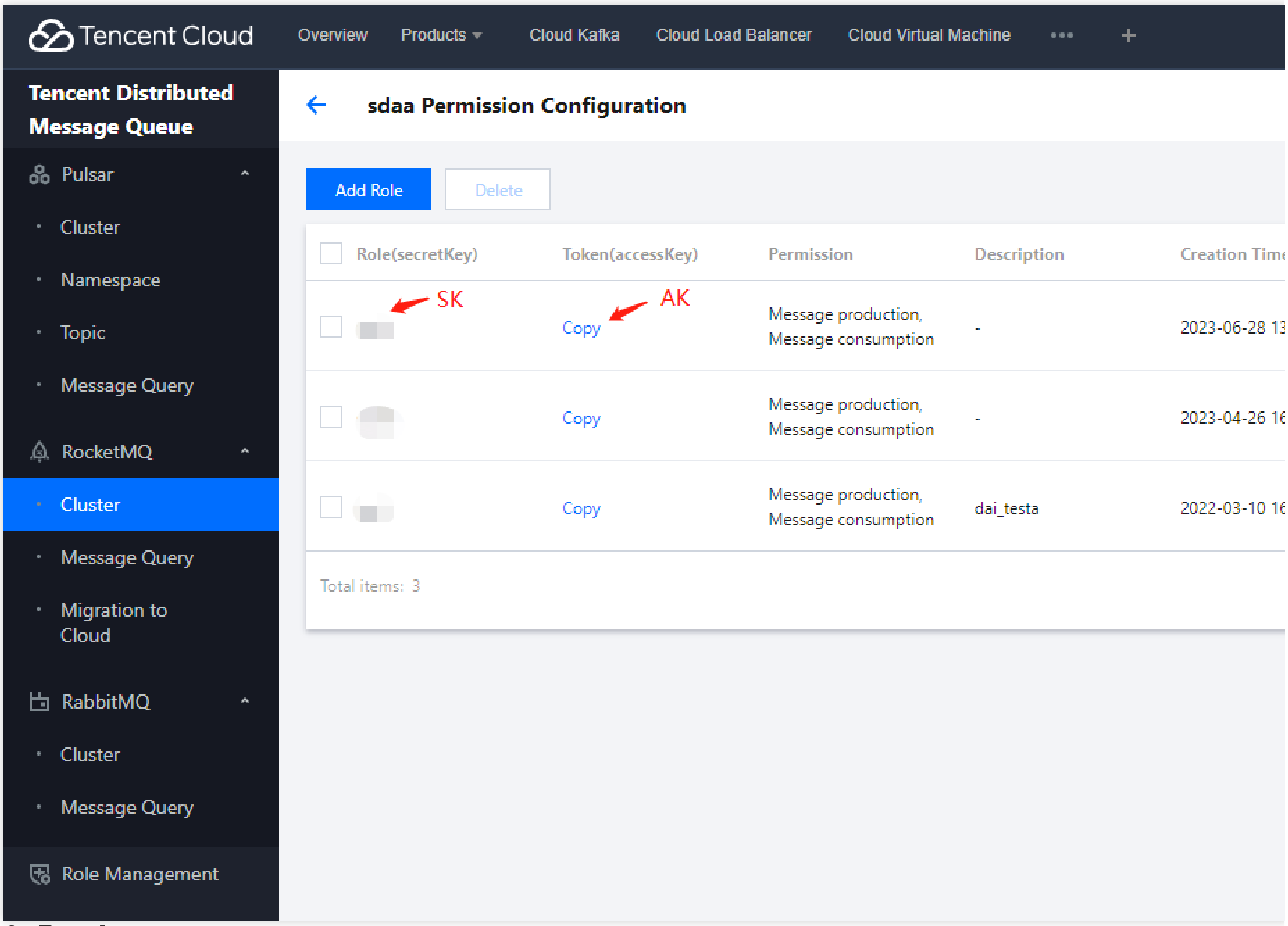
2. Select the **Namespace** tab at the top and click **Configure Permission** on the right to enter the permission configuration page. If the role list is empty, click **Create** to create a role. For more information, see Resource Creation and Preparation.



Namespace Name	Message Retention Period	Description
sdaa rocketmq-	3 days	-

Total items: 1

3. Copy the AK and SK on the page for use in next steps.



sdaa Permission Configuration

Role(secretKey)	Token(accessKey)	Permission	Description	Creation Tim
[redacted]	Copy	Message production, Message consumption	-	2023-06-28 13
[redacted]	Copy	Message production, Message consumption	-	2023-04-26 16
[redacted]	Copy	Message production, Message consumption	dai_testa	2022-03-10 16

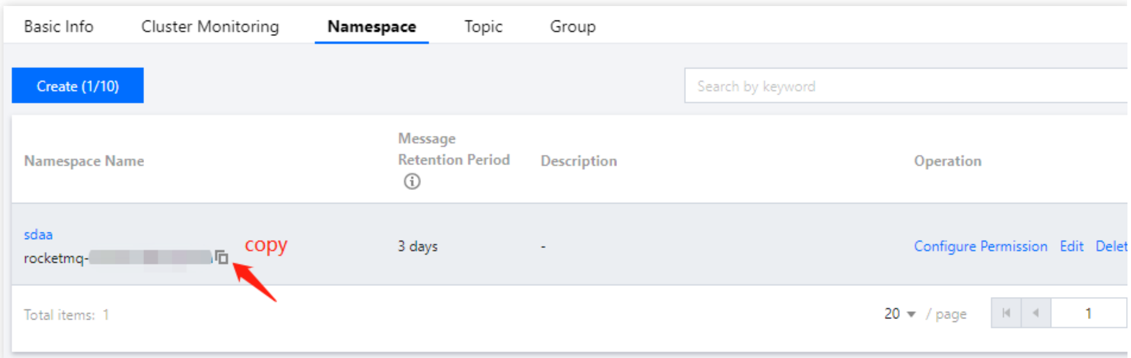
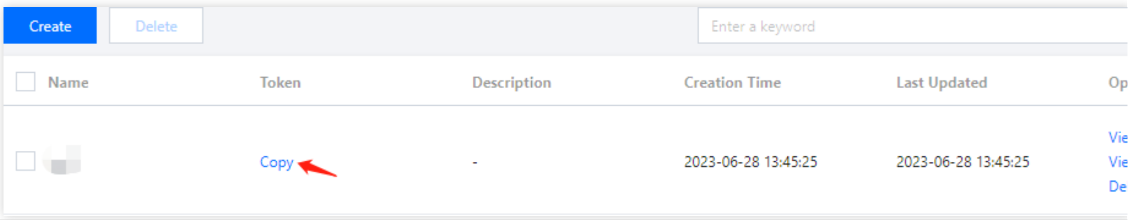
Total items: 3

Step 3. Produce messages

Creating a message producer

```
// Get the client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

// Get the topic producer
MQProducer producer = mqClient.getProducer(namespace, topicName);
```

Parameter	Description
topicName	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
namespace	Namespace name, which can be copied under the **Namespace** tab on the **Cluster** page in the cons

	 Basic Info Cluster Monitoring Namespace Topic Group Create (1/10) Search by keyword Namespace Name | Message Retention Period | Description | Operation sdaa rocketmq-[redacted] copy | 3 days | - | Configure Permission Edit Delet Total items: 1 20 / page 1
endpoint	Cluster access address over HTTP, which can be obtained from **Access Address** in the **Operatio** **Cluster** page in the console.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete Enter a keyword Name | Token | Description | Creation Time | Last Updated | Op [redacted] | Copy | - | 2023-06-28 13:45:25 | 2023-06-28 13:45:25 | Vie Vie De

Sending a message

```
try {
    for (int i = 0; i < 10; i++) {
        TopicMessage pubMsg;
        pubMsg = new TopicMessage(
            ("Hello RocketMQ " + i).getBytes(),
            "TAG"
        );
        // Set the ShardingKey of the partitionally sequential message
        pubMsg.setShardingKey(i % 3);
        TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
        System.out.println("Send mq message success. MsgId is: " + pubResultMsg.get
    }
} catch (Throwable e) {
    System.out.println("Send mq message failed.");
    e.printStackTrace();
}
```

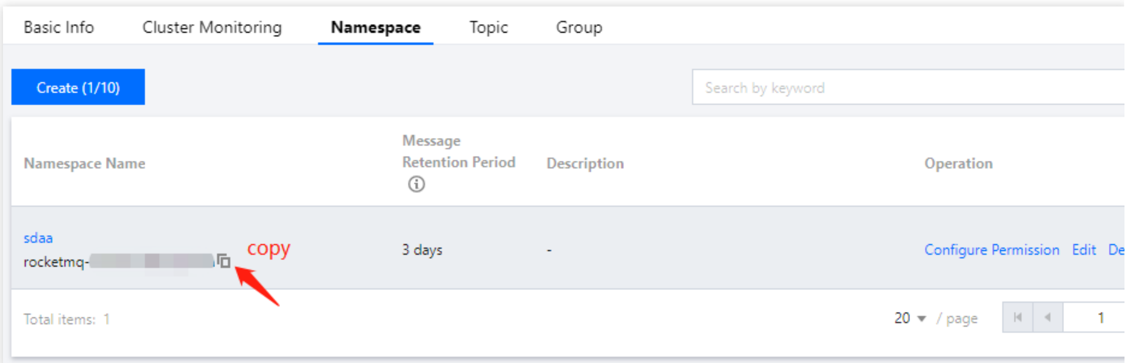
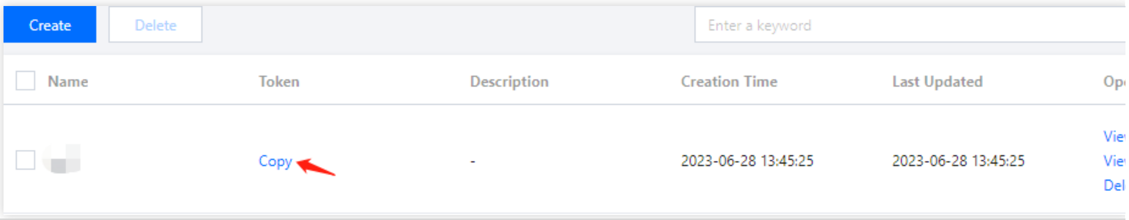
Parameter	Description
TAG	Set the message tag.
ShardingKey	A partition field of sequential messages. Messages with the same ShardingKey will be sent to the same partition.

Step 4. Consume messages

Creating a consumer

```
    // Get the client
    MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

    // Get the topic consumer
    MQProducer consumer = mqClient.getConsumer(namespace, topicName, groupName, "TAG
```

Parameter	Description
topicName	Topic name, which can be copied under the **Topic** tab on the **Cluster** page in the console.
groupName	Producer group name, which can be copied under the **Group** tab on the **Cluster** page in the consol

namespace	Namespace name, which can be copied under the **Namespace** tab on the **Cluster** page in the cons  Basic Info Cluster Monitoring **Namespace** Topic Group; Create (1/10); Search by keyword; Namespace Name: sdaa rocketmq-[redacted] copy; Message Retention Period: 3 days; Description: -; Operation: Configure Permission Edit De; Total items: 1; 20 / page; 1
TAG	Subscribed tag.
endpoint	Cluster access address over HTTP, which can be obtained from **Access Address** in the **Operatio Cluster** page in the console.
secretKey	Role name, which can be copied on the Role Management page.
accessKey	Role token, which can be copied in the **Token** column on the Role Management page.  Create Delete; Enter a keyword; Name: [redacted]; Token: Copy; Description: -; Creation Time: 2023-06-28 13:45:25; Last Updated: 2023-06-28 13:45:25; Op: Vie Vie Del

Subscribing to messages

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

    try {
        // Long polling consumes messages sequentially. Although the messages obtai
        // For sequential consumption, as long as a message in a partition hasn't b
        // For a partition, the next batch of messages can only be consumed after a
        messages = consumer.consumeMessageOrderly(
                Integer.parseInt(batchSize),
                Integer.parseInt(waitSeconds)
        );
    } catch (Throwable e) {
        e.printStackTrace();
    }
    if (messages == null || messages.isEmpty()) {
        System.out.println(Thread.currentThread().getName() + ": no new message, co
        continue;
    }

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

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

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

Parameter	Description
batchSize	The number of messages pulled at a time. Maximum value: 16.
waitSeconds	The polling waiting time for a message pull. Maximum value: 30 seconds.