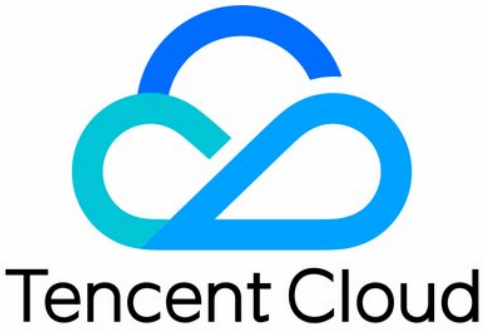
TDMQ for RabbitMQ

Quick Start



Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Quick Start
Using SDK to Send/Receive Message

Last updated: 2024-08-02 12:16:20

Operation scenarios

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

Prerequisites

- You have created the required resources.
- You have installed JDK 1.8 or later.
- You have installed Maven 2.5 or later.
- Download the demo

Operation step

Step 1. Install the Java dependency library

Add the following dependencies to pom.xml:

```
<!-- in your <dependencies> block -->
<dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>5.13.0</version>
</dependency>
```

Step 2. Produce a message

Compile and run MessageProducer.java.

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.tencent.tdmq.demo.cloud.Constant;
```

```
/**

 * Message producer
   */
   public class MessageProducer {

   /**

    * Exchange name
      */
      private static final String EXCHANGE_NAME = "exchange_name";

   public static void main(String[] args) throws Exception {
       // Connection factory
       ConnectionFactory factory = new ConnectionFactory();
       // Set the service address (replace with the access point address
copied in the console)
       factory.setUri("amqp://***");
       // Set virtual hosts (copy the complete vhost name from the open-
source RabbitMQ console)
       factory.setVirtualHost(VHOST_NAME);
       // Set the username (user name in the permission configuration of
the vhost in the open-source RabbitMQ console)
       factory.setUsername(USERNAME);
       // Set the password (key for the corresponding user)
       factory.setPassword("****");
       // Get the connection address and establish the channel
       try (Connection connection = factory.newConnection(); Channel
channel = connection.createChannel()) {
           // Bind the message exchange (EXCHANGE_NAME must exist in the
TDMQ for RabbitMQ console, and the exchange type must be the same as
that in the console)
           channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
           for (int i = 0; i < 10; i++) {
               String message = "this is rabbitmq message " + i;
               // Publish a message to the exchange, which will
automatically deliver the message to the corresponding queue
               channel.basicPublish(EXCHANGE_NAME, "", null,
message.getBytes());
               System.out.println(" [producer] Sent '" + message + "'");
           }
       } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
    }
```

Parameter	Description
EXCHANGE_NAME	Exchange name, which can be obtained from the exchange list in the console.
factory.setUri	Cluster access address, obtained from the **Client Access** section on the cluster basic information page. 
fac	Vhost name, obtained from the Vhost list in the console.

factory.setVirtualHost	
factory.setUsername	Username, as created in the console.
factory.setP	User password, as set when creating the user in the console.

a s s w o r d

Step 3. Consume the message

Compile and run MessageConsumer.java.

```
import com.rabbitmq.client.AMQP;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;
import com.tencent.tdmq.demo.cloud.Constant;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**

 * Message consumer
   */
   public class MessageConsumer1 {

   /**

    * Queue name
      */
      public static final String QUEUE_NAME = "queue_name";

   /**

    * Exchange name
      */
      private static final String EXCHANGE_NAME = "exchange_name";

   public static void main(String[] args) throws Exception {
       // Connection factory
       ConnectionFactory factory = new ConnectionFactory();
```

```
        // Set the service address (replace with the access point address
copied in the console)
        factory.setUri("amqp://***");
        // Set virtual hosts (copy the complete vhost name from the open-
source RabbitMQ console)
        factory.setVirtualHost(VHOST_NAME);
        // Set the username (user name in the permission configuration of
the vhost in the open-source RabbitMQ console)
        factory.setUsername(USERNAME);
        // Set the password (key for the corresponding user)
        factory.setPassword("****");
        // Get the connection address
        Connection connection = factory.newConnection();
        // Establish a channel
        Channel channel = connection.createChannel();
        // Bind the message exchange
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
        // Declare the queue message
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);
        // Bind the message exchange (EXCHANGE_NAME must exist in the
TDMQ for RabbitMQ console, and the exchange type must be the same as
that in the console)
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");
        System.out.println(" [Consumer1] Waiting for messages.");
        // Subscribe to the message
        channel.basicConsume(QUEUE_NAME, false, "ConsumerTag", new
DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope
envelope,
                                       AMQP.BasicProperties properties,
byte[] body)
                    throws IOException {
                // Process received messages with business logic.
                System.out.println("Received: " + new String(body,
StandardCharsets.UTF_8) + ", deliveryTag: " + envelope.getDeliveryTag()
+ ", messageId: " + properties.getMessageId());
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        });
    }
    }
```

Parameter	Description
QUEUE_NAME	Queue name, which can be obtained from the queue list in the console.
EXCHANGE_NAME	Exchange name, which can be obtained from the exchange list in the console.
factory.set	Cluster access address, obtained from the **Client Access** section on the cluster basic i page. 

Uri	
factory.setVirtualHost	Vhost name, obtained from the Vhost list in the console.
factory.setUsername	Username, as created in the console.
factory	User password, as set when creating the user in the console.

r y. s e t P a s s w o r d