消息队列 RabbitMQ 版

SDK 文档

产品文档

腾讯云

文档目录

SDK 文档
Spring Boot Starter 接入

最近更新时间：2024-01-03 11:45:32

操作场景

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

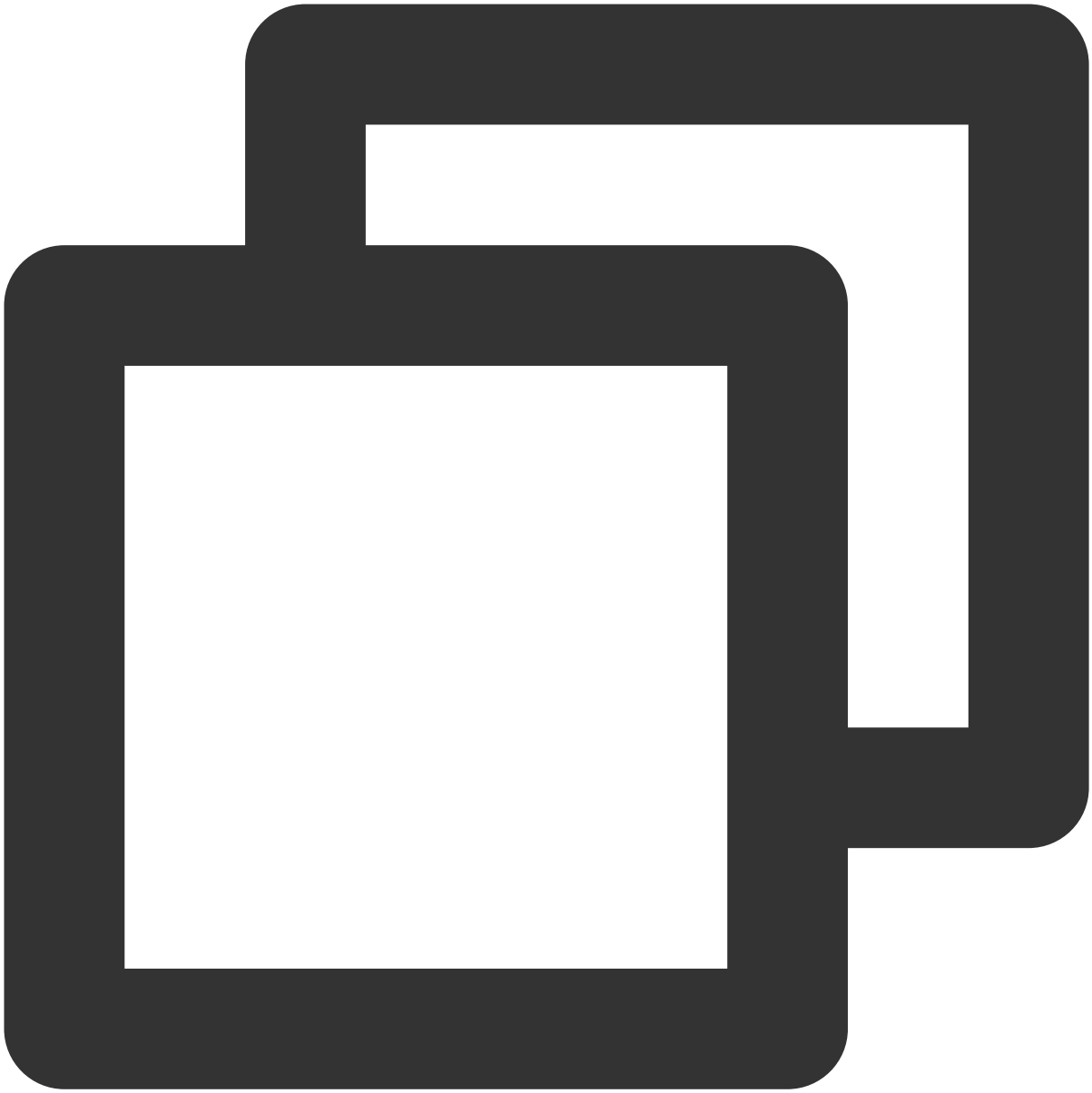
前提条件

完成资源创建与准备
安装1.8或以上版本 JDK
安装2.5或以上版本 Maven
下载 Demo

操作步骤

步骤1：添加依赖

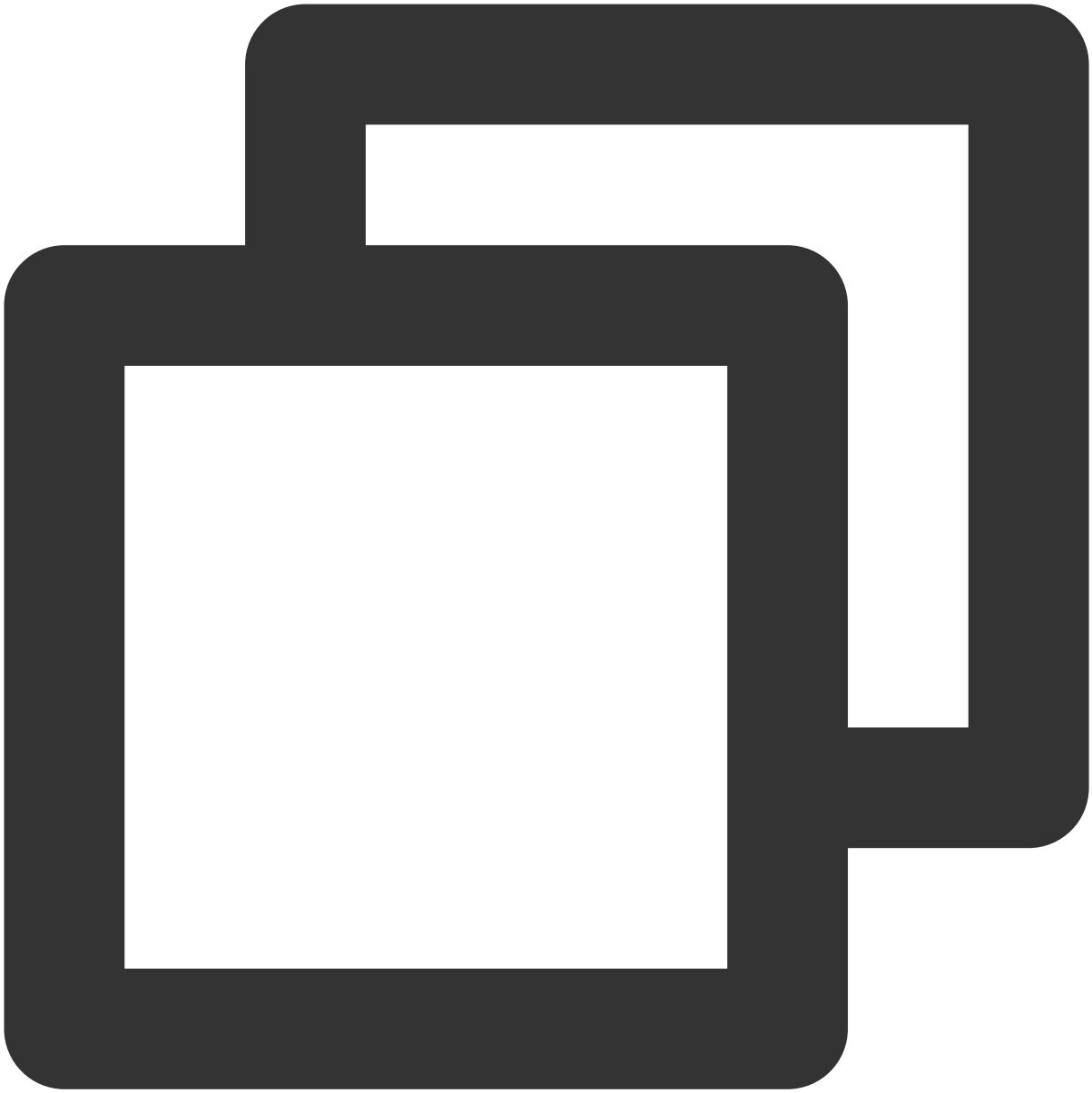
在 pom.xml 中添加依赖。



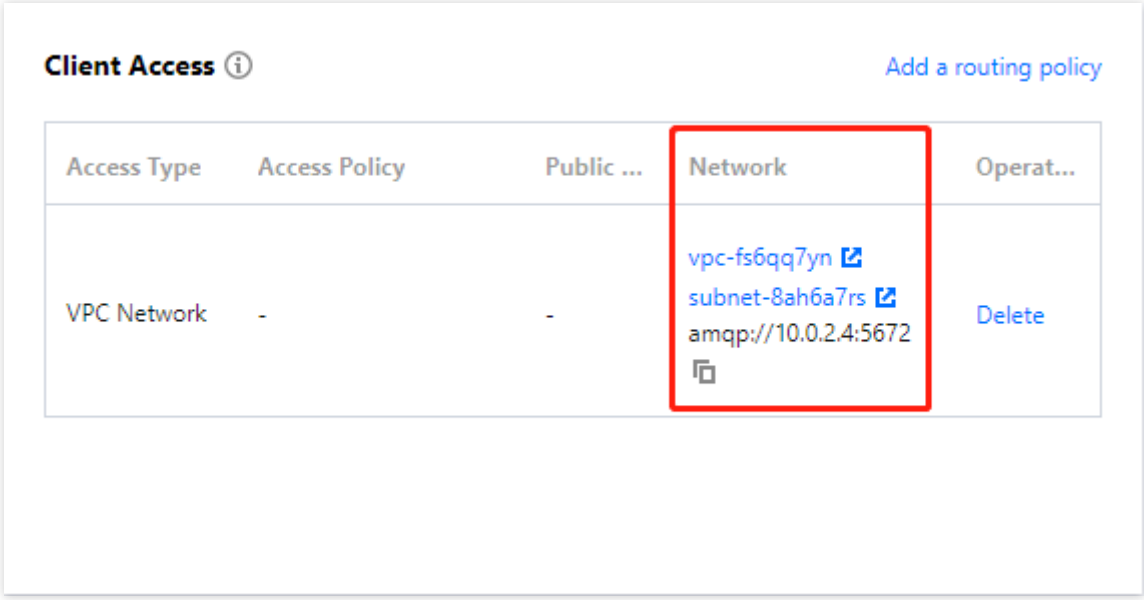
```
<!--rabbitmq-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

步骤2：准备配置

1. 在配置文件中加入 RabbitMQ 配置信息（以 yaml 配置为例）。



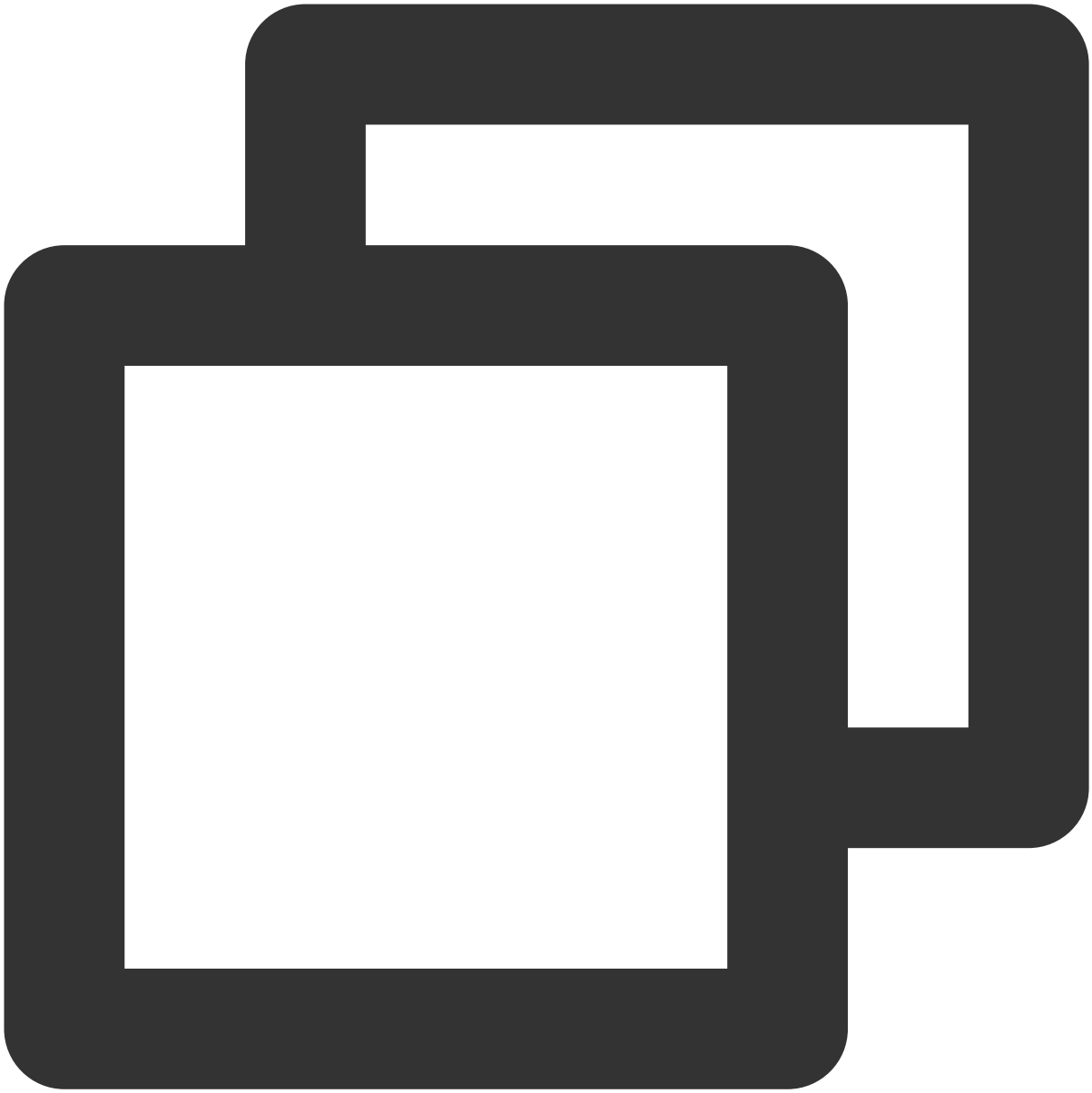
```
spring:
  rabbitmq:
      # host地址可在控制台中获取 或 使用Rabbitmq服务地址
    host: amqp-xx.rabbitmq.x.tencenttdmq.com
    port: 5672
    # 要使用的角色名称 可在角色管理控制台获取
    username: admin
    # 角色密钥
    password: eyJrZXlJZ....
    # Vhost全称，可在集群控制台Vhost tab页获取
    virtual-host: amqp-xxx|Vhost
```

参数	说明
host	集群接入地址，在集群基本信息页面的**客户端接入**模块获取。 
port	集群接入地址端口，在集群基本信息页面的**客户端接入**模块获取。
username	用户名称，填写在控制台创建的用户名称。
password	用户密码，填写在控制台创建用户时填写的密码。
virtual-host	Vhost 名称，在控制台 Vhost 列表获取。

2. 创建配置文件加载程序（以 `Fanout交换机` 为例）。

说明

其他类型的交换机配置可参见具体 Demo 示例。



```
/**
 * Fanout交换机配置
 */
@Configuration
public class FanoutRabbitConfig {

    /**
     * 交换机
     */
    @Bean
    public FanoutExchange fanoutExchange() {
```

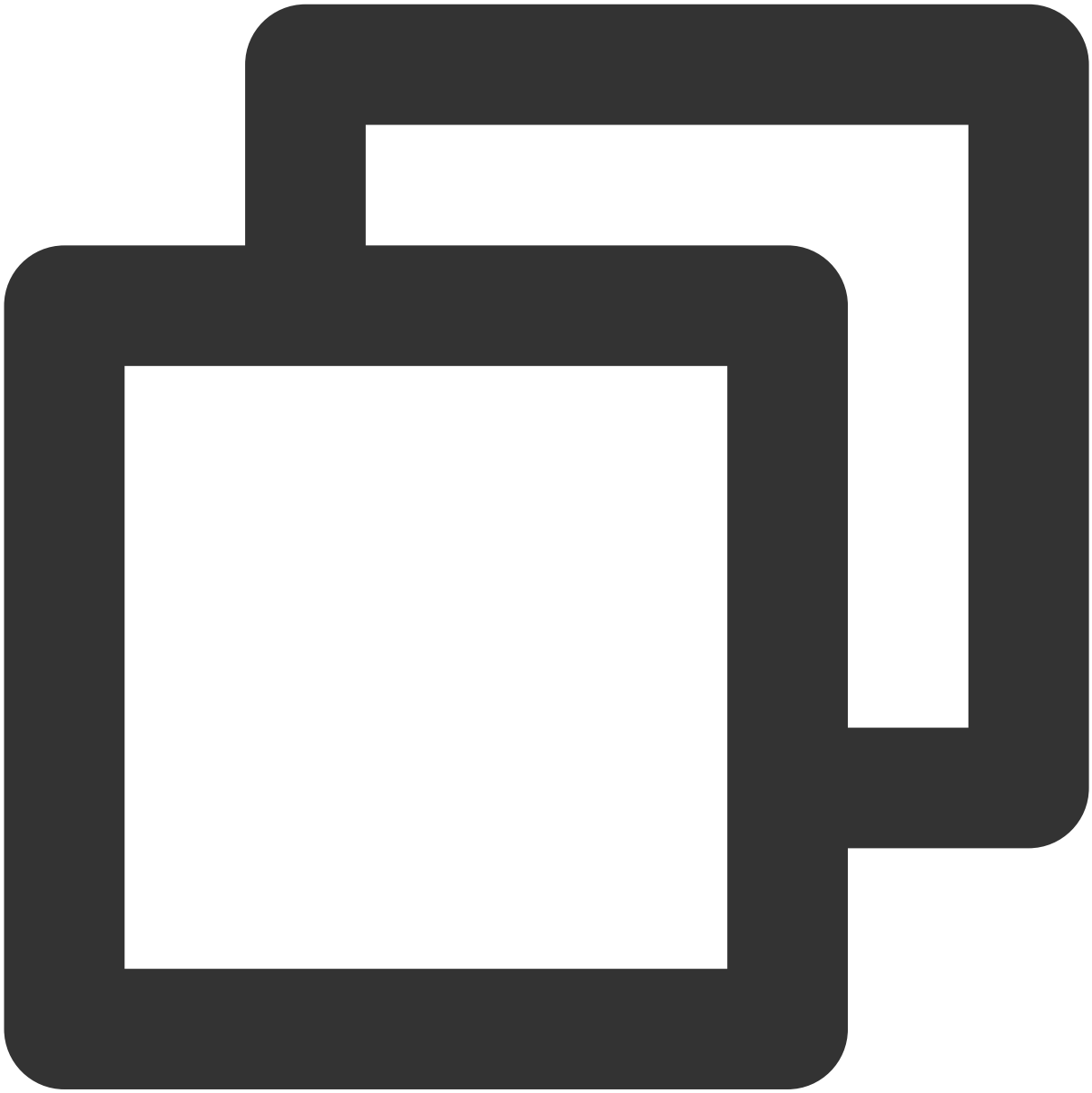
```
        return new FanoutExchange("fanout-logs", true, false);
    }


    /**
     * 消息队列
     */
    @Bean
    public Queue fanoutQueueA() {
        return new Queue("ps_queue", true);
    }

    @Bean
    public Queue fanoutQueueB() {
        // 可通过这种方式绑定死信队列
        Map<String, Object> requestParam = new HashMap<>();
        requestParam.put("x-dead-letter-exchange", "my-deadLetter-exchange");
        // 设置消息过期时长
        requestParam.put("x-message-ttl", 60000);
        return new Queue("ps_queue1", true, false,true, requestParam);
    }

    /**
     * 绑定消息队列与交换机
     */
    @Bean
    public Binding bindingFanoutA() {
        return BindingBuilder.bind(fanoutQueueA())
                .to(fanoutExchange());
    }

    @Bean
    public Binding bindingFanoutB() {
        return BindingBuilder.bind(fanoutQueueB())
                .to(fanoutExchange());
    }
}
```

参数	说明
fanout-logs	绑定的 Exchange 名称，在控制台 Exchange 列表获取。
ps_queue	与 Exchange 绑定的第一个 Queue名称，在控制台 Queue 列表获取。
my-deadLetter-exchange	死信 Exchange 名称，在控制台 Exchange 列表获取。

ps_queue1	与 Exchange 绑定的第二个 Queue 名称，在控制台 Queue 列表获取。

步骤3：发送消息

创建并编译消息发送程序 DemoApplication.java，使用 RabbitTemplate 发送即可。

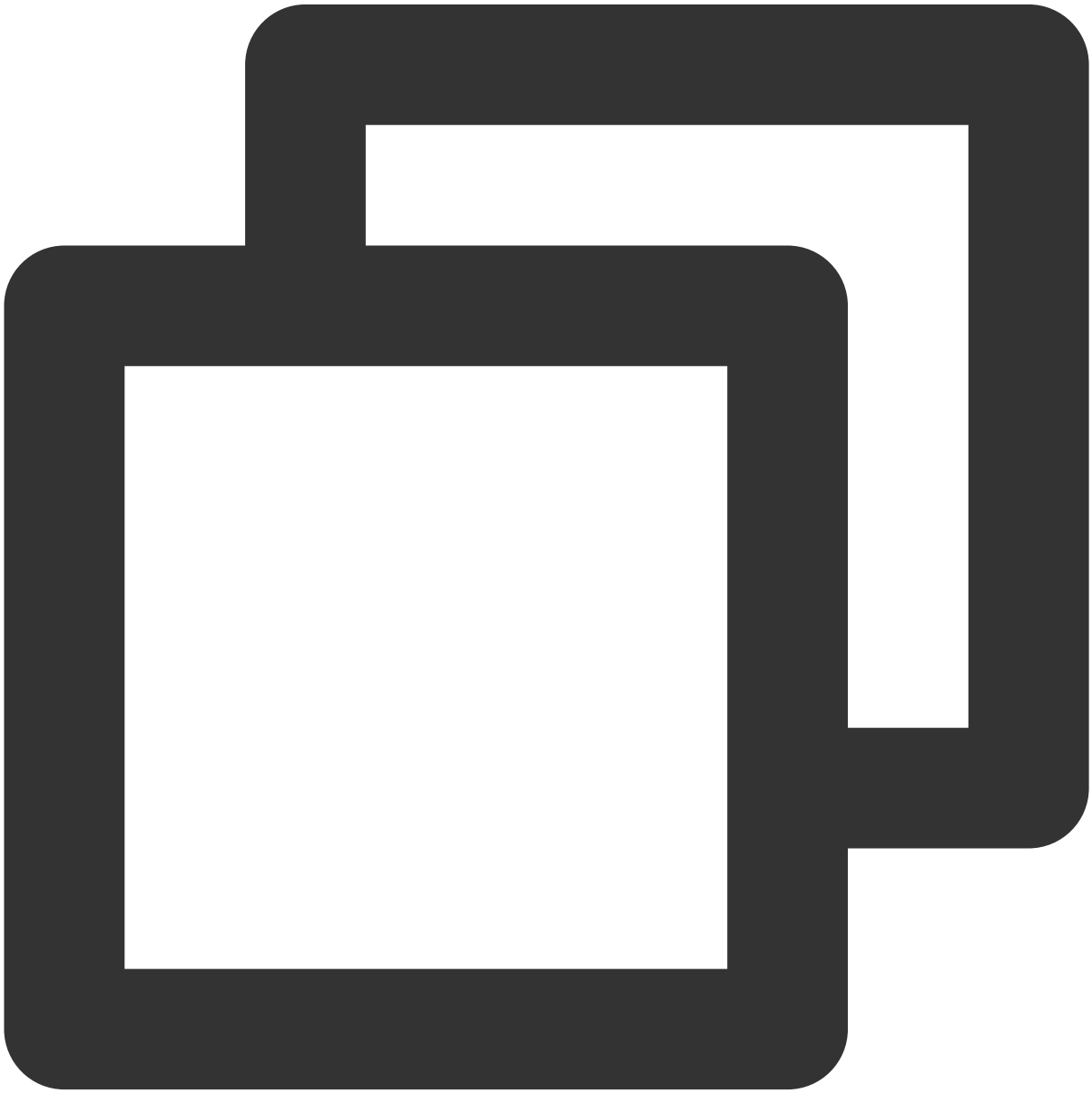


```
@Autowired
private RabbitTemplate rabbitTemplate;

public String send() {
    String msg = "This is a new message.";
```

```
    // 发送消息
    // 参数说明：参数1：交换机名称，在控制台 Exchange 列表获取。  参数2：routing key  参数3：
    rabbitTemplate.convertAndSend("direct_logs", "", msg);
    return "success";
}
```

步骤4：消费消息

创建并编译消息接收程序 FanoutReceiver.java。（以 `Fanout交换机` 为例。）

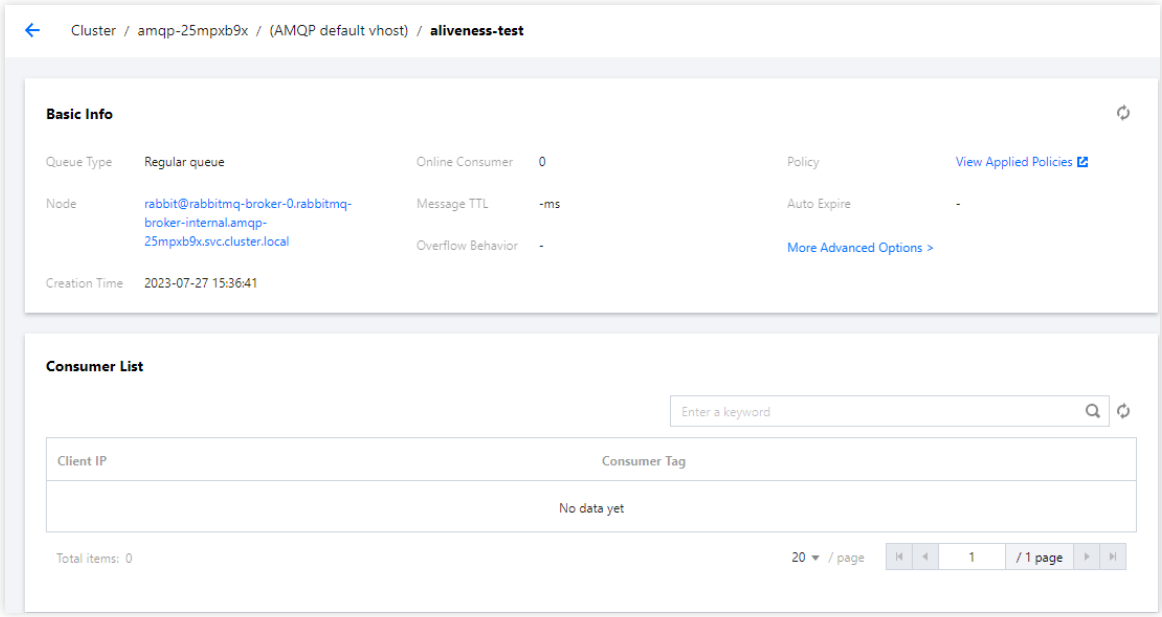


```
@Component
```

```
public class FanoutReceiver {
 // 注册一个listener监听指定消息队列
 @RabbitHandler
 @RabbitListener(queues = "ps_queue")  //与 Exchange 绑定的 Queue名称，在控制台 Queue 列
 public void listenerPsQueue(String msg) {
     // 业务处理...
     System.out.println("(ps_queue) receiver message. [" + msg + "]");
 }
}
```

步骤5：查看消息

如果您想确认消息是否成功发送至 TDMQ RabbitMQ 版，可以在控制台 集群管理 > **Queue** 的基本信息页面查看接入的消费者情况。



说明

其他使用示例请参见 Spring AMQP 官网。

Spring Cloud Stream 接入

最近更新时间：2024-01-03 11:45:32

操作场景

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

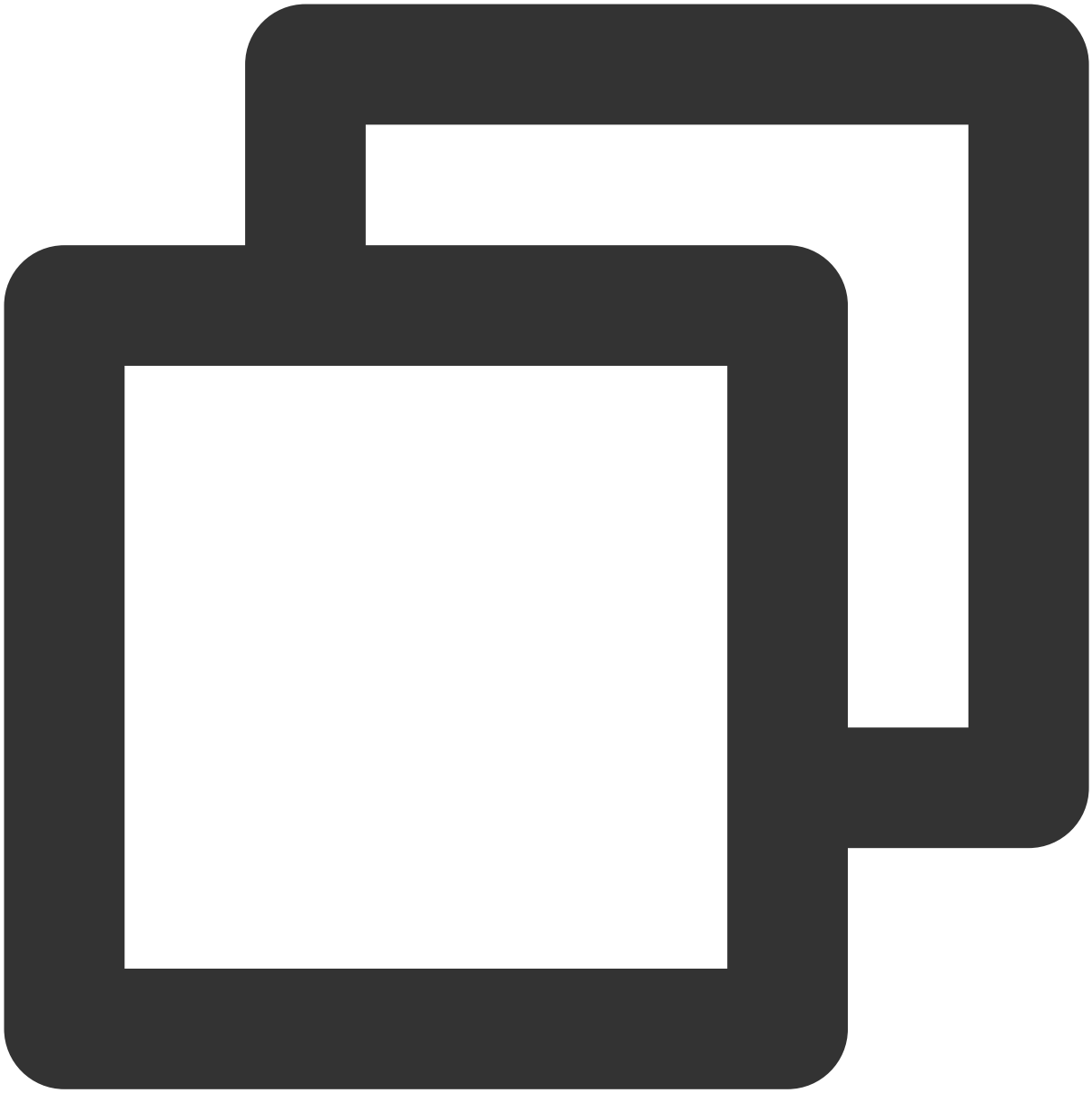
前提条件

完成资源创建与准备
安装1.8或以上版本 JDK
安装2.5或以上版本 Maven
下载 Demo

操作步骤

步骤1：添加依赖

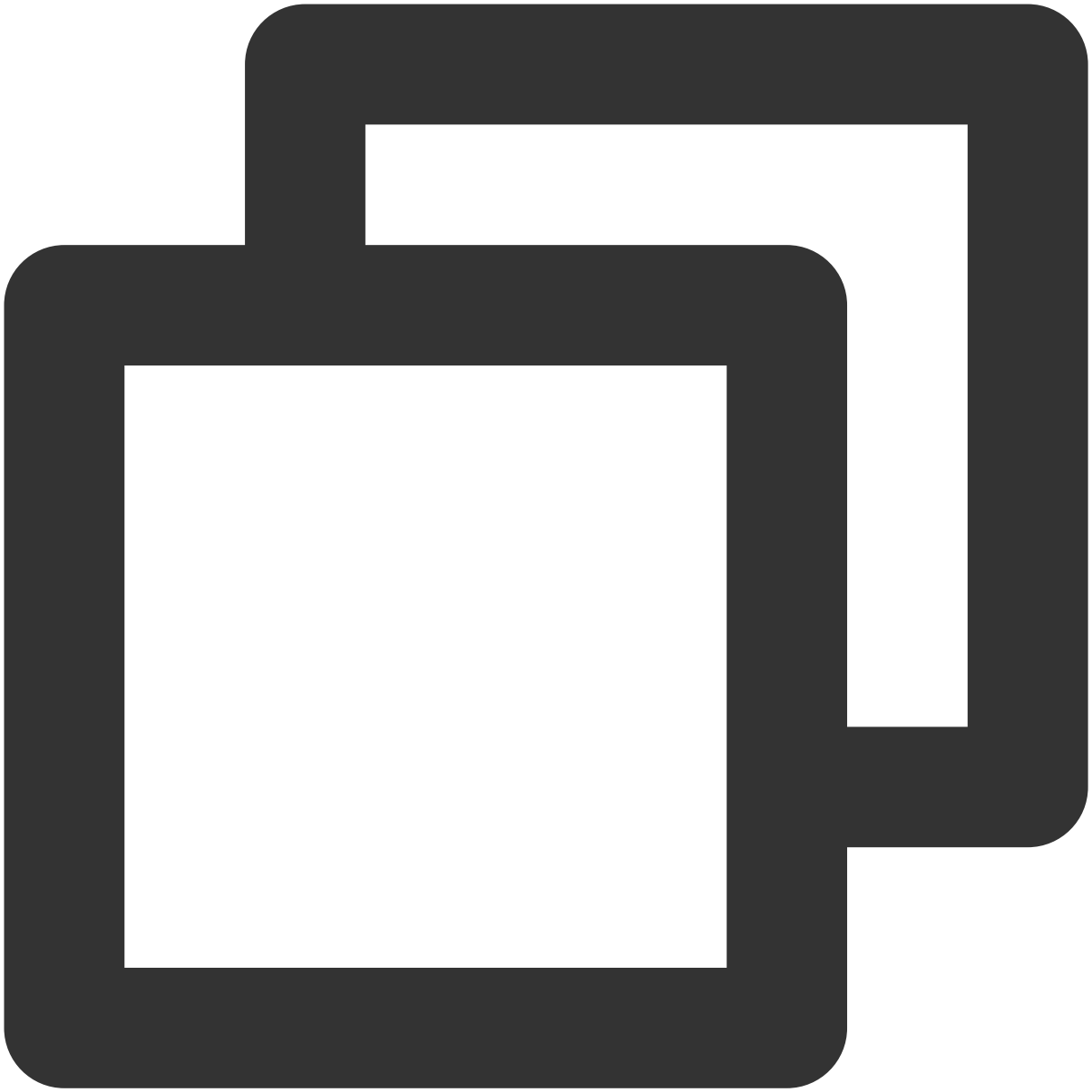
在 pom.xml 中添加 `Stream RabbitMQ` 相关依赖。



```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>
```

步骤2：准备配置

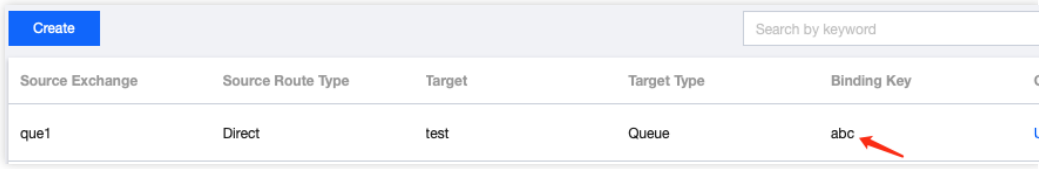
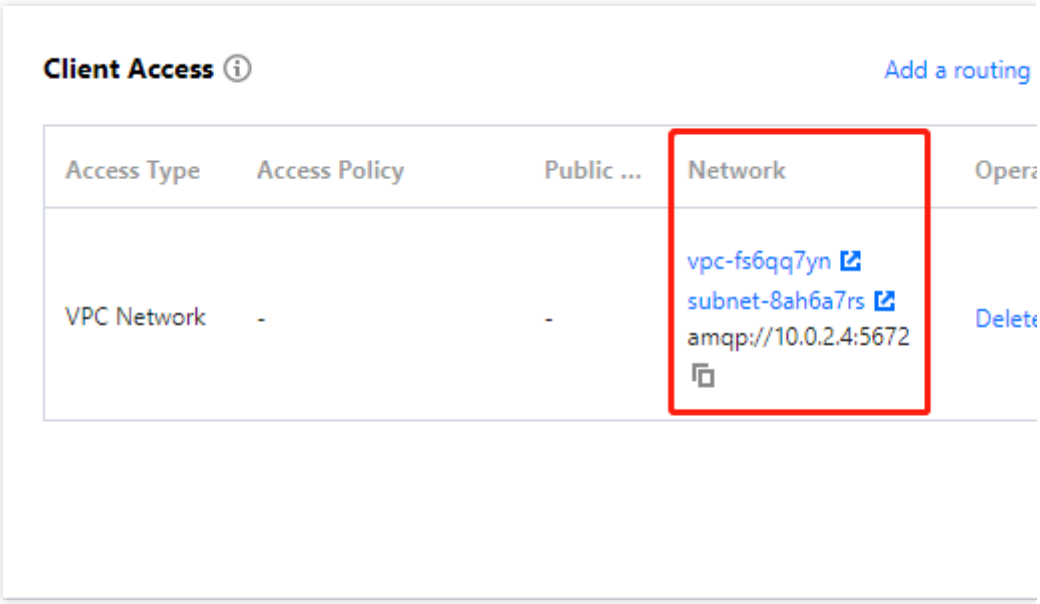
1. 在配置文件中进行相应配置（以 `direct` 交换机配置为例）。



```
spring:
    application:
    name: application-name
    cloud:
    stream:
      rabbit:
        bindings:
          # 输出channel名称
          output:
            # 生产者配置信息
            producer:
```

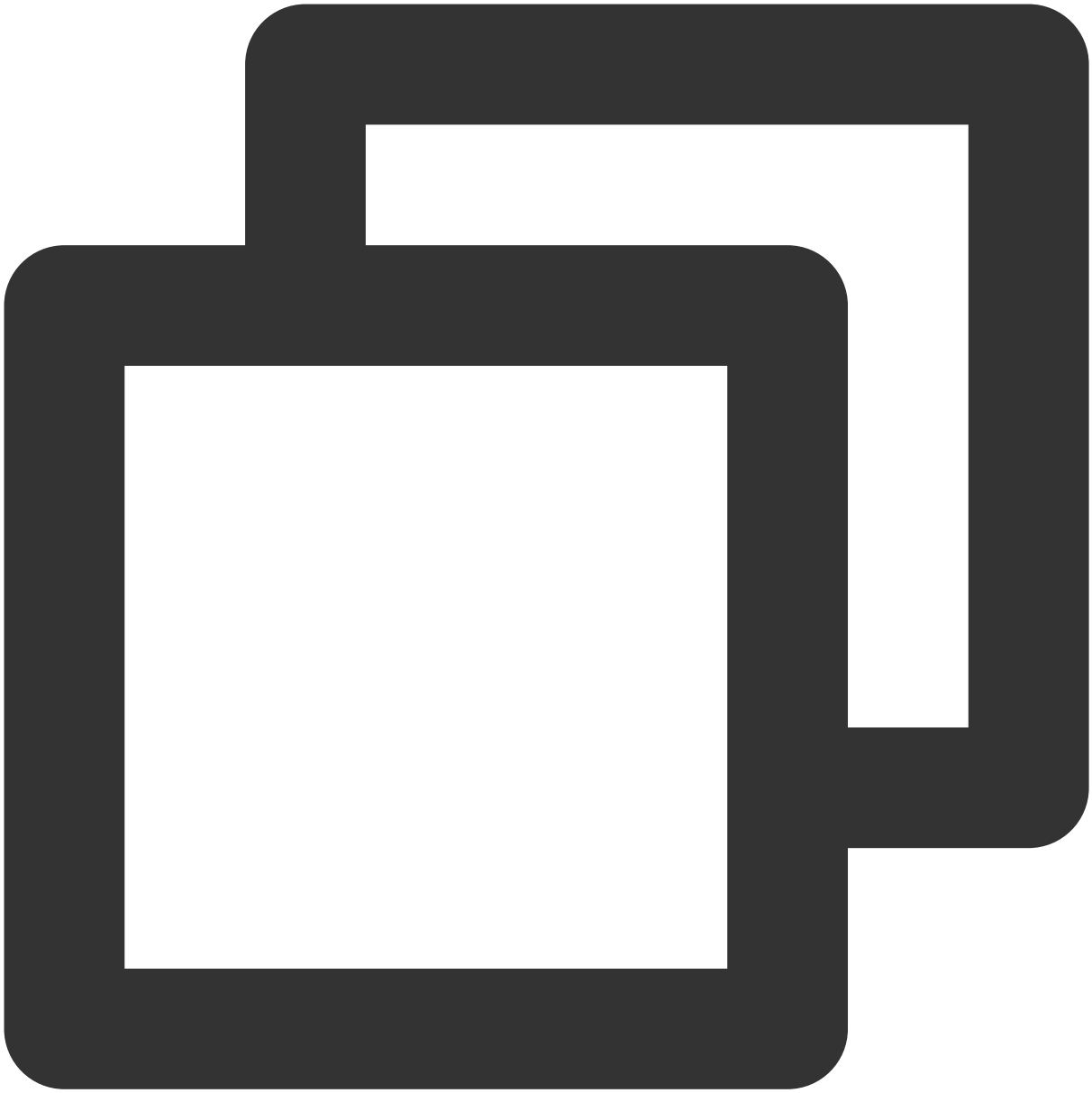
```
            # 生产者使用的交换机类型    如果已存在交换机名称，该类型必须与交换机类型一致
            exchangeType: direct
            # 用于指定 routing key 表达式
            routing-key-expression: headers["routeTo"] # 该值表示使用头信息的routeTo
            queueNameGroupOnly: true
        # 输入channel名称
        input:
          # 消费者配置信息
          consumer:
            # 消费者使用的交换机类型    如果已存在交换机名称，该类型必须与交换机类型一致
            exchangeType: direct
            # 消费者消息队列绑定的 routing key
            bindingRoutingKey: info,waring,error
            # 该配置会对上面的 routing key 进行处理
            bindingRoutingKeyDelimiter: ","  # 该配置表示：使用,切割上面配置的routing
            # 消息确认模式    具体查看AcknowledgeMode
            acknowledge-mode: manual
            queueNameGroupOnly: true
    bindings:
        # 输出channel名称
      output: #通道的名称
        destination: direct_logs #要使用的exchange名称
        content-type: application/json
        default-binder: dev-rabbit
      # 输入channel名称
      input: #通道的名称
        destination: direct_logs #要使用的exchange名称
        content-type: application/json
        default-binder: dev-rabbit
        group: route_queue1 # 要使用的消息队列名称
    binders:
      dev-rabbit:
        type: rabbit
        environment:
          spring:
            rabbitmq:
              host: amqp-xxx.rabbitmq.xxx.tencenttdmq.com #集群接入地址，在集群管理页
              port: 5672
              username: admin #角色名称
              password: password #角色密钥
              virtual-host: vhostnanme #Vhost名称
```

参数	说明
bindingRoutingKey	消费者消息队列绑定的 routing key，消息的路由规则，在控制台绑定关系列表的绑定 Key列

	
direct_log	Exchange 名称，在控制台 Exchange 列表获取。
route_queue1	Queue名称，在控制台 Queue 列表获取。
host	集群接入地址，集群接入地址，在集群基本信息页面的**客户端接入**模块获取。 
port	集群接入地址端口，在集群管理页面操作列的获取接入地址获取。
username	用户名称，填写在控制台创建的用户名称。
password	用户密码，填写在控制台创建用户时填写的密码。
virtual-host	Vhost 名称，在控制台 Vhost 列表获取。

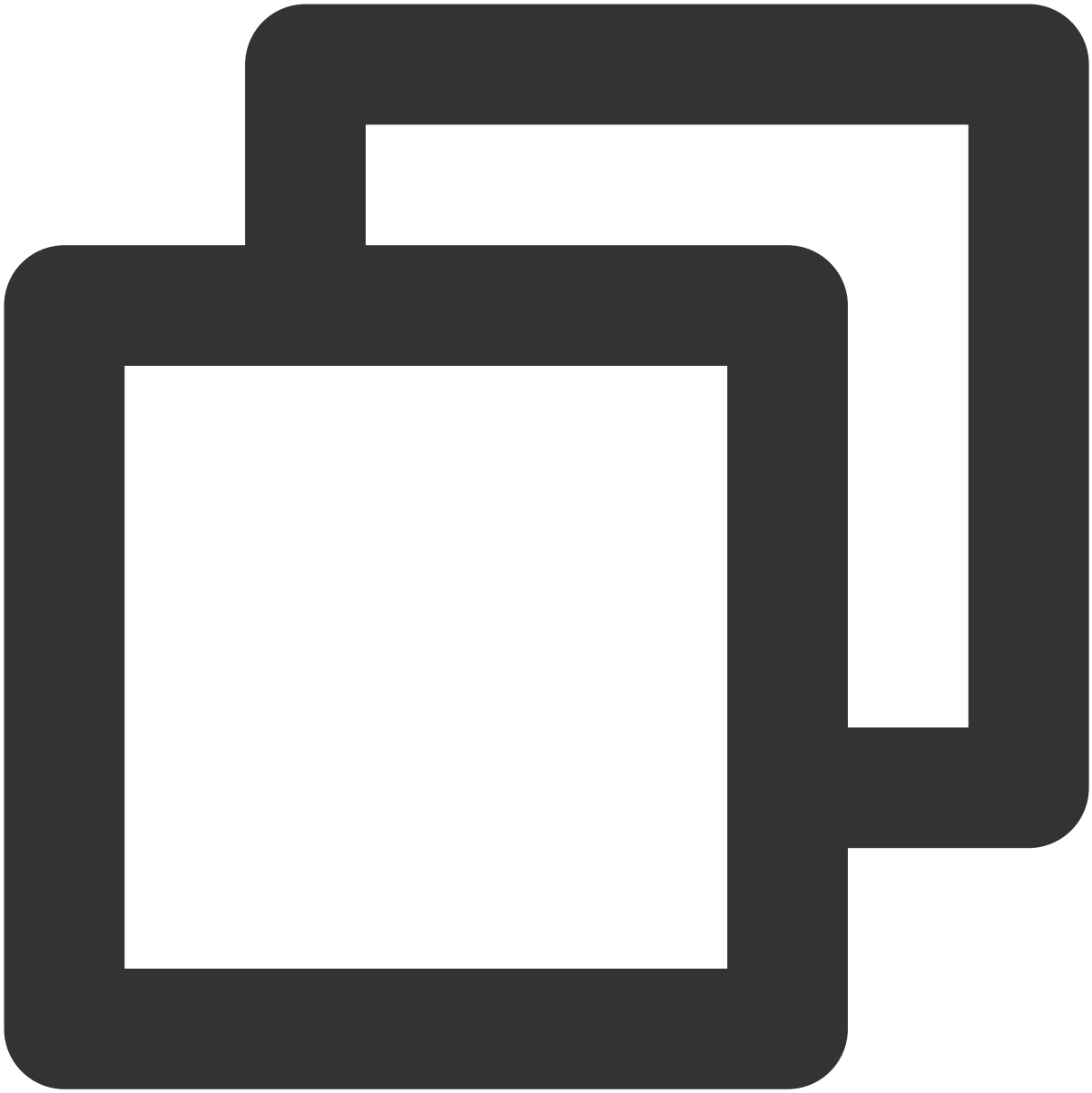
2. 创建配置文件加载程序。

OutputMessageBinding.java



```
public interface OutputMessageBinding {
  /**
     * 要使用的通道名称(输出channel名称)
     */
  String OUTPUT = "output";

  @Output(OUTPUT)
  MessageChannel output();
}
```

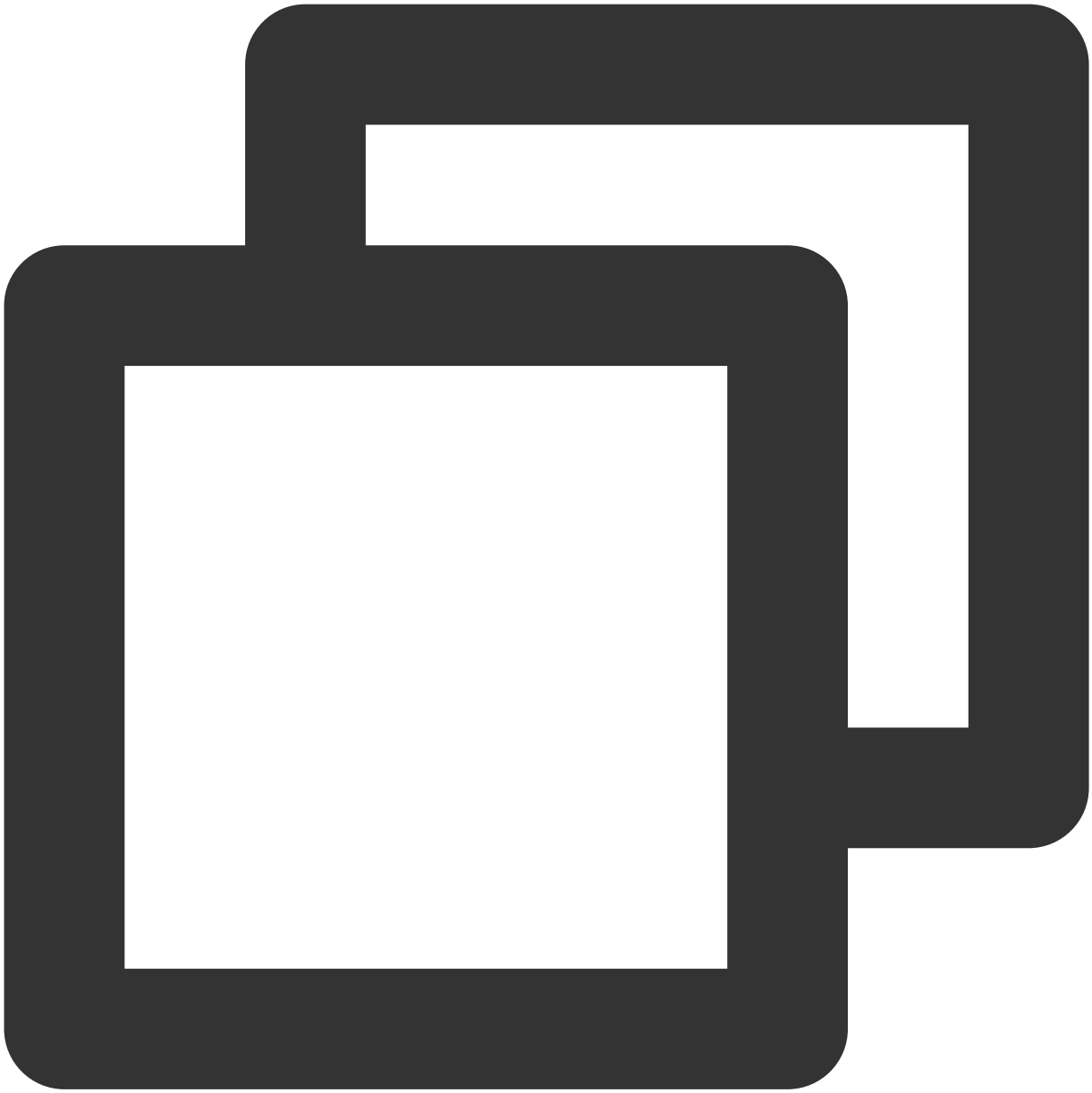
InputMessageBinding.java



```
public interface InputMessageBinding {

  /**
     * 要使用的通道名称
     */
  String INPUT = "input";

  @Input(INPUT)
  SubscribableChannel input();
}
```

步骤3：发送消息

创建并编译消息发送程序 IMessageSendProvider.java。



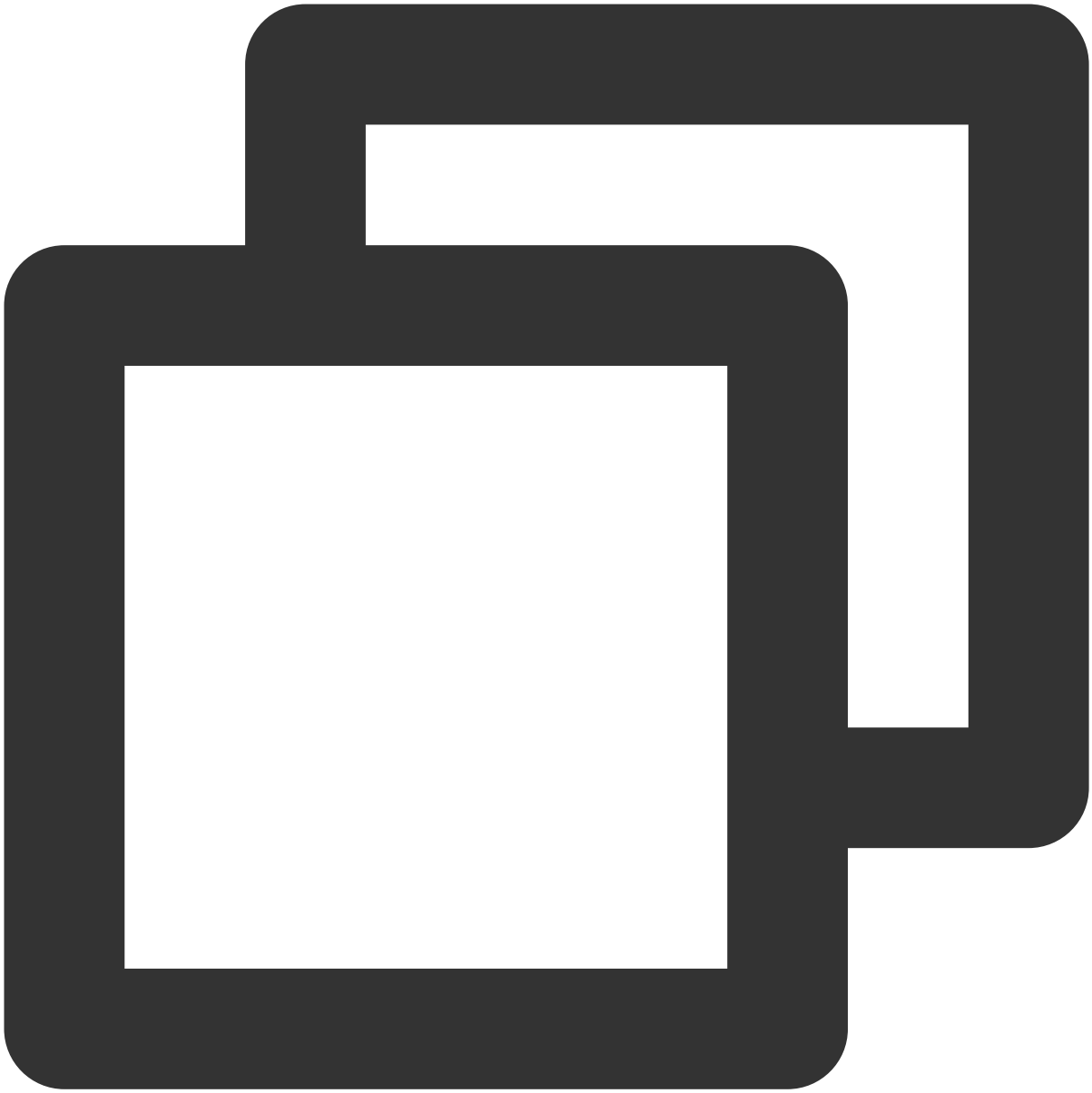
```
// 引入配置类
@EnableBinding(OutputMessageBinding.class)
public class MessageSendProvider {

    @Autowired
    private OutputMessageBinding outputMessageBinding;

    public String sendToDirect() {
```

```
            outputMessageBinding.output().send(MessageBuilder.withPayload("[info] This
            outputMessageBinding.output().send(MessageBuilder.withPayload("[waring] Thi
            outputMessageBinding.output().send(MessageBuilder.withPayload("[error] This
            return "success";
        }

        public String sendToFanout() {
            for (int i = 0; i < 3; i++) {
                outputMessageBinding.output().send(MessageBuilder.withPayload("This is
            }
            return "success";
        }
    }
```

在要发送消息的类中注入 `MessageSendProvider` 即可进行发送消息。

步骤4：消费消息

创建并编译消息消费程序 MessageConsumer.java。**可配置多个通道，可对不同消息队列的监听。**

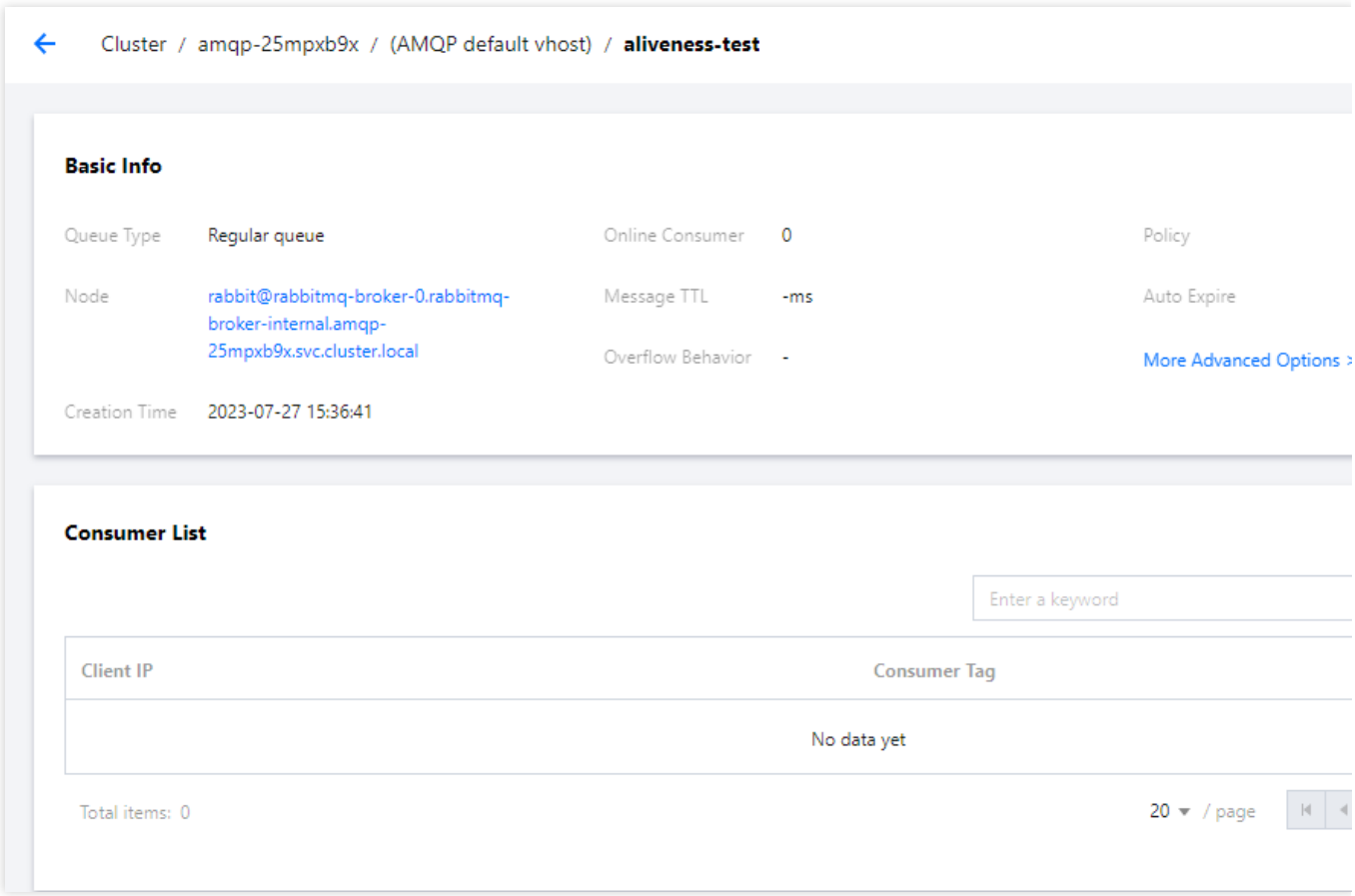


```
@Service
@EnableBinding(InputMessageBinding.class)
public class MessageConsumer {

    @StreamListener(InputMessageBinding.INPUT)
    public void test(Message<String> message) throws IOException {
        Channel channel = (com.rabbitmq.client.Channel) message.getHeaders().get(Am
        Long deliveryTag = (Long) message.getHeaders().get(AmqpHeaders.DELIVERY_TAG
        channel.basicAck(deliveryTag, false);
        String payload = message.getPayload();
        System.out.println(payload);
```

```
        }
    }
```

步骤5：查看消息

如果您想确认消息是否成功发送至 TDMQ RabbitMQ 版，可以在控制台 集群管理 > **Queue** 基本信息页面查看接入的消费者情况。



说明

上述是基于 RabbitMQ 的发布订阅模型的一个简单示例，可根据实际使用进行不同配置，具体可参见 Demo 示例 或 Spring cloud stream 官网。

Java SDK

最近更新时间：2024-01-03 11:45:32

操作场景

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

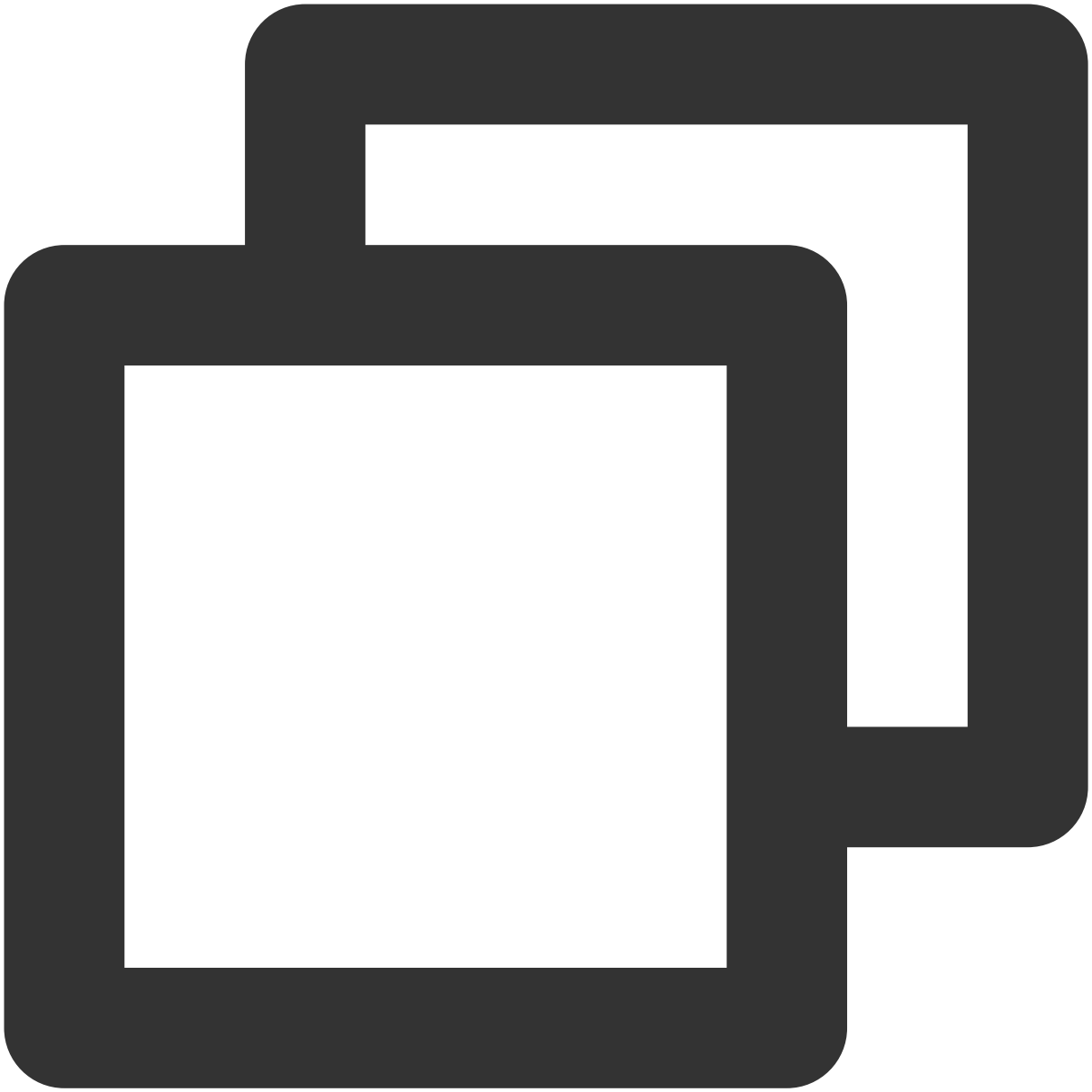
前提条件

完成资源创建与准备
安装1.8或以上版本 JDK
安装2.5或以上版本 Maven
下载 Demo

操作步骤

步骤1：安装 Java 依赖库

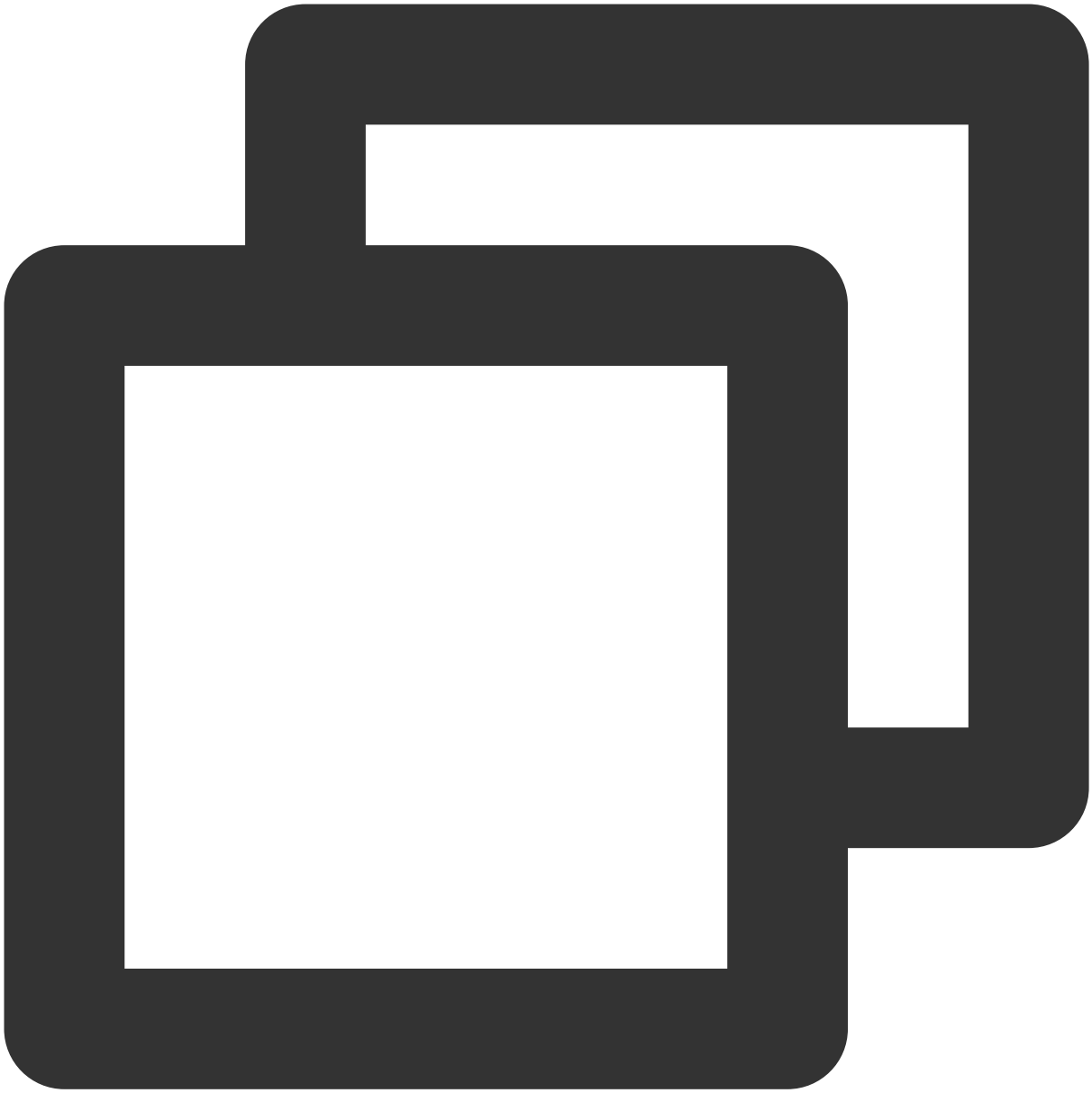
在 pom.xml 添加以下依赖：



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

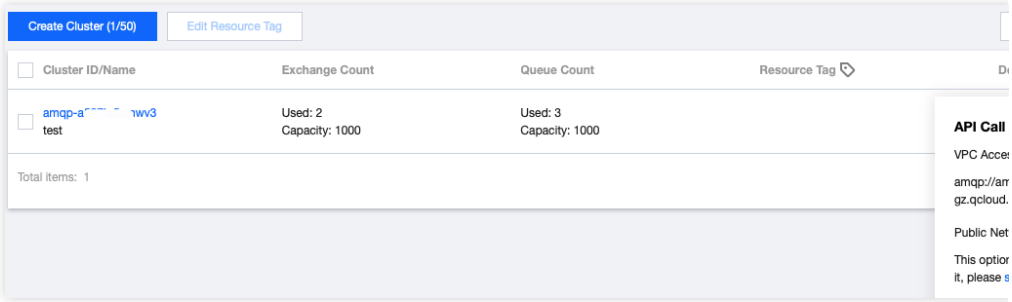
步骤2：生产消息

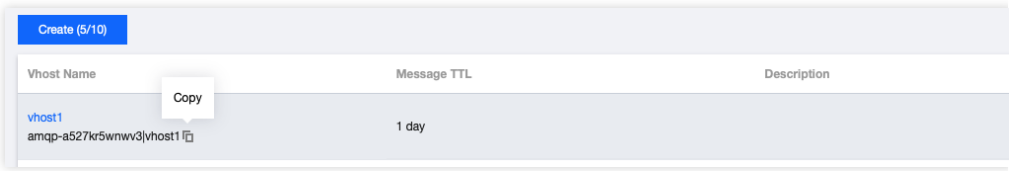
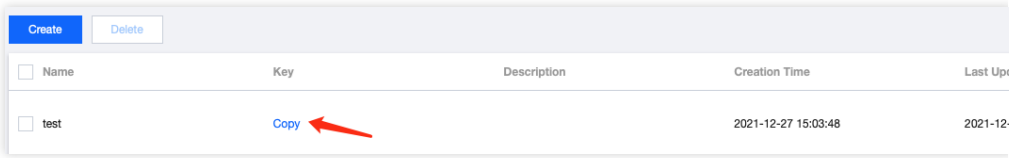
创建并编译运行 MessageProducer.java。



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

/**
 * 消息生产者
 */
public class MessageProducer {

    /**
```

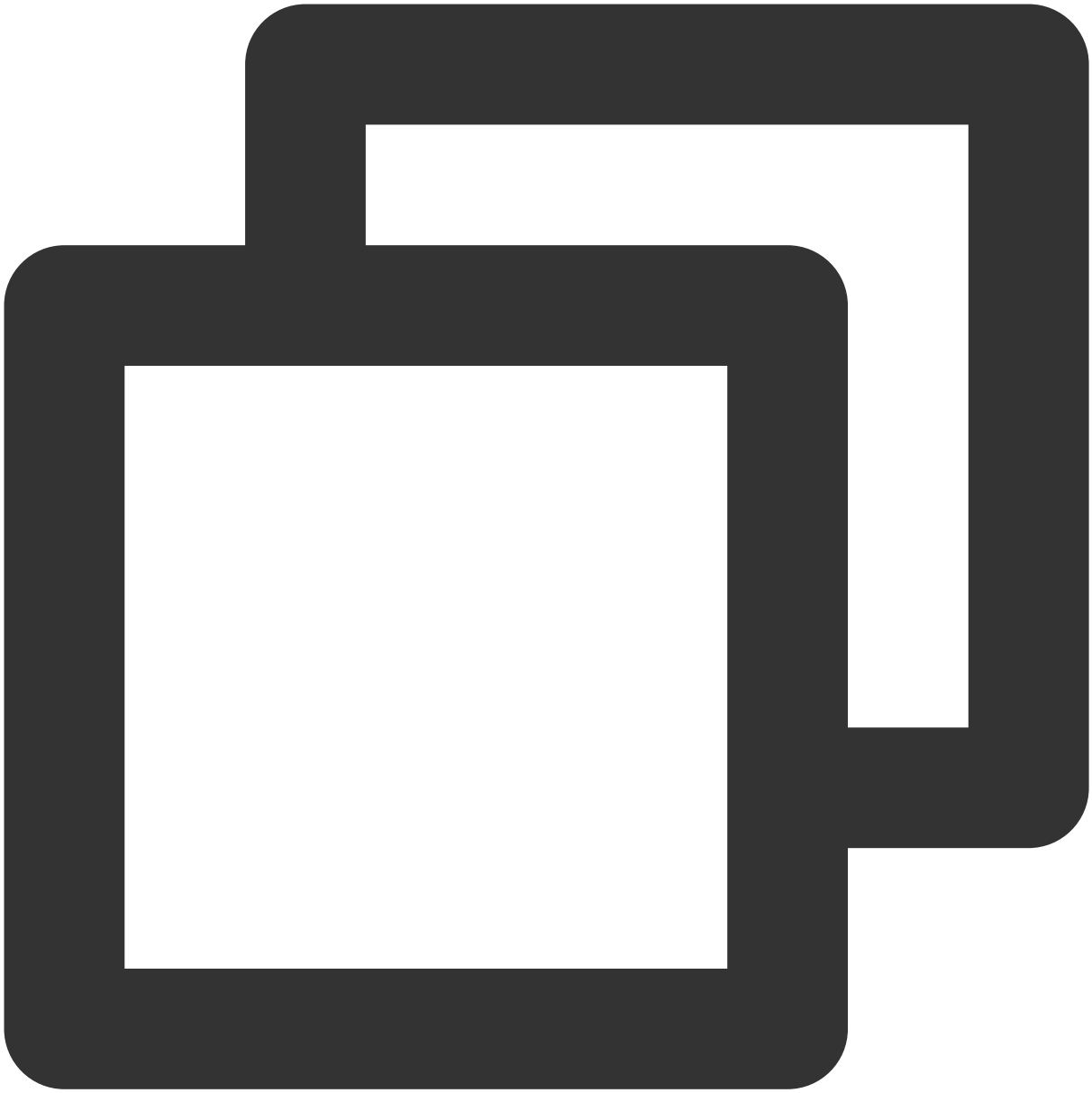
```
     * 交换机名称
     */
    private static final String EXCHANGE_NAME = "exchange_name";

    public static void main(String[] args) throws Exception {
        // 连接工厂
        ConnectionFactory factory = new ConnectionFactory();
        // 设置服务地址（完整复制控制台接入点地址）
        factory.setUri("amqp://***");
        // 设置Virtual Hosts（Vhost控制台复制完整Vhost名称）
        factory.setVirtualHost(VHOST_NAME);
        // 设置用户名（具体使用Vhost的配置权限中的角色名称）
        factory.setUsername(USERNAME);
        // 设置密码（对应角色的密钥）
        factory.setPassword("eyJh****");
        // 获取连接、建立通道
        try (Connection connection = factory.newConnection(); Channel channel = con
            // 绑定消息交换机（EXCHANGE_NAME必须在消息队列RabbitMQ版控制台上已存在，并且Excl
            channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
            for (int i = 0; i < 10; i++) {
                String message = "this is rabbitmq message " + i;
                // 发布消息到交换机，交换机自动将消息投递到相应队列
                channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
                System.out.println(" [producer] Sent '" + message + "'");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

参数	说明
EXCHANGE_NAME	Exchange 名称，在控制台 Exchange 列表获取。
factory.setUri	集群接入地址，在**集群管理**页面操作列的**获取接入地址**获取。 

factory.setVirtualHost	Vhost 名称，在控制台 Vhost 页面复制，格式是“**集群 ID +	+ vhost 名称**”。 
factory.setUsername	角色名称，在 **角色管理** 页面复制。
factory.setPassword	角色密钥，在 **角色管理** 页面复制**密钥**列复制。 

步骤3：消费消息

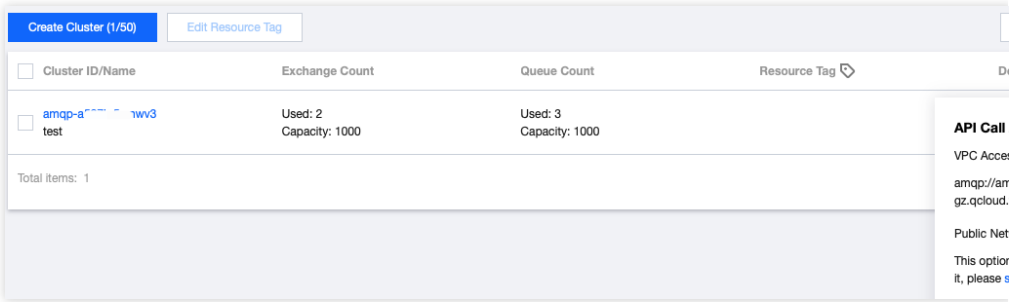
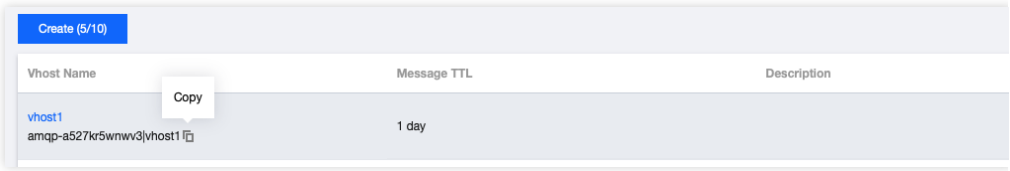
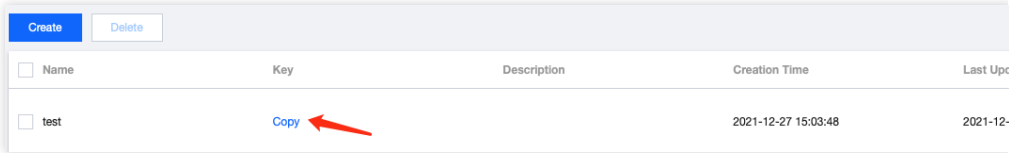
创建并编译运行 MessageConsumer.java。



```
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;
```

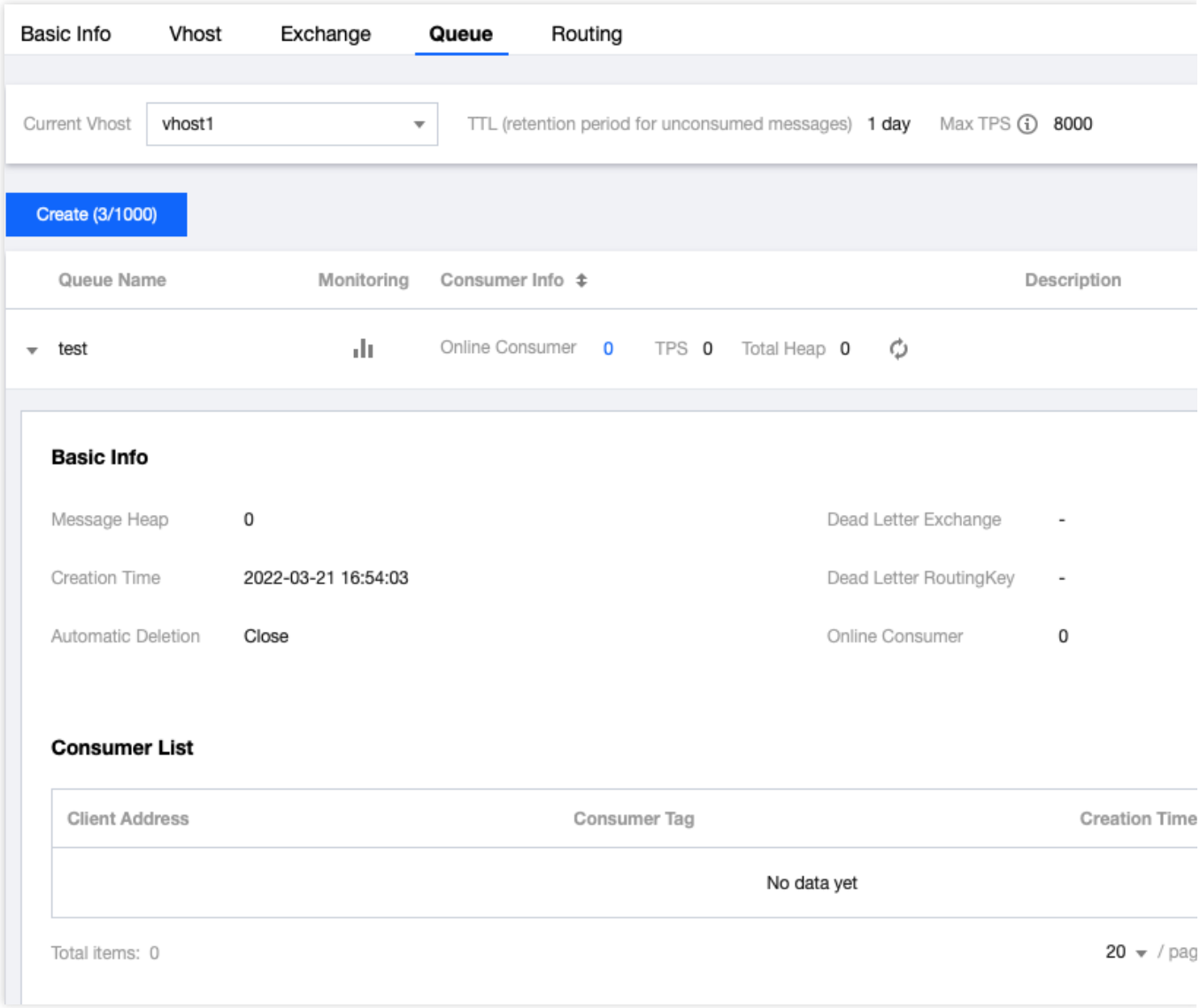
```
/**
 * 消息消费者
 */
public class MessageConsumer1 {

    /**
     * 队列名称
     */
    public static final String QUEUE_NAME = "queue_name";

    /**
     * 交换机名称
     */
    private static final String EXCHANGE_NAME = "exchange_name";

    public static void main(String[] args) throws Exception {
        // 连接工厂
        ConnectionFactory factory = new ConnectionFactory();
        // 设置服务地址（完整复制控制台接入点地址）
        factory.setUri("amqp://***");
        // 设置Virtual Hosts（Vhost控制台复制完整Vhost名称）
        factory.setVirtualHost(VHOST_NAME);
        // 设置用户名（具体使用Vhost的配置权限中的角色名称）
        factory.setUsername(USERNAME);
        // 设置密码（对应角色的密钥）
        factory.setPassword("eyJh****");
        // 获取连接
        Connection connection = factory.newConnection();
        // 建立通道
        Channel channel = connection.createChannel();
        // 绑定消息交换机
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
        // 声明队列信息
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);
        // 绑定消息交换机（EXCHANGE_NAME必须在消息队列RabbitMQ版控制台上已存在，并且Exchange
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");
        System.out.println(" [Consumer1] Waiting for messages.");
        // 订阅消息
        channel.basicConsume(QUEUE_NAME, false, "ConsumerTag", new DefaultConsumer(
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
                                       AMQP.BasicProperties properties, byte[] body
                    throws IOException {
                //接收到的消息，进行业务逻辑处理。
                System.out.println("Received： " + new String(body, StandardCharsets
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
```

```
            });
        }
    }
```

参数	说明
QUEUE_NAME	Queue名称，在控制台 Queue 列表获取。
EXCHANGE_NAME	Exchange 名称，在控制台 Exchange 列表获取。
factory.setUri	集群接入地址，在**集群管理**页面操作列的**获取接入地址**获取。 
factory.setVirtualHost	Vhost 名称，在控制台 Vhost 页面复制，格式是“**集群 ID + | + vhost 名称**”。 
factory.setUsername	角色名称，在 **角色管理** 页面复制。
factory.setPassword	角色密钥，在 **角色管理** 页面复制**密钥**列复制。 

步骤4：查询消息

如果您想确认消息是否成功发送至 TDMQ RabbitMQ 版，可以在控制台 **集群管理** > **Queue** 页面查看接入的消费者情况。



说明：

上述是基于 RabbitMQ 的发布订阅模型的一个简单示例。其他示例可参见 Demo 或 RabbitMQ 官网 实例。

Go SDK

最近更新时间：2024-01-03 11:45:32

操作场景

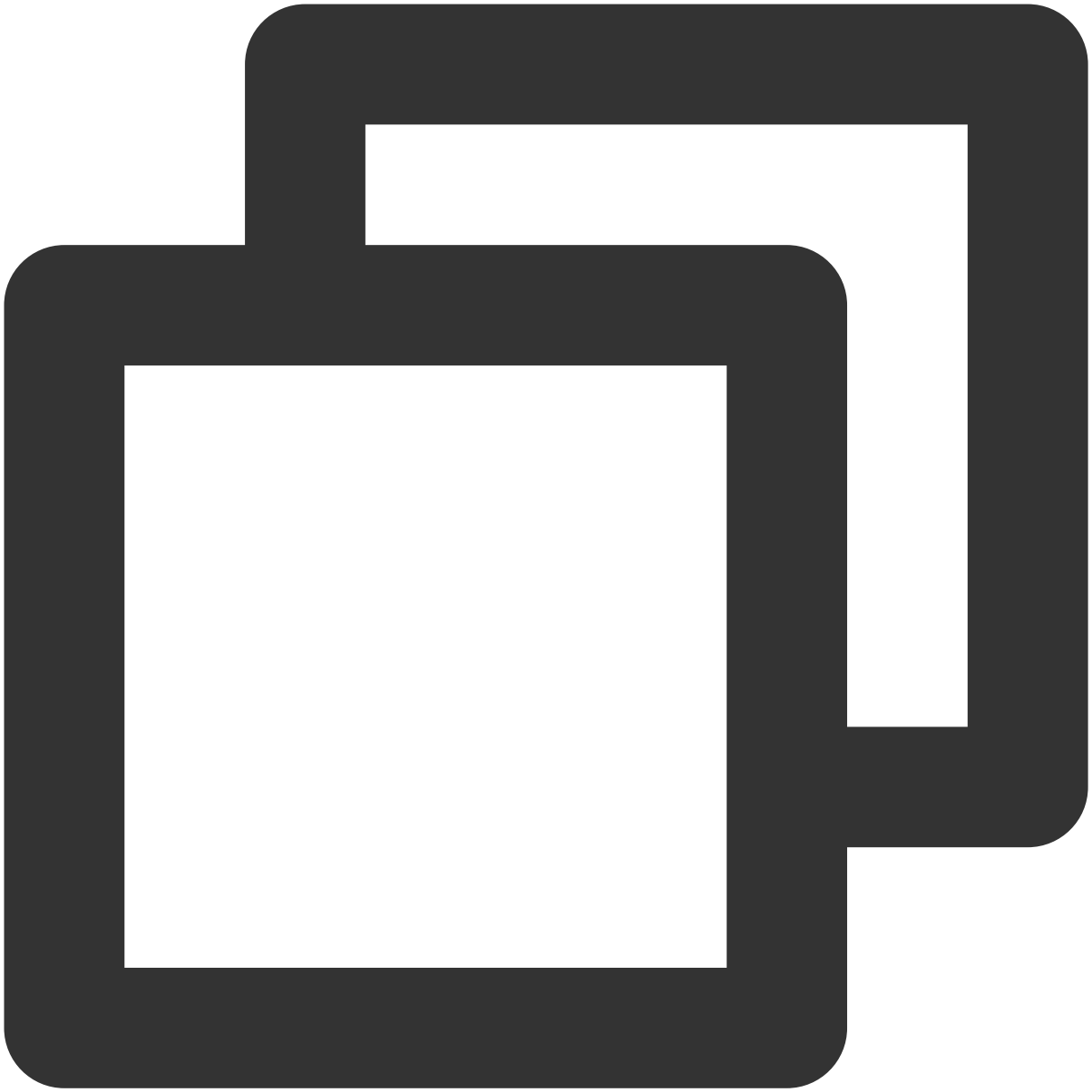
本文以调用 Go SDK 为例介绍通过开源 SDK 实现消息收发的操作过程，帮助您更好地理解消息收发的完整过程。

前提条件

完成资源创建与准备
安装 Go
下载 Demo

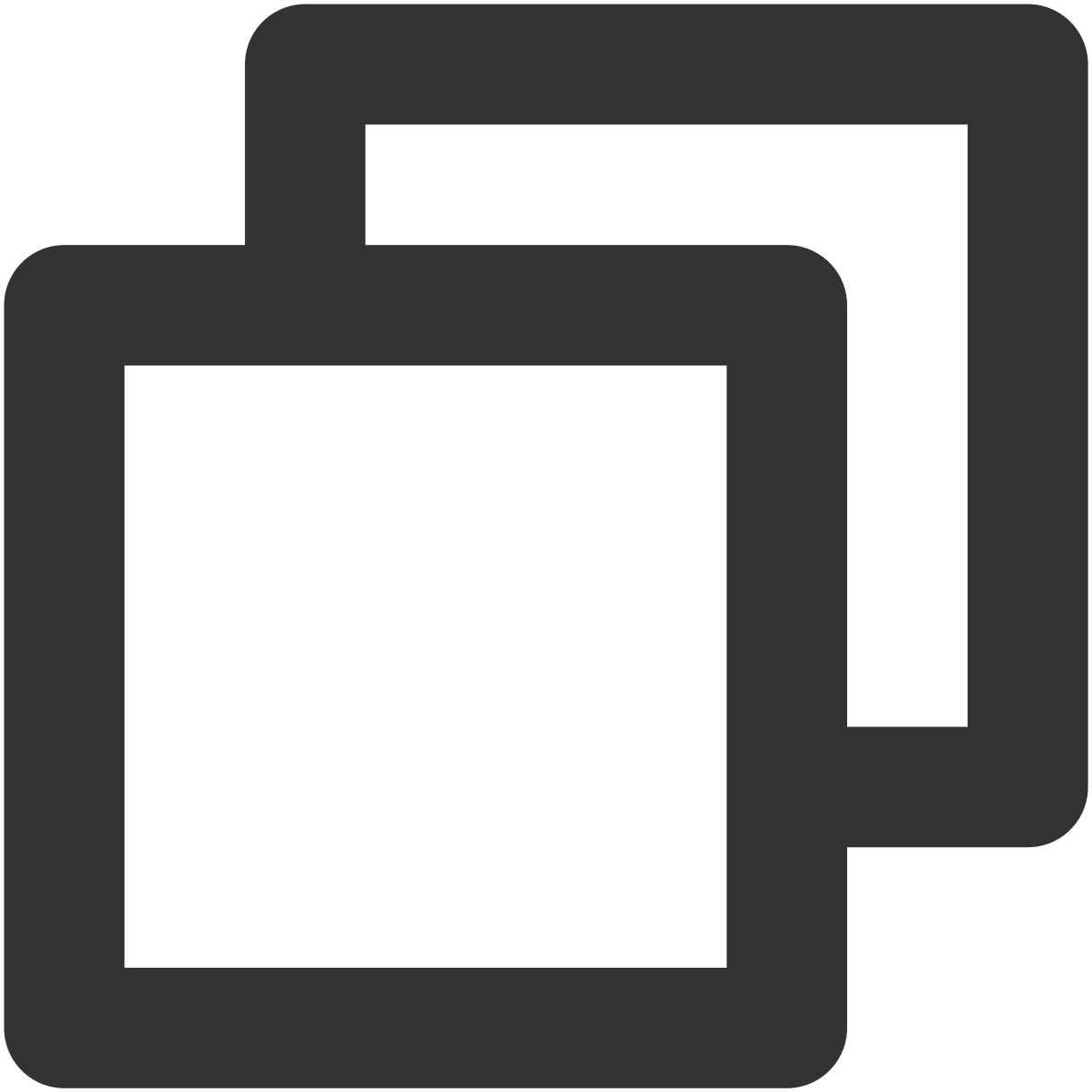
操作步骤

1. 执行如下命令在客户端环境安装所需包。



```
go get "github.com/rabbitmq/amqp091-go"
```

2. 安装完成后，即可引入到您的 GO 工程文件中。

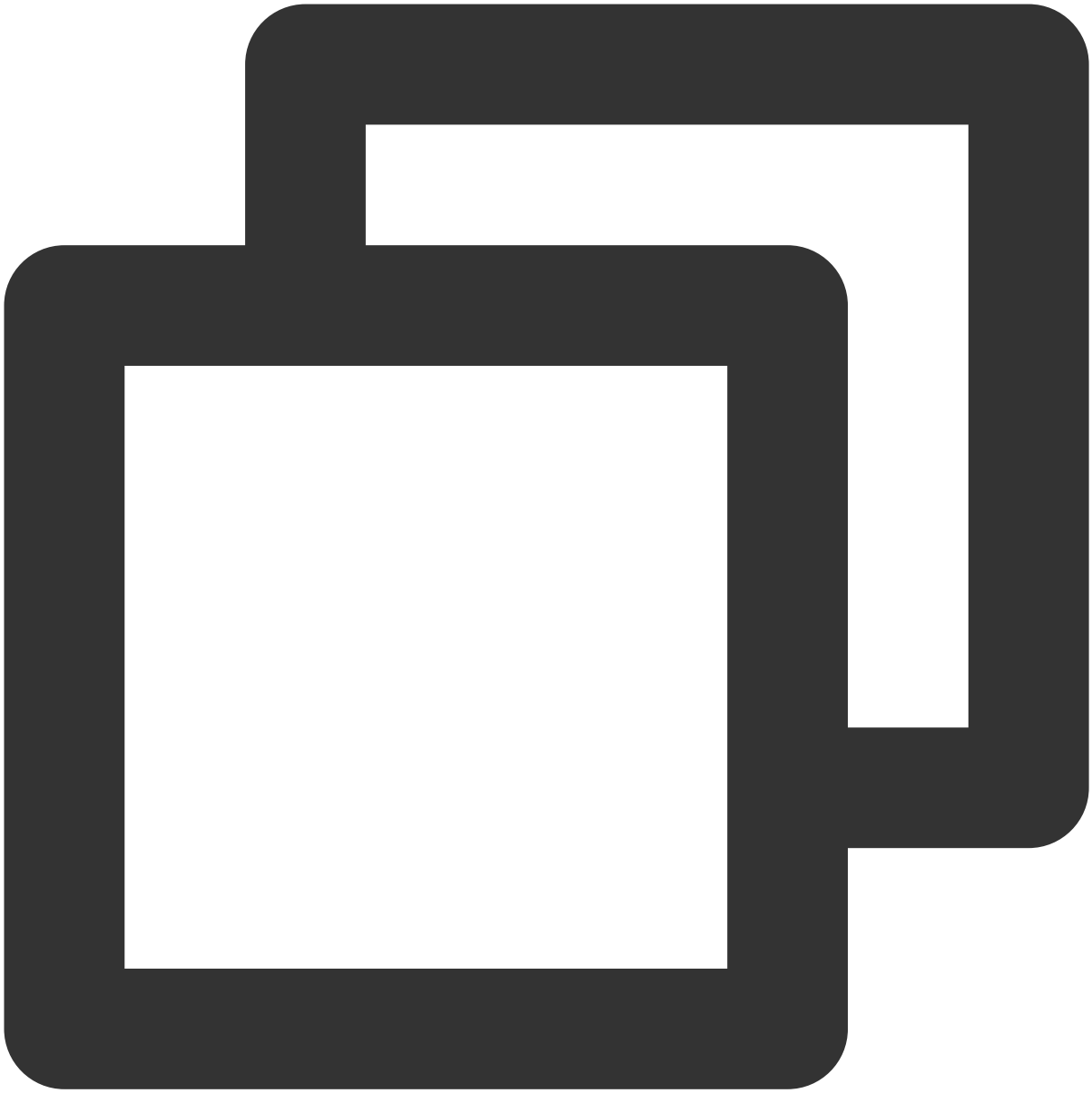


```
import (amqp "github.com/rabbitmq/amqp091-go")
```

引入之后即可在您的项目中使用客户端。

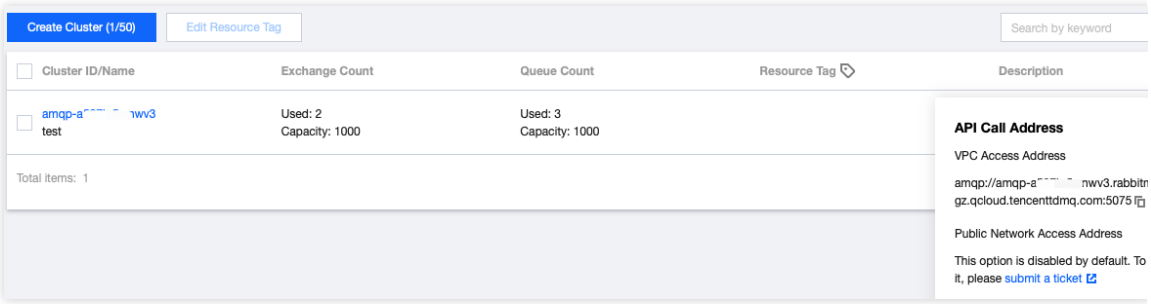
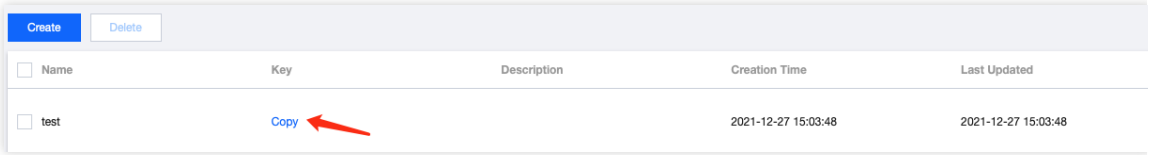
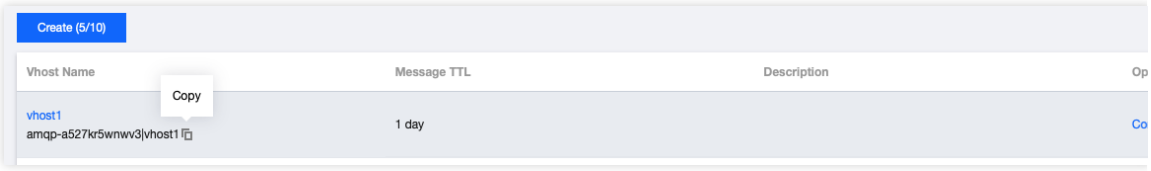
使用示例

1. 建立连接和通信信道。

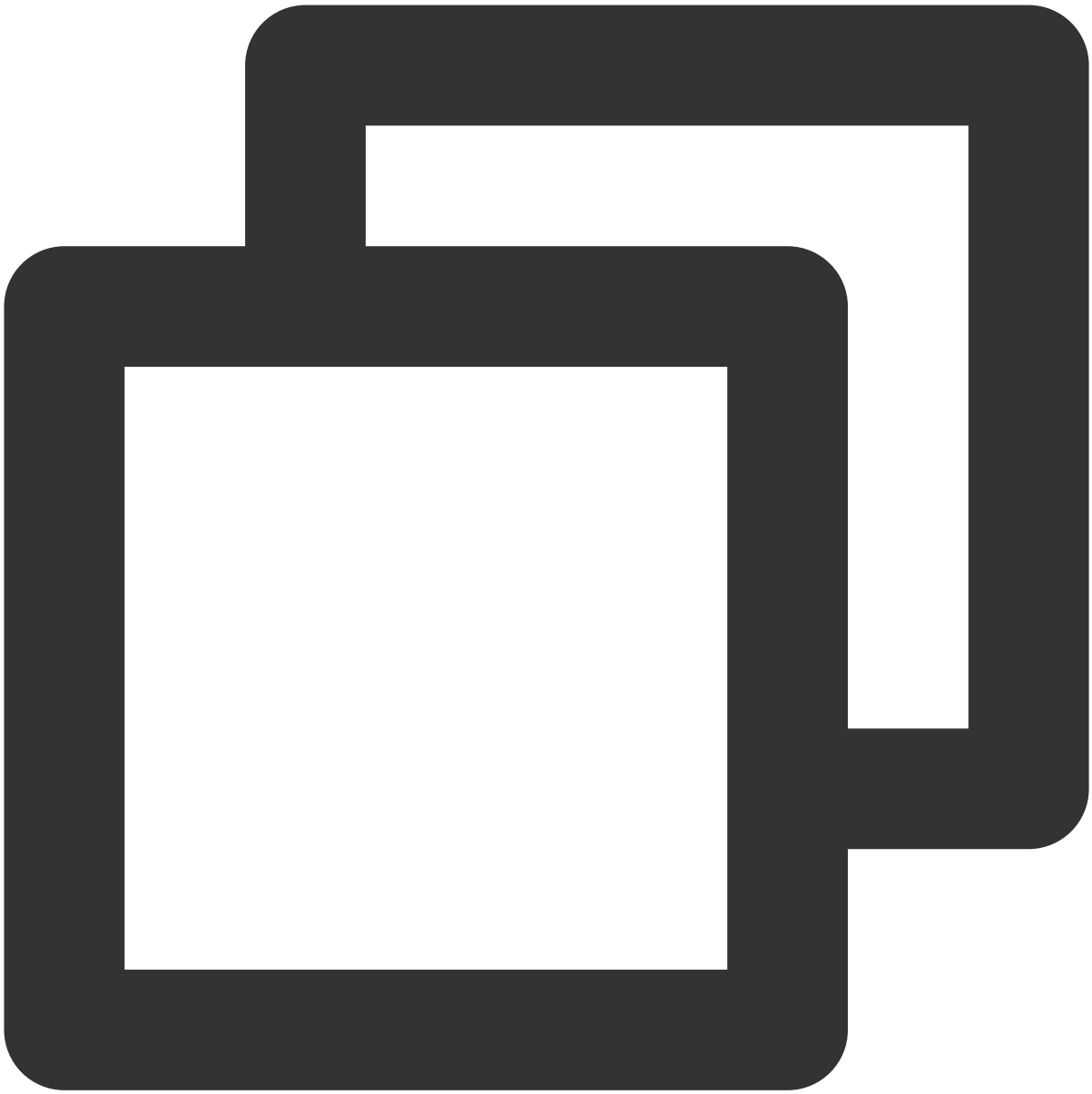


```
// 所需参数
const (
    host     = "amqp-xx.rabbitmq.x.tencenttdmq.com" // 服务接入地址
    username = "roleName" // 角色控制台对应的角色名称
    password = "eyJrZX..." // 角色对应的密钥
    vhost    = "amqp-xx|Vhost" // 要使用的Vhost全称
)
// 创建连接
conn, err := amqp.Dial("amqp://" + username + ":" + password + "@" + host + ":5672/
failOnError(err, "Failed to connect to RabbitMQ")
defer func(conn *amqp.Connection) {
```

```
    err := conn.Close()
    if err != nil {
    }
}(conn)

// 建立通道
ch, err := conn.Channel()
failOnError(err, "Failed to open a channel")
defer func(ch *amqp.Channel) {
    err := ch.Close()
    if err != nil {
    }
}(ch)
```

参数	说明
host	集群接入地址，在**集群管理**页面操作列的**获取接入地址**获取。 
username	角色名称，在 **角色管理** 页面复制。
password	角色密钥，在 **角色管理** 页面复制**密钥**列复制。 
vhost	Vhost 名称，在控制台 Vhost 页面复制，格式是**“集群 ID + | + vhost 名称”**。 

2. 声明交换机。



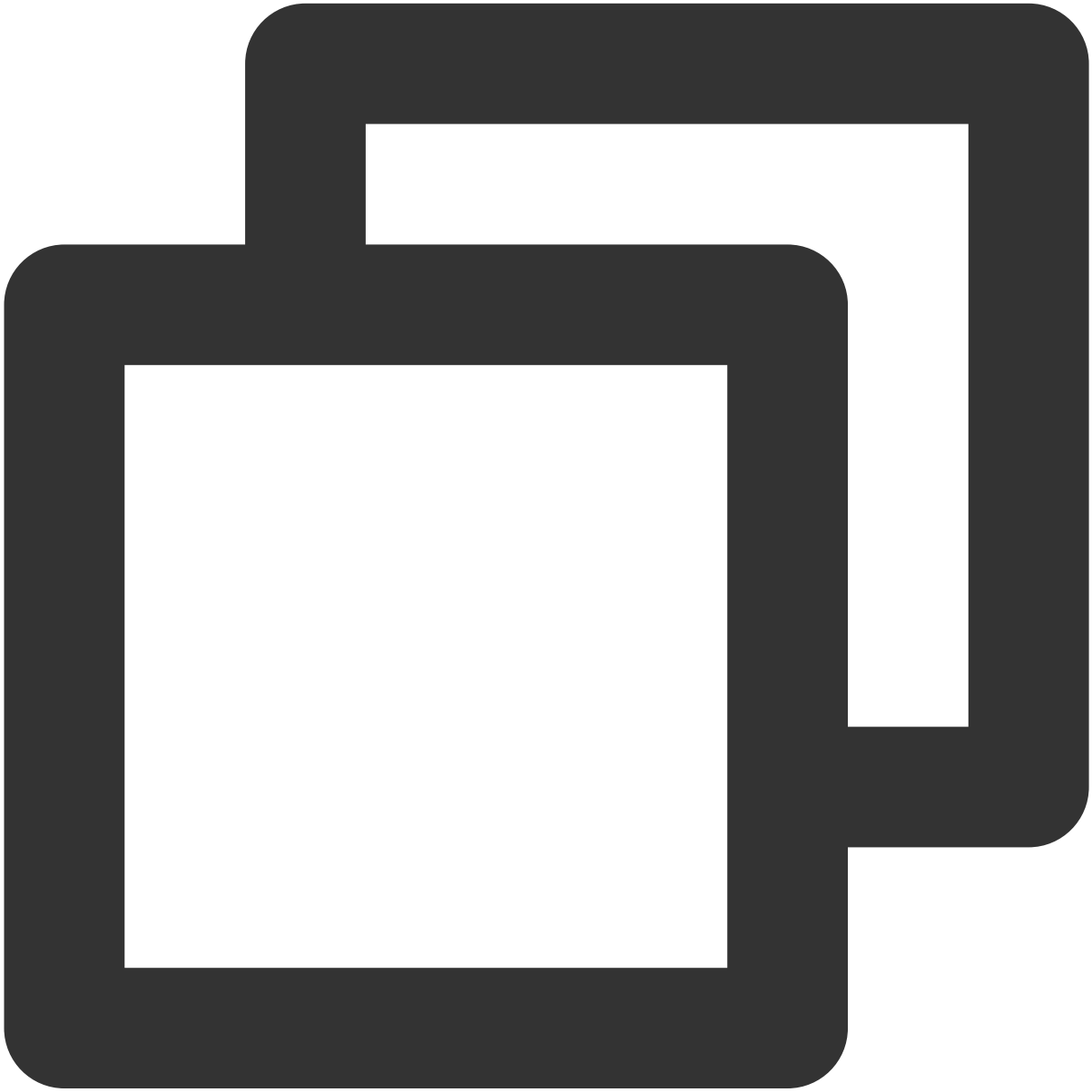
```
// 声明交换机 （名称和类型需要与存在的交换机保持一致）
    err = ch.ExchangeDeclare(
        "logs-exchange",   // 交换机名称
        "fanout", // 交换机类型
        true,     // durable
        false,    // auto-deleted
        false,    // internal
        false,    // no-wait
        nil,      // arguments
```

```
    )
    failOnError(err, "Failed to declare a exchange")
```

3. 发布消息。

消息可发给交换机，也可以直接发到指定队列 (hello world 和 work queues 消息模型)。

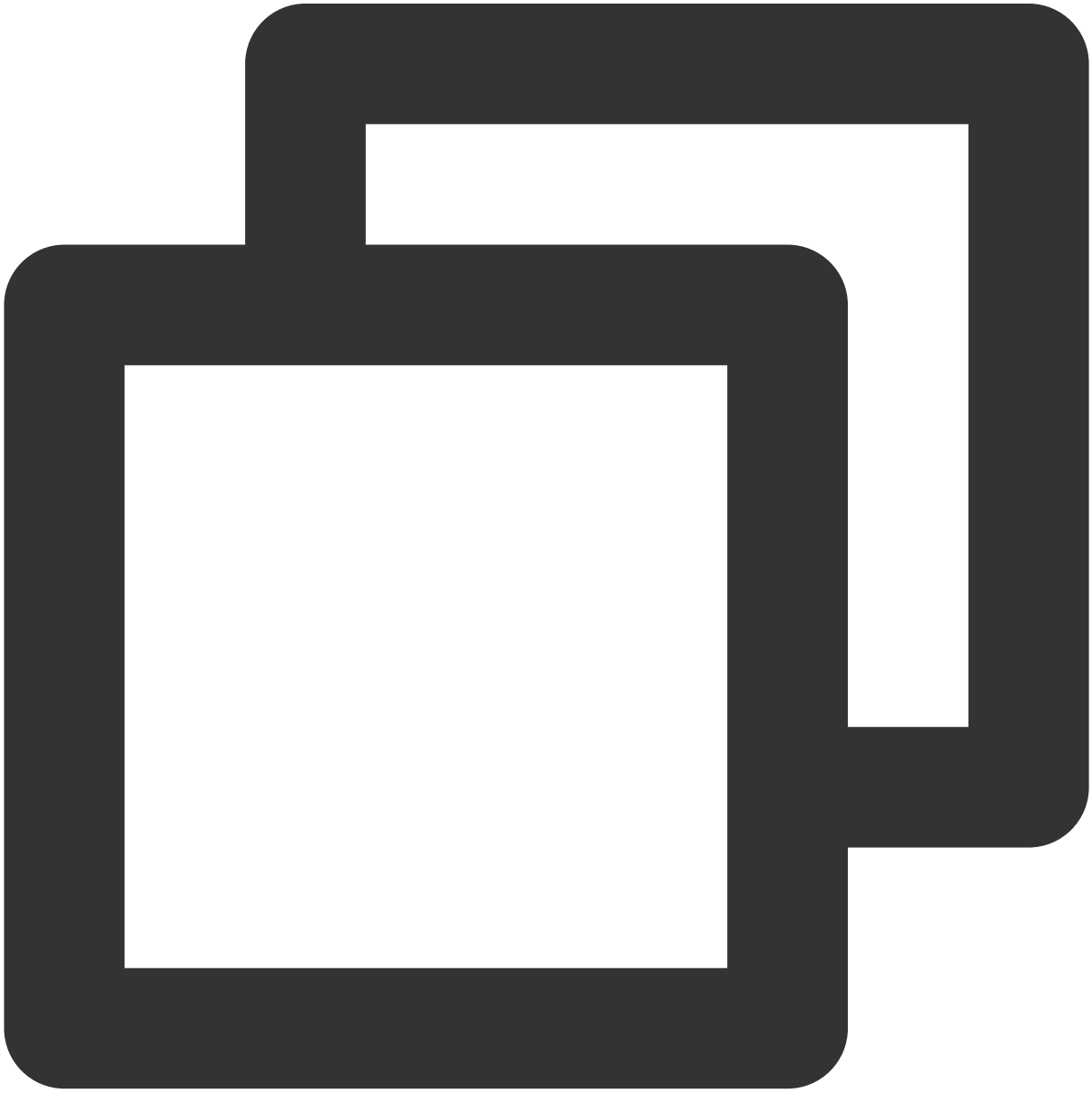
发布消息到交换机：



```
// 消息内容
body := "this is new message."
// 发布消息到交换机
err = ch.Publish(
```

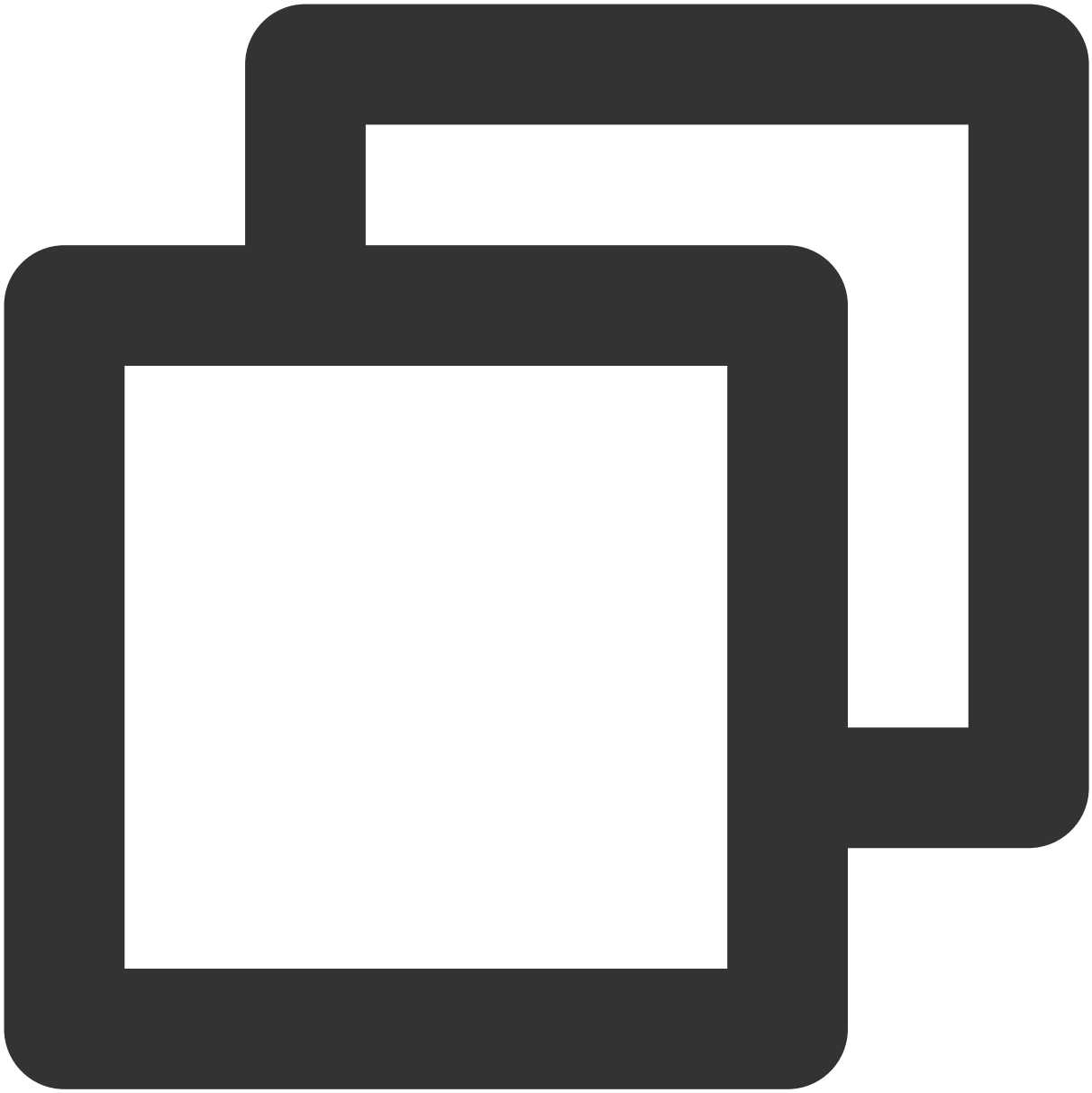
```
    "logs-exchange",         // exchange
    "", // routing key    (根据使用的交换机类型可选择的是否需要routing key)，如果不选择交换
    false,       // mandatory
    false,       // immediate
    amqp.Publishing{
        ContentType: "text/plain",
        Body:        []byte(body),
    })
failOnError(err, "Failed to publish a message")
```

发布消息到指定队列：



```
// 发布消息到指定的消息队列
err = ch.Publish(
    "",          // exchange
    queue.Name, // routing key
    false,       // mandatory
    false,       // immediate
    amqp.Publishing{
        ContentType: "text/plain",
        Body:        []byte(body),
    })
failOnError(err, "Failed to publish a message")
```

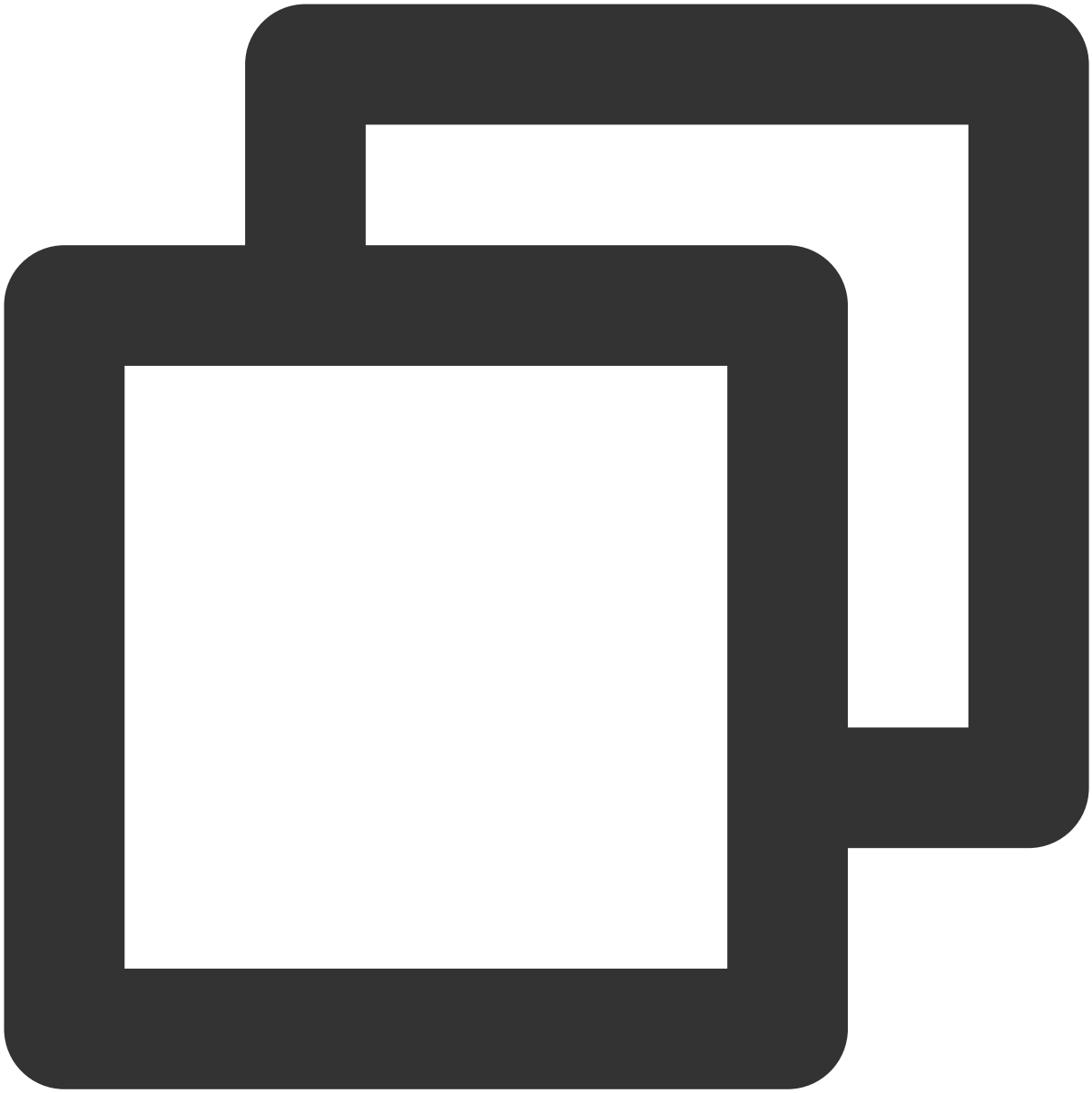
4. 订阅消息。



```
// 创建消费者并消费指定消息队列中的消息
msgs, err := ch.Consume(
    "message-queue", // message-queue
    "",              // consumer
    false,           // 设置为非自动确认(可根据需求自己选择)
    false,           // exclusive
    false,           // no-local
    false,           // no-wait
    nil,             // args
)
failOnError(err, "Failed to register a consumer")
```

```
// 获取消息队列中的消息
forever := make(chan bool)
go func() {
    for d := range msgs {
        log.Printf("Received a message: %s", d.Body)
        t := time.Duration(1)
        time.Sleep(t * time.Second)
        // 手动回复ack
        d.Ack(false)
    }
}()
log.Printf(" [Consumer] Waiting for messages.")
<-forever
```

5. 消费者使用 routing key。



```
// 需要在消息队列中指定 交换机 和 routing key
err = ch.QueueBind(
    q.Name, // queue name
    "routing_key",     // routing key
    "topic_demo", // exchange
    false,
    nil,
)
failOnError(err, "Failed to bind a queue")
```

说明：

详细使用示例可参见 Demo 或 RabbitMQ 官网。

Python SDK

最近更新时间：2024-01-03 11:45:32

操作场景

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

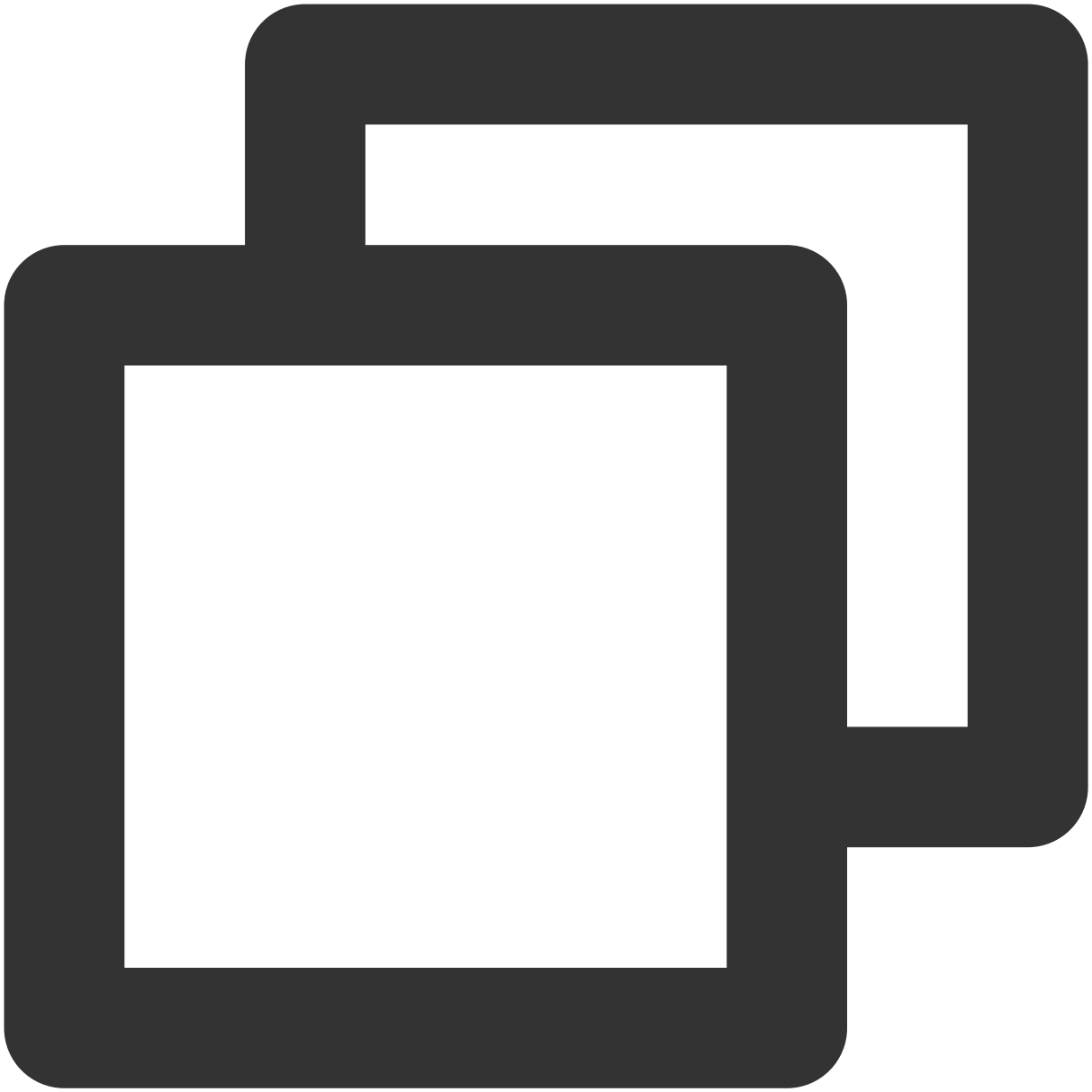
前提条件

完成资源创建与准备
安装 Python
安装 pip
下载 Demo

操作步骤

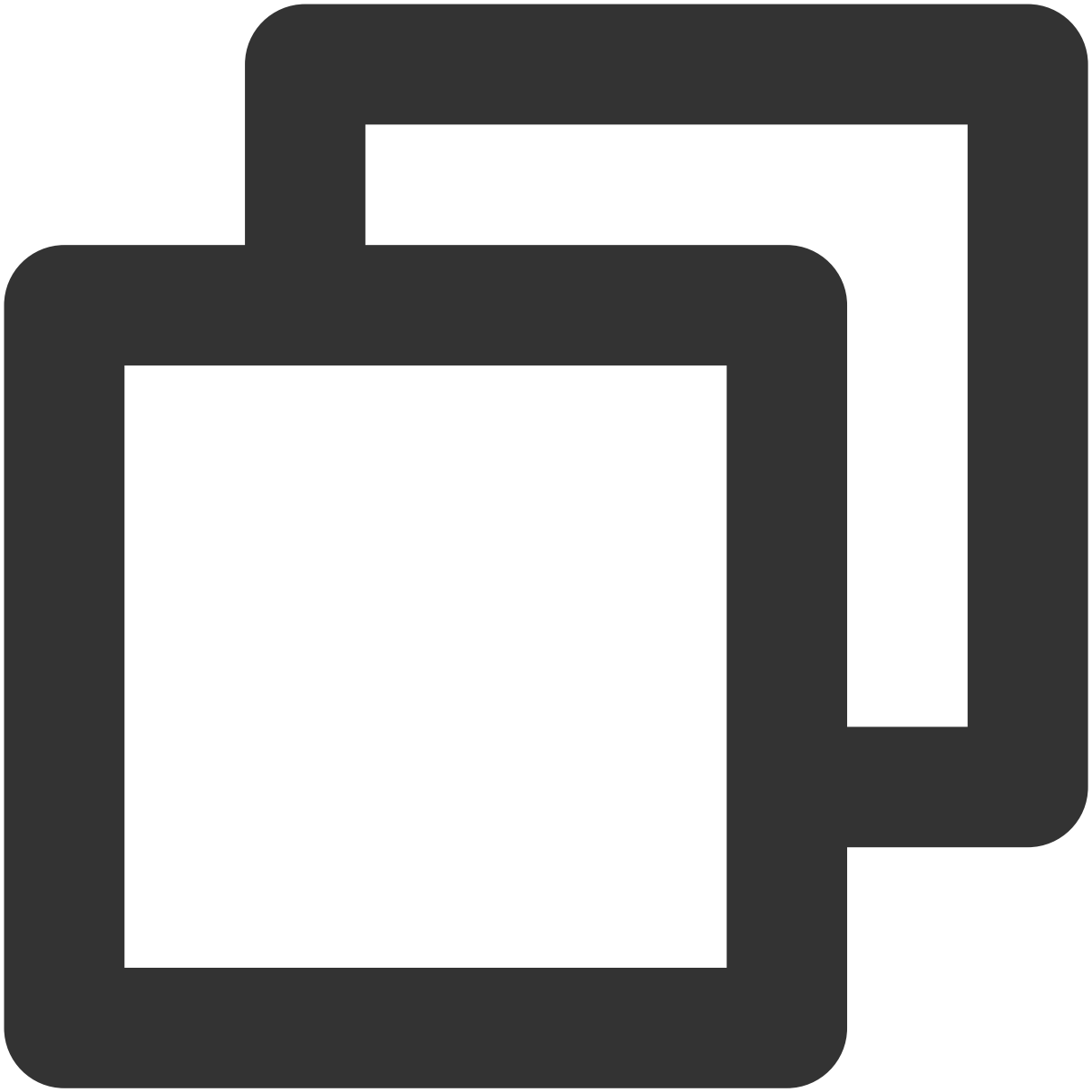
步骤1：添加依赖

1. 根据 RabbitMQ 官网推荐使用 pika，首先要在客户端使用环境中安装 pika。



```
python -m pip install pika --upgrade
```

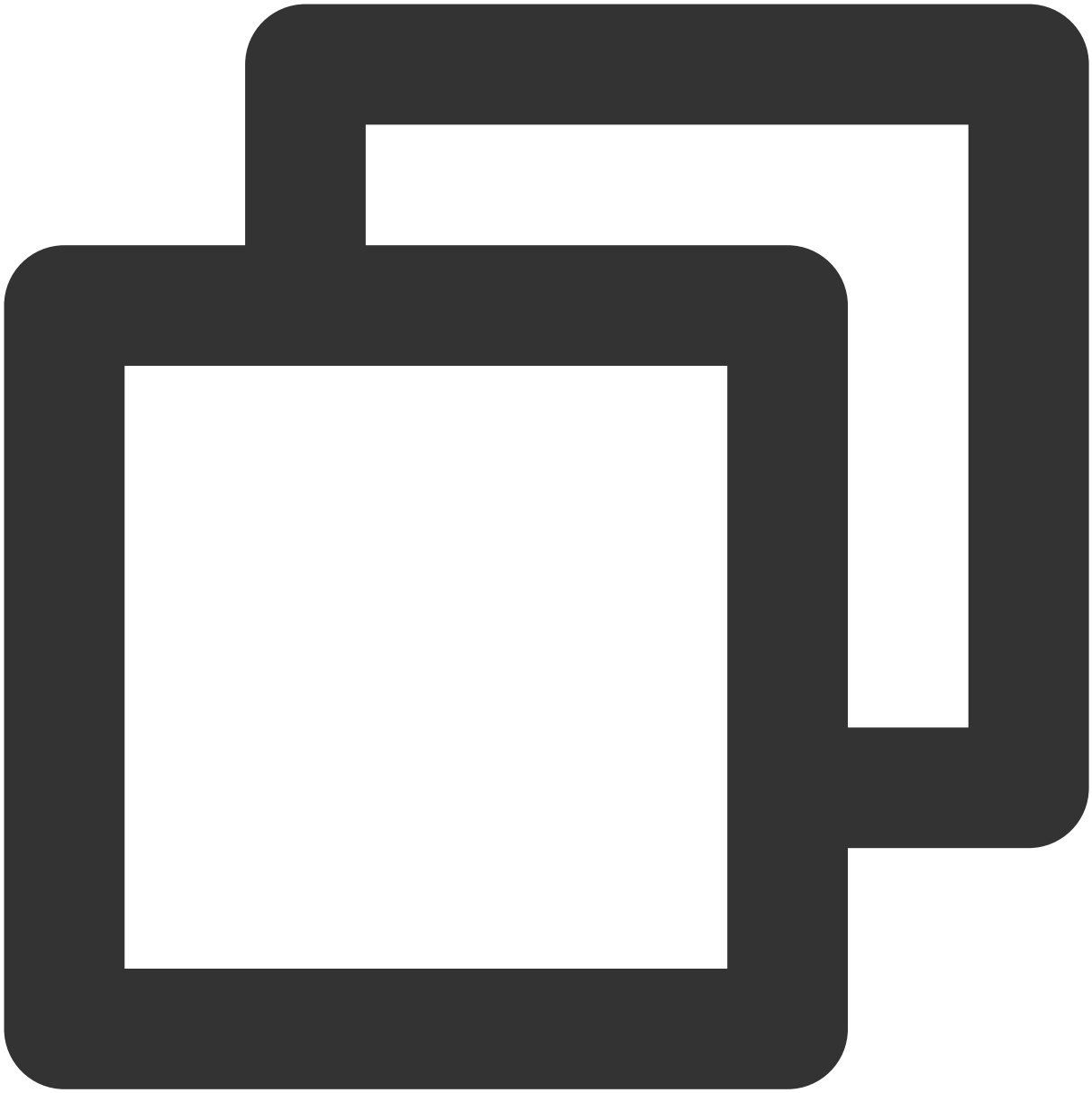
2. 在创建客户端时导入 pika。



```
import pika
```

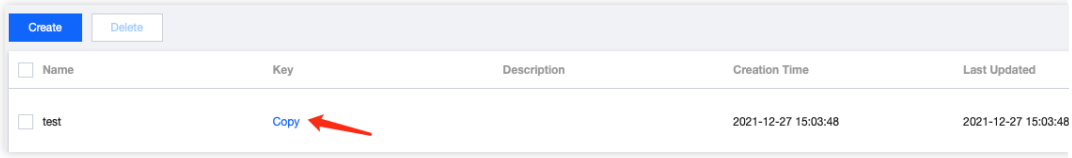
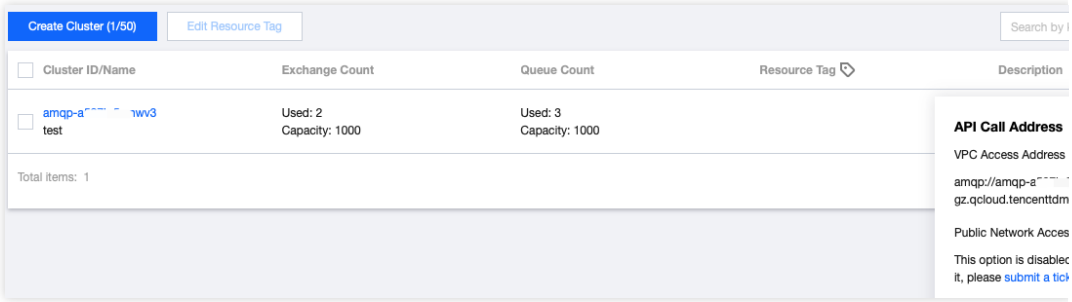
步骤2：生产消息

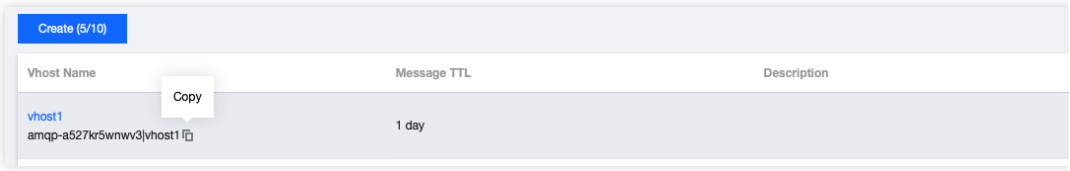
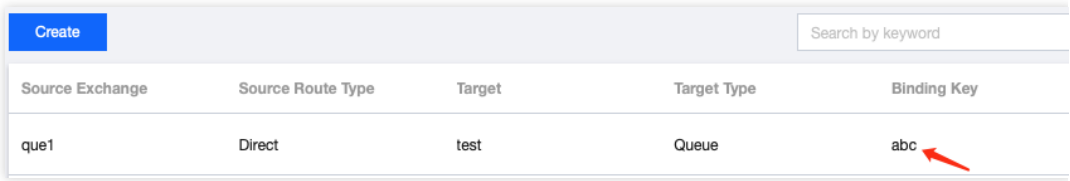
创建并编译运行生产消息程序 messageProducer.py。



```
import pika

# 使用用户名和密码创建登录凭证对象
credentials = pika.PlainCredentials('rolename', 'eyJr***')
# 创建连接
connection = pika.BlockingConnection(pika.ConnectionParameters(
    host='amqp-xx.rabbitmq.x.com', port=5672, virtual_host='amqp-xxx|Vhostname', cr
# 建立信道
channel = connection.channel()
# 声明交换机
channel.exchange_declare(exchange='direct_exchange', exchange_type="direct")
```

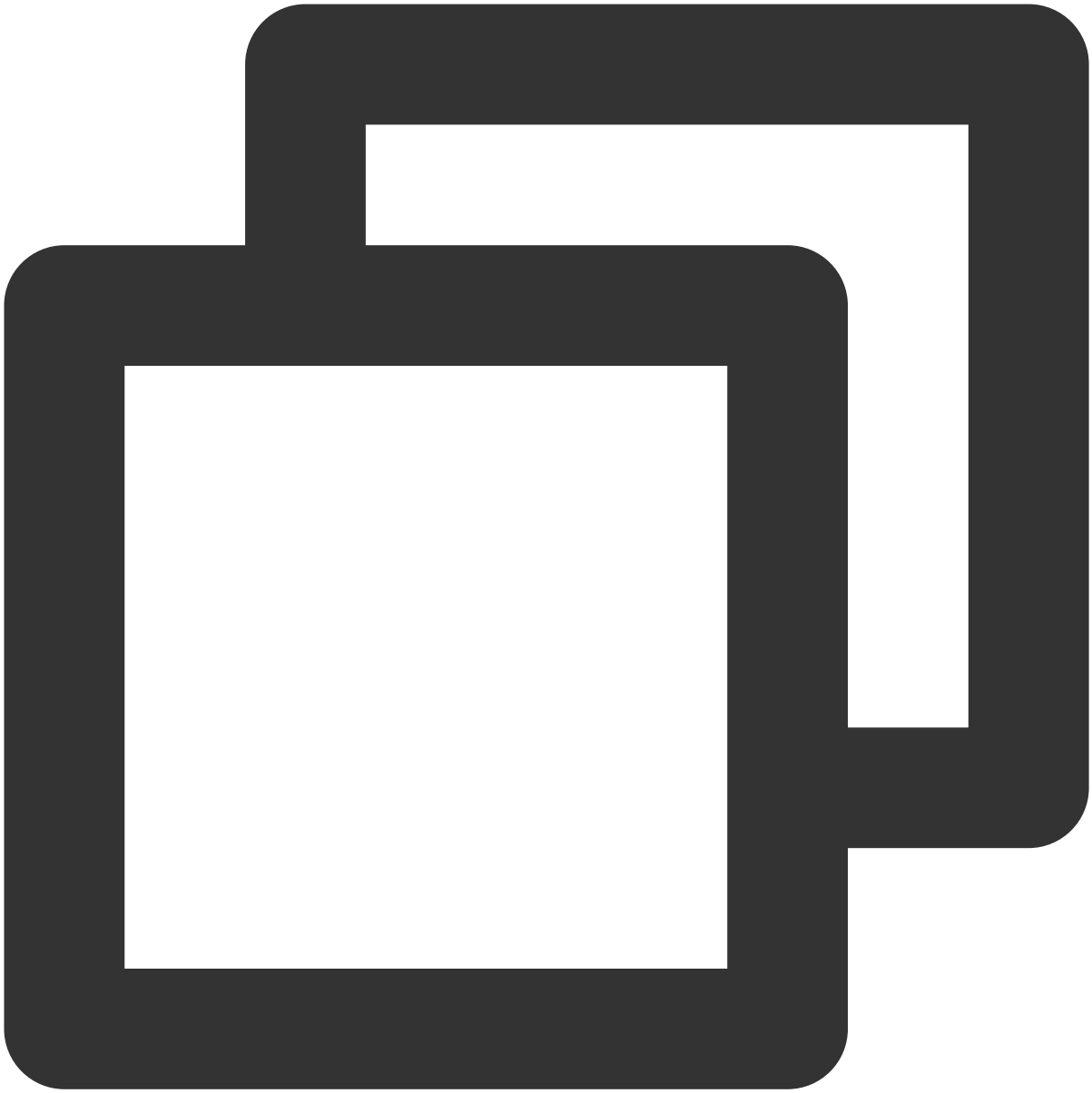
```
routingKeys = ['aaa.bbb.ccc', 'aaa.bbb.ddd', 'aaa.ccc.zzz', "xxx.yyy.zzz"]

for routingKey in routingKeys:
    # 发送消息到指定的交换机
    # 不指定交换机的情况下发送消息，需要指定消息队列，参数routing_key在使用指定交换机时，表示rou
    channel.basic_publish(exchange='direct_exchange',
                          routing_key=routingKey,
                          body=(routingKey + 'This is a new direct message.').encod
                          properties=pika.BasicProperties(
                              delivery_mode=2,  # 设置消息持久化
                          ))
    print('send success msg to rabbitmq')
connection.close()
```

参数	说明
rolename	角色名称，在 **角色管理** 页面复制。
eyJr***	角色密钥，在 **角色管理** 页面复制**密钥**列复制。 
host	集群接入地址，在**集群管理**页面操作列的**获取接入地址**获取。 
port	集群接入地址端口，在**集群管理**页面操作列的**获取接入地址**获取。
virtual_host	Vhost 名称，在控制台 Vhost 页面复制，格式是“**集群 ID + | + vhost 名称**”。

	
direct_exchange	Exchange 名称，在控制台 Exchange 列表获取。
routingKeys	消息的路由规则，在控制台 绑定关系列表的**绑定 Key**列获取。 

步骤3：消费消息

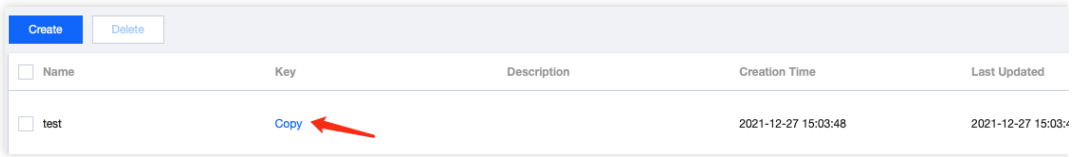
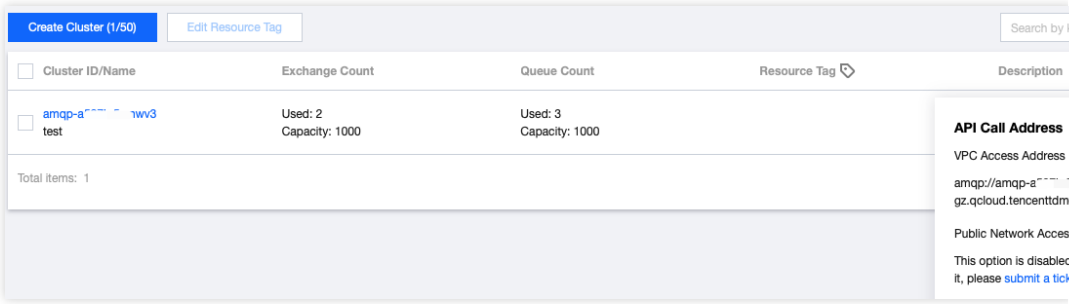
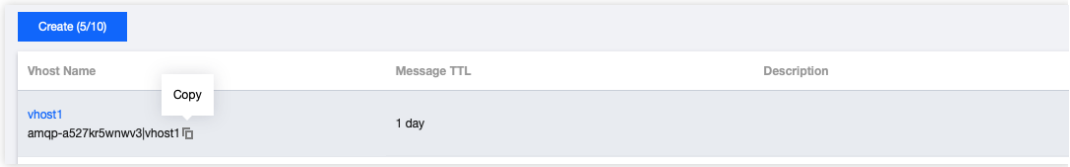
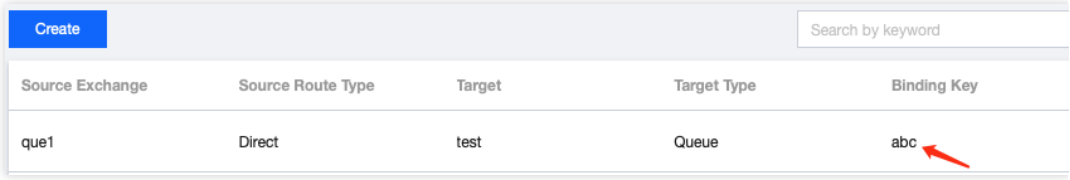
创建并编译运行消费消息程序 messageConsumer.py。



```
import os
import pika
import sys


def main():
    # 使用用户名和密码创建登录凭证对象
    credentials = pika.PlainCredentials('rolename', 'eyJr***')
    # 创建连接
    connection = pika.BlockingConnection(pika.ConnectionParameters(
        host='amqp-xx.rabbitmq.x.com', port=5672, virtual_host='amqp-xxx|Vhostname'
```

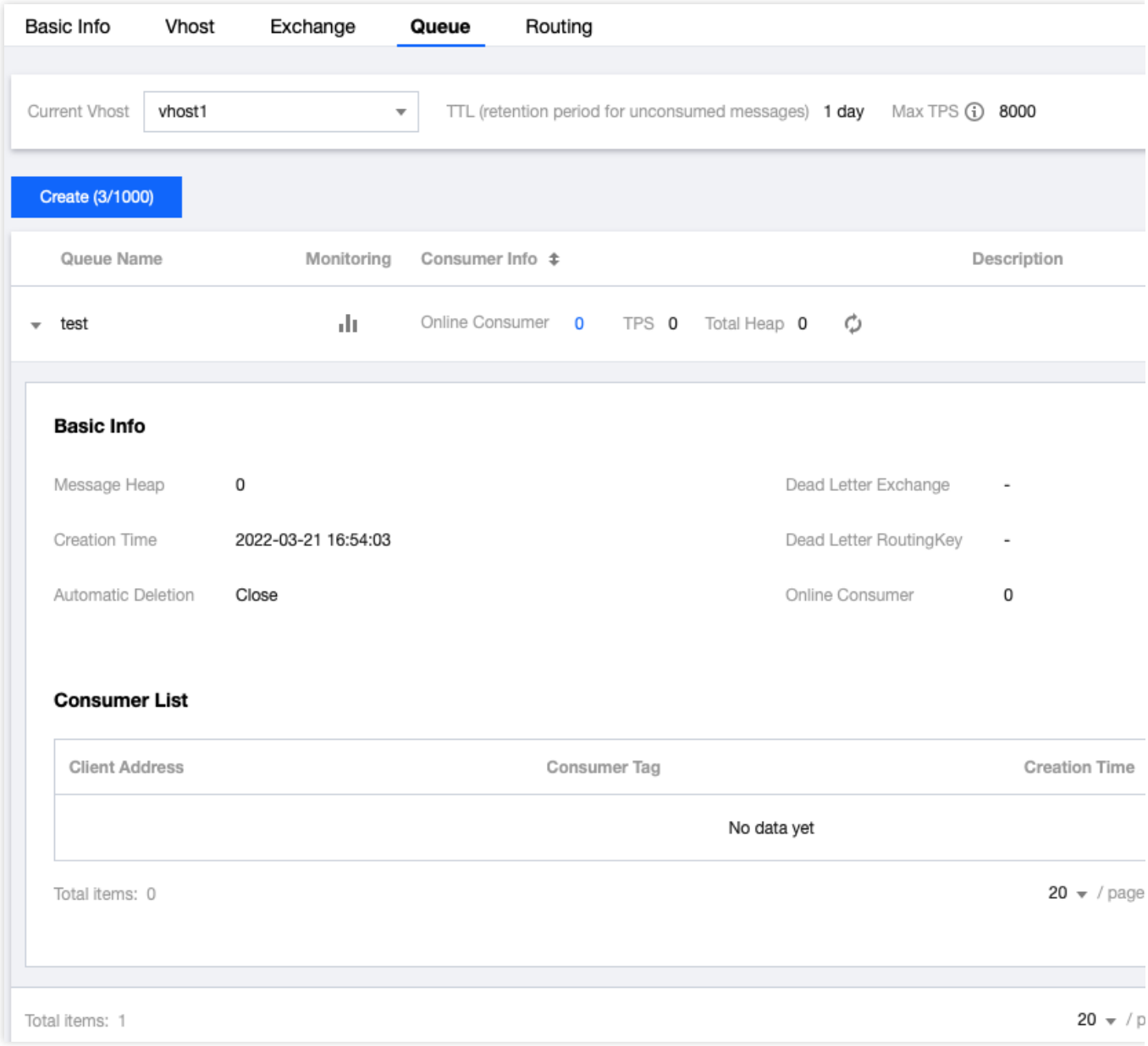
```
    # 建立信道
    channel = connection.channel()
    # 声明消息队列
    channel.queue_declare(queue='route_queue1', exclusive=True, durable=True)
    # 绑定消息队列到交换机，并指定 routing key
    routing_keys = ['aaa.bbb.ccc', 'aaa.bbb.ddd']
    for routingKey in routing_keys:
        channel.queue_bind(exchange='direct_exchange', queue="route_queue1", routin
    # 设置只接受一个未确认消息
    channel.basic_qos(prefetch_count=1)

    # 消息消费逻辑
    def callback(ch, method, properties, body):
        print(" [Consumer1(Direct 'aaa.bbb.ccc'/'aaa.bbb.ddd')] Received (%r)" % bo
        # 手动回复ACK
        ch.basic_ack(delivery_tag=method.delivery_tag)

    # 创建消费者，消费消息队列中的消息
    channel.basic_consume(queue='route_queue1',
                          on_message_callback=callback,
                          auto_ack=False)  # 设置为非自动确认

    print(" [Consumer1(Direct 'aaa.bbb.ccc'/'aaa.bbb.ddd')] Waiting for messages. T
    channel.start_consuming()


if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

参数	说明
rolename	角色名称，在 **角色管理** 页面复制。
eyJr***	角色密钥，在 **角色管理** 页面复制**密钥**列复制。

	
host	集群接入地址，在**集群管理**页面操作列的**获取接入地址**获取。 
port	集群接入地址端口，在**集群管理**页面操作列的**获取接入地址**获取。
virtual_host	Vhost 名称，在控制台 Vhost 页面复制，格式是“**集群 ID + | + vhost 名称**”。 
direct_exchange	Exchange 名称，在控制台 Exchange 列表获取。
route_queue1	Queue名称，在控制台 Queue 列表获取。
routingKey	消息的路由规则，在控制台 绑定关系列表的**绑定 Key**列获取。 

步骤4：查看消息

如果您想确认消息是否成功发送至 TDMQ RabbitMQ 版，可以在控制台 **集群管理** > **Queue** 页面查看接入的消费者情况。



说明：

完整示例或其他使用可参见 Demo 或者 RabbitMQ 官方使用文档。

PHP SDK

最近更新时间：2024-01-03 11:45:32

操作场景

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

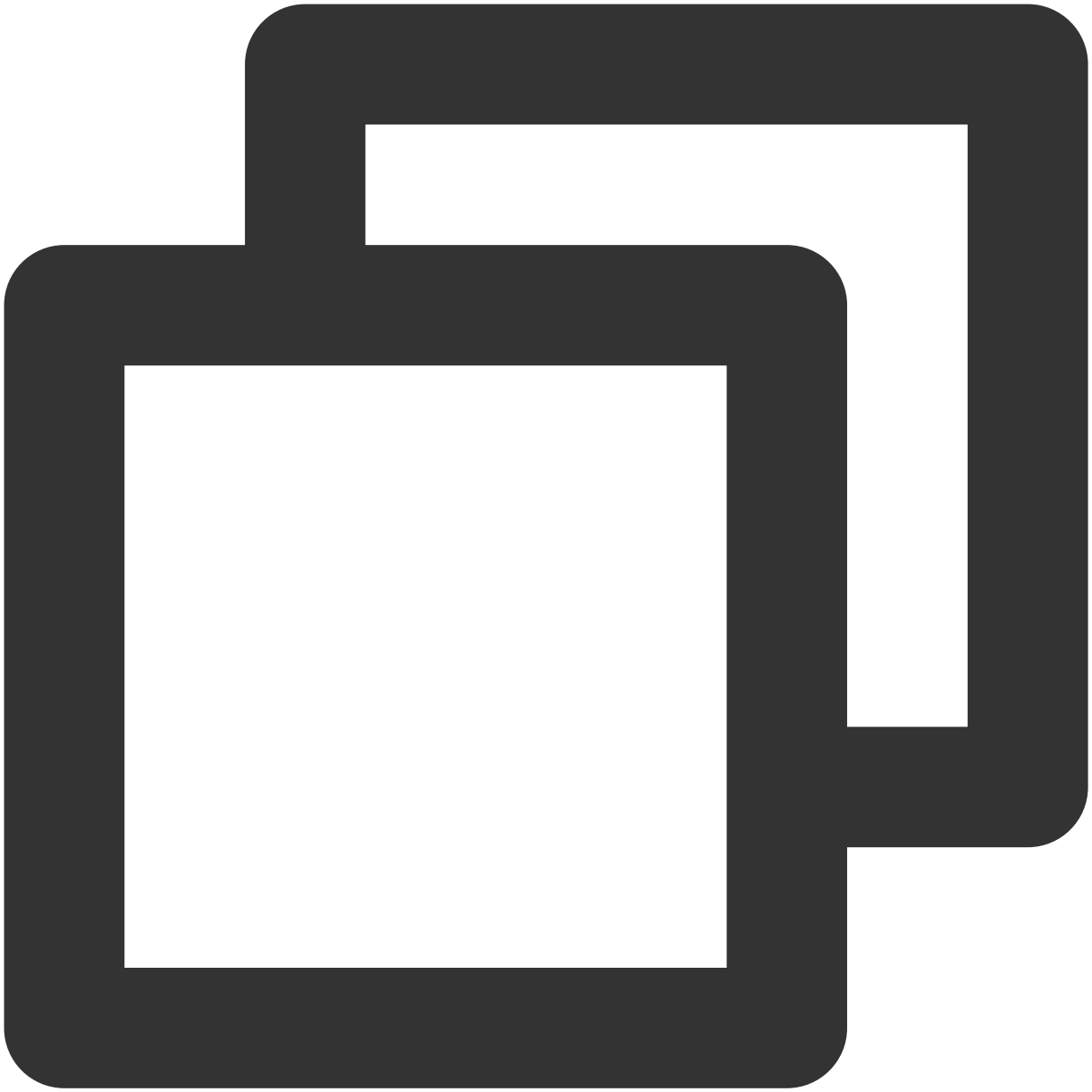
前提条件

安装 PHP 5.6 或以上版本
安装 PEAR
下载 Demo

操作步骤

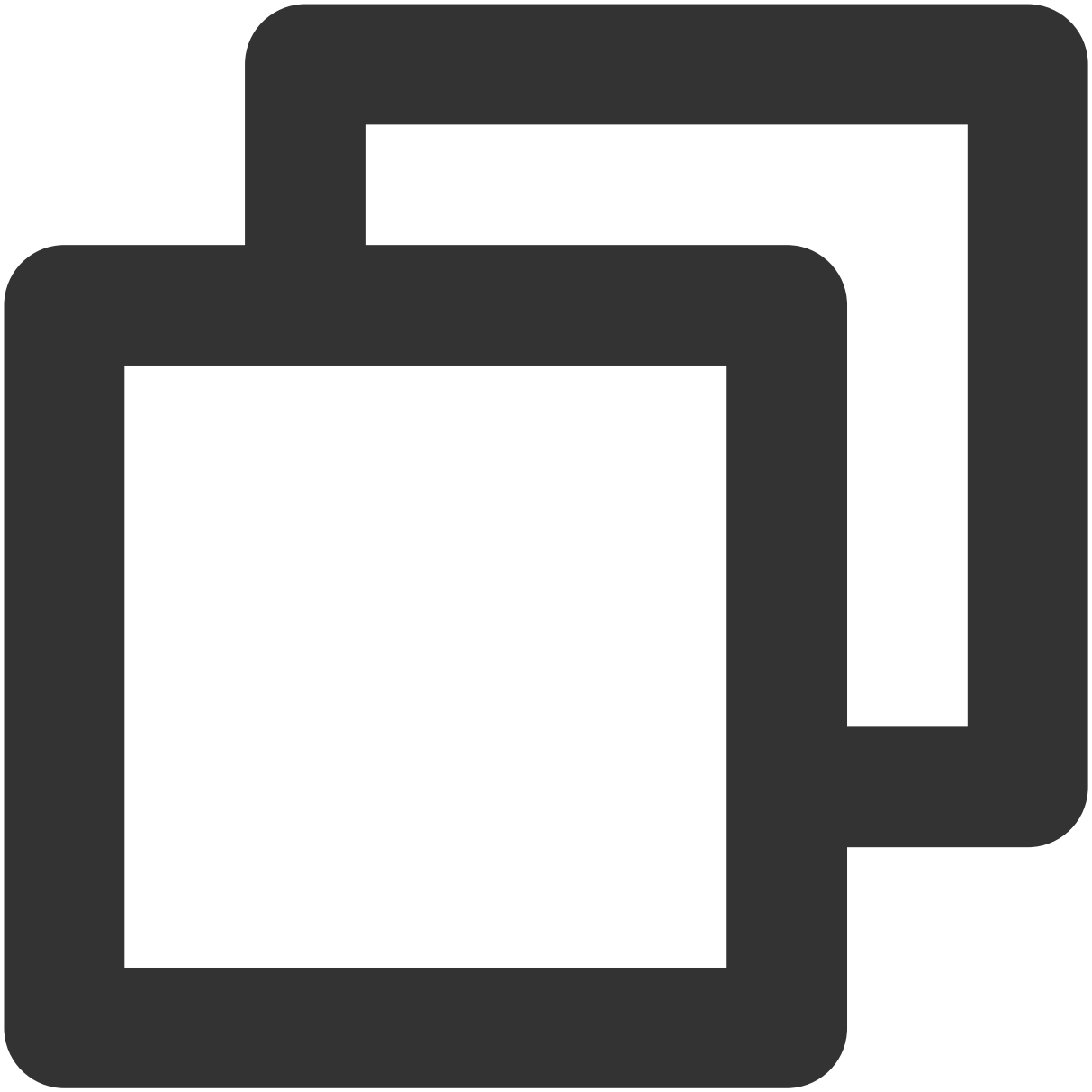
步骤1：安装 php-amqplib 库

根据 RabbitMQ 官网推荐使用 php-amqplib Client，首先需要在项目中引入 php-amqplib 库。
1. 在项目中添加 `composer.json` 文件。



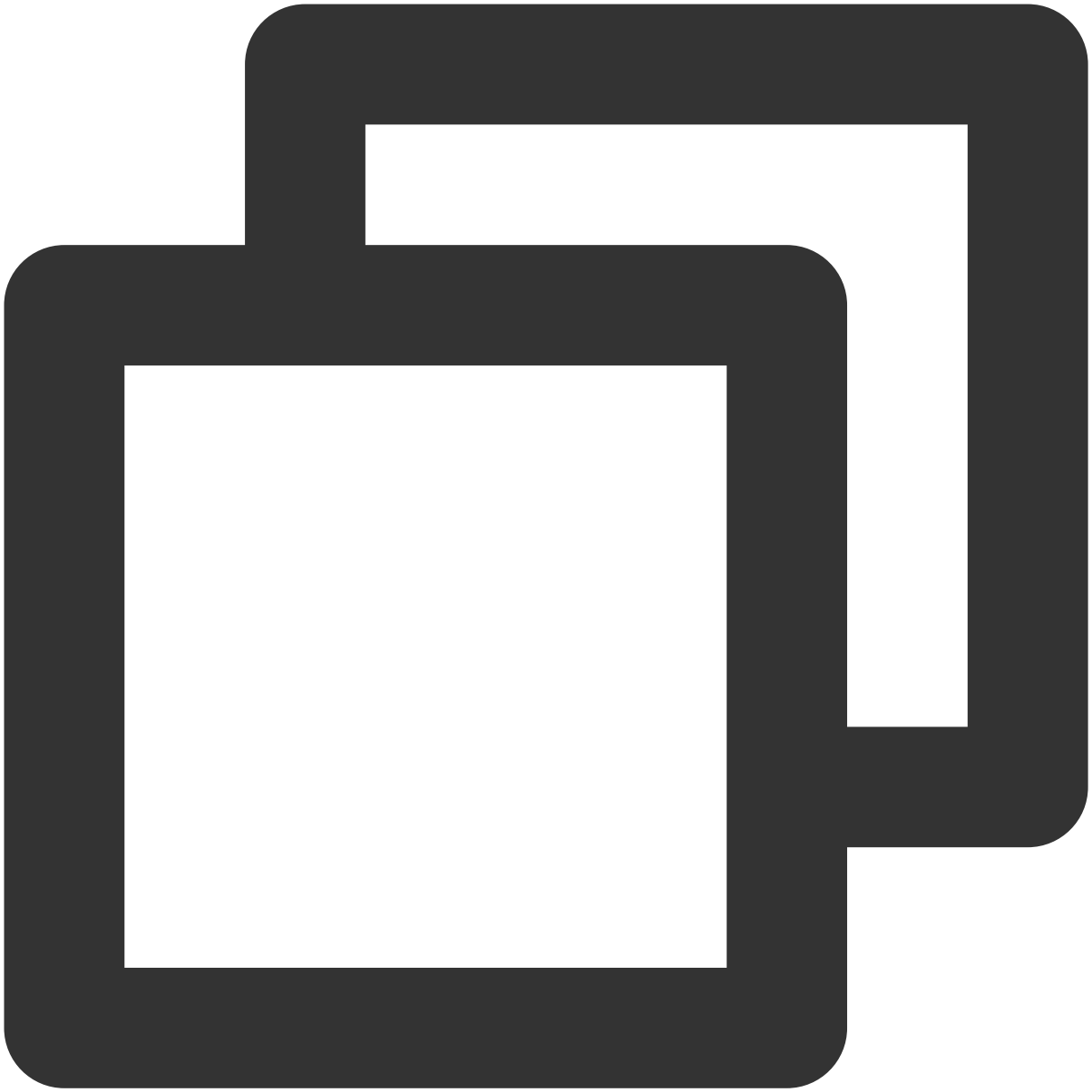
```
{
    "require": {
        "php-amqplib/php-amqplib": ">=3.0"
    }
}
```

2. 使用 Composer 进行安装。



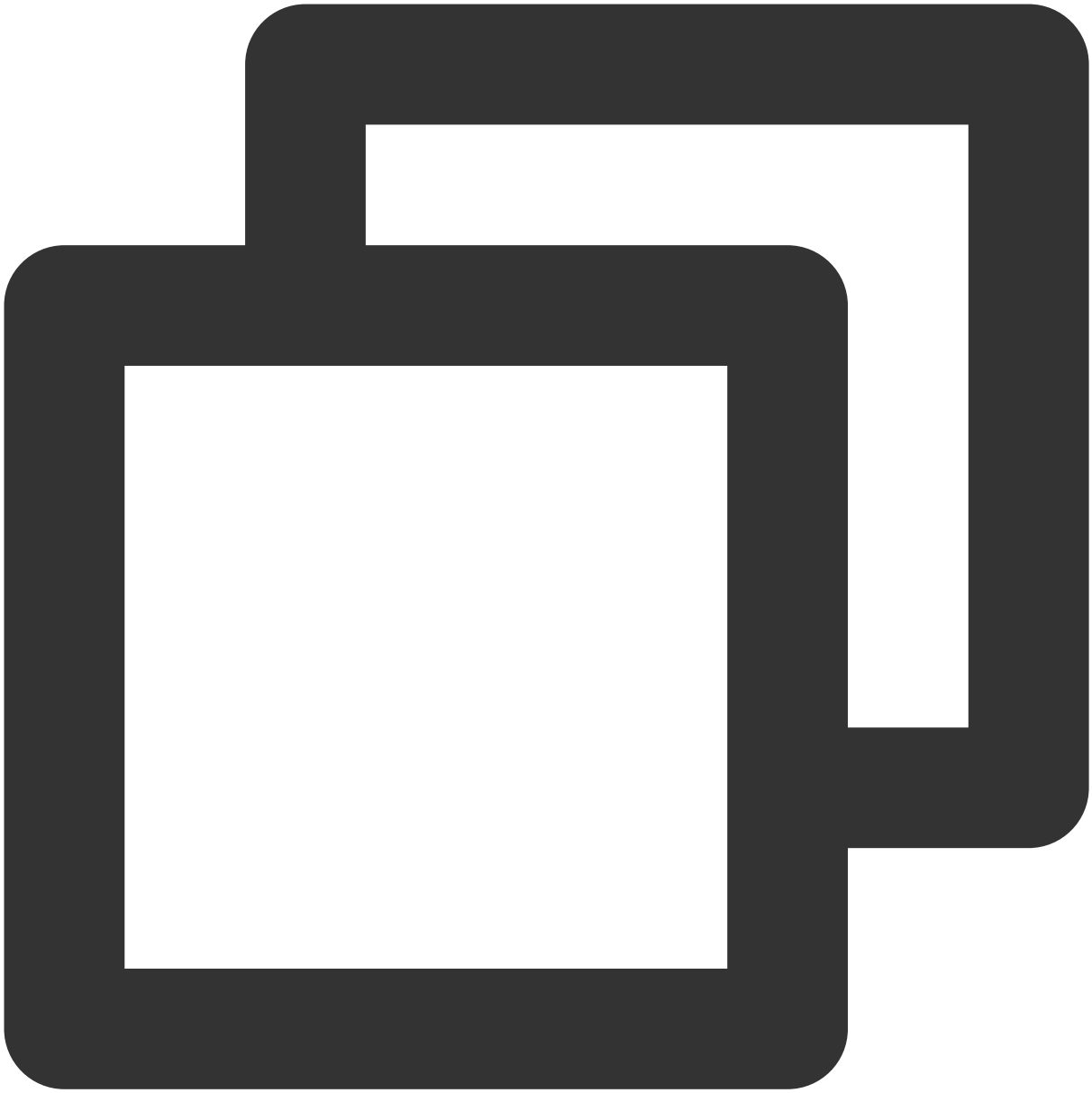
```
composer.phar install
```

或者使用下述命令：



```
composer install
```

3. 创建客户端需要引入库文件，在客户端文件中引入库文件。

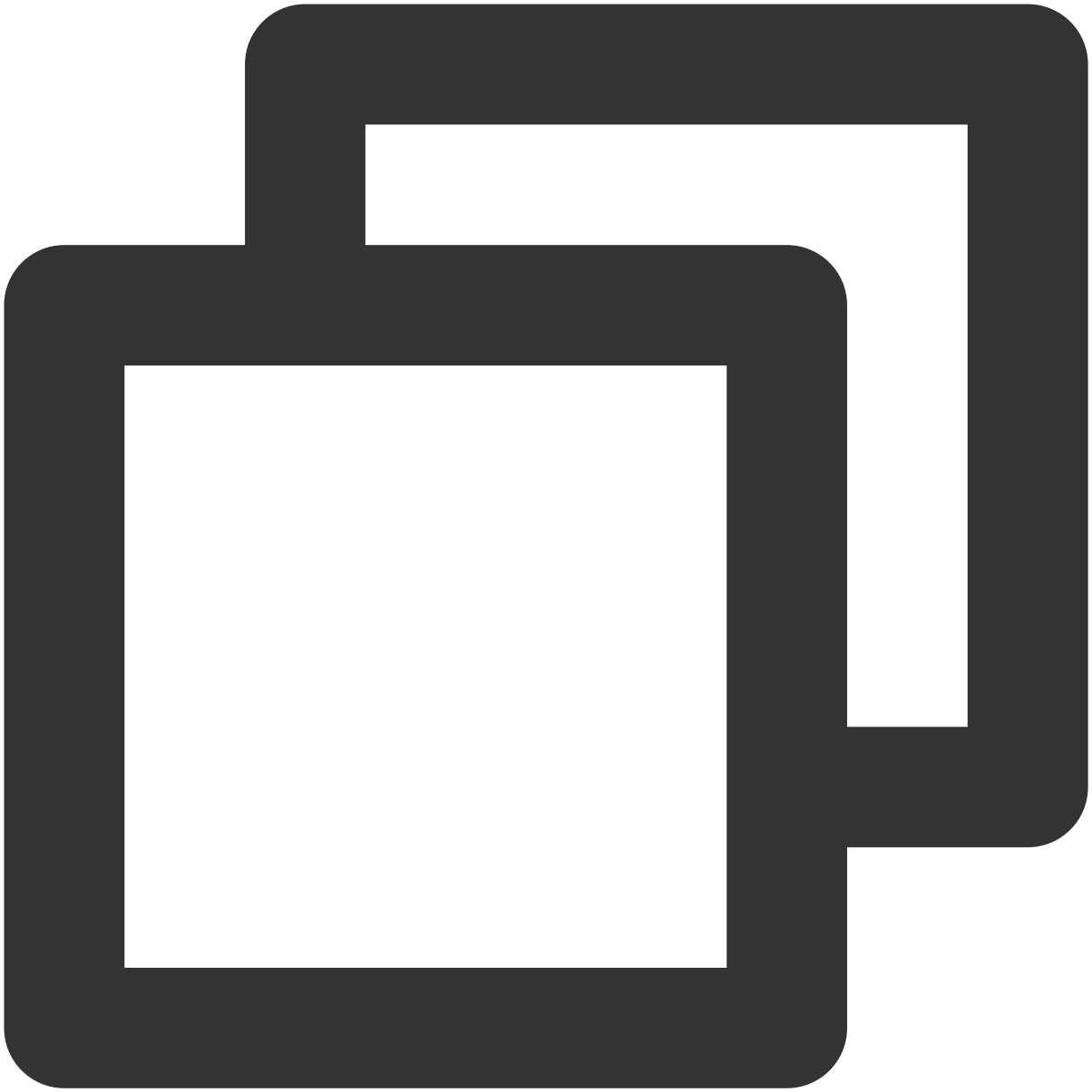


```
require_once('../vendor/autoload.php');
```

完成上述步骤即可创建连接与服务端进行交互。

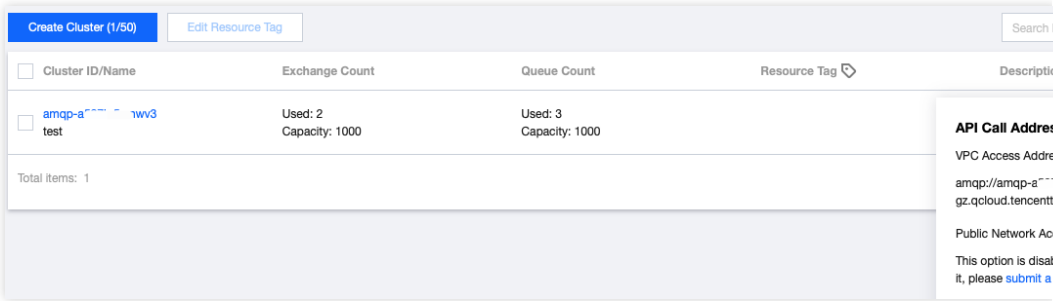
步骤2：发送消息

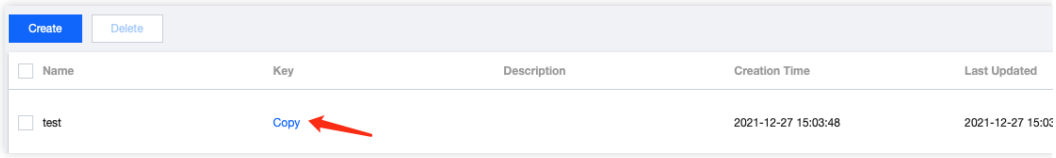
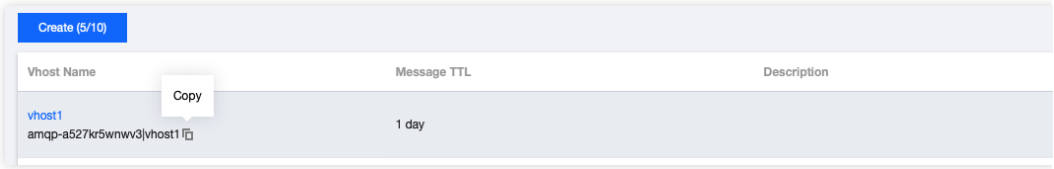
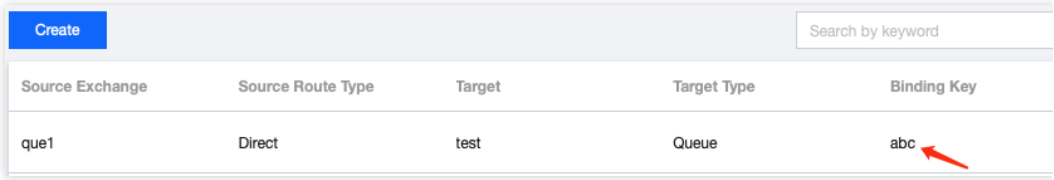
创建并编译生产消息程序（以 direct 类型交换机为例）。



```
require_once('../vendor/autoload.php');

use PhpAmqpLib\\Connection\\AMQPStreamConnection;
use PhpAmqpLib\\Message\\AMQPMessage;

$exchange_name = 'exchange_name';
$exchange_type = 'direct';

// 创建连接
$connection = new AMQPStreamConnection(
    $host,
```

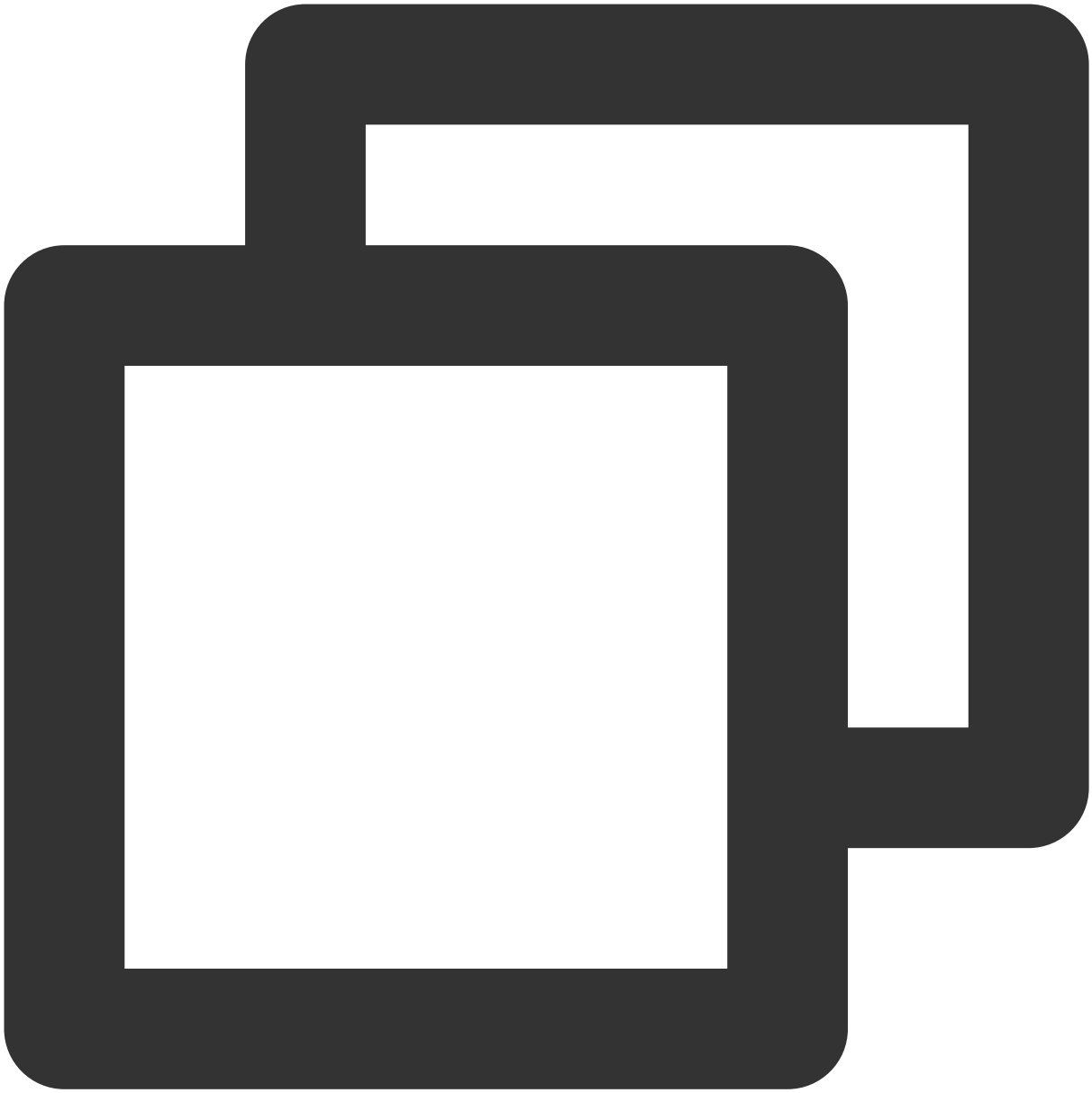
```
    $port,
    $username,
    $password,
    $vhost,
    false,
    'PLAIN');
// 建立通道
$channel = $connection->channel();
// 声明交换机
$channel->exchange_declare($exchange_name, $exchange_type, false, true, false);

// 设定消息路由key
$routing_keys = array('info', 'waring', 'error');

for ($x = 0; $x < count($routing_keys); $x++) {
    // 消息内容
    $msg = new AMQPMessage('This is a direct[' . $routing_keys[$x] . '] message!');
    // 发送消息到指定的交换机并设置routing key
    $channel->basic_publish($msg, $exchange_name, $routing_keys[$x]);

    echo " [Producer(Routing)] Sent '" . $msg->body . "'\\n";
}
// 资源释放
$channel->close();
$connection->close();
```

参数	说明
$exchange_name	Exchange 名称，在控制台 Exchange 列表获取。
$exchange_type	类型需与上述 Exchange 的类型保持一致。
$host	集群接入地址，在**集群管理**页面操作列的**获取接入地址**获取。 
$port	集群接入地址中的端口号。
$username	角色名称，在 **角色管理** 页面复制。

$password	角色密钥，在 **角色管理** 页面复制**密钥**列复制。 
$vhost	Vhost 名称，在控制台 Vhost 页面复制，格式是“**集群 ID + | + vhost 名称**”。 
$routing_keys[$x]	消费者消息队列绑定的 routing key，消息的路由规则，在控制台绑定关系列表的**绑定 Key**列 

步骤3：消费消息

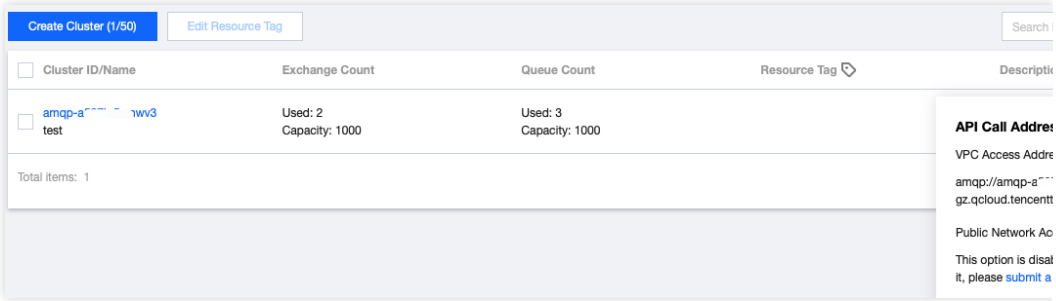
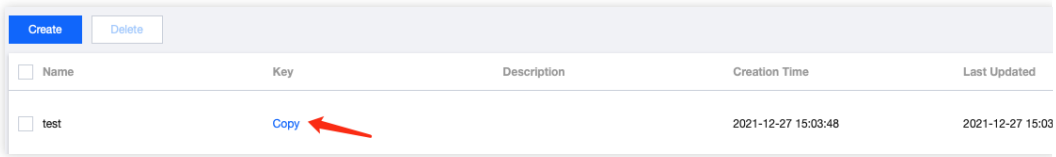
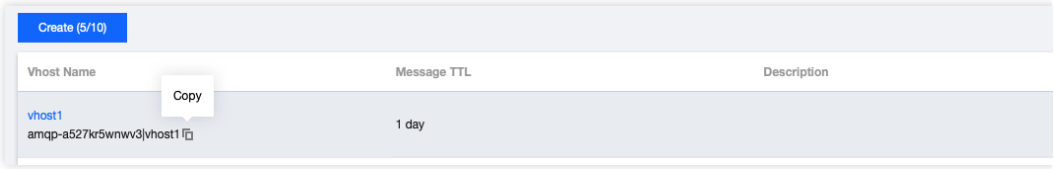
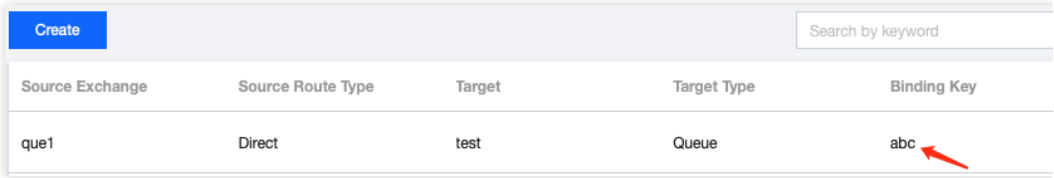
创建并编译消费消息程序。



```
<?php

require_once('../vendor/autoload.php');
require_once('../Constant.php');

use PhpAmqpLib\\Connection\\AMQPStreamConnection;

$exchange_name = 'exchange_name';
$exchange_type = 'direct';
$queue_name = 'route_queue1';
```

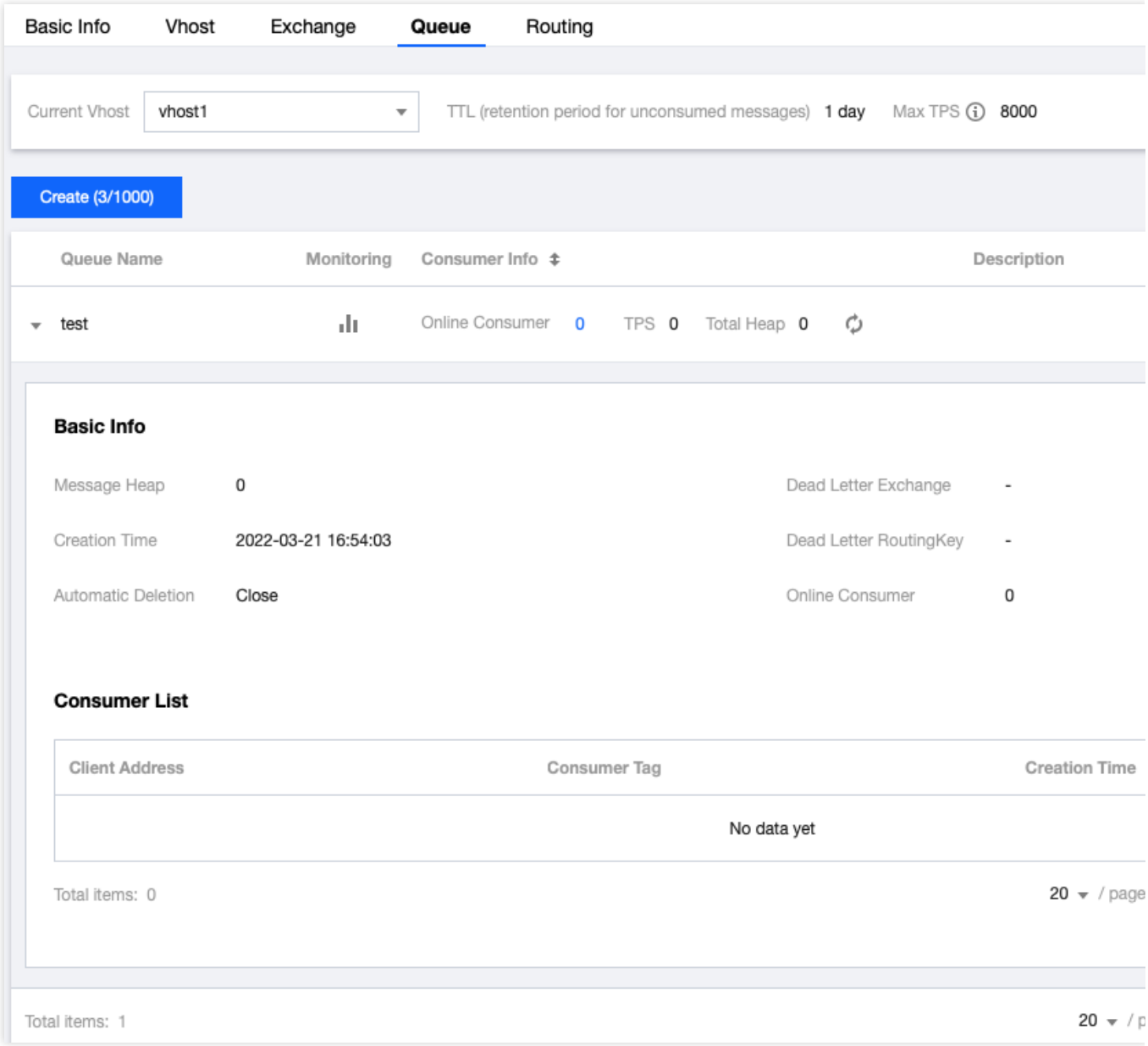
```
// 创建链接
$connection = new AMQPStreamConnection(
    $host,
    $port,
    $username,
    $password,
    $vhost,
    false,
    'PLAIN');
// 建立通道
$channel = $connection->channel();
// 声明交换机
$channel->exchange_declare($exchange_name, $exchange_type, false, true, false);
// 声明消息队列
$channel->queue_declare($queue_name, false, true, false, false);

// 设定队列路由key
$routing_keys = array('info', 'waring', 'error');
for ($x = 0; $x < count($routing_keys); $x++) {
    // 绑定消息队列到指定交换机并设置routingKey
    $channel->queue_bind($queue_name, $exchange_name, $routing_keys[$x]);
}

echo " [Consumer1(Routing: info/waring/error)] Waiting for messages. To exit press

// 消息回调（消息消费逻辑）
$callback = function ($msg) {
    echo ' [Consumer1(Routing: info/waring/error)] Received ', $msg->body, "\\n";
};
// 创建消费者监听指定消息队列
$channel->basic_consume($queue_name, '', false, true, false, false, $callback);

while ($channel->is_open()) {
    $channel->wait();
}
// 关闭资源
$channel->close();
$connection->close();
```

参数	说明
$exchange_name	Exchange 名称，可在控制台 Exchange 列表获取。
$exchange_type	类型需与上述exchange的类型保持一致。
$queue_name	Queue名称，可在控制台 Queue 列表获取。

$host	集群接入地址，在**集群管理**页面操作列的**获取接入地址**获取。 
$port	集群接入地址中的端口号。
$username	角色名称，在 **角色管理** 页面复制。
$password	角色密钥，在 **角色管理** 页面复制**密钥**列复制。 
$vhost	Vhost 名称，在控制台 Vhost 页面复制，格式是“**集群 ID + | + vhost 名称**”。 
$routing_keys[$x]	消息队列支持的routing key。消费者消息队列绑定的 routing key，消息的路由规则，在控制台取。 

步骤4：查看消息

如果您想确认消息是否成功发送至 TDMQ RabbitMQ 版，可以在控制台 **集群管理** > **Queue** 页面查看接入的消费者情况。



说明：

完整示例或其他使用可参见 Demo 或者 RabbitMQ 官方使用文档。

C++ SDK

最近更新时间：2024-01-03 11:45:32

操作场景

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

前提条件

完成资源创建与准备

下载 Demo

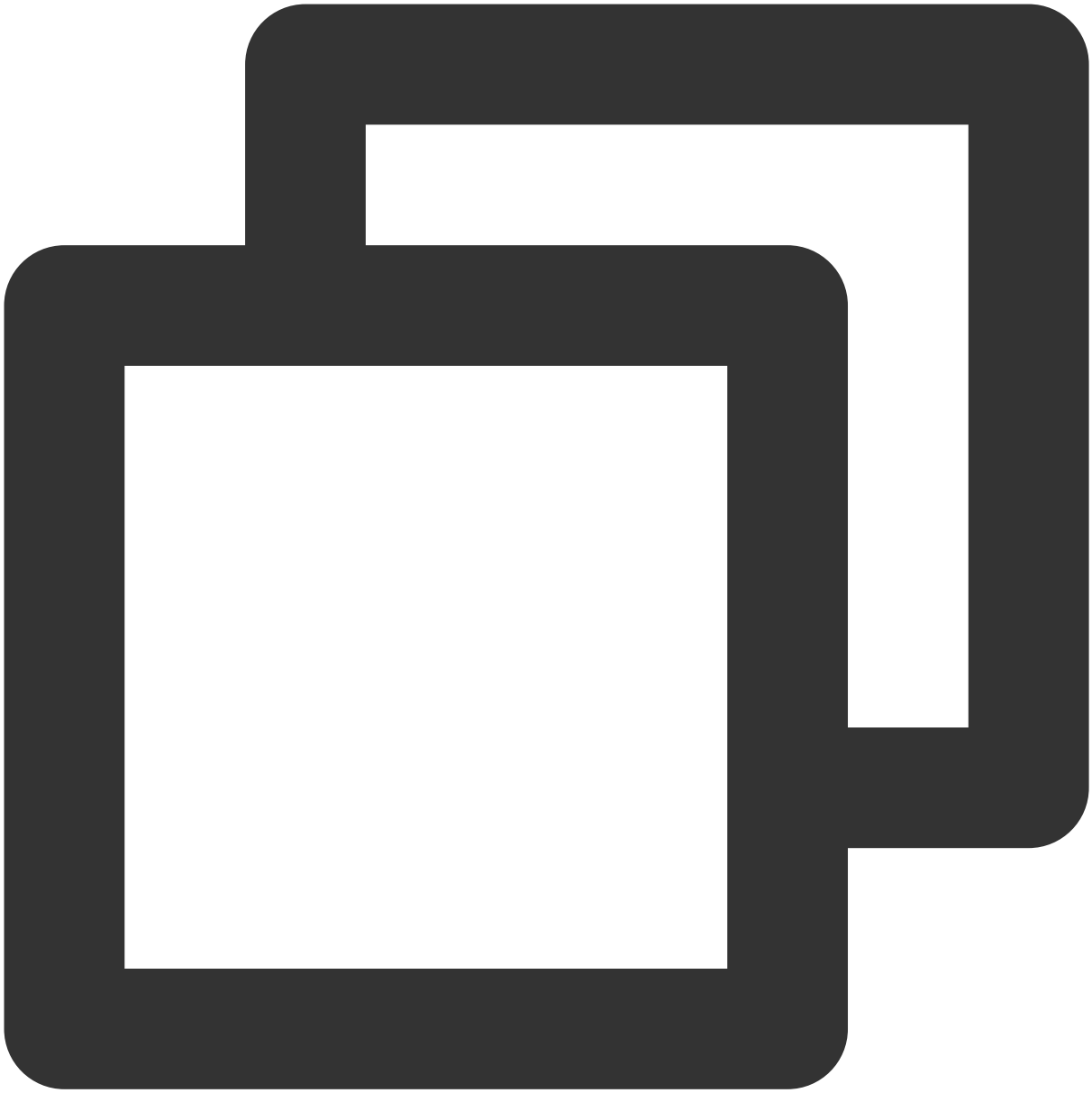
操作步骤

步骤1：环境准备

1. 安装客户端相关的库 C and C++ 库， 本文以 AMQP-CPP 为例。
2. 导入动态库和头文件。

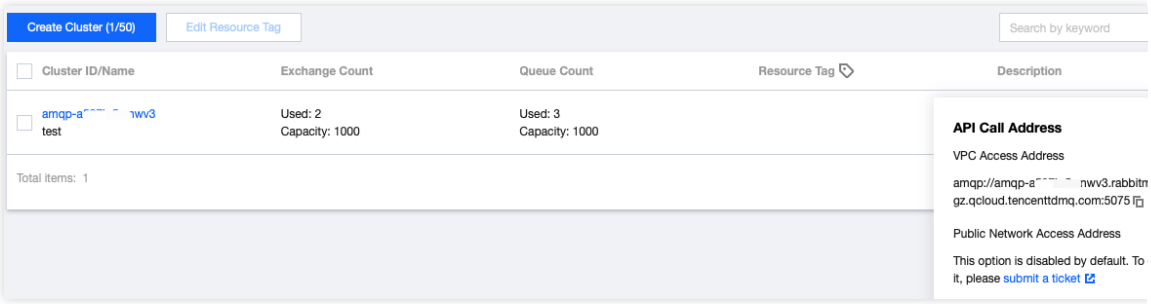
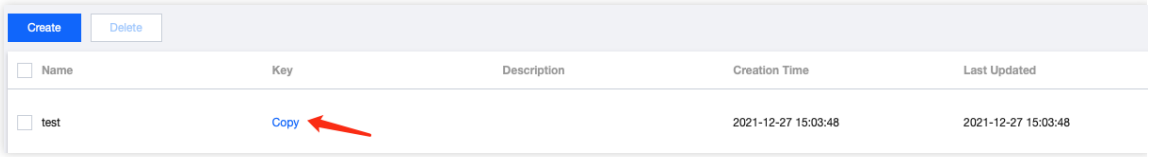
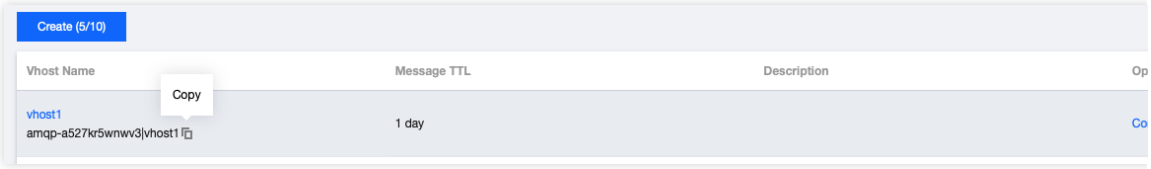
步骤2：生产消息

1. 建立连接。

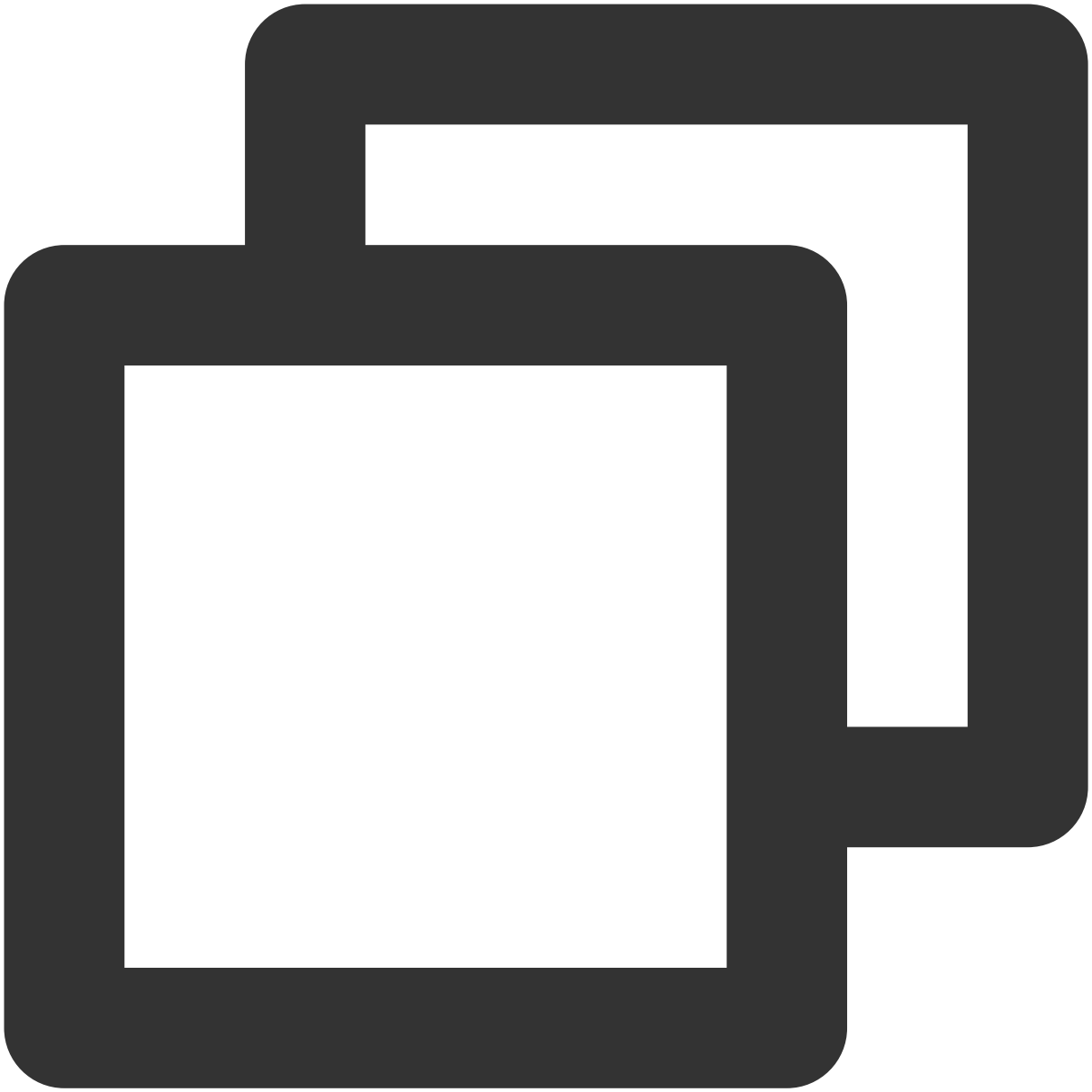


```
auto evbase = event_base_new();
LibEventHandlerMyError hndl(evbase);

// 建立连接
AMQP::TcpConnection connection(&hndl, AMQP::Address(
    "amqp://admin:eyJrZXlJZC...@amqp-xxx.rabbitmq.ap-sh.public.tencenttdmq.com:5
    // 服务地址格式为  amqp://username:password@host:port/vhost
// 建立通道
AMQP::TcpChannel channel(&connection);
channel.onError([&evbase](const char *message) {
    std::cout << "Channel error: " << message << std::endl;
```

```
        event_base_loopbreak(evbase);
    });
```

参数	说明
host	集群接入地址，在集群管理页面操作列的获取接入地址获取。 
port	集群接入地址中的端口号。
username	角色名称，在 角色管理 页面复制。
password	角色密钥，在 角色管理 页面复制密钥列复制。 
vhost	Vhost 名称，在控制台 Vhost 页面复制，格式是**集群 ID + | + vhost 名称**。 

2. 发送消息。

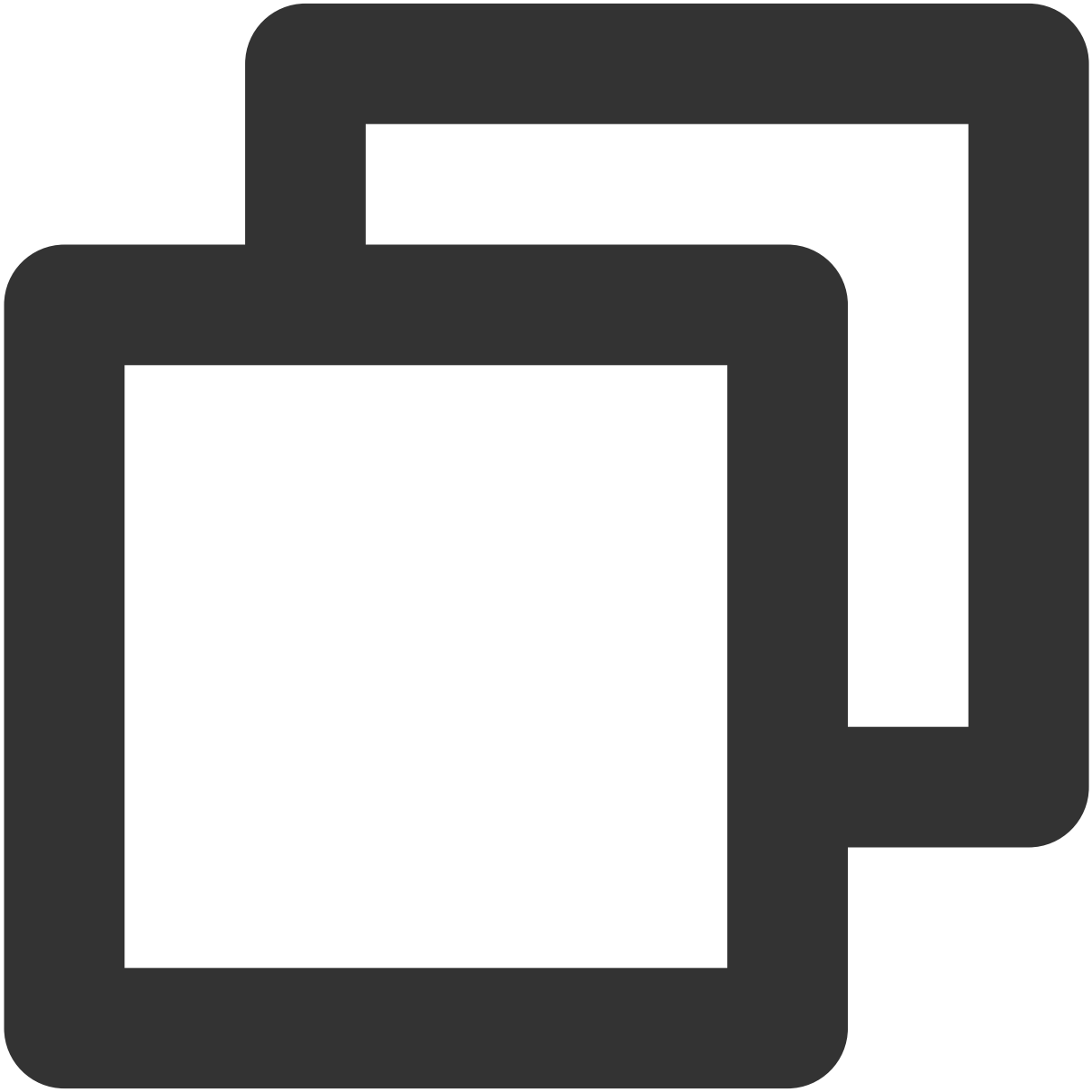


```
// 声明交换机
channel.declareExchange(exchange_name, AMQP::ExchangeType::direct);

// 发送消息到交换机
channel.publish(exchange_name, routing_key, "Hello client this is a info message

event_base_dispatch(evbase);
event_base_free(evbase);
```

参数	说明
exchange_name	Exchange 名称，可在控制台 Exchange 列表获取。
routing_key	消息队列支持的routing key

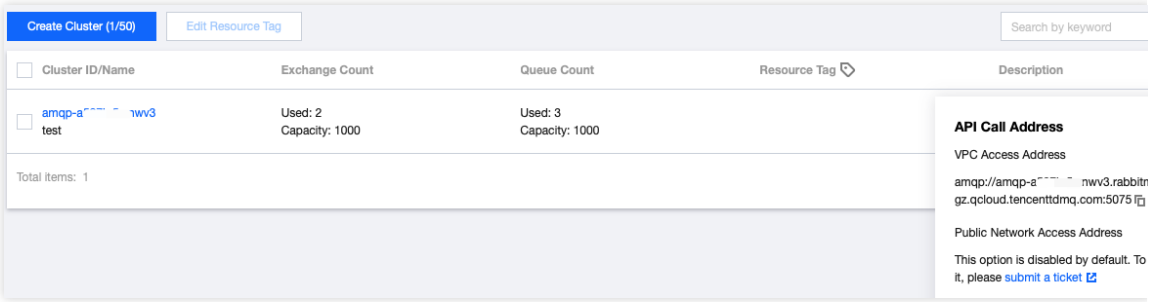
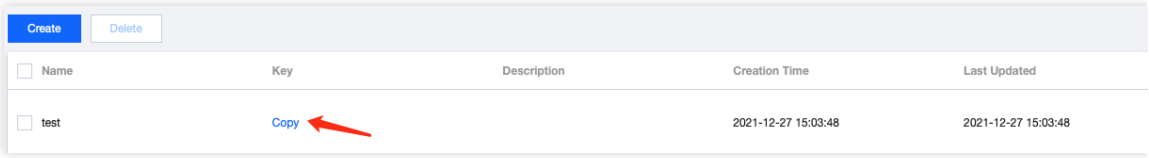
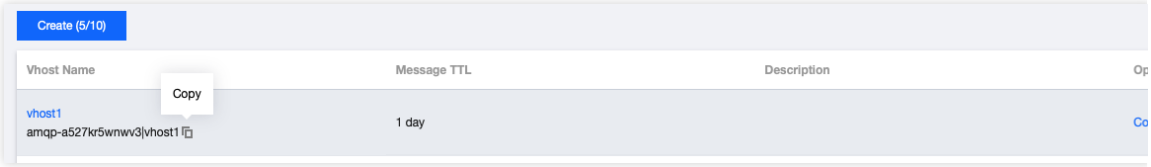
步骤3：消费消息

1. 建立连接。

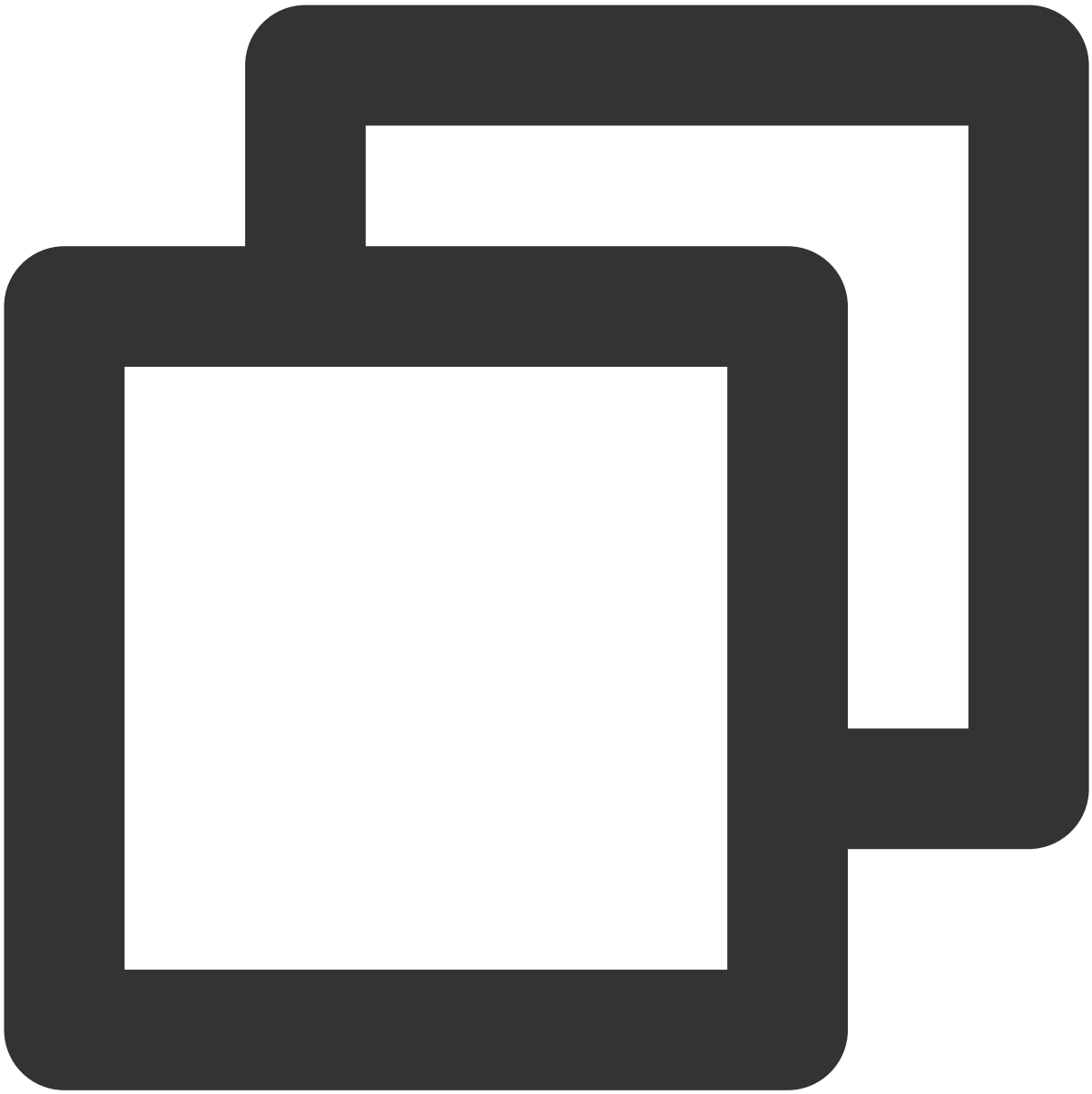


```
ConnHandler handler;
```

```
// 建立连接
AMQP::TcpConnection connection(handler, AMQP::Address(host, part, AMQP::Login(usern
// 建立通道
AMQP::TcpChannel channel(&connection);
channel.onError([&handler](const char *message) {
        std::cout << "Channel error: " << message << std::endl;
        handler.Stop();
});
```

参数	说明
host	集群接入地址，在集群管理页面操作列的获取接入地址获取。 
port	集群接入地址中的端口号。
username	角色名称，在 角色管理 页面复制。
password	角色密钥，在 角色管理 页面复制密钥列复制。 
vhost	Vhost 名称，在控制台 Vhost 页面复制，格式是**集群 ID + | + vhost 名称**。 

2. 声明交换机、消息队列，并将消息队列绑定到交换机

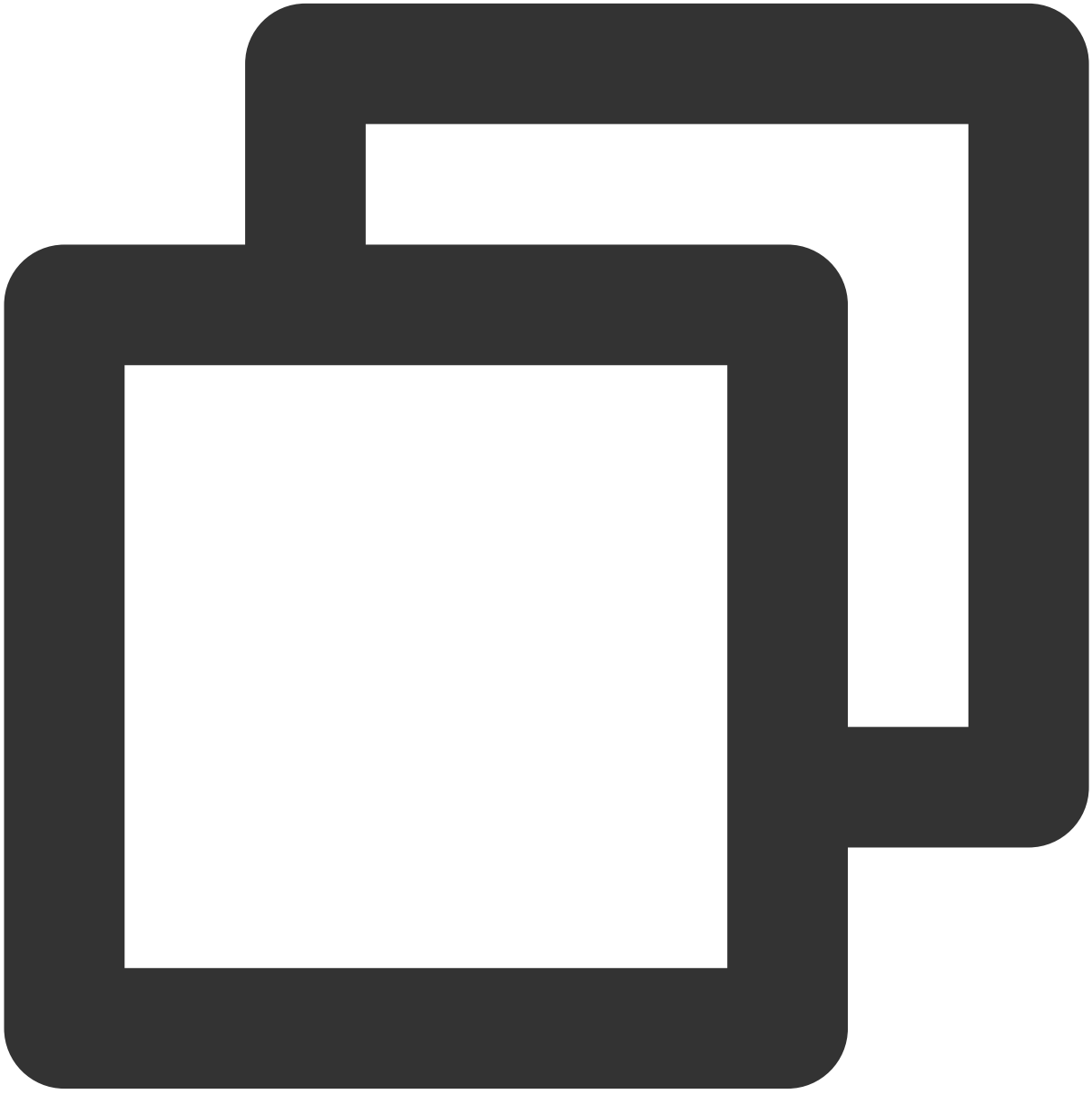


```
// 声明交换机
channel.declareExchange(exchange_name, AMQP::ExchangeType::direct);
// 声明消息队列
channel.declareQueue(queue_name, AMQP::durable);
// 绑定消息队列到交换机
channel.bindQueue(exchange_name, queue_name, routing_key);
```

参数	说明

exchange_name	Exchange 名称，可在控制台 Exchange 列表获取。
queue_name	消息队列名称，可在控制台 Queue 列表获取。
routing_key	消息队列支持的routing key

3. 订阅消息。



```
// 订阅消息
channel.consume(queue_name)
        .onReceived
```

```
        (
        [&channel](const AMQP::Message &msg, uint64_t tag, bool redelivered) {
                // 处理消息
                std::cout << "Received: " << msg.body() << std::endl;
                // 回复ack,   消费失败可回复reject
                channel.ack(tag);
        }
);
handler.Start();
```

参数	说明
queue_name	消息队列名称，可在控制台 Queue 列表获取。

说明：

完整示例或其他使用可参考 Demo 或者 AMQP-CPP 和 AMQP-CPP 示例