消息队列 RocketMQ 版

SDK 文档

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

SDK 文档
兼容性说明

最近更新时间：2025-01-03 14:38:02

RocketMQ 5.0 提供了全新的基于 gRPC 协议的 5.x SDK，新版本 SDK 更加轻量化，多语言支持更好，建议优先使用。同时，消息队列 RocketMQ 版 5.x 系列也支持存量业务继续使用 4.x SDK 访问，兼容性说明如下：

服务端版本	客户端版本		兼容性
5.x	5.x SDK		完全兼容
	4.x SDK	版本 >= 4.9.5	• PushConsumer CONSUME_FROM_TIMESTAMP 暂不生效（控制台可以重置位点）。 • 消费者 setPullBatchSize 最大值为 32。 • 事务消息存在兼容性问题，事务提交或回查可能会失败，暂不建议使用。
		版本 < 4.9.5	• PushConsumer CONSUME_FROM_TIMESTAMP 暂不生效（控制台可以重置位点）。 • 消费者 setPullBatchSize 最大值为 32。 • PullConsumer 消费暂不支持。 • 事务消息存在兼容性问题，事务提交或回查可能会失败，暂不建议使用。

5.x SDK
Java SDK 使用

最近更新时间：2025-06-10 16:17:03

操作场景

消息队列 RocketMQ 版支持多种语言的 SDK 收发不同类型的消息，本文以调用 Java SDK 为例介绍通过 5.x SDK 连接消息队列 RocketMQ 版服务端实现普通消息收发的操作过程，

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 Java 项目中引入相关依赖，以 Maven 工程为例，在 pom.xml 添加以下依赖：

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.6</version>
        </dependency>
</dependencies>
```

> **注意：**
> 如果您是在 Spring Boot 环境下使用，可能会遇到 annotations-api 依赖冲突，这时候，可以增加排除依赖即可。

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.6</version>
```

```
            <exclusions>
                <exclusion>
                    <groupId>org.apache.tomcat</groupId>
                    <artifactId>annotations-api</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
</dependencies>
```

步骤2. 生产消息

在已创建的 Java 工程中，创建发送消息程序并运行。

```
public class NormalMessageSyncProducer {
    private static final Logger log =
LoggerFactory.getLogger(NormalMessageSyncProducer.class);

    private NormalMessageSyncProducer() {
    }

    public static void main(String[] args) throws ClientException,
IOException {
        final ClientServiceProvider provider =
ClientServiceProvider.loadService();

        // 添加配置的ak和sk
        String accessKey = "yourAccessKey"; //ak，可以在控制台“权限管理”页面
获取
        String secretKey = "yourSecretKey"; //sk，可以在控制台“权限管理”页面
获取
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // 填写腾讯云提供的接入地址
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8080";
        ClientConfiguration clientConfiguration =
ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String topic = "yourNormalTopic";
```

```
        // In most case, you don't need to create too many producers,
singleton pattern is recommended.
        final Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the topic name(s), which is optional but recommended.
It makes producer could prefetch the topic
            // route before message publishing.
            .setTopics(topic)
            // May throw {@link ClientException} if the producer is not
initialized.
            .build();
        // Define your message body.
        byte[] body = "This is a normal message for Apache
RocketMQ".getBytes(StandardCharsets.UTF_8);
        String tag = "yourMessageTagA";
        final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // Message secondary classifier of message besides topic.
            .setTag(tag)
            // Key(s) of the message, another way to mark message
besides message id.
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            .build();
        try {
            final SendReceipt sendReceipt = producer.send(message);
            log.info("Send message successfully, messageId={}",
sendReceipt.getMessageId());
        } catch (Throwable t) {
            log.error("Failed to send message", t);
        }
        // Close the producer when you don't need it anymore.
        producer.close();
    }
}
```

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。

secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
topic	Topic 的名称，在控制台 Topic 管理页面复制。 

步骤3. 消费消息

在已创建的 Java 工程中，创建订阅普通消息程序并运行。
腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种类型的消费者客户端，分别为 Push Consumer 和 Simple Consumer。以下代码示例以 Push Consumer 为例：

```
public class NormalPushConsumer {
    private static final Logger log =
LoggerFactory.getLogger(NormalPushConsumer.class);
```

```
    private NormalPushConsumer() {
    }

    public static void main(String[] args) throws ClientException,
IOException, InterruptedException {
        final ClientServiceProvider provider =
ClientServiceProvider.loadService();

        // 添加配置的ak和sk
        String accessKey = "yourAccessKey"; //ak，可以在控制台“权限管理”页面
获取
        String secretKey = "yourSecretKey"; //sk，可以在控制台“权限管理”页面
获取
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // 填写腾讯云提供的接入地址
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8080";
        ClientConfiguration clientConfiguration =
ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        // In most case, you don't need to create too many consumers,
singleton pattern is recommended.
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
            .setMessageListener(messageView -> {
                // Handle the received message and return consume
result.
                log.info("Consume message={}", messageView);
```

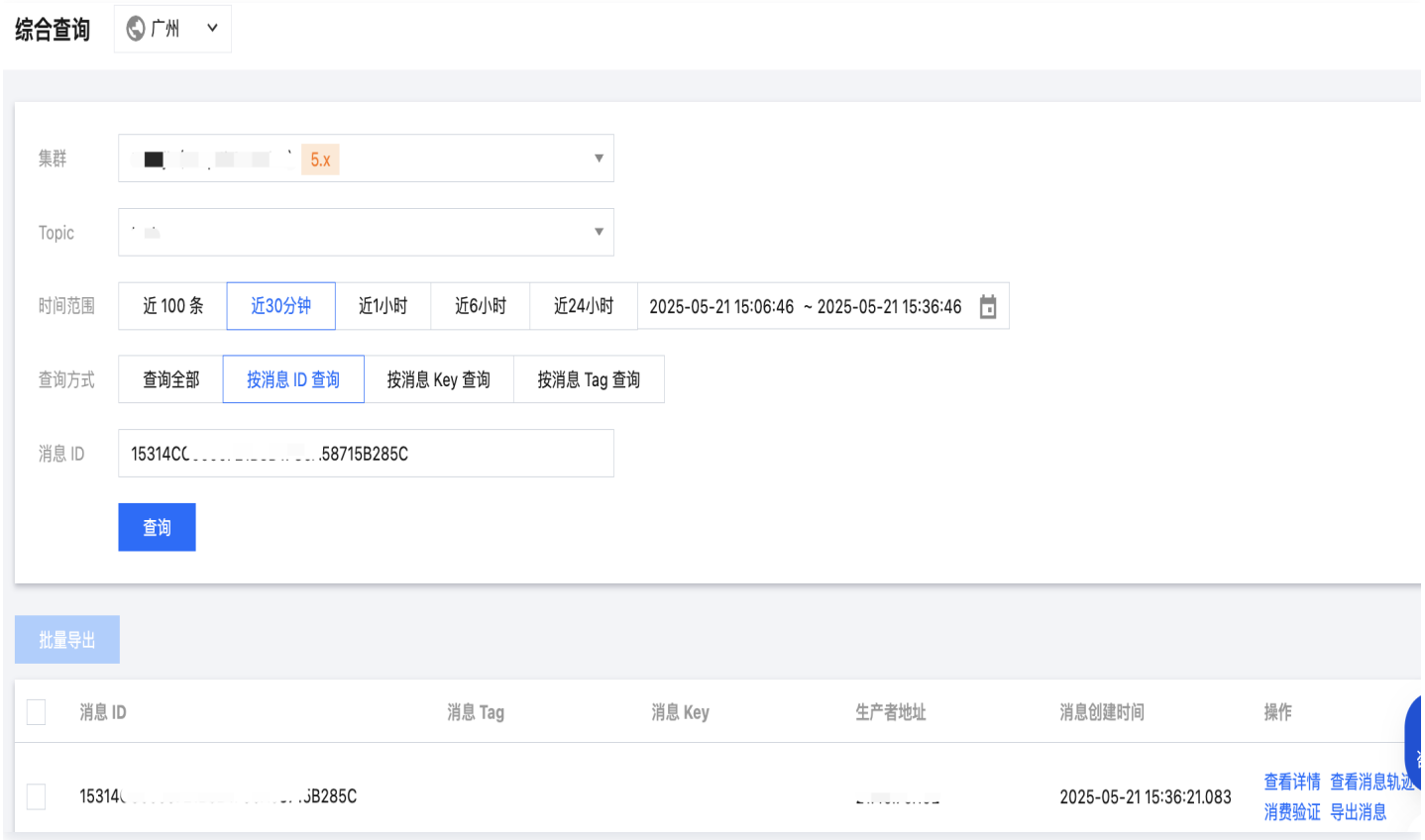
```
                return ConsumeResult.SUCCESS;
            })
            .build();
        // Block the main thread, no need for production environment.
        Thread.sleep(Long.MAX_VALUE);
        // Close the push consumer when you don't need it anymore.
        pushConsumer.close();
    }
}
```

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
consumerGroup	消费者组名称，在控制台 Group 管理页面复制。

	
topic	Topic 的名称，在控制台 Topic 管理页面复制。 

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询 > 综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



Go SDK 使用

最近更新时间：2025-06-10 16:17:03

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 1.13 版本以上的 golang
- 下载 Demo

操作步骤

步骤1：安装 Go 依赖库

在 Golang 项目中引入相关依赖，以 `go get` 为例，使用如下命令：

```
go get github.com/apache/rocketmq-clients/golang/v5
```

步骤2：生产消息

```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "strconv"
    "time"

    rmq_client "github.com/apache/rocketmq-clients/golang/v5"
    "github.com/apache/rocketmq-clients/golang/v5/credentials"
)

const (
    Topic = "xxxxxx"
    // Endpoint 填写腾讯云提供的接入地址
    Endpoint = "xxxxxx"
```

```
    // AccessKey 添加配置的ak
    AccessKey = "xxxxxx"
    // SecretKey 添加配置的sk
    SecretKey = "xxxxxx"
)

func main() {
    os.Setenv("mq.consoleAppender.enabled", "true")
    rmq_client.ResetLogger()
    // In most case, you don't need to create many producers, singleton
pattern is more recommended.
    producer, err := rmq_client.NewProducer(&rmq_client.Config{
        Endpoint: Endpoint,
        Credentials: &credentials.SessionCredentials{
            AccessKey:    AccessKey,
            AccessSecret: SecretKey,
        },
    },
        rmq_client.WithTopics(Topic),
    )
    if err != nil {
        log.Fatal(err)
    }
    // start producer
    err = producer.Start()
    if err != nil {
        log.Fatal(err)
    }
    // graceful stop producer
    defer producer.GracefulStop()

    for i := 0; i < 10; i++ {
        // new a message
        msg := &rmq_client.Message{
            Topic: Topic,
            Body:  []byte("this is a message : " + strconv.Itoa(i)),
        }
        // set keys and tag
        msg.SetKeys("a", "b")
        msg.SetTag("ab")
        // send message in sync
        resp, err := producer.Send(context.TODO(), msg)
        if err != nil {
```

```
            log.Fatal(err)
        }
        for i := 0; i < len(resp); i++ {
            fmt.Printf("%#v\n", resp[i])
        }
        // wait a moment
        time.Sleep(time.Second * 1)
    }
}
```

参数	说明
AccessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
SecretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
Endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
Topic	Topic 的名称，在控制台 Topic 管理页面复制。

步骤3：消费消息

腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种消费模式，分别为 Push Consumer 和 Simple Consumer。

> **说明：**
> 社区版 Golang SDK 目前仅支持 Simple Consumer。

以下代码示例以 Simple Consumer 为例：

```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "os/signal"
    "sync"
    "syscall"
    "time"

    rmq_client "github.com/apache/rocketmq-clients/golang/v5"
    "github.com/apache/rocketmq-clients/golang/v5/credentials"
)

const (
    Topic         = "xxxxxx"
    ConsumerGroup = "xxxxxx"
    // Endpoint 填写腾讯云提供的接入地址
    Endpoint = "xxxxxx"
    // AccessKey 添加配置的ak
    AccessKey = "xxxxxx"
    // SecretKey 添加配置的sk
```

```
    SecretKey = "xxxxxx"
)

var (
    // maximum waiting time for receive func
    awaitDuration = time.Second * 5
    // maximum number of messages received at one time
    maxMessageNum int32 = 16
    // invisibleDuration should > 20s
    invisibleDuration = time.Second * 20
    // receive concurrency
    receiveConcurrency = 6
)

func main() {
    // log to console
    os.Setenv("mq.consoleAppender.enabled", "true")
    rmq_client.ResetLogger()
    // In most case, you don't need to create many consumers, singleton
pattern is more recommended.
    simpleConsumer, err :=
rmq_client.NewSimpleConsumer(&rmq_client.Config{
        Endpoint:      Endpoint,
        ConsumerGroup: ConsumerGroup,
        Credentials: &credentials.SessionCredentials{
            AccessKey:    AccessKey,
            AccessSecret: SecretKey,
        },
    },
        rmq_client.WithAwaitDuration(awaitDuration),

rmq_client.WithSubscriptionExpressions(map[string]*rmq_client.FilterExpr
ession{
            Topic: rmq_client.SUB_ALL,
        }),
    )
    if err != nil {
        log.Fatal(err)
    }
    // start simpleConsumer
    err = simpleConsumer.Start()
    if err != nil {
        log.Fatal(err)
```

```
    }
    // graceful stop simpleConsumer
    defer func() {
        if r := recover(); r != nil {
            fmt.Println(r)
        }
        _ = simpleConsumer.GracefulStop()
    }()

    fmt.Println("start receive message")

    // Each Receive call will only select one broker queue to pop
messages.
    // Enable multiple consumption goroutines to reduce message end-to-
end latency.
    ch := make(chan struct{})
    wg := &sync.WaitGroup{}
    for i := 0; i < receiveConcurrency; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            for {
                select {
                case <-ch:
                    return
                default:
                    mvs, err := simpleConsumer.Receive(context.TODO(),
maxMessageNum, invisibleDuration)
                    if err != nil {
                        fmt.Println("receive message error: " +
err.Error())
                    }
                    // ack message
                    for _, mv := range mvs {
                        fmt.Println(mv)
                        if err := simpleConsumer.Ack(context.TODO(),
mv); err != nil {
                            fmt.Println("ack message error: " +
err.Error())
                        }
                    }
                }
            }
```

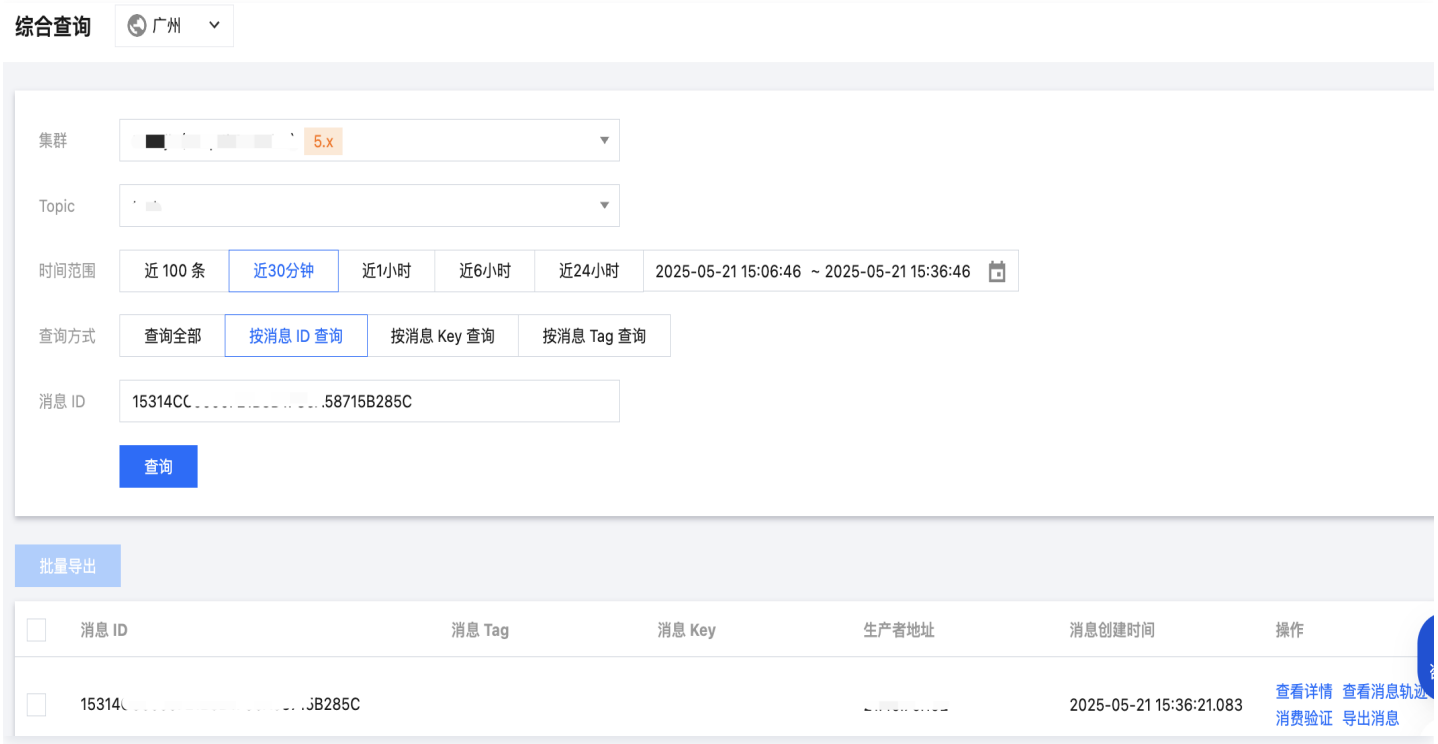
```
        }()
    }

    exit := make(chan os.Signal, 1)
    signal.Notify(exit, syscall.SIGINT, syscall.SIGTERM)

    // wait for exit
    <-exit
    close(ch)
    wg.Wait()
}
```

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
consumerGroup	消费者组名称，在控制台 Group 管理页面复制。

	
topic	Topic 的名称，在控制台 Topic 管理页面复制。 

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询 > 综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



C++ SDK 使用

最近更新时间：2025-06-11 11:56:12

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
- 安装支持 C++11 的编译套件
- 安装 bazel（5.2.0）或 cmake（3.13 及以上）
- 如果使用 cmake 编译，需要提前安装 grpc，推荐安装 1.46.3 版本，较高版本与 SDK 存在不兼容。
- 下载 Demo

操作步骤

步骤1：安装 C++ SDK

- 安装 SDK。

> ⚠ **注意：**
> 腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列暂不支持 TLS，需要拉取 补丁 。

步骤2. 生产消息

```
#include <algorithm>
#include <atomic>
#include <iostream>
#include <memory>
#include <random>
#include <string>
#include <system_error>

#include "rocketmq/CredentialsProvider.h"
#include "rocketmq/Logger.h"
#include "rocketmq/Message.h"
#include "rocketmq/Producer.h"

using namespace ROCKETMQ_NAMESPACE;
```

```
const std::string &alphaNumeric() {
    static std::string
alpha_numeric("0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU
VWXYZ");
    return alpha_numeric;
}

std::string randomString(std::string::size_type len) {
    std::string result;
    result.reserve(len);
    std::random_device rd;
    std::mt19937 generator(rd());
    std::string source(alphaNumeric());
    std::string::size_type generated = 0;
    while (generated < len) {
        std::shuffle(source.begin(), source.end(), generator);
        std::string::size_type delta = std::min({len - generated,
source.length()});
        result.append(source.substr(0, delta));
        generated += delta;
    }
    return result;
}

static const std::string topic = "xxx";
// 填写腾讯云提供的接入地址
static const std::string access_point = "rmq-
xxx.rocketmq.xxxtencenttdmq.com:8081";
// 添加配置的ak和sk
static const std::string access_key = "xxx";
static const std::string access_secret = "xxx";
static const uint32_t total = 32;
static const int32_t message_body_size = 128;


int main(int argc, char *argv[]) {
    CredentialsProviderPtr credentials_provider;
    if (!access_key.empty() && !access_secret.empty()) {
        credentials_provider =
std::make_shared<StaticCredentialsProvider>(access_key, access_secret);
    }
```

```
    // In most case, you don't need to create too many producers,
singletion pattern is recommended.
    auto producer = Producer::newBuilder()
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)

.withCredentialsProvider(credentials_provider)
                                       .withSsl(false)
                                       .build())
            .withTopics({topic})
            .build();

    std::atomic_bool stopped;
    std::atomic_long count(0);

    auto stats_lambda = [&] {
        while (!stopped.load(std::memory_order_relaxed)) {
            long cnt = count.load(std::memory_order_relaxed);
            while (count.compare_exchange_weak(cnt, 0)) {
                break;
            }
            std::this_thread::sleep_for(std::chrono::seconds(1));
            std::cout << "QPS: " << cnt << std::endl;
        }
    };

    std::thread stats_thread(stats_lambda);

    std::string body = randomString(message_body_size);

    try {
        for (std::size_t i = 0; i < total; ++i) {
            auto message = Message::newBuilder()
                    .withTopic(topic)
                    .withTag("TagA")
                    .withKeys({"Key-" + std::to_string(i)})
                    .withBody(body)
                    .build();
            std::error_code ec;
            SendReceipt send_receipt = producer.send(std::move(message),
ec);
            if (ec) {
```

```
                std::cerr << "Failed to publish message to " << topic <<
". Cause: " << ec.message() << std::endl;
            } else {
                std::cout << "Publish message to " << topic << " OK.
Message-ID: " << send_receipt.message_id
                          << std::endl;
                count++;
            }
        }
    } catch (...) {
        std::cerr << "Ah...No!!!" << std::endl;
    }
    stopped.store(true, std::memory_order_relaxed);
    if (stats_thread.joinable()) {
        stats_thread.join();
    }

    return EXIT_SUCCESS;
}
```

参数	说明
access_key	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
access_secret	角色名称，在控制台的集群权限页面 SecretKey 列复制。
access_point	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。

topic	Topic 的名称，在控制台 Topic 管理页面复制。

步骤3. 消费消息

腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种类型的客户端，分别为 Push Consumer 和 Simple Consumer。
以下代码示例以 Push Consumer 为例：

```
#include <chrono>
#include <iostream>
#include <mutex>
#include <thread>

#include "rocketmq/Logger.h"
#include "rocketmq/PushConsumer.h"

using namespace ROCKETMQ_NAMESPACE;

static const std::string topic = "xxx";
// 填写腾讯云提供的接入地址
static const std::string access_point = "rmq-
xxx.rocketmq.xxxtencenttdmq.com:8081";
// 添加配置的ak和sk
```

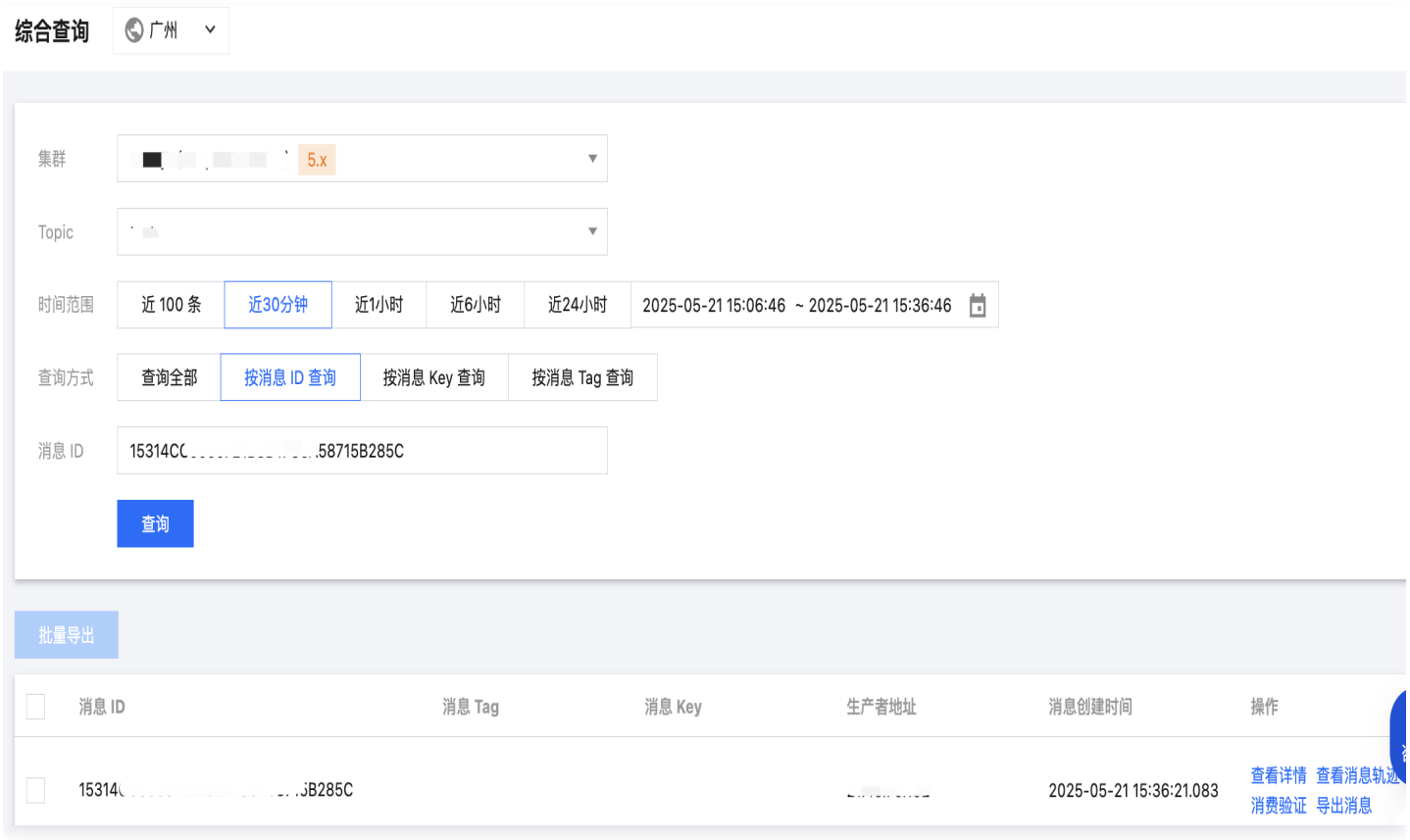
```
static const std::string access_key = "xxx";
static const std::string access_secret = "xxx";
static const std::string group = "group-xxx";

int main(int argc, char *argv[]) {
    auto &logger = getLogger();
    logger.setConsoleLevel(Level::Info);
    logger.setLevel(Level::Info);
    logger.init();

    std::string tag = "*";

    auto listener = [](const Message &message) {
        std::cout << "Received a message[topic=" << message.topic() <<
", MsgId=" << message.id() << "]" << std::endl;
        return ConsumeResult::SUCCESS;
    };

    CredentialsProviderPtr credentials_provider;
    if (!access_key.empty() && !access_secret.empty()) {
        credentials_provider =
std::make_shared<StaticCredentialsProvider>(access_key, access_secret);
    }

    // In most case, you don't need to create too many consumers,
singletion pattern is recommended.
    auto push_consumer = PushConsumer::newBuilder()
            .withGroup(group)
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)

.withRequestTimeout(std::chrono::seconds(3))

.withCredentialsProvider(credentials_provider)
                                       .withSsl(false)
                                       .build())
            .withConsumeThreads(4)
            .withListener(listener)
            .subscribe(topic, tag)
            .build();

    std::this_thread::sleep_for(std::chrono::minutes(30));
```

```
    return EXIT_SUCCESS;
}
```

参数	说明
access_key	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
access_secret	角色名称，在控制台的集群权限页面 SecretKey 列复制。
access_point	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
group	消费者组名称，在控制台 Group 管理页面复制。 
topic	Topic 的名称，在控制台 Topic 管理页面复制。

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询 > 综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



C# SDK 使用

最近更新时间：2025-06-13 10:57:42

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
- 安装 DotNet 环境
- 下载 Demo

操作步骤

步骤1：安装 RocketMQ5 sdk 依赖库

在 C# 项目中引入相关依赖，使用如下命令：

```
dotnet add package RocketMQ.Client --version 5.2.0-rc1
```

步骤2：生产消息

```
using System.Text;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Org.Apache.Rocketmq;
namespace examples
{
    internal static class ProducerNormalMessageDemo
    {
        static readonly ILoggerFactory factory =
LoggerFactory.Create(builder => builder.AddConsole());
        static ILogger logger =
factory.CreateLogger("Program_Producer");
        internal static async Task QuickStart()
        {
            // Enable the switch if you use .NET Core 3.1 and want to
disable TLS/SSL.

AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2Unencrypte
```

```
dSupport", true);
            const string accessKey = "yourAccessKey";
            const string secretKey = "yourSecretKey";


            // Credential provider is optional for client configuration.
            var credentialsProvider = new
StaticSessionCredentialsProvider(accessKey, secretKey);
            const string endpoints = "腾讯云官网接入地址:8080";
            var clientConfig = new ClientConfig.Builder()
                .SetEndpoints(endpoints)
                .SetCredentialsProvider(credentialsProvider)
                .Build();


            const string topic = "topicName";
            // In most case, you don't need to create too many
producers, single pattern is recommended.
            // Producer here will be closed automatically.
            var producer = await new Producer.Builder()
                // Set the topic name(s), which is optional but
recommended.
                // It makes producer could prefetch the topic route
before message publishing.
                .SetTopics(topic)
                .SetClientConfig(clientConfig)
                .Build();


            // Define your message body.
            var bytes = Encoding.UTF8.GetBytes("foobar");
            const string tag = "yourMessageTagA";
            var message = new Message.Builder()
                .SetTopic(topic)
                .SetBody(bytes)
                .SetTag(tag)
                // You could set multiple keys for the single message
actually.
                .SetKeys("yourMessageKey-7044358f98fc")
                .Build();


            var sendReceipt = await producer.Send(message);
```

```
            logger.LogInformation($"Send message successfully,
messageId={sendReceipt.MessageId}");


            // Close the producer if you don't need it anymore.
            await producer.DisposeAsync();


        }
    }
}
```

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
topic	Topic 的名称，在控制台 Topic 管理页面复制。

步骤3：消费消息

腾讯云消息队列 TDMQ RocketMQ 版 5.x 系列支持两种消费模式，分别为 Push Consumer 和 Simple Consumer。

> **说明：**
> 社区版 C# SDK 在 5.2.0-rc1 之后支持 Push Consumer

以下代码示例以 Simple Consumer 为例（在消息量特别少的情况下，使用单线程的 Simple Consumer 消费会有一点延迟，如果业务评估确实消息量少，建议开启多线程拉取，或者使用 Push Consumer）：

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Org.Apache.Rocketmq;


namespace examples
{
    internal static class SimpleConsumerExample
    {
        static readonly ILoggerFactory factory =
LoggerFactory.Create(builder => builder.AddConsole());
        static ILogger logger =
factory.CreateLogger("Program_Consumer");
        internal static async Task QuickStart()
        {
            // Enable the switch if you use .NET Core 3.1 and want to
disable TLS/SSL.

AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2Unencrypte
dSupport", true);
            const string accessKey = "yourAccessKey";
```

```
            const string secretKey = "yourSecretKey";


            // Credential provider is optional for client configuration.
            var credentialsProvider = new
StaticSessionCredentialsProvider(accessKey, secretKey);
            const string endpoints = "rmq-
xxx.rocketmq.gz.qcloud.tencenttdmq.com:8080"; // 腾讯云接入点
            var clientConfig = new ClientConfig.Builder()
                .SetEndpoints(endpoints)
                .SetCredentialsProvider(credentialsProvider)
                .Build();


            // Add your subscriptions.
            const string consumerGroup = "yourConsumerGroup";
            const string topic = "yourTopic";
            var subscription = new Dictionary<string, FilterExpression>
                { { topic, new FilterExpression("*") } };
            // In most case, you don't need to create too many
consumers, single pattern is recommended.
            var simpleConsumer = await new SimpleConsumer.Builder()
                .SetClientConfig(clientConfig)
                .SetConsumerGroup(consumerGroup)
                .SetAwaitDuration(TimeSpan.FromSeconds(15))
                .SetSubscriptionExpression(subscription)
                .Build();


            while (true)
            {
                var messageViews = await simpleConsumer.Receive(16,
TimeSpan.FromSeconds(15));
                foreach (var message in messageViews)
                {
                    logger.LogInformation(
                        $"Received a message, topic={message.Topic},
message-id={message.MessageId}, body-size={message.Body.Length}");
                    await simpleConsumer.Ack(message);
                    logger.LogInformation($"Message is acknowledged
successfully, message-id={message.MessageId}");
                    // await
simpleConsumer.ChangeInvisibleDuration(message,
```

```
TimeSpan.FromSeconds(15));
                    // Logger.LogInformation($"Changing message
invisible duration successfully, message=id={message.MessageId}");
                }
            }
            // Close the simple consumer if you don't need it anymore.
            // await simpleConsumer.DisposeAsync();
        }
    }
}
```

Push Consumer

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Org.Apache.Rocketmq;

namespace examples
{
    public class PushConsumerExample
    {
        private static readonly ILogger Logger =
MqLogManager.CreateLogger(typeof(PushConsumerExample).FullName);

        private static readonly string Endpoint =
Environment.GetEnvironmentVariable("ROCKETMQ_ENDPOINT");

        internal static async Task QuickStart()
        {
            const string accessKey = "yourAccessKey";
            const string secretKey = "yourSecretKey";
            // Enable the switch if you use .NET Core 3.1 and want to
disable TLS/SSL.
            //
AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2Unencrypte
dSupport", true);

            // Credential provider is optional for client configuration.
            var credentialsProvider = new
StaticSessionCredentialsProvider(accessKey, secretKey);
```

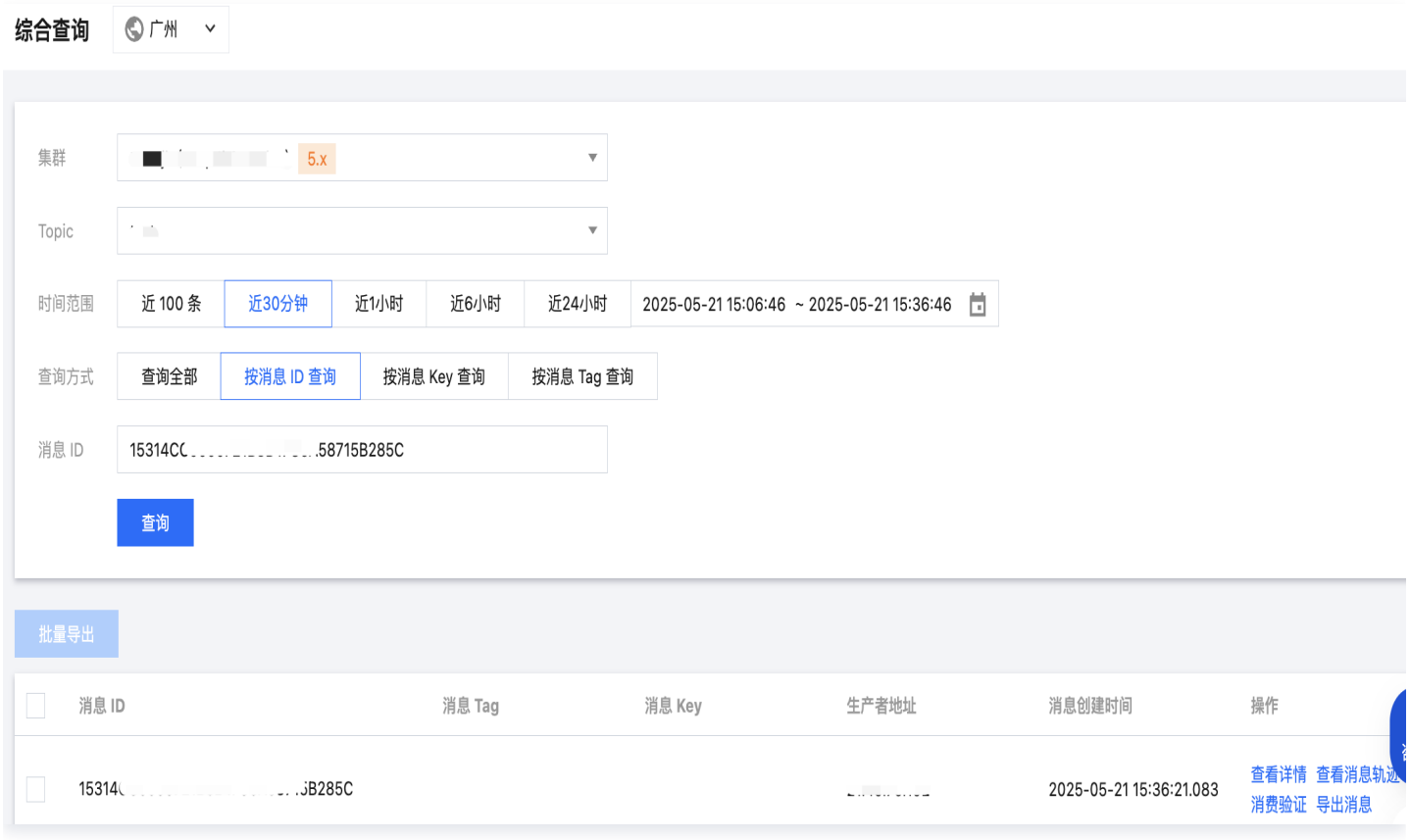
```
            const string endpoints = "rmq-
xxx.rocketmq.gz.qcloud.tencenttdmq.com:8080"; // 腾讯云接入点
            var clientConfig = new ClientConfig.Builder()
                .SetEndpoints(Endpoint)
                .SetCredentialsProvider(credentialsProvider)
                .Build();

            // Add your subscriptions.
            const string consumerGroup = "yourConsumerGroup";
            const string topic = "yourTopic";
            var subscription = new Dictionary<string, FilterExpression>
                { { topic, new FilterExpression("*") } };

            var pushConsumer = await new PushConsumer.Builder()
                .SetClientConfig(clientConfig)
                .SetConsumerGroup(consumerGroup)
                .SetSubscriptionExpression(subscription)
                .SetMessageListener(new CustomMessageListener())
                .Build();

            Thread.Sleep(Timeout.Infinite);

            // Close the push consumer if you don't need it anymore.
            // await pushConsumer.DisposeAsync();
        }

        private class CustomMessageListener : IMessageListener
        {
            public ConsumeResult Consume(MessageView messageView)
            {
                // Handle the received message and return consume
result.
                Logger.LogInformation($"Consume message={messageView}");
                return ConsumeResult.SUCCESS;
            }
        }
    }
}
```

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。

secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
consumerGroup	消费者组名称，在控制台 Group 管理页面复制。 
topic	Topic 的名称，在控制台 Topic 管理页面复制。 

步骤4：查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询** > **综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



Node.js SDK 使用

最近更新时间：2025-06-11 11:56:12

操作场景

本文以调用 Node.js 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
- 安装好 Node.js 开发环境，例如 npm 等。
- 下载 Demo 。

操作步骤

步骤1：安装 Node.js SDK 到项目中

1. 安装 SDK：

```
npm i rocketmq-client-nodejs
```

2. 安装完成后，项目的 packge.json 文件中将会有对应的依赖。

步骤2. 生产消息

```
import { Producer } from 'rocketmq-client-nodejs';

const producer = new Producer({
  endpoints: 'rmq-xxx.rocketmq.gz.qcloud.tencenttdmq.com:8080', // 腾讯云
接入点
  sessionCredentials: {
    accessKey: 'yourAccessKey', // ak
    accessSecret: 'yourSecretKey', // sk
  }
});
await producer.startup();

const receipt = await producer.send({
  topic: 'TopicTest', // 在管控创建的 topic
  body: Buffer.from(JSON.stringify({
    hello: 'rocketmq-client-nodejs world',
```

```
    now: Date(),
  })),
});
console.log(receipt);
```

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
accessSecret	角色名称，在控制台的集群权限页面 SecretKey 列复制。
endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
topic	Topic 的名称，在控制台 Topic 管理页面复制。 

步骤3. 消费消息

Node.js SDK 目前支持 Simple Consumer。以下代码示例以 Simple Consumer 为例：

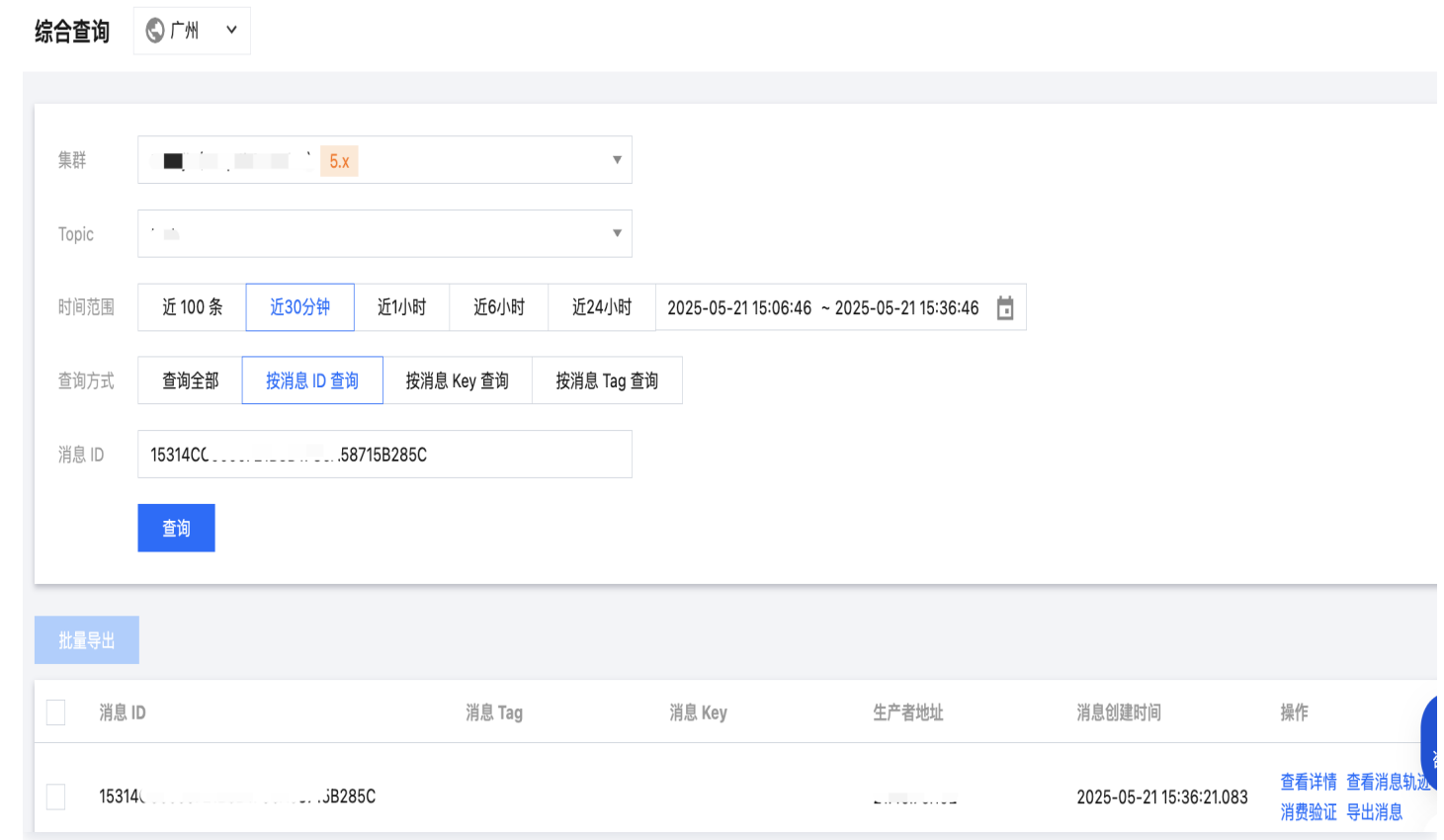
```
import {SimpleConsumer } from 'rocketmq-client-nodejs';
const simpleConsumer = new SimpleConsumer({
    endpoints: 'rmq-xxx.rocketmq.gz.qcloud.tencenttdmq.com:8080', // 腾讯
云接入点
  sessionCredentials: {
    accessKey: 'yourAccessKey', // ak
    accessSecret: 'yourSecretKey', // sk
  },
  consumerGroup: 'nodejs-demo-group', //消费组
  subscriptions: new Map().set('TopicTest', '*'),
});
await simpleConsumer.startup();

const messages = await simpleConsumer.receive(20);
console.log('got %d messages', messages.length);
for (const message of messages) {
  console.log(message);
  console.log('body=%o', message.body.toString());
  await simpleConsumer.ack(message);
}
```

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
accessSecret	角色名称，在控制台的集群权限页面 SecretKey 列复制。
endpoints	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。

consumerGroup	消费者组名称，在控制台 Group 管理页面复制。 
subscriptions	Topic 的名称，在控制台 Topic 管理页面复制。 

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询** > **综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。


综合查询
广州
集群
5.x
Topic
时间范围
近 100 条
近30分钟
近1小时
近6小时
近24小时
2025-05-21 15:06:46 ~ 2025-05-21 15:36:46
查询方式
查询全部
按消息 ID 查询
按消息 Key 查询
按消息 Tag 查询
消息 ID
15314CC.....58715B285C
查询
批量导出
消息 ID
消息 Tag
消息 Key
生产者地址
消息创建时间
操作
15314(.....)B285C
2025-05-21 15:36:21.083
查看详情
查看消息轨迹
消费验证
导出消息

4.x SDK
Spring Boot Starter 接入
发送与接收普通消息

最近更新时间：2025-06-11 11:56:12

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 或者 查看 Github项目

操作步骤

步骤1：添加依赖

在 pom.xml 中添加依赖，建议 2.3.3 最新版本，可以同时支持 Spring Boot 2 和 Spring Boot 3。

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.3.3</version>
     <exclusions>
             <exclusion>
                 <groupId>org.apache.rocketmq</groupId>
                 <artifactId>rocketmq-client</artifactId>
             </exclusion>
             <exclusion>
                 <groupId>org.apache.rocketmq</groupId>
                 <artifactId>rocketmq-acl</artifactId>
              </exclusion>
      </exclusions>
</dependency>
 <dependency>
            <groupId>org.apache.rocketmq</groupId>
```

```
            <artifactId>rocketmq-client</artifactId>
            <version>4.9.7</version>
        </dependency>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-acl</artifactId>
            <version>4.9.7</version>
 </dependency>
```

步骤2：准备配置

在配置文件中添加配置信息。

```
server:
  port: 8082

#rocketmq配置信息
rocketmq:
  # tdmq-rocketmq服务接入地址
  name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
  # 生产者配置
  producer:
    # 生产者组名
    group: group111
    # 角色密钥
    access-key: eyJrZXlJZC....
    # 已授权的角色名称
    secret-key: admin
  # 消费者公共配置
  consumer:
    # 角色密钥
    access-key: eyJrZXlJZC....
    # 已授权的角色名称
    secret-key: admin

  # 自定义配置，根据业务进行配置
  namespace: rocketmq-xxx|namespace1
  producer1:
    topic: testdev1
  consumer1:
    group: group111
    topic: testdev1
    subExpression: TAG1
```

```
consumer2:
  group: group222
  topic: testdev1
  subExpression: TAG2
```

参数	说明
name-server	集群接入地址，在控制台集群基本信息页面的接入信息模块获取。 
group	生产者的名称，用户自行定义，可以使用对应的 Topic 名字。
secret-key	角色名称，在控制台的集群权限页面 SecretKey 列复制。
access-key	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
namespace	命名空间的名称，在控制台命名空间页面复制。**如果您使用的是5.x集群或者 4.x 通用集群，此处填写集群 ID 即可。**
topic	Topic 的名称，在控制台 Topic 管理页面复制。

subExpression	用来设置消息的 TAG。

步骤3：发送消息

1. 在需要发送消息的类中注入 `RocketMQTemplate` 。

```
@Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
      private String topic;  // topic名称 （需要使用topic全称，所以在这里对
topic名称进行拼接）

      @Autowired
      private RocketMQTemplate rocketMQTemplate;
```

2. 发送消息，消息体可以是自定义对象，也可以是 Message 对象（org.springframework.messaging包中）。

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

3. 完整示例如下：

```
/**
* Description: 消息生产者
*/
@Service
public class SendMessage {
// 需要使用topic全称，所以进行topic名称的拼接，也可以自己设置  格式：namespace
全称%topic名称
 @Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
 private String topic;
     @Autowired
 private RocketMQTemplate rocketMQTemplate;
         /**
     * 同步发送
     *
     * @param message 消息内容
     * @param tags    订阅tags
```

```
     */
    public void syncSend(String message, String tags) {
            // springboot不支持使用header传递tags，根据要求，需要在topic后进行
拼接 formats: `topicName:tags`，不拼接标识无tag
            String destination = StringUtils.isBlank(tags) ? topic :
topic + ":" + tags;
            SendResult sendResult =
rocketMQTemplate.syncSend(destination,
                            MessageBuilder.withPayload(message)

.setHeader(MessageConst.PROPERTY_KEYS, "yourKey")    // 指定业务key
                                                .build());
            System.out.printf("syncSend1 to topic %s sendResult=%s %n",
topic, sendResult);
    }
}
```

> **说明**
> 该示例为同步发送。异步发送，单向发送等请参见 Demo 或者前往 GitHub 项目 。

步骤4：消费消息

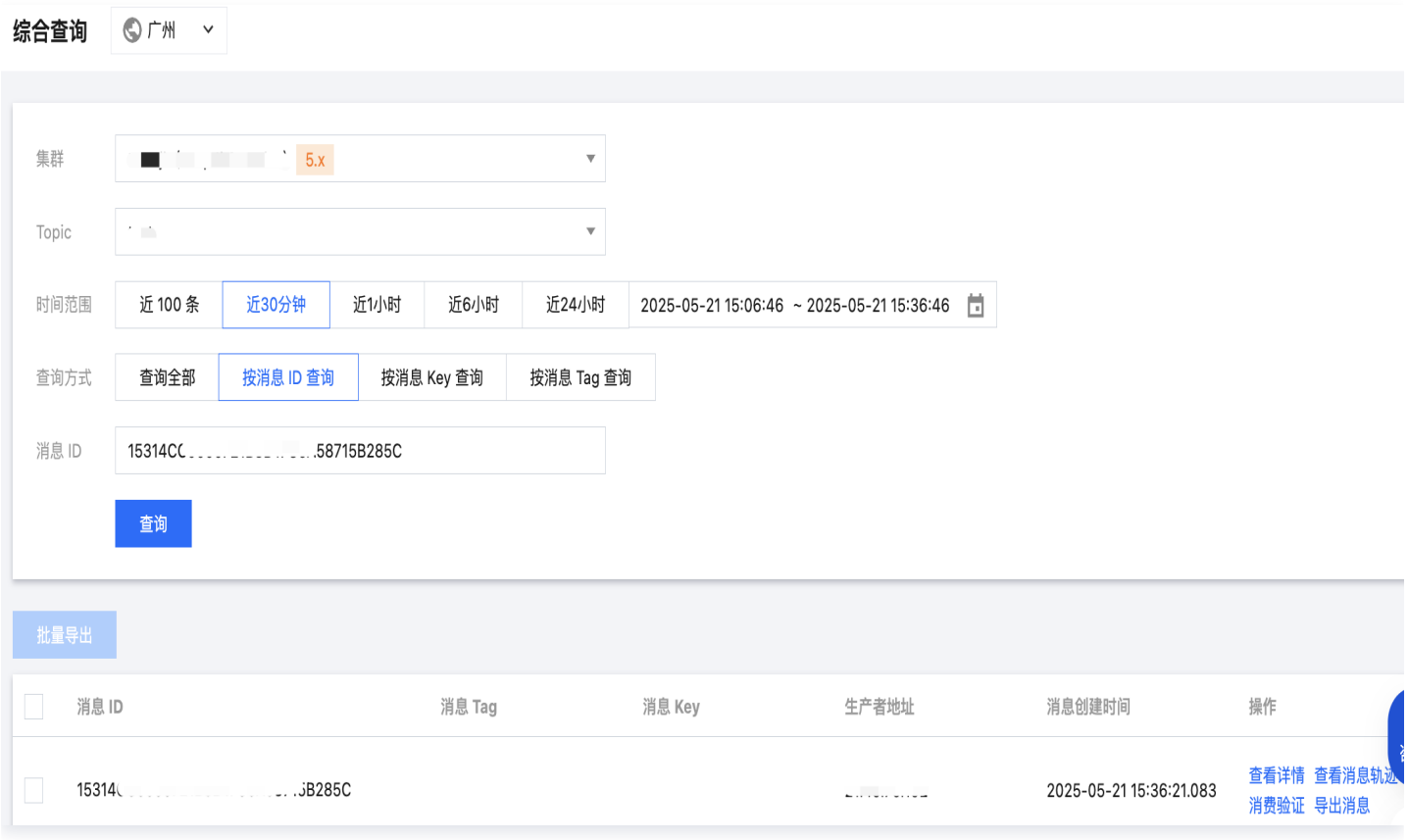
```
@Service
    @RocketMQMessageListener(
            consumerGroup =
"${rocketmq.namespace}%${rocketmq.consumer1.group}",   // 消费组，格式:
namespace全称%group名称
            // 需要使用topic全称，所以进行topic名称的拼接，也可以自己设置   格式:
namespace全称%topic名称
            topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
            selectorExpression = "${rocketmq.consumer1.subExpression}" //
订阅表达式，不配置表示订阅所有消息
    )
    public class MessageConsumer implements RocketMQListener<String> {

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

可根据业务需求配置多个消费者。消费者其他配置可根据具体业务需求进行配置。

> **说明：**
> 完整示例参见 下载 Demo 或者前往 GitHub 项目

步骤5：查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询** > **综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



发送与接收过滤消息

最近更新时间：2025-06-10 16:17:03

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过滤收发的完整过程。

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
- 已经了解 发送与接收普通消息
- 下载 Demo 或者 查看 Github项目

操作步骤

发送消息

和普通发送消息相同，发送时，只需要将 rocketMQTemplate 的发送 Topic 拼接上对应的 tag 即可。

```
 // springboot不支持使用header传递tags，根据要求，需要在topic后进行拼接
formats: topicName:tags，不拼接标识无tag
        String destination = StringUtils.isBlank(tags) ? topic : topic +
":" + tags;
        // object消息类型
        SendResult sendResult = rocketMQTemplate.syncSend(destination,
                MessageBuilder.withPayload(message)
                        .setHeader(MessageConst.PROPERTY_KEYS,
"yourKey")    // 指定业务key
                        .build());
        System.out.printf("syncSend1 to topic %s sendResult=%s %n",
topic, sendResult);
```

例如 topic 是 TopicTest，tag 是 TAG1, 那么调用 rocketMQTemplate 方法的第一个参数将是 TopicTest:TAG1。

消费消息

设置 selectorExpression 字段为对应的过滤 tag 即可。如下代码中，将 rocketmq.consumer1.subExpression 设置为 TAG1 就可以消费 TAG1 的消息。

```
@Service
@RocketMQMessageListener(
        consumerGroup =
"${rocketmq.namespace}%${rocketmq.consumer1.group}",  // 消费组，格式：
namespace全称%group名称
        // 需要使用topic全称，所以进行topic名称的拼接，也可以自己设置  格式：
namespace全称%topic名称
        topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
        selectorExpression = "${rocketmq.consumer1.subExpression}" // 订
阅表达式，不配置表示订阅所有消息
)
public class Tag1Consumer implements RocketMQListener<String> {

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

发送与接收延迟消息

最近更新时间：2025-06-10 16:17:03

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，帮助您更好地理解延迟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
- 已经了解 发送与接收普通消息
- 下载 Demo 或者 查看 Github项目

操作步骤

发送消息

和普通发送消息相同，在调用发送方法的时候，传递对应的延迟级别即可。

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

延迟等级和具体延迟时间的对应关系如下表：

延迟等级	时间
1	1s
2	5s
3	10s
4	30s
5	1m
6	2m

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

消费消息

和普通消息相同，无需特殊处理。

Spring Cloud Stream 接入

最近更新时间：2025-06-10 16:17:03

操作场景

本文以调用 Spring Cloud Stream 接入为例介绍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 或者前往 GitHub 项目

操作步骤

步骤1：引入依赖

在 pom.xml 中引入 spring-cloud-starter-stream-rocketmq 相关依赖。当前建议版本 2021.0.4.0。

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

步骤2：添加配置

在配置文件中增加 RocketMQ 相关配置。

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          # 服务地址全称
          name-server: rocketmq-xxx.rocketmq.ap-
bj.public.tencenttdmq.com:9876
          # 角色名称
```

```
        secret-key: admin
        # 角色密钥
        access-key: eyJrZXlJZ...
        # namespace全称
        namespace: rocketmq-xxx|namespace1
        # producer group
        group: producerGroup
      bindings:
        # channel名称，与spring.cloud.stream.bindings下的channel名称对应
        Topic-TAG1-Input:
          consumer:
            # 订阅的tag类型，根据消费者实际情况进行配置（默认是订阅所有消息）
            subscription: TAG1
        # channel名称
        Topic-TAG2-Input:
          consumer:
            subscription: TAG2
    bindings:
      # channel名称
      Topic-send-Output:
        # 指定topic，对应创建的topic名称
        destination: TopicTest
        content-type: application/json
      # channel名称
      Topic-TAG1-Input:
        destination: TopicTest
        content-type: application/json
        group: consumer-group1
      # channel名称
      Topic-TAG2-Input:
        destination: TopicTest
        content-type: application/json
        group: consumer-group2
```

⚠ 注意

1. 目前只有 `2.2.5-RocketMQ-RC1` 与 `2.2.5.RocketMQ.RC2` 及以上版本支持 namespace 配置，如使用别的版本需要对 topic 和 group 名称进行拼接。
   - 格式如下：
     - rocketmq-pngrpmk94d5o|stream%topic（格式为：namespace全称%topic名称）

- rocketmq-pngrpmk94d5o|stream%group（格式为：namespace全称%group名称）
- 新的虚拟集群与专享集群格式如下：
  - MQ_INST_rocketmqpj79obd2ew7v_test%topic （格式为：namespace全称%topic名称）
  - MQ_INST_rocketmqpj79obd2ew7v_test%group（格式为：namespace全称%group名称）

2. 配置方面 `2.2.5-RocketMQ-RC1` 与 `2.2.5.RocketMQ.RC2` 的订阅配置项为 `subscription`，其他低版本订阅配置项为 `tags`。

其他版本完整配置项参考如下：

```
spring:
     cloud:
       stream:
         rocketmq:
           bindings:
             # channel名称，与spring.cloud.stream.bindings下的channel名称对
应
             Topic-test1:
               consumer:
                 # 订阅的tag类型，根据消费者实际情况进行配置（默认是订阅所有消息）
                 tags: TAG1
             # channel名称
             Topic-test2:
               consumer:
                 tags: TAG2
           binder:
             # 服务地址全称
             name-server: rocketmq-xxx.rocketmq.ap-
bj.public.tencenttdmq.com:9876
             # 角色名称
             secret-key: admin
             # 角色密钥
             access-key: eyJrZXlJZ...
         bindings:
           # channel名称
           Topic-send:
             # 指定topic，对应创建的topic全称，格式为：集群id|namespace名
称%topic名称
             destination: rocketmq-xxx|stream%topic1
```

```
      content-type: application/json
      # 要使用group全称称，格式为：集群id|namespace名称%group名称
      group: rocketmq-xxx|stream%group1
    # channel名称
    Topic-test1:
      destination: rocketmq-xxx|stream%topic1
      content-type: application/json
      group: rocketmq-xxx|stream%group1
    # channel名称
    Topic-test2:
      destination: rocketmq-xxx|stream%topic1
      content-type: application/json
      group: rocketmq-xxx|stream%group2
```

参数	说明
name_server	集群接入地址，在控制台集群管理页面的集群列表操作栏的接入地址处获取。新版共享集群与专享集群命名列表获取。
secret_key	角色名称，在控制台的集群权限页面 SecretKey 列复制。
access	角色密钥，在控制台的集群权限页面 AccessKey 列复制。

s-key

namespace

命名空间的名称，在控制台命名空间页面复制。如果您使用的是4.x通用集群或者5.x集群，此处可填写集

group

生产者组名称，在控制台 Group 管理页面复制。

destination

Topic 的名称，在控制台 Topic 管理页面复制。

步骤3：配置 channel

channel 分为输入和输出，可根据自己的业务进行单独配置。

```
/**
 * 自定义通道 Binder
 */
public interface CustomChannelBinder {

    /**
     * 发送消息(消息生产者)
     * 绑定配置中的channel名称
     */
    @Output("Topic-send-Output")
    MessageChannel sendChannel();


    /**
     * 接收消息1(消费者1)
     * 绑定配置中的channel名称
     */
    @Input("Topic-TAG1-Input")
    MessageChannel testInputChannel1();

    /**
     * 接收消息2(消费者2)
     * 绑定配置中的channel名称
     */
    @Input("Topic-TAG2-Input")
    MessageChannel testInputChannel2();
}
```

步骤4：添加注解

在配置类或启动类上添加相应注解，如果有多个 binder 配置，都要在此注解中进行指定。

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

步骤5：发送消息

1. 在要发送消息的类中，注入 `CustomChannelBinder` 。

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

2. 发送消息，调用对应的输出流 channel 进行消息发送。

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

步骤6：消费消息

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

    /**
     * 监听channel（配置中的channel 名称）
     *
     * @param messageBody 消息内容
     */
    @StreamListener("Topic-TAG1-Input")
    public void receive(String messageBody) {
        logger.info("Receive1: 通过stream收到消息，messageBody = {}",
messageBody);
    }

    /**
     * 监听channel（配置中的channel 名称）
     *
     * @param messageBody 消息内容
     */
    @StreamListener("Topic-TAG2-Input")
    public void receive2(String messageBody) {
        logger.info("Receive2: 通过stream收到消息，messageBody = {}",
messageBody);
    }
}
```

步骤7：本地测试

本地启动项目之后，可以从控制台看到启动成功。

浏览器访问 `http://localhost:8080/test-simple` 可以看到发送成功。观察开发 IDE 的输出日志。

```
2023-02-23 19:19:00.441  INFO 21958 --- [nio-8080-exec-1]
c.t.d.s.controller.StreamController      : Send: 通过stream发送消息，
messageBody = GenericMessage [payload={"key":"value"}, headers=
{id=3f28bc70-da07-b966-a922-14a17642c9c4, timestamp=1677151140353}]
2023-02-23 19:19:01.138  INFO 21958 --- [nsumer-group1_1]
c.t.d.s.StreamDemoApplication            : Receive1: 通过stream收到消息，
messageBody = {"headers":{"id":"3f28bc70-da07-b966-a922-
14a17642c9c4","timestamp":1677151140353},"payload":{"key":"value"}}
```

可以看到。发送了一条 TAG1 的消息，同时也只有 TAG1 的订阅者收到了消息。

> **说明**
> 具体使用可参见 GitHub Demo 或 Spring cloud stream 官网 。

Java SDK
发送与接收普通消息

最近更新时间：2025-06-10 16:17:03

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 或者前往 GitHub 项目

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 maven 工程为例，在 pom.xml 添加以下依赖：

> **说明：**
> 依赖版本要求 ⩾ 4.9.4, 当前建议为4.9.5。

```
<!-- 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>
```

步骤2：生产消息

创建消息生产者

```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey)) // ACL权限
    );
    // 设置NameServer的地址
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();
```

参数	说明
groupName	生产者组名称，建议使用对应的 topic 名字
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secretK	角色名称，在控制台的集群权限页面 SecretKey 列复制。

ey	
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 

发送消息

发送消息有多种方式：同步发送、异步发送、单向发送等。

同步发送

```
for (int i = 0; i < 10; i++) {
            // 创建消息实例，设置topic和消息内容
            Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ
" + i).getBytes(RemotingHelper.DEFAULT_CHARSET));
            // 发送消息
            SendResult sendResult = producer.send(msg);
            System.out.printf("%s%n", sendResult);
        }
```

参数	说明
topic_	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。

name	
TAG	用来设置消息的 TAG。

异步发送

```
// 设置发送失败后不重试
        producer.setRetryTimesWhenSendAsyncFailed(0);
        // 设置发送消息的数量
        int messageCount = 10;
        final CountDownLatch countDownLatch = new
CountDownLatch(messageCount);
        for (int i = 0; i < messageCount; i++) {
            try {
                final int index = i;
                // 创建消息实体，设置topic和消息内容
                Message msg = new Message(topic_name, "TAG", ("Hello
rocketMq " + index).getBytes(RemotingHelper.DEFAULT_CHARSET));
                producer.send(msg, new SendCallback() {
                    @Override
                    public void onSuccess(SendResult sendResult) {
                        // 消息发送成功逻辑
                        countDownLatch.countDown();
                        System.out.printf("%-10d OK %s %n", index,
sendResult.getMsgId());
                    }

                    @Override
                    public void onException(Throwable e) {
                        // 消息发送失败逻辑
                        countDownLatch.countDown();
                        System.out.printf("%-10d Exception %s %n", index,
e);
                        e.printStackTrace();
                    }
                });
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        countDownLatch.await(5, TimeUnit.SECONDS);
```

参数	说明
topic_name	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。 
TAG	用来设置消息的 TAG。

单向发送

```
for (int i = 0; i < 10; i++) {
            // 创建消息实例，设置topic和消息内容
            Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ "
+ i).getBytes(RemotingHelper.DEFAULT_CHARSET));
            // 发送单向消息
            producer.sendOneway(msg);
        }
```

参数	说明
topic_name	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。

name	
TAG	用来设置消息的 TAG。

说明：
批量发送及其他情况可参见 GitHub Demo 或 RocketMQ 官方文档 。

步骤3：消费消息

创建消费者

TDMQ RocketMQ 版支持 Push 和 pull 两种消费模式。推荐 Push 消费模式。

```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey))); //ACL权限
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

参数	说明
groupName	Group 的名称，在控制台 Group 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%group名称` ，例如 MQ_INSTxxx_aaa%GroupTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Group 名称即可。

nameserver

集群接入地址，控制台集群基本信息页面的接入信息模块获取。

secretKey

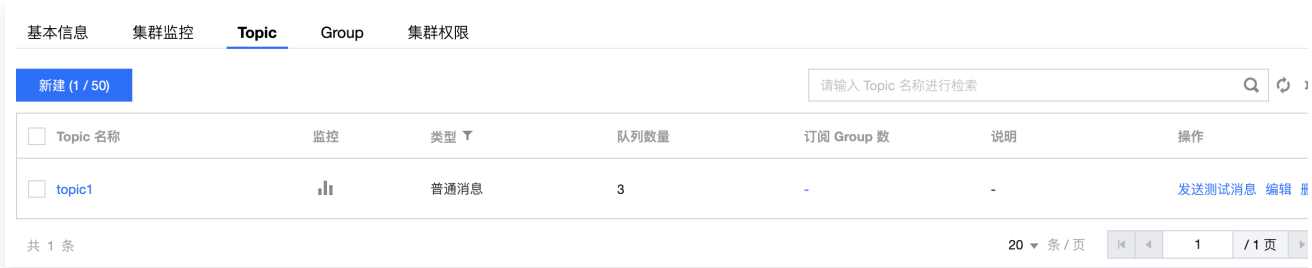
角色名称，在控制台的集群权限页面 SecretKey 列复制。

参数	说明
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 

订阅消息

根据消费模式不同，订阅方式也有所区别。

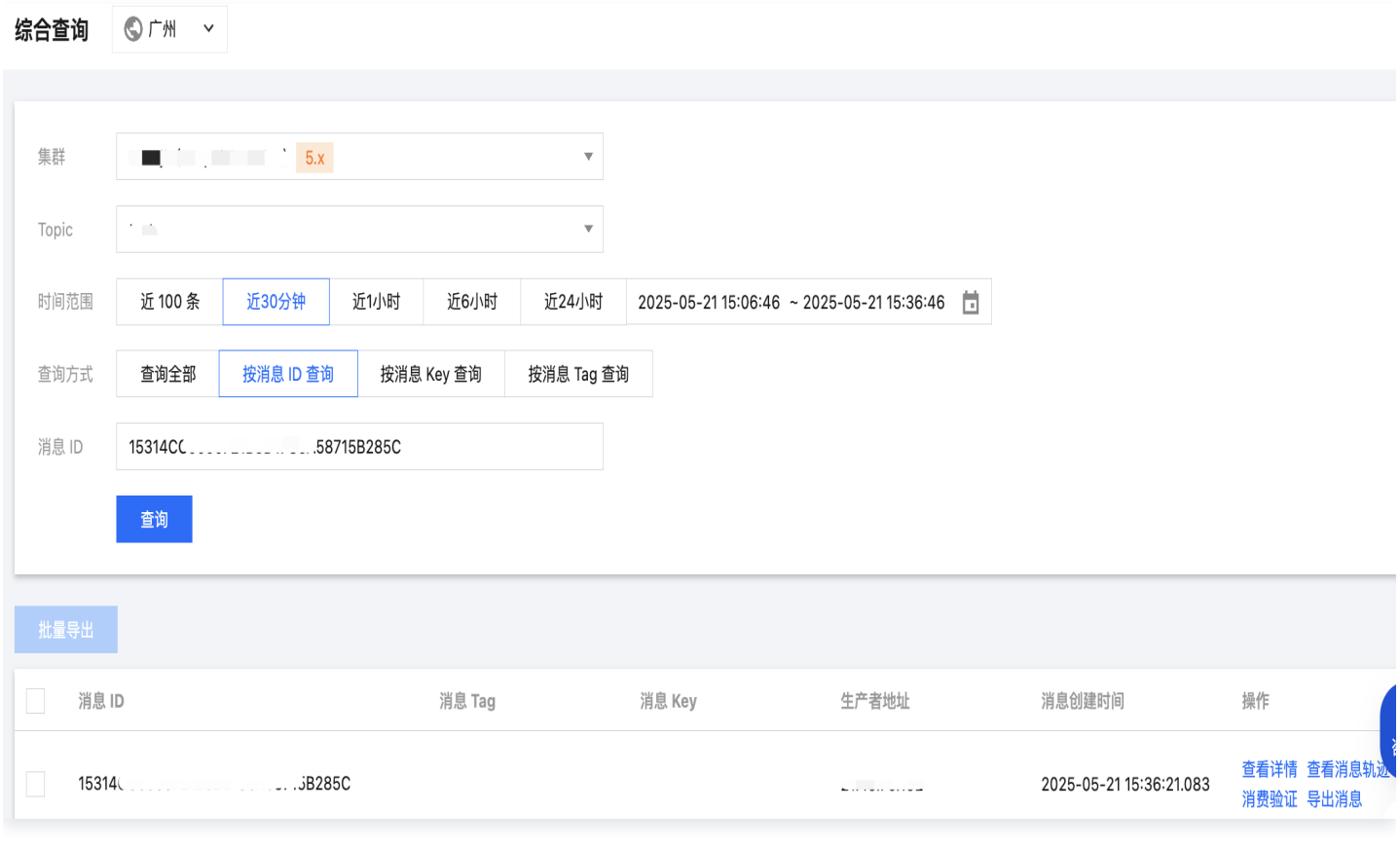
```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        // 启动消费者实例
        pushConsumer.start();
```

参数	说明
topic_name	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。 

"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询** > **综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



说明：

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

发送与接收延迟消息

最近更新时间：2025-06-10 16:17:03

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现定时消息收发的操作过程。

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 或者前往 GitHub 项目

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 maven 工程为例，在 pom.xml 添加以下依赖：

> **说明：**
> 依赖版本要求 ⩾ 4.9.4, 当前建议为4.9.5。

```
<!-- 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>
```

步骤2：生产消息

创建消息生产者

```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey)) // ACL权限
    );
    // 设置NameServer的地址
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupName	生产者组名称，建议使用对应的 topic 名字
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secre	角色名称，在控制台的集群权限页面 SecretKey 列复制。

tKey	
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 

发送消息

固定延迟级别的消息

```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message
" + i).getBytes());
    // 设置消息延迟等级
    message.setDelayTimeLevel(5);
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

任意延迟时间的消息

```
int totalMessagesToSend = 1;
for (int i = 0; i < totalMessagesToSend; i++) {
```

```
    Message message = new Message(TOPIC_NAME, ("Hello timer message " +
i).getBytes());
    // 设置发送消息的时间
    long timeStamp = System.currentTimeMillis() + 30000;
    // 若需要发送定时消息，则需要设置定时时间，消息将在指定时间进行投递，例如消息将在
2022-08-08 08:08:08投递。
    // 若设置的时间戳在当前时间之前，则消息将被立即投递给Consumer。
    //将 __STARTDELIVERTIME 设定到 msg 的属性中
    message.putUserProperty("__STARTDELIVERTIME",
String.valueOf(timeStamp));
    // 发送消息
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

步骤3：消费消息

创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式。

```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey))); //ACL权限
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

> **说明：**
> 以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupNa	Group 的名称，在控制台 Group 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%group名称` ，例如MQ_INSTxxx_aaa%GroupTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Group 名称即可。

me

accessKey

角色密钥，在控制台的集群权限页面 AccessKey 列复制。

secretKey

角色名称，在控制台的集群权限页面 SecretKey 列复制。

nameserver

集群接入地址，控制台集群基本信息页面的接入信息模块获取。

订阅消息

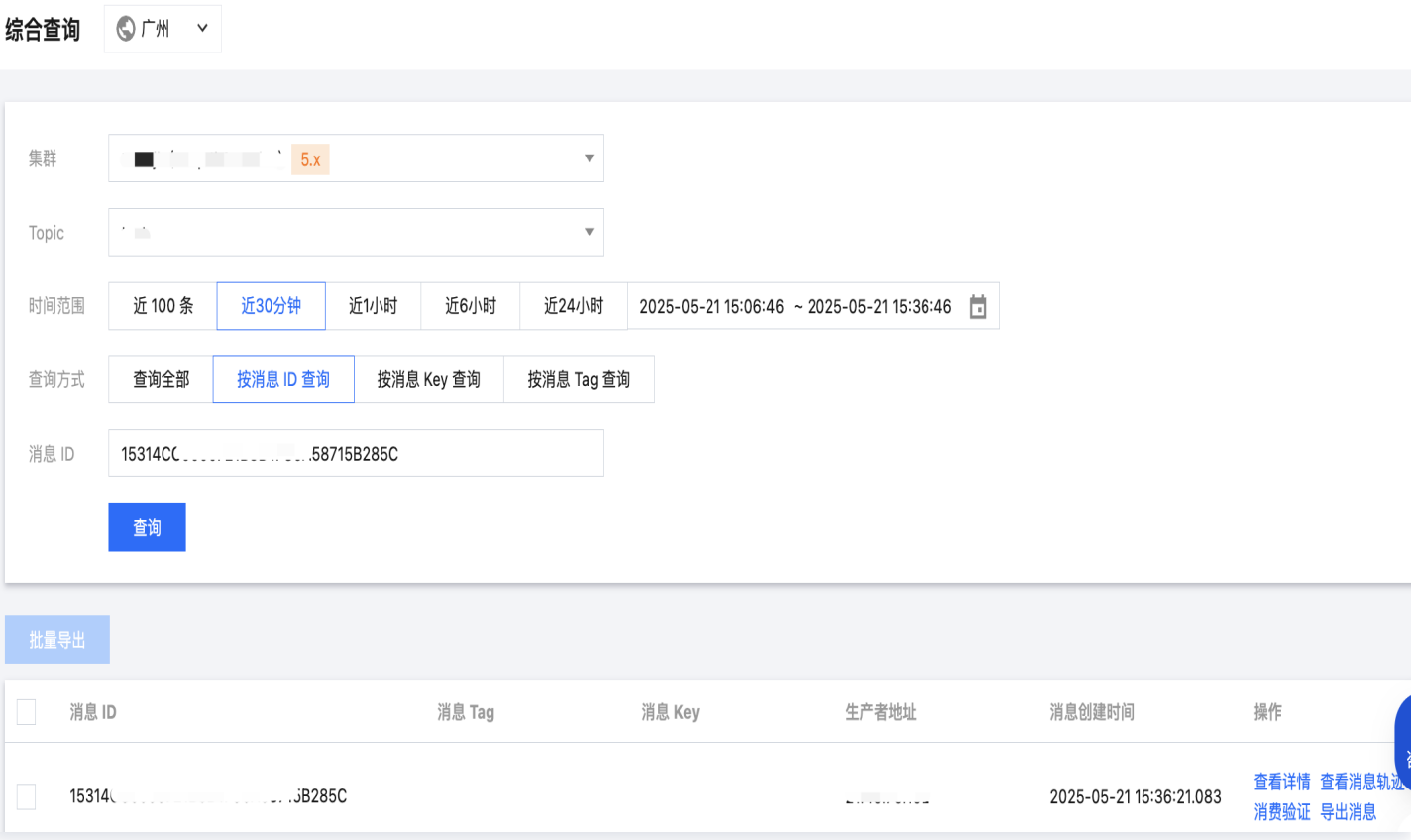
根据消费模式不同，订阅方式也有所区别。

```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        // 启动消费者实例
        pushConsumer.start();
```

参数	说明
topic_name	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。 
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询 > 综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



说明：

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

发送与接收顺序消息

最近更新时间：2025-06-10 16:17:03

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现定时消息收发的操作过程。

前提条件

- 完成资源创建与准备（如果是全局顺序消息，需要创建单队列topic）
- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo 或者前往 GitHub 项目

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 maven 工程为例，在 pom.xml 添加以下依赖：

> **说明：**
> 依赖版本要求 ⩾ 4.9.4, 当前建议为4.9.5。

```
<!-- 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>
```

步骤2：生产消息

创建消息生产者

```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey)) // ACL权限
    );
    // 设置NameServer的地址
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupName	生产者组名称，建议使用对应的 topic 名字。
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secret	角色名称，在控制台的集群权限页面 SecretKey 列复制。

Key	
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 

发送消息

全局顺序消息

发送代码和简单的消息没有区别

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

分区有序的消息

```
for (int i = 0; i < 3; i++) {
    int orderId = i % 3;
    // 构造消息示例
    Message msg = new Message(TOPIC_NAME, "your tag", "KEY" + i,("Hello
RocketMQ " + i).getBytes(StandardCharsets.UTF_8));
```

```
    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);
}
```

步骤3：消费消息

创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式。

```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey))); //ACL权限
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

> **说明：**
> 以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupName	Group 的名称，在控制台 Group 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%group名称` ，例如MQ_INSTxxx_aaa%GroupTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Group 名称即可。

nameserver

集群接入地址，控制台集群基本信息页面的接入信息模块获取。

secretKey

角色名称，在控制台的集群权限页面 SecretKey 列复制。

accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。

订阅消息

根据消费模式不同，订阅方式也有所区别。

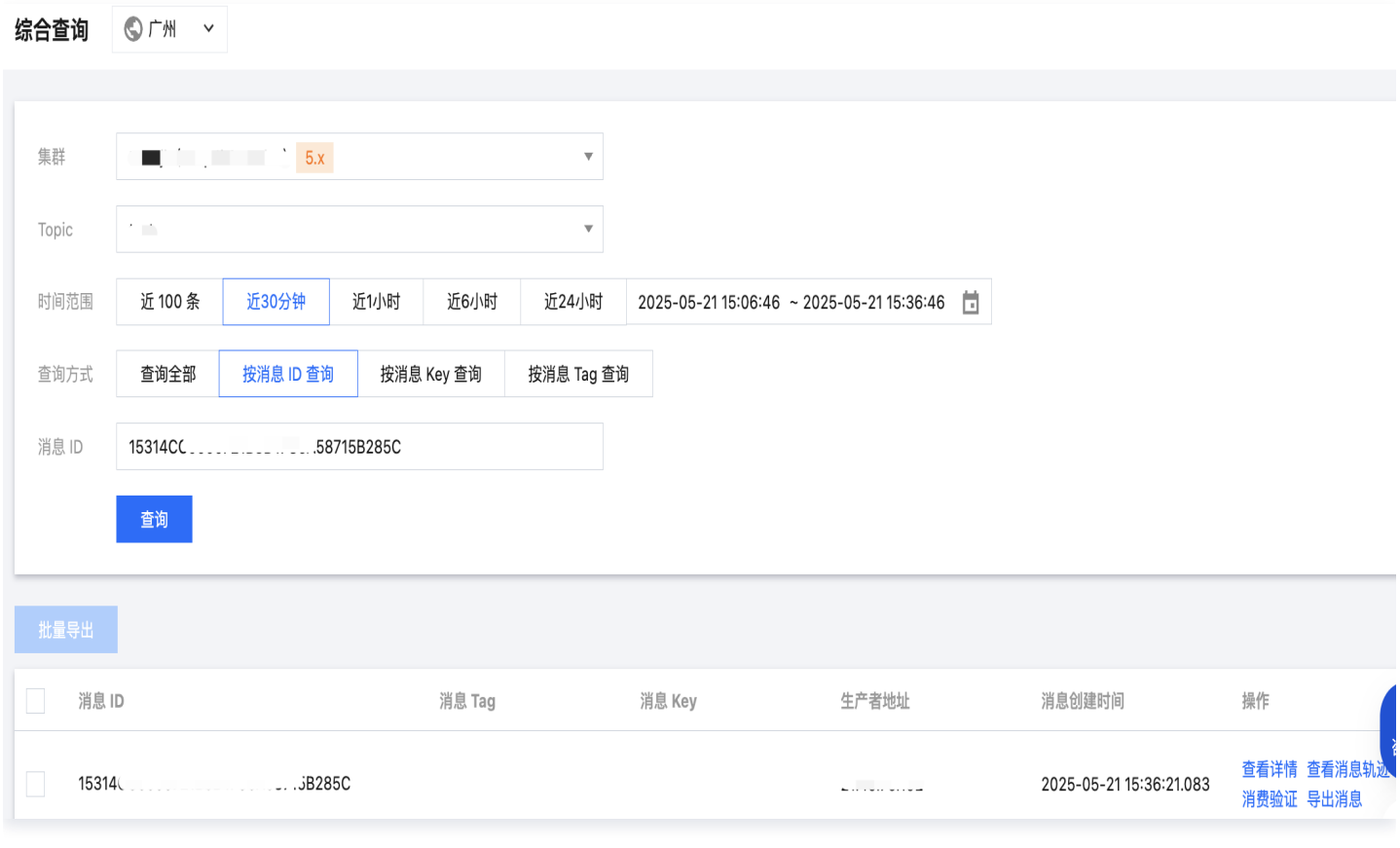
```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        // 启动消费者实例
        pushConsumer.start();
```

参数	说明
topic_name	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。

"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询** > **综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



说明：

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

发送与接收事务消息

最近更新时间：2025-06-10 16:17:03

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现事务消息收发的操作过程。

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 或者前往 GitHub 项目

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 maven 工程为例，在 pom.xml 添加以下依赖：

> **说明：**
> 依赖版本要求 ⩾ 4.9.4, 当前建议为4.9.5。

```
<!-- 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>
```

步骤2：生产消息

实现 TransactionListener

```
public class TransactionListenerImpl implements TransactionListener {

    //半消息发送成功后，回调该方法执行本地事务
    @Override
    public LocalTransactionState executeLocalTransaction(Message msg,
Object arg) {
      //这里执行数据库事务，如果成功，就返回成功，否则返回未知，或者回滚，等待回查
      return LocalTransactionState.UNKNOW;
    }
    //回查本地事务
    @Override
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {
       //这里查询本地db的数据状态，然后决定是否是否提交
       return LocalTransactionState.COMMIT_MESSAGE;
    }

}
```

创建消息生产者

```
//需要用户实现一个TransactionListener 实例，
TransactionListener transactionListener = new TransactionListenerImpl();
// 实例化事务消息生产者
ProducerTransactionMQProducer producer = new
TransactionMQProducer("transaction_group",
// ACL权限
new AclClientRPCHook(new SessionCredentials(ClientCreater.ACCESS_KEY,
ClientCreater.SECRET_KEY)));
// 设置NameServer的地址
producer.setNamesrvAddr(ClientCreater.NAMESERVER);
producer.setTransactionListener(transactionListener);
producer.start();
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
gr	生产者组名称，建议使用对应的 topic 名字

oupName	
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 

发送消息

```
for (int i = 0; i < 3; i++) {
    // 构造消息示例
    Message msg = new Message(TOPIC_NAME, "your tag", "KEY" + i,("Hello
RocketMQ " + i).getBytes(StandardCharsets.UTF_8));
    SendResult sendResult = producer.sendMessageInTransaction(msg,null);
     System.out.printf("%s%n", sendResult);
}
```

步骤3：消费消息

创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式。

```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey))); //ACL权限
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);

pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
            // 消息处理逻辑
            System.out.printf("%s Receive transaction messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            });
```

> **说明：**
> 以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
gro	Group 的名称，在控制台 Group 管理页面复制。

upName	• 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%group名称` ，例如 MQ_INSTxxx_aaa%GroupTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Group 名称即可。 
namesever	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
acce	角色密钥，在控制台的集群权限页面 AccessKey 列复制。

sSKey

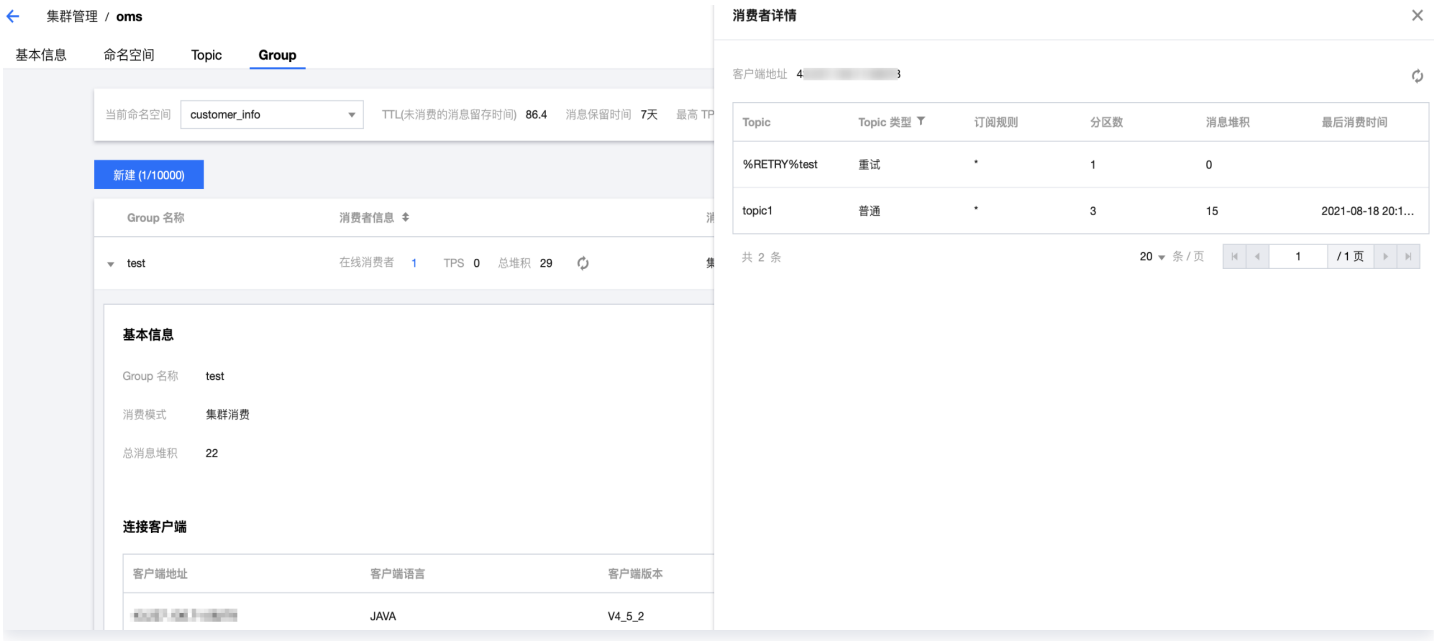
订阅消息

根据消费模式不同，订阅方式也有所区别。

```
// 订阅topic
        pushConsumer.subscribe(topic_name, "*");
        // 注册回调实现类来处理从broker拉取回来的消息
        pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        // 启动消费者实例
        pushConsumer.start();
```

步骤4：查看消费详情

登录 RocketMQ 控制台，在**集群管理 > Group** 页面，可查看与 Group 连接的客户端列表，单击操作列的**查看详情**，可查看消费者详情。



说明：

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

发送与接收过滤消息

最近更新时间：2025-06-10 16:17:03

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现过滤消息收发的操作过程。目前支持两种，分别是Tag 和 SQL 方式。

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 或者前往 GitHub 项目
- 参考普通消息的发送和接收

TAG使用步骤

创建生产者消费者的基础代码不再赘述，可以参考普通消息的发送和接收即可。

- 对于生产消息，主要是在构造消息体的时候，带上TAG。
- 对于消费消息，主要是订阅的时候，带上单个TAG 或者 * 或者多个TAG表达式。

步骤1：生产消息

发送消息

发送代码和简单的消息没有区别 主要是在构造消息体的时候，带上TAG，仅允许一个。

```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME, "Tag1", "Hello
RocketMQ.".getBytes(StandardCharsets.UTF_8));
    // 发送消息
    SendResult sendResult = producer.send(msg);
    System.out.println("sendResult = " + sendResult);
}
```

步骤2：消费消息

订阅消息。

```
// 订阅topic 订阅所有的TAG
pushConsumer.subscribe(topic_name, "*");

//订阅指定的TAG
//pushConsumer.subscribe(TOPIC_NAME, "Tag1");

//订阅多个TAG
//pushConsumer.subscribe(TOPIC_NAME, "Tag1||Tag2");

// 注册回调实现类来处理从broker拉取回来的消息
pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
            // 消息处理逻辑
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// 启动消费者实例
pushConsumer.start();
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
topic_name	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。 
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

> **说明**
> 上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

SQL 使用步骤

创建生产者消费者的基础代码不再赘述，可以参考普通消息的发送和接收即可。

- 对于生产消息，主要是在构造消息体的时候，带上用户自定义的properties。
- 对于消费消息，主要是订阅的时候，带上对应的SQL表达式。

步骤1：生产消息

发送代码和简单的消息没有区别 主要是在构造消息体的时候，带上自定义属性，允许多个。

```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME,"Hello
RocketMQ.".getBytes(StandardCharsets.UTF_8));
    msg.putUserProperty("key1","value1");
    // 发送消息
    SendResult sendResult = producer.send(msg);
    System.out.println("sendResult = " + sendResult);
}
```

步骤2：消费消息

对于消费消息，主要是订阅的时候，带上对应的SQL表达式，其他的和普通的没有区别。

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

// 订阅topic 订阅单个key的sql，最常用
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS
NOT NULL AND key1='value1'"));

//订阅多个属性
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS
NOT NULL AND key2 IS NOT NULL  AND key1='value1' AND key2='value2'"));

// 注册回调实现类来处理从broker拉取回来的消息
pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
          // 消息处理逻辑
```

```
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
            // 标记该消息已经被成功消费，根据消费情况，返回处理状态
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// 启动消费者实例
pushConsumer.start();
```

说明

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

发送与接收广播消息

最近更新时间：2025-06-10 16:17:03

操作场景

本文以调用 Java SDK 为例介绍通过开源 SDK 实现广播消息收发的操作过程。

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 或者前往 GitHub 项目

操作步骤

步骤1：安装 Java 依赖库

在 Java 项目中引入相关依赖，以 maven 工程为例，在 pom.xml 添加以下依赖：

> **说明：**
> 依赖版本要求 ⩾ 4.9.4, 当前建议为4.9.5。

```
<!-- 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>
```

步骤2：生产消息

创建消息生产者

```
// 实例化消息生产者Producer
    DefaultMQProducer producer = new DefaultMQProducer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey)) // ACL权限
    );
    // 设置NameServer的地址
    producer.setNamesrvAddr(nameserver);
    // 启动Producer实例
    producer.start();
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupName	生产者组名称，建议使用对应的 topic 名字
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
secret	角色名称，在控制台的集群权限页面 SecretKey 列复制。

Key	
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 

发送消息

和普通的消息没有区别，广播消息主要表达的是消费方的行为。

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

步骤3：消费消息

创建消费者

TDMQ RocketMQ 版支持 push 和 pull 两种消费模式。推荐Push消费模式。

```
// 实例化消费者
        DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
```

```
            new AclClientRPCHook(new SessionCredentials(accessKey,
secretKey))); //ACL权限
        // 设置NameServer的地址
        pushConsumer.setNamesrvAddr(nameserver);
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupName	Group 的名称，在控制台 Group 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%group名称` ，例如 MQ_INSTxxx_aaa%GroupTest。 • 4.x通用集群此处无需拼接，填写 Group 名称即可。 
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 

secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 

订阅消息

主要增加消费模式的设置。

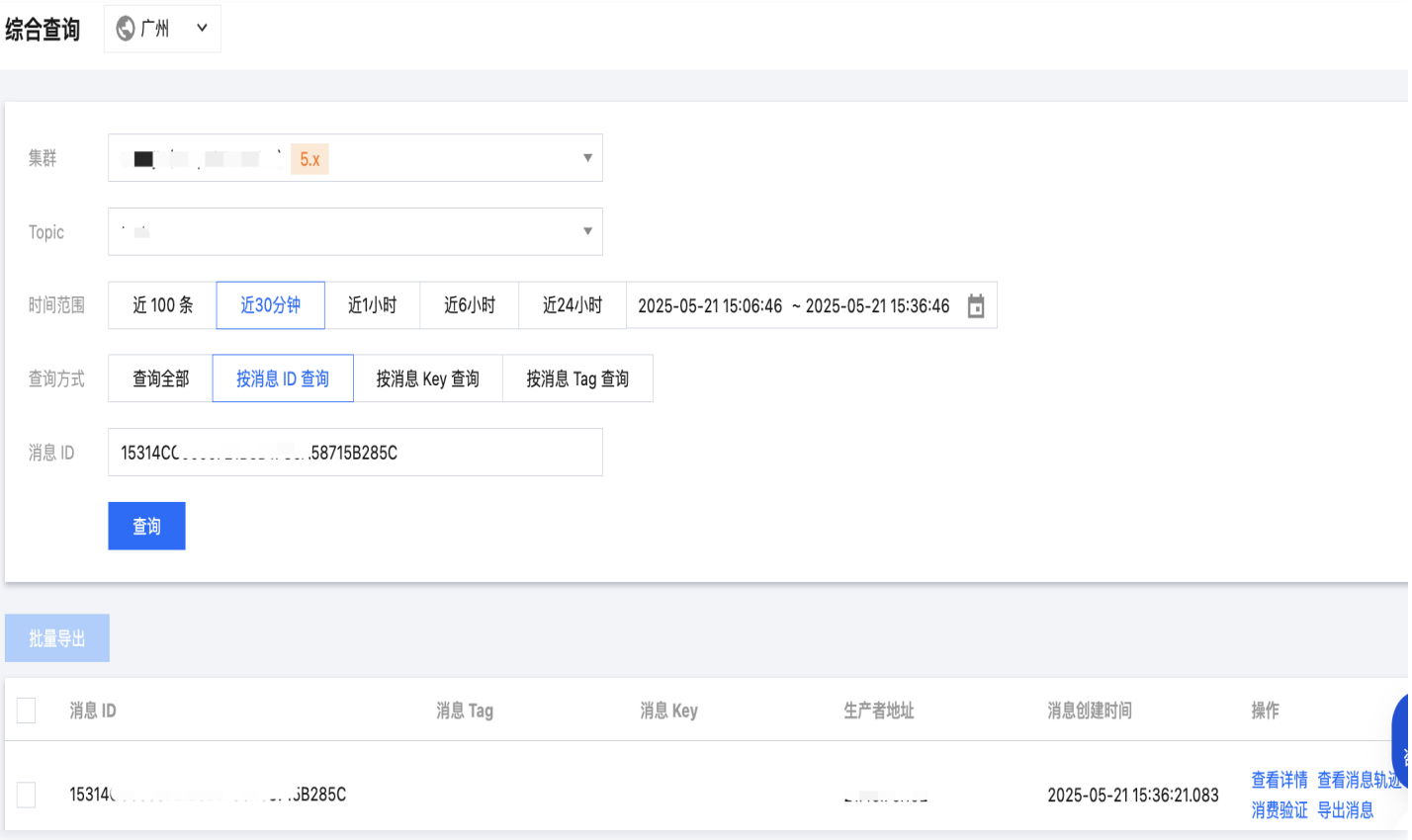
```
//设置广播消费模式
pushConsumer.setMessageModel(MessageModel.BROADCASTING);
// 订阅topic
pushConsumer.subscribe(topic_name, "*");
// 注册回调实现类来处理从broker拉取回来的消息
pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
    // 消息处理逻辑
    System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
    // 标记该消息已经被成功消费，根据消费情况，返回处理状态
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// 启动消费者实例
pushConsumer.start();
```

参数	说明

topic_name	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群：此处无需拼接，填写 Topic 名称即可。 
"*"	订阅表达式如果为 null 或*表达式表示订阅全部，同时支持 "tag1 || tag2 || tag3" 标识订阅多个类型的 tag。

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询 > 综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



说明：

> 上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

C++ SDK

最近更新时间：2025-06-10 16:17:03

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
- 安装 GCC
- 下载 Demo

操作步骤

1. 准备环境。
   1.1 需要在客户端环境安装 RocketMQ-Client-CPP 库，根据官方文档进行安装即可 安装 CPP 动态库，**推荐使用 master 分支构建。**
   1.2 在项目中引入 RocketMQ-Client-CPP 相关头文件及动态库。
2. 初始化消息生产者。

```
// 设置生产组名称
DefaultMQProducer producer(groupName);
// 设置服务接入地址
producer.setNamesrvAddr(nameserver);
// 设置用户权限
producer.setSessionCredentials(
    accessKey,  // 角色密钥
    secretKey, // 角色名称
    "");
// 设置命名空间（命名空间全称）
producer.setNameSpace(namespace);
// 请确保参数设置完成在启动之前
producer.start();
```

> **说明：**
> 以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupName	生产者组名称，在控制台 Group 管理页面复制。 
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
access	角色密钥，在控制台的集群权限页面 AccessKey 列复制。

sKey	
namespace	命名空间的名称，在控制台命名空间页面复制。如果您使用的是4.x通用集群或者5.x集群，此处可填写 

3. 发送消息。

```
// 初始化消息内容
    MQMessage msg(
        topicName,  // topic名称
        TAGS,       // 消息tag
        KEYS,       // 消息业务key
        "Hello cpp client, this is a message."  // 消息内容
    );

    try {
        // 发送消息
        SendResult sendResult = producer.send(msg);
        std::cout << "SendResult:" << sendResult.getSendStatus() << ",
Message ID: " << sendResult.getMsgId()
            << std::endl;
    } catch (MQException e) {
        std::cout << "ErrorCode: " << e.GetError() << " Exception:" <<
e.what() << std::endl;
    }
```

参数	说明
topicName	Topic 的名称，在控制台 topic 页面复制。

TAGS	用来设置消息的 TAG。
KEYS	设置消息业务 key。

4. 资源释放。

```
// 释放资源
    producer.shutdown();
```

5. 初始化消费者。

```
// 消息监听
    class ExampleMessageListener : public MessageListenerConcurrently {
    public:
        ConsumeStatus consumeMessage(const std::vector<MQMessageExt>
&msgs) {
            for (auto item = msgs.begin(); item != msgs.end(); item++)
{
                // 业务
                std::cout << "Received Message Topic:" << item-
>getTopic() << ", MsgId:" << item->getMsgId() << ", TAGS:"
                          << item->getTags() << ", KEYS:" << item-
>getKeys() << ", Body:" << item->getBody() << std::endl;
            }
            // 消费成功返回CONSUME_SUCCESS
            return CONSUME_SUCCESS;
            // 消费失败返回RECONSUME_LATER，该消息将会被重新消费
            // return RECONSUME_LATER;
        }
    };

    // 初始化消费者
    DefaultMQPushConsumer *consumer = new
DefaultMQPushConsumer(groupName);
    // 设置服务地址
    consumer->setNamesrvAddr(nameserver);
    // 设置用户权限
    consumer->setSessionCredentials(
        accessKey,
        secretKey,
        "");
    // 设置命名空间
```

```
    consumer->setNameSpace(namespace);
    // 设置实例名称
    consumer->setInstanceName("CppClient");

    // 请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
    ExampleMessageListener *messageListener = new
ExampleMessageListener();
    // 订阅消息
    consumer->subscribe(topicName, TAGS);
    // 设置消息监听
    consumer->registerMessageListener(messageListener);

    // 准备工作完成，必须调用启动函数，才可以正常工作。
    consumer->start();
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupName	消费组名称，在控制台 Group 管理页面复制。 
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。

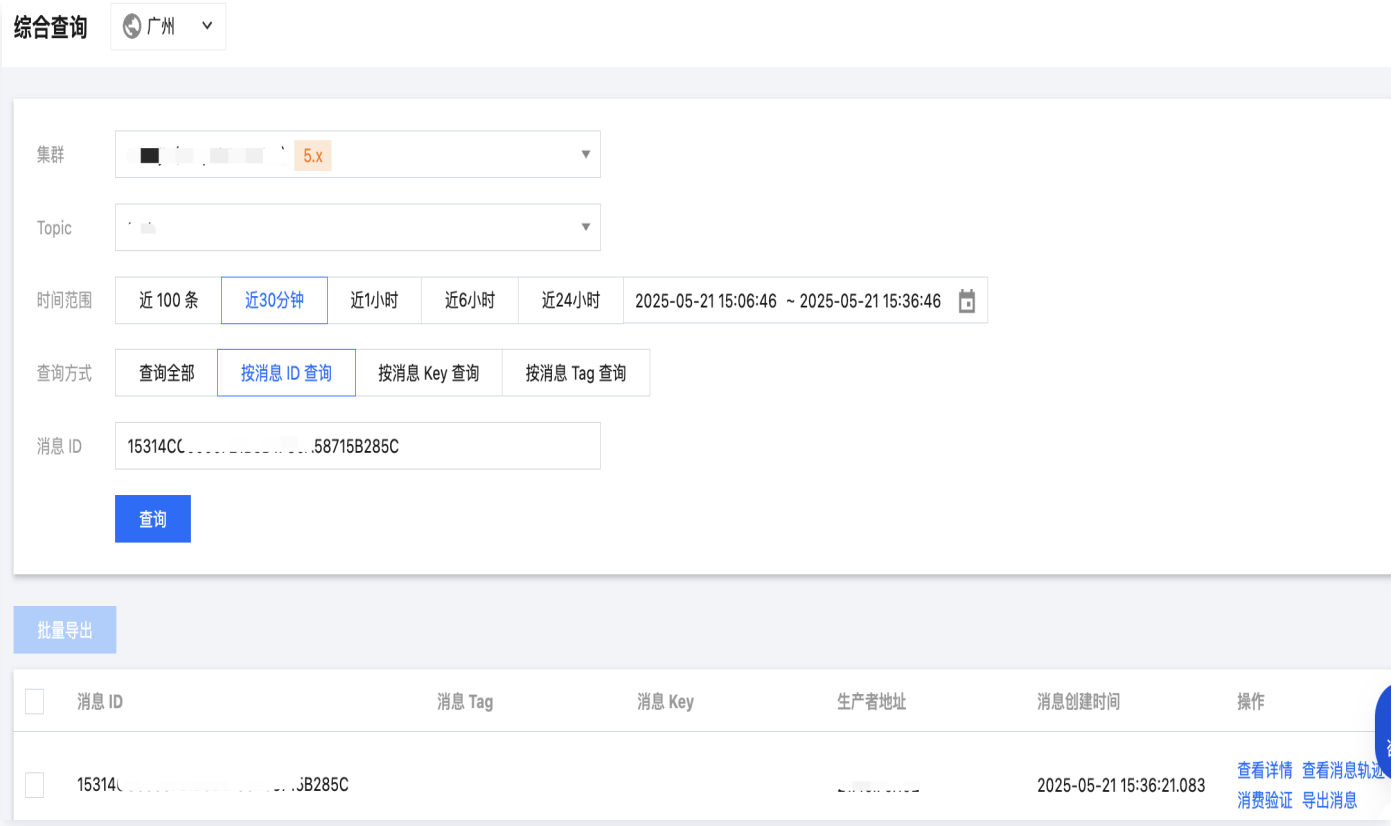
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
namespace	命名空间的名称，在控制台命名空间页面复制。

	
topicName	Topic 的名称，在控制台 topic 页面复制。
TAGS	用来设置订阅消息的 TAG。

6. 资源释放。

```
// 资源释放
consumer->shutdown();
```

7. 查看消费详情。发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询 > 综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



如果需要发送任意延迟消息，设置__STARTDELIVERTIME 属性即可。

```
        MQMessage msg("控制台topic", "tag", "Delay Message.");
        chrono::system_clock::duration d =
chrono::system_clock::now().time_since_epoch();
        chrono::milliseconds mil =
chrono::duration_cast<chrono::milliseconds>(d);
        //定时延时消息，单位毫秒（ms），在指定时间戳（当前时间之后）进行投递
        //设置需要延时或者定时的时间，例如当前时间延迟30秒后投递。
        long expectTime = mil.count() + 30000;
        msg.setProperty("__STARTDELIVERTIME", to_string(expectTime));
```

说明：

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 或 RocketMQ 官方文档。

Go SDK

最近更新时间：2025-06-24 11:59:02

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. 在客户端环境执行如下命令下载 RocketMQ 客户端相关的依赖包。请确保下载的版本高于 v2.1.2.rc2。

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

2. 在对应的方法内创建生产者，如您需要发送普通消息，则在 `syncSendMessage.go` 文件内修改对应的参数。延时消息目前支持任意精度的延时，且不受 delay level 的影响。

普通消息

```
// 服务接入地址 （注意：需要在接入地址前面加上 http:// 或 https:// 否则无法解析）
var serverAddress = "https://rocketmq-xxx.rocketmq.ap-
bj.public.tencenttdmq.com:9876"
// 授权角色名
var secretKey = "admin"
// 授权角色密钥
var accessKey = "eyJrZXlJZC...."
// 命名空间全称
var nameSpace = "MQ_INST_rocketmqem4xxxx"
// 生产者组名称
var groupName = "group1"
// 创建消息生产者
p, _ := rocketmq.NewProducer(
    // 设置服务地址

producer.WithNsResolver(primitive.NewPassthroughResolver([]string{server
Address})),
```

```
    // 设置acl权限
    producer.WithCredentials(primitive.Credentials{
        SecretKey: secretKey,
        AccessKey: accessKey,
    }),
    // 设置生产组
    producer.WithGroupName(groupName),
    // 设置命名空间名称
    producer.WithNamespace(nameSpace),
    // 设置发送失败重试次数
    producer.WithRetry(2),
)
// 启动producer
err := p.Start()
if err != nil {
    fmt.Printf("start producer error: %s", err.Error())
    os.Exit(1)
}
```

延时消息

```
// topic名称
var topicName = "topic1"
// 生产者组名称
var groupName = "group1"
// 创建消息生产者
p, _ := rocketmq.NewProducer(
    // 设置服务地址

producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"https
://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876"})),
    // 设置acl权限
    producer.WithCredentials(primitive.Credentials{
        SecretKey: "admin",
        AccessKey: "eyJrZXlJZC......",
    }),
    // 设置生产组
    producer.WithGroupName(groupName),
    // 设置命名空间名称
    producer.WithNamespace("rocketmq-xxx|namespace_go"),
    // 设置发送失败重试次数
```

```
    producer.WithRetry(2),
)
// 启动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
Client! This is a delay message."))
    // 设置延迟等级
    // 等级与时间对应关系：
    // 1s、 5s、 10s、 30s、 1m、 2m、 3m、 4m、 5m、 6m、 7m、 8m、 9m、
10m、 20m、 30m、 1h、 2h;
    // 1    2    3     4     5    6   7    8   9   10   11   12  13   14
15    16   17   18
    //如果想用延迟级别，那么设置下面这个方法

    msg.WithDelayTimeLevel(3)
    //如果想用任意延迟消息，那么设置下面这个方法，WithDelayTimeLevel 就不要设置了，
单位为具体的毫秒，如下则是10s后投递
    delayMills := int64(10 * 1000)
    msg.WithProperty("__STARTDELIVERTIME",
strconv.FormatInt(time.Now().UnixMilli()+delayMills, 10))
    // 发送消息

     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())
    }
}

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

> **说明：**
> 以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
nameSpace	命名空间的名称，在控制台命名空间页面复制。如果您使用的是4.x通用集群或者5.x集群，此处置空即 
serverAd	集群接入地址，控制台集群基本信息页面的接入信息模块获取。（注意：需要在接入地址前面加上 http https:// 否则无法解析）。

dress	
groupName	生产者组名称，在控制台 Group 管理页面复制。 

3. 发送消息同上（以同步发送方式为例）。

```
// topic名称
var topicName = "topic1"
// 构造消息内容
msg := &primitive.Message{
    Topic: topicName, // 设置topic名称
    Body:  []byte("Hello RocketMQ Go Client! This is a new message."),
}
// 设置tag
msg.WithTag("TAG")
// 设置key
msg.WithKeys([]string{"yourKey"})
// 发送消息
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())
}
```

参数	说明
topicName	Topic 的名称，在控制台 Topic 管理页面复制。 
TAG	消息 TAG 标识。
yourKey	消息业务 key。

资源释放。

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

> **说明：**
> 异步发送、单向发送等，可参见 demo 示例 或参见 rocketmq-client-go 示例 。

4. 创建消费者。

```
// 服务接入地址 （注意：需要在接入地址前面加上 http:// 或 https:// 否则无法解析）
var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876"
// 授权角色名
var secretKey = "admin"
// 授权角色密钥
var accessKey = "eyJrZXlJZC...."
// 命名空间全称
var nameSpace = "rocketmq-xxx|namespace_go"
// 生产者组名称
var groupName = "group11"
// 创建consumer
c, err := rocketmq.NewPushConsumer(
    // 设置消费者组
    consumer.WithGroupName(groupName),
    // 设置服务地址

consumer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddress})),
    // 设置acl权限
    consumer.WithCredentials(primitive.Credentials{
        SecretKey: secretKey,
        AccessKey: accessKey,
    }),
    // 设置命名空间名称
    consumer.WithNamespace(nameSpace),
    // 设置从起始位置开始消费
    consumer.WithConsumeFromWhere(consumer.ConsumeFromFirstOffset),
    // 设置消费模式（默认集群模式）
    consumer.WithConsumerModel(consumer.Clustering),

    //广播消费,设置一下实例名，设置为应用的系统名即可。如果不设置，会使用pid，这会导致重启消费重复
    consumer.WithInstance("xxxx"),
)
if err != nil {
    fmt.Println("init consumer2 error: " + err.Error())
    os.Exit(0)
}
```

说明：

以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
namespace	命名空间的名称，在控制台命名空间页面复制。如果您使用的是4.x通用集群或者5.x集群，此处可填写 ID。 
serverAd	集群接入地址，控制台集群基本信息页面的接入信息模块获取。（注意：需要在接入地址前面加上 http https:// 否则无法解析）。

dress	
groupName	生产者组名称，在控制台 Group 管理页面复制。 

5. 消费消息。

```
// topic名称
var topicName = "topic1"
// 设置订阅消息的tag
selector := consumer.MessageSelector{
    Type:       consumer.TAG,
    Expression: "TagA || TagC",
}
// 设置重新消费的延迟级别，共支持18种延迟级别。下面是延迟级别与延迟时间的对应关系
// 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.MessageExt) (consumer.ConsumeResult, error) {
```

```
    fmt.Printf("subscribe callback len: %d \n", len(msgs))
    // 设置下次消费的延迟级别
    concurrentCtx, _ := primitive.GetConcurrentlyCtx(ctx)
    concurrentCtx.DelayLevelWhenNextConsume = delayLevel // only run
when return consumer.ConsumeRetryLater

    for _, msg := range msgs {
        // 模拟重试3次后消费成功
        if msg.ReconsumeTimes > 3 {
            fmt.Printf("msg ReconsumeTimes > 3. msg: %v", msg)
            return consumer.ConsumeSuccess, nil
        } else {
            fmt.Printf("subscribe callback: %v \n", msg)
        }
    }
    // 模拟消费失败，回复重试
    return consumer.ConsumeRetryLater, nil
})
if err != nil {
    fmt.Println(err.Error())
}
```

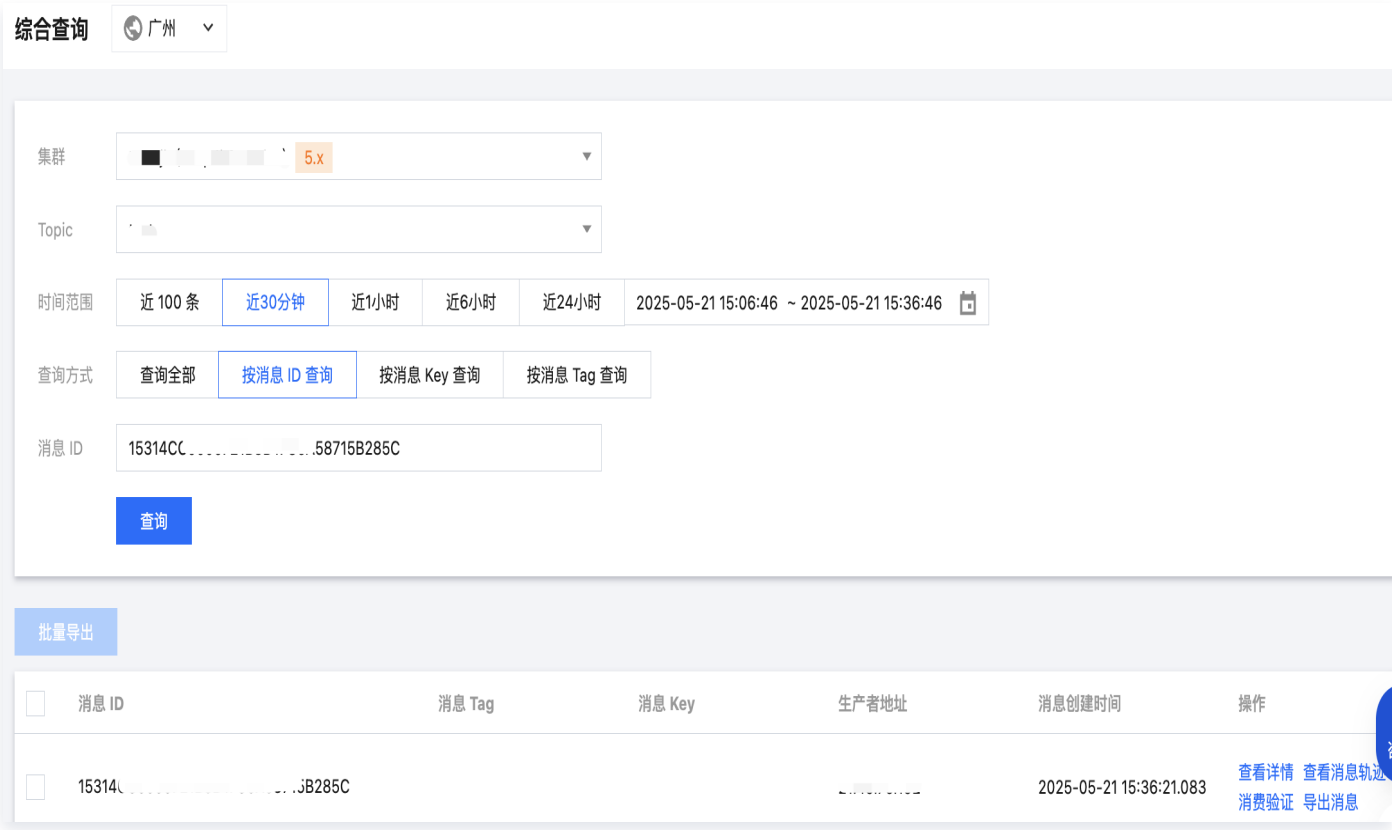
参数	说明
topicName	Topic 的名称，在控制台 Topic 页面复制。
Expression	消息 TAG 标识。
delayLevel	设置重新消费的延迟级别，共支持18种延迟级别。

6. 消费消息（消费者消费消息必须在订阅之后）。

```
// 开始消费
err = c.Start()
if err != nil {
    fmt.Println(err.Error())
    os.Exit(-1)
}
time.Sleep(time.Hour)
// 资源释放
```

```
err = c.Shutdown()
if err != nil {
    fmt.Printf("shundown Consumer error: %s", err.Error())
}
```

7. 查看消费详情。发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询 > 综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



说明：

本文简单介绍了使用 Go 客户端进行简单的收发消息，更多操作可以下载 Demo，或者可参见 rocketmq-client-go 示例。

Python SDK

最近更新时间：2025-06-10 16:17:03

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：准备环境

Rocketmq-client Python 基于 rocketmq-client-cpp 进行包装，因此需要先安装 `librocketmq` 。

> **说明：**
> - 目前 Python 客户端仅支持 Linux 和 macOS 操作系统，暂不支持 Windows 系统。
> - 在使用 Python SDK 时要注意安装的 Python 支持的底层芯片架构类型（x86 或是 ARM），例如使用 `'64bit','ELF'` （即 x86_64 架构）的 Python 版本，则使用 ARM 芯片的 macOS 系统会出现报错。

1. 安装 librocketmq （版本2.0.0及以上）， 安装教程参见 librocketmq 安装 。
2. 执行如下命令安装 rocketmq-client-python。

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

步骤2：生产消息

创建并编译运行生产消息程序。

```
from rocketmq.client import Producer, Message

# 初始化生产者，并设置生产组信息，组名称使用全称，例：rocketmq-
```

```
xxx|namespace_python%group1 生产消息时，客户端无需重复创建。
producer = Producer(groupName)
# 设置服务地址
producer.set_name_server_address(nameserver)
# 设置权限（角色名和密钥）
producer.set_session_credentials(
    accessKey,  # 角色密钥
    secretKey,  # 角色名称
    ''
)
# 启动生产者
producer.start()

# 组装消息
# topic 拼接 namespace
msg = Message(namespace + '%' + topicName)
# 设置keys
msg.set_keys(TAGS)
# 设置tags
msg.set_tags(KEYS)
# 消息内容
msg.set_body('This is a new message.')

# 发送同步消息
ret = producer.send_sync(msg)
print(ret.status, ret.msg_id, ret.offset)
# 资源释放
producer.shutdown()
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
groupN	生产者组名称，在控制台 Group 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%group名称` ，例如 MQ_INSTxxx_aaa%GroupTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Group 名称即可。

name	
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
accessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 

namespace	命名空间的名称，在控制台命名空间页面复制。如果您使用的是4.x通用集群或者5.x集群，此处可填写集 
topicName	Topic 的名称，在控制台 Topic 管理页面复制。 
TAGS	用来设置消息的 TAG。
KEYS	设置消息业务 key。

当前开源社区的 Python 客户端生产消息存在一定缺陷，导致同个 Topic 的不同队列间负载不均，详情可参见 缺陷详情 。

步骤3：消费消息

创建并编译运行消费消息程序。

```
import time

from rocketmq.client import PushConsumer, ConsumeStatus


# 消息处理回调
```

```
def callback(msg):
    # 模拟业务
    print('Received message. messageId: ', msg.id, ' body: ', msg.body)
    # 消费成功回复CONSUME_SUCCESS
    return ConsumeStatus.CONSUME_SUCCESS
    # 消费成功回复消息状态
    # return ConsumeStatus.RECONSUME_LATER


# 初始化消费者，并设置消费者组信息
consumer = PushConsumer(namespace + '%' + groupName)
# 设置服务地址
consumer.set_name_server_address(nameserver)
# 设置权限（角色名和密钥）
consumer.set_session_credentials(
    accessKey,  # 角色密钥
    secretKey,  # 角色名称
    ''
)
# 订阅topic
consumer.subscribe(namespace + '%' + topicName, callback, TAGS)
print(' [Consumer] Waiting for messages.')
# 启动消费者
consumer.start()

while True:
    time.sleep(3600)
# 资源释放
consumer.shutdown()
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

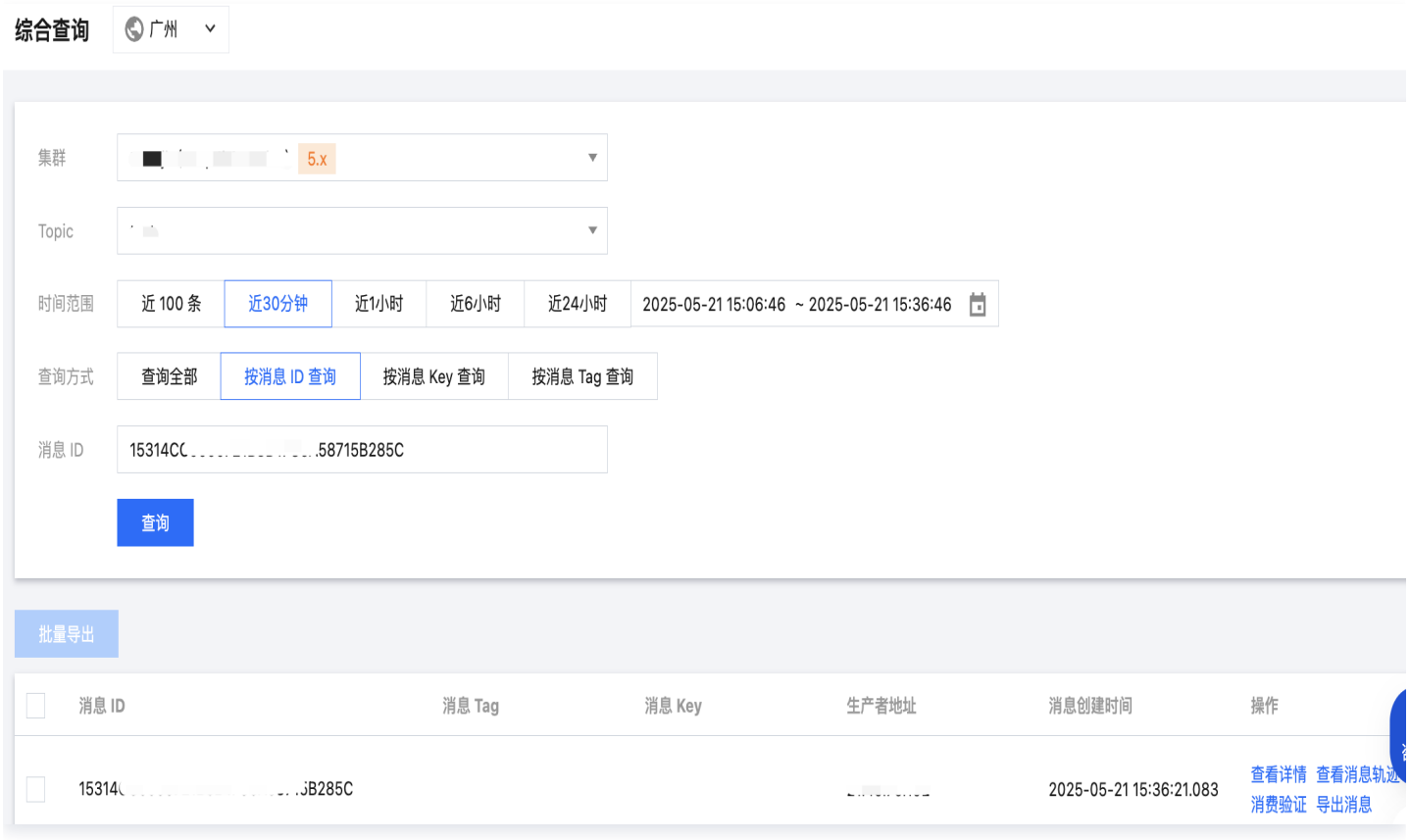
参数	说明
namesp	命名空间的名称，在控制台命名空间页面复制。如果您使用的是4.x通用集群或者5.x集群，此处可填写集

ace	
groupName	生产者组名称，在控制台 Group 管理页面复制。 
nameserver	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 
secretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。
access	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 

Key	
topicName	Topic 的名称，在控制台 Topic 管理页面复制。

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询** > **综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。



说明：

上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 Demo 或 RocketMQ-Client-Python 示例 。

C# SDK

最近更新时间：2025-06-10 16:17:03

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
- 安装 DotNet
- 下载 Demo

操作步骤

步骤1：准备环境

根据自己的 IDE 环境，新建一个项目，然后安装 NewLife.RocketMQ 依赖。

```
dotnet add package NewLife.RocketMQ --version 2.0.2022.325-beta0806
```

步骤2：生产消息

创建并编译运行生产消息程序。

```
//生产者
var mq = new Producer
{
    Topic = "TopicTest1", //对于非通用版集群，需要拼接完整的topic，如
MQ_INSTxxx_aaa%TopicTest
    NameServerAddress = "127.0.0.1:9876",// 腾讯云页面填写
    Log = XTrace.Log,
    AclOptions = new AclOptions
    {
        AccessKey = "页面的ak",
        SecretKey = "页面的sk",
    },
};
mq.Start();
for (var i = 0; i < 10; i++)
{
    var str = "生产消息测试" + i;
```

```
    var sr = mq.Publish(str, "TagA");
}
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

参数	说明
Topic	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。 
NameServerAddress	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 
Sec	角色名称，在控制台的集群权限页面 SecretKey 列复制。

retKey	
AccessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 
TAGS	用来设置消息的 TAG。

步骤3：消费消息

创建并编译运行消费消息程序。

```
//消费者
var consumer = new Consumer
{
    Topic = "TopicTest1",//对于非通用版集群，需要拼接完整的topic，如
MQ_INSTxxx_aaa%TopicTest
    Group = "test", //对于非通用版集群，需要拼接完整的Goup，如
MQ_INSTxxx_aaa%GroupTest
    NameServerAddress = "127.0.0.1:9876",
    FromLastOffset = true,
    SkipOverStoredMsgCount = 0,
    BatchSize = 20,
    Log = XTrace.Log,
    AclOptions = new AclOptions
    {
        AccessKey = "页面的ak",
        SecretKey = "页面的sk",
    },
};
consumer.OnConsume = (q, ms) =>
```

```
{
    XTrace.WriteLine("[{0}@{1}]收到消息 [{2}]", q.BrokerName, q.QueueId,
ms.Length);
    foreach (var item in ms.ToList())
    {
        XTrace.WriteLine($"消息 {item.Keys}，发送时间
{item.BornTimestamp.ToDateTime()}，内容 {item.Body.ToStr()}");
    }
    return true;
};
consumer.Start();
```

说明：
以下参数需登录 TDMQ RocketMQ 版控制台 获取。

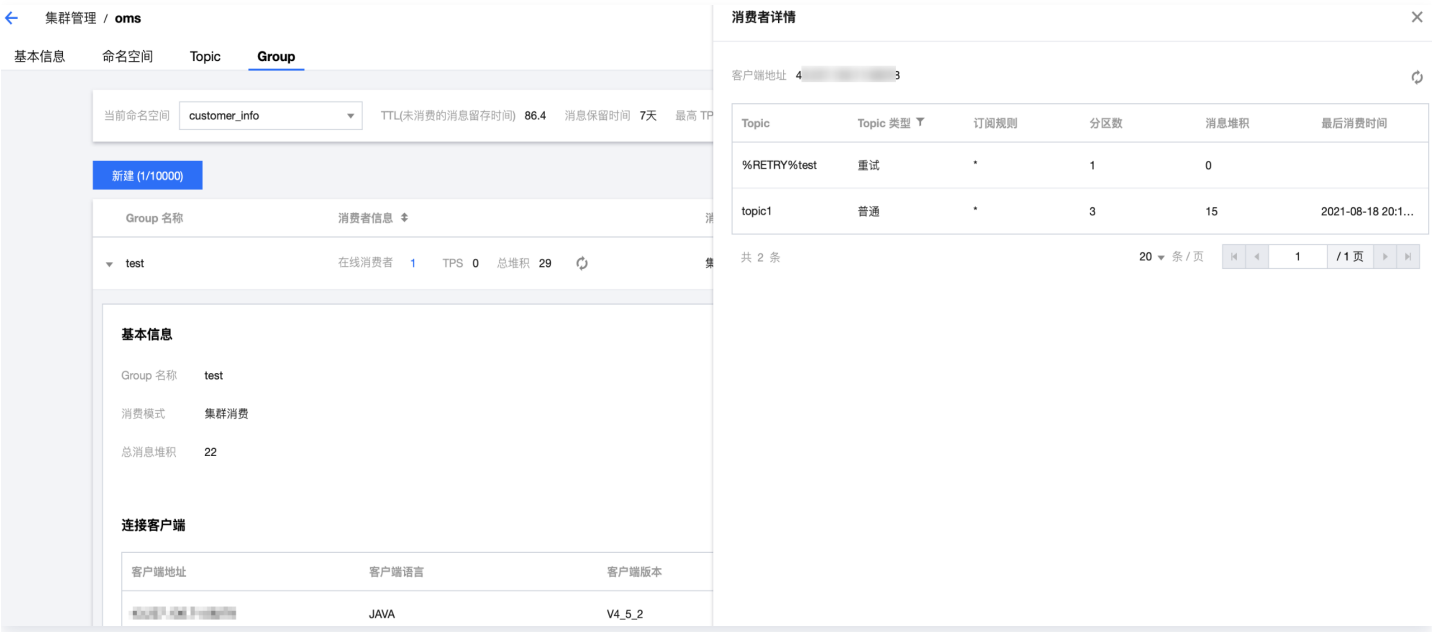
参数	说明
Topic	Topic 的名称，在控制台 Topic 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%topic名称` ，例如 MQ_INSTxxx_aaa%TopicTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Topic 名称即可。 

Group	Group 的名称，在控制台 Group 管理页面复制。 • 4.x虚拟集群/专享集群：此处需拼接命名空间名称，格式为 `namespace全称%group名称` ，例如 MQ_INSTxxx_aaa%GroupTest。 • 4.x通用集群/5.x集群：此处无需拼接，填写 Group 名称即可。 
NameServerAddress	集群接入地址，控制台集群基本信息页面的接入信息模块获取。 
SecretKey	角色名称，在控制台的集群权限页面 SecretKey 列复制。

AccessKey	角色密钥，在控制台的集群权限页面 AccessKey 列复制。 

步骤4：查看消费详情

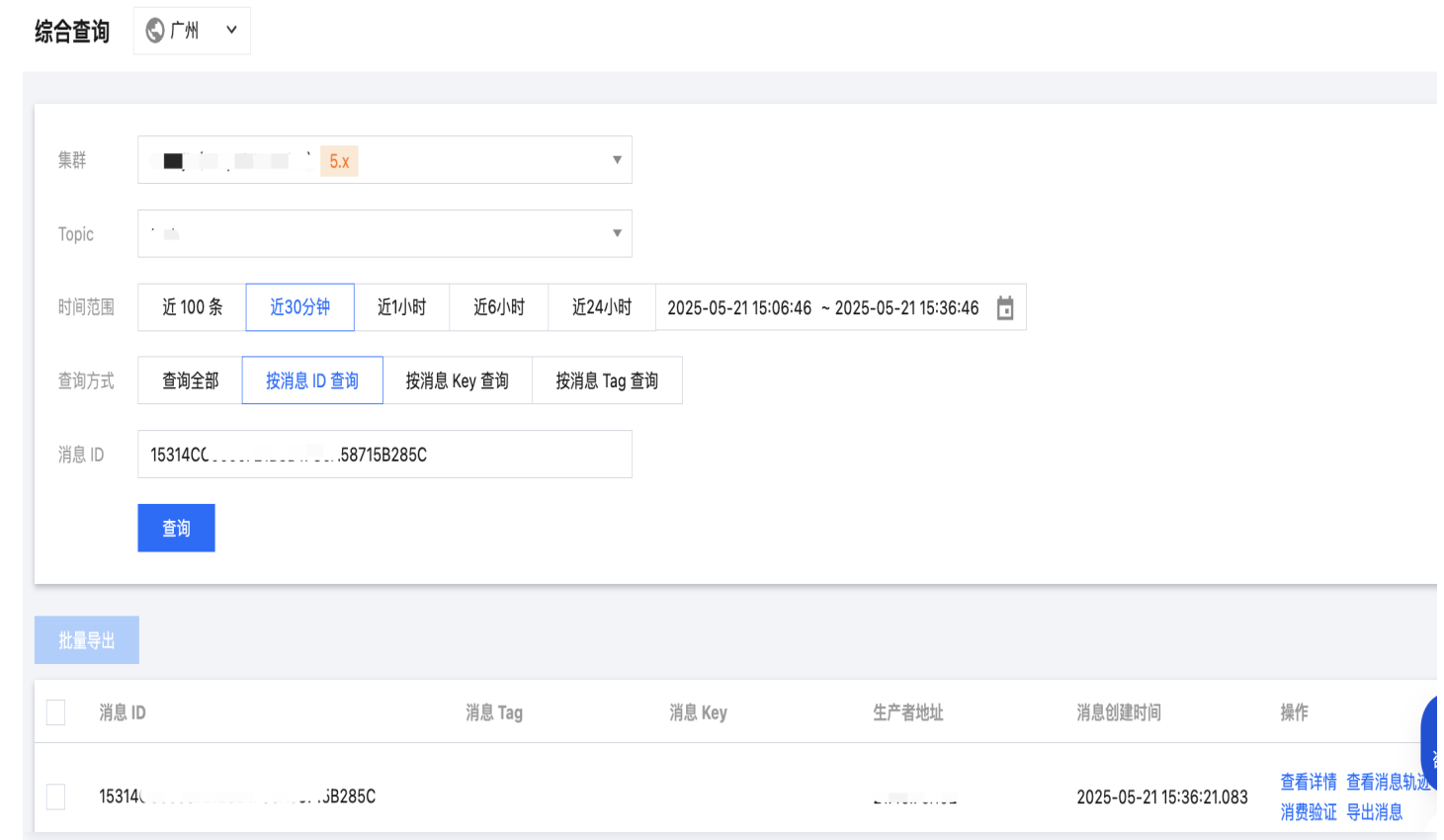
登录 RocketMQ 控制台，在**集群管理** > **Group** 页面，可查看与 Group 连接的客户端列表，单击操作列的**查看详情**，可查看消费者详情。



> **说明：**
> 上述是对消息的发布和订阅方式的简单介绍。更多操作可参见 GitHub Demo 。

步骤4. 查看消息详情

发送完成消息后会得到一个消息 ID（messageID），您可以在控制台的**消息查询** > **综合查询**页面查询刚刚发送的消息，以及该消息的详情和轨迹等信息。


综合查询
广州
集群
5.x
Topic
时间范围
近 100 条
近30分钟
近1小时
近6小时
近24小时
2025-05-21 15:06:46 ~ 2025-05-21 15:36:46
查询方式
查询全部
按消息 ID 查询
按消息 Key 查询
按消息 Tag 查询
消息 ID
15314CC.....58715B285C
查询
批量导出
消息 ID
消息 Tag
消息 Key
生产者地址
消息创建时间
操作
15314(.....)B285C
2025-05-21 15:36:21.083
查看详情
查看消息轨迹
消费验证
导出消息