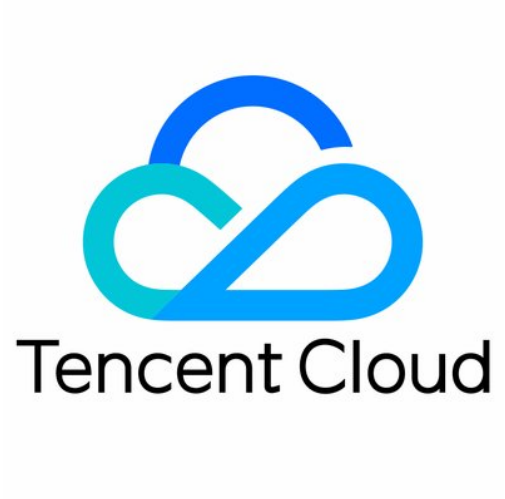
TDMQ for RocketMQ

SDK Documentation

Product Documentation



Contents

SDK Documentation
Compatibility Description

Last updated：2025-07-15 18:32:06

RocketMQ 5.0 introduces a brand-new 5.x SDK based on the gRPC protocol. The new SDK version offers a more lightweight framework and better multilingual support. We highly recommend that you use this version. Moreover, TDMQ for RocketMQ 5.x series continues to support the access through the 4.x SDK for existing businesses. The compatibility details are as follows:

<table>
<tr><th>Server Version</th><th colspan="2">Client Version</th><th>Compatibility</th></tr>
<tr><td rowspan="3">5.x</td><td colspan="2">5.x SDK</td><td>Fully compatible</td></tr>
<tr><td rowspan="2">4.x SDK</td><td>Versions of 4.9.5 or later</td><td>PushConsumer CONSUME_FROM_TIMESTAMP is not effective (you can reset position in the console).
The maximum value of consumer setPullBatchSize is 32.
Transactional messages have compatibility issues</td></tr>
<tr><td>Versions earlier than 4.9.5</td><td>PushConsumer CONSUME_FROM_TIMESTAMP is not effective (you can reset position in the console).
The maximum value of consumer setPullBatchSize is 32.
PullConsumer not currently supported.
Transactional messages have compatibility issues.
Transaction commit or check may fail. Not recommended for use.</td></tr>
</table>

5.x SDK
Use of Java SDK

Last updated：2025-07-15 18:32:06

Overview

TDMQ for RocketMQ supports multiple language SDKs to send and receive different types of messages. This document uses Java SDK calls as an example to introduce the operation process of sending and receiving ordinary messages by connecting to TDMQ for RocketMQ server implementation through the 5.x SDK.

Prerequisites

You have created and prepared the required resources.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo.

Directions:

Step 1: Installing the Java Dependency Library

Incorporate the relevant dependencies in the Java project. A Maven project is used as an example. Add the following dependencies to pom.xml:

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.5</version>
        </dependency>
</dependencies>
```

Notes:

Note: If you are using Spring Boot, you may encounter a Java dependency conflict with annotations-api. In this case, you can just add an exclusion.

```
<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>
```

Step 2: Producing Messages

In the created Java project, create a message sending program and run it.

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

    private NormalMessageSyncProducer() {
    }

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

        //Adding ak and sk in the configuration
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // Fill in the access location provided by Tencent Cloud
        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 pat
        final Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the topic name(s), which is optional but recommended. It makes p
            // 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(Stand
        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 i
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            .build();
        try {
            final SendReceipt sendReceipt = producer.send(message);
            log.info("Send message successfully, messageId={}", sendReceipt.getMess
        } catch (Throwable t) {
            log.error("Failed to send message", t);
        }
        // Close the producer when you don't need it anymore.
        producer.close();
    }
}
```

Parameter	Description
accessKey	Role key, copied from the AccessKey column on the Cluster Permission page in the console.
secretKey	Role name, copied from the SecretKey column on the Cluster Permission page in the console.
endpoints	Obtain the cluster access address from the access information module on the console cluster basic information page.
topic	Copy the Topic name from the Topic management page in the console.

Step 3: Consuming Messages

In the created Java project, create a subscription standard message program and run it.

Tencent Cloud Message Queue TDMQ RocketMQ 5.x series supports two types of consumer clients: Push Consumer and Simple Consumer. The following example code uses Push Consumer as an example:

```
public class NormalPushConsumer {
```

```
    private static final Logger log = LoggerFactory.getLogger(NormalPushConsumer.cl

    private NormalPushConsumer() {
    }

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

        //Adding ak and sk in the configuration
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // Fill in the access location provided by Tencent Cloud
        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, FilterExpress
        String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        // In most case, you don't need to create too many consumers, singleton pat
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic, filterExpre
            .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();
    }
}
```

Parameter	Description

accessKey	Role key, copied from the AccessKey column on the Cluster Permission page in the console.
secretKey	Role name, copied from the SecretKey column on the Cluster Permission page in the console.
endpoints	Obtain the cluster access address from the access information module on the console cluster basic information page.
consumerGroup	Copy the consumer group name from the Group Management page in the console.
topic	Copy the Topic name from the Topic management page in the console.

Step 4: Viewing Message Details

After sending the completion message, you will get a message ID. You can query the just-sent message, as well as its details and path, in the console under **Message Query** > **Comprehensive Query**.

Use of Go SDK

Last updated：2025-07-15 18:32:06

Overview

This document describes how to use an open-source SDK to send and receive messages with the Golang SDK serving as example, for you to better understand the complete process of message sending and receiving.

Prerequisites

You have created and prepared the required resources.
You have installed Golang version 1.13 or higher.
You have downloaded the demo.

Directions:

Step 1: Installing the Golang Dependency Library

Incorporate the relevant dependencies in the Golang project. `go get` is used as example. Run the following command:

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

Step 2: Producing Messages

```
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"
    // Set Endpoint to the access address provided by Tencent Cloud
    Endpoint = "xxxxxx"
    // Add the configured ak to AccessKey
    AccessKey = "xxxxxx"
    // Add the configured sk to SecretKey
    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
    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,
            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)
    }
}
```

Parameter	Description
AccessKey	Role key, copied from the AccessKey column on the cluster permissions page in the console.
SecretKey	Role name, copied from the SecretKey column on the cluster permissions page in the console.
Endpoints	Obtain the cluster access address from the information module on the console cluster basic information page.
Topic	Copy the Topic name from the Topic management page in the console.

Step 3: Consuming Messages

TDMQ for RocketMQ 5.x series by Tencent Cloud supports two types of clients: Push Consumer and Simple Consumer.

Note:

At this time, the community version of the Golang SDK only supports Simple Consumer.

The following sample code uses Simple Consumer as an example:

```
package main

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

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

```
const (
    Topic         = "xxxxxx"
    ConsumerGroup = "xxxxxx"
    // Set Endpoint to the access address provided by Tencent Cloud
    Endpoint = "xxxxxx"
    // Add the configured ak to AccessKey
    AccessKey = "xxxxxx"
    // Add the configured sk to SecretKey
    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 messages in a loop
)

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
    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.FilterExpress
            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 simpleConsumer.GracefulStop()
    for {
        fmt.Println("start receive message")
        mvs, err := simpleConsumer.Receive(context.TODO(), maxMessageNum, invisible
        if err != nil {
            fmt.Println(err)
        }
        // ack message
        for _, mv := range mvs {
            simpleConsumer.Ack(context.TODO(), mv)
            fmt.Println(mv)
        }
        fmt.Println("wait a moment")
        fmt.Println()
        time.Sleep(time.Second * 1)
    }
}
```

Parameter	Description
accessKey	Role key, copied from the AccessKey column on the cluster permissions page in the console.
secretKey	Role name, copied from the SecretKey column on the cluster permissions page in the console.
endpoints	Obtain the cluster access address from the information module on the console cluster basic information page.
consumerGroup	Copy the consumer group name from the Group Management page in the console.
topic	Copy the Topic name from the Topic management page in the console.

Step 4: Viewing Message Details

After sending the completion message, you will get a message ID. You can query the just-sent message, as well as its details and path, in the console under **Message Query** > **Comprehensive Query**.

Use of C++ SDK

Last updated：2025-07-15 18:32:06

Overview

This document describes how to use an open-source SDK to send and receive messages with the SDK for C++ serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Prerequisites

You have created and prepared the required resources.
You have installed a compiler suite supporting C++11.
You have installed Bazel version 5.2.0 or CMake version 3.13 and later.
If you use CMake for compilation, gRPC version 1.46.3 is recommended due to incompatibilities between higher versions and the SDK.
You have downloaded the demo.

Directions:

Step 1: Installing the SDK for C++

Install the SDK.

Note:

TDMQ for RocketMQ 5.x series does not currently support TLS. A patch must be applied.

Step 2: Producing Messages

```
#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("0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHI
    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";
// Enter the access address provided by Tencent Cloud
static const std::string access_point = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
// Add the configured ak and 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_k
    }

    // In most case, you don't need to create too many producers, singletion patter
    auto producer = Producer::newBuilder()
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)
                                       .withCredentialsProvider(credentials_provide
                                       .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:
            } else {
                std::cout << "Publish message to " << topic << " OK. 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;
}
```

Parameter	Description
access_key	Role key, copied from the AccessKey column on the cluster permissions page in the console.
access_secret	Role name, copied from the SecretKey column on the cluster permissions page in the console.
access_point	Get the cluster access address from the access information module on the console cluster basic information page.
topic	Copy the Topic name from the Topic management page in the console.

Step 3: Consuming Messages

Tencent Cloud Message Queue TDMQ RocketMQ 5.x series supports two types of clients: Push Consumer and Simple Consumer.

The following code is an example based on Push Consumer:

```
#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";
// Enter the access address provided by Tencent Cloud
static const std::string access_point = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
// Add the configured ak and 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=" <
        return ConsumeResult::SUCCESS;
    };

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

    // In most case, you don't need to create too many consumers, singletion patter
    auto push_consumer = PushConsumer::newBuilder()
            .withGroup(group)
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)
                                       .withRequestTimeout(std::chrono::seconds(3))
                                       .withCredentialsProvider(credentials_provide
                                       .withSsl(false)
                                       .build())
            .withConsumeThreads(4)
            .withListener(listener)
            .subscribe(topic, tag)
            .build();

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

    return EXIT_SUCCESS;
}
```

Parameter	Description
access_key	Role key, copied from the AccessKey column on the cluster permissions page in the console.
access_secret	Role name, copied from the SecretKey column on the cluster permissions page in the console.
access_point	Get the cluster access address from the access information module on the console cluster basic information page.
group	Copy the consumer group name from the Group Management page in the console.

topic	Copy the Topic name from the Topic management page in the console.

Step 4: Viewing Message Details

After sending the completion message, you will get a message ID. You can query the just-sent message on the console in **Message Query** > **Comprehensive Query**, as well as the message details and path.

C# SDK Usage

Last updated：2025-07-15 18:32:06

Scenarios

In this document, C# SDK call is used as an example to introduce the operation process of sending and receiving messages through the open source SDK, which helps you better understand the whole process of sending and receiving messages.

Prerequisites

Complete resource creation and preparation.
Install the DotNet environment
Download the demo.

Operation Steps

Step 1: Install the RocketMQ5 SDK Dependency Library

To introduce dependencies into a C#project, use the following command:

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

Step 2: Producing Messages

```
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 => bu
        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/S
```

```
            AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2Unencrypt
            const string accessKey = "yourAccessKey";
            const string secretKey = "yourSecretKey";

            // Credential provider is optional for client configuration.
            var credentialsProvider = new StaticSessionCredentialsProvider(accessKe
            const string endpoints = "Tencent Cloud official website access address
            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 pa
            // 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
                .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={sendRecei

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

        }
    }
}
```

Parameter	Description
accessKey	Role key, copied from the AccessKey column on the Cluster Permission page of the console.

secretKey	Role name, copied from the SecretKey column on the Cluster Permission page of the console.
endpoints	Obtain the cluster access address from the access information module on the console cluster basic information page.
topic	Copy the Topic name from the Topic management page in the console.

Step 3: Consuming Messages

Tencent Cloud Message Queue TDMQ RocketMQ 5.x series supports two consumption modes: Push Consumer and Simple Consumer.

Notes:

Community Edition C# SDK supports Push Consumer after 5.2.0-rc1

The following example code uses Simple Consumer (in case of low message volume, there will be some delay when using single-thread Simple Consumer to consume messages. If business assessment confirms low message volume, enable multi-thread pull or use 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 => bu
        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/S
            AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2Unencrypt
            const string accessKey = "yourAccessKey";
            const string secretKey = "yourSecretKey";

            // Credential provider is optional for client configuration.
            var credentialsProvider = new StaticSessionCredentialsProvider(accessKe
            const string endpoints = "rmq-xxx.rocketmq.gz.qcloud.tencenttdmq.com:80
            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 pa
            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.FromSe
                foreach (var message in messageViews)
                {
                    logger.LogInformation(
                        $"Received a message, topic={message.Topic}, message-id={me
                    await simpleConsumer.Ack(message);
                    logger.LogInformation($"Message is acknowledged successfully, m
                    // await simpleConsumer.ChangeInvisibleDuration(message, TimeSp
                    // Logger.LogInformation($"Changing message invisible duration
                }
            }
            // 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(P

    private static readonly string Endpoint = Environment.GetEnvironmentVariabl

    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/S
        // AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2Unencr

        // Credential provider is optional for client configuration.
        var credentialsProvider = new StaticSessionCredentialsProvider(accessKe
        const string endpoints = "rmq-xxx.rocketmq.gz.qcloud.tencenttdmq.com:80
        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;
        }
```

```
            }
        }
    }
```

Parameter	Description
accessKey	Role key, copied from the AccessKey column on the Cluster Permission page of the console.
secretKey	Role name, copied from the SecretKey column on the Cluster Permission page of the console.
endpoints	Obtain the cluster access address from the access information module on the console cluster basic information page.
consumerGroup	Copy the consumer group name from the Group Management page in the console.
topic	Copy the Topic name from the Topic management page in the console.

Step 4: View Message Details

After sending the completion message, you will get a message ID. You can query the just-sent message, as well as its details and path, in the console under **Message Query** > **Comprehensive Query**.

Node.Js SDK Usage

Last updated：2025-07-15 18:32:06

Scenarios

In this document, Node.js SDK call is used as an example to introduce the operation process of sending and receiving messages through the open source SDK, which helps you better understand the whole process of sending and receiving messages.

Prerequisites

Complete resource creation and preparation.
Install the Node.js development environment, such as npm.
Download demo.

Operation Steps

Step 1: Install Node.Js SDK in the Project

1. Install SDK:

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

2. After installation, the corresponding dependencies will be in the project's package.json file.

Step 2. Producing Messages

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

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

const receipt = await producer.send({
```

```
  topic: 'TopicTest', // Topic created in the console
  body: Buffer.from(JSON.stringify({
    hello: 'rocketmq-client-nodejs world',
    now: Date(),
  })),
});
console.log(receipt);
```

Parameter	Description
accessKey	Role key, copied from the AccessKey column on the cluster permissions page in the console.
accessSecret	Role name, copied from the SecretKey column on the cluster permissions page in the console.
endpoints	Get the cluster access address from the access information module on the console cluster basic information page.
topic	Copy the Topic name from the Topic management page in the console.

Step 3. Consuming Messages

The Node.js SDK currently supports Simple Consumer. The following example code demonstrates Simple Consumer:

```
import {SimpleConsumer } from 'rocketmq-client-nodejs';
const simpleConsumer = new SimpleConsumer({
    endpoints: 'rmq-xxx.rocketmq.gz.qcloud.tencenttdmq.com:8080', // Tencent Cloud
  sessionCredentials: {
    accessKey: 'yourAccessKey', // ak
    accessSecret: 'yourSecretKey', // sk
  },
  consumerGroup: 'nodejs-demo-group', //consumption group
  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);
}
```

Parameter	Description
accessKey	Role key, copied from the AccessKey column on the cluster permissions page in the console.
accessSecret	Role name, copied from the SecretKey column on the cluster permissions page in the console.
endpoints	Get the cluster access address from the access information module on the console cluster basic information page.
consumerGroup	Copy the consumer group name from the Group Management page in the console.
subscriptions	Copy the Topic name from the Topic management page in the console.

Step 4. Viewing Message Details

After sending the completion message, you will get a message ID (messageID). You can query the just-sent message on the console in **Message Query** > **Comprehensive Query**, as well as the message details and path, etc.

4.x SDK
Spring Boot Starter
Sending and Receiving General Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Directions

Step 1. Add dependencies

Add dependencies to the `pom.xml` file. Version 2.3.3 (latest) is recommended, as it simultaneously supports Spring Boot 2 and 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>
```

Step 2. Prepare configurations

Add configuration information to the configuration file.

```
server:
  port: 8082

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

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

```
    subExpression: TAG1
  consumer2:
    group: group222
    topic: testdev1
    subExpression: TAG2
```

Parameter	Description
name-server	Retrieve the cluster access address from the access information module on the console cluster basic information page.
group	Producer name. Users can self-define it or use the corresponding Topic name.
secret-key	Role name, copied from the SecretKey column on the Role Management page in the console.
access-key	Role key, copied from the AccessKey column on the Role Management page in the console.
namespace	Namespace name, copied from the namespace page in the console. **If you use a 5.x cluster or 4.x generic cluster, fill in the cluster ID here just.**
topic	Copy the topic name from the Topic management page in the console.
subExpression	A parameter used to set the message tag.

Step 3. Send messages

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

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

      @Autowired
      private RocketMQTemplate rocketMQTemplate;
```

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

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

3. Below is a complete sample.

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

Note:

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

Step 4. Consume messages

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

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

```
            }
        }
```

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

Note:

For a complete sample, download the demo or obtain the demo in TencentCloud/rocketmq-demo in GitHub.

Step 5. View consumption details

After sending the completion message, you will get a message ID. You can query the just-sent message on the console in **Message Query** > **Comprehensive Query**, as well as its details and path information.

Sending and Receiving Filtered Messages

Last updated：2025-07-15 18:32:06

Overview

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

Prerequisites

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

Directions

Sending a message

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

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

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

Consuming a message

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

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

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

Sending and Receiving Delayed Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Directions

Sending a message

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

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

The delay level and delay time mapping is shown in the table below:

Latency Level	Time
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

Consuming a message

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

Use of Spring Cloud Stream

Last updated：2025-07-15 18:32:07

Overview

This document describes how to send and receive messages with the Spring Cloud Stream serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Prerequisites

You have created the required resources.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo or visited the GitHub project.

Directions:

Step 1: Incorporating Dependencies

Incorporate the spring-cloud-starter-stream-rocketmq dependency in the pom.xml file. The current recommended version is 2021.0.4.0.

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

Step 2: Adding Configurations

Add the corresponding RocketMQ configurations to the configuration file.

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          # Full name of the service address
          name-server: rmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080
```

```
          # Role name
          secret-key: admin
          # Role key
          access-key: eyJrZXlJZ...
          # producer group
          group: producerGroup
        bindings:
          # Channel name, corresponding to the channel name under spring.cloud.stre
          Topic-TAG1-Input:
            consumer:
              # Subscribed tag type, configured according to real consumer conditio
              subscription: TAG1
          # Channel name
          Topic-TAG2-Input:
            consumer:
              subscription: TAG2
      bindings:
        # Channel name
        Topic-send-Output:
          # Specify topic, corresponding to the created topic name
          destination: TopicTest
          content-type: application/json
        # Channel name
        Topic-TAG1-Input:
          destination: TopicTest
          content-type: application/json
          group: consumer-group1
        # Channel name
        Topic-TAG2-Input:
          destination: TopicTest
          content-type: application/json
          group: consumer-group2
```

Note:

1. Currently only `2.2.5-RocketMQ-RC1` and `2.2.5.RocketMQ.RC2` and above versions support **namespace** configuration. For other versions, you need to concatenate the topic and group name.

as follows:

rocketmq-pngrpmk94d5o|stream%topic (format: namespace full name%topic name)

rocketmq-pngrpmk94d5o|stream%group (format: namespace full name%group name)

New virtual cluster and dedicated cluster formats are as follows:

MQ_INST_rocketmqpj79obd2ew7v_test%topic (format: namespace full name%topic name)

MQ_INST_rocketmqpj79obd2ew7v_test%group (format: namespace full name%group name)

2. Configuration aspects: `2.2.5-RocketMQ-RC1` and `2.2.5.RocketMQ.RC2` use `subscription` as the subscription configuration item, while other legacy versions use `tags` .

The complete configuration item reference for other versions is as follows:

```
spring:
     cloud:
       stream:
         rocketmq:
           bindings:
             # Channel name, corresponding to the channel name under spring.cloud.s
             Topic-test1:
               consumer:
                 # Subscribed tag type, configured according to real consumer condi
                 tags: TAG1
             # Channel name
             Topic-test2:
               consumer:
                 tags: TAG2
           binder:
             # Full name of the service address
             name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080
             # Role name
             secret-key: admin
             # Role key
             access-key: eyJrZXlJZ...
         bindings:
           # Channel name
           Topic-send:
             # Specified topic
             destination: topic1
             content-type: application/json
             # Use the full name of the group
             group: group1
           # Channel name
           Topic-test1:
             destination: topic1
             content-type: application/json
             group: group1
           # Channel name
           Topic-test2:
             destination: topic1
             content-type: application/json
             group: group2
```

Parameter	Description
name-server	Acquire the cluster access address from the access addresses in the Operation column of the Cluster list on the Cluster Management page in the console. Acquire the name access

	addresses of the shared and dedicated clusters of the new version from the Namespace list.
secret-key	Role name, copied from the SecretKey column on the Role Management page in the console.
access-key	Role key, copied from the AccessKey column on the Role Management page in the console.
namespace	Copy the namespace name from the namespace page in the console. If you use a 4.x generic cluster or 5.x cluster, fill in the Cluster ID here.
group	Copy the producer group name from the Group Management page on the console.
destination	Topic name, which can be copied from the Topic tab on the console.

Step 3: Configuring the Channel

A channel consists of input and output. These can be individually configured as needed.

```
/**
 * Custom channel binder
 */
public interface CustomChannelBinder {

    /**
     * Send the message (message producer)
     * Bind the channel name specified in the configuration settings.
     */
    @Output("Topic-send-Output")
    MessageChannel sendChannel();


    /**
     * Receive Message 1 (Consumer 1)
     * Bind the channel name specified in the configuration settings.
     */
    @Input("Topic-TAG1-Input")
    MessageChannel testInputChannel1();

    /**
     * Receive Message 2 (Consumer 2)
```

```
     * Bind the channel name specified in the configuration settings.
     */
    @Input("Topic-TAG2-Input")
    MessageChannel testInputChannel2();
}
```

Step 4: Adding Annotations

Add relevant annotations to the configuration or boot class. If there are multiple configured binder configurations, each must be specifically specified within these annotations.

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

Step 5: Sending the Messages

1. Inject `CustomChannelBinder` into the class of the message to be sent.

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

2. Send the messages by calling the corresponding output stream channel.

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

Step 6: Consuming the Messages

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

    /**
     * Monitor channel (designated by channel name in configuration)
     *
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG1-Input")
    public void receive(String messageBody) {
        logger.info("Receive1: Message received via stream, messageBody = {}", mess
    }

    /**
     * Monitor channel (designated by channel name in configuration)
     *
```

```
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG2-Input")
    public void receive2(String messageBody) {
        logger.info("Receive2: Message received via stream, messageBody = {}", mess
    }
}
```

Step 7: Local Test

After the project is initiated locally, a successful startup notification will be displayed on the console.

Visit `http://localhost:8080/test-simple` via a browser. You can see a successful transmission. Keep an eye on the output log of your development IDE.

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

You can see that a message with the TAG1 has been sent, and only the subscriber of TAG1 has received the message.

Note:

For specific usage, see the GitHub Demo or Spring Cloud Stream Official Website.

SDK for Java
Sending and Receiving General Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Directions

Step 1. Install the Java dependent library

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

Note:

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

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

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

Step 2. Produce messages

Creating a message producer

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

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
nameserver	Obtain the cluster access address from the access information module on the console cluster basic information page.

Sending messages

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

Sync sending

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

Parameter	Description

topic_name	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: concatenate the namespace name here, format `namespace full name%topic name`, for example MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: no need to concatenate here, just fill in the Topic name.
TAG	A parameter used to set the message tag.

Async sending

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

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

Parameter	Description
topic_name	Copy the Topic name from the Topic management page in the console.

	4.x virtual cluster/dedicated cluster: concatenate the namespace name here, format `namespace full name%topic name`, for example MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: no need to concatenate here, just fill in the Topic name.
TAG	A parameter used to set the message tag.

One-way sending

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

Parameter	Description
topic_name	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: concatenate the namespace name here, format `namespace full name%topic name`, for example MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: no need to concatenate here, just fill in the Topic name.
TAG	A parameter used to set the message tag.

Note:

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

Step 3. Consume messages

Creating a consumer

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

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

Parameter	Description

groupName	Copy the group name from the Group Management page in the console. 4.x virtual cluster/dedicated cluster: concatenate the namespace name here, format `namespace full name%group name`, for example MQ_INSTxxx_aaa%GroupTest. 4.x generic cluster/5.x cluster: no need to concatenate here, just fill in the Group name.
nameserver	Obtain the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.

Subscribing to messages

The subscription modes vary by consumption mode.

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

Parameter	Description
topic_name	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: concatenate the namespace name here, format `namespace full name%topic name`, for example MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: no need to concatenate here, just fill in the Topic name.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Step 4. View consumption details

After sending the completion message, you will get a message ID. You can query the just-sent message, as well as its details and path, etc., in the console under **Message Query** > **Comprehensive Query**.

Note

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

Sending and Receiving Delayed Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Directions

Step 1. Install the Java dependent library

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

Note:

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

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

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

Step 2. Produce messages

Creating a message producer

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
accessKey	Role key, copied from the AccessKey column on the Role Management page in the console.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.

Sending a message

Messages with fixed delay level

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

Messages with random delay time

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

Step 3. Consume messages

Creating a consumer

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Copy the name of group from the Group management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%group name` , such as MQ_INSTxxx_aaa%GroupTest. 4.x generic cluster/5.x cluster: No need to concatenate here, just fill in the Group name.
accessKey	Role key, copied from the AccessKey column on the Role Management page in the console.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.

Subscribing to messages

The subscription modes vary by consumption mode.

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

Parameter	Description
topic_name	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%unique name of topic`, such as MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: No need to concatenate here, just fill in the Topic name.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Step 4. View consumption details

After sending the completion message, you will get a message ID. You can query the just-sent message, as well as its details and trace, in the console under **Message Query** > **Comprehensive Query**.

Note:

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

Sending and Receiving Sequential Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Directions

Step 1. Install the Java dependency library

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

Note:

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

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

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

Step 2. Produce messages

Creating a message producer

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
accessKey	Role key, copied from the AccessKey column on the Role Management page in the console.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.

Sending a message

Globally sequential message

This process is the same as that of general messages.

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

Partitioned sequential message

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

Step 3. Consume messages

Creating a consumer

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Copy the name of group from the Group Management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%group name`, such as MQ_INSTxxx_aaa%GroupTest. 4.x generic cluster/5.x cluster: No need to concatenate here, just fill in the Group name.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.

accessKey	Role key, copied from the AccessKey column on the Role Management page in the console.

Subscribing to messages

The subscription modes vary by consumption mode.

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

Parameter	Description
topic_name	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%topic name`, such as MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: No need to concatenate here, just fill in the Topic name.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Step 4. View consumption details

After sending the completion message, you will get a message ID. You can query the just-sent message, as well as the message details and path, etc., on the console **message query > comprehensive query** page.

Note:

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

Sending and Receiving Transactional Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Directions

Step 1. Install the Java dependent library

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

Note:

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

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

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

```
    </dependency>
```

Step 2. Produce messages

Implementing TransactionListener

```
public class TransactionListenerImpl implements TransactionListener {

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

}
```

Creating a message producer

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
accessKey	Role key, copied from the AccessKey column on the Role Management page in the console.

secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.

Sending a message

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

Step 3. Consume messages

Creating a consumer

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Copy the name of group from the Group Management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%group name`, such as MQ_INSTxxx_aaa%GroupTest. 4.x generic cluster/5.x cluster: No need to concatenate here, just fill in the Group name.

nameserver	Get the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.

Subscribing to messages

The subscription modes vary by consumption mode.

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

Step 4. View consumption details

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

Note:

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

Sending and Receiving Filtered Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Tag-based option

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

Step 1. Produce messages

Sending messages

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

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

Step 2. Consume messages

Subscribing to messages

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

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

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
topic_name	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%topic name` , such as MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: No need to concatenate here, just fill in the Topic name.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. `tag1 || tag2 || tag3` means subscribing to multiple types of tags.

Note:

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

SQL expression-based option

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

Step 1. Produce messages

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

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

Step 2. Consume messages

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

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

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

//Subscribe to multiple properties
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND

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

Note:

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

Sending and Receiving Broadcast Messages

Last updated：2025-07-15 18:32:07

Overview

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

Prerequisites

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

Directions

Step 1. Install the Java dependent library

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

Note:

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

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

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

Step 2. Produce messages

Creating a message producer

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
accessKey	Role key, copied from the AccessKey column on the Role Management page in the console.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.

Sending a message

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

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

Step 3. Consume messages

Creating a consumer

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Copy the name of group from the Group Management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format is `namespace full name%group name` , such as MQ_INSTxxx_aaa%GroupTest. 4.x generic cluster: no need to concatenate here, just fill in the Group name.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.

Subscribing to messages

This process requires setting consumption mode.

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

Parameter	Description

topic_name	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format is `namespace full name%topic name`, such as MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster: no need to concatenate here, just fill in the Topic name.
"*"	If the subscription expression is null or *, it means subscribing to all. It also supports "tag1 || tag2 || tag3" markers to subscribe to multiple types of tags.

Step 4. View consumption details

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

Note:

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

SDK for C++

Last updated：2025-07-15 18:32:08

Overview

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

Prerequisites

You have installed GCC.
You have downloaded the demo.

Directions

1. Prepare the environment.
1.1 Install RocketMQ-Client-CPP in the client environment as instructed in the official documentation. **The master branch is recommended**.
1.2 Import the header files and dynamic libraries related to RocketMQ-Client-CPP to the project.
2. Instantiate the message producer.

```
// Set the producer group name
DefaultMQProducer producer(groupName);
// Set the service access address
producer.setNamesrvAddr(nameserver);
// Set user permissions
producer.setSessionCredentials(
    accessKey,  // Role token
    secretKey, // Role name
    "");
// Set the full namespace name
producer.setNameSpace(namespace);
// Make sure all parameters are configured before the start
producer.start();
```

Note：
The following parameters require logging in to the TDMQ RocketMQ Console to retrieve.

Parameter	Description

groupName	Copy the producer group name from the Group Management page in the console.
nameserver	Obtain the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.
namespace	Namespace name, copied from the Namespace page in the console. If you use a 4.x generic cluster or 5.x cluster, fill in the Cluster ID here.

3. Send a message.

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

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

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

4. Release the resource.

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

5. Initialize the consumer.

```
// Listen on messages
   class ExampleMessageListener : public MessageListenerConcurrently {
   public:
       ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
           for (auto item = msgs.begin(); item != msgs.end(); item++) {
               // Business
               std::cout << "Received Message Topic:" << item->getTopic() << ",
MsgId:" << item->getMsgId() << ", TAGS:"
                         << item->getTags() << ", KEYS:" << item->getKeys() <<
", Body:" << item->getBody() << std::endl;
           }
           // Return CONSUME_SUCCESS if the consumption is successful
           return CONSUME_SUCCESS;
           // Return RECONSUME_LATER if the consumption failed. The message
will be consumed again.
           // return RECONSUME_LATER;
       }
   };

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

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to retrieve.

Parameter	Description
groupName	Copy the consumer group name from the Group Management page in the console.
nameserver	Obtain the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.
namespace	Namespace name, which can be copied on the **Namespace** page in the console.
topicName	Topic name, which can be copied on the **Topic** page in the console.
TAGS	A parameter used to set the message tag.

6. Release the resource.

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

7. View consumption details. After sending the completion message, you will get a message ID (messageID). You can query the just-sent message on the **message query** > **comprehensive query** page in the console, as well as the message details and trace information.

To send any delayed message, set the __STARTDELIVERTIME attribute.

```
        MQMessage msg("console topic", "tag", "Delay Message.");
        chrono::system_clock::duration d = chrono::system_clock::now().time_since_e
        chrono::milliseconds mil = chrono::duration_cast<chrono::milliseconds>(d);
        //Scheduled delayed message in milliseconds (ms), delivered at the specifie
        //Set the delay or scheduled time, such as the current time with a 30-secon
        long expectTime = mil.count() + 30000;
```

```
        msg.setProperty("__STARTDELIVERTIME", to_string(expectTime));
```

Note:

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

SDK for Go

Last updated：2025-07-15 18:32:08

Overview

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

Prerequisites

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

Directions

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

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

2. Create a producer in the corresponding method. If you need to send general messages, modify the corresponding parameters in the `syncSendMessage.go` file.
Delayed messages currently support delays of arbitrary precision without being subject to the delay level.
General Message
Delayed message

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

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

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

```
                os.Exit(1)
        }

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

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

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
secretKey	Role name, copy from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copy from the AccessKey column on the Role Management page in the console.
nameSpace	Namespace name, which can be copied on the **Namespace** page in the console.
serverAddress	Obtain the cluster access address from the access information module on the console cluster basic information page. (Note: add http:// or https:// before the access address, otherwise it cannot be parsed).

groupName	Copy the producer group name from the Group Management page on the console.

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

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

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

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

Release the resource.

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

Note:

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

4. Create a consumer.

```
// Service access address (Note: Add “http://” or “https://” before the access addr
   var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:
   // Authorize the role name
   var secretKey = "admin"
   // Authorize the role token
   var accessKey = "eyJrZXlJZC...."
   // Full namespace name
   var nameSpace = "rocketmq-xxx|namespace_go"
   // Producer group name
   var groupName = "group11"
   // Create a consumer
   c, err := rocketmq.NewPushConsumer(
       // Set the consumer group
       consumer.WithGroupName(groupName),
       // Set the service address
       consumer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddr
       // Set ACL permissions
       consumer.WithCredentials(primitive.Credentials{
           SecretKey: secretKey,
           AccessKey: accessKey,
       }),
       // Set the namespace name
       consumer.WithNamespace(nameSpace),
       // Set consumption from the start offset
       consumer.WithConsumeFromWhere(consumer.ConsumeFromFirstOffset),
       // Set the consumption mode (cluster consumption by default)
       consumer.WithConsumerModel(consumer.Clustering),

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
secretKey	Role name, copy from the SecretKey column on the Role Management page in the console.

accessKey	Role token, copy from the AccessKey column on the Role Management page in the console.
nameSpace	Copy the namespace name from the Namespace page in the console. If you use a 4.x generic cluster or 5.x cluster, fill in the Cluster ID here.
serverAddress	Obtain the cluster access address from the access information module on the console cluster basic information page. (Note: add http:// or https:// before the access address, otherwise it cannot be parsed).
groupName	Copy the producer group name from the Group Management page on the console.

5. Consume a message.

```
// Topic name
    var topicName = "topic1"
    // Set the tag of messages that are subscribed to
    selector := consumer.MessageSelector{
        Type:       consumer.TAG,
        Expression: "TagA || TagC",
    }
    // Set the delay level of consumption retry. A total of 18 levels can be set. Be
    // 1   2   3    4    5   6   7   8   9   10  11  12  13  14   15   16   17  18
    // 1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m, 30m, 1h, 2h
    delayLevel := 1
    err = c.Subscribe(topicName, selector, func(ctx context.Context,
                                                                msgs ...*primitive
        fmt.Printf("subscribe callback len: %d \\n", len(msgs))
        // Set the delay level for the next consumption
        concurrentCtx, _ := primitive.GetConcurrentlyCtx(ctx)
        concurrentCtx.DelayLevelWhenNextConsume = delayLevel // only run when return

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

```
        return consumer.ConsumeRetryLater, nil
    })
    if err != nil {
        fmt.Println(err.Error())
    }
```

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

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

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

7. View consumption details. After sending the completion message, you will get a message ID (messageID). You can query the just-sent message on the console in **Message Query,** as well as the message details and path information.

Note

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

SDK for Python

Last updated：2025-07-15 18:32:08

Overview

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

Prerequisites

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

Directions

Step 1. Prepare the environment

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

Note:
Currently, the Python client only supports Linux and macOS operating systems. It doesn't support Windows systems. When using Python SDK, note the underlying chip architecture type (x86 or ARM) supported by the installed Python. For example, using a Python version with `'64bit','ELF'` (x86_64 architecture) may cause errors on macOS systems with ARM chips.

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

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

Step 2. Produce messages

Create, compile, and run a message production program.

```
from rocketmq.client import Producer, Message
```

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

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

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
groupName	Copy the producer group name from the Group Management page on the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%group name` , such as MQ_INSTxxx_aaa%GroupTest. 4.x generic cluster/5.x cluster: No need to concatenate here, just fill in the Group name.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.

namespace	Copy the namespace name from the namespace page in the console. If you use a 4.x generic cluster or 5.x cluster, fill in the Cluster ID here.
topicName	Topic name, which can be copied on the **Topic** management page in the console.
TAGS	A parameter used to set the message tag.
KEYS	A parameter used to set the message key.

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

Step 3. Consume messages

Create, compile, and run a message consumption program.

```
import time

   from rocketmq.client import PushConsumer, ConsumeStatus


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


   # Initialize the consumer and set the consumer group information
   consumer = PushConsumer(groupName)
   # Set the service address
   consumer.set_name_server_address(nameserver)
   # Set permissions (role name and token)
   consumer.set_session_credentials(
       accessKey,        # Role token
       secretKey,   # Role name
       ''
   )
```

```
    # Subscribe to a topic
    consumer.subscribe(topicName, callback, TAGS)
    print(' [Consumer] Waiting for messages.')
    # Start the consumer
    consumer.start()

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

Note：

The following parameters require logging in to the TDMQ RocketMQ Console to obtain.

Parameter	Description
namespace	Copy the namespace name from the namespace page in the console. If you use a 4.x generic cluster or 5.x cluster, fill in the Cluster ID here.
groupName	Copy the name of consumer group from the Group Management page on the console.
nameserver	Get the cluster access address from the access information module on the console cluster basic information page.
secretKey	Role name, copied from the SecretKey column on the Role Management page in the console.
accessKey	Role token, copied from the AccessKey column on the Role Management page in the console.
topicName	Copy the Topic name from the Topic management page in the console.

Step 4. View consumption details

After sending the completion message, you will get a message ID (messageID). You can query the just-sent message, as well as the message details and trace, on the console **Message Query** > **Comprehensive Query** page.

Note

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

C# SDK

Last updated：2025-07-15 18:32:08

Scenarios

In this document, C# SDK call is used as an example to introduce the operation process of sending and receiving messages through the open source SDK, which helps you better understand the whole process of sending and receiving messages.

Prerequisites

Complete resource creation and preparation.
Install DotNet
Download the demo.

Operation Steps

Step 1: Setting Up Environment

Create a project in your own IDE environment, then install the NewLife.RocketMQ dependency.

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

Step 2: Producing Messages

Create, compile, and run a message production program.

```
//producer
var mq = new Producer
{
    Topic = "TopicTest1", //For non-general clusters, the full topic must be concat
    NameServerAddress = "127.0.0.1:9876",// Fill in the Tencent Cloud page
    Log = XTrace.Log,
    AclOptions = new AclOptions
    {
        AccessKey = "Page ak",
        SecretKey = "Page sk",
    },
};
```

```
mq.Start();
for (var i = 0; i < 10; i++)
{
    var str = "produce messages test" + i;
    var sr = mq.Publish(str, "TagA");
}
```

Note:

Note: The following parameters require logging in to the TDMQ RocketMQ Console to retrieve.

Parameter	Description
Topic	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%topic name` , such as MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: No need to concatenate, just fill in the Topic name.
NameServerAddress	Obtain the cluster access address from the access information module on the console cluster basic information page.
SecretKey	Role name, copy from the SecretKey column on the Role Management page on the console.
AccessKey	Role key, copy from the AccessKey column on the Role Management page on the console.
TAGS	It is used to set the message tag.

Step 3: Consuming Messages

Create, compile, and run a message consumption program.

```
//consumer
var consumer = new Consumer
{
    Topic = "TopicTest1", //For non-general clusters, the full topic must be
concatenated, such as MQ_INSTxxx_aaa%TopicTest
    Group = "test", //For non-general clusters, the full Group must be
concatenated, such as MQ_INSTxxx_aaa%GroupTest
    NameServerAddress = "127.0.0.1:9876",
    FromLastOffset = true,
    SkipOverStoredMsgCount = 0,
    BatchSize = 20,
```

```
    Log = XTrace.Log,
    AclOptions = new AclOptions
    {
        AccessKey = "Page ak",
        SecretKey = "Page sk",
    },
};
consumer.OnConsume = (q, ms) =>
{
    XTrace.WriteLine("[{0}@{1}] receive message [{2}]", q.BrokerName,
q.QueueId, ms.Length);
    foreach (var item in ms.ToList())
    {
        XTrace.WriteLine($"Message {item.Keys}, send time
{item.BornTimestamp.ToDateTime()}, content {item.Body.ToStr()}");
    }
    return true;
};
consumer.Start();
```

Note:

Note: The following parameters require logging in to the TDMQ RocketMQ Console to retrieve.

Parameter	Description
Topic	Copy the Topic name from the Topic management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%topic name`, such as MQ_INSTxxx_aaa%TopicTest. 4.x generic cluster/5.x cluster: No need to concatenate, just fill in the Topic name.
Group	Copy the name of group from the Group Management page in the console. 4.x virtual cluster/dedicated cluster: Concatenate the namespace name here, format `namespace full name%group name`, such as MQ_INSTxxx_aaa%GroupTest. 4.x generic cluster/5.x cluster: No need to concatenate, just fill in the Group name.
NameServerAddress	Obtain the cluster access address from the access information module on the console cluster basic information page.
SecretKey	Role name, copy from the SecretKey column on the Role Management page on the console.
AccessKey	Role key, copy from the AccessKey column on the Role Management page on the console.

Step 4: Viewing Message Details

After sending the completion message, you will get a message ID. You can query the just-sent message, as well as its details and path, in the console under **Message Query** > **Comprehensive Query**.