ALIBABA CLOUD

阿里云

消息队列 MQ

SDK参考

文档版本：20220712

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.SDK参考概述

本文列举了
消息队列Rocket MQ版
所支持的协议以及相关的多语言SDK。

TCP协议

> **注意** 社区版SDK仅在迁移开源Rocket MQ上云且不希望修改代码时使用，其他场景推荐您使用阿里云
> 消息队列Rocket MQ版
> 提供的商业版SDK进行接入。和社区版SDK相比，商业版SDK提供了更加丰富的功能特性并具有更高的稳定性保障。

- **商业版TCP协议SDK**

Java SDK

- 版本说明
- 准备环境
- 代码示例

C/C++ SDK

- 版本说明
- 准备环境
- 代码示例

.NET SDK

- 版本说明
- 准备环境
- 代码示例

- **社区版TCP协议SDK**

Java SDK

- 接入说明
- 准备环境
- 代码示例

C++ SDK

- 接入说明

- 准备环境
- 代码示例

HTTP协议

- **商业版HTTP协议SDK（多语言推荐）**

Java SDK

- 版本说明
- 准备环境
- 代码示例

PHP SDK

- 版本说明
- 准备环境
- 代码示例

Go SDK

- 版本说明
- 准备环境
- 代码示例

Python SDK

- 版本说明
- 准备环境
- 代码示例

Node.js SDK

- 版本说明
- 准备环境
- 代码示例

C# SDK

- 版本说明
- 准备环境
- 代码示例

C++ SDK

- 版本说明

2.商业版TCP协议SDK（推荐）

2.1. Java SDK

2.1.1. 版本说明

本文介绍Java SDK的版本信息，包含下载链接、发布时间、更新点等，以便您按需获取适用的Java SDK收发消息。

> **注意**
>
> - JDK 1.8适用于所有Java ons-client版本，JDK 1.6仅支持ons-client v1.8.4.Final及之前的版本使用。为避免升级SDK版本时出现JDK兼容性问题，建议您下载JDK 1.8版本。
> - 目前仅西南1（成都）、华北1（青岛）和华南1（深圳）地域支持将客户端升级为2.x.x.Final版本，其他地域请勿将SDK升级到Java SDK 2.x.x Final版本，否则将无法访问消息队列RocketMQ版服务。如有特殊需求，请提交工单咨询。
> - 获取Maven依赖，请参见准备环境。

2.0.2.Final

发布时间	发布内容	使用范围	下载
2022-06-16	**问题修复** • 修复消息发送时，有小概率触发死锁的问题。	目前仅如下地域支该版本SDK，其他地域待开放。 • 西南1（成都） • 华北1（青岛） • 华南1（深圳）	ons-client-2.0.2.Final

2.0.1.Final

发布时间	发布内容	使用范围	下载
2021-11-29	**功能优化** • 补充消息轨迹数据。	目前仅如下地域支该版本SDK，其他地域待开放。 • 西南1（成都） • 华北1（青岛） • 华南1（深圳）	ons-client-2.0.1.Final

2.0.0.Final

发布时间	发布内容	使用范围	下载

发布时间	发布内容	使用范围	下载
2021-10-18	**功能优化** • 负载均衡：以消息为粒度进行负载，负载更加均衡。 • 公网接入点：支持TCP协议公网接入点访问。 • Dashboard：新增消息堆积、消息各环节耗时、成功率等相关指标。 • 消息轨迹：新增定时和延时、事务消息及消费环节相关轨迹参数。 • 顺序消息：最大重试次数变更为16次。 • 事务消息：优化事务消息异常场景处理逻辑。 • 广播消费：支持定制消费者启动时的消费位点。 • Push消费：支持消费速度限流；消费线程数异常场景逻辑优化。 • 日志配置：日志默认路径变更；增加日志级别；增加对环境变量的支持。 • 客户端创建：异常场景逻辑优化。 **问题修复** • 修复RAM角色实现跨云账号STS授权场景下，updateCredential方法调用频率较高时，三元组（AccessKey ID、AccessKey Secret和STS Token）更新不具备原子性而导致的鉴权失败问题。 **说明** 具体变更内容，请参见Java SDK版本说明。	目前仅如下地域支该版本SDK，其他地域待开放。 • 西南1（成都） • 华北1（青岛） • 华南1（深圳）	ons-client-2.0.0.Final

1.8.8.5.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2022-05-16	**问题修复** • 修复RAM角色实现跨云账号STS授权场景下，updateCredential方法调用频率较高时，三元组（AccessKey ID、AccessKey Secret和STS Token）更新不具备原子性而导致的鉴权失败问题。 **说明** 本次修复只针对使用RAM角色授权的场景，若您自己更新三元组，还会出现该问题。	ons-client-1.8.8.5.Final	ons-client-ext-1.8.8.5.Final

1.8.8.3.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2022-01-18	**功能优化** • 修复服务端异常导致消费位点跳过的问题。 • 修复客户端消费超时时间单位错误的问题。	ons-client-1.8.8.3.Final	ons-client-ext-1.8.8.3.Final

1.8.8.1.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2021-08-24	**功能优化** • 修复顺序消息重试问题。 • 优化特殊场景下客户端发送重试消息分裂出多条重复消息的问题。	ons-client-1.8.8.1.Final	ons-client-ext-1.8.8.1.Final

1.8.8.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2021-04-02	**功能优化** • 优化了心跳发送逻辑。 • 修复了SDK占用内存过多的问题。 • 修复了消息消费失败发回（sendMessageBack）的问题。 • 修复了客户端Topic级别消息缓存限制未生效的问题。	ons-client-1.8.8.Final	ons-client-ext-1.8.8.Final

1.8.7.4.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2021-02-08	**功能优化** • 修复了特殊场景下顺序消息消费延迟的问题。 • 修复了消息发送端探活导致的端口占用问题。	ons-client-1.8.7.4.Final	ons-client-ext-1.8.7.4.Final

说明 若您使用1.8.7.1.Final及之后版本的Java SDK消费顺序消息，建议您将SDK客户端升级到最新版本。

1.8.7.3.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2021-01-05	**新特性** • 消息获取方式为Push模式时，支持批量消费功能。详细信息，请参见批量消费功能描述和批量消费示例代码。 **功能优化** • 修复了在网络等异常情况下可能会导致消息大量重复的问题。	ons-client-1.8.7.3.Final	ons-client-ext-1.8.7.3.Final

1.8.7.1.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2020-07-09	**新特性** • 支持顺序消息2.0（仅限于铂金版），可以支持全局顺序消息高可用、可扩展，分区顺序二级散列。 • 统一有无命名空间实例的使用方式。 **功能优化** • 优化失败重试逻辑以及服务端熔断机制，进一步提高服务可用性，降低因为服务升级、ECS宕机、VIP抖动造成的业务影响。	ons-client-1.8.7.1.Final	ons-client-ext-1.8.7.1.Final

1.8.5.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2020-05-10	**新特性** • 增加Pull Consumer。	ons-client-1.8.5.Final	ons-client-ext-1.8.5.Final

注意 如需使用Pull Consumer，请确保您的消息队列RocketMQ版实例为企业铂金版。

1.8.4.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2019-09-27	**新特性** • 支持1.6 JDK。 • 支持异步发送重试。 • 支持同步发送brokerbusy重试。	ons-client-1.8.4.Final	ons-client-ext-1.8.4.Final

ons-client v1.8.0.Final

发布时间	发布内容	下载	下载（含Exactly-Once投递语义）
2019-02-21	**问题修复** • 修复自动重试逻辑（默认重试三次），该逻辑适用于消息从生产端同步发送至实例化后新建实例下的Topic失败的场景。	ons-client-1.8.0.Final	ons-client-ext-1.8.0.Final

1.7.8.Final

发布时间	发布内容	下载
2018-07-06	**新特性** • 支持动态更新STS Token。 **问题修复** • 修复日志默认大小为1 GB的问题，修改后的日志默认大小为64 MB。 • 修复日志打印双份的问题。	ons-client-1.7.8.Final

更多历史版本

1.7.7.Final

发布时间	发布内容	下载
2018-04-25	**问题修复** • 同一个进程内初始化多个Consumer或Producer实例的情况下，消息轨迹发送失败的问题，该问题在1.7.5.Final和1.7.6.Final版本中存在，建议进行升级。	ons-client-1.7.7.Final

1.7.6.Final

发布时间	发布内容	下载

发布时间	发布内容	下载
2018-04-04	**新特性** • 客户端兼容任意日志框架。 **问题修复** • 修复log4j2的支持问题。 • 修复client fetchNameserver shutdown问题。 • 升级Fastjson至1.2.48版本。	ons-client-1.7.6.Final

1.7.5.Final

发布时间	发布内容	下载
2018-03-23	**问题修复** • 修复引入了阿里巴巴内部依赖的问题。	ons-client-1.7.5.Final

1.7.4.Final

发布时间	发布内容	下载
2018-03-02	**新特性** • 支持STS Token接入。 • 轨迹消息发送区分优先级，默认优先发送至本集群Broker。 **问题修复** • 修复JDK 1.6兼容问题。	ons-client-1.7.4.Final

1.7.2.Final

发布时间	发布内容	下载
2018-01-25	**新特性** • 企业铂金版支持传输层加密配置，在AccessKeyId和AccessKeySecret签名链路上进行传输层加密，将具有更高的安全性。 • 企业铂金版支持消费端SQL属性过滤功能，加强消息订阅的效率。 • 客户端自动感知NameSrv的变化，方便进行运维切换，客户端将具有更高的可用性。 • 客户端连接时向服务端上报精确的版本信息。	ons-client-1.7.2.Final

1.7.1.Final

发布时间	发布内容	下载

发布时间	发布内容	下载
2017-12-19	**新特性** • 异步发送接口，用户可配置自定义回调线程池。 • 异步发送接口，新增JVM -D参数，用于控制公共线程池的线程数量：Dclient.callback.executor.thread.nums=10。 **问题修复** • 修复客户端消息消费超时SendBack时未扣除缓存计数。 • 修复客户端异步信号量过早释放问题。	ons-client-1.7.1.Final

1.7.0.Final

发布时间	发布内容	下载
2017-10-23	**新特性** • 调整客户端消息缓存策略，考虑消息条数与缓存大小两个维度。 **功能优化** • 优化客户端内置轨迹模块的ProducerName，不同的用户使用不同的值。 **问题修复** • 修复客户端Trace线程阻止客户端正常退出的问题。 • 修复消息轨迹ShutDownHook可能重复创建的问题。	ons-client-1.7.0.Final

1.6.1.Final

发布时间	发布内容	下载
2017-08-31	**功能优化** • 为所有的客户端API添加了详细的Javadoc。 • 优化获取客户端地址的方式，不依赖/etc/hosts中的hostname配置。	ons-client-1.6.1.Final

1.6.0.Final

发布时间	发布内容	下载

发布时间	发布内容	下载
2017-07-31	**新特性** • 客户端在源码级别进行Shade，保证Debug的正确性。 • 客户端暴露BornHost、BornTimestamp消息属性。 • 新增BatchConsumer接口，允许用户以批量的方式消费消息。 • 新增顺序消息、BatchConsumer与Spring集成的Demo。 **功能优化** • 针对分区有序消息，将Sharding Key放入到消息结构中。 • 消息属性设置支持Int型的Value值。	ons-client-1.6.0.Final

后续步骤

准备环境

2.1.2. Demo工程

针对初次接触
消息队列RocketMQ版
的工程师，本文以TCP协议下的Java SDK为例，提供操作示例帮助您从零开始搭建
消息队列RocketMQ版
测试工程。Demo工程包含普通消息、顺序消息、事务消息、定时和延时消息的测试代码，以及相关Spring的配置。

前提条件

- 安装IDE。
  您可以使用IntelliJ IDEA或者Eclipse，本文以IntelliJ IDEA为例。
  请下载IntelliJ IDEA Ultimate版本，并参见IntelliJ IDEA说明进行安装。更多信息，请参见下载地址。
- 下载Demo工程。
  下载到本地并解压后即可看到本地新增了*rocketmq-demo-master*文件夹，该文件夹包括纯Java、Spring以及Spring Boot的示例代码。更多信息，请参见rocketmq-demo。
- 下载安装JDK。更多信息，请参见JDK下载地址。

配置Demo工程

1. 将Demo工程文件导入IntelliJ IDEA。
2. 创建资源。

   您需要先到
   消息队列RocketMQ版
   控制台创建所需资源，包括实例、Topic、Group ID（GID），以及鉴权需要的AccessKey ID（AK）和AccessKey Secret（SK）。
   更多详细信息和操作指导，请参见创建资源。
3. 配置Demo。

   您需将在步骤2中创建好的资源信息配置到 `MqConfig` 类和*common.xml*文件中。

i. 按以下说明配置 `MqConfig` 类。

```
public static final String TOPIC = "您已创建的Topic";
public static final String GROUP_ID = "您已创建的Group ID";
public static final String ORDER_TOPIC = "您已创建的用于收发顺序消息的Topic";
public static final String ORDER_GROUP_ID = "您已创建的用于收发顺序消息的Group ID";
public static final String ACCESS_KEY = "您的阿里云账号的AccessKey ID";
public static final String SECRET_KEY = "您的阿里云账号的AccessKey Secret";
public static final String TAG = "您自定义的消息Tag属性";
public static final String NAMESRV_ADDR = "您已创建的
消息队列RocketMQ版实例的TCP接入点，可在
消息队列RocketMQ版控制台实例详情页面的接入点区域查看";
```

> 说明
>
> - 创建AccessKey（包括AccessKey ID和AccessKey Secret）的具体步骤，请参见创建AccessKey。
> - 如果RAM用户拥有该Topic的权限以及自己的AccessKey，那么也可以使用RAM用户的AccessKey。
> - 参数与接口的更多信息，请参见接口和参数说明。

ii. 配置*common.xml*。

```
 <props>
 <prop key="AccessKey">XXX</prop> <!-- 使用前请修改这些资源信息 -->
 <prop key="SecretKey">XXX</prop>
 <prop key="GROUP_ID">XXX</prop>
 <prop key="Topic">XXX</prop>
 <prop key="NAMESRV_ADDR">XXX</prop>
</props>
```

以Main方式运行Demo

1. 发送消息。
   - 发送普通消息：
     - 以纯Java方式发送普通消息：运行 `SimpleMQProducer` 类。
     - 以Spring方式发送普通消息：运行 `ProducerClient` 类。
     - 以Spring Boot方式发送普通消息：运行 `ProducerClient` 类。
   - 发送事务消息：
     - 以纯Java方式发送事务消息：运行 `SimpleTransactionProducer` 类。`LocalTransactionCheckerImpl` 类为本地事务check接口类，用于校验事务。更多信息，请参见收发事务消息。
     - 以Spring方式发送事务消息：运行 `TransactionProducerClient` 类。
     - 以Spring Boot方式发送事务消息：运行 `TransactionProducerClient` 类。
   - 发送顺序消息：
     - 以纯Java方式发送顺序消息：运行 `SimpleOrderProducer` 类。

- 以Spring方式发送顺序消息：运行 `OrderProducerClient` 类。
- 以Spring Boot方式发送顺序消息：运行 `OrderProducerClient` 类。

此方式下，消息发布和消费都按顺序进行。更多信息，请参见收发顺序消息。

- 发送定时和延时消息：运行 `MQTimerProducer` 类发送消息。延时3秒后投递。
  您也可以指定一个精确的投递时间，最长定时时间为40天。更多信息，请参见收发定时消息。

  在
  消息队列RocketMQ版
  控制台，按Topic查询消息，可以看到消息已经发送至Topic。

2. 接收消息。
   - 接收普通消息：
     - 以纯Java方式接收普通消息：运行 `SimpleMQConsumer` 类。
     - 以Spring方式接收普通消息：运行 `ConsumerClient` 类。
     - 以Spring Boot方式接收普通消息：运行 `ConsumerClient` 类。
   - 接收事务消息：
     - 以纯Java方式接收事务消息：运行 `SimpleMQConsumer` 类。
     - 以Spring方式接收事务消息：运行 `ConsumerClient` 类。
     - 以Spring Boot方式接收事务消息：运行 `ConsumerClient` 类。
   - 接收顺序消息：
     - 以纯Java方式接收顺序消息：运行 `SimpleOrderConsumer` 类。
     - 以Spring方式接收顺序消息：运行 `OrderConsumerClient` 类。
     - 以Spring Boot方式接收顺序消息：运行 `OrderConsumerClient` 类。
   - 接收定时和延时消息：运行 `SimpleMQConsumer` 类。

     > **说明** Spring和Spring Boot框架暂不支持收发定时和延时消息。

     可以看到消息被接收打印的日志。因为有初始化，所以需等待几秒，在生产环境中不会经常初始化。

结果验证：在

消息队列RocketMQ版
控制台，查看消费者状态，可以看到启动的消费端已经在线，并且订阅关系一致。

更多信息

- Spring集成
- 使用Exactly-Once投递语义收发消息
- 发送普通消息（三种方式）
- 发送消息（多线程）

2.1.3. 接口和参数说明

消息队列RocketMQ版
提供Java SDK实现消息发送与订阅，订阅方可通过Push或Pull的方式从

消息队列RocketMQ版
获取消息。本文介绍消息发送和订阅的接口和参数说明。

背景信息

消息队列RocketMQ版
支持以下两种消息获取方式：

- Push：消息由
  消息队列RocketMQ版
  推送至Consumer。Push方式下，
  消息队列RocketMQ版
  还支持批量消费功能，可以将消息统一批量推送至Consumer进行消费。更多信息，请参见批量消费。
- Pull：消息由Consumer主动从
  消息队列RocketMQ版
  拉取。

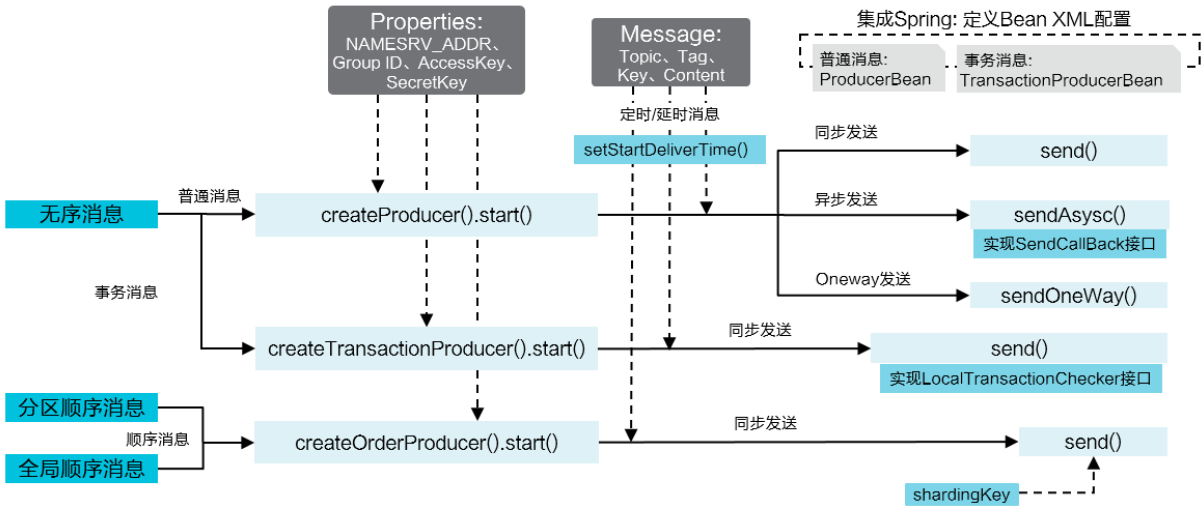
Pull Consumer提供了更多接收消息的选择。相比于Push Consumer，您可以使用Pull Consumer更加自由地控制消息拉取。

> **注意** 如需使用Pull Consumer，请确保您的
> 消息队列RocketMQ版
> 实例为企业铂金版。

通用参数

参数名	参数说明
NAMESRV_ADDR	设置TCP协议接入点，从
消息队列RocketMQ版
控制台的**实例详情**页面获取。
AccessKey	您在阿里云账号管理控制台中创建的AccessKey ID，用于身份认证。
SecretKey	您在阿里云账号管理控制台中创建的AccessKey Secret，用于身份认证。
OnsChannel	用户渠道，默认值为：ALIYUN，聚石塔用户为：CLOUD。

消息发送接口

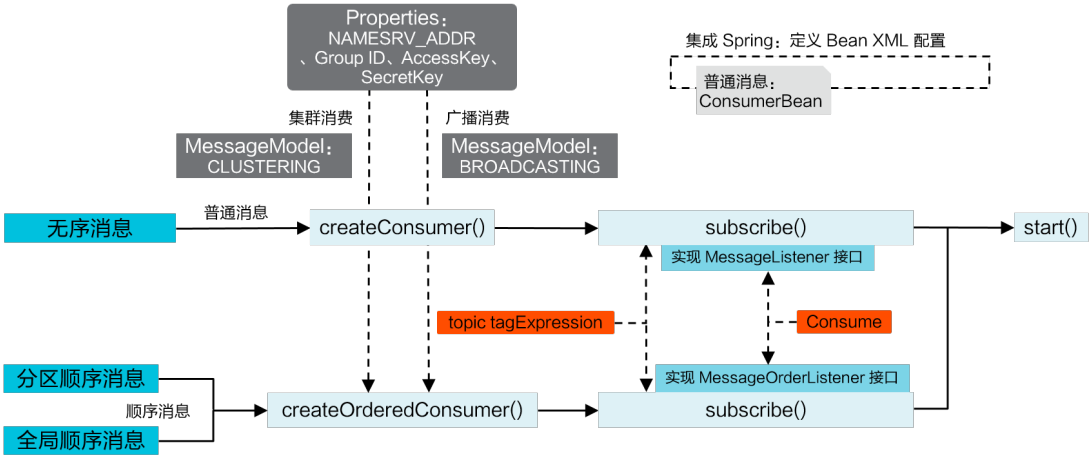


消息发送参数

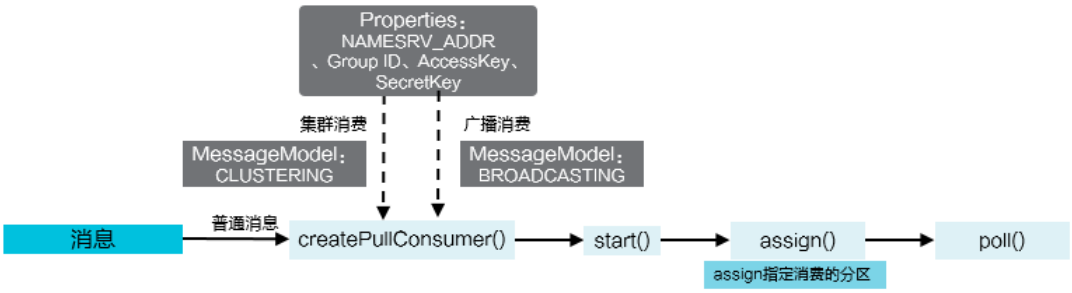
参数名	参数说明
SendMsgTimeoutMillis	设置消息发送的超时时间，默认值：3000；单位：毫秒。
CheckImmunityTimeInSeconds（事务消息）	设置事务消息第一次回查的最快时间，单位：秒。
shardingKey（顺序消息）	顺序消息中用来计算不同分区的值。

消息订阅接口

Push方式接口



Pull方式接口



Pull方式的接口说明如下所示。

Pull Consumer接口说明 ❯

消息订阅参数

通用参数

参数	说明
GROUP_ID	您在 消息队列RocketMQ版 控制台上创建的Group ID，更多信息，请参见基本概念。
MessageModel	设置Consumer实例的消费模式，取值说明如下： • CLUSTERING（默认值）：表示集群消费。 • BROADCASTING：表示广播消费。
ConsumeThreadNums	设置Consumer实例的消费线程数，默认值：20。
MaxReconsumeTimes	设置消息消费失败的最大重试次数，默认值：16。
ConsumeTimeout	设置每条消息消费的最大超时时间，超过设置时间则被视为消费失败，等下次重新投递再次消费。每个业务需要设置一个合理的值，默认值：15，单位：分钟。
suspendTimeMillis（顺序消息）	只适用于顺序消息，设置消息消费失败的重试间隔时间。
maxCachedMessageAmount	消费者客户端本地的最大缓存消息数量，取值范围为[100,50000]，默认值：5000，单位：条。 该参数在客户端级别生效，限定额会平均分配到订阅的Topic上。例如最大缓存数为1000条，某消费者客户端订阅了2个Topic，则每个Topic将限制缓存500条消息。考虑到消费者客户端批量拉取消息的场景，实际最大缓存量会略大于限制值。 建议取值：若某个消费者客户端每秒能处理N条消息，则该参数建议设置为2 * N条。 **注意** 请合理取值，设置过大可能会引起客户端OOM。
maxCachedMessageSizeInMiB	客户端本地的最大缓存消息大小，取值范围：16 MB～2048 MB，默认值：512 MB。

Push方式特有参数（批量消费）

参数	说明
ConsumeMessageBatchMaxSize	批量消费的最大消息数量，缓存的消息数量达到设置的参数值， 消息队列RocketMQ版 会将缓存的消息统一推送给消费者进行批量消费。默认值：32，取值范围：1~1024。
BatchConsumeMaxAwaitDurationInSeconds	批量消费的等待时长，等待时长达到参数设置的值， 消息队列RocketMQ版 会将缓存的消息统一推送给消费者进行批量消费。默认为：0，取值范围：0～450，单位：秒。

Pull方式特有参数

参数	说明
maxCachedMessageSizeInMiB	Consumer单个分区允许在客户端中缓存的最大消息容量，默认值：100 MiB，取值范围：16 MiB~2048 MiB。 **注意** 请合理取值，设置过大可能会引起客户端OOM。
autoCommit	是否允许消费位点自动提交，默认为true。
autoCommitIntervalMillis	消费位点自动提交间隔，默认值：5，单位：秒。
pollTimeoutMillis	每次拉取消息超时时间，默认值：5，单位：秒。

分区和位点的详细说明，请参见基本概念。

更多信息

消息收发示例代码的更多信息，请参见以下文档：

- 发送普通消息（三种方式）
- 收发顺序消息
- 收发定时消息
- 收发延时消息
- 收发事务消息
- 订阅消息

2.1.4. 准备环境

在运行Java代码收发消息前，您需按照本文提供的步骤来准备环境。

操作步骤

1. 通过下面两种方式可以引入依赖（任选一种）：
   - Maven方式引入依赖：

```
<dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>ons-client</artifactId>
    <!--以下版本号请替换为Java SDK的最新版本号-->
    <version>1.8.8.1.Final</version>
</dependency>
```

> **说明** 获取SDK的版本信息，请参见版本说明。

- 下载依赖JAR包：Java SDK最新版本的下载链接，请参见Java SDK的版本说明。

2. 代码里涉及到的资源信息，例如Topic和Group ID等，需要到控制台上创建。

   目前支持两种协议资源的创建：

   - HTTP协议：请参见创建资源。
   - TCP协议：请参见创建资源。

后续步骤

日志配置

更多信息

- Spring集成
- 使用Exactly-Once投递语义收发消息
- 发送普通消息（三种方式）
- 发送消息（多线程）

2.1.5. 日志配置

客户端日志用于记录
消息队列RocketMQ版
客户端运行过程中的异常，帮助您快速定位和修复问题。本文介绍如何为客户端开启日志打印功能、客户端日志的配置项说明以及如何自定义配置客户端日志。

开启客户端日志打印功能

消息队列RocketMQ版
的TCP Java SDK基于SLF4J接口编程。

- **Java SDK 1.7.8.Final及以上版本**：默认支持，无需设置
  消息队列RocketMQ版
  的Java SDK 1.7.8.Final已内置了日志实现，您无需在客户端应用中添加日志实现依赖即可打印
  消息队列RocketMQ版
  的客户端日志。
- **针对Java SDK 1.7.8.Final以下版本**：添加日志实现依赖
  消息队列RocketMQ版
  的Java SDK 1.7.8.Final以下的旧版本不支持Log4j2，只支持Log4j、Logback。您需要在 `pom.xml` 配置文件或者lib中添加对应的日志实现依赖来打印
  消息队列RocketMQ版
  客户端日志。示例代码如下：

- **方式一：依赖Log4j作为日志实现**

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>1.7.7</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.7</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- **方式二：依赖Logback作为日志实现**

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>1.1.2</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.1.2</version>
</dependency>
```

注意 应用中同时依赖Log4j和Logback的日志实现会造成日志冲突导致客户端日志打印混乱。请确保应用只依赖其中一个日志实现，您可以通过 `mvn clean dependency:tree | grep log` 命令进行排查。

客户端日志配置项

客户端日志的配置项及说明如下表所示。客户端启动后，会按照系统默认配置生成日志文件；您也可以自定义配置项，修改日志参数，具体操作，请参见本文中的自定义配置客户端日志。

配置项	系统默认配置	是否支持自定义配置	配置参数	自定义取值
日志保存路径	*/{user.home}/logs/ons.log*，其中*{user.home}*是指您启动当前Java进程的根目录。	是	`ons.client.logRoot`	可自定义为您需要将日志文件保存到本地的路径。 请确保您的应用进程有该路径的写权限，否则日志无法打印。

配置项	系统默认配置	是否支持自定义配置	配置参数	自定义取值
日志级别	INFO	是	`ons.client.logLevel`	取值如下： • ERROR • WARN • INFO • DEBUG
保存历史日志文件的最大个数	10个	是	`ons.client.logFileMaxIndex`	取值范围：1~100。若设置的值超出该范围或格式错误，则以系统默认值（10个）为准。
单个日志文件大小	64 MB	否	不涉及	不涉及

自定义配置客户端日志

> **注意** 若您需要自定义客户端的日志配置，请将Java SDK升级到1.2.5及以上版本。

- **配置方式**：在启动脚本或者IDE的VM options中，通过 `-D` 命令设置客户端日志配置项。
- **配置示例**：
  - **Linux系统配置示例**

```
-Dons.client.logRoot=/home/admin/logs -Dons.client.logLevel=WARN -Dons.client.logFileMaxIndex=20
```

  - **Windows系统配置示例**

```
-Dons.client.logRoot=D:\logs -Dons.client.logLevel=WARN -Dons.client.logFileMaxIndex=20
```

其中*/home/admin/*和*D:*仅为示例，请替换为您实际的系统目录。

2.1.6. Spring集成

本文介绍如何在Spring框架下用
消息队列RocketMQ版
收发消息。

背景信息

消息队列RocketMQ版
支持以下消息类型的生产者和消费者与Spring集成：

- 普通消息的生产者和消费者
- 事务消息的生产者和消费者
- 顺序消息的生产者和消费者

参数说明

与Spring集成中所需配置的参数如下所示。

参数	说明
GROUP_ID	您在控制台创建的Group ID，用于对消费者或生产者实例进行分类。更多信息，请参见基本概念。
AccessKey	阿里云身份验证AccessKey ID，在阿里云用户信息管理控制台获取。
SecretKey	阿里云身份验证AccessKey Secret，在阿里云用户信息管理控制台获取。
NAMESRV_ADDR	设置TCP接入域名，进入控制台的**实例详情**页面，单击**接入点**页签，在**TCP 协议客户端接入点**区域查看。
expression	消息过滤表达式，例如“Tag A||Tag B”，说明消费者订阅了带有Tag A和Tag B两种Tag的消息。更多信息，请参见订阅消息。

Spring框架下支持的更多配置参数请参见Java SDK接口和参数说明。

Demo下载

您可以通过以下链接获取相关Demo：

- spring/java-spring-demo
- springboot/java-springboot-demo

2.1.7. 使用Exactly-Once投递语义收发消息

本文主要介绍如何使用
消息队列RocketMQ版
的Exactly-Once投递语义收发消息，以保证消息的最终处理结果写入到数据库有且仅有一次。

背景信息

> **注意** 目前Exactly-Once投递语义仅在Java SDK中支持。相关SDK下载，请参见版本说明。

消息队列RocketMQ版
的Exactly-Once投递语义适用于接收消息>处理消息>结果持久化到数据库的流程，能够保证您的每一条消息消费的最终处理结果写入到您的数据库有且仅有一次，保证消息消费的幂等。
更多Exactly-Once投递语义的概念和典型使用场景，请参见Exactly-Once投递语义。

操作步骤

若要使用该语义，请按照以下步骤进行操作：

1. 在应用中添加SDK包依赖和Spring 3.0以上版本的依赖。更多信息，请参见步骤一：添加依赖。
2. 在用于存储消息消费结果的数据库中创建transaction_record表。更多信息，请参见步骤二：创建消费事务表。

   > **注意** 存储消息消费结果的数据库系统必须支持本地事务。

3. 在消息生产端使用PropertyKeyConst.EXACTLYONCE_DELIVERY属性设置打开Exactly-Once投递语义。更多信息，请参见步骤三：生产端开启Exactly-Once投递语义。

4. 在消息消费端创建ExactlyOnceConsumer，并开启Exactly-Once的消费模式。更多信息，请参见步骤四：消费端开启Exactly-Once投递语义。

步骤一：添加依赖

消息队列RocketMQ版
的ExactlyOnceConsumer在客户端SDK ons-client-ext-1.8.4.Final中发布，若要使用Exactly-Once投递语义，需在应用中依赖该SDK。
另外，ExactlyOnceConsumer基于Spring实现了通过注解@MQTransaction开启Exactly-Once消费的方式，因此还需要在应用中增加Spring 3.0以上版本的依赖。

完整的依赖内容如下所示。

```
<dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>ons-client-ext</artifactId>
    <version>1.8.4.Final</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring-version}</version>
</dependency>
```

步骤二：创建消费事务表

若要使用

消息队列RocketMQ版
的Exactly-Once投递语义，您需要在业务处理结果持久化的数据库中创建一张transaction_record表，保证此表与存储业务处理结果的表在同一个数据库内，且该数据库支持本地事务。
目前，

消息队列RocketMQ版
的Exactly-Once投递语义支持您的业务访问MySQL和SQLServer两种类型的数据源。这两种类型的数据源下transaction_record表的建表语句如以下所示。

- MySQL

```
CREATE TABLE `transaction_record` (
  `consumer_group` varchar(128) NOT NULL DEFAULT '',
  `message_id` varchar(255) NOT NULL DEFAULT '',
  `topic_name` varchar(255) NOT NULL DEFAULT '',
  `ctime` bigint(20) NOT NULL,
  `queue_id` int(11) NOT NULL,
  `offset` bigint(20) NOT NULL,
  `broker_name` varchar(255) NOT NULL DEFAULT '',
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  UNIQUE KEY `message_id_unique_key` (`message_id`),
  KEY `ctime_key` (`ctime`),
  KEY `load_key` (`queue_id`,`broker_name`,`topic_name`,`ctime`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- SQLServer

```
CREATE TABLE transaction_record
(
    [consumer_group] varchar(128) NOT NULL ,
    [message_id] varchar(255) NOT NULL ,
    [topic_name] varchar(255) NOT NULL ,
    [ctime] bigint NOT NULL ,
    [queue_id] int NOT NULL ,
    [offset] bigint NOT NULL ,
    [broker_name] varchar(50) NOT NULL ,
    [id] bigint IDENTITY(1,1)  PRIMARY KEY
)
CREATE NONCLUSTERED  INDEX load_key ON transaction_record
(queue_id, broker_name, topic_name, ctime);
CREATE UNIQUE NONCLUSTERED INDEX message_id_uniq_key ON transaction_record
(message_id);
CREATE NONCLUSTERED INDEX ctime_key ON transaction_record
(ctime);
```

注意

- 对于企业版SQL Server数据库，建议您通过 `ALTER DATABASE [USERDB] SET PARTNER SAFETY OFF;` 打开异步模式，提高数据库读写性能。
- SQL Server数据库同时还可以通过 `ALTER DATABASE [USERDB] SET DELAYED_DURABILITY=FORCED` 的方式开启Delayed Durability选项，通过开启该选项可以降低对数据库的IOPS。

步骤三：生产端开启Exactly-Once投递语义

在生产端，您仅需要在创建Producer时，将 `PropertyKeyConst.EXACTLYONCE_DELIVERY` 属性设置为true，即可打开Exactly-Once投递语义，示例代码如下。

```
/**
 * TestExactlyOnceProducer启动
 * 通过PropertyKeyConst.EXACTLYONCE_DELIVERY开启exactly-once投递语义。
 */
public class TestExactlyOnceProducer {
    public static void main(String[] args) {
        Properties producerProperties = new Properties();
        producerProperties.setProperty(PropertyKeyConst.GROUP_ID,"{GROUP_ID}");
        producerProperties.setProperty(PropertyKeyConst.AccessKey,"{accessKey}");
        producerProperties.setProperty(PropertyKeyConst.SecretKey,"{secretKey}");
        producerProperties.setProperty(PropertyKeyConst.NAMESRV_ADDR,"{NAMESRV_ADDR}");
        producerProperties.setProperty(PropertyKeyConst.EXACTLYONCE_DELIVERY,"true");
        Producer producer = ExactlyOnceONSFactory.createProducer(producerProperties);
        producer.start();
        System.out.println("Producer Started");
        for (int i = 0; i < 10; i++) {
            Message message = new Message("{topic}", "{tag}", "mq send transaction message
test".getBytes());
            try {
                SendResult sendResult = producer.send(message);
                assert sendResult != null;
                System.out.println(new Date() + " Send mq message success! msgId is: " + se
ndResult.getMessageId());
            } catch (ONSClientException e) {
                System.out.println("发送失败");
                //出现异常意味着发送失败，为避免消息丢失，建议缓存该消息然后进行重试。
            }
        }
        producer.shutdown();
    }
}
```

步骤四：消费端开启Exactly-Once投递语义

使用

消息队列RocketMQ版
的Exactly-Once投递语义进行消费时，消费端需要使用ExactlyOnceONSFactory调用
createExactlyOnceConsumer接口创建ExactlyOnceConsumer，然后通过使用ExactlyOnceConsumer进行Exactly-Once模式的消费。
在使用ExactlyOnceConsumer时，需要注意以下两点：

- 创建ExactlyOnceConsumer时，可以通过设置PropertyKeyConst.EXACTLYONCE_DELIVERY属性打开或者关闭Exactly-Once投递语义。ExactlyOnceConsumer默认打开Exactly-Once投递语义。
- 使用ExactlyOnceConsumer消费时，在消息监听器MessageListener的consume方法中，您的业务处理逻辑需要使用MQDataSource对数据库的进行读写操作。

您可以选择以下任一方式在消费端开启Exactly-Once投递语义：

- 以非Spring方式开启Exactly-Once投递语义
  示例如下。

```
/**
 * ExactlyOnceConsumer启动。
 * 通过PropertyKeyConst.EXACTLYONCE_DELIVERY开启Exactly-Once投递语义。
```

```
 */
 public class TestExactlyOnceConsumer {
    private ExactlyOnceConsumer consumer;
    private TxMessageListener listener;
    public TestExactlyOnceConsumer() {
        Properties consumerProperties = new Properties();
        consumerProperties.setProperty(PropertyKeyConst.GROUP_ID, "{GID}");
        consumerProperties.setProperty(PropertyKeyConst.AccessKey, "{accessKey}");
        consumerProperties.setProperty(PropertyKeyConst.SecretKey, "{secretKey}");
        consumerProperties.setProperty(PropertyKeyConst.NAMESRV_ADDR, "{NAMESRV_ADDR}");
        this.consumer = ExactlyOnceONSFactory.createExactlyOnceConsumer(consumerPropertie
s);
        this.consumer.subscribe("{topic}", "", new TestExactlyOnceListener());
        consumer.start();
        System.out.println("Consumer start success.");
    }
}
/**
 * SimpleListener为使用ExactlyOnceConsumer消费的简单示例。
 * 实现了一个简单的消息记录到数据库的过程，保证每条消息持久化到数据库有且仅有一次生效。
 */
public class SimpleListener implements MessageListener {
    private MQDataSource dataSource;
    public SimpleListener() {
        this.dataSource = new MQDataSource("{url}", "{user}", "{passwd}", "{driver}");
    }
    @Override
    public Action consume(Message message, ConsumeContext context) {
        Connection connection = null;
        PreparedStatement statement = null;
        try {
            /**
             * 在此处对消费到的消息message做业务计算处理，使用MQDataSource将处理结果持久化到数据
库系统。
             * 此示例演示了消费并记录消息到数据库系统的场景，实际的业务处理按照：接收消息->业务处理
->结果持久化的过程。
             * Exactly-Once投递语义保证消息处理的持久化过程有且仅有一次。
             */
            connection = dataSource.getConnection();
            statement = connection.prepareStatement("INSERT INTO app(msg, ctime) VALUES(?
, ?)");
            statement.setString(1, new String(message.getBody()));
            statement.setLong(2, System.currentTimeMillis());
            statement.execute();
            System.out.println("consume message success");
            return Action.CommitMessage;
        } catch (Throwable e) {
            System.out.println("consume message fail:" + e.getMessage());
            return Action.ReconsumeLater;
        } finally {
            if (statement != null) {
                try {
                    statement.close();
                } catch (Exception e) {
                }
```

```
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (Exception e) {
                }
            }
        }
    }
}
```

- MessageListener中以事务方式实现多项数据库操作和消息消费的事务性
示例如下。

```
/**
 * TestExactlyOnceListener实现。
 * 实现了一个事务中对多个业务表进行更新的场景，保证事务内的操作有且仅有一次生效。
 */
public class SimpleTxListener implements MessageListener {
    private MQDataSource dataSource;
    public SimpleTxListener() {
        this.dataSource = new MQDataSource("{url}", "{user}", "{passwd}", "{driver}");
    }
        @Override
    public Action consume(Message message, ConsumeContext context) {
        Connection connection = null;
        Statement statement = null;
        try {
            /**
             *  在此处对消费到的消息message做业务计算处理，使用MQDataSource将处理结果持久化到数据库系统。
             *  此范例演示了在一个事务内对多个表进行更新的业务场景，Exactly-Once投递语义保证事务内的操作有且仅有一次。
             *  实际的业务处理按照：接收消息->业务处理->结果持久化的流程来设计。
             */
            connection = dataSource.getConnection();
            connection.setAutoCommit(false);
            String insertSql = String.format("INSERT INTO app(msg, message_id, ctime) VAL
UES(\"%s\", \"%s\", %d)",
                new String(message.getBody()), message.getMsgID(), System.currentTimeMill
is());
            String updateSql = String.format("UPDATE consume_count SET cnt = count + 1 WH
ERE consumer_group = \"%s\"", "GID_TEST");
            statement = connection.createStatement();
            statement.execute(insertSql);
            statement.execute(updateSql);
            connection.commit();
            System.out.println("consume message :" + message.getMsgID());
            return Action.CommitMessage;
        } catch (Throwable e) {
            try {
                connection.rollback();
            } catch (Exception e1) {
            }
```

```
            System.out.println("consume message fail");
            return Action.ReconsumeLater;
        } finally {
            if (statement != null) {
                try {
                    statement.close();
                } catch (Exception e) { }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (Exception e) { }
            }
        }
    }
}
```

- MessageListener中通过Springboot注解方式实现开启Exactly-Once投递语义
  i. 创建MessageListener，如下所示。

```
/**
 * MessageListener通过注解方式开启Exactly-Once。
 * 只需在MessageListener的consume方法加上@MQTransaction即可开启。
 * 适用于springboot微服务中使用。
 */
public class TestMessageListener implements MessageListener {
    private final static String INSERTSQLFORMAT = "INSERT INTO app(message_id, ctime)
VALUES(\"%s\", %d)";
    private MQDataSource dataSource;
    @Override
    @MQTransaction
    public Action consume(Message message, ConsumeContext context) {
        Connection connection = null;
        Statement statement = null;
        try {
            String insertSql = String.format(INSERTSQLFORMAT, message.getMsgID(), Sys
tem.currentTimeMillis());
            connection = this.dataSource.getConnection();
            statement = connection.createStatement();
            statement.execute(insertSql);
            return Action.CommitMessage;
        } catch (Throwable e) {
            return Action.ReconsumeLater;
        } finally {
            if (statement != null) {
                try {
                    statement.close();
                } catch(Exception e) { }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (Exception e) { }
            }
        }
    }
    public void setDataSource(MQDataSource dataSource) {
        this.dataSource = dataSource;
    }
}
```

ii. 在consumer.xml中定义消费者Bean等信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spr
ingframework.org/schema/beans/spring-beans.xsd">
    <bean id="mqDataSource" class="com.aliyun.openservices.ons.api.impl.rocketmq.exac
tlyonce.datasource.MQDataSource" init-method="init" destroy-method="close">
        <property name="url" value="{url}" />
        <property name="username" value="{user}" />
        <property name="password" value="{passwd}" />
        <property name="driverClass" value="{driver}" />
    </bean>
    <bean id="msgListener" class="com.aliyun.openservices.ons.api.impl.rocketmq.exact
lyonce.spring.TestMessageListener">
        <property name="dataSource" ref="mqDataSource"> <!--消费者配置信息-->
        </property>
    </bean> <!--Listener配置-->
    <!--多GID订阅同一个Topic，可以创建多个ConsumerBean-->
    <bean id="consumer" class="com.aliyun.openservices.ons.api.bean.ExactlyOnceConsum
erBean" init-method="start" destroy-method="shutdown">
        <property name="properties" > <!--消费者配置信息-->
            <props>
                <prop key="GROUP_ID">{gid}</prop>
                <prop key="AccessKey">{accessKey}</prop>
                <prop key="SecretKey">{secretKey}</prop>
                <!--将消费者线程数固定为50个
                <prop key="ConsumeThreadNums">50</prop>
                -->
            </props>
        </property>
        <property name="subscriptionTable">
            <map>
                <entry value-ref="msgListener">
                    <key>
                        <bean class="com.aliyun.openservices.ons.api.bean.Subscriptio
n">
                            <property name="topic" value="{topic}"/>
                            <property name="expression" value="{subExpression}"/><!--
expression 即 Tag，可以设置成具体的 Tag，如 taga||tagb||tagc，也可设置成*。 *仅代表订阅所有
Tag，不支持通配-->
                        </bean>
                    </key>
                </entry>
            </map>
        </property>
    </bean>
</beans>
```

iii. 运行已经与Spring集成好的消费者，如下所示。

```
public class ConsumeWithSpring {
    public static void main(String[] args) {
        /**
         * 消费者Bean配置在consumer.xml中，可通过ApplicationContext获取或者直接注入到其他类（例如具体的Controller）中。
         */
        ApplicationContext context = new ClassPathXmlApplicationContext("consumer.xml");
        System.out.println("Consumer Started");
    }
```

- MessageListener中通过MyBatis方式实现Exactly-Once投递语义
  i. 设计业务数据库写入操作DAO。

```
package com.aliyun.openservices.tcp.example.mybatis;
public interface AppDAO {
    Integer insertMessage(String msgId);
}
```

  ii. 在mapper.xml文件中编写插入操作SQL语句。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <mapper namespace="com.aliyun.openservices.tcp.example.mybatis.AppDAO">
        <insert id="insertMessage">
            INSERT INTO app (message_id, ctime) VALUES (#{msgId}, now())
        </insert>
    </mapper>
```

  iii. 利用MQDataSource实现自定义的DataSourceFactory。

```
public class MQDataSourceFactoty extends DruidDataSourceFactory implements DataSourceFactory {
    protected Properties properties;
    @Override
    public void setProperties(Properties properties) {
        this.properties = properties;
    }
    @Override
    public DataSource getDataSource() {
        try {
            DruidDataSource druidDataSource = (DruidDataSource) createDataSource(this.properties);
            return new MQDataSource(druidDataSource);
        } catch (Exception e) {
            System.out.println("err:" + e.getMessage());
        }
        return null;
    }
}
```

  iv. 在mybatis-config.xml中注册datasource为MQDataSourceFactoty类型。

```
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <!-- 配置数据库连接信息 -->
            <dataSource type="com.aliyun.openservices.tcp.example.mybatis.MQDataSourc
eFactoty">
                <property name="driverClass" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="{url}"/>
                <property name="username" value="{username}"/>
                <property name="password" value="{password}"/>
                <property name="initialSize" value="10" />
                <property name="maxActive" value="20"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="mapper.xml"/>
    </mappers>
</configuration>
```

v. 在监听器中使用MyBatis的方式访问数据库，实现Exactly-Once的消费方式。

```
public class TestMybatisListener implements MessageListener {
    private static SqlSessionFactory sqlSessionFactory;
    static {
        String resource = "mybatis-config.xml";
        Reader reader = null;
        try {
            reader= Resources.getResourceAsReader(resource);
        } catch (IOException e) {
            e.printStackTrace();
        }
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
    }
    @Override
    public Action consume(Message message, ConsumeContext context) {
        long begion = System.currentTimeMillis();
          SqlSession sqlSession = null;
        try {
            sqlSession = sqlSessionFactory.openSession();
            AppDAO appDAO = sqlSession.getMapper(AppDAO.class);
            appDAO.insertMessage(message.getMsgID());
            System.out.println("consume : " + message.getMsgID());
            sqlSession.commit();
            return Action.CommitMessage;
        } catch (Exception e) {
            e.printStackTrace();
              sqlSession.rollback();
            return Action.ReconsumeLater;
        } finally {
            sqlSession.close();
        }
    }
}
```

注意事项

在使用

消息队列RocketMQ版
的ExactlyOnceConsumer消费消息的过程中，请注意以下两点：

- 不可在
  消息队列RocketMQ版
  控制台手动重置消费位点。若您主动重置位点到一个已经消费过的位点，则会失去Exactly-Once的投递语义。
- 每次数据库的INSERT或UPDATE操作会带来一次额外的更新操作，同时
  消息队列RocketMQ版
  的ExactlyOnceConsumer也会有定时的查询或删除操作，会对数据库的IOPS带来一定增长。

2.1.8. 发送普通消息（三种方式）

消息队列RocketMQ版
提供三种方式来发送普通消息：同步（Sync）发送、异步（Async）发送和单向（Oneway）发送。本文介绍了每种发送方式的原理、应用场景、示例代码，以及三种发送方式的对比。

前提条件

通过Java SDK发送普通消息前，您已完成以下操作。

- 下载Java SDK。Java SDK版本说明，请参见版本说明。
- 准备环境。更多信息，请参见准备环境。
- （可选）日志配置。更多信息，请参见日志配置。

同步发送

- 原理
  同步发送是指消息发送方发出一条消息后，会在收到服务端同步响应之后才发下一条消息的通讯方式。

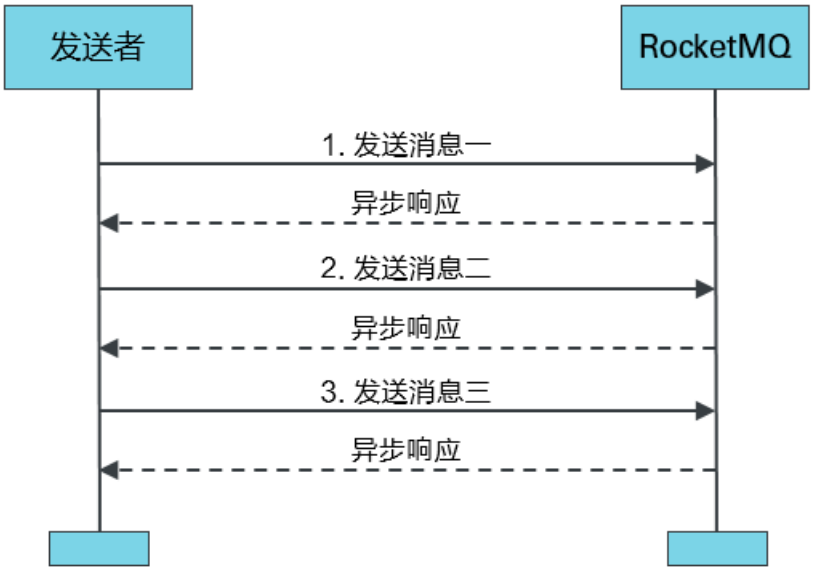
- 应用场景
  此种方式应用场景非常广泛，例如重要通知邮件、报名短信通知、营销短信系统等。
- 示例代码

```
import com.aliyun.openservices.ons.api.Message;
    import com.aliyun.openservices.ons.api.Producer;
    import com.aliyun.openservices.ons.api.SendResult;
    import com.aliyun.openservices.ons.api.ONSFactory;
    import com.aliyun.openservices.ons.api.PropertyKeyConst;
    import java.util.Properties;
    public class ProducerTest {
        public static void main(String[] args) {
            Properties properties = new Properties();
            // AccessKeyId 阿里云身份验证，在阿里云用户信息管理控制台获取。
            properties.put(PropertyKeyConst.AccessKey,"XXX");
            // AccessKeySecret 阿里云身份验证，在阿里云用户信息管理控制台获取。
            properties.put(PropertyKeyConst.SecretKey, "XXX");
            //设置发送超时时间，单位：毫秒。
            properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
            // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            properties.put(PropertyKeyConst.NAMESRV_ADDR, "XXX");
            Producer producer = ONSFactory.createProducer(properties);
            // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
            producer.start();
            //循环发送消息。
```

```
            for (int i = 0; i < 100; i++){
                Message msg = new Message(
                    // 普通消息所属的Topic，切勿使用普通消息的Topic来收发其他类型的消息。
                    "TopicTestMQ",
                    // Message Tag可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤
条件在
消息队列RocketMQ版的服务器过滤。
                    // Tag的具体格式和设置方法，请参见Topic与Tag最佳实践。
                    "TagA",
                    // Message Body可以是任何二进制形式的数据，
消息队列RocketMQ版不做任何干预。
                    // 需要Producer与Consumer协商好一致的序列化和反序列化方式。
                    "Hello MQ".getBytes());
                // 设置代表消息的业务关键属性，请尽可能全局唯一。
                // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
                // 注意：不设置也不会影响消息正常收发。
                msg.setKey("ORDERID_" + i);
                try {
                    SendResult sendResult = producer.send(msg);
                    // 同步发送消息，只要不抛异常就是成功。
                    if (sendResult != null) {
                        System.out.println(new Date() + " Send mq message success. Topic
is:" + msg.getTopic() + " msgId is: " + sendResult.getMessageId());
                    }
                }
                catch (Exception e) {
                    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处
理。
                    System.out.println(new Date() + " Send mq message failed. Topic is:"
+ msg.getTopic());
                    e.printStackTrace();
                }
            }
            // 在应用退出前，销毁Producer对象。
            // 注意：如果不销毁也没有问题。
            producer.shutdown();
        }
    }
```

异步发送

- 原理
  异步发送是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式。
  消息队列RocketMQ版
  的异步发送，需要您实现异步发送回调接口（SendCallback）。消息发送方在发送了一条消息后，不需要等待服务端响应即可发送第二条消息。发送方通过回调接口接收服务端响应，并处理响应结果。



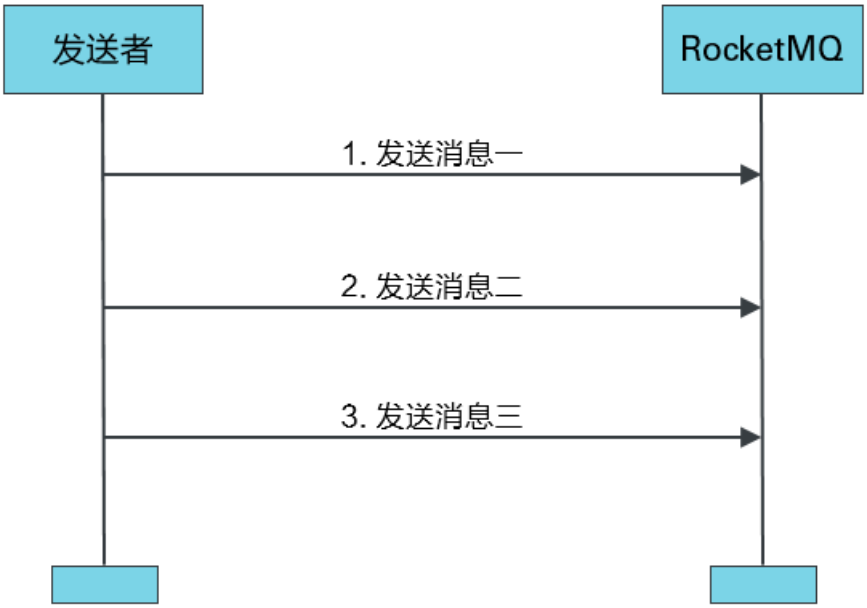
- 应用场景
  异步发送一般用于链路耗时较长，对响应时间较为敏感的业务场景，例如，您视频上传后通知启动转码服务，转码完成后通知推送转码结果等。
- 示例代码

```
    import com.aliyun.openservices.ons.api.Message;
    import com.aliyun.openservices.ons.api.OnExceptionContext;
    import com.aliyun.openservices.ons.api.Producer;
    import com.aliyun.openservices.ons.api.SendCallback;
    import com.aliyun.openservices.ons.api.SendResult;
    import com.aliyun.openservices.ons.api.ONSFactory;
    import com.aliyun.openservices.ons.api.PropertyKeyConst;
    import java.util.Properties;
    import java.util.concurrent.TimeUnit;
    public class ProducerTest {
        public static void main(String[] args) throws InterruptedException {
            Properties properties = new Properties();
            // AccessKeyId 阿里云身份验证，在阿里云服务器管理控制台创建。
            properties.put(PropertyKeyConst.AccessKey, "XXX");
            // AccessKeySecret 阿里云身份验证，在阿里云服务器管理控制台创建。
            properties.put(PropertyKeyConst.SecretKey, "XXX");
            //设置发送超时时间，单位毫秒。
            properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
            // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            properties.put(PropertyKeyConst.NAMESRV_ADDR, "XXX");
            Producer producer = ONSFactory.createProducer(properties);
            // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
            producer.start();
            Message msg = new Message(
                    // 普通消息所属的Topic，切勿使用普通消息的Topic来收发其他类型的消息。
                    "TopicTestMQ",
                    // Message Tag，可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过
滤条件在
消息队列RocketMQ版的服务器过滤。
```

```
                    "TagA",
                    // Message Body，任何二进制形式的数据，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
                    "Hello MQ".getBytes());
            // 设置代表消息的业务关键属性，请尽可能全局唯一。 以方便您在无法正常收到消息情况下，可通
过
消息队列RocketMQ版控制台查询消息并补发。
            // 注意：不设置也不会影响消息正常收发。
            msg.setKey("ORDERID_100");
            // 异步发送消息，发送结果通过callback返回给客户端。
            producer.sendAsync(msg, new SendCallback() {
                @Override
                public void onSuccess(final SendResult sendResult) {
                    // 消息发送成功。
                    System.out.println("send message success. topic=" + sendResult.getTop
ic() + ", msgId=" + sendResult.getMessageId());
                }
                @Override
                public void onException(OnExceptionContext context) {
                    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处
理。
                    System.out.println("send message failed. topic=" + context.getTopic()
+ ", msgId=" + context.getMessageId());
                }
            });
            // 阻塞当前线程3秒，等待异步发送结果。
            TimeUnit.SECONDS.sleep(3);
            // 在应用退出前，销毁Producer对象。注意：如果不销毁也没有问题。
            producer.shutdown();
        }
    }
```

单向发送

- 原理
  发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发，即只发送请求不等待应答。此方式发送消息的过程耗时非常短，一般在微秒级别。



- 应用场景
  适用于某些耗时非常短，但对可靠性要求并不高的场景，例如日志收集。
- 示例代码

```
import com.aliyun.openservices.ons.api.Message;
    import com.aliyun.openservices.ons.api.Producer;
    import com.aliyun.openservices.ons.api.ONSFactory;
    import com.aliyun.openservices.ons.api.PropertyKeyConst;
    import java.util.Properties;
    public class ProducerTest {
        public static void main(String[] args) {
            Properties properties = new Properties();
            // AccessKeyId 阿里云身份验证，在阿里云用户信息管理控制台创建。
            properties.put(PropertyKeyConst.AccessKey, "XXX");
            // AccessKeySecret 阿里云身份验证，在阿里云用户信息管理控制台创建。
            properties.put(PropertyKeyConst.SecretKey, "XXX");
            //设置发送超时时间，单位：毫秒。
            properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
            // 设置TCP接入域名，进入
消息队列RocketMQ版控制台的实例详情页面的接入点区域查看。
            properties.put(PropertyKeyConst.NAMESRV_ADDR,
              "XXX");
            Producer producer = ONSFactory.createProducer(properties);
            // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
            producer.start();
            //循环发送消息。
            for (int i = 0; i < 100; i++){
                Message msg = new Message(
                        // 普通消息所属的Topic，切勿使用普通消息的Topic来收发其他类型的消息。
                        "TopicTestMQ",
                        // Message Tag,
                        // 可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在
消息队列RocketMQ版的服务器过滤。
                        "TagA",
                        // Message Body
                        // 任何二进制形式的数据，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
                        "Hello MQ".getBytes());
                // 设置代表消息的业务关键属性，请尽可能全局唯一。
                // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
                // 注意：不设置也不会影响消息正常收发。
                msg.setKey("ORDERID_" + i);
                // 由于在oneway方式发送消息时没有请求应答处理，如果出现消息发送失败，则会因为没有
重试而导致数据丢失。若数据不可丢，建议选用可靠同步或可靠异步发送方式。
                producer.sendOneway(msg);
            }
            // 在应用退出前，销毁Producer对象。
            // 注意：如果不销毁也没有问题。
            producer.shutdown();
        }
    }
```

三种发送方式的对比

下表概括了三者的特点和主要区别。

发送方式	发送TPS	发送结果反馈	可靠性
同步发送	快	有	不丢失
异步发送	快	有	不丢失
单向发送	最快	无	可能丢失

订阅普通消息

订阅普通消息的示例代码，请参见订阅消息。

2.1.9. 发送消息（多线程）

消息队列RocketMQ版
的消费者和生产者客户端对象是线程安全的，可以在多个线程之间共享使用。
您可以在服务器上（或者多台服务器）部署多个生产者和消费者实例，也可以在同一个生产者或消费者实例里采用多线程发送或接收消息，从而提高消息发送或接收TPS。

> **注意**
>
> - 请避免为每个线程创建一个客户端实例。
> - 顺序消息不建议使用多线程发送。
>   消息队列RocketMQ版
>   服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
>   消息队列RocketMQ版
>   服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

在多线程之间共享Producer的示例代码如下。

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import java.util.Properties;
public class SharedProducer {
    public static void main(String[] args) {
        // producer实例配置初始化。
        Properties properties = new Properties();
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        properties.put(PropertyKeyConst.GROUP_ID,"XXX");
        // AccessKey ID阿里云身份验证，在阿里云服务器管理控制台创建。
        properties.put(PropertyKeyConst.AccessKey,"XXX");
        // AccessKey Secret阿里云身份验证，在阿里云服务器管理控制台创建。
        properties.put(PropertyKeyConst.SecretKey,"XXX");
        // 设置发送超时时间，单位：毫秒。
        properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis,"3000");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR, "XXX");
```

```
        final Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
        producer.start();
        //创建的Producer和Consumer对象为线程安全的，可以在多线程间进行共享，避免每个线程创建一个实例
。
        //在thread和anotherThread中共享Producer对象，并发地发送消息至
消息队列RocketMQ版。
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Message msg = new Message(
                    // 普通消息所属的Topic，切勿使用普通消息的Topic来收发其他类型的消息。
                    "TopicTestMQ",
                    // Message Tag可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条
件在
消息队列RocketMQ版的服务器过滤。
                    "TagA",
                    // Message Body可以是任何二进制形式的数据，
消息队列RocketMQ版不做任何干预。
                    // 需要Producer与Consumer协商好一致的序列化和反序列化方式。
                    "Hello MQ".getBytes());
                    SendResult sendResult = producer.send(msg);
                    // 同步发送消息，只要不抛异常就是成功。
                    if (sendResult != null) {
                        System.out.println(new Date() + " Send mq message success. Topic is
:" + MqConfig.TOPIC + " msgId is: " + sendResult.getMessageId());
                    }
                } catch (Exception e) {
                    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理
。
                    System.out.println(new Date() + " Send mq message failed. Topic is:" +
MqConfig.TOPIC);
                    e.printStackTrace();
                }
            }
        });
        thread.start();
        Thread anotherThread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Message msg = new Message("TopicTestMQ", "TagA", "Hello MQ".getBytes())
;
                    SendResult sendResult = producer.send(msg);
                    // 同步发送消息，只要不抛异常就是成功。
                    if (sendResult != null) {
                        System.out.println(new Date() + " Send mq message success. Topic is
:" + MqConfig.TOPIC + " msgId is: " + sendResult.getMessageId());
                    }
                } catch (Exception e) {
                    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理
。
                    System.out.println(new Date() + " Send mq message failed. Topic is:" +
MqConfig.TOPIC);
```

```
                    e.printStackTrace();
                }
            }
        });
        anotherThread.start();
        //（可选）Producer实例若不再使用时，可将Producer关闭，进行资源释放。
        // producer.shutdown();
    }
}
```

2.1.10. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用TCP协议下的Java SDK收发顺序消息的示例代码。

前提条件

您已完成以下操作：

- 下载1.2.7或以上版本的Java SDK。更多信息，请参见Java SDK版本说明。
- 准备环境。更多信息，请参见准备环境。
- （可选）日志配置。更多信息，请参见日志配置。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息2.0。

> **说明** 对于新手用户，建议在正式收发消息前，阅读Demo工程来了解搭建
> 消息队列RocketMQ版
> 工程的具体步骤。

发送顺序消息

> **注意**
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

具体的示例代码，请以

消息队列RocketMQ版

代码库为准。
全局顺序消息和分区顺序消息的发布方式基本一样，示例代码如下。

```
package com.aliyun.openservices.ons.example.order;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.order.OrderProducer;
import java.util.Properties;
public class ProducerClient {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        properties.put(PropertyKeyConst.GROUP_ID,"XXX");
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.AccessKey,"XXX");
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.SecretKey,"XXX");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR,"XXX");
        OrderProducer producer = ONSFactory.createOrderProducer(properties);
        // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
        producer.start();
        for (int i = 0; i < 1000; i++) {
            String orderId = "biz_" + i % 10;
            Message msg = new Message(
                    // Message所属的Topic。
                    "Order_global_topic",
                    // Message Tag，可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤
条件在
消息队列RocketMQ版的服务器过滤。
                    "TagA",
                    // Message Body，可以是任何二进制形式的数据，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
                    "send order global msg".getBytes()
            );
            // 设置代表消息的业务关键属性，请尽可能全局唯一。
            // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
            // 注意：不设置也不会影响消息正常收发。
            msg.setKey(orderId);
            // 分区顺序消息中区分不同分区的关键字段，Sharding Key与普通消息的key是完全不同的概念。
            // 全局顺序消息，该字段可以设置为任意非空字符串。
            String shardingKey = String.valueOf(orderId);
            try {
                SendResult sendResult = producer.send(msg, shardingKey);
                // 发送消息，只要不抛异常就是成功。
                if (sendResult != null) {
                    System.out.println(new Date() + " Send mq message success. Topic is:" +
msg.getTopic() + " msgId is: " + sendResult.getMessageId());
                }
```

```
            }
            catch (Exception e) {
                // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
                System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.
getTopic());
                e.printStackTrace();
            }
        }
        // 在应用退出前，销毁Producer对象。
        // 注意：如果不销毁也没有问题。
        producer.shutdown();
    }
}
```

订阅顺序消息

全局顺序消息和分区顺序消息的订阅方式基本一样，示例代码如下。

```
package com.aliyun.openservices.ons.example.order;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.order.ConsumeOrderContext;
import com.aliyun.openservices.ons.api.order.MessageOrderListener;
import com.aliyun.openservices.ons.api.order.OrderAction;
import com.aliyun.openservices.ons.api.order.OrderConsumer;
import java.util.Properties;
public class ConsumerClient {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        properties.put(PropertyKeyConst.GROUP_ID,"XXX");
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.AccessKey,"XXX");
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.SecretKey,"XXX");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR,"XXX");
          // 顺序消息消费失败进行重试前的等待时间，单位（毫秒），取值范围：10毫秒~30,000毫秒。
        properties.put(PropertyKeyConst.SuspendTimeMillis,"100");
        // 消息消费失败时的最大重试次数。
        properties.put(PropertyKeyConst.MaxReconsumeTimes,"20");
        // 在订阅消息前，必须调用start方法来启动Consumer，只需调用一次即可。
        OrderConsumer consumer = ONSFactory.createOrderedConsumer(properties);
        consumer.subscribe(
                // Message所属的Topic。
                "Order_global_topic",
                // 订阅指定Topic下的Tags：
                // 1. * 表示订阅所有消息。
                // 2. TagA || TagB || TagC表示订阅TagA或TagB或TagC的消息。
                "*",
                new MessageOrderListener() {
                    /**
                     * 1. 消息消费处理失败或者处理出现异常，返回OrderAction.Suspend。
                     * 2. 消息处理成功，返回OrderAction.Success。
                     */
                    @Override
                    public OrderAction consume(Message message, ConsumeOrderContext context
) {
                        System.out.println(message);
                        return OrderAction.Success;
                    }
                });
        consumer.start();
    }
}
```

2.1.11. 收发事务消息

本文提供使用TCP协议下的Java SDK收发事务消息的示例代码。

消息队列RocketMQ版

提供类似XA或Open XA的分布式事务功能，通过

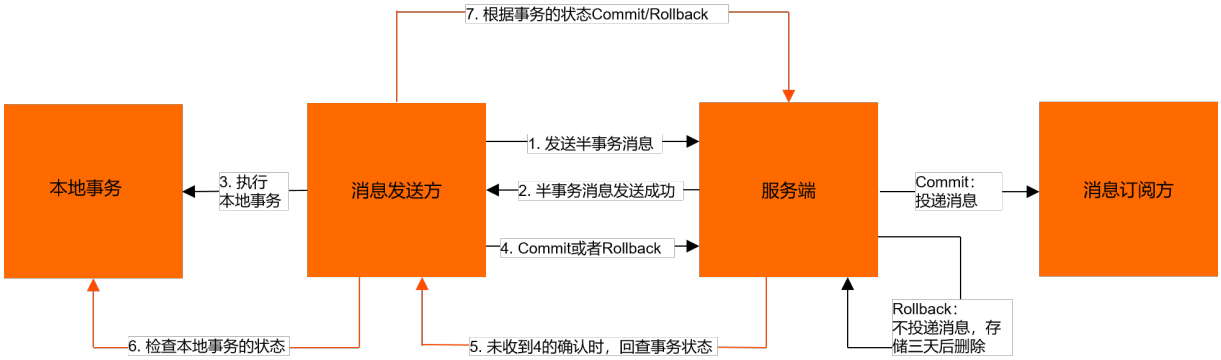
消息队列RocketMQ版

事务消息，能达到分布式事务的最终一致。

> **说明** 对于新手用户，建议在正式收发消息前，阅读Demo工程来了解搭建
> 消息队列RocketMQ版
> 工程的具体步骤。

交互流程

事务消息交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 下载Java SDK。Java SDK版本说明，请参见版本说明。
- 准备环境。更多信息，请参见准备环境。
- （可选）日志配置。更多信息，请参见日志配置。

发送事务消息

> **说明** 具体的示例代码，请以
> 消息队列RocketMQ版
> 代码库为准。

发送事务消息包含以下两个步骤：

1. 发送半事务消息（Half Message）及执行本地事务，示例代码如下。

```
package com.alibaba.webx.TryHsf.app1;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionExecuter;
```

```
import com.aliyun.openservices.ons.api.transaction.TransactionProducer;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
import java.util.Properties;
import java.util.concurrent.TimeUnit;
public class TransactionProducerClient {
 private final static Logger log = ClientLogger.getLog(); // 您需要设置自己的日志，便于排查问题。
 public static void main(String[] args) throws InterruptedException {
     final BusinessService businessService = new BusinessService(); // 本地业务。
     Properties properties = new Properties();
        // 您在消息队列RocketMQ版控制台创建的Group ID。注意：事务消息的Group ID不能与其他类型消息的Group ID共用。
     properties.put(PropertyKeyConst.GROUP_ID,"XXX");
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
     properties.put(PropertyKeyConst.AccessKey,"XXX");
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
     properties.put(PropertyKeyConst.SecretKey,"XXX");
        // 设置TCP接入域名，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
     properties.put(PropertyKeyConst.NAMESRV_ADDR,"XXX");
     TransactionProducer producer = ONSFactory.createTransactionProducer(properties,
             new LocalTransactionCheckerImpl());
     producer.start();
     Message msg = new Message("Topic","TagA","Hello MQ transaction===".getBytes());
     try {
             SendResult sendResult = producer.send(msg, new LocalTransactionExecuter()
{
                 @Override
                 public TransactionStatus execute(Message msg, Object arg) {
                     // 消息ID（有可能消息体一样，但消息ID不一样，当前消息属于半事务消息，所以消息ID在消息队列RocketMQ版控制台无法查询）。
                     String msgId = msg.getMsgID();
                     // 消息体内容进行crc32，也可以使用其它的如MD5。
                     long crc32Id = HashUtil.crc32Code(msg.getBody());
                     // 消息ID和crc32id主要是用来防止消息重复。
                     // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等。
                     // 如果要求消息绝对不重复，推荐做法是对消息体使用crc32或MD5来防止重复消息。
                     Object businessServiceArgs = new Object();
                     TransactionStatus transactionStatus = TransactionStatus.Unknow;
                     try {
                         boolean isCommit =
                             businessService.execbusinessService(businessServiceArgs);
                         if (isCommit) {
                             // 本地事务已成功则提交消息。
                             transactionStatus = TransactionStatus.CommitTransaction;
                         } else {
                             // 本地事务已失败则回滚消息。
                             transactionStatus = TransactionStatus.RollbackTransaction;
                         }
                     } catch (Exception e) {
                         log.error("Message Id:{}", msgId, e);
                     }
```

```
                    }
                    System.out.println(msg.getMsgID());
                    log.warn("Message Id:{}transactionStatus:{}", msgId, transactionSt
atus.name());
                    return transactionStatus;
                }
            }, null);
        }
        catch (Exception e) {
            // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理
。
            System.out.println(new Date() + " Send mq message failed. Topic is:" + msg
.getTopic());
            e.printStackTrace();
        }
    // demo example防止进程退出（实际使用不需要这样）。
    TimeUnit.MILLISECONDS.sleep(Integer.MAX_VALUE);
 }
}
```

2. 提交事务消息状态。
   当本地事务执行完成（执行成功或执行失败），需要通知服务器当前消息的事务状态。通知方式有以下两种：
   - 执行本地事务完成后提交。
   - 执行本地事务一直没提交状态，等待服务器回查消息的事务状态。

   事务状态有以下三种：
   - `TransactionStatus.CommitTransaction`：提交事务，允许订阅方消费该消息。
   - `TransactionStatus.RollbackTransaction`：回滚事务，消息将被丢弃不允许消费。
   - `TransactionStatus.Unknow`：无法判断状态，期待
     消息队列RocketMQ版
     的Broker向发送方再次询问该消息对应的本地事务的状态。

```
public class LocalTransactionCheckerImpl implements LocalTransactionChecker {
   private final static Logger log = ClientLogger.getLog();
   final  BusinessService businessService = new BusinessService();
   @Override
   public TransactionStatus check(Message msg) {
       //消息ID（有可能消息体一样，但消息ID不一样，当前消息属于半事务消息，所以消息ID在
消息队列RocketMQ版控制台无法查询）。
       String msgId = msg.getMsgID();
       //消息体内容进行crc32，也可以使用其它的方法如MD5。
       long crc32Id = HashUtil.crc32Code(msg.getBody());
       //消息ID和crc32Id主要是用来防止消息重复。
       //如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等。
       //如果要求消息绝对不重复，推荐做法是对消息体使用crc32或MD5来防止重复消息。
       //业务自己的参数对象，这里只是一个示例，需要您根据实际情况来处理。
       Object businessServiceArgs = new Object();
       TransactionStatus transactionStatus = TransactionStatus.Unknow;
       try {
           boolean isCommit = businessService.checkbusinessService(businessServiceArgs)
;
           if (isCommit) {
               //本地事务已成功则提交消息。
               transactionStatus = TransactionStatus.CommitTransaction;
           } else {
               //本地事务已失败则回滚消息。
               transactionStatus = TransactionStatus.RollbackTransaction;
           }
       } catch (Exception e) {
           log.error("Message Id:{}", msgId, e);
       }
       log.warn("Message Id:{}transactionStatus:{}", msgId, transactionStatus.name());
       return transactionStatus;
   }
 }
```

工具类

```
import java.util.zip.CRC32;
public class HashUtil {
    public static long crc32Code(byte[] bytes) {
        CRC32 crc32 = new CRC32();
        crc32.update(bytes);
        return crc32.getValue();
    }
}
```

事务回查机制说明

- 发送事务消息为什么必须要实现回查Check机制？
  当步骤1中半事务消息发送完成，但本地事务返回状态为 `TransactionStatus.Unknow` ，或者应用退出导致本地事务未提交任何状态时，从Broker的角度看，这条半事务消息的状态是未知的。因此Broker会定期向消息发送方即消息生产者集群中的任意一生产者实例发起消息回查，要求发送方回查该Half状态消息，并上报其最终状态。
- Check被回调时，业务逻辑都需要做些什么？

事务消息的Check方法里面，应该写一些检查事务一致性的逻辑。
消息队列RocketMQ版
发送事务消息时需要实现 `LocalTransactionChecker` 接口，用来处理Broker主动发起的本地事务状态回查请求，因此在事务消息的Check方法中，需要完成两件事情：

i. 检查该半事务消息对应的本地事务的状态（committed or rollback）。

ii. 向Broker提交该半事务消息本地事务的状态。

订阅事务消息

事务消息的订阅与普通消息订阅一致，更多信息，请参见订阅消息。

2.1.12. 收发延时消息

本文提供使用TCP协议下的Java SDK收发延时消息的示例代码供您参考。

前提条件

您已完成以下操作：

- 下载Java SDK。Java SDK版本说明，请参见版本说明。
- 准备环境。更多信息，请参见准备环境。
- （可选）日志配置。更多信息，请参见日志配置。

背景信息

延时消息用于指定消息发送到

消息队列RocketMQ版
的服务端后，延时一段时间才被投递到客户端进行消费（例如3秒后才被消费），适用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发延迟任务的场景，类似于延迟队列。
延时消息的概念介绍及使用过程中的注意事项，请参见定时和延时消息。

> **说明** 对于新手用户，建议在正式收发消息前，阅读Demo 工程来了解搭建
> 消息队列RocketMQ版
> 工程的具体步骤。

发送延时消息

具体的示例代码，请以

消息队列RocketMQ版
代码库为准。
发送延时消息的示例代码如下：

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import java.util.Properties;
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
```

```
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
        producer.start();
        Message msg = new Message(
                // 您在
消息队列RocketMQ版控制台创建的Topic。
                "Topic",
                // Message Tag,可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在
消息队列RocketMQ版服务器过滤。
                "tag",
                // Message Body可以是任何二进制形式的数据，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
                "Hello MQ".getBytes());
        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发。
        msg.setKey("ORDERID_100");
        try {
            // 延时消息，在指定延迟时间（当前时间之后）进行投递。最大可设置延迟40天投递，单位毫秒（ms
）。
            // 以下示例表示消息在3秒后投递。
            long delayTime = System.currentTimeMillis() + 3000;
            // 设置消息需要被投递的时间。
            msg.setStartDeliverTime(delayTime);
            SendResult sendResult = producer.send(msg);
            // 同步发送消息，只要不抛异常就是成功。
            if (sendResult != null) {
            System.out.println(new Date() + " Send mq message success. Topic is:" + msg.get
Topic() + " msgId is: " + sendResult.getMessageId());
            }
            } catch (Exception e) {
            // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
            System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getT
opic());
            e.printStackTrace();
        }
        // 在应用退出前，销毁Producer对象。
        // 注意：如果不销毁也没有问题。
        producer.shutdown();
    }
}
```

订阅延时消息

延时消息的订阅与普通消息订阅一致，更多信息，请参见订阅消息。

2.1.13. 收发定时消息

本文提供使用TCP协议下的Java SDK收发定时消息的示例代码。

背景信息

定时消息可以做到在指定时间戳之后才可被消费者消费，适用于对消息生产和消费有时间窗口要求，或者利用消息触发定时任务的场景。

定时消息的概念介绍及使用过程中的注意事项，请参见定时和延时消息。

> **说明** 对于新手用户，建议在正式收发消息前，阅读Demo工程来了解搭建消息队列RocketMQ版工程的具体步骤。

前提条件

您已完成以下操作：

- 下载Java SDK。更多信息，请参见Java SDK版本说明。
- 准备环境。更多信息，请参见准备环境。
- （可选）日志配置。更多信息，请参见日志配置。

发送定时消息

> **说明** 具体的示例代码，请以消息队列RocketMQ版代码库为准。

发送定时消息的示例代码如下。

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Properties;
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // 阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // 阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
        producer.start();
        Message msg = new Message(
                // Message所属的Topic。
                "Topic",
                // Message Tag可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在
消息队列RocketMQ版的服务器过滤。
```

```
                "tag",
                // Message Body可以是任何二进制形式的数据，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
                "Hello MQ".getBytes());
        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发。
        msg.setKey("ORDERID_100");
        try {
            // 定时消息，单位毫秒（ms），在指定时间戳（当前时间之后）进行投递，例如2016-03-07 16:21
:00投递。如果被设置成当前时间戳之前的某个时刻，消息将立即被投递给消费者。
            long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2016-03-07
16:21:00").getTime();
            msg.setStartDeliverTime(timeStamp);
            // 发送消息，只要不抛异常就是成功。
            SendResult sendResult = producer.send(msg);
            System.out.println("Message Id:" + sendResult.getMessageId());
        }
        catch (Exception e) {
            // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
            System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getT
opic());
            e.printStackTrace();
        }
        // 在应用退出前，销毁Producer对象。
        // 注意：如果不销毁也没有问题。
        producer.shutdown();
    }
}
```

订阅定时消息

定时消息的订阅方式与普通消息一致，更多信息，请参见订阅消息。

2.1.14. 订阅消息

本文介绍如何通过
消息队列RocketMQ版
的Java SDK订阅消息。

订阅方式

消息队列RocketMQ版
支持以下两种订阅方式：

- 集群订阅
  同一个Group ID所标识的所有Consumer平均分摊消费消息。例如某个Topic有9条消息，一个Group ID有3个Consumer实例，那么在集群消费模式下每个实例平均分摊，只消费其中的3条消息。设置方式如下所示。

  ```
  // 集群订阅方式设置（不设置的情况下，默认为集群订阅方式）。
  properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
  ```

- 广播订阅

同一个Group ID所标识的所有Consumer都会各自消费某条消息一次。例如某个Topic有9条消息，一个Group ID有3个Consumer实例，那么在广播消费模式下每个实例都会各自消费9条消息。设置方式如下所示。

```
// 广播订阅方式设置。
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING);
```

> **说明**
>
> - 请确保同一个Group ID下所有Consumer实例的订阅关系保持一致，更多信息，请参见订阅关系一致。
> - 两种不同的订阅方式有着不同的功能限制，例如，广播模式不支持顺序消息、不维护消费进度、不支持重置消费位点等，更多信息，请参见集群消费和广播消费。

消息获取方式

消息队列Rocket MQ版
支持以下两种消息获取方式：

- Push：消息由
  消息队列Rocket MQ版
  推送至Consumer。Push方式下，
  消息队列Rocket MQ版
  还支持批量消费功能，可以将批量消息统一推送至Consumer进行消费。更多信息，请参见批量消费。
- Pull：消息由Consumer主动从
  消息队列Rocket MQ版
  拉取。

Pull Consumer提供了更多接收消息的选择。相比于Push Consumer，您可以使用Pull Consumer更加自由地控制消息拉取。具体的接口信息请参见接口和参数说明。

> **注意** 如需使用Pull Consumer，请确保您的
> 消息队列Rocket MQ版
> 实例为企业铂金版。

示例代码

具体的示例代码，请以

消息队列Rocket MQ版
代码库为准。以下分别列举Push和Pull两种消息获取方式的示例代码。

- Push方式

```
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Consumer;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import java.util.Properties;
public class ConsumerTest {
   public static void main(String[] args) {
       Properties properties = new Properties();
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
       properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
       properties.put(PropertyKeyConst.AccessKey, "XXX");
        // Accesskey Secret阿里云身份验证，在阿里云服RAM控制台创建。
       properties.put(PropertyKeyConst.SecretKey, "XXX");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
       properties.put(PropertyKeyConst.NAMESRV_ADDR, "XXX");
          // 集群订阅方式(默认)。
          // properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING)
;
          // 广播订阅方式。
          // properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTIN
G);
       Consumer consumer = ONSFactory.createConsumer(properties);
       consumer.subscribe("TopicTestMQ", "TagA||TagB", new MessageListener() { //订阅多个T
ag。
           public Action consume(Message message, ConsumeContext context) {
               System.out.println("Receive: " + message);
               return Action.CommitMessage;
           }
       });
        //订阅另外一个Topic，如需取消订阅该Topic，请删除该部分的订阅代码，重新启动消费端即可。
        consumer.subscribe("TopicTestMQ-Other", "*", new MessageListener() { //订阅全部Tag
。
           public Action consume(Message message, ConsumeContext context) {
               System.out.println("Receive: " + message);
               return Action.CommitMessage;
           }
       });
       consumer.start();
       System.out.println("Consumer Started");
   }
}
```

- Push方式（批量消费）

> **注意 配置**
> 消息队列RocketMQ版
> 的批量消费功能，请升级TCP Java SDK到1.8.7.3或以上版本，详细版本说明和获取方式，请参见Java SDK版本说明。

```
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.batch.BatchConsumer;
import com.aliyun.openservices.ons.api.batch.BatchMessageListener;
import java.util.List;
import java.util.Properties;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.tcp.example.MqConfig;
public class SimpleBatchConsumer {
    public static void main(String[] args) {
        Properties consumerProperties = new Properties();
        consumerProperties.setProperty(PropertyKeyConst.GROUP_ID, MqConfig.GROUP_ID);
        consumerProperties.setProperty(PropertyKeyConst.AccessKey, MqConfig.ACCESS_KEY);
        consumerProperties.setProperty(PropertyKeyConst.SecretKey, MqConfig.SECRET_KEY);
        consumerProperties.setProperty(PropertyKeyConst.NAMESRV_ADDR, MqConfig.NAMESRV_AD
DR);
        // 设置批量消费最大消息数量，当指定Topic的消息数量已经攒够128条，SDK立即执行回调进行消费。
默认值：32，取值范围：1~1024。
        consumerProperties.setProperty(PropertyKeyConst.ConsumeMessageBatchMaxSize, Strin
g.valueOf(128));
        // 设置批量消费最大等待时长，当等待时间达到10秒，SDK立即执行回调进行消费。默认值：0，取值范
围：0~450，单位：秒。
        consumerProperties.setProperty(PropertyKeyConst.BatchConsumeMaxAwaitDurationInSec
onds, String.valueOf(10));
        BatchConsumer batchConsumer = ONSFactory.createBatchConsumer(consumerProperties);
        batchConsumer.subscribe(MqConfig.TOPIC, MqConfig.TAG, new BatchMessageListener()
{
             @Override
            public Action consume(final List<Message> messages, ConsumeContext context) {
                System.out.printf("Batch-size: %d\n", messages.size());
                // 批量消息处理。
                return Action.CommitMessage;
            }
        });
        //启动batchConsumer。
        batchConsumer.start();
        System.out.println("Consumer start success.");
        //等待固定时间防止进程退出。
        try {
            Thread.sleep(200000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

- Pull方式

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.PullConsumer;
import com.aliyun.openservices.ons.api.TopicPartition;
import java.util.List;
import java.util.Properties;
import java.util.Set;
public class PullConsumerClient {
    public static void main(String[] args){
        Properties properties = new Properties();
        properties.setProperty(PropertyKeyConst.GROUP_ID, "GID-xxxxx");
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.AccessKey, "xxxxxxx");
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        properties.put(PropertyKeyConst.SecretKey, "xxxxxxx");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        properties.put(PropertyKeyConst.NAMESRV_ADDR, "xxxxx");
        PullConsumer consumer = ONSFactory.createPullConsumer(properties);
        // 启动Consumer。
        consumer.start();
        // 获取topic-xxx下的所有分区。
        Set<TopicPartition> topicPartitions = consumer.topicPartitions("topic-xxx");
        // 指定需要拉取消息的分区。
        consumer.assign(topicPartitions);
        while (true) {
            // 拉取消息，超时时间为3000 ms。
            List<Message> messages = consumer.poll(3000);
            System.out.printf("Received message: %s %n", messages);
        }
    }
}
```

分区和位点的详细说明，请参见基本概念。

更多信息

消息队列RocketMQ版
消费端流控的最佳实践，请参见消息队列RocketMQ客户端流控设计。

2.2. C/C++ SDK

2.2.1. 版本说明

本文通过介绍C++ SDK的版本信息，包含下载链接、发布时间、更新点等，以便您按需获取相应C++ SDK收发消息。
获取了C++ SDK后，您需按照SDK的使用说明来准备相应环境。不同版本的SDK的环境准备步骤有所不同，具体说明如下：

- 针对v2.0.0及以上版本的SDK的环境准备的具体步骤，请参见环境准备（v2.0.0）。
- 针对v1.x.x版本的SDK的环境准备以及升级至v2.0.0版本的具体步骤，请参见环境准备（v1.x.x）。

> **注意**
> 目前仅西南1（成都）、华北1（青岛）和华南1（深圳）地域支持将客户端升级为3.x.x版本，其他地域请勿将SDK升级到C++ SDK 3.x.x版本，否则将无法访问消息队列RocketMQ版服务。如有特殊需求，请提交工单咨询。

3.0.0

发布时间	发布内容	使用范围	开源代码下载	Linux版下载		CentOS下载	
				Ubuntu 18.04	Ubuntu 20.04	Debian	RPM
2021-10-18	**功能优化** • 负载均衡：以消息为粒度进行负载，负载更加均衡。 • 公网接入点：支持TCP协议公网接入点访问。 • Dashboard：新增消息堆积、消息各环节耗时、成功率等相关指标。 • 消息轨迹：新增定时和延时、事务消息及消费环节相关轨迹参数。 • 顺序消息：最大重试次数变更为16次。 • 广播消费：支持定制消费者启动时的消费位点。 • Push消费：支持消费速度限流；消费线程数异常场景逻辑优化。 • API接口变更。 **说明** 具体变更内容，请参见C++ SDK版本说明。	目前仅如下地域支该版本SDK，其他地域待开放。 • 西南1（成都） • 华北1（青岛） • 华南1（深圳）	ons-client-cpp	aliyun-mq-cpp-sdk-ubuntu18.tar.gz	aliyun-mq-cpp-sdk-ubuntu20.tar.gz	aliyun-mq-linux-cpp-sdk-centos7.2.tar.gz	aliyun-mq-linux-cpp-sdk-centos8.4.tar.gz

2.1.1

发布时间	发布内容	Windows版下载	Linux版下载				Darwin版下载
			TAR.GZ（CentOS 8）	TAR.GZ	Debian	RPM	
2020-11-26	**新特性** • 支持Windows版本（64位）的SDK。 **问题修复** • 限制嵌入式Substrate VM（SVM）heap内存使用量为64 MB，整体运行时内存300 MB。	aliyun-ons-win64-sdk.zip	aliyun-mq-linux-cpp-sdk-centos8.tar.gz	aliyun-mq-linux-cpp-sdk.tar.gz	aliyun-mq-linux-cpp-amd64.deb	aliyun-mq-linux-cpp-1.alios7.x86-64.rpm	aliyun-mq-darwin-cpp-sdk.tar.gz

2.1.0

发布时间	发布内容	Windows版下载	Linux版下载	Darwin版下载
2020-11-06	**问题修复** • 支持通过ROCKETMQ_LOG_HOME环境变量和ONSFactoryProperty::LogPath配置项来指定日志路径。 • 修复Sandy Bridge微架构下，TZCNT导致Message Body截断问题。 • 添加 `ons::Message(const std::string&topic, const std::string&body)` 构造函数，修复与v1.x.x的兼容性。	暂不支持	aliyun-mq-linux-cpp-sdk.tar.gz	aliyun-mq-darwin-cpp-sdk.tar.gz

2.0.0

发布时间	发布内容	Windows版下载	Linux版下载	Darwin版下载

发布时间	发布内容	Windows版下载	Linux版下载	Darwin版下载
2019-06-28	**新特性** • 基于Java ons v1.8.0 SDK内核，使用native-image直接生成C++ native library，功能和现有Java SDK保持一致。 • 基于ons-cpp v1.x.x接口，保持向前兼容，即兼容更早的版本。 • 无第三方依赖，启动速度更快，运行更高效。	暂不支持	aliyun-mq-linux-cpp-sdk.tar.gz	aliyun-mq-darwin-cpp-sdk.tar.gz

1.1.2

发布时间	发布内容	Windows版下载	Linux版下载

2019-01-16	**新特性**
	• 支持实例化用户使用以下两种方式接入（非实例化用户使用方式保持不变）：
	  ◦ 配置包含InstanceId的NAMESRV_ADDR方式接入使用。
	  ◦ 配置InstanceId和不包含InstanceId的NAMESRV_ADDR方式接入使用。
	• ProducerId和ConsumerId改为填写Group ID的值。
		aliyun-mq-windows-cpp-sdk.zip	aliyun-mq-linux-cpp-sdk.tar.gz

1.1.1

发布时间	发布内容	Windows版下载	Linux版下载
2018-07-31	**新特性** • 新增SSL加密传输功能（只适用于消息队列RocketMQ版企业铂金版客户）。 • PushConsumer默认使用异步拉取消息，提高推送效率。 **问题修复** • 修复顺序消息的问题。 • 日志优化，只在ReBalance结果变化时输出日志。 • 修复One-way请求system flag没有正常序列化到请求头的问题。	aliyun-mq-windows-cpp-sdk.zip	aliyun-mq-linux-cpp-sdk.tar.gz

更多历史版本

1.1.0

发布时间	发布内容	Windows版下载	Linux版下载
2017-07-25	**问题修复** • 修复 consumer shutdown 的时候导致的coredump。 • 修复底层URL类在Windows平台上无法进行HTTP访问。 • 修复消息轨迹的时间戳错误。 • 修复消息轨迹显示错误的本地IP。 • 修复Windows平台下的内存泄漏问题。	aliyun-mq-windows-cpp-sdk.zip	aliyun-mq-linux-cpp-sdk.tar.gz

1.0.9

发布时间	发布内容	Windows版下载	Linux版下载
2016-12-29	**新特性** • 增加oneway消息发送。 • 增加顺序消息。 • 新增发送超时时间设置。 • 新增发送重试次数设置。 **问题修复** • 修复shutdown时的资源泄漏问题。 • 修复shutdown时的coredump问题。	无	无

1.0.8

发布时间	发布内容	Windows版下载	Linux版下载
2016-12-02	**新特性** • 借助SWIG生成C# SDK，抛弃老的C# SDK，新版SDK对ASP.NET支持更稳定。 • 增加了自定义日志路径的功能。 • 内置中文utf-8编码，用户不需要显示的编码和解码。 • 新增MQ_GUIDE文档，添加了ASP.NET demo。 **功能优化** • 升级 boost 库到 1.6.2。 **问题修复** • 修复顺序消息退出的时候导致coredump的问题。	无	无

1.0.7

发布时间	发布内容	Windows版下载	Linux版下载
2016-11-15	**新特性** • 消费端消费限流，默认拉取1000条消息后放在内存里面，然后一条条回调用户的回调函数。 • 增加顺序消息。 • 新增发送超时时间设置。 • 新增发送重试次数设置。 **功能优化** • 消息轨迹实现优化，使用单独的线程池发送轨迹数据。 • TCP锁的粒度的优化。 **问题修复** • 修复若干消息轨迹的bug。 • 修复shutdown时的coredump问题。 • 内存泄漏的问题修复。 • 修复消息Tag包含特殊字符“||”导致异常抛出。	无	无

2.2.2. 参数说明

消息队列RocketMQ版
提供C/C++ SDK实现消息发送与订阅，您可通过本文了解消息发送和订阅相关接口的参数说明。

参数名	参数说明
NAMESRV_ADDR	设置TCP协议接入点，从消息队列RocketMQ版控制台的**实例详情**页面获取。
AccessKey	您在阿里云控制台中创建的AccessKey ID，用于身份认证。
SecretKey	您在阿里云控制台中创建的AccessKey Secret，用于身份认证。
OnsChannel	用户渠道，默认值为：ALIYUN，聚石塔用户为：CLOUD。

消息发送参数

参数名	参数说明
ProducerId	您在消息队列RocketMQ版控制台上创建的Group ID，更多信息，请参见基本概念。

参数名	参数说明
SendMsgTimeoutMillis	设置消息发送的超时时间，单位：毫秒，默认值：3000。
shardingKey（顺序消息）	顺序消息中用来计算不同分区的值。

消息订阅参数

参数名	参数说明
ConsumerId	您在消息队列RocketMQ版控制台上创建的Group ID，更多信息，请参见基本概念。
MessageModel	设置Consumer实例的消费模式，取值说明如下： • **CLUSTERING**（默认值）：表示集群消费。 • **BROADCASTING**：表示广播消费。
ConsumeThreadNums	设置Consumer实例的消费线程数，默认值：20。

更多信息

- 收发普通消息
- 收发顺序消息
- 收发定时消息
- 收发事务消息
- 订阅消息

2.2.3. 环境准备（v2.0.0）

本文介绍使用C++ SDK v2.0.0版本接入消息队列RocketMQ版所需完成的准备工作、使用说明以及注意事项，以便后续使用C++ SDK收发消息。

使用前，请注意以下几点：

- 本文仅针对C++ SDK v2.0.0版本进行说明，若您需从当前使用的v1.x.x版本的SDK升级至v2.0.0，请参见环境准备（v1.x.x）完成升级。
- 代码里涉及到的Topic和Group ID，需要到控制台上创建。Message Tag可以完全由应用自定义，更多信息，请参见创建资源。
- 使用消息队列RocketMQ版服务的应用程序需要部署在阿里云ECS上。

v2.0.0版本的C++ SDK暂时仅支持Linux操作系统，系统版本包含CentOS 6（RHEL 6）和CentOS 7（RHEL 7）系列。C++ SDK的下载链接，请参见版本说明。

下载完成后选择对应操作系统内核的版本进行解压，会有如下目录结构，各目录的说明如下：

- demos/

包含了普通消息发送、Oneway消息发送、顺序消息发送、普通消息消费、顺序消息消费等例子，还包含了 `CMakeList.txt` 用于 `demos` 的编译和管理。

- include/
  包含您自己编写的程序需要的头文件。
- lib/
  包含基于x86_64的动态库，分别为接口库（libonsclient4cpp.so）和内核库（librocketmq_client_core.so）。
- changelog
  新版本发布解决的问题和引入的新特性列表。

Linux C++ SDK动态库方案

自2019年6月28日起，新版本的SDK将只提供动态库方案。

消息队列RocketMQ版的库文件在 `lib/` 目录下，需要业务方生成可执行文件时链接 `librocketmq_client_core.so` 和 `libonsclient4cpp.so` 。由于 `demos` 引入了C++ 11的特性和使用CMake来管理，需要提前安装CMake 3.0以上版本和g++ 4.8及以上版本。

> **注意** 由于GCC 5.x及以上版本引入了Dual ABI，编译链接时，请添加 `-D_GLIBCXX_USE_CXX11_ABI=0` 编译选项。

`demos` 的使用方法如下：

```
cd aliyun-mq-linux-cpp-sdk // 下载的SDK解压后的路径。
cd demos // 进入demos目录，修改demos文件，填入自己在消息队列RocketMQ版控制台创建的Topic和Key等相关的信息。
cmake . // 检测依赖和生成编译脚本。
make // 执行编译操作。
cd bin // 到生成的可执行文件目录下运行程序。
```

更多信息

- 收发普通消息
- 收发顺序消息
- 收发定时消息
- 收发事务消息
- 订阅消息

2.2.4. 环境准备（v1.x.x）

本文为您介绍使用v1.x.x版本的C++ SDK接入消息队列RocketMQ版所需完成的准备工作、使用说明以及注意事项，以便后续使用C++ SDK收发消息。

使用前，请注意以下几点：

- 本文仅针对C++ SDK v1.x.x版本进行说明，若需使用v2.0.0版本的SDK，可选择环境准备（v2.0.0）或直接下载。v2.0.0版本的SDK使用说明请参见环境准备（v2.0.0）。
- 代码里涉及到的Topic和Group ID，需要到控制台上创建。Message Tag可以完全由应用自定义，具体创建过程请参见创建资源。
- 使用

消息队列RocketMQ版
服务的应用程序需要部署在阿里云ECS上。

SDK下载

C++ SDK支持Windows和Linux两个操作系统，而且接口完全一致。Linux下支持CentOS 6（RHEL 6）和CentOS 7（RHEL 7）系列。C++ SDK的下载链接，请参见版本说明。

下载完成后进行解压，会有如下目录结构，各目录的说明如下：

- demo/（只针对Windows系统）
  包含了一个创建好的Windows C++ 演示Demo。
- example/
  包含了普通消息发送、Oneway消息发送、顺序消息发送、普通消息消费、顺序消息消费等例子，Linux下还包含了 `Makefile` 用于 `example` 的编译和管理。
- include/
  包含了用户自己编写的程序需要的头文件。
- lib/
  - Linux SDK子目录如下，分别是64的静态库和动态库。

    ```
    lib-boost-share/
      libonsclient4cpp.so
    lib-boost-static/
      libonsclient4cpp.a
    ```

  - Windows SDK子目录如下，是64位系统下SDK的dll库。如果没有安装Visual Studio 2015环境下，需要拷贝安装 `vc_redist.x64` 。 这是Visual C++ 2015的运行时环境。

    ```
    64/
    vc_redist.x64
    ```

- SDK_GUIDE.pdf
  SDK环境准备文档和FAQ。
- changelog
  新版本发布解决的问题和引入的新特性列表。

Linux C++ SDK

自2016年12月02日开始，Linux CPP版本依赖了高性能boost库（1.62.0版本），不仅降低了CPU资源占用率，而且提高了运行效率。目前主要依赖了 `boost_system` 、 `boost_thread` 、 `boost_chrono` 、 `boost_filesystem` 四个库，有静态库和动态库两种解决方案。

静态解决方案

消息队列RocketMQ版的库文件在 `lib/lib-boost-static` 目录下，boost库静态链接到 `libonsclient4cpp.a` 中。对于没有依赖boost库的业务方，可以直接选用静态库方案。静态库方案中，相应的boost库已经链接到 `libonsclient4cpp.a` ，编译时只需要链接 `libonsclient4cpp.a` 即可，无需执行其他操作。使用方式如下。

```
cd aliyun-mq-linux-cpp-sdk  //下载的SDK解压后的路径
cd example                  //进入example目录，修改example文件，填入您创建的Topic和Key等相关的信息
make static=1
```

> **注意** 完全的静态链接请确保机器上安装了libstdc++、pthread等相关的静态库。默认安装的libstdc++ 是没有安装静态库的，所以需要通过 `yum` 或者 `> apt-get` 来安装相关的静态库。

此外使用如上方式会出现一些警告信息如下。

```
warning: Using 'gethostbyaddr' in statically linked applications requires at runtime the sh
ared libraries from the glibc version used for linking
```

建议最佳的方式，不要使用完全的静态链接，而是只静态链接onsclient4cpp，其他库动态链接即可。使用方式如下。

```
g++ -ggdb -Wall -O3 -I../include    ../example/ProducerExampleForEx.cpp  -Wl,-static -lonsc
lient4cpp -L../lib/lib-boost-static/ -Wl,-Bdynamic  -lpthread -ldl -lrt -o ../example/Produ
cerExampleForEx
```

此外，由于GCC 5.x引入Dual ABI，编译链接时，请添加**-D_GLIBCXX_USE_CXX11_ABI=0**编译选项。

动态解决方案

消息队列RocketMQ版的库文件在 `lib/lib-boost-share` 目录下，需要业务方生成可执行文件时链接boost动态库和 `libonsclient4cpp.so` 。对于业务方已经依赖了boost库，需要选择动态库方案的情况，对boost库的依赖需要做如下工作：

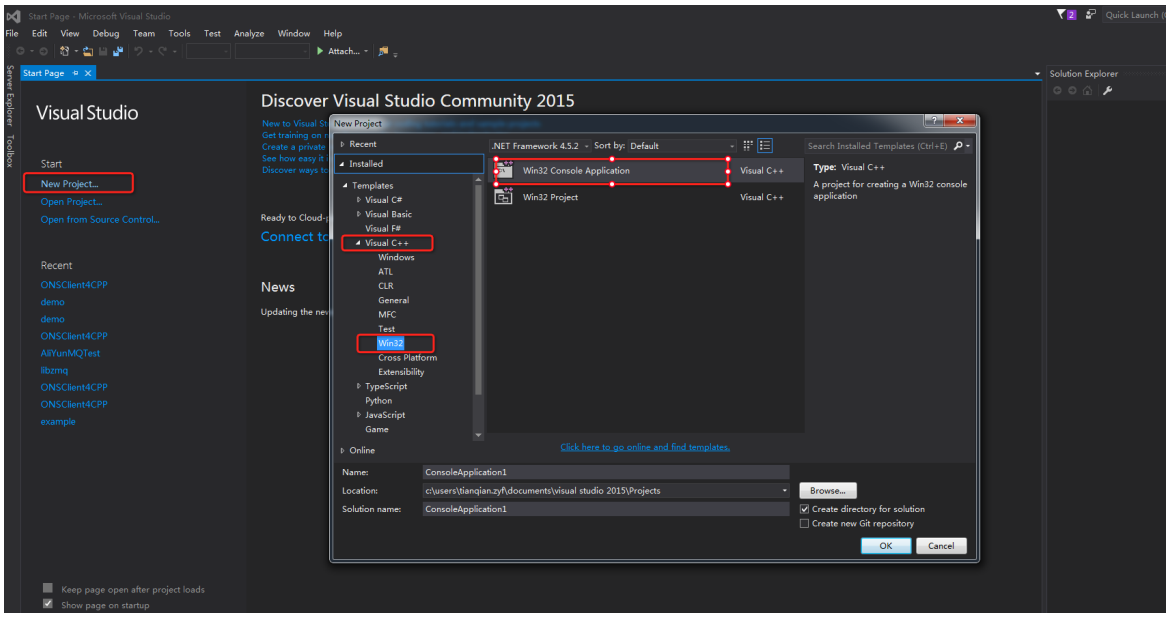
1. 下载boost 1.62.0版本。
   boost 1.62.0
2. 解压boost 1.62.0。
   `tar --bzip2 -xf /path/to/boost_1_62_0.tar.bz2`
3. 安装boost 1.62.0版本：
   i. 进入boost 1.62.0解压后的路径： `cd path/to/boost_1_62_0`
   ii. 配置boost： `./bootstrap.sh`
   iii. 编译boost： `./b2 link=shared runtime-link=shared`
   iv. 安装boost： `./b2 install`
4. 执行 `ldconfig -v|grep libboost` 。如果有相关的输出，则表明boost动态库在动态库搜索路径中。
5. 生成可执行文件时，需要链接boost动态库和消息队列RocketMQ版的动态库。方法如下。

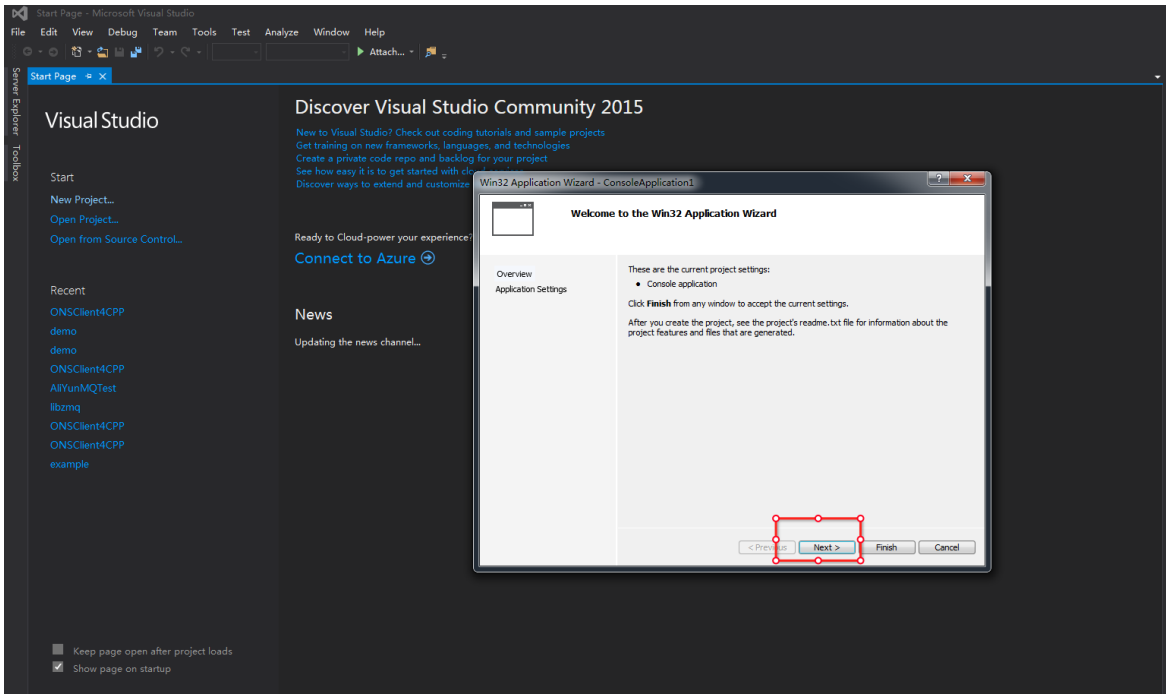
```
cd aliyun-mq-linux-cpp-sdk  //下载的SDK解压后的路径
cd example                  //进入example目录，修改example文件，填入自己在消息队列RocketMQ版控制台创建的Topic和Key等相关的信息
g++ -Wall -Wno-deprecated -L ../lib/lib-boost-share/ -I ../include/ ProducerExampleForE
x.cpp  -lonsclient4cpp -lboost_system -lboost_thread -lboost_chrono -lboost_filesystem
-lpthread
export LD_LIBRARY_PATH="../lib/lib-boost-share/"  //添加动态载入的搜索路径
./a.out  //运行程序
```

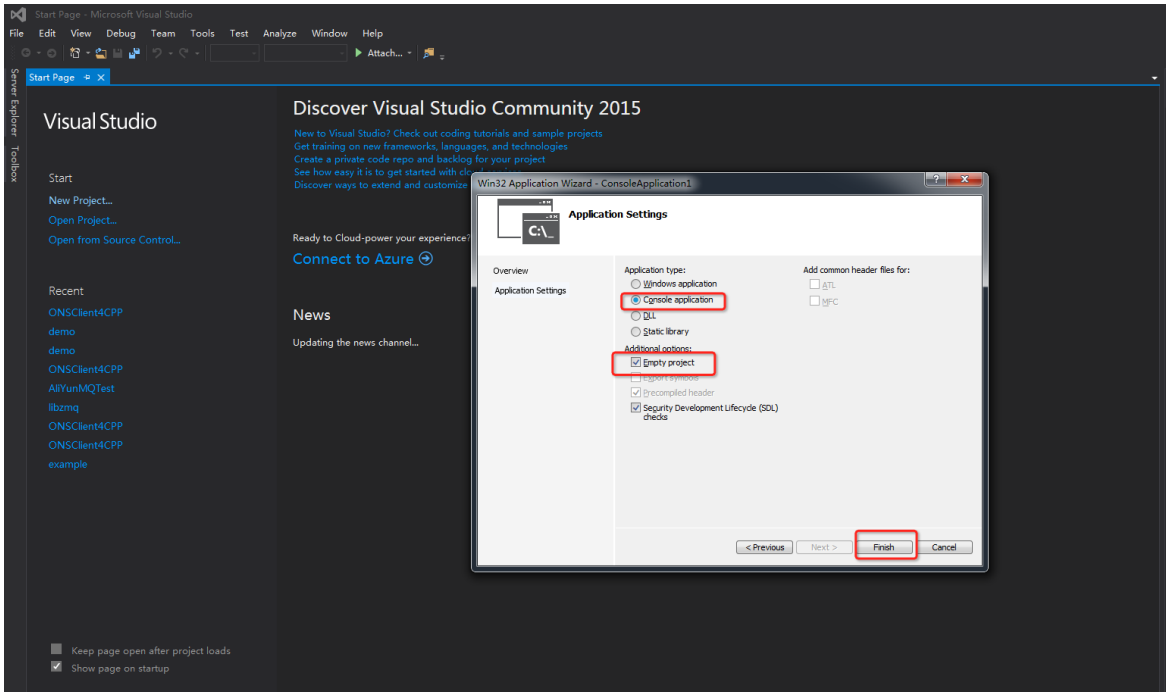
Windows C++ SDK

Visual Studio 2015环境下使用C++ SDK

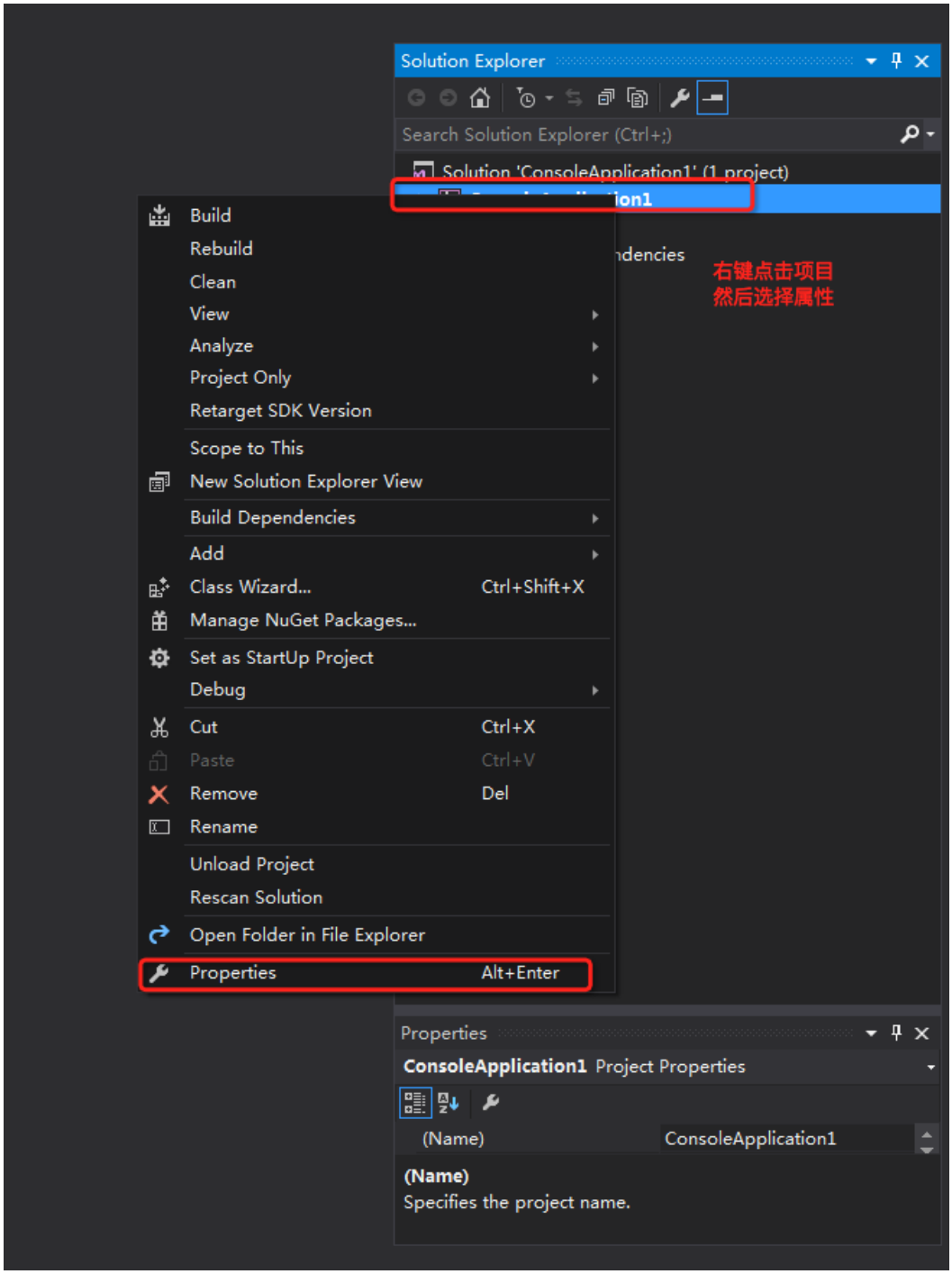
1. 使用Visual Studio 2015创建自己的项目。

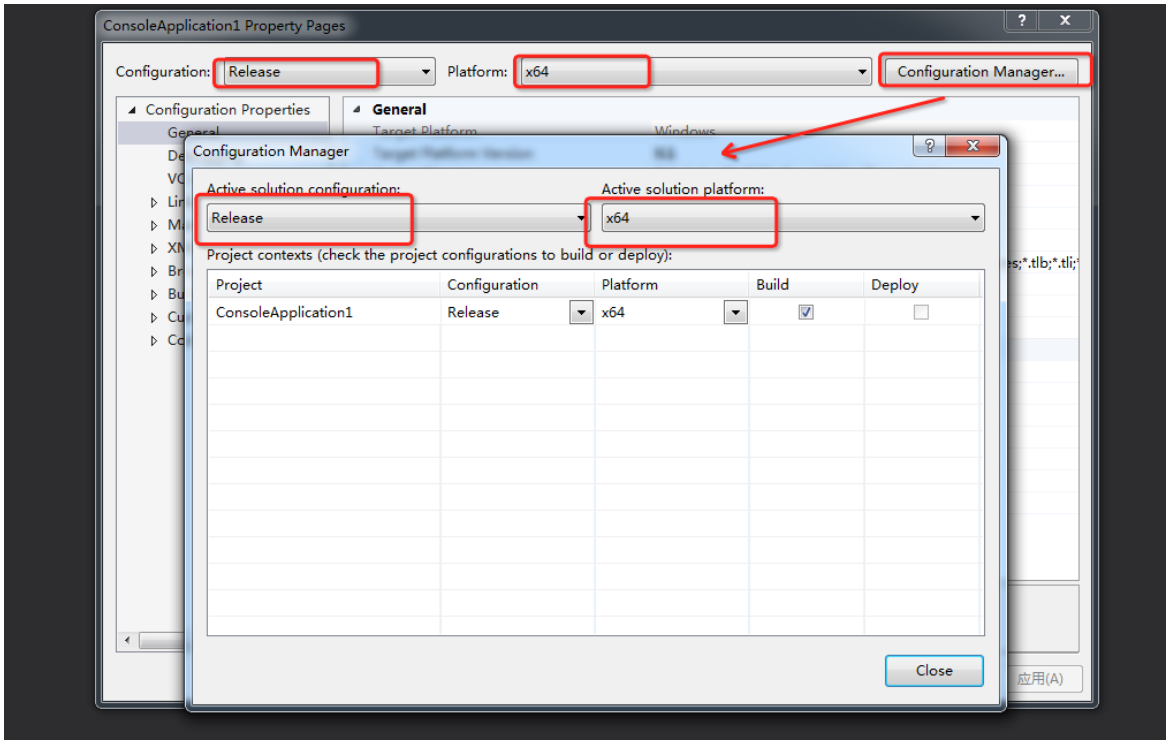




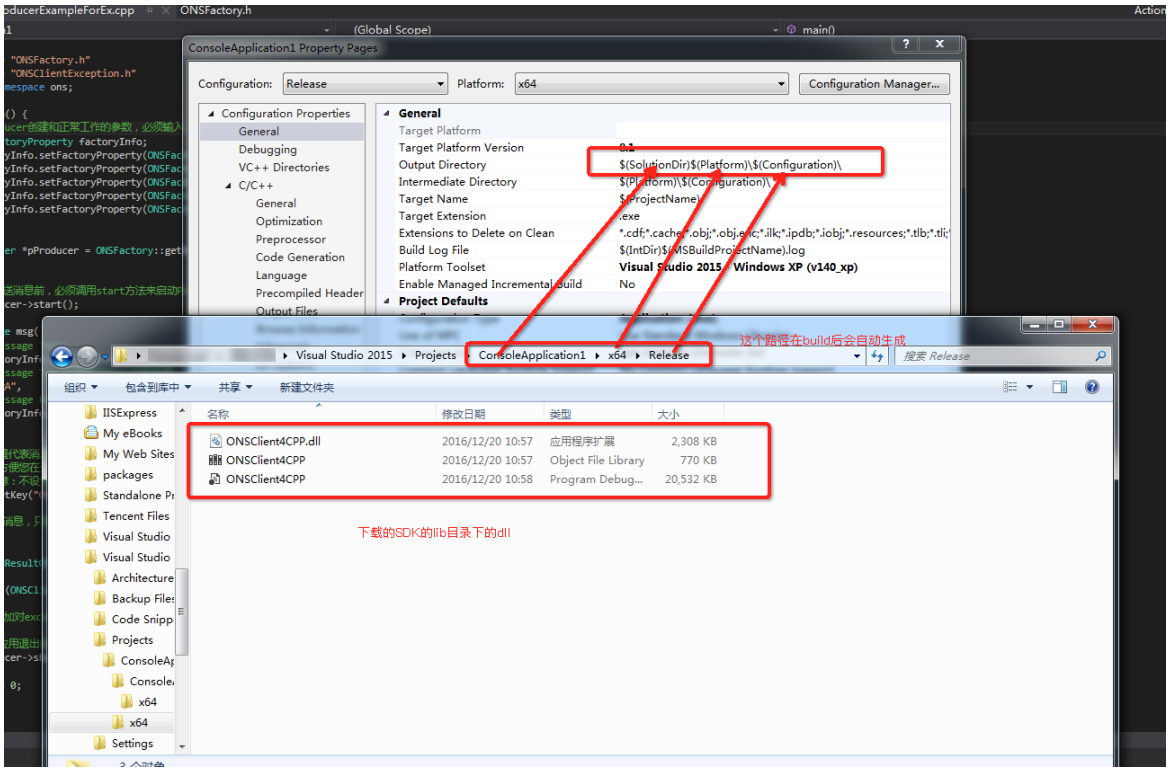


2. 右键单击项目，选择**属性 > 配置管理器**，设置活动解决方案配置为**release**，设置活动解决方案平台为**x64**。

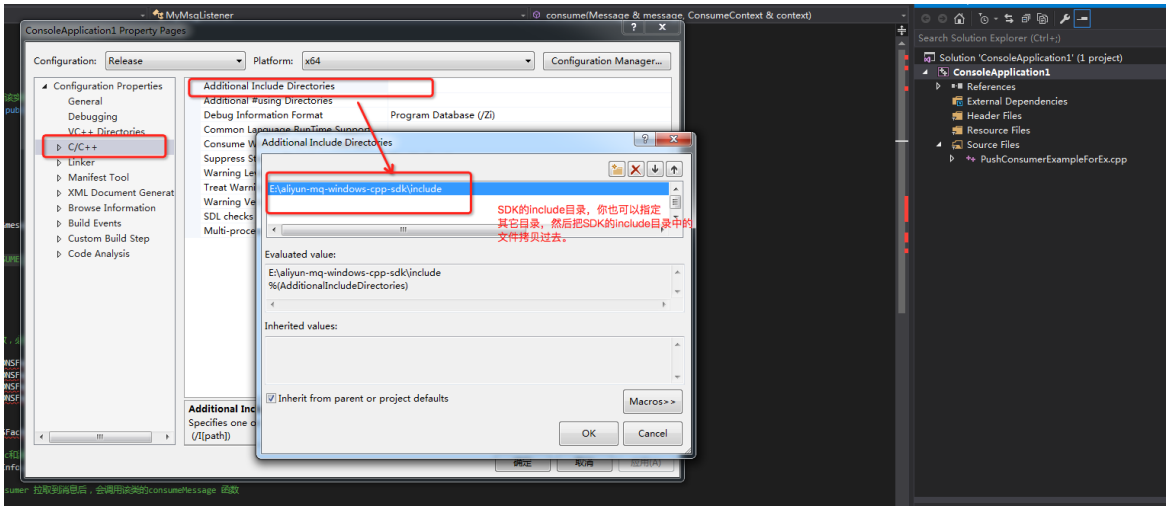

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApplication1' (1 project)
Build
Rebuild
Clean
View
Analyze
Project Only
Retarget SDK Version
Scope to This
New Solution Explorer View
Build Dependencies
Add
Class Wizard... Ctrl+Shift+X
Manage NuGet Packages...
Set as StartUp Project
Debug
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Rescan Solution
Open Folder in File Explorer
Properties Alt+Enter
右键点击项目
然后选择属性
Properties
ConsoleApplication1 Project Properties
(Name) ConsoleApplication1
(Name)
Specifies the project name.

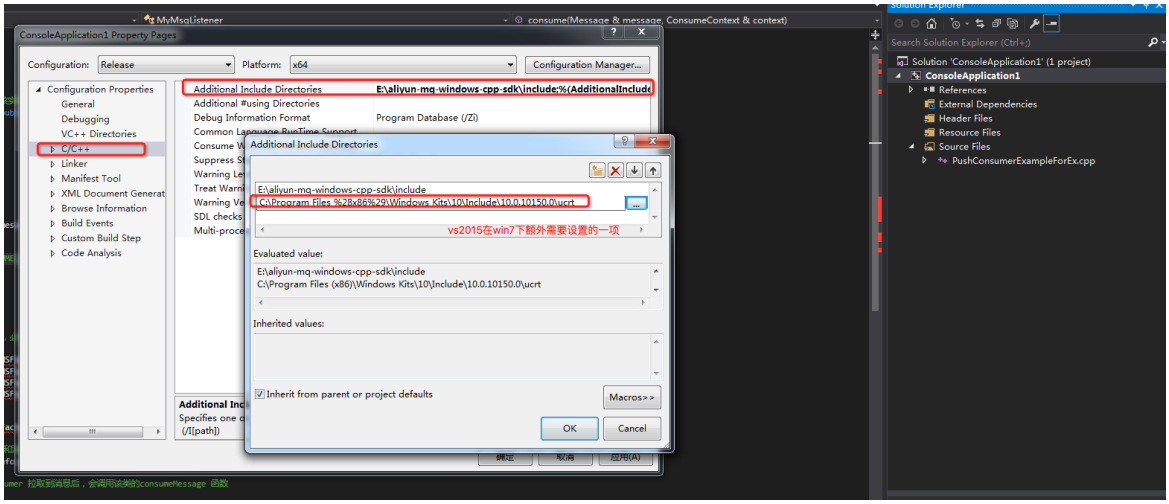


3. 右键单击项目，依次选择**属性 > 配置属性 > 常规 > 输出目录：/A**。按照**活动解决方案平台**的设置，拷贝64位lib目录下的所有文件到输出目录**/A**。

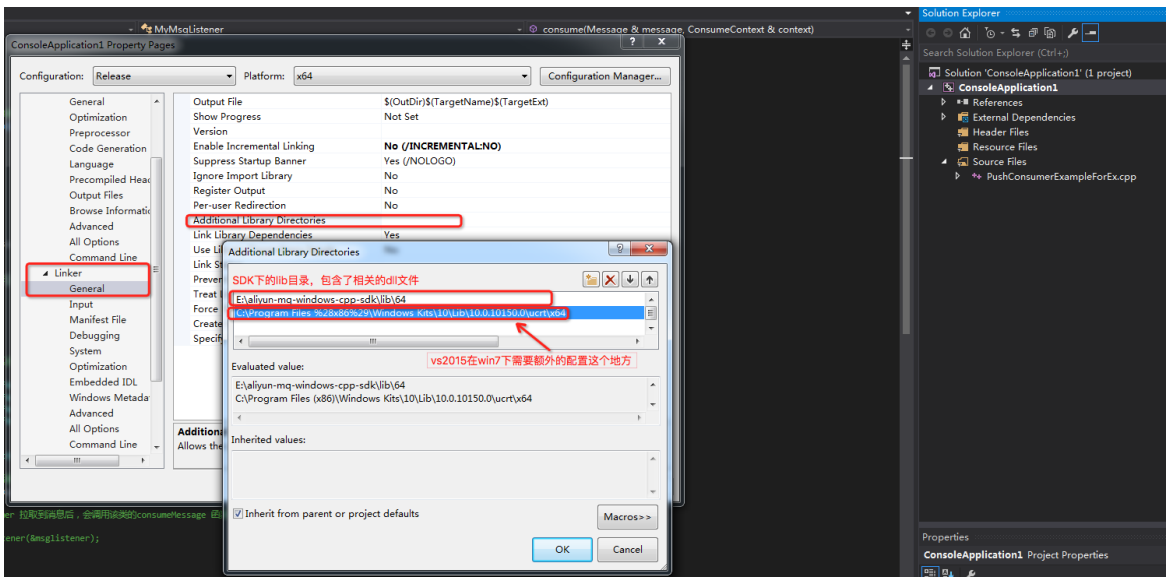


4. 右键单击项目，依次选择**属性 > 配置属性 > C/C++-常规 > 附加包含目录：/B**。拷贝 `include` 目录下的头文件到包含目录：**/B**。

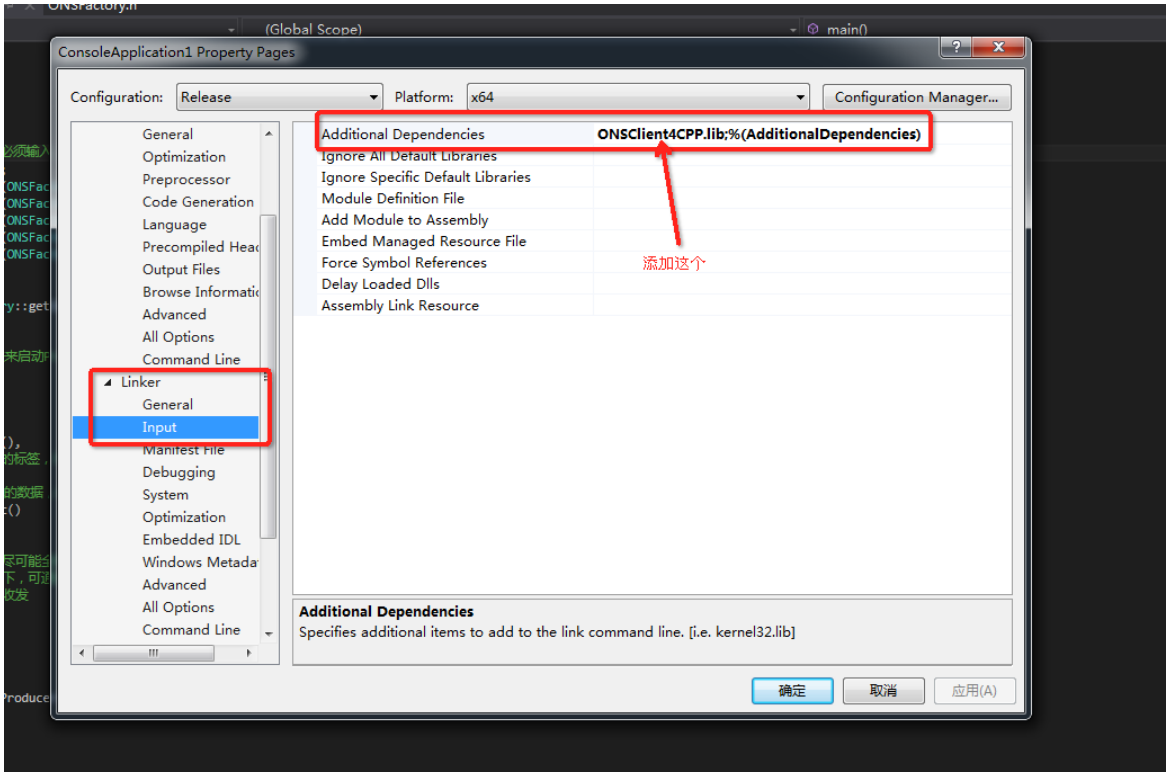




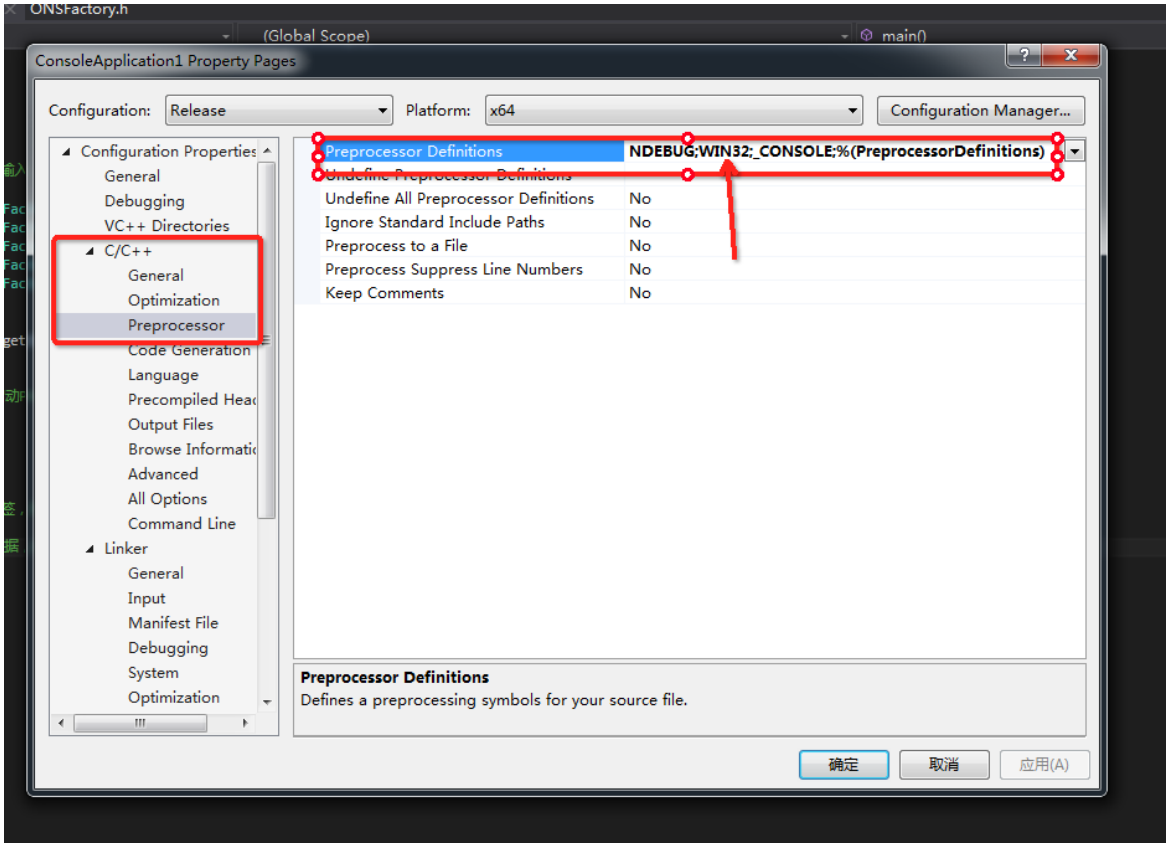
5. 右键单击项目，依次选择**属性 > 配置属性 > 链接 > 常规 > 附加库目录：/A**。



6. 右键单击项目，依次选择**属性 > 配置属性 > 链接 > 输入 > 附加依赖项：ONSClient4CPP.lib**。



7. 右键单击项目，依次选择**属性 > 配置属性 > C/C++-常规 > 预处理器定义**：添加**WIN32**宏。

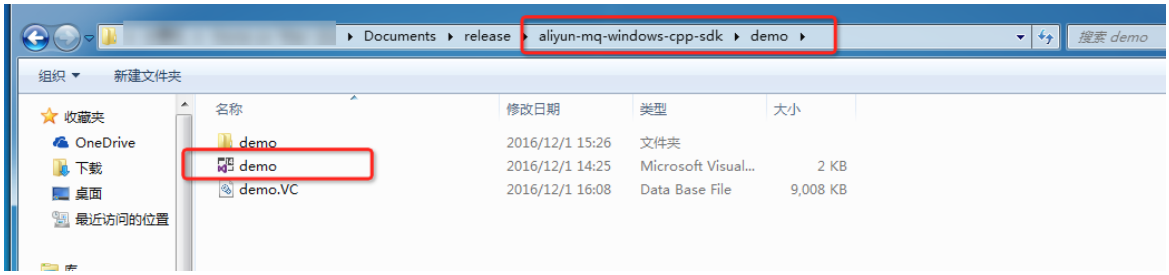


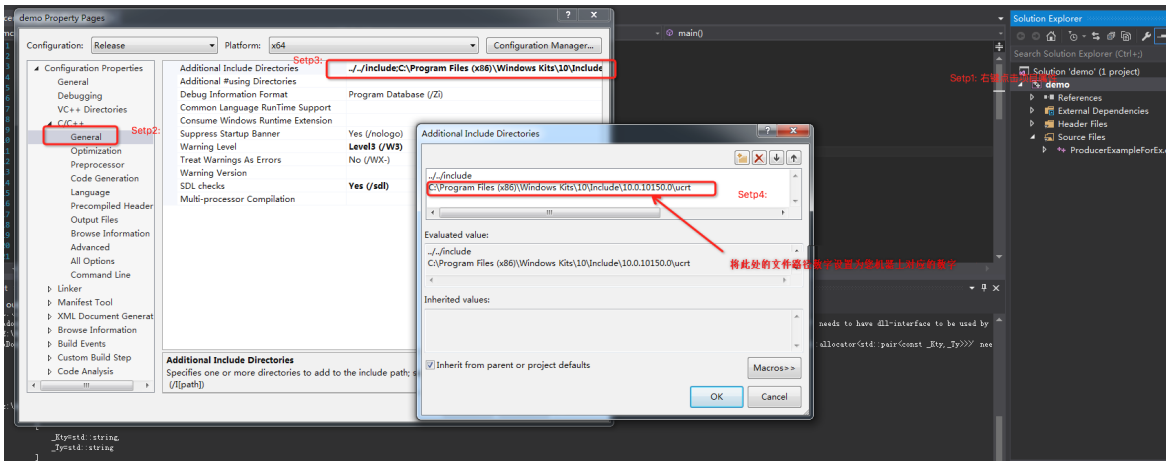
Visual Studio非2015环境下使用C++ SDK

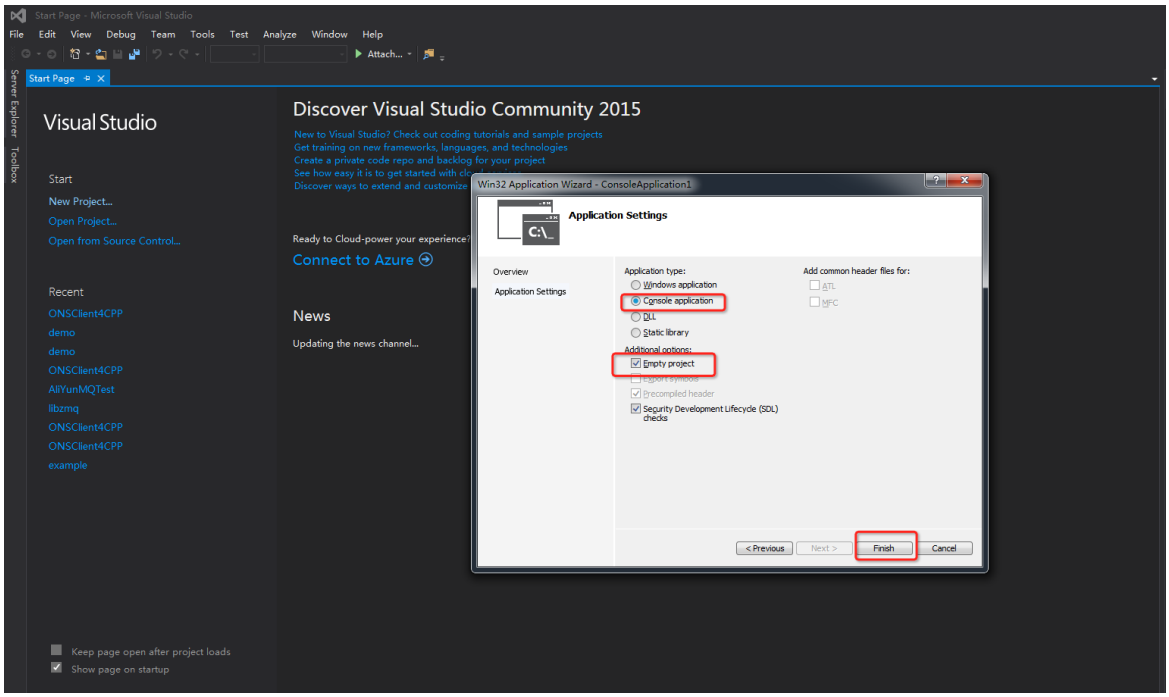
1. 首先需要按照Visual Studio 2015的环境来配置，配置过程同Visual Studio 2015环境下使用C++ SDK。
2. 安装**vc_redist.x64**。这是**Visual C++ 2015**的运行时环境。

> 注意 如果不想进行繁琐的设置，您也可以使用设置好的SDK Demo，下载SDK后进行解压，然后进入demo目录，使用Visual Studio 2015打开工程。

示例如下：







到此为此就设置好编译环境了。单击**Build**，即可编译出可执行的程序，然后拷贝dll到可执行程序的目录下即可运行，或者拷贝dll到系统目录下。

升级v1.x.x版本的SDK至v2.0.0

目前最新版的v2.0.0 SDK和v1.x.x SDK保持API向前兼容（即新版本兼容旧版本），由于实现方式不同，ABI是不兼容的。

请按照以下步骤升级：

1. 将新版SDK的头文件替换老版本的头文件。
2. 将新版本 `lib` 目录下的动态库全部拷贝到存放原版本动态库的目录下。
3. 如果之前使用的是静态链接方案，去掉 `-static` 参数，修改为默认动态路链接方案。
4. 修改编译脚本，增加 `-lrocketmq_client_core` 参数，链接新版内核动态库。
5. 重新编译工程。
6. 若升级失败，则可选择回退，将工程恢复至升级前状态，重新编译即可。

更多信息

- 收发普通消息
- 收发顺序消息
- 收发定时消息
- 收发事务消息
- 订阅消息

2.2.5. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。不同消息类型所属的Topic不能混用，例如收发普通消息的Topic不能用来收发其他类型的消息。本文提供使用TCP协议下的C/C++ SDK收发普通消息的示例代码供您参考。

前提条件

您已完成以下操作：

- 下载C/C++ SDK。更多信息，请参见版本说明。
- 准备环境。请根据您使用的SDK版本做好环境准备工作：
  - 环境准备（v2.0.0）
  - 环境准备（v1.x.x）

发送普通消息

请参考以下示例代码进行消息发送。

```
#include "ONSFactory.h"
#include "ONSClientException.h"
using namespace ons;
int main()
{
    //创建Producer，并配置发送消息所必需的信息。
    ONSFactoryProperty factoryInfo;
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId, "XXX");//您在
消息队列RocketMQ版控制台创建的Group ID。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX"); //设置TCP接入域
名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );// 您在
消息队列RocketMQ版控制台创建的Topic。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//消息内容。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//AccessKey ID阿里
云身份验证，在阿里云服务器管理控制台创建。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "XXX" );//AccessKey Secre
t阿里云身份验证，在阿里云服务器管理控制台创建。
    //create producer;
    Producer *pProducer = ONSFactory::getInstance()->createProducer(factoryInfo);
    //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
    pProducer->start();
    Message msg(
            //普通消息所属的Topic，切勿使用普通消息的Topic来收发其他类型的消息。
            factoryInfo.getPublishTopics(),
            //Message Tag，可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在
消息队列RocketMQ版的服务器过滤。
            "TagA",
            //Message Body，不能为空，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
            factoryInfo.getMessageContent()
    );
    // 设置代表消息的业务关键属性，请尽可能全局唯一。
    // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
    // 注意：不设置也不会影响消息正常收发。
    msg.setKey("ORDERID_100");
    //发送消息，只要不抛出异常，就代表发送成功。
    try
    {
        SendResultONS sendResult = pProducer->send(msg);
    }
    catch(ONSClientException & e)
    {
        //自定义处理exception的细节。
    }
    // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题。
    pProducer->shutdown();
    return 0;
}
```

订阅普通消息

订阅普通消息的说明和示例代码，请参见订阅消息。

2.2.6. 收发顺序消息

顺序消息FIFO（ First In First Out ）是
消息队列Rocket MQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用TCP协议下的C/C++ SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息2.0。

前提条件

您已完成以下操作：

- 下载C/C++ SDK。更多信息，请参见版本说明。
- 准备环境。请根据您使用的SDK版本做好环境准备工作：
  - 环境准备（v2.0.0）
  - 环境准备（v1.x.x）

发送顺序消息

> **注意**
> 消息队列Rocket MQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列Rocket MQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

```
#include "ONSFactory.h"
#include "ONSClientException.h"
#include <iostream>
using namespace ons;
int main()
{
    //Producer创建和正常工作的参数，必须输入。
    ONSFactoryProperty factoryInfo;
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId, "XXX");//您在
消息队列RocketMQ版控制台创建的Group ID。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX"); //设置TCP接入域
名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
```

```
    factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//您在
消息队列RocketMQ版控制台创建的Topic。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//消息内容。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//AccessKey ID阿里
云身份验证，在阿里云服务器管理控制台创建。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "XXX" );//AccessKey Secre
t阿里云身份验证，在阿里云服务器管理控制台创建。
    //创建Producer。
    OrderProducer *pProducer = ONSFactory::getInstance()->createOrderProducer(factoryInfo);
    //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
    pProducer->start();
    Message msg(
                //Message Topic
                factoryInfo.getPublishTopics(),
                //Message Tag，可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在
消息队列RocketMQ版的服务器过滤。
                "TagA",
                //Message Body，任何二进制形式的数据，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
                factoryInfo.getMessageContent()
    );
    // 设置代表消息的业务属性，请尽可能全局唯一。
    // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
    // 注意：不设置也不会影响消息正常收发。
    msg.setKey("ORDERID_100");
    // 分区顺序消息中区分不同分区的关键字段。
    // 如果是全局顺序消息，该字段可以设置为任意非空字符串。
    std::string shardingKey = "abc";
    //带有同一Sharding Key的消息会按照顺序发送。
    try
    {
        //发送消息，只要不抛异常就是成功。
        SendResultONS sendResult = pProducer->send(msg, shardingKey);
    std::cout << "send success" << std::endl;
    }
    catch(ONSClientException & e)
    {
        //添加对exception的处理。
    }
    // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题。
    pProducer->shutdown();
    return 0;
}
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
#include "ONSFactory.h"
using namespace std;
using namespace ons;
//创建消费消息的实例。
//pushConsumer拉取到消息后，会主动调用该实例的consumeMessage函数。
class ONSCLIENT_API MyMsgListener : public MessageOrderListener
{
public:
    MyMsgListener()
    {
    }
    virtual ~MyMsgListener()
    {
    }
    virtual OrderAction consume(Message &message, ConsumeOrderContext &context)
    {
        //根据业务需求，消费消息。
        return Success; //CONSUME_SUCCESS;
    }
};
int main(int argc, char* argv[])
{
    //OrderConsumer创建和工作需要的参数，必须输入。
    ONSFactoryProperty factoryInfo;
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "XXX");//您在
消息队列RocketMQ版控制台创建的Group ID。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//您在
消息队列RocketMQ版控制台创建的Topic。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//AccessKey ID阿里
云身份验证，在阿里云服务器管理控制台创建。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey,  "XXX");//AccessKey Secre
t阿里云身份验证，在阿里云服务器管理控制台创建。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX");//设置TCP接入域名
，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    //创建orderConsumer。
    OrderConsumer* orderConsumer = ONSFactory::getInstance()->createOrderConsumer(factoryIn
fo);
    MyMsgListener  msglistener;
    //指定orderConsumer订阅的消息Topic和消息Tag。
    orderConsumer->subscribe(factoryInfo.getPublishTopics(), "*",&msglistener );
    //注册消息监听的处理实例，orderConsumer拉取到消息后，会调用该类的consumeMessage函数。
    //启动orderConsumer。
    orderConsumer->start();
    for(volatile int i = 0; i < 1000000000; ++i) {
        //wait
    }
    //销毁orderConsumer，在应用退出前，必须销毁Consumer对象，否则会导致内存泄露等问题。
    orderConsumer->shutdown();
   return 0;
}
```

2.2.7. 收发定时消息

本文提供使用TCP协议下的C/C++ SDK收发定时消息的示例代码供您参考。
定时消息可以做到在指定时间戳之后才可被消费者消费，适用于对消息生产和消费有时间窗口要求，或者利用消息触发定时任务的场景。

定时消息的概念介绍及使用过程中的注意事项，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 下载C/C++ SDK。更多信息，请参见版本说明。
- 准备环境。请根据您使用的SDK版本做好环境准备工作：
  - 环境准备（v2.0.0）
  - 环境准备（v1.x.x）

发送定时消息

发送定时消息的示例代码如下。

```
#include "ONSFactory.h"
#include "ONSClientException.h"
#include <windows.h>
using namespace ons;
int main()
{
    //创建Producer，并配置发送消息所必需的信息。
    ONSFactoryProperty factoryInfo;
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId, "XXX");//您在
消息队列RocketMQ版控制台创建的Group ID。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX"); //设置TCP接入域
名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//您在
消息队列RocketMQ版控制台创建的Topic。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "xxx");//消息内容。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxx");//AccessKey ID阿里
云身份验证，在阿里云服务器管理控制台创建。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "xxx" );//AccessKey Secre
t阿里云身份验证，在阿里云服务器管理控制台创建。
    //create producer;
    Producer *pProducer = ONSFactory::getInstance()->createProducer(factoryInfo);
    //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
    pProducer->start();
    Message msg(
            //Message Topic
            factoryInfo.getPublishTopics(),
            //Message Tag，可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在
消息队列RocketMQ版的服务器过滤
            "TagA",
            //Message Body，不能为空，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
            factoryInfo.getMessageContent()
    );
    // 设置代表消息的业务关键属性，请尽可能全局唯一
```

```
    // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
    // 注意：不设置也不会影响消息正常收发。
    msg.setKey("ORDERID_100");
    // deliver time单位ms，指定一个时刻，在这个时刻之后消息才能被消费，这个例子表示3s后才能被消费。
    long deliverTime = GetTickCount64() + 3000;
    msg.setStartDeliverTime(deliverTime);
    //发送消息，只要不抛出异常，就代表发送成功。
    try
    {
        SendResultONS sendResult = pProducer->send(msg);
    }
    catch(ONSClientException & e)
    {
        //自定义处理exception的细节。
    }
    // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题。
    pProducer->shutdown();
    return 0;
}
```

订阅定时消息

定时消息的订阅与普通消息订阅一致，更多信息，请参见订阅消息。

2.2.8. 收发事务消息

本文提供使用TCP协议下的C/C++ SDK收发事务消息的示例代码供您参考。

消息队列RocketMQ版

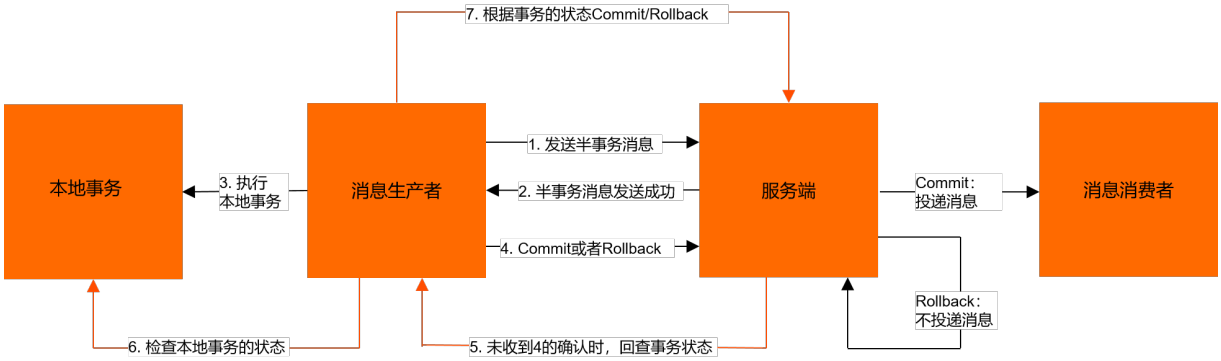
提供类似XA或Open XA的分布式事务功能，通过

消息队列RocketMQ版

事务消息，能达到分布式事务的最终一致。

交互流程

事务消息交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 下载C/C++ SDK。更多信息，请参见版本说明。
- 准备环境。请根据您使用的SDK版本做好环境准备工作：
  - 环境准备（v2.0.0）
  - 环境准备（v1.x.x）

发送事务消息

发送事务消息包含以下两个步骤。

1. 发送半事务消息（Half Message）及执行本地事务。示例代码如下。

```
#include "ONSFactory.h"
#include "ONSClientException.h"
using namespace ons;
    class MyLocalTransactionExecuter : LocalTransactionExecuter
    {
        MyLocalTransactionExecuter()
        {
        }
        ~MyLocalTransactionExecuter()
        {
        }
        virtual TransactionStatus execute(Message &value)
        {
                // 消息ID(有可能消息体一样，但消息ID不一样，当前消息ID在
消息队列RocketMQ版控制台无法查询。)
                string msgId = value.getMsgID();
                // 消息体内容进行crc32，也可以使用其它的如MD5。
                // 消息ID和crc32id主要是用来防止消息重复。
                // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等。
                // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或MD5来防止重复消息。
                TransactionStatus transactionStatus = Unknow;
                try {
                    boolean isCommit = 本地事务执行结果;
                    if (isCommit) {
                        // 本地事务成功、提交消息。
                        transactionStatus = CommitTransaction;
                    } else {
                        // 本地事务失败、回滚消息。
                        transactionStatus = RollbackTransaction;
                    }
                } catch (...) {
                    //exception handle
                }
                return transactionStatus;
        }
    }
    int main(int argc, char* argv[])
    {
        //创建Producer和发送消息所必需的信息。
        ONSFactoryProperty factoryInfo;
        factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId,"XXX");//您在
消息队列RocketMQ版控制台创建的Group ID。
```

```
        factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX"); //设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//输入您在
消息队列RocketMQ版控制台创建的Topic。
        factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//消息Content。
        factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxxxxxxxx");//AccessKey ID阿里云身份验证，在阿里云服务器管理控制台创建。
        factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "xxxxxxxxxxxxxxxxxxxx" );//AccessKey Secret阿里云身份验证，在阿里云服务器管理控制台创建。
        //创建producer，
消息队列RocketMQ版不负责pChecker的释放，需要业务方自行释放资源。
        MyLocalTransactionChecker *pChecker = new MyLocalTransactionChecker();
        g_producer = ONSFactory::getInstance()->createTransactionProducer(factoryInfo,pChecker);
        //在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
        pProducer->start();
        Message msg(
            //Message Topic
            factoryInfo.getPublishTopics(),
            //Message Tag，可理解为Gmail中的标签，对消息进行再归类，方便Consumer指定过滤条件在
消息队列RocketMQ版的服务器过滤。
            "TagA",
            //Message Body，不能为空，
消息队列RocketMQ版不做任何干预，需要Producer与Consumer协商好一致的序列化和反序列化方式。
            factoryInfo.getMessageContent()
        );
        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发。
        msg.setKey("ORDERID_100");
        //发送消息，只要不抛出异常，就代表发送成功。
        try
        {
            //
消息队列RocketMQ版不负责pExecuter的释放，需要业务方自行释放资源。
            MyLocalTransactionExecuter pExecuter = new MyLocalTransactionExecuter();
            SendResultONS sendResult = pProducer->send(msg,pExecuter);
        }
        catch(ONSClientException & e)
        {
            //自定义处理exception的细节。
        }
        // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题。
        pProducer->shutdown();
        return 0;
    }
```

2. 提交事务消息状态。
   当本地事务执行完成（执行成功或执行失败），需要通知服务器当前消息的事务状态。通知方式有以下两种：

- 执行本地事务完成后提交。
- 执行本地事务一直没提交状态，等待服务器回查消息的事务状态。

事务状态有以下三种：

- `TransactionStatus.CommitTransaction`：提交事务，允许订阅方消费该消息。
- `TransactionStatus.RollbackTransaction`：回滚事务，消息将被丢弃不允许消费。
- `TransactionStatus.Unknow`：无法判断状态，期待消息队列RocketMQ版的Broker向发送方再次询问该消息对应的本地事务的状态。

提交事务消息状态的示例代码如下。

```
class MyLocalTransactionChecker : LocalTransactionChecker
{
    MyLocalTransactionChecker()
    {
    }
    ~MyLocalTransactionChecker()
    {
    }
    virtual TransactionStatus check(Message &value)
    {
        // 消息ID（有可能消息体一样，但消息ID不一样，当前消息ID在消息队列RocketMQ版控制台无法查询。）
        string msgId = value.getMsgID();
        // 消息体内容进行crc32，也可以使用其它的如MD5。
        // 消息ID和crc32id主要是用来防止消息重复。
        // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等。
        // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或MD5来防止重复消息。
        TransactionStatus transactionStatus = Unknow;
        try {
            boolean isCommit = 本地事务执行结果;
            if (isCommit) {
                // 本地事务成功、提交消息。
                transactionStatus = CommitTransaction;
            } else {
                // 本地事务失败、回滚消息。
                transactionStatus = RollbackTransaction;
            }
        } catch(...) {
            //exception error
        }
        return transactionStatus;
    }
}
```

事务回查机制说明

- 发送事务消息为什么必须要实现回查Check机制？
  当步骤1中半事务消息发送完成，但本地事务返回状态为 `TransactionStatus.Unknow`，或者应用退出导致本地事务未提交任何状态时，从Broker的角度看，这条半事务消息的状态是未知的。因此Broker会定期向消息发送方即消息生产者集群中的任意一生产者实例发起消息回查，要求发送方回查该Half状态消息，并上报其最终状态。

- Check被回调时，业务逻辑都需要做些什么？
  事务消息的Check方法里面，应该写一些检查事务一致性的逻辑。
  消息队列Rocket MQ版
  发送事务消息时需要实现 `LocalTransactionChecker` 接口，用来处理Broker主动发起的本地事务状态回查请求，因此在事务消息的Check方法中，需要完成两件事情：
  i. 检查该半事务消息对应的本地事务的状态（committed or rollback）。
  ii. 向Broker提交该半事务消息本地事务的状态。

订阅事务消息

事务消息的订阅与普通消息订阅一致，更多信息，请参见订阅消息。

2.2.9. 订阅消息

本文介绍如何通过
消息队列Rocket MQ版
的C/C++ SDK订阅消息。

订阅方式

消息队列Rocket MQ版
支持以下两种订阅方式：

- 集群订阅
  同一个Group ID所标识的所有Consumer平均分摊消费消息。例如某个Topic有9条消息，一个Group ID有3个Consumer实例，那么在集群消费模式下每个实例平均分摊，只消费其中的3条消息。设置方式如下所示。

```
// 集群订阅方式设置（不设置的情况下，默认为集群订阅方式）
factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::CLUSTERING);
```

- 广播订阅
  同一个Group ID所标识的所有Consumer都会各自消费某条消息一次。例如某个Topic有9条消息，一个Group ID有3个Consumer实例，那么在广播消费模式下每个实例都会各自消费9条消息。设置方式如下所示。

```
// 广播订阅方式设置
factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::BROADCASTING);
```

> **说明**
> - 请确保同一个Group ID下所有Consumer实例的订阅关系保持一致，更多信息，请参见订阅关系一致。
> - 两种不同的订阅方式有着不同的功能限制，例如，广播模式不支持顺序消息、不维护消费进度、不支持重置消费位点等，更多信息，请参见集群消费和广播消费。

示例代码

```
#include "ONSFactory.h"
#include <iostream>
#include <thread>
#include <mutex>
```

```
using namespace ons;
std::mutex console_mtx;
class ExampleMessageListener : public MessageListener {
public:
    Action consume(Message& message, ConsumeContext& context) {
        //此处为具体的消息处理过程，确认消息被处理成功请返回CommitMessage。
        //如果有消费异常，或者期望重新消费，可以返回ReconsumeLater，消息将会在一段时间后重新投递。
        std::lock_guard<std::mutex> lk(console_mtx);
        std::cout << "Received a message. Topic: " << message.getTopic() << ", MsgId: "
        << message.getMsgID() << std::endl;
        return CommitMessage;
    }
};
int main(int argc, char* argv[]) {
    std::cout << "=======Before consuming messages=======" << std::endl;
    ONSFactoryProperty factoryInfo;
    //请填写在阿里云
消息队列RocketMQ版控制台上申请的Group ID，从实例化的版本开始，ProducerId和CounsumerId已经统一，此处
设置是为了接口保持向前兼容。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "GID_XXX");
    //请填写您的阿里云账号的AccessKey ID。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "Your Access Key");
    //请填写您的阿里云账号的AccessKey Secret。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "Your Secret Key");
    //请填写阿里云
消息队列RocketMQ版控制台上对应实例的接入点。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR,
                                   "http://xxxxxxxxxxxxxxx.aliyuncs.com:80");
    PushConsumer *consumer = ONSFactory::getInstance()->createPushConsumer(factoryInfo);
    //请填写阿里云
消息队列RocketMQ版控制台上申请的Topic。
    const char* topic_1 = "topic-1";
    // 订阅topic-1中Tag消息属性为tag-1的所有消息。
    const char* tag_1 = "tag-1";
    const char* topic_2 = "topic-2";
    // 订阅topic-2的所有消息。
    const char* tag_2 = "*";
    //请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
    ExampleMessageListener * message_listener = new ExampleMessageListener();
    consumer->subscribe(topic_1, tag_1, message_listener);
    consumer->subscribe(topic_2, tag_2, message_listener);
    //准备工作完成，必须调用启动函数，才可以正常工作。
    consumer->start();
    //请保持线程常驻，不要执行shutdown操作。
    std::this_thread::sleep_for(std::chrono::milliseconds(60 * 1000));
    consumer->shutdown();
    delete message_listener;
    std::cout << "=======After consuming messages======" << std::endl;
    return 0;
}
```

更多信息

消息队列RocketMQ版
消费端流控的最佳实践，请参见
消息队列RocketMQ版
客户端流控设计。

2.3. .NET SDK

2.3.1. 版本说明

本文通过介绍.NET SDK的版本信息，包含各版本的发布时间、发布内容以及下载链接，以便您按需获取相应.NET SDK收发消息。

2.0.1

发布时间	发布内容	Windows版下载	Linux版下载		Darwin版下载
			Ubuntu版本	CentOS版本	
2021-03-31	**问题修复** • 修改*Quick Start*文档，添加使用x64目标库的提示说明。 • 修复API兼容性，修正 `Message#setBody(byte[] body, int size)` 方法。	aliyun-mq-windows-net-sdk.zip	aliyun-mq-linux-net-sdk.tar.gz	aliyun-mq-linux-net-sdk.tar.gz	aliyun-mq-darwin-net-sdk.tar.gz

2.0.0

发布时间	发布内容	Windows版下载	Darwin版下载
2021-03-24	**功能优化** • 通过GraalVM，将功能点和Java与C/C++ SDK对齐。 • 解决共享库稳定性问题。	aliyun-mq-windows-net-sdk.zip	aliyun-mq-darwin-net-sdk.tar.gz

1.1.4

发布时间	发布内容	Windows版下载
2019-11-06	**功能优化** • 支持查询顺序消息的消费轨迹。 • 优化消息拉取流程，避免特殊情况下拉取异常造成的消息堆积。 • 优化顺序消息重试间隔。 • 修复顺序消息重试时，重试次数不准确的问题。 • 修复某些条件下客户端生成Message ID重复的问题。	aliyun-mq-windows-net-sdk.zip

1.1.3

发布时间	发布内容	Windows版下载

2019-02-01	**功能优化**
	• 支持实例化用户使用以下两种方式接入（非实例化用户使用方式保持不变）：
	  ○ 配置包含InstanceId的NAMESRV_ADDR方式接入使用。
	  ○ 配置InstanceId和不包含InstanceId的NAMESRV_ADDR方式接入使用。
	• ProducerId和ConsumerId改为填写Group ID的值。
		aliyun-mq-windows-net-sdk.zip

1.1.2

发布时间	发布内容	Windows版下载
2018-10-24	**功能优化** • Demo中提供中文推荐编解码方式。 **问题修复** • 修复顺序消息消费的问题。	aliyun-mq-windows-net-sdk.zip

更多历史版本

1.1.1

发布时间	发布内容	旧Windows版下载	新Windows版下载
2018-01-09	**新特性** • 新增发送Byte消息的接口。 **问题修复** • 修复消息轨迹获取IP错误的问题。	无（不再维护）	aliyun-mq-windows-net-sdk.zip

1.1.0

发布时间	发布内容	旧Windows版下载	新Windows版下载
2017-07-31	**问题修复** • 修复consumer shutdown的时候导致的coredump。 • 修复底层URL类在Windows平台上无法进行HTTP访问。 • 修复消息轨迹的时间戳错误。 • 修复消息轨迹显示错误的本地IP地址。 • 修复Windows平台下的内存泄漏问题。	无（不再维护）	aliyun-mq-windows-net-sdk.zip

2.3.2. 参数说明

消息队列RocketMQ版
提供.NET SDK实现消息发送与订阅，您可通过本文了解消息发送和订阅相关接口的参数说明。

参数名	参数说明
NAMESRV_ADDR	设置TCP协议接入点，从 消息队列RocketMQ版 控制台的**实例详情**页面获取。
AccessKey	您在阿里云账号管理控制台中创建的AccessKey ID，用于身份认证。
SecretKey	您在阿里云账号管理控制台中创建的AccessKey Secret，用于身份认证。
OnsChannel	用户渠道，默认值为：ALIYUN，聚石塔用户为：CLOUD。

消息发送参数

参数名	参数说明
ProducerId	您在 消息队列RocketMQ版 控制台上创建Group ID的更多信息，请参见基本概念。
SendMsgTimeoutMillis	设置消息发送的超时时间，单位：毫秒，默认值：3000。
shardingKey（顺序消息）	顺序消息中用来计算不同分区的值。

消息订阅参数

参数名	参数说明
ConsumerId	您在 消息队列RocketMQ版 控制台上创建Group ID的更多信息，请参见基本概念。
MessageModel	设置Consumer实例的消费模式，取值说明如下： • CLUSTERING（默认值）：表示集群消费。 • BROADCASTING：表示广播消费。
ConsumeThreadNums	设置Consumer实例的消费线程数，默认值：20。

更多信息

- 收发普通消息
- 收发顺序消息
- 收发定时消息
- 收发事务消息

- 订阅消息

2.3.3. 环境准备

本文介绍使用TCP协议的.NET SDK方式接入
消息队列RocketMQ版
的环境准备工作，以便您后续使用TCP协议的.NET SDK收发消息。
使用前，请注意以下几点：

- 使用
  消息队列RocketMQ版
  服务的应用程序需要部署在阿里云ECS上。
- 代码里涉及到的Topic和Group ID，需要到控制台上创建。Message Tag可以完全由应用自定义。创建步骤，请参见快速入门中的创建资源。

Windows .NET SDK简介

阿里云提供的.NET版本是基于

消息队列RocketMQ版
的CPP版本的托管封装，这样能保证.NET完全不依赖于Windows.NET公共库。内部采用C++多线程并发处理，保证.NET版本的高效稳定。
在使用Visual Studio（VS）开发.NET的应用程序和类库时，默认的目标平台为“Any CPU”，即运行时可根据CPU类型自动选择x86或x64。在运行时，CLR才会将其JIT发射为x86或x64的机器码。而C/C++编译生成的DLL就是机器码。所以，其平台的决策是在编译时决定的。通过编译选项的设置，将C/C++项目编译为x64的64位DLL，因此提供了包含VS2015和.NET Framework 4.5.2编译的release64位版本DLL。其他VS版本也可以使用。

> **注意**
>
> - .NET SDK仅支持Windows 64-bit操作系统。
> - C++ DLL文件需要Visual C++ 2015运行时环境安装包。如果没有安装Visual Studio 2015运行时环境，请执行SDK中提供的vc_redist.x64.exe来完成安装。

下载Windows .NET SDK

新用户或者不考虑升级成本的老用户请下载新版SDK。最新版本的.NET SDK下载链接，请参见版本说明。

下载完成后进行解压，会有以下目录结构，各目录的说明如下：

- lib/
  底层的C++ DLL相关文件，以及Visual C++ 2015运行时环境安装包。如果没有安装Visual Studio 2015运行时环境，需要拷贝安装vc_redist.x64.exe，如下所示。

```
64/
    ONSClient4CPP.lib
    ONSClient4CPP.dll
    ONSClient4CPP.pdb
  vc_redist.x64.exe
```

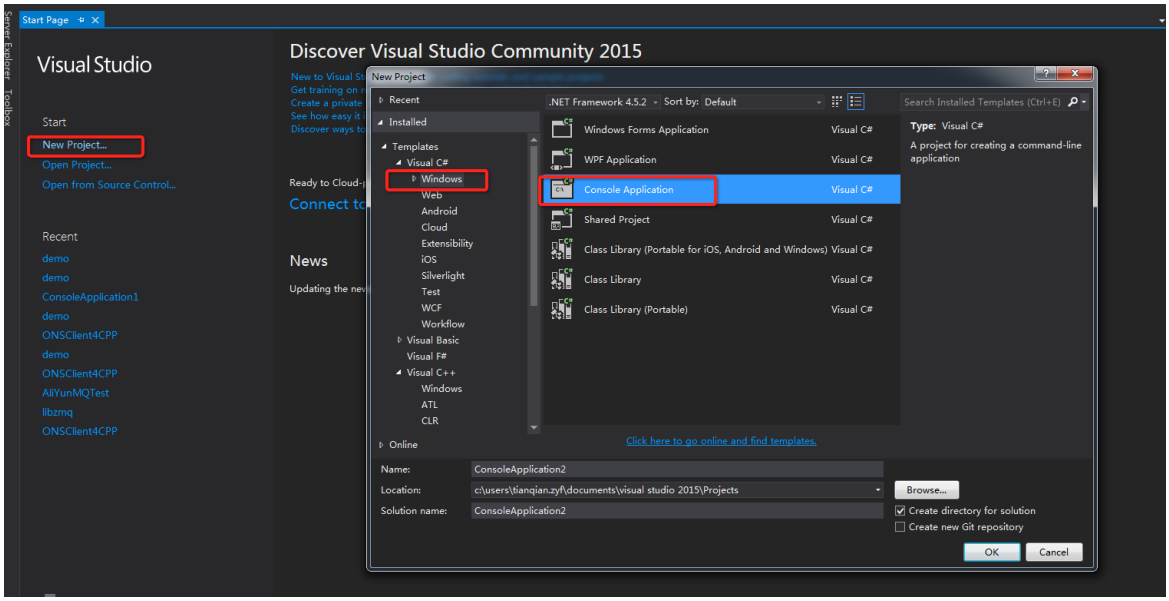
- demo/
  包含了普通消息发送、Oneway消息发送、顺序消息发送、普通消息消费、顺序消息消费等代码示例。
- interface/
  封装PINVOKE调用的代码，需要包含到您的项目代码中。
- SDK_GUIDE.pdf

SDK环境准备文档和FAQ。

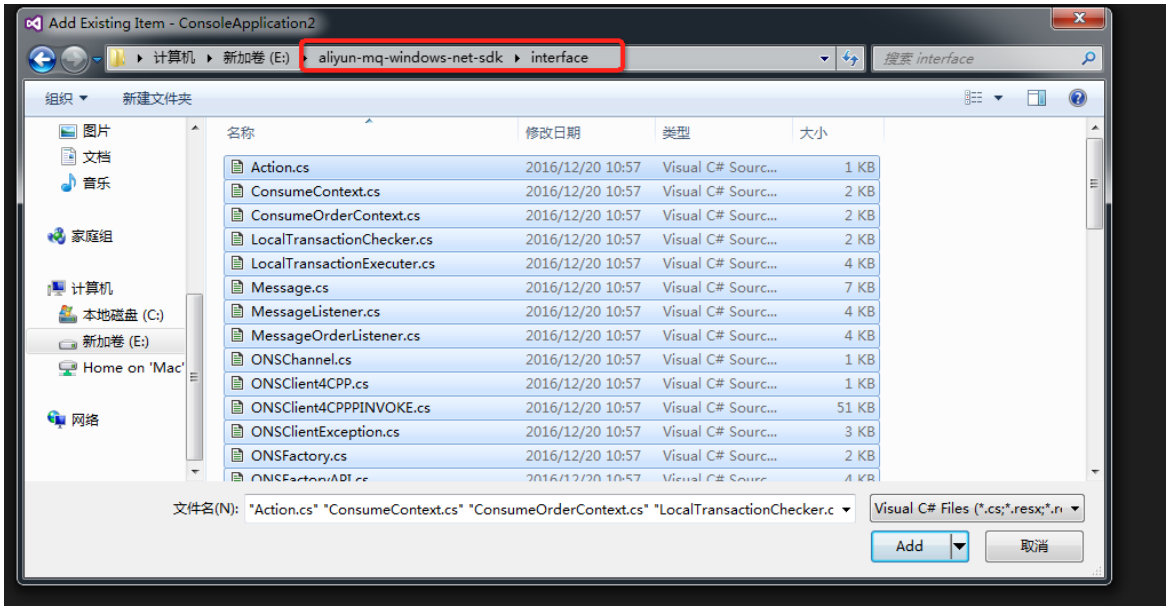
- changelog
  新版本发布解决的问题和引入的新特性列表。

Visual Studio 2015使用.NET SDK配置说明

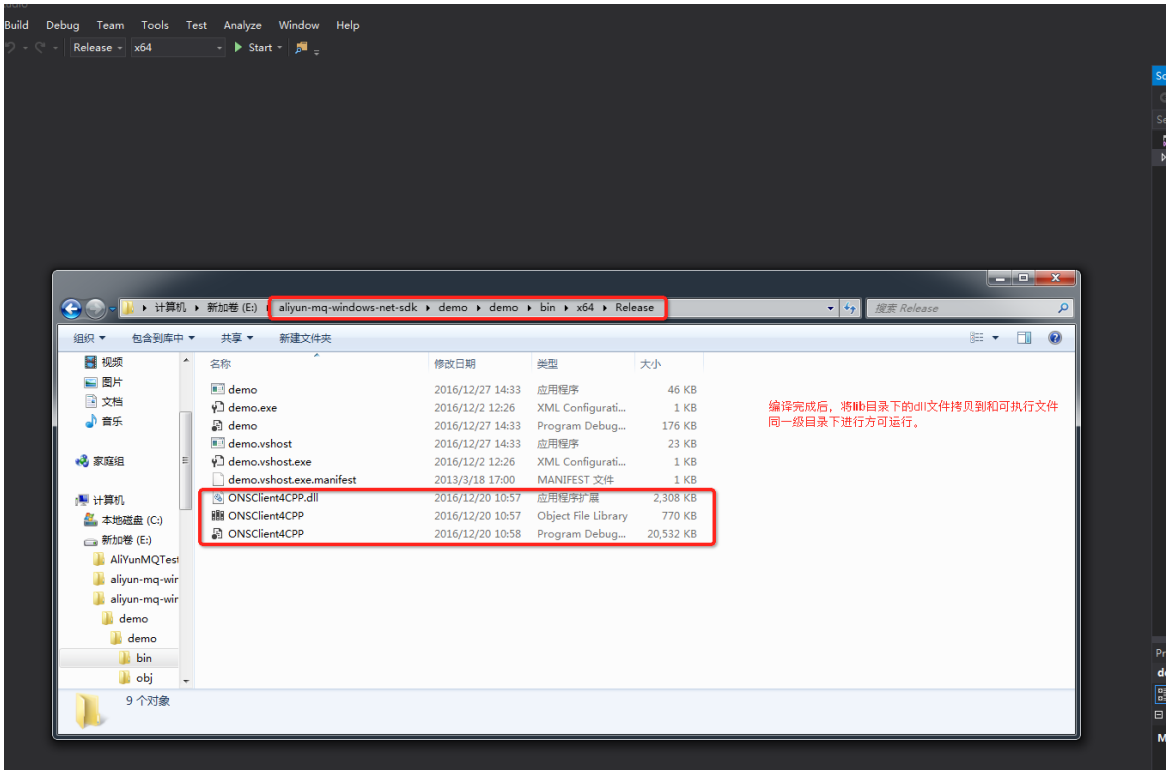
1. 使用Visual Studio 2015创建自己的项目。



2. 右键单击项目选择**添加 > 现存在项**，将下载的SDK中的**interface**目录下的所有文件都添加进去。

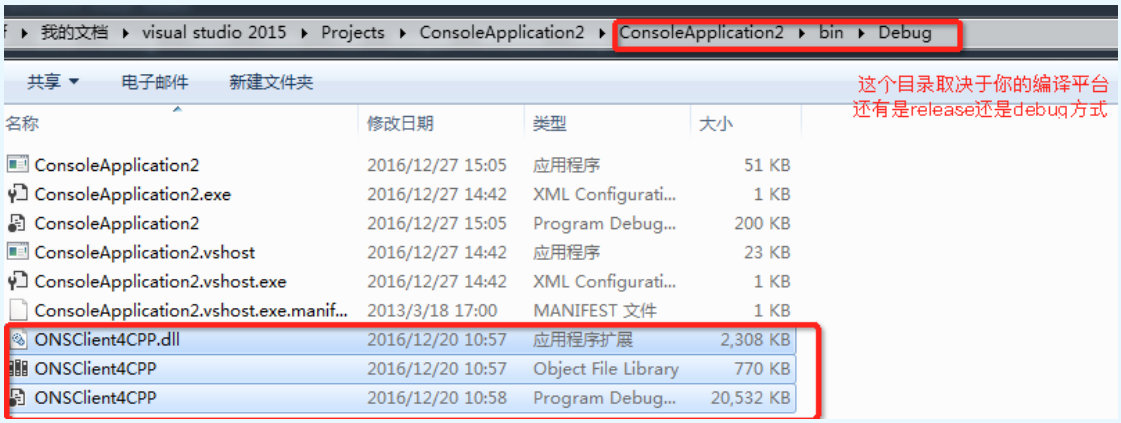


3. 右键单击项目选择**属性 > 配置管理器**。设置**活动解决方案配置**为**release**；设置**活动解决方案平台**为**x64**。
4. 编写测试程序，然后进行编译，最后将SDK下的DLL放到和可执行文件同一级目录下，或者系统目录下即可运行。



说明

SDK提供了设置好的Demo，直接打开工程进行编译即可。运行的时候将相关的DLL文件拷贝到和可执行文件同级目录下，如下图。

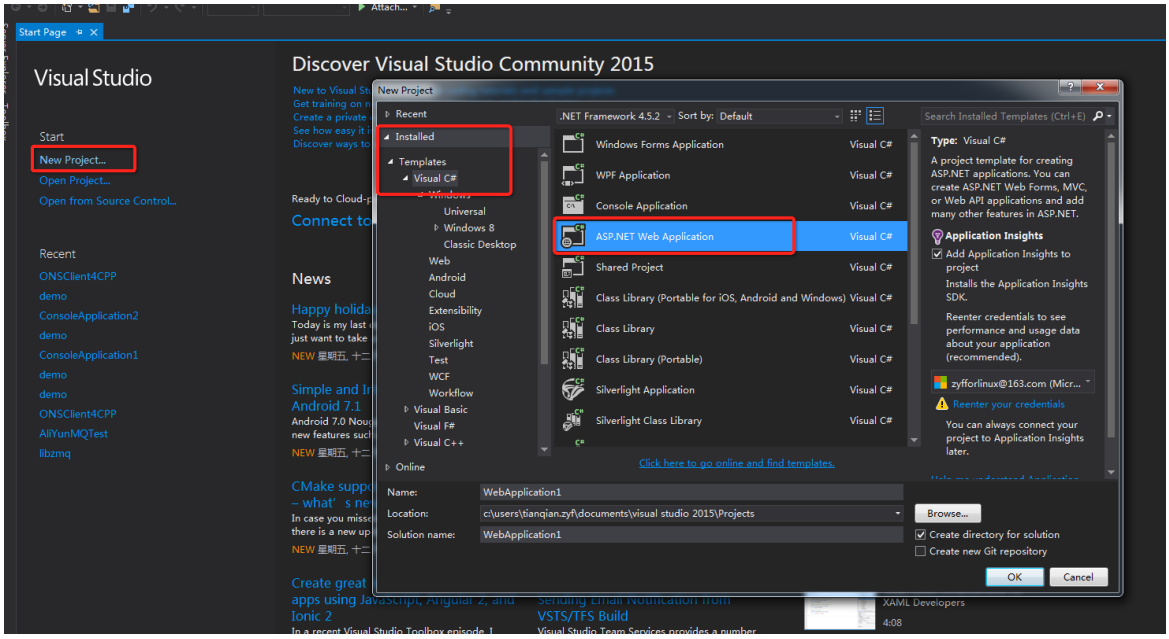


Visual Studio 2015配置ASP.NET使用

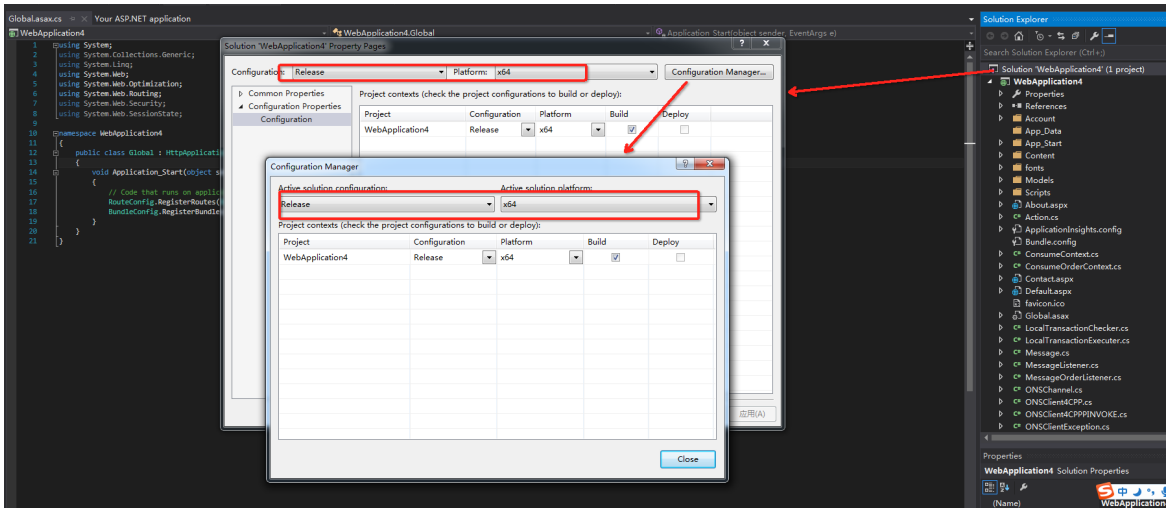
消息队列RocketMQ版

SDK

1. 使用VS2015创建一个ASP.NET的Web Forms项目。



2. 右键单击项目选择**属性 > 配置管理器**。设置**活动解决方案配置**为release；设置**活动解决方案平台**为**x64**。

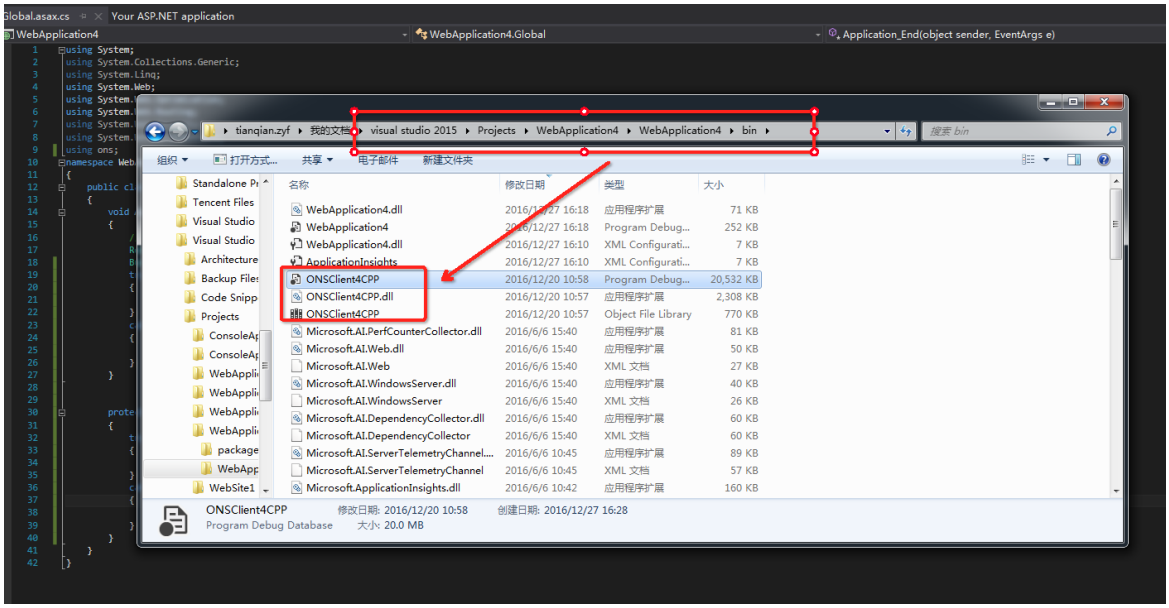


3. 右键单击项目选择**添加 > 现存在项**，将下载的SDK中的interface目录下的所有文件都添加进去。请参考上文中配置普通的.NET项目的步骤2。
4. 在*Global.asax.cs*文件中添加启动和关闭SDK的代码。

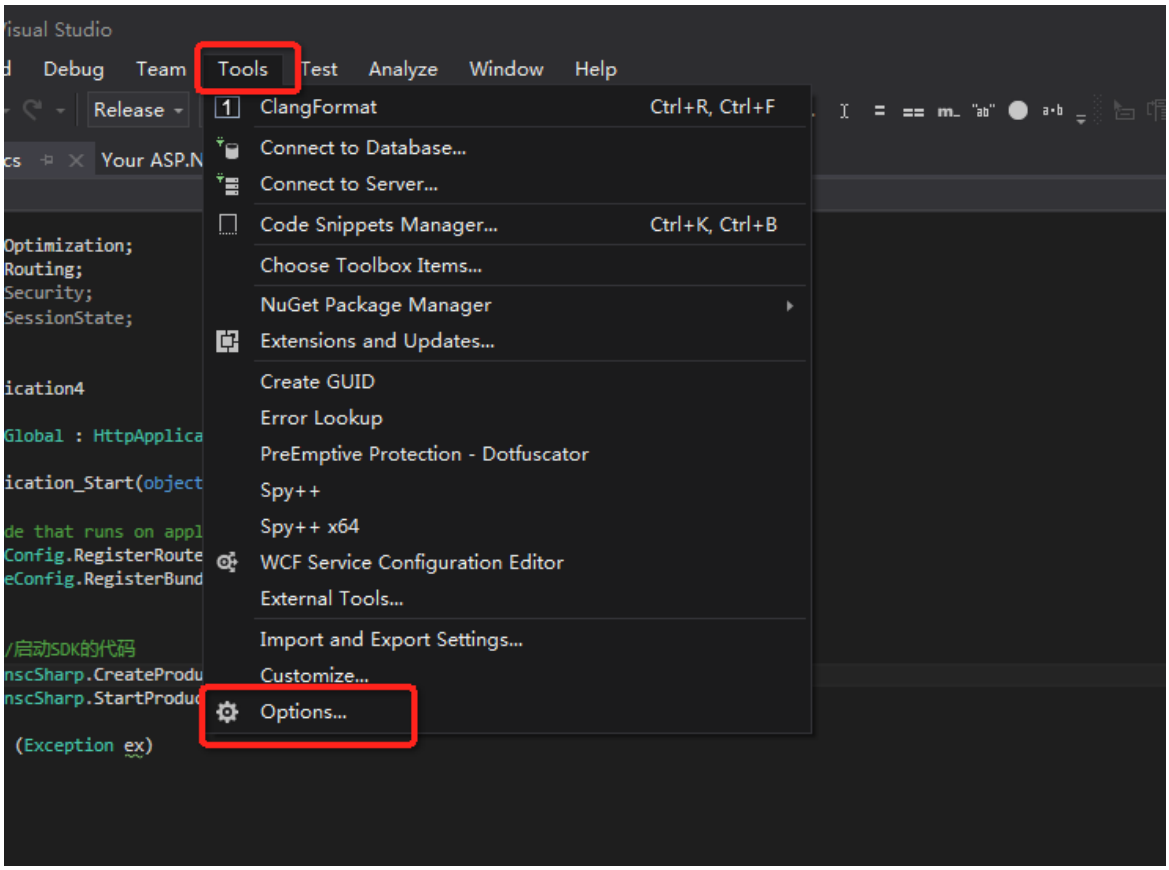
> **注意** 建议将SDK的代码封装成一个单例类，这样可以避免因为作用域的问题被垃圾回收器回收。SDK的example目录下提供了一个Example.cs，实现了一个简单的单例实现。您需要把Example.cs包含到自己的项目中才能使用。

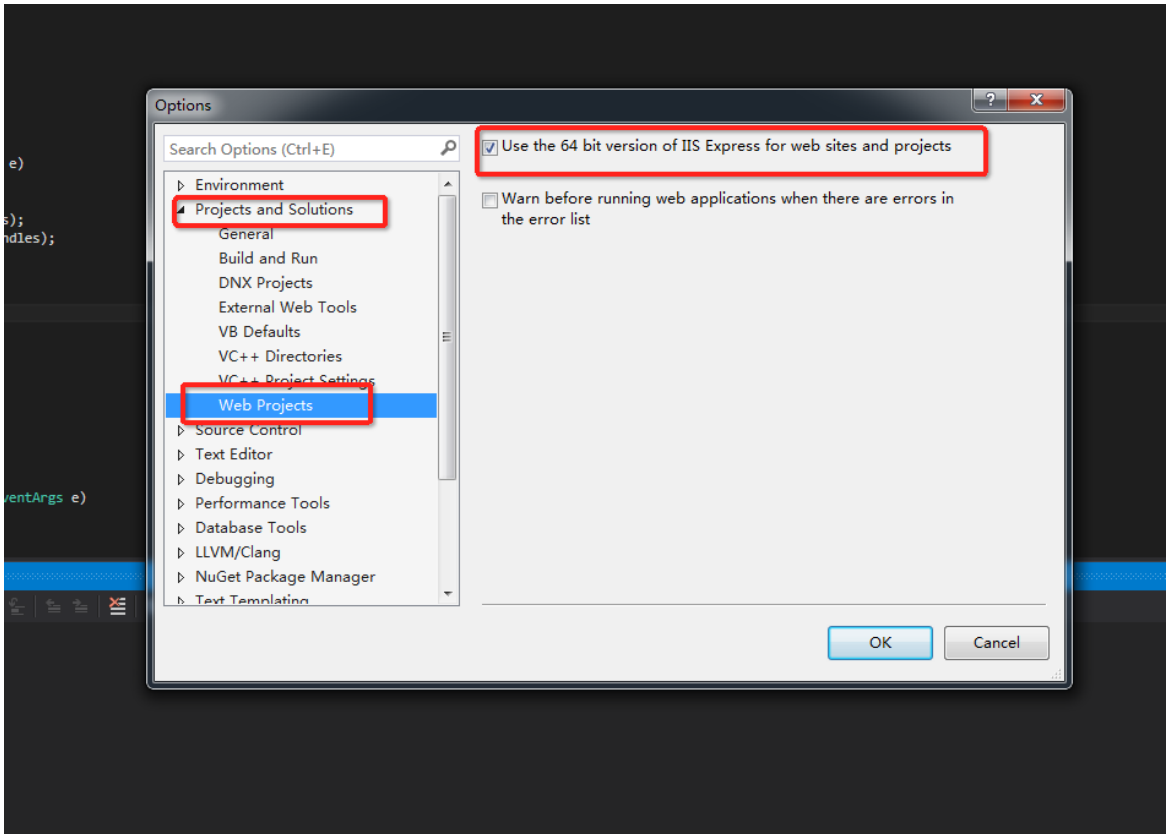
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Optimization;
using System.Web.Routing;
using System.Web.Security;
using System.Web.SessionState;
using ons;    // 这个命名空间是SDK所在的命名空间。
using test;   // 这是一个简单封装SDK的类所在命名空间，建SDK的example目录下的Example.cs。
namespace WebApplication4
{
  public class Global : HttpApplication
  {
       void Application_Start(object sender, EventArgs e)
       {
           // Code that runs on application startup
           RouteConfig.RegisterRoutes(RouteTable.Routes);
           BundleConfig.RegisterBundles(BundleTable.Bundles);
           try
           {
               // 启动SDK的代码，下面是简单封装SDK后的代码。
               OnscSharp.CreateProducer();
               OnscSharp.StartProducer();
           }
           catch (Exception ex)
           {
           // 处理异常。
           }
       }
       protected void Application_End(object sender, EventArgs e)
       {
           try
           {
              // 关闭SDK的代码。
             OnscSharp.ShutdownProducer();
           }
           catch (Exception ex)
           {
           // 处理异常。
           }
       }
  }
}
```

5. 编写测试程序，然后进行编译。
6. 将SDK下的DLL放到和可执行文件同一级目录下，或者系统目录下即可运行。



7. 选择**工具 > 选项 > 项目和解决方案 > Web项目**，然后选中对网站和项目使用**IIS Express**的64位版。





2.3.4. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用TCP协议下的.NET SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 下载.NET SDK。更多信息，请参见版本说明。
- 环境准备。更多信息，请参见环境准备。

发送普通消息

> **说明** 具体的示例代码，请以
> 消息队列RocketMQ版
> 代码库为准。

您可以运行以下代码发送消息。请按说明正确设置相关参数。

```
using System;
using ons;
public class ProducerExampleForEx
{
    public ProducerExampleForEx()
    {
    }
    static void Main(string[] args) {
        // 配置账号，从
消息队列RocketMQ版控制台获取设置。
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
        // AccessKey ID阿里云身份验证，在阿里云服务器管理控制台创建。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
        // AccessKey Secret阿里云身份验证，在阿里云服务器管理控制台创建。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.ProducerId, "GID_example");
        // 您在
消息队列RocketMQ版控制台创建的Topic。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_n
ame");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
        // 设置日志路径。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
        // 创建生产者实例。
        // 说明：生产者实例是线程安全的，可用于发送不同Topic的消息。基本上，您每一个线程只需要一个生产
者实例。
        Producer producer = ONSFactory.getInstance().createProducer(factoryInfo);
        // 启动客户端实例。
        producer.start();
        // 创建消息对象。
        Message msg = new Message(factoryInfo.getPublishTopics(), "tagA", "Example message
body");
        msg.setKey(Guid.NewGuid().ToString());
        for (int i = 0; i < 32; i++) {
            try
            {
                SendResultONS sendResult = producer.send(msg);
                Console.WriteLine("send success {0}", sendResult.getMessageId());
            }
            catch (Exception ex)
            {
                Console.WriteLine("send failure{0}", ex.ToString());
            }
        }
        // 在您的线程即将退出时，关闭生产者实例。
        producer.shutdown();
    }
}
```

订阅普通消息

订阅普通消息的说明和示例代码，请参见订阅消息。

2.3.5. 收发顺序消息

顺序消息（FIFO 消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用TCP协议下的.NET SDK收发顺序消息的示例代码。

顺序消息分类

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息2.0。

前提条件

- 下载.NET SDK。更多信息，请参见版本说明。
- 环境准备。更多信息，请参见环境准备。

发送顺序消息

> 注意
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

具体的示例代码，请以

消息队列RocketMQ版
代码库为准。
发送顺序消息的示例代码如下。

```
using System;
using ons;
public class OrderProducerExampleForEx
{
    public OrderProducerExampleForEx()
    {
    }
    static void Main(string[] args) {
        // 配置您的账号，以下设置均可从控制台获取。
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
        // AccessKey ID阿里云身份验证，在阿里云用户信息管理控制台创建。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
        // AccessKey Secret阿里云身份验证，在阿里云用户信息管理控制台创建。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.ProducerId, "GID_example");
        // 您在
消息队列RocketMQ版控制台创建的Topic。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_n
ame");
        // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
        // 设置日志路径。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
        // 创建生产者实例。
        // 说明：生产者实例是线程安全的，可用于发送不同Topic的消息。基本上，您每一个线程只需要一个生产
者实例。
        OrderProducer producer = ONSFactory.getInstance().createOrderProducer(factoryInfo);
        // 启动客户端实例。
        producer.start();
        // 创建消息对象。
        Message msg = new Message(factoryInfo.getPublishTopics(), "tagA", "Example message
body");
        string shardingKey = "App-Test";
        for (int i = 0; i < 32; i++) {
            try
            {
                SendResultONS sendResult = producer.send(msg, shardingKey);
                Console.WriteLine("send success {0}", sendResult.getMessageId());
            }
            catch (Exception ex)
            {
                Console.WriteLine("send failure{0}", ex.ToString());
            }
        }
        // 在您的线程即将退出时，关闭生产者实例。
        producer.shutdown();
    }
}
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
using System;
using System.Text;
using System.Threading;
using ons;
namespace demo
{
    public class MyMsgOrderListener : MessageOrderListener
    {
        public MyMsgOrderListener()
        {
        }
        ~MyMsgOrderListener()
        {
        }
        public override ons.OrderAction consume(Message value, ConsumeOrderContext context)
        {
            Byte[] text = Encoding.Default.GetBytes(value.getBody());
            Console.WriteLine(Encoding.UTF8.GetString(text));
            return ons.OrderAction.Success;
        }
    }
    class OrderConsumerExampleForEx
    {
        static void Main(string[] args)
        {
            // 配置您的账号，以下设置均可从控制台获取。
            ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
            // AccessKey ID阿里云身份验证，在阿里云用户信息管理控制台创建。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key")
;
            // AccessKey Secret阿里云身份验证，在阿里云用户信息管理控制台创建。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secre
t");
            // 您在
消息队列RocketMQ版控制台创建的Group ID。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.ConsumerId, "GID_example");
            // 您在
消息队列RocketMQ版控制台创建的Topic。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_top
ic_name");
            // 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr")
;
            // 设置日志路径。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
            // 创建消费者实例。
            OrderConsumer consumer = ONSFactory.getInstance().createOrderConsumer(factoryIn
fo);
            // 订阅Topic。
            consumer.subscribe(factoryInfo.getPublishTopics(), "*",new MyMsgOrderListener()
```

```
);
            // 启动消费者实例。
            consumer.start();
            // 让主线程睡眠一段时间。
            Thread.Sleep(30000);
            // 不再使用时，关闭消费者实例。
            consumer.shutdown();
        }
    }
}
```

2.3.6. 收发定时消息

本文提供使用TCP协议下的.NET SDK收发定时消息的示例代码。目前支持的地域包括公网、华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）。
定时消息可以做到在指定时间戳之后才可被消费者消费，适用于对消息生产和消费有时间窗口要求，或者利用消息触发定时任务的场景。

定时消息的概念介绍及使用过程中的注意事项，请参见定时和延时消息。

前提条件

- 下载.NET SDK。更多信息，请参见版本说明。
- 环境准备。更多信息，请参见环境准备。

发送定时消息

> **说明** 具体的示例代码，请以消息队列RocketMQ版代码库为准。

发送定时消息的代码示例如下所示。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;
namespace ons
{
    class onscsharp
    {
        static void Main(string[] args)
        {
            // producer创建和正常工作的参数，必须输入。
            ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
            factoryInfo.setFactoryProperty(ONSFactoryProperty.ProducerId, "XXX ");//您在消息队列RocketMQ版控制台创建的Group ID。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "XXX");//设置TCP接入域名，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "XXX");//您在
```

```
消息队列RocketMQ版控制台创建的Topic。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.MsgContent, "XXX");//消息内容
。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "XXX");//AccessKey
ID阿里云身份验证，在阿里云服务器管理控制台创建。
            factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey,"XXX");//AccessKey
Secret阿里云身份验证，在阿里云服务器管理控制台创建。
            // 创建producer。
     Producer pProducer = ONSFactory.getInstance().createProducer(factoryInfo);
            // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可。
            pProducer.start();
            Message msg = new Message(
                // 消息主题。
                factoryInfo.getPublishTopics(),
                // 消息标签。
                "TagA",
                // 消息主体。
                factoryInfo.getMessageContent()
            );
            // 设置代表消息的业务关键属性，请尽可能全局唯一。
            // 以方便您在无法正常收到消息情况下，可通过
消息队列RocketMQ版控制台查询消息并补发。
            // 注意：不设置也不会影响消息正常收发。
            msg.setKey("ORDERID_100");
            // deliver time，单位：ms。指定一个时刻，在这个时刻之后消息才能被消费，这个例子表示3s后
才能被消费。
            long deliverTime = System.currentTimeMillis() + 3000;
            msg.setStartDeliverTime(deliverTime);
            // 发送消息，只要不抛出异常，就代表发送成功。
            try
            {
                SendResultONS sendResult = pProducer.send(msg);
            }
            catch(ONSClientException e)
            {
                // 发送失败处理。
            }
            // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题。
            pProducer.shutdown();
        }
 }
}
```

订阅定时消息

定时消息的订阅与普通消息订阅一致，更多信息，请参见订阅消息。

2.3.7. 收发事务消息

本文提供使用TCP协议下的.NET SDK收发事务消息的示例代码。目前支持的地域包括公网、华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）。

消息队列RocketMQ版

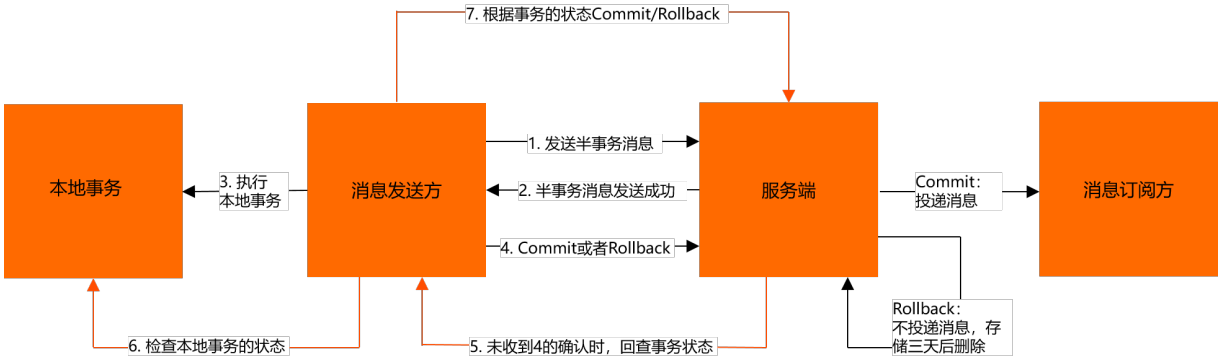
提供类似XA或Open XA的分布式事务功能，通过

消息队列RocketMQ版

事务消息，能达到分布式事务的最终一致。

交互流程

事务消息交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

- 下载.NET SDK。更多信息，请参见版本说明。
- 环境准备。更多信息，请参见环境准备。

发送事务消息

发送事务消息包含以下两个步骤：

1. 发送半事务消息（Half Message）及执行本地事务。示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;
namespace ons
{
public class MyLocalTransactionExecuter : LocalTransactionExecuter
{
    public MyLocalTransactionExecuter()
    {
    }
    ~MyLocalTransactionExecuter()
    {
    }
    public override TransactionStatus execute(Message value)
    {
            Console.WriteLine("execute topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody
:{4}, userProperty:{5}",
            value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.
getBody(), value.getUserProperty("VincentNoUser"));
            // 消息ID(有可能消息体一样，但消息ID不一样。当前消息ID在
消息队列RocketMQ版控制台无法查询)。
            string msgId = value.getMsgID();
            // 消息体内容进行crc32，也可以使用其它的如MD5
```

```
            // 消息体内容进行crc32，也可以使用其它的如MD5。
            // 消息ID和crc32id主要是用来防止消息重复。
            // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或MD5来防止重复消息。
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
            try {
                boolean isCommit = 本地事务执行结果;
                if (isCommit) {
                    // 本地事务成功则提交消息。
                    transactionStatus = TransactionStatus.CommitTransaction;
                } else {
                    // 本地事务失败则回滚消息。
                    transactionStatus = TransactionStatus.RollbackTransaction;
                }
            } catch (Exception e) {
                // 处理异常。
            }
            return transactionStatus;
    }
}
class onscsharp
{
    static void Main(string[] args)
    {
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
        factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "XXX");// 设置TCP接入域名，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "");// 您在
消息队列RocketMQ版控制台创建的Topic。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.MsgContent, "");// 消息内容。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "");// AccessKey
ID阿里云身份验证，在阿里云服务器管理控制台创建。
        factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "");// AccessKey
Secret阿里云身份验证，在阿里云服务器管理控制台创建。
        //create transaction producer
        LocalTransactionChecker myChecker = new MyLocalTransactionChecker();
        TransactionProducer pProducer =ONSFactory.getInstance().createTransactionProdu
cer(factoryInfo,ref myChecker);

        // 在发送消息前，必须调用start方法来启动Producer，只需调用一次即可，启动之后可以多线程
并发发送消息。
        pProducer.start();
            Message msg = new Message(
            //Message Topic
            factoryInfo.getPublishTopics(),
            //Message Tag
            "TagA",
            //Message Body
            factoryInfo.getMessageContent()
        );
        // 设置代表消息的业务关键属性，请尽可能全局唯一。
        // 以方便您在无法正常收到消息情况下，可通
消息队列RocketMQ版过控制台查询消息并补发。
        // 注意：不设置也不会影响消息正常收发。
        msg.setKey("ORDERID_100");
```

```
            // 发送消息，只要不抛出异常，就代表发送成功。
            try
            {
                LocalTransactionExecuter myExecuter = new MyLocalTransactionExecuter();
                SendResultONS sendResult = pProducer.send(msg, ref myExecuter);
            }
            catch(ONSClientException e)
            {
                Console.WriteLine("\nexception of sendmsg:{0}",e.what() );
            }
            // 在应用退出前，必须销毁Producer对象，否则会导致内存泄露等问题。
            // shutdown之后不能重新start此producer。
            pProducer.shutdown();
        }
    }
    }
```

2. 提交事务消息状态。
   当本地事务执行完成（执行成功或执行失败），需要通知服务器当前消息的事务状态。 通知方式有以下两种：
   - 执行本地事务完成后提交。
   - 执行本地事务一直没提交状态，等待服务器回查消息的事务状态。

   事务状态有以下三种：
   - `TransactionStatus.CommitTransaction` 提交事务，允许订阅方消费该消息。
   - `TransactionStatus.RollbackTransaction` 回滚事务，消息将被丢弃不允许消费。
   - `TransactionStatus.Unknow` 无法判断状态，期待Broker向发送方再次询问该消息对应的本地事务的状态。

```
public class MyLocalTransactionChecker : LocalTransactionChecker
{
    public MyLocalTransactionChecker()
    {
    }
    ~MyLocalTransactionChecker()
    {
    }
    public override TransactionStatus check(Message value)
    {
            Console.WriteLine("check topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody:{4
}, userProperty:{5}",
            value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.g
etBody(), value.getUserProperty("VincentNoUser"));
            // 消息ID(有可能消息体一样，但消息ID不一样。当前消息ID在
消息队列RocketMQ版控制台无法查询)。
            string msgId = value.getMsgID();
            // 消息体内容进行crc32，也可以使用其它的如MD5。
            // 消息ID和crc32id主要是用来防止消息重复。
            // 如果业务本身是幂等的，可以忽略，否则需要利用msgId或crc32Id来做幂等。
            // 如果要求消息绝对不重复，推荐做法是对消息体body使用crc32或MD5来防止重复消息。
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
            try {
                boolean isCommit = 本地事务执行结果;
                if (isCommit) {
                    // 本地事务成功、提交消息。
                    transactionStatus = TransactionStatus.CommitTransaction;
                } else {
                    // 本地事务失败、回滚消息。
                    transactionStatus = TransactionStatus.RollbackTransaction;
                }
            } catch (Exception e) {
                //exception handle
            }
            return transactionStatus;
    }
    }
```

事务回查机制说明

- 发送事务消息为什么必须要实现Check机制？
  当步骤1半事务消息发送完成，但本地事务返回状态为 `TransactionStatus.Unknow` 时，亦或是应用退出导致本地事务未提交任何状态时，从Broker的角度看，这条半状态的消息的状态是未知的，因此Broker会定期向消息发送方即消息生产者集群中的任意一生产者实例发起消息回查，要求发送方回查该Half状态消息，并上报其最终状态。
- Check被回调时，业务逻辑都需要做些什么？
  事务消息的Check方法里面，应该写一些检查事务一致性的逻辑。
  消息队列RocketMQ版
  发送事务消息时需要实现 `LocalTransactionChecker` 接口，用来处理Broker主动发起的本地事务状态回查请求；因此在事务消息的Check方法中，需要完成两件事情：
  a. 检查该半事务消息对应的本地事务的状态（committed or rollback）。
  b. 向Broker提交该半事务消息本地事务的状态。

- 本地事务的不同状态对半事务消息的影响：
  - `TransactionStatus.CommitTransaction` 提交事务，允许订阅方消费该消息。
  - `TransactionStatus.RollbackTransaction` 回滚事务，消息将被丢弃不允许消费。
  - `TransactionStatus.Unknow` 无法判断状态，期待Broker向发送方再次询问该消息对应的本地事务的状态。

  具体代码的更多信息，请参见 `MyLocalTransactionChecker` 的实现。

订阅事务消息

事务消息的订阅与普通消息订阅一致，更多信息，请参见订阅消息。

2.3.8. 订阅消息

本文介绍如何通过
消息队列RocketMQ版
的SDK使用.NET语言进行消息订阅。

> **说明** 请确保同一个Group ID下所有Consumer实例的订阅关系保持一致。更多信息，请参见订阅关系一致。

消息队列RocketMQ版

支持以下两种订阅方式：

- 集群订阅
  同一个Group ID所标识的所有Consumer平均分摊消费消息。例如某个Topic有9条消息，一个Group ID有3个Consumer实例，那么在集群消费模式下每个实例平均分摊，只消费其中的3条消息。

```
    // 集群订阅方式设置（不设置的情况下，默认为集群订阅方式）
    factoryInfo.setFactoryProperty(ONSFactoryProperty.MessageModel, ONSFactoryProperty.CL
USTERING);
```

- 广播订阅
  同一个Group ID所标识的所有Consumer都会各自消费某条消息一次。例如某个Topic有9条消息，一个Group ID有3个Consumer实例，那么在广播消费模式下每个实例都会各自消费9条消息。

```
    // 广播订阅方式设置
    factoryInfo.setFactoryProperty(ONSFactoryProperty.MessageModel, ONSFactoryProperty.BR
OADCASTING);
```

示例代码

```
#include "ONSFactory.h"
#include <iostream>
#include <thread>
#include <mutex>
using namespace ons;
std::mutex console_mtx;
class ExampleMessageListener : public MessageListener {
public:
    Action consume(Message& message, ConsumeContext& context) {
        //此处为具体的消息处理过程，确认消息被处理成功请返回CommitMessage。
        //如果有消费异常，或者期望重新消费，可以返回ReconsumeLater，消息将会在一段时间后重新投递。
        std::lock_guard<std::mutex> lk(console_mtx);
```

```
        std::lock_guard<std::mutex> lk(console_mtx);
        std::cout << "Received a message. Topic: " << message.getTopic() << ", MsgId: "
        << message.getMsgID() << std::endl;
        return CommitMessage;
    }
};
int main(int argc, char* argv[]) {
    std::cout << "=======Before consuming messages=======" << std::endl;
    ONSFactoryProperty factoryInfo;
    //请填写在阿里云
消息队列RocketMQ版控制台上申请的Group ID，从实例化的版本开始，ProducerId和CounsumerId已经统一，此处
设置是为了接口保持向前兼容。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "GID_XXX");
    //请填写您的阿里云账号的AccessKey ID。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "Your Access Key");
    //请填写您的阿里云账号的AccessKey Secret。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "Your Secret Key");
    //请填写阿里云
消息队列RocketMQ版控制台上对应实例的接入点。
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR,
                                   "http://xxxxxxxxxxxxxxx.aliyuncs.com:80");
    PushConsumer *consumer = ONSFactory::getInstance()->createPushConsumer(factoryInfo);
    //请填写阿里云
消息队列RocketMQ版控制台上申请的Topic。
    const char* topic_1 = "topic-1";
    // 订阅topic-1中Tag消息属性为tag-1的所有消息。
    const char* tag_1 = "tag-1";
    const char* topic_2 = "topic-2";
    // 订阅topic-2的所有消息。
    const char* tag_2 = "*";
    //请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
    ExampleMessageListener * message_listener = new ExampleMessageListener();
    consumer->subscribe(topic_1, tag_1, message_listener);
    consumer->subscribe(topic_2, tag_2, message_listener);
    //准备工作完成，必须调用启动函数，才可以正常工作。
    consumer->start();
    //请保持线程常驻，不要执行shutdown操作。
    std::this_thread::sleep_for(std::chrono::milliseconds(60 * 1000));
    consumer->shutdown();
    delete message_listener;
    std::cout << "=======After consuming messages======" << std::endl;
    return 0;
}
```

3.商业版HTTP协议SDK（多语言推荐）

3.1. 使用须知

消息队列RocketMQ版
提供了通过HTTP协议的多语言SDK接入的能力，并支持公网访问。本文介绍HTTP协议下的多语言SDK的版本说明。

多语言支持

消息队列RocketMQ版
支持RESTful风格的HTTP协议通信，并提供了7种语言的SDK。HTTP协议的SDK所支持的功能通过不断迭代，与TCP协议的SDK所支持的功能逐渐对齐。HTTP协议的迭代版本所支持的功能与TCP协议功能对比说明，请参见：

- Java SDK版本说明
- Go SDK版本说明
- Python SDK版本说明
- Node.js SDK版本说明
- PHP SDK版本说明
- C# SDK版本说明
- C++ SDK版本说明

注意事项

- 支持HTTP协议SDK接入为
  消息队列RocketMQ版
  商业版的增强功能，开源自建RocketMQ集群不支持该功能，无法使用HTTP协议的SDK接入。
- 确保您需要访问的资源和HTTP接入点在同一地域。
  例如您的
  消息队列RocketMQ版
  实例在华东1（杭州）地域，则只能使用华东1（杭州）地域的接入点来访问该地域的实例。您可以实现以下场景的资源访问：
  - 如果您的客户端在华东1（杭州）地域，为了最佳体验，请使用该实例的HTTP内网接入点访问该实例资源。
  - 如果您的客户端在华东1（杭州）以外的任一地域，请确保客户端可连上互联网，并使用该实例的HTTP公网接入点访问该实例资源。
- TCP协议的客户端和HTTP协议的客户端之间可以实现消息收发。但由于HTTP协议采用XML序列化，因此消息的属性、内容、Tag、Key等必须符合XML规范。

  > **说明** XML的规范详情，请参见XML语法。您也可按需使用第三方工具xml_validator校验XML语法的规范性。

  如果包含了不符合XML规范的相关字符，那么可能出现以下情况：
  - 采用HTTP协议发送消息时，发送消息失败。
  - 采用TCP协议发送消息，HTTP协议消费消息时，消费消息失败。

您可以自行采用Base64编码对发送的消息进行编（解）码，以适用于此类不符合XML规范的消息收发场景。

3.2. 协议说明

3.2.1. 公共参数

本文介绍
消息队列RocketMQ版
的HTTP请求Header中的公共请求参数和公共返回参数。

公共请求头

参数	是否必选	说明
Authorization	是	验证字符串，由 `MQ+`**空格**`+<AccessKeyId>+:+<Signature>` 构成，更多信息，请参见签名机制。
Content-Length	是	HTTP请求Body的长度。
Content-Type	是	请求内容的MIME类型，目前请求仅支持text/xml; charset=utf-8，即MIME类型为XML，编码字符集为UTF-8。
Date	是	请求的构造时间，目前只支持GMT格式，如果和 消息队列RocketMQ版 的服务器时间前后差异超过15分钟将返回本次请求非法。
Host	是	即在控制台**实例详情**页面查看到的HTTP接入点。
x-mq-version	是	调用 消息队列RocketMQ版 的版本号，当前版本为2015-06-06。
Content-MD5	否	HTTP消息体的MD5值。具体格式，请参见Content-MD5 Header Field。

公共返回头

参数	说明
Content-Length	HTTP响应Body的长度。
Connection	HTTP连接状态。

参数	说明
Date	响应的返回时间，GMT时间格式。
x-mq-request-id	此次请求的请求序列号。
x-mq-version	API的版本编号，当前版本是2015-06-06。

3.2.2. 签名机制

消息队列RocketMQ版
会验证每个访问的HTTP请求。每个向
消息队列RocketMQ版
提交的HTTP请求中都包含Authorization，Authorization又包含了签名（Signature）。本文介绍签名的生成机制。

背景信息

AccessKey ID和AccessKey Secret由阿里云官方颁发给访问者（可以通过阿里云管理控制台申请和管理），其中：

- AccessKey ID用于标识访问者的身份。
- AccessKey Secret用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密。

消息队列RocketMQ版
提供的HTTP服务通过使用AccessKey ID和AccessKey Secret进行对称加密的方法来验证请求的发送者身份。如果计算出来的验证码和提供的一样，即认为该请求是有效的；否则，将拒绝处理这次请求，并返回HTTP 403错误。
您必须在HTTP请求中增加Authorization的Header来包含签名信息，表明这个消息已被授权。

获取方式

Authorization信息的格式为：

```
MQ <AccessKey ID>:<Signature>
```

Signature的生成方式如下所示。

```
Signature = base64(hmac-sha1(HTTP_METHOD + "\n"
                + "\n"+ CONTENT-TYPE + "\n"
                + DATE + "\n"
                + "x-mq-version:" + MQVersion + "\n"
                + CanonicalizedResource))
```

- HTTP_METHOD：表示大写的HTTP Method（如PUT、GET、POST、DELETE）。
- CONTENT-TYPE：表示请求内容的类型，即为text/xml; charset=utf-8。
- DATE：表示此次操作的时间，不能为空，目前只支持GMT格式，示例：Thu, 07 Mar 2012 18:49:58 GMT。
- MQVersion：表示
  消息队列RocketMQ版
  接口的版本号，当前版本即为2015-06-06。
- CanonicalizedResource：表示HTTP所请求资源的URI（统一资源标识符），如消费请求的URI /topics/abc/messages?consumer=GID_abc。

> 说明
> - 用来签名的字符串为UTF-8格式。
> - 签名的方法用RFC 2104中定义的HMAC-SHA1方法，其中Key为AccessKey Secret。

3.2.3. 发送消息API

使用发送消息API从生产者发送消息至
消息队列RocketMQ版
服务端。

请求构造

- 请求行
  ```
  POST /topics/TopicName/messages?ns=INSTANCE_ID HTTP/1.1
  ```
  参数说明如下。

参数	是否必选	说明
TopicName	是	将消息发送至的Topic名称。
ns	否	实例ID，针对有命名空间的新建实例，该参数为必填。是否有命名空间可以在控制台**实例详情**页面查看。实例根据其是否有命名空间分为默认实例和新建实例： ○ 默认实例：无命名空间，所有资源命名必需保证全局唯一。 ○ 新建实例：有命名空间，资源命名只需保证在该实例内唯一。 实例命名空间的更多信息，请参见实例化支持。

- 请求内容（XML格式）
  请求内容的参数说明如下。

参数	是否必选	说明
MessageTag	否	消息Tag。
MessageBody	是	消息内容。
Properties	否	消息属性。

  属性（Properties）的序列化，其中特殊用途的键值说明如下：
  - 格式：`key1:value1|key2:value2|key3:value3`

- 键值说明如下。

参数	类型	说明
KEYS	String	消息的Key。
__STARTDELIVERTIME	Long	定时消息的定时绝对时间，UNIX毫秒时间戳。
__TransCheckT	Long	第一次事务消息的回查时间，相对时间，取值范围：10～300，单位：秒。

响应构造

- 响应行
  ```
  HTTP/1.1 201
  ```
- 响应内容
  响应内容的参数说明如下。

参数	类型	说明
MessageId	String	消息ID。
MessageBodyMD5	String	消息内容的MD5。

示例

- 请求示例

```
<?xml version="1.0" encoding="UTF-8"?>
<Message xmlns="http://mq.aliyuncs.com/doc/v1/">
    <MessageBody>a</MessageBody>
<MessageTag>Tag</MessageTag>
<Properties>KEYS:MessageKey|__STARTDELIVERTIME:1571388173000</Properties>
</Message>
```

- 响应示例

```
<Message xmlns="http://mq.aliyuncs.com/doc/v1/">
<MessageId>1E057D566EAD42A579935B5CD874****</MessageId>
<MessageBodyMD5>0CC175B9C0F1B6A831C399E26977****</MessageBodyMD5>
</Message>
```

3.2.4. 订阅消息API

订阅者可通过调用订阅消息API来消费
消息队列RocketMQ版
服务端中的消息。

请求构造

- 请求行

```
GET /topics/TopicName/messages?ns=INSTANCE_ID&consumer=GID&tag=taga&numOfMessages=3&waitseconds=3 HTTP/1.1
```

参数说明如下。

参数	是否必选	说明
TopicName	是	将消息发送至的Topic名称。
ns	否	实例ID，针对有命名空间的新建实例，该参数为必填。 是否有命名空间可以在控制台**实例详情**页面查看。实例根据其是否有命名空间分为默认实例和新建实例： ○ 默认实例：无命名空间，所有资源命名在该默认实例内需全局唯一。 ○ 新建实例：有命名空间，资源命名只需保证在该实例内唯一。 详细信息，请参见实例化支持。
consumer	是	消费者群组的标识，即Group ID。
tag	否	消息Tag。如果不指定Tag，则拉取所有消息。如果需要指定多个Tag，则使用双竖线（||）隔开，例如TagA||TagB。
numOfMessages	是	一次最多消费多少条消息，取值范围：1～16。
waitseconds	否	长轮询时间，不填则为短轮询，取值范围：1～30，单位：秒。

- 请求内容（XML格式）
  无

响应构造

- 有消息可消费
  - 响应行
    ```
    HTTP/1.1 200
    ```
  - 响应内容
    响应内容的参数说明如下。

参数	类型	说明
MessageId	String	消息ID。
MessageBodyMD5	String	消息内容的MD5。
MessageBody	String	消息内容。

参数	类型	说明
ReceiptHandle	String	消息的句柄，用于确认消息消费成功，句柄仅使用一次，对于同一条消息如果存在重试每次拿到的句柄是不同的，消息句柄应该在到NextConsumeTime指定的时间之前使用。
PublishTime	String	消息的发送时间戳，单位：毫秒。
FirstConsumeTime	String	消息第一次消费的时间戳，单位：毫秒。
NextConsumeTime	String	该消息再次被消费到（即重试）的绝对时间戳，单位：毫秒。 **说明** HTTP协议下消息队列RocketMQ版的消息重试机制为：无序消息每隔5分钟重试一次，顺序消息每隔1分钟重试一次，最多重试288次。
ConsumedTimes	String	消息被消费的次数。
MessageTag	String	消息Tag。
Properties	String	消息属性。

属性（Properties）的序列化，其中特殊用途的键值说明如下：

- 格式：`key1:value1|key2:value2|key3:value3`
- 键值说明如下：

参数	类型	说明
KEYS	String	消息的Key。
__STARTDELIVERTIME	Long	表示定时消息的定时绝对时间，UNIX毫秒时间戳。
__TransCheckT	Long	表示第一次事务消息的回查时间，相对时间，单位：秒，取值范围：10～300。

- 无消息可消费
  - 响应行

    ```
    HTTP/1.1 404
    ```

- 响应内容
  响应内容的参数说明如下。

参数	类型	说明
Code	String	错误代码，其中 `MessageNotExist` 表示没有消息可以消费，是正常的响应。
Message	String	错误内容。
RequestId	String	请求ID。
HostId	String	请求Host。

响应示例

- 有消息可消费

```
<?xml version="1.0" ?>
<Messages xmlns="http://mq.aliyuncs.com/doc/v1">
<Message>
  <MessageId>1E057D5E6EAD42A579937046FE17****</MessageId>
  <MessageBodyMD5>0CC175B9C0F1B6A831C399E26977****</MessageBodyMD5>
  <MessageBody>a</MessageBody>
  <ReceiptHandle>1E057D5E6EAD42A579937046FE17****-MTI5N****</ReceiptHandle>
  <PublishTime>1571742900759</PublishTime>
  <FirstConsumeTime>1571742902463</FirstConsumeTime>
  <NextConsumeTime>1571742922463</NextConsumeTime>
  <ConsumedTimes>1</ConsumedTimes>
  <MessageTag>Tag</MessageTag>
  <Properties>KEYS:MessageKey|__BORNHOST:30.5.**.**|</Properties>
</Message>
<Message>
  <MessageId>1E057D5E6EAD42A579937046FE17****</MessageId>
  <MessageBodyMD5>0CC175B9C0F1B6A831C399E26977****</MessageBodyMD5>
  <MessageBody>a</MessageBody>
  <ReceiptHandle>1E057D5E6EAD42A579937046FE17****-MTI5N****</ReceiptHandle>
  <PublishTime>1571742900759</PublishTime>
  <FirstConsumeTime>1571742902463</FirstConsumeTime>
  <NextConsumeTime>1571742922463</NextConsumeTime>
  <ConsumedTimes>1</ConsumedTimes>
  <MessageTag>Tag</MessageTag>
  <Properties>KEYS:MessageKey|__BORNHOST:30.5.**.**|</Properties>
</Message>
</Messages>
```

- 无消息可消费

```
<?xml version="1.0" ?>
<Error xmlns="http://mq.aliyuncs.com/doc/v1">
  <Code>MessageNotExist</Code>
  <Message>Message not exist.</Message>
  <RequestId>5DAEE3FF463541AD6E0322EB</RequestId>
  <HostId>http://123.mqrest.cn-hangzhou.aliyuncs.com</HostId>
</Error>
```

3.2.5. 确认消息API

使用确认消息API确认消息消费状态。

请求构造

- 请求行
  `DELETE /topics/TopicName/messages?ns=INSTANCE_ID&consumer=GID HTTP/1.1`
  参数说明如下。

参数	是否必选	说明
TopicName	是	将消息发送至的Topic名称。
ns	否	实例ID，针对有命名空间的新建实例，该参数为必填。是否有命名空间可以在控制台**实例详情**页面查看。实例根据其是否有命名空间分为默认实例和新建实例： ○ 默认实例：无命名空间，所有资源命名必需保证全局唯一。 ○ 新建实例：有命名空间，资源命名只需保证在该实例内唯一。 实例命名空间更多信息，请参见实例化支持。
consumer	是	消费者群组标识，即Group ID。

- 请求内容（XML格式）
  请求内容的参数说明如下。

参数	是否必选	说明
ReceiptHandle	是	通过订阅消息API获取到的消息句柄，用于确认消息是否消费成功；句柄用且仅能使用一次，对于同一条消息如果存在重试每次拿到的句柄不同，消息句柄应在NextConsumeTime之前使用。

响应构造

- 请求成功
  - 响应行
    `HTTP/1.1 204`

- 响应内容
  无

- 请求失败
  - 响应行
    `HTTP/1.1 404`
  - 响应内容
    参见响应示例

示例

- 请求示例

```
<?xml version="1.0" encoding="UTF-8"?>
<ReceiptHandles xmlns="http://mq.aliyuncs.com/doc/v1/">
<ReceiptHandle>1E057D5E6EAD42A57993704EC383****-MTI5NT****</ReceiptHandle>
<ReceiptHandle>1E057D5E6EAD42A57993704EC383****-MTI5NT****</ReceiptHandle>
<ReceiptHandle>1E057D5E6EAD42A57993704EC383****-MTI5NT****</ReceiptHandle>
</ReceiptHandles>
```

- 响应示例
  - 请求的内容中没有句柄。

```
    <?xml version="1.0" ?>
    <Error xmlns="http://mq.aliyuncs.com/doc/v1">
<Code>MissingReceiptHandle</Code>
<Message>ReceiptHandle is required.</Message>
<RequestId>5DAEF2B9463541AD6E04490F</RequestId>
<HostId>http://123.mqrest.cn-hangzhou.aliyuncs.com</HostId>
    </Error>
```

  - 请求的句柄错误，错误的句柄是：`adfadfadf`。

```
    <?xml version="1.0" ?>
    <Errors xmlns="http://mq.aliyuncs.com/doc/v1">
<Error>
        <ErrorCode>ReceiptHandleError</ErrorCode>
        <ErrorMessage>The receipt handle you provide is not valid.</ErrorMessage>
        <ReceiptHandle>adfadfadf</ReceiptHandle>
</Error>
    </Errors>
```

  - 请求的句柄过期，即使用句柄确认消息的时候超过了消息的NextConsumeTime时间。

```
    <?xml version="1.0" ?>
    <Errors xmlns="http://mq.aliyuncs.com/doc/v1">
<Error>
        <ErrorCode>MessageNotExist</ErrorCode>
        <ErrorMessage>The receipt handle you provided has expired.</ErrorMessage>
        <ReceiptHandle>1E057D5E6EAD42A57993704EC383****-MTI5NT****</ReceiptHandle>
</Error>
    </Errors>
```

3.3. Java SDK

3.3.1. 版本说明

本文介绍HTTP协议下Java SDK的版本信息，包括各版本的发布时间、发布内容以及下载链接，以便您按需获取适用的Java SDK收发消息。

> 注意　获取Maven依赖，请参见准备环境。

1.0.3.2

发布时间	发布内容	下载
2021-12-24	**功能优化** • 去除Log4j依赖，改为使用SLF4J。	mq-http-java-sdk-1.0.3.2.zip

1.0.3.1

发布时间	发布内容	下载
2021-12-13	**功能优化** • Log4j版本升级至2.15.0。	mq-http-java-sdk-1.0.3.1.zip

1.0.3

发布时间	发布内容	下载
2021-01-05	**新特性** • 支持顺序消息。	mq-http-java-sdk-1.0.3.zip

1.0.2

发布时间	发布内容	下载
2020-01-15	**功能优化** • 更新Log4j至Log4j 2。	mq-http-sdk-1.0.2.zip

1.0.1

发布时间	发布内容	下载

发布时间	发布内容	下载
2019-06-10	**新特性** • 支持高级特性消息，包括定时和延时消息、事务消息，暂不支持顺序消息。 • 支持使用Message ID查询消息轨迹。 • 支持消息属性。	mq-http-java-sdk-1.0.1.zip

1.0.0

发布时间	发布内容	下载
2019-01-23	**新特性** • 首次发布。 • 暂不支持高级特性消息，包括顺序消息、事务消息、定时和延时消息。 • 已支持集群消费模式，暂不支持广播消费模式。 • 暂不支持查询消息轨迹。	mq-http-java-sdk-1.0.0.zip

3.3.2. 准备环境

在使用Java SDK收发消息前，您需按照本文提供的内容来准备环境。

环境要求

- 安装JDK 1.6或以上版本。更多信息，请参见安装JDK。
- 安装Maven。更多信息，请参见安装Maven。

安装Java SDK

通过Maven方式引入依赖，在*pom.xml*中添加以下依赖。

```
<dependency>
    <groupId>com.aliyun.mq</groupId>
    <artifactId>mq-http-sdk</artifactId>
    <!--以下版本号请替换为Java SDK的最新版本号-->
    <version>1.0.3.2</version>
    <classifier>jar-with-dependencies</classifier>
</dependency>
```

> **说明** 获取SDK的版本信息，请参见版本说明。

3.3.3. 收发普通消息

普通消息是指
消息队列Rocket MQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用HTTP协议下的Java SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 安装Java SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送普通消息

发送普通消息的示例代码如下。

```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQProducer;
import com.aliyun.mq.http.model.TopicMessage;
import java.util.Date;
public class Producer {
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        // 不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型
的消息。
        final String topic = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实
例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
        // 获取Topic的生产者。
        MQProducer producer;
        if (instanceId != null && instanceId != "") {
            producer = mqClient.getProducer(instanceId, topic);
        } else {
            producer = mqClient.getProducer(topic);
        }
        try {
            // 循环发送4条消息。
            for (int i = 0; i < 4; i++) {
                TopicMessage pubMsg;        // 普通消息。
                pubMsg = new TopicMessage(
                // 消息内容。
                "hello mq!".getBytes(),
                // 消息标签。
                "A"
                );
```

```
                // 设置消息的自定义属性。
                pubMsg.getProperties().put("a", String.valueOf(i));
                // 设置消息的Key。
                pubMsg.setMessageKey("MessageKey");
            // 同步发送消息，只要不抛异常就是成功。
            TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
            // 同步发送消息，只要不抛异常就是成功。
            System.out.println(new Date() + " Send mq message success. Topic is:" + topic +
", msgId is: " + pubResultMsg.getMessageId()
                        + ", bodyMD5 is: " + pubResultMsg.getMessageBodyMD5());
            }
        } catch (Throwable e) {
            // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
            System.out.println(new Date() + " Send mq message failed. Topic is:" + topic);
            e.printStackTrace();
        }
        mqClient.close();
    }
}
```

订阅普通消息

订阅普通消息的示例代码如下。

```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQConsumer;
import com.aliyun.mq.http.common.AckMessageException;
import com.aliyun.mq.http.model.Message;
import java.util.ArrayList;
import java.util.List;
public class Consumer {
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型
的消息。
        final String topic = "${TOPIC}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        final String groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实
例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
```

```
        final String instanceId = ${INSTANCE_ID};
        final MQConsumer consumer;
        if (instanceId != null && instanceId != "") {
            consumer = mqClient.getConsumer(instanceId, topic, groupId, null);
        } else {
            consumer = mqClient.getConsumer(topic, groupId);
        }
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
        do {
            List<Message> messages = null;
            try {
                // 长轮询消费消息。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
                messages = consumer.consumeMessage(
                        3,// 一次最多消费3条消息（最多可设置为16条）。
                        3// 长轮询时间3秒（最多可设置为30秒）。
                );
            } catch (Throwable e) {
                e.printStackTrace();
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }
            }
            // Topic中没有消息可消费。
            if (messages == null || messages.isEmpty()) {
                System.out.println(Thread.currentThread().getName() + ": no new message, continue!");
                continue;
            }
            // 处理业务逻辑。
            for (Message message : messages) {
                System.out.println("Receive message: " + message);
            }
            // 消息重试时间到达前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费的时间戳都不一样。
            {
                List<String> handles = new ArrayList<String>();
                for (Message message : messages) {
                    handles.add(message.getReceiptHandle());
                }
                try {
                    consumer.ackMessage(handles);
                } catch (Throwable e) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    if (e instanceof AckMessageException) {
                        AckMessageException errors = (AckMessageException) e;
                        System.out.println("Ack message fail, requestId is:" + errors.getRequestId() + ", fail handles:");
                        if (errors.getErrorMessages() != null) {
                            for (String errorHandle :errors.getErrorMessages().keySet()) {
                                System.out.println("Handle:" + errorHandle + ", ErrorCode:" + errors.getErrorMessages().get(errorHandle).getErrorCode()
```

```
                                        + ", ErrorMsg:" + errors.getErrorMessages().get(err
orHandle).getErrorMessage());
                                }
                            }
                            continue;
                        }
                        e.printStackTrace();
                    }
                }
            } while (true);
        }
    }
```

3.3.4. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用HTTP协议下的Java SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息。

前提条件

您已完成以下操作：

- 安装Java SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送顺序消息

> 注意
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

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

```
import java.util.Date;
public class OrderProducer {
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
        // 不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型的消息。
        final String topic = "${TOPIC}";
        // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
        // 获取Topic的生产者。
        MQProducer producer;
        if (instanceId != null && instanceId != "") {
            producer = mqClient.getProducer(instanceId, topic);
        } else {
            producer = mqClient.getProducer(topic);
        }
        try {
            // 循环发送8条消息。
            for (int i = 0; i < 8; i++) {
                TopicMessage pubMsg = new TopicMessage(
                        // 消息内容。
                        "hello mq!".getBytes(),
                        // 消息标签。
                        "A"
                );
                // 设置分区顺序消息的Sharding Key，用于标识不同的分区。Sharding Key与消息的Key是完全不同的概念。
                pubMsg.setShardingKey(String.valueOf(i % 2));
                pubMsg.getProperties().put("a", String.valueOf(i));
                // 同步发送消息，只要不抛异常就是成功。
                TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
                // 同步发送消息，只要不抛异常就是成功。
                System.out.println(new Date() + " Send mq message success. Topic is:" + topic + ", msgId is: " + pubResultMsg.getMessageId()
                        + ", bodyMD5 is: " + pubResultMsg.getMessageBodyMD5());
            }
        } catch (Throwable e) {
            // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
            System.out.println(new Date() + " Send mq message failed. Topic is:" + topic);
            e.printStackTrace();
        }
```

```
        }
        mqClient.close();
    }
}
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQConsumer;
import com.aliyun.mq.http.common.AckMessageException;
import com.aliyun.mq.http.model.Message;
import java.util.ArrayList;
import java.util.List;
public class OrderConsumer {
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型
的消息。
        final String topic = "${TOPIC}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        final String groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实
例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
        final MQConsumer consumer;
        if (instanceId != null && instanceId != "") {
            consumer = mqClient.getConsumer(instanceId, topic, groupId, null);
        } else {
            consumer = mqClient.getConsumer(topic, groupId);
        }
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
        do {
            List<Message> messages = null;
            try {
                // 长轮询顺序消费消息，拉取到的消息可能是多个分区的（对于分区顺序消息），一个分区内的
消息一定是顺序的。
                // 对于分区顺序消息，只要一个分区内存在没有被确认消费的消息，那么该分区下次还会消费到
相同的消息。
                // 对于一个分区，只有所有消息确认消费成功才能消费下一批消息。
```

```
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则服务端立即返回响应。
                messages = consumer.consumeMessageOrderly(
                        3,  // 一次最多消费3条消息（最多可设置为16条）。
                        3   // 长轮询时间为3秒（最多可设置为30秒）。
                );
            } catch (Throwable e) {
                e.printStackTrace();
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }
            }
            // Topic中没有消息可消费。
            if (messages == null || messages.isEmpty()) {
                System.out.println(Thread.currentThread().getName() + ": no new message, continue!");
                continue;
            }
            // 处理业务逻辑。
            System.out.println("Receive " + messages.size() + " messages:");
            for (Message message : messages) {
                System.out.println(message);
                System.out.println("ShardingKey: " + message.getShardingKey() + ", a:" + message.getProperties().get("a"));
            }
            // 消息重试时间到达前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            {
                List<String> handles = new ArrayList<String>();
                for (Message message : messages) {
                    handles.add(message.getReceiptHandle());
                }
                try {
                    consumer.ackMessage(handles);
                } catch (Throwable e) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    if (e instanceof AckMessageException) {
                        AckMessageException errors = (AckMessageException) e;
                        System.out.println("Ack message fail, requestId is:" + errors.getRequestId() + ", fail handles:");
                        if (errors.getErrorMessages() != null) {
                            for (String errorHandle :errors.getErrorMessages().keySet()) {
                                System.out.println("Handle:" + errorHandle + ", ErrorCode:" + errors.getErrorMessages().get(errorHandle).getErrorCode()
                                        + ", ErrorMsg:" + errors.getErrorMessages().get(errorHandle).getErrorMessage());
                            }
                        }
                        continue;
                    }
                    e.printStackTrace();
                }
```

```
            }
        } while (true);
    }
}
```

3.3.5. 收发定时消息和延时消息

本文提供使用HTTP协议下的Java SDK收发定时消息和延时消息的示例代码。

背景信息

- 延时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。
- 定时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

在HTTP协议下，定时消息和延时消息的代码配置相同，本质都是根据消息中的属性延迟固定时间后才投递给消费者。
更多信息，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 安装Java SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQProducer;
import com.aliyun.mq.http.model.TopicMessage;
import java.util.Date;
public class Producer {
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
        //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型的消息。
        final String topic = "${TOPIC}";
```

```
        final String topic = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
        // 获取Topic的生产者。
        MQProducer producer;
        if (instanceId != null && instanceId != "") {
            producer = mqClient.getProducer(instanceId, topic);
        } else {
            producer = mqClient.getProducer(topic);
        }
        try {
            // 循环发送4条消息。
            for (int i = 0; i < 4; i++) {
                TopicMessage pubMsg;
                pubMsg = new TopicMessage(
                // 消息内容。
                "hello mq!".getBytes(),
                // 消息标签。
                "A"
                 );
                 // 设置消息的自定义属性。
                 pubMsg.getProperties().put("a", String.valueOf(i));
                 // 设置消息的Key。
                 pubMsg.setMessageKey("MessageKey");
                 // 延时消息，发送时间为10s后。该参数格式为毫秒级别的时间戳。
                 // 若发送定时消息，设置该参数时需要计算定时时间与当前时间的时间差。
                 pubMsg.setStartDeliverTime(System.currentTimeMillis() + 10 * 1000);

                // 同步发送消息，只要不抛异常就是成功。
                TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
                // 同步发送消息，只要不抛异常就是成功。
                System.out.println(new Date() + " Send mq message success. Topic is:" + topic
+ ", msgId is: " + pubResultMsg.getMessageId()
                        + ", bodyMD5 is: " + pubResultMsg.getMessageBodyMD5());
            }
        } catch (Throwable e) {
            // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
            System.out.println(new Date() + " Send mq message failed. Topic is:" + topic);
            e.printStackTrace();
        }
        mqClient.close();
    }
}
```

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

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

```
import com.aliyun.mq.http.model.Message;
import java.util.ArrayList;
import java.util.List;
public class Consumer {
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型
的消息。
        final String topic = "${TOPIC}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        final String groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实
例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
        final MQConsumer consumer;
        if (instanceId != null && instanceId != "") {
            consumer = mqClient.getConsumer(instanceId, topic, groupId, null);
        } else {
            consumer = mqClient.getConsumer(topic, groupId);
        }
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
        do {
            List<Message> messages = null;
            try {
                // 长轮询消费消息。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即
返回客户端。
                messages = consumer.consumeMessage(
                        3,// 一次最多消费3条消息（最多可设置为16条）。
                        3// 长轮询时间3秒（最多可设置为30秒）。
                );
            } catch (Throwable e) {
                e.printStackTrace();
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }
            }
            // Topic中没有消息可消费。
```

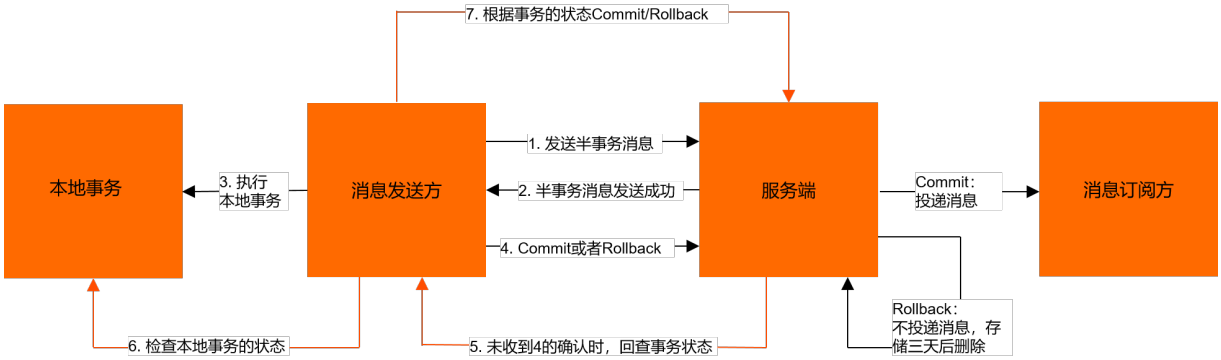
```
            if (messages == null || messages.isEmpty()) {
                System.out.println(Thread.currentThread().getName() + ": no new message, co
ntinue!");
                continue;
            }
            // 处理业务逻辑。
            for (Message message : messages) {
                System.out.println("Receive message: " + message);
            }
            // 消息重试时间到达前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费的时间戳都不一样。
            {
                List<String> handles = new ArrayList<String>();
                for (Message message : messages) {
                    handles.add(message.getReceiptHandle());
                }
                try {
                    consumer.ackMessage(handles);
                } catch (Throwable e) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    if (e instanceof AckMessageException) {
                        AckMessageException errors = (AckMessageException) e;
                        System.out.println("Ack message fail, requestId is:" + errors.getRe
questId() + ", fail handles:");
                        if (errors.getErrorMessages() != null) {
                            for (String errorHandle :errors.getErrorMessages().keySet()) {
                                System.out.println("Handle:" + errorHandle + ", ErrorCode:"
+ errors.getErrorMessages().get(errorHandle).getErrorCode()
                                        + ", ErrorMsg:" + errors.getErrorMessages().get(err
orHandle).getErrorMessage());
                            }
                        }
                        continue;
                    }
                    e.printStackTrace();
                }
            }
        } while (true);
    }
}
```

3.3.6. 收发事务消息

消息队列RocketMQ版
提供类似XA或Open XA的分布式事务功能，通过
消息队列RocketMQ版
事务消息，能达到分布式事务的最终一致。本文提供使用HTTP协议下的Java SDK收发事务消息的示例代码。

背景信息

事务消息的交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 安装Java SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送事务消息

发送事务消息的示例代码如下。

```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQTransProducer;
import com.aliyun.mq.http.common.AckMessageException;
import com.aliyun.mq.http.model.Message;
import com.aliyun.mq.http.model.TopicMessage;
import java.util.List;
public class TransProducer {
    static void processCommitRollError(Throwable e) {
        if (e instanceof AckMessageException) {
            AckMessageException errors = (AckMessageException) e;
            System.out.println("Commit/Roll transaction error, requestId is:" + errors.getR
equestId() + ", fail handles:");
            if (errors.getErrorMessages() != null) {
                for (String errorHandle :errors.getErrorMessages().keySet()) {
                    System.out.println("Handle:" + errorHandle + ", ErrorCode:" + errors.ge
tErrorMessages().get(errorHandle).getErrorCode()
                            + ", ErrorMsg:" + errors.getErrorMessages().get(errorHandle).ge
tErrorMessage());
                }
            }
        }
    }
    public static void main(String[] args) throws Throwable {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
```

```
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        // 不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型
的消息。
        final String topic = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实
例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        final String groupId = "${GROUP_ID}";
        final MQTransProducer mqTransProducer = mqClient.getTransProducer(instanceId, topic
, groupId);
        for (int i = 0; i < 4; i++) {
            TopicMessage topicMessage = new TopicMessage();
            topicMessage.setMessageBody("trans_msg");
            topicMessage.setMessageTag("a");
            topicMessage.setMessageKey(String.valueOf(System.currentTimeMillis()));
            // 设置事务第一次回查的时间，为相对时间。单位：秒，取值范围：10~300。
            // 第一次事务回查后如果消息没有提交或者回滚，则之后每隔10s左右会回查一次，共回查24小时。
            topicMessage.setTransCheckImmunityTime(10);
            topicMessage.getProperties().put("a", String.valueOf(i));
            TopicMessage pubResultMsg = null;
            pubResultMsg = mqTransProducer.publishMessage(topicMessage);
            System.out.println("Send---->msgId is: " + pubResultMsg.getMessageId()
                    + ", bodyMD5 is: " + pubResultMsg.getMessageBodyMD5()
                    + ", Handle: " + pubResultMsg.getReceiptHandle()
            );
            if (pubResultMsg != null && pubResultMsg.getReceiptHandle() != null) {
                if (i == 0) {
                    // 发送完事务消息后能获取到半消息句柄，可以直接提交或回滚事务消息。
                    try {
                        mqTransProducer.commit(pubResultMsg.getReceiptHandle());
                        System.out.println(String.format("MessageId:%s, commit", pubResultM
sg.getMessageId()));
                    } catch (Throwable e) {
                        // 如果提交或回滚事务消息时超过了TransCheckImmunityTime（针对发送事务消息
的句柄）设置的时长，则会失败。
                        if (e instanceof AckMessageException) {
                            processCommitRollError(e);
                            continue;
                        }
                    }
                }
            }
        }
        // 客户端需要有一个线程或者进程来消费没有确认的事务消息。
        // 启动一个线程来检查没有确认的事务消息。
        Thread t = new Thread(new Runnable() {
```

```
        Thread t = new Thread(new Runnable() {
            public void run() {
                int count = 0;
                while(true) {
                    try {
                        if (count == 3) {
                            break;
                        }
                        List<Message> messages = mqTransProducer.consumeHalfMessage(3, 3);
                        if (messages == null) {
                            System.out.println("No Half message!");
                            continue;
                        }
                        System.out.println(String.format("Half---->MessageId:%s,Properties:
%s,Body:%s,Latency:%d",
                                messages.get(0).getMessageId(),
                                messages.get(0).getProperties(),
                                messages.get(0).getMessageBodyString(),
                                System.currentTimeMillis() - messages.get(0).getPublishTime
()));
                        for (Message message : messages) {
                            try {
                                if (Integer.valueOf(message.getProperties().get("a")) == 1)
{
                                    // 确认提交事务消息。
                                    mqTransProducer.commit(message.getReceiptHandle());
                                    count++;
                                    System.out.println(String.format("MessageId:%s, commit"
, message.getMessageId()));
                                } else if (Integer.valueOf(message.getProperties().get("a")
) == 2
                                        && message.getConsumedTimes() > 1) {
                                    // 确认提交事务消息。
                                    mqTransProducer.commit(message.getReceiptHandle());
                                    count++;
                                    System.out.println(String.format("MessageId:%s, commit"
, message.getMessageId()));
                                } else if (Integer.valueOf(message.getProperties().get("a")
) == 3) {
                                    // 确认回滚事务消息。
                                    mqTransProducer.rollback(message.getReceiptHandle());
                                    count++;
                                    System.out.println(String.format("MessageId:%s, rollbac
k", message.getMessageId()));
                                } else {
                                    // 什么都不做，下次再检查。
                                    System.out.println(String.format("MessageId:%s, unknown
", message.getMessageId()));
                                }
                            } catch (Throwable e) {
                                // 如果提交或回滚消息时超过了TransCheckImmunityTime（针对发送事
务消息的句柄）或者超过10s（针对consumeHalfMessage的句柄）则会失败。
                                processCommitRollError(e);
                            }
                        }
```

```
                    } catch (Throwable e) {
                        System.out.println(e.getMessage());
                    }
                }
            }
        });
        t.start();
        t.join();
        mqClient.close();
    }
}
```

订阅事务消息

订阅事务消息的示例代码如下。

```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQConsumer;
import com.aliyun.mq.http.common.AckMessageException;
import com.aliyun.mq.http.model.Message;
import java.util.ArrayList;
import java.util.List;
public class Consumer {
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
                // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
                "${HTTP_ENDPOINT}",
                // AccessKey ID，阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                // AccessKey Secret，阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
        );
        // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
        //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型的消息。
        final String topic = "${TOPIC}";
        // 您在消息队列RocketMQ版控制台创建的Group ID。
        final String groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
        final String instanceId = "${INSTANCE_ID}";
        final MQConsumer consumer;
        if (instanceId != null && instanceId != "") {
            consumer = mqClient.getConsumer(instanceId, topic, groupId, null);
        } else {
            consumer = mqClient.getConsumer(topic, groupId);
        }
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
```

```
        do {
            List<Message> messages = null;
            try {
                // 长轮询消费消息。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
                messages = consumer.consumeMessage(
                        3,// 一次最多消费3条消息（最多可设置为16条）。
                        3// 长轮询时间3秒（最多可设置为30秒）。
                );
            } catch (Throwable e) {
                e.printStackTrace();
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }
            }
            // Topic中没有消息可消费。
            if (messages == null || messages.isEmpty()) {
                System.out.println(Thread.currentThread().getName() + ": no new message, continue!");
                continue;
            }
            // 处理业务逻辑。
            for (Message message : messages) {
                System.out.println("Receive message: " + message);
            }
            // 消息重试时间到达前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费的时间戳都不一样。
            {
                List<String> handles = new ArrayList<String>();
                for (Message message : messages) {
                    handles.add(message.getReceiptHandle());
                }
                try {
                    consumer.ackMessage(handles);
                } catch (Throwable e) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    if (e instanceof AckMessageException) {
                        AckMessageException errors = (AckMessageException) e;
                        System.out.println("Ack message fail, requestId is:" + errors.getRequestId() + ", fail handles:");
                        if (errors.getErrorMessages() != null) {
                            for (String errorHandle :errors.getErrorMessages().keySet()) {
                                System.out.println("Handle:" + errorHandle + ", ErrorCode:" + errors.getErrorMessages().get(errorHandle).getErrorCode()
                                        + ", ErrorMsg:" + errors.getErrorMessages().get(errorHandle).getErrorMessage());
                            }
                        }
                        continue;
                    }
                    e.printStackTrace();
```

```
                }
            }
        } while (true);
    }
}
```

3.4. Go SDK

3.4.1. 版本说明

本文介绍HTTP协议下Go SDK的版本信息，包括各版本的发布时间、发布内容以及下载链接，以便您按需获取适用的Go SDK收发消息。

1.0.3

发布时间	发布内容	下载
2021-01-05	**新特性** • 支持顺序消息。 **功能优化** • 支持设置timeout；在收发消息的返回结果中增加requestId。	mq-http-go-sdk-1.0.3.zip

1.0.1

发布时间	发布内容	下载
2019-06-10	**新特性** • 支持高级特性消息，包括定时和延时消息、事务消息，暂不支持顺序消息。 • 支持使用Message ID查询消息轨迹。 • 支持消息属性。	mq-http-go-sdk-1.0.1.zip

1.0.0

发布时间	发布内容	下载
2019-06-10	**新特性** • 首次发布。 • 暂不支持高级特性消息，包括顺序消息、事务消息、定时和延时消息。 • 已支持集群消费模式，暂不支持广播消费模式。 • 暂不支持查询消息轨迹。	mq-http-go-sdk-1.0.0.zip

3.4.2. 准备环境

在使用Go SDK收发消息前，您需按照本文提供的内容来准备环境。

环境要求

安装Golang。更多信息，请参见安装Golang。

安装完成后，您可以执行 `go version` 命令查看Go语言版本。

安装Go SDK

1. 执行以下命令启用Go mod。

   ```
   go env -w GO111MODULE=on
   ```

2. 执行以下命令设置Go mod代理。

   ```
   go env -w GOPROXY=https://goproxy.cn,direct
   ```

3. 执行以下命令进行初始化，生成*go.mod*文件。

   ```
   go mod init
   ```

4. 执行以下命令安装Go SDK。

   ```
   go get github.com/aliyunmq/mq-http-go-sdk
   ```

3.4.3. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用HTTP协议下的Go SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 安装Go SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送普通消息

发送普通消息的示例代码如下。

```
package main
import (
    "fmt"
    "time"
    "strconv"
    "github.com/aliyunmq/mq-http-go-sdk"
)
func main() {
    // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
    instanceId := "${INSTANCE_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqProducer := client.GetProducer(instanceId, topic)
    // 循环发送4条消息。
    for i := 0; i < 4; i++ {
        var msg mq_http_sdk.PublishMessageRequest
        msg = mq_http_sdk.PublishMessageRequest{
        MessageBody: "hello mq!",         //消息内容。
        MessageTag:  "",                  // 消息标签。
        Properties:  map[string]string{}, // 消息属性。
        }
        // 设置消息的Key。
        msg.MessageKey = "MessageKey"
        // 设置消息自定义属性。
        msg.Properties["a"] = strconv.Itoa(i)
        ret, err := mqProducer.PublishMessage(msg)
        if err != nil {
            fmt.Println(err)
            return
        } else {
            fmt.Printf("Publish ---->\n\tMessageId:%s, BodyMD5:%s, \n", ret.MessageId, ret.MessageBodyMD5)
        }
        time.Sleep(time.Duration(100) * time.Millisecond)
    }
}
```

订阅普通消息

订阅普通消息的示例代码如下。

```
package main
import (
    "fmt"
    "github.com/gogap/errors"
    "strings"
    "time"
    "github.com/aliyunmq/mq-http-go-sdk"
)
func main() {
    // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型的消息
。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    instanceId := "${INSTANCE_ID}"
    // 您在控制台创建的Group ID。
    groupId := "${GROUP_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqConsumer := client.GetConsumer(instanceId, topic, groupId, "")
    for {
        endChan := make(chan int)
        respChan := make(chan mq_http_sdk.ConsumeMessageResponse)
        errChan := make(chan error)
        go func() {
            select {
            case resp := <-respChan:
                {
                    // 处理业务逻辑。
                    var handles []string
                    fmt.Printf("Consume %d messages---->\n", len(resp.Messages))
                    for _, v := range resp.Messages {
                        handles = append(handles, v.ReceiptHandle)
                        fmt.Printf("\tMessageID: %s, PublishTime: %d, MessageTag: %s\n"+
                            "\tConsumedTimes: %d, FirstConsumeTime: %d, NextConsumeTime: %d
\n"+
                            "\tBody: %s\n"+
                            "\tProps: %s\n",
                            v.MessageId, v.PublishTime, v.MessageTag, v.ConsumedTimes,
                            v.FirstConsumeTime, v.NextConsumeTime, v.MessageBody, v.Propert
ies)
                    }
                    // NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
```

```
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    ackerr := mqConsumer.AckMessage(handles)
                    if ackerr != nil {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        fmt.Println(ackerr)
                        if errAckItems, ok := ackerr.(errors.ErrCode).Context()["Detail"].(
[]mq_http_sdk.ErrAckItem); ok {
                            for _, errAckItem := range errAckItems {
                              fmt.Printf("\tErrorHandle:%s, ErrorCode:%s, ErrorMsg:%s\n",
                                errAckItem.ErrorHandle, errAckItem.ErrorCode, errAckItem.Err
orMsg)
                            }
                        } else {
                            fmt.Println("ack err =", ackerr)
                        }
                        time.Sleep(time.Duration(3) * time.Second)
                    } else {
                        fmt.Printf("Ack ---->\n\t%s\n", handles)
                    }
                    endChan <- 1
                }
            case err := <-errChan:
                {
                    // Topic中没有消息可消费。
                    if strings.Contains(err.(errors.ErrCode).Error(), "MessageNotExist") {
                        fmt.Println("\nNo new message, continue!")
                    } else {
                        fmt.Println(err)
                        time.Sleep(time.Duration(3) * time.Second)
                    }
                    endChan <- 1
                }
            case <-time.After(35 * time.Second):
                {
                    fmt.Println("Timeout of consumer message ??")
                    endChan <- 1
                }
            }
        }()
        // 长轮询消费消息，网络超时时间默认为35s。
        // 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回响应。
        mqConsumer.ConsumeMessage(respChan, errChan,
            3, // 一次最多消费3条（最多可设置为16条）。
            3, // 长轮询时间3s（最多可设置为30s）。
        )
        <-endChan
    }
}
```

3.4.4. 收发顺序消息

顺序消息（FIFO消息）是

消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用HTTP协议下的Go SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息。

前提条件

您已完成以下操作：

- 安装Go SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送顺序消息

> 注意
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

```
package main
import (
    "fmt"
    "time"
    "strconv"
    "github.com/aliyunmq/mq-http-go-sdk"
)
func main() {
    // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
    instanceId := "${INSTANCE_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqProducer := client.GetProducer(instanceId, topic)
    // 循环发送8条消息。
    for i := 0; i < 8; i++ {
        msg := mq_http_sdk.PublishMessageRequest{
            MessageBody: "hello mq!",         //消息内容。
            MessageTag:  "",                  // 消息标签。
            Properties:  map[string]string{}, // 消息属性。
        }
        // 设置消息的Key。
        msg.MessageKey = "MessageKey"
        // 设置消息的自定义属性。
        msg.Properties["a"] = strconv.Itoa(i)
        // 设置分区顺序消息的Sharding Key，用于标识不同的分区。Sharding Key与消息的Key是完全不同的概念。
        msg.ShardingKey = strconv.Itoa(i % 2)
        ret, err := mqProducer.PublishMessage(msg)
        if err != nil {
            fmt.Println(err)
            return
        } else {
            fmt.Printf("Publish ---->\n\tMessageId:%s, BodyMD5:%s, \n", ret.MessageId, ret.MessageBodyMD5)
        }
        time.Sleep(time.Duration(100) * time.Millisecond)
    }
}
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
package main
import (
    "fmt"
    "github.com/gogap/errors"
    "strings"
    "time"
    "github.com/aliyunmq/mq-http-go-sdk"
)
func main() {
    // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    instanceId := "${INSTANCE_ID}"
    // 您在控制台创建的Group ID。
    groupId := "${GROUP_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqConsumer := client.GetConsumer(instanceId, topic, groupId, "")
    for {
        endChan := make(chan int)
        respChan := make(chan mq_http_sdk.ConsumeMessageResponse)
        errChan := make(chan error)
        go func() {
            select {
            case resp := <-respChan:
                {
                    // 处理业务逻辑。
                    var handles []string
                    fmt.Printf("Consume %d messages---->\n", len(resp.Messages))
                    for _, v := range resp.Messages {
                        handles = append(handles, v.ReceiptHandle)
                        fmt.Printf("\tMessageID: %s, PublishTime: %d, MessageTag: %s\n"+
                            "\tConsumedTimes: %d, FirstConsumeTime: %d, NextConsumeTime: %d
\n"+
                            "\tBody: %s\n"+
                            "\tProps: %s\n"+
                            "\tShardingKey: %s\n",
                            v.MessageId, v.PublishTime, v.MessageTag, v.ConsumedTimes,
                            v.FirstConsumeTime, v.NextConsumeTime, v.MessageBody, v.Propert
ies, v.ShardingKey)
```

```
                    }
                    // NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    ackerr := mqConsumer.AckMessage(handles)
                    if ackerr != nil {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        fmt.Println(ackerr)
                        if errAckItems, ok := ackerr.(errors.ErrCode).Context()["Detail"].(
[]mq_http_sdk.ErrAckItem); ok {
                            for _, errAckItem := range errAckItems {
                               fmt.Printf("\tErrorHandle:%s, ErrorCode:%s, ErrorMsg:%s\n",
                                  errAckItem.ErrorHandle, errAckItem.ErrorCode, errAckItem.E
rrorMsg)
                            }
                        } else {
                            fmt.Println("ack err =", ackerr)
                        }
                        time.Sleep(time.Duration(3) * time.Second)
                    } else {
                        fmt.Printf("Ack ---->\n\t%s\n", handles)
                    }
                    endChan <- 1
                }
            case err := <-errChan:
                {
                    // Topic中没有消息可消费。
                    if strings.Contains(err.(errors.ErrCode).Error(), "MessageNotExist") {
                        fmt.Println("\nNo new message, continue!")
                    } else {
                        fmt.Println(err)
                        time.Sleep(time.Duration(3) * time.Second)
                    }
                    endChan <- 1
                }
            case <-time.After(35 * time.Second):
                {
                    fmt.Println("Timeout of consumer message ??")
                    endChan <- 1
                }
            }
        }()
                // 拉取到的消息可能是多个分区的（对于分区顺序消息），一个分区内的消息一定是顺序的。
                // 对于分区顺序消息，只要一个分区内存在没有被确认消费的消息，那么该分区下次还会消费到
相同的消息。
                // 对于一个分区，只有所有消息确认消费成功才能消费下一批消息。
                // 长轮询顺序消费消息，网络超时时间为35s。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则服务
端立即返回响应。
        mqConsumer.ConsumeMessageOrderly(respChan, errChan,
            3, // 一次最多消费3条（最多可设置为16条）。
            3, // 长轮询时间3s（最多可设置为30s）。
        )
        <-endChan
    }
```

```
}
```

3.4.5. 收发定时消息和延时消息

本文提供使用HTTP协议下的Go SDK收发定时消息和延时消息的示例代码。

背景信息

- 延时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。
- 定时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

在HTTP协议下，定时消息和延时消息的代码配置相同，本质都是根据消息中的属性延迟固定时间后才投递给消费者。
更多信息，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 安装Go SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
package main
import (
    "fmt"
    "time"
    "strconv"
    "github.com/aliyunmq/mq-http-go-sdk"
)
func main() {
    // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
    instanceId := "${INSTANCE_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqProducer := client.GetProducer(instanceId, topic)
    // 循环发送4条消息。
    for i := 0; i < 4; i++ {
        var msg mq_http_sdk.PublishMessageRequest
            msg = mq_http_sdk.PublishMessageRequest{
            MessageBody: "hello mq!",         // 消息内容。
            MessageTag:  "",                  // 消息标签。
            Properties:  map[string]string{}, // 消息属性。
            }
    // 设置消息的Key。
            msg.MessageKey = "MessageKey"
            // 设置消息自定义属性。
            msg.Properties["a"] = strconv.Itoa(i)
            // 延时消息，发送时间为10s后。该参数格式为毫秒级别的时间戳。
            // 若发送定时消息，设置该参数时需要计算定时时间与当前时间的时间差。
            msg.StartDeliverTime = time.Now().UTC().Unix() * 1000 + 10 * 1000
        ret, err := mqProducer.PublishMessage(msg)
        if err != nil {
            fmt.Println(err)
            return
        } else {
            fmt.Printf("Publish ---->\n\tMessageId:%s, BodyMD5:%s, \n", ret.MessageId, ret.MessageBodyMD5)
        }
        time.Sleep(time.Duration(100) * time.Millisecond)
    }
}
```

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

```
package main
import (
    "fmt"
    "github.com/gogap/errors"
    "strings"
    "time"
    "github.com/aliyunmq/mq-http-go-sdk"
)
func main() {
    // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型的消息
。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    instanceId := "${INSTANCE_ID}"
    // 您在控制台创建的Group ID。
    groupId := "${GROUP_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqConsumer := client.GetConsumer(instanceId, topic, groupId, "")
    for {
        endChan := make(chan int)
        respChan := make(chan mq_http_sdk.ConsumeMessageResponse)
        errChan := make(chan error)
        go func() {
            select {
            case resp := <-respChan:
                {
                    // 处理业务逻辑。
                    var handles []string
                    fmt.Printf("Consume %d messages---->\n", len(resp.Messages))
                    for _, v := range resp.Messages {
                        handles = append(handles, v.ReceiptHandle)
                        fmt.Printf("\tMessageID: %s, PublishTime: %d, MessageTag: %s\n"+
                            "\tConsumedTimes: %d, FirstConsumeTime: %d, NextConsumeTime: %d
\n"+

                            "\tBody: %s\n"+
                            "\tProps: %s\n",
                            v.MessageId, v.PublishTime, v.MessageTag, v.ConsumedTimes,
                            v.FirstConsumeTime, v.NextConsumeTime, v.MessageBody, v.Propert
```

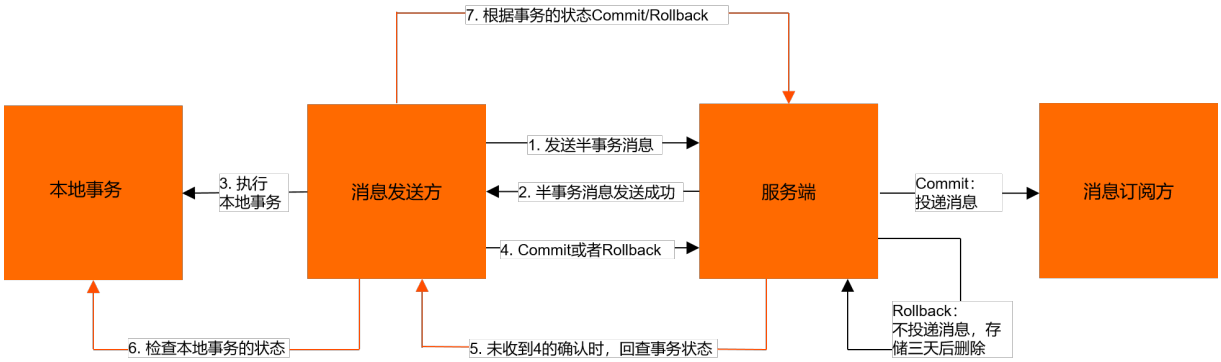
```
ies)
                    }
                    // NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    ackerr := mqConsumer.AckMessage(handles)
                    if ackerr != nil {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        fmt.Println(ackerr)
                        if errAckItems, ok := ackerr.(errors.ErrCode).Context()["Detail"].(
[]mq_http_sdk.ErrAckItem); ok {
                            for _, errAckItem := range errAckItems {
                              fmt.Printf("\tErrorHandle:%s, ErrorCode:%s, ErrorMsg:%s\n",
                                errAckItem.ErrorHandle, errAckItem.ErrorCode, errAckItem.Err
orMsg)
                            }
                        } else {
                            fmt.Println("ack err =", ackerr)
                        }
                        time.Sleep(time.Duration(3) * time.Second)
                    } else {
                        fmt.Printf("Ack ---->\n\t%s\n", handles)
                    }
                    endChan <- 1
                }
            case err := <-errChan:
                {
                    // Topic中没有消息可消费。
                    if strings.Contains(err.(errors.ErrCode).Error(), "MessageNotExist") {
                        fmt.Println("\nNo new message, continue!")
                    } else {
                        fmt.Println(err)
                        time.Sleep(time.Duration(3) * time.Second)
                    }
                    endChan <- 1
                }
            case <-time.After(35 * time.Second):
                {
                    fmt.Println("Timeout of consumer message ??")
                    endChan <- 1
                }
            }
        }()
        // 长轮询消费消息，网络超时时间默认为35s。
        // 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回响应。
        mqConsumer.ConsumeMessage(respChan, errChan,
            3, // 一次最多消费3条（最多可设置为16条）。
            3, // 长轮询时间3s（最多可设置为30s）。
        )
        <-endChan
    }
}
```

3.4.6. 收发事务消息

消息队列RocketMQ版
提供类似XA或Open XA的分布式事务功能，通过
消息队列RocketMQ版
事务消息，能达到分布式事务的最终一致。本文提供使用HTTP协议下的Go SDK收发事务消息的示例代码。

背景信息

事务消息的交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 安装Go SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送事务消息

发送事务消息的示例代码如下。

```
package main
import (
    "fmt"
    "github.com/gogap/errors"
    "strconv"
    "strings"
    "time"
    "github.com/aliyunmq/mq-http-go-sdk"
)
var loopCount = 0
func ProcessError(err error) {
    // 如果Commit或Rollback时超过了TransCheckImmunityTime（针对发送事务消息的句柄）或者超过10s（针对consumeHalfMessage的句柄），则Commit或Rollback失败。
    if err == nil {
        return
    }
    fmt.Println(err)
    for _, errAckItem := range err.(errors.ErrCode).Context()["Detail"].([]mq_http_sdk.ErrAckItem) {
        fmt.Printf("\tErrorHandle:%s, ErrorCode:%s, ErrorMsg:%s\n",
            errAckItem.ErrorHandle, errAckItem.ErrorCode, errAckItem.ErrorMsg)
```

```
    }
}
func ConsumeHalfMsg(mqTransProducer *mq_http_sdk.MQTransProducer) {
    for {
        if loopCount >= 10 {
            return
        }
        loopCount++
        endChan := make(chan int)
        respChan := make(chan mq_http_sdk.ConsumeMessageResponse)
        errChan := make(chan error)
        go func() {
            select {
            case resp := <-respChan:
                {
                    // 处理业务逻辑。
                    var handles []string
                    fmt.Printf("Consume %d messages---->\n", len(resp.Messages))
                    for _, v := range resp.Messages {
                        handles = append(handles, v.ReceiptHandle)
                        fmt.Printf("\tMessageID: %s, PublishTime: %d, MessageTag: %s\n"+
                            "\tConsumedTimes: %d, FirstConsumeTime: %d, NextConsumeTime: %d
\n\tBody: %s\n"+
                            "\tProperties:%s, Key:%s, Timer:%d, Trans:%d\n",
                            v.MessageId, v.PublishTime, v.MessageTag, v.ConsumedTimes,
                            v.FirstConsumeTime, v.NextConsumeTime, v.MessageBody,
                            v.Properties, v.MessageKey, v.StartDeliverTime, v.TransCheckImm
unityTime)
                        a, _ := strconv.Atoi(v.Properties["a"])
                        var comRollErr error
                        if a == 1 {
                            // 确认提交事务消息。
                            comRollErr = (*mqTransProducer).Commit(v.ReceiptHandle)
                            fmt.Println("Commit---------->")
                        } else if a == 2 && v.ConsumedTimes > 1 {
                            // 确认提交事务消息。
                            comRollErr = (*mqTransProducer).Commit(v.ReceiptHandle)
                            fmt.Println("Commit---------->")
                        } else if a == 3 {
                            // 确认回滚事务消息。
                            comRollErr = (*mqTransProducer).Rollback(v.ReceiptHandle)
                            fmt.Println("Rollback---------->")
                        } else {
                            // 什么都不做，下次再检查。
                            fmt.Println("Unknown---------->")
                        }
                        ProcessError(comRollErr)
                    }
                    endChan <- 1
                }
            case err := <-errChan:
                {
                    // Topic中没有消息可消费。
                    if strings.Contains(err.(errors.ErrCode).Error(), "MessageNotExist") {
```

```
                        fmt.Println("\nNo new message, continue!")
                    } else {
                        fmt.Println(err)
                        time.Sleep(time.Duration(3) * time.Second)
                    }
                    endChan <- 1
                }
            case <-time.After(35 * time.Second):
                {
                    fmt.Println("Timeout of consumer message ??")
                    return
                }
            }
        }()
        // 长轮询检查半事务消息。
        // 长轮询表示如果Topic没有消息则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回响应。
        (*mqTransProducer).ConsumeHalfMessage(respChan, errChan,
            3, // 一次最多消费3条（最多可设置为16条）。
            3, // 长轮询时间3秒（最多可设置为30秒）。
        )
        <-endChan
    }
}
func main() {
    // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型的消息
。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    instanceId := "${INSTANCE_ID}"
    // 您在控制台创建的Group ID。
    groupId := "${GROUP_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqTransProducer := client.GetTransProducer(instanceId, topic, groupId)
    // 客户端需要有一个线程或者进程来消费没有确认的事务消息。
    // 启动一个Goroutines来检查没有确认的事务消息。
    go ConsumeHalfMsg(&mqTransProducer)
    // 发送4条事务消息，1条发送完就提交，其余3条通过检查半事务消息处理。
    for i := 0; i < 4; i++ {
        msg := mq_http_sdk.PublishMessageRequest{
            MessageBody:"I am transaction msg!",
            Properties: map[string]string{"a":strconv.Itoa(i)},
```

```
        }
        // 设置事务第一次回查的时间，为相对时间。单位：秒，范围：10~300。
        // 第一次事务回查后如果消息没有Commit或者Rollback，则之后每隔10s左右会回查一次，共回查24小时
。
        msg.TransCheckImmunityTime = 10
        resp, pubErr := mqTransProducer.PublishMessage(msg)
        if pubErr != nil {
            fmt.Println(pubErr)
            return
        }
        fmt.Printf("Publish ---->\n\tMessageId:%s, BodyMD5:%s, Handle:%s\n",
            resp.MessageId, resp.MessageBodyMD5, resp.ReceiptHandle)
        if i == 0 {
            // 发送完事务消息后能获取到半消息句柄，可以直接Commit或Rollback事务消息。
            ackErr := mqTransProducer.Commit(resp.ReceiptHandle)
            fmt.Println("Commit---------->")
            ProcessError(ackErr)
        }
    }
    for ; loopCount < 10 ; {
        time.Sleep(time.Duration(1) * time.Second)
    }
}
```

订阅事务消息

订阅事务消息的示例代码如下。

```
package main
import (
    "fmt"
    "github.com/gogap/errors"
    "strings"
    "time"
    "github.com/aliyunmq/mq-http-go-sdk"
)
func main() {
    // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    endpoint := "${HTTP_ENDPOINT}"
    // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    accessKey := "${ACCESS_KEY}"
    // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    secretKey := "${SECRET_KEY}"
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    //不同消息类型的Topic不能混用，例如普通消息的Topic只能用于收发普通消息，不能用于收发其他类型的消息
。
    topic := "${TOPIC}"
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
```

```
    instanceId := "${INSTANCE_ID}"
    // 您在控制台创建的Group ID。
    groupId := "${GROUP_ID}"
    client := mq_http_sdk.NewAliyunMQClient(endpoint, accessKey, secretKey, "")
    mqConsumer := client.GetConsumer(instanceId, topic, groupId, "")
    for {
        endChan := make(chan int)
        respChan := make(chan mq_http_sdk.ConsumeMessageResponse)
        errChan := make(chan error)
        go func() {
            select {
            case resp := <-respChan:
                {
                    // 处理业务逻辑。
                    var handles []string
                    fmt.Printf("Consume %d messages---->\n", len(resp.Messages))
                    for _, v := range resp.Messages {
                        handles = append(handles, v.ReceiptHandle)
                        fmt.Printf("\tMessageID: %s, PublishTime: %d, MessageTag: %s\n"+
                            "\tConsumedTimes: %d, FirstConsumeTime: %d, NextConsumeTime: %d
\n"+
                            "\tBody: %s\n"+
                            "\tProps: %s\n",
                            v.MessageId, v.PublishTime, v.MessageTag, v.ConsumedTimes,
                            v.FirstConsumeTime, v.NextConsumeTime, v.MessageBody, v.Propert
ies)
                    }
                    // NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    ackerr := mqConsumer.AckMessage(handles)
                    if ackerr != nil {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        fmt.Println(ackerr)
                        if errAckItems, ok := ackerr.(errors.ErrCode).Context()["Detail"].(
[]mq_http_sdk.ErrAckItem); ok {
                           for _, errAckItem := range errAckItems {
                             fmt.Printf("\tErrorHandle:%s, ErrorCode:%s, ErrorMsg:%s\n",
                               errAckItem.ErrorHandle, errAckItem.ErrorCode, errAckItem.Err
orMsg)
                           }
                        } else {
                           fmt.Println("ack err =", ackerr)
                        }
                        time.Sleep(time.Duration(3) * time.Second)
                    } else {
                        fmt.Printf("Ack ---->\n\t%s\n", handles)
                    }
                    endChan <- 1
                }
            case err := <-errChan:
                {
                    // Topic中没有消息可消费。
                    if strings.Contains(err.(errors.ErrCode).Error(), "MessageNotExist") {
                        fmt.Println("\nNo new message, continue!")
                    } else {
```

```
                    } else {
                        fmt.Println(err)
                        time.Sleep(time.Duration(3) * time.Second)
                    }
                    endChan <- 1
                }
            case <-time.After(35 * time.Second):
                {
                    fmt.Println("Timeout of consumer message ??")
                    endChan <- 1
                }
            }
        }()
        // 长轮询消费消息，网络超时时间默认为35s。
        // 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回响应。
        mqConsumer.ConsumeMessage(respChan, errChan,
            3, // 一次最多消费3条（最多可设置为16条）。
            3, // 长轮询时间3s（最多可设置为30s）。
        )
        <-endChan
    }
}
```

3.5. Python SDK

3.5.1. 版本说明

本文介绍HTTP协议下Python SDK的版本信息，包括各版本的发布时间、发布内容以及下载链接，以便您按需获取适用的Python SDK收发消息。

1.0.3

发布时间	发布内容	下载
2021-01-05	**新特性** • 支持顺序消息。	mq-http-python-sdk-1.0.3.zip

1.0.1

发布时间	发布内容	下载
2019-06-10	**新特性** • 支持高级特性消息，包括定时和延时消息、事务消息，暂不支持顺序消息。 • 支持使用Message ID查询消息轨迹。 • 支持消息属性。 • 支持Python 3。	mq-http-python-sdk-1.0.1.zip

1.0.0

发布时间	发布内容	下载
2019-01-23	**新特性** • 首次发布。 • 暂不支持高级特性消息，包括顺序消息、事务消息、定时和延时消息。 • 已支持集群消费模式，暂不支持广播消费模式。 • 暂不支持查询消息轨迹。	mq-http-python-sdk-1.0.0.zip

3.5.2. 准备环境

在使用Python SDK收发消息前，您需按照本文提供的内容来准备环境。

环境要求

- 安装Python。更多信息，请参见安装Python。安装的Python版本要求如下：
  - 若您使用的SDK版本为v1.0.0，您需要安装大于等于2.5且小于3.0版本的Python。
  - 若您使用的SDK版本大于v1.0.0，您需要安装2.5及以上版本的Python。
- 安装pip。更多信息，请参见安装pip。

> **说明** Python 3.4及以上版本自带pip。若您安装的Python版本大于等于3.4，您可以不用单独安装pip。

安装完成后，您可以执行 `python -V` 命令查看Python语言版本。

安装SDK

执行以下命令，安装Python SDK。

```
pip install mq_http_sdk
```

3.5.3. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用HTTP协议下的Python SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 安装Python SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送普通消息

发送普通消息的示例代码如下。

```
import sys
from mq_http_sdk.mq_exception import MQExceptionBase
from mq_http_sdk.mq_producer import *
from mq_http_sdk.mq_client import *
import time
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
     "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
     "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
    )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
producer = mq_client.get_producer(instance_id, topic_name)
# 循环发送4条消息。
msg_count = 4
print("%sPublish Message To %s\nTopicName:%s\nMessageCount:%s\n" % (10 * "=", 10 * "=", top
ic_name, msg_count))
try:
    for i in range(msg_count):
            msg = TopicMessage(
                    # 消息内容。
                    "I am test message %s.hello" % i,
                    # 消息标签。
                    "tag %s" % i
                        )
            # 设置消息的自定义属性。
            msg.put_property("a", i)
            # 设置消息的Key。
            msg.set_message_key("MessageKey")
            re_msg = producer.publish_message(msg)
            print("Publish Message Succeed. MessageID:%s, BodyMD5:%s" % (re_msg.message_id,
re_msg.message_body_md5))
except MQExceptionBase as e:
    if e.type == "TopicNotExist":
         print("Topic not exist, please create it.")
         sys.exit(1)
    print("Publish Message Fail. Exception:%s" % e)
```

订阅普通消息

订阅普通消息的示例代码如下。

```
from mq_http_sdk.mq_exception import MQExceptionBase
```

```
from mq_http_sdk.mq_consumer import *
from mq_http_sdk.mq_client import *
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
     )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# 您在
消息队列RocketMQ版控制台创建的Group ID。
group_id = "${GROUP_ID}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
consumer = mq_client.get_consumer(instance_id, topic_name, group_id)
# 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3秒，3秒内如果有消息可以消费则立即返回响应。
# 长轮询时间3秒（最多可设置为30秒）。
wait_seconds = 3
# 一次最多消费3条（最多可设置为16条）。
batch = 3
print(("%sConsume And Ak Message From Topic%s\nTopicName:%s\nMQConsumer:%s\nWaitSeconds:%s\
n" \
        % (10 * "=", 10 * "=", topic_name, group_id, wait_seconds)))
while True:
    try:
        # 长轮询消费消息。
        recv_msgs = consumer.consume_message(batch, wait_seconds)
        for msg in recv_msgs:
            print(("Receive, MessageId: %s\nMessageBodyMD5: %s \
                            \nMessageTag: %s\nConsumedTimes: %s \
                            \nPublishTime: %s\nBody: %s \
                            \nNextConsumeTime: %s \
                            \nReceiptHandle: %s \
                            \nProperties: %s\n" % \
                  (msg.message_id, msg.message_body_md5,
                   msg.message_tag, msg.consumed_times,
                   msg.publish_time, msg.message_body,
                   msg.next_consume_time, msg.receipt_handle, msg.properties)))
            print(msg.get_property(""))
    except MQExceptionBase as e:
        # Topic中没有消息可消费。
        if e.type == "MessageNotExist":
            print(("No new message! RequestId: %s" % e.req_id))
            continue
        print(("Consume Message Fail! Exception:%s\n" % e))
```

```
            time.sleep(2)
            continue
        # msg.next_consume_time前若不确认消息消费成功，则消息会被重复消费。
        # 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
        try:
            receipt_handle_list = [msg.receipt_handle for msg in recv_msgs]
            consumer.ack_message(receipt_handle_list)
            print(("Ak %s Message Succeed.\n\n" % len(receipt_handle_list)))
        except MQExceptionBase as e:
            print(("\nAk Message Fail! Exception:%s" % e))
            # 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
            if e.sub_errors:
                for sub_error in e.sub_errors:
                    print(("\tErrorHandle:%s,ErrorCode:%s,ErrorMsg:%s" % \
                          (sub_error["ReceiptHandle"], sub_error["ErrorCode"], sub_error["Error
Message"])))
```

3.5.4. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用HTTP协议下的Python SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息。

前提条件

您已完成以下操作：

- 安装Python SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送顺序消息

> **注意**
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

```
import sys
from mq_http_sdk.mq_exception import MQExceptionBase
from mq_http_sdk.mq_producer import *
from mq_http_sdk.mq_client import *
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
     "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
     "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
    )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
producer = mq_client.get_producer(instance_id, topic_name)
# 循环发送8条消息。
msg_count = 8
print("%sPublish Message To %s\nTopicName:%s\nMessageCount:%s\n" % (10 * "=", 10 * "=", top
ic_name, msg_count))
try:
    for i in range(msg_count):
        msg = TopicMessage(
            # 消息内容。
            "I am test message %s.hello" % i,
            # 消息标签。
            "tag %s" % i
        )
        # 设置消息的自定义属性。
        msg.put_property("a", str(i))
        # 设置分区顺序消息的Sharding Key，用于标识不同的分区。Sharding Key与消息的Key是完全不同的概
念。
        msg.set_sharding_key(str(i % 3))
        re_msg = producer.publish_message(msg)
        print("Publish Message Succeed. MessageID:%s, BodyMD5:%s" % (re_msg.message_id, re_
msg.message_body_md5))
except MQExceptionBase as e:
    if e.type == "TopicNotExist":
        print("Topic not exist, please create it.")
        sys.exit(1)
    print("Publish Message Fail. Exception:%s" % e)
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
from mq_http_sdk.mq_exception import MQExceptionBase
from mq_http_sdk.mq_consumer import *
from mq_http_sdk.mq_client import *
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
     "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
     "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
    )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# 您在
消息队列RocketMQ版控制台创建的Group ID。
group_id = "${GROUP_ID}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
consumer = mq_client.get_consumer(instance_id, topic_name, group_id)
# 长轮询顺序消费消息，拉取到的消息可能是多个分区的（对于分区顺序消息），一个分区内的消息一定是顺序的。
# 对于分区顺序消息，只要一个分区内存在没有被确认消费的消息，那么该分区下次还会消费到相同的消息。
# 对于一个分区，只有所有消息确认消费成功才能消费下一批消息。
# 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则服务端立即返回响应。
wait_seconds = 3
# 一次最多消费3条（最多可设置为16条）。
batch = 3
print(("%sConsume And Ak Message From Topic%s\nTopicName:%s\nMQConsumer:%s\nWaitSeconds:%s\
n" \
        % (10 * "=", 10 * "=", topic_name, group_id, wait_seconds)))
while True:
    try:
        recv_msgs = consumer.consume_message_orderly(batch, wait_seconds)
        print("=======>Receive %d messages:" % len(recv_msgs))
        for msg in recv_msgs:
            print("\tMessageId: %s, MessageBodyMD5: %s,NextConsumeTime: %s,ConsumedTimes: %
s,PublishTime: %s\n\tBody: %s \
                    \n\tReceiptHandle: %s \
                    \n\tProperties: %s,ShardingKey: %s\n" % \
                (msg.message_id, msg.message_body_md5,
                 msg.next_consume_time, msg.consumed_times,
                 msg.publish_time, msg.message_body,
                 msg.receipt_handle, msg.properties, msg.get_sharding_key()))
    except MQExceptionBase as e:
        if e.type == "MessageNotExist":
            print(("No new message! RequestId: %s" % e.req_id))
            continue
        print(("Consume Message Fail! Exception:%s\n" % e))
        time.sleep(2)
```

```
            continue
        # msg.next_consume_time前若不确认消息消费成功，则消息会被重复消费。
        # 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
        try:
            receipt_handle_list = [msg.receipt_handle for msg in recv_msgs]
            consumer.ack_message(receipt_handle_list)
            print(("========>Ak %s Message Succeed.\n\n" % len(receipt_handle_list)))
        except MQExceptionBase as e:
            print(("\nAk Message Fail! Exception:%s" % e))
            # 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
            if e.sub_errors:
                for sub_error in e.sub_errors:
                    print(("\tErrorHandle:%s,ErrorCode:%s,ErrorMsg:%s" % \
                          (sub_error["ReceiptHandle"], sub_error["ErrorCode"], sub_error["Error
Message"])))
```

3.5.5. 收发定时消息和延时消息

本文提供使用HTTP协议下的Python SDK收发定时消息和延时消息的示例代码。

背景信息

- 延时消息：Producer将消息发送到
  消息队列RocketMQ版
  服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。
- 定时消息：Producer将消息发送到
  消息队列RocketMQ版
  服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

在HTTP协议下，定时消息和延时消息的代码配置相同，本质都是根据消息中的属性延迟固定时间后才投递给消费者。
更多信息，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 安装Python SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
import sys
from mq_http_sdk.mq_exception import MQExceptionBase
from mq_http_sdk.mq_producer import *
from mq_http_sdk.mq_client import *
import time
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
     "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
     "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
    )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
producer = mq_client.get_producer(instance_id, topic_name)
# 循环发送4条消息。
msg_count = 4
print("%sPublish Message To %s\nTopicName:%s\nMessageCount:%s\n" % (10 * "=", 10 * "=", top
ic_name, msg_count))
try:
    for i in range(msg_count):
            msg = TopicMessage(
            # 消息内容。
            "I am test message %s.hello" % i,
            # 消息标签。
            "tag1"
            )
            # 设置属性。
            msg.put_property("a", "i")
            # 设置消息的Key。
            msg.set_message_key("MessageKey")
            # 延时消息，发送时间为10s后。该参数格式为毫秒级别的时间戳。
            # 若发送定时消息，设置该参数时需要计算定时时间与当前时间的时间差。
            msg.set_start_deliver_time(int(round(time.time() * 1000)) + 10 * 1000)
            re_msg = producer.publish_message(msg)
            print("Publish Timer Message Succeed. MessageID:%s, BodyMD5:%s" % (re_msg.messa
ge_id, re_msg.message_body_md5))
except MQExceptionBase as e:
    if e.type == "TopicNotExist":
        print("Topic not exist, please create it.")
        sys.exit(1)
    print("Publish Message Fail. Exception:%s" % e)
```

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

```
from mq_http_sdk.mq_exception import MQExceptionBase
from mq_http_sdk.mq_consumer import *
from mq_http_sdk.mq_client import *
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
     )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# 您在
消息队列RocketMQ版控制台创建的Group ID。
group_id = "${GROUP_ID}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
consumer = mq_client.get_consumer(instance_id, topic_name, group_id)
# 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3秒，3秒内如果有消息可以消费则立即返回响应。
# 长轮询时间3秒（最多可设置为30秒）。
wait_seconds = 3
# 一次最多消费3条（最多可设置为16条）。
batch = 3
print(("%sConsume And Ak Message From Topic%s\nTopicName:%s\nMQConsumer:%s\nWaitSeconds:%s\
n" \
        % (10 * "=", 10 * "=", topic_name, group_id, wait_seconds)))
while True:
    try:
        # 长轮询消费消息。
        recv_msgs = consumer.consume_message(batch, wait_seconds)
        for msg in recv_msgs:
            print(("Receive, MessageId: %s\nMessageBodyMD5: %s \
                            \nMessageTag: %s\nConsumedTimes: %s \
                            \nPublishTime: %s\nBody: %s \
                            \nNextConsumeTime: %s \
                            \nReceiptHandle: %s \
                            \nProperties: %s\n" % \
                  (msg.message_id, msg.message_body_md5,
                   msg.message_tag, msg.consumed_times,
                   msg.publish_time, msg.message_body,
                   msg.next_consume_time, msg.receipt_handle, msg.properties)))
            print(msg.get_property(""))
    except MQExceptionBase as e:
        # Topic中没有消息可消费。
```

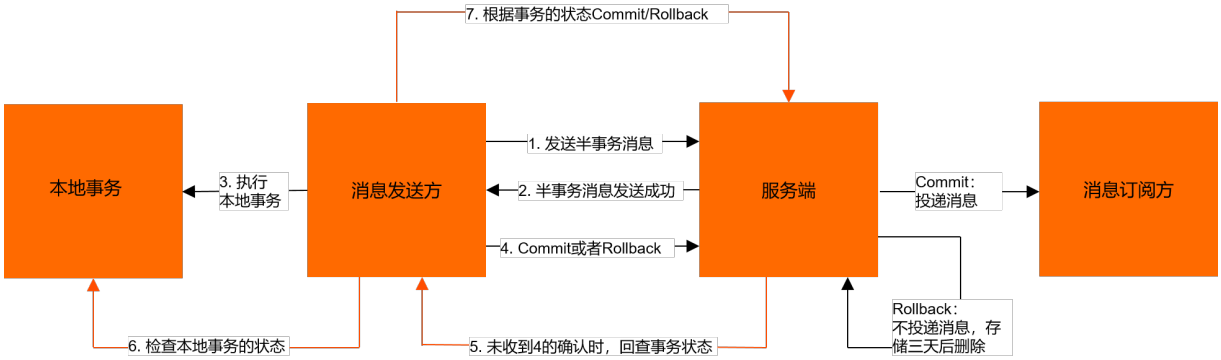
```
        if e.type == "MessageNotExist":
            print(("No new message! RequestId: %s" % e.req_id))
            continue
        print(("Consume Message Fail! Exception:%s\n" % e))
        time.sleep(2)
        continue
    # msg.next_consume_time前若不确认消息消费成功，则消息会被重复消费。
    # 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
    try:
        receipt_handle_list = [msg.receipt_handle for msg in recv_msgs]
        consumer.ack_message(receipt_handle_list)
        print(("Ak %s Message Succeed.\n\n" % len(receipt_handle_list)))
    except MQExceptionBase as e:
        print(("\nAk Message Fail! Exception:%s" % e))
        # 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
        if e.sub_errors:
            for sub_error in e.sub_errors:
                print(("\tErrorHandle:%s,ErrorCode:%s,ErrorMsg:%s" % \
                      (sub_error["ReceiptHandle"], sub_error["ErrorCode"], sub_error["Error
Message"])))
```

3.5.6. 收发事务消息

消息队列RocketMQ版
提供类似XA或Open XA的分布式事务功能，通过
消息队列RocketMQ版
事务消息，能达到分布式事务的最终一致。本文提供使用HTTP协议下的Python SDK收发事务消息的示例代码。

背景信息

事务消息的交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 安装Python SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送事务消息

发送事务消息的示例代码如下。

```
#!/usr/bin/env python
# coding=utf8
import sys
from mq_http_sdk.mq_exception import MQExceptionBase
from mq_http_sdk.mq_producer import *
from mq_http_sdk.mq_client import *
import time
import threading
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
     "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
     "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
    )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# 您在
消息队列RocketMQ版控制台创建的Group ID。
group_id = "${GROUP_ID}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
# 循环发送4条事务消息。
msg_count = 4
print("%sPublish Transaction Message To %s\nTopicName:%s\nMessageCount:%s\n" \
      % (10 * "=", 10 * "=", topic_name, msg_count))
def process_trans_error(exp):
    print("\nCommit/Roll Transaction Message Fail! Exception:%s" % exp)
    # 如果Commit或Rollback时超过了TransCheckImmunityTime（针对发送事务消息的句柄）或者超过10s（针
对consumeHalfMessage的句柄），则Commit或Rollback失败。
    if exp.sub_errors:
        for sub_error in exp.sub_errors:
            print("\tErrorHandle:%s,ErrorCode:%s,ErrorMsg:%s" % \
                  (sub_error["ReceiptHandle"], sub_error["ErrorCode"], sub_error["ErrorMess
age"]))
# 客户端需要有一个线程或者进程来消费没有确认的事务消息。
# 启动一个线程来检查没有确认的事务消息。
class ConsumeHalfMessageThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.count = 0
        # 重新New一个Client
        self.mq_client = MQClient(
                # 设置HTTP接入域名，进入
```

```
消息队列RocketMQ版控制台实例详情页面的HTTP接入点区域查看。
                "${HTTP_ENDPOINT}",
                # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
                "${ACCESS_KEY}",
                # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
                "${SECRET_KEY}"
                )
        self.trans_producer = self.mq_client.get_trans_producer(instance_id, topic_name, gr
oup_id)
    def run(self):
        while 1:
            if self.count == 3:
                break;
            try:
                half_msgs = self.trans_producer.consume_half_message(1, 3)
                for half_msg in half_msgs:
                    print("Receive Half Message, MessageId: %s\nMessageBodyMD5: %s \
                                                \nMessageTag: %s\nConsumedTimes: %s \
                                                \nPublishTime: %s\nBody: %s \
                                                \nNextConsumeTime: %s \
                                                \nReceiptHandle: %s \
                                                \nProperties: %s" % \
                          (half_msg.message_id, half_msg.message_body_md5,
                           half_msg.message_tag, half_msg.consumed_times,
                           half_msg.publish_time, half_msg.message_body,
                           half_msg.next_consume_time, half_msg.receipt_handle, half_msg.pr
operties))
                a = int(half_msg.get_property("a"))
                try:
                    if a == 1:
                        # 确认提交事务消息。
                        self.trans_producer.commit(half_msg.receipt_handle)
                        self.count += 1
                        print("------>commit")
                    elif a == 2 and half_msg.consumed_times > 1:
                        # 确认提交事务消息。
                        self.trans_producer.commit(half_msg.receipt_handle)
                        self.count += 1
                        print("------>commit")
                    elif a == 3:
                        # 确认回滚事务消息。
                        self.trans_producer.rollback(half_msg.receipt_handle)
                        self.count += 1
                        print("------>rollback")
                    else:
                        # 什么都不做，下次再检查。
                        print("------>unknown")
                except MQExceptionBase as rec_commit_roll_e:
                    process_trans_error(rec_commit_roll_e)
            except MQExceptionBase as half_e:
                if half_e.type == "MessageNotExist":
                    print("No half message! RequestId: %s" % half_e.req_id)
                    continue
                print("Consume half message Fail! Exception:%s\n" % half_e)
```

```
                break
consume_half_thread = ConsumeHalfMessageThread()
consume_half_thread.setDaemon(True)
consume_half_thread.start()
try:
    trans_producer = mq_client.get_trans_producer(instance_id, topic_name, group_id)
    for i in range(msg_count):
        msg = TopicMessage(
            # 消息内容。
            "I am test message %s." % i,
            # 消息标签。
            "tagA"
        )
        # 设置消息的自定义属性。
        msg.put_property("a", i)
        # 设置消息的Key。
        msg.set_message_key("MessageKey")
        # 设置事务第一次回查的时间，为相对时间。单位：秒，范围：10~300。
        # 第一次事务回查后如果消息没有Commit或者Rollback，则之后每隔10s左右会回查一次，共回查24小时
。
        msg.set_trans_check_immunity_time(10)
        re_msg = trans_producer.publish_message(msg)
        print("Publish Transaction Message Succeed. MessageID:%s, BodyMD5:%s, Handle:%s" \
              % (re_msg.message_id, re_msg.message_body_md5, re_msg.receipt_handle))
        time.sleep(1)
        if i == 0:
            # 发送完事务消息后能获取到半消息句柄，可以直接Commit或Rollback事务消息。
            try:
                trans_producer.commit(re_msg.receipt_handle)
            except MQExceptionBase as pub_commit_roll_e:
                process_trans_error(pub_commit_roll_e)
except MQExceptionBase as pub_e:
    if pub_e.type == "TopicNotExist":
        print("Topic not exist, please create it.")
        sys.exit(1)
    print("Publish Message Fail. Exception:%s" % pub_e)
while 1:
    if not consume_half_thread.is_alive():
        break
    time.sleep(1)
```

订阅事务消息

订阅事务消息的示例代码如下。

```
from mq_http_sdk.mq_exception import MQExceptionBase
from mq_http_sdk.mq_consumer import *
from mq_http_sdk.mq_client import *
# 初始化client。
mq_client = MQClient(
    # 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
    "${HTTP_ENDPOINT}",
    # AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
    "${ACCESS_KEY}"
```

```
    "${ACCESS_KEY}",
    # AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
    "${SECRET_KEY}"
     )
# 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
topic_name = "${TOPIC}"
# 您在
消息队列RocketMQ版控制台创建的Group ID。
group_id = "${GROUP_ID}"
# Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
# 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入空字符串。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
instance_id = "${INSTANCE_ID}"
consumer = mq_client.get_consumer(instance_id, topic_name, group_id)
# 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3秒，3秒内如果有消息可以消费则立即返回响应。
# 长轮询时间3秒（最多可设置为30秒）。
wait_seconds = 3
# 一次最多消费3条（最多可设置为16条）。
batch = 3
print(("%sConsume And Ak Message From Topic%s\nTopicName:%s\nMQConsumer:%s\nWaitSeconds:%s\
n" \
        % (10 * "=", 10 * "=", topic_name, group_id, wait_seconds)))
while True:
    try:
        # 长轮询消费消息。
        recv_msgs = consumer.consume_message(batch, wait_seconds)
        for msg in recv_msgs:
            print(("Receive, MessageId: %s\nMessageBodyMD5: %s \
                                \nMessageTag: %s\nConsumedTimes: %s \
                                \nPublishTime: %s\nBody: %s \
                                \nNextConsumeTime: %s \
                                \nReceiptHandle: %s \
                                \nProperties: %s\n" % \
                  (msg.message_id, msg.message_body_md5,
                   msg.message_tag, msg.consumed_times,
                   msg.publish_time, msg.message_body,
                   msg.next_consume_time, msg.receipt_handle, msg.properties)))
            print(msg.get_property(""))
    except MQExceptionBase as e:
        # Topic中没有消息可消费。
        if e.type == "MessageNotExist":
            print(("No new message! RequestId: %s" % e.req_id))
            continue
        print(("Consume Message Fail! Exception:%s\n" % e))
        time.sleep(2)
        continue
    # msg.next_consume_time前若不确认消息消费成功，则消息会被重复消费。
    # 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
    try:
        receipt_handle_list = [msg.receipt_handle for msg in recv_msgs]
        consumer.ack_message(receipt_handle_list)
        print(("Ak %s Message Succeed.\n\n" % len(receipt_handle_list)))
    except MQExceptionBase as e:
```

```
        print(("\nAk Message Fail! Exception:%s" % e))
        # 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
        if e.sub_errors:
            for sub_error in e.sub_errors:
                print(("\tErrorHandle:%s,ErrorCode:%s,ErrorMsg:%s" % \
                      (sub_error["ReceiptHandle"], sub_error["ErrorCode"], sub_error["Error
Message"])))
```

3.6. Node.js SDK

3.6.1. 版本说明

本文介绍HTTP协议下Node.js SDK的版本信息，包括各版本的发布时间、发布内容以及下载链接，以便您按需获取适用的Node.js SDK收发消息。

1.0.3

发布时间	发布内容	下载
2021-01-05	**新特性** • 支持顺序消息。	mq-http-nodejs-sdk-1.0.3.zip

1.0.1

发布时间	发布内容	下载
2019-06-10	**新特性** • 支持高级特性消息，包括定时和延时消息、事务消息，暂不支持顺序消息。 • 支持使用Message ID查询消息轨迹。 • 支持消息属性。	mq-http-nodejs-sdk-1.0.1.zip

1.0.0

发布时间	发布内容	下载
2019-01-23	**新特性** • 首次发布。 • 暂不支持高级特性消息，包括顺序消息、事务消息、定时和延时消息。 • 已支持集群消费模式，暂不支持广播消费模式。 • 暂不支持查询消息轨迹。	mq-http-nodejs-sdk-1.0.0.zip

3.6.2. 准备环境

在使用Node.js SDK收发消息前，您需按照本文提供的内容来准备环境。

环境要求

环境要求

安装Node.js 7.6.0或以上版本。更多信息，请参见安装Node.js。
安装完成后，您可以执行 `node -V` 命令查看Node.js语言版本。

安装Node.js SDK

执行以下命令安装Node.js SDK。

```
npm i @aliyunmq/mq-http-sdk
```

3.6.3. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用HTTP协议下的Node.js SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 安装Node.js SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送普通消息

发送普通消息的示例代码如下。

```
const {
  MQClient,
  MessageProperties
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const producer = client.getProducer(instanceId, topic);
(async function(){
  try {
    // 循环发送4条消息。
    for(var i = 0; i < 4; i++) {
      let res;
      msgProps = new MessageProperties();
      // 设置消息的自定义属性。
      msgProps.putProperty("a", i);
      // 设置消息的Key。
      msgProps.messageKey("MessageKey");
      res = await producer.publishMessage("hello mq.", "", msgProps);
      console.log("Publish message: MessageID:%s,BodyMD5:%s", res.body.MessageId, res.body.
MessageBodyMD5);
    }
  } catch(e) {
    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
    console.log(e)
  }
})();
```

订阅普通消息

订阅普通消息的示例代码如下。

```
const {
  MQClient
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
```

```
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
// 您在
消息队列RocketMQ版控制台创建的Group ID。
const groupId = "${GROUP_ID}";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const consumer = client.getConsumer(instanceId, topic, groupId);
(async function(){
  // 循环消费消息。
  while(true) {
    try {
      // 长轮询消费消息。
      // 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回响应。
      res = await consumer.consumeMessage(
          3, // 一次最多消费3条（最多可设置为16条）。
          3 // 长轮询时间3秒（最多可设置为30秒）。
          );
      if (res.code == 200) {
        // 消息消费处理逻辑。
        console.log("Consume Messages, requestId:%s", res.requestId);
        const handles = res.body.map((message) => {
          console.log("\tMessageId:%s,Tag:%s,PublishTime:%d,NextConsumeTime:%d,FirstConsumeTime:%d,ConsumedTimes:%d,Body:%s" +
            ",Props:%j,MessageKey:%s,Prop-A:%s",
              message.MessageId, message.MessageTag, message.PublishTime, message.NextConsumeTime, message.FirstConsumeTime, message.ConsumedTimes,
              message.MessageBody,message.Properties,message.MessageKey,message.Properties.a);
          return message.ReceiptHandle;
        });
        // message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
        // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
        res = await consumer.ackMessage(handles);
        if (res.code != 204) {
          // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
          console.log("Ack Message Fail:");
          const failHandles = res.body.map((error)=>{
            console.log("\tErrorHandle:%s, Code:%s, Reason:%s\n", error.ReceiptHandle, error.ErrorCode, error.ErrorMessage);
            return error.ReceiptHandle;
          });
          handles.forEach((handle)=>{
            if (failHandles.indexOf(handle) < 0) {
              console.log("\tSucHandle:%s\n", handle);
```

```
            }
          });
        } else {
          // 确认消息消费成功。
          console.log("Ack Message suc, RequestId:%s\n\t", res.requestId, handles.join(',')
);
        }
      }
    } catch(e) {
      if (e.Code.indexOf("MessageNotExist") > -1) {
        // 没有消息，则继续长轮询服务器。
        console.log("Consume Message: no new message, RequestId:%s, Code:%s", e.RequestId,
e.Code);
      } else {
        console.log(e);
      }
    }
  }
})();
```

3.6.4. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用HTTP协议下的Node.js SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息。

前提条件

您已完成以下操作：

- 安装Node.js SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送顺序消息

> **注意**
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

```
const {
  MQClient,
  MessageProperties
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const producer = client.getProducer(instanceId, topic);
(async function(){
  try {
    // 循环发送8条消息。
    for(var i = 0; i < 8; i++) {
      msgProps = new MessageProperties();
      // 设置消息的自定义属性。
      msgProps.putProperty("a", i);
      // 设置分区顺序消息的Sharding Key，用于标识不同的分区。Sharding Key与消息的Key是完全不同的概念。
      msgProps.shardingKey(i % 2);
      res = await producer.publishMessage("hello mq.", "", msgProps);
      console.log("Publish message: MessageID:%s,BodyMD5:%s", res.body.MessageId, res.body.MessageBodyMD5);
    }
  } catch(e) {
    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
    console.log(e)
  }
})();
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
const {
  MQClient,
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
// 您在
消息队列RocketMQ版控制台创建的Group ID。
const groupId = "GID_http";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const consumer = client.getConsumer(instanceId, topic, groupId);
(async function(){
  // 循环消费消息。
  while(true) {
    try {
      // 长轮询顺序消费消息，拉取到的消息可能是多个分区的（对于分区顺序消息），一个分区内的消息一定是顺序的。
      // 对于分区顺序消息，只要一个分区内存在没有被确认消费的消息，那么该分区下次还会消费到相同的消息。
      // 对于一个分区，只有所有消息确认消费成功才能消费下一批消息。
      // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则服务端立即返回响应。
      res = await consumer.consumeMessageOrderly(
          3, // 一次最多消费3条（最多可设置为16条）。
          3 // 长轮询时间3秒（最多可设置为30秒）。
          );
      if (res.code == 200) {
        // 消息消费处理逻辑。
        console.log("Consume Messages, requestId:%s", res.requestId);
        const handles = res.body.map((message) => {
          console.log("\tMessageId:%s,Tag:%s,PublishTime:%d,NextConsumeTime:%d,FirstConsumeTime:%d,ConsumedTimes:%d,Body:%s" +
            ",Props:%j,ShardingKey:%s,Prop-A:%s,Tag:%s",
              message.MessageId, message.MessageTag, message.PublishTime, message.NextConsumeTime, message.FirstConsumeTime, message.ConsumedTimes,
              message.MessageBody,message.Properties,message.ShardingKey,message.Properties.a,message.MessageTag);
          return message.ReceiptHandle;
        });
        // message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
        // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
```

```
        // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
        res = await consumer.ackMessage(handles);
        if (res.code != 204) {
          // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
          console.log("Ack Message Fail:");
          const failHandles = res.body.map((error)=>{
            console.log("\tErrorHandle:%s, Code:%s, Reason:%s\n", error.ReceiptHandle, erro
r.ErrorCode, error.ErrorMessage);
            return error.ReceiptHandle;
          });
          handles.forEach((handle)=>{
            if (failHandles.indexOf(handle) < 0) {
              console.log("\tSucHandle:%s\n", handle);
            }
          });
        } else {
          // 消息确认消费成功。
          console.log("Ack Message suc, RequestId:%s\n\t", res.requestId, handles.join(',')
);
        }
      }
    } catch(e) {
      if (e.Code.indexOf("MessageNotExist") > -1) {
        // 没有消息，则继续长轮询服务器。
        console.log("Consume Message: no new message, RequestId:%s, Code:%s", e.RequestId,
e.Code);
      } else {
        console.log(e);
      }
    }
  }
})();
```

3.6.5. 收发定时消息和延时消息

本文提供使用HTTP协议下的Node.js SDK收发定时消息和延时消息的示例代码。

背景信息

- 延时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。
- 定时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

在HTTP协议下，定时消息和延时消息的代码配置相同，本质都是根据消息中的属性延迟固定时间后才投递给消费者。
更多信息，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 安装Node.js SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
const {
  MQClient,
  MessageProperties
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const producer = client.getProducer(instanceId, topic);
(async function(){
  try {
    // 循环发送4条消息。
    for(var i = 0; i < 4; i++) {
      let res;
      msgProps = new MessageProperties();
      // 设置消息的自定义属性。
      msgProps.putProperty("a", i);
      // 设置消息的Key。
      msgProps.messageKey("MessageKey");
      // 延时消息，发送时间为10s后。该参数格式为毫秒级别的时间戳。
      // 若发送定时消息，设置该参数时需要计算定时时间与当前时间的时间差。
      msgProps.startDeliverTime(Date.now() + 10 * 1000);
      res = await producer.publishMessage("hello mq. timer msg!", "TagA", msgProps);
      console.log("Publish message: MessageID:%s,BodyMD5:%s", res.body.MessageId, res.body.
MessageBodyMD5);
    }
  } catch(e) {
    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
    console.log(e)
  }
})();
```

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

```
const {
  MQClient
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
// 您在
消息队列RocketMQ版控制台创建的Group ID。
const groupId = "${GROUP_ID}";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const consumer = client.getConsumer(instanceId, topic, groupId);
(async function(){
  // 循环消费消息。
  while(true) {
    try {
      // 长轮询消费消息。
      // 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回响应。
      res = await consumer.consumeMessage(
          3, // 一次最多消费3条（最多可设置为16条）。
          3 // 长轮询时间3秒（最多可设置为30秒）。
          );
      if (res.code == 200) {
        // 消息消费处理逻辑。
        console.log("Consume Messages, requestId:%s", res.requestId);
        const handles = res.body.map((message) => {
          console.log("\tMessageId:%s,Tag:%s,PublishTime:%d,NextConsumeTime:%d,FirstConsumeTime:%d,ConsumedTimes:%d,Body:%s" +
            ",Props:%j,MessageKey:%s,Prop-A:%s",
              message.MessageId, message.MessageTag, message.PublishTime, message.NextConsumeTime, message.FirstConsumeTime, message.ConsumedTimes,
              message.MessageBody,message.Properties,message.MessageKey,message.Properties.a);
          return message.ReceiptHandle;
        });
        // message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
        // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
        res = await consumer.ackMessage(handles);
```

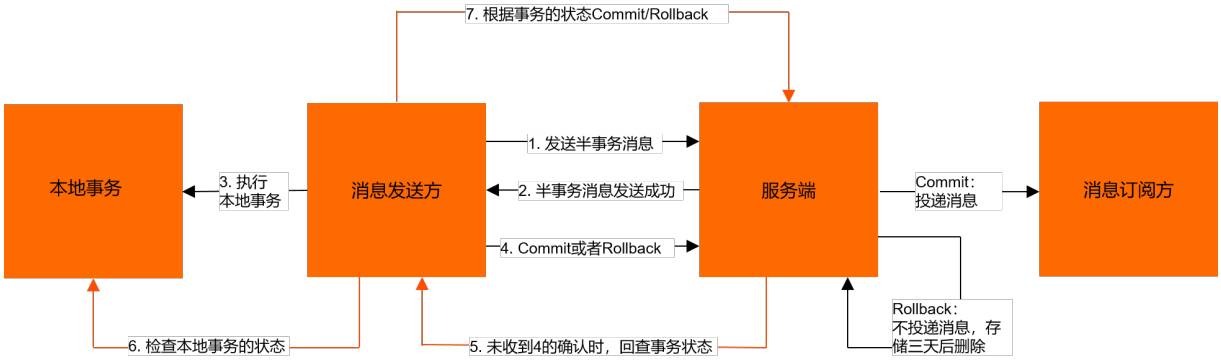
```
        if (res.code != 204) {
          // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
          console.log("Ack Message Fail:");
          const failHandles = res.body.map((error)=>{
            console.log("\tErrorHandle:%s, Code:%s, Reason:%s\n", error.ReceiptHandle, erro
r.ErrorCode, error.ErrorMessage);
            return error.ReceiptHandle;
          });
          handles.forEach((handle)=>{
            if (failHandles.indexOf(handle) < 0) {
              console.log("\tSucHandle:%s\n", handle);
            }
          });
        } else {
          // 确认消息消费成功。
          console.log("Ack Message suc, RequestId:%s\n\t", res.requestId, handles.join(',')
);
        }
      }
    } catch(e) {
      if (e.Code.indexOf("MessageNotExist") > -1) {
        // 没有消息，则继续长轮询服务器。
        console.log("Consume Message: no new message, RequestId:%s, Code:%s", e.RequestId,
e.Code);
      } else {
        console.log(e);
      }
    }
  }
})();
```

3.6.6. 收发事务消息

消息队列RocketMQ版
提供类似XA或Open XA的分布式事务功能，通过
消息队列RocketMQ版
事务消息，能达到分布式事务的最终一致。本文提供使用HTTP协议下的Node.js SDK收发事务消息的示例代码。

背景信息

事务消息的交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 安装Node.js SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送事务消息

发送事务消息的示例代码如下。

```
const {
  MQClient,
  MessageProperties
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
// 您在
消息队列RocketMQ版控制台创建的Group ID。
const groupId = "${GROUP_ID}";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const mqTransProducer = client.getTransProducer(instanceId, topic, groupId);
async function processTransResult(res, msgId) {
    if (!res) {
        return;
    }
```

```
    if (res.code != 204) {
      // 如果Commit或Rollback时超过了TransCheckImmunityTime（针对发送事务消息的句柄）或者超过10s（针对consumeHalfMessage的句柄），则Commit或Rollback失败。
      console.log("Commit/Rollback Message Fail:");
      const failHandles = res.body.map((error) => {
        console.log("\tErrorHandle:%s, Code:%s, Reason:%s\n", error.ReceiptHandle, error.ErrorCode, error.ErrorMessage);
        return error.ReceiptHandle;
      });
    } else {
      console.log("Commit/Rollback Message suc!!! %s", msgId);
    }
}
var halfMessageCount = 0;
var halfMessageConsumeCount = 0;
(async function(){
  try {
    // 循环发送4条事务消息。
    for(var i = 0; i < 4; i++) {
      let res;
      msgProps = new MessageProperties();
      // 设置消息的自定义属性。
      msgProps.putProperty("a", i);
      // 设置消息的Key。
      msgProps.messageKey("MessageKey");
      // 设置事务第一次回查的时间，为相对时间。单位：秒，范围：10~300。
      // 第一次事务回查后如果消息没有Commit或者Rollback，则之后每隔10s左右会回查一次，共回查24小时。
      msgProps.transCheckImmunityTime(10);
      res = await mqTransProducer.publishMessage("hello mq.", "tagA", msgProps);
      console.log("Publish message: MessageID:%s,BodyMD5:%s,Handle:%s", res.body.MessageId, res.body.MessageBodyMD5, res.body.ReceiptHandle);
      if (res && i == 0) {
         // 发送完事务消息后能获取到半消息句柄，可以直接Commit或Rollback事务消息。
          const msgId = res.body.MessageId;
          res = await mqTransProducer.commit(res.body.ReceiptHandle);
          console.log("Commit msg when publish, %s", msgId);
          // 如果Commit或Rollback时超过了TransCheckImmunityTime则会失败。
        processTransResult(res, msgId);
      }
    }
  } catch(e) {
    // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
    console.log(e)
  }
})();
// 客户端需要有一个线程或者进程来消费没有确认的事务消息。
// 检查没有确认的事务消息。
(async function() {
  // 循环检查事务半消息，类似消费普通消息。
  while(halfMessageCount < 3 && halfMessageConsumeCount < 15) {
    try {
        halfMessageConsumeCount++;
      res = await mqTransProducer.consumeHalfMessage(3, 3);
      if (res.code == 200) {
        // 消息消费处理逻辑
```

```
        // 消息消费处理逻辑。
        console.log("Consume Messages, requestId:%s", res.requestId);
        res.body.forEach(async (message) => {
          console.log("\tMessageId:%s,Tag:%s,PublishTime:%d,NextConsumeTime:%d,FirstConsume
Time:%d,ConsumedTimes:%d,Body:%s" +
            ",Props:%j,MessageKey:%s,Prop-A:%s",
            message.MessageId, message.MessageTag, message.PublishTime, message.NextConsu
meTime, message.FirstConsumeTime, message.ConsumedTimes,
            message.MessageBody,message.Properties,message.MessageKey,message.Properties.
a);
          var propA = message.Properties && message.Properties.a ? parseInt(message.Propert
ies.a) : 0;
                  var opResp;
                  if (propA == 1 || (propA == 2 && message.ConsumedTimes > 1)) {
                      opResp = await mqTransProducer.commit(message.ReceiptHandle);
                      console.log("Commit msg when check half, %s", message.MessageId);
                      halfMessageCount++;
                  } else if (propA == 3) {
                      opResp = await mqTransProducer.rollback(message.ReceiptHandle);
                      console.log("Rollback msg when check half, %s", message.MessageId);
                      halfMessageCount++;
                  }
                  processTransResult(opResp, message.MessageId);
        });
      }
    } catch(e) {
      if (e.Code && e.Code.indexOf("MessageNotExist") > -1) {
        // 没有消息，则继续长轮询服务器。
        console.log("Consume Transaction Half msg: no new message, RequestId:%s, Code:%s",
e.RequestId, e.Code);
      } else {
        console.log(e);
      }
    }
  }
})();
```

订阅事务消息

订阅事务消息的示例代码如下。

```
const {
  MQClient
} = require('@aliyunmq/mq-http-sdk');
// 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
const endpoint = "${HTTP_ENDPOINT}";
// AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
const accessKeyId = "${ACCESS_KEY}";
// AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
const accessKeySecret = "${SECRET_KEY}";
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);
// 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
const topic = "${TOPIC}";
```

```
// 您在
消息队列RocketMQ版控制台创建的Group ID。
const groupId = "${GROUP_ID}";
// Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
// 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
const instanceId = "${INSTANCE_ID}";
const consumer = client.getConsumer(instanceId, topic, groupId);
(async function(){
  // 循环消费消息。
  while(true) {
    try {
      // 长轮询消费消息。
      // 长轮询表示如果Topic没有消息，则客户端请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回响应。
      res = await consumer.consumeMessage(
          3, // 一次最多消费3条（最多可设置为16条）。
          3 // 长轮询时间3秒（最多可设置为30秒）。
          );
      if (res.code == 200) {
        // 消息消费处理逻辑。
        console.log("Consume Messages, requestId:%s", res.requestId);
        const handles = res.body.map((message) => {
          console.log("\tMessageId:%s,Tag:%s,PublishTime:%d,NextConsumeTime:%d,FirstConsumeTime:%d,ConsumedTimes:%d,Body:%s" +
            ",Props:%j,MessageKey:%s,Prop-A:%s",
              message.MessageId, message.MessageTag, message.PublishTime, message.NextConsumeTime, message.FirstConsumeTime, message.ConsumedTimes,
              message.MessageBody,message.Properties,message.MessageKey,message.Properties.a);
          return message.ReceiptHandle;
        });
        // message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
        // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
        res = await consumer.ackMessage(handles);
        if (res.code != 204) {
          // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
          console.log("Ack Message Fail:");
          const failHandles = res.body.map((error)=>{
            console.log("\tErrorHandle:%s, Code:%s, Reason:%s\n", error.ReceiptHandle, error.ErrorCode, error.ErrorMessage);
            return error.ReceiptHandle;
          });
          handles.forEach((handle)=>{
            if (failHandles.indexOf(handle) < 0) {
              console.log("\tSucHandle:%s\n", handle);
            }
          });
        } else {
          // 确认消息消费成功。
          console.log("Ack Message suc, RequestId:%s\n\t", res.requestId, handles.join(','));
```

```
        }
      }
    } catch(e) {
      if (e.Code.indexOf("MessageNotExist") > -1) {
        // 没有消息，则继续长轮询服务器。
        console.log("Consume Message: no new message, RequestId:%s, Code:%s", e.RequestId,
e.Code);
      } else {
        console.log(e);
      }
    }
  }
})();
```

3.7. PHP SDK

3.7.1. 版本说明

本文介绍HTTP协议下PHP SDK的版本信息，包括各版本的发布时间、发布内容以及下载链接，以便您按需获取适用的PHP SDK收发消息。

1.0.3.1

发布时间	发布内容	下载
2021-12-13	**问题修复** • 修复了PHP 5.5的兼容性问题。	mq-http-php-sdk-1.0.3.1.zip

1.0.3

发布时间	发布内容	下载
2021-01-05	**新特性** • 支持顺序消息。 **问题修复** • 修复了GuzzleHttp 7的兼容性问题。	mq-http-php-sdk-1.0.3.zip

1.0.1

发布时间	发布内容	下载
2019-06-10	**新特性** • 支持高级特性消息，包括定时和延时消息、事务消息，暂不支持顺序消息。 • 支持使用Message ID查询消息轨迹。 • 支持消息属性。	mq-http-php-sdk-1.0.1.zip

1.0.0

发布时间	发布内容	下载
2019-01-23	**新特性** • 首次发布。 • 暂不支持高级特性消息，包括顺序消息、事务消息、定时和延时消息。 • 已支持集群消费模式，暂不支持广播消费模式。 • 暂不支持查询消息轨迹。	mq-http-php-sdk-1.0.0.zip

3.7.2. 准备环境

在使用PHP SDK收发消息前，您需按照本文提供的内容来准备环境。

环境要求

- 安装PHP 5.5.0或以上版本，更多信息，请参见安装PHP。
- 安装Composer，更多信息，请参见安装Composer。

安装完成后，您可以执行 `php -v` 命令查看PHP语言版本。

安装SDK

执行以下步骤安装PHP SDK。

1. 在您PHP安装目录下的*composer.json*文件中加入以下依赖：

```
{
  "require": {
     "aliyunmq/mq-http-sdk": ">=1.0.3"
  }
}
```

2. 执行以下命令，通过Composer安装依赖。

```
composer install
```

3.7.3. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用HTTP协议下的PHP SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 安装PHP SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送普通消息

发送普通消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ProducerTest
{
    private $client;
    private $producer;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        $topic = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->producer = $this->client->getProducer($instanceId, $topic);
    }
    public function run()
    {
        try
        {
            for ($i=1; $i<=4; $i++)
            {
                $publishMessage = new TopicMessage(
                // 消息内容。
                "hello mq!"
                );
                // 设置消息的自定义属性。
                $publishMessage->putProperty("a", $i);
                // 设置消息的Key。
                $publishMessage->setMessageKey("MessageKey");
                $result = $this->producer->publishMessage($publishMessage);
                print "Send mq message success. msgId is:" . $result->getMessageId() . ", b
odyMD5 is:" . $result->getMessageBodyMD5() . "\n";
            }
        } catch (\Exception $e) {
            print_r($e->getMessage() . "\n");
        }
    }
}
$instance = new ProducerTest();
```

```
$instance->run();
?>
```

订阅普通消息

订阅普通消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ConsumerTest
{
    private $client;
    private $consumer;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        $topic = "${TOPIC}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        $groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId);
    }
    public function run()
    {
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
        while (True) {
            try {
                // 长轮询消费消息。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
                $messages = $this->consumer->consumeMessage(
                    3, // 一次最多消费3条（最多可设置为16条）。
                    3  // 长轮询时间3秒（最多可设置为30秒）。
                );
            } catch (\Exception $e) {
```

```
                if ($e instanceof MQ\Exception\MessageNotExistException) {
                    // Topic中没有消息可消费，继续轮询。
                    printf("No message, contine long polling!RequestId:%s\n", $e->getReques
tId());
                    continue;
                }
                print_r($e->getMessage() . "\n");
                sleep(3);
                continue;
            }
            print "consume finish, messages:\n";
            // 处理业务逻辑。
            $receiptHandles = array();
            foreach ($messages as $message) {
                $receiptHandles[] = $message->getReceiptHandle();
                printf("MessageID:%s TAG:%s BODY:%s \nPublishTime:%d, FirstConsumeTime:%d,
\nConsumedTimes:%d, NextConsumeTime:%d,MessageKey:%s\n",
                    $message->getMessageId(), $message->getMessageTag(), $message->getMessa
geBody(),
                    $message->getPublishTime(), $message->getFirstConsumeTime(), $message->
getConsumedTimes(), $message->getNextConsumeTime(),
                    $message->getMessageKey());
                print_r($message->getProperties());
            }
            // $message->getNextConsumeTime()前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            print_r($receiptHandles);
            try {
                $this->consumer->ackMessage($receiptHandles);
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\AckMessageException) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    printf("Ack Error, RequestId:%s\n", $e->getRequestId());
                    foreach ($e->getAckMessageErrorItems() as $errorItem) {
                        printf("\tReceiptHandle:%s, ErrorCode:%s, ErrorMsg:%s\n", $errorIte
m->getReceiptHandle(), $errorItem->getErrorCode(), $errorItem->getErrorCode());
                    }
                }
            }
            print "ack finish\n";
        }
    }
}
$instance = new ConsumerTest();
$instance->run();
?>
```

3.7.4. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用HTTP协议下的PHP SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息。

前提条件

您已完成以下操作：

- 安装PHP SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送顺序消息

> **注意**
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ProducerTest
{
    private $client;
    private $producer;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        $topic = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
```

```
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->producer = $this->client->getProducer($instanceId, $topic);
    }
    public function run()
    {
        try
        {
            for ($i=1; $i<=4; $i++)
            {
                $publishMessage = new TopicMessage(
                    "hello mq!"// 消息内容。
                );
                // 设置消息的自定义属性。
                $publishMessage->putProperty("a", $i);
                // 设置分区顺序消息的Sharding Key，用于标识不同的分区。Sharding Key与消息的Key是完全不同的概念。
                $publishMessage->setShardingKey($i % 2);
                $result = $this->producer->publishMessage($publishMessage);
                print "Send mq message success. msgId is:" . $result->getMessageId() . ", bodyMD5 is:" . $result->getMessageBodyMD5() . "\n";
            }
        } catch (\Exception $e) {
            print_r($e->getMessage() . "\n");
        }
    }
}
$instance = new ProducerTest();
$instance->run();
?>
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ConsumerTest
{
    private $client;
    private $consumer;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
```

```
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        $topic = "${TOPIC}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        $groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId);
    }
    public function run()
    {
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
        while (True) {
            try {
                // 长轮询顺序消费消息，拉取到的消息可能是多个分区的（对于分区顺序消息），一个分区内的消息一定是顺序的。
                // 对于分区顺序消息，只要一个分区内存在没有被确认消费的消息，那么该分区下次还会消费到相同的消息。
                // 对于一个分区，只有所有消息确认消费成功才能消费下一批消息。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则服务端立即返回响应。
                $messages = $this->consumer->consumeMessageOrderly(
                    3, // 一次最多消费3条（最多可设置为16条）。
                    3  // 长轮询时间3秒（最多可设置为30秒）。
                );
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\MessageNotExistException) {
                    // 没有消息，则继续长轮询服务器。
                    printf("No message, contine long polling!RequestId:%s\n", $e->getRequestId());
                    continue;
                }
                print_r($e->getMessage() . "\n");
                sleep(3);
                continue;
            }
            print "======>consume finish, messages:\n";
            // 处理业务逻辑。
            $receiptHandles = array();
            foreach ($messages as $message) {
                $receiptHandles[] = $message->getReceiptHandle();
                printf("MessageID:%s TAG:%s BODY:%s \nPublishTime:%d, FirstConsumeTime:%d, \nConsumedTimes:%d, NextConsumeTime:%d,ShardingKey:%s\n",
                    $message->getMessageId(), $message->getMessageTag(), $message->getMessageBody(),
                    $message->getPublishTime(), $message->getFirstConsumeTime(), $message->getConsumedTimes(), $message->getNextConsumeTime(),
```

```
                    $message->getShardingKey());
                print_r($message->getProperties());
            }
            // $message->getNextConsumeTime()前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            print_r($receiptHandles);
            try {
                $this->consumer->ackMessage($receiptHandles);
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\AckMessageException) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    printf("Ack Error, RequestId:%s\n", $e->getRequestId());
                    foreach ($e->getAckMessageErrorItems() as $errorItem) {
                        printf("\tReceiptHandle:%s, ErrorCode:%s, ErrorMsg:%s\n", $errorIte
m->getReceiptHandle(), $errorItem->getErrorCode(), $errorItem->getErrorCode());
                    }
                }
            }
            print "=======>ack finish\n";
        }
    }
}
$instance = new ConsumerTest();
$instance->run();
?>
```

3.7.5. 收发定时消息和延时消息

本文提供使用HTTP协议下的PHP SDK收发定时消息和延时消息的示例代码。

背景信息

- 延时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。
- 定时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

在HTTP协议下，定时消息和延时消息的代码配置相同，本质都是根据消息中的属性延迟固定时间后才投递给消费者。
更多信息，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 安装PHP SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ProducerTest
{
    private $client;
    private $producer;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        $topic = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->producer = $this->client->getProducer($instanceId, $topic);
    }
    public function run()
    {
        try
        {
            for ($i=1; $i<=4; $i++)
            {
                $publishMessage = new TopicMessage(
                    "hello mq!"// 消息内容。
                );
                // 设置消息的自定义属性。
                $publishMessage->putProperty("a", $i);
                // 设置消息的Key。
                $publishMessage->setMessageKey("MessageKey");
                // 延时消息，发送时间为10s后。该参数格式为毫秒级别的时间戳。
                // 若发送定时消息，设置该参数时需要计算定时时间与当前时间的时间差。
                $publishMessage->setStartDeliverTime(time() * 1000 + 10 * 1000);
                $result = $this->producer->publishMessage($publishMessage);
                print "Send mq message success. msgId is:" . $result->getMessageId() . ", b
odyMD5 is:" . $result->getMessageBodyMD5() . "\n";
            }
        } catch (\Exception $e) {
            print_r($e->getMessage() . "\n");
```

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

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ConsumerTest
{
    private $client;
    private $consumer;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
        $topic = "${TOPIC}";
        // 您在消息队列RocketMQ版控制台创建的Group ID。
        $groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId);
    }
    public function run()
    {
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
        while (True) {
            try {
                // 长轮询消费消息。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
                $messages = $this->consumer->consumeMessage(
                    3, // 一次最多消费3条（最多可设置为16条）。
```

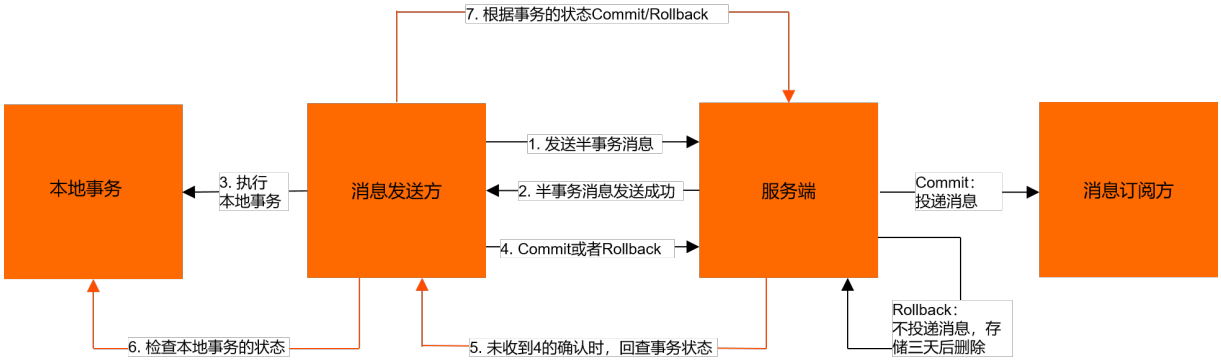
```
                    3  // 长轮询时间3秒（最多可设置为30秒）。
                );
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\MessageNotExistException) {
                    // Topic中没有消息可消费，继续轮询。
                    printf("No message, contine long polling!RequestId:%s\n", $e->getReques
tId());
                    continue;
                }
                print_r($e->getMessage() . "\n");
                sleep(3);
                continue;
            }
            print "consume finish, messages:\n";
            // 处理业务逻辑。
            $receiptHandles = array();
            foreach ($messages as $message) {
                $receiptHandles[] = $message->getReceiptHandle();
                printf("MessageID:%s TAG:%s BODY:%s \nPublishTime:%d, FirstConsumeTime:%d,
\nConsumedTimes:%d, NextConsumeTime:%d,MessageKey:%s\n",
                    $message->getMessageId(), $message->getMessageTag(), $message->getMessa
geBody(),
                    $message->getPublishTime(), $message->getFirstConsumeTime(), $message->
getConsumedTimes(), $message->getNextConsumeTime(),
                    $message->getMessageKey());
                print_r($message->getProperties());
            }
            // $message->getNextConsumeTime()前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            print_r($receiptHandles);
            try {
                $this->consumer->ackMessage($receiptHandles);
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\AckMessageException) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    printf("Ack Error, RequestId:%s\n", $e->getRequestId());
                    foreach ($e->getAckMessageErrorItems() as $errorItem) {
                        printf("\tReceiptHandle:%s, ErrorCode:%s, ErrorMsg:%s\n", $errorIte
m->getReceiptHandle(), $errorItem->getErrorCode(), $errorItem->getErrorCode());
                    }
                }
            }
            print "ack finish\n";
        }
    }
}
$instance = new ConsumerTest();
$instance->run();
?>
```

3.7.6. 收发事务消息

消息队列RocketMQ版
提供类似XA或Open XA的分布式事务功能，通过
消息队列RocketMQ版
事务消息，能达到分布式事务的最终一致。本文提供使用HTTP协议下的PHP SDK收发事务消息的示例代码。

背景信息

事务消息的交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 安装PHP SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送事务消息

发送事务消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ProducerTest
{
    private $client;
    private $transProducer;
    private $count;
    private $popMsgCount;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
```

```
        $topic = "${TOPIC}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        $groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->transProducer = $this->client->getTransProducer($instanceId,$topic, $groupId
);
        $this->count = 0;
        $this->popMsgCount = 0;
    }
    function processAckError($e) {
        if ($e instanceof MQ\Exception\AckMessageException) {
            // 如果Commit或Rollback时超过了TransCheckImmunityTime（针对发送事务消息的句柄）或者超过NextConsumeTime（针对consumeHalfMessage的句柄），则Commit或Rollback会失败。
            printf("Commit/Rollback Error, RequestId:%s\n", $e->getRequestId());
            foreach ($e->getAckMessageErrorItems() as $errorItem) {
                printf("\tReceiptHandle:%s, ErrorCode:%s, ErrorMsg:%s\n", $errorItem->getRe
ceiptHandle(), $errorItem->getErrorCode(), $errorItem->getErrorCode());
            }
        } else {
            print_r($e);
        }
    }
    function consumeHalfMsg() {
        while($this->count < 3 && $this->popMsgCount < 15) {
            $this->popMsgCount++;
            try {
                $messages = $this->transProducer->consumeHalfMessage(4, 3);
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\MessageNotExistException) {
                    print "no half transaction message\n";
                    continue;
                }
                print_r($e->getMessage() . "\n");
                sleep(3);
                continue;
            }
            foreach ($messages as $message) {
                printf("ID:%s TAG:%s BODY:%s \nPublishTime:%d, FirstConsumeTime:%d\nConsume
dTimes:%d, NextConsumeTime:%d\nPropA:%s\n",
                    $message->getMessageId(), $message->getMessageTag(), $message->getMessa
geBody(),
                    $message->getPublishTime(), $message->getFirstConsumeTime(), $message->
getConsumedTimes(), $message->getNextConsumeTime(),
                    $message->getProperty("a"));
                print_r($message->getProperties());
                $propA = $message->getProperty("a");
                $consumeTimes = $message->getConsumedTimes();
                try {
```

```
                    if ($propA == "1") {
                        print "\n commit transaction msg: " . $message->getMessageId() . "\
n";
                        $this->transProducer->commit($message->getReceiptHandle());
                        $this->count++;
                    } else if ($propA == "2" && $consumeTimes > 1) {
                        print "\n commit transaction msg: " . $message->getMessageId() . "\
n";
                        $this->transProducer->commit($message->getReceiptHandle());
                        $this->count++;
                    } else if ($propA == "3") {
                        print "\n rollback transaction msg: " . $message->getMessageId() .
"\n";
                        $this->transProducer->rollback($message->getReceiptHandle());
                        $this->count++;
                    } else {
                        print "\n unknown transaction msg: " . $message->getMessageId() . "
\n";
                    }
                } catch (\Exception $e) {
                    $this->processAckError($e);
                }
            }
        }
    }
    public function run()
    {
        // 循环发送4条事务消息。
        for ($i = 0; $i < 4; $i++) {
            $pubMsg = new TopicMessage("hello,mq");
            // 设置消息的自定义属性。
            $pubMsg->putProperty("a", $i);
            // 设置消息的Key。
            $pubMsg->setMessageKey("MessageKey");
            // 设置事务第一次回查的时间，为相对时间。单位：秒，范围：10~300。
            // 第一次事务回查后如果消息没有Commit或者Rollback，则之后每隔10s左右会回查一次，共回查2
4小时。
            $pubMsg->setTransCheckImmunityTime(10);
            $topicMessage = $this->transProducer->publishMessage($pubMsg);
            print "\npublish -> \n\t" . $topicMessage->getMessageId() . " " . $topicMessage
->getReceiptHandle() . "\n";
            if ($i == 0) {
                try {
                    // 发送完事务消息后能获取到半消息句柄，可以直接Commit或Rollback事务消息。
                    $this->transProducer->commit($topicMessage->getReceiptHandle());
                    print "\n commit transaction msg when publish: " . $topicMessage->getMe
ssageId() . "\n";
                } catch (\Exception $e) {
                    // 如果Commit或Rollback时超过了TransCheckImmunityTime则会失败。
                    $this->processAckError($e);
                }
            }
        }
        // 客户端需要有一个线程或者进程来消费没有确认的事务消息。
        // 检查没有确认的事务消息
```

```
        // 检查没有确认的事务消息。
        $this->consumeHalfMsg();
    }
}
$instance = new ProducerTest();
$instance->run();
?>
```

订阅事务消息

订阅事务消息的示例代码如下。

```
<?php
require "vendor/autoload.php";
use MQ\Model\TopicMessage;
use MQ\MQClient;
class ConsumerTest
{
    private $client;
    private $consumer;
    public function __construct()
    {
        $this->client = new MQClient(
            // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
        );
        // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
        $topic = "${TOPIC}";
        // 您在消息队列RocketMQ版控制台创建的Group ID。
        $groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
        $instanceId = "${INSTANCE_ID}";
        $this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId);
    }
    public function run()
    {
        // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
        while (True) {
            try {
                // 长轮询消费消息。
                // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
                $messages = $this->consumer->consumeMessage(
                    3, // 一次最多消费3条（最多可设置为16条）。
```

```
                    3  // 长轮询时间3秒（最多可设置为30秒）。
                );
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\MessageNotExistException) {
                    // Topic中没有消息可消费，继续轮询。
                    printf("No message, contine long polling!RequestId:%s\n", $e->getReques
tId());
                    continue;
                }
                print_r($e->getMessage() . "\n");
                sleep(3);
                continue;
            }
            print "consume finish, messages:\n";
            // 处理业务逻辑。
            $receiptHandles = array();
            foreach ($messages as $message) {
                $receiptHandles[] = $message->getReceiptHandle();
                printf("MessageID:%s TAG:%s BODY:%s \nPublishTime:%d, FirstConsumeTime:%d,
\nConsumedTimes:%d, NextConsumeTime:%d,MessageKey:%s\n",
                    $message->getMessageId(), $message->getMessageTag(), $message->getMessa
geBody(),
                    $message->getPublishTime(), $message->getFirstConsumeTime(), $message->
getConsumedTimes(), $message->getNextConsumeTime(),
                    $message->getMessageKey());
                print_r($message->getProperties());
            }
            // $message->getNextConsumeTime()前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            print_r($receiptHandles);
            try {
                $this->consumer->ackMessage($receiptHandles);
            } catch (\Exception $e) {
                if ($e instanceof MQ\Exception\AckMessageException) {
                    // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                    printf("Ack Error, RequestId:%s\n", $e->getRequestId());
                    foreach ($e->getAckMessageErrorItems() as $errorItem) {
                        printf("\tReceiptHandle:%s, ErrorCode:%s, ErrorMsg:%s\n", $errorIte
m->getReceiptHandle(), $errorItem->getErrorCode(), $errorItem->getErrorCode());
                    }
                }
            }
            print "ack finish\n";
        }
    }
}
$instance = new ConsumerTest();
$instance->run();
?>
```

3.8. C# SDK

3.8.1. 版本说明

本文介绍HTTP协议下C# SDK的版本信息，包括各版本的发布时间、发布内容以及下载链接，以便您按需获取适用的C# SDK收发消息。

1.0.5

发布时间	发布内容	下载
2022-07-04	**功能优化** • 修复当消息体出现非Unicode定义的特殊字符时导致的消费阻塞异常。	mq-http-csharp-sdk-1.0.5.zip

1.0.3

发布时间	发布内容	下载
2021-01-05	**新特性** 支持顺序消息。 • 支持顺序消息。	mq-http-csharp-sdk-1.0.3.zip

1.0.1

发布时间	发布内容	下载
2019-06-10	**新特性** • 支持高级特性消息，包括定时和延时消息、事务消息，暂不支持顺序消息。 • 支持使用Message ID查询消息轨迹。 • 支持消息属性。	mq-http-csharp-sdk-1.0.1.zip

1.0.0

发布时间	发布内容	下载
2019-01-23	**新特性** • 首次发布。 • 暂不支持高级特性消息，包括顺序消息、事务消息、定时和延时消息。 • 已支持集群消费模式，暂不支持广播消费模式。 • 暂不支持查询消息轨迹。	mq-http-csharp-sdk-1.0.0.zip

3.8.2. 准备环境

在使用C# SDK收发消息前，您需按照本文提供的内容来准备环境。

环境要求

- 安装.NET。更多信息，请参见安装.NET。
- 安装Visual Studio 2015或以上版本，更多信息，请参见安装Visual Studio。

安装完成后，您可以执行 `dotnet --version` 命令查看.NET Core版本。

安装C# SDK

执行以下操作安装C# SDK。

1. 下载C# SDK及工程文件到本地并解压。其中*Aliyun_MQ_SDK*为SDK所在目录，*Aliyun_MQ_SDK.sln*为工程文件。
2. 使用Visual Studio打开*Aliyun_MQ_SDK.sln*文件导入工程。
3. 执行*Samples.cs*文件。在Aliyun_MQ_SDK工程中可以看到*Samples.cs*文件，该文件中提供了C# SDK消息收发的示例代码，将代码中的示例值替换为您实际业务使用的参数值，然后保存并执行。

3.8.3. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用HTTP协议下的C# SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 安装C# SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送普通消息

发送普通消息的示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ.Util;
namespace Aliyun.MQ.Sample
{
    public class ProducerSample
    {
        // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实
```

```
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        private const string _instanceId = "${INSTANCE_ID}";
        private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccesssKey, _endpoint);
        static MQProducer producer = _client.GetProducer(_instanceId, _topicName);
        static void Main(string[] args)
        {
            try
            {
                // 循环发送4条消息。
                for (int i = 0; i < 4; i++)
                {
                TopicMessage sendMsg;
                // 消息内容。
                sendMsg = new TopicMessage("hello mq");
                // 设置消息的自定义属性。
                sendMsg.PutProperty("a", i.ToString());
                // 设置消息的Key。
                sendMsg.MessageKey = "MessageKey";
                TopicMessage result = producer.PublishMessage(sendMsg);
                Console.WriteLine("publis message success:" + result);
                }
            }
            catch (Exception ex)
            {
                Console.Write(ex);
            }
        }
    }
}
```

订阅普通消息

订阅普通消息的示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ;
namespace Aliyun.MQ.Sample
{
    public class ConsumerSample
    {
        // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在
```

```
消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实
例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        private const string _instanceId = "${INSTANCE_ID}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        private const string _groupId = "${GROUP_ID}";
        private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAcces
sKey, _endpoint);
        static MQConsumer consumer = _client.GetConsumer(_instanceId, _topicName, _groupId,
null);
        static void Main(string[] args)
        {
            // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
            while (true)
            {
                try
                {
                    // 长轮询消费消息。
                    // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则
立即返回客户端。
                    List<Message> messages = null;
                    try
                    {
                        messages = consumer.ConsumeMessage(
                            3, // 一次最多消费3条（最多可设置为16条）。
                            3  // 长轮询时间3秒（最多可设置为30秒）。
                        );
                    }
                    catch (Exception exp1)
                    {
                        if (exp1 is MessageNotExistException)
                        {
                            Console.WriteLine(Thread.CurrentThread.Name + " No new message,
" + ((MessageNotExistException)exp1).RequestId);
                            continue;
                        }
                        Console.WriteLine(exp1);
                        Thread.Sleep(2000);
                    }
                    if (messages == null)
                    {
                        continue;
                    }
                    List<string> handlers = new List<string>();
                    Console.WriteLine(Thread.CurrentThread.Name + " Receive Messages:");
                    // 处理业务逻辑。
                    foreach (Message message in messages)
                    {
                        Console.WriteLine(message);
```

```
                        Console.WriteLine("Property a is:" + message.GetProperty("a"));
                        handlers.Add(message.ReceiptHandle);
                    }
                    // Message.nextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    try
                    {
                        consumer.AckMessage(handlers);
                        Console.WriteLine("Ack message success:");
                        foreach (string handle in handlers)
                        {
                            Console.Write("\t" + handle);
                        }
                        Console.WriteLine();
                    }
                    catch (Exception exp2)
                    {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        if (exp2 is AckMessageException)
                        {
                            AckMessageException ackExp = (AckMessageException)exp2;
                            Console.WriteLine("Ack message fail, RequestId:" + ackExp.Reque
stId);
                            foreach (AckMessageErrorItem errorItem in ackExp.ErrorItems)
                            {
                                Console.WriteLine("\tErrorHandle:" + errorItem.ReceiptHandl
e + ",ErrorCode:" + errorItem.ErrorCode + ",ErrorMsg:" + errorItem.ErrorMessage);
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                    Thread.Sleep(2000);
                }
            }
        }
    }
}
```

3.8.4. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用HTTP协议下的C# SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。

- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息。

前提条件

您已完成以下操作：

- 安装C# SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送顺序消息

发送顺序消息的示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ.Util;
namespace Aliyun.MQ.Sample
{
    public class OrderProducerSample
    {
        // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        private const string _instanceId = "${INSTANCE_ID}";
        private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccesssKey, _endpoint);
        static MQProducer producer = _client.GetProducer(_instanceId, _topicName);
        static void Main(string[] args)
        {
            try
            {
                // 循环发送8条消息。
                for (int i = 0; i < 8; i++)
                {
                    // 消息内容和消息的tag。
                    TopicMessage sendMsg = new TopicMessage("hello mq", "tag");
                    // 设置消息的自定义属性
```

```
                    // 设置消息的自定义属性。
                    sendMsg.PutProperty("a", i.ToString());
                    // 设置分区顺序消息的Sharding Key，用于标识不同的分区。Sharding Key与消息的Key是完全不同的概念。
                    sendMsg.ShardingKey = (i % 2).ToString();
                    TopicMessage result = producer.PublishMessage(sendMsg);
                    Console.WriteLine("publis message success:" + result);
                }
            }
            catch (Exception ex)
            {
                Console.Write(ex);
            }
        }
    }
}
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ;
namespace Aliyun.MQ.Sample
{
    public class OrderConsumerSample
{
        // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // 您在消息队列RocketMQ版控制台创建的Group ID。
        private const string _groupId = "${GROUP_ID}";
        // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
private const string _instanceId = "${INSTANCE_ID}";
        private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccessKey, _endpoint);
        static MQConsumer consumer = _client.GetConsumer(_instanceId, _topicName, _groupId, null);
        static void Main(string[] args)
```

```
        {
            // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
            while (true)
            {
                try
                {
                    // 长轮询顺序消费消息，拉取到的消息可能是多个分区的（对于分区顺序消息），一个分区内的消息一定是顺序的。
                    // 对于分区顺序消息，只要一个分区内存在没有被确认消费的消息，那么该分区下次还会消费到相同的消息。
                    // 对于一个分区，只有所有消息确认消费成功才能消费下一批消息。
                    // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则服务端立即返回响应。
                    List<Message> messages = null;
                    try
                    {
                        messages = consumer.ConsumeMessageOrderly(
                            3, // 一次最多消费3条（最多可设置为16条）。
                            3 // 长轮询时间3秒（最多可设置为30秒）。
                        );
                    }
                    catch (Exception exp1)
                    {
                        if (exp1 is MessageNotExistException)
                        {
                            Console.WriteLine(Thread.CurrentThread.Name + " No new message, " + ((MessageNotExistException)exp1).RequestId);
                            continue;
                        }
                        Console.WriteLine(exp1);
                        Thread.Sleep(2000);
                    }
                    if (messages == null)
                    {
                        continue;
                    }
                    List<string> handlers = new List<string>();
                    Console.WriteLine(Thread.CurrentThread.Name + " Receive Messages:");
                    // 处理业务逻辑。
                    foreach (Message message in messages)
                    {
                        Console.WriteLine(message);
                        Console.WriteLine("Property a is:" + message.GetProperty("a"));
                        handlers.Add(message.ReceiptHandle);
                    }
                    // Message.nextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    try
                    {
                        consumer.AckMessage(handlers);
                        Console.WriteLine("Ack message success:");
                        foreach (string handle in handlers)
                        {
                            Console.Write("\t" + handle);
```

```
                        }
                        Console.WriteLine();
                    }
                    catch (Exception exp2)
                    {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        if (exp2 is AckMessageException)
                        {
                            AckMessageException ackExp = (AckMessageException)exp2;
                            Console.WriteLine("Ack message fail, RequestId:" + ackExp.Reque
stId);
                            foreach (AckMessageErrorItem errorItem in ackExp.ErrorItems)
                            {
                                Console.WriteLine("\tErrorHandle:" + errorItem.ReceiptHandl
e + ",ErrorCode:" + errorItem.ErrorCode + ",ErrorMsg:" + errorItem.ErrorMessage);
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                    Thread.Sleep(2000);
                }
            }
        }
    }
}
```

3.8.5. 收发定时消息和延时消息

本文提供使用HTTP协议下的C# SDK收发定时消息和延时消息的示例代码。

背景信息

- 延时消息：Producer将消息发送到
  消息队列RocketMQ版
  服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。
- 定时消息：Producer将消息发送到
  消息队列RocketMQ版
  服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

在HTTP协议下，定时消息和延时消息的代码配置相同，本质都是根据消息中的属性延迟固定时间后才投递给消费者。
更多信息，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 安装C# SDK。更多信息，请参见准备环境。

- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ.Util;
namespace Aliyun.MQ.Sample
{
    public class ProducerSample
    {
        // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        private const string _instanceId = "${INSTANCE_ID}";
        private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccesskey, _endpoint);
        static MQProducer producer = _client.GetProducer(_instanceId, _topicName);
        static void Main(string[] args)
        {
            try
            {
                // 循环发送4条消息。
                for (int i = 0; i < 4; i++)
                {
                    TopicMessage sendMsg;
                    // 消息内容。
                    sendMsg = new TopicMessage("hello mq");
                    // 设置消息的自定义属性。
                    sendMsg.PutProperty("a", i.ToString());
                    // 延时消息，发送时间为10s后。该参数格式为毫秒级别的时间戳。

                    // 若发送定时消息，设置该参数时需要计算定时时间与当前时间的时间差。
                    sendMsg.StartDeliverTime = AliyunSDKUtils.GetNowTimeStamp() + 10 * 1000;

                    TopicMessage result = producer.PublishMessage(sendMsg);
                    Console.WriteLine("publis message success:" + result);
```

```
                }
            }
            catch (Exception ex)
            {
                Console.Write(ex);
            }
        }
    }
}
```

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ;
namespace Aliyun.MQ.Sample
{
    public class ConsumerSample
    {
        // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        private const string _instanceId = "${INSTANCE_ID}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        private const string _groupId = "${GROUP_ID}";
        private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccessKey, _endpoint);
        static MQConsumer consumer = _client.GetConsumer(_instanceId, _topicName, _groupId, null);
        static void Main(string[] args)
        {
            // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
            while (true)
            {
                try
                {
                    // 长轮询消费消息
```

```
                    // 长轮询消费消息。
                    // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
                    List<Message> messages = null;
                    try
                    {
                        messages = consumer.ConsumeMessage(
                            3, // 一次最多消费3条（最多可设置为16条）。
                            3  // 长轮询时间3秒（最多可设置为30秒）。
                        );
                    }
                    catch (Exception exp1)
                    {
                        if (exp1 is MessageNotExistException)
                        {
                            Console.WriteLine(Thread.CurrentThread.Name + " No new message, " + ((MessageNotExistException)exp1).RequestId);
                            continue;
                        }
                        Console.WriteLine(exp1);
                        Thread.Sleep(2000);
                    }
                    if (messages == null)
                    {
                        continue;
                    }
                    List<string> handlers = new List<string>();
                    Console.WriteLine(Thread.CurrentThread.Name + " Receive Messages:");
                    // 处理业务逻辑。
                    foreach (Message message in messages)
                    {
                        Console.WriteLine(message);
                        Console.WriteLine("Property a is:" + message.GetProperty("a"));
                        handlers.Add(message.ReceiptHandle);
                    }
                    // Message.nextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    try
                    {
                        consumer.AckMessage(handlers);
                        Console.WriteLine("Ack message success:");
                        foreach (string handle in handlers)
                        {
                            Console.Write("\t" + handle);
                        }
                        Console.WriteLine();
                    }
                    catch (Exception exp2)
                    {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        if (exp2 is AckMessageException)
                        {
                            AckMessageException ackExp = (AckMessageException)exp2;
                            Console.WriteLine("Ack message fail, RequestId:" + ackExp.RequestId);
```

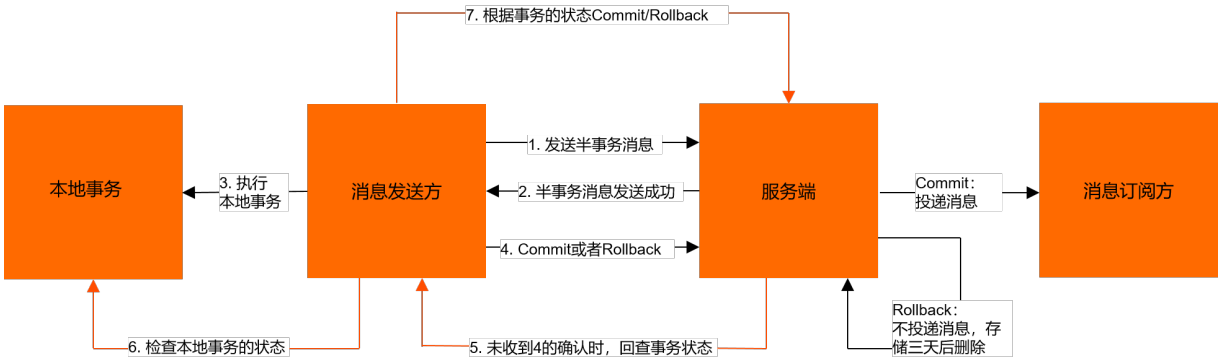
```
                            foreach (AckMessageErrorItem errorItem in ackExp.ErrorItems)
                            {
                                Console.WriteLine("\tErrorHandle:" + errorItem.ReceiptHandl
e + ",ErrorCode:" + errorItem.ErrorCode + ",ErrorMsg:" + errorItem.ErrorMessage);
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                    Thread.Sleep(2000);
                }
            }
        }
    }
}
```

3.8.6. 收发事务消息

消息队列RocketMQ版
提供类似XA或Open XA的分布式事务功能，通过
消息队列RocketMQ版
事务消息，能达到分布式事务的最终一致。本文提供使用HTTP协议下的C# SDK收发事务消息的示例代码。

背景信息

事务消息的交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 安装C# SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送事务消息

发送事务消息的示例代码如下。

```
using System;
using System.Collections.Generic;
```

```
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ.Util;
namespace Aliyun.MQ.Sample
{
    public class TransProducerSample
    {
        // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        private const string _instanceId = "${INSTANCE_ID}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        private const string _groupId = "${GROUP_ID}";
        private static readonly MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccessKey, _endpoint);
        private static readonly MQTransProducer transProducer = _client.GetTransProdcuer(_instanceId, _topicName, _groupId);
        static void ProcessAckError(Exception exception)
        {
            // 如果Commit或Rollback时超过了TransCheckImmunityTime（针对发送事务消息的句柄）或者超过10s（针对consumeHalfMessage的句柄），则Commit或Rollback会失败。
            if (exception is AckMessageException)
            {
                AckMessageException ackExp = (AckMessageException)exception;
                Console.WriteLine("Ack message fail, RequestId:" + ackExp.RequestId);
                foreach (AckMessageErrorItem errorItem in ackExp.ErrorItems)
                {
                    Console.WriteLine("\tErrorHandle:" + errorItem.ReceiptHandle + ",ErrorCode:" + errorItem.ErrorCode + ",ErrorMsg:" + errorItem.ErrorMessage);
                }
            }
        }
        static void ConsumeHalfMessage()
        {
            int count = 0;
            while (true)
            {
                if (count == 3)
                    break;
                try
```

```
                {
                    // 检查事务半消息，类似消费普通消息。
                    List<Message> messages = null;
                    try
                    {
                        messages = transProducer.ConsumeHalfMessage(3, 3);
                    } catch (Exception exp1) {
                        if (exp1 is MessageNotExistException)
                        {
                            Console.WriteLine(Thread.CurrentThread.Name + " No half message
, " + ((MessageNotExistException)exp1).RequestId);
                            continue;
                        }
                        Console.WriteLine(exp1);
                        Thread.Sleep(2000);
                    }
                    if (messages == null)
                        continue;
                    // 处理业务逻辑。
                    foreach (Message message in messages)
                    {
                        Console.WriteLine(message);
                        int a = int.Parse(message.GetProperty("a"));
                        uint consumeTimes = message.ConsumedTimes;
                        try {
                            if (a == 1) {
                                // 确认提交事务消息。
                                transProducer.Commit(message.ReceiptHandle);
                                count++;
                                Console.WriteLine("Id:" + message.Id + ", commit");
                            } else if (a == 2 && consumeTimes > 1) {
                                // 确认提交事务消息。
                                transProducer.Commit(message.ReceiptHandle);
                                count++;
                                Console.WriteLine("Id:" + message.Id + ", commit");
                            } else if (a == 3) {
                                // 确认回滚事务消息。
                                transProducer.Rollback(message.ReceiptHandle);
                                count++;
                                Console.WriteLine("Id:" + message.Id + ", rollback");
                            } else {
                                // 什么都不做，下次再检查。
                                Console.WriteLine("Id:" + message.Id + ", unkonwn");
                            }
                        } catch (Exception ackError) {
                            ProcessAckError(ackError);
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                    Thread.Sleep(2000);
                }
```

```
            }
        }
        static void Main(string[] args)
        {
            // 客户端需要有一个线程或者进程来消费没有确认的事务消息。
            // 启动一个线程来检查没有确认的事务消息。
            Thread consumeHalfThread = new Thread(ConsumeHalfMessage);
            consumeHalfThread.Start();
            try
            {
                // 循环发送4条事务消息，第一条直接在发送完提交事务，其它三条根据条件处理。
                for (int i = 0; i < 4; i++)
                {
                    TopicMessage sendMsg = new TopicMessage("trans_msg");
                    sendMsg.MessageTag = "a";
                    sendMsg.MessageKey = "MessageKey";
                    sendMsg.PutProperty("a", i.ToString());
                    // 设置事务第一次回查的时间，为相对时间。单位：秒，范围：10~300。
                    // 第一次事务回查后如果消息没有Commit或者Rollback，则之后每隔10s左右会回查一次，共回查24小时。
                    sendMsg.TransCheckImmunityTime = 10;
                    TopicMessage result = transProducer.PublishMessage(sendMsg);
                    Console.WriteLine("publis message success:" + result);
                    try {
                        if (!string.IsNullOrEmpty(result.ReceiptHandle) && i == 0)
                        {
                            // 发送完事务消息后能获取到半消息句柄，可以直接Commit或Rollback事务消息。
                            transProducer.Commit(result.ReceiptHandle);
                            Console.WriteLine("Id:" + result.Id + ", commit");
                        }
                    } catch (Exception ackError) {
                        ProcessAckError(ackError);
                    }
                }
            } catch (Exception ex) {
                Console.Write(ex);
            }
            consumeHalfThread.Join();
        }
    }
}
```

订阅事务消息

订阅事务消息的示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Threading;
using Aliyun.MQ.Model;
using Aliyun.MQ.Model.Exp;
using Aliyun.MQ;
namespace Aliyun.MQ.Sample
```

```
{
    public class ConsumerSample
    {
        // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
        private const string _endpoint = "${HTTP_ENDPOINT}";
        // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
        private const string _accessKeyId = "${ACCESS_KEY}";
        // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
        private const string _secretAccessKey = "${SECRET_KEY}";
        // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
        private const string _topicName = "${TOPIC}";
        // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
        // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
        private const string _instanceId = "${INSTANCE_ID}";
        // 您在
消息队列RocketMQ版控制台创建的Group ID。
        private const string _groupId = "${GROUP_ID}";
        private static MQClient _client = new Aliyun.MQ.MQClient(_accessKeyId, _secretAccesssKey, _endpoint);
        static MQConsumer consumer = _client.GetConsumer(_instanceId, _topicName, _groupId, null);
        static void Main(string[] args)
        {
            // 在当前线程循环消费消息，建议多开个几个线程并发消费消息。
            while (true)
            {
                try
                {
                    // 长轮询消费消息。
                    // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
                    List<Message> messages = null;
                    try
                    {
                        messages = consumer.ConsumeMessage(
                            3, // 一次最多消费3条（最多可设置为16条）。
                            3  // 长轮询时间3秒（最多可设置为30秒）。
                        );
                    }
                    catch (Exception exp1)
                    {
                        if (exp1 is MessageNotExistException)
                        {
                            Console.WriteLine(Thread.CurrentThread.Name + " No new message, " + ((MessageNotExistException)exp1).RequestId);
                            continue;
                        }
                        Console.WriteLine(exp1);
                        Thread.Sleep(2000);
                    }
```

```
                    }
                    if (messages == null)
                    {
                        continue;
                    }
                    List<string> handlers = new List<string>();
                    Console.WriteLine(Thread.CurrentThread.Name + " Receive Messages:");
                    // 处理业务逻辑。
                    foreach (Message message in messages)
                    {
                        Console.WriteLine(message);
                        Console.WriteLine("Property a is:" + message.GetProperty("a"));
                        handlers.Add(message.ReceiptHandle);
                    }
                    // Message.nextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
                    // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
                    try
                    {
                        consumer.AckMessage(handlers);
                        Console.WriteLine("Ack message success:");
                        foreach (string handle in handlers)
                        {
                            Console.Write("\t" + handle);
                        }
                        Console.WriteLine();
                    }
                    catch (Exception exp2)
                    {
                        // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                        if (exp2 is AckMessageException)
                        {
                            AckMessageException ackExp = (AckMessageException)exp2;
                            Console.WriteLine("Ack message fail, RequestId:" + ackExp.Reque
stId);
                            foreach (AckMessageErrorItem errorItem in ackExp.ErrorItems)
                            {
                                Console.WriteLine("\tErrorHandle:" + errorItem.ReceiptHandl
e + ",ErrorCode:" + errorItem.ErrorCode + ",ErrorMsg:" + errorItem.ErrorMessage);
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex);
                    Thread.Sleep(2000);
                }
            }
        }
    }
}
```

3.9. C++ SDK

3.9.1. 版本说明

本文介绍HTTP协议下C++ SDK的版本信息，包括各版本的发布时间、发布内容以及下载链接，以便您按需获取适用的C++ SDK收发消息。

1.0.3

发布时间	发布内容	下载
2021-01-05	**新特性** • 支持顺序消息。	mq-http-cpp-sdk-1.0.3.zip

1.0.1

发布时间	发布内容	下载
2019-06-10	**新特性** • 支持高级特性消息，包括定时和延时消息、事务消息，暂不支持顺序消息。 • 支持使用Message ID查询消息轨迹。 • 支持消息属性。	mq-http-cpp-sdk-1.0.1.zip

1.0.0

发布时间	发布内容	下载
2019-01-23	**新特性** • 首次发布。 • 暂不支持高级特性消息，包括顺序消息、事务消息、定时和延时消息。 • 已支持集群消费模式，暂不支持广播消费模式。 • 暂不支持查询消息轨迹。	mq-http-cpp-sdk-1.0.0.zip

3.9.2. 准备环境

在使用C++ SDK收发消息前，您需要按照本文提供的内容来准备环境。

环境要求

- 安装SCons。更多信息，请参见安装SCons。
- 使用SCons，需要先安装Python 3.5或以上版本。更多信息，请参见安装Python。
- 安装Visual Studio 2015或以上版本。更多信息，请参见安装Visual Studio。

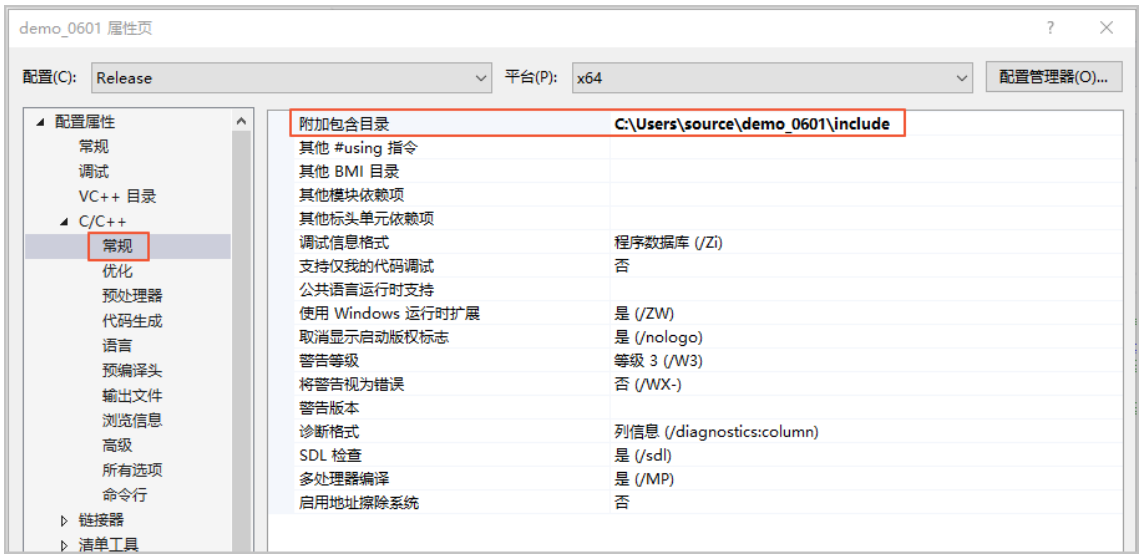
> **说明** 仅Windows系统需要安装Visual Studio，本文环境配置以Visual Studio 2019为例。

安装C++ SDK

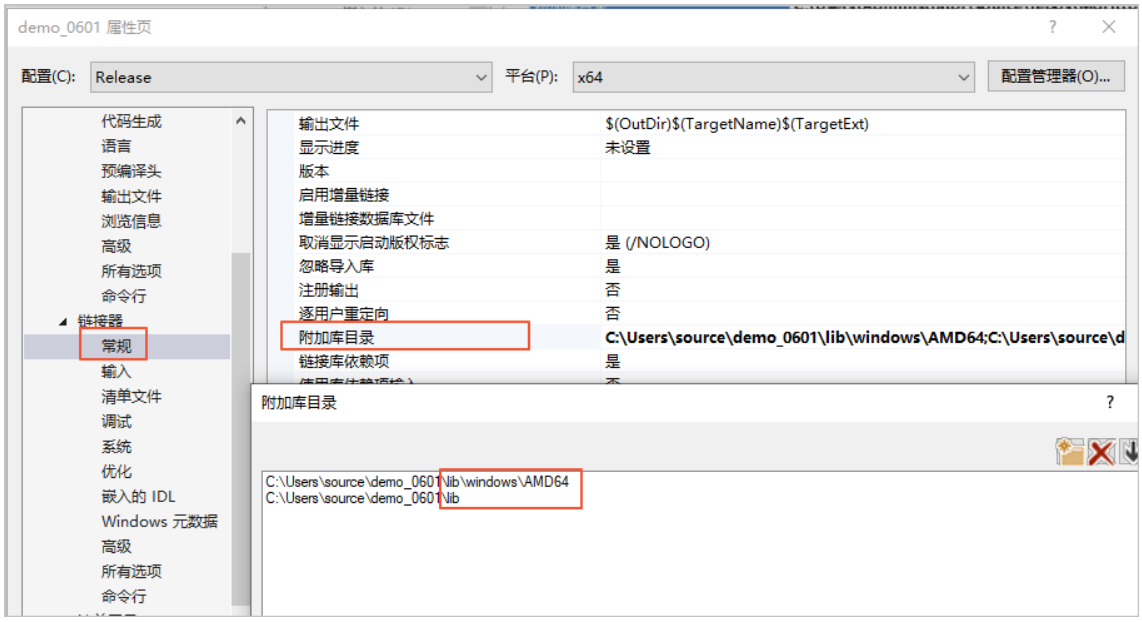
1. 下载SDK到本地并解压。SDK下载链接，请参见版本说明。
2. 在下载的SDK目录下，执行以下命令编译项目。

```
scons
```

3. 编译成功后，将SDK目录下的*include*和*lib*文件夹复制到您本地创建的C++项目目录下。
4. 在Visual Studio设置项目属性。右键单击项目，选择**属性**。
   - **设置附加包含目录**
     在项目属性页中，选择**配置属性 > C/C++ > 常规 > 附加包含目录**，将**附加包含目录**设置为步骤3中复制后的*include*文件夹路径。

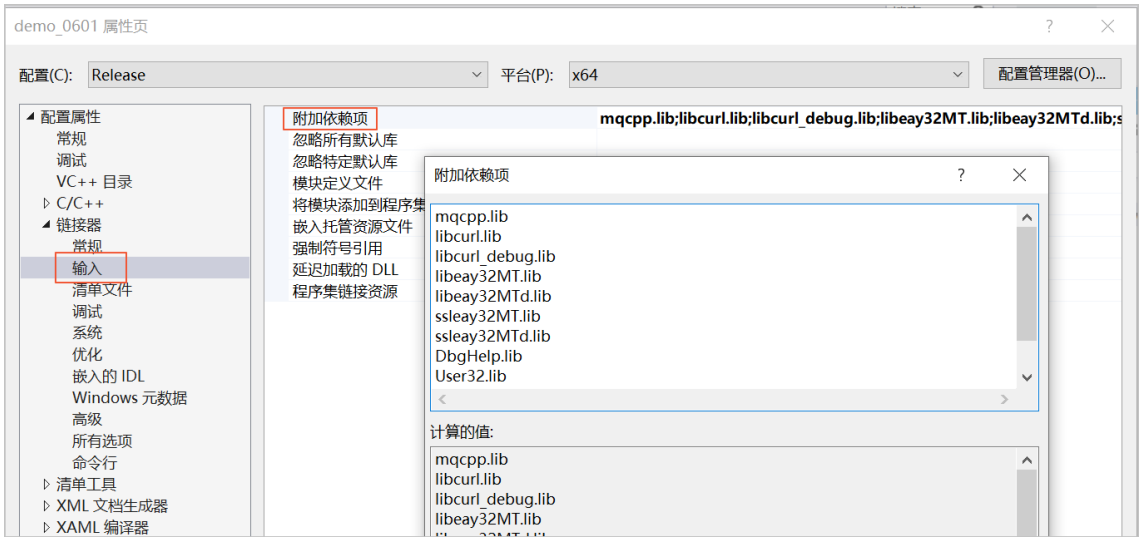


   - **设置附加库目录**
     在项目属性页中，选择**配置属性 > 链接器 > 常规 > 附加库目录**，将**附加库目录**设置为步骤3中复制后的*lib*文件夹路径，以及*lib\windows\{平台目录}*文件夹路径。其中，*{平台目录}*根据您使用的操作系统选择，64位操作系统选择*AMD64*，32位操作系统选择*I386*。



- **设置附加依赖项**
  在项目属性页中，选择**配置属性 > 链接器 > 输入 > 附加依赖项**，将以下内容添加到**附加依赖项**中。

```
mqcpp.lib
libcurl.lib
libcurl_debug.lib
libeay32MT.lib
libeay32MTd.lib
ssleay32MT.lib
ssleay32MTd.lib
DbgHelp.lib
User32.lib
GDI32.lib
Advapi32.lib
```



5. 将示例代码复制到项目文件中，并按照注释修改参数并保存。示例代码下载路径，请参见示例代码。
6. 单击按钮开始编译。

> **说明** 以下操作以CentOS系统为例。

1. 下载SDK到本地并解压。SDK下载链接，请参见版本说明。
2. 分别执行以下命令安装 `libcurl-devel` 和 `openssl-devel` 库。

```
yum install libcurl-devel
```

```
yum install openssl-devel
```

3. 在下载的SDK目录下，执行以下命令编译项目。

```
scons
```

4. 编译成功后，将SDK目录下的*include*和*lib*文件夹复制到您本地创建的C++项目目录下。
5. 将示例代码复制到您本地的项目文件中，并按照注释修改参数并保存。示例代码下载路径，请参见示例代码。

6. 执行以下命令进行编译。

```
# producer.cpp替换为您本地创建的项目文件名称。
g++ producer.cpp -o producer   lib/libmqcpp.a -I include/ -lcurl -lcrypto
```

Windows系统

Linux系统

3.9.3. 收发普通消息

普通消息是指
消息队列RocketMQ版
中无特性的消息，区别于有特性的定时和延时消息、顺序消息和事务消息。本文提供使用HTTP协议下的C++ SDK收发普通消息的示例代码。

前提条件

您已完成以下操作：

- 安装C++ SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送普通消息

发送普通消息的示例代码如下。

```
#include <fstream>
#include <time.h>
#include "mq_http_sdk/mq_client.h"
using namespace std;
using namespace mq::http::sdk;
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
            );
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    MQProducerPtr producer;
    if (instanceId == "") {
        producer = mqClient.getProducerRef(topic);
    } else {
```

```
        producer = mqClient.getProducerRef(instanceId, topic);
    }
    try {
        // 循环发送4条消息。
        for (int i = 0; i < 4; i++)
        {
            PublishMessageResponse pmResp;
            // 消息内容。
            TopicMessage pubMsg("Hello, mq!have key!");
            // 设置消息的自定义属性。
            pubMsg.putProperty("a",std::to_string(i));
            // 设置消息的Key。
            pubMsg.setMessageKey("MessageKey" + std::to_string(i));
            producer->publishMessage(pubMsg, pmResp);
            cout << "Publish mq message success. Topic is: " << topic
                << ", msgId is:" << pmResp.getMessageId()
                << ", bodyMD5 is:" << pmResp.getMessageBodyMD5() << endl;
        }
    } catch (MQServerException& me) {
        cout << "Request Failed: " + me.GetErrorCode() << ", requestId is:" << me.GetReques
tId() << endl;
        return -1;
    } catch (MQExceptionBase& mb) {
        cout << "Request Failed: " + mb.ToString() << endl;
        return -2;
    }
    return 0;
}
```

订阅普通消息

订阅普通消息的示例代码如下。

```
#include <vector>
#include <fstream>
#include "mq_http_sdk/mq_client.h"
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
using namespace std;
using namespace mq::http::sdk;
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
            );
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建
```

```
消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // 您在
消息队列RocketMQ版控制台创建的Group ID。
    string groupId = "${GROUP_ID}";
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    MQConsumerPtr consumer;
    if (instanceId == "") {
        consumer = mqClient.getConsumerRef(topic, groupId);
    } else {
        consumer = mqClient.getConsumerRef(instanceId, topic, groupId, "");
    }
    do {
        try {
            std::vector<Message> messages;
            // 长轮询消费消息。
            // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客
户端。
            consumer->consumeMessage(
                    3, // 一次最多消费3条（最多可设置为16条）。
                    3, // 长轮询时间3秒（最多可设置为30秒）。
                    messages
            );
            cout << "Consume: " << messages.size() << " Messages!" << endl;
            // 处理业务逻辑。
            std::vector<std::string> receiptHandles;
            for (std::vector<Message>::iterator iter = messages.begin();
                    iter != messages.end(); ++iter)
            {
                cout << "MessageId: " << iter->getMessageId()
                    << " PublishTime: " << iter->getPublishTime()
                    << " Tag: " << iter->getMessageTag()
                    << " Body: " << iter->getMessageBody()
                    << " FirstConsumeTime: " << iter->getFirstConsumeTime()
                    << " NextConsumeTime: " << iter->getNextConsumeTime()
                    << " ConsumedTimes: " << iter->getConsumedTimes()
                    << " Properties: " << iter->getPropertiesAsString()
                    << " Key: " << iter->getMessageKey() << endl;
                receiptHandles.push_back(iter->getReceiptHandle());
            }
            // 确认消息消费成功。
            // Message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            AckMessageResponse bdmResp;
            consumer->ackMessage(receiptHandles, bdmResp);
            if (!bdmResp.isSuccess()) {
                // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                const std::vector<AckMessageFailedItem>& failedItems =
                    bdmResp.getAckMessageFailedItem();
                for (std::vector<AckMessageFailedItem>::const_iterator iter = failedItems.b
```

```
egin();
                        iter != failedItems.end(); ++iter)
                {
                    cout << "AckFailedItem: " << iter->errorCode
                        << "  " << iter->receiptHandle << endl;
                }
            } else {
                cout << "Ack: " << messages.size() << " messages suc!" << endl;
            }
        } catch (MQServerException& me) {
            if (me.GetErrorCode() == "MessageNotExist") {
                cout << "No message to consume! RequestId: " + me.GetRequestId() << endl;
                continue;
            }
            cout << "Request Failed: " + me.GetErrorCode() + ".RequestId: " + me.GetRequest
Id() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
#endif
        } catch (MQExceptionBase& mb) {
            cout << "Request Failed: " + mb.ToString() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
#endif
        }
    } while(true);
}
```

3.9.4. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用HTTP协议下的C++ SDK收发顺序消息的示例代码。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息。

前提条件

您已完成以下操作：

- 安装C++ SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送顺序消息

> **注意**
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

```
//#include <iostream>
#include <fstream>
#include <time.h>
#include "mq_http_sdk/mq_client.h"
using namespace std;
using namespace mq::http::sdk;
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
            );
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    MQProducerPtr producer;
    if (instanceId == "") {
        producer = mqClient.getProducerRef(topic);
    } else {
        producer = mqClient.getProducerRef(instanceId, topic);
    }
    try {
        // 循环发送4条消息。
        for (int i = 0; i < 8; i++)
        {
            PublishMessageResponse pmResp;
            // 消息内容。
            TopicMessage pubMsg("Hello, mq!order msg!");
```

```
            // 设置分区顺序消息的Sharding Key，用于标识不同的分区。Sharding Key与消息的Key是完全不同的概念。
            pubMsg.setShardingKey(std::to_string(i % 2));
            // 设置消息的自定义属性。
            pubMsg.putProperty("a",std::to_string(i));
            producer->publishMessage(pubMsg, pmResp);
            cout << "Publish mq message success. Topic is: " << topic
                << ", msgId is:" << pmResp.getMessageId()
                << ", bodyMD5 is:" << pmResp.getMessageBodyMD5() << endl;
        }
    } catch (MQServerException& me) {
        cout << "Request Failed: " + me.GetErrorCode() << ", requestId is:" << me.GetRequestId() << endl;
        return -1;
    } catch (MQExceptionBase& mb) {
        cout << "Request Failed: " + mb.ToString() << endl;
        return -2;
    }
    return 0;
}
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
#include <vector>
#include <fstream>
#include "mq_http_sdk/mq_client.h"
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
using namespace std;
using namespace mq::http::sdk;
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
            );
    // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // 您在消息队列RocketMQ版控制台创建的Group ID。
    string groupId = "${GROUP_ID}";
    // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在
```

```
即名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    MQConsumerPtr consumer;
    if (instanceId == "") {
        consumer = mqClient.getConsumerRef(topic, groupId);
    } else {
        consumer = mqClient.getConsumerRef(instanceId, topic, groupId, "");
    }
    do {
        try {
            std::vector<Message> messages;
            // 长轮询顺序消费消息，拉取到的消息可能是多个分区的（对于分区顺序消息），一个分区内的消息一定是顺序的。
            // 对于分区顺序消息，只要一个分区内存在没有被确认消费的消息，那么该分区下次还会消费到相同的消息。
            // 对于一个分区，只有所有消息确认消费成功才能消费下一批消息。
            // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则服务端立即返回响应。
            consumer->consumeMessageOrderly(
                    3, // 一次最多消费3条（最多可设置为16条）。
                    3, // 长轮询时间3秒（最多可设置为30秒）。
                    messages
            );
            cout << "Consume: " << messages.size() << " Messages!" << endl;
            // 处理业务逻辑。
            std::vector<std::string> receiptHandles;
            for (std::vector<Message>::iterator iter = messages.begin();
                    iter != messages.end(); ++iter)
            {
                cout << "MessageId: " << iter->getMessageId()
                    << " PublishTime: " << iter->getPublishTime()
                    << " Tag: " << iter->getMessageTag()
                    << " Body: " << iter->getMessageBody()
                    << " FirstConsumeTime: " << iter->getFirstConsumeTime()
                    << " NextConsumeTime: " << iter->getNextConsumeTime()
                    << " ConsumedTimes: " << iter->getConsumedTimes()
                    << " Properties: " << iter->getPropertiesAsString()
                    << " ShardingKey: " << iter->getShardingKey() << endl;
                receiptHandles.push_back(iter->getReceiptHandle());
            }
            // 确认消息消费成功。
            // Message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            AckMessageResponse bdmResp;
            consumer->ackMessage(receiptHandles, bdmResp);
            if (!bdmResp.isSuccess()) {
                // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                const std::vector<AckMessageFailedItem>& failedItems =
                    bdmResp.getAckMessageFailedItem();
                for (std::vector<AckMessageFailedItem>::const_iterator iter = failedItems.begin();
                        iter != failedItems.end(); ++iter)
                {
                    cout << "AckFailedItem: " << iter->errorCode
```

```
                    << "  " << iter->receiptHandle << endl;
                }
            } else {
                cout << "Ack: " << messages.size() << " messages suc!" << endl;
            }
        } catch (MQServerException& me) {
            if (me.GetErrorCode() == "MessageNotExist") {
                cout << "No message to consume! RequestId: " + me.GetRequestId() << endl;
                continue;
            }
            cout << "Request Failed: " + me.GetErrorCode() + ".RequestId: " + me.GetRequest
Id() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
#endif
        } catch (MQExceptionBase& mb) {
            cout << "Request Failed: " + mb.ToString() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
#endif
        }
    } while(true);
}
```

3.9.5. 收发定时消息和延时消息

本文提供使用HTTP协议下的C++ SDK收发定时消息和延时消息的示例代码。

背景信息

- 延时消息：Producer将消息发送到
  消息队列RocketMQ版
  服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。
- 定时消息：Producer将消息发送到
  消息队列RocketMQ版
  服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

在HTTP协议下，定时消息和延时消息的代码配置相同，本质都是根据消息中的属性延迟固定时间后才投递给消费者。
更多信息，请参见定时和延时消息。

前提条件

您已完成以下操作：

- 安装C++ SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
#include <fstream>
#include <time.h>
#include "mq_http_sdk/mq_client.h"
using namespace std;
using namespace mq::http::sdk;
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
            );
    // 消息所属的Topic，在消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // Topic所属的实例ID，在消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的命名空间可以在消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    MQProducerPtr producer;
    if (instanceId == "") {
        producer = mqClient.getProducerRef(topic);
    } else {
        producer = mqClient.getProducerRef(instanceId, topic);
    }
    try {
        // 循环发送4条消息。
        for (int i = 0; i < 4; i++)
        {
            PublishMessageResponse pmResp;
            // 消息内容。
            TopicMessage pubMsg("Hello, mq!have key!");
            // 设置消息的自定义属性。
            pubMsg.putProperty("a",std::to_string(i));
            // 设置消息的Key。
            pubMsg.setMessageKey("MessageKey" + std::to_string(i));
            // 延时消息，发送时间为10s后。该参数格式为毫秒级别的时间戳。
            // 若发送定时消息，设置该参数时需要计算定时时间与当前时间的时间差。
            pubMsg.setStartDeliverTime(time(NULL) * 1000 + 10 * 1000);
            producer->publishMessage(pubMsg, pmResp);
            cout << "Publish mq message success. Topic is: " << topic
                << ", msgId is:" << pmResp.getMessageId()
                << ", bodyMD5 is:" << pmResp.getMessageBodyMD5() << endl;
        }
    } catch (MQServerException& me) {
        cout << "Request Failed: " + me.GetErrorCode() << ", requestId is:" << me.GetReques
```

```
tId() << endl;
        return -1;
    } catch (MQExceptionBase& mb) {
        cout << "Request Failed: " + mb.ToString() << endl;
        return -2;
    }
    return 0;
}
```

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

```
#include <vector>
#include <fstream>
#include "mq_http_sdk/mq_client.h"
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
using namespace std;
using namespace mq::http::sdk;
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
            );
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // 您在
消息队列RocketMQ版控制台创建的Group ID。
    string groupId = "${GROUP_ID}";
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    MQConsumerPtr consumer;
    if (instanceId == "") {
        consumer = mqClient.getConsumerRef(topic, groupId);
    } else {
        consumer = mqClient.getConsumerRef(instanceId, topic, groupId, "");
    }
    do {
        try {
            std::vector<Message> messages;
```

```
            // 长轮询消费消息。
            // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
            consumer->consumeMessage(
                    3, // 一次最多消费3条（最多可设置为16条）。
                    3, // 长轮询时间3秒（最多可设置为30秒）。
                    messages
            );
            cout << "Consume: " << messages.size() << " Messages!" << endl;
            // 处理业务逻辑。
            std::vector<std::string> receiptHandles;
            for (std::vector<Message>::iterator iter = messages.begin();
                    iter != messages.end(); ++iter)
            {
                cout << "MessageId: " << iter->getMessageId()
                    << " PublishTime: " << iter->getPublishTime()
                    << " Tag: " << iter->getMessageTag()
                    << " Body: " << iter->getMessageBody()
                    << " FirstConsumeTime: " << iter->getFirstConsumeTime()
                    << " NextConsumeTime: " << iter->getNextConsumeTime()
                    << " ConsumedTimes: " << iter->getConsumedTimes()
                    << " Properties: " << iter->getPropertiesAsString()
                    << " Key: " << iter->getMessageKey() << endl;
                receiptHandles.push_back(iter->getReceiptHandle());
            }
            // 确认消息消费成功。
            // Message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            AckMessageResponse bdmResp;
            consumer->ackMessage(receiptHandles, bdmResp);
            if (!bdmResp.isSuccess()) {
                // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                const std::vector<AckMessageFailedItem>& failedItems =
                    bdmResp.getAckMessageFailedItem();
                for (std::vector<AckMessageFailedItem>::const_iterator iter = failedItems.begin();
                        iter != failedItems.end(); ++iter)
                {
                    cout << "AckFailedItem: " << iter->errorCode
                        << "  " << iter->receiptHandle << endl;
                }
            } else {
                cout << "Ack: " << messages.size() << " messages suc!" << endl;
            }
        } catch (MQServerException& me) {
            if (me.GetErrorCode() == "MessageNotExist") {
                cout << "No message to consume! RequestId: " + me.GetRequestId() << endl;
                continue;
            }
            cout << "Request Failed: " + me.GetErrorCode() + ".RequestId: " + me.GetRequestId() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
```

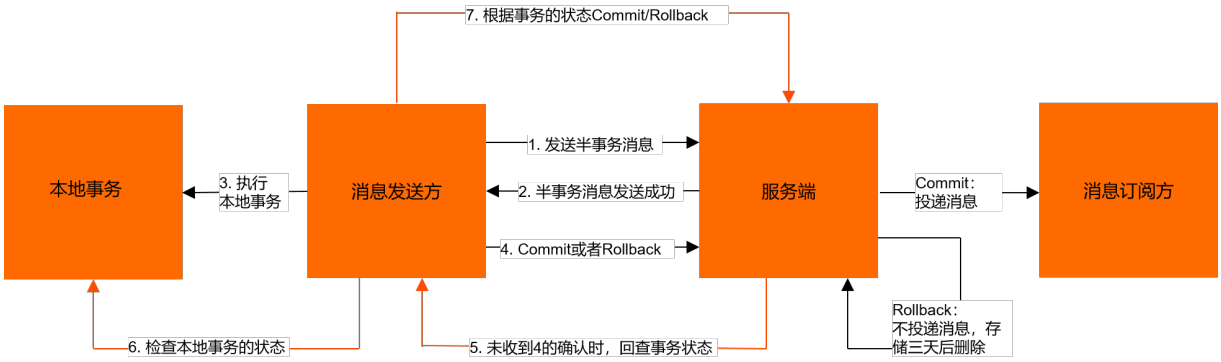
```
            usleep(2000 * 1000);
#endif
        } catch (MQExceptionBase& mb) {
            cout << "Request Failed: " + mb.ToString() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
#endif
        }
    } while(true);
}
```

3.9.6. 收发事务消息

消息队列RocketMQ版
提供类似XA或Open XA的分布式事务功能，通过
消息队列RocketMQ版
事务消息，能达到分布式事务的最终一致。本文提供使用HTTP协议下的C++ SDK收发事务消息的示例代码。

背景信息

事务消息的交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 安装C++ SDK。更多信息，请参见准备环境。
- 创建资源。代码中涉及的资源信息，例如实例、Topic和Group ID等，需要在控制台上提前创建。更多信息，请参见创建资源。

发送事务消息

发送事务消息的示例代码如下。

```
//#include <iostream>
#include <fstream>
#ifdef _WIN32
#include <windows.h>
#include <process.h>
#else
#include "pthread.h"
#endif
```

```
#endif
#include "mq_http_sdk/mq_client.h"
using namespace std;
using namespace mq::http::sdk;
const int32_t pubMsgCount = 4;
const int32_t halfCheckCount = 3;
void processCommitRollError(AckMessageResponse& bdmResp, const std::string& messageId) {
    if (bdmResp.isSuccess()) {
        cout << "Commit/Roll Transaction Suc: " << messageId << endl;
        return;
    }
    const std::vector<AckMessageFailedItem>& failedItems =
        bdmResp.getAckMessageFailedItem();
    for (std::vector<AckMessageFailedItem>::const_iterator iter = failedItems.begin();
            iter != failedItems.end(); ++iter)
    {
        cout << "Commit/Roll Transaction ERROR: " << iter->errorCode
            << "  " << iter->receiptHandle << endl;
    }
}
#ifdef WIN32
unsigned __stdcall consumeHalfMessageThread(void *arg)
#else
void* consumeHalfMessageThread(void *arg)
#endif
{
    MQTransProducerPtr transProducer = *(MQTransProducerPtr*)(arg);
    int count = 0;
    do {
        std::vector<Message> halfMsgs;
        try {
            // 长轮询消费消息。
            // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
            transProducer->consumeHalfMessage(
                    1, // 一次最多消费1条（最多可设置为16条）。
                    3, // 长轮询时间3秒（最多可设置为30秒）。
                    halfMsgs
                    );
        } catch (MQServerException& me) {
            if (me.GetErrorCode() == "MessageNotExist") {
                cout << "No half message to consume! RequestId: " + me.GetRequestId() << endl;
                continue;
            }
            cout << "Request Failed: " + me.GetErrorCode() + ".RequestId: " + me.GetRequestId() << endl;
        }
        if (halfMsgs.size() == 0) {
            continue;
        }
        cout << "Consume Half: " << halfMsgs.size() << " Messages!" << endl;
        // 处理事务半消息。
        std::vector<std::string> receiptHandles;
        for (std::vector<Message>::iterator iter = halfMsgs.begin();
```

```
            iter != halfMsgs.end(); ++iter)
        {
            cout << "MessageId: " << iter->getMessageId()
                << " PublishTime: " << iter->getPublishTime()
                << " Tag: " << iter->getMessageTag()
                << " Body: " << iter->getMessageBody()
                << " FirstConsumeTime: " << iter->getFirstConsumeTime()
                << " NextConsumeTime: " << iter->getNextConsumeTime()
                << " ConsumedTimes: " << iter->getConsumedTimes()
                << " Properties: " << iter->getPropertiesAsString()
                << " Key: " << iter->getMessageKey() << endl;
            int32_t consumedTimes = iter->getConsumedTimes();
            const std::string propA = iter->getProperty("a");
            const std::string handle = iter->getReceiptHandle();
            AckMessageResponse bdmResp;
            if (propA == "1") {
                cout << "Commit msg.." << endl;
                transProducer->commit(handle, bdmResp);
                count++;
            } else if(propA == "2") {
                if (consumedTimes > 1) {
                    cout << "Commit msg.." << endl;
                    transProducer->commit(handle, bdmResp);
                    count++;
                } else {
                    cout << "Commit Later!!!" << endl;
                }
            } else if(propA == "3") {
                cout << "Rollback msg.." << endl;
                transProducer->rollback(handle, bdmResp);
                count++;
            } else {
                transProducer->commit(handle, bdmResp);
                cout << "Unkown msg.." << endl;
            }
            // 如果Commit或Rollback时超过了NextConsumeTime的时间，则Commit或Rollback会失败。
            processCommitRollError(bdmResp, iter->getMessageId());
        }
    } while(count < halfCheckCount);
#ifdef WIN32
    return 0;
#else
    return NULL;
#endif
}
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
```

```
            );
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    // 您在
消息队列RocketMQ版控制台创建的Group ID。
    string groupId = "${GROUP_ID}";
    MQTransProducerPtr transProducer;
    if (instanceId == "") {
        transProducer = mqClient.getTransProducerRef(topic, groupId);
    } else {
        transProducer = mqClient.getTransProducerRef(instanceId, topic, groupId);
    }
    // 客户端需要有一个线程或者进程来消费没有确认的事务消息。
    // 启动一个线程来检查没有确认的事务消息。
#ifdef WIN32
    HANDLE thread;
    unsigned int threadId;
    thread = (HANDLE)_beginthreadex(NULL, 0, consumeHalfMessageThread, &transProducer, 0, &
threadId);
#else
    pthread_t thread;
    pthread_create(&thread, NULL, consumeHalfMessageThread, static_cast<void *>(&transProdu
cer));
#endif
    try {
        for (int i = 0; i < pubMsgCount; i++)
        {
            PublishMessageResponse pmResp;
            TopicMessage pubMsg("Hello, mq, trans_msg!");
            pubMsg.putProperty("a",std::to_string(i));
            pubMsg.setMessageKey("ImKey");
            pubMsg.setTransCheckImmunityTime(10);
            transProducer->publishMessage(pubMsg, pmResp);
            cout << "Publish mq message success. Topic:" << topic
                << ", msgId:" << pmResp.getMessageId()
                << ", bodyMD5:" << pmResp.getMessageBodyMD5()
                << ", Handle:" << pmResp.getReceiptHandle() << endl;
            if (i == 0) {
                // 发送完事务消息后能获取到半消息句柄，可以直接Commit或Rollback事务消息。
                // 如果Commit或Rollback时超过了TransCheckImmunityTime的时间，则Commit或Rollbac
k会失败。
                AckMessageResponse bdmResp;
                transProducer->commit(pmResp.getReceiptHandle(), bdmResp);
                processCommitRollError(bdmResp, pmResp.getMessageId());
            }
        }
    } catch (MQServerException& me) {
```

```
        cout << "Request Failed: " + me.GetErrorCode() << ", requestId is:" << me.GetReques
tId() << endl;
    } catch (MQExceptionBase& mb) {
        cout << "Request Failed: " + mb.ToString() << endl;
    }
#ifdef WIN32
    WaitForSingleObject(thread, INFINITE);
    CloseHandle(thread);
#else
    pthread_join(thread, NULL);
#endif
    return 0;
}
```

订阅事务消息

订阅事务消息的示例代码如下。

```
#include <vector>
#include <fstream>
#include "mq_http_sdk/mq_client.h"
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
using namespace std;
using namespace mq::http::sdk;
int main() {
    MQClient mqClient(
            // 设置HTTP协议客户端接入点，进入
消息队列RocketMQ版控制台实例详情页面的接入点区域查看。
            "${HTTP_ENDPOINT}",
            // AccessKey ID阿里云身份验证，在阿里云RAM控制台创建。
            "${ACCESS_KEY}",
            // AccessKey Secret阿里云身份验证，在阿里云RAM控制台创建。
            "${SECRET_KEY}"
            );
    // 消息所属的Topic，在
消息队列RocketMQ版控制台创建。
    string topic = "${TOPIC}";
    // 您在
消息队列RocketMQ版控制台创建的Group ID。
    string groupId = "${GROUP_ID}";
    // Topic所属的实例ID，在
消息队列RocketMQ版控制台创建。
    // 若实例有命名空间，则实例ID必须传入；若实例无命名空间，则实例ID传入null空值或字符串空值。实例的
命名空间可以在
消息队列RocketMQ版控制台的实例详情页面查看。
    string instanceId = "${INSTANCE_ID}";
    MQConsumerPtr consumer;
    if (instanceId == "") {
        consumer = mqClient.getConsumerRef(topic, groupId);
    } else {
```

```
        consumer = mqClient.getConsumerRef(instanceId, topic, groupId, "");
    }
    do {
        try {
            std::vector<Message> messages;
            // 长轮询消费消息。
            // 长轮询表示如果Topic没有消息，则请求会在服务端挂起3s，3s内如果有消息可以消费则立即返回客户端。
            consumer->consumeMessage(
                    3, // 一次最多消费3条（最多可设置为16条）。
                    3, // 长轮询时间3秒（最多可设置为30秒）。
                    messages
            );
            cout << "Consume: " << messages.size() << " Messages!" << endl;
            // 处理业务逻辑。
            std::vector<std::string> receiptHandles;
            for (std::vector<Message>::iterator iter = messages.begin();
                    iter != messages.end(); ++iter)
            {
                cout << "MessageId: " << iter->getMessageId()
                    << " PublishTime: " << iter->getPublishTime()
                    << " Tag: " << iter->getMessageTag()
                    << " Body: " << iter->getMessageBody()
                    << " FirstConsumeTime: " << iter->getFirstConsumeTime()
                    << " NextConsumeTime: " << iter->getNextConsumeTime()
                    << " ConsumedTimes: " << iter->getConsumedTimes()
                    << " Properties: " << iter->getPropertiesAsString()
                    << " Key: " << iter->getMessageKey() << endl;
                receiptHandles.push_back(iter->getReceiptHandle());
            }
            // 确认消息消费成功。
            // Message.NextConsumeTime前若不确认消息消费成功，则消息会被重复消费。
            // 消息句柄有时间戳，同一条消息每次消费拿到的都不一样。
            AckMessageResponse bdmResp;
            consumer->ackMessage(receiptHandles, bdmResp);
            if (!bdmResp.isSuccess()) {
                // 某些消息的句柄可能超时，会导致消息消费状态确认不成功。
                const std::vector<AckMessageFailedItem>& failedItems =
                    bdmResp.getAckMessageFailedItem();
                for (std::vector<AckMessageFailedItem>::const_iterator iter = failedItems.begin();
                        iter != failedItems.end(); ++iter)
                {
                    cout << "AckFailedItem: " << iter->errorCode
                        << "  " << iter->receiptHandle << endl;
                }
            } else {
                cout << "Ack: " << messages.size() << " messages suc!" << endl;
            }
        } catch (MQServerException& me) {
            if (me.GetErrorCode() == "MessageNotExist") {
                cout << "No message to consume! RequestId: " + me.GetRequestId() << endl;
                continue;
            }
            cout << "Request Failed: " + me.GetErrorCode() + " RequestId: " + me.GetRequest
```

```
Id() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
#endif
        } catch (MQExceptionBase& mb) {
            cout << "Request Failed: " + mb.ToString() << endl;
#ifdef _WIN32
            Sleep(2000);
#else
            usleep(2000 * 1000);
#endif
        }
    } while(true);
}
```

3.10. HTTP SDK错误码列表

本文介绍使用
消息队列RocketMQ版
提供的商业版HTTP协议的SDK收发消息时可能遇到的错误码，以及建议的处理方式。

错误码	错误描述	HTTP状态码	建议处理方式
AccessDenied	The OwnerId that your Access Key Id associated to is forbidden for this operation.	403	权限不足，请检查AccessKey ID和AccessKey Secret等配置是否正确。
InvalidAccessKeyId	The AccessKey Id you provided is not exist.	403	AccessKey ID不存在，请检查AccessKey ID是否正确。
InternalError	Interal error.	500	系统错误，请提交工单联系技术支持人员处理。
InternalServerError	Interal error.	500	系统错误，请提交工单联系技术支持人员处理。
InvalidAuthorizationHeader	The Authorization header format is invalid.	400	签名格式不正确。详细信息，请参见签名机制。
InvalidDateHeader	The Date header format is invalid.	400	Date字段不合法。详细信息，请参见公共参数。

错误码	错误描述	HTTP状态码	建议处理方式
InvalidArgument	The XML you provided did not validate against our published schema, cause by Element①.	400	XML结构不正确。正确的XML结构，请参见对应的API文档： • 发送消息API • 订阅消息API • 确认消息API
InvalidArgument	The value of Element②should between Low③ and High④ seconds/bytes.	400	参数值非法，请根据提示调整。
InvalidDigest	The Content-MD5 you specified is invalid.	400	请求Header中Content-MD5不正确。请传递正确的取值或将该值置空。
InvalidRequestURL	Http request URL format invalid.	400	请求的URL不正确。正确的URL，请参见对应的API文档： • 发送消息API • 订阅消息API • 确认消息API
InvalidQueryString	Http request URL contains invalid querystring item “Element⑤”.	400	请求的URL不正确。正确的URL，请参见对应的API文档： • 发送消息API • 订阅消息API • 确认消息API
MalformedXML	The XML you provided was not well-formed.	400	XML结构异常。正确的XML结构，请参见对应的API文档： • 发送消息API • 订阅消息API • 确认消息API
MissingAuthorizationHeader	Authorization header is required.	400	请求Header缺少字段，请参见公共参数。
MissingDateHeader	Date header is required.	400	请求Header缺少字段，请参见公共参数。

错误码	错误描述	HTTP状态码	建议处理方式
MissingReceiptHandle	ReceiptHandle is required.	400	请求缺少参数。详细信息，请参见对应的API文档： • 发送消息API • 订阅消息API • 确认消息API
MessageNotExist	Message not exist.	404	Topic中无可用的消息。
	The receipt handle you provided has expired.	404	消费消息过慢导致消息重新回到队列生成新的ReceiptHandle，之前的ReceiptHandle失效。
ReceiptHandleError	The receipt handle you provide is not valid.	400	请求参数不合法。详细信息，请参见对应的API文档： • 发送消息API • 订阅消息API • 确认消息API
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check your key and signing method.	403	请求签名异常。详细信息，请参见签名机制。
TimeExpired	The http request you sent is expired.	408	请求时间与消息队列RocketMQ版HTTP服务器时间相差超过15分钟，建议检查本地时间。
QpsLimitExceeded	The qps limit is exceeded ⑥.	400	Topic每秒的请求次数超过QPS限制。如果需要提高QPS限制，请提交工单联系技术支持人员处理。
HTTPForbidden	Consume message through HTTP and TCP at the same time is forbidden.	400	同一个Group ID不能同时被TCP和HTTP使用。

错误码	错误描述	HTTP状态码	建议处理方式
InvalidArgument	The length of message should not be larger than MaximumMessageSize.	400	消息体过长。详细信息，请参见对应的API文档： • 发送消息API • 订阅消息API • 确认消息API

说明

- ①②：在请求内容的XML元素名称。
- ③：在消息队列RocketMQ版中某个参数的下限值。
- ④：在消息队列RocketMQ版中某个参数的上限值。
- ⑤：在URL请求中QueryString的元素。
- ⑥：当前消息队列RocketMQ版单个Topic的QPS上限。具体限制，请参见使用限制。

4.社区版TCP协议SDK（仅供开源用户上云使用）

4.1. 概述

您可获取社区版TCP协议的SDK，来访问阿里云
消息队列RocketMQ版
。

> **注意** 所有社区版的SDK均由Apache RocketMQ社区提供，您可获取源码自行编译，但不在阿里云RocketMQ的SLA范围。
> 和社区版SDK相比，商业版的SDK提供了更加丰富的功能特性并具有更高的稳定性保障，推荐您使用商业版SDK访问阿里云
> 消息队列RocketMQ版
> 。更多信息，请参见商业版TCP协议SDK。

适用场景

使用社区版TCP协议的SDK访问阿里云

消息队列RocketMQ版
适用于以下场景：

- 云迁移场景：从开源RocketMQ迁移到阿里云
  消息队列RocketMQ版
  上，且不希望修改客户端的代码。
- 混合云场景：您既有部署在IDC的开源RocketMQ，也有部署在阿里云公有云上的
  消息队列RocketMQ版
  ，且需要同时访问两种部署模式下的消息队列服务端。
- 您有两套环境，测试环境和线上环境；其中测试环境部署的是开源RocketMQ，线上环境部署的是阿里云
  消息队列RocketMQ版
  ，您需同时访问部署在两套环境中的消息队列服务端。

访问步骤

针对不同的语言，操作有所不同。请按需参见以下文档实现社区版SDK访问阿里云

消息队列RocketMQ版
：

- 社区版TCP协议Java SDK接入说明
- 社区版TCP协议C++ SDK接入说明

暂不支持通过社区版TCP协议的Go SDK和Python SDK访问阿里云
消息队列RocketMQ版
，推荐您使用商业版HTTP协议的SDK接入。更多信息，请参见：

- 商业版HTTP协议Go SDK接入说明
- 商业版HTTP协议Python SDK接入说明

4.2. 社区版Java SDK接入说明

4.2.1. 概述

本文介绍使用社区版TCP协议的Java SDK访问阿里云
消息队列RocketMQ版
来收发消息的流程。

使用说明

社区版SDK仅在开源RocketMQ迁移上云且不希望修改代码时使用，其他场景推荐您使用阿里云

消息队列RocketMQ版
提供的商业版SDK进行接入。和社区版SDK相比，商业版SDK提供了更加丰富的功能特性并具有更高的稳定性保障。更多信息，请参见商业版Java SDK。

版本说明

使用社区版TCP协议的Java SDK访问阿里云

消息队列RocketMQ版
，您需要下载4.5.2以上版本，推荐您优先使用最新版本的SDK。下载地址：社区版TCP协议Java SDK。

准备工作

请按顺序执行以下操作：

1. 引入Maven依赖。
2. 在阿里云
   消息队列RocketMQ版
   控制台创建资源。

更多信息，请参见准备工作。

（可选）参数说明

在使用社区版TCP协议的Java SDK接入阿里云

消息队列RocketMQ版
收发消息时，需要配置相应的参数。更多信息，请参见参数说明。
您也可以参见社区版TCP协议的Java SDK中的注释了解参数说明。

使用社区版TCP协议的Java SDK收发消息

- 收发普通消息（三种方式）
- 收发顺序消息
- 收发定时消息和延时消息
- 收发事务消息

4.2.2. 参数说明

本文介绍您在使用社区版Java SDK接入阿里云
消息队列RocketMQ版
时，需要配置的参数。

通用参数

参数名	参数说明
NAMESRV_ADDR	设置TCP协议接入点，从消息队列RocketMQ版控制台的**实例详情**页面获取。
AccessKey	您在阿里云账号管理控制台中创建的AccessKey ID，用于身份认证。
SecretKey	您在阿里云账号管理控制台中创建的AccessKey Secret，用于身份认证。
AccessChannel	用于指定使用云上消息轨迹，上云设置为：AccessChannel:CLOUD。

消息发送参数

参数名	参数说明
producerGroup	Producer组名，多个Producer如果属于一个应用，发送同样的消息，则应该将它们归为同一组，即您在阿里云RocketMQ控制台上创建的Group ID，更多信息，请参见基本概念。
sendMsgTimeout	发送消息超时时间，单位：毫秒。
compressMsgBodyOverHowmuch	消息Body超过多大开始压缩（Consumer收到消息会自动解压缩），默认值：4，单位：KB。
retryTimesWhenSendFailed	如果消息发送失败，最大重试次数，该参数只对同步发送模式起作用。
maxMessageSize	客户端限制的消息大小，超过报错，同时服务端也会限制，所以需要跟服务端配合使用，默认值：4，单位：MB。

消息订阅参数

参数名	参数说明
consumerGroup	Consumer组名，多个Consumer如果属于一个应用，订阅同样的消息，且消费逻辑一致，则应该将它们归为同一组，即您在阿里云RocketMQ控制台上创建的Group ID，详情请参见基本概念。
consumeFromWhere	新的Consumer Group启动后，用于确定从何处开始拉取，默认从最新位点拉取。
consumeThreadMin	消费线程池最小线程数，默认值：20。
consumeThreadMax	消费线程池最大线程数，默认值：20。请与最小线程数保持一致。

参数名	参数说明
consumeConcurrentlyMaxSpan	单队列并行消费位点允许的最大跨度，默认值：2000，允许区间为[1,65535]。
pullThresholdForQueue	拉消息本地队列缓存消息最大数，默认值：1000，单位：条，允许区间为[1,65535]。
pullThresholdSizeForQueue	单队列本地最大缓存消息数量，默认值：100，单位：MB，允许区间为[1,1024]。
maxReconsumeTimes	最大重试次数，默认值：16，单位：次。
suspendCurrentQueueTimeMillis	顺序消息最小重试间隔，默认值：1000，单位：毫秒，允许区间为[10,30000]。

更多信息

- 概述
- 准备工作
- 收发普通消息（三种方式）
- 收发顺序消息
- 收发事务消息

4.2.3. Demo工程

本文以社区版Apache RocketMQ Client为例，提供操作示例帮助您从零开始搭建阿里云消息队列RocketMQ版收发消息的测试工程。Demo工程包含普通消息、顺序消息、事务消息的测试代码。

前提条件

- 安装IDE。
  您可以使用IntelliJ IDEA或者Eclipse，本文以IntelliJ IDEA为例。
  在IntelliJ IDEA官网下载IntelliJ IDEA Ultimate版本，并参考IntelliJ IDEA说明进行安装。
- 下载Demo工程。
  在消息队列RocketMQ版代码库下载Demo工程到本地，然后解压即可看到本地新增了`mq-demo`文件夹。
- 下载安装JDK。

配置Demo工程

1. 将Demo工程文件导入IntelliJ IDEA。
   i. 在IntelliJ IDEA界面，选择**Import Project**，选择**mq-demo/rocketmq/java-rocketmq-demo**文件夹。
   ii. 选择Import类型为**Maven**。
   iii. 默认单击**Next**，直到导入完成。Demo工程需要加载依赖的JAR包，因此导入过程需要等待2-3分钟。
2. 创建资源。
   您需要先到控制台创建所需资源，包括阿里云

消息队列RocketMQ版
的实例、Topic、Group ID（GID），以及鉴权需要的AccessKey（AK）。
更多详细信息和操作指导，请参见创建资源。

3. 配置Demo。

   您需将在步骤2中创建好的资源信息配置到文件：`MqConfig`类。

```
public static final String TOPIC = "刚才创建的Topic";
public static final String GROUP_ID = "刚才创建的Group ID";
public static final String ACCESS_KEY = "您的阿里云账号的AccessKeyId";
public static final String SECRET_KEY = "您的阿里云账号AccessKeySecret";
public static final String NAMESRV_ADDR = "您的阿里云RocketMQ实
例的TCP接入点，在控制台实例详情页面的获取接入点信息区域获取";
```

   > 说明
   > - 创建AccessKey（包括AccessKeyId和AccessKeySecret）的具体步骤，请参见获取AccessKey。
   > - 如果RAM子账号拥有该Topic的权限以及自己的AccessKey，那么也可以使用RAM用户的AccessKey。

以Main方式运行Demo

1. 发送消息。
   - 发送普通消息：运行`RocketMQProducer`类。
   - 发送事务消息：运行`RocketMQTransactionProducer`类。
     `LocalTransactionCheckerImpl`类为本地事务Check接口类，用于校验事务。更多信息，请参见收发事务消息。
   - 发送顺序消息：运行`RocketMQOrderProducer`类。
     注意创建的Topic应该为顺序消息的Topic。更多信息，请参见收发顺序消息。
2. 接收消息。
   - 接收普通消息：运行`RocketMQConsumer`类。
   - 接收事务消息：运行`RocketMQConsumer`类。
   - 接收顺序消息：运行`RocketMQOrderConsumer`类。

     可以看到消息被接收打印的日志。因为有初始化，所以需等待几秒，在生产环境中不会经常初始化。

     从
     消息队列RocketMQ版
     控制台进入Group详情，可以看到启动的消费端已经在线，并且订阅关系一致。

更多信息

- 参数说明
- 收发普通消息（三种方式）
- 收发顺序消息
- 收发事务消息

4.2.4. 准备工作

在运行社区版Java SDK代码收发消息前，您需按照本文提供的步骤来准备环境。

操作步骤

1. 通过下面两种方式可以引入依赖（任选一种）：
   - 使用Maven方式引入依赖。

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.5.2</version>
    <!--更新为最新版本-->
</dependency>
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-acl</artifactId>
    <version>4.5.2</version>
    <!--更新为最新版本-->
</dependency>
```

   - 下载依赖JAR包。
     社区版Java SDK支持连接阿里云
     消息队列RocketMQ版
     ，推荐使用的社区版Java SDK版本为4.5.2，请访问Apache RocketMQ官网下载。
2. 代码里涉及到的资源信息，例如阿里云
   消息队列RocketMQ版
   实例、Topic和Group ID等，需要到阿里云
   消息队列RocketMQ版
   控制台上创建。

   更多信息，请参见阿里云账号快速入门中的创建资源。

后续步骤

- 收发普通消息（三种方式）
- 收发顺序消息
- 收发事务消息

4.2.5. 收发普通消息（三种方式）

阿里云
消息队列RocketMQ版
提供三种方式来发送普通消息：同步发送、异步发送和单向（Oneway）发送。本文介绍了每种发送方式的原理、使用场景、示例代码，以及三种发送方式的对比；此外还提供了订阅普通消息的示例代码。

前提条件

您已完成以下操作：

- 下载4.5.2或以上版本的社区版Java SDK。
- 准备工作。

同步发送

- 原理

同步发送是指消息发送方发出一条消息后，会在收到服务端返回响应之后才发下一条消息的通讯方式。

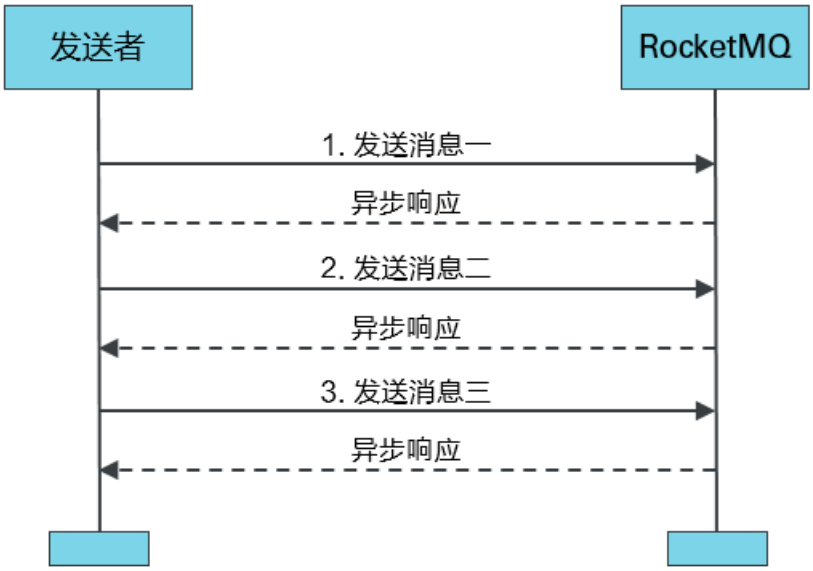
- 应用场景
  此种方式应用场景非常广泛，例如重要通知邮件、报名短信通知、营销短信系统等。
- 示例代码

```
import java.util.Date;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class RocketMQProducer {
        /**
         * 替换为您阿里云账号的AccessKey ID和AccessKey Secret。
         */
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRE
T KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         *创建Producer，并开启消息轨迹。设置为您在阿里云消息队列RocketMQ版控制台创建的Group ID。
         *如果不想开启消息轨迹，可以按照如下方式创建：
         *DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHo
ok());
         */
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook
(), true, null);
        /**
         *设置使用接入方式为阿里云，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息轨迹功能，
则运行不设置此项。
         */
```

```
        producer.setAccessChannel(AccessChannel.CLOUD);
        /**
         *设置为您从阿里云消息队列RocketMQ版控制台获取的接入点信息，类似“http://MQ_INST_XXXX.aliyuncs.com:80”。
         */
        producer.setNamesrvAddr("YOUR ACCESS POINT");
        producer.start();
        for (int i = 0; i < 128; i++) {
            try {
                Message msg = new Message("YOUR TOPIC",
                    "YOUR MESSAGE TAG",
                    "Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
                SendResult sendResult = producer.send(msg);
                System.out.printf("%s%n", sendResult);
            } catch (Exception e) {
                //消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
                System.out.println(new Date() + " Send mq message failed.");
                e.printStackTrace();
            }
        }
        //在应用退出前，销毁Producer对象。
        //注意：如果不销毁也没有问题。
        producer.shutdown();
    }
}
```

异步发送

- 原理
  异步发送是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式。阿里云消息队列RocketMQ版
  的异步发送，需要实现异步发送回调接口（SendCallback）。消息发送方在发送了一条消息后，不需要等待服务端响应即可发送第二条消息。发送方通过回调接口接收服务端响应，并处理响应结果。



- 应用场景

异步发送一般用于链路耗时较长，对响应时间较为敏感的业务场景，例如，您视频上传后通知启动转码服务，转码完成后通知推送转码结果等。

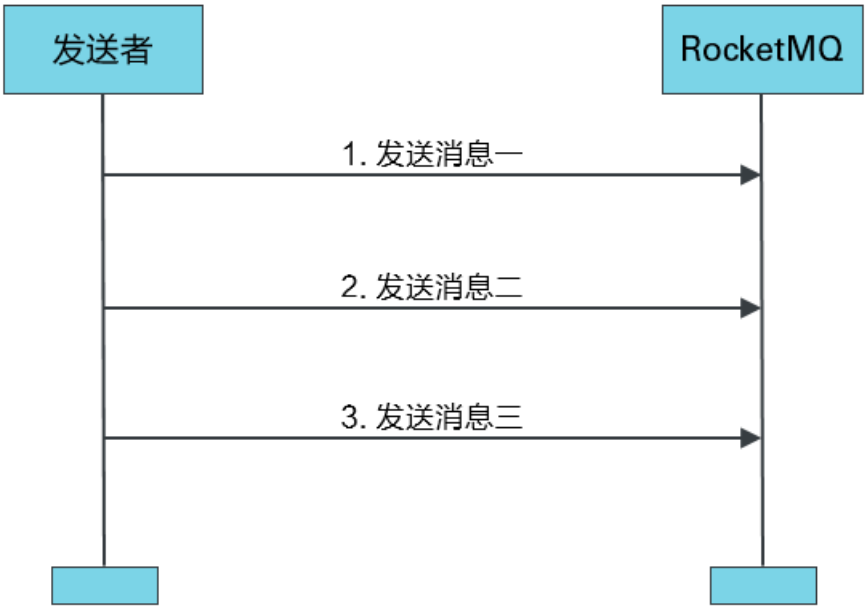
- 示例代码

```
import java.util.Date;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendCallback;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class RocketMQAsyncProducer {
        /**
         * 替换为您阿里云账号的AccessKey ID和AccessKey Secret。
         */
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRE
T KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建Producer，并开启消息轨迹。设置为您在阿里云消息队列RocketMQ版控制台创建的Group ID。
         * 如果不想开启消息轨迹，可以按照如下方式创建：
         * DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCH
ook());
         */
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook
(), true, null);
        /**
         * 设置使用接入方式为阿里云，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息轨迹功能，则运行不设置此项。
         */
        producer.setAccessChannel(AccessChannel.CLOUD);
        /**
         * 设置为您从阿里云消息队列RocketMQ版控制台获取的接入点信息，类似“http://MQ_INST_XXXX.al
iyuncs.com:80”。
         */
        producer.setNamesrvAddr("YOUR ACCESS POINT");
        producer.start();
        for (int i = 0; i < 128; i++) {
            try {
                Message msg = new Message("YOUR TOPIC",
                    "YOUR MESSAGE TAG",
                    "Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
                producer.send(msg, new SendCallback() {
                    @Override public void onSuccess(SendResult result) {
                        // 消费发送成功。
                        System.out.println("send message success. msgId= " + result.getMs
gId());
                    }
```

```
                    @Override public void onException(Throwable throwable) {
                        // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
                        System.out.println("send message failed.");
                        throwable.printStackTrace();
                    }
                });
            } catch (Exception e) {
                // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
                System.out.println(new Date() + " Send mq message failed.");
                e.printStackTrace();
            }
        }
        // 在应用退出前，销毁Producer对象。
        // 注意：如果不销毁也没有问题。
        producer.shutdown();
    }
}
```

单向（Oneway）发送

- 原理
  发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发，即只发送请求不等待应答。此方式发送消息的过程耗时非常短，一般在微秒级别。



- 应用场景
  适用于某些耗时非常短，但对可靠性要求并不高的场景，例如日志收集。
- 示例代码

```
import java.util.Date;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
```

```
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class RocketMQOnewayProducer {
        /**
         * 替换为您阿里云账号的AccessKey ID和AccessKey Secret。
         */
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRE
T KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建Producer，并开启消息轨迹。设置为您在阿里云消息队列RocketMQ版控制台创建的Group ID。
         * 如果不想开启消息轨迹，可以按照如下方式创建：
         * DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCH
ook());
         */
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook
(), true, null);
        /**
         * 设置使用接入方式为阿里云，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息轨迹功能
，则运行不设置此项。
         */
        producer.setAccessChannel(AccessChannel.CLOUD);
        /**
         * 设置为您从阿里云消息队列RocketMQ版控制台获取的接入点信息，类似“http://MQ_INST_XXXX.al
iyuncs.com:80”。
         */
        producer.setNamesrvAddr("YOUR ACCESS POINT");
        producer.start();
        for (int i = 0; i < 128; i++) {
            try {
                Message msg = new Message("YOUR TOPIC",
                    "YOUR MESSAGE TAG",
                    "Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
                producer.sendOneway(msg);
            } catch (Exception e) {
                // 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
                System.out.println(new Date() + " Send mq message failed.");
                e.printStackTrace();
            }
        }
        // 在应用退出前，销毁Producer对象。
        // 注意：如果不销毁也没有问题。
        producer.shutdown();
    }
}
```

三种发送方式的对比

下表概括了三者的特点和主要区别。

发送方式	发送TPS	发送结果反馈	可靠性
同步发送	快	有	不丢失
异步发送	快	有	不丢失
单向发送	最快	无	可能丢失

订阅普通消息

订阅普通消息的方式只有以下一种。

```
import java.util.List;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.message.MessageExt;
import org.apache.rocketmq.remoting.RPCHook;
public class RocketMQPushConsumer {
       /**
         * 替换为您阿里云账号的AccessKey ID和AccessKey Secret。
         */
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRET
KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建Consumer，并开启消息轨迹。设置为您在阿里云消息队列RocketMQ版控制台创建的Group ID。
         * 如果不想开启消息轨迹，可以按照如下方式创建：
         * DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR GROUP ID", getA
clRPCHook(), new AllocateMessageQueueAveragely());
         */
         DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR GROUP ID", getAcl
RPCHook(), new AllocateMessageQueueAveragely(), true, null);
        //设置为阿里云消息队列RocketMQ版实例的接入点。
        consumer.setNamesrvAddr("http://xxxx.mq-internet.aliyuncs.com:80");
        //阿里云上消息轨迹需要设置为CLOUD方式，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息
轨迹功能，则运行不设置此项。
        consumer.setAccessChannel(AccessChannel.CLOUD);
        // 设置为您在阿里云消息队列RocketMQ版控制台上创建的Topic。
        consumer.subscribe("YOUR TOPIC", "*");
        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
                ConsumeConcurrentlyContext context) {
                System.out.printf("Receive New Messages: %s %n", msgs);
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        consumer.start();
    }
}
```

4.2.6. 收发顺序消息

顺序消息（FIFO消息）是
消息队列RocketMQ版

提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用TCP协议下的社区版Java SDK收发顺序消息的示例代码供您参考。

背景信息

顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息2.0。

前提条件

您已完成以下操作：

- 下载4.5.2或以上版本的社区版Java SDK。
- 准备工作。

发送顺序消息

> **注意**
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

发送顺序消息的示例代码如下。

```
import java.util.List;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.MessageQueueSelector;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.common.message.MessageQueue;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class RocketMQOrderProducer {
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRET
KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建Producer，并开启消息轨迹。
         * 如果不想开启消息轨迹，可以按照以下方式创建：
         * DefaultMQProducer producer = new DefaultMQProducer("YOUR ORDER GROUP ID", getAcl
RPCHook());
```

```
         */
        DefaultMQProducer producer = new DefaultMQProducer("YOUR ORDER GROUP ID", getAclRPC
Hook(), true, null);
        /**
         * 设置使用接入方式为阿里云，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息轨迹功能，则运行不设置此项。
         */
        producer.setAccessChannel(AccessChannel.CLOUD);
        /**
         * 设置为您从阿里云控制台获取的接入点信息，类似“http://MQ_INST_XXXX.aliyuncs.com:80”。
         */
        producer.setNamesrvAddr("http://xxxx.mq-internet.aliyuncs.com:80");
        producer.start();
        for (int i = 0; i < 128; i++) {
            try {
                int orderId = i % 10;
                Message msg = new Message("YOUR ORDER TOPIC",
                    "YOUR MESSAGE TAG",
                    "OrderID188",
                    "Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
                SendResult sendResult = producer.send(msg, new MessageQueueSelector() {
                    @Override
                    public MessageQueue select(List<MessageQueue> mqs, Message msg, Object
arg) {
                        // 选择适合自己的分区选择算法，保证同一个参数得到的结果相同。
                        Integer id = (Integer) arg;
                        int index = id % mqs.size();
                        return mqs.get(index);
                    }
                }, orderId);
                System.out.printf("%s%n", sendResult);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        producer.shutdown();
    }
}
```

订阅顺序消息

订阅顺序消息的示例代码如下。

```
import java.util.List;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeOrderlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeOrderlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerOrderly;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.consumer.ConsumeFromWhere;
import org.apache.rocketmq.common.message.MessageExt;
import org.apache.rocketmq.remoting.RPCHook;
public class RocketMQOrderConsumer {
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRET
KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建Consumer，并开启消息轨迹。
         * 如果不想开启消息轨迹，可以按照如下方式创建：
         * DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR ORDER GROUP ID"
, getAclRPCHook(), null);
         */
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR ORDER GROUP ID", g
etAclRPCHook(), new AllocateMessageQueueAveragely(), true, null);
        /**
         * 设置使用接入方式为阿里云，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息轨迹功能，则运行不设置此项。
         */
        consumer.setAccessChannel(AccessChannel.CLOUD);
        /**
         * 设置为您从阿里云控制台获取的接入点信息，类似“http://MQ_INST_XXXX.aliyuncs.com:80”。
         */
        consumer.setNamesrvAddr("http://xxxx.mq-internet.aliyuncs.com:80");
        consumer.subscribe(“YOUR ORDER TOPIC”, "*");
        consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
        consumer.registerMessageListener(new MessageListenerOrderly() {
            @Override
            public ConsumeOrderlyStatus consumeMessage(List<MessageExt> msgs, ConsumeOrderl
yContext context) {
                context.setAutoCommit(true);
                System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().
getName(), msgs);
                return ConsumeOrderlyStatus.SUCCESS;// 消费失败则挂起重试返回：ConsumeOrderlyS
tatus.SUSPEND_CURRENT_QUEUE_A_MOMENT
            }
        });
        consumer.start();
        System.out.printf("Consumer Started.%n");
    }
```

4.2.7. 收发定时消息和延时消息

本文提供使用社区版TCP协议下的Java SDK收发定时消息和延时消息的示例代码。

背景信息

- 定时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。
- 延时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。

更多信息，请参见定时和延时消息。

> **注意** 社区版的Apache RocketMQ和阿里云消息队列RocketMQ版配置方式不同，实现效果也有所差异。社区版的Apache RocketMQ支持延时消息，但不支持定时消息，因此没有专门的定时消息接口。阿里云消息队列RocketMQ版不仅同时支持配置延时消息和定时消息，并且定时和延时时间可以精确到秒级、拥有更高的并发性。建议您优先使用云上定时延时的方式，使用方法，请参考以下步骤。

前提条件

您已完成以下操作：

- 下载4.5.2或以上版本的社区版Java SDK。
- 准备工作。

发送定时消息或延时消息

发送定时消息或延时消息的示例代码如下。

```
import java.util.Date;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class RocketMQProducer {
        /*
        *替换为您阿里云账号的AccessKey ID和AccessKey Secret。
        */
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRET
KEY"));
    }
    public static void main(String[] args) throws MQClientException {
```

```
        /**
         *创建Producer，并开启消息轨迹。设置为您在阿里云
消息队列RocketMQ版控制台创建的Group ID。
         *如果不想开启消息轨迹，可以按照如下方式创建：
         *DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook
());
         */
        DefaultMQProducer producer = new DefaultMQProducer("YOUR GROUP ID", getAclRPCHook()
, true, null);
        /**
         *设置使用接入方式为阿里云，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息轨迹功能，则
运行不设置此项。
         */
        producer.setAccessChannel(AccessChannel.CLOUD);
        /**
         *设置为您从阿里云
消息队列RocketMQ版控制台获取的接入点信息，类似“http://MQ_INST_XXXX.aliyuncs.com:80”。
         */
        producer.setNamesrvAddr("YOUR ACCESS POINT");
        producer.start();
        for (int i = 0; i < 128; i++) {
            try {
                /*设置为您在
消息队列RocketMQ版控制台创建的Topic。*/
                Message msg = new Message("YOUR TOPIC",
                    /*设置消息的Tag。*/
                    "YOUR MESSAGE TAG",
                    /*消息内容。*/
                    "Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
                /*发送延时消息，需要设置延时时间，单位毫秒（ms），消息将在指定延时时间后投递，例如消
息将在3秒后投递。*/
                long delayTime = System.currentTimeMillis() + 3000;
                msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(delayTime));
                /**
                *若需要发送定时消息，则需要设置定时时间，消息将在指定时间进行投递，例如消息将在2021-0
8-10 18:45:00投递。
                *定时时间格式为：yyyy-MM-dd HH:mm:ss，若设置的时间戳在当前时间之前，则消息将被立即
投递给Consumer。
                * long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2021-
08-10 18:45:00").getTime();
                * msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(timeStamp));
                */
                SendResult sendResult = producer.send(msg);
                System.out.printf("%s%n", sendResult);
            } catch (Exception e) {
                //消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理。
                System.out.println(new Date() + " Send mq message failed.");
                e.printStackTrace();
            }
        }
        //在应用退出前，销毁Producer对象。
        //注意：如果不销毁也没有问题。
        producer.shutdown();
    }
```

```
}
```

订阅定时消息或延时消息

订阅定时消息或延时消息的示例代码如下。

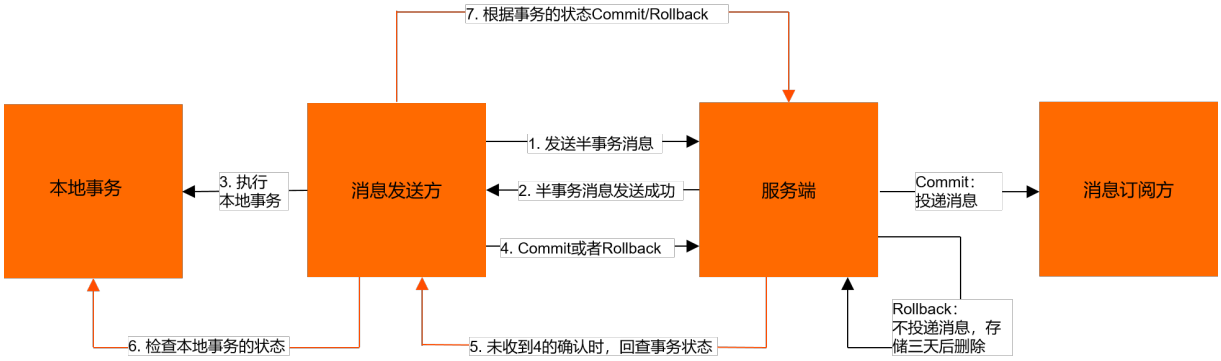
```
import java.util.List;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.message.MessageExt;
import org.apache.rocketmq.remoting.RPCHook;
public class RocketMQPushConsumer {
       /**
         * 替换为您阿里云账号的AccessKey ID和AccessKey Secret。
         */
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRET
KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建Consumer，并开启消息轨迹。设置为您在阿里云消息队列RocketMQ版控制台创建的Group ID。
         * 如果不想开启消息轨迹，可以按照如下方式创建：
         * DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR GROUP ID", getA
clRPCHook(), new AllocateMessageQueueAveragely());
         */
         DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR GROUP ID", getAcl
RPCHook(), new AllocateMessageQueueAveragely(), true, null);
        //设置为阿里云消息队列RocketMQ版实例的接入点。
        consumer.setNamesrvAddr("http://xxxx.mq-internet.aliyuncs.com:80");
        //阿里云上消息轨迹需要设置为CLOUD方式，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息
轨迹功能，则运行不设置此项。
        consumer.setAccessChannel(AccessChannel.CLOUD);
        // 设置为您在阿里云消息队列RocketMQ版控制台上创建的Topic。
        consumer.subscribe("YOUR TOPIC", "*");
        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
                ConsumeConcurrentlyContext context) {
                System.out.printf("Receive New Messages: %s %n", msgs);
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        consumer.start();
    }
}
```

4.2.8. 收发事务消息

本文提供使用TCP协议下的社区版Java SDK收发事务消息的示例代码供您参考。

交互流程

事务消息交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

您已完成以下操作：

- 下载4.5.2或以上版本的社区版Java SDK。
- 准备工作。

发送事务消息

发送事务消息包含以下两个步骤：

1. 发送半事务消息（Half Message）及执行本地事务，示例代码如下。

```
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.LocalTransactionExecuter;
import org.apache.rocketmq.client.producer.LocalTransactionState;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.client.producer.TransactionMQProducer;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.RPCHook;
import org.apache.rocketmq.remoting.common.RemotingHelper;
public class RocketMQTransactionProducer {
    private static RPCHook getAclRPCHook() {
        /**
         * 替换为您阿里云账号的AccessKey ID和AccessKey Secret。
         */
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SEC
RET KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建事务消息Producer，并开启消息轨迹。设置为您在
消息队列RocketMQ版控制台创建的Group ID。
         * 如果不想开启消息轨迹，可以按照如下方式创建：
```

```
         * TransactionMQProducer transactionMQProducer = new TransactionMQProducer("YOU
R TRANSACTION GROUP ID", getAclRPCHook());
         */
         TransactionMQProducer transactionMQProducer = new TransactionMQProducer(null,
"YOUR TRANSACTION GROUP ID", getAclRPCHook(), true, null);
         /**
        * 设置为您从阿里云
消息队列RocketMQ版控制台获取的接入点信息，类似“http://MQ_INST_XXXX.aliyuncs.com:80”。
        */
     transactionMQProducer.setNamesrvAddr("http://xxxx.mq-internet.aliyuncs.com:80");
        transactionMQProducer.setTransactionCheckListener(new LocalTransactionCheckerIm
pl());
        transactionMQProducer.start();
        for (int i = 0; i < 10; i++) {
            try {
                Message message = new Message("YOUR TRANSACTION TOPIC",
                    "YOUR MESSAGE TAG",
                    "Hello world".getBytes(RemotingHelper.DEFAULT_CHARSET));
                SendResult sendResult = transactionMQProducer.sendMessageInTransaction(
message, new LocalTransactionExecuter() {
                    @Override public LocalTransactionState executeLocalTransactionBranc
h(Message msg, Object arg) {
                        System.out.println("开始执行本地事务：  " + msg);
                        return LocalTransactionState.UNKNOW;
                    }
                }, null);
                assert sendResult != null;
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

2. 提交事务消息状态
   当本地事务执行完成（执行成功或执行失败），需要通知服务器当前消息的事务状态。通知方式有以下两种：
   - 执行本地事务完成后提交。
   - 执行本地事务一直没提交状态，等待服务器回查消息的事务状态。

   事务状态有以下三种：
   - `LocalTransactionState.COMMIT_MESSAGE`：提交事务，允许订阅方消费该消息。
   - `LocalTransactionState.ROLLBACK_MESSAGE`：回滚事务，消息将被丢弃不允许消费。
   - `LocalTransactionState.UNKNOW`：无法判断状态，期待阿里云消息队列RocketMQ版的Broker向发送方再次询问该消息对应的本地事务的状态。

```
import org.apache.rocketmq.client.producer.LocalTransactionState;
import org.apache.rocketmq.client.producer.TransactionCheckListener;
import org.apache.rocketmq.common.message.MessageExt;
/**
 *
消息队列RocketMQ版发送事务消息本地Check接口实现类。
 */
public class LocalTransactionCheckerImpl implements TransactionCheckListener {
    /**
     * 本地事务Checker。更多信息，请参见事务消息。
     */
    @Override public LocalTransactionState checkLocalTransactionState(MessageExt msg) {
        System.out.println("收到事务消息的回查请求，MsgId: " + msg.getMsgId());
        return LocalTransactionState.COMMIT_MESSAGE;
    }
}
```

事务回查机制说明

- 发送事务消息为什么必须要实现回查Check机制？
  当步骤1中半事务消息发送完成，但本地事务返回状态为 `LocalTransactionState.UNKNOW` ，或者应用退出导致本地事务未提交任何状态时，从Broker的角度看，这条Half状态的消息的状态是未知的。因此Broker会定期要求发送方能Check该Half状态消息，并上报其最终状态。
- Check被回调时，业务逻辑都需要做些什么？
  事务消息的Check方法里面，应该写一些检查事务一致性的逻辑。
  消息队列RocketMQ版
  发送事务消息时需要实现 `LocalTransactionChecker` 接口，用来处理Broker主动发起的本地事务状态回查请求；因此在事务消息的Check方法中，需要完成两件事情：
  i. 检查该半事务消息对应的本地事务的状态（committed or rollback）。
  ii. 向Broker提交该半事务消息本地事务的状态。

订阅事务消息

事务消息的订阅与普通消息订阅一致，如下所示。

```
import java.util.List;
import org.apache.rocketmq.acl.common.AclClientRPCHook;
import org.apache.rocketmq.acl.common.SessionCredentials;
import org.apache.rocketmq.client.AccessChannel;
import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.consumer.rebalance.AllocateMessageQueueAveragely;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.message.MessageExt;
import org.apache.rocketmq.remoting.RPCHook;
public class RocketMQPushConsumer {
       /**
         * 替换为您阿里云账号的AccessKey ID和AccessKey Secret。
         */
    private static RPCHook getAclRPCHook() {
        return new AclClientRPCHook(new SessionCredentials("YOUR ACCESS KEY", "YOUR SECRET
KEY"));
    }
    public static void main(String[] args) throws MQClientException {
        /**
         * 创建Consumer，并开启消息轨迹。设置为您在阿里云消息队列RocketMQ版控制台创建的Group ID。
         * 如果不想开启消息轨迹，可以按照如下方式创建：
         * DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR GROUP ID", getA
clRPCHook(), new AllocateMessageQueueAveragely());
         */
         DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("YOUR GROUP ID", getAcl
RPCHook(), new AllocateMessageQueueAveragely(), true, null);
        //设置为阿里云消息队列RocketMQ版实例的接入点。
        consumer.setNamesrvAddr("http://xxxx.mq-internet.aliyuncs.com:80");
        //阿里云上消息轨迹需要设置为CLOUD方式，在使用云上消息轨迹的时候，需要设置此项，如果不开启消息
轨迹功能，则运行不设置此项。
        consumer.setAccessChannel(AccessChannel.CLOUD);
        // 设置为您在阿里云消息队列RocketMQ版控制台上创建的Topic。
        consumer.subscribe("YOUR TOPIC", "*");
        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
                ConsumeConcurrentlyContext context) {
                System.out.printf("Receive New Messages: %s %n", msgs);
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        consumer.start();
    }
}
```

4.3. 社区版C++ SDK接入说明

4.3.1. 概述

本文主要介绍如何使用开源社区发布的rocketmq-client-cpp二进制包进行部署和连接阿里云消息队列RocketMQ版的过程。如需源码编译，请您自行前往开源社区下载。

使用说明

社区版SDK仅在开源RocketMQ迁移上云且不希望修改代码时使用，其他场景推荐您使用阿里云消息队列RocketMQ版提供的商业版SDK进行接入。和社区版SDK相比，商业版SDK提供了更加丰富的功能特性并具有更高的稳定性保障。更多信息，请参见商业版C++ SDK。

安装CPP动态库

请参见安装CPP动态库。

（可选）参数说明

在使用C++ SDK接入阿里云消息队列RocketMQ版收发消息时，需要配置相应的参数。更多信息，请参见参数说明。
您也可以参见C++ SDK中的注释了解参数说明。

使用C++ SDK收发消息

C++ SDK支持收发以下消息：

- 收发普通消息
- 收发顺序消息
- 收发事务消息
- 收发定时和延时消息

更多信息

如果在整个流程中遇到问题，请提交工单。

4.3.2. 安装CPP动态库

在使用社区版C++ SDK接入阿里云消息队列RocketMQ版来收发消息前，您需按本文提供的操作步骤安装CPP动态库。

前提条件

在开始做准备工作前，请确保您的操作系统满足以下条件：

- Linux：CentOS 6.8、CentOS 7.2、RHEL 6.x、RHEL 7.x
- Darwin：macOS Mojave 10.14.x
- Debian：Ubuntu 18.04

> **说明** 本文不提供gcc环境的安装指导，请确保机器的gcc/g++环境版本在4.8以上。

安装CPP动态库

> **注意** CPP动态库默认安装到系统动态库目录下，请确保当前账号拥有sudo权限或请使用root账号来执行操作。

目前CPP的动态库已经提供了二进制release，可直接获取开源代码。更多信息，请参见Release Notes。为方便安装，本文以社区版2.0.1版本为例，针对不同的操作系统分别进行说明：

- CentOS 7.2和RHEL 7.x
  CentOS默认支持RPM管理，RPM包名为rocketmq-client-cpp-2.0.1，可以执行以下 `rpm` 命令直接安装。
  ```
  rpm -ivh https://github.com/apache/rocketmq-client-cpp/releases/download/2.0.1/rocketmq-client-cpp-2.0.1-centos7.x86_64.rpm
  ```
- CentOS 6.8和RHEL 6.x
  CentOS 6.8与CentOS 7相比，安装步骤相同，仅使用的RPM包不一样，参考以下命令。
  ```
  rpm -ivh https://github.com/apache/rocketmq-client-cpp/releases/download/2.0.1/rocketmq-client-cpp-2.0.1-centos6.x86_64.rpm
  ```
- macOS Mojave 10.14
  macOS下不提供包管理工具，您可以执行以下命令手动安装动态库。
  ```
  mkdir cppsdk
  cd cppsdk
  wget https://github.com/apache/rocketmq-client-cpp/releases/download/2.0.1/rocketmq-client-cpp-2.0.1-bin-release-darwin.tar.gz
  tar -xzf rocketmq-client-cpp-2.0.1-bin-release-darwin.tar.gz
  cp rocketmq-client-cpp/lib/* /usr/local/lib/
  mkdir -p /usr/local/include/rocketmq/
  cp rocketmq-client-cpp/include/* /usr/local/include/rocketmq/
  install_name_tool -id "@rpath/librocketmq.dylib" /usr/local/lib/librocketmq.dylib
  ```
- Ubuntu 18.04
  Ubuntu 18.04操作系统内核是Debain系统，使用的默认包管理工具为dpkg，包名为rocketmq_2.0.1_amd64.deb。您可执行以下命令安装。
  ```
  wget https://github.com/apache/rocketmq-client-cpp/releases/download/2.0.1/rocketmq-client-cpp-2.0.1.amd64.deb
  dpkg -i rocketmq-client-cpp-2.0.1.amd64.deb
  ```

至此，您已完成CPP动态库的安装。

后续步骤

- 收发普通消息
- 收发顺序消息
- 收发事务消息
- 收发定时和延时消息

4.3.3. 参数说明

本文介绍您在使用社区版C++ SDK接入阿里云
消息队列RocketMQ版
时，需要配置的参数。

通用参数

参数名	说明
NameServer	设置TCP协议接入点，从消息队列RocketMQ版控制台的**实例详情**页面获取。
GroupID	您在消息队列RocketMQ版控制台上创建的Group ID，更多信息，请参见基本概念。
AccessKey	您在阿里云账号管理控制台中创建的AccessKey ID，用于身份认证。
SecretKey	您在阿里云账号管理控制台中创建的AccessKey Secret，用于身份认证。
Channel	用户渠道，默认值为ALIYUN。

4.3.4. 收发普通消息

普通消息是指阿里云消息队列RocketMQ版中无特性的消息，区别于有特性的定时消息、顺序消息和事务消息。本文提供使用TCP协议下的社区版C++ SDK收发普通消息的示例代码供您参考。

前提条件

安装CPP动态库

发送普通消息

1. 将以下代码拷贝到*ProducerDemo.cpp*文件中，修改相应的参数后，使用g++命令进行编译，生成可执行文件，即可直接运行。

```
g++ -o producer_demo -std=c++11 -lz -lrocketmq ProducerDemo.cpp
```

2. 发送普通消息的示例代码如下。

```
#include <iostream>
#include <chrono>
#include <thread>
#include "DefaultMQProducer.h"
using namespace std;
using namespace rocketmq;
int main() {
    std::cout << "=======Before sending messages=======" << std::endl;
    //您在阿里云消息队列RocketMQ控制台上申请的GID。
    DefaultMQProducer producer("GID_XXXXXXXXX");
    //设置TCP协议接入点，从阿里云RocketMQ控制台的实例详情页面获取。
    producer.setNamesrvAddr("http://MQ_INST_XXXXXXXX.mq-internet-access.mq-internet.aliyuncs.com:80");
    //您在阿里云账号管理控制台中创建的AccessKey ID和AccessKey Secret，用于身份认证，用户渠道，默认值为：ALIYUN。
    producer.setSessionCredentials("AK","SK","ALIYUN");
    //请确保参数设置完成之后启动Producer。
    producer.start();
    auto start = std::chrono::system_clock::now();
    int count = 32;
    for (int i = 0; i < count; ++i) {
        //发送消息时请设置您在阿里云消息队列RocketMQ控制台上申请的Topic。
        MQMessage msg("YOURTOPIC","HiTAG","HelloCPPSDK.");
        try {
            SendResult sendResult = producer.send(msg);
            std::cout <<"SendResult:"<<sendResult.getSendStatus()<< ", Message ID: " << sendResult.getMsgId() << std::endl;
            this_thread::sleep_for(chrono::seconds(1));
        } catch (MQException e) {
            std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() << std::endl;
        }
    }
    auto interval = std::chrono::system_clock::now() - start;
    std::cout << "Send " << count << " messages OK, costs "
              << std::chrono::duration_cast<std::chrono::milliseconds>(interval).count() << "ms" << std::endl;
    producer.shutdown();
    std::cout << "=======After sending messages=======" << std::endl;
    return 0;
}
```

消费普通消息

1. 将以下代码拷贝到*ConsumerDemo.cpp*文件中，修改相应的参数后，使用g++命令进行编译，生成可执行文件，即可直接运行。
   ```
   g++ -o consumer_demo -std=c++11 -lz -lrocketmq ConsumerDemo.cpp
   ```
2. 消费普通消息的示例代码如下。

```
#include <iostream>
#include <thread>
#include "DefaultMQPushConsumer.h"
using namespace rocketmq;
class ExampleMessageListener : public MessageListenerConcurrently {
public:
    ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
        for (auto item = msgs.begin(); item != msgs.end(); item++) {
            std::cout << "Received Message Topic:" << item->getTopic() << ", MsgId:" <<
item->getMsgId() << std::endl;
        }
        return CONSUME_SUCCESS;
    }
};
int main(int argc, char *argv[]) {
    std::cout << "=======Before consuming messages=======" << std::endl;
    //您在阿里云消息队列RocketMQ控制台上申请的GID。
    DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer("GID_XXXXXXXX");
    //设置TCP协议接入点，从阿里云消息队列RocketMQ控制台的实例详情页面获取。
    consumer->setNamesrvAddr("http://MQ_INST_XXXXXXXXXXX.mq-internet-access.mq-internet
.aliyuncs.com:80");
    //您在阿里云账号管理控制台中创建的AccessKey ID和AccessKey Secret，用于身份认证，用户渠道，
默认值为：ALIYUN。
    consumer->setSessionCredentials("AK", "SK", "ALIYUN");
    auto start = std::chrono::system_clock::now();
    //register your own listener here to handle the messages received.
    //请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
    ExampleMessageListener *messageListener = new ExampleMessageListener();
    consumer->subscribe("YOURTOPIC", "*");
    consumer->registerMessageListener(messageListener);
    //Start this consumer
    //准备工作完成，必须调用启动函数，才可以正常工作。
    // ******************************************
    // 1.确保订阅关系的设置在启动之前完成。
    // 2.确保相同GID下面的消费者的订阅关系一致。
    // ******************************************
    consumer->start();
    //Keep main thread running until process finished.
    //请保持线程常驻，不要执行shutdown操作。
    std::this_thread::sleep_for(std::chrono::milliseconds(60 * 1000));
    consumer->shutdown();
    std::cout << "=======After consuming messages======" << std::endl;
    return 0;
}
```

4.3.5. 收发顺序消息

顺序消息（FIFO消息）是阿里云
消息队列RocketMQ版
提供的一种严格按照顺序来发布和消费的消息类型。本文提供使用TCP协议下的社区版C++ SDK收发顺序消息的示例代码供您参考。
顺序消息分为两类：

- 全局顺序：对于指定的一个Topic，所有消息按照严格的先入先出FIFO（First In First Out）的顺序进行发布和消费。
- 分区顺序：对于指定的一个Topic，所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段，和普通消息的Key是完全不同的概念。

更多信息，请参见顺序消息2.0。

前提条件

安装CPP动态库

发送顺序消息

> **注意**
> 消息队列RocketMQ版
> 服务端判定消息产生的顺序性是参照单一生产者、单一线程并发下消息发送的时序。如果发送方有多个生产者或者有多个线程并发发送消息，则此时只能以到达
> 消息队列RocketMQ版
> 服务端的时序作为消息顺序的依据，和业务侧的发送顺序未必一致。

1. 将以下代码拷贝到*OrderProducerDemo.cpp*文件中，修改相应的参数后，使用g++命令进行编译，生成可执行文件，即可直接运行。

```
g++ -o order_producer_demo -std=c++11 -lz -lrocketmq OrderProducerDemo.cpp
```

2. 发送顺序消息的示例代码如下。

```
#include <iostream>
#include <chrono>
#include <thread>
#include "DefaultMQProducer.h"
using namespace std;
using namespace rocketmq;
class ExampleSelectMessageQueueByHash : public MessageQueueSelector {
public:
    MQMessageQueue select(const std::vector<MQMessageQueue> &mqs, const MQMessage &msg,
void *arg) {
        // 实现自定义分区逻辑，根据业务传入arg参数即分区键，计算路由到哪个队列，这里以arg为int型参数为例。
        int orderId = *static_cast<int *>(arg);
        int index = orderId % mqs.size();
        return mqs[0];
    }
};
int main() {
    std::cout << "=======Before sending messages=======" << std::endl;
    //您在阿里云消息队列RocketMQ控制台上申请的GID。
    DefaultMQProducer producer("GID_XXXXXXXX");
    //设置TCP协议接入点，从阿里云消息队列RocketMQ控制台的实例详情页面获取。
    producer.setNamesrvAddr("http://MQ_INST_XXXXXXXXXX.mq-internet-access.mq-internet.aliyuncs.com:80");
    //您在阿里云账号管理控制台中创建的AccessKey Id和AccessKey Secret，用于身份认证，用户渠道，默认值为：ALIYUN。
    producer.setSessionCredentials("AK", "SK", "ALIYUN");
    //请确保参数设置完成之后启动Producer。
```

```
    producer.start();
    auto start = std::chrono::system_clock::now();
    int count = 32;
    //可以设置发送重试的次数，确保发送成功。
    int retryTimes = 1;
    //参考接口MessageQueueSelector，通过设置的自定义参数arg，计算发送到指定的路由队列中，此处的arg便是分区ID。
    ExampleSelectMessageQueueByHash *pSelector = new ExampleSelectMessageQueueByHash();
    for (int i = 0; i < count; ++i) {
        //发送消息时请设置您在阿里云消息队列RocketMQ控制台上申请的Topic。
        MQMessage msg("YOUR ORDERLY TOPIC", "HiTAG", "Hello,CPP SDK, Orderly Message.");
        try {
            SendResult sendResult = producer.send(msg, pSelector, &i, 1, false);
            std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID: " << sendResult.getMsgId()
                      << "MessageQueue:" << sendResult.getMessageQueue().toString() << std::endl;
            this_thread::sleep_for(chrono::seconds(1));
        } catch (MQException e) {
            std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() << std::endl;
        }
    }
    auto interval = std::chrono::system_clock::now() - start;
    std::cout << "Send " << count << " messages OK, costs "
              << std::chrono::duration_cast<std::chrono::milliseconds>(interval).count() << "ms" << std::endl;
    producer.shutdown();
    std::cout << "=======After sending messages=======" << std::endl;
    return 0;
}
```

消费顺序消息

1. 将以下代码拷贝到*OrderConsumerDemo.cpp*文件中，修改相应的参数后，使用g++命令进行编译，生成可执行文件，即可直接运行。

```
g++ -o order_consumer_demo -std=c++11 -lz -lrocketmq OrderConsumerDemo.cpp
```

2. 消费顺序消息的示例代码如下。

```
#include <iostream>
#include <thread>
#include "DefaultMQPushConsumer.h"
using namespace rocketmq;
class ExampleOrderlyMessageListener : public MessageListenerOrderly {
public:
    ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
        for (auto item = msgs.begin(); item != msgs.end(); item++) {
            std::cout << "Received Message Topic:" << item->getTopic() << ", MsgId:" <<
item->getMsgId() << std::endl;
        }
        return CONSUME_SUCCESS;
    }
};
int main(int argc, char *argv[]) {
    std::cout << "=======Before consuming messages=======" << std::endl;
    //您在阿里云消息队列RocketMQ控制台上申请的GID。
    DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer("GID_XXXXXXXXXX");
    //设置TCP协议接入点，从阿里云消息队列RocketMQ控制台的实例详情页面获取。
    consumer->setNamesrvAddr("http://MQ_INST_XXXXXXXXXXXXXX.mq-internet-access.mq-inter
net.aliyuncs.com:80");
    //您在阿里云账号管理控制台中创建的AccessKey ID和AccessKey Secret用于身份认证，用户渠道，默
认值为：ALIYUN。
    consumer->setSessionCredentials("AK", "SK", "ALIYUN");
    auto start = std::chrono::system_clock::now();
    //register your own listener here to handle the messages received.
    //请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
    ExampleOrderlyMessageListener *messageListener = new ExampleOrderlyMessageListener(
);
    consumer->subscribe("YOUR ORDERLY TOPIC", "*");
    consumer->registerMessageListener(messageListener);
    //Start this consumer
    //准备工作完成，必须调用启动函数，才可以正常工作。
    // ****************************************
    // 1.确保订阅关系的设置在启动之前完成。
    // 2.确保相同GID下面的消费者的订阅关系一致。
    // ****************************************
    consumer->start();
    //Keep main thread running until process finished.
    //请保持线程常驻，不要执行shutdown操作。
    std::this_thread::sleep_for(std::chrono::seconds (60 ));
    consumer->shutdown();
    std::cout << "=======After consuming messages======" << std::endl;
    return 0;
}
```

4.3.6. 收发事务消息

本文提供使用TCP协议下的社区版C++ SDK来收发事务消息的示例代码供您参考。
阿里云

消息队列RocketMQ版

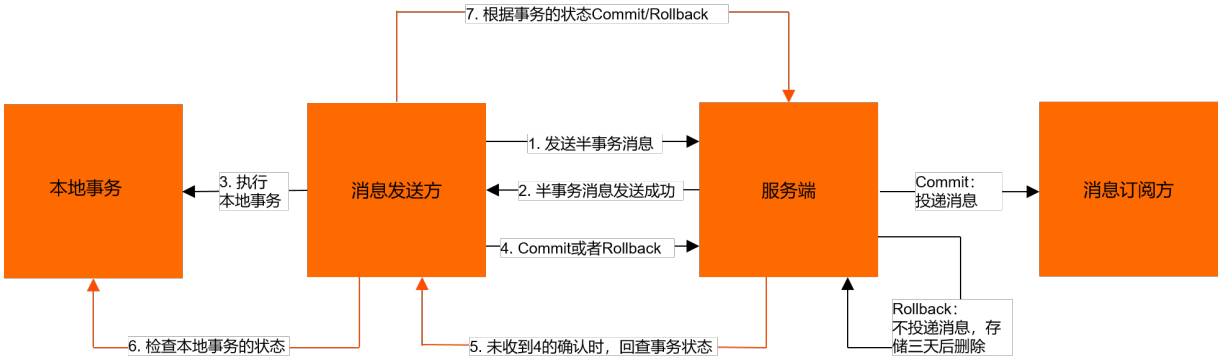
提供类似XA或Open XA的分布式事务功能，通过阿里云

消息队列RocketMQ版

事务消息，能达到分布式事务的最终一致。

交互流程

事务消息交互流程如下图所示。



更多信息，请参见事务消息。

前提条件

安装CPP动态库

发送事务消息

1. 将以下代码拷贝到*TransProducerDemo.cpp*文件中，修改相应的参数后，使用g++命令进行编译，生成可执行文件，即可直接运行。

```
g++ -o trans_producer_demo -std=c++11 -lz -lrocketmq TransProducerDemo.cpp
```

2. 发送事务消息的示例代码如下。

```
#include <iostream>
#include <chrono>
#include <thread>
#include "TransactionMQProducer.h"
#include "MQClientException.h"
#include "TransactionListener.h"
using namespace std;
using namespace rocketmq;
class ExampleTransactionListener : public TransactionListener {
public:
    LocalTransactionState executeLocalTransaction(const MQMessage &msg, void *arg) {
        //执行本地事务，成功返回COMMIT_MESSAGE，失败返回ROLLBACK_MESSAGE，不确定返回UNKNOWN。
        //如果返回UNKNOWN，则会触发定时任务回查状态。
        std::cout << "Execute Local Transaction,Received Message Topic:" << msg.getTopic() << ", MsgId:"
                  << msg.getBody() << std::endl;
        return UNKNOWN;
    }
    LocalTransactionState checkLocalTransaction(const MQMessageExt &msg) {
        //回查本地事务执行情况，成功返回COMMIT_MESSAGE，失败返回ROLLBACK_MESSAGE，不确定返回UNKNOWN。
        //如果返回UNKNOWN，则等待下次定时任务回查。
        std::cout << "Check Local Transaction,Received Message Topic:" << msg.getTopic() << ", MsgId:" << msg.getMsgId()
                  << std::endl;
```

```
        return COMMIT_MESSAGE;
    }
};
int main() {
    std::cout << "=======Before sending messages=======" << std::endl;
    //您在阿里云消息队列RocketMQ版控制台上申请的GID。
    TransactionMQProducer producer("GID_XXXXXXXXXXXXXXX");
    //设置TCP协议接入点，从阿里云消息队列RocketMQ版控制台的实例详情页面获取。
    producer.setNamesrvAddr("http://MQ_XXXXXXXXXXX.mq-internet-access.mq-internet.aliyuncs.com:80");
    //您在阿里云账号管理控制台中创建的AccessKey ID和AccessKey Secret，用于身份认证，用户渠道，默认值为：ALIYUN。
    producer.setSessionCredentials("AK", "SK", "ALIYUN");
    //本地事务执行和回查函数。
    ExampleTransactionListener *exampleTransactionListener = new ExampleTransactionListener();
    producer.setTransactionListener(exampleTransactionListener);
    //请确保参数设置完成之后启动Producer。
    producer.start();
    auto start = std::chrono::system_clock::now();
    int count = 3;
    for (int i = 0; i < count; ++i) {
        //发送消息时请设置您在阿里云消息队列RocketMQ版控制台上申请的Topic。
        MQMessage msg("YOUR TRANSACTION TOPIC", "HiTAG", "Hello,CPP SDK, Transaction Message.");
        try {
            SendResult sendResult = producer.sendMessageInTransaction(msg, &i);
            std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID:" << sendResult.getMsgId()
                      << std::endl;
            this_thread::sleep_for(chrono::seconds(1));
        } catch (MQException e) {
            std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() << std::endl;
        }
    }
    auto interval = std::chrono::system_clock::now() - start;
    std::cout << "Send " << count << " messages OK, costs "
              << std::chrono::duration_cast<std::chrono::milliseconds>(interval).count() << "ms" << std::endl;
    std::cout << "Wait for local transaction check..... " << std::endl;
    for (int i = 0; i < 6; ++i) {
        this_thread::sleep_for(chrono::seconds(10));
        std::cout << "Running "<< i*10 + 10 << " Seconds......"<< std::endl;
    }
    producer.shutdown();
    std::cout << "=======After sending messages=======" << std::endl;
    return 0;
}
```

消费事务消息

1. 将以下代码拷贝到*ConsumerDemo.cpp*文件中，修改相应的参数后，使用g++命令进行编译，生成可执行文件，即可直接运行。

```
g++ -o consumer_demo -std=c++11 -lz -lrocketmq ConsumerDemo.cpp
```

2. 消费事务消息的示例代码如下。

```
#include <iostream>
#include <thread>
#include "DefaultMQPushConsumer.h"
using namespace rocketmq;
class ExampleMessageListener : public MessageListenerConcurrently {
public:
    ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
        for (auto item = msgs.begin(); item != msgs.end(); item++) {
            std::cout << "Received Message Topic:" << item->getTopic() << ", MsgId:" <<
item->getMsgId() << std::endl;
        }
        return CONSUME_SUCCESS;
    }
};
int main(int argc, char *argv[]) {
    std::cout << "=======Before consuming messages=======" << std::endl;
    //您在阿里云消息队列RocketMQ控制台上申请的GID。
    DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer("GID_XXXXXXXX");
    //设置TCP协议接入点，从阿里云消息队列RocketMQ控制台的实例详情页面获取。
    consumer->setNamesrvAddr("http://MQ_INST_XXXXXXXXXXX.mq-internet-access.mq-internet
.aliyuncs.com:80");
    //您在阿里云账号管理控制台中创建的AccessKey ID和AccessKey Secret，用于身份认证，用户渠道，
默认值为：ALIYUN。
    consumer->setSessionCredentials("AK", "SK", "ALIYUN");
    auto start = std::chrono::system_clock::now();
    //register your own listener here to handle the messages received.
    //请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
    ExampleMessageListener *messageListener = new ExampleMessageListener();
    consumer->subscribe("YOURTOPIC", "*");
    consumer->registerMessageListener(messageListener);
    //Start this consumer
    //准备工作完成，必须调用启动函数，才可以正常工作。
    // ******************************************
    // 1.确保订阅关系的设置在启动之前完成。
    // 2.确保相同GID下面的消费者的订阅关系一致。
    // ******************************************
    consumer->start();
    //Keep main thread running until process finished.
    //请保持线程常驻，不要执行shutdown操作。
    std::this_thread::sleep_for(std::chrono::milliseconds(60 * 1000));
    consumer->shutdown();
    std::cout << "=======After consuming messages======" << std::endl;
    return 0;
}
```

4.3.7. 收发定时和延时消息

本文提供使用TCP协议下的社区版C++ SDK来收发定时和延时消息的示例代码供您参考。

背景信息

- 定时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。
- 延时消息：Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。

更多信息，请参见定时和延时消息。

> **注意** 社区版的Apache RocketMQ和阿里云消息队列RocketMQ版配置方式不同，实现效果也有所差异。社区版的Apache RocketMQ支持延时消息，但不支持定时消息，因此没有专门的定时消息接口。阿里云消息队列RocketMQ版不仅同时支持配置延时消息和定时消息，并且定时和延时时间可以精确到秒级、拥有更高的并发性。建议您优先使用云上定时延时的方式，使用方法，请参考以下步骤。

前提条件

安装CPP动态库

发送定时消息

1. 将以下代码拷贝到*DelayProducerDemo.cpp*文件中，修改相应的参数后，使用**g++**命令进行编译，生成可执行文件，即可直接运行。
   ```
   g++ -o delay_producer_demo -std=c++11 -lz -lrocketmq DelayProducerDemo.cpp
   ```
2. 发送定时消息的示例代码如下。
   ```
   #include <iostream>
   #include <chrono>
   #include <thread>
   #include "DefaultMQProducer.h"
   using namespace std;
   using namespace rocketmq;
   int main() {
       std::cout << "=======Before sending messages=======" << std::endl;
       //您在阿里云消息队列RocketMQ版上申请的GID。
       DefaultMQProducer producer("GID_XXXXXXXXXXX");
       //设置TCP协议接入点，从阿里云消息队列RocketMQ版控制台的实例详情页面获取。
       producer.setNamesrvAddr("http://MQ_INST_XXXXXXX.mq-internet-access.mq-internet.aliyuncs.com:80");
       //您在阿里云账号管理控制台中创建的AccessKey ID和AccessKey Secret，用于身份认证；用户渠道，默认值为：ALIYUN。
       producer.setSessionCredentials("AK", "SK", "ALIYUN");
       //请确保参数设置完成之后启动Producer。
       producer.start();
       auto start = std::chrono::system_clock::now();
       int count = 32;
       for (int i = 0; i < count; ++i) {
           //发送消息时请设置您在阿里云
   ```

```
消息队列RocketMQ版控制台上申请的Topic。
        MQMessage msg("YOUR DELAY TOPIC", "HiTAG", "Hello,CPP SDK, Delay Message.");
        chrono::system_clock::duration d = chrono::system_clock::now().time_since_epoch
();
        chrono::milliseconds mil = chrono::duration_cast<chrono::milliseconds>(d);
        //定时延时消息，单位毫秒（ms），在指定时间戳（当前时间之后）进行投递，例如2020-03-07 16:
21:00投递。
        //如果被设置成当前时间戳之前的某个时刻，消息将立刻投递给消费者。
        //设置需要延时或者定时的时间，例如当前时间延迟10秒后投递。
        long exp = mil.count() + 10000;
        msg.setProperty("__STARTDELIVERTIME", to_string(exp));
        std::cout << "Now: " << mil.count() << " Exp:" << exp << std::endl;
        try {
            SendResult sendResult = producer.send(msg);
            std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID:
" << sendResult.getMsgId()
                      << std::endl;
            this_thread::sleep_for(chrono::seconds(1));
        } catch (MQException e) {
            std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() <<
std::endl;
        }
    }
    auto interval = std::chrono::system_clock::now() - start;
    std::cout << "Send " << count << " messages OK, costs "
              << std::chrono::duration_cast<std::chrono::milliseconds>(interval).count(
) << "ms" << std::endl;
    producer.shutdown();
    std::cout << "=======After sending messages=======" << std::endl;
    return 0;
}
```

消费定时消息

1. 将以下代码拷贝到*DelayConsumerDemo.cpp*文件中，修改相应的参数后，使用**g++**命令进行编译，生成可执行文件，即可直接运行。

   ```
   g++ -o delay_consumer_demo -std=c++11 -lz -lrocketmq DelayConsumerDemo.cpp
   ```

2. 消费定时消息的示例代码如下。

```
#include <iostream>
#include <thread>
#include <chrono>
#include "DefaultMQPushConsumer.h"
using namespace rocketmq;
using namespace std;
class ExampleDelayMessageListener : public MessageListenerConcurrently {
public:
    ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
        for (auto item = msgs.begin(); item != msgs.end(); item++) {
            chrono::system_clock::duration d = chrono::system_clock::now().time_since_e
poch();
            chrono::milliseconds mil = chrono::duration_cast<chrono::milliseconds>(d);
            std::cout << "Now: " << mil.count() << " Received Message Topic:" << item->
getTopic() << ", MsgId:"
                      << item->getMsgId() << " DelayTime:" << item->getProperty("__STAR
TDELIVERTIME") << std::endl;
        }
        return CONSUME_SUCCESS;
    }
};
int main(int argc, char *argv[]) {
    std::cout << "=======Before consuming messages=======" << std::endl;
    //您在阿里云
消息队列RocketMQ版控制台上申请的GID。
    DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer("GID_XXXXXXXXXX");
    //设置TCP协议接入点，从阿里云
消息队列RocketMQ版控制台的实例详情页面获取。
    consumer->setNamesrvAddr("http://MQ_INST_XXXXXXXX.mq-internet-access.mq-internet.al
iyuncs.com:80");
    //您在阿里云账号管理控制台中创建的AccessKeyId和AccessKeySecret，用于身份认证，用户渠道，默
认值为：ALIYUN。
    consumer->setSessionCredentials("AK", "SK", "ALIYUN");
    auto start = std::chrono::system_clock::now();
    //register your own listener here to handle the messages received.
    //请注册自定义侦听函数用来处理接收到的消息，并返回响应的处理结果。
    ExampleDelayMessageListener *messageListener = new ExampleDelayMessageListener();
    consumer->subscribe("YOUR DELAY TOPIC", "*");
    consumer->registerMessageListener(messageListener);
    //Start this consumer
    //准备工作完成，必须调用启动函数，才可以正常工作。
    // ******************************************
    // 1.确保订阅关系的设置在启动之前完成。
    // 2.确保相同GID下面的消费者的订阅关系一致。
    // ******************************************
    consumer->start();
    //Keep main thread running until process finished.
    //请保持线程常驻，不要执行shutdown操作。
    std::this_thread::sleep_for(std::chrono::seconds(600));
    consumer->shutdown();
    std::cout << "=======After consuming messages======" << std::endl;
    return 0;
}
```

5.订阅消息常见问题

5.1. 消息负载均衡策略

消息队列RocketMQ版
的消息负载均衡策略针对生产者和消费者有所差异。对消费者而言，消息负载均衡策略在一定程度上影响消息堆积。

背景信息

随着SDK版本的升级，

消息队列RocketMQ版
的负载均衡策略也有所优化，根据SDK版本，负载均衡策略可分为：

- 负载均衡策略（客户端Java v2.x.x.Final和C++ v3.x.x）
- 负载均衡策略（客户端Java v1.x.x.Final和C++ v1.x.x/v2.x.x）

负载均衡策略（客户端Java v2.x.x.Final和C++ v3.x.x）

前提条件

- **实例所属地域**
  - 西南1（成都）
  - 华北1（青岛）
  - 华南1（深圳）
- **SDK版本**
  - TCP协议Java SDK：升级至2.x.x.Final版本
  - TCP协议C++ SDK：升级至3.x.x版本

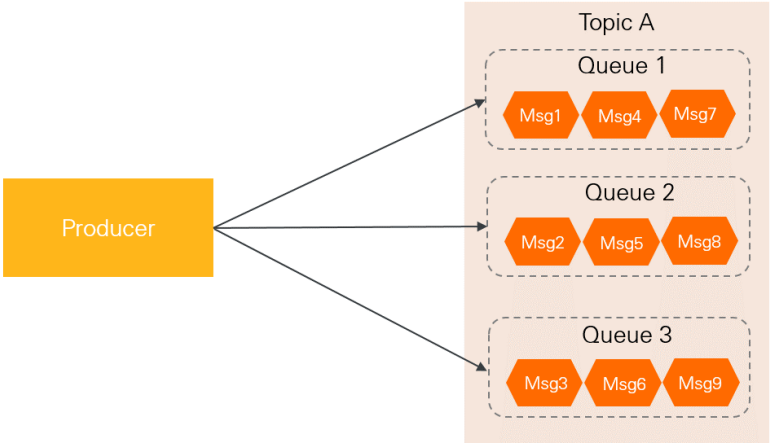
优势

客户端Java v2.x.x.Final和C++ v3.x.x版本的负载均衡策略有以下优势：

- 消息消费更加灵活，不再按照Queue维度消费，同一Queue的消息可以被多个消费者消费。
- 当消费者个数多于Queue的个数时，也不会产生消费者空转的现象，避免资源浪费。

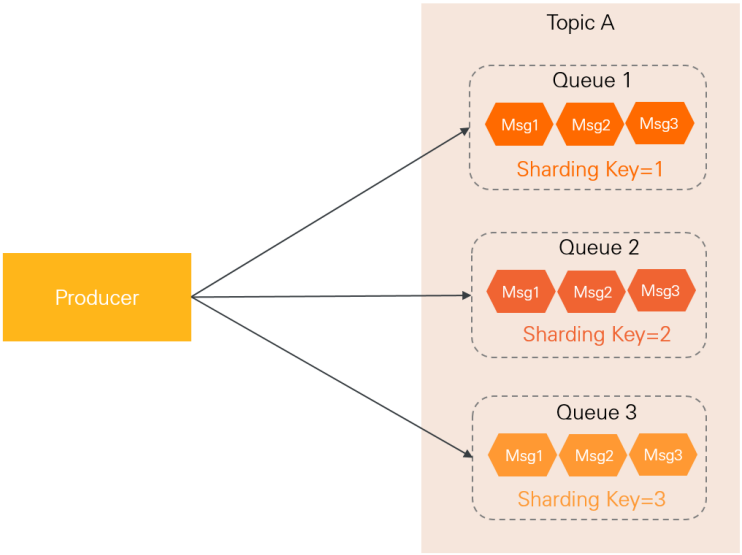
生产者负载均衡策略

- 无序消息（普通消息、事务消息、定时和延时消息）：Producer将消息以轮询的方式发送至Queue，如下图所示：



图中Msg1、Msg2代表第一条消息、第二条消息，生产者会把第一条消息发送至Queue 1，然后第二条消息发送至Queue 2，第三条消息发送至Queue 3，以此类推，第四条消息又发送至Queue 1，循环往复。

- 顺序消息：相同Sharding Key的消息被发送至同一个Queue，如下图所示：

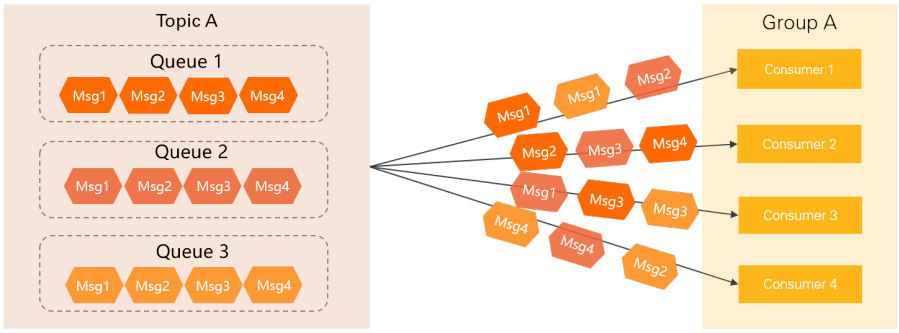


图中ShardingKey为1的消息都同时被发送到Queue1中，同样Sharding Key为2的消息同时被发送至Queue 2中。

消费者负载均衡策略

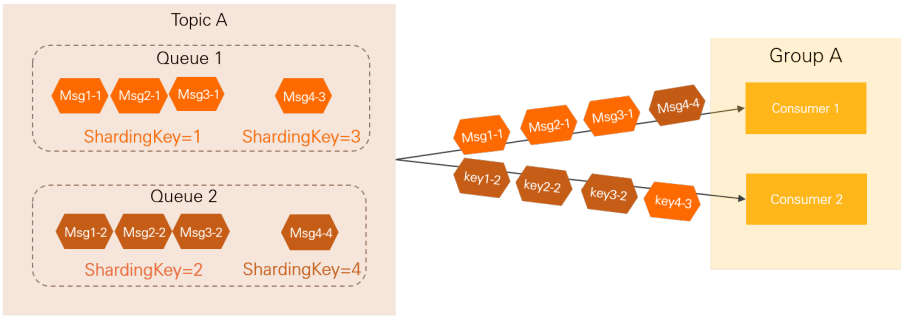
- 无序消息：
  消息队列RocketMQ版
  Broker按照消息维度将Topic中的所有消息平均分配给消费者集群中的多个消费者处理。

客户端Java v2.x.x.Final和C++ v3.x.x版本的负载均衡不再按照Queue维度消费，同一Queue的消息可以被多个消费者消费。具体示例如下图所示：



图中每一个Queue中的多条消息都可同时被不同的消费者消费，例如Queue 1中的四条消息被分配给了Consumer 1、Consumer 2、Consumer 3及Consumer 4消费。

- 顺序消息：消息消费根据Sharding Key负载均衡，不同Sharding Key的消息可被均衡的负载到多个Queue中消费，且为保证顺序消息的语义，相同Sharding Key的消息会被同一个消费者消费，具体示例如下图所示：



> **说明**
>
> - Msg2-1：该图标背景色表示消息属于Queue 1；Msg2-1表示该消息为Queue1中的第2条消息，且消息的ShardingKey为1。
> - Msg3-2：该图标背景色表示消息属于Queue 2；Msg3-2表示该消息为Queue2中的第3条消息，且消息的ShardingKey为2。

图中Msg1-1、Msg2-1和Msg3-1这三条消息的Sharding Key都是1，因此被同一个消费者Consumer 1消费，同样Sharding Key都为2的消息同时被Consumer 2消费。
Msg4-3和Msg4-4的Sharding Key不同，分别为3和4，因此可被分配给不同的消费者消费。

负载均衡策略（客户端Java v1.x.x.Final和C++ v1.x.x/v2.x.x）

生产者负载均衡策略

与客户端Java v2.x.x.Final和C++ v3.x.x的负载均衡策略相同，具体示例，请参见负载均衡策略（客户端Java v2.x.x.Final和C++ v3.x.x）。

- 无序消息：Producer的消息以轮询的方式发送至Queue。
- 顺序消息：相同Sharding Key的消息被发送至同一个Queue。

消费者负载均衡策略

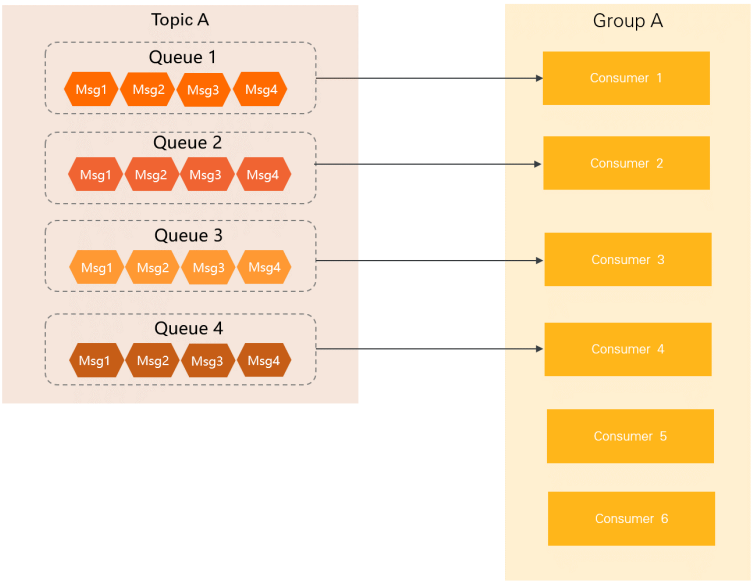
> **说明** 无序消息和顺序消息的负载均衡策略相同。

消息队列RocketMQ版
包含Broker和Name Server等节点，其中Broker节点负责将Topic的路由信息上报至Name Server节点。
假设目前

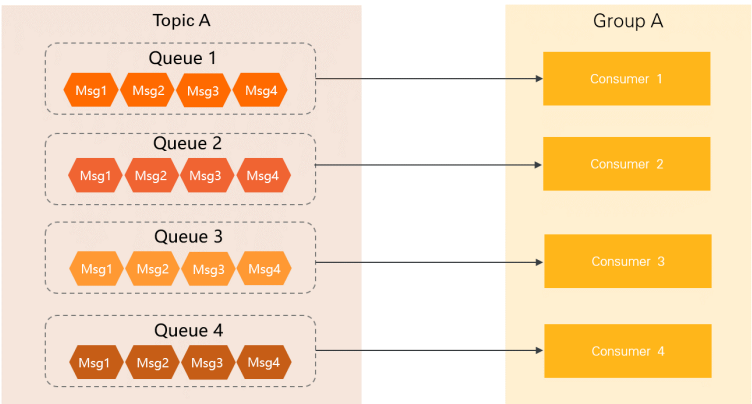
消息队列RocketMQ版
只有一个Broker节点，消息从Producer发送至
消息队列RocketMQ版
的Topic，默认会将这些Topic下的消息均衡负载至8个Queue（逻辑概念）。
消息队列RocketMQ版
Broker会将这些Queue再平均分配至属于同一个Group ID的消费者集群。
在

消息队列RocketMQ版
中，一个Queue只能同时被分配至一个消费者消费。因此，每台消费者机器处理的Queue的数量有以下几种可能：

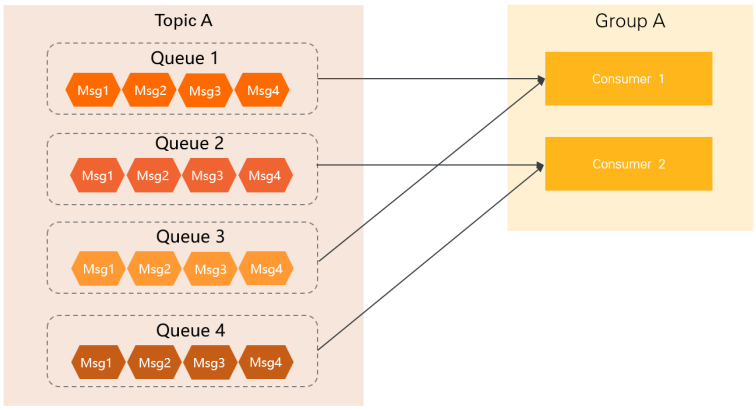
- 若消费者机器数量大于Queue的数量，则超出Queue数量的机器会处理0个Queue上的消息，如下图所示：



- 若消费者机器数量等于Queue的数量，则每台机器会处理1个Queue上的消息，如下图所示：



- 若消费者机器数量小于Queue的数量，则每台机器会处理多个Queue上的消息，如下图所示：



因为消费者负载均衡策略是以Queue为粒度维护，如果消费者集群中一台机器处理变慢，可能是机器硬件、系统、远程RPC调用或Java GC等原因导致分配至此机器上的Queue的消息不能及时处理，则整个Queue上的消息都会堆积。

5.2. 消息队列RocketMQ版客户端如何设置消费线程数？

消息队列RocketMQ版

所提供的TCP Java SDK支持多线程消费，且适用于所有消息类型，本文介绍如何设置消费线程数的方法。

在启动Consumer时，设置一个ConsumeThreadNums属性即可。具体示例如下所示。

```
public static void main(String[] args) {
    Properties properties = new Properties();
    properties.put(PropertyKeyConst.GROUP_ID, "GID_001");
    properties.put(PropertyKeyConst.AccessKey, "xxxxxxxxxxx");
    properties.put(PropertyKeyConst.SecretKey, "xxxxxxxxxxx");
    /**
     * 设置消费端线程数固定为20
     */
    properties.setProperty(PropertyKeyConst.ConsumeThreadNums,"20");
    Consumer consumer =ONSFactory.createConsumer(properties);
    consumer.subscribe("TestTopic", "*", new MessageListener() {
        public Action consume(Message message, ConsumeContext context) {
            System.out.println("Receive: " + message);
            return Action.CommitMessage;
        }
    });
    consumer.start();
    System.out.println("Consumer Started");
}
```

5.3. 发送消息时返回“MQClientException: No route info of this topic”错误

问题现象

使用TCP协议SDK发送消息时，
消息队列RocketMQ版
服务端返回如下错误：

```
Caused by: com.aliyun.openservices.shade.com.alibaba.rocketmq.client.exception.MQClientException: No route info of this topic
```

可能原因

- 代码中设置的接入点和
  消息队列RocketMQ版
  控制台上提供的不一致。
- 代码中设置的Topic名称和已创建的Topic的名称不一致。
- SDK版本不匹配。针对有命名空间的实例，使用的SDK版本必须大于1.7.9.Final。若实例有命名空间，且错误信息后没有 `{instancId}%{topic}` 内容，说明使用的SDK版本不正确。

解决方案

操作步骤

1. 登录
   消息队列RocketMQ版
   控制台。
2. 在**实例详情**页面的**接入点**页签查看实例的接入点，检查代码中设置的接入点是否和控制台提供的一致。
3. 在**Topic 管理**页面查看代码中设置的Topic是否已创建且拼写正确。
4. 在**实例详情**页面的**基础信息**区域查看实例是否有命名空间。若实例有命名空间，且错误信息中没有 `{instanceId}%{topic}` ，说明SDK版本不正确，请确保使用的SDK版本大于1.7.9.Final。

   如果您的问题还未解决，请提交工单联系技术支持。

5.4. .NET SDK不接受byte[]类型，无法兼容Protobuf序列化怎么处理？

您可以使用不带body的构造函数，然后通过以下接口设置body，即可兼容Protobuf序列化出来的ByteString类型。

```
public void setBody(byte[] byte_msgbody, int len)
```