消息队列 MQ

TCP 接入(专业)

TCP 接入(专业)

Java SDK

环境准备

运行本节描述的 Java 代码之前，请按以下说明准备好环境。

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>
<version>1.2.5</version>
</dependency>
```

下载依赖 Jar 包：

下载链接

代码里涉及到的 Topic， Producer ID， Consumer ID，需要到 MQ 控制台上创建。 Message Tag 可以完全由应用自定义，具体创建过程可参考 申请 MQ 资源。

使用 MQ 服务的应用程序需要部署在 ECS 上。

MQ 日志配置

本文档主要介绍 MQ 客户端日志的正常打印方式，MQ 客户端日志格式解析以及如何自定义 MQ 客户端日志配置。

打印 MQ 客户端日志

MQ 客户端日志在问题定位排查中扮演着非常重要的角色，通过日志记录客户端运行过程中的异常，能够帮助尽可能真实的还原某个时间点的异常场景，最终达到快速定位、修复 Bug 的目的。

TCP Java SDK 打印 MQ 客户端日志

MQ 的 TCP Java SDK 基于 SLF4J 接口编程，客户端日志的打印依赖用户在配置文件中指定的日志实现。目前支持 log4j（暂不支持log4j2）、logback，可在 pom.xml 或者 lib 中添加对应的日志实现依赖即可：

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。确保应用只依赖其中一个日志实现，是正确打印 MQ 客户端日志的前提条件，建议通过 mvn clean dependency:tree | grep log 命令排查。

MQ 客户端日志默认配置

在应用中添加了唯一的日志实现后启动 MQ 客户端，MQ 客户端将会按照如下的配置生成日志文件：

- 日志保存路径：/{user.home}/logs/ons.log，其中{user.home}是指启动当前 Java 进程的用户的根

目录
- 单个日志文件大小：64MB
- 保存历史日志文件的最大个数：10个
- 日志级别：INFO

自定义日志配置

MQ 客户端支持用户自定义 **日志保存路径**、**日志级别**以及**保存历史日志文件的最大个数**；考虑到日志传输以及阅读的便利性，暂不允许自定义**单个日志文件大小**，仍保持默认64MB；可通过配置**保存历史日志文件的最大个数**来自定义日志文件保存的时间范围。

各个参数配置说明如下：

- 日志保存路径：请确保应用进程有对该路径写的权限，否则日志不会打印。
- 保存历史日志文件的最大个数：支持1到100之前的数值，超出范围或者格式错误默认保存10个。
- 日志级别：支持 ERROR、WARN、INFO、DEBUG 中任何一种，不匹配默认 INFO。

TCP Java SDK 自定义 MQ 客户端日志

自定义 MQ 客户端日志配置，请升级 TCP Java SDK 版本到1.2.5及以上。

在 TCP Java SDK 中自定义 MQ 客户端日志配置，请设置如下系统参数：

- ons.client.logRoot ：日志保存路径
- ons.client.logFileMaxIndex ：保存历史日志文件的最大个数
- ons.client.logLevel ：日志级别

举例说明，可在启动脚本中或者 IDE 的 VM options 中添加如下系统参数：

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

发送普通消息（三种方式）

MQ 发送普通消息有三种实现方式：可靠同步发送、可靠异步发送、单向(Oneway)发送。本文介绍了每种实现的原理、使用场景以及三种实现的异同，同时提供了代码示例以供参考。

可靠同步发送

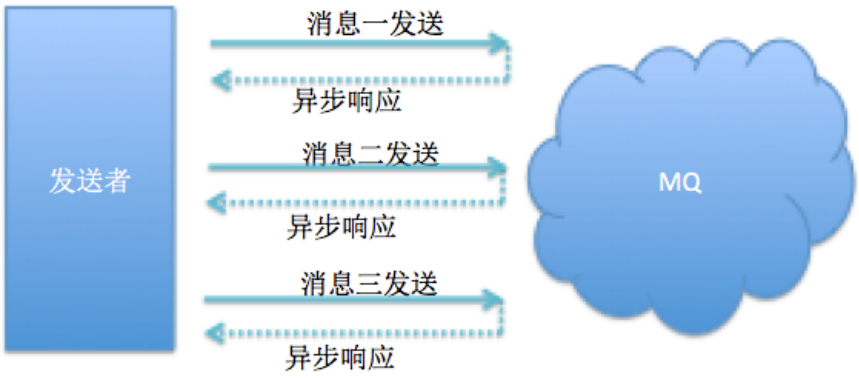
原理：同步发送是指消息发送方发出数据后，会在收到接收方发回响应之后才发下一个数据包的通讯

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。

可靠异步发送

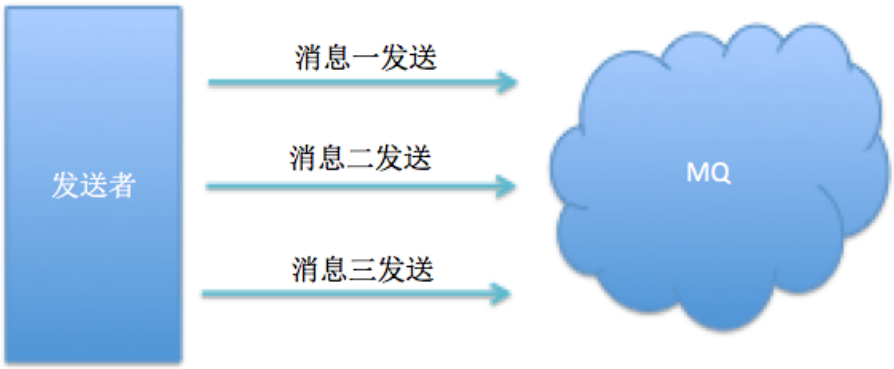
原理：异步发送是指发送方发出数据后，不等接收方发回响应，接着发送下个数据包的通讯方式。MQ 的异步发送，需要用户实现异步发送回调接口（SendCallback），在执行消息的异步发送时，应用不需要等待服务器响应即可直接返回，通过回调接口接收务器响应，并对服务器的响应结果进行处理。



应用场景：异步发送一般用于链路耗时较长，对 RT 响应时间较为敏感的业务场景，例如用户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。

单向（Oneway）发送

原理：单向（Oneway）发送特点为只负责发送消息，不等待服务器回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。

下表概括了三者的特点和主要区别。

	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

示例代码

同步发送

```
public class ProducerTest {
public static void main(String[] args) {
Properties properties = new Properties();
properties.put(PropertyKeyConst.ProducerId, "XXX");//您在控制台创建的Producer ID
properties.put(PropertyKeyConst.AccessKey,"XXX");// AccessKey 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");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");//设置发送超时时间，单位毫秒
//公有云生产环境：http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//公有云公测环境：http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet
//杭州金融云环境：http://jbponsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//杭州深圳云环境：http://mq4finance-sz.addr.aliyun.com:8080/rocketmq/nsaddr4client-internal
properties.put(PropertyKeyConst.ONSAddr,
"http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公有云生产环境为例
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( //
// Message Topic
"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MQ服务器过滤
"TagA",
```

```
// Message Body 可以是任何二进制形式的数据， MQ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，可通过阿里云服务器管理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);

// 同步发送消息，只要不抛异常就是成功
SendResult sendResult = producer.send(msg);
System.out.println(sendResult);
}

// 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

```
    public static void main(String[] args) {
Properties properties = new Properties();
properties.put(PropertyKeyConst.AccessKey, "DEMO_AK");// AccessKey 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, "DEMO_SK");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
properties.put(PropertyKeyConst.ProducerId, "DEMO_PID");//您在控制台创建的Producer ID
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");//设置发送超时时间，单位毫秒

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 Topic
"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MQ服务器过滤
"TagA",
// Message Body，任何二进制形式的数据，MQ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MQ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) {
// 消费发送成功
System.out.println("send message success. topic=" + sendResult.getTopic() + ", msgId=" +
sendResult.getMessageId());
```

```
}

@Override
public void onException(OnExceptionContext context) {
// 消息发送失败
System.out.println("send message failed. topic=" + context.getTopic() + ", msgId=" + context.getMessageId());
}
});

// 在callback返回之前即可取得msgId。
System.out.println("send message async. topic=" + msg.getTopic() + ", msgId=" + msg.getMsgID());

// 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();
}
```

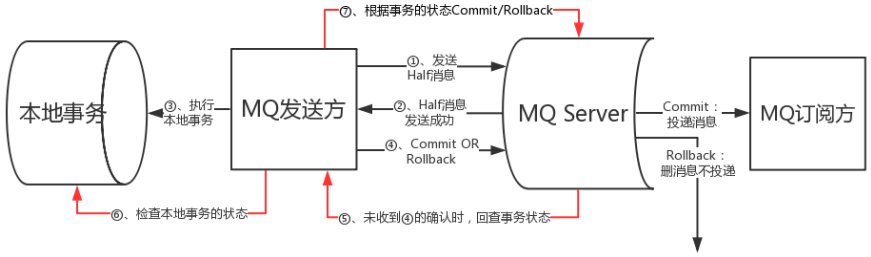
单向(Oneway)发送

```
    public static void main(String[] args) {
Properties properties = new Properties();
properties.put(PropertyKeyConst.AccessKey, "DEMO_AK");// AccessKey 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, "DEMO_SK");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
properties.put(PropertyKeyConst.ProducerId, "DEMO_PID");//您在控制台创建的Producer ID
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");//设置发送超时时间，单位毫秒
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(
// Message Topic
"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MQ服务器过滤
"TagA",
// Message Body
// 任何二进制形式的数据，MQ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，可通过阿里云服务器管理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);

// oneway发送消息，只要不抛异常就是成功
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();
}
```

发送分布式事务消息

目前支持的域包括公网测试、华东1、华北2、华东2、华南1。

MQ事务消息交互流程如下：



发送事务消息包含以下两个步骤：

发送半消息及执行本地事务

```
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(); // 用户需要设置自己的log, 记录日志便于排查问题

public static void main(String[] args) throws InterruptedException {
final BusinessService businessService = new BusinessService(); // 本地业务Service
Properties properties = new Properties();
properties.put(PropertyKeyConst.ProducerId, ""); // 您在控制台创建的Producer ID
properties.put(PropertyKeyConst.AccessKey, ""); // 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, ""); // 阿里云身份验证，在阿里云服务器管理控制台创建

//公有云生产环境：http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//公有云公测环境：http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet
//杭州金融云环境：http://jbponsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//杭州深圳云环境：http://mq4finance-sz.addr.aliyun.com:8080/rocketmq/nsaddr4client-internal
properties.put(PropertyKeyConst.ONSAddr,
"http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公有云生产环境为例
TransactionProducer producer = ONSFactory.createTransactionProducer(properties,
new LocalTransactionCheckerImpl());
producer.start();
```

```
Message msg = new Message("Topic", "TagA", "Hello MQ transaction===".getBytes());
// 输入您在控制台创建的Topic
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ID在控制台无法查询)
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body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) {
log.error("Message Id:{}", msgId, e);
}
System.out.println(msg.getMsgID());
log.warn("Message Id:{}transactionStatus:{}", msgId, transactionStatus.name());
return transactionStatus;
}
}, null);
// demo example 防止进程退出(实际使用不需要这样)
TimeUnit.MILLISECONDS.sleep(Integer.MAX_VALUE);
}
}
```

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MQ Broker向发送方再次询问该消息对应的本地事务的状态

```
import com.alibaba.rocketmq.client.producer.LocalTransactionState;
```

```
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Half 消息，所以消息ID在控制台无法查询)
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、消息本 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, 实际需要用户根据情况来处理
Object businessServiceArgs = new Object();
TransactionStatus transactionStatus = TransactionStatus.Unknow;
try {
boolean isCommit = businessService.checkbusinessService(businessServiceArgs);
if (isCommit) {
//本地事务已成功、提交消息
transactionStatus = TransactionStatus.CommitTransaction;
} else {
//本地事务已失败、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Half消息发送完成，但本地事务返回状态为TransactionStatus.Unknow时，或者应用退出导致本地事务未提交任何状态时，从MQ Broker的角度看，这条Half状态的消息的状态是未知的，因此MQ 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？

MQ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。MQ发送事务消息时需要实现LocalTransactionChecker接口，用来处理MQ 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：

(1) 检查该Half消息对应的本地事务的状态（commited or rollback）

(2) 向MQ Broker提交该Half消息本地事务的状态

发送延时消息

目前支持的域包括公网测试、华东1、华北2、华东2、华南1。MQ 客户端请使用最新版本1.2.2。

延时消息用于指定消息发送到MQ服务器端后，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。

代码示例

```
public class ProducerDelayTest {
public static void main(String[] args) {
Properties properties = new Properties();
properties.put(PropertyKeyConst.ProducerId, "XXX");// 您在控制台创建的 Producer ID
properties.put(PropertyKeyConst.AccessKey, "XXX");// AccessKey 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");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

//公有云生产环境：http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//公有云公测环境：http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet
//杭州金融云环境：http://jbponsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//杭州深圳云环境：http://mq4finance-sz.addr.aliyun.com:8080/rocketmq/nsaddr4client-internal
properties.put(PropertyKeyConst.ONSAddr,
"http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公有云生产环境为例
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 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MQ服务器过滤
"tag",
// Message Body 可以是任何二进制形式的数据， MQ 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 MQ 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");
// 延时时间单位为毫秒（ms），指定一个时刻，在这个时刻之后才能被消费，这个例子表示 3秒 后才能被消费
long delayTime = 3000;
```

```
msg.setStartDeliverTime(System.currentTimeMillis() + delayTime);
// 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());
// 在应用退出前，销毁Producer对象<br>
// 注意：如果不销毁也没有问题
producer.shutdown();
}
}
```

发送定时消息

目前支持的域包括公网测试、华东1、华北2、华东2、华南1。

定时消息可以做到在指定时间戳之后才可被消费者消费，用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发定时任务的场景。

代码示例

```
public class ProducerDelayTest {
public static void main(String[] args) {
Properties properties = new Properties();
properties.put(PropertyKeyConst.ProducerId, "XXX");//您在 MQ 控制台创建的Producer ID
properties.put(PropertyKeyConst.AccessKey, "XXX");// 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");// 阿里云身份验证，在阿里云服务器管理控制台创建

//公有云生产环境：http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//公有云公测环境：http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet
//杭州金融云环境：http://jbponsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//杭州深圳云环境：http://mq4finance-sz.addr.aliyun.com:8080/rocketmq/nsaddr4client-internal
properties.put(PropertyKeyConst.ONSAddr,
"http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公有云生产环境为例
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 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MQ服务器过滤
"tag",
// Message Body 可以是任何二进制形式的数据， MQ 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 MQ 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");
/**
* 定时消息投递，设置投递的具体时间戳，单位毫秒例如2016-03-07 16:21:00投递
*/
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());
// 在应用退出前，销毁 Producer 对象<br>
// 注意：如果不销毁也没有问题
producer.shutdown();
}
}
```

集群方式订阅消息

集群订阅即某个消费者集群只消费指定的 Topic，而不是消费所有 Topic。

```
public class ConsumerTest {
public static void main(String[] args) {
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, "XXX");// 您在控制台创建的 Consumer ID
properties.put(PropertyKeyConst.AccessKey, "XXX");// AccessKey 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");// SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

//公有云生产环境：http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//公有云公测环境：http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet
//杭州金融云环境：http://jbponsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//杭州深圳云环境：http://mq4finance-sz.addr.aliyun.com:8080/rocketmq/nsaddr4client-internal
properties.put(PropertyKeyConst.ONSAddr,
"http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公有云生产环境为例
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicTestMQ", "*", new MessageListener() {
public Action consume(Message message, ConsumeContext context) {
System.out.println("Receive: " + message);
return Action.CommitMessage;
}
});
consumer.start();
System.out.println("Consumer Started");
}
}
```

Spring 集成

本文介绍如何在 Spring 框架下用 MQ 收发消息。主要包括三部分内容：普通消息生产者和 Spring 集成，事务消息生产者和 Spring 集成， 消息消费者与 Spring 集成。

生产者与 Spring 集成

1.在 producer.xml 中定义生产者 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.springframework.org/schema/beans/spring-beans.xsd">

<bean id="producer" class="com.aliyun.openservices.ons.api.bean.ProducerBean" init-method="start" destroy-
method="shutdown">
<property name="properties" > <!--生产者配置信息-->
<props>
<prop key="ProducerId">PID_DEMO</prop> <!--请替换XXX-->
<prop key="AccessKey">XXX</prop>
<prop key="SecretKey">XXX</prop>
</props>
</property>
</bean>

</beans>
```

2.通过已经与 Spring 集成好的生产者生产消息。

```
package demo;

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.exception.ONSClientException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ProduceWithSpring {
public static void main(String[] args) {
/**
* 生产者Bean配置在producer.xml中,可通过ApplicationContext获取或者直接注入到其他类(比如具体的Controller)中.
*/
ApplicationContext context = new ClassPathXmlApplicationContext("producer.xml");

Producer producer = (Producer) context.getBean("producer");
//循环发送消息
for (int i = 0; i < 100; i++) {
Message msg = new Message( //
// Message Topic
"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MQ服务器过滤
"TagA",
// Message Body 可以是任何二进制形式的数据， MQ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MQ 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，只要不抛异常就是成功
try {
```

```
SendResult sendResult = producer.send(msg);
assert sendResult != null;
System.out.println("send success: " + sendResult.getMessageId());
}catch (ONSClientException e) {
System.out.println("发送失败");
}

}
}
}
```

事务消息生产者与 Spring 集成

有关事务消息的概念请查看发送事务消息。

1.首先需要实现一个 LocalTransactionChecker，如下所示。

```
package demo;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionChecker;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;

public class DemoLocalTransactionChecker implements LocalTransactionChecker {
public TransactionStatus check(Message msg) {
System.out.println("开始回查本地事务状态");
return TransactionStatus.CommitTransaction; //根据本地事务状态检查结果返回不同的TransactionStatus
}
}
```

2.其次，在 transactionProducer.xml 中定义事务消息生产者 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.springframework.org/schema/beans/spring-beans.xsd">

<bean id="localTransactionChecker" class="demo.DemoLocalTransactionChecker"></bean>

<bean id="transactionProducer" class="com.aliyun.openservices.ons.api.bean.TransactionProducerBean" init-
method="start" destroy-method="shutdown">
<property name="properties" > <!--事务消息生产者配置信息-->
<props>
<prop key="ProducerId">PID_DEMO</prop> <!--请替换XXX-->
<prop key="AccessKey">AKDEMO</prop>
<prop key="SecretKey">SKDEMO</prop>
</props>
</property>
<property name="localTransactionChecker" ref="localTransactionChecker"></property>
</bean>

</beans>
```

3.通过已经与 Spring 集成好的生产者生产事务消息。

```
package demo;

import com.aliyun.openservices.ons.api.Message;
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 org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ProduceTransMsgWithSpring {

public static void main(String[] args) {
/**
* 事务消息生产者Bean配置在transactionProducer.xml中,可通过ApplicationContext获取或者直接注入到其他类(比如具体的Controller)中.
* 请结合例子"发送事务消息"
*/
ApplicationContext context = new ClassPathXmlApplicationContext("transactionProducer.xml");

TransactionProducer transactionProducer = (TransactionProducer) context.getBean("transactionProducer");

Message msg = new Message("XXX", "TagA", "Hello MQ transaction===".getBytes());

SendResult sendResult = transactionProducer.send(msg, new LocalTransactionExecuter() {
@Override
public TransactionStatus execute(Message msg, Object arg) {
System.out.println("执行本地事务");
return TransactionStatus.CommitTransaction; //根据本地事务执行结果来返回不同的TransactionStatus
}
}, null);
}
}
```

消费者与 Spring 集成

1.创建 MessageListener，如下所示。

```
package demo;

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.MessageListener;

public class DemoMessageListener implements MessageListener {

public Action consume(Message message, ConsumeContext context) {

System.out.println("Receive: " + message.getMsgID());
try {
```

```
//do something..
return Action.CommitMessage;
}catch (Exception e) {
//消费失败
return Action.ReconsumeLater;
}
}
}
```

2.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.springframework.org/schema/beans/spring-beans.xsd">

<bean id="msgListener" class="demo.DemoMessageListener"></bean> <!--Listener配置-->

<bean id="consumer" class="com.aliyun.openservices.ons.api.bean.ConsumerBean" init-method="start" destroy-
method="shutdown">
<property name="properties" > <!--消费者配置信息-->
<props>
<prop key="ConsumerId">CID_DEMO</prop> <!--请替换XXX-->
<prop key="AccessKey">AKDEMO</prop>
<prop key="SecretKey">SKDEMO</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.Subscription">
<property name="topic" value="TopicTestMQ"/>
<property name="expression" value="*"/><!--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>
<!--更多的订阅添加entry节点即可-->
</map>
</property>
</bean>

</beans>
```

3.运行已经与 Spring 集成好的消费者，如下所示。

```
package demo;

import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

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(比如具体的Controller)中
*/
ApplicationContext context = new ClassPathXmlApplicationContext("consumer.xml");
System.out.println("Consumer Started");
}
}
```

C/C++ SDK

环境准备

用 C++ SDK 方式接入 MQ，需要完成以下准备工作。

注意：

> 代码里涉及到的 Topic， Producer ID， Consumer ID，需要到 MQ 控制台上创建。 Message Tag 可以完全由应用自定义，具体创建过程可参考 申请MQ资源。
>
> 使用 MQ 服务的应用程序需要部署在阿里云 ECS 上。

下载 SDK

CPP 支持 Windows 和 Linux 两个跨平台的 SDK，而且接口完全一致。下载依赖 SDK 包链接：

> Linux cpp 下载链接
>
> Windows cpp 下载链接

下载完成后进行解压，会有如下目录结构:

- example/
- include/
- lib/
- readme.txt
- ALIYUN_MQ_CLIENT_FOR_CPP_AND_NET_USER_GUIDE.doc

- release note.txt

上面的目录和文件的作用如下:

example：给的测试 demo，一个发送 demo，一个消费 demo。

include： 用户自己编写的程序需要 include 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，分别是 32 位和 64 位系统下 SDK 的 dll 库。如果没有安装 Visual Studio 2013 环境下，需要拷贝这两个文件到 C:\windows\system32\ 目录下。

```
 32/
64/
msvcp120.dll
msvcr120.dll
```

readme.txt： 如何使用SDK的简要说明。

ALIYUN_MQ_CLIENT_FOR_CPP_AND_NET_USER_GUIDE.doc： SDK环境准备文档。

release note.txt： 新版本发布解决的问题和引入的新特性列表。

Linux C++ SDK 使用

自2016.04.06开始, Linux cpp 版本依赖了高性能 boost 库(1.56.0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 三个库。我们提供了静态库和动态库两种解决方案：

静态解决方案

MQ 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 ibonsclient4cpp.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demo目录,修改demo文件，填入自己申请的topic，key相关的信息
g++ -static -I ../include -L ../lib/lib-boost-static ProducerExampleForEx.cpp -lonsclient4cpp -lpthread –ldl</font>
```

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 shared 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 lonsclient4cpp，其他库动态链接即可。使用方式如下:

```
g++  -I ../include -L ../lib/lib-boost-static ProducerExampleForEx.cpp –Wl,-dn -lonsclient4cpp –Wl,-dy -lpthread –ldl
```

动态解决方案

MQ 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：

下载 boost 1.56.0 版本:

boost 1.56.0

解压 boost 1.56.0:

tar —bzip2 -xf /path/to/boost_1_56_0.tar.bz2

安装 boost 1.56.0 版本:

1) cd path/to/boost_1_56_0

2) 配置 boost：./bootstrap.sh

3) 编译 boost: ./b2 link=shared runtime-link=shared

4) 安装 boost: ./b2 install

执行 ldconfig -v|grep libboost。如果有相关的输出表明 boost 动态库在动态库搜索路径中。

生成可执行文件时，需要链接 boost 动态库和 MQ 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 demo 目录，修改 demo 文件，填入自己在 MQ 控制台申请的 Topic，key 相关的信息
g++ -Wall -Wno-deprecated -L ../lib/lib-boost-share/ -I ../include/ ProducerExampleForEx.cpp -lonsclient4cpp -lboost_system -lboost_thread -lboost_chrono -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 使用

Visual Studio 2013 环境下使用 C++ SDK

使用 Visual Studio 2013 创建自己的项目。

右键单击项目选择**属性**。选择**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 **x86** 或 **x64**。

右键单击项目选择**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 32 位或者 64 位 lib 目录下的所有文件到输出目录 **/A**。

右键单击项目选择**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**。

右键单击项目选择**属性**>**配置属性**>**链接**>**常规**>**附加库目录**：**/A**。

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

非 Visual Studio 2013环境下使用 C++ SDK

首先需要按照 Visusal Studio 2013 的环境来配置，配置过程同上。

右键单击项目选择**属性**>**配置属性**>**C/C++-代码生成**>**运行时检查**，选择**默认**。

右键单击项目选择**属性**>**配置属性**>**清单工具**>**输入和输出**>**嵌入清单**，选择**否**。

拷贝 msvcp120.dll 和 msvcr120.dll 到运行目录或者 C:\Windows\System32 目录。

MQ 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和发送消息所必需的信息;
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId, "XXX");//您在控制台创建的Producer ID
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );// 消息内容
factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//消息内容
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//AccessKey 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" );//SecretKey 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MQ服务器过滤
"TagA",
//Message Body,不能为空，MQ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 MQ 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;
}
```

发送定时消息

目前支持的域包括公网测试、华东1、华北2、华东2、华南1。

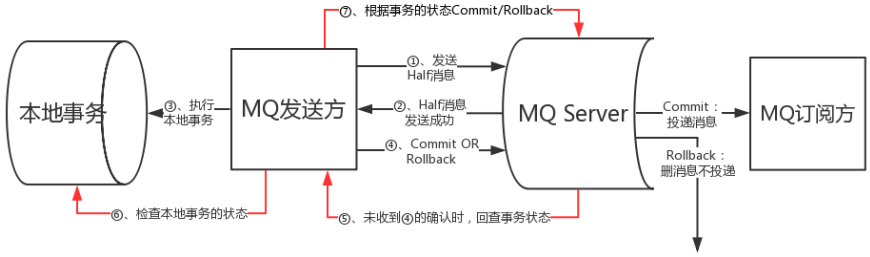
定时消息可以做到在指定时间之后才可被消费者消费，用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发定时任务的场景，类似于延迟队列。代码示例如下：

```
#include "ONSFactory.h"
#include "ONSClientException.h"
using namespace ons;
int main()
{

//创建producer和发送消息所必需的信息;
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId, "XXX");//您在MQ控制台创建的producer
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//您在MQ控制台申请的topic
factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "xxx");//msg content
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxx");//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" );//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 MQ 服务器过滤
"TagA",
//Message Body, 不能为空，MQ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 MQ 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，在这个时刻之后才能被消费，这个例子表示3s后才能被消费
long deliverTime = 获取系统当前时间(ms) + 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;
}
```

发送分布式事务消息

目前支持的域包括公网测试、华东1、华北2、华东2、华南1。

MQ 事务消息交互流程如下。



发送事务消息包含两个步骤：

1.发送半消息（Half Message）及执行本地事务

```
#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console控制不可能查询)
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[])
{
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId, "XXX");//您在控制台创建的Producer ID
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//输入您在控制台创建的Topic
factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//msg content
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxxxxxxxx");//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" );//阿里云身份验证，在阿里云服务器管理控制台创建

//创建producer，MQ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ONS服务器过滤
"TagA",
//Message Body,不能为空，MQ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MQ Console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
{
//MQ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 MQ Broker 向发送方再次询问该消息对应的本地事务的状态。

```
 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console控制不可能查询)
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

1、发送事务消息为什么必须要实现 check 机制？

当步骤1发送半消息完成，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 MQ Broker 的角度看，这条半状态的消息的状态是未知的，因此 MQ Broker 会定期要求发送方能 check 该半状态消息，并上报其最终状态。

2、Check 被回调时，业务逻辑都需要做些什么？

MQ 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。MQ 发送事务消息时需要实现 LocalTransactionChecker 接口，用来处理 MQ 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：

(1) 检查该半消息对应的本地事务的状态（commited or rollback）；

(2) 向 MQ Broker 提交该半消息本地事务的状态。

3、本地事务的不同状态对Half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 MQ Broker 向发送方再次询问该消息对应的本地事务的状态。

具体代码详见 MyLocalTransactionChecker 的实现。

集群方式订阅消息

集群订阅即某个消费者集群只消费指定的Topic，而不是消费所有Topic。

```
#include "ONSFactory.h"
using namespace ons;

// MyMsgListener：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consume 函数
class MyMsgListener : public MessageListener
{

public:

MyMsgListener()
{
}

virtual ~MyMsgListener()
```

```
{
}

virtual Action consume(Message &message, ConsumeContext &context)
{
//自定义消息处理细节
return CommitMessage; //CONSUME_SUCCESS;
}
};


int main(int argc, char* argv[])
{

//pushConsumer创建和工作需要的参数，必须输入
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "XXX");//您在MQ控制台申请的consumerId
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//您在MQ控制台申请的msg topic
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//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");//阿里云身份验证，在阿里云服务器管理控制台创建

//create pushConsumer
PushConsumer* pushConsumer = ONSFactory::getInstance()->createPushConsumer(factoryInfo);

//指定pushConsumer 订阅的消息topic和tag, 注册消息回调函数
MyMsgListener msglistener;
pushConsumer->subscribe(factoryInfo.getPublishTopics(), "*",&msglistener );

//start pushConsumer
pushConsumer->start();

//NOTE:直到不再接收消息，才能调用shutdown；调用shutdown之后，consumer退出，不能接收到任何消息

//销毁pushConsumer, 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
pushConsumer->shutdown();
return 0;

}
```

.NET SDK

环境准备

用 .NET SDK 方式接入 MQ，需要完成以下准备工作。

注意：

代码里涉及到的 Topic， Producer ID， Consumer ID，需要到 MQ 控制台上创建。 Message Tag 可以完全由应用自定义，具体创建过程可参考 申请 MQ 资源。

使用 MQ 服务的应用程序需要部署在阿里云 ECS 上。

下载 SDK

Windows .NET 版本:

我们提供的.NET 版本是基于 MQ 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，内部才用 C++ 多线程并发处理，保证.NET版本的高效稳定。

在使用 VS 开发 .NET 的应用程序和类库时，默认的目标平台为 “Any CPU” ，即会在运行时可根据 CPU 类型自动选择 X86 或 X64，拥有这样的能力是因为 .NET 编译后的程序集是基于 IL 的。在运行时，CLR 才会将其 JIT 发射为 X86 或 X64 的机器码。而 C 或 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，我们将 C/C++ 项目编译为 X86 的32位 dll 或者 X64 的64位 dll，因此我们提供了包含 VS2013 编译的 release 32 位和64位版本 DLL，其他 VS 版本也可以使用。

Windows .NET SDK

下载完成后进行解压，会有如下目录结构:

- example/
- include/
- lib/
- readme.txt
- ALIYUN_MQ_CLIENT_FOR_CPP_AND_NET_USER_GUIDE.doc
- release note.txt

上面的目录和文件的作用如下:

example： 给的测试 demo，一个发送 demo，一个消费 demo。

include： 底层 C++ 库暴露出的头文件，供使用者参考。

lib：

```
  32/
NSClient4CPP.lib
ONSClient4CPP.dll
ONSClient4CPP.pdb
ManagedONS.dll
ManagedONS.pdb
64/
NSClient4CPP.lib
ONSClient4CPP.dll
ONSClient4CPP.pdb
```

```
ManagedONS.dll
ManagedONS.pdb
msvcp120.dll
msvcr120.dll
```

readme.txt： 如何使用SDK的简要说明。

ALIYUN_MQ_CLIENT_FOR_CPP_AND_NET_USER_GUIDE.doc： SDK 环境准备文档。

release note.txt： 新版本发布解决的问题和引入的新特性列表。

NET SDK 配置说明

Visual Studio 2013 使用 .NET SDK 配置说明

使用 Visual Studio 2013 创建自己的项目。

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，**活动解决方案平台**为 x86 或 x64。

C# 工程只需引用 ManagedONS.dll，ManagedONS.dll 自动加载 ONSClient4CPP.dll。

C# 工程属性设置(没有特殊说明，代表 32 位和 64 位的设置一致)：

1). 右键单击 C# 工程选择**属性**>**应用程序**，将目标框架设为 .NET Framework 4.5 或更高版本；

2). 右键单击 C# 工程选择**属性**>**生成**>**配置**，设置为 **release**；

3). 右键单击 C# 工程选择**属性**>**生成**>**目标平台**，设置如下：

如果使用 32 位 DLL，设置为 x86 如果使用 64 位 DLL，设置为 x64

4). 右键单击 C# 工程选择**属性**>**生成**>**输出**>**输出路径**，即程序运行目录 /A，设置为与所有 dll 所保存的目录保持一致，拷贝 32 位或者 64 位 lib 目录下的所有文件到运行目录 /A；

5). 右键单击 C# 工程选择**属性**>**调试**，打开启用本机调试，否则可能会找不到 ONSClient4CPP.dll 等程序集。

非 VS2013 使用 .NET SDK 特殊配置

右键单击 C# 工程选择**属性**>**调试**，勾选**启用非托管调试**，否则可能会找不到 ONSClient4CPP.dll 等程序集。

MQ 发送普通消息

您可以运行以下代码进行消息发送。请按说明正确设置相关参数。

```
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(factoryInfo.getProducerIdName(), "PID_xxxx ");//您在MQ控制台申请的Producer ID
factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "xxx");//您在MQ控制台申请的Topic
factoryInfo.setFactoryProperty(factoryInfo.getMsgContentName(), "xxx");//msg content
factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(), "xxx");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(),"xxx");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

//创建producer
ONSFactory onsfactory = new ONSFactory();
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(
//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，可通过 MQ 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)
{
```

```
//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();

}
}
}
```

MQ 发送定时消息

目前支持的域包括公网测试、华东1、华北2、华东2、华南1。

定时消息可以做到在指定时间之后才可被消费者消费，用于解决一些消息生产和消费有时间窗口要求的场景，或者通过消息触发定时任务的场景，类似于延迟队列。代码示例如下。

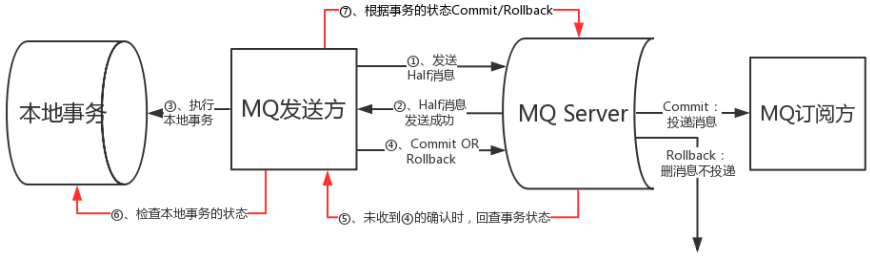
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;

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(factoryInfo.getProducerIdName(), "XXX ");//您在MQ控制台申请的Producer ID
factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "XXX");//您在MQ控制台申请的Topic
factoryInfo.setFactoryProperty(factoryInfo.getMsgContentName(), "XXX");//消息内容
factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(), "XXX");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(),"XXX");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

//创建producer
ONSFactory onsfactory = new ONSFactory();
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(
//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，可通过MQ Console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，在这个时刻之后才能被消费，这个例子表示3s后才能被消费
long deliverTime = 获取系统当前时间(ms) + 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();

}
}
}
```

发送分布式事务消息

目前支持的域包括公网测试、华东1、华北2、华东2、华南1。

MQ事务消息交互流程如下。



发送事务消息包含以下两个步骤：

发送半消息（Half Message）及执行本地事务

```
 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(ref 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在控制台控制不可能查询)
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 ;
}
}
class onscsharp
{

static void Main(string[] args)
{
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
factoryInfo.setFactoryProperty(factoryInfo.getProducerIdName(), "");//您在MQ控制台申请的Producer ID
factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "");// 您在MQ控制台申请的Topic
factoryInfo.setFactoryProperty(factoryInfo.getMsgContentName(), "");//message body
factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(), "");//AccessKey 阿里云身份验证，在阿里云服
```

```
务器管理控制台创建
factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(), "");//SecretKey 阿里云身份验证，在阿里云服
务器管理控制台创建

//create transaction producer
ONSFactory onsfactory = new ONSFactory();
LocalTransactionChecker myChecker = new MyLocalTransactionChecker();
TransactionProducer pProducer = onsfactory.getInstance().createTransactionProducer(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，可通过MQ K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();
}
}
}
```

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MQ 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(ref 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.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在控制台控制不可能查询)
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

1、发送事务消息为什么必须要实现 check 机制？

当步骤 1 半消息发送完成，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 MQ Broker 的角度看，这条半状态的消息的状态是未知的，因此 MQ Broker 会定期要求发送方能 check 该半状态消息，并上报其最终状态。

2、Check 被回调时，业务逻辑都需要做些什么？

MQ 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。MQ 发送事务消息时需要实现 LocalTransactionChecker 接口，用来处理 MQ 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：

（1）检查该半消息对应的本地事务的状态（commited or rollback）；

（2）向 MQ Broker 提交该半消息本地事务的状态。

3、本地事务的不同状态对半消息的影响？

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 MQ Broker 向发送方再次询问该消息对应的本地事务的状态。

具体代码详见 MyLocalTransactionChecker 的实现。

集群方式订阅消息

集群订阅即某个消费者集群只消费指定的Topic，而不是消费所有Topic。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;

namespace ons
{

//pushConsumer拉取到消息后，会主动调用该实例的consume 函数
public class MyMsgListener : MessageListener
{
public MyMsgListener()
{
}

~MyMsgListener()
{
}

public override Action consume(ref Message value)
{
/*
所有中文编码相关问题都在sdk压缩包包含的文档里做了说明，请仔细阅读
```

```
*/
return ons.Action.CommitMessage;
}
}

class onscsharp
{
static void Main(string[] args)
{
//pushConsumer创建和工作需要的参数，必须输入
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
factoryInfo.setFactoryProperty(factoryInfo.getConsumerIdName(), "XXX");//您在MQ控制台申请的Consumer ID
factoryInfo.setFactoryProperty(factoryInfo.getPublishTopicsName(), "XXX");//您在MQ控制台申请的Topic
factoryInfo.setFactoryProperty(factoryInfo.getAccessKeyName(),"xx");//AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
factoryInfo.setFactoryProperty(factoryInfo.getSecretKeyName(), "xxxx");//SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建

//create consumer
ONSFactory onsfactory = new ONSFactory();
PushConsumer pConsumer = onsfactory.getInstance().createPushConsumer(factoryInfo);

//register msg listener and subscribe msg topic
MessageListener msgListener = new MyMsgListener();
pConsumer.subscribe(factoryInfo.getPublishTopics(), "*", ref msgListener);

//start consumer
pConsumer.start();
//consumer启动后，会自动拉取消息，拉取到消息后，会自动调用MyMsgListener实例的consume函数；

//确定消费完成后，调用shutdown函数；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
pConsumer.shutdown();
}
}
}
```