天翼云分布式消息服务

RocketMQAPI 说明文档（Java）

中国电信股份有限公司云计算分公司

目 录

1 生产者

1.1接口方法概述

1.1.1客户端通用接口

1) 建立客户端连接

public int connect() throws MQException

2) 关闭客户端连接

public void close() throws MQException;

1.1.2生产者接口

com.ctg.mq.api.IMQProducer

1）发送消息，返回值为 MQSendResult，抛出异常 MQProducerException

```
public MQSendResult send(MQMessage message) throws MQProducerException;
```

2）发送消息（根据指定 groupId hash 到指定分区）

```
public MQSendResult sendByGroupId(MQMessage message) throws MQProducerException;
```

3）发送消息，oneway，不需要返回值

```
public void sendOneway(MQMessage message) throws MQProducerException;
```

4）发送消息，异步

```
public void sendAsync(MQMessage message, MQSendCallback sendCallback) throws MQProducerException;
```

1.2创建生产者

1.2.1生产者实例化

```
Properties properties = new Properties();
properties.setProperty(PropertyKeyConst.ProducerGroupName, "producer_group");
properties.setProperty(PropertyKeyConst.NamesrvAddr, "127.0.0.1:9876");
properties.setProperty(PropertyKeyConst.NamesrvAuthID, "app4test");
properties.setProperty(PropertyKeyConst.NamesrvAuthPwd, "******");
properties.setProperty(PropertyKeyConst.ClusterName, "defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID, "defaultMQTenantID");

IMQProducer producer = CTGMQFactory.createProducer(properties);
```

1.2.2生产者 PropertyKeyConst 说明

常量字段	说明
ClientCallbackExecutorThreads	客户端回调线程数，默认等于当前可用核数
ClientWorkerThreads	NettyClientConfig 工作线程数，默认 4
CompressMsgBodyOverHowmuch	消息体多大时开始压缩，默认 4k
InstanceName	客户端实例名
MaxMessageSize	消息最大字节数，默认 1024 * 128
NamesrvAddr	namesrv 地址，必要， eg:192.168.10.10:9876;192.168.10.11:9876
NamesrvAuthID	namesrv 用户名，必填

NamesrvAuthPwd	namesrv 密码，必填
ClusterName	客户端订阅 broker 的集群名，根据实际情况设置
TenantID	租户 ID，根据实际情况设置
ProducerGroupName	生产组名称，生产者必填
SendMaxRetryTimes	发送失败默认重试次数，默认 2
SendMsgTimeout	发送超时时间，默认 3s
VipChannelEnabled	true 或者 false，默认是 true，生产和消费端口分开，如果是 false，那就共用一个端口（2.2.6 版本的服务端必须设置为 false）
EnalbeCipher	true 或者 false，默认是 false 客户端 NamesrvAuthPwd 字段填明文密码，如果设置为 true，NamesrvAuthPwd 字段填密文

1.3方法调用说明

1.3.1创建&连接

```
Properties properties = new Properties();
properties.setProperty(PropertyKeyConst.ProducerGroupNa
me, "producer_group");
properties.setProperty(PropertyKeyConst.NamesrvAddr,
"127.0.0.1:9876");
properties.setProperty(PropertyKeyConst.NamesrvAuthID,
"app4test");
properties.setProperty(PropertyKeyConst.NamesrvAuthPwd,
"******");
properties.setProperty(PropertyKeyConst.ClusterName,
"defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID,
"defaultMQTenantID");
```

```
IMQProducer                    producer                    =
CTGMQFactory.createProducer(properties);
producer.connect();
```

1.3.2 关闭连接

```
producer.close();
```

1.3.3 发送消息 sync

函数名：send

功能描述：同步发送消息，设置 message 相关属性，将消息发送至服务器

入参说明：

参数	类型	是否可为空	说明
message	MQMessage	N	消息

输出说明：

参数	类型	说明
mqSendResult	MQSendResult	包含消息 ID

使用例子：

```
MQMessage message = new MQMessage(
        "test_topic_1",// topic
        "ORDER_KEY_1",// key
        "ORDER_TAG",//tag
        ("HELLO ORDER 1").getBytes()// body
);
MQSendResult sendResult = producer.send(message);
```

1.3.4发送消息 async

函数名：sendAsync

功能描述：同步发送消息，设置 message 相关属性，将消息发送至服务器

入参说明：

参数	类型	是否可为空	说明
message	MQMessage	N	消息
mQSendCallback	MQSendCallback	N	回调实例

输出说明：无返回值

使用例子：

```
producer.sendAsync(message, new MQSendCallback() {
    @Override
    public void onSuccess(MQSendResult sendResult) {
        //TODO send success
    }

    @Override
    public void onException(Throwable e) {
        //TODO send Exception
    }
})
```

1.3.5发送消息 oneway

函数名：sendOneway

功能描述：同步发送消息，设置 message 相关属性，将消息发送至服务器

入参说明：

参数	类型	是否可为空	说明
message	MQMessage	N	消息

输出说明：无返回值

使用例子：

```
producer.sendOneway(message);
```

1.3.6发送消息(根据 groupId hash 到指定分区)

函数名：sendAsync

功能描述：同步发送消息，设置 message 相关属性，将消息发送至服务器

入参说明：

参数	类型	是否可为空	说明
message	MQMessage	N	消息，需设置 groupId

输出说明：无返回值

使用例子：

```
message.setGroupId(message.getKey());//本例子用 key 作为
groupId，可以是其它任何类型的字段
producer.sendByGroupId(message);
```

1.4代码示例

```
package com.ctg.guide;

import com.ctg.mq.api.CTGMQFactory;
import com.ctg.mq.api.IMQProducer;
import com.ctg.mq.api.PropertyKeyConst;
import com.ctg.mq.api.bean.MQMessage;
import com.ctg.mq.api.bean.MQSendResult;
import com.ctg.mq.api.exception.MQException;
import com.ctg.mq.api.exception.MQProducerException;

import java.util.Properties;

/**
 * Producer，发送消息
 */
public class Producer {
    public static void main(String[] args) throws
InterruptedException, MQException {

        Properties properties = new Properties();

properties.setProperty(PropertyKeyConst.ProducerGroupNa
me, "producer_group");

properties.setProperty(PropertyKeyConst.NamesrvAddr,
"127.0.0.1:9876");

properties.setProperty(PropertyKeyConst.NamesrvAuthID,
"app4test");

properties.setProperty(PropertyKeyConst.NamesrvAuthPwd,
"******");
properties.setProperty(PropertyKeyConst.ClusterName,
"defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID,
"defaultMQTenantID");

        IMQProducer producer =
```

```
CTGMQFactory.createProducer(properties);//建议应用启动时创
建
        int connectResult = producer.connect();
        if(connectResult != 0){
            return;
        }
        for (int i = 0; i < 10; i++) {
            try {
                MQMessage message = new MQMessage(
                        "test_topic_1",// topic
                        "ORDER_KEY_"+i,// key
                        "ORDER_TAG",//tag
                        ("HELLO     ORDER     BODY"     +
i).getBytes()// body
                );
                MQSendResult         sendResult         =
producer.send(message);
                //System.out.println(sendResult);
                //TODO
            } catch (MQProducerException e) {
                //TODO
            }
        }

        producer.close();//建议应用关闭时关闭

    }
}
```

2 消费者(Push)【建议使用】

注意：同等条件下，push 可以满足绝大部份的使用场景，pull 与 push 模式同时可以满足使用需求的情况下，建议优先使用 push 模式。

2.1 接口方法概述

2.1.1 客户端通用接口

1) 建立客户端连接

public int connect() throws MQException

2) 关闭客户端连接

public void close() throws MQException;

2.1.2 Push 接口

com.ctg.mq.api.IMQPushConsumer

1）订阅主题

```
public void listenTopic(final String topic, final String
subExpression,
    final ConsumerTopicListener topicListener) throws
MQException;
```

2）取消指定主题的订阅

```
public int unListenTopic(String topicName) throws
MQException;
```

3）订阅主题（严格有序）

```
public void listenQueue(final String queue, final String
subExpression,
    final ConsumerQueueListener queueListener) throws
MQException;
```

4）取消指定主题的订阅（严格有序）

```
public int unListenQueue(String queue) throws
MQException;
```

2.2创建消费者

2.2.1消费者实例化

```
final Properties properties = new Properties();
properties.setProperty(PropertyKeyConst.ConsumerGroupName, "test_consumer_1");
properties.setProperty(PropertyKeyConst.NamesrvAddr, "127.0.0.1:9876");
properties.setProperty(PropertyKeyConst.NamesrvAuthID, "app4test");
properties.setProperty(PropertyKeyConst.NamesrvAuthPwd, "******");
properties.setProperty(PropertyKeyConst.ClusterName, "defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID, "defaultMQTenantID");
properties.setProperty(PropertyKeyConst.ConsumeFromWhere,MQConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET.name());
IMQPushConsumer consumer = CTGMQFactory.createPushConsumer(properties);
int connectResult = consumer.connect();
```

2.2.2消费者 PropertyKeyConst 说明

常量字段	说明
CacheMsgTotalSizeThresholdForBroker	push 模式针对单个 broker 缓存消息大小的上限, 默认 200 * 1024 * 1024;
ClientCallbackExecutorThreads	客户端回调线程数，默认等于当前可用核数
ClientWorkerThreads	NettyClientConfig 工作线程数，默认 4
ClusterConsume	消费模式分为集群模式与广播模式 默认为集群模式，枚举值 MessageModel，默认 clustering

ConsumeFromWhere	从哪个位置开始消费,建议设置为 ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET.name()
ConsumeMessageBatchMaxSize	设置 push 批量消费的条数，针对 push 模式，默认是 1
ConsumerGroupName	消费组名称，消费者必要
ConsumerPushThresholdForQueue	push 模式单个队列拉消息上限，默认 1000
ConsumeThreadMax	最大消费线程数，针对 push 模式，默认 64
ConsumeThreadMin	最小消费线程数，针对 push 模式，默认 20
ConsumeTimeoutInSec	消费超时时间 1.默认不设或设置 ConsumeTimeoutInSec<=0：表示不做超时处理，应用自己处理签收 2.设置 ConsumeTimeoutInSec>0：表示一旦超过此消费超时时间（应用未做签收），客户端将**自动签收失败**，消息进入重试队列。
InstanceName	客户端实例名
NamesrvAddr	namesrv 地址，必要， eg:192.168.10.10:9876;192.168.10.11:9876
NamesrvAuthID	namesrv 用户名，必填
NamesrvAuthPwd	namesrv 密码，必填
ClusterName	客户端订阅 broker 的集群名，默认 defaultMQBrokerCluster
TenantID	租户 ID，默认 defaultMQTenantID
TimeoutStrategy	超时策略

2. 3Push 方式的调用说明

2. 3. 1订阅主题

函数名：listenTopic

功能描述：以监听的方式订阅主题消息，这是消费者被动接受消息的方式，即推模式。增加 tag 过滤条件，注意消息的业务逻辑只需由实现 consumer 接口方法。

入参说明：

参数	类型	是否可为空	说明
topicName	String	N	主题名称
SubExpression	String	Y	订阅过滤表达式字符串，服务器依据此表达式进行过滤。只支持或运算 * eg: "tag1 || tag2 || tag3" 如果 subExpression 等于 null 或者*，则表示全部订阅
Listener	ConsumeListener	N	监听器

使用例子：

```
consumer.listenTopic("test_topic_1",      null,      new
ConsumerTopicListener() {
    @Override
    public ConsumerTopicStatus onMessage(List<MQResult>
mqResultList) {
        for(MQResult result : mqResultList) {
```

```
                //TODO
            }
            return  ConsumerTopicStatus.CONSUME_SUCCESS;//对
消息批量确认(成功)
            //return  ConsumerTopicStatus.RECONSUME_LATER;//
对消息批量确认(失败)
        }
});
```

2.3.2取消指定主题的订阅

```
consumer.unListenTopic("test_topic_1");
```

2.3.3订阅主题（有序）

函数名：listenQueue

功能描述：以监听的方式订阅队列消息，这是消费者被动接受消息的方式，即推模式。增加 tag 过滤条件，注意消息的业务逻辑只需由实现 consumer 接口方法。保证消息有序

入参说明：

参数	类型	是否可为空	说明
QueueName	String	N	主题名称
SubExpression	String	Y	订阅过滤表达式字符串，服务器依据此表达式进行过滤。只支持或运算 * eg: "tag1 || tag2 || tag3"

			如果 subExpression 等于 null 或者*，则表示全部订阅
listener	ConsumeListener	N	监听器

使用例子：

```
consumer.listenQueue("order_test_topic_1",   null,   new
ConsumerQueueListener() {
    @Override
    public ConsumerQueueStatus onMessage(List<MQResult>
mqResultList) {
        for(MQResult result : mqResultList) {
            //TODO
        }
        return ConsumerQueueStatus.SUCCESS;//对消息批量签
收(成功)
        //return ConsumerQueueStatus.RECONSUME_LATER;//
对消息批量签收(失败)
    }
});
```

注意：要实现严格有序，创建 topic 时，必须使用有序队列

2.3.4取消指定主题的订阅（有序）

```
consumer.unListenQueue("order_test_topic_1");
```

2.4代码示例

```
package com.ctg.guide;

import com.ctg.mq.api.enums.MQConsumeFromWhere;
import com.ctg.mq.api.CTGMQFactory;
import com.ctg.mq.api.IMQPushConsumer;
import com.ctg.mq.api.PropertyKeyConst;
```

```
import com.ctg.mq.api.bean.MQResult;
import com.ctg.mq.api.listener.ConsumerTopicListener;
import com.ctg.mq.api.listener.ConsumerTopicStatus;

import java.util.List;
import java.util.Properties;

public class PushConsumer {

    public static void main(String[] args) throws
Exception {
        final Properties properties = new Properties();

properties.setProperty(PropertyKeyConst.ConsumerGroupNa
me, "test_consumer_1");

properties.setProperty(PropertyKeyConst.NamesrvAddr,
"127.0.0.1:9876");

properties.setProperty(PropertyKeyConst.NamesrvAuthID,
"app4test");

properties.setProperty(PropertyKeyConst.NamesrvAuthPwd,
"******");
properties.setProperty(PropertyKeyConst.ClusterName,
"defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID,
"defaultMQTenantID");


        IMQPushConsumer consumer =
CTGMQFactory.createPushConsumer(properties);
        int connectResult = consumer.connect();
        if (connectResult != 0) {
            return;
        }
        consumer.listenTopic("test_topic_1", null, new
ConsumerTopicListener() {
            @Override
            public ConsumerTopicStatus
onMessage(List<MQResult> mqResultList) {
                //mqResultList 默认为 1，可通过批量消费数量设
```

```
置
                for(MQResult result : mqResultList) {
                    //TODO
                    System.out.println(result);
                }
                return
ConsumerTopicStatus.CONSUME_SUCCESS;//对消息批量确认(成功)
                //return
ConsumerTopicStatus.RECONSUME_LATER;//对消息批量确认(失败)
            }
        });
    }
}
```

3 消费者(Pull)

3.1接口方法概述

3.1.1客户端通用接口

1) 建立客户端连接

public int connect() throws MQException

2) 关闭客户端连接

public void close() throws MQException;

3.1.2Pull 接口

com.ctg.mq.api.IMQPullConsumer

1) 消费主题消息

```
public   List<MQResult>   consumeMessagesByTopic(String
topic, String subExpression, int maxNum,
    int timeout) throws MQException;
```

2）消费重试队列消息

```
public List<MQResult> consumeSendBackMessage(int maxNum)
throws MQException;
```

注意：

✧ 当 签 收 失 败 consumer.ackMessage(result, ConsumeStatus.RECONSUME_LATER);消息会进入重试队列。

✧ 原生 pull 需调用 consumeSendBackMessage 才可能消费重试队列的消。

3）消息签收（单条）

```
public void ackMessage(MQResult mqResult, ConsumeStatus
status) throws MQAckException;
```

4）消息签收（批量）

```
public void ackMessages(List<MQResult> mqResults,
ConsumeStatus status) throws MQException;
```

3.2创建消费者

3.2.1消费者实例化

```
final Properties properties = new Properties();
properties.setProperty(PropertyKeyConst.ConsumerGroupNa
me, "test_consumer_1");
properties.setProperty(PropertyKeyConst.NamesrvAddr,
"127.0.0.1:9876");
properties.setProperty(PropertyKeyConst.NamesrvAuthID,
"app4test");
properties.setProperty(PropertyKeyConst.NamesrvAuthPwd,
"******");
properties.setProperty(PropertyKeyConst.ClusterName,
"defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID,
"defaultMQTenantID");
properties.setProperty(PropertyKeyConst.ConsumeFromWher
e,MQConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET.name());
```

```
IMQPullConsumer                    consumer                    =
CTGMQFactory.createPullConsumer(properties);
int connectResult = consumer.connect();
```

3.2.2消费者 PropertyKeyConst 说明

常量字段	说明
ClientCallbackExecutorThreads	客户端回调线程数，默认等于当前可用核数
ClientWorkerThreads	NettyClientConfig 工作线程数，默认 4
ClusterConsume	消费模式分为集群模式与广播模式 默认为集群模式，枚举值 MessageModel，默认 clustering
ConsumeFromWhere	从哪个位置开始消费
ConsumerGroupName	消费组名称，消费者必要
ConsumeTimeoutInSec	消费超时时间 1.默认不设或设置 ConsumeTimeoutInSec<=0：表示不做超时处理，应用自己处理签收 2.设置 ConsumeTimeoutInSec>0：表示一旦超过此消费超时时间（应用未做签收），客户端将**自动签收失败**，消息进入重试队列。
InstanceName	客户端实例名
NamesrvAddr	namesrv 地址，必要， eg:192.168.10.10:9876;192.168.10.11:9876
NamesrvAuthID	namesrv 用户名，必填
NamesrvAuthPwd	namesrv 密码，必填
ClusterName	客户端订阅 broker 的集群名，根据实际情况设置

TenantID	租户 ID，根据实际情况设置

3.3方法调用说明

3.3.1从主题拉取消息

函数名：consumeMessagesByTopic

功能描述：ctgmq 客户端主动从 mq 服务器的指定队列拉取指定数量消息，一次性最大只能拉取 32 条，故 num>32?32:num，如果在指定的超时时间内成功拉取到消息消息，则返回消息列表，否则返回 null。每次返回的消息数量小于或等于指定的数量。

参数	类型	是否可为空	说明
Topic	String	N	主题名称
SubExpression	String	Y	订阅过滤表达式字符串，服务器依据此表达式进行过滤。只支持或运算 eg: "tag1 || tag2 || tag3" 如果subExpression等于null或者*，则表示全部订阅
maxNum	Int	N	最大拉取消息设置
timeout	Int	N	拉取超时时间

输出说明：

参数	类型	说明
MqresultList	List<MQResult>	消息结果

使用例子：

```
List<MQResult>                mqResultList                =
consumer.consumeMessagesByTopic(
        //topic 名称
        "test_topic_1",
        //过滤条件
        null,
        //一次拉取数量，最大 32 条
        32,
        //拉取超时时间
        3000);
```

3.3.2 从主题拉取消息（有序）

函数名：consumeMessagesByQueue

功能描述：ctgmq 客户端主动从 mq 服务器的指定队列拉取指定数量消息，一次性最大只能拉取 32 条，故 num>32?32:num，如果在指定的超时时间内成功拉取到消息消息，则返回消息列表，否则返回 null。每次返回的消息数量小于或等于指定的数量。

入参说明：

参数	类型	是否可为空	说明
Queue	String	N	队列名称
SubExpression	String	Y	订阅过滤表达式字符串，服务器依据此表达式进行过滤。只支持或运算 eg: "tag1 || tag2 || tag3" 如果subExpression等于null或者*，则表示全部订阅

maxNum	Int	N	最大拉取消息设置
timeout	Int	N	拉取超时时间

输出说明：

参数	类型	说明
MqresultList	List<MQResult>	消息结果

使用例子：

```
List<MQResult>                    mqResultList                    =
consumer.consumeMessagesByQueue(
        //topic 名称
        "test_order_topic_1",//必须是有序队列
        //过滤条件
        null,
        //一次拉取数量，最大 32 条
        32,
        //拉取超时时间
        3000);
```

3.3.3 签收消息（失败）

```
consumer.ackMessage(result,
ConsumeStatus.RECONSUME_LATER);
```

3.3.4 签收消息（成功）

```
consumer.ackMessage(result,
ConsumeStatus.CONSUME_SUCCESS);
```

3.3.5批量签收消息（只适用于无序消费）

```
consumer.ackMessages(mqResultList,ConsumeStatus.CONSUME
_SUCCESS);
```

3.3.6消费重试队列里面的消息（只适用于无序消费）

```
List<MQResult>                retryResultList                =
consumer.consumeSendBackMessage(32);
```

3.4代码示例

```
package com.ctg.guide;

import com.ctg.mq.api.enums.MQConsumeFromWhere;
import com.ctg.mq.api.CTGMQFactory;
import com.ctg.mq.api.IMQPullConsumer;
import com.ctg.mq.api.PropertyKeyConst;
import com.ctg.mq.api.bean.ConsumeStatus;
import com.ctg.mq.api.bean.MQResult;
import com.ctg.mq.api.exception.MQException;

import java.util.List;
import java.util.Properties;

/**
 * pull 拉取消息
 */
public class PullConsumer {

    public  static  void  main(String[]  args)  throws
Exception {
        final Properties properties = new Properties();

properties.setProperty(PropertyKeyConst.ConsumerGroupNa
me, "test_consumer_1");

properties.setProperty(PropertyKeyConst.NamesrvAddr,
"127.0.0.1:9876");
```

```
properties.setProperty(PropertyKeyConst.NamesrvAuthID,
"app4test");

properties.setProperty(PropertyKeyConst.NamesrvAuthPwd,
"******");
properties.setProperty(PropertyKeyConst.ClusterName,
"defaultMQBrokerCluster");
properties.setProperty(PropertyKeyConst.TenantID,
"defaultMQTenantID");

        IMQPullConsumer            consumer            =
CTGMQFactory.createPullConsumer(properties);
        int connectResult = consumer.connect();
        if (connectResult != 0) {
            return;
        }
        while (true) {
            List<MQResult>        mqResultList        =
consumer.consumeMessagesByTopic(
                    //topic 名称
                    "test_topic_1",
                    //过滤条件
                    null,
                    //一次拉取数量，最大 32 条
                    32,
                    //拉取超时时间
                    3000);
            for (MQResult result : mqResultList) {
                try {
                    //TODO
                    System.out.println(result);
                    //业务处理正常，签收成功
                    consumer.ackMessage(result,
ConsumeStatus.CONSUME_SUCCESS);
                    //业务处理失败时，应签收失败，消息进入重试
```

```
队列
                    //consumer.ackMessage(result,
ConsumeStatus.RECONSUME_LATER);
                } catch (MQException e) {
                    //TODO
                }
            }

            //拉取重试队列的消息（注意：重试队列有签收失败的消
息才会存在，如果没有签收失败的消息，消费重试队列会抛出异常）
            try {
                List<MQResult>     retryResultList     =
consumer.consumeSendBackMessage(32);
                for (MQResult result : retryResultList)
{
                    try {
                        //TODO
                        System.out.println(result);
                        //业务处理正常，签收成功
                        consumer.ackMessage(result,
ConsumeStatus.CONSUME_SUCCESS);
                        //业务处理失败时，应签收失败
                        //consumer.ackMessage(result,
ConsumeStatus.RECONSUME_LATER);
                    } catch (MQException e) {
                        //TODO
                    }
                }
            }catch (MQException e){
                //TODO LOG no message queue of topic
            }

        }
    }
}
```

4 事务消息

4.1接口方法概述

4.1.1客户端通用接口

1) 建立客户端连接

public int connect() throws MQException

2) 关闭客户端连接

public void close() throws MQException;

4.1.2生产者接口

com.ctg.mq.api.impl.MQTransactionProducerImpl

1）发送消息，返回值为 MQSendResult，抛出异常 MQProducerException

```
public MQSendResult sendTransactionMessage(final MQMessage message, final MQLocalTransactionExecuter executer, Object arg)
```

4.2创建生产者

4.2.1生产者实例化

```
MQTransactionProducerImpl producer = new MQTransactionProducerImpl(namesrvaddr,
"CrmProducerGroup", new LocalTransactionCheckerImpl());
producer.setAuthID("app4test");
producer.setAuthPWD("******");
producer.setTenantID("defaultMQTenantID");
producer.setClusterName("defaultMQBrokerCluster");
producer.setSendMsgTimeout(timeout);
```

4.2.2生产者属性说明

常量字段	说明
CompressMsgBodyOverHowmuch	消息体多大时开始压缩，默认 4k
InstanceName	客户端实例名
MaxMessageSize	消息最大字节数，默认 1024 * 128
NamesrvAddr	namesrv 地址，必要， eg:192.168.10.10:9876;192.168.10.11:9876
AuthID	namesrv 用户名，必填
AuthPwd	namesrv 密码，必填
ClusterName	客户端订阅 broker 的集群名，默认 defaultMQBrokerCluster
TenantID	租户 ID，默认 defaultMQTenantID
ProducerGroupName	生产组名称，生产者必填
SendMsgTimeout	发送超时时间，默认 3s

4.3方法调用说明

4.3.1创建&连接

```
MQTransactionProducerImpl producer = new MQTransactionProducerImpl(namesrvaddr,
"CrmProducerGroup", new LocalTransactionCheckerImpl());
producer.setAuthID("app4test");
producer.setAuthPWD("******");
producer.setTenantID("defaultMQTenantID");
producer.setClusterName("defaultMQBrokerCluster");
producer.setSendMsgTimeout(timeout);
```

```
int connect = producer.connect();
```

4.3.2 关闭连接

```
producer.close();
```

4.3.3 发送消息

函数名：sendTransactionMessage

功能描述：同步发送消息，设置 message 相关属性，将消息发送至服务器

入参说明：

参数	类型	是否可为空	说明
message	MQMessage	N	消息
executer	LocalTransactionExecuterImpl	N	本地事务执行器，用于判断事务是否提交

输出说明：

参数	类型	说明
mqSendResult	MQSendResult	包含消息 key 等属性

使用例子：

```
MQMessage message = new MQMessage(
        "test_topic_1",// topic
        "ORDER_KEY_1",// key
        "ORDER_TAG",//tag
        ("HELLO ORDER 1").getBytes()// body
);
MQSendResult                    send                    =
producer.sendTransactionMessage(message,      executer,
null);
```

4.4代码示例

示例解析

启动生产者，后启动消费者

生产者代码示例：

```
1.  import java.util.Random;
2.  import com.ctg.mq.api.bean.MQMessage;
3.  import com.ctg.mq.api.bean.MQSendResult;
4.  import com.ctg.mq.api.exception.MQException;
5.  import com.ctg.mq.api.impl.MQTransactionProducerImpl;
6.
7.  public class TransactionProducer {
8.      public static String getRandomString(int length) {
9.          String str = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
    23456789";
10.         Random random = new Random();
11.         StringBuffer sb = new StringBuffer();
12.         for (int i = 0; i < length; i++) {
13.             int number = random.nextInt(62);
14.             sb.append(str.charAt(number));
15.         }
16.         return sb.toString();
17.     }
18.     public static void main(String[] args) throws MQException {
19.         final String namesrvaddr = args.length >= 1 ? args[0] : "127.0.0.1:8
    876";
20.         final int num = args.length >= 2 ? Integer.parseInt(args[1]) : 100;

21.         final int timeout = args.length >= 3 ? Integer.parseInt(args[2]) : 5
    000;
22.         MQTransactionProducerImpl producer = new MQTransactionProducerImpl(n
    amesrvaddr, "CrmProducerGroup", new LocalTransactionCheckerImpl());
23.         producer.setAuthID("app4test");
24.         producer.setAuthPWD("******");
25.         producer.setTenantID("defaultMQTenantID");
26.         producer.setClusterName("defaultMQBrokerCluster");
27.         producer.setSendMsgTimeout(timeout);
28.         String topic = "consistent_gz_gd_crm_0";
```

```
29.
30.         int connect = producer.connect();
31.         if ( connect==0 ) {
32.             LocalTransactionExecuterImpl executer = new LocalTransactionExec
    uterImpl();//本地事务执行器
33.             for (int i = 0; i < num; i++) {
34.                 MQMessage message = new MQMessage(topic, i + "", "", "consis
    tent".getBytes());
35.                 int a = (int) Math.round(Math.random() * (10 - 2) + 1); // (
    数据类型)(最小值+Math.random()*(最大值-最小值+1))
36.                                                                     // ,
    2-10 之间的随机整数
37.                 message.setBody(getRandomString(1024 * a).getBytes());
38.                 MQSendResult send = producer.sendTransactionMessage(message,
     executer, null);//发送事务消息，第二个参数为本地事务执行器
39.                 System.out.println("No "+ i +" messageId:" + send.getMessage
    ID());
40.             }
41.             producer.close();
42.         }
43.     }
44. }
```

```
1.  import com.ctg.mq.api.bean.MQMessageExt;
2.  import com.ctg.mq.api.transaction.MQLocalTransactionChecker;
3.  import com.ctg.mq.api.transaction.MQLocalTransactionState;
4.
5.
6.
7.  /**
8.   * 本地事务检查器，回查本地事务，Broker 回调 Producer，将未结束的事务发给
    Producer，由 Producer 来再次决定事务是提交还是回滚
9.   */
10. public class LocalTransactionCheckerImpl implements MQLocalTransactionChecke
    r {
11.     final  BusinessService businessService = new BusinessService();
12.
13.
14.     @Override
```

```
15.     public MQLocalTransactionState checkLocalTransactionState(MQMessageExt m
    sg) {
16.         String businessServiceArgs = msg.getKey();
17.         MQLocalTransactionState transactionStatus = MQLocalTransactionState.
    UNKNOW;
18.         try {
19.             boolean isCommit =
20.                 //  根据业务定制 execbusinessService 方法
21.                 businessService.execbusinessService(businessServiceArgs);
22.             if (isCommit) {
23.                 //本地事务已成功、提交消息
24.                 transactionStatus = MQLocalTransactionState.COMMIT_MESSAGE;

25.             } else {
26.                 //本地事务已失败、回滚消息
27.                 transactionStatus = MQLocalTransactionState.ROLLBACK_MESSAGE
    ;
28.             }
29.         } catch (Exception e) {
30.             e.printStackTrace();
31.         }
32.         return transactionStatus;
33.     }
34. }
```

```
1.  import com.ctg.mq.api.bean.MQMessage;
2.  import com.ctg.mq.api.transaction.MQLocalTransactionExecuter;
3.  import com.ctg.mq.api.transaction.MQLocalTransactionState;
4.
5.
6.  /**
7.   * 本地事务执行器    本地事务成功就提交，否则事务则会回滚
8.   */
9.  public class LocalTransactionExecuterImpl implements MQLocalTransactionExecu
    ter {
10.     final BusinessService businessService = new BusinessService();
11.
12.     @Override
13.     public MQLocalTransactionState execute(MQMessage msg, Object arg) {
14.         String businessServiceArgs = msg.getKey();
```

```
        MQLocalTransactionState transactionStatus = MQLocalTransactionState.UNKNOW;
        try {
            //根据业务编写 execbusinessService 方法，此处将消息 key 作为参数，偶数提交，奇数不提交
            boolean isCommit =
                businessService.execbusinessService(businessServiceArgs);
            if (isCommit) {
                // 本地事务成功、提交消息
                transactionStatus = MQLocalTransactionState.COMMIT_MESSAGE;

            } else {
                // 本地事务失败、回滚消息
                transactionStatus = MQLocalTransactionState.ROLLBACK_MESSAGE;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return transactionStatus;
    }
}
```

```
/**
 * 事务提交逻辑 service，可根据实际需求作事务提交的定制
 */
public class BusinessService {
    //可以根据业务自行定义逻辑
    public boolean execbusinessService(String businessServiceArgs) {
        int x = Integer.valueOf(businessServiceArgs);
        if (x % 2 == 0) {
            System.out.println("Success!");
            return true;
        } else {
            System.out.println("Failed");
            return false;
        }
    }
}
```

消费者示例：

```
1. import java.util.List;
2. import java.util.Properties;
3. import java.util.concurrent.atomic.AtomicInteger;
4. import com.ctg.mq.api.CTGMQFactory;
5. import com.ctg.mq.api.IMQPullConsumer;
6. import com.ctg.mq.api.PropertyKeyConst;
7. import com.ctg.mq.api.bean.ConsumeStatus;
8. import com.ctg.mq.api.bean.MQResult;
9. import com.ctg.mq.api.enums.MQConsumeFromWhere;
10. import com.ctg.mq.api.exception.MQException;
11. /**
12.  * 资料一致性
13. pull 模式消费
14.  */
15. public class Consumer {
16.     private static AtomicInteger countConsumer = new AtomicInteger(0);
17.     public static void main(String[] args) throws MQException {
18.         final int timeout = 3000;
19.
20.         String topic = "consistent_gz_gd_crm_0";
21.         Properties properties = new Properties();
22.         properties.setProperty(PropertyKeyConst.ConsumerGroupName,"gz_consis
    tent_consumer1");
23.         properties.setProperty(PropertyKeyConst.InstanceName, "crmInstance")
    ;
24.         properties.setProperty(PropertyKeyConst.NamesrvAddr, "127.0.0.1:8876
    ");
25.         properties.setProperty(PropertyKeyConst.NamesrvAuthID, "app4test");

26.         properties.setProperty(PropertyKeyConst.NamesrvAuthPwd, "******");
27.         properties.setProperty(PropertyKeyConst.ClusterName, "defaultMQBroke
    rCluster");
28.         properties.setProperty(PropertyKeyConst.TenantID, "defaultMQTenantID
    ");
29.         properties.setProperty(PropertyKeyConst.ConsumeMessageBatchMaxSize,
    "32");
30.         properties.setProperty(PropertyKeyConst.ConsumeTimeoutInSec, String.
    valueOf(timeout));
31.         properties.setProperty(PropertyKeyConst.ConsumeFromWhere, MQConsumeF
    romWhere.CONSUME_FROM_FIRST_OFFSET.name());
32.
33.         IMQPullConsumer consumer = CTGMQFactory.createPullConsumer(propertie
    s);
```

```
34.         int connectResult = consumer.connect();
35.         if (connectResult == 0) {
36.             while (true) {
37.                 try {
38.                     List<MQResult> mqResultList = consumer.consumeMessagesBy
    Queue(topic, null, 1, timeout);
39.                     if (mqResultList.size() > 0) {
40.                         for (MQResult result : mqResultList) {
41.                             //客户端需要自行做消费比对，防止重复消费
42.                             consumer.ackMessage(result, ConsumeStatus.CONSUM
    E_SUCCESS);//签收消息
43.                             System.out.println("messageid : " + result.getMe
    ssage().getMessageID() + ", 数
    量:" + countConsumer.addAndGet(1) + ", key:" + result.getMessage().getKey()
    + ", content:" + new String(result.getMessage().getBody()));
44.                         }
45.                     }
46.                 } catch (MQException e) {
47.                     // TODO: handle exception
48.                     System.out.println(e);
49.                 }
50.             }
51.         }
52.
53.     }
54. }
```

5 附录

5.1PropertyKeyConst 说明

常量字段	说明

CacheMsgTotalSizeThresholdForBroker	push 模式针对单个 broker 缓存消息大小的上限, 默认 200 * 1024 * 1024;
ClientCallbackExecutorThreads	客户端回调线程数，默认等于当前可用核数
ClientWorkerThreads	NettyClientConfig 工作线程数，默认 4
ClusterConsume	消费模式分为集群模式与广播模式 默认为集群模式，枚举值 MessageModel，默认 clustering
CompressMsgBodyOverHowmuch	消息体多大时开始压缩，默认 4k
ConsumeFromWhere	从哪个位置开始消费
ConsumeMessageBatchMaxSize	设置 push 批量消费的条数，针对 push 模式，默认是 1
ConsumeOrderly	是否有序是否顺序消费，默认否。仅支持 BDB 消费模式 eg:”true”
ConsumerGroupName	消费组名称，消费者必要
ConsumerPushThresholdForQueue	push 模式单个队列拉消息上限，默认 1000
ConsumeThreadMax	最大消费线程数，针对 push 模式，默认 64
ConsumeThreadMin	最小消费线程数，针对 push 模式，默认 20
ConsumeTimeoutInSec	消费超时时间，超过此消费超时时间则直接签收并发送到重试队列，默认不做超时处理 <=0 表示不判断超时，应用自己处理 >0 表示判断超时，客户端执行超时处理，应用可能存在线程卡死情况，应用需注意，甚至告警
InstanceName	客户端实例名
MaxMessageSize	消息最大字节数，默认 1024 * 128

NamesrvAddr	namesrv 地址，必要， eg:192.168.10.10:9876;192.168.10.11:9876
NamesrvAuthID	namesrv 用户名，必填
NamesrvAuthPwd	namesrv 密码，必填
ClusterName	客户端订阅 broker 的集群名，根据实际情况设置
TenantID	租户 ID，根据实际情况设置
ProducerGroupName	生产组名称，生产者必填
SendMaxRetryTimes	发送失败默认重试次数，默认 2
SendMsgTimeout	发送超时时间，默认 3s
TimeoutStrategy	超时策略

5.2MQMessage 说明

属性	类型	说明
sourceType	int	0：主题 1：队列
sourceName	String	主题或队列名称
body	byte[]	消息体
Key	String	消息体的 key
Tag	String	消息的标签
groupId	Object	发往指定分区的关键词，可以为空， 如果设置将发往其 hash 值的分区

5.3MQMessageExt 说明

属性	类型	说明
message	MQMessage	消息对象
queueId	Int	分区号
queueOffset	long	逻辑偏移量
bornTimestamp	long	消息在客户端创建时间戳
bornHost	SocketAddress	消息来自哪里
storeTimestamp	long	消息在服务器存储时间戳
storeHost	SocketAddress	消息存储在哪个服务器
msgId	String	消息 ID
commitLogOffset	long	物理偏移量
reconsumeTimes	Int	当前消息被某个订阅组重新消费了几次
brokerName	String	消息所在的 broker 名字
consumerStatus	MessageStatus	消息的状态
Properties	Map<String,String>	消息属性，都是系统属性

5.4MQResult 说明

属性	类型	说明
message	MQMessageExt	消息对象

5.5异常 MQException 说明

属性	类型	说明
errorCode	int	异常编码
expDesc	String	异常描述
Cause	Throwable	异常对象

5.6异常码 MQExceptionCode 说明

分类	异常码	说明	建议处理
明确失败错误： CtgMQ 继承异常码	SERVICE_BUSY	服务端由于线程池拥堵，系统繁忙	间隔重试、告警处理
	SERVICE_NOT_AVAILABLE	服务不可用，可能是正在关闭或者权限问题, 或磁盘空间满了	间隔重试、告警处理
	REQUEST_CODE_NOT_SUPPORTED	请求代码不支持	检查包版本
	VERSION_NOT_SUPPORTED	Broker, Namesrv 版本号或请求代码不支持	检查版本
	MESSAGE_ILLEGAL	消息非法	检查消息内容，如 body 大小

	NO_PERMISSION	Broker, Namesrv 无权限执行此操作，可能是发、收、或者其他操作	检查队列、broker、订阅组权限
	TOPIC_NOT_EXIST	Broker, Topic 不存在	检查主题是否存在，可通过管理台查看
明确失败错误：CtgMQ 扩展异常码	QUERY_NOT_FOUND	查询不到消息	检查查询条件是否正确
	INIT_FAIL	初始化失败	查看启动日志
	INVALID_ARGUMENT	参数不合法	查看日志，检查参数
	AUTHENTICATION_FAIL	认证失败	检查用户名密码
	INTERRUPTED	被中断	查看进程、线程控制逻辑
	INTERNAL_ERROR	MQ 内部错误	告警上报
	FAIL_TO_COMPUTE_PULL_FROM_WHERE	无法获取初始消费位置	查看日志，检查错误
	FAIL_TO_FETCH_MESSAGE_QUEUE	无法获取队列信息	检查 topic 已经创建，或已有消费实例在消费
	ORDER_MESSAGE_NOT_ACK	顺序消息有消息在消费，无法继续拉取消息	顺序消息签收完再拉取
	OVER_MAX_NUM_NOT_ACK	超过限制数目消息未签收，无法继续拉取消息	签收完已拉取消息再拉取新消息
	OVER_MAX_SPAN_NOT_ACK	超时限制间隔消息未签收，无法继续拉取消息	签收完已拉取消息再拉取新消息

	PULL_TIMEOUT	拉取消息超时	增大超时时间，或查看错误信息
	BATCH_ACK_ORDER_MSG_NOT_ALLOWED	顺序消息不能批量签收	单条签收
	MESSAGE_QUEUE_NOT_IN_PROCESSING	签收消息时消息状态异常	检查错误信息，确认消息是同一个实例中拉取和签收
	MESSAGE_QUEUE_IS_DROPPED	签收消息时，队列已经不属于当前实例	查看队列是否被分配给了同一个消费组的其他消费实例
	REMOTING_ERROR	网络故障	查看异常日志，检查网络与服务是否正常
	SYSTEM_ERROR	系统错误	查看异常日志，检查服务是否正常
	ORDER_MESSAGE_NOT_ACK	拉消息，顺序消费时有消息没有 ack，无法拉取新消息	Broker 端的错误，消息签收完再拉取
	MESSAGE_ALREADY_ACKED	消息已经被签收，无法再签收	记录异常，记录消息状态
	MESSAGE_IS_NOT_CONSUMING	消息没有在消费，无法签收	检查是否消息从同一个消费者实例拉取和签收
	SKIP_ACK_NOT_ALLOWED_FOR_ORDER_MESSAGE	顺序消息不能跳跃签收，只能按顺序签收	检查是否存在拉取多条顺序消息，但是进行了跳跃签收

	MASSAGE_ID_INVALID	Broker 端错误, 通过 id ack 消息，但是找不到此 id	解析消息 id 失败或者对应的 broker 发生主备切换，建议重试
	BROKER_IS_SLAVE	Broke 端错误， ack 消息，不能在 slave 上进行	建议重试，等主备切换后，将 ack 消息发到主上
	TOO_MUCH_MESSAGE_NOT_ACK	Broker 端错误, 拉消息，太多消息没有 ack，不能拉取新消息	建议签收消息以后再拉取
	CONSUME_GROUP_MECHANISM_ERROR	订阅组消费机制有误，不可变更，不可消费，请求失败	创建订阅组时指定消费机制与当前消费组选取的下消费机制不符合。建议更改消费者的消费机制（当前二种消费机制，原生版本 1.x 版本模式消费机制和 BDB 模式消费机制，2.2.0 版本新增错误码）。
	CONSUME_GROUP_MECHANISM_NOT_EXIST	订阅组未指定消费机制，请求失败	订阅组的消费机制没有指定，建议更新订阅组信息
	CONSUMER_OFFSET_RESETING	消费进度正在重置中，无法再次重置	建议等上一次的消费进度重置成功后，再次执

			行（2.2.0 版本新增错误码）
不确定异常错误：CtgMQ 扩展异常码（针对消息发送）	PRODUCER_UNCERTAIN_FLUSH_DISK_TIMEOUT	发送异常：Broker 持久化磁盘超时	记录消息，介入处理，只要 broker 所在机器不重启，则消息可以正常消费。
	PRODUCER_UNCERTAIN_SLAVE_NOT_AVAILABLE	发送异常：Broker 同步双写，Slave 不可用	记录消息，介入处理，只要 broker 所在机器不故障，则消息可以正常消费。
	PRODUCER_UNCERTAIN_FLUSH_SLAVE_TIMEOUT	发送异常：Broker 同步双写，等待 Slave 应答超时	记录消息，介入处理，只要 broker 所在机器不故障，则消息可以正常消费。
	PRODUCER_UNCERTAIN_REMOTING_ERROR	发送异常：发送网络故障	记录消息，查询确认消息是否成功发送，如果不成功则重发
	PRODUCER_UNCERTAIN_SEND_TIMEOUT	发送异常：发送超时	记录消息，查询确认消息是否成功发送，如果不成功则重发
不确定异常错误：CtgMQ 扩展错误码	ACK_UNCERTAIN_REMOTING_ERROR	签收异常：发送网络故障	记录消息，查询确认消息是否成功签收

（针对消息签收）	ACK_UNCERTAIN_ACK_TIMEOUT	签收异常：签收超时	记录消息，查询确认消息是否成功签收

特别的，为兼容 1.x 老版本，我们新增了错误码:

```
/**
* 消息已经被签收，无法再签收
*/
public static final int MESSAGE_ALREADY_ACKED_V1 = 123;

/**
* 没有消息在消费，无法签收
*/
public static final int MESSAGE_IS_NOT_CONSUMING_V1 = 124;
public static final int SKIP_ACK_NOT_ALLOWED_FOR_ORDER_MESSAGE_V1 = 125;
```

2.X 版本与 1.X 版本冲突的错误

2.x 中定义 public static final int FAIL_TO_FETCH_BROKER_QUEUE = 112;

1.x 中定义 public static final int ORDER_MESSAGE_NOT_ACK = 112;