消息队列 CMQ

消息队列（Queue）模型

腾讯云

【联系我们】

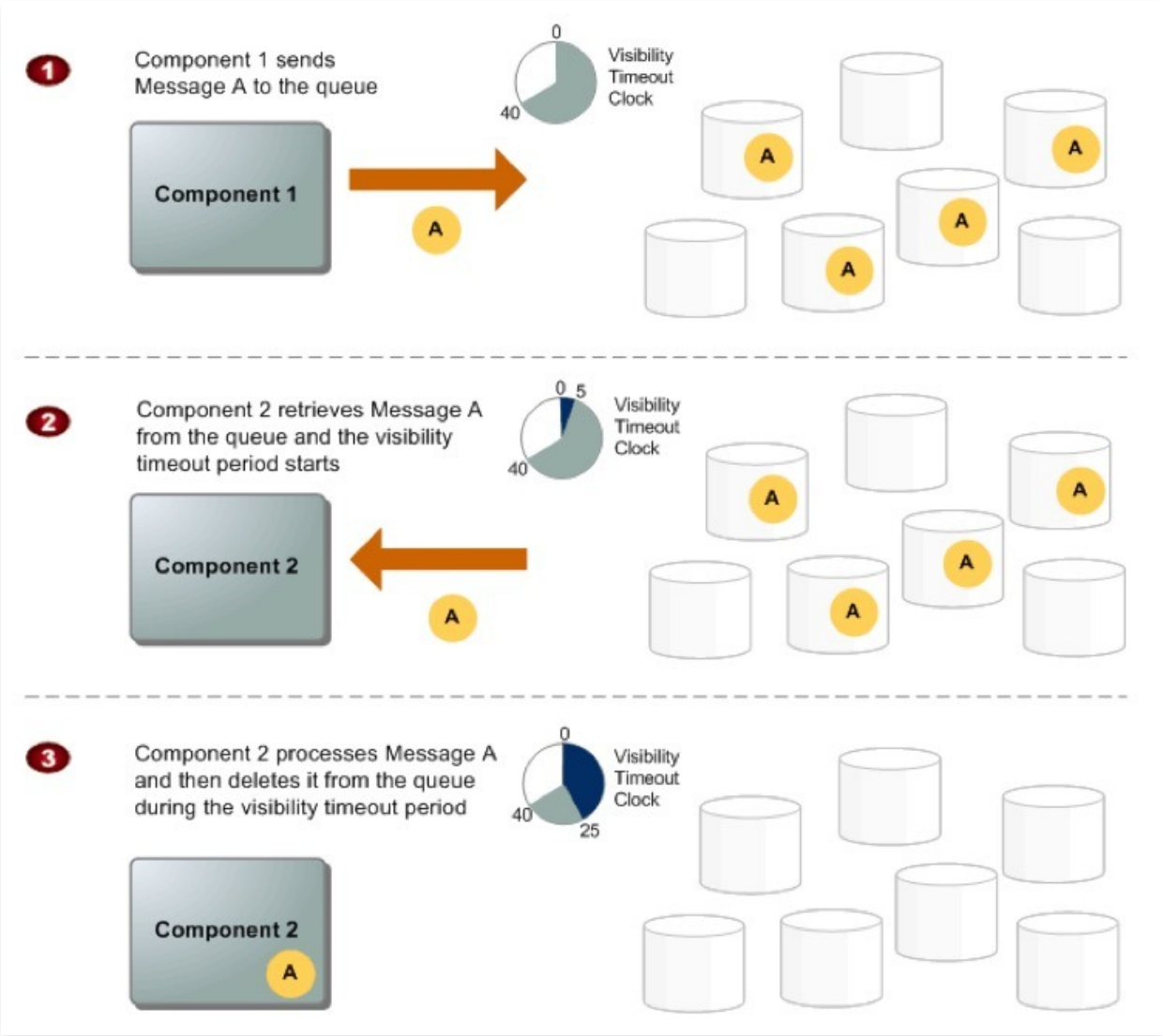
我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

消息队列（Queue）模型
消息生命周期

最近更新时间：2020-07-31 17:48:49

普通消息被发送到普通消息队列时， 初始状态为 **Active**，当其被取走后在 VisibilityTimeout 的时间内状态为 **Inactive**，若超过 VisibilityTimeout 时间后消息还未被删除，消息会重新变成 **Active** 状态；如果在 VisibilityTimeout 时间内被删除，消息状态将变为 **Deleted**。消息的最长存活时间由创建队列时指定的 MessageRetentionPeriod 属性值决定， 超过此时间后消息状态变成 **Expired** 并将被回收。
消费者只能取到处于 **Active** 状态的消息。这保证了同一条消息不会同时被多次消费，但可被顺序性地多次消费。



- Component 1 将 Message A 发送到一个队列，该消息在 CMQ 服务器间提供多份冗余。
- 当 Component 2 准备好处理消息时，就从队列检索消息，然后 Message A 返回。在 Message A 处理期间，它仍然停留在队列中，在**取出消息隐藏时长**阶段，其他业务不可获取 Message A。
- Component 2 可从队列删除 Message A，以避免一旦**取出消息隐藏时长**过期后该消息被再次接受并处理；也可以不删除 Message A，该消息可以被其他业务多次消费。

队列和消息标识符

最近更新时间：2020-07-31 17:47:20

使用腾讯云 CMQ 时，用户首先需要熟悉以下三个标识符：队列名称、消息 ID 和接收句柄。

1. 队列名称

创建新队列时，用户必须提供在此地域范围内唯一的队列名称，不同地域间队列名称可以重复。腾讯云 CMQ 使用地域和队列名称唯一标识一个队列，每当用户要对队列执行操作时，都需要提供这两个参数。

2. 消息 ID

每条消息都会收到一个由腾讯云系统分配的消息 ID，该 ID 可由 SendMessage 接口请求中返回给用户。此标识符用于识别消息。需要注意的是，删除消息时用户需要消息的接收句柄，而不是消息 ID。消息 ID 有形如“Msg-XXXXXXXX”的样式。

3. 接收句柄

每当收到来自队列的消息时，用户都会收到该消息的接收句柄。消息句柄始终与接收消息的操作相关联，与消息本身无关。要删除消息或更改消息属性，用户则必须提供接收句柄，而不是消息 ID。这意味着，必须始终先接收消息，然后才能删除/更改它。

> **说明**
> 如果多次接收某条消息，则每次接收该消息时，用户都会获得不同的接收句柄。在请求删除该消息时，您必须提供最近收到的接收句柄，否则可能无法删除该消息。

延迟消息功能

最近更新时间：2021-08-04 11:48:25

CMQ 消息定时器允许您为要添加到队列的消息指定初始的不可见时段，称为**飞行状态**。例如：如果您发送一条消息并将 DelaySeconds 参数设置为45，则使用者在该消息进入队列后的前45秒将看不到该消息。DelaySeconds 的默认值为0。

延迟消息设置范围：指定 Queue 生产消息时，可增加 DelaySeconds 入参，取值范围为0 - 3600秒，即消息最长不可见时长为1小时。若为空，则无延迟效果。

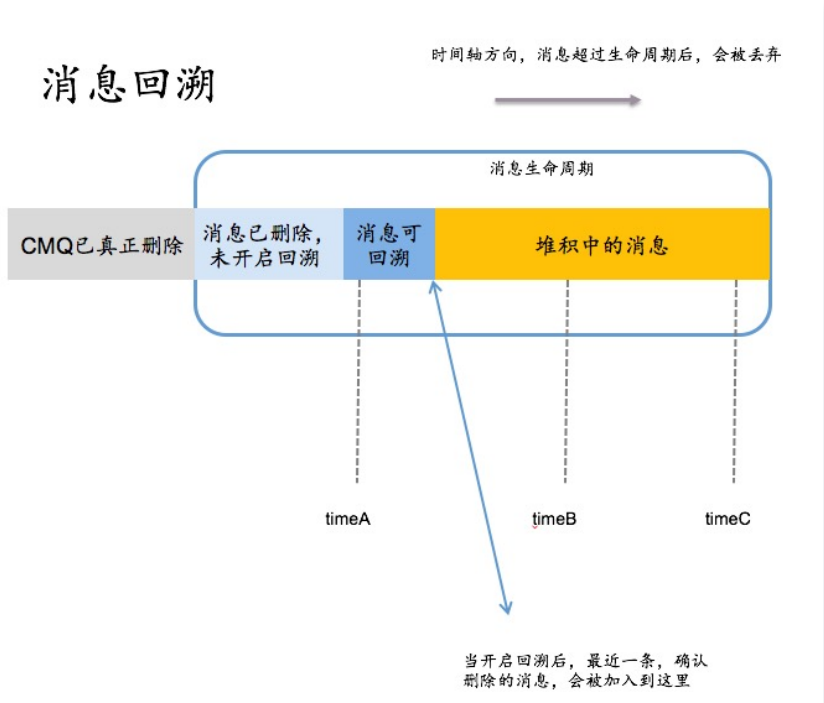
使用限制：每个队列处于飞行状态的消息数限制为2万条，若队列中处于飞行状态的消息超过2万条，则新生产的消息不允许消费，需要等待当前飞行状态的消息到期才可以继续消费。Topic 模式下该能力暂不支持。

消息回溯功能

最近更新时间：2020-07-31 17:41:51

CMQ 提供类似于 Kafka 的消息回溯能力。使用消息回溯，您可在业务成功消费并删除消息后重新消费已删除的消息。此功能便于核心金融业务做业务对账、业务系统重试等操作。

功能说明

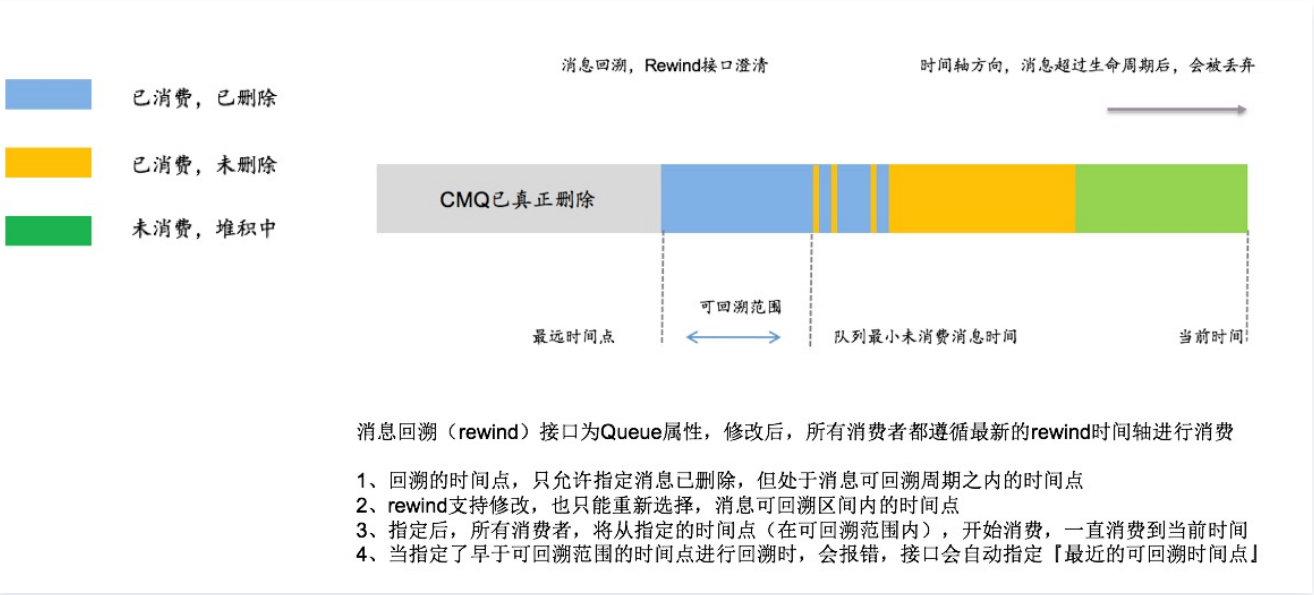


如上图，消息的生命周期为蓝色框内的片段。开启消息回溯能力后，已被消费者消费且确认删除的消息会进入**消息可回溯**区域，CMQ 后端还会保存该信息。但消息超过 Queue 的消息生命周期时（假设为1天），达到生命周期后，该消息会自动删除，不可回溯。具体产品逻辑如下：

- **开启：**若未开启消息回溯能力，则消费者已消费且确认删除的消息会立即删除。开启该功能时，须指定回溯的可回溯周期，可回溯周期的范围必须小于等于消息的生命周期。
- **里程碑：**根据上一条策略，开启消息回溯后，随着消费者的不断消费及删除，可回溯的消息数量会不断增多。
- **关闭：**关闭消息回溯后，消息可回溯区域的消息将被立即删除，且不可回溯。
- **队列属性：**消息回溯是 Queue 的属性，可在创建时或在修改配置处进行设置。指定回溯（rewind）的时间点后，所有消费者都会从该时间点的消息往后消费。
- **计费：**开启消息回溯能力后，可回溯部分的消息会产生一定的堆积费用，单位价格与消息堆积的费用共同计算。
- **指定回溯时间点：**消费者发起回溯消费，需要指定 Queue Name 及具体的回溯时间。且从最远的时间点，往回回溯。时间为 key，不可逆向消费。如图所示，只能从 timeA 到 timeB/timeC 消费，不支持反向消费。
- **指定回溯时间范围：**0 – 15天，控制台开启该能力后，删除的消息才可被回溯。建议关键应用，长期开启消息回溯能力。且消息回溯周期，与消息生命周期设成一致。
- **不可指定堆积中的消息回溯：**若消息仍在堆积中，未被消费，则无法指定某一个具体的位置进行消费。

可回溯范围

最大可回溯时间点 = 当前时间 – 设置的可回溯时长。消息生产时间在这个值之前的不可回溯，之后的可回溯，如下图所示：



消息回溯（rewind）接口为Queue属性，修改后，所有消费者都遵循最新的rewind时间轴进行消费

1、回溯的时间点，只允许指定消息已删除，但处于消息可回溯周期之内的时间点
2、rewind支持修改，也只能重新选择，消息可回溯区间内的时间点
3、指定后，所有消费者，将从指定的时间点（在可回溯范围内），开始消费，一直消费到当前时间
4、当指定了早于可回溯范围的时间点进行回溯时，会报错，接口会自动指定『最近的可回溯时间点』

时间轴

消息回溯 以消息生产的时间为排序标准，与被删除的先后无关。如下图所示：

创建队列

最近更新时间：2021-08-04 11:42:35

在【消息队列 CMQ 】>【队列服务】>【 队列 】中，单击页面左上角【新建】，即可创建一个消息队列（Queue）。
创建队列时，用户需要指定以下属性值：

属性	说明	取值
队列名称	QueueName，为队列的名称。	作为资源的唯一标识，调用 API 接口进行操作时，以 Queue name 为准，创建成功后无法修改。为了防止混淆，不允许创建大小写同名队列，请注意使用时严格区分大小写。
消息生命周期	队列的 msgRetentionSeconds 属性， 消息在本队列中最长的存活时间，从发送到该队列开始经过此参数指定的时间后，不论消息是否被取出过都将被删除。	单位：秒，有效值范围：60 – 1296000秒，即1分钟 – 15天。
消息接收长轮询等待时间	PollingWaitSeconds，长轮询等待时，一个消息消费请求只会在取到有效消息或长轮询超时时才返回响应，类似于 Ajax 请求的长轮询。	单位：秒。有效值范围：200毫秒 – 30秒。默认值为200毫秒。
取出消息隐藏时长	队列的 VisibilityTimeout 属性。每条 Message 都有个默认的 VisibilityTImeout，Worker 在接收到消息后，timeout 就开始计时了。如果 Worker 在 timeout 时间内没能处理完 Message，则消息就有可能被其他 Worker 接收到并处理。	单位：秒。有效值范围：1 – 43200秒，即1秒 – 12小时。默认值默认设为30秒。
消息最大长度	队列的 MaxMsgSize 属性， 限定允许发送到该队列的消息体的最大长度。	单位：byte。有效值范围：1024 – 65536byte，即1KB – 64KB。默认值为64KB。
堆积消息数量上限	该限制为单个队列，最大消息堆积个数（未被删除）。	单个队列的堆积消息上限为1亿条，最小值为1百万条。如需提升额度，请联系技术支持。
消息回溯	若未开启“消息回溯”能力，则消费者已消费，且确认删除的消息，会立即删除，开启该功能时，须指定回溯的“可回溯周期”。	“可回溯周期”的范围，必须小于等于消息的生命周期。建议将回溯周期与消息的生命周期设置为相同的值，便于定位问题。消息回溯功能产生的费用为0.01元/百万条/小时。详情请参考 消息回溯功能 。
指定时间范围	当开启消息回溯后可配置时间范围项。控制台默认不开启。开启后时间默认跟消息生命周期设置相同值。	时间范围：1 – 15天，最大可回溯时间点 = 当前时间 – 设置的可回溯时间范围。消息生产时间在这个值之前的不可回溯。

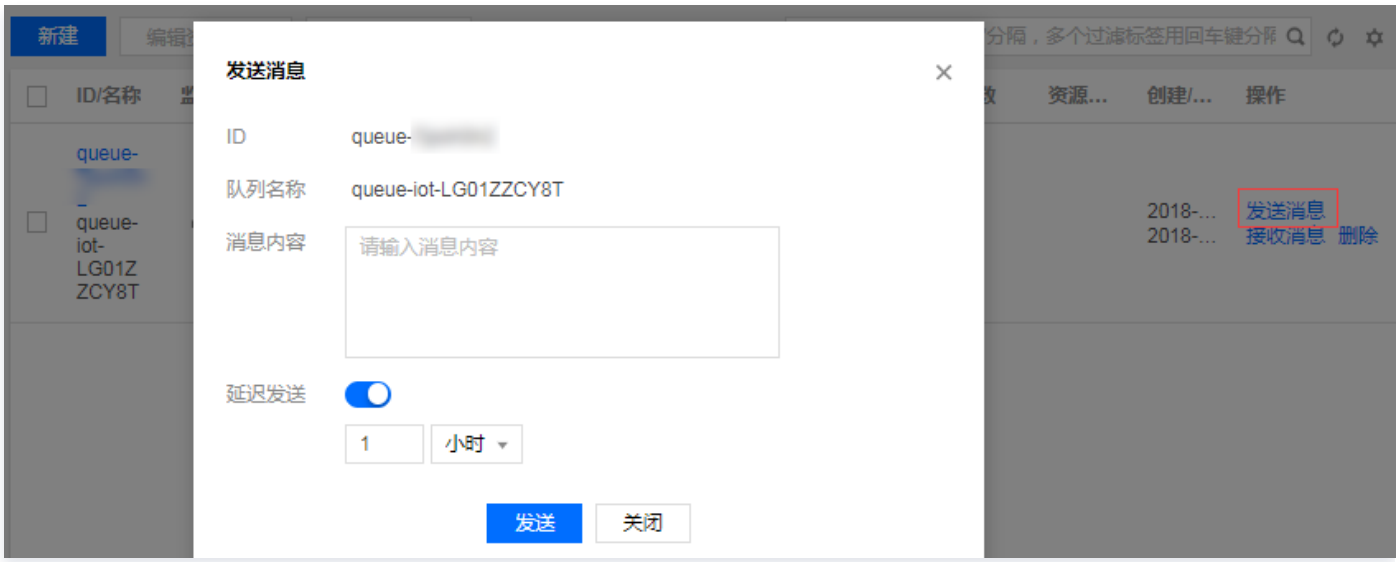
对应控制台界面如下：


新建消息队列
所属地域
广州
队列名称
请输入队列名称、创建后
配置队列属性
消息生命周期
天
范围在1分钟到15天
消息接收长轮询等待时间
0
秒
范围在0到30秒
取出消息隐藏时长
30
秒
范围在1秒到12小时
消息最大长度
64
KB
范围在1到1024KB
堆积消息和回溯设置
堆积消息数量上限
百万
范围在1百万到1亿条
消息回溯
指定时间范围
1
天
范围在1秒到15天
提交
关闭

生产消息

最近更新时间：2020-10-28 21:02:29

1. 登录 消息队列 CMQ 控制台，在左侧导航栏单击【队列服务】>【队列】。
2. 在队列列表中选择目标队列，单击操作列的【发送消息】。
3. 填写消息内容，单击【发送】，向消息接收侧发送测试消息。



- 消息内容：填写发送的内容，至少1Byte，最大长度受限于设置的队列消息最大长度属性。
- 延迟发送：可开启延迟消息，开启后消息会在延迟时间后发送，延迟时间范围为1秒 – 1小时。

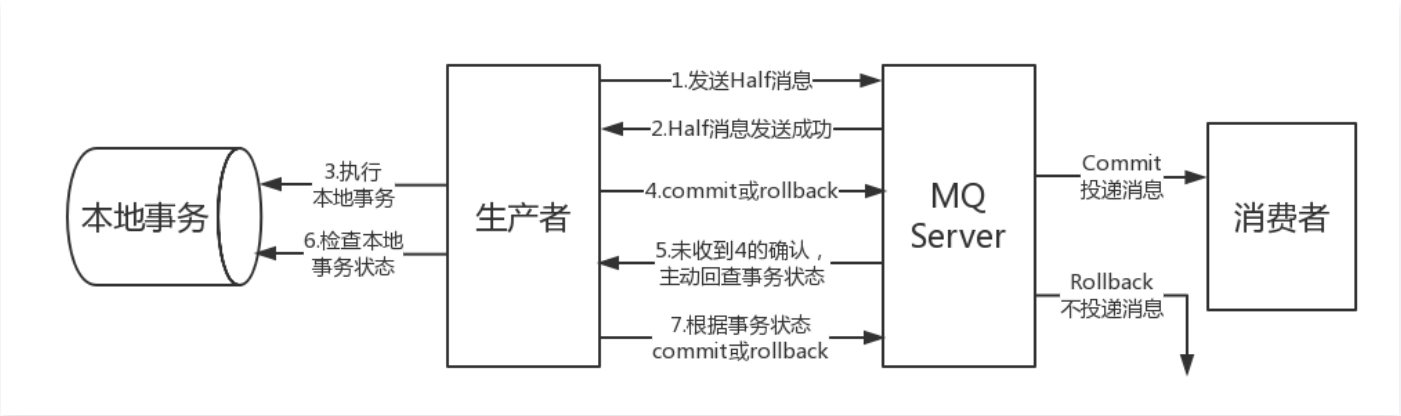
事务消息

最近更新时间：2020-10-28 21:13:46

当消息生产者本地事务处理成功与消息发送成功不一致时，传统的处理方式无法解决该问题，事务消息实现了消息生产者本地事务与消息发送的原子性，保证了消息生产者本地事务处理成功与消息发送成功的最终一致。用户实现类似 X/Open XA 的分布事务功能，通过 CMQ 事务消息能达到分布式事务的最终一致。

> **说明**
> 该功能目前处于灰度测试阶段，暂时支持广州、上海、孟买地域，其他地域在逐步支持中，如需试用请通过 提交工单 的方式开通白名单。

模块交互图



其中，事务消息发送对应步骤1、2、3、4，事务消息回查对应步骤5、6、7。

1. 生产者向 MQ 服务端发送消息（为了方便说明，整个 MQ 服务端用 MQ Server 来表示）。
2. MQ Server 将消息持久化成功之后，向生产者 ACK 确认消息已经发送成功，此时消息为半消息（暂不能投递的消息）。
3. 生产者发送消息成功后，开始执行本地事务逻辑。
4. 生产者根据本地事务执行结果向 MQ Server 提交二次确认（Commit 或是 Rollback），MQ Server 收到 Commit 状态则将半消息标记为可投递，消费者最终将收到该消息；MQ Server 收到 Rollback 状态则删除半消息（直接置消息已消费状态），消费者将不会接受该消息。
5. 在断网或者是生产者应用重启的特殊情况下，上述步骤4提交的二次确认最终未到达 MQ Server，经过固定时间后 MQ Server 将对该消息发起消息回查。
6. 生产者收到消息回查后，需要检查对应消息的本地事务执行的最终结果。生产者根据检查得到的本地事务的最终状态再次提交二次确认，MQ Server 仍按照步骤4对半消息进行操作。

发送事务消息

发送事务消息包含以下两个步骤：

1. 发送半消息并执行本地事务
2. 提交本地事务执行状态

其中，提交本地事务执行状态有两种方式：

- 执行本地事务完成后，SDK 主动提交。
- 执行本地事务后一直没有提交状态，MQ Server 会主动发送回查，此时 SDK 提交本地事务执行状态。

事务执行状态有以下三种情况：

- TransactionStatus.COMMIT 提交事务，消费者可以消费到该消息。

- TransactionStatus.ROLLBACK 回滚事务，消息被丢弃，消费者不会消费到该消息。
- TransactionStatus.UN_KNOW 无法判断状态，等待 MQ Server 再次发送回查。

发送事务消息示例

```
package demo;

import com.qcloud.cmq.client.common.ClientConfig;
import com.qcloud.cmq.client.common.LogHelper;
import com.qcloud.cmq.client.common.ResponseCode;
import com.qcloud.cmq.client.common.TransactionStatus;
import com.qcloud.cmq.client.consumer.Message;
import com.qcloud.cmq.client.exception.MQClientException;
import com.qcloud.cmq.client.producer.*;
import org.slf4j.Logger;

import java.util.ArrayList;
import java.util.List;

public class ProducerTransactionDemo {

    private final static Logger logger = LogHelper.getLog();

    public static void main(String[] args) {
        TransactionProducer producer = new TransactionProducer();
        // 设置 Name Server地址，在控制台上获取， 必须设置
        producer.setNameServerAddress("http://cmq-nameserver-dev.api.tencentyun.com");
        // 设置 SecretId，在控制台上获取，必须设置
        producer.setSecretId("AKID******w7D7");
        // 设置 SecretKey，在控制台上获取，必须设置
        producer.setSecretKey("qV2N******TgiY");
        // 设置签名方式，可以不设置，默认为 SHA1
        producer.setSignMethod(ClientConfig.SIGN_METHOD_SHA256);
        // 设置发送消息失败时，重试的次数，设置为0表示不重试，默认为2
        producer.setRetryTimesWhenSendFailed(3);
        // 设置请求超时时间， 默认3000ms
        producer.setRequestTimeoutMS(5000);
        // 设置首次回查的时间间隔，默认5000ms
        producer.setFirstCheckInterval(5);
        // 消息发往的队列，在控制台创建
        String queue = "test_transaction";
        // 设置回查的回调函数，检查本地事务的校验结果
        producer.setChecker(queue, new TransactionStatusCheckerImpl());


        try{
            // 启动对象前必须设置好相关参数
            producer.start();
            String msg = "test_message";

            TransactionExecutor executor = new TransactionExecutor() {
                @Override
                public TransactionStatus execute(String msg, Object arg) {
                    //执行用户的本地事务 ... ...
                    System.out.println("do local transaction service!");
```

```
                return TransactionStatus.COMMIT;
            }
        };

        // 同步发送单条事务消息
        SendResult result = producer.sendTransactionMessage(queue, msg, executor, "test");

        if (result.getReturnCode() == ResponseCode.SUCCESS) {
            System.out.println("==> send success! msg_id:" + result.getMsgId() + " request_id:" +
result.getRequestId());
        } else {
            System.out.println("==> code:" + result.getReturnCode() + " error:" + result.getErrorMsg());
        }

    }catch (MQClientException e){
        e.printStackTrace();
    }

  }
}

class TransactionStatusCheckerImpl implements TransactionStatusChecker{

  @Override
  public TransactionStatus checkStatus(Message msg) {
    //用户实现，检查本地事务的执行状态 ... ...
    return TransactionStatus.COMMIT;
  }
}
```

消费事务消息

消费事务消息与从普通队列或者订阅中消费一致，详情请参考 队列模型消费消息 。

死信队列

最近更新时间：2023-06-05 16:51:52

> **说明**
> 该功能目前处于灰度测试阶段，如需试用请通过 提交工单 的方式开通白名单。

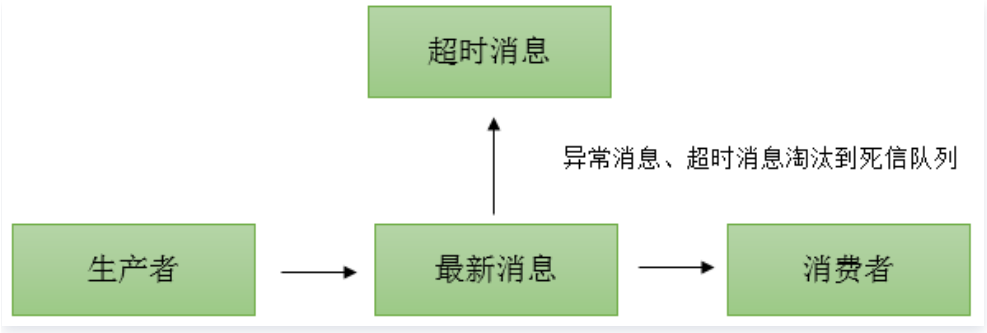
死信队列（Dead-Letter-Queue，DLQ）用于处理无法被正常消费的消息。达到最大重试次数后，若消费依然失败，则表明消费者在正常情况下无法正确地消费该消息，此时 CMQ 不会立刻将消息丢弃，而是将其发送到该消费者对应的特殊队列中，这个特殊队列就是 DLQ。对新建的队列，或存量的队列，您都可以启用死信队列。

应用场景

- **定位问题**：例如，某条消息被多次消费后却未被删除，一般是由于该消息未被正确消费，可能存在问题需要回溯定位。 您可以设置最大接收次数，超额后该消息会被淘汰到指定的死信队列，便于后续问题发现。
- **优先级队列**：例如，摩拜单车等 O2O 客户，对访问时延、实时性要求非常高。单车开锁逻辑中，当 CMQ 堆积了1亿条消息时，会优先处理最新生产的部分消息，老的消息淘汰到死信队列，在消费者有能力时再处理。
  老的消息（如用户扫码开车场景）在等待时已流失了用户，老的消息的价值会偏小，建议优先处理最新消息。

技术原理

客户端消费源队列信息，broker 检测是否满足死信策略配置的触发条件，若满足则投递到死信队列。



死信策略：触发死信消息的方式。
最大接收次数：将消息发送到死信队列前允许接收该消息的最大次数，支持设定值为1 – 1000次。
最大未消费时间：将消息发送到死信队列前达到了最大未消费时间，允许设置5分钟 – 12小时。

> **注意**
> 当 CMQ 触发高可用性时，CMQ 服务器 leader 切换有可能会导致计算消费次数不准确。

使用规则

- 一个队列只能绑定一个死信队列。
- 绑定的死信队列必须是同地域、同账号下的队列。
- 如果死信队列已被其他队列绑定，则不能直接删除该死信队列。
- 开启事务消息的队列，不能成为其他队列的死信队列。
- 多个队列（最多6个）可以指定同一个队列作为死信队列，但死信队列本身不能再指定其他队列作为死信队列，避免嵌套。
- 可以在源队列（客户端生产、消费的数据队列）中解绑死信队列，解绑后无其他队列绑定，可删除死信队列。
- 一个已存在的普通队列在 T 时刻被指定为死信队列，若在 T 时刻之前队列中还存在未被消费的消息，需要将 T 时刻之前的消息全部消费完成后才可以触发死信策略。