密 级：内部公开

金蝶Apusic消息队列产品使用指南

深圳市金蝶天燕云计算股份有限公司对本文件资料享受著作权及其它专属权利，未经书面许可，不得将该等文件资料（其全部或任何部分）披露予任何第三方，或进行修改后使用。

|  |  |  |  |
| --- | --- | --- | --- |
| 文件更改摘要： | | | |
| 日期 | 版本号 | 修订说明 | 修订人 |
| 2022/10/12 | 2.2 | 初版 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[第1章 快速上手 4](#_Toc134638973)

[1.1 Java客户端样例 4](#_Toc134638974)

[2.1.1 普通生产消息 4](#_Toc134638975)

[2.1.2 普通消费消息 5](#_Toc134638976)

[2.1.3 定时消息 6](#_Toc134638977)

[2.1.3 延时消息 8](#_Toc134638978)

[2.1.4 事务消息 9](#_Toc134638979)

[2.1.5 顺序消息 12](#_Toc134638980)

[1.2 Go客户端样例 17](#_Toc134638981)

[2.3.1普通生产消息 18](#_Toc134638982)

[2.3.2 普通消费消息 19](#_Toc134638983)

[2.3.3 延时/定时消息 20](#_Toc134638984)

[2.3.4 事务消息 22](#_Toc134638985)

[2.3.5 顺序消息 25](#_Toc134638986)

# 快速上手

## Java客户端样例

首先准备sdk rocketmq-client.zip. 解压导入maven工程，选择jdk1.8版本。

前置操作：1.管控台已经创建好集群，并且成功启动集群。2.确保客户端与服务端能够通信，默认服务不修改端口的话，使用telnet检查9876、10911端口；

常见参数含义：

accesskey： 分为集群超管token。用户token获取方式请移步 1.4.5、1.5.2；

secretkey：组成为“用户名+123456”,例：用户名为xxxx的secretkey为“xxxx123456”，超管用户默认为 “admq123456”；

namespace：在管控台界面上创建的命名空间名称；

groupname：在管控台界面上创建的消费者组或者生产者组名；

topic：在管控台界面新建的topic；

nameserver：获取方式为集群详情具体获取请参照管控台界面：选择“集群管理”页签，点击集群列表中需要查看的集群名字跳转集群详情页面；查看“rocketmq服务地址”,值得注意的是代码若是集群使用时需要使用英文“;”号隔开；

tag：发送消息指定的tag；

Key：发送消息指定的key；

常用鉴权方法：

AclRpcHook：AclClientRPCHook aclClientRPCHook = new AclClientRPCHook(new SessionCredentials("accesskey", "secretkey"));

### 2.1.1 普通生产消息

使用rocketmq-client进行消息生产，下面是java demo。

1. **public** **static** **void** main(String[] args)**throws** Exception{
2. // 实例化消息生产者Producer
3. AclClientRPCHook aclClientRPCHook = **new** AclClientRPCHook(**new** SessionCredentials("accesskey", "secretkey"));
4. DefaultMQProducer producer = **new** DefaultMQProducer("namespace","groupname",aclClientRPCHook);
5. // 设置NameServer的地址
6. producer.setNamesrvAddr("nameserver");
7. // 启动Producer实例
8. producer.start();
9. **for** (**int** i = 0; i < 10; i++) {
10. // 创建消息，并指定Topic，Tag和消息体
11. Message msg = **new** Message("topic" /\* Topic \*/,
12. "tag" /\* Tag \*/,
13. ("Hello RocketMQ " + i).getBytes(RemotingHelper.DEFAULT\_CHARSET) /\* Message body \*/
14. );
15. // 发送消息到一个Broker
16. SendResult sendResult = producer.send(msg);
17. // 通过sendResult返回消息是否成功送达
18. System.out.printf("%s%n",(System.currentTimeMillis()/1000)+":"+sendResult);
19. }
20. // 如果不再发送消息，关闭Producer实例。
21. producer.shutdown();
22. }

### 2.1.2 普通消费消息

1. **public** **static** **void** main(String[] args) **throws** InterruptedException, MQClientException {
3. AclClientRPCHook aclClientRPCHook = **new** AclClientRPCHook(**new** SessionCredentials("accesskey", "secretkey"));
4. DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer("namespace","groupname",aclClientRPCHook);
5. consumer.setNamesrvAddr("nameserver");
7. consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME\_FROM\_LAST\_OFFSET);
8. AtomicLong count = **new** AtomicLong(0L);
9. ConcurrentHashMap<String, Integer> map = **new** ConcurrentHashMap<>();
10. consumer.subscribe("topic", "\*");
12. consumer.registerMessageListener(**new** MessageListenerConcurrently() {
13. @Override
14. **public** ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
15. ConsumeConcurrentlyContext context) {
16. msgs.forEach((msg) -> {
17. **try** {
18. String time = (System.currentTimeMillis() / 1000) + "";
19. System.out.printf("time:%s,Msg payload: %s, topic is:%s, MsgId is:%s%n",time,
20. **new** String(msg.getBody(), "GBK"), msg.getTopic(), msg.getMsgId());
21. Integer count1 = map.get(time);
22. **if** (count1 == **null**){
23. count1 = 1;
24. }**else**{
25. count1++;
26. }
27. map.put(time,count1);
29. } **catch** (UnsupportedEncodingException e) {
30. e.printStackTrace();
31. }
32. System.out.println("total ====> " + count.get());
33. });
34. **return** ConsumeConcurrentlyStatus.CONSUME\_SUCCESS;
35. }
36. });
37. consumer.start();
38. System.out.printf("Consumer Started.%n");
39. }

### 2.1.3 定时消息

使用定时消息的示例代码。

Producer 将消息发送到消息队列 MQ 服务端，但并不期望这条消息立马投递，而是推迟到在当前时间点之后的某一个时间投递到 Consumer 进行消费，该消息即定时消息。

生产者：

1. import java.nio.charset.StandardCharsets;
2. import org.apache.rocketmq.client.producer.DefaultMQProducer;
3. import org.apache.rocketmq.common.message.Message;
4. public class EveryTimeProducer {
5. /\*\*
6. 生产者
7. \*/
8. public static void main(String[] args) throws Exception {
9. //DefaultMQProducer producer = new DefaultMQProducer("namespace","groupname",Global.getAclRPCHook());
10. DefaultMQProducer producer = new DefaultMQProducer("test2");
11. producer.setNamesrvAddr("nameserver");
12. producer.start();
13. final byte[] bytes = "this is cutomer 2m2s message" .getBytes(StandardCharsets.UTF\_8);
14. Message message = new Message("topic", "\*", bytes);
15. message.putUserProperty("SPECIFY\_DELAY\_TIME\_SECONDS", "11");//延迟多少秒
16. message.putUserProperty("SPECIFY\_DELAY\_TIME\_TIMESTAMP", "1663641660");
17. producer.sendOneway(message);
18. producer.shutdown();
19. }
20. }

消费者

1. import com.apusic.admq.rocketmq.example.Global;
2. import java.nio.charset.StandardCharsets;
3. import java.util.Date;
4. import java.util.List;
5. import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
6. import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
7. import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
8. import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
9. import org.apache.rocketmq.client.exception.MQClientException;
10. import org.apache.rocketmq.common.consumer.ConsumeFromWhere;
11. import org.apache.rocketmq.common.message.MessageExt;
12. public class EveryTimeConsumer {
13. /\*\*
14. 消费端
15. \*/
16. public static void main(String[] args) throws MQClientException {
17. DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("namespace","groupname",Global.getAclRPCHook());
18. //DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("groupname");
19. consumer.setNamesrvAddr("nameserver");
20. consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME\_FROM\_FIRST\_OFFSET);
21. //Topic的名称
22. consumer.subscribe("topic", "\*");
23. consumer.setSuspendCurrentQueueTimeMillis(30);
24. consumer.registerMessageListener(new MessageListenerConcurrently() {
25. @Override
26. public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> list, ConsumeConcurrentlyContext consumeConcurrentlyContext) {
27. for (MessageExt ext : list) {
28. final long time = System.currentTimeMillis() - ext.getBornTimestamp();
29. System.out.println("消息\t" + new String(ext.getBody(), StandardCharsets.UTF\_8) + "接收到的消息间隔消息\t" + time / 1000.0 );
30. Date date = new Date();
31. System.out.println("接收到消息时间："+date);
32. }
33. return ConsumeConcurrentlyStatus.CONSUME\_SUCCESS;
34. }
35. });
36. consumer.start();
37. }
38. }

### 2.1.3 延时消息

使用延时消息的示例代码。延时消息用于指定消息发送到消息队列的服务端后，延时一段时间才被投递到客户端，功能主要用于解决一些消息生产和消费有时间窗口要求的场景。其次是让消息投递后触发任务的场景。

关于延时级别说明：

1. import com.apusic.admq.rocketmq.example.Global;
2. import java.util.List;
3. import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
4. import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
5. import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
6. import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
7. import org.apache.rocketmq.common.message.MessageExt;
8. public class DelayConsumer {
9. public static void main(String[] args) throws Exception {
10. // 实例化消费者
11. DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("groupname");
12. consumer.setNamesrvAddr("nameserver");
13. // 订阅Topics 延时topic
14. consumer.subscribe("topic", "\*");
15. // 注册消息监听者
16. consumer.registerMessageListener(new MessageListenerConcurrently() {
17. @Override
18. public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> messages, ConsumeConcurrentlyContext context) {
19. for (MessageExt message : messages) {
20. // Print approximate delay time period
21. System.out.println("Receive message[msgId=" + message.getMsgId() + "] " + (System.currentTimeMillis() - message.getBornTimestamp()) + "ms later");
22. }
23. return ConsumeConcurrentlyStatus.CONSUME\_SUCCESS;
24. }
25. });
26. // 启动消费者
27. consumer.start();
28. System.out.println("DelayConsumer Started.");
29. }
30. }

### 2.1.4 事务消息

1. import java.util.Date;
2. import java.util.concurrent.ConcurrentHashMap;
3. import java.util.concurrent.atomic.AtomicInteger;
4. import org.apache.rocketmq.client.producer.LocalTransactionState;
5. import org.apache.rocketmq.client.producer.TransactionListener;
6. import org.apache.rocketmq.common.message.Message;
7. import org.apache.rocketmq.common.message.MessageExt;
8. public class TransactionListenerImpl implements TransactionListener {
9. private AtomicInteger transactionIndex = new AtomicInteger(1);
10. private ConcurrentHashMap<String, Integer> localTrans = new ConcurrentHashMap<>();
11. @Override
12. public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
13. // 模拟本地事务执行，并将结果存入localTransMap中
14. System.out.printf("%s执行本地事务msg：%s%n,arg:%s\n", new Date(System.currentTimeMillis()),msg.toString(), arg);
15. int value = transactionIndex.getAndIncrement();
16. int status = value % 3;
17. localTrans.put(msg.getTransactionId(), status);
18. // 返回Unknown，触发事务回查
19. return LocalTransactionState.UNKNOW;
20. }
21. @Override
22. public LocalTransactionState checkLocalTransaction(MessageExt msg) {
23. // 根据msg.getTransactionId()回查事务状态，来决定是提交还是回滚
24. System.out.printf("%s,事务结果回查%s%n\n", new Date(System.currentTimeMillis()),msg.toString());
25. Integer status = localTrans.get(msg.getTransactionId());
26. if (null != status) {
27. switch (status) {
28. case 0:
29. System.out.println("UNKNOW");
30. return LocalTransactionState.UNKNOW;
31. case 1:
32. System.out.println("提交消息");
33. return LocalTransactionState.COMMIT\_MESSAGE;
34. case 2:
35. System.out.println("回滚消息");
36. return LocalTransactionState.ROLLBACK\_MESSAGE;
37. default:
38. System.out.println("提交消息");
39. return LocalTransactionState.COMMIT\_MESSAGE;
40. }
41. }
42. System.out.println("提交消息");
43. return LocalTransactionState.COMMIT\_MESSAGE;
44. }
45. }
46. #######################################################################
47. import com.apusic.admq.rocketmq.example.Global;
48. import java.io.UnsupportedEncodingException;
49. import java.util.Date;
50. import java.util.concurrent.ArrayBlockingQueue;
51. import java.util.concurrent.ExecutorService;
52. import java.util.concurrent.ThreadFactory;
53. import java.util.concurrent.ThreadPoolExecutor;
54. import java.util.concurrent.TimeUnit;
55. import org.apache.rocketmq.client.exception.MQClientException;
56. import org.apache.rocketmq.client.producer.SendResult;
57. import org.apache.rocketmq.client.producer.TransactionListener;
58. import org.apache.rocketmq.client.producer.TransactionMQProducer;
59. import org.apache.rocketmq.common.message.Message;
60. import org.apache.rocketmq.remoting.common.RemotingHelper;
61. public class TransactionProducer {
62. public static void main(String[] args) throws MQClientException, InterruptedException {
63. // 事务监听器
64. TransactionListener transactionListener = new TransactionListenerImpl();
65. // 生产者
66. TransactionMQProducer producer = new TransactionMQProducer("groupname");//生产者groupname
67. // 事务状态回查用的线程池
68. ExecutorService executorService = new ThreadPoolExecutor(2, 5, 100, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(2000), new ThreadFactory() {
69. @Override
70. public Thread newThread(Runnable r) {
71. Thread thread = new Thread(r);
72. thread.setName("client-transaction-msg-check-thread");
73. return thread;
74. }
75. });
76. producer.setExecutorService(executorService);
77. producer.setTransactionListener(transactionListener);
78. producer.setNamesrvAddr("nameserver");
79. producer.start();
80. // 发送事务消息
81. for (int i = 0; i < 1; i++) {
82. try {
83. Message msg =
84. new Message("topic", "tag", "key" + i,
85. ("Hello RocketMQ " + i).getBytes(RemotingHelper.DEFAULT\_CHARSET));
86. SendResult sendResult = producer.sendMessageInTransaction(msg, null);
87. System.out.printf("%s,半消息发送结果%s%n",new Date(System.currentTimeMillis()),sendResult);
88. Thread.sleep(10);
89. } catch (MQClientException | UnsupportedEncodingException e) {
90. e.printStackTrace();
91. }
92. }
93. // 让主线程等待，以便执行事务状态回查
94. for (int i = 0; i < 100000; i++) {
95. Thread.sleep(1000);
96. }
97. producer.shutdown();
98. }
99. }

### 2.1.5 顺序消息

1. import com.apusic.admq.rocketmq.example.Global;
2. import java.text.SimpleDateFormat;
3. import java.util.ArrayList;
4. import java.util.Date;
5. import java.util.List;
6. import org.apache.rocketmq.client.producer.DefaultMQProducer;
7. import org.apache.rocketmq.client.producer.MessageQueueSelector;
8. import org.apache.rocketmq.client.producer.SendResult;
9. import org.apache.rocketmq.common.message.Message;
10. import org.apache.rocketmq.common.message.MessageQueue;
11. public class OrderProducer {
12. public static void main(String[] args) throws Exception {
13. DefaultMQProducer producer = new DefaultMQProducer("groupname");//生产groupname
14. producer.setNamesrvAddr("nameserver");
15. producer.start();
16. String[] tags = new String[]{"TagA", "TagC", "TagD"};//消息tag
17. // 订单列表
18. List<OrderStep> orderList = new OrderProducer().buildOrders();
19. Date date = new Date();
20. SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
21. String dateStr = sdf.format(date);
22. for (int i = 0; i < 10; i++) {
23. // 加个时间前缀
24. String body = dateStr + " Hello RocketMQ " + orderList.get(i);
25. Message msg = new Message("topic", tags[i % tags.length], "key" + i, body.getBytes());//顺序消息topic，消息key
26. SendResult sendResult = producer.send(msg, new MessageQueueSelector() {
27. @Override
28. public MessageQueue select(List<MessageQueue> mqs, Message msg, Object arg) {
29. Long id = (Long) arg; //根据订单id选择发送queue
30. long index = id % mqs.size();
31. return mqs.get((int) index);
32. }
33. }, 1l);//订单id
34. System.out.println(String.format("SendResult status:%s, queueId:%d, body:%s",
35. sendResult.getSendStatus(),
36. sendResult.getMessageQueue().getQueueId(),
37. body));
38. }
39. producer.shutdown();
40. }
41. /\*\*
42. \* 订单的步骤
43. \*/
44. private static class OrderStep {
45. private long orderId;
46. private String desc;
47. public long getOrderId() {
48. return orderId;
49. }
50. public void setOrderId(long orderId) {
51. this.orderId = orderId;
52. }
53. public String getDesc() {
54. return desc;
55. }
56. public void setDesc(String desc) {
57. this.desc = desc;
58. }
59. @Override
60. public String toString() {
61. return "OrderStep{" +
62. "orderId=" + orderId +
63. ", desc='" + desc + '\'' +
64. '}';
65. }
66. }
67. /\*\*
68. \* 生成模拟订单数据
69. \*/
70. private List<OrderStep> buildOrders() {
71. List<OrderStep> orderList = new ArrayList<OrderStep>();
72. OrderStep orderDemo = new OrderStep();
73. orderDemo.setOrderId(15103111039L);
74. orderDemo.setDesc("创建");
75. orderList.add(orderDemo);
76. orderDemo = new OrderStep();
77. orderDemo.setOrderId(15103111065L);
78. orderDemo.setDesc("创建");
79. orderList.add(orderDemo);
80. orderDemo = new OrderStep();
81. orderDemo.setOrderId(15103111039L);
82. orderDemo.setDesc("付款");
83. orderList.add(orderDemo);
84. orderDemo = new OrderStep();
85. orderDemo.setOrderId(15103117235L);
86. orderDemo.setDesc("创建");
87. orderList.add(orderDemo);
88. orderDemo = new OrderStep();
89. orderDemo.setOrderId(15103111065L);
90. orderDemo.setDesc("付款");
91. orderList.add(orderDemo);
92. orderDemo = new OrderStep();
93. orderDemo.setOrderId(15103117235L);
94. orderDemo.setDesc("付款");
95. orderList.add(orderDemo);
96. orderDemo = new OrderStep();
97. orderDemo.setOrderId(15103111065L);
98. orderDemo.setDesc("完成");
99. orderList.add(orderDemo);
100. orderDemo = new OrderStep();
101. orderDemo.setOrderId(15103111039L);
102. orderDemo.setDesc("推送");
103. orderList.add(orderDemo);
104. orderDemo = new OrderStep();
105. orderDemo.setOrderId(15103117235L);
106. orderDemo.setDesc("完成");
107. orderList.add(orderDemo);
108. orderDemo = new OrderStep();
109. orderDemo.setOrderId(15103111039L);
110. orderDemo.setDesc("完成");
111. orderList.add(orderDemo);
112. return orderList;
113. }
114. }

消费者

1. import com.apusic.admq.rocketmq.example.Global;
2. import java.util.List;
3. import java.util.Random;
4. import java.util.concurrent.TimeUnit;
5. import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
6. import org.apache.rocketmq.client.consumer.listener.ConsumeOrderlyContext;
7. import org.apache.rocketmq.client.consumer.listener.ConsumeOrderlyStatus;
8. import org.apache.rocketmq.client.consumer.listener.MessageListenerOrderly;
9. import org.apache.rocketmq.common.consumer.ConsumeFromWhere;
10. import org.apache.rocketmq.common.message.MessageExt;
11. public class OrderConsumer {
12. public static void main(String[] args) throws Exception {
13. DefaultMQPushConsumer consumer = new
14. DefaultMQPushConsumer("groupname");//消费组groupname
15. consumer.setNamesrvAddr("nameserver");
16. /\*\*
17. \* 设置Consumer第一次启动是从队列头部开始消费还是队列尾部开始消费<br>
18. \* 如果非第一次启动，那么按照上次消费的位置继续消费
19. \*/
20. consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME\_FROM\_FIRST\_OFFSET);
21. consumer.subscribe("topic", "TagA || TagC || TagD");//顺序消息topic,消息tag
22. consumer.registerMessageListener(new MessageListenerOrderly() {
23. Random random = new Random();
24. @Override
25. public ConsumeOrderlyStatus consumeMessage(List<MessageExt> msgs, ConsumeOrderlyContext context) {
26. context.setAutoCommit(true);
27. for (MessageExt msg : msgs) {
28. // 可以看到每个queue有唯一的consume线程来消费, 订单对每个queue(分区)有序
29. System.out.println("consumeThread=" + Thread.currentThread().getName() + ",queueId=" + msg.getQueueId() + ", content:" + new String(msg.getBody()));
30. }
31. try {
32. //模拟业务逻辑处理中...
33. TimeUnit.SECONDS.sleep(random.nextInt(10));
34. } catch (Exception e) {
35. e.printStackTrace();
36. }
37. return ConsumeOrderlyStatus.SUCCESS;
38. }
39. });
40. consumer.start();
41. System.out.println("Consumer Started.");
42. }
43. }

## Go客户端样例

首先准备sdk rocketmq-client-go.zip. 解压安装并引入到go.mod中或者可以直接用vendor将源代码放入到vendor中。

前置操作：1.管控台已经创建好集群，并且成功启动集群。2.确保客户端与服务端能够通信，默认服务不修改端口的话，使用telnet检查9876、10911端口；

常见参数含义：

accesskey： 分为集群超管token。用户token获取方式请移步 1.4.5、1.5.2；

secretkey：组成为“用户名+123456”,例：用户名为xxxx的secretkey为“xxxx123456”，超管用户默认为 “admq123456”；

namespace：在管控台界面上创建的命名空间名称；

groupname：在管控台界面上创建的消费者组或者生产者组名；

topic：在管控台界面新建的topic；

nameserver：获取方式为集群详情具体获取请参照管控台界面：选择“集群管理”页签，点击集群列表中需要查看的集群名字跳转集群详情页面；查看“rocketmq服务地址”，追加到数组中即可。

常用鉴权方法：

producer.WithCredentials(primitive.Credentials{ SecretKey: "{secretKey}", AccessKey: "{accessKey}", })

### 2.3.1普通生产消息

使用rocketmq-client-go进行消息生产，下面是go demo。

1. import (
2. "context"
3. "fmt"
5. "gitlab.apusic.net/admq/rocketmq-client-go"
6. "gitlab.apusic.net/admq/rocketmq-client-go/primitive"
7. "gitlab.apusic.net/admq/rocketmq-client-go/producer"
8. )
10. // 生产普通消息
11. func ProduceCommonMessage() {
12. // 创建消息生产者
13. p, err := rocketmq.NewProducer(
14. // 设置服务地址
15. producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"{nameServer}"})),
16. // 设置acl权限
17. producer.WithCredentials(primitive.Credentials{
18. SecretKey: "{secretKey}",
19. AccessKey: "{accessKey}",
20. }),
21. // 设置生产组
22. producer.WithGroupName("{groupName}"),
23. // 设置命名空间名称
24. producer.WithNamespace("{nameSpace}"),
25. // 设置发送失败重试次数
26. producer.WithRetry(2),
27. )
28. **if** err != nil {
29. fmt.Printf("new producer error: %s\n", err.Error())
30. **return**
31. }
32. // 启动producer
33. err = p.Start()
34. **if** err != nil {
35. fmt.Printf("start producer error: %s\n", err.Error())
36. **return**
37. }
38. // 发送消息
39. result, err := p.SendSync(context.Background(), &primitive.Message{
40. Topic: "{topicName}",
41. Body:  []byte("{message}"),
42. })
43. **if** err != nil {
44. fmt.Printf("send message error: %s\n", err.Error())
45. **return**
46. }
47. fmt.Printf("send message success: result=%s\n", result.String())
48. //关闭生产者
49. p.Shutdown()
50. **return**
51. }

### 2.3.2 普通消费消息

1. import (
2. "context"
3. "fmt"
4. "os"
6. "gitlab.apusic.net/admq/rocketmq-client-go"
7. "gitlab.apusic.net/admq/rocketmq-client-go/consumer"
8. "gitlab.apusic.net/admq/rocketmq-client-go/primitive"
9. )
10. // 消费消息
11. func Consumer() {
12. ch := make(chan **struct**{})
13. // 创建消息消费者
14. c, err := rocketmq.NewPushConsumer(
15. // 设置服务地址
16. consumer.WithNameServer([]string{"{nameServer}"}),
17. consumer.WithConsumerModel(consumer.Clustering),
18. // 设置消费组
19. consumer.WithGroupName("{groupName}"),
20. // 设置acl权限
21. consumer.WithCredentials(primitive.Credentials{
22. SecretKey: "{secretKey}",
23. AccessKey: "{accessKey}",
24. }),
25. // 设置命名空间名称
26. consumer.WithNamespace("{nameSpace}"),
27. )
28. // 关闭消费者
29. defer c.Shutdown()
30. **if** err != nil {
31. fmt.Printf("newPushConsumer Err:%s", err.Error())
32. **return**
33. }
34. // 订阅topic
35. err = c.Subscribe("{topicName}", consumer.MessageSelector{},
36. func(ctx context.Context, msgs ...\*primitive.MessageExt) (consumer.ConsumeResult, error) {
37. **for** i := range msgs {
38. fmt.Printf("subscribe callback : %v \n", msgs[i])
39. }
40. **return** consumer.ConsumeSuccess, nil
41. })
43. **if** err != nil {
44. fmt.Printf("subscribe message error: %s\n", err.Error())
45. }
47. // 启动consumer
48. err = c.Start()
50. **if** err != nil {
51. fmt.Printf("consumer start error: %s\n", err.Error())
52. os.Exit(-1)
53. }
54. //阻塞主协程
55. <-ch
56. }

### 2.3.3 延时/定时消息

使用定时消息的示例代码。

Producer 将消息发送到消息队列 MQ 服务端，但并不期望这条消息立马投递，而是推迟到在当前时间点之后的某一个时间投递到 Consumer 进行消费，该消息即定时/延时消息。

1. import (
2. "context"
3. "fmt"
5. "gitlab.apusic.net/admq/rocketmq-client-go"
6. "gitlab.apusic.net/admq/rocketmq-client-go/primitive"
7. "gitlab.apusic.net/admq/rocketmq-client-go/producer"
8. )
10. // 生产定时消息
11. func ProduceDelayedMessage() {
12. // 创建消息生产者
13. p, err := rocketmq.NewProducer(
14. // 设置服务地址
15. producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"{nameServer}"})),
16. // 设置acl权限
17. producer.WithCredentials(primitive.Credentials{
18. SecretKey: "{secretKey}",
19. AccessKey: "{accessKey}",
20. }),
21. // 设置生产组
22. producer.WithGroupName("{groupName}"),
23. // 设置命名空间名称
24. producer.WithNamespace("{nameSpace}"),
25. // 设置发送失败重试次数
26. producer.WithRetry(2),
27. )
28. **if** err != nil {
29. fmt.Printf("new producer error: %s\n", err.Error())
30. **return**
31. }
32. // 启动producer
33. err = p.Start()
34. **if** err != nil {
35. fmt.Printf("start producer error: %s\n", err.Error())
36. **return**
37. }
38. msg := &primitive.Message{
39. //topic名称
40. Topic: "{topicName}",
41. //消息内容
42. Body: []byte("{message}"),
43. }
44. // 级别 1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h
45. // 3是10s以此类推
46. msg = msg.WithDelayTimeLevel(3)
47. // 发送消息
48. result, err := p.SendSync(context.Background(), msg)
49. **if** err != nil {
50. fmt.Printf("send message error: %s\n", err.Error())
51. **return**
52. }
54. fmt.Printf("send message success: result=%s\n", result.String())
55. p.Shutdown()
56. **return**
57. }

### 2.3.4 事务消息

1、客户端Producer发送事务消息

2、收到服务器返回的确认消息。此时消息是半事务消息，消息处于不可投递状态

4、执行本地事务逻辑

4、客户端Producer根据本地事务执行状态向服务器请求commit或rollback，

- 服务端确认结果commit，半事务消息转为可投递，并进行投递

- 服务端确认结果rollback：回滚事务，将半事务消息清除

5、如因其他情况，服务端迟迟没有收到客户端请求或服务端收到的请求数据异常，在一段时间后，服务端会对客户端Producer发起回查。

1. import (
2. "context"
3. "fmt"
4. "os"
5. "sync"
6. "sync/atomic"
7. "time"
9. "gitlab.apusic.net/admq/rocketmq-client-go"
10. "gitlab.apusic.net/admq/rocketmq-client-go/primitive"
11. "gitlab.apusic.net/admq/rocketmq-client-go/producer"
12. )
14. type Trans **struct** {
15. localTrans       \*sync.Map
16. transactionIndex int32
17. }
19. // 创建事务检查对象
20. func NewTrans() \*Trans {
21. **return** &Trans{
22. localTrans: **new**(sync.Map),
23. }
24. }
26. // CheckLocalTransaction 检查本地事务
27. func (tr \*Trans) CheckLocalTransaction(msg \*primitive.MessageExt) primitive.LocalTransactionState {
28. fmt.Printf("%v msg transactionID : %v\n", time.Now(), msg.TransactionId)
29. v, existed := tr.localTrans.Load(msg.TransactionId)
30. **if** !existed {
31. fmt.Printf("unknow msg: %v, return Commit", msg)
32. **return** primitive.CommitMessageState
33. }
35. // 校验本地事务状态
36. state := v.(primitive.LocalTransactionState)
37. **switch** state {
38. **case** 1:
39. fmt.Printf("checkLocalTransaction COMMIT\_MESSAGE: %v\n", msg)
40. **return** primitive.CommitMessageState
41. **case** 2:
42. fmt.Printf("checkLocalTransaction ROLLBACK\_MESSAGE: %v\n", msg)
43. **return** primitive.RollbackMessageState
44. **case** 3:
45. fmt.Printf("checkLocalTransaction unknow: %v\n", msg)
46. **return** primitive.UnknowState
47. **default**:
48. fmt.Printf("checkLocalTransaction default COMMIT\_MESSAGE: %v\n", msg)
49. **return** primitive.CommitMessageState
50. }
51. }
53. // ExecuteLocalTransaction 执行本地事务逻辑
54. // msg: 事务消息
55. // return: 本地事务状态
56. func (tr \*Trans) ExecuteLocalTransaction(msg \*primitive.Message) primitive.LocalTransactionState {
57. nextIndex := atomic.AddInt32(&tr.transactionIndex, 1)
58. fmt.Printf("nextIndex: %v for transactionID: %v\n", nextIndex, msg.TransactionId)
59. status := nextIndex % 3
60. tr.localTrans.Store(msg.TransactionId, primitive.LocalTransactionState(status+1))
62. **return** primitive.UnknowState
63. }
65. func ProducerTranMessage() {
66. ch := make(chan **struct**{})
67. // 创建事务生产者
68. p, err := rocketmq.NewTransactionProducer(
69. // 事务对象
70. NewTrans(),
71. // 设置服务地址
72. producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"{nameServer}"})),
73. // // 设置acl权限
74. // producer.WithCredentials(primitive.Credentials{
75. //  SecretKey: "{secretKey}",
76. //  AccessKey: "{accessKey}",
77. // }),
78. // // 设置生产组
79. // producer.WithGroupName("{groupName}"),
80. // // 设置命名空间名称
81. // producer.WithNamespace("{nameSpace}"),
82. // 设置发送失败重试次数
83. producer.WithRetry(1),
84. )
85. **if** err != nil {
86. fmt.Printf("new producer error: %s\n", err.Error())
87. **return**
88. }
89. // 启动producer
90. err = p.Start()
91. **if** err != nil {
92. fmt.Printf("start producer error: %s\n", err.Error())
93. os.Exit(1)
94. }
95. // 关闭producer
96. defer p.Shutdown()
97. // 构造事务消息
98. msg := primitive.NewMessage("{topicName}", []byte("{message}"))
99. // 发送事务消息
100. res, err := p.SendMessageInTransaction(context.Background(), msg)
101. **if** err != nil {
102. fmt.Printf(err.Error())
103. **return**
104. } **else** {
105. fmt.Printf("预投递成功 resid : %s \n", res.MsgID)
106. }
107. <-ch
108. }

### 2.3.5 顺序消息

1. import (
2. "context"
3. "fmt"
5. "gitlab.apusic.net/admq/rocketmq-client-go"
6. "gitlab.apusic.net/admq/rocketmq-client-go/primitive"
7. "gitlab.apusic.net/admq/rocketmq-client-go/producer"
8. )

11. func ProducerFifoMessage() {
12. // 创建消息生产者
13. p, err := rocketmq.NewProducer(
14. // 设置服务地址
15. producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"{nameServer}"})),
16. // 设置acl权限
17. producer.WithCredentials(primitive.Credentials{
18. SecretKey: "{secretKey}",
19. AccessKey: "{accessKey}",
20. }),
21. // 设置生产组
22. producer.WithGroupName("{groupName}"),
23. // 设置命名空间名称， 和Message中的Topic参数构成实际的Topic名称 namespace%Topic
24. producer.WithNamespace("{nameSpace}"),
25. //表明使用要发送的队列就是msg中定义的queue
26. producer.WithQueueSelector(producer.NewManualQueueSelector()),
27. // 设置发送失败重试次数
28. producer.WithRetry(2),
29. )
30. **if** err != nil {
31. fmt.Printf("new producer error: %s\n", err.Error())
32. **return**
33. }
34. // 启动生产者
35. err = p.Start()
36. **if** err != nil {
37. fmt.Printf("producer start err:%v\n", err)
38. **return**
39. }
41. // 构建消息，并发送到topic的制定队列
42. msg := &primitive.Message{
43. Topic: "{topicName}",
44. Body:  []byte("{message}"),
45. Queue: &primitive.MessageQueue{
46. //消息发送到RocketMQ上的topic名称
47. Topic:      "{namespaceTopic}",
48. BrokerName: "{brokerName}",
49. QueueId:    0,
50. },
51. }
52. //同步发送
53. res, err := p.SendSync(context.Background(), msg)
54. **if** err != nil {
55. fmt.Printf("send message error: %s\n", err)
56. **return**
57. } **else** {
58. fmt.Printf("send message success: result=%s\n", res.String())
59. }
60. //关闭生产者
61. err = p.Shutdown()
62. **if** err != nil {
63. fmt.Printf("shutdown producer error: %s", err.Error())
64. **return**
65. }
66. **return**
67. }