

RocketMQ：发送消息和消费消息

一：普通消息

RocketMQ提供了三种方式来发送普通消息：可靠同步发送、可靠异步发送、单向发送。

1.1 可靠同步发送

同步发送是指发送消息后必须等待RocketMQ服务返回发送的结果，这里会一直同步阻塞，直到拿到RocketMQ服务返回发送的结果才继续往下执行代码。同步发送一般应用于对发送成功可靠性要求很严格的场景。

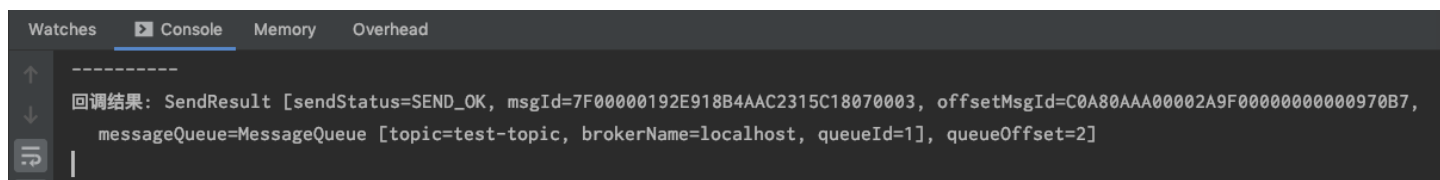
```
1 // 同步发送会有发送结果返回值
2 // rocketMQTemplate 使用冒号将topic和tag分割
3 SendResult sendResult = rocketMQTemplate.syncSend("test-topic:test-tag", "test msg");
```

1.2 可靠异步发送

异步发送是指发送消息后不需要等待RocketMQ服务器返回的发送结果，而是直接执行后面的逻辑。发送方通过设置回调接口来接收RocketMQ服务器异步返回的发送结果，并根据具体的发送结果进行相应处理。

```
1 rocketMQTemplate.asyncSend("test-topic:test-tag", "test message", new SendCallback() {
    @Override
2     public void onSuccess(SendResult sendResult) {
3         // 成功回调
4         System.out.println("回调结果: " + sendResult);
5     }
6
7     @Override
8     public void onException(Throwable throwable) {
9         // 异常回调
10        System.out.println(throwable);
11    }
12 });
13
14 System.out.println("-----");
15
```

先打印 ----- 后打印回调函数。



1.3 单向发送

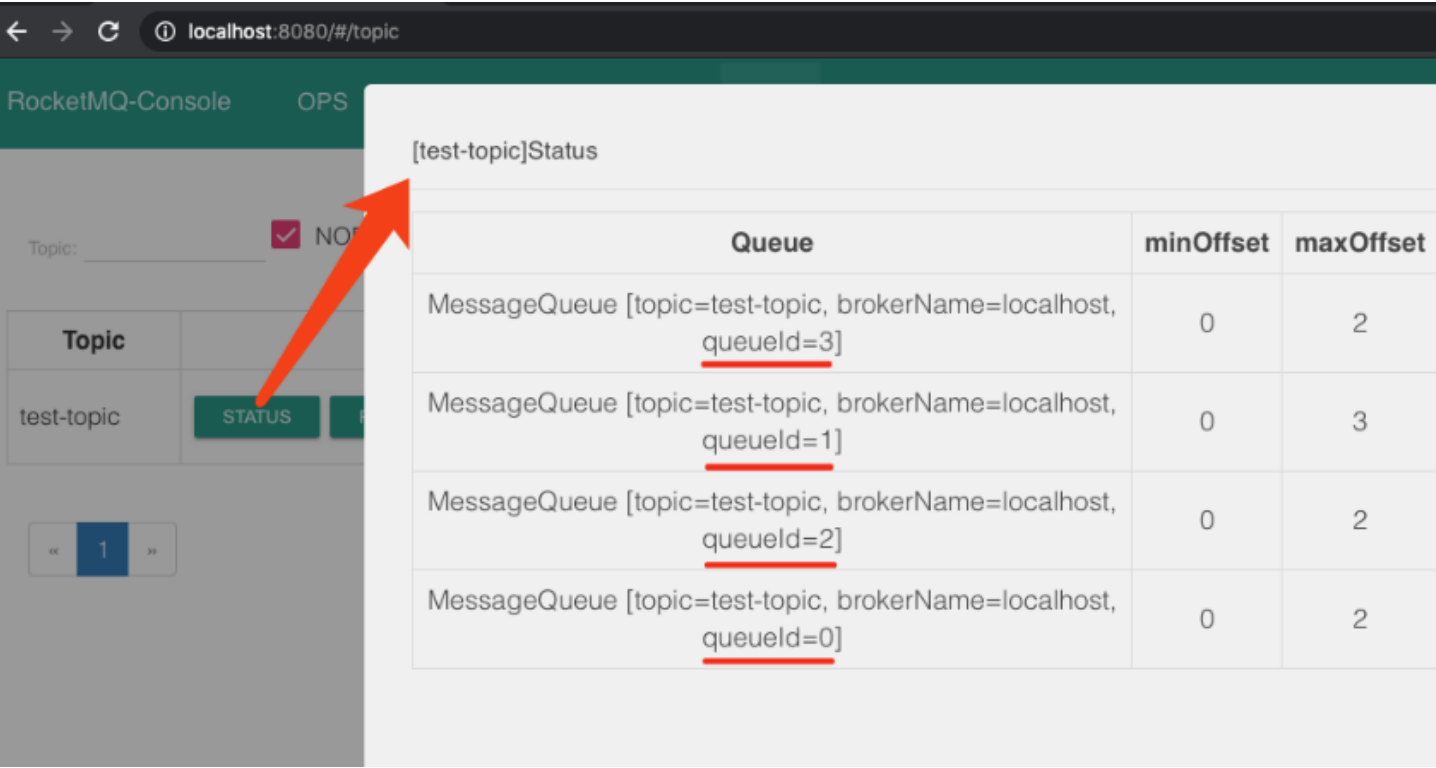
单向发送只负责发送消息，不等待RocketMQ服务器返回的发送结果，也不提供回调函数来接收RocketMQ服务器的响应结果，只负责发送至于发送成功还是发送失败我不关心。单向发送通常用于对可靠性要求不高的场景。

```
1 rocketMQTemplate.sendOneWay("test-topic:test-tag", "test msg");
```

注意：普通消息(以上三种)不保证发送消息的顺序和消费者消费的顺序一致性。

二：顺序消息

一个topic默认有4个消息队列(Message Queue)，生产者在发送同一个topic的多个消息会被存储到不同的消息队列中，默认是按照轮询的模式来依次存储到每个队列中的，而消费消息首先从哪个队列中获取消息是不确定的，这导致发送的顺序和消费的消息很可能会不一致。



如果想要保证发送消息的顺序和消费消息的顺序要一致，只需要保证将消息都发送到同一个消息队列上即可，而不是轮询的放到每个队列上。顺序消息只需要在原来的发送方法后面增加Orderly后缀(syncSendOrderly、asyncSendOrderly、sendOneWayOrderly)，并在最后传入一个值Hash Key，RocketMQ会根据这个值计算要发送到哪一个队列上。

```

1 String orderId = order.getId().toString();
2 for (int i = 0; i < 10; i++) {
3     // Hash Key 一般是一组顺序消息的对应的一个唯一值，如 张三创建订单、张三订单付款、张三订单完
    成，这里hash key就用张三对应的订单id或者订单号
4     rocketMQTemplate.sendOneWayOrderly("test-topic", order, orderId);
5 }

```

10个消息都发送到queueId=3的队列上了。

[test-topic]Status

Queue	minOffset	maxOffset
MessageQueue [topic=test-topic, brokerName=localhost, queueId=3]	0	12
MessageQueue [topic=test-topic, brokerName=localhost, queueId=1]	0	3
MessageQueue [topic=test-topic, brokerName=localhost, queueId=2]	0	2
MessageQueue [topic=test-topic, brokerName=localhost, queueId=0]	0	2

消费消息时需要指定消费者顺序(单线程)获取消息。

```

1 @Component
2 @RocketMQMessageListener(consumerGroup = "testConsumerGroup", topic = "test-topic", consume
    Mode = ConsumeMode.ORDERLY)
3 public class TestTopicConsumer1Listener implements RocketMQListener<String> {
4     @Override
5     public void onMessage(String message) {
6         System.out.println("TestTopicConsumerListener 消费消息: " + message);
7     }
8 }

```

顺序消息：保证同一组内的消息按顺序消费。

```
Overhead
INFO 74446 --- [nio-8888-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
INFO 74446 --- [nio-8888-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 7 ms
消费消息: {"id":1,"desc":"创建"}
消费消息: {"id":2,"desc":"创建"}
INFO 74446 --- [MessageThread_2] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905BA0000 cost: 0 ms
消费消息: {"id":3,"desc":"创建"}
INFO 74446 --- [MessageThread_1] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C30008 cost: 2 ms
消费消息: {"id":2,"desc":"付款"}
INFO 74446 --- [MessageThread_3] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C5000E cost: 1 ms
消费消息: {"id":3,"desc":"付款"}
INFO 74446 --- [MessageThread_1] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C4000A cost: 0 ms
INFO 74446 --- [MessageThread_3] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C60010 cost: 0 ms
消费消息: {"id":2,"desc":"完成"}
INFO 74446 --- [MessageThread_1] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C5000C cost: 0 ms
消费消息: {"id":1,"desc":"付款"}
INFO 74446 --- [MessageThread_4] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C00002 cost: 0 ms
消费消息: {"id":1,"desc":"完成"}
INFO 74446 --- [MessageThread_4] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C10004 cost: 0 ms
消费消息: {"id":1,"desc":"通知"}
INFO 74446 --- [MessageThread_4] a.r.s.s.DefaultRocketMQListenerContainer : consume 7F00000122CE18B4AAC273E905C10006 cost: 0 ms
```

三：延迟消息

延迟消息是指当生产者发送消息到Broker中不会被消费者立即消费，而是延迟指定的时间消费，注意RocketMQ不支持自定义延迟任意时间，只提供了18个固定时间级别供选择，级别从1开始到18，1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m , 6m, 7m, 8m, 9m, 10m, 20m, 30m, 1h, 2h。注意由于网络延迟，所以延迟消息消费也会大于指定的延迟时间。延迟消息在发送时只需要指定delayLevel参数即可。

```
1 public SendResult syncSend(String destination, Message<?> message, long timeout, int delayLevel);
```

```
1 GenericMessage message = new GenericMessage(order.toString());
  rocketMQTemplate.syncSend("test-topic", message, 3000, 3);
2
```

四：批量消息

使用Template发送批量消息时，发送的消息必须是org.springframework.messaging.Message的子类。批量发消息对消息的内容长度有限制，最大为4M，如果超过4M只能分为多批次发送。

```

1 // 方式一
2 List<GenericMessage> list = new ArrayList<>();
3 list.add(new GenericMessage("1"));
4 list.add(new GenericMessage("2"));
5
6 rocketMQTemplate.syncSend("test-topic", list);
7
8 // 方式二：使用生产者批量发送，Message是org.apache.rocketmq.common.message.Message
9 rocketMQTemplate.getProducer.send(Collection<Message> msgs);

```

五：消费消息模式

消息模式有两种，默认是负载均衡模式：

- 负载均衡(MessageModel.CLUSTERING)：多个消费者共同瓜分所有消息，一个消息只能被一个消费者消费掉。
- 广播模式(MessageModel.BROADCASTING)：每个消费者都会消费同样的消息。一个消息会被所有消费者消费掉。

消费模式使用messageModel参数来指定，注意：如果要测试多个消费者，不能写多个监听器，否则SpringBoot启动失败，只能写一个监听器，然后分别使用不同的端口分别启动Spring Boot。

```

1 @Component
2 @RocketMQMessageListener(consumerGroup = "testConsumerGroup", topic = "test-topic", message
   Model = MessageModel.BROADCASTING)
3 public class TestTopicConsumer1Listener implements RocketMQListener<String> {
4     @Override
5     public void onMessage(String message) {
6         System.out.println("TestTopicConsumerListener 消费消息: " + message);
7     }
8 }

```

```

Overhead
INFO 73762 --- [      main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8888 (http) with context path ''
INFO 73762 --- [      main] c.e.r.SpringbootRocketmqApplication    : Started SpringbootRocketmqApplication in 4.756 seconds (JVM running

INFO 73762 --- [nio-8888-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]     : Initializing Spring DispatcherServlet 'dispatcherServlet'
INFO 73762 --- [nio-8888-exec-1] o.s.web.servlet.DispatcherServlet      : Initializing Servlet 'dispatcherServlet'
INFO 73762 --- [nio-8888-exec-1] o.s.web.servlet.DispatcherServlet      : Completed initialization in 12 ms
消费消息: test message

```

```

otRocketmqApplication (1) x
Overhead
messageModel=BROADCASTING)
INFO 73764 --- [      main] o.a.r.s.a.ListenerContainerConfiguration : Register the listener to container,
icConsumer1Listener, containerBeanName:org.apache.rocketmq.spring.support.DefaultRocketMQListenerContainer_1
INFO 73764 --- [      main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8889 (http) with context path ''
INFO 73764 --- [      main] c.e.r.SpringbootRocketmqApplication    : Started SpringbootRocketmqApplication in 4.041 seconds (JVM running

消费消息: test message

```

六：过滤消息

消费者过滤消息可以通过两种方式：

- Tag，生产者发送消息时可以指定Tag，消费者消费消息可以指定Tag过滤，也可以使用通配符*表示所有Tag，也可以使用||来表示多个Tag，使用Template发送带Tag的消息是使用冒号分隔跟在主题后面。
- SQL 基本语法。

6.1 Tag过滤

selectorType默认是Tag，selectorExpression默认是*，可以通过或符号||来选择消费多个Tag。

```
1 @Component
2 @RocketMQMessageListener(
3     consumerGroup = "testConsumerGroup",
4     topic = "test-topic",
5     selectorType = SelectorType.TAG,
6     selectorExpression = "test-tag || dev-tag")
7 public class TestTopicConsumerListener implements RocketMQListener<MessageExt> {
8     @Override
9     public void onMessage(MessageExt messageExt) {
10         System.out.println(messageExt);
11         System.out.println("body: " + new String(messageExt.getBody()));
12     }
13 }
```

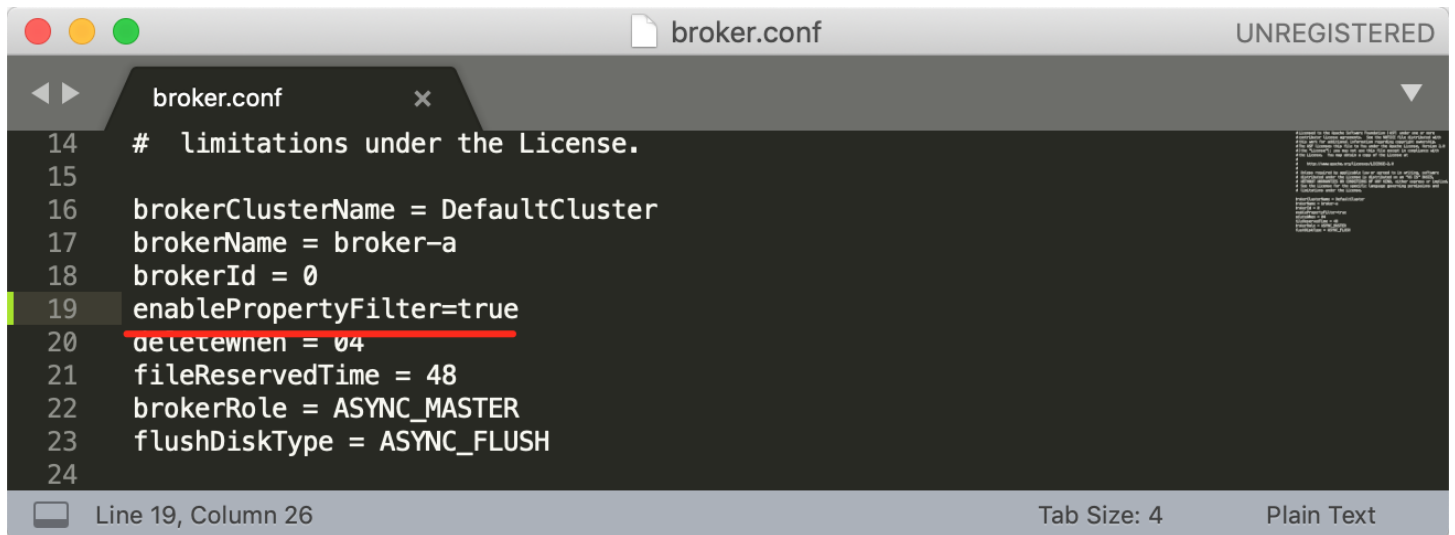
6.2 SQL92过滤

在发送消息的时候RocketMQ运行携带一个Map，可以通过Map中的每个值作为条件来过滤消息。SQL92过滤可以使用SQL中常用的Where条件来过滤要消费的消息，支持如下常用的SQL条件：

```
1 AND, OR
2 >, >=, <, <=, =
3 BETWEEN A AND B, equals to >=A AND <=B
4 NOT BETWEEN A AND B, equals to >B OR <A
5 IN ('a', 'b'), equals to ='a' OR ='b', this operation only support String type.
6 IS NULL, IS NOT NULL, check parameter whether is null, or not.
7 =TRUE, =FALSE, check parameter whether is true, or false.
8
9 样例：
10 (a > 10 AND a < 100) OR (b IS NOT NULL AND b=TRUE)
```

RocketMQ默认是关闭SQL92过滤方式的，如果需要需要通过配置文件配置，然后重启RocketMQ。

```
1 org.apache.rocketmq.client.exception.MQClientException: CODE: 1 DESC: The broker does not support consumer to filter message by SQL92
```



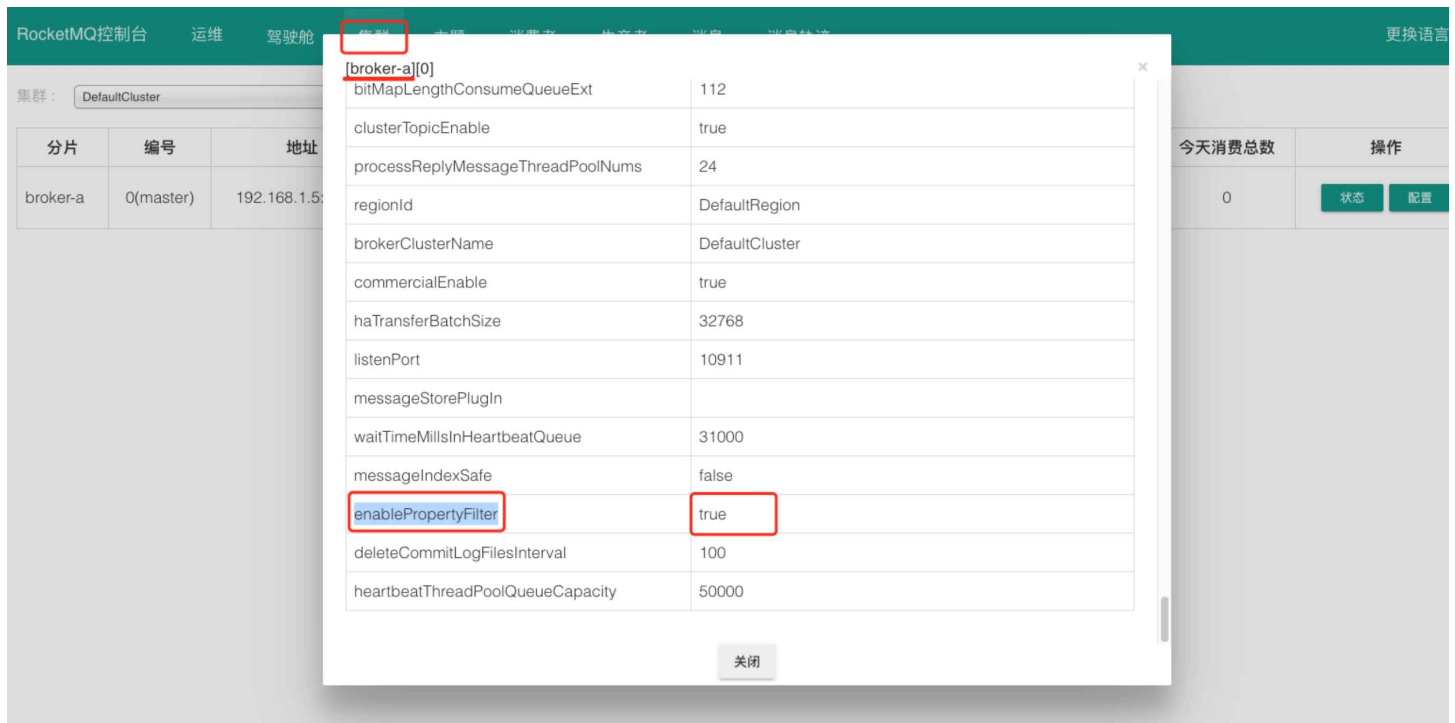
```
14 # limitations under the License.
15
16 brokerClusterName = DefaultCluster
17 brokerName = broker-a
18 brokerId = 0
19 enablePropertyFilter=true
20 deleteWhen = 04
21 fileReservedTime = 48
22 brokerRole = ASYNC_MASTER
23 flushDiskType = ASYNC_FLUSH
24
```

Line 19, Column 26 Tab Size: 4 Plain Text

启动时指定broker.conf

```
1 nohup sh bin/mqbroker -n localhost:9876 -c ./conf/broker.conf &
```

通过控制台查看SQL92过滤方式是否启用。



分片	编号	地址
broker-a	0(master)	192.168.1.5

属性	值
bitMapLengthConsumeQueueExt	112
clusterTopicEnable	true
processReplyMessageThreadPoolNums	24
regionId	DefaultRegion
brokerClusterName	DefaultCluster
commercialEnable	true
haTransferBatchSize	32768
listenPort	10911
messageStorePlugIn	
waitTimeMillsInHeartbeatQueue	31000
messageIndexSafe	false
enablePropertyFilter	true
deleteCommitLogFilesInterval	100
heartbeatThreadPoolQueueCapacity	50000

关闭

```
1 // 方式一：使用Spring封装方式携带属性是通过setHeader来设置的
2 // setHeader 就是设置userProperty，注意header不能使用id,timestamp作为key
3 rocketMQTemplate.syncSend("test-filter-topic",
4     MessageBuilder.withPayload("msg boday").setHeader("age", 2).build());
5
6 // 方式二：使用rocketmq-cliet原生的发送方法是在消息对象上通过putUserProperty(String key, String
7     value) 来设置的
8 Message message = new Message();
9 message.setTopic("test-topic");
10 message.putUserProperty("age", "2");
11 rocketMQTemplate.getProducer().send(message1);
12
13 @RocketMQMessageListener(
14     consumerGroup = "testConsumerGroup",
15     topic = "test-filter-topic",
16     selectorType = SelectorType.SQL92,
17     selectorExpression = "age=2"
18 )
```