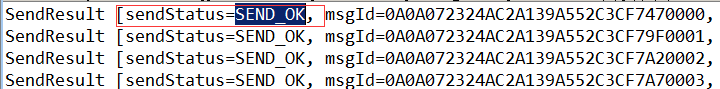
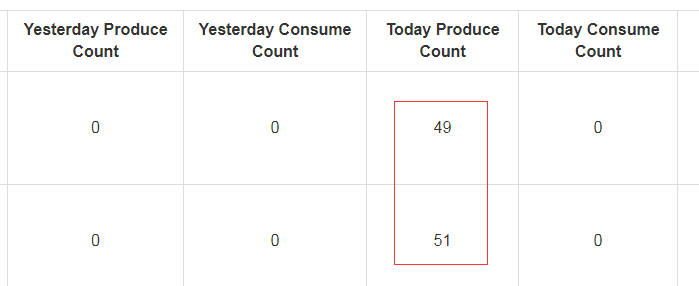
# 生产者

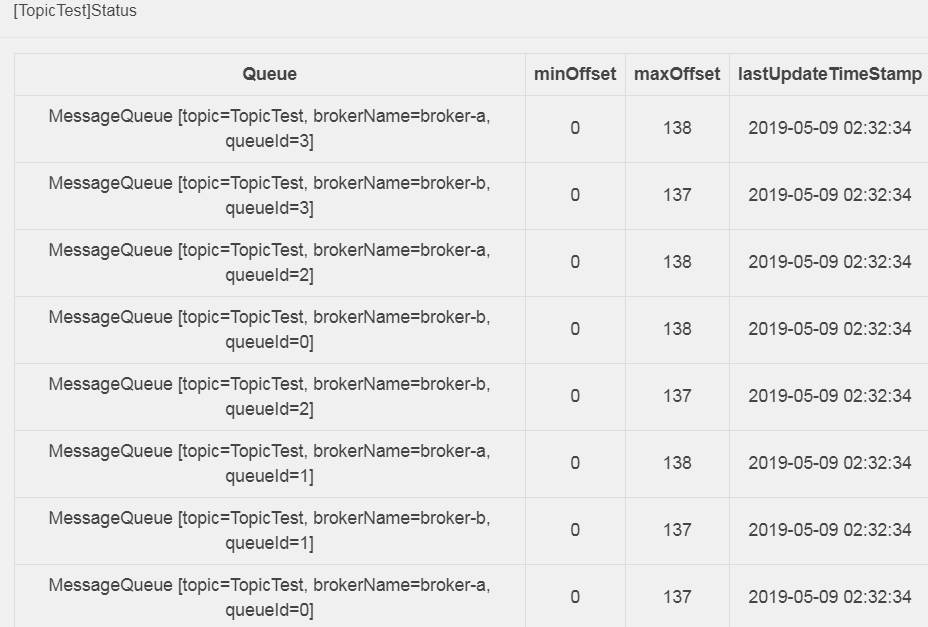
## 同步发送

|  |
| --- |
| **package** com.cjl.rocketmq.simple;  **import** org.apache.rocketmq.client.producer.DefaultMQProducer;  **import** org.apache.rocketmq.client.producer.SendResult;  **import** org.apache.rocketmq.common.message.Message;  **import** org.apache.rocketmq.remoting.common.RemotingHelper;  **public** **class** SyncProducer {  **public** **static** **void** main(String[] args) **throws** Exception {  // Instantiate with a producer group name.  DefaultMQProducer producer = **new** DefaultMQProducer("please\_rename\_unique\_group\_name");  // Specify name server addresses.  producer.setNamesrvAddr("192.168.57.100:9876");  // Launch the instance.  producer.start();  **for** (**int** i = 0; i < 100; i++) {  // Create a message instance, specifying topic, tag and message body.  Message msg = **new** Message("TopicTest" /\* Topic \*/, "TagA" /\* Tag \*/,  ("Hello RocketMQ " + i).getBytes(RemotingHelper.***DEFAULT\_CHARSET***) /\* Message body \*/  );  // Call send message to deliver message to one of brokers.  SendResult sendResult = producer.send(msg);  System.***out***.printf("%s%n", sendResult);  }  // Shut down once the producer instance is not longer in use.  producer.shutdown();  }  } |



管控台查看发送结果：



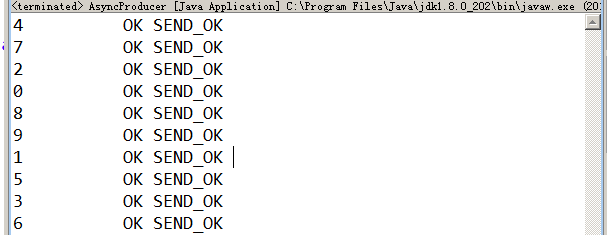


使用场景：对响应时间要求不高，但是对可靠性要求比较高的业务。例如：SMS系统。

## 异步发送

|  |
| --- |
| **package** com.cjl.rocketmq.simple;  **import** org.apache.rocketmq.client.producer.DefaultMQProducer;  **import** org.apache.rocketmq.client.producer.SendCallback;  **import** org.apache.rocketmq.client.producer.SendResult;  **import** org.apache.rocketmq.common.message.Message;  **import** org.apache.rocketmq.remoting.common.RemotingHelper;  **public** **class** AsyncProducer {  **public** **static** **void** main(String[] args) **throws** Exception {  // Instantiate with a producer group name.  DefaultMQProducer producer = **new** DefaultMQProducer("please\_rename\_unique\_group\_name");  // Specify name server addresses.  producer.setNamesrvAddr("192.168.57.100:9876");  // Launch the instance.  producer.start();  producer.setRetryTimesWhenSendAsyncFailed(0);  **for** (**int** i = 0; i < 10; i++) {  **final** **int** index = i;  // Create a message instance, specifying topic, tag and message body.  Message msg = **new** Message("TopicTestAsync", "TagA", "OrderID188",  "Hello world".getBytes(RemotingHelper.***DEFAULT\_CHARSET***));  producer.send(msg, **new** SendCallback() {  @Override  **public** **void** onSuccess(SendResult sendResult) {  System.***out***.printf("%-10d OK %s %n", index, sendResult.getMsgId());  }  @Override  **public** **void** onException(Throwable e) {  System.***out***.printf("%-10d Exception %s %n", index, e);  e.printStackTrace();  }  });  }  // Shut down once the producer instance is not longer in use.  //producer.shutdown();  }  } |

**注意：异步发送时，请注意不要关闭生产者。否则会因为对象关闭，导致回调方法执行失败！！！！！！！**

****

使用场景：对响应时间敏感的业务。

## 单向发送

|  |
| --- |
| **package** com.cjl.rocketmq.simple;  **import** org.apache.rocketmq.client.producer.DefaultMQProducer;  **import** org.apache.rocketmq.common.message.Message;  **import** org.apache.rocketmq.remoting.common.RemotingHelper;  **public** **class** OnewayProducer {  **public** **static** **void** main(String[] args) **throws** Exception {  // Instantiate with a producer group name.  DefaultMQProducer producer = **new** DefaultMQProducer("please\_rename\_unique\_group\_name");  // Specify name server addresses.  producer.setNamesrvAddr("192.168.57.100:9876");  // Launch the instance.  producer.start();  **for** (**int** i = 0; i < 10; i++) {  // Create a message instance, specifying topic, tag and message body.  Message msg = **new** Message("onewaytopic" /\* Topic \*/, "TagB" /\* Tag \*/,  ("onewaytopic" + i).getBytes(RemotingHelper.***DEFAULT\_CHARSET***) /\* Message body \*/  );  // Call send message to deliver message to one of brokers.  **producer.sendOneway(msg);**  }  // Shut down once the producer instance is not longer in use.  producer.shutdown();  }  } |

没有返回值！！！！！只管发送，对数据的可靠性要求不高的场景：例如：日志收集系统。

## 总结

### 发送方式：

**Sync：**同步的发送方式，会等待发送结果后才返回

**Async：**异步的发送方式，发送完后，立刻返回。Client 在拿到 Broker 的响应结果后，会回调指定的 callback. 这个 API 也可以指定 Timeout，不指定也是默认的 3000ms.

**Oneway：**比较简单，发出去后，什么都不管直接返回。一般用于日志收集。

### 发送结果：

org.apache.rocketmq.client.producer.SendStatus

**SEND\_OK**：消息发送成功   
**FLUSH\_DISK\_TIMEOUT**：消息发送成功，但是服务器刷盘超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失   
**FLUSH\_SLAVE\_TIMEOUT：**消息发送成功，但是服务器同步到 Slave 时超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失   
***SLAVE\_NOT\_AVAILABLE***：

消息发送成功，但是此时 slave 不可用，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢

# 消费者

|  |
| --- |
| **package** com.cjl.rocketmq.simple;  **import** java.util.List;  **import** org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;  **import** org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;  **import** org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;  **import** org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;  **import** org.apache.rocketmq.client.exception.MQClientException;  **import** org.apache.rocketmq.common.message.MessageExt;  **public** **class** Consumer {  **public** **static** **void** main(String[] args) **throws** InterruptedException, MQClientException {  // Instantiate with specified consumer group name.  **DefaultMQPushConsumer** consumer = **new** DefaultMQPushConsumer("please\_rename\_unique\_group\_name");    // Specify name server addresses.  consumer.setNamesrvAddr("192.168.57.100:9876");    // Subscribe one more more topics to consume.  consumer.subscribe("TopicTest", "\*");    // Register callback to execute on arrival of messages fetched from brokers.  consumer.registerMessageListener(**new** MessageListenerConcurrently() {  @Override  **public** ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,  ConsumeConcurrentlyContext context) {  System.***out***.printf("%s Receive New Messages: %s %n", Thread.*currentThread*().getName(), msgs);  **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  });  //Launch the consumer instance.  consumer.start();  System.***out***.printf("Consumer Started.%n");  }  } |

消费模式本质上都是pull，我们一般使用更方便的**DefaultMQPushConsumer对象。**

# 顺序消息

# 定时消息

message.setDelayTimeLevel(3);

