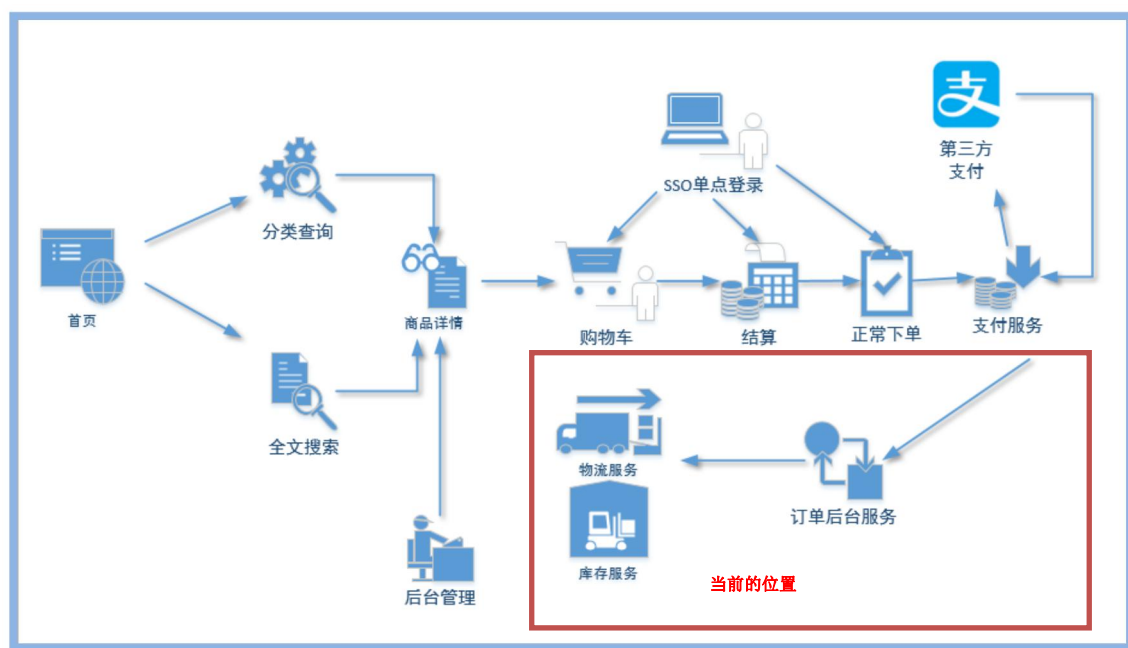


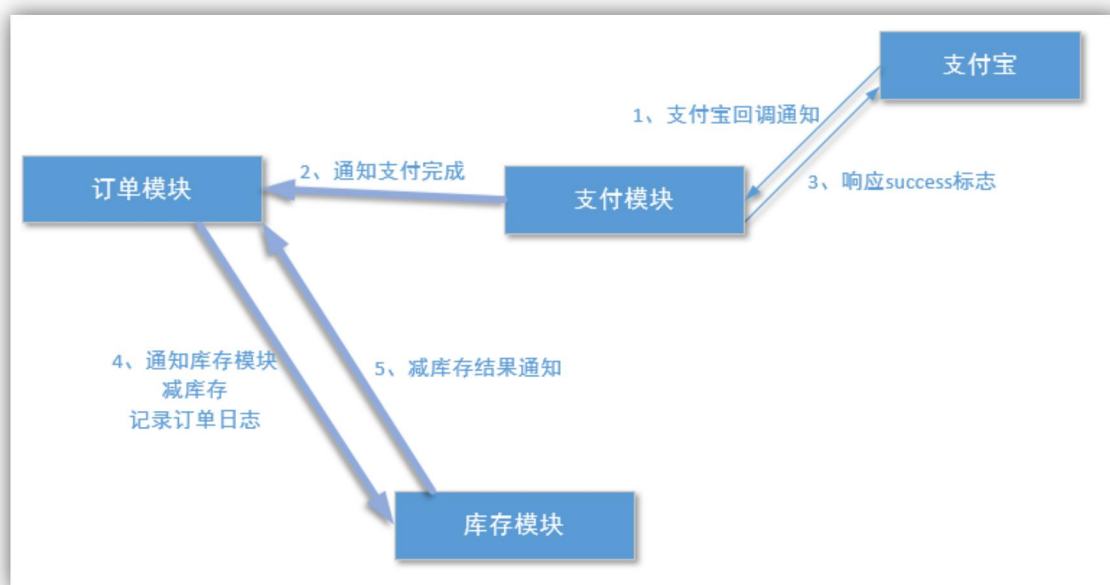
异步通信

版本: V 1.0

www.atguigu.com



一、分布式的业务场景



1、如何高效完成各个分布式系统的协作

通过消息队列来达到异步解耦的效果，减少了程序之间的阻塞等待时间，降低了因为服务之间调用的依赖风险。

2、消息的弊端？如何解决？

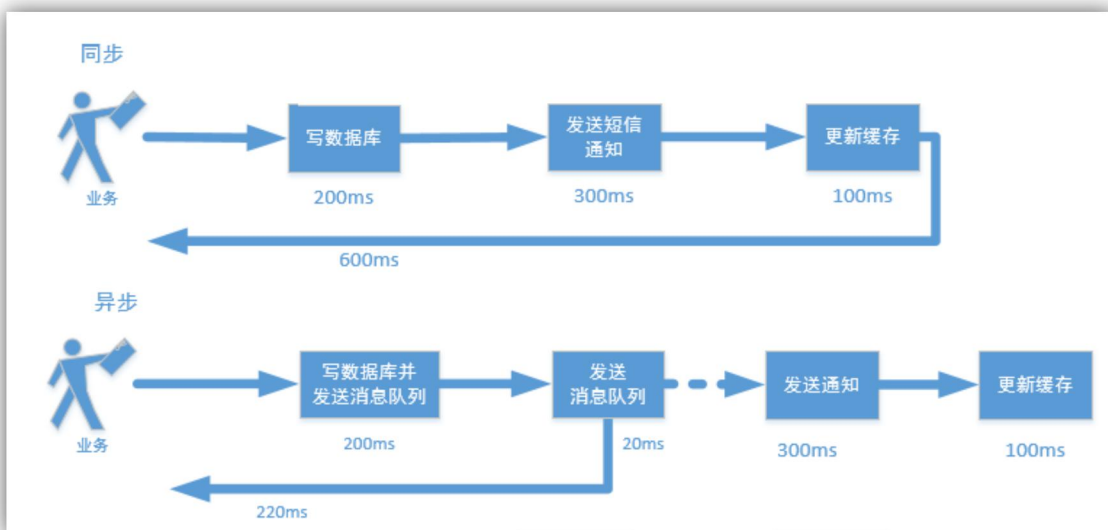
消息队列的问题在于**不确定性**，不能绝对保证消息的准确到达，所以要引入延迟、周期性的主动轮询，来发现未到达的消息，从而进行补偿。

二、消息队列简介

消息队列，也叫消息中间件。消息的传输过程中保存消息的容器。

消息队列都解决了什么问题？

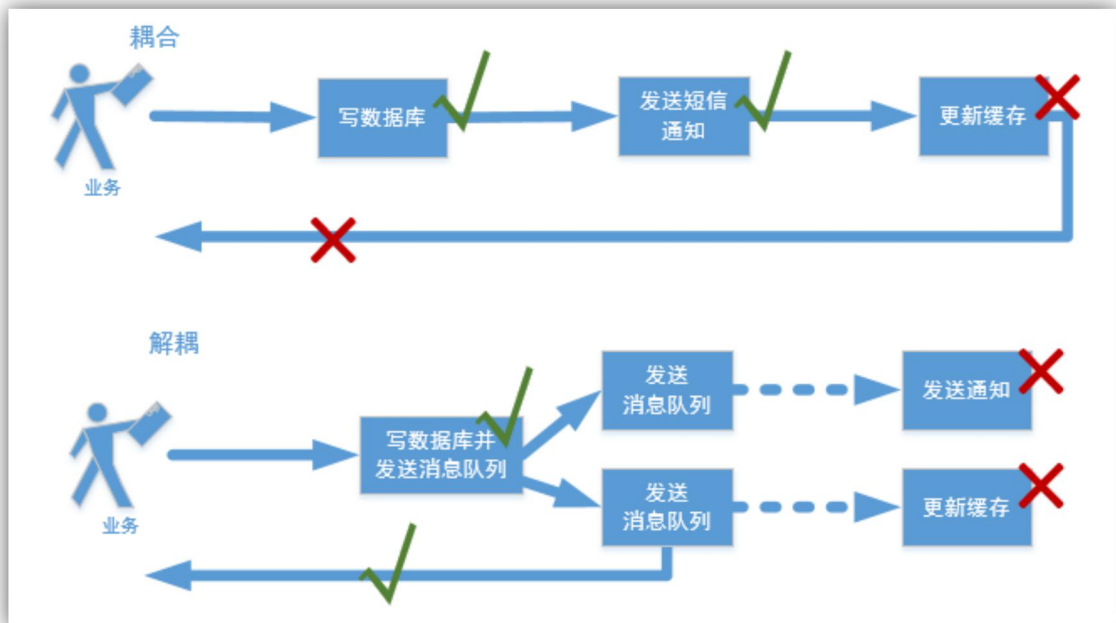
1、异步



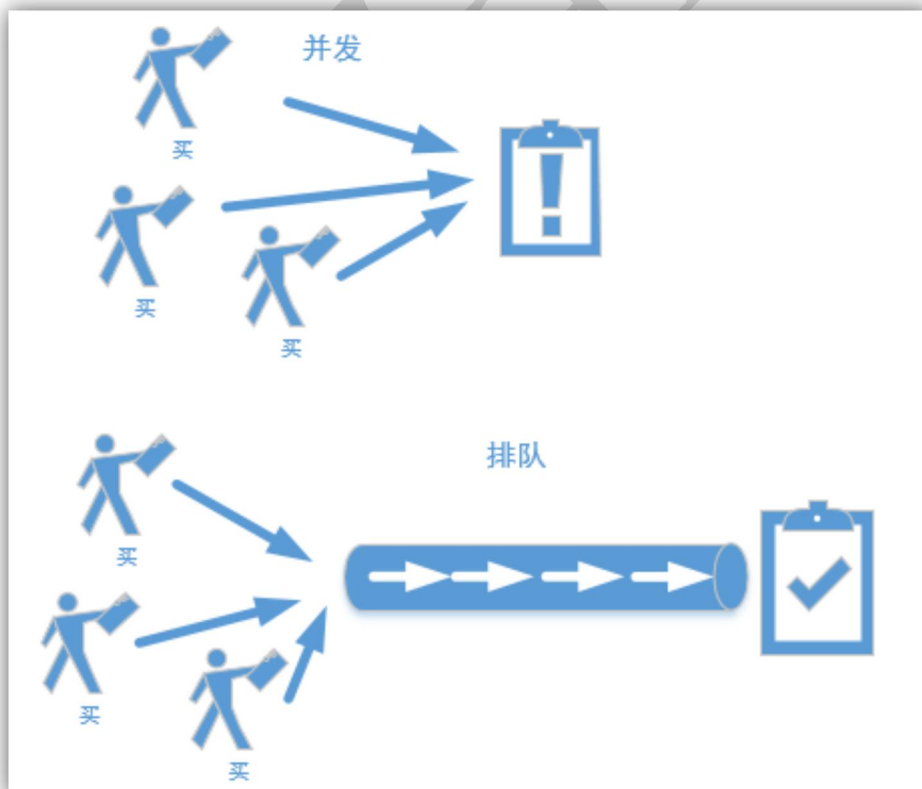
2、并行



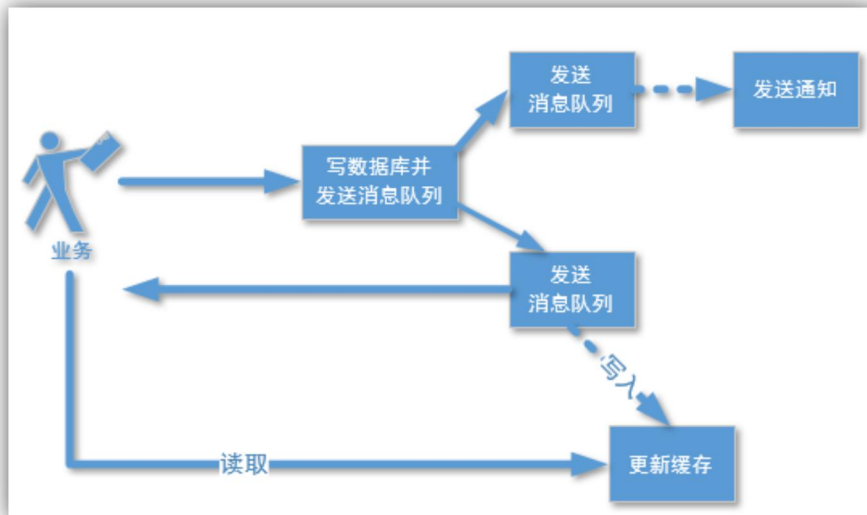
3、解耦



4、排队（削峰填谷）



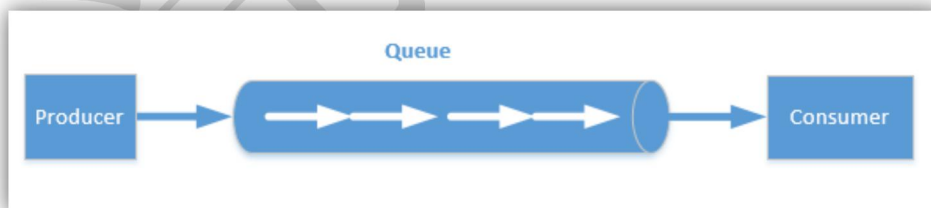
5 弊端：不确定性和延迟



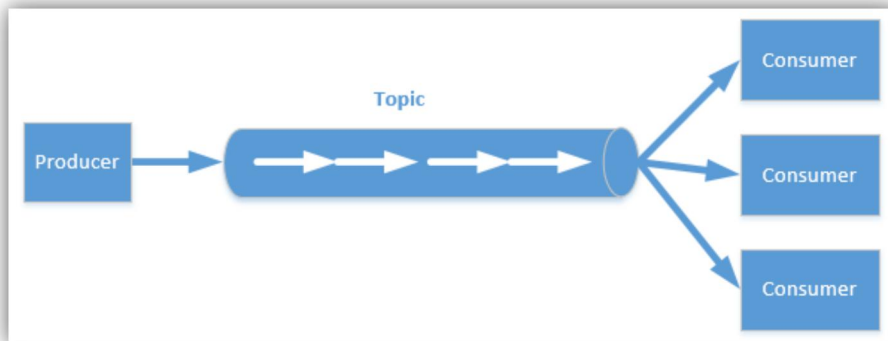
解决方案：最终一致

消息模式

点对点



订阅



三 消息队列工具 ActiveMQ

1 、简介



同类产品： RabbitMQ 、 Kafka、 Redis（List）

1.1 对比 RabbitMQ

最接近的同类型产品，经常拿来比较，性能伯仲之间，基本上可以互相替代。最主要区别是二者的协议不同 RabbitMQ 的协议是 AMQP(Advanced Message Queueing Protocol)，而 ActiveMQ 使用的是 JMS(Java Messaging Service)协议。顾名思义 JMS 是针对 Java 体系的传输协议，队列两端必须有 JVM，所以如果开发环境都是 java 的话推荐使用 ActiveMQ，可以用 Java 的一些对象进行传递比如 Map、Blob、Stream 等。而 AMQP 通用行较强，非 java 环境经常使用，传输内容就是标准字符串。

另外一点就是 RabbitMQ 用 Erlang 开发，安装前要装 Erlang 环境，比较麻烦。ActiveMQ 解压即可用不用任何安装。

1.2 对比 Kafka

Kafka 性能超过 ActiveMQ 等传统 MQ 工具，集群扩展性好。

弊端是：

在传输过程中可能会出现消息重复的情况，

不保证发送顺序

一些传统 MQ 的功能没有，比如消息的事务功能。

所以通常用 Kafka 处理大数据日志。

1.3 对比 Redis

其实 Redis 本身利用 List 可以实现消息队列的功能，但是功能很少，而且队列体积较大时性能会急剧下降。对于数据量不大、业务简单的场景可以使用。

2 安装 ActiveMQ

拷贝 apache-activemq-5.14.4-bin.tar.gz 到 Linux 服务器的/opt 下

解压缩 `tar -zxvf apache-activemq-5.14.4-bin.tar.gz`

重命名 `mv apache-activemq-5.14.4 activemq`

`vim /opt/activemq/bin/activemq`

```
46 # -----
47 # -----
48 # IMPROVED DEBUGGING (execute with bash -x)
49 # export PS4=' ${BASH_SOURCE}:${LINENO}({${FUNCNAME[0]}) '
50 # -----
51 # Backup invocation parameters
52 COMMANDLINE_ARGS="$@"
53 EXEC_OPTION=""
54 JAVA_HOME="/opt/jdk1.8.0_152"
55 JAVA_CMD="/opt/jdk1.8.0_152/bin"
56 # -----
57 # HELPERS
```

增加两行

```
JAVA_HOME="/opt/jdk1.8.0_152"
JAVA_CMD="/opt/jdk1.8.0_152/bin"
```

注册服务

```
ln -s /opt/activemq/bin/activemq /etc/init.d/activemq
chkconfig --add activemq
```

启动服务

service activemq start

```
[root@jack init.d]# service activemq start
INFO: Loading '/opt/activemq/bin/env'
INFO: Using java '/opt/jdk1.8.0_152/bin/java'
INFO: Starting - inspect logfiles specified in logging.properties and log4j.properties to get details
INFO: pidfile created : '/opt/activemq/data/activemq.pid' (pid '4846')
[root@jack init.d]# ps -ef|grep java
```

关闭服务

service activemq stop

通过 netstat 查看端口

tcp6	0	0	:::61614	:::*	LISTEN	4846/java
tcp6	0	0	:::34222	:::*	LISTEN	4846/java
tcp6	0	0	:::61616	:::*	LISTEN	4846/java
tcp6	0	0	:::1883	:::*	LISTEN	4846/java
tcp6	0	0	127.0.0.1:7005	:::*	LISTEN	882/java
tcp6	0	0	:::8161	:::*	LISTEN	4846/java
tcp6	0	0	:::7009	:::*	LISTEN	882/java
tcp6	0	0	:::39811	:::*	LISTEN	878/java
tcp6	0	0	:::2181	:::*	LISTEN	878/java

activemq 两个重要的端口，一个是提供消息队列的默认端口：61616

另一个是控制台端口 8161

通过控制台测试

启动消费端

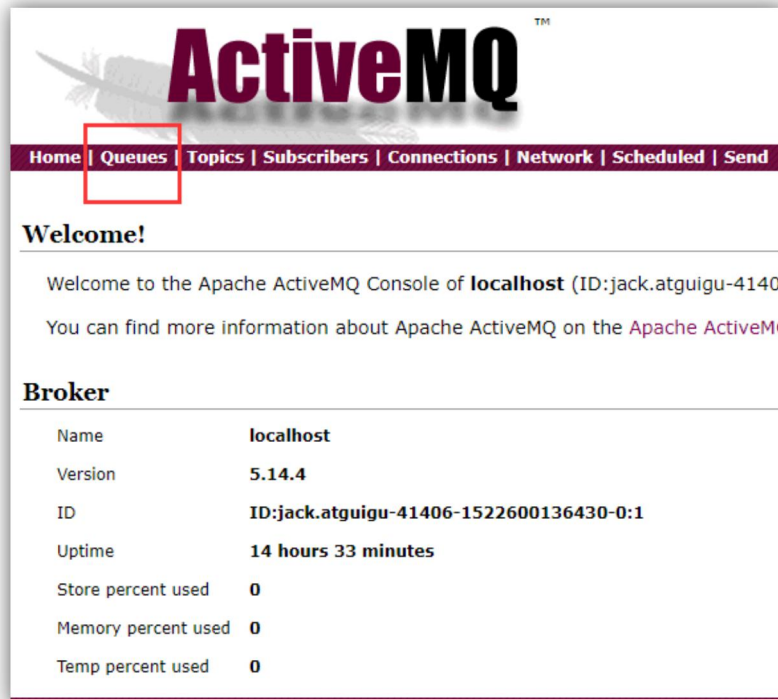
```
[root@jack init.d]# service activemq consumer
INFO: Loading '/opt/activemq/bin/env'
INFO: Using java '/opt/jdk1.8.0_152/bin/java'
Java Runtime: Oracle Corporation 1.8.0_152 /opt/jdk1.8.0_152/jre
Heap sizes: current=62976k free=61992k max=932352k
JVM args: -Xms64M -Xmx1G -Djava.util.logging.config.file=logging.pro
/opt/activemq/conf/login.config -Dactivemq.classpath=/opt/activemq/conf:
/activemq/ -Dactivemq.base=/opt/activemq/ -Dactivemq.conf=/opt/activemq/
Extensions classpath:
[/opt/activemq/lib,/opt/activemq/lib/camel,/opt/activemq/lib/optional.
a]
ACTIVEMQ_HOME: /opt/activemq
ACTIVEMQ_BASE: /opt/activemq
ACTIVEMQ_CONF: /opt/activemq/conf
ACTIVEMQ_DATA: /opt/activemq/data
INFO | Connecting to URL: failover://tcp://localhost:61616 (null:null)
INFO | Consuming queue://TEST
INFO | Sleeping between receives 0 ms
INFO | Running 1 parallel threads
INFO | Successfully connected to tcp://localhost:61616
INFO | consumer-1 wait until 1000 messages are consumed
```

进入网页控制台



账号/密码默认: admin/admin

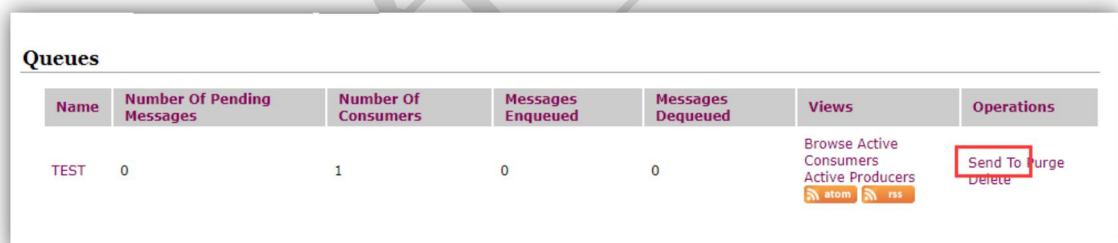
点击 Queues



The screenshot shows the Apache ActiveMQ console interface. The 'Queues' tab is highlighted in the top navigation bar. Below the navigation bar, there is a 'Welcome!' section and a 'Broker' section displaying details for the 'localhost' broker.

Broker Details:

Property	Value
Name	localhost
Version	5.14.4
ID	ID:jack.atguigu-41406-1522600136430-0:1
Uptime	14 hours 33 minutes
Store percent used	0
Memory percent used	0
Temp percent used	0



The screenshot shows the 'Queues' tab in the Apache ActiveMQ console. It displays a table with queue statistics and a list of operations for the 'TEST' queue.

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
TEST	0	1	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

Send a JMS Message

Message Header		
Destination	<input type="text" value="TEST"/>	Queue or Topic
Correlation ID	<input type="text"/>	Persistent Delivery
Reply To	<input type="text"/>	Priority
Type	<input type="text"/>	Time to live
Message Group	<input type="text"/>	Message Group Sequence
delay(ms)	<input type="text"/>	Time(ms) to wait before
Number of repeats	<input type="text"/>	Use a CRON string for
Number of messages to send	<input type="text" value="1"/>	Header to store the co

Message body

观察客户端

```
ACTIVEMQ_HOME: /opt/activemq
ACTIVEMQ_BASE: /opt/activemq
ACTIVEMQ_CONF: /opt/activemq/conf
ACTIVEMQ_DATA: /opt/activemq/data
INFO | Connecting to URL: failover://tcp://localhost:61616 (null)
INFO | Consuming queue://TEST
INFO | Sleeping between receives 0 ms
INFO | Running 1 parallel threads
INFO | Successfully connected to tcp://localhost:61616
INFO | consumer-1 wait until 1000 messages are consumed
INFO | consumer-1 Received hello activemq
```

3 在 Java 中使用消息队列

3.1 在 gmall-service-util 中导入依赖坐标

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-activemq</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-pool</artifactId>
  <version>5.15.2</version>
  <exclusions>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

3.2 producer 端

```
public static void main(String[] args) {  
    ConnectionFactory connect = new  
    ActiveMQConnectionFactory("tcp://192.168.67.163:61616");  
    try {  
        Connection connection = connect.createConnection();  
        connection.start();  
        // 第一个值表示是否使用事务，如果选择 true，第二个值相当于选择 0  
        Session session = connection.createSession(true, Session.SESSION_TRANSACTED);  
        Queue testqueue = session.createQueue("TEST1");  
  
        MessageProducer producer = session.createProducer(testqueue);  
        TextMessage textMessage = new ActiveMQTextMessage();  
        textMessage.setText("今天天气真好！");  
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);  
        producer.send(textMessage);  
        session.commit();  
        connection.close();  
    } catch (JMSException e) {  
        e.printStackTrace();  
    }  
}
```

3.3 consumer

```
public static void main(String[] args) {  
    ConnectionFactory connect = new  
    ActiveMQConnectionFactory(ActiveMQConnection.DEFAULT_USER, ActiveMQConnec  
    tion.DEFAULT_PASSWORD, "tcp://192.168.67.163:61616");  
    try {  
        Connection connection = connect.createConnection();  
        connection.start();  
        // 第一个值表示是否使用事务，如果选择 true，第二个值相当于选择 0  
        Session session = connection.createSession(false,  
        Session.AUTO_ACKNOWLEDGE);  
        Destination testqueue = session.createQueue("TEST1");  
  
        MessageConsumer consumer = session.createConsumer(testqueue);
```

```
consumer.setMessageListener(new MessageListener() {
    @Override
    public void onMessage(Message message) {
        if(message instanceof TextMessage){
            try {
                String text = ((TextMessage) message).getText();
                System.out.println(text);

                //session.rollback();
            } catch (JMSEException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
});

} catch (Exception e){
    e.printStackTrace();
}
}
```

3.4 关于事务控制

producer 提交时的任务	事务开启	只执行 send 并不会提交到队列中，只有当执行 session.commit() 时，消息才被真正的提交到队列中。
	事务不开启	只要执行 send ，就进入到队列中。
consumer 接收时的任务	事务开启，签收必须写	收到消息后，消息并没有

务	Session. SESSION_TRANSACTED	真正的被消费。消息只是被锁住。一旦出现该线程死掉、抛异常，或者程序执行了 <code>session.rollback()</code> 那么消息会释放，重新回到队列中被别的消费端再次消费。
	事务不开启，签收方式选择 Session.AUTO_ACKNOWLEDGE	只要调用 <code>comsumer.receive</code> 方法，自动确认。
	事务不开启，签收方式选择 Session.CLIENT_ACKNOWLEDGE	需要客户端执行 <code>message.acknowledge()</code> ，否则视为未提交状态，线程结束后，其他线程还可以接收到。 这种方式跟事务模式很像，区别是不能手动回滚，而且可以单独确认某个消息。
	事务不开启，签收方式选择 Session.DUPS_OK_ACKNOWLEDGE	在Topic模式下做批量签收时用的，可以提高性能。但是某些情况消息可能会被重复提交，使用这种模式的 <code>consumer</code> 要可以处理重复提交的问题。

3.5 持久化与非持久化

通过 `producer.setDeliveryMode(DeliveryMode.PERSISTENT)` 进行设置

持久化的好处就是当 `activemq` 宕机的话，消息队列中的消息不会丢失。非持久化会丢

失。但是会消耗一定的性能。

四 与 springboot 整合

1 配置类 ActiveMQConfig

```
@Configuration
public class ActiveMQConfig {

    @Value("${spring.activemq.broker-url:disabled}")
    String brokerURL ;

    @Value("${activemq.listener.enable:disabled}")
    String listenerEnable;

    @Bean
    public ActiveMQUtil getActiveMQUtil() throws JMSEException {
        if(brokerURL.equals("disabled")){
            return null;
        }
        ActiveMQUtil activeMQUtil=new ActiveMQUtil();
        activeMQUtil.init(brokerURL);
        return activeMQUtil;
    }

    //定义一个消息监听器连接工厂，这里定义的是点对点模式的监听器连接工厂
    @Bean(name = "jmsQueueListener")
    public DefaultJmsListenerContainerFactory
jmsQueueListenerContainerFactory(ActiveMQConnectionFactory activeMQConnectionFactory ) {
        DefaultJmsListenerContainerFactory factory = new
DefaultJmsListenerContainerFactory();
        if(!listenerEnable.equals("true")){
            return null;
        }

        factory.setConnectionFactory(activeMQConnectionFactory);
        //设置并发数
        factory.setConcurrency("5");

        //重连间隔时间
    }
}
```



```
factory.setRecoveryInterval(5000L);
factory.setSessionTransacted(false);
factory.setSessionAcknowledgeMode(Session.CLIENT_ACKNOWLEDGE);

return factory;
}

@Bean
public ActiveMQConnectionFactory activeMQConnectionFactory ( ){

    ActiveMQConnectionFactory activeMQConnectionFactory =
        new ActiveMQConnectionFactory( brokerURL);
    return activeMQConnectionFactory;
}
}
```

2 工具类 ActiveMQUtil

```
public class ActiveMQUtil {
    PooledConnectionFactory pooledConnectionFactory=null;

    public ConnectionFactory init(String brokerUrl) {

        ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(brokerUrl);
        //加入连接池
        pooledConnectionFactory=new PooledConnectionFactory(factory);
        //出现异常时重新连接
        pooledConnectionFactory.setReconnectOnException(true);
        //
        pooledConnectionFactory.setMaxConnections(5);
        pooledConnectionFactory.setExpiryTimeout(10000);
        return pooledConnectionFactory;
    }

    public ConnectionFactory getConnectionFactory(){
        return pooledConnectionFactory;
    }
}
```

五 在支付业务模块中应用

1 支付成功通知

支付模块利用消息队列通知订单系统，支付成功

在支付模块中配置 application.properties

```
spring.activemq.broker-url=tcp://mq.server.com:61616
```

在 PaymentServiceImpl 中增加发送方法：

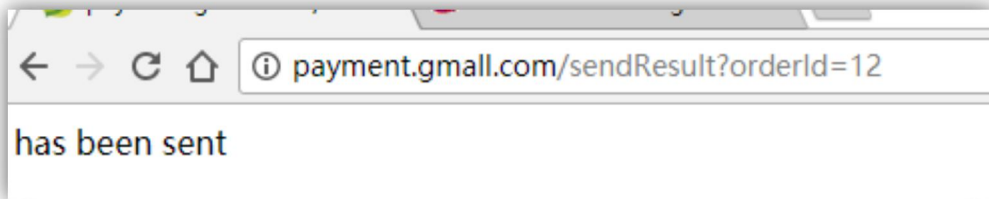
```
public void sendPaymentResult(String orderId,String result){
    ConnectionFactory connectionFactory = activeMQUtil.getConnectionFactory();
    Connection connection=null;
    try {
        connection = connectionFactory.createConnection();
        connection.start();
        Session session = connection.createSession(true, Session.SESSION_TRANSACTED);
        Queue paymentResultQueue = session.createQueue("PAYMENT_RESULT_QUEUE");
        MapMessage mapMessage=new ActiveMQMapMessage();
        mapMessage.setString("orderId",orderId);
        mapMessage.setString("result",result);
        MessageProducer producer = session.createProducer(paymentResultQueue);
        producer.send(mapMessage);
        session.commit();

        producer.close();
        session.close();
        connection.close();
    } catch (JMSException e) {
        e.printStackTrace();
    }
}
```

在 PaymentController 中增加一个方法用来测试

```
@RequestMapping("sendResult")
@ResponseBody
public String sendPaymentResult(@RequestParam("orderId") String orderId){
    paymentService.sendPaymentResult(orderId,"success" );
    return "has been sent";
}
```

在浏览器中访问：



查看队列内容：有一个在队列中没有被消费的消息。

Queues						
Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
PAYMENT_RESULT	1	0	1	0	Browse Active Consumers Active Producers	Send To Purge Delete

Browse PAYMENT_RESULT			
Message ID	Correlation ID	Persistence	Priority
ID:DESKTOP-S7LNAI5-65005-1523027856139-1:1:1:1:1		Persistent	4
View Consumers 0			

Message Details
{result=success, orderId=12}

2 订单模块消费消息

application.properties

```
spring.activemq.broker-url=tcp://mq.server.com:61616
activemq.listener.enable=true
```

订单消息消费后要更新订单状态，先准备好订单状态更新的方法

```
public void updateProcessStatus(String orderId, ProcessStatus processStatus,
Map<String,String>... paramMaps) {
    OrderInfo orderInfo = new OrderInfo();
    orderInfo.setId(orderId);
    orderInfo.setOrderStatus(processStatus.getOrderStatus());
    orderInfo.setProcessStatus(processStatus);

    //动态增加需要补充更新的属性
    if (paramMaps != null && paramMaps.length > 0) {
        Map<String, String> paramMap = paramMaps[0];
        for (Map.Entry<String, String> entry : paramMap.entrySet()) {
            String properties = entry.getKey();
            String value = entry.getValue();
            try {
                BeanUtils.setProperty(orderInfo, properties, value);
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
        }
    }
    orderInfoMapper.updateByPrimaryKeySelective(orderInfo);
}
```

消息队列的消费端

```
@JmsListener(destination = "PAYMENT_RESULT_QUEUE",containerFactory =
"jmsQueueListener")
public void consumePaymentResult(MapMessage mapMessage) throws JMSException {
    String orderId = mapMessage.getString("orderId");
    String result = mapMessage.getString("result");
    if(!"success".equals(result)){
        orderService.updateProcessStatus( orderId, ProcessStatus.PAY_FAIL);
    }else{
        orderService.updateProcessStatus( orderId, ProcessStatus.PAID);
    }
}
```

```
}  
    orderService.sendOrderResult(orderId);  
}
```

3 订单模块发送减库存通知

订单模块除了接收到请求改变单据状态，还要发送库存系统

查看《库存管理系统接口手册》中【减库存的消息队列消费端接口】中的描述，组织相应的消息数据进行传递。

```
@Transactional  
public void sendOrderResult(String orderId){  
    OrderInfo orderInfo = getOrderInfo(orderId);  
    Map<String, Object> messageMap = initWareOrderMessage(orderInfo);  
    String wareOrderJson= JSON.toJSONString(messageMap);  
    Session session = null;  
    try {  
        Connection conn = activeMQUtil.getConnection();  
  
        session = conn.createSession(true, Session.SESSION_TRANSACTED);  
        Queue queue = session.createQueue("ORDER_RESULT_QUEUE");  
        MessageProducer producer = session.createProducer(queue);  
  
        TextMessage message =new ActiveMQTextMessage();  
        message.setText(wareOrderJson);  
        producer.send(message);  
  
        updateProcessStatus(orderInfo.getId(), ProcessStatus.NOTIFIED_WARE);  
  
        session.commit();  
        producer.close();  
        conn.close();  
    } catch (JMSEException e) {  
        e.printStackTrace();  
    }  
}
```

针对接口手册中需要的消息进行组织

```
public Map<String,Object> initWareOrderMessage( OrderInfo orderInfo ) {  
  
    //准备发送到仓库系统的订单  
    String wareId = orderInfo.getWareId();
```

```
HashMap<String, Object> hashMap = new HashMap<>();
hashMap.put("orderId", orderInfo.getId());
hashMap.put("consignee", orderInfo.getConsignee());
hashMap.put("consigneeTel", orderInfo.getConsigneeTel());
hashMap.put("orderComment", orderInfo.getOrderComment());
hashMap.put("orderBody", orderInfo.getOrderSubject());

hashMap.put("deliveryAddress", orderInfo.getDeliveryAddress());
hashMap.put("paymentWay", "2");//1 货到付款 2 在线支付

hashMap.put("wareId", wareId);

List<HashMap<String, String>> details = new ArrayList<>();
List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
for (OrderDetail orderDetail : orderDetailList) {
    HashMap<String, String> detailMap = new HashMap<>();
    detailMap.put("skuld", orderDetail.getSkuld());
    detailMap.put("skuNum", "" + orderDetail.getSkuNum());
    detailMap.put("skuName", orderDetail.getSkuName());
    details.add(detailMap);
}

hashMap.put("details", details);

return hashMap;
}
```

4 消费减库存结果

给仓库系统发送减库存消息后，还要接受减库存成功或者失败的消息。

同样根据《库存管理系统接口手册》中【商品减库结果消息】的说明完成。消费该消息的消息队列监听程序。

接受到消息后主要做的工作就是更新订单状态。

```
@JmsListener(destination = "SKU_DEDUCT_QUEUE", containerFactory = "jmsQueueListener")
public void consumeSkuDeduct(MapMessage mapMessage) throws JMSException {
    String orderId = mapMessage.getString("orderId");
    String status = mapMessage.getString("status");
    if ("DEDUCTED".equals(status)) {
        orderService.updateProcessStatus(orderId, ProcessStatus.WAITING_DELEVER);
        return;
    } else {
        orderService.updateProcessStatus(orderId, ProcessStatus.STOCK_EXCEPTION);
        return;
    }
}
```

```
}
```

最后一次支付完成后，所有业务全部走通应该可以在订单列表中，查看到对应的订单是待发货状态。

5 验证结果

