**品优购电商系统开发**

**第13章**

**消息中间件解决方案**

传智播客.黑马程序员

# 课程目标

目标1：消息中间件、RocketMQ

目标2：完成Spring整合RocketMQ

目标3：完成商品上架导入索引库

目标4：完成商品下架移除索引库

目标5: 完成商品上架生成商品详细页

目标6: 完成商品下架删除商品详细页

# 1.消息中间件

## 1.1什么是消息中间件

消息中间件利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行[分布式系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E4%B8%AD%E9%97%B4%E4%BB%B6/_blank)的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。对于消息中间件，常见的角色大致也就有Producer（生产者）、Consumer（消费者）

## 1.2常见的消息中间件

（1）ActiveMQ

ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现。【5W/s吞吐量】

（2）RabbitMQ

AMQP协议的领导实现，支持多种场景。淘宝的MySQL集群内部有使用它进行通讯，OpenStack开源云平台的通信组件，最先在金融行业得到运用。【5W/s吞吐量】

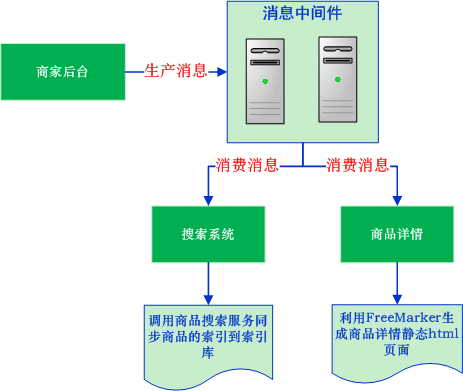
（3）RocketMQ

阿里开源分布式消息队列系统 【10W/s吞吐量】

（4）Kafka

Apache下的一个子项目 。特点：高吞吐，在一台普通的服务器上既可以达到16W/s的吞吐量；完全的分布式系统。适合处理海量数据。

## 1.3系统模块之间调用关系

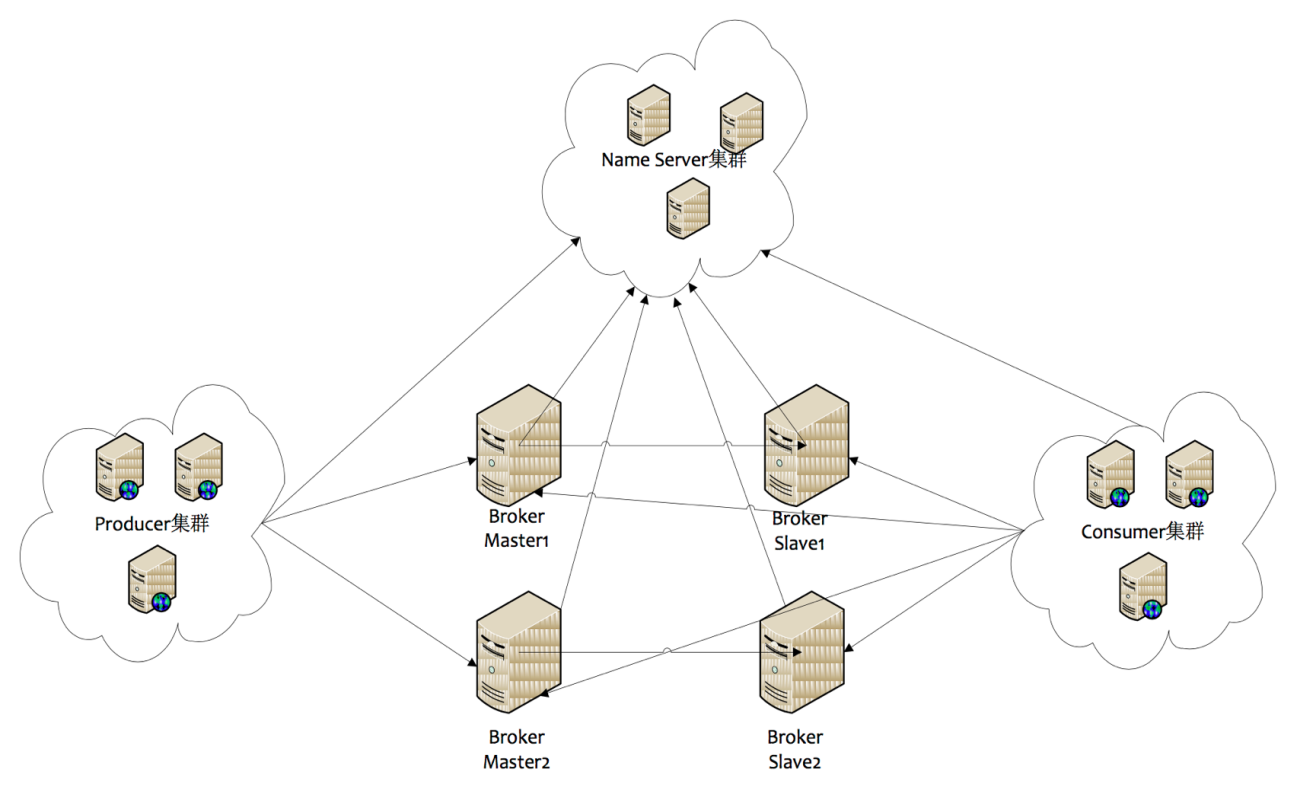


我们通过引入消息中间件RocketMQ,使得商家后台系统与搜索系统、商品详情系统**解除了耦合**，提高商家后台系统的**吞吐量**。

## 1.4 RocketMQ简介

### 1.4.1什么是RocketMQ

RocketMQ作为一款纯java、分布式、队列模型的开源消息中间件，支持事务消息、顺序消息、批量消息、定时消息、消息回溯等。RocketMQ由阿里巴巴开源。



如图所示为RocketMQ基本的部署结构，主要分为**NameServer**集群、**Broker**集群、**Producer**集群和**Consumer**集群四个部分。

### 1.4.2 NameServer集群

NameServer的作用是注册中心，类似于Zookeeper，但又有区别于它的地方。每个NameServer节点互相之间是独立的，没有任何信息交互，也就不存在任何的选主或者主从切换之类的问题，因此NameServer与Zookeeper相比更轻量级。单个NameServer节点中存储了活跃的Broker列表（包括master和slave），这里活跃的定义是与NameServer保持有心跳。

### 1.4.3 Broker集群

Broker是具体提供业务的服务器，单个Broker节点与所有的NameServer节点保持长连接及心跳，并会定时将Topic信息注册到NameServer，顺带一提底层的通信和连接都是基于Netty实现的。  
 Broker中分master和slave两种角色，每个master可以对应多个slave，但一个slave只能对应一个master，master和slave通过指定相同的Brokername，不同的BrokerId （master为0）成为一个组。master和slave之间的同步方式分为同步双写和异步复制，异步复制方式master和slave之间虽然会存在少量的延迟，但性能较同步双写方式要高出10%左右。

### 1.4.4 Producer集群

* 与nameserver的关系

单个Producer和一台nameserver保持长连接，定时查询topic配置信息，如果该nameserver挂掉，生产者会自动连接下一个nameserver，直到有可用连接为止，并能自动重连。与nameserver之间没有心跳。

* 与broker的关系

单个Producer和与其关联的所有broker保持长连接，并维持心跳。默认情况下消息发送采用轮询方式，会均匀发到对应Topic的所有queue中。

### 1.4.5 Consumer集群

* 与nameserver的关系

单个Consumer和一台nameserver保持长连接，定时查询topic配置信息，如果该nameserver挂掉，消费者会自动连接下一个nameserver，直到有可用连接为止，并能自动重连。与nameserver之间没有心跳。

* 与broker的关系

单个Consumer和与其关联的所有broker保持长连接，并维持心跳，失去心跳后，则关闭连接，并向该消费者分组的所有消费者发出通知，分组内消费者重新分配队列继续消费。

### 1.4.6消费模式

* 集群模式

在默认情况下，就是集群消费，此时消息发出去后将只有一个消费者能接收到消息。

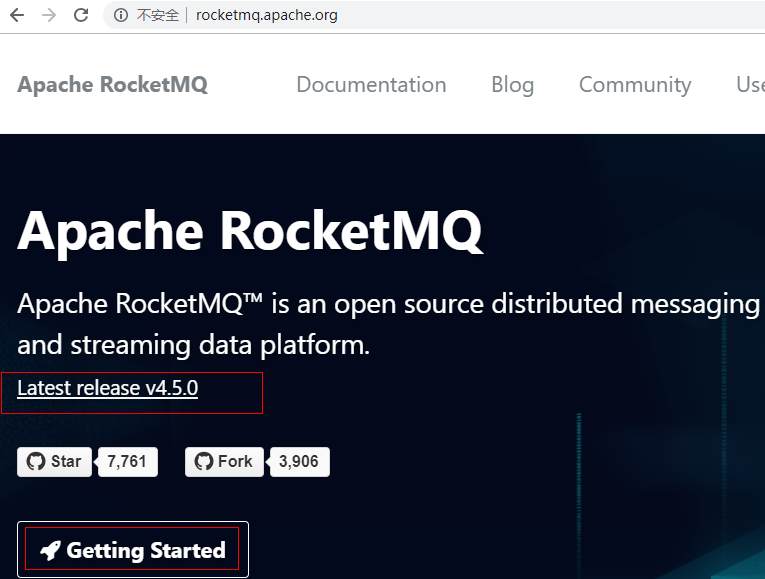
* 广播模式

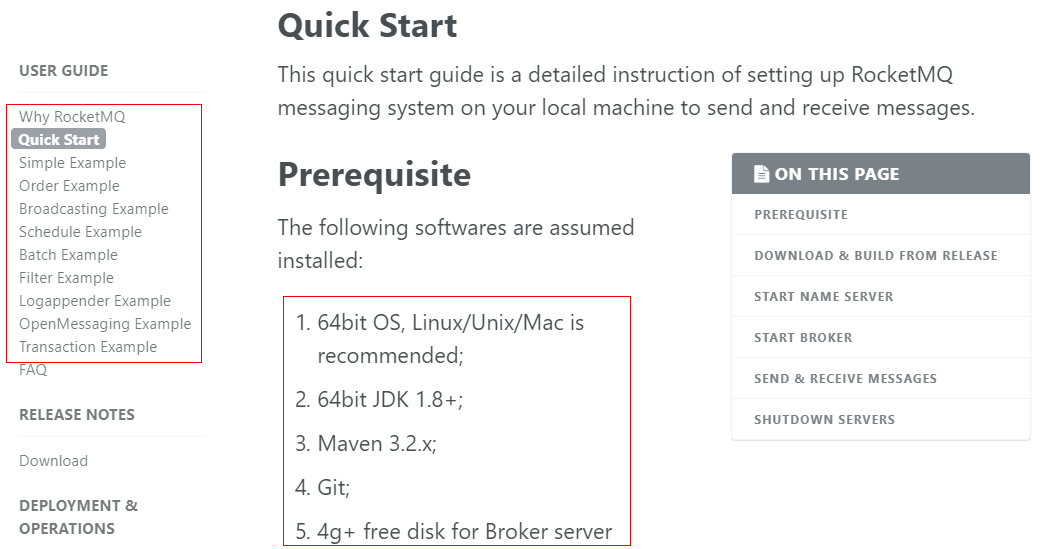
另一种消费模式，是广播消费。广播消费，类似于ActiveMQ中的发布订阅模式，消息会发给Consume Group中的每一个消费者进行消费。

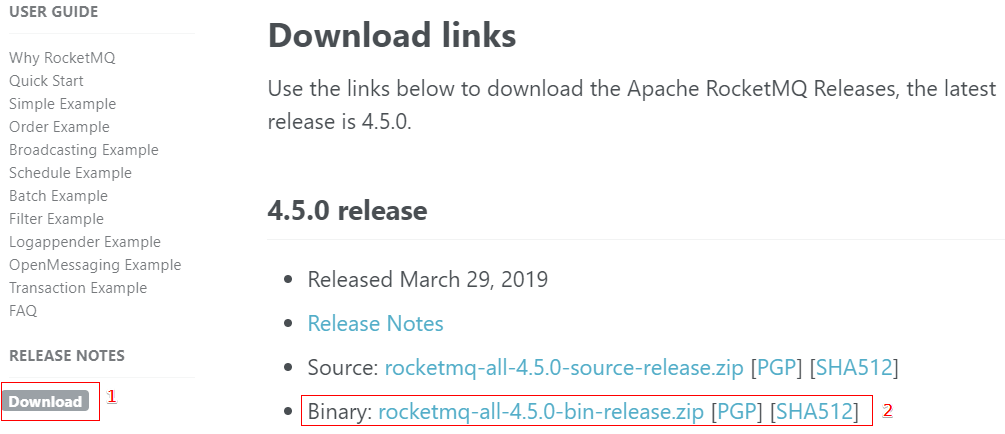
## 1.5 RocketMQ下载与安装

### 1.5.1 RocketMQ下载

官网地址：http://rocketmq.apache.org





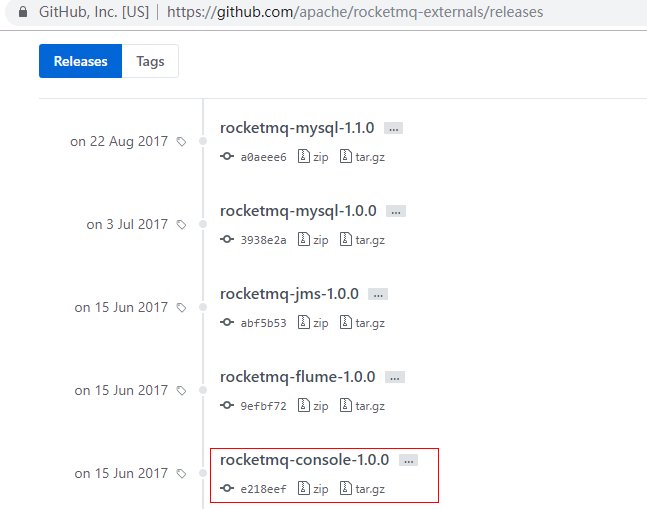


### 1.5.2管理控制台下载

RocketMQ管理控制台是一个基于SpringBoot编写的可视化Web应用，主要用于对RocketMQ提供了可视化的管理界面。

下载地址：<https://github.com/apache/rocketmq-externals/releases>

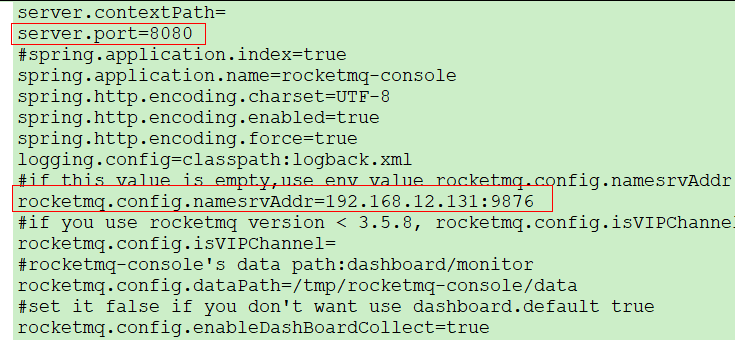
下载压缩包：rocketmq-externals-rocketmq-console-1.0.0.zip



* 修改application.properties:

解压：rocketmq-externals-rocketmq-console-1.0.0.zip

rocketmq-externals-rocketmq-console-1.0.0\rocketmq-console\src\main\resources\application.properties



* 编译启动：

进入rocketmq-externals-rocketmq-console-1.0.0\rocketmq-console\目录下执行mvn命令(cmd)：

mvn clean package -Dmaven.test.skip=true

进入rocketmq-externals-rocketmq-console-1.0.0\rocketmq-console\target

目录下执行java -jar命令(cmd)：

java -jar rocketmq-console-ng-1.0.0.jar

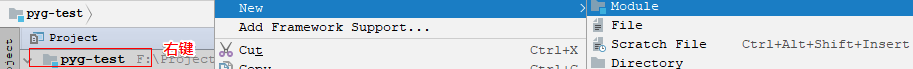
### 1.5.3 Linux安装RocketMQ

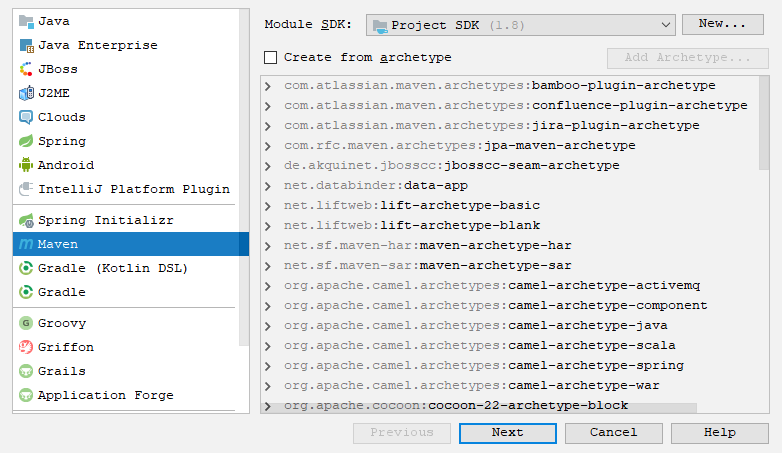
参考安装文档：**【资料\安装资料\Linux安装RocketMQ.docx】**

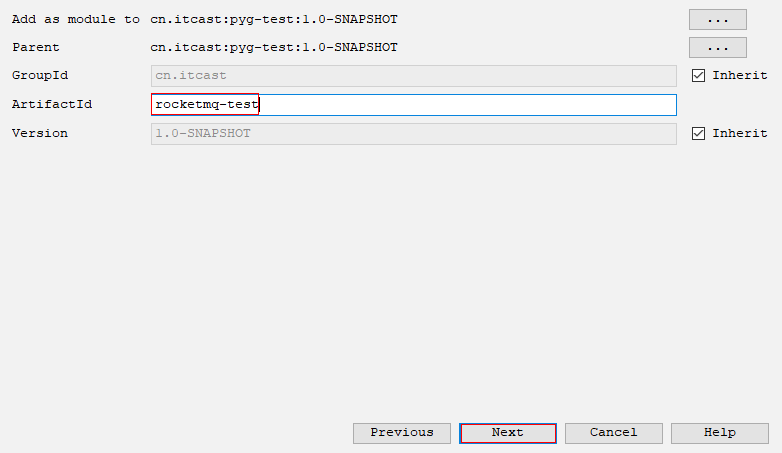
# 2.RocketMQ入门示例

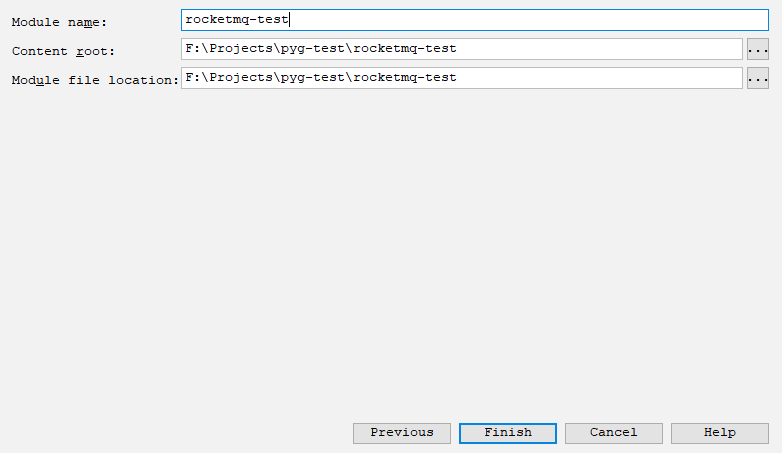
## 2.1创建测试工程

切换到pyg-test项目，创建rocketmq-test模块(jar)









rocketmq-test/pom.xml引入依赖：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**parent**>  <**artifactId**>pyg-test</**artifactId**>  <**groupId**>cn.itcast</**groupId**>  <**version**>1.0-SNAPSHOT</**version**>  </**parent**>  <**modelVersion**>4.0.0</**modelVersion**>  <**artifactId**>rocketmq-test</**artifactId**>  <**dependencies**>  *<!-- 配置rocketmq-client -->* <**dependency**>  <**groupId**>org.apache.rocketmq</**groupId**>  <**artifactId**>rocketmq-client</**artifactId**>  <**version**>4.5.0</**version**>  </**dependency**>  </**dependencies**> </**project**> |

## 2.2简单消息

### 2.2.1消息生产者

* 创建默认的MQ消息生产者
* 创建消息对象
* 消息生产者发送消息
* 关闭消息生产者

rocketmq-test\src\main\java\cn\itcast\rocketmq\simple\ProducerTest.java，消息生产者:

|  |
| --- |
| **public class** ProcuderTest {  **public static void** main(String[] args) **throws** Exception{  ***// 1. 创建默认的MQ消息生产者(指定组名)*** DefaultMQProducer producer = **new** DefaultMQProducer(**"producerGroup1"**);  *// 1.1 设置NameServer连接地址，如果集群环境，多个之间用分号分隔* producer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 启动，开始连接* producer.start();  ***// 2. 创建消息对象*** Message message = **new** Message(**"demo1"**, *// 主题名称* **"tag1"**, *// 标签名称* **"key1"**, *// key名称* **"RocketMQ，我来了！"**.getBytes(  RemotingHelper.***DEFAULT\_CHARSET***)); *// 消息内容体* ***// 3. 消息生产者发送消息*** producer.send(message);   ***// 4. 关闭消息生产者*** producer.shutdown();  } } |

### 2.2.2消息消费者

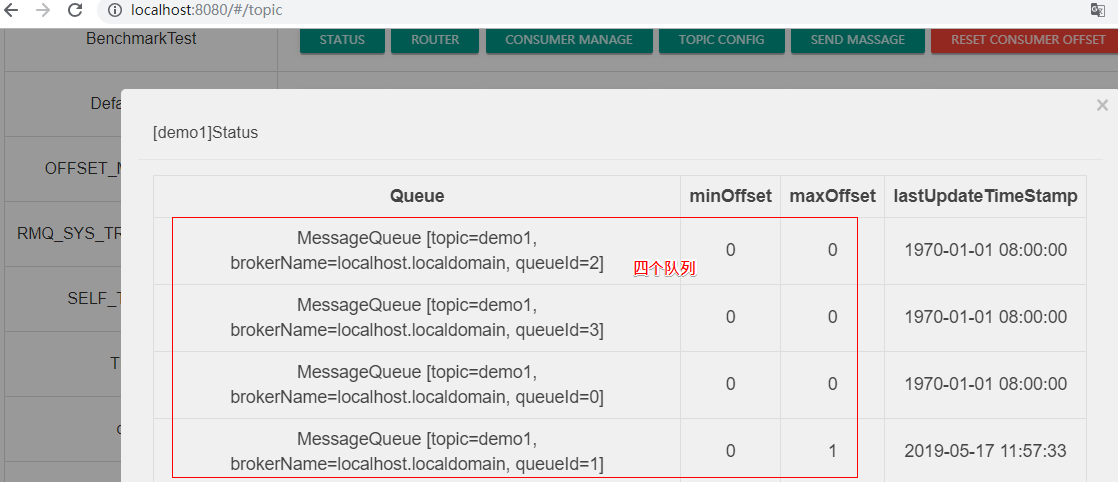
* 创建默认的消息消费者
* 注册消息监听器，消费消息
* 启动消息消费者

rocketmq-test\src\main\java\cn\itcast\rocketmq\simple\ConsumerTest.java，消息消费者：

|  |
| --- |
| **public class** ConsumerTest {  **public static void** main(String[] args) **throws** Exception{  ***// 1. 创建默认消息消费者(指定组名)*** DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer(**"consumerGroup1"**);  *// 1.1 设置NameServer地址，集群环境，多个之间用分号分隔* consumer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 设置从哪里开始消费  // ConsumeFromWhere.CONSUME\_FROM\_FIRST\_OFFSET:从队列头部开始消费  // ConsumeFromWhere.CONSUME\_FROM\_LAST\_OFFSET: 从队列尾部开始消费* consumer.setConsumeFromWhere(ConsumeFromWhere.***CONSUME\_FROM\_LAST\_OFFSET***);  *// 1.3 设置订阅的主题与标签* consumer.subscribe(**"demo1"**, **"tag1"**);   ***// 2. 注册消息监听器，消费消息*** consumer.registerMessageListener(**new** MessageListenerConcurrently() {  @Override  **public** ConsumeConcurrentlyStatus consumeMessage(  List<MessageExt> messageExts,ConsumeConcurrentlyContext context) {  **try** {  *// 获取消息(默认每次取一个消息)* MessageExt messageExt = messageExts.get(0);  *// 获取主题* String topic = messageExt.getTopic();  *// 获取标签* String tags = messageExt.getTags();  *// 获取消息内容* **String content = new String(messageExt.getBody(), "UTF-8");**  System.***out***.println(**"topic = "** + topic);  System.***out***.println(**"tags = "** + tags);  System.***out***.println(**"content = "** + content);  }**catch** (Exception ex){  *// 稍后再消费* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  });  ***// 3. 开启消息消费者*** consumer.start();  } } |

## 2.3顺序消息

消息有序指的是可以按照消息的发送顺序来消费，RocketMQ可以严格的保证消息有序。默认把消息分布式存储在topic中的4个队列中，我们可以指定消息存储到topic中的1个队列中，就样消息就会有顺序。



### 2.3.1消息生产者

* 创建默认的MQ消息生产者
* 创建消息对象
* 消息生产者发送消息
* 关闭消息生产者

rocketmq-test\src\main\java\cn\itcast\rocketmq\order\ProducerTest.java，消息生产者:

|  |
| --- |
| **public class** ProcuderTest {  **public static void** main(String[] args) **throws** Exception{  ***// 1. 创建默认的MQ消息生产者(指定组名)*** DefaultMQProducer producer = **new** DefaultMQProducer(**"producerGroup2"**);  *// 1.1 设置NameServer连接地址，如果集群环境，多个之间用分号分隔* producer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 启动，开始连接* producer.start();  ***// 2. 创建消息对象*** **for** (**int** i = 1; i<= 20; i++) {  Message message = **new** Message(**"demo2"**, *// 主题名称* **"tag2"**, *// 标签名称* **"key2"**, *// key名称* (**"RocketMQ，我来了，顺序消息。"** + i).getBytes(  RemotingHelper.***DEFAULT\_CHARSET***)); *// 消息内容体*  ***// 3. 消息生产者发送消息*** producer.send(message, **new** MessageQueueSelector() {  @Override  **public** MessageQueue select(List<MessageQueue> list,  Message message, Object i) {  **return** list.get((Integer)i);  }  }, **1**); ***// 队列的索引号，从0开始*** }  ***// 4. 关闭消息生产者*** producer.shutdown();  } } |



### 2.3.2消息消费者

* 创建默认的消息消费者
* 注册消息监听器，消费消息
* 启动消息消费者

rocketmq-test\src\main\java\cn\itcast\rocketmq\order\ConsumerTest.java，消息消费者：

|  |
| --- |
| **public class** ConsumerTest {   **public static void** main(String[] args) **throws** Exception{   ***// 1. 创建默认消息消费者(指定组名)*** DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer(**"consumerGroup2"**);  *// 1.1 设置NameServer地址，集群环境，多个之间用分号分隔* consumer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 设置从哪里开始消费  // ConsumeFromWhere.CONSUME\_FROM\_FIRST\_OFFSET:从队列头部开始消费  // ConsumeFromWhere.CONSUME\_FROM\_LAST\_OFFSET:从队列尾部开始消费* consumer.setConsumeFromWhere(ConsumeFromWhere.***CONSUME\_FROM\_LAST\_OFFSET***);  *// 1.3 设置批量消费消息最大数量* consumer.setConsumeMessageBatchMaxSize(5);  *// 1.4 设置订阅的主题与标签* consumer.subscribe(**"demo2"**, **"tag2"**);   ***// 2. 注册消息监听器，消费消息*** consumer.registerMessageListener(**new MessageListenerOrderly**() {  @Override  **public** ConsumeOrderlyStatus consumeMessage(  List<MessageExt> messageExts,ConsumeOrderlyContext context) {  **try** {  *// 循环消费消息(一次5个消息)* **for** (MessageExt messageExt : messageExts) {  *// 获取主题* String topic = messageExt.getTopic();  *// 获取标签* String tags = messageExt.getTags();  *// 获取消息内容* String content = **new** String(messageExt.getBody(), **"UTF-8"**);  System.***out***.println(**"topic = "** + topic);  System.***out***.println(**"tags = "** + tags);  System.***out***.println(**"content = "** + content);  }  }**catch** (Exception ex){  ex.printStackTrace();  }  *// 消费成功* **return** ConsumeOrderlyStatus.***SUCCESS***;  }  });   ***// 3. 开启消息消费者*** consumer.start();  } } |

## 2.4广播消息

* 集群模式：集群消费，此时消息发出去后将只有一个消费者能接收到消息。(默认)
* 广播模式：广播消费，类似于ActiveMQ中的发布订阅模式，消息会发给Consume Group中的每一个消费者进行消费(多个消费者)。

### 2.4.1消息生产者

* 创建默认的MQ消息生产者
* 创建消息对象
* 消息生产者发送消息
* 关闭消息生产者

rocketmq-test\src\main\java\cn\itcast\rocketmq\broadcast\ProducerTest.java，消息生产者:

|  |
| --- |
| **public class** ProcuderTest {  **public static void** main(String[] args) **throws** Exception{  ***// 1. 创建默认的MQ消息生产者(指定组名)*** DefaultMQProducer producer = **new** DefaultMQProducer(**"producerGroup3"**);  *// 1.1 设置NameServer连接地址，如果集群环境，多个之间用分号分隔* producer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 启动，开始连接* producer.start();   ***// 2. 创建消息对象*** Message message = **new** Message(**"demo3"**, *// 主题名称* **"tag3"**, *// 标签名称* **"key3"**, *// key的名称* **"RocketMQ，我来了！广播消息。"**.getBytes(  RemotingHelper.***DEFAULT\_CHARSET***)); *// 消息内容体* ***// 3. 消息生产者发送消息*** producer.send(message);   ***// 4. 关闭消息生产者*** producer.shutdown();  } } |

### 2.4.2消息消费者

* 创建默认的消息消费者
* 注册消息监听器，消费消息
* 启动消息消费者

rocketmq-test\src\main\java\cn\itcast\rocketmq\broadcast\ConsumerTest.java，消息消费者：

|  |
| --- |
| **public class** ConsumerTest {  **public static void** main(String[] args) **throws** Exception{   ***// 1. 创建默认消息消费者(指定组名)*** DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer(**"consumerGroup3"**);  *// 1.1 设置NameServer地址，集群环境，多个之间用分号分隔* consumer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 设置从哪里开始消费  // ConsumeFromWhere.CONSUME\_FROM\_FIRST\_OFFSET:从队列头问开始消费  // ConsumeFromWhere.CONSUME\_FROM\_LAST\_OFFSET:从队列尾部开始消费* consumer.setConsumeFromWhere(ConsumeFromWhere.***CONSUME\_FROM\_LAST\_OFFSET***);  *// 1.3 设置订阅的主题与标签* consumer.subscribe(**"demo3"**, **"tag3"**);  *// 1.4 设置消息的模式  // MessageModel.BROADCASTING: 广播模式  // MessageModel.CLUSTERING: 集群模式* consumer.setMessageModel(MessageModel.***BROADCASTING***);   ***// 2. 注册消息监听器，消费消息*** consumer.registerMessageListener(**new** MessageListenerConcurrently() {  @Override  **public** ConsumeConcurrentlyStatus consumeMessage(  List<MessageExt> messageExts,ConsumeConcurrentlyContext context) {  **try** {  *// 获取消息(默认每次取一个消息)* MessageExt messageExt = messageExts.get(0);  *// 获取主题* String topic = messageExt.getTopic();  *// 获取标签* String tags = messageExt.getTags();  *// 获取消息内容* String content = **new** String(messageExt.getBody(), **"UTF-8"**);  System.***out***.println(**"topic = "** + topic);  System.***out***.println(**"tags = "** + tags);  System.***out***.println(**"content = "** + content);  }**catch** (Exception ex){  *// 消费失败* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  });   ***// 3. 开启消息消费者*** consumer.start();  } } |

## 2.5批量消息

### 2.5.1消息生产者

* 创建默认的MQ消息生产者
* 循环创建消息对象
* 消息生产者发送消息
* 关闭消息生产者

rocketmq-test\src\main\java\cn\itcast\rocketmq\batch\ProducerTest.java，消息生产者:

|  |
| --- |
| **public class** ProcuderTest {  **public static void** main(String[] args) **throws** Exception{  ***// 1. 创建默认的MQ消息生产者(指定组名)*** DefaultMQProducer producer = **new** DefaultMQProducer(**"producerGroup4"**);  *// 1.1 设置NameServer连接地址，如果集群环境，多个之间用分号分隔* producer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 启动，开始连接* producer.start();   ***// 2. 循环创建消息对象*** List<Message> messages = **new** ArrayList<>();  **for** (**int** i = 1; i <= 20; i++) {  Message message = **new** Message(**"demo4"**, *// 主题名称* **"tag"** + i, *// 标签名称* **"key"** + i, *// key名称* (**"RocketMQ，我来了！批量消息。"** + i).getBytes(  RemotingHelper.***DEFAULT\_CHARSET***)); *// 消息内容体* messages.add(message);  }   ***// 3. 消息生产者批量发送消息*** producer.send(messages);   ***// 4. 关闭消息生产者*** producer.shutdown();  } } |

### 2.5.2消息消费者

* 创建默认的消息消费者
* 注册消息监听器，消费消息
* 启动消息消费者

rocketmq-test\src\main\java\cn\itcast\rocketmq\batch\ConsumerTest.java，消息消费者：

|  |
| --- |
| **public class** ConsumerTest {  **public static void** main(String[] args) **throws** Exception{  ***// 1. 创建默认消息消费者(指定组名)*** DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer(**"consumerGroup4"**);  *// 1.1 设置NameServer地址，集群环境，多个之间用分号分隔* consumer.setNamesrvAddr(**"192.168.12.131:9876"**);  *// 1.2 设置从哪里开始消费  // ConsumeFromWhere.CONSUME\_FROM\_FIRST\_OFFSET:从队列头部开始消费  // ConsumeFromWhere.CONSUME\_FROM\_LAST\_OFFSET:从队列尾部开始消费* consumer.setConsumeFromWhere(ConsumeFromWhere.***CONSUME\_FROM\_LAST\_OFFSET***);  *// 1.3 设置批量消费消息的最大数量* consumer.setConsumeMessageBatchMaxSize(5);  *// 1.4 设置订阅的主题与标签* consumer.subscribe(**"demo4"**, **"\*"**);  *// 1.5 设置消息的模式  // MessageModel.BROADCASTING: 广播模式  // MessageModel.CLUSTERING: 集群模式* consumer.setMessageModel(MessageModel.***CLUSTERING***);   ***// 2. 注册消息监听器，消费消息*** consumer.registerMessageListener(**new** MessageListenerConcurrently() {  @Override  **public** ConsumeConcurrentlyStatus consumeMessage(  List<MessageExt> messageExts,ConsumeConcurrentlyContext context) {  **try** {  *// 循环消费消息(一次5个消息)* **for** (MessageExt messageExt : messageExts) {  *// 获取主题* String topic = messageExt.getTopic();  *// 获取标签* String tags = messageExt.getTags();  *// 获取消息内容* String content = **new** String(messageExt.getBody(), **"UTF-8"**);  System.***out***.println(**"topic = "** + topic);  System.***out***.println(**"tags = "** + tags);  System.***out***.println(**"content = "** + content);  }  }**catch** (Exception ex){  *// 消费失败* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  });   ***// 3. 开启消息消费者*** consumer.start();  } } |

# Spring整合RocketMQ

## 3.1 配置依赖jar包

|  |
| --- |
| *<!-- 配置rocketmq-client -->* <**dependency**>  <**groupId**>org.apache.rocketmq</**groupId**>  <**artifactId**>rocketmq-client</**artifactId**>  <**version**>4.5.0</**version**> </**dependency**> *<!-- spring-context -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-context</**artifactId**>  <**version**>5.1.3.RELEASE</**version**> </**dependency**> |

## 3.2 消息生产者

* 配置部分:

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd"**>   *<!--############# 配置消息生产者 #############-->  <!-- 配置默认的MQ消息生产者 -->* <**bean id="mqProducer"**  **class="org.apache.rocketmq.client.producer.DefaultMQProducer"  init-method="start"  destroy-method="shutdown"**>  *<!-- 设置组名 -->* <**property name="producerGroup" value="prodeucerGroup5"** />  *<!-- 设置NameServer连接地址-->* <**property name="namesrvAddr" value="192.168.12.131:9876"** />  </**bean**> </**beans**> |

* 编程部分：

rocketmq-test\src\main\java\cn\itcast\rocketmq\spring\ProducerTest.java

|  |
| --- |
| **public class** ProducerTest {  **public static void** main(String[] args) **throws** Exception{  ApplicationContext ac = **new** ClassPathXmlApplicationContext(**"applicationContext-rocketmq.xml"**);  MQProducer producer = ac.getBean(DefaultMQProducer.**class**);  *// 创建消息对象* Message message = **new** Message(  **"demo5"**, *// 主题名称* **"tag5"**, *// 标签名字* **"Spring整合RocketMQ！"**  .getBytes(**"UTF-8"**));*//消息内容字节数组  // 发送消息* SendResult result = producer.send(message);  System.***out***.println(**"result = "** + result);  } } |

## 3.3 消息消费者

* 配置部分：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd"**>   *<!--############# 配置消息消费者 #############-->  <!-- 配置默认的MQ消息消费者 -->* <**bean id="mqPushConsumer"  class="org.apache.rocketmq.client.consumer.DefaultMQPushConsumer"  init-method="start"  destroy-method="shutdown"**>  *<!-- 设置消息模式: (广播BROADCASTING | 集群CLUSTERING) -->* <**property name="messageModel" value="CLUSTERING"**/>  *<!-- 设置消费者组名 -->* <**property name="consumerGroup" value="consumerGroup5"**/>  *<!-- 设置NameServer地址 -->* <**property name="namesrvAddr" value="192.168.12.131:9876"**/>  *<!-- 设置消息监听器 -->* <**property name="messageListener" ref="messageListener"**/>  *<!-- 设置订阅的主题与标签 -->* <**property name="subscription"**>  <**map**>  *<!-- key: 主题 value：标签-->* <**entry key="demo5" value="tag5"**/>  </**map**>  </**property**>  </**bean**>  *<!-- 配置消息监听器 -->* <**bean id="messageListener"**  **class="cn.itcast.rocketmq.spring.listener.MessageListener"**/> </**beans**> |

* 编程部分：

|  |
| --- |
| *\*\* 自定义消息监听器 \*/* **public class** MessageListener **implements** MessageListenerConcurrently{  */\*\* 消费消息 \*/* @Override  **public** ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,  ConsumeConcurrentlyContext context) {  **try** {  *// 获取消息内容封装对象* MessageExt messageExt = msgs.get(0);  *// 获取消息主题* String topic = messageExt.getTopic();  *// 获取消息标签* String tags = messageExt.getTags();  *// 获取消息内容体* String content = **new** String(messageExt.getBody(),**"UTF-8"**);  *// 输出消息信息* System.***out***.println(**"topic:"** + topic +**",tags:"** + tags + **",content:"** + content);  } **catch** (Exception e) {  *// 消费失败* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  } } |

**说明：自定义消息监听器需实现MessageListenerConcurrently接口。**

# 4.商品上架【导入索引库】

## 4.1需求分析

商家执行商品上架后，向RocketMQ发送消息（goodsIds），搜索系统从RocketMQ接收到消息查询上架通过的SKU商品数据并同步到ElasticSearch索引库。

## 4.2消息生产者【商家后台】

### 4.2.1整合RocketMQ

#### **4.2.1.1配置依赖**

pinyougou-shop-web/pom.xml，引入依赖：

|  |
| --- |
| *<!-- rocketmq-client -->* <**dependency**>  <**groupId**>org.apache.rocketmq</**groupId**>  <**artifactId**>rocketmq-client</**artifactId**> </**dependency**> |

#### **4.2.1.2配置文件**

pinyougou-shop-web/src/main/resources/目录下，创建

applicationContext-rocketmq.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd"**>    *<!--############# 配置消息生产者 #############-->  <!-- 配置默认的MQ消息生产者 -->* <**bean id="mqProducer"  class="org.apache.rocketmq.client.producer.DefaultMQProducer"  init-method="start"  destroy-method="shutdown"**>  *<!-- 设置组名 -->* <**property name="producerGroup" value="ITEM\_PRODUCER"**/>  *<!-- 设置NameServer连接地址-->* <**property name="namesrvAddr" value="${namesrvAddr}"**/>  </**bean**> </**beans**> |

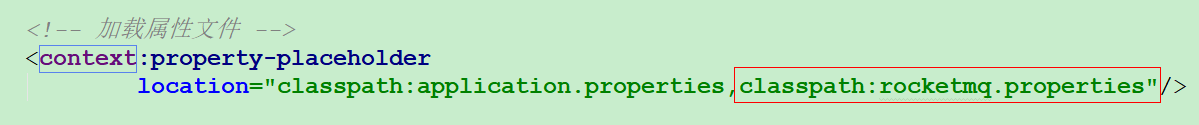
修改pinyougou-shop-web/src/main/webapp/WEB-INF/web.xml文件：

|  |
| --- |
| ***......***  *<!-- 配置Spring MVC前端控制器(核心控制器) -->* <**servlet**>  <**servlet-name**>pinyougou</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:springmvc.xml,  classpath:applicationContext-shiro.xml,  classpath:applicationContext-rocketmq.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**> </**servlet**>  **......** |

新增pinyougou-shop-web/src/main/resources/rocketmq.properties：

|  |
| --- |
| *# 配置nameserver连接地址* **namesrvAddr**=**192.168.12.131:9876** |

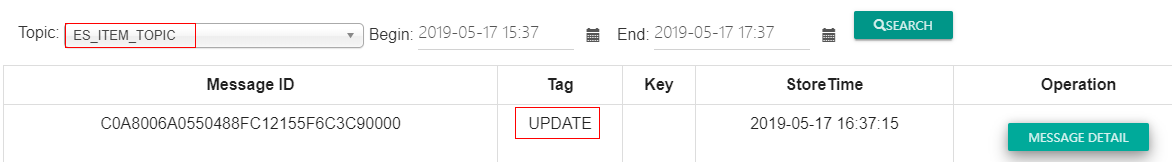
修改pinyougou-shop-web\src\main\resources\springmvc.xml：



### 4.2.2发送消息

pinyougou-shop-web/src/main/java/com/pinyougou/shop/controller/GoodsController.java：

|  |
| --- |
| @Autowired **private** MQProducer **mqProducer**;  */\*\* 商家商品上下架(修改可销售状态) \*/* @GetMapping(**"/updateMarketable"**) **public boolean** updateMarketable(Long[] ids, String status){  **try**{  *// 修改数据库表可销售状态* **goodsService**.updateStatus(**"is\_marketable"**, ids, status);  *// 判断商品上下架状态* **if** (**"1"**.equals(status)){ *// 表示商品上架  // 发送消息，生成商品索引* **mqProducer**.send(**new** Message(**"ES\_ITEM\_TOPIC"**, **"UPDATE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**))); }**else** { *// 表示商品下架* }  **return true**;  }**catch** (Exception ex){  ex.printStackTrace();  }  **return false**; } |



## 4.3消息消费者【搜索系统】

### 4.3.1整合RocketMQ

#### **4.3.1.1配置依赖**

pinyougou-search-web/pom.xml，添加依赖：

|  |
| --- |
| <**dependencies**>  *<!-- rocketmq-client -->* <**dependency**>  <**groupId**>org.apache.rocketmq</**groupId**>  <**artifactId**>rocketmq-client</**artifactId**>  </**dependency**> </**dependencies**> |

#### **4.3.1.2配置文件**

在pinyougou-search-web/src/main/resources/目录下，创建

applicationContext-rocketmq.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd"**>  *<!-- 配置加载属性文件 -->* <**context:property-placeholder  location="classpath:rocketmq.properties"**/>   *<!--############# 配置消息消费者 #############-->  <!-- 配置默认的MQ消息消费者 -->* <**bean id="mqPushConsumer"  class="org.apache.rocketmq.client.consumer.DefaultMQPushConsumer"  init-method="start"  destroy-method="shutdown"**>  *<!-- 设置消息模式: (广播BROADCASTING | 集群CLUSTERING) -->* <**property name="messageModel" value="CLUSTERING"**/>  *<!-- 设置消费者组名 -->* <**property name="consumerGroup" value="ES\_ITEM\_CONSUMER"**/>  *<!-- 设置NameServer地址 -->* <**property name="namesrvAddr" value="${namesrvAddr}"**/>  *<!-- 设置消息监听器 -->* <**property name="messageListener" ref="messageListener"**/>  *<!-- 设置订阅信息 -->* <**property name="subscription"**>  <**map**>  *<!-- 指定主题和标签 -->* <**entry key="ES\_ITEM\_TOPIC" value="\*"**/>  </**map**>  </**property**>  </**bean**>  *<!-- 配置消息监听器 -->* <**bean id="messageListener"  class="com.pinyougou.search.listener.ItemMessageListener"**/> </**beans**> |

新增pinyougou-search-web/src/main/resources/rocketmq.properties：

|  |
| --- |
| *# 配置nameserver连接地址* **namesrvAddr**=**192.168.12.131:9876** |

pinyougou-search-web/src/main/resources/springmvc.xml：

|  |
| --- |
| *<!-- 配置开启组件扫描 -->* <**context:component-scan base-package="com.pinyougou.search"**/>  *......*  *<!-- 配置采用包扫描来引用服务，产生服务接口的代理对象 -->* <**dubbo:annotation package="com.pinyougou.search"**/> |

修改pinyougou-search-web/src/main/webapp/WEB-INF/web.xml文件：

|  |
| --- |
| *<!-- 配置Spring MVC前端控制器(核心控制器) -->* <**servlet**>  <**servlet-name**>pinyougou</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:springmvc.xml,  classpath:applicationContext-rocketmq.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**> </**servlet**> <**servlet-mapping**>  <**servlet-name**>pinyougou</**servlet-name**>  <**url-pattern**>/</**url-pattern**> </**servlet-mapping**> |

### 4.3.2消费消息

pinyougou-search-web/src/main/java/com/pinyougou/search/listener包下，创建ItemMessageListener.java消息监听器：

|  |
| --- |
| */\*\* 商品消息监听器 \*/* **public class** ItemMessageListener **implements** MessageListenerConcurrently{   @Reference(timeout = 30000)  **private** GoodsService **goodsService**;  @Reference(timeout = 30000)  **private** ItemSearchService **itemSearchService**;   @Override  **public** ConsumeConcurrentlyStatus  consumeMessage(List<MessageExt> list,  ConsumeConcurrentlyContext consumeConcurrentlyContext) {  System.***out***.println(**"===ItemMessageListener==="**);  **try** {  *// 获取消息封装对象* MessageExt messageExt = list.get(0);  *// 获取消息内容* String content = **new** String(messageExt.getBody(),**"UTF-8"**);  *// 把json字符串转化成List集合* List<Long> ids = JSON.*parseArray*(content, Long.**class**);  System.***out***.println(**"ids:"** + ids);  System.***out***.println(**"tags:"** + messageExt.getTags());  *// 同步更新商品索引数据* **if** (**"UPDATE"**.equals(messageExt.getTags())) {  *// 查询上架的SKU商品数据* List<Item> itemList = **goodsService**.findItemByGoodsId(ids);  *// 判断集合* **if** (itemList.size() > 0) {  *// 把List<Item>转化成List<SolrItem>* List<EsItem> esItems = **new** ArrayList<>();  **for** (Item item1 : itemList) {  EsItem esItem = **new** EsItem();  esItem.setId(item1.getId());  esItem.setTitle(item1.getTitle());  esItem.setPrice(item1.getPrice().doubleValue());  esItem.setImage(item1.getImage());  esItem.setGoodsId(item1.getGoodsId());  esItem.setCategory(item1.getCategory());  esItem.setBrand(item1.getBrand());  esItem.setSeller(item1.getSeller());  esItem.setUpdateTime(item1.getUpdateTime());  *// 把SKU的规格json字符串，转化成Map集合(fastjson)* Map<String, String> spec = JSON  .*parseObject*(item1.getSpec(), Map.**class**);  *// 规格嵌套Field* esItem.setSpec(spec);  *// 添加到集合* esItems.add(esItem);  }  *// 把SKU商品数据同步到索引库* **itemSearchService**.saveOrUpdate(esItems);  }  }  *// 删除商品索引数据* **if** (**"DELETE"**.equals(messageExt.getTags())) {    }  }**catch** (Exception ex){  *// 消费失败* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  } } |

### 4.3.3查询上架商品列表

pinyougou-interface\src\main\java\com\pinyougou\service\GoodsService.java：

|  |
| --- |
| */\*\* 查询上架的SKU商品数据 \*/* List<Item> findItemByGoodsId(List<Long> ids); |

pinyougou-sellergoods-service\src\main\java\com\pinyougou\sellergoods\service\impl\GoodsServiceImpl.java：

|  |
| --- |
| */\*\* 查询上架的SKU商品数据 \*/* @Override **public** List<Item> findItemByGoodsId(List<Long> ids){  **try**{  */\*\* 创建示范对象 \*/* Example example = **new** Example(Item.**class**);  */\*\* 创建查询条件对象 \*/* Example.Criteria criteria = example.createCriteria();  */\*\* 添加in查询条件 \*/* criteria.andIn(**"goodsId"**, ids);  */\*\* 查询数据 \*/* **return itemMapper**.selectByExample(example);  }**catch** (Exception ex){  **throw new** RuntimeException(ex);  } } |

### 4.3.4添加或修改商品索引

pinyougou-interface\src\main\java\com\pinyougou\service\ItemSearchService.java：

|  |
| --- |
| */\*\* 批量添加或修改商品索引 \*/* **void** saveOrUpdate(List<EsItem> esItems); |

pinyougou-search-service\src\main\java\com\pinyougou\search\service\impl\ItemSearchServiceImpl.java：

|  |
| --- |
| *// 添加或修改商品索引* @Override **public void** saveOrUpdate(List<EsItem> esItems) {  **try**{   **esItemDao.saveAll(esItems);**  }**catch** (Exception ex){  **throw new** RuntimeException(ex);  } } |

# 5.商品下架【移除索引库】

## 5.1需求分析

商家执行商品下架时，发送消息通知搜索系统删除索引库该SPU对应SKU商品的所有索引。

## 5.2消息生产者【商家后台】

### 5.2.1发送消息

pinyougou-shop-web/src/main/java/com/pinyougou/shop/controller/GoodsController.java：

|  |
| --- |
| */\*\* 商家商品上下架(修改可销售状态) \*/* @GetMapping(**"/updateMarketable"**) **public boolean** updateMarketable(Long[] ids, String status){  **try**{  *// 修改数据库表可销售状态* **goodsService**.updateStatus(**"is\_marketable"**, ids, status);  *// 判断商品上下架状态* **if** (**"1"**.equals(status)){ *// 表示商品上架  // 发送消息，生成商品索引* **mqProducer**.send(**new** Message(**"ES\_ITEM\_TOPIC"**, **"UPDATE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));}**else** { *// 表示商品下架  // 发送消息，删除商品索引* **mqProducer**.send(**new** Message(**"ES\_ITEM\_TOPIC"**, **"DELETE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));}  **return true**;  }**catch** (Exception ex){  ex.printStackTrace();  }  **return false**; } |

## 5.3消息消费者【搜索系统】

### 5.3.1消费消息

pinyougou-search-web\src\main\java\com\pinyougou\search\listener\ItemMessageListener.java消息监听器：

|  |
| --- |
| */\*\* 商品消息监听器 \*/* **public class** ItemMessageListener **implements** MessageListenerConcurrently{  @Reference(timeout = 30000)  **private** GoodsService **goodsService**;  @Reference(timeout = 30000)  **private** ItemSearchService **itemSearchService**;  @Override  **public** ConsumeConcurrentlyStatus  consumeMessage(List<MessageExt> list,  ConsumeConcurrentlyContext consumeConcurrentlyContext) {  System.***out***.println(**"===ItemMessageListener==="**);  **try** {  *// 获取消息封装对象* MessageExt messageExt = list.get(0);  *// 获取消息内容* String content = **new** String(messageExt.getBody(),**"UTF-8"**);  *// 把json字符串转化成List集合* List<Long> ids = JSON.*parseArray*(content, Long.**class**);  System.***out***.println(**"ids:"** + ids);  System.***out***.println(**"tags:"** + messageExt.getTags());  *// 同步更新商品索引数据* **if** (**"UPDATE"**.equals(messageExt.getTags())) {  ......  }  *// 删除商品索引数据* **if** (**"DELETE"**.equals(messageExt.getTags())) {  **itemSearchService**.delete(ids);  }  }**catch** (Exception ex){  *// 消费失败* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  } } |

### 5.3.2删除商品索引

pinyougou-interface\src\main\java\com\pinyougou\service\ItemSearchService.java：

|  |
| --- |
| */\*\* 删除商品索引 \*/* **void** delete(List<Long> goodsIds); |

pinyougou-search-service\src\main\java\com\pinyougou\search\service\impl\ItemSearchServiceImpl.java：

|  |
| --- |
| @Override **public void** delete(List<Long> goodsIds) {  **try**{  DeleteQuery dq = **new** DeleteQuery();  dq.setIndex(**"pinyougou"**);  dq.setType(**"item"**);  dq.setQuery(QueryBuilders.*termsQuery*(**"goodsId"**, goodsIds));  **esTemplate**.delete(dq);  }**catch** (Exception ex){  **throw new** RuntimeException(ex);  } } |

# 6.商品上架【生成商品详细页】

## 6.1为什么需要静态化

商品详情页是消费者了解商品的主要途径，访问的频率非常高。

所以需要对商品详情页进行优化，提高访问的速度。

优化方案：

A.使用redis添加缓存

redis的访问速度快，能够较大的提升查询数据的速度，减轻MySQL数据库的访问压力。

B.使用静态化

**静态化**：把动态页面（jsp，php，asp等)，转变成静态页面（html）

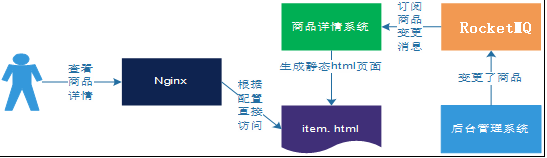
优点：

① 访问静态页面不需要经过程序处理，可以提高速度

② 稳定性高

③ 静态页面相对于动态页面更容易被搜索引擎收录（SEO）

## 6.2静态化访问流程



## 6.3需求分析

商家执行商品上架后，向RocketMQ发送消息（商品ID），商品详情系统从RocketMQ接收到消息后执行网页生成操作。

## 6.4消息生产者【商家后台】

### 6.4.1发送消息

pinyougou-shop-web/src/main/java/com/pinyougou/manager/controller/GoodsController.java：

|  |
| --- |
| */\*\* 商家商品上下架(修改可销售状态) \*/* @GetMapping(**"/updateMarketable"**) **public boolean** updateMarketable(Long[] ids, String status){  **try**{  *// 修改数据库表可销售状态* **goodsService**.updateStatus(**"is\_marketable"**, ids, status);  *// 判断商品上下架状态* **if** (**"1"**.equals(status)){ *// 表示商品上架  // 发送消息，生成商品索引* **mqProducer**.send(**new** Message(**"ES\_ITEM\_TOPIC"**, **"UPDATE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));   */\*\* 发送消息，生成静态网页 \*/* **mqProducer**.send(**new** Message(**"PAGE\_ITEM\_TOPIC"**, **"CREATE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));  }**else** { *// 表示商品下架  // 发送消息，删除商品索引* **mqProducer**.send(**new** Message(**"ES\_ITEM\_TOPIC"**, **"DELETE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));  }  **return true**;  }**catch** (Exception ex){  ex.printStackTrace();  }  **return false**; } |

## 6.5消息消费者【商品详情】

### 6.5.1整合RocketMQ

#### **6.5.1.1配置依赖**

在pinyougou-item-web/pom.xml，引入依赖：

|  |
| --- |
| *<!-- rocketmq-client -->* <**dependency**>  <**groupId**>org.apache.rocketmq</**groupId**>  <**artifactId**>rocketmq-client</**artifactId**> </**dependency**> |

#### **6.5.1.2配置文件**

pinyougou-item-web/src/main/resources/目录下，添加

applicationContext-rocketmq.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd"**>   *<!-- 配置加载属性文件 -->* <**context:property-placeholder  location="classpath:application.properties"**/>   *<!--############# 配置消息消费者 #############-->  <!-- 配置默认的MQ消息消费者 -->* <**bean id="mqPushConsumer"  class="org.apache.rocketmq.client.consumer.DefaultMQPushConsumer"  init-method="start"  destroy-method="shutdown"**>  *<!-- 设置消息模式: (广播BROADCASTING | 集群CLUSTERING) -->* <**property name="messageModel" value="BROADCASTING"**/>  *<!-- 设置消费者组名 -->* <**property name="consumerGroup" value="PAGE\_ITEM\_CONSUMER"**/>  *<!-- 设置NameServer地址 -->* <**property name="namesrvAddr" value="${namesrvAddr}"**/>  *<!-- 设置消息监听器 -->* <**property name="messageListener" ref="messageListener"**/>  *<!-- 设置订阅信息 -->* <**property name="subscription"**>  <**map**>  *<!-- 指定主题和标签 -->* <**entry key="PAGE\_ITEM\_TOPIC" value="\*"**/>  </**map**>  </**property**>  </**bean**>  *<!-- 配置消息监听器 -->* <**bean id="messageListener"  class="com.pinyougou.item.listener.PageMessageListener"**/> </**beans**> |

pinyougou-item-web/src/main/resources/springmvc.xml：

|  |
| --- |
| *<!-- 配置开启组件扫描 -->* <**context:component-scan base-package="com.pinyougou.item"**/>  *......*  *<!-- 配置采用包扫描来引用服务，产生服务接口的代理对象 -->* <**dubbo:annotation package="com.pinyougou.item"**/> |

pinyougou-item-web/src/main/resources/application.properties：

|  |
| --- |
| *# 配置nameserver连接地址* **namesrvAddr**=**192.168.12.131:9876** *# 配置生成商品静态html页面存储路径* **page.dir**=**F:/pyg/item/** |

pinyougou-item-web/src/main/webapp/WEB-INF/web.xml：

|  |
| --- |
| *<!-- 配置Spring MVC前端控制器(核心控制器) -->* <**servlet**>  <**servlet-name**>pinyougou</**servlet-name**> <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:springmvc.xml,  classpath:applicationContext-rocketmq.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**> </**servlet**> |

### 6.5.2消费消息

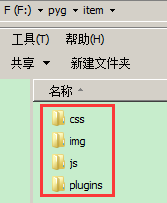
pinyougou-item-web/src/main/java/com/pinyougou/item/listener包下，创建PageMessageListener.java消息监听器：

|  |
| --- |
| */\*\* 消息监听器(生成静态html页面) \*/* **public class** PageMessageListener **implements** MessageListenerConcurrently {  @Value(**"${page.dir}"**)  **private** String **pageDir**;  @Autowired  **private** FreeMarkerConfigurer **freeMarkerConfigurer**;  @Reference(timeout = 10000)  **private** GoodsService **goodsService**;   @Override  **public** ConsumeConcurrentlyStatus  consumeMessage(List<MessageExt> list,  ConsumeConcurrentlyContext consumeConcurrentlyContext) {  **try** {  System.***out***.println(**"=====PageMessageListener======"**);  *// 获取消息封装对象* MessageExt messageExt = list.get(0);  *// 获取消息内容* String content = **new** String(messageExt.getBody(),**"UTF-8"**);  *// 把json字符串转化成List集合* List<Long> goodsIds = JSON.*parseArray*(content, Long.**class**);  System.***out***.println(**"goodsIds:"** + goodsIds);  System.***out***.println(**"tags:"** + messageExt.getTags());   *// 生成商品详情静态页面* **if** (**"CREATE"**.equals(messageExt.getTags())) {  *// 根据模板文件获取模板对象* Template template = **freeMarkerConfigurer** .getConfiguration().getTemplate(**"item.ftl"**);  *// 循环生成静态页面* **for** (Long goodsId : goodsIds) {  *// 获取数据模型* Map<String, Object> dataModel = **goodsService**.  getGoods(Long.*valueOf*(goodsId));  *// 创建输出流* OutputStreamWriter writer = **new** OutputStreamWriter(  **new** FileOutputStream(**pageDir** + goodsId + **".html"**), **"UTF-8"**);  *// 填充模板生成静态的html页面* template.process(dataModel, writer);  *// 关闭输出流* writer.close();  }  }  *// 删除商品详情静态页面* **if** (**"DELETE"**.equals(messageExt.getTags())) {  }  }**catch** (Exception ex){  *// 消费失败* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  } } |

## 6.6 Nginx静态资源服务器

### 6.6.1添加资源到html存储路径

复制pinyougou-item-web/src/main/webapp路径下的css、js、img、plugins文件夹到生成html页面所在路径。



### 6.6.2配置nginx.conf

在nginx的配置文件nginx.conf中添加如下配置信息：

|  |
| --- |
| server {  listen 80;  server\_name item.pinyougou.com;  proxy\_set\_header X-Forwarded-Host $host;  proxy\_set\_header X-Forwarded-Server $host;  proxy\_set\_header Host $host;  location / {  root F:/pyg/item;  proxy\_connect\_timeout 600;  proxy\_read\_timeout 600;  # 如果当前请求的文件路径不存在，  # 则跳转到tomcat服务器9105  if (!-e $request\_filename){  proxy\_pass http://127.0.0.1:9105;  }  }  } |

# 7.商品下架【删除商品详细页】

## 7.1需求分析

商家执行商品下架后，删除商品详细静态网页。

## 7.2消息生产者【商家后台】

### 7.2.1发送消息

pinyougou-shop-web/src/main/java/com/pinyougou/manager/controller/GoodsController.java：

|  |
| --- |
| */\*\* 商家商品上下架(修改可销售状态) \*/* @GetMapping(**"/updateMarketable"**) **public boolean** updateMarketable(Long[] ids, String status){  **try**{  *// 修改数据库表可销售状态* **goodsService**.updateStatus(**"is\_marketable"**, ids, status);  *// 判断商品上下架状态* **if** (**"1"**.equals(status)){ *// 表示商品上架  // 发送消息，生成商品索引* **mqProducer**.send(**new** Message(**"ES\_ITEM\_TOPIC"**, **"UPDATE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));   */\*\* 发送消息，生成静态网页 \*/* **mqProducer**.send(**new** Message(**"PAGE\_ITEM\_TOPIC"**, **"CREATE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));  }**else** { *// 表示商品下架  // 发送消息，删除商品索引* **mqProducer**.send(**new** Message(**"ES\_ITEM\_TOPIC"**, **"DELETE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));   */\*\* 发送消息，删除静态网页 \*/* **mqProducer**.send(**new** Message(**"PAGE\_ITEM\_TOPIC"**, **"DELETE"**,  JSON.*toJSONString*(ids).getBytes(**"UTF-8"**)));  }  **return true**;  }**catch** (Exception ex){  ex.printStackTrace();  }  **return false**; } |

## 7.3消息消费者【商品详情】

### 7.3.1消费消息

pinyougou-item-web\src\main\java\com\pinyougou\item\listener\PageMessageListener.java消息监听器

|  |
| --- |
| */\*\* 消息监听器(生成静态html页面) \*/* **public class** PageMessageListener **implements** MessageListenerConcurrently {  @Value(**"${page.dir}"**)  **private** String **pageDir**;  @Autowired  **private** FreeMarkerConfigurer **freeMarkerConfigurer**;  @Reference(timeout = 10000)  **private** GoodsService **goodsService**;   @Override  **public** ConsumeConcurrentlyStatus  consumeMessage(List<MessageExt> list,  ConsumeConcurrentlyContext consumeConcurrentlyContext) {  **try** {  System.***out***.println(**"=====PageMessageListener======"**);  *// 获取消息封装对象* MessageExt messageExt = list.get(0);  *// 获取消息内容* String content = **new** String(messageExt.getBody(),**"UTF-8"**);  *// 把json字符串转化成List集合* List<Long> goodsIds = JSON.*parseArray*(content, Long.**class**);  System.***out***.println(**"goodsIds:"** + goodsIds);  System.***out***.println(**"tags:"** + messageExt.getTags());   *// 生成商品详情静态页面* **if** (**"CREATE"**.equals(messageExt.getTags())) {  ......  }  *// 删除商品详情静态页面* **if** (**"DELETE"**.equals(messageExt.getTags())) {  **for** (Long goodsId : goodsIds){  File file = **new** File(**pageDir** + goodsId + **".html"**);  *// 判断文件是否存在* **if** (file.exists()){  file.delete();  }  }  }  }**catch** (Exception ex){  *// 消费失败* **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  *// 消费成功* **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  } } |