# **ActiveMq**

# 入门概述

## 1.1前言

## 1.2生活案例

系统之间接口耦合比较严重

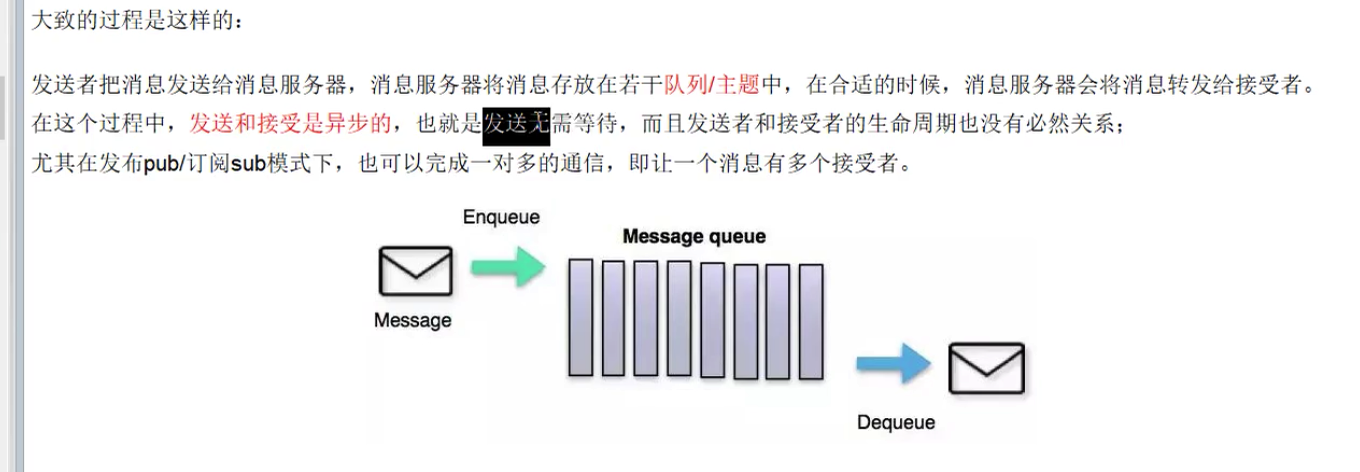
面对大流量并发时，容易被冲垮

等待同步存在性能问题

## 1.3是什么

定义：面向消息的中间件

对列/主题



## 1.4能干嘛

解耦，削峰，异步

mq解决的问题？

1. 解约耦合调用
2. 异步模型
3. 抵御洪峰流量，达到保护主业务，削峰

## 1.5 去哪下载

<http://activemq.apache.org/components/classic/download/>

## 1.6 怎么玩

1.最主要的功能：实现高可用、高性能、高伸缩、易用和安全的企业级面向消息服务的系统

2.异步消息的消费和处理

3.控制消息的消费顺序

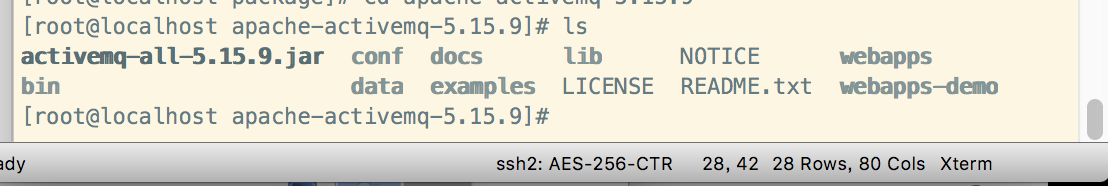
4.可以和spring/springboot整合简化编码

5.配置集群容错的MQ集群

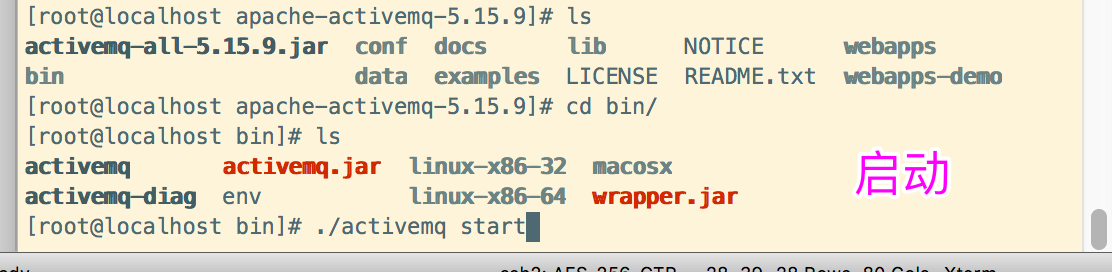
# ActiveMQ安装和控制台

首先要安装jdk

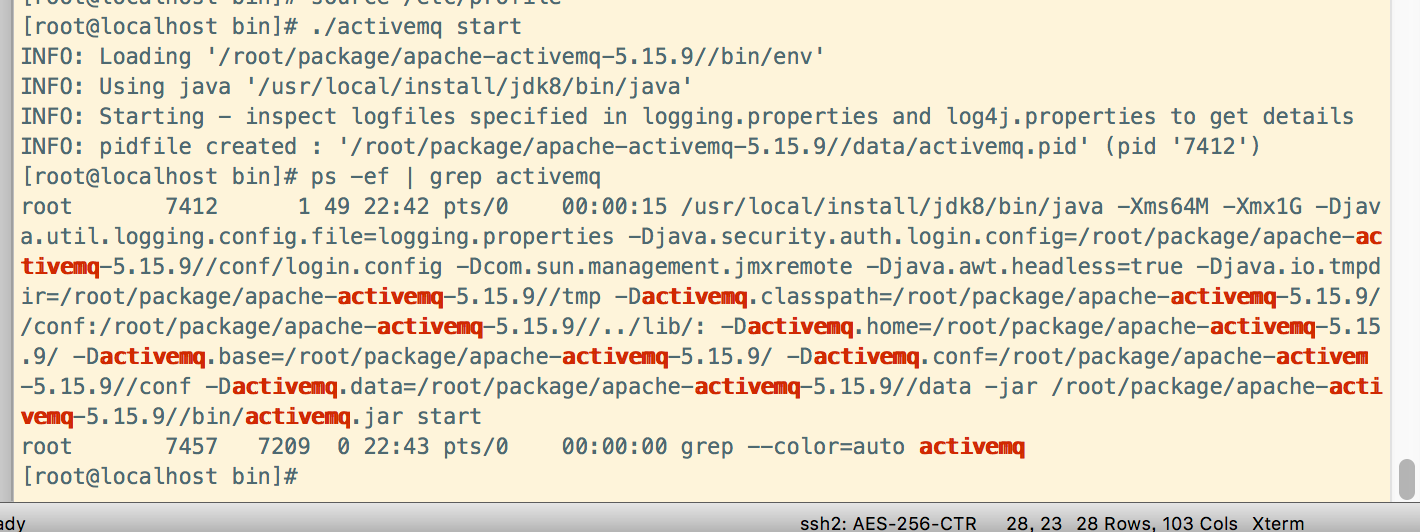
1. 下载apache-activemq-5.15.9-bin.tar.gz
2. Tar -vxf apache-activemq-5.15.9-bin.tar.gz



1. 启动



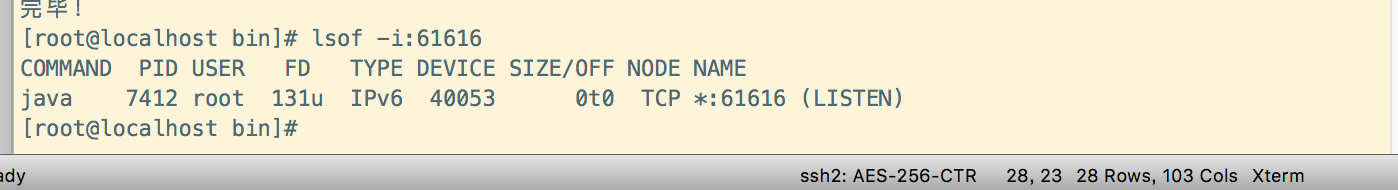
Ps -ef|grep activemq|grep -v grep



Netstat -anp|grep 61616

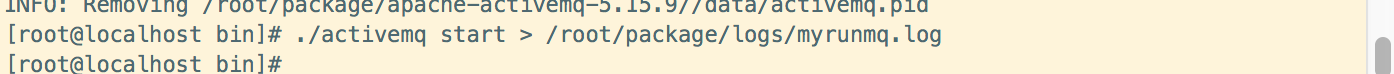


Lsof -i:61616

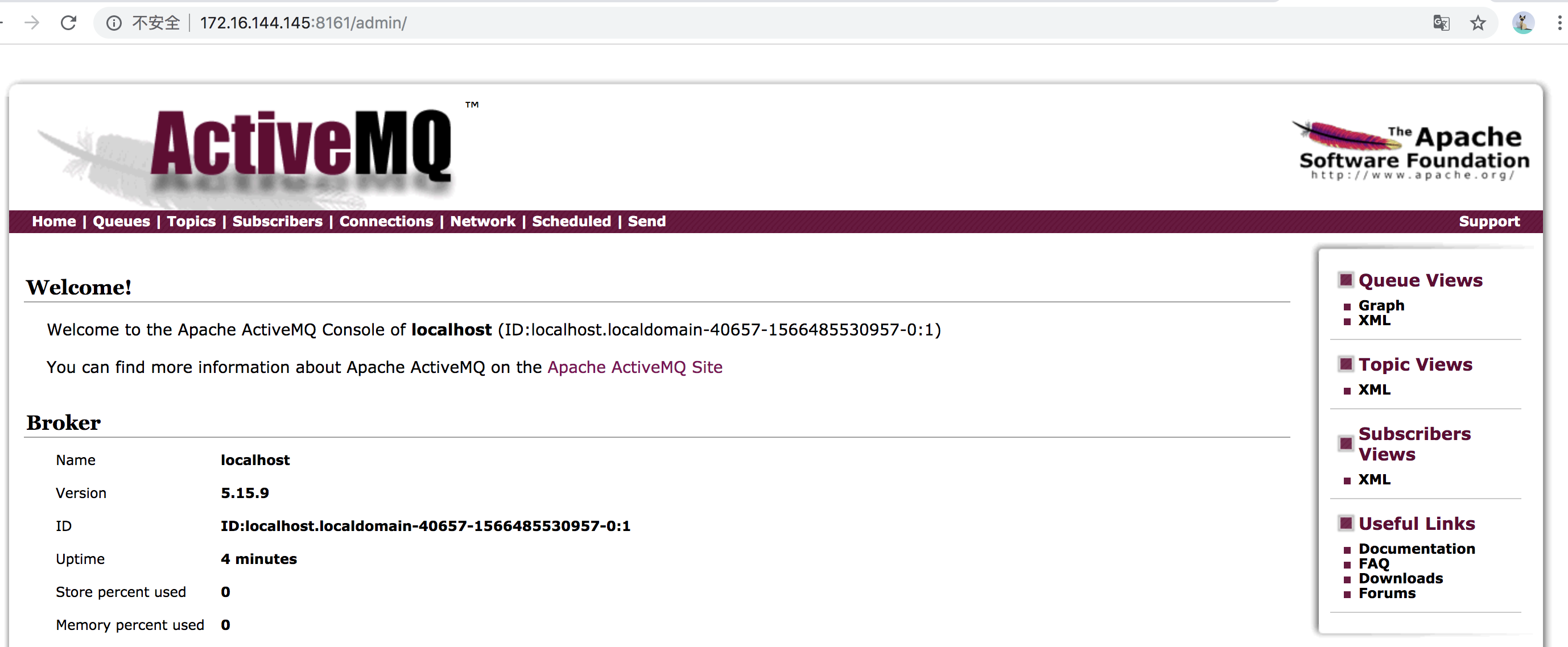


启动输出日志

./activemq start > /root/package/logs/myrunmq.log



<http://172.16.144.145:8161/admin/> admin/admin 访问客户端



# Java编码实现ActiveMQ通讯

## 3.1新建工程

|  |
| --- |
| 工程结构    Parent工程 Pom.xml  <?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">  <modelVersion>4.0.0</modelVersion>   <groupId>com.gardenia.activemq</groupId>  <artifactId>activemq-parent</artifactId>  <version>1.0-SNAPSHOT</version>   <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>1.7.26</version>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>1.2.3</version>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.18.2</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>   <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-all</artifactId>  <version>5.15.9</version>  </dependency>  <dependency>  <groupId>org.apache.xbean</groupId>  <artifactId>xbean-spring</artifactId>  <version>3.16</version>  </dependency>  </dependencies>  </dependencyManagement>  </project>  Demo01 的pom文件和parent目前一样的 |

## 3.2消息发送（生产消息）

|  |
| --- |
| 1. 生产消息   package com.gardenia.activemq.study;  import org.apache.activemq.ActiveMQConnectionFactory;  import javax.jms.\*;  public class JmsProduce {   public static final String *ACTIVEMQ\_URL* = "tcp://172.16.144.145:61616";  public static final String *QUEUE\_NAME* = "queue01";    public static void main(String[] args) throws JMSException {   // 1.创建连接工厂，按照给定的url地址，采用默认用户名和密码  ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  // 2.通过连接工厂，获得连接connection并启动  Connection connection = activeMQConnectionFactory.createConnection();  connection.start();  // 3.创建会话session  // 两个参数，第一个叫事务/第二个叫签收  Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  // 4.创建目的地（具体是对列是主题topic）  Queue queue = session.createQueue(*QUEUE\_NAME*);//  // 5.创建消息的生产者  MessageProducer messageProducer = session.createProducer(queue);  // 6.循环创建多条消息  for (int i = 0; i < 3; i++) {  // 7.创建消息  TextMessage textMessage = session.createTextMessage("msg----" + i);  // 8.通过messageProducer发送Mq  messageProducer.send(textMessage);   }   messageProducer.close();  session.close();  connection.close();     }  }  发送成功后，在后台可以看到生产的待消费的消息 |

## 3.3消息接收（消费消息）

|  |
| --- |
| 1. 接收消息   package com.gardenia.activemq.study;  import org.apache.activemq.ActiveMQConnectionFactory;  import javax.jms.\*;  public class JmsConsumer {   public static final String *ACTIVEMQ\_URL* = "tcp://172.16.144.145:61616";  public static final String *QUEUE\_NAME* = "queue01";    public static void main(String[] args) throws JMSException {   // 1.创建连接工厂，按照给定的url地址，采用默认用户名和密码  ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  // 2.通过连接工厂，获得连接connection并启动  Connection connection = activeMQConnectionFactory.createConnection();  connection.start();  // 3.创建会话session  // 两个参数，第一个叫事务/第二个叫签收  Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  // 4.创建目的地（具体是对列是主题topic）  Queue queue = session.createQueue(*QUEUE\_NAME*);//  // 5.创建消息的消费者  MessageConsumer consumer = session.createConsumer(queue);  // 6.循环创建多条消息  while (true) {  TextMessage textMessage = (TextMessage) consumer.receive();  if (null != textMessage) {  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*消息：" + textMessage.getText());  } else {  break;  }   }  consumer.close();  session.close();  connection.close();    }  }  2.     1. 接收消息方式二 监听器   package com.gardenia.activemq.study;  import org.apache.activemq.ActiveMQConnectionFactory;  import javax.jms.\*; import java.io.IOException;  public class JmsConsumer {   public static final String *ACTIVEMQ\_URL* = "tcp://172.16.144.145:61616";  public static final String *QUEUE\_NAME* = "queue01";    public static void main(String[] args) throws JMSException, IOException {   // 1.创建连接工厂，按照给定的url地址，采用默认用户名和密码  ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  // 2.通过连接工厂，获得连接connection并启动  Connection connection = activeMQConnectionFactory.createConnection();  connection.start();  // 3.创建会话session  // 两个参数，第一个叫事务/第二个叫签收  Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  // 4.创建目的地（具体是对列是主题topic）  Queue queue = session.createQueue(*QUEUE\_NAME*);//  // 5.创建消息的消费者  MessageConsumer consumer = session.createConsumer(queue);  // 6.循环创建多条消息 // while (true) { // // 同步阻塞方式(receive()) // //订阅者或接收者调用MessageConsumer的receive()方法来接受消息，receive方法在能够接受到消息职期间将一致阻塞 // TextMessage textMessage = (TextMessage) consumer.receive(); // if (null != textMessage) { // System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*消息：" + textMessage.getText()); // } else { // break; // } // // } // consumer.close(); // session.close(); // connection.close();   //通过监听的方式来消费消息，MessageConsumer messageConsumer = session.createConsumer(queue);   consumer.setMessageListener(new MessageListener() {  public void onMessage(Message message) {  if (null != message && message instanceof TextMessage) {  TextMessage textMessage = (TextMessage) message;  try {  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*消息：" + textMessage.getText());  } catch (JMSException e) {  e.printStackTrace();  }  }  }  });   // 这里要阻塞  System.*in*.read();  consumer.close();  session.close();  connection.close();   }  } |

## 3.4topic

1. 生产者将消息发布到topic中，每个消息可以有多个消费者，术语1：N的关系
2. 生产者和消费者之间有时间上的相关性。订阅某一个主题的消费者智能消费自它订阅之后发布的消息
3. 生产者生产时，topic不保存消息它是无状态的不略低，加入无人订阅就去生产，那就是一条废消息，所以，一般先启动消费者在启动生产者

例子

|  |
| --- |
| 1,消费者  package com.gardenia.activemq.study;  import org.apache.activemq.ActiveMQConnectionFactory;  import javax.jms.\*; import java.io.IOException;  public class JmsConsumerTopic {   public static final String *ACTIVEMQ\_URL* = "tcp://172.16.144.145:61616";  public static final String *TOPIC\_NAME* = "topic-xiaoming";    public static void main(String[] args) throws JMSException, IOException {  // 1.创建连接工厂，按照给定的url地址，采用默认用户名和密码  ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  // 2.通过连接工厂，获得连接connection并启动  Connection connection = activeMQConnectionFactory.createConnection();  connection.start();  // 3.创建会话session  // 两个参数，第一个叫事务/第二个叫签收  Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  // 4.创建目的地（具体是对列是主题topic）  Topic topic = session.createTopic(*TOPIC\_NAME*);//  // 5.创建消息的消费者  MessageConsumer consumer = session.createConsumer(topic);   //通过监听的方式来消费消息，MessageConsumer messageConsumer = session.createConsumer(queue);  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*consumer 2\*\*\*\*\*\*\*\*\*\*");  consumer.setMessageListener(new MessageListener() {  public void onMessage(Message message) {  if (null != message && message instanceof TextMessage) {  TextMessage textMessage = (TextMessage) message;  try {  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*topic消息：" + textMessage.getText());  } catch (JMSException e) {  e.printStackTrace();  }  }  }  });   // 这里要阻塞  System.*in*.read();  consumer.close();  session.close();  connection.close();    }  }   1. 生产者   package com.gardenia.activemq.study;  import org.apache.activemq.ActiveMQConnectionFactory;  import javax.jms.\*;  public class JmsProducerTopic {   public static final String *ACTIVEMQ\_URL* = "tcp://172.16.144.145:61616";  public static final String *TOPIC\_NAME* = "topic-xiaoming";    public static void main(String[] args) throws JMSException {   // 1.创建连接工厂，按照给定的url地址，采用默认用户名和密码  ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  // 2.通过连接工厂，获得连接connection并启动  Connection connection = activeMQConnectionFactory.createConnection();  connection.start();  // 3.创建会话session  // 两个参数，第一个叫事务/第二个叫签收  Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  // 4.创建目的地（具体是对列是主题topic）  Topic topic = session.createTopic(*TOPIC\_NAME*);// // 5.创建消息的生产者  MessageProducer messageProducer = session.createProducer(topic);  // 6.循环创建多条消息  for (int i = 0; i < 6; i++) {  // 7.创建消息  TextMessage textMessage = session.createTextMessage("msg----" + i);  // 8.通过messageProducer发送Mq  messageProducer.send(textMessage);   }   messageProducer.close();  session.close();  connection.close();   System.*out*.println("发布完成");    } }  启动连个消费者 |

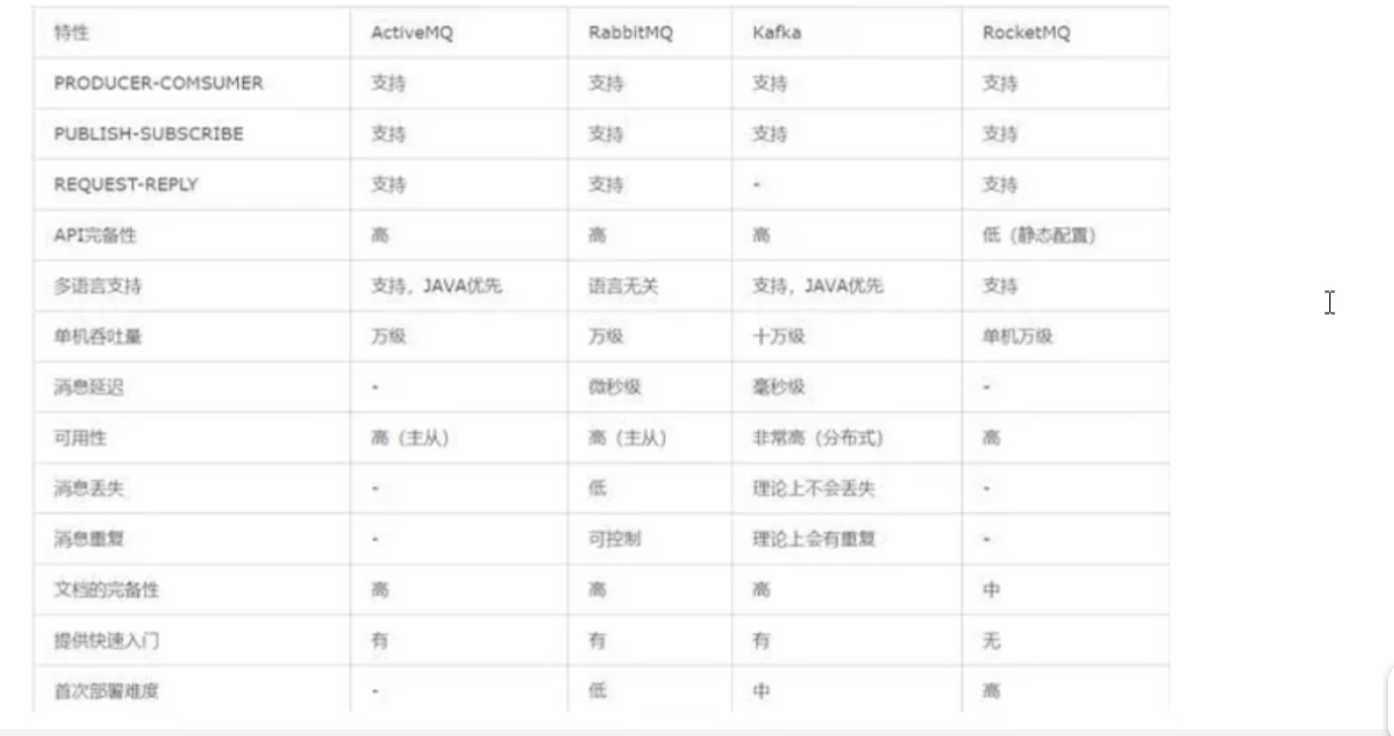


# JMS规范和落地产品

## 4.1jms是什么

两个应用程序之间进行一步通信的API，它为标准消息协议和消息服务提供了一组通用接口，包括创建，发送，读取消息等，用于支持JAVA引用程序开发。

## 4.2jms其他落地产品



## 4.3jms组成结构和特点

1. JMS provider

实现JMS接口和规范的消息中间件，也就是我们的MQ服务器

1. JMS producer

消息生产者，创建和发送JMS消息的客户端应用

1. JMS consumer

消息消费者，接收和发送JMS消息的客户端应用

1. JMS message

消息头：

JMSDestination:消息发送的目的地，主要是指Queue和Topic

JMSDeliveryMode:持久模式

JMSExpiration:消息过期时间

JMSPriority:消息优先级，0-4普通级别，5-9加急，默认是4级

JMSMessageID:唯一识别消息的标识

消息属性：

除了消息消息体和消息头之外的消息内容，kv键值对

消息体：

封装具体的消息数据：

TextMessage：普通字符串消息，包含一个String

MapMessage：一个Map类型的消息，key为string类型

BytesMessage：二进制数据消息，包含一个byte[]

StreamMessage：Java数据流消息，用标准的流操作来顺序的填充和读取

ObjectMessage：对象消息，包含一个可序列化的Java兑对象

5中消息体格式

发送和接收的消息体类型必须一致对应

## 4.4 jms的可靠性

### 4.4.1 PERSISENT：持久

参数设置说明：

非持久：messageProducer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT)

非持久化，当服务器宕机，消息体不存在

持久：messageProducer.setDeliveryMode(DeliveryMode.PERSISTENT)

非持久化，当服务器宕机，消息体存在

默认是持久化的

持久的Queue：

持久的Topic：

1. 一定要先运行一次消费者，等于向MQ注册，类似我订阅了了这个主题
2. 然后再运行生产者发送消息，
3. 无论消费者是否在线，都会接收到，不在线的话，下次连接的时候，会把没有收过得消息都接收下来

例子

|  |
| --- |
| 1. 生产者   package com.gardenia.activemq.study;  import org.apache.activemq.ActiveMQConnectionFactory;  import javax.jms.\*;  public class JmsProducerTopicPersist {   public static final String *ACTIVEMQ\_URL* = "tcp://172.16.144.145:61616";  public static final String *TOPIC\_NAME* = "topic-persist";    public static void main(String[] args) throws JMSException {   ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  Connection connection = activeMQConnectionFactory.createConnection();  connection.start();  Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  Topic topic = session.createTopic(*TOPIC\_NAME*);//  MessageProducer messageProducer = session.createProducer(topic);  for (int i = 0; i < 6; i++) {  TextMessage textMessage = session.createTextMessage("msg-persist----" + i);  messageProducer.send(textMessage);   }  messageProducer.close();  session.close();  connection.close();   System.*out*.println("发布完成");    } }   1. 消费者（持久化topic）   package com.gardenia.activemq.study;  import org.apache.activemq.ActiveMQConnectionFactory;  import javax.jms.\*; import java.io.IOException;  public class JmsConsumerTopicPersist {   public static final String *ACTIVEMQ\_URL* = "tcp://172.16.144.145:61616";  public static final String *TOPIC\_NAME* = "topic-persist";    public static void main(String[] args) throws JMSException, IOException {  // 1.创建连接工厂，按照给定的url地址，采用默认用户名和密码  System.*out*.println("---------z3消费者-------");  ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  // 2.通过连接工厂，获得连接connection并启动  Connection connection = activeMQConnectionFactory.createConnection();  connection.setClientID("Z3");    // 3.创建会话session  // 两个参数，第一个叫事务/第二个叫签收  Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  // 4.创建目的地（具体是对列是主题topic）  Topic topic = session.createTopic(*TOPIC\_NAME*);//  TopicSubscriber topicSubscriber = session.createDurableSubscriber(topic, "remark....");  connection.start();   Message message = topicSubscriber.receive();   while (null != message) {  TextMessage textMessage = (TextMessage) message;  System.*out*.println("\*\*\*\*\*\*\*\*\*收到持久化topic：" + textMessage.getText());  message = topicSubscriber.receive();  }   session.close();  connection.close();    }  }  持久化消费者，；离线 再次上下后，都能收到消息 |

### 4.4.2 事务：（片生产者）

producer提交事务：

False:只要执行send，就进入到对列中中

关闭事务，那第2个签收参数的设置需要有效

True：先执行send再执行commit，消息才被真正的提交到队列中

消息需要批量发送，需要缓冲区处理

commit提交事务：

False：消费完，对列中消息即消失

True：未提交事务，则对列中消息不会消失，会出现重复消息的问题

提交事务，就不会出现这种问题

### 4.4.3 Acknowledge：签收

非事务：

自动签收:Session.AUTO\_ACKNOWLEDGE

手动签收：Session.CLIENT\_ACKNOWLEDGE

客户端调用acknowledge方法手动签收

允许重复消息：Session.DUPS\_OK\_ACKNOWLEDGE

事务：

生产事务开启：只有commit后才能将全部消息表位已消费

消息生产者：

消息消费者：

签收和事务的关系：

## 4.5 点对点总结

对点点模型是基于对列的，生产者发消息到对列，消费者从对列接收消息，对列的存在使得消息的异步传输称为可能。和我们平时发送短信类似

1：如果在Session关闭时有部分消息已被收到但还没有被签收(acknowledged),那当消费者下次连接到相同的对列时，这些消息还会被再次接收。

2：对列可以长久地保存消息直到消费者收到消息。消费者不需要因为担心消息会丢失而时刻和对列保持激活状态，充分体现了异步传输模式的优势

## 4.6 发布订阅总结

JMS Pub/Sub模型定义了如何向一个内容节点发布和订阅消息，这些节点被称作topic

主题可以是被认为消息的传输中介，发布者(publisher)法布施消息到主题，订阅者（subscribe）从主题订阅消息。

主题是的消息订阅者和消息发布者保持先相互独立，不需要解除即可保证消息的传送。

一句话：先要订阅驻车才能接受到发布，只给订阅者发布消息。

客户端首先向MQ注册一个自己的身份ID识别号，当这个客户端处于离线时，生产者会为这个ID保存所有发送主题的消息，当客户再次连接到MQ时，会根据消费者的ID得到所有当自己处于离线时发送到主题的消息。

非持久订阅状态下，不能回复或重新跑送一个未签收的消息。

持久订阅才能恢复或重新派送一个未签收的消息。

# ActiveMQ的Broker

## 5.1是什么

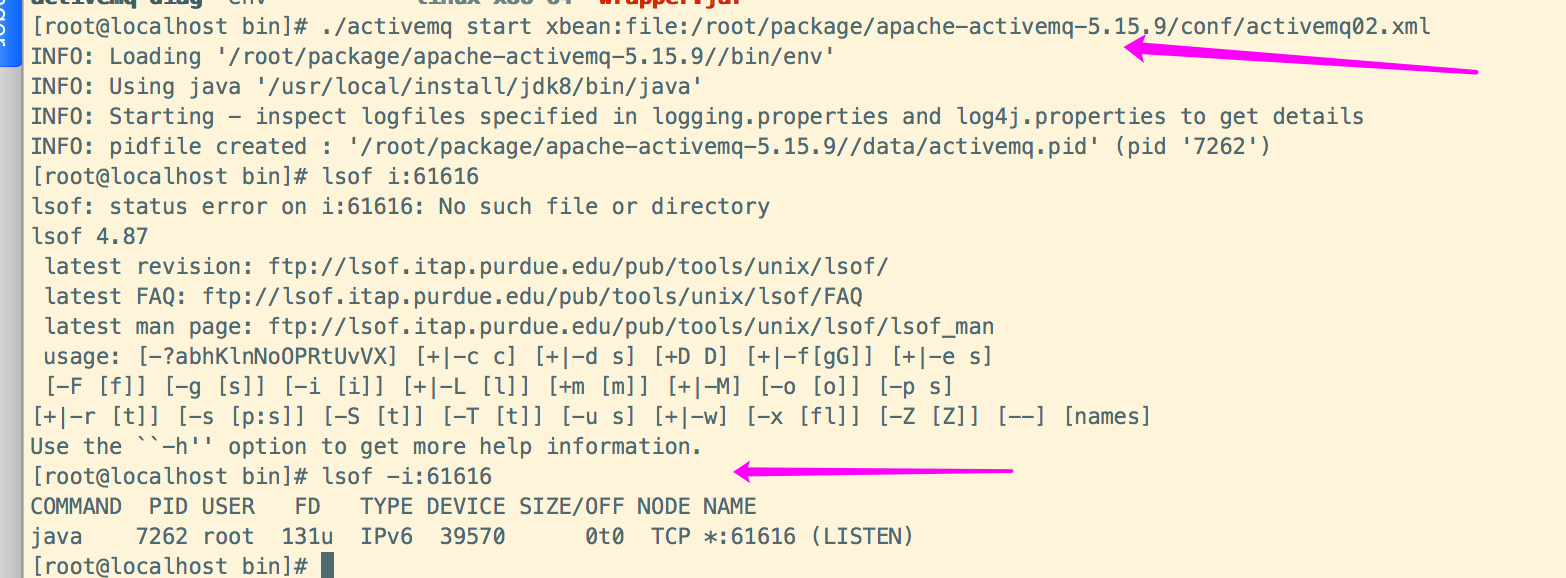
broker相当于一个ActiveMQ服务器的实例

说白了，Broker其实就是实现了用代码的形式启动ActiverMQ将MQ嵌入到Java代码中，以便随时启动，在用的时候再去启动这样能节省了资源，也保证了可靠性。

## 5.2配置文件（按配置文件启动）

新的启动命令

./activemq start xbean:file:/root/package/apache-activemq-5.15.9/conf/activemq02.xml



## 5.3嵌入式Broker

用ActiveMQ Broker 作为独立的消息服务器来构建JAVA应用

ActvieMQ也支持vm中通信基于嵌入式的broker，能够无缝的继承其他的java应用

# Spring整合ActvieMQ

## 6.1 Maven 修改，需要添加Spring支持JMS的包

|  |
| --- |
| Parent pom.xml  <?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">  <modelVersion>4.0.0</modelVersion>   <groupId>com.gardenia.activemq</groupId>  <artifactId>activemq-parent</artifactId>  <version>1.0-SNAPSHOT</version>   <properties>  <slf4j.version>1.7.26</slf4j.version>  <logback.version>1.2.3</logback.version>  <lomback.version>1.18.2</lomback.version>  <junit.version>4.12</junit.version>  <activemq.version>5.15.9</activemq.version>  <xbean.version>3.16</xbean.version>  <jackson.version>2.9.5</jackson.version>  <spring.version>4.3.23.RELEASE</spring.version>   </properties>   <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>${logback.version}</version>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>${lomback.version}</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  </dependency>   <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-all</artifactId>  <version>${activemq.version}</version>  </dependency>  <dependency>  <groupId>org.apache.xbean</groupId>  <artifactId>xbean-spring</artifactId>  <version>${xbean.version}</version>  </dependency>   <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>${jackson.version}</version>  </dependency>   <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-pool</artifactId>  <version>${activemq.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jms</artifactId>  <version>${spring.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>${spring.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>    </dependencies>  </dependencyManagement>  </project>  子工程demo01 pom.xml  <?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>activemq-parent</artifactId>  <groupId>com.gardenia.activemq</groupId>  <version>1.0-SNAPSHOT</version>  <relativePath>../activemq-parent/pom.xml</relativePath>  </parent>  <modelVersion>4.0.0</modelVersion>   <artifactId>activemq-demo01</artifactId>   <dependencies>   <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  </dependency>   <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-all</artifactId>  </dependency>  <dependency>  <groupId>org.apache.xbean</groupId>  <artifactId>xbean-spring</artifactId>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  </dependency>   <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-pool</artifactId>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jms</artifactId>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  </dependencies>  </project>  项目结构：    applicationContext.xml  <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:tx="http://www.springframework.org/schema/tx" xmlns:p="http://www.springframework.org/schema/p"  xmlns:util="http://www.springframework.org/schema/util" xmlns:jdbc="http://www.springframework.org/schema/jdbc"  xmlns:cache="http://www.springframework.org/schema/cache"  xsi:schemaLocation="  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd  http://www.springframework.org/schema/jdbc  http://www.springframework.org/schema/jdbc/spring-jdbc-3.1.xsd  http://www.springframework.org/schema/cache  http://www.springframework.org/schema/cache/spring-cache-3.1.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/util  http://www.springframework.org/schema/util/spring-util.xsd">    <context:component-scan base-package="com.gardenia.activemq.study"></context:component-scan>   <!-- 配置生产者-->  <bean id="jmsFactory" class="org.apache.activemq.pool.PooledConnectionFactory" destroy-method="stop">  <property name="connectionFactory">  <bean class="org.apache.activemq.ActiveMQConnectionFactory">  <property name="brokerURL" value="tcp://172.16.144.145:61616"></property>  </bean>  </property>  <property name="maxConnections" value="100"></property>  </bean>   <!-- 对列的目的地，点对点-->  <bean id="destinationQueue" class="org.apache.activemq.command.ActiveMQQueue">  <constructor-arg index="0" value="spring-active-queue"/>  </bean>   <!-- 这个是个主题-->  <bean id="destinationTopic" class="org.apache.activemq.command.ActiveMQTopic">  <constructor-arg index="0" value="spring-active-topic"/>  </bean>   <!-- Spring提供的JMS工具类，它可以进行消息发送-->  <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">  <property name="connectionFactory" ref="jmsFactory"></property>  <property name="defaultDestination" ref="destinationTopic"></property>  <property name="messageConverter">  <bean class="org.springframework.jms.support.converter.SimpleMessageConverter"></bean>  </property>  </bean>   </beans>  Queue SpringMQ\_Producer  package com.gardenia.activemq.study.spring;   import org.apache.xbean.spring.context.ClassPathXmlApplicationContext; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.ApplicationContext; import org.springframework.jms.core.JmsTemplate; import org.springframework.jms.core.MessageCreator; import org.springframework.stereotype.Service;  import javax.jms.JMSException; import javax.jms.Message; import javax.jms.Session; import javax.jms.TextMessage;  @Service public class SpringMQ\_Producer {   @Autowired  private JmsTemplate jmsTemplate;   public static void main(String[] args) {   ApplicationContext app = new ClassPathXmlApplicationContext("applicationContext.xml");   SpringMQ\_Producer produce = (SpringMQ\_Producer) app.getBean("springMQ\_Producer");  // produce.jmsTemplate.send(new MessageCreator() { // public Message createMessage(Session session) throws JMSException { // // TextMessage textMessage = session.createTextMessage("\*\*\*\*\*\*spring和ActiveMQ的整合case\*\*\*\*\*"); // return textMessage; // } // });  produce.jmsTemplate.send(session -> session.createTextMessage("\*\*\*\*\*\*spring和ActiveMQ的整合case\*\*\*\*\*"));   System.*out*.println("\*\*\*\*\*\* send message over\*\*\*\*\*");  }  }  Queue SpringMQ\_Consumer  package com.gardenia.activemq.study.spring;  import org.apache.xbean.spring.context.ClassPathXmlApplicationContext; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.ApplicationContext; import org.springframework.jms.core.JmsTemplate; import org.springframework.stereotype.Service;  @Service public class SpringMQ\_Consumer {    @Autowired  private JmsTemplate jmsTemplate;   public static void main(String[] args) {   ApplicationContext app = new ClassPathXmlApplicationContext("applicationContext.xml");   SpringMQ\_Consumer consumer = (SpringMQ\_Consumer) app.getBean("springMQ\_Consumer");   String value = (String) consumer.jmsTemplate.receiveAndConvert();   System.*out*.println("\*\*\*\*\*\* consumer message " + value + "\*\*\*\*\*");  } }  Topic SpringTopicMQ\_Producer  package com.gardenia.activemq.study.spring;   import org.apache.xbean.spring.context.ClassPathXmlApplicationContext; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.ApplicationContext; import org.springframework.jms.core.JmsTemplate; import org.springframework.stereotype.Service;  @Service public class SpringTopicMQ\_Producer {   @Autowired  private JmsTemplate jmsTemplate;   public static void main(String[] args) {   ApplicationContext app = new ClassPathXmlApplicationContext("applicationContext.xml");   SpringTopicMQ\_Producer produce = (SpringTopicMQ\_Producer) app.getBean("springTopicMQ\_Producer");  // produce.jmsTemplate.send(new MessageCreator() { // public Message createMessage(Session session) throws JMSException { // // TextMessage textMessage = session.createTextMessage("\*\*\*\*\*\*spring和ActiveMQ的整合case\*\*\*\*\*"); // return textMessage; // } // });  produce.jmsTemplate.send(session -> session.createTextMessage("\*\*\*\*\*\*spring和ActiveMQ的整合case topic message \*\*\*\*\*"));   System.*out*.println("\*\*\*\*\*\* send message over\*\*\*\*\*");  }  }  Topic SpringTopicMQ\_Consumer  package com.gardenia.activemq.study.spring;  import org.apache.xbean.spring.context.ClassPathXmlApplicationContext; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.ApplicationContext; import org.springframework.jms.core.JmsTemplate; import org.springframework.stereotype.Service;  @Service public class SpringTopicMQ\_Consumer {    @Autowired  private JmsTemplate jmsTemplate;   public static void main(String[] args) {   ApplicationContext app = new ClassPathXmlApplicationContext("applicationContext.xml");   SpringTopicMQ\_Consumer consumer = (SpringTopicMQ\_Consumer) app.getBean("springTopicMQ\_Consumer");   String value = (String) consumer.jmsTemplate.receiveAndConvert();   System.*out*.println("\*\*\*\*\*\* consumer message " + value + "\*\*\*\*\*");  } } |

## 6.2 Spring配置文件

## 6.3 对列

## 6.4 主题

## 6.5 在Spring里面实现消费者不启动，直接通过配置监听完成

在applicationContext.xml中加监听器

|  |
| --- |
| applicationContext.xml 增加监听器配置  <!-- 配置监听程序--> <bean id="jmsContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer">  <property name="connectionFactory" ref="jmsFactory"></property>  <property name="destination" ref="destinationTopic"></property>  <property name="messageListener" ref="myMessageListener"></property> </bean>  增加监听器的实现类  package com.gardenia.activemq.study.spring;  import org.springframework.stereotype.Component;  import javax.jms.JMSException; import javax.jms.Message; import javax.jms.MessageListener; import javax.jms.TextMessage;  @Component public class MyMessageListener implements MessageListener {   @Override  public void onMessage(Message message) {  if (null != message && message instanceof TextMessage) {  TextMessage textMessage = (TextMessage) message;  try {  System.*out*.println(textMessage.getText());  } catch (JMSException e) {  e.printStackTrace();  }  }  }  }  然后 直接启动SpringTopicMQ\_Producer,不用启动消费者，控制台会自动输出生产的消息 |

# Springboot整合ActvieMQ