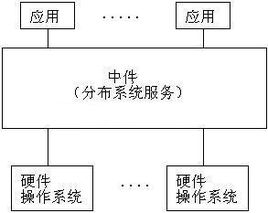
# 中间件

Middleware is computer software that connects software components and applications. The software consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. This technology evolved to provide for interoperability in support of the move to coherent distributed architectures, which are used most often to support and simplify complex, distributed applications. It includes web servers, transaction monitors, and messaging-and-queueing software.

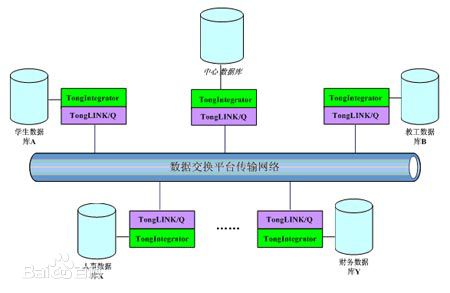
中间件是一类连接软件组件和应用的计算机软件，它包括一组服务。以便于运行在一台或多台机器上的多个软件通过网络进行交互。该技术所提供的互操作性，推动了一致[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F" \t "_blank)体系架构的演进，该架构通常用于支持并简化那些复杂的[分布式应用程序](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)，它包括web服务器、[事务](https://baike.baidu.com/item/%E4%BA%8B%E5%8A%A1" \t "_blank)[监控器](https://baike.baidu.com/item/%E7%9B%91%E6%8E%A7%E5%99%A8)和[消息队列](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E9%98%9F%E5%88%97)[软件](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6)。

**通常一个分布式系统架构需要的中间件：消息中间件，服务中间件，数据库中间件**



# 消息中间件（MQ）

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)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**。



常用的消息中间件介绍：

1，RabbitMQ

**RabbitMQ**是一个在[AMQP](http://baike.baidu.com/item/AMQP" \t "_blank)基础上完成的，可复用的企业消息系统。他遵循**Mozilla Public License**[开源协议](http://baike.baidu.com/item/%E5%BC%80%E6%BA%90%E5%8D%8F%E8%AE%AE)。[MQ](http://baike.baidu.com/item/MQ)是**消费-生产者**模型的一个典型的代表，一端往[消息队列](http://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E9%98%9F%E5%88%97)中不断写入消息，而另一端则可以读取或者订阅队列中的消息。MQ和[JMS](http://baike.baidu.com/item/JMS" \t "_blank)类似，但不同的是JMS是SUN JAVA[消息中间件](http://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E4%B8%AD%E9%97%B4%E4%BB%B6)服务的一个标准和API定义，而MQ则是遵循了AMQP协议的具体实现和产品。

2，**RocketMQ**

**阿里巴巴，唯一一个提供商业服务的MQ**

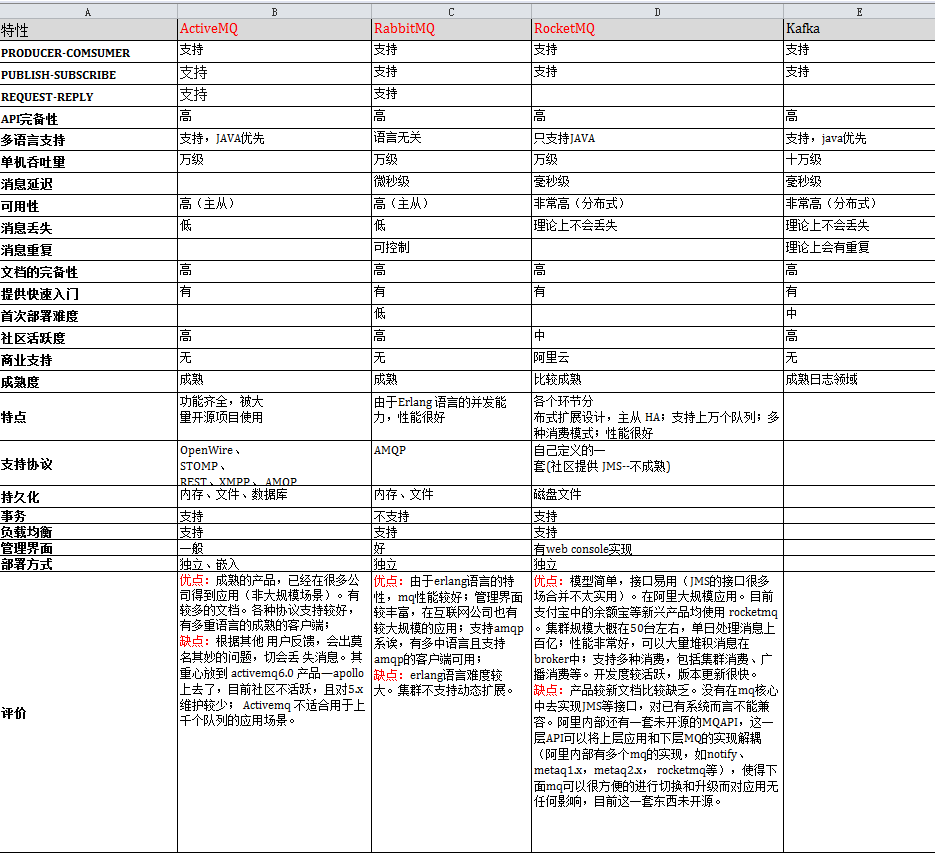
**3，ActiveMQ：**

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，尽管JMS规范出台已经是很久的事情了，但是***JMS***在当今的J2EE应用中间仍然扮演着特殊的地位。

**4，**Kafka

Kafka是一种高吞吐量的[分布式](http://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F" \t "_blank)发布订阅消息系统，它可以处理消费者规模的网站中的所有动作流数据。 这种动作（网页浏览，搜索和其他用户的行动）是在现代网络上的许多社会功能的一个关键因素。 这些数据通常是由于吞吐量的要求而通过处理日志和日志聚合来解决。 对于像[Hadoop](http://baike.baidu.com/item/Hadoop" \t "_blank)的一样的[日志](http://baike.baidu.com/item/%E6%97%A5%E5%BF%97/2769135" \t "_blank)数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。Kafka的目的是通过[Hadoop](http://baike.baidu.com/item/Hadoop" \t "_blank)的并行加载机制来统一线上和离线的消息处理，也是为了通过[集群](http://baike.baidu.com/item/%E9%9B%86%E7%BE%A4" \t "_blank)来提供实时的消费。

**对比：**



我们学习**activemq**，因为它是用java实现的，所以要安装java环境！！

# 准备虚拟机

ip 192.168.19.140

# 安装jdk 1.8

安装上传软件：

yum install lrzsz -y

上传文件到虚拟机

rz



使用rpm命令安装程序

rpm -ivh jre-7u79-linux-i586.rpm

默认安装位置：/usr/java/jre1.7.0\_79

|  |
| --- |
| [root@activemq ~]# ll /usr/java/  total 4  lrwxrwxrwx. 1 root root 16 Jul 16 17:01 default -> /usr/java/latest  drwxr-xr-x. 6 root root 4096 Jul 16 17:01 jre1.7.0\_79  lrwxrwxrwx. 1 root root 21 Jul 16 17:01 latest -> /usr/java/jre1.7.0\_79 |

/usr/bin/java java -> /usr/java/default/bin/java

一共使用了三层连接

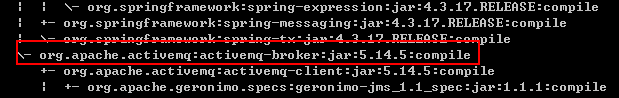
测试：

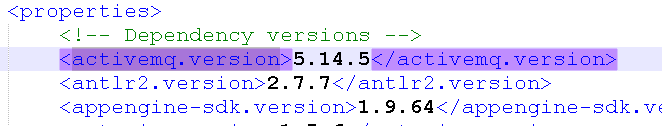
**java -version**

# 安装activemq

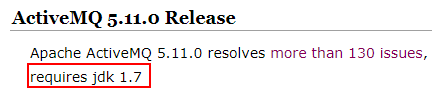
所有版本下载页面：<http://activemq.apache.org/download-archives.html>

Spring-boot-1.5.15.RELEASE依赖的是activemq-5.14.5





我们使用：**5.14.5 http://activemq.apache.org/activemq-5145-release.html**

****

安装下载软件：yum install wget -y

下载：

wget <http://archive.apache.org/dist/activemq/5.11.4/apache-activemq-5.11.4-bin.tar.gz>

或者上传

rz….

解压缩

**tar zxf apache-activemq-5.11.4-bin.tar.gz**

移动到默认安装位置

mv apache-activemq-5.11.4 /usr/local/apache-activemq-5.11.4

消息通讯端口：**61616**

管理控制台默认端口号：**8161**，在**jetty**的配置文件中，路径：解压目录**/conf/jetty.xml**

<bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">

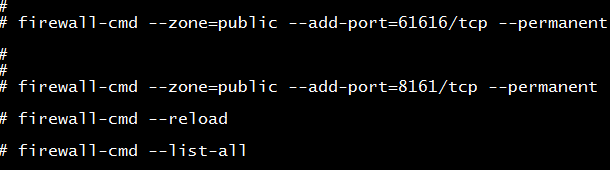
<!-- the default port number for the web console -->

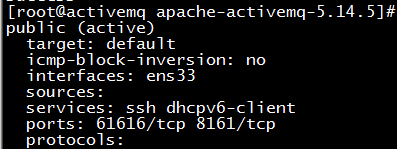
<property name="host" value="0.0.0.0"/>

<property name="port" value="8161"/>

</bean>

修改防火墙





cd bin/ （路径的相对位置，现在是在父目录下）

启动服务

**./activemq start**

访问管理后台

浏览器输入：[http://192.168.136.50:8161/](http://192.168.221.140:8161/)



管理**broker**

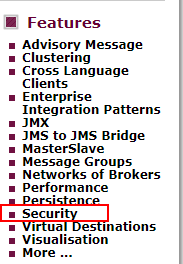
需要登录



默认管理员密码 **admin/admin**

安全配置，默认任何人都可以随意发消息，参考官网：<http://activemq.apache.org/security.html>

在生产环境中，我们通常需要认证之后才能发送消息，需要添加用户验证！！



**vi conf/activemq.xml**

|  |
| --- |
| <broker>     <plugins>     <simpleAuthenticationPlugin>      <users>          <authenticationUser username="root" password="111111" groups="users,admins"/>    </users>  </simpleAuthenticationPlugin>      </plugins>  </broker> |

重启activemq

./activemq restart

控制台用户名，密码设置文件：

Vi conf/jetty.xml

|  |
| --- |
| <bean id="securityConstraint" class="org.eclipse.jetty.util.security.Constraint">  <property name="name" value="BASIC" />  <property name="roles" value="user,admin" />  <!-- set authenticate=false to disable login -->  <property name="authenticate" value="true" />  </bean> |

**vi conf/jetty-realm.properties**

|  |
| --- |
| # Defines users that can access the web (console, demo, etc.)  # username: password [,rolename ...]  admin: admin, admin  user: user, user |

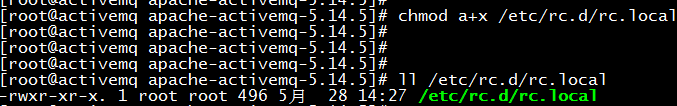
开机自启

**Vi /etc/rc.local**

|  |
| --- |
| /usr/local/apache-activemq-5.11.4/bin/activemq start |



修改文件权限：



**关于无法正常关闭activemq的问题：**

参考：https://stackoverflow.com/questions/11019313/activemq-exception-when-stopping

解决方法，修改 **conf/activemq.xml**文件的该节点配置，如下图



# 理论

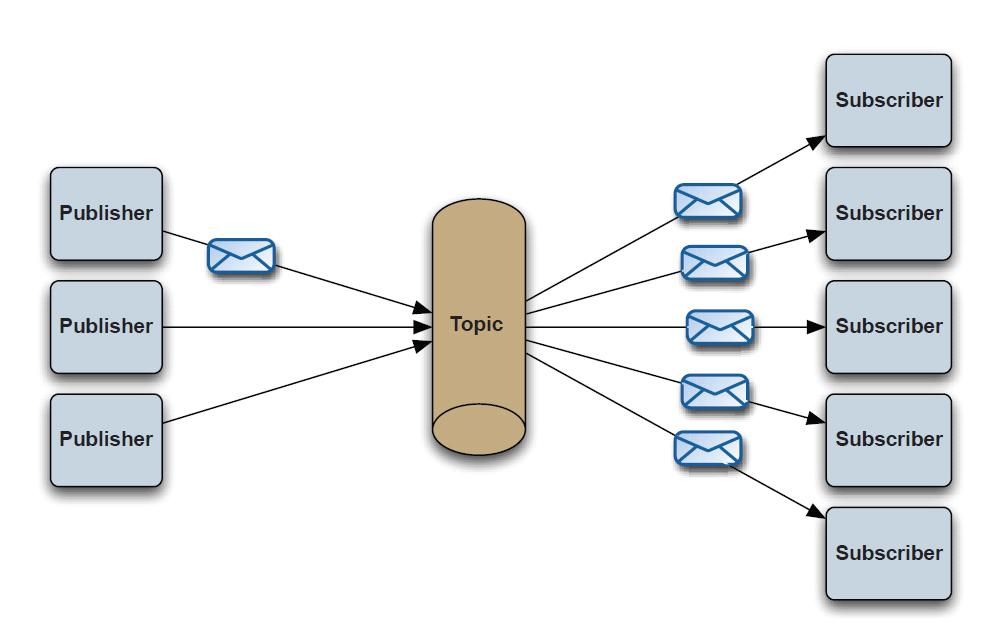
JMS即[Java消息服务](https://baike.baidu.com/item/Java%E6%B6%88%E6%81%AF%E6%9C%8D%E5%8A%A1" \t "_blank)（Java Message Service）应用程序接口，是一个[Java平台](https://baike.baidu.com/item/Java%E5%B9%B3%E5%8F%B0" \t "_blank)中关于面向[消息中间件](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E4%B8%AD%E9%97%B4%E4%BB%B6/5899771)（MOM）的[API](https://baike.baidu.com/item/API/10154" \t "_blank)，用于在两个应用程序之间，或[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/4905336" \t "_blank)中发送消息，进行异步通信。Java消息服务是一个与具体平台无关的API，绝大多数MOM提供商都对JMS提供支持。

**MQ软件结构设计：**

是基于生产者（producer）和消费者（consumer）模式。

**发布，订阅(publish subscribe)**

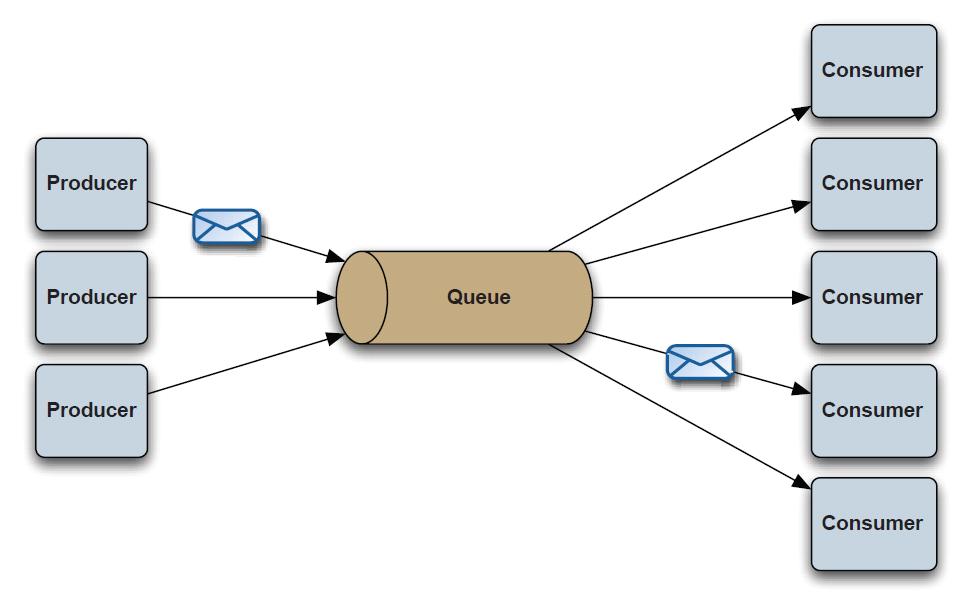
发布订阅模式有点类似于我们日常生活中订阅报纸。每年到年尾的时候，邮局就会发一本报纸集合让我们来选择订阅哪一个。在这个表里头列了所有出版发行的报纸，那么对于我们每一个订阅者来说，我们可以选择一份或者多份报纸。比如北京日报、潇湘晨报等。那么这些个我们订阅的报纸，就相当于发布订阅模式里的**topic**。有很多个人订阅报纸，也有人可能和我订阅了相同的报纸。那么，在这里，相当于我们在同一个topic里注册了。对于一份报纸发行方来说，它和所有的订阅者就构成了一个**1对多**的关系。这种关系如下图所示：



**点对点（队列）**：

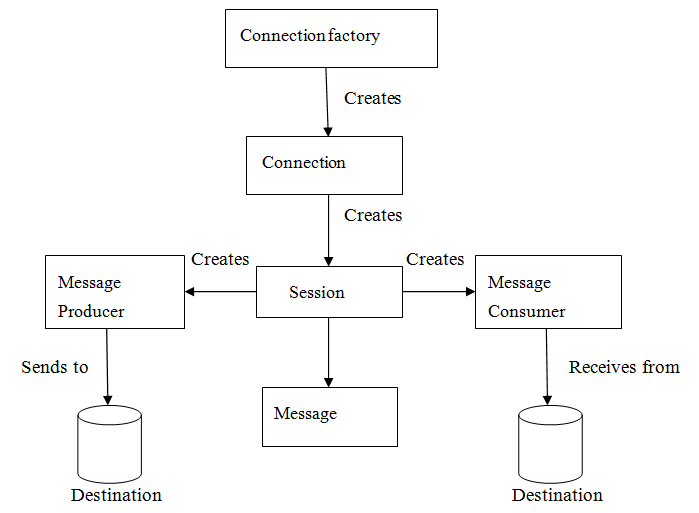
 p2p的过程则理解起来更加简单。它好比是两个人打电话，这两个人是独享这一条通信链路的。一方发送消息，另外一方接收，就这么简单。在实际应用中因为有多个用户对使用p2p的链路，它的通信场景如下图所示：

 在p2p的场景里，相互通信的双方是通过一个类似于队列的方式来进行交流。和前面pub-sub的区别在于一个topic有一个发送者和多个接收者，而在p2p里一个queue只有一个发送者和一个接收者。



## JMS编程模型

1. 管理对象（Administered objects）-连接工厂（Connection Factories）和目的地（Destination）
2. 连接对象（Connections）
3. 会话（Sessions）
4. 消息生产者（Message Producers）
5. 消息消费者（Message Consumers）
6. 消息监听者（Message Listeners）



　　　　（1）、Connection Factories

　　　　　　　　创建Connection对象的工厂，针对两种不同的jms消息模型，分别有QueueConnectionFactory和TopicConnectionFactory两种。可以通过JNDI来查找ConnectionFactory对象。客户端使用一个连接工厂对象连接到JMS服务提供者，它创建了JMS服务提供者和客户端之间的连接。JMS客户端（如发送者或接受者）会在JNDI名字空间中搜索并获取该连接。使用该连接，客户端能够与目的地通讯，往队列或话题发送/接收消息。

QueueConnectionFactory queueConnFactory = (QueueConnectionFactory) initialCtx.lookup ("primaryQCF");

Queue purchaseQueue = (Queue) initialCtx.lookup ("Purchase\_Queue");

Queue returnQueue = (Queue) initialCtx.lookup ("Return\_Queue");

　　　　（2）、Destination

 　　　　　　　　目的地指明消息被发送的目的地以及客户端接收消息的来源。JMS使用两种目的地，队列和话题。如下代码指定了一个队列和话题：

　　　　创建一个队列Session：

QueueSession ses = con.createQueueSession (false, Session.AUTO\_ACKNOWLEDGE); //get the Queue object

Queue t = (Queue) ctx.lookup ("myQueue"); //create QueueReceiver

QueueReceiver receiver = ses.createReceiver(t);

　　　　（3）、Connection

　　　　　　Connection表示在客户端和JMS系统之间建立的链接（对TCP/IP socket的包装）。Connection可以产生一个或多个Session。跟ConnectionFactory一样，Connection也有两种类型：QueueConnection和TopicConnection。

　　　　　　连接对象封装了与JMS提供者之间的虚拟连接，如果我们有一个ConnectionFactory对象，可以使用它来创建一个连接。

Connection connection = connectionFactory.createConnection();

　　　　（4）、Session

　　　　　　Session 是我们对消息进行操作的接口，可以通过session创建生产者、消费者、消息等。Session 提供了事务的功能，如果需要使用session发送/接收多个消息时，可以将这些发送/接收动作放到一个事务中。

　　　　　　我们可以在连接创建完成之后创建session：

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

　　　　　　这里面提供了参数两个参数，第一个参数是是否支持事务，第二个是事务的类型

　　　　（5）、Producter

　　　　　　消息生产者由Session创建，用于往目的地发送消息。生产者实现MessageProducer接口，我们可以为目的地、队列或话题创建生产者；

MessageProducer producer = session.createProducer(dest);

MessageProducer producer = session.createProducer(queue);

MessageProducer producer = session.createProducer(topic);

　　　　（6）、Consumer

　　　　　　消息消费者由Session创建，用于接收被发送到Destination的消息。

MessageConsumer consumer = session.createConsumer(dest);

MessageConsumer consumer = session.createConsumer(queue);

MessageConsumer consumer = session.createConsumer(topic);

　　　　（7）、MessageListener

　　　　　　消息监听器。如果注册了消息监听器，一旦消息到达，将自动调用监听器的onMessage方法。EJB中的MDB（Message-Driven Bean）就是一种MessageListener。

# 使用示例

企业应用：例如发邮件，发短信，**非强一致性事务的管理，多个分布式软件的协同工作**的实现手段。

等等。。。

## Boot-mq-starter

<https://docs.spring.io/spring-boot/docs/1.5.15.RELEASE/reference/htmlsingle/>

Spring Boot can also configure a ConnectionFactory when it detects that ActiveMQ is available on the classpath. If the broker is present, an embedded broker is started and configured automatically (as long as no broker URL is specified through configuration).

|  |
| --- |
| [Note] |
| If you are using spring-boot-starter-activemq the necessary dependencies to connect or embed an ActiveMQ instance are provided, as well as the Spring infrastructure to integrate with JMS. |

ActiveMQ configuration is controlled by external configuration properties in spring.activemq.\*. For example, you might declare the following section inapplication.properties:

spring.activemq.broker-url=tcp://192.168.1.210:9876

spring.activemq.user=admin

spring.activemq.password=secret

You can also pool JMS resources by adding a dependency to org.apache.activemq:activemq-pool and configure the PooledConnectionFactory accordingly:

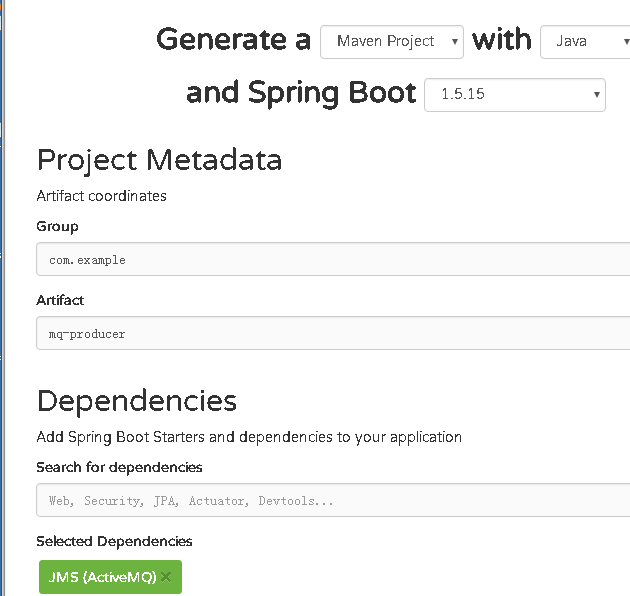
spring.activemq.pool.enabled=true

spring.activemq.pool.max-connections=50

|  |
| --- |
| [Tip] |
| See [ActiveMQProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.13.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQProperties.java" \t "_top) for more of the supported options. You can also register an arbitrary number of beans implementing ActiveMQConnectionFactoryCustomizer for more advanced customizations. |

By default, ActiveMQ creates a destination if it does not exist yet, so destinations are resolved against their provided names.

## 生产者工程



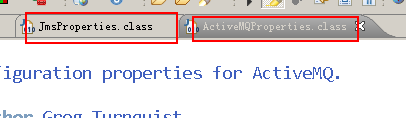
配置文件pom.xml：

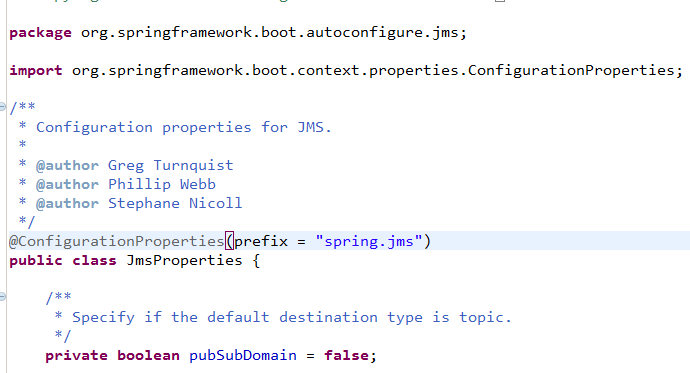
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>mq-producer</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>mq-producer</name>  <description>Demo project for Spring Boot</description>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.15.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-activemq</artifactId>  </dependency>  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-pool</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

Application.yml

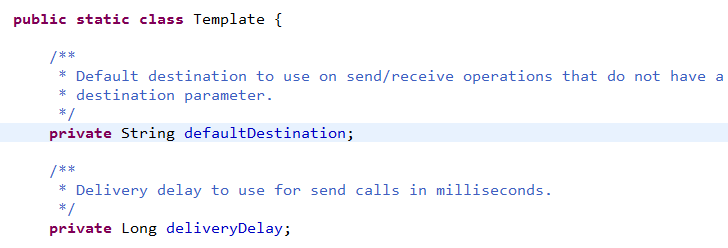
|  |
| --- |
| spring:  activemq:  brokerUrl: tcp://192.168.192.150:61616  user: root  password: 111111  pool:  enabled: **true**  max-connections: 3  jms:  template:  defaultDestination: mq-queue |

具体配置参考配置类！！！





内部类



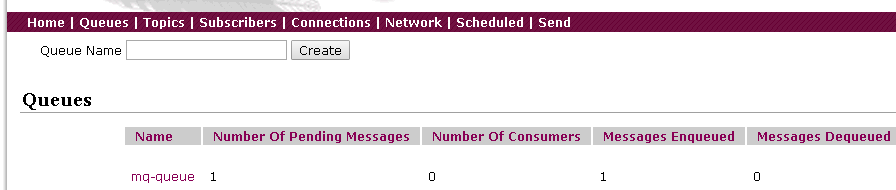
向消息服务器发送一条消息

启动类

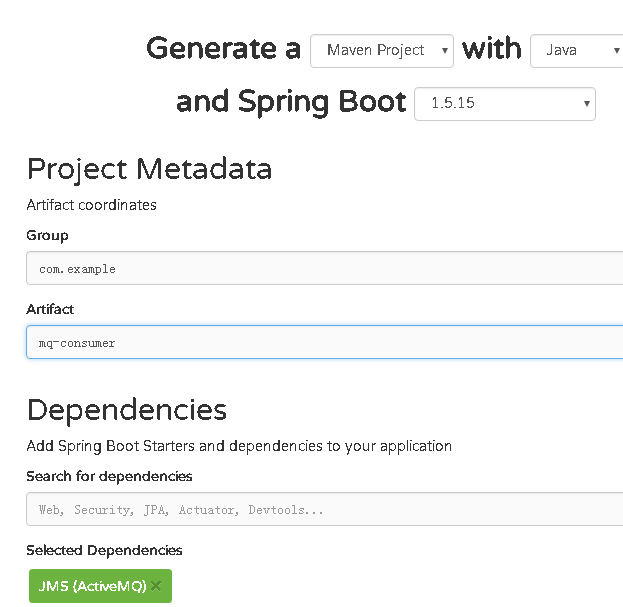
|  |
| --- |
| package com.cjl.mq.product;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class ProductMain {  public static void main(String[] args) {  SpringApplication.run(ProductMain.class, args);  }  } |

mqproducer.java

|  |
| --- |
| package com.example.mqproducer;  import javax.jms.JMSException;  import javax.jms.Message;  import javax.jms.Session;  import javax.jms.TextMessage;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jms.core.JmsTemplate;  import org.springframework.jms.core.MessageCreator;  import org.springframework.stereotype.Service;  @Service  public class MqProducer {  @Autowired  private JmsTemplate jmsTemplate;    public void send() {  jmsTemplate.send(new MessageCreator() {  //发送一条简单的消息  @Override  public Message createMessage(Session session) throws JMSException {  TextMessage textMessage = session.createTextMessage();  textMessage.setText("我是一条消息！！");  return textMessage;  }  });  }  } |



## 消费者

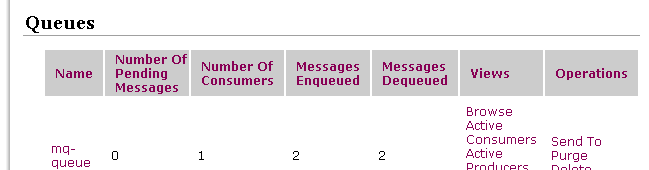


Pom及配置文件同生产者！！

MqConsumer.java

|  |
| --- |
| package com.example.mqconsumer;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.jms.annotation.JmsListener;  import org.springframework.stereotype.Service;  @Service  public class MqConsumer {  private static final Logger log = LoggerFactory.getLogger(MqConsumer.class);  @JmsListener(destination="mq-queue")  public void receive(String text) {  log.info("接收到一条消息："+text);  }  } |

启动项目测试



# 电子邮件协议

常用的电子邮件协议有SMTP、POP3、IMAP4，它们都隶属于TCP/IP协议簇，默认状态下，分别通过TCP端口25、110和143建立连接。

SMTP的全称是“Simple Mail Transfer Protocol”，即[简单邮件传输协议](http://baike.baidu.com/item/%E7%AE%80%E5%8D%95%E9%82%AE%E4%BB%B6%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE" \t "_blank)。它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。SMTP 协议属于TCP/IP协议簇，它帮助每台计算机在发送或中转信件时找到下一个目的地。SMTP 服务器就是遵循[SMTP协议](http://baike.baidu.com/item/SMTP%E5%8D%8F%E8%AE%AE" \t "_blank)的发送[邮件服务器](http://baike.baidu.com/item/%E9%82%AE%E4%BB%B6%E6%9C%8D%E5%8A%A1%E5%99%A8)。SMTP认证，简单地说就是要求必须在提供了**账户名和密码之后才可以登录 SMTP 服务器**，这就使得那些[垃圾邮件](http://baike.baidu.com/item/%E5%9E%83%E5%9C%BE%E9%82%AE%E4%BB%B6)的散播者无可乘之机。增加 SMTP 认证的目的是为了使用户避免受到[垃圾邮件](http://baike.baidu.com/item/%E5%9E%83%E5%9C%BE%E9%82%AE%E4%BB%B6" \t "_blank)的侵扰。

**只能发送邮件！！！！**

POP协议

POP[邮局协议](http://baike.baidu.com/item/%E9%82%AE%E5%B1%80%E5%8D%8F%E8%AE%AE)负责从[邮件服务器](http://baike.baidu.com/item/%E9%82%AE%E4%BB%B6%E6%9C%8D%E5%8A%A1%E5%99%A8)中检索电子邮件。它要求邮件服务器完成下面几种任务之一：从邮件服务器中检索邮件并从服务器中删除这个邮件；从邮件服务器中检索邮件但不删除它；不检索邮件，只是询问是否有新邮件到达。POP[协议支持](http://baike.baidu.com/item/%E5%8D%8F%E8%AE%AE%E6%94%AF%E6%8C%81)多用户互联网邮件扩展，后者允许用户在电子邮件上附带二进制文件，如文字处理文件和[电子表格](http://baike.baidu.com/item/%E7%94%B5%E5%AD%90%E8%A1%A8%E6%A0%BC)文件等，实际上这样就可以传输任何格式的文件了，包括图片和声音文件等。在用户阅读邮件时，POP命令所有的邮件信息立即下载到用户的计算机上，不在服务器上保留。

**POP3(**Post Office Protocol 3)即邮局协议的第3个版本,是因特网电子邮件的第一个离线协议标准。

**只能接受邮件！！**

IMAP协议

互联网信息访问协议（IMAP）是一种优于POP的新协议。和POP一样，IMAP也能下载邮件、从服务器中删除邮件或询问是否有新邮件，但IMAP克服了POP的一些缺点。例如，它可以决定客户机请求邮件服务器提交所收到邮件的方式，请求邮件服务器只下载所选中的邮件而不是全部邮件。客户机可先阅读邮件信息的标题和发送者的名字再决定是否下载这个邮件。通过用户的客户机电子邮件程序，IMAP可让用户在服务器上创建并管理邮件文件夹或邮箱、删除邮件、查询某封信的一部分或全部内容，完成所有这些工作时都不需要把邮件从服务器下载到用户的个人计算机上。

支持种IMAP的常用[邮件客户端](http://baike.baidu.com/item/%E9%82%AE%E4%BB%B6%E5%AE%A2%E6%88%B7%E7%AB%AF" \t "_blank)有：ThunderMail,Foxmail,Microsoft Outlook等。

**只能接受邮件！！**

实战举例：开通smtp和pop3服务

163



qq



# 邮件消息生产者

封装我们发送的邮件信息对象。

MailParam.java

|  |
| --- |
| **package** edu.cjl.activemq.producer;  **public** **class** MailParam {  **private** String from;  **private** String to;  **private** String subject;  **private** String content;    **public** MailParam(){}    **public** MailParam(String from,String to,String subject,String content){  **this**.from = from;  **this**.to = to;  **this**.subject = subject;  **this**.content = content;  }    **public** String getFrom() {  **return** from;  }  **public** **void** setFrom(String from) {  **this**.from = from;  }  **public** String getTo() {  **return** to;  }  **public** **void** setTo(String to) {  **this**.to = to;  }  **public** String getSubject() {  **return** subject;  }  **public** **void** setSubject(String subject) {  **this**.subject = subject;  }  **public** String getContent() {  **return** content;  }  **public** **void** setContent(String content) {  **this**.content = content;  }    } |

## JsonUtils

|  |
| --- |
| package com.example.mqproducer;  import java.util.List;  import com.fasterxml.jackson.core.JsonProcessingException;  import com.fasterxml.jackson.databind.JavaType;  import com.fasterxml.jackson.databind.ObjectMapper;  public class JsonUtils {  // 定义jackson对象  private static final ObjectMapper MAPPER = new ObjectMapper();  public static String objectToJson(Object data) {  try {  String string = MAPPER.writeValueAsString(data);  return string;  } catch (JsonProcessingException e) {  e.printStackTrace();  }  return null;  }    public static <T> T jsonToPojo(String jsonData, Class<T> beanType) {  try {  T t = MAPPER.readValue(jsonData, beanType);  return t;  } catch (Exception e) {  e.printStackTrace();  }  return null;  }    public static <T>List<T> jsonToList(String jsonData, Class<T> beanType) {  JavaType javaType = MAPPER.getTypeFactory().constructParametricType(List.class, beanType);  try {  List<T> list = MAPPER.readValue(jsonData, javaType);  return list;  } catch (Exception e) {  e.printStackTrace();  }    return null;  }    } |

## MailMessageProducer

|  |
| --- |
| package com.example.mqproducer;  import javax.jms.JMSException;  import javax.jms.Message;  import javax.jms.Session;  import javax.jms.TextMessage;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jms.core.JmsTemplate;  import org.springframework.jms.core.MessageCreator;  import org.springframework.stereotype.Service;  @Service  public class MailMessageProducer {  @Autowired  private JmsTemplate jmsTemplate;    public void send(MailParam mails) {  jmsTemplate.send(new MessageCreator() {  //发送一条mail的消息  @Override  public Message createMessage(Session session) throws JMSException {  TextMessage textMessage = session.createTextMessage();  String message = JsonUtils.objectToJson(mails);  textMessage.setText(message);  return textMessage;  }  });  }  } |

## 测试类

|  |
| --- |
| package com.example.mqproducer;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.test.context.junit4.SpringRunner;  @RunWith(SpringRunner.class)  @SpringBootTest  public class MailProducerTest {  @Autowired  private MailMessageProducer mmp;    @Test  public void testSendMailMessage() {  MailParam mails = new MailParam();  mails.setFrom("gududerencjl@163.com");  mails.setTo("gududerencjl@163.com");  mails.setSubject("测试啦-01");  mails.setContent("你中大奖了！！！！");  mmp.send(mails );  }  } |

# 邮件消费者

Sending email

The Spring Framework provides an easy abstraction for sending email using the JavaMailSender interface and Spring Boot provides auto-configuration for it as well as a starter module.

|  |
| --- |
| [Tip] |
| Check the [reference documentation](https://docs.spring.io/spring/docs/4.3.17.RELEASE/spring-framework-reference/htmlsingle/#mail) for a detailed explanation of how you can use JavaMailSender. |

If spring.mail.host and the relevant libraries (as defined by spring-boot-starter-mail) are available, a default JavaMailSender is created if none exists. The sender can be further customized by configuration items from the spring.mail namespace, see the [MailProperties](https://github.com/spring-projects/spring-boot/tree/v1.5.13.RELEASE/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailProperties.java" \t "_top) for more details.

In particular, certain default timeout values are infinite and you may want to change that to avoid having a thread blocked by an unresponsive mail server:

spring.mail.properties.mail.smtp.connectiontimeout=5000

spring.mail.properties.mail.smtp.timeout=3000

spring.mail.properties.mail.smtp.writetimeout=5000

spring Mail API都在**org.springframework.mail**及其子包**org.springframework.mail.javamail**中封装，且只提供了邮件**发送的封装**。

**SimpleMailMessage**: 对邮件的一个简单封装，只能用于表示一个纯文本的邮件，也不能包含附件等。

**JavaMailSenderImpl**: 邮件发送器，主要提供了邮件发送接口、透明创建Java Mail的MimeMessage、及邮件发送的配置(如:host/port/username/password...)。

MimeMailMessage、MimeMessageHelper：对MimeMessage进行了封装。Spring还提供了一个回调接口MimeMessagePreparator, 用于准备JavaMail的MIME信件。

## 添加mail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">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>mq-producer</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>mq-producer</name>  <description>Demo project for Spring Boot</description>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.15.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-activemq</artifactId>  </dependency>  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-pool</artifactId>  </dependency>  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-mail</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>com.fasterxml.jackson.core</groupId>**  **<artifactId>jackson-databind</artifactId>**  **</dependency>**  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

application.yml

|  |
| --- |
| spring:  activemq:  brokerUrl: tcp://192.168.192.150:61616  user: root  password: 111111  pool:  enabled: **true**  max-connections: 3  jms:  template:  defaultDestination: mq-queue  mail:  host: smtp.163.com  port: 25  username: gududerencjl@163.com  password: cjl123  properties:  mail:  smtp:  auth: **true**  connectiontimeout: 5000  timeout: 3000  writetimeout: 5000 |

## 发送邮件类

|  |
| --- |
| package com.example.mqconsumer;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jms.annotation.JmsListener;  import org.springframework.mail.SimpleMailMessage;  import org.springframework.mail.javamail.JavaMailSender;  import org.springframework.stereotype.Service;  @Service  public class MqConsumer {    private static final Logger log = LoggerFactory.getLogger(MqConsumer.class);    @Autowired  private JavaMailSender javaMailSender;    @JmsListener(destination="mq-queue")  public void receive(String text) {  //log.info("接收到一条消息："+text);  MailParam mails = JsonUtils.jsonToPojo(text, MailParam.class);  //发送邮件  SimpleMailMessage smm = new SimpleMailMessage();  smm.setTo(mails.getTo());  smm.setFrom(mails.getFrom());  smm.setSubject(mails.getSubject());  smm.setText(mails.getContent());  javaMailSender.send(smm);    log.info("发送邮件成功！！");  }  } |

**运行测试之前，请先开启邮箱发送服务器的smtp协议功能！！**



## 常见错误

**om.sun.mail.smtp.SMTPSendFailedException: 553 Mail from must equal authorized user**

**原因：请设置发送人和登录人一致！！**

面试题：

1.7之前（包括）匿名内部类，不能访问方法局部变量，要在变量前加final关键字。

1.8之后不需要加。

