咕泡学院 VIP 课：分布式消息通信 ActiveMQ（一）

课程目标

1. 关于消息中间件的发展
2. ActiveMQ 简介
3. 从 JMS 规范来了解 ActiveMQ
4. JMS 规范
5. JMS+ActiveMQ
6. JMS 的基本功能
7. 消息的确认方式
8. 消息中间件的应用场景
9. JMS 基本概念和模型

10. 实现一个基于 JMS 的消息发送接收示例
11. JMS 的基本功能
12. JMS 的可靠性机制
13. JMS 发布订阅功能

课堂笔记

消息中间件的初步认识

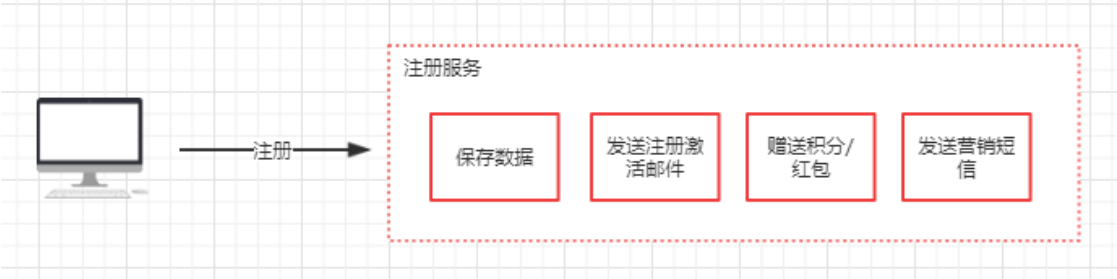
什么是消息中间件？

消息中间件是值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的集成。通过提供消息传递和消息排队模型，可以在分布式架构下扩展进程之间的通信。

消息中间件能做什么？

消息中间件主要解决的就是分布式系统之间消息传递的问题，它能够屏蔽各种平台以及协议之间的特性，实现应用程序之间的协同。举个非常简单的例子，就拿一个电商平台的注册功能来简单分析下，用户注册这一个服务，不单

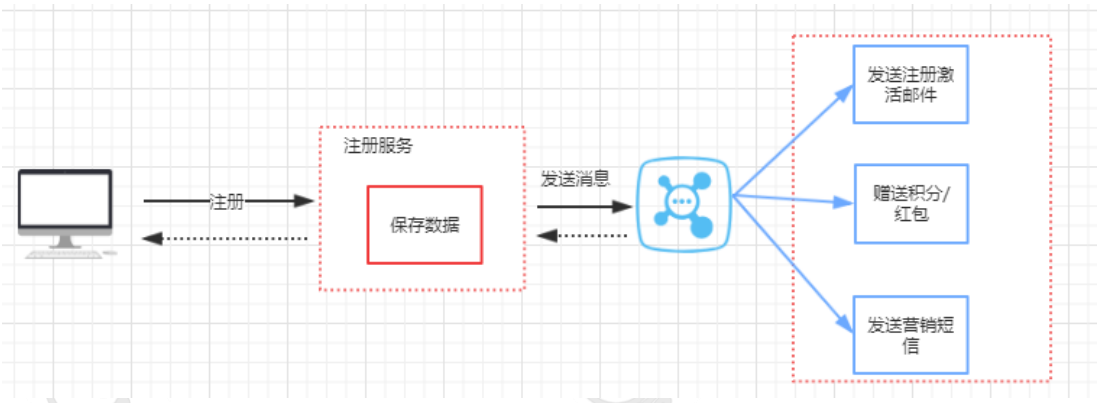
单只是 insert 一条数据到数据库里面就完事了，还需要发送激活邮件、发送新人红包或者积分、发送营销短信等一系列操作。假如说这里面的每一个操作，都需要消耗 1s，那么整个注册过程就需要耗时 4s 才能响应给用户。



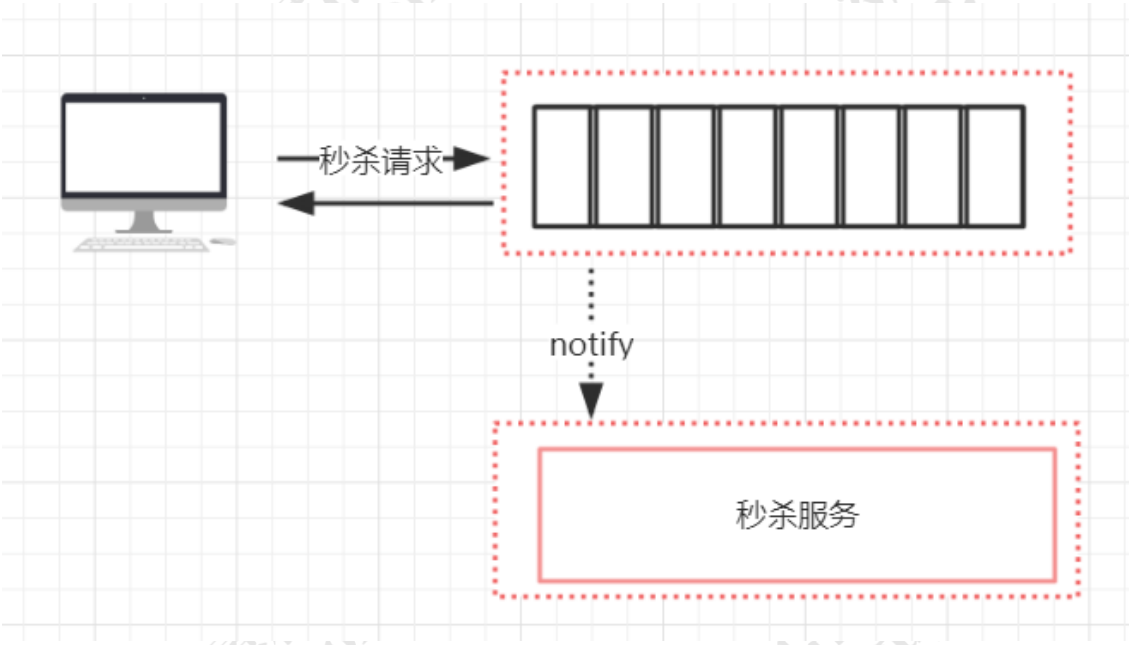
但是我们从注册这个服务可以看到，每一个子操作都是相对独立的，同时，基于领域划分以后，发送激活邮件、发送营销短信、赠送积分及红包都属于不同的子域。所以我们可以对这些子操作进行来实现异步化执行，类似于多线程并行处理的概念。

如何实现异步化呢？用多线程能实现吗？多线程当然可以实现，只是，消息的持久化、消息的重发这些条件，多线程并不能满足。所以需要借助一些开源中间件来解决。而分布式消息队列就是一个非常好的解决办法，引入分布式消息队列以后，架构图就变成这样了（下图是异步消息队列的场景）。通过引入分布式队列，就能够大大提升程序的处理效率，并且还解决了各个模块之间的耦合问题

- 这个是分布式消息队列的第一个解决场景【异步处理】



我们再来展开一种场景，通过分布式消息队列来实现流量整形，比如在电商平台的秒杀场景下，流量会非常大。通过消息队列的方式可以很好的缓解高流量的问题



- 用户提交过来的请求，先写入到消息队列。消息队列是有长度的，如果消息队列长度超过指定长度，直接抛弃
- 秒杀的具体核心处理业务，接收消息队列中消息进行处理，这里的消息处理能力取决于消费端本身的吞吐量

当然，消息中间件还有更多应用场景，比如在弱一致性事务模型中，可以采用分布式消息队列的实现最大能力通知

方式来实现数据的最终一致性等等

ActiveMQ 简介

ActiveMQ 是完全基于 JMS 规范实现的一个消息中间件产品。是 Apache 开源基金会研发的消息中间件。ActiveMQ 主要应用在分布式系统架构中，帮助构建高可用、高性能、可伸缩的企业级面向消息服务的系统

ActiveMQ 特性

1. 多语言和协议编写客户端
   语言：java/C/C++/C#/Ruby/Perl/Python/PHP
   应用协议：openwire/stomp/REST/ws/notification/XMPP/AMQP
2. 完全支持 jms1.1 和 J2ee1.4 规范
3. 对 spring 的支持，ActiveMQ 可以很容易内嵌到 spring 模块中

ActiveMQ 安装

1. 登录到 http://activemq.apache.org/activemq-5150-release.html，找到 ActiveMQ 的下载地址
2. 直接 copy 到服务器上通过 tar -zxvf apache-activeMQ.tar.gz

3. 启动运行

   a) 普通启动：到 bin 目录下，　sh activemq start

   b) 启动并指定日志文件　sh activemq start > /tmp/activemqlog

4. 检查是否已启动

   ActiveMQ 默认采用 61616 端口提供 JMS 服务，使用 8161 端口提供管理控制台服务，执行以下命令可以检查是否成功启动 ActiveMQ 服务

   netstat -an|grep 61616

5. 通过 http://192.168.11.156:8161 访问 activeMQ 管理页面，默认帐号密码 admin/admin

6. 关闭 ActiveMQ;　sh activemq stop

从 JMS 规范来了解 ActiveMQ

JMS 定义

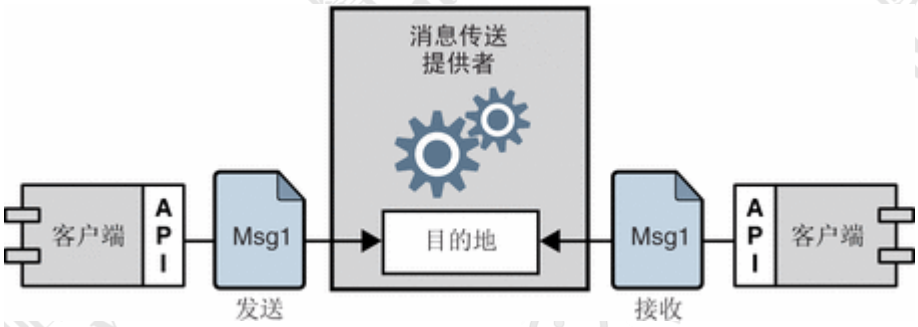
Java 消息服务（Java Message Service）是 java 平台中关于面向消息中间件的 API，用于在两个应用程序之间，或者分布式系统中发送消息，进行异步通信。

JMS 是一个与具体平台无关的 API，绝大多数 MOM（Message Oriented Middleware）（面向消息中间件）提供商都对 JMS 提供了支持。今天给大家讲的 ActiveMQ 就

是其中一个实现

什么是 MOM

MOM 是面向消息的中间件，使用消息传送提供者来协调消息传送操作。MOM 需要提供 API 和管理工具。客户端使用 api 调用，把消息发送到由提供者管理的目的地。在发送消息之后，客户端会继续执行其他工作，并且在接收方收到这个消息确认之前，提供者一直保留该消息。



MOM 的特点

1. 消息异步接收，发送者不需要等待消息接受者响应
2. 消息可靠接收，确保消息在中间件可靠保存。只有接收方收到后才删除消息

Java 消息传送服务规范最初的开发目的是为了使 Java 应用程序能够访问现有 MOM 系统。引入该规范之后，它已被许多现有的 MOM 供应商采用并且已经凭借自身的功能实现为异步消息传送系统。

其他开源的 JMS 提供商

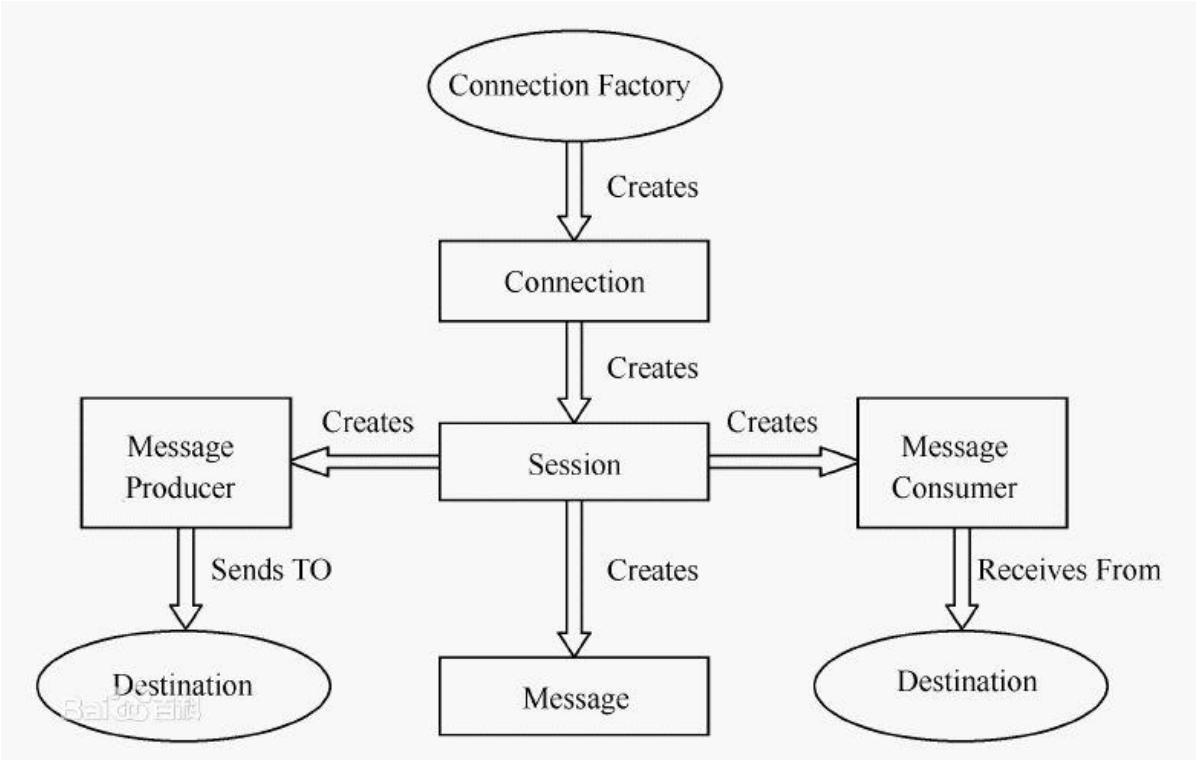
JbossMQ(jboss4)、jboss messaging(jboss5)、joram、ubermq、mantamq、openjms...
大部分基于的 JMS provider 开源的消息中间件都已经停止维护了，剩下的几个都抱到了大腿，比如 Jboss mq 和 jboss、joram 与 jonas(objectweb 组织)、ActiveMQ 与 Geronimo(apache 基金组织)。

JMS 规范

我们已经知道了 JMS 规范的目的是为了使得 Java 应用程序能够访问现有 MOM (消息中间件)系统，形成一套统一的标准规范，解决不同消息中间件之间的协作问题。在创建 JMS 规范时，设计者希望能够结合现有的消息传送的精髓，比如说

1. 不同的消息传送模式或域，例如点对点消息传送和发布/订阅消息传送
2. 提供于接收同步和异步消息的工具
3. 对可靠消息传送的支持
4. 常见消息格式，例如流、文本和字节

JMS 的体系结构

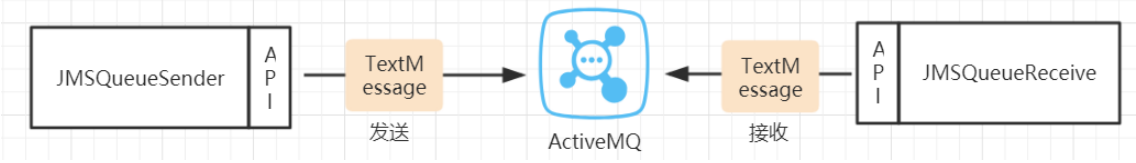


通过 JMS 规范结合 ActiveMQ 实现消息发送案例

➢ 查看 gitlab 上的代码

案例总结

这个案例的架构图如下



细化 JMS 的基本功能

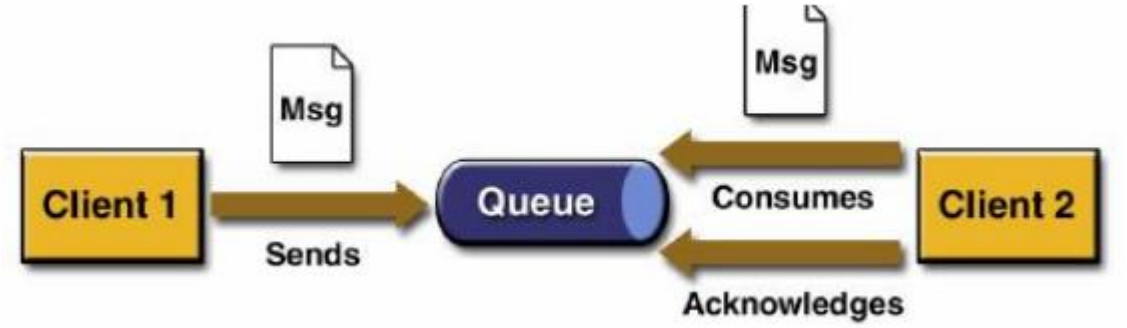
通过前面的内容讲解以及案例演示，我们已经知道了 JMS 规范以及他的基本功能是用于和面向消息中间件相互通信的应用程序的接口，那么 JMS 提供的具体标准有哪些呢？我们来仔细去研究下

消息传递域

JMS 规范中定义了两种消息传递域：点对点（point-to-point）消息传递域 和发布/订阅 消息传递域(publish/subscribe)
简单理解就是：有点类似于我们通过 qq 聊天的时候，在群里面发消息和给其中一个同学私聊消息。在群里发消息，所有群成员都能收到消息。私聊消息只能被私聊的学员能收到消息，

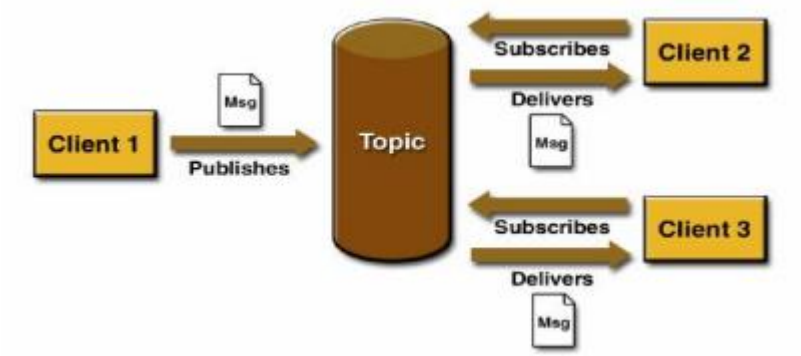
点对点消息传递域

1. 每个消息只能有一个消费者
2. 消息的生产者和消费者之间没有时间上的相关性。无论消费者在生产者发送消息的时候是否处于运行状态，都可以提取消息



发布订阅消息传递域

1. 每个消息可以有多个消费者
2. 生产者和消费者之间有时间上的相关性。订阅一个主题的消费者只能消费自它订阅之后发布的消息。JMS 规范允许客户创建持久订阅，这在一定程度上降低了时间上的相关性要求。持久订阅允许消费者消费它在未处于激活状态时发送的消息



消息结构组成

JMS 消息由及部分组成：消息头、属性、消息体

消息头

消息头(Header) - 消息头包含消息的识别信息和路由信息，消息头包含一些标准的属性如：

JMSDestination　　消息发送的目的地，queue 或者 topic)

JMSDeliveryMode　传送模式。持久模式和非持久模式

JMSPriority　　消息优先级（优先级分为 10 个级别，从 0(最低)到 9(最高). 如果不设定优先级，默认级别是 4。需要注意的是，JMS provider 并不一定保证按照优先级的顺序提交消息)

JMSMessageID　　唯一识别每个消息的标识

属性

按类型可以分为应用设置的属性，标准属性和消息中间件定义的属性

1. 应用程序设置和添加的属性，比如
   Message.setStringProperty(“key”,”value”);
   通过下面的代码可以获得自定义属性的，在接收端的代码中编写

在发送端，定义消息属性

```
message.setStringProperty("Mic","Hello
World");
```

在接收端接收数据

```
Enumeration
enumeration=message.getPropertyNames();
while(enumeration.hasMoreElements()){
    String
name=enumeration.nextElement().toString();

System.out.println("name:"+name+":"+messag
e.getStringProperty(name));
    System.out.println();
}
```

2. JMS 定义的属性

使用"JMSX"作为属性名的前缀，通过下面这段代码可以返回所有连接支持的 JMSX 属性的名字

```
Enumeration names=connection.getMetaData().getJMSXPropertyNames
while(names.hasMoreElements()){
    String name=(String)names.nextElement();
    System.out.println(name);
}
```

3. JMS provider 特定的属性

消息体

就是我们需要传递的消息内容，JMS API 定义了 5 中消息

体格式，可以使用不同形式发送接收数据，并可以兼容现有的消息格式，其中包括

TextMessage	java.lang.String 对象，如 xml 文件内容
MapMessage	名/值对的集合，名是 String 对象，值类型可以是 Java 任何基本类型
BytesMessage	字节流
StreamMessage	Java 中的输入输出流
ObjectMessage	Java 中的可序列化对象
Message	没有消息体，只有消息头和属性。

绝大部分的时候，我们只需要基于消息体进行构造

持久订阅

持久订阅的概念，也很容易理解，比如还是以 QQ 为例，我们把 QQ 退出了，但是下次登录的时候，仍然能收到离线的消息。

持久订阅就是这样一个道理，持久订阅有两个特点：

1. 持久订阅者和非持久订阅者针对的 Domain 是 Pub/Sub，而不是 P2P
2. 当 Broker 发送消息给订阅者时，如果订阅者处于 未激活状态状态：持久订阅者可以收到消息，而非持久订阅者则收不到消息。

当然这种方式也有一定的影响：当持久订阅者处于 未激活状态时，Broker 需要为持久订阅者保存消息；如果持久订阅者订阅的消息太多则会溢出。

消费端改动

```
connection=connectionFactory.createConnectio
n();
connection.setClientID("Mic-001");
connection.start();
Session
session=connection.createSession(Boolean.TRU
E,Session.AUTO_ACKNOWLEDGE);
Topic
destination=session.createTopic("myTopic");
MessageConsumer
consumer=session.createDurableSubscriber(des
tination,"Mic-001");
TextMessage
message=(TextMessage)consumer.receive();
System.out.println(message.getText());
```

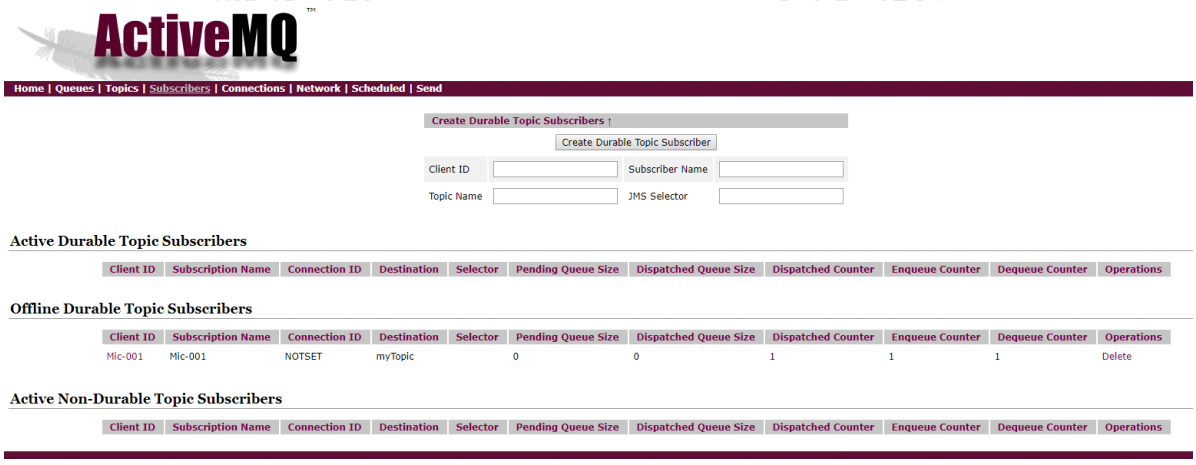
修改三处地方，然后先启动消费端去注册一个持久订阅。

持久订阅时，客户端向 JMS 服务器注册一个自己身份的 ID，当这个客户端处于离线时，JMS Provider 会为这个 ID 保存所有发送到主题的消息，当客户再次连接到 JMS Provider 时，会根据自己的 ID 得到所有当自己处于离线时发送到主题的消息。

这个身份 ID，在代码中的体现就是 connection 的 ClientID，这个其实很好理解，你要想收到朋友发送的 qq 消息，前提就是你得先注册个 QQ 号，而且还要有台能上网的设备，电脑或手机。设备就相当于是 clientId 是唯一的；qq 号相当于是订阅者的名称，在同一台设备上，不能用同一个 qq 号挂 2 个客户端。连接的 clientId 必须是唯一的，订阅者的名称在同一个连接内必须唯一。这样才能唯一的确定连接和订阅者。

activeMQ 控制台的截图

设置持久订阅以后，在控制台能看到下图的变化



JMS 消息的可靠性机制

理论上来说，我们需要保证消息中间件上的消息，只有被消费者确认过以后才会被签收，相当于我们寄一个快递出去，收件人没有收到快递，就认为这个包裹还是属于待签收状态，这样才能保证包裹能够安全达到收件人手里。消息中间件也是一样。

消息的消费通常包含 3 个阶段：客户接收消息、客户处理消息、消息被确认

首先，来简单了解 JMS 的事务性会话和非事务性会话的概念

JMS Session 接口提供了 commit 和 rollback 方法。事务提交意味着生产的所有消息被发送，消费的所有消息被确认；事务回滚意味着生产的所有消息被销毁，消费的所有消息被恢复并重新提交，除非它们已经过期。 事务性的会话总是牵涉到事务处理中，commit 或 rollback 方法一旦被调用，一个事务就结束了，而另一个事务被开始。关闭事务性会话将回滚其中的事务

在事务型会话中

在事务状态下进行发送操作，消息并未真正投递到中间件，而只有进行 session.commit 操作之后，消息才会发送到中

间件，再转发到适当的消费者进行处理。如果是调用 rollback 操作，则表明，当前事务期间内所发送的消息都取消掉。通过在创建 session 的时候使用 true or false 来决定当前的会话是事务性还是非事务性

connection.createSession(Boolean.TRUE,Session.AUTO_ACKNOWLEDGE);

在事务性会话中，消息的确认是自动进行，也就是通过 session.commit()以后，消息会自动确认。

- 必须保证发送端和接收端都是事务性会话

在非事务型会话中

消息何时被确认取决于创建会话时的应答模式(acknowledgement mode). 有三个可选项

Session.AUTO_ACKNOWLEDGE

当客户成功的从 receive 方法返回的时候，或者从 MessageListenner.onMessage 方法成功返回的时候，会话自动确认客户收到消息。

Session.CLIENT_ACKNOWLEDGE

客户通过调用消息的 acknowledge 方法确认消息。

CLIENT_ACKNOWLEDGE 特性

在这种模式中，确认是在会话层上进行，确认一个被消费的消息将自动确认所有已被会话消费的消息。列如，如果一个消息消费者消费了 10 个消息，然后确认了第 5 个消息，那么 0~5 的消息都会被确认 ->

演示如下：发送端发送 10 个消息，接收端接收 10 个消息，但是在 i==5 的时候，调用 message.acknowledge()进行确认，会发现 0~4 的消息都会被确认

Session.DUPS_ACKNOWLEDGE

消息延迟确认。指定消息提供者在消息接收者没有确认发送时重新发送消息，这种模式不在乎接受者收到重复的消息。

消息的持久化存储

消息的持久化存储也是保证可靠性最重要的机制之一，也就是消息发送到 Broker 上以后，如果 broker 出现故障宕机了，那么存储在 broker 上的消息不应该丢失。可以通过下面的代码来设置消息发送端的持久化和非持久化特性

```
MessageProducer
producer=session.createProducer(destination)
;
```

```
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
```

- 对于非持久的消息，JMS provider 不会将它存到文件/数据库等稳定的存储介质中。也就是说非持久消息驻留在内存中，如果 jms provider 宕机，那么内存中的非持久消息会丢失
- 对于持久消息，消息提供者会使用存储-转发机制，先将消息存储到稳定介质中，等消息发送成功后再删除。如果 jms provider 挂掉了，那么这些未送达的消息不会丢失；jms provider 恢复正常后，会重新读取这些消息，并传送给对应的消费者。