内容提要

- 网络I/O模型
- 基于服务治理的RPC解决方案
- 消息机制
- 分布式数据存储
- 互联网分布式架构设计与开发

MQ概念 – 中间件

中间件类型	产品举例
面向消息中间件 (MOM)	**IBM MQSeries, Microsoft MSMQ, BEA MessageQ, JBossMQ**
数据连接	**ODBC, JDBC, etc.**
远程过程调用 (RPC)	**Dubbo,Thrift**
对象请求代理 (ORB)	符合**CORBA**标准的，如**Orbix, Visibroker, BEA Objectbroke，Java IIOP**；还有**Java RMI**
交易流程控制 (TPM)	**Microsoft Transaction Server (MTS), IBM CICS, IBM Encina, BEA Tuxedo**

问题引入

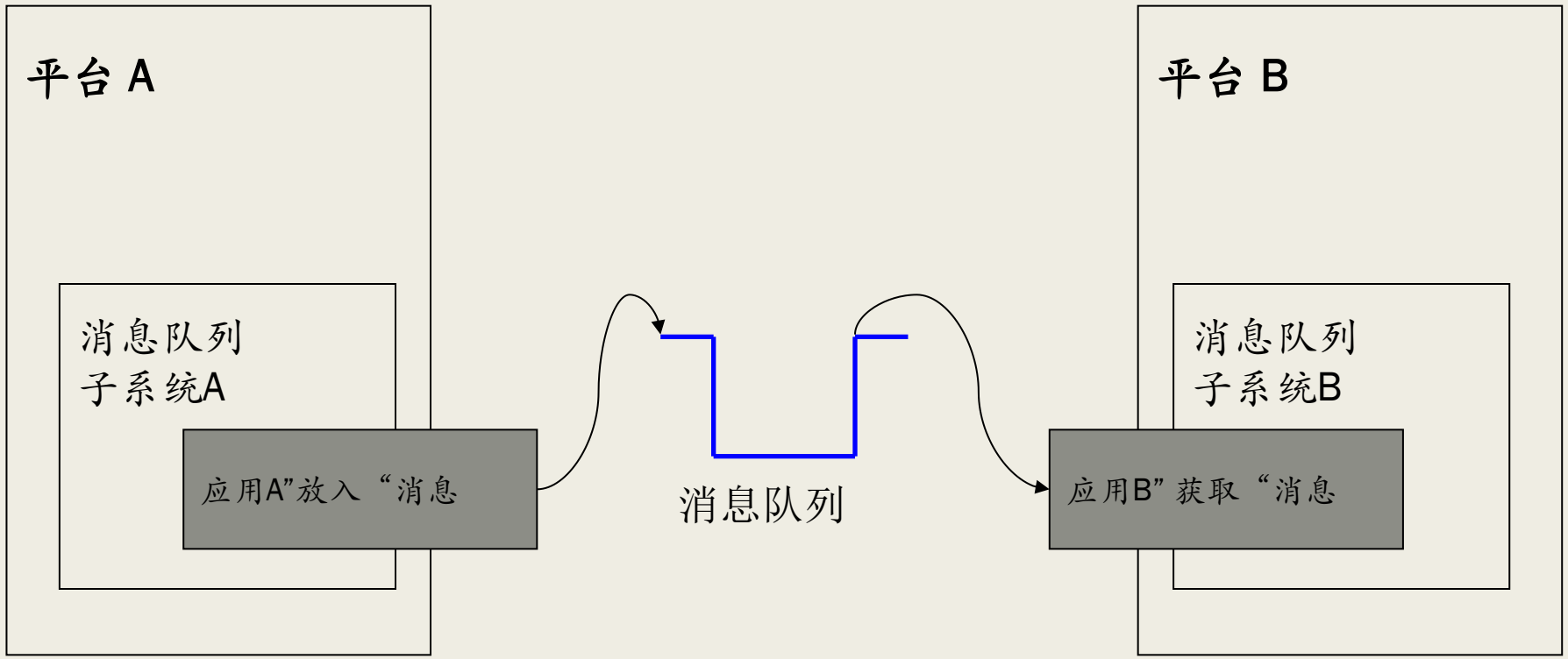
- Why?
  - *很多非实时异步需求，同步成本太高*
- How?
  - *消息队列机制*
- Where?
  - *异步处理、应用解耦、流量削锋、日志处理和消息通讯*

MQ概念 – 异步 vs. 同步通信

- 消息队列框架的通信模式是异步的!
- 同步: 应用发送请求后一直等待，直到请求被处理。
  - *客户端请求的同时，服务必须在线。*
- 异步: 应用发送请求，然后在将来检查请求是否被处理完成。
  - *客户端请求时，服务无需在线。*

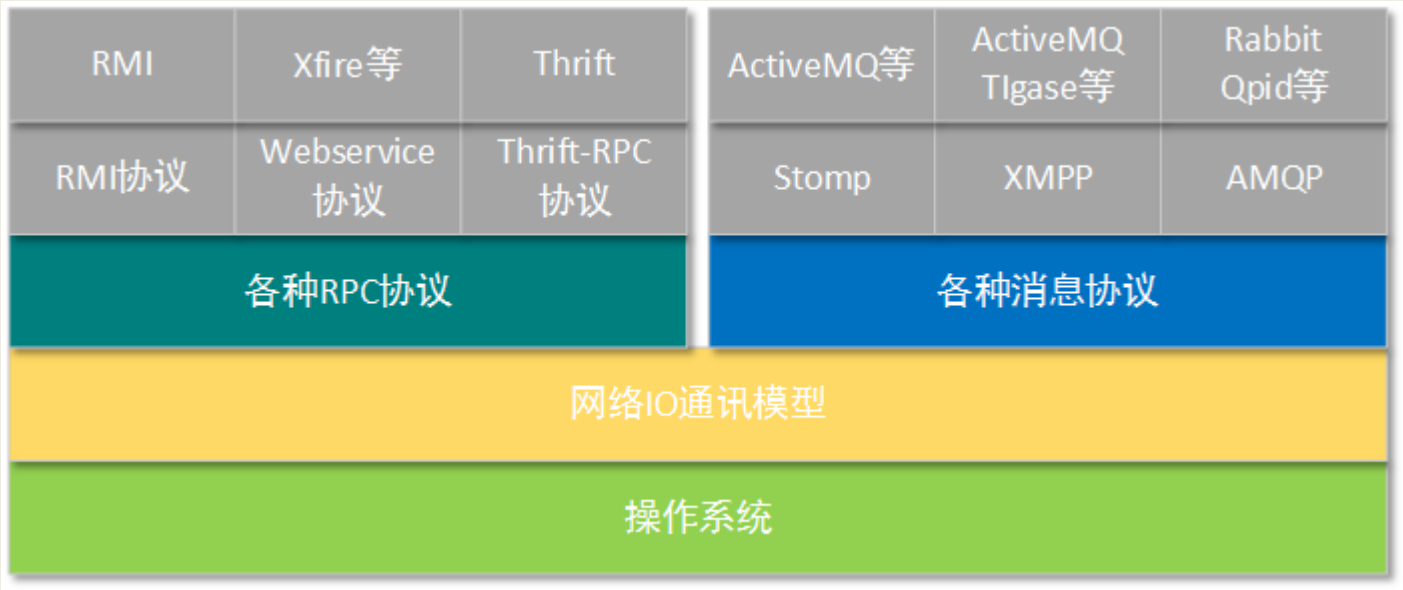
MQ概念 – Message Oriented Middleware

MOM中间件允许一个应用向另一个应用发送消息，而无论该应用是否在线。



中间件类型	同步	异步
MOM		**X**
Data Connectivity	**X**	
RPC	**X**	
ORB	**X**	
TPM	**X**	

分布式架构的流行应用案例



同步的 good/bad

Good

易编程

输出立即可知

更易恢复错误(通常)

更好实时响应(通常)

Bad

服务必须启动、在线

请求方阻碍占用资源

通常要求面向连接的通信协议

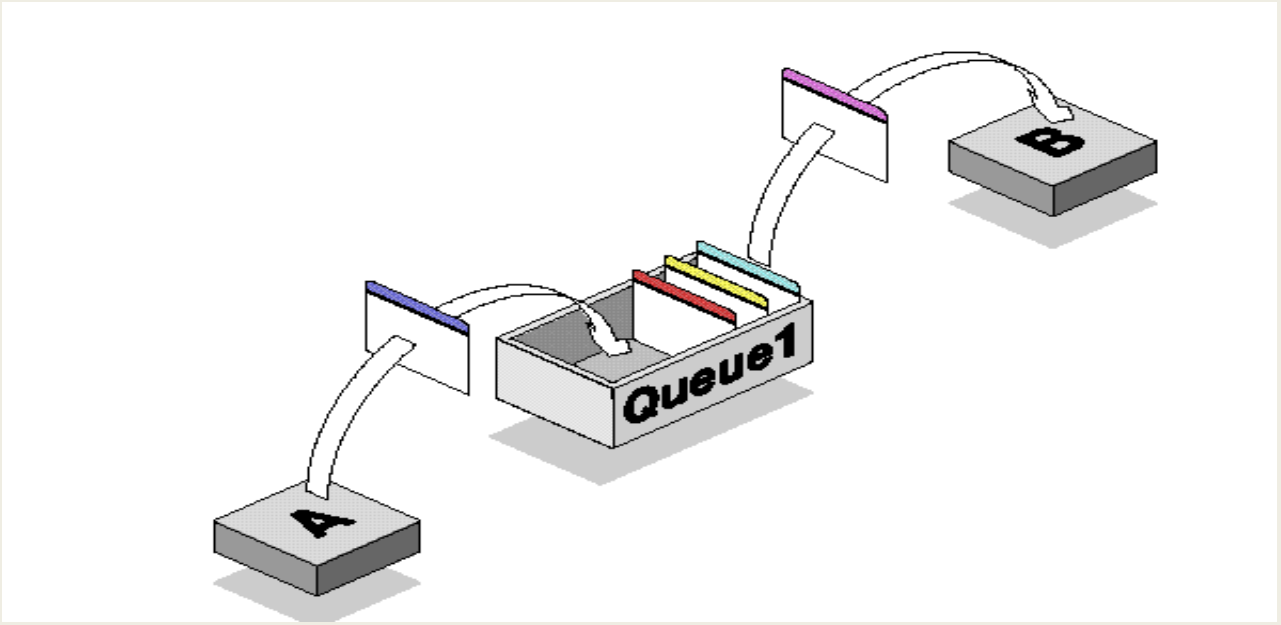
异步的 good/bad

Good	Bad
请求无需指定服务器 服务无需在线 由于没有占用，资源可以释放 可以使用非连接协议	响应时间不可预测 错误处理通常复杂 难以设计程序?

消息机制的优势

- ✓ 我们可以集中精力去做应用本身的设计.
- ✓ 再也不用管有关环境方面的细节了.
- ✓ 应用变得可以移植、扩展了.

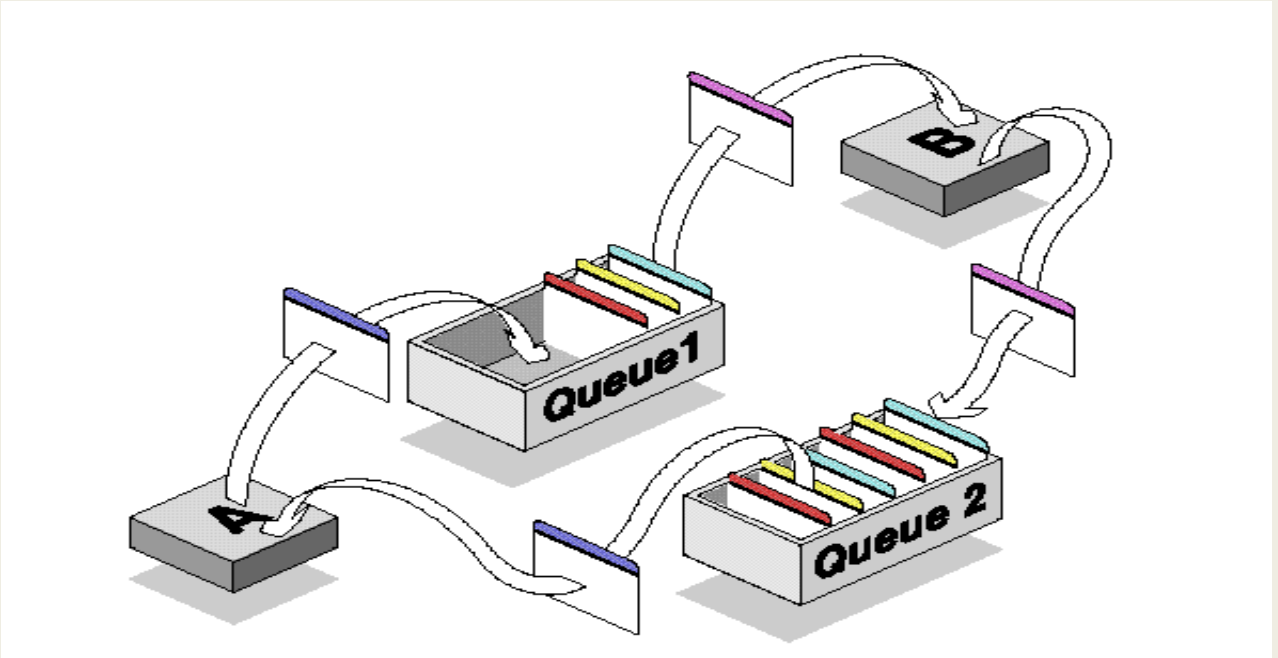
程序通过放置消息到队列来进行通信. 图中, A程序 将消息放入Queue1, B程序读取Queue1的消息.



Note: A 和 B 不必位于同一台机器上!

消息工作原理 (2)

通信可以是单向或者双向。图中, A 发送到 B上的Queue1, 然后B又响应到 A 的Queue2

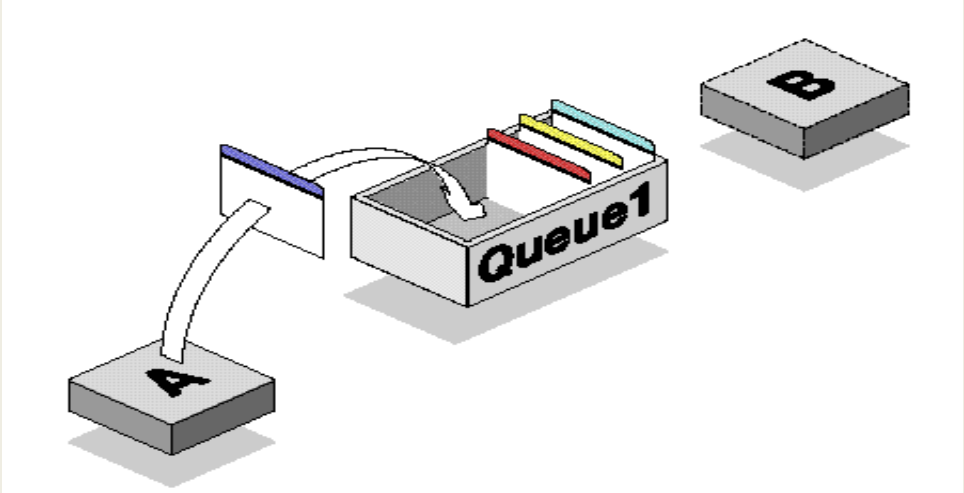


消息工作原理 (3) 之 消息队列特性

- 关于消息队列，有三个关键事实使得它不同于其他通信方式:
  1) *要通信的应用程序可以运行在不同时间.*
  2) *对于应用间的结构没有限制.*
  3) *对于应用程序来说，底层的环境差异被屏蔽掉.*

消息队列特性 - 应用程序可以运行在不同时间

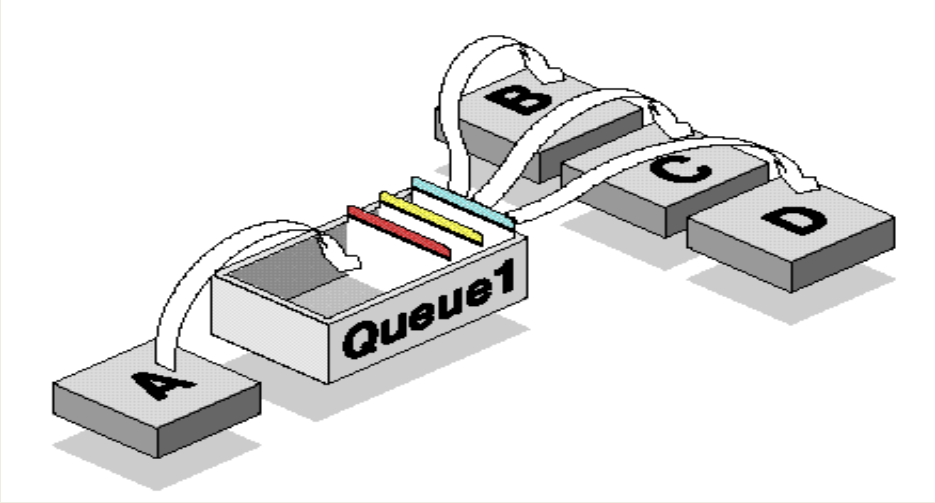
程序A,B不必同时在线



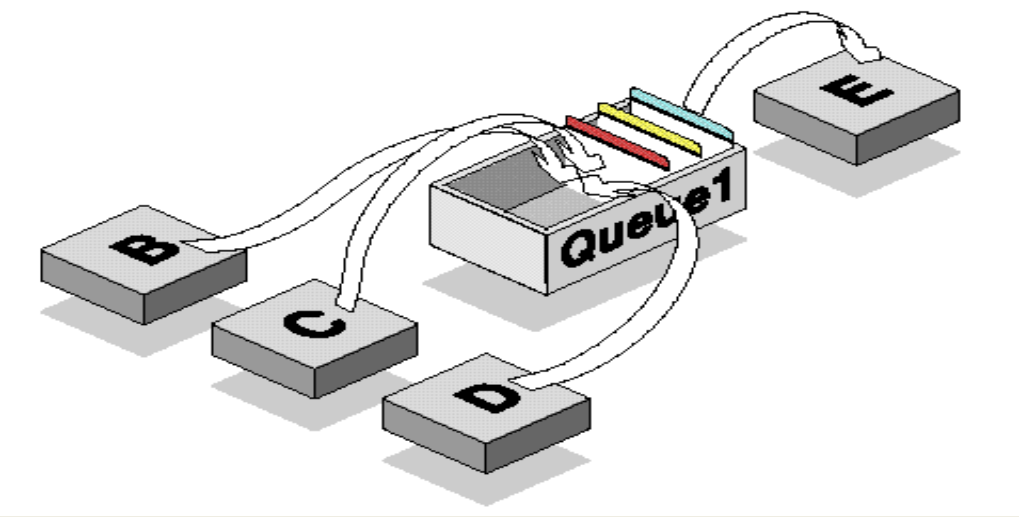
关键：消息队列相对于使用它们的应用而言是独立存在的!

消息队列特性 – 对于应用间的结构没有限制

应用之间可能是一对多



或者是多对一



消息队列特性 – 环境差异被屏蔽掉

- 不用关心运行在什么OS之上.
- 不用关心程序由什么语言写的.
- 不用关心底层使用什么通信协议.

消息基本概念

- 消息：信息的载体
- 消息协议：需要按照一种统一的格式描述消息，这种统一的格式称之为消息协议。
- 消息队列：消息从某一端发出后，首先进入一个容器进行临时存储，当达到某种条件后，再由这个容器发送给另一端。这个容器的一种具体实现就是消息队列

常见的消息协议

- XMPP：基于XML，用于IM系统的开发
- Stomp：**结构简单**
- AMQP：经典

JMS介绍

Java Message Service（JMS）是SUN提出的旨在统一各种MOM（Message-Oriented Middleware）系统接口的规范，它包含点对点（Point to Point，PTP）和发布/订阅（Publish/Subscribe，pub/sub）两种消息模型，提供可靠消息传输、事务和消息过滤等机制。

JMS是规范（什么是规范？），不是实现。

JMS和消息队列MQ

- JMS不是消息队列，更不是某种消息队列协议。
- **是一套规范的JAVA API 接口**
- **JMS和消息中间件厂商无关，需要各个厂商进行实现，**大部分消息中间件产品都支持JMS 接口规范
- **可以使用JMS API来连接Stomp协议的产品（例如ActiveMQ）**
- **类似与JDBC和mysql的关系**

JMS模型

Java消息服务应用程序结构支持两种模型：

1.点对点模型(基于队列)

每个消息只能有一个消费者。消息的生产者和消费者之间没有时间上的

相关性.可以由多个发送者，但只能被一个消费者消费。

- 一个消息只能被一个接受者接受一次
- 生产者把消息发送到队列中(Queue)，这个队列可以理解为电视机频道(channel)
- 在这个消息中间件上有多个这样的channel
- 接受者无需订阅，当接受者未接受到消息时就会处于阻塞状态

2. 发布者/订阅者模型（基于主题的）

每个消息可以有多个消费者。

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

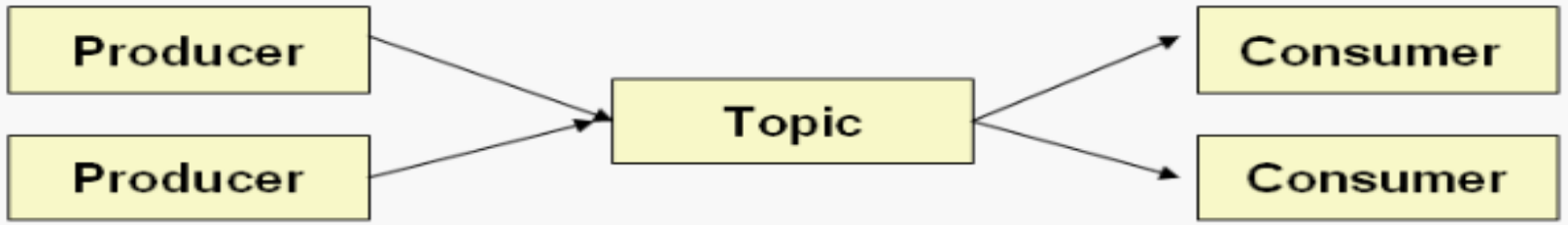
费自它订阅之后发布的消息.

- 允许多个接受者，类似于广播的方式
- 生产者将消息发送到主题上(Topic)
- 接受者必须先订阅

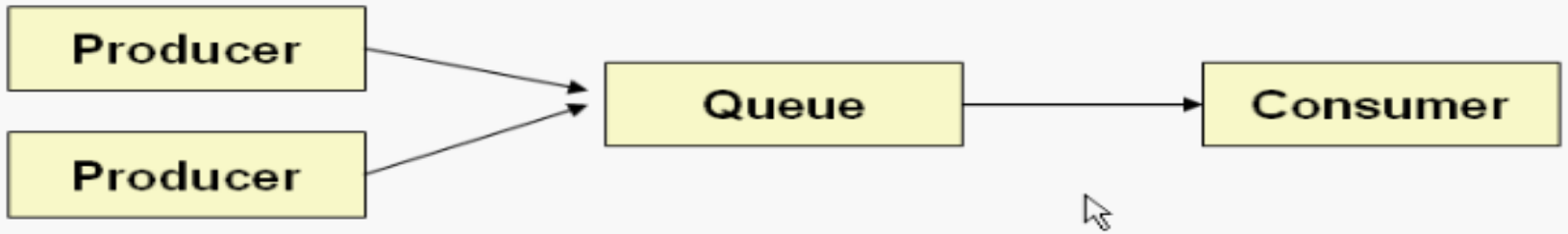
注:持久化订阅者：特殊的消费者，告诉主题，我一直订阅着，即使网络断开，消息服务器也记住所有持久化订阅者，如果有新消息，也会知道必定有人回来消费。

JMS消息发送模式

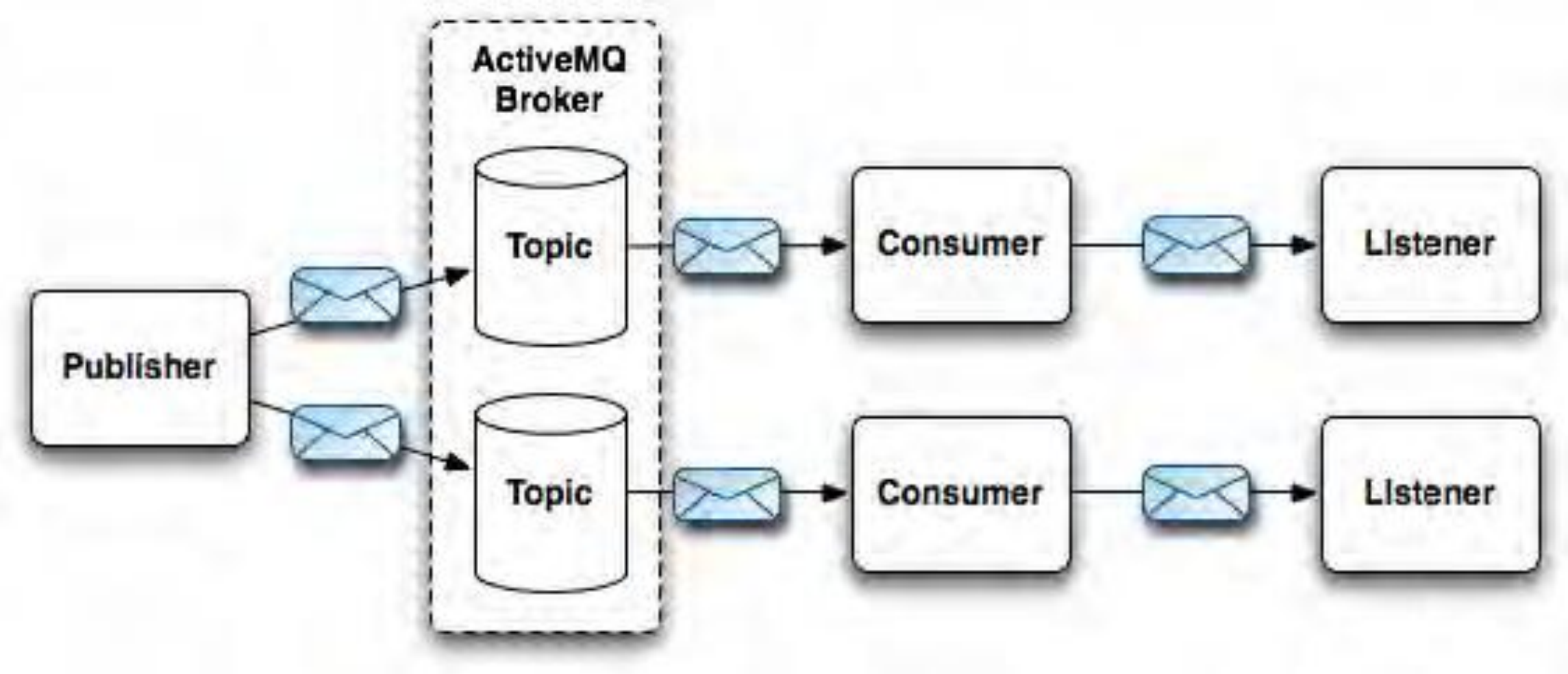
- Publish/subscribe (pub/sub)



- Point-to-pint (PTP)



Topic 发送模式


ActiveMQ
Broker
Topic
Topic
Publisher
Consumer
Consumer
Listener
Listener

JMS体系架构

- 提供者：连接面向消息中间件的，JMS接口的一个实现。
- 客户：生产或消费基于消息的Java的应用程序或对象。
- 生产者：创建并发送消息的JMS客户。
- 消费者：接收消息的JMS客户。
- 消息：包括可以在JMS客户之间传递的数据的对象。
- 队列：一个容纳那些被发送的等待阅读的消息的区域。
- 主题：一种支持发送消息给多个订阅者的机制。

JMS对象模型

- 连接工厂（Connection Factory）
- 连接（Connection）
- 会话（Session）
- 目的（Destination：Queue、Topic）
- 生产者（Message Producer）
- 消费者（Message Consumer）

JMS公共接口

JMS 公共	点对点域	发布/订阅域
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver	TopicSubscriber

JMS的基本对象说明

- **连接工厂**：
  - *连接工厂是客户用来创建连接的对象，例如ActiveMQ提供的ActiveMQConnectionFactory。*
- **连接**： JMS Connection封装了JMS 客户端到JMS Provider 的连接与JMS提供者之间的一个虚拟的连接。
- **会话**： JMS Session是生产和消费消息的一个单线程上下文。会话用于创建消息的生产者（producer），消费者（consumer），消息（message）等
  - *会话,是一个事务性的上下文。*
  - *消息的生产和消费不能包含在同一个事务中。*

JMS的基本对象

- **生产者**：MessageProducer 由Session 对象创建的用来发送消息的对象
- **消费者**：MessageConsumer 由Session 对象创建的用来发送消息的对象
- **消息**：Message jms消息包括消息头和消息体以及其它的扩展属性。
  - *JMS定义的消息类型有TextMessage、MapMessage、BytesMessage、StreamMessage和ObjectMessage。*
- **目的地**：Destination，消息的目的地，是用来指定生产的消息的目标和它消费的消息的来源的对象。
- **消息队列**：Queue 点对点的消息队列
- **消息主题**：Tipic 发布订阅的消息队列

JMS消息存储方式

- NON_PERSISTENT：禁止固化消息，仅将消息放到内存中，适合消息量较少、可靠性要求不高的系统。
- PERSISTENT：固化消息，将消息以文件或者数据库的方式进行固化，可有效提高可靠性，但会降低性能。

JMS消息格式

- StreamMessage：Java原始值的数据流
- MapMessage：键-值对
- TextMessage：字符串对象（最常用）
- ObjectMessage：序列化的 Java对象
- BytesMessage：字节流

JMS消息实体

- 消息头（必须）：包含用于识别和为消息寻找路由的操作设置。
- 消息属性（可选）：包含额外的属性，支持其他提供者和用户的兼容。
- 消息体（可选）：消息具体内容。

JMS提供者实现

- ActiveMQ：apache出品，开源，国际上比较流行，不支持超大规模。
- RabbitMQ：mozilla出品，开源，不支持事务，erlang语言开发。
- RocketMQ：阿里巴巴出品，部分开源，该公司内部大量使用，集群50台，日消息量百亿级。
- SonicMQ：progress出品，商业，部分国内网站使用。
- ZeroMQ：C语言开发，效率极高，但不成熟，开源。
- kafka：擅长大数据\海量日志处理
- 其他IBM、微软等商业产品

ActiveMQ定义

- MQ即Message Queue。Apache出品，最流行的，能力强劲的开源消息总线，是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现
- 已经在很多公司得到应用，社区成熟，学习文档多。

ActiveMQ特性

- 支持多语言和多协议
- 完全支持持久化、事务
- 支持内嵌至Spring
- 支持部署至JBoss、WebLogic等服务器上
- 支持集群
- 友好管理界面，测试方便

ActiveMQ下载与安装

1、到官网下载对应操作系统的压缩包

ActiveMQ™

Overview > Download > ActiveMQ 5.13.2 Release

ActiveMQ 5.13.2 Release

Apache ActiveMQ 5.13.2 includes over 15 resolved issues and improvements.

Getting the Binary Distributions

Description	Download Link	*Verify*
Windows Distribution	apache-activemq-5.13.2-bin.zip	ASC, MD5, SHA1
Unix/Linux/Cygwin Distribution	apache-activemq-5.13.2-bin.tar.gz	ASC, MD5, SHA1

ActiveMQ下载与安装

2、

将压缩包解压。Windows使用好压、360压缩、winrar等软件；

Linux使用tar、unzip等指令

ActiveMQ下载与安装

3、

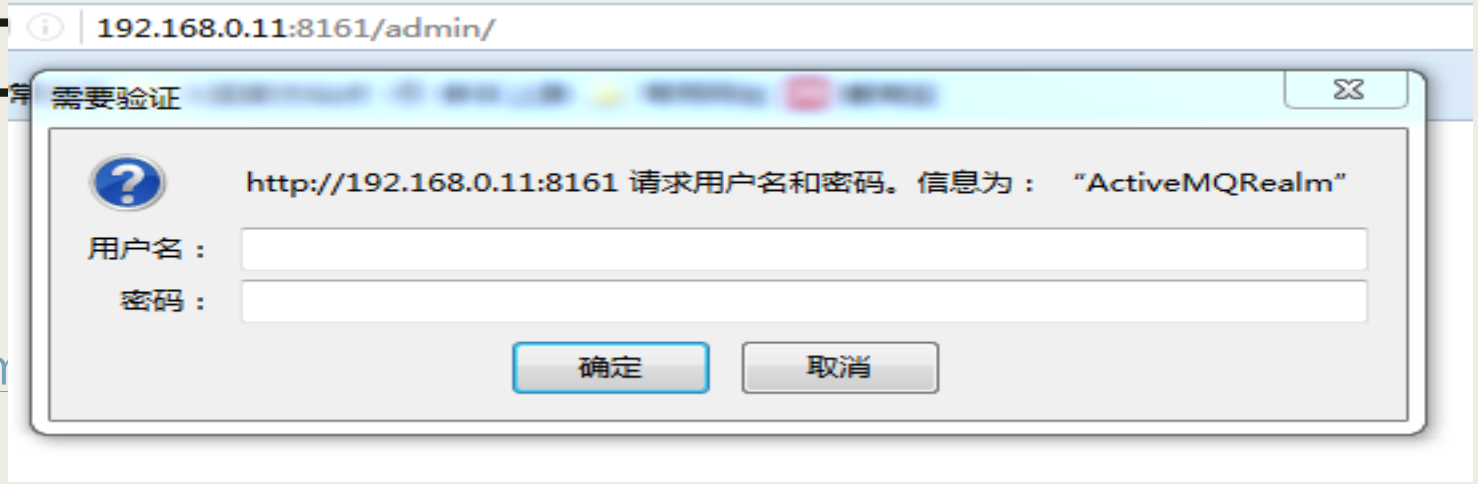
运行主目录下bin/activemq.bat(windows)

bin/activemq start(linux)来启动服务。

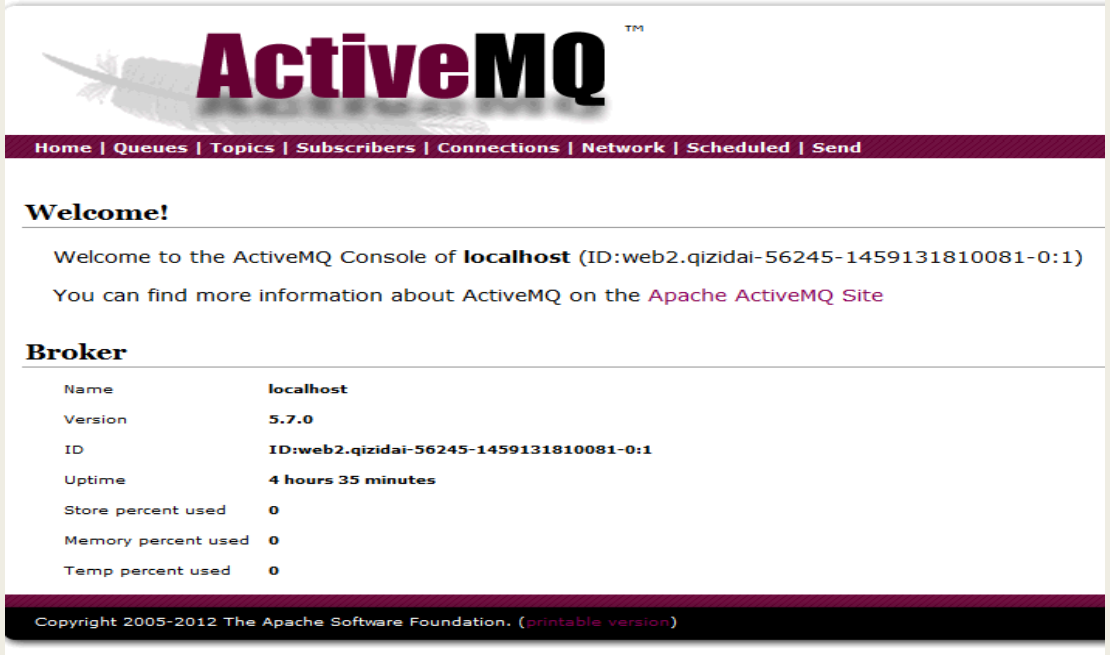
测试

- ActiveMQ的默认端口61616
- 通过netstat 命令查看该端口是否打开，判断是否正确运行

ActiveMQ开发D



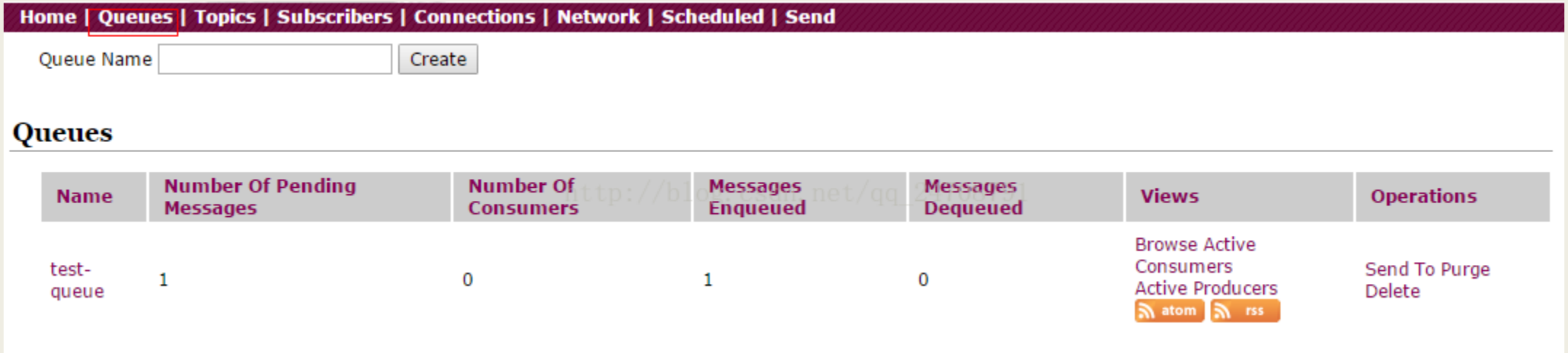
- 输入网址http://ip:8161/adm
  行登录:
- 登录成功页面如下:



主界面


192.168.25.129:8161/admin/
ActiveMQ
Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send
Welcome!
Welcome to the Apache ActiveMQ Console of localhost (ID:localhost.localdomain-41032-1502969174439-0:1)
You can find more information about Apache ActiveMQ on the Apache ActiveMQ Site
Broker
Name localhost
Version 5.12.0
ID ID:localhost.localdomain-41032-1502969174439-0:1
Uptime 58 minutes
Store percent used 0
Memory percent used 0
Temp percent used 0

点对点消息列表


Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send
Queue Name
Create
Queues
Name
Number Of Pending Messages
Number Of Consumers
Messages Enqueued
Messages Dequeued
Views
Operations
test-queue
1
0
1
0
Browse Active Consumers Active Producers
atom
rss
Send To Purge Delete

ActiveMQ开发DEMO

JMS入门-DEMO

- 点对点模式
- 发布/订阅模式

点对点模式Demo-消息生产者客户端

```
//1.创建连接工厂
ConnectionFactory connectionFactory=new ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session（参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建队列对象
Queue queue = session.createQueue("test-queue");
//6.创建消息生产者
MessageProducer producer = session.createProducer(queue);
//7.创建消息
TextMessage textMessage = session.createTextMessage("欢迎来到神奇的消息世界");
//8.发送消息
producer.send(textMessage);
//9.关闭资源
producer.close();
session.close();
connection.close();
```

点对点模式Demo-消息消费者客户端

```
/1.创建连接工厂
ConnectionFactory connectionFactory=new ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session（参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建队列对象
Queue queue = session.createQueue("test-queue");
//6.创建消息消费
MessageConsumer consumer = session.createConsumer(queue);
//7.监听消息
consumer.setMessageListener(new MessageListener() {
public void onMessage(Message message) {
TextMessage textMessage=(TextMessage)message;
try {
System.out.println("接收到消息： "+textMessage.getText());
} catch (JMSException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}}
});
//8.等待键盘输入
System.in.read();
//9.关闭资源
consumer.close();
session.close();
connection.close();
```

测试运行

- 同时开启2个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现只有一个消费者会接收到消息

发布/订阅模式Demo-生产者

```
/1.创建连接工厂
ConnectionFactory connectionFactory=new ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session（参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建主题对象
Topic topic = session.createTopic("test-topic");
//6.创建消息生产者
MessageProducer producer = session.createProducer(topic);
//7.创建消息
TextMessage textMessage = session.createTextMessage(“欢迎来到神奇的消息世界");
//8.发送消息
producer.send(textMessage);
//9.关闭资源
producer.close();
session.close();
connection.close();
```

```
//1.创建连接工厂
ConnectionFactory connectionFactory=new ActiveMQConnectionFactory("tcp://192.168.25.135:61616");
//2.获取连接
Connection connection = connectionFactory.createConnection();
//3.启动连接
connection.start();
//4.获取session（参数1：是否启动事务,参数2：消息确认模式)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
//5.创建主题对象
//Queue queue = session.createQueue("test-queue");
Topic topic = session.createTopic("test-topic");
//6.创建消息消费
MessageConsumer consumer = session.createConsumer(topic);
//7.监听消息
consumer.setMessageListener(new MessageListener() {  public void onMessage(Message message) {
TextMessage textMessage=(TextMessage)message;try {System.out.println("接收到消息：  "+textMessage.getText());
} catch (JMSException e) {// TODO Auto-generated catch block
e.printStackTrace();}}});
//8.等待键盘输入
System.in.read();
//9.关闭资源
consumer.close();
session.close();
connection.close();
```

运行测试

- 同时开启2个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现每个消费者会接收到消息。