1、快速入门

1.1、前言

> MQ 理解

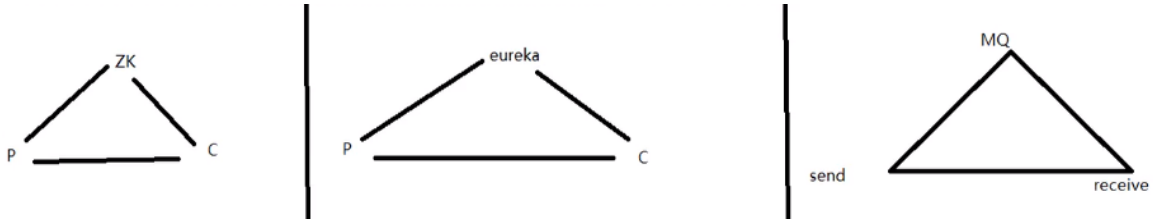
MQ = Message Queue = 消息队列 = 消息中间件

我们将其分为两个词理解：

1、消息理解

- 微信、短信、语言、漂流瓶.....
- 是消息就一定有一个 发送者 和一个 接收者

2、中间件理解



> MQ 的产品种类

天上飞的理念必定有落地的实现，MQ 这么多怎么学?

任正非先生说过一句话：对着一个城墙口发起冲锋，总分总研究学习，找到学习方法总结经验，内化吸收!

MQ 产品 种类:

- Kafka
- RabbitMQ
- RocketMQ
- ActiveMQ （学会它，然后举一反三）
    - API 发送和接收
    - MQ 的高可用性
    - MQ 的集群和容错配置
    - MQ 的持久化
    - 延时发送、定时投递
    - 签收机制
    - 整合Spring、SpringBoot

1.2、为什么要MQ

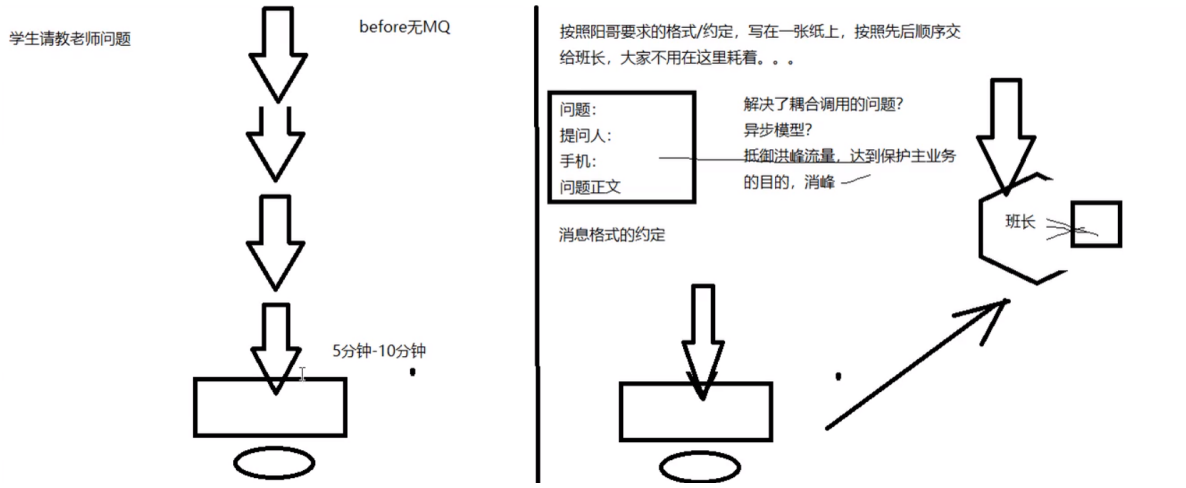
> 常见面试题：你不会说，就说明你对其不够理解！所以底子很重要！

1、在何种场景下使用了消息中间件？

2、为什么要在系统里引入消息中间件？

3、引入了消息中间件有哪些好处？又有哪些坏处？你是如何避免的！

> 生活中的例子



MQ 能干嘛?

- 解耦
- 消峰
- 异步

> 为什么要引入MQ，问题的产生背景

系统之间直接调用在实际工程中存在的问题?

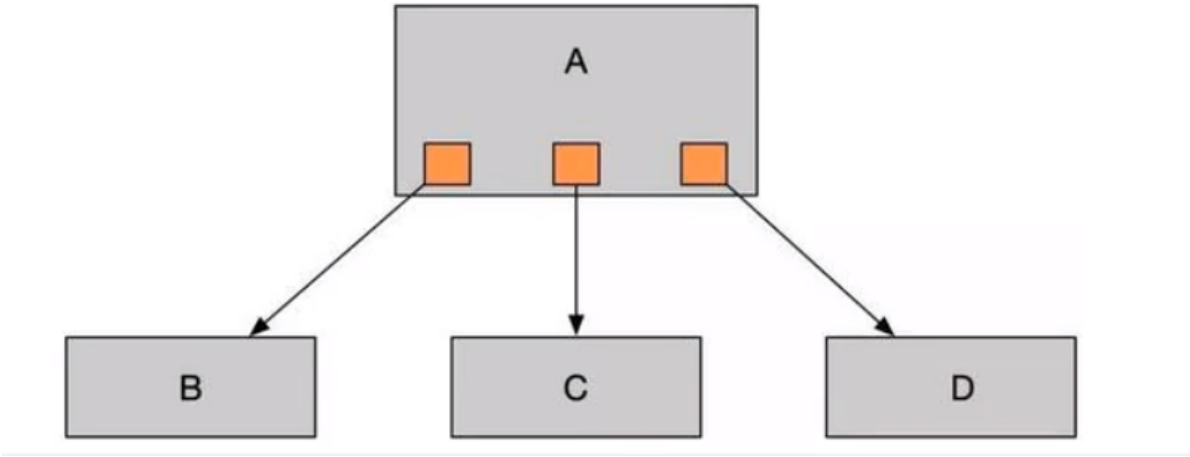
微服务架构后，链式调用是我们在写程序的时候的一般流程，为了完成一个整体功能会将其拆分成多个函数（或子模块），比如模块A调用模块B，模块B调用模块C，模块C调用模块D，但在大型分布式应用中，系统间的RPC交互繁杂，一个功能背后要调用上百个接口并非不可能，从单机架构过渡到分布式微服务架构的通例，这种架构会有哪些问题?

1、系统之间接口耦合比较严重

每新增一个下游功能，都要对上游的相关接口进行改造。比如 系统A 要发送数据给系统B 和 C，发送给每个系统的数据可能有差异，因此系统A对要发送给每个系统的数据进行了组装，然后逐一发送；、

当代码上线后又新增了一个需求：

把数据也发给D，新上了一个D系统也要接受A系统的数据。此时就需要修改A系统，让他感知到D的存在。同时把数据处理好再给D。这个过程中你会看到，每接入一个下游系统，都要对A系统进行代码改造，开发联调的效率很低。



2、面对大流量并发时，容易被冲垮

每个接口模块的吞吐能力是有限的，这个上限能力比作是堤坝，当大流量（洪水）来临时，容易被冲垮。

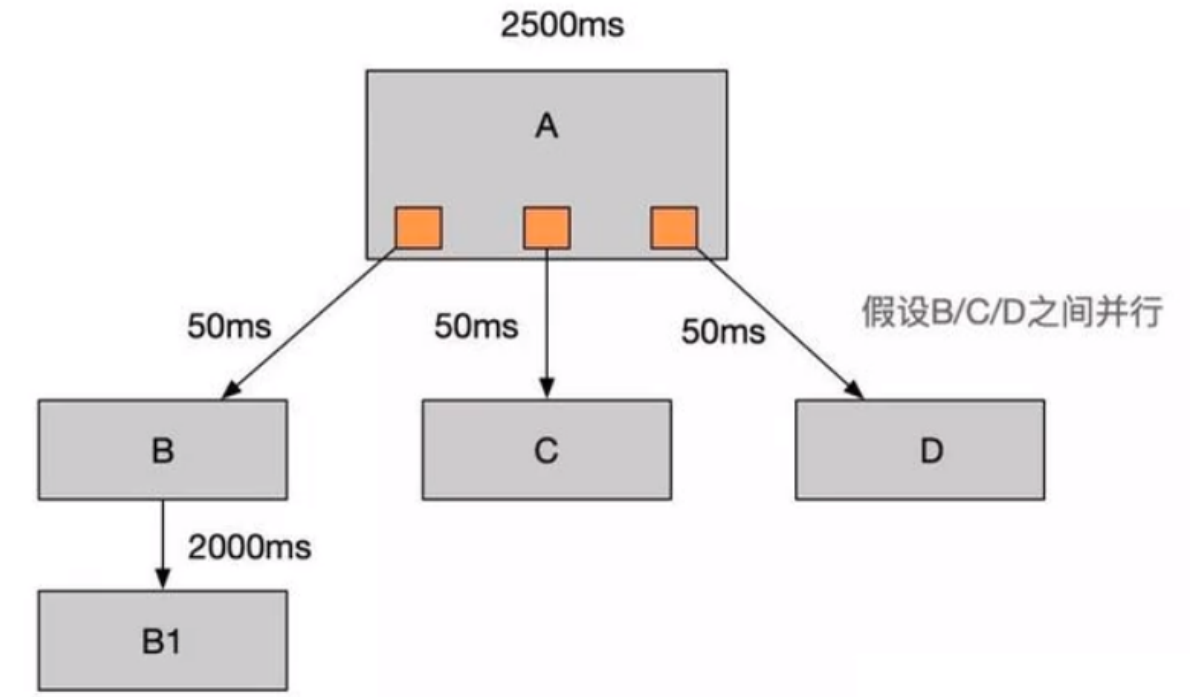
比如秒杀业务：

上游系统发起下单购买操作，我就是下单一个操作，下游系统完成秒杀业务逻辑（读取订单，库存检查，库存冻结，余额检查，余额冻结，订单生成，余额扣减，库存扣减，生成流水，余额解冻，库存解冻）

3、等待同步存在性能问题

RPC接口基本上是同步调用，整体的服务性能遵循 “木桶理论”，即整体系统的耗时取决于链路中最慢的那个接口。

比如A调用B/C/D都是50ms，但此时B又调用了B1，花费2000ms，那么直接就拖累了整个服务性能。



需要有一种技术能够摆平上述情况：

1、要做到系统解耦，当新的模块接进来时，可以做到代码改动最小；**能够解耦**

2、设置流量缓冲池，可以让后端系统按照自身吞吐能力进行消费，不被冲垮； **能够削峰**

3、强弱依赖梳理能将非关键调用链路的操作异步化并提升整体系统的吞吐能力；**能够异步**

1.3、是什么

> 定义

面向消息的中间件（message-oriented middleware）MOM 能够很好的解决以上问题。

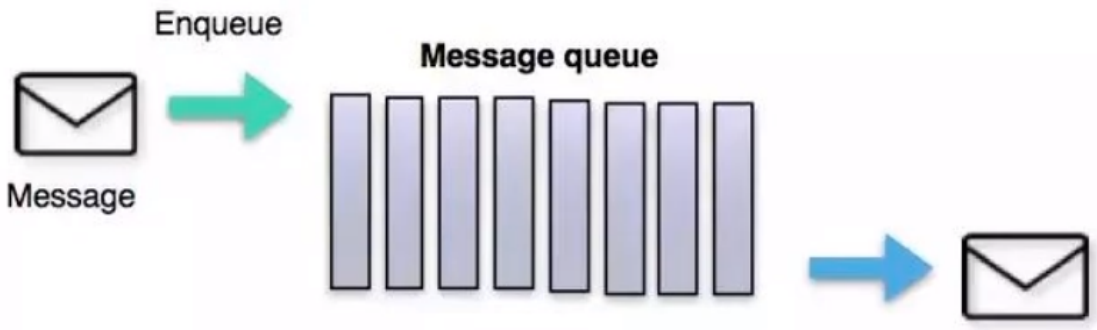
是指利用高效可靠的消息传递机制进行与平台无关的数据交流，并基于数据通信来进行分布式系统的集成。

通过提供**消息传递**和**消息排队**模型在分布式环境下提供应用解耦、弹性伸缩、冗余存储、流量削峰、异步通信、数据同步等功能。

大致的过程如下：

发送者把消息发送给消息服务器，消息服务器将消息存放在若干队列/主题 中，在合适的时候，消息服务器会将消息转发给接受者。在这个过程中，发送和接收是异步的，也就是发送无需等待，而且发送者和接收者的生命周期也没有必然关系。

尤其在发布pub/订阅sub模式下，也可以完成一对多的通信，即让一个消息 有多个接受者。



> 特点

采用异步处理模式

消息发送者可以发送一个消息而无需等待响应。消息发送者将消息发送到一条虚拟的通道（队列、主题）上；

消息接收者则订阅或监听该通道。一条信息可能最终转发给一个或多个消息接收者，这些接收者都无需对消息发送者做出同步回应。整个过程都是异步的。

例子：

也就是说，一个系统跟另外一个系统之间进行通信的时候，假如系统A希望发送一个消息给系统B，让他去处理。

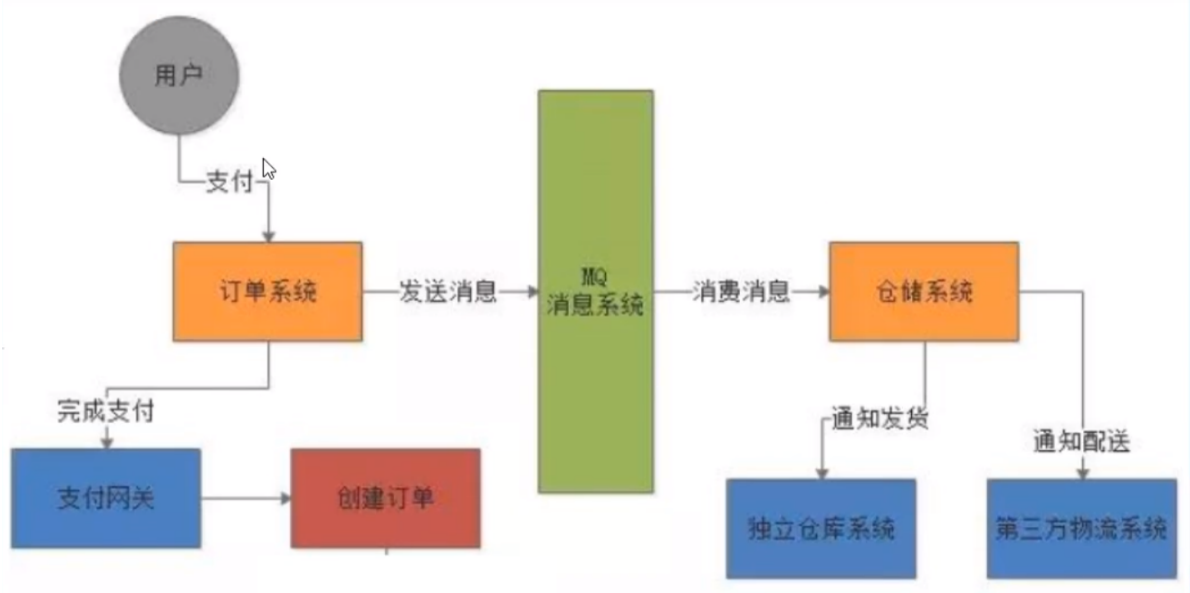
但是系统A不关注系统B到底怎么处理或者有没有处理好，所以系统A把消息发送给 MQ，然后就不管这条消息的 死活的，接着系统B从MQ里消费出来处理即可。至于怎么处理，是否处理完毕，什么时候处理，都是系统B的事儿，与系统A无关。

这样的一种通信方式，就是所谓的 “异步” 通信方式，对于系统A来说，只要把消息发给MQ，然后系统B就会异步的去进行处理了，系统A 不需要 “同步” 的等待系统B处理完。这样的好处是什么呢？两个字 “解耦”。

应用系统之间解耦合

发送者和接收者不必了解对方，只需要确认消息；

发送者和接收者不必同时在线。



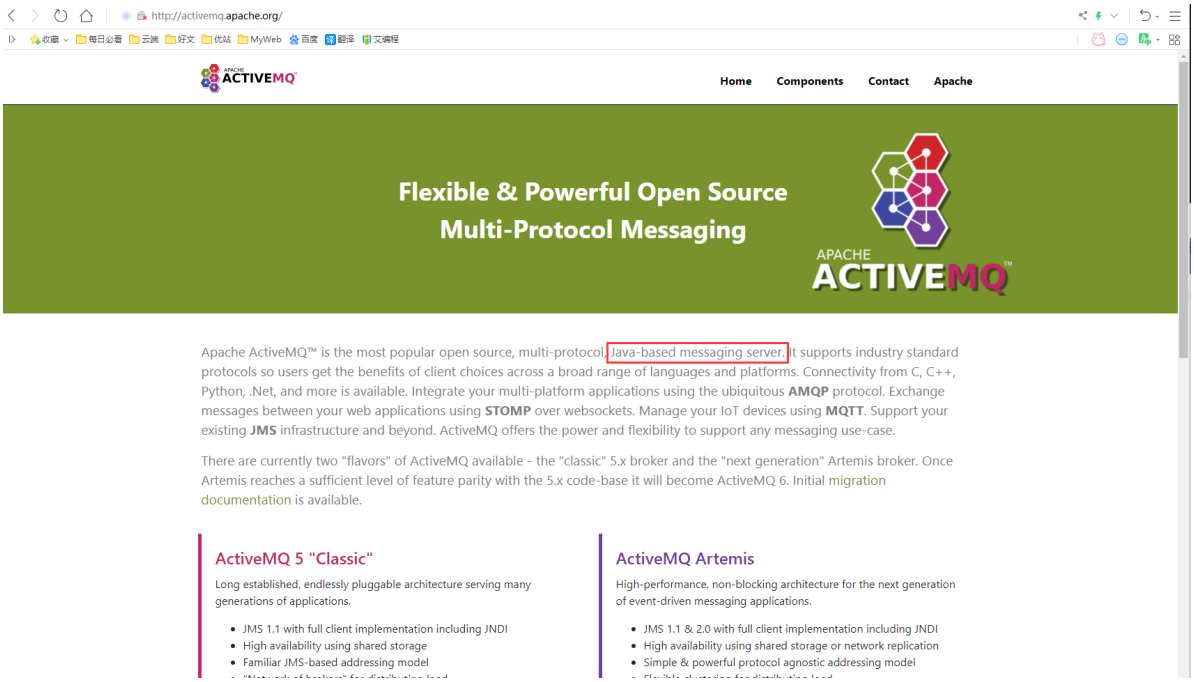
1.4、下载

> 如何使用操作

- 实现高可用、高性能、可伸缩、易用和安全的企业级面向消息服务的系统
- 异步消息的消费和处理
- 控制消息的消费顺序
- 可以和Spring、SpringBoot 整合简化编码
- 配置集群容错的MQ集群
- .....

下载

官网地址：http://activemq.apache.org/

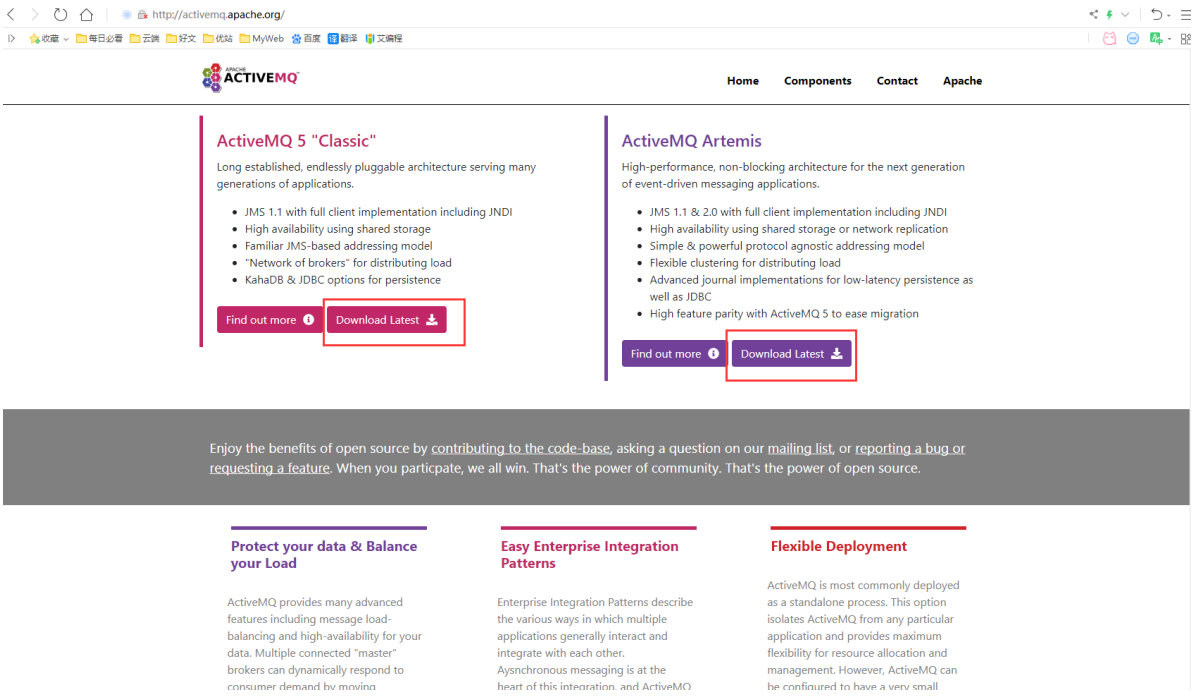


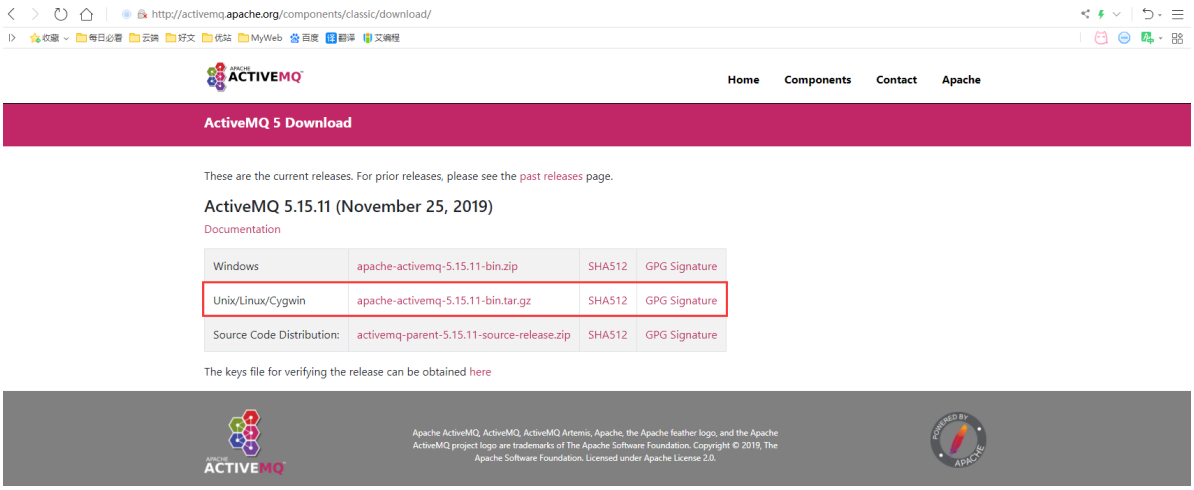
翻译：

Apache ActiveMQ™是最流行的开源，多协议，基于Java的消息传递服务器。它支持行业标准协议，因此用户可以通过广泛的语言和平台从客户选择中受益。可以使用C，C ++，Python，.Net等进行连接。使用无处不在的**AMQP**协议集成您的多平台应用程序。在Websocket上使用**STOMP**在Web应用程序之间交换消息。使用**MQTT**管理您的IoT设备。支持您现有的**JMS**基础结构及其他。ActiveMQ提供了强大的功能和灵活性来支持任何消息传递用例。

ActiveMQ当前有两种“风味”可用-“经典” 5.x代理和“下一代” Artemis代理。一旦Artemis与5.x代码库达到足够的功能奇偶校验级别，它将变为ActiveMQ6。

点击下方下载即可：





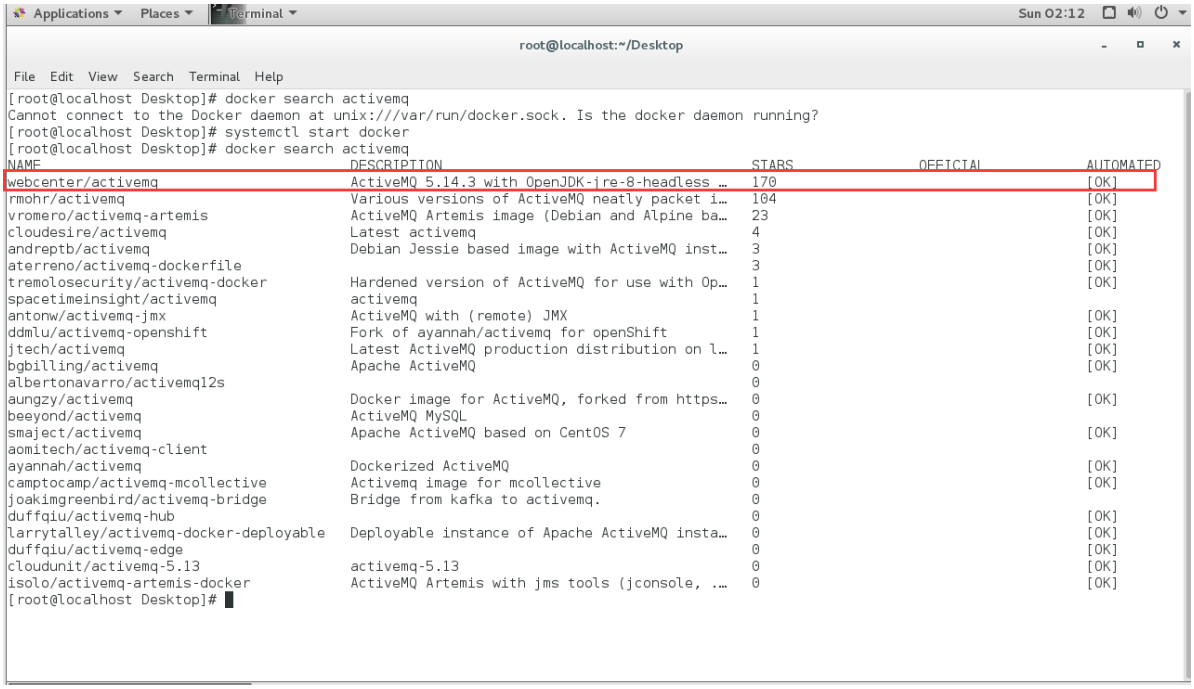
下载Linux版本！我们这里使用 Docker 安装！

1.5、安装ActiveMQ

我们这里使用 Docker 安装！

1、启动Docker `systemctl start docker` 停止Docker `systemctl stop docker`

2、搜索 ActiveMQ 镜像 `docker search activemq`



3、获取 ActiveMQ 镜像 `docker pull webcenter/activemq`

4、查看本地镜像 `docker image`

```
[root@localhost Desktop]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              fce289e99eb9        13 months ago       1.84kB
webcenter/activemq  latest              3af156432993        3 years ago         422MB
```

5、docker 启动 ActiveMQ 命令

`docker run -d --name activemq -p 61616:61616 -p 8161:8161 webcenter/activemq`

```
[root@localhost Desktop]# docker run -d --name activemq -p 61616:61616 -p 8161:8161 webcenter/activemq
39df41b9bf5a280b0f2fbf7fa6c44e7efdb5f98189dfab15e28c1921cfb70664
```

- 61616是 activemq 的容器使用端口，提供JMS服务（映射为61616）
- 8161是 web 页面管理端口（对外映射为8161）

`docker stop 容器id` 停止

6、使用 docker ps 查看 ActiveMQ 已经运行了 `docker ps -a`

```
[root@localhost Desktop]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS
                                                NAMES
39df41b9bf5a        webcenter/activemq  "/app/run.sh"       About a minute ago  Up About a minute           1883/tcp, 5672/tcp, 0.0.0.0:8161->8161/tcp, 61613-61614/tcp, 0.0.0.0:61616->61616/tcp   activemq
b96a5154052c        hello-world         "/hello"            12 days ago         Exited (0) 12 days ago
                                                loving_mclaren
[root@localhost Desktop]#
```

7、使用 `docker exec -it activemq /bin/bash` 进入 ActiveMQ， `exit`退出

```
[root@localhost Desktop]# docker exec -it activemq /bin/bash
root@39df41b9bf5a:/opt/activemq# ks
bash: ks: command not found
root@39df41b9bf5a:/opt/activemq# ls
LICENSE  README.txt               bin   conf.tmp  docs      lib  webapps
NOTICE   activemq-all-5.14.3.jar  conf  data      examples  tmp  webapps-demo
root@39df41b9bf5a:/opt/activemq# exit
exit
[root@localhost Desktop]#
```

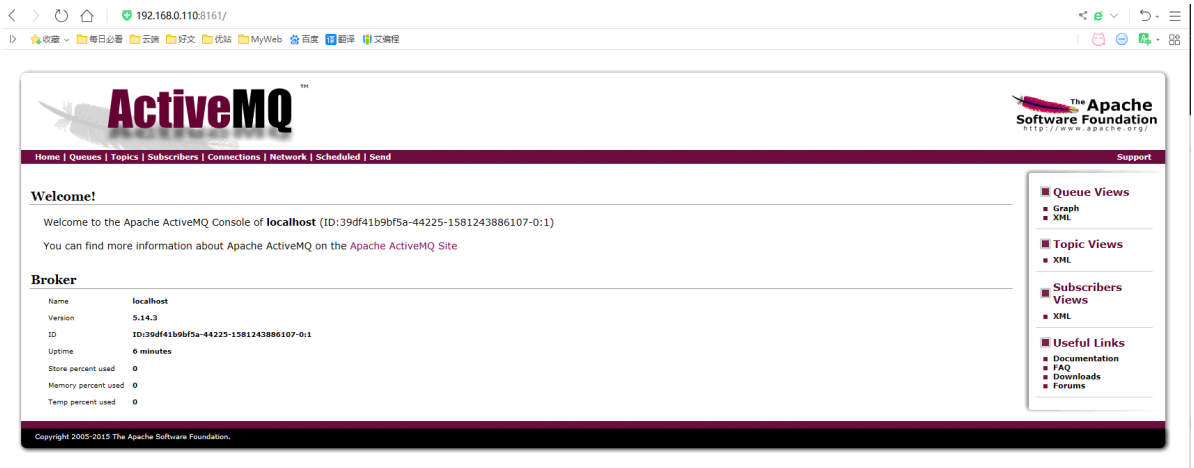
8、Linux开启端口号供外界访问

```
#查看防火墙状态
firewall-cmd --state

#开启防火墙
systemctl start firewalld.service

#开启61616端口
firewall-cmd --zone=public --add-port=61616/tcp --permanent
firewall-cmd --zone=public --add-port=8161/tcp --permanent

解释这个命令：
--zone=public：表示作用域为公共的；
--add-port=8161/tcp：添加tcp协议的端口8161；
--permanent：永久生效，如果没有此参数，则只能维持当前服务生命周期内，重新启动后失效；

# 重启防火墙
systemctl restart firewalld.service

# 输入命令重新载入配置
firewall-cmd --reload
#查看开启的端口列表
firewall-cmd --permanent --list-port
```

```
[root@localhost Desktop]# firewall-cmd --state
running
[root@localhost Desktop]# firewall-cmd --zone=public --add-port=61616/tcp --permanent
success
[root@localhost Desktop]# firewall-cmd --zone=public --add-port=8161/tcp --permanent
success
[root@localhost Desktop]# systemctl restart firewalld.service
[root@localhost Desktop]# firewall-cmd --reload
success
[root@localhost Desktop]# firewall-cmd --permanent --list-port
61616/tcp 8161/tcp 8080/tcp
[root@localhost Desktop]#
```

测试访问控制台：

192.168.0.110:8161/admin　　默认账号密码、admin-admin



2、上手使用

2.1、标准API

> 基础测试项目构建

1、新建一个Maven项目 activemq-study

2、导入对应的pom依赖

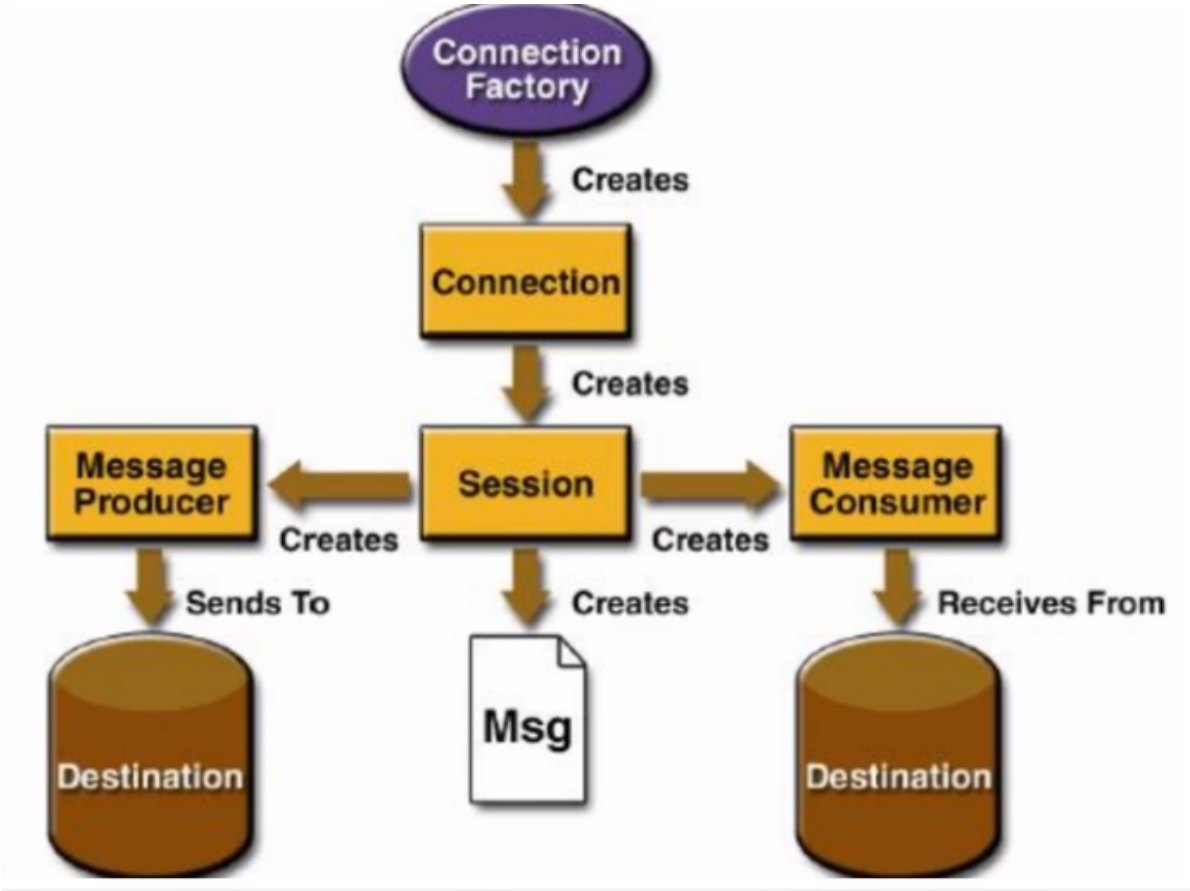
```
<!-- 根据版本下载 -->
<dependency>
```

```
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-all</artifactId>
    <version>5.14.3</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.xbean/xbean-spring -->
<dependency>
    <groupId>org.apache.xbean</groupId>
    <artifactId>xbean-spring</artifactId>
    <version>4.14</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.8</version>
</dependency>
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.30</version>
</dependency>
```

> JMS 编码总体架构

JMS 百度百科：什么是 JMS

JMS 编码总体架构：



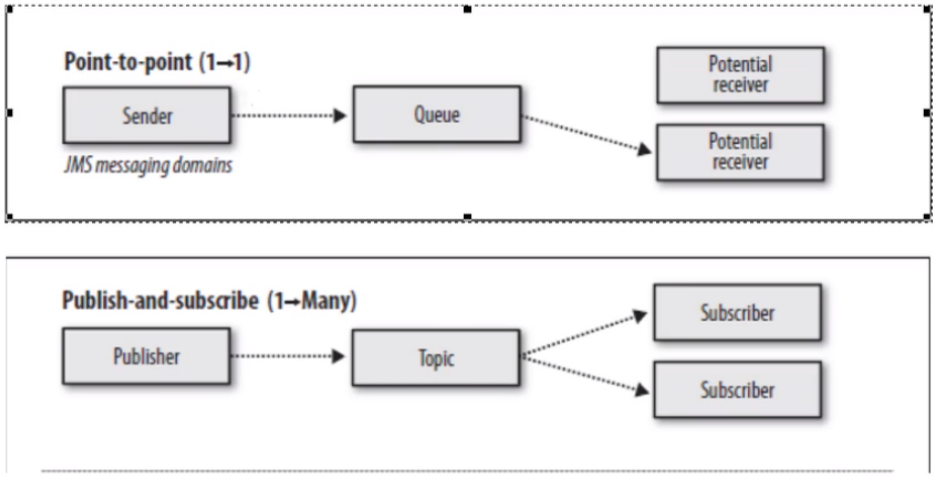
回顾DBC的编码套路：

第1步：注册驱动 (仅仅做一次)	Class.forName(“com.mysql.jdbc.Driver”);
第2步：建立连接(Connection)	Connection conn =DriverManager.getConnection(url, user, password);
第3步：创建运行SQL的语句(Statement)	Statement st = connection.createStatement();
第4步：运行语句	ResultSet rs =st.executeQuery(sql);
第5步：处理运行结果(ResultSet)	省略.....
第6步：释放资源	省略.....

> 粗说目的地 Destination 队列和主题

消息队列包括两种模式，点对点模式（point to point， queue）和发布/订阅模式（publish/subscribe，topic）。

两大模式特性：



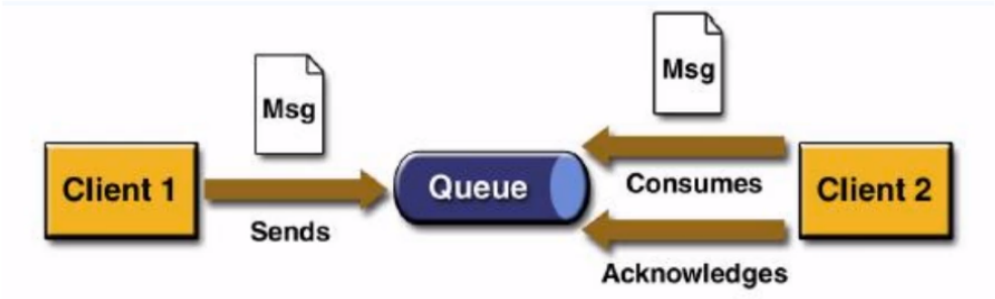
2.2、队列

在点对点的消息传递域中，目的地被称为队列（Queue）

1、每个消息只能有一个消费者，类似1对1的关系。好比个人快递自己领取自己的。

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，消费者都可以提取消息。好比我们的发送短信，发送者发送后不见得接收者会即收即看。

3、消息被消费后队列中不会再存储，所以消费者不会消费到已经被消费掉的消息。

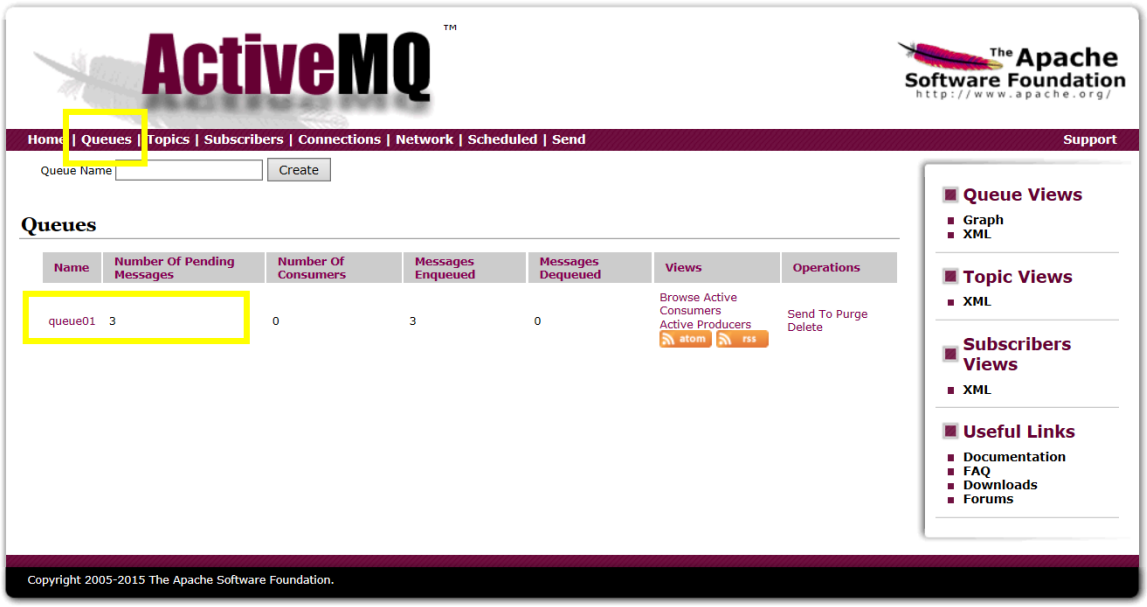


1、消息生产者

```
1  package com.kuang.activemq.queue;
2
3  import org.apache.activemq.ActiveMQConnectionFactory;
4
5  import javax.jms.*;
6
7  /**
8   * @author 秦疆 24736743@qq.com
9   */
10 public class JmsProduce {
11
12     public static final String ACTIVEMQ_URL = "tcp://192.168.0.110:61616";
13
14     public static void main(String[] args) throws JMSException {
```

```
        // 1、创建连接工厂，分析ActiveMQConnectionFactory得到参数!
        //    使用默认的用户名和密码
        ActiveMQConnectionFactory factory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);

        // 2、通过连接工厂获得 Connection ，然后连接访问
        Connection connection = factory.createConnection();
        connection.start();

        // 3、创建会话Session
        // 参数：事务、签收  默认：AUTO_ACKNOWLEDGE 自动确认
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

        // 4、创建目的地（具体是队列还是topic）
        Queue queue = session.createQueue("queue01");

        // 5、创建消息的生产者
        MessageProducer producer = session.createProducer(queue);

        // 6、通过 producer 生产3条消息发送到MQ的队列中
        for (int i = 1; i <= 3; i++) {
            // 7、创建消息
            TextMessage textMessage = session.createTextMessage("msg=>" +
i);
            // 8、通过生产者发布
            producer.send(textMessage);
        }

        // 9、关闭资源
        producer.close();
        session.close();
        connection.close();

        System.out.println("消息发布到MQ完成");

    }
}
```

发布完成之后，我们可以去控制台查看是否成功!



控制台说明：

Number Of Pending Messages	等待消费的消息	这个是当前未出队列的数量。总接收数-总出队列数
Number Of Consumers	消费者数量	消费者端的消费者数量
Messages Enqueued	进队消息数	进入队列的总数量，包括出队列的，只增不减
Messages Dequeued	出队消息数	可以理解为是消费者消费掉的数量

总结：

当有一个消息进入这个队列时，等待消费的消息是1，进入队列的消息是1。

当消息消费后，等待消费的消息是0，进入队列的消息是1，出队列的消息是1。

再来一条消息时，等待消费的消息是1，进入队列的消息就是2。

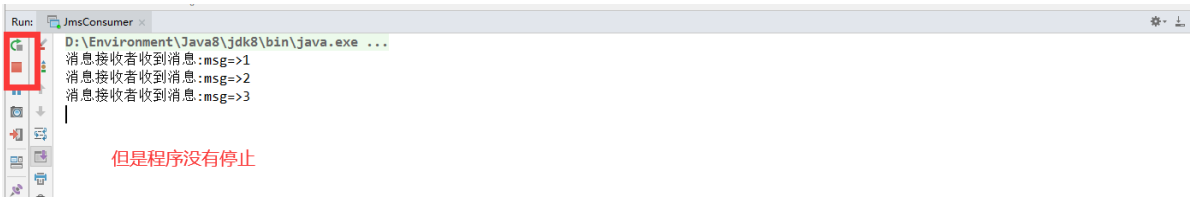
2、消息消费者-1

通过接受消息进行消费

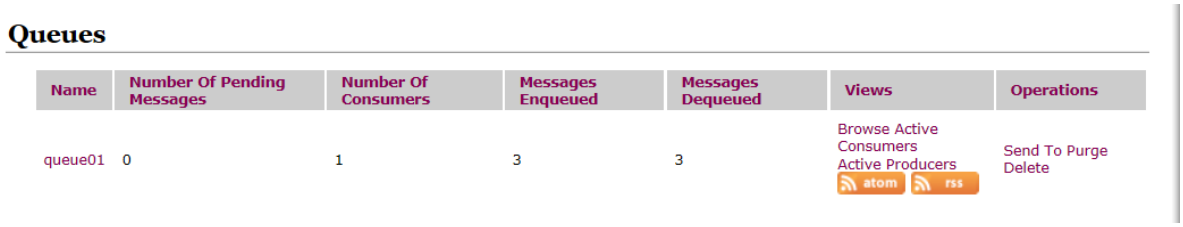
```
package com.kuang.activemq.queue;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;

/**
 * @author 秦疆 24736743@qq.com
 */
public class JmsConsumer {

    public static final String ACTIVEMQ_URL = "tcp://192.168.0.110:61616";

    public static void main(String[] args) throws JMSException {
        // 1、创建连接工厂，分析ActiveMQConnectionFactory得到参数！
        //    使用默认的用户名和密码
        ActiveMQConnectionFactory factory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);

        // 2、通过连接工厂获得 Connection ，然后连接访问
        Connection connection = factory.createConnection();
        connection.start();

        // 3、创建会话Session
        // 参数：事务、签收  默认：AUTO_ACKNOWLEDGE 自动确认
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

        // 4、创建目的地（具体是队列还是topic）
        Queue queue = session.createQueue("queue01");

        // 5、创建消息的消费者 consumer
        MessageConsumer consumer = session.createConsumer(queue);
```

```
32
33          while (true){
34              // consumer.receive(); 一直等
35              // consumer.receive(long time); 超时就不等了
36              // 这里消费的类型格式一定要正确，和生产的类型一一对应！
37              TextMessage receive = (TextMessage) consumer.receive();
38              if (receive!=null){
39                  System.out.println("消息接收者收到消息:"+receive.getText());
40              }else {
41                  break;
42              }
43          }
44
45          // 9、关闭资源
46          consumer.close();
47          session.close();
48          connection.close();
49
50          System.out.println("消息接收完毕");
51
52      }
53  }
```

成功获取数据：但是程序没有停止！



去查看web控制台的数据变化：可以发现消费者一直有一个，因为我们的程序没有停止！停止程序后则为0；



receive 方法说明：

```
1  // consumer.receive(); 一直等
2  // consumer.receive(long time); 超过多少时间没有消费就结束不等了，单位ms
```

3、消息消费者-2

通过监听器的方式来消息消费

```
1  package com.kuang.activemq.queue;
2
3  import lombok.SneakyThrows;
4  import org.apache.activemq.ActiveMQConnectionFactory;
5
```

```java
import javax.jms.*;
import java.io.IOException;

/**
 * @author 秦疆 24736743@qq.com
 */
public class JmsConsumer2 {

    public static final String ACTIVEMQ_URL = "tcp://192.168.0.110:61616";

    public static void main(String[] args) throws JMSException, IOException {
        // 1、创建连接工厂，分析ActiveMQConnectionFactory得到参数!
        //    使用默认的用户名和密码
        ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(ACTIVEMQ_URL);

        // 2、通过连接工厂获得 Connection ，然后连接访问
        Connection connection = factory.createConnection();
        connection.start();

        // 3、创建会话Session
        // 参数：事务、签收  默认：AUTO_ACKNOWLEDGE 自动确认
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

        // 4、创建目的地（具体是队列还是topic）
        Queue queue = session.createQueue("queue01");

        // 5、创建消息的消费者 consumer
        MessageConsumer consumer = session.createConsumer(queue);

        // setMessageListener，通过监听器消费,异步非阻塞
        consumer.setMessageListener(new MessageListener() {
            public void onMessage(Message message) {
                if (message!=null && message instanceof TextMessage){
                    TextMessage textMessage = (TextMessage)message;
                    try {
                        System.out.println(textMessage.getText());
                    } catch (JMSException e) {
                        e.printStackTrace();
                    }
                }
            }
        });

        // System.in.read(); 阻塞等待

        // 关闭资源
        consumer.close();
        session.close();
        connection.close();

        System.out.println("消息接收完毕");

    }
}
```

4、消费者3大问题

1、先生产消息，只启动1号消费者，1号消费者可以消费吗？ Y

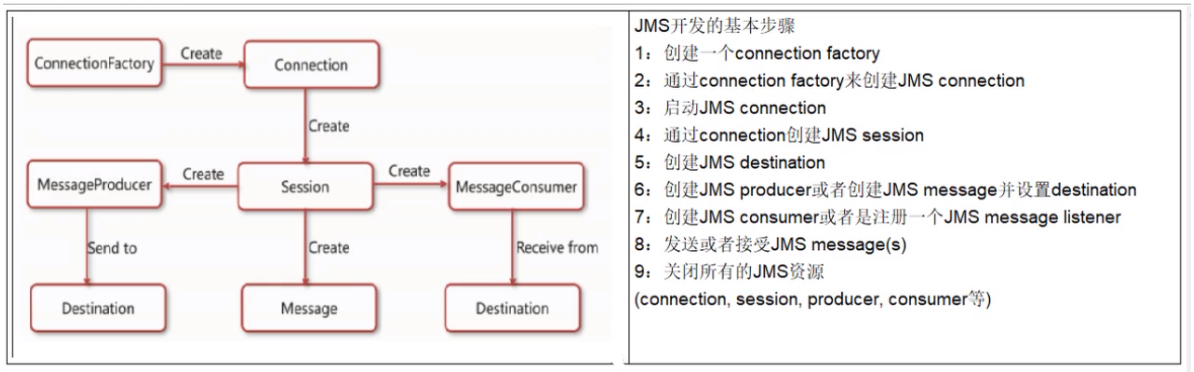
2、先生产消息，先启动1号消费者，再启动2号消费者。2号消费者可以消费吗？ N

3、先启动两个消费者，再生产6个消息，请问，消费情况如何？ 两个消费者一人一半

5、小结

JMS 开发的基本步骤：

JMS开发的基本步骤



两种消费方式：

1、同步阻塞式：consumer.reveive()

2、异步非阻塞式：监听器 consumer.setMessageListener

2.3、主题

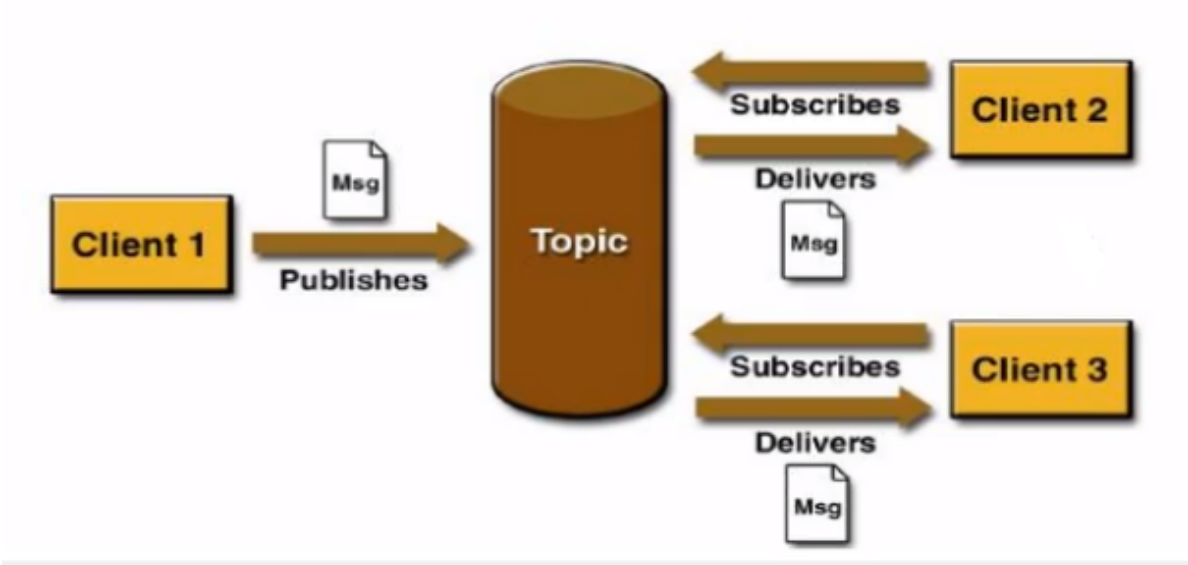
在发布订阅消息传递域中，目的地被称为主题（Topic）

1、生产者将消息发布到Topic中，每个消息可以有多个消费者，属于 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只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落地，假如无人订阅就去生产，那就是一条废消息，所以，一般先启动消费者在启动生产者。

JMS 规范允许客户创建持久订阅，这在一定程度上放松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。一句话，好比我们的微信公众号订阅



1、生产者

和队列的是一样的，只是创建的对象不同！ `session.createTopic`

```
package com.kuang.activemq.topic;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;

/**
 * @author 秦疆 24736743@qq.com
 * 发布即生产者
 */
public class JmsProduceTopic {

    public static final String ACTIVEMQ_URL = "tcp://192.168.0.110:61616";

    public static void main(String[] args) throws JMSException {
        // 1、创建连接工厂，分析ActiveMQConnectionFactory得到参数!
        //    使用默认的用户名和密码
        ActiveMQConnectionFactory factory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);

        // 2、通过连接工厂获得 Connection ，然后连接访问
        Connection connection = factory.createConnection();
        connection.start();

        // 3、创建会话Session
        // 参数：事务、签收  默认：AUTO_ACKNOWLEDGE 自动确认
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

        // 4、创建目的地（具体是队列还是topic）
        Topic topic = session.createTopic("topic-kuangshen");

        // 5、创建消息的生产者
        MessageProducer producer = session.createProducer(topic);

```

```
        // 6、通过 producer 生产3条消息发送到MQ的队列中
        for (int i = 1; i <= 3; i++) {
            // 7、创建消息
            TextMessage textMessage = session.createTextMessage("topic-msg=>" + i);
            // 8、通过生产者发布
            producer.send(textMessage);
        }

        // 9、关闭资源
        producer.close();
        session.close();
        connection.close();

        System.out.println("topic消息发布到MQ完成");

    }
}
```

2、消费者

和队列的是一样的，只是创建的对象不同！ `session.createTopic`

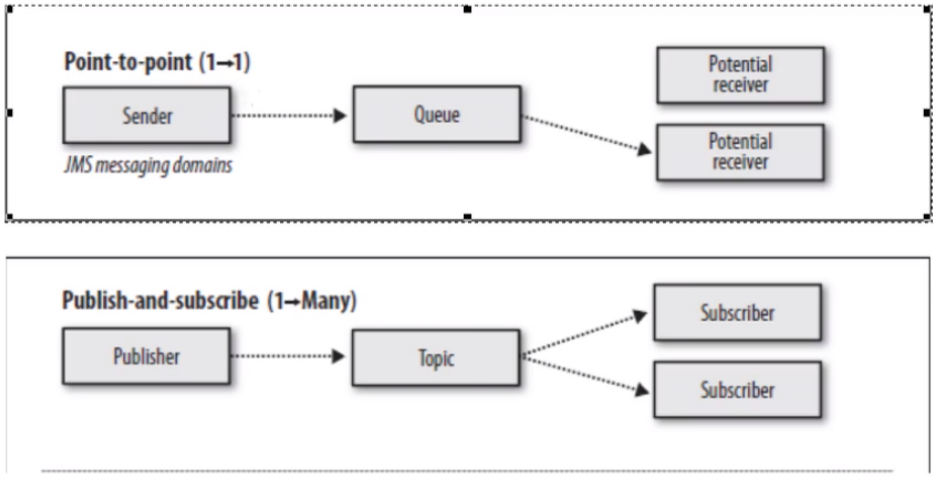
```
package com.kuang.activemq.topic;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;
import java.io.IOException;

/**
 * @author 秦疆 24736743@qq.com
 * 订阅即消费者
 */
public class JmsConsumerTopic {

    public static final String ACTIVEMQ_URL = "tcp://192.168.0.110:61616";

    public static void main(String[] args) throws JMSException, IOException {
        // 1、创建连接工厂，分析ActiveMQConnectionFactory得到参数！
        //    使用默认的用户名和密码
        ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(ACTIVEMQ_URL);

        // 2、通过连接工厂获得 Connection ，然后连接访问
        Connection connection = factory.createConnection();
        connection.start();

        // 3、创建会话Session
        // 参数：事务、签收  默认：AUTO_ACKNOWLEDGE 自动确认
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

        // 4、创建目的地（具体是队列还是topic）
```

```
30          Topic topic = session.createTopic("topic-kuangshen");
31
32          // 5、创建消息的消费者 consumer
33          MessageConsumer consumer = session.createConsumer(topic);
34
35          // setMessageListener，通过监听器消费
36          consumer.setMessageListener((message) -> {
37              if (message!=null && message instanceof TextMessage){
38                  TextMessage textMessage = (TextMessage)message;
39                  try {
40                      System.out.println("接收到Topic消
    息："+textMessage.getText());
41                  } catch (JMSException e) {
42                      e.printStackTrace();
43                  }
44              }
45          });
46
47          System.in.read(); // 阻塞等待
48
49          // 关闭资源
50          consumer.close();
51          session.close();
52          connection.close();
53
54          System.out.println("消息接收完毕");
55
56      }
57
58  }
```

注意启动顺序：先启动订阅再启动发布，不然发送的消息是废消息，可以启动多个消费者

注意启动顺序：先启动订阅再启动发布，不然发送的消息是废消息，可以启动多个消费者

3、两大模式对比



比较项目	Topic 模式队列	Queue 模式队列
工作模式	“订阅-发布”模式，如果当前没有订阅者，消息将会被丢弃，如果有多个订阅者，那么这些订阅者都会收到消息。	负载均衡 模式，如果当前没有消费者，消息也不会丢弃，如果有多个消费者，那么一条消息也只会发送给其中一个消费者，并且要求消费者ack信息。
有无状态	无状态	Queue 数据默认会在 mq 服务器上以文件形式保存，比如 Active MQ 一般保存在 $AMQ_HOME\data\kr-store\data下面，也可以配置成DB存储。
传递完整性	如果没有订阅者，消息会被丢弃	消息不会丢弃
处理效率	由于消息要按照订阅者的数量进行复制，所以处理性能会随着订阅者的增加而明显降低，并且还要结合不同消息协议自身的性能差异	由于一条消息只发送给一个消费者，所以就算消费者再多，性能也不会有明显降低。当然不同消息协议的具体性能也是有差异的。

2.4、JMS规范

1、什么是JMS

> 回顾什么是 JavaEE

JavaEE 是一套使用Java进行企业级应用开发的大家一致遵循的13个核心规范工业标准。JavaEE 平台提供了一个基于组件的方法来加快设计、开发、装配及部署企业应用程序。

1、JDBC （Java Database） 数据库连接

2、JNDI （Java Naming and Directory Interfaces）Java的命名和目录接口

3、EJB （Enterprise JavaBean）企业JavaBean

4、RMI（Remote Method Invoke）远程方法调用

5、Java IDL（Interface Description Language）/ CORBA（CommonObject Broker Architecture）接口定义语言、公用对象请求代理程序体系结构

6、JSP （Java Server Pages）

7、Servlet

8、XML （Extensible Markup Language）可扩展的标记语言

9、JMS（Java Message Service） Java消息服务

10、JTA （Java Transaction API）Java 事务 API

11、JTS（Java Transaction Service）Java 事务服务

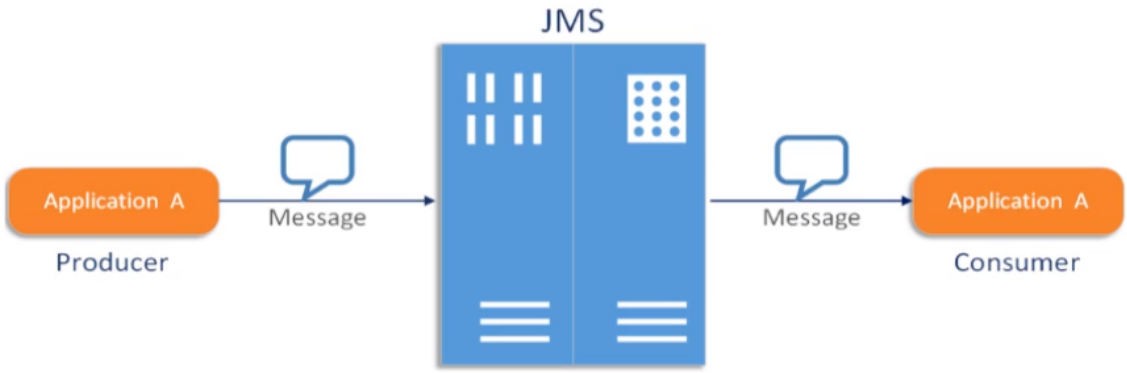
12、JavaMail

13、JAF （JavaBean Activation Framework）

> Java Message Service （Java 消息服务是 JavaEE 中的一个技术）

什么是 Java 消息服务

Java消息服务指的是两个应用程序之间进行异步通信的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应用程序开发。在JavaEE中，当两个应用程序使用JMS进行通信时，他们之间并不是直接相连的。而是通过一个共同的消息收发服务组件关联起来以达到解耦、异步、削峰的效果。



2、各种MQ中间件

消息队列的详细比较：

–	RabbitMQ	ActiveMQ	RocketMQ	Kafka
所属社区/公司	Mozilla Public License	Apache	Ali	Apache
成熟度	成熟	成熟	比较成熟	成熟
授权方式	开源	开源	开源	开源
开发语言	Erlang	Java	Java	Scala&Java
客户端支持语言	官方支持Erlang, Java, Ruby等, 社区产出多种语言API，几乎支持所有常用语言	Java、C、C++、Python、PHP、Perl、.net 等	Java C++（不成熟）	官方支持Java, 开源社区有多语言版本, 如PHP, Python, Go, C/C++, Ruby, NodeJS等编程语言, 详见Kafka 客户端列表
协议支持	多协议支持:AMQP，XMPP, SMTP, STOMP	OpenWire、STOMP、REST、XMPP、AMQP	自己定义的一套(社区提供JMS--不成熟)	自有协议，社区封装了HTTP协议支持
消息批量操作	不支持	支持	支持	支持
消息推拉模式	多协议, Pull/Push均有支持	多协议, Pull/Push均有支持	多协议, Pull/Push均有支持	Pull
HA	master/slave模式, master提供服务, slave仅作备份	基于ZooKeeper + LevelDB 的 Master-Slave 实现方式	支持多Master 模式、多 Master 多 Slave 模式，异步复制模式、多 Master 多 Slave 模式，同步双写	支持replica机制, leader宕掉后, 备份自动顶替，并重新选举leader(基于Zookeeper)
数据可靠性	可以保证数据不丢, 有slave用作备份	master/slave	支持异步实时刷盘，同步刷盘，同步复制，异步复制	数据可靠，并且有replica机制, 有容错容灾能力
单机吞吐量	其次(万级)	最差(万级)	最高(十万级)	次之(十万级)
消息延迟	微秒级	\	比kafka快	毫秒级
持久化能力	内存、文件，支持数据堆积,但数据堆积反过来影响生产速率	内存、文件、数据库	磁盘文件	磁盘文件, 只要磁盘容量够，可以做到无限消息堆积
是否有序	若想有序，只能使用一个Client	可以支持有序	有序	多Client保证有序
事务	不支持	支持	支持	不支持, 但可以通过Low Level API保证仅消费一次
集群	支持	支持	支持	支持
负载均衡	支持	支持	支持	支持
管理界面	较好	一般	命令行界面	官方只提供了命令行版, Yahoo开源自己的Kafka Web管理界面Kafka-Manager
部署方式	独立	独立	独立	独立

3、JMS组成元素

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

2、JMS producer　消息生产者，创建和发送 JMS消息的客户端应用

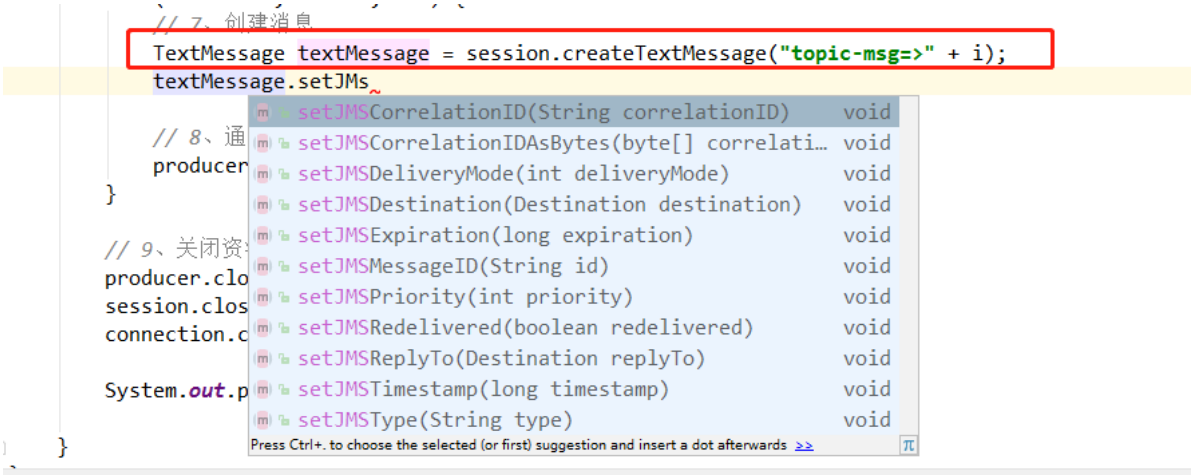
3、JMS consumer　消息消费者，接收和处理 JMS消息的客户端应用

4、JMS message　消息，分为消息头，消息属性，消息体三部分

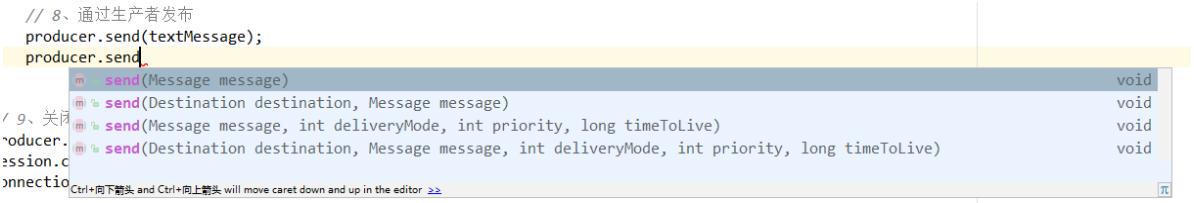
4、Message

> 消息头：不会带大家看全部的，只讲平时用的最多的五大特性！

可以通过setJMS 来定义个属性！



通过send 来发送，可以看到send 方法的重载



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

`JMSDeliveryMode` 持久模式和非持久模式

一条持久性的消息：应该被传送 “一次仅仅一次”，这就意味着如果JMS 提供者出现故障，该消息并不会丢失，它会在服务器恢复之后再次传递消息。

一条非持久的消息：最多会传送一次，这意味着服务器出现故障，该消息将永久丢失。

`JMSExpiration` 可以设置消息在一定时间后过期，默认是永不过期。

消息过期时间，等于Destination的send方法中的timeToLive值加上发送时刻的GMT时间值。

如果timeToLive值等于零，则 JMSExpiration 被设为零，表示该消息永不过期。

如果发送后，在消息过期时间之后消息还没有被发送到目的地，则该消息被清除。

`JMSPriority` 消息优先级

从0~9 十个级别，0~4 是普通消息，5~9是加急消息。

JMS 不要求 MQ 严格按照这十个优先级发送消息，但必须保证加急消息要先于普通消息到达，默认是4级。

`JMSMessageID` 唯一识别每个消息的标识，由MQ产生。

> 消息体 : 封装具体的消息数据，发送和接收的消息体类型必须一致对应!

五种消息体格式:

`TextMessage` 普通字符串消息，包含一个 string。 重点

`MapMessage` 一个Map类型的消息，key为String类型，而值为Java的基本类型。 重点

```java
MapMessage mapMessage = session.createMapMessage();
mapMessage.setString("k1","v1");

// 发送和接收的消息体类型必须一致对应！
```

`BytesMessage` 二进制数组消息，包含一个byte[] 。

`StreamMessage` Java数据流消息，用标准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 对象。

发送和接收的消息体类型必须一致对应！

发送和接收的消息体类型必须一致对应！

> 消息属性

如果需要除消息头字段以外的值，那么可以使用消息属性！

识别、去重、重点标注等操作非常有用的方法！

什么是消息属性：

他们是以属性名和属性值对的形式指定的。可以 将属性是为 消息头得扩展，属性指定一些消息头没有包括的附加信息，比如可以在属性里指定消息选择器。

消息的属性就像可以分配给一条消息的附加消息头一样。他们允许开发者添加有关消息的不透明附加信息。

它们还用于暴露消息选择器在消息过滤时使用的数据。

```java
TextMessage message = session.createTextMessage();
message.setText(text);
message.setStringProperty("username","kuangshen"); // 自定义属性
```

5、消息的可靠性-持久

PERSISTENT : 持久性

> 持久和非持久 队列

队列设置 持久化模式即可！

```java
// 默认就是持久化的！
MessageProducer producer = session.createProducer(topic);

// 非持久化：当服务器宕机，消息不存在。
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

// 持久化：当服务器宕机，消息依旧存在
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
```

持久化消息：

这是队列的默认传送模式，此模式保证这些消息只被传送一次和成功使用一次。对于这些消息，可靠性是优先考虑的因素。

可靠性的另一个重要方面是确保持久性消息传送至目标后，消息服务在向消费者传送他们之前不会丢失这些信息。

> 持久和非持久 Topic

先启动订阅在启动发布！ 类似于微信订阅公众号

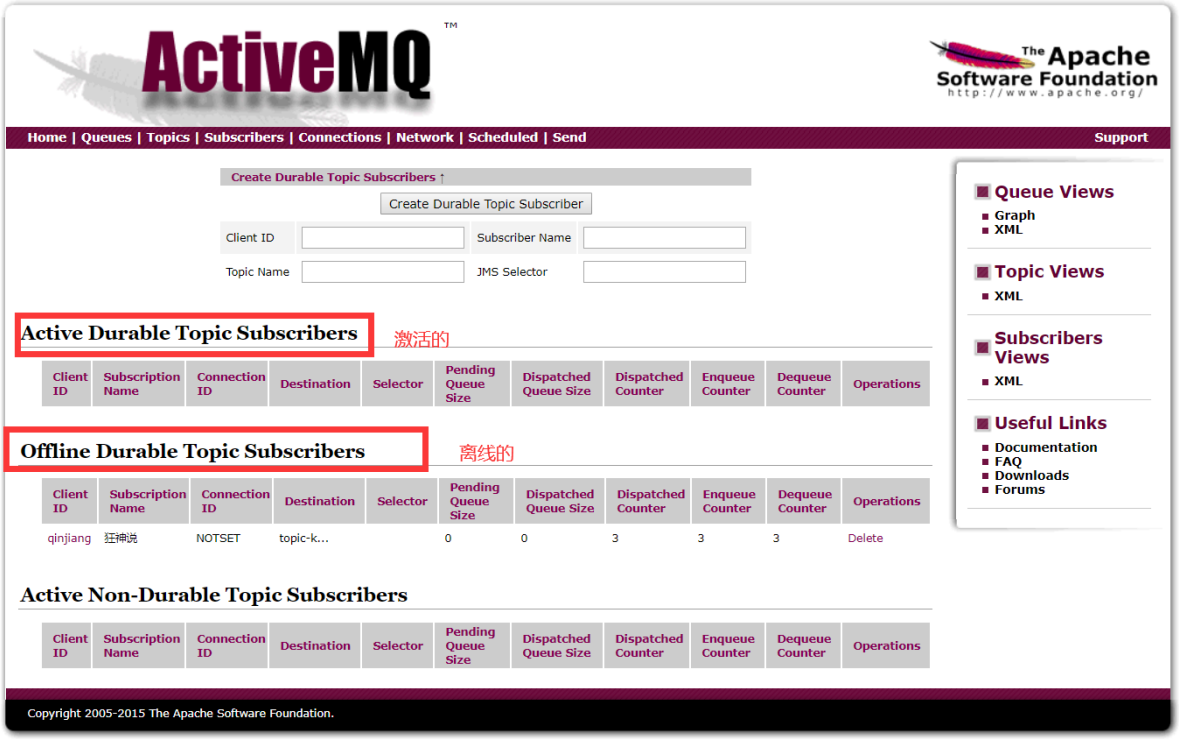
发布方，需要先设定持久化，然后再连接！连接顺序需要变化~

```
package com.kuang.activemq.topic;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;

/**
 * @author 秦疆 24736743@qq.com
 * 发布持久化
 */
public class JmsProduceTopic_Persist {

    public static final String ACTIVEMQ_URL = "tcp://192.168.0.110:61616";

    public static void main(String[] args) throws JMSException {

        ActiveMQConnectionFactory factory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        Connection connection = factory.createConnection();
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
        Topic topic = session.createTopic("topic-kuangshen-Persist");
        MessageProducer producer = session.createProducer(topic);

        // 设置持久化策略
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);
        // ============ 在这里进行连接 ===============
        connection.start();


        for (int i = 1; i <= 3; i++) {
            TextMessage textMessage = session.createTextMessage("topic-
msg=>" + i);
            producer.send(textMessage);
        }

        producer.close();
        session.close();
        connection.close();

        System.out.println("topic消息发布到MQ完成");

    }
}
```

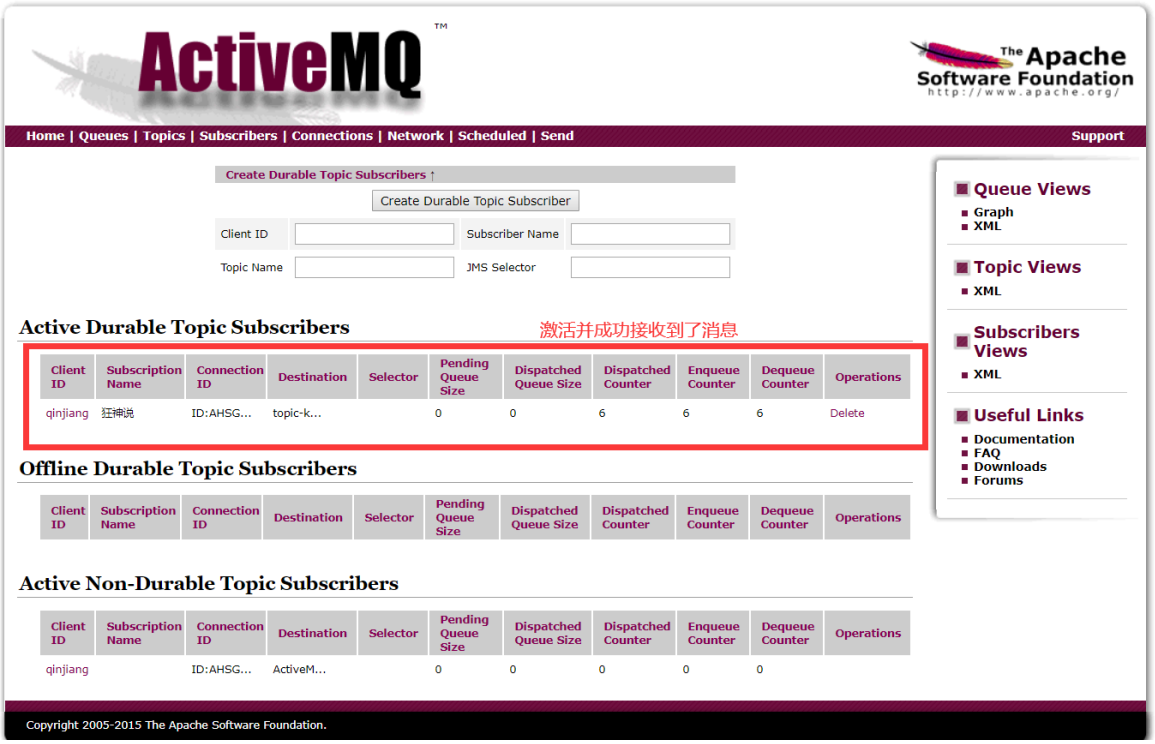
订阅方，需要创建持久订阅者！

```java
package com.kuang.activemq.topic;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;
import java.io.IOException;

/**
 * @author 秦疆 24736743@qq.com
 * 订阅持久化
 */
public class JmsConsumerTopic_Persist {

    public static final String ACTIVEMQ_URL = "tcp://192.168.0.110:61616";

    public static void main(String[] args) throws JMSException, IOException {
        ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(ACTIVEMQ_URL);
        Connection connection = factory.createConnection();

        // 设置id，好比你的微信id;
        connection.setClientID("qinjiang");

        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Topic topic = session.createTopic("topic-kuangshen-Persist");

        // 创建主题持久用户(参数二：订阅者)
        TopicSubscriber topicSubscriber = session.createDurableSubscriber(topic, "狂神说");

        connection.start(); // 连接

        // 消费消息
        Message message = topicSubscriber.receive();
        while (message!=null){
            TextMessage textMessage = (TextMessage) message;
            System.out.println("收到的持久化topic：" +textMessage.getText());
            message = topicSubscriber.receive(5000L); //持久化订阅的时间，不写就永久订阅！
        }

        session.close();
        connection.close();

    }

}
```

先启动订阅在启动发布！测试OK，查看控制台 `Subscribers 订阅人



由于是持久化操作，我们这次先启动发布者，在启动订阅者，看是否可以接受到消息：



注意：

1、一定要先运行一次消费者，等于向MQ注册，代表我订阅了这个主题

2、然后再运行生产者发送信息，此时，消费者无论是否在线，都会接收到，不在线的话，下次连接的时候，会把没有收过的消息都接收下来！

6、消息的可靠性-事务

为什么需要事务：一组操作要么同时成功，要么同时失败

> 参数说明

```
// 3、创建会话Session
// 参数1：事务、参数2：签收
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

事务偏生产者、签收偏消费者

Producer 提交时的事务：两个状态 true、false

false：

```
只要执行send，就进入到队列中。
关闭事务，那第二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。
```

> 代码测试

拷贝，队列消息生产者和whlie消费者的代码！

消息生产者事务测试：

```
// 开启事务
Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
try {
    session.commit(); // 开启了事务就一定要commit提交
} catch (JMSException e) {
    session.rollback(); // 出现异常，事务回滚
    e.printStackTrace();
}
```

消息消费者事务测试：只开启消费者的事务，设置为true，测试。

- 问题：消息消费到了，但是控制台数量没增加，**消息可以重复消费**
- 所以消费者要事务的话，也一定要提交！

7、消息的可靠性-签收

分析源码：查看签收的4种方式

```java
public interface Session extends Runnable {
    static final int AUTO_ACKNOWLEDGE = 1;   // 自动签收（默认）

    static final int CLIENT_ACKNOWLEDGE = 2;  // 客户端手动确认
           // 需要显示调用 message.acknowledge(); 进行签收，否则也会导致重复消费

    static final int DUPS_OK_ACKNOWLEDGE = 3;  // 自动批量确认

    static final int SESSION_TRANSACTED = 0;    // 事务提交并确认
}
```

注意：事务开启和没开启是有区别的

在事务性会话中，当一个事务被成功提交则消息被自动签收。如果事务回滚，则消息会被再次传送。

在非事务性会话中，消息何时被确认取决于创建会话时的应答模式（ACKNOWLEDGE）

8、点对点总结

点对点模型是基于队列的，生产者发消息到队列，消费者从队列接收消息，队列的存在使得消息的异步传输成为可能。和我们平时给朋友发短信类似。

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

2、队列可以长久地保存消息直到消费者接收到消息。消费者不需要因为担心消息会丢失而时刻和队列保持激活的连接状态，**充分体现了异步传输模式的优势。**

9、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。

主题可以被认为是消息的传输中介，发布者（publisher）发布消息到主题，订阅者（subscribe）从主题订阅消息。

主题使得消息订阅者和消息发送者保持互相独立，不需要接触即可保证消息的传送。

> 非持久订阅

非持久订阅只有当客户端处于激活状态，也就是和MQ保持连接状态才能收到发送到某个主题的消息。

如果消费者处于离线状态，生产者发送的主题消息将会丢失作废，消费者永远不会收到。

一句话：先要订阅注册才能接受到发布，只给订阅者发布。

> 持久订阅

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保存所有发送到主题的消息，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，不能恢复或重新派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。

当所有的消息必须被接收，则使用持久订阅，如果允许消息丢失，则使用非持久订阅

10、Broker

> 什么是 Broker

相当于ActiveMQ服务器实例

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

在Linux中，我们的ActiveMQ启动的时候去读取了配置文件 ActiveMQ.xml，我们可以选择配置不同的文件去启动。

> 嵌入式 Broker

官方文档：http://activemq.apache.org/how-do-i-embed-a-broker-inside-a-connection.html

api文档：http://activemq.apache.org/maven/apidocs/org/apache/activemq/broker/BrokerService.html

用ActiveMQ Broker 作为独立的消息服务器来构建Java应用。

ActiveMQ 也支持在 vm 中通信基于嵌入式的 broker，能够无缝的集成其他 java 应用。

1、导入pom依赖

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.10.0</version>
</dependency>
```

2、EmbedBroker.java

```
package com.kuang.activemq.embed;

import org.apache.activemq.broker.BrokerService;

/**
 * EmbedBroker 嵌入式代理
 * @author 狂神说Java
 */
public class EmbedBroker {
    public static void main(String[] args) throws Exception {

        BrokerService brokerService = new BrokerService();
        // jmx是java语言的新框架
        // 它允许你把所有资源(包括软硬件)封装成java对象,然后暴露在分布式环境中。
        brokerService.setUseJmx(true);
        brokerService.addConnector("tcp://localhost:61616");
```

```
        brokerService.start();
    }
}
```

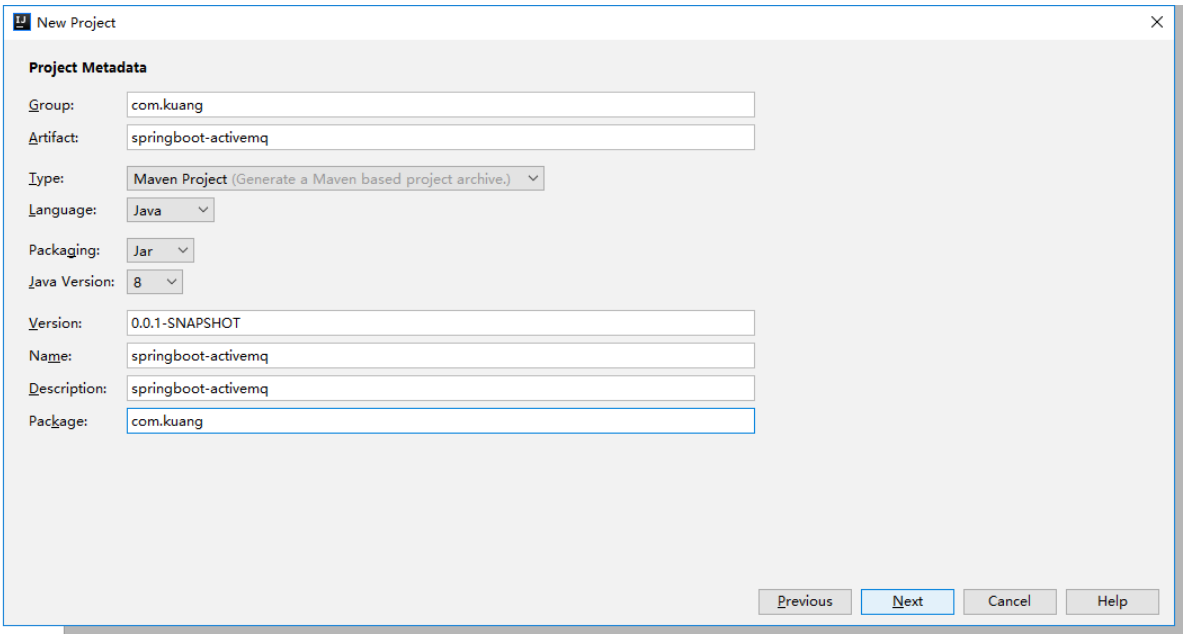
3、使用生产者消费者测试一下，看是否可以发布到此地址！

效果一样

3、整合Springboot

3.1、队列-发送者

1、新建一个springboot工程



2、导入pom依赖

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
```

3、配置application.yml

```yaml
server:
  port: 6666
spring:
  activemq:
    broker-url: tcp://192.168.0.110:61616 # 自己的MQ服务器地址
    user: admin
    password: admin
  jms:
    pub-sub-domain: false      # false = Queue   true = Topic

# 队列名称
myqueue: springboot-activemq-queue
```

4、ActiveMQ配置Bean，注意包

```java
package com.kuang.config;

import org.apache.activemq.command.ActiveMQQueue;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;

import javax.jms.Queue;

/**
 * @author 狂神说Java
 */
@Configuration
@EnableJms // 声明对 JMS 注解的支持。
public class ActiveMQConfig {
    @Value("${myqueue}")
    private String myqueue;

    @Bean
    public Queue queue(){
        return new ActiveMQQueue(myqueue);
    }

}
```

5、编写队列的消息生成者

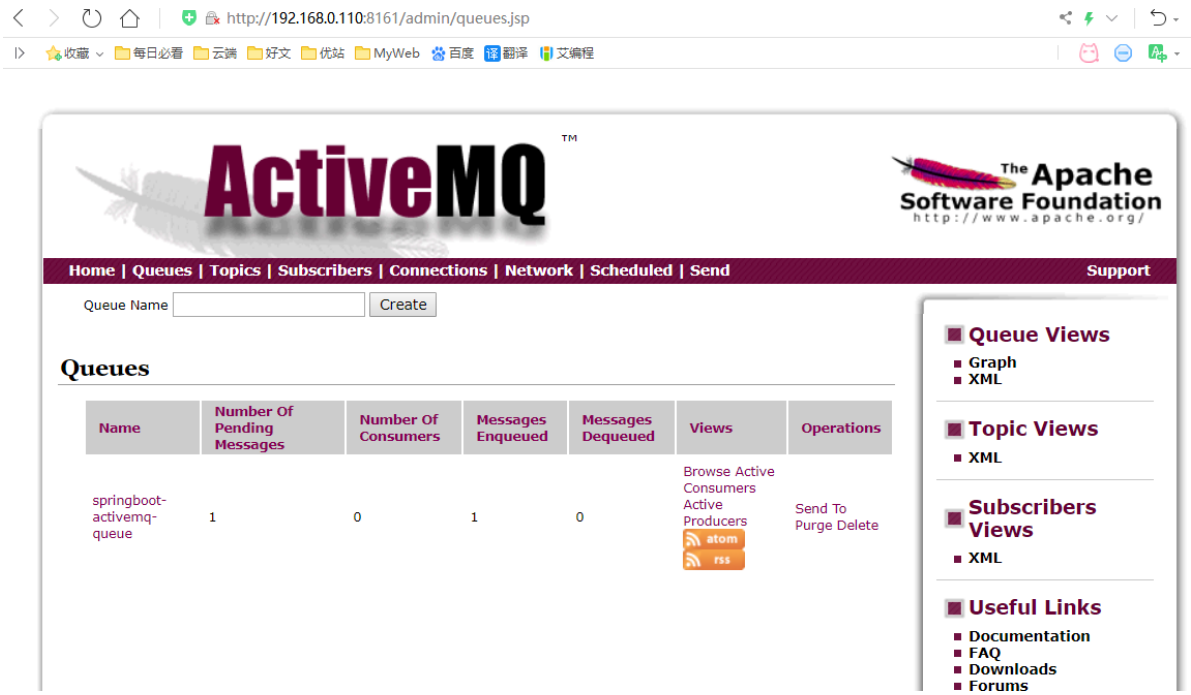
```java
package com.kuang.queue;

```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.stereotype.Component;

import javax.jms.Queue;

@Component
public class QueueProduce {

    @Autowired
    private JmsMessagingTemplate template;

    @Autowired
    private Queue queue;

    public void produceMsg(){
        template.convertAndSend(queue,"produceMsg");
    }

}
```

6、Controller测试类

```java
package com.kuang.controller;


import com.kuang.queue.QueueProduce;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ActiveMQController {

    @Autowired
    private QueueProduce queueProduce;

    // 理论每请求一次，都会发送一条消息到队列
    @RequestMapping("/queue/produce")
    public void queue_send(){
        queueProduce.produceMsg();
        System.out.println("消息发送成功！");
    }

}
```

7、启动主启动类测试！



> 新要求每隔3秒钟往MQ推送消息，定时发送

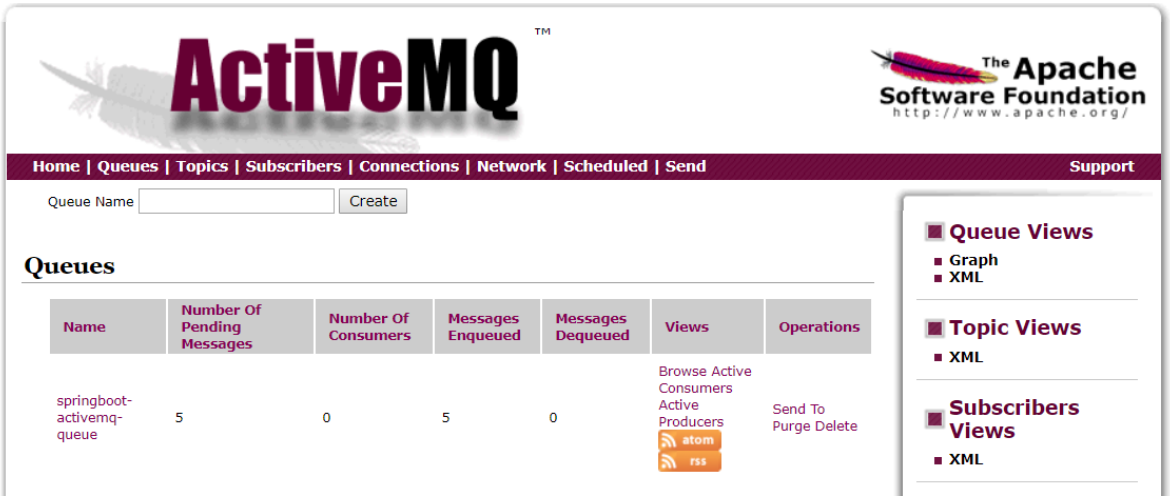
1、在队列发送类中新增方法

```
// 带定时功能的
@Scheduled(fixedDelay = 3000)
public void produceMsgScheduled(){
    template.convertAndSend(queue,"produceMsgScheduled");
    System.out.println("系统定时投递 produceMsgScheduled");
}
```

2、在主启动类上也要增加一个对应的支持注解

```
@SpringBootApplication
@EnableScheduling // 增加支持
public class SpringbootActivemqApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootActivemqApplication.class, args);
    }

}
```

3、启动主启动类进行测试，观察是否能够定时自动发送！

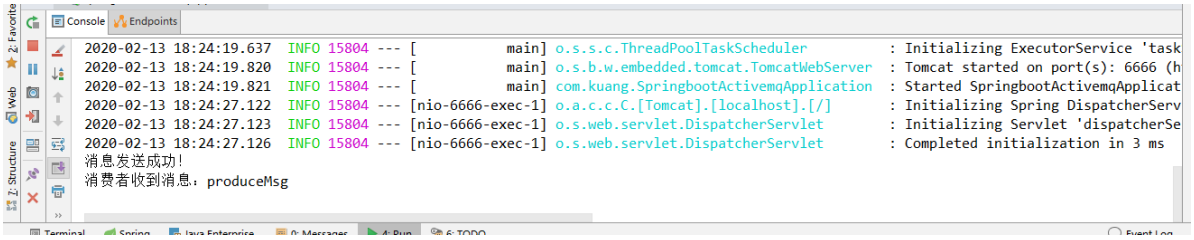


3.2、队列-消费者

1、新增消息消费者方法类

```java
package com.kuang.queue;

import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.JmsListener;

import javax.jms.JMSException;
import javax.jms.TextMessage;

@Configuration
public class QueueConsumer {

    @JmsListener(destination = "${myqueue}")
    public void receive(TextMessage textMessage) throws JMSException {
        System.out.println("消费者收到消息：“+textMessage.getText());
    }

}
```

2、启动测试，手动发送一条消息，看看是否能够接受到消息！



3.3、主题-生产者

1、修改 jms.pub-sub-domain 为true， 然后新增一个主题名称

```yaml
server:
  port: 6666
spring:
```

```yaml
  activemq:
    broker-url: tcp://192.168.0.110:61616 # 自己的MQ服务器地址
    user: admin
    password: admin
  jms:
    pub-sub-domain: true      # false = Queue   true = Topic

# 队列名称
myqueue: springboot-activemq-queue
# 主题名称
mytopic: springboot-activemq-topic
```

2、修改ActiveMQ的配置类

```java
package com.kuang.config;

import org.apache.activemq.command.ActiveMQTopic;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.EnableJms;
import javax.jms.Topic;


/**
 * @author 狂神说Java
 */
@Configuration
@EnableJms // 声明对 JMS 注解的支持。
public class ActiveMQConfig {
    @Value("${myqueue}")
    private String myqueue;

    // 读取配置文件
    @Value("${mytopic}")
    private String mytopic;

    // 修改返回的bean为 topic
    @Bean
    public Topic topic(){
        return new ActiveMQTopic(mytopic);
    }

}
```

3、新建一个主题发布者类

```java
package com.kuang.topic;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsMessagingTemplate;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import javax.jms.Topic;

@Component
public class TopicProduce {
```

```
    @Autowired
    private JmsMessagingTemplate jmsMessagingTemplate;

    @Autowired
    private Topic topic;

    @Scheduled(fixedDelay = 3000)
    public void produceTopic(){
        jmsMessagingTemplate.convertAndSend(topic,"=>produceTopic");
        System.out.println("消息推送完毕");
    }

}
```

4、主题发布需要有消费者等待，所以我们等消费者写完一起测试！

3.4、主题-消费者

1、编写topic的consumer

```
package com.kuang.queue;

import org.springframework.context.annotation.Configuration;
import org.springframework.jms.annotation.JmsListener;

import javax.jms.JMSException;
import javax.jms.TextMessage;

@Configuration
public class QueueConsumer {

    @JmsListener(destination = "${mytopic}")
    public void receive(TextMessage textMessage) throws JMSException {
        System.out.println("消费者收到主题发布消息: "+textMessage.getText());
    }

}
```

2、启动项目测试，测试通过！

```
2020-02-14 14:14:12.785  INFO 8052 --- [           m
2020-02-14 14:14:13.094  INFO 8052 --- [           m
2020-02-14 14:14:13.098  INFO 8052 --- [           m
消息推送完毕
消费者收到主题发布消息: =>produceTopic
消息推送完毕
消费者收到主题发布消息: =>produceTopic
消息推送完毕
消费者收到主题发布消息: =>produceTopic
```

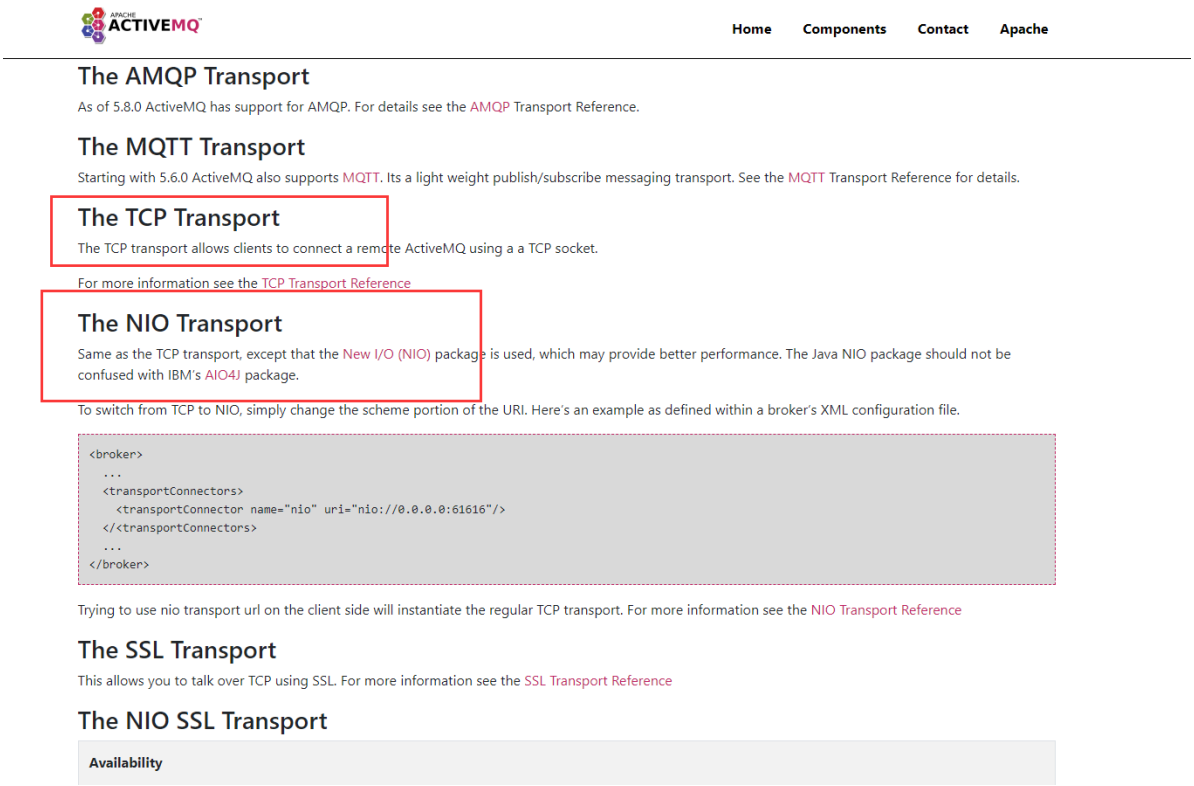
4、ActiveMQ的传输协议

4.1、前言

问题1：默认的61616端口如何修改?

问题2：你生产上的链接协议如何配置的？使用Tcp吗?

官网地址：http://activemq.apache.org/configuring-version-5-transports.html



> 是什么

ActiveMQ支持的client-broker通讯协议有：TCP、NIO、UDP、SSL、Http(s)、VM。

其中配置Transport Connector的文件在activeMQ安装目录的 conf/activemq.xml 中的 < transportConnectors> 标签之类。

1、使用 `docker exec -it activemq /bin/bash` 进入 ActiveMQ， `exit` 退出

```
[root@localhost Desktop]# docker exec -it activemq /bin/bash
root@39df41b9bf5a:/opt/activemq# ls
LICENSE  NOTICE  README.txt  activemq-all-5.14.3.jar  bin  conf  conf.tmp  data  docs  examples  lib  tmp  webapps  webapps-demo
root@39df41b9bf5a:/opt/activemq# cd conf
root@39df41b9bf5a:/opt/activemq/conf# ls
activemq.xml           broker.ks  client.ks  credentials-enc.properties  groups.properties  jetty-realm.properties  jmx.access    log4j.properties    login.config
broker-localhost.cert  broker.ts  client.ts  credentials.properties      java.security      jetty.xml               jmx.password  logging.properties  users.properties
root@39df41b9bf5a:/opt/activemq/conf#
```

2、打开 activemq.xml。

```
root@39df41b9bf5a: /opt/activemq/conf
File  Edit  View  Search  Terminal  Help
        </systemUsage>

        <!--
            The transport connectors expose ActiveMQ over a given protocol to
            clients and other brokers. For more information, see:

            http://activemq.apache.org/configuring-transports.html
        -->
        <transportConnectors>
            <!-- DOS protection, limit concurrent connections to 1000 and frame size to 100MB -->
            <transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
        </transportConnectors>

        <!-- destroy the spring context on shutdown to stop jetty -->
        <shutdownHooks>
            <bean xmlns="http://www.springframework.org/schema/beans" class="org.apache.activemq.hooks.SpringContextHook" />
        </shutdownHooks>

    </broker>
```

在配置文件给出的配置信息中，URL描述信息的头部都是采用协议名称，例如：

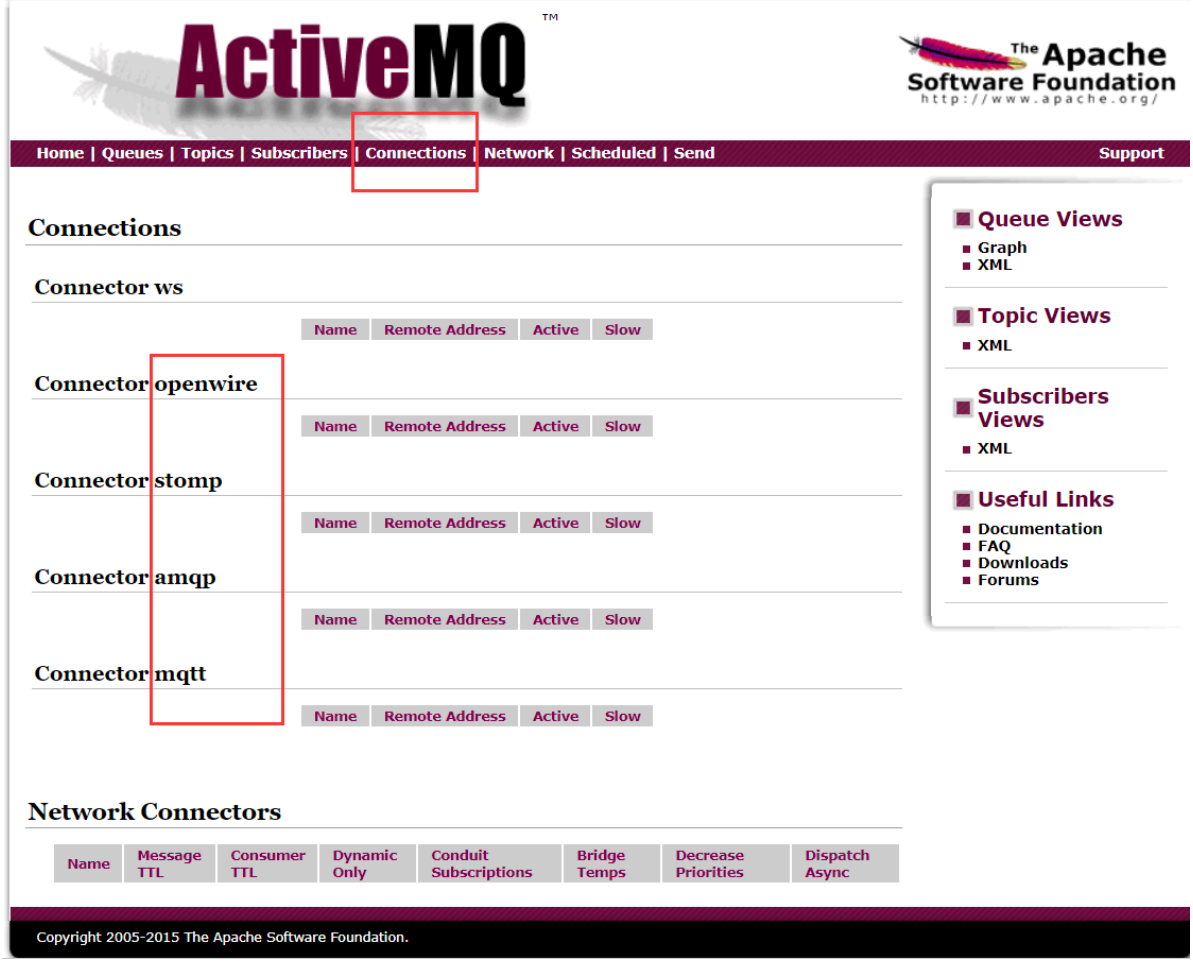
描述 amqp 协议的监听端口时，采用的URL描述格式为 “amqp://....”

描述 Stomp 协议的监听端口时，采用的URL描述格式为 “stomp://....”

唯独在进行 openwire 协议描述时，URL，头却采用的 “tcp://....”

这是因为ActiveMQ中默认的消息协议就是 openwire

在控制台中，点击连接查看，可以看到支持的协议。



4.2、不同种类分析

> Transmission Control Protocol （TCP） 默认的

1、这是默认的Broker配置，TCP的Client监听端口61616

2、在网络传输数据前，必须要序列化数据，消息是通过一个叫 wire protocol 的来序列化成字节流。默认情况下 ActiveMQ 把 wire protocol 叫做 OpenWire，它的目的是促使网络上的效率和数据快速交互。

3、TCP 连接的URL形式如：tcp://hostname:port?key=value&key=value，后面的参数是可选。

4、TCP传输的优点：

- TCP协议传输可靠性高，稳定性强
- 高效性：字节流方式传递，效率很高
- 有效性、可用性：应用广泛，支持任何平台

5、关于 Transport 协议的可配置参数可以参考官网：http://activemq.apache.org/tcp-transport-reference

New I/O API Protocol （NIO）

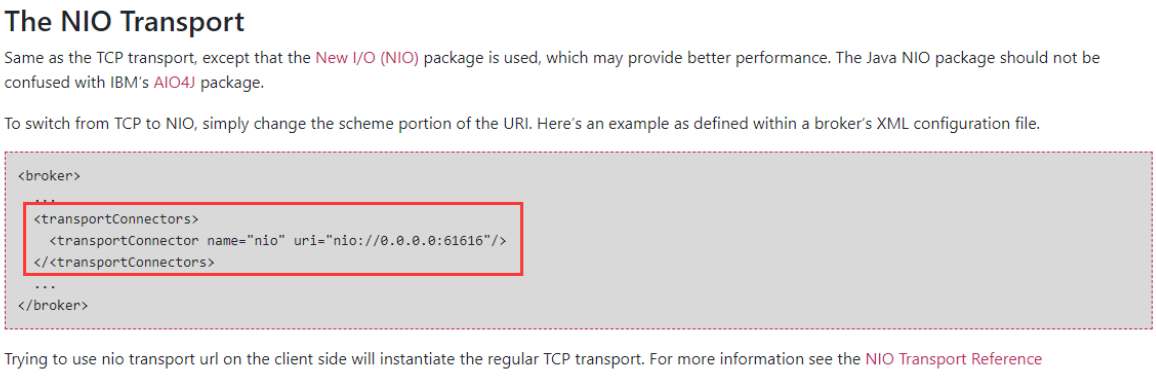
1、NIO 协议和 TCP 协议类似，但NIO更侧重于底层的访问操作。它允许开发人员对同一资源可有更多的client调用和服务端有更多的负载。

2、适合使用NIO协议的场景：

- 可能有大量的Client去连接到Broker上，一般情况下，大量的 Client 去连接 Broker 是被操作系统的线程所限制的。因此，NIO 的实现比 TCP 需要更少的线程去运行，所以建议使用NIO协议。
- 可能对于 Broker 有一个很迟钝的网络传输，NIO 比 TCP 提供更好的性能。

3、NIO连接的URL形式：nio://hostname:port?key=value

4、Transport Connector 配置示例，参考官网：http://activemq.apache.org/nio-transport-reference



The NIO Transport

Same as the TCP transport, except that the New I/O (NIO) package is used, which may provide better performance. The Java NIO package should not be confused with IBM's AIO4J package.

To switch from TCP to NIO, simply change the scheme portion of the URI. Here's an example as defined within a broker's XML configuration file.

```
<broker>
  ...
  <transportConnectors>
    <transportConnector name="nio" uri="nio://0.0.0.0:61616"/>
  </transportConnectors>
  ...
</broker>
```

Trying to use nio transport url on the client side will instantiate the regular TCP transport. For more information see the NIO Transport Reference

AMQP 协议

即 Advanced Message Queuing Protocol，一个提供统一消息服务的应用层标准高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计，基于此协议的客户端与消息中间件可传递消息。并不受客户端、中间件不同的产品，不同的开发语言等条件限制。

Stomp 协议

STOMP，Streaming Text Orrientated Message Protocol，是流文本定向消息协议，是一种为MOM（Message Proented Middleware，面向消息的中间件）设计的简单文本协议。

Secure Sockets Layer Protocol （SSL）

安全套接字层协议。连接的URL形式： ssl://hostname:port?key=value

> mqtt 协议

MQTT （Message Queuing Telemetry Transport，消息队列遥测传输）是IBM开发的一个即时通讯协议，有可能成为物联网的重要组成部分。该协议支持所有平台，几乎可以把所有联网物品和外部连接起来，被用来当做传感器和致动器（比如通过Twitter让房屋联网）的通信协议。

>

ActiveMq支持的网络协议

协议	描述
TCP	默认的协议，性能相对可以
NIO	基于TCP协议之上的，进行了扩展和优化，具有更好的扩展性
UDP	性能比TCP更好，但是不具有可靠性
SSL	安全链接
HTTP(S)	基于HTTP或者HTTPS
VM	VM本身不是协议，当客户端和代理在同一个Java虚拟机(VM)中运行时,他们之间需要通信,但不想占用网络通道，而是直接通信，可以使用该方式

4.3、测试NIO协议

1、停止ActiveMQ

2、拷贝activemq.xml 为 activemq1.xml

3、修改activemq.xml

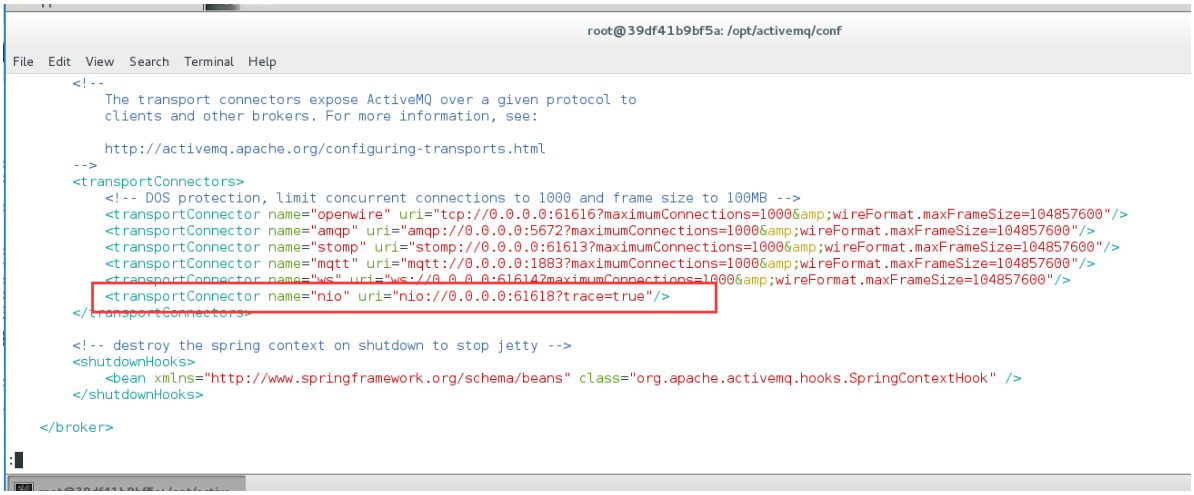
```
<transportConnectors>
    <transportConnector name="nio" uri="nio://0.0.0.0:61618?trace=true"/>
</transportConnectors>
```

一定要开放防火墙端口

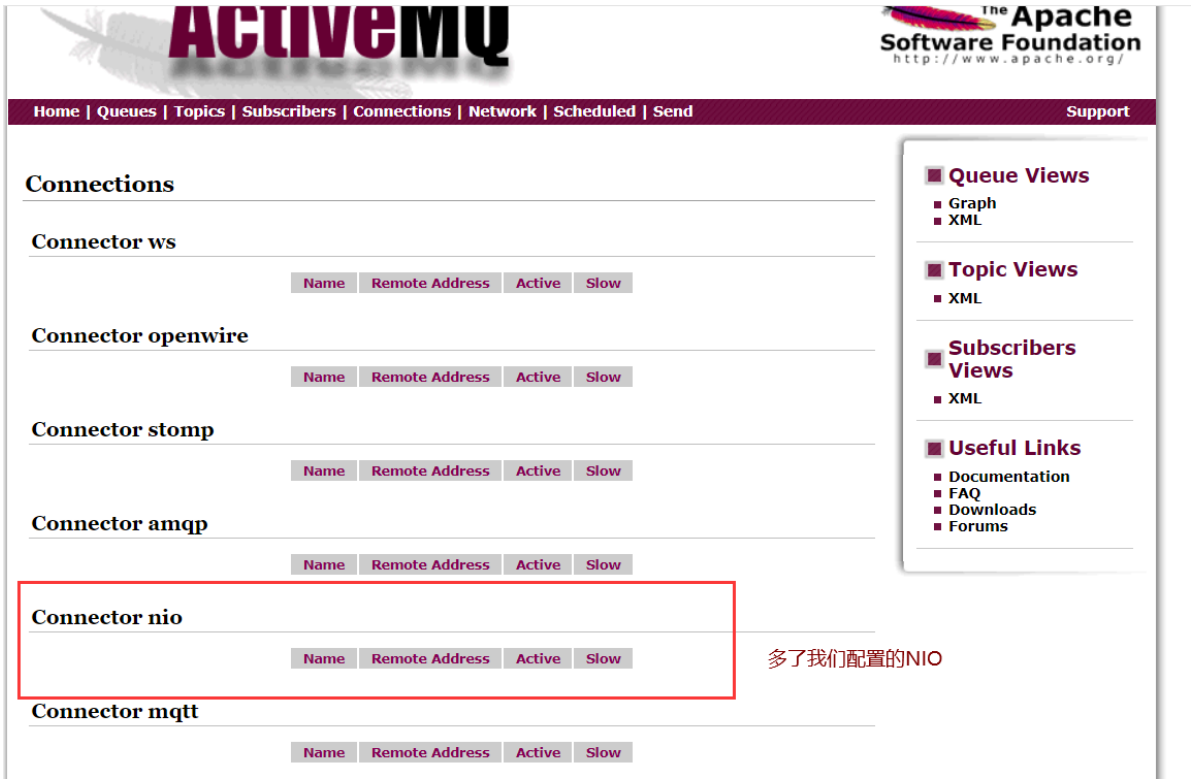
如果你不特别指定ActiveMQ的网络监听端口，那么这些端口都将使用BIO网络IO模型。

所以为了首先提高单节点的网络吞吐性能，我们需要明确指定Active的网络IO模型。

如下所示：URL格式头是 nio 开投诉，表示这个端口使用以 TCP 协议为基础的NIOI网络IO模型



4、重启 activeMQ，查看控制台

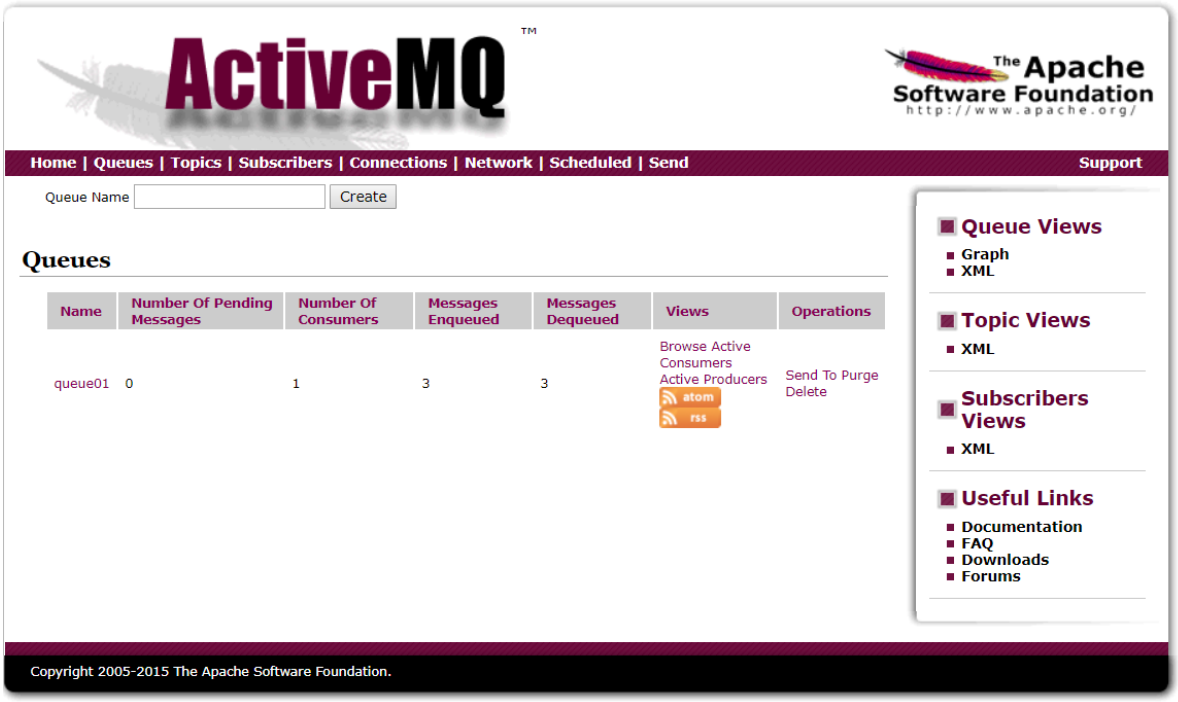


5、使用Java程序连接测试，需要修改链接的 url 为

```
1  public static final String ACTIVEMQ_URL = "nio://192.168.0.110:61618";
```

问题：由于Docker没有开启这个端口映射，所以我们需要重新生成一下容器镜像；

- 停止activeMQ
- 删除容器
- 启动 `docker run -d --name activemq -p 61616:61616 -p 61618:61618 -p 8161:8161 webcenter/activemq`
- 修改配置文件，再次测试！



4.4、NIO增强

上述的NIO性能不错了，如何进一步优化?

问题：URL格式头以 “nio” 开头，表示这个端口使用Tcp协议为基础的NIO网络IO模型。但是这样的设置方式，只能使用这个端口支持Openwire协议。

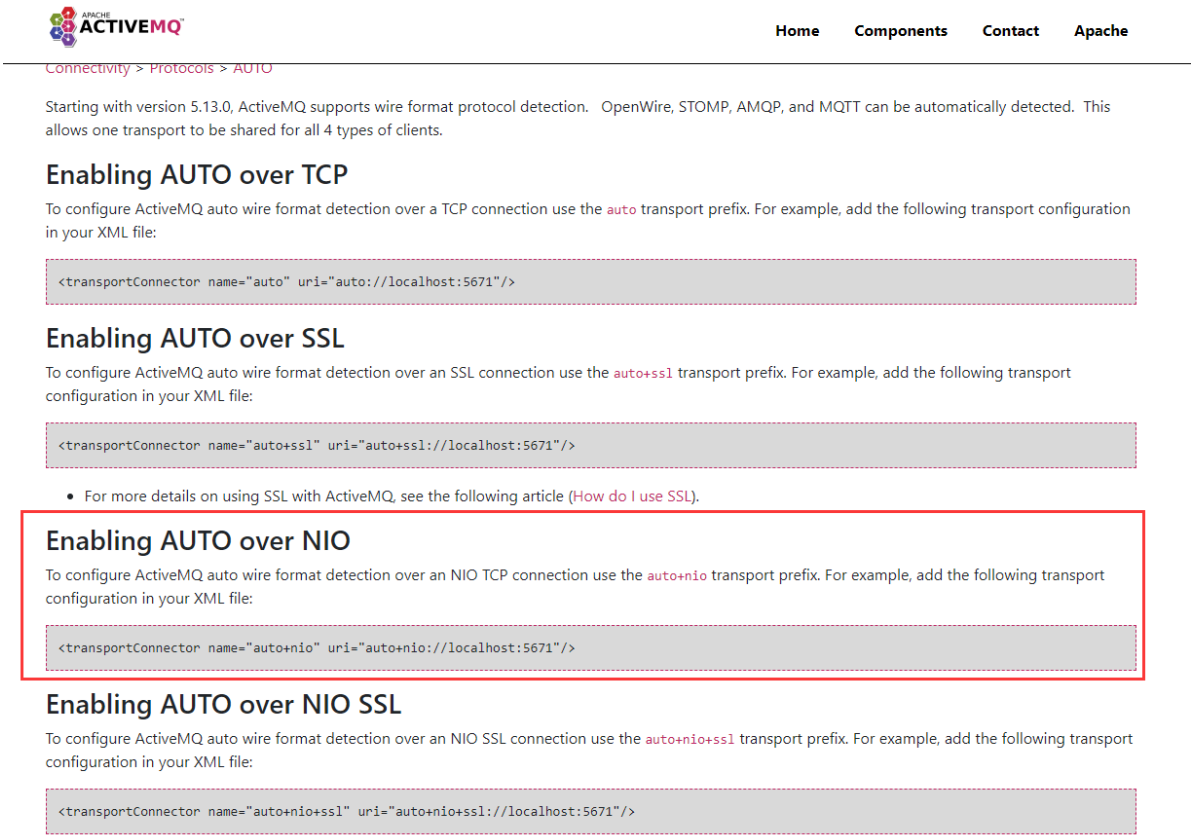
那么我们怎么既让这个端口支持NIO网络IO模型，又让它支持多个协议呢?

> 解决　官方文档：http://activemq.apache.org/auto

1、使用 auto 关键字。

2、使用 "+" 符号来为端口设置多种特性

3、如果我们既需要某一个端口支持NIO网络IO模型，又需要它支持多个协议。



5、消息持久化

为了测试方便，我们在Windows也安装一个 ActiveMQ；

1、下载并解压 ActiveMQ

2、进入bin目录。打开cmd `activemq-admin.bat start` 启动

3、http://localhost:8161/admin/ 测试控制台

5.1、什么是持久化机制

一句话：MQ服务器宕机了，消息不会丢失的机制

为了避免意外宕机以后丢失信息，需要做到重启后可以恢复消息队列，消息系统一般都会采用持久化机制。

ActiveMQ的消息持久化机制有 JDBC、AMQ、KahaDB 和 LevelDB，无论使用哪种持久化方式，消息的存储逻辑都是一致的。

就是在发送者将消息发送出去后，消息中心首先将消息存储到本地数据文件、内存数据库或者远程数据库等。再试图将消息发送给接收者，成功则将消息从存储中删除，失败则继续尝试发送。

消息中心启动以后首先要检查指定的存储位置。如果有未发送成功的消息，则需要把消息发出去。

> AMQ Message Store（了解）

基于文件的存储方式，是以前的默认消息存储，现在不用了

AMQ 是一种文件存储形式，它具有写入速度快和容易恢复的特点。消息存储在一个个文件中，文件的默认大小为32M，当一个存储文件中的消息已经被全部消费，那么这个文件将被标识为可删除，在下一个清除阶段，这个文件被删除。AMQ适用于ActiveMQ5.3之前的版本。

5.2、KahaDB(默认的)

KahaDB：这个5.4版本之后出现的**默认的持久化方式**，与AMQ很相似，不同的是只为Destination创建一个索引。

KahaDb恢复时间远远小于其前身AMQ并且使用更少的数据文件，所以可以完全代替AMQ。（AMQ适用于ActiveMQ5.3之前的版本。）KahaDB的持久化机制同样是基于日志文件，索引和缓存。

配置方式：

在`activemq.xml`配置文件中配置，更加详细参数请参考：https://activemq.apache.org/kahadb。

```
<broker brokerName="broker">
    <persistenceAdapter>
      <kahaDB directory="数据存储目录" journalMaxFileLength="32mb"/>
    </persistenceAdapter>
 </broker>
```

默认的数据存储位置在，data目录下的 kahadb 文件夹！ `${activemq.data}/kahadb`

> **KahaDB主要特性**

1、日志形式存储消息；

2、消息索引以B-Tree结构存储，可以快速更新；

3、完全支持JMS事务；

4、支持多种恢复机制；

> 说明

KahaDB是目前默认的存储方式，可用于任何场景，提高了性能和恢复能力。

消息存储使用一个事务日志和仅仅用一个索引文件来存储它所有的地址。

KahaDB是一个专门针对消息持久化的解决方案，它对典型的消息使用模式进行了优化。

数据被追加到 data logs中。当不再需要log文件中的数据的时候，log文件会被丢弃。

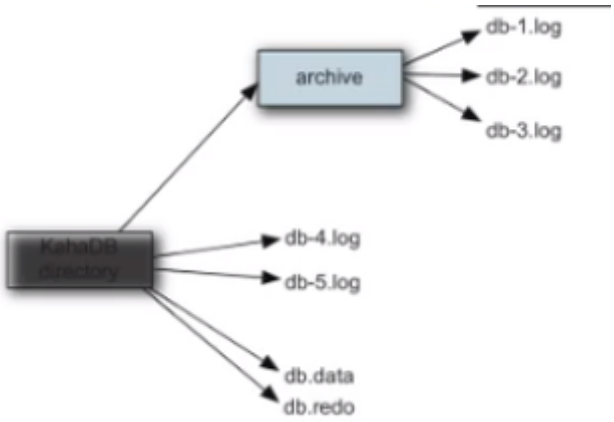
> kahaDB的存储原理

kahadb 在消息保存目录中只有4类文件和一个lock，跟ActiveMQ的其他几种文件存储引擎相比这就非常简洁了；

```
/myactiveMQ/apache-activemq-5.15.9/data/kahadb
[root@zzyy kahadb]# ll
总用量 844
-rw-r--r--. 1 root root 33554432 6月  14 12:24 db-1.log
-rw-r--r--. 1 root root    69632 6月  14 12:34 db.data
-rw-r--r--. 1 root root       21 6月  14 12:34 db.free
-rw-r--r--. 1 root root    69760 6月  14 12:34 db.redo
-rw-r--r--. 1 root root        8 6月  14 12:23 lock
```

db-[number].log

KahaDB存储消息到预定义大小的数据记录文件中，文件命名为 db-[number].log 。当数据文件已满时，一个新的文件会随之创建，number数值也会随之递增，它随着消息数量的增多，如每32M一个文件，文件名按照数字进行编号，如 db-1.log、db-2.log、... 。当不再有引用到数据文件中的任何消息时，文件会被删除或归档。



db.data

该文件包含了持久化的BTree索引，索引了消息数据记录中的消息，它是消息的索引文件，本质上是B-Tree（B树），使用B-Tree作为索引指向 db-[number].log 里面存储的消息。

db.free

当前 db.data 文件里哪些页面是空闲的，文件具体内容是所有空闲页的ID。

db.redo

用来进行消息恢复，如果KahaDB消息存储在强制退出后启动，用于恢复BTree索引。

lock

lock文件锁，表示当前获得 kahadb 读写权限的broker。

四类文件+一把锁 ==> KahaDB

5.3、了解LevelDB

这种文件系统是从ActiveMQ 5.8 之后引进的，它和 KahaDB 非常相似，也是基于文件的本地数据库存储形式，但是它提供比 KahaDB更快的持久性。

但它不使用自定义 B-Tree 实现来索引预写日志，而是使用基于LevelDB的索引。

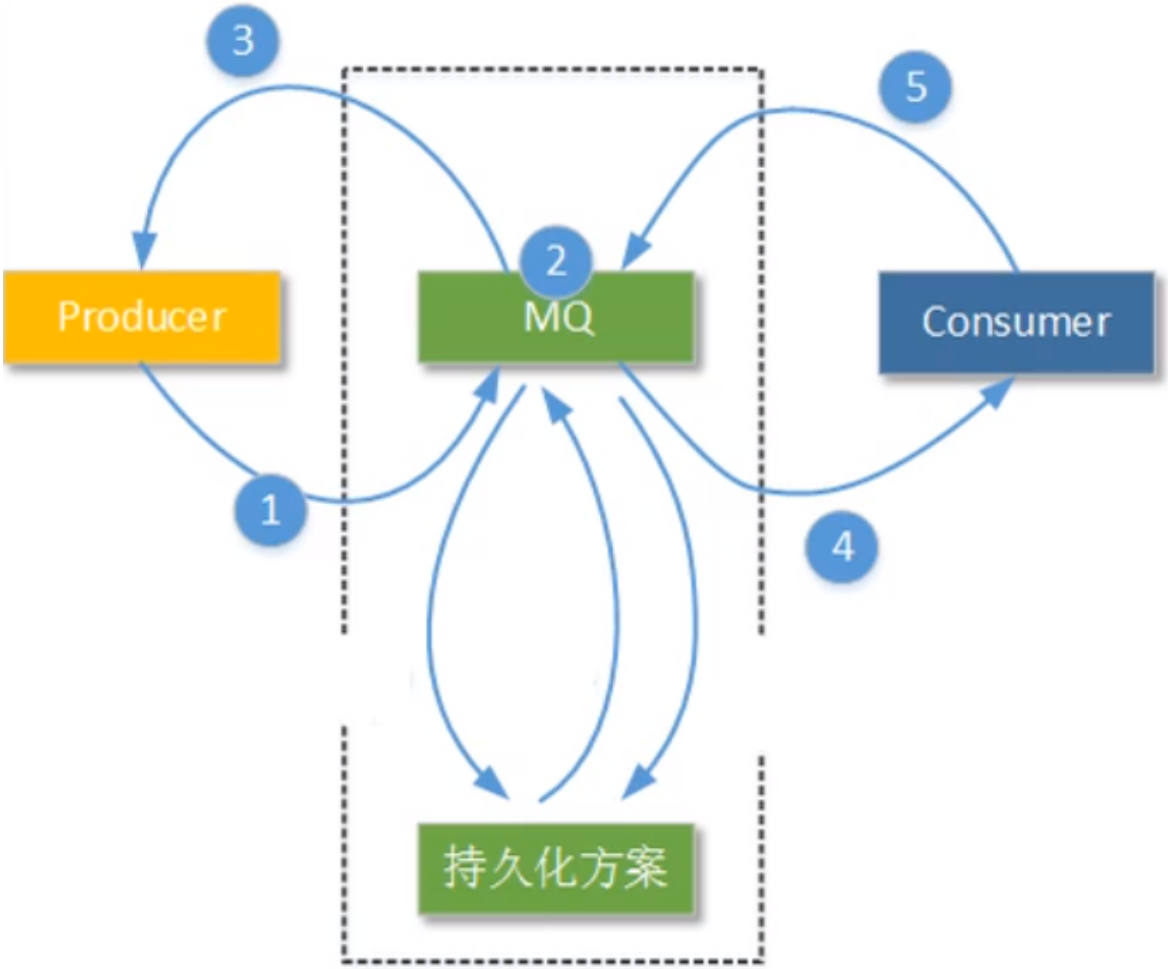
题外话：为什么LeavelDB 更快，并且5.8 以后就支持，为什么还是默认 KahaDB 引擎，因为 activemq 官网本身没有定论，LeavelDB 之后又出了可复制的LeavelDB 比LeavelDB 更性能更优越，但需要基于 ZooKeeper 所以这些官方还没有定论，仍旧使用 KahaDB。

默认配置如下：

```
<persistenceAdapter>
    <levelDB directory="activemq-data">
</persistenceAdapter>
```

5.4、配置JDBC

> MQ + MySQL 模型



> ActiveMQ 配置 JDBC 持久化操作

官网：https://dns.xuexi.icu/-----https://activemq.apache.org/persistence

1、添加 Mysql 数据库的驱动包到 lib 文件夹

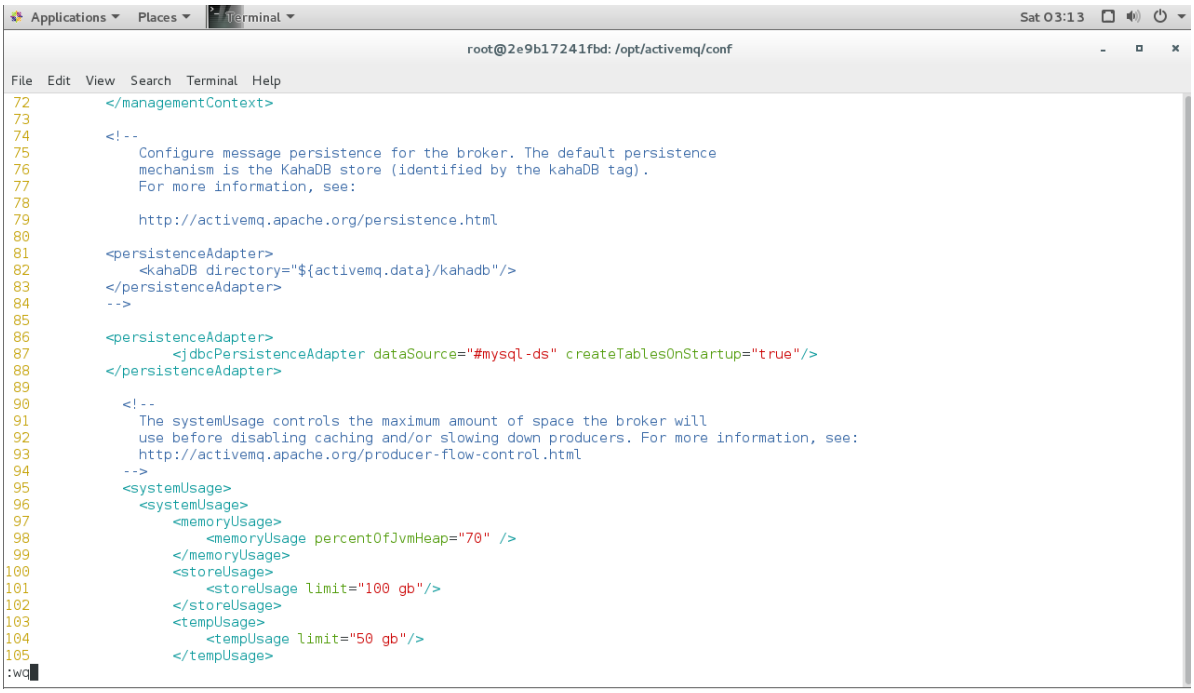
```
[root@localhost Desktop]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
                                                NAMES
2e9b17241fbd        webcenter/activemq   "/app/run.sh"       25 hours ago        Up 3 hours          1883/tcp, 5672/tcp, 0.0.0.0:8161->8161/
tcp, 0.0.0.0:61616->61616/tcp, 61613-61614/tcp, 0.0.0.0:61618->61618/tcp   activemq
[root@localhost Desktop]# docker cp /mnt/hgfs/share/mysql-connector-java-5.1.47.jar 2e9b17241fbd:/opt/activemq/lib
[root@localhost Desktop]#
```

```
root@2e9b17241fbd:/opt/activemq/lib# ls
activemq-broker-5.14.3.jar           activemq-protobuf-1.1.jar   geronimo-j2ee-management_1.1_spec-1.0.1.jar  optional
activemq-client-5.14.3.jar           activemq-rar.txt            geronimo-jms_1.1_spec-1.1.1.jar              slf4j-api-1.7.13.jar
activemq-console-5.14.3.jar          activemq-spring-5.14.3.jar  geronimo-jta_1.0.1B_spec-1.0.1.jar           web
activemq-jaas-5.14.3.jar             activemq-web-5.14.3.jar     hawtbuf-1.11.jar
activemq-kahadb-store-5.14.3.jar     camel                       jcl-over-slf4j-1.7.13.jar
activemq-openwire-legacy-5.14.3.jar  extra                       mysql-connector-java-5.1.47.jar
```

2、修改 activemq.xml 配置文件

修改前kahaDB	修改后jdbcPersistenceAdapter
<persistenceAdapter> <kahaDB directory="${activemq.data}/kahadb"/> </persistenceAdapter>	<persistenceAdapter> <jdbcPersistenceAdapter dataSource="#mysql-ds"/> </persistenceAdapter>
	dataSource指定将要引用的持久化数据库的bean名称， createTablesOnStartup是否在启动的时候创建数据表，默认值是true，这样每次启动都会去创建数据表了，一般是第一次启动的时候设置为true之后改成false。

```
<persistenceAdapter>
  <jdbcPersistenceAdapter dataSource="#mysql-ds" createTablesOnStartup="true"/>
</persistenceAdapter>
```



3、上面指向了 mysql-ds 的数据源，所以我们需要配置一个数据源！

```
<bean id="mysql-ds" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://192.168.0.106:3306/activemq?relaxAutoCommit=true" />
    <property name="username" value="root" />
    <property name="password" value="123456" />
    <property name="maxTotal" value="200" />
    <property name="poolPreparedStatements" value="true" />
</bean>
```

注意bean存放的位置，不要放错了

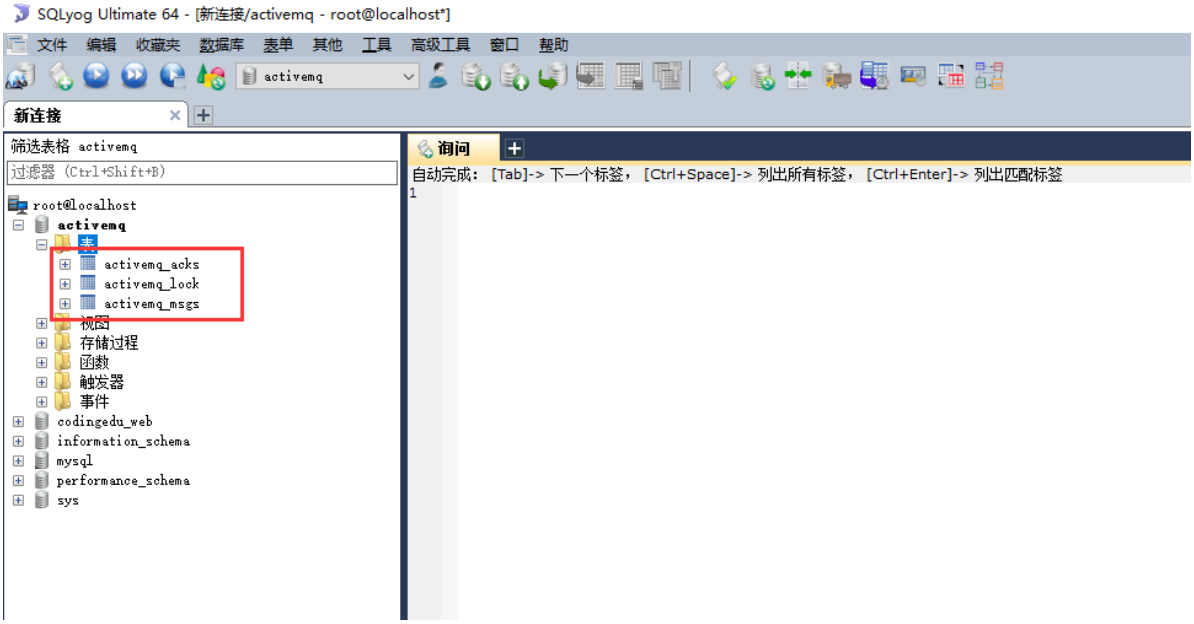
```
File  Edit  View  Search  Terminal  Help
23
24      <!-- Allows us to use system properties as variables in this configuration file -->
25      <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
26          <property name="locations">
27              <value>file:${activemq.conf}/credentials.properties</value>
28          </property>
29      </bean>
30
31    <!-- Allows accessing the server log -->
32     <bean id="logQuery" class="io.fabric8.insight.log.log4j.Log4jLogQuery"
33           lazy-init="false" scope="singleton"
34           init-method="start" destroy-method="stop">
35     </bean>
36
37     <bean id="mysql-ds" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close">
38         <property name="driverClassName" value="com.mysql.jdbc.Driver" />
39         <property name="url" value="jdbc:mysql://192.168.0.106:3306/activemq?relaxAutoCommit=true" />
40         <property name="username" value="root" />
41         <property name="password" value="123456" />
42         <property name="maxTotal" value="200" />
43         <property name="poolPreparedStatements" value="true" />
44     </bean>
45
46      <!--
47          The <broker> element is used to configure the ActiveMQ broker.
48      -->
49      <broker xmlns="http://activemq.apache.org/schema/core" brokerName="localhost" dataDirectory="${activemq.data}">
50
```

4、创建一个activemq的数据库

```
1  -- 创建数据库
2  CREATE DATABASE `activemq` CHARACTER SET utf8 COLLATE utf8_general_ci;
```

ActiveMQ 重新启动后会自动在 mysql 的activemq 数据库下创建三张表：activemq_msgs 、activemq_acks、activemq_lock

- activemq_acks：用于存储订阅关系。如果是持久化Topic，订阅者和服务器的订阅关系在这个表保存
- activemq_lock：在集群环境中才有用，只有一个Broker可以获得消息，称为Master Broker
- activemq_msgs：用于存储消息，Queue和Topic都存储在这个表中



5.5、测试JDBC

> 队列

1、新建一个JDBC包

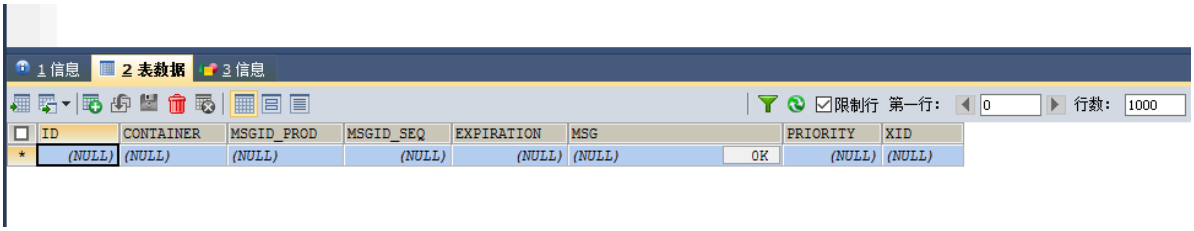
2、新建一个队列生产者测试类，运行后查看数据库

```java
package com.kuang.activemq.jdbc;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;

/**
 * @author 秦疆 24736743@qq.com
 */
public class JmsProduce {

    public static final String ACTIVEMQ_URL = "tcp://127.0.0.1:61616";

    public static void main(String[] args) throws JMSException {

        ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(ACTIVEMQ_URL);
        Connection connection = factory.createConnection();
        connection.start();

        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue("jdbc-queue01");
        MessageProducer producer = session.createProducer(queue);

        // 一定要开启持久化
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);

        for (int i = 1; i <= 3; i++) {
            TextMessage textMessage = session.createTextMessage("msg=>" + i);
            producer.send(textMessage);
        }

        producer.close();
        session.close();
        connection.close();

        System.out.println("消息发布到MQ完成");

    }

}
```

3、新建一个队列消费者测试类，运行后查看数据库

```
package com.kuang.activemq.jdbc;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;

/**
 * @author 秦疆 24736743@qq.com
 */
public class JmsConsumer {

    public static final String ACTIVEMQ_URL = "tcp://127.0.0.1:61616";

    public static void main(String[] args) throws JMSException {
        ActiveMQConnectionFactory factory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        Connection connection = factory.createConnection();
        connection.start();

        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue("jdbc-queue01");
        MessageConsumer consumer = session.createConsumer(queue);

        while (true){
            TextMessage receive = (TextMessage) consumer.receive(4000L);
            if (receive!=null){
                System.out.println("消息接收者收到消息:"+receive.getText());
```

```
            }else {
                break;
            }
        }

        // 9、关闭资源
        consumer.close();
        session.close();
        connection.close();

    }
}
```



点到点队列小结：

在点对点类型中，当DeliveryMode设置为NON_PERSISTENCE 时，消息被保存在内存中；

当DeliveryMode设置为PERSISTENCE 时，消息被保存在Broker的相应的文件或者数据库中。

而且点对点类型中消息一旦被Consumer消费就从Broker中删除

> Topic 主题测试

1、编写消息生产者

```
package com.kuang.activemq.jdbc;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;
```

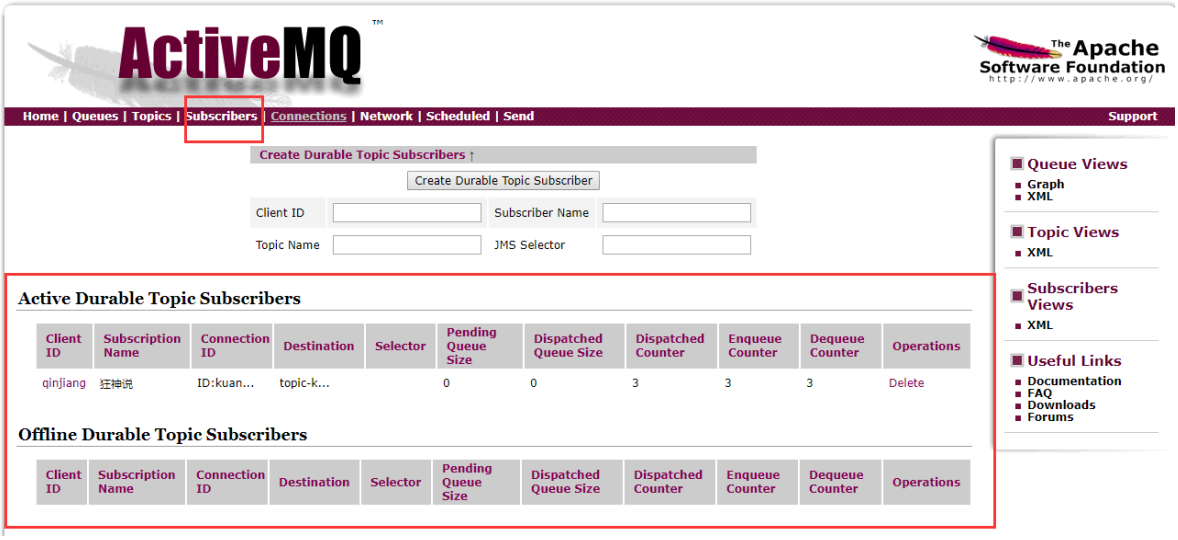
```
6
7  /**
8   * @author 秦疆 24736743@qq.com
9   * 发布
10  */
11 public class JmsProduceTopic {
12
13     public static final String ACTIVEMQ_URL = "tcp://127.0.0.1:61616";
14
15     public static void main(String[] args) throws JMSException {
16
17         ActiveMQConnectionFactory factory = new
   ActiveMQConnectionFactory(ACTIVEMQ_URL);
18         Connection connection = factory.createConnection();
19         Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);
20         Topic topic = session.createTopic("topic-kuangshen");
21         MessageProducer producer = session.createProducer(topic);
22
23         // 一定要设置完持久化在连接connection
24         producer.setDeliveryMode(DeliveryMode.PERSISTENT);
25         connection.start();
26
27         for (int i = 1; i <= 3; i++) {
28             TextMessage textMessage = session.createTextMessage("topic-
   msg=>" + i);
29             producer.send(textMessage);
30         }
31
32         producer.close();
33         session.close();
34         connection.close();
35
36         System.out.println("topic消息发布到MQ完成");
37     }
38 }
```

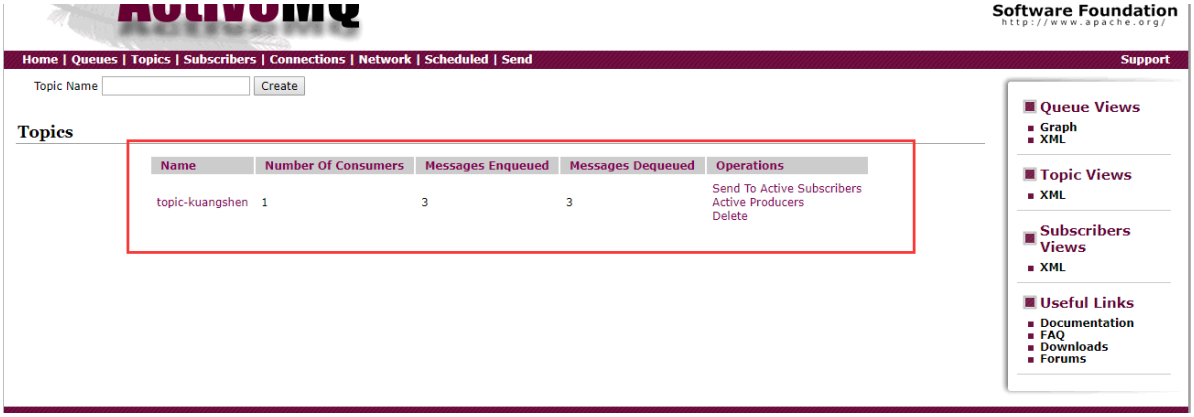
2、编写消费者

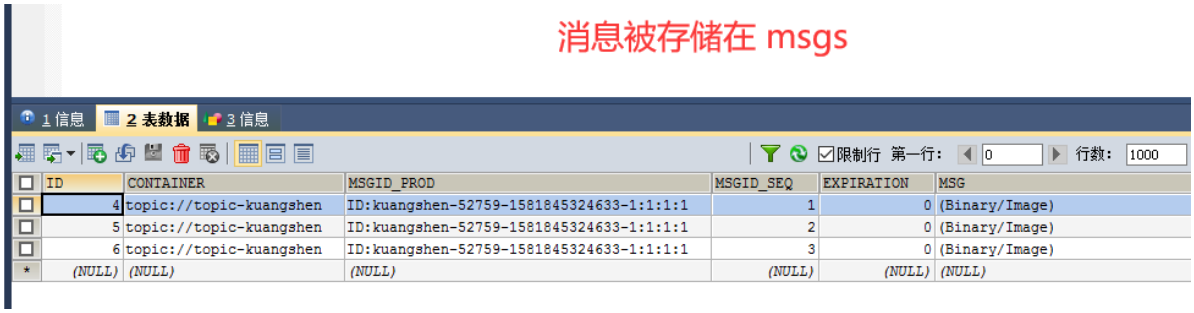
```
1  package com.kuang.activemq.jdbc;
2
3  import org.apache.activemq.ActiveMQConnectionFactory;
4
5  import javax.jms.*;
6  import java.io.IOException;
7
8  /**
9   * @author 秦疆 24736743@qq.com
10  * 订阅即消费者
11  */
12 public class JmsConsumerTopic {
13
14     public static final String ACTIVEMQ_URL = "tcp://127.0.0.1:61616";
15
16     public static void main(String[] args) throws JMSException, IOException
   {
17         ActiveMQConnectionFactory factory = new
   ActiveMQConnectionFactory(ACTIVEMQ_URL);
```

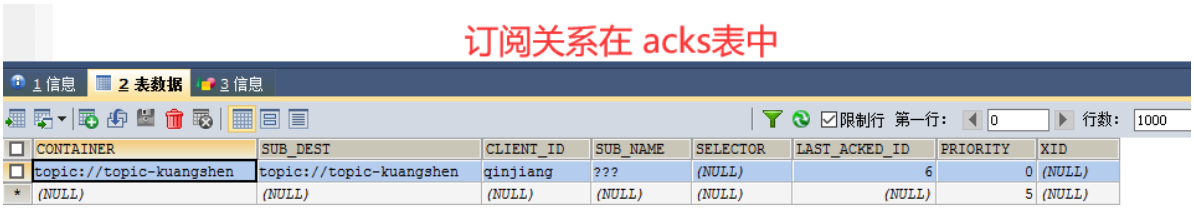
```
        Connection connection = factory.createConnection();

        // 设置id，好比你的微信id;
        connection.setClientID("qinjiang");

        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
        Topic topic = session.createTopic("topic-kuangshen");

        // 创建主题持久用户(参数二：订阅者)
        TopicSubscriber topicSubscriber =
session.createDurableSubscriber(topic, "狂神说");

        connection.start(); // 连接

        // 消费消息
        Message message = topicSubscriber.receive();
        while (message!=null){
            TextMessage textMessage = (TextMessage) message;
            System.out.println("收到的持久化topic："+textMessage.getText());
            //message = topicSubscriber.receive(5000L); //持久化订阅的时间，不写
就永久订阅！
            message = topicSubscriber.receive();
        }

        session.close();
        connection.close();
    }
}
```

3、启动测试，先启动消费者，在启动生产者，一定要订阅成功才可以！









> 小结

Queue

在没有消费者消费的情况下会将消息保存到 activemq_msgs表中，只要有任意一个消费者已经消费过了，消费之后这些消息将会立即被删除。

Topic

一般是先启动消费订阅然后再启动生产，消息会被保留在 activemq_msgs表中，订阅关系在 acks 表中。

注意

1、数据库的jar包和数据库连接池的jar包一定要导入lic目录下。

2、在createTableOnStartup属性设置了true的，在第一次生成完表之后，需要改为flase；

3、操作系统机器名不能存在 `-`，否则会报错：BeanFactory not initialized or already closed；

5.6、Journal

> 说明

这种方式克服了 JDBC Store的不足，JDBC每次消息过来，都需要去写库和读库。

ActiveMQ Journal，使用高速缓存写入技术，大大提高了性能。

当消费者的消费速度能够及时跟上生产者消息的生产速度时，Journal文件能够大大减少需要写入到DB中的消息。

举个例子，生产者生产了1000条消息，这1000条消息会保存到 journal 文件，如果消费者的消费速度很快的情况下，在 journal 文件还没有同步到DB之前，消费者已经消费了90%以上的信息，那么这个时候只需要同步剩余的10%的信息到DB。如果消费者的消费速度很慢，这个时候 Journal 文件可以使消息以批量方式写到 DB。

配置

修改 activemq.xml 配置文件

修改前PersistenceAdapter	修改后persistenceFactory
<persistenceAdapter> <jdbcPersistenceAdapter dataSource="#mysql-ds"/> </persistenceAdapter>	<persistenceFactory> <journalPersistenceAdapterFactory journalLogFiles="4" journalLogFileSize="32768" useJournal="true" useQuickJournal="true" dataSource="#mysql-ds" dataDirectory="activemq-data" /> </persistenceFactory>

配置完成后，重新启动 activemq 就可以了！

```
<persistenceFactory>
    <journalPersistenceAdapterFactory
                                    journalLogFiles="4"
                                    journalLogFileSize="32768"
                                    useJournal="true"
                                    useQuickJournal="true"
                                    dataSource="#mysql-ds"
                                    dataDirectory="activemq-data" />
</persistenceFactory>
```

小结

持久化消息主要是指：

MQ所在的服务器down了消息不会丢失的机制；

持久化机制演化过程：

从最初的 AMQ Message Store 方案到 ActiveMQ V4 版本中推出的 High performance journal（高性能事务支持）附件，并且同步推出了关于关系型数据库的存储方案。ActiveMQ 5.3 版本中又推出了对KahaDB的支持（V5.4版本后称为 ActiveMQ 默认的持久化方案），后来 ActiveMQ V5.8版本支持 LevelDB，到现在，V5.9 + 版本提供了标准的 Zookeeper + LevelDB 集群化方案。我们重点介绍了KahaDB、LevelDB和mysql数据库这三种持久化存储方案。

ActiveMQ的消息持久化机制有：

AMQ　　基于日志文件

KahaDB　　基于日志文件，从ActiveMQ 5.4 开始默认的持久化插件

JDBC 基于第三方数据库

LevelDB 基于文件的本地数据库存储，从ActiveMQ5.8版本之后又推出了 LevelDB 的持久化引擎，性能高于KahaDB。

Replicated LevelDB Store 从 ActiveMQ5.9提供了基于 LevelDB和 Zookeeper 的数据复制方式，用于Master-slave方式的首选数据复制方案。

无论使用哪种持久化方式，消息的存储逻辑都是一致的：

就是在发送者将消息发送出去后，消息中心首先将消息存储到本地数据文件、内存数据库。或者远程数据库等，然后试图将消息发送给接收者，发送成功则将消息从存储中删除，失败则继续尝试。消息中心启动以后首先要检查指定的存储位置。如果有未发送成功的消息，则需要把消息发送出去。

6、多节点集群

问题：引入消息队列之后该如何保证其高可用性?

答案：基于Zookeeper和LevelDB搭建ActiveMQ集群，集群仅提供主备方式的高可用集群功能，避免单点故障。

6.1、三种集群方式对比

官网地址：http://activemq.apache.org/masterslave.html

Master Slave Type	Requirements	Pros	Cons
Shared File System Master Slave	A shared file system such as a SAN	Run as many slaves as required. Automatic recovery of old masters	Requires shared file system
JDBC Master Slave	A Shared database	Run as many slaves as required. Automatic recovery of old masters	Requires a shared database. Also relatively slow as it cannot use the high performance journal
Replicated LevelDB Store	ZooKeeper Server	Run as many slaves as required. Automatic recovery of old masters. Very fast.	Requires a ZooKeeper server.

1、基于 sharedFileSystem共享文件系统（KahaDB默认）

2、基于JDBC

3、基于可复制的LevelDB

说明：

5.6 版本之后推出了LevelDB的持久化引擎，它使用了自定义的索引代替常用的Btree索引，其持久化性能高于KahaDB，虽然默认的持久化方式还是KahaDB，但是 LevelDB 可能会是趋势。

在5.9版本还提供了基于 LevelDB 和 Zookeeper的数据复制方式，作为 Master-Slave 方式的首选数据复制方案。

6.2、搭建集群测试

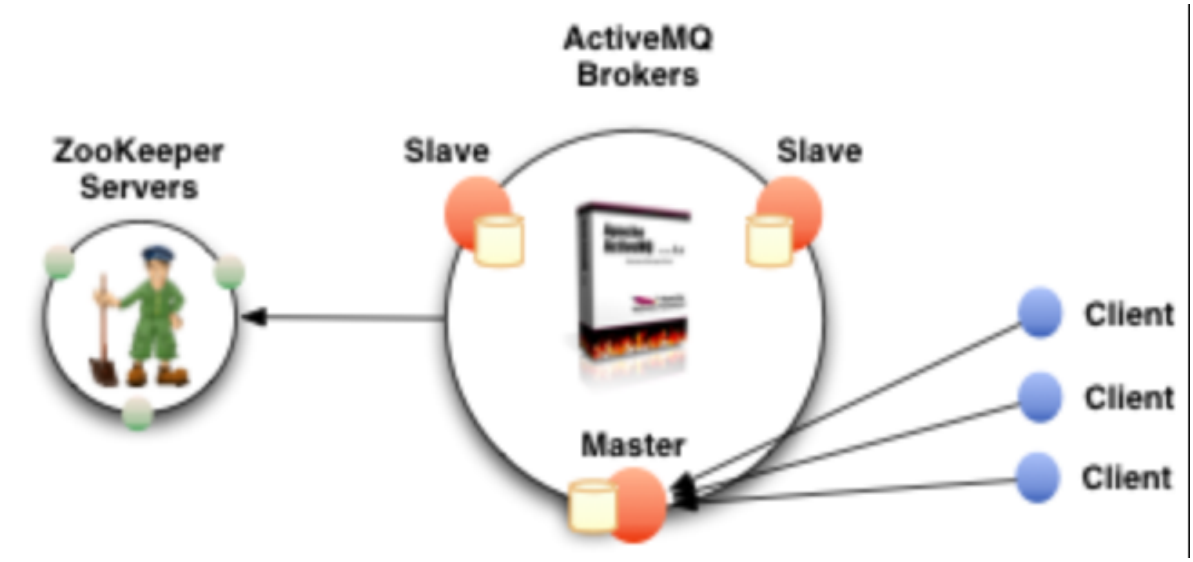
> 说明

从 ActiveMQ 5.9 开始，ActiveMQ 的集群实现方式取消了传统的Master-Slave 方式。

增加了基于 Zookeeper + LevelDB的 Master-Slave实现方式，从5.9版本后也是官网的推荐。

基于 Zookeeper 和 LevelDB 搭建 ActiveMQ 集群。集群仅提供主备方式的高可用集群功能，避免单点故障。

官网集群原理图：http://activemq.apache.org/replicated-leveldb-store



.

原理说明：

使用Zookeeper集群注册所有的 ActiveMQ Broker，但只有其中的一个Broker可以提供服务，它将被视为Master。其他的Broker处于待机状态被视为 Slave。

如果 Master 因故障而不能提供服务，Zookeeper会从 Slave 中选举出一个 Broker 充当 Master。

Slave 连接 Master 并同步他们的存储状态，Slave 不接受客户端连接。所有的存储操作都会被复制到连接至Master 的 Slaves。

如果 Master 宕机，得到了最新更新的 Slave 会成为 Master。故障节点在恢复后会重新加入到集群中并连接 Master 进入 Slaves 模式。

所有需要同步的消息操作，都将等待存储状态被复制到其他法定节点的操作完成才能完成。

所以，如果你配置了 replicas=3，那么法定大小是（3/2）+1 = 2。 Master 将会存储并更新然后等待（2-1）= 1 个 Slave 存储和更新完成，才汇报 success。至于为什么是 2-1，Zookeeper 讲解过自行复习。

有一个 node 要作为观察者存在。当一个新的 Master 被选中，你需要至少保障一个法定 node 在线以能够找到拥有最新状态的node。这个node才可以成为新的 Master 。

因此，推荐运行至少3个 replica nodes 以防止一个 node 失败后服务中断。

> 搭建Zookeeper集群

--	服务端口	集群通信端口
zk1	2181	9000:7000
zk2	2182	9001:7001
zk3	2183	9002:7002

1、Zookeeper 下载：http://archive.apache.org/dist/zookeeper/zookeeper-3.5.7/

2、解压zk包，进入zookeeper-3.5.7\conf目录，修改zoo_sample.cfg文件为zoo.cfg.如果没有特殊需求，不需要修改配置文件，直接使用默认配置文件即可.

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
```

各个参数的意义：

`tickTime`：这个时间是作为 Zookeeper 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳 。
`dataDir`：顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里.
`clientPort`：这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。
`initLimit`：集群中的follower服务器(F)与leader服务器(L)之间初始连接时能容忍的最多心跳数（tickTime的数量）
`syncLimit`：集群中的follower服务器与leader服务器之间请求和应答之间能容忍的最多心跳数（tickTime的数量）。

启动zookeeper，windows下执行 bin >zkServer.cmd；linux下ookeeper-3.4.6\bin >zkServer.sh start 启动，确保不报错!

3、在zookeeper1的目录下新建一个data文件夹，里面新建一个没有后缀的myid文件，写入数字1

4、修改zookeeper1的配置文件。增加集群配置

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
```

```
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
# 数据目录切换到自己定义的的目录
dataDir=D:/Environment/zookeeper/apache-zookeeper-1/data
# the port at which the clients will connect
clientPort=2181
# 设置集群配置，格式： server.A = B:C:D
# A:是一个数字，表示第几号服务器
# B:服务器IP地址
# C:是一个端口号，用来集群成员的信息交换,表示这个服务器与集群中的leader服务器交换信息的端口
# D:是在leader挂掉时专门用来进行选举leader所用的端口

server.1=127.0.0.1:9000:7000
server.2=127.0.0.1:9001:7001
server.3=127.0.0.1:9002:7002
```

5、将zookeeper1复制3份，改名为 zookeeper2、zookeeper3

6、修改每个目录下的data文件中的myid文件，分别为2， 3

7、修改其与两个zookeeper节点的配置文件，主要是端口号，data目录，以及集群配置，同上！

```
# 数据目录切换到自己定义的的目录
dataDir=D:/Environment/zookeeper/apache-zookeeper-2/data
clientPort=2182
server.1=127.0.0.1:9000:7000
server.2=127.0.0.1:9001:7001
server.3=127.0.0.1:9002:7002
```

```
# 数据目录切换到自己定义的的目录
dataDir=D:/Environment/zookeeper/apache-zookeeper-3/data
clientPort=2183
server.1=127.0.0.1:9000:7000
server.2=127.0.0.1:9001:7001
server.3=127.0.0.1:9002:7002
```

8、启动顺序为server1、server2、server3。在启动server1，server2时zk会报错，当所有节点全部启动时错误会消失

9、可以通过Zk提供的简易客户端来进行验证，双击下 bin/zkCli.cmd来启动Zk简易客户端，然后通过使用`ls /`命名（列出Zk指定节点下的所有子节点）来验证Zk已经启动完成。

10、为了方便启动，我们还可以编写一个 bat 脚本文件一次性启动

```
start apache-zookeeper-1\bin\zkServer.cmd
start apache-zookeeper-2\bin\zkServer.cmd
start apache-zookeeper-3\bin\zkServer.cmd
```

> 搭建 activemq 集群

配置参考官方：http://activemq.apache.org/replicated-leveldb-store

--	服务端口	jetty控制台端口
node1	61616	8161
node2	61617	8162
node3	61618	8163

1、下载并解压 ActiveMQ，修改文件名为 apache-activemq-1，复制3份，1 2 3

2、修改3个节点的配置，保证BrokerName的名称一致，默认是localhost。

```
</bean>

<!--
    The <broker> element is used to configure the ActiveMQ broker.
-->
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="localhost" dataDirectory="${activemq.

    <destinationPolicy>
```

3、3个节点的持久化配置、然后端口需要修改61616端口为 61617 和 61618

```
<persistenceAdapter>
    <replicatedLevelDB
                        directory="${activemq.data}/leveldb"
                        replicas="3"
                        bind="tcp://0.0.0.0:0"

zkAddress="127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183"
                        hostname="broker-ms"
                        zkPath="/activemq/leveldb-stores"
                        sync="local_disk"
                        />
</persistenceAdapter>
```

```
    -->
    <transportConnectors>
        <!-- DOS protection, limit concurrent connections to 1000 and frame size to 100MB -->
        <transportConnector name="openwire" uri="tcp://0.0.0.0:61617?maximumConnections=1000&amp;wireForm
        <transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wireFormat.m
        <transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;wireForma
        <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wireFormat.m
        <transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFormat.maxF
    </transportConnectors>

    <!-- destroy the spring context on shutdown to stop jetty -->
```

```
    -->
    <transportConnectors>
        <!-- DOS protection, limit concurrent connections to 1000 and frame size to 100MB -->
        <transportConnector name="openwire" uri="tcp://0.0.0.0:61618?maximumConnections=1000&amp;wireForm
        <transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wireFormat.m
        <transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;wireForma
        <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wireFormat.m
        <transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFormat.maxF
    </transportConnectors>
```

4、jetty控制台端口号修改，修改jetty.xml 中的端口配置

```
</bean>

<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>
```

5、为了方便启动，我们可以编写一个启动脚本来启动3个 activemq

```
start apache-activemq-1\bin\activemq.bat start
start apache-activemq-2\bin\activemq.bat start
start apache-activemq-3\bin\activemq.bat start
```

> 启动测试集群

1、启动zk集群

2、启动ActiveMQ集群

3、进入zk客户端，查看

```
ls /
=  [activemq, zookeeper]
ls /activemq/leveldb-stores
=  [00000000000, 00000000001, 00000000002]
get /activemq/leveldb-stores/00000000000
get /activemq/leveldb-stores/00000000001
get /activemq/leveldb-stores/00000000002
```

```
WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] ls /
[activemq, zookeeper]
[zk: localhost:2181(CONNECTED) 1] ls /activemq/leveldb-stores
[00000000000, 00000000001, 00000000002]
[zk: localhost:2181(CONNECTED) 2] get /activemq/leveldb-stores/00000000000
{"id":"localhost","container":null,"address":"tcp://broker-ms:56967","position":-1,"weight":1,"elected":"0000000000"}
[zk: localhost:2181(CONNECTED) 3] get /activemq/leveldb-stores/00000000001
{"id":"localhost","container":null,"address":null,"position":-1,"weight":1,"elected":null}
[zk: localhost:2181(CONNECTED) 4] get /activemq/leveldb-stores/00000000002
{"id":"localhost","container":null,"address":null,"position":-1,"weight":1,"elected":null}
[zk: localhost:2181(CONNECTED) 5]
```

Master

4、连接测试发现，只有Master节点的ActiveMQ才能连接通，其与两个结点无法通过

说明：elected 不为null 就是 Master 节点

> 代码执行测试

一定要先启动集群环境

Deployment Tips

Clients should be using the Failover Transport to connect to the broker nodes in the replication cluster. e.g. using a URL something like the following:

客户端连接的的URL变化

```
failover:(tcp://broker1:61616,tcp://broker2:61616,tcp://broker3:61616)
```

You should run at least 3 ZooKeeper server nodes so that the ZooKeeper service is highly available. Don't overcommit your ZooKeeper servers. An overworked ZooKeeper might start thinking live replication nodes have gone offline due to delays in processing their 'keep-alive' messages.

注意：URL一定要改为 支持故障转移传输的URL

1、消息生产者

```
public class JmsProduce {

    public static final String ACTIVEMQ_URL = "failover:(tcp://127.0.0.1:61616,tcp://127.0.0.1:61617,tcp://127.0.0.1:61618)";
```

2、消息消费者

```
public class JmsConsumer {

    public static final String ACTIVEMQ_URL = "failover:(tcp://127.0.0.1:61616,tcp://127.0.0.1:61617,tcp://127.0.0.1:61618)";
```

3、停止一个activemq的主节点，它会自动切换到另一个或者的节点

> 说明及总结

ActiveMQ 的客户端只能访问 Master 的 Broker，其他处于Slave的Broker不能访问，所以客户端连接的Broker应该使用 failover协议，即故障转移协议；

当一个ActiveMQ节点挂掉或者一个Zookeeper节点挂掉，ActiveMQ服务依然正常运转，如果仅剩一个ActiveMQ节点，由于不能选举Master，所以ActiveMQ不能正常运行；

同样的，如果Zookeeper仅剩一个节点活动，不管ActiveMQ各节点存活，ActiveMQ也不能正常提供服务。

ActiveMQ集群的高可用依赖于Zookeeper集群的高可用。

7、提高

> 常见面试题：引入消息队列之后该如何保证其高可用性

zookeeper + replicated-leveldb-store 的主从集群；

7.1、异步投递

> 说明

官网：http://activemq.apache.org/async-sends

背景

ActiveMQ支持以同步或异步模式将消息发送到代理。使用的模式对发送呼叫的延迟有很大的影响。由于延迟通常是生产者可以实现的吞吐量的重要因素，因此使用异步发送可以显着提高系统的性能。

好消息是，在某些情况下，ActiveMQ默认情况下以异步模式发送消息。只有在JMS规范要求使用同步发送的情况下，我们才默认使用同步发送。我们被迫以同步模式发送的情况是在事务外部发送持久性消息时。

如果您不使用事务并且正在发送持久消息，则每次发送都将同步并阻塞，直到代理已向生产者发送回确认消息已将消息安全保存到磁盘为止。该ack保证了消息不会丢失，但是由于客户端被阻止，因此还付出了巨大的等待时间。

许多高性能应用程序旨在承受故障情况下的少量消息丢失。如果您的应用程序是按照这种方式设计的，则即使使用持久性消息，也可以启用异步发送来提高吞吐量。

对于一个 Slow Consumer，使用同步发送消息可能出现 Producer 堵塞等情况，慢消费者适合使用异步发送。

> 什么是异步投递

ActiveMQ 支持同步，异步两种发送的模式将消息发送到 broker，模式的选择对发送延时有巨大的影响。producer 能达到怎样的产出率 （产出率=发送数据总量/时间）主要受发送延时的影响，使用异步发送可以显著的提高发送的性能。

ActiveMQ默认使用异步发送的模式：除非明确指定使用同步发送的方式或者在未使用事务的前提下发送持久化的消息，这两种情况都是同步发送的。

如果你**没有使用事务且发送的是持久化的消息**，每一次发送都是同步发送的且会阻塞 producer ，直到 broker 返回一个确认，表示消息已经被安全的持久化到磁盘。确认机制提供了消息安全的保障。但同时会阻塞客户端，带来了很大的延时。

很多高性能的应用，允许在失败的情况下有少量的数据丢失。如果你的应用满足这个特点，你可以使用异步发送来提高生产率，即使发送的是持久化的消息。

异步发送

它可以最大化producer端的发送效率。我们通常在发送消息量比较密集的情况下使用异步发送，它可以很大的提升Producer性能；不过这也带来了额外的问题。就是需要消耗较多的Client端内存同时也会导致 broker 端性能消耗增加；

此外它不能有效的确保消息的发送成功。在`useAsyncSend=true`的情况下，客户端需要容忍消息丢失的可能。

> 如何配置

1、使用连接URI配置异步发送

```
cf = new ActiveMQConnectionFactory("tcp://locahost:61616?
jms.useAsyncSend=true");
```

2、在ConnectionFactory级别配置异步发送

```
((ActiveMQConnectionFactory)connectionFactory).setUseAsyncSend(true);
```

3、在连接级别配置异步发送

```
((ActiveMQConnection)connection).setUseAsyncSend(true);
```

> 面试题：异步发送如何确认发送成功?

异步发送丢失消息的场景是：生产者设置 UseAsyncSend=true，使用producer.send(msg) 持续发送消息。

由于消息不阻塞，生产者会认为所有send的消息均被发送至 MQ。

如果MQ突然宕机，此时生产者端内存中尚未被发送至MQ的消息都会丢失。

所以，正确的异步发送方法是需要接收回调的

同步发送和异步发送的区别就在于此，同步发送等 send 不阻塞了就表示一定发送成功了，异步发送需要接收回执并由客户端再判断一次是否发送成功。

代码测试说明：

```java
// 修改MessageProducer 为 ActiveMQMessageProducer 细粒度操作
ActiveMQMessageProducer producer = (ActiveMQMessageProducer)
session.createProducer(queue);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

for (int i = 1; i <= 3; i++) {

    final TextMessage textMessage = session.createTextMessage("msg=>" + i);
    // 给消息设置一个编号
    textMessage.setJMSMessageID(UUID.randomUUID().toString()+"<= myid");
    String jmsMessageID = textMessage.getJMSMessageID();
    // 通过生产者发布，发送的时候带上异步回调
    producer.send(textMessage, new AsyncCallback() {
        public void onSuccess() {
            System.out.println(jmsMessageID + "发送OK");
        }

        public void onException(JMSException e) {
            System.out.println(jmsMessageID + "发送失败");
        }
    });
}
```

7.2、定时延时投递

> 说明

官网：http://activemq.apache.org/delay-and-schedule-message-delivery.html

ACTIVEMQ 家 组件 联系 阿帕奇

5.4版的ActiveMQ在ActiveMQ消息代理中内置了一个可选的持久性调度程序。通过在Xml Configuration中将broker **schedulerSupport**属性设置为true可以启用它。ActiveMQ客户端可以通过使用以下消息属性来利用延迟传递：

检查您的邮件属性

消息属性`scheduledJobId` 保留供作业计划程序使用。如果在发送之前设置了此属性，则消息将立即发送，而不是计划的。另外，在接收到计划的消息之后，`scheduledJobId` 将在接收到的消息上设置属性，因此，如果使用像骆驼之路这样的东西，在重新发送消息时可能会自动复制属性，请记住这一点。

物业名称	类型	描述
AMQ_SCHEDULED_DELAY	长	在代理计划将消息传递之前，消息将等待的时间（以毫秒为单位）
AMQ_SCHEDULED_PERIOD	长	在开始时间之后再等待调度消息之前要等待的时间（以毫秒为单位）
AMQ_SCHEDULED_REPEAT	整型	重复安排邮件传递的次数
AMQ_SCHEDULED_CRON	串	使用Cron条目设置时间表

需要修改xml配置:将broker schedulerSupport属性设置为true

为了增强Java JMS客户端的性能，在`org.apache.activemq.ScheduledMessage`上有一个带有用于调度的属性名称的接口。

例如，要安排在60秒内发送邮件-您需要设置*AMQ_SCHEDULED_DELAY*属性：

```
MessageProducer producer = session.createProducer(destination);
TextMessage message = session.createTextMessage("test msg");
long time = 60 * 1000;
message.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_DELAY, time);
producer.send(message);
```

您可以将消息设置为等待初始延迟，然后重复发送10次，每次重新发送之间等待10秒：

```
MessageProducer producer = session.createProducer(destination);
TextMessage message = session.createTextMessage("test msg");
long delay = 30 * 1000;
long period = 10 * 1000;
int repeat = 9;
message.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_DELAY, delay);
message.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_PERIOD, period);
message.setIntProperty(ScheduledMessage.AMQ_SCHEDULED_REPEAT, repeat);
producer.send(message);
```

您还可以使用CRON安排邮件的时间，例如，如果希望将邮件安排为每小时发送一次，则需要将CRON条目设置为`0 * * * *`--例如

```
MessageProducer producer = session.createProducer(destination);
TextMessage message = session.createTextMessage("test msg");
message.setStringProperty(ScheduledMessage.AMQ_SCHEDULED_CRON, "0 * * * *");
producer.send(message);
```

CRON调度优先于使用消息延迟-但是，如果为CRON条目设置了重复和延时，则ActiveMQ调度程序将在每次CRON条目触发时调度消息的传递。用一个例子更容易解释。假设您希望传递一条消息10次，每条消息之间有一秒钟的延迟-并且您希望每小时发生一次-您可以这样做：

```
MessageProducer producer = session.createProducer(destination);
TextMessage message = session.createTextMessage("test msg");
message.setStringProperty(ScheduledMessage.AMQ_SCHEDULED_CRON, "0 * * * *");
message.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_DELAY, 1000);
message.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_PERIOD, 1000);
message.setIntProperty(ScheduledMessage.AMQ_SCHEDULED_REPEAT, 9);
producer.send(message);
```

7.3、消息重发

> 常见面试题

具体哪些情况会引起消息重发?

1、Client 用了 transactions 且在 session 中调用了 rollback()

2、Client 用了 transactions 且在 调用 commit() 之前关闭或没有 commit()

3、Client在Client_Ackonwledge的传递模式下，在 session 中调用了 recover()

请说说消息重发时间间隔和重发次数吗?

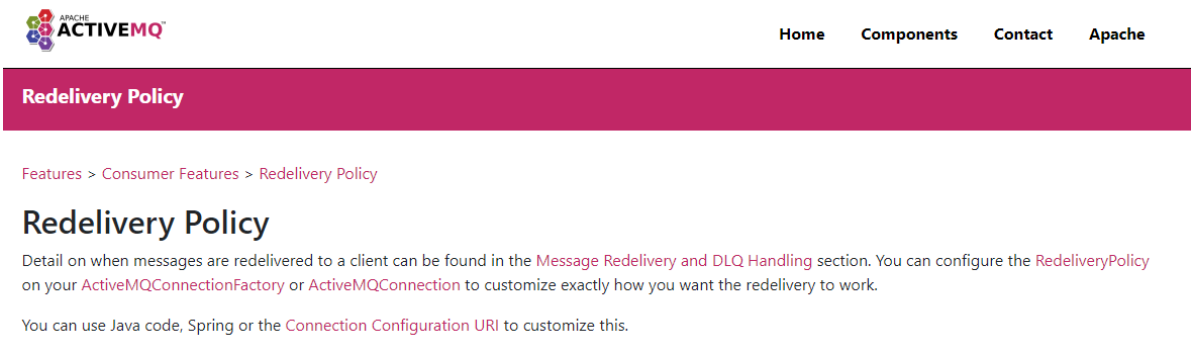
间隔：1， 次数：6

有毒消息Poison Ack 谈谈你的理解?

一个消息被 redelivedred 超过默认的最大重发次数（默认6次）时，消费端会给MQ发送一个 “ poison Ack” 表示这个消息有毒，告诉 broker 不要再发了。这个时候 broker 会把这个消息放到 DLQ （死信队列）

官网说明

官网：http://activemq.apache.org/redelivery-policy



属性说明：

1：collisionAvoidanceFactor：设置防止冲突范围的正负百分比，只有启用useCollisionAvoidance参数时才生效。也就是在延迟时间上再加一个时间波动范围。默认值为0.15
2：maximumRedeliveries：最大重传次数，达到最大重连次数后抛出异常。为-1时不限制次数，为0时表示不进行重传。默认值为6。
3：maximumRedeliveryDelay：最大传送延迟，只在useExponentialBackOff为true时有效（V5.5），假设首次重连间隔为10ms，倍数为2，那么第二次重连时间间隔为 20ms，第三次重连时间间隔为40ms，当重连时间间隔大的最大重连时间间隔时，以后每次重连时间间隔都为最大重连时间间隔。默认为-1。
4：initialRedeliveryDelay：初始重发延迟时间，默认1000L
5：redeliveryDelay：重发延迟时间，当initialRedeliveryDelay=0时生效，默认1000L
6：useCollisionAvoidance：启用防止冲突功能，默认false
7：useExponentialBackOff：启用指数倍数递增的方式增加延迟时间，默认false
8：backOffMultiplier：重连时间间隔递增倍数，只有值大于1和启用useExponentialBackOff参数时才生效。默认是5

代码测试

1、生成者正常发送消息

2、消费者需要开启提交事务，但是不要commit(); 则会产生重复接收问题。6次之后则接收不到消息；

自定义修改消息重发策略

从ActiveMQ v5.7.0开始，您现在可以基于每个目标配置RedeliveryPolicy。`ActiveMQConnection`现在， 工厂类公开了RedeliveryPolicyMap属性，该属性允许使用命名目标或目标通配符来分配RedeliveryPolicy。下面的代码片段显示了如何为主题和队列配置不同的RedeliveryPolicy。

```
1  ActiveMQConnection connection ...  // Create a connection
```

```
2
3  RedeliveryPolicy queuePolicy = new RedeliveryPolicy();
4  queuePolicy.setInitialRedeliveryDelay(0);
5  queuePolicy.setRedeliveryDelay(1000);
6  queuePolicy.setUseExponentialBackOff(false);
7  queuePolicy.setMaximumRedeliveries(2);
8
9  RedeliveryPolicy topicPolicy = new RedeliveryPolicy();
10 topicPolicy.setInitialRedeliveryDelay(0);
11 topicPolicy.setRedeliveryDelay(1000);
12 topicPolicy.setUseExponentialBackOff(false);
13 topicPolicy.setMaximumRedeliveries(3);
14
15 // Receive a message with the JMS API
16 RedeliveryPolicyMap map = connection.getRedeliveryPolicyMap();
17 map.put(new ActiveMQTopic(">"), topicPolicy);
18 map.put(new ActiveMQQueue(">"), queuePolicy);
```

7.4、死信队列

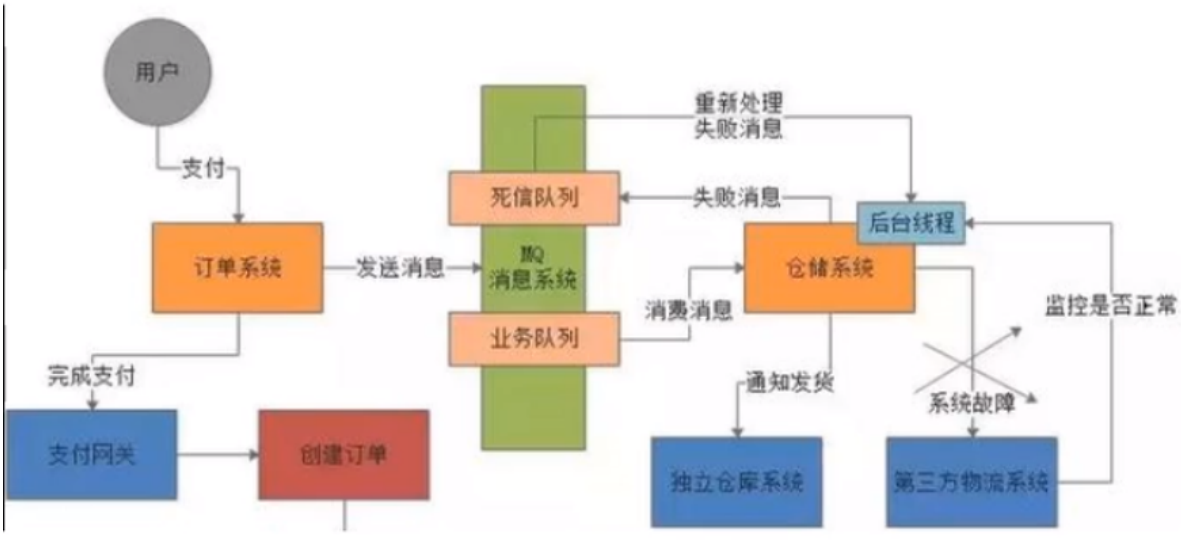
> 什么是死信队列

ActiveMQ 中引入了 “死信队列” （Dead Letter Queue）的概念。

即一条消息在被重发了多次后（默认是6次），将会被ActiveMQ移入 “死信队列”。开发人员可以在这个Queue中查看处理出错的消息，进行人工干预；

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
ActiveMQ.DLQ	3	0	3	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete
queue-redelivery	0	0	3	3	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

死信队列就是用来处理失败的消息



- 一般生产环境中，在使用MQ的时候设计两个队列：一个是核心业务队列，一个是死信队列；
- 核心业务队列，就是比如上图专门用来让订单系统发送订单消息的，然后另外一个死信队列就是用来处理异常情况的。

- 假如第三方物流系统故障了，此时无法请求。那么仓储系统每次消费到一条订单消息，尝试通知发货和配送都会遇到对方的接口报错。此时仓储系统就可以把这条消息拒绝访问或者标志为处理失败。一旦标志这条消息处理失败了后，MQ就会把这条消息转入提前设置好的一个死信队列中。然后你会看到的就是，在第三方物流系统故障期间，所有订单消息全部处理失败，全部会转入死信队列。然后你的仓储系统得专门有一个后台线程，监控第三方物流系统是否正常。能否请求的，不停的监视。一旦发现对方恢复正常，这个后台线程就从死信队列消费出来处理失败的订单，重新执行发货和配送的通知逻辑。

> 死信队列的配置介绍

SharedDeadLetterStrategy

将所有的 DeadLetter 保存在一个共享的队列中，这就是 ActiveMQ Broker 端默认的策略。

共享队列默认为 “ActiveMQ.DLQ”，可以通过 “deadLetterQueue” 属性来设定。

```
<deadLetterStrategy>
    <sharedDeadLetterStrategy deadLetterQueue="DLQ-QUEUE"/>
</deadLetterStrategy>
```

IndividualDeadLetterStrategy

把DeadLetter放入各自的死信通道中，

对于Queue而言，死信通道的前缀默认为 “ActiveMQ.DLQ.Queue.”；

对于Topic而言，死信通道的前缀默认为 “ActiveMQ.DLQ.Topic.”；

比如队列 Order，那么它对应的死信通道为 “ActiveMQ.DLQ.Queue.Order”。

我们使用 “queuePrefix” “topicPrefix” 来指定上述前缀。

默认情况下，无论是Topic 还是 Queue，broker 将使用 Queue 来保存 DeadLeader，即死信通道常常为Queue，不过开发者也可以指定为Topic。

```
<policyEntry queue="order">
    <deadLetterStartegy>
        <individualDeadLetterStrategy queuePrefix="DLQ."
useQueueForQueueMessages="false"/>
    </deadLetterStrategy>
</policyEntry>
```

将队列Order中出现的 DeadLetter 保存在DLQ.Order，不过此时 DLQ.Order 为 Topic。

属性 “useQueueForTopicMessages”，此值表示是否将Topic的DeadLetter 保存在 Queue 中，默认为 true；