

---

# ActiveMQ

ACTIVEMQ.....	1
ACTIVEMQ.....	3
一、 JMS 简介.....	3
二、 ACTIVEMQ 简介.....	3
1 Destination.....	4
2 Producer.....	4
3 Consumer 【Receiver】 .....	4
4 Message.....	4
5 ConnectionFactory.....	4
6 Connection.....	4
7 Session.....	5
8 Queue & Topic.....	5
9 PTP.....	5
10 PUB & SUB.....	5
三、 ACTIVEMQ 安装.....	5
1 下载资源.....	5
1.1 版本说明.....	6
2 上传至 Linux 服务器.....	6
3 解压安装文件.....	6
4 检查权限.....	6
5 复制应用至本地目录.....	6
6 配置文件简介.....	6
6.1 activemq.xml.....	7
6.2 jetty.xml.....	7
6.3 users.properties.....	7
7 启动 ActiveMQ.....	7
8 测试 ActiveMQ.....	7
8.1 检查进程.....	7
8.2 管理界面.....	7
8.3 修改访问端口.....	8
9 重启 ActiveMQ.....	8
10 关闭 ActiveMQ.....	8
四、 ACTIVEMQ 应用.....	8
1 PTP 处理模式 (Queue) .....	8
2 Publish/Subscribe 处理模式 (Topic) .....	9
3 PTP 和 PUB/SUB 简单对比.....	10
五、 ACTIVEMQ 安全认证.....	10
六、 ACTIVEMQ 的持久化.....	11
1 kahadb 方式.....	12
2 AMQ 方式.....	12

---

3	JDBC 持久化方式.....	12
七、	API 简介.....	14
1	Producer API 简介.....	14
1.1	发送消息.....	14
1.2	消息有效期.....	14
1.3	消息优先级.....	14
1.3.1	开启.....	15
1.3.2	强顺序.....	15
1.3.3	严格顺序.....	15
2	Consumer API 简介.....	16
2.1	消息的过滤.....	16
八、	SPRING&ACTIVE MQ.....	16
九、	ACTIVE MQ 集群.....	16
1	Master-Slave.....	16
1.1	安装 ZooKeeper.....	17
1.1.1	解压缩.....	17
1.1.2	复制.....	17
1.1.3	创建 data 数据目录.....	17
1.1.4	编写 Zookeeper 配置文件.....	17
1.1.5	复制两份同样的 Zookeeper.....	18
1.1.6	为 Zookeeper 服务增加服务命名.....	18
1.1.7	修改 Zookeeper 配置文件 zoo.cfg.....	18
1.1.8	启动 Zookeeper 测试.....	18
1.2	安装 ActiveMQ.....	18
1.2.1	安装 ActiveMQ 实例.....	18
1.2.2	修改配置信息.....	19
1.3	启动主从.....	20
1.4	查看主从状态.....	20
1.4.1	使用客户端连接 ZooKeeper.....	20
1.4.2	查看状态信息.....	20
2	集群.....	21

---

# ActiveMQ

## 一、 JMS 简介

全称：Java Message Service 中文：Java 消息服务。

JMS 是 Java 的一套 API 标准，最初的目的是为了使用应用程序能够访问现有的 MOM 系统（MOM 是 Message Oriented Middleware 的英文缩写，指的是利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行分布式系统的集成。）；后来被许多现有的 MOM 供应商采用，并实现为 MOM 系统。【常见 MOM 系统包括 Apache 的 ActiveMQ、阿里巴巴的 RocketMQ、IBM 的 MQSeries、Microsoft 的 MSMQ、BEA 的 RabbitMQ 等。（并非全部 MOM 系统都遵循 JMS 规范）】

基于 JMS 实现的 MOM，又被称为 JMS Provider。

“消息”是在两台计算机间传送的数据单位。消息可以非常简单，例如只包含文本字符串；也可以更复杂，可能包含嵌入对象。

消息被发送到队列中。“消息队列”是在消息的传输过程中保存消息的容器。消息队列管理器在将消息从它的源中继到它的目标时充当中间人。队列的主要目的是提供路由并保证消息的传递；如果发送消息时接收者不可用，消息队列会保留消息，直到可以成功地传递它。

消息队列的主要特点是异步处理，主要目的是减少请求响应时间和解耦。所以主要的使用场景就是将比较耗时而且不需要即时（同步）返回结果的操作作为消息放入消息队列。同时由于使用了消息队列，只要保证消息格式不变，消息的发送方和接收方并不需要彼此联系，也不需要受对方的影响，即解耦和。如：

跨系统的异步通信，所有需要异步交互的地方都可以使用消息队列。就像我们除了打电话（同步）以外，还需要发短信，发电子邮件（异步）的通讯方式。

多个应用之间的耦合，由于消息是平台无关和语言无关的，而且语义上也不再是函数调用，因此更适合作为多个应用之间的松耦合的接口。基于消息队列的耦合，不需要发送方和接收方同时在线。

在企业应用集成（EAI）中，文件传输，共享数据库，消息队列，远程过程调用都可以作为集成的方法。

应用内的同步变异步，比如订单处理，就可以由前端应用将订单信息放到队列，后端应用从队列里依次获得消息处理，高峰时的大量订单可以积压在队列里慢慢处理掉。由于同步通常意味着阻塞，而大量线程的阻塞会降低计算机的性能。

消息驱动的架构（EDA），系统分解为消息队列，和消息制造者和消息消费者，一个处理流程可以根据需要拆成多个阶段（Stage），阶段之间用队列连接起来，前一个阶段处理的结果放入队列，后一个阶段从队列中获取消息继续处理。

应用需要更灵活的耦合方式，如发布订阅，比如可以指定路由规则。

跨局域网，甚至跨城市的通讯，比如北京机房与广州机房的应用程序的通信。

## 二、 ActiveMQ 简介

**多种语言和协议编写客户端。** 语言: Java,C,C++,C#,Ruby,Perl,Python,PHP。应用协议:

---

OpenWire,Stomp REST,WS Notification,XMPP,AMQP

完全支持 JMS1.1 和 J2EE 1.4 规范（持久化，XA 消息，事务）

**对 Spring 的支持**，ActiveMQ 可以很容易内嵌到使用 Spring 的系统里面去

通过了常见 J2EE 服务器（如 Geronimo,JBoss 4,GlassFish,WebLogic)的测试，其中通过 JCA 1.5 resource adaptors 的配置，可以让 ActiveMQ 可以自动的部署到任何兼容 J2EE 1.4 商业服务器上

**支持多种传送协议**：in-VM,TCP,SSL,NIO,UDP,JGroups,JXTA

支持通过 JDBC 和 journal 提供高速的消息持久化

**从设计上保证了高性能的集群**，客户端-服务器，点对点

支持 Ajax

支持与 Axis 的整合, WebServices

可以很容易的调用内嵌 JMS provider，进行测试

## 1 Destination

目的地，JMS Provider（消息中间件）负责维护，用于对 Message 进行管理的对象。MessageProducer 需要指定 Destination 才能发送消息，MessageConsumer 需要指定 Destination 才能接收消息。

## 2 Producer

消息生成者(客户端、生成消息)，负责发送 Message 到目的地。应用接口为 MessageProducer。在 JMS 规范中，所有的标准定义都在 javax.jms 包中。

## 3 Consumer【Receiver】

消息消费者（处理消息），负责从目的地中消费【处理|监听|订阅】Message。应用接口为 MessageConsumer

## 4 Message

消息(Message)，消息封装一次通信的内容。常见类型有：StreamMessage、BytesMessage、TextMessage、ObjectMessage、MapMessage。

## 5 ConnectionFactory

链接工厂，用于创建链接的工厂类型。注意，不能和 JDBC 中的 ConnectionFactory 混淆。

## 6 Connection

链接。用于建立访问 ActiveMQ 连接的类型，由链接工厂创建。注意，不能和 JDBC 中的 Connection 混淆。

---

## 7 Session

会话，一次 **持久有效有状态** 的访问。由链接创建。是具体操作消息的基础支撑。

## 8 Queue & Topic

Queue 是队列目的地，Topic 是主题目的地。都是 Destination 的子接口。

Queue 特点：队列中的消息，默认只能由唯一的一个消费者处理。一旦处理消息删除。

Topic 特点：主题中的消息，会发送给所有的消费者同时处理。只有在消息可以重复处理的业务场景中可使用。

## 9 PTP

Point to Point。点对点消息模型。就是基于 Queue 实现的消息处理方式。

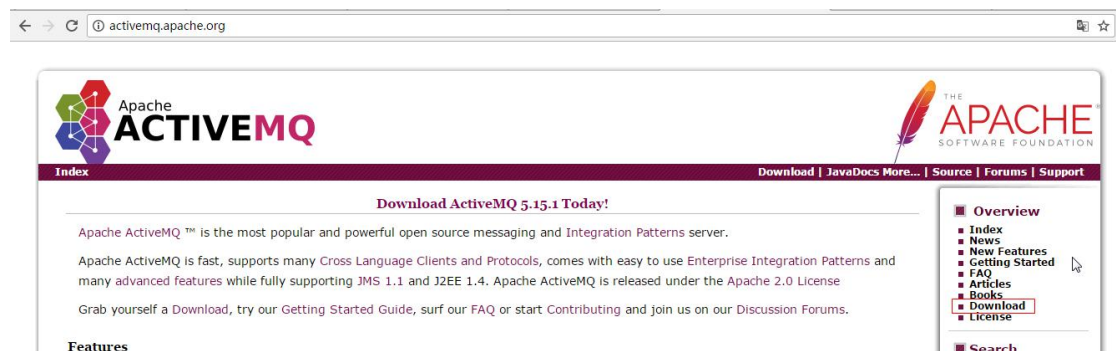
## 10 PUB & SUB

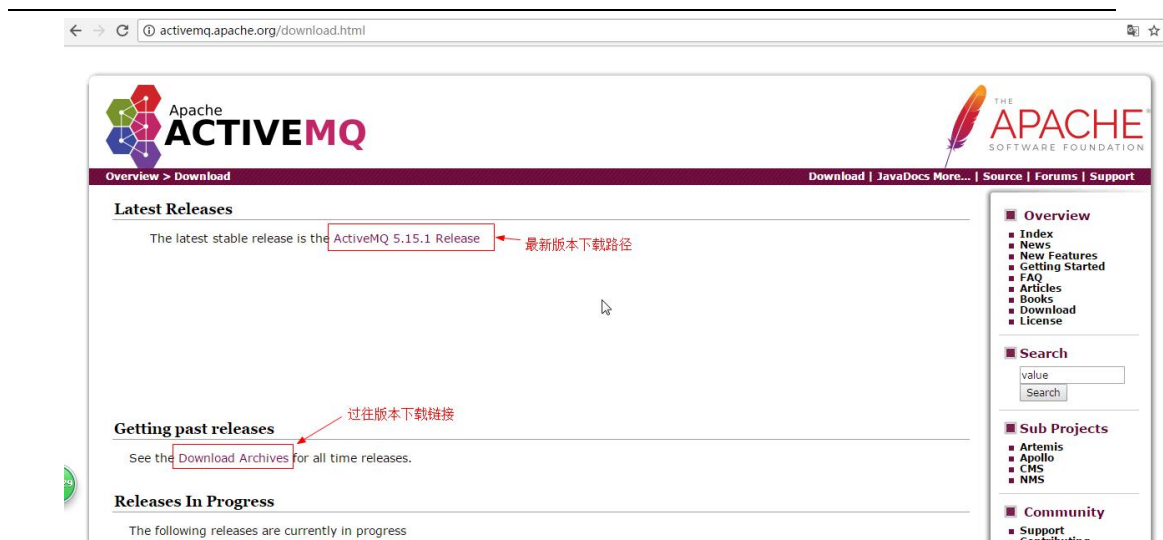
Publish & Subscribe 。消息的发布/订阅模型。是基于 Topic 实现的消息处理方式。

# 三、 ActiveMQ 安装

## 1 下载资源

ActiveMQ 官网： <http://activemq.apache.org>





## 1.1 版本说明

ActiveMQ5.10.x 以上版本必须使用 JDK1.8 才能正常使用。

ActiveMQ5.9.x 及以下版本使用 JDK1.7 即可正常使用。

## 2 上传至 Linux 服务器

## 3 解压安装文件

```
tar -zxf apache-activemq-5.9.0-bin.tar.gz
```

## 4 检查权限

```
ls -al apache-activemq-5.9.0/bin
```

如果权限不足,则无法执行,需要修改文件权限:

```
chmod 755 activemq
```

## 5 复制应用至本地目录

```
cp -r apache-activemq-5.9.0 /usr/local/activemq
```

## 6 配置文件简介

/usr/local/activemq/conf/\* - 配置文件.

需要关注的配置文件有: activemq.xml, jetty.xml, users.properties

任何配置文件修改后,必须重启 ActiveMQ,才能生效.

---

## 6.1 activemq.xml

就是 spring 配置文件. 其中配置的是 ActiveMQ 应用使用的默认对象组件.

transportConnectors 标签 - 配置链接端口信息的. 其中的端口号 61616 是 ActiveMQ 对外发布的 tcp 协议访问端口. 就是 java 代码访问 ActiveMQ 时使用的端口.

## 6.2 jetty.xml

spring 配置文件, 用于配置 jetty 服务器的默认对象组件.

jetty 是类似 tomcat 的一个中间件容器.

ActiveMQ 默认支持一个网页版的服务查看站点. 可以实现 ActiveMQ 中消息相关数据的页面查看.

8161 端口, 是 ActiveMQ 网页版管理站点的默认端口.

在 ActiveMQ 网页版管理站点中,需要登录, 默认的用户名和密码都是 admin.

## 6.3 users.properties

内容信息: 用户名=密码

是用于配置客户端通过协议访问 ActiveMQ 时,使用的用户名和密码.

## 7 启动 ActiveMQ

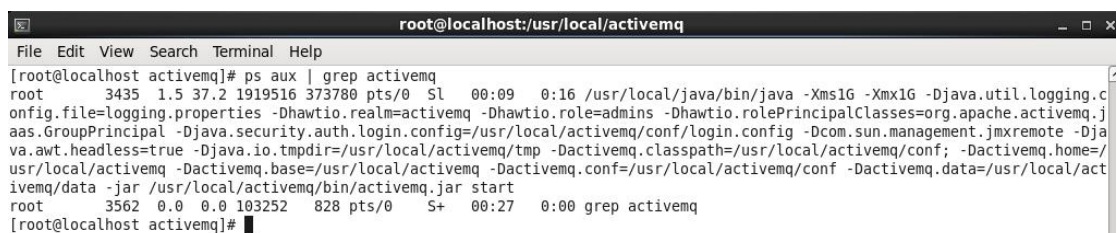
/usr/local/activemq/bin/activemq start

## 8 测试 ActiveMQ

### 8.1 检查进程

ps aux | grep activemq

见到下述内容即代表启动成功



```
root@localhost:usr/local/activemq
File Edit View Search Terminal Help
[root@localhost activemq]# ps aux | grep activemq
root      3435  1.5 37.2 1919516 373780 pts/0    Sl   00:09   0:16 /usr/local/java/bin/java -Xms1G -Xmx1G -Djava.util.logging.c
onfig.file=logging.properties -Dhawtio.realm=activemq -Dhawtio.role=admin -Dhawtio.rolePrincipalClasses=org.apache.activemq.j
aas.GroupPrincipal -Djava.security.auth.login.config=/usr/local/activemq/conf/login.config -Dcom.sun.management.jmxremote -Dja
va.awt.headless=true -Djava.io.tmpdir=/usr/local/activemq/tmp -Dactivemq.classpath=/usr/local/activemq/conf; -Dactivemq.home=/
usr/local/activemq -Dactivemq.base=/usr/local/activemq -Dactivemq.conf=/usr/local/activemq/conf -Dactivemq.data=/usr/local/act
ivemq/data -jar /usr/local/activemq/bin/activemq.jar start
root      3562  0.0  0.0 103252   828 pts/0    S+   00:27   0:00 grep activemq
[root@localhost activemq]#
```

### 8.2 管理界面

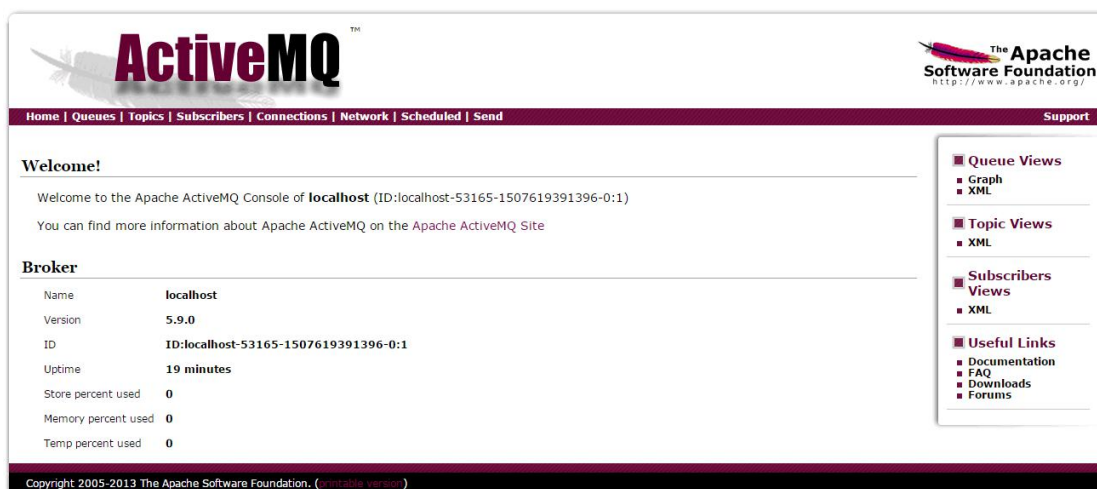
使用浏览器访问 ActiveMQ 管理应用, 地址如下:

<http://ip:8161/admin/>

用户名: admin

密码: admin

ActiveMQ 使用的是 jetty 提供 HTTP 服务.启动稍慢,建议短暂等待再访问测试.  
见到如下界面代表服务启动成功



## 8.3 修改访问端口

修改 ActiveMQ 配置文件: /usr/local/activemq/conf/jetty.xml

```
115 <!-- the default port number for the web console -->
116 <property name="port" value="8161"/>
117 </bean>
```

可以修改为其他可用端口

配置文件修改完毕, 保存并重新启动 ActiveMQ 服务。

## 9 重启 ActiveMQ

/usr/local/activemq/bin/activemq restart

## 10 关闭 ActiveMQ

/usr/local/activemq/bin/activemq stop

## 四、 ActiveMQ 应用

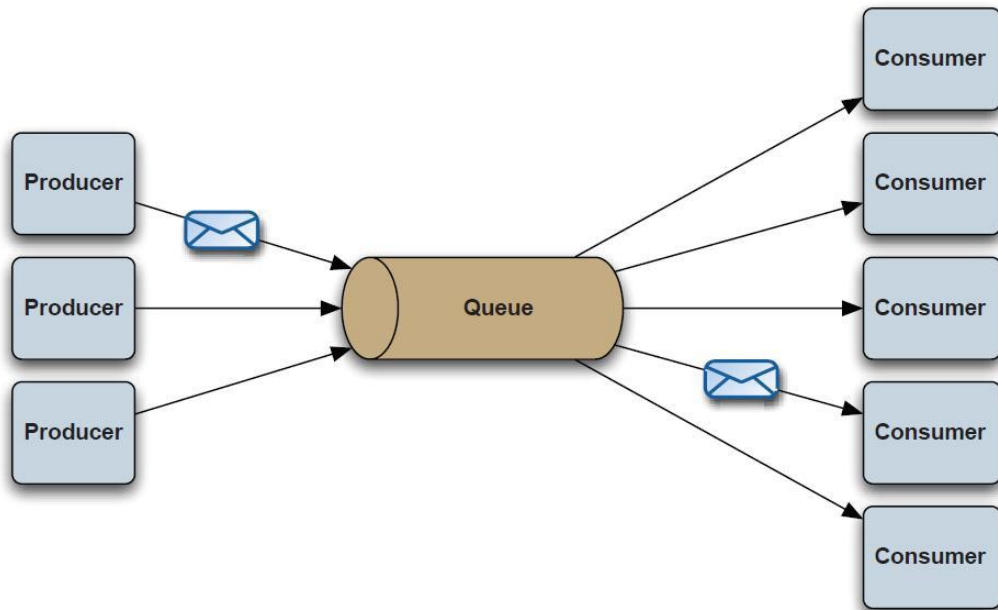
### 1 PTP 处理模式 (Queue)

消息生产者生产消息发送到 queue 中, 然后消息消费者从 queue 中取出并且消费消息。  
消息被消费以后, queue 中不再有存储, 所以消息消费者不可能消费到已经被消费的消息。

Queue 支持存在多个消费者, 但是对一个消息而言, 只会有一个消费者可以消费、其它的则不能消费此消息了。

当消费者不存在时, 消息会一直保存, 直到有消费消费

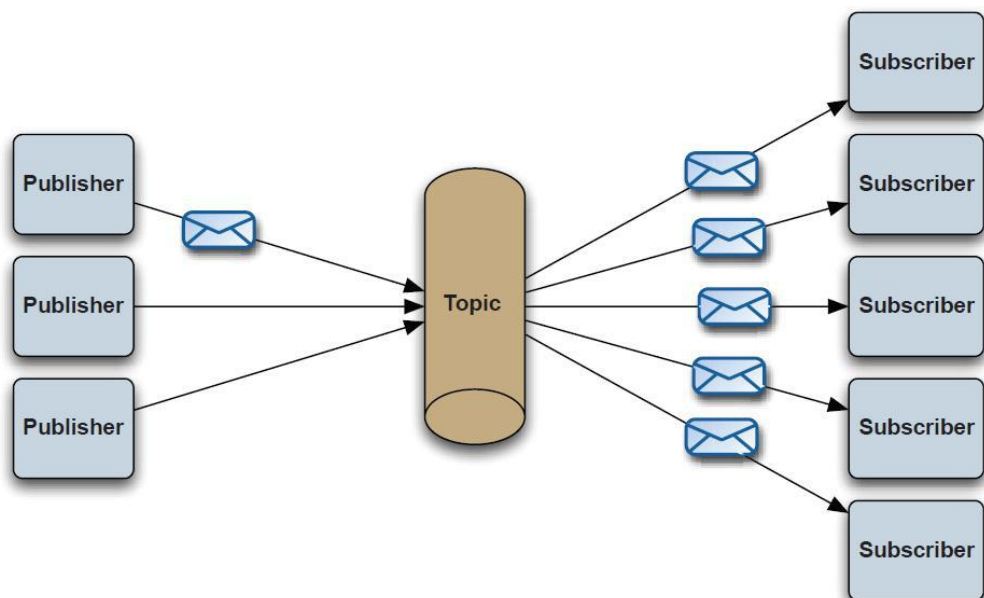




## 2 Publish/Subscribe 处理模式 (Topic)

消息生产者（发布）将消息发布到 topic 中，同时有多个消息消费者（订阅）消费该消息。

和点对点方式不同，发布到 topic 的消息会被所有订阅者消费。  
当生产者发布消息，不管是否有消费者。都不会保存消息  
一定要先有消息的消费者，后有消息的生产者。



### 3 PTP 和 PUB/SUB 简单对比

	Topic	Queue
概要	Publish Subscribe messaging 发布订阅消息	Point-to-Point 点对点
有无状态	topic 数据默认不落地，是无状态的。	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\kahadb 下面。也可以配置成 DB 存储。
完整性保障	并不保证 publisher 发布的每条数据，Subscriber 都能接受到。	Queue 保证每条数据都能被 receiver 接收。消息不超时。
消息是否会丢失	一般来说 publisher 发布消息到某一个 topic 时，只有正在监听该 topic 地址的 sub 能够接收到消息；如果没有 sub 在监听，该 topic 就丢失了。	Sender 发送消息到目标 Queue，receiver 可以异步接收这个 Queue 上的消息。Queue 上的消息如果暂时没有 receiver 来取，也不会丢失。前提是消息不超时。
消息发布接收策略	一对多的消息发布接收策略，监听同一个 topic 地址的多个 sub 都能收到 publisher 发送的消息。Sub 接收完通知 mq 服务器	一对一的消息发布接收策略，一个 sender 发送的消息，只能有一个 receiver 接收。receiver 接收完后，通知 mq 服务器已接收，mq 服务器对 queue 里的消息采取删除或其他操作。

## 五、 ActiveMQ 安全认证

ActiveMQ 也提供了安全认证。就是用户名密码登录规则。ActiveMQ 如果需要使用安全认证的话，必须在 activemq 的核心配置文件中开启安全配置。配置文件就是 conf/activemq.xml

在 conf/activemq.xml 配置文件的 **broker 标签**中增加下述内容。

<jaasAuthenticationPlugin configuration="activemq" />指定了使用 JAAS 插件管理权限，至于 configuration="activemq"是在 login.conf 文件里定义的

<authorizationEntry topic="名字" read="用户组名" write="用户组名" admin="用户组名" />指定了具体的 Topic/Queue 与用户组的授权关系

<authorizationEntry topic="ActiveMQ.Advisory.>" read="admins" write="admins" admin="admins"/>这个是必须的配置，不能少

```
<plugins>
    <!-- use JAAS to authenticate using the login.config file on the classpath to
    configure JAAS -->
    <!-- 添加 jaas 认证插件 activemq 在 login.config 里面定义,详细见 login.config-->
```

```

<jasAuthenticationPlugin configuration="activemq" />
  <!-- lets configure a destination based authorization mechanism -->
  <authorizationPlugin>
    <map>
      <authorizationMap>
        <authorizationEntries>
          <authorizationEntry      topic=">"      read="admins"
write="admins" admin="admins" />
          <authorizationEntry      queue=">"      read="admins"
write="admins" admin="admins" />
          <authorizationEntry      topic="ActiveMQ.Advisory.>"
read="admins" write="admins" admin="admins"/>
          <authorizationEntry      queue="ActiveMQ.Advisory.>"
read="admins" write="admins" admin="admins"/>
        </authorizationEntries>
      </authorizationMap>
    </map>
  </authorizationPlugin>
</plugins>

```

开启认证后，认证使用的用户信息由其他配置文件提供。

conf/login.config

```

activemq {
    org.apache.activemq.jaas.PropertiesLoginModule required
        org.apache.activemq.jaas.properties.user="users.properties"
        org.apache.activemq.jaas.properties.group="groups.properties";
};

```

user 代表用户信息配置文件，group 代表用户组信息配置文件。寻址路径为相对当前配置文件所在位置开始寻址。

conf/users.properties

```

#用户名=密码
admin=admin

```

conf/groups.properties

```

#用户组名=用户名，用户名
admins=admin

```

## 六、 ActiveMQ 的持久化

ActiveMQ 中，持久化是指对消息数据的持久化。在 ActiveMQ 中，默认的消息是保存在内存中的。当内存容量不足的时候，或 ActiveMQ 正常关闭的时候，会将内存中的未处理

的消息持久化到磁盘中。具体的持久化策略由配置文件中的具体配置决定。

ActiveMQ 的默认存储策略是 kahadb。如果使用 JDBC 作为持久化策略，则会将所有的需要持久化的消息保存到数据库中。

所有的持久化配置都在 conf/activemq.xml 中配置，配置信息都在 broker 标签内定义。

## 1 kahadb 方式

是 ActiveMQ 默认的持久化策略。kahadb 是一个文件型数据库。是使用内存+文件保证数据的持久化的。kahadb 可以限制每个数据文件的大小。不代表总计数据容量。

```
<persistenceAdapter>
  <!-- directory:保存数据的目录; journalMaxFileLength:保存消息的文件大小 -->
  <kahaDB directory="${activemq.data}/kahadb" journalMaxFileLength="16mb"/>
</persistenceAdapter>
```

特性是：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；

## 2 AMQ 方式

只适用于 5.3 版本之前。

AMQ 也是一个文件型数据库，消息信息最终是存储在文件中。内存中也会有缓存数据。

```
<persistenceAdapter>
  <!-- directory:保存数据的目录 ; maxFileLength:保存消息的文件大小 -->
  <amqpPersistenceAdapter directory="${activemq.data}/amq" maxFileLength="32mb"/>
</persistenceAdapter>
```

性能高于 JDBC，写入消息时，会将消息写入日志文件，由于是顺序追加写，性能很高。为了提升性能，创建消息主键索引，并且提供缓存机制，进一步提升性能。每个日志文件的大小都是有限制的（默认 32m，可自行配置）。

当超过这个大小，系统会重新建立一个文件。当所有的消息都消费完成，系统会删除这个文件或者归档。

主要的缺点是 AMQ Message 会为每一个 Destination 创建一个索引，如果使用了大量的 Queue，索引文件的大小会占用很多磁盘空间。

而且由于索引巨大，一旦 Broker（ActiveMQ 应用实例）崩溃，重建索引的速度会非常慢。

虽然 AMQ 性能略高于 Kaha DB 方式，但是由于其重建索引时间过长，而且索引文件占用磁盘空间过大，所以已经不推荐使用。

## 3 JDBC 持久化方式

ActiveMQ 将数据持久化到数据库中。不指定具体的数据库。可以使用任意的数据库中。本环节中使用 MySQL 数据库。

下述文件为 activemq.xml 配置文件部分内容。不要完全复制。

首先定义一个 mysql-ds 的 MySQL 数据源，然后在 persistenceAdapter 节点中配置 jdbcPersistenceAdapter 并且引用刚才定义的数据源。

dataSource 指定持久化数据库的 bean，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。

```
<broker brokerName="test-broker" persistent="true"
  xmlns="http://activemq.apache.org/schema/core">
  <persistenceAdapter>
    <jdbcPersistenceAdapter dataSource="#mysql-ds"
      createTablesOnStartup="false"/>
  </persistenceAdapter>
</broker>
<bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource"
  destroy-method="close">
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="url"
    value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/>
  <property name="username" value="activemq"/>
  <property name="password" value="activemq"/>
  <property name="maxActive" value="200"/>
  <property name="poolPreparedStatements" value="true"/>
</bean>
```

配置成功后，需要在数据库中创建对应的 database，否则无法访问。表格 ActiveMQ 可以自动创建。

activemq\_msgs 用于存储消息，Queue 和 Topic 都存储在这个表中：

ID：自增的数据库主键

CONTAINER：消息的 Destination

MSGID\_PROD：消息发送者客户端的主键

MSG\_SEQ：是发送消息的顺序，MSGID\_PROD+MSG\_SEQ 可以组成 JMS 的 MessageID

EXPIRATION：消息的过期时间，存储的是从 1970-01-01 到现在的毫秒数

MSG：消息本体的 Java 序列化对象的二进制数据

PRIORITY：优先级，从 0-9，数值越大优先级越高

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：

主要的数据库字段如下：

CONTAINER：消息的 Destination

SUB\_DEST：如果是使用 Static 集群，这个字段会有集群其他系统的信息

CLIENT\_ID：每个订阅者都必须有一个唯一的客户端 ID 用以区分

SUB\_NAME：订阅者名称

SELECTOR：选择器，可以选择只消费满足条件的消息。条件可以用自定义属性实现，可支持多属性 AND 和 OR 操作

LAST\_ACKED\_ID：记录消费过的消息的 ID。

---

表 `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，

其他的只能作为备份等待 Master Broker 不可用，才可能成为下一个 Master Broker。这个表用于记录哪个 Broker 是当前的 Master Broker。

只有在消息必须保证有效，且绝对不能丢失的时候。使用 JDBC 存储策略。

如果消息可以容忍丢失，或使用集群/主备模式保证数据安全的时候，建议使用 levelDB 或 Kahadb。

## 七、 API 简介

### 1 Producer API 简介

#### 1.1 发送消息

`MessageProducer`.

`send(Message message)`; 发送消息到默认目的地，就是创建 Producer 时指定的目的地。

`send(Destination destination, Message message)`; 发送消息到指定目的地，Producer 不建议绑定目的地。也就是创建 Producer 的时候，不绑定目的地。`session.createProducer(null)`。

`send(Message message, int deliveryMode, int priority, long timeToLive)`; 发送消息到默认目的地，且设置相关参数。`deliveryMode`-持久化方式（`DeliveryMode.PERSISTENT` | `DeliveryMode.NON_PERSISTENT`）。`priority`-优先级。`timeToLive`-消息有效期（单位毫秒）。

`send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive)`; 发送消息到指定目的地，且设置相关参数。

#### 1.2 消息有效期

消息过期后，默认会将失效消息保存到“死信队列（ActiveMQ.DLQ）”。

不持久化的消息，在超时后直接丢弃，不会保存到死信队列中。

死信队列名称可配置，死信队列中的消息不能恢复。

死信队列是在 `activemq.xml` 中配置的。

#### 1.3 消息优先级

**不需特殊关注。**

我们可以在发送消息时，指定消息的权重，broker 可以建议权重较高的消息将会优先发送给 Consumer。在某些场景下，我们通常希望权重较高的消息优先传送；不过因为各种原因，**priority 并不能决定消息传送的严格顺序(order)**。

JMS 标准中约定 `priority` 可以为 0~9 的整数数值，值越大表示权重越高，默认值为 4。不过 activeMQ 中各个存储器对 `priority` 的支持并非完全一样。比如 JDBC 存储器可以支持 0~9，因为 JDBC 存储器可以基于 `priority` 对消息进行排序和索引化；但是对于 kahadb/levelDB 等这种基于日志文件的存储器而言，`priority` 支持相对较弱，只能识别三种优先级(LOW: < 4, NORMAL: =4, HIGH: > 4)。

### 1.3.1开启

在 broker 端，默认是不存储 priority 信息的，我们需要手动开启，修改 activemq.xml 配置文件，在 broker 标签的子标签 policyEntries 中增加下述配置：

```
<policyEntry queue="" prioritizedMessages="true"/>
```

不过对于“非持久化”类型的消息(如果没有被 swap 到临时文件)，它们被保存在内存中，它们不存在从文件 Paged in 到内存的过程，因为可以保证优先级较高的消息，总是在 prefetch 的时候被优先获取，这也是“非持久化”消息可以担保消息发送顺序的优点。

Broker 在收到 Producer 的消息之后，将会把消息 cache 到内存，如果消息需要持久化，那么同时也会把消息写入文件；如果通道中 Consumer 的消费速度足够快(即积压的消息很少，尚未超过内存限制，我们通过上文能够知道，每个通道都可以有一定的内存用来 cache 消息)，那么消息几乎不需要从存储文件中 Paged In，直接就能从内存的 cache 中获取即可，这种情况下，priority 可以担保“全局顺序”；不过，如果消费者滞后太多，cache 已满，就会触发新接收的消息直接保存在磁盘中，那么此时，priority 就没有那么有效了。

在 Queue 中，prefetch 的消息列表默认将会采用“轮询”的方式(roundRobin，注意并不是 roundRobinDispatch)[备注：因为 Queue 不支持任何 DispatchPolicy]，依次添加到每个 consumer 的 pending buffer 中，比如有 m1-m2-m3-m4 四条消息，有 C1-C2 两个消费者，那么：m1->C1,m2->C2,m3->C1,m4->C2。这种轮序方式，会对基于权重的消息发送有些额外的影响，假如四条消息的权重都不同，但是(m1,m3)->C1，事实上 m2 的权重>m3,对于 C1 而言，它似乎丢失了“顺序性”。

### 1.3.2强顺序

```
<policyEntry queue="" strictOrderDispatch="true"/>
```

strictOrderDispatch“严格顺序转发”，这是区别于“轮询”的一种消息转发手段；不过不要误解它为“全局严格顺序”，它只不过是 prefetch 的消息依次填满每个 consumer 的 pending buffer。比如上述例子中，如果 C1-C2 两个消费者的 buffer 尺寸为 3，那么 (m1,m2,m3)->C1,(m4)->C2;当 C1 填充完毕之后，才会填充 C2。由此这种策略可以保证 buffer 中所有的消息都是“**权重临近的**”、有序的。(需要注意：strictOrderDispatch 并非是解决 priority 消息顺序的问题而生，只是在使用 priority 时需要关注它)。

### 1.3.3严格顺序

```
<policyEntry queue="" prioritizedMessages="true" useCache="false"
expireMessagesPeriod="0" queuePrefetch="1"/>
```

useCache=false 来关闭内存，强制将所有的消息都立即写入文件(索引化，但是会降低消息的转发效率)；queuePrefetch=1 来约束每个 consumer 任何时刻只有一个消息正在处理，那些消息消费之后，将会从文件中重新获取，这大大增加了消息文件操作的次数，不过每次读取肯定都是 priority 最高的消息。



---

## 2 Consumer API 简介

### 2.1 消息的确认

Consumer 拉取消息后，如果没有做确认 `acknowledge`，此消息不会从 MQ 中删除。

消息的如果拉去到 consumer 后，未确认，那么消息被锁定。如果 consumer 关闭的时候仍旧没有确认消息，则释放消息锁定信息。消息将发送给其他的 consumer 处理。

消息一旦处理，应该必须确认。类似数据库中的事务管理机制。

### 2.2 消息的过滤

对消息消费者处理的消息数据进行过滤。这种处理可以明确消费者的角色，细分消费者的功能。

设置过滤：

```
Session.createConsumer(Destination destination, String messageSelector);
```

过滤信息为字符串，语法类似 SQL92 中的 `where` 子句条件信息。可以使用诸如 `AND`、`OR`、`IN`、`NOT IN` 等关键字。详细内容可以查看 `javax.jms.Message` 的帮助文档。

注意：消息的生产者在发送消息的时候，必须设置可过滤的属性信息，所有的属性信息设置方法格式为：`setXxxxProperty(String name, T value)`。其中方法名中的 `Xxxx` 是类型，如 `setObjectProperty/setStringProperty` 等。

## 八、 Spring&ActiveMQ

详见代码

ActiveMQ 的开发，和 Spring 的整合是非常方便的。且 Spring 有对 JMS 提供的 Template 机制。所以 Spring 管理 ActiveMQ 访问操作是非常方便的。

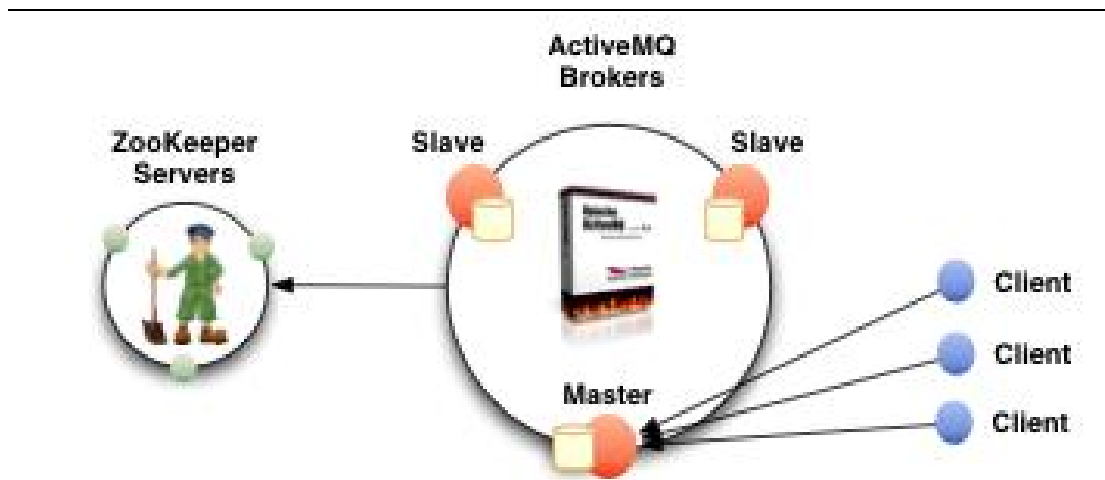
## 九、 ActiveMQ 集群

使用 ZooKeeper+ActiveMQ 实现主从和集群。

### 1 Master-Slave

主从模式是一种高可用解决方案。在 ZooKeeper 中注册若干 ActiveMQ Broker，其中只有一个 Broker 提供对外服务（Master），其他 Broker 处于待机状态（Slave）。当 Master 出现故障导致宕机时，通过 ZooKeeper 内的选举机制，选举出一台 Slave 替代 Master 继续对外提供服务。





官方文档: <http://activemq.apache.org/replicated-leveldb-store.html>

## 1.1 安装 ZooKeeper

搭建伪集群, 在同一个 Linux 中安装三个 ZooKeeper 实例。使用不同的端口实现同时启动。端口分配如下:

主机	服务端口	投票端口	选举端口
192.168.159.130	2181	2881	3881
192.168.159.130	2182	2882	3882
192.168.159.130	2183	2883	3883

### 1.1.1 解压缩

```
tar -zxf zookeeper
```

### 1.1.2 复制

```
cp -r zookeeper /usr/local/zookeeper1
```

### 1.1.3 创建 data 数据目录

在 zookeeper1 目录中创建子目录 data 目录

```
mkdir data
```

### 1.1.4 编写 Zookeeper 配置文件

```
vi /usr/local/solrcloude/zookeeper1/conf/zoo.cfg
```

修改数据目录

---

### 1.1.5 复制两份同样的 Zookeeper

```
cp zookeeper1 zookeeper2 -r
cp zookeeper1 zookeeper3 -r
```

### 1.1.6 为 Zookeeper 服务增加服务命名

在每个 Zookeeper 应用内的 data 目录中增加文件 myid  
内定义每个服务的编号. 编号要求为数字,是正整数可  
以使用回声命名快速定义 myid 文件  
echo 1 >> myid

### 1.1.7 修改 Zookeeper 配置文件 zoo.cfg

修改端口号.  
提供多节点服务命名

port=2181 客户端访问端口. 三个 Zookeeper 实例不能端口相同.  
server.编号=IP:投票端口:选举端口  
投票端口: 用于决定正在运行的主机是否宕机.  
选举端口: 用于决定哪一个 Zookeeper 服务作为主机.  
三个 Zookeeper 应用配置一致.  
server.1=192.168.120.132:2881:3881  
server.2=192.168.120.132:2882:3882  
server.3=192.168.120.132:2883:3883

### 1.1.8 启动 Zookeeper 测试

要至少启动两个 Zookeeper 启动. 启动单一 Zookeeper,无法正常提供服务.

## 1.2 安装 ActiveMQ

在同一个 Linux 中安装三个 ActiveMQ 实例, 使用不同端口实现同时启动。端口分配如下:

主机	M-S 通讯端口	服务端口	jetty 端口
192.168.159.130	62626	61616	8161
192.168.159.130	62627	61617	8162
192.168.159.130	62628	61618	8163

### 1.2.1 安装 ActiveMQ 实例

---

## 1.2.2 修改配置信息

### 1.2.2.1 修改 jetty 端口

修改 conf/jetty.xml 中的端口配置。分别是 8161、8162、8163

```
<bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort"
init-method="start">
    <!-- the default port number for the web console -->
    <property name="port" value="8161"/>
</bean>
```

### 1.2.2.2 统一所有主从节点 Broker 命名

修改 conf/activemq.xml 文件。修改 broker 标签属性信息，统一所有节点的 broker 命名。

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="mq-cluster"
dataDirectory="${activemq.data}">
```

### 1.2.2.3 修改持久化配置

修改 conf/activemq.xml 文件。修改 broker 标签中子标签 persistenceAdapter 相关内容。

replicas 属性代表当前主从模型中的节点数量。按需配置。

bind 属性中的端口为主从实例之间的通讯端口。代表当前实例对外开放端口是什么，三个实例分别使用 62626、62627、62628 端口。

zkAddress 属性代表 ZooKeeper 安装位置，安装具体情况设置。

zkPath 是 ActiveMQ 主从信息保存到 ZooKeeper 中的什么目录内。

hostname 为 ActiveMQ 实例安装 Linux 的主机名，可以在/etc/hosts 配置文件中设置。设置格式为：IP 主机名。如： 127.0.0.1 mq-server

```
<persistenceAdapter>
    <!-- <kahaDB directory="${activemq.data}/kahadb"/> -->
    <replicatedLevelDB
        directory="${activemq.data}/levelDB"
        replicas="3"
        bind="tcp://0.0.0.0:62626"
        zkAddress="192.168.159.130:2181,192.168.159.130:2182,192.168.159.130:2183"
        zkPath="/activemq/leveldb-stores"
        hostname="mq-server"
    />
</persistenceAdapter>
```

### 1.2.2.4 修改服务端口

修改 ActiveMQ 对外提供的服务端口。原默认端口为 61616。当前环境使用的端口为：

---

61616、61617、61618。

修改 conf/activemq.xml 配置文件。修改 broker 标签中子标签 transportConnectors 的相关配置。**只修改强调内容。**

```
<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&wireFormat.maxFrameSize=1048576
00"/>
  <transportConnector                                name="amqp"
uri="amqp://0.0.0.0:5672?maximumConnections=1000&wireFormat.maxFrameSize=104857
600"/>
  <transportConnector                                name="stomp"
uri="stomp://0.0.0.0:61613?maximumConnections=1000&wireFormat.maxFrameSize=1048
57600"/>
  <transportConnector                                name="mqtt"
uri="mqtt://0.0.0.0:1883?maximumConnections=1000&wireFormat.maxFrameSize=104857
600"/>
  <transportConnector                                name="ws"
uri="ws://0.0.0.0:61614?maximumConnections=1000&wireFormat.maxFrameSize=1048576
00"/>
</transportConnectors>
```

## 1.3 启动主从

将三个 ActiveMQ 实例分别启动。\${activemq-home}/bin/active start。启动后，可以查看日志文件，检查启动状态，日志文件为\${activemq-home}/data/activemq.log。

## 1.4 查看主从状态

### 1.4.1 使用客户端连接 ZooKeeper

\${zkHome}/bin/zkCli.sh

### 1.4.2 查看状态信息

连接成功后，可以使用命令 ‘ls’ 查看 ZooKeeper 中的目录结构如：

```
ls /
```

```
ls /activemq/leveldb-stores
```

找到对应的内容后，可以使用命令 ‘get’ 查看 ZooKeeper 中的数据内容  
get /activemq/leveldb-stores/00000000005

```
[zk: localhost:2181(CONNECTED) 12] get /activemq/leveldb-stores/00000000005
{"id":"mq-cluster","container":null,"address":"tcp://mq-server:62627","position":-1,"weight":1,"elected":"00000000005"}
cZxid = 0x10000002b
ctime = Wed May 16 01:23:21 PDT 2018
mZxid = 0x10000003a
mtime = Wed May 16 01:41:21 PDT 2018
pZxid = 0x10000002b
cversion = 0
dataVersion = 6
aclVersion = 0
ephemeralOwner = 0x16367aacc680007
dataLength = 118
numChildren = 0
```

其中主节点的 elected 及 address 属性一定有数据。从节点则数据为 'null'。

## 2 集群

准备多份主从模型。在所有的 ActiveMQ 节点中的 conf/activemq.xml 中增加下述配置：  
(每个主从模型中的 networkConnector 都指向另外一个主从模型)

```
<networkConnectors>
  <networkConnector uri="static://(tcp://ip:port,tcp://ip:port)" duplex="false">
  </networkConnector>
</networkConnectors>
```

注意配置顺序，Networks 相关配置必须在持久化相关配置之前。如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://activemq.apache.org/schema/core">
  <broker xmlns="http://activemq.apache.org/schema/core" brokerName="mq-cluster"
dataDirectory="${activemq.data}" >
    <networkConnectors>
      <networkConnector uri=" static://(tcp://ip:port,tcp://ip:port)"/>
    </networkConnectors>
    <persistenceAdapter>
      < replicatedLevelDB directory = "xxx"/>
    </persistenceAdapter>
  </broker>
</beans>
```

如： 主从模型 1 - 192.168.159.129 主从模型 2 - 192.168.159.130

在主从模型 1 的所有节点 activemq.xml 配置文件中增加标签：

```
<networkConnectors>
<networkConnector
uri="static://(tcp://192.168.159.130:61616,tcp://192.168.159.130:61617)"/>
</networkConnectors>
```

在模型 2 中所有节点增加配置：

```
<networkConnectors>
<networkConnector
uri="static://(tcp://192.168.159.129:61616,tcp://192.168.159.129:61617)"/>
</networkConnectors>
```