# ActiveMQ集群的由来

单点的ActiveMQ作为企业应用无法满足业务的需求,因为单点的ActiveMQ存在单点故障问题,当该节点宕机以后,就会直接影响我们业务的正常运转,所以我们需要搭建高可用的ActiveMQ集群来支撑我们的业务系统

# ActiveMQ集群的主要部署方式

## 默认的单机部署（kahadb）

activeMQ的默认存储的单机方式，以本地kahadb文件的方式存储，所以性能指标完全依赖本地磁盘IO，不能提供高可用。

<persistenceAdapter>  
<kahaDB directory="${activemq.data}/kahadb"/>

</persistenceAdapter>

## 基于共享数据库的主从（Shared JDBC Master/Slave）

可以基于postgres、mysql、oracle等常用数据库。每个节点启动都会争抢数据库锁，从而保证master的唯一性，其他节点作为备份，一直等待数据库锁的释放。因为所有消息读写，其实都是数据库操作，activeMQ节点本身压力很小，性能完全取决于数据库性能。优点：实现高可用和数据安全,简单灵活，2台节点就可以实现高可用. 缺点：稳定性依赖数据库,性能依赖数据库.

<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:3306/amq?relaxAutoCommit=true"/>  
<property name="username" value="root"/>  
<property name="password" value="root"/>  
<property name="maxActive" value="20"/>  
<property name="poolPreparedStatements" value="true"/>

</bean>

<persistenceAdapter>  
<jdbcPersistenceAdapter dataDirectory="${activemq.data}" dataSource="#mysql-ds"  
createTablesOnStartup="false"/>

</persistenceAdapter>

## 基于zookeeper以及可复制的 LevelDB实现

5.9.0新推出的主从实现，基于zookeeper来选举出一个master，其他节点自动作为slave实时同步消息。因为有实时同步数据的slave的存在，master不用担心数据丢失，所以leveldb会优先采用内存存储消息，异步同步到磁盘。所以该方式的activeMQ**读写性能都最好**，特别是写性能能够媲美非持久化消息。优点：实现高可用和数据安全性能较好.缺点：因为选举机制要超过半数，所以**最少需要3台节点**，才能实现高可用。  
LevelDB 是 Google 开发的一套用于持久化数据的高性能类库。 LevelDB 并不是一种服务,用户需要自行实现 Server。 是单进程的服务，能够处理十亿级别规模 Key-Value 型数据，占用内存小。

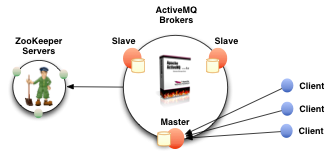
<persistenceAdapter>  
<replicatedLevelDB  
 directory="${activemq.data}/leveldb"  
 replicas="3"  
 bind="tcp://0.0.0.0:62621"  
 zkAddress="localhost:2181,localhost:2182,localhost:2183"  
 hostname="localhost"  
 zkPath="/activemq/leveldb-stores"  
/>

</persistenceAdapter>

了解完毕ActiveMQ集群的三种部署方式以后,我们本教程将采用第三种方式来建ActiveMQ集群,因为第三种的读写性能都最好的.

# ActiveMQ高可用原理

使用ZooKeeper（集群）注册所有的ActiveMQ Broker。只有其中的一个Broker可以提供服务，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。所有的存储操作都将被复制到 连接至 Master的Slaves。如果Master宕了，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Slave模式。



# 基于zookeeper以及leveldb实现高可用ActiveMQ集群

了解完毕ActiveMQ集群的三种部署方式以后,我们本教程将采用第三种方式来建ActiveMQ集群,因为第三种的读写性能都最好的.

## ActiveMQ集群部署规划

环境： CentOS 6.6 x64 、 JDK7  
版本： ActiveMQ 5.11.1

zookeeper集群说明: 192.168.221.141:2181 , 192.168.221.141:2182 , 192.168.221.141:2183

集群节点规划说明:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **主机** | **集群间通信端口** | **消息端口** | **管控台端口** | **其他端口** | **安装目录** |
| 192.168.221.136 | 62621 | 11616 | 8161 | 以1开始即可 | /usr/local/src/activemq-cluster |
| 192.168.221.136 | 62622 | 21616 | 8162 | 以2开始即可 |
| 192.168.221.136 | 62623 | 31616 | 8163 | 以3开始即可 |

## 搭建ActiveMQ步骤

### 准备环境

* 在/usr/local/src/在创建activemq-cluster目录: mkdir –p /usr/local/src/activemq-cluster
* 上传activemq安装包到linux服务器下
* 解压activemq的安装包: tar –zxvf –C /usr/local/src/activemq-cluster
* 重命名: mv apache-activemq-5.12.0/ activemq-cluster-node01
* 以1节点为基础复制两个节点出来
* cp –r activemq-cluster-node01 activemq-cluster-node02
* cp –r activemq-cluster-node01 activemq-cluster-node03

### 修改管理控制台端口可在 conf/jetty.xml 中修改

node01管控台端口

|  |
| --- |
| <bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">  <property name="host" value="0.0.0.0"/>  <property name="port" value="**8161**"/>  </bean> |

node02管控台端口

|  |
| --- |
| <bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">  <property name="host" value="0.0.0.0"/>  <property name="port" value="**8162**"/>  </bean> |

node03管控台端口

|  |
| --- |
| <bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">  <property name="host" value="0.0.0.0"/>  <property name="port" value="**8163**"/>  </bean> |

### 配置持久化适配器

在 3 个 ActiveMQ 节点中配置 **conf/activemq.xml** 中的持久化适配器。修改其中 bind、zkAddress、hostname 和 zkPath.**注意:每个 ActiveMQ 的 BrokerName 必须相同，否则不能加入集群。**

node01-持久化适配器:

|  |
| --- |
| <broker xmlns="http://activemq.apache.org/schema/core" brokerName=**"activemq-cluster"** dataDirectory="${activemq.data}">  <persistenceAdapter>  <replicatedLevelDB  directory="${activemq.data}/leveldb"  replicas="3"  bind="tcp://192.168.221.136:**62621**"  zkAddress="**192.168.80.129:2182,192.168.80.129:2183,192.168.80.129:2184**"  hostname="**192.168.80.129**"  zkPath="/activemq/leveldb-stores"  />  </persistenceAdapter>  </broker> |

node02-持久化适配器:

|  |
| --- |
| <broker xmlns="http://activemq.apache.org/schema/core" brokerName=**"activemq-cluster"** dataDirectory="${activemq.data}">  <persistenceAdapter>  <replicatedLevelDB  directory="${activemq.data}/leveldb"  replicas="3"  bind="tcp://192.168.221.136:**62622**"  zkAddress="**192.168.221.145:2181,192.168.221.145:2182,192.168.221.145:2183**"  hostname="192.168.221.136"  zkPath="/activemq/leveldb-stores"  />  </persistenceAdapter>  </broker> |

node03-持久化适配器:

|  |
| --- |
| <broker xmlns="http://activemq.apache.org/schema/core" brokerName=**"activemq-cluster"** dataDirectory="${activemq.data}">  <persistenceAdapter>  <replicatedLevelDB  directory="${activemq.data}/leveldb"  replicas="3"  bind="tcp://192.168.221.136:**62623**"  zkAddress="**192.168.221.145:2181,192.168.221.145:2182,192.168.221.145:2183**"  hostname="192.168.221.136"  zkPath="/activemq/leveldb-stores"  />  </persistenceAdapter>  </broker> |

### 修改消息服务端口

node-01消息服务端口:

|  |
| --- |
| <transportConnectors>  <transportConnector name="openwire" uri="tcp://0.0.0.0:**11616**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="amqp" uri="amqp://0.0.0.0:**1672**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="stomp" uri="stomp://0.0.0.0:**11613**?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:**11614**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  </transportConnectors> |

node-02消息服务端口:

|  |
| --- |
| <transportConnectors>  <transportConnector name="openwire" uri="tcp://0.0.0.0:**21616**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="amqp" uri="amqp://0.0.0.0:**2672**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="stomp" uri="stomp://0.0.0.0:**21613**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="mqtt" uri="mqtt://0.0.0.0:**2883**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="ws" uri="ws://0.0.0.0:**21614**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  </transportConnectors> |

node-03消息服务端口:

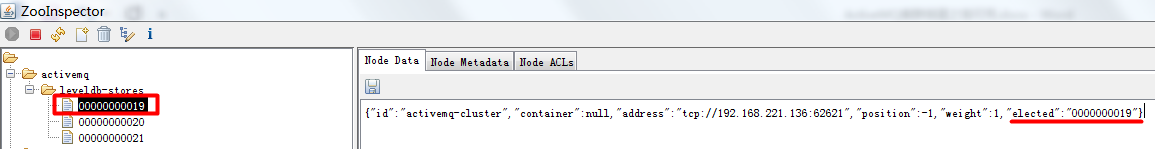
|  |
| --- |
| <transportConnectors>  <transportConnector name="openwire" uri="tcp://0.0.0.0:**31616**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="amqp" uri="amqp://0.0.0.0:**3672**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="stomp" uri="stomp://0.0.0.0:**31613**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="mqtt" uri="mqtt://0.0.0.0:**3883**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  <transportConnector name="ws" uri="ws://0.0.0.0:**31614**?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>  </transportConnectors> |

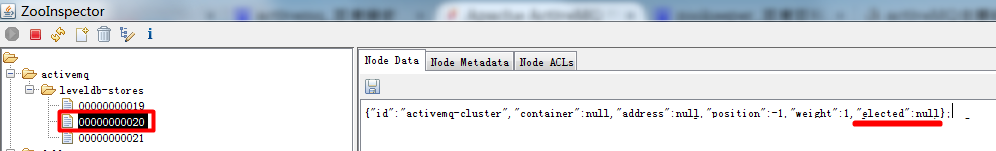
### 启动3个ActiveMQ集群节点

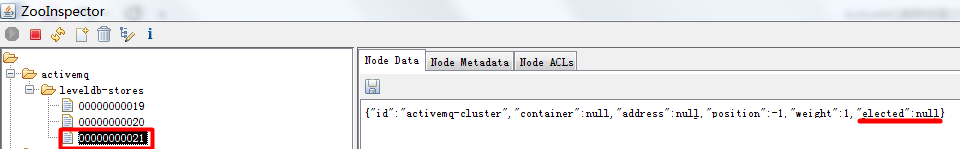
|  |
| --- |
| /usr/local/src/activemq-cluster/apache-activemq-01/bin/activemq start  /usr/local/src/activemq-cluster/apache-activemq-02/bin/activemq start  /usr/local/src/activemq-cluster/apache-activemq-03/bin/activemq start  监听日志  tail -f /usr/local/src/activemq-cluster/apache-activemq-01/data/activemq.log  tail -f /usr/local/src/activemq-cluster/apache-activemq-02/data/activemq.log  tail -f /usr/local/src/activemq-cluster/apache-activemq-03/data/activemq.log |

### 集群的节点状态分析

使用集群启动后对 ZooKeeper 数据的抓图，可以看到 ActiveMQ 的有 3 个节点，分别是 00000000019，000000000020， 00000000021。以下第一张图展现了 00000000019 的值，可以看到 elected 的值是不为空，说明这个节点是 Master，其他两个节点是 Slave。







### 集群测试

ActiveMQ的客户端只能访问MasterBroker,其他处于Slave的Broker不能访问.所以客户端连接Broker应该使用failover(故障转移)协议

**failover:(tcp://192.168.221.136:11616,tcp:// 192.168.221.136:21616,tcp:// 192.168.221.136:31616)?randomize=false**

### 客户端连接url配置优化

updateURIsURL，通过 URL（或者本地路径）获取重连的 url，这样做具有良好的扩展性，因为客户端每次连接都是从 URL（或文件）中加载一次，所以可以随时从文件中更新 url 列表，做到动态添加 MQ 的备点。failover:()?randomize=false&updateURIsURL=file:/home/wusc/activemq/urllist.txt, urllist.txt 中的地址通过英文逗号分隔，示例：

**tcp://192.168.221.136:11616,tcp:// 192.168.221.136:21616,tcp:// 192.168.221.136:31616**

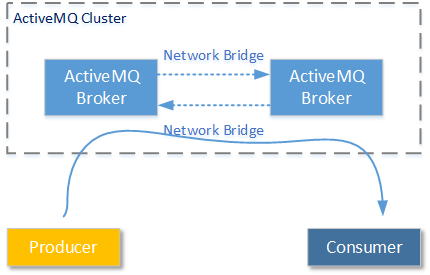
### 集群测试结果说明

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,因为不能选举Master,ActiveMQ不能正常运转;同样的,如果ZooKeeper仅剩一个节点活动.不管ActiveMQ 各节点是否存活,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）。

# 具有负载均衡能力的高可用ActiveMQ集群的搭建

## 具有负载均衡能力的高可用ActiveMQ集群原理

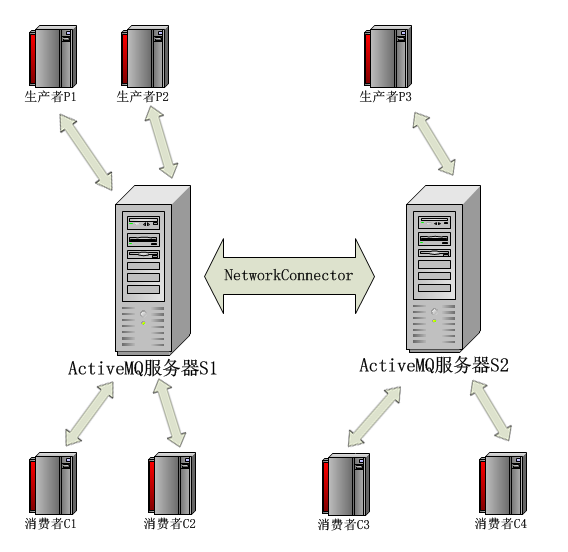
上面我们搭建了一个ActiveMQ集群是master-slave模式的.可以实现高可用的效果.但是模式的集群不具有负载均衡的能力(只有一个master对外提供服务).当我们的消息数量很大的时候一个master提供服务,效率势必不会很高.那么为了提供高效率的消息发送和接收我们需要实现负载均衡.我们需要使用两套以上的master-slave集群来实现负载均衡.原理图如下:



通过这张图我们可以清楚的看到存在两套master-slave集群,两套集群之间使用网桥进行连接(互相进行数据共享), 而每一套master-slave集群都存在一个master,就说明是两个master同时在进行消息的发送以及接收.

## 网络连接器

我们知道，ActiveMQ中的TransportConnector（传输连接器）主要用于配置ActiveMQ服务端和客户端之间的通信方式，NetworkConnector（网络连接器）则主要用来配置ActiveMQ服务端与服务端之间的通信。在某些场景，网络拓扑中我们可能会需要大量的生产者和消费者，也就是说我们会有大量的ActiveMQ客户端，这样，我们就需要在网络环境中有多个MQ服务器，服务器之间是可以透传消息的，这是，我们就需要用到NetworkConnector，我们先从简单的分析起，假设我们需要部署两台MQ服务器，如下图：



如上图所示，服务器S1和S2通过NewworkConnector相连，则生产者P1发送消息，消费者C3和C4都可以接收到，而生产者P3发送的消息，消费者C1和C2同样也可以接收到.要想完成这样的数据透传,我们就需要使用网络连接器

### 静态网络连接器

网络连接器大致可以分为两种一种是静态网络连接器一种是动态网络连接器.静态网络连接器的配置.如果我们想让生产者P1发送的消息可以被消费者C3和C4接收到.我们就需要在S1服务器上的每一个节点的activemq.xml文件中配置静态网络连接器,配置如下:

|  |
| --- |
| <networkConnectors>  <networkConnector uri="static:(tcp://192.168.221.141:11616, tcp://192.168.221.141:21616, tcp://192.168.221.141:31616)" duplex="false"/>  </networkConnectors> |

同理如果P3生产者产生的消息需要可以C1和C2消费者接收到,我们就需要在S2服务器的每一个节点的activemq.xml配置静态网络连接器,配置如下:

|  |
| --- |
| <networkConnectors>  <networkConnector uri="static:(tcp://192.168.221.136:11616, tcp:// 192.168.221.136:21616, tcp:// 192.168.221.136:31616)" duplex="false"/>  </networkConnectors> |

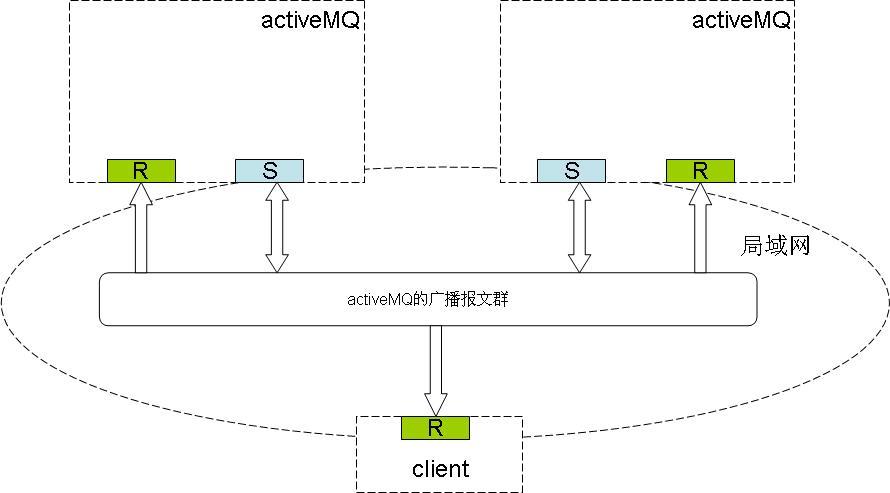
networkConnector中使用uri属性来指定远端的Broker地址（下游网络节点，可以有多个），“static”是一个逻辑协议，和Client端使用的“failover”协议一样，可以实现链接重连和列表遍历重试特性。通常“static”协议用在networkConnector中，“failover”协议用在Client端。broker将会依次尝试与uri中每个broker建立链接，直到与其一建立成功为止。通常uri中broker列表为master-slave模式，这也是broker高可用设计要求.

### 动态网络连接器

有人会想，如果只有两三台MQ服务器，我们可以像上面那样简单配置，如果MQ服务器非常多，我们就不太可能在每个MQ服务器的xml文件中去一个个配置其它服务器地址了，所以，我们希望MQ服务器能在自己的网络中发现其他的MQ服务器并自动建立“桥接”，幸运的是，ActiveMQ提供了多播发现（Multicast Discovery）的功能，我们可以在每台MQ服务器的activemq.xml中的broker节点下添加如下配置：

* 动态网络连接器原理

下图为搭建展示了动态网络连接器的原理:



图中有两个activeMQ、一个客户端(消息发送者/消费者)。我们使用activeMQ的自动发现功能让它们来发现彼此的存在。自动发现功能的好处在activeMQ里面，我想主要能使用在当目前的activeMQ集群处理能力不行的时候，可以动态加入新的activeMQ来分担负载，这就可以很好的水平扩展。首先，我们对图一中的带颜色部分做一个说明，带颜色的小矩形都代表着能发送和接收广播数据包的通讯端口，有‘S’标志的表示该端口主要向外发送广播数据包，有‘R’表示该端口只接收广播数据包。具体自动发现过程描述如下：activeMQ通过端口向外广播它们自己，当某activeMQ收到一个广播包后，解析该广播包，并从中建立activeMQ之间的通讯桥梁。而当某客户端收到包含activeMQ的基本信息的广播包后，它也会把这个信息保存住，在需要进行重连接的时候进行重连接。

* 动态网络连接器的配置

|  |
| --- |
| 动态网络连接器的配置:  <networkConnectors>  <networkConnector **uri="multicast://default"**/>  </networkConnectors>  传输连接器配置:  <transportConnectors>  <transportConnectorname="openwire"uri="tcp://0.0.0.0:11616? " **discoveryUri="multicast://default"**/>  </transportConnectors> |