[Apache ActiveMQ](http://activemq.apache.org/) ™ is the most popular and powerful open source messaging and [Integration Patterns](http://activemq.apache.org/enterprise-integration-patterns.html) server.

Apache ActiveMQ is fast, supports many [Cross Language Clients and Protocols](http://activemq.apache.org/cross-language-clients.html), comes with easy to use [Enterprise Integration Patterns](http://activemq.apache.org/enterprise-integration-patterns.html) 企业集成模式and many [advanced features](http://activemq.apache.org/features.html) while fully supporting [JMS 1.1](http://java.sun.com/products/jms/) and J2EE 1.4. Apache ActiveMQ is released under the [Apache](http://www.apache.org/) [2.0 License](http://www.apache.org/licenses/LICENSE-2.0.html)

# ****RabbitMQ、ActiveMQ和ZeroMQ****

作者：chszs，转载需注明。博客主页：http://blog.csdn.net/chszs  
  
RabbitMQ、ActiveMQ和ZeroMQ都是极好的消息中间件，但是我们在项目中该选择哪个更适合呢？很多开发者面临这个烦恼。下面我会对这三个消息中间件做一个比较，看了后你们就心中有数了。  
  
RabbitMQ是AMQP协议领先的一个实现，它实现了代理(Broker)架构，意味着消息在发送到客户端之前可以在中央节点上排队。此特性使得RabbitMQ易于使用和部署，适宜于很多场景如路由、负载均衡或消息持久化等，用消息队列只需几行代码即可搞定。但是，这使得它的可扩展性差，速度较慢，因为中央节点增加了延迟，消息封装后也比较大。  
  
  
ZeroMQ是一个非常轻量级的消息系统，专门为高吞吐量/低延迟的场景开发，在金融界的应用中经常可以发现它。与RabbitMQ相比，ZeroMQ支持许多高级消息场景，但是你必须实现ZeroMQ框架中的各个块（比如Socket或Device等）。ZeroMQ非常灵活，但是你必须学习它的80页的手册（如果你要写一个分布式系统，一定要阅读它）。  
  
  
ActiveMQ居于两者之间，类似于ZemoMQ，它可以部署于代理模式和P2P模式。类似于RabbitMQ，它易于实现高级场景，而且只需付出低消耗。它被誉为消息中间件的“瑞士军刀”。  
  
要注意一点，ActiveMQ的下一代产品为Apollo。  
  
  
最终，这三个产品：  
1. 都有客户端API且支持多种编程语言；  
2. 都有大量的文档；  
3. 都提供了积极的支持。

// 获取本机IP  
InetAddress addr = InetAddress.getLocalHost();  
ip = addr.getHostAddress().toString();  
catch (UnknownHostException ex) {  
ex.printStackTrace();  
ip = "";  
}  
if(!"".equals(ip)) {  
System.out.println("CLIENT: " + ip);  
// 设置订阅客户端ID  
connection.setClientID(ip);  
connection.start();  
Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);  
// 创建主题  
Topic topic = session.createTopic("myTopic.messages");  
// 创建订阅  
MessageConsumer consumer = session.createDurableSubscriber(topic, "test");  
consumer.setMessageListener(new MessageListener() {  
// 订阅接收方法  
public void onMessage(Message message) {  
TextMessage tm = (TextMessage) message;  
try {  
System.out.println("Received message: " + tm.getText());  
} catch (JMSException e) {  
e.printStackTrace();  
}  
}  
});  
}  
3.4 注意事项  
 普通订阅的情况下，客户端只有在连接到服务器的情况下，才能接收服务器上的主题消息。  
 持久订阅后，当客户端在线时，服务器端会把客户端在上次下线之后到本次上线之间的所有消息一并推送给客户端；这样就保证了客户端不会有丢失的消息。  
 持久订阅会引发另一个问题：当新增一个订阅客户端时，这个客户端会收到服务器上该主题下的所有未过期消息。

### **Features**

* Supports a variety of [Cross Language Clients and Protocols](http://activemq.apache.org/cross-language-clients.html) from Java, C, C++, C#, Ruby, Perl, Python, PHP
  + [OpenWire](http://activemq.apache.org/openwire.html) for high performance clients in Java, C, C++, C#
  + [Stomp](http://activemq.apache.org/stomp.html) support so that clients can be written easily in C, Ruby, Perl, Python, PHP, ActionScript/Flash, Smalltalk to talk to ActiveMQ as well as any other popular Message Broker
  + [AMQP](http://activemq.apache.org/amqp.html) v1.0 support
  + [MQTT](http://activemq.apache.org/mqtt.html) v3.1 support allowing for connections in an IoT environment.
* full support for the [Enterprise Integration Patterns](http://activemq.apache.org/enterprise-integration-patterns.html) both in the JMS client and the Message Broker
* Supports many [advanced features](http://activemq.apache.org/features.html) such as [Message Groups](http://activemq.apache.org/message-groups.html), [Virtual Destinations](http://activemq.apache.org/virtual-destinations.html), [Wildcards](http://activemq.apache.org/wildcards.html) and [Composite Destinations](http://activemq.apache.org/composite-destinations.html)
* Fully supports JMS 1.1 and J2EE 1.4 with support for transient, persistent, transactional and XA messaging
* [Spring Support](http://activemq.apache.org/spring-support.html) so that ActiveMQ can be easily embedded into Spring applications and configured using Spring's XML configuration mechanism
* Tested inside popular J2EE servers such as [TomEE](http://tomee.apache.org/), [Geronimo](http://geronimo.apache.org/), JBoss, GlassFish and WebLogic
  + Includes [JCA 1.5 resource adaptors](http://activemq.apache.org/resource-adapter.html) for inbound & outbound messaging so that ActiveMQ should auto-deploy in any J2EE 1.4 compliant server
* Supports pluggable [transport protocols](http://activemq.apache.org/uri-protocols.html) such as [in-VM](http://activemq.apache.org/how-do-i-use-activemq-using-in-jvm-messaging.html), TCP, SSL, NIO, UDP, multicast, JGroups and JXTA transports
* Supports very fast [persistence](http://activemq.apache.org/persistence.html) using JDBC along with a high performance journal
* Designed for high performance clustering, client-server, peer based communication
* [REST](http://activemq.apache.org/rest.html) API to provide technology agnostic and language neutral web based API to messaging
* [Ajax](http://activemq.apache.org/ajax.html) to support web streaming support to web browsers using pure DHTML, allowing web browsers to be part of the messaging fabric
* [CXF and Axis Support](http://activemq.apache.org/axis-and-cxf-support.html) so that ActiveMQ can be easily dropped into either of these web service stacks to provide reliable messaging
* Can be used as an in memory JMS provider, ideal for [unit testing JMS](http://activemq.apache.org/how-to-unit-test-jms-code.html)

1. 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

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

3. 对Spring的支持,ActiveMQ可以很容易内嵌到使用Spring的系统里面去,而且也支持Spring2.0的特性

4. 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 resourceadaptors的配置,可以让ActiveMQ可以自动的部署到任何兼容J2EE1.4商业服务器上

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

6. 支持通过JDBC和journal提供高速的消息持久化

7. 从设计上保证了高性能的集群,客户端-服务器,点对点

8. 支持Ajax

9. 支持与Axis的整合

10. 可以很容易得调用内嵌JMS provider,进行测试

ActiveMQ的使用分为两种，一种是嵌入式，即把ActiveMQ作为内嵌的JMS provider集成到Tomcat或其他Web服务器中；另一种是独立部署，即ActiveMQ单独部署，独立运行，通过ActiveMQ提供的API进行进程外访问。  
我们采用独立部署的方式使用ActiveMQ，这样的做法更有利于业务的解耦和工程部署结构的分离；也有利于对ActiveMQ进行配置定制和优化。  
我们知道JMS规范中约定了两种消息发送方式：P2P和Publish/Subscribe。根据业务的需求，我们采用P/S方式更为合理，也能更好的发挥ActiveMQ本身的优势。

 主要是通过共享存储目录来实现master和slave的热备，所有的ActiveMQ应用都在不断地获取共享目录的控制权，哪个应用抢到了控制权，它就成为master。

 多个共享存储目录的应用，谁先启动，谁就可以最早取得共享目录的控制权成为master，其他的应用就只能作为slave。

    与shared filesystem方式类似，只是共享的存储介质由文件系统改成了数据库而已。

3）Replicated LevelDB Store方式

         这种主备方式是ActiveMQ5.9以后才新增的特性

使用ZooKeeper协调选择一个node作为master。

被选择的master broker node开启并接受客户端连接。

其他node转入slave模式，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死了，得到了最新更新的slave被允许成为master。

fialed node能够重新加入到网络中并连接master进入slave mode。

所有需要同步的disk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，熟悉Zookeeper的应该知道，有一个node要作为观擦者存在。

单一个新的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将会成为新的master。因此，推荐运行至少3个replica nodes，以防止一个node失败了，服务中断。

2、Broker-Cluster部署方式

         前面的Master-Slave的方式虽然能解决多服务热备的高可用问题，但无法解决负载均衡和分布式的问题。Broker-Cluster的部署方式就可以解决负载均衡的问题。

         Broker-Cluster部署方式中，各个broker通过网络互相连接，并共享queue。当broker-A上面指定的queue-A中接收到一个message处于pending状态，而此时没有consumer连接broker-A时。如果cluster中的broker-B上面由一个consumer在消费queue-A的消息，那么broker-B会先通过内部网络获取到broker-A上面的message，并通知自己的consumer来消费。

1）static Broker-Cluster部署

         在activemq.xml文件中静态指定Broker需要建立桥连接的其他Broker：

1、  首先在Broker-A节点中添加networkConnector节点：

<networkConnectors>

                <networkConnector   uri="static:(tcp:// 0.0.0.0:61617)"duplex="false"/>

</networkConnectors>

2、  修改Broker-A节点中的服务提供端口为61616：

<transportConnectors>

         <transportConnectorname="openwire"uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

3、  在Broker-B节点中添加networkConnector节点：

<networkConnectors>

                <networkConnector   uri="static:(tcp:// 0.0.0.0:61616)"duplex="false"/>

</networkConnectors>

4、  修改Broker-A节点中的服务提供端口为61617：

<transportConnectors>

         <transportConnectorname="openwire"uri="tcp://0.0.0.0:61617?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

5、分别启动Broker-A和Broker-B。

2）Dynamic Broker-Cluster部署

         在activemq.xml文件中不直接指定Broker需要建立桥连接的其他Broker，由activemq在启动后动态查找：

1、  首先在Broker-A节点中添加networkConnector节点：

<networkConnectors>

                <networkConnectoruri="multicast://default"

           dynamicOnly="true"

           networkTTL="3"

           prefetchSize="1"

           decreaseNetworkConsumerPriority="true" />

</networkConnectors>

2、修改Broker-A节点中的服务提供端口为61616：

<transportConnectors>

         <transportConnectorname="openwire"uri="tcp://0.0.0.0:61616? " discoveryUri="multicast://default"/>

</transportConnectors>

3、在Broker-B节点中添加networkConnector节点：

<networkConnectors>

                <networkConnectoruri="multicast://default"

           dynamicOnly="true"

           networkTTL="3"

           prefetchSize="1"

           decreaseNetworkConsumerPriority="true" />

</networkConnectors>

4、修改Broker-B节点中的服务提供端口为61617：

<transportConnectors>

         <transportConnectorname="openwire"uri="tcp://0.0.0.0:61617" discoveryUri="multicast://default"/>

</transportConnectors>

1. 启动Broker-A和Broker-B

配置  
ActiveMQ的配置存放在安装目录的conf/activemq.xml文件中。  
因为ActiveMQ采用了Jetty作为容器，因此Jetty相关的配置在conf/jetty.xml文件中。

 集群  
ActiveMQ的集群由服务器端和客户端共同完成。服务器端通过部署Master/Slaver机制，通过进行分布式部署，以实现服务器集群的平行扩展。而客户端则采取静态地址发现，或者动态地址发现的方式，实现服务器的负载均衡选择。  
5.2.1 服务器端的部署  
ActiveMQ支持Master/Slaver机制，但简单Master/Slaver方式有一定的局限性，不适合服务器集群的平行扩展（当然，简单Master/Slaver已经足够支撑一般的商业应用）。因此，ActiveMQ提供了支持大并发请求的集群方式：共享文件系统的集群，以及基于JDBC的集群。  
 共享文件系统的集群  
实际上就是基于文件系统进行集群部署（前面提到过，ActiveMQ默认的消息存储就是基于文件系统的），可以通过分布式存储系统或共享数据目录来实现。这种方式只需要修改conf/activemq.xml：  
<persistenceAdapter>  
<journaledJDBC dataDirectory="/sharedFileSystem/broker"/>  
</persistenceAdapter>  
 基于JDBC的集群  
原理与共享文件系统一致，只不过把文件系统换成了[数据库](http://www.2cto.com/database/" \t "http://www.2cto.com/kf/201304/_blank)平台。即：多台ActiveMQ连接同一个数据库，从而实现ActiveMQ的服务器集群。配置同5.1。  
5.2.2 客户端的使用  
服务器端的集群对客户端而言是透明的，但如果客户端希望得到集群和负载均衡的功能支持，则必须在代码中有所体现。  
最常规的方法就是failover协议，fileover支持客户端在当前服务器断开的情况下，自动重新连接到新的服务器上，而新的服务器地址可以来源于静态地址列表，也可以来源于动态地址广播。  
 静态地址发现的常规用法  
ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("failover:(tcp://primary:61616,tcp://secondary:61616)?randomize=false");  
 动态地址发现的常规用法  
ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("failover:(multicast://host:6255)");  
当然，ActiveMQ还支持更多的协议，如：fanout、discovery等。  
5.3 管理与监控  
ActiveMQ提供了一个Web后台用于查看服务器运行状态，并提供了对消息队列、主题及订阅者等进行管理的功能。  
另外，ActiveMQ也可以通过配置支持Nagios的集成监控。

2、Master-Slave与Broker-Cluster相结合的部署方式

         可以看到Master-Slave的部署方式虽然解决了高可用的问题，但不支持负载均衡，Broker-Cluster解决了负载均衡，但当其中一个Broker突然宕掉的话，那么存在于该Broker上处于Pending状态的message将会丢失，无法达到高可用的目的。

         由于目前ActiveMQ官网上并没有一个明确的将两种部署方式相结合的部署方案，所以我尝试者把两者结合起来部署：

1、部署的配置修改

         这里以Broker-A + Broker-B建立cluster，Broker-C作为Broker-B的slave为例：

1）首先在Broker-A节点中添加networkConnector节点：

<networkConnectors>

                <networkConnector   uri="masterslave:(tcp://0.0.0.0:61617,tcp:// 0.0.0.0:61618)" duplex="false"/>

</networkConnectors>

2）修改Broker-A节点中的服务提供端口为61616：

<transportConnectors>

         <transportConnectorname="openwire"uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

3）在Broker-B节点中添加networkConnector节点：

<networkConnectors>

                <networkConnector   uri="static:(tcp:// 0.0.0.0:61616)"duplex="false"/>

</networkConnectors>

4）修改Broker-B节点中的服务提供端口为61617：

<transportConnectors>

         <transportConnectorname="openwire"uri="tcp://0.0.0.0:61617?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

5）修改Broker-B节点中的持久化方式：

      <persistenceAdapter>

           <kahaDB directory="/localhost/kahadb"/>

        </persistenceAdapter>

6）在Broker-C节点中添加networkConnector节点：

<networkConnectors>

                <networkConnector   uri="static:(tcp:// 0.0.0.0:61616)"duplex="false"/>

</networkConnectors>

7）修改Broker-C节点中的服务提供端口为61618：

<transportConnectors>

         <transportConnectorname="openwire"uri="tcp://0.0.0.0:61618?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

8）修改Broker-B节点中的持久化方式：

      <persistenceAdapter>

           <kahaDB directory="/localhost/kahadb"/>

       </persistenceAdapter>

9）分别启动broker-A、broker-B、broker-C，因为是broker-B先启动，所以“/localhost/kahadb”目录被lock住，broker-C将一直处于挂起状态，当人为停掉broker-B之后，broker-C将获取目录“/localhost/kahadb”的控制权，重新与broker-A组成cluster提供服务。