**ActiveMq**

#### ActiveMq基础

摘自：https://www.cnblogs.com/cyfonly/p/6380860.html

1. 简介
2. ActiveMQ

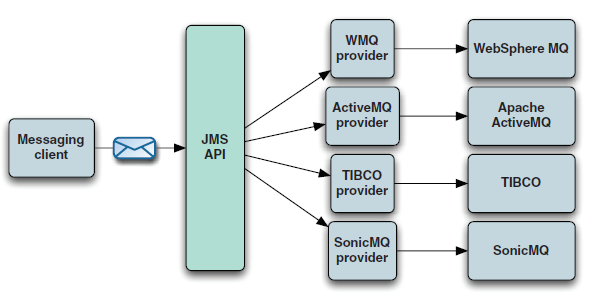
ActiveMQ是一个 MOM，具体来说是一个实现了 JMS 规范的系统间远程通信的消息代理。

1. MOM

MOM 就是面向消息中间件（Message-oriented middleware），是用于以分布式应用或系统中的异步、松耦合、可靠、可扩展和安全通信的一类软件。MOM 的总体思想是它作为消息发送器和消息接收器之间的消息中介,这种中介提供了一个全新水平的松耦合。

1. JMS

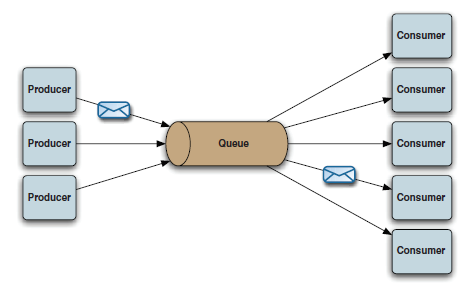
JMS 叫做 Java 消息服务（Java Message Service）,是 Java 平台上有关面向 MOM 的技术规范，旨在通过提供标准的产生、发送、接收和处理消息的 API 简化企业应用的开发，类似于 JDBC 和关系型数据库通信方式的抽象。



1. 通信方式
2. 点对点---P2P

点对点的消息域使用Queue作为Destination，消息可以被同步和异步的发送与接受，每个消息只会给一个Consumer传送一次。Consumer可以使用MessageConsumer.receive()，同步的接受消息，也可以使用MessageConsumer.setMessageListener()注册一个MessageListener实现异步接收。

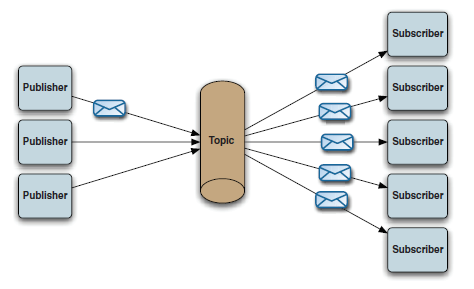
多个Consumer可以注册到同一个Queue上，但一个消息只能被一个Consumer所接收，并由Consumer确认消息。并且在这种情况下，Provider对所有注册的Consumer以轮询的方式发送消息。



1. 发布订阅---Pub/Sub

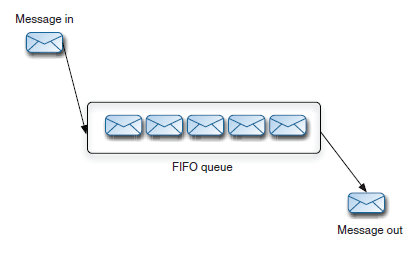
发布订阅使用Topic作为Destination，发布者向Topic发布消息，订阅者接收来自Topic的消息。发送到Topic的任何消息都将自动传给所有的订阅者。接收方式（同步/异步）与P2P相同。

除非显示指定，否则Topic不会为订阅者保留消息。当然，可以通过持久化（Durable）订阅来实现消息的保存。在这种情况下，当订阅者与Provider断开连接时，Provider会为它存储消息。当订阅者重新连接时，会受到断连期间所有未消费的消息。



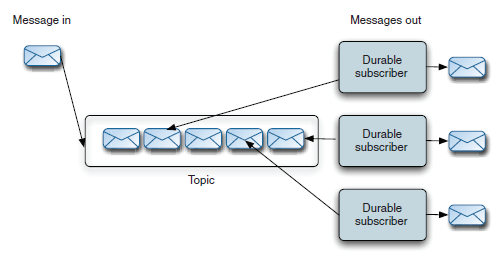
1. Message存储
2. P2P

ActiveMq在Queue中存储Message时，采用先进先出的顺序存储。同一时间一个消息被分配给单个消费者，且只有当Message被消费并确认时才能从Queue中删除。



1. Pub/Sub

对于持久化订阅者来说，每个消费者获得Message的副本。为了节省存储空间，Provider仅存储消息的一个副本。持久化订阅者维护了一个指向下一个Message的指针，并将其副本分配给消费者。以这种方式实现消息存储，因为每个持久化订阅者可能以不同的速率消费Message，或者它们可能不是同时运行。此外，每个Message可能有多个消费者，所以在它被成功的传递给所有的持久化订阅者之前，不能从内存中删除。



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 消息类型 | 是否持久化 | 是否有Durable订阅者 | 消费者延迟启动时，消息是否保留 | Broker重启时，消息是否保留 |
| Queue | N | - | Y | N |
| Queue | Y | - | Y | Y |
| Topic | N | N | N | N |
| Topic | N | Y | Y | N |
| Topic | Y | N | N | N |
| Topic | Y | Y | Y | Y |

1. 存储方式
2. KahaDB

ActiveMQ 5.3 版本起的默认存储方式。KahaDB存储是一个基于文件的快速存储消息，设计目标是易于使用且尽可能快。它使用基于文件的消息数据库意味着没有第三方数据库的先决条件。

要启用 KahaDB 存储，需要在 activemq.xml 中进行以下配置：

<broker brokerName="broker" persistent="true" useShutdownHook="false">

<persistenceAdapter>

<kahaDB directory="${activemq.data}/kahadb" journalMaxFileLength="16mb"/>

</persistenceAdapter>

</broker>

1. AMQ

与 KahaDB 存储一样，AMQ存储使用户能够快速启动和运行，因为它不依赖于第三方数据库。AMQ 消息存储库是可靠持久性和高性能索引的事务日志组合，当消息吞吐量是应用程序的主要需求时，该存储是最佳选择。但因为它为每个索引使用两个分开的文件，并且每个 Destination 都有一个索引，所以当你打算在代理中使用数千个队列的时候，不应该使用它。

<persistenceAdapter>

<amqPersistenceAdapter

directory="${activemq.data}/kahadb"

syncOnWrite="true"

indexPageSize="16kb"

indexMaxBinSize="100"

maxFileLength="10mb" />

</persistenceAdapter>

1. JDBC

选择关系型数据库，通常的原因是企业已经具备了管理关系型数据的专长，但是它在性能上绝对不优于上述消息存储实现。事实是，许多企业使用关系数据库作为存储，是因为他们更愿意充分利用这些数据库资源。

<beans>

<broker brokerName="test-broker" persistent="true" xmlns="http://activemq.apache.org/schema/core">

<persistenceAdapter>

<jdbcPersistenceAdapter dataSource="#mysql-ds"/>

</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>

</beans>

1. 内存存储

内存消息存储器将所有持久消息保存在内存中。在仅存储有限数量 Message 的情况下，内存消息存储会很有用，因为 Message 通常会被快速消耗。在 activema.xml 中将 broker 元素上的 persistent 属性设置为 false 即可。

<broker brokerName="test-broker" persistent="false" xmlns="http://activemq.apache.org/schema/core">

<transportConnectors>

<transportConnector uri="tcp://localhost:61635"/>

</transportConnectors>

</broker>

1. 基于zookeeper的存储
2. 部署模式
3. 单例模式
4. 无共享主从模式

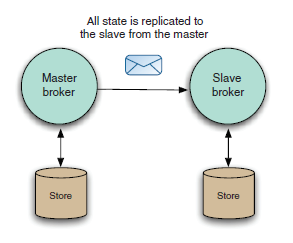
这是最简单的 Provider 高可用性的方案，主从节点分别存储 Message。从节点需要配置为连接到主节点，并且需要特殊配置其状态。在可以接受一些故障停机时间的情况下，可以使用该模式。

所有消息命令（消息，确认，订阅，事务等）都从主节点复制到从节点，这种复制发生在主节点对其接收的任何命令生效之前。并且，当主节点收到持久消息，会等待从节点完成消息的处理（通常是持久化到存储），然后再自己完成消息的处理（如持久化到存储）后，再返回对 Producer 的回执。

从节点不启动任何传输，也不能接受任何客户端或网络连接，除非主节点失效。当主节点失效后，从节点自动成为主节点，并且开启传输并接受连接。这是，使用 failover 传输的客户端就会连接到该新主节点。

但是，这种部署模式有一些限制，

* 主节点只会在从节点连接到主节点时复制其活动状态，因此当从节点没有连接上主节点之前，任何主节点处理的 Message 或者消息确认都会在主节点失效后丢失。不过你可以通过在主节点设置 waitForSlave 来避免，这样就强制主节点在没有任何一个从节点连接上的情况下接受连接。
* 就是主节点只能有一个从节点，并且从节点不允许再有其他从节点。
* 把正在运行的单例配置成无共享主从，或者配置新的从节点时，你都要停止当前服务，修改配置后再重启才能生效。



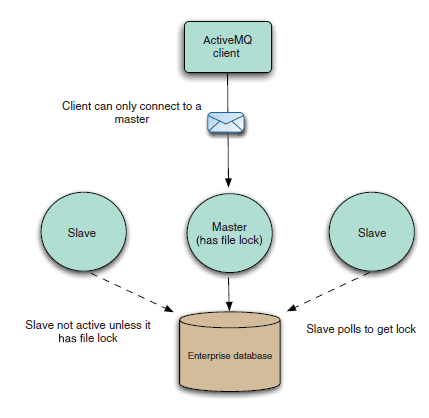
1. 共享存储主从模式

允许多个代理共享存储，但任意时刻只有一个是活动的。这种情况下，当主节点失效时，无需人工干预来维护应用的完整性。另外一个好处就是没有从节点数的限制。

有两种细分模式：

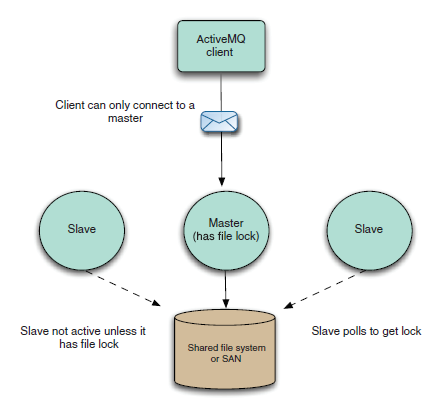
* 1. 基于数据库

它会获取一个表上的排它锁，以确保没有其他 ActiveMQ 代理可以同时访问数据库。其他未获得锁的代理则处于轮询状态，就会被当做是从节点，不会开启传输也不会接受连接。



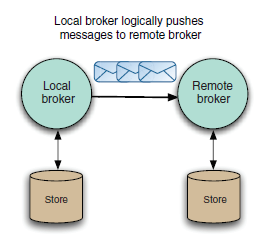
* 1. 基于文件系统

需要获取分布式共享文件锁，linux 系统下推荐用 GFS2。



1. 网络连接
2. 代理网络

支持将 ActiveMQ 消息代理链接到不同拓扑，这就是被人们熟知的代理网络。ActiveMQ 网络使用存储和转发的概念，其中消息总是存储在本地代理中，然后通过网络转发到另一个代理。



当连接建立后，远程代理将把包含其所有持久和活动消费者目的地的信息传递给本地代理，本地代理根据信息决定远程代理感兴趣的 Message 并将它发送给远程代理。

如果希望网络是双向的，您可以使用网络连接器将远程代理配置为指向本地代理，或将网络连接器配置为双工，以便双向发送消息。

<networkConnectors>

<networkConnector uri="static://(tcp://backoffice:61617)"

name="bridge"

duplex="true"

conduitSubscriptions="true"

decreaseNetworkConsumerPriority="false">

</networkConnector>

</networkConnectors>

注意，配置的顺序很重要：

网络连接——需要在消息存储前建立好连接，对应 networkConnectors 元素  
    消息存储——需要在传输前配置好，对应 persistenceAdapter 元素  
    消息传输——最后配置，对应 transportConnectors 元素

1. 网络发现
2. 动态发现

使用多播来支持网络动态发现。配置如下：

<networkConnectors>

　　　　<networkConnector uri="multicast://default"/>

</networkConnectors>

其中，multicast:// 中的默认名称表示该代理所属的组。因此使用此方式时，强烈推荐你使用一个独特的组名，避免你的代理连接到其他不相关代理。

1. 静态发现

静态发现接受代理 URI 列表，并将尝试按列表中确定的顺序连接到远程代理。

<networkConnectors>

　　　　<networkConnector uri="static:(tcp://remote-master:61617,tcp://remote-slave:61617)"/>

</networkConnectors>

相关配置如下：

initialReconnectDelay：默认值1000，表示尝试连接前的时延。

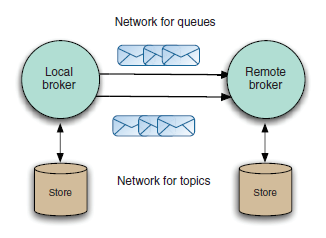
maxReconnectDelay：默认值30000，表示连接失败后到重新建立连接之间的时延，仅在 useExponentialBackOff 启用时生效。

useExponentialBackOff：默认值 true，如果启用，表示每次失败后增加重建连接的时延。

backOffMultiplier：默认值2，表示启用 useExponentialBackOff 后每次的时延增量需要注意的是，网络连接将始终尝试建立到远程代理的连接。

需要注意的是，网络连接将始终尝试建立到远程代理的连接。

1. 多链接场景



当网络负载高时，使用多连接很有意义。但是你需要确保不会重复传递消息，这可以通过过滤器来实现。

<networkConnectors>

　　　　<networkConnector uri="static://(tcp://remotehost:61617)"

name="queues\_only"

duplex="true"

　　　　　　　　<excludedDestinations>

　　　　　　　　　　　　<topic physicalName=">"/>

　　　　　　　　</excludedDestinations>

　　　　</networkConnector>

　　　　<networkConnector uri="static://(tcp://remotehost:61617)"

name="topics\_only"

duplex="true"

　　　　　　　　<excludedDestinations>

　　　　　　　　　　　　<queue physicalName=">"/>

　　　　　　　　</excludedDestinations>

　　　　</networkConnector>

</networkConnectors>

#### ActiveMq应用场景

消息队列中间件是分布式系统中重要的组件，主要解决应用耦合，异步消息，流量削锋等问题。实现高性能，高可用，可伸缩和最终一致性架构。是大型分布式系统不可缺少的中间件。

1. 异步处理

场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种

1.串行的方式；2.并行方式。

（1）串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端。



（2）并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间。



假设三个业务节点每个使用50毫秒钟，不考虑网络等其他开销，则串行方式的时间是150毫秒，并行的时间可能是100毫秒。

因为CPU在单位时间内处理的请求数是一定的，假设CPU1秒内吞吐量是100次。则串行方式1秒内CPU可处理的请求量是7次（1000/150）。并行方式处理的请求量是10次（1000/100）。

小结：如以上案例描述，传统的方式系统的性能（并发量，吞吐量，响应时间）会有瓶颈。如何解决这个问题呢？

引入消息队列，将不是必须的业务逻辑，异步处理。改造后的架构如下：



按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是50毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是50毫秒。因此架构改变后，系统的吞吐量提高到每秒20 QPS。比串行提高了3倍，比并行提高了两倍。

1. 应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。



传统模式的缺点：

1） 假如库存系统无法访问，则订单减库存将失败，从而导致订单失败；

2） 订单系统与库存系统耦合；

如何解决以上问题呢？引入应用消息队列后的方案：



订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功。

库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作。

假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦。

1. 流量削峰

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛。

应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

（1）可以控制活动的人数；

（2）可以缓解短时间内高流量压垮应用；

用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面；

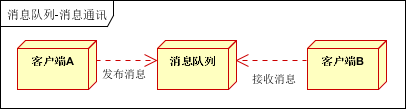
秒杀业务根据消息队列中的请求信息，再做后续处理。

1. 消息通讯

消息通讯是指，消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等。

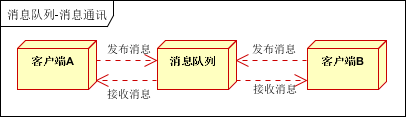
点对点通讯：

客户端A和客户端B使用同一队列，进行消息通讯。



聊天室通讯：

客户端A，客户端B，客户端N订阅同一主题，进行消息发布和接收。实现类似聊天室效果。



#### ActiveMq的特点

1. 特点

异步/顺序读写/高性能/协议简单。异步消息机制保证正确发送，不保证及时发送，短信就是这样

1. 优点

（1） activemq可以很好的运行在任何JVM上，而不只是集成到JBoss的应用服务器中

（2） activemq支持大量的跨语言客户端

（3） activemq支持许多不同的协议，如Ajax，REST，Stomp，OpenWire，XMPP

（4）activemq支持许多高级功能，例如MessageGroups，ExclusiveConsumer，CompositeDestinations

（5） AdvisoryMessage

（6） activemq支持可靠连接并且具有可配置的自动重连接

（7） activemq对Spring有很好的支持

（8） activemq支持跨网络的分布式目的地

（9） activemq是速度非常快；一般要比jbossmq快10倍缺点

1. 缺点

ActiveMQ默认的配置性能偏低，需要优化配置，但是配置文件复杂，ActiveMQ本身不提供管理工具；示例代码少；主页上的文档看上去比较全面，但是缺乏一种有效的组织方式，文档只有片段，用户很难由浅入深进行了解，二、文档整体的专业性太强。在研究阶段可以通过查maillist、看Javadoc、分析源代码来了解。