# 消息中间件

## 定义

基于队列和消息传递技术，在网络环境中提供异步或同步，可靠的消息传输的支持性系统；一般认为，消息中间件属于分布式系统中一个子系统，**关注于数据的发送和接收**，利用高效可靠的异步消息传递机制对分布式系统中的其余各个子系统进行集成

## 为什么要用消息中间件

消息中间件主要解决分布式系统之间消息的传递，同时为分布式系统中的子系统提供了伸缩性和扩展性，好处有：

低耦合：不同系统或者服务之间可以使用消息中间件进行间接通信

异步通信：子系统之间得以充分执行自己的逻辑而无需等待。

缓冲能力：消息中间件像是一个巨大的蓄水池，将高峰期大量的请求存储下来慢慢交给后台进行处理，对于秒杀业务来说尤为重要。

## 应用场景

### 异步处理

场景说明：用户注册后，需要发送邮件和注册短信，传统做法有两种：串行方式和并行方式

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。



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。将消息写入消息队列中，从消息队列中取出来进行逻辑处理



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。

### 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。

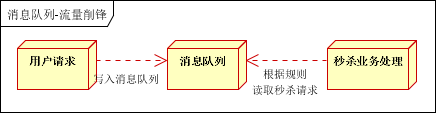


假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦。

### 流量削峰

流量削峰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：可以控制活动的人数；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。

### 日志处理

日志处理是指将消息队列用在日志处理中，比如Kafka的应用，解决大量日志传输的问题。架构简化如下：



日志采集客户端，负责日志数据采集，定时写入Kafka队列：Kafka消息队列，负责日志数据的接收，存储和转发；日志处理应用：订阅并消费kafka队列中的日志数据；

### 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。

# JMS

Java消息服务（java message service）java平台面向消息中间件的api,用于多个应用程序或者分布式系统之间进行消息异步通信

Jms是一个与具体平台无关的api,绝大多数mom（面向消息的中间件），提供了对jms的支持ActiveMq就是这个规范的一个具体实现

## Jms对象模型

ConnectionFactory 连接工厂，负责创建一个jms连接

Connection jms连接，表示客户端和服务端之间的一个活动的连接，是由客户端通过调用连接工厂的方法创建的； 封装客户端与JMS provider之间的一个虚拟的连接

Session jms会话表示jms客户端和服务端之间的会话状态。Jms会话建立在jms连接上，表示客户端和服务端之间的一个会话过程 生产和消费消息的一个单线程上下文； 用于创建producer、consumer、message、queue..\

Destination 消息发送或者消息接收的目的地

MessageProducer/consumer 消息生产者/消费者,实例通过session创建，用来发送和接收消息

消费消息的方式

1. 同步消费：通过调用消费者的receive方法从目的地中显示的提前消息，receive方法可以一直阻塞直到获取到了消息
2. 异步消费：为消费者注册一个监听，消息到达时进行业务处理

## Jms中的消息

Jms中消息由以下三种元素组成

消息头：包含消息的识别信息和路由信息

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

消息体：JMS 定义的消息类型有TextMessage、MapMessage、BytesMessage、StreamMessage 和 ObjectMessage。

## Jms中的消息模型

### 点对点(p2p)

每个消息只能有一个消费者

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生产发送消息时是否处于运行状态，当消费者去提前消息时都会获取消息

如果session关闭时，有些消息以及受到，但还没有被签收，那么当消费者下次连接到相同队列时，消息还会被签收

如果用户在receive方法中设定了消息选择条件，那么不符合条件的消息会留在队列中不会被接收

队列可以长久保存消息直到消息被消费者签收。消费者不需要担心因为消息丢失而时刻与jms provider保持连接状态

### 发布订阅

每个消息可以有多个消费者

消息的生产者和消费者之间存在时间上的相关性，**订阅一个主题的消费者只能消费自它订阅之后发布的消息**。JMS规范允许提供客户端创建持久订阅

订阅可以分为非持久化订阅和持久化订阅

当需要所有消息必须接收的时候，则需要用到持久订阅，反之则用非持久化订阅

# ActiveMq的使用

## 安装部署

下载 Windows版 ActiveMQ，解压，运行bin目录下的activemq.bat即可。Linux下操作类似（进入bin目录运行./activemq start启动，./activemq stop关闭）。

下载地址：http://activemq.apache.org/activemq-580-release.html

运行后在浏览器中访问<http://127.0.0.1:8161/admin>，即可看到ActiveMQ的管理控制台

ActiveMQ中，61616为服务端口，8161为管理控制台端口。

## 消息可靠性机制

JMS消息之后被确认后，才会认为是被成功消费。消息的消费包含三个阶段： 客户端接收消息、客户端处理消息、消息被确认

消息的持久化

1. 持久化
2. 非持久化

### 创建会话

Connection是连接对象，这个的创建略过

Session session = connection.createSession(Boolean.FALSE, Session.AUTO\_ACKNOWLEDGE);

第一个参数:是否支持事务，如果为true，则会忽略第二个参数，自动被jms服务器设置为SESSION\_TRANSACTED

### 事务性会话

（createSession的第一个参数为true）

生产者调用session的commit方法来确认将消息发送到了broke，事务中的消息会异步发送，send()方法不会被阻塞，但是commit方法会阻塞，直到收到确认消息，表示broker已经成功接收到消息

消费者调用session的commit方法来签收消息

**在事务性会话中一个消息被成功提交则消息被自动化签收，如果事务回滚，则消息会被再次传送**

### 非事务性会话

（createSession的第一个参数为false）

**在这种模式下，消息何时被确认取决于创建会话时设置的应答模式**

**AUTO\_ACKNOWLEDGE**:当客户端成功调用recive方法返回之后，或者MessageListener.onMessage方法成功返回之后会自动确认消息

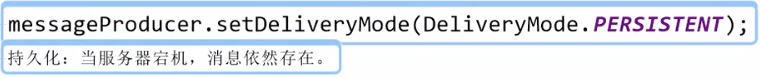
指定消息接收者在每次收到消息时自动发送确认。消息只向目标发送一次，但传输过程中可能因为错误而丢失消息。

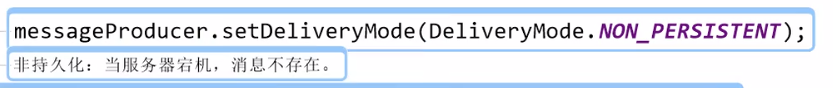
**CLIENT\_ACKNOWLEDGE**：客户端调用acknowledge()方法确认消息；在这种模式中，如果一个消息消费者消费一共是10个消息，那么消费了5个消息，然后在第5个消息通过textMessage.acknowledge()，那么之前的所有消息都会被消确认

**DUPS\_OK\_ACKNOWLEDGE：**指定消息提供者在消息接收者没有确认发送时重新发送消息（这种确认模式不在乎接收者收到重复的消息）。

### 消息持久化

Java默认是持久化的





默认情况下，topic发布时，如果订阅者不在线。消息就会丢失但我们可以用持久化Topic来解决这个问题。

## 消息发送策略

生产者

默认情况下生产者发送的消息是持久化的。消息发送到broker之后，producer会等待broker对这条消息的处理情况的反馈

使用send() 方法向ActiveMQ发送消息，默认情况下，持久化消息以同步方式发送，send() 方法会被阻塞，直到 broker 发送一个确认消息给生产者，这个确认消息表示broker已经成功接收到消息，

也可以设置消息发送持久化消息的异步方式

connectionFactory.setUseAsyncSend(true);

回执窗口大小设置

connectionFactory.setProducerWindowSize();

设置非持久化消息

textMessage.setJMSDeliveryMode(DeliveryMode.NON\_PERSISTENCE);

非持久化消息模式下，默认的就是异步发送过程，如果需要对非持久化消息的每次发送的消息都获得broker的回执的话，调用setAlwaysSyncSend方法

connectionFactory.setAlwaysSyncSend();

消费者

默认情况下，mq服务器（broker）采用异步方式向客户端主动推送消息(push)。也就是说broker在向某个消费者会话推送消息后，不会

等待消费者响应消息，直到消费者处理完消息以后，主动向broker返回处理结果

Prefetchsize 预取消息数量

broker端一旦有消息，就主动按照默认设置的规则推送给当前活动的消费者。 每次推送都有一定的数量限制，而这个数量就是prefetchSize；假如prefetchSize=0 .此时对于consumer来说，就是一个pull模式

消息确认

ACK\_TYPE，消费端和broker交换ack指令的时候，还需要告知brokerACK\_TYPE。

ACK\_TYPE表示确认指令的类型，broker可以根据不同的ACK\_TYPE去针对当前消息做不同的应对策略

REDELIVERED\_ACK\_TYPE (broker会重新发送该消息)重发侧策略

DELIVERED\_ACK\_TYPE 消息已经接收，但是尚未处理结束

STANDARD\_ACK\_TYPE 表示消息处理成功

## ActiveMQ消息持久化

### KabaDB默认的存储方式

在配置文件activemq.xml中添加如下配置即可

<persistenceAdapter>

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

</persistenceAdapter>

KahaDB是基于文件的本地数据库储存形式，虽然没有AMQ的速度快，但是它具有强扩展性，恢复的时间比AMQ短，从5.4版本之后KahaDB做为默认的持久化方式。

**由于KahaDB是默认的持久化存储方案。所以即使你不配置任何的KahaDB参数信息，ActiveMQ也会启动KahaDB**。这种情况下，KahaDB文件所在位置是你的ActiveMQ安装路径下的/data/KahaDB子目录。

### AMQ消息存储

基于文件的存储方式，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，如果一条消息的大小超过了32M，那么这个值必须设置大一点。当一个存储文件中的消息已经全部被消费，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。

### JDBC消息存储

消息基于JDBC数据库存储的

ACTIVEMQ\_ACKS： 存储持久订阅的信息

ACTIVEMQ\_LOCK： 锁表（用来做集群的时候，实现master选举的表）

ACTIVEMQ\_MSGS： 消息表

Jdbc持久化方式

1.修改配置文件conf/activemq.xml：

将其中的这段配置：

<persistenceAdapter>

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

</persistenceAdapter>

修改为为：

<persistenceAdapter>

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

</persistenceAdapter>

（在persistenceAdapter 中将标签kahadb，修改为jdbcPersistenceAdapter，增加属性dataSource）

2、然后在</**broker**>标签后，增加数据源的配置：

<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://localhost:3306/activemq?relaxAutoCommit=true&amp;useUnicode=true&amp;characterEncoding=utf-8&amp;serverTimezone=UTC"/>

<property name="username" value="root"/>

<property name="password" value="123456"/>

<property name="poolPreparedStatements" value="true"/>

</bean>

其中**?relaxAutoCommit=true**必须有，其他的属性根据数据库的配置自行决定。

3、将mysql-connector-java-5.1.34-bin.jar（版本可以自行选择）放到ActiveMQ的/ lib目录下。

4、在Mysql数据库中增加在连接字符串中设置的数据库名activemq

5、运行后，会发现在库中增加了3个表

**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

**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。

**activemq\_msgs：**用于存储消息，Queue和Topic都存储在这个表中。主要的数据库字段如下：

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

### Memory 基于内存的存储

基于内存的消息存储，内存不属于持久化

### 死信队列

用来保存处理失败或者过期的消息

出现下面情况时，消息会被重发：

1.事务会话被回滚。

2.事务会话在提交之前关闭。

3.会话使用CLIENT\_ACKNOWLEDGE模式，并且Session.recover()被调用。

4.自动应答失败

当一个消息被重发超过最大重发次数（缺省为6次，消费者端可以修改）时，会给broker发送一个"有毒标记“，这个消息被认为是有问题，这时broker将这个消息发送到**死信队列**，以便后续处理。

在配置文件(activemq.xml)来调整死信发送策略。



可以单独使用死信消费者处理这些死信。

# Activmq集群

## 静态网络连接

NetworkConnector

主要用来配置broker与broker之间的通信连接

修改activemq.xml，增加如下内容

<networkConnectors>

<networkConnector uri="static://(tcp://192.168.0.101:61616,tcp://192.168.0.102:61616)"/>

</networkConnectors>

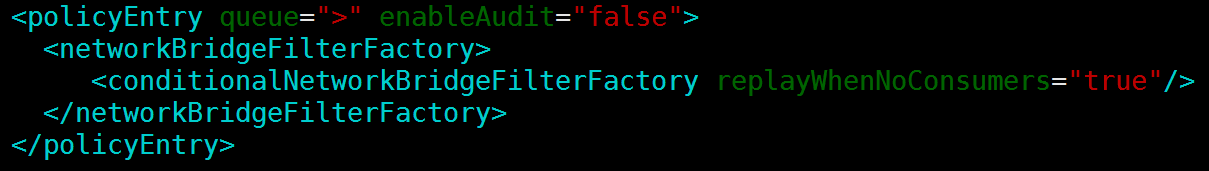
两个Brokers通过一个staic的协议来进行网络连接。一个Consumer连接到BrokerB的一个地址上，当Producer在BrokerA上以相同的地址发送消息是，此时消息会被转移到BrokerB上，也就是说BrokerA会转发消息到BrokerB上

### 丢失的消息

一些consumer连接到broker1、消费broker2上的消息。消息先被broker1从broker2消费掉，然后转发给这些consumers。假设，转发消息的时候broker1重启了，这些consumers发现brokers1连接失败，通过failover连接到broker2.但是因为有一部分没有消费的消息被broker2已经分发到broker1上去了，这些消息就好像消失了。除非有消费者重新连接到broker1上来消费

从5.6版本开始，在destinationPolicy上新增了一个选项replayWhenNoConsumers属性，这个属性可以用来解决当broker1上有需要转发的消息但是没有消费者时，把消息回流到它原始的broker。同时把enableAudit设置为false，为了防止消息回流后被当作重复消息而不被分发

通过如下配置，在activeMQ.xml中。 分别在两台服务器都配置。即可完成消息回流处理



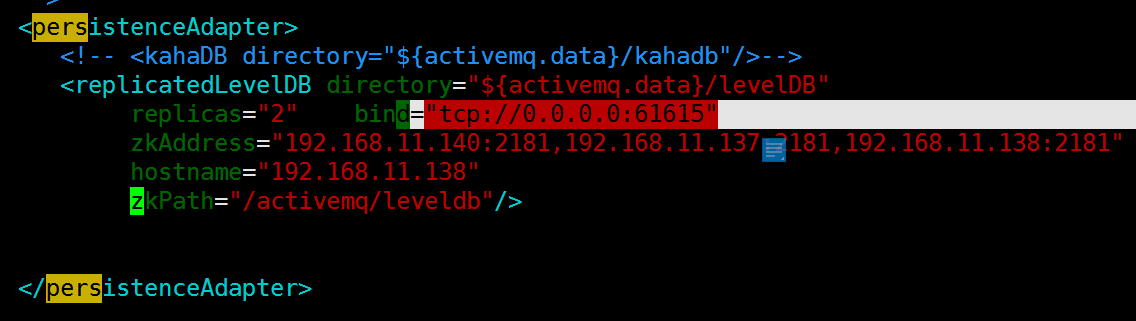
## 动态网络连接

networkConnector是一个高性能方案，但不是一个高可用方案

通过zookeeper和activemq实现高可用

（master/slave模型）

1.修改activeMQ



2. 启动zookeeper服务器

3. 启动activeMQ

参数的意思

directory： levelDB数据文件存储的位置

replicas：计算公式（replicas/2）+1 ， 当replicas的值为2的时候， 最终的结果是2. 表示集群中至少有2台是启动的

bind: 用来负责slave和master的数据同步的端口和ip

zkAddress： 表示zk的服务端地址

hostname：本机ip

## 其他方案

Jdbc存储的主从方案：基于LOCK锁表的操作来实现master/slave

基于共享文件系统的主从方案：挂载网络磁盘，将数据文件保存到指定磁盘上即可完成master/slave模式

# ActiveMQ生产者和消费者优化策略

## 一、生产者优化策略

默认情况下，ActiveMQ服务端认为生产者端发送的是PERSISTENT Message。所以如果要发送NON\_PERSISTENT Message，那么生产者端就要明确指定。发送NON\_PERSISTENT Message时，消息发送方默认使用异步方式：即是说消息发送后发送方不会等待NON\_PERSISTENT Message在服务端的任何回执。为避免MQ消息堆积但发送方不知道无法采取策略的情况，消息发送者会在发送一定大小的消息后等待服务端进行回执，可以通过代码设置回执点或者设置每次都等待服务端回执。connectionFactory.setProducerWindowSize(102400); 设置消息发送者在累计发送102400byte大小的消息后（可能是一条消息也可能是多条消息）

等待服务端进行回执,以便确定之前发送的消息是否被正确处理，确定服务器端是否产生了过量的消息堆积，需要减慢消息生产端的生产速度。

如果您不特意指定消息的发送类型，那么消息生产者默认发送PERSISTENT Meaage。这样的消息发送到ActiveMQ服务端后将被进行持久化存储（比较耗时），并且消息发送者默认等待ActiveMQ服务端对这条消息处理情况的回执。为了提高ActiveMQ在接受PERSISTENT Meaage时的性能，ActiveMQ允许开发人员遵从JMS API中的设置方式，为消息发送端在发送PERSISTENT Meaage时提供异步方式，connectionFactory.setUseAsyncSend(true);此时要设置回执点。

JMS规范中支持带事务的消息，也就是说您可以启动一个事务（并由消息发送者的连接会话设置一个事务号Transaction ID），然后在事务中发送多条消息。这个事务提交前这些消息都不会进入队列（无论是Queue还是Topic）。

生产流控制，是ActiveMQ消息生产者端最为重要的性能策略，它主要设定了在ActiveMQ服务节点在产生消息堆积，并超过限制大小的情况下，如何进行消息生产者端的限流。在ActiveMQ的主配置文件activemq.xml中，关于ProducerFlowControl策略的控制标签是“destinationPolicy”和它的子标签，可以配置每个队列是否启用生产者流控，以及每个Queue的最大内存限制。有关于policyEntry标签的所有配置选项都有完整说明：http://activemq.apache.org/per-destination-policies.html。

## 二、消费者端优化策略

比起消息生产者来说消息消费者的性能更能影响ActiveMQ系统的整体性能，因为要成功完成一条消息的处理，它的工作要远远多于消息生产者。默认情况下ActiveMQ服务端采用异步方式向客户端推送消息。也就是说ActiveMQ服务端在向某个消费者会话推送消息后，不会等待消费者的响应信息，直到消费者处理完消息后，主动向服务端返回处理结果。

ActiveMQ系统中，默认的策略是ActiveMQ服务端一旦有消息，就主动按照设置的规则推送给当前活动的消费者。其中每次推送都有一定的数量限制，这个限制值就是prefetchSize。针对Queue工作模型的队列和Topic工作模型的队列，ActiveMQ有不同的默认“预取数量”；针对NON\_PERSISTENT Message和PERSISTENT Message，ActiveMQ也有不同的默认“预取数量”：

PERSISTENT Message—Queue：prefetchSize=1000

NON\_PERSISTENT Message—Queue：prefetchSize=1000

PERSISTENT Message—Topic：prefetchSize=100

NON\_PERSISTENT Message—Topic：prefetchSize=32766

ActiveMQ中设置的各种默认预取数量一般情况下不需要进行改变。但是非必要情况下，请不要设置prefetchSize=1，因为这样就是一条一条的取数据；也不要设置为prefetchSize=0，因为这将导致关闭服务器端的推送机制，改为客户端主动请求。

JMS规范除了为消息生产者端提供事务支持以外，还为消费服务端准备了事务的支持。您可以通过在消费者端操作事务的commit和rollback方法，向服务器告知一组消息是否处理完成。采用事务的意义在于，一组消息要么被全部处理并确认成功，要么全部被回滚并重新处理。

如果一条消息被不断的处理失败，那么最可能的情况就是这条消息承载的业务内容本身就有问题。那么无论重发多少次，这条消息还是会处理失败。为了解决这个问题，ActiveMQ中引入了“死信队列”（Dead Letter Queue）的概念。即一条消息再被重发了多次后（默认为重发6次redeliveryCounter==6），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。默认情况下“死信队列”只接受PERSISTENT Message，如果NON\_PERSISTENT Message超过了重发上限，将直接被删除。