消息队列

## 简介

消息队列是一种应用程序对应用程序的通信方法。应用程序通过读写入列出列来通信。消息队列可驻留在磁盘或内存上，队列存储消息直到他们被出列。是分布式应用交换信息的一种技术。

消息队列是一个至关重要的组件，对于任何架构和应用。下面是使用消息队列的几点理由

1. 解耦
2. 冗余
3. 扩展性
4. 灵活性&峰值处理能力
5. 可恢复性
6. 送达保证
7. 排序保证
8. 缓冲
9. 理解数据流
10. 异步通信

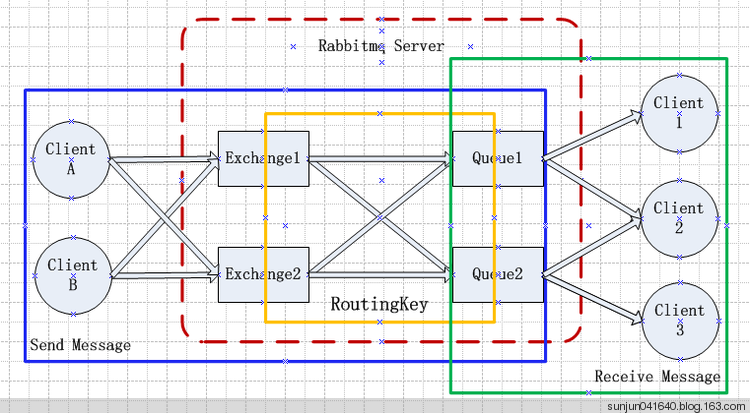
## 主流的消息队列产品

### RabbitMQ

* 1. **简介:**

RabbitMQ是使用Erlang编写的一个开源的消息队列，本身支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正因如此，它非常重量级，更适合于企业级的开发。同时实现了Broker构架，这意味着消息在发送给客户端时先在中心队列排队。对路由、载均衡或者数据持久化都有很好的支持。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。

RabbitMQ的结构如下：



概念说明：

Broker：简单来说就是消息队列服务器实体。  
Exchange：消息交换机，它指定消息按什么规则，路由到哪个队列。  
Queue：消息队列载体，每个消息都会被投入到一个或多个队列。  
Binding：绑定，它的作用就是把exchange和queue按照路由规则绑定起来。  
Routing Key：路由关键字，exchange根据这个关键字进行消息投递。  
vhost：虚拟主机，一个broker里可以开设多个vhost，用作不同用户的权限分离。  
producer：消息生产者，就是投递消息的程序。  
consumer：消息消费者，就是接受消息的程序。  
channel：消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务。

消息队列的使用过程大概如下：

（1）客户端连接到消息队列服务器，打开一个channel。  
（2）客户端声明一个exchange，并设置相关属性。  
（3）客户端声明一个queue，并设置相关属性。  
（4）客户端使用routing key，在exchange和queue之间建立好绑定关系。  
（5）客户端投递消息到exchange。

exchange接收到消息后，就根据消息的key和已经设置的binding，进行消息路由，将消息投递到一个或多个队列里。

Exchange的类型：direct fanout topic三种。

RabbitMQ支持消息的持久化，也就是数据写在磁盘上，为了数据安全考虑，我想大多数用户都会选择持久化。消息队列持久化包括3个部分：  
（1）exchange持久化，在声明时指定durable => 1  
（2）queue持久化，在声明时指定durable => 1  
（3）消息持久化，在投递时指定delivery\_mode => 2（1是非持久化）

如果exchange和queue都是持久化的，那么它们之间的binding也是持久化的。如果exchange和queue两者之间有一个持久化，一个非持久化，就不允许建立绑定。

* 1. **集群及高可用环境搭建：**

RabbitMQ的环境依赖于Erlang语言，所以安装rabbitmq运行需要Erlang的环境，

先安装erlang环境，在安装rabbitmq。

集群的配置可以通过命令行，也可以通过配置文件。集群节点分为RAM节点和DISK节点，一个集群最好至少有一个DISK节点保存集群的状态。集群中节点重启加入节点后，不会同步其他节点的消息。如果没有持久化配置，消息及其他组件信息则消失。

命令行配置集群（3台机器node1、node2、node3）：

1. 首先启动3台服务的rabbitmq的服务 rabbitmq-service –detached
2. 然后在节点node2、node3分别执行 停止 rabbitmqctl stop\_app
3. 添加节点rabbitmqctl join\_cluster rabbit@node1 –ram/disc
4. 启动节点 rabbitmqctl start\_app,并查询集群状态 rabbitmqctl cluster\_staus

这样3台机器上的rabbitmq服务组成了一个集群，但是集群中Master节点失效后，后从slave节点中选取最长的queue作为master;集群中Salve节点失效后，本地存在disc也会消失，新加入Salve节点后，数据不会自动同步的,所以这样的集群的不是高可用的。Rabbitmq的集群支持动态添加和删除节点，无需重新启动。

高可用策略：rabbitmq提供了镜像复制的功能。

在master节点上执行：

rabbitmqctl set\_policy -p / ha-allqueue "^" '{"ha-mode":"all"}'

就可以做到集群节点之间 会执行复制策略，保证的消息的完整性及整个集群环境的高可用性。

注意：

Rabbitmq做集群配置的时候，需要对对所有节点上的 .erlang.cookie做同一处理，hostname能够正常解析，Centos路径 /var/lib/rabbitmq/.erlang.cookie

### Redis

* 1. **简介**

Redis是一个基于Key-Value对的NoSQL数据库，开发维护很活跃。虽然它是一个Key-Value数据库存储系统，但它本身支持MQ功能， 所以完全可以当做一个轻量级的队列服务来使用。

Redis提供了两种方式来作消息队列。一个是使用生产者消费模式模式，另外一个方法就是发布订阅者模式。前者会让一个或者多个客户端监听消息队列，一旦消息到达，消费者马上消费，谁先抢到算谁的，如果队列里没有消息，则消费者继续监听。后者也是一个或多个客户端订阅消息频道，只要发布者发布消息，所有订阅者都能收到消息。

### AtiveMQ

* 1. **简介**

ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现,尽管JMS规范出台已经是很久的事情了,但是JMS在当今的J2EE应用中间仍然扮演着特殊的地位。

AtiveMQ是Apache下的一个子项目。它能够以代理和点对点的技术实现队列。同时类似于RabbitMQ，它少量代码就可以高效地实现高级应用场景。

ActiveMQ的消息队列分为两种：点对点消息（queue）和主题消息(topic),点对点消息就是生产/消费者模式 一个消息只能有由一个消费者处理；主题消息就是发布/订阅者模式，一个消息可以有多个消费者处理。

JMS消息

JMS消息是服务提供者和客户端之间传递信息所使用的信息单元。JMS消息由以下三部分组成：消息头（header）、属性（property）和消息体（body）。

消息主体

消息主体包含了消息的核心数据。

JMS 定义了5中消息类型： TextMessage、MapMessage、BytesMessage、

StreamMessage和ObjectMessage

选择最合适的消息类型可以使JMS最有效 的处理消息。

* **TextMessage（文本消息）**

将数据作为简单字符串存放在主体中(XML就可以作为字符串发)

TextMessage msg = session.createTextMessage();

msg.setText(text);

有些厂商支持一种XML专用的消息格式，带来了便利，但是不是标准的JMS类型，影响

移植性。

只自己定义了两个方法setText(String s)、getText()

* **MapMessage（映射消息）**

使用一张映射表来存放其主体内容（参照Jms API）

MapMessage msg = session.createMapMessage();

msg.setString(“CUSTOMER\_NAME”,”John”);

msg.setInt(“CUSTOMER\_AGE”,12);

String s = msg.getString(“CUSTOMER\_NAME”);

int age = msg.getInt(“CUSTOMER\_AGE”);

* **BytesMessage（字节消息）**

将字节流存放在消息主体中。适合于下列情况：必须压缩发送的大量数据、需要与现有

消息格式保持一致等（参照Jms API）

byte[] data;

BytesMessage msg = session.createBytesMessage();

msg.wirte(data);

byte[] msgData = new byte[256];

int bytesRead = msg.readBytes(msgData);

* **StreamMessage（流消息）**

用于处理原语类型。这里也支持属性字段和MapMessage所支持的数据类型。使用这种

消息格式时，收发双方事先协商好字段的顺序，以保证写读顺序相同（参照Jms API）

StringMessage msg = session.createStreamMessage();

msg.writeString(“John”);

msg.writeInt(12);

String s = msg.readString();

Int age = msg.readInt();

（PS：个人认为有点像socket的信息收发）

* **ObjectMessage（对象消息）**

用于往消息中写入可序列化的对象。

消息中可以存放一个对象，如果要存放多个对象，需要建立一个对象集合，然后把这个

集合写入消息。

客户端接收到一个ObjectMessage时，是read-only模式。如果一个客户端试图写

message，将会抛出MessageNotWriteableException。如果调用了clearBody方法，message既可以读又可以写

自己只单独定义了两个方法：getObject()和setObject(Serializable s)

ObjectMessage包含的只是object的一个快照，set之后object的修改对ObjectMessage的body无效 （从两个方法可以看出，这种消息已经强制要你实现java.io. Serializable接口）

Message只读时被set抛出MessageNotWriteableException;

set和get时，如果对象序列化失败抛出MessageFormatException

消息的通信方式（点对点通信和发布/订阅方式）

1. 点对点方式（point-to-point）

点对点的消息发送方式主要建立在 Message Queue、Sender、Receiver上，

Message Queue 存贮消息，Sender 发送消息，Receiver接收消息.具体点就是Sender Client发送Message Queue ,而Receiver Client从Queue中接收消息和"发送消息已接受"到Queue,确认消息接收。消息发送客户端与接收客户端没有时间上的依赖，发送客户端可以在任何时刻发送信息到Queue，而不需要知道接收客户端是不是在运行。

1. 发布/订阅方式（publish/subscriber Messaging）

发布/订阅方式用于多接收客户端的方式.作为发布订阅的方式，可能存在多个接收客户

端，并且接收端客户端与发送客户端存在时间上的依赖。一个接收端只能接收他创建以后发送客户端发送的信息。作为subscriber ,在接收消息时有两种方法，destination的receive方法，和实现message listener 接口的onMessage 方法。

消息产生者向JMS发送消息的步骤

1. 创建连接使用的工厂类JMS ConnectionFactory
2. 使用管理对象JMS ConnectionFactory建立连接Connection
3. 使用连接Connection建立会话Session
4. 使用会话Session和管理对象Destination创建消息生产者MessageSender
5. 使用消息生产者MessageSender发送消息

消息消费者从JMS接受消息的步骤

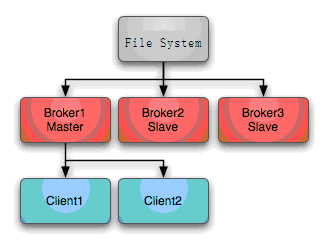
1. 创建连接使用的工厂类JMS ConnectionFactory
2. 使用管理对象JMS ConnectionFactory建立连接Connection
3. 使用连接Connection 建立会话Session
4. 使用会话Session和管理对象Destination创建消息消费者MessageReceiver
5. 使用消息消费者MessageReceiver接受消息，需要用setMessageListener将MessageListener接口绑定到MessageReceiver 消息消费者必须实现了MessageListener接口，需要定义onMessage事件方法。
   1. 集群及高**可用**环境搭建：

ActiveMQ提供了master-slave（高可用）、broker cluster（负载均衡）等多种部署方式

**1、Master-Slave部署方式**

**1）shared filesystem Master-Slave部署方式**

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。

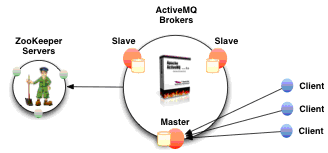


**2）shared database Master-Slave方式**

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-Clusters部署方式(负载均衡方式)**

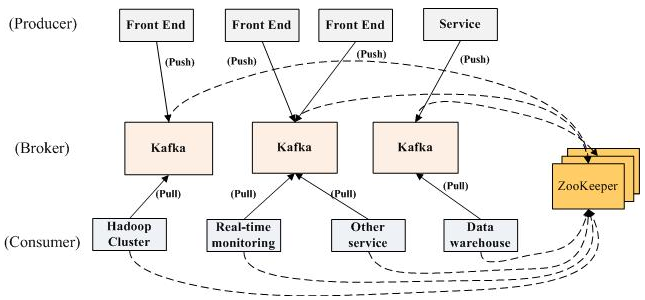
Broker Cluster主要是通过network of Brokers在多个ActiveMQ实例之间进行消息的路由。Broker的集群分为Static Discovery和Dynamic Discovery两种；可以提高消息的消费能力，但不能保证消息的可靠性。

### Kafka

* 1. **简介**

Kafka是分布式发布-订阅消息系统。它最初由LinkedIn公司开发，之后成为Apache项目的一部分。Kafka是一个分布式的，可划分的，冗余备份的持久性的日志服务。它主要用于处理活跃的流式数据。

Kafka的结构



1. Topic：特指Kafka处理的消息源（feeds of messages）的不同分类。一个Topic可以认为是一类消息，每个topic将被分成多个partition(区),每个partition在存储层面是append log文件。任何发布到此partition的消息都会被直接追加到log文件的尾部，每条消息在文件中的位置称为offset（偏移量），offset为一个long型数字，它是唯一标记一条消息。它唯一的标记一条消息。kafka并没有提供其他额外的索引机制来存储offset，因为在kafka中几乎不允许对消息进行“随机读写”。
2. Partition：Topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列。partition中的每条消息都会被分配一个有序的id（offset）。
3. Message：消息，是通信的基本单位，每个producer可以向一个topic（主题）发布一些消息。
4. Producers：消息和数据生产者，向Kafka的一个topic发布消息的过程叫做producers。
5. Consumers：消息和数据消费者，订阅topics并处理其发布的消息的过程叫做consumers。
6. Broker：缓存代理，Kafa集群中的一台或多台服务器统称为broker。

消息处理流程

1. Producer根据指定的partition方法（round-robin、hash等），将消息发布到指定topic的partition里面
2. kafka集群接收到Producer发过来的消息后，将其持久化到硬盘，并保留消息指定时长（可配置），而不关注消息是否被消费。
3. Consumer从kafka集群pull数据，并控制获取消息的offset

Kafka的设计优势

1、吞吐量

高吞吐是kafka需要实现的核心目标之一，为此kafka做了以下一些设计：

1. 数据磁盘持久化：消息不在内存中cache，直接写入到磁盘，充分利用磁盘的顺序读写性能
2. zero-copy：减少IO操作步骤
3. 数据批量发送
4. 数据压缩
5. Topic划分为多个partition，提高parallelism

2、负载均衡

1. producer根据用户指定的算法，将消息发送到指定的partition
2. 存在多个partiiton，每个partition有自己的replica，每个replica分布在不同的Broker节点上
3. 多个partition需要选取出lead partition，lead partition负责读写，并由zookeeper负责fail over
4. 通过zookeeper管理broker与consumer的动态加入与离开

3、拉取系统

由于kafka broker会持久化数据，broker没有内存压力，因此，consumer非常适合采取pull的方式消费数据，具有以下几点好处：

1. 简化kafka设计
2. consumer根据消费能力自主控制消息拉取速度
3. consumer根据自身情况自主选择消费模式，例如批量，重复消费，从尾端开始消费等

4、可扩展性

当需要增加broker结点时，新增的broker会向zookeeper注册，而producer及consumer会根据注册在zookeeper上的watcher感知这些变化，并及时作出调整。

Kafka的应用场景:

1. 常规的消息队列
2. 行为跟踪
3. 信息监控
4. 日志收集

Kafka的缺点:

kafka并没有提供JMS中的"事务性""消息传输担保(消息确认机制)""消息分组"等企业级特性;kafka只能使用作为"常规"的消息系统,在一定程度上,尚未确保消息的发送与接收绝对可靠(比如,消息重发,消息发送丢失等)

## 测试与对比：

网上测试结果，

后期准备自己做一下测试。

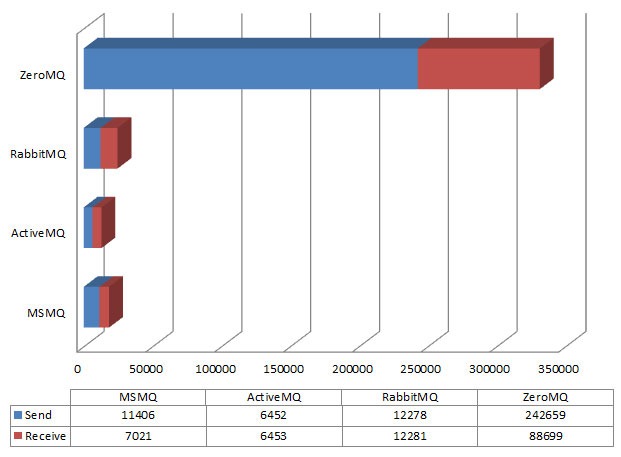
1. rabbitMQ与Redis的对比结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 入队 | | | | 出队 | | | |
|  | 128B | 512B | 1K | 10K | 128B | 512B | 1K | 10K |
| Redis | 16088 | 15961 | 17094 | 25 | 15955 | 20449 | 18098 | 9355 |
| RabbitMQ | 10627 | 9916 | 9370 | 2366 | 3219 | 3174 | 2982 | 1588 |

对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。测试数据分为128Bytes、512Bytes、 1K和10K四个不同大小的数据。实验表明：入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则 慢的无法忍受；出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于Redis。

1. rabbitMQ与ActiveMQ的对比：

下面是测试结果。显示的是发送和接受的每秒钟的消息数。整个过程共产生1百万条1K的消息。测试的执行是在一个Windows Vista上进行的。



## 参考资料：

<http://www.oschina.net/translate/top-10-uses-for-message-queue>（使用消息队列的优点）

<http://www.oschina.net/news/17973/message-queue-shootout>（鸡肋消息队列的对比）

<http://kb.cnblogs.com/page/212710/>(可能并不需要消息队列)

<http://blog.163.com/sir_876/blog/static/11705223201332444647261/>（使用场景）

<http://www.rabbitmq.com/getstarted.html> (RabbitMQ官方文档)

<http://blog.csdn.net/column/details/rabbitmq.html> (RabbitMQ的介绍与使用)

<http://blog.csdn.net/whycold/article/details/41119807> (RabbitMQ的简介)

<http://www.redis.cn/> (Redis 中文社区)

<http://www.cnblogs.com/si812cn/p/4042992.html> (Redis使用场景)

<http://www.redis.cn/topics/benchmarks.html> (Redis 速度评测)

<http://activemq.apache.org/> (Active 官方文档)

<http://www.open-open.com/lib/view/open1400126457817.html>（ActiveMQ的介绍）

<http://blog.csdn.net/column/details/activemq.html> (ActiveMQ的技术详解)

<http://www.cnblogs.com/likehua/p/3999538.html>(Kafka)