几种MQ产品比较

1. MQ产品介绍

ZeroMQ : 扩展性好，开发比较灵活，采用C语言实现，实际上他只是一个socket库的重新封装，如果我们做为消息队列使用，需要开发大量的代码。号称最快的消息队列系统，尤其针对大吞吐量的需求场景。ZMQ能够实现RabbitMQ不擅长的高级/复杂的队列，但是开发人员需要自己组合多种技术框架，技术上的复杂度是对这MQ能够应用成功的挑战。ZeroMQ具有一个独特的非中间件的模式，你不需要安装和运行一个消息服务器或中间件，因为你的应用程序将扮演了这个服务角色。你只需要简单的引用ZeroMQ程序库，可以使用NuGet安装，然后你就可以愉快的在应用程序之间发送消息了。但是ZeroMQ仅提供非持久性的队列，也就是说如果down机，数据将会丢失。其中，Twitter的Storm中使用ZeroMQ作为数据流的传输。

RabbitMQ: 结合erlang语言本身的并发优势，性能较好，但是不利于做二次开发和维护

ActiveMQ: 历史悠久的开源项目，已经在很多产品中得到应用，实现了JMS1.1规范，可以和spring-jms轻松融合，实现了多种协议，不够轻巧（源代码比RocketMQ多），支持持久化到数据库，对队列数较多的情况支持不好，不过我们的项目中并不会建很多的队列.

Redis: 做为一个基于内存的K-V数据库，其提供了消息订阅的服务，可以当作MQ来使用，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。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。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |

RocketMQ: 阿里巴巴的MQ中间件，在其多个产品下使用，并能够撑住双十一的大流量，他并没有实现JMS规范，使用起来很简单。部署由一个 命名服务（nameserver）和一个代理（broker）组成，nameserver和broker以及producer都支持集群，队列的容量受机器硬盘的限制，队列满后可以支持持久化到硬盘（也可以自己适配代码，将其持久化到NOSQL数据库中），队列满后会影响吞吐量，可以采用主备来保证稳定性，支持回溯消费，可以在broker端进行消息过滤.

针对消息中间件的选择可以从以下方面进行考虑：(主要对比ActiveMQ和RocketMQ)

* 优先级：我们的项目对此需求不是特别明显，RocketMQ需要新建一个特殊队列来接收优先级高的队列，无法实现从0-65535这种细粒度的控制，ActiveMQ可以精细控制
* 顺序：我们的消息总线中的消息应该都是无状态的，所以对消息的处理顺序没有严格的要求，如果有特殊要求的话可以在业务层进行控制，activeMQ无法保证严格的顺序，RocketMQ可以保证严格的消费顺序
* 持久化：都支持
* 稳定性：RoketMQ在稳定性上可能更值得信赖，支持多种集群方案，毕竟已经撑过几个双十一
* 消息过滤：ActiveMQ仅支持在客户端消费的时候进行判断是否是自己需要的消息，RocketMQ可以在broker端进行过滤，对于我们的消息总线，这里可以节省大量的网络传输是否会有消息重发造成的重复消费：RocketMQ可以保证，ActiveMQ无法保证
* 回溯消费：即重新将某一个时刻之前的消息重新消费一遍，我们对于这种需求应该很少，RocketMQ支持，ActiveMQ不支持（RocketMQ的队列是持久化到硬盘的，定期进行清除
* 事务：都支持
* 定时消费：RocketMQ支持
* 消息堆积：就是当缓存消息的内存满了之后的解决方案，一种是丢弃策略，这种不会影响吞吐量，还有一种就是将消息持久化到磁盘，这种会影响吞吐量，在评估影响程度上，RocketMQ的成绩稍微好一点
* 客户端不在线：RocketMQ可以在客户端上线后继续将未消费的消息推送到客户端

1. MQ产品比较

目前比较活跃的几种MQ中间件产品的对比如下：（仅统计开源的项目）

|  | **ActiveMQ** | **RabbitMQ** | **RocketMq** | **ZeroMQ** |
| --- | --- | --- | --- | --- |
| 关注度 | 高 | 高 | 中 | 中 |
| 成熟度 | 成熟 | 成熟 | 比较成熟 | 不成熟 |
| 所属社区/公司 | Apache | Mozilla Public License | Alibaba |  |
| 社区活跃度 | 高 | 高 | 中 | 低 |
| 文档 | 多 | 多 | 中 | 中 |
| 特点 | 功能齐全，被大量开源项目使用 | 由于Erlang 语言的并发能力，性能很好 | 各个环节分布式扩展设计，主从 HA；支持上万个队列；多种消费模式；性能很好 | 低延时，高性能，最高 43万条消息每秒 |
| 授权方式 | 开源 | 开源 | 开源 | 开源 |
| 开发语言 | Java | Erlang | Java | C |
| 支持的协议 | OpenWire、STOMP、REST、XMPP、AMQP | AMQP | 自己定义的一套(社区提供JMS–不成熟) | TCP、UDP |
| 客户端支持语言 | Java、C、C++、Python、PHP、Perl、.net 等 | Java、C、C++、Python、 PHP、Perl 等 | Java C++（不成熟） | python、 java、 php、.net 等 |
| 持久化 | 内存、文件、数据库 | 内存、文件 | 磁盘文件 | 在消息发送端保存 |
| 事务 | 支持 | 支持 | 支持 | 不支持 |
| 集群 | 支持 | 支持 | 支持 | 不支持 |
| 负载均衡 | 支持 | 支持 | 支持 | 不支持 |
| 管理界面 | 一般 | 好 | 好 | 无 |
| 部署方式 | 独立、嵌入 | 独立 | 独立 | 独立 |
| 评价 | 优点：成熟的产品，已经在很多公司得到应用（非大规模场景）。有较多的文档。各种协议支持较好，有多重语言的成熟的客户端；缺点：根据其他用户反馈，会出莫名其妙的问题，切会丢失消息。 其重心放到activemq6.0 产品—apollo 上去了，目前社区不活跃，且对 5.x 维护较少；Activemq 不适合用于上千个队列的应用场景 | 优点： 由于erlang语言的特性，mq 性能较好；管理界面较丰富，在互联网公司也有较大规模的应用；支持amqp系诶，有多中语言且支持 amqp 的客户端可用缺点：erlang语言难度较大。集群不支持动态扩展。 | 优点：模型简单，接口易用（JMS 的接口很多场合并不太实用）。在阿里大规模应用。目前支付宝中的余额宝等新兴产品均使用rocketmq。集群规模大概在50 台左右，单日处理消息上百亿；性能非常好，可以大量堆积消息在broker 中；支持多种消费，包括集群消费、广播消费等。开发度较活跃，版本更新很快。缺点：没有在 mq 核心中去实现JMS 等接口 |  |

1. ActiveMQ 性能测试

测试环境：

  
 硬盘：1T,5400

测试结果：

持久：

插入200000条JSON，共消耗：25.175 s

平均：7944.389275074478 条/秒

插入200000条JSON，共消耗：34.47 s

平均：5802.146794313896 条/秒

插入200000条JSON，共消耗：29.937 s数量：1400000

平均：6680.696128536593 条/秒

插入200000条JSON，共消耗：29.094 s

平均：6874.269608854059 条/秒

非持久：

插入200000条JSON，共消耗：11.35 s数量：1800000

平均：17621.14537444934 条/秒

插入200000条JSON，共消耗：10.714 s

平均：18667.16445771887 条/秒

插入200000条JSON，共消耗：11.153 s

平均：17932.394871335066 条/秒

插入200000条JSON，共消耗：10.717 s数量：2400000

平均：18661.93897545955 条/秒

log的默认大小是32MB,当超过之后会新建一个新的log文件，完成操作后，activeMQ又将旧的log删除了。

以上测试实在PC机上进行，如果换做服务器，性能应该会更好。