计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(6):83−86 [doi: 10.15888/j.cnki.csa.006381]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 RocketMQ 的 MQTT 消息推送服务器分布式部署方案①

马　跃, 颜睿陽, 孙建伟

(中国科学院大学, 北京 100049)
(中国科学院 沈阳计算技术研究所, 沈阳 110168)
通讯作者: 颜睿陽, E-mail: syydf@sina.com

摘　要: 伴随着互联网的飞速发展, 特别是在近几年中, 移动互联网的发展更为迅猛. 在移动互联网中, 消息推送是其中很重要的一部分, 它是手机客户端信息发布和通信的重要方式. MQTT 协议是 Android 系统中消息推送的实现技术之一, 由于其具有低功耗、节省流量和可扩展性强的优点, 目前已得到了众多应用. 同时, RocketMQ 作为一种分布式消息队列, 在服务器分布式部署上具有很大优势, 具有高性能、高可靠、高实时、分布式特点. 本文介绍了 MQTT 协议与 RocketMQ 的这种开源项目的应用, 并通过 RocketMQ 与 Mosquitto 相结合的方式, 实现了一种基于 RocketMQ 的 MQTT 消息推送服务器及其分布式部署.

关键词: MQTT; RocketMQ; 消息推送服务器

引用格式: 马跃,颜睿陽,孙建伟.基于 RocketMQ 的 MQTT 消息推送服务器分布式部署方案.计算机系统应用,2018,27(6):83–86. http://www.c-s-a.org.cn/1003-3254/6381.html

Distributed Deployment Scheme of MQTT Message Push Server Based on RocketMQ

MA Yue, YAN Rui-Yang, SUN Jian-Wei

(University of Chinese Academy of Sciences, Beijing 100049, China)
(Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)

Abstract: With the rapid development of Internet, especially in recent years, the development of mobile Internet is increasingly rapid. Message push, an important way for mobile client to publish information and make communication, plays a significant role in mobile Internet. MQTT protocol is one of the implementation technologies of message push in Android system with low power consumption and high scalability. In addition, it can save Internet traffic, so it has been used in many applications. Meanwhile, as a distributed message queue, RocketMQ has great advantages in distributed deployment of servers. It has the characteristics of high performance, high reliability, and high real time ability. This paper introduces the MQTT protocol and the application of RocketMQ, and implements an MQTT protocol message push server based on RocketMQ through the combination of RocketMQ and Mosquitto and its distributed deployment.

Key words: MQTT; RocketMQ; message push server

随着智能终端普及率的提高, 大部分网民已经从传统的 PC 接入网络变为移动终端接入, 即时通信的发展前景一片光明. 然而基于私有通信协议开发的即时通信类应用, 严重影响了扩展网络功能和技术的进步的步伐, 因此急需一种基于标准协议并能满足大众业务需求的消息推送平台. MQTT 协议是 Android 系统中消息推送的实现技术之一, 由于此协议的简单、便捷, 目前已得到了众多应用. 但传统 MQTT 服务器在分布式部署方面存在较大不足, 基于 RocketMQ 具有高性能、高可靠、高实时、分布式的特点[1], 本文提出

① 收稿时间: 2017-09-30; 修改时间: 2017-11-01; 采用时间: 2017-11-02; csa 在线出版时间: 2018-05-28

了一种基于 RocketMQ 的 MQTT 消息推送服务器并实现了分布式部署.

本文主要分为 4 个部分, 第 1 部分对 RocketMQ 进行了概述. 第 2 部分设计并实现了基于 RocketMQ 的 MQTT 消息推送服务器. 第 3 部分介绍了服务器的分布式部署情况. 第 4 部分对本文设计的消息推送服务器进行了测试和分析. 第 5 部分对本文进行了总结以及对未来工作进行了概述.

1 RocketMQ 介绍

1.1 RocketMQ 介绍

RocketMQ 是阿里巴巴消息中间团队开发的一种基于队列模型的消息中间件, 具有高性能、高可靠、高实时、分布式特点[1], 其中 Producer、Consumer 以及队列都可以实现分布式. 发送消息时, Producer 以此发送消息到各个队列, 这个队列集合称为 Topic. 一个 Consumer 实例消费一个 Topic 对应的所有队列的情况称为广播消费, 多个 Consumer 实例平均消费一个 topic 对应的队列集合的情况称为集群消费[2]. 其通信组件使用了 Netty-4.0.9.Final, 并在此之上进行了简单的协议封装. 其消息格式如表 1 所示.

表 1 RocketMQ 消息格式

Length	Header length	Header data	Body data

其中 length 为 4 字节整数, 其数值等于后 3 部分长度的和. Header length 为 4 字节整数, 其数值等于 header data 的长度; Header data 部分使用 json 序列化数据; Body data 使用阿里自定义的二进制序列化数据.

2 基于 RocketMQ 的消息推送服务器设计

2.1 系统结构设计

本文所设计的消息推送服务器结构如图 1 所示. 整个服务器分为 3 层. 第一层为 MQTT Broker, 主要负责将客户端与服务器进行连接, 并且基于 pub/sub 机制, 实现 MQTT 协议的相关功能; 第二层为协议转换层, 主要将 MQTT 消息包转换为 RocketMQ 支持的消息格式, 其负责实现 MQTT Broker 与 RocketMQ 的对接, 将消息推送到 RocketMQ 的消息队列中以及将消息推送到 MQTT Broker 端; 第三层为 RocketMQ broker, 主要负责将第二层发送过来的消息发送到总线当中, 与分布式服务器进行通讯, 完成消息的分发与接收.

2.2 服务器功能

本文服务器的功能主要是实现消息推送与 pub/sub 功能, 并且在该功能的基础上, 实现用户个体之间的即时通信、预订阅等相关功能. 即时通信可以传输文本、图片、语音、视频和文件等消息, 预订阅用于客户端第一次连接到代理服务器时, 帮助用户预先订阅话题, 从而达到节省流量, 优化用户体验的目的.

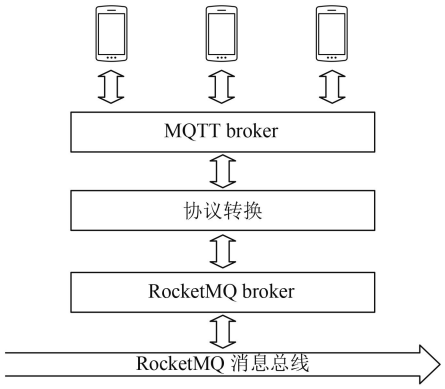

图 1 服务器结构

2.3 消息推送模块的实现

现如今, 比较主流的 MQTT 协议代理有 IBM Websphere MQ Telemetry 和 Mosquitto[3]. 本文选取了 Mosquitto 作为 MQTT 的消息代理. Mosquitto 是一款轻量级开源的消息代理, 它相对完整的实现了 MQTT 协议中的各项功能, 并能够完美运行在 Linux 系统中. 所以, 本文选取并在其开源项目的基础上, 对其进行了优化, 使其能够实现多终端同步.

通过对比 MQTT 协议的格式以及 RocketMQ 消息的格式, 不难发现两者都是基于 Topic 模块, 而且两者参数存在很多相同. 所以, 我们通过协议转换模块, 将 mosquitto 接收到的 Topic、QoS 及 payload 等相关参数, 转换成 RocketMQ 的 Topic、QoS 及 Body 等参数, 并经由 producer 发送至总线中, 由 Consumer 负责消费. 流程图如图 2 所示.

消息推送模块完成消息发布到被消费的具体流程:

1) 当客户端发布消息时, 由轻量级 Mosquitto 消息代理将消息接受并且返回给客户端消息送达状态.

2) 将 MQTT 消息包内的 Topic、payload 以及 Qos 等相关信息进行提取, Topic 在原有基础上增加_rocketmq 组成新的 Topic 传递给 rocketmq; 将 payload 格式由 mqtt_string 转换为 string, 并赋值给 Body 传递给 rocketmq; 将 MQTT 的 QoS 参数直接传递给 Rocketmq 的 QoS 参数. 组成新的数据包之后, 经由 RocketMQ broker 将消息推送到消息总线中, 并由总线返回消息送达状态.

3) RocketMQ 的 Consumer 根据 Topic 对总线中的

消息进行消费, 并将 RocketMQ 的 Topic 直接传递给 MQTT 的 Topic, 即 topic_rocketmq. 并将 Body 和 QoS 重新按照 MQTT 格式进行封装, 并且推送给订阅了该主题的客户端.

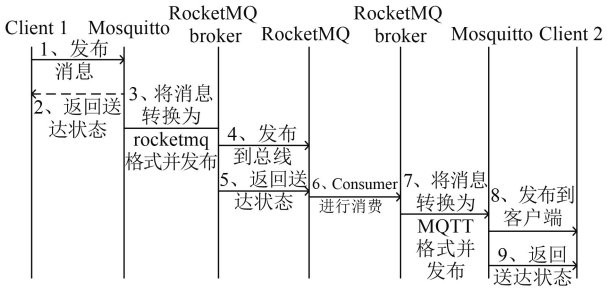

图 2 消息流程图

Broker 可分为 Master 与 Slave. 通过指定相同的 BrokerName, 不同的 BrokerId 来定义 Master 与 Slave 的对应关系. 想要一个代理成为 Master, 设置 BrokerId 为 0, 设置 BrokerId 的值非 0 则为 Slave. Master 可以部署多个. 每个 Broker 与 NameServer 集群中的所有节点建立长连接, 定时向所有的 NameServer 注册 Topic 信息[3].

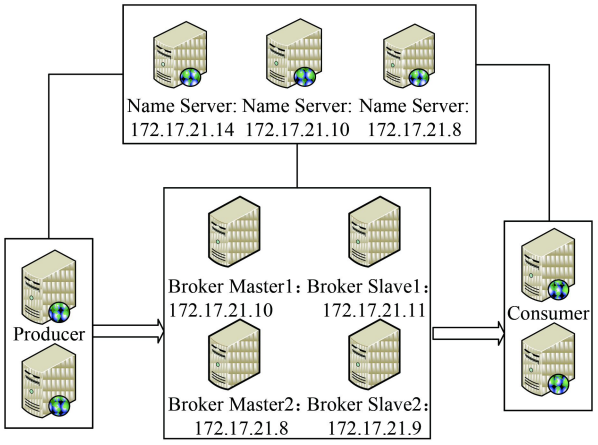

图 3 分布式部署结构图

3.3 服务器配置

Host 文件配置:

vim/etc/hosts

172.17.21.8 mqnameserver1

172.17.21.10 mqnameserver2

172.17.21.14 mqnameserver3

172.17.21.8 rocketmq-master2

172.17.21.9 rocketmq-slave2

172.17.21.10 rocketmq-master1

172.17.21.11 rocketmq-slave1

由于篇幅有限, 本文只举例说明 Master1 服务器配置, 其余配置基本相同.

vim/usr/local/rocketmq/conf/2m-noslave/broker-1.properties

brokerName=broker-1

brokerId 为 0 为 Master, 非零为 Slave

brokerId=0

namesrvAddr=mqnameserver1:9876;mqnameserver2:9876;mqnameserver3:9876

storePathRootDir=/data/rocketmq/store

storePathCommitLog=/data/rocketmq/store/commitlog

sendMessageThreadPoolNums=128

pullMessageThreadPoolNums=128

3 分布式部署结构

3.1 服务器角色分配

对服务器的架构角色分配如表 2 所示.

3.2 服务器物理结构

图 3 中, 由于 Name Server 是一个几乎无状态节点, 故部署为集群, 节点之间无任何消息同步[4]. Producer 与 Consumer 完全无状态, 集群部署, Producer 选择 Name Server 集群中的一个随机节点建立长连接[5]. Consumer 基本与 Producer 相同, 只是其要想提供 Topic 服务的 Master、Slave 建立长连接并定时发送心跳. 其即可以从 Master 订阅消息, 也可以从 Slave 订阅消息.

表 2 服务器架构角色

IP 地址	主机名	操作系统	角色	架构模式
172.17.21.8	rocketmq-master2	CentOS 6.5	nameserver、brokerserver	Master2
172.17.21.9	rocketmq-slave2	CentOS 6.5	brokerserver	Slave2
172.17.21.10	rocketmq-master1	Ubuntu 16.04	nameserver、brokerserver	Master1
172.17.21.11	rocketmq-slave1	Ubuntu 16.04	brokerserver	Slave1
172.17.21.14	rocketmq-nameserver	CentOS 6.5	nameserver	

4 消息服务器测试

4.1 单 Broker 测试

本部分将对本文所设计的服务器在增加负载的情况下进行性能和鲁棒性的测试. 服务器配置如表 3 所示.

表 3 服务器配置

	参数
CPU	Intel corl i5 2.3 GHz
内存	64 GB
存储	1 T
OS	CentOS 6.5
网卡	1000 Mbps

本文通过使用 Erlang 语言开发的, 开源测试工具 emqtt_benchmark 模拟客户端登录并发送消息对服务器进行负载压力测试. 测试网络结构图如图 4 所示. 其中, 黑色连接线代表网络连接, 箭头表示数据流向.

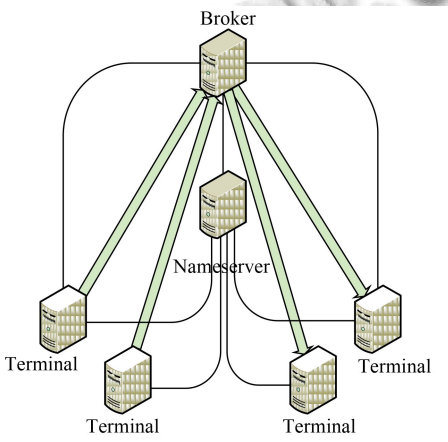

图 4 测试网络结构

两台 terminal 模拟用户起大量线程, 不间断的向 broker 发送大小为 2 K 的消息, 测试结果如图 5 所示.

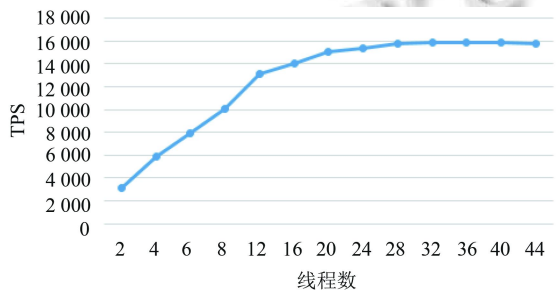

图 5 TPS 测试结果

通过测试, 单机 Broker 的 TPS 可达到 15 900 左右, 继续增大消息发送量时, 可以看到 TPS 有下降的趋势. 通过测试可以观测服务器在大量消息情况下依然能够保持一个稳定的状态, 基本能够满足日常服务器需求.

4.2 双 Master、双 Slave 测试

接下来模拟大量用户进行连接, 并在连接完成后进行消息收发, 测试其 CPU 占有率及消息全部送达所需时间. 测试结果如表 4 所示.

表 4 服务器性能测试结果

Connections	CPU usage (%)	1~100 push (s)	1~1000 push (s)	All push (s)
10000	14.61	0.093	0.112	11.961
20000	25.45	0.105	0.231	27.235
30000	33.47	0.279	0.434	43.601
40000	41.12	0.394	0.673	66.812
50000	46.08	0.485	0.869	109.398
75000	55.21	0.676	1.106	208.182
100000	63.50	0.843	1.431	315.873

由表中数据可以得出, 本文所设计的消息推送服务器, 响应速度基本满足当前移动互联网领域中的需求, 服务器负载也在可接受范围内, 连接并发数量尚可, 服务器性能稳定.

5 总结与展望

本文在研究 MQTT 协议以及 RocketMQ 的基础上, 论述了基于 RocketMQ 的 MQTT 消息推送服务器的设计, 并对其进行了实现. 本文重点阐述了消息推送服务器的消息接收和发送、消息格式转换功能、消息推送服务器的性能测试, 同时与客户端结合完成了消息推送服务器的功能测试. 最后根据测试结果可以看出, 本文设计的服务器完成了消息收发、协议转换等功能要求, 同时具有良好的抗压能力和鲁棒性.

下一步的工作主要集中在提高服务器的并发处理能力上, 同时进一步提高服务器的性能及鲁棒性.

参考文献

1 架构说: 阿里中间件技术: 消息中间件篇. http://www.jiagoushuo.com/article/1000141.html. [2016-04-18].

2 Apache RocketMQ: An open source distributed messaging and streaming data platform. http://rocketmq.apache.org. [2016-12-29].

3 Mosquitto. An open source MQTT v3.1/v3.1.1 broker. http://mosquitto.org/. [2015].

4 Kobejayandy. RocketMQ 入门. http://blog.csdn.net/kobejayandy/article/details/52831213. [2016-10-16].

5 欧志芳. 基于 RocketMQ 实现异构数据库同步. 网络安全技术与应用, 2016, (12): 99–100. [doi: 10.3969/j.issn.1009-6833.2016.12.066]