WebSocket 与 MQTT 在 Web 即时通信系统中的应用①

刘 峰[1], 陈 朴[1,2], 贾军营[1]

[1](中国科学院 沈阳计算技术研究所, 沈阳 110168)
[2](中国科学院大学, 北京 100049)

摘 要: 传统 Web 通信系统多采用轮询拉取方式, 此种方式存在实时性低、网络资源消耗大、扩展性差等缺点. 针对上述问题, 本文研究了 WebSocket 中的长连接技术, 结合 MQTT 协议, 提出了基于 pub/sub 模型的 Web 端即时通信解决方案, 以推送取代传统拉取方式. 文中重点阐述了 IM 与通知类消息格式设计, 并针对通知类消息提出了 agent 代理模型, 同时为了进一步减少网络资源消耗, 提出了预订阅模式. 最后将该解决方案与现流行的 bosh+xmpp 方式在时延、带宽消耗方面做了对比分析, 以验证该方案的性能优势.

关键词: B/S; 即时通信; WebSocket; MQTT; Pub/Sub

Application of WebSocket and MQTT in Web Real-Time Communication System

LIU Feng[1], CHEN Pu[1,2], JIA Jun-Ying[1]

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

Abstract: Polling pull strategy, which is widely used in traditional web communication system, has some problems, such as low real-time, heavy consumption of network resource and poor scalability. To solve these problems, much research about long connection technology in WebSocket has been done in the paper. Combining WebSocket with MQTT protocol, a new web instant messaging solution based on pub/sub model was proposed to replace polling strategy with push strategy. The formation design of IM and notification message was described in detail. And agent model was proposed for notification message. Meanwhile, a pre-subscription scheme was put forward to decrease net resource consumption further. Moreover, a comparative analysis on delay, bandwidth consumption between the solution and a current popular way bosh+xmpp was done to verify the advantages of the solution.

Key words: B/S; real-time communication; WebSocket; MQTT; Pub/Sub

随着 HTML5 规范制定完成, Web 应用迎来了新的发展机遇[1]. Web 通信系统以其跨平台、易于其他业务系统融合的特点, 将进一步受到重视.

传统 B/S 通信系统建立在 HTTP 协议的基础上, HTTP 设计之初的目的是提供一种发布和接收 HTML 页面的方法, 其严格遵循“请求-应答”模型. 在网络资源有限的情况下, 这种设定有利于节省带宽. 然而实时通信系统需要及时获得最新消息, 因此浏览器必须频繁地发起询问请求. 尽管开发者在此约束下提出了大量优化方案, 例如轮询、长轮询、流技术等, 但是其始终建立在频繁访问的基础上, 存在消息及时性差、网络资源消耗大等诸多问题.

不同于传统 B/S 架构通信系统, 本文采用 WebSocket 作为数据交换通道, 结合轻量型 MQTT 协议, 设计了一款发布/订阅模型下的 Web 即时通信系统. 在保留传统 B/S 架构通信系统易部署、易升级、易与其他业务系统融合等特性的同时[2], 增强了 Web 通信系统的实时性、扩展性, 改善了服务器和网络资源的利用率. 文中实现了多种媒体方式的点对点、群组 IM, 以及通知类消息推送功能, 重点描述了 Topic、消息 payload 格式设计, 同时提出了 agent 代理模型和预订阅方案, 并在流量消耗方面与当下流行的 bosh+xmpp

① 收稿时间:2015-08-18;收到修改稿时间:2015-11-02

方式作出比较, 以验证此方案的可行性和性能优势.

1 协议介绍

1.1 WebSocket 概述

WebSocket 是 HTML5 的新协议, 通过 WebSocket 浏览器和服务器之间可以建立起长连接, 并在其基础上自由的发送数据, 实现真正的实时通信. 相比以前轮询等方式, 此方式能够很大程度上增强消息的及时性. HTML5 提供了大量的 WebSocketAPI, 越来越多的浏览器厂商在其新产品中开始支持 WebSocket[3]. 此规范的出现, 为 Web 应用提供了无限的可能.

```
GET /example HTTP/1.1
Host: 127.0.0.1:9001
Upgrade: websocket
Connection: Keepalive,Upgrade
Sec-WebSocket-Key: +nEpO6skTwqlTwmmVnciJA==
Sec-WebSocket-Protocol: mqtt
Sec-WebSocket-Version: 13
Origin: http://127.0.0.1:8080

HTTP/1.1 101 Switching Protocols
Connection:Upgrade
Sec-WebSocket-Accept:YtlOS3MKwO6HDl9Gyd+Lf4
Sec-WebSocket-Protocol: mqtt
```

典型的 WebSocket 连接请求和响应如上所示. WebSocket 协议本质上还是基于 TCP 的协议. 为建立 WebSocket 连接, 客户端向服务器发送 http 请求, 这个请求不同于普通的 http 请求, 其包含了“Upgrade: websocket”字段, 表示申请协议升级. 之所以使用 HTTP 发起请求, 是希望与 HTTP 使用同样的 80(非安全)和 443(安全)端口. 服务器端接收到此请求后, 如果其能够识别 WebSocket 协议便给与同意建立连接的响应信息. 其中的 Sec-WebSocket-Accept 由连接请求中的 Sec-WebSocket-Key 计算而来. 客户端得到服务器端响应信息 ACK, 验证 Sec-WebSocket-Accept, 如果正确, 表明服务器支持 WebSocket, 两者即可建立起全双工通信[4].

1.2 MQTT 与 mosquitto

在 1.1 节中, 本文说到了 WebSocket 解决了浏览器与服务器间的全双工通信, 但是为了实现即时通信, 简单的数据传输是不够的, 对消息的控制是更加重要的环节. MQTT(Message Queuing Telemetry Transport 消息队列遥测传输)协议由 IBM 于 1999 年开发, 其是一款基于发布/订阅的轻量级消息传输协议. ?2013 年 3 月 OASIS Standard 宣布 MQTT 成为物联网消息传递首选协议[5]. MQTT 协议以其能耗低、轻量、能够有效的进行消息分发等特性, 被国内许多互联网企业用在移动端以实现消息推送[6].

一个 MQTT 消息由三部分构成, 即固定头部, 可变头部和有效的数据载荷. 固定头部是每个 MQTT 消息必须包含的部分. 可变头部根据不同消息类型提供相关标识数据. 有效载荷为消息体部分. 具体如表 1 所示.

表 1 MQTT 消息格式

	7	6	5	4	3	2	1	0
byte0	消息类型				UDP flag	Qos		retain
byte1	remaining length							
bytek	remaining length(if have)							
	payload							

固定头部中第 4 到第 7bit 为消息类型, 其中 0 和 15 保留, 共 14 中消息类型. 丰富的消息类型使得 MQTT 协议拥有非常好的可控性. 此外 MQTT 协议为二进制协议, 其轻量的特点能够较好的减少网络资源消耗. 同时, 其提供多种质量的消息服务, 例如至少一次, 至多一次, 仅有一次等, 使其可以应用在不同的环境中.

MQTT 是基于 Pub/Sub 的协议, 其消息发布的基本流程如图 1 所示. 消息的发布者不需要关注订阅者是谁、有多少订阅者, 其仅需要将想要发布的消息发布到一个主题上, 订阅此主题的所有用户便可收到此消息. Broker 为服务器端代理, 其负责维护所有的订阅关系和消息的转发.

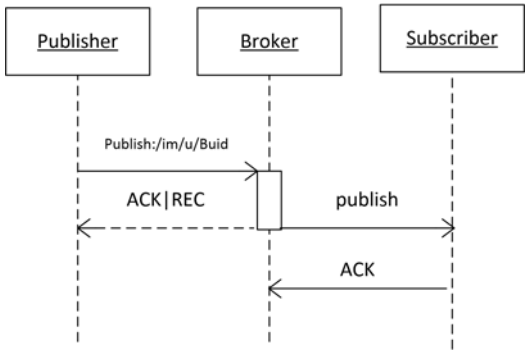

图 1 MQTT 消息发布基本流程

MQTT 协议从设计之初到目前的广泛使用，许多的组织和个人，开发出大量相关项目和工具类，以支持 MQTT 的使用，包括客户端和代理服务端. mosquitto 为其中较为人熟知服务器代理，其采用 C 语言编写，性能优越、单个 mosquitto 节点可并发处理近 100000 个连接. 同时其代码开源，众多开源机构不断对其升级维护. 其订阅树的管理办法，思路清晰，易扩展[7]. 从 1.4 版本开始对 WebSocket 支持，这也是本文采用此代理的一个重要原因.

2　系统框架设计

整个系统框架可分为两部分，一部分为浏览器端，另一部分为服务器端. 浏览器端为连接请求的发起端，服务器端负责数据的维护，以及话题的订阅和消息的推送.

服务器端主要由六部分组成. WebServer 负责维护 HTML 页面和相关 Js 资源，浏览器发起请求页面后返回. mosquitto 代理负责维护所有的话题、订阅关系、以及消息的分发. Elgg 是一个开源的社交网络引擎，其拥有简单灵活的数据模型，内置了 webservice 框架[8]，与其他接口共同维护此系统的数据资源. Agent 模块负责将对数据库的操作封装成 MQTT 消息，并发布到对应的话题，以实现通知类消息推送. pre_sub 模块负责为新登录用户预订阅话题. MessageQueue 维护一个消息队列，由 webservice 对数据库的操作事件均发布到此消息队列上，agent 和 pre_sub 模块监听此消息队列，不断从中获取消息并作出处理. 为当面临较高并发要求时，Elgg 和 mosquitto 均可采用集群的方式扩展[9]. 系统框架如图 2 所示.

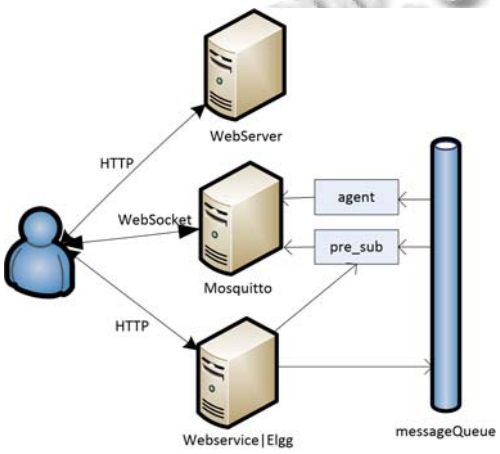

图 2　系统总体框图

3　系统核心模块实现

3.1 话题设计

MQTT 是基于 Sub/Pub 模型的消息传输协议，作为此模型的核心内容 Topic，其设计对整个系统的功能、性能以及扩展性起到非常重要的作用. 优秀的话题设计能够以最小的数据发送量实现预期目标，并能以最小的改动对现有功能扩展.

即时通信系统中，涉及到的会话主要为点对点、群组 IM(Instant Messaging)，其为系统成员之间最基本的沟通方式. 此外系统成员的数据维护同样是很重要的内容，例如成员数据、成员在线状态更新，群成员加入离开等，数据的变化需要及时通知终端用户. 本文将此类型的消息称为通知类消息. 需针对此需求定义话题.

MQTT 中话题采用分层结构，/表示层次. 话题设计如图 3 所示.

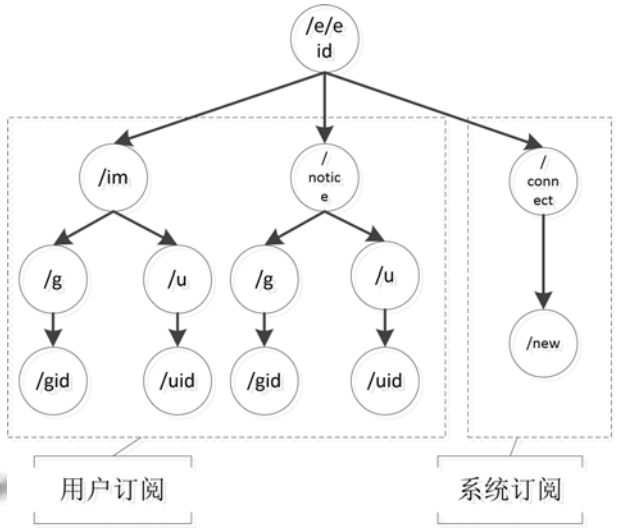

图 3　话题设计树形图

本文针对仅描述的 IM、通知以及部分系统功能定义话题. 添加新功能时，可类比此对话题扩展. 话题分为用户订阅话题和系统订阅话题. 用户订阅话题为实现某些客户端功能定义的话题，例如本文涉及到的 IM 和通知. IM 类话题具体为 /e/eid/im/g/gid 和 /e/eid/im/u/uid. 其中第一和第二层次为当前组织的 id. 本系统可同时为多个组织提供服务，eid 为组织唯一标识符、其作为特定组织订阅树的最上层节点. 系统订阅话题为服务端不同模块间交互的桥梁.

3.2 Payload 格式设计

MQTT 消息并不是针对 IM 设计的消息协议，为

使其能够满足 IM 环境, 具有较好扩展性的 payload 设计是非常重要的内容. 本文针对 IM 和通知类消息 payload 格式设计做出说明. 对于一条推送消息, 需要的基本信息包括消息的类型, 消息发送时间、消息内容, 消息发送者. 针对一些特殊的需求, 还需要携带更多的信息.

3.2.1 IM payload 格式设计

IM 作为即时通信中最重要的部分, 优秀的消息格式能够给用户带来丰富的功能体验. 本文的 IM 消息体 payload 格式设计如表 2 所示.

表 2 IM payload 格式设计

name	ts	type	type1	c-l-1	con1
byte	0-3	4	5	6-9	n1+9
name	type2	.....	sen-l	sen	
byte	N1+10	.....	N+1	N+n+1	

0-3 字节标识消息发送的时间, 第 4 字节标识消息类型, 高位 7-4bits 标识 IM 类型. 例如 0x1 表示 WEB 到 IOS 客户端的 IM 等. 低位 3-0bits 标识同时可携媒体的数量, 最多可携带 7 个媒体.

此 payload 格式设计可支持一个 IM 中同时携带多种媒体. 其中 typek 占一个字节, 标识第 k 个媒体类型, 例如 0x00 文本, 0x01 图片, 0x02 音频, 0x03 视频, 0x04word, 0x05ppt 等. c-l-k 标识第 k 个媒体长度, conk 为第 k 个媒体内容. 最后的内容为消息的发送者标识长度, 以及标识内容. Content 消息的内容, 结构为 Json 字符串, 具体的格式如下:

```
{
   “co”:”url 或者文本”   //媒体内容
   “fn”:”show.ppt”       //媒体文件名, 如果不是文
                           件, 则没有该字段
   “ml”:”500”            //媒体大小
 }
```

对于文件, 因其长度较大, 本文不选择把其放在 IM 消息中. 在 IM 消息中仅给出此文件的 url 以及文件名称和大小. 此处的 url 指向媒体服务器资源. 在用户发送文件类型的 IM 消息时, 会将文件上传至媒体服务器. 收到此类型 IM 的用户, 根据此 url 从 MediaServer 下载资源, 真正拿到消息内容.

3.2.2 通知类型消息 payload 格式设计

通知类消息负责维护系统成员基本信息, 例如成员在线状态更新、群信息变化、通讯录变化等. 主要包括个人通知和群组通知. 具体通知类消息的发送流程见 3.2 节中所述. 通知类消息 payload 格式设计如表 3 所示.

表 3 通知类消息 payload 格式设计

name	ts	type	content
byte	0-3	4	5+n

与 IM 消息格式一样, payload 中, 0-3 字节为通知发送的时间, 第 4 字节为通知类型, content 为 json 数据, 包含对应类型通知的基本数据. 终端收到通知后, 可根据通知类型, 以及 content 中 json 字段进行相关操作. 表 4 中描述了部分通知类消息 payload 设计.

表 4 部分通知类消息及其 content

Value	Content
0x28 被迫下线, 同种终端上线	无该字段
0x30 通讯录变化	Json: 发生变化的用户 uid
0x31 成员在线信息变化	Json: 发生变化的用户 uid, 最新的用户在线状态码 0x00 PC 在线、0x01 手机在线、0x02 Web 在线
0x37 群成员加入	Json: 加入群组的 id, 加入者用户 id

3.3 Agent 代理模块

在前面章节中本文提到了通知类消息, 用户通过 webservice 等方式更改相关数据, 例如用户昵称头像、群组基本信息等, 这些数据的变化需要转变为通知类 MQTT 消息进而推送给终端, 这就是 Agent 代理要完成的任务.

在服务器启动之后, agent 代理模块建立与 mosquitto 的连接. 当用户通过 webservice 请求对相关数据作出改变时, webservice 操作数据库的同时, 将该请求的最基本信息放入消息队列. 之所以放入最基本信息, 是防止消息队列过于庞大而影响性能. 此请求可以是加入群组, 联系人变化, 成员在线状态变化等. Agent 不断的从消息队列中获取消息, 根据消息信息, 从数据库中获得详细信息, 并整合成为 MQTT 消息, 发布到 mosquitto 对应话题上, 进而订阅该话题的用户即可接收到此通知. 具体 agent 工作流程如图 4 所示.

例如群组成员加入时, 用户发出 webservice 请求, webservice 对数据库中群信息作出操改变后, 将消息 {“type”:”g_join”,”gid”:”xx”,”uid”:”xx”}添加到消息队列. 此处以 json 形式表示此消息. Agent 从消息队列中获取该消息后, 根据 gid、uid 从数据库获得群组和加入成员的详细信息, 之后整合为 mqtt 消息发布到

/e/eid/notice/g/gid 话题上, 该群组的其他成员及可收到该成员加入的通知.

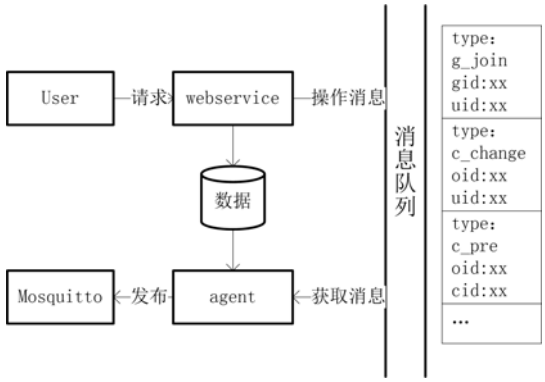

图 4　agent 工作流程

3.4 预订阅模块

话题订阅一般由客户端完成, 然而, 当需订阅的话题众多, 大量的客户端订阅数据将给网络带来巨大压力. 同时为了减轻客户端负担, 使其能集中精力在用户体验上, 本文采用预订阅的方案, 即在服务器端帮助新连接的客户完成相关话题的订阅工作. 预订阅流程如图 5 所示.

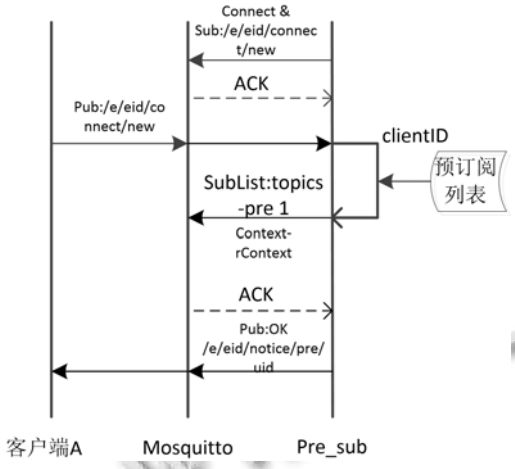

图 5　预订阅时序图

在服务器端, 本文编写 pre_sub 模块, 负责新登录用户预订阅任务. 在服务器端 mosquitto 启动后 pre_sub 建立与其的连接, 并订阅话/e/eid/connect/new. 当客户端 A 第一次登录系统后, agent 代理会发布一条消息到/e/eid/connect/new 话题, 并在消息体中携带此用户的 ClientID. 通过 mosquitto 代理的分发, pre_sub 模块会收到此消息, 其解析消息后, 获得新连接用户 CientID, 以其作为唯一标识, 通过 webservice 去外部服务器获得此用户的预订阅话题列表.

预订阅话题条目设计如下:

<ClientID><aciton><topicCount><topic1><topic1Qos>...<topicN><topicNQos>

其可以 json 数据格式传送.

Pre_sub 模块得到预订阅列表后, 根据预订阅条目向 mosquitto 代理发出订阅请求, 订阅消息中携带新登录用户的 ClientID. mosquitto 代理收到订阅请求后, 解析消息体. 根据消息体中的 clientID 信息, 从当前代理所有连接客户端中查找新登录用户的 context 信息, 以其完成相关话题的订阅. 在完成订阅后, mosquitto 向预订阅模块返回确认信息.

4　功能与性能测试

4.1 功能测试

测试环境为 tomcat+mosquitto+libwebsocket, 其中 libwebsocket 为一个简单的 WebSocket 服务器. mosquitto 在 6767 端口监听 WebSocket 连接请求. 本文以单聊、群聊(文本和文件)和成员加入群通知消息为测试用例. 为了方便测试, 其订阅所有话题#. 同时为了使消息数据更易观察, 以消息最初的数据为显示内容. 测试结果如图 6 所示. 从测试结果可知, 可以根据具体需求对话题和消息格式进行设计, 从而满足多种应用场景, 其拥有较好的扩展性.

图 6　IM 与通知消息测试

4.2 性能测试

相比传统 Web 即时通信方式, 本文在流量和时延方面给与测试, 以验证该方案的优越性. 目前 Web 端即时通信多采用 bosh+xmpp 方案. bosh 为长轮询技术, xmpp 为较成熟基于 xml 的消息控制协议[10]. 经过实践检验, bosh+xmpp 方案明显好于其他轮询方案[11]. 所以本文仅与此方案比较.

WebSocket+mqtt 方案可由服务器主动推送数据到浏览器, 其时延肯定要比轮询好, 这也是 WebSocket

产生的最初动机，所以本文仅对网络流量给与测试，时延不再作为测试重点.

本文根据有无消息发送，测试两种方案下的网络带宽消耗. 无消息发送时，网络消耗在询问和保活数据包上，保活数据发送周期均为 60s. 测试结果如表 5 和表 6 所示. 可以看到，在无消息发送时，bosh 方式发送的保活包数量为 WebSocket 方式的近 2 倍，保活包大小也为其的 2 倍左右，总体的带宽消耗是其 3 倍之多. 发送消息时，xmpp 方式数据包大小为 mqtt 方式的近 6 倍，发送的数据包数量为其 2 倍，整体带宽消耗远远高于 mqtt 方式. 可见，本文提出的方案相比当前流行的 bosh+xmpp 方式可以大大节省 Web 即时通信的网络带宽消耗.

表 5 10mins 内无消息时的网络数据(询问请求和保活)

	Bosh+xmpp	WebSocket+mqtt
Packets	143	84
Bytes	18972	5796
Avg.packets/sec	0.237	0.14
Avg.size/packets	132.671	69

表 6 1min 发送 12 个“hello”消息时网络数据

	Bosh+xmpp	WebSocket+mqtt
Packets	52	28
Bytes	23087	2052
Avg.packets/sec	0.908	0.513
Avg.sizes/packet	443.981	73.286

5 结语

随着 Web 技术的进步，Web 富应用功能越来越强大[12]. 本文针对原有 B/S 架构即时通信系统实时性低、网络资源消耗大、扩展性差等问题，研究了 WebSocket 与 MQTT 协议，设计了一款基于 Pub/Sub 模型的 Web 即时通信系统，以推送取代传统的拉取方式. 完成了扩展性较好的系统设计，实现了点对点、群组 IM 以及通知推送功能，并对 IM 和通知消息格式设计给予了详细说明. agent 代理模式的提出很好的解决了通知类消息推送问题. 为进一步降低浏览器端业务压力和减少网络流量，提出了预订阅方案. 最后的测试结果表明，该方案能够很好的实现即时通信能力，并拥有较好的扩展能力. 同时，相比传统 Web 通信方案，其在提高消息实时性的同时，能够较大程度上减少网络资源消耗.

参考文献

1 Anthes G. HTML5 leads a web revolution. Communications of the ACM. 2012, 55 (7): 16−17.

2 张艺.基于 WebSocket 的即时通信系统的研究与实现.软件,2015,36(3):89−94.

3 韩安.HTML5 WebSocket 技术研究.电子世界,2013,20:5−6.

4 陆晨,冯向阳,苏厚勤.HTML5 WebSocket 握手协议的研究与实现.计算机应用与软件,2015,32(1):128−131.

5 IBM. MQ Telemetry Transport. http://mqtt.org. [2013-06-05].

6 朱艳.移动应用的消息推送与 MQTT 协议.无线互联科技,2015,8:1−3.

7 Mosquitto. Mosquitto Broker. http://mosquitto.org. [2015-06-10].

8 Elgg. https://elgg.org/features.php. [2015-06-10].

9 任亨.基于 MQTT 协议的消息推送集群系统的设计与实现[硕士学位论文].北京:中国科学院研究生院,2014.

10 苟海燕,伦冠民,徐亚波,范少华.XMPP 协议研究.电子制作,2013,11:120.

11 王璐.web 模式下基于 xmpp 的即时通信系统的设计与实现[硕士学位论文].北京:北京邮电大学,2010.

12 陆钢,李慧云.HTML5 技术应用现状与发展趋势研究.广东通信技术,2013.5:1−5.