Android 平台基于 MQTT 协议的推送机制①

许金喜, 张新有

(西南交通大学 信息科学与技术学院, 成都 610031)

摘 要: Android 平台的迅速崛起对消息推送技术提出了更严峻的挑战. 传统的推送通知方式主要有 Polling, SMS Push, IP Push 三种, 但对 Android 平台上的应用来说, 这些推送方式都有各自的短板. 针对该情况, 首先对 Android 平台上几种推送机制进行了分析, 之后着重对基于 MQTT 协议的推送机制进行研究, 最后借助 IBM 提供的开源工具 Mosquitto 通过编码加以实现. 经实验测试, 证明基于 MQTT 协议的推送机制效率高, 功耗低, 可以稳定地用于 Android 平台上的应用.

关键词: Android; 推送; MQTT 协议; QoS; Mosquitto

Push Mechanism on Android Platform Based on MQTT Protocol

XU Jin-Xi, ZHANG Xin-You

(School of Information Science & Technology, Southwest Jiaotong University, Chengdu 610031, China)

Abstract: The rapid development of Android platform brings forward more severe challenge for message push technology. Traditional ways of push notification mainly contain Polling, SMS Push and IP Push . But for applications on Android platform, any of these push technology has its own shortcomings. Firstly, an analysis was conducted on various kinds of push mechanism on Android Platform. Then a push mechanism based on protocol MQTT was researched in detail. Finally it was implemented through coding with the help of open source tool Mosquitto supported by IBM. Experimental results showed that this way has a low power consumption and high efficiency and it could be used for applications steadily on Android platform.

Key words: Android; push; MQTT protocol; QoS; Mosquitto

1 引言

支持多种应用是 Android 平台的精髓所在. 这些应用作为客户端, 要收到来自服务器的消息和通知有两种方式, 一种是 pull(拉), 即客户端主动连接服务器获取信息. 另一种是 push(推), 即由服务器主动将信息推送到客户端, 用户无需主动检查更新, 待收到推送消息后可选择进一步获取详细信息或忽略该消息. 显然, 无论是从电量、流量消耗方面衡量, 还是基于用户体验性方面考虑, push 都要优于 pull[1].

当前 Android 平台的消息推送方式主要有 Polling, SMS Push, IP Push 三种. 本文首先对这三种方式进行了分析, 着重研究了基于 MQTT (Message Queuing Telemetry Transport)协议的推送机制的工作原理以及 QoS 保证机制. 最后通过设计推送系统, 验证了 MQTT 推送方式的优良性能.

2 Android平台推送机制分析

Push(推送)技术又称网播技术, 最早由 Point Cast Network 公司于 1996 年提出, 其目的是为了提高用户获取信息的针对性和时效性[2]. 根据各种推送机制的实现原理, 当前推送机制大体可分为 Polling、SMS Push、IP Push 三种.

2.1 Polling

Polling 即轮询, 该机制其实是一种“伪 push”, 其本质是使用传统的 pull 技术, 通过定时向服务器发送请求检查更新来实现 push 效果. 这种方式简单灵活,

① 收稿时间:2014-04-24;收到修改稿时间:2014-05-26

© 中国科学院软件研究所 http://www.c-s-a.org.cn

可由用户决定更新策略. 但由于周期性地连接网络, 对电量和流量消耗比较大, 且如果服务端无更新, 客户端就做了无用功. 该种机制只适用于一些实时性要求不是很高的应用, 如新闻、天气等.

2.2 SMS Push

SMS(Short Messaging Service)即短信, 该机制本质上是利用移动运营商的服务来实现推送. 由于移动基站的便利性, 只要手机信号通畅且未欠费即可使用此服务. 但该机制收费较高, 且推送的消息种类和大小都有局限性.

基于 SMS Push 原理的还有一种推送机制, 即通过运营商转发进行推送, 其代表为 RIM 公司的 Pushmail 服务. RIM 公司作为邮件服务商和运营商合作, 在操作系统层面封装推送服务, 使持有 Blackberry 手机的用户收到由运营商转发的邮件[3]. 该服务曾在数年前风靡全球, 但费用高昂, 现在使用人数减少了很多.

2.3 IP Push

IP Push是当前大部分应用中最为常见的推送机制. 其本质为拥有固定IP地址的客户端通过heartbeat的方式与服务器保持一个持久的长连接[4].

2.3.1 GCM

Google 公司的 GCM(Google Cloud Messaging)从操作系统层面封装推送服务, 保持与客户端的长连接. 该服务提供一个统一的接口, 允许开发者从应用服务器向客户端发送数据[5]. 但该服务需要客户端 Android 版本高于 2.2, 且低于 4.0 的版本需要注册 google 账号才能使用该服务. 最为严重的是, 由于网络原因, 该服务在国内很不稳定.

针对 GCM 的弊端, 一种替代方式为选择第三方推送平台, 如国内的极光推送、个推等, 开发者下载专门的 SDK 集成到自己的应用中接收推送通知. 区别于 GCM, 这些方式是在应用层提供推送服务. 该领域目前处于起步阶段, 基础服务暂时免费. 对于实力相对薄弱的中小型公司, 该方式是不错的选择.

2.3.2 自主推送平台

在实际开发中采用第三方推送平台很多信息也必然被暴露. 若要突破此限制, 只有自主搭建推送平台, 目前实现推送通知最主流的两种协议是 XMPP 和 MQTT. XMPP 协议成熟、强大, 已由国际标准化组织完成了标准化工作. 但该协议比较复杂, 部署成本较高, 且基于 XML, 冗余很大[6]. MQTT 协议是基于发布/订阅消息模式的轻量级协议, 可扩展性强、开销小, 且传输消息时对负载内容屏蔽, 极适用于移动平台的消息推送.

经过上文对各种推送机制的详细分析, 可绘制出表 1, 该表从收费情况, 电量、流量消耗, 速度, 安全性, 复杂度六个角度对以上几种推送机制进行了比较. 由表 1 可看出, 在各种免费的推送机制中, 要满足功耗低, 效率高的要求, 基于 MQTT 协议的推送机制是目前最佳的选择.

表 1 各种推送机制的性能比较

推送机制		收费	电量	流量	速度	安全性	复杂度
Polling		免费	较高	较高	慢	极高	极低
SMS Push		收费	极低	极低	快	高	极低
IP Push	GCM	免费	较低	低	较快	高	较低
	第三方	暂免	高	低	较快	低	较低
	自主开发 XMPP	免费	高	低	较快	极高	高
	MQTT	免费	低	较低	较快	极高	低

3 MQTT协议

3.1 MQTT 简介

MQTT 是基于发布/订阅模式的轻量级消息传输协议, 为 IBM 的 Andy Stanford-Clark 博士及 Arcom 公司的 Arlen Nipper 博士于 1999 年发明. 其设计思想是开放、简单、轻量、易于实现, 最初是为大量计算能力有限, 且工作在低带宽、不可靠的网络的远程传感器和控制设备通讯而设计的协议[7]. MQTT 已成为国内外很多企业实现消息推送的首选协议.

MQTT 协议支持绝大部分平台和编程语言, 它具有以下主要几项特性:

(1) 使用发布/订阅消息模式, 提供一对多的消息发布, 解除应用程序耦合;

(2) 对负载内容屏蔽的消息传输;

(3) 使用 TCP/IP 提供网络连接;

(4) 有三种消息发布服务质量:

- “至多一次”, 消息发布完全依赖底层 TCP/IP 网络, 会发生消息丢失或重复. 这一级别可用于环境传感器数据采集等, 丢失一次读记录无所谓, 因为不久后还会有第二次发送.
- “至少一次”, 确保消息到达, 但消息重复可能会

© 中国科学院软件研究所 http://www.c-s-a.org.cn

发生.

- “只有一次”, 确保消息到达一次. 这一级别可用于计费系统等, 消息重复或丢失会导致不正确的结果.

(5) 开销很小(固定长度的头部是 2 字节), 协议交换最小化, 以降低网络流量;

(6) 使用 Last Will 和 Testament 特性通知有关各方客户端异常中断的机制;

3.2 MQTT 原理分析

3.2.1 MQTT 原理模型分析

MQTT 协议定义了三种角色, 消息发布者 Publisher, 消息代理 Broker, 消息订阅者 Subscriber. MQTT 协议的模型图如图 1 所示.

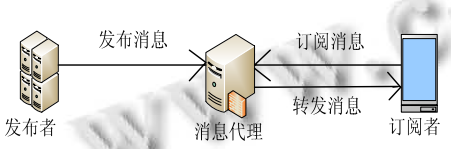

图 1 MQTT 协议模型图

消息代理是服务器端, 主要有以下几个功能:

- 接受发布者和订阅者的注册并负责管理维护;
- 管理主题;
- 存储发布者发来的消息或转发给订阅者;

订阅者作为客户端, 向代理服务器注册并提交订阅的主题, 之后就等待接收该主题的消息.

发布者既可以作为客户端, 也可以作为服务端. 作为客户端是相对消息代理服务器而言, 需要向其注册并发布特定主题的消息. 作为服务端是对订阅者而言, 充当应用服务器的角色, 向订阅者投放消息.

3.2.2 MQTT 消息类型

MQTT 在其固定头部为消息类型字段分配了四个比特位, 这就决定了 MQTT 协议最多支持 16 种消息类型, 去掉被保留的全 0 和全 1 字段, 包含 14 种消息. CONNECT 和 CONNACK 分别用于连接请求和确认, PUBLISH 和 PUBACK 分别用于发布请求和确认, PUBREC、PUBREL 和 PUBCOMP 为 QoS=2 时所用, SUBSCRIBE 和 SUBACK 分别用于订阅请求与确认, UNSUBSCRIBE 和 UNSUBACK 分别用于取消订阅请求与确认, PINGREQ 和 PINGRESP 分别用于维持连接请求与确认, DISCONNECT 用于断开连接. 以 QoS 2 为例, 其报文时序图如图 2 所示.

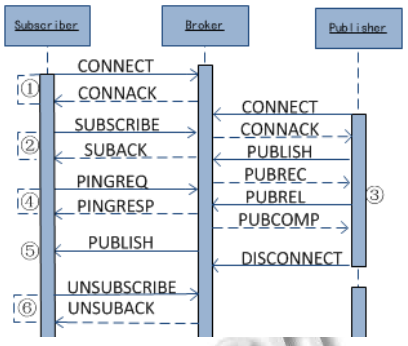

图 2 MQTT 报文时序图

(1) Subscriber 与 Broker 建立连接;

(2) Subscriber 向 Broker 订阅特定主题的消息;

(3) Publisher 与 Broker 建立连接, 向 Broker 发布特定主题的消息, 断开连接;

(4) Subscriber 与 Broker 维持心跳连接, 其方式为 Subscriber 发送 PINGREQ 报文, Broker 回应 PINGREP 报文;

(5) Broker 把从 Publisher 收到的消息转发给 Subscriber;

(6) Subscriber 取消订阅特定主题的信息.

3.3 MQTT 消息流

如前所述, MQTT 协议定义了三种发布消息的服务质量(Quality of Service), 分别是 QoS 0, QoS 1 和 QoS 2.

3.3.1 QoS 0: 至多发送一次

这种情况下消息发送方只向接收方发送一次消息, 不需要接收方的回应, 随后将消息丢弃, . 至于消息是及时送达还是延时、丢失, 发送方并不知情, 完全取决于底层 TCP/IP 网络的状况. QoS 0 的协议流图如图 3 所示.

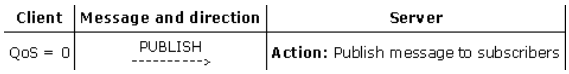

图 3 QoS 0 协议流图

3.3.2 QoS 1: 至少发送一次

消息的发送方通过收到接收方回复的确认报文来保证消息发送成功. 如果由于通信链路或发送设备出现状况导致发送失败, 或者在特定的时间段内发送方没有收到确认报文, 则发送方重发消息, 并将报文头

© 中国科学院软件研究所 http://www.c-s-a.org.cn

部表示消息是否重复的DUP位置为1. QoS 1的协议流图如图4所示.

Client	Message and direction	Server
QoS = 1 DUP = 0 Message ID = x **Action:** Store message	PUBLISH ---------->	**Actions:** • Store message • Publish message to subscribers • Delete message
Action: Discard message	PUBACK <----------	

图4　QoS 1协议流图

由图4可知, Broker收到来自Publisher的重复的PUBLISH报文后, 会向Subscriber重复转发该报文, 并向Publisher回复PUBACK报文. 因此当QoS被设置为1时, Subscriber可能会收到重复的消息.

3.3.3 QoS 2: 精确发送一次

为确保Subscriber不会收到重复的消息, MQTT定义了只适用于PUBLISH型消息的QoS 2. 这是消息传输的最高标准, 会增大网络传输压力, 适用于决不允许出现重复消息的情形, 如计费系统等, 图5为其协议流图.

Client	Message and direction	Server
QoS = 2 DUP = 0 Message ID = x **Action:** Store message	PUBLISH ---------->	**Action:** Store message *or* **Actions:** • Store message ID • Publish message to subscribers
	PUBREC <----------	Message ID = x
Message ID = x	PUBREL ---------->	**Actions:** • Publish message to subscribers • Delete message *or* **Action:** Delete message ID
Action: Discard message	PUBCOMP <----------	Message ID = x

图5　QoS 2协议流图

由图5可看出, Broker在收到PUBLISH报文后有两种处理方式可供选择, 第一种是先存储消息, 向Publisher发送PUBREC报文, 收到PUBREL报文后向Subscriber转发消息并将本地消息删除, 最后向Publisher回复PUBCOMP报文. 第二种是先存储16位的消息ID并向Subscriber转发消息, 然后回复PUBREC报文, 收到PUBREL报文后再删除消息ID, 最后向Publisher回复PUBCOMP报文. 这两种方式的区别在于向Subscriber转发消息的时间点不同, 但两种方式都没有对QoS 2下的消息发布服务质量造成影响.

4　系统设计与实现

4.1 服务器端部署

Mosquitto是一个实现了MQTT v3.1协议的开源代理服务器, 由MQTT协议创始人之一的Andy Stanford-Clark开发, 它为我们提供了非常理想的轻量级数据交换的解决方案[8].

Mosquitto支持Windows、Linux、QNX等多种平台, 本文选择在Ubuntu 12.04上进行安装与配置. 首先从Mosquitto官网获取最新源码包并解压, 解压目录下的config.mk文件包含了Mosquitto支持的所有功能的安装选项, 如TLS/PSK、版本兼容、内存监控等, 可以根据需要对相应功能进行剪裁. config.mk文件修改完成后可使用make install命令进行安装.

安装完成后会在系统命令行里发现mosquitto、mosquitto_passwd、mosquitto_pub和mosquitto_sub四个工具, 分别用于启动代理、管理密码、发布消息和订阅消息. 本文主要用到的是作为Broker的mosquitto和作为Publisher的mosquitto_pub两个命令行工具.

Mosquitto最重要的配置文件mosquitto.conf中定义了端口、心跳时间、客户端最大并发数等重要参数, 可在mosquitto服务启动前进行设置.

4.2 客户端设计与实现

Android客户端使用Eclipse在Windows平台上进行APK开发, 由于Android平台的开源特性, 基本的开发环境配置在网络上和各种开发资料中都有详细介绍, 这里不再赘述.

基于MQTT协议的开发包wmqqt.jar是本次开发所需要的最重要的一个包, 该包可在IBM官网上免费下载, 下载完成后导入Android客户端工程即可.

Android客户端工程主要实现了两个核心类, pushdemo和mqttservice. 前者继承自android. app. Activity类, 实现业务的封装和消息的处理. 后者继承自android.app.Service并实现MqttSimpleCallback接口, 该接口定义了客户端收到推送消息时以及与代理服务器的连接意外断开时需要调用的方法.

客户端在系统中开启后台服务, 当收到推送消息时调用系统通知栏提示对用户进行提示. 图6为Android客户端收到主题为PushTopic, 内容为PushMessage的推送消息时通知栏实际效果图.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

图 6 Android 客户端通知栏

5 系统测试

电量、流量消耗及推送速度是衡量 Android 平台推送系统最重要的三个指标, 本文在 Android 2.3.5 的实体手机上对这三个参数进行了测试.

5.1 测试方法

本文采用美国密歇根大学 Mark GorDon 等人开发的 PowerTutor[9]软件测试手机电量, 通过调用 Android SDK 中提供的 TrafficStats 类的方法来统计流量. 推送时延的测试则通过 Wireshark 软件抓包来实现.

本次测试均在空载状态下进行, 即 Android 客户端程序运行后台服务, 定时发送心跳包保持连接.

5.2 测试结果及分析

图 7 分别对三种 QoS 下传输负载增大时的端到端时延进行了测试. 由图中可以看出负载低于 40KB 时, 三种 QoS 的时延差距很小, 负载继续增大, 三者时延差距逐渐拉大, 但时延都在 1 秒以内.

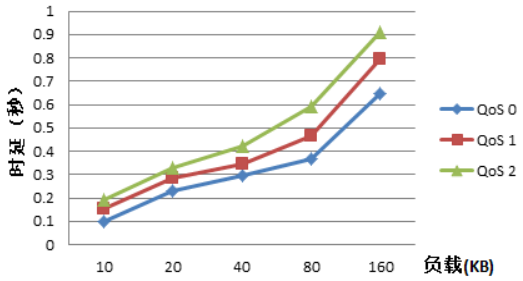

图 7 三种 QoS 下传输负载与时延关系图

图 8 为 12 小时内手机空载状态下消耗的电量图, 其中 T 代表发送心跳包的间隔时间. 由图可看出, 电量消耗与时间增长基本呈线性关系, T=5min 时明显比 T=10min 时电量消耗大很多, 而当 T 继续增大时, 实验发现连接经常断开. 基于电量及系统稳定性考虑, 一般选取 T=10min 比较合适. 经计算 T=10min 时, 每小时消耗电量 0.76J, 24 小时即 18.24J. 按照一般手机电压 3.7V 的标准, 即每天消耗手机电量 1.37mAh. 对于容量一般 1000mAh 以上的手机电池来说, 这点消耗几乎可以忽略.

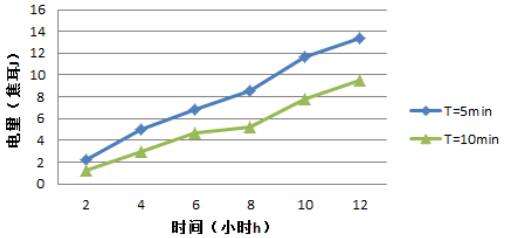

图 8 12 小时内手机空载状态下的电量消耗图

由于 MQTT 固定头部只有 2B, 每次心跳包只需 4B 即可, T=10min 时, 每小时发送心跳包 24B. 心跳包经过网络底层协议封包传输后, 每小时消耗流量约 1.2KB, 即每天消耗流量约 36KB. 本文在空载状态下的流量测试结果与该数值基本吻合.

经以上数据及分析可看出, 基于 MQTT 协议的推送方式功耗低, 效率高, 非常适用于搭载 Android 平台的手机终端.

6 结语

移动互联网的迅速普及使人们对于手机的依赖性越来越高, 消息推送服务不仅节省手机的电量和流量, 还提高了用户获取信息的及时性和有效性. 传统推送方式不区分目标用户, 采取广播机制将信息推送到所有客户端, 不仅加大了网络的拥塞和系统资源的浪费, 还降低了用户体验性. MQTT 协议可以根据目标群体特征实现精准推送, 有效地解决以上问题.

本文对 Android 平台的推送机制及 MQTT 协议的关键技术进行了深入的研究, 设计出基于 MQTT 协议的推送系统, 并在 Android 实体机上进行了测试. 该系统可以使用户及时获取到推送消息, 并大大的减少手机电量和流量的消耗.

参考文献

1 周乐钦,燕彩蓉,苏厚勤.基于 Web-Socket 协议的推送数据技术在监控系统中的应用研究.计算机应用与软件,2013, 30(5):229–232

2 宋新晓.基于 android 的 push 平台功能的实现[学位论文].

© 中国科学院软件研究所 http://www.c-s-a.org.cn

北京:北京交通大学,2012.

3 王克锋.基于 Android 的信息推送管理系统的设计和实现[学位论文].大连:大连理工大学,2012.

4 任超,王鹏,董静宜,等.云模式及其在物联网中的应用.成都信息工程学院学报,2010,25(5):453–456.

5 邹海,李强,邱慧丽.基于 Android C2DM 服务的云端推送研究与实现.计算机技术与发展,2012,22(7):29–32.

6 张弛,陈建明,刘敏,等.基于J2ME平台的IPv6 Jabber系统实现.计算机应用研究,2006,23(5):167–191.

7 Tang KL, Wang Y, Liu H, et al. Design and implementation of push notification system based on the MQTT protocol. International Conference on Information Science and Computer Applications. 2013. 116–119.

8 Lee SH, Kim HW, Hong DK, et al. Correlation analysis of MQTT loss and delay according to QoS level. International Conference on Information Networking(ICOIN). 2013. 714–717.

9 Zhang L, Birjodh T, Qian ZY, et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). 2010. 105–114.

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn