文本复制检测报告单(全文标明引文)

№:ADBD2018R_2018053015312720180530154832440174107975　　检测时间:2018-05-30 15:48:32

检测文献:　53140311_迟喻天_计算机科学与技术_基于Android的游戏信息推送系统的设计与实现

作者:　迟喻天

检测范围:　中国学术期刊网络出版总库

中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库

中国重要会议论文全文数据库

中国重要报纸全文数据库

中国专利全文数据库

图书资源

优先出版文献库

大学生论文联合比对库

互联网资源(包含贴吧等论坛资源)

英文数据库(涵盖期刊、博硕、会议的英文数据以及德国Springer、英国Taylor&Francis 期刊数据库等)

港澳台学术文献库

互联网文档资源

CNKI大成编客-原创作品库

个人比对库

时间范围:　1900-01-01至2018-05-30

检测结果

总文字复制比：0.7%　　跨语言检测结果：0%

去除引用文献复制比：0.7%　　去除本人已发表文献复制比：0.7%

单篇最大文字复制比：0.4%（消息提示类控件使用之Notification(状 态栏通知)详解 - MakeYourChance的专栏 - CSDN博客）

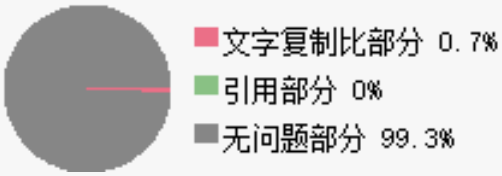
重复字数：[215]　　总段落数：[7]

总字数：[29214]　　疑似段落数：[3]

单篇最大重复字数：[115]　　前部重合字数：[0]

疑似段落最大重合字数：[115]　　后部重合字数：[215]

疑似段落最小重合字数：[47]



指　标：☐疑似剽窃观点　☑疑似剽窃文字表述　☐疑似自我剽窃　☐疑似整体剽窃　☐过度引用

表格：0　　公式：0　　疑似文字的图片：0　　脚注与尾注：0

0%（0）	中英文摘要等（总1257字）
0%（0）	第1章绪论（总6214字）
2.2%（53）	第2章信息推送服务原理（总2391字）
1.1%（47）	第3章游戏信息推送系统的设计（总4270字）
1.5%（115）	第4章游戏信息推送系统的实现（总7676字）
0%（0）	第5章游戏信息推送在实际项目中的应用（总3599字）
0%（0）	第6章总结与展望（总3807字）

（注释：无问题部分　文字复制比部分　引用部分）

1. 中英文摘要等　　总字数：1257

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

本科生毕业论文（设计）

中文题目基于Android的游戏信息推送系统的设计与实现。

英文题目Design and implementation of game infomation push system based on Android.

学生姓名迟喻天班级 14级3班学号 21140311
学院计算机科学与技术学院
专业计算机科学与技术
指导教师杨滨职称讲师

吉林大学学士学位论文（设计）承诺书

本人郑重承诺：所呈交的学士学位毕业论文（设计），是本人在指导教师的指导下，独立进行实验、设计、调研等工作基础上取得的成果。除文中已经注明引用的内容外，本论文（设计）不包含任何其他个人或集体已经发表或撰写的作品成果。对本人实验或设计中做出重要贡献的个人或集体，均已在文中以明确的方式注明。本人完全意识到本承诺书的法律结果由本人承担。

学士学位论文（设计）作者签名：

2017年5月20日

摘要

游戏行业信息推送技术的发展和应用使服务器下发通知给客户端的需求更加简单有效，但国内不能直接使用Android提供的系统级推送产品GCM，文章简单介绍了目前游戏行业信息推送需求实现的一些机制与现状并提供了一种行之有效的基于Android-Service组件的无限推送请求设计结构，并实现了这种结构的基本框架，对比并分析了这种设计结构与其他模式的推送结构的异同与适用场景。

基于Android的游戏信息推送系统的设计与实现

关键字：推送通知，游戏，Android-Service

Abstract

The development and application of information push technology in the game industry makes the demand of sending notification to the client more simple and effective, but the system push product GCM provided by Android can not be directly used in China. The article briefly introduces some mechanism and status of the implementation of information push demand in the game industry and provides a kind of line. The design structure of infinite push request based on Android-Service component is effective, and the basic framework of this structure is realized, and the similarities and differences between this design structure and other modes are compared and analyzed.

Design and implementation of game infomation push system based on Android

Key words：push notification，games，Android-Service

目录

2. 第1章绪论

总字数：6214

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第1章绪论

1.1内容安排

本文按照基础、理论、设计、实现、总结的顺序，共分五个部分，第二章主要介绍一些基础的技术性内容；在此基础上，第三章介绍该设计的设计思路及遵循的设计理念；第四章具象化设计，以实际的问题及解决方案为例讨论此设计的适用场景；第五章则重点介绍这种技术在游戏行业中的应用场景；第六章为总结性章节，讨论这个技术的发展始末以及对未来的展望。

1.2课题背景及研究现状

普通的主机游戏运行在本机，除一些防盗版手段外并不需要与服务器端进行交互。但自从1997年由美国艺电公司（Electronic Arts 简称EA）开发的第一款网络游戏《Ultima Online》问世后，客户端服务器信息交互的需求就迫在眉睫。

由于游戏信息交互被用于诸多不同场景，因此诞生了数种不同的信息交互模式，主流的交互模式有：http端口通信、在线tcp交互通信、udp实时信息转发通信、离线推送等。

下面介绍几种信息交互模式及其试用场景：

1.2.1 http端口通信

服务器模块监听一个指定的端口如http://localhost:8080, 根据接收到的请求内容的不同触发不同的逻辑。

在参数合法的情况下会调用handleRequest(WebProxy proxy, HttpServletRequest req, HttpServletResponse resp)方法，具体的内部逻辑由该抽象类的实现类定义，其返回值作为结果显示在网页页面上。

在handleRequest中可以调用服务器内的一些方法达成自己的目的，例如：刷新服务器排行榜、发送全服邮件等。

由于该通信方式基于http协议，不适用于频繁的请求，故这种通信方式一般用于服务器与服务器之间或gm后台与服务器之间命令的触发与执行。

图1-1 http请求示例

```
1. protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
2. Zone zone = getZone(req.getParameter("zoneId"));
3. String ret = null;
4. if (zone != null) {
5. WebProxy proxy = (WebProxy) zone.getProperty("proxy");
6. boolean webCheck = proxy.checkRequestIp(getClass().getName(), req.getRemoteAddr());
7. if (!webCheck) {
8. resp.setCharacterEncoding("utf-8");
9. resp.setContentType("text/html; charset=utf-8");
10. resp.getWriter().write("ip error");
11. } else {
12. try {
13. ClassLoader clazzLoader = Thread.currentThread().getContextClassLoader();
14. try {
15. Thread.currentThread().setContextClassLoader(proxy.getClass().getClassLoader());
16. ret = handleRequest(proxy, req, resp);
17. } finally {
```

```
18. Thread.currentThread().setContextClassLoader(clazzLoader);
19. }
20. } catch (Exception e) {
21. e.printStackTrace();
22. }
23. }
24. } else {
25. resp.setCharacterEncoding("utf-8");
26. resp.setContentType("text/html; charset=utf-8");
27. resp.getWriter().write("zone error");
28. }
29. if (null != ret) {
30. resp.setCharacterEncoding("utf-8");
31. resp.setContentType("text/html; charset=utf-8");
32. resp.getWriter().write(ret);
33. }
34. logger.info("web request: servlet path {}, path info: {}, zone {}", req.getServletPath(), req.getPathInfo(), zone != null ? zone.getName() : "null");
35. }
36.
37. protected abstract String handleRequest(WebProxy proxy, HttpServletRequest req, HttpServletResponse resp) throws Exception;
```

1.2.2在线tcp交互通信

Tcp连接作为最常见的一种游戏内客户端服务器的交互方式在各个游戏项目中广泛使用，玩家登陆时，客户端与服务器建立tcp连接，在tcp连接建立之后每次的协议请求或服务器信息下发均通过此tcp连接传递。

如下代码中的extension对象调用的方法handleClientRequest即是服务器处理客户端请求协议的过程。

```
1. public void execute() throws Throwable {
2. ISFSObject msgObj = request.getMsgObj();
3. String command = msgObj.getUtfString("c");
4. if (StringUtils.isBlank(command)) {
5. throw new RequestException("Game Request error, no cmd");
6. }
7. ISFSObject param = msgObj.getSFSObject("p");
8. User user = COKServer.getInstance().getUserManager().getUserBySession(this.request.getSession());
9. if (user == null) {
10. throw new RequestException("user is null, request: " + this.request.toString());
11. }
12. IServerExtension extension = user.getZone().getExtension();
13. if (extension == null) {
14. throw new COKExtensionException("extension is null");
15. }
16. extension.handleClientRequest(command, user, param);
17. }
```

除对实时性要求极高的战斗模块可能使用udp协议进行连接通信外，大部分其余模块均使用玩家登陆时建立的tcp连接来处理玩家的请求及推送必要信息。（如：玩家正在被其他用户攻打、收到其他玩家赠送的礼物等。）

图1-2 tcp连接示例

1.2.3 udp实时信息转发通信

在游戏行业中，udp协议往往备受争议，因其自身并不保证信息的准确性，但传输速度一般情况下要比tcp快，所以一般只有对数据精度要求不高且要求即时反馈的模块才会使用udp协议进行通信（比如ARPG类游戏的实时战斗）。

这样的通信模式在服务器端有一个特点，即：服务器往往只进行信息的转发而不对信息加以验证。

图1-3 udp转发示例

适用场景举例：两个玩家决斗的胜负以首先推送胜负的客户端为准，因此在推送胜负结果时客户端往往会带一个毫秒级时间戳以便于服务器判断先取得获胜结果的客户端以及客户端和服务器约定好的私钥以保证安全性。只有客户端校验胜利且服

务器下发胜利确认信息后客户端才弹出战斗胜利提示。

1.2.4离线推送

当玩家并不处于在线状态而存在一些信息需要通知给用户时，就涉及到离线推送需求。

当前的主流移动平台有IOS和Android两种，IOS和Android背后的苹果和谷歌公司都为其提供了系统级推送软件，在Android上有GCM，而在IOS上有APNS，这两种推送方式被由于其性能的优越性被世界上绝大多数国家和地区接受。但在中国大陆GCM尚不可用且Android平台品类繁多，因此Android的离线推送对国内的移动应用而言需要更多的精力来维护。诸多国内应用程序开发者采用了这样的一种策略：通过Android的后台Service维持与服务端的通信，这个通信既可以是基于Http协议的端口监听也可以是基于Tcp协议的socket接口，使用tcp协议时的性能和效率可能优于http协议，但不论采用哪种网络协议，将服务端的待推送信息下发给客户端都是其最终要实现的需求。

但是，这种做法存在一些技术难点：首先，客户端需要通过定时的心跳信号来维持与服务器的tcp长连接，这个心跳的频率难以掌控，心跳频率高则客户端负担加重，心跳频率低则可能使连接被断开；其次，客户端的网络环境极不稳定，使用的网络切换、基站的切换都会引发网络波动，这样的情况下，不同环境下，tcp的重连动作需要独立分析；再者，服务器需要维护成百上千的客户端tcp长连接，对一个服务器来说这样的规模该预分配多大的空间需要大量的数据分析统计，能否实现动态空间变换仍旧是难以解决的问题；针对IOS、Android的不同进行两套信息推送系统维护的成本对于小团队而言确实太大。

在中国大陆有很多专门为此需求提供解决方案的第三方推送软件，它们提供统一的接口，并向其支持的所有Android转发信息，以此来实现一次推送多处响应的便利性需求。

如下代码即是亚马逊的第三方消息推送接口的使用：

```
1. public static void pushMessageExceptTarget(String uid, String message, PlayerParsePushInfo.PUSH_TYPE pushType, String soundKey, boolean oldVersionAlert, PushTarget exceptTarget, ParseStatType statType) {
2. try {
3. if (StringUtils.isBlank(uid) || StringUtils.isBlank(message)) {
4. return;
5. }
6. ParseInfo parseInfo = ParseInfo.getWithKey(uid, pushType.ordinal());
7. if (parseInfo != null) {
8. if (parseInfo.getStatus() == PlayerParsePushInfo.PUSH_STATUS.OFF.ordinal()) {
9. return;
10. }
11. }
12. if (soundKey == null) {
13. if (parseInfo != null) {
14. soundKey = String.valueOf(parseInfo.getAudio());
15. } else {
16. soundKey = String.valueOf(PlayerParsePushInfo.PUSH_STATUS.ON.ordinal());
17. }
18. }
19. if (AwsPushKeys.isPush()) {
20. UserPush userPush = UserPush.getWithUid(uid);
21. if (userPush != null && userPush.getActive() == 1 && !StringUtils.isBlank(userPush.getUserid())) {
22. UserProfile userProfile = GameEngine.getInstance().getOnlyUserProfiele(uid);
23. if (userProfile != null && !userProfile.isUserMovedToOtherServer()) {
24. ISFSObject pushData = new SFSObject();
25. pushData.putUtfString("pushType", "" + pushType.ordinal());
26. if (StringUtils.isBlank(userPush.getName())) {
27. userPush.setName(userProfile.getName());
28. }
29. pushDataByAws(oldVersionAlert, true, message, soundKey, userPush, exceptTarget, statType, pushData);
30. }
31. }
32. } else {
33. PairUtil<String> parseUser = UserService.getParseUser(uid);
34. postData(oldVersionAlert, true, message, soundKey, parseUser.getFirst(), parseUser.getSecond(), exceptTarget, statType);
```

35. }
36. } catch (Exception e) {
37. COKLoggerFactory.monitorException("push error", ExceptionMonitorType.OTHER, COKLoggerFactory.ExceptionOwner.COMMON, e); }}

3. 第2章信息推送服务原理

总字数：2391

相似文献列表　文字复制比：2.2%(53)　疑似剽窃观点：(0)

1	201054080203_郑晓萌_基于wifi的android音乐共享与传输系统设计与开发_彭玉旭 郑晓萌 - 《大学生论文联合比对库》- 2014-06-04	2.2%（53） 是否引证：否
2	基于Android的个人日程管理软件 柯彰 - 《大学生论文联合比对库》- 2015-05-15	2.2%（53） 是否引证：否
3	1106102002 柯彰 - 《大学生论文联合比对库》- 2015-05-25	2.2%（53） 是否引证：否
4	基于Android平台的“乐云记事” 张昕 - 《大学生论文联合比对库》- 2015-06-01	2.2%（53） 是否引证：否
5	112448_张昕_基于Android平台的“乐云记事”应用程序开发_论文定稿_1433665739998 张昕 - 《大学生论文联合比对库》- 2015-06-12	2.2%（53） 是否引证：否

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第2章信息推送服务原理

2.1基于Android的Activity组件与Service组件

2.1.1 Activity介绍

1.Activity 简介

作为Android的基础接口组件，Activity提供了与用户之间的交互接口。其需要layout来设置界面，一般而言我们通过setContentView(View)部署对应的UI到其activity窗口或使用ActivityGroup嵌套activity来实现activity与全屏窗口的连接关系。[1]

2. Activity的生命周期

Activity的生命周期如图2-1所示。

从图2-1可见Activity共有启动、停滞、停止、毁灭四种不同的状态，刚刚启动应用程序后Activity触发onCreate-onStart-onResume进入启动状态；当锁屏或被其他Activity覆盖则触发onPause进入停滞状态；而停滞状态下的Activity可通过onResume恢复启动状态，这一过程一般伴随着解锁屏或是被覆盖状态的复原；在停滞状态下长时间不被使用的activity通过调用onStop方法被停止，再通过onDestory方法被毁灭。

总结一下，Activity的启动态通过onPause进入停滞态；停滞态通过onStop进入停止态，通过onResume恢复启动态；停止态通过onDestory进入毁灭态，通过onRestart恢复启动态；一旦进入毁灭态，Activity就只能重新通过onCreate创建。

从onResume到onPause这一过程中，Activity是用户可见的，Activity将频繁地在停滞态和启动态之间切换，如静止的手机即将进入休眠状态。因此在停滞态和启动态之间的逻辑代码必须是非常轻量级的，不能产生过多的程序开销，否则移动应用将变得十分卡顿。

图2-1 Activity生命周期

2.1.2 Service介绍

Serveice是Android提供的基础组件，它的使命是去执行那些后台线程（所谓后台线程就是指需要执行那些一直运行且不必直接与前台页面交互的耗时任务的线程）。Service必须建立在创建它本身的进程，一旦创建Service的进程被杀死则Service一定会停止，而且我们也无法将Service放进那些单独运行的进程之中去，因为它们不能够与Android的其他组件进行直接的通信和交互，如果那样做就失去了我们构建Service的意义了。我们可以通过Service记录很多信息，例如每日计步这样的功能就很适合构建在Service中。总结一下Service就是它是对用户透明的后台功能载体。[2]

Service和Thread的关系：

Thread是java中一个非常常见的概念，通常情况下Thread都被用于开启一个新的子线程。在Android中构建Thread的目的一般是为了执行一些耗时操作以避免主线程被阻塞。一般而言执行完其任务的线程就会被杀死以回收Android手机那有限的内存资源，而有些优良的设计模式考虑到线程的开辟与杀死也需要消耗一定的性能所以提出了线程池的概念，也就是说固定开辟指定个数的线程将其维护在线程池（ThreadPool）中，每当需要线程执行一个独立任务时就从线程池中取出线程，当线程执行完其任务后不进行资源回收而是直接将其放置回线程池中。然而这种设计模式中，线程池的规模与业务逻辑的关系比较难掌握，因此这是一种需要丰厚的研发经验的设计模式。

Service与Thread的关系是一种平台关系，Service本身并非子线程的一部分，Thread被构建在Service上，并依赖Service的存在而存在。Service则向外界提供与子线程Thread的交互接口。

Service的存在是为Activity和其他组件提供一个与Thread交互的通道，如果没有Service，Activity等众多组件都无法与

Thread进行直接的交互，只能通过共享内存等其他的技巧，这就为Android编程带来了复杂性。尤其是当前Activity进入毁灭态以后，前一个被销毁的Activity所创建的子线程将无法再被获取到。但因为有了Service，Thread就被独立了出来，所有Activity与Service的关联都非常简单。即是当前的Activity被销毁了，只要通过Binder就可以获取Service的信息。因此在Service处理后台操作时，Android编程的复杂性就会下降。

2.2 两种离线推送实现策略

2.2.1 无限访问请求

服务器构建消息队列MessageQueue，客户端构建Service每隔一定时间间隔向服务器发送一次请求，查看当前消息队列MessageQueue中是否有待推送的消息；如果有待推送消息，则消息队列MessageQueue中第一条信息出队列，将此信息作为待推送信息推送给客户端；如果没有待推送消息，服务器通知一个特殊标志给客户端以达成不进行消息推送的目的。在此Service存活的前提下，服务器向Service推送消息后，Service根据特殊标志进行反应。

2.2.2 指定ip推送消息

服务器在接收到待推送的消息后，直接向所有客户端推送此消息；Android客户端构建专门接收服务器信息的Service，当从中接收到服务器的推送消息后，将信息弹出显示。但这样的实现策略存在一个问题：真实环境下客户端的ip地址会随着WI-FI、蜂窝、信号源的变换而变换，服务器需要构造单独的客户管理模块以动态获悉客户端的唯一ip。

由于单独的客户管理模块往往与具体业务衔接紧密，实现和构造单独的客户管理模块难度较高，故此设计采用2.2.1中的无线访问请求策略来实现离线推送功能需求。

指　标

疑似剽窃文字表述

---

1. Activity的生命周期

   Activity的生命周期如图2-1所示。

   从图2-1可见Activity

4. 第3章游戏信息推送系统的设计　　总字数：4270

相似文献列表　文字复制比：1.1%(47)　疑似剽窃观点：(0)

1	基于APNS的消息推送服务的设计与实现 吴栋淦; -《电脑与信息技术》- 2016-02-15	1.1%（47） 是否引证：否

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第3章游戏信息推送系统的设计

3.1 概述

3.1.1 特殊概念简介

保活策略[3]：

Service虽然提供了Android后台管理线程，但当线程死亡，对应的Thread也将被中止，无法继续其工作。这种前提下就诞生了保证线程存活的需求，对应的保活策略有很多种，下面介绍两种主流的解决方案：

1.双进程守护策略

启用进程二实时查看进程一的状态，如果发现进程一已经死亡则对其进行状态恢复。

2.JobScheduler执行任务调度保活

JobScheduler类由google于21版本新推出的api提供，官方文档中对此类的介绍含义大致如下：当收到回调，框架将智能处理，并尝试尽可能推迟。一般情况下，如果不指定工作期限，则其会根据作业调度器的当前状态的内部队列运行在任何时候，然而它仍然可能被延迟到下一次该设备连接到一个电源。因此这个任务其实执行在设备空闲期，且系统设计的这个api对硬件性能消耗极低，其本意是用来执行一些任务调度的，但如果用这个类来执行我们的检测进程，那么也一定会执行在设备空闲期。所以只要在这个类中对进程进行检测，就可以在不新启动一个进程的情况下由系统直接校验并开启对应的进程。

3.1.2 设计前的考量

本设计采用2.2.1中介绍的无线访问请求策略，采用无线访问请求策略主要是出于以下两点考量：

1.降低双端程序耦合度

2.2.2中介绍的指定ip推送消息策略，依赖于动态的客户管理模块构建的一一对应的mac地址与ip地址的实时映射关系，而客户管理模块对此映射关系的维护往往与其他大量信息存在耦合：例如该用户的地理信息、当前网络状态、历史平均网络ping值和历史行为等等，且对应的服务器处理模块也存在大量的信息耦合：例如接收到的信息处理状态限制等，从中单独剥离出ip推送消息存在较大的成本。

2.剥离保活需求

2.2.1中介绍的无限请求访问，可根据客户端的需要随时开闭请求访问通道，通过这样的方式控制访问通道，可剥离保活

策略对应用的影响，从而减小客户端程序的涉及面，降低客户端程序的复杂度。

3.1.3 信息推送系统设计原理

无限访问请求并非成熟的推送服务系统的解决方案，这只是众多实现离线推送设计中的一个解决方案的demo。这里我们采用双端设计（即服务器与客户端，这也是最普遍的框架模式）。客户端与服务器通过预先设定好的接口进行通信，当特定的接口被触发后，服务器/客户端按照预先的设定处理业务逻辑并将返回值提供给对方。这是一种简化的通信模式，出于简单的考虑，这里采用了性能较低的http协议而未使用tcp协议。使用tcp协议的好处在于只需要建立一次tcp连接，而http协议是基于tcp协议的，每一次http通信都伴随着一个tcp连接的建立与关闭，这里的性能开销要大于tcp协议，但http协议有现成的jar包API，降低了编程难度。

总体而讲此推送服务系统设计如下图所示：

图3-1 架构示意图

其中，客户端在Tread开启的状态下每隔三秒向服务器发送一次请求，请求附带唯一注册的appKey，根据服务器传递回来的json信息判断是否推送此消息。服务端在收到带有appKey的请求之后，从消息队列池MessageQueuePool中取出对应的消息队列MessageQueue（appKey与MessageQueue存在一一对应关系，对应的MessageQueue在服务器启动时根据注册表注册），并取出最早一条消息message进行推送，如果此时messageQueue为空队列则返回值为空，在服务端逻辑层发现返回值为空则设置空状态信号state发送给客户端。

3.2 服务器设计

服务端逻辑的核心在于对消息队列appKey-MessageQueue映射的处理。因服务端要服务于多个用户端，而针对每一位用户，其消息队列应是唯一的，故需要存在一个一对一映射关联用户的唯一标识和其唯一消息队列。在操作用户的映射关系时，存在多个线程操作同一个内存map的情况，在这种情况下考虑到HashMap并非线程安全的类，在这里选择线程安全的ConcurrentHashMap来进行映射关系的存储（ConcurrentHashMap通过分段锁技术实现真正的并发访问）。

Servlet注册Web框架，通过访问Servlet并携带url参数对核心消息队列池进行访问，考虑到涉及到对MessageQueuePool的处理涉及到添加信息和取出信息两方面，需设计两个不同的Servlet来完成这两个请求的逻辑细节。

服务器设计遵循以下原则：

抽象原则：由于多次使用核心映射关系appKey-MessageQueue，故整合其映射关系为一个Pool类，在外部调用发生时都通过此Pool类进行交互。将用户信息推送需求抽象为消息推送队列MessageQueue，每位用户唯一对应一个MessageQueue。

线程安全：线程安全是指多个线程同时访问或处理同一处内存结构或数据时其结果始终一致的特性，ConcurrentHashMap的内部间隙锁机制保证了在多个线程同时访问一个映射关系map时的内存结构正确性。

核心封闭：对外保证此MessageQueue的私有性，外部只能通过对应的公开Pool接口getMessage、getSize、insertMessage查看或操作其MessageQueue的状态详情。

高内聚、低耦合：此系统可与其他服务器设计轻松关联起来。内部联系紧密，每一块逻辑、框架都与实现信息从服务器到客户端的传递关系密切。

应用服务器的功能主要有两个：处理客户端请求消息和后台URL添加信息给所有消息队列。下面以推送信息流程为例来讨论服务器的设计思路。处理客户端请求功能需要获得对应的唯一标识appKey，其中客户端发送请求时将客户端绑定的appKey作为参数传递过来。若传递的appKey合法且服务器的消息队列中有待推送的信息则服务器能够获取到最新的待推送信息并将之返回给客户端，由客户端识别其标识合法性后进行推送处理。后天URL添加信息给所有队列功能由gm后台直接调用，将信息作为参数传入其中，并遍历MessageQueue，将其信息存入队列之中。

服务器使用2个接口来处理主要业务逻辑，其中PushSmsServletApi接口负责处理客户端请求，UserDataServletApi接口负责向所有MessageQueue中添加信息，服务器工作流程见图3-1中Server部分所示。

PushSmsServletApi所需参数及返回值详细情况如下：

入参：appKey，唯一标识一个应用程序所属用户，在服务器注册。

返回值：json字段，含有标识信息state字段，为0时表示无有效信息，不进行推送，为1时表示存在有效信息，需进行推送；content，待推送正文；date待推送信息的推送时间；number，剩余待推送信息的数量。

UserDataServletApi调用url所需参数及返回值详细情况如下：

入参：message，待注入到所有用户消息队列的信息。

返回值：字符串，已经加入到用户消息队列之中的信息。

3.3 客户端设计

客户端的主要业务逻辑是向服务器发送消息并接收其返回值，并根据其消息返回值的状态校验是否应该以通知的形式弹出。在状态合法通过的情况下进行通知。

客户端的核心模块是后台服务器推送消息的服务Service，在Service的内部构建了线程Thread以实现异步与服务器进行交互的需求。在Service构建的Thread内向构建好的org.apache.http包下的HttpClient对象中请求对应的url以实现访问请求，在这里使用http协议进行客户端和服务器的交互。发起http请求并处理其返回结果的过程如下：

1. 构建HttpUriRequest对象，通过实例化其子类HttpGet并向构造方法中传入指定的url以get方式访问对应的url构建对应

的HttpUriRequest对象。

2. 构建对应HttpClient对象，为了成为http协议的发起者，客户端需定义自身的HttpClient对象才能执行对应的HttpUriRequest，这里取默认的HttpClient设置。

3. 通过HttpClient执行HttpUriRequest，向服务器发起http请求并取服务器返回信息HttpResponse对象httpResponse，根据返回的httpResponse内部状态码StatusCode对此次通信结果进行确认。当对应的response返回值为200时可确认此次传输成功。

4. 通过缓冲读入器对httpResponse的返回结果进行读入，将返回的json串存为字符串json。

5. 对字符串json进行反序列化获得其内部结构并取出内部的状态码state、正文content、信息发送时间date、剩余待推送信息数量number。

6. 根据状态码state校验是否发送通知信息，若校验通过则使用通知管理器向系统发起通知推送请求。

在客户端要添加对应的开关以控制网络通信访问，只有当开关开启的情况下，客户端才向服务器提起请求。因此在核心Activity对应的界面layout上设计了两个button按钮作为开关，当点击开启按钮后，Service内局部变量flag赋值；点击关闭按钮后，flag值变换。在进行推送时，根据flag实时值进行信息交互校验。

客户端设计遵循以下原则：

兼容原则：由于AndroidSDK不同版本之间的差异性明显，在进行设计时，对于即将在后续版本消失的类，均采用其他的形式实现。对于之前的SDK版本支持但当前版本已经删除的类，使用了之前版本兼容的手段进行实现。

多线程延时操作原则：延时操作是指那些需要较长时间进行的操作，如不使用多线程进行，一个延时操作会阻塞程序主线程造成操作卡顿的糟糕体验。在客户端中，所有的延时操作如网络通信，均使用多线程方式进行处理。

简约原则：对客户端UI与架构的设计均以最简为优，没有冗余繁杂的设计及操作以保证信息的展示的纯粹性。

客户端调用通知管理器NotificationManager进行信息推送时，使用了其内部类Notification.Builder构造链进行推送消息的设置，其主要功能是提示功能。

第4章游戏信息推送系统的实现

4.1概述

服务器代码和客户端均使用java编写，语言环境选择目前已稳定的jdk1.8，客户端AndroidSDK选择Android API 26 PlatForm。服务器选择tomcat7.0.86进行承载。

java8提供了一种名为lambda表达式的新特性，这种特性支持函数式编程较明显地简化了代码。AndroidSDK版本和tomcat版本选择均出于稳定性考量。

4.2 服务器实现

按照3.2中对服务器设计阐述，服务器拥有两个关键接口PushSmsServlet.java、Sms.java。

PushSmsServlet.java的核心操作是根据传入的appKey获取其唯一对应的消息队列MessageQueue中的最近一条信息，当消息队列MessageQueue中没有消息或传入的appKey非法，则在业务逻辑中对这种情况添加处理，设置其状态值避免推送。

在这里先介绍json并对与json同类的服务器客户端推送解决方案protoBuffer和flatBuffer进行简单的对比。

Json是一种相当简化的数据交互形式，其将对象中的各个属性以键值对的形式表现出来，例如["state":"1"；"content"："jack"]的形式来表现一个state属性值为1，content属性值为jack的实例对象。其作用就是将信息序列化并以可传输的序列形式准确标识原本的信息。

序列化问题目前还有另外两种应用面较广的解决方案，protoBuffer和flatBuffer。

与json不同的，protoBuffer采用的序列化形式是二进制文件。正如我们都了解的，计算机以二进制的形式进行数据的处理，也就是说，json序列化的结果在计算机处理上，和二进制文件本质上是一样的，但是为了便于人眼处理，提升可读性，json序列化的结果对应的文件规模要大；而protoBuffer直接将结构信息序列化为二进制文件，就避免了信息的冗余，极大地缩小的待处理信息的规模。以开发经验的角度讲，protoBuffer序列化的结果序列一般是json序列化结果规模的十分之一。因此

可以认为protoBuffer对序列化问题的性能方面有了十倍的提升。但protoBuffer也有其美中不足的两方面：二进制文件的可读性问题和伴随的结构体关系统一。二进制文件和字符串，在不考虑数据规模的前提下后者的可读性要远远高于由0和1组成的二进制文件bin；一个单独的二进制文件只能序列化其传输的数据内容，而服务端与客户端要想解析对应的数据，必须均拥有相同的关系型内存数据结构，这就意味着protoBuffer产生的二进制文件不能自己表述自己。双端（客户端和服务端）仅仅根据二进制文件无法将其复原为内存结构，必须同时配以数据文件才能解析。

如果说protoBuffer提供的二进制文件和对应数据关系模型还保留了一定的可读性（至少根据类结构可以分析出其想要表示的意义），那么flatBuffer就是在可读性与程序性能上走向了后者的极端。是的，flatBuffer与protoBuffer类似，但其并不需要生成对应的二进制文件，flatBuffer本身在内存中就以二进制的形式存储，需要对其进行使用时也直接处理为需要的格式。正因如此，flatBuffer在进行客户端服务器通信时就如同同一台PC机上两个类的相互调用一样简单。但作为代价，flatBuffer的可读性一般较差，对flatBuffer构建的模块进行逻辑修改是相当痛苦的一件事。

总结json、protoBuffer和flatBuffer，json效率最低，可读性强，使用简单；protoBuffer效率偏高，可读性一般，使用较难；flatBuffer效率最高，可读性极低，使用极难。在本设计中考虑到主要的目的是设计和实现信息推送系统，因此使用了最简单的json来维持客户端与服务器之间的通信。

Sms则是json序列化生成类，为了实现客户端与服务器的通信，需要json来将内存数据和对应结构序列化并传给客户端进行信息反序列化和业务处理。Sms类内共有四条属性：int类型的state标识此次通信的信息状态、String类型的content为带推送消息正文，String类型的date是本次通信发送时服务器时间，String类型的number是当前appKey在服务器唯一对应的消息队列MessageQueue中还剩下的消息的数量。

UserDataServlet.java

为了向用户的信息类中添加待推送消息，构建了UserDataServlet接口，待输入参数message字段标识此次向消息队列中添加的信息。

在UserDataServlet中预设了条件筛选器condition，其作用在于筛选待推送消息的用户（例如只给40岁以上的用户推送这一条消息）。具体的限制条件可以以lambda表达式注册在服务器的UserData模块中，在推送消息时可以对条件组conditionGroup进行且校验处理，只有符合全部条件的用户消息队列才会被注册这一信息，并在下一次请求发送至服务器时进行信息的下发。

Pool.java

Pool是服务器操作的消息队列核心，内部的静态不可变常量对象map维系着appKey与消息队列MessageQueue的一一映射关系。这里的静态不可变属性由static final关键字保证，而不可变也并不意味着map不可修改，此不可变指的是map对应的内存地址在类加载到服务器进程中止期间不可修改，至于此内存地址上的值的修改并不被这个关键字限制。

Pool提供了三个不同的外部交互接口：getSize(String)、getMessage(String)、insertMessage(String)。由于映射关系map的核心重要性，不能够对外部公开，因此在这里使用private封闭map并向外部提供接口来与map交互。

getSize(String)：入参String为唯一的appKey，此方法将返回合法appKey对应的消息队列中剩余信息的条目数。对于非法的appKey，getSize返回-1，对于空消息队列，getSize返回0。

getMessage（String）：入参String为唯一的appKey，此方法将返回合法appKey对应的消息队列中下一条待推送消息的内容。对于非法的appKey或空消息队列，getMessage返回null。

insertMessage（String）：入参String为待插入的信息Message，此方法目前对所有的消息队列都进行一次信息设置。未来如果有用户限制可在参数中加入限制条件。

4.3 客户端实现

核心业务PushSmsService：

在PushSmsService的onCreate中添加了id为my_channel_01的通道channel，这个channel主要是用于推送的设置。Notification.Builder在设置信息的时候缺少channel则NotificationManager无法推送其通知给系统。因此必须预先为NotificationManager设置好一个channel，在onCreate中添加此channel目的在于不重复设置channel，提升一定的客户端程序响应速度。

这里介绍一下PushSmsService中对http请求返回信息基于httpResponse的处理：http协议在请求结束时的状态码statusCode标识了这次访问的过程：状态码为200意味着这次访问正常；状态码为404则代表请求的资源未找到，出现这种情况可能是原本的资源被删除或是客户端发起请求时的资源名输入错误；状态码为500则意味着在请求过程中出现了异常，一般情况下错误堆栈会随500的状态码一起显示在网页上.....

不同的状态码标识了不同的信息，客户端根据这些预先约定好的信息对每一种情况进行着特殊的业务逻辑处理。

在PushSmsService中构建的线程MyThread自运行启动就开始了三秒钟一次的间隙访问，这里每三秒钟进行一次的设计是通过线程休眠(ThreadSleep)来实现的，这样的定时触发逻辑还可以通过定时器（ScheduledExcutorService）来实现。

ScheduledExcutorService使用了我们介绍过的线程池思想，凡是进行定时任务且共同使用同一个定时器ScheduledExcutorService的任务都共享一个线程池ThreadPool，当任务真正执行时从ThreadPool中获取一个线程执行其任务，当任务结束后不释放线程资源而是将其放回到线程池中去，为下一次线程任务调度做准备。在判断任务应当执行之前，ScheduledExcutorService一直以一种轮询状态在运行，直到其找到第一个待执行任务，才进行任务执行状态。

出于对客户端复杂度的考量，本设计采用原本的线程休眠设计简约实现问题，但如果涉及到多个线程共同处理同一个定时任务，则应当使用定时器模式，因为ScheduledExcutorService内部包含间隙锁可以保证线程安全，而简单的线程休眠会导致进程的阻塞。但在单线程的前提下，线层休眠的代码简约性和使用便利性都要优于ScheduledExcutorService。

通知管理方法notification():

为了将服务器下发的消息显示为通知弹出，需要一个单独的模块来实现这一处理。通知管理方法notification就是这一模块的核心，调用Android操作系统弹出通知的操作如下：

第一步：从Android操作系统获取通知栏构造器Notification.Builder的一个实例builder。

第二步：获取状态通知栏管理notificationManager实例化对象。注意：（通过Android操作系统提供的getSystemService(NOTIFICATION_SERVICE)方法获得的实例化对象需要经过向下转型转为NotificationManager才能使用NotificationManager中提供的方法）

第三步：构建通知栏PendingIntent类实例化对象contentIntent。（可同时设置点击意图等信息）

第四步：通过构造者模式对Builder进行配置，将预先准备好的标题title、正文content、点击意图contentIntent、角标Ticker、通道标识channelId均以set方法设置到builder之中。如果需要设置呼吸灯声音等属性可使用setDefaults方法将Notification内的相关配置作为参数传入。相关配置信息如下：

Notification.DEFAULT_VIBRATE 添加默认震动提醒，需要一定的用户权限才可以调用（VIBRATE permission）

Notification.DEFAULT_SOUND 添加默认声音提醒

Notification.DEFAULT_LIGHTS 添加默认三色灯提醒

Notification.DEFAULT_ALL 添加默认以上3种全部提醒

表4-1 Notification默认配置项

除了setDefaults之外，还有几种方法可供选择：

setLights(Color.YELLOW, 300, 0) 单独设置呼吸灯，一般三种颜色：红，绿，蓝，经测试，小米支持黄色

setSound(url) 单独设置声音

setVibrate(new long[] { 100, 250, 100, 250, 100, 250 }) 单独设置震动

表4-2 Notification自定义配置项

另一个有趣的需求需要在这里特殊讨论以下。服务器推送了消息给客户端，但有可能由于网络波动，服务器的信息已经下发了，而客户端却没有收到这条信息。这就导致了另一个典型的问题：待推送信息丢失。如何避免这种问题呢？有一种策略是这样的：构建一个消息确认接收队列池（MessageSureAcceptQueuePool），这种池存在的意义在于延长服务器对信息成功推送的确认周期。

策略细节如下：

服务器向客户端下发消息，下发的消息放置在上文讨论的确认接受池中，客户端接收到消息后，成功处理完成，向服务器报告已经处理完成的信息编号。服务器接受到客户端报告的信息，在接受池中找到对应的消息，将消息从接受池中取出并标记其状态为已成功推送。对于那些未能到达客户端的消息，服务器构建计时器，发现在池中时间达到指定时间的对象后将其销毁。

对应的标记状态不只未推送与成功推送两种，任何架构都可以根据自己的需要配置自定义状态。这样的策略是所有双端通信状态缺失问题的通用解决方案，但其背后仍然存在缓冲池规模问题和计时器时长问题。任何一个游戏项目每个阶段的玩家访问量级、周期性访问变化都存在其内部规律且变动频繁的，刚上线的新游戏和已经运营数年的老游戏对缓冲池规模的需求相去甚远；爆发期的游戏与衰退期的游戏每日登陆负载需求相差巨大；游戏的类型为mmorpg与moba的游戏每一时间段内玩家的请求协议数和tcp连接数差距也很大。所有这些变数的掌握都需要建立在丰富的从业经验之上。

近年来，出现了一种弹性的潮汐架构平衡运维工具。核心原理是当某一模块的内存数量扩大到当前服务器的一个预估值一段时间后，服务器集群自动扩大该模块的内存数量;而当该模块内存消耗数量低于一个预估值一段时间后，服务器集群自动回收闲置的内存空间。这样的运维产品在各游戏公司颇受追捧。很多游戏小厂选择接入成熟的第三方技术平台来支撑这一需求。而主流的大制作厂商更多的出于成本考虑选择成立专门的运维工具部内部孵化这样的工具同时应用于自己旗下的多款游戏项目。

4.4 实现过程中出现的问题及解决过程

Android原本使用构建AsyncHttpResponseHandler类，提供onSuccess方法和onFailure方法进行通信成功或失败情况下的处理的方式进行与服务端内容的通信。这里简单介绍一下AsyncHttpResponseHandler类，这是一个Android提供的异步处理http返回对象response的API接口，其基于AsyncHttpClient架构设计，是异步http后台操作的解决方案之一。被广泛用于Android应用程序的http请求处理。在使用此类时，先构建对应的异步网络处理客户端实例AsyncHttpClient，再在其基础上触发对某个url的请求并返回AsyncHttpResponseHandler对象，其中就包括对这个url进程处理后服务器的写入返回内容responseWriter，当此次请求成功并返回正确的状态码200时触发onSuccess，而在失败的情况下则触发onFailure方法进行报错。一般情况下，客户端与服务器的通信使用json格式进行数据交互；在这里从responseWriter中获取到服务器的返回内容并以json字符串的方式接收数据格式并反序列化为对应的内存结构。

但在调试过程中发现请求处理的结果始终触发onFailure方法，报错数据请求失败，报错信息为http方案未注册。确认本地

idea需对http进行注册后才能如此调用，于是改用DemoHttp的默认http参数注册http方案并请求远端url。经此修改后，http请求得以正常进行连接并提供服务器返回的json序列，经反序列化后将解析出的信息存入内存并进入通知发送模块进行服务器推送消息的个性化处理。

在个性化处理的几种情况中，有一种经典的处理方案是以通知的形式弹出提示。在尝试实现此功能的过程中出现了图4-1中的提示信息。经过对很多文献资料的查阅发现在Android5.0以上的操作系统中进行通知推送时，需要提前设置好通道（channel），并提供与之唯一对应的通道标识（channelId），在通知推送的建造者模式下设置好待推送的通道方可正常实现推送逻辑。Android5.0提出通道这一概念主要是为了解决单个移动应用对多种不同推送样式的需求：通过设置channel的不同属性以达成不同的推送效果（例如不同的提示音；不同的icon图标；不同的通知点击意图；甚至是不同的layout界面布局）。

原本的通知推送并未设置channel，构造者模式中也自然没有配置对应的channelId，甚至连默认的系统级channel也没有选择，因此在Android5.0以上的系统无法完成通知。为了解决这一问题，在Service组件的onCreate中创建了channel并为builder对象设置了channelId后，弹出了正常的通知信息。

图4-1 错误信息提示

4.5 测试

测试过程中预先设置了如下测试case：

测试case 验收结果

Case1：开启http访问开关，服务端存在待推送信息，验证信息是否可以接受并显示，延迟不得超过5s。 Result1:服务端的带推送信息正常接收并显示，100次测试中未出现延迟超过5s的现象，平均延迟时间在3~4秒。

Case2：服务端存在待推送信息的情况下关闭http访问开关，验证信息在此情况下被阻隔 Result2:关闭http访问开关后信息被阻隔，服务端仍存在未推送的信息。

Case3：打开http访问开关后未发送的信息依次被接收并弹出提示。 Result3:打开http访问开关后未发送的信息依次被客户端接收并弹出提示。

Case4：服务端存在带推送下次情况下先关闭http访问开关再重新打开，观察信息是否如逾期一样先暂停接收再接收。(Error)Result4:出现问题，关闭http访问开关再重新打开后，信息接收停止。

Case5：服务端应用出现故障（意外中止）的情况下客户端的使用现象。 Result5:正常运行状态下突然中止服务端进程，客户端无特别现象，存在未推送信息时，信息下发中止，符合预期。

表4-3 预设测试case及验收结果

在以上的测试case中，Case4出现问题，经排查发现问题成因如下：

在Service中建立的Thread线程进行循环请求发送（每3秒发送一次http请求给服务端）。当http的访问开关被关闭后，循环发现flag值为false，在其中使用了return而非continue。

从程序设计角度讲这是一个微小的错误，但测试过程暴露了错误并给程序开发者修正它的机会，从这个小的事例上可以看到测试流程对应用程序开发的重要意义。

6. 第5章游戏信息推送在实际项目中的应用

总字数：3599

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第5章游戏信息推送在实际项目中的应用

5.1 转发与提示

游戏信息推送系统最适合于两客户端彼此通信服务器作中间转发平台的功能需求，在游戏项目之中，最典型的例子像是A玩家被B玩家攻打；这时由B玩家触发服务器请求，服务器处理到A玩家的业务逻辑后向A玩家推送一条消息通知其被攻打的信息。

图5-1 服务器处理转发信息过程

类似的，不仅仅是攻击这一种逻辑，包括赠送道具、组队、添加好友等业务逻辑通知都可以使用这一逻辑对客户端进行告知。而客户端接受到这一消息后所做出的行为也不仅仅是弹出一条通知这样简单，在玩家被攻打时客户端可以在玩家主城进行高亮提示；组队请求时客户端可以限制玩家只能对弹出的dialog窗口进行操作（接受还是拒绝）；赠送道具时客户端可以对获得的新道具进行高亮显示（如图5-2所示）。

图5-2 应用示例

5.2 信息同步

游戏行业区分客户端与服务器，双端对同一套玩家信息会维护两组不同的值，一般情况下为了防止客户端使用插件作弊，运算操作都放在服务端；而客户端维护的状态值是为了显示玩家当前的状态信息。这就涉及到某些情况下前后端运算结果不一致的情况。

下面举两个具体的例子：

在游戏中玩家的各种buff存在叠加效果，以出征上限为例，客户端运算后的出征上限为A（如图5-3所示出征上限为79000），服务器运算的出征上限为B。

图5-3 应用示例

显然客户端会对出征的士兵数量作直属拦截，正常情况下客户端是不会发超过A的军队的。在A<=B的情况下客户端的表现正常，不会暴露问题。但如果A>B，且玩家发兵数量C>B，那么这种情况下客户端通过校验而服务器校验失败，服务器就会弹出错误提示给客户端，玩家就无法正常发兵，这就出现了由于前后端buff运算不一致导致“卡流程”的现象。

在这种情况下，应用信息推送技术，在客户端打开UI的时候将服务器的出征上限B同步给客户端，即A=B，这样的信息同步保证了不会出现上述问题。

虚拟道具是网络游戏中频繁出现的字眼，在客户端服务器双端模式下，虚拟道具也是其共同维护的两套用户数据之一。那么如果出现前后端不一致的情况，大致有以下两种后果：1.客户端显示道具充足而服务器扣除失败（客户端道具数量大于服务器）；2.只使用了一个道具却提示扣除了多个道具（客户端道具数量小于服务器）。在这种情况下，多次对双端逻辑进行筛查是一个痛苦的过程，信息推送技术在这种情形下提出了优秀的解决方案。

观察参数中的isPush，这是用来标识是否对客户端进行推送的标识字段。

1. decItem(UserProfile userProfile, String goodsId, int num, LoggerUtil.GoodsUseType goodsUseType, boolean isPush)

当此字段为true则意味着需要通知客户端剩余的道具数量情况。也就是说当我们扣除了虚拟道具且不希望客户端对数据进行运算时，直接推送一条名为 ConstantsPushDefine.PUSH_DEL_ITEM的消息即可直接同步客户端与服务器的该道具剩余情况，也就避免了上述的双端信息不一致问题。

服务器对客户端的信息转发不仅停留在一来一往的协议式交互，还存在两种更密切也更频繁的高级信息同步转发形式：状态同步、帧同步。

在动画制作播出的过程中，一个概念成为关键帧：即电脑同步渲染的起点与终点。我们知道在动画制作过程中只需要绘制起始帧和终止帧，中间的过程帧可以由电脑内部生成。在游戏中存在类似的操作系统级原件，可以支持关键帧的同步，也就是说只有接收到了待播放的动画的终止关键帧，否则客户端不会显示游戏的下一步操作。

一般而言，竞技性游戏对信息转发同步的要求比较高，会采用帧同步级别的信息同步转发，这也就是为什么网络环境不好时竞技类游戏总是卡在一幅画面不动。

状态同步相对于帧同步的要求降低了很多，往往只是在客户端未接收到状态解除的指令之前保持状态的不变。（如一些mmorpg中的眩晕、致盲、迷乱等实时状态）

而上述例子中的信息转发与提示只是最简单的单数据同步。

5.3 离线推送

离线推送是信息推送系统最直接解决的问题，在玩家不在线的情况下向其发送通知以引起玩家对应用程序的关注度，如图5-4所示。

在一款游戏中类似的通知数量往往非常惊人，这也在另一方面反映了离线推送的重要性。

图5-4 应用示例

类似的通知和提示非常多，下表是一个配置文件中dialogtext文本的节选。

153928=棒球176256=拍卖行出现珍贵道具了，快来拍卖行竞拍吧！176257=一口价176258=安全、便捷、自由的交易平台，记住拍卖三字诀：快、准、狠！176259=输入错误，含有非法字符，请输入数字！176260=您的竞价被超过，前往拍卖行查看！176261=竞拍物品已被拍走，前往拍卖行查看！176262=拍卖行已经开启，点击前往参加竞拍！176263=请输入您的竞拍加价！176264=竞价176265=当前价格176266=加价错误!加价不能小于系统加价{0}101000=赠送101001=您搜索的玩家不存在或不在本王国中101002=是否赠送 {0} 给 {1}？101003=礼包将以邮件形式发送给对方！赠送礼包不会影响您自身购买礼包。101004=赠送成功！101005=我向你赠送 {0}101006=亲爱的领主大人，您的朋友{0}向您赠送了{1}（价值{2}）。礼包内容如下，记得要感谢您朋友的慷慨哦！101007=来自{0}的礼物101008=已赠送101009=您已向{0}赠送了礼包，无法再次赠送相同礼包给同一个领主101010=联盟发展101011=超值热卖101012=赠送物品需要消耗1个中级火漆信封，您现在有{0}个中级火漆信封！\n注意：你最多只能选择15个物品。101013=赠送物品需要消耗1个联盟火漆信封，您现在有{0}个联盟火漆信封！\n注意：你需要有足够的物品才能赠送。101014=领主大人，你身上的物品不足以赠送这么多盟友！101015=伟大俄罗斯101016=领主大人，您还未加入联盟，无法使用联盟火漆信封！101017=.101018=购买此礼包可获得门票一张！101019=诚挚邀请你参加我们的派对！101020=《列王的纷争Clash of Kings》VIP全台豪华派对即将开始，请领取门票并点下列网址，前往报名唷！\n小天兵提醒，透过报名参与派对活动者，入场就送精美礼物，派对活动应有尽有，让你畅玩整个夏天！详情请看官方中文交流区～101021=注意：奖励务必当日领取，次日不累计！101022=全台豪华派对礼包101023=码头101024=剩余天数101025=超值首充101026=点数充值101027=争霸远古战场101028=资源礼包101029=一周年礼包101030=时间礼包101090=加速时间：{0}（免费时间：{1}）101091=母亲节快乐101092=Double the Joy101093=购买成功101094=您获得以下物品101095=感谢领主大人101096=奖励已发送101097=点击激活101098=您购买的礼包已送到，请您确认101099=领主大人，您购买了以下物品，请核对。感谢您对《列王的纷争》的支持！101100=距累计充值大反馈结束还有101101=去充值101102=领主大人，无法给不同王国的好友发送火漆信101103=粮食礼包101104=木头礼包101105=铁礼包101106=秘银礼包101107=新手累积充值回馈将于{0}后结束101108=免费金币！101109=Smartone好礼相赠，祝您游戏愉快！101110=金马车

礼包101111=领主大人，您购买的物品已到账，如需查看详细列表，推荐您更新到最新版本。感谢您对《列王的纷争》的支持！101112=SVIP礼包101113=快速征兵礼包101114=战后恢复礼包101115=橙色装备101116=金色装备101117=办年货101118=网络故障请稍后重试101119=新春礼包101120=神龙皮肤礼包101121=快速成长装备宝箱101122=该宝箱内含全套10级蓝色装备101123=农历新年101124=猴年大吉101125=城建专享101126=造兵专享

表5-1 dialog配置示例

在程序中，客户端构造一个dialog-String映射，当需要推送消息时，客户端推送的时dialog编号，而具体的String由配置中的信息来决定。这样的设计对于多语言全球化运营的游戏是最适合不过的，因为程序可以以最小的成本支持一个地区的文本翻译工作。

7. 第6章总结与展望

总字数：3807

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第6章总结与展望

本设计提供了一种信息推送需求的解决方案，应用Android的Service组件维护Android后台与服务端的通信连接，实时访问查看服务端消息队列MessageQueue中是否存在待推送的消息Message；服务端确认访问有效且存在待推送的消息Message并下发消息给客户端，再由客户端根据服务端下发的消息进行个性化处理。简单的框架和设计展示了这种信息推送技术的原理，并介绍了目前在游戏行业中该技术的不同应用场景。

通过对推送系统的设计和实现，我认识到设计模式对编程的重要性：一个良好的设计模式可以提升代码可读性，理清编程思路，极大简化后续的研发过程；而糟糕的设计模式会增加模块间的耦合，使业务逻辑和框架融合在一起，后续的可维护性和可拓展性几乎荡然无存。

推送系统试图打破全球Web的标准拉式交互模式，并简化信息发现和传播过程。推出简单的推送系统用于提供与电视，广播或印刷机类似的服务，但技术能力通过互联网得到增强。基本思想是一样的：用户查询可用信息渠道的“渠道指南”，根据用户的兴趣订阅其中的一些信息，然后不断获取信息，即信息获取过程从用户发起的拉取变为提供方的推送。与其强迫用户重复询问信息或检查新信息是否可用，不如用户只订阅一次并继续接收。

推送系统的需求源于几个原因。最重要的是万维网基于简单的请求/回复方案，要求用户在他/她需要信息时发出请求。这实施了“同步”交互方案，而推送系统允许异步信息分发：理想情况下，只要用户选择的信息变得可用，它就得到分发。在推送系统弥补以下一个问题之后，达成公认的理想主义假设，万维网的几个关键问题可以得到解决：

定位目前查找信息是用户的主要问题。即使搜索引擎再先进，其所搜索到的信息质量仍然与用户的知识和技能（关键词选择，直觉）相关。推送技术有望通过信息渠道和订阅的概念将信息提供者的角色转移到信息提供者身上，以此弥补这一点。

聚焦使用推送系统时，用户需要明确说明他/她的偏好。因此，提供有重点的信息很容易。

定制用户可以限制数据和其在交付之前要应用的属性的要求，例如数据格式，优先级，关键字等。

即时一旦数据可用，就可以主动发布数据。信息提供者可能会使陈旧的数据失效。

配置用户不仅可以定制数据及其属性，还可以定制提供者。提供者可以控制用户何时看到哪些信息。这个决定可以基于订阅信息，兴趣分析等提供的用户配置文件，并且允许将信息拖到用户的配置文件当中。它也是一个强大的广告工具。

流量减少推送系统有助于减少网络流量。传统的pull技术尝试查找信息可能会导致大量流量。由于推送系统可以将数据定制到用户的配置文件，因此流量可能会大大减少。另外，适当的传输基础架构可以进一步减少网络带宽，例如通过使用中继器。

许多具有不同属性的产品都被归纳为“推”的概念。它们在结构，组件，灵活性和交互模式方面差异很大。它们的主要共性是自动传递信息的抽象模型。由于缺乏参考架构，可用推送系统的范围难以评估和比较。本文提出了推送系统的参考架构，以更好地理解推送系统的概念，并支持对现有产品进行评估和分类。

信息推送技术在服务器客户端通信架构里拥有举足轻重的地位，其亟待解决的问题一度存在了相当长的时间。我国信息推送技术相关文章最早出现在1999年《计算机系统应用》第五期，沈艺的《信息推送技术及其应用》[4]，她在文中指出了已有的用户通过Internet主动搜索网上服务器信息的方式在 Internet迅速发展信息膨胀的情况下，查找信息的困难性和高负荷、不准确等问题，并介绍了网络公司使用信息推送技术建立网络广播站，智能代理从网络取回用户所需信息，分类并设计预存信息提供给用户，并提供实时更新维护。而客户机的操作则是当有新信息可用则自动弹出，告知用户。 此时大部分客户机均使用预约模式，按预定义时间间隔询问信息服务器，以确认当前是否有新的信息内容可提供。部分客户机软件可对接口做出定制化处理，使用不同的方式通知用户：释放声音、 发送email、显示符号或弹出某应用、弹窗、激活屏幕保护程序、显示通知单或墙纸。这时的信息推送客户端的信息载体是台式主机上的应用程序软件，当软件被关闭的时候，所对应的功能也都无法生效。所以，这时的信息推送技术无论在架构上还是定义上都与我们今天所说的信息推送技术有较大差别，但可以确定的一点是，这就是我们今天所说的信息推送系统的结构雏形。

在2010年，G3网络的普及为移动互联网的发展提供了基床。2012年《计算机工程与设计》第八期 ，廖轶宸的《基于移

动网络的混合型信息推送系统的研究》[5]中指出：传统的基于固定网络的、网络信号较为稳定的信息推送系统并不适用于通信质量较差G3网络，这时的信息推送系统采取了对应的措施和机制，提供不同的信息推送方式与跟踪推送结果方案，确保信息的正确推送。不仅如此 ，在推送失败的情况下可即时通知客户端推送失败。同时，考虑到这时移动终端的硬件条件较为严苛，在信息推送前，要对信息进行一定的预处理，降低传输失败的可能。再从数据挖掘的角度提升信息推送的准确度。依靠这样的策略，使信息推送系统满足移动网络的使用条件。

今天的移动网络已经几近成熟，信息推送的方式减少了一定的限制。信息推送客户端的主流信息载体也从台式主机换成了智能手机、平板电脑等移动设备。在这样的情况下，一般应用程序都会同时开启后台监听进程，即便已经关闭了应用程序，来自服务器的推送也可以由后台程序接收并展示 。当用户点击推送出的弹窗后则自动启动应用程序并显示其待体现的操作（声音、弹窗、甚至是客户端内部的一些特殊处理条件）。在国外，google push service包揽了几乎所有Android手机的推送机制，应用程序研发商将google提供的接口对接到自己的应用程序后，即可实现所有Android平台手机的信息推送系统；但在国内，各种Android平台提供不同的调用接口，应用程序研发商必须在服务器对接了某品牌的Android平台提供的接口后才能对该品牌的手机推送消息，这对应用程序研发商来说提高了Android版本的软件研发成本，因此，国内的环境催生出一批第三方推送平台 ，应用程序研发商将希望推送的信息发送到第三方平台，第三方平台再将信息转发给各个不同的 Android平台并收取一定的利益，其商业化的模式已经成熟，在这种情况下，寡头化会成为商业上应用此技术的一种趋势。

随着移动互联网时代的到来，信息通知提醒或是在不运行手机应用时接收服务端通知的现象越来越广泛和频繁地出现在我们眼前，有些信息大有用处，比如：网约车软件约到的车到达上车地点；有些信息有利于流量曝光，比如：个税改革等重磅新闻的提醒；有些信息有利于商业发展，比如 “双十一”的重大折扣，还有些信息有利于应用软件更多地获得用户的关注，游戏类、旅游类、音乐类应用大多如此。

近两年来，中国大陆互联网圈掀起了一阵人工智能、大数据风潮，大数据其对用户数据的收集甚至使程序了解了用户的习惯，人工智能甚至可以推断出用户下一次搜索的大致时间和搜索范围。信息推送系统与这些新技术的结合会更好地服务大众并带来更多的创新。笔者认为，信息推送系统有更多更好地结合人工智能和大数据的趋势，精准性、预先性在信息推送系统中将具有愈加重要的地位，同时，个性化推送将成为一种成本更高的新型广告模式。

致谢

我要最先感谢的是我的毕业设计直接辅导老师杨滨老师从开题到答辩一路来对我的悉心指导。

感谢在北京智明互动科技有限公司实习工作期间我的校外指导老师，同时也是我的学长张文国前辈对我一直以来的照顾和对我产生的诸多疑问的解答，在实习过程中能遇到这样一位前辈的我无疑是幸运的。

感谢王喆老师在我遇到困难时对我的关心和建议，考虑我的自身情况把我分配给杨滨老师指导，这对我的帮助非常之大。

感谢我开发过程中使用的工具的创造者，它们之中有planetB(在线美化程序代码工具)、有idea（高效编译器）、jdk、tomcat、sdk等等。在今后的工作生活中我也会秉持开源精神为互联网的发展贡献出自己的一份力量。

特别感谢大学四年以来我所有的老师和同学，这一段时间里我收获了知识和友谊，基于此我才有能力完成这篇毕业设计。

[1]CSDN博主可乐百事.Android四大组件之Activity[EB/OL].https://blog.csdn.net/zhengweilxl/article/details/51458391.html

[2]博客园博主生命壹号.Android组件系列----Android Service组件深入解析[EB/OL].https://www.cnblogs.com/smyhvae/p/4070518.html

[3]CSDN博主潘浩 .干货！Android的保活的两种解决方案[EB/OL].https://blog.csdn.net/pan861190079/article/details/72773549

[4]沈艺. 信息推送技术及其应用[J]. 计算机系统应用, 1999 (5): 26-27.

[5]廖轶宸. 基于移动网络的混合型信息推送系统的研究[J]. 计算机工程与设计, 2012, 33(8): 3268-3272.