河海大學

毕 业 设 计（论 文）

基于 Android 的即时通讯软件设计和实现

专业年级 计算机科学与技术 09 级

学　　号 0906010401

姓　　名 李　艳

指导教师 娄渊胜

评 阅 人

2013 年 6 月

中国　南京

河 海 大 学

本科毕业设计（论文）任务书

Ⅰ、毕业设计（论文）题目：

基于 Android 的即时通讯软件设计和实现

Ⅱ、毕业设计（论文）工作内容（从综合运用知识、研究方案的设计、研究方法和手段的运用、应用文献资料、数据分析处理、图纸质量、技术或观点创新等方面详细说明）：

Android 是一种基于 Linux 的自由及开放源代码的操作系统，主要使用于便携设备，如智能手机和平板电脑，而即时通讯（Instant Messaging）是目前 Internet 上最为普遍的通讯方式之一。随着智能手机的不断流行，基于 Android 的软件也越来越多。本次论文的题目是基于 Android 的即时通讯软件设计和实现。

首先，需要了解 Android 系统知识，包括 Android 系统框架、应用程序的生命周期和应用组件以及 Intent 类和数据处理等。其次，了解即时通讯的发展状况，即时通讯协议大多数是基于开源的 XMPP 协议，服务器端采用开源的 Openfire 服务器，通过 API 框架的 Smack 实现服务端和客户端的通信。然后，对 XMPP 协议、Openfire 服务器和 Smack API 进行深入了解。在理解了 Android 以及相关技术知识的基础上，研究和设计即时通讯系统，采用客户端/服务器的体系架构。在其服务端采用 MySQL 数据库存储数据；在其客户端，由于数据量较少，因此采用 Android 自身的轻量数据存储机制 SharedPreferences。

本系统采用客户端(C)／服务端(S)架构的体系结构，具有服务器端和客户端，采用开源的 XMPP 协议作为通讯协议。客户端是基于 Android 平台进行开发。通过无线网络与 Internet 网络建立连接，通过服务器实现与 PC 机客户端之间的即时通讯。客户端负责初始化通信过程。进行即时通讯时，由客户端负责向服务器发起创建连接请求。服务端采用开源的 Openfire 服务器，允许多个客户端同时登录并且并发的连接到一个服务器上。服务器对每个客户端的连接进行认证，对认证通过的客户端创建会话，客户端与服务器端之间的通讯就在该会话的上下文中进行。

本即时通讯系统的功能包括用户的注册和登录、接收与发送消息或是附件、更改用户状态和密码、添加好友以及注销，额外的功能包括更换系统界面皮肤。

Ⅲ、进度安排：

2012 年 12 月 8 日至 2013 年 2 月 28 日，熟悉所研究课题的基本情况和涉及到的相关技术，阅读相关的文献资料，提出初步思路和总体框架。

2013 年 3 月 1 日至 2013 年 3 月 20 日，熟悉编程环境，掌握编程工具应用。

2013 年 3 月 20 日至 2013 年 5 月 20 日，编写和调试程序。

2013 年 5 月 20 日至 2013 年 6 月 10 日，写毕业论文，准备答辩。

2013 年 6 月 8 日至 2013 年 6 月 9 日，论文评阅，毕业答辩。

Ⅳ、主要参考资料：

[1] Ed Burnette 著. 田俊静,张波,黄湘情 等译. Android 基础教程(第 3 版)[M], 2011-6

[2] 吴亚峰 索依娜 等著. Android 核心技术与实例讲解[M]. 电子工业出版社, 2011-6

[3] 陈钊. Android 程序主要组成部分概述[J]. 中国新技术新产品. 2011(17):42

[4] 李宁 著. Android 开发完全讲义(第二版)[M]. 水利水电出版社. 2012

[5] P Saint-Andre Ed. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence[EB/OL] http://xmpp.org/internet-drafts/draft-saintandre-rfc3921bis-07.html, October 24, 2008

[6] 庞怡 许洪光 姜媛. 即时通讯工具现状及发展趋势分析[J]. 科技情报开发与经济，2006(10):169-170

[7] 剧忻. 基于 MINA 开发高性能网络应用程序——以实现 XMPP 协议 Openfire 3.3.3 为例[J]. 重庆工学院学报（自然科学版）. 2008, 22(10):121-125

[8] Jack Moffitt 著. 杨明军 译. XMPP 高级编程——使用 JavaScript 和 jQuery[M]. 清华大学出版社, 2011-6

[9] 卡尔佛特 多纳霍 著. 周恒民 译. Java TCP/IP Socket 编程(原书第二版)[M]. 机械工业出版社, 2009-1

[10] 张彦 夏清国. Jabber/XMPP 技术的研究与应用[J]. 科学技术与工程. 2007, 7(6)

[11] Jason Kichten 著, 刘建华译. 用基于 XML 的即时消息开发 Jabber[[EB/OL]. http://www.bitscn.com/pdb/dotnet/200701/88917.html

[12] 潘凤 王华军 苗放 李刚. 基于 XMPP 协议和 Openfire 的即时通讯系统的开发[J]. 计算机时代. 2008(3)

[13] http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/

[14] 马志强. 基于 Android 平台即使通信系统的设计与实现[D]. 北京交通大学，2009

[15] Peter S A.XMPP Instant Messaging and Presenee.RFC 3921 [E],2004

[16] Pankaj Jalote 著.罗飞 邵凌霜 等译.软件工程导论[M].清华大学出版社, 2012

[17] Wei-Meng Lee 著. 何晨光 李洪刚译. Android 编程入门经典[M]. 清华大学出版社, 2012-4

[18] 张海燕. Java 多线程技术在手机互联网中的应用[J]. 农业网络信息, 2008(3): 97-98

指导教师：_____ ， _____年_____月_____日

学生姓名：_____ ，专业年级：_____

系负责人签字：_____ ， _____年_____月_____日

摘要

随着移动通信与 Internet 的飞速发展以及相互融合，GPRS 和 WIFI 使智能手机连通 Internet 成为现实，移动用户从而可以享受到 Internet 提供的服务。同时，智能手机的普及以及性能的提升也为即时通讯系统从传统的 PC 机到手机的移植提供了良好的条件。现在，智能手机的用户逐渐在扩大，基于手机操作系统的即时通讯软件的需求也越来越多。

Android 是一种基于 Linux 的自由及开放源代码的操作系统，主要使用于便携设备，如智能手机和平板电脑。XMPP 是基于 XML 的开源的即时通讯协议，因此基于 XMPP 协议和 Android 平台开发即时通讯系统具有很好的应用前景。本文是基于 Android 的即时通讯软件的设计与实现。

本文主要工作如下：

1. 简述了即时通讯的研究背景。介绍了 Android 的基本知识，包括 Android 的特征、体系架构、应用程序的生命周期和四大应用组件，接着又阐述了基于 Android 即时通讯研究的相关技术，进一步叙述了即时通讯的服务器 Openfire、应用的 jar 包 Smack 和以之为基础的 XMPP 协议。
2. 按照软件设计流程进行系统开发，首先在需求分析中给出系统应满足何种要求，然后根据此类要求绘制系统总体流程图，再结合 Openfire 服务器、Smack 包和数据库绘制体系架构图，最后进行系统设计。在系统设计阶段先是数据设计，即在服务器端使用 MySQL 数据库和 Android 客户端使用轻量数据存储机制 SharedPreferences，其次是界面设计，这里使用 Intent 机制实现界面跳转，再者是各个模块的具体功能设计，包括使用的方法和编码的安排。
3. 系统设计完成后，就是部署运行本系统并执行测试，将测试结果以图片的形式在文章中展现出来。

关键词：Android，即时通讯，Openfire，Smack，XMPP 协议

Abstract

With the rapid development and integration of mobile communication and Internet, GPRS and WIFI make it become reality that the smart phone access to the Internet and mobile users can enjoy services provided by Internet. With the constant popularity of mobile phones as well as the continuous performance improvement，it is time for instant messaging to transplant from the traditional PC to Mobile. Additionally, with the users that use smart phone increasing, the requirements of Instant Messing are becoming more and more.

Android is an operating system of Linux based on free and open source code, mainly used in portable devices, such as smart mobile phone and tablet computer. XMPP which is opened source is an instant communication protocol base on XML. Therefore, with XMPP protocol and Android platform, the development of instant messaging system has a good prospect. This paper is a study for the instant communication of Android system.

This paper contains:

1. Introducing the background of this study and the basic knowledge of Android that contains the Android system structure, system architecture, the life cycle and four application components. Third, this paper explains the related technology of the instant messaging based on Android. It describes Openfire which is used as server, the jar package Smack and XMPP protocol.
2. The system is designed according to the software design process. First, I should find requirements that the system should meet in the demand analysis. Then draw system flow chart. After that, draw system architecture diagram with the Openfire server, Smack and database. In the period of designing of this system, designing data table must be done firstly. The database of server is MySQL and the database of client is SharedPreferences. After that, the skip of UI is realized with Intent. Then what is done is detail designing of every part, which contains the method that is used and coding.
3. Running and testing the system and record the test results.

Key Word: Android, Instant Message, Openfire, Smack, XMPP protocol.

目　录

第一章　绪论 1
1.1　课题背景 1
1.2　本文所做的主要工作 1
1.3　论文的组织结构 2
第二章　相关技术 3
2.1　Android 系统简介 3
2.1.1　Android 系统架构 3
2.1.2　应用程序框架 4
2.1.3　应用程序的生命周期 4
2.1.4　Android 的应用组件 5
2.2　即时通讯协议 XMPP 协议 7
2.2.1　XMPP 协议网络架构 8
2.2.2　XMPP 协议的地址格式 9
2.2.3　XMPP 协议消息格式 9
2.2.4　XMPP 协议优点 9
2.3　Openfire 服务器 10
2.3.1　Openfire 优点 10
2.3.2　Openfire 通信 11
2.4　Smack 包 11
2.5　本章小结 12
第三章　系统的设计与实现 13
3.1　系统的需求分析 13
3.2　系统总体流程设计 14
3.3　系统的体系结构设计 15
3.4　系统的数据库设计 16
3.5　系统的界面设计 17
3.5.1　主界面设计 18
3.5.2　好友界面设计 18
3.5.3　设置界面设计 19
3.5.4　页面交互设计 19
3.6　系统的功能设计与实现 21
3.6.1　登录/注销功能 21
3.6.2　注册功能 23
3.6.3　获取好友列表功能 24
3.6.4　用户状态功能 25
3.6.5　会话功能 25
3.6.6　更改密码功能 29
3.6.7　添加好友功能 29
3.6.8　附件浏览功能 30
3.6.9　更改界面皮肤功能 30

3.7 本章小结 31
第四章 系统的部署与展示 32
4.1 系统的部署 32
4.2 系统的展示 32
4.2.1 系统的初始状态 32
4.2.2 注册 33
4.2.3 登录 34
4.2.4 会话 34
4.2.5 更改用户状态 35
4.2.6 更改密码 35
4.2.7 添加好友 36
4.2.8 更改界面皮肤 37
4.2.9 用户注销 37
4.3 本章小结 37
第五章 结语 38
参考文献 39
致谢 40
附录 41

第一章 绪论

1.1 课题背景

随着移动通信领域的发展以及互联网逐步向移动终端的普及，用户和网络对移动终端的要求也越来越高，而由于 PalmOS、Symbian 等手机平台过于封闭，用户的需求不能得到很好的满足，因此一个开放性很强的平台成为市场迫的切需要。Android 是一种基于 Linux 的自由及开放源代码的操作系统，Android 使用刚刚在编译的 Dalvik 虚拟机运行 Dalvik 字节码，通常是从 Java 字节码翻译，主要应用于移动设备，如智能手机和平板电脑，由 Google 公司和开放手机联盟领导及开发。尚未有统一中文名称，中国大陆地区较多人使用“安卓”或“安致”。

Android 操作系统最初由 Andy Rubin 团队开发，主要支持手机，2005 年 8 月由 Google 收购注资。2007 年 11 月，Google 与 84 家硬件制造商、软件开发商及电信营运商组建开放手机联盟共同研发改良 Android 系统。随后 Google 以 Apache 开源许可证的授权方式，发布了 Android 的源代码。第一部 Android 智能手机发布于 2008 年 10 月。Android 逐渐扩展到平板电脑及其他领域上，如电视、数码相机、游戏机等。

即时通讯(Instant Messaging，简称 IM)是一个终端服务，允许两人或多人使用网络即时地传递文字信息、档案、语音与视频交流。即时通讯根据装载的对象分为 PC 即时通讯与手机即时通讯，根据使用用途又可分为网站即时通讯和企业即时通讯；视频、网站即时通讯如 QQ、MSN 等应用形式，手机即时通讯如短信。大部分的即时通讯提供的功能有在线提醒、显示好友名单、显示好友是否在线、能否与好友进行交谈等。与 e-mail 的不同，即时通讯的交谈是即时的。即时通讯在近几年发展特别迅速，其功能也日益丰富，由 PC 客户端发展到移动客户端，渐渐发展成一个综合化信息平台，国内市场上知名的即时通讯工具有微信、QQ、中国移动飞信等。

1.2 本文所做的主要工作

本文主要工作如下：

1）通过阅读各种关于 Android 和即时通讯的相关书籍、期刊、文献等资料，了解 Android 的基础知识与即时通讯的基础知识；

2）熟悉掌握基于 Android 操作系统的各种应用程序开发，以及掌握即时通讯软件的开发；

3）在熟悉两种开发之后，运用相关技术设计与实现基于 Android 的即时通讯软件，该软件要求的功能包括用户的登录与注销、用户注册、添加好友、接收/发送消息与附件和更改用户密码等功能。

1.3 论文的组织结构

本文首先分析了系统的研究背景，介绍了当前 Android 操作系统、即时通讯软件的发展现状。在摘要中简述了本论文的主要内容。

第一章给出了基于 Android 即时通讯研究的背景，包括 Android 背景和即时通讯的背景。

第二章具体介绍了 Android 系统的基础知识和其他基于 Android 即时通讯软件的设计的相关技术知识。Android 系统的基础知识包括它的系统架构，Activity 运行的生命周期。Android 包含的四大组件为活动（Activity）、服务（Server）、广播接收器（BroadcastReceiver)和内容供应商（ContentProvider）。基于 Android 的即时通讯研究的相关技术包括 XMPP 协议、Openfire 服务器和 Smack 包。XMPP 协议是使用 XML 作为消息传递中介的发送接收处理消息的协议。Openfire 是采用 Java 开发的开源的实时协作服务器，它基于 XMPP 协议。Smack 是一个 XMPP 协议的 Java 实现，提供一套可扩展的 API，是客户端与服务器端传递消息的媒介。

第三章介绍了 Android 端的即时通讯软件的设计，首先给出了需求设计，据此得出系统的总体流程图，将系统划分成模块，然后结合 Openfire 与 Smack 画出系统的体系架构图，展示系统与服务器间的交互。之后主要讲解了客户端的数据设计，采用 SharedPreferences 保存数据；在界面设计部分，简单介绍了几个主要界面的设计以及各个界面的交互流程。最后在功能设计部分，详述了各个功能的具体设计实现，采用哪些类的哪些方法，同时给出相应的流程图和代码以便加深理解。

第四章是系统的部署与展示，从注册、登录、会话、更改用户状态、更改密码、添加好友和更改系统界面皮肤等方面进行了展示。

第五章则是结语，总结了本文的主要研究内容和研究成果，还包括一些有待改进的地方。

第二章 相关技术

2.1 Android 系统简介

2.1.1 Android 系统架构

Android 的系统架构和其操作系统一样，采用了分层的架构如图 2.1 所示。

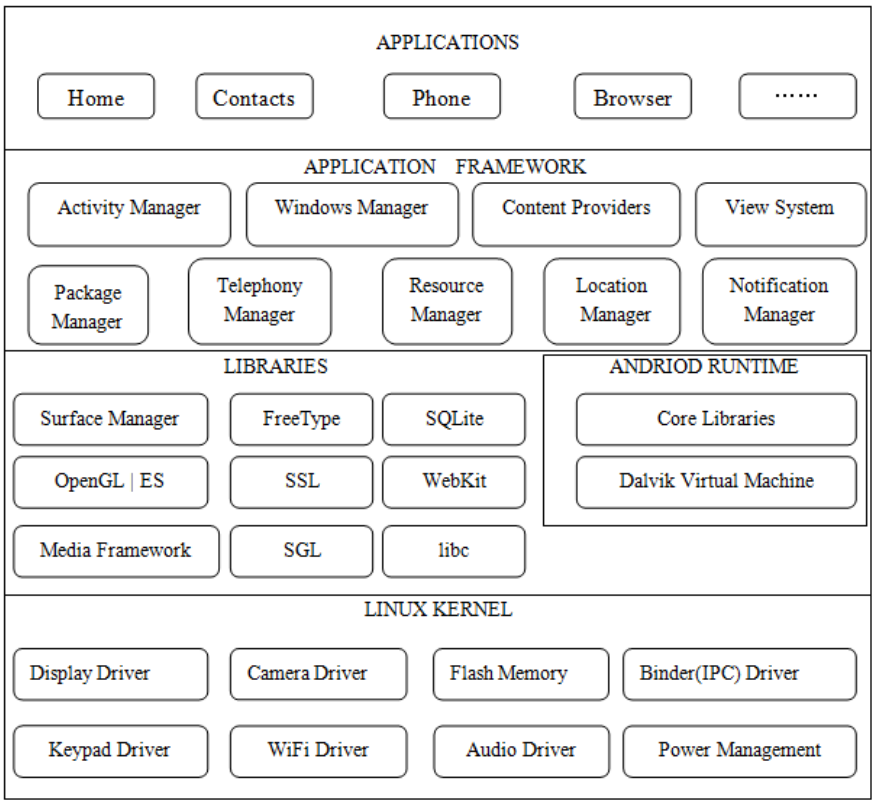

图 2.1 Android 系统架构图

Android 操作系统大致可以在 4 个主要层面上分为以下 5 个部分[1]：

- Linux 内核（Linux Kernel）——这是 Android 所基于的核心。这一层包括了一个 Android 设备的各种硬件组件的所有底层设备驱动程序。
- 库（Libraries）——包括了提供 Android 操作系统的主要功能的全部代码。例如，SQLite 库提供了支持应用程序进行数据存储的数据库。Webkit 库为浏览 Web 提供了众多功能。
- Android 运行时（Android Runtime）——它与库同处一层，提供了一组核心库，可以使开发人员使用 Java 编程语言来写 Android 应用程序。Android 运行时还包括 Dalvik 虚拟机，这使得每个 Android 应用程序都在它自己的进程中运行，都拥有一个自己的 Dalvik 虚拟机实例（Android 应用程

序被编译成 Dalvik 可执行文件）。Dalvik 是特别为 Android 设计，并为内存和 CPU 受限的电池供电的移动设备进行过优化的专门的虚拟机。

- 应用程序框架（Application Framework）——对应用程序开发人员公开了 Android 操作系统的各种功能，使他们可以在应用程序中使用这些功能。
- 应用程序（Applications）——在这个最顶层中，可以找到 Android 设备自带的应用程序（例如电话、联系人、浏览器等），以及可以从 Android Market 应用程序商店下载和安装的应用程序。开发人员所写的应用程序都处于这一层。

2.1.2 应用程序框架

Android 使新颖的和开发极其丰富的应用程序成为可能，通过提供一个开放的开发平台。开发人员可以自由地访问位置信息、使用设备的硬件、设置闹钟、运行后台服务、添加状态栏的提示等等。

同时，如同使用核心应用程序一样，开发人员可以随意地使用框架的 API 来实现自己的功能，应用程序框架对组件之间的重用做了简化。在 Android 平台下，任何应用程序都可以发布自己的功能模块，其他的应用程序依据一定的安全限制就可以无阻碍地访问这些模块。同时，这种机制还允许用户随意更换组件。

2.1.3 应用程序的生命周期

应用程序进程从创建到结束的全过程便是应用程序的生命周期。与其他系统不同的是，Android 应用程序的生命周期是不受进程自身控制的，而是由 Android 系统来决定的。一般情况下，Android 系统会根据应用程序对用户的重要性及当前系统的负载来决定生命周期的长短。

Android 系统将所有的进程大致分为以下 5 类进行管理[2]。

1、前台进程

前台进程，即当前正在前台运行的进程，说明用户当前正通过该进程与系统进行交互，所以该进程为最重要的进程，除非系统的内存已经到不堪重负的情况，否则系统是不会将该进程中止的。

2、可见进程

可见进程一般还是显示在屏幕中，但是用户并没有直接与之进行交互。例如某个应用程序运行时，根据用户的操作正在显示某个对话框，此时对话框后面的进程便可视为可见进程。该进程对用户来说同样是非常重要的进程，除非为了保证前台进程的正常运行，否则 Android 系统一般不会将该进程中止的。

3、服务进程

服务进程便是拥有 Server 的进程，该进程一般是在后台为用户提供服务的，例如音乐播放器的播放、后台的任务管理等。一般情况下，Android 系统是不会将其中断的，除非系统的内存已经达到崩溃的边缘，必须通过释放该进程才能保证前台进程的正常运行，才可能将其中止。

4、后台进程

该进程一般对用户的作用不大，缺少该进程并不会影响用户对系统的体验。所以如果系统需要中止某个进程才能保证系统正常运行，那么有非常大的几率将该进程中止。

5、空进程

空进程是对用户没有任何作用的进程，该进程一般是为了缓存机制服务的，当系统需要中止某个进程以保证系统的正常服务时，会首先将该进程中止。

2.1.4 Android 的应用组件

1、Activity

Android 中，Activity 主要用于表现功能，是所有程序的根本，所有程序的流程都运行在 Activity 之中。在 Android 的程序当中，Activity 一般代表手机屏幕的一屏。一般一个 Android 应用是由多个 Activity 组成的，这多个 Activity 之间可以进行相互跳转，例如：按下一个 Button 按钮后，可能会跳转到其他的 Activity。和网页跳转稍微有些不一样的是，Activity 之间的跳转可能有返回值，例如：从 Activity A 跳转到 Activity B，那么当 Activity B 运行结束的时候，有可能会给 Activity A 一个返回值。当打开一个新的屏幕时，之前一个屏幕会被置为暂停状态，并且压入历史堆栈中。用户可以通过回退操作返回到以前打开过的屏幕。可以选择性的移除一些没有必要保留的屏幕，因为 Android 会把每个应用从开始到当前的每个屏幕保存在堆栈中[3]。Activity 的生命周期如图 2.2 所示。

Activity 的生命周期主要包含三个状态：

1） 运行态

处于运行态的 Activity 拥有焦点，正在与用户进行交互，该状态的 Activity 可以为用户提供信息并接受用户的事件响应。

2） 暂停态

处于暂停态的 Activity 失去焦点，一般被运行态的 Activity 所替代，当前台显示的 Activity 不是全屏时，可以看见此状态下的 Activity。

3） 停止态

处于停止态的 Activity 没有焦点，并且是不可见的，系统随时可以将其释放掉。

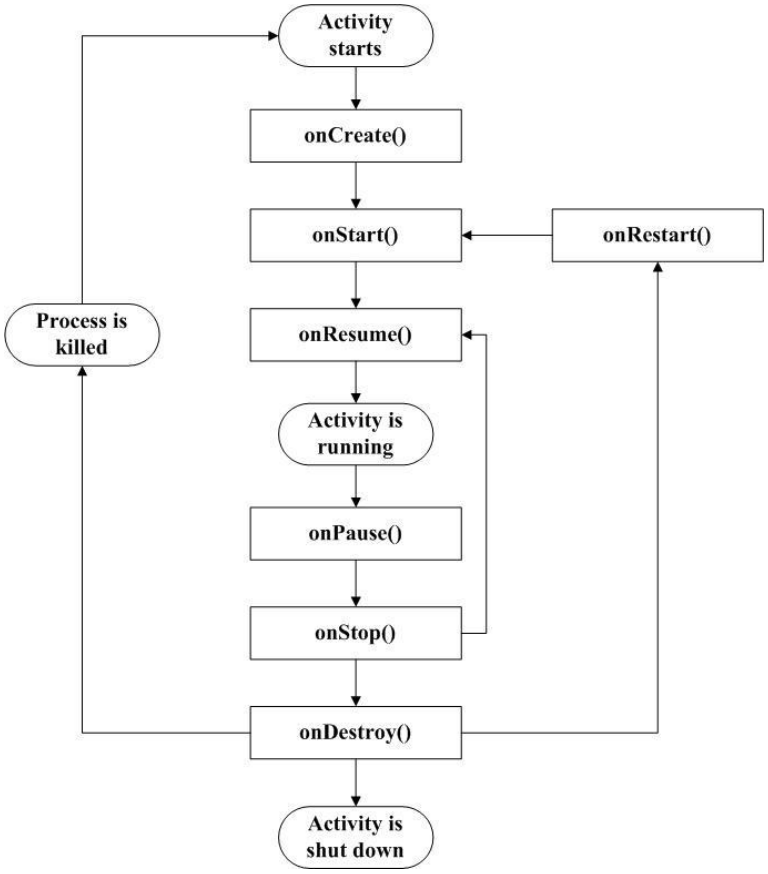

图 2.2　Activity 的生命周期图

2、Service

Service 是 android 系统中的一种组件，它时后台运行服务[4]，不提供界面呈现，但可以和其他组件进行交互。Service 是没有界面的长生命周期的代码，也是一种程序，可以运行很长时间，但却没有用户界面。Service 可以在多场合的应用中使用，比如检测 SD 卡上文件的变化，比如播放多媒体的时候用户启动了其他的 Activity，这个时候程序要在后台继续播放，再比如在后台记录地理信息位置的改变等等。

开启 Service 有两种方式：

1) startService 方式启动

当 Activity 调用 startService 方法启动 Service 时，会依次调用 onCreate 和 onStart 方法来启动 Service，而当调用 stopService 方法结束 Service 时，又会调用 onDestroy 方法结束 Service。Service 同样可以在自身调用 stopSelf 或 stopService 方法来结束 Service。

2) bindService 方式启动

另一种启动方式是调用 bindService 方法启动 Service，此时会依次调用 onCreate 和 onBind 方法启动 Service。而当通过 unbindService 方法结束 Service 时，则会依次调用 onUnbind 和 onDestroy 方法。

3、BroadcastReceiver

在 Android 中，Broadcast 是一种广泛运用的在应用程序之间传输信息的机制，用于接收广播。而 BroadcastReceiver 是对发送出来的 Broadcast 进行过滤接受并响应的一类组件。可以使用 BroadcastReceiver 来让应用对一个外部的事件做出响应。

在 BroadcastReceiver 的使用过程中，首先在 Intent 中封装将需要广播的消息，然后通过调用 sendBroadcast()、sendOrderedBroadcast()和 sendBroadcast()三种方法中一种将 Intent 广播出去，再通过 IntentFilter 对象来过滤所发送的实体 Intent，最后重写 onReceive。

注册 BroadcastReceiver 对象的方式有两种，一种是在 AndroidMainfest.xml 中声明，另一种是在 Java 代码中设置。

4、ContentProvider

ContentProvider 是用来实现应用程序之间数据共享的类。当需要进行数据共享时，一般使用 ContentProvider 为需要共享的数据定义一个 URI，然后其他应用程序通过 Context 获得 ContentResolver 并将数据的 URI 传入即可。

android.provider 包含一些 ContentProvider，这些 ContentProvider 是 Android 系统为一些常用的数据创建的。只要有相应的权限，自己开发的应用程序就可以访问这些数据。

对于 ContentProvider 而言，最重要的就是数据模型和 URI。

- 数据模型——ContentProvider 为所有需要共享的数据创建一个数据表，在表中，每一行表示一条记录，而每一列代表某个数据，并且其中每一条数据记录都有一个“_ID”字段，它是用来标识每条数据的。
- URI——每个 ContentProvider 都会对外提供一个公开的 URI 来标识自己的数据集，当管理多个数据集时，将会为每个数据集分配一个独立地 URI，所有的 URI 都以“content://”开头。

2.2 即时通讯协议 XMPP 协议

XMPP（Extensible Messaging and Presence Protocol，可扩展的消息与出席协议）是一种基于 XML 的传递出席信息（Presence）和消息路由的协议[5]，它为不同的网络之间互联提供了一种安全而简单的编程语言，是 Jabber 系统的基础，IETF 成立了 XMPP 工作组并已发布了若干项草案[6]。它是一种公开的协议，有很多 IM 都使用了 XMPP。

XMPP 是目前主流的四种 IM 协议之一，其他三种协议分别为：IMPP（Instant Messaging And Presence Protocol）、PRIM（Presence and Instant Messaging Protocol）和 SIMPLE（SIP for Instant Messaging and Presence Leveraging Extensions）。在

这四种协议中，XMPP 是最灵活的。XMPP 是一种基于 XML 的协议，它继承了 XML 的灵活性和可扩展性。因此，基于 XMPP 的应用也同样具有超强的灵活性和可扩展性。经过扩展后的 XMPP 可以通过发送扩展的信息来处理用户的需求，以及在 XMPP 的顶端建立如内容发布系统和基于地址的服务等应用程序。而且，XMPP 包含了针对服务器端的软件协议，使之能与另一端进行通话，这使得开发者更容易建立客户应用程序或给一个系统添加功能。

XMPP 的前身是 Jabber，一个开源形式组织产生的网络即时通信协议[7]。目前 IETF 国际标准组织完成了 XMPP 的标准化工作。标准化的核心结果分为以下两部分：

- 核心的 XML 流传输协议；
- 基于 XML 流传输的即时通讯扩展应用。

由于 XML 流传输协议的定义，使得 XMPP 能够在一个比以往网络通信协议更规范的平台上。同时，XMPP 的协议之所以能够非常漂亮，也是因为 XML 易于解析和阅读的特性。

XMPP 的即时通讯扩展应用部分是根据 IETF 在这之前对即时通讯的一个抽象定义的，与其他已经得到广泛使用的即时通讯协议（诸如 QQ 等）有功能完整、完善等先进性。

2.2.1 XMPP 协议网络架构

XMPP 的特点是将复杂性从客户端转移到服务器端，它的网络结构图如图 2.3 所示。这使得客户端编写变得非常容易，更新系统功能也同样变得容易。XMPP 中定义了三个角色：XMPP 客户端、XMPP 服务器和网关[8]。在这三者的任意两个之间，通信能够双向发生。

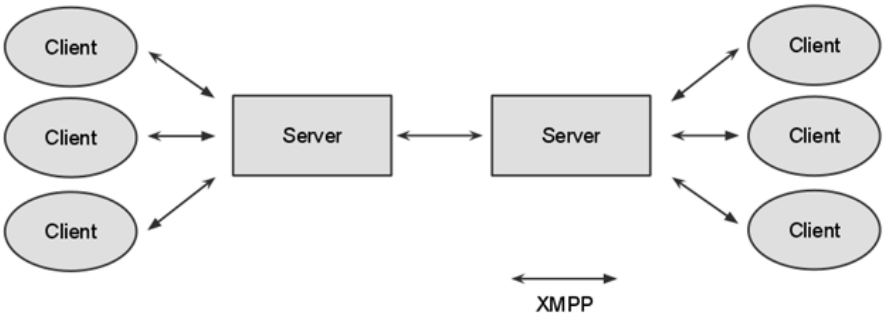

图 2.3　XMPP 协议网络结构图

- 服务器：同时承担了连接管理和信息的路由功能以及客户端信息记录；
- 客户端：通过 TCP 套接字[9]与 XMPP 服务器进行通信；
- 网关：负责与异构即时通信系统的互联互通。

2.2.2 XMPP 协议的地址格式

XMPP 协议的核心是一种类似与电子邮件的逻辑地址方案，在 Jabber 系统中，这一地址被称为 Jabber ID[10]，每个客户端都需要拥有一个地址标识用于定位。地址标识的格式如下：

node@domain[/resource]

例如：charley@gmail.com/spark

上述例子可以解释为：在 gmail.com 服务器注册的 charley 用户，且使用 spark 客户端软件登录。资源(resource)是用来识别属于用户的位置或设备等，一个用户可以同时以多个客户端与同一个 XMPP 服务器连接。

用户地址标识的认证由提供 XMPP 服务的服务器执行。例如，注册于 gmail 服务器的账号由 gmail 服务器进行验证,其他服务器发往 gmail.com 域名的数据包均通过域名查询与服务间验证后发往 gmail 服务器，而不用考虑 gmail 服务器与下属账号间的通信。

2.2.3 XMPP 协议消息格式

XMPP 传输的是与即时通讯相关的指令。以前的传输指令方式分别是：使用纯文本指令加空格加参数加换行符的方式传输，例如 MSN；使用 2 进制的形式传输这些指令，例如 QQ。然而 XMPP 传输的即时通讯指令的逻辑与以往相仿，只是协议的形式变成了 XML 格式的纯文本。这样，不但解析容易，阅读也容易，开发和查错也变得方便了。而 XMPP 的核心部分就是一个在网络上分片断发送 XML 的流协议。它是 XMPP 的即时通讯指令的传递基础，也是一个非常重要的可以被进一步利用的网络基础协议。所以，XMPP 用 TCP 传输的是 XML 流。

XMPP 协议包括三个顶层 XML 元素：Presence、Message 和 IQ。

- Presence 用来表示用户的状态，如在线、离线等等，当用户改变自己的状态时，就会插入一个 Presence 元素在数据流的上下文中，用来表示用户现在的状态。
- Message 用来表示传输的消息，当用户发送一条消息时，就会插入一个 Message 元素在流的上下文中，中间包含用户发送的相关信息。
- IQ 用来表示一种请求 / 响应机制，从一个实体发送请求，另外一个实体接受请求并进行响应。

2.2.4 XMPP 协议优点

1、开放：XMPP 协议是自由、开放和公开的，并且易于了解，而且在客户端、服务器、组件和源码库等方面，都已经各自有多种实现。

2、证实可用：第一个 Jabber(现在 XMPP)技术是 Jeremie Miller 在 1998 年开发的，现在已经相当稳定；此后，数以百计的开发者为 XMPP 技术而努力。现今的互联网上，可以看到更多的 XMPP 服务器，同时也有更多的人们使用 XMPP 即时通讯软件。

3、标准：互联网工程工作小组（IETF）已经将 Jabber 的核心 XML 流协议以 XMPP 之名正式列为认可的实时通信及 Presence 技术[11]。而 XMPP 的技术规格已被定义在 RFC 3920 及 RFC 3921。任何 IM 供应商在遵循 XMPP 协议下，都可与 Google Talk 实现连接。

4、安全：任何 XMPP 协议的服务器都可以独立于公众 XMPP 网络，另一方面，在 XMPP 的技术规格中，内置了 SASL 及 TLS 等技术以确保安全性能。

5、分布式：XMPP 网络的架构和电子邮件十分相像。XMPP 核心协议通信方式是先创建一个 stream，XMPP 以 TCP 传递 XML 数据流，没有中央主服务器，任何人都可以运行自己的 XMPP 服务器，使个人及组织能够掌控他们的即时通讯体验。

6、可扩展：XMPP 的数据传输基于 XML 格式，可扩展性强。XMPP 的核心协议栈 (Core Stack) 部分只定义了基础的 Presence、Message 和 IQ 等最主要数据格式和传输逻辑，更多的功能则通过定义扩展 (Extensions) 实现。

2.3 Openfire 服务器

Openfire 是采用 Java 开发、开源的实时协作（RTC）服务器，主要基于 XMPP（Jabber）协议。可以使用它轻易的构建高效率的即时通信服务器。Openfire 服务器的内核主要由连接管理组件、服务器连接管理组件、会话管理组件、注册登录管理组件、管理更新组件、外部管理组件、数据存储组件、文件传输管理组件和传输器组件等组件组成。Openfire 安装和使用都非常简单，并利用 Web 进行管理。单台服务器可支持上万并发用户。由于是采用开放的 XMPP 协议，可以使用各种支持 XMPP 协议的 IM 客户端软件登陆服务。

2.3.1 Openfire 优点

选择 Openfire 作为业务服务器是因为其自身的诸多优点[12]：

（1）Openfire 内部集成 Resin Web 服务器，可以设计基于 Web 的管理程序。

（2）实现了插件机制，方便扩展。服务器在运行的时候，会定时地扫描一个特定目录下的文件，当发现有新的 Jar 包出现时，就读入它所有的类，分析类中有没有支持插件接口的类，如果有，就加载并运行它。

（3）用户容量方面，单台服务器可支持上万并发用户。在测试环境下可以支持 5000 用户同时在线，每秒可以转发 2000 个包。

（4）Openfire 安装和使用都非常简单。

2.3.2 Openfire 通信

Openfire 的通信处理基于 Apache MINA 框架实现。Apache MINA 是一个网络应用程序框架，用来帮助用户简单地开发高性能和高可靠的网络应用程序。它提供一个通过 Java NIO 在不同的传输（例如 UDP/IP 和 TCP/IP）上抽象的事件驱动的异步 API。

Apache MINA 框架的特性：

- 超载保护和传输流量控制；
- 为不同的传输类型（TCP/UDP）提供了统一的 API；
- 低级（字节缓存）和高级（用户定义的消息对象和编码）的 API；
- 过滤器作为一个扩展特性，类似 Servlet 过滤器；
- 高度定制化线程模型（单线程/线程池）。

Mina 框架使用一种比较好的非阻塞式的和高性能的 I/O 底层，采取异步 I/O 事件和事件驱动机制，使服务器具有很高的效率和性能，从而能够及时处理大量的客户端连接和 I/O 随机突发性，单台服务器可支持上万并发用户。

2.4 Smack 包

Smack 是一个开源、易于使用的 XMPP（Jabber）客户端类库[13]。Smack 是一个 XMPP 协议的 Java 实现，提供一套可扩展的 API。

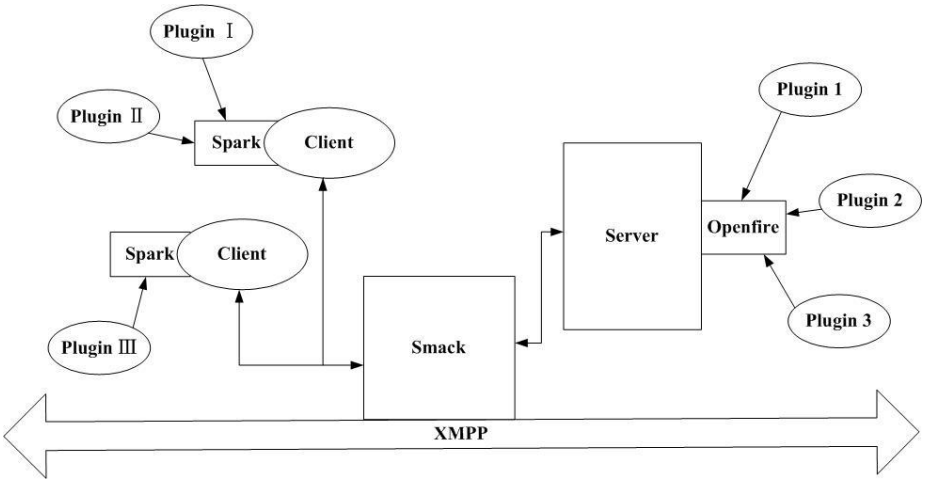

图 2.4 Openfire、Spark 和 Smack 三者之间的关系

Smack、Spark 和 Openfire 三者之间的关系如图 2.4 所示。从图上可以了解到，client 端和 server 端都可以通过插件的方式来进行扩展，Smack 是二者传递数据的媒介。

Smack 的主要优势体现如下：

使用简单且拥有强大的 API，向用户发送一条文本消息只需用以下三行代码即可完成：

```
XMPPConnection conn = new XMPPConnection("speak.com");
conn.login("username", "password");
conn.createChat("hello@openfire.cn").sendMessage("Hello!”);
```

Smack 提供更高级别的、智能的结构，因此开发人员不需要熟悉 XMPP XML 格式。它还提供简单的机器到机器的通讯，同时还允许你对每一条消息的任何数字属性进行设置，也包括 Java 对象的属性。此外，你可将其用于商业的和非商业的应用，因为 Smack 是 Apache 许可的开放源码。

2.5 本章小结

本章主要叙述了基于 Android 即时通讯研究的相关技术。介绍了 Android 的相关基础知识、XMPP 协议及其地址格式、Openfire 服务器及其优势和通信处理基于的 Apache MINA 框架、Smack 包以及 Smack 与 Openfire 和 Spark 之间的联系，也简述了 Smack 包的优势。

第三章 系统的设计与实现

本系统采用客户端(C)／服务端(S)架构的体系结构[14]，具有服务器端和客户端，采用开源的 XMPP 协议作为通讯协议[15]。

客户端：在 Android 2.2 系统上开发。通过无线网络与 Internet 网络建立连接，通过服务器实现与各个客户端之间的即时通讯。通信过程中的初始化由客户端负责，在进行即时通讯时，客户端向服务器发起创建连接的请求。

服务端：采用开源的 Openfire 服务器，允许多个客户端同时登录并且并发的连接到一个服务器上。至于每个客户端的连接，服务器均需要对其进行认证；服务器只对认证通过的客户端创建会话，客户端与服务器端之间的通讯就在该会话的上下文中进行。

3.1 系统的需求分析

本系统一个基于 Android 操作系统的即时通讯系统，它能够使用户在手机上收发即时通讯消息和传输文件附件以及图片。此外，本系统还添加了一些额外的功能，各项功能如下所述：

1、注册用户到 Openfire 服务端

需要输入注册用户的用户名与密码；注册成功则跳转到登录界面，否则给出注册失败的提示。要求输入密码时需要有确认密码，两次输入一致才可以成功注册。

2、用户登录

使用对应的用户名与密码进行登录，登录成功则该用户的状态改为在线，并跳转到主界面；否则给出登录失败的提示，仍旧回到登录界面。

3、注销用户登录

在服务端注销该用户的登录，状态设置为“离线”，要求注销时需要给出一个对话框询问是否注销。

4、发送和接受消息与附件

显示每一条发送和接受的消息，同时显示发送者（或是接受者）以及时间。对于发送附件，给出发送成功与否的提示；对于接收附件，首先给出是否接受附件的对话框，然后接受附件，并给出接受成功与否的提示。发送的附件可以是文本附件也可以是图片。

5、添加好友或是和删除好友

用户可以通过输入好友名称的方式添加好友。用户也可以删除好友。

6、添加分组

用户可以通过输入分组的名称添加一个分组列表。

7、更改用户密码

对于已经登录的用户可以更改其登录密码，操作为直接输入新密码来替代旧密码；

8、更换聊天界面

可以随用户喜好更换系统界面，包括主要的边框背景和会话背景。

9、消息通知

可以更改消息的提示方式，可以振动提示，也可以更换铃声提示的铃声。

10、退出

即退出系统时要给出是否退出的对话框询问。

主要界面设计如图 3.1 所示，会话界面的设计如图 3.2 所示。

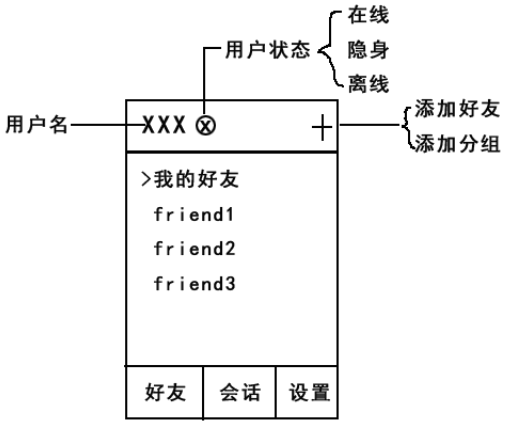

图 3.1　主界面

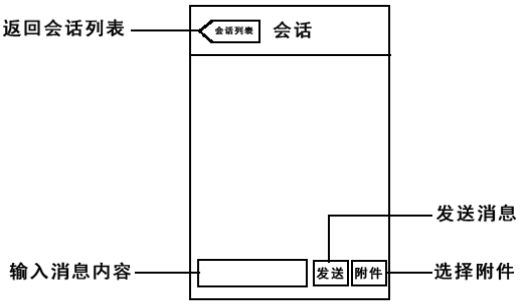

图 3.2　会话界面

3.2　系统总体流程设计

根据需求分析[16]，首先将系统按照其功能划分模块：

登录模块：实现登录功能，由用户输入用户名和密码，登录到 Openfire 服务器上。

注册模块：用户提供用户名和密码注册新的账户到 Openfire 服务器。

附件模块：读取 Android 端的存储卡上的信息，获取指定文件或图片的路径。

会话模块：负责与服务器建立通讯，发送与接收消息，上传与接收附件。

更改用户状态模块：负责更改用户的状态，包括“在线”与“隐身”。

更改用户密码模块：对于已经登录的用户，输入新的密码，更改用户的密码。

添加好友模块：添加已经在 Openfire 服务器上注册的用户，并与之成为好友。

删除好友模块：删除某一好友。

添加分组模块：添加分组列表。

更改聊天界面模块：系统提供三套皮肤供用户选择，更改标题文字的背景条以及会话的背景色。

更改消息通知模块：更改消息通知的方式，振动或者是选取不同的铃声。

关于模块：显示系统的信息，包括系统名称、版本、开发者等。

综上所述，系统总的系统流程图，各模块间的联系如图 3.3 所示：

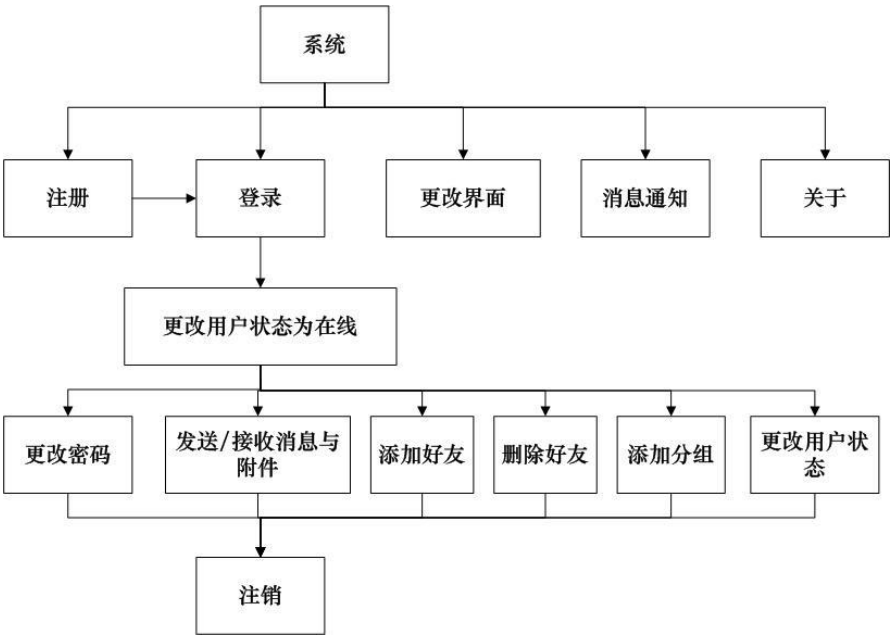

图 3.3　系统总的业务流程

各个功能之间的联系为新用户可以注册账户，注册之后进行登录，登录的同时将用户状态改为在线。

用户在登录的情况下可以执行的操作有修改自己的密码、与他人会话、添加好友、删除好友或是添加分组，也可以更改自己的在线状态，即“在线”或是“隐身”，当然也可以注销自己的登录。

用户在未登录的情况下，可以执行的操作有更改系统的界面皮肤，更改消息的通知方式，查看系统的关于信息。

3.3　系统的体系结构设计

本系统采用开源的 Openfire 来建立即时通讯服务器。Openfire 实现了插件机制，方便扩展。若要与其它的通讯软件实现通讯，只要安装与异构网络通讯的插件就行了。采用 Apache 的 Mina 框架在网络连接中实现网络连接。

在 Openfire 服务器中，针对每个用户的请求，创建一个线程来进行响应，同时采用线程池来创建和管理线程，提高服务器的执行效率。线程池的大小是根据服务器在运行的过程中接收到的用户请求的数量自动调整的。如果长时间没有用户请求，则可以销毁一定数量的线程，使线程池中保持一定的线程数。当有用户请求时而线程池中没有空闲线程，且此时线程池中的线程数目没有达到最大线程数目时，则创建新的线程，如果达到最大线程数目而又没有空闲线程时，则用户请求进入队列等待，直到线程池中有空闲线程，当用户退出时，回收线程到线程池中。

本系统的主要通讯模块包括注册、登录、更改密码、发送/接收消息和附件、添加好友、删除好友、添加分组、更改状态和注销。将这些模块与 Openfire 服务器以及数据库相结合，得到如图 3.4 所示的体系架构图。

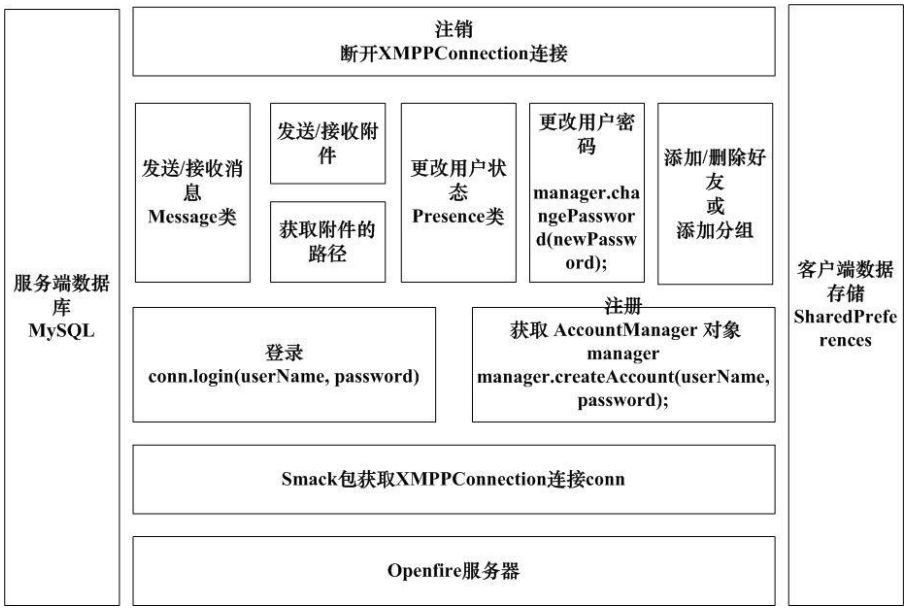

图 3.4　系统的体系架构

该系统的体系架构从下至上总共分为五个层次：

最底层是 Openfire 提供服务支持。

第二层是通过 Smack 包中的 XMPPConnection 类与服务器取得连接；

第三层登录和注册功能的实现。调用 login 方法进行的登录，使用 AccountManager 类的 createAccount 方法完成新用户的注册。

第四层是登录用户可以执行的操作，包括使用 Message 类发送/接收消息或附件，使用 Presence 类更改用户的状态，调用 AccountManager 类的 changePassword 方法更改用户的密码，还可以添加好友与分组。

最上层是注销，注销后的用户不可以在于他人进行会话。

服务器的数据存储采用 MySQL 数据库，客户端由于数据量较少，所以数据存储采用 SharedPreferences 轻量数据存储以提高系统的运行效率。

3.4　系统的数据库设计

系统服务器端的数据存储在 MySQL 数据库中，主要来源于 Openfire 服务器，无需人工设计，因此主要研究客户端的数据存储。

系统的客户端的数据存储采用 SharedPreferences。SharedPreferences 是一种轻型的数据存储方式，它的本质是基于 XML 文件存储 key-value 键值对数据，通常用来存储一些简单的配置信息。其存储位置在/data/data/<包名>/shared_prefs

目录下。SharedPreferences 对象本身只能获取数据而不支持存储和修改，存储和修改是通过 Editor 对象实现。实现 SharedPreferences 存储的步骤如下：

1）根据 Context 获取 SharedPreferences 对象

2）利用 edit()方法获取 Editor 对象。

3）通过 Editor 对象存储 key-value 键值对数据。

4）通过 commit()方法提交数据。

在程序代码中，通过 getXXX 方法，可以方便的获得对应 key 的 value 值，如果 key 值错误或者此 key 无对应的 value 值，SharedPreferences 提供了一个赋予默认值的机会，以此保证程序的健壮性。在访问一个不存在 key 值过程中，并无任何异常抛出。

SharedPreferences 对象与 SQLite 数据库相比，免去了创建数据库、创建表、写 SQL 语句等诸多操作，相对而言更加方便，简洁。

本系统包含两个 XML 文件，分别是 save.xml 和 username_info.xml 文件。

save.xml 文件用于保存系统当前的信息，包括当前用户信息和界面信息具体介绍如表所示：

表 1

列名	属性	备注
current_user	String	系统的当前使用用户的用户名
login_flag	String	系统的当前使用用户的用户状态，available（在线），unavailable（隐身），offonline（未登录）
bar	int	系统的背景条，存储的是图片的资源标识符
talk_back	int	会话的背景，存储的是图片的资源标识符

username_info.xml 文件中的 username 为登录用户的名称，该文件用于存储用户的好友信息，具体介绍如表 2 所示：

表 2

列名	属性	备注
friend	String	用户的好友，每个好友之间用“#”隔开

3.5 系统的界面设计

本系统主要包括主界面（SpeakActivity）、好友界面（Friend）、会话列表界面（TalkList）、设置界面（Set）、会话界面（Talk）和附件浏览界面（File），此外还包括选取皮肤界面（Skin）、登录界面（Login）和注册界面（Register）。以下主要叙述了几个较复杂的界面的设计。

3.5.1 主界面设计

本系统的主界面为用户进入系统后看到的界面，采用 Tab 标签，使用户能够在好友界面、会话列表界面和设置界面三个界面中随意切换。

主界面的类 SpeakActivity 继承自 ActivityGroup，其布局文件内部定义好了 TabHost，可以通过 getTabHost 方法获取。TabHost 包含了两种子元素：一些可以自由选择的 Tab 和 tab 对应的内容 tabContent，在 Layout 的<TabHost>下它们分别对应 TabWidget 和 FrameLayout。这里<TabWidger> 和<FrameLayout>的 Id 必须使用系统 id，分别为“android:id/tabs”和“android：id/tabcontent”。

FrameLayout 为帧布局，它在屏幕上开辟出一块区域，可以添加多个子控件在这块区域中，但是所有的子控件都被对齐到屏幕的左上角。帧布局的大小由子控件中尺寸最大的那个子控件来决定，如果子控件一样大，同一时刻只能看到最上面的子控件。

布局文件写好后，接着向 TabHost 添加 tabs，即调用 tabHost.addTab(TabSpec tabSpec) 方法。TabSpec 类主要包含了以下两个方法：

- setIndicator 方法：指定 Tab，包括标签的名称和背景图片；
- setContent 方法：指定 tabContent，即对应的 Activity。

最后，通过 tabHost.setCurrentTab 方法来指定首要显示的界面，本系统中首要显示的界面为好友界面。

在主界面中，按下键盘上的 menu 键会显示退出、登录、注册和注销菜单。通过 Menu 类来实现，在 onCreateOptionsMenu 方法中通过 menu.add 方法加入四个菜单，再在 onOptionsItemSelected()方法中对菜单进行监听。

其中当点击“退出”和“注销”按钮时会显示对话框询问是否要退出或是注销。

3.5.2 好友界面设计

好友界面的布局整体采用 LinearLayout 线性布局，朝向为垂直方向。

标题栏部分采用 RelativeLayout 相对布局，其中的子控件均位于父控件的正中间位置。在相对布局中，子控件的位置是相对兄弟控件或父容器而决定的。出于性能的考虑，在设计相对布局时要按照控件之间的依赖关系排列，在本界面中，从右侧起先是添加图片的控件 ImageView，它与父控件的右侧对齐并有一定间距，其次是用户名的控件 TextView，它与父控件的左侧对齐并有一定距离，顶部与添加图片的 ImageView 的顶部对齐，最后是用户状态控件 ImageView，它位于用户名控件 TextView 的右侧，底部与其的底部对齐。

对用户状态控件 ImageView 添加事件监听器，当点击该图片时则显示一个单选按钮对话框，单选列表分别为“在线”和“隐身”。

对添加图片 ImageView 同样添加一个事件监听器，当点击该图片时则显示一个列表对话框，列表内容为“添加好友”和“添加分组”。点击两者中的任何一个，均出现一个包含可编辑的文本框的对话框，在文本框中输入要添加的好友或是分组的名称。

中间是一个 TextView，内容为“我的好友”。

接着是一个 ListView，用于显示好友的名称。ListView 是一种列表视图，将 ListAdapter 所提供的各个控件显示在一个垂直且滚动的列表中。在此 ListView 中只包含一个 TextView 控件用于显示好友的名称，适配器采用 BaseAdapter 基础适配器。

3.5.3 设置界面设计

设置界面的标题栏部分为一个 TextView，内部元素的布局方式为正中间对齐（center），并显示“设置”两字。

主要内容部分用一个 ListView 显示，列表内容包括“更改密码”、“消息通知”、“聊天背景”和“关于”。列表采用 BaseAdapter 基础适配器。

然后调用 setOnItemClickListener 方法对 ListView 列表添加事件监听器：

- ✧ 更改密码——显示一个普通对话框，对话框内包含一个 EditText 控件，用于输入新的密码；
- ✧ 消息通知——显示一个单选对话框，用于更改消息的提示铃声；
- ✧ 聊天背景——跳转到选取皮肤界面；
- ✧ 关于——显示一个普通对话框，用于说明本系统的相关信息。

3.5.4 页面交互设计

Android 使用了 Intent 类来实现各个活动之间的切换工作[17]。Intent 起着一种媒体中介的作用，专门提供组件互相调用的相关信息，实现调用者与被调用者之间的解耦。它主要负责对应用中一次操作的动作、动作涉及的数据和附加数据进行描述。Android 根据 Intent 的描述找到对应的组件，将 Intent 传递给调用的组件并完成组件的调用。在应用中，我们可以以两种形式来使用 Intent：

- ➢ 间接 Intent：没有指定 component 属性的 Intent，则需要包含足够的信息，这样系统才能根据这些信息，在所有的可用组件中，确定满足此 Intent 的组件。
- ➢ 直接 Intent：指定了 component 属性的 Intent，通过指定具体的组件类，通知应用启动对应的组件；

Intent 是由组件名称、Action、Data、Category、Type、Extra 和 Flag 七部分组成的。

- 组件名称：它实际上就是一个 ComponentName 对象，用于标识唯一的应用程序组件。
- Action：它实际上是一个描述了 Intent 所触发的动作名称的字符串，在 Intent 类中，已经定义好多字符串常量来表示不同的 Action，开发人员也可以自己定义 Action。
- Data：主要是对 Intent 消息中数据的封装，主要描述 Intent 的动作所操作到的数据的 URI 及类型，不同类型的 Action 会有不同的 Data 封装，正确的 Data 封装对 Intent 匹配请求同样非常重要。
- Category：它是对目标组件类别信息的描述，同样作为一个字符串对象，一个 Intent 中可以包含多个 Category。与 Category 相关的方法有三个：addCategory 添加一个 Category，getCategory 得到一个 Category，而 removeCategory 删除一个 Category。Android 系统定义了一组静态字符常量来表示 Intent 的不同类别。
- Type。Type 信息，是用 MIME 来表示的，比如 text/plain。
- Extra：封装了一些额外的附加信息，这些信息主要是以键值对的形式存在的。Intent 通过 putExtra()与 getExtra()方法来存储和获取 Extra。
- Flag：一些有关系统如何启动组件的标志位，Android 同样对其封装。

Intent 解析机制主要是通过查找已注册在 AndroidManifest.xml 中的所有 IntentFilter 及其中定义的 Intent，最终找到匹配的 Intent。在这个解析过程中，Android 是通过 Intent 的 action、type 和 category 这三个属性来进行判断的。

Intent 的跳转流程如下所示：

1）创建一个 Intent 对象 intent；

2）调用 setClass 方法，该方法包含两个参数，第一个参数是本 Activity，第二个参数是要跳转到的 Activity 的名字；

3）调用 startActivity 方法，实现页面跳转。

本系统使用 Intent 机制实现各个页面的跳转与切换。首先主界面采用 Tab 标签，可以在好友界面、会话列表界面和设置界面之间随意切换，也可以通过 menu 菜单切换到登录界面和注册界面，登录成功后返回到主界面，还可以通过点击好友名称切换到会话界面，会话界面可以返回到好友界面或是切换到会话列表界面。然后，登录界面可以切换到注册界面，注册成功后可以返回到登录界面进行登录。最后设置界面可以切换到更换皮肤界面，皮肤更换成功后回到主界面。各个界面切换的关系如图 3.5 所示。

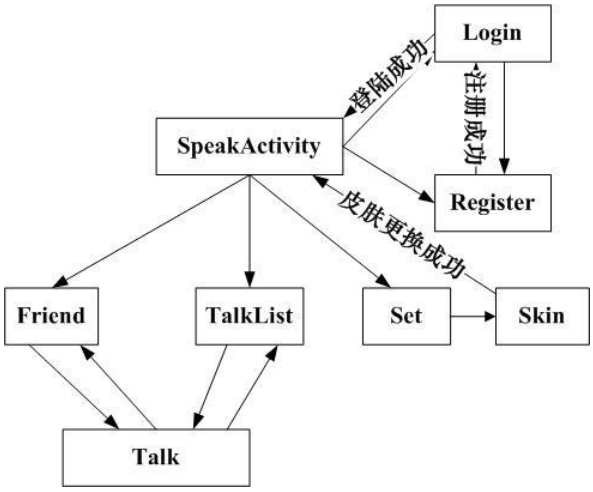

图 3.5　页面交互流程

3.6　系统的功能设计与实现

本系统的功能包括登录与注销、注册、获取好友、更改用户状态、会话（包括文本消息和文件）、更改密码添加好友、SD 卡上的文件浏览（附件浏览）以及更改界面皮肤，以下依次叙述了各个功能的设计与实现。

3.6.1　登录/注销功能

本系统采用开源的 Openfire 作为服务器，以 Smack 作为媒介实现客户端与服务端相互通信。

单独设计一个 XmppTool 类用来实现与服务器的连接关系，即获取以及关闭 XMPPConnection 连接，这样也方便了系统以后的扩展。XmppTool 的代码如下所示：

```
public class XmppTool {
    private static XMPPConnection con = null;
        private static void openConnection() {
        try {
            ConnectionConfiguration connConfig =
                        new ConnectionConfiguration("10.0.2.2", 5222);
            con = new XMPPConnection(connConfig);
            con.connect();
        } catch (XMPPException xe) {
            Log.i("time", "time is out");
            xe.printStackTrace();
        }
```

```
    }
    public static XMPPConnection getConnection() {
        if (con == null) {
            openConnection();
        }
        return con;
    }
    public static void closeConnection() {
        con.disconnect();
        con = null;
    }
}
```

ConnectionConfiguration(String ipAddress, int port)方法中的第一个参数为服务器所在的 IP 地址，由于本系统以个人电脑作为 Openfire 服务器，因此这个参数的值为“10.0.2.2”，即 Android 模拟器连接所在电脑时的 IP；第二个参数为 Openfire 服务器的端口，一般情况下为默认值 5222。

获取了与服务器的连接，接着调用 XMPPConnection 的 login()方法登录到服务器，代码如下所述：

```
XmppTool.getConnection().login(USERID, PWD); //登录服务器
Presence presence = new Presence(Presence.Type.available);//将状态改为在线
XmppTool.getConnection().sendPacket(presence);//设置用户状态
```

其中 Presence.Type 表示注册到服务器的即时通讯的用户的状态，共有 7 个状态，本系统只使用到如下的 2 个状态：

- available：此状态下用户可以接收消息，对其好友可见，为默认状态；
- unavailable：此状态下用户无法接收消息，对其好友不可见。

sendPacket()方法是发送指定的数据包到服务器，此处为用户的状态数据包。

登录的流程如图 3.6 所示，首先在客户端输入用户名和密码，点击登录，则采用如上代码登录到服务器。登录成功则跳转到主界面，并对 save.xml 文件作如下更改：

- login_flag 值为 available（在线）；
- current_user 值为登录的用户名；

同时创建一个 username_info.xml 文

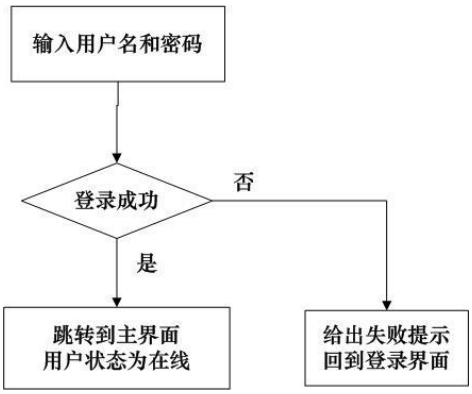

图 3.6　用户登录流程图

件，用于存放该用户的好友信息。

登录失败则继续留在登录界面，并给出登录失败的提示。

注销，即在服务器端注销某一用户的登录，注销后，该用户不可以再进行与服务器相关的操作，如接受消息/发送消息与附件以及更改密码等等。通过调用 XMPPConnection 的 disconnect 方法关闭与服务器的连接，并将 save.xml 文件中的 login_flag 的值设置为 offline，刷新整个系统。

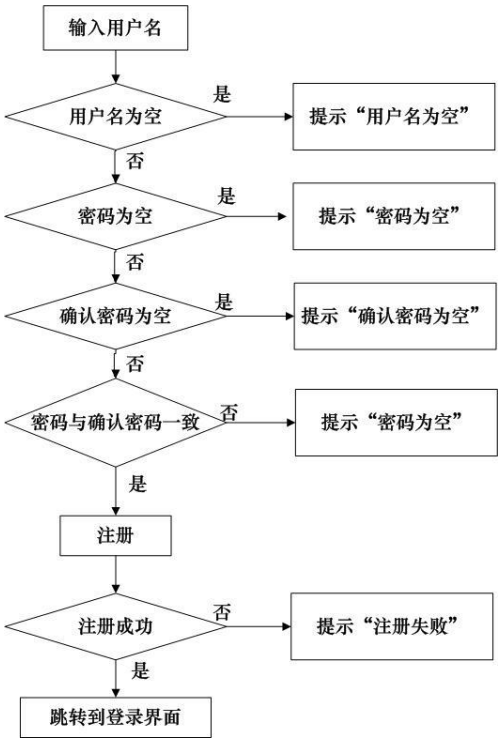

图 3.7　用户注册流程图

3.6.2　注册功能

为保证用户登录信息的安全问题，用户在注册时需要输入两次密码进行确认。首先检测用户名是否为空，用户名不为空时检测密码是否为空，密码不为空时检测确认密码是否为空，确认密码不为空时检测确认密码与密码是否一致，两者一致时调用 AcountManager 类的 createAccount 方法向服务器注册新用户。用户注册成功后，跳转到登录界面；否则，给出相应提示并留在注册页面。具体的流程图如图 3.7 所示。

注册部分的主要代码如下所示：

```
if (loginCheck(userName, password, passwordAgain)){//检测是否为空
    AccountManager account =
                    XmppTool.getConnection().getAccountManager();
    try{
        account.createAccount(userName, password); //注册用户
        Intent intent = new Intent(Registered.this, Login.class);
        startActivity(intent); //注册成功后跳转到登录页面
    } catch (Exception e){
        Toast.makeText(getApplication(), Constants.REGISTER_FAIL,
                toast.LENGTH_SHORT).show();
    }
}
```

其中loginCheck()方法的作用为检测用户名、密码和确认密码是否合法。当不合法时通过Toast显示错误提示。

Toast是Android中用来显示显示信息的一种机制，它向用户提供比较快速的即时消息，当Toast被显示时，虽然其悬浮于应用程序的最上方，但是Toast从不获得焦点，而且Toast显示的时间有限，过一定的时间就会自动消失。因为设计Toast时就是为了让其在提示有用信息时尽量不显眼。Toast对象的创建是通过Toast类的静态方法makeText来实现的，该方法有两个重载实现，主要的不同是一个接受字符串，而另一个接受字符串的资源标识符作为参数；本系统使用前一种重载方法。Toast对象创建好后，调用其show()方法即可将消息提示显示到屏幕上。

3.6.3 获取好友列表功能

在即时通讯系统中，存放某一个用户所有好友信息的类似好友列表的表格称为花名册（Roster），它可以让你很清楚的知道其他可用的用户。用户可以被分成像“朋友”、“合作者”这样的组。当成功登陆服务器后，可以使用XMPPConnection.getRoster()获得 Roster 类的实例。在花名册里，好友的显示信息分为以下两种：

1）未分组的好友，显示形式为 cat@example.com；

2）已分组的好友，显示形式为cat: cat@example.com [Friends]，其中Friends为分组的名称，cat为用户名。

获取联系人列表时，客户端需要发送get类型的IQ数据包。服务器收到请求后，返回roster列表。在客户端进行实现时的主要代码如下：

```
Roster roster = XmppTool.getConnection().getRoster();
Collection<RosterEntry> entries = roster.getEntries();
StringBuffer friendBuffer = new StringBuffer();//用于存放好友名
for (RosterEntry entry : entries) {
int indexOfDouble = entry.toString().indexOf(":");
if(indexOfDouble == -1){
    int index = entry.toString().indexOf('@');
    friendBuffer.append(entry.toString().substring(0, index));
    friendBuffer.append("#");
}
else{
   friendBuffer.append(entry.toString().substring(0, indexOfDouble));
   friendBuffer.append("#");
}
}
```

采用StringBuffer暂时保存好友的用户名，StringBuffer对象是一个字符串变量，

可以自由扩充与修改。对于花名册中的一条信息，首先查看该好友是否已被分组，即调用indexOf()方法查找是否存在“:”，若存在则记下其位置的下标，然后调用subString()方法截取从开头到该下标的字符串；若不存在则继续调用indexOf()方法，找到“@”的所在位置的下标，然后用subString()方法截取从开头到该下标的字符串。再将截取的字符串保存到StringBuffer对象中，然后填加一个“#”用作用户名间的分隔符。最后，将StringBuffer对象的内容写入username_info.xml文件中，当从文件中获取好友名称时，调用split方法将各个用户名分开，并用ListView显示出来。

3.6.4 用户状态功能

用户状态分为“在线”和“隐身”两种状态，分别对应 Presence.Type 枚举类型的 available 和 unavailable。当设置成 available 时，对其好友可见，可以进行接收与发送消息；当设置成 unavailable 时，对其好友不可见，也不能接收消息，此类状态下仍未与服务器断开连接，可以进行除会话外的其他操作。

更改用户状态时，客户端向服务器发送了 presence 的数据包，服务端收到 presence 后，会自行填充 from 和 to 属性，发送到订阅 (subscribed) 了该用户状态信息的联系人服务器上。

以下代码是一个将你的当前状态改为“隐身”，从而不被别人看到的例子：

```
// 创建一个 Presence 对象，设置其值为 unavailable
Presence presence = new Presence(Presence.Type.UNAVAILABLE);
XmppTool.getConnection().sendPacket(presence);
```

在创建Presence对象时，在其构造函数中传入相应状态的值，再实例化，通过调用XMPPConnection的sendPacket()方法将含有用户状态的数据包发送到服务器，更改用户的状态。与此同时，将用户的状态写入SharedPreferences的save.xml文件中，在刷新系统的同时从该文件中获取更改后的login_flag的值并予以相应的显示。如若是在未登录的状态下更改用户的状态，则会给出“请先登录”的提示，提示方式采用Toast显示机制。

3.6.5 会话功能

服务器针对每个连接到其上的客户端均创建一个会话，并且该会话一直活跃在服务器中，直到该会话所对应的用户端断开与服务器端的连接。客户端之间实现消息的发送和接收是通过在服务器中所对应的会话的上下文实现的。当服务器收到每个用户发送过来的消息时，经过一些处理之后交给目标用户的会话，目标用户会话再把消息发送到目标用户客户端。会话管理组件负责管理所有的会话，

它通过线程[18]来提高消息传输的性能。当服务器启动时，一定数量的线程被指派到线程池中，当服务器其他部分的装载组件反馈消息给会话管理组件时，会话管理组件动态地从线程池中取出没有使用的线程并将它们指派给消息端口，负责客户端的连接监听。

在进行会话时，客户端向服务器发送了名为 message 的数据包，其中 to 属性所对应的值指定了要发送到的用户的 JID，body 标签包含了会话的内容，所以在客户端编码实现时可以通过 setbody()与 getbody()方法来设置和获取会话的内容。

以下程序片示例了如何与一个用户进行聊天并发送一段文本消息：

```
//假设我们已经获取了一个 XMPPConnection 对象 connection
Chat newChat = connection.createChat("sun1@liyan-pc");
newChat.sendMessage("Hello!");
```

Chat.sendMessage(String str)方法可以很方便的创建一个消息对象，方法体使用字符串类型的参数，然后发送消息。如果想在发送消息前对消息设置额外的设置，可以使用 Chat 类的 createMessage()方法和 sendMessage(Message messsage)方法，如下例所示：

```
//假设我们已经获取了一个 XMPPConnection 对象 connection
Chat newChat = connection.createChat("sun1@liyan-pc ");
Message newMessage = newChat.createMessage();
newMessage.setBody("Hello!");
message.setProperty("favoriteColor", "red");
newChat.sendMessage(newMessage);
```

Chat.nextMessage()方法可以接受来自好友的消息，其为Message对象，然后可以通过Message类的getBody()方法将接受到的消息显示出来。

```
//假设我们已经获取了一个XMPPConnection对象connection
Chat newChat = connection.createChat("sun1@liyan-pc ");
Message message = newChat.nextMessage();
if(message.getBody() != null){
        System.out.println(message.getBody());
}
```

也可以使用 ChatManager 类设置一个消息监听器，监听来自好友的消息，通过 addChatListener()方法注册一个新的监听器 ChatManagerListener 来接收新的聊天信息。

```
//假设我们已经获取了一个 XMPPConnection 对象 connection
ChatManager manager =connection.getChatManager();
```

```
        manager.addChatListener(new ChatManagerListener() {
            public void chatCreated(Chat chat, boolean arg1) {
                chat.addMessageListener(new MessageListener() {
                public void processMessage(Chat arg0, Message message) {
                  //来自好友 sun1 的消息
                  if(message.getFrom().contains("sun1@liyan-pc ")){
                    //接受好友信息
                    }
                  else{
                  //其它.
                  }
                }
            });
        }
});
```

这里窗口的管理是使用线程的，当有一个新的会话时创建一个线程。

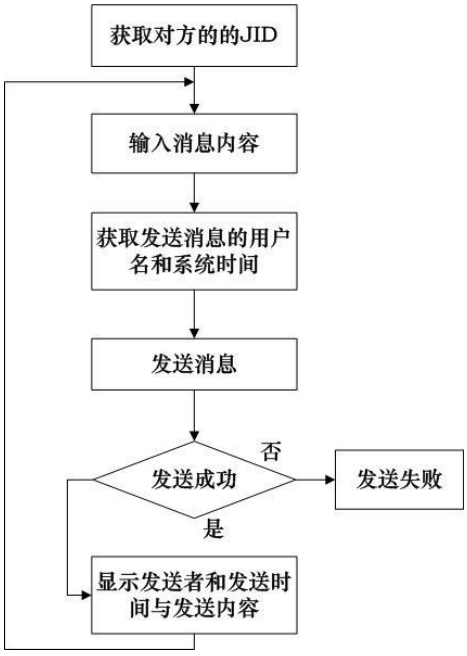

图 3.8　发送消息流程图

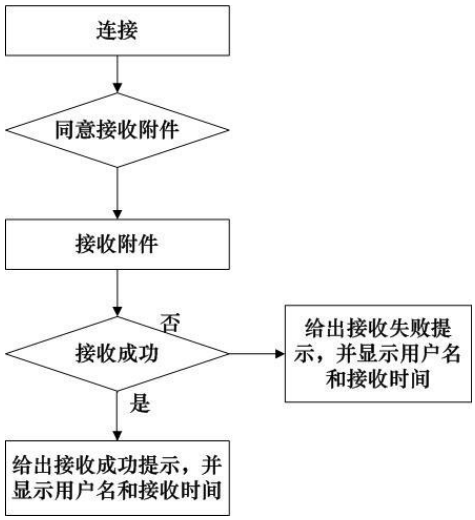

图 3.9　接收消息流程图

本系统的设计是首先从 Friend 类中获取好友的名称(如 sun1)，通过 Intent 传递，再将其规范成 JID 格式(如 sun1@liyan-pc)，然后按照上述的代码发送和接收文字消息。发送与接收消息的同时，将每一条消息保存到 List 的对象中，再加上

时间和发送者等修饰信息并用 ListView 显示出来。发送消息的流程图如图 3.8 所示，接收消息的流程图如图 3.9 所示。

除了接收和发送文字信息外，本系统还提供接收与发送文档附件或是图片附件。至于文件的管理，则主要使用 FileTransferManager 类和 File 类相结合。

FileTransferManager 类是一个文件传输管理类，负责文件的发送与接收，实例化其对象时需要传入 XMPPConnection 的对象。然后调用该类的 createOutgoingFileTransfer 方法获得 OutgoingFileTransfer 类对象，再通过 OutgoingFileTransfer 类的 sendFile 方法发送文件，文件的路径获取如附件浏览功能所述。

```
//假设我们已经获取了一个XMPPConnection对象connection
FileTransferManager fileTransferManager =
new FileTransferManager(connection);
//向sun1发送附件，sun1在Spark端登录
OutgoingFileTransfer fileTransfer =
fileTransferManager.createOutgoingFileTransfer("sun1@liyan-pc /Spark 2.6.3");
//文件路径filepath
File file = new File(filepath);
fileTransfer.sendFile(file, "Sending");
```

接收文件时需要不断监听对方是否有文件发送过来，所以调用 addFileTranferListener 方法来进行监听，当有文件发送过来则注册一个新的监听器 FileTransferListener 来接收文件。接受文件时首先会弹出一个对话框，询问是否接受文件，同意时才会接收文件。

```
//假设我们已经获取了一个XMPPConnection对象connection
FileTransferManager fileTransferManager =
                        new FileTransferManager(connetion);
fileTransferManager.addFileTransferListener(new FileTransferListener{
    public void fileTransferRequest(FileTransferRequest prequest){
        file = new File("mnt/sdcard/" + prequest.getFileName());
        request = prequest;
            IncomingFileTransfer infiletransfer = request.accept();
    }
 }
```

3.6.6 更改密码功能

本系统提供更改密码的功能，对于已经登录的用户，可以通过本系统更改自己的密码。主要通过 AccountManager 类的 changePassword 方法来更改密码，在该方法中传入新的密码。更换新密码成功后，跳转到登录页面重新登录。

更改用户密码是以对话框的形式展现，在文本框中输入新的密码，点击确认按钮进行更改。更改密码的主要代码如下所示：

```
//假设我们已经获取了一个 XMPPConnection 对象 connection
AccountManager account = connection.getAccountManager();
//新密码为 newPassword
account.changePassword(newPassword);
```

更改密码的流程图如图 3.10 所示：

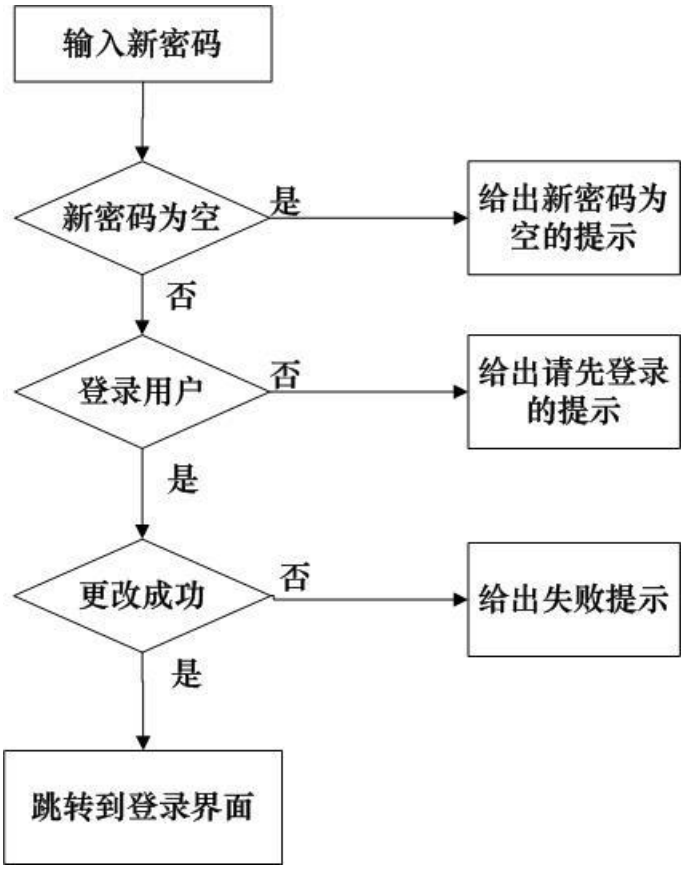

图 3.10 更改用户密码流程图

3.6.7 添加好友功能

通过 Roster 的 createEntry 方法发送好友请求，该方法有三个参数，分别是好友的 JID 号、好友的昵称和好友所在的分组，本系统默认分组为 null，即不对新

添加的好友进行分组限制。之后再将好友的名称写入 username_info.xml 文件中。对于该文件中 key 为“friend”所对应的值得修改分为以下两种情况：

1）该用户之前没有任何好友，则直接将该好友写入“friend”所对应的值中；

2）该好友之前存在好友，则需要将“friend”所对应的值的内容备份，然后与加入分隔符“#”和新添加的好友名称结合后，再覆盖原来的“friend”所对应的值得内容。

最后，刷新整个系统。

3.6.8 附件浏览功能

在 Java 中，File 类是用来构造文件或文件夹的类，在其构造函数中要求传入一个 String 类型的参数，用于指示文件所在的路径。File 对象实际上并不代表一个文件， File 对象本身并不是物理文件或路径，它封装了路径名(pathname)或引用(reference)，该引用指向硬盘上可能存在也可能不存在的物理文件或目录。

操作系统和用户界面使用与系统相关的路径名字符串来命名文件和目录。此类呈现分层路径名的一个与系统无关的、抽象的视图。抽象路径名有两个组件：

- ✧ 一个可选的与系统有关的前缀字符串；
- ✧ 零个或多个字符串名称的序列。

抽象路径名中的第一个名称是目录名，之后的每个名称表示一个目录；最后一个名称既可以表示目录，也可以表示文件。空抽象路径名没有前缀和名称序列。

本系统包含一个 File.java 文件，用于读取 SD 卡的文件信息。首先创建了两个 List 对象 items 和 pathlist，分别用于存放文件的名称和路径。初始路径为 SD 卡的根路径“/”，创建 File 对象，并判断其是否是目录，如果是目录，则分别调用 getName 方法和 getPath 方法读取文件的名称和路径，将文件名保存到 items 中，文件的路径保存到 pathlist 中。

文件的显示通过 ListView 实现，在本系统总创建了一个 FileAdapter 类继承 BaseAdapter 类，重写其 getCount()、getItem()、getItemId()和 getView()方法，其构造方法中传入 Context 参数、存储文件名的 List 对象和存储文件路径的 List 对象。

3.6.9 更改界面皮肤功能

本系统内置三套皮肤，分别为粉色、黄色和蓝色，初始默认设置为粉色皮肤。主要更改的是各个界面标题栏的背景色和会话的背景。

本系统皮肤数据的保存采用 SharedPreferences，采用 ListView 将三种皮肤显示出来，点击 ListView 列表的一个条目是，则将该条目所对应的皮肤信息包括标题栏的背景色和会话背景的资源标识符暂时保存到一个一维整型数组中，然后

只有当点击更换皮肤的按钮时，才会将整型数组的值保存到 SharedPreferences 的 save.xml 文件中，最终皮肤更改成功后回到主界面。

每个 Activity 在开启时均根据 save.xml 文件中保存的皮肤信息的资源标识符找到该图片，并进行相应的设置。

3.7 本章小结

本章首先阐述了本系统的需求分析，然后从需求分析着手，画出系统的总体流程图，再与 Openfire 服务器和数据库相连接给出系统的架构。此后，详述了系统的数据处理，再讲解了一些主要页面的设计与交互。最后具体叙述了系统的各个功能的设计，包括代码和流程图。

第四章 系统的部署与展示

4.1 系统的部署

本系统是基于 Android2.2 操作系统开发的，所以本系统客户端的运行环境不应低于 Android2.2。

本系统的部署需求如下所示：

- 操作系统为 Windows 7 的 PC 机一台；
- 服务器软件 Openfire 3.7.1；
- 数据库软件 MySQL 5.5；
- PC 端即时通讯软件 Spark 2.6.3；
- 配有 Android2.2 系统的 Android 模拟器。

首先在 Windows 7 系统上安装 MySQL 数据库，主要用于充当 Openfire 服务器的数据库。然后安装 Openfire 服务器，将 Openfire 配置好后并运行，运行图如图 4.1 所示。最后，在电脑上安装 Spark 客户端，同时在 Android 模拟器上安装本系统。

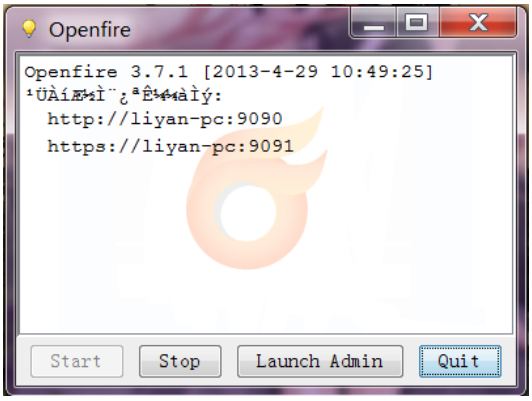

图 4.1 Openfire 运行

4.2 系统的展示

4.2.1 系统的初始状态

如图 4.2 所示为系统的初始化状态，Tab 标签分别为好友、会话和设置，首要显示好友页面内容。由于刚安装的系统没有任何用户，所以标题栏部位的用户名称处显示“null”，用户的状态为“离线”；同时好友处显示“无任何好友”。点击键盘上的 menu 菜单，显示退出、登录、注销和注册四个菜单。在系统初始化时，系统只创建了 save.xml 文件。

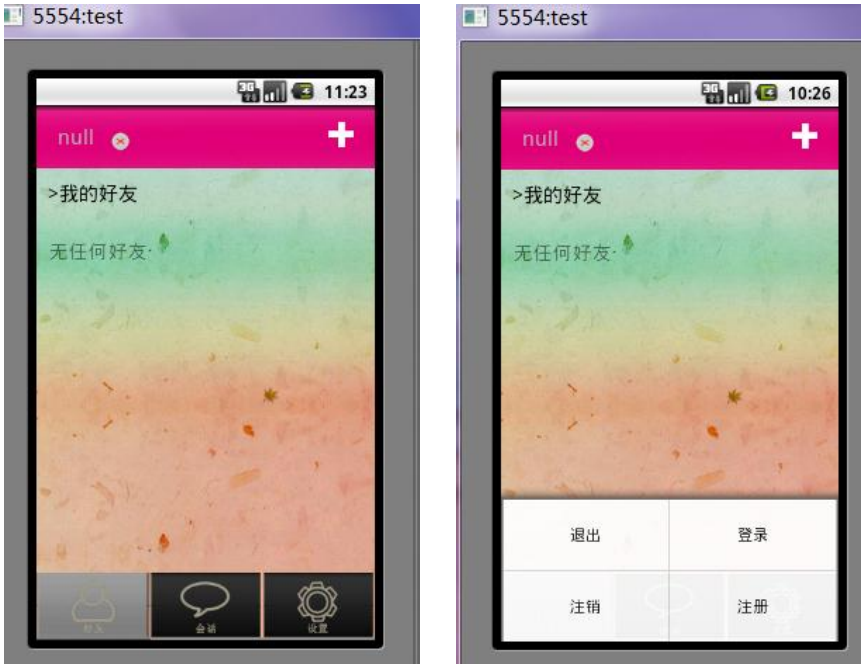

图 4.2　系统的初始界面（右图为 menu 菜单）

4.2.2　注册

点击 menu 菜单的注册按钮，进入注册页面，输入用户名、密码和确认密码，点击注册按钮注册新用户。如图 4.3 所示，注册一个新用户 cat2，左图为 Android 端，右图为 Openfire 服务器端。

图 4.3　注册新用户

4.2.3 登录

如图 4.4 所示，在文本框中输入用户名 sun1 和其密码，点击登录按钮。中间为正在登录中，以进度条 ProgressBar 的形式展示，右图为登录成功后跳转的主界面。此时用户的状态为“在线”，以绿色对号显示，该用户的好友有 sun2 和 sun3，好友之间用横线分割。

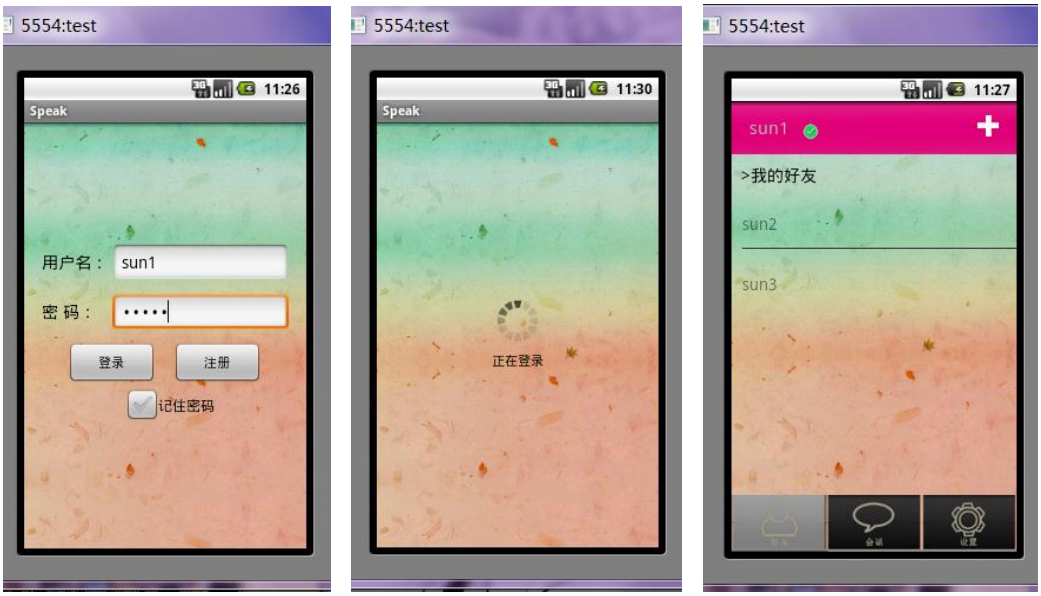

图 4.4 登录

4.2.4 会话

sun1 在本系统登录后，sun2 在电脑端的 Spark 上登录，两者已经是好友的关系，可以进行会话。在文本框中输入要发送的内容，点击发送则发送消息。点击附件按钮则浏览附件。

如图 4.5 所示，每条信息都附有发送者的名称和发送时间。发送附件成功时，消息内容为“发送附件成功”；接受附件是首先显示一个对话框询问是否接受附件，当接受附件成功时，消息内容为“接收附件成功”并给出了附件的保存路径。图 4.5 的中图为当好友发来附件时，是否同意接收对话框，右图为附件浏览界面。附件浏览时，不同的文件的图标不同，同时显示文件的名称。

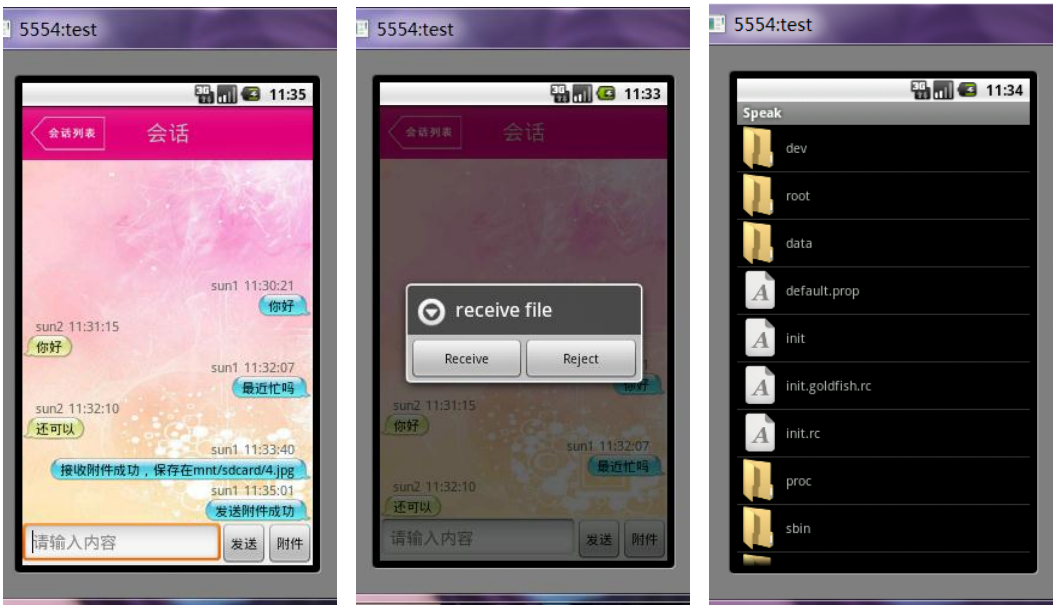

图 4.5　会话

4.2.5　更改用户状态

如图 4.6 所示，更改用户状态以对话框的形式展现，选取不同状态，点击确定按钮进行更换。

图 4.6　更改状态

图 4.7 更改密码

4.2.6　更改密码

如图 4.7 所示，更改用户密码以对话框的形式展现，在文本框中输入新的密码，点击确定按钮，更改用户的密码。

4.2.7 添加好友

如图 4.8 所示，添加好友，点击右上角的添加按钮，再点击“添加好友”条目，之后在对话框中输入好友的名称，如输入已经注册的用户名称 cat1，即添加 cat1 到其好友列表。

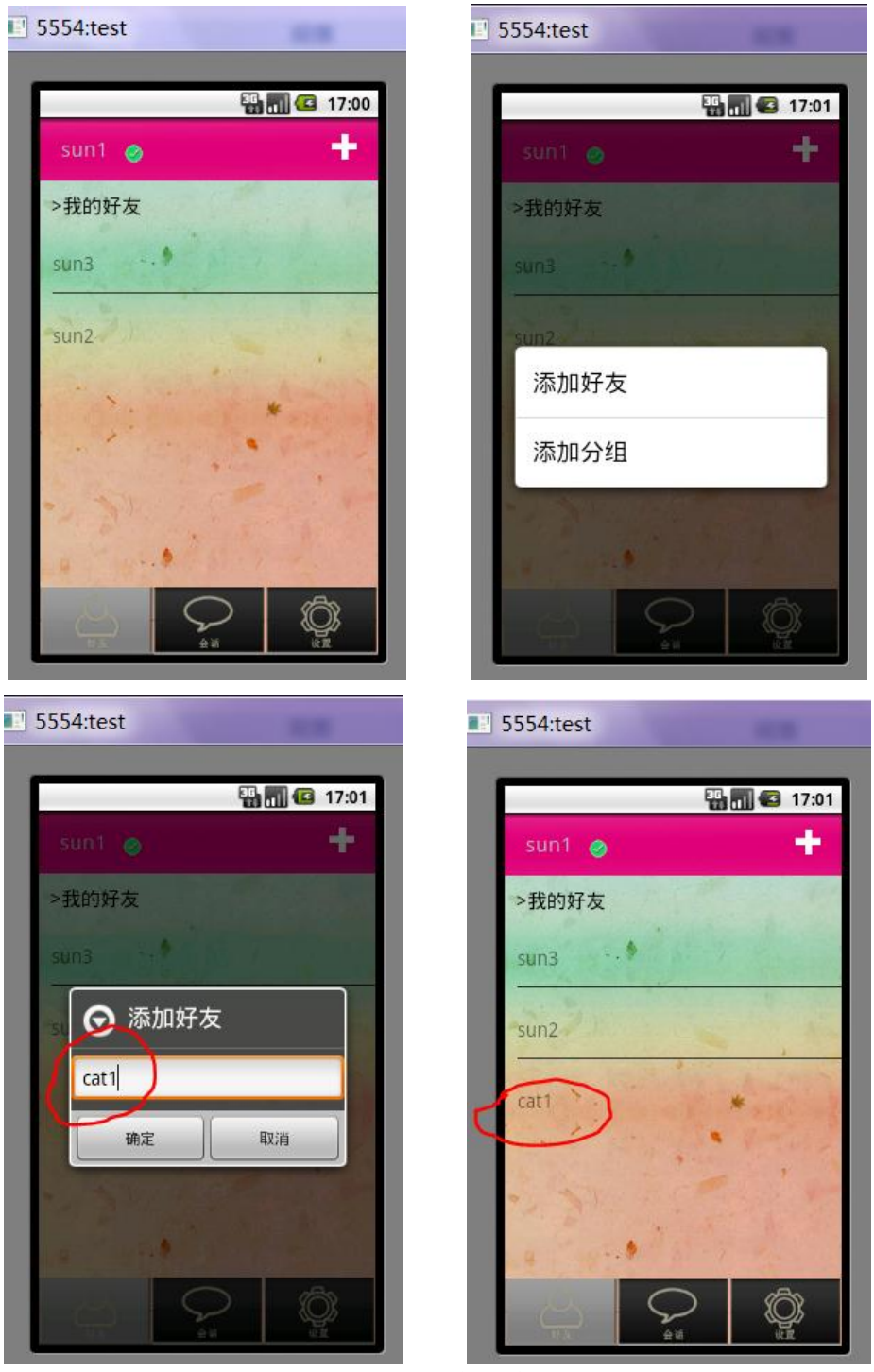

图 4.8 添加好友

4.2.8 更改界面皮肤

系统提供三套皮肤，分别为粉色、黄色和蓝色的皮肤界面，其中粉色为默认皮肤。点击设置中的更改界面皮肤条目，选择不同的皮肤再点击确定按钮，则更改界面皮肤，成功后回到主界面。如图 4.9 所示，三套不同的皮肤。

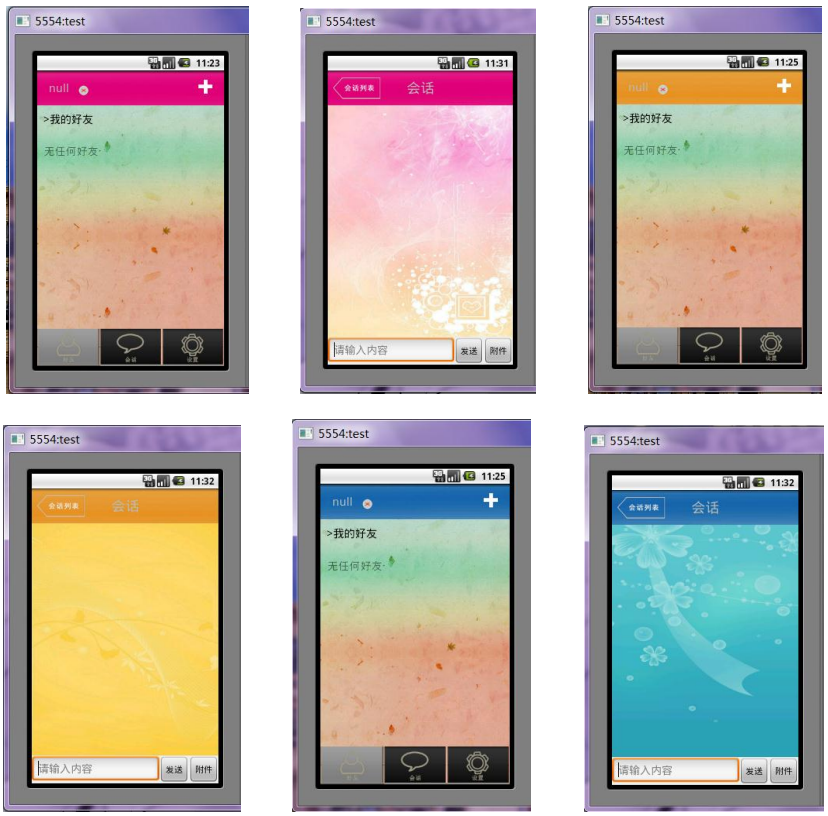

图 4.9 换肤

4.2.9 用户注销

用户注销时，出现对话框提示，如图 4.10 所示，点击确定按钮才会在服务器端注销该用户饿登录。

图 4.10 注销

4.3 本章小结

本章主要为系统的部署和展示部分，首先是系统的部署，然后针对每个功能模块进行展示，展示的内容为运行后的结果图。

第五章 结语

本系统是在使用开源的 Openfire 作为即时通讯服务器的基础上，使用 Eclipse 开发环境和 Java 开发语言开发的即时通讯手机客户端软件。本系统基于 XMPP 协议，各个客户端通过 openfire 即时通讯服务器进行通讯。最后在 Android2.2 模拟器上安装运行，成功传输文字消息和附件。

在系统设计和开发过程中，主要经历了一下几个阶段：

1、对即时通讯相关基础知识的学习过程，有 Android 系统结构、Android 应用框架、Activity 生命周期、Android 应用组件、XMPP 协议、Openfire 服务器的安装与使用和 Smack 包的 API。

2、对系统进行需求分析与整体架构的设计，画出系统流程图和系统结构图，用 Eclipse 进行相关页面的 UI 设计。

3、在有系统框架的基础上，细化系统相关的服务程序，进行分层设计与实现；并在 UI 界面的基础上将服务程序和多线程处理程序等等加入到系统中。

4、完成系统的代码编写，进行系统的安装与运行以及各个功能的展示。

但因个人能力和时间有限，还有很多问题需要解决：

- 添加和删除分组，接收好友添加请求，删除好友；
- 群聊；
- 更改消息通知的方式。

相信随着研究的进一步深入，这些问题会逐步得到解决。

经过这个毕业设计的磨练，对移动通讯行业有了更深入的了解，也有了更多的知识来充实自己的大脑，知道了即时通讯已经成为语音及文本的在线即时通信的主要技术，它的特点决定了其在 Internet 应用、在线协作以及未来移动商务中将扮演更为重要的角色。手机网络平台也会随着移动通信技术的迅猛发展而逐渐拓展，因此移动即时通信也有了更强有力的支撑平台。随着该技术的发展，实时多媒体技术也将会移植到移动即时通信应用中来。相信基于 Android 平台的即时通讯系统会有着很好的应用前景。

参考文献

[1] Ed Burnette 著. 田俊静,张波,黄湘情 等译. Android 基础教程(第 3 版)[M], 2011-6

[2] 吴亚峰 索依娜 等著. Android 核心技术与实例讲解[M]. 电子工业出版社, 2011-6

[3] 陈钊. Android 程序主要组成部分概述[J]. 中国新技术新产品. 2011(17):42

[4] 李宁 著. Android 开发完全讲义(第二版)[M]. 水利水电出版社. 2012

[5] P Saint-Andre Ed. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence[EB/OL]. http://xmpp.org/internet-drafts/draft-saintandre-rfc3921bis-07.html, October 24, 2008

[6] 庞怡 许洪光 姜媛. 即时通讯工具现状及发展趋势分析[J]. 科技情报开发与经济，2006(10):169-170

[7] 剧忻. 基于 MINA 开发高性能网络应用程序——以实现 XMPP 协议 Openfire3.3.3 为例[J]. 重庆工学院学报（自然科学版）. 2008, 22(10):121-125

[8] Jack Moffitt 著. 杨明军 译. XMPP 高级编程——使用 JavaScript 和 jQuery[M]. 清华大学出版社, 2011-6

[9] 卡尔佛特 多纳霍 著. 周恒民 译. Java TCP/IP Socket 编程(原书第二版)[M]. 机械工业出版社, 2009-1

[10] 张彦 夏清国. Jabber/XMPP 技术的研究与应用[J]. 科学技术与工程. 2007, 7(6)

[11] Jason Kichten 著, 刘建华译.用基于 XML 的即时消息开发 Jabber[EB/OL]. http://www.bitscn.com/pdb/dotnet/200701/88917.html

[12] 潘凤 王华军 苗放 李刚. 基于 XMPP 协议和 Openfire 的即时通讯系统的开发[J]. 计算机时代. 2008(3)

[13] http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/

[14] 马志强. 基于 Android 平台即使通信系统的设计与实现[D]. 北京交通大学，2009

[15] Peter S A.XMPP Instant Messaging and Presenee.RFC 3921 [E],2004

[16] Pankaj Jalote 著.罗飞 邵凌霜 等译.软件工程导论[M].清华大学出版社, 2012

[17] Wei-Meng Lee 著. 何晨光 李洪刚译. Android 编程入门经典[M]. 清华大学出版社, 2012-4

[18] 张海燕. Java 多线程技术在手机互联网中的应用[J]. 农业网络信息, 2008(3): 97-98

致谢

该课题是本人在河海大学学习期间完成的。在这里，最先要感谢我的导师娄渊胜教授，感谢他在学术上给予我的悉心指导。在学习期间，您的严谨的治学态度、渊博的知识和孜孜不倦的教诲给我留下了极为深刻的印象，在认真完成毕业设计的同时，我的专业技能和独立工作的能力也得到了增强。此次毕业设计也让我学会了许多做人的道理。在本论文的写作过程中，娄教授在论文结构安排和内容上给我提出了许多宝贵意见。我在河海大学这四年中，不仅学到了很多的知识，也懂得了很多做人的道理。感谢计算机与信息学院的所有老师，感谢他们对我无私的教导和帮助。衷心感谢学长们对我的引导，谢谢你们和我一起查阅资料，陪我一起研究。

这里也要感谢网络这个平台，让我在对系统关键技术的进展愁眉苦脸的时候，找到了解决问题的方案，这里有许多论坛，论坛里有许多有智慧的朋友，都是值得学习和借鉴的。

李艳

2013 年 6 月

附录

外文文献

Design and Implementation of Web Instant Message System Based on XMPP

Bai Xuefang　　Yang Ming

Dept. of Transp. Manage., Dalian Maritime Univ., Dalian, Chin

IEEE June 2012 (83-88)

II. RELATED BACKGROUND KNOWLEDGE

A. XMPP Introduction

Extensible Messaging and Presence Protocol (XMPP) is an open-standard communications protocol for message-oriented middleware based on XML (Extensible Markup Language) [1], which powers a wide range of applications including instant messaging, presence, media negotiation, collaboration, lightweight middleware, content syndication, and generalized XML routing. The protocol was originally named Jabber, and was developed by the Jabber open-source community in 1999 for near-real-time, extensible instant messaging (IM), presence information, and contact list maintenance [2]. XMPP is open, flexible and extensible, making it the protocol of choice for real-time communications over the Internet, regardless of differences in operating systems and browsers. It enables the reliable transport of any structured XML data between individuals or applications, including RPC and SOAP calls. Numerous mission-critical business applications use XMPP, including chat and IM network management and financial trading. With inherent security features and support for cross-domain server federation, XMPP is more than able to meet the needs of the most demanding environments [3]. Eventually, XMPP is expected to support IM applications with authentication, access control, a high measure of privacy, hop-by-hop encryption, end-to-end encryption, and compatibility with other protocols.

B. XMPP Network and Architecture

In practice, an XMPP network is composed of the XMPP clients and servers that can reach each other on a single computer network. The biggest XMPP network is available on the Internet and connects public XMPP servers. However, people are free to create private XMPP networks within a single company's internal LAN, on secure corporate virtual private networks. Within each XMPP network, each user is assigned

a unique XMPP address. As a result, end-to-end communication in XMPP is logically peer-to-peer but physically client-to-server-to-server-to-client, as illustrated in the below figure.

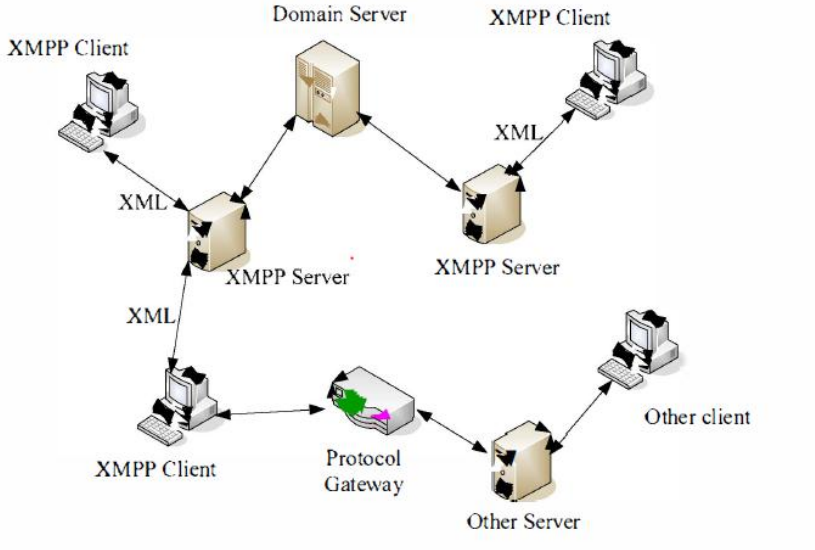

图 1　XMPP network

In the above network, XMPP server acts as an intelligent abstraction layer for XMPP communications. It provides basic messaging, presence, and XML routing features. Its primary responsibility is to manage connections or sessions for other entities, in the form of XML streams to and from authorized clients, servers, and other entities. Besides, it also helps to route appropriately-addressed XML stanzas among such entities. Most XMPP-compliant servers also assume responsibility for the storage of data that is used by clients [4]. While most clients connect directly to a server over a TCP (Transmission Control Protocol) connection and use XMPP to take full advantage of the functionality provided by a server and any associated services. Multiple resources may connect simultaneously to a server on behalf of each authorized client, with each resource differentiated by the resource identifier of an XMPP address [5]. A gateway is a special-purpose server-side service whose primary function is to translate XMPP into the protocol used by a foreign (non-XMPP) messaging system, as well as to translate the return data back into XMPP.

C. Addressing Scheme and Identifier

The core of XMPP routing is an internationalized logical addressing scheme that is best represented as node@domain/resource. In the Jabber IM system this scheme is referred to as the Jabber ID (JID) [6]. The domain portion (e.g. mydomain.com) is typically the fully qualified domain name of the server, component or plug-in. Like Email addresses, in Simple Mail Transfer Protocol, servers connect with one another

on behalf of users. In XMPP, the node portion can denote an IM user, an application or a service. The resource portion is a connection identifier that lets a single user be logged on multiple times simultaneously.

Every user on the network has a unique JID. To avoid requiring a central server to maintain a list of IDs, the JID is structured as username@example.com [7]. Each resource may have specified a numerical value called priority. Messages simply sent to username@example.com will go to the client with highest priority. Besides, JIDs without a username part are also valid, and may be used for system messages and control of special features on the server. A resource remains optional for these JIDs as well.

D. XML Streams and XML Stanzas

a. XML Stream

An XML stream is a container for the exchange of XML elements between any two entities over a network [8]. Once a long-lived TCP connection between client and server has been established, an XML stream between client and server is initiated. An XMPP XML stream takes the form of a lengthy document that records communication between client and server each time a message has been passed. Streams are initiated by sending an opening <stream:stream> tag. After the opening tag establishes the XML stream, messages of the following 3 types of XML stanzas are exchanged between entities: <presence/>, <message/> and <iq/>. By the time the connection is closed, there will be one valid XML document for each stream [9].

b. XML Stanza

An XML stanza is a discrete semantic unit of structured information that is sent from one entity to another over an XML stream. An XML stanza exists at the direct child level of the root <stream/> element and is said to be well-balanced if it matches the production [10] content of [XML]. An XML stanza may contain child elements (with accompanying attributes, elements, and XML character data) as necessary in order to convey the desired information. The only XML stanzas defined herein are the <message/>, <presence/>, <iq/>, and we have generalized their features as follows:

- <presence/>

The presence stanza enables entities to share their availability and status with others. For example, an instant messaging client allows you to see whether your contacts are online or not and what their current status (away, busy, do not disturb, etc.) is. Presence subscriptions allow you to determine who can see whether you are online or not. In order for others to see whether you are online or not, they must send

you a subscription request and you must approve it. Unlike many IM systems, XMPP presence systems are directional.

- <message/>

Message stanzas are used to send messages between entities. They are similar to e-mail messages but different from <iq/> messages in that after a message has been sent, the sender has no way of knowing whether it has been received by the intended recipient or not. In an Instant Messaging context, messages are what users' text between each other. The five types of messages are normal, chat, group chat, headline and error.

- <iq/>

Unlike Message and Presence, Info/Query (IQ) stanzas are two-way. They are intended for use when one entity queries another with the expectation of a response. A get or set IQ stanza must always receive a reply from the recipient. In Instant Messaging systems, IQ is primarily used for roster management. The four types of IQ messages are get, set, result and error.

III. OVERALL ANALYSES AND DESIGNS OF EIMS

A. Basic Requirements of EIMS

EIMS are required to provide basic functions of business real-time interaction among people, such as instant messaging, file transfer, roster management and conference room and so on. In this regard; enterprises can complete the specific modular client customization according to their own business demands. In addition to the basic functional requirements, EISM also need to ensure efficiency, integrity and security of communication data, and can support multi-protocol interoperability, so by analyzing and investigating the overall demands of the existing EISM, according to the demand conditions of business web application, the article has summed up the basic business requirements as follows:

- Simple to apply and friendly interface. EIMS is different from the general public instant messaging products, which concerns much about the corporate image and office efficiency, thus the basic functions of the operating process should be simple, easy to understand and grasp, and the interface should be designed to maintain commercial and seriousness image for enterprise, avoiding excessive entertainment applications, so as to create a professional real-time communication platform.
- Safe and manageable, adapt to the enterprise's internal network. Enterprise-class products must have a high safety and controllability. The

information transferred between different business users must be encrypted, and business managers can implement unified management and monitoring for internal users by instant messaging server, besides, the installed XMPP server should support internal communication within the enterprise without accessing the Internet network.

- Integration of the existing office management platform. In order to satisfy the requirement of EIMS of the cooperative office, when enterprises choose to apply real-time communication systems, typically require instant communication system to be integrated into OA, ERP and other office management and application system, with which can enable real-time business progress, make business processes more timely and realize combination of communication and management in business.
- Provide secondary development interface. When select the instant messaging system, enterprises consider much more about the system upgrade in future maintenance, so EIMS are expected to be developed with good structure programming language and open interfaces, which can enable enterprises to implement secondary development and rich client function by applying the available interfaces. Support multi-protocol communication, interoperability. IM software developers, such as MSN and GTalk, are all developing their own protocols and standards due to the commercial purpose, which was impossible for transferring data between different protocol-based IM software. Thus, the modem EIMS should break the limitation of IM protocols; interconnection with other communication products is important aspect when choosing a business instant messaging system.

B. Design Framework of EIMS

As shown in Figure 2, the paper has designed the overall framework of EIMS based on the XMPP protocol and open source software, the Client-side is displayed on the left part, by logging in the XMPP client, IM users can establish a connection with XMPP server and communication between each other. Regarding to our project, the XMPP client was developed based on an open source software and XMPP protocol, in addition to the basic function provided and supported by XMPP standard protocol, such as instant messaging, roster management and presence management and so on, secondary development for enriching the function in client need to be satisfied when deciding a EIM client, the required functions such as file transfer, conference room and broadcast have been listed in the above figure, which will also provide a more

efficient and convenient operation mode for end users. While the right part is the server-side, by which the system administrator can operate and control database of both EIM server and other application servers by management module, for example, the creation and integration of system user, user group management, authorization and role management for conference rooms. Besides, the server-side can design and implement plug-in functionality through plug-in development according their own requirement and situation. By configuring the gateway provided by XMPP, the company can exchange information between internal instant messaging systems with external instant messaging systems.

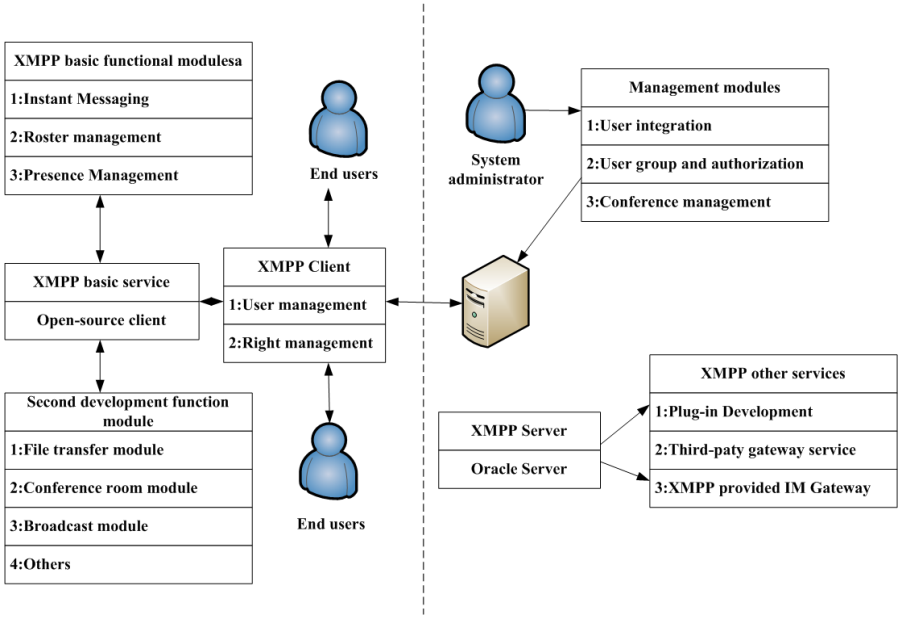

Figure 2. Framework of EIM system based on XMPP and open source software

C. Developing Platform for EIMS

a. XMPP-based Openfire Server

It's been a long time since the days when IBM and Microsoft dominated the EIM market. Nowadays, there are a range of EIM platforms in addition to IBM Lotus Sometime and Microsoft Office Live Communications Server. And some enterprises have tried and applied such kind of EIM for their office real-time communication, however, in this regard; the enterprises found that there were still some restrictions when the requirement changed or the system user database changed, so the research and application of XMPP-based servers have become more and more popular because of overcoming these problems, there are also many XMPP-based EIM servers available on the market, Openfire, which is written in Java, implements most features of XMPP, developed and maintained by Jive Software open-source organization, is

one of such server.

Openfire is open source real time collaboration (RTC) server that enables anyone to create communication environments like MSN Messenger, GTalk, etc. With an easy to setup-administer yet rock-solid security-performance system, Openfire application can run on both Windows & Linux/Unix servers and it comes with various plug-ins for function extension. By now, the latest version is Openfire3.7.l released on October 2, 2011 [11], Openfire hasn't used the popular technical architecture (SSH), simply use JSP and JavaBeans instead, its own system design has good programming structure, and the management console supports plug-ins to enrich the server's functions. Under Openfire's programming design, the JSP page can directly call the business processing logic class instance method by packaging request object so as to pass the variables and jump the pages, and adopt decorative framework to display JSP pages; thus, Openfire has better flexibility and configurability in EIM server development and application. In this article, Openfire is designed and developed as EIM server, the mainly features and advantages of Openfire are as follows:

- Completely open-source GPL license, based on mature XMPP protocol. Openfire using the open XMPP protocol, therefore, the server can support the XMPP protocol IM client software, which can achieve interoperability with other instant messaging software based on the protocol.
- Platform independent, pure Java. Based on the Java programming language code, Openfire has a clear structure and programming specifications, easy to understand and secondary development, which can also satisfy the development platform of both Windows and Linux operating systems.
- Web-based administration panel and management interface. The Openfire internal integration of resin web server, so users can design a Web-based management program or client program according their own enterprise requirement.
- Based on the plug-in development framework, high scalability; Openfire has achieved plug-in mechanism, which has facilitated its service expansion. When the server startup, Openfire will read all the files under the plug-in directory to determine whether there is a new package of plug-ins, once discovering the new plug-in deployment, Openfire will visit all classes this plug-in included, and determine whether there is the

supporting plug-in interface class, if there is, Openfire will load and run the plug-in. The plug-in extension mechanism applied by Openfire has satisfied the requirement for users to design and expand the server function perfectly.

- Support thousands of concurrent users with stable function. Along with the procedure of updating multiple versions and experiences of lager number users practical utilizing, the basic functions of Openfire have been gradually stabilized, the process of installation and configuration are very simple, in terms of user capacity, and a server can support thousands of users to log in, with a strong data communication capabilities.

b. Openfire Messaging Process

XMPP protocol served as IM, which core function is messages transmission. Thus the most important core code for Openfire are listening and processing client message packet. The above figure has shown the message packet process of Openfire, and this network process mainly includes message listener service, message encapsulation and message packet processing. In this paper, we would like to analyze the process by introducing some important classes and services.

- Firstly, acquiring the current session class Stanza-Handler, which is also assumed as business process class (Client-Stanza-Handler class), instantiating XMPP Packet Reader (parsing and reading XML data, packaging packet), calling Client-Stanza-Handler's process () method and entering into, then according to the judgment of XML packet, determining which kind of XML element should be processed, thus entering into the process(doc) method.
- Secondly, if the requested XML package is IQ type, same procedure with message and presence type, in this regard, the process will enter into processIQ(packet) method and the router. route (packet) method will be called to route this kind of packet, then enters into the specific IQRouter class handle (packet) method, by running this method, the process will first execute IQHandler class process () method, then enters into IQRegisterHandler class process () method, packaging XML element into a user object, calling the method of underlying database and storing XML format data, and generate a reply packet for returning to client.
- Finally, Session. Process (reply); return reply to the client. The instance

class of session is LocalClientSession, entering into the parent class (LocalSession) process () method. Calling deliver () method and return to LocalClientSession class deliver () method. Entering into and calling Conn. Deliver (packet), then entering NIOConnection class and determining whether the connection is disconnected, if connect, continue down. Thus the XML packet can be received and processed between client and server.

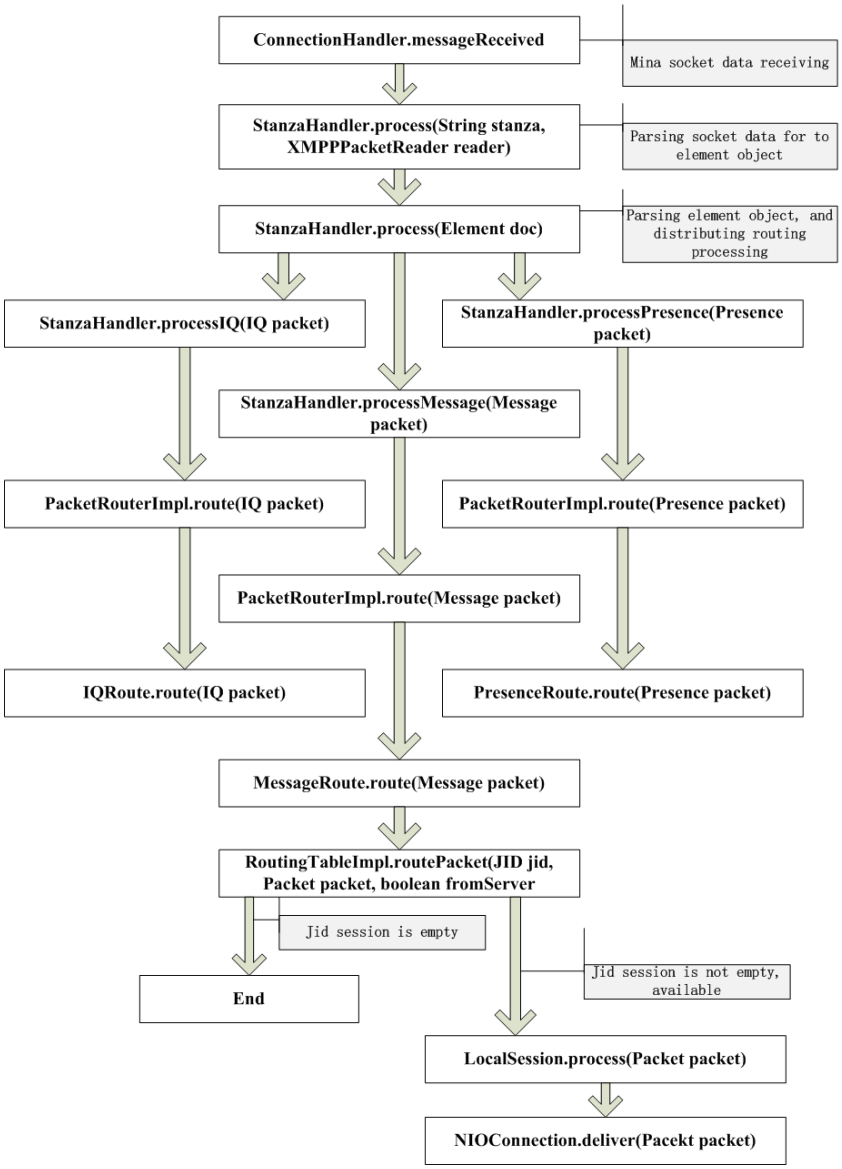

Figure 3. Openfire message packet sending and receiving processes

D. Message Processing Between XMPP Server and Openfire Client

In this real-time communication system, the information handling process between client and server has been shown in the below figure. Firstly, by accessing the IM web interface, users will be requested to input username and password information, by which operation can trigger a series of relevant listener events, then the events functions will be called to handle the information flows and packaged them into XML

structured data, in this regard, the message will be transferred to message processing module in XML format; Secondly, when the message processing module receives the packaged data, it will append some attributes on them, such as sender, receiver and type and so on, thus the XML stream can be processed and sent to Openfire server by transceiver processing module. Finally, when Openfire server received the XML stream, it will check the session pool and determine whether this stream has established a connection with server, if the connection sessions do exist, the server will parse the XML stream and generate XML packet, then the generated packet will be pressed into the packet queue waiting to be monitored and handled, and Openfire will process the packet and return a result in form of XML packet, before sending this packet to client, the session's existence judging for the XML stream is also required by the server, the server can only send message to the connected client in session pool. Similarly, message handling process from server to client is exactly the reverse version of the above mentioned process. In short, in order to set up XMPP-based EIM, the IM client must acquire a unique session ID to connect server, and the transformed message format must be structured in XML format and in strict compliance with XMPP, besides, the client should monitor and parse the XML stream correctly and efficiently.

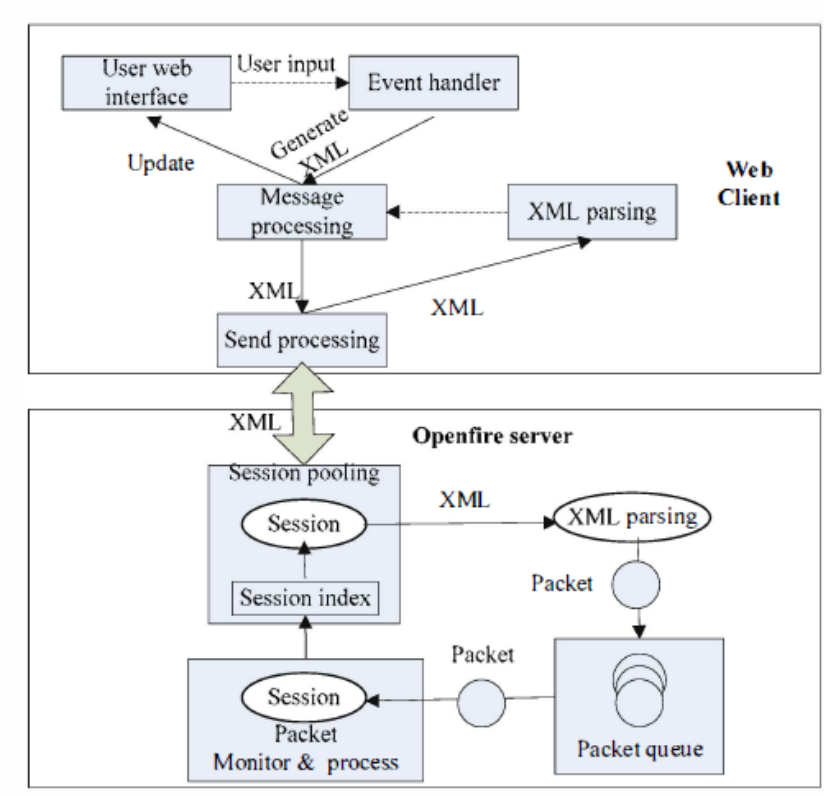

Figure 4. Message handling process between client and server

中文翻译

基于XMPP的网络即时消息系统的设计和实施

白雪芳　杨明

第二部分 相关背景知识

A . XMPP介绍

可扩展的消息与出席协议(XMPP)是一种以 XML(Extensible Markup Language) 为面向消息的中间件的开放标准的通信协议[1]。XML 的应用范围包括即时通讯、出席、媒体谈判、合作、轻量级中间件、内容聚合和广义 XML 路由。XMPP 协议最先是被 Jabber 所命名，是 Jabber 开源社区在 1999 年为实时的、可扩展的即时通信(IM)、出席信息和联系人列表维护而发展的[2]。XMPP 协议是开放的、灵活的和可扩展的。这使得它成为网络实时交流的首选协议，并且不考虑操作系统与浏览器。它保障了了结构化的 XML 数据的在个体或是应用程序之间的可靠传输，包括 RPC 和 SOAP 调用。许多的关键业务应用程序大多使用 XMPP 协议，包括聊天和即时通讯网络管理以及金融交易。伴随着固定的安全特性和跨域服务器联盟的支持，XMPP 希望能够满足最苛刻的要求[3]。最终，XMPP 协议将与其它协议一起被用来支持即时通讯应用程序与身份验证、访问控制、高度隐私、逐跳加密、端到端加密。

B．XMPP 网络和体系结构

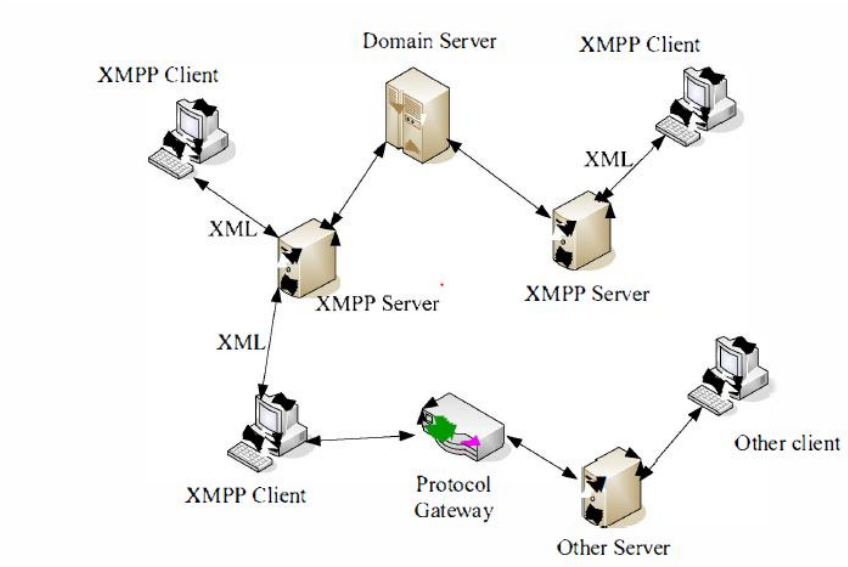

图 1　XMPP 网络

在实践中，一个 XMPP 网络是由 XMPP 客户机群与可以提供互连的服务器组成。在因特网上可以看到最大的 XMPP 网络，其连接公共的 XMPP 服务器。

人们也可以在一个公司的内部局域网内自由创建私人的 XMPP 网络，来确保企业虚拟私人网络。在每个 XMPP 网络内，每个用户分配到一个唯一的 XMPP 地址。因此，XMPP 的端到端通信是逻辑上对等的，也就是说物理上的客户机到服务器和服务器到客户端是对等的，如图 1 所示。

在图 1 的 XMPP 网络中，XMPP 服务器在 XMPP 通信中充当一个智能抽象层。它提供了基本的消息传递、出席信息和 XML 路由功能。它的主要责任是管理连接和对其他客户端的会话以及在授权客户端、服务器和实体间以 XML 流的形式往返。此外,它还有助于解决 XML 节在实体中的适当路线的选取。大多数 XMPP 兼容的服务器还承担存储经常被客户端使用到的数据的任务[4]。然而，大多数客户端直接与服务器建立 TCP(传输控制协议)连接并使用 XMPP 协议来充分利用一个服务器和相关联的服务器提供的功能服务。尽管每个资源的在 XMPP 地址[5]中的资源标识符不同，多个资源也可以同时连接到一个服务器来代表每个授权客户端。网关是一个专用的服务器端的服务，它的主要功能负责将 XMPP 协议翻译成非 XMPP 协议信息系统中，然后再翻译返还的数据到 XMPP 协议中。

C．解决方案和标识符

XMPP 路由核心是一个国际化的逻辑寻址方案，表示为 node@domain/resource。在 Jabber IM 系统里，被称为 Jabber ID(JID)[6]。domain(例如 mydomain.com)是服务器、组件或插件的全称域名。如同简单邮件传输协议里的电子邮件地址，服务器代表用户群连到另一端，在 XMPP 中，node 部分表示一个 IM 用户、应用程序或服务，resource 部分是一个连接标识符，允许一个用户同时使用不同的设备登录。

网络上的每个用户都有唯一的 JID。为了避免请求中央服务器维护一个 id 列表，JID 结构设定为 username@example.com[7]。每个资源可能指定一个数值称为优先级。发送到 username@example.com 的消息，优先级高的先发送到客户端。此外。没有用户名部分的 JIDs 也是有效的，它可以用于发送系统消息和控制服务器上的特殊功能，现在也保留着这样的一些 JIDs。

D．XML 流和 XML 节

a．XML 流

XML 流是一个为了在网络上不同实体间交换 XML 元素的容器[8]。当一个长期的客户端与服务器之间的 TCP 连接建立后，客户端和服务器之间的 XML 流也被初始化。XMPP 中 XML 流展现的形式是一个冗长的文档，每次当有消息通过，它则记录客户机和服务器之间的通信。XML 流信息开始于<stream:stream>标签。展开 XML 流标签后，其余的三个 XML 元素标签是：<presence/>，<message/>和<iq/>。在连接关闭时，对于每个流将会有一个有效的 XML 文档[9]。

b．XML 节

XML 节是一个从一个实体发送到另一个实体的 XML 流的结构化信息的离散的语义单元,。XML 节存在于根<stream/>元素的直接子层次中,应该符合 XML 文档的结构层次[10]。为了传达所需的信息，XML 节应该包含子元素(附带属性、元素和 XML 字符数据)。XML 节是指<message/>，<presence/>和<iq/>，它们的描述大体如下所示：

- <presence/>

<presence/>节使实体之间共享可用性和状态。例如，不管你是否在线，即时通信客户机都能够让你看到他们的当前状态(离开、忙碌、请勿打扰等等)。Presence 订阅允许你决定谁可以看到你是否在线。为了看到你是否在线，他们必须向你发送一个订阅请求，你必须同意。不同于许多即时通讯系统，XMPP 协议的 Presence 是定向的。

- <message/>

<message/>节是用来在实体间发送消息。它们类似于电子邮件消息，但是不同于<iq/ >消息。在消息被发送后，发送方没有办法知道它是否已经被预期的接收者接收。在即时通信的上下文中，消息是用户彼此之间的文本信息。消息的五个类型分别是 normal、chat、group chat、headline 和 error。

- <iq/>

不同于<message/>和<presence/>，Info/Query (IQ)节是双向的。他们适用于知道预期的结果的前提下一个实体向另一个实体发送查询的情况。获取或设置 IQ 节必须始终从接收者那得到回复。在即时通信系统中，IQ 主要是用于花名册管理。IQ 的四种类型分别是 get、set、result 和 error。

第三部分　总体分析和设计的 EIMS

A．EIMS 的基本要求

EIMS 需要提供人们之间业务实时交互的基本的功能，如即时消息、文件传输、执勤人员表和会议室等。在这方面,企业可以根据自己的业务需求完成特定的客户定制。除了基本的功能需求，EISM 还需要确保效率、完整和通信数据的安全以及支持多媒体协议的交互。因此通过分析和研究现有的 EISM 的总的需求，根据商业 web 应用程序的需求条件，本文总结了基本的业务需求，如下所示:

- 简单的应用和友好的界面。EIMS 不同于一般的即时通讯产品，它涉及很多企业形象和办公效率，因此操作流程的基本功能应该简单，易于理解和掌握。还有从企业方面考虑，界面设计应保持商业和严肃性形象，避免过度娱乐程序，以便创建一个专业的实时通信平台。
- 安全的和易管理的，适应于企业的内部网络。企业级的产品必须具有较高的安全性和可控性。不同业务用户间的信息传输必须加密，业务经理可以通过即时消息服务器实现统一的管理和监测内部用户。另外，已安

装的 XMPP 服务器应该能够实现在不连接 Internet 的情况下在企业内部通信。

- 集成现有的办公管理平台。为了满足合作办公室的 EIMS 需求，当企业选择应用实时通信系统时，通常需要即时通信系统集成到 OA、ERP 和其他办公室管理和应用系统中。这可以使实时业务进展成为可能，使业务流程更加及时以及实现沟通和管理在业务上的结合。
- 提供二次开发接口。当选择即时通讯系统时，企业更多考虑的是在以后维护时的系统升级，所以 EIMS 应具有良好结构的编程语言和开放接口，这样可以使企业通过运用可用的接口实现二次开发和丰富客户端功能。支持多协议通信，互操作。IM 软件，如 MSN 和 GTalk，都在发展自己的协议和标准来适应商业目的——数据在不同 IM 软件协议之间的传输是不可能的。因此，现代 EIMS 应该打破 IM 协议的限制；在选择企业即时通讯系统时，与其他通信产品的互连是很重要。

B．设计框架的 EIMS

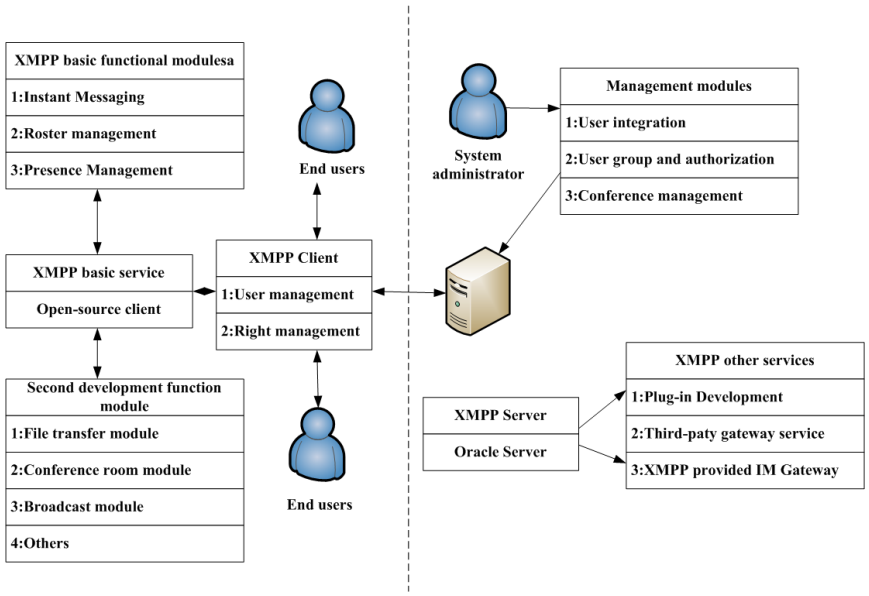

图2　基于XMPP和开放源码软件的EIM系统的框架

如图 2 所示，本文设计了基于 XMPP 协议和开放源码软件的 EIMS 的整体框架，客户端显示在左边的部分，通过登录客户端，IM 用户可以建立与 XMPP 服务的连接并彼此交流。关于我们的项目，XMPP 客户端是基于一个开放源码软件和 XMPP 协议开发的。此外，除了基本的功能和标准协议，如 XMPP 即时消息、名单管理和出席信息管理等等，为丰富客户端的功能的二次开发还需要满足的有 EIM 客户机的决定以及所需的功能，如文件传输、会议室和广播已经被列在上面的图中，它还将为最终用户提供一个更有效和方便的操作方式。右部分是服务器端，通过服务器系统管理员可以在管理模块部分操作和控制 EIM 服务器

和其他应用程序的数据库，例如创建和集成的系统用户、用户组管理、授权和角色管理会议室。此外，开发者可以根据自己的情况和需求在服务端进行插件开发。通过配置 XMPP 提供的网关，公司可以在内部即时通讯系统与外部的即时消息系统之间交换信息。

C．开发平台 EIMS

a.基于 XMPP 的服务器 Openfire

自从 IBM 和微软主导市场 EIM 已经过了很长的时间。如今，除了 IBM 的 Lotus Sometime 和微软的 Office Live Communication Server，还有一系列的 EIM 平台。一些企业已经尝试和应用这种 EIM 作为他们的办公室实时通信。然而，在这方面,企业发现当需求改变和用户数据改变时仍有一些限制，因此，为了克服了这些问题，研究和基于 XMPP 的服务器的应用已经变得越来越流行。在市场上，有许多基于 XMPP 协议的 EIM 服务器。Openfire 是由 Java 编写的、使用 XMPP 协议，由 Jive Software 开源组织开发和维护的服务器。

Openfire 是开源的实时协作(RTC)服务器，它可以让任何人创造交流环境，像 MSN 和 GTalk 等等。通过一个容易设置管理然而可靠的安全性能系统，Openfire 可以在 Windows 和 Linux / Unix 服务器上运行，Openfire 有多种插件扩展。现在，最新的版本是 Openfire 3.7.1，发布于 2011 年 10 月 2 日[11]。Openfire 没有使用流行的技术架构(SSH)，仅简单地使用 JSP 和 JavaBeans。它自己的系统设计具有良好的编程结构，管理控制台支持各种插件来丰富服务器的功能。在 Openfire 的编程设计下， JSP 页面可以通过包请求直接调用业务处理逻辑类的实例方法以便变量传递与页面跳转，采用装饰框架显示 JSP 页。因此，在 EIM 服务器开发和应用中，Openfire 具有更好的灵活性和可配置性。在本文中，EIM 服务器采用 Openfire。其主要的特色和优势如下：

- 完全开源的 GPL 许可证，基于成熟的 XMPP 协议。Openfire 使用开放的 XMPP 协议，因此，服务端可以支持所有基于 XMPP 协议的 IM 客户端软件间的交流。
- 平台独立、纯 Java。基于 Java 编程语言，Openfire 有一个清晰的结构和编程规范，易于理解和二次开发，还适用于 Windows 和 Linux 操作系统。
- 基于网络的管理面板和管理接口。Openfire 内部集成了 resin web 服务器，因此用户可以根据自己的企业需求设计一个基于 web 的管理程序或是客户端程序。
- 基于插件开发框架、高可伸缩性。Openfire 已经实现了插件机制以促进其服务扩张。当服务器启动时，Openfire 将读取插件目录下的所有文件以确定是否有一个新的插件。一旦发现新的插件，Openfire 将访问此插件内的所有类，并确定是否有支持该插件的接口类，如果有，Openfire 将加载并

运行该插件。插件扩展机制已经使 Openfire 满足了用户的设计和服务端的功能扩展的需求。

- 支持数千个并发用户，功能稳定。随着版本的更新过程和用户的实际经验，Openfire 的基本功能已经逐渐稳定，安装和配置的过程非常简单。根据用户容量，一个服务器可以支持数千用户登录和强大的数据通信。

b. Openfire 消息传递过程

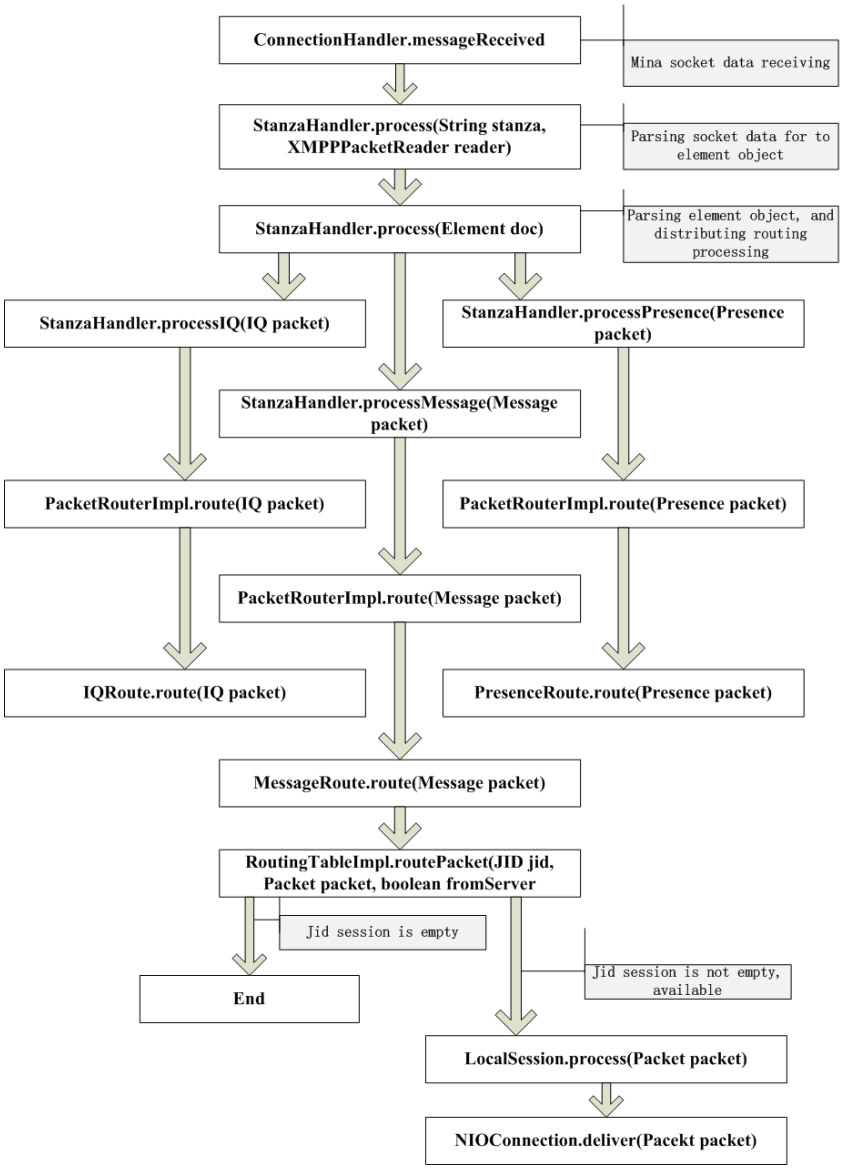

图 3　Openfire 消息数据包发送和接收过程

XMPP 协议作为 IM，核心功能是消息传输。因此对于 Openfire，最重要的核心代码是接听和处理客户消息包。上图显示 Openfire 消息包过程，这个网络过程主要包括消息侦听器服务、消息封装和消息包处理。在本文中，通过引入一些重要的类和服务，我们将分析这个过程。

- 首先，获取当前会话类节处理程序，假定为业务流程类(Client-Stanza-Handler)，实例化 XMPP Packet Reader (解析和阅读 XML 数据，包装包)，调用 Client-Stanza-Handler 的 process()方法并进入，然后根据判断 XML 包，确定哪些类型的 XML 元素应该被处理，从而进入 process(doc)方法。
- 其次，如果请求的 XML 包是 IQ 类型，和 message 和 presence 类型的流程相同，流程将执行 processIQ(packet)方法和 router 方法。router()方法将被调用来寻找这种包，然后，执行指定的 IQRouter 类的 handle()方法中，通过运行该方法，流程将首先执行 IQHandler 类的 process()方法，然后执行 IQRegisterHandler 类的 process()方法，封装 XML 元素到一个用户对象，并生成一个应答包，返回给客户端。
- 最后，会话。Process (reply)：返回给客户端的答复。这个会话的实例类是 LocalClientSession，执行父类(LocalSession)的 process()方法。调用 deliver()方法以及返回 LocalClientSession 类的 deliver()方法。执行和调用 Conn.deliver(packet)，然后执行 NIOConnection 类并判断连接是否断开。如没有断开，继续执行。所以，XML 包可以客户端和服务器之间被接收并处理。

D．XMPP 服务器和 Openfire 客户端之间的消息处理

图 4 体现了本即时通信系统中客户端和服务器之间的信息处理过程。首先，通过访问 IM web 界面，用户将被要求输入用户名和密码信息，之后才可以执行一系列的相关事件。然后事件方法被调用来处理消息流动和封装消息为 XML 结构数据，之后消息将以 XML 的格式传输到消息处理模块。其次，当消息处理模块接收数据包时，它将添加一些额外属性，比如发送方、接收方和类型等等，因此 XML 流可以通过发送处理模块被处理和发送到 Openfire 服务器。当 Openfire 服务器收到 XML 流时，它将检查会话池并确定这个流是否与服务器建立了一个连接，如果连接会话仍旧存在，服务器将解析 XML 流和生成 XML 包，然后生成的数据包将被压入到数据包队列中等待被监控和处理，接着 Openfire 将处理这个包并以 XML 的格式返回一个结果。在发送这个数据包到客户端之前， XML 流的会话的存在判断对于服务器是必要的，服务器只能发送消息到会话池中已经有连接的客户端上。同样地，从服务器到客户端的信息处理过程与上述过程是完全相反的。总之，为了建立基于 XMPP 的 EIM，IM 客户端必须获得一个唯一的会话 ID 来连接服务器，传输的消息的格式必须是结构化的 XML 格式并且严格按照 XMPP 协议，此外，客户端应该正确和有效地监控和解析 XML 流。

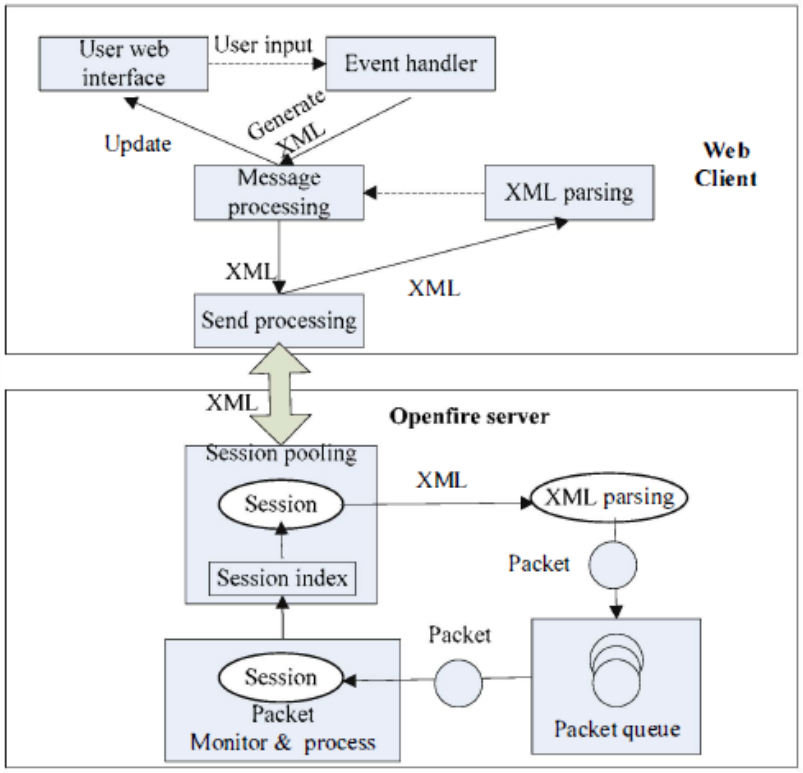

图 4　消息在客户端与服务端的处理流程