四川大學

本科生毕业论文（设计）

题　　目 基于 Linux 的聊天工具设计与实现

学　　院 软件学院

专　　业 软件工程

学生姓名 张伟

学　　号 0743111229 年级 2007

指导教师 梁刚

教务处制表

二O一一年五月二十日

基于 Linux 的聊天工具设计与实现--Ztalk

软件工程

学生 张伟 指导老师 梁刚

[摘要] 本文设计并实现了一个简单的即时通讯软件—Ztalk。本系统包括服务端和客户端两部分。服务端基于流行的开源软件 jabberd2，数据流格式及系统内涉及的技术均采用国际标准，这就确保了 Ztalk 将具有良好的国际通用性。同时，本文通过一个邮件客户端的设计提供了 Jabberd2 的一种扩展思路，对其他有类似需求的项目来说有一定借鉴意义。客户端使用 QT 开发，界面友好，使用简便，方便了多个平台的移植。本文按照“界面-控制-数据”的模块结构，详细论述了客户端中聊天，用户管理，账户管理，群聊天等功能的实现，并单独详述了 Linux 远程桌面的实现方案。

[主题词] 即时通讯, Linux, jabberd2, XMPP 协议

One Instant Messaging Tool's Designing and Realizing on the Linux Platform--Ztalk

Software Engineering

Student： Zhang Wei Adviser: Liang gang

[Abstract] This document designed and implemented a simple instant messaging software, including server and client two parts.The server was developed based on the popular open source software jabberd2, the data stream format and the technology involved in the system are of international standards, this ensures that the Ztalk will have a good international versatility. At the same time, the paper designed an e-mail client which provides an extension method of Jabberd2 for other projects with similar needs. The client part used the QT to develop, which made it user-friendly, easy to use and easy to transplant to multiple platforms. This article discussed the implementation of functions such as P2P chat, user management, account management and group chat in detail with the "interface-control-data" module structure sequence, and provided a separate detailed implementation scheme of linux remote desktop control.

[Key Words] Instant Messaging, Linux, Jabberd2, XMPP protocol

目 录

1 前言

1.1 概述

本系统为 Linux 下的即时通讯工具，客户端及服务端均在 Linux 下开发及应用。服务端基于开源软件 Jabberd2，能满足 IM 服务端对于承载力和处理速度的需求。客户端使用 QT 开发，使用简便，界面友好。功能方面除实现了 IM（即时通讯）类工具核心的好友聊天，群聊天，用户管理，账户管理等功能外，在其中嵌入了邮件客户端和 Linux 下的远程桌面两个附加功能。

系统详细功能需求如下：

- 好友管理
  - 支持好友列表，并能正确的自动更新
  - 能显示在线好友，并与不在线好友予以区分
  - 能查找、添加、删除好友
  - 支持自定义好友分组
  - 支持自动好友排序
  - 支持好友来消息提示
- 群管理
  - 能查找、加入、退出群
  - 能创建新群
- 信息及其它管理
  - 能显示个人、好友、群信息
  - 能修改基本的个人信息
  - 支持修改个人在线状态
  - 支持对好友、群添加备注
  - 支持设置隐私权限
  - 支持加入黑名单

- ✧ 支持从好友列表中搜索特定好友
- ✧ 能主面板最小化、最大化、隐藏操作
- ✧ 支持多账户同时登陆，且互不影响
- ✧ 支持屏蔽特定好友消息

➢ 聊天界面

- ✧ 显示输入消息窗口，能及时显示输入的消息
- ✧ 能显示接收到的好友消息
- ✧ 支持设置消息字体
- ✧ 支持发送图片
- ✧ 支持聊天消息离线记录
- ✧ 支持发送、接收文件
- ✧ 支持显示好友登陆 IP
- ✧ 能打开多个聊天窗口，且互不影响

➢ 服务器端

- ✧ 能正确处理客户端登陆
- ✧ 能注册账号
- ✧ 能查询、添加、删除好友
- ✧ 能查询、加入、退出、新建群
- ✧ 能正确实现对个人信息进行维护、修改、更新
- ✧ 支持高并发连结
- ✧ 能正确实现个人账户中好友列表、群列表的维护
- ✧ 对不在线好友发送消息时，能暂时保存聊天消息
- ✧ 能正确维护各登陆账户
- ✧ 实现自动判定登陆账号是否离线

➢ 附加功能

- ✧ 实现远程桌面功能；
- ✧ 实现邮件的收发。

1.2 项目背景

本项目来源于四川大学某实验室实际项目。

原项目要求客户端搭载在手机上，服务端具备足够的承载力，能容纳千人同时在线。由于客户端硬件的特殊性，要求客户端软件尽量小巧精简，将大部分处理过程移交服务端完成，客户端仅提供结果显示，客户除能够完成普通 IM 工具具有的聊天，群聊功能外，还能够收发邮件，及收发短信。考虑到系统的稳定性及通用性，服务端搭载在 Linux 系统上，采用流行的开源 IM 服务器程序—Jabberd2。

本软件即实现上述部分功能。由于以手机为载体将极大限制客户端的能力，拟将客户端同样搭载在 Linux 下，采用 Qt 开发。除去收发短信功能，另添加远程桌面功能作为此版本客户端的特色功能。搭载在 Linux 下的客户端程序将具有手机客户端无法比拟的处理速度和丰富性，实用性和通用性都将更优。

通过本系统的实现，作者期望对当今的 IM 技术做深入了解，学习如何实现一个稳定和良好分层的 IM 服务器，同时对涉及的其他技术，如 XML 解析，流安全与验证等有初步的了解，方便进一步学习。同时，对 Linux 下的图形化编程进行初步的掌握和练习。

1.3 研究内容

本项目的目标为开发一款 Linux 下的即时通讯软件。主要研究内容为：

a) 通过对 Jabberd2 的阅读，了解一个即时通讯软件的制作原理，以及一个高性能服务器程序的编写规范。

b) 通过在 Jabberd2 服务端程序上增加邮件模块，研究在现有的 Jabberd2 基础上进行扩展的可行性，提出一个遵从“开放”原则的较好扩展方案。

c) 通过客户端界面的编写，学习 Linux 下图形界面的编写。

d) 通过实现一个 Linux 下的远程桌面，研究 Linux 桌面的控制技术。

e) 研究远程帧缓冲及 XMPP 等协议实际的实现方式及在现实中的应用。

1.4 名词解释

1) IM：Instant Messaging，即时通讯，IM 工具即为即时通讯工具。

2) Jabberd2：一个国际流行的开源即时通信服务器程序，本系统服务端即是基于 jabberd2 服务器开发。采用 jabberd2 开发，系统将具有更高的可扩展性和通用性，具有极大的优势。

3) XMPP 协议：jabberd2 采用的通讯协议，以 XML 为数据流传输格式，定义于 RFC3920。

4) Jabber 协议：同 XMPP 协议。

5) Jabber 小组： jabberd2 服务器程序的开发小组，也是 XMPP 协议的提议者和开发者。

6) Jabber 用户：使用 jabber 客户端进行即时通讯的客户。

7) XML：可扩展置标语言（英语：eXtensible MarkupLanguage，简称:XML），又称可扩展标记语言，是一种置标语言。置标指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种信息的文章等。如何定义这些标记，既可以选择国际通用的标记语言，比如 HTML，也可以使用像 XML 这样由相关人士自由决定的标记语言,这就是语言的可扩展性。XML 是从标准通用置标语言（SGML）中简化修改出来的。

8) 好友：某一 jabber 用户，可以同其进行即时通讯。此系统中提及的好友概念并不局限于存在于自己好友列表中的 jabber 用户。

9) 群：相当于组，可以将多个 jabber 用户添加到同一个群（组）中，在一个窗口内发送的内容对群组内的所有用户可见。

10) 初始化实体：初始化一个数据流，打开一个会话的实体，通常是客户端。

11) 接收实体：接收一个会话，对初始化实体进行授权，会话管理等服务的实体，通常为特定服务器。

2 相关技术&协议介绍

2.1 XMPP 协议

2.1.1 简述

本节简单介绍 XMPP 协议的基本知识，方便后续对软件设计过程的理解。

XMPP 是一个开放式的 XML 协议，设计用于准实时消息和出席信息以及请求-响应服务。其基本的语法和语义最初主要是由 Jabber 开放源代码社区于 1999 年开发的。2002 年，XMPP 工作组被授权接手开发和改编 Jabber 协议以适应 IETF 的即时消息和出席信息技术。作为 XMPP 工作组的成果，本文定义了 XMPP 1.0 的核心功能；在 RFC 2779 [IMP-REQS] 中指定的提供即时消息和出席信息功能的扩展，定义在 XMPP-IM 协议 [the Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence] 中[4]。

2.1.2 地址空间

每个 jabber 用户都必须有一个唯一的 ID，用于标识用户身份，这个 ID 又被称为 Jabber Identifier 或 jid，一个合法的 jid 必须包含域名（domain identifier），节点名（node identifier），和资源名（resource identifier）。

Jid 的语法定义如下：

jid = [node "@"] domain ["/" resource]

domain = fqdn / address-literal

fqdn = (sub-domain 1*("." sub-domain))

sub-domain = (internationalized domain label)

address-literal = IPv4address / IPv6address

具体阐释参见 RFC3920。

域名（domain）是一个合法 JID 唯一必需的元素，仅有域名的 JID 也是合法的。通常一个域名代表了网关或者服务器所在。域名可以是 IP 地址，且必须是 DNS 可以解析的合法的域名格式，如 example.com 等。

节点（node）是可选的第二 ID，通常用来表示特定服务器上的某一实体。如 zw@cookie 代表 cookie 服务器上的 zw 实体（用户）。

资源（resource）为可选的第三 ID，通常标识特定会话，如设备或者客户端。举例如同一账号分别使用 PSI，Gajim 两个客户端登陆，标识这两个不同会话的即为资源名。

综上，如 JID 为 zw@cookie/psi 可能的意思即是，登陆用户为 cookie 服务器上的 zw 用户，使用的客户端为 psi。

2.1.3 XML 流与 XML 节

XML 流的定义：一个 XML 流是一个容器，包含了两个实体之间通过网络交换的 XML 元素。一个 XML 流是由一个 XML 打开标签 <stream> (包含适当的属性和名字空间声明)开始的,流的结尾则是一个 XML 关闭 L 标签 </stream> 。在流的整个生命周期，初始化它的实体可以通过流发送大量的 XML 元素，用于流的握手(例如 TLS 握手或 SASL 握手或 XML 节（在这里指符合缺省名字空间的元素,包括<message/>,<presence/>, 或 <iq/> 元素)。“初始的流”由初始化实体（通常是一个客户端或服务器）和接收实体（通常是一个服务器）握手，从接收实体来看，它就是那个初始实体的"会话".初始化流允许从初始化实体到接收实体的单向通信;为了使接收实体能够和初始实体交换信息,接收实体必须发起一个反向的握手(应答流). [4]

XML 节的定义: 一个 XML 节是一个实体通过 XML 流向另一个实体发送的结构化信息中的一个离散的语义单位。一个 XML 节直接存在于根元素<stream/>的下一级.任何 XML 节都是从一个 XML 流的下一级的某个打开标签（如 <presence>）开始，到相应的关闭标签（如 </presence>）。一个 XML 节可以（MAY）包含子元素(相关的属性,元素, 和 XML 字符数据等) 以表达完整的信息.在这里定义的 XML 节仅限于<message/>, <presence/>, 和 <iq/> 元素。[4]

基本上，一个 XML 流相当于一个会话期间所有 XML 节的一个信封。我们可以简单的把它描述如下：

|----------

| <stream>

|----------

| <presence>

| <show/>

| </presence>

|----------

| <message to='foo'>

| <body/>

| </message>

|----------

| <iq to='bar'>

| <query/>

| </iq>

|----------

| ...

|----------

| </stream>

|----------

流的初始使用<stream>来标识打开，使用</stream>标识流的关闭。Jabber 中定义了 XML 节<message><presence>和<iq>，详细介绍会在后续章节提及。

下面是一个简单的会话示例[4]：

C: <?xml version='1.0'?>

<stream:stream to=’cookie.com' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>

S: <?xml version='1.0'?>

<stream:stream from=cookie.com' id='someid' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>

…encryption, authentication, and resource binding…

C: <message from='juliet@ cookie.com ' to='romeo@ cookie.com ' xml:lang='en'>

<body>Art thou not Romeo, and a Montague?</body>

</message>

S: <message from='romeo@ cookie.com ' to='juliet@ cookie.com ' xml:lang='en'>

<body>Neither, fair saint, if either thee dislike.</body>

</message>

C: </stream:stream>

S: </stream:stream>

一个不成功的会话：

C: <?xml version='1.0'?>

<stream:stream to='cookie.com' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>

S: <?xml version='1.0'?>

<stream:stream from='cookie.com' id='someid' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>

...encryption, authentication, and resource binding...

C: <message xml:lang='en'>

<body>Bad XML, no closing body tag!

</message>

S: <stream:error>

<xml-not-well-formed xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>

</stream:error>

S: </stream:stream>

2.1.4 流验证

如图 2.1 所示，TCP 连接处于最下层，其上使用了 TLS 保证流安全性，在此基础上使用 SASL 协议验证流，最上层才是 XMPP 的交互过程。

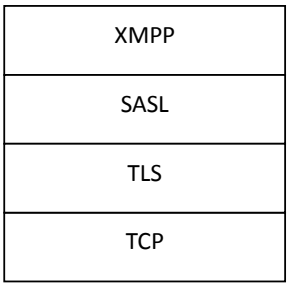

图 2.1 XMPP 层次图

TLS（Transport Layer Security）是一个传输层安全协议的频道加密方法, 模拟了类似的其他"STARTTLS"(见 RFC 2595 [USINGTLS])的扩展,如 IMAP [IMAP], POP3 [POP3], 和 ACAP [ACAP]."STARTTLS"的扩展名字空间是'urn:ietf:params:xml:ns:xmpp-tls'.

一个给定域的管理员可以要求客户端和服务器通信以及服务器之间通信时使用 TLS，或者两者都要求。客户端应该尝试完成 SASL 握手之前使用 TLS，服务器应该在两个域之间使用 TLS 以保证服务器间通信的安全。TLS 详细使用规则参见 RFC3920，RFC2595。

SASL（简单验证和安全层）为 XMPP 提供了一个简单通用的验证流的方法，下文详述。

2.1.5 XML 内置节

1) stream 节属性

<stream>节打开一个数据流，包含的主要属性介绍如下：

To: 仅出现在初始化实体（客户端）发送给接收端（服务端）的流头之中，标识接收端所在的服务器名称。

From: 仅出现在接收实体（服务端）发送给初始化实体（服务端）的应答流中，标识为当前初始化实体授权的服务器主机名称。

Id: 仅出现在接收实体发送给初始化实体的流头之中，标识一个特定的会话。必须具有唯一性和不可预见性，即随机性。

Xml:lang: 标识 XML 流中可读字符缺省的语言，出现在初始化实体发送给接收实体的流头之中。如果此属性没有出现，服务端必须指定默认的语言。

Version: 标识当前 jabber 版本，最少为 1.0,如果 version 属性未出现，服务端认为版本为 0.0. 若版本为 1.0 及以上，客户端必须支持 TLS，SASL 和流错误处理。

2) iq 节属性

to: 表示预期接收者的 JID。如果此接收者不存在，则服务端返回一个错误给发送者。

from: 指示了发送者的 jid。

id: 跟踪会话的标志。必须具有随机性和唯一性且必须存在。

type: 指示 iq 节的意图和详细的上下文信息。type=get 代表是对信息的查询; type=set 代表设置属性; type=result 代表是对 set 或者 get 的返回结果; type=error 代表出现了错误。

xml:lang: 同 stream。

3) message 节属性

to，from, id, xml:lang 同 iq。

Type：指明会话的上下文，并不是必须的属性，建议存在。如果 type 存在，其值必须是以下几个：

Chat—代表一对一的聊天过程，一个兼容的客户端需要提供单人聊天的支持，并且能适量保持聊天记录。

Error—发送了某种类型的错误。

Groupchat—代表了群聊天的过程，群聊天必须支持名册管理和保存适当的聊天记录。

Headline--一个消息可能是由一个递送或广播内容的自动化服务生成的(新闻，体育，市场信息, RSS feeds, 等等.).此类型不属于本系统讨论的范围。

Normal—这个消息是在一对一聊天和群聊之外的单独消息，并且希望接收者尽快回复，不需要保存会话记录。

4) presence 节属性

to，from, id, xml:lang 同 iq。

Type: 指明在线消息类型，建议存在。如果 type 为空，默认 type=’presence’。Type 的属性值必须为下面中的一个。

Presence—用户在线。

Unavailable—用户不在线。

Subscribe—订阅用户的在线状态。

Subscribed—同意对方的订阅。

Unsubscribed—拒绝对方的订阅。

2.2 Qt

2.2.1 概述

Qt 是 Trolltech 公司的一个产品。它提供给应用程序开发者建立图形用户界面应用程序所需的所有功能。Qt 是完全面向对象的，它很容易扩展，并且允许真正的组件编程。自从 1996 年早些时候，Qt 进入商业领域，它已经成为全世界范围内数千种成功的应用程序的基础。Qt 也是流行的 Linux 桌面环境 KDE 的基础。[19]

Qt 是一种通过名为“一次编写，任意平台编译”的方法，实现跨平台图形化编程的 C++框架。Qt 使开发者的应用程序能够不需要重写代码而实现跨平台运行，Qt 可运行的平台有 Windows， Mac OS X, Linux, Solaris, HP-UX 以及任何使用 X11 的 Unix 版本。Qt 的库和工具是 Qtopia 核心的一部分。

2.2.2 Qt 的组成

Qt 提供了一组范围相当广泛的 C++类库，并包含了几种命令行和图形界面的工具，有效地使用这些工具可以加速开发过程。

Qt Designer：Qt 设计器。用来可视化地设计应用程序界面。

Qt Linguist：Qt 语言学家。用来翻译应用程序。以此提供对多种语言的支持。

Qmake：使用此工具可以由简单的、与平台无关的工程文件来生成编译所需的 Makefile。

Qt Assistant：关于 Qt 的帮助文件。类似于 MSDN。可以快速地发现你所需要的帮助。

moc：元对象编译器。

uic：用户界面编译器。在程序编译时被自动调用，通过 ui_*.h 文件生成应用程序界面。

qembed：转换数据，比如，将图片转换为 C++代码。[19]

2.2.3 信号和槽

由于 Qt 仅是本系统客户端图形界面的实现工具，非本文论述重点，这里不过多介绍 Qt 的编程方法。这里仅叙述 Qt 的重要机制之一—信号/槽机制。

使用信号和槽的基本格式为：

connect (sender, SIGNAL (signal), receiver, SLOT (slot));

sender 和 receiver 是指向 QObject 对象的两个指针，signal 为信号函数，slot 为槽函数。

有时当我们一个窗口的内容发生变化时我们希望通知另一个窗口此变化。较老的机制有“回调”机制。回调机制是通过向处理函数传递另一个函数的指针（回调函数），当处理函数运行时会调用回调函数来执行进一步处理。回调函数有几个明显的缺点，首先他们不是类型安全的，我们无法保证处理函数传递了正确的参数形式给回调函数；其次回调函数必须和处理函数紧密的绑定在一起，因为处理函数必须调用哪一个回调函数。

Qt 中使用了新的机制来代替回调函数—信号/槽机制。当一个事件发生时，Qt 发射一个特定信号，与此信号绑定的槽函数将运行。信号和槽并不是一一对应的关心，一个信号可以对应多个槽函数，一个槽函数也可以由多个信号触发，因而它减少了处理函数和响应函数的耦合性。另外，槽函数和信号函数的参数类型必须一致（槽函数的参数数据可以少于信号函数，但是前面多个参数必须一致），这样保证了类型安全。

当然，我们也可以将一个信号与另一个信号连接起来，这样当一个信号发射，另一个信号也会发射。或者一个信号不与任何槽或信号相连，这样，此信号将被忽略。通过此机制，各个模块宽松的连接在了一起，在保证了通讯成功的同时，最大限度的保证了程序灵活性和安全性。[19]

有关 Qt 的更多知识参阅 Qt 手册。

2.3 Libvnc 编程

2.3.1 概述

VNC（Virtual Network Computing）是一个图形化的桌面共享系统，它使用 RFB（远程帧缓冲）协议来远程控制另一台电脑。它传输键盘和鼠标事件给另一台电脑，然后将更新过的桌面图像传递给自身。VNC 是平台无关的，一个 VNC viewer 可以连接到任意一个操作系统上的 VNC 服务器。

LibVNCServer/LibVNCClient 是两个 C 语言编写的库，通过这两个库我们可以很容易实现自己的 VNC 服务器和客户端。

一个简单的使用 LibVNCServer 库编写的服务器程序如下：

```
#include <rfb/rfb.h>
int main(int argc,char** argv)
{
    rfbScreenInfoPtr server=rfbGetScreen(&argc,argv,400,300,8,3,4);
    server->frameBuffer=malloc(400*300*4);
    rfbInitServer(server);
    rfbRunEventLoop(server,-1,FALSE);
    return(0);
}
```

通过上面 4 步我们就能很简单的实现一个 VNC 服务器。为了能正确的显示对方桌面，我们还需要不断将更新过的桌面图像传递回客户端。在上述事例程序基础上，也可以添加更加复杂的功能，如定制鼠标图片，设置桌面颜色等。LibVNC 库仅提供了数据的传输，提供了 RFB 协议的开源实现，对于 Linux 平台下的桌面控制，则需要配合 X11 编程，LibVNC 并不提供控制能力的实现。

2.3.2 RFB 协议

RFB（Remote Frame Buffer, 远程帧缓冲）协议为 VNC 采用的协议，因为它工作在帧缓存级别上，所以它可以应用于所有的窗口系统，如 Windows，Linux 和 Mac。

RFB 采用的是“瘦客户端”模式，绝大部分功能都是在服务器上实现，这样将减小客户端的硬件需求，方便客户端在不同平台下运行。同时，客户端是“无状态”的，即是说客户端从服务端断开之后，另一客户端连接到服务器上将得到上一个客户端相同的状态，用户的状态将在服务端得到保存，而自身并无状态，这样将使用户接口变得更加便捷。[16]

RFB 协议涉及到很多图像的处理和显示，内容较为复杂，本文不赘叙，关于此协议详细内容参见 RFB 协议文档。

2.4 X11 编程

2.4.1 概述

X Window 是 MIT 设计的一个网络透明的窗口系统。X Window 服务器从同一台机器或是网络中的某一台机器上接受用户输出或者分发输入数据。Xlib 是一个 C 语言的子函数库，利用此函数库，用户可以通过数据流连接同窗口系统交互。尽管客户端通常只与自身所在的机器交互，这个需求仍然十分重要。

Xlib—C 语言 X 接口，是一个底层的 C 语言 X 视窗系统协议接口。此函数库假定用户具备图形视窗系统和 C 语言编程的基础知识。其他的高层次的抽象（例如，那些 X 窗口的工具包）均建立在 Xlib 库基础上。X 视窗系统协议（The X Window System Protocol）提供了 X 窗口行为的最终解释。

3 系统架构

3.1 整体架构

Ztalk 服务端基于网络开源项目 jabberd2 开发，采用国际通用的 XMPP 协议。XMPP 协议定义于 RFC3920，数据流以 XML 方式格式化，处理方便，通用性好，此协议由 jabber 工作组创建。Linux 下多个客户端，如 gajim，pidgin, psi 均支持 xmpp。采用 xmpp 协议开发，即使客户端和服务端为两个互相独立的个体开发，仍然能够良好的配合。具有极好的通用性。

Jabberd2 模块结构如下：

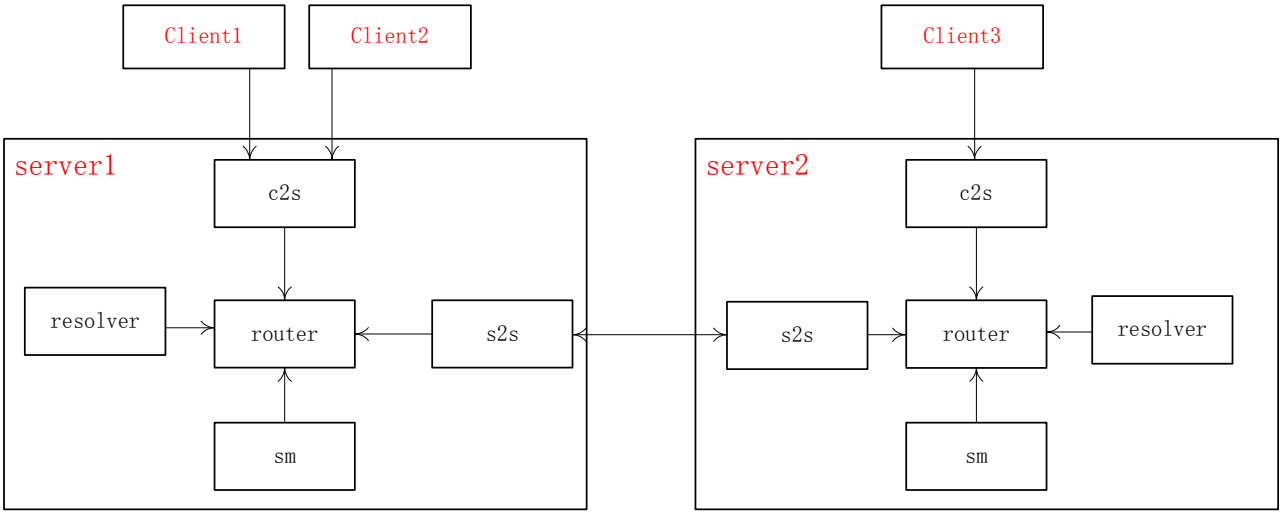

图 3.1 jabberd 结构图

说明：

Jabberd 程序共分五个模块，router, resolver, sm, c2s, s2s。router 的作用为数据包流向判断和中转，resolver 负责解析域名，转化为可识别的 IP 地址， sm 为 session manager 的简称，负责会话管理，c2s 负责接收客户的连接，登录判断，及回显其他用户对其的聊天信息等，直接与客户端接触，s2s 负责服务器与服务器的交互，处理不同服务器上的用户交流。所有的模块均以 router 为中心，connect 到 router 上。

举例说明：

1) Client1 和 Client2 处在同一个服务器上，即域名相同，如 Client1 为 John@example.com, Client2 为 Max@example.com, Cient1 发送信息给 Client2，则信息传递给 c2s, c2s 确认 John 是系统用户，转发数据给 router,router 通过 resolver 模块解析 example.com 为本机，转发给本机的 sm 模块，此后逆向返回给在线用户 client2，此时 client2 即可收到 client1 的聊天信息。

2) Client1 发送信息给另一服务器的用户 Client3，如 Cient3 为 Ben@linuxfoundation.com，则同样，client1 的信息交由 c2s，转移到 router，经 resolver 解析发现在 server2 上面，则 router 将数据包转交给 s2s 模块，s2s 将其转交给 server2 的 s2s 模块，逆向返回到 client3，则 server2 上的 Client3 即可收到 sever1 上的 Client1 的信息。

Ztalk 整体架构图如下:

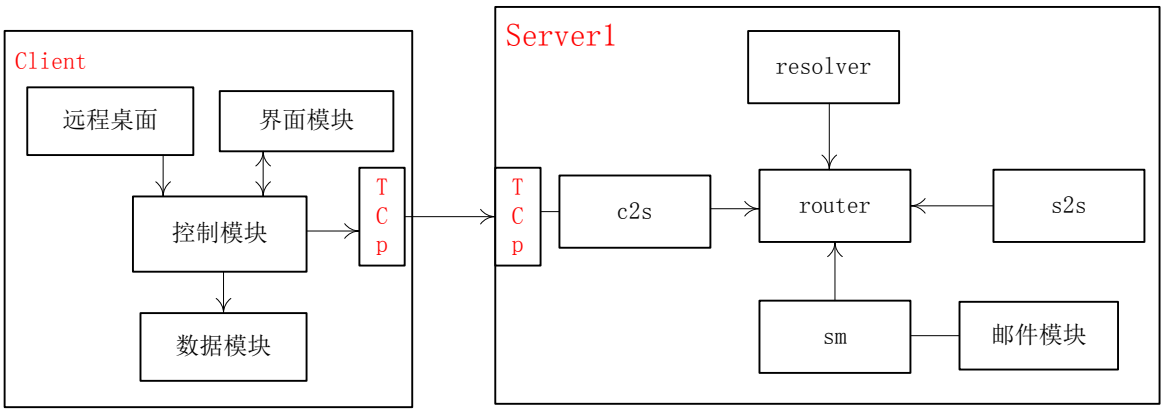

图 3.2 系统整体架构图

如上图，客户端同服务端通过 TCP 连接，客户端中界面部分负责呈现数据，控制部分负责将 TCP 连接中收到的数据传送到数据模块进行处理，再返回控制模块由其决定回复信息，远程桌面功能不属于任何一个模块，故单独划分为一个模块在外；服务端各模块功能见上，其中邮件模块设计为集成在 sm 模块中。

3.2 远程控制的实现

Linux 下的远程控制的实现主要有两个步骤：第一步为将控制机的数据，包括鼠标位置，键盘按键等信息传递给被控制机，由被控制机进行响应；第二部被控制机收到控制机的指令，计算出实际的桌面鼠标键盘信息，由被控制机自身复制控制机的指令。简言之，控制端并不能直接控制被控制机的桌面，而仅仅是发送指令，实质性的控制行为由被控制机自身实现，而这一过程对控制机来说是透明的，所以在控制机看来，被控制机处在直接控制下。

Linux 下的图形界面基于 X11 窗口系统开发，故要实现桌面控制，必然涉及到 X11 视窗系统编程，通过给 X11 窗口发送指令，达到控制鼠标和键盘的目的。

远程控制模块架构图如下：

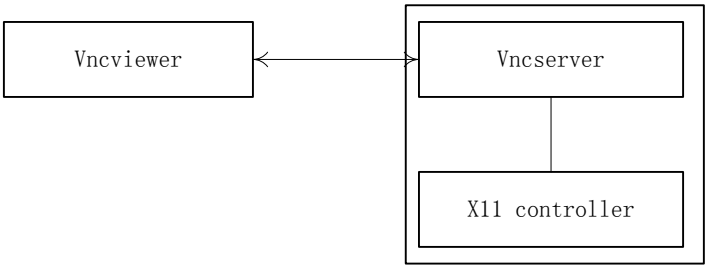

图 3.3 远程控制架构图

如图所示，客户端仅仅是一个简单的 VNC viewer，由于 RFB 协议的设计重点在于服务端，客户端的任务极少，使用 libvncclient 库编写的简易客户端或是系统自带的 vncviewer 均可以满足要求，实现难度较小。

服务端 VNC server 的实现则采用 libvncserver 库编写。Libvncserver 是一个开源的 RFB 协议的实现，在这里调用此函数库接口即可简单的实现 RFB 协议，上图不包括 libvncserver 的内部实现，故 vncserver 作为一个模块出现而不继续详细划分。VNC server 的作用即是从客户端接收指令，完成远程控制的第一步。

X11 controller 是 Linux 图形界面的实质控制部分，由 vncserver 接收到的客户数据，包括鼠标和键盘事件均交由 X11 controller 响应，此模块负责计算鼠标位置，并在相应的鼠标位置或者焦点窗口上复制客户的鼠标和键盘事件。Libvncserver 库并没有控制功能，仅负责客户端和服务端之间的数据传输。

3.3 邮件功能的实现

完整的 jabberd 实现了一整套 IM 工具具有的功能，对于 IM 工具基本的聊天功能，服务端采用 jabberd 完全可以胜任。但是 jabberd 的邮件功能模块需要另外添加。对于此功能的实现，我们提出以下几种方案：

方案一：

如图 3.4 所示，此方案在 jabberd 服务器上另外添加一个邮件网关模块，与 router 相连，当检测到为邮件发送或接受的数据流时，router 将数据包路由到邮件网关模块上，由邮件网关模块将这一数据包翻译并发送到邮件服务器上。这一设计思路符合 jabberd 的实现方式。对于 jabberd 本身代码改动较小，符合“对扩展开放，对实现关闭”的“开放”原则，从这一层面上来讲最优。但是，这一方案对于邮件网关的实现要求较高，工作量较大，短时间内要完整实现风险较大。

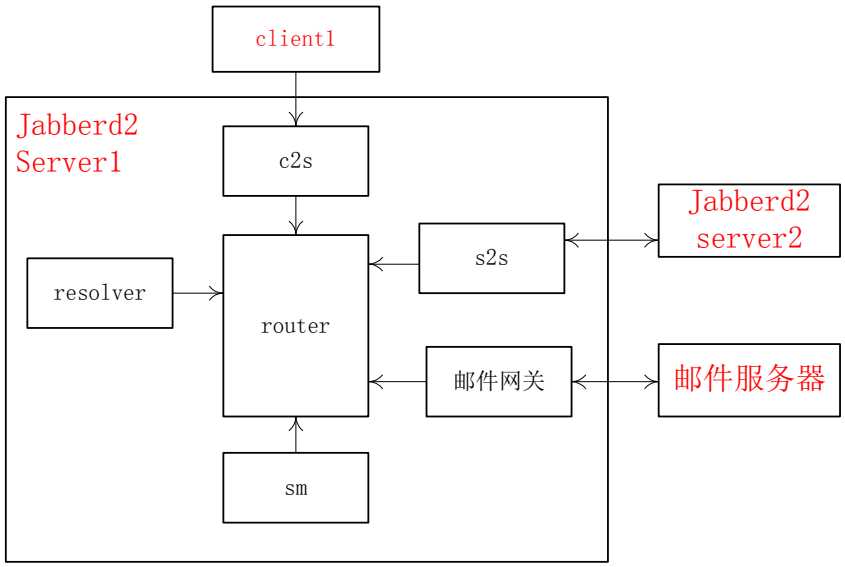

图 3.4 实现方案一

方案二：

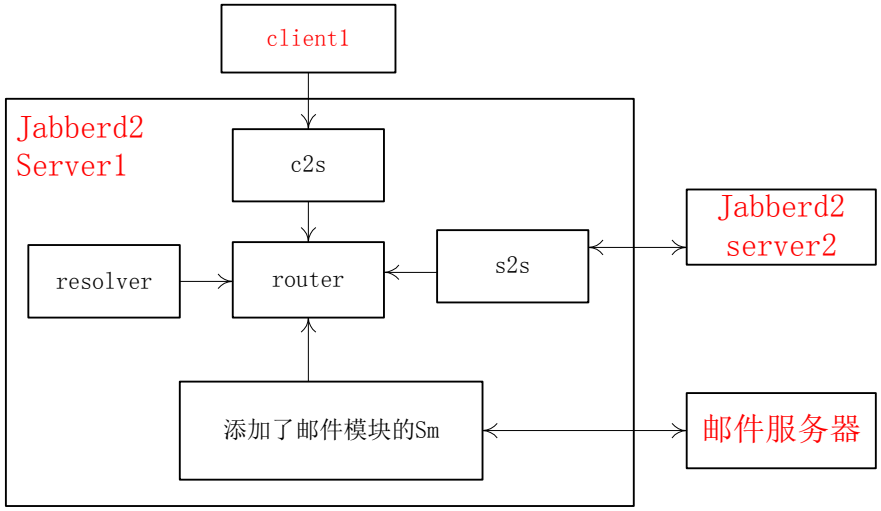

图 3.5 实现方案二

如图 3.5 所示，此方案在 jabberd 的 sm（session manage）部分，添加一个邮件模块，所有的邮件数据包由此模块拦截，并转发到邮件服务器。此方案对 jabberd 本身实现进行了改动，原则上破坏了 jabberd 的完整性，但是，此实现工作量相对要小，实现较容易，是最佳选择。

方案三：

此方案直接将邮件模块集成在客户端上，在实现方面，相比前两种方案，由于不需要对服务端进行大的改动，实现较容易，但是对于 C/S 架构的软件，我们偏向于选择“瘦”客户端模型，即是要尽可能减小客户端的体积，这样，即使需要将客户端移植到处理能力较弱的终端上（如手机上），邮件功能也不受影响，客户端软件也不需要改写，这样将更便

于维护。

总结：综合上述分析，在设计思想上，方案一无疑是最优的，但是方案一的工作量过大，实现风险较高。而考虑到方案三对将来维护可能产生的不便，我们采纳折中的方案--方案二。

注：Jabberd 的优秀之处在于所有的数据处理均以模块和插件的形式实现，具有极高的可扩展性。

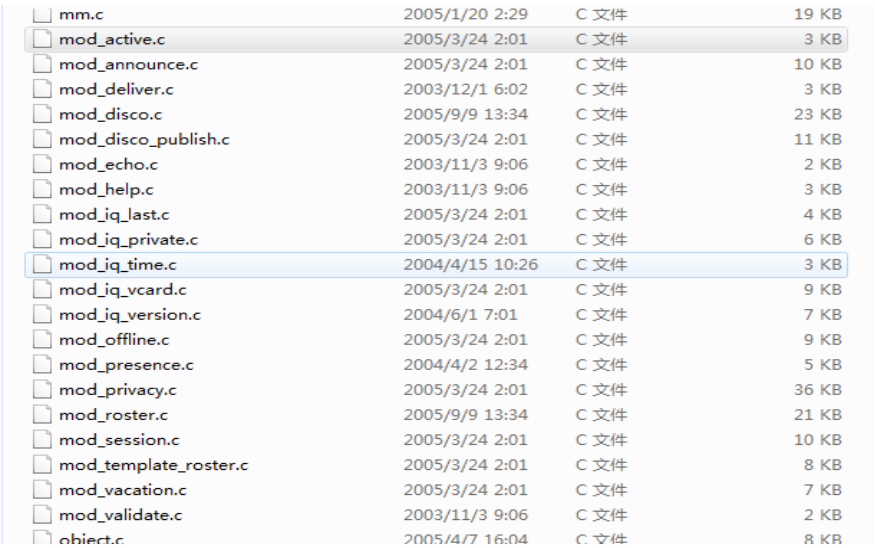

图 3.6 jabberd 模块源码

方案二将在 jabberd 的 SM（会话管理）部分添加一个邮件模块 mod_email.c，检测到邮件数据包时会自动将其交付给 mod_email 模块来处理，包括邮件的收和发。

4 服务端设计

4.1 模块结构

Jabberd 详细模块结构如下：

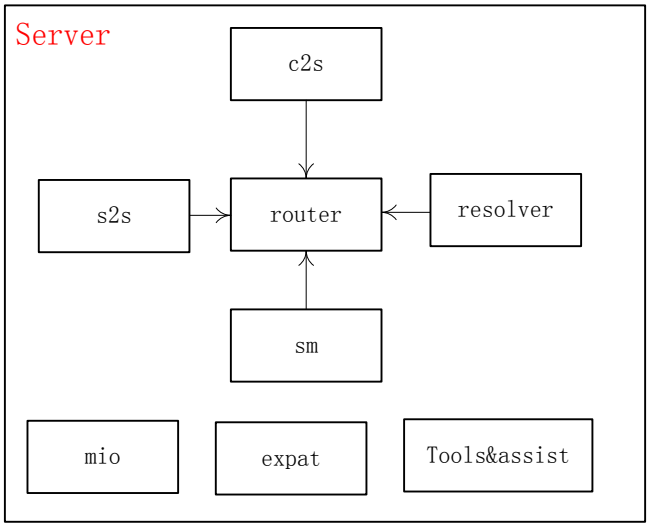

图 4.1 Jabberd 模块图

上图即为详细的 Jabberd 模块图,各模块的作用如下：

i. Router：决定数据包的中转方向，如果客户的数据包需要转移到其他的服务器，则首先通过 resolver 解析域名，后通过 s2s 将数据包转交另一台 jabber 服务器。

ii. Resolver: 负责解析域名，将域名解析为可访问的 IP 地址。

iii. C2s: 负责接收客户端软件到服务器的连接，并验证用户的身份，决定用户是否可以登录到本服务器上。

iv. Sm：负责会话管理。负责处理本机的 iq 查询设置及 message 等信息。

v. S2s：负责 server 到 server 之间的通信。

vi. Mio：此模块负责 TCP 连接的处理，五个核心模块之间的连接都是用此模块提供的机制进行处理。

vii. Expat：此模块来源于另一个开源项目，提供 XML 数据流的处理，C 语言编写，易于使用且功能强大。

viii. Tools&assist: 工具和助手等。此模块比较杂乱，内含 SASL 机制，加密解密机制，内存管理，自定义函数等一系列系统所需而无法明确归类的文件。

各模块对应的文件夹如表 4-1 所示：

表 4.1 模块文件表

所属模块	文件夹	功能
c2s	c2s	实现完整的 c2s 模块功能
resolver	resolver	实现完整的 resolver 模块功能
router	router	实现完整的 router 模块功能
sm	sm	实现完整的 sm 模块功能
s2s	s2s	实现完整的 s2s 模块功能
expat	expat	实现完整的 expat 模块功能
mio	mio	实现完整的 mio 模块功能
tools&assist	scod	提供 SASL 认证机制
tools&assist	subst	提供重定义的网络函数
tools&assist	sx	提供 SASL，SSL，IO 等辅助函数
tools&assist	util	工具和辅助函数

各个模块的运作机制下文详述。

4.2 各模块分析

4.2.1 五核心模块

此处的五核心模块指模块 router，resolver, sm, c2s, s2s，Jabberd 服务启动也是按照上述顺序启动五个核心模块，首先启动 router 模块，监听 5347 端口，后续 4 个模块启动时会尝试连接 router 模块，并在 router 中登记。五个模块通过 router 进行协调中转，通过 TCP 连接“松耦合”在一起。由于五个模块的实现机制极其类似，仅仅作用不同，仅以 router 模块为例说明，其余概要叙述。

以 router 模块的 main 函数为入口：

第一步为填充 struct router_st 结构体，方式为从 router.xml 中提取用户的设置，例如端口，模块名称等，按照 jabberd 预定义配置填充到的结构体各属性值如下：

```
router
struct router_st {
Char            *id = “router”;
Config_t    config = config_new();
Xht       users = {{“jabberd”,” secret”}};
Time_t          users_load = time(NULL);
Log_t   log =log_new( log_SYSLOG, “Jabberd/
router”,” local3”);
Log_type_t        log_type = log_SYSLOG;
Char            *log_facility = local3;
Char            *log_ident = “Jabberd/router”;
Char            *local_ip = “0.0.0.0”;
Int             local_port = 5347;
Char            *local_secret = “secret”;
Char            *local_pemfile =” /usr/local/etc/jabberd/
server.pem”;
Int             io_max_fds = 1024;
Access_t          access = Access_new(0)( “allow,deny”);
Int             conn_rate_total = 0;
int             conn_rate_seconds;
int             conn_rate_wait;
Xht    conn_rates = xhash_new(101);
Int             byte_rate_total = 0;
int             byte_rate_seconds;
int             byte_rate_wait;
Sx_env_t          sx_env = sx_env_new();
sx_plugin_t       sx_ssl;
sx_plugin_t   sx_sasl= sx_env_plugin(r->sx_env,
sx_sasl_init, _router_sx_sasl_callback, (void *) r,
sd_flag_GET_PASS);
Mio_t       mio = mio_new(1024);
Int             fd = 3（socket（））;
Int             check_interval = 60;
Int             check_keepalive = 0;
time_t          next_check;
Prep_cache_t       pc  = prep_cache_new();
Xht      components = xhash_new(101);
= {“ip:port”,component1},{“ip:port”,component2}
Xht      routes = xhash_new(101);
char     *default_route;
Xht      log_sinks = xhash_new(101);
Alias_t          aliases = NULL;
Xht    aci = {{“all” -- “jabberd”)};
Jqueue_t    dead = jqueue_new();
};
```

图 4.2 router_st 结构体

此图为本文作者依据 router.xml 配置所填，配置方式不同，结构体中各属性的值也不尽相同。其他模块均有相似结构体，依次为 struct resolver_st, struct c2s_st, struct sm_st, struct s2s_st，所含的变量相似但略有不同，如 c2s 中有两套端口配置，一份为需要连到到的 router 地址及端口，一份为监听地址及端口，接收来自客户端的连接。

当提取设置完成之后，router 模块进入循环，处理数据：

```
while(!router_shutdown)
{
    mio_run(r->mio, 5);
    if(router_logrotate){
        重新打开日志文件
    }
    /*清理已经“死掉”的 sx_ts */
    while(jqueue_size(r->dead) > 0)
        sx_free((sx_t) jqueue_pull(r->dead));
    /* time checks */
    if(r->check_interval > 0 && time(NULL) >= r->next_check) {
        定时检查
    }
}
```

以上即为循环处理的框架，其中最重要的是第一步的处理函数 mio_run, mio_run 函数隶属于 mio 模块，采用多路复用的方式检查哪一条连接中存在数据需要进行读和写，然后调用回调函数进行数据处理，mio 多路复用机制下文描述。

Router 存在两个重要的回调函数：router_mio_callback 和_router_sx_callback，前者负责处理 TCP 连接，包括接收发起连接，收发数据等；后者根据数据包类型决定系统动作。_router_sx_callback 采用一系列 switch…case…处理各种状况，参数为 sx_event_t，如 event_WANT_READ，event_WANT_WRITE 分别为处理读和写事件，而 event_PACKET 最重要，处理经过授权之后的数据包，负担着会话内容的处理。如确定为 event_PACKET 事件，则首先检测此数据包的类型，根据类型的不同，生成新的回复数据包，回复数据包将发送给其他四个模块之一，根据会话协议而定。

Router 模块之外的其他四个核心模块均采用类似架构，同样含有这两个回调函数（名称略有区别），通过这两个回调函数，可以完成数据包的收发及处理，并能在服务器内部进行正确的通信与协调，由于 Jabber 内部通信规范比较复杂，此处不一一解释。

4.2.2 Expat 模块

Expat 组件本身隶属于另一个网络开源项目，从版权文件容易看出

/* Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd

See the file COPYING for copying permission.*/

Expat 是一个 C 语言编写的 XML 解析库，他是一个面向流的解析器，应用程序可以用这个为可能在 XML 文件中发现的元素（如开始标记）注册处理句柄。

通过 Expat 几个比较重要的函数我们可以看到 Expat 基本的运作原理：

XML_Parser XMLCALL

XML_ParserCreate(const XML_Char *encoding);

构造一个新的解析器。如果 encoding 非空，它为文档提供了一个特定的编码。这将覆盖将要解析的文档的编码声明。有四种内建的编码：

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1

任何其他值都将触发对 UnknownEncodingHandler 函数的调用。

enum XML_Status XMLCALL

XML_Parse(XML_Parser p, const char *s, int len, int isFinal);

enum XML_Status {

XML_STATUS_ERROR = 0,

XML_STATUS_OK = 1

};

解析文档。字符串 s 包含部分（或者是全部）文档。len 指示了 s 中的字节数。这意味中 s 不一定要以终结符结尾。这也意味着如果 len 大于了 s 指向的字符串长度，有可能出现内存错误。isFinal 指示 parser 这是文档需要解析的最后一部分，通常，最后一块长度为 0.如果出现错误，返回 XML_STATUS_ERROR，否则返回 XML_STATUS_OK。

void XMLCALL

XML_SetStartElementHandler(XML_Parser p, XML_StartElementHandler start);

typedef void (XMLCALL *XML_StartElementHandler)(void *userData, const

XML_Char *name, const XML_Char **atts);

为开始（或者空）标记设置句柄（即处理函数）。属性以指向字符数组的指针的格式传递给开始标记处理函数。每个树形在数组中占用两个位置。属性名，后面随机跟随着属性值（如 type='set',则 atts[0]=”type”, atts[1]=”set”，类推）。这种成对的属性值以一个空指针结尾。注意一个空标记产生一个开始句柄和结束句柄的按顺序调用。

void XMLCALL

XML_SetEndElementHandler(XML_Parser p, XML_EndElementHandler);

typedef void (XMLCALL *XML_EndElementHandler)(void *userData, const XML_Char *name);

设置结束标记处理函数（结束句柄），同上，一个空标记产生一个开始句柄的调用和一个结束句柄的调用。

void XMLCALL

XML_SetUserData(XML_Parser p, void *userData);

这个函数设置传递给句柄的用户数据指针。它重写此指针指向的以前数据。注意，当此函数调用完毕之后，释放 userData 指向的空间是合理的。所以当已经有一个指针在哪里的时候，你调用函数，并且之后没有释放它关联的指针，你可能已经导致了内存泄漏。

Expat 的功能非常强大，因而接口较多，较为繁琐。Jabberd 也使用 Expat 解析 XML 数据流，这样，数据较为规整，容易处理。

4.2.3 Mio 模块

Mio 模块负责着整个系统的 TCP 连接管理，包括监听（listen），连接（connect），读（read），写（write）等功能都由此模块封装后的函数完成，上文提到的 mio_run 函数也是由这个模块提供。

此模块比较重要的接口为 mio_new, mio_free, mio_listen, mio_connect, mio_fd, mio_app, mio_close, mio_write, mio_read, mio_run, 关于这些函数的作用，参加代码实现，此处不详述。

其中 mio_run 内部调用了核心模块的两个重要的回调函数（上文已述）。

Mio 模块为了提高 TCP 处理的速度和提高并发度，采用了多路复用的方式，目前采用的方式有 poll（轮询）和 select（选择）两种方式，分别实现于 mio_poll.h, mio_select.h, 这

两个文件分别提供了两种机制的宏实现定义，mio.c 与 mio.h 只需提供框架，然后选择某一种方式，将其宏定义套入实现过程即可。Jabberd 如不自定义安装方式，则默认使用 poll（轮询）方式。

4.2.4 Tools&assist 模块

此处将无法具体分类的函数及文件组为 Tools&assist 模块，此模块内容较杂，本身包括四个文件夹的内容：scod, subst, sx, util，大部分为工具函数，如内存管理，IO 管理以及提供了一些 HASH 函数，加密解密机制等。如 scod 文件夹中提供了 SASL 认证机制的实现，包括 plain（明文），MD5，anonymous（匿名）三个机制；Sx 文件夹则提供了 Sasl 及 ssl 的实现；subst 提供了重写或包装后的网络函数及 linux 系统函数；util 则提供了包括多个 hash 函数实现在内的工具函数。由于此模块杂乱且复杂，此处不一一详解。

5 客户端设计

5.1 模块结构

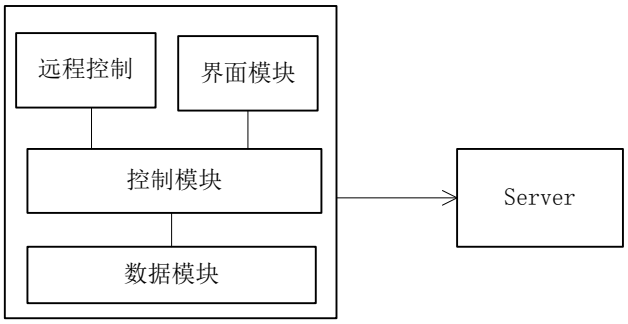

图 5.1 客户端模块图

上图即为客户端的模块结构图，其中界面模块仅仅负责显示数据及处理用户的操作。控制模块为中枢，负责将界面中提取的用户输入转化为对 server 的数据流，并使用 TCP 连接将 server 中得到的数据流交给数据模块。由于数据流为 XML 格式，数据模块需要将数据格式化并提取其中传递的信息，以方便控制模块进行分析并作出反应。

客户端的详细构成如下：

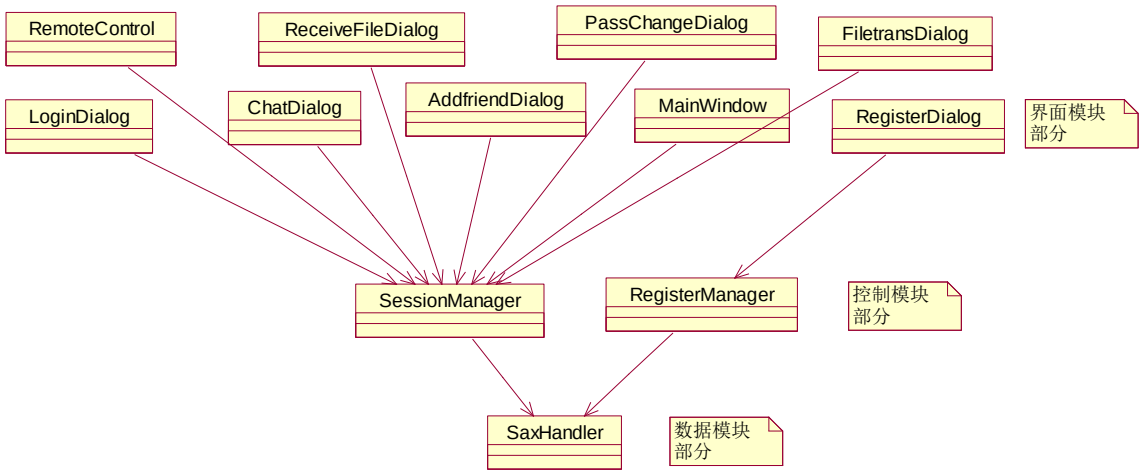

图 5.2 客户端详细构成

其中类 SessionManager 和 RegisterManager 属于控制模块，SaxHandler 属于数据模块，上层 dialog 类属于界面模块。当控制模块从 TCP 连接中获得数据后，将首先交给 SaxHandler 进行格式化，将 XML 流格式化成“树”的形式，处理过的数据格式明朗且去掉了不需要的信息，因为更容易分析。控制模块进一步分析数据模块提交的信息，若不需要用户响应，则直接在后台发送相应的回复信息给服务端，若需要客户进行操作或是界面元素需要进行更改，则提交给上层界面模块。

5.2 各模块分析

5.2.1 控制模块

由于控制模块有着中枢地位，此处详细解释其运作原理。

SessionManager 与 RegisterManager 运作原理相同，为了简化控制模块的复杂度，才将此模块分为两个文件，RegisterManager 仅负责注册的管理，其余均为 SessionManager 负责。其中控制模块处理的第一步交由 SaxHandler 处理，所得到的 XML 树进行进一步分析，根据数据生成当前状态，根据当前状态生成相应的回复信息。

某些回复信息并不需要用户的确认，因而可以在后台直接生成，如登录过程。某些信息需要根据用户的回馈来确定回复信息，如添加好友。回复信息的格式需要严格按照 XML 标准以及 RFC 标准，否则服务端将无法识别而关闭连接。

Jabberd 服务以数据为中心，任何操作均有相应的数据来标识，此处详细介绍各个操作所产生的数据流方便对软件实现的理解，实际实现参见类 SessionManager 中 generateResponse()函数。

1) 登录

假设登录用户为 zw@cookie, 密码为 123123,资源名为 ztalk。S 为 Server 简写，C 为 Client 简写。

图 5.3 登录时序图

i. 客户端请求打开一个流

C: <?xml version='1.0'?>：

<stream:stream to='cookie' xmlns='jabber:client' xmlns:stream= 'http://etherx.jabber.org/streams' version='1.0'>

注：第一行声明 xml 版本为 1.0（详细参见 XML 定义），使用 stream 节初始化一个流，名字空间为'jabber:client'，版本为 1.0，即要求必须支持 TLS 和 SASL。

ii. 服务端确认打开一个数据流：

S: <?xml version='1.0'?>

<stream:stream from='cookie' id='randomId' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>

注：回复信息同样声明 XML 版本 1.0，授权服务器名称为 cookie，且生成了随机会话 id。

iii. SASL 认证过程（见下）

iv. 服务端向客户端声明会话特性：

S: <stream:features>

<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>

<session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>

</stream:features>

注：feature 节声明了服务器支持的流特性，feature 节目前仅用于 TLS，SASL 以及资源绑定的声明。上述即展示了服务器支持资源绑定(bind)和会话(session)打开请求。

v. 客户端申请绑定资源：

C: <iq type="set" id="33">

<bind xmlns="urn:ietf:params:xml:ns:xmpp-bind">

<resource>ztalk</resource>

</bind>

</iq>

注：绑定(bind)的名字空间必须是"urn:ietf:params:xml:ns:xmpp-bind"，声明资源名称为

ztalk

vi. 服务器发回绑定成功信息：

S: <iq xmlns='jabber:client' id='33' type='result'>

<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>

<jid>zw@cookie/ztalk</jid>

</bind>

</iq>

注：type=’result’代表此 IQ 节是一条 set 的结果信息，并通知客户端客户端绑定资源后的全称 jid 为 zw@cookie/ztalk。

vii. 客户端向服务端请求会话：

C: <iq to='cookie' type='set' id=' sess_1'>

<session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>

</iq>

viii. 服务端通知客户端会话建立：

S: <iq from='example.com' type='result' id='sess_1'/>

ix. 客户端请求拉取好友列表

C: <iq xmlns="jabber:client" type="get" id="39">

<query xmlns="jabber:iq:roster" />

</iq>

注：query 代表此条 iq 节是一条查询请求，名字空间"jabber:iq:roster"表明此条查询是名册查询信息，请求服务端发送用户的个人名册,即好友列表。

x. 服务端回复好友列表

S: <iq xmlns='jabber:client' id='39' type='result'>

<query xmlns='jabber:iq:roster'>

<item name='test' subscription='both' jid='test@cookie'>

```
            <group>special</group>
        </item>
        <item name='dude' subscription='both' jid='dude@cookie'>
            <group>friends</group>
        </item>
        <item name='sister' subscription='both' jid='sister@cookie'>
            <group>friends</group>
        </item>
    </query>
</iq>
```

注：IQ 节 type=result 代表这是查询的结果，item 为一条条目，item 的属性中，name 代表好友的昵称；subscription 代表订阅状态，subscription=both 代表了双方互相订阅了对方的在线状态，subscription=from 代表对方订阅了自己的在线状态，subscription=to 代表仅自己订阅了对方的在线状态；jid 指明了好友的 jabber id；group 代表了好友所在的组名，此组名为添加好友时自己为好友制定的组名。

xi. 客户端发送在线信息

C: <presence />

Presence 没有 type 值时默认 type=’presence’，会向所有订阅了自己状态的好友发送广播信息。

xii. 服务端回复好友在线信息

S: <presence xmlns='jabber:client' to='zw@cookie/Gajim' from='dude@cookie/zw-laptop'>

若客户端支持电子名片信息（vCard）,则在线信息类似如下形式：

C: <presence xmlns='jabber:client' to='zw@cookie/Gajim' from='dude@cookie/zw-laptop'>

<x xmlns='jabber:x:delay' from='dude@cookie/zw-laptop' stamp='20110419T12:45:02'/>

<priority>5</priority>

<c xmlns='http://jabber.org/protocol/caps' ext='cs ep-notify html' ver='caps-b75d8d2b25' node='http://psi-im.org/caps'/>

</presence>

注：上面所示的即是好友列表中的 dude@cookie 且以资源名 zw-laptop 登录的好友在线。

关于电子名片的详细使用方法参见 RFC2426.

2) SASL 认证

图 5.4 SASL 认证时序图

i. 提示可用的机制：

S：<stream:features xmlns:stream='http://etherx.jabber.org/streams'>

<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

<mechanism>DIGEST-MD5</mechanism>

<mechanism>PLAIN</mechanism>

</mechanisms>

</stream:features>

第一步提示了可选的机制，包括明文（PLAIN）和 MD5 加密（DIGEST-MD5）两种，为了安全性，建议选择 MD5 加密方法。SASL 认证过程的名字空间一定为

'urn:ietf:params:xml:ns:xmpp-sasl'

ii. 开始授权

C：<auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanism="DIGEST-MD5" />

确定选择 MD5 加密算法

iii. 第一次 challenge

S：<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

cmVhbG09ImNvb2tpZSIsbm9uY2U9ImFmNjMwNTExMWJkNjMyODI2YzU1ZWNhNmU2MTY5ZGFlYWM1MGEzZmMiLHFvcD0iYXV0aCIsY2hhcnNldD11dGYtOCxhbGdvcml0aG09bWQ1LXNlc3M=

</challenge>

以上的字符经过了 base64 编码，解码之后的内容如下：

realm="cookie",nonce="af6305111bd632826c55eca6e6169daeac50a3fc",qop="auth",charset=utf-8,algorithm=md5-sess

第一次 challenge 为 step1，定义如下：

digest-challenge=1#(realm | nonce | qop-options | stale | maxbuf | charset |

algorithm | cipher-opts | auth-param)

realm = "realm" "=" <"> realm-value <">

realm-value = qdstr-val

nonce = "nonce" "=" <"> nonce-value <">

nonce-value = qdstr-val

qop-options = "qop" "=" <"> qop-list <">

qop-list = 1#qop-value

qop-value = "auth" | "auth-int" | "auth-conf" | token

stale = "stale" "=" "true"

maxbuf = "maxbuf" "=" maxbuf-value

maxbuf-value = 1*DIGIT

charset = "charset" "=" "utf-8"

algorithm = "algorithm" "=" "md5-sess"

cipher-opts = "cipher" "=" <"> 1#cipher-value <">

cipher-value= "3des" | "des" | "rc4-40" | "rc4" | "rc4-56" | token

auth-param = token "=" (token | quoted-string)

realm 的值代表服务器名称，nonce 为随机数值，对后续 response 的计算有重要作用，长度需足够。上述值的详细语义参见 RFC2831，此处我们仅做展示，不详述。

iv. 第一次 response

C：<response xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

Y2hhcnNldD11dGYtOCx1c2VybmFtZT0ienciLHJlYWxtPSJjb29raWUiLG5vbmNlPSJhZjYzMDUxMTFiZDYzMjgyNmM1NWVjYTZlNjE2OWRhZWFjNTBhM2ZjIixuYz0wMDAwMDAwMSxjbm9uY2U9IjU0M2RiYTQ5NjZhMzNiYjk1NjEyNGE0YTNjZmQ4MTVlZTNiYTE3N2EwODQ0OTMxMyIsZGlnZXN0LXVyaT0ieG1wcC9jb29raWUiLHJlc3BvbnNlPWJkZmY4ODgzMmQ2OGE4ZTFhNGFkNTljMDg5MWYyNDY5LHFvcD1hdXRo

</response>

第一次 response 为 step2，base64 解码之后的结果如下

charset=utf-8,username="zw",realm="cookie",nonce="af6305111bd632826c55eca6e6169daeac50a3fc",nc=00000001,cnonce="543dba4966a33bb956124a4a3cfd815ee3ba177a08449313",digest-uri="xmpp/cookie",response=bdff88832d68a8e1a4ad59c0891f2469,qop=auth

step2 定义如下：

digest-response = 1#(username | realm | nonce | cnonce | nonce-count | qop | digest-uri

| response | maxbuf | charset | cipher | authzid |auth-param)

Username = "username" "=" <"> username-value <">

username-value = qdstr-val

cnonce = "cnonce" "=" <"> cnonce-value <">

cnonce-value = qdstr-val

nonce-count = "nc" "=" nc-value

nc-value = 8LHEX

qop = "qop" "=" qop-value

digest-uri = "digest-uri" "=" <"> digest-uri-value <">

digest-uri-value = serv-type "/" host ["/" serv-name]

serv-type = 1*ALPHA

host = 1*(ALPHA | DIGIT | "-" | ".")

serv-name = host

response = "response" "=" response-value

response-value = 32LHEX

LHEX = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |"8" | "9"

| "a" | "b" | "c" | "d" | "e" | "f"

Cipher = "cipher" "=" cipher-value

Authzid = "authzid" "=" <"> authzid-value <">

authzid-value = qdstr-val

其中 realm，nonce，qop 的值与 step1 challenge 中提供的值相同，respnose 的值计算过程比较复杂，是 step2 的重点和精华，如若 response 值计算错误，则整个 SASL 认证过程以失败告终。Response 计算公式如下：

response-value = HEX(KD (HEX(H(A1)), { nonce-value, ":" nc-value, ":",cnonce-value, ":", qop-value, ":", HEX(H(A2)) }))

注：HEX 代表将不可读数据转化为可读的 16 进制字符，假设一个字符 X 的 ASCII 码为 31，则 HEX（X）= 1F；H(s)=MD5(s),即是对字符串 S 进行 16 进制 MD5 HASH 计算；KD(k, s) = H({k, ":", s})，例如 KD(“a”, ”b”) = H(“a:b”)。

上述公式中，A1 计算过程如下：

如果 Authzid 声明了确切值，计算过程如下

A1 = { H({ username-value, ":", realm-value, ":", passwd }),

":", nonce-value, ":", cnonce-value, ":", authzid-value }

如果 Authzid 值没有指明，计算过程则如下

A1 = { H({ username-value, ":", realm-value, ":", passwd }),

:", nonce-value, ":", cnonce-value }

A2 计算过程如下：

如果 qop 的值为"auth"，那么 A2 计算过程如下

A2= { "AUTHENTICATE:", digest-uri-value }

如果 qop 的值为"auth-int"或者"auth-conf"，A2 计算过程如下

A2 = { "AUTHENTICATE:", digest-uri-value, ":00000000000000000000000000000000" }

其他值的讲解及 response 详细计算过程参见 RFC2831，此处不详述。

v. 第 n 次 challenge

S：<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

cnNwYXV0aD1kYTZjMmZhNTIxYjI4Nzk0YWE1OTJlMTIxNGJiMTdjZQ==

</challenge>

BASE64 解码之后内容如下：

rspauth=da6c2fa521b28794aa592e1214bb17ce

此步骤为 step3，若 SASL 不需要子认证，则 n=2，即第二次 challenge 且 response 正确之后，SASL 认证已成功。Step3 定义如下：

response-auth = "rspauth" "=" response-value

vi. 第 n 次 response

C：<response xmlns="urn:ietf:params:xml:ns:xmpp-sasl" />

若不需要子认证且 response 值正确，n=2。收到了 rspauth=da6c2fa521b28794aa592e1214bb17ce 信息，客户端即可发送空内容 response 信息进行回复。

vii. 服务器确认认证成功

S：<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>

服务端确认 SASL 认证成功。

viii. 重新打开流

客户端重新发送<stream>节打开一个新的数据流，见[登录]数据流。且服务端亦回复<stream>节确认数据流打开，开始会话过程

3) 注册

用户的注册协议单独记载于 XEP-0077。

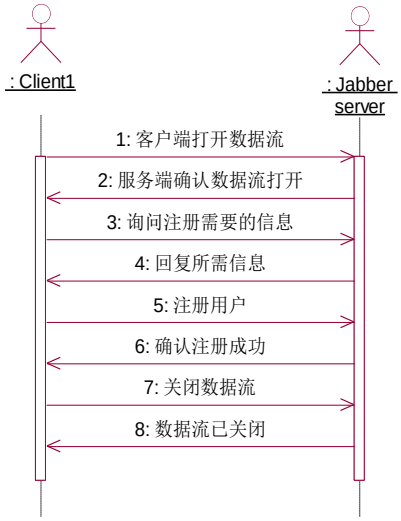

图 5.5 注册用户时序图

i. 客户端打开数据流

<?xml version='1.0'?>

<stream:stream xmlns='jabber:client' to='cookie' version='1.0' xmlns:stream='http://etherx.jabber.org/streams' xml:lang='zh'>

ii. 服务端确认数据流打开

<?xml version='1.0'?>

<stream:stream xmlns:stream='http://etherx.jabber.org/streams' xmlns='jabber:client' from='cookie' version='1.0' id='bvpx84ppvupao7h1y5q9286f41ynhwjtr9q0efqa'>

<stream:features xmlns:stream='http://etherx.jabber.org/streams'>

<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

```
            <mechanism>DIGEST-MD5</mechanism>
            <mechanism>PLAIN</mechanism>
        </mechanisms>
    </stream:features>
```

iii. 询问注册所需要的信息

```
<iq type='get' id='reg1'>
    <query xmlns='jabber:iq:register'/>
</iq>
```

名字空间为'jabber:iq:register'的 query 节询问服务器注册所需要的信息。

iv. 服务端回复所需要的信息

```
<iq xmlns='jabber:client' id='reg1' type='result'>
    <query xmlns='jabber:iq:register'>
        <username/>
        <password/>
        <instructions>
            Enter a username and password to register with this server.
        </instructions>
    </query>
</iq>
```

字段<username/><password/>要求用户输入用户名和密码，<instructions>中的内容向用户进行了说明。

v. 客户端注册用户

```
<iq type='set' id='reg'>
    <query xmlns='jabber:iq:register'>
        <username>test1</username>
```

<password>123123</password>

</query>

</iq>

向服务端注册 ID 为 test1 的用户，密码设为 123123，注册域名即为请求的服务端域名。

vi. 服务端确认注册成功

<iq xmlns='jabber:client' id='reg' type='result'/>

若成功，服务端发送 type=’result’的 iq 节。

若不成功则服务端会回复相应的<error>节，有关 error 节的介绍详见 RFC3290 文档

vii. 关闭数据流

</stream:stream>

viii. 服务器确认数据流关闭

</stream:stream>

4) 更改在线状态

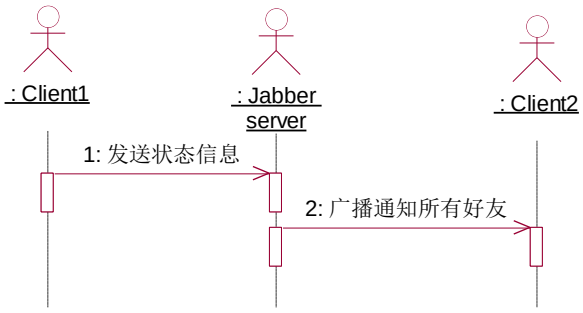

图 5.6 更改在线状态时序图

i. 发送状态信息

<presence type=[state]/>

若离线，state='unavailable'，若在线，state=’presence’,也可以省略 type，则默认为在线信息。

ii. 广播通知所有好友状态信息

5) 好友聊天

假设 test@cookie 欲发送 hello 信息至 zw@cookie，则数据流如下：

<message to='zw@cookie' from='test@cookie/ztalk' type='chat'>

<body>hello</body>

</message>

其中，message 节中当 type=chat 时即为点对点的单人聊天，type 为其他值时参见上文 message 节元素简述。

6) 添加好友(对方同意)

假设用户 zw@cookie(Client1)想要添加 test@cookie(Client2)为好友

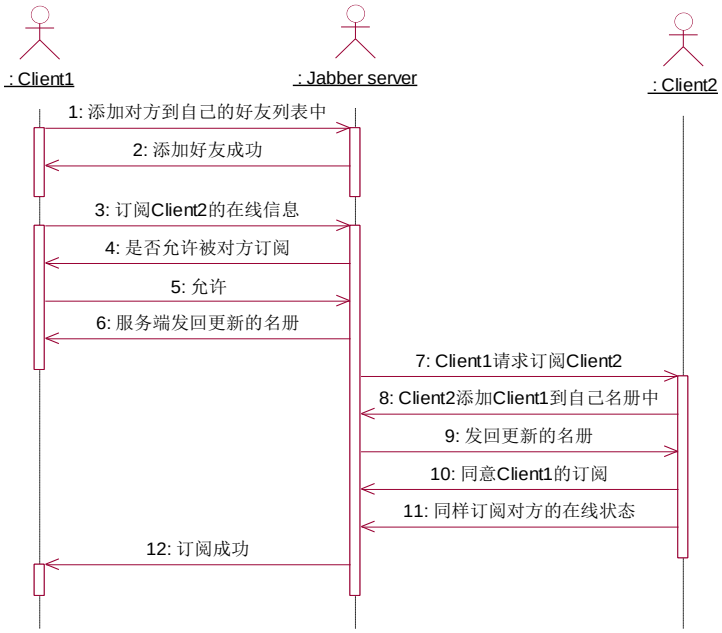

图 5.7 同意添加好友时序图

i. Client1 添加 Client2 到好友列表中

<iq from='zw@cookie/ztalk' type='set' id='roster_4'>

<query xmlns='jabber:iq:roster'>

<item jid='test@cookie' name='test'>

<group>special</group>

</item>

</query>

```
</iq>
```

注：iq 节当 type=set 时，代表设置某些属性值，在 query 节中，名字空间为'jabber:iq:roster'，预示更新名册信息，添加了一个 item，jid 为'test@cookie'，昵称为'test'，即是 Client2 的信息，添加到组 special 中，无 special 组则创建。

ii. 服务端确认添加好友成功

```
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='53e0tt4v' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='test' subscription='none' jid='test@cookie'>
            <group>special</group>
        </item>
    </query>
</iq>
<iq xmlns='jabber:client' id='roster_4' type='result'/>
```

服务端确认节点信息，并返回更新成功状态。注意的是，item 节中含有一个 subscription 属性，代表此好友与自己的订阅状态，此时订阅状态为 none，代表互相之间无订阅。Iq 节 type=result 代表名册更新成功。

iii. Client1 请求订阅对方的在线信息

```
<presence from='zw@cookie/ztalk' to='test@cookie' type='subscribe'/>
```

注：Presence 节当 type='subscribe'，表明想要订阅对方的在线状态。

iv. 系统回复，询问是否允许被订阅

```
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='3xf35n54' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='test' ask='subscribe' subscription='none' jid='test@cookie'>
            <group>special</group>
        </item>
    </query>
```

```
</iq>
```

注：此处需要注意 item 节中存在 ask 属性，ask=’subscribe’，可以理解为“是否允许对方反过来订阅自己的在线状态”，也可以理解为 server 的确认信息，“是否确定订阅对方的在线状态？”。

v. Client1 回复允许被订阅

```
<presence to='test@cookie' type='subscribed'/>
```

注：此条为上条信息的回复，表明同意。此信息可以内嵌入客户端中，不需要客户的确认，直接回复同意信息。

vi. 服务端发回更新的名册

```
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='2hmeddi6' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='test' subscription='from' jid='test@cookie'>
            <group>special</group>
        </item>
    </query>
</iq>
```

注：服务端发回更新的名册信息，此处注意 subscription 值从 none 变为了 from，表明接受对方对自己的订阅。至于自身对对方的订阅仍需要等待对方的回复。

vii. Client2 收到请求

```
<presence from='zw@cookie/ztalk' to='test@cookie' type='subscribe'/>
```

注：Client2 收到来自 zw@cookie 的订阅请求。

viii. Client2 添加对方到自己名册中

```
<iq from='test@cookie/ztalk' type='set' id='roster_4'>
    <query xmlns='jabber:iq:roster'>
        <item jid='zw@cookie' name='zw'>
            <group>friends</group>
```

```
        </item>
    </query>
</iq>
```

注：Client2 同意，则先将对方加入自己的花名册中，这一步亦可由服务器自动完成。建议此处由客户端添加，可以明确昵称及分组信息。

ix. Server 发回更新的名册

```
<iq xmlns='jabber:client' to='test@cookie/ztalk' id='tf52gzmm' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='zw' subscription='none' jid='zw@cookie'>
            <group>friends</group>
        </item>
    </query>
</iq>
<iq xmlns='jabber:client' id='roster_4' type='result'/>
```

注：Server 发回更新的名册，这一步与上面相同。

x. Client2 同意对方的订阅

```
<presence to='zw@cookie' type='subscribed'/>
```

Client2 发送同意信息。Presence 中 type='subscribed'，代表此条为订阅认可信息。

xi. 同样也要订阅对方的在线状态

此过程与 Client1 订阅 Client2 的过程完全相同，不累述。

xii. Client1 收到订阅成功信息

```
<!-- IN -->
<presence xmlns='jabber:client' to='zw@cookie/ztalk' from='test@cookie/ztalk' />
<!-- IN -->
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='nkc1xv7r' type='set'>
```

```
<query xmlns='jabber:iq:roster'>
    <item name='test' subscription='both' jid='test@cookie'>
        <group>special</group>
    </item>
</query>
</iq>
<!-- IN -->
<presence xmlns='jabber:client' from='test@cookie' type='subscribed' to='zw@cookie'/>
```

注：第一条 presence 表明对方目前在线。Iq 节更新了名册，主要为将 subscription 更新为 both 状态，即双方互相订阅了在线状态。第二条 presence 信息中 type='subscribed'，代表了订阅已成功。

7) 添加好友(对方拒绝)

假设用户 zw@cookie(Client1)想要添加 test@cookie(Client2)为好友

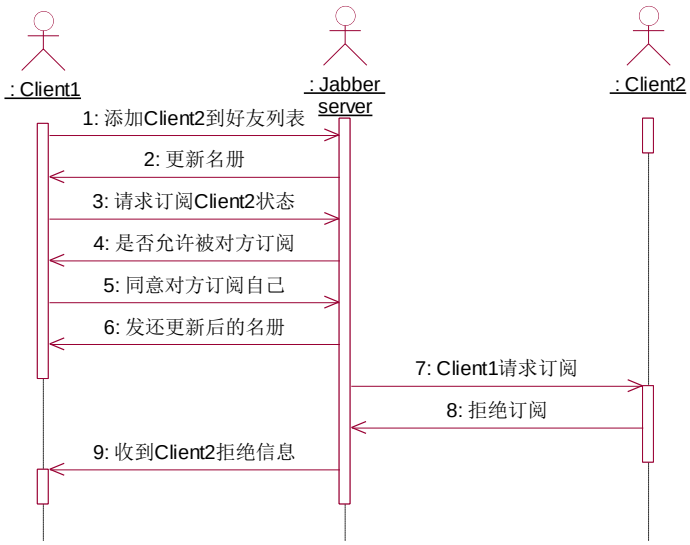

图 5.8 拒绝添加好友时序图

i. Client1 添加好友 Client2 到列表中

```
<iq from='zw@cookie/ztalk' type='set' id='roster_4'>
    <query xmlns='jabber:iq:roster'>
```

```
        <item jid='test@cookie' name='test'>
            <group>special</group>
        </item>
    </query>
</iq>
```

此处 1~6 同 6.7 相同，不详述

ii. Server 发回更新的名册

```
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='53e0tt4v' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='test' subscription='none' jid='test@cookie'>
            <group>special</group>
        </item>
    </query>
</iq>
<iq xmlns='jabber:client' id='roster_4' type='result'/>
```

iii. Client1 订阅好友 Client2 的在线状态

```
<presence from='zw@cookie/ztalk' to='test@cookie' type='subscribe'/>
```

iv. Server 发回确认信息

```
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='3xf35n54' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='test' ask='subscribe' subscription='none' jid='test@cookie'>
            <group>special</group>
        </item>
    </query>
</iq>
```

v. Client1 确认允许对方的订阅

```
<presence to='test@cookie' type='subscribed'/>
```

vi. Server 发还 Client1 更新后名册

```
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='2hmeddi6' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='test' subscription='from' jid='test@cookie'>
            <group>special</group>
        </item>
    </query>
</iq>
```

注：更新 Client2 的订阅状态为 from，即接受对方对自己状态的订阅

vii. Client2 收到订阅请求

```
<presence from='zw@cookie/ztalk' to='test@cookie' type='subscribe'/>
```

注：收到来自 zw@cookie 的订阅请求。

viii. Client2 拒绝对方的订阅

```
<presence to='zw@cookie' type='unsubscribed'/>
```

注：type='unsubscribed'表明拒绝对方的订阅

ix. Client1 收到拒绝信息

```
<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='xa9t5h19' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='test' subscription='from' jid='test@cookie'>
            <group>friends</group>
        </item>
    </query>
</iq>
```

<presence xmlns='jabber:client' from='test@cookie' type='unsubscribed' to='zw@cookie'/>

注：iq 节中 subscription='from'不变，表明只允许对方订阅自己，无法订阅对方状态。Presence 节当中 type='unsubscribed'，明确说明对方拒绝了自己的订阅。

8) 删除好友

假设 zw@cookie(Client1)欲从好友列表中删除 test@cookie(Client2)

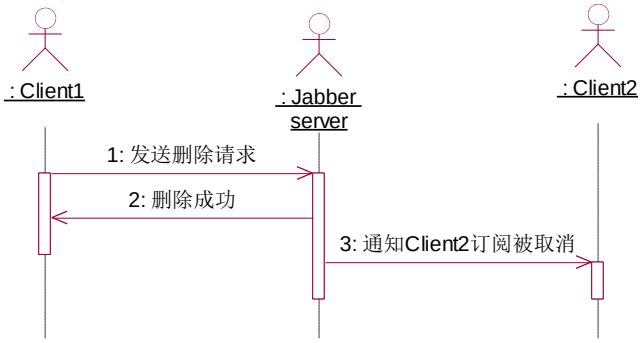

图 5.9 删除好友时序图

i. Client1 发送删除请求

<iq from='zw@cookie/ztalk' type='set' id='roster_4'>

<query xmlns='jabber:iq:roster'>

<item jid='test@cookie' subscription='remove'/>

</query>

</iq>

注：删除请求只需要在 item 节中将 subscription 属性值设为 remove 即表明将此 jid 的节点从自己的名册中删除。

ii. Server 确认删除信息

<iq xmlns='jabber:client' to='zw@cookie/ztalk' id='2asj0781' type='set'>

<query xmlns='jabber:iq:roster'>

<item subscription='remove' jid='test@cookie'/>

</query>

</iq>

<iq xmlns='jabber:client' id='roster_4' type='result'/>

注：服务端返回信息进行确认，并通知 Client1，删除已成功（type=’result’）。

iii. Client2 收到订阅取消信息

```
<iq xmlns='jabber:client' to='test@cookie/ztalk' id='oeh0uwfb' type='set'>
    <query xmlns='jabber:iq:roster'>
        <item name='zw' subscription='from' jid='zw@cookie'>
            <group>friends</group>
        </item>
    </query>
</iq>
```

<presence xmlns='jabber:client' from='zw@cookie' to='test@cookie' type='unsubscribed'/>

注：Client2 收到订阅被取消的信息，第一条 iq 信息中，subscription 从 both 值改变为 from，表明自己已经失去了对对方的订阅。第二条 type=’unsubscribed’，确认 Client1 取消了自己对对方的订阅。

9) 更改密码

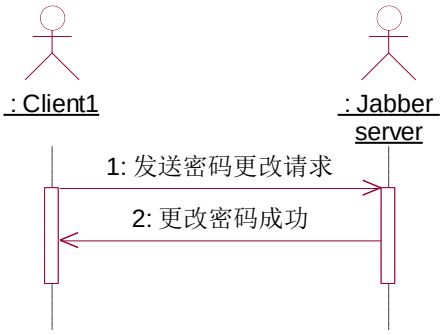

图 5.10 更改密码时序图

i. 发送更改密码请求

```
<iq type='set' to='cookie' id='change1'>
    <query xmlns='jabber:iq:register'>
        <username>test1</username>
```

<password>123456</password>

</query>

</iq>

仔细查看上述更改密码请求，名字空间为'jabber:iq:register'的 query 节，与注册用户的数据流完全相同。因此，更改密码的实质是重新注册了一个新的同名用户，将原先的用户替换掉。

ii. 更改密码成功

<iq xmlns='jabber:client' id='change1' type='result'/>

10) 注销账户

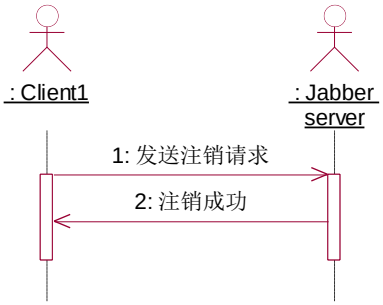

图 5.11 注销成功

i. 发送注销请求

<iq type='set' from='test1@cookie/ztalk' id='unreg1'>

<query xmlns='jabber:iq:register'>

<remove/>

</query>

</iq>

消息空间仍然与注册相同，<remove>标识欲删除账户。

ii. 注销成功

<iq xmlns='jabber:client' id='unreg1' type='result'/>

</stream:stream>

收到注销成功的通知后接连收到关闭数据流的通知。

11) 发送文件（同意）

假设此处文件发送方为 test@cookie(Client1)，文件接收方为 zw@cookie(Client2)。这里为了方便说明，忽视 server 的中转过程，所有请求与回复信息均视为直接发送给对方。有关文件发送的详细叙述，参见 XEP-0096。

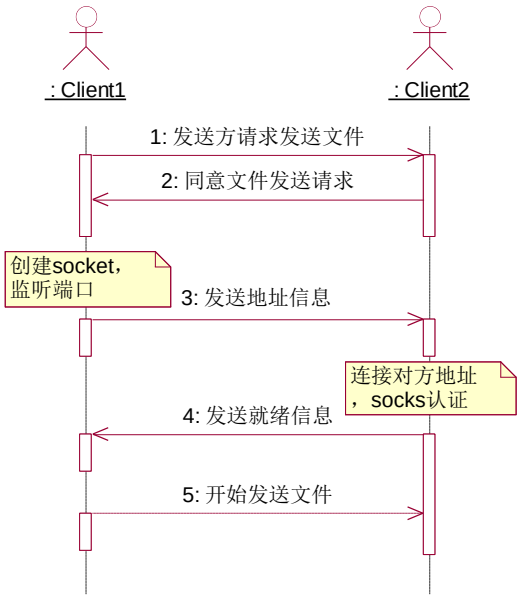

图 5.12 同意发送文件时序图

i. 发送方 Client1 发送流协商字段，请求发送文件

```
<iq type="set" to="zw@cookie/ztalk" id="aac3a">
    <si xmlns="http://jabber.org/protocol/si"
profile="http://jabber.org/protocol/si/profile/file-transfer" id="s55_280e018f10ed34a1">
        <file xmlns="http://jabber.org/protocol/si/profile/file-transfer" size="4295344"
    name="vmlinuz">
            <desc></desc>
            <range/>
        </file>
        <feature xmlns="http://jabber.org/protocol/feature-neg">
            <x xmlns="jabber:x:data" type="form">
                <field type="list-single" var="stream-method">
```

```
                    <option>
                        <value> http://jabber.org/protocol/bytestreams </value>
                    </option>
                </field>
            </x>
        </feature>
    </si>
</iq>
```

这里需要注意 file 字段，name 属性指示了文件名称，size 指明了文件大小，desc 字段可以添加文件描述，range 字段表明发送的文件范围，默认为 0~size-1，即文件全部内容传送，其他内容参加 XEP 文档。

ii. 接收方同意接收文件

```
<iq to="test@cookie/ztalk" type="result" id="s55_280e018f10ed34a1">
    <si xmlns="http://jabber.org/protocol/si">
        <feature xmlns="http://jabber.org/protocol/feature-neg">
            <x xmlns="jabber:x:data" type="submit">
                <field var="stream-method">
                    <value>http://jabber.org/protocol/bytestreams </value>
                </field>
            </x>
        </feature>
    </si>
</iq>
```

iii. 发送方 Client1 发送监听地址（IP 及端口信息）

```
<iq xmlns='jabber:client' from='test@cookie/ztalk' id='aac4a' to='zw@cookie/ztalk'
```

type='set'>

<query xmlns='http://jabber.org/protocol/bytestreams' sid='s55_280e018f10ed34a1' mode='tcp'>

<streamhost jid='test@cookie/ztalk' host='192.168.0.101' port='53264'/>

<fast xmlns='http://affinix.com/jabber/stream'/>

</query>

</iq>

文件发送方 Client1 首先创建一个 socket，监听某个端口，然后将自身 IP 及端口 port 信息都发送给对方，准备接受对方的连接。

iv. SOCKS5 带外数据验证

Jabberd 的文件传输是通过新建一个 socket 连接来单独完成的，又成带外传送。在传送文件之前，必须经过 SOCKS 5 带外数据验证过程。如下：

[接收端 Client2]发送 050100

[发送端 Client1]接收到 050100,发送 0500

[接收端 Client2]接收到 0500,发送 05010003+digest+0000

注：digest = SHA1 Hash of: (SessionID+发送者 JID+接收者 JID)

[发送端 Client1]接收到 05010003+digest+0000,发送 05000003+digest+0000

经过这一步骤，SOCKS 协商完毕，文件可以开始传送。有关 SOCKS5 带外数据验证的详细说明，参见 XEP-0065。

v. 接收方发送“就绪”信息

<!-- IN -->

<iq xmlns='jabber:client' to='test@cookie/ztalk' id='aac4a' type='result' from='zw@cookie/ztalk'>

<query xmlns='http://jabber.org/protocol/bytestreams'>

<streamhost-used jid='test@cookie/ztalk'/>

</query>

</iq>

streamhost-used 字段表明接收端使用了发送方设置的 IP 和端口，并且 connect 成功。

vi. 发送方发送文件

12) 发送文件（拒绝）

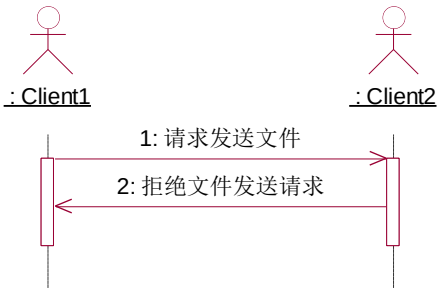

图 5.13 拒绝发送文件时序图

i. 发送方 Client1 发送流协商（SI）请求

<iq type="set" to="zw@cookie/ztalk" id="aac3a">

<si xmlns="http://jabber.org/protocol/si" profile="http://jabber.org/protocol/si/profile/file-transfer" id="s55_052f375ee30f8981">

<file xmlns="http://jabber.org/protocol/si/profile/file-transfer" size="4295344" name="vmlinuz">

<desc></desc>

<range/>

</file>

<feature xmlns="http://jabber.org/protocol/feature-neg">

<x xmlns="jabber:x:data" type="form">

<field type="list-single" var="stream-method">

<option>

<value>http://jabber.org/protocol/bytestreams </value>

</option>

</field>

```
            </x>
        </feature>
    </si>
</iq>
```

此过程同上。

ii. 接收方拒绝文件发送请求

```
<!-- OUT -->
<iq to="test@cookie/ztalk" type="error" id="s55_052f375ee30f8981">
    <error code="403" type="cancel">
        <forbidden />
        <text xmlns="urn:ietf:params:xml:ns:xmpp-stanzas">Offer Declined</text>
    </error>
</iq>
```

拒绝文件请求实质上是一个错误信息，信息包含<forbidden />字段，即表明请求被拒绝。

13) 远程控制

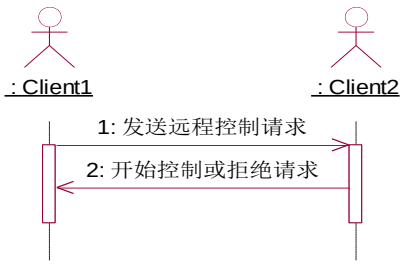

图 5.14 远程控制时序图

i. Client1 请求 Client2 远程协助：

```
<message to='zw@cookie' from='test@cookie/ztalk'>
    <body>__remote_control_request@192.168.0.101 </body>
</message>
```

由于远程协助超过了 Jabberd 的范畴，无通用数据流格式定义。为了简便，我们定义远程协助请求的数据流格式为在聊天信息中，发送__remote_control_request@ + 自身 IP 地址。之后远程控制服务端打开，此处我们使用了 libvncserver 库，协议为 RFB（远程帧缓冲），监听 5900 端口

ii. 接收方回复

若接受方同意控制对方电脑，则无需发送回复信息，直接 connect 对方地址 5900 端口。

若接收方拒绝，则发送如下信息：

<message to='test@cookie' from='zw@cookie/ztalk'>

<body>__remote_control_reject</body>

</message>

此处同样为自定义数据流，格式为在 message 信息中发送__remote_control_reject 字段。

5.2.2 界面模块

界面类实现主要依靠 Qt 图形开发库完成，功能为将数据以图形化的形式展示出来，并将用户操作反馈给下层控制模块。以 MainWindow 为例，通过接口的说明对界面模块的实现进行笼统的说明。

类 MainWindow 主要函数实现：

public:

explicit MainWindow(QWidget *parent = 0); 构造函数，初始化数据

signals: //信号函数，通知其他窗口某事件发生

void send(QString); 通过 TCP 连接向服务器发送数据，此函数进行了实质性的“发送工作”

public slots: //槽函数，处理某一种信号

void updateRosters()：更新好友列表

void updatePresence()：更新所有好友的在线状态，根据不同的在线状态，设置好友图片。

void comboBoxChanged(int index)：当状态栏被改动时，调用此函数。

void chatActionClicked()：聊天 action 被点击

void fileTransActionClicked()：文件传送 action 被点击

void addFriendActionClicked()：添加好友 action 被点击

void removeFriendActionClicked()：删除好友 action 被点击

void destroyActionClicked()：注销账户 action 被点击

void passChangeActionClicked()：密码变更 action 被点击

void aboutActionClicked()：关于 action 被点击

void remoteControlActionClicked()：远程控制 action 被点击

void readyReceiveFile(QString,QString,int, QString)：准备好接收文件

void destroyRosterSuccess()：销毁账户成功

void chat(QTreeWidgetItem*)：聊天

void receiveMessage(QString jid, QString message)：收到消息

void subscribeFrom(QStringList jids)：处理来自用户的请求

void subscribeAsk(QStringList jids)：处理来自系统的 ask 请求

界面模块中其他 dialog 类实现过程均类似，仅涉及到 Qt 图形开发，不一一说明，详细可参照 Qt 开发手册。

5.2.3 数据模块

数据模块处理来自服务端 TCP 连接中取得的数据，将其格式化成“XML 树”的形式，方便后续处理，此处仍然通过接口实现进行说明。

类 SaxHandler 主要函数实现：

public:

bool startElement(const QString &namespaceURI,const QString &localName,

const QString &qName, const QXmlAttributes &attributes)

bool endElement(const QString &namespaceURI,const QString &localName,

const QString &qName)

bool characters(const QString &str)

startElement 在元素开始时被调用，endElement 在元素结束时被调用,characters 处理元素之间的内容。举例说明，<body>hello</body>，当 XML 解析器遇到<body>元素时，调用 startElement 函数，继续处理到达 hello，调用 characters 函数将 hello 存储到合适位置，</body>标识着元素的结束，调用 endElement 函数。

bool fatalError(const QXmlParseException &exception)：XML 中产生的解析错误

QTreeWidget *getXmlTree()

返回解析生成的 XML 树，XML 树以 QTreeWidget*的格式保存。举例说明：

假设 XML 流为

<doc>

<item type=set>item1</item>

<element>ele1</element>

</doc>

生成的 XML 树则形如：

-doc	
Item	item1
element	ele1

图 5.15 XML 树形式

至于 item 的属性 type=set 没有明文显示出来，但是实际上存储在 item 的 data 属性中，当控制模块需要属性值进行消息分析的时候，可简单的从 QTreeWidgetItem 的 data 属性中取出。

6 远程控制

6.1 模块结构

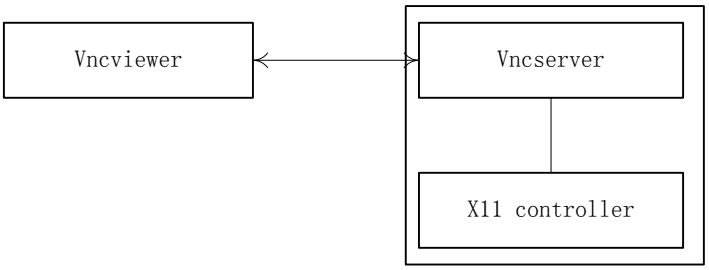

图 6.1 模块结构

远程控制通过 Qt 的截屏功能，定时发送屏幕图像给对方，以达到查看的目的。对于远程控制，则通过客户端发送鼠标，键盘消息给服务端，服务端再通过 X11 库编程达到控制 Linux 桌面动作的功能。这两者之间的数据传送通过 RFB（远程帧缓冲协议）传送，对于 RFB 协议的实现，则是使用了 Linux 的开源库 Libvncserver 来实现。

上图展示了远程控制的模块结构图，其中客户端使用了系统自带的 vncviewer，并没有使用代码实现，当然使用 libvncclient 库能够很方便的实现一个自定义客户端。此远程控制的实现强调了服务端的实现。下文分别介绍两个模块。

6.2 各模块分析

6.2.1 VNC Server 模块

VNCServer 的工作流程如下：

1) 获取 Server 端的屏幕大小

通过 XLib 库函数 DefaultRootWindow 取得 X11 的 root 窗口 winRoot，后从 winRoot 的属性值中得到窗口的总大小 width 和 height，这两个值将在后续获取 rfbScreen 的过程中使用。

2) 填充结构体 rfbScreenInfoPtr 的对象 rfbScreen

这里要填充的属性值包括：

```
rfbScreen->desktopName = "LibVNCServer Example";
rfbScreen->frameBuffer = (char*)malloc(maxx*maxy*bpp);
rfbScreen->alwaysShared = TRUE;
rfbScreen->ptrAddEvent = doptr;
rfbScreen->kbdAddEvent = dokey;
rfbScreen->newClientHook = newclient;
initBuffer((unsigned char*)rfbScreen->frameBuffer);
MakeRichCursor(rfbScreen);
```

desktopName 为窗口名称；frameBuffer 为帧缓冲区，分配的大小根据桌面大小而定，其中，maxx 和 maxy 分别为窗口宽度和高度的 3/4；ptrAddEvent 为鼠标事件句柄，使用 doptr 处理鼠标事件；kbdAddEvent 为键盘事件句柄，采用 dokey 处理键盘事件；newClientHook 为客户端钩子函数，采用 newclient 处理新到的客户端连接。

InitBuffer 初始化 rfbScreen 的缓冲区，主要方式为取得当前屏幕画面（可直接使用 Qt 的截屏函数），压缩图像，然后将图像逐象素拷贝到 frameBuffer 中，此时传送到客户端的即是当前画面。InitBuffer 函数同时设立了一个新的时钟，每 100ms 更新一次屏幕图像，这样客户端看到的就是比较实时的画面了。

MakeRichCursor 可以自定义鼠标形状和颜色，例如*或#，任何形状均可以制作，开发者可充分发挥自己的想象力。

3) 初始化服务端

使用 Libvncserver 库函数初始化服务端。

```
rfbInitServer(rfbScreen);
```

4) 开始事件循环

```
rfbRunEventLoop(rfbScreen,-1,TRUE);
```

其中第三个参数为 true 意味着事件循环将在后台运行，为非阻塞性的，若为 false 则意味着事件循环是阻塞性的，程序将阻塞在此处不断运行。

6.2.2 X11 Controller 模块

X11 Controller 的主要功能为控制，因为 libvncserver 库并不含控制函数，此处需要独立使用 Xlib 库进行编程。它的实现实质上并无独立的模块，而是紧密的嵌入 vncserver 的

实现过程中，主要方式为在鼠标和键盘事件的处理函数中嵌入 X11 的控制代码。

1) 鼠标控制

第一步定义一个鼠标事件 XEvent 对象 event。

第二步将 libvncserver 的鼠标掩码 buttonMask 转化为 X11 可识别的鼠标按键，规则为左键按下：1；中键按下：2；右键按下：4；滚轮向上：8；滚轮向下：16；松开：0。

第三步填充 event 对象各属性，包括第二步转化出的按键值，以及其他各必需属性。

第四步使用 XSendEvent 函数发射鼠标事件，此时，服务端 X11 server 收到此事件并进行响应。这样，客户端的鼠标移动与点击成功复制到了服务端所在的机器上。

2) 键盘控制

第一步通过 XGetInputFocus 获得当前焦点窗口。

第二步复制一个键盘事件，方式为定义一个 XKeyEvent 类型对象 event，然后设置 event 的属性，包括 x,y 坐标，键盘码，事件类型（KeyPress, KeyRelease）等。

第三步使用 XSendEvent 函数将键盘事件发送出去，即完成了一次远程键盘事件响应。

不足之处在于，此处实现的键盘控制较为简单，没有考虑复杂情况，比如组合键。此不足将在后续解决。

7 测试

7.1 注册

注册一个系统不存在的用户—成功：

图 7.1 注册测试图 1

注册系统已经存在的用户—系统报冲突错误：

图 7.2 注册测试图 2

7.2 登录

使用刚刚注册用户名登录—成功

图 7.3 登录测试图 1

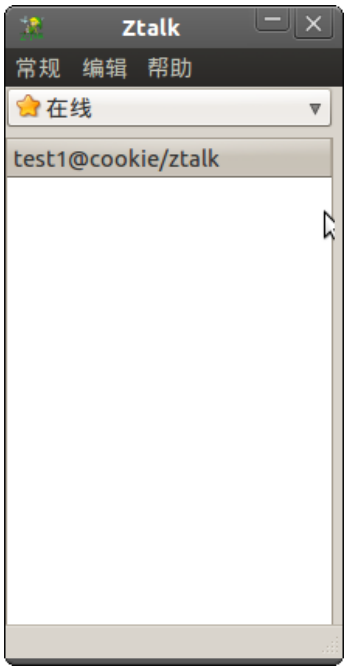

图 7.4 登录测试图 2

7.3 添加好友

添加一个好友

图 7.5 添加好友测试图 1

对方拒绝—添加的好友显示在线状态为未订阅，对方列表内无自己 ID：

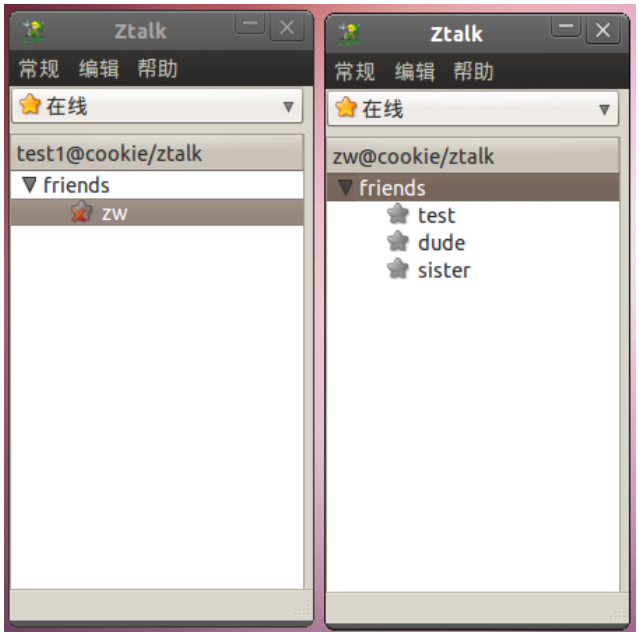

图 7.6 添加好友测试图 2

对方同意—双方建立了互相的订阅，显示状态为在线：

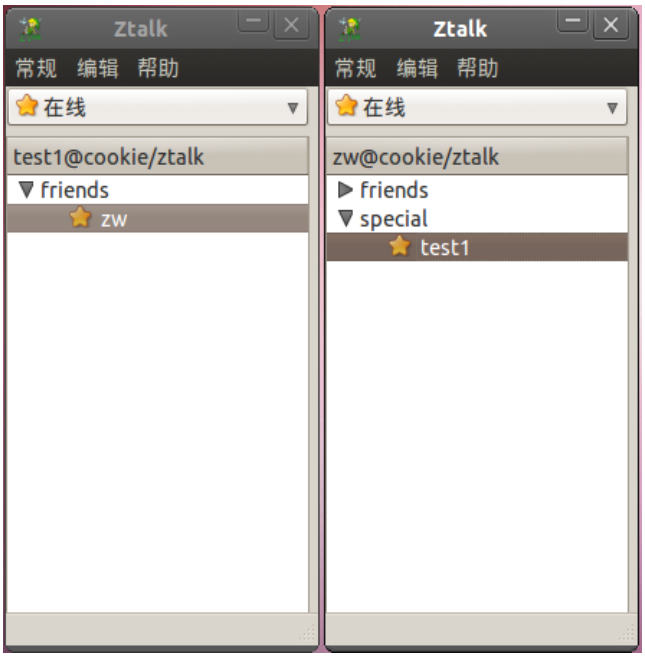

图 7.7 添加好友测试图 3

7.4 删除好友

选中某好友，选择删除：

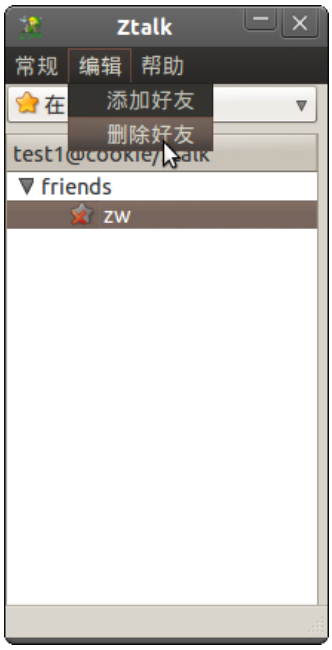

图 7.8 删除好友测试图 1

删除成功—好友从自己列表中移除：

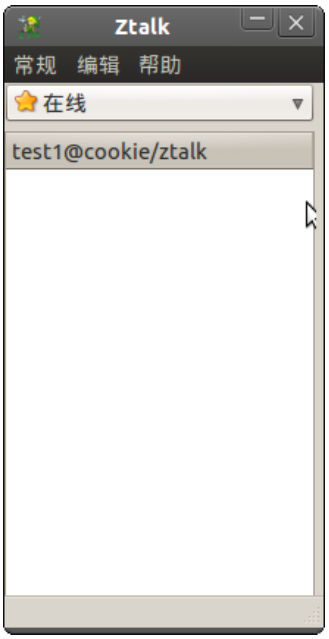

图 7.9 删除好友成功

7.5 好友聊天

选择一个好友，双击或者选择常规—聊天：

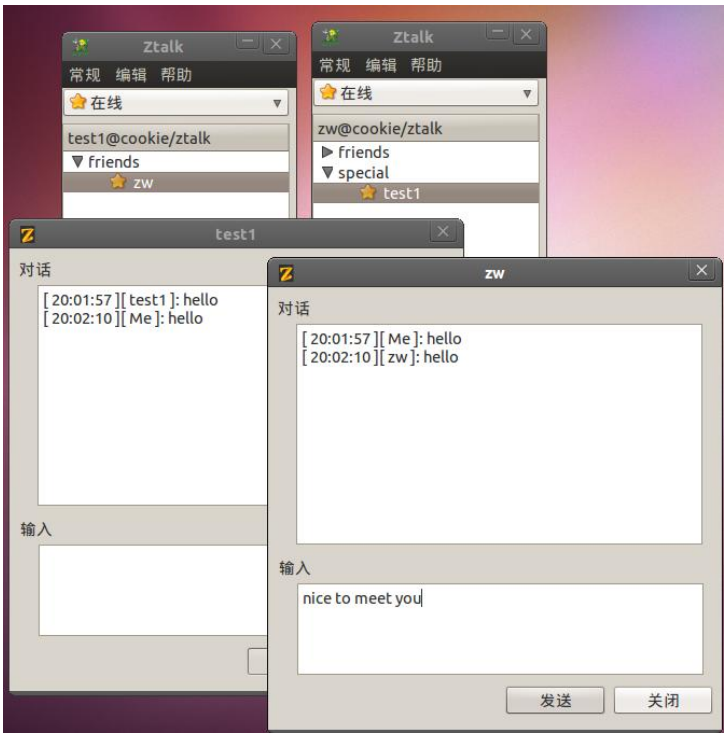

图 7.10 好友聊天

7.6 传送文件

选择好友及要发送的文件，点击发送：

图 7.11 发送文件测试图 1

对方同意接收—文件发送成功：

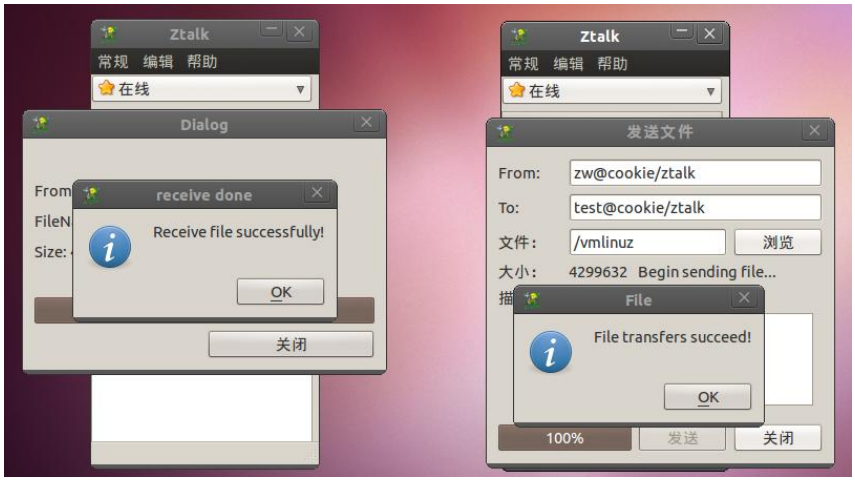

图 7.12 发送文件测试图 2

对方拒绝接收—文件发送过程终止。

7.7 修改密码

选择修改密码—修改成功：

图 7.13 修改密码测试图 1

使用修改过的密码登录—登录成功。

7.8 注销账户

选择注销账户—两次选择确定，注销账户成功：

图 7.14 注销账户测试图 1

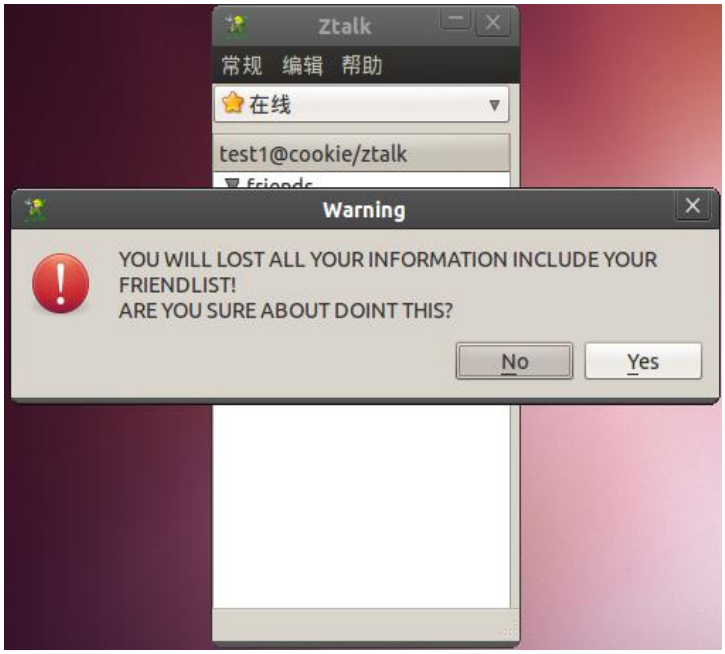

图 7.15 注销账户测试图 2

7.9 远程控制

一用户在本机登录，另一用户在虚拟机中登录，测试使用本机控制虚拟机内的桌面—能够成功控制对方桌面，鼠标键盘操作正常：

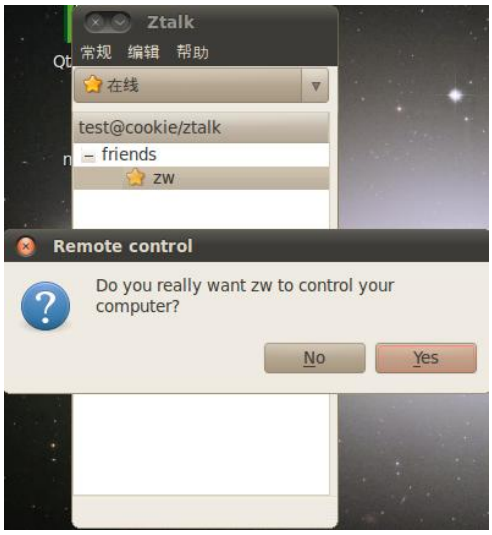

图 7.16 虚拟机内用户发送远程控制请求

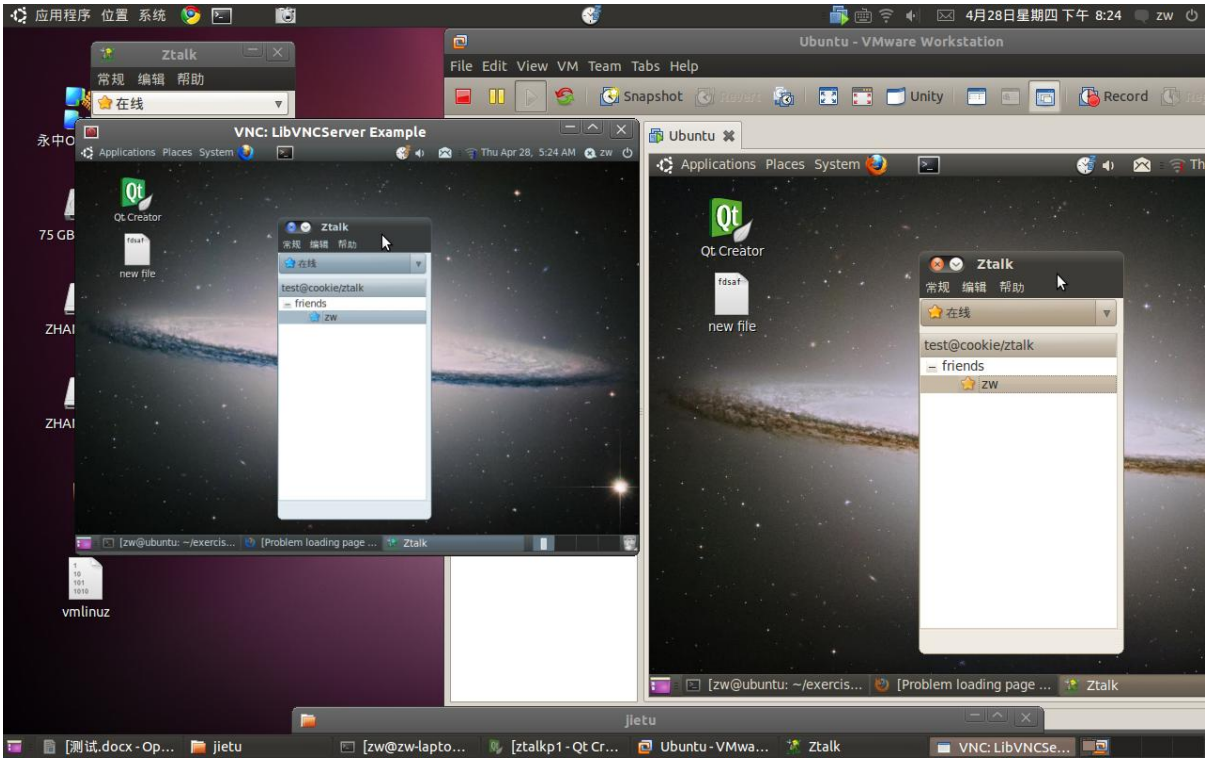

图 7.17 主机接收远程控制请求

8 小结

本文实现了一个简单的即时通讯工具，基于国际上比较流行的 XMPP 协议，采用 Qt 开发，具有良好的通用性及平台兼容性。实现了注册，聊天，添加删除好友，发送文件等功能，并成功实现了特色功能--远程桌面，图形界面比较友好，容易使用。

但是本系统仍然有很多不足之处:

由于群聊功能尚未有正式的 RFC 文档，目前仅有草稿，服务端无正式的实现，所以客户端没有添加群聊功能。

由于系统延迟或是其他原因，有时候几个数据包会同时到达，系统处理不及时有可能出现不正确的操作结果。

远程控制尚存在问题，比如颜色方案错误（这可能同颜色的显示有关），服务端无法及时关闭导致二次控制出现错误等等，此些问题将会在后续解决。

文件发送通用性不佳，原因在于其中的 SOCKS 5 协商过程仍然有未解决问题，导致本客户端文件传送功能无法同其他客户端软件互通。

虽然本系统仍然存在较多问题，但是基本功能均运作良好，上述问题将会在后续开发中一一得到解决。

参考文献

[1] What is Jabber? http://www.jabber.org/about/overview.shtml 2005.01

[2] Thomas, Peter. Jabber Protocol Overview. 2000

[3] Peter Saint-Andre. Jabber Programmer Guide. 2000

[4] P. Saint-Andre. Request for Comments: 3920: Extensible Messaging and Presence Protocol (XMPP): Core. Jabber Softfware Foundation,2004

[5] P. Saint-Andre. Request for Comments: 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Jabber Softfware Foundation,2004

[6] P. Saint-Andre. Request for Comments: 4622: Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP). Jabber Softfware Foundation,2006

[7] R. Rivest. Request for Comments: 1321: The MD5 Message-Digest Algorithm. MIT Laboratory for Computer Science. 1992

[8] M. Leech. Request for Comments: 1928: SOCKS Protocol Version 5. Bell-Northern Research Ltd. 1996

[9] H. Krawczyk. Request for Comments: 2104: HMAC: Keyed-Hashing for Message Authentication. 1997

[10]C. Newman. Request for Comments: 2595: Using TLS with IMAP, POP3 and ACAP. Innosoft.1999

[11]F. Dawson. Request for Comments: 2426: vCard MIME Directory Profile. Lotus Development Corporation.1998

[12]J. Myers. Request for Comments: 2222: Simple Authentication and Security Layer (SASL). Netscape Communications.1997

[13]C. Newman. Request for Comments: 2245: Anonymous SASL Mechanism. Innosoft.1997

[14]C. Newman. Request for Comments: 2444: The One-Time-Password SASL Mechanism Innosoft.1998

[15]P. Leach. Request for Comments: 2831: Using Digest Authentication as a SASL Mechanism Microsoft.2000

[16]Tristan Richardson. The RFB Protocol. RealVNC Ltd. 2010

[17]G.Mohr J.Vincent. Instant Messaging / Presence Protocol Requirements. The Internet Society. 2000

[18]J.Rosenberg H.Sugano. A Model for Presence and Instant Messaging. The Internet Society.2000

[19]Jasmin Blanchette, Mark Summerfield.C++ GUI Programming with Qt 4[M]. Prentice Hall. June 21, 2006.

[20]XMPP 中文翻译计划.http://wiki.jabbercn.org/Category:%E5%B7%B2%E7%BF%BB%E8%AF%91. 2011

[21]Base64 在线解码：http://blbear.com/tools/base64-gb2312.php

[22]Peter Saint-Andre. XEP-0045: 多用户聊天. Jabber Softfware Foundation,2008

[23]Peter Saint-Andre. XEP-0065: SOCKS5 字节流. Jabber Softfware Foundation,2007

[24]Peter Saint-Andre. XEP-0077:带内注册. Jabber Softfware Foundation,2009

[25]Peter Saint-Andre. XEP-0096: 文件传输. Jabber Softfware Foundation,2004

[26]Jabber 对即时讯息的统一构想. http://industry. ccidnet. com/ 2003. 05

[27]W.Richard Stevens 著，范建华 胥光辉等译.TCP/IP 详解卷 1：协议.机械工业出版社,2003

[28]W.Richard Stevens 著，施振川 周利民等译.UNIX 网络编程.清华大学出版社,2001

[29]Jason Kitchen 著，刘建华译.用基于 XML 的即时消息开发 Jabber. 2003

[30]詹舒波，易文强．面向移动通信终端的业务支撑平台们．北京邮电大学学报,2006，29(增)：61-62．

[31]白云锋．无线增值业务平台研究与实现 fDl．天津：天津大学，2007：1--4．

声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得四川大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

本学位论文成果是本人在四川大学读书期间在导师指导下取得的，论文成果归四川大学所有，特此声明。

学位论文作者（签名）

论文指导教师（签名）

2011 年 05 月 日

致 谢

本文是我在四川大学本科学习的一份总结，在此瑾向所有关心、帮助我的老师、同学、朋友和亲人致以最诚挚的感谢。

首先我在这里衷心感谢我的指导老师梁刚的指点和帮助。在毕业设计和论文的撰写过程中，我得到了梁老师的悉心指导和关怀。感谢梁老师在百忙之中审阅我的论文，他的建议使我顺利地完成了本文，他的严谨求实的治学态度和精益求精的工作作风时刻鼓励着我。

感谢我的父母在精神、学业和生活上对我支持和关心，在这里对远在家乡的他们表示深深的思念。

感谢我的室友和同学，是他们的帮助使我比较高效率地完成了论文。

最后，设计的完成，为我在川大的学习生涯画上了一个圆满的句号。在四川大学度过的时光将成为我人生中最美好的记忆。