

本科毕业论文（设计）

基于QT的广域网通讯软件开发

* Development of WAN communication software based on QT

|  |  |
| --- | --- |
| 所 属 学 院 | 数理与电子信息工程学院 |
| 专 业 | 通信工程 |
| 班 级 | 通信194班 |
| 姓 名 | 黄全瑞 |
| 学 号 | 193023020437 |
|  | |
| 指导教师单位 | 广西民族师范学院 |
| 指导教师姓名 | 郭敏杰 |
| 指导教师职称 | 未定级 |
| **完成日期** | **2023年4月10日** |

基于QT的广域网通讯软件开发

**摘要**

为解决Linux桌面操作系统兴起前提下的跨平台网络交流问题，本文设计了一款采用S/C网络架构的跨平台网络通信平台。所设计的通信平台软件在服务器上采用IO复用技术保证了通信平台软件的高并发需求。采用C++ Qt的方式编写UI程序。通过将简单文字通信流程、文件互传等功能集成在所设计的UI界面上，能够轻松解决Windows系统与Linux的互通问题。软件经实测，所设计的系统能够实现预定功能，有利于促进后续互联网和计算机领域的发展。

**关键词：** S/C架构；网络通信；软件开发

* Development of WAN communication software based on QT

**Abstract**

In order to solve the cross-platform network communication problem under the rise of Linux desktop operating system, this paper designs a cross-platform network communication platform using S/C network architecture. The designed communication platform software adopts IO multiplexing technology on the server to ensure the high concurrency requirements of the communication platform software. Write UI programs in C++ Qt. By integrating simple text communication processes, file transfer and other functions on the designed UI interface, it can easily solve the problem of interoperability between Windows system and Linux. After actual testing, the designed system can achieve the intended function, which is conducive to promoting the subsequent development of the Internet and computer fields.

**Key words:** S/C architecture; Network communication;

目录

[1 绪论 1](#_Toc134008019)

[1.1 研究背景及意义 1](#_Toc134008020)

[1.2 研究内容以及难点 1](#_Toc134008021)

[2 研究思路和方案设计 3](#_Toc134008022)

[2.1 项目简介 3](#_Toc134008023)

[2.2 主流实现的基本通信功能 3](#_Toc134008024)

[2.3 程序通信架构方案 4](#_Toc134008025)

[2.4 界面程序架构方案 5](#_Toc134008026)

[2.5 技术需求分析 5](#_Toc134008027)

[3 技术探讨与选择 6](#_Toc134008028)

[3.1 探讨内容概要 6](#_Toc134008029)

[3.2 网络协议 6](#_Toc134008030)

[3.2.1 网络协议简介 6](#_Toc134008031)

[3.2.2 TCP协议 6](#_Toc134008032)

[3.2.3 UPD协议 6](#_Toc134008033)

[3.2.4 HTTP协议 7](#_Toc134008034)

[3.2.5 WebSockect协议 7](#_Toc134008035)

[3.2.6 协议选择 7](#_Toc134008036)

[3.3 高并发技术 8](#_Toc134008037)

[3.3.1 高并发服务器开发简介 8](#_Toc134008038)

[3.3.2 select技术 8](#_Toc134008039)

[3.3.3 poll技术 8](#_Toc134008040)

[3.3.4 epoll技术 9](#_Toc134008041)

[3.4 数据库 9](#_Toc134008042)

[3.4.1 数据库简介 9](#_Toc134008043)

[3.4.2 数据库选择 9](#_Toc134008044)

[3.5 网络开源库 10](#_Toc134008045)

[3.5.1 为什么选择开源库 10](#_Toc134008046)

[3.5.2 网络编程的常见问题 10](#_Toc134008047)

[3.5.3 网络开源库选择 11](#_Toc134008048)

[3.6 图形框架 11](#_Toc134008049)

[3.6.1 图形框架简介 11](#_Toc134008050)

[3.6.2 图形框架选择 12](#_Toc134008051)

[4 跨平台软件设计 13](#_Toc134008052)

[4.1 章节概要 13](#_Toc134008053)

[4.2 客户端架构设计 13](#_Toc134008054)

[4.3 登录流程 14](#_Toc134008055)

[4.4 登录状态切换 15](#_Toc134008056)

[4.5 数据交换流程 16](#_Toc134008057)

[4.6 数据交换状态 17](#_Toc134008058)

[4.7 服务器架构设计 18](#_Toc134008059)

[4.8 注册请求处理 19](#_Toc134008060)

[4.9 登录请求处理 20](#_Toc134008061)

[4.10 转发请求处理 21](#_Toc134008062)

[5 交换协议与数据模型 22](#_Toc134008063)

[5.1 章节概要 22](#_Toc134008064)

[5.2 请求响应模型 22](#_Toc134008065)

[5.3 数据转发模型 23](#_Toc134008066)

[5.4 交互协议结构 24](#_Toc134008067)

[6 实测结果与讨论 25](#_Toc134008068)

[6.1 章节概要 25](#_Toc134008069)

[6.2 登录窗口测试 25](#_Toc134008070)

[6.3 聊天窗口测试 26](#_Toc134008071)

[6.4 文件进度测试 27](#_Toc134008072)

[6.5 好友申请测试 28](#_Toc134008073)

[7 总结和展望 30](#_Toc134008074)

[8 参考文献 31](#_Toc134008075)

[9 致谢 32](#_Toc134008076)

[10 附录 A 完整的转发协议文件 33](#_Toc134008077)

[11 附录 B 服务器任务分发代码 38](#_Toc134008078)

[12 附录 C 客户端的任务转发代码 40](#_Toc134008079)

# 1 绪论

## 研究背景及意义

网络聊天工具的发展依托于现代互联网的流行，其在一定程度上取代了过去的异地通话及短信服务。作为一种基于计算机网络的终端通信工具，网络聊天工具最显著的特点便是便捷性与灵活性。现阶段最具流行性的便是腾讯公司旗下的QQ和微信两款通信软件。当然其中也包括诸如探探、陌陌等成人向交友软件。虽然此类软件在社交方式与主流业务上有很多不同，但其本质仍旧是社交通信软件。

但现阶段主流的网络聊天工具主要服务于Windows操作系统平台，随着Linux桌面操作系统的兴起，已经有一部分用于开始选择使用Linux系统作为日常使用的操作系统。但由于Linux系统主要服务于开发者而非普通用户，导致Linux操作系统在日常用户的软件生态上非常缺失，在Linux操作系统的上可用的聊天软件可谓抓襟见肘。为解决这样的问题，本文拟研究建立一个可跨平台的、既支持以软件的形式启动客户端、还支持以浏览器网页形式启动客户端聊天通信软件，并以达到快捷、即时的通信目的。

基于以上论述可以看出，对于网络聊天工具的研究在当今社会具有一定的重要性，它有效的提高了沟通效率，缩短了交流所需要的时间成本，在个人生活的各方面都起着重要作用。同时，网络聊天工具也为企业提供了一个方便的沟通渠道，有助于公司提高客户服务效率。再者，网络聊天工具也可以为学校提供一种有效的学习沟通方式。因为有了便捷的网络聊天工具，它们极大地改善社会的沟通方式以及沟通效率。

## 研究内容以及难点

本文拟基于Qt图形框架开发广域网通信工具，所设计的软件将在Qt、C++和Linux环境下开发，并实现注册、登录、信息发送、文件传输等功能。

本文的研究难点在于，在高并发、多连接的高性能服务器网络编程上很难突破现有框架限制，因为一个高质量的网络框架需要经过工业级别的多年锤炼才可以流行于市面上。同时，采用WebSocket协议作为底层协议虽然为在未来实现浏览器客户端留下了可能，但应用层协议带来的开销往往是性能的瓶颈，很难突破。再者，在服务器上，数据库交互也是造成性能瓶颈的主要原因，所以对于数据表进行范式设计与优化、对数据的类型结构拆解也是难点之一。

# 2 研究思路和方案设计

## 项目简介

本文拟设计一个聊天软件，该类聊天软件的业务一般为文字聊天、语音/视频聊天、文件传输、多开同时会话、可发送表情及动画、保存聊天记录、创建群组等七大功能。而本项目将重心放置到UI结构设计、软件架构、传输交换协议处理、以及用户交互上来，业务逻辑部分暂不讨论。本文提供了登录与账号注册功能、发送文字与发送表情包功能、传输文件功能、添加好友功能、保存聊天记录功能，且将服务器部署至互联网。

## 主流实现的基本通信功能

该软件依照现在比较流行的聊天软件的主流逻辑与通信功能做出来以下几点的设计，分别是：

1. 提供互联网通信，部署公网服务器，采用S/C网络架构通信。
2. 提供账号注册功能，用于登陆验证用户身份。
3. 提供好友申请等组群关系。
4. 提供基本的文字消息发送功能。
5. 提供基本的表情包图片发送功能。
6. 提供基本的文件传输功能。
7. 提供历史消息记录功能。

依据上述的功能，接下来将对程序的总体框架进行设计，对整体程序的通信框架，以及UI界面的总体布局提出合理的架构设计方案。

## 程序通信架构方案

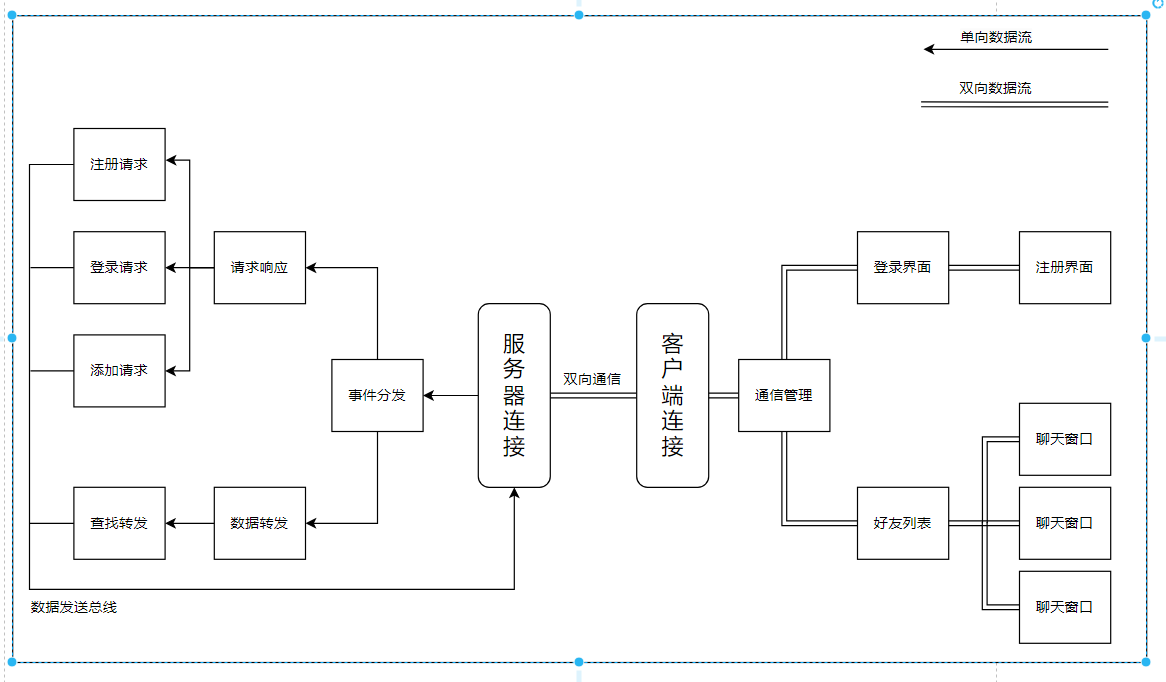


图 2.1 通信架构

如图2.1所示为程序的整体通信架构，程序的通信架构分为两个部分，分别是客户端连接与服务器连接。从图中可以看出，客户端采用双向的数据流，但被层级限制，数据需要层层传递才能到达最深处，这样的设计有利于模块化程序的结构。所有数据必须经过通信管理层，使得客户端功能模块开发不需要关心网络通信，达到模块解耦的功能，有利于业务功能的开发。

服务器的数据流则被设置为单向流数据传输，这样设计的原因是因为任务需要被事件分发器分发到指定的任务处理函数。处理模块有直接与客户端通信的权利，当任务处理函数执行完成之后，任务处理函数直接与客户端通信，对请求任务进行反馈。

## 界面程序架构方案

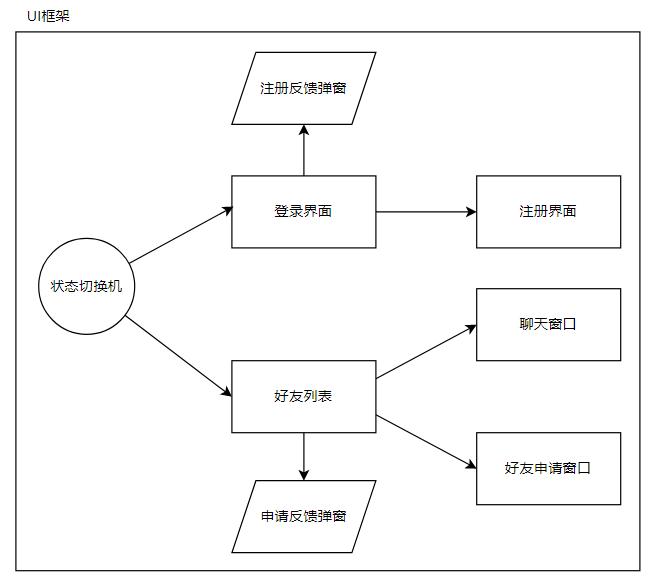


图 2.2 界面架构

如图2.2所示为程序图形界面的总体架构，图形界面架构被分为两个部分，分别是登录前的登录界面和登录之后的好友列表，两个UI界面互不干扰，由状态切换机来控制两者的显示。

登录界面分别控制注册界面与注册反馈弹窗，注册功能的界面由登录界面启动。好友列表则控制聊天窗口、好友申请窗口、申请反馈弹窗，聊天功能与好友申请由好友列表窗口提供。

## 技术需求分析

本文需要实现高性能网络服务器，实现UI图形界面与用于的交互，实现客户端与服务器的数据交互，需要数据库存储用于信息，还涉及到Windows与Linux两个操作系统的跨平台问题。从开发者角度出发，该软件开发触及到网络服务器开发，桌面UI图形界面开发，跨平台开发，数据库操作，应用层业务协议设计，网络服务器部署等技术需求，对技术的深度与广度都有一定的需求。

# 3. 跨平台软件设计

## 章节概要

本章将列出客户端以及服务器的实现细节，对整个程序的实现做出一个整体的展示。其中将详细介绍客户端与服务器的设计框架，以及解析程序运行流程图的实现逻辑与设计思路。

## 客户端架构设计

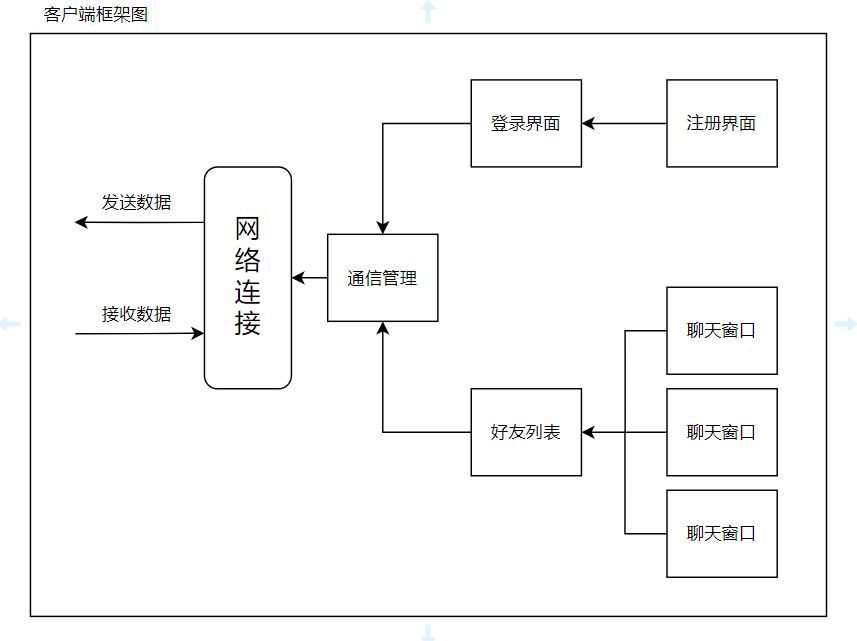


图 4.1 客户端架构

由客户端架构图可以看出，客户端的框架设计中在网络接连处安插了一层通信管理，通信管理负责整个客户端程序与服务器的数据交互，所有从客户端发送或者从服务器接收的消息都用通信管理层统一分配到各个模块。

其中登录模块与好友列表模块是分离的，由通信管理模块进行整合，如登录成功时，登录模块以及完成任务，则由通信管理模块负责将其关闭，再启动好友列表模块。

## 登录流程

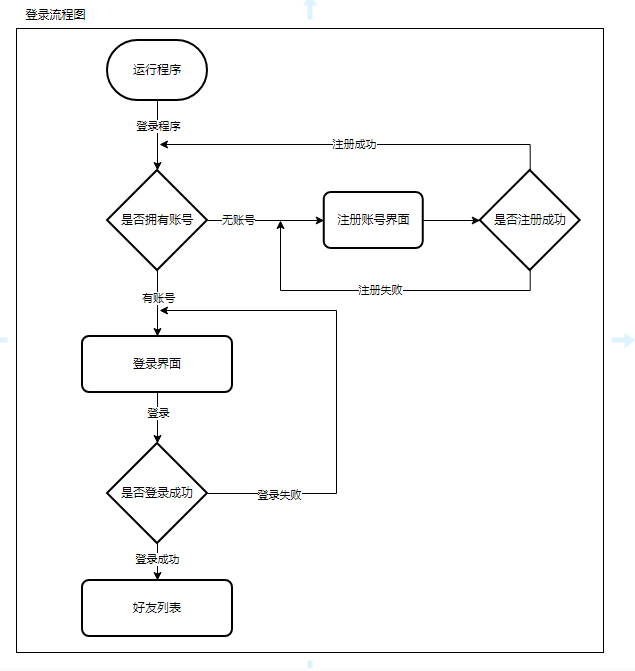


图 4.2 登录流程

从登录流程图可以看出，程序的登录流程。程序的登录大致可以分为两个状态，有账号与无账号状态。当有账号时，可直接输入账号密码登录。无账号时，需申请账号，并登录到好友列表界面。

## 登录状态切换

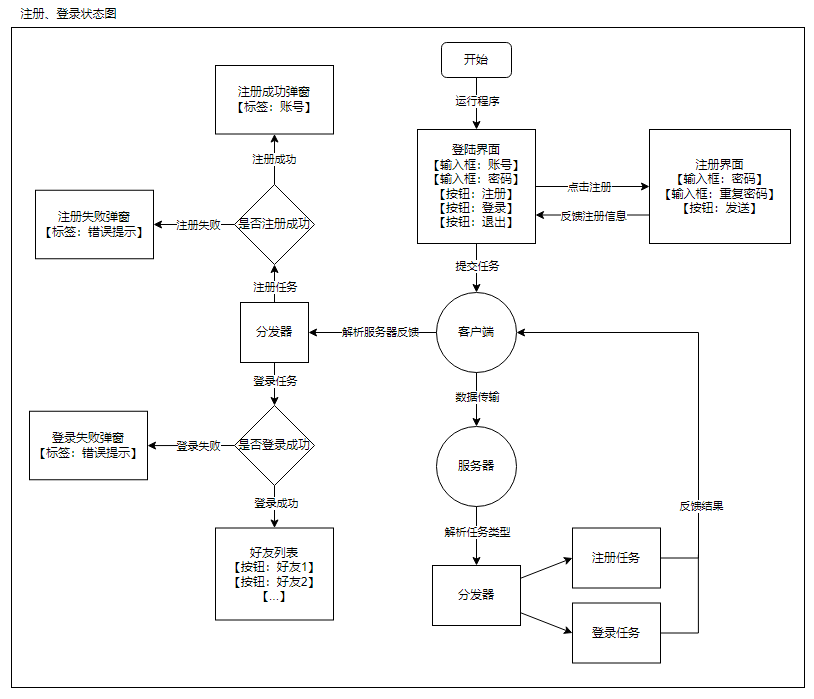


图 4.3 登录状态

由登陆状态图可以看出，客户端登录、注册时与数据与服务器的交互状态，以及UI切换的状态。图中可以清晰的看出，程序运行到成功登录的所有状态。首先程序运行会弹出登录界面，登录界面提供了UI控件，其中包括账号密码输入，以及登录、注册等按钮。

如果用户进入注册界面并注册账号时，注册界面会收集用户的信息，并将注册信息回传到登录界面，由登录界面程序将注册信息发送到服务器。服务器通过任务分发器将任务分发到指定的处理函数，再由处理函数将处理结果反馈到客户端。客户端通过任务反馈分发器做出不同状态的切换。

客户端上的任务反馈分发器将判断注册、登录是否成功，并执行指定任务。如果注册成功则显示注册的账号，失败则反馈错误信息。当用户登录成功时，好友列表UI将显示，注册、登录UI将被关闭。

## 数据交换流程

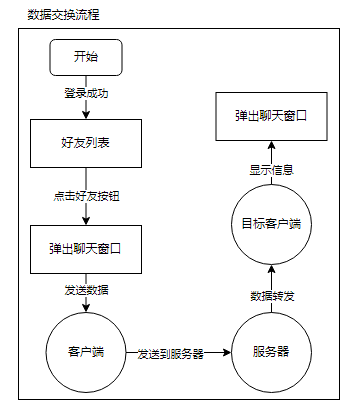


图 4.4 数据交换流程

从图4.4中可以看出，程序在从一方客户端将数据发送到目标客户端时的基本流程。当用户点击好友按钮时，会弹出对应的聊天窗口，此时可以与之通信。所有发送的数据由服务器转发到目标客户端上，发送的数据将在目标客户端中对应的发送方的聊天窗口上显示。

## 数据交换状态

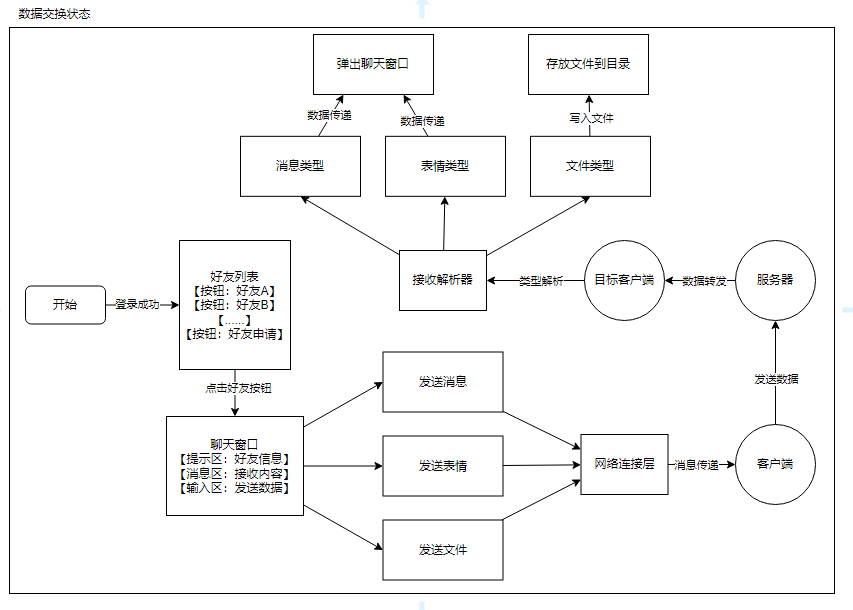


图 4.5 数据交换状态

由图4.5得知，发送数据交换时状态响应，UI界面的状态反应。数据交换主要有聊天窗口提供，聊天窗口由数据发送和接收数据两大部件组成。数据发送主要提供三大功能，分别是消息、表情包、文件。

发送任何一种类型的数据，都首先传递到网络连接层，将消息类型预处理，之后发送到服务器。服务器将数据转发到目标客户端时，由接收解析器分析类型，再调用对应的处理函数。不管收到何种消息类型，目标客户端都会将聊天窗口弹出，以表示接收到对话。

## 服务器架构设计

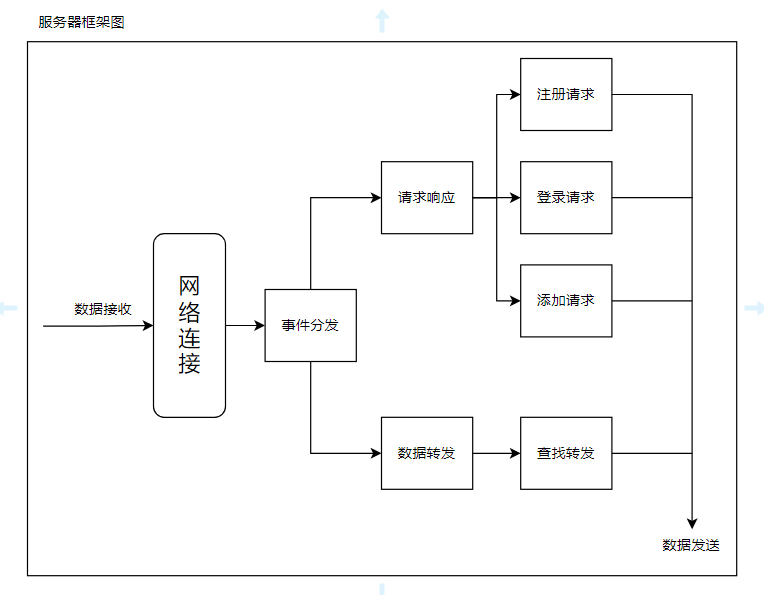


图 4.6 服务器架构

由图4.6可以得知，当客户端向服务器发送消息时，由事件分发器判断类型，如果是数据转发事件则交由数据转发模块进行处理。如果是请求响应事件则再一步确认任务类型，最后提交到具体的任务处理函数中。可以看出服务器的数据发送并不经过网络连接部分，而是直接发出，这意味着每个任务处理函数都可以拿到直接与客户端通信的权限。这一点与客户端的通行方式不同。

## 注册请求处理

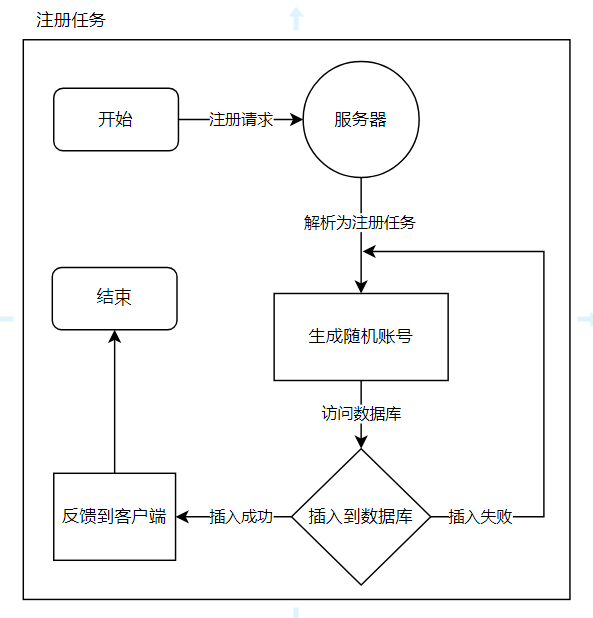


图 4.7 注册任务

从图4.7可知，服务器在处理客户端发起注册任务时的响应流程。服务器首先分析任务类型，得出为注册类型之后生成10位随机的正整数账号，并试图插入数据库。数据库的账号字段被设置成主键，不可重复插入，如果插入失败则代表生成的账号为重复值，重新生成并尝试插入。插入成功则将账号信息反馈到发起请求任务的客户端。

## 登录请求处理

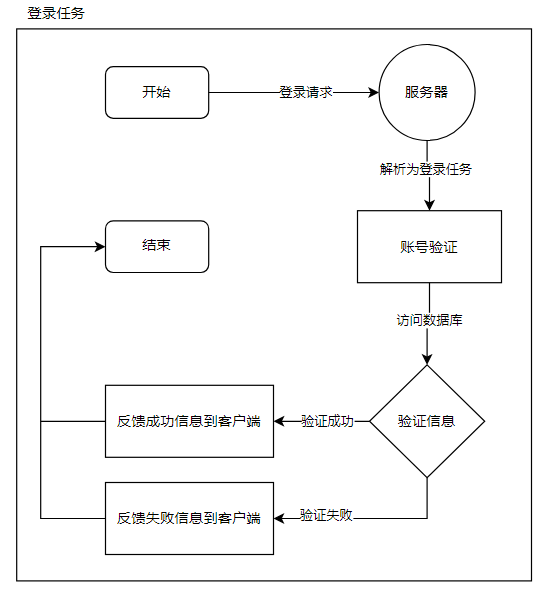


图 4.8 登录任务

由图4.8可看出服务器处理登录请求时的基本流程。服务器解析出登录任务后开始验证登录信息，登录信息存放在数据库中，通过查询数据库对比验证，判断客户端的登录信息是否与数据库内的信息一致。如果一致则将登录成功的内容反馈到客户端，否则反馈失败。

## 转发请求处理

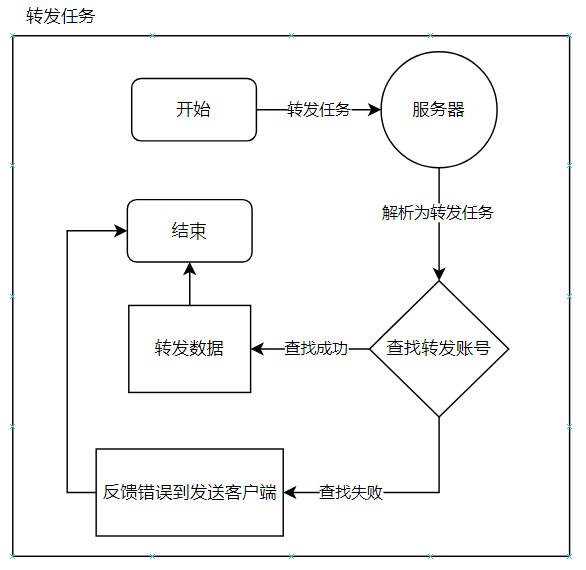


图 4.9 转发任务

由图4.9可知服务器处理转发任务的流程。当服务器解析出任务类型为转发任务时，调用转发处理函数，该函数通过转发目标账号，查询该账号是否登录，如果登录则转发到目标客户端。当目标账号未登录时，将返回提示信息到发送请求的客户端。

服务器在所有账号登录成功时，将登录账号与套接字绑定到登录索引容器，当服务器需要转发任务时，登录索引容器通过Key-Value的形式快速通过账号索引到套接字，此时将需要转发的数据写入该套接字即可完成转发。

# 4. 交换协议与数据模型

## 章节概要

本章主要讲解程序设计时数据交互的业务协议，以及业务协议对功能实现的影响。在网络程序中，所有的数据都需要遵循一定的协议才能被解读，新功能的制定也需要协议支持。应用层的业务协议是服务器与客户端之间的桥梁，通过协议搭建起一套完整的通信机制。

本程序使用了两个协议模型，分别是请求响应模式与数据转发模型，基于这两个模型已经实现了账号登录、发送消息、文件传输等一系列功能。接下来将接收功能的协议模型与功能的实现细节。

## 请求响应模型

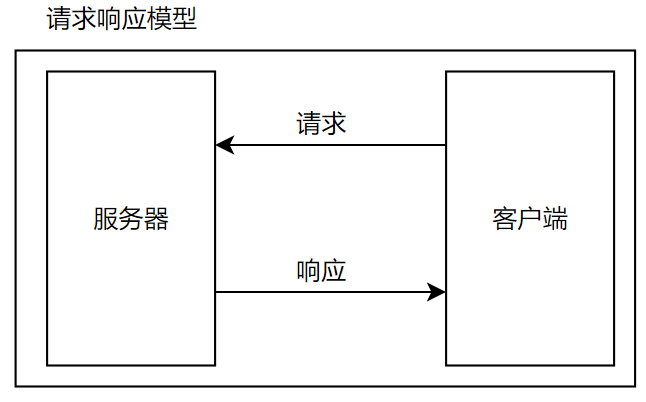


图 5.1请求响应模型

从图 5.1请求响应模型可以看出，客户单与服务器的数据传递方式非常的简单，是一问一答的形式。当客户端向服务器发送请求时，服务器会处理客户端的请求，并客户端发送响应包。这种形式的协议模型最常见的便是Web服务器与浏览器的信息交互，对功能的处理十分的简单有效，在注册、登录、退出等功能我们引用了这种通信形式。

## 数据转发模型

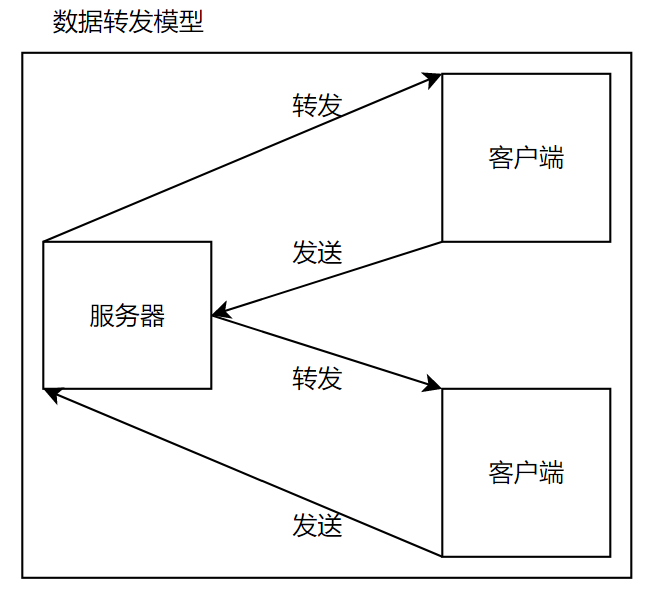


图 5.2 数据转发模型

从图 5.2 数据转发模型可以看出，数据转发模型用于多个客户端之间的数据转发。由于是S/C网络架构，客户端之间的数据必须通过服务器进行转发，便引用了该模型。数据转发模型在处理上，可以单方向转发，即客户端收到另一个客户端的消失时不会反馈，也可以在接收到信息之后发送确认反馈，反馈确认信息可以保证数据的到达状态。

数据转发模型被用于发送消息和文件传输功能上，其中发送消息采用了单向转发，即接收方不反馈。文件传输功能采用双向反馈，即接收到文件传输请求时，反馈接收状态，如是否接收文件、是否成功接收、是否接收完成等反馈确认信息。

## 交互协议结构

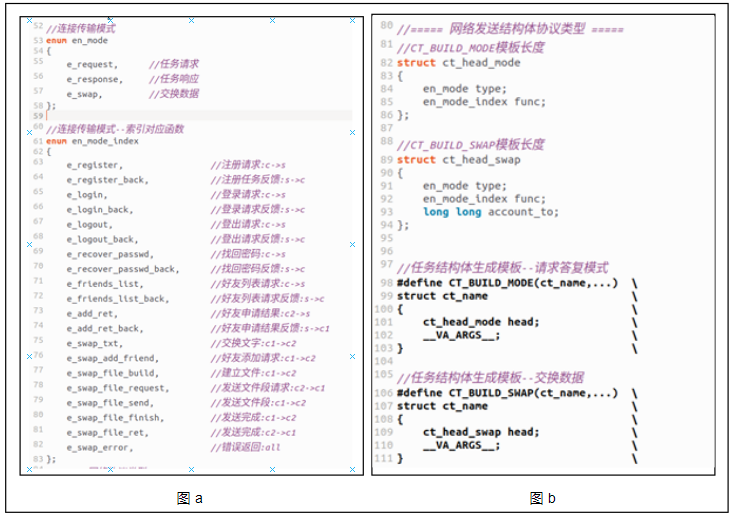


图 5.3 交互协议结构

在图 5.3 交互协议结构中，图a为协议类型与任务类型枚举，图b为交互协议传输结构体。传输结构体是网络编程中较为常用的数据打包手段，主要用于将信息放入结构体中，并将结构体转为二进制字节数组通过套接字发出，接收方接收到二进制字节数据后转成对应的结构体类型，从而完成数据打包。

从图b可以看出，结构体分为两类，其中ct\_head\_mode为请求响应模型的传输结构体，携带en\_mode与en\_mode\_index两个枚举类型的成员,分别用于接收方区分协议类型与请求任务类型。

ct\_head\_swap为数据转发模型的传输结构体，于请求响应模型结构类似，相比于请求响应模型结构体多了一个long long account\_to的成员变量，该成员用于表示数据转发的目标账号，当服务器接收到客户端的转发任务时，可快速锁定目标账号并转发。

# 5. 实测结果与讨论

## 章节概要

本章将展示该程序的测试结果，测试的内容主要是客户端的各个UI界面，其中包括登录与注册窗口、错误信息反馈窗口、好友列表与聊天窗口、发送消失时聊天窗口的变化等客户端窗口界面内容。在数据交换测试中，会依次对比发送、接收两端对表情包与文件的不同处理。

## 登录窗口测试



图 6.1 登录测试

在图 6.1 登录测试中，图a为登录界面，图b为注册界面。登录窗口与注册窗口都相对简单，其中登录窗口的输入密码部分使用了掩码进行防护，而设置密码与输入账号时采用显示模式，让用户可以清晰查看输入内容以防止错误。

在UI设计上，整个程序都采用了圆滑无边框设计，并增加暗黄色背景图，自定义的个性控件，其独特风格会贯穿整个程序，使UI保持统一风格。在UI控件的设计上，为了与背景融合，全部采用透明背景设计，与主流的UI界面设计不同。虽然背景透明，但各类控件依旧保证着显著特征，在不影响用户使用上增加个性化元素。



图 6.2 弹窗测试

在图 6.2 弹窗测试中，图a为注册申请反馈窗口，图b为登录失败弹窗。两者均为操作时的反馈窗口，用于提示用户对应操作反馈的结果，不管是操作成功还是操作失败都会得到反馈。

## 聊天窗口测试

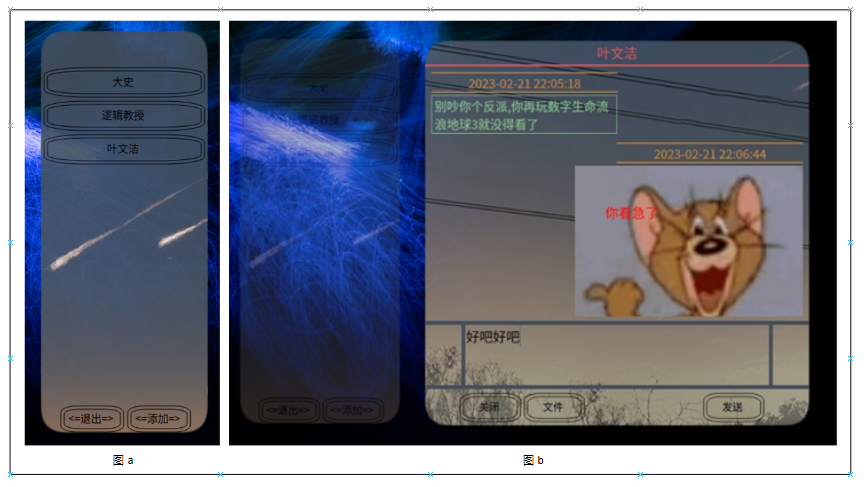


图 6.3 聊天窗口测试

在图 6.3 聊天窗口测试中，图a为好友列表窗口，图b为聊天窗口。其中好友列表窗口列出了所有好友，每位好友对应一个好友按钮以及聊天窗口，点击好友按钮会弹出聊天窗口。

从图b可以看出，聊天窗口分为显示区和输入区，红线与第一条蓝线为显示区，两根蓝线中间为输入区。在显示区可以显示接收到的消息以及表情包，发送区则可以输入文件并发送。当需要发送表情包时，只需要将表情包拖入到显示区即可发送。

从图b可以看出，当聊天窗口弹出时，好友列表几乎处于透明状态，其透明度的降低有利于用于将注意力转移到聊天窗口中。在整个程序中，只要窗口没有鼠标焦点，其透明度就会降低，此设计的初衷是帮助用户排除其他窗口的信息干扰，但需要是又可以快速调出。

## 文件进度测试



图 6.4文件进度测试

在图 6.4文件进度测试中，图a为接收方文件进度，图b为发送方文件进度。可以看出发送文件时，显示区窗口会右对齐进度条，并显示文件路径、发送当前进度，文件发送状态等信息。接收方除窗口对齐方式为左对齐之外，与发送方显示的信息格式几乎一致，其中显示的文件路径是保存路径。

## 好友申请测试



图 6.5 申请发送测试

在图 6.5 申请发送测试中，图a为聊天窗口中的申请好友按钮，图b为申请好友窗口。申请好友的方式为点击申请按钮，弹出申请窗口之后在添加输入框输入希望添加的好友账号，点击发送则向服务器发送申请好友的请求。

在图b中，第一个浅白色的输入框为本机的账号，当别人希望添加你为好友时，可通过输入该账号申请添加你为好友。



图 6.6 申请反馈测试

在图 6.6 申请反馈测试中，图a为好友申请成功的反馈，图b为申请成功之后好友列表的改动。当申请好友请求发出时，我们等待对方的确认，对方不管是同意或者是不同意都活反馈一个提示消息。本次测试为，当好友同意了请求，反馈了成功添加的窗口，当用户关闭窗口时，客户端重新向服务器请求好友列表信息，用于更新追加的好友按钮。

更新请求是双向的，不管是好友申请方还是发送方，在服务器接收到添加反馈时，同时向双方客户端下发好友列表信息，用于更新客户端的好友列表窗口。

# 6. 总结和展望

本文基于……………………技术设计了一款……………………。所设计的………………包含………………功能。经由实测，所设计…………能够实现预期功能，符合设计要求。但是在设计过程中，该…………也存在诸如………………的缺点，需要在后续过程中继续改正。

# 参考文献

1. Stanley B.LippmanBarbara E.MooJosée.C++primer 第三版[M]. 广东:中国电力出版社,2002:102-207.
2. 谢希仁.计算机网络[第7版] [M].北京:电子工业出版社,2017:303-359.
3. 鸟哥.鸟哥的Linux私房菜-基础篇 第四版[M].北京:[人民邮电出版社](https://book.douban.com/press/2609),2018:102-302.
4. 鸟哥.鸟哥的Linux私房菜服务器架设篇[M].北京:[人民邮电出版社](https://book.douban.com/press/2609),2008:56-97.
5. W.，Richard，Stevens .UNIX环境高级编程中文第三版[M].北京:[人民邮电出版社](https://book.douban.com/press/2609),2019:791-921.
6. W.，Richard，Stevens .UNIX网络编程卷1中文版[M].北京:[人民邮电出版社](https://book.douban.com/press/2609),2010:335-408.
7. 侯捷. STL源码剖析[M].湖北:华中科技大学出版社.2002,34-36.
8. Matthew H.Austern.泛型编程与STL[M].河北:中国电力出版社.2003:91-106.
9. 苏瑞、李牧.C语言程序设计[M].辽宁:大连理工大学出版社.2001:331-401.
10. 谭浩强.C语言程序设计(第五版)[M].北京:清华大学出版社.2017.
11. 马丁.代码整洁之道[M].北京:[人民邮电出版社](https://book.douban.com/press/2609).2011,229-302.
12. Martin Fowle.重构：改善既有代码的设计[M]. 北京:[人民邮电出版社](https://book.douban.com/press/2609).2011:77-92.
13. 徐胜昭.基于C\_S模式的局域网络聊天软件系统的设计与实现[D].电子科技大学,2012:1-9.
14. 董敏.基于Linux的网络聊天系统的设计[J].南京航空航天大学.2017:5-8.
15. 邓亚君.基于Python的网络聊天室设计[J].西华大学,2018:3-15.

# 致谢

四年的学习生活即将结束，回首往事，难以忘怀在这四年的学习和生活中给予我关怀和支持的老师和同学们。

我能够顺利地完成本科阶段的学习，首先要感谢所有任课老师、指导老师、热心相助的同学、康概解疑的前辈与相互帮衬的同事，感谢他们的乐于助人、无私奉献的精神。

在开始编写论文时遇到了很多的困难，而对于如何解决这些困难我却一无所知，辛亏得到了指导老师的悉心教导与专业课程的任课老师耐心付出才最终写下这一份论文。在一次次的遇到困难与解决困难的过程中我学到了很多，对于从无到有的知识累计与逐步获得成就感的学习过程感到非常高兴。

在撰写论文的时间里，正好也是实习期间，在实习时认识到了很多前辈与同事。在专业知识的实际运用时，常常遇到不懂的地方，当遇到疑惑时向前辈请教都得到了他们慷慨的解答与帮助。他们的帮助让我认识到实际运用与理论的差距，而这些知识与经验最终都帮助我完成了毕设的课题，也帮助我对人生这一重大课题做了进一步解答，在此非常感谢他们的热心帮助。

# 附录 A 完整的转发协议文件

#ifndef WEB\_PROTOCOL\_H

#define WEB\_PROTOCOL\_H

//#include "web\_protocol.h"

//===== 解析结构体 =====

//解体结构体信息

#include <string>

//结构体转string

template <class T\_ct>

static std::string to\_str(T\_ct ct)

{ return std::string((char\*)&ct,sizeof(T\_ct)); }

//string转结构体

template <class T\_ct>

static void to\_ct(const std::string &str,T\_ct &ct)

{ ct = \*(T\_ct\*)str.c\_str(); }

//===== 解析结构体 =====

//===== 网络协议类型 =====

//连接传输模式

enum en\_mode

{

e\_request, //任务请求

e\_response, //任务响应

e\_swap, //交换数据

};

//文件转发类型

enum en\_build\_file

{

e\_spic, //图片表情包

e\_file, //普通文件

};

//文件转发类型

enum en\_swap\_error

{

e\_error\_disconnect, //目标未连接:s->(c1,c2)

e\_error\_swap, //转发失败:s->(c1,c2)

e\_error\_build, //建立文件-错误反馈 :s->(c1)

e\_error\_send, //发送文件段-错误反馈 :s->(c1)

e\_error\_request, //发送文件段-错误反馈 :s->(c2)

e\_error\_finish, //发送完成-错误反馈 :s->(c1)

e\_error\_finish\_back, //接收文件完整性错误 :s->(c2)

};

//连接传输模式--索引对应函数

enum en\_mode\_index

{

e\_register, //注册请求:c->s

e\_register\_back, //注册任务反馈:s->c

e\_login, //登录请求:c->s

e\_login\_back, //登录请求反馈:s->c

e\_logout, //登出请求:c->s

e\_logout\_back, //登出请求反馈:s->c

e\_recover\_passwd, //找回密码:c->s

e\_recover\_passwd\_back, //找回密码反馈:s->c

e\_friends\_list, //好友列表请求:c->s

e\_friends\_list\_back, //好友列表请求反馈:s->c

e\_add\_ret, //好友申请结果:c2->s

e\_add\_ret\_back, //好友申请结果反馈:s->c1

e\_swap\_txt, //交换文字:c1->c2

e\_swap\_add\_friend, //好友添加请求:c1->c2

e\_swap\_file\_build, //建立文件:c1->c2

e\_swap\_file\_request, //发送文件段请求:c2->c1

e\_swap\_file\_send, //发送文件段:c1->c2

e\_swap\_file\_finish, //发送完成:c1->c2

e\_swap\_file\_ret, //发送完成:c2->c1

e\_swap\_error, //错误返回:all

};

//===== 网络协议类型 =====

//===== 网络发送结构体协议类型 =====

//CT\_BUILD\_MODE模板长度

struct ct\_head\_mode

{

en\_mode type;

en\_mode\_index func;

};

//CT\_BUILD\_SWAP模板长度

struct ct\_head\_swap

{

en\_mode type;

en\_mode\_index func;

long long account\_to;

};

//任务结构体生成模板--请求答复模式

#define CT\_BUILD\_MODE(ct\_name,...) \

struct ct\_name \

{ \

ct\_head\_mode head; \

\_\_VA\_ARGS\_\_; \

} \

//任务结构体生成模板--交换数据

#define CT\_BUILD\_SWAP(ct\_name,...) \

struct ct\_name \

{ \

ct\_head\_swap head; \

\_\_VA\_ARGS\_\_; \

} \

//快速声明结构体--请求(参数1：变量名，参数2：半截枚举)

#define MAKE\_CT\_REQ(in\_name,in\_type) \

ct\_##in\_type in\_name; \

in\_name.head.type = en\_mode::e\_request; \

in\_name.head.func = en\_mode\_index::e\_##in\_type \

//快速声明结构体--应答(参数1：变量名，参数2：半截枚举)

#define MAKE\_CT\_RES(in\_name,in\_type) \

ct\_##in\_type in\_name; \

in\_name.head.type = en\_mode::e\_response; \

in\_name.head.func = en\_mode\_index::e\_##in\_type \

//快速声明结构体--交换(参数1：变量名，参数2：半截枚举，参数3：目标账号)

#define MAKE\_CT\_SWAP(in\_name,in\_type,in\_account) \

ct\_##in\_type in\_name; \

in\_name.head.type = en\_mode::e\_swap; \

in\_name.head.func = en\_mode\_index::e\_##in\_type; \

in\_name.head.account\_to = in\_account \

//==客户与服务器==

//注册请求

CT\_BUILD\_MODE(ct\_register,

char passwd[64];

char name[64];

);

//注册任务反馈

CT\_BUILD\_MODE(ct\_register\_back,

long long account;

bool is\_success;

char passwd[64];

);

//登录请求

CT\_BUILD\_MODE(ct\_login,

long long account;

char passwd[64];

);

//登录请求反馈

CT\_BUILD\_MODE(ct\_login\_back,

long long account;

bool is\_success;

);

//登出请求

CT\_BUILD\_MODE(ct\_logout,

long long account;

);

//登出请求反馈

CT\_BUILD\_MODE(ct\_logout\_back,

long long account;

bool is\_success;

);

//找回密码

CT\_BUILD\_MODE(ct\_recover\_passwd,

long long account;

);

//找回密码反馈

CT\_BUILD\_MODE(ct\_recover\_passwd\_back,

long long account;

bool is\_success;

char passwd[64];

);

//好友列表请求

CT\_BUILD\_MODE(ct\_friends\_list,

long long account;

);

//好友列表请求反馈

CT\_BUILD\_MODE(ct\_friends\_list\_back,

bool is\_end;

long long account;

char name[64];

);

//好友添加请求结果

CT\_BUILD\_MODE(ct\_add\_ret,

long long account\_from;

long long account\_to;

bool is\_agree;

);

//好友添加请求结果

CT\_BUILD\_MODE(ct\_add\_ret\_back,

long long account\_from;

bool is\_agree;

bool is\_self;

);

//==客户与服务器==

//==客户与客户==

//交换文字

CT\_BUILD\_SWAP(ct\_swap\_txt,

long long account\_from;

char buf\_txt[1024];

);

//好友添加请求

CT\_BUILD\_SWAP(ct\_swap\_add\_friend,

long long account\_from;

);

//建立文件

CT\_BUILD\_SWAP(ct\_swap\_file\_build,

long long account\_from;

long long size\_block; //发送块大小

long long size\_file; //文件总大小

en\_build\_file type;

char path[255];

char filename[255];

);

//发送文件段请求

CT\_BUILD\_SWAP(ct\_swap\_file\_request,

long long account\_from;

long long block\_count;

char filename[255];

);

//发送文件段

CT\_BUILD\_SWAP(ct\_swap\_file\_send,

long long account\_from;

long long off\_file; //当前文件偏移

long long size\_buf; //本次发送的真实字节

bool is\_next;

char filename[255];

char buf[4096];

);

//发送完成

CT\_BUILD\_SWAP(ct\_swap\_file\_finish,

long long account\_from;

char filename[255];

);

//发送完成反馈

CT\_BUILD\_SWAP(ct\_swap\_file\_ret,

long long account\_from;

en\_build\_file type;

bool is\_success;

char filename[255];

);

//错误反馈

CT\_BUILD\_SWAP(ct\_swap\_error,

long long account\_from;

en\_swap\_error err;

en\_mode\_index swap\_func;

);

//==客户与客户==

#endif // WEB\_PROTOCOL\_H

了解一份完整的通信协议非常重要，代码中利用CT\_BUILD\_MODE和CT\_BUILD\_SWAP宏代码快速生成请求响应模型和数据转发模型的结构体代码。

# 附录 B 服务器任务分发代码

1. 头文件任务声明

//===== 任务函数 =====

map<en\_mode\_index,func\_task> map\_task\_func; //任务函数查询容器

void task\_register(const web\_sock& sock, const string& meg); //账号注册

void task\_login(const web\_sock& sock, const string& meg); //登录请求

void task\_logout(const web\_sock& sock, const string& meg); //登出请求

void task\_recover\_passwd(const web\_sock& sock, const string& meg); //忘记密码

void task\_friends\_list(const web\_sock& sock, const string& meg); //好友列表

void task\_swap(const web\_sock& sock, const string& meg); //数据交换

void task\_add\_ret(const web\_sock& sock, const string& meg); //好友申请

1. 构造函数添加任务到容器

//===== 添加任务函数到容器 =====

//==原型:

//map\_task\_func.insert(pair<en\_mode\_index,func\_task>

// (en\_mode\_index::e\_login,bind(&ux\_web\_server::task\_login,this,\_1,\_2)));

//===== 添加任务函数到容器 =====

#define MAP\_TASK\_ADD(func) \

map\_task\_func.insert(pair<en\_mode\_index,func\_task> \

(e\_##func,bind(&ux\_web\_server::task\_##func,this,\_1,\_2)))

MAP\_TASK\_ADD(register); //账号注册

MAP\_TASK\_ADD(login); //登录请求名

MAP\_TASK\_ADD(logout); //登出请求

MAP\_TASK\_ADD(recover\_passwd); //忘记密码

MAP\_TASK\_ADD(friends\_list); //好友列表

MAP\_TASK\_ADD(add\_ret); //好友申请

//===== 添加任务函数到容器 =====

1. 事件分发函数

void ux\_web\_server::on\_message(const web\_sock &sock, const string &meg)

{

//执行匹配的任务函数

ct\_head\_mode ct;

to\_ct(meg,ct);

if(ct.type == en\_mode::e\_swap) //进入交换函数

{ task\_swap(sock,meg); }

else if(ct.type == en\_mode::e\_request) //进入应答函数

{

auto it\_func = map\_task\_func.find(ct.func);

if(it\_func != map\_task\_func.end())

{ (std::bind(it\_func->second,sock,meg))(); }

else { vloge("not find func: e\_request func"); }

}

else //未发现协议

{ vloge("on\_message not find: " vv(ct.type) vv(ct.func)); }

}

服务器任务分发代码分为三个部分，分别是第一部分，头文件任务声明，该部分在服务器头文件列出所有任务的处理函数。

第二分部，构造函数添加任务到容器，这段代码在服务器构造函数执行时，插入到map容器中，其中任务处理函数作为Value，任务类型为Key。

第三部分，事件分发函数，每当套接字接收到消息时，分解出任务类型，并将任务类型通过map查询与之绑定的任务处理函数，如果存在则执行任务处理函数完成事件分发。

# 附录 C 客户端的任务转发代码

//发送--用户输入信息

connect(v\_friends\_list,&wid\_friends\_list::fa\_send\_news,this,

[=](en\_info en,long long account,QString info){

if(en == en\_info::e\_send\_txt)

{

vlogf("e\_send\_txt" vv(account) vv(qtos(info)));

v\_net->ask\_swap\_txt(account,info);

}

else if(en == en\_info::e\_send\_pic)

{

vlogf("e\_send\_pic" vv(account) vv(qtos(info)));

QFileInfo info\_file(info);

v\_net->ask\_swap\_file(account,info\_file.fileName(),info,en\_build\_file::e\_spic);

}

else if(en == en\_info::e\_send\_file)

{

vlogf("e\_send\_file" vv(account) vv(qtos(info)));

QFileInfo info\_file(info);

v\_net->ask\_swap\_file(account,info\_file.fileName(),info,en\_build\_file::e\_file);

}

});

//发送--添加好友请求

connect(v\_friends\_list,&wid\_friends\_list::fa\_add\_friend,this,

[=](long long account){

v\_net->ask\_swap\_add\_friend(account);

});

//发送--登陆

connect(v\_login,&wid\_login::fa\_login,this,

[=](long long account,QString passwd){

vlogf("fa\_login" vv(account) vv(qtos(passwd)));

v\_net->ask\_login(account,passwd);

});

//发送--注册

connect(v\_login,&wid\_login::fa\_register,this,

[=](QString name,QString passwd){

vlogf("fa\_register" vv(qtos(passwd)) vv(qtos(name)));

v\_net->ask\_register(passwd,name);

});

//===== 网络反馈 =====

//==服务器响应

connect(v\_net,&net\_connect::fa\_open,this,[=](){

vlogf("fa\_open");//

});

connect(v\_net,&net\_connect::fa\_close,this,[=](){

vlogf("fa\_close");//

wid\_dialog dia("网络连接断开[确认并重新连接]");

dia.exec();

if(dia.get\_status())

{ v\_net->open\_connect(ip,port); }

else { exit(-1); }

});

connect(v\_net,&net\_connect::fa\_register\_back,this,

[=](long long account,QString passwd,bool ok){

vlogf("fa\_register" vv(account) vv(qtos(passwd)) vv(ok));//

v\_login->show\_register\_back(ok,account);

});

connect(v\_net,&net\_connect::fa\_login\_back,this,

[=](long long account,bool ok){

vlogf("fa\_login\_back" vv(account) vv(ok));

if(ok) { v\_net->ask\_friends\_list(v\_net->get\_account()); }

else

{

wid\_dialog dia("登录失败[帐号或者密码错误]");

dia.exec();

}

});

connect(v\_net,&net\_connect::fa\_logout\_back,this,

[=](long long account,bool ok){

vlogf("fa\_logout\_back" vv(account) vv(ok));

if(ok) v\_net->close\_connect();

});

connect(v\_net,&net\_connect::fa\_recover\_passwd\_back,this,

[=](long long account,QString passwd,bool ok){

vlogf("fa\_recover\_passwd\_back" vv(account) vv(qtos(passwd)) vv(ok));//

if(ok) v\_net->close\_connect();

});

connect(v\_net,&net\_connect::fa\_friends\_list\_back,this,

[=](QMap<long long,string> map){

//第一次进入时的窗口状态切换

if(v\_login->isHidden() == false)

{

v\_friends\_list->set\_account(v\_net->get\_account());

v\_friends\_list->move(v\_login->pos());

v\_login->close();

}

v\_friends\_list->into\_friends(map);

v\_friends\_list->show();

});

connect(v\_net,&net\_connect::fa\_add\_ret\_back,this,

[=](long long account,bool ok,bool self){

if(self == true) { v\_net->ask\_friends\_list(v\_net->get\_account()); }

else

{

QString info;

if(ok) { info = "成功添加账号: " + QString::number(account); }

else { info = "对方不同意添加为好友: " + QString::number(account); }

wid\_dialog temp(info);

temp.exec();

if(ok) { v\_net->ask\_friends\_list(v\_net->get\_account()); }

}

});

//交换函数

connect(v\_net,&net\_connect::fa\_swap\_txt,this,

[=](long long account\_from,QString txt){

vlogf("fa\_prog\_send" vv(account\_from) vv(qtos(txt)));//

v\_friends\_list->into\_news(en\_info::e\_send\_txt,account\_from,txt);

});

connect(v\_net,&net\_connect::fa\_swap\_add\_friend,this,

[=](long long account\_from){

QString info = "同意添加[ "+ QString::number(account\_from) +" ]为好友吗";

wid\_dialog temp(info);

temp.exec();

v\_net->ask\_add\_ret(account\_from,temp.get\_status());

});

connect(v\_net,&net\_connect::fa\_swap\_build,this,

[=](long long account\_from,QString filename,en\_build\_file type){

if(type == en\_build\_file::e\_file)

{

vlogf("fa\_prog\_send: " vv(account\_from) vv(qtos(filename)));//

v\_friends\_list->into\_news(en\_info::e\_send\_file,account\_from,filename);

}

});

connect(v\_net,&net\_connect::fa\_swap\_file\_finish,this,

[=](long long account\_from,QString filename,en\_build\_file type,bool is\_ok){

if(type == en\_build\_file::e\_file)

{

//文件进度(需要三个字段:文件名,进度值,是否完成(prog/finish))(分割符:##)

QString info = filename+"##100##1";

v\_friends\_list->into\_news(en\_info::e\_send\_file\_prog,account\_from,info);

}

else if(type == en\_build\_file::e\_spic)

{ v\_friends\_list->into\_news(en\_info::e\_send\_pic,account\_from,filename); }

vlogd("fa\_swap\_file\_finish" vv(account\_from) vv(qtos(filename)) vv(type) vv(is\_ok));//

});

connect(v\_net,&net\_connect::fa\_swap\_file\_ret,this,

[=](long long account\_from,QString filename,en\_build\_file type,bool is\_ok){

if(type == en\_build\_file::e\_file)

{

//文件进度(需要三个字段:文件名,进度值,是否完成(prog/finish))(分割符:##)

QString info = filename+"##100##1";

v\_friends\_list->into\_news(en\_info::e\_send\_file\_prog,account\_from,info);

}

vlogd("fa\_swap\_file\_ret" vv(account\_from) vv(qtos(filename)) vv(type) vv(is\_ok));//

});

connect(v\_net,&net\_connect::fa\_swap\_error,this,

[=](long long account\_from,en\_swap\_error err){

vlogf("fa\_swap\_error" vv(account\_from) vv(err));//

});

connect(v\_net,&net\_connect::fa\_prog\_send,this,

[=](QString filename,long long account\_from,int now){

//文件进度(需要三个字段:文件名,进度值,是否完成(prog/finish))(分割符:##)

QString info = filename+"##"+QString::number(now)+"##0";

v\_friends\_list->into\_news(en\_info::e\_send\_file\_prog,account\_from,info);

});

connect(v\_net,&net\_connect::fa\_prog\_recv,this,

[=](QString filename,long long account\_from,int now){

//文件进度(需要三个字段:文件名,进度值,是否完成(prog/finish))(分割符:##)

QString info = filename+"##"+QString::number(now)+"##0";

v\_friends\_list->into\_news(en\_info::e\_send\_file\_prog,account\_from,info);

});

客户端的任务代码转发部分主要是由Qt的UI界面线程到网络连接线程的数据转发与任务封装。

在GUI程序中，由于Widget类组件事件循环中等待，网络连接线程直接操控Widget类组件会使得程序直接崩溃，必须通过Qt的connect信号转发，或者自定义事件触发事件循环才可以将网络线程的数据发送到GUI线程。客户端任务转发部分代码就是Qt主线程跟网络连接线程的胶水层，非常重要。

广西民族师范学院本科毕业论文(设计)诚信保证书

（学生用）

对于毕业论文（设计）《基于QT的广域网通讯软件开发》，本人郑重承诺：其内容真实、可靠，系本人在 郭敏杰 教师的指导下独立完成；如果被查出有弄虚作假、抄袭剽窃行为，本人愿承担全部责任。

承诺者（签名）：

年 月 日