

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 |  |
| 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** |

**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通过讲简单文字通信流程、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；

**Abstract**

In order to solve the cross-platform network communication problem with the emergence of Linux desktop operating system, this paper designs a cross-platform network communication platform based on S/C network architecture. The communication platform software designed by the simple text communication process, file transmission and other functions integrated in the designed UI interface, can easily solve the Windows system and Linux interworking problem. The measured software shows that the designed system can realize the intended function, which is conducive to promoting the development of the Internet and computer field.

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

目录

[1 前言 1](#_Toc131522045)

[1.1 课题背景及意义 1](#_Toc131522046)

[1.2 网络聊天的概念 1](#_Toc131522047)

[1.3 网络聊天发展现状 2](#_Toc131522048)

[1.4 课题内容以及难点 2](#_Toc131522049)

[1.5 本文的组织结构 3](#_Toc131522050)

[2 平台开发和技术介绍 4](#_Toc131522051)

[2.1 项目简介 4](#_Toc131522052)

[2.2 基础网络协议简介与分析 4](#_Toc131522053)

[2.3 WebSockect协议简介 5](#_Toc131522054)

[2.4 IO复用技术 5](#_Toc131522055)

[2.5 数据库分析与使用 6](#_Toc131522056)

[2.6 网络开源库 7](#_Toc131522057)

[2.7 开发环境以及工具 8](#_Toc131522058)

[3 跨平台软件设计 9](#_Toc131522059)

[3.1 跨平台问题 9](#_Toc131522060)

[3.2 操作系统API跨平台 9](#_Toc131522061)

[3.3 数据长度跨平台 10](#_Toc131522062)

[4 分析软件需求 12](#_Toc131522063)

[4.1 通信聊天软件的功能简介 12](#_Toc131522064)

[4.2 主流实现的基本通信功能 12](#_Toc131522065)

[5 客户端具体实现与基本流程 13](#_Toc131522066)

[5.1 注册流程 13](#_Toc131522067)

[5.2 登录状态图与控件图介绍 14](#_Toc131522068)

[5.3 交换流程 15](#_Toc131522069)

[5.4 交换状态图与处理图介绍： 16](#_Toc131522070)

[5.5 数据交换模型 17](#_Toc131522071)

[5.6 本章小结 18](#_Toc131522072)

[6 服务器具体实现与基本流程 20](#_Toc131522073)

[6.1 请求响应模式简介 20](#_Toc131522074)

[6.1.1 注册请求处理 20](#_Toc131522075)

[6.1.2 登录请求处理 21](#_Toc131522076)

[6.2 数据交换流程 22](#_Toc131522077)

[6.3 本章小结 22](#_Toc131522078)

[7 软件测试与效果展示 23](#_Toc131522079)

[7.1 类关系图 23](#_Toc131522080)

[7.2 注册与登录界面展示 23](#_Toc131522081)

[7.3 聊天窗口展示 25](#_Toc131522082)

[7.4 发送文件与发送表情包 27](#_Toc131522083)

[7.5 添加好友 29](#_Toc131522084)

[7.6 本章小结 31](#_Toc131522085)

[8 软件设计的前置知识 32](#_Toc131522086)

[8.1 内容概要 32](#_Toc131522087)

[8.2 客户端开发前置知识： 32](#_Toc131522088)

[8.3 服务器开发前置知识 33](#_Toc131522089)

[9 编程代码实现与细节： 34](#_Toc131522090)

[9.1 程序入口 34](#_Toc131522091)

[9.2 信号转发 36](#_Toc131522092)

[9.3 网络通信 37](#_Toc131522093)

[9.3.1 客户端 37](#_Toc131522094)

[9.3.2 服务端 39](#_Toc131522095)

[9.4 文件传输协议 40](#_Toc131522096)

[9.5 本章小结 42](#_Toc131522097)

[10 参考文献 43](#_Toc131522098)

[11 致谢 44](#_Toc131522099)

# 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图形框架开发的广域网通信工具。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。

# 研究思路和方案设计

## 项目简介

本项目是主要实现一个聊天软件，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。

该项目只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。

## 程序通信架构设计

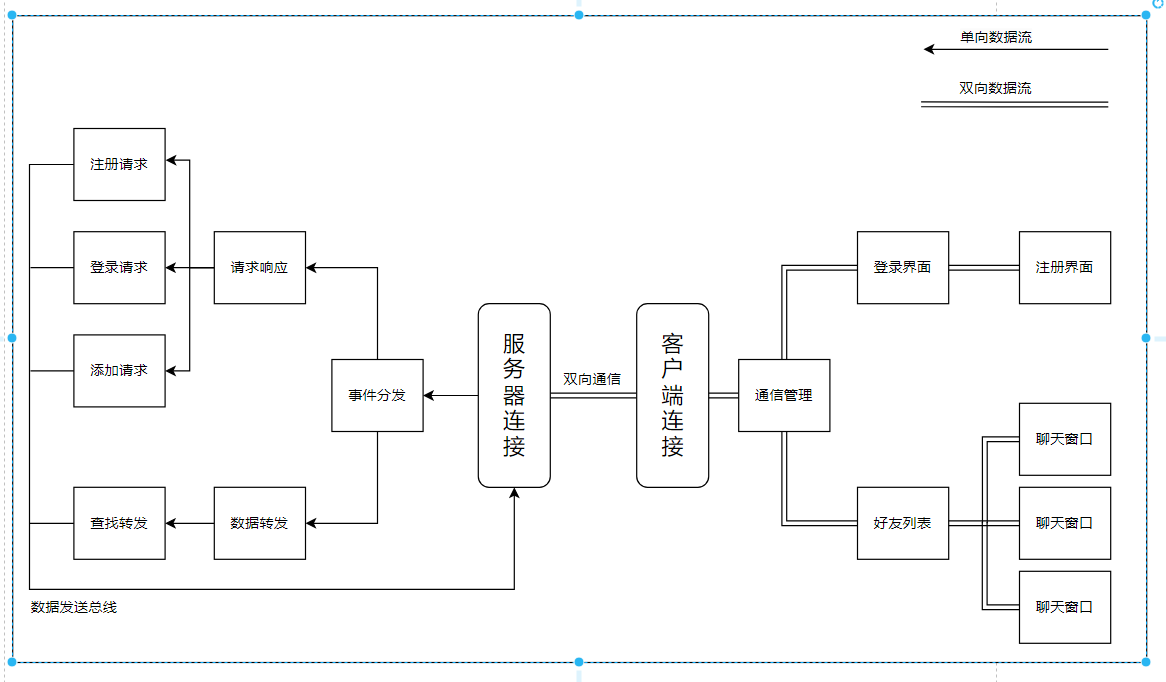


图 2.1 通信架构

由图 2.1 通信架构得知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。

## 客户端架构设计

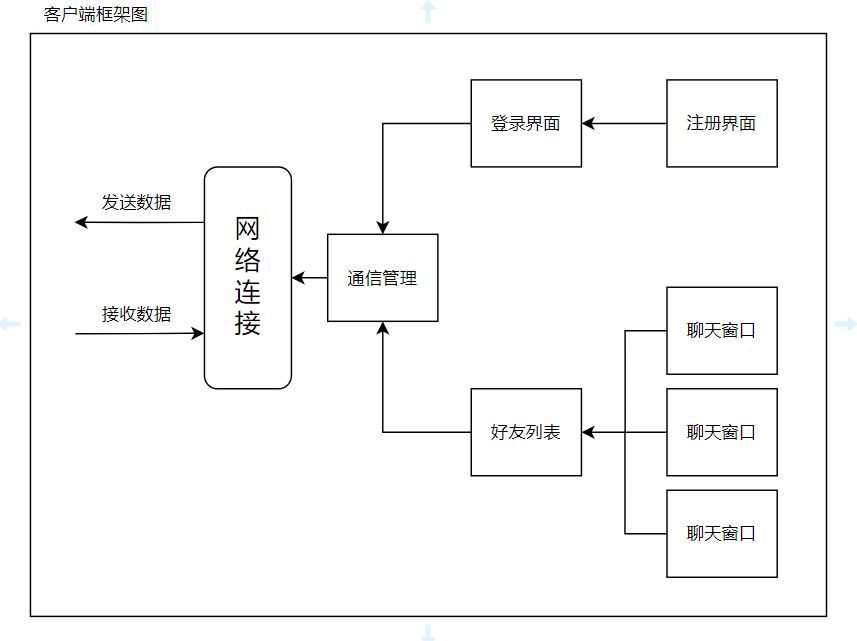


图 2.2 客户端架构

由图 2.1 客户端架构可以看出，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。

## 服务器架构设计

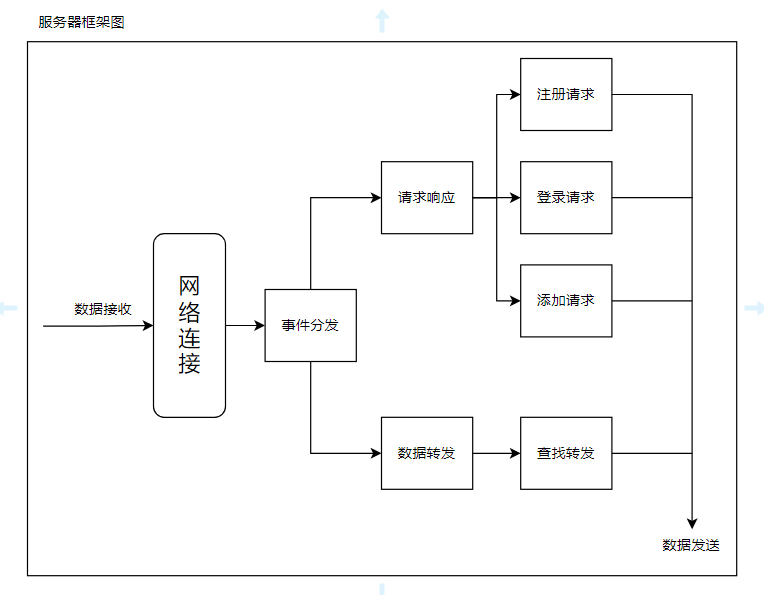


图 2.3 服务器架构

由图 2.2 服务器架构可以得知，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。

## 技术需求分析

本项目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。下一章将对技术需求做详细探讨。

# 技术探讨与选择

## 网络协议

### 网络协议简介

既然要实现网络聊天功能，网络间的通信协议便是不可不谈的话题，网络协议是计算机之间通信的桥梁。在目前的主流的网络协议分别为TCP协议、UPD协议、HTTP协议、WebSockect协议。其中TCP协议和UPD协议是传输层的网络协议，应用范围非常广泛，大部分的通信软件所使用的网络协议，为其中一种很或者两者皆有。HTTP协议和WebSockect协议则是建立在TCP协议的应用层通信协议，主要用于服务器与浏览器的通信。接下里将对比各个协议之间的优点与劣势。

### TCP协议

TCP提供的面向连接、提供可靠交付、面向字节流、稳定且安全可靠协议解决了应用层的零丢失问题，所以TCP协议才会被大量使用。尽管TCP协议需要三次握手建立连接，四次挥手的操作，而且是面向字节流，在应用层开发上会存在粘包以及压栈等问题，不过凭借着在传输层的滑动窗口的拥塞控制、全双工的通信连接减少了应用层对于网络流量的管理，而且使用连续ARQ的重传机制提速，速度并不慢，由此看来TCP协议被广泛采用不无道理。

### UPD协议

UDP提供了无连接、尽最大努力交付、面向报文、是快速且不安全功能，其特点就是快速，且是无连接无反馈的快速发送，但缺点也是无连接造成的保报文丢失等不可靠因素，需要应用层实现可靠通信。在网络环境良好的区域上，建立在可靠的UDP协议基础之上的应用协议将得到非常优秀交换体验，而当网络不稳定时，应用层实现的可靠和重发功能往往会将UDP的快速优势清扫的一干二净，速度比TCP还低数倍。

### HTTP协议

HTTP协议最开始是出现在浏览器上的一种基于TCP的应用层协议，该协议的主要任务就是用浏览器向数据服务器发送请求报文，服务器根据报发内容反馈内容并马上断开连接。所以HTTP协议的特点就是简单快速、灵活且无连接、无状态的一种短时TCP连接。其中无连接无状态指的是HTTP建立了请求/响应模型来对服务器进行通信，一旦本次内容提交并由服务器反馈完成则不保留如何连接记录。浏览器想保存信息就必须将全部的内容重新发送，而这就是浏览器会使用Cookies记录的原因，Cookies只是保证了客户端的信息记录，对于服务器来说依旧是一次全新的对话。

### WebSockect协议

WebSockect协议是一种基于TCP的应用层协议，与HTTP协议一样用于服务器与浏览器的通信。WebSockect协议主要解决了HTTP协议推送困难的问题。HTTP协议想要实现推送功能，所用的技术是短轮询。轮询是指由浏览器每隔一段时间向服务器发出 HTTP 请求，然后服务器将新数据到浏览器。之所以由浏览器定时请求服务器的原因就在于HTTP协议是短连接、无状态的，服务器无法主动向浏览器发送内容，只能被动回应。为了解决这个问题WebSockect协议采用了长连接有状态的连接方式，保留浏览器信息，使得服务器可以主动的向浏览器推送消息。

### 协议选择

本项目选择WebSockect协议作为主要通信协议的原因很简单，因为WebSockect协议可以由服务器主动向浏览器推向消息，且协议包头只有较小的开销，支持二进制内容的传输，为文件传输留下方便，内容是带传输格式，采用WebSockect协议使得网络部分的开发相对简单。

## 高并发技术

### 高并发服务器开发简介

高并发指的是服务器在短时间内遇到大量操作请求的情况，高并发服务器开发则尽最大可能的处理更多的请求。在实现通信软件之前，必须考虑到服务器通信的承载量，保证客户端的高并发需要，以及最大可能的发挥出服务器性能。

服务器早起采用的并发技术为多进程服务器与多线程服务器。多进程服务器指的是服务器一旦建立新连接则开启一个新进程与客户端连接，并完成通信，主进程继续监听。多线程服务器的连接处理类似，只不过由建立新进程改为了建立新线程，但不管是建立进程还是线程，对于服务器的资源消耗都相当高，于是就引出了IO复用技术。

IO复用技术，即用一个线程处理大量的IO操作，可以减少服务器的资源消耗，让服务器可以承载更多的IO连接与处理请求。主流的IO复用技术有三种，分别是select技术、poll技术和epoll技术，接下来将分别介绍这三种技术。

### select技术

select技术是将已经连接的套接字放入select连接集合进行管理， select将轮询整个连接集合，即连接集合中的套接字主动进行轮询监测，一旦发现连接的套接字可写或可读则对套接字进行标记，提示开发者套接字可操作。

select技术存在一个缺点，即连接数不能超过1024，也就是说最大只能接收1024个套接字，其原因是因为连接数量增大会影响性能。

### poll技术

poll技术与select技术十分类型，是对select技术的优化，提供了无限制的套接字连接数量，但连接数的增加依旧会影响性能，原因是poll技术与select技术一样，采用无差别轮询，即只有轮询到套接字才知道该套接字是否可读或者可写。

采用无差别轮询原因是因为知道某一个套接字可读或者可写，但不知道具体是哪一个套接字。两者都在不活跃的套接字上消耗了太多的时间，从而导致连接数影响了性能。不活跃指的是套接字不可读，活跃指套接字可读，且数据已经送往内核缓冲区。

### epoll技术

epoll技术是与select技术、poll技术相同的是，依靠连接集合管理大量已经连接的套接字，并进行监控。不同的是，通知方式不再是轮询而是改为事件驱动，只有活跃的套接字才会加入事件循环，使得epoll的事件循环中的每一个轮询都有事件可以处理，其中包括可读、可写、连接、关闭等事件。epoll技术不会被不活跃的套接字影响性能，使得无限制连接成为可能，让服务器轻易可处理高达百万的连接请求。

### 技术选择

介绍了IO复用技术select、poll和epoll的特点之后，本次项目选择了epoll技术作为服务器的高并发解决方案，因为其独特的机制可以提供更高的连接数量，以及减少服务器的性能开销。尽管在小型项目中，三者都可以轻易完成对服务器部署的需求，但选择epoll技术则表现出了对于未来前瞻性以及期盼。

## 数据库

### 数据库简介

数据库是指以一定方式储存在一起、能与多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合。数据库的主要用于存储大量的数据信息，且与应用程序实现分离，并能快速查询数据和操作数据。

主流的数据库有关系型数据库MySql、Oracle、SqlServer、Sqlist3、Access等，其中MySql、Oracle、SqlServer数据库需要与数据库服务器进行连接，此类数据库用于大规模数据或超大规模数据管理与存储。Sqlist、Access数据库则是轻量级，用于存储小量数据，甚至是文本级别的微量数据。

### 数据库选择

该项目对数据库的选择主要用于服务器的存储，考虑到服务器的部署问题与服务器数据量较小等因素，最后选择Sqlist3数据库主要数据存储。选择Sqlist3数据库的理由很简单，Sqlist3数据库无需连接服务器，产生的db数据库文件可以随时迁移，且因为受到广泛使用已经有一套完善的API可用于开发，查询速度与MySql媲美，非常适合小型项目作为主选数据库。

## 网络开源库

### 为什么选择开源库

网络编程绝不是套接字的发送与接收这么简单，服务器开发有一套完善的规则，其中开源库主要是为网络编程搭建一个框架，搭建一个高性能服务器程序框架，一个高性能服务器离不开高性能IO、IO复用、定时器、多线程编程、线程池等必要组件，以及协议解析、网络数据传输、解封包等数据协议的封装与解析。以及网络编程存在的问题，接下来将列举出网络编程的常见问题。

### 网络编程的常见问题

1. TCP粘包问题，非常常见，因为是TCP是面向流字节传输的，数据无边界，如果对数据进行封装，会导致收发数据的结构不对等。
2. UDP报文丢失问题，使用UDP是必须在应用层实现可靠的重发策略，保证数据可靠或者选择性丢弃。
3. 超时重传问题，因为网络复杂，偶尔会出现延时导致数据无法及时发送或者接收，导致超时，需要在超时后选择断开连接或者重新传输数据。
4. TCP长连接问题，虽然TCP底层已经存在保活设置，不过要是由于硬件设备的突然中断所导致的连接断开则不反馈，所以应有层需要实现自己的心跳（或称ping-pong）机制来保持长连接与处理意外断开。
5. 流量控制问题，虽然TCP提供来滑动窗口来做拥塞控制，但是确是基于路由器交换的，如果如果是上层应用的不当操作所造成的TCP数据压栈会影响其他进程的无法发送数据，所以要对数据发送的速度做限制，以避免应用层的数据大量发送，达到流量控制的目的。
6. 断线重连问题,TCP是一个长连接的协议，但是网络却不是永远都这么好，当网络发生错误导致TCP套接字断开时应用层应该考虑TCP的重新连接以达到用户友好的目的。
7. 通信加密问题，TCP采用的是明文传输，在数据安全性上未留下余地，导致重要信息被轻易获取，所以在互联网中传输数据时安全性不得不好好考虑一番。
8. 大小端问题，由于硬件的不统一，在内存的硬件存放上存在大端、小端两种不同的内存数据正反拜访顺序，它们的数据在内存中按字节端相反，如果没有对大小端进行处理则会读取到与预期不符的数据。
9. 多线程顺序问题，在客户端中发送消息通常只有单线程发送与接收，但在服务器上由于到同时处理多个sock连接的数据，所有都会提前生成线程池等待数据处理，而多线程在发送或者接收时可能会因为时间并发，如果读取的返回时间不一致会导致先读的后返回，从而导致获取的内容顺序错乱。
10. 串包问题，该问题是由原本发送请求的连接因为网络原因断开之后，其原本的sock被新连接客户端占用，而服务器接收到请求并进行反馈时却将反馈内容返送到了占用的sock上，导致断线重连的客户端未能接收到反馈包，而服务器错误的将包发送到被占用的sock上引发的错误。

### 网络开源库选择

C++语言是一个庞大且复杂的语言，实现各种特性与标准库，却迟迟没有在标准库中添加网络库，其原因是各领域对网络库的需求不一致，很难达到统一。尽管如此C++的能用的网络库却非常多，其原因是C++开发者喜欢造轮子。目前主流的网络库有libevent、libev、libuv、libhv、ACE、libcurl、boost::asio等重多稳定可靠的网络库，且实现各类通信协议以及封装跨平台接口。

本次选择的网络库是libhv网络开源库，该库实现了WebSocket协议，封装跨平台C++API，实现SSL数据传输安全，接口简单易用，性能很高，且开源代码可自定义。对于无特殊需求的服务器开发，上述任选一种也能达到需求。

## 图形框架

### 图形框架简介

开发桌面UI的图形框架有很多，其中C# 语言有 WinForm、WPF等开发框架，C++ 有Qt、MFC、WTL、wxWidgets等多个图形界面可供选择，Java 语言 有AWT、Swing两大框架，在图像界面框架上选择众多。三个编程语言中C#是最适合开发界面的，但对Linux系统跨平台的支持并不友好，故不做考虑。

### 图形框架选择

考虑到本项目需要跨平台开发，且自身对C++编程语言更加熟悉，所以选择了C++ and Qt的图形框架进行UI界面的开发。选择Qt的一个重要原因是，Qt提供了完善的API开发帮助文档，对于开发学习非常有帮助，且对跨平台支持非常友好。

# ==修改内容线==

# 研究思路和方案设计

## 项目简介

本项目是主要实现一个聊天软件，一般聊天软件提供的业务内容大概是文字聊天、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。而本次我想实现的是其中业务上一小部分的内容，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。

该项目只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，并搭建可在互联网通信的服务器。

## 基础网络协议简介与分析

既然要实现网络聊天功能，网络间的通信协议便是不可不谈的问题，在TCP/IP四层概念模型中TCP/IP协议族的UDP和TCP是传输层协议不可或缺的重量级选手。其中UDP提供了无连接、尽最大努力交付、面向报文、是快速且不安全功能，其特点就是快速，且是无连接无反馈的快速发送，但缺点也是无连接造成的保报文丢失等不可靠因素，需要应用层实现可靠通信。在网络环境良好的区域上，建立在可靠的UDP协议基础之上的应用协议将得到非常优秀交换体验，而当网络不稳定时，应用层实现的可靠和重发功能往往会将UDP的快速优势清扫的一干二净，速度比TCP还低数倍。当然在实时内容上，UDP一般会采用不完全可靠的实现方式，及加快来传输速度，也在大量报文丢失时跳过重传来保证视频的实时性，如直播视频等大流量实时画面。

UDP固然在速度上有很多优势，但现在的网络确实大量基于TCP实现，其原因就是在复杂的互联网环境中，报文十分容易丢失而很多内容都是不容忍丢失的，如文件传输等。TCP提供的面向连接、提供可靠交付、面向字节流、稳定且安全可靠协议解决了应用层的零丢失问题，所以TCP协议才会被大量使用。尽管TCP协议需要三次握手建立连接，四次挥手的操作，而且是面向字节流，在应用层开发上会存在粘包以及压栈等问题，不过凭借着在传输层的滑动窗口的拥塞控制、全双工的通信连接减少了应用层对于网络流量的管理，而且使用连续ARQ的重传机制也不是慢到那里去，由此看来TCP协议被广泛采用不无道理。

回到聊天软件的实现，TCP协议固然有很多好处，但是这个协议确实针对传输层开发，对应用层使用的是流传输，存在大量数据时的粘包问题，如果只是单点通信那这个问题也不会是问题，而用于服务器上则可能会存在堵塞线程的情况，给应用层的处理添加了不少麻烦。所以本次将使用WebSockect作为通信的网络连接协议。

## WebSockect协议简介

WebSockect协议是建立在TCP基础之上的全双工长连接协议，主要用于浏览器与服务器的通信。在介绍WebSockect协议时，应该先介绍一下HTTP协议。HTTP协议最开始是出现在浏览器上的一种基于TCP的应用层协议，该协议的主要任务就是用浏览器向数据服务器发送请求报文，服务器根据报发内容反馈内容并马上断开连接。所以HTTP协议的特点就是简单快速、灵活且无连接、无状态的一种短时TCP连接。其中无连接无状态指的是HTTP建立了请求/响应模型来对服务器进行通信，一旦本次内容提交并由服务器反馈完成则不保留如何连接记录，浏览器想保存信息就必须重新发送，而这就是浏览器会使用Cookies记录并重发的原因，而Cookies尽管只是保证了客户端的信息记录，对于服务器来说依旧存在忙点。相比于HTTP的短连接、快释放的方式，WebSockect协议则采用了长连接有状态的通行方式，使得完成一次请求任务之后客户端与服务器之间依旧保持连接，WebSockect的出现将改变HTTP由服务器下发通知到客户端时只能采用轮训的方式，而是直接与已连接的客户端通信，极大地简化了服务器对浏览器的任务下发，同时减轻了服务器的资源开销。为什么该项目会采用WebSockect协议作为通信媒介呢，这是因为WebSockect与TCP在应用层的使用极为相似，且保留了将来在浏览器实现聊天软件的一种可能。

## IO复用技术

本项目的服务器主要使用WebSockect协议通信，并部署在云服务器Linux系统的发行版ubuntu系统服务器版本上，与互联网连接并开放端口提供客户端连接。服务器所有用的网络编程技术是Linux操作系统提供IO复用技术epoll。

epoll是Linux内核为处理大批量文件描述符而作为select和poll的改进版本，它能显著提高程序在大量并发连接中只有少量活跃连接时的网络性能，主要用于解决网络IO编程中频繁操作系统提供的select/poll函数受文件描述符限制而无法满足高性能的情况。因为epoll中得到系统的事件支持，当连接无事件被记录在连接池中的sock并不会被加入到轮询容器中，只有触发了事件的连接才会进入轮询，避免了select/poll中无事件sock依旧进入轮询导致连接数增多会严重增加IO轮询时间，最后服务器因为轮询时间过长而导致新连接丢失或者任务超时的问题。

其中epoll接口提出的主要特点是，在一个进程中可以监听大量的文件描述符，而不会出现调用select或poll的最大描述符值的限制，一般select或poll的最大限制最佳数是1024个sock，而且epoll也支持高效地进行添加，修改和删除文件描述符的操作。

epoll的内部维护一个epoll的实例的文件描述符的变更信息并触发事件, epoll提供了水平触发模式和边缘触发模式，当数据量巨大时采用边缘触发提高IO性能，从而提供了更好的控制。当检测到文件描述符有数据变更时，epoll会以事件的形式通知应用程序。

IO复用在网络编程中非常重要，依靠单线程连接处理以及少量线程池就可以处理大量的高并发连接与数据交换，减少系统的线程开销就是变相提高系统资源的可用性。

## 数据库分析与使用

服务器上除了不可忽视的IO复用技术之外，数据库的使用也尤为重要。服务器上的数据库用于存放用户信息，而用户信息量会随着时间的推移而累加，虽然该项目的数据量相对较少，但如果采用文件的方式存放并频繁的对数据进行增删查改的话，结果在文件操作上必然是服务器性能的瓶颈，而数据库对数据得操作速度则避免了这个问题。该项目采用了SQList3的轻量级数据库，数据库拥有轻量级的，无服务器连接的轻量级可携带等优点。SQList3支持标准的SQL语言，采用无数据类型，所以可以保存任何类型的数据，支持标准数结构类型，而且是无配置的，只生成一个.db的数据库文件，将该文件带到哪里，哪里就是数据库。其便携性可以在多地、不同设备开发而不用考虑数据库数据迁移问题，是轻量级项目的最优选择。

## 网络开源库

建立通信连接时固然可以直接使用C库的sockect()函数API，不过在实际开发中都会采用网络库，或者重新编写一个网络库，绝不会单纯的使用sockect套接字，因为该函数是提供了最基础的数据交发送功能。其中网络编程中存在的几大问题，TCP粘包问题，非常常见，因为是TCP是面向流字节传输的；UDP的报文丢失问题，使用UDP是必须在应用层实现可靠的重发策略；TCP的长连接问题，虽然TCP底层已经存在保活设置，不过要是由于硬件设备的突然中断所导致的连接断开则不反馈，所以应有层需要实现自己的心跳来处理意外断开；流量控制问题，虽然TCP提供来滑动窗口来做拥塞控制，但是确是基于路由器交换的，如果如果是上层应用的不当操作所造成的TCP数据压栈会影响其他进程的无法发送数据，所以要对数据发送的速度做限制，以避免应用层的数据大量发送，达到流量控制的目的；断线重连问题,TCP是一个长连接的协议，但是网络却不是永远都这么好，当网络发生错误导致TCP套接字断开时应用层应该考虑TCP的重新连接以达到用户友好的目的；通信加密问题，TCP采用的是明文传输，在数据安全性上未留下余地，导致重要信息被轻易获取，所以在互联网中传输数据时安全性不得不好好考虑一番；大小端问题，由于硬件的不统一，在内存的硬件存放上存在大端、小端两种不同的内存数据正反拜访顺序，它们的数据在内存中按字节端相反，如果没有对大小端进行处理则会读取到与预期不符的数据；多线程顺序问题，在客户端中发送消息通常只有单线程发送与接收，但在服务器上由于到同时处理多个sock连接的数据，所有都会提前生成线程池等待数据处理，而多线程在发送或者接收时可能会因为时间并发，如果读取的返回时间不一致会导致先读的后返回，从而导致获取的内容顺序错乱。串包问题，这个问题是由原本发送请求的连接因为网络原因断开之后其原本的sock被新连接的数据占用，而服务器接收到请求并进行反馈时却将反馈内容返送到了新连接的sock上，导致断线重连的sock未能接收到反馈包，而服务器也错误的将包发送到其他sock的错误。

认识到上述网络编程可能存在的问题之后，选择一个合适网络库将可以避免错误发生的同时减少开发时间，减轻了网络编程的可靠负担。在几大热门libevent、libev、libuv、libhv网络库中，选择了libhv网络库作为本次项目的网络通信开发。libhv库是近几年才兴起的网络库，因为其简单易用，且提供C++接口；提供SSL/TLS的加密通信增加安全性；提供心跳、重连、线程安全等编程接口；提供装包、拆包模式；提供HTTP和WebSocket协议支持；提供Linux、Widows、MacOS、Android、iOS等常见操作系统的跨平台支持；提供IO复用事件循环等众多优点，使得在Github开源社区上拥有越来越多的使用者和维护者。

## 开发环境以及工具

随着Linux系统的桌面版本盛行，一度让Linux这个居身于服务器幕后的操作系统逐步成为了普通计算机使用者的其他操作系统选择，而不再是开发者的工具。本项目的开发环境在Linux发行版Ubuntu 22.02桌面版本。

客户端桌面开发工具使用了Qt的QtCreator编辑器，这个编辑器为开发Qt程序专用，对于Qt程序的开发十分便捷，且支持qmake + g++一键编译，可以快速编译、调试，十分便捷。

服务器开发则采用VSCode开发工具，因为该编辑器可以下载大量插件，扩展性极强，其中的Remote Development插件用于本地连接远程服务器，可以将实现ssh连接的远程开发。ssh可以同步远程的文件以及文件夹到本地，在本地修改之后又同步到远程服务器，而且编辑器提供的代码高亮与补全等功能在远程环境下依旧有效，可实现快速远程开发，对于需要远程开发的开发者来说VSCode十分便捷。

# 跨平台软件设计

## 跨平台问题

由于Linux桌面版操作系统开始发光，目前大多数软件都开始做跨平台兼容，让一份代码可以在多个操作系统上编译并运行，由于本次开发使用的是C++这种较为底层的编程语言，其语言本身并不像Java语法那样带有挂平台属性，所以需要开发者手动编写跨平台代码。跨平台代码主要分为两种，一种是操作系统API的跨平台，另一种是数据类型的跨平台操作。不过在本次开发中使用C语言API与Qt库可以解决大部分跨平台问题。

## 操作系统API跨平台

操作系统API跨平台代码演示：socket函数的跨平台编写

Linux下socket函数原型：

int socket(int domain, int type, int protocol);

win32下的socket函数原型：

SOCKET WSAAPI socket(int af, int type,int protocol );

定义一个跨平台的socket\_c函数，由该函数完成不同跨平台调用。

跨平台函数原型：

int socket\_c(int domain, int type, int protocol);

//==实现代码段==

int socket\_c(int domain, int type, int protocol)

{

#ifdef \_WIN32\_ //win32平台代码实现

SOCKET tm = socket(domain, type, protocol);

if(tm != INVALID\_SOCKET) return 1;

else return WSAGetLastError();

#endif

#ifdef \_LINUX\_ //Linux平台代码实现

return socket(domain, type, protocol);

#endif

}

//==实现代码段==

从代码中可以看出，win32和Linux的socket函数返回值并不一致，需要自己提供统一接口达到跨平台的目的。

## 数据长度跨平台

上述是操作系统API的跨平台设计，众所周知，不同平台，不同CPU位数的操作系统在基本数据类型的内容长度上也可能不一致，而数据长度的变化可能与与设计者的初心不符从而导致BUG的出现，所以跨平台设计需要将不同长度的数据类型做统一。

典型的长度不一致如Linux long类型8字节，win32 long 类型4字节，而long类型一般默认为8字节，所以win32下需要使用 long long 代替。其中，C语言提供来stdint.h库用于跨平台开发的固定长度类型，下面是使用示例。

类型长度跨平台处理：自定义 int64\_t 、 uint 64\_t类型，

#ifdef \_WIN32\_

#defien int64\_t long long

#defien uint64\_t unsigned long long

#endif

#ifdef \_LINUX\_

#defien int64\_t long

#defien uint64\_t unsigned long

#endif

声明变量统一使用int64\_t、uint64\_t。

int64\_t a; //声明64位整形

uint64\_t b; //声明64位无符号整形

# 分析软件需求

## 通信聊天软件的功能简介

现在市面上的即时通讯工具的基本功能一般都能会有：即时聊天、语音电话、发送个人/团体信息、附近用户利、添加好友、讨论组、群聊、即时语音、图片/视频、红包等多项的社交娱乐功能，本项目主要着重的研究点是通信协议、UI设计与网络架构方面，所以只实现其中部分关于信息传输、基本需求等主要内容，业务逻辑部分暂不讨论。

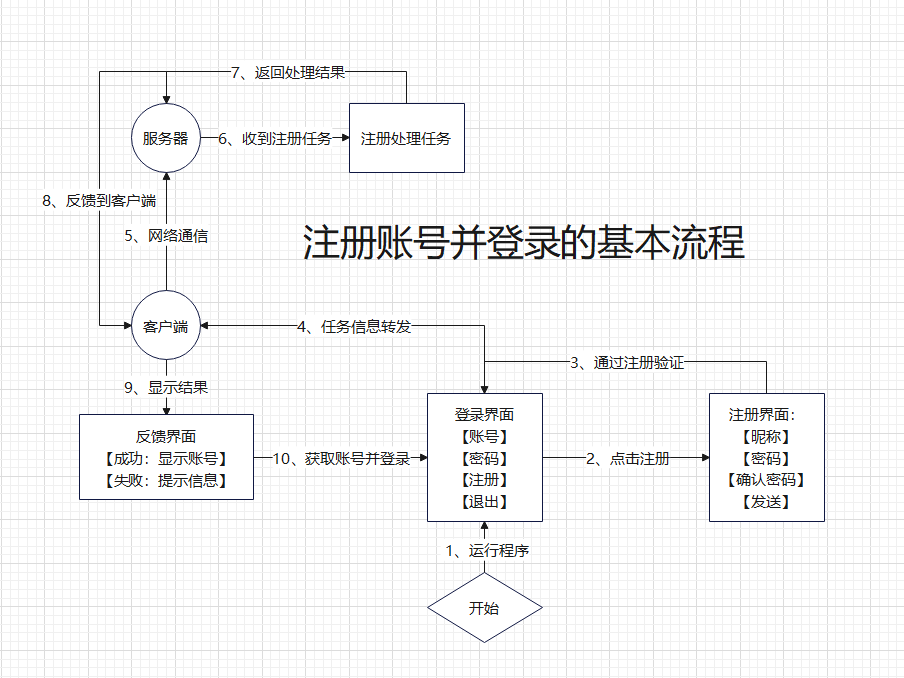
## 主流实现的基本通信功能

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，架构公网服务器，采用S/C的服务器、客户端网络架构。
    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。

# 客户端具体实现与基本流程

## 注册流程



图表 1 注册流程图

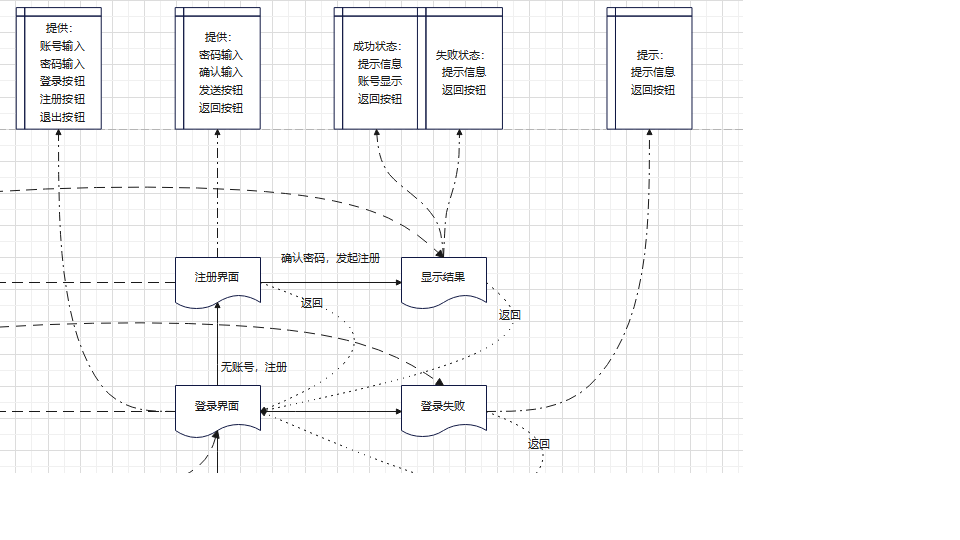
中流程图中看出从无账号状态，在软件中注册账号并成功登录到程序界面。可以看出注册界面与登录界面之间的关系，注册界面为登录界面的子类，注册界面的所有操作都会反馈到登录界面，再有登录界面反馈到管理类。管理类在初始化中已经建立与服务器了WebSockect的网络通信，最后注册界面的信息会由管理类发送到注册界面。

服务器在网络连接中接收到了客户端的信息并判断协议类型进行处理，处理结束之后反馈到客户端，此时再有客户端接收服务器的反馈信息，根据反馈的成功与失败在界面上做出相应的提示。在客户端的响应中，不管是成功或者失败最终都会重新回到登录界面上给用户。

## 登录状态图与控件图介绍



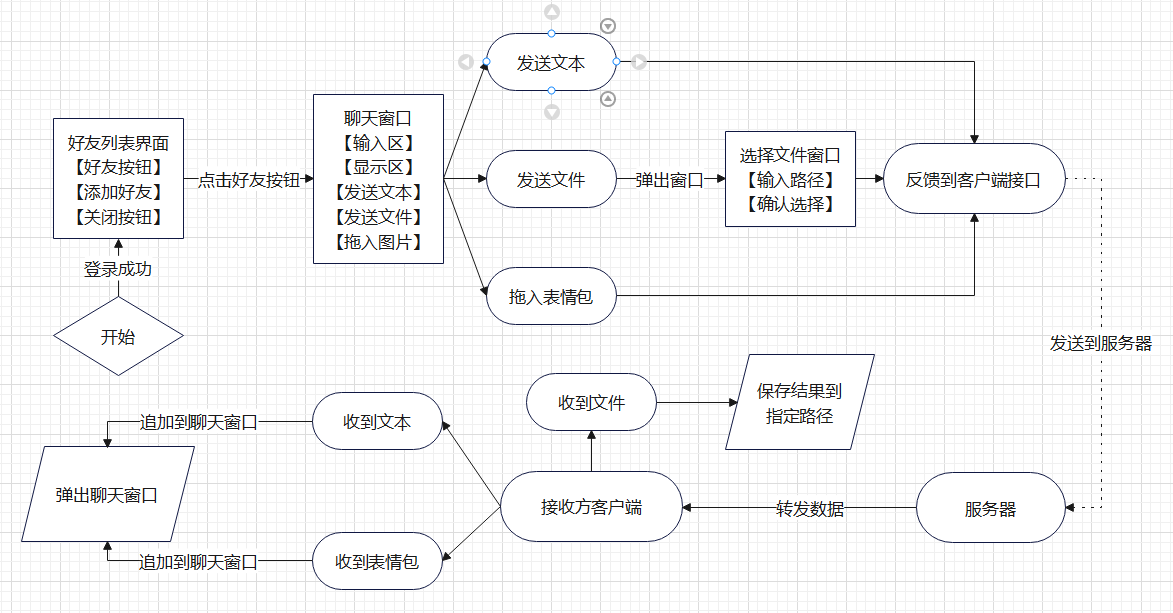
图表 2 UI切换状态状态图



图表 3 登录界面UI控件图

从状态图可以看到各种任务的成功失败与UI窗口的反馈关系，状态图简化了网络之间的通信连接，可以更加直观的看到不同操作下成功或者失败之后操作与对UI的反馈与切换，其状态图的反馈到成功登录并显示好友聊天窗口为止。关于在状态图中的各种UI显示细节在UI控件图中都有显示，其中列出了各个UI窗口输入与显示所使用的控件细节。

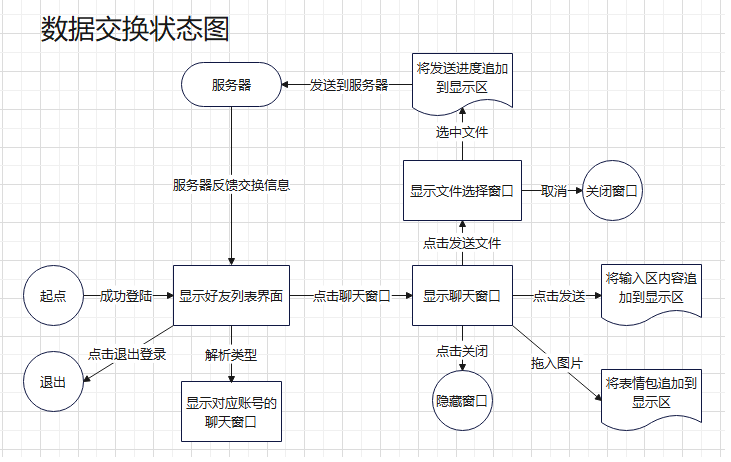
## 交换流程



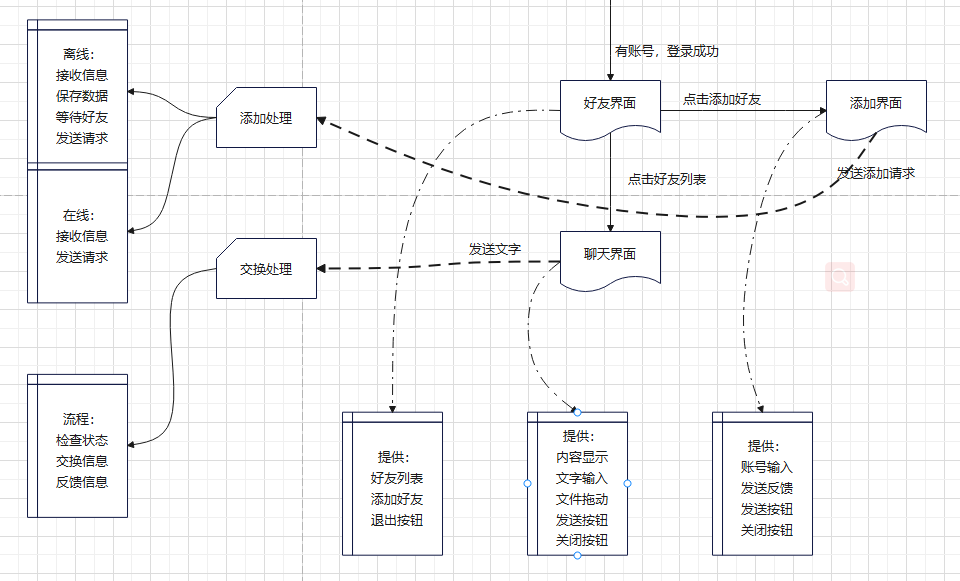
图表 4 交换流程图

在完成注册和登录流程之后，接下来就进入来正式开启聊天的过程。聊天的第一步就是要将自己的的信息成功的发送到另一个程序上来，而交换流程图就是试图去将这个过程以图形的方式呈现出来。从交换流程图中可以看出，登录成功之后的好友界面提供了好友按钮列表，该列表上拥有已经成功添加的好友，点击好友按钮就会弹出对应的聊天窗口。聊天窗口分别提供来三种输入方式，发送文本、拖入表情包和发送文件。其中拖入表情包是无需确认的，拖入即发送，发送文本是等待确认键按下后发送出已经输出在输入窗口的文字，而发送文件则会提供一个窗口让用户选择想要发送的文件，如确认要发送的文件之后，在显示区就可以看到发送文件的进度条信息。其中，不管是发送了哪一种类型的信息，其信息都会立刻追加到显示窗口，而接收方会在收到服务器的转发显示时解析出对应的信息类型并显示在他的聊天窗口上。如果没有任何动作的情况下发送聊天窗口突然弹出来了，不要惊讶，一定是有人给你发消息了。

## 交换状态图与处理图介绍：



图表 5 交换状态图

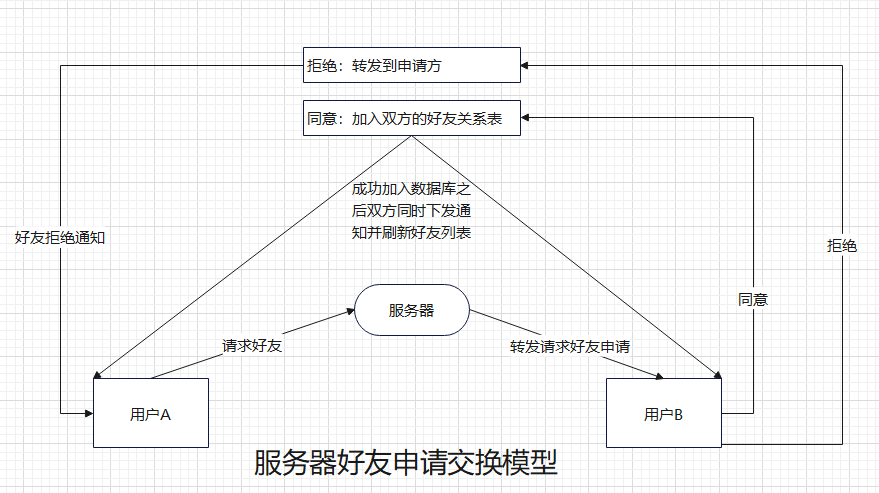


图表 6 交换处理图

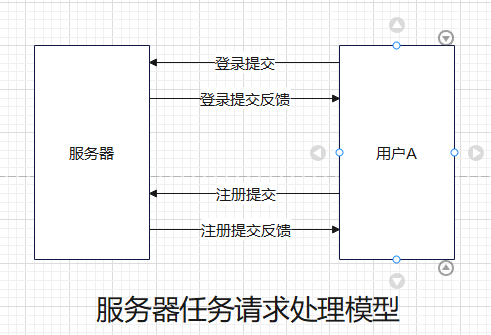
从状态图中可以清晰的看出交换数据时的各个UI控件的转换，在点击好友按键之后，会弹出对应好友的聊天窗口，而接收到服务器的交换信息之后也会弹出对应的好友聊天窗口。在接收方的交换信息的显示中，有一点需要注意，发送文本与发送表情包都是在全部数据接收完成之后一次性完成显示的，而发送文件则会将信息以进度条的方式持续显示，这是因为文件一般都比较大，短时间内无法完成交换，在本项目中并没有限制发送文件的大小。

在服务器的交换处理图中可以看出，因为不确定在发送信息时对方是否可以马上接受，所有设置了发送缓冲区确保了当好友不在线时依旧可以发送信息，而好友上线时去检索离线缓冲区内是否有发给他的信息，如果有则可以在上线时接收到信息。

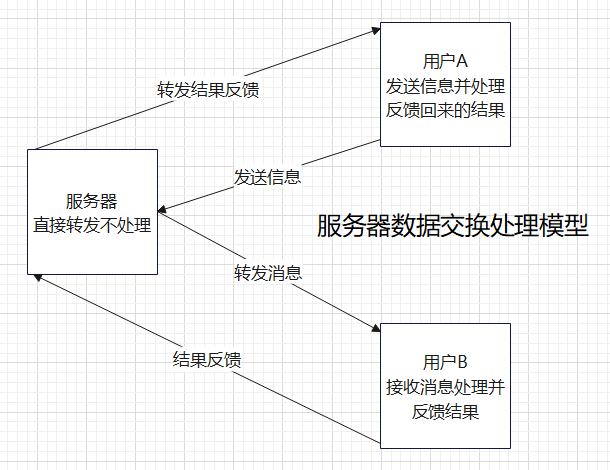
## 数据交换模型



图表 7 好友添加处理模型



图表 8 任务请求响应模型



图表 9 数据交换模型

好友添加模型在数据转发到另一个客户端时对数据进行了处理，既设计到对数据的处理也涉及到数据的转发，所以从服务器的三个交换模型中可以看出，好友添加的处理模式是较为复杂的。因为好友添加时涉及到来对另一个用户的请求答复，而答复的结果又会影响到服务器的行为，所以服务器在转发时做了更多处理。与数据交换模型相比，好友添加模型多了服务器对数据的处理，因为数据交换在服务器不需要处理任何数据，只需要直接查询转发即可，故而简单；与请求响应模型相比，请求响应模型只需要将接收到的数据进行处理并原路返回结果即可，不涉及其他用户客户端，故而简单。好友添加处理模型的设计其实是为了优化用户体验而单独设计的，因为在成功添加好友之后服务器需要同时向双方下发反馈结构，提示客户端重新拉去好友列表的信息，将新好友及时更新到好友列表中。添加好友处理模型并不是必须的，因为将数据交换模型与请求响应模型结合在一起使用也可以达到一样的目的，只不过需要在客户端做更多的处理，也因为多次交换而浪费更多网络资源，故而采用新的处理模型优化处理流程。

## 本章小结

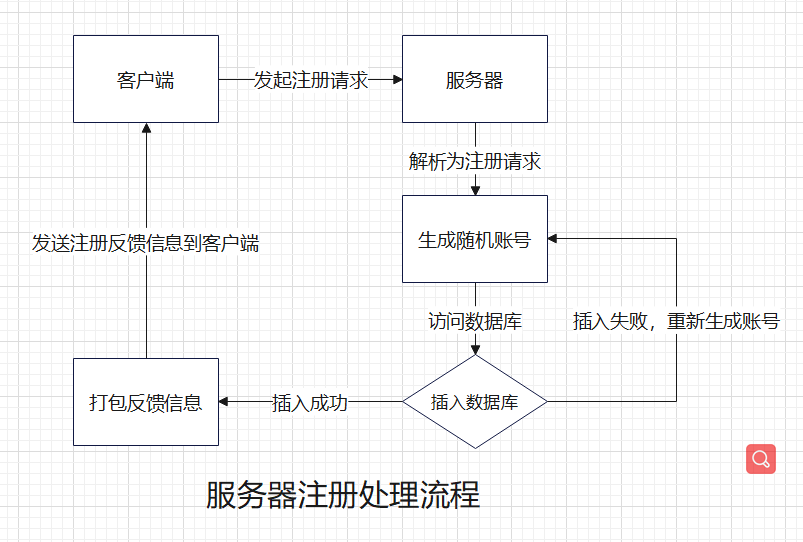
本章用图表的方式展示来客户端的登录流程、注册流程与数据交换流程，其中登录流程与注册流程中包含了UI界面切换的状态图与UI各个界面的详细控件布局，记录着每个窗口所提供给用户的操作与操作对应的触发流程。在数据交换简介的环节中还列出了任务请求模型、数据交换模型以及好友申请时经过特殊处理的交换模型，可以从模型中看出数据在客户端与服务器之间是如何进行交互，如何将处理任务与交换模型绑定在一起的。

# 服务器具体实现与基本流程

## 请求响应模式简介

请求响应模式是服务器一对一处理的简单模型，由客户端发送消息到服务器，服务器会解析出任务类型之后立刻进行处理，并将处理结构原路反馈到连接的套接字上。该处理方式因为需要原地处理，通常需要处理的任务都相对简单且不耗费太对CPU资源，否则将会造成服务器长时间被占用而无法接收到其他任务请。

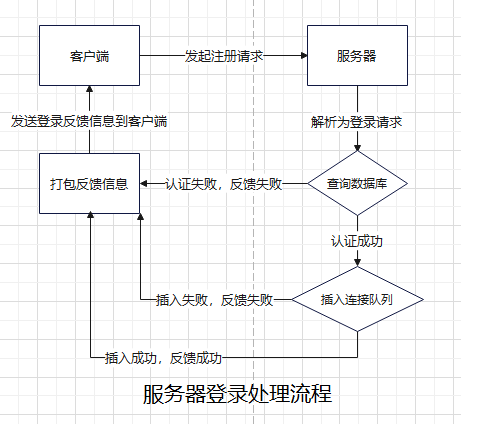
### 注册请求处理



图表 10 服务器注册处理

从图中可以看出，每当客户端进行注册时，服务器会生成随机账号并插入到数据库，如果可以重新插入数据库则表示当前已经成功生成出唯一账号，且当插入失败时会重新生成账号。在服务器的实际实现上，账号长度为十位长度的整数，随机生成的上限为十万次，如果十万次都无法成功插入数据库则表明此时的数据库几乎达到了满载状态，在这种状态下服务器会直接禁用注册功能以保证现有数据库的正常使用。

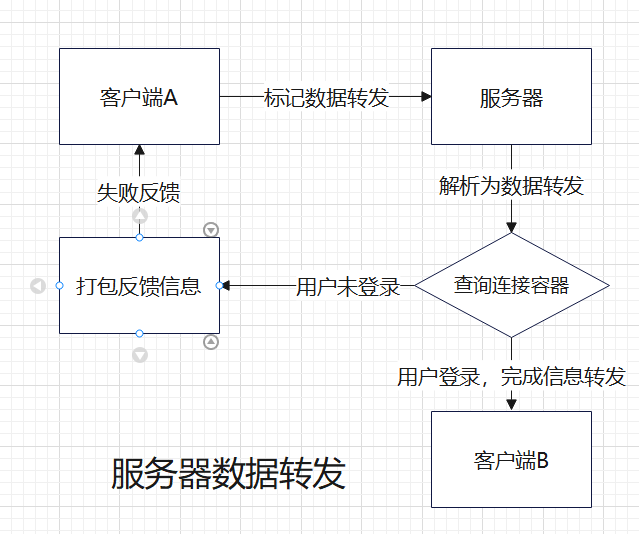
### 登录请求处理



图表 11 服务器登录处理

从登录流程可以看出，与注册流程不一致的地方是登录流程会进行双重处理，不仅要保证在数据库中查询到的登录信息正确，还要保证是否可以成功插入到连接队列。在服务器中维护着一个连接队列容器，该容器存储所有成功登录的用户，每当有用户成功登录，就存储该用户的账号信息与连接套接字。将账号与套接字绑定之后就可以通过账号索引到套接字从而进行数据转发与信息下发。

## 数据交换流程



图表 12 服务器转发处理

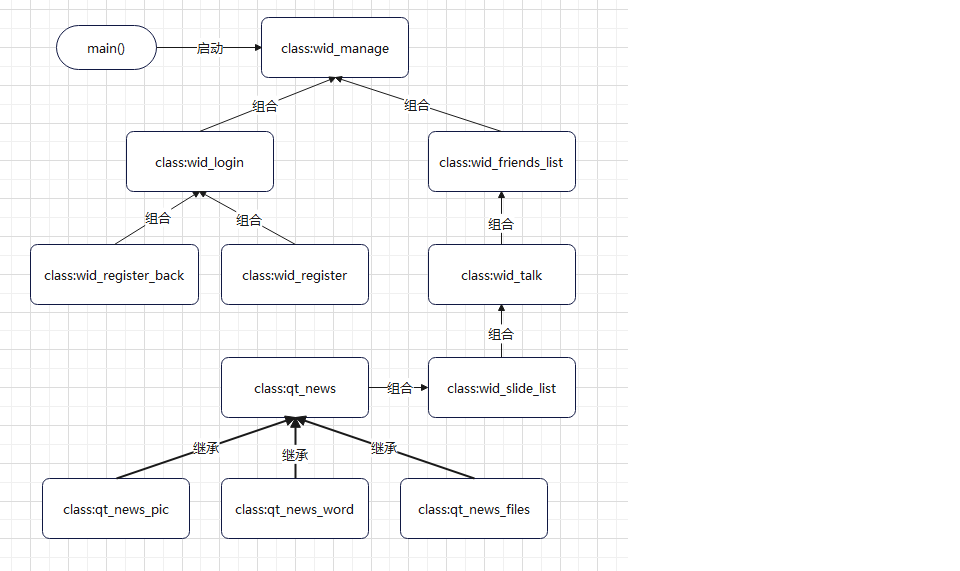
在服务器的信息转发处理上，服务器并不会做太多处理，只是将数据包的头部信息解析出来，在传输协议中分别设置了两种类型的类型，其中一种就是数据转发类型，该类型上附带了源账号与目标账号两个索引值，在到达目的之后方便对方的回信。在服务器成功解析出转发目标的账号后便将信息原封不动的全部发送到目标用户，所有的信息都由目标客户进行处理。如果查找到目标用户不存在则原路反馈错误信息到发起用户的套接字上。

## 本章小结

本章简单的介绍了服务器应对请求响应模型与数据交换请求模型两种不同的处理方式。服务器在处理登录、注册这类请求响应类型时会原地处理所有信息，所以这种信息通常不会占用太多的CPU时间，处理数据交换信息时只是简单的解析出目标地址后便将数据原文转发。其中，在请求响应模型中也详细的展示了注册、登录请求对应的服务器处理全步骤。

# 软件测试与效果展示

## 类关系图



图表 13 UI类关系图

在进入测试之前，先来了解以下类的关系可以更明白的理解测试结果。从关系图中可以看出，wid\_manage类组合两个子类wid\_login类与wid\_friends\_list类，wid\_login类负责登录之前的注册操作，而wid\_friends\_list类，负责登录之后创建聊天窗口。wid\_friends\_list类下的成员wid\_talk类负责将内容展示到界面，展示的内容又交给wid\_slide\_list类管理。其中wid\_slide\_list类是一个可滑动的列表窗口，将QWidget的子类加入列表并提供滚动的展示，其中类内存在一个容器记录来qt\_news类的指针，而且qt\_news类是虚类，不提供实例，wid\_slide\_list类管理的所有的内容都由继承自qt\_news的子类提供。

## 注册与登录界面展示

从图中得知，wid\_manage类的成员wid\_login类是一个图形界面类，提供客户端的登录界面。登录界面中会提供账号输入框、密码输入框、确认登录按钮、账号注册界面、退出登录按钮。



图表 14 wid\_login类

在无账号下wid\_login类提供账号注册wid\_register类与wid\_register\_back类，wid\_register类需要申请者提供密码，如果注册成功则由服务器返回一个一组被登记的账号信息，该信息由wid\_register\_back类展示给用于。



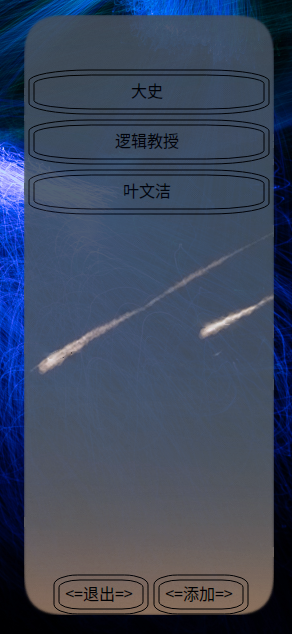
图表 15 wid\_register类



图表 16 wid\_register\_back类

注册类为登录类的成员，在点击注册之后wid\_register类将发送一个附带着昵称和密码的信号给wid\_login类，接收到信号之后再将该内容转发给服务器，服务器处理结束之后反馈信息到wid\_login类，在将反馈信息交个wid\_register\_back类完成整个注册流程。

## 聊天窗口展示



图表 17 wid\_friends\_list类

登录成功之后就进入wid\_friends\_list类的界面，该类记录本账户的所有好友，并提供一个好用列表按钮组。wid\_friends\_list类的按钮容器记录所有按钮以及对应的好友账号，点击按钮是会触发对应账号的wid\_talk类聊天窗口，同时在窗口下方放置了添加好友和退出登录的两个按钮。



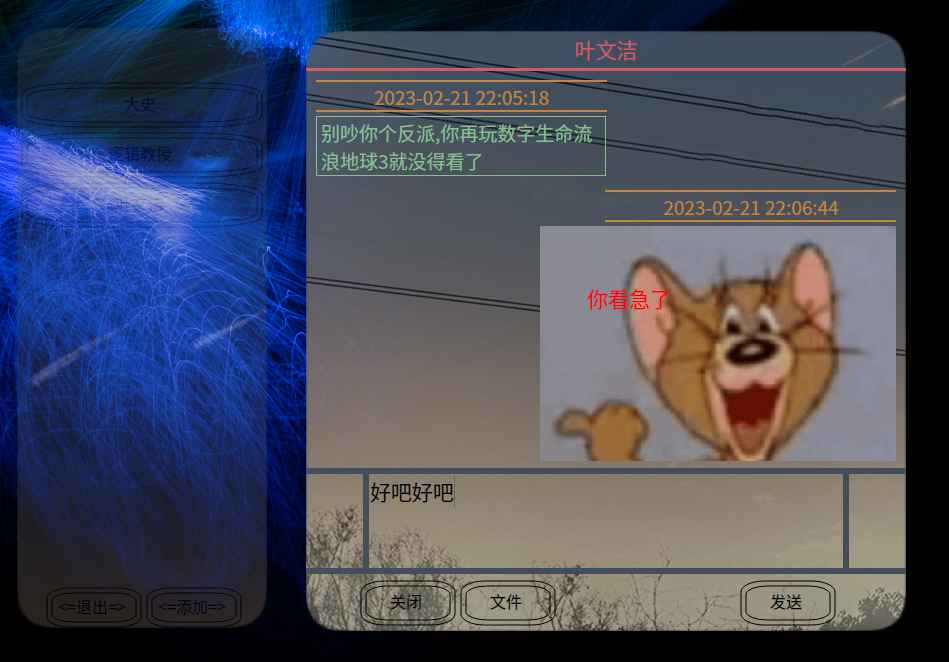
图表 wid\_talk类

wid\_talk类由两大部分组成，分别是输入区qt\_edit\_text类和显示区wid\_slide\_list类。qt\_edit\_text类由Qt控件QTextEdit接收键盘内容到文字缓冲区，QPushButton控件确定发送文字缓冲区到显示区。wid\_slide\_list类是一个长度可变但是只能通过固定大小窗口预览的内容显示窗口，该类由QScrollArea和QWidget两个控件组成，QScrollArea将QWidget设置成一个可滑动区域，当有内容不断写入显示区，QWidget控件长度被拉长时，即可达到窗口滑动预览的效果。

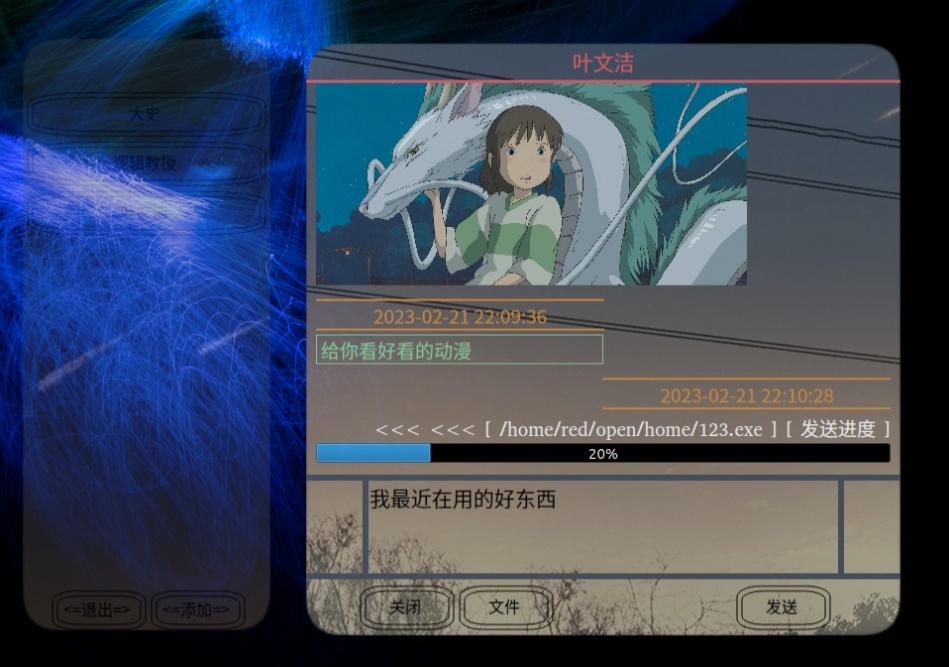
wid\_slide\_list类在内部有一个vector的容器成员，记录着所有加载到显示区的内容。当文字被加载到显示区时，被加载的内容是一个QWidget界面而非单纯的文字，所有被加载的控件都必须从qt\_news 类继承，并重写to\_string\_info纯虚函数以获取历史记录，显示文字的继承类为qt\_news\_word类。

显示区继承qt\_news 类的设计理念是因为显示区需要回收记录记录，而发送到显示区的不仅是文字还有图片、视频、图片、表情包、动态包、特效等多种多样的展示，不过目前仅提供来三种展示效果，分别是:qt\_news\_pic类显示表情包、qt\_news\_word类显示图片、qt\_news\_files类显示文件传输状态。如果局限于文字将无法获取良好的展示效果，而区展示的内容是QWidget类则完全不同，QWidget控件内可以随意填充各种想要显示的画面将带来非常大的灵活性，同时强制重写to\_string\_info转字符内容的纯虚函数为各类型的历史记录保存带来了统一的接口，在显示和聊天内容回放上都十分灵活。

## 发送文件与发送表情包



图表 18 发送表情包



图表 19 发送文件进度条

从上图中可以看出，发送类型为表情包时发送方可以将图片直接展示到显示窗口上，接收方在图片接收成功之后也会直接展示到显示区上。发送文件时双方会在文件传输之前建立连接，如果成功建立连接则双方会同时显示进度条的发送状态，传输完成之后会将文件保存到指定位置。在应用上虽然实现了两种不一样的显示方式，但是在实际数据传输上使用的依旧是同一套文件传输手段，达到代码复用的效果。因为无论是发送表情包还是发送文件其实都是对文件进行传输，只是应用层的显示不一致。

## 添加好友



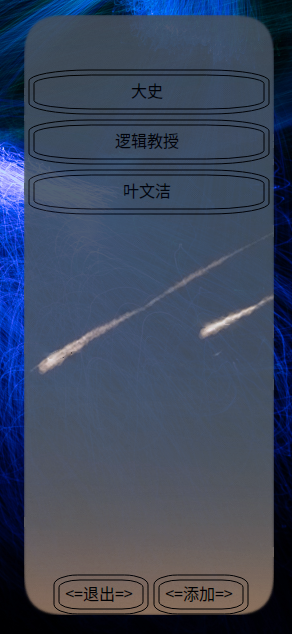
图表 20 好友添加



图表 21 添加好友账号



图表 22 成功添加



图表 23 刷新好友列表

上图为添加好友时的实际情况，当用户选择添加好友之后便会弹出窗口询问所添加好友的账号信息，发送好友申请之后可以等待好友确认，如果好友确认添加对方为好友则申请方会弹出提示窗口提示信息。因为好友申请使用的好友申请模型会在对方同意之后同时下达反馈，所以双方客户端会在确认之后同时向服务器重新拉去好友列表，此时便会将新添加的好友更新到界面，可以立刻开始新的聊天。

## 本章小结

本章简单的介绍了客户端UI类的关系以及实际展示了注册、登录、发送文本、发送文件、发送表情包以及添加好友等实际操作效果图。这些简单功能的完成可以实现简单的交流，加上特殊设计的UI界面已经让用户获得愉快的使用体验。在程序退出后重新登录还可以根据本地的聊天记录存储保存历史聊天功能，该程序已近完备了快捷社交的所需基础功能，历史记录的具体实现操作将在下文中提到。

# 软件设计的前置知识

## 内容概要

即将进入代码环节，原本的论文编写内容主要以介绍代码实现功能的以及实现细节为主，但考虑并不是所有的老师都会接触到编程语言，所以将大多数代码的实现画成了流程图或者状态图的形式展现方面理解，并将代码部分放置在最后环节，且简化了大量的内容与细节。 在进入正式的代码介绍之前首先放置了一些编程知识的前置内容以回答之后遇到的问题。

## 客户端开发前置知识：

该项目使用C++的Qt库配合QtCreator工具开发客户端的UI界面，Qt库虽然是C++中的一部分，编程语法上却不是标准C++的规范，而是在C++的语法之上扩展出来信号槽的概念，在拥有C++的基础之后请务必对Qt的信号槽机制有所了解才行。Qt的各类事件消息依靠信号发送信息，槽接受消息，信号与槽之间没有关联，依靠QObject类提供的connect信号槽连接函数将信号与槽之间建立起连接，否则两者完全独立。信号槽的使用可以有效的降低模块编程的耦合性，但信号槽在用法上由三点限制。第一，声明一个信号的类必须继承QObject基类；第二，类内必须声明Q\_OBJECT宏才可以使用信号槽机制；第三，编写的Qt信号槽类必须由Qt提供的Meta-Object Compiler元模板编译器先编译出带moc\_前缀的C++文件之后才能进入到正常的编译流程。这三点限制算是Qt的一个特点，在Qt中所有的Qt界面控件都继承自QObject类，由QObject类提供connect函数实现观察者模式，将子类的信号与槽进行连接，当信号发送时通知槽函数。Qt中大部分事件信息都是用类信号发出，接收时只需定义槽函数与控件类信号连接并将功能在槽函数中实现即可。Qt的所有界面类都是继承于QObject，而这些类的事件函数绝大多数都是虚函数，这使得由子类重写事件函数并重新实现功能成为了可能。这是Qt带来的高度可定制化的一套模式，不管是如何一个界面类控件都可以重新继承并重写，从而实现定制化的UI展示效果。

## 服务器开发前置知识

本项目的服务器主要使用WebSockect协议通信，并部署在云服务器Linux系统的发行版ubuntu系统服务器版本上，与互联网连接并开放端口提供客户端连接。服务器所有用的网络编程技术是Linux操作系统提供IO复用技术epoll。

epoll是Linux内核为处理大批量文件描述符而作为select和poll的改进版本，它能显著提高程序在大量并发连接中只有少量活跃连接时的网络性能，主要用于解决网络IO编程中频繁操作系统提供的select/poll函数受文件描述符限制而无法满足高性能的情况。因为epoll中得到系统的事件支持，当连接无事件被记录在连接池中的sock并不会被加入到轮询容器中，只有触发了事件的连接才会进入轮询，避免了select/poll中无事件sock依旧进入轮询导致连接数增多会严重增加IO轮询时间，最后服务器因为轮询时间过长而导致新连接丢失或者任务超时的问题。

其中epoll接口提出的主要特点是，在一个进程中可以监听大量的文件描述符，而不会出现调用select或poll的最大描述符值的限制，一般select或poll的最大限制最佳数是1024个sock，而且epoll也支持高效地进行添加，修改和删除文件描述符的操作。

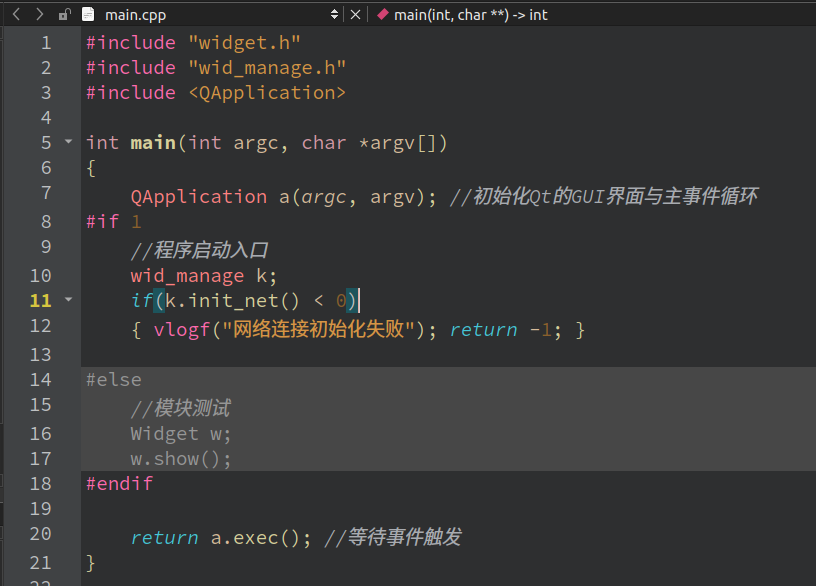
epoll的内部维护一个epoll的实例的文件描述符的变更信息并触发事件, epoll提供了水平触发模式和边缘触发模式，当数据量巨大时采用边缘触发提高IO性能，从而提供了更好的控制。当检测到文件描述符有数据变更时，epoll会以事件的形式通知应用程序。

IO复用在网络编程中非常重要，依靠单线程连接处理以及少量线程池就可以处理大量的高并发连接与数据交换，减少系统的线程开销就是变相提高系统资源的可用性。

# 编程代码实现与细节：

在通信篇中会介绍整个程序是如何将信息层层传递出去的，因为在实际的开发中，将一条信息传递出去并不会和想象中的简单，因为要考虑到整个系统的模块化。模块化系统可以降低整个系统的复杂度，可以解耦不同模块之间的关闭，使得其他模块无需考虑别人在做什么，只需要做好自己的事情便可以由全局控制的管理类掌控整个程序的流程。在该项目上也会尽量的使用这种编程方式，不过因为整个项目的体量不够大，有时模块化编程会给人造成一种多余的感觉，但这些多余大体量的项目和多人联合的实际开发来说是非常有用的，该项目给出了一种解决方案。在这一篇开始请先具备基础的编程知识，因为从这一篇开始会以程序员的视角出发，进入到实际的编程环境当中。

## 程序入口



图表 24 代码入口

QApplication是Qt应用程序框架的第一个步骤，是一个应用程序对象，会启动应用程序的主循环，处理和转发所有事件，在任何Qt的GUI类时必须先创建一个QApplication对象，而且该对象也必须且只能创建一次。 QApplication对象会完成程序的初始化，它从程序的命令行参数获取应用的设置和窗口系统设置，并且在运行时用于Qt与窗口系统之间进行交互。

一旦初始化QApplication类之后，由该类的exec()函数等待时间循环，这个函数会堵塞进程并等待事件的触发。要注意的是，Qt中所有的GUI类都必须在QApplication对象之后，exec()函数之前完成初始化，否则将不会被Qt的主事件循环包裹，会失去原本希望实现的功能。

从图片中可以看出，在QApplication类对象与exec()函数之间我们声明了一个wid\_manage的管理类，以及调用来init\_net()函数用于初始化网络连接。

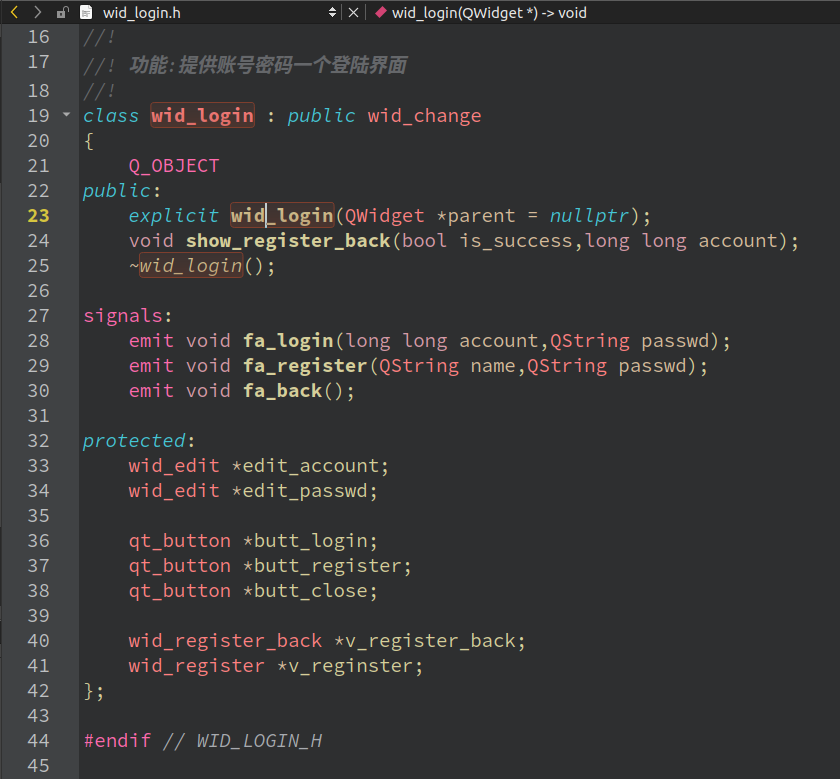
图表 管理类声明



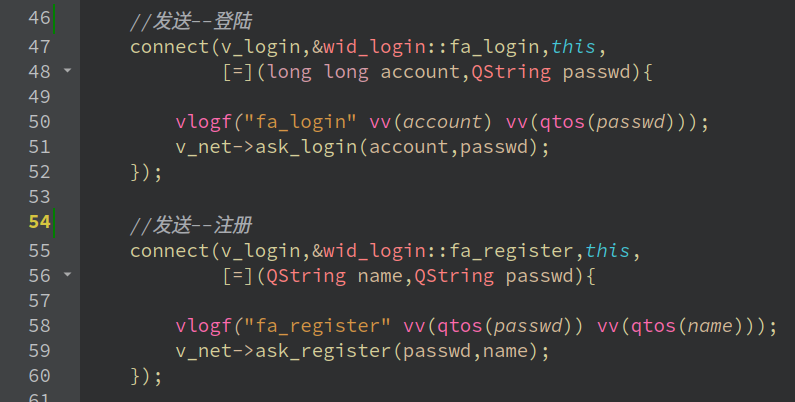
图表 25 管理类声明

在wid\_manage类中，声明了两个窗口和客户端的网络接口，其中wid\_login类是用于处理登录之前的所有操作，而wid\_friends\_list类是处理登录成功之后的所有操作，这样的设计使得登录成功与否都与两个类没有管理，只有wid\_manage类负责调度，而他们只需要做好自己的事情，这样的操作可以将业务逻辑与功能实现进行解耦。net\_connect类则负责将信息转发到服务器，该类是网络通信的唯一接口，所有的网络通信都必须由该类提供接口方可传输信息，这样的设计是为了将本地的操作与网络连接二者分离，从而达到解耦的目的。在本地的所有操作都无需关心信息是如何传输出去的，只需要向net\_connect类提供所传输的信息即可，net\_connect起到了类似管道的作用，它将所有的网络操作都封装在内部，使得任何内容的传输都像在本地一样，它只会反馈失败与成功。

## 信号转发



图表 26 登录窗口代码



图表 27 转发代码

从图中可以看出，在wid\_login类注册了三个信号，其中的两个分别是登录信号与注册信号，而这两个信号会在wid\_manage类进项槽函数的绑定，而这两个槽函数都分别调用了net\_connect类的网络接口将信息发送出去。从这一片的代码段得知，wid\_login类无需关心注册信号发出后会怎么样，而wid\_manage类也无需关心将信息原封不动的传递给net\_connect类的提供网络接口之后接下来会怎么样。它们三者之间只是负责信息的传递，而无需关系具体的实现，net\_connect类对于wid\_login类来说是透明的，它们不知道彼此的存在，只需要由wid\_manage类将两者绑定在一起。

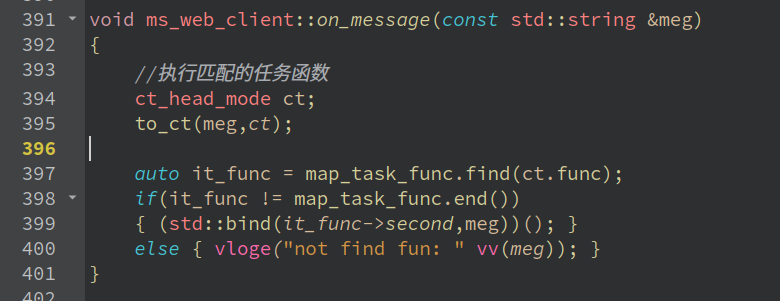
## 网络通信

### 客户端



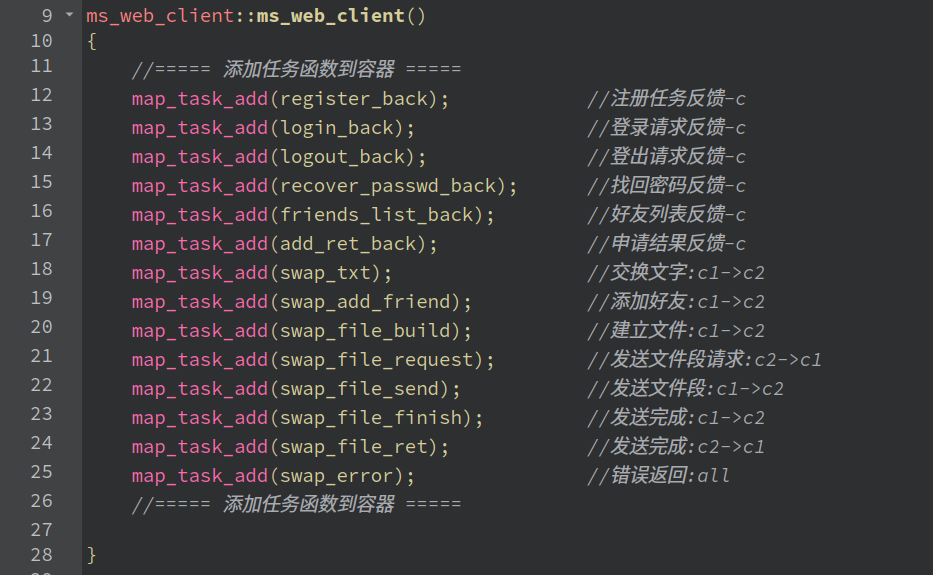
图表 28 网络接口-客户端

从上图可以看出，客户端依赖一个网络接口类，其中类由三个回调函数，分别是on\_open()回调，连接新接入时会触发该回调，说明此时客户端已经与服务器建立连接；on\_message()回调，当函数触发是说明接收到了服务器发来的消息，此时参数meg的内容就是服务器发送过来的消息。



图表 29 on\_message回调

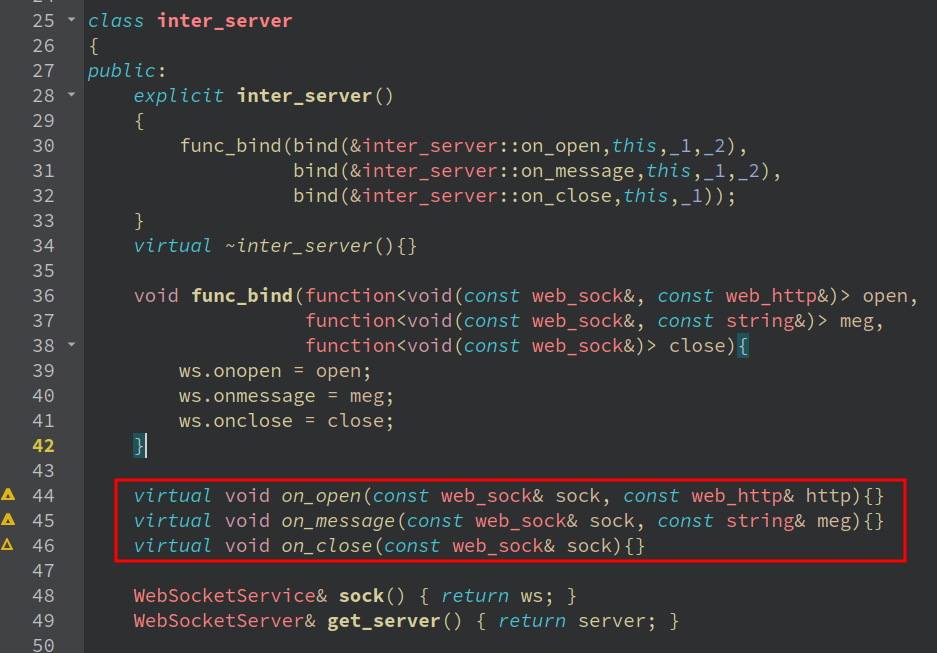
on\_message回调函数负责接收消息，通过判断不同的消息类型去调用不同任务函数，不过从上图中可以看出on\_message回调函数写的非常简单，这是一个良好的设计方式。通常来讲，on\_message回调函数的普通写法是通过一大堆的if-else语句去判断任务类型，每一种类型对应一个if判断，同时每新增一个任务函数都需要对该写一个if判断。这里的on\_message回调函数采用了map索引法，将每种任务类型与对应的任务函数绑定在一起，当索引到任务类型时，此时map的value就是对应的任务函数，此时执行任务函数即可完成类型任务类型与对应任务函数的判断过程。



图表 30 任务类型与任务函数绑定

从上图中可以看出，在构造函数将任务函数与任务类型绑定在一起，即可在触发on\_message回调函数时通过任务类型索引到对应的任务函数，完成类似if-else的选择判断。

### 服务端

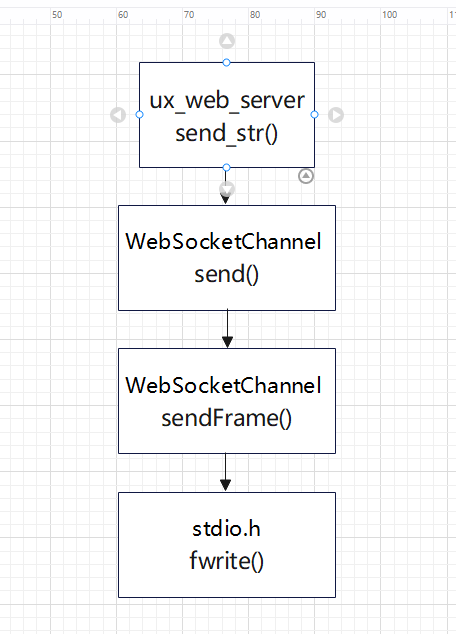


图表 31 网络接口-服务器

上图中，服务器与客户端的网络接口封装十分类型，不同的是，由于服务器会有多个客户端连接，每一次触发的消息都不知道是哪一个客户端，所以每一个函数接口都有一个sock参数，这个sock参数中携带着客户端的附加信息，其中包括IP地址、端口等。



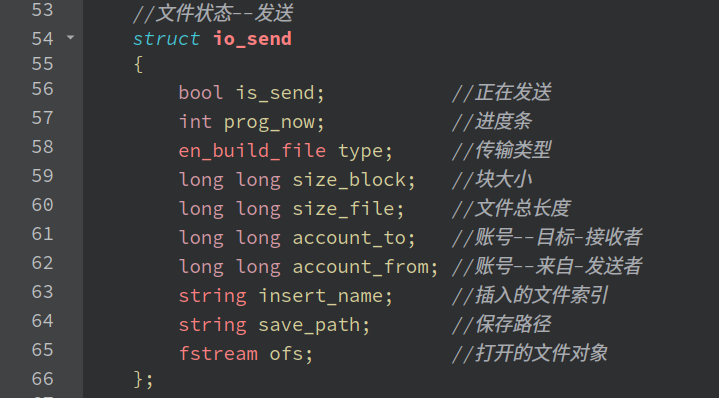
图表 32 发送接口



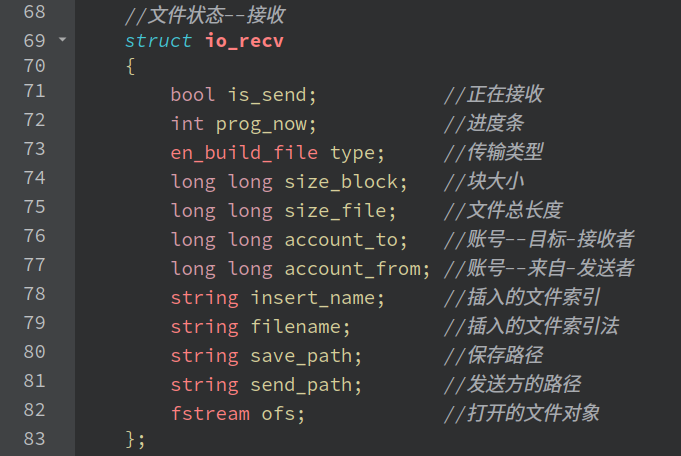
图表 33 调用函数流程

从上图可以看出，发送一段BUF的调用流程会反复的包装与传递之后才会通过C库，在由调用系统拷贝到TCP栈缓冲内等待发出。这些封装往往是必要的，因为TCP是字节流传输，但是应用层的协议往往需要控制格式并解析出对应结构、多线程发送数据时对缓冲区的内容同步、发送缓冲溢出时建立重发机制等多重保障，而每一层需求都需要建立一层发送函数的封装。

## 文件传输协议



图表 34 文件发送结构体



图表 35 文件接收结构体

上图为文件传输协议的结构体，在网络中传输文件并不能像TCP一样点对点的发送与接收，因为服务器是一对多，且网络环境并不稳定，必须考虑到读写的堵塞问题，发送进度反馈，多文件同时发送，同时发送与接收，文件暂停与继续等问题。

字段的功能解析：

|  |  |
| --- | --- |
| Bool is\_send | 判断是否为发送状态，如果false则准备停止 |
| Int prog\_now | 用于客户端显示百分比的发送进度 |
| En\_build\_file type | 文件显示类型，针对类型做不同的显示处理 |
| Long long size\_block | 文件块，用于中断传输恢复时的起点 |
| Long long size\_file | 记录文件总长度 |
| Long long account\_to | 发送的目标账号，用于服务器转发 |
| Long long account\_from | 发送者的账号，用于服务器转发 |
| String insert\_name | 给接收者用于判断文件，与发送者一致 |
| String filename | 发送的文件名，保存时文件时可能不一致 |
| String save\_path | 接收者的文件路径，本地保留不发送 |
| String send\_path | 发送者的文件路径，本地保留不发送 |
| Fstream ofs | 打开文件的IO流对象 |

通过以上字段的控制，可以让发送文件时做到多文件同时发送与接收，通过将文件分割为固定大小的块可以在传输中断时，恢复到块标记进行双方的字节同步。如果是记录发送或者接收的字节数进行记录并恢复传输，往往会出现文件损坏的情况，因为读取并写入文件时中断传输，数据可能还停留在内存上而不是写入磁盘，此时的读写字节数并非是写入磁盘的字节数，实际的存储字节于不等于API返回字节。如果想避免这种情况，必须给定一个提前量，在中断传输之前一大段的位置重置传输点，以保证数据已经实际的写入到磁盘中，这样的处理方式则可以最大可能的保证恢复传输完成时不会出现文件损坏。

## 本章小结

本章介绍了从main()函数开始的整个程序流程，以及Qt界面程序与普通的C++程序在编写时的不同点，以及Qt信号槽机制在C++语言上的语法扩展与使用，利用信号槽机制将数据从多线程数据收发下反馈到界面主线程。介绍了在网络数据传输中，如何对数据进行收发，以及应用层数据传输时如何定义简单并可靠的传输协议。

# 参考文献

1. Stanley B. LippmanBarbara E. Moo Josée. C++primer 第三版[M]. 广东:中国电力出版社.2002
2. 谢希仁. 计算机网络[第7版] [M]. 北京:电子工业出版社.2017
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
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
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
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.
7. 侯捷. STL源码剖析[M]. 湖北:华中科技大学出版社.2002.
8. Matthew H. Austern. 泛型编程与STL[M]. 河北:中国电力出版社.2003.
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.
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.
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.
13. Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides.设计模式：可复用面向对象软件的基础[M]. 北京:机械工业出版社.2000.
14. 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,2012:23-40.
15. 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.
16. 邓亚君.基于Python的网络聊天室设计[J].西华大学,2018:1-4.

# 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。