江 西 理 工 大 学

本 科 毕 业 设 计（论文）

题　　目：十万级分布式 IM 系统设计与开发

学　　院：信息工程学院

专　　业：网络工程专业

班　　级：网络工程 152 班

学　　号：1520153366

学　　生：袁小龙

指导教师：温卫　　职称：讲师

时间：　　2019 年　5 月　28　日

江 西 理 工 大 学

本 科 毕 业 设 计（论文）任 务 书

学院 信息工程学院　专业 网络工程 15 级（届）152 班 学号 15 学生 袁小龙

题　目：十万级分布式 IM 系统设计与开发

原始依据(包括设计（论文）的工作基础、研究条件、应用环境、工作目的等)：

现今市面上的各类 IM 工具虽然种类繁多，功能强大，但是大都只局限于移动端，windows PC 端或 mac PC 端平台。相对来说 Linux PC 的用户较少，在 Linux PC 端上的 IM 工具发展的相当缓慢，因此研究和开发 linux PC 下的 IM 系统工具有重要意义。本文旨在设计开发一个 Linux PC 下的高并发，高可用，低时延，易扩展，支持跨局域网通信的 IM 系统。

该系统提供的服务主要有注册账号，删除账号，用户登录，用户登出，查看用户列表，用户通信。

主要内容和要求：（包括设计（研究）内容、主要指标与技术参数，并根据课题性质对学生提出具体要求）：

主要内容：主要分为总体架构设计，私有协议设计，route server 设计，work server 设计和 client 设计五部分。其中，总体架构设计主要包括传输层协议选择，系统架构选择，服务器集群负载均衡以及系统各个模块之间的交互关系（机器级别）；私有协议设计主要包括确定报文结构和行为；route server 设计主要包括路由规则确定，进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 route server 中各个模块之间的交互关系；work server 设计主要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 work server 中各个模块之间的交互关系；client 设计主要要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 client 中各个模块之间的交互关系。

主要要求：在基础知识方面需要了解 c++的基础语法（c++STL 容器以及 c++中的线程操作 API，进程操作 API，系统信号处理 API，socket 网络编程 API 和 IO 复用相关 API），mysql 数据库，mycat 分布式数据库中间件，mysql++框架；需要熟悉进程间通信和线程间通信，统一事件源系统信号处理；需要掌握内网穿透技术，I/O 复用技术，负载均衡技术。

日程安排：

第 1－2 周：系统调研和需求分析。

第 3－5 周：收集资料、开发工具的学习使用。

第 6－7 周：数据结构分析和数据库设计

第 8－9 周：系统规划和设计

第 10－11 周：系统实现编程、系统测试。

第 12－13 周：整理文档及撰写毕业设计论文。

第 14 周：毕业设计论文答辩。

主要参考文献和书目：

1、《数据库系统概论》萨师煊，王珊 等，高等教育出版社，

2、《软件工程导论》张海藩 等，清华大学出版社，

3、《内存泄漏检测方法研究综述》汪明晔 等，电脑知识与技术，

4、《探究 c++编程中常见问题与解决对策》洪建 等，电脑知识与技术，

5、《浅析 Visual C++编程技巧》李丽薇 等，黑龙江科技信息

指导教师（签字）：

年　　月　　日

注：本表可自主延伸，各专业根据需要调整。

江 西 理 工 大 学

本 科 毕 业 设 计（论文）开 题 报 告

学院 信息工程学院　专业 网络工程 15 级（届）152 班 学号 15 学生 袁小龙

题　目：十万级分布式 IM 系统设计与开发

本课题来源及研究现状：

1、课题来源

1996 年 11 月全球首个 IM 工具 ICQ 甫一问世便受到广大网民的喜爱，风靡全球。而后，各类 IM 工具更是如雨后春笋一般不断冒出。目前国内最流行的 IM 工具是 QQ 和 WeChat，它们极大的方便了人与人之间的信息交流，同时这两款 IM 工具在国内成为了网络世界不可或缺的一部分。现今市面上的各类 IM 工具虽然种类繁多，功能强大，但是大都只局限于移动端，windows PC 端或 mac PC 端平台。相对来说 Linux PC 的用户较少，在 Linux PC 端上的 IM 工具发展的相当缓慢，因此研究和开发 linux PC 下的 IM 系统工具有重要意义。

2、研究现状

对网络上现有的 Linux PC 下的 IM 工具进行初步调研，可以发现它们大都是学生们的闲暇之作。在设计和实现上，这些 IM 工具大都千篇一律的使用的 C/S 模型加上数据库服务的三层架构，其中 S 为单台功能服务器，该服务器中使用单进程多线程模型；数据库服务中单纯的使用的 mysql 数据库服务。通信机制使用的服务器转发机制或单纯的点对点机制。由于设计和技术上的不成熟，导致这些 IM 工具大都存在这样或那样的缺陷。如 server 端使用的单台功能服务器会影响并发度和服务可用性；在功能服务器中使用单进程多线程模型会影响多核机器的使用效率，同时由于线程间切换时间片会提高服务时延；在数据库服务中单纯地使用 mysql 服务，会导致用户量达到一定程度后候数据库服务的时延非常高，影响 IM 系统的并发度和性能；使用服务器转发通信机制会加大服务器的负载，影响 IM 系统的性能；使用单纯的点对点通信机制会导致客户端之间无法跨局域网通信。

课题研究目标、内容、方法和手段：

1、目的：

设计开发一个 Linux PC 下的高并发，高可用，低时延，易扩展，支持跨局域网通信的 IM 系统。

2、内容：

主要分为总体架构设计，私有协议设计，route server 设计，work server 设计和 client 设计五部分。其中，总体架构设计主要包括传输层协议选择，系统架构选择,服务器集群负载均衡以及系统各个模块之间的交互关系(机器级别)；私有协议设计主要包括确定报文结构和行为；route server 设计主要包括路由规则确定,进程/线程模型选择,进程/线程间通信方式选择,I/O 处理模型确定，确定 route server 中各个模块之间的交互关系；work server 设计主要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 work server 中各个模块之间的交互关系；client 设计主要要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 client 中各个模块之间的交互关系。

3、方法与手段：

使用 C/S 模型加上数据库服务，其中 S 是一个两层的负载均衡结构，分为一台路由服务器和多台功能服务器，总体来看这是一个四层架构模型。Server 端的两层负载均衡结构能很大程度上的提高服务可用性和并发度,同时这是一个非常容易扩展的模型,加入功能服务器时只要简单的将功能服务器接入到路由服务器即可。另外，路由服务器和功能服务器中都使用的多进程单线程模型，可以提高多核机器的使用效率,提高服务性能,降低时延;数据库服务使用 mycat+mysql，mycat 支持读写分离，分表分库等功能，在用户量达到一定程度后能很大程度上降低数据库服务时延，提高数据库服务性能；使用点对点通信机制，同时在客户端上使用内网穿透技术,使得在不增加服务端负载的情况下支持跨局域网通信等等。

设计（论文）提纲及进度安排：

1、提纲：

1）绪言

论文研究背景及国内外现状，分析现有的 IM 系统缺点，提出解决方法。

2）技术要点概述

简略介绍一下文中使用到的一些数据库以及软件工程上的关键技术。

3）需求分析

对系统进行功能，接口，性能，数据等方面进行分析，确定系统需要提供哪些功能，需要制定哪些接口，在性能上要达到什么标准，确定业务流程图，数据流图和数据字典。

4）系统设计

总体架构设计，私有协议设计，route server 设计，work server 设计和 client 设计。其中，总体架构设计主要包括传输层协议选择，系统架构选择，服务器集群负载均衡以及系统各个模块之间的交互关系（机器

级别）；私有协议设计主要包括确定报文结构和行为；route server 设计主要包括路由规则确定，进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 route server 中各个模块之间的交互关系（进程级别）；work server 设计主要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 work server 中各个模块之间的交互关系（进程级别）；client 设计主要要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 client 中各个模块之间的交互关系（线程级别）。

5）系统详细设计与实现

确定 route server，work server 和 client 的详细流程图，并使用 c++ 语言实现各个功能模块。

6）系统测试

对已经实现的各个功能模块进行测试并编写测试用例。

2、安排：

第 1－2 周：系统调研和需求分析。

第 3－5 周：收集资料、开发工具的学习使用。

第 6－7 周：数据结构分析和数据库设计

第 8－9 周：系统规划和设计

第 10－11 周：系统实现编程、系统测试。

第 12－13 周：整理文档及撰写毕业设计论文。

主要参考文献和书目：

1、《数据库系统概论》萨师煊，王珊 等，高等教育出版社，

2、《软件工程导论》张海藩 等，清华大学出版社，

3、《内存泄漏检测方法研究综述》汪明晔 等，电脑知识与技术，

4、《探究 c++编程中常见问题与解决对策》洪建 等，电脑知识与技术，

5、《浅析 Visual C++编程技巧》李丽薇 等，黑龙江科技信息

指导教师审核意见：

指导教师（签字）：　　年　　月　　日

注：本表可自主延伸

摘　要

IM（即时通讯）是现今互联网上最受欢迎的通讯方式。各种 IM 软件更是泛滥成灾，其中的佼佼者如 QQ，微信更是功能繁多，资讯丰富，形成了独立的生态圈。但是这些 IM 软件大都只能在移动端，windows pc 端，mac pc 端工作。

本文旨在设计开发一个 linux pc 端下的 IM 系统。本文中运用到的计算机基础知识主要有 c/c++编程语言，socket 网络编程，epoll I/O 复用，进程间通信，线程间通信，linux 系统信号处理等。对于架构，本系统使用 C/S 模型，且在服务器端使用集群负载均衡技术，在单台服务器中使用多进程单线程模型，在客户端上使用多线程模型，在存储方面使用分布式数据库。该系统提供的服务主要有注册账号，删除账号，用户登录，用户登出，查看用户列表，用户通信等。

相对 linux 端下的同类型产品，由于使用了 tcp 内网穿透，分布式数据库，集群负载均衡等技术，本系统的优点是跨局域网通信，操作时延低，服务高可用，且在服务器数量足够的情况下能够支持十万以上的用户同时在线。

关键词：IM 系统；C/S 架构；负载均衡；分布式；网络编程

ABSTRACT

Instant messaging (IM) is the most popular mode of communication on the Internet nowadays. Various IM software are flooding. Among them, the outstanding ones such as QQ and Wechat have many functions and abundant information, forming an independent ecosystem. But most of these IM software can only work on mobile, Windows PC and mac PC.

The purpose of this paper is to design and develop an IM system on Linux PC. The basic computer knowledge used in this paper mainly includes c/c++ programming language, socket network programming, epoll I/O multiplexing, inter-process communication, inter-thread communication, Linux system signal processing, etc. For the architecture, the system uses C/S model, and uses cluster load balancing technology on the server side, in a single server. Using multi-process single-threaded model, using multi-threaded model on the client, using distributed database in storage. The services provided by the system mainly include registration account, deletion account, user login, user logout, viewing user list, user communication and so on.

Compared with the similar products on linux, the advantages of this system are inter-LAN communication, low operation delay, high availability of services, and can support more than 100,000 users online at the same time when the number of servers is sufficient, due to the use of technologies such as TCP intranet penetration, distributed database, cluster load balancing and so on.

Key words: IM system; C/S architecture; load balancing; distributed; network programming

目录

第一章　序言

1996 年 11 月全球首个 IM 工具 ICQ 甫一问世便受到广大网民的喜爱，风靡全球。而后，各类 IM 工具更是如雨后春笋一般不断冒出。目前国内最流行的 IM 工具是 QQ 和 WeChat，它们极大的方便了人与人之间的信息交流，同时这两款 IM 工具在国内成为了网络世界不可或缺的一部分。现今市面上的各类 IM 工具虽然种类繁多，功能强大，但是大都只局限于移动端，windows PC 端或 mac PC 端平台。相对来说 Linux PC 的用户较少，在 Linux PC 端上的 IM 工具发展的相当缓慢，因此研究和开发 linux PC 下的 IM 系统工具有重要意义。

对网络上现有的 Linux PC 下的 IM 工具进行初步调研，可以发现它们大都是学生们的闲暇之作。在设计和实现上，这些 IM 工具大都千篇一律的使用的 C/S 模型加上数据库服务的三层架构，其中 S 为单台功能服务器，该服务器中使用单进程多线程模型；数据库服务中单纯的使用的 mysql 数据库服务。通信机制使用的服务器转发机制或单纯的点对点机制。由于设计和技术上的不成熟，导致这些 IM 工具大都存在这样或那样的缺陷。如 server 端使用的单台功能服务器会影响并发度和服务可用性；在功能服务器中使用单进程多线程模型会影响多核机器的使用效率，同时由于线程间切换时间片会提高服务时延；在数据库服务中单纯地使用 mysql 服务，会导致用户量达到一定程度后候数据库服务的时延非常高，影响 IM 系统的并发度和性能；使用服务器转发通信机制会加大服务器的负载，影响 IM 系统的性能；使用单纯的点对点通信机制会导致客户端之间无法跨局域网通信等等。

本文旨在设计开发一个 Linux PC 下的高并发，高可用，低时延，易扩展，支持跨局域网通信的 IM 系统。在设计和实现上，本系统使用 C/S 模型加上数据库服务，其中 S 是一个两层的负载均衡结构，分为一台路由服务器和多台功能服务器，总体来看这是一个四层架构模型。Server 端的两层负载均衡结构能很大程度上的提高服务可用性和并发度，同时这是一个非常容易扩展的模型，加入功能服务器时只要简单的将功能服务器接入到路由服务器即可。另外，路由服务器和功能服务器中都使用的多进程单线程模型，可以提高多核机器的使用效率，提高服务性能，降低时延；数据库服务使用 mycat+mysql，mycat 支持读写分离，分表分库等功能，在用户量达到一定程度后能很大程度上降低数据库服务时延，提高数据库服务性能；使用点对点通信机制，同时在客户端上使用内网穿透技术，使得在不增加服务端负载的情况下支持跨局域网通信等等。

第二章　技术要点概述

本章主要简略介绍一下本文中使用到的一些数据库以及软件工程上的关键技术。

2.1 数据库相关

2.1.1 mysql

MySQL 是一非常高效的关联数据库管理系统，它用元组记录关系数据，元组存储在数据表中，一个数据库可以有多个数据表。MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。在存储容量上，MySQL 支持千万条记录级别的数据仓库。在性能上 MySQL 支持 MyISAM 存储引擎，MyISAM 数据库与磁盘非常地兼容而不占用过多的 CPU 和内存；同时 MySQL 还支持 InnoDB 存储引擎，InnoDB 在内存足够的情况下几乎相当于一个内存型数据库，具有非常高的运行效率；另外，由于 MySQL 中还设计了查询缓存机制，优化器等模块，使其在一般情况下也具有非常不错的操作数度。在扩展性上，遵循开源协议的 MySQL 拥有大量的辅助插件，可以轻松的完成分库，分表，数据同步等等操作。

2.1.2 Mycat

Mycat 是一个开源的分布式数据库中间件，是一个介于数据库与应用之间，进行数据处理与交互的中间服务。对数据进行分片处理之后，从原有的一个库，被切分为多个分片数据库，所有的分片数据库集群构成了整个完整的数据库存储。用户可以把它看做是一个数据库代理，用 MySQL 客户端工具和命令行访问，其核心功能是分库分表，即将一个大表水平分割为 N 个小表，存储在后端 MySQL 服务器里或者其他数据库里。Mycat 具有非常丰富的功能，它遵循 mysql 原生协议，支持读写分离、mysql 主从，支持 sql 黑名单、sql 注入攻击拦截，支持全局序列号，支持分布式下的主键生成，支持单库 join、跨库多表 join，实现了高效的多表 join 查询等。在管理上，Mycat 集群基于 ZooKeeper 管理，支持在线升级、扩容、智能优化。

在合适的场景下能极大的增加数据库的存储数据量和减少数据库操作时间。同时也降低了数据库集群的管理难度。

2.3 软件工程及相关知识

2.3.1 c++语言

c++在 c 语言的基础上对其进行了基本继承和部分改进。 c ++既可以进行过程化程序设计，又可以进行面向对象的程序设计。另外，c++语言还具备 c 语言的高效运行效率。c++适合于对性能要求较高的程序编写。

为了完成本论文，需要了解 c++的基础语法，c++STL 容器以及 c++中的线程操作 API，进程操作 API，系统信号处理 API，socket 网络编程 API 和 IO 复用相关 API。

2.3.2 mysql++

mysql++是对原始的 mysql c 语言 API 的一个封装，为 c++开发者提供类似于 c++STL 容器一样方便的操作数据库的一组接口。另外，mysql++还实现了 mysql 连接池，只要在其基础上进行二次开发即可以实现自己的连接池。mysql++的连接池类为 mysqlpp::ConnectionPool，可以从该类中继承一个类作为自己的连接池类，只需要在派生类中实现 mysqlpp::ConnectionPool 中的纯虚函数方法以及派生类独有的方法即可。

2.3.3 进程间通信和线程间通信

1. 进程间通信方式

本文中主要用到了管道（socketpair），信号量，共享内存，socket 等进程间通信方式。socketpair 是 socket api 提供的一个全双工管道，由于父进程中打开的文件描述符在子进程中也是打开的这一特性，socketpair 可以用于有亲缘关系的进程间通信；信号量本身不能传输复杂消息，只能用于数据同步，在实际使用中通常使用信号量建立临界保护区，配合共享内存一起使用；共享内存是速度最快的进行间通信方式，但是由于可能会产生多个进程同时操作一个内存区的情况，需要进行数据同步；socket 主要分为 internet domain socket 和 unix domain socket，前者是通过端口进行通信的，后者是通过本地共享文件进行通信的。本地进程间通信时可以选择后者，而网际进程间通信则只能选择前者。

2. 线程间通信方式

线程间的通信方式主要有全局变量方式和消息机制。进程中的所有线程都可以访问所有的全局变量，因此可以通过全局变量来进行线程间通信；另外也可以通过 Message 的 PostMessage 和 PostThreadMessage 接口进行线程间通信，即消息机制方式。

2.3.4 系统信号处理

系统信号即系统发送给目标进程的信息，用来通知目标进程某个状态的改变或系统异常。在 Linux 系统中有很多种系统信号以及一大堆的系统信号处理

API。在此处只要了解 Linux 系统中的少数几个系统信号以及统一事件源相关的信号处理 API 即可。

1. 系统信号

SIGCHLD	子进程状态发生变化（退出或暂停信号）
SIGTERM	终止进程信号 kill
SIGINT	键盘输入 ctrl + c
SIGPIPE	往读端被关闭的管道或 socket 写数据

2. 统一事件源

信号是一种异步事件，信号处理函数和程序的主循环是两条不同的执行路线。为了避免一些竞态条件，信号在处理期间系统不会再次触发它，因此信号处理函数需要尽可能快的执行完毕以确保该信号不会被屏蔽太久。解决方案是：将信号的主要处理逻辑放到程序的主循环中，当信号处理函数被触发时它只是简单地通知主循环接受信号，并把信号值通过管道传递给主循环。同时，设置 I/O 复用监听主循环的管道读端。这样，信号事件就可以和其他 I/O 事件一样被处理，此即统一事件源。

2.3.5 socket 网络编程

socket 的本质是对 tcp/ip 协议族的封装。它提供了一组用于网络编程的 api，网络开发者通过这些 api 进行网络通信。在 socket tcp 通信中，分为主动发起连接的客户端和被动接收连接的服务器，两者在工作流程上是不一样的。在客户端上，首先是创建 socket 套接字，然后通过 connect 系统调用向目标服务器发起连接，建立连接成功后即可接收/发送消息；由于需要指定服务端口，在服务器上，在创建 socket 套接字后需要先进行 bind 命名操作，接着才能通过 listen 系统调用监听客户端的连接，发现客户端内核队列中有客户连接到来时可以通过 accept 系统调用接受该连接，然后才能接收/发送消息。socket 通信流程见图 2.1。

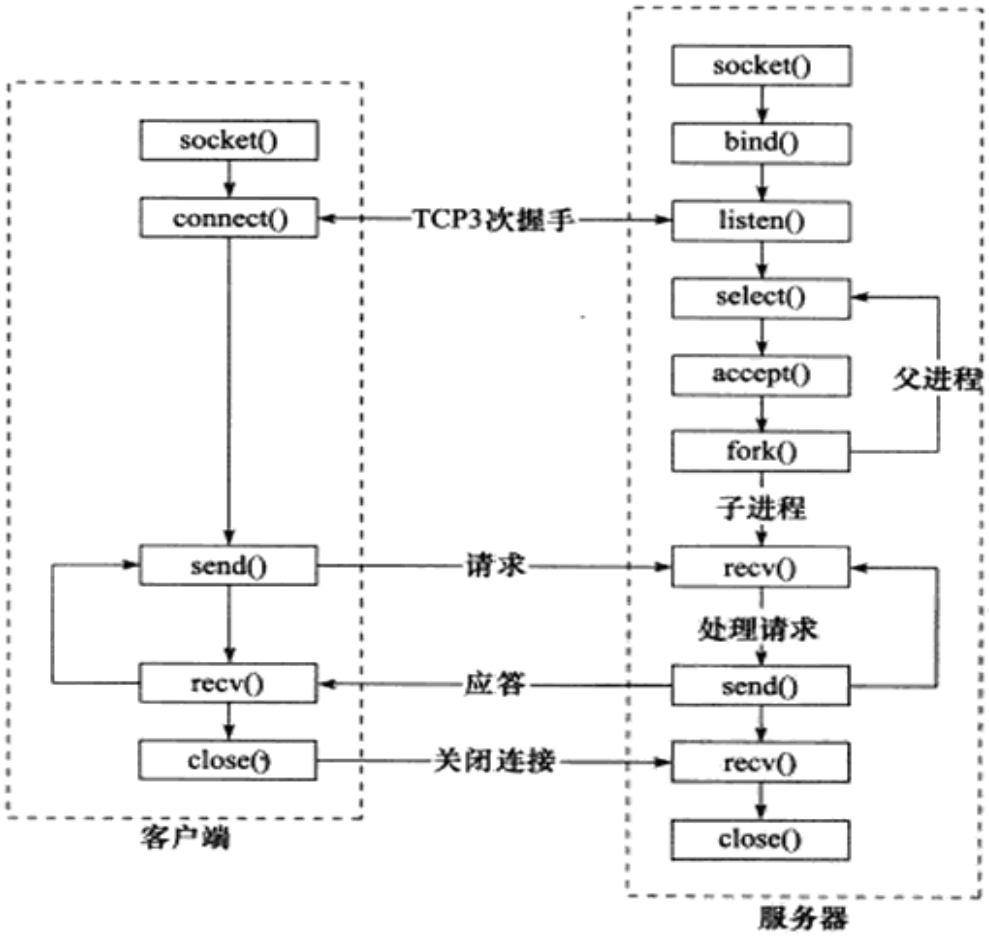

图 2.1 socket tcp 通信机制

2.3.6 I/O 复用

I/O 复用技术使得程序能同时监听多个文件描述符，在一些场景中能很大程度上的提高程序的性能。如客户端需要同时处理多个 socket，客户端要同时处理多个 socket，服务器要同时处理监听 socket 和连接 socket 等。Linux 下实现 I/O 复用的系统调用主要有 select、poll 和 epoll。因为开发过程中主要使用的 epoll，所以此处只了解 epoll 即可。

epoll 对文件描述符的操作有两种模式：lt（电频触发）模式和 et（边沿触发）模式。lt 是默认的工作模式，这种模式下 epoll 相当于一个效率较高的 poll。当往 epoll 内核事件表中注册一个文件描述符上的 EPOLLET 事件时，epoll 将以 et 模式来操作该文件描述符。 et 模式是 epoll 的高效工作模式。对于采用 lt 工作模式的文件描述符，当 epoll_wait 检测到其上有事件发生并将此事件通知应用程序后， 应用程序可以不立即处理该事件。 当应用程序下一次调用 epoll_wait 时，epoll_wait 还会再次响应应用程序通告此事件，直到该事件被处理。而对于采用 et 工作模式的文件描述符，当

epoll_wait 检测到其上有事件发生并将此事件通知应用程序后，应用程序必须立即处理该事件，因为后续的 epoll_wait 调用将不再通知该事件。 显然 et 模式在很大程度上降低了同一个 epoll 事件被重复触发的次数，因此效率要比 lt 模式高。

2.3.7 TCP 内网穿透

考虑到通信的双方可能处于两个不同的局域网内，由于内网机器访问外网内容时需要将私有网络地址映射到一个公有网络地址上，在不知道彼此映射后的公网地址时是无法直接通信的。需要做内网穿透。

tcp 内网穿透即，在 client 上使用端口复用将 listen 和 connect 系统调用绑定同一个端口，client 在初始化时先和一个具有公网 ip 的第三方 server 建立 tcp 连接，如此一来，具有公网 ip 的 server 中记录了所有 client NAPT 转换后的公网地址。clientA 在向 clientB 发起通信时先从 server 获取 clientB 的 NAPT 转换后的公网地址，然后再向该地址发起 tcp 连接请求。又因为 clientB 中 listen 和 connect 系统调用绑定的同一个端口，它们 NAPT 转换后的公网地址也是一致的，所以 clientA 和 clientB 是能够正常通信的。另外，该方式只在锥形网络中有效，因为只有在锥形网络中所有的公网－私网会话中会维持一个固定的端口映射。

第三章　需求分析

3.1 需求分析的目的和任务

对系统进行功能，接口，性能，数据等方面进行分析，确定系统需要提供哪些功能，需要制定哪些接口，在性能上要达到什么标准，确定业务流程图，数据流图和数据字典。

3.2 综合需求

3.2.1 功能需求

1. 注册账号：获取一个新的系统账号，该账号由数字组成且具有唯一性。每个账号对 应一个用户，一个用户可以注册多个账号；
2. 删除账号：从系统中删除一个已存在的账号，删除后该账号将失效；
3. 用户登录：用户登录即通过用户账号和密码的对应关系来验证用户身份的合法性， 以确保某些操作的安全性；
4. 分页查看用户列表：获取系统中某些用户的账号，用户名，是否在线等信息。考虑到一次获取太多用户信息的话会导致操作时延过长，信息太多查看不方便等问题，采用分页的方式，默认页面容量为 20 个用户；
5. 查看指定用户是否在线：只有都登录了的用户双方才能进行通信，查看指定用户是否在线即查看指定的用户是否已经登录；
6. 向指定用户发送消息：用户输入一个 user id 向目标用户 user id 发起会话，如果成功则进入会话状态，接下来输入的消息都将发送给目标用户 user id。如果发起 会话失败，则返回相关错误信息；
7. 退出聊天状态：当用户处于会话状态时，输入′ quit′ 断开该会话，用户单向退出会话 状态；
8. 退出登录：当用户处于登录状态时，输入退出登录关键字，该用户退出登录状态；
9. 用户维活：对于一个处于登录状态的用户，它需要向 IM 系统说明它处于登录状态。一旦该用户退出登录状态，IM 系统需要立即知道该消息；
10. 异常提示：当上述功能出现异常时，用户需要收到异常提示信息。

3.2.2 接口需求

1. 注册账号接口：用户输入用户名，密码信息，从 IM 系统中注册一个新账号，系统返回新账号的 user id；
2. 删除账号接口：用户输入用户账号和密码，从 IM 系统中删除一个已经注册的账号，删除成功返回′ 1′ ，失败则返回′ 0′ ；

3. 用户登录接口：用户输入用户名，密码信息在 IM 系统中进行登录操作即更改该用户的在线状态，登录成功返回′1′，失败则返回′0′；
4. 分页查看用户列表接口：用户输入 page（页数）值，number（每页的用户数量），在 IM 系统中查询表中第 page 页所有用户的 user id，user name，是否在线等信息（默认每页的用户数目不超过 20）；
5. 查看指定用户是否在线接口：用户输入一个 user id，从 IM 系统中查询该用户当前是否在线，若在线返回′1′，不在线返回′0′；
6. 向指定用户发送消息接口：用户输入一个 user id 向 user id 发起会话，如果成功则进入会话状态，接下来输入的消息都将发送给目标用户 user id。如果发起会话失败，则返回相关错误信息。

3.2.3 性能需求

1. 制定高效的私有协议；
2. 高可用，容错率高：当 IM 系统中部分服务器异常时，仍然能提供正常服务。即便只剩下一台服务器正常工作，也能提供正常的服务；
3. 支持大量用户同时在线：达到单机支持一万用户同时在线，拥有十台以上服务器时能支持十万以上用户同时在线；
4. 低时迟：保证在网络正常的情况下，用户之间发送/接收消息的平均时延不超过 5s，其他 IM 系统提供的服务平均时延不超过 10s；

3.2.4 数据需求

该 IM 系统的主要功能是用户管理，信息查询和用户之间发起会话。涉及到的基本信息有用户 id，用户名，用户密码，用户是否在线，用户登录的地址。

用户管理主要分为四个方面:注册账号，删除账号，用户登录，退出登录。注册账号时需要记录用户注册信息（用于用户的后续操作）。用户注册信息包括系统用户号，用户名，用户密码，当前用户是否在线，当前用户登录的地址（ip:port）。其中用户名和用户密码由用户输入，其他信息由系统初始化；删除账号需要用户输入用户号和密码信息，删除账号操作即删除对应的用户信息表元组；用户登录需要输入用户名和密码进行验证。判定用户成功登录后需要修改当前用户是否在线信息，修改当前用户登录地址信息；退出登录操作需要用户已经处于登录状态，退出登录后修改该用户是否在线信息，修改该用户登录地址信息。

信息查询主要分为两个方面:获取用户列表，判断指定用户是否在线。获取用户列表操作需要执行该操作的用户已经处于登录状态。执行获取用户列表操作需要用户输入 page（第几页），number（每页用户容量）信息，执行操作后返回第 page 页用户列表，包含每个用户的用户号和用户名称信息；判断指定用

户是否在线操作需要执行该操作的用户已经处于登录状态。用户输入要判断是否在线的 user id，如果用户号为 user id 的用户在线则返回 1，否则返回 0。

向指定用户发起会话需要发起会话的用户已经处于登录状态，发起会话的用户输入会话的目标 user id，系统会判断用户号为 user id 的用户是否已经登录，如果该用户没有登录则发起会话失败，如果该用户已经登录了，则系统会返回该用户的登录地址（ip:port），接着发起会话的用户会向目标用户发起 tcp 连接，进入会话状态。

结合前面的功能需求分析，接口需求分析以及数据需求分析，用户和系统中数据流向的来龙去脉也就一清二楚了。从用户流向 IM 系统的数据有用户管理请求携带的用户信息，查询请求信息和会话请求信息。从用户流向另一个用户的数据只有用户之间的会话信息。从 IM 系统流向用户的数据有用户列表信息，目标用户在线状态，会话目标用户登录地址和系统异常信息。另外，IM 系统还需要从系统时钟获取当前日期。其数据流图见图 3.1。

在前面的数据分析中将系统的功能分成了用户管理，信息查询和用户之间发起会话三类。据此可对图 3.1 更进一步细化，将其中的 IM 系统部分细分为处理用户管理请求，处理查询请求和处理会话请求三部分。具体见图 3.2。

图 3.2 中的加工 1 （处理用户管理请求）包括处理用户注册请求，处理用户注销请求和处理用户登录请求；流入加工 1 的用户信息包括用户注册信息，用户注销信息和用户登录信息；流出加工 1 的用户状态包括新注册的用户号，用户注销成功标志和用户登录成功标志。将该部分分解得图 3.3。同理，对图 3.2 中的加工 3 （处理用户查询请求）分解得图 3.4。

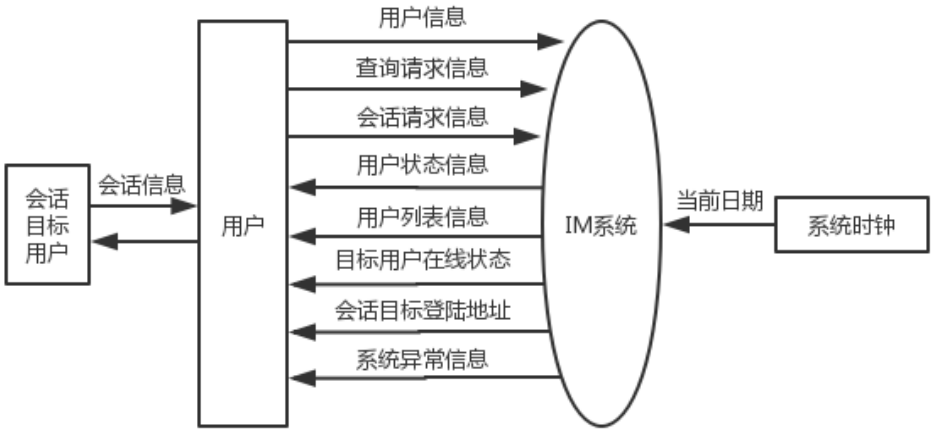

图 3.1 系统数据流图-顶层图

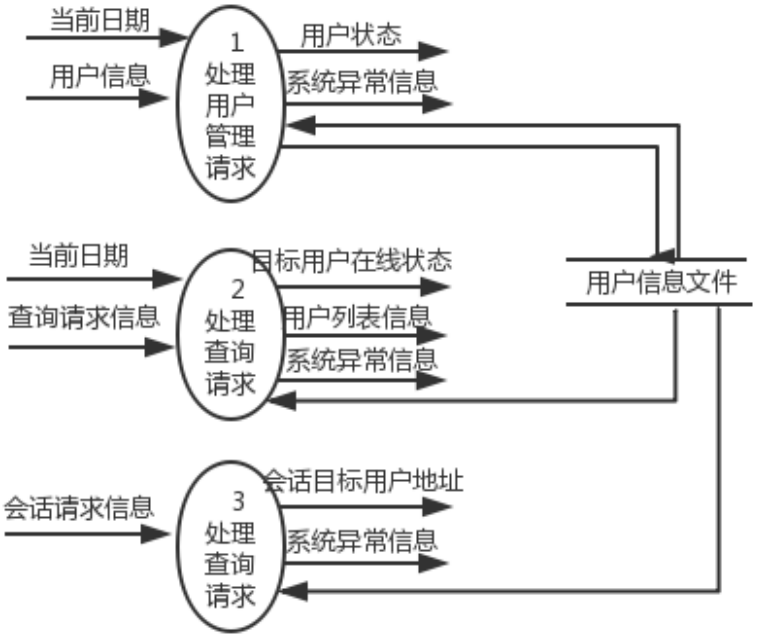

图 3.2 系统数据流图-零层图

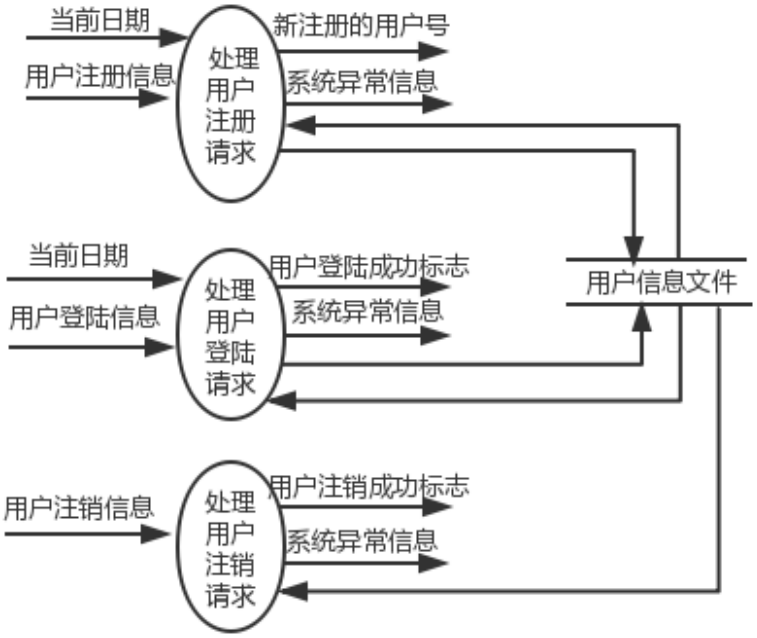

图 3.3 系统加工 1 分解图

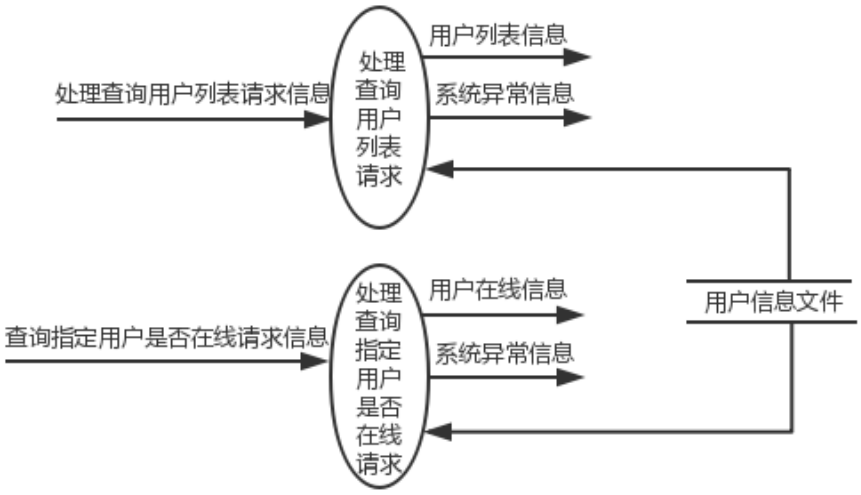

图 3.4 系统加工 2 分解图

数据字典

会话信息＝用户从键盘输入发送给会话目标用户的文本

用户信息＝［用户注册信息｜用户注销信息｜用户登录信息］

查询请求信息＝［查询用户列表请求信息｜查询指定用户是否在线请求信息］

会话请求信息＝［用户号］

处理用户管理请求＝［处理用户注册请求｜处理用户注销请求｜处理用户登录请求］

处理查询请求＝［处理查询用户列表请求｜处理查询指定用户是否在线请求］

用户状态＝［用户号｜用户注销成功标志｜用户登录成功标志］

目标用户在线状态＝［′ 0 ′ ｜′ 1 ′ ］

用户列表信息＝｛用户基础信息｝

会话目标用户地址＝会话目标用户登录的 ip 和端口

用户注册信息＝用户名＋用户密码

用户注销信息＝用户号＋用户密码

用户登录信息＝用户注销信息

查询用户列表请求信息＝目标页码＋页码容量

处理查询指定用户是否在线请求信息＝用户号

用户号＝正整数

用户注销成功标志＝目标用户在线状态

用户登录成功标志＝目标用户在线状态

用户基础信息＝用户号＋用户名＋目标用户在线状态

用户名＝数字和英文字母组成的长度不超过 1 6 字节长度的字符串

密码＝用户名

目标页码＝正整数

页码容量＝正整数

3.3 业务流程

该系统的业务流程主要分为三个阶段，其一是用户登录前，其二是用户登录后发起会话前，其三是发起会话后。用户登录前可以选择登录，注册账号，删除账号等操作；用户登录后开始接收其他用户发送的消息，同时在功能上可以选择退出登录，发起会话，判断用户是否在线，获取用户列表；发起会话成功后用户进入会话状态，此时输入的消息都将发送给目标会话用户，可以输入′quit′关键字退出会话状态。详细业务流程图见图 3.5。

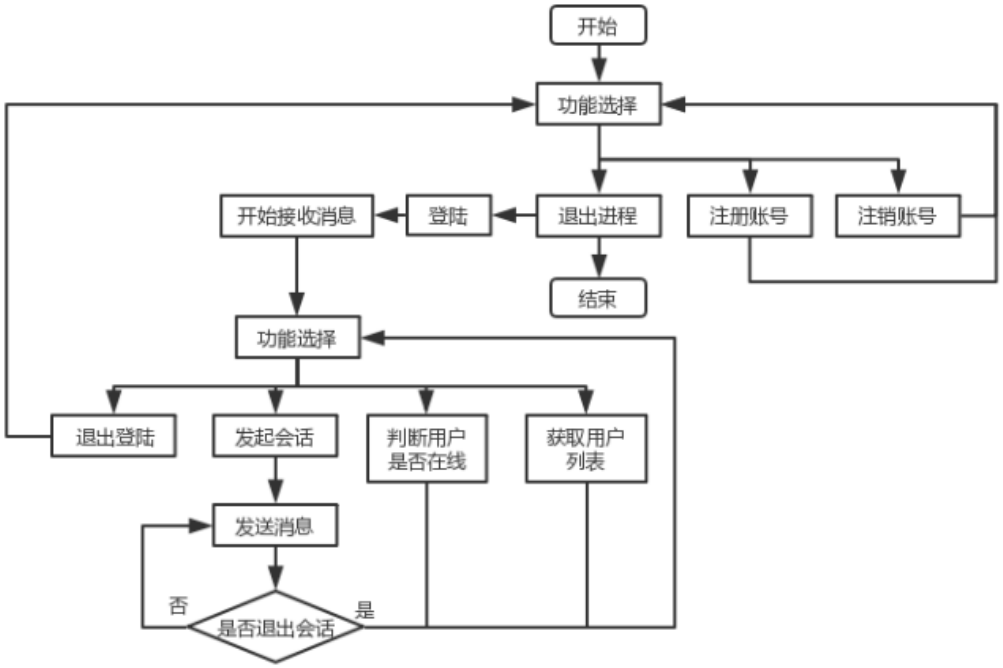

图 3.5 业务流程图

第四章　系统设计

本章节主要分为总体架构设计，私有协议设计，route server 设计，work server 设计和 client 设计五部分。其中，总体架构设计主要包括传输层协议选择，系统架构选择，服务器集群负载均衡以及系统各个模块之间的交互关系（机器级别）；私有协议设计主要包括确定报文结构和行为；route server 设计主要包括路由规则确定，进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 route server 中各个模块之间的交互关系（进程级别）；work server 设计主要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 work server 中各个模块之间的交互关系（进程级别）；client 设计主要要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 client 中各个模块之间的交互关系（线程级别）。

4.1 总体架构设计

本节包括的内容主要有传输层协议选择，系统架构选择，服务器集群负载均衡以及系统各个模块之间的交互关系（机器级别）。

传输层协议选择：考虑到本项目设计的是 10w 级 IM 系统，在系统资源方面可以支持 tcp 协议，且每个客户端只维持一条 tcp 连接，单个 tcp 连接的时延对客户端来说是可以接受的。对 udp 协议来说，虽然其吞吐量更大，但是在复杂网络环境中，upd+自己实现的简单的错误/超时重传实现的可靠传输效率未必比 tcp 协议高，并且实现 udp 的重传以及心跳机制会一定程度上增加编程的复杂性。因此在该项目中使用 tcp 作为传输层协议。

1. 系统架构选择

影响架构选择的需求有使用私有协议，系统支持十万以上用户同时在线，具有高可用性（部分服务器故障仍然能正常提供服务），易扩展，设计简单。

B/S 架构即浏览器请求，服务器响应的工作模式。其优点是后续服务器维护和更新很方便，但是该模式的通信协议需要被浏览器支持。该 IM 系统需要使用私有协议，因此排除 B/S 架构。

P2P 架构即每个节点既是服务器又是客户端的工作模式，P2P 架构适用于多种网络拓扑模型。其没有中心节点，容错能力非常强，支持海量数据通信服务，可以使用私有协议。但是为了满足服务器之间彼此发现，协议设计较为复杂，且有大量非用户数据占用带宽。因此该 IM 系统可以使用 P2P 架构，但是仅仅要求支持十万级用户同时在线，使用 P2P 架构性价比不是很高。

C/S 架构即客户端服务器模式。其支持私有协议，服务器端可以通过服务器负载均衡支持十万级用户同时在线，高可用性，易于扩展，实现简单。综上所述，该 IM 系统选择 C/S 架构，如图 4.1。

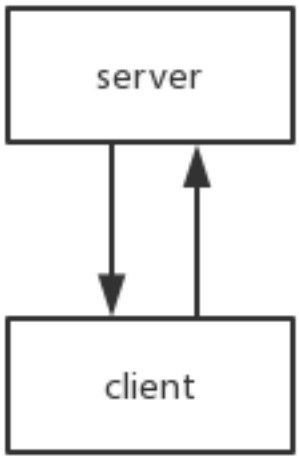

图 4.1 C/S 模型

2. 服务器集群负载均衡

单机服务能力上限较低且容错能力极差，为了满足十万用户同时在线且提高可用性，需要对 server 做两层负载均衡。第一层为一台具备公网 ip 的 route server 服务器，提供路由服务，当客户端请求到来时返回一个 work server 的网络地址给客户端。第二层为多台 work server 服务器，提供面向客户端的具体功能服务。

健康检查：每个 work server 和 route server 之间都维持一条 tcp 连接，且 work server 每隔 10s 通过 tcp 连接向 work server 汇报一次自己当前负责的客户端数量。当 route server 发现某台 work server 的 tcp 连接断开则认为该 work server 出现异常，不再将该 work server 的网络地址发给客户端。

负载均衡算法：为了充分使用每台服务器的资源，不至于有些 work server 空闲而有些 work server 过于繁忙。每次 route server 接受到客户端的请求时，返回当前负责客户端数目最少的 work server 地址，如果有多台这样的 work server 则返回其中任意一台 work server 的地址。

会话保持：client 只在初始化时向 route server 获取一次 work server 地址，后续的所有服务请求都发给该 work server。能够自动确保每次服务请求都打到同一台 work server。

3. 各个模块之间的数据交互

综上所述，能确定 client，work server 和 route server 三者之间的数据交互关系。work server 和 route server 之间的数据交互关系为 work sever 每隔 10s 向 route server 汇报一次自己当前负责的客户数；client 和 route server 之间的数据交互关系为 client 向 route server 请求节点信息，route server 回复 client 一个当前负责客户数最少的 work server 的地址；client 和 work server 之间的交互关系为 client 向 work server 发起服务请求，如果 work server 接受到的是会话请求则返回会话目标客户登录网络地址，其他请求则进行相应的操作且返回操作结果给 client。详见图 4.2。

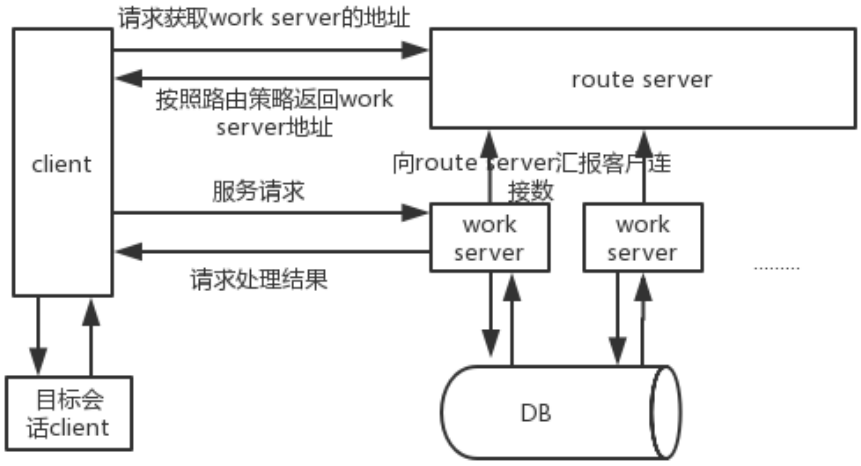

图 4.2 系统模型

4.2 私有协议设计

选择 tcp 协议作为传输层协议，需要在应用层中处理粘包问题，本项目中采用固定头部的方式解决该问题。设计应用层协议的报文结构为：固定头部+不定长的消息内容，固定头部＝16bit 的 type 字段+16bit 的数据总长度。

固定头部 type 的值，宏及其对应的含义

0 OBTAIN_SERVER_ADDR client 向 route 请求获取前客户数最少的 work server 的网络地址

1 REPORT_CONNECT work server 向 route server 汇报当前客户数

2 REGISRERED 注册账号请求/回应

3 LOGIN 登录请求/回应

4 USER_LIST 列出用户列表/回应

5 FIND_USER 查找 xxx 是否在线/回应

6 OBTAIN_CLIENT_ADDR 向 work server 请求获取 xxx 用户的网络地址

7 USER_CONVERSATION 表用户之间的会话内容
12 LOGOUT 退出登录
13 ERROR 错误报告
14 DELETE_USER 删除账号

协议的行为

1. work server 每 10s 向 route server 发送 type=1，内容为本服务器当前 tcp 连接数量的报文汇报本机当前服务的客户端数量。用于 route server 决策将新来的客户端分配给哪个 work server；
2. 客户端启动时向 route server 发送一个 type=0，内容为空的报文。route server 回应该客户端一个 type=0 内容当前客户数最少的 work server 的网络地址；
3. client 向 work server 发送 type=2，内容为用户名：密码的报文（用户名和密码都不超过 16 字节，由字母，数字组成）。work server 回应 client 一个 type=2，内容为′1′/′0′的报文，其中′1′表注册成功，′0′表注册失败；
4. client 向 work server 发送一个 type=3，内容为用户名:密码的报文。Work server 回应一个 type=3，内容为′1′/′0′的报文，其中′1′表登录成功，′0′表登录失败；
5. client 向 work server 发送一个 type=4，内容为 page:cap 的报文（其中 page 表第 page 页列表，cap 表每页的容量）。Work server 回应一个 type=4，内容为当前在线用户名列表（每行的内容包括 uid，uname 和 online 字段）的报文；
6. client 向 work server 发送一个 type=5，内容为一个用户名 xxx 的报文。Work server 回应一个 type=5，内容为′1′/′0′的报文。其中′1′表用户 xxx 当前在线，′0′表用户 xxx 不在线；
7. client 向 work server 发送一个 type=6，内容为 xxx 用户名的报文。Work server 回应一个 type=6，内容为 xxx 用户 ip+port 的报文；
8. 向另一个 client 发送一个 type=7，内容为对端 client ip port +会话消息的报文。表向对端 client 发送一条消息；
9. 向 work server 发送一个 type=12，内容为空的报文，表退出登录；
10. 当 work server 接收到的数据格式错误，或 work server 出现系统故障时向发送请求的客户端返回一个 type=13，内容为错误提示的报文，表出现异常；
11. client 向 work server 发送一个 tyep=14，内容为用户号：密码的报文，表删除账号；

4.3 route server 设计

route server 设计主要包括路由规则确定，进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 route server 中各个模块之间的交互关系（进程级别）。

为了尽量使所有 work server 负载强度相近，不至于某些机器过载的时候另一些机器却处于空闲状态，路由策略制定为：当客户端访问 route server 时，route server 给客户端返回一个当前负责客户连接数最少的 work server。

route server 的功能是管理 work server 以及给 client 选择合适的 work server。属于 I/O 密集型业务。为了提高并发能力，这里考虑使用多进程单线程模型，父进程处理网络 I/O，子进程中处理业务逻辑。如此一来，同时确定了进程/线程模型以及 I/O 模型。在多进程单线程模型中我们需要解决 I/O 监听问题，会话保持问题，新连接分配问题，进程间通信问题。其中进程间通信问题包含父子进程之间通信以及多个子进程之间通信问题。

I/O 监听：此处为 I/O 密集型业务，考虑到提高性能，使用 ET 模式的 epoll 监听网络 I/O。

会话保持：会话保持即如何确保一个 socket 上的后续事件都在同一个子进程中处理。每个子进程都有一个独立的 epoll 内核表。父进程中只监听 socket 连接事件， 当父进程中接收到新连接时不对连接进行任何处理，直接通知一个子进程获　取该 socket 连接，并将该 socket 连接注册到该子进程中的 epoll 内核表上。因此该 socket 连接上的可读事件到来时只有该子进程能获取，保证了一个 socket 上的后续事件都在同一个子进程中处理。

新连接分配算法：新连接到来的情况有两种，client 初始化时向 route server 请求获取 work server 地址和有新的 work server 加入。这两种请求的处理时长都非常短，才用轮转法即可让每个子进程工作强度基本一致；

父子进程通信：当新连接到来时父进程需要通知子进程来处理，父进程对子进程进行管理时需要发送信号给子进程。因为这些通信需求都是两个进程之间通信，可以在父进程和每个子进程之间都建立一个双向管道即可。

子进程之间通信：子进程中对处理 client 请求的业务逻辑是返回当前负责客户数最少的 work server 地址。因此每个子进程都需要知道所有 work server 的地址以及负责客户数的实时数据。即所有子进程需要共享所有 work server 的地址以及负责客户数。因此使用共享内存来记录这些信息。

经过这些分析 route server 的工作模型也就确定了，在父进程中使用 epoll 监听用户请求和 work server 的心跳包，发现了新的连接到来则按轮转法确定一个子进程并通过管道通知该进程。被通知的子进程从监听 socket 上获

取连接 socket 并将其注册到自己的 epoll 内核事件表上，该 socket 连接上的后续可读事件都将由该子进程负责。同时，所有子进程共享所有 work server 的客户连接数信息，当客户端访问 route server 时 route server 返回客户连接数最少的 work server 地址给该客户端。如图 4.3。

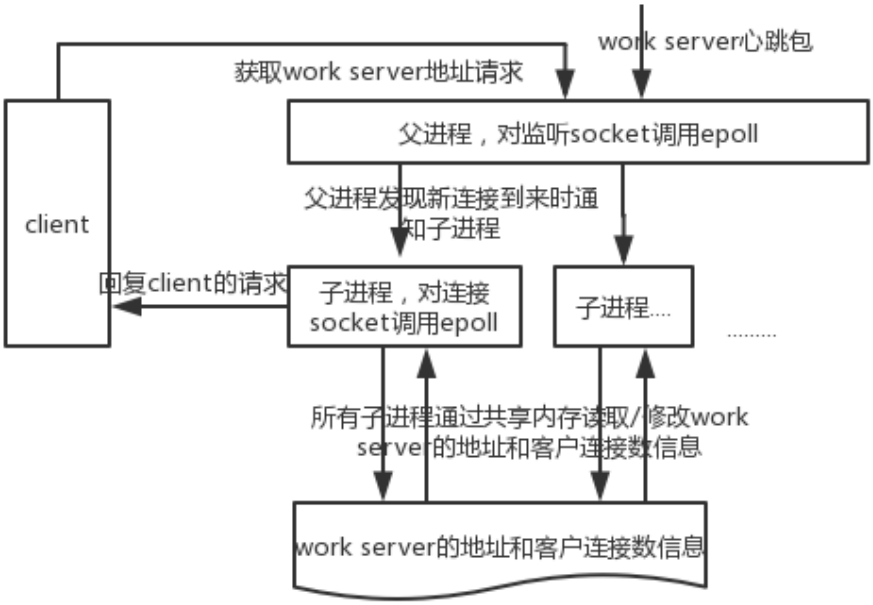

图 4.3 route server 模型

4.4 work server 设计

work server 设计主要包括进程/线程模型选择，进程/线程间通信方式选择，I/O 处理模型确定，确定 work server 中各个模块之间的交互关系（进程级别）。

work server 的功能是为 client 提供注册账号，删除账号，用户登录，分页查看用户列表，查看指定用户是否在线接，向指定用户发其会话等服务。为了提高并发能力，这里同样考虑使用多进程单线程模型，父进程处理网络 I/O，子进程中处理业务逻辑。同时父进程中另外创建一个子线程用于向 route server 汇报 work server 当前负责的用户数。类似于 route server 中，这里的多进程单线程模型中同样需要处理 I/O 监听问题，会话保持问题，新连接分配问题，进程间通信问题。另外，考虑到可能会使用内网的机器做 work server，因此此处还需要解决内网穿透问题。

I/O 监听问题：此处为 I/O 密集型业务，考虑到提高性能，使用 ET 模式的 epoll 处理网络 I/O。

会话保持：会话保持即如何确保一个 socket 上的后续事件都在同一个子进程中处理。每个子进程都有一个独立的 epoll 内核表。父进程中只监听 socket

连接事件， 当父进程中接收到新连接时不对连接进行任何处理，直接通知一个子进程获取该 socket 连接，并将该 socket 连接注册到该子进程中的 epoll 内核表上。因此该 socket 连接上的可读事件到来时只有该子进程能获取，保证了一个 socket 上的后续事件都在同一个子进程中处理。

新用户分配算法：轮转法。

父子进程通信：当新连接到来时父进程需要通知子进程来处理，另外父子进程之间需要传递系统信号。对于系统信号采用统一事件源（在第二章有提到）方式处理。对 于父进程通知子进程处理新到的连接，因为这些通信需求都是两个进程之间通信，可以在父进程和每个子进程之间都建立一个双向管道即可。

子进程之间通信：由于新客户连接到来/离开的消息只有某一个子进程才知道，因此要统计机器当前客户数的话需要这些子进程共同维护客户数信息。对此采用共享内存通信方式。

内网穿透问题：为了使用没有公网 IP 的机器做 work server，需要解决内网穿透问题。当 work server 向 route server 发起 tcp 连接时，route server 能获取 NAT 转换后的网络地址。可以通过端口复用技术让 work server 用同一个端口给 route server 汇报 work server 当前负责的客户数和监听 client 的请求。如此，route server 获取的 NAT 转换后的网络地址即为提供服务的网络地址。

经过这些分析 work server 的工作模型也就确定了，在父进程中另开一个子线程用于向 route server 发送心跳包，心跳包的内容为本机当前负责的客户连接数。同时在父进程中使用 epoll 监听用户请求，发现了新的连接到来则按轮转法确定一个子进程并通过管道通知该进程。被通知的子进程从监听 socket 上获取连接 socket 并将其注册到自己的 epoll 内核事件表上，该 socket 连接上的后续可读事件都将由该子进程负责。如图 4.4。

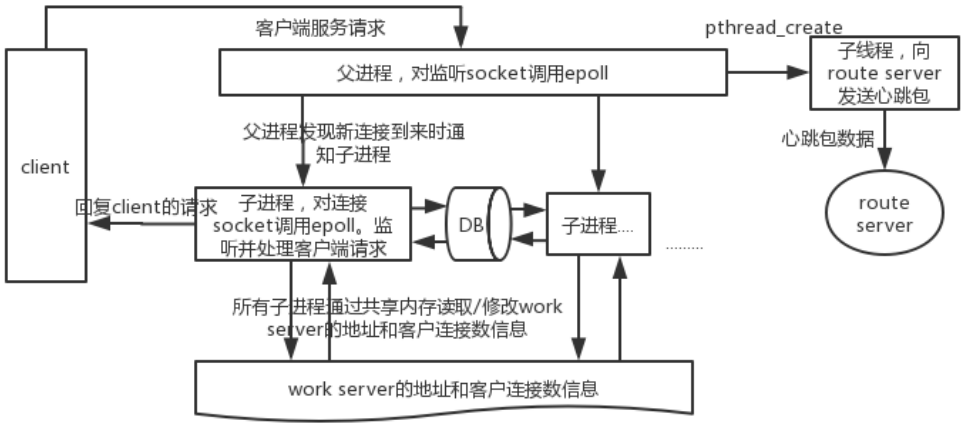

图 4.4 work server 模型

4.5 client 设计

client 的功能是按照定制的私有协议访问 work server 提供的服务以及和其他 client 之间发送/接收消息。为了实现同时发送消息给其他 client 和接收多个 client 的消息，即发送消息和接收消息不相互阻塞。这里使用多线程模型，开启两个线程，主线程中运行和 work server 之间的交互代码以及发送消息给其他 client 的逻辑代码，子线程用于监听其他 client 发送过来的消息。Client 工作模型见图 4.5。

内网穿透问题：由于进行通信的两个 client 可能处于两个不同的局域网内，需要解决 client 之间的内网穿透问题。client 在向其他 client 发送消息前需要先登录，因此 work server 能获取 client 发送登录信息的 socket 绑定的网络地址进行 NAT 转换后的地址。所以只要通过端口复用让监听其他 client 消息的 socket 和发送登录信息的 socket 绑定同一个网络地址，然后 client 向 work server 发起会话请求时获取的目标会话用户的网络地址即为目标会话用户监听 client 消息的 socket 绑定的网络地址。

线程间通信问题：考虑如下情景，client_a 主动向 client_b 发起会话，这时候 client_a 和 client_b 之间建立历来一条 tcp 连接（需求分析中有说明传输层协议使用 tcp），client_a 为 connect 端，client_b 为 listen 端。如果此时 client_b 要向 client_a 发送消息的话显然不需要再向 client_a 建立另一个 tcp 连接。即 client 发起会话时有两种情况，一种情况是该 client 和目标会话 client 之间已经建立了 tcp 连接，另一种情况是两个 client 之前没有没有建立 tcp 连接。因此发起会话时需要先判断有没有 listen 到会话目标 client 的 tcp 连接，如果没有的话发起会话方 client 通过 connect 向会话目标 client 发起 tcp 连接。考虑到 listen 系统调用处于子线

程中而发送消息逻辑处于主线程中，需要进行线程间通信。此处可以通过全局变量进行线程间通信。

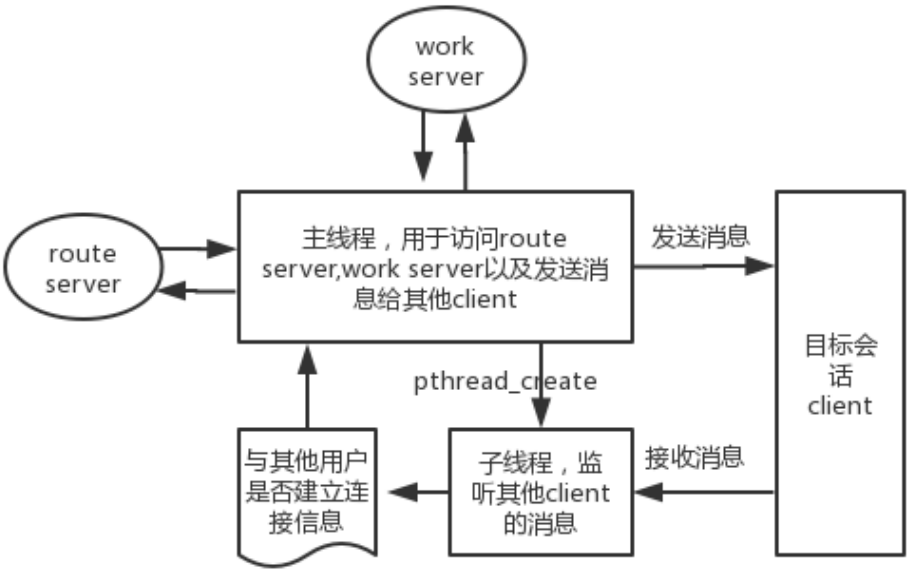

图 4.5 client 模型

第五章　系统详细设计与实现

5.1 开发环境及工具

1. 开发环境

操作系统:Ubuntu 18.04.2 LTS

CPU 型号:Intel（R） Core（TM） i5-4210U CPU @ 1.70GHz

内存大小:3.8 G

磁盘型号: TOSHIBA-TR200

2. 开发工具

编译器:gcc7.3.0 g++7.3.0

编辑器:sublime_text3.0

5.2 route server 详细设计与实现

1. route server 详细流程图设计。由于 route server 的详细实现逻辑稍微有点复杂，这里将其流程图分为 5 幅图。总体流程图见图 5.1，其中用 listenfd、和 cups 初始化一个进程池对象，父进程逻辑和子进程逻辑三部分较复杂，分别用的 A 部分,B 部分,C 部分指代，这三部分的流程图分别是图 5.2，图 5.3 和图 5.4。同理，图 5.4 中的 a 部分指代的图 5.5。

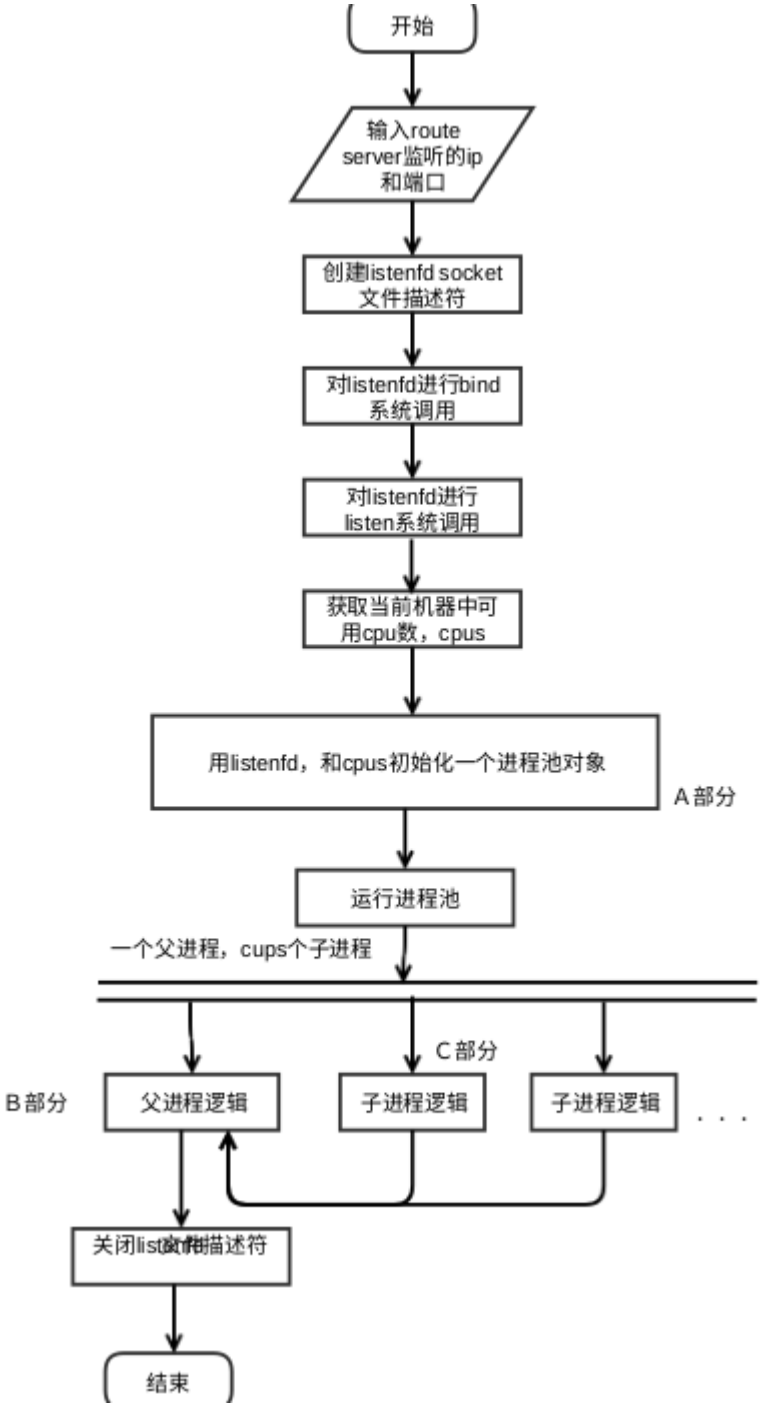

图 5.1 route server 流程图-总体

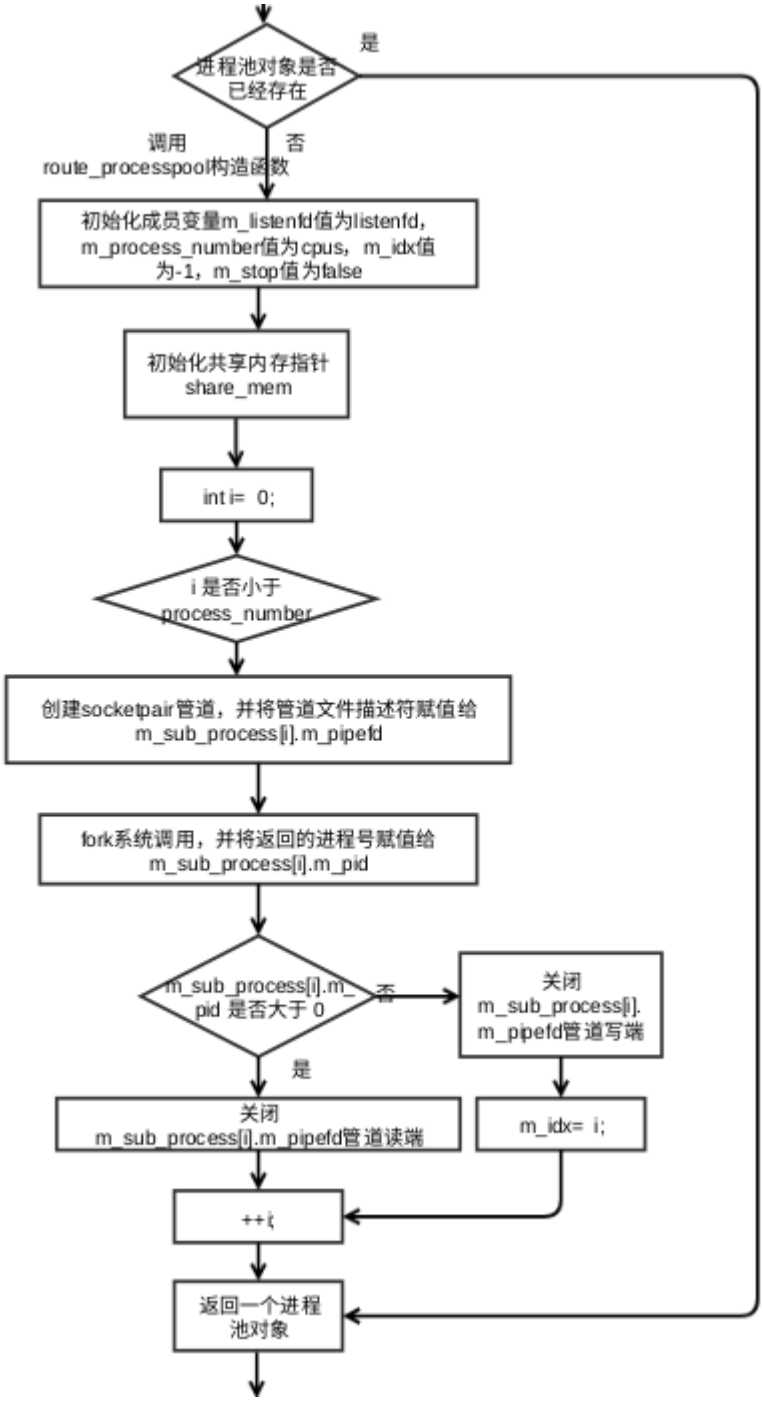

图 5.2 route server 流程图-A 部分

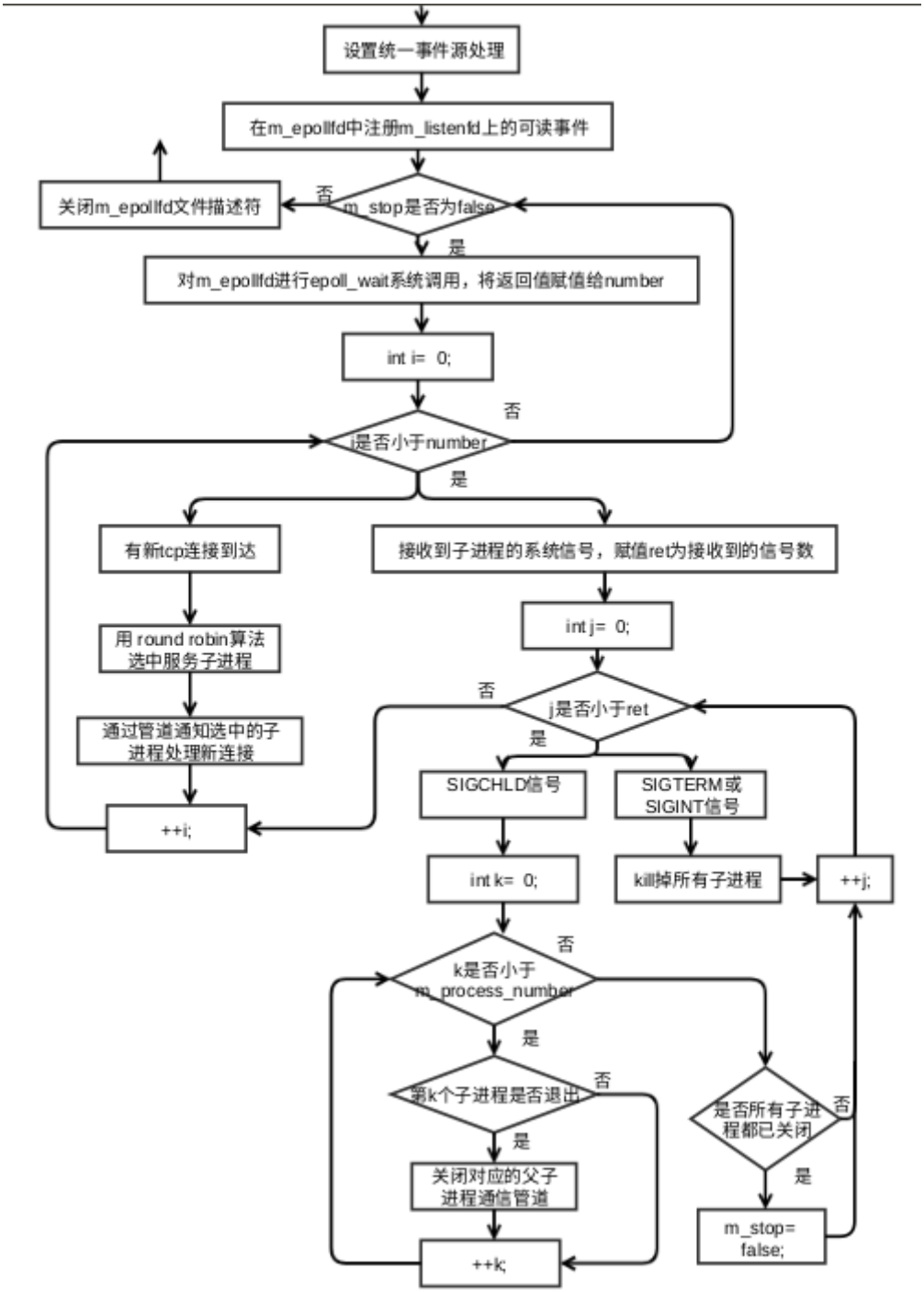

图 5.3 route server 流程图-B 部分

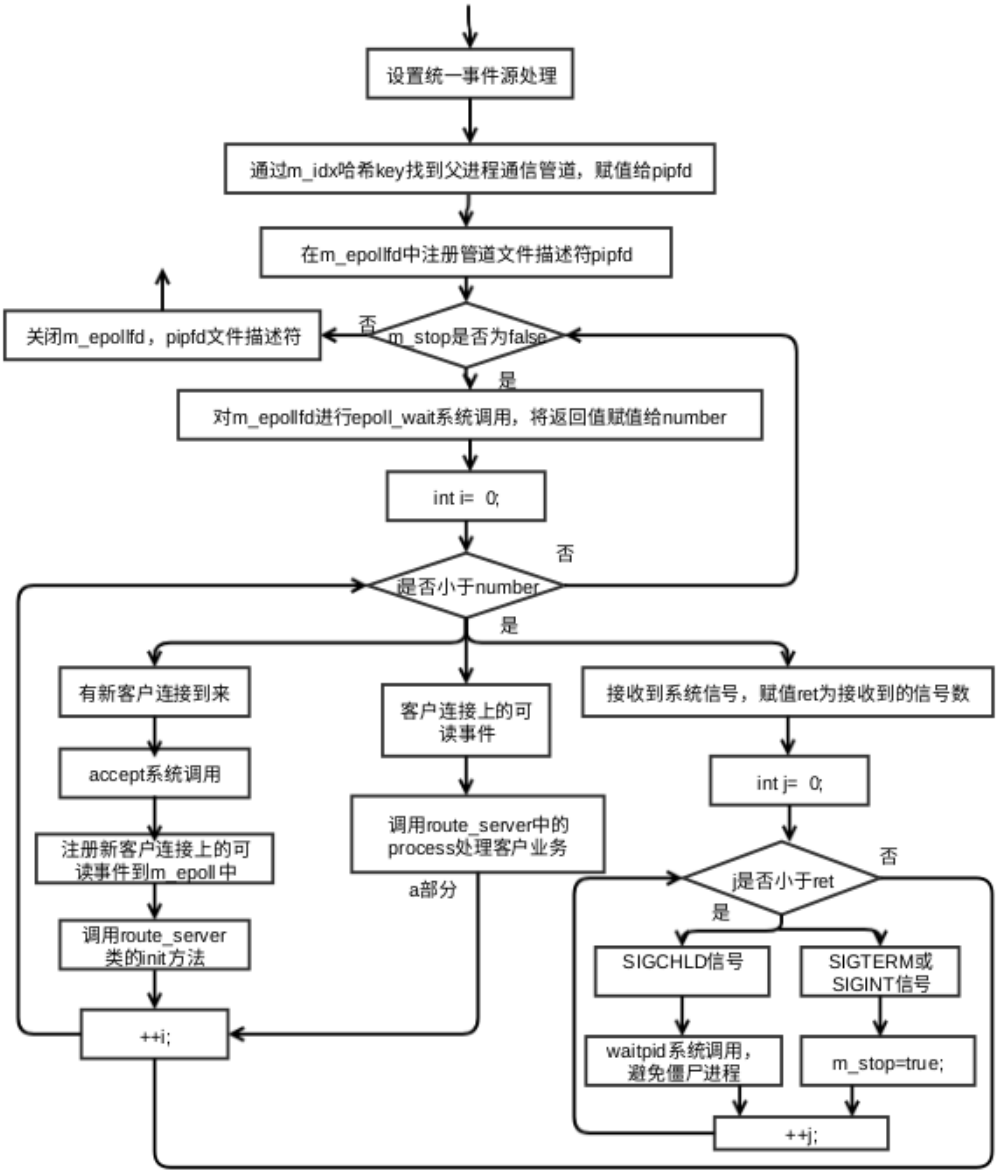

图 5.4 route server 流程图-C 部分

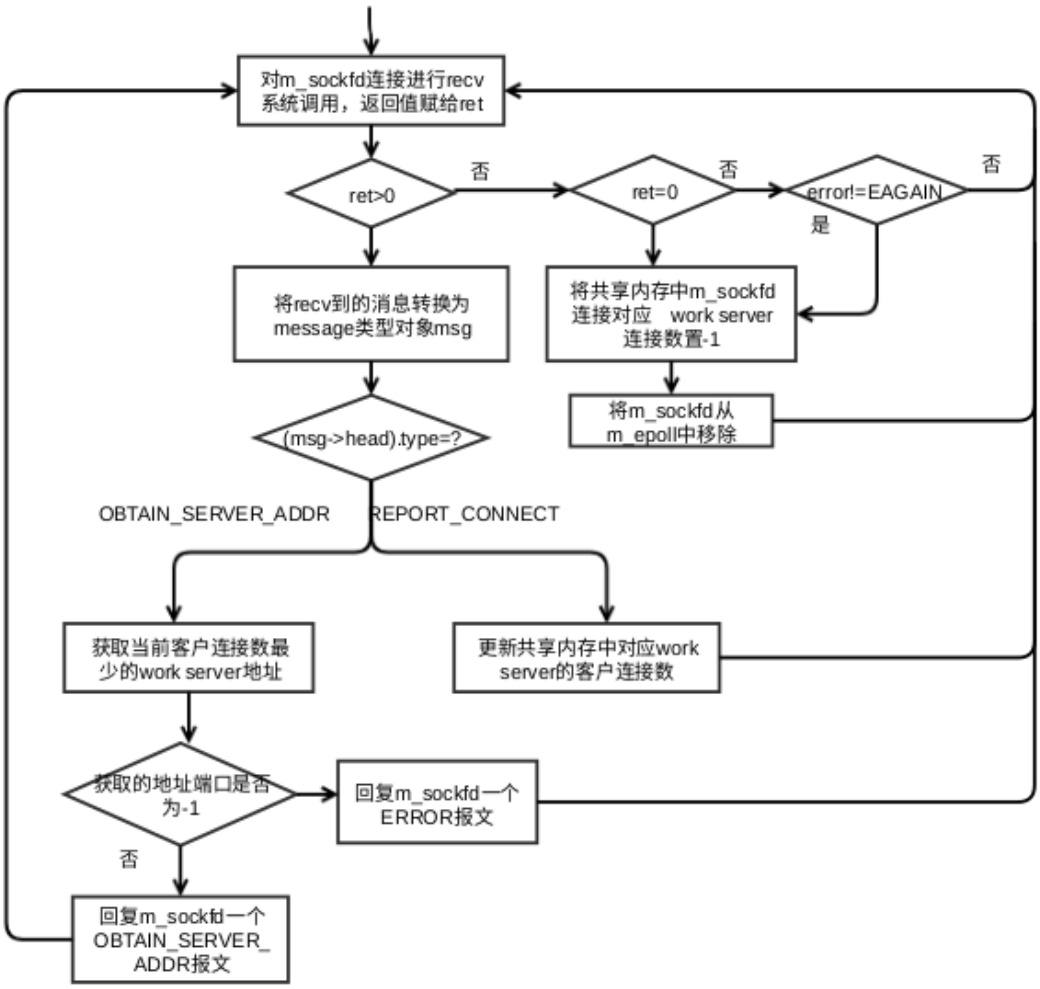

图 5.5 route server 流程图-C 部分-a 部分

2. 部分核心代码

统一事件源实现

```
template <typename T>
void route_processpool<T>::setup_sig_pipe() {
    // 每个子进程都有一个 epoll 事件监听表
    m_epollfd = epoll_create(5);
    assert(m_epollfd != -1);
    // 创建信号管道
    int ret = socketpair(PF_UNIX, SOCK_STREAM, 0, sig_pipefd);
    assert(ret != -1);
    setnonblocking(sig_pipefd[1]);
    // 将信号管道 sig_pipefd[0] 端注册到每个子进程 m_epollfd 内核事件表上
    // 使每个子进程都能从 sig_pipefd[0] 接收到付进程从 sig_pipe[1] 发过
```

```
来的信号
        addfd(m_epollfd, sig_pipefd[0]);
        // 设置信号处理函数
        // 子进程状态发生变化(退出或暂停信号)
        addsig(SIGCHLD, sig_handler);
        // 终止进程信号 kill
        addsig(SIGTERM, sig_handler);
        // 键盘输入 ctrl + c
        addsig(SIGINT, sig_handler);
        // 往读端被关闭的管道或 socket 写数据
        addsig(SIGPIPE, SIG_IGN);
    }
    static void sig_handler(int sig) {
        int save_errno = errno;
        int msg = sig;
        send(sig_pipefd[1], (char*)&msg, 1, 0);
        errno = save_errno;
    }
    static void addsig(int sig, void(hanler)(int), bool restart = true) {
        struct sigaction sa;
        memset(&sa, /\0/, sizeof(sa));
        sa.sa_handler = hanler;
        if(restart) {
            sa.sa_flags |= SA_RESTART;
        }
        sigfillset(&sa.sa_mask);
        assert(sigaction(sig, &sa, NULL) != -1);
    }
```

父进程中 round robin 算法实现

```
    // 如果有新的连接到来，采用 round Robin 方式将其分配给一个子进程处理
    int i = m_sub_process_counter;
    do {
        if(m_sub_process[i].m_pid != -1) {
            break;
```

```
        }
        i = (i + 1) % m_process_number;
    } while(i != m_sub_process_counter);
    if(m_sub_process[i].m_pid == -1) {
        m_stop = true;
        break;
    }
    m_sub_process_counter = (i + 1) % m_process_number;
    // 告诉第 i 个子进程有新客户到达
    send(m_sub_process[i].m_pipefd[0], (char*)&new_conn, sizeof(new_conn), 0);
```

4.3 work server 详细设计与实现

1. work server 详细流程图设计。考虑到 work server 的实现逻辑较为负责，这里将其流程图分为 3 副图。Work server 的总体流程图为图 5.6，该图中的父进程逻辑和子进程逻辑中的 process 业务较为复杂，分别用 A 部分和 B 部分指代，A 部分对应图 5.7，B 部分对应图 5.8。

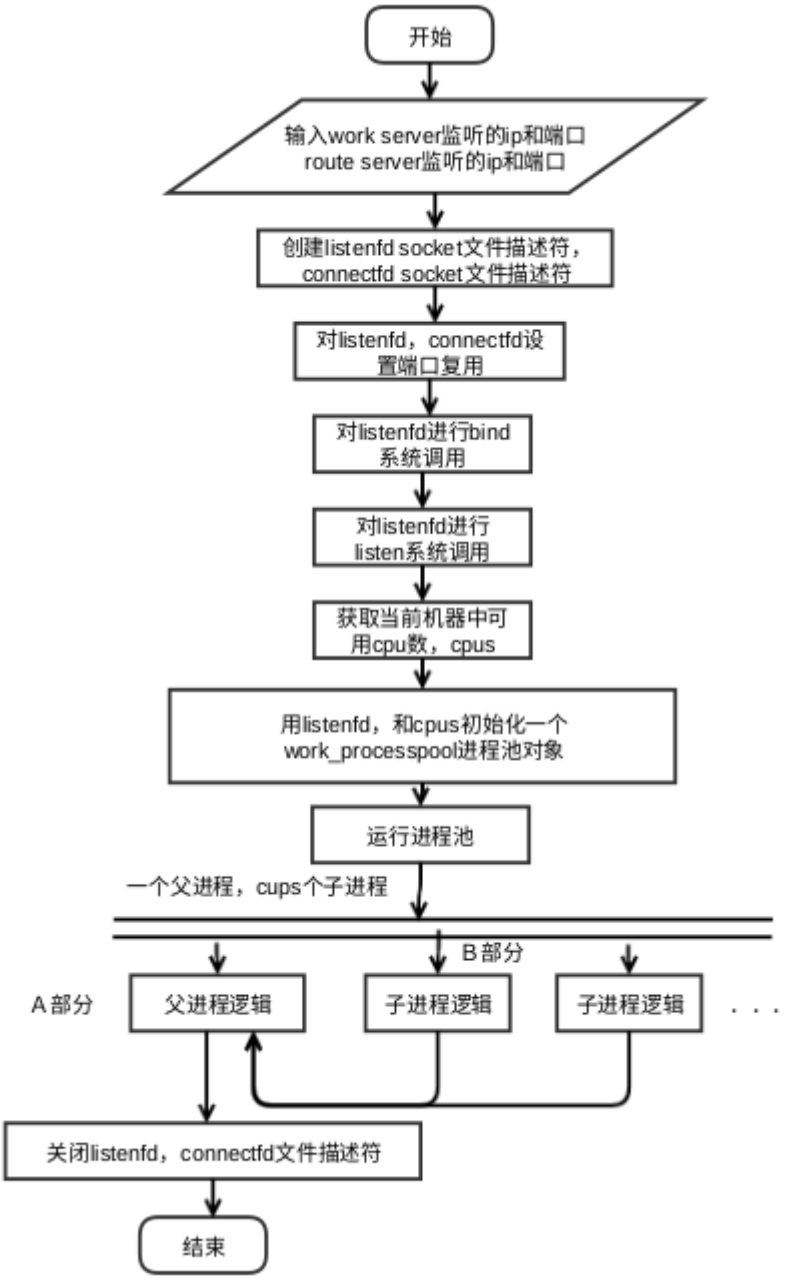

图 5.6 work server 流程图-总体

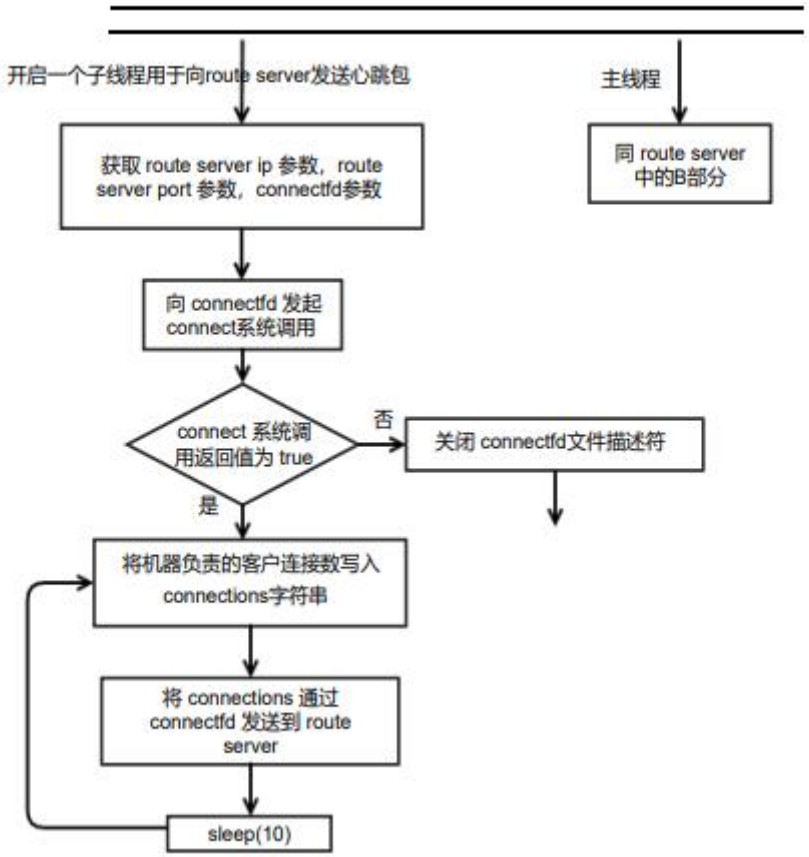

图 5.7 work server 流程图-A 部分

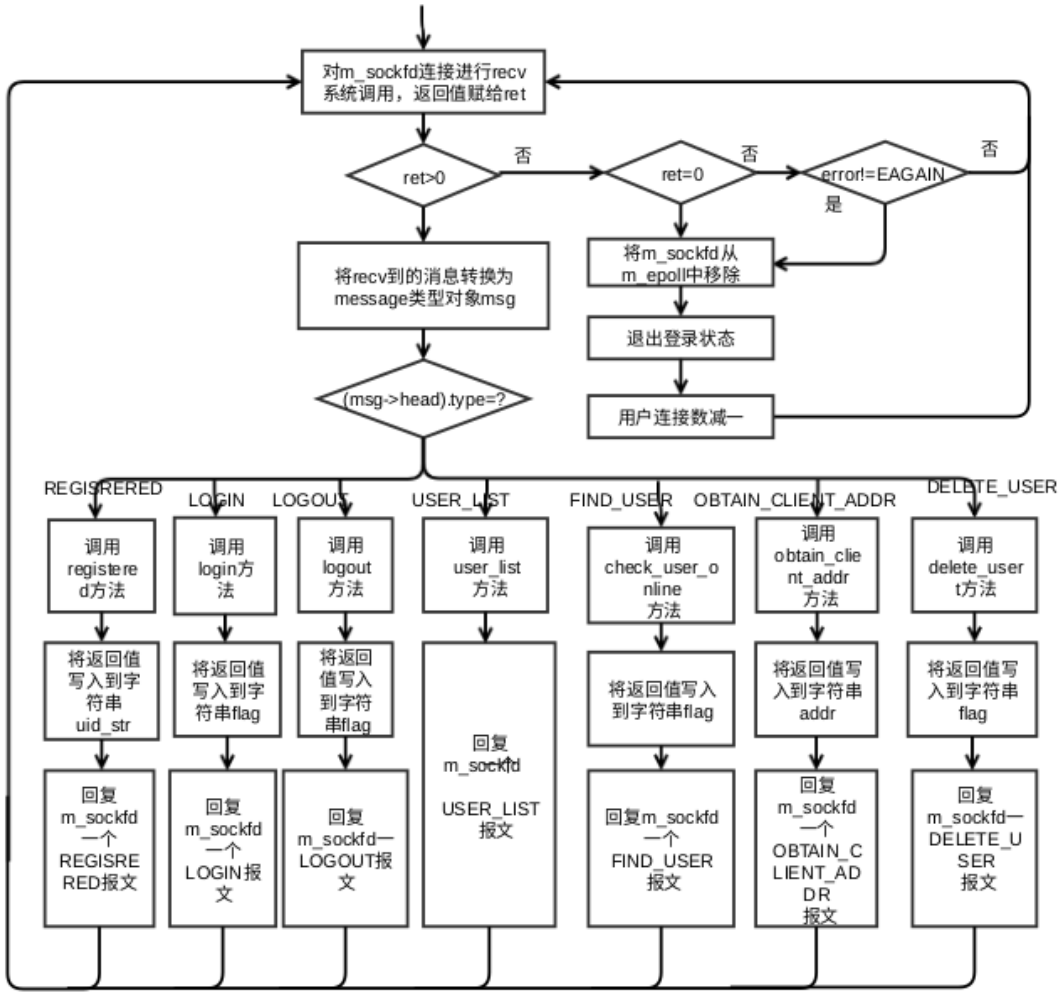

图 5.8 work server 流程图-B 部分

1．部分核心代码

发送心跳包子线程实现

```
// 向 route_server 汇报本机器当前用户数
template<typename T>
void* work_processpool<T>::report_connect(void *data) {

    int ret = 0;
    pthread_create_param *route_server = (pthread_create_param*)data;

    char *route_server_ip = (route_server->addr).ip;
    int route_server_port = (route_server->addr).port;
    int connectfd = route_server->connectfd;

    // printf("%s\n", route_server_ip);
```

```
        // printf("%d\n", route_server_port);

        struct sockaddr_in route_address;
        bzero(&route_address, sizeof(route_address));
        route_address.sin_family = AF_INET;
        inet_pton(AF_INET, route_server_ip, &route_address.sin_addr);
        route_address.sin_port = htons(route_server_port);

        if(connect(connectfd, (struct sockaddr*)&route_address,
sizeof(route_address)) < 0) {
            printf("connection failed\n");
            close(connectfd);
            return NULL;
        }

        char connections[6];

        while(true) {

            sprintf(connections, "%d", *share_mem);
            send_msg(connectfd, 1, connections);
            sleep(REPORT_CONNECT_TIME_INTERVAL);
            printf("----\n");
        }
    }
```

父进程中信号处理

```
    switch(signals[i]) {
        case SIGCHLD:{
            pid_t pid;
            int stat;
            while((pid = waitpid(-1, &stat, WNOHANG)) > 0) {\
                for(int i = 0; i < m_process_number; ++i) {
                    // 如果进程池中第 i 个子进程退出了，则主进程关闭相应的
通信管道，并设置相应的 m_pid 为 -1 表示该子进程已退出
```

```
                    if(m_sub_process[i].m_pid == pid) {
                        printf("child %d leave\n", i);
                        close(m_sub_process[i].m_pipefd[0]);
                        m_sub_process[i].m_pid = -1;
                    }
                }
            }
            // 如果所有子进程都退出了，则父进程也关闭
            m_stop = true;
            for(int i = 0; i < m_process_number; ++i) {
                if(m_sub_process[i].m_pid != -1) {
                    m_stop = false;
                    break;
                }
            }
            break;
        }
        case SIGTERM:
        case SIGINT:{
            // 如果父进程接收到终止信号，那么就杀死所有子进程并等待它们
全部结束
            printf("kill all the child now\n");
            for(int i = 0; i < m_process_number; ++i) {
                int pid = m_sub_process[i].m_pid;

                if(pid != -1) {
                    kill(pid, SIGTERM);
                }
            }
            break;
        }
        default:{
            break;
        }
    }
```

5.4 client 详细设计与实现

1．client 详细流程图设计。见图 5.9。

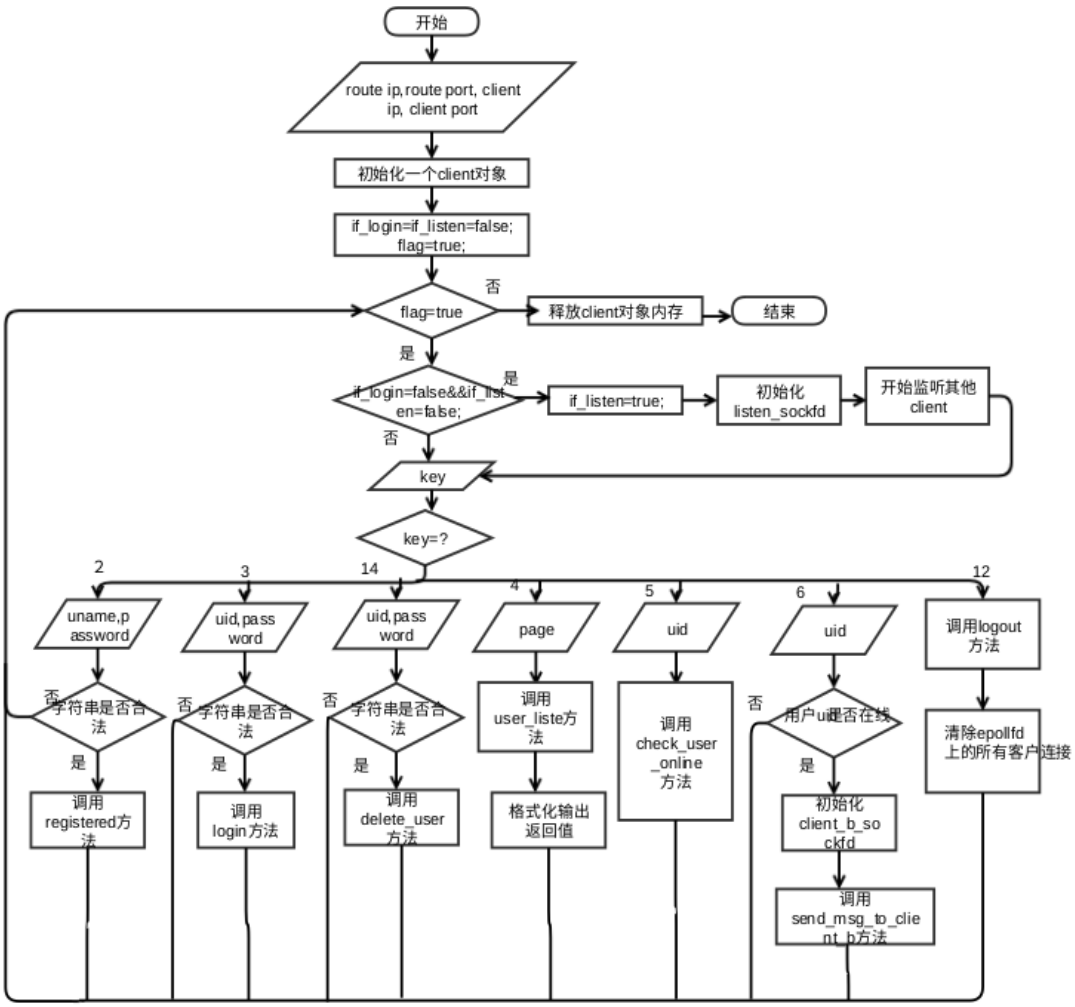

图 5.9 client 流程图

第六章　系统测试

6.1 测试概述

本章节主要进行功能测试和压力测试。功能测试即通过用户输入不同的样例来测试所有功能是否正常工作；压力测试即通过异步 connect 技术来测试该系统最多能支持多少用户同时登录以及支持多少用户同时在线。

6.2 功能测试

通过客户端输入测试注册账号，删除账号，用户登录，获取用户列表，判定用户是否在线，发起会话等功能是否符合预期。

测试注册账号功能：选择功能 2，然后按照提示输入 user name 和 password。注册成功后返回 user id，如图 6.1 测试成功返回了 user id 15。

测试用户登陆功能：选择功能 3，然后按照提示输入 user id 和对应的 password。登录成功后返回 " login success~ " 提示，失败则提示 " logout failed~ " ，如图 6.2。

测试用户列表功能：选择功能 4，然后输入接着输入要查看的页数。如图 6.3。

测试判断用户是否在线功能：选择功能 5，接着输入要查看的用户 id。若用户 xx 在线则返回 " user: xx is online~ " ，否则返回 " user: xx is not online~ " 。如图 6.4。

测试发起发起会话功能：选择功能 6，接着输入目标会话用户的 id。如果目标用户不在线会发起会话失败，反之则进入会话状态。在进入会话状态后可以输入 " quit " 退出会话状态。在向目标用户发起会话前可以先查看一下目标用户是否在线。下图测试中，用户 15 先向用户 10 发起会话，由于用户 10 不在线进入会话状态失败。接着用户 15 又向用户 11 发起了会话并成功进入到了会话状态，然后向用户 11 发送了消息 " hello~ " 。用户 11 接收到了消息，接着用户 11 也向用户 15 发起了会话，用户 11 成功进入会话状态后向用户 15 发送了消息 " hi~ " 。用户 15 接收到了该消息。最后用户 11 输入 " quit " 退出了会话。

退出登陆功能测试：选择功能 12。退出登录成功后会显示提示 " logout success~ " ，否则会显示 " log out failed~ " 。下图测试中用户 15 退出了登录状态。接着在用户 11 客户端上查看用户 15 是否在线，返回了符合预期的提示 " user: 15 is not online~ " 。

删除账号功能测试：选择功能 14，接着按照提示输入要删除的用户 id 和对应的 password。删除成功后返回提示 " delete user xxx success~ " ，否则

会显示 " delete user xxx failed~ " 。如下图测试中删除了用户 15，然后再尝试登录用户 15 返回了登录失败提示。

```
yuan@yuan-Lenovo-G50-70:~/edit/chat_geloutingyu/client$ g++ main.cpp client.cpp
../internal/message.cpp -o client -lpthread
yuan@yuan-Lenovo-G50-70:~/edit/chat_geloutingyu/client$ ./client 127.0.0.1 12345
 127.0.0.1 22347
welcome to chat_geloutingyu~
-------------------------------------------
you can input 2 to registered an account number
you can input 3 to login
you can input 14 to delete user
you can input -1 to quit
you can input 0 to get help
-------------------------------------------

>>>2
please input user name and password(user name and password are                n
o more than 16 bytes, consisting of letters and numbers)
user name:abc
password:123
registered success~
your user id is: 15
```

图 6.1 测试注册账号功能

```
>>>3
please input your user id and password~
uid:15
password:123
login success~
then you can do this...
-------------------------------------------
you can input 4 x to show the user list(其中 x 表查看第 x 页的用户列表，每页最多
 20 个用户)
you can input 5 x to find if a user is online(其中 x 为要查找的 user_id)
you can input 6 x to send message to x(其中 x 为在线用户 user_id)
you can input 12 to logout
you can input 0 to get help
-------------------------------------------
```

图 6.2 测试用户登陆功能

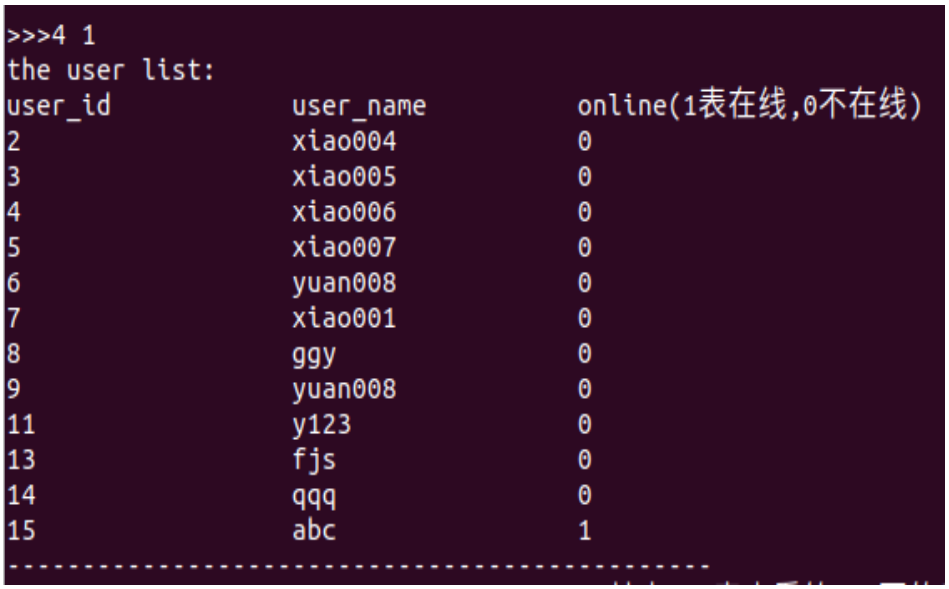

图 6.3 测试用户列表功能

图 6.4 测试判断用户是否在线功能

图 6.5 测试发起会话功能

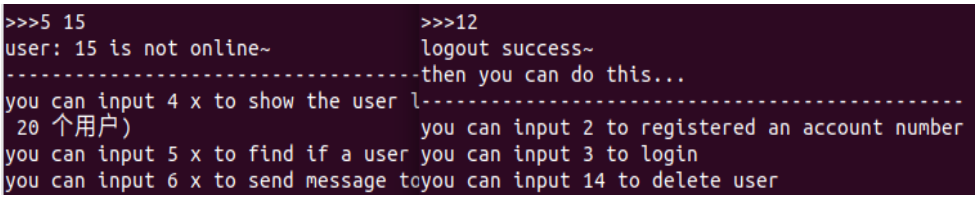

图 6.6 测试退出登陆功能

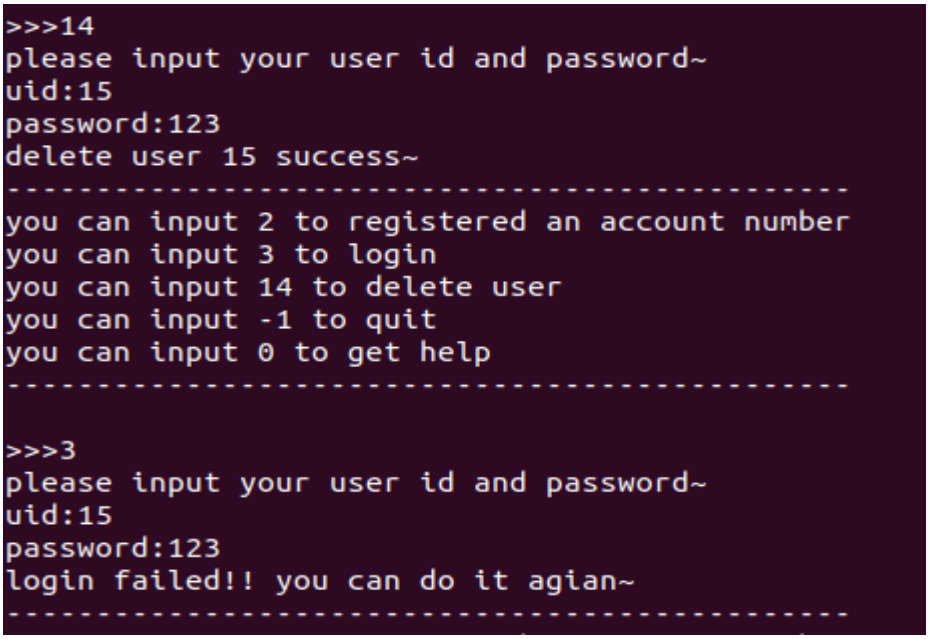

图 6.7 测试删除账号功能

参考文献

[1]萨师煊，王珊.数据库系统概论[M].北京：高等教育出版社，2002 年 2 月：3-460.
[2]孙涌.现代软件工程[M].北京：北京希望电子出版社，2003 年 8 月：1-246.
[3] 张海藩.软件工程导论[M].北京：清华大学出版社，2003：1-347.
[4]徐英，谷雨.对分课堂在提升《C++程序设计》课程教学效果的实践与思考[J].教育教学论坛，2017,（01）：213-214.
[5]徐洪智，张彬连，钟键.《C++程序设计》课程实验教学改革与探索[J].现代计算机（专业版），2017,（15）：50-53.
[6]刘祜，腰善丛，张濡亮.基于 C++Builder 与 MATFOR 平台的 MT/AMT 数据处理软件开发[J]. 铀矿地质，2017,33（01）：29-36.
[7]董昌源，刘疆，王仁鹤.浅谈 C 和 C++异同[J].艺术科技，2016,29（12）：74.[2017-10-10].
[8]李素若.基于慕课的编程类课程混合式教学模式研究与实践--以《C++程序设计》为例[J].软件导刊，2017,16（01）：189-191.
[9]洪建.探究 c++编程中常见问题与解决对策[J].信息化建设，2018,（05）：1-91.
[10] 汪明晔.内存泄漏检测方法研究综述[J].电脑知识与技术，2018,（28）：210-276.
[11] 陈廷勇，李冰洁.C/C++运算中数据类型隐含转换溢出分析[J].中国高新技术企业，2018,（17）1-130.
[12]郜非.基于实践的 C++互动教学模式的建立[J].高教学刊，2017,（08）：127-128.
[13]李峰，刘洞波.基于翻转课堂的 C++课程教学模式探究[J].黑龙江教育（高教研究与评估），2017,（06）：13-14.
[14]李丽薇.浅析 Visual C++编程技巧[J].黑龙江科技信息，2017,（16）：190.
[15]郑步芹，石鲁生.“项目案例驱动”在《C++面向对象程序设计》课程改革中的应用研究[J] .电脑知识与技术，2017,13（02）：164-165.

致　谢

经过两个半月的努力专研这篇论文终于大功告成了。本作品是用 c++实现的，大概 6000 行代码，所有的技术实现都是自己在理论的基础上独立完成的，没有参考任何其他人的 demo。由于此前并没有写过这么大的 c++网络相关的项目，在开发的过程中遇到过许多问题，如由于 tcp 连接断开后端口处于 time wait 状态这一特性让我无法完成 tcp 内网穿透实现，后面考虑到端口复用才解决这个问题。同时，由于该项目较为复杂，开发易于扩展，维护的代码对我来说也是非常困难的。另外，这是我第一次写论文，在写论文的过程中也遇到了许多问题。此处要非常感谢我的指导老师温卫老师在期间给我的指导，让我得以顺利完成论文。

同时，我也要感谢本论文所引用的各位学者的专著，如果没有这些学者的研究成果的启发和帮助，我将无法完成本篇论文的最终写作。至此，我也要感谢我的朋友和同学，他们在我写论文的过程中给予我了很多有用的素材，也在论文的排版和撰写过程中提供热情的帮助！金无足赤，人无完人。由于我的学术水平有限，所写论文难免有不足之处，恳请各位老师和同学批评和指正！

附　录

work server 进程池类定义

```
//server 进程池类
template <typename T>
class work_processpool {
private:
    //使用单例模式, 将构造函数定义为私有的
    // listenfd 为监听 socket
    // connecfd 为发送心跳包的 socket
    // 默认创建 4 个子进程
    work_processpool(int listenfd, int connectfd, struct address route_addr, int process_number = 4);

public:
    //获取实列的函数
    // 监听 socket, 发送心跳包的 socket, 进程池中创建的进程数
    static work_processpool<T>* create(int listenfd, int connectfd, struct address route_addr, int process_number = 4) {
        if(!m_instance) {
            m_instance = new work_processpool<T>(listenfd, connectfd, route_addr, process_number);
        }

        return m_instance;
    }

    ~work_processpool() {
        delete [] m_sub_process;

        // 解除共享文件内存映射
        shm_unlink(shm_name);
    }

    // 启动进程池
```

```
        void run();

    private:
        // 统一事件源
        void setup_sig_pipe();

        // 父进程中运行的逻辑代码
        virtual void run_parent();

        // 子进程中运行的逻辑代码
        void run_child();

        // 发送心跳包 & 向 route server 汇报当前机器的用户数，用于 route server 决策 client 路由
        static void* report_connect(void *data);
        // void* report_connect(void *data);

    private:
        // 进程池默认允许的最大子进程数量为 8
        static const int MAX_PROCESS_NUMBER = 8;

        // 每个子进程最多能处理的客户数量
        static const int USER_PER_PROCESS = 65536;

        // epoll 最多能处理的事件数量
        static const int MAX_EVENT_NUMBER = 10000;

        // 共享内存文件映射地址 & 记录当前机器的用户连接数
        static int *share_mem;

        // 共享内存文件描述符
        int shmfd;

        // 进程池中的进程总数
        int m_process_number;
```

```
    // 子进程在池中的序号, 从 0 开始. 进程中的 m_idx 值为 -1
    int m_idx;

    // 每个进程都有一个 epoll 内核事件表, 用 m_epollfd 标识
    int m_epollfd;

    // 监听 socket
    int m_listenfd;

    // 发送心跳包的 socket
    int m_connectfd;

    // route server addr
    struct address m_route_addr;

    // 进程通过 m_stop 来决定是否停止运行
    int m_stop;

    // 保存所有子进程的描述信息
    process *m_sub_process;

    // 信号量集标识符
    int sem_id;

    // 信号量设置选项
    union semun sem_un;

    // 进程池的静态实列
    static work_processpool<T> *m_instance;
};

// 类外初始化进程池类的静态实列变量
template<typename T>
work_processpool<T>* work_processpool<T>::m_instance = NULL;
```

```
// 类外初始化进程池类的静态实列变量
template<typename T>
int* work_processpool<T>::share_mem = NULL;
```

work server 父进程实现

```
// 父进程逻辑代实现
template <typename T>
void work_processpool<T>::run_parent() {

    int ret;

    pthread_create_param *data = new pthread_create_param;
    // 发送心跳包的 socket
    data->connectfd = m_connectfd;
    // route server addr
    data->addr = m_route_addr;

    // 另开一个线程用于向 route server 汇报本机器当前的用户数
    pthread_t report_connect_pid;

    // printf("%s\n", (data->addr).ip);
    // printf("%d\n", (data->addr).port);

    ret = pthread_create(&report_connect_pid, NULL, report_connect, (void*)data);
    // pthread_join(report_connect_pid, NULL);


    // 统一信号源
    setup_sig_pipe();

    // 父进程监听 m_listenfd
    addfd(m_epollfd, m_listenfd);
```

```
epoll_event events[MAX_PROCESS_NUMBER];
int m_sub_process_counter = 0;
int new_conn = 1;
int number = 0;
ret = -1;

while(!m_stop) {
    number = epoll_wait(m_epollfd, events, MAX_PROCESS_NUMBER, -1);

    if((number < 0) && (errno != EINTR)) {
        printf("epoll failure\n");
        break;
    }

    for(int i = 0; i < number; ++i) {
        int sockfd = events[i].data.fd;

        if(sockfd == m_listenfd) {
            // 如果有新的连接到来，采用 round Robin 方式将其分配给一个子进程处理
            int i = m_sub_process_counter;
            do {
                if(m_sub_process[i].m_pid != -1) {
                    break;
                }

                i = (i + 1) % m_process_number;

            } while(i != m_sub_process_counter);

            if(m_sub_process[i].m_pid == -1) {
                m_stop = true;
                break;
            }
```

```
                m_sub_process_counter = (i + 1) % m_process_number;

                // 告诉第 i 个子进程有新客户到达
                send(m_sub_process[i].m_pipefd[0],          (char*)&new_conn,
sizeof(new_conn), 0);

                printf("send request to child: %d\n",    i);

            } else if((sockfd == sig_pipefd[0]) && (events[i].events & EPOLLIN))
{
                int sig;
                char signals[1024];

                ret = recv(sig_pipefd[0], signals, sizeof(signals), 0);

                if(ret <= 0) {
                    continue;

                } else {
                    for(int i = 0; i < ret; ++i) {
                        switch(signals[i]) {
                            case SIGCHLD:{
                                pid_t pid;
                                int stat;

                                while((pid = waitpid(-1, &stat, WNOHANG)) >
0) {\
                                    for(int i = 0; i < m_process_number; ++i) {
                                        // 如果进程池中第 i 个子进程退出
了，则主进程关闭相应的通信管道，并设置相应的 m_pid 为 -1 表示该子进
程已退出
                                        if(m_sub_process[i].m_pid == pid) {
                                            printf("child %d leave\n", i);
```

```
    close(m_sub_process[i].m_pipefd[0]);

                                        m_sub_process[i].m_pid = -1;

                                    }

                                }
                            }

                            // 如果所有子进程都退出了，则父进程也关闭
                            m_stop = true;
                            for(int i = 0; i < m_process_number; ++i) {
                                if(m_sub_process[i].m_pid != -1) {
                                    m_stop = false;
                                    break;
                                }
                            }
                            break;
                        }

                        case SIGTERM:
                        case SIGINT:{
                            // 如果父进程接收到终止信号，那么就杀死所有子进程并等待它们全部结束
                            // m_stop = true;
                            printf("kill all the child now\n");
                            for(int i = 0; i < m_process_number; ++i) {
                                int pid = m_sub_process[i].m_pid;

                                if(pid != -1) {
                                    kill(pid, SIGTERM);
                                }
                            }
                            break;
```

```
                                }

                                default:{
                                    break;
                                }
                            }
                        }
                    }

                } else {
                    break;
                }
            }
        }

        // 由创建者关闭对应的文件描述符
        close(m_epollfd);

        // 由创建者释放对应的堆内存
        delete data;
}
```

work server 子进程实现

```
// 子进程逻辑实现
template<typename T>
void work_processpool<T>::run_child() {

    setup_sig_pipe();

    // 每个子进程都通过其在进程池中的序号值 m_idx 找到与父进程通信的管道
    int pipefd = m_sub_process[m_idx].m_pipefd[1];

    // 子进程需要监听管道文件描述符 pipefd，因为父进程将通过它来通知子进程 accept 新连接
```

```
    addfd(m_epollfd, pipefd);

    epoll_event events[MAX_PROCESS_NUMBER];
    T *users = new T[USER_PER_PROCESS];
    assert(users);

    int number = 0;
    int ret = -1;

    while(!m_stop) {
        number = epoll_wait(m_epollfd, events, MAX_PROCESS_NUMBER, -1);

        if((number < 0) && (errno != EINTR)) {
            printf("epoll failure\n");
            break;
        }

        for(int i = 0; i < number; i++) {
            int sockfd = events[i].data.fd;

            // 从消息管道接收到父进程的信号，即有新客户到来
            if((sockfd == pipefd) && (events[i].events & EPOLLIN)) {
                int client = 0;

                // 从父，子进程之间的管道读取数据，并将结果保存到变量
client 中．如果读取成功，则表示有新客户连接到来
                ret = recv(sockfd, (char*)&client, sizeof(client), 0);

                if(((ret < 0) && (errno != EAGAIN)) || ret == 0) {
                    continue;

                } else {
                    struct sockaddr_in client_address;
                    socklen_t client_addrlength = sizeof(client_address);
```

```
                    // 从 socket 监听上获取新到来的客户连接
                    int connfd = accept(m_listenfd, (struct sockaddr*)&client_address, &client_addrlength);

                    if(connfd < 0) {
                        printf("errno is: %s\n", strerror(errno));
                        continue;
                    }

                    // 将读取到的新用户连接上的可读事件注册到本进程的 epoll 内核事件表上
                    addfd(m_epollfd, connfd);

                    // 模板类 T 必须实现 init 方法，以初始化一个客户连接．我们直接使用 connfd 来索引逻辑处理对象(T 类型对象)，以提高程序效率
                    users[connfd].init(m_epollfd, connfd, client_address, share_mem, sem_id);

                }

            } else if((sockfd == sig_pipefd[0]) && (events[i].events & EPOLLIN)) {
                // 处理子进程接收到的信号
                int sig;
                char signals[1024];

                ret = recv(sig_pipefd[0], signals, sizeof(signals), 0);

                if(ret <= 0) {
                    continue;

                } else {
                    for(int i = 0; i < ret; ++i) {
                        switch(signals[i]) {
```

```
                                // 子进程状态发生变化(退出或暂停信号)
                                case SIGCHLD:{
                                    pid_t pid;
                                    int stat;

                                    while((pid = waitpid(-1, &stat, WNOHANG)) >
0) {
                                        continue;
                                    }
                                    break;
                                }

                                // 终止进程信号 kill
                                case SIGTERM:
                                // 键盘输入 ctrl + c
                                case SIGINT:{
                                    m_stop = true;
                                    break;
                                }

                                default:{
                                    break;
                                }
                            }
                        }
                    }
                } else if(events[i].events & EPOLLIN) {
                    // 如果是其他可读数据，则必然是客户请求到来．调用逻辑处
理对象的 process 方法处理之
                    users[sockfd].process();

                } else {
                    continue;
                }
            }
```

```
    }

    // 文件描述符以及堆内存由哪个函数创建就应该由哪个函数释放
    delete []users;
    users = NULL;
    close(pipefd);
    close(m_epollfd);
}
```

十万级分布式 IM 系统设计

袁小龙

（1. 江西理工大学，信息工程学院，江西 赣州 341000；）

摘　要： IM 是现今互联网上最受欢迎的通讯方式。各种 IM 软件泛滥成灾，其中的佼佼者如 QQ,微信更是功能繁多,形成了独立的生态圈。但是这些 IM 软件大都只能在移动端,windows pc 端，mac pc 端工作。本文旨在设计开发一个 Linux PC 下的高并发，高可用，低时延，易扩展，支持跨局域网通信的 IM 系统。本文中运用到的计算机基础知识主要有 c/c++编程语言，socket 网络编程，epoll I/O 复用，进程间通信，线程间通信，linux 系统信号处理等。对于架构，本系统使用 C/S 模型，且在服务器端使用集群负载均衡技术，在单台服务器中使用多进程单线程模型，在客户端上使用多线程模型，在存储方面使用分布式数据库。该系统提供的服务主要有注册账号，删除账号，用户登录，用户登出，查看用户列表，用户通信等。

关键词： IM 系统；C/S 架构；负载均衡；网络编程

ABSTRACT: IM is the most popular mode of communication on the Internet today. A variety of IM software flooded into disasters, among which the best such as QQ, Wechat is a variety of functions, forming an independent ecosystem. But most of these IM software can only work on mobile, Windows PC and MAC PC. The purpose of this paper is to design and develop an IM system with high concurrency, high availability, low latency and scalability under Linux PC to support cross-LAN communication. The basic computer knowledge used in this paper mainly includes c/c++ programming language, socket network programming, epoll I/O multiplexing, inter-process communication, inter-thread communication, Linux system signal processing and so on. For architecture, the system uses C/S model, cluster load balancing technology on server side, multi-process single-thread model on single server, multi-thread model on client side and distributed database on storage side. The main services provided by the system are registration account, deletion account, user login, user logout, user list viewing, user communication and so on.

Key words: IM system; C/S architecture; load balancing; network programming

0 前言

本文旨在设计开发一个 Linux PC 下的高并发，高可用，低时延，易扩展，支持跨局域网通信的 IM 系统。在设计和实现上，本系统使用 C/S 模型加上数据库服务，其中 S 是一个两层的负载均衡结构，分为一台路由服务器和多台功能服务器，总体来看这是一个四层架构模型。Server 端的两层负载均衡结构能很大程度上的提高服务可用性和并发度，同时这是一个非常容易扩展的模型，加入功能服务器时只要简单的将功能服务器接入到路由服务器即可。另外，路由服务器和功能服务器中都使用的多进程单线程模型，可以提高多核机器的使用效率，提高服务性能，降低时延；数据库服务使用 mycat+mysql，mycat 支持读写分离，分表分库等功能，在用户量达到一定程度后能很大程度上降低数据库服务时延，提高数据库服务性能；使用点对点通信机制，同时在客户端上使用内网穿透技术，使得在不增加服务

端负载的情况下支持跨局域网通信等等。

1 需求分析

1.1 功能需求

1. 注册账号：获取一个新的系统账号，该账号由数字组成且具有唯一性。每个账号对应一个用户，一个用户可以注册多个账号；
2. 删除账号：从系统中删除一个已存在的账号，删除后该账号将失效；
3. 用户登录：用户登录即通过用户账号和密码的对应关系来验证用户身份的合法性，以确保某些操作的安全性；
4. 分页查看用户列表：获取系统中某些用户的账号，用户名，是否在线等信息。考虑到一次获取太多用户信息的话会导致操作时延过长，信息太多查看不方便等问题，采用分页的方式，默认页面容量为 20 个用户；
5. 查看指定用户是否在线：只有都登录了的用户双方才能进行通信，查看指定用户是否在线即查看指定的用户是否已经登录；
6. 向指定用户发送消息：用户输入一个 user id 向目标用户 user id 发起会话，如果成功则进入会话状态，接下来输入的消息都将发送给目标用户 user id。如果发起会话失败，则返回相关错误信息；
7. 退出聊天状态：当用户处于会话状态时，输入′quit′断开该会话，用户单向退出会话状态；
8. 退出登录：当用户处于登录状态时，输入退出登录关键字，该用户退出登录状态；
9. 用户维活：对于一个处于登录状态的用户，它需要向 IM 系统说明它处于登录状态。一旦该用户退出登录状态，IM 系统需要立即知道该消息；
10. 异常提示：当上述功能出现异常时，用户需要收到异常提示信息。

1.2 接口需求

1. 注册账号接口：用户输入用户名，密码信息，从 IM 系统中注册一个新账号，系统返回新账号的 user id；
2. 删除账号接口：用户输入用户账号和密码，从 IM 系统中删除一个已经注册的账号，删除成功返回’1’，失败则返回’0’；
3. 用户登录接口：用户输入用户名，密码信息在 IM 系统中进行登录操作即更改该用户的在线状态，登录成功返回’1’，失败则返回’0’；
4. 分页查看用户列表接口：用户输入 page（页数）值，number（每页的用户数量），在 IM 系统中查询表中第 page 页所有用户的 user id，user name，是否在线等信息（默认每页的用户数目不超过 20）；
5. 查看指定用户是否在线接口：用户输入一个 user id，从 IM 系统中查询该用户当前是否在线，若在线返回’1’，不在线返回’0’；
6. 向指定用户发送消息接口：用户输入一个 user id 向 user id 发起会话，如果成功则进入会话状态，接下来输入的消息都将发送给目标用户 user id。如果发起会话失败，则返回相关错误信息。

1.3 性能需求

5. 制定高效的私有协议；
6. 高可用，容错率高：当 IM 系统中部分服

务器异常时，仍然能提供正常服务。即便只剩下一台服务器正常工作，也能提供正常的服务；

7. 支持大量用户同时在线：达到单机支持一万用户同时在线，拥有十台以上服务器时能支持十万以 上用户同时在线；
8. 低时迟：保证在网络正常的情况下，用户之间发送’接收消息的平均时延不超过 5s，其他 IM 系统提供的服务平均时延不超过 10s；

2 系统设计

2.1 总体架构设计

本系统使用 C/S 模型加上数据库服务，其中 S 是一个两层的负载均衡结构，分为一台 route server 和多台 work server，总体来看这是一个四层架构模型。工作流程为 clien 初始化时从 route server 获取一个 work server 的地址，接着用户访问 work server 的服务。work server 和 route server 之间的数据交互关系为 work sever 每隔 10s 向 route server 汇报一次自己当

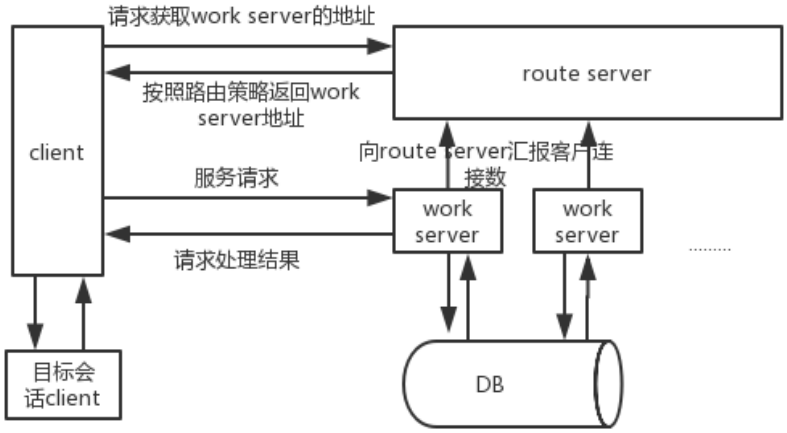

图 1 系统模型

前负责的客户数；client 和 route server 之间的数据交互关系为 client 向 route server 请求节点信息，route server 回复 client 一个当前负责客户数最少的 work server 的地址；client 和 work server 之间的交互关系为 client 向 work server 发起服务请求，如果 work server 接受到的是会话请求则返回会话目标客户登录网络地址，其他请求则进行相应的操作且返回操作结果给 client。详见图 1。

2.2 route server 设计

route server 的工作模型为多进程单线程模型。在父进程中使用 epoll 监听用户请求和 work server 的心跳包，发现了新的连接到来则按轮转法确定一个子进程并通过管道通知该进程。被通知的子进程从监听 socket 上获取连接 socket 并将其注册到自己的 epoll 内核事件表上，该 socket 连接上的后续可读事件都将由该子进程负责。同时，所有子进程共享所有 work server 的客户连接数信息，当客户端访问 route server 时 route server 返回客户连接数最少的 work server 地址给该客户端。如图 2。

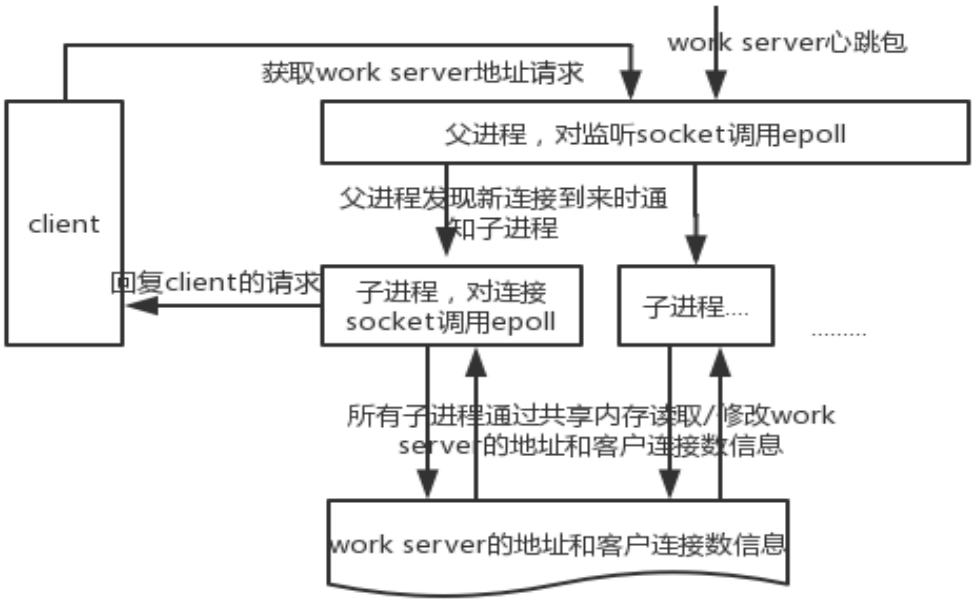

图 2 route server 模型

2.3 work server 设计

work server 的工作模型为多进程单线程模型。在父进程中另开一个子线程用于向 route server 发送心跳包，心跳包的内容为本机当前负责的客户连接数。同时在父进程中使用 epoll 监听用户请求，发现了新的连接到来则按轮转法确定一个子进程并通过管道通知该进程。被通知的子进程从监听 socket 上获取连接 socket 并将其注册到自己的 epoll 内核事件表上，该 socket 连接上的后续可读事件都将由该子进程负责。如图 3。

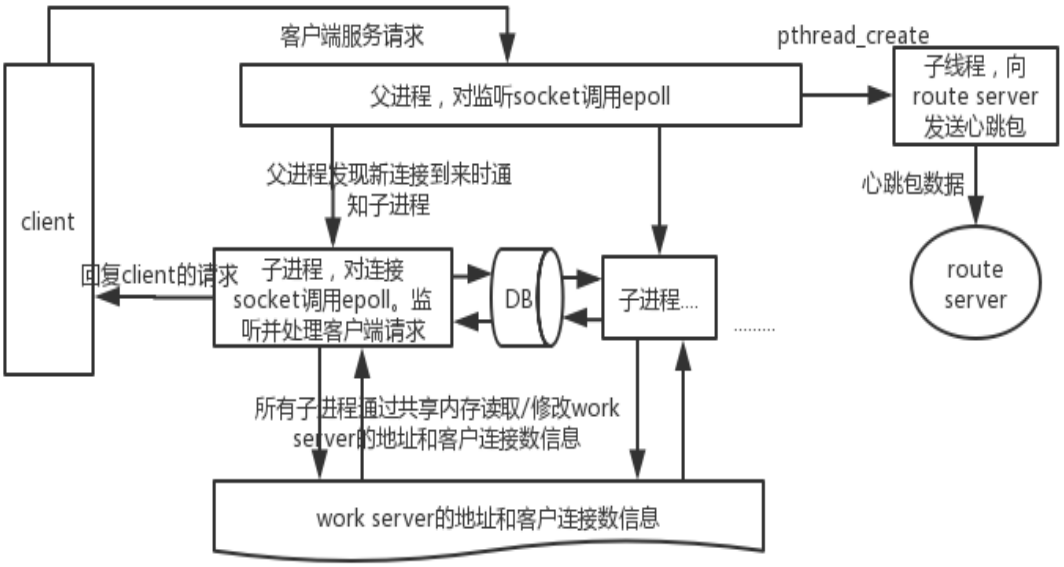

图 3 work server 模型

2.4 client 设计

client 的功能是按照定制的私有协议访问 work server 提供的服务以及和其他 client 之间发送/接收消息。为了实现同时发送消息给其他 client 和接收多个 client 的消息，即发送消息和接收消息不相互阻塞。这里使用多线程模型，开启两个线程，主线程中运行和 work server 之间的交互代码以及发送消息给其他 client 的逻辑代码，子线程用于监听其他 client 发送过来的消息。Client 工作模型见图 4。

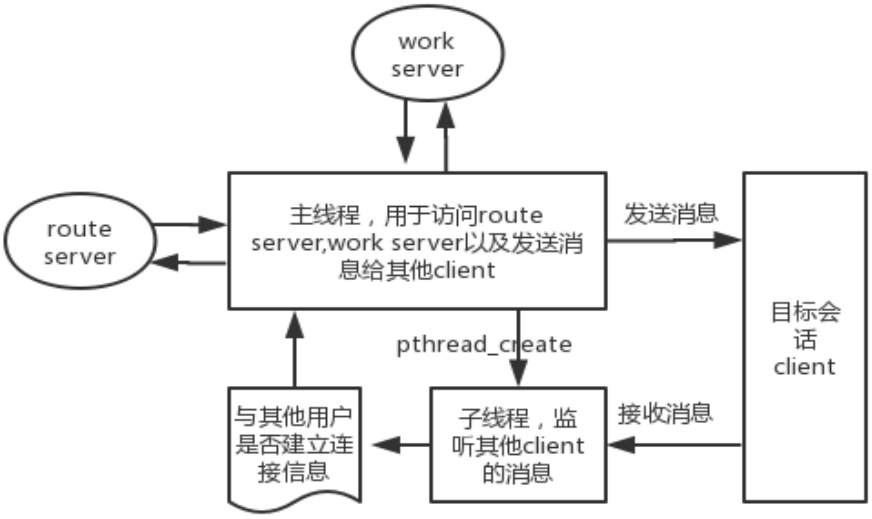

图 4 client 模型

3 总结

本系统使用的 route server，work server，client 和数据库服务四层架构，route server 管理着多台 work server。要扩展服务规模时只需要将新加入的 work server 接入到 route sever 即可，同理要减小服务规模时可以将接入 route server 的 work server 断开。Route server 和 work server 都使用的多进程单线程模型，在父进程中处理网络 I/O，在子进程中处理具体业务。数据库服务使用的 mycat+mysql。Client 中使用的多线程模型，主线程中处理访问 work server 逻辑，子线程中监听其他 client 的信息。工作流程为 clien 初始化时从 route server 获取一个 work server 的地址，接着用户访问 work server 的服务。

参考文献：

[1] 萨师煊，王珊.数据库系统概论[M].北京：高等教育出版社，2002 年 2 月：3-460.

[2] 孙涌.现代软件工程[M].北京：北京希望电子出版社，2003 年 8 月：1-246.

[3] 张海藩.软件工程导论[M].北京：清华大学出版社，2003：1-347.

[4] 徐英,谷雨.对分课堂在提升《C++程序设计》课程教学效果的实践与思考[J].教育教学论坛,2017,(01)：213-214.

[5] 徐洪智，张彬连，钟键.《C++程序设计》课程实验教学改革与探索[J].现代计算机（专业版），2017,（15）：50-53.