

数据库大作业报告

基于 muduo 开发的集群聊天服务器

作者：宋文浩

学号：2023211975

目录

[第一章 需求分析 2](#_Toc16366)

[第二章 概要设计 2](#_Toc30983)

[项目特点 3](#_Toc11391)

[第三章 详细设计 4](#_Toc21503)

[3.1. 数据库设计 4](#_Toc15664)

[3.1.1数据库详细设计 4](#_Toc12862)

[3.1.2数据库操作模块设计 6](#_Toc4644)

[3.2关键技术 7](#_Toc622)

[3.2.1基于 muduo 网络库开发网络核心模块 7](#_Toc23053)

[3.2.2使用 Nginx 的 TCP 负载均衡功能 8](#_Toc4199)

[3.2.3基于发布-订阅的服务器中间件redis消息队列 10](#_Toc20904)

[第四章 测试报告 11](#_Toc19755)

# 第一章 需求分析

本项目是我在学习c++技术栈的过程中，跟着网上的教程做的一个练手小项目，做这个报告不仅加深我对数据库的应用学习，同时让我对这个小项目有了更清晰的理解。

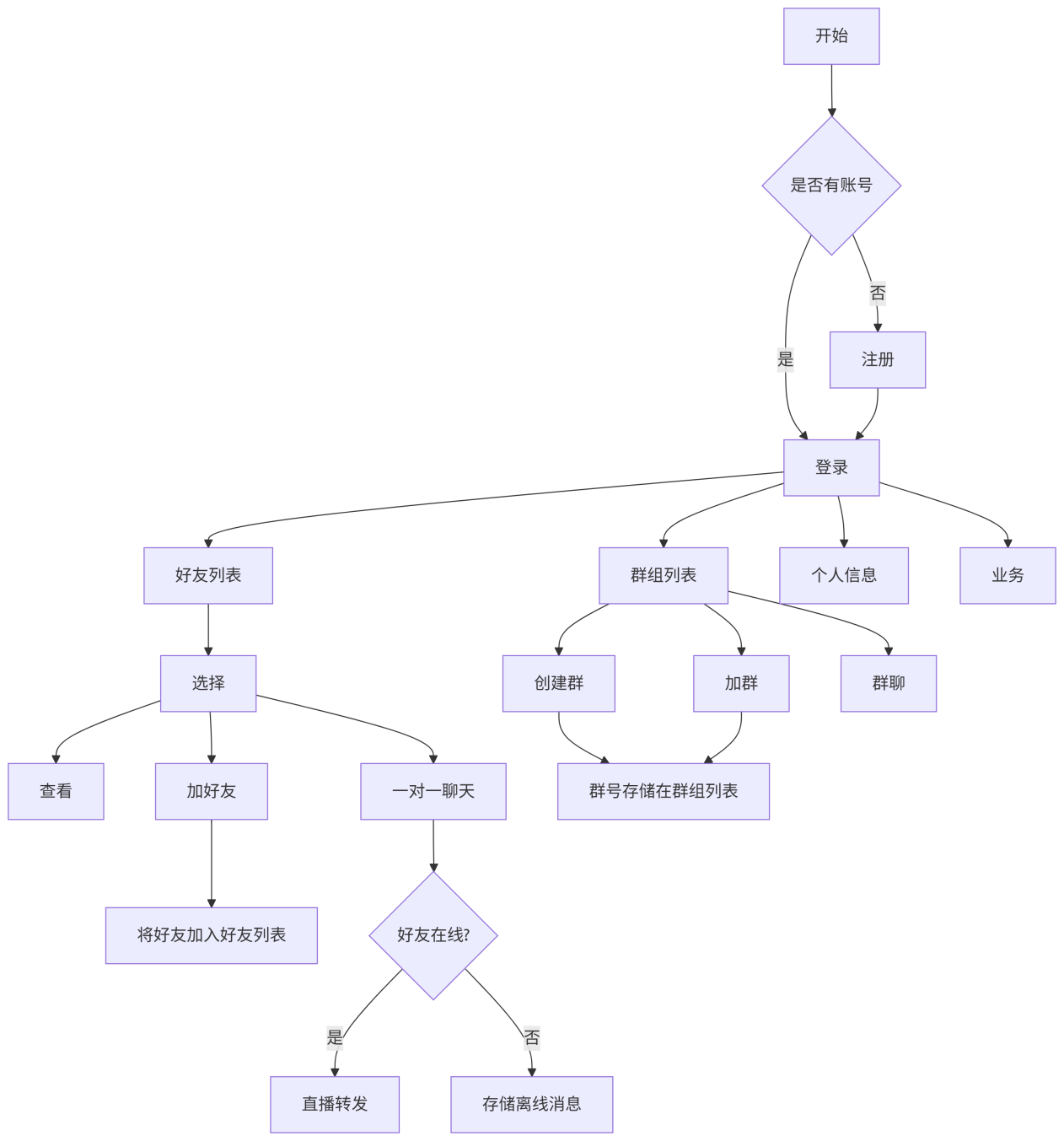
随着互联网的发展和应用需求的不断增长，构建高效、可靠、可扩展的分布式聊天服务器已成为研究的热点。该项目旨在帮助学习者理解大型分布式系统设计的基础，掌握C++在高性能网络编程中的应用，以及集群系统中的关键技术。

通过本项目，学生可以从理论到实践，学习到消息通信、负载均衡、muduo网络库编程、服务器集群、数据库设计，json序列化和反序列化等多个方面的知识。

# 第二章 概要设计

在 Linux 环境下基于 muduo 开发的集群聊天服务器。实现新用户注册、用户登录、添加好友、添加群组、好友通信、群组聊天、保持离线消息等功能。

详细业务关系图如下：



## 项目特点

* 基于 muduo 网络库开发网络核心模块，实现高效通信
* 使用第三方 JSON 库实现通信数据的序列化和反序列化
* 使用 Nginx 的 TCP 负载均衡功能，将客户端请求分派到多个服务器上，以提高并发处理能力
* 基于发布-订阅的服务器中间件redis消息队列，解决跨服务器通信难题
* 封装 MySQL 接口，将用户数据储存到磁盘中，实现数据持久化
* 基于 CMake 构建项目

# 第三章 详细设计

## 3.1. 数据库设计

### 3.1.1数据库详细设计

首先就是要解决数据的问题，作为一个聊天系统，我们的服务器端肯定要有用户的信息，比如说账号，用户名，密码等。 在****登录****的时候，我们可以查询这个表里面的信息对用户身份进行验证，在****注册****的时候，我们则可以往表里面去写入数据。

User表

| **字段名称** | **字段类型** | **字段说明** | **约束** |
| --- | --- | --- | --- |
| id | INT | 用户id | PRIMARY KEY、AUTO\_INCREMENT |
| name | VARCHAR(50) | 用户名 | NOT NULL, UNIQUE |
| password | VARCHAR(50) | 用户密码 | NOT NULL |
| state | ENUM('online', 'offline') | 当前登录状态 | DEFAULT 'offline' |

用户登录之后，首先就是进行聊天业务，我们必须要知道该用户的好友都有谁。 在****加好友****时，我们就可以往这张表里面去写入信息并在****一对一聊天****时查询这里面的信息去看好友是否在线。

****Friend表****

| **字段名称** | **字段类型** | **字段说明** | **约束** |
| --- | --- | --- | --- |
| userid | INT | 用户id | NOT NULL、联合主键 |
| friendid | INT | 好友id | NOT NULL、联合主键 |

然后便是群组业务了，群组中我们需要有一个记录群组信息的表，方便我们****创建群****时往其中去写入数据；

****AllGroup表****

| **字段名称** | **字段类型** | **字段说明** | **约束** |
| --- | --- | --- | --- |
| id | INT | 组id | PRIMARY KEY、AUTO\_INCREMENT |
| groupname | VARCHAR(50) | 组名称 | NOT NULL, UNIQUE |
| groupdesc | VARCHAR(200) | 组功能描述 | DEFAULT '' |

同时群里面肯定是有群员的，我们就需要一个记录群成员的表，我们在****加入群****的时候，把用户id写入这个表。并且在****发送群消息****的时候查询这个表由服务器向这些成员转发消息。

****GroupUser表****

| **字段名称** | **字段类型** | **字段说明** | **约束** |
| --- | --- | --- | --- |
| groupid | INT | 组id | NOT NULL、联合主键 |
| userid | INT | 组员id | NOT NULL、联合主键 |
| grouprole | ENUM('creator', 'normal') | 组内角色 | DEFAULT 'normal' |

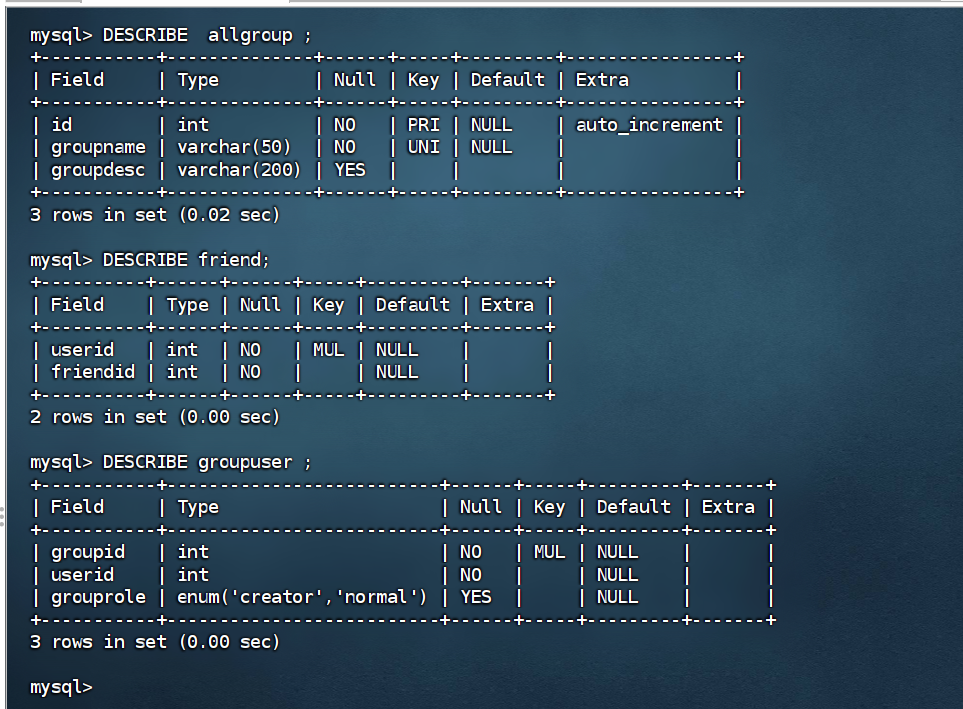
但是我们的设计目的又要存储离线消息，这就涉及到离线消息发给谁，谁发的，发的什么三个问题，所以我们又需要一个新表来存储离线消息。这样我们一旦有****离线消息****便可以往这个表里面去写入数据。

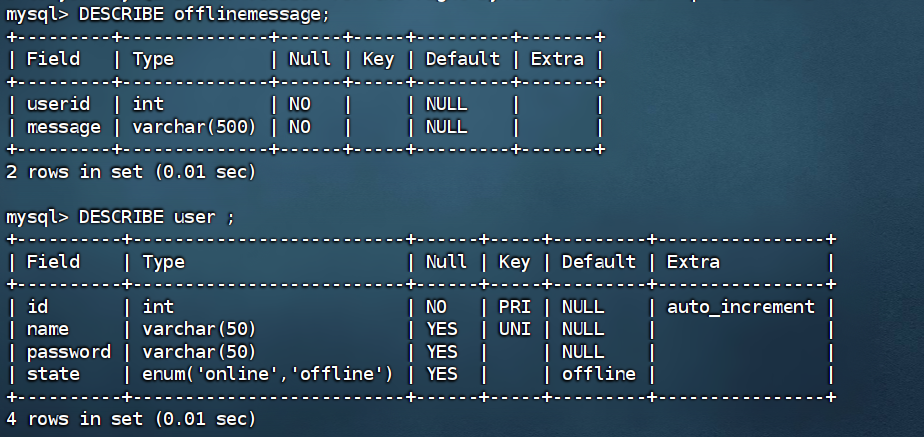
****OfflineMessage表****

| **字段名称** | **字段类型** | **字段说明** | **约束** |
| --- | --- | --- | --- |
| userid | INT | 用户id | NOT NULL |
| message | VARCHAR(50) | 离线消息（存储Json字符串） | NOT NULL |

实操数据库如图：







### 3.1.2数据库操作模块设计

**整个项目中的数据库模块可以分成三个层次**

1.数据库操作层mysql类

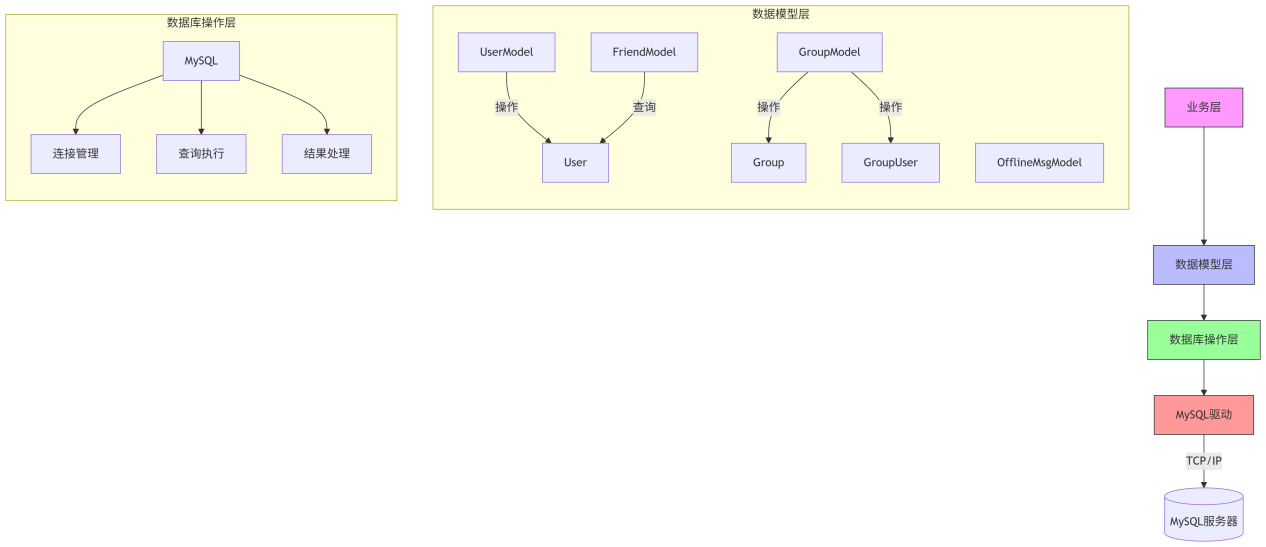
其是数据库模块中设计的最底层，为上层各个表以及其操作模块提供基础的服务.封装着对数据库的连接、查询、更新、释放连接几个操作

2.数据实体层(表的ORM类)，User,Group,GroupUser

为数据库中的每条记录提供一个标准化的容器。当从数据库查询到数据时，将数据"装入"这些盒子；当需要存储数据时，从这些盒子"取出"数据。它隔离了数据库表结构和内存对象表示。

3.数据操作模型层usermodel, groupmodel,FriendModel,OfflinMegModel

负责在业务逻辑和数据存储之间传递信息。它知道如何把业务需求（如添加好友）转化为数据库操作，也懂得如何把数据库查询结果转化为业务可用的对象。

**分层设计的意义：**

1.分离变化点

数据库变更 → 修改操作层

数据结构变更 → 修改实体层

业务逻辑变更 → 修改模型层

2.降低复杂度

每层只需关注单一职责：

操作层：如何执行SQL

实体层：如何表示数据

模型层：如何实现业务需求

3.提升复用性

多个模型复用同一个数据库操作层

4.增强可测试性

可对每层独立测试：

操作层：测试SQL执行

实体层：测试数据封装

模型层：Mock数据库测试业务

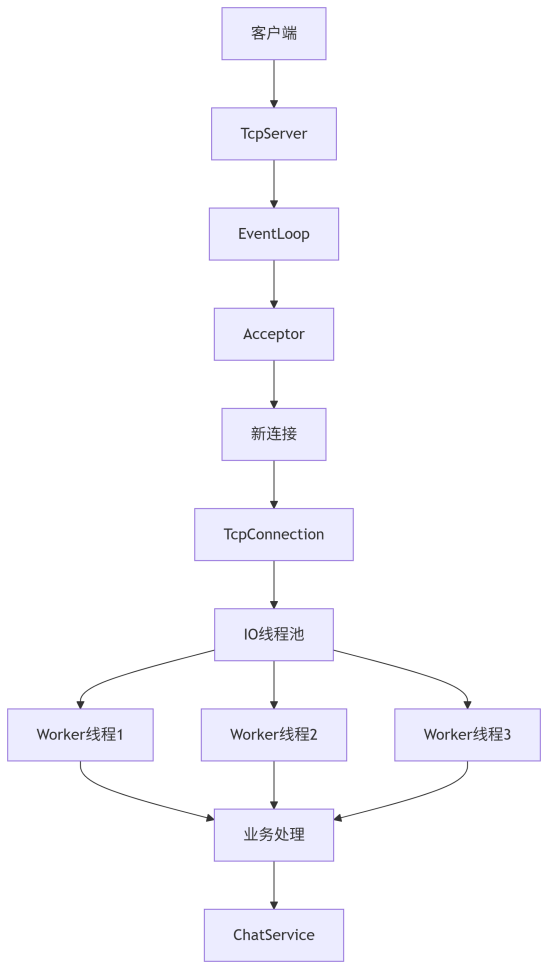
## 3.2关键技术

### 3.2.1基于 muduo 网络库开发网络核心模块

**（1）Muduo 网络库的核心优势**

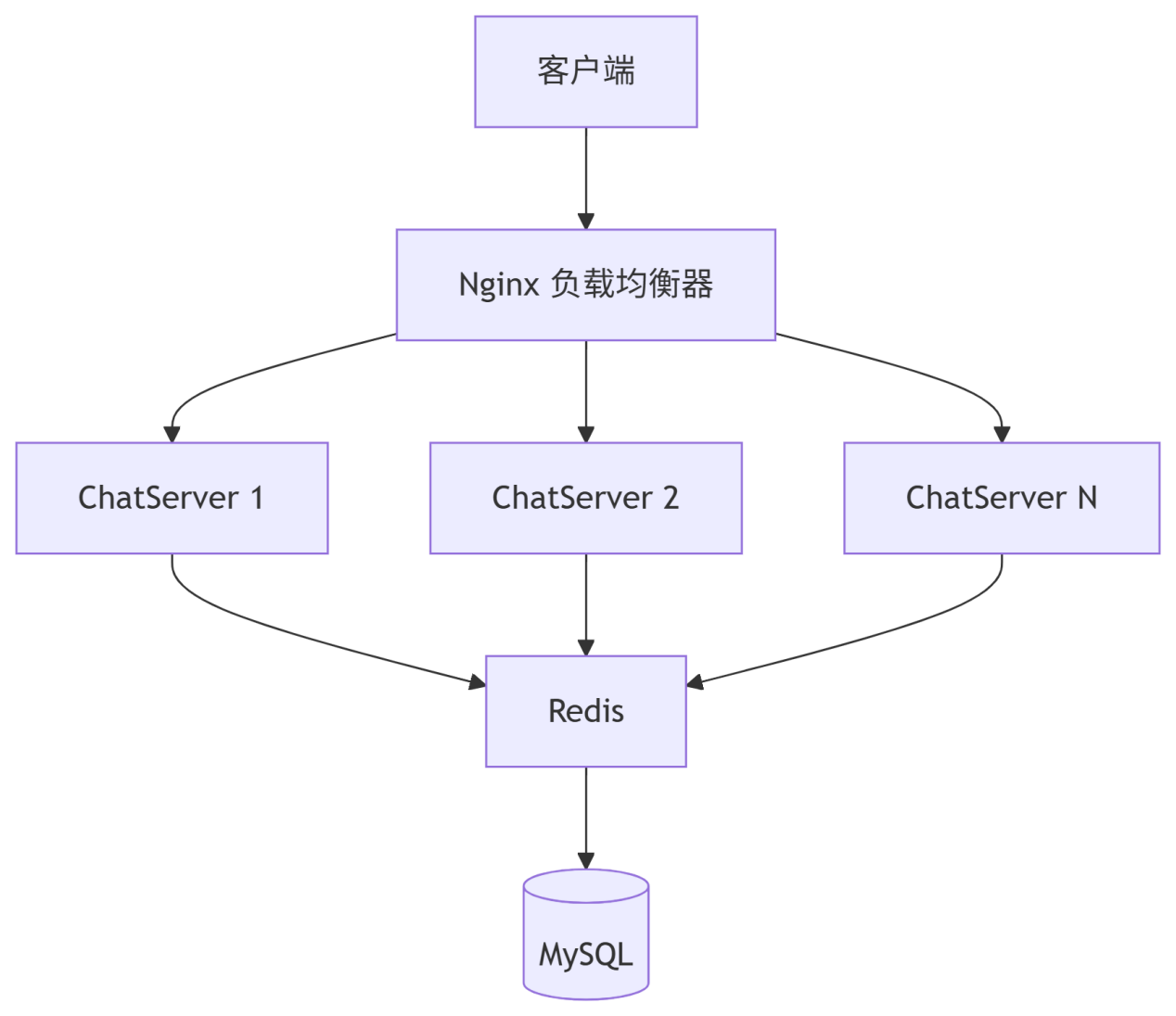
| ****特性**** | ****传统方案**** | ****Muduo 方案**** | ****性能提升**** |
| --- | --- | --- | --- |
| 并发模型 | 线程-per-connection | Reactor + 线程池 | 3-5倍 |
| I/O 效率 | 阻塞式 I/O | 非阻塞 Epoll ET | 10,000+ 连接 |
| 内存管理 | 频繁分配释放 | 对象池复用 | 减少 40% 内存碎片 |
| 线程安全 | 手动加锁 | 无锁设计+线程隔离 | 降低 30% 锁竞争 |
| 资源消耗 | 高线程开销 | 4线程处理20K连接 | CPU 降低 50% |

**（2）Muduo网络核心模块架构设计**



### 3.2.2使用 Nginx 的 TCP 负载均衡功能

**Nginx TCP 负载均衡架构设计**



**（1）使用nginx负载均衡的优点：**

1.由于网络请求规模巨大，在高并发情况下，单点压力大，容易宕机或出现延迟，所以可以通过多台服务器（如多个ChatServer实例）分担压力

2.一台服务器挂掉，客户连接不会中断，Nginx自动切换到其他健康节点

3.客户端只需连接Nginx的单一入口端口（如8000），无需关心后端用哪个服务器

4. 支持动态扩展，灵活添加节点

你可以随时将新的ChatServer加入到集群，只需新增一台服务器并在Nginx配置中定义即可

不用重启客户端或影响已有连接

5.方便维护和升级

维护节点时，只需在负载调度中临时下线，不影响整体服务

**（2）工作流程详解：**客户端连接：客户端只连接Nginx 8000端口

请求接收：Nginx接收到连接后，通过load balancing策略选择后端服务器

请求转发：Nginx将客户端的请求“代理”到被选中的ChatServer

通信交互：后端的ChatServer处理业务（登录、聊天、群组等），并返回响应

会话管理：对于每个客户端连接，Nginx只负责转发，不参与业务逻辑

### 3.2.3基于发布-订阅的服务器中间件redis消息队列

**（1）使用原因：**

客户端可能连接在不同的服务器节点（比如多台ChatServer实例），业务逻辑需要跨服务器将消息传达给正确的用户，传统方案（如：轮询、主动推送、数据库轮询等）容易带来延迟高、架构复杂、维护成本大等问题。

**（2）解决关键点：**

跨服务器的高效、可靠、异步地传递消息

维护用户状态（在线/离线）信息

用户在线时及时推送消息，离线时存入离线消息表（持久化存储）

**（3）发布-订阅模式简介**

Redis的Pub/Sub机制是一种消息中间件模式，

发布者（Publisher）：向某个“频道”发布消息

订阅者（Subscriber）：针对某个“频道”订阅，接收来自该频道的消息

特点：

高性能（基于Redis底层的IO模型）

异步非阻塞通信

简单高效，支持多客户端、多订阅

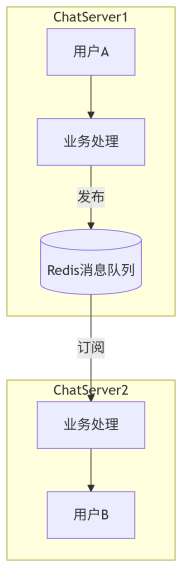
**（4）工作流程**

持续监听订阅连接

解析消息格式（["message", "频道", "内容"]）

提取频道ID和消息内容

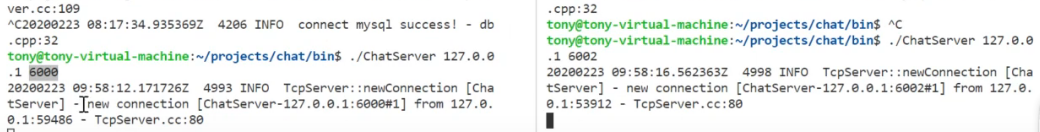
通过回调函数传递给业务层

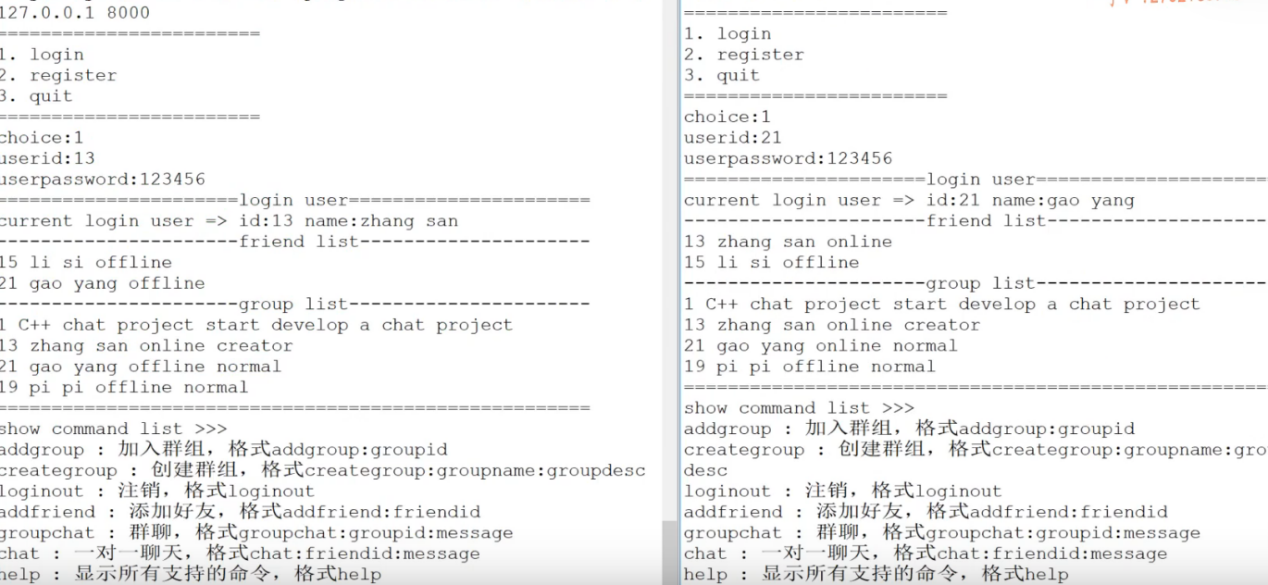


# 测试报告

1. 首先启动两个服务器，端口分别为6000和6002，再启动两个客户端连接8000端口（nginx监听端口）

如图 这是两个服务端



这是两个客户端  


可以看到两个客户端成功连接到服务端，并且通过负载均衡，给每一台服务器都分配了一个用户

1. 如图，登录到两台不同服务器的客户进行跨服务通信



1. 项目总结

本项目基于C++技术栈与muduo网络库开发了一款集群聊天服务器，结合MySQL数据库、Nginx负载均衡及Redis消息队列等技术，完成了用户注册、登录、好友通信、群组聊天及离线消息存储等核心功能。通过本次实验，不仅加深我对数据库的应用学习，同时让我对这个小项目有了更清晰的理解。

在数据库设计方面，项目通过分层架构实现了数据的高效管理。用户信息表（User）、好友关系表（Friend）、群组表（AllGroup）及离线消息表（OfflineMessage）的设计逻辑清晰，满足了聊天系统的核心业务需求。数据库操作层（MySQL类）、数据实体层（ORM类）及模型层（UserModel等）的分层设计有效降低了模块间的耦合度，提升了代码的可维护性与扩展性。例如，通过将数据库变更与业务逻辑解耦，未来若需调整数据表结构或更换数据库类型，仅需修改操作层代码，而无需改动上层业务逻辑。

在技术实现上，项目充分结合了高性能网络编程与分布式系统的关键技术。基于muduo网络库的Reactor模式，通过非阻塞 I/O 和线程池管理，使得服务器能够高效处理大量并发连接；Nginx的TCP负载均衡功能实现了客户端请求的动态分配，避免了单点故障问题，并通过“健康节点检测”机制提升了系统的容错能力；Redis的发布-订阅模式则解决了跨服务器通信的难题，通过异步消息传递实现了用户状态同步与消息转发的低延迟处理。这些技术的综合应用，使系统在高并发场景下依然保持了良好的稳定性与扩展性。

测试阶段，通过模拟多客户端连接与跨服务器通信，验证了系统功能的完整性。例如，两个客户端通过Nginx负载均衡分别连接到不同服务器实例后，成功实现了跨服务器的消息传递，体现了Redis消息队列在跨节点通信中的核心作用，证明了数据库与消息队列协同工作的可靠性。

综上所述，本项目不仅实现了预期的数据库功能与系统架构目标，也为后续学习c++技术栈与Linux技能提供了宝贵的实践经验。