

LiteCoze 项目文档

技术小组 se3-se1604 小组

小组成员 杜虹朋 王婷 万坤 汪惠玲 范珈

指导教师 龚伟

起止时间 2018 年 10 月 ~ 2019 年 7 月

重庆师范大学软件工程系
[git@github.com:se1604/LiteCoze.git](https://github.com/se1604/LiteCoze)

摘要

由于目前 QQ 过于繁琐，所以我们决定开发一款轻聊版的 QQ（LiteCoze）。我们希望用户能通过我们的软件轻松、快速的和朋友进行交流，实时分享自己的动态。该软件主要利用 Boost 库、Qt 开发框架等工具完成。本文将详细阐述用户需求、用况描述、数据库设计、整个系统框架等内容。

关键词：实时、用况、boost 库、架构设计、需求分析

目录

第1章 愿景文档.....	6
1.1. 问题陈述.....	6
1.问题一.....	6
1.2. 涉众与用户.....	6
1.涉众.....	6
2. 用户.....	6
1.3.关键涉众和用户需求.....	7
1.4 产品概述.....	7
1.产品定位陈述.....	7
2.完整的产品概述.....	8
1.5.特性.....	8
1.6.其他产品需求.....	9
第2章 用况模型.....	9
2.1. 术语表.....	9
2.2. LiteCoze 的主要用况.....	9
2.3.用况描述——私聊.....	10
1.简要描述.....	10
2.用况图.....	10
3.前置条件.....	10
4.基本流.....	10
5.备选流.....	12
6.子流.....	12
7.操作说明.....	13
2.4.用况描述——群聊.....	15
1.简要描述.....	15
2.用况图.....	16
3.前置条件.....	16
4.基本流.....	16
5.备选流.....	17
6.子流.....	18
7.操作说明.....	19
2.5.用况描述——管理动态.....	22
1.简要描述.....	22
2.用况图.....	22
3.前置条件.....	22
4.基本流.....	22
5.备选流.....	23
6.子流.....	23
7.操作说明.....	24
2.6 包含用况——登录.....	25
1.简要描述.....	25
2.用况图.....	25
3.前置条件.....	25
4.基本流.....	25

5.子流.....	26
6.备选流.....	26
第三章 健壮性分析.....	26
3.1 通信图——私聊.....	26
3.2 通信图——群聊.....	27
3.3 通信图——登录.....	27
3.4 通信图——注册.....	28
3.5 通信图——加好友.....	29
3.6 通信图——查找好友.....	29
3.7 通信图——接受添加群.....	30
3.8 通信图——添加群.....	30
3.9 通信图——查找群.....	31
3.10 通信图——初始化系统.....	32
3.11 通信图——接受好友请求.....	32
第四章 交互模型分析.....	33
4.1 顺序图——注册.....	33
4.2 顺序图——登录.....	34
4.3 顺序图——加群.....	35
4.4 顺序图——接受加群请求.....	35
4.5 顺序图——加好友.....	36
4.6 顺序图——群聊.....	37
4.7 顺序图——找群.....	37
4.8 顺序图——私聊.....	38
4.9 顺序图——初始化系统.....	39
4.10 顺序图——查找好友.....	39
4.11 顺序图——接受加好友.....	40
第五章 状态机分析.....	41
第六章 系统架构设计.....	41
6.1 系统类模型.....	41
6.2 系统架构.....	42
1.开发技术和平台：.....	42
2.C/S 结构：.....	42
3.MVC 框架.....	43
4.网络协议的选择——TCP/IP 协议.....	43
5.Boost 库.....	43
6.数据传输选择——Json.....	43
6.3.系统实体类图详细设计.....	44
第七章 数据库设计.....	45
7.1 使用 Mariadb 数据库工具，整体表关系.....	45
7.2 将所有实体类做成相应的表.....	45
1.网民类.....	45
2.私聊房间类.....	46
3.netizen_privateroom 表.....	46
4.群聊房间类.....	47
5.netizen_grouproom 表.....	47

6.群成员表.....	48
7.消息类.....	48
第八章 系统详细设计.....	48
8.1 抽象类设计.....	48
8.2 服务端交互设计.....	50
8.3 客户端交互设计.....	50
第九章 实现.....	50
9.1tcp 数据传输.....	50
9.2 数据管理.....	51
9.3 网络通信.....	52
第十章 运行部署.....	53
10.1 登录界面.....	53
10.2 注册界面.....	53
10.3 聊天.....	54
10.4 查找好友并添加.....	54
10.5 查找群并添加.....	55
10.6 处理添加请求.....	55
参考文献.....	55

第 1 章 愿景文档

1.1. 问题陈述

1.问题一

要素	描述
问题	目前的 QQ 功能过于繁琐
影响	追求软件简洁的用户
结果	占据大量内存
优点	使之更为简洁，节省空间

1.2. 涉众与用户

1.涉众

涉众	涉众类型	简要描述
项目发起人	发起者	项目最初的提出者
支持人员	开发人员	提供技术支持
编码员	开发人员	项目的实现
系统维护人员	开发人员	修复项目 bug，提高性能
消费者	客户	购买软件
标准组织	权威	约束软件开发过程
监管机构	权威	监管网络安全
分析人员	开发人员	需求分析
设计师	开发人员	设计软件
测试人员	开发人员	测试软件
项目经理	开发人员	管理整个开发过程
评估员	客户	评估软件价值

2. 用户

年轻人	用户	使用一些较复杂的，新颖的功能
-----	----	----------------

中老年人	用户	主要是使用简单的功能，很难学会一些复杂的操作
初级管理员	用户	做一些简单的审核工作
高级管理员	用户	完成权限更高的审核工作

1.3.关键涉众和用户需求

关键涉众：消费者

用户需求：

- 方便联系朋友
- 能够通过发图片，语音交流
- 聊天界面干净，聊天方式简捷
- 没有隐形（强制性）消费
- 不能泄漏自己的隐私信息给其他人
- 便于操作
- 易于管理
- 查看聊天记录
- 和朋友们实时通信
- 相互之间发送文件
- 在空间中发布动态

1.4 产品概述

1.产品定位陈述

for	社交群体
who	希望系统更简洁
the	一款聊天交友软件
that	界面简洁，操作简单，节省运行空间
Unlike	现有 QQ
Our Product	界面简约，操作简单

2.完整的产品概述

LiteCoze 是全新的“轻聊的 QQ”，它是一款基于网络的实时通讯软件。它支持在线聊天交友（文字、语音、图片聊天），空间动态等功能。界面清爽简洁，操作简单。

1.5.特性

Identifier	Description	Priority
特性#1	登录	must
特性#2	群聊	must
特性#3	私聊	must
特性#4	设置主页	should
特性#5	实名认证	must
特性#6	简洁	should
特性#7	注册	must
特性#8	语音聊天	should
特性#9	添加群	must
特性#10	添加好友	must
特性#11	发布个人动态	should
特性#12	评论动态	should
特性#13	转发动态	should
特性#14	找回密码	must
特性#15	系统安全	must
特性#16	发送图片	should

1.6.其他产品需求

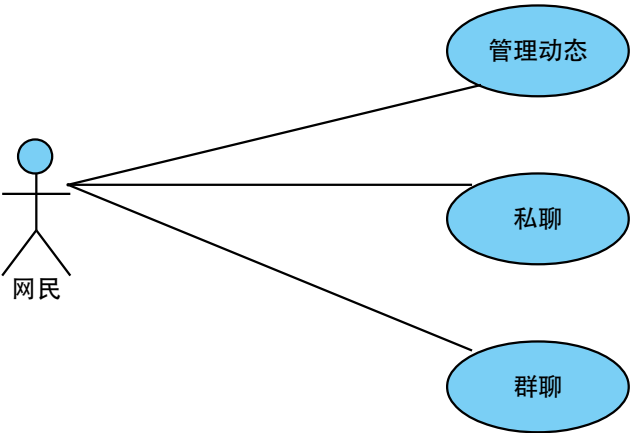
软件具有良好的性能，便于管理，运营，维护。

第2章 用况模型

2.1. 术语表

终端	登录使用聊天服务的软件客户端
好友	网民的所有联系人,也是一个网民
资料信息	头像，昵称等信息
在线好友	正在使用该系统的网民的好友
高亮	表示系统会加亮网民的在线好友
聊天消息	网民在聊天过程中发送的文字、图片、语音内容
回显	网民发送的消息显示到自己的终端上
客服	给网民回答一些基本问题
帐号（ID）	系统为区分网民所分配的唯一标识
登录信息	包括网民的帐号密码信息
动态	网民编辑文字图片等发在空间让好友看见
群	提供多人聊天的房间
成员	群里的网民
昵称	网民的名字
敏感词	含有反党反社会、涉黄毒赌的言论

2.2. LiteCoze 的主要用况



管理动态

该用况描述了网民浏览、发布、转发、评论动态。

私聊

该用况描述了网民使用该系统进行一对一聊天的交互过程。

群聊

该用况描述了一群网民同时使用该系统进行聊天的交互过程。

2.3.用况描述——私聊

1.简要描述

该用况描述了网民使用该系统进行一对一聊天的交互过程。

2.用况图



3.前置条件

- 1.系统已启动并可以和终端通过网络连通。
- 2.网民已注册拥有登录信息。

4.基本流

- 1.网民启动一个终端，启动用况。

{登录}

- 2.包含用况 **登录** 描述了网民登录到该系统。

{查询网民好友信息}

- 3.系统查询网民的所有好友的资料信息。

{显示网民的好友列表}

4.系统显示网民的好友列表，其中高亮显示在线好友。

{获取网民的离线消息}

5.系统获取该网民的离线消息，并提示网民有离线消息。若网民查看离线消息则跳至{系统显示之前的聊天消息}；若不查看则继续执行下一步。

{网民选择好友}

6.网民选择其中一个在线好友作为聊天对象与其聊天。

{读取本地聊天消息}

7.系统读取保存在当前设备上的与该好友的聊天消息。

{系统显示之前的聊天消息}

8.系统显示与该好友之前的聊天消息。

{编辑聊天消息}

9.网民输入聊天消息，并请求发送。

{回显聊天消息}

10.系统回显聊天消息。

{发送聊天消息}

11.系统将网民发送的聊天消息保存，之后发送到该好友启动的终端。

{接收消息}

12.该好友的终端接收来自系统发送的聊天消息,并保存到该好友的设备上。

{显示聊天消息}

13.系统提示该好友有新的聊天消息，并显示到其终端上。

{相互通信}

14.两个网民之间相互发送消息通信，重复{编辑聊天消息}到{显示聊天消息}的过程，直到通信结束。

{用况终止}

15.用况终止。

5.备选流

A1.在{系统发送聊天消息}处如果发送失败

(1) 如果未连接到网络，系统提示网民未联网。

(2) 如果网络连接正常，尝试重新发送消息

a.系统提示发送消息失败。

b.网民请求重新发送。

c.回到{发送聊天消息}。

(3) 如果网络连接正常，消息发送失败，则将遇到的问题向客服求助。

6.子流

S1.添加好友，系统中网民可能在登录后随时请求添加好友

{搜索好友}

1.网民通过 ID 或昵称搜索好友。

{申请添加好友}

2.网民的终端发送添加好友申请给系统。

{发送添加好友请求}

3.系统将添加好友信息发送给另一网民。

{接受添加好友请求}

4.网民接受系统发出的添加好友申请，并反馈给系统。

{存储好友关系}

5.系统建立两个网民的关系，并在网民的好友列表中增加该好友。

6.回到{网民选择好友}

S2 .删除好友，系统中网民可能在登录后随时请求删除好友

{删除好友}

1.网民向系统发出删除某好友的请求。

{系统解除好友关系}

2.系统接受请求，解除网民的好友关系。

3.回到{网民选择好友}

7.操作说明

Context:Message

1.Operation specification: getLocalMessage(Netizen friend,Netizen individual)

Operation intent: 返回网民之间聊天的本地消息

Operation signature: Message::getLocalMessage(Netizen friend,Netizen individual)

time:string content:string

Logic description:

pre:self→exists()

friend is valid

individual is valid

post:return time:string content:string

Other operation called: none

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

2.Operation specification: createNewMessage(Netizen friend,Netizen individual,String content)

Operation intent: 创建一个新的消息实体

Operation signature: Message::createNewMessage(Netizen friend,Netizen individual,String content)

Logic description:

pre:self→exists()

friend is valid

individual is valid

content is not null

post:a newMessage exists

the newMessage is linked to the friend object and individual object

self→time=setCurrentTime

Other operation called: Message.echoMessage()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

Context:Netizen

1.Operation specification: getFriends()

Operation intent: 得到所有好友信息

Operation signature: Netlizen::getFriends()

Friends:Netizen

Logic description:

pre:self→exists()

post:result = self.friends

Other operation called: Netizen.getFriendInformation()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

2.Operation specification: getFriendInformation()

Operation intent: 返回好友的信息

Operation signature: Netlizen::getFriendInformation()

headPortrait:Picture ifonline:bool id:string nickname:string

Logic description:

pre:self→exists()

post:result headPortrait:Picture ifonline:bool id:string nickname:string

Other operation called: none

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

3.Operation specification: getLocalMessages()

Operation intent: 得到与好友的本地聊天消息

Operation signature: Netlizen::getLocalMessage()
messages:Message

Logic description:

pre: self→exists()

post: result = self.localMessages

Other operation called: Netizen::getLocalMessage()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

4.Operation specification: addNewMessage(Netizen friend, string content)

Operation intent: 创建一个新消息

Operation signature: Netlizen::addNewMessage(Netizen friend, string content)
content:string friend:Netizen

Logic description:

pre: self→exists()

post: none

Other operation called: Netizen::createNewMessage()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

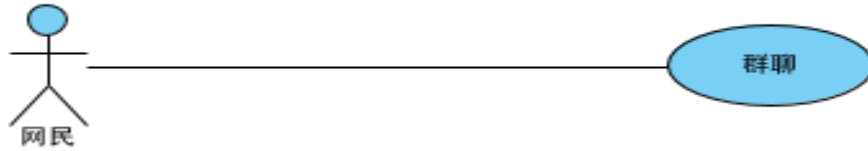
Non-functional requirements: none defined

2.4.用例描述——群聊

1.简要描述

该用例描述了一群网民同时使用该系统进行聊天的交互过程。

2.用况图



3.前置条件

- 1.系统已启动并可以和终端通过网络连通。
- 2.网民已注册并拥有登录信息。

4.基本流

- 1.网民启动一个终端，启动用况。

{登录}

- 2.包含用况 **登录** 描述了网民登录到该系统。

{查询网民群信息}

- 3.系统查询网民的所有群的资料信息。

{显示网民的群列表}

- 4.系统显示网民的群列表。

{获取网民的离线消息}

- 5.系统获取该网民的离线消息，并提示网民有离线消息。若网民查看离线消息则跳至{系统显示之前的聊天消息}；若不查看则继续执行下一步。

{网民选择群}

- 6.网民选择其中一个群作为聊天对象与其聊天。

{读取本地聊天消息}

7.系统读取保存在当前设备上该群的聊天消息。

{系统显示之前的聊天消息}

8.系统显示该群之前的聊天消息。

{编辑聊天消息}

9.网民输入聊天消息，并请求发送。

{回显聊天消息}

10.系统回显聊天消息。

{发送聊天消息}

11.系统将网民发送的聊天消息保存，之后发送到拥有该群的每个网民的终端。

{接收消息}

12.该群的每个网民的终端接收来自系统发送的聊天消息,并保存到该网民的设备上。

{显示聊天消息}

13.系统提示该网民有新的聊天消息，并显示到其终端上。

{相互通信}

14.多个网民之间相互发送消息通信，重复{编辑聊天消息}到{显示聊天消息}的过程，直到通信结束。

{用况终止}

15.用况终止。

5.备选流

A1.在{系统发送聊天消息}处如果发送失败

(1) 如果未连接到网络，系统提示网民未联网。

(2) 如果网络连接正常，尝试重新发送消息

- a.系统提示发送消息失败。
- b.网民请求重新发送。
- c.回到{发送聊天消息}。

(3) 如果网络连接正常，消息发送失败，则将遇到的问题向**客服**求助。

6.子流

S1.加入群，系统中网民在登录后可以请求加入群。

{搜索群}

1.网民通过 ID 或昵称搜索群。

{申请加入某个群}

2.网民发送加入该群的申请给系统。

{系统发送申请给群主}

3.系统将加群的申请发送给该群的群主。

{群主处理申请信息}

4.群主处理系统发来的申请信息，若同意加入，则发送同意信息给系统，进行下一步。

{系统添加网民到群}

5.系统将该网民添加到该群中，建立网民与群的联系。

{显示群给网民}

6.系统在网民的群列表增加该群。

7.回到{网民选择群}

S2.退出群聊，在{网民选择群}这个扩展点中，网民可能选择群退出

{网民选择退出群聊}

1.网民选择退出群，并发送退出群请求给系统。

{系统解除网民和群之间的关系}

2.系统删除网民列表的该群并删除该群里网民的信息。

3.回到{网民选择群}

S3.群主添加成员,在{网民选择群}这个扩展点中，群主有权限添加群成员

{发送添加请求}

1.群主选择好友发送邀请，并向系统发送添加群成员请求。

{系统发出邀请}

2.系统响应群主请求，向网民发送加群邀请。

{网民收到邀请}

3.网民接收到邀请，若同意，则请求系统进群，进行下一步。

{系统处理同意请求}

4.系统建立网民与该群之间的关系，在好友的群列表增加该群，并在该群添加好友的信息。

S4.群主删除成员，在{网民选择群}这个扩展点中，群主有权限删除群成员

{选择删除成员}

1.群主选择删除的网民，并向系统发出删除请求。

{系统删除该网民}

2.系统解除群与网民的关系，删除网民群列表的该群并删除该群里该网民的信息。

S5.创建群，系统中网民登录后可能申请创建群

{网民创建群}

1.网民向系统发出创建一个群请求。

{系统创建群聊}

2.系统创建一个群，并把网民放入该群。

3.回到{网民选择群}

7.操作说明

Context:Message

1.Operation specification: getGroupLocalMessage(Group group,Group individual)

Operation intent: 返回群聊天的本地消息

Operation signature: Message::getGroupLocalMessage(Group group,Group individual)

time:string content:string

Logic description:

pre:self→exists()

group is valid

individual is valid

post:return time:string content:string

Other operation called: none

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

2.Operation specification: createNewMessage(Netizen friend,Netizen individual,String content)

Operation intent: 创建一个新的消息实体

Operation signature: Message::createNewMessage(Group group,Group individual,String content)

Logic description:

pre:self→exists()

group is valid

individual is valid

content is not null

post:a newMessage exists

the newMessage is linked to the group object and individual object

self→time=setCurrentTime

Other operation called: Message.echoMessage()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

Context:Group

1.Operation specification: getGroups()

Operation intent: 得到所有群信息

Operation signature: Group::getGroups()
Groups:Group

Logic description:

pre:self→exists()

post:result = self.groups

Other operation called: Group.getGroupInformation()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

2.Operation specification: getGroupInformation()

Operation intent: 返回群的信息

Operation signature: Group::getGroupInformation()
headPortrait:Picture ifonline:bool id:string nickname:string

Logic description:

pre:self→exists()

post:result headPortrait:Picture ifonline:bool id:string nickname:string

Other operation called: none

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

3.Operation specification: getGroupLocalMessages()

Operation intent: 得到群的本地聊天消息

Operation signature: Group::getGroupLocalMessage()
messages:Message

Logic description:

pre: self→exists()

post: result = self.localMessages

Other operation called: Group::getGroupLocalMessage()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

4.Operation specification: addNewMessage(Group group, string content)

Operation intent: 创建一个新消息

Operation signature: Group::addNewMessage(Group group, string content)
content:string group:Group

Logic description:

pre: self→exists()

post: none

Other operation called: Group::createNewMessage()

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

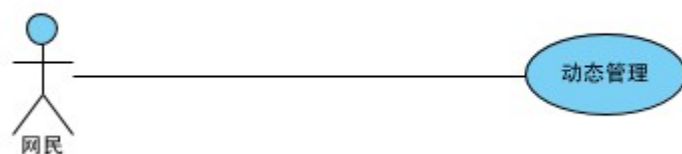
Non-functional requirements: none defined

2.5.用况描述——管理动态

1.简要描述

该用况描述了网民浏览、发布、转发、评论动态。

2.用况图



3.前置条件

1.系统已启动并可以和终端通过网络连通。

2.网民已注册拥有登录信息。

4.基本流

1.网民启动一个终端，启动用况。

{登录}

2.包含用况 **登录** 描述了网民登录到该系统。

{查询网民好友动态信息}

3.系统查询网民的所有好友的动态信息。

{显示网民的好友动态}

4.系统刷新显示网民的好友动态。

{用况终止}

5.网民退出空间，用况终止。

5.备选流

A1 在{显示网民的好友动态}处如果显示失败

(1) 如果未连接到网络，系统提示网民未联网。

(2) 如果网络连接正常，尝试刷新

(3) 如果刷新后依旧不能查看，则将遇到的问题向客服求助。

6.子流

S1.发布动态，在{显示网民的好友动态}后可以选择发布动态

{编辑动态}

1.网民编辑动态，并请求发送。

{发送动态}

2.网民的终端将动态发送给系统。

{审核动态}

3.系统审核动态内容是否有敏感词，若存在敏感词，则系统发送提示信息给网民；若没有敏感词则进行下一步。

{发布动态}

4.系统将该动态发送到与该网民是好友关系的其他所有网民的终端上。

S2.转发动态

{选择动态}

1.网民选择一条好友已发布的动态。

{编辑动态}

2.网民编辑转发的动态，并请求系统转发

(1) @好友。

(2) 添加转发理由。

{审核动态}

3.系统审核动态内容是否有敏感词,若存在敏感词,则系统发送提示信息给网民;若没有敏感词则进行下一步。

{发布动态}

4.系统将该动态发送到与该网民是好友关系的其他所有网民的终端上。

5.4.评论动态

{评论动态}

1.网民对好友已发布的动态进行评论,并请求发送。

{发评论请求}

2.网民的终端将评论发送给系统。

{发布评论}

3.系统将该评论发送到与该网民是好友关系的另一网民的终端上。

7.操作说明

Context:Dynamic

1.Operation specification: getDynamics()

Operation intent: 得到所有动态信息

Operation signature: `Dynamic::getDynamics()`
 dynamics: `Dynamic`

Logic description:

```
pre:self→exists()
```

```
post:result = self.dynamics
```

Other operation called: `Dynamic.getDynamicInformation()`

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

Non-functional requirements: none defined

2.Operation specification: getDynamicInformation()

Operation intent: 返回动态的信息

Operation signature: Dynamic::getDynamicInformation()

headPortrait:Picture ifonline:bool id:string nickname:string

Logic description:

pre:self→exists()

post:result headPortrait:Picture ifonline:bool id:string nickname:string

Other operation called: none

Events transmitted to other objects: none

Attributes set: none

Response to exception: none defined

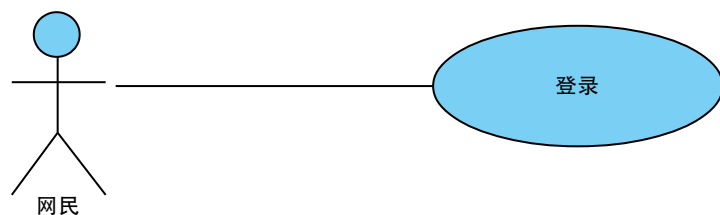
Non-functional requirements: none defined

2.6 包含用况——登录

1.简要描述

其他用况包含了此用况，本用况用来验证网民是否有权登录系统。

2.用况图



3.前置条件

系统已启动并可以和终端通过网络连通。

4.基本流

{请求登录}

1.系统要求网民输入登录信息。

2.网民输入登录信息后请求登录。

{验证登录信息}

3.系统验证登录信息

1) 如果验证成功，则网民成功登录系统。

2) 如果验证失败

- a.如果**账号**正确但密码错误，则系统提示网民密码错误，请输入正确的密码后重新**登录**，回到第 1 步。
- b.如果账号不存在，则系统提示**网民该帐号**不存在，若想使用系统则需进行**账号注册**。

5.子流

{请求注册}

- 1.系统要求**网民**输入**注册信息**。
- 2.**网民**输入注册信息后请求注册。

{注册成功}

- 3.注册成功后进入{请求登录}。

6.备选流

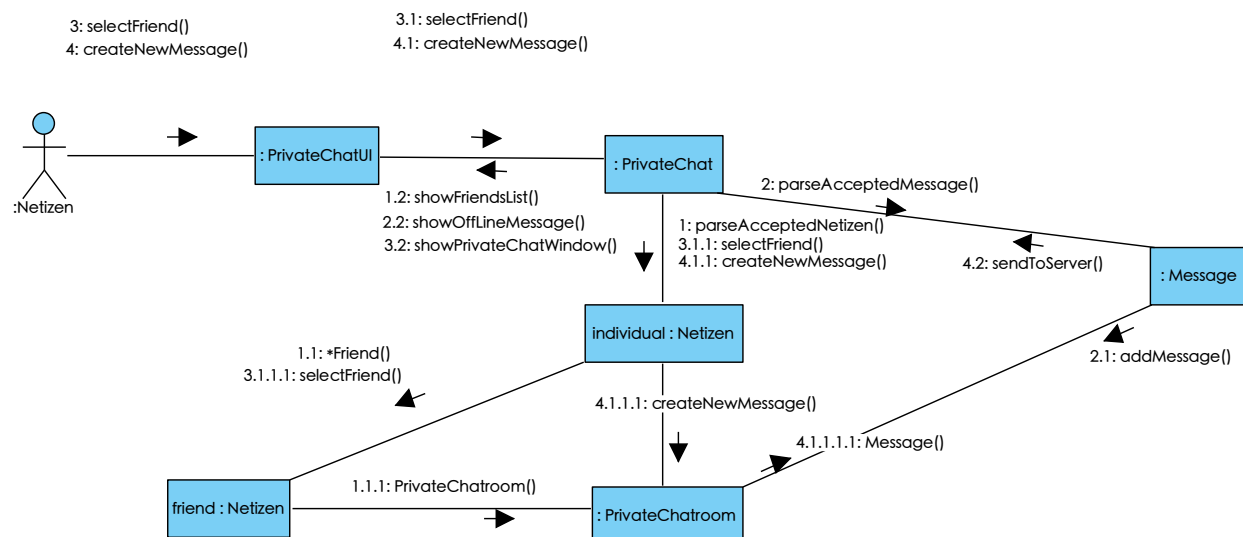
在{注册成功}处如果失败

- (1) 如果未连接到网络，系统提示网民未联网。
- (2) 如果网络连接正常，尝试刷新
- (3) 如果刷新后依旧不能注册，则将遇到的问题向**客服**求助。

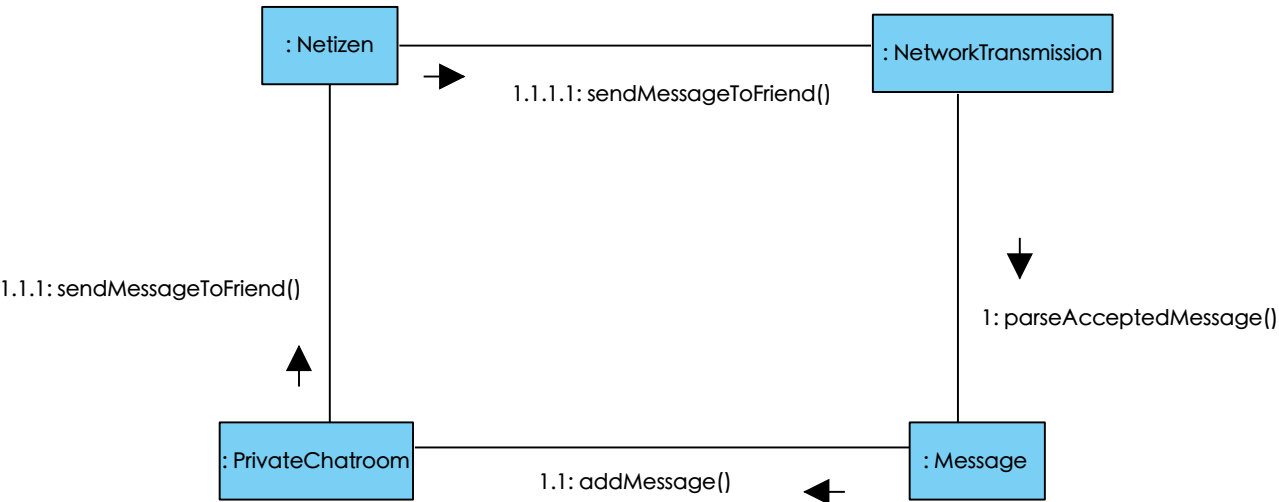
第三章 健壮性分析

3.1 通信图——私聊

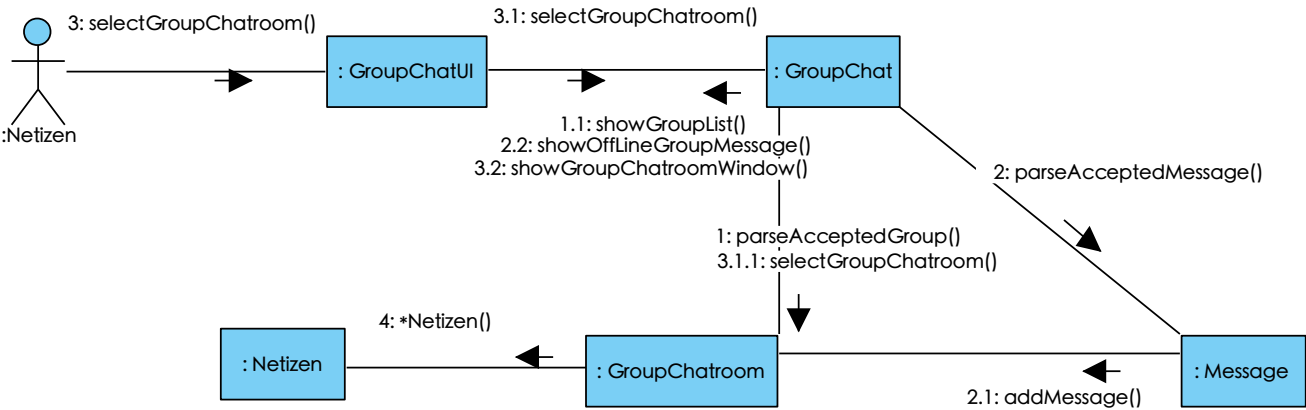
客户端



服务器

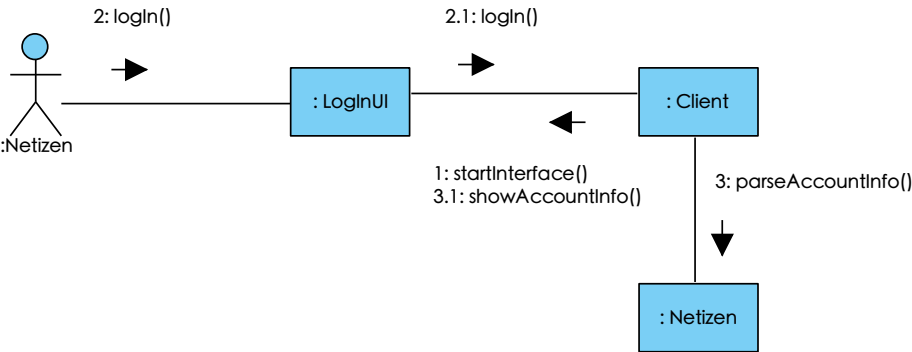


3.2 通信图——群聊

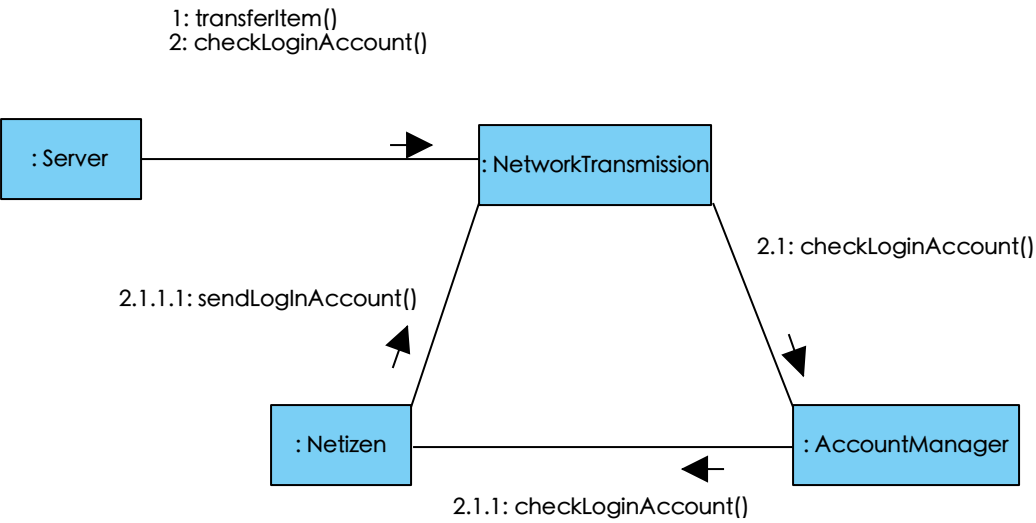


3.3 通信图——登录

客户端

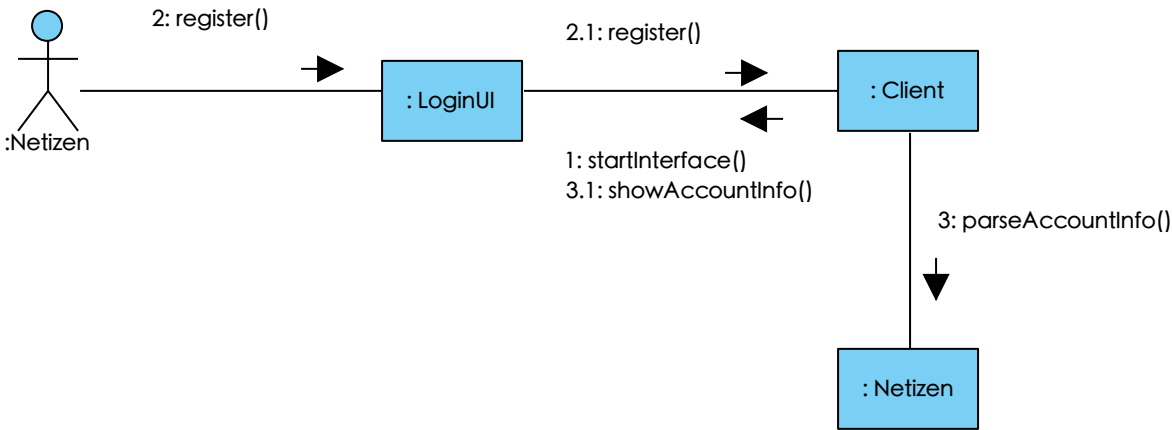


服务器

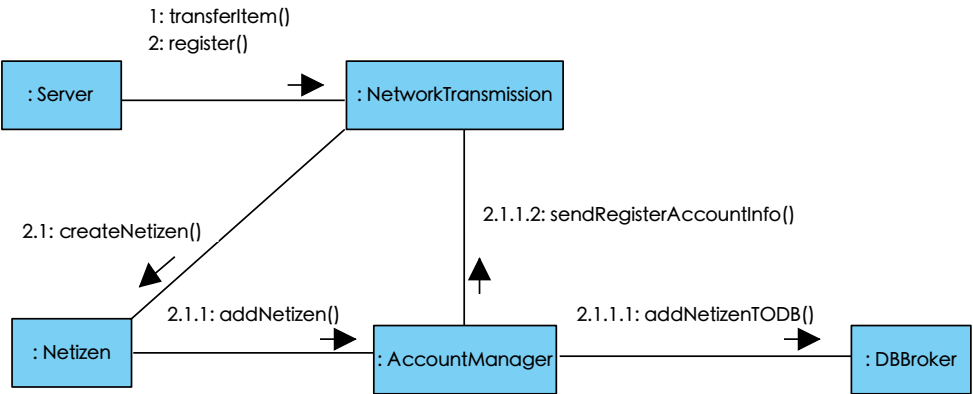


3.4 通信图——注册

客户端

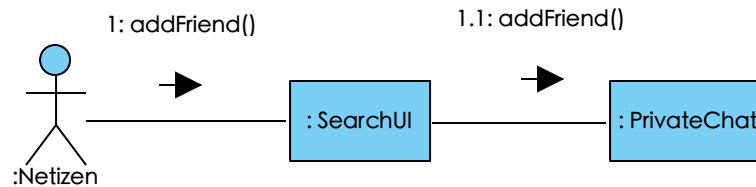


服务器

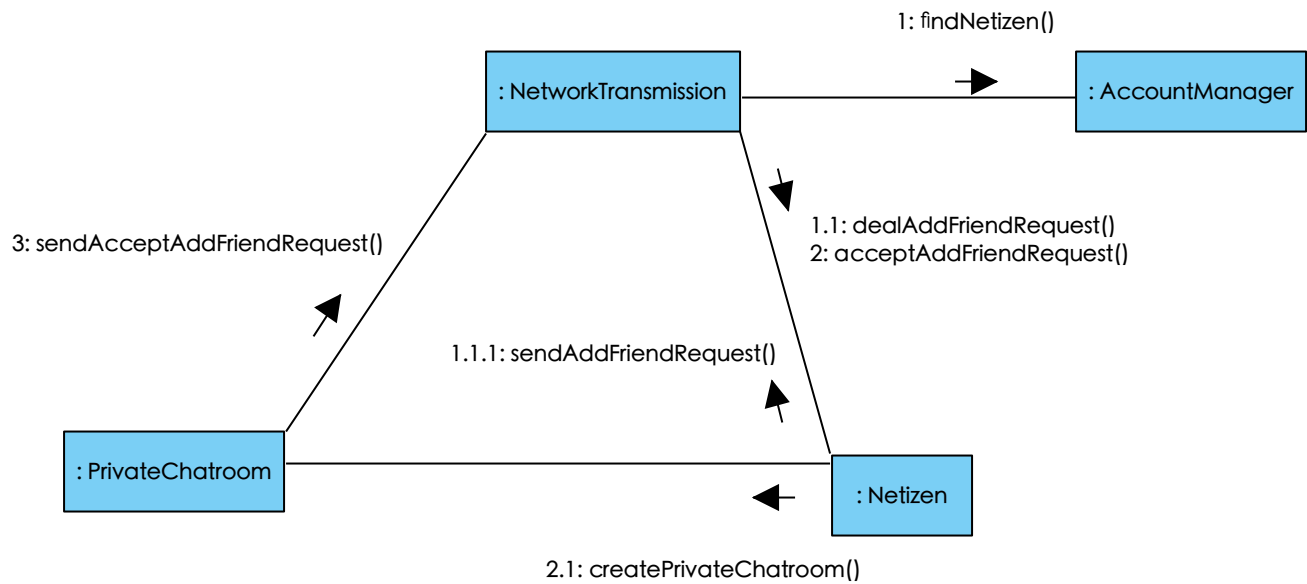


3.5 通信图——加好友

客户端

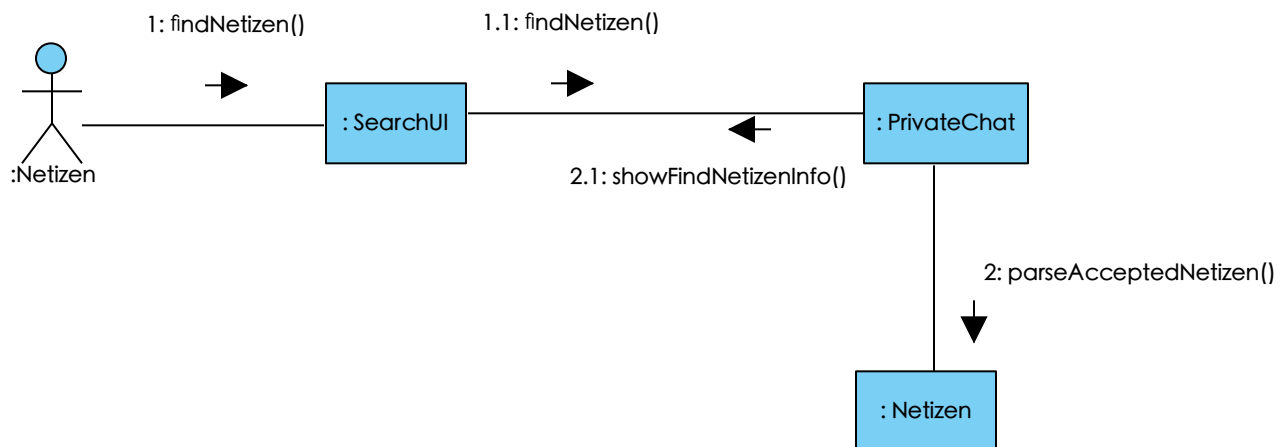


服务器

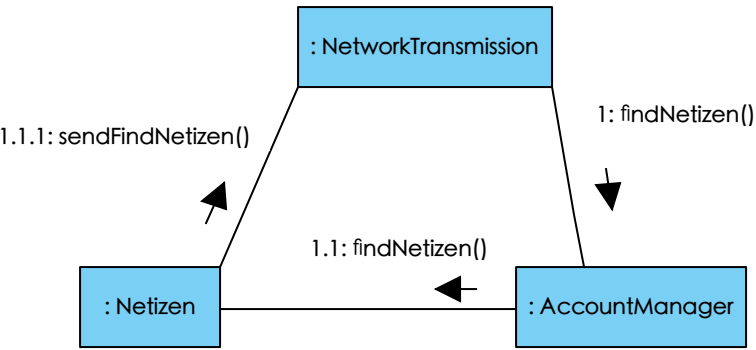


3.6 通信图——查找好友

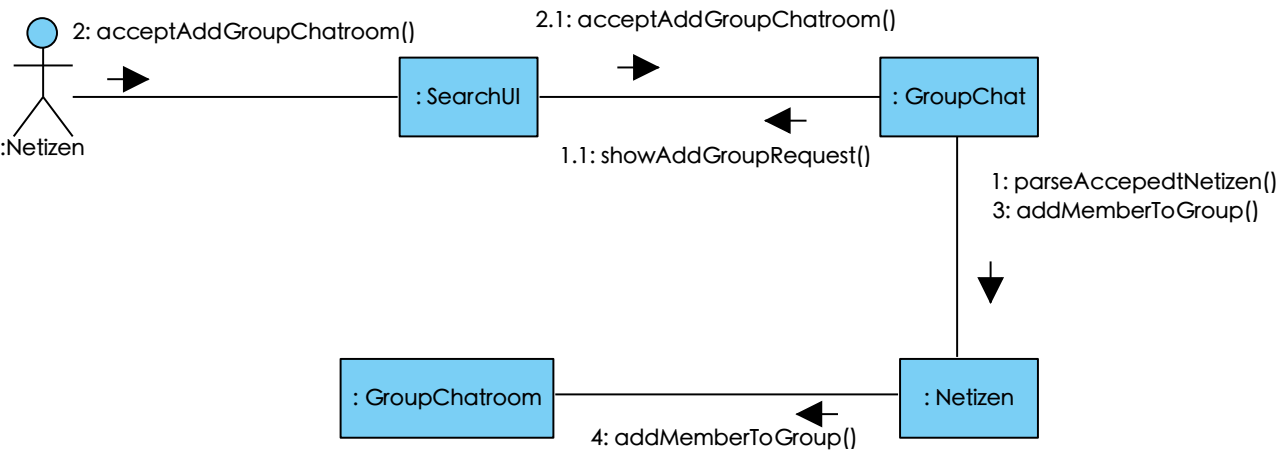
客户端



服务器

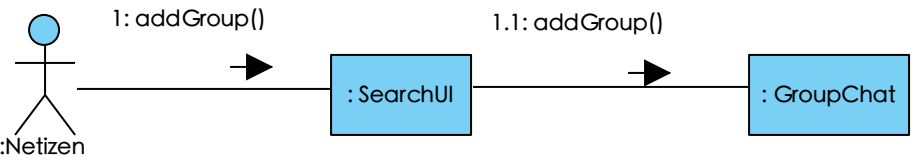


3.7 通信图——接受添加群

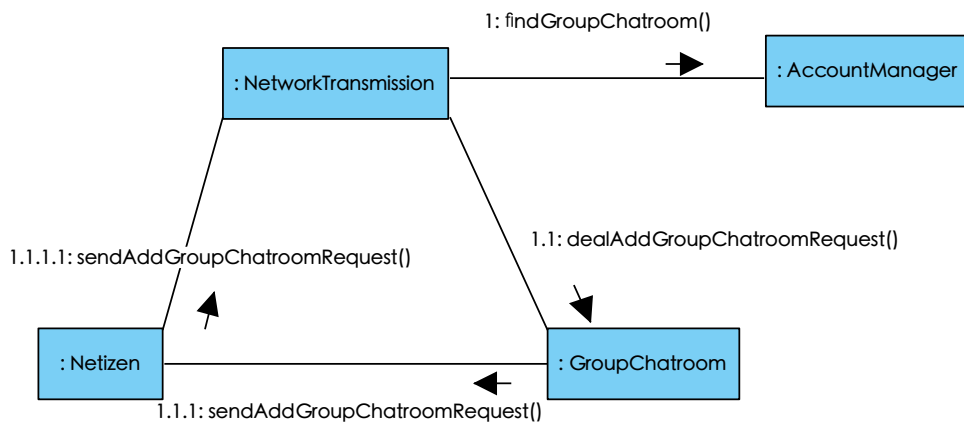


3.8 通信图——添加群

客户端

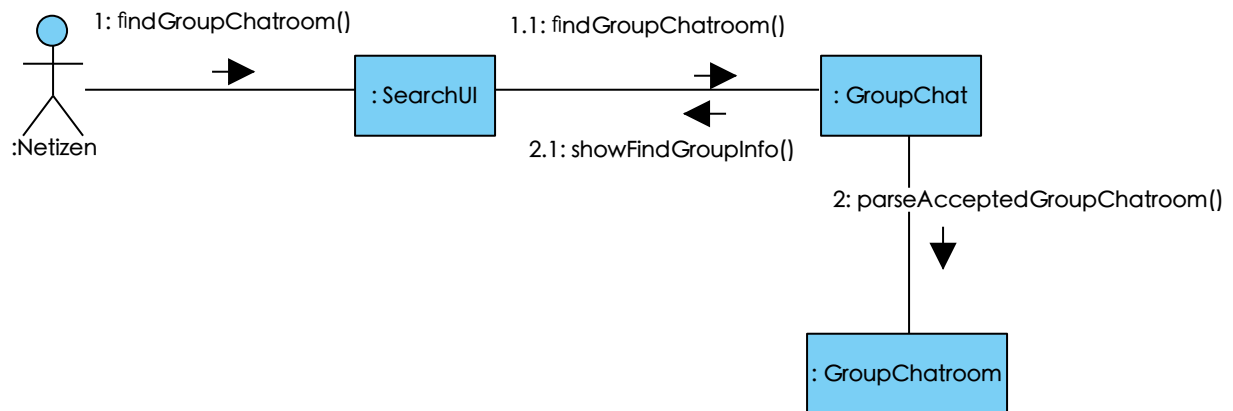


服务器

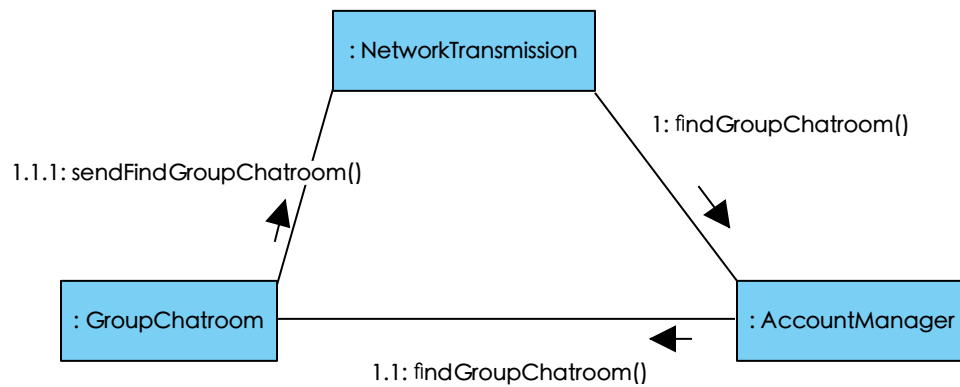


3.9 通信图——查找群

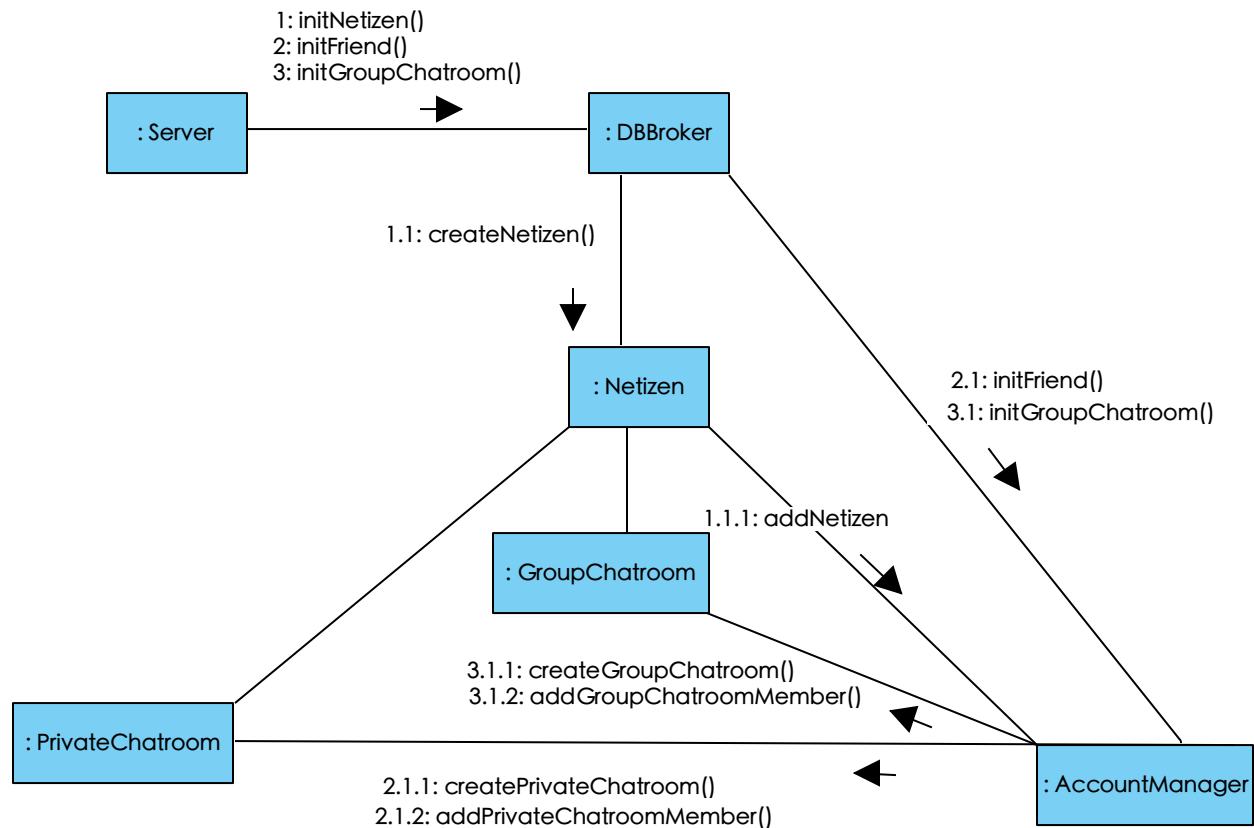
客户端



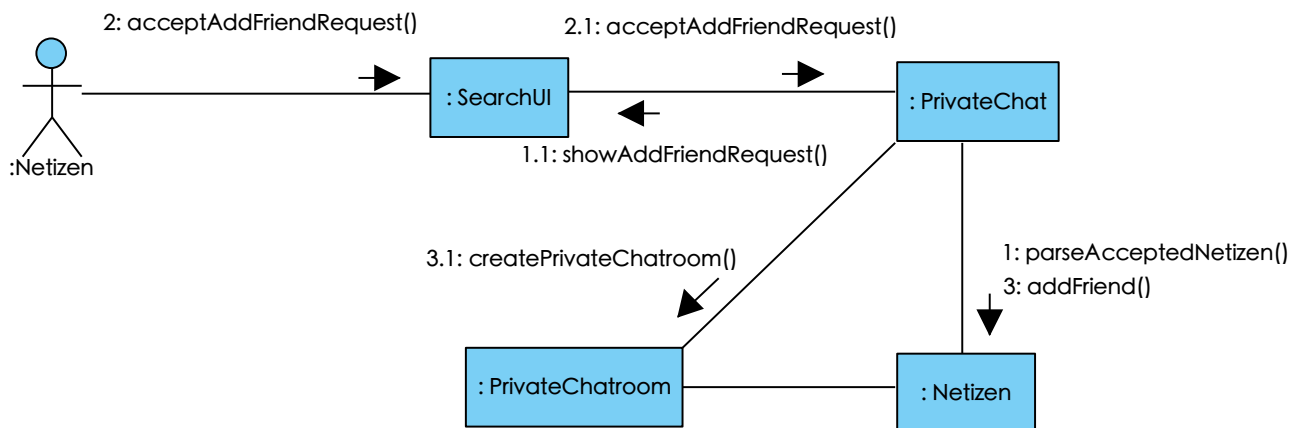
服务器



3.10 通信图——初始化系统



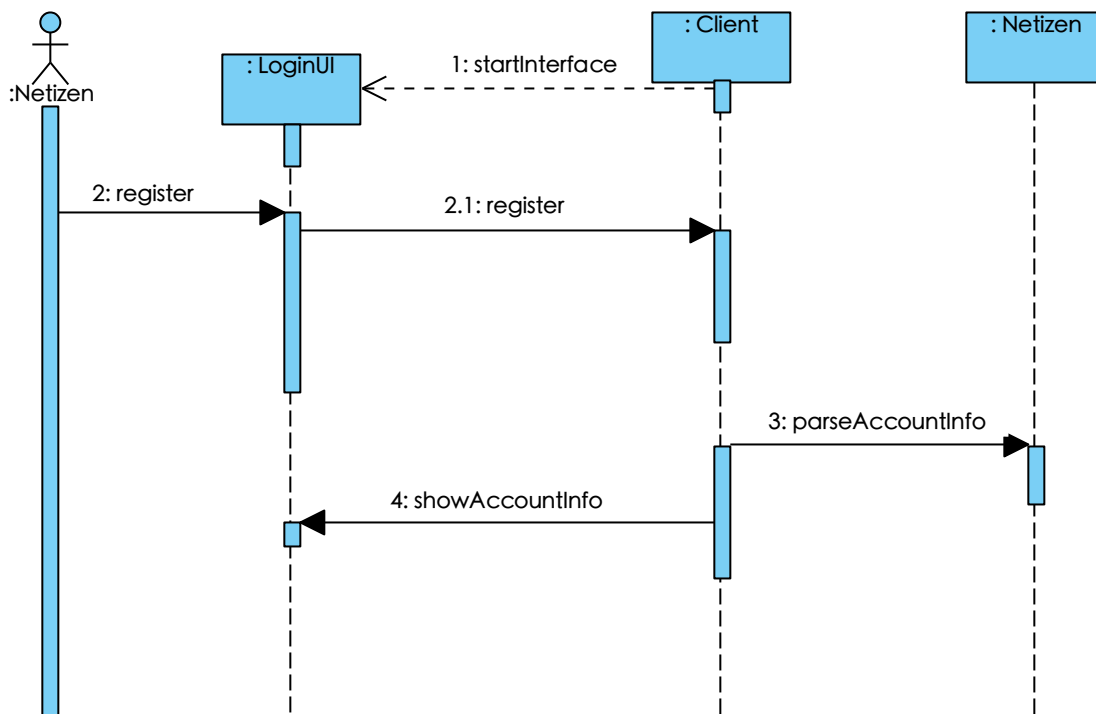
3.11 通信图——接受好友请求



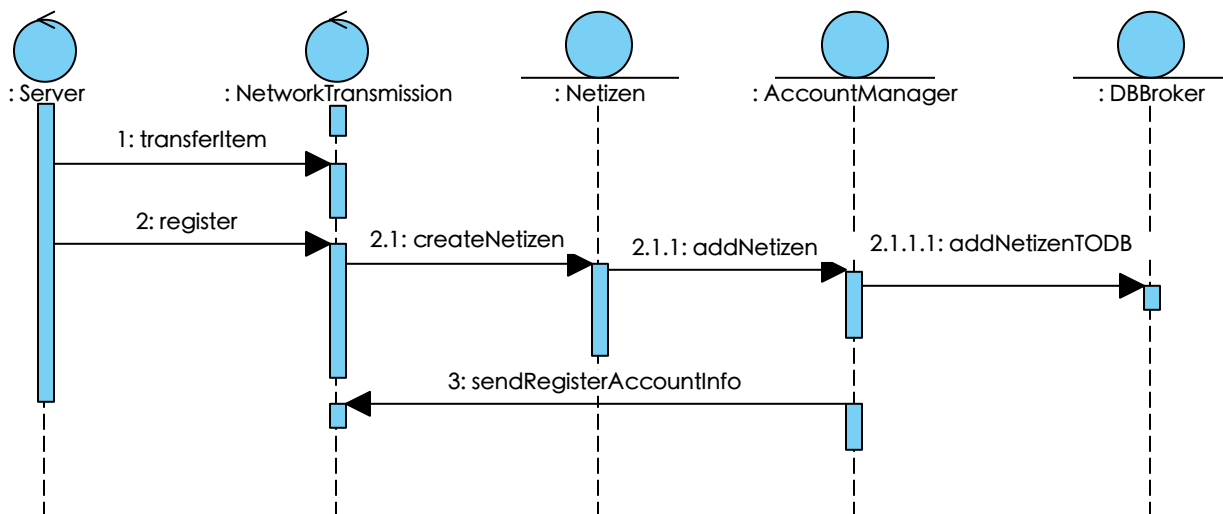
第四章 交互模型分析

4.1 顺序图——注册

客户端

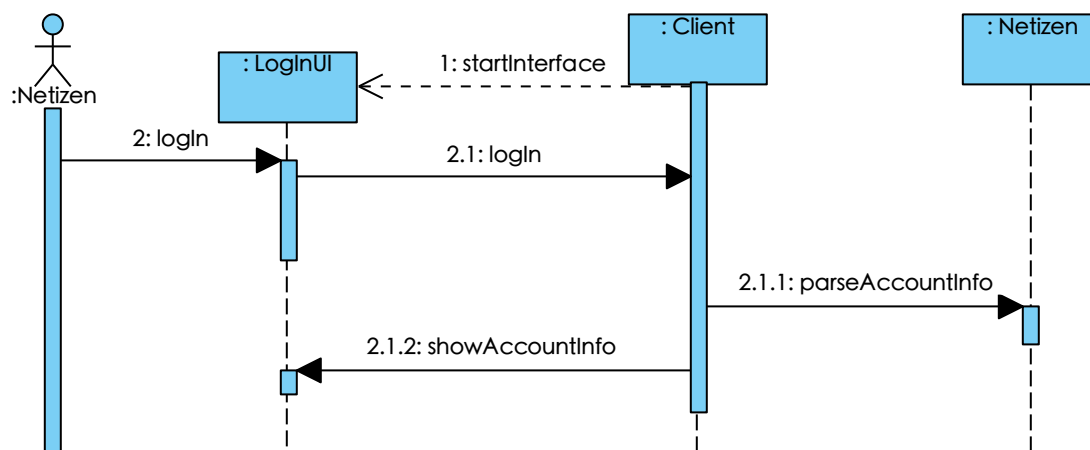


服务器

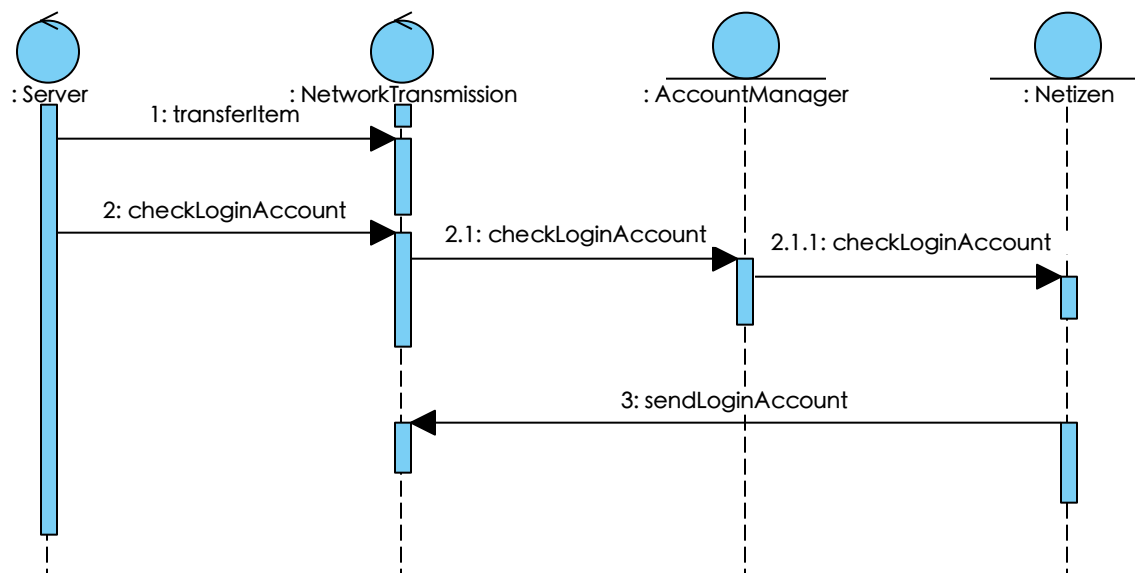


4.2 顺序图——登录

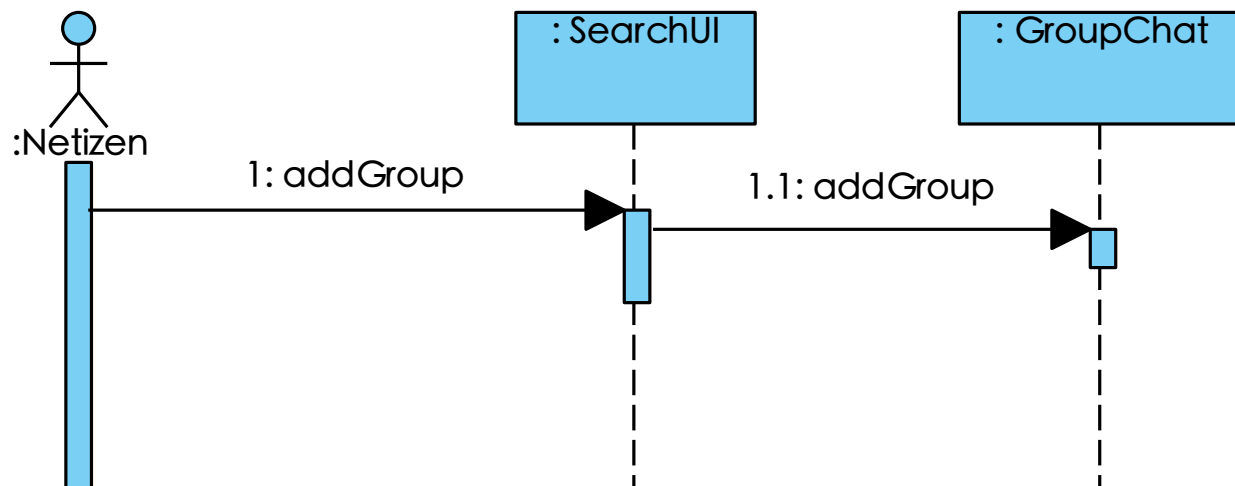
客户端



服务器

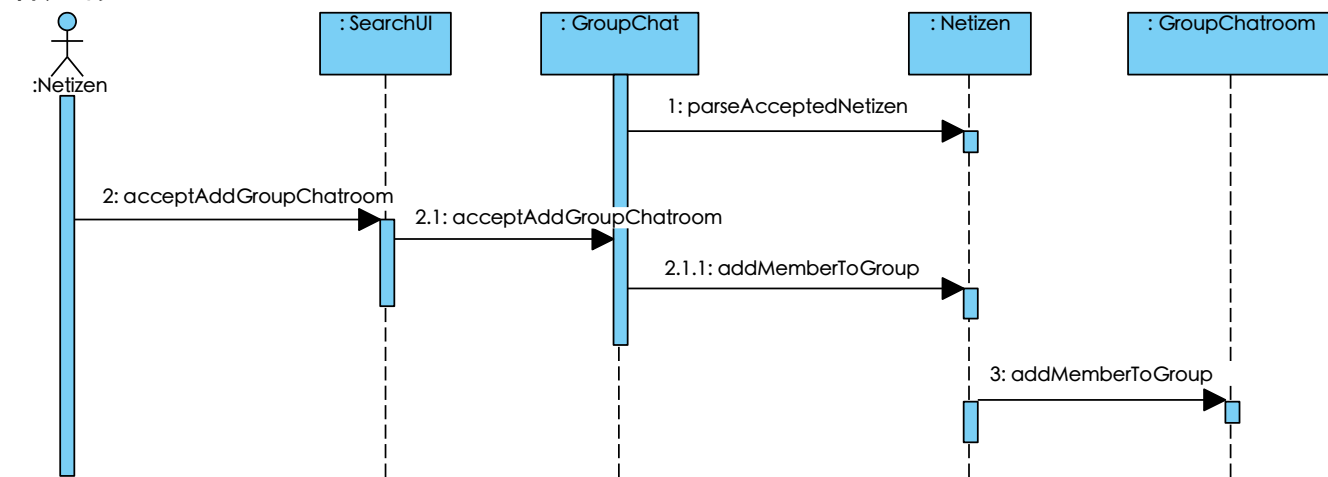


4.3 顺序图——加群

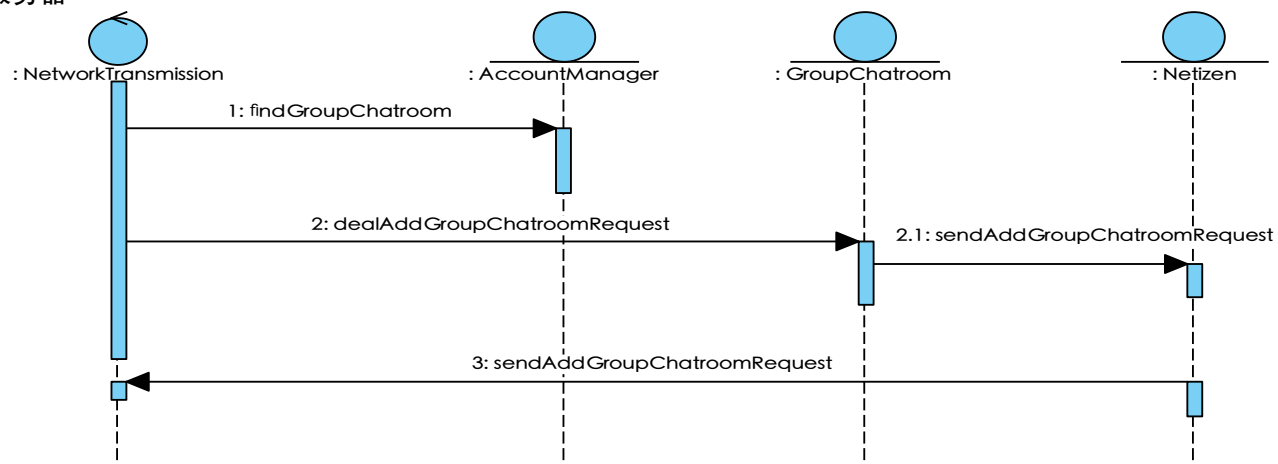


4.4 顺序图——接受加群请求

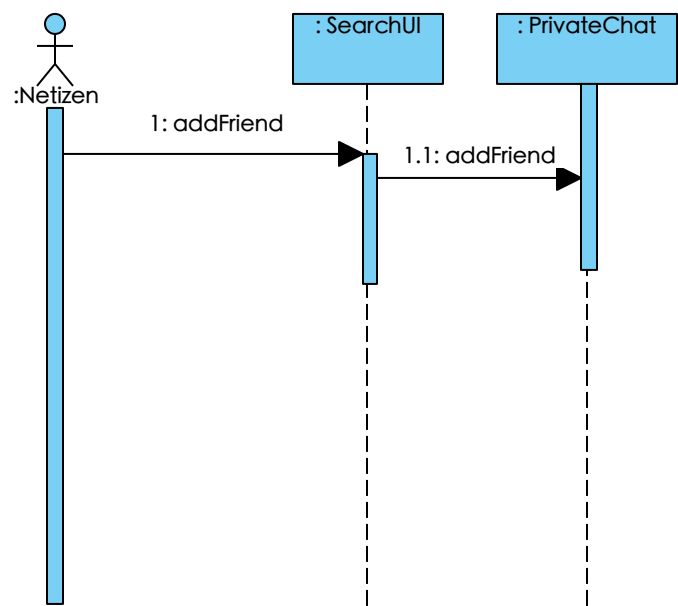
客户端



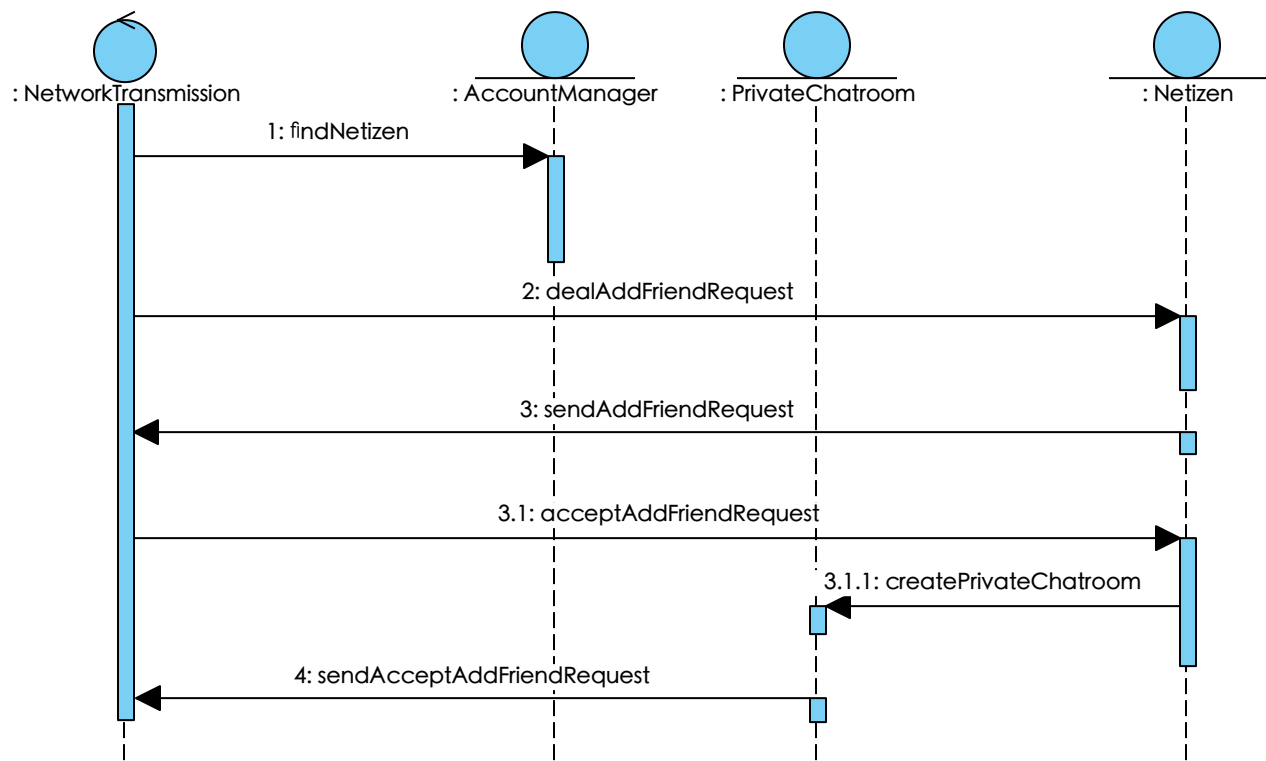
服务器



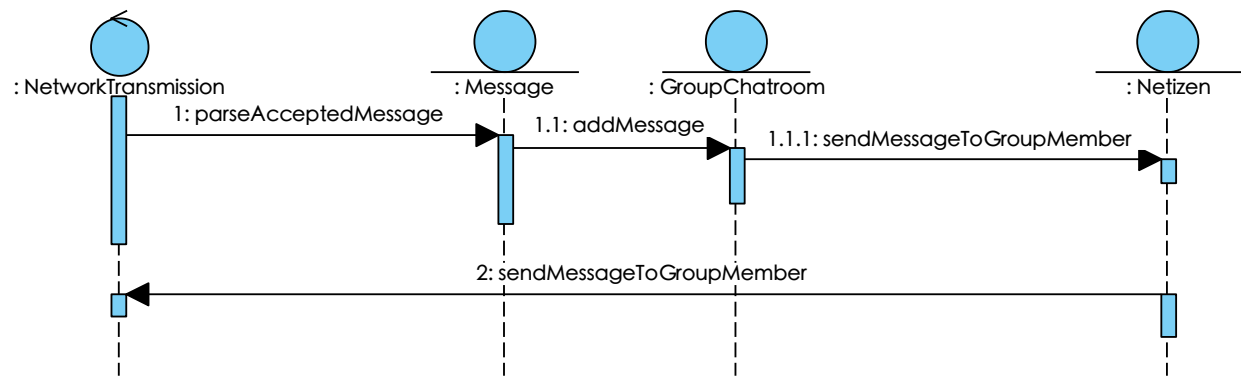
4.5 顺序图——加好友
客户端



服务器

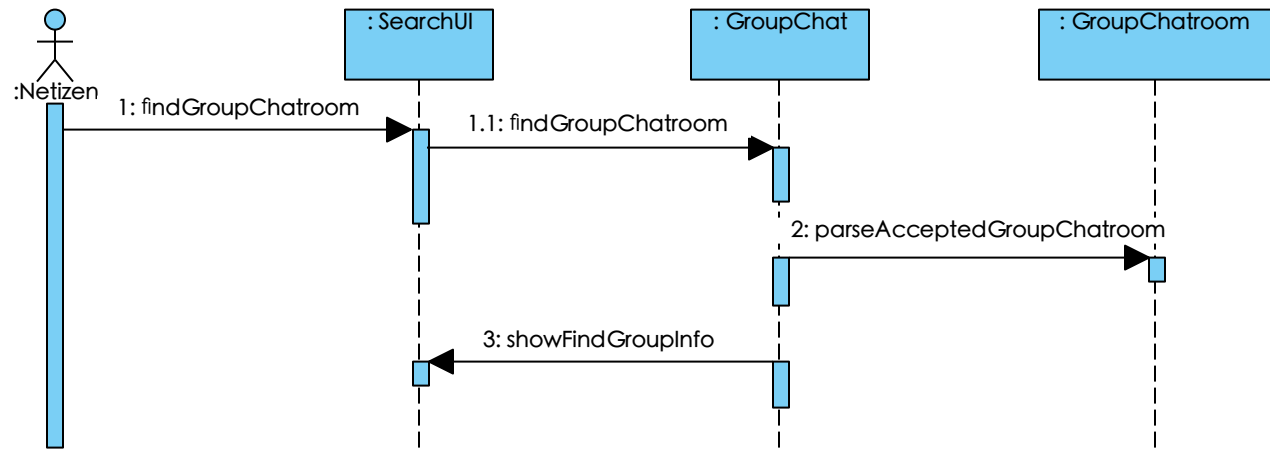


4.6 顺序图——群聊

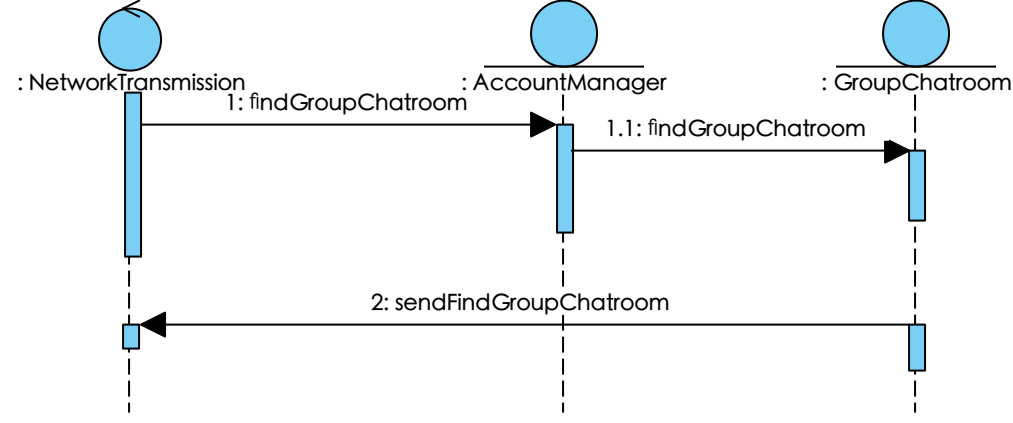


4.7 顺序图——找群

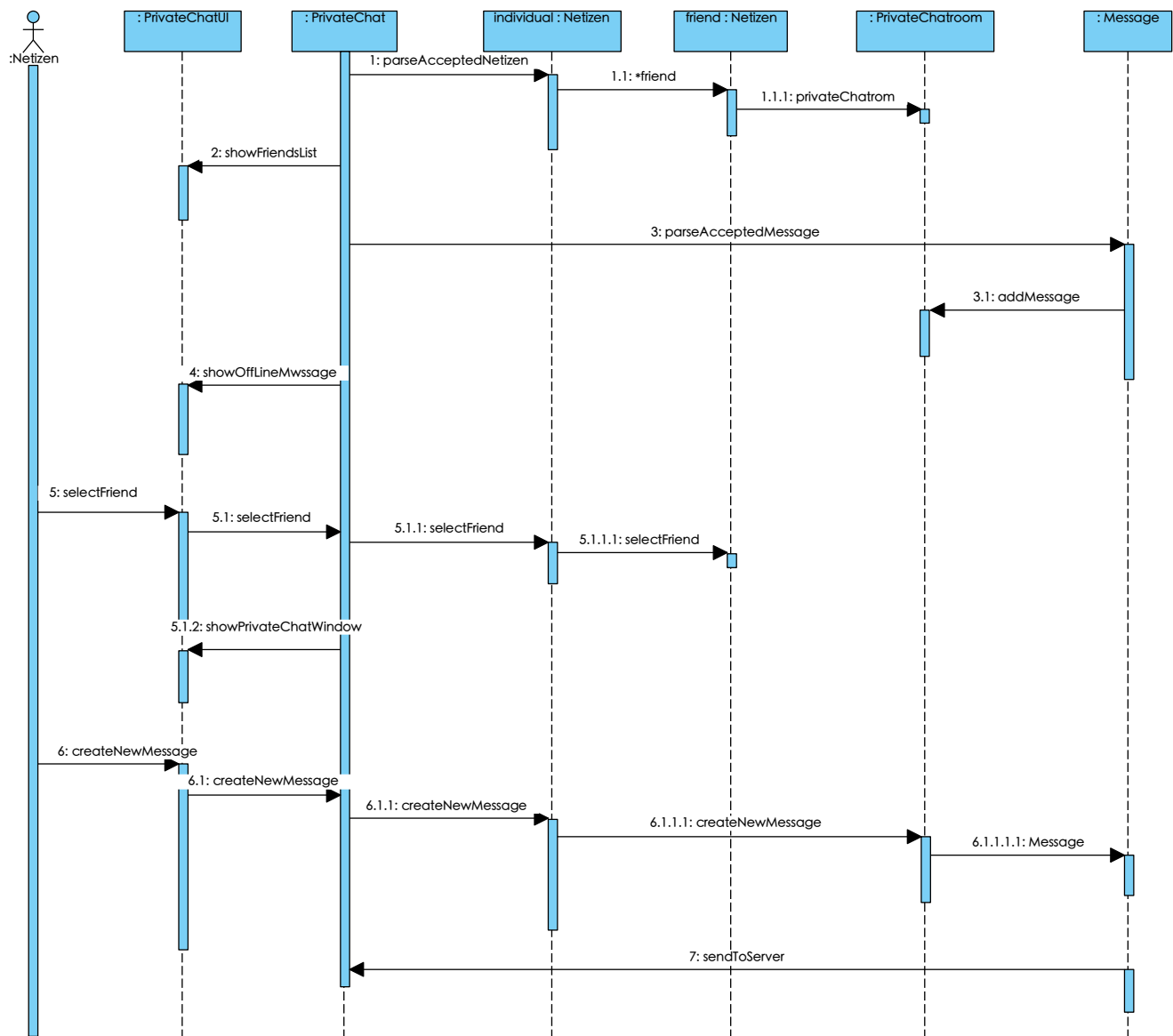
客户端

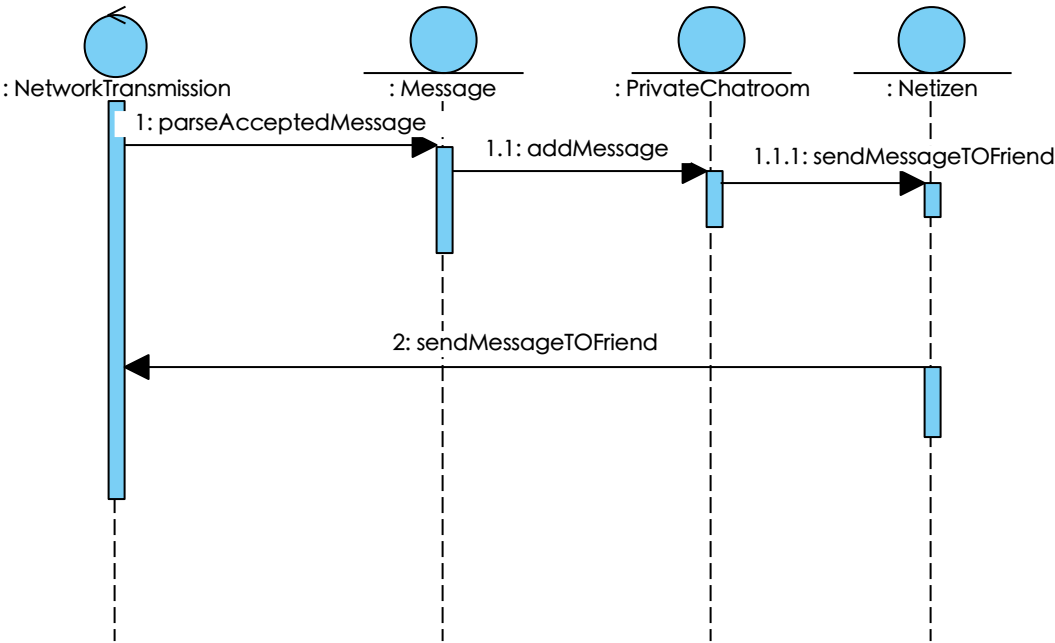


服务器

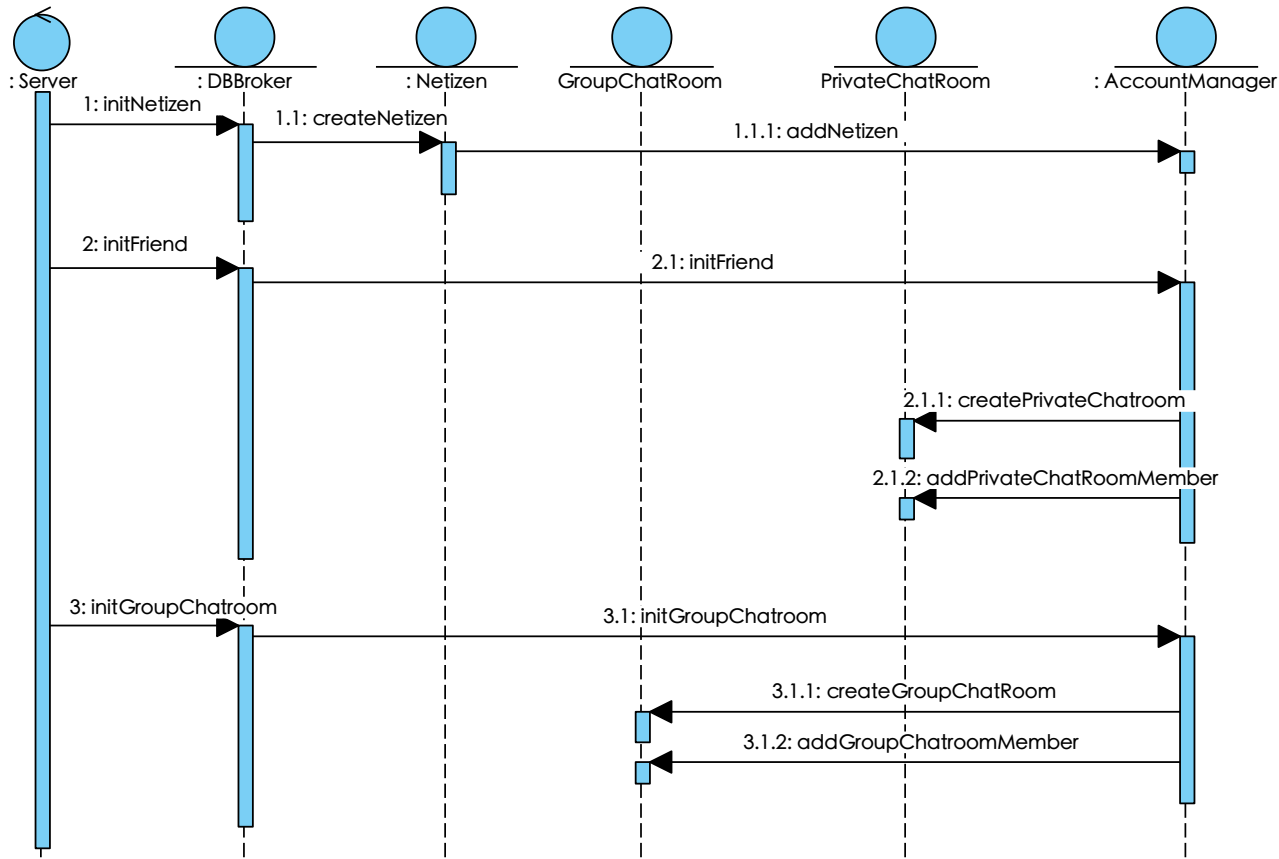


客户端

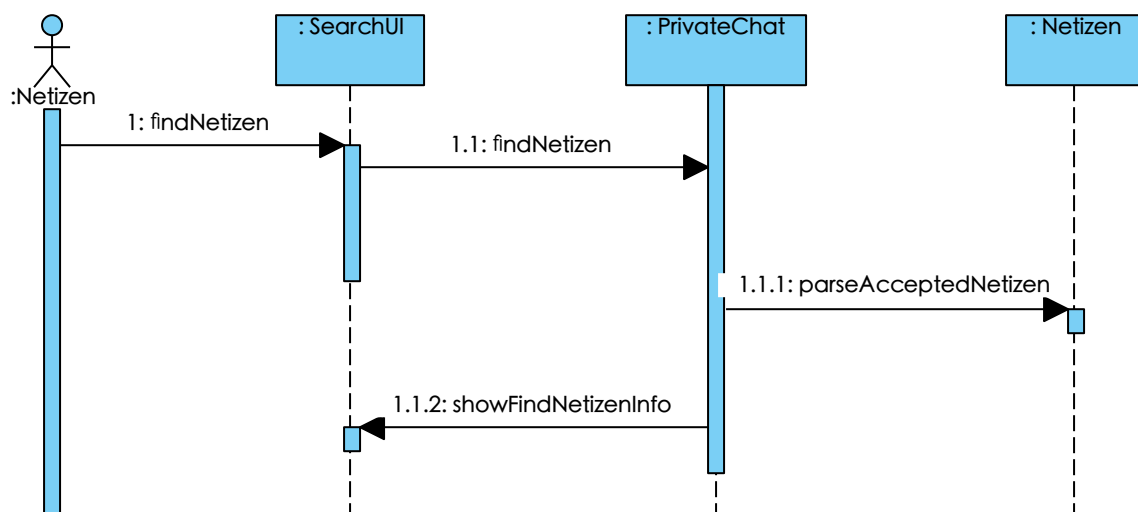




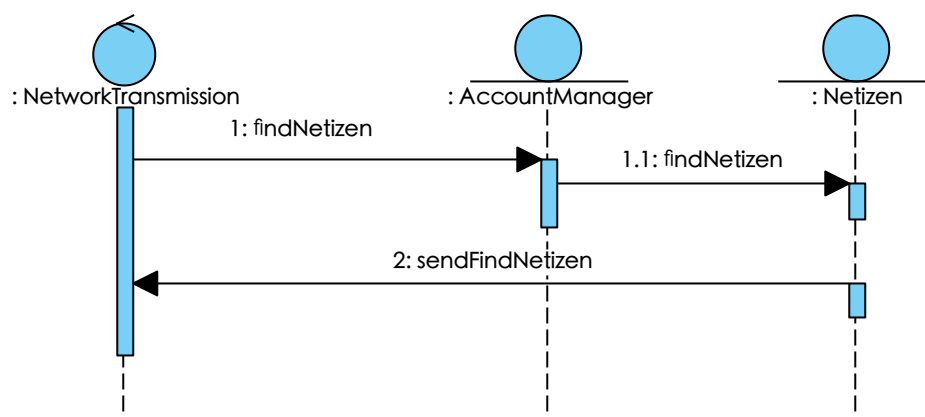
4.9 顺序图——初始化系统



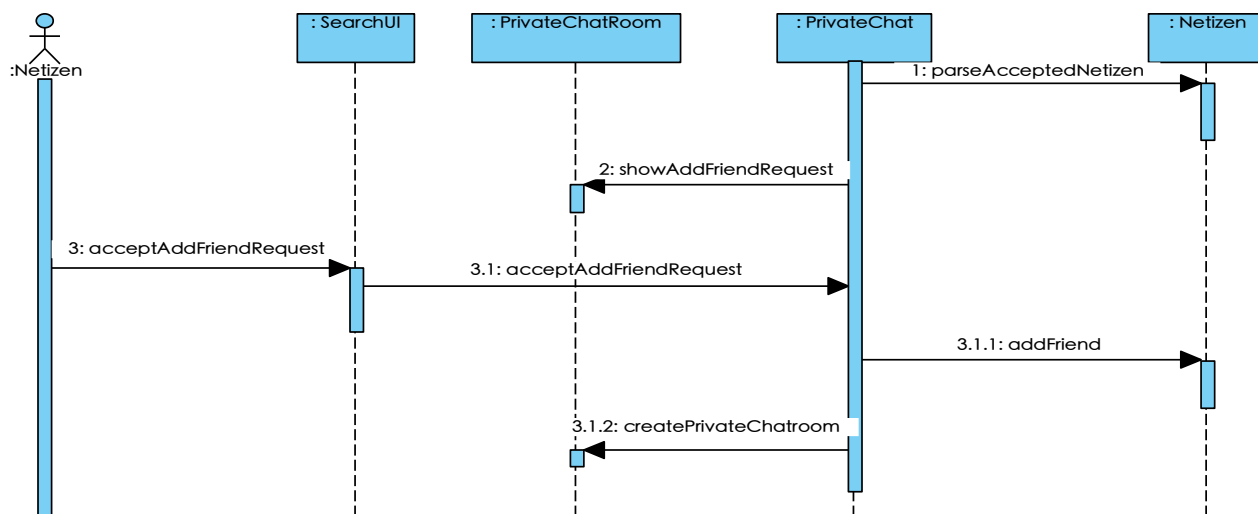
4.10 顺序图——查找好友



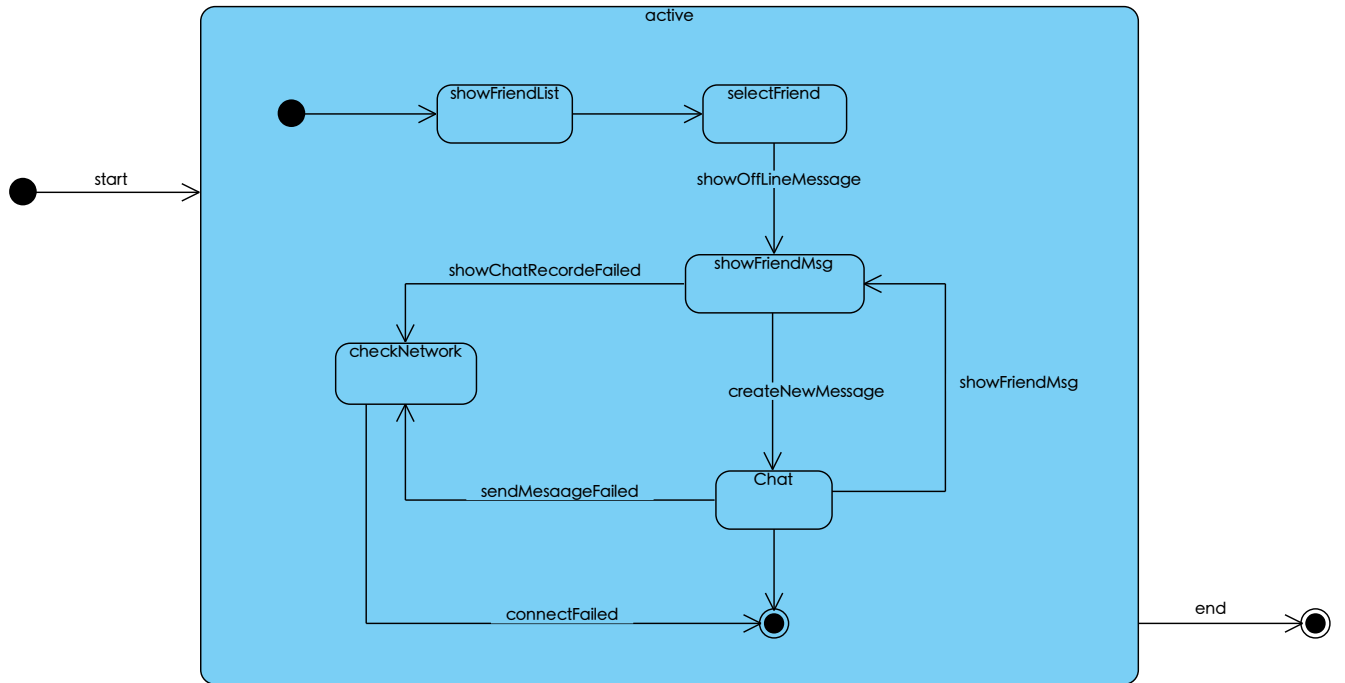
服务器



4.11 顺序图——接受加好友

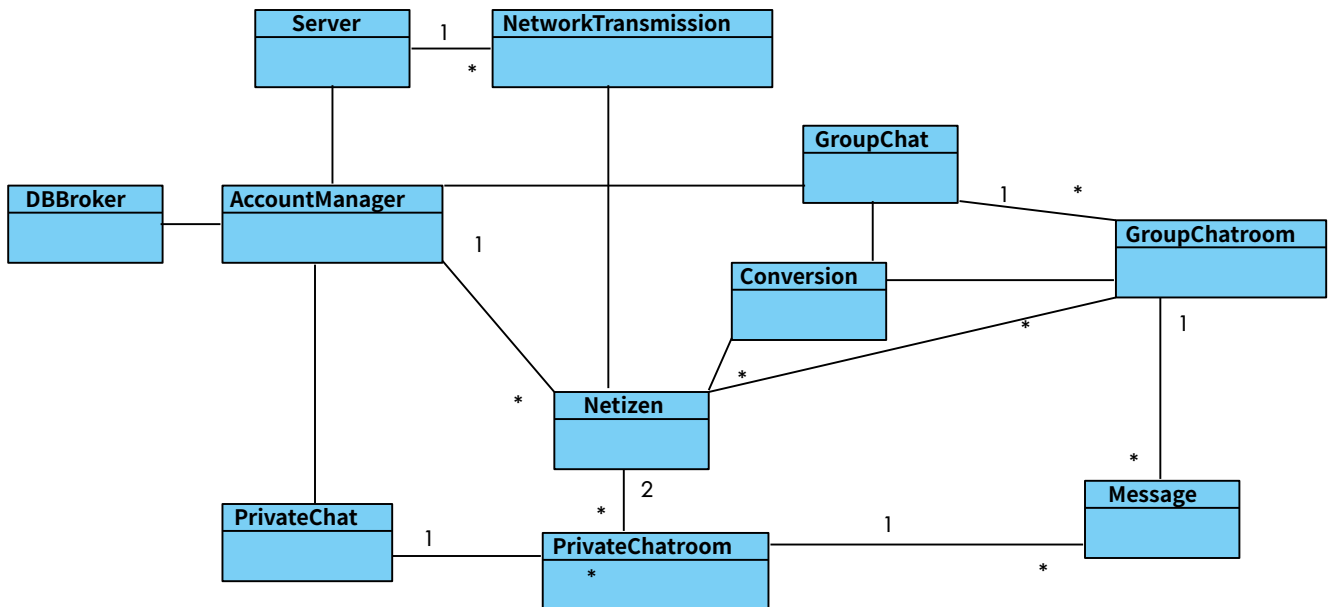


第五章 状态机分析



第六章 系统架构设计

6.1 系统类模型



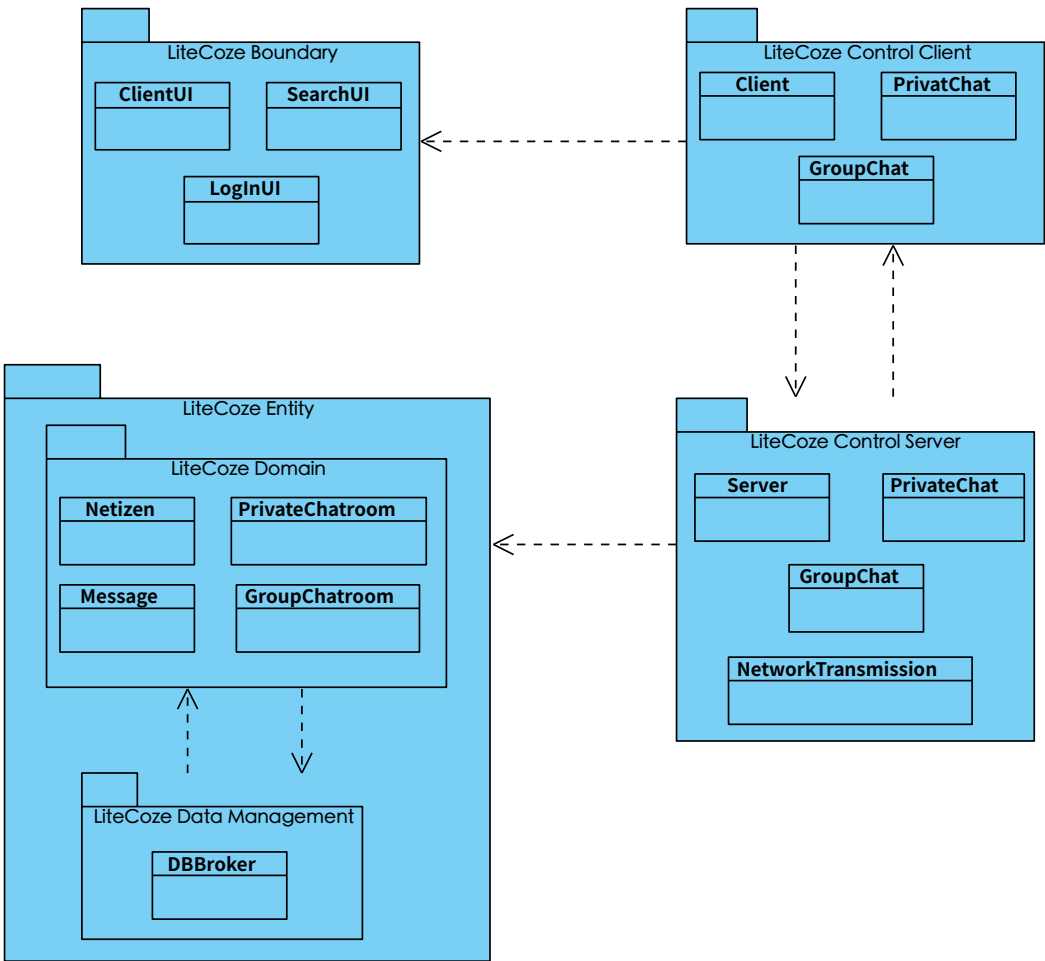
6.2 系统架构

1.开发技术和平台：

- 1) 开发语言：qml、C++
- 2) 开发框架和软件库：Qt 开发框架、boost.asio 库
- 3) 平台：服务器、客服端的搭建平台均为 Linux
- 4) 数据管理策略：服务器端所有实体类都转换为表存入 Mariadb 数据库，客服端将聊天消息本地储存

2.C/S 结构：

本系统采用了三层 C/S 软件风格体系结构，主要设计有一个服务器：LiteCoze Control Server 用于管理客户端数据和处理客户端的数据请求，一个客户端用于显示服务器返回的数据。该系统提供对客户端请求的接受和处理，重点在于服务器和客户端之间的交互。其中 Network-Transmission 用于在客户端和服务端之间进行网络传输。PrivateChat、GroupChat 为控制对象，对系统中的实体进行管理，具体的实体对象将存储在数据库中，下图为系统架构图。



3.MVC 框架

MVC 为模型（Model）、视图（View）、控制器（Controller）的缩写。应用程序被抽象为这 3 个部分，即分工又合作的完成用户提交的每一项任务。在本系统中采用 MVC 框架对客户端进行设计，模型（Model）层包含用户的数据部分，视图（View）层包含客户端的界面展示，控制器层用来控制界面与数据之间的交互。以此实现动态加载数据到客户端界面。

4.网络协议的选择——TCP/IP 协议

本系统之选择 TCP/IP 协议作为网络传输协议，是因为它低成本、可在不同的平台间进行通信的能力和开放的特性。TCP/IP 协议标准完全开放，可供用户免费使用，并独立于特定的计算机硬件和操作系统；可以运行在广域网，更适合于互联网；网络地址统一分配，网络中每一设备和终端都具有唯一地址；高层协议标准化，可提供多种多样可靠网络服务。

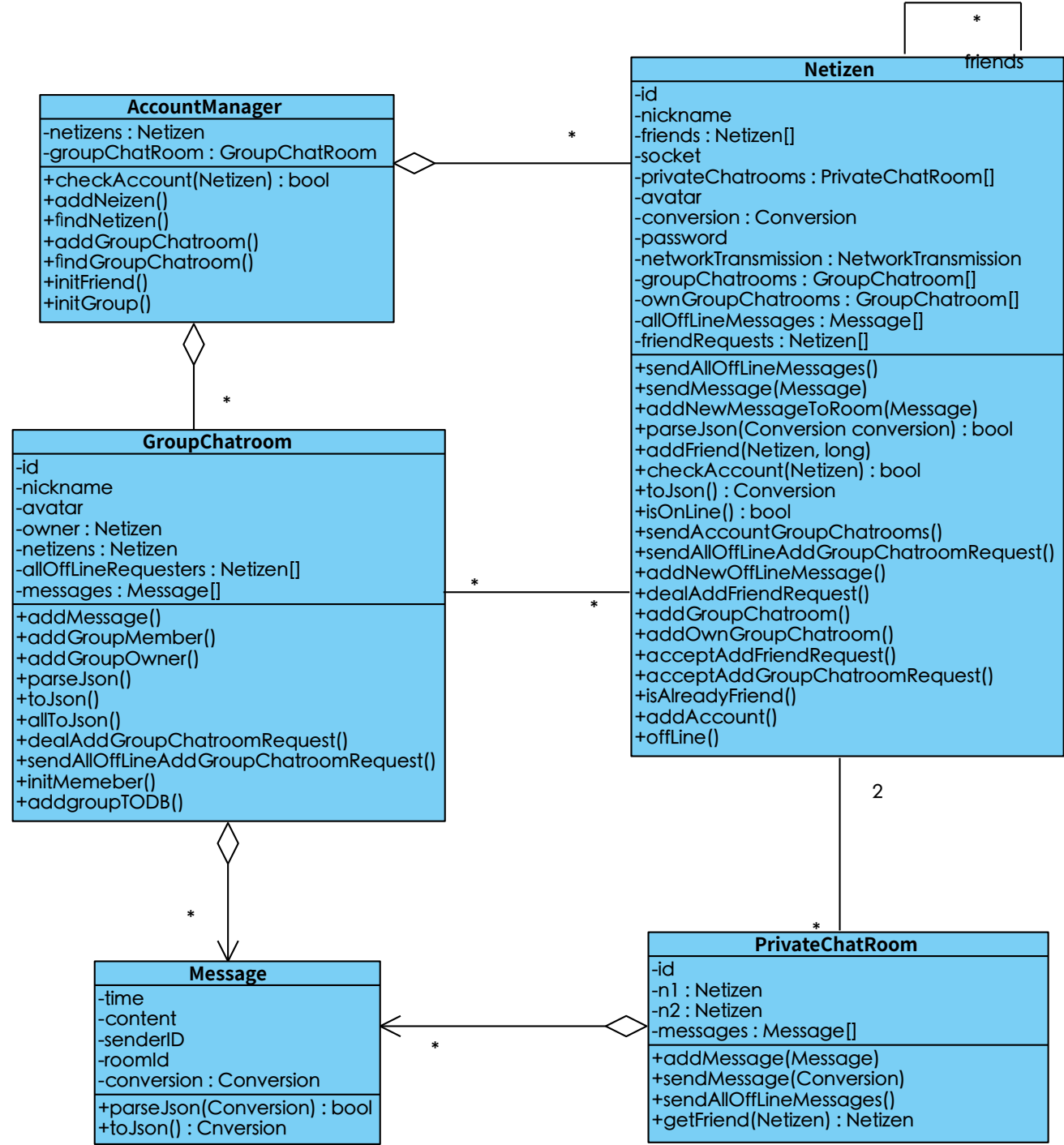
5.Boost 库

Boost.asio 是一个跨平台的、主要用于网络和其他一些底层输入/输出编程的 c++库。本系统采用 boost.asio 库实现异步网络编程。Boost 库可以在大多数操作系统上使用，可以同时支持数千个并发的连接。

6.数据传输选择——Json

Json 是一种轻量级的数据交换格式。简洁和清晰的层次结构使得 Json 成为理想的数据交换语言。Json 使用结构化的方式来标记数据，易于人阅读和编写，同时也易于机器解析和生成，并有效提升网络传输的效率。本系统使用 Json 在客户端、服务器之间进行传输。

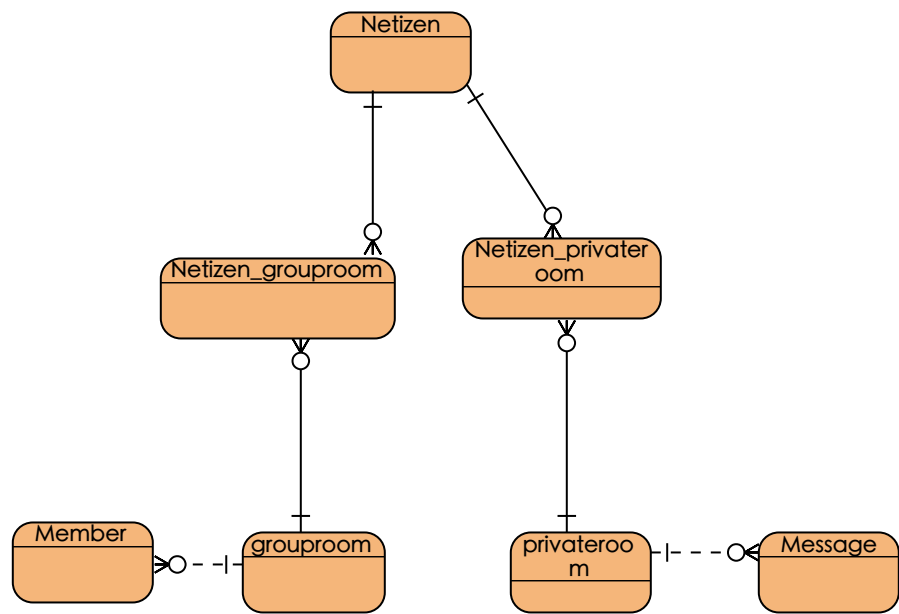
6.3.系统实体类图详细设计



第七章 数据库设计

7.1 使用 Mariadb 数据库工具，整体表关系

系统通过数据类和管理实体的 accountManager 类相关联来管理实体对象的数据。网民和私聊房间、群聊房间均为多对多的关系，所以加入中间表以表示它们的关联。私聊房间跟消息是一对多的关系，群聊房间跟群成员之间为一对多的关系。



7.2 将所有实体类做成相应的表

1.网民类

1.netizen 表描述了系统中所有已注册的网民的信息，其中网民的 id 是主键（唯一）

MariaDB [LiteCoze]> describe netizen;

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	NULL	
password	varchar(15)	NO		NULL	
nickname	varchar(50)	NO		NULL	
avatar	blob	YES		NULL	
signature	varchar(180)	YES		NULL	

2.mysql表示: create table netizen(id BIGINT primary key,
password VARCHAR(15) not null,
nickname VARCHAR(50) not null,
avatar blob,
signature VARCHAR(180))

2.私聊房间类

1.privateroom表描述了系统中所有私聊房间信息,存有该房间 id (主键,唯一标识)和存在好友关系的两个网民的 id

```
MariaDB [LiteCoze]> describe privateroom;
```

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	NULL	
netizen1_id	bigint(20)	YES		NULL	
netizen2_id	bigint(20)	YES		NULL	

2.mysql表示: create table privateroom(id BIGINT primary key,
netizen1_id BIGINT,
netizen2_id BIGINT)

3.netizen_privateroom 表

1.该表是网民表和私聊房间表的中间表用来表示它们之间多对多的关联,以网民的 id 和私聊房间 id 作为外键

```
MariaDB [LiteCoze]> describe netizen_privateroom;
```

Field	Type	Null	Key	Default	Extra
room_id	bigint(20)	YES	MUL	NULL	
netizen_id	bigint(20)	YES	MUL	NULL	

2.mysql表示: create table netizen_privateroom(
room_id BIGINT,

```
foreign key (room_id) references room(id),
netizen_id BIGINT,
foreign key (netizen_id) references netizen(id))
```

4.群聊房间类

1.grouproom 表描述了系统中所有群聊房间信息，存有一个群的 id（主键，唯一标识）和该群的名称

```
MariaDB [LiteCoze]> describe grouproom;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id     | bigint(20)    | NO   | PRI | NULL    |       |
| name   | varchar(50)   | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+
```

2.mysql表示：create table grouproom(id BIGINT primary key,name VARCHAR(50) not null)

5.netizen_grouproom 表

1.该表是网民表和群聊房间表的中间表用来表示它们之间多对多的关联，以网民的 id 和群聊房间 id 作为外键

```
MariaDB [LiteCoze]> describe netizen_grouproom;
+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| grouproom_id   | bigint(20)    | YES  | MUL | NULL    |       |
| netizen_id     | bigint(20)    | YES  | MUL | NULL    |       |
+-----+-----+-----+-----+-----+
```

2.mysql表示：create table netizen_grouproom(grouproom_id BIGINT,

```
foreign key (grouproom_id) references grouproom(id),
netizen_id BIGINT,
foreign key (netizen_id) references netizen(id))
```

6.群成员表

- 1.该表是表名由“g 一个群聊房间 id”构成，描述了某个群的所有网民成员，以构成网民与群的关系
- 2.mysql 表示：create table %s(netizen_id BIGINT,foreign key (netizen_id) references netizen(id))

7.消息类

- 1.message 表描述了系统中所有聊天消息信息，存有发送网民的 id、房间号、内容

```
MariaDB [LiteCoze]> describe message;
```

Field	Type	Null	Key	Default	Extra
sender_id	bigint(20)	NO	PRI	NULL	
content	varchar(1000)	YES		NULL	
room_id	bigint(20)	YES	MUL	NULL	

- 2.mysql 表示：create table message(sender_id BIGINT primary key,
content VARCHAR(1000),
room_id BIGINT, foreign key (room_id) references room(id)

第八章 系统详细设计

8.1 抽象类设计

在 LiteCoze 系统具体实现过程中，采用了 socket 网络编程技术，使用了 Boost.Asio 网络库，主要应用在服务端 Server 类与 NetworkTransmission 类和客户端 Client 类上中。

8.1.1 Server 类

Server 类是服务端的一个单例类，主要用于接收客户端的连接，其中 acceptor 绑定了 TCP/IP 协议和端口号，accept() 用于监听服务端的连接请求，checkLogInAccount 用于检查登录信息，transferItem() 用于打开接收消息的接口。如图 8-1 为 Server 的类图。

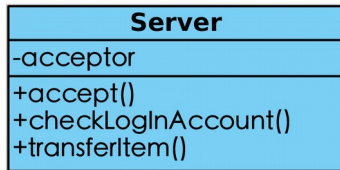


图 8-1 Server 类图

8.1.2 NetworkTransmission 类

NeteorkTransmission 类主要用于与客户端之间传输消息。Server 每接收到一个客户端的连接就产生一个新的 NetworkTransmission 对象，其中 socket 与新连接上的客户端的 socket 相绑定，do_accept_head() 用于接收消息的头部信息，do_accept_body() 用于接收消息的具体内容，do_send() 用于发送消息。如图 8-2 为 NetwokTransmission 的类图。

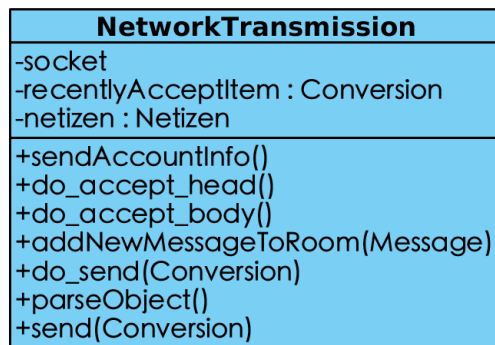


图 8-2 NetworkTransmission 类图

8.1.3 Client 类

Client 类是客户端的一个单例类，主要用于与服务端之间传输消息，其中 socket 绑定的服务器的 IP 地址与端口号，connectServer() 用于连接服务器，do_accept_head() 用于接收消息的头部信息，do_accept_body() 用于接收消息的具体内容，do_send() 用于发送消息。如图 8-3 为 Client 的类图。

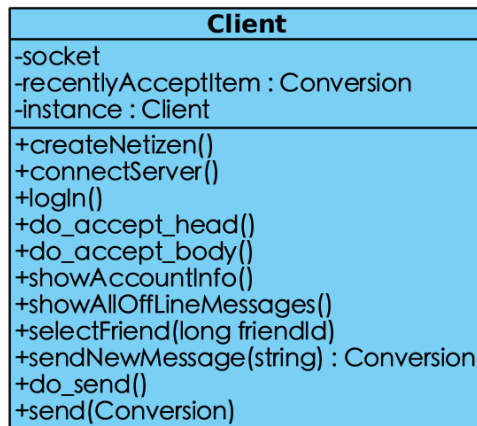


图 8-3 Client 类图

8.2 服务端交互设计

- 1) 启动服务器，初始化系统，将数据库中的相关信息读取到程序中。
- 2) 系统初始化结束后，创建 Server 对象，调用 `async_accept()` 函数轮询监听客户端的连接，当一个客户端成功连接后，继续调用 `async_accept()` 函数轮询监听下一个客户端的连接。
- 3) 每监听到新连接时，Server 对象会创建一个新的 `NetworkTransmission` 对象，开始与客户端交换信息。
- 4) `NetworkTransmission` 对象的 `async_read()` 函数轮询等待客户端发送的消息，当成功接收一条消息，则继续调用 `async_read()` 函数轮询等待下一条消息。
- 5) 每成功接收到一条消息，则解析消息类型。解析成功后将消息交给对应的函数进行处理，处理完成后通过 `async_write()` 函数将结果返回给对应的客户端。

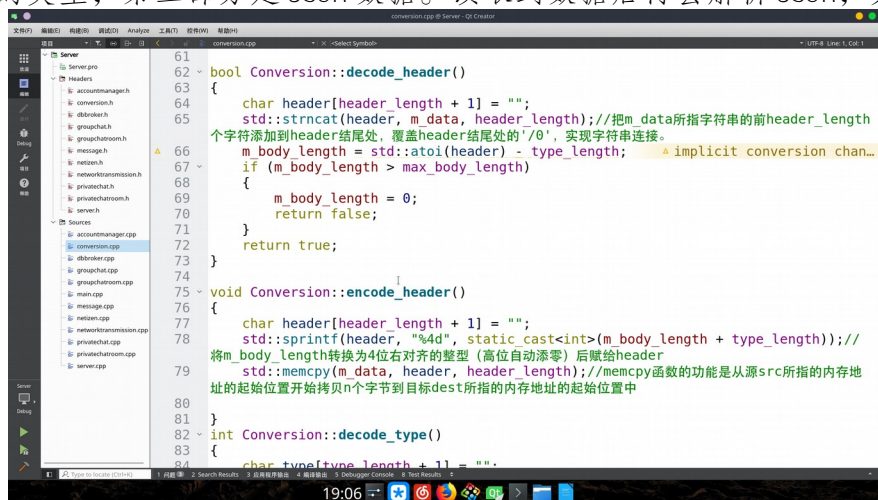
8.3 客户端交互设计

- 1) 启动客户端，客户端通过 Client 对象的 `async_connect()` 函数连接服务端。
- 2) 接收 UI 界面传来的请求。
- 3) Client 对象通过 `async_write()` 函数将请求发送到服务端，通过 Client 对象的 `async_read()` 函数轮询等待服务端返回的结果，当成功接收一条消息，则继续调用 `async_read()` 函数轮询等待下一条消息。

第九章 实现

9.1tcp 数据传输

在本系统中 tcp 协议最大传送数据为 5000，在客户端有请求发出后，会通过套接字发送 tcp 数据到服务器。在服务器会读取 tcp 数据，整个数据分成三部分，头部存放后两部分的长度，第二部分是消息的类型，第三部分是 Json 数据。读取到数据后再去解析 Json，具体代码如下：



```
61
62 bool Conversion::decode_header()
63 {
64     char header[header_length + 1] = "";
65     std::strncat(header, m_data, header_length); //把m_data所指字符串的前header_length
        个字符添加到header结尾处，覆盖header结尾处的'\0'，实现字符串连接。
66     m_body_length = std::atoi(header) - type_length; //implicit conversion chan...
67     if (m_body_length > max_body_length)
68     {
69         m_body_length = 0;
70         return false;
71     }
72     return true;
73 }
74
75 void Conversion::encode_header()
76 {
77     char header[header_length + 1] = "";
78     std::sprintf(header, "%d", static_cast<int>(m_body_length + type_length)); //
        将m_body_length转换为4位右对齐的整型（高位自动添零）后赋给header
79     std::memcpy(m_data, header, header_length); //memcpy函数的功能是从源src所指的内存地址
        的起始位置开始拷贝n个字节到目标dest所指的内存地址的起始位置中
80 }
81
82 int Conversion::decode_type()
83 {
84     char type[type_length + 1] = "";
```

```

void NetworkTransmission::do_accept_head()
{
    _recentlyAcceptItem = new Conversion();
    boost::asio::async_read(m_socket,
        boost::asio::buffer(_recentlyAcceptItem->data(), Conversion::header_length),
        [this](boost::system::error_code ec, std::size_t /*length*/)
        {
            cout << "ec: " << ec << endl;
            cout << _recentlyAcceptItem->data();
            if (!ec && _recentlyAcceptItem->decode_header())
            {
                do_accept_body();
            }
            else
            {
                m_socket.close();
                _netizen->offLine();
            }
        }
    );
}

void NetworkTransmission::do_accept_body()
{
    boost::asio::async_read(m_socket,
        boost::asio::buffer(_recentlyAcceptItem->type(), _recentlyAcceptItem->body_length() +
        Conversion::type_length),
        [this](boost::system::error_code ec, std::size_t /*length*/)
        {
            if (!ec && _recentlyAcceptItem->decode_type())
            {
                cout << "接收成功" << endl;
                cout << _recentlyAcceptItem->data() << endl;
                //cout << _recentlyAcceptItem->data() << endl;
                parseObject();
                do_accept_head();
                std::cout << "\n";
            }
        }
    );
}

```

9.2 数据管理

通过数据库类管理系统中的实体对象，并使用单例模式实现

```

#include <iostream>
#include <mysql/mysql.h>
#include "netizen.h"

class AccountManager;
class DBBroker
{
public:
    static DBBroker* getInstance();
    ~DBBroker();
    bool connect(); //连接数据库
    void initAccount(); //初始化所有netizen的账户信息
    long stolong(std::string str); //string转化为long
    std::string longToS(long id);
    void initFrinedInfo(Netizen *netizen); //初始化netizen所有好友信息
    void initGroupInfo(Netizen *netizen); //初始化netizen的群列表
    void queryMemberOfGroup(GroupChatroom *grouproom, Netizen* netizen); //查询一个群里面的所有群成员
    void addFriendToDB(long room, long n1_id, long n2_id); //添加好友关系到数据库
    void addAccountToDB(long id, std::string pw, std::string name); //添加注册的账户信息到数据库
    void addGroupToDB(long id, std::string name, long netizen_id); //创建一个群，添加到数据库
    void createMemberList(long id);
    void addGroupInfo(long id, long netizen_id);
    void addMember(long id, long netizen_id);
private:
    DBBroker();
    MYSQL * connect; //数据库连接的句柄
    MYSQL_RES * _result; //返回行的查询结果
    MYSQL_ROW m_row; //一次存储一行结果集里面的数据
    //std::vector<Netizen*> _netizens;
    static DBBroker* _instance;
};

```

9.3 网络通信

```
void Client::connectServer(const tcp::resolver::results_type &endpoints)
{
    boost::asio::async_connect(m_socket, endpoints,
                               [this](boost::system::error_code ec, tcp::endpoint)
    {
        if (!ec)
        {
            std::cout << "连接成功" << std::endl;
            do_accept_head();
        }
        else{
            cout << "连接失败" << endl;
        }
    });
}

void Client::login(long id, string password)
{
    netizen = new Netizen(id, password);
    send(netizen->toJson());
}

void Client::sendMessage(string content)
{
    send(netizen->createNewMessage(content));
}

void Client::do_accept_head()
{
    _recentlyAcceptItem = new Conversion();
    boost::asio::async_read(m_socket,
                           boost::asio::buffer(_recentlyAcceptItem->data(), Conversion::header_length),
                           [this](boost::system::error_code ec, std::size_t /*length*/)
    {
        cout << "ec: " << ec << endl;
        if (!ec && _recentlyAcceptItem->decode_header())
        {
            do_accept_body();
        }
    });
}

Server::Server(boost::asio::io_context &io_context, const boost::asio::ip::tcp::endpoint &endpoint):
    m_acceptor(io_context, endpoint)
{
}

void Server::accept()
{
    m_acceptor.async_accept(
        [this](boost::system::error_code ec, tcp::socket socket)
    {
        if (!ec)
        {
            std::cout << "连接成功"<<std::endl;
            //auto networkTransmission = new NetworkTransmission(std::move(socket));
            checkLoginAccount(std::move(socket));
        }
        accept();
    });
}
```

第十章 运行部署

10.1 登录界面



A screenshot of a login interface. It features a grey header bar with a small icon on the left. Below the header, there are two input fields: the first is labeled '帐号' (Account) and the second is labeled '密码' (Password). Below these fields is a grey button labeled '登录' (Login).

10.2 注册界面

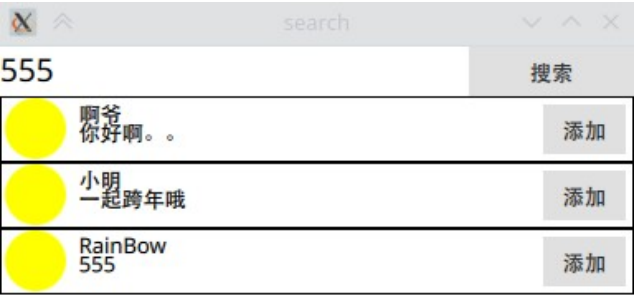


A screenshot of a registration interface. It features a grey header bar with a small icon on the left. Below the header, there are three input fields: the first is labeled '帐号' (Account), the second is labeled '昵称' (Nickname), and the third is labeled '密码' (Password). Below these fields is a grey button labeled '注册' (Register).

10.3 聊天



10.4 查找好友并添加



10.5 查找群并添加



10.6 处理添加请求



参考文献

- [1] 《Object-Oriented Systems Analysis and Design》By Simon Bennett、Steve McRobb and Ray Farmer
- [2] 《Use Case Modeling》By Kurt Bittner、Ian Spence
- [3] 《Object-Oriented Modeling and Design with UML》By Michael Blaha、James Runmbaugh
- [4] 《Managing Software Requirements A Use Case Approach》By Dean Leffingwell、Don Widrig