ourChat开发文档-软工组队

0. 组员名单

- 田田
- 彭扬达
- 金佳希
- 黄宇成
- 李子轩

1. 需求分析

1.1 实现功能

用户管理:

功能类型	简要描述
基本管理	用户注册、注销
用户认证	登录登出
	用户名密码、邮箱验证码认证
	为私聊/群聊设置二次验证
	验证身份后修改个人信息
好友关系	用户查找
	好友申请和删除
	好友列表展示及分组

在线会话:

功能类型	简要描述
聊天界面显示	文本信息发送
	参与人头像显示和气泡颜色区分
	发送时间、已读情况和未读计数显示
消息内容	多媒体信息（jpg, wav, mp4, pdf）内容发送
	URL链接格式化显示及单击跳转
	提及成员和提及反馈

消息扩展列表	右键弹出扩展列表
	消息撤回
	消息回复及跳转
	外语翻译
	多选消息及合并转发
	语音转文字
聊天记录	查看、条件筛选和删除记录
会话管理	设置消息免打扰和置顶会话
群聊功能	通过好友列表选取或以私聊为基础创建群聊
	群聊信息（群聊名称、群成员、群公告）展示
	添加群管理员、群主转让、移除群员、撤销消息、群公告设置
	邀请成员入群及成员自主退出群聊

1.2 用户故事

对于更详细的需求，通过以下用户故事进行描述：

用户管理

1. 作为一名用户，我希望能够用多种方式进行登录认证，以便于在我忘记用户名或密码时仍有登录账号的手段。
2. 作为一名用户，我希望能够对我的私聊和群聊设置二次验证，以便于更好的保护我的聊天记录和个人信息。
3. 作为一名用户，我希望能够修改我的用户名、密码、头像、邮箱等个人信息，以便我更好地对外展示自己的信息并满足特定场景下的需求，同时提高账户安全性。
4. 作为一名用户，我希望能够查找其他用户并浏览其基本信息，以便于添加他们为好友进行会话。
5. 作为一名用户，我希望能够添加好友和删除好友，以便于与我的好友进行私聊和分享信息。
6. 作为一名用户，我希望能够看到我的全部好友并对其进行分组，以便于更好的管理我的好友列表。
7. 作为一名用户，我希望能够自主注销我的账户，以便于自如地终止使用IM系统，并且我的个人信息、聊天记录等数据可以被安全删除，我的好友关系和群聊也能够被妥善处理。

在线会话通用

1. 作为一名用户，我希望能够在聊天界面中可以看到消息的发送时间、已读情况和未读计数，以便更好地掌握消息的状态。
2. 作为一名用户，我希望能够发送文本、图片、视频、语音、文件等消息，以便更好地表达自己的意见和想法，向聊天对象传达信息。
3. 作为一名用户，我希望能够在聊天界面中能够支持URL链接的格式化显示，并且能够单击链接跳转到相应网页，以便更好的识别连接以及更快捷地获取相关信息。

4. 作为一名用户，我希望能够在群聊中能够提及特定成员向其发送消息，以便在群聊中更好地引起他们的注意并获得响应。
5. 作为一名用户，我希望能够在群聊中提及特定成员后能够及时获得该成员是否已读的反馈，以便更好地掌握消息的状态。
6. 作为一名普通用户，我希望能够在合适的时间限制内撤回我所发送的消息，以便更好地把控我的聊天内容。
7. 作为一名用户，我希望能够回复特定消息，以便让聊天对象更好地了解我的表达内容。
8. 作为一名用户，我希望能够在聊天界面中进行外语翻译，以便更好地实现多语言交流。
9. 作为一名用户，我希望能够在聊天界面中选择多条消息，并支持将其合并转发，以便更快捷地转达相关信息。
10. 作为一名用户，我希望能够在聊天界面中能够将语音消息转换为文字消息，以便在特定场景下获取信息和保持沟通。
11. 作为一名用户，我希望能够查看、筛选和删除聊天记录，以便更好地了解和回顾历史聊天内容、提取相关信息。
12. 作为一名用户，我希望能够设置消息免打扰和置顶会话，以便更好地管理聊天记录和消息通知。

群聊需求

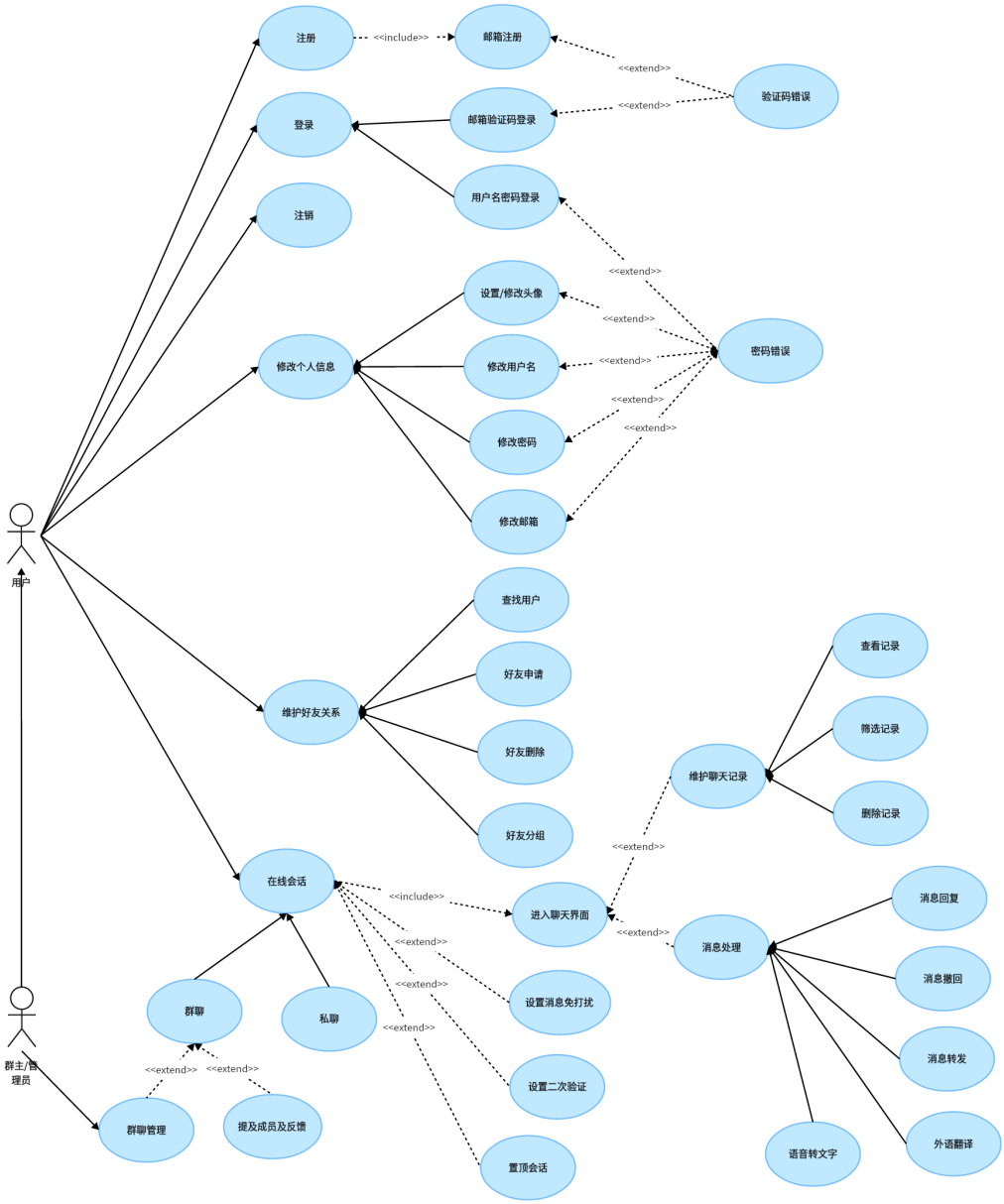
1. 作为群成员，我希望能够查看当前所在群聊的名称、群成员列表和群公告，以便了解群聊的基本情况。
2. 作为群成员，我希望能够邀请其他好友到当前群聊中，以便共同沟通。
3. 作为群成员，我希望能够自主退出当前所在群聊，以便不再接收该群聊消息。
4. 作为群主，我希望能够转让当前群聊的群主身份给其他成员，以便更好地管理当前群聊。
5. 作为群主，我希望能够添加其他成员为群管理员，以便让他们协助我管理当前群聊。
6. 作为群主/群管理员，我希望能够设置当前群聊的名称和群公告，以便即时更新群聊的基本信息。
7. 作为群主/群管理员，我希望能够移除当前群聊的某个成员，以便维护当前群聊。
8. 作为群主/群管理员，我希望能够撤销某成员发送的消息，以便维护当前群聊的聊天内容，避免不合适的消息显示在聊天界面中。
9. 作为群主/群管理员，我希望能够审核其他用户的入群请求，以便更好地管理群聊成员，避免不相关用户加入会话。

整体需求

1. 作为一名用户，我希望消息送达延迟低于200ms，以保证沟通的实时性和效率。
2. 作为一名用户，我希望消息送达保证可靠完整，以保证沟通的有效性。
3. 作为一名用户，我希望我在ourChat上发送和接收的消息能够被加密存储而不发生信息泄露，以保证个人信息和沟通的安全性。
4. 作为一名用户，我希望UI设计和布局合理，并且对操作有一定提示，以便快速找到相应功能，保证沟通的效率。

1.3 用例

此处以用例图的形式描述IM系统的主要功能。



上图描述了ourChat中用户注册注销、登录登出、个人信息编辑、好友列表维护、在线回话等功能，其中，在线会话是本系统的核心。

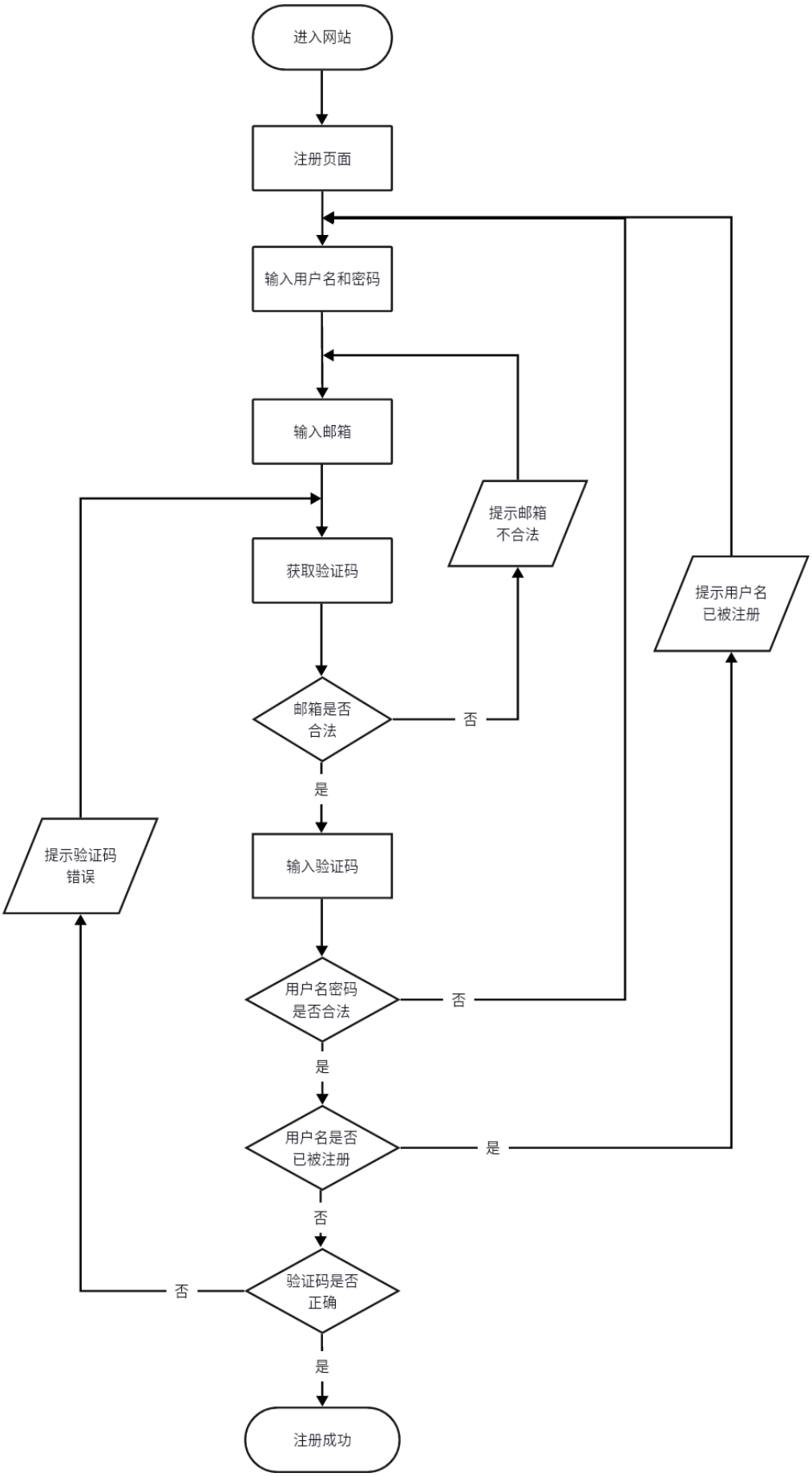
本系统提供完善的聊天界面显示，包括发送时间、已读（私聊）、已读成员列表（群聊）、未读计数等辅助信息显示，还包括转发、撤回和翻译消息以及处理聊天记录功能。在会话列表界面，用户可设置消息免打扰和会话二次验证。在此基础上，群聊部分实现群聊信息管理和成员管理。

1.4 交互流程

以下展示ourChat功能中的基本功能交互。

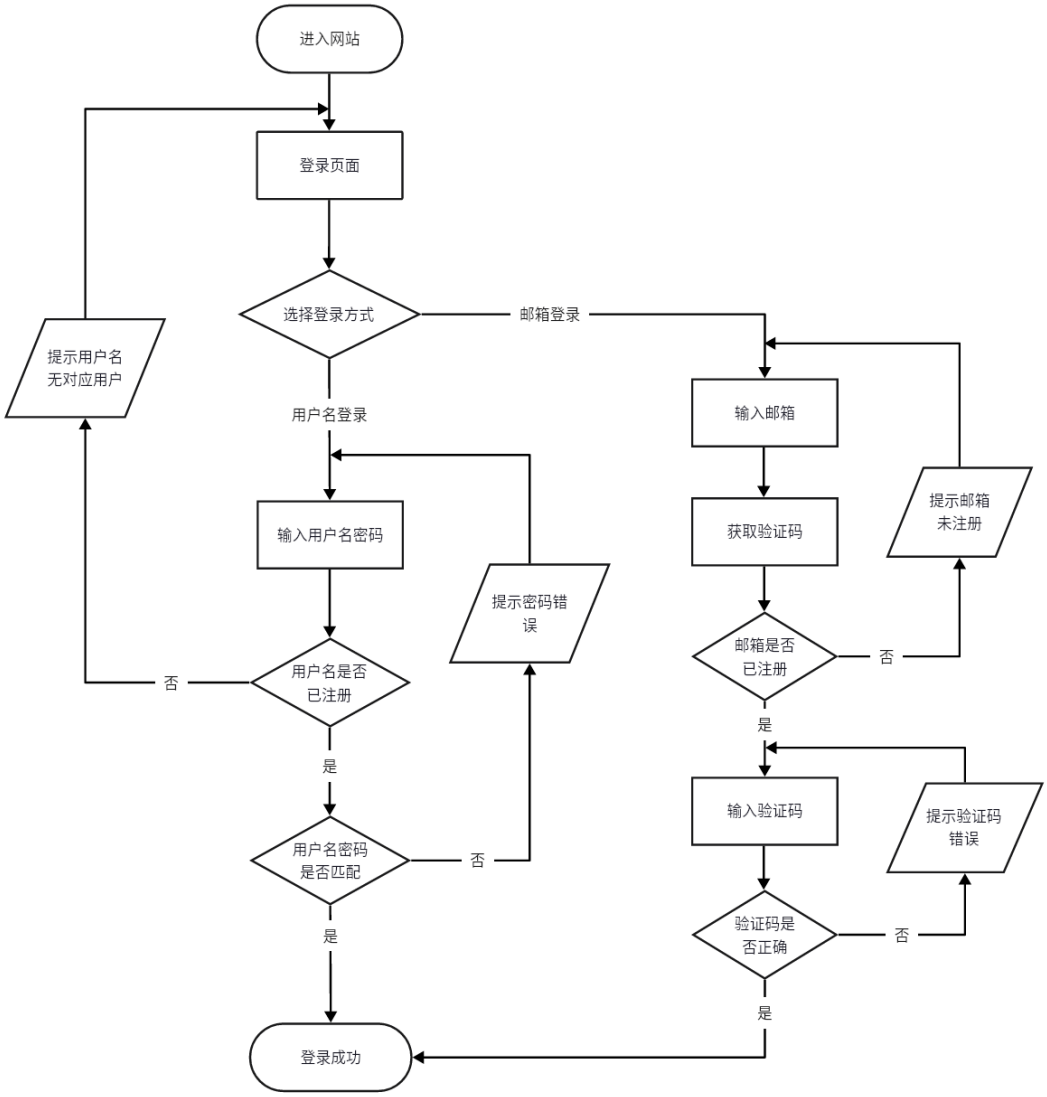
1.4.1 用户注册

用户输入用户名、密码和邮箱进行注册，服务端验证邮箱格式并发送验证码至用户邮箱，然后判断用户输入的验证码是否正确，之后服务端将成功注册的用户信息写入数据库中。其流程图如下：


进入网站
注册页面
输入用户名和密码
输入邮箱
获取验证码
邮箱是否合法
否
提示邮箱不合法
是
输入验证码
提示验证码错误
用户名密码是否合法
否
是
用户名是否已被注册
是
提示用户名已被注册
否
验证码是否正确
否
是
注册成功

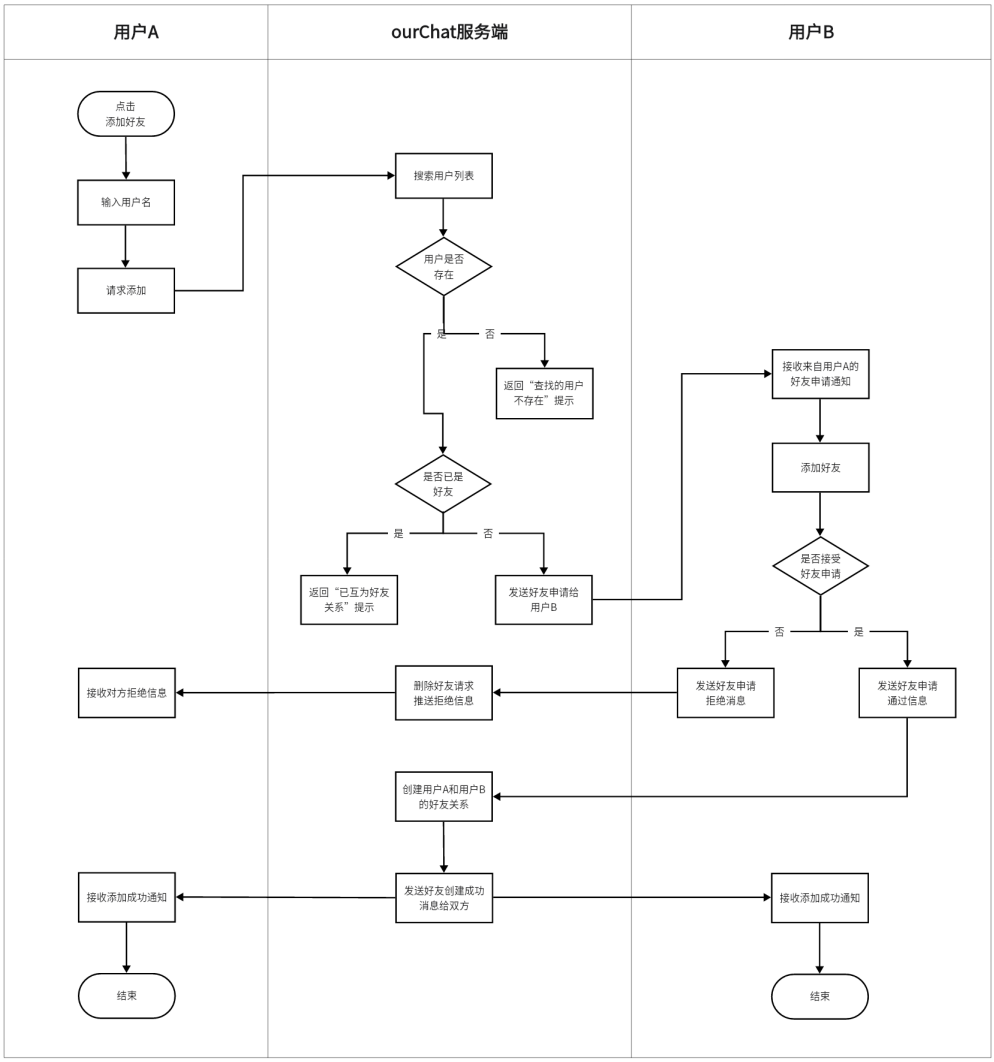
1.4.2 用户登录

ourChat为用户提供了两种登录方式：用户名密码登录和邮箱验证码登录，其流程图如下：



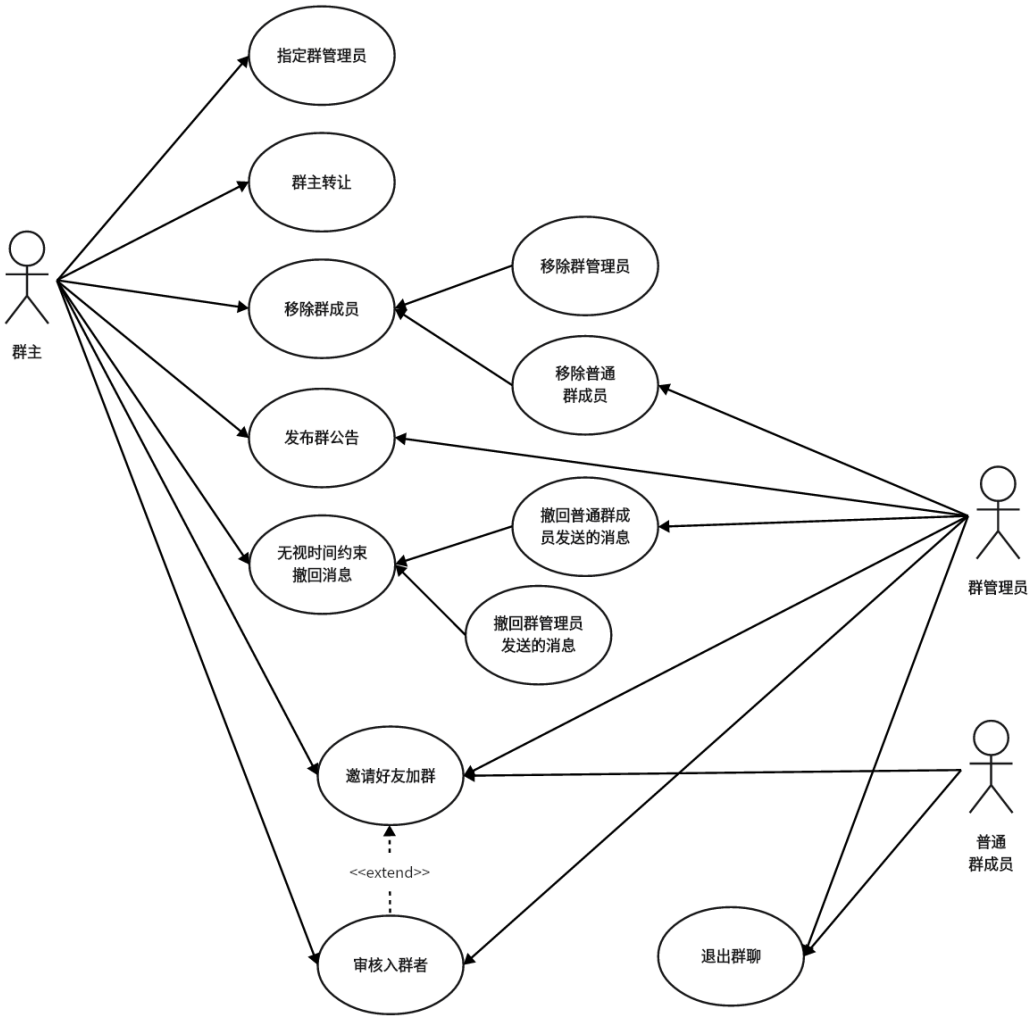
1.4.3 添加好友

通过查找用户名添加好友，服务端需先判断用户是否存在以及好友关系是否已存在，该过程的泳道图如下：



1.4.4 群聊管理

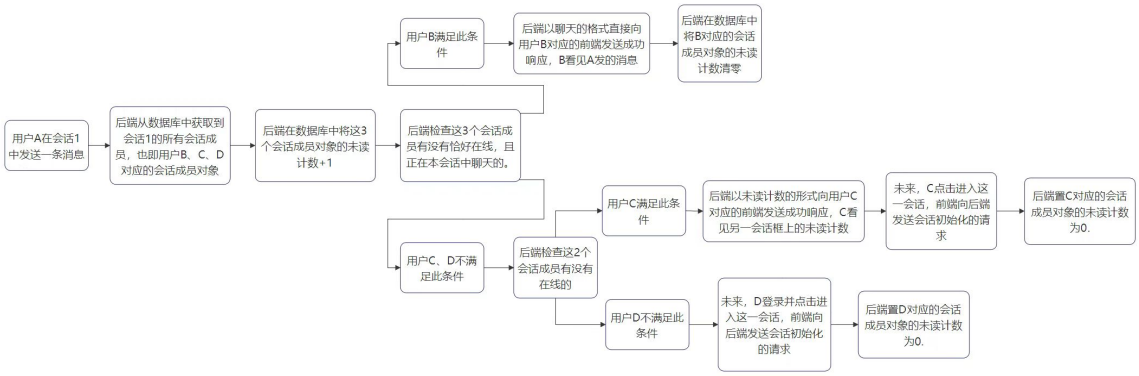
群成员包括群主、由群主指定的群管理员和普通群成员，其相关权限通过用例图表示如下：



1.4.5 未读计数

下图演示了ourChat实现未读计数的大致流程。

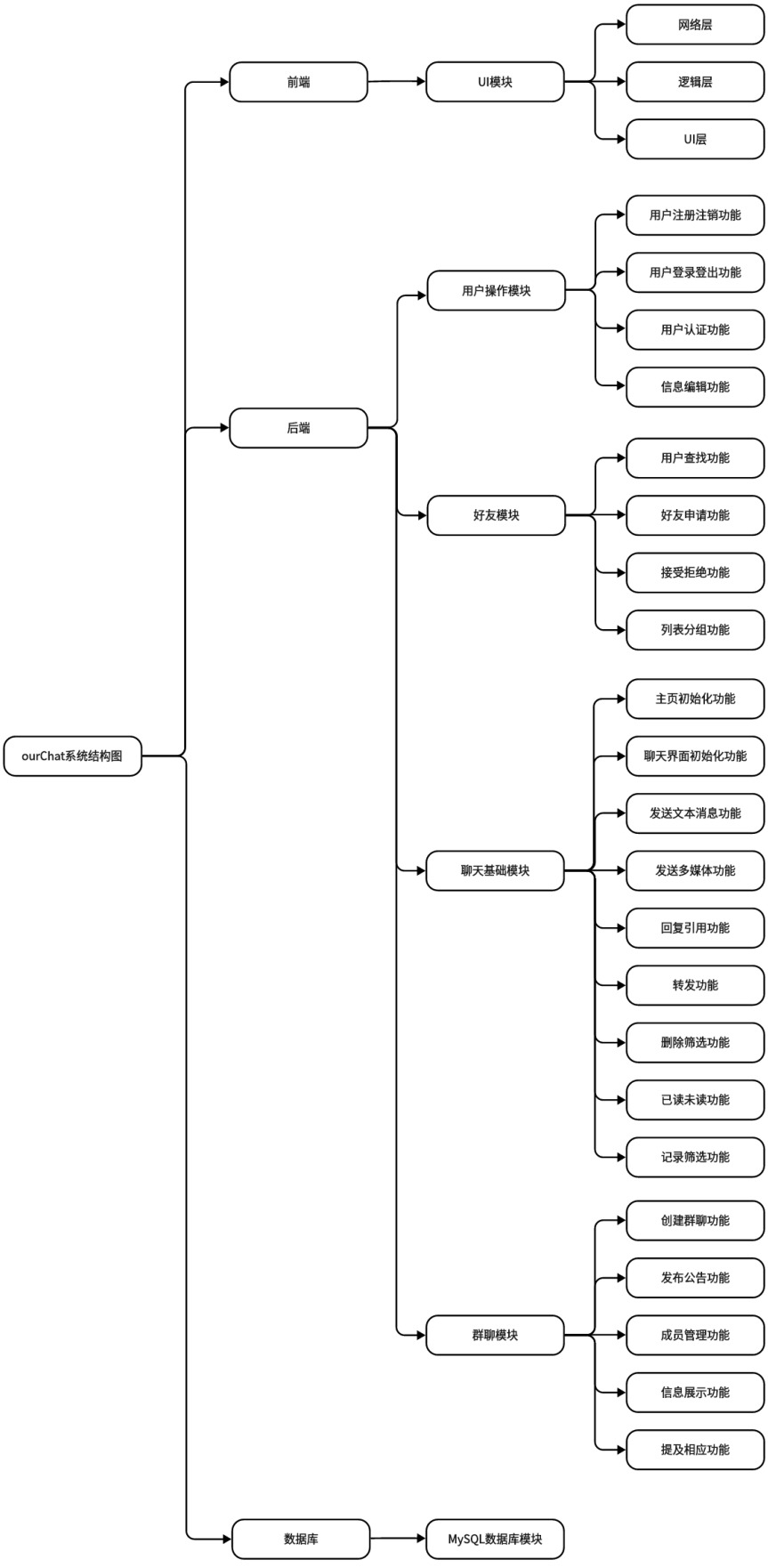
为了简单易懂且有代表性，我们假设有一个会话名为会话1，其中有A、B、C、D四人。



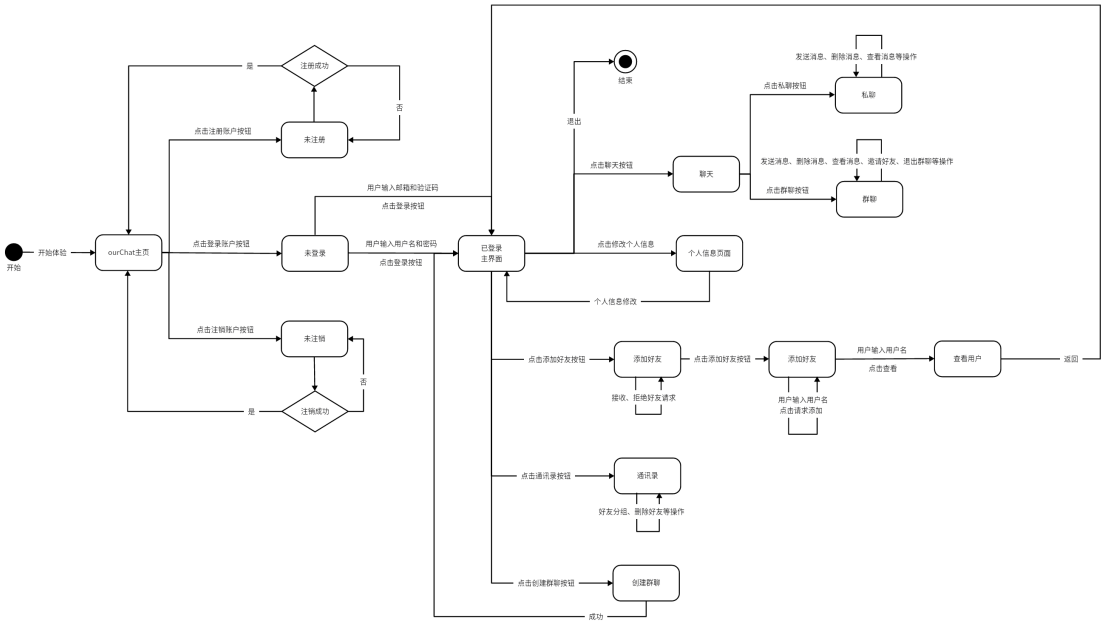
2. 模块设计

以下展示ourChat实现的前后端模块设计。

- 首先，ourChat整体的系统结构图如下：



- 其次，ourChat系统状态机图如下：



2.1 前端

2.1.1 与后端的通信

前端使用http协议和原生websocket与后端进行通信。我们建立了两个websocket长连接`socket_connection`和`friend_request_connection`，分别用于处理聊天和好友功能，http协议用于实现剩余的功能。

具体的通信接口将在API文档部分详细说明。

2.1.2 组件构建

- 前端使用了`next.js`框架实现搭建，文件结构如下：

```
📦src
 ┣ 📂pages
 ┃ ┣ 📂views
 ┃ ┃ ┣ 📂funcs
 ┃ ┃ ┃ ┣ 📜AddFriend.js
 ┃ ┃ ┃ ┣ 📜ChatBox.js
 ┃ ┃ ┃ ┣ 📜chatVerification.js
 ┃ ┃ ┃ ┣ 📜reportWebVitals.js
 ┃ ┃ ┃ ┗ 📜RequestInfo.js
 ┃ ┃ ┣ 📜App.js
 ┃ ┃ ┗ 📜Start.js
 ┃ ┣ 📜friendinfo.js
 ┃ ┣ 📜index.js
 ┃ ┣ 📜layout.js
 ┃ ┣ 📜login.js
 ┃ ┣ 📜logout.js
 ┃ ┣ 📜person.js
 ┃ ┣ 📜register.js
 ┃ ┗ 📜userinfo.js
 ┣ 📂tests
 ┃ ┗ 📜test.js
```

```
└ setupTests.js
```

我们接下来将逐文件介绍其中的组件和组件中的变量与可调用的函数。

2.1.2.1 Addfriend文件

- 该文件对应着好友功能的模块，包含 `Friend`、`Friendlist`、`Showfriendlist`和 `AddFriend`四个组件，`AddFriend`组件用于提供搜索好友、发送好友申请、查看好友申请的交互界面，`Friend`对应的是带有删除好友、添加聊天、添加标签以及删除标签功能的好友组件，`Friendlist`则是用来渲染Friend组件列表的父组件，`Showfriendlist`组件用于渲染`Friendlist`。
- `Friend`组件：用于对一个好友进行相关操作
  - 组件的状态变量
    - `name`：该好友的用户名
    - `visible`：功能下拉列表是否可见的布尔变量，初始为`false`
    - `input`：用于存储用户输入的标签名称
  - 可调用的函数
    - `handletagAdd`：参数为`name`，表示被进行操作的好友的用户名，功能为借助父组件的tagAdd函数为这个用户添加操作者输入的标签
    - `handletagDel`：参数为`name`，表示被进行操作的好友的用户名， 功能为借助父组件的tagDel函数为这个用户删去操作者输入的标签
- `Friendlist`组件
  - 将父组件传过来的friendlist通过map函数，转换为一个带有滑轮功能的Friend组件列表
- `Showfriendlist`组件
  - 将一个Friendlsit组件挂载到页面上，并传递一系列属性以及函数
- `AddFriend`组件
  - 是一个能打开添加好友窗口的函数组件
  - 组件的状态变量
    - `name`：记录想要添加的好友的名字
  - 函数
    - `viewFriend`：借助父组件传递属性的clickView函数查看特定用户
    - `nameChange`：记录用户输入的用户名

2.1.2.2 chatBox文件

- 该文件对应着聊天框的模块，包含`Tiji`、`ChatBubble`、`ChatBubbleQuote`、`VoiceToText`、`Translation`、`Message`、`Dialog`和`Chatbox`组件，`Tiji`组件用于渲染普通的文本信息，能够带有提及人特别显示及链接跳转、链接格式化的功能，`ChatBubble`对应的是渲染`Tiji`子组件的父组件，`ChatBubbleQuote`是用来渲染引用内容的组件，`VoiceToText`组件对应的是语音转文字功能，`Translation`组件则是对应的翻译功能，`Message`组件对应的是信息功能，负责渲染各种各样的不同信息类型的组件，`Dialog`则是用map方法将父组件传递的属性渲染成`Message`列表的组件，`Chatbox`组件则是用来渲染`Dialog`组件的。
- `Tiji`组件：用于处理文本信息中的提及以及链接格式化

    - 提及功能通过正则表达式来判断文本信息中的提及部分，将其转换为带有Link跳转的文本信息，其中的mentionUserInfo函数则是用来告知后端用户进行了提及
    - 链接格式化借助AutoLinkText库达到链接格式化的效果，
- `ChatBubble`组件
    - 渲染一个`Tiji`组件
- `ChatBubbleQuote`组件
    - 对应的是引用框内容，每一条消息都有对应的引用内容，普通的消息对应的引用内容是空字符串，此时不予显示，有回复内容的消息则显示引用框内容，引用框由该组件负责渲染，文本信息直接显示，非文本信息则不显示本来信息，只是告知是何种类型的信息。该组件提供回复跳转功能`scrollToMessage`，即点击回复信息内容可以使得滑轮滑动至回复信息进入视野框。
- `VoiceToText`组件
    - 提供语音转文字功能，采用http向后端请求对应接口，接收到内容返回后将其渲染出来
- `Tanslation`组件
    - 提供翻译功能，采用http向后端请求对应接口，接收到内容返回后将其渲染出来
- `Message`组件
    - 消息类组件
    - 组件的状态变量
        - `text`：消息内容，有可能是文本消息的内容，也有可能是多媒体消息的二进制字符串
        - `menuvisible`：消息辅助菜单栏的可见与否的布尔控制变量
        - `id`：消息id
        - `iftranslate`：是否翻译的布尔控制变量
        - `iftransform`：是否语音转文字的控制变量
        - `showQuote`：是否显示引用框的布尔控制变量
        - `quoteText`：引用的内容字符串
    - 函数
        - `setBase64Format`：将多媒体内容转换为base64字符串的函数
        - `sendername`：从消息内容中提取发送者信息的函数
        - `allocate`：该函数是对消息进行分类，对应不同展示功能的函数
    - 消息自带的辅助功能共有六个：删除、撤回、引用、转发、翻译、语音转文字，前四个函数均是通过父组件传递的函数实现的，后面两个则是通过http向后端请求实现
- `Dialog`组件
    - 消息列表类组件
    - 通过map方法，将person界面传递过来的各种属性传递给`Message`组件列表，依次包括提醒后端提及的函数、删除函数、撤回函数、引用函数、转发函数、消息内容字符串、消息时间、已读列表、头像、消息id、消息类型、引用消息内容及id、是否是自己的消息的布尔变量、能够提及的名字列表、点击引用框跳转函数、以及是否要提醒后端提及的布尔标志变量，即
        - `mentionUserInfo={this.props.mentionUserInfo}//提醒后端提及的函数`
        - `onDeleteClick={(msg) => this.props.onDeleteClick(msg)}//删除对应消息函数`
        - `onWithdrawClick={(msg) => this.props.onWithdrawClick(msg)}//撤回对应消息函数`

        - `mark={this.props.mark}//引用对应消息函数`
        - `forward={this.props.forward}//转发对应消息函数`
        - `item={item}//消息内容字符串`
        - `time={new Date().toLocaleTimeString('en-US', { hour: 'numeric', minute: 'numeric' })}//消息时间`
        - `havereadlist={JSON.stringify(displayedItemsRead[index])}//已读列表`
        - `avatar={displayedItemsAvatar[index]}//用户头像`
        - `id={displayedItemsId[index]}//消息id`
        - `type={displayedItemsType[index]}//消息类型`
        - `markid={displayedItemsId[displayedItems.indexOf(displayedItemsMark[index].split("//回复次数//")[0])]}//引用消息id`
        - `markmessage={(displayedItemsMark[index])}//引用消息内容`
        - `ifmine={item.indexOf(this.props.name + ":") !== 0 ? false : true}//是否是自己的消息的布尔变量`
        - `tijilist={this.props.tijilist}//能够提及的名字列表，供正则表达式构造`
        - `scrollToMessage={this.scrollToMessage}//点击引用框跳转函数`
        - `iftiji={(index === (displayedItems.length - 1)) ? true : false}//是否要提醒后端提及的布尔标志变量,只对最新消息起效`
- `Chatbox`组件
    - 该组件负责的就是渲染一个`Dialig`组件，并传递给它需要的属性，即
        - `onDeleteClick={(msg) => props.onDeleteClick(msg)}//删除消息对应的函数`
        - `onWithdrawClick={(msg) => props.onWithdrawClick(msg)}//撤回消息对应的函数`
        - `historymessage={props.historymessage}//历史消息列表`
        - `historymessageid={props.historymessageid}//历史消息id列表`
        - `name={props.name}//当前用户的用户名`
        - `mark={props.mark}//引用消息对应的函数`
        - `forward={props.forward}//转发消息对应的函数`
        - `tijilist={props.tijilist}//可以提及的成员名列表`
        - `markmessage={props.markmessage}//引用的消息内容列表`
        - `messagetype={props.historymessagetype}//引用的消息类型的列表`
        - `mentionUserInfo={props.mentionUserInfo}//向后端请求提及用户的函数`
        - `timelist={props.timelist}//消息时间列表`
        - `avatarlist={props.avatarlist}//消息头像列表`
        - `talkingobject={props.talkingobject}//当前聊天名字`
        - `readlist={props.readlist}//消息已读列表的列表`
    - 这里传递的函数与之前介绍的一样，传递的属性则大多是所有消息属性的列表，`Dialog`组件则是将列表中对应秩的属性发送给下属的`Message`，三者其实是祖、父、子组件的关系

2.1.2.3 chatVerification文件

- 该文件对应着聊天框二次验证的模块，包含 `Dialog` 和 `Chatverification` 组件，`Dialog` 组件用于渲染一个掩盖聊天框的加密框，要求用户输入密码以打开聊天框，其中输入密码后会 向后端对应接口以http的方式询问，验证正确后返回消除加密框，`chatVerification` 组件则是将 `dialog` 组件渲染出来并向其传递一系列属性
- `Dialog` 组件：
  - 状态变量：
    - `password`：用户输入的二次验证密码
    - `curname`：用户的用户名
    - `id`：聊天的id

2.1.2.4 reportWebVitals.js

- 该文件是框架自带的检查网络问题的报告文件

2.1.2.5 RequestInfo.js

- 该文件对应着收到的好友申请模块，包含 一个 `Showrequestlist` 组件：用于展示收到的好友申请列表
- `Showrequestlist` 组件通过map方法将得到的属性requestlist渲染成一个带有接受和拒绝功能的申请列表
- 函数
  - 接受申请函数:通过父组件传递过来的接收函数实现，即 `AcceptFriendRequest`
  - 拒绝申请函数:通过父组件传递过来的拒绝函数实现，即 `RefuseFriendRequest`

2.1.2.6 App.js

- 该文件对应着根页面，渲染了一个 `Start.js` 文件的 `Start` 组件

2.1.2.7 Start.js

- 该文件对应着开始界面模块
- 主要介绍了前后端开发人员的基本信息
- 提供了一个进入基本功能页面layout的按钮，按钮点击运行 `goLayoutHandler` 函数

2.1.2.8 friendinfo.js

- 该文件对应着他人信息查看功能模块，为提及跳转、搜索好友等功能提供接口，包含 一个 `Friend` 组件：用于展示向后端请求到的用户信息
- `Friend` 组件
  - 通过向后端对应接口发送http申请，得到正确数据返回后将其渲染出来，包括用户名、id、邮箱、头像
  - 状态变量：

    - `name` 用户名
    - `userData` 用户数据
    - `loading` 加载标识符

2.1.2.9 index.js

- 该文件对应着根页面，渲染了一个 `App.js` 文件里的 `App` 组件

2.1.2.10 layout.js

- 该文件对应着基本功能界面，包括一个返回根页面的按钮以及登录、注销、注册三种功能——通过三个按钮支持三个具体功能页面的跳转
- 按钮
    - 返回首页：回到根页面，`startHandler`
    - 登录：去往登陆页面，`logInHandler`
    - 注册：去往注册页面，`registerHandler`
    - 注销：去往注销页面，`logOutHandle`

2.1.2.11 login.js

- 该文件对应着登录界面，包括账号密码登陆方法以及邮箱验证码登陆方法
- 状态变量：
    - `name` 用户名
    - `password` 用户密码
    - `type` 登录标识符，0代表账户密码登录，1代表邮箱验证码登录
    - `email` 邮箱账号
    - `emailpassword` 邮箱验证码
- 函数：
    - `returnToLayOutPage`：返回基本功能界面的函数
    - `nameChange`：记录用户输入的账户名
    - `sendEmailPassword`：向后端请求发送邮箱验证码
    - `passwordChange`：记录一下用户输入的密码
    - `emailPasswordChange`：记录用户输入的邮箱验证码
    - `emailChange`：记录用户输入的邮箱
    - `logIn`：向后端请求验证登录，根据类型选取不同的函数验证
    - `NPlogIn`：向后端请求验证账户密码登录
    - `EmaillogIn`：向后端请求邮箱验证登录

2.1.2.12 logout.js

- 该文件对应着注销界面， 要求用户输入账号名以及密码
- 状态变量：
  - `name`用户名
  - `password`用户密码
- 函数：
  - `returnToLayOutPage`：返回基本功能界面的函数
  - `nameChange`：记录用户输入的账户名
  - `passwordChange`：记录一下用户输入的密码
  - `logOut`：向后端请求注销账户

2.1.2.13 person 文件

- 这个文件用于展示用户登录后的主体界面，汇集了绝大多数的功能在这里。
- 该文件包含`NewFriendModal`、`ChatList`、`ChatObject`和`PersonalPage`四个组件。`NewFriendModal`组件用于提供处理其他用户发送来的好友申请的界面，`ChatList`组件用于渲染展示自己的聊天列表，`ChatObject`组件用于展示单个聊天（不是聊天界面）并对其进行置顶、加密、免打扰和删除这四个操作，`PersonalPage`用于展示当前的聊天界面，集成了聊天相关的所有功能。
- `NewFriendModal`组件：提供处理其他用户发送来的好友申请的界面，会在`PersonalPage`组件中被调用
  - 组件中的变量：这所有的变量组成`props`被传入组件
    - `ifvisible`：目前陌生人的信息是否可见
    - `addvisible`：控制添加朋友的界面是否可见
    - `clickView`：点击跳转到查看好友信息的界面
    - `setclose`：执行时将`addvisible`状态设置为`false`，设置是否关闭界面
    - `requestlist`：收到的好友申请的列表
    - `AcceptFriendRequest`：接受朋友请求的回调函数
    - `RefuseFriendRequest`：拒绝朋友请求的回调函数
    - `onClose`：执行时将`newfriendvisible`状态设置为`false`
    - `addFriend`：添加朋友的回调函数
    - `onClick`：点击事件的回调函数，执行时将`addvisible`状态设置为`true`。
    - `name`：当前用户的用户名
  - 可调用的函数
    - 没有可以调用的函数，这个组件会将输入的变量整合成显示处理好友列表的界面并return回去
- `ChatList`组件：负责渲染自己的聊天列表
  - 组件中的变量
    - `chatList`：在聊天的所有对象的列表
  - 可调用的函数

        - 无，会将 `chatList` 中的所有对象渲染成聊天列表并return
- `ChatObject` 组件：展示单个聊天并对其进行置顶、加密、免打扰和删除这四个操作
    - 组件中的变量：通过 `props` 输入这些变量
        - `number`：当前聊天的 `id`
        - `verify`：向后端通过 `http` 协议请求为当前聊天添加二次验证的函数
        - `handledel`：把当前的这个聊天移除
        - `deleteverify`：取消当前聊天的二次验证
        - `ifveri`：当前聊天是否需要进行二次验证
        - `ifsilent`：当前聊天是否被免打扰
        - `silent`：把当前聊天设定为免打扰
        - `deletesilent`：把当前聊天解除免打扰
        - `objectname`：当前聊天的名称
        - `up`：把当前聊天进行置顶
        - `deleteup`：把当前聊天取消置顶
        - `ifup`：记录当前聊天是否需要进行置顶
        - `onClick`：当前聊天被点击后调用，初始化当前聊天的基本信息（聊天对应 `id` 和聊天的名称）
    - 可调用的函数
        - 无，会把输入的所有变量渲染成可以对单个聊天进行操作的界面并return
- `PersonalPage` 组件：展示当前的聊天界面，集成了聊天相关的所有功能，非常复杂
    - 组件中的变量
        - `ifchat`：这个聊天是否处于聊天状态
        - `curname`：用户的用户名
        - `collapsed`：目前侧边栏是否折叠
        - `currentchatid`：当前聊天的 `id`
        - `talkingobject`：当前聊天的名称
        - `ifgroup`：当前聊天是否是群聊
        - `curgroupnotice`：（如果是群聊的话）群聊的所有群公告
        - `curgrouplist`：（如果是群聊的话）群聊的成员列表
        - `message`：目前在聊天的输入框输入的信息
        - `chat_visible`：聊天界面是否可见
        - `friendlist`：用户的好友列表
        - `taglist`：标签列表（给用户分组）
        - `chatlist`：所有的聊天
        - `chatidlist`：所有的聊天的 `id`
        - `messagenumber`：所有的消息的 `id`
        - `verilist`：需要进行二次验证的聊天

        - `silent_list`：需要被消息免打扰的聊天
        - `veri_visible`：是否显示二次验证
        - `historymessage`：历史记录的消息（按照`id`顺序）
        - `historymessagetype`：历史记录消息类型（多媒体or文本）
        - `historymessageid`：历史记录消息`id`
        - `markmessagelist`：被选中的消息（转发的时候用）
        - `messageavatarlist`：发送消息的所有人的头像（用base64字符串存储）
        - `messagetimelist`：所有消息被发送的时间
        - `readlist`：被对方已读的消息
        - `showchatlist`：所有聊天的列表是否可见
        - `showalllist`：自己好友列表对应的界面是否可见
        - `newfriendvisible`：好友申请相关的界面是否可见
        - `addvisible`：添加好友的按钮是否可见
        - `upend`：所有被置顶聊天的末尾位置
        - `ifuserinfo`：是否跳转到了展示用户信息的界面（用来判断当前组件生命周期结束是正常的页面跳转导致的，还是退出or窗口关闭导致的）
        - `requestlist`：收到的好友请求列表
        - `markmessage`：被选中要回复的信息
        - `markmessageid`：被选中要回复的信息的`id`
        - `addmanagerlist`：群聊中选择的要请求添加成为管理员的成员的用户们
        - `ifforward`：目前要发送的消息是否是转发类型
        - `forwardidlist`：被转发的所有的信息的`id`
        - `forwardobjectids`：接受转发的信息的所有聊天的`id`
        - `grouplist`：存储创建群聊的时候选中的所有成员
        - `ifviewgroup`：是否查看群聊
        - `searhisvisible`：搜索历史记录的交互界面是否可见（消息筛选）
        - `selectedoption`：消息筛选中搜索的类型（时间/消息类型/发送人）
        - `searchbody`：用户输入的搜索内容
        - `searlist`：后端返回的搜索到的结果
        - `announcementContent`：当前最新的群公告的内容
        - `myavatar`：当前用户的头像
        - `socket_connection`：聊天的`websocket`连接
        - `friend_request_connection`：好友的`websocket`连接
    - 函数：这些函数会自动调用上面的变量，所以很多不需要传入参数
        - `shouldComponentUpdate`：参数为下一个更新的`nextProps`和`nextState`（两个全局变量的集合）。判断需要重新渲染哪些组件，是框架自带的函数，会自动执行，不需要调用。

- `componentWillUnmount`：没有参数，是框架自带的函数，在组件生命周期终止时执行。会按照API文档中`log_out`部分的规定，发送目前当前用户最后的设定信息（聊天置顶顺序，是否免打扰等）给后端，并关闭两个`websocket`连接。
- `componentDidMount`：没有参数，无需调用，是初始化界面设定的函数，会在这个组件生命周期开始时被自动执行，创建两个`websocket`连接，以及监听后端的成功响应和失败响应，然后根据响应内容修改上面声明的全局变量，变量被修改后会通过`shouldComponentUpdate`调用之后声明的具体执行函数重新渲染组件，把更新的内容展示出来。**这里的具体操作会在下一部分详细解释。**这里还会向后端自动通过`http`协议请求当前用户的好友列表以及设定展示当前用户的头像。
- `mentionUserInfo`：参数`name_list`为提及的所有用户的用户名的列表。这个函数会向后端报告，当前用户请求在当前聊天中提及参数中声明的所有用户。
- `handleAddFriendGroup`：参数`members`是当前群聊中所有的成员，函数会渲染一个交互界面，让用户可以在当前群聊的所有用户中选择向谁发送好友申请。
- `searchChatHistory`：参数`check_type`为搜索的类型，`check_body`为搜索的内容。这个函数会向后端发送请求体，请求在当前用户的所有聊天中搜索类型为`check_type`内容为`check_body`的信息。
- `sendMediaMessage`：参数`media_content`为多媒体信息的内容，`media_type`为多媒体信息的类型，`chat_id`为当前聊天的`id`。这个函数会向后端发送请求体，告知当前用户想在当前聊天中发送一条多媒体信息。
- `withdrawGroupMessage`：参数`msg_id`为想要撤回的那条消息的`id`。这个函数会向后端发送请求体，告知当前用户想在当前聊天中撤回一条信息。
- `acceptInvite`：参数`id`为当前聊天的`id`，`invitor_name`为发出邀请的人的用户名，`invited_name`为被邀请的用户的用户名。这个函数会向后端发送请求体，告知后端当前用户同意了请求体中声明的入群请求。
- `rejectInvite`：参数`id`为当前聊天的`id`，`invitor_name`为发出邀请的人的用户名，`invited_name`为被邀请的用户的用户名。这个函数会向后端发送请求体，告知后端当前用户拒绝了请求体中声明的入群请求。
- `groupWithidraw`：没有参数，这个函数会向后端发送请求体，告知当前用户退出了当前群聊。
- `groupRequest`：没有参数，这个函数会向后端发送请求体，请求所有当前用户作为群主的群收到的所有没有处理的入群申请。
- `ifveri`：参数`id`为想要判断是否需要二次验证的聊天的`id`。这个函数会判断聊天是否需要进行二次验证。
- `handleclick`：参数`name`为聊天的名称，`id`为当前聊天的`id`。这个函数会向后端发送请求体，请求当前聊天的所有历史信息来渲染显示。
- `groupInfo`：没有参数，这个函数会向后端发送请求体，请求当前聊天（认为是群聊）的所有信息（成员、群公告...）。
- `handleRightClickofChatObject`：无需调用，是框架自带的函数，用来在用户右键点击的时候改为显示我们构建的界面，而不是浏览器正常有右键点击应该显示的界面。
- `transferOwner`：参数`new_owner`为请求设定为群主的用户的用户名，这个函数会向后端发送请求体，请求把用户名为`new_owner`设定为群主。
- `messageChange`：函数无需调用，会监听聊天界面消息输入框的变化，并将用户输入的值赋给上面的全局变量。设定0.3s调用一次，防止页面过于频繁渲染影响体验。

- `AcceptFriendRequest`：参数 `name` 为发送好友申请的人，这个函数会向后端发送请求体，告知后端当前用户同意了对方发来的好友申请。
- `RefuseFriendRequest`：参数 `name` 为发送好友申请的人，这个函数会向后端发送请求体，告知后端当前用户拒绝了对方发来的好友申请。
- `deletemessage`：参数 `msgid` 为被删除的信息的 `id`，函数会向后端发送请求体，告知后端当前用户删除了 `id` 为 `msgid` 的信息。
- `withdrawmessage`：参数 `msgid` 为请求撤回的信息的 `id`，函数会向后端发送请求体，向后端请求撤回 `id` 为 `msgid` 的信息。
- `permit`：参数 `name` 为聊天的名称，`id` 为当前聊天的 `id`。这个函数和 `handleclick` 的作用相同（就是直接调用 `handleclick`），是开发中沟通失误导致，修改时只需要修改 `handleclick`。
- `addChat`：参数 `friend_name` 为请求里添加聊天的用户名，这个函数会向后端发送请求体，请求当前用户和 `friend_name` 用户的私聊的信息。这个函数可以在用户删除了和 `friend_name` 的私聊后，又想和对方发起聊天时使用。
- `pushChat`：作用和 `addChat` 相同，会直接调用 `addChat`，是开发中沟通失误导致，如果要修改，只需要直接修改 `addChat`。
- `sendMessage`：没有参数，向后端发送请求体，告知后端当前用户在当前聊天中发送了一条信息（全局变量 `message` 存储的）。会根据这条消息是否有引用（功能表中要求的回复别人发送的消息）调用不同的函数执行。
- `friendGroupDel`：参数 `friend_name` 为被操作的好友的用户名，`tag` 为标签的名称。函数会向后端发送请求体，告知后端当前用户请求把好友 `friend_name` 从标签 `tag` 中删除。
- `friendGroup`：参数 `friend_name` 为被操作的好友的用户名，`tag` 为标签的名称。函数会向后端发送请求体，告知后端当前用户请求把好友 `friend_name` 移动到标签 `tag` 中。
- `replyMessage`：参数 `msg_body` 为被回复的信息的内容，`reply_msg_body` 为当前用户发送的回复的信息的内容，`msg_id` 为被回复的信息的 `id`。函数会向后端发送请求体，告知后端当前用户回复了一条信息。
- `messageForward`：参数 `msg_id_list` 包含了所有被转发的信息的 `id`。函数会向后端发送请求体，告知后端当前用户想把这些消息转发到全局变量 `forwardpbjectids` 对应的所有聊天中。
- `deleteFriend`：参数 `name` 为请求删除的好友的用户名。函数会向后端发送请求体，告知后端当前用户想删除和 `name` 的好友关系。
- `addMem`：参数 `name` 为想要被邀请进群的用户的用户名。函数会向后端发送请求体，告知后端当前用户想把 `name` 对应的用户拉到当前群聊中（后端会给群主发送响应告知这一信息）。
- `setManager`：参数 `id` 为聊天的 `id`，`list` 为请求添加的群管理员的 `list`。函数会向后端发送请求体，告知后端当前用户想把这些用户设定为当前的群聊的群管理员。
- `addFriend`：参数 `name` 为接受好友申请的用户，函数会向后端发送请求体，告知后端当前用户想向 `name` 对应的用户发送好友申请。
- `toggle`：没有参数，改变目前侧边栏是否折叠的状态。
- `handleMenuSignOutClick`：没有参数，是登出时调用的函数，跳转到初始页面。

- `signOut`：登出时调用的函数，和 `handleMenuSignOutClick` 作用相同，是开发时沟通失误导致的。如果想要修改函数，修改这个函数即可。
- `clickView`：参数 `name` 为想要查看信息的用户的用户名，函数会跳转到 `friendinfo` 界面，调用那个文件中的组件来显示 `name` 对应的用户的相关信息。
- `handleMenuInfoClick`：和 `userinfo` 函数的作用相同，是开发时沟通失误导致的。如果想要修改函数，修改 `userinfo` 函数即可。
- `userInfo`：没有参数，跳转到 `userinfo` 界面来查看修改当前用户的个人信息，会调用对应的组件来实现。
- `renderChatObject`：参数 `name` 为聊天的名称，`id` 为聊天的 `id`。函数会调用 `Chatobject` 组件，展示 `id` 对应的聊天并让用户可以对其进行置顶、加密、免打扰和删除这四个操作。
- `showChatList`：没有参数，会通过修改全局变量把聊天列表设定成显示出来。
- `showAllList`：没有参数，会向后端请求当前用户的好友列表，然后展示好友列表。
- `isStringSizeExceeded`：参数 `str` 是被判断的 `base64` 字符串。函数会判断这个字符串的内容是否超过 `4MB`。
- `handleUpload`：参数 `file` 是上传多媒体文件时传入的内容，函数会把它转为 `base64` 字符串，判断大小是否超标（如果超标就转化成文本信息“【很抱歉，这条多媒体信息大小过大，为了保证网页的运行流畅性，我们将不予显示渲染您发送的内容。】”），然后发送给后端，告知当前用户发送了一条多媒体信息。
- `menu`：没有参数，会返回元件对应界面的两个跳转按钮“修改个人信息”和“退出”。
- `chatList`：没有参数，会渲染全局变量中存储的好友列表并返回。
- `handleNewFriendClick`：没有参数，在用户想要处理好友申请时调用，把全局变量中决定是否显示处理好友申请界面的全局变量改成 `true` 并隐藏其他界面。
- `mark`：参数 `msg_id` 为选中的信息的 `id`，用来在回复功能中使用，通过 `id` 找到回复的信息的内容，并存储在全局变量 `markmessage` 中。
- `forward`：参数 `msgid` 是要转发的一条信息的 `id`，函数会把这条信息的 `id` 添加到全局变量 `forwardidlist` 中。
- `handleForwardSend`：和 `messageForward` 函数的作用相同，是开发中沟通失误导致的。如果想修改内容，修改 `messageForward` 函数即可。
- `handleCheckboxChange`：参数 `id` 代表想要把消息转发给谁，函数会把这个人的信息存储到全局变量中，以供 `handleForward` 函数调用。
- `handleForward`：参数 `msgidlist` 是要转发的所有信息的 `id`，函数会渲染一个交互界面提供给用户，让用户可以选择把消息转发到哪个聊天中，然后函数会调用 `handleForwardSend` 函数执行向后端请求转发的步骤。
- `requestInfo`：参数 `id` 是请求的群聊的 `id`，函数会向后端发送请求体，请求这个群的成员列表和当前用户的好友列表。
- `sendGroupAnnouncement`：参数 `id` 是群聊的 `id`，`content` 是群公告的内容。函数会向后端发送请求体，告知后端当前用户想在群聊中发送一条群公告。
- `handleGroupSend`：参数 `grouplist` 是所有成员的列表，`group_name` 是群的名称。函数会向后端发送请求体，告知后端当前用户想要创建一个群聊。
- `handleGroupCheckboxChange`：参数 `friend` 是所有被选中的成员列表，函数会把他们存储到 `grouplist` 全局变量中，来让别的函数调用。

- `handleManagerCheckboxChange`：参数 `friend` 是所有被选中要成为管理员的成员列表，函数会把他们存储到 `addmanagerlist` 全局变量中，来让别的函数调用。
- `requestAvatar`：参数 `name` 是用户名，函数会向后端发送请求体，请求这个用户的头像对应的 `base64` 字符串。
- `handleindirectGroup`：参数 `friend` 是所有被选中的用户的列表，在用户创建群聊时选择了基于私聊创建时调用，返回用户选择了哪些成员的交互界面。
- `direct`：没有参数，用户创建群聊时选择了直接创建时调用，会返回交互界面，让用户选择拉取哪些成员，并把结果存储在全局变量中。
- `indirect`：没有参数，用户创建群聊时选择了基于私聊创建时调用，会返回交互界面，让用户选择拉取哪些成员，并把结果存储在全局变量中。
- `handleGroupClick`：没有参数，返回一个交互列表，让用户可以选择是用直接拉取的方式还是基于私聊创建的方式创建群聊，然后根据用户选择的结果调用函数执行。
- `handlechOk`：没有参数，用来处理搜索历史记录中的Cancel按钮被点击之后的触发，把相关的全局变量设定成退出这个搜索界面之后的状态。
- `handlechCancel`：没有参数，用来处理搜索历史记录中的Cancel按钮被点击之后的触发，把相关的全局变量设定成退出这个搜索界面之后的状态。和OK按钮的效果其实相同。
- `handlechChange`：用于存储历史记录搜索的类型（时间/发送者/消息类型），没有参数，会存储到全局变量中。
- `handleSearchHistory`：没有参数，通过修改全局变量把历史记录的搜索框显示出来，在选择进行搜索历史记录操作时被调用。
- `handlechinputChange`：用于存储历史记录搜索的内容（时间/发送者/消息类型），没有参数，会存储到全局变量中。
- `groupNotice`：没有参数，返回一个发布群公告的交互界面，把用户输入的信息存储到全局变量中，并通过 `sendGroupAnnouncement` 函数告知后端。
- `transferGroupOwner`：没有参数，返回一个选择把群主转让给谁的交互界面，把用户选择的信息存储到全局变量中，然后通过 `transferOwner` 函数告知后端。
- `delMem`：参数 `name` 为请求从群中删除的成员的用户名，函数会向后端发送请求体，告知后端当前用户想从当前群聊中移除这个成员。
- `addGroupMem`：没有参数，返回一个让用户选择把谁拉入当前群聊的交互界面，把用户选择的成员存储在全局变量中，通过调用 `addMem` 函数发送给后端。
- `delGroupMem`：没有参数，返回一个让用户选择把谁从当前群聊中移除的交互界面，把用户选择的成员存储在全局变量中，通过调用 `delMem` 函数发送给后端。
- `addManager`：没有参数，返回一个让用户选择把谁设定为当前群聊的管理员的交互界面，把用户选择的成员存储在全局变量中，通过调用 `setManager` 函数发送给后端。
- `memberManage`：没有参数，返回一个让用户选择对当前群聊进行什么操作的交互界面，可以选择邀请成员、移除成员、转让群主、添加管理员、查看申请和退出群聊这六个操作，函数会调用对应的函数去提供新的交互界面。
- `handleGroupProps`：没有参数，调用 `requestInfo` 函数请求当前群聊的成员列表和当前用户的好友列表，是对 `requestInfo` 函数的修饰器。
- `tagfriend`：参数 `tag` 为标签的名称，函数会向后端发送请求体，向后端请求当前用户所有标签为 `tag` 的好友，并将他们存储到全局变量中。

- `handleTag`：没有参数，会返回一个交互界面，让用户查看自己设置的所有标签。
- `setBase64Format`：参数 `base64` 是 `base64` 字符串，函数会去除这个字符串之前的发送人名称，把这个字符串调整成能够让前端组件直接识别的形式。
- `senderName`：参数 `base64` 是 `base64` 字符串，函数会去除这个字符串之前的后面的多媒体对应内容，返回这条信息的发送人的名称。
- `tra`：参数 `item` 是同时含有发送人和多媒体信息的 `base64` 字符串，函数会判断这个字符串的类型，把它解析成对应的形式，最后渲染成能够直接在聊天界面显示的组件，然后return。
- `render`：没有参数，会调用上面实现的所有组件和其他文件中实现的组件，形成用户聊天界面。

- `socket_connection` 监听到信息后的具体操作：会通过识别 `conversation_type` 来判断消息的类型。有 `conversation_type` 就是成功响应，会根据具体的内容来判断是什么功能的成功响应；如果没有这一项就是错误响应，会直接调用 `utils.js` 中的函数，增加错误响应的可读性后通过 `antdesign` 的 `message` 弹窗告知用户。下面我们只解释成功响应的部分。
  - `conversation_type` 是 `public_init`：后端发送了群聊的历史记录，这里会对后端发来的信息进行解读和格式调整，然后存储到全局变量中。
  - `conversation_type` 是 `private_init`：后端发送了私聊的历史记录，这里会对后端发来的信息进行解读和格式调整，然后存储到全局变量中。
  - `conversation_type` 是 `chat`：收到了一条信息，这里会根据信息的类型做不同的格式处理，然后调用组件进行展示。
  - `conversation_type` 是 `log_out`：登出的成功响应，会弹窗提醒用户。
  - `conversation_type` 是 `private_withdraw`：撤回信息的成功响应，会把响应信息存储到全局变量中，并重新渲染前端页面，显示撤回信息之后的聊天界面。
  - `conversation_type` 是 `reply_message`：成功回复信息的成功响应，会把响应信息存储到全局变量中，包装这条信息，并重新渲染前端页面，显示发送出去这条信息之后的聊天界面。
  - `conversation_type` 是 `message forwarding`：成功转发信息的成功响应，会清零相关的全局变量。
  - `conversation_type` 是 `delete_message`：成功删除信息的成功响应，会修改相关的全局变量，并重新渲染前端页面，显示删除这条信息之后的聊天界面。
  - `conversation_type` 是 `mention_feedback`：你提及的对象看到了你的提及的成功响应，会弹窗提醒用户。
  - `conversation_type` 是 `public_create`：创建群聊的成功响应，会弹窗提醒当前用户，并根据是否所有人都成功入群（如果被拉取的对象把你删了，你就不能把他成功拉到群里）和你是群主还是群成员分别进行不同的前端包装渲染。
  - `conversation_type` 是 `view_init`：收到了所有聊天的信息，会存储到全局便来给你中并包装渲染显示出来，置顶顺序、是否免打扰、是否二次加密、未读信息等都是在这里接收。
  - `conversation_type` 是 `choose_group_manager`：成功把某个用户设定为群管理员的成功响应，会弹窗提醒用户。
  - `conversation_type` 是 `group owner transfer`：成功转让群主的成功响应，会弹窗提醒用户。

        - `conversation_type`是`group announcement`：成功发布群公告的成功响应，会根据这条群公告是否是你发布的分别进行包装渲染显示和弹窗提醒。
        - `conversation_type`是`group_info`：成功请求到这个群聊的基本信息的成功响应，会把这些信息包装渲染显示出来。
        - `conversation_type`是`invite`：成功给好友发送了入群要求的成功响应，会弹窗提醒用户。
        - `conversation_type`是`invite_to_check`：用户管理的群里有成员发送了拉别人入群的申请的成功响应，会弹窗提醒去查看。
        - `conversation_type`是`invite_init`：收到了自己管理的所有群的入群申请的成功响应，会把入群申请渲染出来，提供一个交互界面让用户决定是否通过。
        - `conversation_type`是`invite_approve`：成功通过了一条用户管理的群的入群申请的成功响应，会把信息存储到全局变量中并弹窗显示。
        - `conversation_type`是`invite_reject`：成功拒绝了一条用户管理的群的入群申请的成功响应，会弹窗显示。
        - `conversation_type`是`exit_group`：成功从群聊中退出的成功响应，会弹窗提醒。
        - `conversation_type`是`member_change`：所在的某个群中成员变化了的成功响应，会修改对应的全局变量。
        - `conversation_type`是`unread count`：后端提醒某个聊天收到了一条未读信息的成功响应，会渲染修改那个聊天显示的未读信息的数量。
        - `conversation_type`是`public_withdraw`：成功在群中撤回了一条信息的成功响应，会修改对应的全局变量，并渲染显示修改。
        - `conversation_type`是`multimedia message`：收到了一条多媒体信息的成功响应，会修改对应的全局变量，并调用相关的函数渲染显示。
        - `conversation_type`是`mention_user`：成功提及了某个用户的成功响应，会弹窗提醒当前用户。
        - `conversation_type`是`mentioned`：当前用户被提及了的成功响应，会弹窗提醒用户去查看。
        - `conversation_type`是`leave_chat`：用户成功离开聊天的成功响应，不需要进行相关操作。
        - `conversation_type`是`have_read`：某个用户已读某个聊天的信息的成功响应，会更新相关的全局变量，并渲染修改已读的提醒信息。
        - `conversation_type`是`add_conversation`：成功添加和另一个用户的私聊的成功响应，修改相关的全局变量并显示这个私聊。
        - `conversation_type`是`group_member_and_friend`：请求群的成员列表和自己的好友列表的成功响应，返回一个交互界面，可以对群进行操作（例如移除群成员或者拉自己的好友入群）。
    - `friend_request_connection`监听到信息后的具体操作：有`conversation_type`就是成功响应，会根据具体的内容来判断是什么功能的成功响应；如果没有这一项就是错误响应，会直接调用`utils.js`中的函数，增加错误响应的可读性后通过`antdesign`的`message`弹窗告知用户。下面我们只解释成功响应的部分。
        - `conversation_type`是`request_init`：从后端收到了所有未处理的好友申请的成功响应，会把这些信息存储到全局变量中。

        - `conversation_type`是`request_forward_accept`：你通过了对方发送的好友申请的成功响应，修改全局变量并渲染显示。
        - `conversation_type`是`request_forward_accepted`：你发送的好友申请被通过了的成功响应，修改全局变量并渲染显示。
        - `conversation_type`是`request_send`：你成功发出了好友申请的成功响应，会弹窗提醒用户。
        - `conversation_type`是`request_receive`：你收到了好友申请的成功响应，会弹窗提醒用户。
        - `conversation_type`是`request_forward_refuse`：你成功拒绝了对方发出的好友申请的成功响应，会弹窗提醒用户并存储到全局变量中。
        - `conversation_type`是`request_forward_refused`：对方拒绝了你发送的好友申请的成功响应，会弹窗提醒用户。

2.1.2.14 register.js

- 该文件对应着注册界面
- 状态变量：
    - `name`用户名
    - `password`用户密码
    - `email`邮箱账号
    - `emailpassword`邮箱验证码
- 函数：
    - `returnToLayOutPage`：返回基本功能界面的函数
    - `nameChange`：记录用户输入的账户名
    - `sendEmailPassword`：向后端请求发送邮箱验证码
    - `passwordChange`：记录一下用户输入的密码
    - `emailChange`：记录用户输入的邮箱
    - `emailPasswordChange`：记录用户输入的邮箱验证码
    - `register`：向后端请求验证注册

2.1.2.15 userinfo.js

- 该文件对应着查看以及编辑个人信息查看功能模块， 有`UserInfo`、`VerificationContainer`以及`ImageUpload`组件
    - `UserInfo`
        - 函数：
            - `handleInputChange`记录用户输入内容
            - `changeFormat`调整修改类型的格式
            - `handleSave`向后端请求改变个人信息
            - `requestAvatar`向后端请求头像信息
        - 状态变量：

        - `name`
        - `editobject//编辑内容`
        - `avatar//用户头像`
        - `editingField: '用户名'//编辑领域`
        - `edit_type//编辑类型`
        - `email//用户邮箱`
        - `emailpassword//邮箱验证码`
    - `VerificationContainer`
        - 函数:
            - `GetVerificationCode` 向后端请求得到邮箱验证码
            - `emailchange` 记录用户输入的邮箱
            - `emailPasswordChange` 记录输入的邮箱验证码
            - `handleVerification` 向后端请求进入编辑模式
            - `checkVerification_other` 请求编辑非头像内容
            - `checkVerification_avatar` 请求编辑头像信息
        - 状态变量:
            - `isverified//是否通过验证的标识符`
            - `name//用户名字`
            - `edit_type//编辑类型`
            - `email//用户邮箱`
            - `emailpassword//邮箱验证码`
    - `ImageUpload`
        - 函数:
            - `handleImageChange` 记录重新改变的头像
            - `base64ToBinary` 将base64字符串转换为二进制码
            - `requestAvatar` 请求得到头像
            - `sendChage` 向后端提交头像修改
        - 状态变量:
            - `file//上传的图片文件`
            - `name//用户名字`
            - `imagePreviewUrl//图片查看路径`
            - `imageData//图片数据`
            - `used_avatar//之前的头像数据`

2.1.2.16 test.js

- 对 `utils.js` 中处理无法使用邮箱登录的错误响应的函数进行了测试

2.1.2.17 setupTests.js

是框架自带的文件。

2.2 后端

后端主要分为以下两大部分：http主要接口、WebSocket通信实现。

2.2.1 http主要接口

在这一部分中，我们选用了传统的django框架，简单快捷地实现对MySQL数据库的查询。

- 使用框架：django
- 后端接口数量：20个
- 在查询数据库时，使用的是传统的 `ClassName.objects.filter()` 的方式。

2.2.2 WebSocket通信实现

为了实现长连接，我们选用了WebSocket。

- 使用框架：`Channels` 中的 `WebsocketConsumer` 类。我们基于这个类实现了 `ChatConsumer` 类和 `FriendRequestConsumer` 类。我们在这两个类中实现了 `__init__()`, `connect()`, `disconnect()`, `receive()`, `request_ws_failed()` 几个函数，其中主要在 `receive()` 函数中实现了绝大多数功能。
  - 在具体实现的时候，由于 `WebsocketConsumer` 对象不能存储在数据库中，我们采用了“随时创建，随时删除”的方式。具体说来，是创建了两个全局的列表，分别存放当前存在的所有 `ChatConsumer` 或 `FriendRequestConsumer` 对象。
  - 当前端实现连接时，调用 `as_asgi()` 方法，为每一个user-connection初始化一个 `Consumer` 对象
  - 在两种 `Consumer` 的 `connect()` 函数中，将 `self` 加入对应的全局列表中。在 `Consumer` 执行 `disconnect()` 时，将 `self` 从对应的全局列表中移除，并且显式地删除 `self`。
  - 这样，我们就实现了“随时创建，随时删除”。
- `Channels` 是一个基于Django框架的第三方插件，它提供了一种高效、可扩展的方式来处理异步请求和后台任务。它的优点包括：
  - 提供异步IO处理，可以更好地处理大量并发请求。Channels通过使用多进程或者异步处理模块，能够同时处理多个请求，执行效率非常高。
  - 不仅仅支持HTTP协议，还支持WebSockets、ASGI等多种协议，能够满足不同应用场景的需求。
  - 支持自定义的路由、协议和网络层，开发者可以根据需要进行扩展和定制。
  - 与Django的集成非常紧密，可以直接使用Django中的ORM和模板系统，使得开发者可以轻松地将已有的Django应用程序移植至Channels平台。

- `WebsocketConsumer`是`Channels`中处理WebSocket通信的核心部分，它可以非常方便地实现长连接通信，将WebSocket请求转换为Python函数，可以在这些函数中实现业务逻辑，并通过WebSocket将结果返回给客户端。它的优点包括：
    - 可以快速构建Websocket应用程序，无需过多关注底层细节，可以专注于业务逻辑的实现。
    - 可以实现全双工的通信，能够保持连接状态，提高通信的可靠性与稳定性

3. API设计

为了清晰起见并划分功能，我们将API文档组织成了四大主要模块，展示如下：

- 对于使用http协议通信的部分，小标题的格式是**URL/some_method**.
- 对于使用WebSocket通信的部分，小标题的格式是**一段中文介绍。**

3.1 用户操作模块

URL/request_verification_register

在注册时，前端向后端发出请求，要求后端给指定邮箱发验证码。

由于是在注册时的操作，这个邮箱并不应该对应一个已存在的用户。

请注意：

- 请求体、成功与错误响应与下一小节相似。
- 只不过前端需要在不同的情境下选择到底请求哪个URL。
- 只有在注册时，才请求此URL。其他时候（登录、编辑），请求下一小节对应的URL。

POST

使用 POST 方法请求该API，要求后端发送验证码到指定邮箱。

请求体

请求体的格式为：

```
{
    "email": "abc@163.com",
    "type": "request_verification"
}
```

- `email`字段需要是字符串且不能为空。
- `type`字段为常量值`"request_verification"`

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `email` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [email]"`
    - `email` 字段已经被其他人注册过，不能再将它作为你的注册邮箱：`info` 为 `"[email] has been registered by others, so it can't be used"`

URL/request_verification

前端向后端发出请求，要求后端给指定邮箱发验证码。

后端根据请求体确定已存在的某一用户。

POST

使用 POST 方法请求该API，要求后端发送验证码到指定邮箱。

请求体

请求体的格式为：

```
{
    "email": "abc@163.com",
    "type": "request_verification"
}
```

- `email` 字段需要是字符串且不能为空。
- `type` 字段为常量值 `"request_verification"`

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed"
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `email` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [email]"`
    - `email` 字段未被注册过，从而无法关联到某个特定用户：`info` 为 `"[email] has not been registered"`

URL/user

这一部分实现用户的注册和注销。

POST

使用 POST 方法请求该API，实现注册。

请求体

请求体的格式为：

```
{
    "name": "LiHua",
    "password": "wo_shi_Li_Hua111",
    "email": "abc@163.com",
    "type": "register",
    "verification": "547533"
}
```

- `name`：表示该用户的用户名。对应于后端同名变量。请注意，用户名要求只含有大小写字母、数字、下划线，不得为空，长度不得超过20位。
- `password`：表示该用户的密码。对应于后端同名变量。请注意，密码要求只含有大小写字母、数字、下划线，不得为空，并且长度不得短于8位。
- `email`：表示该用户的邮箱。对应于后端同名变量。请注意，邮箱要求为字符串，不能为空。任意两个用户的邮箱不能相同。
- `type`：表示该请求的类型。为 `register` 即为注册，为 `deregister` 即为注销。在POST里面，只能以 `register` 作为类型，其余要报错。
- `verification`：表示验证码。必须是6位数字。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "user_id": 14
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `name` 字段已被别的用户注册过：`info` 为 `"[name] has been registered"`
    - `password` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [password]"`
    - `password` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [password]"`
    - `password` 字段长度不合要求：`info` 为 `"Bad length of [password]"`
    - `email` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [email]"`
    - `email` 字段已被别的用户注册过：`info` 为 `"[email] has been registered"`
    - `type` 字段不是规定好的值：`info` 为 `"Invalid request [type]"`
    - `verification` 字段和后端存储的真实验证码不符：`info` 为 `"Incorrect verification code"`
- 若读写数据中途抛出错误，错误响应的状态码为 500 Internal Server Error，`code` 字段为 `-4`，`info` 为 `"An error occured when accessing data"`
- 若用户没请求过验证码就要求注册，则错误响应的 `code` 字段为 `1`，状态码为 403 Forbidden，`info` 为 `"you haven't asked for verification code"`

DELETE

使用 DELETE 方法请求该API，实现注销用户。

请求体

请求体的格式为：

```
{
    "name": "LiHua",
    "password": "wo_shi_Li_Hua111",
    "type": "deregister"
}
```

- `name`：表示该用户的用户名。对应于后端同名变量。请注意，用户名要求只含有大小写字母、数字、下划线，不得为空，长度不得超过20位。

- `password`：表示该用户的密码。对应于后端同名变量。请注意，密码要求只含有大小写字母、数字、下划线，不得为空，并且长度不得短于8位。
- `type`：表示该请求的类型。为`register`即为注册，为`deregister`即为注销。在DELETE里面，只能以`deregister`作为类型，其余要报错。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "user_id": 14
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
    - `name`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [name]"`
    - `name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [name]"`
    - `name`字段长度不合要求：`info`为`"Bad length of [name]"`
    - `password`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [password]"`
    - `password`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [password]"`
    - `password`字段长度不合要求：`info`为`"Bad length of [password]"`
    - `type`字段不是规定好的值：`info`为`"Invalid request [type]"`
- 若根据请求体中的用户名找不到用户，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`为`"User not found"`
- 若在请求体中的用户名密码无法匹配，即密码错误，错误响应的`code`字段为`1`，状态码为 403 Forbidden，`info`为`"Wrong password"`
- 若读写数据中途抛出错误，错误响应的`code`字段为`-4`，状态码为 500 Internal Server Error，`info`为`"An error occured when accessing data"`

URL/user_login/name_password

这一段实现用户的用户名-密码登录。

POST

使用 POST 方法请求该API.

请求体

请求体的格式为：

```
{
    "name": "LiHua",
    "password": "wo_shi_Li_Hua111"
}
```

- `name`和 `password`字段的要求和注册时一样。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "user_id": 14
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `name`字段字段缺失或非字符串类型：`info`为 `"Missing or error type of [name]"`
  - `name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为 `"Invalid char in [name]"`
  - `name`字段长度不合要求：`info`为 `"Bad length of [name]"`
  - `password`字段字段缺失或非字符串类型：`info`为 `"Missing or error type of [password]"`
  - `password`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为 `"Invalid char in [password]"`
  - `password`字段长度不合要求：`info`为 `"Bad length of [password]"`
- 若根据请求体中的用户名找不到用户，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 为 `"User not found"`

- 若在请求体中的用户名密码无法匹配，即密码错误，错误响应的 `code` 字段为 `1`，状态码为 403 Forbidden，`info` 为 `"Wrong password"`
- 若读写数据中途抛出错误，错误响应的 `code` 字段为 `-4`，状态码为 500 Internal Server Error，`info` 为 `"An error occured when accessing data"`

URL/user_login/email_verification

这一段实现用户的邮箱-验证码登录。

这一段必须在发送完验证码之后再请求。

POST

使用 POST 方法请求该API.

请求体

我们设定只有两种登录认证手段，即

1. 通过用户名、密码；
2. 通过邮箱、验证码。

请求体的格式为：

```
{
    "email": "abc@163.com",
    "verification": "765643"
}
```

- `email` 字段需要是字符串且不能为空。
- `verification`：表示验证码。必须是6位数字。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "name": "LiHua"
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。

    - `email`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [email]"`
    - `verification`字段和后端存储的真实验证码不符：`info`为`"Incorrect verification code"`
- 若根据请求体中的邮箱找不到用户，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`为`"User not found"`
- 若读写数据中途抛出错误，错误响应的`code`字段为`-4`，状态码为 500 Internal Server Error，`info`为`"An error occured when accessing data"`

URL/user_info_edit

这一段实现用户的个人信息修改与编辑，包括用户名、密码、邮箱。

请注意这一部分中不包含“头像修改”，因为它和另外三者有较大的差别。“头像修改”详见**URL/user_avatar_edit**.

POST

使用 POST 方法请求该API，请求修改“用户名、密码、邮箱”中任意一项。

请求体

请求体的格式为：

```
{
    "name": "LiHua",
    "info_type": "name",
    "new_value": "LiHuaHua"
}
```

- `name`为当前用户的名字。
- `info_type`为待修改的信息类型，只能取以下几个值：`"name", "password", "email"`。
- `new_value`为新的用户信息。对它的要求由`info_type`决定。比如如果是新用户名，则`new_value`值需要满足先前对于用户名的一切约定。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "user_id": 14
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段：
        - `name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
        - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
        - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
        - `name` 字段不是一个现存用户的名字：`info` 为 `"[name] has not been registered"`
    - 若 `info_type` 为 `name`
        - `new_value` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [new_name]"`
        - `new_value` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [new_name]"`
        - `new_value` 字段长度不合要求：`info` 为 `"Bad length of [new_name]"`
        - `new_value` 字段和现在名字一样：`info` 为 `"Your new name is the same as your old name, modification is invalid"`
        - `new_value` 字段已被别的用户注册过：`info` 为 `"[new_name] has been registered by others"`
    - 若 `info_type` 为 `password`
        - `new_value` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [new_password]"`
        - `new_value` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [new_password]"`
        - `new_value` 字段长度不合要求：`info` 为 `"Bad length of [new_password]"`
        - `new_value` 字段和现在密码一样：`info` 为 `"Your new password is the same as your old password, modification is invalid"`
    - 若 `info_type` 为 `email`
        - `new_value` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [new_email]"`
        - `new_value` 和现在邮箱一样：`info` 为 `"Your new email is the same as your old email, modification is invalid"`
        - `new_value` 字段已被别的用户注册过：`info` 为 `"[new_email] has been registered by others"`
    - 若 `info_type` 为其他值，`info` 为 `"Invalid type of [info_type]"`
- 若读写数据中途抛出错误，错误响应的 `code` 字段为 `-4`，状态码为 500 Internal Server Error，`info` 为 `"An error occured when accessing data"`

URL/user_request_avatar

前端需要渲染用户头像图片的时候，要向后端发送请求，要求后端把保存好的图片文件以base64位字符串的形式发送过来。这一部分实现上述功能。

POST

使用 POST 方法请求该API.

请求体

请求体的格式为：

```
{
    "name": "LiHua"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "avatar": "[Base64位字符串]"
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段：
        - `name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
        - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
        - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
- 若根据请求体中的 `name` 字段找不到用户，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 为 `"[name] has not been registered"`
- 若若读写数据中途抛出错误，错误响应的 `code` 字段为 `-4`，状态码为 500 Internal Server Error，`info` 字段的具体内容根据具体错误确定。
    - 若后端没能成功读取用户的 `avatar`，或者根据该 `avatar` 没能成功获取到base64字符串，则 `info` 为 `"failed to get avatar"`

URL/user_avatar_edit

POST

使用 POST 方法请求该API.

前端需要将前端需要将用户选择的图片转换成Base64位字符串，然后向后端发送这一格式的信息，后端保存这一信息数据于文本文件中。日后，如果前端需要渲染头像，就向后端请求Base64位字符串格式的数据，后端再从对应的文本文件中读取数据，并发送给前端。

请求体

请求体的格式为：

```
{
    "name": "LiHua",
    "new_avatar": "[Base64位字符串]"
}
```

- `name`即要修改头像的用户的名字。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "user_id": 14
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name`字段：
        - `name`字段缺失或非字符串类型：`info`为 `"Missing or error type of [name]"`
        - `name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为 `"Invalid char in [name]"`
        - `name`字段长度不合要求：`info`为 `"Bad length of [name]"`
    - 如果 `new_avatar`缺失或非字符串类型：`info`为 `"Missing or error type of [new_avatar]"`。

- 若根据请求体中的 `name` 字段找不到用户，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 为 `"[name] has not been registered"`

3.2 好友模块

URL/friend

这一段实现好友关系相关功能，包括展示好友列表，好友删除。

POST

使用 POST 方法请求该API，表示请求展现用户的好友列表。

请求体

请求体包含了发出查找自己好友的用户的用户名。请求体的格式为：

```
{
    "name": "c7w"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "friends_ids": friends_list,
    "friends_names": friends_names,
    "friends_tags": tag_list,
}
```

其中 `friend` 包含了所有好友的数据，每一个好友对应一个json，其中包含了 `id`（整型数字），`name`（字符串）和加好友的时间（python中是 `datetime.date` 类型）。
`tag_list` 里面是所有的标签，但不和 `friend_ids` 和 `friends_names` 一一对应，只是把所有已有的 `tag` 发送过来

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`

    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`

DELETE

使用 DELETE 方法请求该API即表示删除好友

请求体

请求体的格式为：

```
{
    "name": "LiHua",
    "deleted_name": "c7w"
}
```

发出者和被删除者的 `name`。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "view_order": view_order_list,
    "view_id_order": view_id_order_list,
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `deleted_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [deleted_name]"`
    - `deleted_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [deleted_name]"`
    - `deleted_name` 字段长度不合要求：`info` 为 `"Bad length of [deleted_name]"`

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据 `deleted_name` 查找的用户不存在：`info` 为 `"Friend to delete not found"`
    - `name` 对应的用户和 `deleted_name` 对应的用户之间没有好友关系：`info` 为 `"Receiver is not your friend"`

基于ws的好友申请相关操作

连接的首次发送

连接了聊天的WebSocket之后，前端就需要给后端发送一个请求体，以便后端获取用户信息。

请求体

请求体的格式为：

```
{
    "name": "Li"
}
```

`"name"` 字段是该用户的名字

成功响应

```
{
    "code": 0
}
```

错误响应

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`

发送申请

请求体

请求体的格式为：

```
{
    "request_type": "request_send",
    "sender_name": "LiHua",
    "receiver_name": "c7w",
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，发送者收到的成功响应格式为：

```
{
    "request_type": "request_send",
    "sender_name": "LiHua",
    "receiver_name": "c7w",
}
```

发送者收到的响应中 `request_type` 为 `"request_send"`

接收者收到的响应中 `request_type` 为 `"request_receive"`

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `sender_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [sender_name]"`
  - `sender_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [sender_name]"`
  - `sender_name` 字段长度不合要求：`info` 为 `"Bad length of [sender_name]"`
  - `receiver_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [receiver_name]"`
  - `receiver_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [receiver_name]"`
  - `receiver_name` 字段长度不合要求：`info` 为 `"Bad length of [receiver_name]"`
  - 已经存在 `name` 对应的用户和 `receiver_name` 对应的用户之间的好友申请，无论发送方是谁：`info` 为 `"Application has existed"`
  - `name` 和 `receiver_name` 之间已经存在好友关系：`info` 为 `"Can't send friend request to a friend"`
  - `name` 和 `receiver_name` 一样：`info` 为 `"Can't send friend request to yourself"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
  - 根据 `sender_name` 查找的用户不存在：`info` 为 `"Sender not found"`
  - 根据 `receiver_name` 查找的用户不存在：`info` 为 `"Receiver not found"`

接受、拒绝好友申请的初始渲染

根据请求体中的 `type` 区分初始渲染和在线处理请求。

这一段是初始渲染，渲染所有未处理的好友请求，即初始化。

请求体

请求体的格式为：

```
{
    "request_type": "request_init",
    "receiver_name": "c7w",
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "request_type": "request_init",
    "receiver_name": "c7w",
    "request_body": unprocessed_request
}
```

`unprocessed_request` 是一个 `list`，包含所有未处理的好友请求的发送方的名字。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `receiver_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [receiver_name]"`
    - `receiver_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [receiver_name]"`
    - `receiver_name` 字段长度不合要求：`info` 为 `"Bad length of [receiver_name]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `receiver_name` 查找的用户不存在：`info` 为 `"Receiver not found"`

在线接受好友申请

请求体

请求体的格式为：

```
{
    "request_type": "request_forward_accept",
    "sender_name": "LiHua",
    "receiver_name": "c7w",
}
```

forward意为转发，`type`字段为`"request_forward_accept"`意为接受好友请求。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "request_type": "request_forward_accept",
    "sender_name": "LiHua",
    "receiver_name": "c7w",
    "conversation_id": 5
}
```

接受好友申请之后，会在两个人之间新建一个对话，`conversation_id`即为该对话的`id`

申请发送者如果在线，也会收到响应，`request_type`字段为`"request_forward_accepted"`。接受者收到的响应则如上所示。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
    - `sender_name`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [sender_name]"`
    - `sender_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [sender_name]"`
    - `sender_name`字段长度不合要求：`info`为`"Bad length of [sender_name]"`
    - `receiver_name`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [receiver_name]"`
    - `receiver_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [receiver_name]"`
    - `receiver_name`字段长度不合要求：`info`为`"Bad length of [receiver_name]"`

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `sender_name` 查找的用户不存在：`info` 为 `"Sender not found"`
    - 根据 `receiver_name` 查找的用户不存在：`info` 为 `"Receiver not found"`
    - `sender` 和 `reciever` 代表的用户之间没有好友请求：`info` 为 `"Request not found"`

在线拒绝好友申请

请求体

请求体的格式为：

```
{
    "request_type": "request_forward_refuse",
    "sender_name": "LiHua",
    "receiver_name": "c7w",
}
```

forward意为转发，`type` 字段为 `"request_forward_refuse"` 意为拒绝好友请求。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "request_type": "request_forward_refuse",
    "sender_name": "LiHua",
    "receiver_name": "c7w",
}
```

申请发送者如果在线，也会收到响应，`request_type` 字段为 `"request_forward_refused"`。拒绝者收到的响应则如上所示。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `sender_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [sender_name]"`
    - `sender_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [sender_name]"`
    - `sender_name` 字段长度不合要求：`info` 为 `"Bad length of [sender_name]"`
    - `receiver_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [receiver_name]"`

    - `receiver_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [receiver_name]"`
    - `receiver_name`字段长度不合要求：`info`为`"Bad length of [receiver_name]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`字段的具体内容根据具体错误确定。
    - 根据`sender_name`查找的用户不存在：`info`为`"Sender not found"`
    - 根据`receiver_name`查找的用户不存在：`info`为`"Receiver not found"`
    - `sender`和`reciever`代表的用户之间没有好友请求：`info`为`"Request not found"`

URL/user_search

这一段实现用户查找功能。

POST

使用 POST 方法请求该API，表示根据关键词查找用户。

`key_word`为搜索姓名（出于隐私考虑，不应让用户被通过`id`搜索到）

请求体

请求体的格式为：

```
{
    "key_word": "c7w"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，响应体包含被搜索用户的基本信息，包括用户的`id`、用户名，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "user_name": "c7w",
    "user_id": 1,
    "user_email": "c7w@163.com",
    "user_avatar": base64位字符串
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。

    - `key_word` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [key_word]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段为 `"Can't find relative user"`。

URL/friend_tag

这一段实现好友分组，通过标签的方式进行分组。

POST

通过 POST 方法请求该API，表示给指定好友添加标签。

请求体

请求体的格式为：

```
{
    "name": "Li", // 用户自己的名字
    "friend_name": "c7w", // 好友的名字
    "tag": "TA", // 标签的具体内容
}
```

成功响应

请求成功时，应当设置状态码为 200 OK

```
{
    "code": 0,
    "info": "Succeed",
    "tag_list": tag_list, //存放当前所有标签的list
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `friend_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [friend_name]"`
    - `friend_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [friend_name]"`

    - `friend_name` 字段长度不合要求：`info` 为 `"Bad length of [friend_name]"`
    - `tag` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [tag]"`
    - 已经存在 `name` 对应的用户和 `friend_name` 对应的用户之间没有好友关系：`info` 为 `"You are not friends"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据 `friend_name` 查找的用户不存在：`info` 为 `"Friend may have deregistered"`

DELETE

通过 DELETE 方法请求该API，表示从某个组中移除指定好友

请求体

请求体的格式为：

```
{
    "name": "Li", // 用户自己的名字
    "friend_name": "c7w", // 好友的名字
    "tag": "TA", // 标签的具体内容
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "tag_list": tag_list, // 存放当前所有标签的list
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`

    - `friend_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [friend_name]"`
    - `friend_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [friend_name]"`
    - `friend_name` 字段长度不合要求：`info` 为 `"Bad length of [friend_name]"`
    - `tag` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [tag]"`
    - 已经存在 `name` 对应的用户和 `friend_name` 对应的用户之间没有好友关系：`info` 为 `"You are not friends"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据 `friend_name` 查找的用户不存在：`info` 为 `"Friend may have deregistered"`

URL/get_friend_tag

获得某个分组下的所有好友。

POST

通过 POST 方法请求该API，获得这个标签下的所有好友的用户名。

请求体

请求体的格式为：

```
{
    "name": "Li", // 用户自己的名字
    "tag": "TA", // 标签的具体内容
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "user_list": "一个列表，装着所有人的用户名",
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。

    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `tag` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [tag]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`

3.3 聊天基础功能模块

这一段实现用户聊天，绝大多数基于WebSocket因此不区分post、put等方法。这一部分根据请求体中的 `conversation_type` 区分场景以返回不同的响应。

连接后的首次发送

连接了聊天的WebSocket之后，前端就需要给后端发送一个请求体，以便后端获取用户信息。

请求体

请求体的格式为：

```
{
    "conversation_type": "get_name",
    "name": "Li"
}
```

- `name` 字段是当前用户的名字。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "get_name",
    "name": "Li"
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`

请求初始渲染

前端向后端发送这一请求，以渲染当前用户聊天界面各个聊天的顺序。

请求体

请求体的格式为：

```
{
    "conversation_type": "view_init",
    "name": "LiHua"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "view_init",
    "name": "LiHua",
    "view_order": view_order_list,
    "view_id_order" : view_id_order_list,
    "silent_list": silent_list,
    "unread_count_list": [1,0,8,9,0],
    "selected_conversation_list": [5],
    "selected_sender_list": ["c7w"],
    "tag_list": ["THU", "Family"],
    "twice_verification_list": [True, False],
    "num_of_top": 3
}
```

- `view_order` 是聊天名称的排序。
- `view_id_order` 是聊天 `id` 的排序。
- `silent_list` 表示的是 `view_order` 对应的各个聊天是否需要免打扰，每一项是 `True/False`。
- `unread_count_list` 为该用户每个聊天的未读计数的列表。
- `selected_conversation_list` 表明该用户在哪些群聊中被提及。
- `selected_sender_list` 表示谁提及了你，和上一个list是一一对应的。
- `tag_list` 表示该用户有哪些好友标签。
- `twice_verification_list` 表示 `view_order` 里面各个群聊是否需要进行二次验证。
- `num_of_top` 表示置顶的聊天数目。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `conversation_type` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [conversation_type]"`
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`

私聊&请求初始渲染

请求体

请求体的格式为：

```
{
    "conversation_type": "private_init",
    "conversation_id": 5,
    "name": "LiHua",
    "receiver_name": "c7w"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "private_init",
    "is_private_conversation": True,
    "name": "LiHua",
    "receiver_name": "c7w",
    "msg_list": message_list,
    "message_id_list": message_id_list,
    "message_type_list": message_type_list,
    "message_time_list": message_time_list,
    "message_avatar_rank_list": [0,1,1,0,1],
    "message_reply_list": [-1, 1, 1, -1, 4],
    "message_have_read_list":[["XiaoHong"], ["XiaoHong"], ["XiaoHong"],
["XiaoHong"], []],
    "message_num_of_reply_list": [2, 0, 1, 0, 0],
    "view_order": view_order_list,
    "view_id_order": view_id_order_list,
    "avatar_list": avatar_list
}
```

- `is_private_conversation` 是一个布尔变量，`True` 表示是私聊，`False` 表示是群聊。

- `msg_list`为历史聊天记录的列表。
- `message_id_list`为历史聊天记录所对应的`id`的列表
- `message_type_list`是消息类型的列表，指示前端要以什么方式渲染内容。其值为`0,1,2,3,4`分别意味着消息类型是文本、图片、音频、视频和文件。
- `message_time_list`是每条消息的发送时间列表。
- `message_avatar_rank_list`是一个01串，为0意味着这条消息是`avatar_list[0]`对应的用户发的，为1意味着这条消息是`avatar_list[1]`对应的用户发的。
- `message_reply_list`意思是：若`message_reply_list[i] = j`，则意味着第`i`条消息回复了`id`为`j`的消息。`j = -1`意味着第`i`条消息没回复任何一条消息。注意，这里的"第`i`条"消息指的是id为`message_id_list[i]`的消息。
- `message_have_read_list`的每一项，对应于`message_id_list`的那一项的已读者列表。
- `message_num_of_reply_list`意思是：若`message_num_of_reply_list[i] = j`，意味着id为`message_id_list[i]`的消息被回复了`j`次。
- `view_order`是聊天名称的排序。
- `view_id_order`是聊天`id`的排序。
- `avatar_list`是两个人的头像的列表。

如果聊天中有人提及了该用户，则用户点击进该聊天之后即向提及发送者发送响应(如果他在线的话)

```
{
    "conversation_type": "mention_feedback",
    "selected_user_name": "LiHua",
    "conversation_id": 5
}
```

- `selected_user_name`是提及你的人的名字。
- `conversation_id`表明哪个群聊中的提及得到了反馈。
- 请注意，如果私聊的另一个人已经注销，那么后端会在成功响应里面加一字段`"caution": "your receiver no longer exists"`

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
    - `name`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [name]"`
    - `name`字段不符合"只含有大小写字母、数字、下划线"的要求：`info`为`"Invalid char in [name]"`
    - `name`字段长度不合要求：`info`为`"Bad length of [name]"`

    - `conversation_id`字段缺失或非整数类型：`info`为`"Missing or error type of [conversation_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`字段的具体内容根据具体错误确定。
    - 根据`name`查找的用户不存在：`info`为`"User not found"`
    - 根据`conversation_id`查找的聊天不存在：`info`为`"Conversation not found"`
    - `name`对应的用户和同在该私聊中的另一位用户之间没有好友关系：`info`为`"Receiver is not your friend"`

聊天进行中

这一部分既处理私聊，也处理群聊。

对于该聊天中的所有人，假如这个人在线且在这个聊天中，则向他发送本节中的成功响应。如果他在线但是不在这个聊天中，则向他发送`unread count`响应。

请求体

请求体的格式为：

```
{
    "conversation_type": "chat",
    "conversation_id": 4,
    "name": "LiHua",
    "msg_body": "Hello, c7w!"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "chat",
    "conversation_id": 4,
    "sender_name": "LiHua",
    "msg_body": "Hello, c7w!",
    "message_id": 5,
    "message_time": “2023-05-15 21:45:54.199149”,
    "sender_avatar": "[base64字符串]",
    "have_read_list": ["LiHua", "c7w"]
}
```

- `msg_body`是信息的内容。
- `message_id`是这条信息对应的id。
- `message_time`是发出的时间。
- `sender_name`是消息的发出者。
- `sender_avatar`是发出者的头像。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `conversation_id` 字段字段缺失或非数字类型：`info` 为 `"Missing or error type of [conversation_id]"`
    - `msg_body` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [msg_body]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据 `conversation_id` 查找的聊天不存在：`info` 为 `"Conversation not found"`
    - 如果是私聊，并且双方没有好友关系，则 `info` 为 `"No friendship between users in a private chat"`
    - 如果是群聊，并且当前用户已经被踢出群聊，则 `info` 为 `"You have been removed from the group chat"`
    - 如果是私聊，对方已经注销，则 `info` 为 `"You can't send message because your receiver no longer exists"`

退出登录

这一段实现用户的登出。

请求体

请求体的格式为：

```
{
    "conversation_type": "log_out",
    "user_name": "c7w",
    "view_order": view_order_list,
    "view_id_order": view_id_order_list,
    "silent_list": silent_list,
    "need_twice_verification": need_twice_verification_list,
    "num_of_top": num_of_top
}
```

- 用户登出的时候，前端把各个聊天的排列顺序发送给后端；用户登录的时候，后端把聊天顺序再发送给前端。
- `user_name`是用户的名字。
- `view_order`是一个列表，按顺序存放了不同聊天的名字。
- `view_id_order`按顺序存放了不同聊天的`id`

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type" : "log_out",
    "code": 0,
    "info": "Succeed"
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
    - 若`view_order`字段缺失或不是list：`info`为`"Missing or error type of [view_order]"`
    - 若`view_id_order`字段缺失或不是list：`info`为`"Missing or error type of [view_id_order]"`
    - 若`silent_list`字段缺失或不是list：`info`为`"Missing or error type of [silent_list]"`
    - 若`num_of_top`字段缺失或不是整数：`info`为`"Missing or error type of [num_of_top]"`
    - `user_name`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [user_name]"`
    - `user_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [user_name]"`
    - `user_name`字段长度不合要求：`info`为`"Bad length of [user_name]"`

私聊&撤回消息

对于私聊而言，只有三分钟以内的信息可以撤回。不能撤回别人发的消息。

请求体

请求体的格式为：

```
{
    "conversation_type": "private_withdraw",
    "name": "LiHua",
    "conversation_id": 3,
    "msg_body": "Hello, c7w!",
    "message_id": 5
}
```

- `conversation_type` 为 `"private_withdraw"` 意味着意味着该次请求为私聊聊天进行中，撤回消息。
- `conversation_id` 为这个聊天的id。
- `msg_body` 是那条待撤回的消息。
- `message_id` 是那条待撤回的消息对应在数据库中的id。

成功响应

请求成功时，后端直接删掉这条消息。应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "conversation_type": "private_withdraw",
    "conversation_id": 3,
    "msg_body": "Hello, c7w!",
    "message_id": 5,
    "info": "Succeed"
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `msg_body` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [msg_body]"`
    - `message_id` 字段字段缺失或非整数类型：`info` 为 `"Missing or error type of [message_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `message_id` 和 `msg_body` 查找的信息不存在：`info` 为 `"Message not found"`

- 若该消息已经超过了三分钟的期限，无法撤回，则错误响应的`code`字段为`2`，状态码为403 Forbidden，`info`字段的具体内容为`"Fail to withdraw message due to time limit"`。
- 若该消息不是本人发的，无法撤回，则错误响应的`code`字段为`2`，状态码为403 Forbidden，`info`字段的具体内容为`"Fail to withdraw message due to limits of authority"`。

回复消息

用户回复某一条消息。

对于该聊天中的所有人，假如这个人在线且在这个聊天中，则向他发送本节中的成功响应。如果他在线但是不在这个聊天中，则向他发送`unread count`响应。

请求体

请求体的格式为：

```
{
    "conversation_type": "reply_message",
    "current_user_name": "LiHua",
    "msg_body": "Hello everyone, I am c7w!",
    "reply_msg_body": "Hello c7w, nice to meet you.",
    "message_id": 10,
    "conversation_id": 9
}
```

- `current_user_name`：当前前端的这个用户的名字。
- `msg_body`：要被回复的这条消息的内容。
- `reply_msg_body`：回复的内容。
- `message_id`：要被回复的这条消息的id。
- `conversation_id`：当前所在的聊天的id。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "conversation_type": "reply_message",
    "message_responder": "LiHua",
    "msg_body": "Hello everyone, I am c7w!",
    "reply_msg_body": "Hello c7w, nice to meet you.",
    "message_id": 13,
    "message_time": “2023-05-15 21:45:54.199149”,
    "num_of_reply": 1,
    "message_responder_avatar": "[base64字符串]",
    "have_read_list": ['LiHua', 'c7w', 'gg']
}
```

- `message_responder`是回复消息的人的名字。
- `num_of_reply`是要被回复的那条消息已经被多少条消息所回复。
- `message_responder_avatar`是回复消息的人的头像。

- `have_read_list` 是哪些人已读了新发的这条消息。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `current_user_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [current_user_name]"`
    - `current_user_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [current_user_name]"`
    - `current_user_name` 字段长度不合要求：`info` 为 `"Bad length of [current_user_name]"`
    - `msg_body` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [msg_body]"`
    - `reply_msg_body` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [reply_msg_body]"`
    - `conversation_id` 字段字段缺失或非整数类型：`info` 为 `"Missing or error type of [conversation_id]"`
    - `message_id` 字段字段缺失或非整数类型：`info` 为 `"Missing or error type of [message_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - `message_id` 对应的那条信息不存在于数据库中：`info` 为 `"Message not found"`
    - `conversation_id` 对应的那条信息不存在于数据库中：`info` 为 `"Conversation not found"`
    - 如果当前聊天是群聊，并且当前用户已经被移出：`info` 为 `"You have been removed from the group chat"`

删除信息

用户告诉后端自己删除了哪条信息，后端需要记住，然后在下次这个用户登录申请历史信息的时候，不把删除的信息发给他。

请求体

请求体的格式为：

```
{
    "conversation_type": "delete_message",
    "current_user_name": "LiHua",
    "message_id": 10
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "delete_message",
    "current_user_name": "LiHua",
    "message_id": 10,
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `message_id` 字段字段缺失或非整数类型：`info` 为 `"Missing or error type of [message_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
  - `current_user_name` 对应的用户不存在于数据库中：`info` 为 `"User not found"`
  - `message_id` 对应的那条信息不存在于数据库中：`info` 为 `"Message not found"`

URL/translation

允许用户将外语文本消息翻译为汉语。

POST

请求体

请求体的格式为：

```
{
    "msg_body": "my name is LiHua and i love eating"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "transalated_msg_body": "我叫李华，我喜欢吃"
}
```

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]",
}
```

- 若服务器调用接口失败，没能完成翻译，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段为 `"Failed to translate due to unexpected error"`.一般情况下，这是不会出现的情况。

消息转发

允许用户在弹出拓展列表后选择多选消息，用户多选该会话中部分消息后，允许将用户选定的消息合并转发至其他会话中。

例如将会话2中的几条消息合并转发到会话2或4，流程如下：

1. 前端选定会话2中部分消息——前端给后端发送这些消息的 `id` 组成的列表，以及当前的用户名字——后端根据 `id`，定位数据库中的这些消息及其发出者——后端将这些消息按照时间排序——后端将这些信息包装好——存储这条包装好的信息；
2. 接下来，如果合并信息是要转发到当前会话，即会话2的，则：效仿“聊天进行中”的格式，使用相同的 `conversation_type` 给当前会话发回来；
3. 如果合并信息是要转发到另一个会话，如会话4的，则如本节成功响应所示。这一响应只发送给当前的会话。
4. 但是，对于会话4，则：效仿“聊天进行中”的格式，使用相同的 `conversation_type` 给会话4发回来。

请求体

请求体的格式为：

```
{
    "conversation_type": "message forwarding",
    "message_id_list": [1, 3, 4, 9],
    "sender_name": "LiHua",
    "current_conversation_id": 2,
    "target_conversation_id": 4
}
```

- `message_id_list`：被选定的信息的 `id` 的列表。

- `sender_name`：执行转发操作的用户名。
- `current_conversation_id`：当前用户正在使用的聊天的`id`。
- `target_conversation_id`：想要转发到的聊天的`id`。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "conversation_type": "message forwarding",
    "target_conversation_id": 4,
    "sender_name": "LiHua"
}
```

- `message_list`指的是`message_id_list`对应的信息文本所组成的列表。
  - 举例：`msg_body`：`"LiHua:  ['LiHua: hi!', 'c7w: hello.', 'LiHua: how are u?', 'c7w: i'm fine.']"`
- `sender_name`为`LiHua`指示的是：这次转发是当前聊天里的`LiHua`这个人干的。

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
  - `sender_name`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [sender_name]"`
  - `sender_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [sender_name]"`
  - `sender_name`字段长度不合要求：`info`为`"Bad length of [sender_name]"`
  - `current_conversation_id`字段字段缺失或非整数类型：`info`为`"Missing or error type of [current_conversation_id]"`
  - `target_conversation_id`字段字段缺失或非整数类型：`info`为`"Missing or error type of [target_conversation_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`字段的具体内容根据具体错误确定。
  - 根据`current_conversation_id`找不到对应的聊天对象：`info`为`"Current_conversation not found"`
  - 根据`target_conversation_id`找不到对应的聊天对象：`info`为`"Target_conversation not found"`

- 如果当前聊天是群聊，并且当前用户已经被移出：`info`为`"You have been removed from the group chat"`
- 如果当前聊天是私聊，你的朋友已经注销，你将被禁止向当前聊天转发消息：`info`为`"You can't forward messages to the current chat"`

未读计数（用户在线时）

本方法不需要请求体，是后端向前端单向的输送。

当一条消息被保存，后端就要发送一次响应。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "conversation_type": "unread count",
    "conversation_id": 5,
    "conversation_name": "A与C的聊天"
}
```

未读计数（用户不在线时）

仅仅让未读计数+1.

未读计数_获取未读的数字

详见`view_init`处。

在`view_init`时，把未读计数列表发给前端。在`private_init`或者`public_init`时，置未读计数为0.

多媒体消息

统一使用Base64字符串的形式在前后端之间传递信息。

具体流程大致如下：

1. A向B发一个文件
2. A的前端将文件转为字符串，包装在请求体里发给后端
3. 后端接收到字符串，将其转为文本文件存储起来，将文本文件的名字作为`Message`模型的`msg_body`，把响应发给A和C
4. C的前端接收字符串，然后把字符串再转换回文件。

请求体

请求体的格式为：

```
{
    "conversation_type": "multimedia message",
    "multimedia_info": "[base64位字符串]",
    "multimedia_type": "pdf",
    "sender_name": "LiHua",
    "conversation_id": 9
}
```

- `multimedia_type` 只可以是 `jpg`，`wav`，`mp4`，`pdf` 这四种，即图片、语音、视频和文件，且规定后缀必须是这四个。
- 这四个后缀分别对应编号1, 2, 3, 4.
- 后端不会对 `multimedia_type` 的具体值做检查，只会检查它是否为空。然后会把它直接发还给前端。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "conversation_type": "multimedia message",
    "multimedia_info": "[base64位字符串]",
    "multimedia_type": "pdf",
    "sender_name": "LiHua",
    "conversation_id": 9,
    "message_id": 17,
    "message_time": "2023-05-15 21:45:54.199149",
    "sender_avatar": "[base64位字符串]",
    "have_read_list": ["LiHua", "c7w", "iuwehdui"]
}
```

- `multimedia_info` 为多媒体字符串格式的信息。
- 由于多媒体消息的本质也是消息，所以 `message_id` 指的就是这条多媒体消息对应的id。
- `sender_avatar` 为发送者的头像。
- `have_read_list` 是这个聊天中已读这条消息的人的列表。

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `multimedia_info` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [multimedia_info]"`
    - `multimedia_type` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [multimedia_type]"`
    - `sender_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [sender_name]"`
    - `sender_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [sender_name]"`
    - `sender_name` 字段长度不合要求：`info` 为 `"Bad length of [sender_name]"`

    - `conversation_id`字段字段缺失或非整数类型：`info`为`"Missing or error type of [conversation_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`字段的具体内容根据具体错误确定。
    - 根据`conversation_id`找不到对应的聊天对象：`info`为`"Conversation not found"`
    - 根据`sender_name`找不到对应的用户：`info`为`"User not found"`
    - 如果当前聊天是群聊，并且当前用户已经被移出：`info`为`"You have been removed from the group chat"`
    - 如果是私聊，对方已经注销，则`info`为`"You can't send message because your receiver no longer exists"`

URL/voice_to_text

这一段实现语音转文字，基于http实现。

尽管这和多媒体消息有关，但是为了分离处理，在用户使用这一功能时，前端应将音频的Base64字符串形式再次给后端发一遍。后端调用百度云API实现转换，并将转换后的字符串发还前端。

POST

使用 POST 方法请求该API.

请求体

请求体的格式为：

```
{
    "voice_info": "[base64位字符串]",
    "voice_type": "wav"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed",
    "text_info": "你好，我叫李华"
}
```

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。

    - `voice_info`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [voice_info]"`
- 若服务器调用接口失败，没能完成转换，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`字段为`"Failed to convert due to unexpected error"`. 不过，一般情况下这不会出现。

已读消息

不需要请求体。

每当一个用户初始化时，给“所有正在群聊里的前端“发响应。

每当一个用户聊天/回复信息/转发时，“所有正在群聊里的前端“将会给它发响应。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "have_read",
    "conversation_id": 3,
    "have_read_name": "c7w"
}
```

URL/chat_history

这一段实现聊天消息的搜索，基于http实现。

POST

使用 POST 方法请求该API.

请求体

请求体的格式为：

```
{
    "check_type": "time",
    "check_body": "2023/1/1",
    "conversation_id": 5
}
```

- `check_type`代表了根据什么信息筛选聊天内容。合法内容有：`time、message_type、sender_name`。
- `check_body`代表了具体用于搜索的信息。
- `time`对应的`check_body`的合法格式有：`xxxx.xx.xx、xxxx/xx/xx`和`xxxx_xx_xx、xxxx-xx-xx`，即支持年月日搜索。
- `message_type`对应的`check_body`的合法内容有：`text、image、wav、mp4、pdf`
- `sender_name`对应的`check_body`格式要求与对姓名的要求相同。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "check_type": "time",
    "message_list": message_list
}
```

- `message_list` 是符合筛选条件的所有消息组成的列表。

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `check_type` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [check_type]"`
  - `check_type` 字段不是合法内容：`info` 为 `"Illegal content of [check_type]"`
  - `check_body` 字段内容不合法：`info` 为 `"Illegal content of [check_body]"`
  - `conversation_id` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [conversation_id]"`
- 若根据请求体中的 `conversation_id` 找不到群聊，错误响应的 `code` 字段为 `1`，`info` 为 `"Conversation not found"`
- 若 `check_type` 是 `sender_name`，按照 `check_body` 找不到对应的用户：`info` 为 `"User not found"`

URL/set_twice_verification

对指定的聊天设置二次验证。

POST

使用POST请求该API，表示将聊天设置成需要二次验证的聊天

请求体

请求体的格式为：

```
{
    "conversation_id": 5,
    "name": "LiHua",
    "password": "1234567"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "code": 0,
    "info": "Succeed"
}
```

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `password` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [password]"`
    - `password` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [password]"`
    - `password` 字段长度不合要求：`info` 为 `"Bad length of [password]"`
    - `conversation_id` 字段字段缺失或非整数类型：`info` 为 `"Missing or error type of [conversation_id]"`
- 若根据请求体中的用户名找不到用户，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 为 `"User not found"`
- 若在请求体中的用户名密码无法匹配，即密码错误，错误响应的 `code` 字段为 `1`，状态码为 403 Forbidden，`info` 为 `"Wrong password"`
- 如果该聊天已经是需要二次验证的聊天，则错误响应的 `code` 字段为 `1`，状态码为 403 Forbidden，`info` 为 `"Already set"`

DELETE

使用 DELETE 方法请求该API，表示将该聊天设置成不需要二次验证的聊天。

- 如果该聊天本就不需要二次验证，则状态码设为 403 Forbidden，`code` 为 `1`，`info` 为 `"Already set"`。
- 其余各种请求与响应格式与 POST 方法相同。

URL/twice_verification

二次验证，检查用户名密码是否正确。

这一部分内容同**URL/set_twice_verification**.

消息免打扰

采用和`view_order`同样的逻辑，登出时前端向后端发送是否免打扰，登入时后端将该信息发送回去，见**退出登录**和**请求初始渲染。**

添加聊天

右键点击好友，添加私聊聊天。

请求体

请求体的格式为：

```
{
    "conversation_type": "add_conversation",
    "my_name": "c7w",
    "friend_name": "用户名",
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "add_conversation",
    "conversation_id": 5,
    "conversation_name": "A and B", // 私聊的名字
}
```

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
    - `conversation_type`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [conversation_type]"`
    - `my_name`字段字段缺失或非字符串类型：`info`为`"Missing or error type of [my_name]"`
    - `my_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [my_name]"`
    - `my_name`字段长度不合要求：`info`为`"Bad length of [my_name]"`

    - `friend_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [friend_name]"`
    - `friend_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [friend_name]"`
    - `friend_name` 字段长度不合要求：`info` 为 `"Bad length of [friend_name]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `my_name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据 `friend_name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据两个 `name` 找不到对应的私聊：`info` 为 `"Conversation not found"`

特定群聊的当前成员列表和当前用户的好友列表

为了实现特定的功能，例如在某个用户改名之后，他的朋友和群友能立刻知晓，需要实现这样的API.

请求体

请求体的格式为：

```
{
    "conversation_type": "group_member_and_friend",
    "conversation_id": 3,
    "name": "c7w"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "group_member_and_friend",
    "conversation_id": 3,
    "name": "c7w",
    "group_member_name_list": ["LIHUA", "hhh", "c7w"],
    "friend_name_list": ["LIHUA", "GGBb"]
}
```

- `group_member_name_list` 是当前群成员名字列表，只含有仍在群中的人，包括自己。
- `friend_name_list` 是当前用户好友列表，只含有未注销的好友，不包括自己。

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。

    - `conversation_id` 字段字段缺失或非整数类型：`info` 为 `"Missing or error type of [conversation_id]"`
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"Conversation not found"`

3.4 群聊功能模块

基于ws的群聊

这一段实现用户聊天，基于websocket因此不区分post、put等方法。这一部分根据请求体中的 `"conversation_type"` 区分场景以返回不同的响应。

创建群聊

创建群聊可以看做类似于不需要经过同意的好友申请（注意和成员邀请不同），采用websocket进行创建。
创建的方式分为列表创建和私聊创建，两种创建的方式的请求体和响应是相同的。

请求体

请求体的格式为：

```
{
    "conversation_type": "public_create",
    "name": "LiHua",  // 群主
    "group_name": "kj,lx,zy",
    "group_member": name_list,  // 群主创建群聊邀请的群成员的名字的list
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "public_create",
    "conversation_id": 5,
    "name": "LiHua",  // 群主
    "group_name": "seteamp",
    "accept_list" : accept_list,
    "reject_list": reject_list // 拒绝入群的人们
}
```

- `accept_list` 里面存储了成功入群的群成员的名字。
- `reject_list` 里面存储了被邀请但由于某些原因没成功邀请的用户的姓名。可能的原因包括：对方注销了账号、对方删除了和你的好友关系。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定或建立群聊失败，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `group_member` 字段内容确实或非list类型：`info` 为 `"Missing or error type of [group_member]"`
    - `group_member` 中用户数量不大于1：`info` 为 `"You should invite more than 2 people excluding you to create a group."`
    - 前端发来 `group_member` 有不少于2个人，但是所有被邀请者都没能进群：`info` 为 `"Maybe users invited all can't join in the group."`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`

群聊&请求初始渲染

请求体

请求体的格式为：

```
{
    "conversation_type": "private_init", // 前端并不知道一个聊天是否为群聊，因此该请求头和私聊一样
    "conversation_id": 5, // 群聊的id
    "name": "c7w",  // 群聊成员(发送请求者)的名字
    "receiver_name": "T,TT,YYY"
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "public_init",
    "is_private_conversation": False,
    "name": "c7w",  // 群聊成员(发送请求者)的名字
    "owner_name": "LiHua", // 群主的姓名
    "msg_list": msg_list,
    "message_id_list": message_id_list,
```

```
    "message_type_list": message_type_list,
    "message_time_list": message_time_list,
    "message_avatar_rank_list": [0,1,1,0,2,3,1],
    "message_reply_list": [-1, 1, 1, -1, 4, 2, 2],
    "message_have_read_list": [["LiHua","AAA"],["LiHua","AAA"],["LiHua","AAA"],
["LiHua","AAA"],["LiHua"], [], []],
    "message_num_of_reply_list": [2, 2, 1, 0, 0, 0, 0],
    "member_list": member_list,
    "member_avatar_list": member_avatar_list,
}
```

- `is_private_conversation` 是一个 `bool` 变量，`False` 表示是群聊
- `msg_list` 包含了群聊的历史消息。
- `member_list` 是群聊成员列表，只含有当前成员
- `member_avatar_list` 是成员头像列表，含有所有曾经在此群内的成员（有可能已经离开）
- `message_avatar_rank_list` 是一个数组，其中的数字都在 `0 ~ len(member_avatar_list)-1` 之间。`message_avatar_rank_list[i]` 为 `j` 意味着这条（第 `i` 条）`message` 是由 `member_avatar_list[j]` 对应的用户发的。
- `message_reply_list` 意思是：若 `message_reply_list[i] = j`，则意味着第 `i` 条消息回复了 `id` 为 `j` 的消息。`j = -1` 意味着第 `i` 条消息没回复任何一条消息。注意，这里的“第 `i` 条”消息指的是 `id` 为 `message_id_list[i]` 的消息。
- `message_num_of_reply_list` 意思是：若 `message_num_of_reply_list[i] = j`，意味着id为 `message_id_list[i]` 的消息被回复了 `j` 次。
- `message_have_read_list` 的每一项，对应于 `message_id_list` 的那一项的已读者列表。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
  - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
  - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
  - `conversation_id` 字段缺失或非整数类型：`info` 为 `"Missing or error type of [conversation_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
  - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
  - 根据 `conversation_id` 对应的群聊不存在：`info` 为 `"Conversation not found"`

- 如果当前用户已经被移出这个群聊：`info`为`"you have been removed from the group chat"`

群聊&指定群管理员

后端需要把所有管理员存起来。

请求体

请求体的格式为：

```
{
    "conversation_type": "choose_group_manager",
    "conversation_id": 5,
    "name": "LiHua",
    "manager_name_list": ["c7w", "xiaoming", "xiaohong"]
}
```

- `conversation_id`为当前群聊的`id`
- `name`为当前前端这个用户，也必须是群主。
- `manager_name_list`是群主想指定的管理员们的名字，可以是空的。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "choose_group_manager",
    "conversation_id": 5,
    "info": "Succeed",
    "manager_name_list": ["c7w", "xiaoming", "xiaohong"],
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的`code`字段为`1`，状态码为 404 Not Found，`info`字段的具体内容根据具体错误确定。
  - 根据`conversation_id`查找的群聊不存在：`info`为`"group chat not found"`
  - 根据`name`查找的用户不存在：`info`为`"User not found"`
  - 根据`name`查找的用户不是群主：`info`为`"User is not owner"`
  - 根据`manager_name_list`查找的某个用户不存在：`info`为`"User corresponding to manager_name not found"`

    - 根据`manager_name_list`查找的某个用户不在当前群里：`info`为`"User corresponding to manager_name not in the group"`
    - 根据`manager_name_list`查找的某个用户已经是管理员：`info`为`"User selected is already a manager`
    - 根据`manager_name_list`查找的某个用户是群主，不能当管理员：`info`为`"User selected is owner, thus can't be a manager"`
- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
    - `name`字段缺失或非字符串类型：`info`为`"Missing or error type of [name]"`
    - `name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [name]"`
    - `name`字段长度不合要求：`info`为`"Bad length of [name]"`
    - `conversation_id`字段缺失或非数字类型：`info`为`"Missing or error type of [conversation_id]"`

群聊&群主转让

请求体

请求体的格式为：

```
{
    "conversation_type": "group owner transfer",
    "conversation_id": 5,
    "name": "LiHua",
    "new_owner_name": "xiaohong"
}
```

- `new_owner_name`是新的群主。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "group owner transfer",
    "conversation_id": 5,
    "info": "Succeed",
    "new_owner_name": "xiaohong",
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
  - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`
  - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
  - 根据 `new_owner_name` 查找的用户不存在：`info` 为 `"new owner not found"`
  - 根据 `new_owner_name` 查找的用户不在当前群里：`info` 为 `"new owner not in the group"`
  - 若当前群主就是新群主，即群主把群主转给自己，这是无意义的，`info` 为 `"you can't transfer ownership to yourself"`
- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
  - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
  - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
  - `conversation_id` 字段缺失或非数字类型：`info` 为 `"Missing or error type of [conversation_id]"`
  - `new_owner_name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [new_owner_name]"`
  - `new_owner_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [new_owner_name]"`
  - `new_owner_name` 字段长度不合要求：`info` 为 `"Bad length of [new_owner_name]"`
- 若用户没有特定的权限，错误响应的 `code` 字段为 `-1`，`info` 字段的具体内容根据具体错误确定。
  - `current_user_name` 对应的用户不是群主，则不能转让群主，`info` 为 `"you don't have the access to transfer ownership since you are not the owner"`

群聊&群公告

只有群主和群管理员有资格发布群公告。不过，任何一名用户都可以去试图发布群公告（只不过会被拒绝）。

请求体

请求体的格式为：

```
{
    "conversation_type": "group announcement",
    "conversation_id": 5,
    "name": "LiHua",
    "announcement_content": "Everyone please come to school for meeting tomorrow."
}
```

- `name` 是当前这个前端的名字。
- `announcement_content` 是想要发布的群公告的内容。

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "group announcement",
    "announcer_name": "LiHua",
    "announcement_content": "群公告：Everyone please come to school for meeting
tomorrow.",
    "conversation_id": 5,
    "message_id": 45
}
```

- 响应将发送给群聊中的所有用户。
- `announcer_name` 为公告发布者的名字。
- `message_id` 为这条公告对应的id。
- `announcement_content` 多加了一个头"群公告："。

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
  - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`
  - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
  - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
  - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
  - `conversation_id` 字段缺失或非数字类型：`info` 为 `"Missing or error type of [conversation_id]"`
  - 若用户没有特定的权限，错误响应的 `code` 字段为 `-1`，`info` 字段的具体内容根据具体错误确定。
  - `name` 对应的用户既不是群主又不是管理员，则不能发表公告，`info` 为 `"you don't have the access to post announcement"`

群聊信息展示

对应功能文档中的信息展示。

请求体

请求体的格式为：

```
{
    "conversation_type": "group_info",
    "conversation_id": 5,
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "group_info",
    "conversation_id": 5,
    "conversation_name": "group", // 群聊名称
    "owner_name": "c7w", // 群主的名字
    "group_member": group_member_list, //一个存放了所有群成员姓名的列表，包括群主
    "announce_history": announce_list, //一个存放了所有历史群公告的列表
    "group_manager": group_manager_list, //一个存放了所有群管理员姓名的列表
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `conversation_id` 字段缺失或非数字类型：`info` 为 `"Missing or error type of [conversation_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`

成员邀请

群成员可以邀请好友加入群聊，经过群管理员或群主审核通过后加入群聊。

请求体

请求体的格式为：

```
{
    "conversation_type": "invite",
    "conversation_id": 5,
    "invitor_name": "LiHua", // 发出邀请者的名字
    "invited_name": "c7w", // 被邀请者的名字
}
```

成功响应

请求成功时，应当设置状态码为 200 OK。

给发出邀请者:

```
{
    "conversation_type": "invite",
    "conversation_id": 5,
    "invitor_name": "LiHua", // 发出邀请者的名字
    "invited_name": "c7w", // 被邀请者的名字
}
```

给群管理员和群主:

```
{
    "conversation_type": "invite_to_check",
    "conversation_id": 5,
    "invitor_name": "LiHua",
    "invited_name": "c7w",
}
```

类似好友申请，如果用户登录了，前端应该发送请求初始化自己是群主/群管理员的群收到的入群申请。

错误响应

只发给邀请发出者。所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`
    - 根据 `invitor_name` 或 `invited_name` 查找的用户不存在：`info` 为 `"User not found"`
    - 邀请方和被邀请方之间没有好友关系：`info` 为 `Invited user is not your friend`
    - 如果当前聊天是群聊，并且当前用户已经被移出：`info` 为 `"you have been removed from the group chat"`
- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `invitor_name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [invitor_name]"`

- `invitor_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [invitor_name]"`
- `invitor_name`字段长度不合要求：`info`为`"Bad length of [invitor_name]"`
- `invited_name`字段缺失或非字符串类型：`info`为`"Missing or error type of [invited_name]"`
- `invited_name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [invited_name]"`
- `invited_name`字段长度不合要求：`info`为`"Bad length of [invited_name]"`
- `conversation_id`字段缺失或非数字类型：`info`为`"Missing or error type of [conversation_id]"`
- `conversation_type`字段缺失或非字符串类型：`info`为`"Missing or error type of [conversation_type]"`
- 已经存在`invitor`向`invited`发送的入群邀请：`info`为`"Invitation exists"`
- 被邀请者已经在该群聊里了：`info`为`"Invited user has been in the group"`

群主或群管理员处理入群申请

上面提到，如果群主或者群管理员在线，就会收到响应，但是处理入群申请是在一个单独的地方处理（就比如一个按钮，上面写的入群申请）。
点击那个按钮之后，后端会把这个用户有权处理的申请发送过来，然后该用户对这些申请进行操作，类似好友申请。

请求体

请求体的格式为：

```
{
    "conversation_type": "invite_init",
    "name": "c7w",
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "invite_init",
    "conversation_id_list": [], // 存放需要该用户处理的入群申请对应的群聊的id
    "conversation_name_list": [], // 存放需要该用户处理的入群申请对应的群聊的名称
    "invited_name_list":[], // 申请入群者(也即被邀请者)的姓名
    "invitor_name_list":[],  // 邀请者的姓名
}
```

注意，这几个列表中的元素是按照下标一一对应的，比如群聊1有3个入群申请，是a、b、c要入群，`conversation_id_list`就会是`[1, 1, 1]`，`invited_name_list`对应是`["a", "b", "c"]`

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`

群主或群管理员同意/拒绝了入群申请

请求体

同意了入群申请:

```
{
    "conversation_type": "invite_approve",
    "conversation_id": 5,
    "invitor_name": "LiHua", // 发出邀请者的名字
    "invited_name": "c7w", // 被邀请者的名字
}
```

拒绝:

```
{
    "conversation_type": "invite_reject",
    "conversation_id": 5,
    "invitor_name": "LiHua", // 发出邀请者的名字
    "invited_name": "c7w", // 被邀请者的名字
}
```

成功响应

请求成功时，应当设置状态码为 200 OK。给同意/拒绝的操作者发送响应。

同意:

```
{
    "conversation_type": "invite_approve",
    "conversation_id": 5,
    "code": 1,
    "invitor_name": "LiHua", // 发出邀请者的名字
    "invited_name": "c7w", // 被邀请者的名字
    "member_now": member_list
}
```

同意之后群成员变化，给所有在线的群成员发送响应

```
{
    "conversation_type": "member_change",
    "conversation_id": 5,
    "member_now": member_list // 当前群成员有哪些
}
```

拒绝:

```
{
    "conversation_type": "invite_reject",
    "conversation_id": 5,
    "code": 1,
    "invitor_name": "LiHua", // 发出邀请者的名字
    "invited_name": "c7w", // 被邀请者的名字
}
```

错误响应

只发给审批申请的人。所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
  - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`
  - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
  - 不存在由 `invitor_name` 对应用户向 `invited_name` 对应用户发送的入群邀请：`info` 为 `"Application not found"`
  - 当前操作者没有操作权限：`info` 为 `"You are not owner or manager"`
- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
  - `invitor_name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [invitor_name]"`
  - `invitor_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [invitor_name]"`
  - `invitor_name` 字段长度不合要求：`info` 为 `"Bad length of [invitor_name]"`
  - `invited_name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [invited_name]"`
  - `invited_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [invited_name]"`
  - `invited_name` 字段长度不合要求：`info` 为 `"Bad length of [invited_name]"`
  - `conversation_id` 字段缺失或非数字类型：`info` 为 `"Missing or error type of [conversation_id]"`

- `conversation_type` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [conversation_type]"`

成员退出

群成员可以自由地退出群聊

请求体

请求体的格式为：

```
{
    "conversation_type": "exit_group",
    "conversation_id": 5,
    "name": "LiHua",
}
```

成功响应

请求成功时，应当设置状态码为 200 OK.

发送给操作者:

```
{
    "conversation_type": "exit_group",
    "name": "LiHua",
    "conversation_id": 5,
    "member_now": member_list
}
```

发送给在线的群成员:

```
{
    "conversation_type": "member_change",
    "conversation_id": 5,
    "member_now": member_list // 当前群成员有哪些
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
  - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`
  - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
  - `name` 对应的用户不在 `conversation_id` 对应的群聊中：`info` 为 `Not in the group`

    - 如果当前用户已经被移出这个群聊：`info`为`"you have been removed from the group chat"`
- 若请求体中数据格式不满足格式规定，错误响应的`code`字段为`-2`，状态码为 400 Bad Request，`info`字段的具体内容根据具体错误确定。
    - `name`字段缺失或非字符串类型：`info`为`"Missing or error type of [name]"`
    - `name`字段不符合“只含有大小写字母、数字、下划线”的要求：`info`为`"Invalid char in [name]"`
    - `name`字段长度不合要求：`info`为`"Bad length of [name]"`
    - `conversation_id`字段缺失或非数字类型：`info`为`"Missing or error type of [conversation_id]"`
    - `conversation_type`字段缺失或非字符串类型：`info`为`"Missing or error type of [conversation_type]"`

注：如果退出群聊的成员是群主，则优先从管理员中随机选一位为新群主，若无管理员则从群成员中选一位为新群主。

群聊&撤销消息

群主与群管理员可以无视撤回消息的时间约束直接撤回消息，群主可以撤回任何消息，群管理员仅能撤回非群主、非群管理员的群成员的消息。

为了方便处理，这部分中还要包括普通成员在群聊中撤销消息。他们只能撤回三分钟以内的消息。

请求体

请求体的格式为：

```
{
    "conversation_type": "public_withdraw",
    "name": "LiHua",
    "conversation_id": 5,
    "message_id": 32
}
```

成功响应

请求成功时，应当设置状态码为 200 OK，成功响应格式为：

```
{
    "conversation_type": "public_withdraw",
    "name": "LiHua",
    "conversation_id": 5,
    "message_id": 32
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`
    - 根据 `name` 查找的用户不存在：`info` 为 `"User not found"`
    - 根据 `message_id` 查找的消息不存在：`info` 为 `"message not found"`
- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [name]"`
    - `name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [name]"`
    - `name` 字段长度不合要求：`info` 为 `"Bad length of [name]"`
    - `conversation_id` 字段缺失或非数字类型：`info` 为 `"Missing or error type of [conversation_id]"`
    - `message_id` 字段缺失或非数字类型：`info` 为 `"Missing or error type of [message_id]"`
- 若用户没有特定的权限，错误响应的 `code` 字段为 `-1`，`info` 字段的具体内容根据具体错误确定。
    - 如果当前聊天是群聊，并且当前用户已经被移出：`info` 为 `"you have been removed from the group chat"`
    - `name` 对应的用户既不是群主又不是管理员，却想要撤回自己在3分钟之前发的消息，导致失败，则 `info` 为 `"fail to withdraw message due to time limit"`
    - `name` 对应的用户既不是群主又不是管理员，却想撤回其他人发的消息，导致失败，则 `info` 为 `"fail to withdraw message due to your identity as normal member"`
    - `name` 对应的用户是管理员，但是不是群主，却想撤回群主或者管理员发的消息，导致失败，则 `info` 为 `"fail to withdraw message due to your identity as manager"`

移除群成员

群管理员可以移除非群主和非管理员的成员；群主可以移除任意成员；不能移除自己。

请求体

请求体的格式为：

```
{
    "conversation_type": "member_remove",
    "conversation_id": 5,
    "operator_name": "c7w", // 发出移除成员动作的用户
    "remove_name": "Li", // 要被移除的用户
}
```

成功响应

请求成功时，应当设置状态码为 200 OK.

发送给操作发出者：

```
{
    "conversation_type": "member_remove",
    "conversation_id": 5,
    "operator_name": "c7w", // 发出移除成员动作的用户
    "remove_name": "Li", // 要被移除的用户
    "member_now": member_list // 当前群成员有哪些
}
```

发送给在线的群成员：

```
{
    "conversation_type": "member_change",
    "conversation_id": 5,
    "member_now": member_list // 当前群成员有哪些
}
```

错误响应

所有错误响应的格式均为：

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `conversation_id` 查找的群聊不存在：`info` 为 `"group chat not found"`
    - 根据 `operator_name` 或 `remove_name` 查找的用户不存在：`info` 为 `"User not found"`
    - 如果操作者已经被移出这个群聊：`info` 为 `"you have been removed from the group chat"`
- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `operator_name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [operator_name]"`
    - `operator_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [operator_name]"`
    - `operator_name` 字段长度不合要求：`info` 为 `"Bad length of [operator_name]"`
    - `remove_name` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [remove_name]"`
    - `remove_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [remove_name]"`

    - `remove_name`字段长度不合要求：`info`为`"Bad length of [remove_name]"`
    - `conversation_id`字段缺失或非数字类型：`info`为`"Missing or error type of [conversation_id]"`
    - `conversation_type`字段缺失或非字符串类型：`info`为`"Missing or error type of [conversation_type]"`
- 若用户没有特定的权限，错误响应的`code`字段为`-1`，`info`字段的具体内容根据具体错误确定。
    - `operator_name`和`remove_name`字段相同，即自己移除自己，则`info`为`"You may want to exit the group"`
    - `operator_name`对应的用户既不是群主又不是管理员(如果他要移除自己报上面的错)，`info`为`"You are not owner or manager"`
    - `operator_name`对应的用户是管理员，但`remove_name`是群主或群管理员，导致失败，`info`为`"Target is owner or manager"`

提及

用户选中成员列表中的某人后，消息内容中的提及应当以特殊格式显示（如蓝色字体）并且允许单击跳转到该成员基本信息页面或鼠标悬浮显示该成员基本信息。

前端向后端发送被选中的用户的名字，后端返回这个用户的信息：名字、邮箱、头像。头像以base64字符串的形式传输。

请求体

当一个用户发送的消息里包含@群成员的信息时，发送请求。请求体的格式为：

```
{
    "conversation_type": "mention_user",
    "select_user_name": "Li", // 谁发出了@
    "selected_user_name_list": selected_list, // 谁被@了
    "conversation_id" : 5,
}
```

成功响应

请求成功时，应当设置状态码为 200 OK。

发送给提及发出者的响应：

```
{
    "code": 0,
    "conversation_id": 5,
    "conversation_type": "mention_user",
    "selected_user_name_list": ["c7w"],
    "selected_user_id_list": [2]
}
```

发送给被提及且在线的人的响应：

```
{
    "conversation_id": 5,
    "conversation_type": "mentioned",
    "select_user_name": "c7w",
}
```

错误响应

所有错误响应的格式均为:

```
{
    "code": *,
    "info": "[Some message]"
}
```

- 若请求体中数据格式不满足格式规定，错误响应的 `code` 字段为 `-2`，状态码为 400 Bad Request，`info` 字段的具体内容根据具体错误确定。
    - `selected_user_name_list` 字段缺失或非字符串类型：`info` 为 `"Missing or error type of [selected_user_name_list]"`
    - `select_user_name` 字段字段缺失或非字符串类型：`info` 为 `"Missing or error type of [select_user_name]"`
    - `select_user_name` 字段不符合“只含有大小写字母、数字、下划线”的要求：`info` 为 `"Invalid char in [select_user_name]"`
    - `select_user_name` 字段长度不合要求：`info` 为 `"Bad length of [select_user_name]"`
    - `covnersation_id` 字段缺失或非整数类型：`info` 为 `"Missing or error type of [covnersation_id]"`
- 若服务器无法根据请求体找到对应的资源，错误响应的 `code` 字段为 `1`，状态码为 404 Not Found，`info` 字段的具体内容根据具体错误确定。
    - 根据 `select_user_name` 不能找到对应的用户：`info` 为 `"User not found"`
    - 根据 `conversation_id` 不能找到对应的聊天：`info` 为 `"Conversation not found"`

提及反馈

在 `private_init` 里面进行反馈。

请求体

无，在 `private_init` 的时候就进行反馈。

成功响应

见 `private_init`

错误响应

见 `private_init`

4. 数据库设计

4.1 表结构展示

现将本项目的所有表结构，即模型存储架构展示如下。

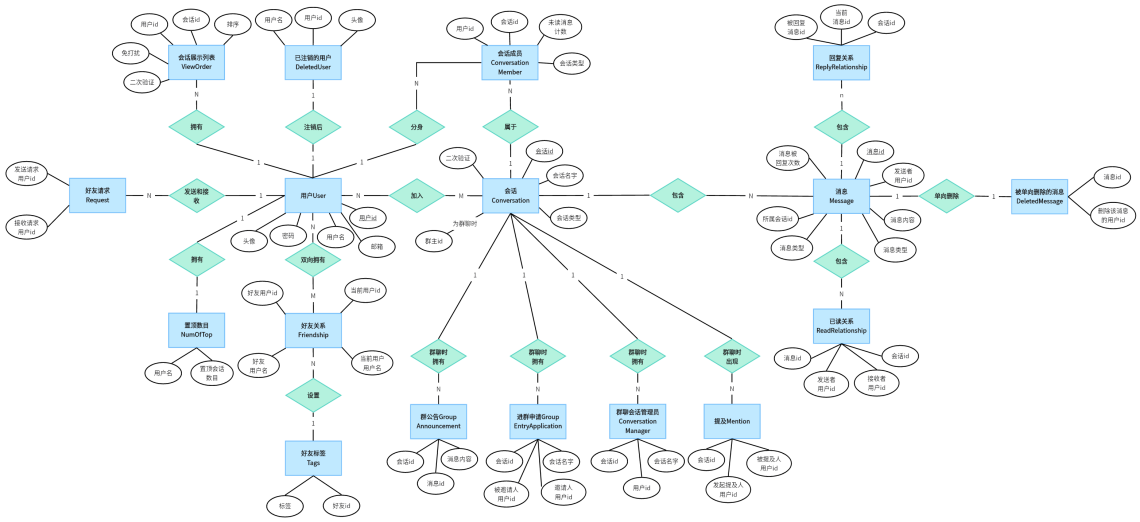
- 用户`User`
  - `user_id`：用户id，全局唯一的用户标识符，整型数据
  - `name`：用户名，字符串数据
  - `avatar`：用户头像文件的路径，字符串数据
  - `password`：用户的密码，加密字符串数据
  - `email`：用户的邮箱，加密邮箱字符串数据
  - `register_time`：用户注册时间，时间戳
  - `login_time`：用户登录时间，时间戳
  - `verification_code`：用户的验证码，仅在请求验证码之后、登录/注册/验证之前保存，其余时候置为空字符串。字符串数据
- 已注销的用户`DeletedUser`
  - `user_id`：就是用户注销前的id，整型数据
  - `name`：用户名，字符串数据
  - `avatar`：用户头像文件的路径，字符串数据
- 好友请求`FriendRequest`
  - `user_id`：发出请求的用户的id，整型数据
  - `update_time`：请求更新的时间，时间戳
  - `receiver_id`：接收请求的用户的id，整型数据
- 好友关系`Friendship`
  - `user_id`：当前用户的id，整型数据
  - `update_time`：好友关系更新的时间，时间戳
  - `friend_user_id`：当前用户的这一好友的id，整型数据
  - `friend_tag`：当前用户给这一好友加的标签，字符串数据
- 好友标签`Tags`
  - `tag`：这一标签的文本内容，字符串数据
  - `user_id`：被加上此标签的这一好友的id，整型数据
- 消息`Message`
  - `msg_id`：消息的id，全局唯一的消息标识符，整型数据
  - `msg_body`：消息的内容，字符串数据
  - `create_time`：消息的发送时间，时间戳
  - `sender_id`：消息发送者的用户id，整型数据
  - `conversation_id`：消息所属的会话的id，整型数据
  - `num_of_reply`：消息被回复的次数，整型数据

    - `multimedia_type`：消息的多媒体类型。为0则为文本，1则为图片（jpg），2则为音频（wav），3则为视频（mp4），4则为文件（pdf）
- 被单方面删除的消息`DeletedMessage`
    - `msg_id`：消息的id，整型数据
    - `user_id`：要求在其视角中将这条消息删除的用户的id，整型数据
- 会话`Conversation`
    - `conversation_id`：这一会话的id，全局唯一的会话标识符，整型数据
    - `conversation_name`：这一会话的名字，字符串数据
    - `create_time`：这一会话的创建时间，时间戳
    - `update_time`：这一会话的更新时间，时间戳
    - `is_private_conversation`：这一会话是否为私聊，为1则为私聊，为0则为群聊。布尔数据
    - `need_twice_verification`：这一会话是否需要二次验证，为1则需要，为0则不需要。布尔数据
    - `owner_user_id`：若这一会话为群聊，则这个变量指示群主的id，整型数据
- 会话成员`ConversationMember`
    - `conversation_id`：这一会话成员所处的会话的id，整型数据
    - `member_user_id`：这一会话成员作为用户的id，整型数据
    - `join_time`：这一会话成员首次加入会话的时间，时间戳
    - `update_time`：这一会话成员加入会话的时间，时间戳
    - `unread_count`：这一会话成员未读的消息数目，整型数据
    - `is_private_conversation`：这一会话成员所处的会话是否为私聊，为1则为私聊，为0则为群聊。布尔数据
- 群聊会话管理员`ConversationManager`
    - `conversation_id`：这一群聊会话的id，整型数据
    - `conversation_name`：这一群聊会话的名字，字符串数据
    - `user_id`：管理员作为用户的id，整型数据
- 会话展示列表`ViewOrder`
    - `viewer_id`：查看会话的用户的id，整型数据
    - `viewed_conversation_id`：被查看会话的id，整型数据
    - `rank`：该会话在聊天列表中的排序，整型数据
    - `is_silent`：该会话是否被设置为免打扰，布尔型数据
    - `need_twice_verification`：该会话是否被设置为需要二次验证，布尔型数据
- 进群申请`GroupEntryApplication`
    - `conversation_id`：这一群聊的id，整型数据
    - `conversation_name`：这一群聊的名字，字符串数据
    - `invited_id`：被邀请人的用户id，整型数据
    - `invited_name`：被邀请人的用户名，整型数据
    - `invitor_id`：邀请人的用户id，整型数据

    - `invitor_name`：邀请人的用户名，整型数据
- 群公告`GroupAnnouncement`
    - `conversation_id`：这一群公告所属的群聊的id，整型数据
    - `msg_id`：这一群公告作为消息的id，整型数据
    - `msg_body`：这一群公告的内容，字符串数据
- 提及`Mention`
    - `conversation_id`：这一提及所属的会话的id，整型数据
    - `select_user_id`：发起提及的用户的id，整型数据
    - `select_user_name`：发起提及的用户的名字，字符串数据
    - `selected_user_id`：被提及的用户的id，整型数据
    - `selected_user_name`：被提及的用户的名字，字符串数据
- 回复关系`ReplyRelationship`
    - `msg_id`：被回复的消息的id，整型数据
    - `reply_msg_id`：回复上一者的消息的id，整型数据
    - `conversation_id`：回复关系所属的聊天的id，整型数据
- 已读关系`ReadRelationship`
    - `msg_id`：消息的id，整型数据
    - `sender_id`：发消息者的id，整型数据
    - `receiver_id`：收消息者的id，整型数据
    - `conversation_id`：已读关系所属的聊天的id，整型数据
    - `have_read`：收消息者是否已读该消息，是则为1，否则为0。布尔数据
- 置顶数目`NumOfTop`
    - `name`：用户的名字
    - `num_of_top`：该用户置顶聊天的数目

4.2 表间关系展示

上述罗列的表间关系用E-R图表示如下：


会话展示列表 ViewOrder
用户id
会话id
排序
免打扰
二次验证
拥有
已注销的用户 DeletedUser
用户名
用户id
头像
注销后
会话成员 Conversation Member
用户id
会话id
未读消息计数
会话类型
分身
属于
回复关系 ReplyRelationship
被回复消息id
当前消息id
会话id
包含
好友请求 Request
发送请求用户id
接收请求用户id
发送和接收
用户User
头像
密码
用户名
昵称
简介id
邮箱
加入
二次验证
为群聊时
群主id
会话 Conversation
会话id
会话名字
会话类型
包含
消息 Message
消息id
发送者用户id
消息内容
消息类型
消息版回复次数
所属会话id
单向删除
被单向删除的消息 DeletedMessage
消息id
删除该消息的用户id
拥有
置顶数目 NumOfTop
用户名
置顶会话数目
双向拥有
好友关系 Friendship
好友用户id
当前用户id
好友用户名
当前用户用户名
设置
好友标签 Tags
标签
好友id
包含
已读关系 ReadRelationship
消息id
会话id
发送者用户id
接收者用户id
群聊时拥有
群公告Group Announcement
会话id
消息内容
消息id
群聊时拥有
进群申请Group EntryApplication
会话id
会话名字
被邀请人用户id
邀请人用户id
群聊时拥有
群聊会话管理员 Conversation Manager
会话id
会话名字
用户id
群聊时出现
提及Mention
会话id
被提及人用户id
发起提及人用户id