设计报告

作品名称：　多功能聊天软件系统

完成日期：　2024 年 5 月 24 日

目录

1. 引言

TCP/IP Socket 网络编程在计算机软件开发的多个领域都有重要应用，例如在网络通信应用开发领域，TCP/IP Socket 网络编程可以用来实现客户端和服务器之间的通信，使得不同设备之间能够进行数据交换和通信。在网络游戏开发领域，TCP/IP Socket 网络编程可以用来实现游戏客户端和服务器之间的通信。因此本课程的目标是设计一个多功能聊天软件系统。

1.1 动机

近些年来随着人工智能的不断发展，越来越多的人工智能衍生出来的应用开始进入人们的生活[1]，例如一直爆火的大语言模型 ChatGPT。基于 TCP/IP Socket 网络编程实现多人聊天室和爬虫都是一些比较传统的东西了。我想在此基础上加入点比较前沿的 AI 技术。使得该软件系统除了能实现单聊功能、群聊功能和爬虫功能外，还能实现和虚拟 AI 聊天的功能[2]。增加我在 AI 方面的相关知识。

所以，本系统旨在开发一个多功能聊天软件系统。

1.2 要解决的问题

为实现多功能聊天软件系统，本系统主要有以下几个功能：

（1）群聊功能的实现：服务端需要时刻接收连接进来的用户和用户发送的信息，然后将某个用户信息广播给所有用户。客户端需要接收广播数据，这样大家都能收到。

（2）私聊功能的实现：客户端选择要私聊的对象并发送消息后，服务器端只需要对指定客户端发送数据。

（3）爬虫功能的实现：解析浏览器的网页数据的规律后就可以爬取并从中提炼出有价值的信息，然后将数据保存下来并进行数据分析。

（4）聊天 AI 的实现：使用大语言模型的 API，编写一个 python 函数调用该 API 接口实现请求和响应。然后自己训练声音合成的模型，使 AI 的回答以语音输出。

（5）数据库的实现：使用免费且操作简单的 SQLite 数据库，并使用 Navicat 对数据库进行操作。

（6）GUI 界面的实现：在 Qt Designer 拖动组件设计好界面后使用 pyuic5 命令将.ui 文件转换为.py 文件，修改并调用。

2. 系统设计

2.1 系统总体框架

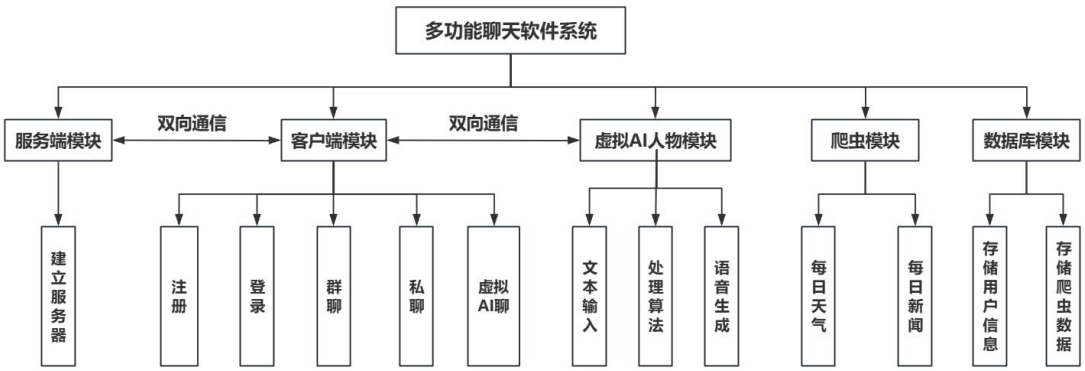


图 2.1 系统总体框架图

基于 1.2 中要解决的问题，我设计了如图 2.1 所示的系统。在系统总体框架中，从整体上可以分为服器端模块、客户端模块、虚拟 AI 人物模块、爬虫模块和数据库模块。服务端只需要建立服务器即可，来充当各个客户端的中转站。在客户端模块中，用户可以进行登录、注册、群聊、私聊以及虚拟 AI 聊天，AI 的回答可以以某个游戏角色的音色语音输出。虚拟 AI 人物模块需要对用户输入的文本进行处理，并将回答以特殊语音输出。服务端、客户端和虚拟 AI 人物之间都是双向通信的。爬虫模块可以爬取某个天气网站和新闻网站的数据，并以简洁明了的形式展示，方便用户查看。数据库模块负责存储用户的账号、密码和爬虫爬取的数据等信息。

2.2 系统总体流程

系统总体流程如图 2.2 所示。运行系统后，服务端会先启动，服务端启动完成后用户可以登录，登录时系统会连接数据库来进行用户账号与密码验证。登录成功后系统会开始爬取某个网站的每日天气和每日新闻数据并以简洁明了的样式展示给用户。用户可选择是群聊、私聊或和 AI 聊天，选择群聊或私聊后系统会将用户输入的信息发给服务端处理，服务端会首先对用户发来的数据进行判断是否是私聊，如果是私聊服务端会将信息发给指定用户，否则，服务端会对所有用户进行广播，将信息发给全体用户。如果用户选择和 AI 聊天（也就是和虚拟偶像聊天），用户输入消息后，后端会对用户输入的文本进行处理，调用大语言模型的 API 发送请求，在收到响应后会得到大语言模型的回答文本，然后调用之

前训练好的模型，将文本转换为某个二次元角色的 AI 语音，最后输出。服务端和客户端会一直进行双向通信，直至程序终止。

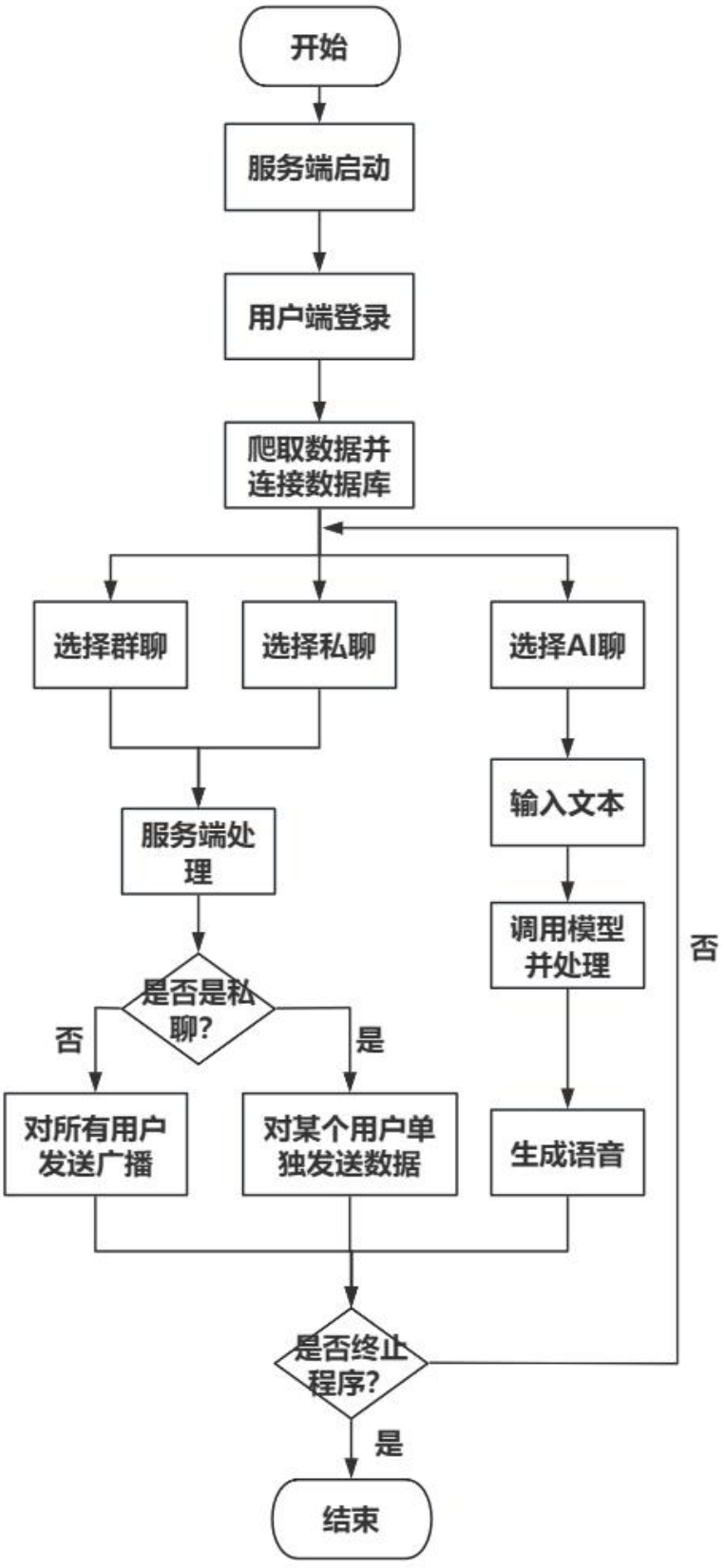


图 2.2 系统总体流程图

2.3 系统功能模块

本系统主要分为五大模块，分别是服务器端模块、客户端模块、虚拟 AI 人物模块、爬虫模块和数据库模块。每个模块功能的详细介绍如下：

1.服务端模块

该模块主要负责建立服务器。服务端相当于中转站，客户端相当于各个站点。服务端通过绑定到特定的网络地址和端口，并监听来自客户端的连接请求，实现了对网络流量的管理和调度。一旦建立连接，服务端负责接受客户端发送的数据，并根据预定的协议和逻辑进行处理。服务端还负责生成响应，并将其发送回客户端，完成一次通信交互。在多客户端情况下，服务端能够同时处理多个连接请求，确保系统的稳定性。

2.客户端模块

该模块可以发起连接，它通过指定目标服务器的地址和端口，向服务端发送连接请求，并在连接建立后，负责向服务端发送数据请求或命令，等待服务端的响应，并处理接收到的数据，从而实现与服务端的通信和数据交换。

在客户端用户使用前需要进行账号的注册和登录，登录成功后可以看到每日天气和每日新闻。在本系统的功能页中可以选择群聊，私聊和虚拟 AI 聊。在群聊中用户发送的信息会被所有用户接收到，而私聊中用户发送的信息只能被对方收到。在选择和虚拟 AI 聊天时，用户发出的问题会被大语言模型回答，并以某个二次元角色的语音回答。在客户端使用的时候，服务端和客户端会实时进行双通信。

3.虚拟 AI 人物模块

该模块负责对用户输入的文本进行处理和分析，并根据预设的大语言模型的逻辑和知识库生成相应的回答。这些回答可以以特殊语音的形式输出，模拟原神中的一些游戏角色的声音，与用户进行沟通交流，提供个性化的服务体验。

4.爬虫模块

该通过网络爬虫技术自动地访问指定的中国天气网和澎湃新闻网，从网页中提取相关信息，如天气预报、实时气温、新闻标题、摘要和链接等。随后，对这些信息进行解析和处理，筛选出用户感兴趣的内容，并以清晰简洁的形式呈现给用户。这样，用户无需手动浏览网页，便能方便快捷地获取所需的天气和新闻信息，提高用户体验和信息获取效率。

5.数据库模块

该模块的负责管理和存储系统中的各种数据，如用户账号信息、密码等。它负责创建、更新和查询数据库，确保数据的安全性和一致性。通过数据库模块，系统能够有效地管理用户信息，实现用户的登录、注册功能，并将数据永久化存储，以备后续使用和查询。

3. 数据结构设计

3.1 JSON

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，易于人们阅读和编写。它基于 JavaScript 语言的子集，但也适用于多种编程语言。JSON 使用键值对的形式来组织数据，数据由逗号分隔，键值对由冒号分隔，最外层使用花括号{}括起来表示对象，使用方括号[]括起来表示数组。JSON 广泛应用于网络数据传输、配置文件、API 接口等场景，由于其简洁、易读、易解析的特点，成为了数据交换的标准格式之一。

因此对于服务端和客户端的双向通信的数据交换，我打算使用 json 格式来处理，要发送数据前使用 json.dumps(some)打包后发送，接收到数据后再使用 json.loads(some)解包读取信息。

我使用的 json 数据结构如表 3.1 所示：

表 3.1 json 数据结构

json 数据结构
1: data = {
2: "ip": ,#客户端的 ip 地址
3: "port": ,#客户端的端口号
4: "mode": ,#用户选择的模式（群聊，单聊，刷新）
5: "sender": ,#信息发送者
6: "receiver": ,# 信息接受者
7: "format": ,#消息类型（text）
8: "online": [],#在线用户列表
9: "message": ,#用户发送的消息
10: }

3.2 列表

Python 中的列表是一种可变的、有序的数据结构，可以随时添加和删除其中的元素。列表非常适合利用顺序和位置定位某一元素，尤其是当元素的顺序或内容经常发生改变时。与字符串不同，列表是可变的。可以直接对原始列表进行修改：添加新元素、删除或覆盖已有元素创建列表。

在使用爬虫爬取澎湃新闻网时，网页各个标题，每个标题对应一个 URL，进

入那个URL后才是新闻的具体内容,所以需要将抓取的标题的URL放入URL列表，因此爬虫需要使用列表这个数据结构，每次将要爬取的的URL放入列表中，然后同时爬取列表中的所有URL。爬虫列表的使用方式如表3.2所示：

表 3.2 爬虫列表使用方式

爬虫列表
1: def begin_craw(self, url): 2: # 获取接下来要遍历的url列表 3: hrefList = self.get_title(url) 4: threads=[]#多线程对象列表 5: for title, href in hrefList.items(): 6: thread = threading.Thread(target=self.get_text, args=(href, title, folder_path)) 7: threads.append(thread) 8: for thread in threads: 9: thread.start()#开启多线程，同时爬取所有url 10: for thread in threads: 11: thread.join()

3.3 字典

Python中的字典是一种可变的无序集合，用于存储键值对。字典中的键必须是唯一的，而值可以是任何数据类型。可以使用大括号{}来创建字典，使用冒号:来分隔键值对。

在爬取完网页数据后要对数据进行分析，所以要知道每个上海地区对应的天气网址，因此使用字典比较方便。爬虫字典的使用方式如表3.3所示：

表 3.3 爬虫字典使用方式

爬虫字典
1: urls=[fhttp://www.weather.com.cn/weather/10102{str(idx).zfill(2)}00.shtml for idx in range(1, 18)] # 上海城市url 2: elf.citys = {} # 城市字典 3: def parse_today_weather(self, html, url): 4: parse_html() 5: city = jd['od']['od1'] 6: self.citys[url] = city # 记录城市字典

4. 关键技术与技术实现

4.1 关键技术介绍

4.1.1 Socket

Socket 是一种用于在计算机网络中进行通信的编程接口，它提供了一种标准的方式，让不同计算机之间可以进行数据交换。通过 Socket，程序可以在网络上建立连接、发送和接收数据，并且可以在不同计算机之间实现客户端和服务器之间的通信，包括 TCP 和 UDP 两种常见的协议。Socket 提供了服务端的方式和客户端的方式。服务端处理用户发起的请求[3]。客户端发送数据和接收服务端的响应数据。Python 库中也内置了 socket 模块，因此 socket 很适合做聊天室。

4.1.2 PyQt5

PyQt5 是一个流行的 Python 库，用于创建图形用户界面（GUI）应用程序，它是基于 Qt GUI 框架的 Python 绑定。PyQt5 提供了丰富的功能和工具，使开发者能够轻松创建交互式和跨平台的 GUI 应用程序。它包括了 Qt 的所有模块，涵盖了从窗口布局、按钮和文本框等基本组件到高级图形和多媒体功能的全面支持。开发者可以使用 PyQt5 进行快速开发，通过简单的 Python 代码实现复杂的 GUI 界面，同时利用其丰富的文档和社区支持，快速解决问题和学习新的功能。PyQt5 的跨平台兼容性让我的系统能够在 Windows、macOS 和 Linux 等多个操作系统上无缝运行，省去了为每个平台单独编写代码的需求和该系统以后的可拓展性和泛用性，所以我使用 PyQt5 来建立 UI 界面。

4.1.3 SQLite

SQLite 是一个开源的嵌入式关系数据库，实现了自给自足的、无服务器的、配置无需的、事务性的 SQL 数据库引擎。它是一个零配置的数据库，这意味着与其他数据库系统不同，比如 MySQL、PostgreSQL 等，SQLite 不需要在系统中设置和管理一个单独的服务。这也使得 SQLite 是一种非常轻量级的数据库解决方案，非常适合小型项目、嵌入式数据库或者测试环境中。因为本项目对数据库的需求不是很大，因此我打算使用 SQLite 作为本系统的数据库。

4.1.4 爬虫

爬虫是一种网络数据采集工具，通过模拟浏览器的行为，自动访问网页并提取其中的信息。它可以通过网络协议与网站进行通信，从网页中解析并抓取出所需的数据，如文本、图片、链接等，然后将这些数据保存到本地或者进行进一步的处理和分析[4]。爬虫技术被广泛应用于搜索引擎、数据挖掘、信息监控、商业情报等领域，能够快速、高效地从互联网上收集大量的信息，为用户提供有价值的数据支持。

4.1.5 多线程

Python 中的多线程可以通过内置的 threading 模块来实现。多线程是指在同一进程内运行的多个线程，每个线程都有独立的执行路径，可以提高程序的运行效率，允许多个代码块同时运行，从而实现并发操作[5]。而线程池是一种多线程处理形式，处理过程中将任务添加到队列，然后在创建线程后自动启动这些任务。

4.1.6 异步

异步（协程）爬虫是一种高效的爬取网页数据的方式，它可以同时处理多个请求，提高爬取速度，并减少资源的浪费[6]。传统的爬虫是同步的，即每次只能处理一个请求，必须等待上一个请求完成后才能进行下一个请求。这种方式效率较低，特别是在需要爬取大量数据的时候。而异步爬虫通过利用非阻塞的 IO 操作，可以在发送请求后立即进行下一个请求，从而充分利用网络资源，提高爬取效率。

4.1.7 Matplotlib

Matplotlib 最早是为了可视化癫痫病人的脑皮层电图相关的信号而研发，因为在函数的设计上参考了 MATLAB，所以叫做 Matplotlib。Matplotlib 是 Python 中最常用的可视化工具之一，可以非常方便地创建海量类型地 2D 图表和一些基本的 3D 图表，可根据数据集（DataFrame，Series）自行定义 x,y 轴，绘制图形（线形图，柱状图，直方图，密度图，散布图等等），能够解决大部分的需要[7]。Matplotlib 中最基础的模块是 pyplot。

4.1.8 LLM

大语言模型（英文：Large Language Model，缩写 LLM），也称大型语言模型，是一种人工智能模型，旨在理解和生成人类语言。它们在大量的文本数据上进行训练，可以执行广泛的任务，包括文本总结、翻译、情感分析等等[8]。LLM 的特点是规模庞大，包含数十亿的参数，帮助它们学习语言数据中的复杂模式。这些模型通常基于深度学习架构，如转化器，这有助于它们在各种 NLP 任务上取得令人印象深刻的表现。

4.1.9 VITS

VITS（Variational Inference with adversarial learning for end-to-end Text-to-Speech）是一种结合变分推理（variational inference）、标准化流（normalizing flows）和对抗训练的高表现力语音合成模型。VITS 通过隐变量而非频谱串联起来语音合成中的声学模型和声码器，在隐变量上进行随机建模并利用随机时长预测器，提高了合成语音的多样性，输入同样的文本，能够合成不同声调和韵律的语音[9]。

4.1.10 API

API(即应用程序编程接口)是一组定义的规则，使不同的应用程序能够相互通信。 它充当处理系统之间数据传输的中间层，使公司能够向外部第三方开发人员、业务合作伙伴和公司内部部门开放其应用程序数据和功能。API 中的定义和协议可帮助企业连接日常运营中使用的许多不同应用程序，从而节省员工时间并打破孤岛，有利于协作和创新。对于开发人员来说，API 文档提供了应用程序之间通信的接口，简化了应用程序集成[10]。

4.2 系统实现

4.2.1 服务端实现

服务端负责建立服务器。服务端通过绑定到特定的网络地址和端口，并监听来自客户端的连接请求。一旦建立连接，服务端负责接受客户端发送的数据，并根据预定的协议和逻辑进行处理。服务端还负责生成响应，并将其发送回客户端，完成一次通信交互。在多客户端情况下，服务端能够同时处理多个连接请求。

服务端模块使用 socket 建立通信和使用多线程来与多个客户端保持通信连接。因此，服务端的基本流程如图 4.1 所示。当每次建立起一个新的连接时，就使用 threading 模块创建一个新的线程，向新的线程中传入该客户端套接字的信息，并保持通信。

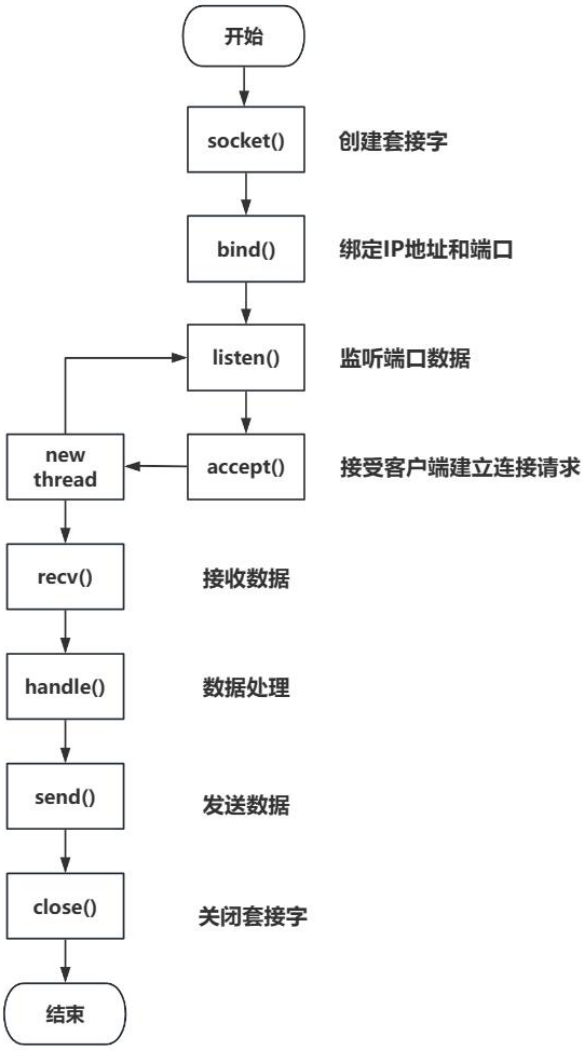


图 4.1 服务端模块运行流程图

服务端的运作服务器函数的伪代码如表 4.1 所示，服务器在创建好套接字后

会绑定地址并开始监听，然后一直阻塞运行直至有新的连接请求，当新的连接来临后，会用多线程给其分配一个单独的线程，最后启动线程。

表 4.1 服务端运作服务器函数的伪代码

运作服务器函数
1: 初始化一些参数
2: socket()创建套接字
3: bind()绑定地址
4: listen()监听
5: while True:
6: accept()阻塞状态等待新连接
7: Thread 开启多线程，一个客户端分配一个线程
8: start()启动线程

服务端的处理客户端函数的伪代码如表 4.2 所示，当客户端和服务端保持连接时，服务端会一直阻塞运行直至有新的消息接收，接收到消息后服务端会判断是给所有用户广播还是单独发给某个用户，如果某个用户退出，那么服务端会关闭和该用户的连接。

表 4.2 服务端处理客户端函数的伪代码

处理客户端函数
1: 初始化一些参数
2: while 客户端保持连接 do:
3: if 用户选择退出 then
4: 关闭连接
5: else if 用户选择私聊 then
6: 将消息单独发给指定用户
7: else
8: 将消息广播给所有用户
9: end while

要实现服务端，首先要定义一个处理客户端的函数，这个函数会进入一个死循环，一直试图接收服务端传来的消息，在接收到消息之前会一直阻塞，接收到新消息后就会继续执行后面的代码。首先解析客户端发来的 json 文件，获得一些参数，然后根据客户端的模式选择来调用对应操作函数，具体代码如下：

```
1. def hande_client(con, addr):
2.     """处理客户端"""
3.     connected = True
4.     while connected:
```

```
        msg = con.recv(BUFFER_SIZE).decode(FORMAT)  # 一直阻塞，直至收到新消息
        data = json.loads(msg)#客户端发来的数据
        ip = data["ip"]
        port = data["port"]
        mode = data["mode"]
        sender= data["sender"]
        receiver = data["receiver"]
        format = data["format"]
        online = data["online"]
        message = data["message"]
        users[sender] = addr #得到在线用户列表
        if message == DISCONNECT_MSG:
            connected = False
            del clients[addr]
        if mode =="group_chat":
            broadcast(sender, message)  # 向所有用户广播
        elif mode ==" group_alone ":
            alone(sender, message,receiver)#向指定用户发送
        elif mode ==" init ":
            init (sender, message)  # 初始化
con.close()  # 断开连接
```

在服务器对用户传来的 json 数据解析完成后，会根据用户的需求模式来调用对应的处理函数，如果为群聊模式，程序会将服务端处理后的消息广播给所有客户端，具体代码如下：

```
def broadcast(id, user, msg):
    """向所有已连接的客户端广播消息"""
    userList = []#先把所有在线用户遍历一遍，得到用户名列表传过去
    for k in users.values():
        userList.append(k)
    data = {
        "ip": SERVER,
        "port": PORT,
        "mode": "group_chat",
        "sender": sender,
        "receiver": "all",
        "format": "text",
        "online": userList,
        "message": msg,}
    for con in clients.values():
        con.send(json.dumps(data).encode(FORMAT))
```

如果为单聊模式，程序会将服务端处理后的消息发送给指定客户端。我定义了一个在线用户字典：key 为 user，value 为(ip+port)也就是 addr；一个客户

端字典：key 为 addr，value 为 con。首先要根据接收者的名字在在线用户字典中得到该用户对应的 addr，然后根据 addr 在客户端字典中得到该 addr 对应的 con。最后便可通过 con 中的 send 方法将 json 数据编码后发送给该用户。具体代码如下：

```
def alone(sender, msg, receiver):
    """给指定用户发消息"""
    # 先把所有在线用户遍历一遍，得到用户名列表传过去
    userList = []
    for k in users.keys():
        userList.append(k)
    data = {
        "ip": SERVER,#ip 地址
        "port": PORT,#端口号
        "mode": "group_alone",#模式
        "sender": sender,#发送者
        "receiver": receiver,#接收者
        "format": "text",#信息格式
        "online": userList,#在线用户列表
        "message": msg,#发送的信息
    }
    clients[users[receiver]].send(json.dumps(data).encode(FORMAT))
```

我加入了在线列表模块，用户可以查看当前有在聊天室的所有用户。要实现这个功能，肯定要通过服务器那边来统计所有接入它的客户端。服务器想监听所有客户端还必须要有客户端连接它才能监听。因此，我加入了初始化模式，当用户进入聊天室后，客户端会首先给服务端发送一个初始化的消息，服务端接收到这个消息后会进行处理，这样服务端就能统计所有接入它的客户端了，处理完成后服务端会将在线用户列表反馈给客户端，客户端就可以更新在线用户列表了。具体代码如下：

```
def init(sender, msg):
    """向所有已连接的客户端广播消息"""
    userList = []
    for k in users.keys():
        userList.append(k)
    data = {
        "ip": SERVER,
        "port": PORT,
        "mode": "init",
        "sender": sender,
        "receiver": "all",
        "format": "text",
        "online": userList,
```

```
        "message": msg,
    }
    for con in clients.values():
        con.send(json.dumps(data).encode(FORMAT))
```

只有这些操作还不够，服务端要为每个客户端单独开一个线程来并发处理他们的需求，这就需要用到多线程了，具体代码如下：

```
def start():
    """运作服务器"""
    s.bind(ADDR)  # 绑定地址
    s.listen(5)
    while True:
        con, addr = s.accept()  # 一直阻塞，直到有新的连接
        clients[addr] = con
        users[addr] = ''
        # 开启多线程处理客户端
        thread = threading.Thread(target=hande_client, args=(con, addr))  # 一个客户端占用一个线程
        thread.start()  # 启动线程
```

4.2.2 客户端实现

客户端可以发起连接，它通过指定目标服务器的地址和端口，向服务端发送连接请求，并在连接建立后，负责向服务端发送数据请求或命令，等待服务端的响应，并处理接收到的数据，从而实现与服务端的通信和数据交换。

客户端模块连接到服务端后，可以进行群聊和单聊。期间服务端和客户端会利用 Socket 库不断进行双向通信，其工作流程如图 4.2 所示。服务端使用 socket()创建套接字之后，通过 bind()方法绑定端口，然后使用 listen()对端口进行阻塞式地监听，等待客户端发来建立连接的请求。当接收到建立连接的请求时，使用 accept()方法接受客户端的连接请求，此后进入 recv()和 send()不断进行接收数据和发送数据的操作。最后，使用 close()关闭套接字终止程序。客户端同样使用 socket()和 close()来创建和关闭套接字，并使用 connect()向目标地址和端口发出建立连接的请求，建立连接成功之后就会进入 recv()和 send()中不断进行接收数据和发送数据的操作。

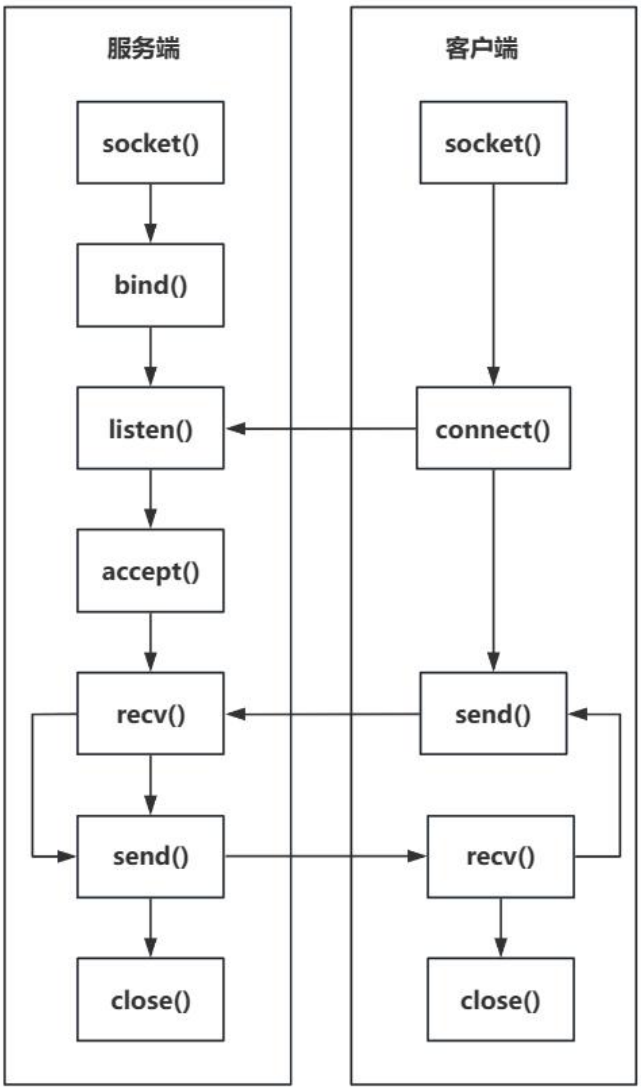


图 4.2 Socket 双向通信工作流程图

客户端的发送消息函数的伪代码如表 4.3 所示，客户端会一直询问用户要发送的消息，用户输入消息后，因为双向通信的数据传输只能是字节流的，所以客户端会对消息进行编码，使其变为字节流，然后发给服务器。如果用户选择退出，那么客户端会停止询问。

表 4.3 客户端的发送消息函数的伪代码

发送消息函数
1: 初始化一些参数
2: while 用户在线 do:
3: 询问用户输入
4: 将用户输入进行 encode()字节编码
5: 把消息发给服务器
6: end while

客户端的接收消息函数的伪代码如表 4.4 所示，客户端会一直阻塞运行直至接收到服务器的发过来的消息，然后对消息进行解码并展现给用户，如果用户退出，那么客户端将停止接收消息。

表 4.4 客户端的接收消息函数的伪代码

接收消息函数
1: 初始化一些参数
2: while 用户在线 do:
3: 阻塞运行，等待消息接收
4: 将消息进行 decode()字节解码
5: 把消息展现给用户
6: end while

本项目的客户端有两大核心模块：群聊模块和单聊模块，接下来我会详细介绍一下如何具体实现两个模块。

一、群聊模块

实现群聊的逻辑是有一个客户端发送消息给服务端后，服务端会将消息广播给所有连接的客户端。

本项目有 GUI 的实现，用的是 pyqt5 库，Qt 窗口本质上时一个死循环，启动后 Qt 代码会在主程序中执行，app.exec_()实会让主线程进入一个死循环，如果这个时候我们有一些耗时的操作在主程序中执行，那么 Qt 界面就会一直阻塞，可能会卡死。因此需要新开子线程来进行其他操作。在 Qt 启动时，我新开了个子线程，用来接收服务端发来的消息并一直死循环下去，在收到消息后要更新到界面上。具体代码如下：

```
1. def receive():  # 接收消息
2.     while True:
3.         msg = self.c.recv(self.BUFFER_SIZE).decode(self.FORMAT)
4.         self.ms.text_print.emit(msg)  # 发出信号
5. # 启动时发一次消息
6. def client_init():
7.         global global_id
8.         global global_user
9.         # 构建 JSON 格式的消息
10.        data = {
11.            "ip": self.client_ip,
12.            "port": self.client_port,
13.            "mode": "init",
14.            "id": global_id,
15.            "user": global_user,
16.            "receiver": "all",
```

```
            "format": "text",
            "online": [],
            "message": self.plainTextEdit_InPut.toPlainText(),
        }
        self.c.send(json.dumps(data).encode(self.FORMAT))  # 发送消息给服务器

if MenuWindow.re_flag:  # 创建一个接收消息线程
    thread_receive = threading.Thread(target=client_init)
    thread_receive.start()
    thread_receive = threading.Thread(target=receive)
    thread_receive.start()
MenuWindow.re_flag = False
```

做 Qt 开发要牢记一个原则：更新界面操作要在主线程中进行，在另外一个线程直接操作界面，可能会导致意想不到的问题，比如：输出显示不全，甚至程序崩溃。这时候就要引入信号这个机制。当客户端的子线程收到消息后会发送一个信号给主线程，主线程收到信号后就调用更新函数来将收到的信息更新到界面中。具体代码如下：

```
def update_text(self, msg):
    """更新消息显示框函数"""
    # 解析 JSON 信息
    try:
        data = json.loads(msg)
        ip = data["ip"]
        port = data["port"]
        mode = data["mode"]
        id = data["id"]
        user = data["user"]
        receiver = data["receiver"]
        format = data["format"]
        online = data["online"]
        message = data["message"]
        if mode == 'refresh':
            # 实例化列表模型，添加数据
            slm = QStringListModel()
            qList = online
            # 设置模型列表视图，加载数据列表
            slm.setStringList(qList)
            # 设置列表视图的模型
            self.listView_ChatList.setModel(slm)
            self.listView_ChatList_3.setModel(slm)
            # 单击触发自定义的槽函数
            self.listView_ChatList_3.clicked.connect(partial(self.clicked, qList))
        elif mode == 'alone_chat':
```

```
27.             self.create_bubble_alone(message, user)
28.             # 实例化列表模型，添加数据
29.             slm = QStringListModel()
30.             qList = online
31.             # 设置模型列表视图，加载数据列表
32.             slm.setStringList(qList)
33.             # 设置列表视图的模型
34.             self.listView_ChatList_3.setModel(slm)
35.             # 单击触发自定义的槽函数
36.             self.listView_ChatList_3.clicked.connect(partial(self.clicked, qList))
37.         elif mode == 'group_chat':
38.             self.create_bubble(message, user)
39.             # 实例化列表模型，添加数据
40.             slm = QStringListModel()
41.             qList = online
42.             # 设置模型列表视图，加载数据列表
43.             slm.setStringList(qList)
44.             # 设置列表视图的模型
45.             self.listView_ChatList.setModel(slm)
46.         else:
47.             # 实例化列表模型，添加数据
48.             slm = QStringListModel()
49.             qList = online
50.             # 设置模型列表视图，加载数据列表
51.             slm.setStringList(qList)
52.             # 设置列表视图的模型
53.             self.listView_ChatList.setModel(slm)
54.             self.listView_ChatList_3.setModel(slm)
55.             # 单击触发自定义的槽函数
56.             self.listView_ChatList_3.clicked.connect(partial(self.clicked, qList))
57.     except Exception as e:
58.         print("聊天出错！")
59.         print(e)
```

信号与槽是 Qt 特有的消息传输机制，在 Qt 中信号与槽用得十分广泛。在编程的过程中，我们都会遇到消息传递的事情，本质上就是发出命令，执行命令。比如单击窗口上一个按钮然后弹出一个对话框，那么就可以将这个按钮的单击信号和自定义的槽关联起来，信号是按钮的单击信号，槽实现了创建一个对话框并显示的功能。信号与槽就是实现对象之间通信的一种机制。

信号是当对象改变其状态时，信号就由该对象发射（emit）出去，而且对象只负责发送信号，它不知道另一端是谁在接收这个信号；槽用于接收信号，而且槽只是普通的对象成员函数。一个槽并不知道是否有任何信号与自己相连接。

Qt 编程中，会自带一些信号和槽函数，最常用的就是单击按钮触发某个事件，

我将发送消息的函数绑定到发送按钮上面，当用户单击发送按钮后就会触发发送消息函数，函数会读取用户输入的内容，将其发送给客户端并清空输入框。具体代码如下：

```
def send_msg_many(self):
    """发送群聊消息"""
    global NAME
    self.add_bubble_many_right(self.plainTextEdit_InPut.toPlainText(), NAME)
    def send():  # 发送函数
        try:
            print("已发送群聊消息")
            global IP
            global PORT
            # 构建 JSON 格式的消息
            data = {
                "ip": IP,
                "port": PORT,
                "mode": "group_chat",
                "sender": NAME,
                "receiver": "all",
                "format": "text",
                "online": [],
                "message": self.plainTextEdit_InPut.toPlainText(),
            }
            self.c.send(json.dumps(data).encode(self.FORMAT))  # 发送消息给服务器
            self.plainTextEdit_InPut.clear()  # 清空输入框
        except Exception as e:
            QMessageBox.warning(self, '发送失败', str(e))
            return
    # 子线程发送信号更新界面，发送和接收各开一个线程
    thread = threading.Thread(target=send)
    thread.start()
```

二、单聊模块

单聊模块比群聊模块稍微复杂一点，群聊模块只需要把所有连接到服务端的客户端对应的套接字对象遍历一遍，广播消息就行了。但单聊模块需要服务器将消息发给指定客户端，要发给指定客户端需要知道它对应的套接字对象。它的复杂之处就在于客户端无法获得它的套接字对象，客户端给服务端发数据只能发送用户名，并不能直接发送套接字对象，因此需要服务端根据客户端发来的用户名来找到该客户端对应的套接字对象。

用户首先要选择要单聊的对象。如何没选择就发送系统会提示不能发送。选择完成后就可以发送消息了，点击发送按钮后界面会将该用户发送的消息的聊天

气泡在主线程中更新到右侧，然后开一个子线程将消息发送到服务端。具体代码如下：

```
def send_msg_alone(self):
    """发送单聊消息"""
    global NAME
    self.add_bubble_alone_right(self.plainTextEdit_InPut_alone.toPlainText(), NAME)
    receiver = self.receiver
    if receiver is None:
        QMessageBox.warning(self, '警告', '未选择聊天对象！')
        return
    def send(receiver):  # 发送函数
        try:
            global IP
            global PORT
            # 构建 JSON 格式的消息
            data = {
                "ip": IP,
                "port": PORT,
                "mode": "group_alone",
                "sender": NAME,
                "receiver": receiver,
                "format": "text",
                "online": [],
                "message": self.plainTextEdit_InPut_alone.toPlainText(),
            }
            self.c.send(json.dumps(data).encode(self.FORMAT))  # 发送消息给服务器
            self.plainTextEdit_InPut_alone.clear()  # 清空输入框
        except Exception as e:
            QMessageBox.warning(self, '发送失败', str(e))
            return
    # 子线程发送信号更新界面，发送和接收各开一个线程
    thread = threading.Thread(target=send, args=(receiver,))
    thread.start()
```

在发送和接收完消息后，Qt 界面会更新消息气泡框，我是通过在界面初始化时隐藏掉气泡框，然后每收到一个消息就在界面上增加一个气泡框，用来显示聊天记录，如果是本人的消息会显示在左边，如果是别人的消息会显示在右边，方便查看。具体代码如下：

```
def add_bubble(self, ico, text, dir, user):  # 头像，文本，方向
    """增加气泡框函数"""
    self.widget = QtWidgets.QWidget(self.scrollAreaWidgetContents)
    self.widget.setLayoutDirection(dir)
    self.widget.setMinimumSize(QtCore.QSize(600, 200))
```

```
6.      self.widget.setMaximumSize(QtCore.QSize(600, 200))
7.      self.widget.setObjectName("widget")
8.      self.horizontalLayout_9 = QtWidgets.QHBoxLayout(self.widget)
9.      self.horizontalLayout_9.setContentsMargins(11, -1, 11, -1)
10.     self.horizontalLayout_9.setSpacing(7)
11.     self.horizontalLayout_9.setObjectName("horizontalLayout_9")
12.     self.verticalLayout_9 = QtWidgets.QVBoxLayout()
13.     self.verticalLayout_9.setObjectName("verticalLayout_9")
14.     self.label_Avatar = QtWidgets.QLabel(self.widget)
15.     self.label_Avatar.setMaximumSize(QtCore.QSize(50, 50))
16.     self.label_Avatar.setText("")
17.     self.label_Avatar.setPixmap(QtGui.QPixmap(ico))
18.     self.label_Avatar.setScaledContents(True)
19.     self.label_Avatar.setObjectName("label_Avatar")
20.     self.verticalLayout_9.addWidget(self.label_Avatar)
21.     spacerItem = QtWidgets.QSpacerItem(20, 20, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Preferred)
22.     self.verticalLayout_9.addItem(spacerItem)
23.     self.verticalLayout_9.setStretch(0, 3)
24.     self.verticalLayout_9.setStretch(1, 7)
25.     self.horizontalLayout_9.addLayout(self.verticalLayout_9)
26.     self.verticalLayout_8 = QtWidgets.QVBoxLayout()
27.     self.verticalLayout_8.setObjectName("verticalLayout_8")
28.     self.label_Time = QtWidgets.QLabel(self.widget)
29.     self.label_Time.setObjectName("label_Time")
30.     self.verticalLayout_8.addWidget(self.label_Time)
31.     self.textBrowser_OutPut = QtWidgets.QTextBrowser(self.widget)
32.     self.textBrowser_OutPut.setLayoutDirection(QtCore.Qt.LeftToRight)
33.     self.textBrowser_OutPut.setStyleSheet("background-color: rgba(71,121,214,20);\n"
34.                                           "font-family:微软雅黑;\n"
35.                                           "font-size:30px;\n"
36.                                           "color: #000000 ;\n"
37.                                           "")
38.     self.textBrowser_OutPut.setObjectName("textBrowser_OutPut")
39.     self.textBrowser_OutPut.setText(text)
40.     self.verticalLayout_8.addWidget(self.textBrowser_OutPut)
41.     self.verticalLayout_8.setStretch(0, 1)
42.     self.verticalLayout_8.setStretch(1, 9)
43.     self.horizontalLayout_9.addLayout(self.verticalLayout_8)
44.     self.verticalLayout_7.addWidget(self.widget)
45.     self.scrollArea_Chat.setWidget(self.scrollAreaWidgetContents)
46.     global global_user
47.     name = str(user) + '||' + str(datetime.datetime.now())
48.     _translate = QtCore.QCoreApplication.translate
```

```
49.     self.label_Time.setText(_translate("MainWindow", name))
50.     if user == global_user:
51.         # 设置文本靠右对齐
52.         self.label_Time.setAlignment(QtCore.Qt.AlignRight | QtCore.Qt.AlignVCenter)
```

4.2.3 注册与登录实现

注册与登录模块在应用中扮演着关键角色，通过验证用户身份、管理用户数据、增强系统安全性和提升用户体验等方面发挥作用，是保障系统安全和用户隐私、实现个性化服务的重要组成部分。

如图 4.3 所示，用户如果想注册他的账户，则需要点入注册界面输入其想要注册的账号、密码和用户名，如果其账号、密码和用户名都合乎规范的话则其密码会送入数据库中进行保存。

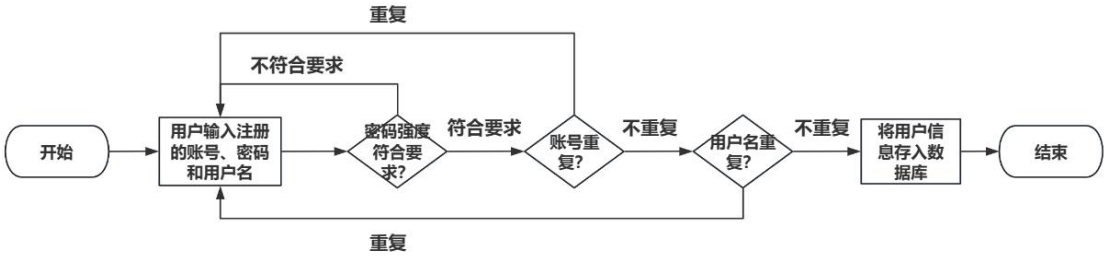


图 4.3 用户注册部分信息处理流程图

注册模块的伪代码如表 4.5 所示，用户注册账号后，系统会首先对用户的密码强度进行判断，如果强度不符合要求就要重新注册，以保证用户账号的安全性。然后系统会连接数据库并将用户的注册信息与数据库中的信息进行匹配，如果出现重复，那么用户需要重新注册。

表 4.5 注册模块的伪代码

注册模块
1: 初始化一些参数
2: 连接数据库
3: while 用户注册 do:
4: 用户输入账号、密码、用户名
5: 与数据库匹配
6: if 密强度不符合要求 then
7: continue
8: else if 账号已存在 then
9: continue

```
    else if 用户名已存在 then
        continue
    else
        break
end while
用户注册成功
```

要实现注册功能，首先要读取用户输入的内容，然后依次判断是否满足各个条件。首先看账号是否满足 11 位，接着看用户名是否满足 6 位，然后再判断两次密码是否相同，最后判断数据库中是否有重复账号，如果没有那就注册成功。具体代码如下：

```
def SingUp(self):
    """注册函数"""
    SignUpID = self.lineEdit_SignUpID.text()
    SignUpUser = self.lineEdit_SignUpUser.text()
    SignUpPW = self.lineEdit_SignUpPW.text()
    SignUpPW2 = self.lineEdit_SignUpPW2.text()
    if len(SignUpID) != 11:
        QMessageBox.critical(
            self, '错误', '账号不满足 11 位')
    elif len(SignUpUser) > 6:
        QMessageBox.critical(
            self, '错误', '用户名不满足小于 6 位')
    elif SignUpPW != SignUpPW2:
        QMessageBox.critical(
            self, '错误', '两次密码不相同')
    else:
        self.insert_signup(SignUpID, SignUpUser, SignUpPW)
        QMessageBox.information(
            self, '注册成功', '注册成功！')
def insert_signup(self, SignUpID, SignUpUser, SignUpPW):
    """插入数据库"""
    conn = sqlite3.connect("./data/Users.db")  # 建立一个基于硬盘的数据库实例
    cur = conn.cursor()  # 通过建立数据库游标对象，准备读写操作
    cur.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='T_users'")
# 判断表是否存在
    existing_table = cur.fetchone()
    if not existing_table:  # ID 列在插入数据时会自动递增
        cur.execute(
            "CREATE TABLE T_users(ID INTEGER PRIMARY KEY AUTOINCREMENT, SignUpID TEXT,
SignUpUser TEXT, SignUpPW TEXT)")  # 根据上表结构建立对应的表结构对象
    cur.execute("INSERT INTO T_users(SignUpID, SignUpUser, SignUpPW) VALUES(?, ?, ?)",
```

```
30.                 (SignUpID, SignUpUser, SignUpPW))  # 示例插入一行记录
31.     conn.commit()  # 保存提交，确保数据保存成功
32.     conn.close()  # 关闭与数据库的连接
```

如图 4.4 所示，用户登录模块中的信息处理过程是，用户先需要登录系统，系统收到用户输入的账号和密码后会自动将其输入的账号到数据库中进行匹配，如果存在对应的账号，则数据库会找到对应的密码，如果密码与用户输入的密码匹配则会进入系统的主界面。

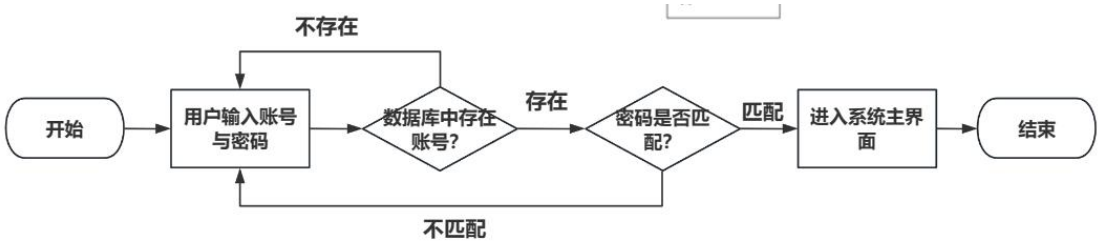


图 4.4 用户登录部分信息处理流程图

登录模块的伪代码如表 4.6 所示，用户登录账号后，系统会连接数据库并将用户的账号和密码与数据库中的信息进行匹配，如果匹配失败，那么用户需要重新登录。

表 4.6 登录模块的伪代码

登录模块
1: 初始化一些参数
2: 连接数据库
3: while 用户登录 do:
4: 用户输入账号、密码、用户名
5: 与数据库匹配
6: if 账号不存在 then
7: continue
8: else if 密码不匹配 then
9: continue
10: else
11: break
12: end while
13: 用户登录成功

要实现登录功能，首先要读取用户输入的内容，然后依次判断是否满足各个条件。首先链接数据库，然后遍历数据库中的所有账号，看是否有与输入相匹配的，如果有就检查该账号对应的密码是否与用户输入的密码相匹配，如果匹配就登录成功，否则登录失败。具体代码如下：

```
def LogIn(self):
    """登录函数"""
    LogInID = self.lineEdit_LogInID.text()
    LogInPW = self.lineEdit_LogInPW.text()
    user = self.check_login(LogInID, LogInPW)
    if user != None:
        # 设置用户账号和用户名为全局变量
        global global_id
        global global_user
        global_id = LogInID
        global_user = user
        QMessageBox.information(
            self,
            '登录成功',
            '欢迎使用本系统！')
        # 实例化另外一个窗口
        self.menuWd = MenuWindow()
        # 显示新窗口
        self.menuWd.show()
        # 关闭自己
        self.close()
    else:
        print('error')
        QMessageBox.critical(
            self,
            '错误',
            '账号或密码错误，请重新输入！')
def check_login(self, LogInID, LogInPW):
    """判断账号密码是否正确"""
    conn = sqlite3.connect("./data/Users.db")
    cur = conn.cursor()
    # 查询是否存在匹配的 LogInID
    cur.execute("SELECT * FROM T_users WHERE SignUpID=?", (LogInID,))
    row = cur.fetchone()
    if row:
        # 如果找到了匹配的 LogInID，检查密码是否匹配
        if row[3] == LogInPW:
            return row[2]  # 返回 SignUpUser
    conn.close()
    return None  # 如果没有匹配的 LogInID 或密码不匹配，返回 None
```

4.2.4 爬虫实现

爬虫可以通过网络协议与网站进行通信，能够快速、高效地从互联网上收集大量的信息，从网页中解析并抓取出每日天气和每日新闻的数据，然后将这些数据进行进一步的处理和分析并保存下来，使其能够以简洁明了的方式呈现在用户面前，为用户提供有价值的信息。

如图 4.5 所示，从指定的起始 URL 开始，爬虫发起 HTTP 请求获取网页内容，然后解析页面，提取所需数据或链接，将数据存储到本地或数据库中，并获取页面中的其他链接作为下一个目标。爬虫会不断重复这些操作，直到满足停止条件。

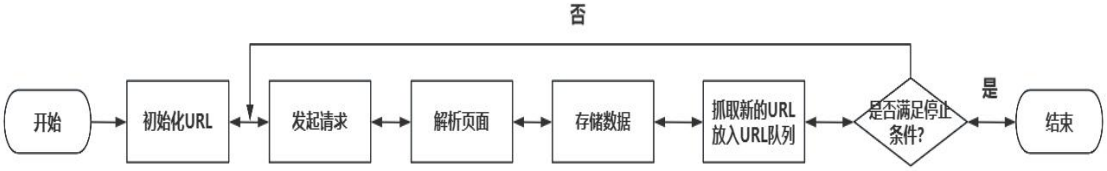


图 4.5 爬虫爬取网页流程图

爬虫模块的伪代码如表 4.7 所示，想要爬取每日天气和每日新闻网页，首先要初始化 URL，将这些网页的 URL 拷贝下来，接着浏览器发送一个 request 请求去获取 URL 的 html 文件，服务器把 response 文件对象发送回给浏览器。浏览器解析 response 中的 HTML。当所有的文件都下载成功后，爬虫模块会根据 HTML 语法结构，解析页面并完整地显示出来。然后将数据保存到本地。并接着爬取新的 URL，直到满足停止条件。要实现多线程爬虫，可以借助 threading 库，将爬虫任务分给多个线程，然后将这些线程同时执行。这样可以大大提高爬取速度，节省时间。

表 4.7 爬虫模块的伪代码

爬虫模块
1: 初始化 URL
2: while 不满足停止条件 do:
3: 浏览器发起请求
4: 解析网页源码
5: 数据处理并存储
6: 获取新的 URL 放入爬取队列
7: end while

本项目的爬虫有两大核心模块：每日天气爬虫模块和每日新闻爬虫模块，接下来我会详细介绍一下如何具体实现两个模块。

一、每日天气爬虫模块

天气预报我们每天都会关注，我们可以根据未来的天气增减衣物、安排出行，每天的气温、风速风向、相对湿度、空气质量等成为关注的焦点。本项目使用python中requests、aiohttp和BeautifulSoup库对中国天气网当天的数据进行爬取，保存为csv文件，之后用matplotlib、numpy、pandas库对数据进行可视化处理和分析，得到温度变化曲线，相对湿度变化曲线、降雨量变换曲线、风向雷达图等结果，使用户能快速地获得当天的天气信息。

要实现每日天气爬虫模块首先要进行数据获取，查看中国天气网的上海浦东新区网址 http://www.weather.com.cn/weather1d/101020600.shtml 如果想爬取不同的地区只需修改最后的101020600地区编号即可，weather1d代表当天。然后采用requests.get()方法，请求网页，如果成功访问，则得到的是网页的所有字符串文本。因为我使用的是异步爬虫，非常的快，爬取的过程中ip可能会被封，所有要用代理池，里面存有很多个可用的代理ip，每次爬取会随机选取代理池中的一个ip使用，且每当其中一个被封掉之后就会把它从代理池中剔除。具体代码如下：

```
async def get_text(self, session, province, city, county, url):
    """获取天气内容"""
    global proxy
    try:
        self.url_count += 1
        print(" 开始爬取: ", url)
        proxy = random.choice(self.PROXY_POOL)  # 从代理 IP 池中随机选择一个代理 IP
        async with session.get(url, headers=self.headers) as res:  # , proxy=proxy
            html = await res.text()
            print("爬取结果: ", url, len(html))
            await self.parse_text(province, city, county, html)  # 解析网页内容
    except Exception as e:  # 更新代理池
        print("爬取失败: ", e)
        self.PROXY_POOL.remove(proxy)
        print("剔除代理: ", proxy)
        if not self.PROXY_POOL:
            self.get_ip_port()
            time.sleep(1.1)
```

得到网页的所有字符串文本后要提取有用信息，本项目采用BeautifulSoup库对刚刚获取的字符串进行数据提取，首先对网页进行检查（按F12），找到需要获取数据的标签，如图4.6所示：

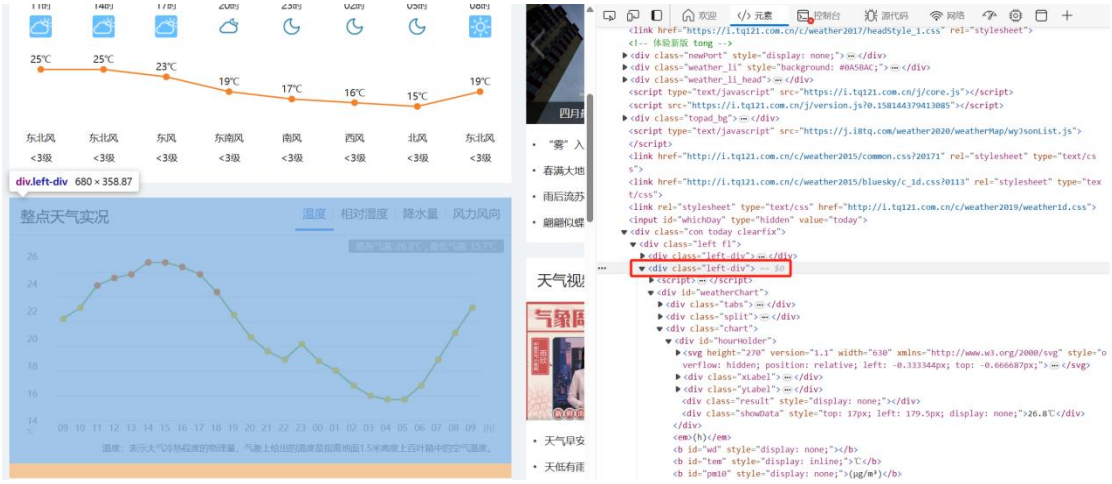


图 4.6 浦东新区每日天气网页元素

可以发现当天的数据信息在 div 标签中并且 class=left-div，所以可以使用 BeautifulSoup 对获取的网页文本进行查找 div 标签 class=left-div，找出他包含的所有 json 数据，之后提取 json 数据中相应的数据值。具体代码如下：

```
async def parse_text(self, province, city, county, html):
    """解析网页内容"""
    try:
        soup = BeautifulSoup(html, "html.parser")  # 解析 html 元素
        body = soup.body  # 获取 html 的 body 部分
        data = body.find_all('div', {'class': 'left-div'})  # 当日天气数据
        text = data[2].find('script').string  # 获取第三个 left-div 容器中的 json 内容
        text = text[text.index('=') + 1:-2]  # 移除 var observe24h_data = 将其变为 json
数据
        jd = json.loads(text)  # 解析 json 数据为 python 的字典格式
        day_data = jd['od']['od2']  # 找到当天的数据
        final_data = []  # 存放当天的数据
        count = 0
        for data in reversed(day_data):
            temp = []
            if count <= 24:
                temp.append(data['od21'])  # 添加时间
                temp.append(data['od22'])  # 添加当前时刻温度
                temp.append(data['od24'])  # 添加当前时刻风力方向
                temp.append(data['od25'])  # 添加当前时刻风级
                temp.append(data['od26'])  # 添加当前时刻降水量
                temp.append(data['od27'])  # 添加当前时刻相对湿度
                temp.append(data['od28'])  # 添加当前时刻空气质量
                final_data.append(temp)
                count = count + 1
        print("解析完成:", county)
        await self.write_to_csv(province, city, county, final_data)
```

```
27.     except:
28.         print('解析失败: ', {county})
```

提取 json 数据中相应的数据值后要将据添加到列表中，本项目引入 csv 库，利用 f_csv.writerow(header)和 f_csv.writerows(data)方法，分别写入表头和每一行的数据，具体代码如下：

```
1. async def write_to_csv(self, province, city, county, data):
2.     """保存为 csv 文件"""
3.     folder_path = f'./weather/{province}/{city}/{county}'
4.     if not os.path.exists(folder_path):
5.         os.makedirs(folder_path)
6.     async with aiofiles.open(f'{folder_path}/weather.csv', 'w', encoding='utf-8',
errors='ignore', newline='') as f:
7.         header = ['小时', '温度', '风力方向', '风级', '降水量', '相对湿度', '空气质量']
8.         f_csv = csv.writer(f)
9.         await f_csv.writerow(header)
10.        # await f_csv.writerows(data)
11.        for row in data:  # 逐行写入
12.            await f_csv.writerow(row)
13.    print(f'{county}写入完成')
```

保存完当日天气数据后就要对数据进行可视化分析，对于一天的温度，采用 matplotlib 中 plt.plot()方法绘制出一天 24 小时的温度变化曲线，并用 plt.text()方法点出最高温、最低温和平均温度，并画出平均温度线。具体代码如下：

```
1. def tem_curve(self, data, file_name):
2.     """温度曲线绘制"""
3.     hour = list(data['小时'])
4.     tem = list(data['温度'])
5.     for i in range(0, 25):  # 没有数据就赋值为前面一个
6.         if math.isnan(tem[i]) == True:
7.             tem[i] = tem[i - 1]
8.     tem_ave = sum(tem) / 25  # 求平均温度
9.     tem_max = max(tem)
10.    tem_max_hour = tem.index(tem_max)  # 求最高温度
11.    tem_min = min(tem)
12.    tem_min_hour = tem.index(tem_min)  # 求最低温度
13.    x = []  # 实际横坐标,可以当成索引
14.    x_label = []  # 展示横坐标
15.    y = []
16.    for i in range(0, 25):
17.        x.append(i)
18.        x_label.append(hour[i])
19.        y.append(tem[i])
```

```
    plt.figure(1)
    plt.plot(x, y, color='red', label='温度')  # 画出温度曲线
    plt.scatter(x, y, color='red')  # 点出每个时刻的温度点
    plt.plot([0, 25], [tem_ave, tem_ave], color='blue', linestyle='--', label='平均温度')  # 画出平均温度虚线
    plt.text(25 + 0.15, tem_ave + 0.15, str(round(tem_ave, 2)) + '℃', ha='center', va='bottom',
             fontsize=10.5)  # 标出平均温度
    plt.text(tem_max_hour + 0.15, tem_max + 0.15, str(tem_max) + '℃', ha='center', va='bottom',
             fontsize=10.5)  # 标出最高温度
    plt.text(tem_min_hour + 0.15, tem_min + 0.15, str(tem_min) + '℃', ha='center', va='bottom',
             fontsize=10.5)  # 标出最低温度
    plt.xticks(x, x_label)
    plt.legend()
    plt.title(f'{file_name}一天温度变化曲线图')
    plt.xlabel('时间/h')
    plt.ylabel('摄氏度/℃')
    folder_path = self.folder_path + file_name
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
    file_png = f'{folder_path}/{file_name}_tem_curve.png'
    if os.path.exists(file_png):
        os.remove(file_png)  # 删除已存在的同名文件
    plt.savefig(file_png, dpi=300, bbox_inches='tight')
    plt.close()
    print(f'{file_name}温度曲线绘制完成')
```

对于一天的相对湿度，采用 matplotlib 中 plt.plot()方法绘制出一天 24 小时的湿度变化曲线，并画出平均相对湿度线，曲线图展示了一天内每小时的相对湿度变化情况，包括平均相对湿度、最高相对湿度和最低相对湿度，并在图中标出了这些关键数据点。具体代码如下：

```
def hum_curve(self, data, file_name):
    """相对湿度曲线绘制"""
    hour = list(data['小时'])
    hum = list(data['相对湿度'])
    for i in range(0, 25):
        if math.isnan(hum[i]) == True:
            hum[i] = hum[i - 1]
    hum_ave = sum(hum) / 25  # 求平均相对湿度
    hum_max = max(hum)
    hum_max_hour = hum.index(hum_max)  # 求最高相对湿度
    hum_min = min(hum)
```

```
    hum_min_hour = hum.index(hum_min)  # 求最低相对湿度
    x = []
    x_label = []
    y = []
    for i in range(0, 25):
        x.append(i)
        x_label.append(hour[i])
        y.append(hum[i])
    plt.figure(2)
    plt.plot(x, y, color='blue', label='相对湿度')  # 画出相对湿度曲线
    plt.scatter(x, y, color='blue')  # 点出每个时刻的相对湿度
    plt.plot([0, 25], [hum_ave, hum_ave], c='red', linestyle='--', label='平均相对湿度')
# 画出平均相对湿度虚线
    plt.text(25 + 0.15, hum_ave + 0.15, str(round(hum_ave, 2)) + '%', ha='center',
va='bottom',
             fontsize=10.5)  # 标出平均相对湿度
    plt.text(hum_max_hour + 0.15, hum_max + 0.15, str(hum_max) + '%', ha='center',
va='bottom',
             fontsize=10.5)  # 标出最高相对湿度
    plt.text(hum_min_hour + 0.15, hum_min + 0.15, str(hum_min) + '%', ha='center',
va='bottom',
             fontsize=10.5)  # 标出最低相对湿度
    plt.xticks(x, x_label)
    plt.legend()
    plt.title(f'{file_name}一天相对湿度变化曲线图')
    plt.xlabel('时间/h')
    plt.ylabel('百分比/%')
    folder_path = self.folder_path + file_name
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
    file_png = f'{folder_path}/{file_name}_hum_curve.png'
    if os.path.exists(file_png):
        os.remove(file_png)  # 删除已存在的同名文件
    plt.savefig(file_png, dpi=300, bbox_inches='tight')
plt.close()
```

对于一天的降雨量，采用 matplotlib 中 plt.bar ()方法绘制出一天 24 小时的降雨量的变化柱状图，并画出平均降雨量，柱状图展示了一天内每小时的降雨量变化情况，包括平均降雨量、最高降雨量和最低降雨量，并根据不同的降雨量程度采用不同的颜色进行标识，以直观展示降雨强度。具体代码如下：

```
def rain_curve(self, data, file_name):
    """降雨量曲线绘制"""
    hour = list(data['小时'])
    rain = list(data['降水量'])
```

```
    for i in range(0, 25):
        if math.isnan(rain[i]) == True:
            rain[i] = rain[i - 1]
    rain_ave = sum(rain) / 25  # 求平均降雨量
    rain_max = max(rain)
    rain_max_hour = rain.index(rain_max)  # 求最高降雨量
    rain_min = min(rain)
    rain_min_hour = rain.index(rain_min)  # 求最低降雨量
    x = []
    x_label = []
    y = []
    for i in range(0, 25):
        x.append(i)
        x_label.append(hour[i])
        y.append(rain[i])
    plt.figure(3)
    for i in range(0, 25):
        if y[i] <= 10:
            plt.bar(x[i], y[i], color='lightgreen', width=0.7)  # 小雨
        elif y[i] <= 25:
            plt.bar(x[i], y[i], color='wheat', width=0.7)  # 中雨
        elif y[i] <= 50:
            plt.bar(x[i], y[i], color='orange', width=0.7)  # 大雨
        elif y[i] <= 100:
            plt.bar(x[i], y[i], color='orangered', width=0.7)  # 暴雨
        elif y[i] <= 250:
            plt.bar(x[i], y[i], color='darkviolet', width=0.7)  # 大暴雨
        elif y[i] > 250:
            plt.bar(x[i], y[i], color='maroon', width=0.7)  # 特大暴雨
    plt.plot([0, 25], [rain_ave, rain_ave], c='black', linestyle='--')  # 画出平均降雨
量虚线
    plt.text(25 + 0.15, rain_ave, str(round(rain_ave, 2)) + 'mm', ha='center',
va='bottom',
             fontsize=10.5)  # 标出平均降雨量
    plt.text(rain_max_hour + 0.15, rain_max, str(rain_max) + 'mm', ha='center',
va='bottom',
             fontsize=10.5)  # 标出最高降雨量
    plt.text(rain_min_hour + 0.15, rain_min, str(rain_min) + 'mm', ha='center',
va='bottom',
             fontsize=10.5)  # 标出最低降雨量
    plt.xticks(x, x_label)
    plt.title(f'{file_name}一天降雨量变化曲线图')
    plt.xlabel('时间/h')
    plt.ylabel('深度/mm')
```

```
    folder_path = self.folder_path + file_name
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
    file_png = f'{folder_path}/{file_name}_rain_curve.png'
    if os.path.exists(file_png):
        os.remove(file_png)  # 删除已存在的同名文件
    plt.savefig(file_png, dpi=300, bbox_inches='tight')
    plt.close()
```

对于一天的风力和风向，由于风力风向使用极坐标的方式展现较好，所以采用的是极坐标的方式展现一天的风力风向图，将圆分为 8 份，每一份代表一个风向，半径代表平均风力，并且随着风级增高，蓝色加深，具体代码如下：

```
def wind_radar(self, data, file_name):
    """风向雷达图"""
    wind = list(data['风力方向'])
    wind_speed = list(data['风级'])
    plt.figure(4)
    for i in range(0, 25):
        if wind[i] == "北风":
            wind[i] = 90
        elif wind[i] == "南风":
            wind[i] = 270
        elif wind[i] == "西风":
            wind[i] = 180
        elif wind[i] == "东风":
            wind[i] = 360
        elif wind[i] == "东北风":
            wind[i] = 45
        elif wind[i] == "西北风":
            wind[i] = 135
        elif wind[i] == "西南风":
            wind[i] = 225
        elif wind[i] == "东南风":
            wind[i] = 315
    degs = np.arange(45, 361, 45)
    temp = []
    for deg in degs:
        speed = []
        # 获取 wind_deg 在指定范围的风速平均值数据
        for i in range(0, 25):
            if wind[i] == deg:
                speed.append(wind_speed[i])
        if len(speed) == 0:
            temp.append(0)
```

```
        else:
            temp.append(sum(speed) / len(speed))
    N = 8
    theta = np.arange(0. + np.pi / 8, 2 * np.pi + np.pi / 8, 2 * np.pi / 8)
    # 数据极径
    radii = np.array(temp)
    # 绘制极区图坐标系
    plt.axes(polar=True)
    # 定义每个扇区的 RGB 值（R,G,B），x 越大，对应的颜色越接近蓝色
    colors = [(1 - x / max(temp), 1 - x / max(temp), 0.6) for x in radii]
    plt.bar(theta, radii, width=(2 * np.pi / N), bottom=0.0, color=colors)
    plt.title(f'{file_name}一天风向雷达图', x=0.2, fontsize=20)
    folder_path = self.folder_path + file_name
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
    file_png = f'{folder_path}/{file_name}_wind_radar.png'
    if os.path.exists(file_png):
        os.remove(file_png)  # 删除已存在的同名文件
    plt.savefig(file_png, dpi=300, bbox_inches='tight')
plt.close()
```

只爬取浦东新区肯定是不够的，所以可以用异步快速爬取全国所有地区的每日天气数据，首先要有一个记录全国所有地区天气代码的 xml 文件，接着遍历这个文件中的所有天气代码并添加到字典中，然后将字典中所有的天气代码添加到任务队列中，最后将任务列表交给异步爬虫爬取。具体代码如下：

```
async def start(self, provinces):
    """启动爬虫"""
    # 加载 XML 文件
    tree = ET.parse('./cityCode2.xml')
    root = tree.getroot()
    # 定义一个空列表来存储提取的信息
    weather_data = []
    # 遍历 XML 树
    for province in root.findall('province'):
        province_name = province.attrib['name']
        for city in province.findall('city'):
            city_name = city.attrib['name']
            for county in city.findall('county'):
                county_name = county.attrib['name']
                weather_code = county.attrib['weatherCode']
                # 将提取的信息存储为字典
                data = {
                    'province_name': province_name,
                    'city_name': city_name,
```

```
                    'county_name': county_name,
                    'weather_code': weather_code
                }
                # 将字典添加到列表中
                weather_data.append(data)
    timeout = aiohttp.ClientTimeout(total=600)  # 将超时时间设置为 600 秒
    connector = aiohttp.TCPConnector(force_close=True, limit=50)  # 将并发数量降低
    async with aiohttp.ClientSession(connector=connector, timeout=timeout) as session:
        # async with aiohttp.ClientSession() as session:
        tasks = [asyncio.create_task(
            self.get_text(session, data['province_name'], data['city_name'],
data['county_name'],
                          self.base_url + data['weather_code'] + '.shtml')) for data in
weather_data if
            data['province_name'] == provinces]
        for task in tasks:
            # await asyncio.sleep(random.uniform(1.1, 1.3))  # 添加延迟
            await task
        done, pending = await asyncio.wait(tasks)
```

二、每日新闻爬虫模块

新闻我们每天也会关注，通过新闻我们可以知道每天发生的天下大事。本项目使用 python 中 requests、aiohttp、BeautifulSoup 库对澎湃新闻网当天的数据进行爬取，保存为 txt 文件，为用户打造每日新闻资讯速达小工具。

要实现每日新闻爬虫模块首先要进行数据获取，查看澎湃新闻网的科技栏目网址 https://www.thepaper.cn/channel_119908 如果想爬取不同的栏目只需修改最后的 119908 编号即可。进入网页后，按 F12 打开检查工具，然后定位到文章的标题，如图 4.7 所示：

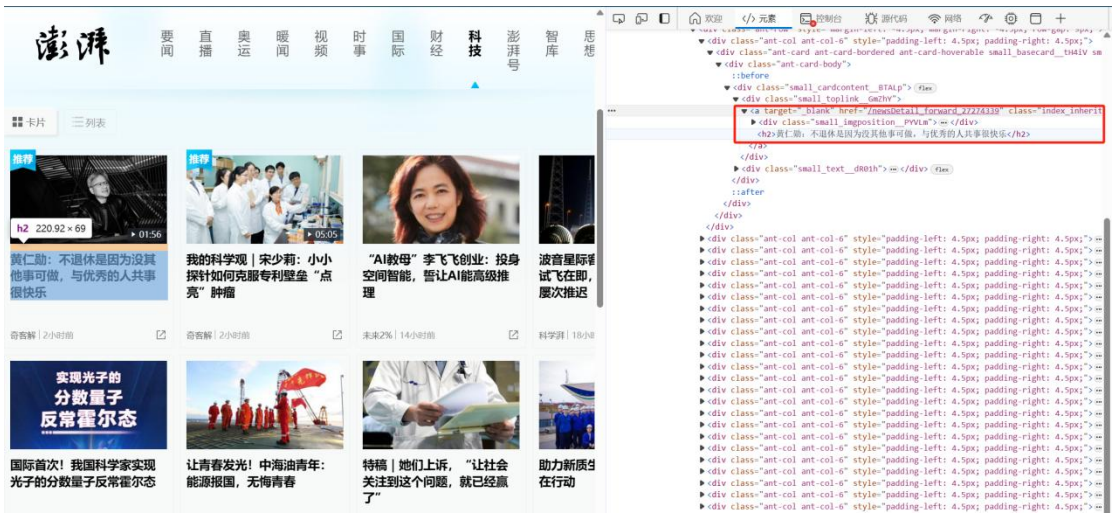


图 4.7 澎湃新闻科技栏目的网页元素

可以看到右边控制台是定位到了 h2，而 h2 里面有个 a 标签，a 标签里面有个 href，对应着有一条链接，点击去这个链接就得到对应的新闻详情页了，那么这个 href 对应的链接就是要找新闻详情页的链接，接下来就要用代码把这些链接全部提取出来，首先要提取出所有的 a 标签,使用正则表达式提取出所有包含 newsDetail_forward_的元素，然后提取 href 元素中的链接，并在前面加上 https://www.thepaper.cn/，这样就得到了科技栏目所有标题的链接，具体代码如下：

```
def get_href(self, url, name):
    """获取新闻链接"""
    try:
        print(" 发送请求: ", url)
        response = requests.get(url, headers=self.headers)  # ,proxies=self.proxies
        response.encoding = 'utf-8'  # 设置编码为 utf-8，要不然会打印出乱码
        html = response.text
        print("得到结果: ", url, len(html))
        p = re.compile(r'newsDetail_forward_\d+')  # 正则表达式找出所有 href
        text = p.findall(html)
        href_list = []
        for aa in text:  # 将 href 变成可访问链接
            href_list.append(self.base_url + aa)
            self.url_count += 1

        asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())  # 防止报错
        asyncio.run(self.start(href_list, name))  # 创建一个新的事件循环并运行异步函数
    except Exception as e:
        print("爬取失败: ")
        print(e)
```

得到科技栏目所有标题的链接后就要对这些链接继续爬取，得到每个标题对应的具体内容，首先要提取出所有的 div 标签,并且 class= index_contentWrap，然后提取 p 元素的所有文本，这样就得到每个标题对应的内容，然后将内容存入 txt 文件中，文件名就是新闻标题。具体代码如下：

```
async def get_text(self, session, url, name):
    """获取新闻内容"""
    folder_path = f'./news/{name}'
    if os.path.exists(folder_path):  # 删除旧新闻
        shutil.rmtree(folder_path)
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)

    print(" 开始爬取: ", url)
```

```
    async with session.get(url, headers=self.headers, verify_ssl=False) as res:
        html = await res.text()
        print("爬取结果: ", url, len(html))
        try:
            p = re.compile(r'<h1.*?">(.*?)</h1>')
            title = p.findall(html)[0]
            soup = BeautifulSoup(html, "html.parser")  # 解析 html 元素
            body = soup.body  # 获取 html 的 body 部分
            text = body.find('div', {'class': 'index_cententWrap__Jv8jK'})  # 获取新闻内
容

            news = text.findAll('p', {'class': ''})  # 获取新闻段落
            async with aiofiles.open(f"./news/{name}/{title}.txt", "w", encoding="utf-8")
as f:
                for paragraph in news:  # 获取每段的文本
                    if paragraph.text.strip():  # 检查段落文本是否为空
                        ppp = '    ' + paragraph.text + '\n'
                        await f.write(ppp)
            print('写入成功: ', title)
        except Exception as e:
            print("解析出错! ", e)
```

只爬取科技栏目肯定是不够的，所以可以用线程池同时爬取多个栏目的每日新闻数据，首先要把新闻各个栏目的 URL 字典设计好，然后编写一个启动函数，用于为各个 URL 创建单独的线程并启动爬虫函数来爬取和解析。具体代码如下：

```
def main(self):
    """线程池获取链接"""
    folder_path = './news'
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)  # 新建文件夹
    news_column_urls = {'时事': 'https://www.thepaper.cn/channel_25950',
                        '国际': 'https://www.thepaper.cn/channel_122908',
                        '财经': 'https://www.thepaper.cn/channel_25951',
                        '科技': 'https://www.thepaper.cn/channel_119908',
                        '智库': 'https://www.thepaper.cn/channel_119489',
                        '思想': 'https://www.thepaper.cn/channel_25952',
                        '生活': 'https://www.thepaper.cn/channel_25953'
                        }
    with concurrent.futures.ThreadPoolExecutor(7) as executor:
        for name, url in news_column_urls.items():
            executor.submit(self.get_href, url, name)
async def start(self, urls, name):
    """启动爬虫"""
    async with aiohttp.ClientSession() as session:
```

```
20.         tasks = [asyncio.create_task(self.get_text(session, url, name)) for url in urls]
21.         for task in tasks:
22.             # await asyncio.sleep(0.5)  # 添加延迟
23.             await task
24.         done, pending = await asyncio.wait(tasks)
```

4.2.5 虚拟伙伴实现

虚拟伙伴具有处理用户输入文本、生成相应回答以及以特定角色的语音形式输出的功能。通过对用户输入进行处理和分析，能够根据预设的大语言模型生成个性化的回答，并利用语音合成技术将回答以特定角色的语音形式输出，实现了与用户的沟通交流，提供个性化的服务体验。

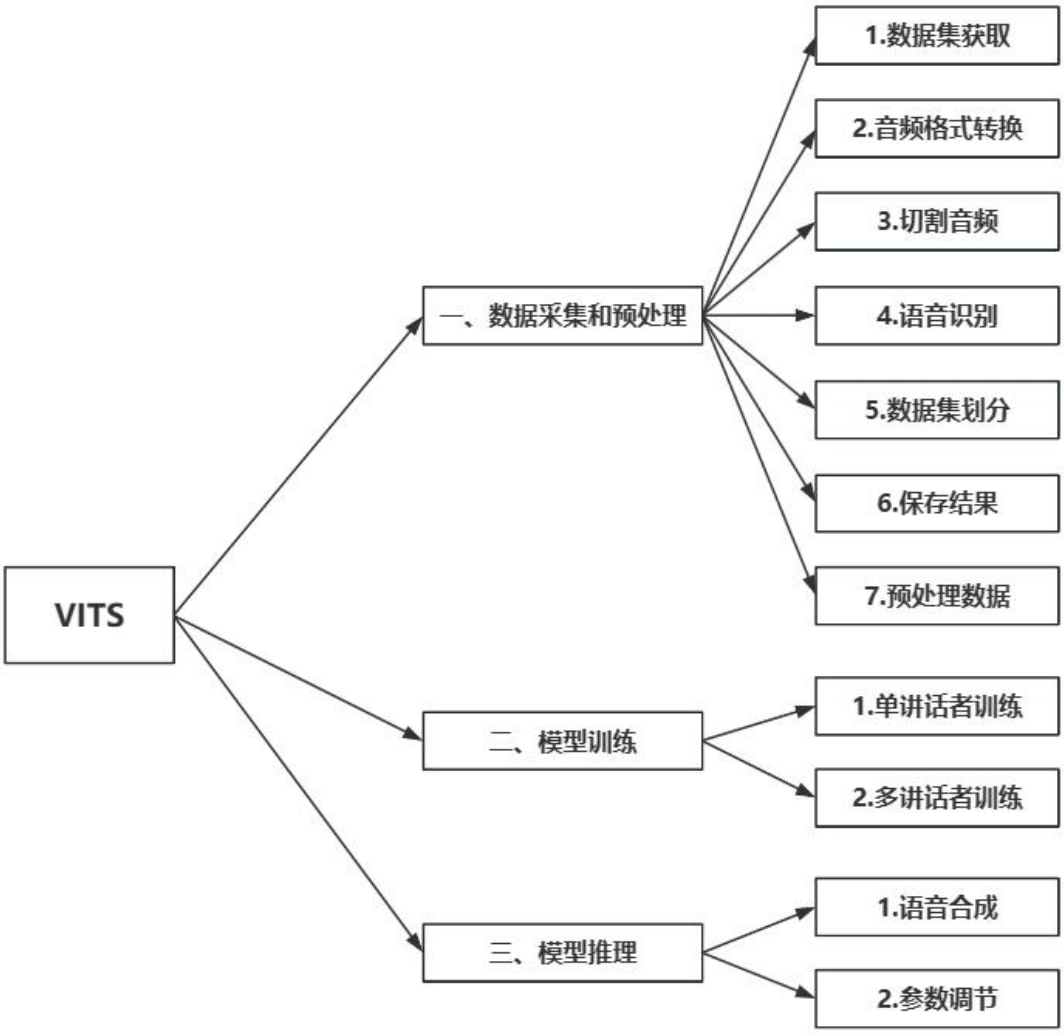


图 4.8 VITS 语音合成使用流程图

我选择的是 VITS 作为语音合成的基准模型，使用 VITS 需要经过如图 4.8 中

的步骤。具体细节如下：

1.数据集采集和预处理

首先，要进行数据集的获取，可以网上找相关的音频，单对音频质量的要求比较高，背景不能有噪声，要不然模型从语调和声音清晰度上会不理想。然后，将数据的音频格式要求转换为：wav，单声道、22050Hz，PCM 16bit。

音频格式转换完成后，需要把长影片切割成小段音频，不然模型训练负担过重，使用 auto-slicer 项目做音频切割，通过静音检测这种简单而有效的方式，根据预设的条件和参数，将音频文件切割并输出切割后的音频片段。然后标注音频片段，获取标注文本。

标注完成后将数据集划分为训练集和验证集，并保存到指定目录中，然后清洗数据，将一些质量不太好的数据集剔除。

2.模型训练

模型构建与训练的处理流程如图 4.9 所示。数据集准备完成后，在训练开始前要进行数据预处理，进行数据增强，这样可以提高模型的鲁棒性。然后就可以将数据集送入模型进行训练，训练完成后要检查模型的指标，如果基准模型达不到想要的指标我便会重新进行数据收集等步骤，直至模型达标。最后导出最好的模型。

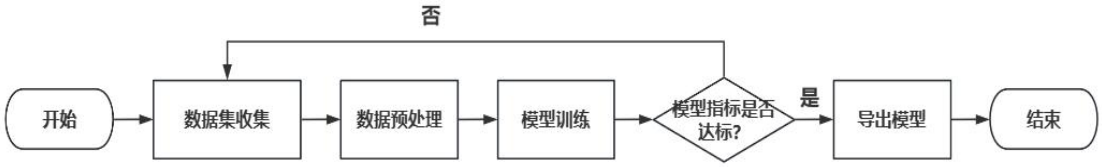


图 4.9 模型构建与训练流程图

3.模型推理

得到最好的模型后就可以部署在本地使用了，使用流程如图 4.10 所示。用户输入文本后，系统会调用比较先进的大语言模型进行回答。得到回答后调用模型，将回答的文本输入给模型进行语音合成，合成成功后将语音输出给用户。

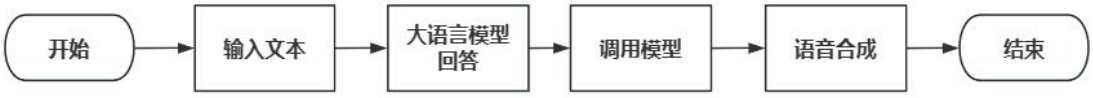


图 4.10 模型使用流程图

AI 语音合成的伪代码如表 4.8 所示，我使用的深度学习框架是 pytorch。首先要导入 pytorch 框架，通过 pytorch 的 model 模块加载.pth 模型，并初始化一些参数。然后便开始获取用户的输入，将用户的输入文本模型发给大语言模型，得到回答后将回答的文本交给之前调用的模型处理，处理完成后会生成音频文件，最终语音输出给用户。

表 4.8 AI 语音合成的伪代码

AI 语音合成
1: 导入 pytorch 框架
2: 加载.pth 模型
3: 初始化一些参数
4: while 用户输入文本 do:
5: 通过 API 调用大语言模型
6: 获得回答
7: 将回答的文本交给模型处理
8: 生成音频文件
9: 输出语音
10: end while

本项目的虚拟伙伴有两大核心模块：大语言模型模块和语言合成模块，接下来我会详细介绍一下如何具体实现两个模块。

一、大语言模型模块

众所周知，每个人都有其独特的性格特点，正是因为如此，每个人才独一无二。二次元游戏的角色也不例外，正因为每个角色都有其独特的性格特点，才能让她们有辨识度，才能使每个角色都令人印象深刻，有血有肉。因此首先我要让大语言模型扮演不同的角色，就要先赋予它不同角色的性格特点，实现角色扮演，一人饰演多角。我使用的是国产大语言模型——智谱 AI 中的 charglm-3 模型，每次跟它对话时，都要首先指定用户部分的名字和背景介绍以及智谱 AI 部分的名字和性格特点。同时我还创建了一个记录历史对话的列表，每次对话的内容会以字典的形式存入列表中，并将其传给大语言模型，这样就可以实现多轮上下文对话了，得到智谱 AI 的回答后会返回最新的回答。具体代码如下：

```
1. def chat_zhipu(self, prompt, select):
2.     """和智谱 AI 聊天"""
3.     zhipuai.api_key = "5857cb6dc4405f76c0f5eaff81d267b5.UBVyEEuWSr52VbbR"
4.     new_prompt = {
5.         "role": "user",
6.         "content": prompt
7.     }
8.     print("开始聊天")
9.     if select == 'paimeng':
10.         self.prompt_paimeng.append(new_prompt)  # 记录用户问题
11.         response = zhipuai.model_api.sse_invoke(
12.             model="charglm-3",
13.             meta={
```

```
14.                 "user_info": "我叫旅行者，旅行者是游戏《原神》中的主角。从世界之外漂流而来的旅行者，被神带走血亲，自此踏上寻找七神之路。",
15.                 "bot_info": "游戏开篇旅行者坠入提瓦特世界并苏醒过来后，一边通过在沙滩上绘画描述自己的遭遇一边回忆起在水中钓到的奇妙生物，派蒙。之后派蒙自愿作为向导与旅行者一同寻找家人。游戏中旅行者少言寡语，绝大部分交流与非战斗互动都是通过派蒙进行的。派蒙在剧情中与旅行者经历了各式各样的冒险。虽然她个性贪吃爱财，听到有宝藏、奖励丰厚、有好吃的等字眼就会立刻变得热心起来，催促旅行者去帮忙，但非常关心旅行者的安危。同时非常珍视与旅行者的友谊，屡次强调自己是旅行者最好的伙伴，是不会和旅行者分开的。并衷心的希望旅行者能找到他/她的家人。",
16.                 "bot_name": "派蒙",
17.                 "user_name": "旅行者"
18.             },
19.             prompt=self.prompt_paimeng,
20.             incremental=True
21.         )
22.         res = ''
23.         for event in response.events():
24.             res += event.data
25.         new_res = {
26.             "role": "assistant",
27.             "content": res
28.         }
29.         self.prompt_paimeng.append(new_res)  # 记录模型回答
30.         print("聊天结束！")
31.         return res
```

二、语言合成模块

只有独特的性格特点还不够，还要有其独特的声线，这样才能更有辨识度。在得到智谱 AI 的回答后不仅要输出文本，同时还要输出声音。实现语言合成，首先要把文本交给语音合成模型，然后还要有参考文本及对应的音频文件，并指导要用什么语言（我这边用的是中文 zh）用什么 sovits 模型和 gpt 模型。因为本机显卡有限，所以我去 AutoDL 服务器上租了一个 4090 显卡，并把模型部署在服务器上，这样每次推理只需要向服务器发送一个 POST 请求，由服务器来推理，推理完成后将结果返回到本机，然后播放就行了，这样做可以大大提高推理速度。具体代码如下：

```
1. def get_wav(self, text):
2.     """获取音频文件"""
3.     if self.select == 'paimeng':
4.         data = {
5.             "refer_wav_path": "./reference_audio/说话-既然罗莎莉亚说足迹上有元素力，用元素视野应该能很清楚地看到吧。.wav",
6.             "prompt_text": "既然罗莎莉亚说足迹上有元素力，用元素视野应该能很清楚地看到吧。",
```

```
            "prompt_language": "zh",
            "text": text,
            "text_language": "zh",
            "custom_sovits_path": "./SoVITS_weights/paimeng-jian_e50_s1600.pth",
            "custom_gpt_path": "./GPT_weights/paimeng-jian-e15.ckpt"
        }
        response = requests.post("http://localhost:9880", json=data)
        print("开始合成")
        if (response.status_code == 400):
            raise Exception(f"语惑 GPT-SOVITS 日玩指说:{response.message}")
        with open("success.wav", "wb") as f:
            f.write(response.content)
            print("合成成功！")
        print("开始播放")
        filename = "success.wav"
        winsound.PlaySound(filename, winsound.SND_FILENAME)
        print("播放完成")
```

5.系统运行结果

5.1 运行环境

5.1.1 硬件环境

本项目所用硬件环境如下：
CPU：AMD Ryzen 9 5900HX with Radeon Graphics 3.30 GHz
内存：32.0 GB
GPU：NVIDlA GeForce RTX 3080 Laptop GPU

5.1.2 软件环境

本项目所用软件环境如下：
操作系统：Windows 11 家庭中文版 64 位操作系统，基于 x64 的处理器
编程语言：Python
集成开发环境：Pycharm

5.2 运行与测试结果

5.2.1 系统运行界面

打开终端，先运行 server.py 启动服务端，然后再开一个终端运行 main.py 即可进入系统。首先会弹出登录窗口（如图 5.1 所示），用户如果已有账号可以在此登录，如果没有账号可在注册窗口选择注册（如图 5.2 所示）。

图 5.1 登录窗口

图 5.2 注册窗口

用户输入账号和密码登录成功后就会进入菜单界面（如图 5.3 所示），在菜单界面有四个选项，分别是：网络聊天、设置参数、每日资讯、虚拟伙伴。用户点击对应按钮就会跳转到对应界面。

图 5.3 系统主界面

5.2.2 群聊模块运行测试

同时开四个终端，启动四个客户端，登录对应账号，用户名分别是“刘培富”、“张毅杰”、“王俊翔”、“刘飞明”。界面左上角会显示用户名。先由刘培富发消息，发送完毕后四个客户端会同时显示刘培富发的消息，刘培富的客户端中他自己的消息是在右侧显示的，而在其他的客户端中刘培富的消息是在左侧显示的。然后张毅杰也发一个消息，所有客户端都显示正常。如图 5.4 所示：

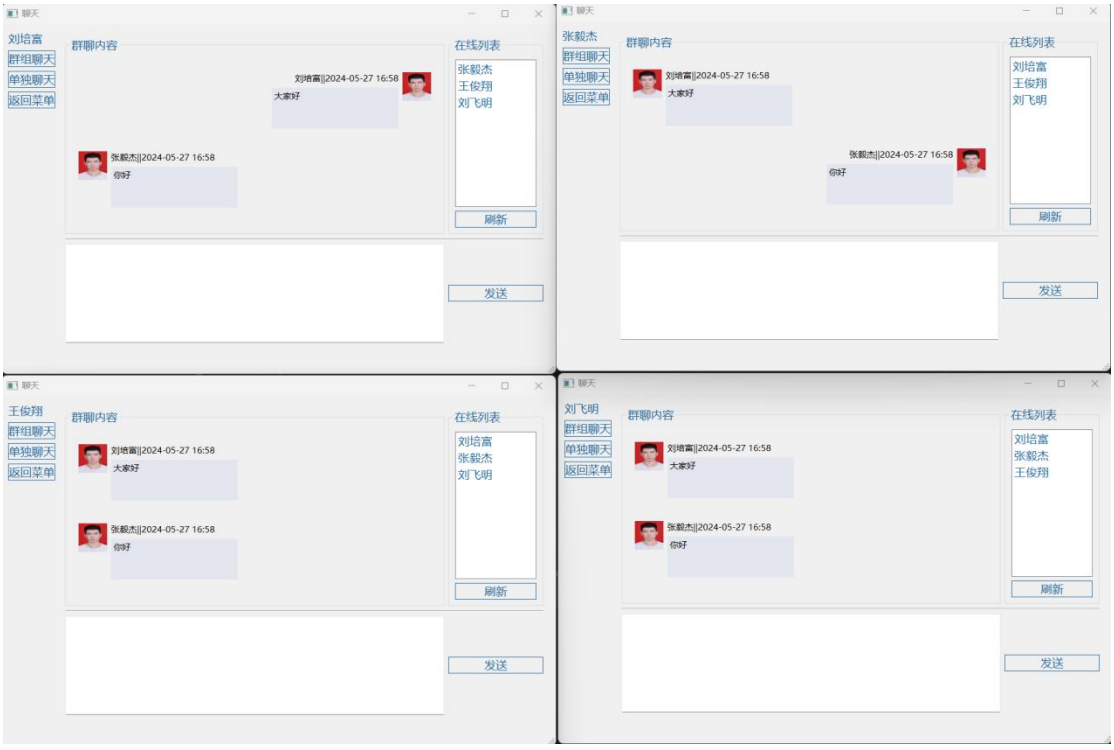


图 5.4 四个用户正在群聊

5.2.3 单聊模块运行测试

四个用户都点击“单独聊天”按钮进入单聊界面，先点击“刷新”按钮刷新一下在线用户列表。然后刘培富点击在线列表中的张毅杰和他私聊，刘培富给张毅杰发送消息后，只有刘培富和张毅杰这两个用户显示聊天记录，其他两个用户不显示。然后张毅杰选择刘培富和他私聊，张毅杰给刘培富发消息后也是只有他们两个能收到消息，其他用户不显示消息。如图 5.5 所示：

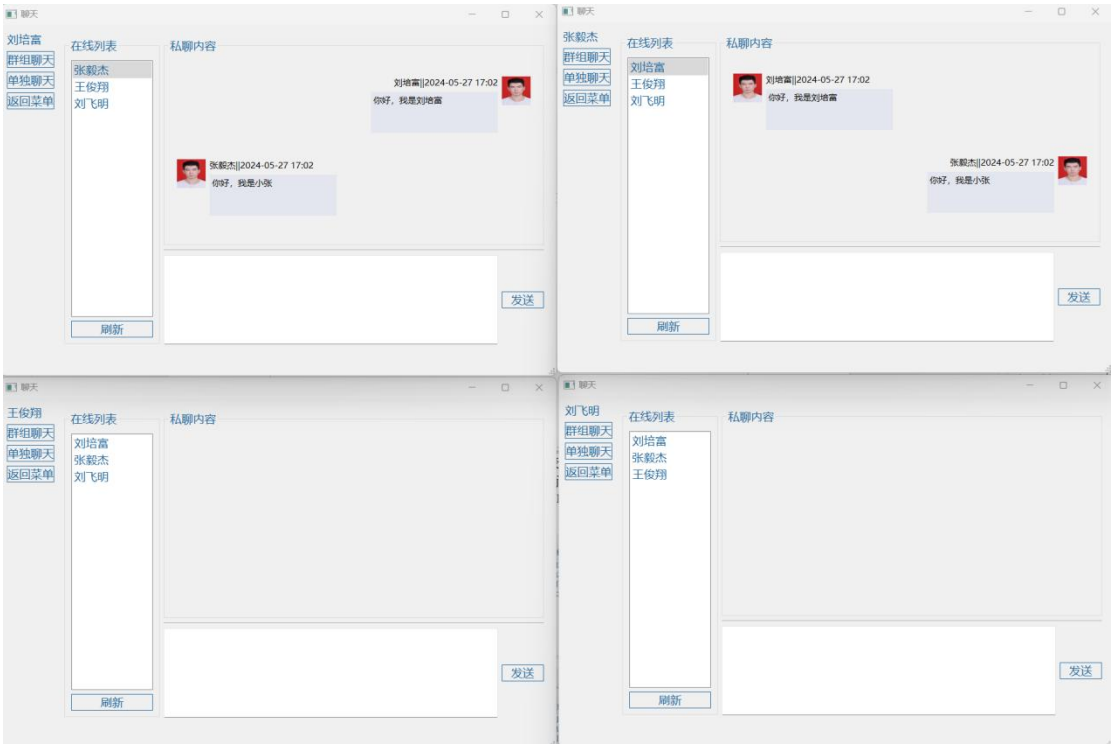
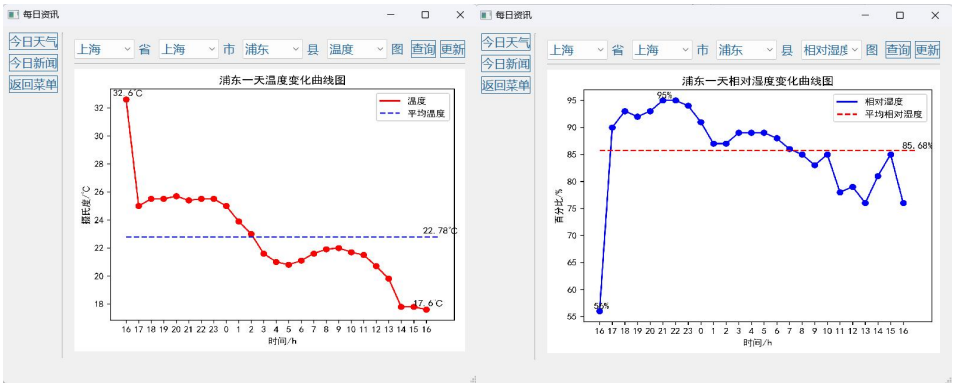


图 5.5 两个用户正在私聊

5.2.4 每日天气模块运行测试

用户点击“今日天气”按钮进入天气查询界面，点击“更新”按钮后程序会开始爬取指定省份（这次选择的是上海）的所有地区的今日天气，爬取完成后用户可以选择该省的某个地区，然后点击“查询”按钮即可查看该地区今天的天气情况。这边查询的是浦东新区的今日 24 小时天气分析，如图 5.6 所示：



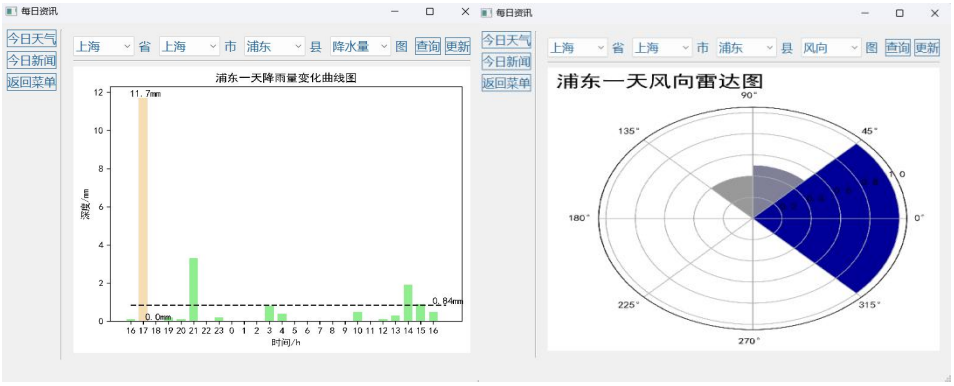


图 5.6 用户查看浦东新区今日天气分析

5.2.5 每日新闻模块运行测试

用户点击“今日新闻”按钮进入新闻查询界面，点击“更新”按钮后程序会开始爬取澎湃新闻部分栏目的今日新闻，爬取完成后用户可以选择某个栏目的某个标题，然后点击“查询”按钮即可查看该栏目今天的新闻。界面上边是新闻标题，下边是新闻内容，用户可滑动鼠标滚轮来查看新闻的上下文。这边查看的是科技栏目的今日新闻，如图 5.7 所示：

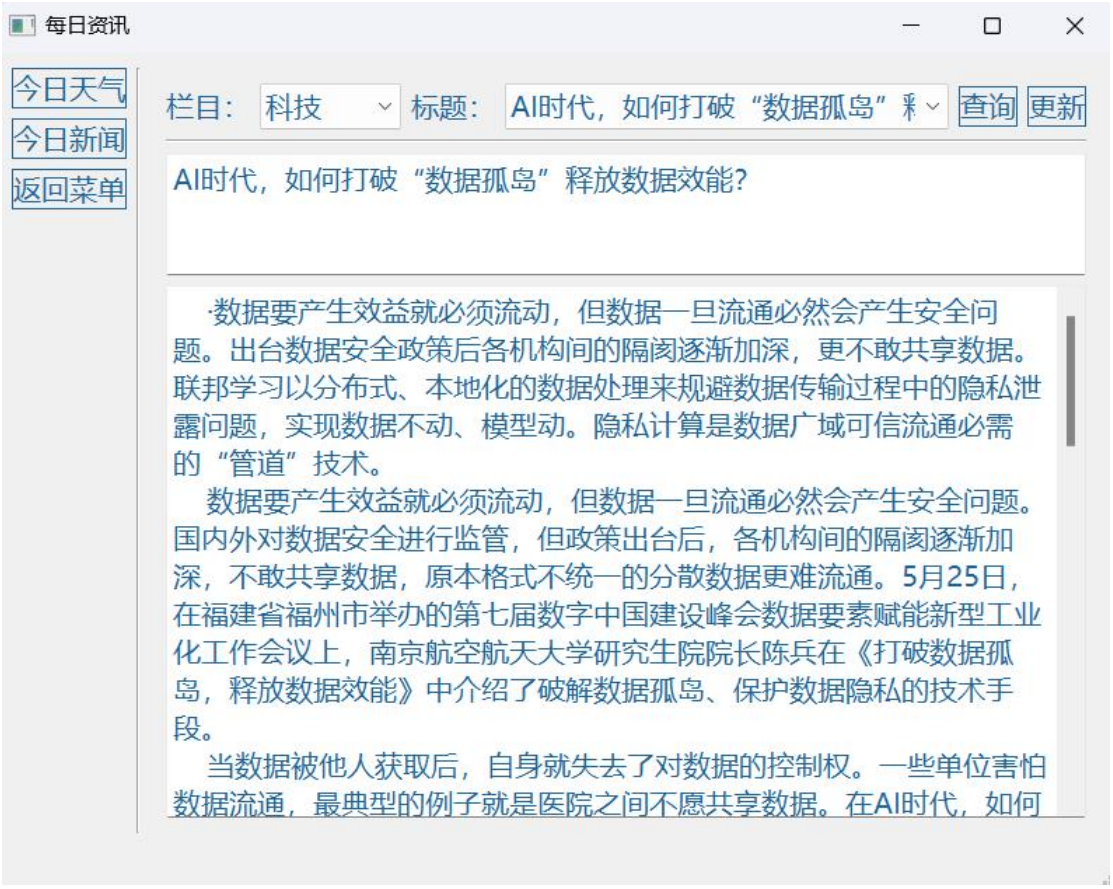


图 5.7 用户查看科技栏目今日新闻

5.2.6 虚拟伙伴模块运行测试

用户点击“虚拟伙伴”按钮进入虚拟伙伴界面，本系统目前共有十个虚拟伙伴，都是原神中的角色，分别为派蒙、纳西妲、可莉、刻晴、八重神子、雷电将军、巴巴托斯、神里凌华、肖宫、钟离。本次选的是和刻晴聊天。用户向刻晴发送“你好，你是谁？”的消息后刻晴会用回答介绍她自己的文本并会用语音输出。如图 5.8 所示：

图 5.8 用户正在和虚拟伙伴刻晴聊天

5.3 实验结果分析

5.3.1 每日天气模块各爬虫速度对比

在每日天气模块中，我分别写了同步爬虫、多线程爬虫和异步爬虫，并把它们都运行了一遍。每次爬取都是爬取 2564 个链接，其中同步爬虫花费时间为 1872.5340263843536 秒，多线程爬虫花费时间为 381.6039741039276 秒，异步爬虫花费时间为 301.370320558548 秒。可以看出多线程相比于同步所花费的时间大大减少，而异步相比于多线程所花费的时间减少的就比较少了。具体如图 5.9 所示：

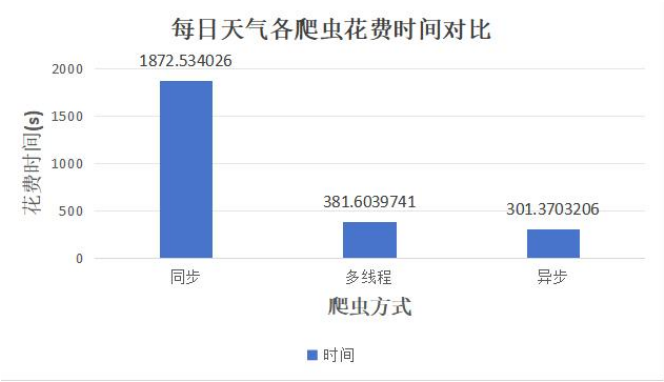


图 5.9 每日天气各爬虫花费时间对比

5.3.2 每日新闻模块各爬虫速度对比

在每日新闻模块中，我分别写了同步爬虫、多线程爬虫和异步爬虫，并把它们都运行了一遍。每次爬取都是爬取 159 个链接，其中同步爬虫花费时间为 47.80669569969177 秒，多线程爬虫花费时间为 7.524033308029175 秒，异步爬虫花费时间为 1.9563357830047607 秒。可以看出多线程相比于同步所花费的时间大大减少，异步相比于多线程所花费的时间减少幅度比较小。异步爬虫是最快的。具体如图 5.10 所示：

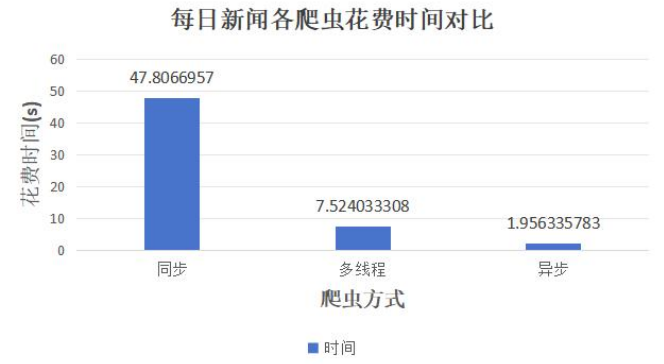


图 5.10 每日新闻各爬虫花费时间对比

6. 调试和改进

在本次项目开发过程中我遇到了很多问题，但都一一解决了，具体如下：

1.程序引用库出错

问题：我将客户端的代码放在了 code 文件夹下，界面代码放在了 ui 文件夹下，终端下运行 code 文件夹下的代码，调用 ui 文件夹下的类，会出现找不到模块的报错。

解决方法：我在项目根目录下新建一个 main.py 文件，将客户端代码都放在里面，再调用 ui 文件夹下的类就可以调用了。

2.Qt 界面崩溃

问题：我爬取完网页后直接在子线程下更新 Qt 界面的数据，程序会崩溃闪退。

解决方法：自定义一个信号，当子线程爬取完网页后给主线程发信号，主线程来更新 Qt 界面，这样程序就不会崩溃了。

3.绘制不同地区的曲线分析图重合

问题：使用 matplotlib.pyplot 库绘制完第一个地区的温度曲线图后再画第二个地区的会和第一个地区的重合。

解决方法：因为我每次调用函数时都是在同一个 plt.figure() 下进行绘图，所以第二次调用函数时会将图形绘制在已存在的图形上，导致重叠。为了解决这个问题，我在每次调用函数时创建一个新的图形对象，通过使用 plt.close() 函数，并给不同的图形指定不同的编号。这样绘图就不会重叠了。

4. 多线程中调用 matplotlib 绘图报错

问题：在运行时报出 UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail. plt.figure(nam=file_name)

解决方法：多线程过程中若果调用 matplotlib 会导致绘图顺序打乱，甚至出现图形不对，所以 matplotlib 干脆就禁止了在多线程中的应用。所以我将 matplotlib 放到后台取渲染，这样就不会和 pyqt5 的多线程冲突了。在多线程代码页顶部加入两行代码即可：

```
import matplotlib
matplotlib.use('agg')
```

5. 优化绘图逻辑

问题：起初，我想用多线程在爬虫爬取网页数据的时候顺便一次性把所有上海地

区的温度曲线图全部画完存下来，但困难重重，而且很耗费时间。

改进：我在爬虫爬取网页数据的时候只写入数据就行了，先不绘图。后面等查询哪个城市的就用哪个城市的数据现场画就行了。这样可以将爬虫处理的速度提高75%。

6. 单聊时其他用户也会收到消息的 bug

问题：单聊时会向指定用户发送一次消息，但会向其他用户发送两次消息。

改进：在客户端初始化的时候先向服务器发送一次消息，服务器收到后会更新在线用户列表，但不会给客户端发消息。当客户端点击发送或刷新按钮后会给服务器发送消息，服务器收到消息后会根据模式选择给客户端广播还是给指定客户端发。

7. python 协程的 bug

问题：在使用 python 协程中的 asyncio.run()运行会出现报错 RuntimeError: Event loop is closed。

解决：将 asyncio.run(main())替换为下面两行

```
loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

8. 优化每日天气爬虫

问题：起初，我企图使用异步爬虫一次性把全国所有地区的每日天气数据都爬下来，但是每次爬到三分之二的时候 ip 都会被封，我买的代理池也不够用了。

改进：为了节省经费，我选择每次只爬取用户所选省份的所有地区的每日天气数据，这样速度也挺快的，只是多了一步更新指定省份天气数据的步骤。

9. 修改 QScrollArea 背景色透明时子控件也会透明

问题：我想在虚拟伙伴聊天界面插入每个角色对应的专属背景图片，只有 Qwidget 中的 page 能设置背景图片，所以我先在 page 中设置好背景图片，然后将 QScrollArea 的背景色设置为透明，但是我发现在其中的聊天气泡也变成透明的了，字体根本看不清。

解决：在 QScrollArea 或者父控件中设置：

```
QScrollArea{background-color:transparent;}
```

在 scrollAreaWidgetContents 控件或者父控件中设置：

```
#scrollAreaContents
{background: transparent; }
```

7. 心得和结论

7.1结论和体会

本系统实现了三大模块：网络聊天模块、每日资讯模块和虚拟偶像模块。其中，网络聊天模块实现了群聊功能和单聊功能；每日资讯模块实现了每日天气爬虫和每日新闻爬虫；虚拟偶像模块实现了可以和十个不同的原神 AI 角色（派蒙、纳西妲、可莉、刻晴、八重神子、雷电将军、巴巴托斯、神里凌华、肖宫、钟离）聊天并且可以听到他们的声音。本系统带有 UI 界面，在网络聊天和虚拟偶像聊天时消息会以聊天泡的形式展示在聊天框中，看起来清晰明了，但我把模型部署在服务器上了，所以每次使用虚拟伙伴模块时都要开一下服务器，比较麻烦。在每日天气爬虫中使用了异步爬虫，每日新闻爬虫中使用了线程池加异步爬虫，速度非常快，但一次不能爬太多，不然会被封 IP。

在初期我先系统的学习了课设可能要用到的东西，我先跟着 B 站 UP 主“白月黑羽”学习了 python 的 socket 编程，发现其中涉及到多线程，于是又去学习了 python 的多线程。然后我尝试着实现纯后端的多人聊天室，发现消息格式不统一很麻烦，于是我又去学习了 python 的 json 格式，使得每次消息都能按照规范发送，这样就不会出错了，也方便处理。在实现了纯后端的多人聊天室后，我觉得只有后端肯定是不够大，为了能做出 UI 界面，我又跟着 B 站 UP 主“白月黑羽”系统学习了 pyqt，其实我寒假接触过 pyqt，但是不系统，就导致我做的界面很不规范，界面会出很多 bug，也很难看。在经过这次系统的学习后我也知道了做界面的规范步骤，我先使用 qt desinger 来把界面拖好并把布局从内到外设置好，这样界面就可以自适应窗口，看起来比较舒服，然后将 ui 文件转换为 python 代码就能在程序中调用了。因为不同 py 文件之间调用需要 python 中的类，python 类我有点忘了，于是我又跟着 B 站 UP 主“GenJi 是真想教会你”系统速通了一下 python，这个课是个合集，里面除了 python 教学还要爬虫教学，我挺感兴趣的，于是顺便把爬虫也学了。因此是爬虫的初学者，我就爬取一些可以直接静态爬取的网页，但是爬取的数据量比较大，单线程速度很慢，于是我又去学习了多线程和异步爬虫，速度提升很快。但太快也不行，IP 会被封，于是我又去买了代理池，里面有五个高速代理 IP，但还是不够用，代理池每天要花七十多块钱，考虑到经费问题，我放弃了代理池这条思路，而是优化爬虫逻辑，降低每次爬取的数据量，这样 IP 就不会被封了。利用多线程和异步爬虫，我实现了爬取中国天气网和澎湃新闻网当日数据，但只有数据还不够，还要利用这些数据，于是我又去学习了 matplotlib，用它来进行每日天气的数据分析。至此，

计划的功能已经做了一半了。

在中期，我在原来的基础上开始搞虚拟伙伴部分，之前打过有关 LLM 越狱攻击与防御的比赛，对象是智谱 AI，对它比较熟悉，所以本系统大语言模型用的是智谱 AI，语音合成模型用的是目前最新的 GPT-SoVITS，我先跟着 B 站 UP“白菜工厂 1145 号员工”的视频教程用作者提供的整合包走了一遍全流程，熟悉了这个模型后我开始试着用代码来使用它，但是因为环境很难配，我的电脑显存也不太够，于是我就在 AutoDL 上租了个 4090，在上面配好环境后我就用 API 技术给服务器发请求，服务器推理完成后就会将 wav 文件返回给我，我就可以直接用电脑播放了。单独测试没问题后我就设置把虚拟伙伴加入到系统中，但是在加入的过程中我发现原来的 UI 所有功能都挤在一个界面太丑了，也不方便未来的拓展。于是，在强迫症的驱使下，我直接废弃掉 1.0 版本，新建了一个 2.0 版本的项目，从头开始设计 UI 界面，并将后端代码也进行了重构，各个模块的代码都规范书写，这样方便我使用，将在 1.0 版本已经实现的功能在 2.0 重新实现。

在后期，因为答辩时间稍微延期了一点，于是我就将虚拟伙伴拓展了一下，由原来的一个角色增加到十个角色，好在角色的声线模型网上有开源的，可以直接用。但是虚拟伙伴的 UI 要加个角色按钮，还要为每个角色单独设置一个聊天框，这个工作不难，但十分繁琐，在 qt desinger 上设计好前端后我还要为每个角色单独编写后端逻辑和聊天气泡的代码，写到最后我都快神志不清了，期间还遇到了一些前端显示的 bug，不过好在一一解决了，做这个系统给我的感觉就是单独开发后端很简单，单独开发前端也很简单，但是把它俩合到一起就很困难，就像我做比赛最难的部分就是上位机和下位机之间的通信一样。

在开发本系统的过程中学到新知识让我很兴奋，遇到这种各样的 bug 让我很折磨，UI 设计让我很疲惫，前后端通信让我很苦恼。通过这次课设，我系统的学到了很多新的知识，每一次挫折都让我有了新的成长。最终，完整地开发出一个电脑软件让我很有成就感，也极大地增强了我的自信，这次课设，收获满满！

7.2进一步改进方向

本次系统依然有很多不足，如果时间充裕，本系统在原来的基础上还可以作如下改进：

1.开发网页版：本系统原来是软件版本，如果要使用需要下载到电脑上，十分不方便，在未来本系统可以移植到网页上，方便用户随时打开使用。

2.聊天加入文件系统：本系统目前只实现了文字聊天功能，未来可以加入文件系统，用户可以发送图片、视频等文件，同时也可以进行语音通话，聊天记录也会被保存下来。

主要参考文献

[1]刘畅,谷生然.人工智能时代人的发展境遇审思[J].理论导刊,2024(04):72-78.

[2] SoundHound AI Voice Assistant with Integrated ChatGPT Rolls in Stellantis DS Automobiles[J]. Manufacturing Close - Up,2024.

[3] AlMugeiren M O ,Assari S A ,Alshehri A K , et al. Placement of immediate dental implants in extraction sockets exhibiting the apical pathosis. A meta-analysis.[J]. Journal of oral biology and craniofacial research,2024,14(3).

[4]徐圣方,王金阳.Python 爬虫获取豆瓣观众影评数据及可视化分析[J].网络安全技术与应用,2024(04):59-62.

[5]郑晓波,吴文超,贾守波,等.基于多线程消息框架的相控阵天线测试系统优化设计[J].火控雷达技术,2024,53(01):116-122.DOI:10.19472/j.cnki.1008-8652.2024.01.020.

[6]李阳,范伊红,李彦蓉,等.基于 aiohttp 的异步爬虫的设计与实现[J].无线互联科技,2021,18(17):56-57.

[7]畅巨峥,李飏,崔粉娥,等.Python 在秦皇岛大雾气候特征统计分析中的应用[J].气象水文海洋仪器,2024,41(02):122-125.DOI:10.19441/j.cnki.issn1006-009x.2024.02.021.

[8]褚乐阳,王浩,陈向东.面向大语言模型的青少年人工智能教育[J].中国电化教育,2024,(04):32-44.

[9]王嘉文,高定国,尼琼,等.基于 VITS 模型的藏语康巴方言语音合成研究[J].电脑知识与技术,2024,20(04):8-10+15.DOI:10.14004/j.cnki.ckt.2024.0226.

[10]吴毅杰,张志明.C/S 与 B/S 的比较及其数据库访问技术[J].舰船电子工程,2003,(02):32-35.