北京郵電大學

# 程序文档

项目：领域特定脚本语言的解释器的设计与实现

| | |
|---|---|
| 班　　级： | **2019211308** |
| 学　　号： | **2019211509** |
| 姓　　名： | 庞仕泽 |
| 学　　院： | 计算机学院 |

**2021** 年 **12** 月 **29** 日

# 目录

# 1 软件开发任务

领域特定语言（Domain Specific Language，DSL）可以提供一种相对简单的文法，用于特定领域的业务流程定制。本作业要求定义一个领域特定脚本语言，这个语言能够描述在线客服机器人（机器人客服是目前提升客服效率的重要技术，在银行、通信和商务等领域的复杂信息系统中有广泛的应用）的自动应答逻辑，并设计实现一个解释器解释执行这个脚本，可以根据用户的不同输入，根据脚本的逻辑设计给出相应的应答。

基本要求：

1. 脚本语言的语法可以自由定义，只要语义上满足描述客服机器人自动应答逻辑的要求。
2. 程序输入输出形式不限，可以简化为纯命令行界面。
3. 应该给出几种不同的脚本范例，对不同脚本范例解释器执行之后会有不同的行为表现。。

# 2 总体方案设计

本篇将对系统的整体架构进行说明，本系统名称为 chat。

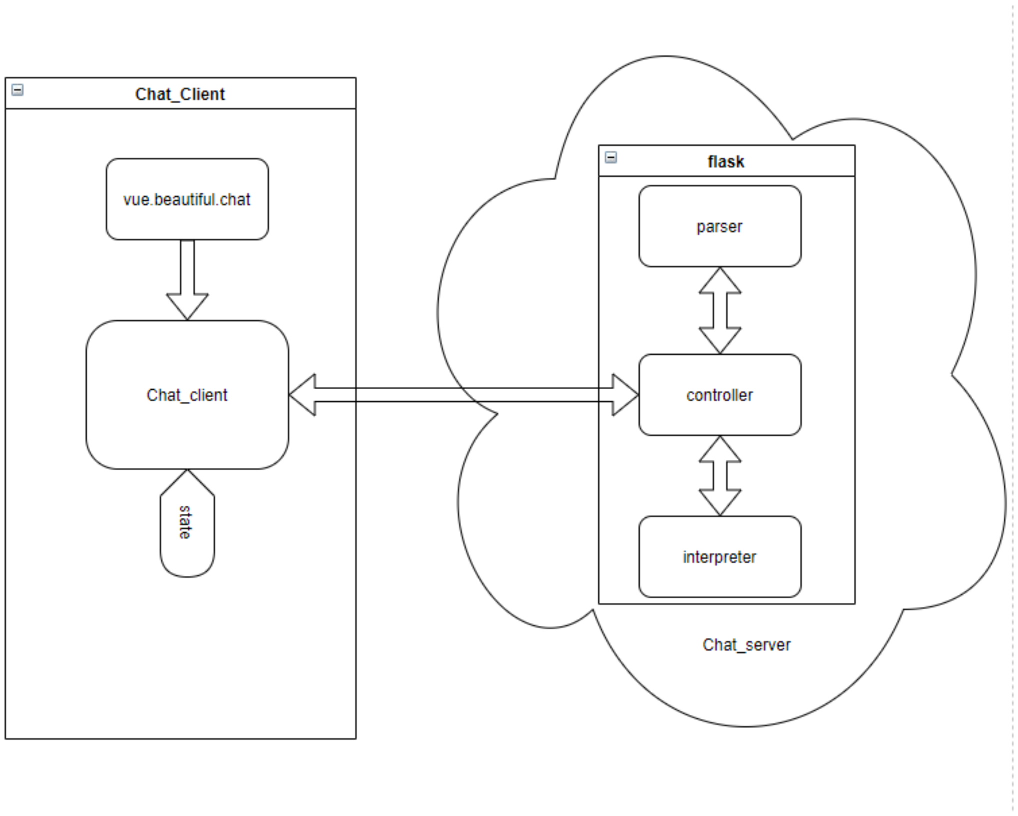
## 2.1 开发环境

- Microsoft Windows 版本 20H2（操作系统内部版本 19042.985）
- Python 3.10.0
- Flask 2.0.2
- CORS 1.0.2
- yarn 1.22.15
- @vue/cli 4.5.15
- axios 0.24.0
- vue3-beautiful-chat
- pytest 6.2.5

## 2.2 总体结构

以下是 chat 的结构：

```
├──chat_service\bot_service
│  ├───controller.py              #  主程序
│  ├──classes.py              #  数据结构
│  ├──parser                       # 分析程序
│  ├──interpreter                     #  bot 运行程序
│  ├──script                      #  脚本文件
│  │    └──script                 #脚本 1
│  └──test_main.bat             #  自动测试脚本，其他 test 开头为测试桩
├─chat_client
│  ├──public                       # vue 静态资源
│  ├──src                           # vue 程序
│  │     ├──assets                    # vue 资源
│  │     ├──App.vue                     # vue 主程序
│  │     ├──main.js                    # 页面主程序
│  └──......                       #  vue 配置文件
```



项目框架

## 2.3 系统架构

本系统基于前后端分离的交互模式，使用 flask 框架与 VueJS 框架开发，以下是对系统架构及子架构的综述。

### 2.3.1 前端交互架构

#### 整体工作流程

前端通过直接引入 vue.beautiful.chat 插件，调用聊天窗口等插件实现与用户的可视化交互。整个沟通流程为一个状态机，在前端存储当前用户所处的状态 state，前端将用户输入以及当前所处状态 post 到后端中，经由后端处理后将用户的下一个状态以及机器人的回复传回，调用 vue.beautiful.chat 插件的接口模拟客服机器人输出回复并更新用户的状态。

#### 前端逻辑

前端整体调用 vue.beautiful.chat 插件以及 axios 通信架构，该插件暴露接口 onMessageWasSent 中可以读取用户输入信息。调用 axios 将当前用户 state 以及用户输入 request_c 组成 json 发送给后端，同步接受后端响应后，对后端传入信息今昔解耦，通过 state_id 更新当前用户状态，并将响应中的 reply 组成一条机器人的输出消息并调用接口输出到聊天框中。

vue.beautiful.chat 插件的聊天消息本质为一个 list，其元素组成为

```
{
        type : 'text',//信息格式
        author : `bot`, //信息发送主体
        data : {
          text: replyJson.reply    //信息
          }
 }
```

通过将一个元素 push 到 list 中便可以快速的实现机器人方消息的发出

#### Axios

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中，拥有功能：

- 从浏览器中创建 XMLHttpRequests
- 从 node.js 创建 http 请求
- 支持 Promise API
- 拦截请求和响应
- 转换请求数据和响应数据
- 取消请求
- 自动转换 JSON 数据
- 客户端支持防御 XSRF

本程序调用 axios 的 api 实现前端向后端发送 post 并接收消息的功能，注意发送的请求为跨域请求，后端需要对跨域请求进行处理

#### vue.beautiful.chat

参见：https://github.com/mattmezza/vue-beautiful-chat

### 2.3.2 后端架构

#### 后端工作流程

后端服务器部署在 http://127.0.0.1:5000/bot 上，此外还可以直接打开 http://127.0.0.1:5000 的 hello_world 程序测试 server 是否运行并且成功链接。后端服务器在启动时将目标脚本文件的地址传入解析器并得到解析后能被程序读取的脚本信息，后端服务器收到前端传入的 json 后，flask 会自动解构包含的 json 信息，将脚本信息与 json 信息传入 interpreter 中，运行得到客户的下一个状态以及机器人需要输出的信息。

如当前用户处于状态 1，输入了”back“信息，interpreter 运行后得出用户将跳转到状态 0，并且机器人输出用户跳转到状态 0 时需要输出的信息。

由于前端传入为跨域请求，我们需要调用 CORS 的 flask 架构的 cor 处理前端的跨域请求。

#### Flask

Flask 作为现代 python 后端常见的架构，“flask”并不代表整个应用只能塞在一个 Python 文件内，当然塞在单一文件内也没有问题。 “flask”也不代表 Flask 功能不强。微框架中的“flask”字表示 Flask 的目标是保持核心简单而又可扩展。 Flask 不会替用户做出许多决定，比如选用何种数据库。类似的决定，如使用何种模板引擎，是非常容易改变的。 Flask 可以变成任何想要的东西，一切恰到好处，由用户做主。

缺省情况下，Flask 不包含数据库抽象层、表单验证或者其他已有的库可以处理的东西。然而， Flask 通过扩展为应用添加这些功能，就如同这些功能是 Flask 原生的一样。大量的扩展用以支持数据库整合、表单验证、上传处理和各种开放验证等等，Flask 已经为满足各种生产需要做出了充足的准备。

本程序基于 flask 架构实现了部署、监听运行、并发等后端必须的性质，从而可以专心于工作流程的构建与 parser 和 interpreter 的编写，节省了编程时间。

### 2.3.3 parser/interpreter

#### parser

本程序的脚本基于状态机编写，所以 parser 总体是对类状态机信息的读取。在主程序传

入脚本文件路径后，parser 对其进行按行读取并根据该行的第一个字符串构造各个状态的详细信息。基于按行读取的逻辑，parser 有正常模式和 reply 模式两种模式，reply 模式是为当前状态机的 reply 信息超过一行设置，可以实现多行 reply 的读入并忽略关键字的功能，直到碰到 end reply 转换成正常模式。在读取完一个状态（遇到关键字 state）后，该状态会停止更新并插入到脚本表中，最终 parser 会返回一个拥有脚本中全部状态的脚本表，该脚本表也会有一个 default 字符串用于用户输入无法识别的语句时的机器人回复。脚本定义及状态机的数据结构参见 2.4 节

**interpreter**

在主程序传入解析好的脚本表，用户的状态以及用户的输入后，interpreter 会根据用户状态与用户状态于脚本表中比对，如果比对成功则返回下一个状态以及该状态的回复，如果比对失败（即用户输入无法识别）则返回原有状态以及 default 字符串的回复，注意状态机的 id 与脚本载入顺序无关，仅依赖于脚本中 state 关键字后的数字。此外，interpreter 也会识别非法输入 id 并还原状态为第 0 号状态，所以脚本推荐从 0 开始输入。

## 2.4 脚本

### 2.4.1 DSL 定义

这里先给出 dls 样例，该样例被部署在机器人上：

```
state 0

reply: choose the game store you want to know

steam

epic

origin

uplay

end reply

event: steam goto: 1

event: epic goto: 4

event: origin goto: 6
```

event: uplay goto: 7

state 1

reply: Steam is a video game digital distribution service by Valve.

It was launched as a standalone software client in September 2003 as a way for Valve to provide automatic updates for their games, and expanded to include games from third-party publishers.

Steam has also expanded into an online web-based and mobile digital storefront. Steam offers digital rights management (DRM), server hosting, video streaming, and social networking services. It also provides the user with installation and automatic updating of games, and community features such as friends lists and groups, cloud storage, and in-game voice and chat functionality.

you can enter:

"buy" to jump to the store,

"search" to jump to the game information search engine,

"back" to return to the menu,

"" shouldn't be entered

end reply

event: buy goto 2

event: search goto 3

event: back goto 0

state 2

reply: https://store.steampowered.com

you can search the game you want and buy it in this website

enter: "back" to return to the menu

end reply

event: back goto 0

state 3

reply: https://steamdb.info

this website is the database of steam which you can search history price and more information of games

you can search the information of the game you want in this website

enter: "back" to return to the menu

end reply

event: back goto 0

state 4

reply: The Epic Games Store is a digital video game storefront for Microsoft Windows and macOS, operated by Epic Games.

It launched in December 2018 as both a website and a standalone launcher, of which the latter is required to download and play games.

The storefront provides a basic catalog, friends list management, matchmaking, and other features.

Epic Games has further plans to expand the feature set of the store front but it does not plan to add as many features as other digital distribution platforms, such as discussion boards or user reviews, instead using existing social media platforms to support these.

the website is: https://www.epicgames.com/store/zh-CN.

enter: "free" to jump to the weekly free game of epic,

enter: "back" to return to the menu,

end reply

event: free goto 5

event: back goto 0

state 5

reply: https://www.epicgames.com/store/zh-CN/free-games

this website is the free game that epic will send different game which used to be charged for free

you must buy the game(cost 0) so that you can get the present

enter: "back" to return to the menu

end reply

event: back goto 0

state 6

reply: Origin is a digital distribution platform developed by Electronic Arts for purchasing and playing video games.

The platform's software client is available for personal computer and mobile platforms.

Origin contains social features such as profile management, networking with friends with chat and direct game joining along with an in-game overlay.

the website is: https://www.origin.com/hkg/zh-tw/store

enter: "back" to return to the menu

end reply

event: back goto 0

state 7

reply: Ubisoft Connect (formerly Uplay) is a digital distribution, digital rights management, multiplayer and communications service developed by Ubisoft to provide an experience similar to the achievements/trophies offered by various other game companies.

The service is provided across various platforms. Ubisoft Connect is used exclusively by first-party Ubisoft games, and although some third-party ones are sold through the Ubisoft store, they do not use the Ubisoft Connect platform.

the website is: https://store.ubi.com/cn/home

enter: "back" to return to the menu

end reply

event: back goto 0

default: I can't understand.Please ask in form.

本机器人声明脚本的语法特征如下：

·新状态必须以 state 开头，state 后数字为当前状态的 id

·关键字必须定义在行头，除 reply 外一行只能有一个语句

本机器人声明脚本拥有以下语句：

**state id**

　　表示新的状态，id 为该状态的唯一标识（id 不可重复）

**event text goto id**

　　表示该状态下遇到 **"text "**的请求，状态机会跳转到 **id** 状态，注意 text 不支持空格

**default:**

　　表示碰到无法解析的请求是的统一回复,一个脚本只能有一个 default(当有多个 default 时会选择第一个作为统一回复的文本）

**reply:**

---

**end reply**

　　当遇到 reply： 关键字后，解析器将进入 reply 模式，其后所有的文字将被记录在该状态的机器人回复中，并忽略所有关键字直到遇到 end reply，且 end reply 为某一行开头时停止读入，进入正常读取关键字模式。

### 2.4.2 脚本存储数据结构

本程序使用一个类来存储状态机的结构：

```
class State:
    """ this class using to store the states read from the script

    This class is used by any module who need to use the content
    from the script.It will help to store and read different data
    directly.

    Attributes:
        reply:the text the bot need to reply to customers
        id:the id of the state.the id of state 0 is 0
        event:a dict to store events.It's struct is {test_string :
state_id }
        test_string: the test of customer's request
        state_id: next state jump to
    """
    def __init__(self, id):
        """Inits StateClass with id"""
        self.id = id
        self.event = dict()
        self.reply = "please enter reply in script"

    def add(self,key,id):
        """Add test_string : state_id to the event dict"""
        self.event[key] = id
```
其中 id 存储当前状态机的 id

reply 存储跳转到当前状态机时机器人的回复

event 为一个字典，存储不同请求下跳转的目标状态

# 3 程序实现

本程序基于 google 开源项目风格 python 规范编写

网页地址：https://zh-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/python_style_rules

## 3.1 后端实现

本程序的后端存储在工程文件中 chat_service\bot_service 中

### 3.1.1 controller

本程序的主程序如下：

```
"""the is the main app of the bot sever in flask structure

Recieve HTTP get request and analyse it.
Return the next state and the reply of the bot

Using http://127.0.0.1:5000 as a "hello world" to check server running
and connect
Using http://127.0.0.1:5000/bot as bot server and must post/get a json
with
```

```
{state_id : , request_c :}
"""
from flask import Flask
from flask import request
from flask_cors import CORS


import json
path = "script\\script.txt"
app = Flask(__name__)
app.debug = True

from parser import load_script
from interpreter import generate
#main bot server,cors:Cross-Origin Resource Sharing
CORS(app, resources=r'/*')
@app.route("/bot",methods=['POST', 'GET'])
def server():
    with open(path, 'r', encoding='utf8') as f:
        script = load_script(f)
    req = request.json
    fo = open("foo1.txt", "w")
    fo.write(str( req['state_id'])+req['request_c'])
    answerlist = generate(script, req['state_id'], req['request_c'])
    answerdict = {}
    answerdict['state_id'] = answerlist[0]
    answerdict['reply'] = answerlist[1]
    return answerdict
#test server running and connect
@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

### 3.1.2 classes

脚本状态类声明

```
"""the module to store thing will be global used

Powered by Python,this module defines a class to store the script
in global enviroment. This module shouldn't import any thing from
program to avoid circular import

Typical usage example:
```

```
state = State(0)
"""
class State:
    """ this class using to store the states read from the script

    This class is used by any module who need to use the content
    from the script.It will help to store and read different data
    directly.

    Attributes:
        reply:the text the bot need to reply to customers
        id:the id of the state.the id of state 0 is 0
        event:a dict to store events.It's struct is {test_string :
state_id }
        test_string: the test of customer's request
        state_id: next state jump to
    """
    def __init__(self, id):
        """Inits StateClass with id"""
        self.id = id
        self.event = dict()
        self.reply = "please enter reply in script"

    def add(self,key,id):
        """Add test_string : state_id to the event dict"""
        self.event[key] = id
```

### 3.1.3 parser

脚本解析器

```
"""the module to parser the script.txt to defined struct

Recieve the file stream and load data from file
parser the data to the struct bot_interpreter can read

Typical usage example:
script = load_script(0)
"""



from typing import IO
from classes import State
```

```
def analyze(lines) -> list:

    """ Parsing the script.txt, return formatted script structure

    First, splint the data read in as lines. Read the head to decide
which
    type of the lines stand for.Because of It will check the test if in
reply,
    so first check if it's end reply,then will check replyFlag = True,
    which means in reply mode,it will read until the end reply and only
add text.
    Then, line title analysis and add the information of different
situations to the state structure.

    Args:
        lines (list[str]): whole content form script text

    Returns:
        list[State,str]: formatted script Classes and default string
(list)
    """

    result = []
    replyFlag = False
    defaultFlag = False
    tmp = State(-1)
    for line in lines:
        if line[0] == "\n":
            continue
        line_element = line.split()
        if line_element[0] == "end" :                    #check the end
before check replyFlag
            if line_element[1] == "reply":
                replyFlag = False
        if replyFlag:                                   #in reply mode
            tmp.reply = tmp.reply + line
        else :
            if line_element[0] == "state" :
                result.append(tmp)
                tmp = State(int(line_element[1]))
            elif line_element[0] == "reply:" :
                replyFlag = True
                text_tmp = line.replace("reply: ","")
                tmp.reply = text_tmp
            elif line_element[0] == "event:" :
                tmp.add(line_element[1],int(line_element[3]))
            elif line_element[0] == "default:" and defaultFlag ==
False :  #one default is enough
                result.append(tmp)
                text_tmp = line.replace("default: ","")
                result.append(text_tmp)
                defaultFlag = True
    return result

def load_script(f: IO) -> list:
    """parse a script file stream to program readable script

    Args:
        f (IO): fp

    Returns:
        list[State,str]: formatted script Classes and default string
(list)
    """
    lines = f.readlines()
    script = analyze(lines)

    if script is None:
        return None

    return script
```

## 3.1.4 interpreter

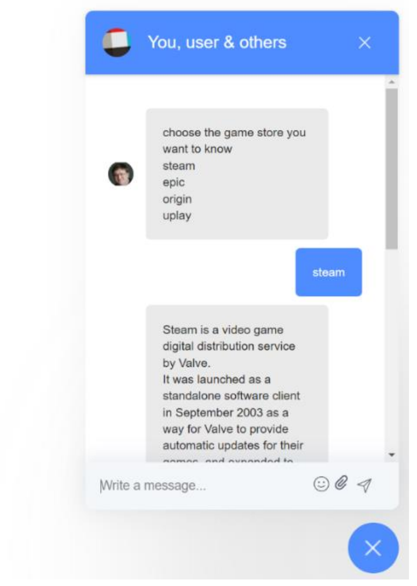
interpreter 程序

```
"""the module to analyse the request and build response with formatted
script

Recieve formatted script,client's state,client's request text
check the script and build response with next client'id and reply text

Typical usage example:
result = generate(script,0,"hello_world")
"""
from classes import State

def generate(script,state_id,request):
    """ build response with formatted script and return

    Check state_id and request in the Script table and the event of
aimed script
    If the aimed script has the key, get key's value and get
state[value]'s reply
    If not,return the default reply.
    Offered state_id check to insure the security.

    Args:
        script (list[State,str]): list of script States and a default
reply string
        state_id (int): received id from client which stands for
client's id currently
        request (str): received text from client which is client's entry

    Returns:
        list[int,str]: next state id client goto and bot's reply
    """
    answer = []
    i = 0
    nowid = 0
    for tmp in script:                          #the sequence of Script
is not the Script's id
        if isinstance(tmp,State):
            if tmp.id == state_id:
                nowid = i
                break
        i = i + 1
    if request in script[nowid].event:
        answer.append(script[nowid].event[request])
        toid = 0
        j = 0
        for tmp in script:
            if isinstance(tmp,State):
                if tmp.id == script[nowid].event[request]:
                    toid = j
                    break
            j = j + 1
        answer.append(script[toid].reply)
    else :
        if state_id > len(script):
            state_id = 0
        answer.append(state_id)
        for tmp in script:
            if isinstance(tmp,str):
                answer.append(tmp)
                break
    return answer
```

## 3.2 前端实现

本程序前端文件存储在工程文件中 chat_client 文件下，可视化效果如下：

聊天图标在页面右下角，点击后展开



choose the game store you want to know
steam
epic
origin
uplay

steam

Steam is a video game digital distribution service by Valve.
It was launched as a standalone software client in September 2003 as a way for Valve to provide automatic updates for their

聊天界面

### 3.2.1 main.js

前端项目入口文件

```
import { createApp } from 'vue'
import App from './App.vue'
import Chat from 'vue3-beautiful-chat'

createApp(App).use(Chat).mount('#app')
```

### 3.2.2 App.vue

前端网站主程序

```
<template>
  <div>
    <beautiful-chat
      :participants="participants"
      :titleImageUrl="titleImageUrl"
      :onMessageWasSent="onMessageWasSent"
      :messageList="messageList"
      :newMessagesCount="newMessagesCount"
      :isOpen="isChatOpen"
      :close="closeChat"
      :icons="icons"
      :open="openChat"
      :showEmoji="true"
      :showFile="true"
      :showEdition="true"
      :showDeletion="true"
      :deletionConfirmation="true"
      :showTypingIndicator="showTypingIndicator"
      :showLauncher="true"
      :showCloseButton="true"
      :colors="colors"
      :alwaysScrollToBottom="alwaysScrollToBottom"
      :disableUserListToggle="false"
      :messageStyling="messageStyling"
      @onType="handleOnType"
      @edit="editMessage" />
  </div>
</template>

<script>
```

```
import axios from 'axios';
export default {
  name: 'app',
  data() {
    return {
      state: 0,
      participants: [
        {
          id: 'user',
          name: 'user',
          imageUrl:
'https://gimg2.baidu.com/image_search/src=http%3A%2F%2Finews.gtimg.com%
2Fnewsapp_match%2F0%2F12115359878%2F0.jpg&refer=http%3A%2F%2Finews.gtim
g.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=jpeg?sec=1643375804&
t=5393a77e0e5dcb8f425838cd4795f076'
        },
        {
          id: 'bot',
          name: 'bot',
          imageUrl:
'https://gimg2.baidu.com/image_search/src=http%3A%2F%2Finews.gtimg.com%
2Fnewsapp_bt%2F0%2F14003895865%2F1000.jpg&refer=http%3A%2F%2Finews.gtim
g.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=jpeg?sec=1643375766&
t=5e0b4299d35465235b59bec03c1752d1'
        }
      ], // the list of all the participant of the conversation. `name`
is the user name, `id` is used to establish the author of a message,
`imageUrl` is supposed to be the user avatar.
      titleImageUrl: 'https://a.slack-edge.com/66f9/img/avatars-
teams/ava_0001-34.png',
      messageList: [
          { type: 'text', author: `bot`, data: { text: `choose the game
store you want to know
steam
epic
origin
uplay` } }
      ], // the list of the messages to show, can be paginated and
adjusted dynamically
      newMessagesCount: 0,
      isChatOpen: false, // to determine whether the chat window should
be open or closed
      showTypingIndicator: '', // when set to a value matching the
participant.id it shows the typing indicator for the specific user
```

```
      colors: {
        header: {
          bg: '#4e8cff',
          text: '#ffffff'
        },
        launcher: {
          bg: '#4e8cff'
        },
        messageList: {
          bg: '#ffffff'
        },
        sentMessage: {
          bg: '#4e8cff',
          text: '#ffffff'
        },
        receivedMessage: {
          bg: '#eaeaea',
          text: '#222222'
        },
        userInput: {
          bg: '#f4f7f9',
          text: '#565867'
        }
      }, // specifies the color scheme for the component
      alwaysScrollToBottom: false, // when set to true always scrolls
the chat to the bottom when new events are in (new message, user starts
typing...)
      messageStyling: true // enables *bold* /emph/ _underline_ and such
(more info at github.com/mattezza/msgdown)
    }
  },
  methods: {
    onMessageWasSent (message) {
      // called when the user sends a message
      var replyJson = {};                //store the response.data from
server
      this.messageList.push(message)       // client's entry
      axios.post('http://127.0.0.1:5000/bot', {
        state_id : this.state,
        request_c : message.data.text
        })
        .then((response) => {
            console.log(response);
            replyJson = response.data
            this.state = replyJson.state_id
            var bot_message = {
              type : 'text',
              author : `bot`,
              data : {
                text: replyJson.reply
                }
              };
          this.messageList.push(bot_message)        //bot's entry
          console.log(this.state);
        });
    },
    openChat () {
      // called when the user clicks on the fab button to open the chat
      this.isChatOpen = true
      this.newMessagesCount = 0
    },
    closeChat () {
      // called when the user clicks on the botton to close the chat
      this.isChatOpen = false
    },
    handleScrollToTop () {
      // called when the user scrolls message list to top
      // leverage pagination for loading another page of messages
    },
    handleOnType () {
      console.log('Emit typing event')
    },
    editMessage(message){
      const m = this.messageList.find(m=>m.id === message.id);
      m.isEdited = true;
      m.data.text = message.data.text;
    }
  }
}
</script>

<style>

</style>
```

# 4 测试、构建与运行

## 4.1 测试

本程序使用 pytest 作为测试框架。

pytest 是一个非常成熟的全功能的 Python 测试框架，主要特点有以下几点：

- 1、简单灵活，容易上手，文档丰富；
- 2、支持参数化，可以细粒度地控制要测试的测试用例；
- 3、能够支持简单的单元测试和复杂的功能测试，还可以用来做 selenium/appnium 等自动化测试、接口自动化测试（pytest+requests）；
- 4、pytest 具有很多第三方插件，并且可以自定义扩展，比较好用的如 pytest-selenium（集成 selenium）、pytest-html（完美 html 测试报告生成）、pytest-rerunfailures（失败 case 重复执行）、pytest-xdist（多 CPU 分发）等；
- 5、测试用例的 skip 和 xfail 处理；
- 6、可以很好的和 CI 工具结合，例如 jenkins

本程序编写了 test_parser 和 test_interpreter 作为 parser 和 interpreter 的单元测试，此处展示 test_parser：

```
import  pytest
from parser import load_script
from classes import State
path = "script\\script.txt"

def test_load():
    with open(path, 'r', encoding='utf8') as f:
        script = load_script(f)
    #test script type
    assert script is not None
    assert isinstance(script[8], State)
    assert isinstance(script[1], State)
    assert isinstance(script[2], State)
    assert isinstance(script[3], State)
    assert isinstance(script[4], State)
    assert isinstance(script[5], State)
    assert isinstance(script[6], State)
    assert isinstance(script[7], State)
    assert isinstance(script[9], str)
    #test script content
    assert script[1].id == 0
```

```python
    assert script[1].event['steam'] == 1
    assert script[1].event['epic'] == 4
    assert script[1].event['origin'] == 6
    assert script[1].event['uplay'] == 7
    assert script[2].event['buy'] == 2
    assert script[2].event['search'] == 3
    assert script[2].event['back'] == 0
    assert script[3].event['back'] == 0
    assert script[4].event['back'] == 0
    assert script[5].event['free'] == 5
    assert script[5].event['back'] == 0
    assert script[6].event['back'] == 0
    assert script[3].reply == ("https://store.steampowered.com\n"
    "you can search the game you want and buy it in this website\n"
    "enter: \"back\" to return to the menu\n")
```

运行该测试桩需要安装 pytest：

```
pip install pytest
```

进入该程序的地址后输入以下命令来运行测试 test_parser：

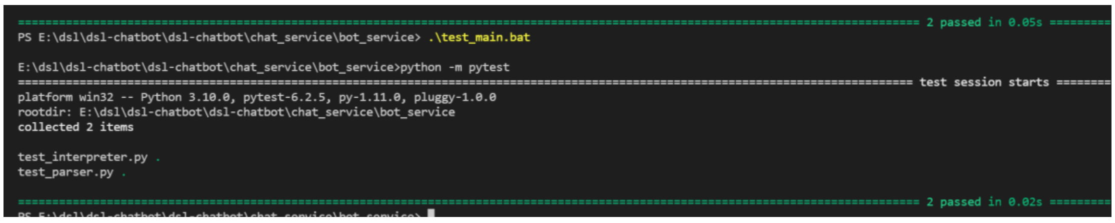
dsl-chatbot\chat_service\bot_service：

pytest test_parser.py -s

interpreter 同理。

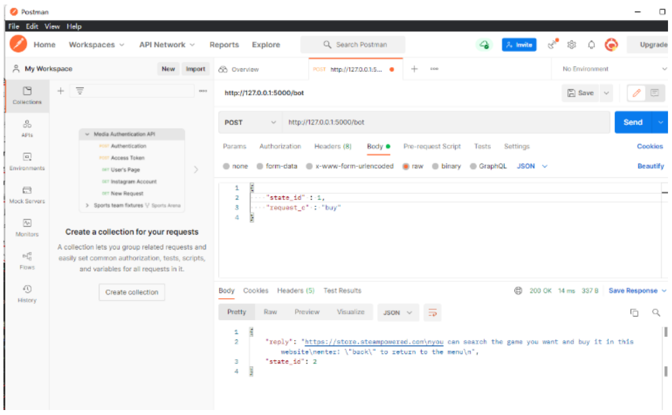
此外根据 pytest 的要求，添加新的测试桩文件名必须以 test_开头或_test 结尾

对于多个测试桩，编写了 test_main.bat 用于一键执行所有 pytest 文件，执行结果如下：



```
=============================================================================================================== 2 passed in 0.05s ========
PS E:\dsl\dsl-chatbot\dsl-chatbot\chat_service\bot_service> .\test_main.bat

E:\dsl\dsl-chatbot\dsl-chatbot\chat_service\bot_service>python -m pytest
============================================================================================================== test session starts =======
platform win32 -- Python 3.10.0, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: E:\dsl\dsl-chatbot\dsl-chatbot\chat_service\bot_service
collected 2 items

test_interpreter.py .
test_parser.py .

=============================================================================================================== 2 passed in 0.02s ========
```

对于后端网络测试，采用 postman 进行接收测试：



```
POST http://127.0.0.1:5000/bot
Body: raw JSON
{
    "state_id" : 1,
    "request_c" : "buy"
}

200 OK  14 ms  337 B
{
    "reply": "https://store.steampowered.com\nyou can search the game you want and buy it in this website\nenter: \"back\" to return to the menu\n",
    "state_id": 2
}
```

## 4.2 构建与运行

本程序需要安装好 python、vue3、npm 或 yarn 以进行安装和运行

**后端程序**

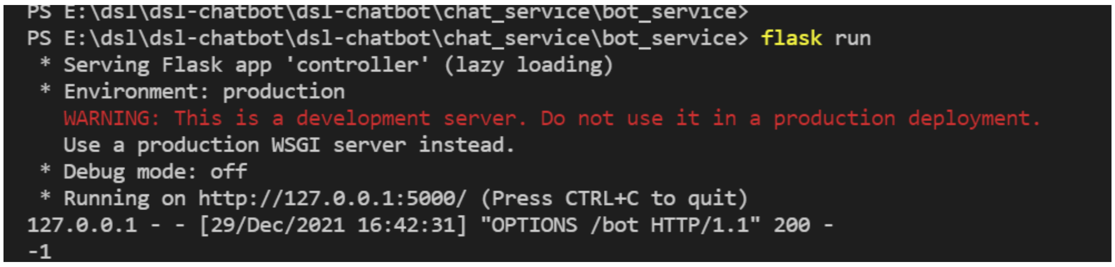
pip install Flask

pip install flask-cors

在 dsl-chatbot\chat_service\bot_service 下

$env:FLASK_APP = "controller"

flask run



```
PS E:\dsl\dsl-chatbot\dsl-chatbot\chat_service\bot_service>
PS E:\dsl\dsl-chatbot\dsl-chatbot\chat_service\bot_service> flask run
 * Serving Flask app 'controller' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [29/Dec/2021 16:42:31] "OPTIONS /bot HTTP/1.1" 200 -
-1
```

**前端程序**

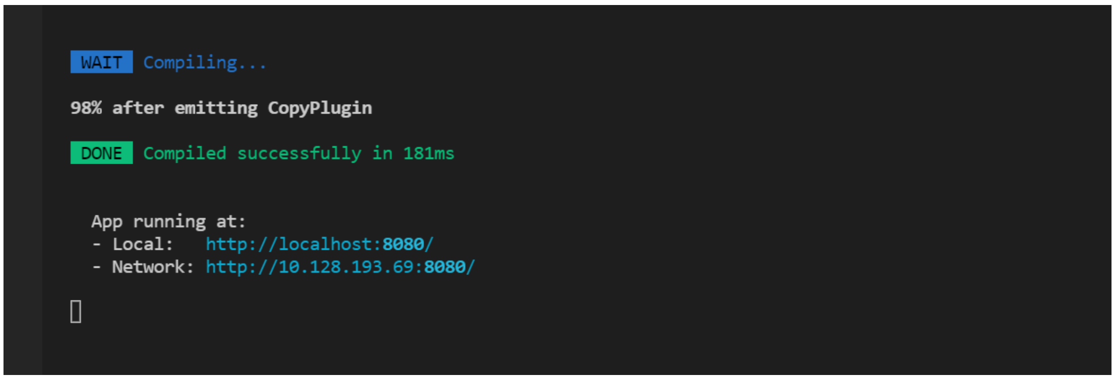
进入 chat_client

yarn add vue3-beautiful-chat

yuan axios

yarn serve

会花费一定时间编译，成功结果如下：



```
 WAIT  Compiling...

98% after emitting CopyPlugin

 DONE  Compiled successfully in 181ms


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://10.128.193.69:8080/

```

使用浏览器进入地址以打开可视化界面。

# 5 接口与可移植性

本程序前端基于 vueJS 编写，支持所有兼容 ECMAScript 5 的浏览器（主流浏览器均支持），前端及通信基于成熟框架编写，编写、维护成本较低，且可直接在 linux 等操作系统中部署。

本程序后端基于 Flask 架构编写，作为一个成熟的后端框架，可部署在 macOS、linux、windows 等环境中，并支持并发等后端需求功能，可移植性较强。

## 接口：

本程序前端请求体：

```
axios.post('http://127.0.0.1:5000/bot', {
        state_id : this.state,
        request_c : message.data.text
        })
```

功能：发送当前状态，发送用户输入的文本

响应体：

```
{
    "reply": "https://store.steampowered.com\nyou can search the game you wa
nt and buy it in this website\nenter: \"back\" to return to the menu\n",
    "state_id": 2
}
```

功能：接收机器人回复 reply，接收状态机的下一状态

后端接受与响应与前端请求与响应对应