北京郵電大學

题目： 一种领域特定脚本语言的解释器的设计与实现

学　　号：2020211472

姓　　名：林志

学　　院：计算机学院（国家示范性软件学院）

指导教师：王智立

2022 年 11 月 28 日

目录

一、 程序设计题目及要求

任务要求：领域特定语言（Domain Specific Language，DSL）可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。

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。

程序输入输出形式不限，可以简化为纯命令行界面。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。

二、 系统设计

本系统基于前后端分离的设计思想，采用 Flask 框架实现后端（服务器）服务，采用 PyQt5 + QtQuick 编写前端（客户端）服务。基于 Python 的 Pyparsing 库实现脚本语言解析任务，基于有限状态自动机的思想构建客服机器人运行逻辑，根据 Restful 接口风格，开发前后端通信接口，根据 python 的 unittest 框架构建单元测试脚本，根据 PyQt5 + QtQuick 实现图形化 GUI，结合 strom 库和 SQLite, 构建数据库，实现信息持久化。

2.1 系统交互设计

系统工作过程：

程序采用前后端分离的设计思想，首先启动服务端，服务端将读取对应语言脚本文件，构建客服机器人应答逻辑，启动一个 WebApp。随后客户端方可对服务端进行访问。

使用系统服务需要用户拥有相应账户，账户根据用户名作为唯一标识。每个未登录用户将默认获得一个访客账户。服务端以用户名为键值，为每个用户分配一个 User 对象，每个用户对象包含一个用户名，一个用户状态对象（记录用户信息），一个计时器（记录用户时间操作）。每个用户在连接时都会取得一个永久有效的 Jwt 令牌，当用户进行注册时，由于用户名发生改变，原令牌将被废弃，用户将重新取得一个新令牌。

客服机器人基于脚本语言进行不同操作，当客户输入不同信息时，客户端将向服务端发送 send 消息，服务端基于 send 信息中的用户请求，已经当前用户状态，执行各类相应服务或动作，并向客户端返回回复信息。脚本语言中含计时操作，为实现该操作，客服端将定时向服务器发送 echo 信息，echo 信息中包含当前计时信息，当计时达到对应标准，服务端将做出相应操作。当用户进入结束状态，会话将终止，对应用户信息将被释放。

2.2 后端架构设计

后端采用 python 语言编写，基于 Flask 框架实现，运行在 Flask 框架的默认 URL: 127.0.0.1:5000。

后端在基于 MVC 设计设计模式的基础上进行。MVC 模型全称 Model-View-Controller（模型-视图-控制器）三层模型，采用 Flask 框架实现 WebApp 时，服务端无需实现视图模块，因此，在本程序设计中，后端仅包含 Model 模块和 Controller 模块。

服务端中，Model 模块为 server 目录，下辖：

database 模块，负责建立相应数据库

parser 模块，解析脚本语言

state 模块，构建基于脚本语言分析结构的有限状态自动机

user 模块，管理和处理用户信息

util 模块，工具包，内含各异常处理

该模块不考虑网络接口的处理，仅对 Controller 层发送来的数据进行处理，并返回处理结果。

服务端中，Controller 模块为 app.py 文件，该模块完成了 Flask 框架的基本构建，并实现了接口。该模块不负责任何业务逻辑的处理，仅负责接受客户端请求，

确认请求合法后交由 Model 层处理，最后将 Model 层返回的处理结果发送到客户端上。

2.3 前端架构设计

前端（客户端）采用 PyQt5 + QtQuick 编写

客户端采用信号-槽结构与界面通信，对于界面上按钮的交互触发一个信号，在客户端中对应的槽对相应的请求进行处理。

消息页面维护一个消息列表，属性有消息内容和消息发送方（一个布尔值），当客户端对消息列表进行更新后，触发 onChanged 信号，界面上的消息就会更新，实现对用户的信息回复。

三、 程序实现

3.1 脚本语言

3.1.1 脚本语言定义

本脚本语言基于 BNF(巴科斯范式)进行定义

BNF 关键字如下

::= : 意为"被定义为"

" " : 代表着字符本身，某些情况下为加粗表示

< > : 包含的为必选项

( ) : 包含的所有项为一组，用于控制表达式的优先级

[ ] : 包含的为可选项，即可出现零次或一次

{ } : 包含的为可重复出现或不出现的项，即可出现零次到无数次

| : 左右两边为并列选项，类似"or"

首先，我们的客服机器人的工作流程大致可以划分为

1. 用户登录 机器人进入工作状态，输出提示信息，并提供选择
2. 用户输入文字，机器人根据不同的请求信息，进行状态转移
3. 因为，在不同的状态下有不同的输出，可以考虑采用米兰机或摩尔机，本程序中，本人采用米兰机进行程序设计。

```
const_num = {"-" | "+"} <number>+
real_num = {"-" | "+"} {<number>} {"."} <number>+ {("e" | "E") {"-" |"+"} <number>+}
const_string = '"' {character} '"'
var = "$"(<letter> | <number> | "_")+
one_var_define = var ("Int" <const_num>)|("Real" <real_num>)|("Text" <const_string>)
vars_define = "Var:" {<one_var_define>}
judge_length = "Length" + ("<"|">"|"<="|">="|"=") + <const_num>
judge_contain = "Contain:"<const_string>
judge_type = "Type"|"Int"|"Real"
judge_equal = <const_string>
judgement = <judge_length>|<judge_contain>|<judge_type>|<judge_equal>
exit_action ="Exit"
goto_action = "Goto"{character}
set_action = "Update:"<var>( ( ("Add"|"Sub"|"Set") (<real_num>|"Input") )|("Set"
<const_string>|"Input"))
content = var | const_string
speak_action = "Speak:"(<content> {"+" <content>}
speak_input = "Speak:"<content>|"Input"{<content>|"Input"}
service = "Service:"<judgement>{<set_action>|<speak_input>}[<exit_action>|< goto_action>]
default = "Default:"{<set_action>|<speak_input>}[<exit_action>|< goto_action>]
wait = "Wait:"<const_num>{<set_action>|<speak_input>}[<exit_action>|< goto_action>]
state_define = "State:"{character}["Logined]{<speak_action>}{<service>}<default>{<wait>}
basic = "Basic:""Name:"{character}"Database:"{character}
```

```
language = basic + {state_define | vars_define}
```

3.1.2 脚本语言介绍

```
针对上述 BNF 给出介绍
本语言的关键字包括 "Basic:"  声明脚本语言基本信息
                "Name:"  声明脚本语言名称
                "Database:" 数据库选项
                "State:" 状态声明
                "Logined"  登录操作关键字
                "Service:" 服务声明
                "Default:" 默认操作声明
                "Wait:" 等待时间操作声明
                "Speak:" 回复动作关键字
                "Goto" 跳转动作关键字
                "Exit" 退出动作关键字
                "Update:"  修改动作关键字
                "ADD" 修改动作之增加动作（针对数字）
                "Sub" 修改动作之减少动作 (针对数字)
                "Set" 修改动作之设置动作
                "Inupt" 用户输入
                "Type" 类型定义
                "INT" 整数型
                "REAL" 实数型
                "TEXT" 文本型          此三种变量借鉴了 SQLlite 数据库的变量定义，
                                    便于后续数据库操作
                "Contain:" 计算用户信息是否包含特定数据
                "Length" 计算用户输入信息长度
                "$" 单一变量声明
                "Vars:"  多变量声明
基于上述关键子的标准，结合 BNF 范式定义的文法，我们可以构造本项目所对应的脚本语言。
```

在本脚本语言中，一级模块包含 脚本语言基本信息、变量定义、状态定义。基于此给出如下介绍。

1. 脚本语言基本信息

   该模块以关键字 Basic:起始，下辖两个分支，包括以 关键字 Name: 起始的脚本语言名，和以关键字 Database：起始的数据库选项。

2. 变量定义

   该模块以关键字 Vars:起始，后跟随任意个遍历定义语句

3. 状态定义

   该模块以关键字 State:起始，后跟随一个状态名，一个可选关键字 Logined，一个 Speak:关键字开头的回复动作,零个或多个 Service：关键字开头的服务分支,一个 Default：关键字开头的默认服务分支，零个或多个 Wait：关键字开头的等待服务分支。

其中，服务分支后需跟随条件判断和跟进动作，而条件判断用于核验用户输入，只有当用户输入满足条件判断时，方可执行服务分支的后续动作。

基于状态定义，我们可以得到，客服机器人需要如下的操作：进入一个状态，读取用户

请求，根据用户请求执行相应服务，根据服务结果做出回复并进入新的状态（新状态可能不变）。

因此得出了 BNF 定义的语言中的 动作 action 和判断 judgemnet

动作 Action:

包括四类

1. 回复动作，以 Speak:关键字开头，返回一段文字信息
2. 跳转动作，以 Goto 关键字开头，转移到一个新状态
3. 退出动作，仅含 Exit 关键字，退出服务
4. 更新动作，以 Updata:关键字开头，可以进行增操作（Add 关键字）减操作（Sub 关键字），改操作（Set 关键字）

判断 judgement:

包括四类

1. 字符串全等判断，检查用户输入字符串和指定字符串是否完全相同
2. 字符串包含判断，检查用户输入字符串内是否包含指定字符串
3. 参数类型判断，检查用户输入字符串是否可以转化为指定类型
4. 长度判断，检查用户输入值是否符合长度规范

3.1.3 脚本语言示例

```
示例:
    Basic:
    Name: robot
    Database: True
    Var:
        $name Text "用户"
    State: Begin
    Speak: $name + "你好"
    Speak: "请输入 退出 以退出"
    Service: "退出"
        Exit
    Default:
    Wait: 30
        Speak: "您已经很久没有操作了，即将于 30 秒后退出"
    Wait: 60
        Exit
```

脚本语言分析结果:

```
    [['Basic:', ['Name:', 'robot'], ['Database:', 'True']], ['Var:', [['$name', 'Text', '"用户"']]], ['State:', 'Begin', [], [['Speak:', ['$name', '"你好"']], ['Speak:', ['"请输入 退出 以退出"']]], [['Service:', '"退出"', [['Exit']]]], ['Default:', []], [['Wait:', 30, [['Speak:', ['"您已经很久没有操作了，即将于 30 秒后退出"']]]], ['Wait:', 60, [['Exit']]]]]
```

分析结果以列表的形式返回，通过列表嵌套在逻辑上形成一颗语法树，树的叶子结点（即列表嵌套数为 0）即为语法元素。

针对该示例，我们可以得出以下结论该语言的基本信息为

名称：robot　是否需要创建数据库：是

内含一个变量 Text 型 name 默认值为“用户”

内含一个状态 Begin

Begin 状态下，操作与服务

提示语： 用户名 + “你好”

提示语：“请输入 退出 以退出”

服务 1：“退出”

进行退出操作

默认服务：

无操作

用户无操作，等待时间 30s 后

提示语："您已经很久没有操作了，即将于 30 秒后退出"

用户无操作，等待时间 60s 后

进行退出操作

注:本程序中，为了保证客服机器人正常执行应答功能，每个脚本语言都要定义一个初始状态 Begin,否则将出现语法错误

3.2 后端程序设计

Model 层,位于 server 目录下

3.2.1 database 模块

概述

database 模块 定义了一个数据库类 Database，用于记录脚本语言解析获得的变量信息，同时用于记录后续用户使用过程中产生的信息，实现数据持久化。为方便变量定义和数据库操作，此处的数据库采用 SQLite,通过 strom 库进行实现 orm 访问。同时，因为 storm 库并非线程安全，所以数据类中定义了一个 lock 变量作为互斥锁，保证数据访问安全。

3.2.2 parser 模块

概述

parser 模块 定义了一个脚本语言分析器类 Parser,用于读取指定脚本文件，分析其语法结构最后返回一个列表类型的语法树，用于构造有限状态自动机。

API

class Parser(object):

脚本语言分析类

根据脚本语言文件，解析脚本语言文法

def analyse_file(file: str) -> list[pp.ParseResults]:

brief 解析一个脚本，脚本存储在文件中

参数:

file: 文件名。

返回值:解析脚本得到的语法树，以列表形式返回，根据列表嵌套层数构成树状结构

3.2.3 state 模块

概述

state 模块 定义了脚本语言对应的有限状态自动机类 StateMachine

服务类 Service

脚本语言的动作基类 Action 及其派生类 ExitAction,GotoAction
UpdataAction,SpeakAction
脚本语言的条件判断基类 JudgeMent 及其派生类
LengthJudgement，ContainJudgement，TypeJudgement，EqualJudgement

该模块通过调用 parser 模块，获取脚本语言对应的语法树，调用 database 模块,建立脚本语言和数据库之间的关系。同时，根据语法树构造有限状态自动机，生成不同状态下对应的服务和回复动作。

API

state.py 文件：

class StateMachine(object):

状态机。

成员变量:

states: 状态集合。

db: 数据库对象

speak_action: 状态默认的 speak 语句集合。

service: 状态的条件分支集合。

default: 状态的默认分支。

wait: 状态的超时转移分支。

def construct_action(self, message: list, target: list[Action], index: int, logined: list[bool], value_check: Optional[str],) -> None:

构建一个动作列表。

参数:

message: 语法树信息，包含一系列动作。

target: 存储动作列表。

index: 状态编号。

logined: 状态是否需要登录验证。

value_check: 校验修改值类型。

def speak(self, user_state: UserState) -> list[str]:

输出某个状态的回复 动作。

参数:

user_state: 用户状态。

返回值: 回复信息。

def state_transform(self, user_state: UserState, msg: str) -> list[str]:

状态转移。

参数:

user_state: 用户状态

msg: 用户输入。

返回值: 回复信息列表。

def timeout_transform(self, user_state: UserState, now_seconds: int) -> (list[str], bool, bool):

超时转移。
参数:
user_state: 用户状态。
now_seconds: 用户未执行操作的秒数。
返回值: 输出的字符串列表、是否需要结束会话、是否转移到新的状态。

service.py 文件:
class Service(object):
条件分支。
成员变量:
judgement: 条件。
actions: 满足条件后执行的动作列表。

action.py 文件:
class Action(metaclass=ABCMeta):
动作抽象基类。

def exec(self, user_state: UserState, response: list[str], request: str, db: Database) -> None:
执行一个动作。
参数:
db:数据库对象
user_state: 用户状态对象
response: 回复信息。
request: 用户请求信息。

class ExitAction(Action):
退出动作，结束一个会话。

def exec(self, user_state: UserState, response: list[str], request: str, db: Database) -> None:
执行退出动作 -- 用户状态置为 -1
参数:
同动作抽象基类 Action.exec

class GotoAction(Action):
状态转移动作，转移到一个新状态。
成员变量:
next: 将转移到的状态。
logined: 新状态是否需要登录验证。

def exec(self, user_state: UserState, response: list[str], request: str, db: Database) -> None:
执行跳转动作 -- 修改用户状态

参数:

同动作抽象基类 Action.exec

class UpdateAction(Action):

更新用户变量动作。

成员变量:

variable: 待更新变量名。

op: 更新操作类型，Add(增） Sub（减) Set（设） 之一

value: 更新的值，可为字符串常量、数字常量、用户输入

value_check: 对动作进行类型检查，查看更新值是否符合类型

def exec(self, user_state: UserState, response: list[str], request: str, db: Database) -> None:

执行数据修改动作

参数:

同动作抽象基类 Action.exec

class SpeakAction(Action):

产生回复动作。

成员变量:

contents: 回复内容列表。

def exec(self, user_state: UserState, response: list[str], request: str, db: Database) -> None:

执行客服回复动作

参数:

同动作抽象基类 Action.exec

judgement.py 文件:

class Judgement(metaclass=ABCMeta):

条件判断抽象基类。

def check(self, check_string: str) -> bool:

判断是否满足条件。

参数:

check_string: 需要判断的字符串。

返回值: 如果满足条件，返回 True；否则返回 False。

class LengthJudgement(Judgement):

长度判断条件，判断用户输入是否满足长度限制。

成员变量:

op: 判断运算符，可以为 ``<``、``>``、``<=``、``>=``、``=`` 其中之一。

length: 长度。

def check(self, check_string: str) -> bool:

参数：同 Judgement.exec

class TypeJudgement(Judgement):

字符串字面值类型判断，判断用户输入是否是某种类型。

成员变量:

type：类型

def check(self, check_string: str) -> bool:

参数：同 Judgement.exec

class EqualJudgement(Judgement):

字符串相等判断，判断用户输入是否和某一个串完全相等。

成员变量:

string：字符串。

def check(self, check_string: str) -> bool:

参数：同 Judgement.exec

3.2.4 user 模块

概述

user 模块定义了 User 类,UserState 类，UserManage 类,VariableSet 类，该模块负责记录用户状态，以及结合脚本语言分析结果，记录用户各类信息。同时基于 Jwt,对用户信息进行加密和解码。

API

class User(object):

用户类。

成员变量:

timer：计时器，根据设定时间，释放用户对象

state：用户状态。

username：用户名。

class VariableSet(object):

脚本语言变量集，与数据库关联。

类变量:

type：表中各列的类型。

username：用户名

password：密码

成员变量:

username：用户名

password：密码

class State(object):

用户状态对象。

成员变量:

state: 用户在状态机中所处的状态。
have_login: 用户是否已经登录。
last_time: 距离用户上次发送消息过去的秒数。
lock: 互斥锁。
username: 用户名。

def register(self, username: str, password: str, db: Database) -> bool:
注册新用户。
参数:
db: 数据库对象
username: 用户名。
password: 密码。
返回值: 如果注册成功，返回 True；否则返回 False。

def login(self, username: str, password: str, db: Database) -> bool:
用户登录。
参数:
db: 数据库对象
username: 用户名。
password: 密码。
返回值: 如果登录成功，返回 True；否则返回 False。

class UserManage(object):
用户管理类。
成员变量:
users: 从用户名映射到 User 对象的字典。
lock: 互斥访问 users 字典的互斥锁。
key: JWT 加密密钥。

def jwt_encode(self, username: str) -> str:
JWT 令牌编码。
参数:
username: 用户名。
返回值:JWT 令牌。

def jwt_decode(self, token: str) -> User:
JWT 令牌解码。
参数:
token: JWT 令牌。
返回值: 如果解码成功，并且用户存在，则返回对应的 ``User`` 对象。
异常:jwt.InvalidTokenError: 当解码失败或者用户名不存在时触发。

def connect(self) -> (User, str):
处理新客户端连接到服务器的请求。

返回值:User 对象和 JWT 令牌。

def login(self, user: User, username: str, password: str, db: Database) -> Optional[str]:

处理登录请求。

参数:

db: 数据库对象

user: 客户端对应的 ``User`` 对象。

username: 注册的用户名。

password: 注册的密码。

返回值: 如果登录成功，返回新 JWT 令牌。否则返回 None。

def register(self, user: User, username: str, password: str, db: Database) -> Optional[str]:

处理注册请求。

参数:

db: 数据库对象

user: 客户端对应的 ``User`` 对象。

username: 注册的用户名。

password: 注册的密码。

返回值: 如果注册成功，返回新 JWT 令牌。否则返回 None。

def timeout(self, username: str) -> None:

超时状态处理函数。

参数:

username: 超时的用户名。

3.2.5 util 模块

概述

util 模块为工具类模块，在本项目中，util 类模块仅包含三个异常类，包括基础异常类 CommonException 未登录异常类 LoginException,语法错误异常类 GrammarException

Controller 层，位于 app.py 文件中:

服务端采用 Flask 封装了 RESTful API。 RESTful API 就是 REST 风格的 API，即 rest 是一种架构风格，跟编程语言无关，跟平台无关，采用 HTTP 做传输协议。

Flask 原生支持多线程，因此服务器的其他组件只需要处理好线程互斥访问，就可以获得较好的并发性。

服务端提供了五个路由接口，定义在 app.py 文件中。分别如下:

connect()函数，对应

GET /

一个新的客户端连接到服务器时，请求一个 token。

Return 返回一个消息列表和 token，格式为：{"msg": ["xxx", "xxx"], "token":

"xxx"}。

Status Codes

- 200 OK – 成功建立会话。

一个客户端与服务器建立连接时，或者客户端开始一个新的会话时，从此路由获取一个token。服务器默认分配一个访客账户，如果设置了默认的问候消息，还会返回消息列表。

send()函数，对应
GET /send

客户端发送一条新消息，服务器返回响应。

Param 客户端发送一条消息和 token，格式为：{"msg": "xxx", "token": "xxx"}。

Return 返回一个消息列表和是否结束会话的标志，格式为：{"msg": ["xxx", "xxx"], "exit": false}。

Status Codes

- 200 OK – 鉴权成功，服务器产生响应。
- 400 Bad Request – 客户端请求消息格式有误。
- 401 Unauthorized – 用户未登录，需要登录。
- 403 Forbidden – 鉴权失败。

一个客户端通过此路由向服务器发送一条消息。收到消息后，服务器首先对 token 进行鉴权，之后对消息进行处理并产生响应，返回一个消息列表。如

果服务器需要终止一个会话，则设 exit 为 True，该 token 过期，客户端需要重新开启一个会话。

echo()函数，对应
GET /echo

客户端发送一条 echo，服务器返回响应。

Param 客户端发送闲置时间和 token，格式为：{"seconds": 5, "token": "xxx"}。

Return 返回一个消息列表、是否结束会话的标志和是否要求用户重置计时器的标志，格式

为：{"msg": ["xxx", "xxx"], "exit": false, reset: false}。

Status Codes

- 200 OK – 鉴权成功，服务器产生响应。
- 400 Bad Request – 客户端请求消息格式有误。
- 403 Forbidden – 鉴权失败。
- 401 Unauthorized – 用户是访客，需要登录。

一个客户端通过此路由向服务器发送一条 echo，表明自己仍然存活和用户闲置的时间。

收到 echo 后，服务器首先对 token 进行鉴权，之后依照闲置时间进行处理并产生响应，返回一个消息

列表。如果服务器需要终止一个会话，则设 exit 为 1，该 token 立即过期，客户端需要重新开启一个

会话。如果服务器要求客户端重置闲置时间计时器，则设 reset 为 1，客户端应当重

启计时器。

register()函数，对应
GET /register

客户端请求注册，服务器返回新的 token。
Param 客户端发送用户名、密码和 token，格式为：{"username": "xxx", "passwd": xxx, "token": "xxx"}。
Return 返回一个新的 token，格式为：{"token": "xxx"}。
Status Codes

- 200 OK – 鉴权并注册成功。
- 400 Bad Request – 客户端请求消息格式有误。
- 403 Forbidden – 鉴权失败。

一个客户端通过此路由向服务器发送一个注册请求。
收到请求后，服务器首先对 token 进行鉴权，之后验证用户名是否合法，如果验证通过，则返回一个新
的 token。原有的 token 立即过期，客户端需要使用新的 token 继续会话。

login()函数，对应
GET /login

客户端请求登录，服务器返回新的 token。
Param 客户端发送用户名、密码和 token，格式为：{"username": "xxx", "passwd": xxx, "token": "xxx"}。
Return 返回一个新的 token，格式为：{"token": "xxx"}。
Status Codes

- 200 OK – 鉴权并登录成功。
- 400 Bad Request – 客户端请求消息格式有误。
- 403 Forbidden – 鉴权失败。

一个客户端通过此路由向服务器发送一个登录请求。
收到请求后，服务器首先对 token 进行鉴权，之后验证用户名和密码，如果验证通过，则返回一个新的
token。原有的 token 立即过期，客户端需要使用新的 token 继续会话。

3.3 前端程序设计

前端采用 PyQt5 + QtQuick 编写前端
main.py 文件内定义了前端数据交互逻辑，负责同服务端进行通信。
main.qml 文件内定义了前端的图形化界面，同时调用 main.py 内的客户端模型类 ClientModel 进行交互。
客户端模型 Client 定义如下：
class Client(QObject):
客户端模型。与 QML 交互。
成员变量：

message：消息列表。

lock：互斥锁。

token：令牌。

logined：是否已经登录。

timer：超时计时器，监测用户闲置时间。

time_count：用户闲置时间计数器。

类变量

message_changed = pyqtSignal()

用于同 qml 进行交互

API:

def get_message(self) -> QQmlListProperty:

为图形化界面获取消息列表

def append_message(self, msg: Message) -> None:

消息列表数据追加

def send_message(self, msg: str) -> None:

向服务器发送 send 信息，请求数据

def login(self, username: str, passwd: str) -> None:

向服务器端发送登录信息，尝试登录

def register(self, username: str, passwd: str) -> None:

向服务器端发送注册信息，尝试注册

def connect(self) -> None:

客户端向服务器请求连接

def timeout(self) -> None:

用户长时间未操作，进行超时处理

效果图如下：


DSL
客服机器人应答系统
登录/注册
年后
很抱歉，没有能够理解您的意思
你好，用户
请输入 余额 以查看余额，输入 改名 以修改名字，输入 投诉 进行投诉，输入 介绍 显示服务信息，输入 退出 以结束会话，
投诉
请输入您的建议，不超过200个字符
您已经很久没有操作了，即将返回主菜单
你好，用户
请输入 余额 以查看余额，输入 改名 以修改名字，输入 投诉 进行投诉，输入 介绍 显示服务信息，输入 退出 以结束会话，
text
发送


DSL
用户登录/注册
用户名
密码
登录
注册

四、测试

测试模块基于 python 内置的单元测试框架 Unittest 编写，其中，所有测试方法均需以 test 开头。

有如下几个部分

test_parser.py	脚本语言解析器测试
test_speak_action.py	回复动作测试
test_update_action.py	更新动作测试
test_user_state.py	用户状态测试
test_state.py	有限状态自动机构建测试
test_stub.py	客户端测试，生成一个简易服务端作为测试桩
test_pressure.py	压力测试
test_app.py	服务端测试

在测试模块中，采用了大量的断言操作进行测试验证

如在 test_parser.py 模块中，测试脚本读取了两个正确脚本文件，case1.txt, 和 case2.txt，调用 parser 类中的 anlyse_file 函数进行脚本解析,并断言其结果等于与 result1.txt,result2.txt 两个结果文件内的信息。

同时，读取了三个错误文件 case3.txt,case4.txt,case5.txt, 并断言其将引发 ParseException 异常（即脚本语言解析错误）。

测试结果如下：

```
D:\Python3.10\python.exe "D:\PyCharm Community Edition 2022.1.3\plugins\python-ce\helpers\pycharm\_jb_unittest_runner.py" --path D:/Desktop
Testing started at 22:40 ...
Launching unittests with arguments python -m unittest D:/Desktop/programDesign/DSL/test/test_parser.py in D:\Desktop\programDesign\DSL\test


Ran 1 test in 0.021s

OK

进程已结束,退出代码0
```

表明断言全部成功，说明该模块执行正确，同时我们使用反例，断言对 case1.txt 文件的测试将发生 ParseException 异常，测试结果如下：

```
Ran 1 test in 0.014s

FAILED (failures=1)

Failure
Traceback (most recent call last):
  File "D:\Desktop\programDesign\DSL\test\test_parser.py", line 21, in test_analyse_file
    with self.assertRaises(ParseException):
AssertionError: ParseException not raised
```

异常没有产生，说明断言失败，说明 case1.txt 文法正确，表明测试方法无误。

针对回复动作模块进行测试 test_speak_action.py 文件内采用了默认输入的

方式。通过指定输入参数信息，随后获取回复值，同预期值进行断言比较，从而实现测试。测试结果如下：

```
D:\Python3.10\python.exe "D:\PyCharm Community Edition 2022.1.3\plugins\python-ce\helpers\pycharm\_jb_unittest_runner.py" --path D:/Desktop/p
Testing started at 0:24 ...
Launching unittests with arguments python -m unittest D:/Desktop/programDesign/DSL/test/test_speak_action.py in D:\Desktop\programDesign\DSL\


Ran 1 test in 0.026s

OK

进程已结束,退出代码0
```

test_update_action.py

test_user_state.py

test_state.py

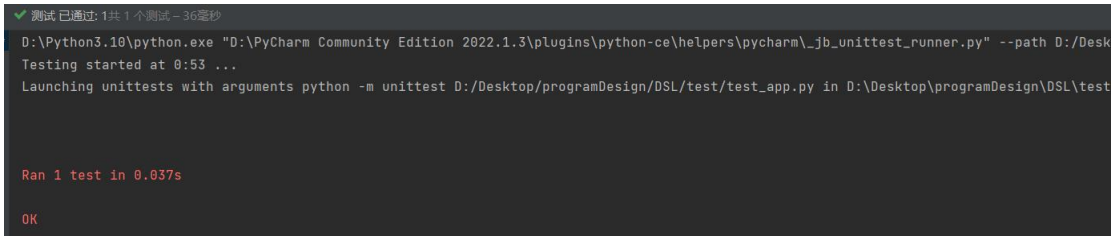
上述三个测试模块同理，皆采用默认参数输入的方式（即测试桩以默认参数的形式代替），对返回值进行断言判断，当断言成功，证明程序可以正确执行。

test_stub.py 测试模块用于测试客户端，该模块模拟了一个简易服务端（即测试桩），用于同客户端进行交互，验证各通信接口是否正常，客户端 GUI 页面能否正常显示，测试结果如下：

客户端程序能够正常运行。

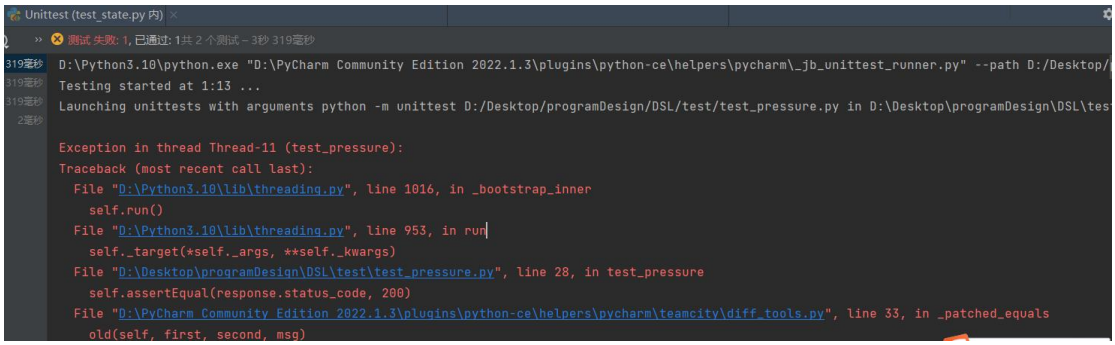
test_app.py 测试模块用于测试服务端功能。测试脚本中，通过以默认参数值模拟客户端发送请求，后对服务端的响应进行断言，由此进行测试。

测试结果如下：



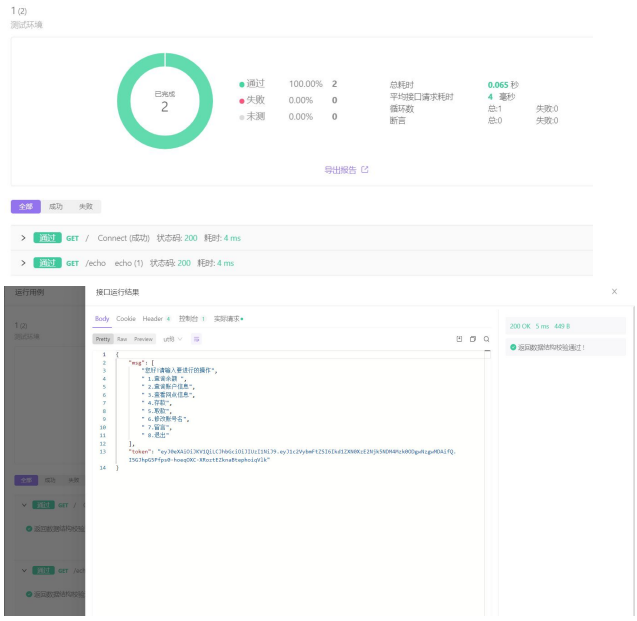
最后是 test_pressure.py 模块,该模块负责进行压力测试，测试程序通过同时开启 n 个客户端（n 可变），对服务端进行访问，查看服务器对多客户端同时发送请求的承载能力。

测验结果如下：



当有 100 个客户端同时发送访问请求时，约 30%的客户端将访问失败，程序并发性仍有提升空间。

另外，为了对服务端接口进行单独测试，我借助了 Apifox 提供的自动化测试工具进行接口测试，测试如下：



最后，test.sh 文件整合了所有测试模块，执行根目录下的 test.sh 文件，即可运行所有测试模块。

五、用户使用手册

依赖安装：

pip install -r requirements.txt

服务端运行：

Python -m flask run --host --port

默认 127.0.0.1:5000

客户端运行：

cd client python -m main.py

测试集运行

./test.sh