

# Python程序设计报告

设计题目：基于socket通信的多人聊天室系统

学生姓名：刘嘉伟

专 业：计算机创新实验18-1班

班 级：1班

学 号：2018213106

指导教师：李培培

完成日期：2020年6月16日

# （一）需求和规格说明

**问题描述：**这是一个具有图形界面的多人聊天室系统，在数据库中存放多个登陆系统的账号密码，启动服务器，就能开启多个聊天程序，通过账号密码登陆进入聊天室，开始多人一起聊天。

**编程任务：**

1. 设计服务器的通信协议（信息传送格式）
2. 编写服务器的配置文件，包含IP，端口号，数据库用户名密码等
3. 编写服务器传送信息的工具类（对信息解码或编码）
4. 编写服务器的通信，转发，响应登陆请求等代码
5. 将服务器连接至数据库
6. 编写客户端配置文件
7. 编写客户端传送信息工具类（对信息解码或编码）
8. 编写客户端核心代码，发送，登陆等功能代码
9. 设计登陆界面GUI界面
10. 设计聊天室GUI界面

# （二）设计

## 1．设计思想

本项目的主要设计思路是使用套接字进行服务器与客户端的通信，对各个客户端服务的并行服务需要多线程技术。通过套接字不断接收客户端的请求，一旦接收到请求，就将该请求分配给一个新的新的线程去执行，并同时将该用户记录下来。这个线程不断接收对应的客户端发送来的信息，如果是登陆信息，就到数据库中去查询用户名与密码是否一致，如果一致则返回登陆成功信息，否则返回失败信息。多人聊天的思想是：服务器接收到某一个用户发来的聊天信息，就将该信息提取出来，然后转发至除了本用户的已经记录下来的其他在线用户中。然后将该信息显示在每个用户的聊天面板上，就能实现多人聊天的效果。一旦有用户退出系统，服务器就会将该用户从在线列表中移除，不转发信息。

## 2. 设计表示

1. 通信协议设计：

|  |  |  |
| --- | --- | --- |
| **变量** | **值** | **描述** |
| REQUEST\_LOGIN | **'0001'** | 登陆请求 |
| REQUEST\_CHAT | **'0002'** | 聊天请求 |
| RESPONSE\_LOGIN\_RESULT | **'1001'** | 登陆响应结果 |
| RESPONSE\_CHAT | **'1002'** | 聊天响应结果 |
| DELIMITER | **'|'** | 自定义协议数据分隔符 |

1. 服务器：

|  |  |
| --- | --- |
| **类名** | **说明** |
| ServerSocket | 自定义套接字，负责初始化服务器套接字需要的相关参数 |
| ResponseProtocol | 服务器响应协议的格式字符串处理类 |
| Server | 服务器核心类 |
| SocketWrapper | 套接字包装类 |
| DB | 数据库管理操作类 |

1. 客户端：

|  |  |
| --- | --- |
| **类名** | **说明** |
| ClientSocket | 客户端套接字的自定义处理 |
| RequestProtocol | 客户端响应协议的格式字符串处理类 |
| Client | 客户端核心类 |
| WindowChat | 聊天窗口界面类 |
| WindowLogin | 登陆窗口界面类 |

1. 类函数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| server | 函数 | void | startup | 程序启动函数 |
| void | request\_handle | 客户端请求响应函数 |
| dict | parse\_request\_text | 请求数据解析函数 |
| void | request\_login\_handle | 登陆请求处理函数 |
| void | request\_chat\_handle | 消息发送处理函数 |
| 数据 | ServerSocket | server\_socket | 自定义套接字 |
| dict | clients | 在线用户套接字字典 |
| dict | request\_handle\_function | 响应处理函数字典 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| server | 函数 | void | startup | 程序启动函数 |
| void | reponse\_handle | 服务器返回的处理函数 |
| dict | parse\_ reponse \_text | 响应数据解析函数 |
| void | response\_login\_handle | 登陆请求发送函数 |
| void | response \_chat\_handle | 消息发送发送函数 |
| 数据 | ClientSocket | conn | 套接字 |
| string | username | 当前用户的用户名 |
| bool | is\_running | 程序运行标记 |

1. 通信流程

你好

你好

**服务器将”你好”转发给其他在线用户**

客户端A

客户端B

客户端C

服务器

**发送聊天内容”你好”到服务器**

**服务器-客户端，客户端-客户端通信流程**

1. 设计流程

创建套接字并监听指定端口

开启线程处理当前客户端请求

接收客户端请求，并判断请求类型

登陆请求处理

聊天请求处理

**服务器设计流程图**

登陆成功

否

是

用户登陆

向服务器发送登陆请求

用户与服务器请求处理

发送消息请求处理

服务器转发消息处理

**客户端设计流程图**

## 3. 核心算法

服务器核心算法：

1. 开启客户端服务线程

|  |
| --- |
| **def** startup(self):  *'''获取客户端连接，并提供服务'''* **while True**:  *# 获取客户端连接* print(**'获取客户端连接~~~'**)  soc, addr = self.server\_socket.accept()  print(**'连接成功'**)  *# 使用套接字生成包装对象* client\_soc = SocketWrapper(soc)  *# 收发消息* Thread(target=**lambda**: self.request\_handle(client\_soc)).start() |

通过一个死循环，不断接收客户端发来的请求，若发现请求，就将该请求送到函数self.request\_handle去处理，传递的参数是包装好的套接字，然后就能继续就收别的客户端的请求

1. 用户消息转发

|  |
| --- |
| **def** request\_chat\_handle(self, client\_soc, request\_data):  *'''处理聊天功能'''* print(**'收到消息，准备处理'**)  print(request\_data)  *# 获取消息内容* username = request\_data[**'username'**]  messages = request\_data[**'messages'**]  nickname = self.clients[username][**'nickname'**]  *# 拼接发送给客户端的消息文本* msg = ResponseProtocol.resopnse\_chat(nickname, messages)  *# 转发给在线用户* **for** u\_name, info **in** self.clients.items():  **if** u\_name != username: *#不需要向发送消息的账号转发数据* info[**'sock'**].send\_data(msg) |

传进来的request\_data是包含发送消息用户名，消息内容，昵称的字典，通过工具类将转发消息转换为协议数据，然后遍历事先准备好的包含所有在线用户的列表，将消息逐个转发给除了自己的在线用户，info是每个在线用户的套接字，通过套接字能很方便的将信息发送出去。

1. 数据库查询用户信息

|  |
| --- |
| **def** get\_one(self,sql):  *'''使用sql语句查询用户信息'''  # 执行sql语句* self.cursor.execute(sql)  *# 获取查询结果* query\_result = self.cursor.fetchone()  *# 判断是否有结果* **if not** query\_result:**return None** *# 获取字段名称列表* fields=[field[0]**for** field **in** self.cursor.description]  *# 使用字典和数据合成字典，供返回使用* return\_data={}  **for** field,value **in** zip(fields,query\_result):  return\_data[field]=value  **return** return\_data |

通过SQL语句查询该用户信息是否存放在数据库中，如果没有就返回None。

1. 客户端请求处理

|  |
| --- |
| **def** request\_handle(self, client\_soc):  *'''处理客户端请求'''* **while True**:  *# 接收客户端数据* recv\_data = client\_soc.recv\_data()  **if not** recv\_data:  *# 没有接收到数据客户端应该已经关闭* self.remove\_offline\_user(client\_soc)  client\_soc.close()  **break** *# 解析数据* parse\_data = self.parse\_request\_text(recv\_data)   *# 分析请求类型，并根据请求类型调用相应的处理函数* handle\_function = self.request\_handle\_function.get(parse\_data[**'request\_id'**])  **if** (handle\_function):  handle\_function(client\_soc, parse\_data) |

通过一个死循环，不断接收客户端发来的协议消息，将协议消息进行解析，然后传给对应请求类型的处理函数去处理，如果是登陆请求0001，就传给0001对应的函数去处理，如果聊天请求0002，就传给0002对应的函数去处理。这里使用了一个技巧，将每种类型的请求代码和请求函数通过字典绑定在一起，使用和拓展起来都会很方便，只需要在初始化函数中，将请求类型与处理函数进行注册就可以，不需要修改逻辑代码。

客户端核心算法：

1. 开启用户消息处理子线程

|  |
| --- |
| **def** startup(self):  *'''开启窗口主循环'''* self.conn.connect()  Thread(target=self.response\_handle).start() *# 创建并开启子线程来接受消息* self.window.mainloop() |

1. 登陆结果响应

|  |
| --- |
| **def** response\_login\_handle(self,response\_data):  *'''登陆结果响应'''* print(**'接收到登陆信息'**,response\_data)  result = response\_data[**'result'**]  **if** result==**'0'**:  showinfo(**'提示'**,**'登陆失败,账号或密码错误'**)  print(**'登陆失败'**)  **return** showinfo(**'提示'**,**'登陆成功!'**)  *# 登陆成功获取用户信息* nickname = response\_data[**'nickname'**]  self.username = response\_data[**'username'**] *# 保存登陆用户的账号，供发送消息使用  # print('昵称：',nickname)  # print('账号：',username)  # print('登陆成功')  # 显示聊天窗口* self.window\_chat.set\_title(nickname)  self.window\_chat.update()  self.window\_chat.deiconify()  *# 隐藏登陆窗口* self.window.withdraw() |

如果从服务器接收到登陆失败的信息，就弹出提示框，登陆失败。否则如果服务器发送的是登陆成功，就将登陆窗口给隐藏，将聊天窗口显示出来，并保存登陆用户的账号。

# （三）用户手册

首先手动开启服务器程序，让服务器处于监听状态，然后检查数据库是否打开了，如未打开，手动打开数据库程序。

运行客户端程序，输入用户名与密码，然后点击登陆，如果用户名与密码正确，就能成功进入聊天室，聊天室顶部有你的用户名信息，在底部输入框中输入要发送的信息，点击发送就能把消息发送到聊天室，所有在线的用户都能看见你发送的消息，你也能看见所有在线用户向你发送的消息。如果要退出，点右上角的×就能退出聊天室，退出后就不能收到聊天内容了。

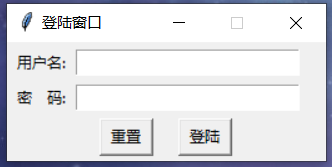
# （四）调试及测试

## 1. 功能演示：

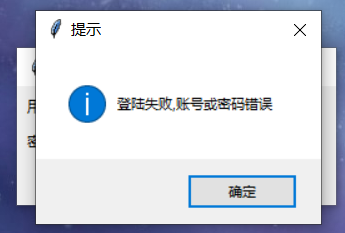
开启服务器：



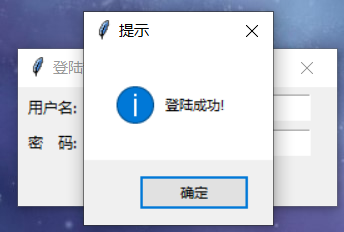
开启客户端：



登陆密码错误时：



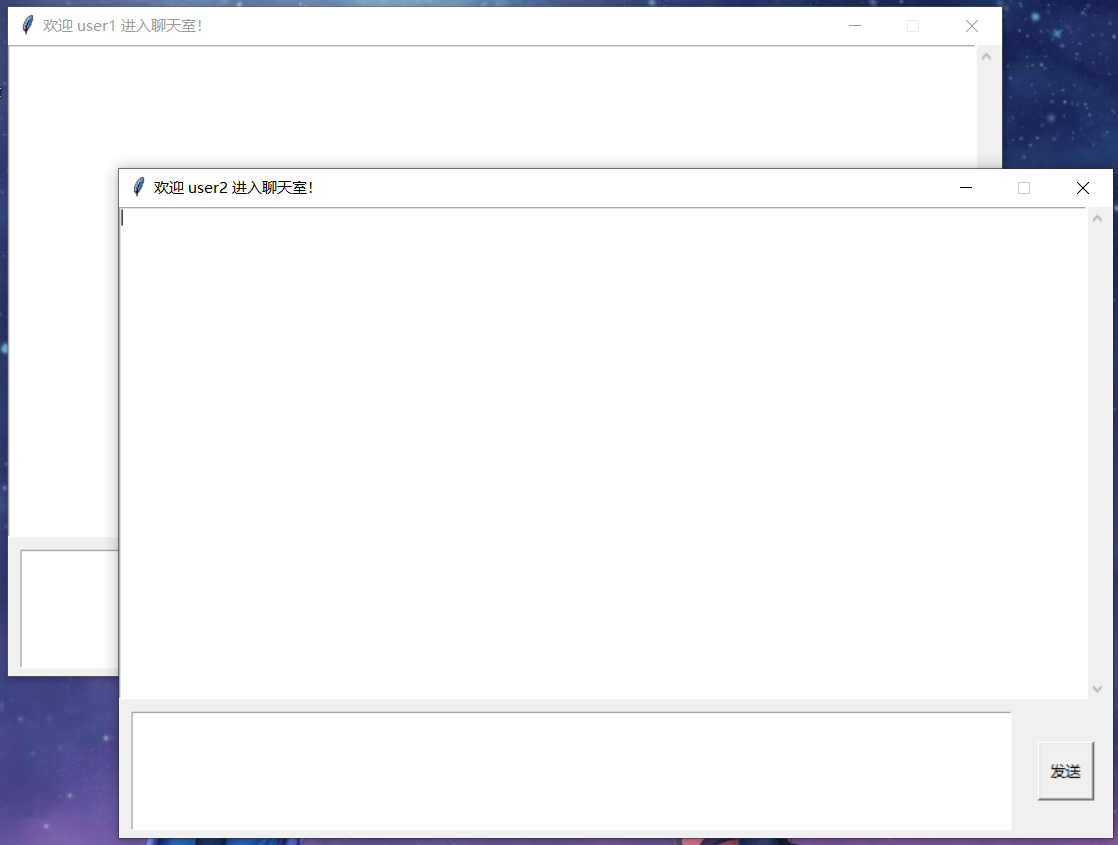
密码正确时：



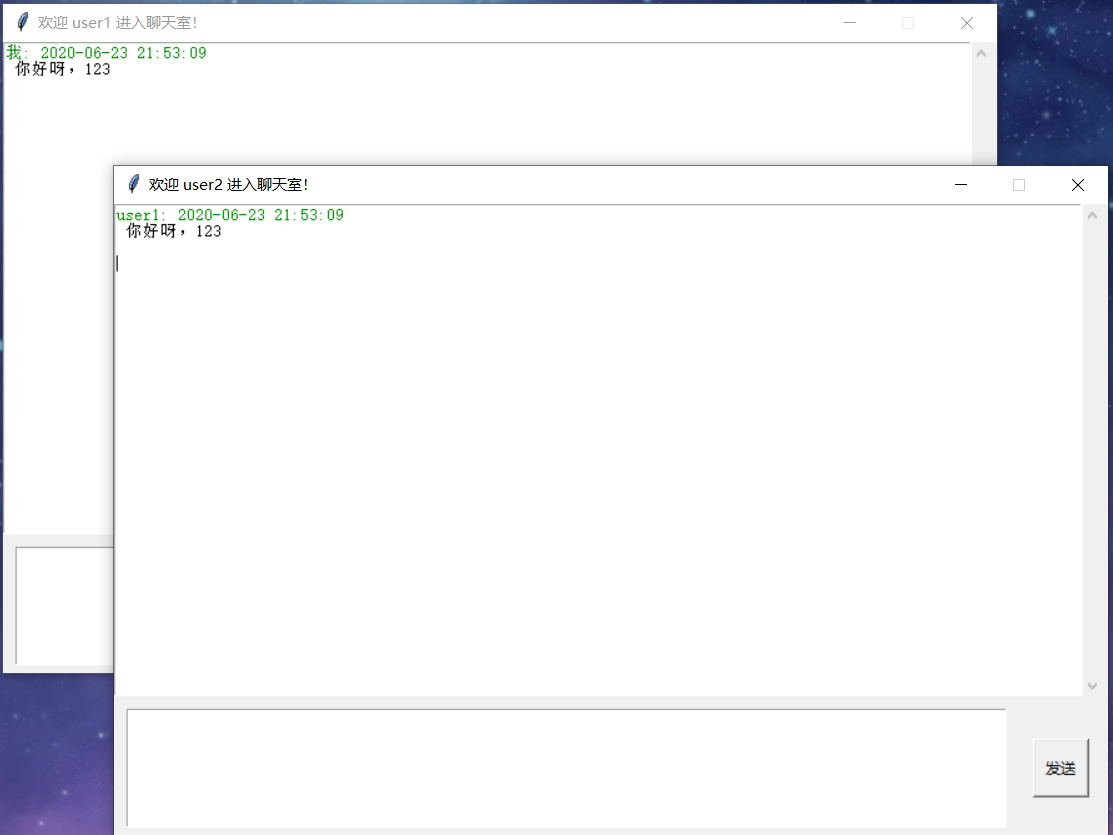
登陆成功进入聊天室界面：

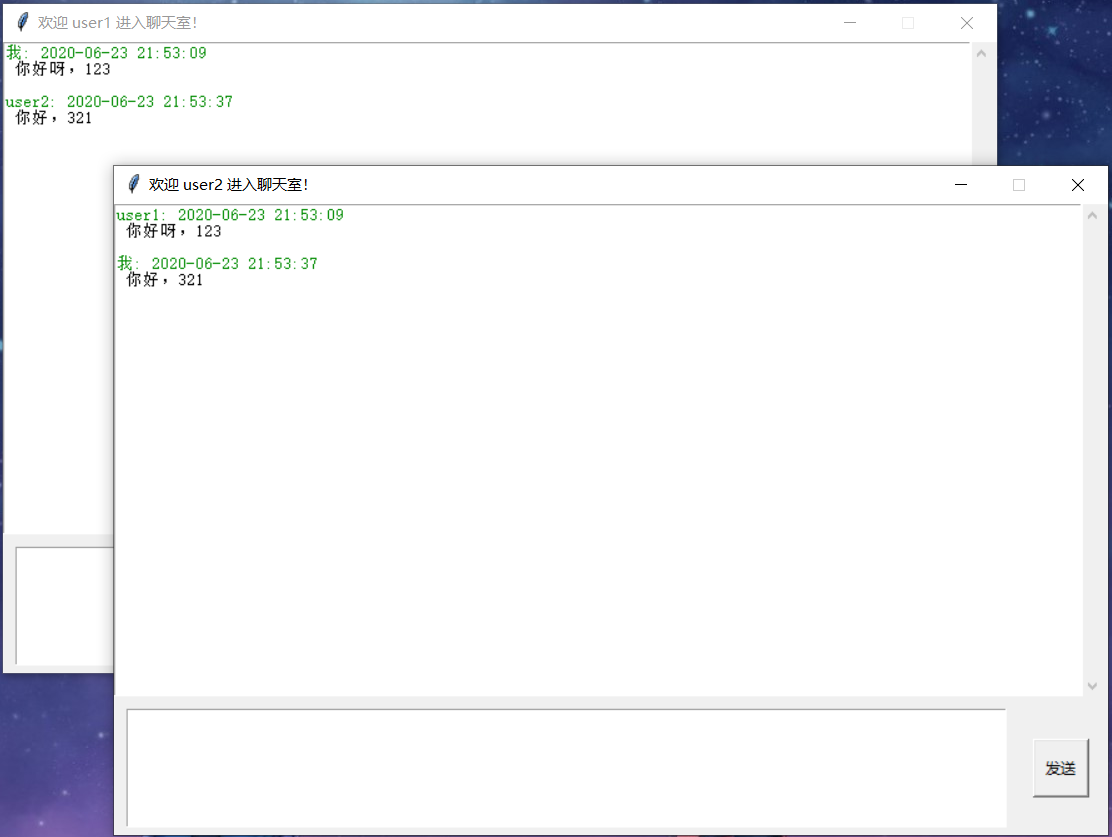


多用户在线：

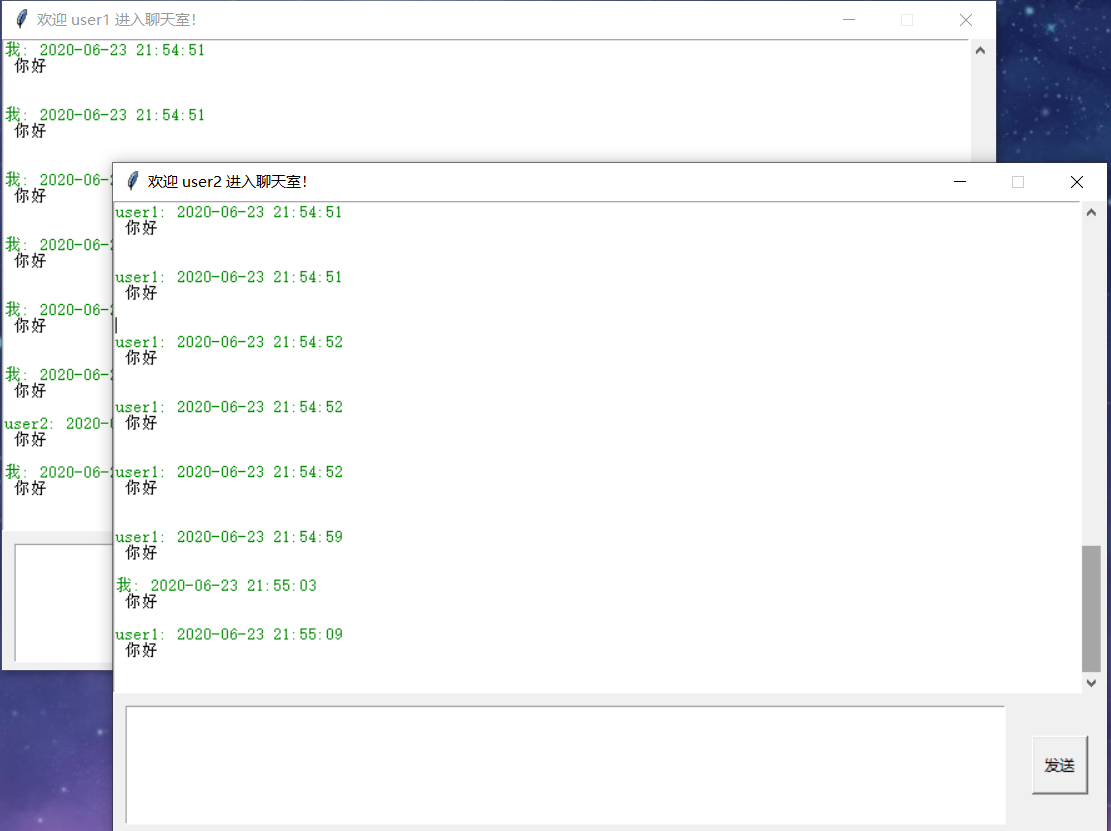


多用户聊天：





当一整面消息满时继续发送，页面会自动向下滚动



## 2. 进一步改进：

（1）没有将整个软件进行发布成一个exe可执行文件，需要手动在IDE内打开服务器，打开客户端，后续可以将软件发布成可执行文件，双击打开就能运行。

（2）没有实现用户注册功能，需要手动在数据库中添加用户，可以在登陆界面添加一个用户注册功能，将新用户直接添加到数据库中，不用手动写sql语句添加。

（3）只实现了在一台计算机上通信，可以将软件进一步完善，使得在不同网段下的计算机都能实时通讯（可以通过配置文件修改连接的IP地址，但是没有测试过，也不知道怎么能连接别的IP下的计算机）

（4）图形界面还比较简陋，可以继续将界面美化，添加更多功能，比如查看在线用户。

# (五) 感想

这是我第一次写python的图形界面程序，tkinter的用法还是比较简单的，与其他语言虽然有区别，但是区别不大，很好上手。我所制作的这个多人聊天软件的面向对象编程的特点十分明显，每一个模块都有一个属于自己的类，然后在核心类中通过调用各个模块，来实现整体的功能。通过这个项目的实现，我也学到了很多编程习惯和python的很多独特的技巧，比如注册函数。通过一个键为事先设定好的标记，值为对应的处理函数的字典，可以很方便很灵活的调用不同的函数来处理可能发生的事件，将用户登陆请求和用户处理请求放在同一段代码中执行，这是其他语言所没有的特性。还学到了配置文件的重要性，在实际的项目工程中，如果要修改传输的协议的信息，或者主机IP，或登陆数据库的用户名和密码，在源代码中直接修改是不太现实的，必须要写一个配置文件，里面包含了所有的可能会发生变化的信息，比如数据库名称，用户名密码，IP，port等等信息，这样才能实现数据与程序分离，否则后期的变动会导致代码修改非常困难。注册思想也非常重要，就是之前所提到的字典。对于业务的拓展，如果使用这种字典的方式，只需要添加业务代码，然后将代码和标记进行注册，不需要修改逻辑代码，便能使用这个函数，通过这种方式后期修改的时候可以减少很多不必要的错误，也很好的体现了模块化编程的思想，而这种思想是python独有的。总之，这个项目对我的收获很大。