装订线

山东农业大学

毕 业 论 文

题目：基于 Python 的旅行信息服务聊天机器人的设计与实现

院　　部 信息科学与工程学院

专业班级 计算机科学与技术 2016-5

届　　次　2020 届

学生姓名　刘天睿

学　　号　20164185

指导教师　张广梅(副教授)

二 O 二 O 年五月二十日

...
...

基于 Python 的旅行信息服务聊天机器人的设计与实现

Design And Implementation Of Chatbot For Travel Information Service Based On Python

专业 Speciality	计算机科学与技术 Computer Science and Technology
学生 Undergraduate	刘天睿 Liu Tian Rui
指导教师 Supervisor	张广梅 Zhang Guang Mei

基于 Python 的旅行信息服务聊天机器人的设计与实现

【摘要】本文详细介绍了基于 python 的旅行信息服务聊天机器人的设计与实现的过程，包括 python 环境的搭建，所需数据的获取及存储，聊天机器人框架的搭建，以及可视化界面的实现。具体研究工作如下：在获取数据方面，利用 selenium 和 parsel 构建网络爬虫对动态目标网页的内容的爬取与清洗。在数据存储方面，使用 MongoDB 非关系型数据库对所获取的数据进行存储，同时为防止数据损失，利用 csv 文件对爬取的数据进行备份。在数据分析层面，使用 python 第三方库 matplotlib 对所获取的数据进行分析与统计图表的制作。利用 Rasa 框架搭建本系统的核心——聊天机器人。在数据可视化方面，使用 Html5 与 css 搭建系统前端，并通过接口与后台服务进行对接。本系统能够实现与用户对话的基本功能，且供用户查询并收集各类旅行信息。实现了一个完整的针对于旅行信息服务的任务型聊天机器人。

【关键词】Python；旅行信息服务；网络爬虫；MongoDB；Rasa 框架；

Design And Implementation Of Chatbot For Travel Information Service Based On Python

【**Abstract**】This paper describes the Design and implementation of chatbot for travel information service based on python, including the construction of python environment, the acquisition and storage of required data, the construction of chatbot framework, and the realization of visual interface. The specific research work is as follows: in terms of data acquisition, selenium and parsel were used to build web crawlers to crawl and clean the contents of dynamically targeted web pages. In terms of data storage, MongoDB non-relational database was used to store the acquired data, and CSV file was used to backup the crawled data in order to prevent data loss. At the data analysis level, the third party library of python matplotlib is used to analyze and make statistical charts for the acquired data. Rasa framework is used to build the chatbot, the core of the system. In terms of data visualization, Html5 and CSS are used to build the front end of the system and interface with the background services. The system can achieve the basic function of dialogue with users, and for users to query and collect all kinds of travel information. A complete mission chatbot for travel information service is implemented.

【**Key words**】Python; Travel information service; Web crawl; MongoDB; Rasa framework

1 引言

1.1 研究背景及意义

在改革开放之后的几十年的历史中，团体旅游作为一种较为全方位的旅游方式，为人们的出行娱乐提供了便捷，逐步成为一种领先的旅行方式。人们选择出门旅行的目的也各不相同，但拥有一次相对自由并且舒心的出行则是大多数人所向往的。但是大多数从事旅游业的人却将这种团体旅游作为了一种以赚钱为目的的事业。于是，问题也随之出现：团体旅游缺乏自主性，缺少个人自由，自己想要看的景点并没有时间看，还有很多地方的强制消费也逐步被曝光，使得人们对于团体旅游产生了一些担忧。

而随着互联网以及无线网络的普及和发展以及人们生活水平的提高，越来越多的人转向自助游。自助游是由用户自己设计旅行路线以及旅途安排的一种旅行方式，这在一定程度上提高了用户出行的自由性。随着各种服务 APP 的出现，用户可以选择更优质便捷的食宿、交通等，同时也避免了强制消费的出现。

在网络上，关于旅行景点等信息越来越多，许多网站也都有专门的模块对景点进行介绍。但是互联网在快捷、方便的带来大量信息的同时，也带来了一些问题：网站样式繁多复杂，广告繁多，信息形式不一致等。于是用户需要花费大量的时间，来搜集一些旅行的信息。根据一份对周围人的调查结果显示，在出行之前做旅行攻略的人要远远大于不做的人，而做旅行攻略的人也大多反映时间长，内容多的问题。如何克服这些问题，即是本文所研究的系统要做的内容。

近年来，随着互联网数据规模的爆发式增长和人工智能技术的进步，聊天机器人作为一种具有良好的人机交互性产品，逐渐成为了学术界和工业界的研究重点[1]。而聊天机器人在功能上可以分为任务型聊天机器人以及闲聊型聊天机器人。如淘宝智能客服、苹果的手机助手 Siri 都是任务型聊天机器人，它们都是以某一任务进行驱动的。对话系统的研究与发展，尤其是对任务型对话系统的探索，有助于提高人机交互模式下的社会工作效率[2]。基于此，本文设计并实现了一款以旅游信息服务为任务驱动的任务型聊天机器人。

1.2 研究与发展现状

目前，供用户查询旅行地信息的渠道多种多样，网站只是一种最基本的方式。APP 的出现也使得信息查询朝着更加便捷的方向发展。目前很多旅行攻略 APP[3]，是将社会大多数实体旅游业的旅游资源进行线上整合的虚拟平台，用户只需用 APP 客户端搜索想要查询的旅游信息，就可以迅速检索到自己想要的内容。

在国外，旅游 APP 占领了旅游市场的重要份额。根据艾瑞数据显示，在线旅游移动端用户使用旅游 APP 查询主要集中在休闲度假类产品，占比 97%，远超占比 59%的商业旅行。如国外的 Airbnb 与 Booking 都是以自助式与家庭式的旅行模式进行运营。

但是在网络上查询的信息，格式繁杂，内容冗长，与用户的交互性较差。聊天机器人的出现可以较好的解决这一问题。聊天机器人能够将网上的信息进行整合，去掉一些烦冗的内容，以更精简、更智能、更人性化的语言反馈给用户。而在各类聊天机器人快速发展的今天，基于机器学习的任务型聊天机器人也得到了广泛的应用。于是可以将这一类的聊天机器人与旅行信息的服务相结合，制作出以旅行信息服务为任务驱动的聊天机器人，方便用户的查询与使用。

2 需求分析

2.1 聊天机器人的总体需求

该聊天机器人的设计是为了方便用户为出行定制计划，为用户提供每一次出行所需要的各类旅行信息。基于此，该机器人需要通过搜集旅行网站上已存在的旅行信息，再将获取的信息进行一定程度上的处理，在用户需要的时候以更便捷的方式和更精简的语言反馈给用户。为了实现此目的，本系统在设计方面上，需要首先通过网络爬虫爬取网站上的各类旅行信息，并将获取到的信息进行相应的清洗和处理、分类和整理，之后存入数据库中方便后续的使用。同时使用获取到的数据通过一定处理制作出排行统计图及词云等图片类文件，在与用户交流的过程中反馈给用户，方便用户更直观的了解信息。该机器人的总体需求主要包含以下几部分：第一部分为网络爬虫的设计与实现，通过网络爬虫对旅行信息网站进行解析和数据获取，再利用特定的包对获取到的网站源码等信息进行进一步的元素定位与内容获取，从而得到用户所需要的数据，并将数据进行进一步处理后存入数据库。第二部分为数据处理的实现，对从旅行信息网站上获取到的数据制作排行统计图以及词云等。第三部分为聊天机器人的搭建，也是本系统的核心。通过框架搭建好聊天机器人，并利用先前获取到的数据对聊天机器人进行训练，能够让机器人与用户进行有效的沟通。第四部分为前端的设计与实现，即用户可以通过可视化窗口与机器人进行对话。该系统时序图如图 2.1 所示。

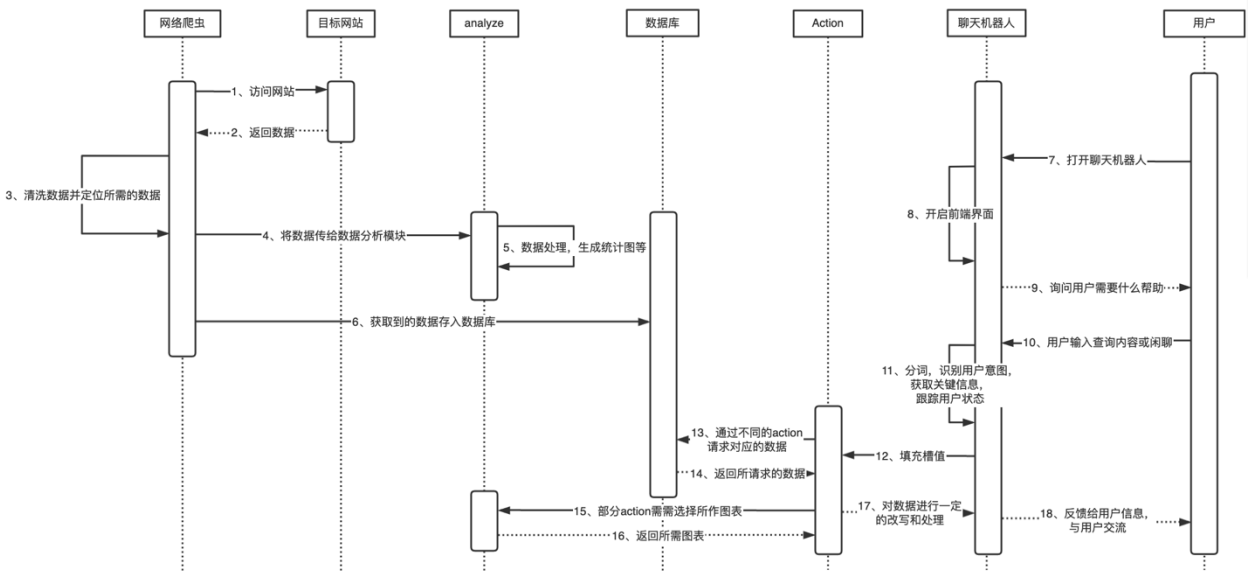

图 2.1 系统运行时序图

网络爬虫主要解决对于网站上特定信息的获取。在该聊天机器人中，即是对与旅行网站上的对用户有用的、用户可能查询的信息进行查询和获取。

聊天机器人的搭建是本系统的核心。主要通过与用户的多轮交流，获取用户的需求，

在数据库中获取对应的处理好的数据，通过前端反馈给用户。

该聊天机器人的业务流程图如图 2.2 所示。

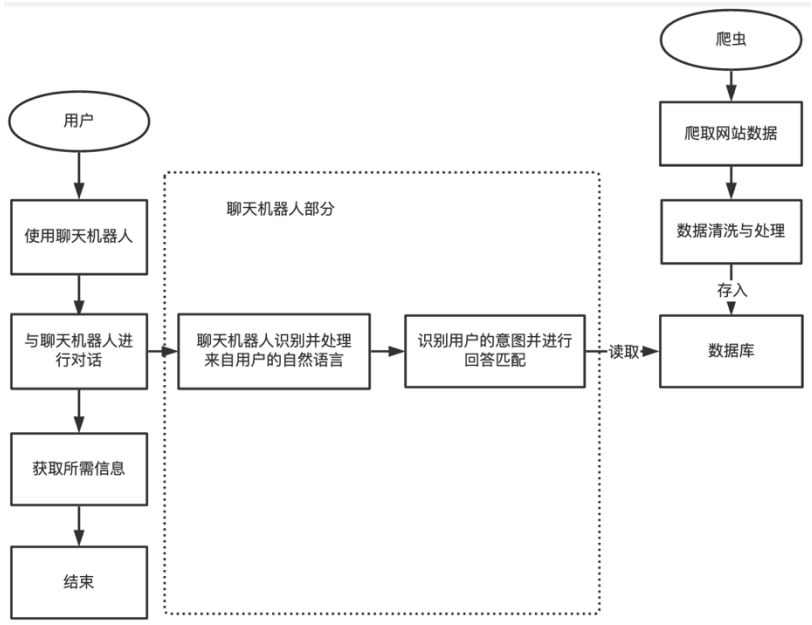

图 2.2 聊天机器人业务流程图

2.2 聊天机器人的功能需求

用户可以通过前端与聊天机器人进行交互式对话。前端页面即制作的聊天窗口，用户可以在聊天窗口中的输入框输入消息与机器人进行对话。机器人识别用户的消息后，经过对消息的处理理解用户意图，从数据库中取出相应的内容作为答案，并进行一定程度的整理，通过前端反馈给用户。

主要功能需求如下：

（1）支持用户询问机器人各种有关于旅游的信息，机器人经过分析及槽值填充判断意图并获取数据反馈给用户。

（2）支持用户与机器人闲聊，能够处理一定量的与旅行信息服务不相关或相关度较低的问题，能够回复一定的闲聊问题。

（3）支持用户查询各城市及景点的热点图，以及通过游记、评论制作的词云。

（4）支持用户存储自己喜欢的景点，并在会话过程中可以读取所有自己喜欢的景点构成一份简单攻略。

（5）支持导航功能，可将用户喜爱的景点制作一份路线图，供用户参考。

3 系统设计

3.1 系统总体规划

本系统旨在实现用户与机器的交互式聊天，机器人为用户提供旅行信息的服务，用户能够高效、便利的整合自己所需的旅行信息，为出行做好更加充分的准备。

本系统需要网络爬虫对网站上的数据进行解析和获取，并对获取后的数据进行分类整理和清洗。对数据进行处理后制作所需图表。之后搭建聊天机器人框架，并针对获取到的数据设计用户意图与聊天故事情境，并对意图与故事进行训练，从而达到机器人能够流畅的回答用户提出的问题并能够与用户进行聊天的功能。

3.2 概要设计

3.2.1 用户功能模块设计

本系统通过读取数据库中保存的数据，为用户提供的功能如图 3.1 所示。

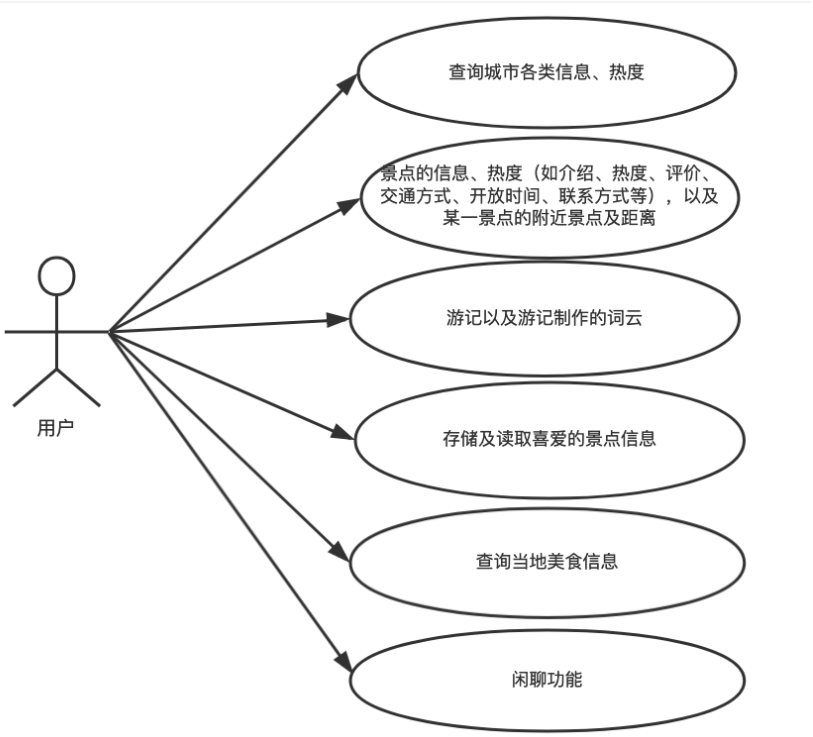

图3.1 用户功能模块图

3.2.2 爬虫功能模块设计

本系统需要通过网络爬虫获取各项数据。爬虫实现的功能如图 3.2 所示。

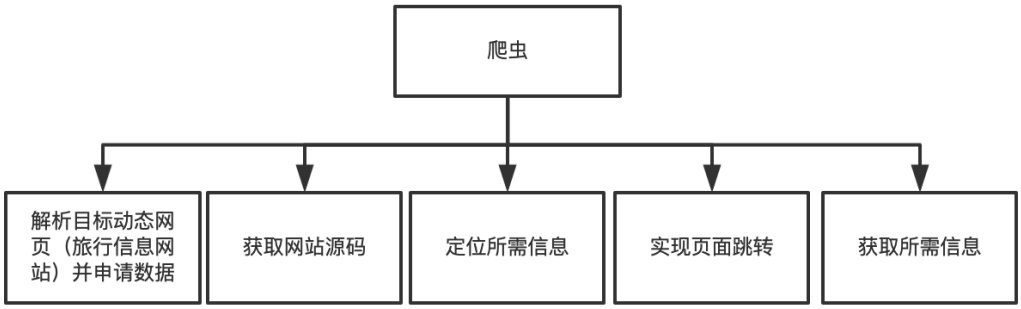

图3.2 爬虫功能模块图

3.3 数据库设计

3.3.1 数据库的使用

由于系统需要，同时避免爬虫运行过程中可能出现的数据库读写错误（如主键为空等），本系统采用 MongoDB 实现数据库的构建。

MongoDB 是一种分布式文件存储的非关系型数据库，是非关系型数据库中功能最丰富、最像关系型数据库的一种数据库。它存储数据非常方便，面向集合存储，易存储对象类型的数据。存储在集合中的文档，被存储为 key-value 的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是复杂的文件类型，如文件、数组、文档数组等。在存入数据时，会自动生成数据类型为 ObjectId 的主键_id，该值唯一但不自增。ObjectId 是一个 12 字节的 BSON 类型（一种 JSON 的扩展）字符串。按照字节顺序，依次代表：4 字节：UNIX 时间戳，3 字节：表示运行 MongoDB 的机器，2 字节：表示生成此_id 的进程，3 字节：由一个随机数开始的计数器生成的值。本系统是基于 python 制作的，对于数据库的写操作，Python 中的字典就可以直接存入数据库，使得对数据库的操作更加方便。

MongoDB 数据库支持在 terminal 中进行操作，官网也支持该数据库的可视化。于是本系统采用数据库可视化的方式，在官网上下载 MongoDB Compass 作为数据可视化工具。能够更加方便的查看保存的数据以及对数据进行操作。

在 MongoDB 数据库中，表名为 Collection，行为 document，字段为 field，索引、数据库的名称与 MySQL 中相同，分别为 index、database。

3.3.2 数据库的设计

（1）本系统使用 MongoDB 数据库，所创建的各表及详细信息如图 3.3 所示

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
city_info	108	705.7 B	74.4 KB	1	36.0 KB	
food_rank	120	105.0 B	12.3 KB	1	36.0 KB	
scenic_info	6,477	1.1 KB	7.2 MB	1	164.0 KB	
travel_notes	503	4.6 KB	2.3 MB	1	44.0 KB	
visit_city_num	126	64.6 B	7.9 KB	1	32.0 KB	

图 3.3 数据库信息

其中，各表具体描述如下：

city_info：用于存放与城市相关的各种信息

food_rank：用于存放各地美食信息与排名

scenic_info：用于存放所有地区所有景点的信息

travel_notes：用于存放各地游记

visit_city_num：用于存放地区旅行人数及热度

该数据库设计 ER 图如图 3.4 所示。

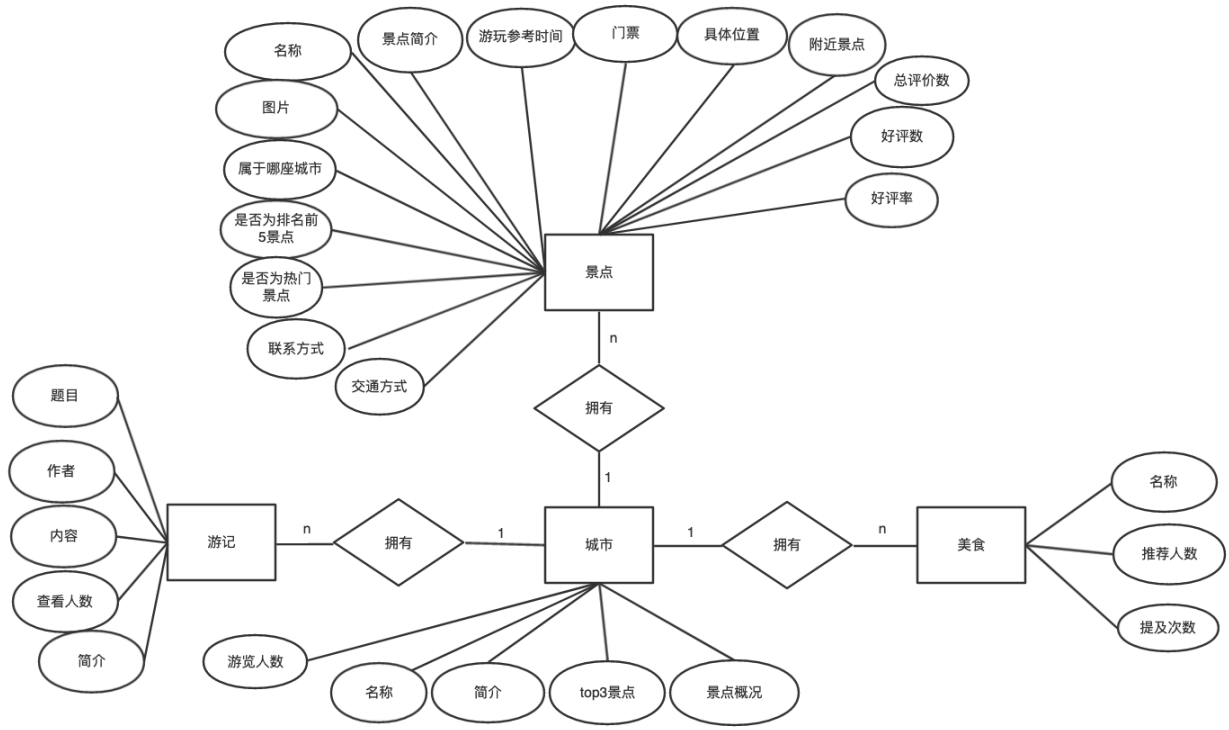

图 3.4 数据库设计 ER 图

（2）城市信息表 city_info(_id, city_name, profile, top1, top2, top3, jingdian_gaikuang)，记录城市的有关信息。具体说明如表 3.5 所示。

表 3.5 城市信息表

序号	字段名称	数据类型	字段说明	主键
1	_id	ObjectId	自动生成标号	√
2	city_name	String	城市名称	
3	profile	String	城市简介	
4	top1	String	城市 top1 景点	
5	top2	String	城市 top2 景点	
6	top3	String	城市 top3 景点	
7	jingdian_gaikuang	String	城市景点概况	

（3）城市信息表 scenic_info(_id, name, img, locate_city, top, hot, profile, time_reference, open_time, tel, traffic, tickets, location, nearby, sum_eva_num, good_eva_num, good_eva_rate)，记录各景点的信息。具体说明如表 3.6 所示。

表 3.6 景点信息表

序号	字段名称	数据类型	字段说明	主键
1	_id	ObjectId	自动生成标号	√
2	name	String	景点名称	
3	img	String	景点参考图片	
4	locate_city	String	景点所属城市	
5	top	Int	景点是否排名前 5	
6	hot	Int	是否为热门景点	
7	profile	String	景点简介	
8	time_reference	String	游玩时间参考	
9	open_time	String	景点开放时间	
10	tel	String	景点联系方式	

11	traffic	String	交通方式	
12	tickets	String	景点门票	
13	location	String	景点位置	
14	nearby	Array	该景点周围景点（每一临近景点(Object类型)包含两个 String 类型的数据，为景点名称(nearby_name)和距该景点距离(nearby_distance)）	
15	sum_eva_num	String	总评价数	
16	good_eva_num	String	好评数	
17	good_eva_rate	String	好评率	

（4）游记内容表 travel_notes(_id, city, title, author, view, profile, content)，用于存放网络爬虫获取的游记信息及内容。具体说明如表 3.7 所示。

表 3.7 游记信息表

序号	字段名称	数据类型	字段说明	主键
1	_id	ObjectId	自动生成标号	√
2	city	String	记录的城市	
3	title	String	游记标题	
4	author	String	游记作者	
5	view	String	浏览量	
6	profile	String	游记简介	
7	content	String	游记内容	

（5）美食排行表 food_rank(_id, city_name, food_num, top(i))，用于存放网络爬虫爬取的美食排行信息。具体说明如表 3.8 所示。

表 3.8 美食信息表

序号	字段名称	数据类型	字段说明	主键
1	_id	ObjectId	自动生成标号	√
2	city_name	String	城市名	

3	food_num	String	美食数量	
4	top(i)	Object	top(i)(0<=i<= int(food_num))表示第 i 种美食，每一种美食中包含三个 String 类型的元素：美食名称(food_name)、游记提及次数(food_travel_notes_like)、推荐次数(food_recommend)	

（6）城市浏览人数表 visit_city_num(_id, city_name, visit_num)，具体说明如表 3.5 所示。

表 3.9 城市浏览人数表

序号	字段名称	数据类型	字段说明	主键
1	_id	ObjectId	自动生成标号	√
2	city_name	String	城市名	
3	visit_num	String	旅行人数	

4 系统实现

4.1 研发工具

本系统是基于 Python 制作的，因此选择 Pycharm 作为系统的开发工具。借助 python 的第三方库。部分核心的配置信息如下：

名称	版本	说明
Python	3.6	
action	1.4.4	用于获取隐藏的下拉菜单
actions	1.0.3	
bar	0.2.1	用于制作图表（柱形图）
parsel	1.5.2	用与将请求后的网页解析成 re,xpath,css 进行内容的定位
pymongo	3.8.0	用于操作 MongoDB 数据库
matplotlib	2.2.5	用于数据处理、统计
jieba	0.42.1	中文分词
wordcloud	1.6.0	制作词云
requests	2.23.0	用于网页请求
rasa	1.9.6	用于构建聊天机器人框架
rasa-core	0.14.5	
rasa-core-sdk	0.14.0	
rasa-nlu	0.15.1	
rasa-sdk	1.9.0	
pip	19.0.3	用于安装 python 所需的第三方库

4.2 关键技术

4.2.1 爬虫相关理论技术

网络爬虫是一种按照一定的规则，自动地获取网站上的信息的脚本。爬虫旨在将目标网页的数据和信息下载到本地，方便之后对于获取到的数据进行处理和分析。爬虫技术可以由多种语言实现，但由于 python 语言较简单，用户可以很容易的使用，同时也具备了很多第三方库以及工具包使得对爬虫的操作更加简便，所以有很多网络爬虫是由 python 语言进行编写。网络爬虫按照应用又可以分为聚焦网络爬虫和通用网络爬虫。前者是一种能够选择性地获取与预先定义好的相关页面的网络爬虫，而后者是一种适用于

搜索引擎搜索广泛的主题的一种爬虫。

Selenium 是一个用于 Web 自动化测试的工具。它可以直接运行在浏览器中，就好像是真正的用户在操作浏览器一样。它可以帮助用户自动按照代码要求完成点击、打开、验证等操作。它几乎支持在目前所有的主流浏览器中的测试。对于爬虫程序而言，selenium 可以帮助用户解决 Web 中的 Js 渲染问题。即部分网站需要通过 Js 渲染数据，如果直接获取网站的源码，则无法得到渲染的数据。而 selenium 就可以帮助用户获取这种动态网页的数据。

Parsel 模块是由 Scrapy 出品的，一种将请求到的网页解析成 re、xpath、css 进行内容的匹配的一个模块。但是 Scrapy 将 parsel 独立出来做了一个第三方模块。因此，在不使用 Scrapy 框架的情况下，依旧可以使用 parsel 模块获取数据。parsel 模块使得对数据的解析变得非常方便且高效。selenium 也可以通过 xpath 对按钮、内容等进行定位，但不利于对大量网页进行操作，自身的局限性较高。如某一网页中含有的按钮在另一相似度较高的网页中无法通过相同的 xpath 对其定位，就会导致程序报错或抛出异常。但是 parsel 模块通过 xpath 进行定位，若无法定位则返回空，不影响程序的正常运行。

基于此，本系统采用 Python+Selenium+Parsel 构建了一个以旅游信息为目的的聚焦型网络爬虫。

4.2.2 聊天机器人相关理论技术

聊天机器人又称语音助手、聊天助手等，是一种自动问答系统。其核心是对自然语言的处理。聊天机器人按照功能可以分为问答型聊天机器人、任务型聊天机器人和闲聊型聊天机器人，按照模式可以分为基于检索模式和生成式模式的聊天机器人。聊天机器人的基本实现过程为：用户输入数据，分析用户的意图，获取关键的参数，匹配最佳的回答，并将回答反馈给用户。

本系统旨在搭建任务型聊天机器人。任务型聊天机器人是以完成用户某项任务为目标的聊天机器人。在与用户对话的过程中，有时用户的需求较为复杂，在一次的交谈中无法获取用户所需内容的所有信息，因此无法给出完整的答复。所以此时需要机器人对用户的需求进行一轮或多轮的询问，以获取所有必要信息从而给出相应的回答。这就需要聊天机器人能够实现多轮的对话并且能从上下文中获取必要的信息。

本系统采用 Rasa 框架搭建所需的任务型聊天机器人。Rasa 是一个开源的机器学习的框架，用于构建上下文 AI 助手和聊天机器人。它的核心模块为 NLU 与 Core。NLU 为自然语言理解（Natural Language Understanding），主要实现对于用户输入的数据的意图的理解，以及获取对话中包含的必要信息（槽值），能够将用户的输入转换为结构化的数据[4]。Core 模块是一个对话管理模块，用于预测下一步应该做什么。对话管理是指根据对话历史信息来决定此刻对用户的反应。Rasa 的工作原理如图 4.1 所示。

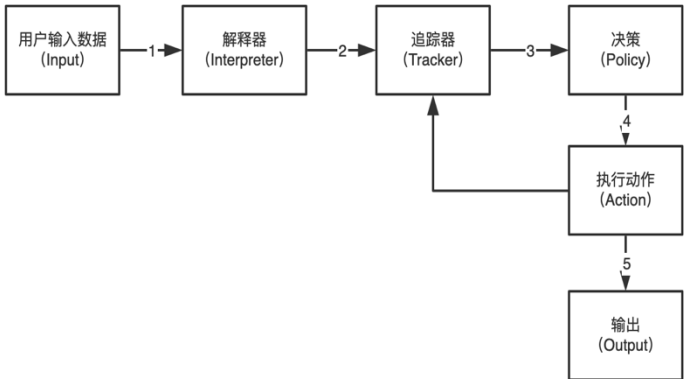

图4.1 RASA 工作原理

对于图 4.1 中各个标注过程的解释如下：

（1）用户输入文字，送入解释器（Interpreter），即 Rasa NLU 模块。该模块主要用于识别意图“intent”以及所包含的实体“entities”。

（2）解释器（Interpreter）给出结果，并将所识别的意图和实体以及文本信息“text”传递给 Tracker 模块。该模块主要作用是跟踪对话状态。

（3）Tracker 跟踪对话状态，为 Policy 提供用户意图、上一步系统动作、实体（槽值）的 Embedding（低维向量，使距离相近的向量对应的物体有相近的含义）。

（4）Policy 给出决策并选择执行相应的 Action。

（5）执行 Action 后将执行结果反馈给用户。

该框架的优点是与传统 RNN 构成的聊天机器人相比，代码量较少，不需要写复杂逻辑，能够快速实现，快速验证。缺点是需要写很多的 stories，同时需要准备大量的数据。

Rasa Core 是整个聊天机器人的核心，主要的任务是维护更新对话状态和动作选择，然后对用户的输入作出响应。对话状态是包含可能影响下一步决策的信息的一种聊天数据的特征。动作选择是指选择接下来应该做的动作。Rasa Core 包含了交流所必需的 stories，即用户与机器人之间的对话情境。Core model 以训练 stories 的形式从真实的对话数据中学习。这也体现了机器学习的一个优点，即机器人不知道应该做什么的时候，可以用一种方式教它。Rasa Core 还包含了 domain，定义了助手所处的领域，即它应该获得用户的输入，应该能够进行的预测的操作，以及如何响应和要储存的信息等。在 domain 领域中主要包含如下 5 个内容：意图（intents），即机器人拥有识别哪些意图的能力；动作（actions、forms），即机器人可以做的事情和做法；回复（response/templates），用作机器人回复的模版字符串；会话配置（session_config），用于配制一次对话的时间限制；实体/槽值（entities/slots），实体即文本中的关键字，机器人识别后填充到槽值中，用于后续的操作。

该框架还有一个必要的配置文件，其中包含了框架所需的管道（pipeline），以及决策方式（Policy）。Pipeline 是 Rasa NLU 的一部分，本系统中使用的 pipeline 如下：

（1）MitieNLP：一种自然语言处理工具。用于初始化 mitie 结构。mitie 需要一个

语言模型(.dat)，且必须在 configs.yml 配置中指定。

（2）JiebaTokenizer：分词器。使用 Jieba 对中文进行分词。

（3）MitieEntityExtractor：实体抽取器。比较适合训练自定义的实体。

（4）EntitySynonymMapper：实体同义词映射。确保将训练数据中的同义词映射到同一个值。

（5）MitieFeaturizer：为意图分类创建特征。

（6）SklearnIntentClassifier：意图分类器，需要首先配制 Featurizer，为意图创建分类特征。

Policy 的作用就是使用合适的策略来预测对话后要执行的动作。原理是衡量命中的 Policy 哪些置信度比较高，置信度高的 Policy 选择动作执行。在本系统中，所使用的 Policy 如下：

（1）KerasPolicy: 策略是应用 Keras 框架实现的神经网络来选择下一个要执行的动作，其框架默认使用 LSTM(长短期记忆网络)算法。

（2）FallbackPolicy：如果识别的用户输入意图置信度过低，则会通过该策略使机器人执行指定的动作，在本系统中是自动回复的动作。

（3）MemoizationPolicy：只对训练数据中的对话有记忆，若训练数据中存在，则置信度为 1，预测下一个动作。否则为 0。

（4）FormPolicy：处理表单动作的填充。当一个表单动作被调用的时候，该策略会持续预测表单动作，直到表单中的所有槽值都被填充，然后再选择对应的动作执行。

4.3 聊天机器人的实现

4.3.1 爬虫功能实现

本系统爬取了几个旅行信息网站的信息构建的数据库。本文以其中一个网站为例，讲解爬虫技术的实现。

该网站的部分数据是由 Js 进行渲染的，因此如果直接请求页面信息则只能获取到部分数据，系统所需要的关键数据无法获取。因此，本系统采用了 selenium 模拟浏览器进行操作。之后使用 parsel 解析并通过各元素的 xpath 定位所需要的数据，将数据以 key-value 的形式存入字典，再以字典的形式存入 MongoDB 非关系型数据库中，方便之后的使用。

在网络爬虫的运行过程中，主要爬取的信息参见之前的数据库表。爬取该网站的函数、功能及实现过程如下所示：

（1）jingdian_basicinfo 函数：用于获取城市名称、简介、top3 的景点、景点概况以及各景点的信息。实现过程如图 4.2 所示。

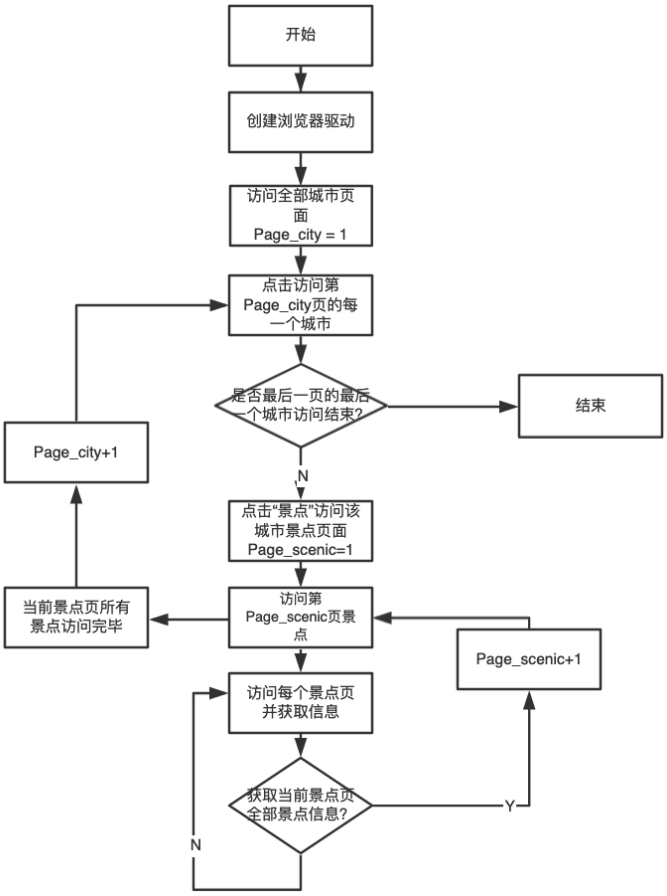

图 4.2 jingdian_basicinfo 函数实现流程

（2）save_travelnotes 函数：用于获取各个城市的游记。由于游记数量较多，所以在该函数中设置过滤条件，如选取浏览量相对较高的游记，每一个城市在保证包含所有可能条件（如日期、出行天数、花销等）的情况下选取一定数量的游记等。在爬虫执行过程中，除了会将游记题目、概况、内容等存入数据库中，还会在目录下构建以游记题目为名称的文件夹，将游记正文内容存入做备份，同时选取游记中的部分图片保存下来。该爬虫的实现过程如图 4.3 所示。

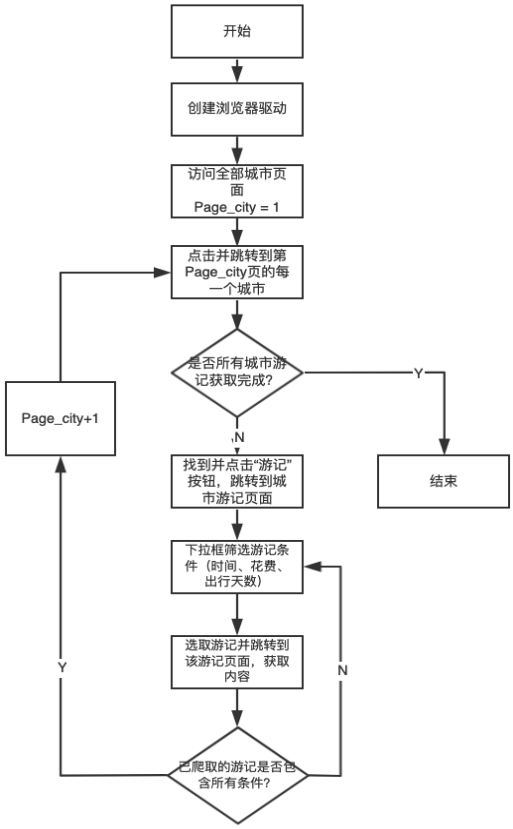

图 4.3 save_travelnotes 函数实现流程

（3）food_rank 函数：用于获取各个城市的美食排行榜。由于在该网站上有专门的一栏为美食，所以只需要获取每个城市该模块的内容即可。包括美食名称，美食的游记提及次数以及美食的推荐人数等。具体实现过程如图 4.4 所示。

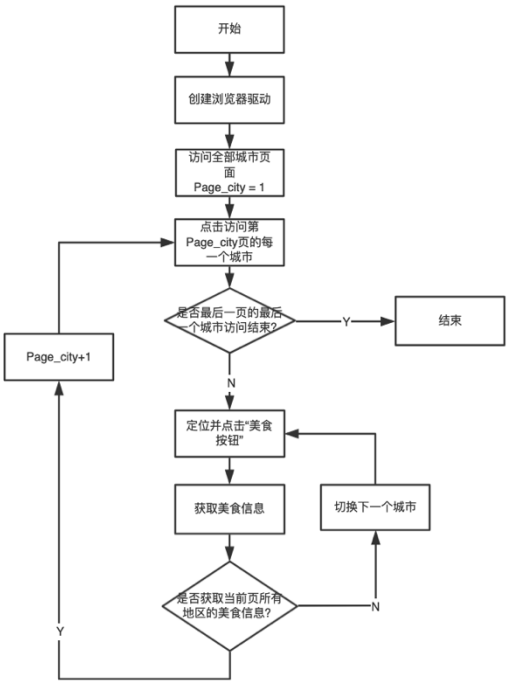

图 4.4 food_rank 函数实现流程

（4）save_visit_city_num 函数：用于保存城市游览人数，并将获取到的数据存入

到数据库中，方便之后的数据处理及向用户推荐旅游城市。

4.3.2 数据处理模块实现

由于本系统需要为用户提供旅行信息的服务，所以能够让用户更加直观的感受出各个城市的一些信息的对比与差异，本系统根据收集到的部分数据制作出不同的统计图。同时为了让一些用户更加直观的查看某一城市的游记或评价等，本系统根据不同的游记制作了不同的词云供用户参考。基于此，用户可以更加直观的了解城市的各种信息。

统计图的制作依赖于 python 的第三方库 matplotlib，使用其中的 bar 模块，构建柱形统计图。所制作的统计图主要包括：各个城市的景点热度（排名前 40），城市的美食排行表，近期各地区浏览人数统计图（排名前 20）。主要流程如下：从数据库的对应 Collection 中获取相应的数据，再对数据进行排序，利用 matplotlib.pyplot.bar 的函数构建柱形图，设置标题、图例、横纵坐标、柱状图颜色、柱状图标注等参数，最后将柱状图保存。若对于各个城市进行统计，则需要加一个循环。以各个城市的景点热度为例，核心实现代码如下所示：

```
db = MongoDB('city_info')
result1 = db.selectdata()
for r in result1:
    city_name.append(r['city_name'])
# Collection: scenic_info 城市名称 + 景点名称 + 评价总数 + 好评率
for flag in range(0,len(self.list)):
    self.list[flag]['sum_eva_num'] = self.list[flag]['sum_eva_num'].replace('（共有', '')
    self.list[flag]['sum_eva_num'] = self.list[flag]['sum_eva_num'].replace('条真实评价）', '')
    if self.list[flag]['sum_eva_num']== '':
        self.list[flag]['sum_eva_num'] = '0'
self.list.sort(key=lambda x: int(x['sum_eva_num']), reverse=True)
for i in range(0,len(city_name)):
    scenic_name = []
    eva_num = []
    good_rate = []
    good_rate_num = []
    #数据处理
    for r in self.list:
        if r['locate_city'] == city_name[i]:
            try:
                x = r['good_eva_rate']
            except:
                continue
            scenic_name.append(r['name'])
            str_temp = r['sum_eva_num']
```

```
            str_temp = str_temp.replace('（共有', '')
            str_temp = str_temp.replace('条真实评价）', '')
            if str_temp == '':
                str_temp = '0'
            int_eva_num = int(str_temp)
            eva_num.append(int_eva_num)
            if r['sum_eva_num'] == '0' or r['good_eva_num'] == '':
                good_rate.append('0%')
            else:
                good_rate.append(r['good_eva_rate'])
            if r['sum_eva_num'] != '0':
                good_rate_str = r['good_eva_rate']
                good_rate_str = good_rate_str.replace('%', '')
                good_rate_num.append(int(float(float(good_rate_str)/100) * int_eva_num))
            else:
                good_rate_num.append(0)
        if len(scenic_name) == 0:
            continue
        plt.figure(figsize=(14, 10))
        if len(scenic_name) > 40:
            num = plt.bar(np.arange(len(scenic_name[:40])), eva_num[:40], label='各景点评价数及好评率', tick_label=scenic_name[:40])
            num1 = plt.bar(np.arange(len(scenic_name[:40])), good_rate_num[:40], facecolor='yellow', edgecolor='none')
        else:
            num = plt.bar(np.arange(len(scenic_name)), eva_num, label='各景点评价数及好评率',
                          tick_label=scenic_name)
            num1 = plt.bar(np.arange(len(scenic_name)), good_rate_num, facecolor='yellow',
                           edgecolor='none')
        plt.legend((num, num1), ("评价总数", "好评率"), bbox_to_anchor=(0.5, 1.02), frameon=False,
                   loc='upper right', ncol=2)
        plt.tick_params(labelsize=10)
        plt.tick_params(axis='x', labelsize=8)
        plt.title(city_name[i]+' 景点热度排名', fontsize=20)
        plt.xlabel(u'景点名称', fontsize=14) # 设置 x 轴，并设定字号大小
        plt.ylabel(u'热度', fontsize=14) # 设置 y 轴，并设定字号大小
        plt.xticks(rotation=90)
        plt.rcParams['font.sans-serif'] = 'Arial Unicode MS'
        self.add_labels(num)
```

```
    self.add_labels_rate(num1, eva_num)
    plt.savefig('./' + city_name[i] + '景点热度.jpg')
```

实现结果如 4.5 所示。

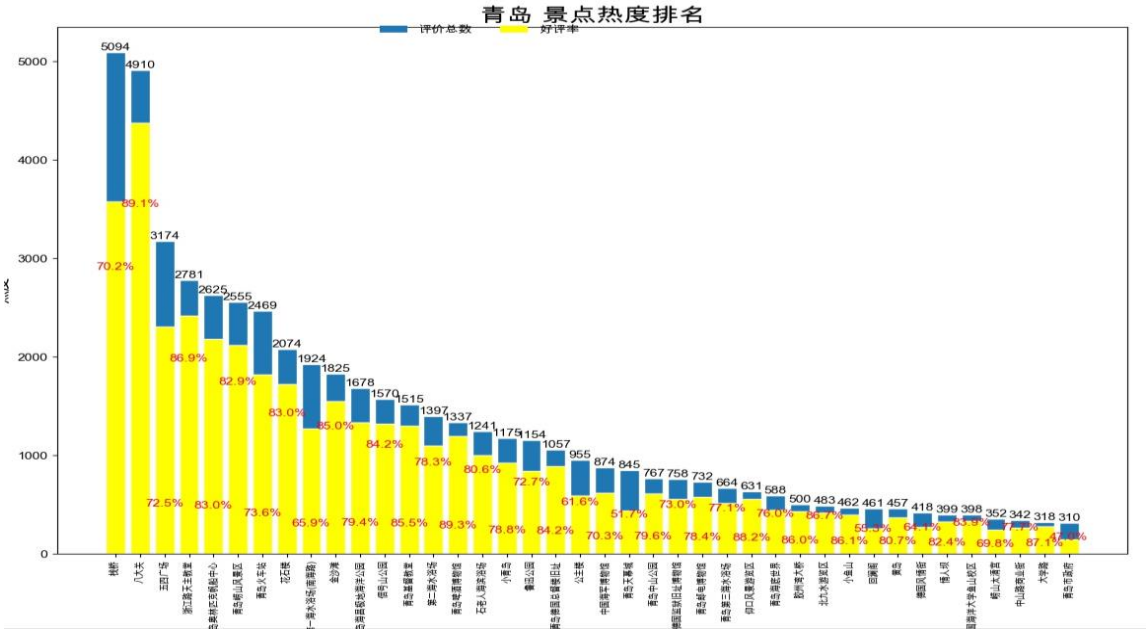

图 4.5 景点热度统计图实现

词云的制作依赖于 python 的第三方库 wordcloud。同时需要使用到 jieba 库对游记及评价进行分词。还需要通过 wordcloud 库中导入 STOPWORDS，用于去掉停用词。基于此，从数据库中获取游记，对于每一份游记，制作出一份词云供用户参考，并将制作好的词云图按照城市分类保存在以城市名命名的文件夹中。词云的核心代码如下：

```
elf.mkdir('./wordcloud')
for i in range(0, len(self.list)):
    self.mkdir('./wordcloud/'+self.list[i]['city']) # 用于构建文件夹
    city_name.append(self.list[i]['city'])
     if self.list[i]['content'] == '':
          continue
     travel_notes.append(self.list[i]['content'])
     wordlist = jieba.cut(self.list[i]['content'], cut_all=True)
     wl = " ".join(wordlist)
     wordcloud = WordCloud(
     background_color='white',
     max_words=200,
     stopwords=STOPWORDS,
     max_font_size=300,
     font_path='/Users/user/PycharmProjects/chattbot/simhei.ttf', # 设置字体，路径在电脑内
     width=1600,
```

```
 height=1000,
 random_state=30, # 设置有多少种随机生成状态
).generate(wl)
plt.imshow(wordcloud)
 plt.axis("off")
 plt.savefig('./wordcloud/' + city_name[i] + '/' + city_name[i] + str(i) + '_词.jpg')
```

词云的两种具体实现如图4.6和4.7所示。

图 4.6 词云实现（分词）

图 4.7 词云实现（不分词）

4.3.3 聊天机器人框架的搭建与实现

本系统采用 Rasa 框架搭建聊天机器人。在实现方面，该框架所包含的文件夹及文件夹中所包含的文件如下所示：

（1）configs：用于存放 config.yml，credentials.yml，domain.yml 以及 endpoints.yml 四个配置文件。其中 credentials.yml 用于配置连接到其他服务的信息。endpoints.yml 用于指定 action 的 url，即使 rasa 能够连接到其他的 web。

（2）data：用于存放本系统所需的训练数据，包括 nlu.json，nlu.md，stories.md 文件，以及配置 MitieNLP 所必需的 total_word_feature_extractor_zh.dat 文件。

（3）model：用于存放本系统经过对 nlu 以及 stories 训练过后的数据文件。该文件在训练过后自动生成。

（4）actions：该文件包含 action.py 文件，用于书写系统所需要的各个动作的具体实现。

（5）server： 用于存放本系统的前端接口调用及可视化实现。

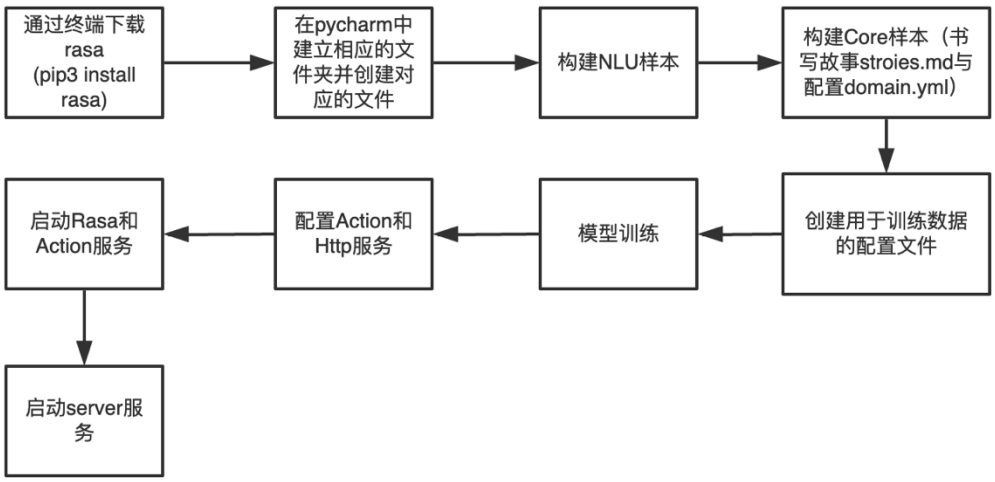

图4.8 Rasa 框架搭建流程图

该框架搭建及实现的具体流程如图 4.8 所示。

如上图所示，本系统在实现的过程中，需要构建 NLU 样本。Rasa 框架提供了两种 NLU 模型训练样本数据的格式，即 Markdown（.md 文件）和 JSON。Markdown 和 Json 文件相比，Json 的文件可读性较差，但 Markdown 文件构建起来有些麻烦。所以本系统在 NLU 样本的构建方面，采用 Markdown 和 JSON 文件结合的方式。

在本系统中，Markdown 文件主要包含 Common examples 和 synonyms。前者表示对意图、文本和实体的表示和标注，以“##”开头表示意图名称，以“-”开头表示文本，以”[entity_text](entity_name)”的方式来对实体进行标注。后者是用来标注一些同义词，即在实体识别的时候可以被识别为同一意思。

在本系统 NLU 样本中使用的 JSON 格式的文件，是通过 Chatito 生成工具进行生成的。在 Chatito 中，与构建 Markdown 文件类似，需要写明意图（%[intent]）、同义词（~[synonyms]）、实体（@[entity]）、训练次数等信息，但不同的是在 Chatito 生成 JSON 文件时，会将每一意图下的文本及其同义词进行全排列，从而在训练次数内生成更多的 NLU 样本数据，使得 NLU 样本数据变得更加丰富。通过 Chatito 生成数据后的 JSON 文件格式如图 4.9 所示。

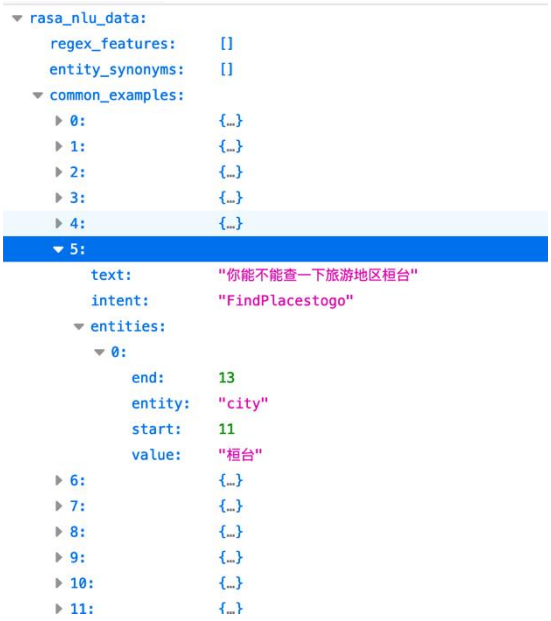

图4.9 生成的JSON文件

Rasa Core 模块作为整个聊天机器人的核心，需要为其配置所属领域以及训练数据的形式——故事。本系统是以为用户提供旅行信息服务为目的，因此所属领域中所设置的意图、实体、动作、表单动作、自动回复等均是为机器人能够有效的回复给用户准确的信息而设置的。domain 配置文件中部分配置信息如图 4.10 所示。

图4.10 domain文件部分配置信息

Stories 用于训练 Rasa 的对话管理模型，是用户和聊天机器人之间的对话情景。将用户输入表示为特定的意图，机器人对用户的意图作出判断并给出相应的操作名称。主要包含三部分：用户输入（如*greet）、动作（如- utter_answer_greet）、表单动作(如- form{“name”: “askscenic”})。由于一个聊天机器人的构建需要丰富的故事情境的编写，且编写故事的过程中涉及的槽值填充、表单提交/注销、事件的响应等较为复杂，所以如果人工编写故事数据很容易出现考虑不周或者出错的情况。这就极大的提高了构建训练模型的难度。为解决此困难，Rasa Core 提供了交互式学习的方式供使用者训练故事数据。即使用者与机器人进行交谈，为机器人提供反馈，使得机器人知道某种意图应该作何反应。本系统构建故事训练数据采用交互式学习的方式，具体流程如图 4.11 所示，系统提供的实现过程的可视化界面如图 4.12 所示。

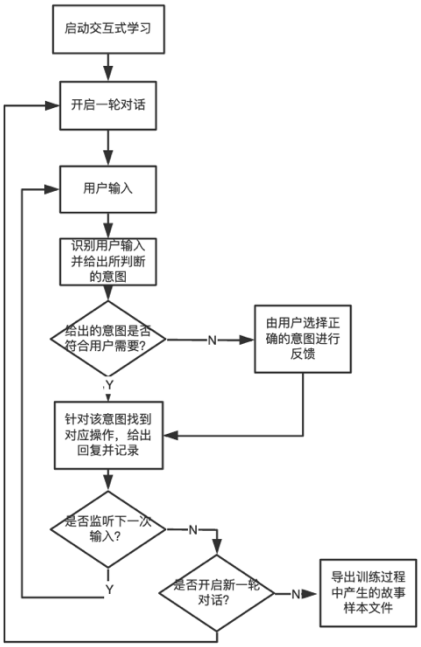

图 4.11 交互学习流程图

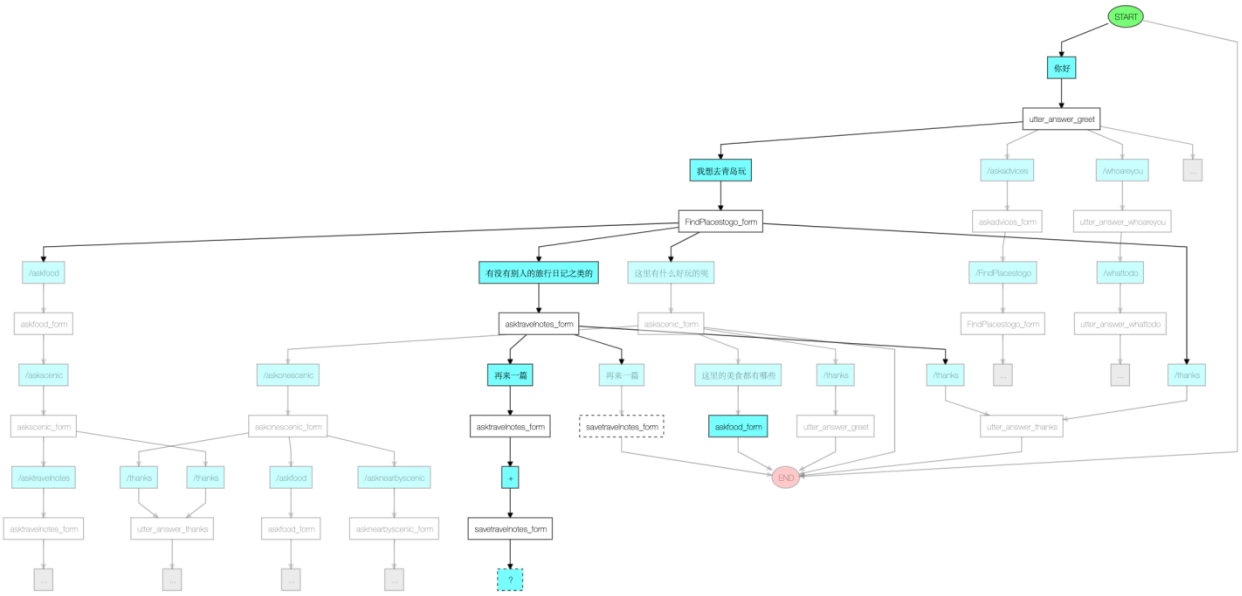

图 4.12 交互式训练可视化界面

聊天机器人的动作模块，是聊天机器人对用户的输入进行响应的模块。在本系统中，主要包含两种动作：用于发送特定的消息给用户的动作和编写代码对用户意图进行回复的动作。对于 domain 中配置的表单动作，在 Action 文件中需要给出具体的实现。本文主要包含的需要编写代码进行回复的表单动作如下所示：

（1）FindPlacestogo：当用户意图为想要寻找一个地方去旅行的时候，系统识别该意图并获取用户输入中的 city 槽值，通过该类在数据库中查找该城市的信息，回馈给用户。若用户输入中不存在 city 槽值，则系统会使用发送特定消息的动作(utter_ask_city)询问用户想要出行的城市。

（2）AskscenicForm：当用户意图为想了解某一城市的景点的信息时，系统识别该意图并获取用户输入中的 item_scenic 槽值，通过该槽值查找对应信息回馈给用户。

（3）AskTravelnotesForm：当用户意图为想要查看游记时，系统识别该意图并获取用户输入中的 item_travelnotes 槽值，在数据库中选取游记反馈给用户。同时支持再次查询，若用户不满意当前游记，可选择更换游记浏览，同样使用该表单完成。

（4）AskFoodForm：当用户意图为想要了解该城市美食时，系统识别该意图并获取 item_food 槽值，并根据先前获得的 city 槽值，反馈给用户该城市的美食排行图。

（5）AskOnescenicForm：当用户意图为想要了解某一景点时，系统识别该意图并获取 one_scenic 槽值，即需填充景点名称，系统查询数据库并给出相应回复。

（6）AskNearbyscenic：当用户意图为想要了解某一景点的附近景点时，系统识别该意图并获取 nearby 槽值，根据先前获取的 one_scenic 槽值的值，在数据库中查询对应景点的附近景点，给出用户回复附近景点及距离。

（7）ActionDefaultFallback：用于默认回复的动作。若用户输入为机器人无法识别的内容，则调用该类进行回复。该类调用图灵机器人 api，需要使用外网。通过用户的

输入获取回复的内容。该 api 每日的使用次数有限制。

（8）AskAdvices：当用户意图为寻求建议时，系统识别 advice 槽值并给出相应回复。

（9）SaveTravelnotes：当用户意图为想要存储某一景点时，系统识别 symbol 槽值并将当前景点的重要信息添加进文件。若当前景点为第一个则创建文件并存储。

（10）Gettravelnotes：当用户想要查询自己储存过的景点时，系统识别并将用户收藏文件以文字的形式发送给用户，并配以导航图。导航图的制作即将用户所有保存的景点，通过调用百度 api 制作一份地图，图中包含到达各个景点的路线。以用户存储的第一个景点为开始，以最后一个景点为结束。

其中一个表单动作的实现代码如下所示。

```
class FindPlacestogoForm(FormAction):
    def name(self) -> Text: # 定义表单动作
        return "FindPlacestogo_form"
    @staticmethod
    def required_slots(tracker: Tracker) -> List[Text]: # 所需要填充的槽值
        return ["city"]
    def slot_mappings(self) -> Dict[Text, Union[Dict, List[Dict]]]:
        return {
            "city": self.from_entity(entity="city")
        }
    def submit(
        self,
        dispatcher: CollectingDispatcher, # 用来将消息发送给用户
        tracker: Tracker, # 当前用户的状态记录。可用来获取槽值。
        domain: Dict[Text, Any], # 机器的 domain
    ) -> List[Dict]:
        city = tracker.get_slot('city')
        print(city)
        mongodb = MongoDB('city_info')
        dic = {'city_name': city}
        result = mongodb.select_one_data(dic)
        mongodb1 = MongoDB('visit_city_num')
        dic = {'city_name': city}
        num = mongodb1.select_one_data(dic)
        print(result)
        if len(result) != 0:
            str = ""
            str += '我们一起了解一下' + city + "吧！ " + result['profile']
            dispatcher.utter_message(text=str)
```

```
        str = ""
        if len(num) != 0:
            str += '这个地方有很多人呢！据不完全统计，今年一共有' + num['visit_num'] + '人次去游览过呢！'
            dispatcher.utter_message(text=str)
        str = ""
        str += "这个地方最有名的景点莫过于：" + result['top1'] + "、" + result['top2'] + "、" + result['top3'] + "。"
        str += result['jingdian_gaikuang']
        dispatcher.utter_message(text=str)
    return [SlotSet("city", city)]
```

4.3.4 系统前端的实现

5 系统测试

在该系统爬虫、数据分析、聊天机器人的搭建以及前端的实现结束之后，对本系统的核心——聊天机器人作出一次测试。测试结果如下。

（1）对于聊天机器人核心功能的测试

该聊天机器人的核心即是对用户查询的旅行信息做出响应且给出较为准确的答复。答复的内容来自于数据库存取的网站数据。在测试的过程中，需要在终端通过指令“python -m rasa run --port 5005 --endpoints configs/endpoints.yml --credentials configs/credentials.yml —debug”启动 Rasa 服务，实现自然语言理解和对话管理的功能。启动成功，结果如图 5.1 所示。

图 5.1 启动 Rasa 服务结果图

还需要在终端启动 Action 服务，通过输入指令“python -m rasa run actions --port 5055 --actions actions —debug”完成。结果如图 5.2 所示。

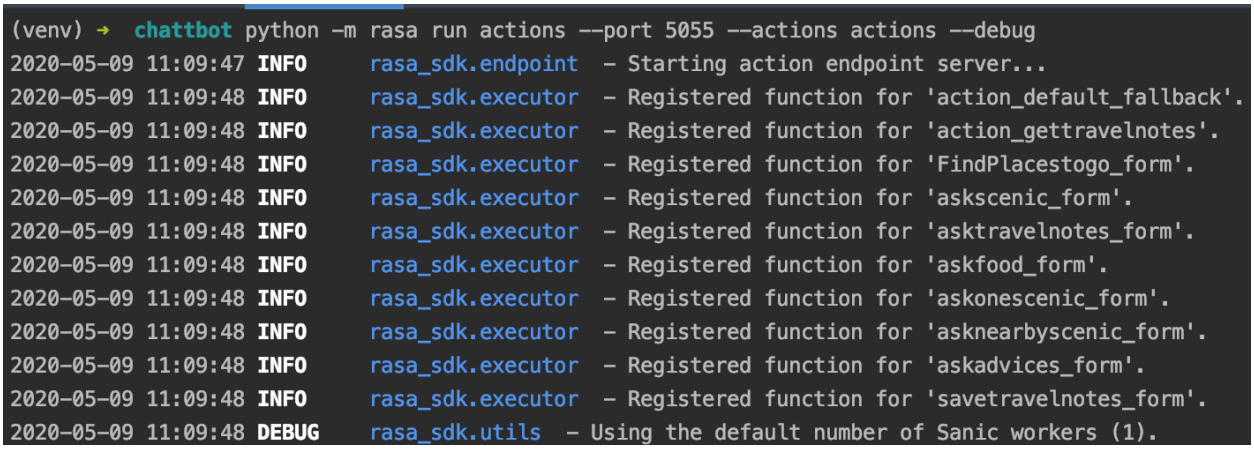

图 5.2 启动 Action 结果图

本次以设计的一个短的对话情境，来对该聊天机器人的文字回复、图片回复、回复内容、回复速度等做一次测试。在对话情境内，机器人作出的回应如图 5.3 所示，测试过程中，终端自然语言理解与对话管理模块执行内容如图 5.4 所示。Action 的执行内容如图 5.5 所示。

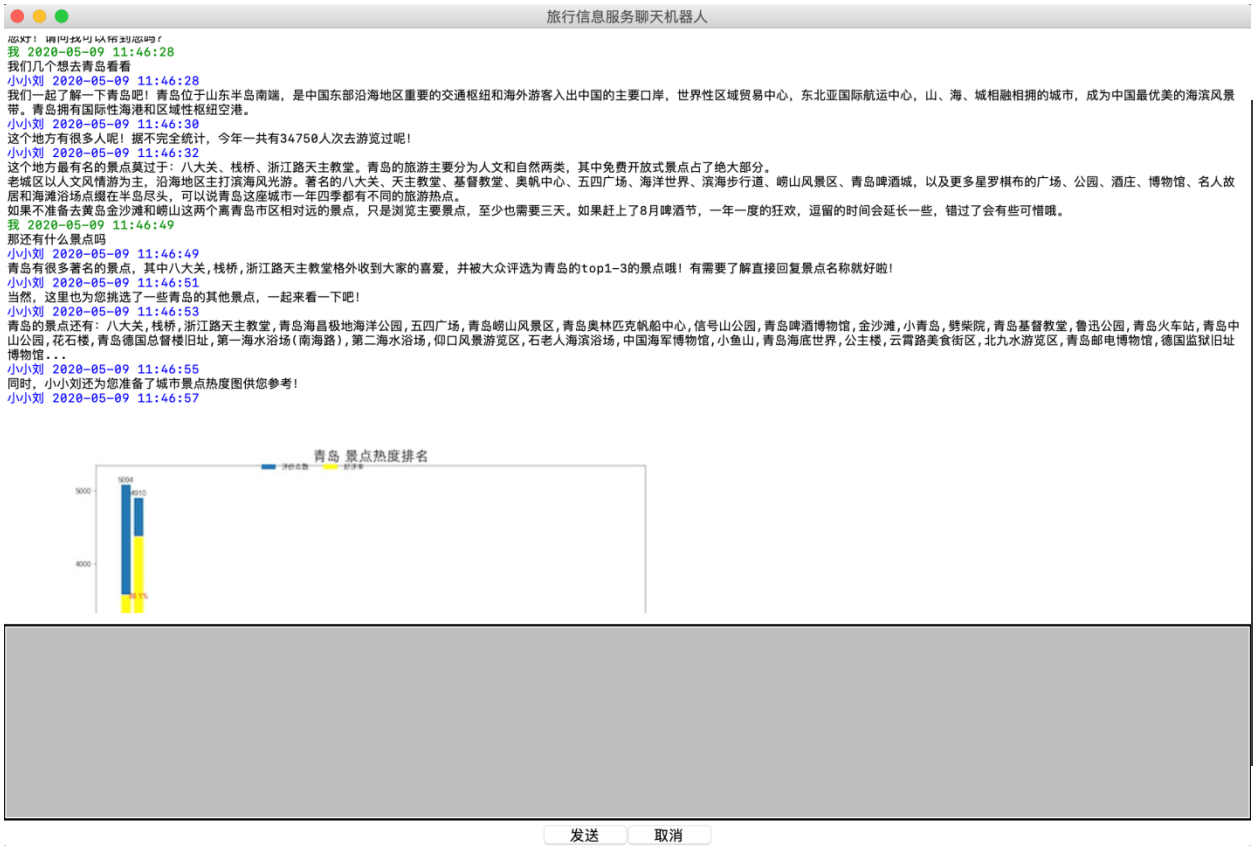

图 5.3 聊天机器人部分核心功能测试

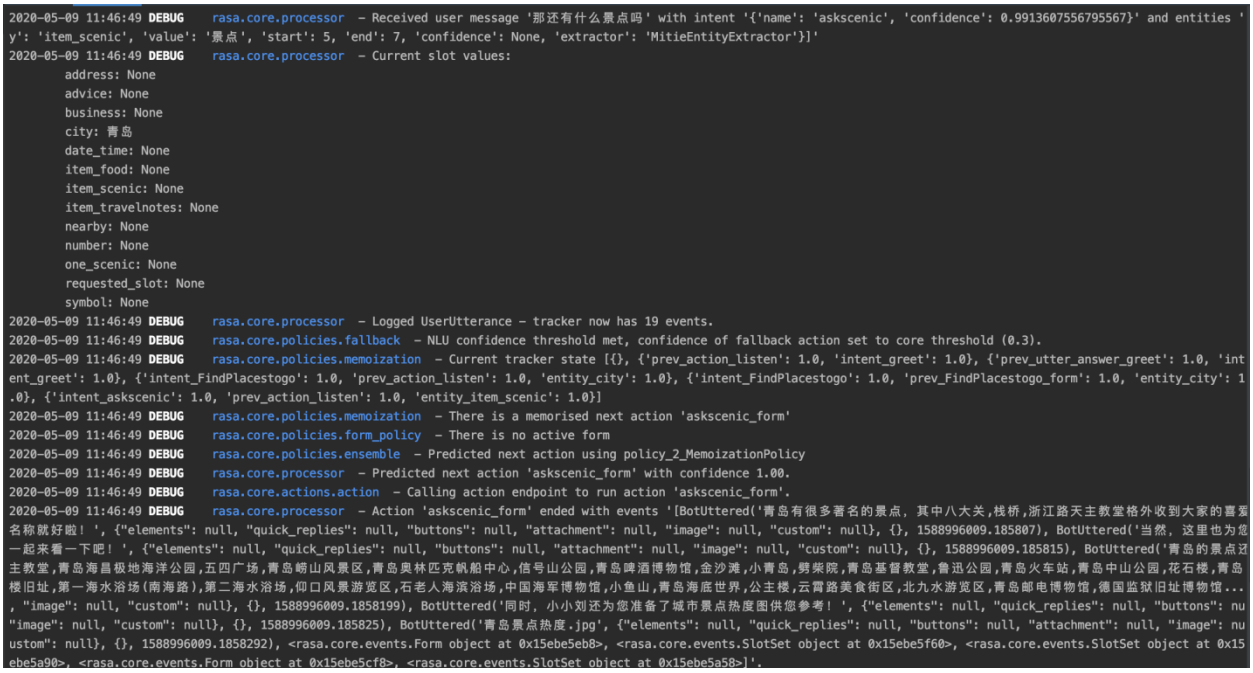

图 5.4 自然语言理解与对话模块部分内容

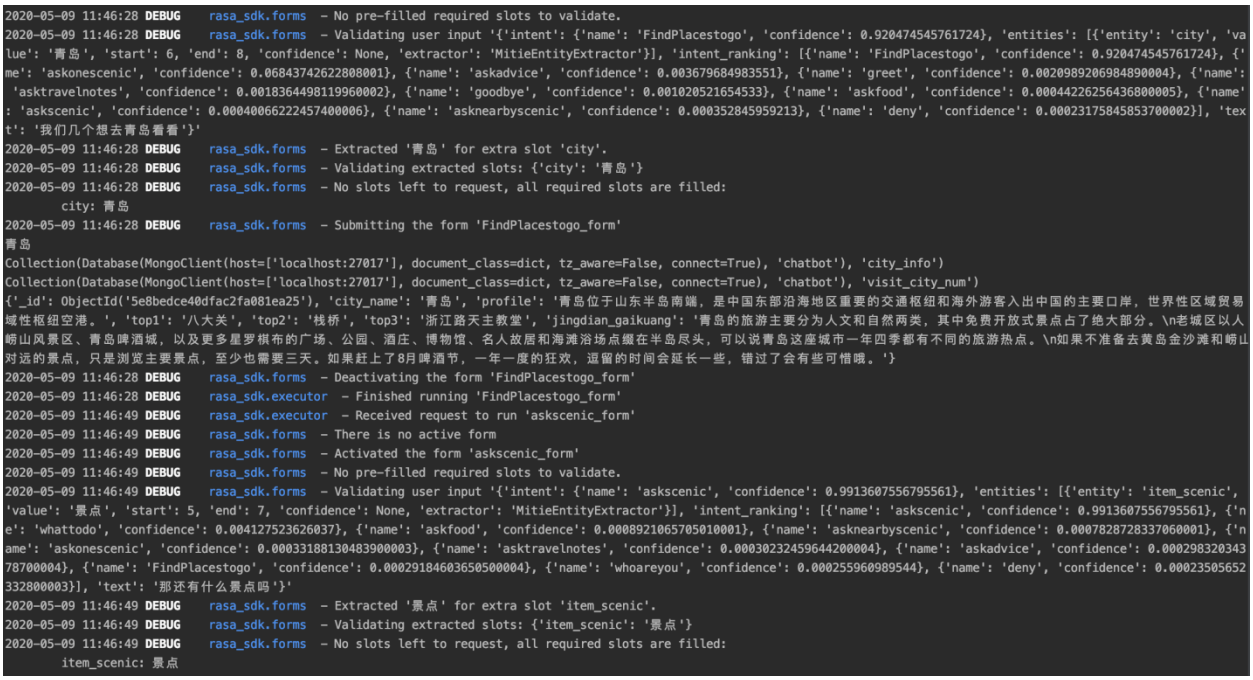

图 5.5 Action 所执行的部分内容

由图 5.3 可以看出，程序可以正常运行。聊天机器人可以正常地且较为准确地回复用户所需要查询的内容，并且在需要为用户提供后台数据处理所做的图时，可以正常的显示。对于回复速度，机器人能够按照后台设置的每两秒发送一条消息的速度为用户做出回复。

由图 5.4 和图 5.5 可以看出，该聊天机器人能够识别用户输入中的槽值，并对槽值进行填充，且基本能够正确识别用户意图，通过 Action 对用户输入作出正确的回复。

（2）对于聊天机器人附加功能的测试

该聊天机器人主要解决用户对于旅行信息方面存在的问题。对于其他方面的聊天，包括部分机器人可以回复的闲聊内容，测试结果如图 5.6 所示。

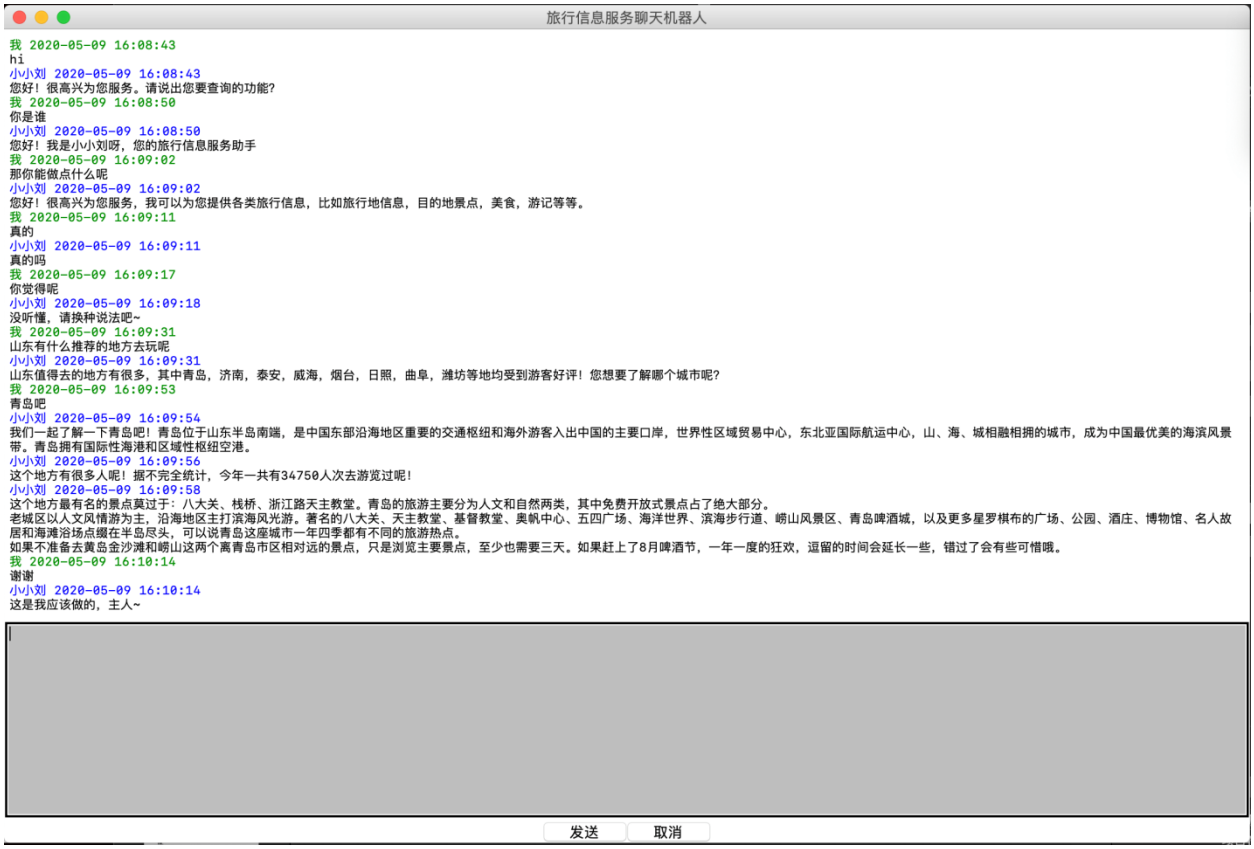

图 5.6 聊天机器人其他功能测试

6 总结与展望

6.1 开发过程收获

在整个开发过程中，知识上的收获包括学习到了 Python 的基本用法、用 Python 做一个网络爬虫来获取系统所需要的信息、如何使用第三方库来构建统计图等、如何搭建一个聊天机器人的 Rasa 框架、如何制作一个简单的 GUI 以及如何使用一种非关系型数据库 MongoDB。在之前的学习当中，并没有学习到 Python 的有关知识。这一次做这样一个聊天机器人，也是对自己的一个挑战。在这个过程中，遇到了很多的问题，通过在网上查阅相关的资料，一步一步将问题解决。也让我从最开始对机器学习一点都不了解，变得初步了解一些东西并且有继续学习的兴趣和想法。

在开发之前的规划中，通过查阅有关资料发现很多的聊天机器人都有着不同的分类，以及相应的实现方法。其中比较多的涉及到用循环神经网络来制作聊天机器人。通过对循环神经网络 RNN 的学习，也收获了一些知识。了解到了其工作原理，但是对于机器学习没有过多了解和系统学习的自己而言，使用循环神经网络搭建一个聊天机器人显得有些困难。于是通过查阅各种资料，才发现了这个 Rasa 框架，并通过学习和一步步实践，让我完成此次系统的开发。

6.2 后续工作设想

在该聊天机器人完成的阶段，也发现了该聊天机器人有些许的不足。这样的第一代产品，对于其数据库的内容的构建，还可以在后续的工作中持续扩大。并在多搜集、多获取数据的同时，更新掉一些过时的数据，用最新的数据训练聊天机器人。同时，该聊天机器人的故事数据相对来讲能够完成一些普通的对话，能够基本满足用户查询信息、聊天的一些需求。但是，在之后的工作中，还可以为该机器人扩展更加丰富的故事情境，能够让聊天机器人在更加复杂的对话环境中更加精准的作出回复。而且可以为该机器人扩展更丰富的语料和语料库，让该聊天机器人不仅仅局限于查询旅行信息，可以实现为用户进行规划、订票等实际操作，满足用户更多的需求。

对于该聊天机器人的前端，本系统只是做了一个简单的前端。在之后的工作中可以根据需要，制作出更加符合用户体验的前端，或者依托于其他 APP，来与用户进行沟通和对话，亦或是搭建一个服务器，能够让该聊天机器人服务于更多的人。