

科研总结报告

——智能聊天机器人的制作

李享

北京理工大学

2019. 2. 13

目录

- 1. 研究背景2
- 2. 项目内容3
 - 2.1 实现功能3
 - 2.2 技术要点3
- 3. 具体实现4
 - 3.1 识别用户意图.....4
 - 3.2 建立状态机和策.....6
 - 3.3 设置回复语句的模板.....8
 - 3.4 提取命名实体.....9
 - 3.5 设计本地存储的数据.....9
 - 3.6 数据库查询10
 - 3.7 地铁路径规划.....11
 - 3.8 公交路径规划.....12
 - 3.9 部署至微信平台.....13
- 4. 结果展示14

1. 研究背景

处理人类所使用的自然语言具有较大的难度，这主要是因为人类语言的语义较为模糊。同样的一个词、一句话很可能具有多种不同的理解方法，并可能与上下文相关联，甚至有些表达会很隐晦，这都不便于计算机的处理。

20 世纪 60 年代，全球第一个聊天机器人 ELIZA 诞生，其使得计算机首次可以与人用英语进行交谈。在随后的数十年中，随着人工智能的迅速发展，聊天机器人也变得愈发智能化。如今，其不仅能以自然语言的形式与人进行交互，对于语义的理解也更加准确，并可被应用于更广泛的领域之中。在 Facebook、微信等即时通信平台上，聊天机器人已经得到了广泛的应用。其可为人们提供客户服务、推送新闻、播报天气等诸多服务，为人类提供了更加便利的交互方式。

2. 项目内容

实现了一个智能聊天机器人，用户可以使用自然语言作为输入，查询北京地铁的相关信息并进行路径规划。

2.1 实现功能

- 1) 理解用户输入语句的意图，并据此给予相应的回复
- 2) 可以问好并介绍本聊天机器人的功能
- 3) 针对用户的提问，给出对应地铁线路的首末班车时间和站点信息
- 4) 针对用户指定的特定地铁站，给出该站的线路信息
- 5) 通过用户输入的相关语句，判断用户提出的问题是否得到了解决
- 6) 可以进行路径规划，给出两站点间的最优换乘方案
- 7) 调用相关 API，为用户提供公交车的换乘方案

2.2 技术要点

- 1) 基于 Rasa NLU 构建训练数据，实现对自然语言的意图识别
- 2) 利用关键词提取识别用户意图

- 3) 利用正则表达式进行命名实体识别
- 4) 构建数据库及对应的查询机制
- 5) 对同一个问题提供多种选择性的回答
- 6) 建立状态机并实现多轮多次的查询技术，可以基于当前语境提供回答
- 7) 使用广度优先搜索，给出两节点间的最短路径
- 8) 调用百度地图 API，实现信息的实时查询
- 9) 将实现的聊天机器人部署至微信上

3. 具体实现

全部代码可大致划分为以下九部分：

- 1) 识别用户意图
- 2) 建立状态机和策略
- 3) 设置回复语句的模板
- 4) 提取命名实体
- 5) 设计本地存储的数据
- 6) 数据库查询
- 7) 地铁路径规划
- 8) 公交路径规划
- 9) 部署至微信平台

3.1 识别用户意图

首先确定出几种潜在的用户意图：

greeting: 问候

function: 询问机器人的功能

time_table: 查询某条地铁线路的时刻表

line_info: 查询某条地铁线路的站点信息

station_info: 查询某地铁站的线路信息

route_planning: 规划地铁换乘路线

specify: 在选择上述 4 种功能的情况下，用户又提供了更确切的信息（例如线路编号、地铁站名称、规划路径的始末点）

solved: 问题已得到解决

bus: 规划公交换乘路线

对于每种意图，都构建若干的训练数据。例如“hello”、“good morning”等语句的意图都被归为 greeting。

```
{
  "text": "hey",
  "intent": "greeting",
  "entities": []
},
{
  "text": "hello",
  "intent": "greeting",
  "entities": []
},
{
  "text": "hi",
  "intent": "greeting",
  "entities": []
},
{
  "text": "good morning",
  "intent": "greeting",
  "entities": []
},
{
  "text": "good afternoon",
  "intent": "greeting",
  "entities": []
},
```

利用 RASA NLU 训练上述模型。

```

from rasa_nlu.training_data import load_data
from rasa_nlu.config import RasaNLUModelConfig
from rasa_nlu.model import Trainer
from rasa_nlu import config

# Create a trainer
trainer = Trainer(config.load("config_spacy.yml"))

# Load the training data
training_data = load_data("training_data.json")

# Create an interpreter by training the model
interpreter = trainer.train(training_data)

```

至此，机器人便具备了识别用户意图的能力。其可计算出当前语句属于每种意图的可能性，并选择可能性最高的一项作为该语句的意图。例如对于输入“*What can you do?*”，机器人可正确识别该句话意在询问其功能。

```

{'intent': {'name': 'function', 'confidence': 0.3940570437413696},
 'entities': [],
 'intent_ranking': [{'name': 'function', 'confidence': 0.3940570437413696},
 ,
 {'name': 'solved', 'confidence': 0.20337820149088714},
 {'name': 'route_planning', 'confidence': 0.12599530219099178},
 {'name': 'line_info', 'confidence': 0.1007942012940104},
 {'name': 'station_info', 'confidence': 0.0842878122001784},
 {'name': 'time_table', 'confidence': 0.06251378601316959},
 {'name': 'greeting', 'confidence': 0.028973653069393358}],
 'text': 'What can you do?'}

```

3.2 建立状态机和策略

先对所有的可能状态进行定义，为每个状态分配一个一位数的编码。

```

INITIAL = 0
TIME_TABLE = 1
LINE_INFO = 2
STATION_INFO = 3
ROUTE_PLANNING = 4
TIME_TABLE_DONE = 5
LINE_INFO_DONE = 6
STATION_INFO_DONE = 7
ROUTE_PLANNING_DONE = 8
SOLVED = 9

```

其中状态 1-4 表示用户已选定了机器人的 4 种主要功能之一，但尚未输入完整的查询所需信息。若在此状态下，用户补充输入了相关的信息（即该条消息的意图为 *specify*），状态编号将会加上 4，转化为 5-8 中的一个，表示用户通过两轮的问询已完成了该种功能的使用。这一过程对应了如下的策略。

```
(TIME_TABLE, "specify"): (TIME_TABLE_DONE, 15),
(LINE_INFO, "specify"): (LINE_INFO_DONE, 16),
(STATION_INFO, "specify"): (STATION_INFO_DONE, 17),
(ROUTE_PLANNING, "specify"): (ROUTE_PLANNING_DONE, 18),
```

```
entities = pattern.findall(message)
if(intent in ["time_table", "line_info", "station_info", "route_planning"] and len(entities) > 0):
    state = state + 4
    response_NO = response_NO + 4
```

上图中两位数的编码代表着机器人的不同回复模式。

同时，用户也可在一条消息中同时确定选择何种功能并给出相应的信息，如“Do you know the train schedule of Line 9?”中用户表明希望获知某条地铁线路的时刻表，并指定了这条线路就是9号线。在此情形下，状态将不经过1-4，而直接跳转至5-8中的对应项。

继续完成剩余的策略编写，使得机器人可以实现在各状态间的合理转换。

```
# Define the policy rules
policy = {
    (INITIAL, "greeting"): (INITIAL, 10),
    (INITIAL, "function"): (INITIAL, 19),
    (INITIAL, "time_table"): (TIME_TABLE, 11),
    (INITIAL, "line_info"): (LINE_INFO, 12),
    (INITIAL, "station_info"): (STATION_INFO, 13),
    (INITIAL, "route_planning"): (ROUTE_PLANNING, 14),

    (SOLVED, "greeting"): (INITIAL, 10),
    (SOLVED, "function"): (INITIAL, 19),
    (SOLVED, "time_table"): (TIME_TABLE, 11),
    (SOLVED, "line_info"): (LINE_INFO, 12),
    (SOLVED, "station_info"): (STATION_INFO, 13),
    (SOLVED, "route_planning"): (ROUTE_PLANNING, 14),

    (TIME_TABLE, "specify"): (TIME_TABLE_DONE, 15),
    (LINE_INFO, "specify"): (LINE_INFO_DONE, 16),
    (STATION_INFO, "specify"): (STATION_INFO_DONE, 17),
    (ROUTE_PLANNING, "specify"): (ROUTE_PLANNING_DONE, 18),

    (TIME_TABLE_DONE, "time_table"): (TIME_TABLE, 11),
    (TIME_TABLE_DONE, "line_info"): (LINE_INFO, 12),
    (TIME_TABLE_DONE, "station_info"): (STATION_INFO, 13),
    (TIME_TABLE_DONE, "route_planning"): (ROUTE_PLANNING, 14),
    (TIME_TABLE_DONE, "solved"): (SOLVED, 20),

    (LINE_INFO_DONE, "time_table"): (TIME_TABLE, 11),
    (LINE_INFO_DONE, "line_info"): (LINE_INFO, 12),
    (LINE_INFO_DONE, "station_info"): (STATION_INFO, 13),
    (LINE_INFO_DONE, "route_planning"): (ROUTE_PLANNING, 14),
    (LINE_INFO_DONE, "solved"): (SOLVED, 20),
```



```

(STATION_INFO_DONE, "time_table"): (TIME_TABLE, 11),
(STATION_INFO_DONE, "line_info"): (LINE_INFO, 12),
(STATION_INFO_DONE, "station_info"): (STATION_INFO, 13),
(STATION_INFO_DONE, "route_planning"): (ROUTE_PLANNING, 14),
(STATION_INFO_DONE, "solved"): (SOLVED, 20),

(ROUTE_PLANNING_DONE, "time_table"): (TIME_TABLE, 11),
(ROUTE_PLANNING_DONE, "line_info"): (LINE_INFO, 12),
(ROUTE_PLANNING_DONE, "station_info"): (STATION_INFO, 13),
(ROUTE_PLANNING_DONE, "route_planning"): (ROUTE_PLANNING, 14),
(ROUTE_PLANNING_DONE, "solved"): (SOLVED, 20),
(ROUTE_PLANNING_DONE, "bus"): (ROUTE_PLANNING_DONE, 21)
}

```

3.3 设置回复语句的模板

为方便能对同一个问题可以有多种选择性的回答，对每种类型的回复进行了两位数的编码。

以不需要填充参数的回复为例，先为每一种类别的回复设定多个回答语句。

```

# Import the random reply module
responses = {
    10: [
        "Hello~",
        "Hi, may I help you?"
    ],
    11: [
        "So, which line's timetable do you want to know?",
        "You need to tell me the specific line.",
        "The schedule of which line?"
    ],
    12: [
        "About which line?",
        "Okay, then I need to know the line number."
    ],
    13: [
        "About which station?",
        "You need to tell me the name of the station."
    ],
    14: [
        "I need to know the starting point and the destination.",
        "What's your current location and where do you want to go?"
    ],
    19: [
        "I know everything about Beijing Subway. Besides, you can ask me to plan a trip"
    ],
    20: [
        "I'm glad I solved your question."
        "Is there any other questions?."
    ]
}

```

在生成具体回答时引入了随机机制，可令机器人的回答更符合人类语言特点。

```

# Generate the response
if response_NO in [10, 11, 12, 13, 14, 19, 20]:
    response = random.choice(responses[response_NO])

```


而对于诸如地铁线路信息的回复则需要填充参数。该类回复中既包含了用户提问时使用的关键词（entities），也包含通过搜索、计算所得的结果（result）。

```
if response_NO == 16:
    result = search_line_info(entities[0])
    response = 'The route of Line {}: \n{}'.format(str(entities[0]), str(result[0][0]))
```

3.4 提取命名实体

机器人需要能够识别并提取出用户输入中的一些特定实体，如“海淀黄庄”（地铁站名）、“FangShan”（地铁线路名）、“Line 13”（地铁线路名）。

为此，先要编写对应的正则表达式，使机器人可识别数字、汉字和特定的线路名称。

```
import re
pattern = re.compile('\d+|BaTong|Airport|ChangPing|YanFang|YiZhuang|XiJiao|FangShan|[\u4e00-\u9fa5]+')
pattern_line_only = re.compile('\d+|BaTong|ChangPing|YanFang|YiZhuang|XiJiao|FangShan')
pattern_bus = re.compile('bus|Bus')
```

在消息中匹配编写好的正则表达式，提取出对应的命名实体。

```
entities = pattern.findall(message)
```

3.5 设计本地存储的数据

一些地铁相关的数据需要在本地存储。对于线路信息和站点信息，分别使用一张数据表进行存储。而路径规划所需的数据则用字典类型表示。

数据表 table_lines 用于存储各个地铁线路的信息，包含以下属性：

name：线路名称（如“6”和“ChangPing”）

direction_A：上行运行方向的始末站（如“天通苑北-->宋家庄”）

direction_B：下行运行方向的始末站

running_time_A：上行运行方向的首末班车时间（如“5:10-23:30”）

running_time_B：下行运行方向的首末班车时间

stations：本线路的所有运行站点（如“巴沟-颐和园西门-茶棚-万安-植物园-香山”）

数据表 `table_stations` 用于存储所有地铁站的信息，包含 `name` 和 `lines` 来两个属性，分别表示站名（如“安河桥北”）和经过该站的线路（如“Line 4”）。

同时使用字典 `beijing_subway` 存储各地铁站间的连接关系。将地铁站的名称作为 `key`，每个 `key` 对应的 `value` 同样也是一个字典类型。`value` 字典中的项数等于与当前地铁站相连的其他地铁站的数目。

如下图中的“朱辛庄”站，其通过昌平线可与“巩华城”和“生命科学园”两站相连，通过 8 号线与“育知路”站相连，故该 `key-value` 对应被设置为：

“朱辛庄”：{“巩华城”：“Line ChangPing”，“生命科学园”：“Line ChangPing”，“育知路”：“Line 8”}



3.6 数据库查询

对于时刻表、站点信息和线路信息的查询都通过 SQL 语句完成。先构建查询语句 `query` 确定要查询的表名、要选择的属性和选择条件。再连接数据库、初始化光标，并将 `query` 与参数 `t`（即提取出的命名实体）拼接成完整的查询语句。最后执行该语句，返回符合的结果。

下图以查询站点信息为例。

```
def search_station_info(station_name):
    query = 'SELECT lines FROM table_stations WHERE name=?'
    conn = sqlite3.connect("beijing_subway.db")
    c = conn.cursor()
    t = (station_name,)
    c.execute(query, t)
    return c.fetchall()
```

3.7 地铁路径规划

为获得两节点间的最短路径，此处采用了广度优先搜索，保证可以找到发出地至目的地间最少站数的路径。

```
# BFS, plan a subway route
def plan_path(starting_point, destination):
    if starting_point == destination:
        return [starting_point]
    explored = set()
    queue = [ [starting_point] ]
    while queue:
        path = queue.pop(0)
        s = path[-1]
        for state, action in beijing_subway[s].items():
            if state not in explored:
                explored.add(state)
                path2 = path + [action, state]
                if state == destination:
                    return path2
                else: queue.append(path2)
    return []
```

但此时的输出为 list 类型，内容的表示方式并不符合正常人的习惯。例如，从“香山”到“东直门”站的换乘方案结果为：

```
['香山', 'Line XiJiao', '植物园', 'Line XiJiao', '万安', 'Line XiJiao', '茶棚', 'Line XiJiao', '颐和园西门', 'Line XiJiao', '巴沟', 'Line 10', '苏州街', 'Line 10', '海淀黄庄', 'Line 10', '知春里', 'Line 10', '知春路', 'Line 13', '大钟寺', 'Line 13', '西直门', 'Line 2', '积水潭', 'Line 2', '鼓楼大街', 'Line 2', '安定门', 'Line 2', '雍和宫', 'Line 2', '东直门']
```

因此该结果还需经过进一步改写，才能作为机器人的输出。

在下面的函数中，站名和其所属线路被分类成两个 list，然后获取在换乘方案中出现过的每条线路的最后出现位置，据此将换乘方案改写为字典类型，以方便最终的输出。

```
# Iterate the subway route
def organize_path(result):
    print(result)
    elements_station = []
    elements_line = []
    i = 0
    while i < len(result):
        if i % 2 == 0:
            elements_station.append(result[i])
        else:
            elements_line.append(result[i])
        i += 1
```

```

index_of_line = {}
i = 0
while i < len(elements_line):
    index_of_line[elements_line[i]] = i
    i += 1

path_output = {}
i = 0
for line, index in index_of_line.items():
    stations_per_line = []
    while i <= index + 1:
        stations_per_line.append(elements_station[i])
        i += 1
    i -= 1
    path_output[line] = stations_per_line

return path_output

```

经过改写后的换乘方案：

```

{'Line XiJiao': ['香山', '植物园', '万安', '茶棚', '颐和园西门', '巴沟'], 'Line 10': ['巴沟', '苏州街', '海淀黄庄', '知春里', '知春路'], 'Line 13': ['知春路', '大钟寺', '西直门'], 'Line 2': ['西直门', '积水潭', '鼓楼大街', '安定门', '雍和宫', '东直门']}

```

最终的输出结果：

```

BOT : Here is the shortest path:
Line XiJiao: 香山->植物园->万安->茶棚->颐和园西门->巴沟
Line 10: 巴沟->苏州街->海淀黄庄->知春里->知春路
Line 13: 知春路->大钟寺->西直门
Line 2: 西直门->积水潭->鼓楼大街->安定门->雍和宫->东直门

```

3.8 公交路径规划

为获得仅乘坐公交车的换乘方案，需要调用百度地图 API。因为在该 API 中，路径规划所需的输入参数为起点和终点的经纬度，故要先使用地理编码服务将地址解析为对应的位置坐标。

参照服务文档，设置请求参数，经两次 GET 请求分别获得起点和终点的经纬度坐标，并调整其格式以满足路径规划服务的参数要求。

```

# Acquire the GPS coordinate
url = 'http://api.map.baidu.com/geocoder/v2/'
params_starting_point = {
    'address': starting_point,
    'city': '北京',
    'output': 'json',
    'ak': '2b4f75ed3baea23d71dc1828331f8dfc'
}

params_destination = {
    'address': destination,
    'city': '北京',
    'output': 'json',
    'ak': '2b4f75ed3baea23d71dc1828331f8dfc'
}

```

```

r = requests.get(url,params_starting_point)
r_js = r.json()
longitude_starting_point = r_js['result']['location']['lng']
latitude_starting_point = r_js['result']['location']['lat']
r = requests.get(url,params_destination)
r_js = r.json()
longitude_destination = r_js['result']['location']['lng']
latitude_destination = r_js['result']['location']['lat']

location_starting_point = str(latitude_starting_point)[0:9] + ',' + str(longitude_starting_point)[0:10]
location_destination = str(latitude_destination)[0:9] + ',' + str(longitude_destination)[0:10]

```

调用路径规划服务以获取换乘方案，并调整输出格式。

```

# Acquire the bus route
url = 'http://api.map.baidu.com/direction/v2/transit'
params = {
    'origin': location_starting_point,
    'destination': location_destination,
    'tactics_incity': '3',
    'output': 'json',
    'ak': '2b4f75ed3baea23d71dc1828331f8dfc'
}
r = requests.get(url,params)
r_js = r.json()
steps = r_js['result']['routes'][0]['steps']
i = 0

# Generate the reply
response = ''
while i < len(steps):
    response = response + 'Step {}: '.format(i + 1) + str(steps[i][0]['instructions'])
    if(i != len(steps) - 1):
        response = response + '\n'
    i += 1
return response

```

3.9 部署至微信平台

利用 wxpy 可将完成的聊天机器人部署至微信上。如此一来，用户就可以使用微信直接与机器人进行对话。

```

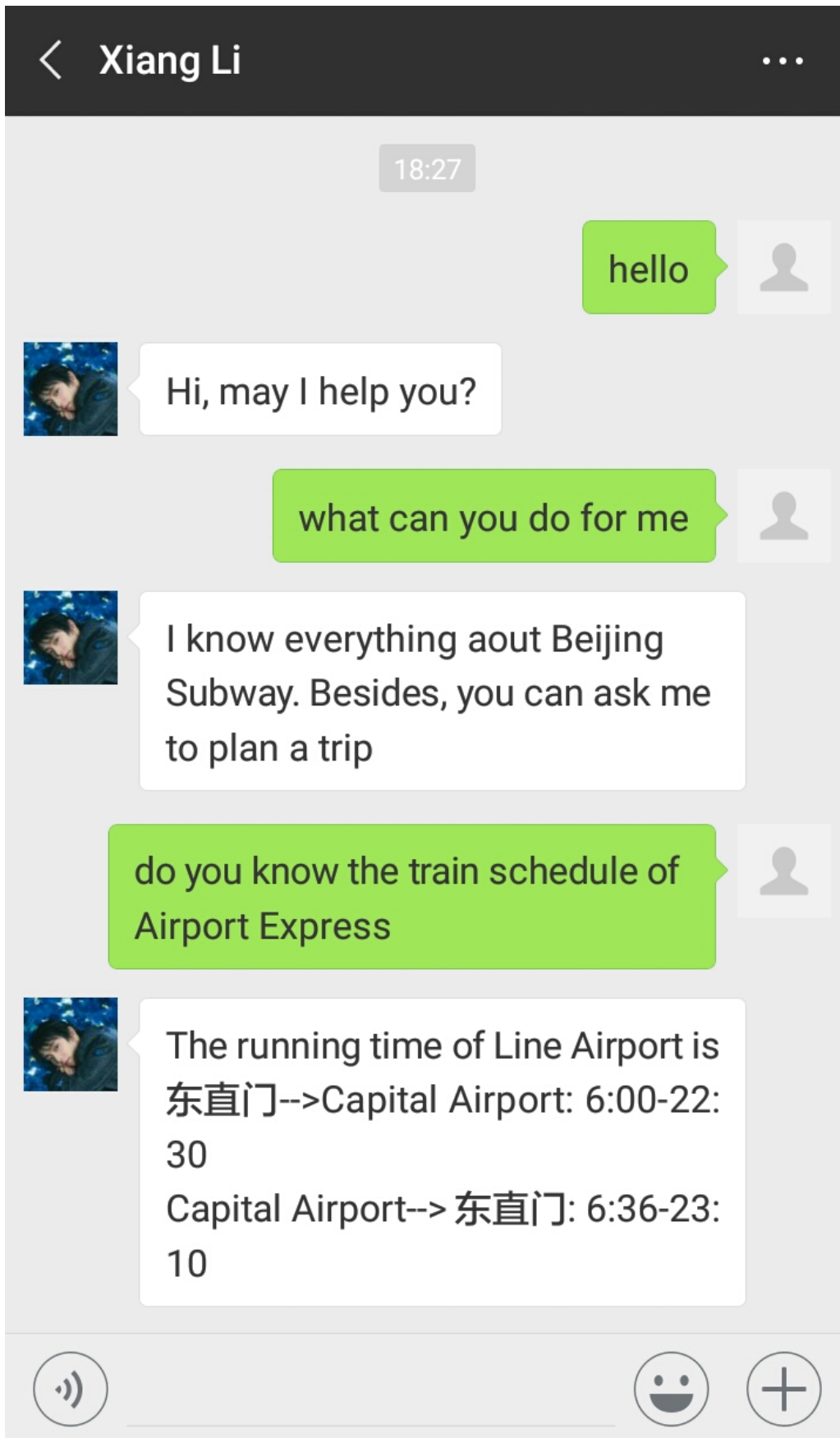
# Initialize the wechat robot
from wxpy import *
bot = Bot()

# Search for wechat friend
from wxpy import *
my_friend = bot.friends().search('Xiang Li', province='北京')[0]

# Return the response on wechat
@bot.register(my_friend)
def reply_my_friend(msg):
    return respond(policy, msg.text)

```


4. 结果展示



I want to know the timetable



So, which line's timetable do you want to know?

uh...line 10?



The running time of Line 10 is
苏州街--> 苏州街(counter-clockwise): 6:19-22:32
巴沟--> 巴沟(clockwise): 4:54-22:31

great



I'm glad I solved your question.

tell me all stations of line 16



The route of Line 16:
西苑-农大南路-马连洼-西北旺-永丰
南-永丰-屯佃-稻香湖路-温阳路-北
安河

which line can I take from 西直门



You can take Line 2 & Line 4 &
Line 13 at 西直门

can you plan a route for me



What's your current location and
where do you want to go?

from 良乡大学城北 to 安定门





Here is the shortest path:

Line FangShan: 良乡大学城北站-> 广阳城-> 篱笆房-> 长阳-> 稻田-> 大葆台-> 郭公庄

Line 9: 郭公庄-> 丰台科技园-> 科怡路-> 丰台南路-> 丰台东大街-> 七里庄-> 六里桥-> 六里桥东-> 北京西站-> 军事博物馆-> 白堆子-> 白石桥南-> 国家图书馆

Line 4: 国家图书馆-> 动物园-> 西直门

Line 2: 西直门-> 积水潭-> 鼓楼大街-> 安定门

what if I just want to travel by bus?



Step 1: 步行 149 米

Step 2: 地铁良乡大学城西站乘 895 路(南纬路方向)经过 11 站到南纬路站

Step 3: 步行 314 米

Step 4: 天桥站乘特 11 路(时代庄园北站方向)经过 13 站到地坛西门站

Step 5: 步行 473 米

