



**本科毕业设计（论文）**

**题目名称：**基于机器学习的微信聊天机器人的设计与实现

**学 院：** 计算机科学技术学院

**专业年级：** 软件工程14级

**学生姓名：** 孙晓聪

**班级学号：** 2班19号

**校内指导教师：** 贺 薪 宇

**企业指导教师：** 李宇

**二〇一八 年 六 月 十 日**

# 摘 要

近几年机器学习的热潮不断高涨，基于机器学习的应用不断出现，例如Google Assistance，Apple Siri。这些应用的出现进一步证实了机器学习具备了走出实验室，进入生产环境的条件。本文论述了基于机器学习的微信聊天机器人的设计与实现，主要从概述，方案论证，需求分析等几个方面循序渐进地论述。基于机器学习的微信聊天机器人主要是做一个可以代替微信用户被动与好友聊天的聊天机器人。主要设计思路：首先用户将微信账号与该应用绑定，用户上传以往聊天内容或者其他内容以供训练聊天机器人，应用被动接受用户微信好友聊天内容，最后通过RNN（循环神经网络）产生被动回复内容并发送给微信好友。该项目主要分为两个子项目：EVE（前端Web APP），MO（后端服务）。主要技术栈：前端React （开源前端框架）搭配 Redux （单向数据流模式）构建单页面Web APP，后端 Python Flask （基于Python的Web 框架）搭配 Tensorflow（谷歌机器学习框架） 和 Itchat 实现该项目的主要业务服务。该项目遵循敏捷开发的思想，采用快速迭代和持续集成的软件开发方式，旨在提供高质量，响应变化等创新型的软件。

**关键词：**循环神经网络；谷歌机器学习框架；React；Flask；敏捷开发；持续集成

# Abstract

In recent years, the enthusiasm for machine learning has continued to rise, and applications based on machine learning continue to emerge, such as Google Assistance and Apple Siri. The emergence of these applications further confirmed that machine learning has the conditions to get out of the laboratory and into the production environment. This article discusses the design and implementation of the Wechat chat robot based on machine learning. It mainly discusses gradually from several aspects such as overview, program demonstration and demand analysis. The machine learning-based WeChat chat robot is mainly a chat robot that can replace the WeChat users and chat with their friends passively. Main design ideas: First, the user binds the WeChat account with the application, the user uploads past chat content or other content for training the chat robot, the application passively accepts the user WeChat friend chat content, and finally generates a passive reply content through the RNN (recurrent neural network). And send it to WeChat friends. The project is divided into two sub-projects: EVE (front-end Web APP) and MO (back-end service). Main technology stack: front-end React (open-source front-end framework) builds a single-page Web APP with Redux (one-way data flow model), back-end Python Flask (Python-based Web framework) with Tensorflow (Google Machine Learning Framework) and Itchat to implement the project The main business services. The project follows the idea of ​​agile development, adopts rapid iterative and continuous integration software development methods, and aims to provide innovative software with high quality and responsiveness.

**Key words：**RNN；Tensorflow；React；Flask；Agile；CI/CD

# 目 录

[摘 要 I](#_Toc516494332)

[Abstract II](#_Toc516494333)

[第1章 概述 1](#_Toc516494335)

[1.1 课题的背景、意义 1](#_Toc516494336)

[1.1.1 课题的背景 1](#_Toc516494337)

[1.1.2 课题的意义 1](#_Toc516494338)

[1.2 国内外发展状况 1](#_Toc516494339)

[第2章 方案论证 3](#_Toc516494340)

[2.1 方案提出 3](#_Toc516494341)

[2.1.1 方案一 3](#_Toc516494342)

[2.1.2 方案二 3](#_Toc516494343)

[2.1.3 方案三 3](#_Toc516494344)

[2.2 可行性分析 4](#_Toc516494345)

[2.2.1 经济可行性 4](#_Toc516494346)

[2.2.2 技术可行性 4](#_Toc516494347)

[2.2.3 操作可行性 4](#_Toc516494348)

[2.2.4 法律可行性 4](#_Toc516494349)

[2.3 方案确定 5](#_Toc516494350)

[2.4 相关技术介绍 5](#_Toc516494351)

[第3章 需求分析 6](#_Toc516494352)

[3.1 功能需求分析 6](#_Toc516494353)

[3.2 性能需求分析 6](#_Toc516494354)

[3.3 系统数据流图 7](#_Toc516494355)

[3.4 系统E-R图 8](#_Toc516494356)

[3.5 系统数据字典 11](#_Toc516494357)

[3.6 系统用例图 12](#_Toc516494358)

[第4章 过程论述 13](#_Toc516494359)

[4.1 概要设计 13](#_Toc516494360)

[4.2 数据库设计 14](#_Toc516494361)

[4.2.1 数据库逻辑结构设计 14](#_Toc516494362)

[4.2.2 表实现 14](#_Toc516494363)

[4.3 详细设计 16](#_Toc516494364)

[4.3.1 微信绑定模块 16](#_Toc516494365)

[4.3.2 聊天内容发布模块 18](#_Toc516494366)

[4.3.3 聊天内容发布模块 19](#_Toc516494367)

[4.3.4 常用聊天内容管理模块 20](#_Toc516494368)

[4.3.5 算法设计模块 22](#_Toc516494369)

[第5章 软件测试与结果分析 25](#_Toc516494370)

[5.1 测试概述 25](#_Toc516494371)

[5.2 测试方法 25](#_Toc516494372)

[5.3 测试计划 26](#_Toc516494373)

[5.4 测试过程 26](#_Toc516494374)

[5.5 测试内容 27](#_Toc516494375)

[5.6 测试结论 29](#_Toc516494376)

[结 论 30](#_Toc516494377)

[致 谢 31](#_Toc516494378)

[参考文献 32](#_Toc516494379)

[附 录 33](#_Toc516494380)

# 第1章 概述

## 1.1 课题的背景、意义

### 1.1.1 课题的背景

本课题提出的时候正值人工智能热潮高涨之时，人工智能技术就像电一样将会给我们带来一场深入到每一个行业的技术革命。人工智能的一个子领域自然语言处理同样处于热潮之中，用于自然语言处理领域的机器学习框架层出不穷。同时即时聊天领域的发展也取得了举世瞩目的成绩，例如微信和QQ等，基本上都成为了我们生活的一部分。但是人们在这些即时聊天工具上花费了大量的时间去处理一些相同的工作。基于机器学习的微信聊天机器人正是在这样的背景下提出，旨在通过人工智能的手段减少人们花费在聊天工具上的时间。

### 1.1.2 课题的意义

本课题的意义在于给自动化聊天提供一种人工智能的解决方案，探索人工智能在自然语言处理领域的实际应用能力。微信聊天作为一种聊天工具，与我们的生活息息相关，几乎成为了我们生活不可分割的一部分。据微信发布的用户数据显示，微信日登陆用户达9亿多，人均每天花费在微信上的时间约一小时，也就是说我们每天花费在即时聊天上的时间大约是自己工作时间的八分之一。类似Google Assistance，Apple Siri 等，微信聊天助手是在占据大量时间的即使聊天中，自动化模板聊天，以减少用户花在这些方面的精力，这样的意义在于让自己的工作和生活更加精简。

## 1.2 国内外发展状况

中国的即使聊天的工具虽然起步晚，但是后起勃发，现今中国的即使聊天也处于领先水平。当然国外也有很多优秀的即使聊天工具，例如Whatsapp，Facebook Messenger等。值得注意的是在Facebook 2017 F8 大会上，Facebook 发布了Messenger Chat Extensions，其中包括 Smart Replies。Smart Replies 是Facebook Messenger 内置的AI功能，它提供了学习用户常用的问题回答，来帮助用户自动回复一些常用回答。Facebook Messenger 的目的旨在将自动化的服务带给小微企业，给予其提供7\*24小时不间断服务。Smart Replies 是自动从公司的主页采集信息，识别用户的问题，然会进行回答。

国内的微信和QQ是主流的即使聊天工具，占据了大量的市场份额。QQ没有相关助手的扩展，其更核心的业务是提供更加娱乐性的功能。相对与QQ，微信提供更加简洁即使聊天体验，虽然都是腾讯的产品，其给与客户的体验截然不同。微信提供了助手相关的扩展，例如微信公众号。在微信公众号内，微信公众号的运营者可以提供给用户定制化的服务，包括定制化的聊天助手等，但是还没有针对微信用户的个人聊天助手。

# 第2章 方案论证

## 2.1 方案提出

### 2.1.1 方案一

方案一技术栈：前端Vue + 后端 SSM + 机器学习 Tensorflow [1]+ 微信接口 Itchat。

提出课题后不久，就开始着手整个方案的选择相关事宜。基于自己的技术栈，前端选择Vue 框架。Vue 是一套用户快速构建用户界面的渐进式框架，搭配其他Vue生态圈的框架，可以构建用户友好的前端。SSM 是Spring MVC，Spring 和 Mybatis 的缩写，也是Java 后端主流解决方案之一，与 SSH ( Struts，Spring 和 Hibernate )相比，SSM具有更好的安全性和更加灵活的设计[2]。Tensorflow 是Google Brain 开源的机器学习框架。Tensorflow 可以被用于语音识别，图像识别和深度学习等多项领域，并且可以运行在多种平台[3]。Itchat 是Github 上面开源的一个微信聊天接口包装库，微信源生并不支持将用户的聊天开放的接口，这个开源库只是包装微信Web为API，并暴漏为Python的API。结合以上所述的Vue，SSM，Tensorflow和Itchat 理论上可以实现该课题的主要内容。

### 2.1.2 方案二

方案二技术栈：前端React + 后端 Spring One Stop + Tensorflow + Itchat。

与上一种方案相比，前端改用了 React ，后端改用了Spring One Stop。React 是Facebook 开源的一个前端框架，与Vue相比，React 更加适合构建单页面Web APP， 同时React 的生态圈更加丰富，搭配Redux 可以更加很好的管理单页面数据流。Spring One Stop 是Spring 生态圈的一个很好体现，Spring 拥有完备的生态圈，提供了Spring Boot， Spring Data JPA等[4]。如果没有特殊需求，业界更加希望采用Spring One Stop 解决方案，这样可以让后端的每一个子模块更加无缝衔接。

### 2.1.3 方案三

方案三技术栈：前端React +后端 Python Flask + Tensorflow + Itchat。

与前一种方案相比，后端改用了Python Flask。Flask 是基于 Python 的一个Web 开发框架，提供了更加精简的Web开发核心，通过扩展来显示的添加其他功能，而不是内设在开发框架内部[5]。Python 作为机器学习领域的主流语言之一，具有先天的优势。Tensorflow 虽然底层使用C++ 开发，但是应用层API暴漏的是Python接口，同时Itchat 也是暴漏的Python API。后端采用基于Python 的Flask Web 框架，可以使得后端，Tensorflow 和 Itchat 更好的无缝集成。

## 2.2 可行性分析

### 2.2.1 经济可行性

在2.1方案选择里所论述的三种解决方案，所有涉及的技术都是开源技术，都已开源协议的形式开源，在不商用的情况下不涉及授权问题。同时开发只需要个人笔记本电脑一台，可以连接网络即可。结合以上论述的内容，本课题基本不涉及大量经济投入，经济可行。

### 2.2.2 技术可行性

如2.1方案选择里所述的方案，开发本课题有多种选择方案。后端数据库所有方案统一采用MySQL 开源数据库，采用开源操作系统Ubuntu和社区版的IntelliJ IDEA。在以前的学习过程中，有基于MySQL，Ubuntu 和 IDEA 的开发经验，同时在校外学习的过程中有相关敏捷开发，CI/CD等实习经验，可以将其思想引入到本项目，以减少项目在设计实现过程中遇到的问题。在Ubuntu 操作系统的社区中有很多开源软件，可以提供基本完备的工具集合，开发中需要用到的工具基本都可以在这里找到，例如版本控制器Git，虚拟网络VPN，容器Docker等。同时作为开发者，使用Ubuntu开发，可以很好的将开源软件整合到一个平台，大多数开源软件都会提供一个Linux 发行版的二进制包，这样即增加了项目的技术可行性也有利于后期将项目使用Docker 部署到云主机[6]。综合这些论述，不存在较大技术阻碍，技术可行。

### 2.2.3 操作可行性

本项目暴漏给微信用户的是一个单页面的Web App，用户只需扫码或则在微信里面输入相应的网址即可跳转到该Web App。用户所有相关操作都只需在该Web App 中操作。绑定微信账号到系统，用户只需扫描系统呈现的二维码，授权登陆即可。其他相关功能，用户只需同使用其他App一样的习惯即可上手本Web App。同时本项目结合了CI/CD 采用 Jenkins 和 Docker持续集成项目[7]，用户无需等待项目的发布[8]。结合以上论述的几点，本项目无复杂用户不友好操作，操作可行。

### 2.2.4 法律可行性

纵观以上论述的技术，大多数技术都是出自开源社区，版权通过开源协议的形式授权，例如：React 是Facebook 以MIT 开源协议授权的一个前端开源框架，及被授权人有使用，修改和合并等权力[9]。其他少数商业软件都是采用的社区版本，社区版本的商业软件是无需授权即可使用。同时本课题没有介入微信任何相关业务，只是引用包装了微信客户接口的第三方库来实现接口服务。且本项目为综合实践项目，不存在商业目的。综合以上几点，无严重法律问题，法律可行。

## 2.3 方案确定

方案的确立一波三折，从最开始的Vue+SSM+Tensorflow+Itchat 到 React+SOS+Tensorflow+Itcaht 最后到 React+Flask+Tensorflow+Itchat。每一种方案都有各自的好处，都有值得选择的理由。第一种方案，更像是传统互联网项目解决方案，具有普遍性。第二种方案后端技术更加契合，前端采用更加成熟的React 使得项目在理论上更加具有可行性。最后一种方案是采用Python Flask 这种方案主要是考虑到 Tensorflow 和 Itchat 都是暴漏Python API，如果采用 Flask 作为后端框架，这将使得后端跟其他服务能够更加紧密的衔接。

综合以上论述，最后采用方案三（React+Python Flask+Tensorflow + Itchat）。数据库选择MySQL，开发操作系统选择Ubuntu，开发IDE 选择社区版的 IntelliJ IDEA 和 Pycharm，版本控制器选择 Git[10]，并且将代码托管到GitHub，部署环境选择 CentOS 搭载 Docker 和 Jenkins，敏捷看板采用Github 提供的Project[11]。

## 2.4 相关技术介绍

经过方案确定，最终选择方案三。方案三涉及到的技术栈有React，Flask，Tensor flow，Itchat和MySQL。React 是Feacebook 开源的前端框架，React 主要用于构建前端用户交互页面，React 拥有强大的生态圈，本项目将采用其中一个单向数据流设计模式Redux，这样的设计更加有利于数据的维护。Flask 是一个基于Python 编程语言的Web 开发框架，该框架提供简单易用的API以供构建RESTFul 的Web 服务。Tensorflow 是谷歌大脑项目开源的一个机器学习框架，Tensor 是张量及数据，Flow 代表流图及数据处理的流程。Tensorflow是一个应用广泛的机器学习框架，可以广泛用于机器视觉，自然语言处理等领域。同时该框架可以运行与各种平台包括CPU，GPU等。Itchat是开源社区对微信Web协议的封装，以便便捷的调用微信接口。MySQL是甲骨文公司旗下的数据库产品，提供关系型存储。该数据库在中小型企业应用广泛，同时该数据库提供一个开源版本，可以免费使用。

# 第3章 需求分析

## 3.1 功能需求分析

本项目包含五大模块：微信账号绑定模块，聊天内容识别模块，聊天内容发布模块，聊天管理模块，深度学习算法设计模块。

1. 微信账号绑定模块

微信用户初次进入该系统，需要将微信与该系统绑定，同时授权该系统登陆微信以获取用户聊天记录。主要操作流程为：用户进入系统，系统展示二维码，用户扫描二维码并授权登陆，系统与微信账号绑定。

1. 聊天内容识别模块
2. 文字聊天内容识别：当微信好友发送文字内容给系统时，系统识别为文字内容，并将文字内容交由上层算法模块。
3. 语音聊天内容识别：当微信好友发送语音内容给系统时，系统接口将语音转换为文字，并将文字内容交由上层算法模块。
4. 文本文件内容识别：当微信好友发送文本文件内容给系统时，系统将文本文件内容解析为文字，并将文字交由上层算法模块。
5. 聊天内容发布模块

文字内容发布：当接受到上层算法模块的文字内容时，将文字内容发送给指定的微信好友。

1. 常用聊天内容管理模块
2. 上传常用聊天内容：通过文本文件上传常用聊天内容，通过该内容训练聊天机器人。
3. 删除常用聊天内容： 删除以前上传的聊天内容，并且从新训练聊天机器人。
4. 查询常用聊天内容： 通过别名查询常用聊天内容。
5. 列出聊天记录： 通过列表的形式列出聊天内容记录列表。
6. 算法设计模块

聊天机器人算法：设计和实现聊天机器人，可以被动回复文本。

## 3.2 性能需求分析

系统的反应加载速度应尽可能的快，作为单页面Web APP 在系统第一次加载的过程中，系统会将所有的资源加载完，这样会导致第一次加载的时候稍微缓慢，所以需通过响应的技术减少第一次加载的时间，例如：资源压缩，JS代码压缩等，同时需要被动回复的时间在用户接受范围内。后端通过Restful API 给前端暴漏接口，这就需要后端服务的响应时间不能太久，如果超过限定的超时时间，前端应该给与超时的友好提示，但是尽量避免这样的情况发生。

## 3.3 系统数据流图

1．根据需求得到如下图所示的顶级数据流图

图3-1 顶级数据流图

2．对顶级数据流图细化得到如下二级数据流图

1. 微信绑定系统数据流图

图3-2 微信绑定系统数据流图

1. 聊天流程数据流图

图3-3 聊天流程数据流图

1. 聊天内容管理数据流图

图3-4 聊天内容管理数据流图

## 3.4 系统E-R图

根据以上的分析，罗列出了以下系统E-R图。

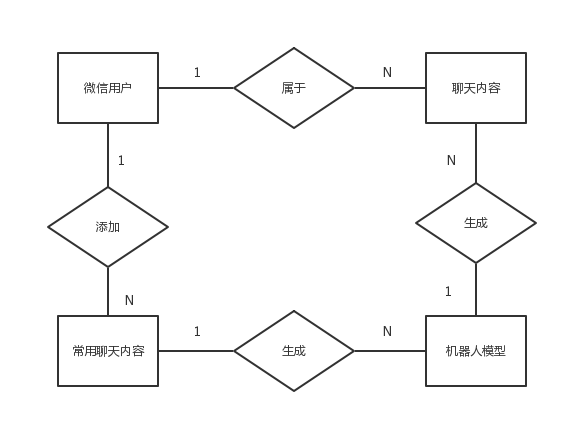
1. 系统E-R图

图3-5 系统E-R图

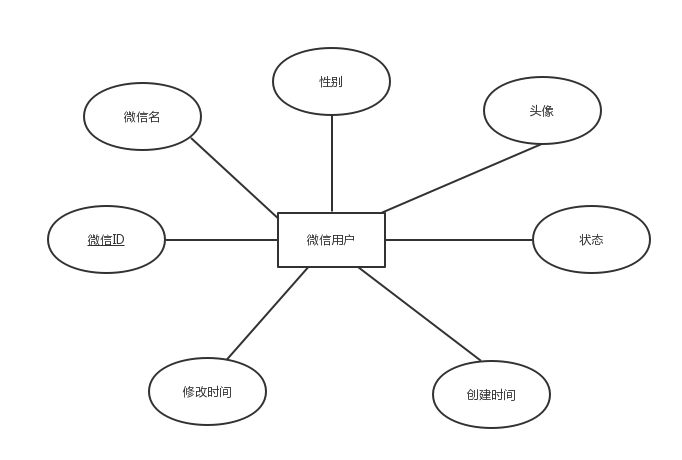
1. 微信用户实体图

图3-6 微信用户实体图

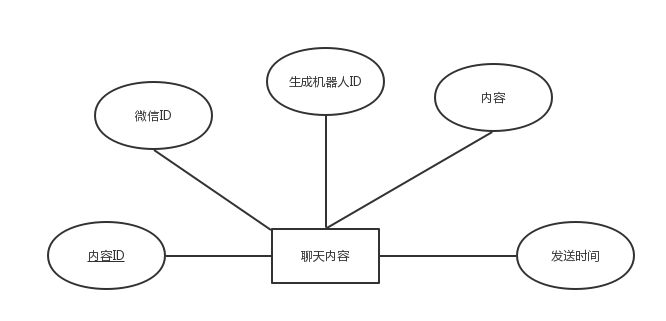
1. 聊天内容实体图

图3-7 聊天内容实体图

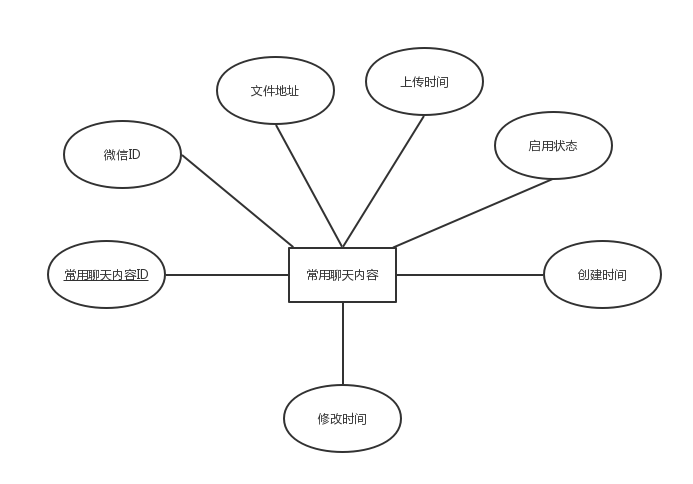
1. 常用聊天内容实体图

图3-8 常用聊天内容实体图

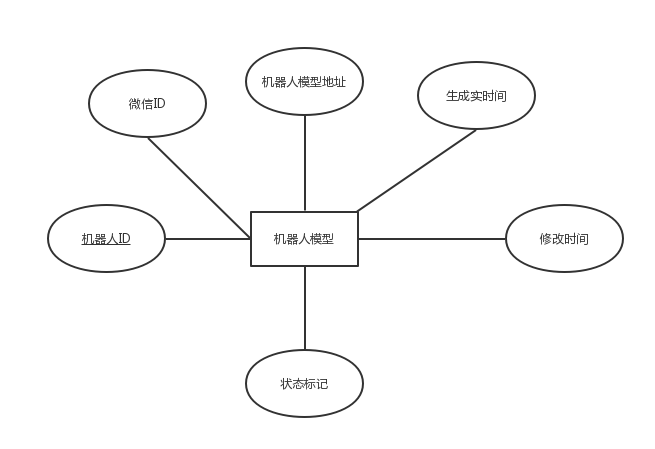
1. 机器人模型实体图

图3-9 机器人模型实体图

## 3.5 系统数据字典

经过以上分析得出以下系统数据字典

1. 数据结构：微信用户

含义说明：记录微信用户信息

组成：微信ID，微信名，微信用户性别，微信头像，创建时间，修改时间。

1. 数据项：微信ID含义说明：微信用户唯一标识。
2. 数据项：微信名含义说明：微信用户昵称。
3. 数据项：微信用户性别含义说明：微信用户性别。
4. 数据项：微信头像含义说明：微信头像地址。
5. 数据项：创建时间含义说明：该微信用户第一次绑定时间
6. 数据项：修改时间含义说明：该微信用户上一次绑定时间

2. 数据结构：聊天内容

含义说明：微信用户的聊天记录

组成：内容ID，微信ID，聊天内容，生成模型，发送时间，生成机器人ID。

数据项：系统开发时标识的唯一管理员帐号与密码。

1. 数据项：内容ID含义说明：内容唯一标识。
2. 数据项：微信ID含义说明：微信用户ID标识。
3. 数据项：聊天内容含义说明：发送给微信好友的的聊天内容。
4. 数据项：发送时间含义说明：发送给微信好友的时间。
5. 数据项： 生成机器人ID含义说明：生成该聊天内容的机器人ID。

3. 数据结构：常用聊天内容

含义说明：微信用户常用聊天内容，以供训练聊天机器人。

组成：常用聊天内容ID，微信ID，常用聊天内容文件地址，上传时间，启用状态，创建时间，修改时间。

1. 数据项：常用聊天内容ID含义说明：常用聊天内容唯一标识。
2. 数据项：微信ID含义说明：常用聊天内容所属微信用户标识。
3. 数据项：常用聊天内容文件地址含义说明：文件地址。
4. 数据项：上传时间含义说明：文件上传时间。
5. 数据项：启用状态含义说明：启用状态标记。
6. 数据项：创建时间含义说明：该记录创建时间。
7. 数据项：修改时间含义说明：上一次修改记录时间。

4. 数据结构：机器人模型

含义说明：机器人模型记录。

组成：机器人ID，微信ID，机器人模型地址，生成时间，状态标记，修改时间。

1. 数据项：机器人ID含义说明：机器人唯一标识。
2. 数据项：微信ID含义说明：所属微信标识。
3. 数据项：机器人模型地址含义说明：机器人模型文件保存地址。
4. 数据项：生成时间含义说明：机器人生成时间。
5. 数据项：状态标记含义说明：启用状态标记。
6. 数据项：修改时间含义说明：上一次修改时间。

## 3.6 系统用例图

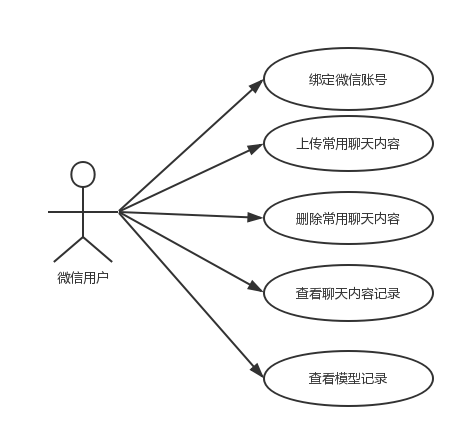
根据以上分析绘制了如下系统用例图

图3-10 系统用例图

# 第4章 过程论述

## 4.1 概要设计

本系统主要分为：微信绑定模块，聊天内容识别模块，聊天内容发布模块，聊天内容管理模块，深度学习算法设计模块。具体流程为微信用户绑定微信账户与系统，微信用户可以查看绑定的微信账户信息，上传和管理常用聊天内容，查看算法模型记录，查看聊天内容记录等。总体功能结构如下图：

图4-1 系统功能结构图

## 4.2 数据库设计

### 4.2.1 数据库逻辑结构设计

E-R图转化为关系模型就是将每个实体类型转化成一个关系模式。实体的属性对应着关系模式的属性。实体的标识对应着关系模式的键，故将以上E-R图转化成关系模型。

微信用户(微信ID，微信名，微信用户性别，微信头像，状态，创建时间，修改时间)；

聊天内容(内容ID，微信ID，生成机器人ID，聊天内容，发送时间)；

常用聊天内容 (常用聊天内容ID，微信ID，常用聊天内容地址，上传时间，启用状态，创建时间，修改时间)；

机器人模型(机器人ID，微信ID，机器人模型地址，生成时间，状态标记，修改时间)；

### 4.2.2 表实现

本系统大量数据都是存储在数据库里面，除了本分训练文本和模型是通过文本文件的形式存储在文件系统当中，但是也是将对应的文件地址存储在数据库中。虽然数据所涉及的表不是很多，但是力求更好的设计以支撑程序的设计。下面列举了数据库表的详细设计。

1. 微信用户信息表（wechat\_info）

微信用户信息表的主要重用是保存与系统绑定的微信用户的信息，包括微信ID，微信名等，其中微信ID是唯一标识，并且作为表的主键，详情如下表。

表4-1 微信用户信息表(wechat\_info)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名** | **主键** | **类型** | **长度** | **允许空** | **字段说明** |
| 1 | wechat\_id | 是 | varchar | 128 | 否 | 微信ID |
| 2 | wechat\_name |  | varchar | 64 | 否 | 微信名 |
| 3 | gender |  | tinyint unsigned | 1 | 否 | 0男1女 |
| 4 | avatar |  | varchar | 128 | 否 | 头像地址 |
| 5 | gmt\_modified |  | datetime |  | 否 | 修改时间 |
| 6 | gmt\_create |  | datetime |  | 否 | 创建时间 |
| 7 | status |  | tinyint unsigned | 1 | 否 | 状态 |

1. 微信聊天内容记录表（wechat\_records）

微信聊天内容记录表主要是记录微信聊天机器人发送给微信好友的聊天信息，包括发送的微信用户ID，发送内容以及时间等，详情如下表。

表4-2 微信聊天内容记录表(wechat\_records)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名** | **主键** | **类型** | **长度** | **允许空** | **字段说明** |
| 1 | id | 是 | int unsigned | 4 | 否 | 唯一标识 |
| 2 | wechat\_id |  | varchar | 128 | 否 | 微信ID |
| 3 | chat\_content |  | varchar | 256 | 否 | 聊天内容 |
| 4 | robot\_id |  | int unsigned | 4 | 否 | 聊天机器人id |
| 5 | gmt\_sent |  | datetime |  | 否 | 发送时间 |
| 6 | status |  | Tinyint unsigned | 1 | 否 | 状态 |

1. 微信常用聊天内容表

微信常用聊天内容表主要保存用语训练深度学习算法的聊天记录，包括常用聊天记录文件地址等，详情如下表。

表4-3 微信常用聊天内容表(wechat\_sample)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名** | **主键** | **类型** | **长度** | **允许空** | **字段说明** |
| 1 | id | 是 | int unsigned | 4 | 否 | 唯一标识 |
| 2 | wechat\_id |  | varchar | 128 | 否 | 微信ID |
| 3 | txt\_addr |  | varchar | 256 | 否 | 文件地址 |
| 5 | gmt\_modified |  | datetime |  | 否 | 最后修改时间 |
| 6 | gmt\_create |  | datetime |  | 否 | 上传时间 |
| 7 | status |  | tinyint unsigned | 1 | 否 | 状态:1启用0禁用 |

1. 机器人模型记录表

机器人模型记录表主要是保存通过常用聊天内容训练出来的模型，主要包括模型文件地址，所属微信ID等，详细如下表。

表4-4 机器人模型记录表(wechat\_robot)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **字段名** | **主键** | **类型** | **长度** | **允许空** | **字段说明** |
| 1 | id | 是 | int unsigned | 4 | 否 | 唯一标识 |
| 2 | wechat\_id |  | varchar | 128 | 否 | 微信ID |
| 3 | model\_addr |  | varchar | 256 | 否 | 模型文件地址 |
| 5 | gmt\_modified |  | datetime |  | 否 | 最后修改时间 |
| 6 | gmt\_create |  | datetime |  | 否 | 生成时间 |
| 7 | status |  | tinyint unsigned | 1 | 否 | 状态:1启用0禁用 |

## 4.3 详细设计

### 4.3.1 微信绑定模块

微信用户绑定系统是此系统至关重要的一步，微信用户在没有授权系统登陆微信账户之前，系统无法获取好友聊天内容，大部分服务都无法进行。所以在使用系统的任何服务之前都需要微信用户扫描并授权系统登陆。

1. 前端微信二维码扫描页面

图4-2 扫描二维码绑定微信图

如上图所示，图中包含一个微信绑定二维码，二维码扫描提示，底部菜单以及侧边栏菜单，整个页面设计大方得体符合主流Web APP的操作习惯。整个页面通过React 实现，在此项目中前端项目是一个单独的子项目并且命名为eve，eve一词来源于斯皮尔伯格导演的《机器人总动员》中的eve角色，寓意集成eve完美的外观和优雅的动作。

在这个页面中，主要由组件FrameContainer 将内容包裹。FrameContainer 是一个共享想的组件，处理侧边栏菜单和底部菜单逻辑。主要代码如下，部分导入代码没有贴出来，详情请参照附录：

type Props = {

menuStatus: boolean,

bottomNavCheckedIndex: number,

toggleMenuStatus: () => void,

handleBottomNavClick: (number) => void,

push: (string) => void

};

const drawerWidth = 240;

class FrameContainer extends React.Component<Props> {

\_onBottomNavClick = (event, index) => {

const {handleBottomNavClick, push} = this.props;

handleBottomNavClick(index);

push(mapMenuIndexToURL(index));

};

render() {

const {classes, menuStatus, bottomNavCheckedIndex, t

oggleMenuStatus} = this.props;

return (

<div className={classes.appFrame}>

… the frameContainer component

</div>

);

}

}

export default \_.flowRight(

withRoot,

withStyles(styles)

)(FrameContainer);

1. 后端API设计

支撑前端显示二维码的是后端Flask提供的Restful API，通过将前后端分离的形式，可以更加有效的管理前端和后端的代码。

获取二维码API设计与传统的API不一样，此系统的API采用Restful 的API设计，这种设计方式出自Roy Fielding 的博士论文，他是曾经参与过Http协议的设计，同时也是 Apache Web Sever 的缔造者之一。此请求设计为如下：

HTTP/1.1 200

Content-Type: application/json

{

"code": 200,

"message": "Request Successfully",

"description": "The QR code is generated successfully",

"data": {

"url":"http://stage.icusin.com/auth/qr",

"type":"png",

"size":"14000",

"timeout":"360000",

"hint":"请扫描二维码绑定微信！",

"uuid":"axxxuuid",

},

"link":{

"rel":"collection http://stage.icusin.com/auth",

"href":"http://stage.icusin.com/auth/help",

"title":"Auth Help",

"type":"application/json",

}

}

1. 后端Flask 服务设计

后端Web服务采用Python Flask 框架编写，整个后端服务都是在此项目的一个子项目中完成，此子项目命名为mo，同样出自于《机器人总动员》中的mo，一个星外微生物清洁员，寓意为跟这个mo机器人一样灵活。二维码获取的服务放在flaskr下面的auth 模块中，由于篇幅有限具体代码就不放在下面，详情请参照附件。

### 4.3.2 聊天内容发布模块

内容识别模块是该微信好友发送过来的文本消息，语音消息或者文本消息转化为文本消息，并且传送给上层算法模块，这部分是算法有效输入的关键，通过监听微信好友发送给系统的消息来实现。部分相关Python 代码如下，详情代码请参考附录。

def run(self):

print("开始线程：" + self.name)

self.wechat\_instance.login(picDir=self.pic\_dir, loginCallback=self.login\_callback,

exitCallback=self.logout\_callback)

@self.wechat\_instance.msg\_register(itchat.content.TEXT)

def tuling\_reply(msg):

self.msg\_q.put(msg['Text'])

self.single.wait()

chatbot\_reply = self.reply\_q.get()

default\_msg = 'I received: ' + msg['Text']

tuling\_msg = tuling\_response(msg['Text'])

reply = tuling\_msg or default\_msg

record\_chat(msg, reply)

return reply

def record\_chat(msg, reply):

with app.app\_context():

wechat\_record\_instance = WechatRecord(self.wechat\_instance.loginInfo['User'].Uin,

self.service\_id, msg, reply)

db.session.add(wechat\_record\_instance)

db.session.commit()

self.wechat\_instance.run()

print("退出线程：" + self.name)

### 4.3.3 聊天内容发布模块

聊天内容发布模块是将通过算法生成的回复内容发送给该微信好友，该模块在Web APP 有一个可以查看聊天记录的菜单项通过该菜单项，微信用户可以查看曾经的聊天记录。

1. 前端聊天内容记录页面

图4-3 聊天内容记录页面图

正如上图所示，聊天内容记录页面包含一个展现聊天内容记录的列表，同样的也是被包裹在FrameContainer 组件内的一个组件。详细代码请参照附录。

1. 后端API设计

支撑前端显示聊天内容记录列表的是后端Flask提供的Restful API，通过将前后端分离的形式，可以更加有效的管理前端和后端的代码。Api Response 如所示，详细请参考附件。

HTTP/1.1 200

Content-Type: application/json

{

"code": 200,

"message": "Request Successfully",

"description": "The log list query successfully",

"data": [{

"replyContent":"记录1",

"replyTime ":"2018-05-17 11:19 01",

"status":1,},{

"replyContent":"记录5",

"replyTime ":"2018-05-17 11:00 08",

"status":1,

}],

"link":{

"rel":"collection http://stage.icusin.com/content/sent",

"href":"http://stage.icusin.com/content/sent",

"title":"Sent Content",

"type":"application/json",

}

}

### 4.3.4 常用聊天内容管理模块

常用聊天内容管理模块包括上传常用聊天内容和删除已经上传的聊天内容，是管理常用聊天内容的模块。常用聊天内容将和语料库一起训练人机交互模型，上传常用聊天内容将会通过文本文件的形式上传，并且当前页面会列出已经上传过的常用聊天记录的列表。

1. 前端常用聊天内容管理模块页面

图4-4 常用聊天内容管理页面图

通过如上图片可以看出，常用聊天管理页面包含一个常用聊天文件列表以及添加新的常用聊天内容按钮，同时这些组件都是通过FramContainer包裹，从而构建当前页面。详细代码请参考附录。

1. 后端API设计

支撑前端显示内容管理中常用聊天内容列表的是后端Flask提供的Restful API，通过将前后端分离的形式，可以更加有效的管理前端和后端的代码。Api Response 如所示，详细请参考附件。

HTTP/1.1 200

Content-Type: application/json

{

"code": 200,

"message": "Request Successfully",

"description": "The content list query successfully",

"data": [{

"name":"常用聊天内容1",

"gmt\_create":"2018-05-17 12:12 12",

"gmt\_modified":"2018-05-17 12:12 12",

"status":1,

"file\_addr":"http://stage.icusin.com/resources/ content/01283.txt"

},{

"name":"常用聊天内容2",

"gmt\_create":"2018-05-17 11:47 19",

"gmt\_modified":"2018-05-17 11:47 19",

"status":1,

"file\_addr":"http://stage.icusin.com/resources/ content/01823.txt"

}],

"link":{

"rel":"collection http://stage.icusin.com/content/add",

"href":"http://stage.icusin.com/content/add",

"title":"Add Content",

"type":"application/json",

}

}

### 4.3.5 算法设计模块

本课题的核心内容是人机对话，是自然语言处理（NLP）的一个应用[12]。自然语言处理作为人工智能中至关重要的技术，同机器视觉（CV）等一样，受到业界持续研究。与传统的人机对话不同的是，传统的人机对话是通过程序预先设定好一些常用的回答，当机器人接收到一个问题的时候，就去之前设定好的回答库里面去搜索相关的答案。当然Facebook 也有一些尝试，通过机器学习的分类器将提前设定好的问题和答案进行分类，从而来识别用户的意图。

如今机器学习在各个领域都有用武之地，自然语言处理领域也不例外。通过深度学习可以构建不需要通过预先设定问答数据的人机交互，只需在人机交互之前通过数据集将人工神经网络（NN）训练好[13]。

1. 加载数据

训练模型需要大量数据集训练该模型，及需要加载这些数据集。所有的数据集事先都保存在/data/train/目录下，只需加载该目录下面的数据集即可，以供后续步骤训练[14]。代码如下，详细代码请参考附录。

print("—Data payload")

dataset = neuralconvo.DataSet(neuralconvo

.CornellMovieDialogs("data/tran/ "),{

loadFirst = options.dataset, -- define the data size

minWordFreq = options.minWordFreq –- define the miminal freq

})

1. 建模

建模用到的是seq2seq，它包含两个 LSTM 递归神经网络，第一个是 encoder 负责处理 input，第二个是 decoder 负责生成 output[15]。如下图所示：

图4-5 seq2seq图解图

深度人工神经网络（DNN）的 inputs 和 outputs 的维度是固定的，然而我们接收的是一句话，输出的也是一句话或则串单词，我们需要一个模型可以保持一定长度的记忆。LSTM可以将变长的inputs转化为固定维度的向量表达。给了足够多的数据后，模型可以将两个相似的问题识别成同一个 thought vector 表达出来。学习模型之后，可以同时得到权重和thought vector[16]。

图4-6 LSTM图

建模的流程分为加载数据集，预处理，切词，问答训练和转换为向量并保存为二进制文件，具体流程图如下图。

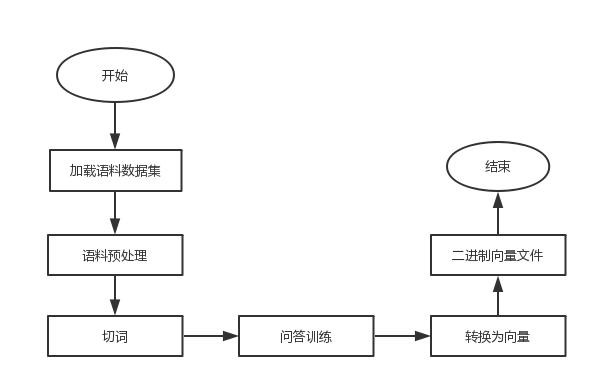


图4-7 建模流程图

建模源码如下，详细请参考附录。

class Chatbot(threading.Thread):

def \_\_init\_\_(self, wechat\_id, sample\_file\_path):

threading.Thread.\_\_init\_\_(self)

self.sample\_file\_path = sample\_file\_path

self.wecaht\_id = wechat\_id

def run(self):

print("Start training")

save\_path = MODEL\_SAVE\_DIR\_PRE\_FIX + str(uuid.uuid1())

extract\_data(self.sample\_file\_path, save\_path)

start\_train(save\_path)

with app.app\_context():

wechat\_robot\_instance = WechatRobot(self.wecaht\_id, save\_path)

db.session.add(wechat\_robot\_instance)

db.session.commit()

class ChatService(threading.Thread):

def \_\_init\_\_(self, model\_addr):

threading.Thread.\_\_init\_\_(self)

self.model\_addr = model\_addr

def run(self):

print("Start chat service...")

start\_service(self.model\_addr, msg\_q, reply\_q)

print("End chat service...")

# 第5章 软件测试与结果分析

## 5.1 测试概述

测试作为软件开发过程中至关重要的一步，测试的完备性关乎到软件交互的质量。本项目遵循敏捷开发的思想，将测试过程融入到每一次迭代过程中，通过CI/CD将测试运行集成到每一步开发中，本章将循序渐进地论述该项目的测试过程和测试方法。

## 5.2 测试方法

本项目分为两个子项目：前端项目和后端服务项目，每一个项目都有自己独立测试的方法，最后通过集成测试将前端和后端整合到一起测试以保证系统整体的可用性。

1. 前端测试方法

（1）前端单元测试

本项目前端采用React -Redux构建的单页面Web APP，整个APP的UI通过组件的形式组合构成。单元测试将保证每一个组建功能的完整性以及与业务的契合度，所以采用Mocha 作为单元测试的框架，结合Enzyme对React 做单元测试。

（2）前端E2E测试

采用Mocha 做单元测试，但是对于整个Web App单独在浏览器里面的运行效果以及是否与业务吻合需要另外的测试来覆盖。Nightwatch 作为E2E 测试的框架，搭配Selenium Server 可以对整个前端在浏览器的行为表现做测试覆盖。

（3）前端UI测试

由于前端交付的复杂性，我们无法通过单元测试和E2E测试去覆盖整个APP的UI是否和最初的设计吻合，引入BackStopJS 对APP UI 做测试覆盖。

1. 后端测试方法

本项目采用Python Flask 构建后端服务，所以测试采用Python unittest库对Flask 做单元测试以达到逻辑覆盖。

1. API测试

后端将API暴漏给前端，前端通过调用API以取得相应的服务，为了保证API的服务正常，引入Postman 对API 进行自动化测试。

## 5.3 测试计划

为了将上述测试手段引入到项目，制定了如下测试计划。通过将测试引入到CI/CD中，保证在每一次提交后自动运行测试脚本，测试脚本包括如上论述的前端单元测试，E2E测试，UI测试，后端单元测试以及API测试，通过运行来达到对每一次代码修改的质量保证。

表5-1 测试运行计划表

|  |  |  |
| --- | --- | --- |
| **序号** | **测试内容** | **测试类型** |
| 1 | 前端Web APP | Mocha单元测试 |
| 2 | 前端Web APP | Nightwatch端到端测试 |
| 3 | 前端Web APP | Backstop UI测试 |
| 4 | 后端服务 | Unittest单元测试 |
| 5 | 后端API | Postman API测试 |

## 5.4 测试过程

 整个测试过程遵循以上论述的测试计划，整个测试过程将在Jenkins Test Stage中完成，为了更加直观的体现测试流程绘制了如下测试流程图。

图5-1 测试流程图

## 5.5 测试内容

测试作为程序不可分割的一部分，每一段逻辑都包含大量的测试，由于测试测数量较为庞大，且以上论述的大多为百合测试，加上黑盒测试，测试的内容本节将无法全部展示，所以每一种测试抽取简单样本以供参考。

1. E2E 测试微信绑定流程

图5-2 微信绑定E2E测试流程图

E2E测试作为黑盒测试的一种，保证了业务流程与预期结果符合。根据如上的微信绑定E2E测试流程图列出了如下更加详细的测试用例表。

表5-2 微信绑定E2E 测试用例表

|  |  |  |
| --- | --- | --- |
| **测试用例** | **预期结果** | **是否达到预期** |
| 微信用户未绑定系统，进入系统 | 显示二维码 | 是 |
| 微信用户绑定系统，进入系统 | 显示微信用户信息 | 是 |
| 微信用户扫描并授权登陆 | 显示微信用户信息 | 是 |
| 微信用户扫描不授权 | 显示二维码 | 是 |

部分测试代码如下：

module.exports = {

'Should appear QR code' : function (browser) {

browser

.url(STAGE\_DEAMO\_URL')

.waitForElementVisible(BODY, 1000)

.click(BUTTON\_PROFILE\_SELECTOR)

.waitForElementVisible('button[name=btnG]', 1000)

.assert.containsText(QR\_CODE\_HINT\_SELECTOR,QR\_SCAN\_HIN)

.assert.elementPresent(QR\_CODE\_IMG\_SELECTOR)

};

1. 前端单元测试

前端单元测试作为更为细粒度的一种测试，可以对组件逻辑显示进行有效覆盖。如下将以FrameContainer 容器组件为例，论述其部分单元测试过程。

表5-3 FrameContainer容器组件测试用例表

|  |  |  |
| --- | --- | --- |
| **测试用例** | **预期结果** | **是否达到预期** |
| menuStatus=false | menuBar closed | 是 |
| menuStatus=true | menuBar open | 是 |
| bottomNavCheckedIndex=0 | .profile green | 是 |
| bottomNavCheckedIndex=1 | .content green | 是 |

部分代码如下：

describe('Frame Container',function () {

context(’Hanmmer’,function(){

it('should render hammer when menuStatus is true',function(){

let wrapper = render(<FrameContainer menuStatus={true}/>); expect(wrapper.find(‘Hanmmer’)).to.be.present();

});

it('should not render hammer when menuStatus is false',function(){

let wrapper = render(<FrameContainer menuStatus={false}/>); expect(wrapper.find(‘Hanmmer’)).to.not.be.present();

});

it('should trun the bottom profile to green when bottomNavCheckedIndex eq to 0',function(){

let bottomNavCheckedIndex=0;

let wrapper = render(<FrameContainer bottomNavCheckedIndex={bottomNavCheckedIndex}/>); expect(wrapper.find(‘.profile’)).to.have.prop(‘color’,’green’);

});

… the rest of the code can be find in the append.

});

});

通过对系统各个模块进行有效测试得出如下，测试结果表。

表5-4 系统模块测试结果图

|  |  |  |
| --- | --- | --- |
| **系统模块** | **预期结果** | **测试结果** |
| 微信用户绑定模块 | 微信用户与系统绑定 | 达到预期结果 |
| 内容发布模块 | 能够发布内容 | 达到预期内容 |
| 内容识别模块 | 识别好友发送的内容 | 达到预期结果 |
| 内容管理模块 | 能够对内容管理 | 达到预期结果 |
| 算法设计模块 | 生成聊天内容 | 达到预期j结果 |

## 5.6 测试结论

通过对系统加入以上论述的测试，整个项目代码进行了有效测试的覆盖，在每一次修改代码的过程中，都会有测试保证修改不会造成已经完成的功能的崩溃，有效的减少了问题的出现，提高了代码的质量。测试，是一把双刃剑，虽然测试能有效的覆盖代码以此减少程序员疏忽造成的问题，但是测试需要付出时间和精力来完成，这样会拖慢整个项目的进度。

本项目系统通过集成单元测试，E2E测试，UI测试和API测试来保证软件质量，这样不仅提高了代码的质量和软件的质量，同时也有利于后期维护和对代码的重构。通过这个项目让我意识到虽然写测试很麻烦，甚至有可能是台下十年功，台上一分钟的效果。但是这样确实让自己对自己写的软件的质量更有信心，让软件更加易于维护。

# 结 论

通过前几章的论述，对整个系统的功能和设计思想都有了较为清楚的论述。本系统是一个基于机器学习的微信聊天机器人，使用常用聊天内容来训练RNN 模型，并通过RNN 模型产生聊天回复，并将回复发送给该微信好友。整个系统通过有效的整合前端React 后端 Flask ，Tensorflow 和 Ithchat实现了整个系统服务，并且整合了自己在校外实习期间所学习的技术相关知识。

本次课程设计可谓是历经艰辛，学习了很多自己从未接触过的知识。刚开始确立题目后自己就在Github（Git项目托管平台）上面建立了一个depoma的仓库，以此提醒自己每天关注自己的毕业设计，同时为了满足开发的需要，自己忍痛割爱，将自己的笔记本装成了Ubuntu (开源Linux操作系统)和Windows 双系统，并且将自己两个云主机投入开发，一个作为部署环境，一个作为CI/CD Jenkins托管环境。期间查阅了很多技术相关的文档，例如：Docker ，刚开始接触Docker 的时候只是知道Docker是一个容器化部署工具，并没有亲自实践。当自己毕业设计需要用到此工具的时候，自己硬是将Docker官方文档认认真真的阅读了一遍。有时候当你遇到难题不熟悉的东西的时候，自己不愿意接触，但是当你认认真真却接触这个东西，并将隐藏在后面的东西探索明白的时候一切都不再是那么的难以解读。

就像在选题目的中论述的一样，选择这个基于机器学习的微信聊天机器人并不是心血来潮，自己在没有进行校外实习之前就对机器学习产生了浓厚的兴趣，但是一直没有机会去实践。正是本毕业设计课题让我有了这样近距离的再次接触机器学习，之前机器学习就像是被神秘面纱蒙住过的美少女，让人有一种想靠近但是又不知道怎么着手一样。虽然毕业后自己可能不会从事这方面的工作但是对于这方面的研究兴趣并不会减弱，自己将会在这个领域去探索自己感兴趣的技术。

# 致 谢

本系统的开发离不开各位辛苦工作在教育一线的老师们的辛勤指导，本系统是在贺薪宇老师辛勤指导下完成，感谢贺老师给予的辛勤指导，同时感谢企业实习导师李宇给与的指导。

整个系统的设计能够完成也少不了开源社区的支持，本项目是站在巨人的肩膀上开发的。感谢有像Facebook React，以及 React 生态圈[17]，Python Flask, Google Tensorlfow, Itchat 等一批优秀的开源软件。当下软件的开发离不开开源社区的支持，如今已经不是那个单枪匹马就能写出自己软件的年代，再次感谢开源社区的贡献。当然阿里云和腾讯云推出的校园计划让经济微薄的在校学生有能力试用云主机，感谢阿里云和腾讯云对在校生学习的支持。最后感谢写代码的时候帮我煮饭的人，订外卖帮我取外卖的，帮我倒水的室友。

最后给予帮助过我的人以真诚的感谢，晓聪敬上!

# 参考文献

[1]李锐，李鹏. 机器学习实战[M]. 北京: 人民邮电出版社.2013, 35-176.

[2] Craig Walls，耿渊，张卫滨. Spring实战[M]. 北京: 人民邮电出版社.2016，46-98.

[3] 郑泽宇，梁博文，顾思宇. TensorFlow：实战Google深度学习框架[M]. 北京: 电子工业出版社.2018，42-84.

[4] Srping Community. Spring Framework Docs[G]. https://spring.io/.

[5] 格林布戈，安道. ·Flask Web开发[M]. 北京: 人民邮电出版社.2015，10-68.

[6] 何晓龙. 完美应用Ubuntu [M] .北京: 电子工业出版社.2017，24-146.

[7] 伊恩·米尔，艾丹·霍布森·塞耶斯，吴佳兴. Docker实践[M]. 北京: 人民邮电出版社. 2018，13-123.

[8] John Ferguson，郝树伟. Jenkins权威指南[M]. 北京: 电子工业出版社[M].2016，10-121.

[9] Alex，Banks，Eve，Porcello，邓世超. React学习手册[M]. 北京: 中国电力出版社.2017，12-80.

[10] 乔恩·罗力格，马修·麦卡洛，王迪，丁彦. Git版本控制管理[M]. 北京: 人民邮电出版社.2015，12-126.

[11] 科恩，廖靖斌，吕梁岳. Scrum敏捷软件开发[M]. 北京: 清华大学出版社.2010，42-163.

[12] Giuseppe Ciaburro, Balaji Venkateswaran. Neural Networks with R[M]. First Edition Packt Publishing.2017，4 - 86.

[13] Simon Haykin，申富饶. 神经网络与机器学习[M]. 北京: 机械工业出版社.2011，18-164.

[14] 格兰特·英格索尔,王斌. 驾驭文本:文本的发现、组织和处理[M]. 北京: 电子工业出版社.2015，8-68.

[15] 郑捷. NLP汉语自然语言处理原理与实践[M]. 北京:电子工业出版社.2017， 6- 48.

[16] 伯德 Steven Bird, 克莱恩Ewan Klein, 陈涛, 张旭. Python自然语言处理[M].北京: 第一版人民邮电出版社. 2014，12-98.

[17] Facebook. React Framework Docs[G]. https://reactjs.org/.

# 附 录

本项目分为两个完全独立的子项目，前端EVE子项目和后端服务MO子项目，所有的代码都是通过Git版本控制器管理。由于代码过多就不在本文论述范围内，详细代码可以参照源码。

1. 使用说明书

用户在使用本系统之前需要将本系统部署于一台服务器。对于服务器没有过多要求，因为本项目是采用Docker 进行集成，所以只需一台安装Docker的服务其即可。需要将本项目的两个子项目MO（后端服务项目）和EVE（前端项目）分别构建为Docekr 镜像，相应的Linux 脚本已经放在了本项目的根目录，然后将Docker 镜像启动为对应的容器。

本项目是采用React 构建的单页面Web APP应用，可以在各种设备跨平台应用。当项目启动过后用户可以通过用户代理及浏览器访问前端页面，首先是提供了一个微信用户与系统绑定的界面，用户需要用自己的微信扫描该微信二维码并授权才能将系统与用户微信绑定。当用户微信与系统绑定过后，用户可以在该Web APP查看历史服务记录，常用聊天内容模板，聊天机器人以及聊天机器人等。

1. 核心代码

微信后端服务交互代码，主要负责处理与微信之间交互，包括登陆，接受消息和回复消息，主要实现方式是通过Python 多线线程，通过微信消息监听线程和机器人线程来实现消息的被动回复。

class wechat\_login(threading.Thread):

def \_\_init\_\_(self, name, service\_id):

threading.Thread.\_\_init\_\_(self)

self.name = name + service\_id

self.service\_id = service\_id

self.pic\_dir = env.QR\_SAVE\_DIR\_PRE\_FIX + service\_id + '.png'

self.wechat\_instance = itchat.new\_instance()

self.wechat\_info = None

self.msg\_q = Queue()

self.reply\_q = Queue()

self.single = threading.Event()

def run(self):

print("开始线程：" + self.name)

self.wechat\_instance.login(picDir=self.pic\_dir, loginCallback=self.login\_callback,

exitCallback=self.logout\_callback)

@self.wechat\_instance.msg\_register(itchat.content.TEXT)

def tuling\_reply(msg):

# since my server memory is not support this so using tuling instead.

# self.msg\_q.put(msg['Text'])

# self.single.wait()

# chatbot\_reply = self.reply\_q.get()

# do not using this reply cause it's really bad.

default\_msg = 'I received: ' + msg['Text']

tuling\_msg = tuling\_response(msg['Text'])

reply = tuling\_msg or default\_msg

record\_chat(msg, reply)

return reply

def record\_chat(msg, reply):

with app.app\_context():

wechat\_record\_instance = WechatRecord(self.wechat\_instance.loginInfo['User'].Uin,

self.service\_id, msg, reply)

db.session.add(wechat\_record\_instance)

db.session.commit()

self.wechat\_instance.run()

print("退出线程：" + self.name)

def login\_callback(self):

with app.app\_context():

uin = self.wechat\_instance.loginInfo['User'].Uin

wechat\_info\_instance = WechatInfo.query.filter\_by(wechat\_id=uin).first()

if wechat\_info\_instance:

wechat\_info\_instance.login\_status = 1

wechat\_info\_instance.service\_id = self.service\_id

wechat\_info\_instance.gmt\_modified = datetime.datetime.now()

else:

wechat\_info\_instance = WechatInfo(self.wechat\_instance.loginInfo, self.service\_id)

db.session.add(wechat\_info\_instance)

service\_log\_instance = ServiceLog(wechat\_info\_instance.wechat\_id, wechat\_info\_instance.service\_id)

db.session.add(service\_log\_instance)

# the same reason not using this at this time.

# wechat\_robot\_instance = WechatRobot.query.filter\_by(wechat\_id=uin).first()

# if wechat\_robot\_instance:

# chatbot\_instance = ChatService(wechat\_robot\_instance.robot\_model\_addr, self.msg\_q,

# self.reply\_q, self.single)

# chatbot\_instance.start()

db.session.commit()

self.wechat\_info = wechat\_info\_instance

def logout\_callback(self):

with app.app\_context():

uin = self.wechat\_instance.loginInfo['User'].Uin

wechat\_instance = WechatInfo.query.filter\_by(wechat\_id=uin).first()

wechat\_instance.login\_status = 0

service\_log\_instance = ServiceLog.query.filter\_by(service\_id=self.service\_id).first()

service\_log\_instance.end\_time = datetime.datetime.now()

db.session.commit()

def tuling\_response(msg):

data = {

'key': KEY,

'info': msg,

'userid': 'wechat-robot',

}

try:

r = requests.post(API\_URL, data=data).json()

return r.get('text')

except:

return

微信机器人是本项目的核心，主要处理并生成微信消息的被动回复内容，如下的核心代码启动了一个机器人线程对于每一个用户的登陆请求。

class Chatbot(threading.Thread):

def \_\_init\_\_(self, wechat\_id, sample\_file\_path):

threading.Thread.\_\_init\_\_(self)

self.sample\_file\_path = sample\_file\_path

self.wecaht\_id = wechat\_id

def run(self):

print("Start training")

save\_path = MODEL\_SAVE\_DIR\_PRE\_FIX + str(uuid.uuid1())

extract\_data(self.sample\_file\_path, save\_path)

start\_train(save\_path)

with app.app\_context():

wechat\_robot\_instance = WechatRobot(self.wecaht\_id, save\_path)

db.session.add(wechat\_robot\_instance)

db.session.commit()

print("End of training")

class ChatService(threading.Thread):

def \_\_init\_\_(self, model\_addr, msg\_q, reply\_q, single\_msg):

threading.Thread.\_\_init\_\_(self)

self.model\_addr = model\_addr

self.msg\_q = msg\_q

self.reply\_q = reply\_q

self.single\_msg = single\_msg

def run(self):

print("Start chat service...")

start\_service(self.model\_addr, self.msg\_q, self.reply\_q, self.single\_msg)

print("End chat service...")

后端API是后端暴露其服务的主要方式，包括登陆，聊天记录等的API都是通过Flask的RESTFul API方法暴露给前端，如下是相关代码。

api\_v1 = Blueprint('api\_v1', \_\_name\_\_, url\_prefix='/api/v1/')

CORS(api\_v1)

@api\_v1.route('/login', methods=['POST'])

def login():

service\_id = str(uuid.uuid1())

login\_thread = wechat.wechat\_login('login', service\_id)

login\_thread.start()

pic\_link = env.server\_resources\_prefix() + '/qr/' + service\_id + '.png'

time.sleep(2)

return jsonify(link=pic\_link, serviceId=service\_id), 201

@api\_v1.route('/login/status/<service\_id>', methods=['GET'])

def login\_status(service\_id):

wechat\_info = WechatInfo.query.filter\_by(service\_id=service\_id).first()

if wechat\_info:

return jsonify(loginStatus=wechat\_info.login\_status), 200

else:

return jsonify(msg='not login'), 404

@api\_v1.route('/chat/records/<wechat\_id>', methods=['GET'])

def chat\_records(wechat\_id):

wechat\_records = WechatRecord.query.filter\_by(wechat\_id=wechat\_id).all()

records = [record.to\_dict() for record in wechat\_records]

return jsonify(records=records), 200

@api\_v1.route('/wechat/info/<service\_id>', methods=['GET'])

def wechat\_info(service\_id):

wechat\_info\_instance = WechatInfo.query.filter\_by(service\_id=service\_id).first()

if wechat\_info\_instance:

return jsonify(wechatInfo=wechat\_info\_instance.to\_dict()), 200

else:

return jsonify(msg='not login'), 404

@api\_v1.route('/wechat/sample/add', methods=['POST'])

def add\_wechat\_sample():

wechat\_id = request.form['wechatId']

if 'contentFile' not in request.files:

return jsonify(msg='file not present'), 406

content\_file = request.files['contentFile']

file\_name = str(uuid.uuid1()) + '.conv'

sample\_file\_path = env.SAMPLE\_SAVE\_DIR\_PRE\_FIX + file\_name

content\_file.save(sample\_file\_path)

wechat\_sample\_instance = WechatSample(file\_name, wechat\_id)

db.session.add(wechat\_sample\_instance)

db.session.commit()

chatbot\_instance = Chatbot(wechat\_id, sample\_file\_path)

chatbot\_instance.start()

return jsonify(sample=wechat\_sample\_instance.to\_dict()), 200

@api\_v1.route('/wechat/samples/<wechat\_id>', methods=['GET'])

def get\_wechat\_samples(wechat\_id):

wechat\_samples = WechatSample.query.filter\_by(wechat\_id=wechat\_id, status=1).all()

samples = [sample.to\_dict() for sample in wechat\_samples]

return jsonify(samples=samples), 200

@api\_v1.route('/wechat/sample/delete/<sample\_id>', methods=['DELETE'])

def delete\_wechat\_sample(sample\_id):

wechat\_sample\_instance = WechatSample.query.filter\_by(id=sample\_id).first()

wechat\_sample\_instance.status = 0

db.session.commit()

return jsonify(sample=wechat\_sample\_instance.to\_dict()), 200

@api\_v1.route('/wechat/robots/<wechat\_id>', methods=['GET'])

def list\_wechat\_robots(wechat\_id):

wechat\_robots = WechatRobot.query.filter\_by(wechat\_id=wechat\_id, status=1).all()

robots = [robot.to\_dict() for robot in wechat\_robots]

return jsonify(robots=robots), 200

@api\_v1.route('/service/logs/<wechat\_id>', methods=['GET'])

def list\_service\_logs(wechat\_id):

service\_logs = ServiceLog.query.filter\_by(wechat\_id=wechat\_id, status=1).all()

logs = [log.to\_dict() for log in service\_logs]

return jsonify(serviceLogs=logs), 200

由于前端交付复杂，构建用户交互的代码过多，下面是前端交互的核心容器代码，其他相关代码请参照源码。

type Props = {

menuStatus: boolean,

bottomNavCheckedIndex: number,

toggleMenuStatus: () => void,

handleBottomNavClick: (number) => void,

push: (string) => void

};

const drawerWidth = 240;

const styles = theme => ({

appFrame: {

zIndex: 1,

overflow: 'hidden',

display: 'flex'

},

appBar: {

position: 'fixed',

top: 0,

transition: theme.transitions.create(['margin', 'width'], {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.leavingScreen,

}),

},

appBarShift: {

width: `calc(100% - ${drawerWidth}px)`,

transition: theme.transitions.create(['margin', 'width'], {

easing: theme.transitions.easing.easeOut,

duration: theme.transitions.duration.enteringScreen,

}),

marginLeft: drawerWidth,

},

menuButton: {

marginLeft: 12,

marginRight: 20,

},

drawerPaper: {

width: drawerWidth,

},

contentContainer: {

flexGrow: 1,

marginTop: 60,

backgroundColor: theme.palette.background.default,

transition: theme.transitions.create('margin', {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.leavingScreen,

})

},

contentContainerShift: {

transition: theme.transitions.create('margin', {

easing: theme.transitions.easing.easeOut,

duration: theme.transitions.duration.enteringScreen,

})

},

bottomBarContainer: {

zIndex: 100,

position: 'fixed',

bottom: '0',

width: '100%',

display: 'flex'

},

bottomBar: {

alignSelf: 'flex-end',

flexGrow: 1

}

});

class FrameContainer extends React.Component<Props> {

onBottomNavClick = (event, index) => {

const {handleBottomNavClick, push} = this.props;

handleBottomNavClick(index);

push(mapMenuIndexToURL(index));

};

onSideBarItemClick = (url, index) => {

const {push, toggleMenuStatus, handleBottomNavClick} = this.props;

return () => {

toggleMenuStatus();

push(url);

handleBottomNavClick(index);

};

};

render() {

const {classes, menuStatus, bottomNavCheckedIndex, toggleMenuStatus} = this.props;

return (

<div className={classes.appFrame}>

<AppBar

className={classNames(classes.appBar, {

[classes.appBarShift]: menuStatus,

})}

>

<Toolbar disableGutters>

<IconButton

color='inherit'

aria-label='open drawer'

onClick={toggleMenuStatus}

className={classes.menuButton}

>

{menuStatus ? <ArrowBackIcon/> : <MenuIcon/>}

</IconButton>

<Typography variant='title' color='inherit' noWrap>

{TextConstants.TITLE\_HOME}

</Typography>

</Toolbar>

</AppBar>

<Drawer

anchor='left'

open={menuStatus}

classes={{paper: classes.drawerPaper}}

onClose={toggleMenuStatus}

>

<Divider/>

<List>

<ListItem onClick={this.onSideBarItemClick(URIConstants.PROFILE, 0)} button>

<ListItemIcon>

<AccountCircleIcon/>

</ListItemIcon>

<ListItemText primary={TextConstants.PROFILE}/>

</ListItem>

<ListItem onClick={this.onSideBarItemClick(URIConstants.ROBOT, 2)} button>

<ListItemIcon>

<BugReportIcon/>

</ListItemIcon>

<ListItemText primary={TextConstants.AUTO}/>

</ListItem>

<ListItem onClick={this.onSideBarItemClick(URIConstants.CONTENT, 1)} button>

<ListItemIcon>

<DescriptionIcon/>

</ListItemIcon>

<ListItemText primary={TextConstants.CONTENT}/>

</ListItem>

<ListItem onClick={this.onSideBarItemClick(URIConstants.LOG, 3)} button>

<ListItemIcon>

<ListIcon/>

</ListItemIcon>

<ListItemText primary={TextConstants.LOG}/>

</ListItem>

</List>

<Divider/>

<List>

<ListItem onClick={this.onSideBarItemClick(URIConstants.SETTING, 0)} button>

<ListItemIcon>

<SettingsIcon/>

</ListItemIcon>

<ListItemText primary={TextConstants.SETTINGS}/>

</ListItem>

<ListItem onClick={this.onSideBarItemClick(URIConstants.FEEDBACK, 0)} button>

<ListItemIcon>

<FeedbackIcon/>

</ListItemIcon>

<ListItemText primary={TextConstants.FEEDBACK}/>

</ListItem>

<ListItem onClick={this.onSideBarItemClick(URIConstants.ABOUT, 0)} button>

<ListItemIcon>

<InfoIcon/>

</ListItemIcon>

<ListItemText primary={TextConstants.ABOUT}/>

</ListItem>

</List>

</Drawer>

<div className={classes.bottomBarContainer}>

<BottomNavigation

value={bottomNavCheckedIndex}

className={classes.bottomBar}

onChange={this.onBottomNavClick}

showLabels

>

<BottomNavigationAction label={TextConstants.PROFILE} icon={<AccountCircleIcon/>}/>

<BottomNavigationAction label={TextConstants.CONTENT} icon={<DescriptionIcon/>}/>

<BottomNavigationAction label={TextConstants.AUTO} icon={<BugReportIcon/>}/>

<BottomNavigationAction label={TextConstants.LOG} icon={<ListIcon/>}/>

</BottomNavigation>

</div>

<div

className={classNames(classes.contentContainer, {

[classes.contentContainerShift]: menuStatus,

})}

>

{this.props.children}

</div>

</div>

);

}

}

export default \_.flowRight(

withRoot,

withStyles(styles)

)(FrameContainer);